



ΕΛΛΗΝΙΚΟ ΑΝΟΙΚΤΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ

Συστήματα Κινητού και Διάχυτου Υπολογισμού (ΣΔΥ)

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

**Σχεδιασμός και Υλοποίηση
Πλατφόρμας Έξυπνης Γεωργίας με
Ενσωμάτωση IoT Κόμβων**

Φοιτητής:

Γεώργιος Καλογείτονας

A.M.: 153514

Επιβλέπων Καθηγητής:

Νικόλαος Σκλάβος

ΠΑΤΡΑ
ΜΑΪΟΣ 2026

Η παρούσα εργασία αποτελεί πνευματική ιδιοκτησία του/της φοιτητή/φοιτήτριας («συγγραφέας/δημιουργός») που την εκπόνησε. Στο πλαίσιο της πολιτικής ανοικτής πρόσβασης ο συγγραφέας/δημιουργός εκχωρεί στο ΕΑΠ, μη αποκλειστική άδεια χρήσης του δικαιώματος αναπαραγωγής, προσαρμογής, δημόσιου δανεισμού, παρουσίασης στο κοινό και ψηφιακής διάχυσής τους διεθνώς, σε ηλεκτρονική μορφή και σε οποιοδήποτε μέσο, για διδακτικούς και ερευνητικούς σκοπούς, άνευ ανταλλάγματος και για όλο το χρόνο διάρκειας των δικαιωμάτων πνευματικής ιδιοκτησίας. Η ανοικτή πρόσβαση στο πλήρες κείμενο για μελέτη και ανάγνωση δεν σημαίνει καθ' οιονδήποτε τρόπο παραχώρηση δικαιωμάτων διανοητικής ιδιοκτησίας του συγγραφέα/δημιουργού ούτε επιτρέπει την αναπαραγωγή, αναδημοσίευση, αντιγραφή, αποθήκευση, πώληση, εμπορική χρήση, μετάδοση, διανομή, έκδοση, εκτέλεση, «μεταφόρτωση» (downloading), «ανάρτηση» (uploading), μετάφραση, τροποποίηση με οποιονδήποτε τρόπο, τμηματικά ή περιληπτικά της εργασίας, χωρίς τη ρητή προηγούμενη έγγραφη συναίνεση του συγγραφέα/δημιουργού. Ο συγγραφέας/δημιουργός διατηρεί το σύνολο των ηθικών και περιουσιακών του δικαιωμάτων.

Σχεδιασμός και Υλοποίηση Πλατφόρμας Έξυπνης Γεωργίας με Ενσωμάτωση IoT Κόμβων

Γεώργιος Καλογεΐτονας

Επιτροπή Επίβλεψης Πτυχιακής / Διπλωματικής Εργασίας

Επιβλέπων Καθηγητής:

Νικόλαος Σκλάβος

Καθηγητής ΕΑΠ

Συν-Επιβλέπων Καθηγητής:

ΚΑΜΕΑΣ ΑΧΙΛΛΕΑΣ

Καθηγητής ΕΑΠ

**ΠΑΤΡΑ
ΜΑΪΟΣ 2026**

Περίληψη

Η παρούσα διπλωματική εργασία παρουσιάζει τον σχεδιασμό και την υλοποίηση της πλατφόρμας «Agroponos», ενός καινοτόμου, πολυενοικιακού (multi-tenant) συστήματος Έξυπνης Γεωργίας (Smart Agriculture) βασισμένου στο Διαδίκτυο των Πραγμάτων (IoT). Ο πυρήνας του συστήματος αξιοποιεί το πλαίσιο Laravel 12 για την ανάπτυξη ενός ισχυρού backend και τη βιβλιοθήκη Vue.js για τη δημιουργία μιας διαδραστικής εφαρμογής μοναδικής σελίδας (SPA), εξασφαλίζοντας αυστηρή απομόνωση δεδομένων μεταξύ των χρηστών μέσω Global Scopes. Για τη βέλτιστη διαχείριση του τεράστιου όγκου των τηλεμετρικών δεδομένων, η αρχιτεκτονική διαχωρίζει δομικά τα σχεσιακά μεταδεδομένα (MySQL) από τις μετρήσεις των αισθητήρων, οι οποίες αποθηκεύονται αποδοτικά σε μια βάση χρονοσειρών (InfluxDB). Στο επίπεδο του υλικού (edge computing), αναπτύχθηκε αρθρωτό (modular) υλικολογισμικό αντικειμενοστραφούς σχεδίασης σε C++ για μικροελεγκτές ESP32. Οι τηλεμετρικοί κόμβοι μεταδίδουν τα δεδομένα είτε μέσω δικτύων WiFi—κάνοντας χρήση πρωτοκόλλων MQTT και HTTP—είτε μέσω ασύρματων δικτύων χαμηλής κατανάλωσης LoRa για απομακρυσμένες γεωργικές εγκαταστάσεις. Παράλληλα, το σύστημα υποστηρίζει αμφίδρομη επικοινωνία σε πραγματικό χρόνο (real-time) μέσω WebSockets, επιτρέποντας την άμεση ενημέρωση της διεπαφής χρήστη (Laravel Reverb). Ένας από τους σημαντικότερους πυλώνες του έργου είναι η εφαρμοσμένη ασφάλεια συστημάτων (Cybersecurity). Ειδικότερα, για τους κόμβους LoRa, υλοποιήθηκε μια αρχιτεκτονική end-to-end (E2E) κρυπτογράφησης με τον αλγόριθμο AES-128-CTR και δυναμική διαχείριση μετρητών (anti-replay frame counters). Η προσέγγιση αυτή καθιστά τους ενδιάμεσους κόμβους (LoRa Gateways) απλούς «τυφλούς» αναμεταδότες (blind forwarders), διασφαλίζοντας την απόλυτη ακεραιότητα και εμπιστευτικότητα των μετρήσεων από τη συσκευή πεδίου έως το cloud. Συνολικά, η πλατφόρμα Agroponos αποτελεί μια κλιμακώσιμη, ασφαλή και ακαδημαϊκά άρτια λύση, που γεφυρώνει τις σύγχρονες κατανεμημένες τεχνολογίες λογισμικού με τις πραγματικές ανάγκες του αγροτικού τομέα.

Λέξεις – Κλειδιά

IoT, Έξυπνη Γεωργία, Ασύρματα Δίκτυα Αισθητήρων, InfluxDB, MQTT, ESP32

Design and Implementation of a Smart Agriculture Platform with IoT Node Integration

Georgios Kalogeitonas

Abstract

This thesis presents the design and implementation of the "Agronos" platform, an innovative, multi-tenant Smart Agriculture system based on the Internet of Things (IoT). The core of the system utilizes the Laravel 12 framework to develop a robust backend and the Vue.js library to create an interactive Single Page Application (SPA), ensuring strict data isolation between users through Global Scopes. To optimally manage the vast volume of telemetry data, the architecture structurally separates relational metadata (MySQL) from sensor measurements, which are efficiently stored in a time-series database (InfluxDB). At the hardware level (edge computing), modular object-oriented firmware was developed in C++ for ESP32 microcontrollers. The telemetry nodes transmit data either via WiFi networks—making use of MQTT and HTTP protocols—or via low-power LoRa wireless networks for remote agricultural installations. Simultaneously, the system supports bidirectional real-time communication via WebSockets, allowing instant updates to the user interface (Laravel Reverb). One of the most important pillars of the project is applied cybersecurity. Specifically, for the LoRa nodes, an end-to-end (E2E) encryption architecture was implemented using the AES-128-CTR algorithm and dynamic frame counter management (anti-replay). This approach renders intermediate nodes (LoRa Gateways) as simple "blind forwarders", ensuring absolute integrity and confidentiality of measurements from the field device to the cloud. Overall, the Agronos platform constitutes a scalable, secure, and academically sound solution, bridging modern distributed software technologies with the real-world needs of the agricultural sector.

Keywords

IoT, Smart Agriculture, Wireless Sensor Networks, InfluxDB, MQTT, ESP32

Περιεχόμενα

Περίληψη	iv
Abstract	v
Κατάλογος Εικόνων / Σχημάτων	xiv
1 Εισαγωγή	1
1.1 Αντικείμενο και Κίνητρο	1
1.2 Προκλήσεις	2
1.3 Στόχοι της Εργασίας	4
2 Θεωρητικό Υπόβαθρο και Ανασκόπηση Βιβλιογραφίας	6
2.1 Το IoT στην Έξυπνη Γεωργία	6
2.2 Πρωτόκολλα Επικοινωνίας	10
2.2.1 LoRa	11
2.2.2 Πρωτόκολλο LoRaWAN	12
2.2.3 Δίκτυα LoRa Mesh	14
2.2.4 Wi-Fi	15
2.2.5 Wi-Fi HaLow (IEEE 802.11ah)	16
2.2.6 NB-IoT	17
2.2.7 Sigfox	18
2.2.8 WebSocket	18
2.2.9 HTTP/HTTPS	19
2.2.10 CoAPS (Constrained Application Protocol Secure)CoAP	19
2.2.11 MQTT (Message Queuing Telemetry Transport)	21
2.2.12 Bluetooth Low Energy (BLE)	22
2.2.13 Κυψελοειδής Τηλεφωνία ως Υποδομή Οπισθόζευξης	22
2.2.14 Ενσύρματη Υποδομή: DSL και Οπτικές Ίνες	23
2.3 Διαχείριση Δεδομένων	24
2.3.1 Σχεσιακές βάσεις δεδομένων	24
2.3.2 Βάσεις δεδομένων χρονοσειρών	24
Πτυχιακή / Διπλωματική Εργασία	vi

2.4	Υλικό	27
2.4.1	Μικροελεγκτές και Μονάδες Επεξεργασίας	27
2.4.2	Αισθητήρες	35
2.5	Επισκόπηση Ερευνητικών Συστημάτων και Πειραματικών Διατάξεων . . .	40
2.5.1	Τοπικά Συστήματα Αυτόνομου Ελέγχου Άρδευσης	40
2.5.2	Συστήματα Συνδεδεμένα με Υπολογιστικό Νέφος μέσω Wi-Fi	41
2.5.3	Αυτόνομα Συστήματα Μεγάλης Εμβέλειας με Τεχνολογία LoRa . . .	41
2.5.4	Πολυλειτουργικά Συστήματα με Ενσωμάτωση Μηχανικής Μάθησης	42
2.5.5	Συγκριτική Αξιολόγηση	43
3	Ανάλυση Απαιτήσεων & Προδιαγραφές	44
3.1	Εισαγωγή και Πεδίο Εφαρμογής	44
3.1.1	Σκοπός του Συστήματος	44
3.1.2	Τεχνικό Πεδίο Εφαρμογής	44
3.2	Προσωπικότητες Χρηστών	45
3.2.1	Περσونا 1: Μαρία (Η Αστική Καλλιεργήτρια)	45
3.2.2	Περσونا 2: Νίκος (Ο Επαγγελματίας Παραγωγός)	45
3.2.3	Περσونا 3: Ερευνητής Γεωπόνος	46
3.3	Λειτουργικές Απαιτήσεις	46
3.4	Μη Λειτουργικές Απαιτήσεις	47
3.5	Σενάρια Εφαρμογής	48
3.5.1	Σενάριο Α: Καταχώρηση Αγρού και Εγγραφή Αισθητήρα με Γεω- γραφική Τοποθεσία	48
3.5.2	Σενάριο Β: Αστικός Κήπος και Συνδεσιμότητα Wi-Fi	49
3.5.3	Σενάριο Γ: Απομακρυσμένη Καλλιέργεια και Δίκτυο LoRa	49
3.5.4	Σενάριο Δ: Ανάλυση Ιστορικών Δεδομένων από τον Ερευνητή Γεω- πόνο	50
3.5.5	Σενάριο Ε: Μετεγκατάσταση Αισθητήρα στο Πεδίο	50
4	Αρχιτεκτονική Συστήματος	52
4.1	Επισκόπηση Αρχιτεκτονικής	52
4.2	Μοντέλο Οντοτήτων & Γεωγραφική Σήμανση	55
4.2.1	Διαχωρισμός Συσκευής και Αισθητήρα	56

4.2.2	Σχέση 1 : N και Γεωγραφική Σήμανση	56
4.2.3	Προ-κοινοποιημένα Διαπιστευτήρια	56
4.3	Αρχιτεκτονική Πολυμίσθωσης	58
4.3.1	Στρατηγική Υλοποίησης	59
4.3.2	Μηχανισμός Αυτόματης Ανάθεσης	60
4.3.3	Ιεραρχία Οντοτήτων και Εφαρμογή	60
4.3.4	Ενίσχυση μέσω Πολιτικών Εξουσιοδότησης	61
4.3.5	Απομόνωση στη Βάση Χρονοσειρών	61
4.4	Επικοινωνιακή Αρχιτεκτονική	61
4.4.1	Μονοπάτι 1: WiFi → HTTP → Πλατφόρμα	63
4.4.2	Μονοπάτι 2: WiFi → MQTT Broker → Πλατφόρμα	65
4.4.3	Μονοπάτι 3: LoRa → Gateway → MQTT Broker → Πλατφόρμα	67
4.4.4	Πρωτόκολλο LoRa — Κρυπτογράφηση και Ακεραιότητα	68
4.4.5	Ασύγχρονη Επεξεργασία και Ενημέρωση Πραγματικού Χρόνου	69
4.5	Αρχιτεκτονική Αποθήκευσης Δεδομένων	72
4.5.1	Σχεσιακή Βάση Δεδομένων	72
4.5.2	Βάση Χρονοσειρών	73
4.5.3	Αιτιολόγηση Διττής Αρχιτεκτονικής	75
4.6	Σύνοψη	77
5	Σχεδιασμός και Ανάπτυξη του Κεντρικού Συστήματος Διαχείρισης (Backend)	
	και της Διαδικτυακής Εφαρμογής	78
5.1	Οργάνωση Κώδικα και Εργαλεία Ανάπτυξης	78
5.1.1	Αρχιτεκτονική Βασισμένη σε API	78
5.1.2	Η Στρατηγική Επιλογή του Laravel	78
5.1.3	Τεχνικά Χαρακτηριστικά και Υλοποίηση	79
5.1.4	Ανάπτυξη Διεπαφής Χρήστη και Ενοποίηση	79
5.2	Επίπεδο Δεδομένων	80
5.2.1	Eloquent Μοντέλα και Σχέσεις	80
5.2.2	Enums: Τύποι και Καταστάσεις	85
5.2.3	Υβριδική Αποθήκευση: MySQL + InfluxDB	87
5.3	Επίπεδο Εφαρμογής και Παρουσίασης	89
5.3.1	Διαχείριση Αγρών	89

5.3.2	Διαχείριση Συσκευών	93
5.3.3	Διαχείριση Αισθητήρων	97
5.3.4	Βελτιστοποίηση Απόδοσης — Deferred Props (Inertia v2)	101
5.4	REST API Συσκευών	103
5.4.1	Αυθεντικοποίηση Συσκευών (DeviceAuthController)	103
5.4.2	Λήψη Δεδομένων μέσω HTTP (DeviceDataController)	104
5.4.3	Διαπιστευτήρια MQTT (MqttCredentialService)	106
5.4.4	LoRa Webhook (LoRaDataController)	108
5.4.5	Form Requests και Επικύρωση	110
5.5	Κεντρική Υπηρεσία Δεδομένων	111
5.5.1	Η μέθοδος processSensorData()	111
5.5.2	Εργασία Ουράς	113
5.5.3	Κατασκευή Ωφέλιμου Φορτίου	114
5.5.4	Υπηρεσία Σύνδεσης InfluxDB	115
5.6	Επικοινωνία Πραγματικού Χρόνου	116
5.6.1	Γεγονός SensorReadingEvent	116
5.6.2	Εξουσιοδότηση Καναλιών	117
5.6.3	Αρχικοποίηση Laravel Echo και Reverb	117
5.6.4	Εγγραφή Components σε Κανάλια	118
5.7	Ασφάλεια Εφαρμογής	120
5.7.1	Πολιτικές Εξουσιοδότησης	120
5.7.2	Επικύρωση Δεδομένων Εισόδου	122
5.7.3	Σύνοψη Μηχανισμών Ασφαλείας	124
5.8	Σύνοψη	125
6	Υλοποίηση Υλικολογισμικού	126
6.1	Εισαγωγή και Επιλογή Τεχνολογιών	126
6.1.1	Πλατφόρμα Υλικού: ESP32	126
6.1.2	Περιβάλλον Ανάπτυξης: PlatformIO	127
6.1.3	Γλώσσα Προγραμματισμού	129
6.2	Αρχιτεκτονική Λογισμικού	129
6.2.1	Επίπεδο Αφαίρεσης Αισθητήρων	129
6.2.2	Δυναμική Δημιουργία Αντικειμένων	131

6.2.3	Αρθρωτή Σχεδίαση και Διαχωρισμός Ευθυνών	133
6.3	Υλοποίηση Αισθητήρων	133
6.3.1	Κοινοχρησία Στιγμοτύπων Υλικού	134
6.3.2	Αισθητήρας Εδαφικής Υγρασίας	135
6.3.3	Προσομοιωμένος Αισθητήρας	136
6.4	Υλοποίηση WiFi Κόμβων	136
6.4.1	Μηχανισμός Provisioning: Enhanced Captive Portal	136
6.4.2	Ροή Εκκίνησης και Αυθεντικοποίηση	140
6.4.3	Δρομολόγηση Δεδομένων: MQTT-First με HTTP Fallback	141
6.4.4	Κύριος Βρόχος Εκτέλεσης (Main Loop)	144
6.5	Υλοποίηση LoRa Κόμβων	145
6.5.1	Αρχιτεκτονική και Διαφοροποίηση από WiFi	145
6.5.2	Ροή Εκτέλεσης	146
6.5.3	Δυναδική Σειριοποίηση (Binary Payload)	149
6.5.4	Κρυπτογράφηση AES-128-CTR	150
6.5.5	Διαχείριση Μετρητή Πλαισίου (Frame Counter)	152
6.5.6	Οδηγός Ραδιοσυχνότητας (LoRa Radio Driver)	154
6.5.7	Προφίλ Συσκευής LoRa	156
6.6	Υλοποίηση LoRa Gateway (Bridge Layer)	157
6.6.1	Πλατφόρμα Υλικού	157
6.6.2	Αρχιτεκτονική Firmware	158
6.6.3	Κύριος Βρόχος Εκτέλεσης	159
6.6.4	Διαχείριση Ρυθμίσεων (LittleFS)	160
6.6.5	Captive Portal και Web-Based Παραμετροποίηση	161
6.6.6	Λήψη και Προώθηση Πακέτων	162
6.6.7	Οθόνη OLED και Επιτόπια Διάγνωση	164
6.6.8	Προστασία Μπαταρίας	165
6.7	Διαχείριση Ενέργειας και Διεπαφή Χρήστη	166
6.7.1	Deer Sleep και Μοντέλα Ενεργειακού Κύκλου	166
6.7.2	Παρακολούθηση Στάθμης Μπαταρίας	168
6.7.3	Μηχανισμός Επαναφοράς Εργοστασιακών Ρυθμίσεων	170
6.7.4	Φυσική Εγκατάσταση	171

7	Δοκιμές, Μετρήσεις & Αξιολόγηση	173
7.1	Μεθοδολογία Πειραμάτων	173
7.1.1	Πειράματα Κόμβου WiFi/MQTT	173
7.1.2	Συγκριτική Αξιολόγηση Πρωτοκόλλων Επικοινωνίας	179
7.1.3	Σύνοψη Αποτελεσμάτων Πειραμάτων	182
7.2	Πιλοτική Μελέτη Χρηστικότητας	183
7.2.1	Στόχος και Σχεδιασμός Αξιολόγησης	184
7.2.2	Διαδικασία Διεξαγωγής	185
7.2.3	Εργασίες των Δοκιμαστών	185
7.2.4	Μετρικές και Κριτήρια Αξιολόγησης	186
7.2.5	Παρουσίαση και Σύνοψη Αποτελεσμάτων	187
7.2.6	Αποτελέσματα και Ανάλυση Παρατηρήσεων	188
7.2.7	Προτάσεις Βελτίωσης	189
8	Συμπέρασμα και Μελλοντικές Επεκτάσεις	190
8.1	Συμπεράσματα	190
8.2	Επεκτάσεις	191
8.2.1	Υλοποίηση Δικτύου Πλέγματος (LoRa Mesh) για Γεωγραφικά Διε- σπαρμένες Ιδιοκτησίες	191
8.2.2	Πλήρης Ενσωμάτωση Πρωτοκόλλου LoRaWAN και Χρήση Πολυ- καναλικών Πυλών (LoRa Concentrators)	192
8.2.3	Ενσωμάτωση Ενεργοποιητών (Actuators) και Συστημάτων Αυτομα- τισμού	193
8.2.4	Ενσωμάτωση Τεχνολογίας NB-IoT για Απομακρυσμένους Ενεργο- ποιητές (Actuators)	194
8.2.5	Αντικατάσταση Σύγχρονου Webhook με Ασύγχρονη Αρχιτεκτονική Go MQTT Client / Redis Queue / Laravel Queue Workers	196
8.2.6	Δυναμική Βαθμονόμηση Αισθητήρων μέσω Captive Portal	197
8.2.7	Ενοποιημένο Σύστημα Παραμετροποίησης μέσω Captive Portal και για τους Κόμβους LoRa	198
Α΄	Παράρτημα Α: Κώδικας Backend	200
Α΄.1	Eloquent Models και Trait πολυμίσθωσης	200

Α΄.2	Enums τύπων και χρονικών εύρων	208
Α΄.3	Υπηρεσία Χρονοσειρών	209
Α΄.4	Controllers Εφαρμογής	217
Α΄.5	REST API Controllers	232
Α΄.6	Υπηρεσίες API	241
Α΄.7	Υπηρεσία Δεδομένων και Ουρά Εργασιών	249
Β΄	Παράρτημα Β: Κώδικας Firmware	256
Β΄.1	Ρύθμιση Μεταγλώττισης	257
Β΄.2	Διεπαφή Αισθητήρων και Εργοστάσιο	259
Β΄.3	Υλοποιήσεις Αισθητήρων	262
Β΄.4	Κόμβος WiFi	288
Β΄.5	Κόμβος LoRa	328

Κατάλογος Εικόνων / Σχημάτων

1	Αρχιτεκτονική 3 επιπέδων για έξυπνη γεωργία [5].	7
2	Αρχιτεκτονική 4 επιπέδων για έξυπνη γεωργία [4].	8
3	Σύγκριση χρόνου απόκρισης μεταξύ MySQL, InfluxDB και Prometheus [21].	26
4	Σύγκριση αποθηκευτικού χώρου δεδομένων χρονοσειρών και σχεσιακών βάσεων δεδομένων [21].	27
5	Δομικά στοιχεία του SoC ESP8266 [24].	28
6	Λειτουργικά μπλοκ του μικροελεγκτή ESP32 [25].	30
7	Μοντέλο τεσσάρων επιπέδων αρχιτεκτονικής της πλατφόρμας <i>Agronos</i> . .	54
8	ER διάγραμμα με τις βασικές οντότητες και τις σχέσεις τους	55
9	Sequence diagram εγγραφής συσκευής μέσω web UI	57
10	Διάγραμμα στρωμάτων ασφαλείας multi-tenancy	59
11	Δενδρικό διάγραμμα ιεραρχίας tenant → entities	61
12	Συνολικό διάγραμμα και των τριών μονοπατιών δεδομένων	63
13	Sequence diagram αποκλειστικά για το HTTP μονοπάτι	64
14	Sequence diagram για WiFi κόμβο μέσω MQTT	65
15	Sequence diagram πλήρους ροής LoRa	67
16	Block diagram δομής πακέτου	68
17	Flowchart επεξεργασίας στο SensorDataService	71
18	Sequence diagram εγγραφής μέτρησης	74
19	Πώς συνεργάζονται οι δύο βάσεις	76
20	Σελίδα λίστας αγρών (Farms Index).	89
21	Σελίδα λεπτομερειών αγρού με κάρτες στατιστικών, χάρτη και λίστα αι- σθητήρων.	92
22	Φόρμα δημιουργίας αγρού.	93
23	Φόρμα εγγραφής συσκευής μέσω σάρωσης QR code ή χειροκίνητης εισα- γωγής.	93
24	Σελίδα λίστας συσκευών (Devices Index).	94
25	Σελίδα λεπτομερειών συσκευής με ένδειξη μπαταρίας, λίστα αισθητήρων και MQTT διαπιστευτήρια.	96
26	Φόρμα δημιουργίας αισθητήρα με δυνατότητα σάρωσης QR code.	97

27	Σελίδα λεπτομερειών αισθητήρα με γράφημα μετρήσεων, στατιστικά και πρόσφατες τιμές.	101
28	Στιγμιότυπο του Captive Portal σε κινητή συσκευή — φόρμα επιλογής WiFi δικτύου και ρύθμισης παραμέτρων συσκευής.	138
29	Διεπαφή ρύθμισης του LoRa Gateway — ρυθμίσεις WiFi, LoRa και προώθησης δεδομένων.	162
30	Οθόνη OLED του LoRa Gateway κατά τη λειτουργία — ένδειξη WiFi, μπαταρίας, SF/RSSI και μετρητών Rx/Fwd.	165
31	Σελίδα συσκευής στην πλατφόρμα Αγρονος — εμφανίζεται η στάθμη μπαταρίας (79,6%), τα MQTT credentials, οι αισθητήρες με τις τελευταίες τιμές, και ιστορικό μετρήσεων μπαταρίας.	169
32	Κόμβος WiFi σε αδιάβροχο κουτί IP65 — FireBeetle 2 ESP32-C6 με μπαταρία LiPo, breadboard και αισθητήρες.	171
33	Πτώση ποσοστού μπαταρίας κατά τη διάρκεια 5 ημερών (Πείραμα 1).	174
34	Πτώση ποσοστού μπαταρίας κατά τη διάρκεια 5 ημερών χωρίς αισθητήρες (Πείραμα 2).	175
35	Πτώση ποσοστού μπαταρίας κατά τη διάρκεια 5 ημερών με μεμονωμένο αισθητήρα DHT20 (Πείραμα 3).	177
36	Πτώση ποσοστού μπαταρίας κατά τη διάρκεια 5 ημερών με μεμονωμένο αισθητήρα υγρασίας εδάφους (Πείραμα 4).	178
37	Έντυπο QR codes συσκευής και αισθητήρα που δόθηκε στους δοκιμαστές.	186

Συντομογραφίες & Ακρωνύμια

AES	Advanced Encryption Standard
API	Application Programming Interface
HTTP	Hypertext Transfer Protocol
IoT	Internet of Things
JSON	JavaScript Object Notation
JWT	JSON Web Token
LoRa	Long Range
MQTT	Message Queuing Telemetry Transport
NVS	Non-Volatile Storage
SPA	Single Page Application

1 Εισαγωγή

1.1 Αντικείμενο και Κίνητρο

Η γεωργία αποτελεί διαχρονικά τον θεμέλιο λίθο της ανθρώπινης κοινωνίας, διαδραματίζοντας καθοριστικό ρόλο στη διασφάλιση της βιωσιμότητας του περιβάλλοντος, της οικονομικής σταθερότητας και της παγκόσμιας επισιτιστικής ασφάλειας. Στη σύγχρονη εποχή, ο τομέας αυτός βρίσκεται αντιμέτωπος με μια πρωτοφανή πρόκληση, καθώς η ραγδαία αύξηση του πληθυσμού απαιτεί τη μεγιστοποίηση της παραγωγής. Εκτιμάται ότι η τρέχουσα γεωργική παραγωγή πρέπει να αυξηθεί κατά 60% έως 100% προκειμένου να καλυφθούν οι ανάγκες ενός παγκόσμιου πληθυσμού που αναμένεται να αγγίξει τα 10 δισεκατομμύρια έως το έτος 2050 [1].

Αυτή η ανάγκη για εντατικοποίηση της παραγωγής έρχεται σε άμεση σύγκρουση με την εντεινόμενη κλιματική κρίση και τους αυστηρούς περιορισμούς των φυσικών πόρων. Οι παραδοσιακές γεωργικές πρακτικές έχουν αφήσει ένα σημαντικό περιβαλλοντικό αποτύπωμα, ενώ η κλιματική αλλαγή επιβάλλει την άμεση προσαρμογή των καλλιεργειών σε μεταβαλλόμενες συνθήκες και ακραία καιρικά φαινόμενα. Η λειψυδρία και η περιορισμένη διαθεσιμότητα καλλιεργήσιμης γης καθιστούν επιτακτική τη μετάβαση από την εμπειρική διαχείριση σε μια προσέγγιση ακριβείας, η οποία θα επιτρέπει την παραγωγή περισσότερων αγαθών με τη χρήση λιγότερων πόρων [2].

Η λύση σε αυτό το σύνθετο πρόβλημα εντοπίζεται στην υιοθέτηση της "Έξυπνης Γεωργίας" (Smart Agriculture), η οποία μετατρέπει τον αγροτικό τομέα σε ένα οικοσύστημα βασισμένο στα δεδομένα. Σύμφωνα με την ανάλυση στο "Enhancing Crop Yields through IoT-Enabled Precision Agriculture" [3], ο εκσυγχρονισμός των τεχνικών μέσω του Internet of Things (IoT) επιτρέπει την παρακολούθηση και την ανάλυση των αναγκών των καλλιεργειών σε επίπεδο μεμονωμένου αγρού και σε πραγματικό χρόνο. Το IoT λειτουργεί ως καταλύτης μετασχηματισμού, καθώς η ενσωμάτωση αισθητήρων και αυτοματισμών μειώνει την ανάγκη για ανθρώπινη παρέμβαση, βελτιστοποιεί τη χρήση νερού και λιπασμάτων και παρέχει αξιοποιήσιμες πληροφορίες που διασφαλίζουν τη βιωσιμότητα και την αποδοτικότητα της παραγωγής.

Επιπρόσθετα, η υιοθέτηση των τεχνολογιών IoT υπερβαίνει τη στενή έννοια της αύξησης της παραγωγικότητας, προσφέροντας σημαντικά κοινωνικοοικονομικά οφέλη και

αναβαθμίζοντας την ποιότητα ζωής των παραγωγών [1]. Η αυτοματοποίηση των διαδικασιών και η απομακρυσμένη παρακολούθηση μειώνουν την ανάγκη για επίπονη χειρωνακτική εργασία και συνεχή φυσική παρουσία στον αγρό, επιτρέποντας στους αγρότες να διαχειρίζονται τις εκτάσεις τους με μεγαλύτερη ευκολία και αποδοτικότητα. Παράλληλα, αυτή η τεχνολογική μετάβαση ενισχύει την οικονομική σταθερότητα των αγροτικών περιοχών και διασφαλίζει την επισιτιστική ασφάλεια, μετατρέποντας τον παραγωγό από έναν απλό χειριστή μηχανημάτων σε έναν σύγχρονο διαχειριστή πληροφορίας [1] [2].

Ένα εξίσου κρίσιμο πλεονέκτημα αφορά τη διαφάνεια και την ασφάλεια στην εφοδιαστική αλυσίδα, καθώς οι επιδράσεις της Έξυπνης Γεωργίας εκτείνονται πέρα από τα όρια του αγρού. Μέσω της συνεχούς καταγραφής και ανάλυσης δεδομένων, καθίσταται δυνατή η πλήρης ιχνηλασιμότητα των προϊόντων, εξασφαλίζοντας την ποιότητα και την αυθεντικότητά τους από το στάδιο της παραγωγής έως τον τελικό καταναλωτή. Η χρήση εξειδικευμένων αισθητήρων κατά την αποθήκευση και τη μεταφορά διασφαλίζει τη διατήρηση των βέλτιστων συνθηκών για τα ευπαθή προϊόντα, μειώνοντας δραστικά τις απώλειες και εγγυώμενη την ασφάλεια των τροφίμων [1].

Τέλος, το σύστημα IoT λειτουργεί ως εργαλείο προληπτικής διαχείρισης και αντιμετώπισης εξειδικευμένων κινδύνων που παραδοσιακά απειλούσαν την παραγωγή, όπως ο παγετός και οι αιφνίδιες ασθένειες [1]. Η ικανότητα της πλατφόρμας να αναγνωρίζει πρώιμα σημάδια περιβαλλοντικής καταπόνησης ή παρασίτων επιτρέπει την εκτέλεση στοχευμένων παρεμβάσεων ακριβείας [2]. Αυτή η προσέγγιση όχι μόνο προστατεύει το περιβάλλον από την αλόγιστη χρήση χημικών σκευασμάτων, αλλά διασφαλίζει και τη βιωσιμότητα των φυσικών πόρων, όπως το νερό και η ενέργεια, μέσω της ορθολογικής τους χρήσης [1].

1.2 Προκλήσεις

Παρά τις προόδους στην αγροτική τεχνολογία, η πλειονότητα των παραγωγών εξακολουθεί να βασίζεται στην εμπειρική διαχείριση των καλλιεργειών, μια πρακτική που οδηγεί σε αναποτελεσματική χρήση των πόρων και μειωμένη παραγωγικότητα. Το κύριο πρόβλημα εντοπίζεται στην έλλειψη δεδομένων ακριβείας σε πραγματικό χρόνο. Η απουσία συνεχούς παρακολούθησης των εδαφικών και περιβαλλοντικών συνθηκών οδηγεί συχνά σε λανθασμένες εκτιμήσεις σχετικά με τις ανάγκες άρδευσης και λίπανσης, με αποτέλεσμα την άσκοπη σπατάλη νερού και την υποβάθμιση του εδάφους από την υπερβολική χρήση

χημικών [3]. Αυτή η «ψηφιακή τυφλότητα» καθιστά τις καλλιέργειες ευάλωτες στις απότομες κλιματικές μεταβολές, καθώς οι παραγωγοί αδυνατούν να αντιδράσουν προληπτικά σε κινδύνους όπως η ξηρασία ή ο παγετός.

Επιπλέον, η υιοθέτηση υπάρχουσών λύσεων IoT παρεμποδίζεται από το υψηλό τεχνικό και οικονομικό φράγμα εισόδου. Πολλά συστήματα που κυκλοφορούν στην αγορά είναι είτε υπερβολικά πολύπλοκα για τον μέσο χρήστη, είτε στερούνται επεκτασιμότητας και ευελιξίας στην επικοινωνία. Η ετερογένεια των αγροτικών περιοχών απαιτεί λύσεις που να μπορούν να λειτουργήσουν τόσο σε αστικά περιβάλλοντα με κάλυψη Wi-Fi όσο και σε απομακρυσμένες εκτάσεις όπου η συνδεσιμότητα είναι περιορισμένη [1]. Η έλλειψη μιας ενοποιημένης πλατφόρμας που να διαχειρίζεται απρόσκοπτα διαφορετικά πρωτόκολλα επικοινωνίας (όπως HTTP και LoRa) και να παρέχει μια φιλική προς τον χρήστη εμπειρία εγκατάστασης, διασφαλίζοντας παράλληλα την ακεραιότητα και την ασφάλεια των δεδομένων, αποτελεί ένα κρίσιμο τεχνολογικό κενό.

Μια επιπλέον κρίσιμη πρόκληση αφορά τους περιορισμούς στη συνδεσιμότητα λόγω των μεγάλων αποστάσεων που χαρακτηρίζουν τις αγροτικές εκμεταλλεύσεις. Οι παραδοσιακές ασύρματες τεχνολογίες, όπως το Wi-Fi, ενώ προσφέρουν υψηλούς ρυθμούς μετάδοσης δεδομένων, περιορίζονται από μικρή εμβέλεια που καθιστά αδύνατη την κάλυψη εκτεταμένων αγρών. Η ανάγκη για τεχνολογίες LPWAN (Low Power Wide Area Network), όπως το LoRa (Long Range), είναι επιτακτική για τη διασφάλιση της επικοινωνίας σε ευρείες περιοχές όπου η υποδομή τηλεπικοινωνιών είναι ελλιπής. Η δυσκολία αυτή επιτείνεται από την ανάγκη για αξιόπιστη μεταφορά δεδομένων από απομακρυσμένους αισθητήρες προς κεντρικά gateways, χωρίς την απώλεια κρίσιμων πληροφοριών λόγω περιβαλλοντικών εμποδίων [1].

Παράλληλα, η ενεργειακή αυτονομία των συσκευών IoT αποτελεί έναν από τους σημαντικότερους ανασταλτικούς παράγοντες για τη μακροπρόθεσμη βιωσιμότητα τέτοιων συστημάτων. Σε περιοχές που αντιμετωπίζουν ελλείψεις στην παροχή ηλεκτρικού ρεύματος, η ανάγκη για ενεργειακά αυτόνομους κόμβους είναι ζωτικής σημασίας. Η πρόκληση έγκειται στον σχεδιασμό συσκευών που χρησιμοποιούν μικροελεγκτές χαμηλής κατανάλωσης και αξιοποιούν ανανεώσιμες πηγές, όπως η ηλιακή ενέργεια, για τη συνεχή λειτουργία τους. Η έλλειψη αποτελεσματικής διαχείρισης ισχύος και η εξάρτηση από μπαταρίες περιορισμένης διάρκειας αυξάνουν το κόστος συντήρησης και μειώνουν την αξιοπιστία της πλατφόρμας, καθιστώντας απαραίτητη την ενσωμάτωση τεχνολογιών που ελαχιστοποιούν

το ενεργειακό αποτύπωμα [2].

Τέλος, η διαχείριση των παραγόμενων δεδομένων αποτελεί από μόνη της μια πρόκληση. Η απλή συλλογή μετρήσεων δεν αρκεί· το πρόβλημα έγκειται στην αδυναμία των παραδοσιακών συστημάτων να επεξεργαστούν μεγάλους όγκους χρονοσειρών (time-series data) και να τους μετατρέψουν σε κατανοητή πληροφορία. Η ανάγκη για ένα σύστημα που όχι μόνο καταγράφει, αλλά οπτικοποιεί και αυτοματοποιεί τη λήψη αποφάσεων είναι επιτακτική για την επίτευξη της βιωσιμότητας που απαιτεί η σύγχρονη Agriculture 4.0, [2]. Το Agronos σχεδιάστηκε ακριβώς για να γεφυρώσει αυτό το χάσμα, προσφέροντας μια αρχιτεκτονική που συνδυάζει την ευκολία χρήσης με την τεχνική αρτιότητα και την ασφάλεια.

1.3 Στόχοι της Εργασίας

Ο πρωταρχικός στόχος της παρούσας διπλωματικής εργασίας είναι ο σχεδιασμός και η υλοποίηση μιας ολοκληρωμένης, επεκτάσιμης και ασφαλούς πλατφόρμας IoT για την Έξυπνη Γεωργία, η οποία θα γεφυρώνει το χάσμα μεταξύ της συλλογής δεδομένων πεδίου και της λήψης στρατηγικών αποφάσεων. Κεντρική επιδίωξη αποτελεί η ανάπτυξη μιας αρχιτεκτονικής που υποστηρίζει πολλαπλά πρωτόκολλα επικοινωνίας, επιτρέποντας την απρόσκοπτη λειτουργία τόσο συσκευών Wi-Fi για αστικά περιβάλλοντα όσο και LoRa κόμβων για απομακρυσμένες καλλιέργειες. Με αυτόν τον τρόπο, το σύστημα Agronos ανταποκρίνεται στις απαιτήσεις που θέτουν οι Singh et al. (2023) [1] για ευέλικτες τεχνολογικές υποδομές που μπορούν να προσαρμοστούν στις διαφορετικές ανάγκες της παγκόσμιας γεωργικής παραγωγής.

Στο επίκεντρο της υλοποίησης βρίσκεται η διασφάλιση της ακεραιότητας και της ιδιωτικότητας των δεδομένων μέσω ενός υβριδικού μοντέλου ασφαλείας, προσαρμοσμένου στις ιδιαιτερότητες κάθε πρωτοκόλλου επικοινωνίας. Για τους κόμβους που συνδέονται μέσω Wi-Fi, στόχος είναι η αξιοποίηση του πρωτοκόλλου TLS (HTTPS) για τη δημιουργία κρυπτογραφημένων συνόδων μεταφοράς απευθείας με το backend. Παράλληλα, για το υποσύστημα LoRa, η εργασία εστιάζει στην εφαρμογή ενός μοντέλου κρυπτογράφησης από άκρο σε άκρο. Κεντρικός στόχος εδώ είναι η υλοποίηση κρυπτογράφησης AES-128-CTR από άκρο σε άκρο, η οποία διασφαλίζει ότι οι μετρήσεις των αισθητήρων παραμένουν αδιάβαστες για οποιονδήποτε ενδιάμεσο σταθμό IoT (Gateway), με την αποκρυπτογράφηση να εκτελείται αποκλειστικά στο backend με τη χρήση μοναδικών κλειδιών ανά συσκευή.

Παράλληλα, η εργασία στοχεύει στη βελτιστοποίηση της διαχείρισης μεγάλου όγκου δεδομένων με τη χρήση της InfluxDB, μιας εξειδικευμένης βάσης δεδομένων χρονοσειρών (time-series data). Η ικανότητα επεξεργασίας και οπτικοποίησης ιστορικών δεδομένων είναι κρίσιμη για την κατανόηση των τάσεων της καλλιέργειας και την πρόβλεψη μελλοντικών αναγκών, στόχος που επιτυγχάνεται μέσω του δυναμικού dashboard της πλατφόρμας [3].

Τέλος, η εργασία επιδιώκει να εκδημοκρατίσει την πρόσβαση στην τεχνολογία αιχμής, εστιάζοντας στην εμπειρία του χρήστη και την ευκολία εγκατάστασης. Στόχος είναι η δημιουργία ενός μηχανισμού provisioning μέσω QR codes και captive portals, ο οποίος εξαλείφει την ανάγκη για εξειδικευμένες τεχνικές γνώσεις από την πλευρά του παραγωγού.

2 Θεωρητικό Υπόβαθρο και Ανασκόπηση Βιβλιογραφίας

2.1 Το IoT στην Έξυπνη Γεωργία

Η ενσωμάτωση του Internet of Things (IoT) στον αγροτικό τομέα αποτελεί τον κύριο καταλύτη για τη μετάβαση από τις παραδοσιακές πρακτικές στην ψηφιακή γεωργία, η οποία χαρακτηρίζεται πλέον ως μια δραστηριότητα προσανατολισμένη στα δεδομένα (data-centric agriculture). Σύμφωνα με το [4], η έξυπνη γεωργία υπερβαίνει την απλή παρακολούθηση περιβαλλοντικών παραμέτρων, καθώς δημιουργεί ένα ολοκληρωμένο οικοσύστημα όπου οι αισθητήρες, οι ενεργοποιητές και οι πλατφόρμες ανάλυσης συνεργάζονται για τη βελτιστοποίηση των εισροών. Αυτός ο μετασχηματισμός επιτρέπει στους παραγωγούς να λαμβάνουν τεκμηριωμένες αποφάσεις σε πραγματικό χρόνο, μειώνοντας το λειτουργικό κόστος και ενισχύοντας τη βιωσιμότητα των καλλιεργειών μέσα από την ακριβή διαχείριση των διαθέσιμων πόρων. Η υιοθέτηση αυτών των τεχνολογιών θεωρείται πλέον η βέλτιστη λύση για τον εκσυγχρονισμό της παραγωγής, προσφέροντας εργαλεία για τον αυτόματο έλεγχο της άρδευσης και τη βελτιστοποίηση της λίπανσης μέσω συστημάτων απομακρυσμένης παρακολούθησης [3].

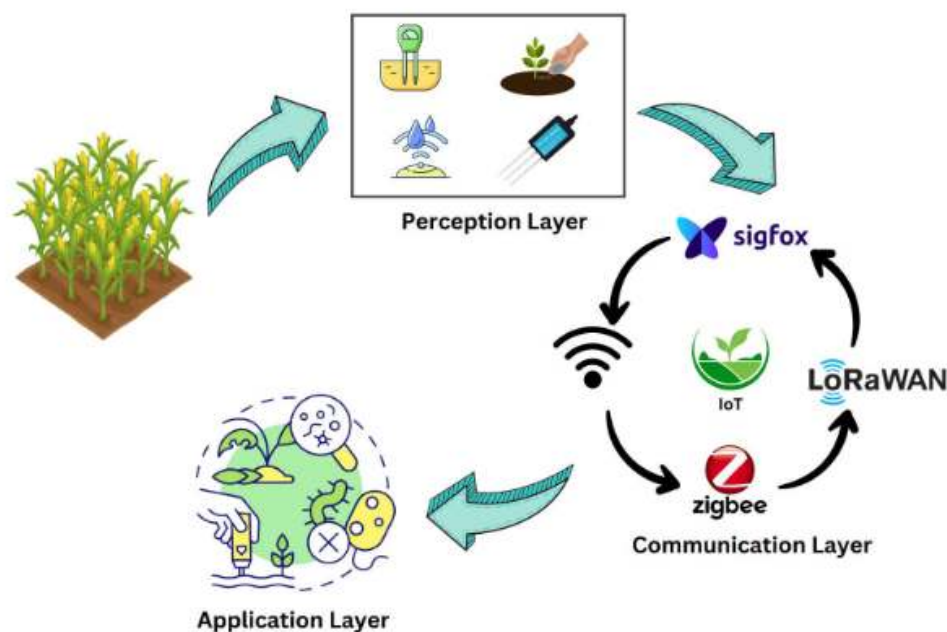
Η αρχιτεκτονική δομή ενός συστήματος IoT για τη γεωργία δεν είναι μονοδιάστατη, αλλά εξελίσσεται ανάλογα με την πολυπλοκότητα της εφαρμογής και τις ανάγκες διαχείρισης των δεδομένων.

Η βάση της θεωρητικής προσέγγισης παραμένει το Κλασικό Μοντέλο Τριών Επιπέδων [5], το οποίο διακρίνει το σύστημα στα εξής επίπεδα:

Επίπεδο Αντίληψης: Αποτελείται από το φυσικό υλικό (hardware) που αλληλεπιδρά με το περιβάλλον. Περιλαμβάνει τους αισθητήρες (π.χ. υγρασίας εδάφους, θερμοκρασίας), τους ενεργοποιητές και τους μικροελεγκτές, και είναι υπεύθυνο για τη συλλογή δεδομένων από το περιβάλλον (π.χ. υγρασία εδάφους, θερμοκρασία, φωτεινότητα).

Επίπεδο Δικτύου: Είναι υπεύθυνο για την ασφαλή και αξιόπιστη μεταφορά της πληροφορίας από το Επίπεδο Αντίληψης στο Επίπεδο Εφαρμογής. Εδώ εντάσσονται τα πρωτόκολλα επικοινωνίας και οι υποδομές δικτύου, με την επιλογή μεταξύ Wi-Fi για εφαρμογές μικρής κλίμακας (π.χ. θερμοκήπια) και LoRaWAN για μεγάλες γεωργικές εκτάσεις να αποτελεί κεντρικό σημείο της σύγχρονης βιβλιογραφίας.

Επίπεδο Εφαρμογής: Αφορά το τελικό στάδιο όπου τα δεδομένα γίνονται αξιοποιήσι-



Σχήμα 1: Αρχιτεκτονική 3 επιπέδων για έξυπνη γεωργία [5].

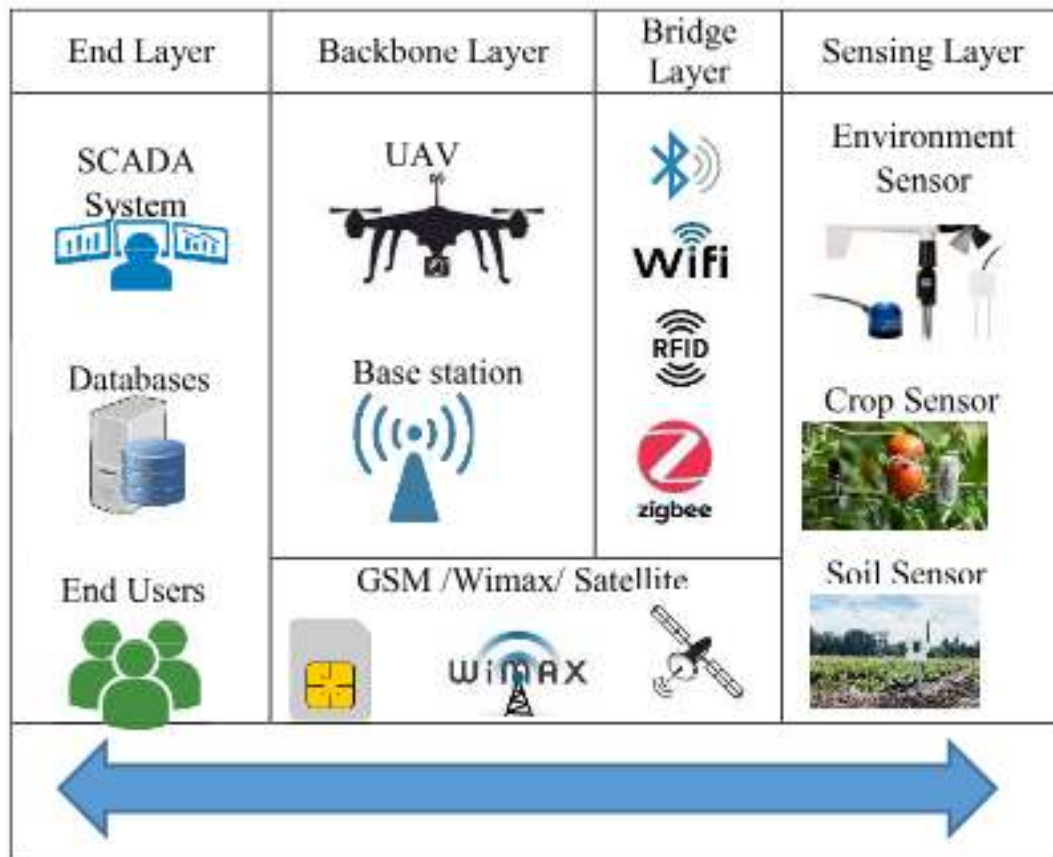
μα και μετατρέπονται σε γνώση. Περιλαμβάνει web εφαρμογές, mobile apps, dashboards, συστήματα ειδοποιήσεων και αυτοματοποίησης. Το επίπεδο αυτό λειτουργεί ως ένας «έξυπνος κόμβος» λήψης αποφάσεων, περιλαμβάνοντας ελεγκτές και τερματικά που ρυθμίζουν τις λειτουργίες του αγρού βάσει των αναγκών της καλλιέργειας [6].

Αν και λειτουργικό, το μοντέλο αυτό θεωρείται πλέον απλοϊκό για σύνθετα συστήματα που απαιτούν ενδιάμεση επεξεργασία.

Για την αρτιότερη περιγραφή της ροής πληροφορίας «από άκρο σε άκρο», οι Ayaz κ.ά. (2019) προτείνουν το Μοντέλο Τεσσάρων Επιπέδων [4]. Εδώ, η αρχιτεκτονική εξειδικεύεται στα επίπεδα Sensing (αισθητήρες), Bridge (τοπική διασύνδεση), Backbone (υποδομή δικτύου ευρείας εμβέλειας) και End Layer (αποθήκευση και ανάλυση). Η προσέγγιση αυτή είναι ιδιαίτερα χρήσιμη για τον διαχωρισμό των τεχνολογιών μικρής εμβέλειας (π.χ. Wi-Fi) από τις τεχνολογίες κορμού (π.χ. Cloud Gateway).

Καθώς η γεωργία 4.0 εστιάζει πλέον στην επιχειρηματική ευφυΐα, το μοντέλο επεκτείνεται στην Αρχιτεκτονική Πέντε Επιπέδων. Εδώ προστίθενται ακόμα δύο κρίσιμα στρώματα: το Middleware Layer, το οποίο αναλαμβάνει τη διαχείριση και το φιλτράρισμα των δεδομένων, και το Business Layer, το οποίο μετατρέπει την πληροφορία σε στρατηγικές αποφάσεις και αυτοματισμούς βάσει επιχειρηματικών κανόνων [1].

Το μοντέλο των έξι επιπέδων, όπως προτάθηκε από τον Ray (2017) [7] και αναλύεται



Σχήμα 2: Αρχιτεκτονική 4 επιπέδων για έξυπνη γεωργία [4].

στην επισκόπηση των Kaur και Shukla (2022) [6], προσφέρει μια λεπτομερή αποδόμηση της ροής πληροφορίας στα αγροτικά συστήματα IoT. Η δομή αυτή ξεκινά από το **Φυσικό Επίπεδο**, όπου αναπτύσσονται αισθητήρες και ενεργοποιητές για την πρωτογενή συλλογή δεδομένων από την καλλιέργεια. Ακολουθεί το **Επίπεδο Δικτύου**, το οποίο εξασφαλίζει τη συνδεσιμότητα (Wi-Fi, LTE, GSM/CDMA) για τη μεταφορά των δεδομένων, ενώ το **Επίπεδο Middleware** αναλαμβάνει τον κεντρικό έλεγχο και τη διαχείριση των συσκευών. Στη συνέχεια, το **Επίπεδο Υπηρεσιών** λειτουργεί ως συνδετικός κρίκος με το υπολογιστικό νέφος παρέχοντας λογισμικό ως υπηρεσία. Η επεξεργασία των μετρήσεων για την εξαγωγή συμπερασμάτων σχετικά με την απόδοση της σοδειάς και το κλίμα πραγματοποιείται στο **Επίπεδο Ανάλυσης**. Η αρχιτεκτονική ολοκληρώνεται με το **Επίπεδο Εμπειρίας Χρήστη**, το οποίο επιτρέπει στον παραγωγό να αλληλεπιδρά με το σύστημα και να λαμβάνει κρίσιμες ενημερώσεις.

Πέρα από την ιεραρχική κατανομή των επιπέδων, η σύγχρονη βιβλιογραφία εισάγει την έννοια του Fluid Computing, η οποία κατανέμει την υπολογιστική ισχύ από το επίπεδο του Mist (επεξεργασία απευθείας στον μικροελεγκτή) και του Fog (τοπικοί εξυπηρετητές/gateways) προς το Cloud [4]. Η προσέγγιση αυτή είναι καθοριστική για τη μείωση της καθυστέρησης (latency) και τη βελτιστοποίηση της κατανάλωσης ενέργειας, ιδιαίτερα σε εφαρμογές που διαχειρίζονται δεδομένα μικρού μεγέθους (small-sized data), όπως οι μετρήσεις αισθητήρων [4].

Η λειτουργική αρτιότητα των αρχιτεκτονικών μοντέλων IoT στη γεωργία βασίζεται σε έναν κυκλικό μηχανισμό τριών φάσεων, ο οποίος περιλαμβάνει τη Διάγνωση (Diagnosis / Detection), τη Λήψη Απόφασης (Decision Making) και την Εκτέλεση (Performing / Execution). Η φάση της διάγνωσης αφορά την αναγνώριση των αναγκών μέσω της συλλογής αισθητηριακών δεδομένων, η λήψη απόφασης περιλαμβάνει την ανάλυση αυτών των δεδομένων μέσω προηγμένων συστημάτων υποστήριξης, ενώ η εκτέλεση αφορά την υλοποίηση των αποφάσεων στο πεδίο μέσω αυτοματοποιημένων μηχανημάτων ή ενεργοποιητών (actuators). Αυτό το λειτουργικό τρίπτυχο σηματοδοτεί τη μετάβαση από την ψηφιακή γεωργία (Industry 3.0), όπου ο άνθρωπος διατηρούσε τον κεντρικό ρόλο στη λήψη αποφάσεων, προς το μοντέλο της Γεωργίας 4.0 (Agriculture 4.0). Στο νέο αυτό περιβάλλον, οι παραγωγικές διαδικασίες καθίστανται πιο αυτόνομες και ευφυείς, αξιοποιώντας την υπολογιστική ισχύ του νέφους και του άκρου (Cloud and Edge computing) για την αυτοματοποίηση κρίσιμων αποφάσεων, όπως η άρδευση και η λίπανση. Με τον τρόπο αυτό,

ο ρόλος του παραγωγού μετατοπίζεται σταδιακά από την εκτέλεση χειρωνακτικών εργασιών στην απομακρυσμένη επίβλεψη και τη στρατηγική διαχείριση της καλλιέργειας, διασφαλίζοντας τη βιωσιμότητα και την αποδοτικότητα της παραγωγής. [8]

Η ζωνική διαχείριση αποτελεί θεμελιώδες εργαλείο για την ενίσχυση της αξιοπιστίας των συστημάτων IoT στη γεωργία ακριβείας, καθώς επιτρέπει την αναγνώριση της χωρικής ετερογένειας μέσω της διαίρεσης του αγρού σε τμήματα με παρόμοια χαρακτηριστικά [4]. Η προσέγγιση αυτή βασίζεται στην ακριβή αποτύπωση κρίσιμων παραμέτρων του εδάφους, όπως η οξύτητα (pH) και τα επίπεδα των μακροθρεπτικών συστατικών αζώτου, φωσφόρου και καλίου (NPK), τα οποία καθορίζουν τις ανάγκες κάθε ζώνης και επιτρέπουν την εφαρμογή της μεταβλητής άρδευσης (Variable Rate Irrigation) βάσει δυναμικών δεικτών καταπόνησης [4]. Σύμφωνα με τους [3], η ενσωμάτωση μοντέλων μηχανικής μάθησης επιτρέπει την επεξεργασία αυτών των δεδομένων για την παροχή εξατομικευμένων προτάσεων σχετικά με την καταλληλότητα των καλλιεργειών, διασφαλίζοντας ότι η αλληλεπίδραση εδάφους και νερού παραμένει σε βέλτιστα επίπεδα. Με αυτόν τον τρόπο, το σύστημα δεν αντιμετωπίζει τον αγρό ως μια ενιαία μονάδα, αλλά ως ένα σύνθετο σύνολο διαφορετικών μικροκλιμάτων και θρεπτικών αναγκών σε "έξυπνες" υδροπονικές εγκαταστάσεις— που απαιτούν εξατομικευμένη φροντίδα για τη μεγιστοποίηση της παραγωγικότητας [3, 4].

2.2 Πρωτόκολλα Επικοινωνίας

Η αποτελεσματικότητα ενός συστήματος έξυπνης γεωργίας εξαρτάται άμεσα από τη διασύνδεση των επιπέδων της αρχιτεκτονικής του μέσω κατάλληλων δικτύων και πρωτοκόλλων επικοινωνίας. Τα πρωτόκολλα αυτά αποτελούν καθορίζουν τον τρόπο με τον οποίο τα δεδομένα μεταφέρονται από τους κόμβους αισθητήρων προς το υπολογιστικό νέφος, εξισορροπώντας κρίσιμες παραμέτρους όπως η εμβέλεια μετάδοσης, η κατανάλωση ισχύος και το διαθέσιμο εύρος ζώνης (bandwidth). Στο σύγχρονο αγροτικό περιβάλλον, η ανάγκη για κάλυψη τόσο περιορισμένων χώρων (π.χ. θερμοκήπια) όσο και εκτεταμένων καλλιεργειών επιβάλλει τη χρήση ενός ετερογενούς δικτύου που συνδυάζει τεχνολογίες μικρής εμβέλειας, όπως το Wi-Fi, με λύσεις χαμηλής ισχύος και μεγάλης εμβέλειας (LPWAN), όπως το LoRaWAN και το αναδυόμενο πρότυπο Wi-Fi HaLow (IEEE 802.11ah). Παράλληλα, σε επίπεδο εφαρμογής, η επιλογή μεταξύ πρωτοκόλλων όπως το HTTP για άμεση επικοινωνία και το MQTT για αποδοτική ανταλλαγή μηνυμάτων μέσω broker, καθορίζει

την κλιμακωσιμότητα και την αξιοπιστία της ροής δεδομένων, διασφαλίζοντας την απρόσκοπτη λειτουργία της πλατφόρμας Agropos σε πραγματικές συνθήκες πεδίου

2.2.1 LoRa

Η τεχνολογία LoRa αποτελεί μια λύση ασύρματης επικοινωνίας που λειτουργεί στο φυσικό επίπεδο (physical layer) του μοντέλου αναφοράς και βασίζεται στην τεχνική διαμόρφωσης διασποράς φάσματος μέσω κελαηδίσματος (Chirp Spread Spectrum - CSS). Το κύριο χαρακτηριστικό της είναι η ικανότητα παροχής επικοινωνίας σε εξαιρετικά μεγάλες αποστάσεις με ταυτόχρονη χαμηλή κατανάλωση ενέργειας, γεγονός που την καθιστά ιδανική για εφαρμογές στο Διαδίκτυο των Πραγμάτων (IoT) [9]. Σύμφωνα με πρόσφατες έρευνες, το LoRaWAN αποτελεί την πλέον διαδεδομένη τεχνολογία LPWAN στην έξυπνη γεωργία, κατέχοντας το 35% των ερευνητικών συνεισφορών στα δίκτυα IoT-WSN κατά την τελευταία πενταετία [5].

Η απόδοση και η προσαρμοστικότητα του δικτύου LoRa καθορίζονται από τρεις βασικές παραμέτρους λειτουργίας:

- **Παράγοντας Διασποράς (Spreading Factor - SF):** Καθορίζει τον αριθμό των chirps ανά σύμβολο. Η επιλογή του SF (από SF7 έως SF12) επιτρέπει τη στρατηγική διαχείριση του συμβιβασμού (trade-off) μεταξύ της εμβέλειας μετάδοσης και του ρυθμού δεδομένων [5]. Υψηλότερες τιμές του SF αυξάνουν την ευαισθησία του δέκτη και την εμβέλεια (φτάνοντας έως και τα 45 km), αλλά μειώνουν τον ωφέλιμο ρυθμό μετάδοσης [5].
- **Εύρος Ζώνης (Bandwidth - BW):** Αντιπροσωπεύει το εύρος των συχνοτήτων στο οποίο διαχέεται το σήμα. Η αύξηση του εύρους ζώνης επιτρέπει ταχύτερη μετάδοση δεδομένων, αλλά συνεπάγεται μειωμένη εμβέλεια επικοινωνίας.
- **Ρυθμός Κωδικοποίησης (Coding Rate - CR):** Χρησιμοποιείται για την εφαρμογή μεθόδων διόρθωσης σφαλμάτων (Forward Error Correction - FEC), βελτιώνοντας την ανθεκτικότητα του σήματος έναντι παρεμβολών και θορύβου [9].

Ένα από τα σημαντικότερα τεχνικά πλεονεκτήματα της τεχνολογίας είναι η δυνατότητα των δεκτών να διαχειρίζονται ταυτόχρονα πολλαπλά σήματα. Αυτό επιτυγχάνεται μέσω [9]:

1. Της **σχεδόν ορθογώνιας φύσης** των διαφορετικών τιμών του Παράγοντα Διασποράς, η οποία επιτρέπει τη λήψη σημάτων που μεταδίδονται ταυτόχρονα στο ίδιο

κανάλι με διαφορετικό SF χωρίς σημαντικές παρεμβολές.

2. Του **φαινομένου σύλληψης (capture effect)**, όπου ένας δέκτης μπορεί να αποδιαμορφώσει επιτυχώς το ισχυρότερο σήμα ακόμη και όταν αυτό επικαλύπτεται χρονικά από ένα ασθενέστερο σήμα.

Στο πλαίσιο της έξυπνης γεωργίας, το LoRa υπερτερεί άλλων τεχνολογιών (όπως το Wi-Fi) λόγω της εκτεταμένης κάλυψης και της ευελιξίας που προσφέρει σε απομακρυσμένες αγροτικές περιοχές [5]. Οι κυριότερες εφαρμογές του περιλαμβάνουν την έξυπνη άρδευση μέσω παρακολούθησης της υγρασίας του εδάφους, την ανίχνευση ασθενειών σε καλλιέργειες όπως τα αμπέλια και οι φράουλες, καθώς και τη δυνατότητα λειτουργίας των κόμβων με συστήματα συλλογής ενέργειας (energy harvesting) από ηλιακά πάνελ για πλήρη ενεργειακή αυτονομία [5].

2.2.2 Πρωτόκολλο LoRaWAN

Το LoRaWAN αποτελεί ένα πρωτόκολλο δικτύωσης ευρείας περιοχής χαμηλής ισχύος (LPWAN), το οποίο είναι χτισμένο πάνω από το φυσικό επίπεδο (physical layer) της τεχνολογίας LoRa και καθορίζει την αρχιτεκτονική του συστήματος καθώς και τις παραμέτρους επικοινωνίας στο επίπεδο MAC. Σε αντίθεση με τα παραδοσιακά δίκτυα πλέγματος, το LoRaWAN υιοθετεί μια αστεροειδή τοπολογία (star topology), η οποία διασφαλίζει τη μέγιστη εξοικονόμηση ενέργειας των τερματικών συσκευών[9].

Η αρχιτεκτονική του δικτύου αποτελείται από τέσσερα βασικά στοιχεία [9]:

1. **Τερματικές Συσκευές (End Nodes):** Κόμβοι αισθητήρων που συλλέγουν δεδομένα και τα μεταδίδουν ασύρματα μέσω LoRa.
2. **Πύλες (Gateways):** Συσκευές που λειτουργούν ως γέφυρες, λαμβάνοντας τα πακέτα LoRa και προωθώντας τα μέσω IP συνδέσεων (Ethernet, Wi-Fi ή κινητό δίκτυο) στον διακομιστή δικτύου.
3. **Διακομιστής Δικτύου (Network Server):** Το κεντρικό σημείο διαχείρισης που αναλαμβάνει την αφαίρεση διπλότυπων πακέτων, τον έλεγχο ταυτότητας, τη δρομολόγηση και τη διαχείριση του προσαρμοστικού ρυθμού δεδομένων (ADR).
4. **Διακομιστής Εφαρμογών (Application Server):** Υπεύθυνος για την επεξεργασία των δεδομένων και την παροχή τους στον τελικό χρήστη.

Για τη διαχείριση της πρόσβασης στο ασύρματο μέσο, το LoRaWAN χρησιμοποιεί ένα πρωτόκολλο πολλαπλής πρόσβασης τύπου ALOHA, όπου οι συσκευές εκπέμπουν δεδομένα χωρίς προηγούμενο συντονισμό. Αν και αυτό αυξάνει την πιθανότητα συγκρούσεων, απλοποιεί σημαντικά τη λειτουργία του δικτύου και μειώνει την κατανάλωση ενέργειας. Μια κρίσιμη λειτουργία για τη βελτιστοποίηση της επικοινωνίας είναι ο *Προσαρμοστικός Ρυθμός Μετάδοσης (Adaptive Data Rate - ADR)*, ο οποίος επιτρέπει στον διακομιστή δικτύου να ρυθμίζει αυτόματα τον Παράγοντα Διασποράς και την ισχύ εκπομπής κάθε συσκευής ανάλογα με τις συνθήκες του καναλιού [9].

Το πρωτόκολλο ορίζει τρεις κατηγορίες λειτουργίας συσκευών (Classes) [9], προσφέροντας διαφορετικές ισορροπίες μεταξύ καθυστέρησης (latency) και κατανάλωσης ισχύος:

- **Κατηγορία A (Class A):** Η πλέον ενεργειακά αποδοτική. Η λήψη δεδομένων από τη συσκευή επιτρέπεται μόνο σε δύο σύντομα παράθυρα αμέσως μετά από μια εκπομπή.
- **Κατηγορία B (Class B):** Ενσωματώνει προγραμματισμένα παράθυρα λήψης σε τακτά χρονικά διαστήματα, συγχρονισμένα μέσω μηνυμάτων (beacons) από την πύλη.
- **Κατηγορία C (Class C):** Η συσκευή βρίσκεται σε συνεχή κατάσταση λήψης, προσφέροντας τη χαμηλότερη καθυστέρηση αλλά με κόστος την υψηλή κατανάλωση ενέργειας.

Τέλος, η λειτουργία του LoRaWAN διέπεται από αυστηρούς περιορισμούς κυκλικού χρόνου λειτουργίας (duty cycle), οι οποίοι κυμαίνονται συνήθως από 1% έως 10%. Αυτοί οι κανονισμοί περιορίζουν τη διάρκεια εκπομπής τόσο για τις τερματικές συσκευές όσο και για τις πύλες, επηρεάζοντας τον συνολικό χρόνο μετάδοσης και τη χωρητικότητα του δικτύου [9].

Το LoRaWAN παρουσιάζει σημαντικά συγκριτικά πλεονεκτήματα έναντι άλλων ασύρματων πρωτοκόλλων, προσφέροντας υψηλότερη αποδοτικότητα, ευρύτερη κάλυψη δικτύου και εξαιρετικά χαμηλές απαιτήσεις ισχύος. Σε σύγκριση με το SigFox, το LoRaWAN χαρακτηρίζεται από μεγαλύτερη ευελιξία, γεγονός που το καθιστά καταλληλότερο για εξειδικευμένες αγροτικές εγκαταστάσεις που απαιτούν αυξημένο έλεγχο ή ιδιαίτερα πρότυπα κάλυψης. Επιπλέον, πειραματικά δεδομένα υποδεικνύουν ότι το πρωτόκολλο αυτό απαιτεί λιγότερη ενέργεια για τη λειτουργία του σε σχέση με το ZigBee και το WiFi,

γεγονός που εξηγεί την κυριαρχία του στην επιστημονική έρευνα, όπου κατέχει το 35% των συνολικών συνεισφορών τα τελευταία πέντε έτη [5].

Οι εφαρμογές του πρωτοκόλλου LoRaWAN καλύπτουν ένα ευρύ φάσμα κρίσιμων γεωργικών αναγκών, από την έξυπνη άρδευση έως την έγκαιρη διάγνωση ασθενειών. Χρησιμοποιείται για τη μετάδοση δεδομένων από αισθητήρες υγρασίας και θερμοκρασίας εδάφους προς ανάλυση, ενώ επιτρέπει την αποτελεσματική παρακολούθηση εκτεταμένων καλλιεργειών ρυζιού με αξιόπιστη μετάδοση σήματος σε αποστάσεις έως και 3,5 χλμ. Στον τομέα της προστασίας των φυτών, το LoRaWAN αξιοποιείται για την έξυπνη ανίχνευση ασθενειών σε καλλιέργειες φράουλας και αμπελώνων —συντά μέσω του εξειδικευμένου προτύπου LoRaWAN-EU868— ενώ η δυνατότητα σύνδεσης των κόμβων του με ηλιακά πάνελ επιτρέπει τη δημιουργία πλήρως αυτόνομων συστημάτων παρακολούθησης σε απομακρυσμένες περιοχές [5].

2.2.3 Δίκτυα LoRa Mesh

Τα δίκτυα LoRa Mesh επεκτείνουν τις δυνατότητες της τεχνολογίας LoRaWAN επιτρέποντας στους κόμβους να επικοινωνούν μεταξύ τους μέσω πολλαπλών αναμεταδόσεων (multi-hop communication), αντί της απευθείας αποστολής δεδομένων σε μια πύλη. Ενώ στα παραδοσιακά δίκτυα LoRaWAN οι συσκευές ακολουθούν μια αστεροειδή τοπολογία επικοινωνώντας αποκλειστικά με την πύλη, στην αρχιτεκτονική mesh οι κόμβοι μπορούν να λειτουργήσουν ως αναμεταδότες ή δρομολογητές. Αυτή η δομή εξασφαλίζει σημαντικά μεγαλύτερη κάλυψη, επιτρέποντας στα δεδομένα να διασχίζουν ενδιάμεσες συσκευές για να φτάσουν σε απομακρυσμένες περιοχές που δεν θα ήταν εύκολα προσβάσιμες σε ένα απλό δίκτυο [10].

Η υιοθέτηση της αρχιτεκτονικής mesh προσφέρει επιπλέον πλεονεκτήματα:

- **Αυξημένη Αξιοπιστία:** Η ύπαρξη εναλλακτικών διαδρομών για τη μεταφορά των δεδομένων μειώνει την πιθανότητα απώλειας πληροφορίας σε περίπτωση βλάβης ενός ενδιάμεσου κόμβου.
- **Ενεργειακή Βελτιστοποίηση Τερματικών:** Οι συσκευές δεν απαιτείται να επικοινωνούν απευθείας με μια απομακρυσμένη πύλη, αλλά μπορούν να αξιοποιούν κοντινότερους κόμβους, εκπέμποντας με λιγότερη ισχύ και μειώνοντας την κατανάλωση ενέργειας.
- **Διαχείριση Φορτίου:** Τα δίκτυα αυτά παρουσιάζουν μεγαλύτερη ανθεκτικότητα σε

μεγάλο αριθμό συσκευών, περιορίζοντας τα προβλήματα συμφόρησης του δικτύου.

[10].

Παρά τα οφέλη αυτά, η βασική πρόκληση των δικτύων mesh παραμένει η αυξημένη κατανάλωση ενέργειας των κόμβων-αναμεταδοτών, οι οποίοι πρέπει να παραμένουν ενεργοί για μεγαλύτερα διαστήματα. Μια καινοτόμος προσέγγιση για την επίλυση αυτού του ζητήματος είναι η αρχιτεκτονική **CottonCandy**, η οποία εισάγει μια αυτο-οργανωμένη δομή χαμηλής ισχύος. Το CottonCandy χρησιμοποιεί συγχρονισμένους κύκλους λειτουργίας (duty cycles), όπου οι κόμβοι εναλλάσσονται μεταξύ ενεργού και ανενεργού κατάστασης για να αποφεύγεται η συνεχής ακρόαση του καναλιού.

Επιπλέον, η συλλογή των δεδομένων γίνεται βάσει αιτημάτων, ελαχιστοποιώντας τις άσκοπες μεταδόσεις. Η δυναμική προσαρμογή της ισχύος εκπομπής και η χρήση τυχαίων καθυστερήσεων για την αποφυγή συγκρούσεων, επιτρέπουν στο σύστημα να διατηρεί υψηλά ποσοστά επιτυχημένων μεταδόσεων, υπερβαίνοντας το 90%. Με αυτόν τον τρόπο, το CottonCandy αποτελεί μια αποτελεσματική και κλιμακούμενη λύση για την υλοποίηση αποδοτικών δικτύων LoRa Mesh σε εφαρμογές μεγάλης κλίμακας [11].

2.2.4 Wi-Fi

Η τεχνολογία Wi-Fi (Wireless Fidelity), βασισμένη στο πρότυπο IEEE 802.11g, αποτελεί έναν από τους βασικούς πυλώνες της έξυπνης γεωργίας, κατέχοντας το 36% των ερευνητικών συνεισφορών στον τομέα. Λειτουργεί σε ένα ευρύ φάσμα συχνοτήτων από 2,4 έως 60 GHz και προσφέρει εξαιρετικά υψηλούς ρυθμούς μετάδοσης δεδομένων, που μπορούν να φτάσουν έως και τα 700 Mbps. Λόγω της υψηλής της απόδοσης (throughput) και της ευκολίας ενσωμάτωσης με διαδικτυακές υπηρεσίες, θεωρείται ιδανική για τη σύνδεση των κόμβων συλλογής (sink nodes) με τις πύλες (gateways) ή την απευθείας μεταφόρτωση δεδομένων στο υπολογιστικό νέφος (cloud) [5].

Παρόλο που η τυπική εμβέλεια του Wi-Fi περιορίζεται συνήθως στα 10-50 μέτρα, η μέλτη επισημαίνει ότι η χρήση κεραιών υψηλής απολαβής μπορεί να επεκτείνει την κάλυψη σε αποστάσεις από 3 έως 7 χιλιόμετρα, καθιστώντας το βιώσιμο για ευρύτερες αγροτικές εκτάσεις. Το δίκτυο οργανώνεται συνήθως σε τοπολογία αστέρα, παρέχοντας ευελιξία και σημαντική εξοικονόμηση κόστους σε σύγκριση με άλλες υποδομές. Στο πλαίσιο της έξυπνης γεωργίας, το Wi-Fi αξιοποιείται συχνά για την παρακολούθηση κρίσιμων περιβαλλοντικών παραμέτρων, όπως η ποσότητα του νερού, η υγρασία του εδάφους και η θερμοκρασία, καθώς και σε προηγμένες εφαρμογές γεωργίας ακριβείας εντός θερμοκη-

πίων [5].

Η πρακτική εφαρμογή του Wi-Fi έχει αποδειχθεί ιδιαίτερα αποτελεσματική σε ποικίλα σενάρια, όπως σε μεγάλης κλίμακας αρδευτικά σχέδια —με χαρακτηριστικό παράδειγμα επιτυχημένης υλοποίησης στην Τυνησία— και σε συστήματα ανίχνευσης ασθενειών των καλλιεργειών. Παρά τον έντονο ανταγωνισμό από τεχνολογίες χαμηλής κατανάλωσης όπως το LoRaWAN, το Wi-Fi παραμένει εξαιρετικά δημοφιλές λόγω της ευρείας διαθεσιμότητας συμβατών συσκευών και της μοναδικής του ικανότητας να διαχειρίζεται μεγάλους όγκους δεδομένων, κάτι που είναι απαραίτητο για την απομακρυσμένη διάγνωση και τον έλεγχο σε πραγματικό χρόνο [5].

2.2.5 Wi-Fi HaLow (IEEE 802.11ah)

Το πρότυπο IEEE 802.11ah, γνωστό και ως Wi-Fi HaLow, αποτελεί μια πρόσφατη τεχνολογία ασύρματης επικοινωνίας ειδικά σχεδιασμένη για εφαρμογές στο Διαδίκτυο των Πραγμάτων (IoT). Λειτουργώντας σε συχνότητες κάτω του 1 GHz, το Wi-Fi HaLow προσφέρει την απαραίτητη εμβέλεια και αποδοτική κατανάλωση ενέργειας για την υποστήριξη μεγάλου αριθμού συσκευών, οι οποίες συχνά βρίσκονται σε απομακρυσμένες περιοχές με περιορισμένη ενεργειακή διαθεσιμότητα. Με τη δυνατότητα να καλύπτει μεγάλες αποστάσεις και να παρέχει αξιόπιστη σύνδεση, αποτελεί μια σημαντική λύση στα προβλήματα που αντιμετωπίζουν τα σύγχρονα δίκτυα IoT [12].

Οι τεχνολογίες ασύρματης επικοινωνίας που αξιοποιούνται στο IoT διακρίνονται κυρίως σε δύο κατηγορίες: τα δίκτυα ασύρματης προσωπικής περιοχής (WPAN), όπως το ZigBee και το Bluetooth Χαμηλής Ενέργειας (BLE), και τα δίκτυα μεγάλης εμβέλειας χαμηλής κατανάλωσης (LPWAN), όπως το LoRa, το Sigfox και το 5G NB-IoT. Τα δίκτυα WPAN προσφέρουν μικρές εμβέλειες που φτάνουν μερικές δεκάδες μέτρα και μεσαίες ταχύτητες μετάδοσης δεδομένων. Από την άλλη πλευρά, τα δίκτυα LPWAN εξυπηρετούν εφαρμογές που απαιτούν μετάδοση δεδομένων σε μεγάλες αποστάσεις αρκετών χιλιομέτρων, αλλά με χαμηλότερες ταχύτητες. Κανένα από τα υπάρχοντα δίκτυα δεν παρέχει ταυτόχρονα μεγάλη εμβέλεια και υψηλές ταχύτητες μετάδοσης, κάτι που δημιουργεί σημαντικούς περιορισμούς σε απαιτητικές εφαρμογές [12].

Το Wi-Fi HaLow γεφυρώνει αυτό το κενό προσφέροντας εμβέλεια έως και 1 km και ταχύτητες από 150 Kbps έως 78 Mbps. Χάρη σε αυτά τα χαρακτηριστικά, μπορεί να καλύψει ένα ευρύ φάσμα εφαρμογών από αισθητήρες χαμηλής κατανάλωσης έως συσκευές που απαιτούν υψηλή απόδοση δεδομένων. Για να επιτύχει καλύτερη απόδοση και εξοικονόμη-

ση ενέργειας στα μεγάλης κλίμακας δίκτυα IoT, το πρότυπο εισάγει τις εξής καινοτομίες:

- **Παράθυρο Περιορισμένης Πρόσβασης (Restricted Access Window - RAW):** Επιτρέπει την ομαδοποίηση των συσκευών και περιορίζει την πρόσβασή τους στο κανάλι επικοινωνίας σε συγκεκριμένα χρονικά διαστήματα, μειώνοντας έτσι τις συγκρούσεις.
- **Κατακερματισμός του Χάρτη Ένδειξης Κυκλοφορίας (TIM Segmentation):** Επιτρέπει στις συσκευές που βρίσκονται σε κατάσταση εξοικονόμησης ενέργειας να ενεργοποιούνται μόνο όταν πρέπει να παραλάβουν δεδομένα, προσφέροντας σημαντικά οφέλη στην αυτονομία τους.
- **Στόχος Χρόνου Αφύπνισης (Target Wake Time - TWT):** Επιτρέπει στις συσκευές να συνεννοούνται εκ των προτέρων με το σημείο πρόσβασης σχετικά με το πότε ακριβώς θα ξυπνήσουν για να αποστείλουν ή να λάβουν δεδομένα.

Μια από τις πιο αντιπροσωπευτικές χρήσεις του Wi-Fi HaLow είναι στα «έξυπνα» δίκτυα μέτρησης και παρακολούθησης, όπου πολυάριθμοι αισθητήρες μεταδίδουν τακτικά δεδομένα σε ένα κεντρικό σημείο. Επιπλέον, είναι ιδανικό για την αποτελεσματική συγκέντρωση δεδομένων σε εφαρμογές που απαιτούν τόσο αξιόπιστη επικοινωνία όσο και περιορισμένη κατανάλωση ενέργειας [12].

2.2.6 NB-IoT

Το Narrowband Internet of Things (NB-IoT) είναι μια τεχνολογία χαμηλής κατανάλωσης ενέργειας και μεγάλης εμβέλειας (LPWAN), ειδικά σχεδιασμένη για τη μαζική δικτύωση IoT συσκευών. Εισήχθη από το 3GPP στην έκδοση 13 και λειτουργεί σε αδειοδοτημένο φάσμα διασφαλίζοντας υψηλή αξιοπιστία για βιομηχανικές και αστικές εφαρμογές.

Το NB-IoT βασίζεται σε κυψελωτά δίκτυα επιτρέποντας την αποστολή μικρών πακέτων δεδομένων σε μεγάλες αποστάσεις με πολύ μικρή κατανάλωση ενέργειας. Μπορεί να υλοποιηθεί τόσο σε GSM δίκτυα όσο και σε LTE μειώνοντας το κόστος ανάπτυξης IOT εφαρμογών. Το NB-IoT προσφέρει τρεις τρόπους λειτουργίας (in-band, guard-band και stand-alone) ώστε να ενσωματώνεται εύκολα με τις υπάρχουσες υποδομές.

Τα βασικά του πλεονεκτήματα περιλαμβάνουν την εκτεταμένη κάλυψη ακόμα και σε υπόγεια ή απομακρυσμένες περιοχές και τη χαμηλή κατανάλωση ενέργειας. Παρέχει επίσης υψηλή αξιοπιστία και ασφάλεια χάρη στην κρυπτογράφηση LTE, καθώς και αυ-

ξημένο ενεργειακό περιθώριο που συχνά είναι κάτω από 25 KHz, εξασφαλίζοντας χαμηλότερη κατανάλωση. Η απλοποιημένη αρχιτεκτονική αστέρα επιτρέπει απευθείας σύνδεση των συσκευών με τους σταθμούς βάσης χωρίς την ανάγκη επιπλέον κόμβων αναμετάδοσης.

Λόγω αυτών των χαρακτηριστικών, το NB-IoT χρησιμοποιείται ευρέως σε έξυπνους μετρητές ρεύματος, νερού και αερίου, σε εφαρμογές έξυπνων πόλεων (π.χ. φωτισμός, έξυπνο parking), αλλά και σε βιομηχανική παρακολούθηση, όπως η ανίχνευση διαρροών σε αγωγούς. Επιπλέον, εφαρμόζεται στην παρακολούθηση περιουσιακών στοιχείων όπως ζώα, οχήματα και εξοπλισμός [13].

2.2.7 Sigfox

Είναι μια τεχνολογία LPWAN που χρησιμοποιεί διαμόρφωση εξαιρετικά στενής ζώνης UNB με σταθερό εύρος ζώνης 100 Hz και διαφορική δυαδική διαμόρφωση μετατόπισης φάσης [14] για επικοινωνία ανοδικής ζεύξης. Υποστηρίζει ρυθμό μετάδοσης δεδομένων 100bps, επιτρέποντας χαμηλή κατανάλωση ενέργειας και εκτεταμένη διάρκεια ζωής της μπαταρίας. Το Sigfox μεταδίδει μικρά πακέτα των 12 bytes με συνολικό μέγεθος στον αέρα 26 bytes για την ελαχιστοποίηση της χρήσης ενέργειας. Παρέχει κάλυψη μεγάλης εμβέλειας, φτάνοντας πάνω από 40 χιλιόμετρα σε υπαίθριες περιοχές με ελάχιστη υποδομή, και μπορεί να διεισδύσει σε υπόγεια ή εσωτερικά περιβάλλοντα. Η ασφάλεια του Sigfox περιλαμβάνει συμμετρική κρυπτογράφηση, παράδοση δεδομένων χωρίς IP, προστασία VPN/SSL και μηχανισμούς για την αποτροπή επιθέσεων επανάληψης, αν και δεν διαθέτει ενσωματωμένη κρυπτογράφηση από προεπιλογή.

2.2.8 WebSocket

Το WebSocket αναπτύχθηκε στο πλαίσιο του HTML5 για να επιτρέψει πλήρη αμφίδρομη επικοινωνία πάνω από TCP, ξεπερνώντας τους περιορισμούς του HTTP που αρχικά δεν είχε σχεδιαστεί για real-time ανταλλαγή δεδομένων [15]. Έτσι το WebSocket έχει αναδειχθεί σε ιδανική λύση για εφαρμογές πραγματικού χρόνου όπως συστήματα παρακολούθησης περιβαλλοντικών παραμέτρων και εφαρμογές IoT [16].

Με το WebSocket, η επικοινωνία είναι συνεχής και αδιάκοπη βελτιώνοντας σημαντικά την ταχύτητα και την αποτελεσματικότητα. Ένα από τα βασικά οφέλη του WebSocket είναι η μείωση του φόρτου στον διακομιστή. Σε αντίθεση με την επικοινωνία μέσω HTTP, όπου κάθε αίτηση ανοίγει ξεχωριστή σύνδεση, το WebSocket χρειάζεται μόνο ένα αρχικό handshake για να δημιουργήσει μία μόνιμη σύνδεση. Έτσι, οι επόμενες ανταλλαγές

δεδομένων δεν επιβαρύνουν συνεχώς τον διακομιστή [16].

Εκτιμάται ότι το WebSocket μειώνει την καθυστέρηση κατά περίπου τρεις φορές συγκριτικά με τη χρήση HTTP. Αυτό είναι ζωτικής σημασίας για εφαρμογές πραγματικού χρόνου όπως συστήματα IoT και περιβαλλοντική παρακολούθηση, όπου η ταχύτητα απόκρισης είναι κρίσιμη [17].

2.2.9 HTTP/HTTPS

Το HTTP (Hypertext Transfer Protocol) αποτελεί το θεμελιώδες πρωτόκολλο επικοινωνίας πελάτη/εξυπηρετητή για την ανάπτυξη εφαρμογών ιστού. Η ευρέως διαδεδομένη έκδοση HTTP/1.1 βασίζεται στο μοντέλο αίτησης/απόκρισης και υποστηρίζει τις λειτουργίες CRUD μέσω των τυποποιημένων μεθόδων POST, GET, PUT και DELETE. Όταν το πρωτόκολλο συνδυάζεται με το TLS (Transport Layer Security) για την επίτευξη ασφαλών και κρυπτογραφημένων επικοινωνιών, προκύπτει η ασφαλής έκδοσή του, γνωστή ως HTTPS.

Η επικοινωνία στο HTTP/HTTPS πραγματοποιείται μέσω του πρωτοκόλλου TCP, το οποίο εξασφαλίζει την αξιόπιστη μεταφορά μεγάλων όγκων δεδομένων. Ωστόσο, η χρήση του σε περιβάλλοντα IoT με περιορισμένους πόρους παρουσιάζει προκλήσεις, καθώς η δημιουργία σύνδεσης TCP για τη σποραδική αποστολή μικρών ποσοτήτων δεδομένων συνεπάγεται υψηλή καθυστέρηση και σημαντική επιβάρυνση (overhead). Παρά την ενσωματωμένη ασφάλεια που προσφέρει το HTTPS, το πρωτόκολλο θεωρείται συνολικά ανεπαρκές για πολύ περιορισμένες συσκευές λόγω της πολυπλοκότητάς του, των μεγάλων επικεφαλίδων και της υψηλής κατανάλωσης ενέργειας που απαιτεί. Παρόλα αυτά, παραμένει μια δημοφιλής επιλογή σε πειραματικές εφαρμογές IoT που δεν έχουν αυστηρούς περιορισμούς, λόγω της τεράστιας εξοικείωσης των προγραμματιστών με αυτό.

Το HTTP/2.0 επιχειρεί να βελτιώσει την αποδοτικότητα μέσω της συμπίεσης επικεφαλίδων και της παράλληλης ανταλλαγής μηνυμάτων στην ίδια σύνδεση. Επιπλέον, εισάγει τη λειτουργία server push, η οποία επιτρέπει στον διακομιστή να προωθεί δεδομένα προς τους πελάτες χωρίς προηγούμενο αίτημα. Τα χαρακτηριστικά αυτά κρίνονται χρήσιμα για εφαρμογές IoT [17].

2.2.10 CoAPS (Constrained Application Protocol Secure)CoAP

Το πρωτόκολλο CoAP (Constrained Application Protocol) σχεδιάστηκε για να καλύψει τις ανάγκες επικοινωνίας σε περιβάλλοντα με περιορισμένους πόρους, όπου οι συσκευές διαθέτουν ελάχιστη μνήμη και επεξεργαστική ισχύ [18]. Η ασφαλής έκδοσή του, το CoAPS, επιτυγχάνεται μέσω της ενσωμάτωσης του επιπέδου ασφαλείας DTLS (Datagram Transport

Layer Security) πάνω από το πρωτόκολλο μεταφοράς UDP [18]. Αυτός ο συνδυασμός επιτρέπει την παροχή κρίσιμων λειτουργιών ασφαλείας, όπως η κρυπτογράφηση δεδομένων, η αυθεντικοποίηση των κόμβων και η διασφάλιση της ακεραιότητας των πληροφοριών, παραμένοντας ταυτόχρονα πιο ελαφρύ σε σχέση με τις λύσεις που βασίζονται στο TCP [18].

Η λειτουργία του CoAPS βασίζεται στο μοντέλο Request-Response, χρησιμοποιώντας διευθύνσεις URI και ενέργειες όπως GET, POST, PUT και DELETE [18]. Το πρωτόκολλο υποστηρίζει δύο κύριες μεθόδους απόκρισης: την piggyback, όπου η απάντηση ενσωματώνεται στο μήνυμα επιβεβαίωσης λήψης (ACK), και τη separate, η οποία χρησιμοποιείται όταν ο server χρειάζεται περισσότερο χρόνο για να επεξεργαστεί το αίτημα [18]. Για τη διασφάλιση της αξιοπιστίας, το CoAPS χρησιμοποιεί επιβεβαιώσιμα (confirmable) μηνύματα που απαιτούν ACK, ενώ παρέχει και τη δυνατότητα μη επιβεβαιώσιμων (non-confirmable) μηνυμάτων για λιγότερο κρίσιμες μεταδόσεις [18].

Ένα από τα σημαντικότερα πλεονεκτήματα του CoAPS είναι η υποστήριξη της άμεσης επικοινωνίας μεταξύ συσκευών (Device-to-Device), η οποία διευκολύνει την ταχύτατη ανταλλαγή μικρών πακέτων δεδομένων [18]. Παρά τα πλεονεκτήματα αυτά, η βιβλιογραφία επισημαίνει ότι η χρήση του DTLS μπορεί να αποδειχθεί υπολογιστικά "βαριά" για εξαιρετικά περιορισμένους κόμβους, επηρεάζοντας την απόδοσή τους [18]. Επιπλέον, το CoAPS παρουσιάζει ευπάθεια σε επιθέσεις τύπου Denial of Sleep (DoS), όπου κακόβουλα αιτήματα εμποδίζουν τη συσκευή από το να εισέλθει σε κατάσταση χαμηλής κατανάλωσης, οδηγώντας στην ταχεία εξάντληση της μπαταρίας της [18].

Στο πλαίσιο των εφαρμογών έξυπνης γεωργίας, το πρωτόκολλο CoAP αναδεικνύεται ως μια από τις πλέον υποσχόμενες λύσεις για τη διασύνδεση κόμβων με εξαιρετικά περιορισμένους πόρους [17]. Η υπεροχή του πρωτοκόλλου έγκειται στην ικανότητά του να διαχειρίζεται αποδοτικά αισθητήρες που αποστέλλουν μικρές ποσότητες δεδομένων σε σποραδικά χρονικά διαστήματα (sporadic data), καθώς η χρήση του UDP εξαλείφει την ανάγκη για τη συνεχή εγκαθίδρυση συνδέσεων TCP [17]. Σε αντίθεση με το πρωτόκολλο HTTP, όπου η δημιουργία μιας σύνδεσης TCP για κάθε μετάδοση επιφέρει σημαντική καθυστέρηση (delay) και υψηλό λειτουργικό φόρτο (overhead), το CoAP λειτουργεί ως μια δυαδική εκδοχή του HTTP βελτιστοποιημένη για το IoT, προσφέροντας ασύγχρονες ανταλλαγές μηνυμάτων με εξαιρετικά μικρές επικεφαλίδες πακέτων [17]. Αυτή η αρχιτεκτονική προσέγγιση το καθιστά ιδανικό για τη μείωση της κατανάλωσης ενέργειας και τη

βέλτιστη χρήση του εύρους ζώνης, παράγοντες που είναι κρίσιμοι για τη μακροζωία των συσκευών που λειτουργούν με μπαταρία στο πεδίο [17].

2.2.11 MQTT (Message Queuing Telemetry Transport)

Είναι ένα ελαφρύ, ευέλικτο πρωτόκολλο ανταλλαγής μηνυμάτων που έχει σχεδιαστεί για επικοινωνία IoT και Machine-to-Machine (M2M), χρησιμοποιώντας ένα μοντέλο δημοσίευσης-παραλαβής μέσω TCP. Αποτελείται από πολλούς πελάτες που συνδέονται με έναν κεντρικό διαμεσολαβητή (broker), ο οποίος δρομολογεί μηνύματα από τους εκδότες στους συνδρομητές με βάση τα θέματα. Το MQTT υποστηρίζει ασύγχρονη επικοινωνία και παρέχει τρία επίπεδα Ποιότητας Υπηρεσίας (QoS): Το MQTT παρέχει τρία διακριτά επίπεδα Ποιότητας Υπηρεσίας (QoS), επιτρέποντας στον δημιουργό του συστήματος να εξισορροπήσει την αξιοπιστία της μετάδοσης με την υπολογιστική επιβάρυνση του δικτύου [18]:

- **QoS 0 (Το πολύ μία φορά / At most once):** Αποτελεί το χαμηλότερο επίπεδο αξιοπιστίας, όπου το μήνυμα αποστέλλεται χωρίς απαίτηση επιβεβαίωσης λήψης από τον παραλήπτη. Ο εκδότης δεν αποθηκεύει το μήνυμα για αναμετάδοση, γεγονός που καθιστά τη διαδικασία εξαιρετικά γρήγορη αλλά ευάλωτη σε απώλειες δεδομένων εάν η σύνδεση διακοπεί.
- **QoS 1 (Τουλάχιστον μία φορά / At least once):** Εγγυάται ότι το μήνυμα θα φτάσει στον προορισμό του τουλάχιστον μία φορά. Ο παραλήπτης οφείλει να στείλει ένα μήνυμα επιβεβαίωσης λήψης στον εκδότη. Εάν ο εκδότης δεν λάβει την επιβεβαίωση εντός συγκεκριμένου χρόνου, αναμεταδίδει το αποθηκευμένο μήνυμα, κάτι που μπορεί να οδηγήσει σε λήψη διπλότυπων μηνυμάτων από τον συνδρομητή.
- **QoS 2 (Ακριβώς μία φορά / Exactly once):** Είναι το πιο ασφαλές και αξιόπιστο επίπεδο, διασφαλίζοντας ότι το μήνυμα παραδίδεται ακριβώς μία φορά στον παραλήπτη. Η λειτουργία αυτή επιτυγχάνεται μέσω μιας διαδικασίας τετραπλής χειραψίας (four-way handshake), η οποία εξαλείφει την πιθανότητα διπλότυπων μηνυμάτων, απαιτώντας όμως τους περισσότερους πόρους επικοινωνίας και αυξάνοντας την καθυστέρηση μετάδοσης.

διασφαλίζοντας την αξιοπιστία του μηνύματος. Η απλότητά του διευκολύνει τη χαμηλή κατανάλωση ενέργειας και την ελάχιστη χρήση εύρους ζώνης. Ωστόσο, η εξάρτηση του MQTT από το TCP/IP αυξάνει την επιβάρυνση της επικοινωνίας και η εξάρτησή του από

έναν διαμεσολαβητή εισάγει ένα μοναδικό σημείο αστοχίας, επηρεάζοντας την ευρωστία του συστήματος όταν ο διαμεσολαβητής αποτύχει [18].

Στην έξυπνη γεωργία, το MQTT προκρίνεται ως το πλέον διαδεδομένο πρωτόκολλο για τη διασύνδεση αισθητήρων με υπολογιστικές πλατφόρμες νέφους (Cloud), λόγω της ικανότητάς του να λειτουργεί αξιόπιστα σε δίκτυα με περιορισμένο εύρος ζώνης και υψηλή καθυστέρηση [17]. Σε σύγκριση με παραδοσιακά πρωτόκολλα όπως το HTTP, το MQTT επιτυγχάνει σημαντικά χαμηλότερη κατανάλωση ενέργειας και ελάχιστο λειτουργικό φόρτο στις κεφαλίδες των πακέτων (headers), καθιστώντας το ιδανικό για τη μετάδοση σποραδικών μετρήσεων [17]. Ωστόσο, η ενσωμάτωση προτύπων ασφαλείας όπως το TLS/SSL για την προστασία των δεδομένων μπορεί να επιφέρει σημαντική υπολογιστική επιβάρυνση στις περιορισμένων πόρων συσκευές, γεγονός που συχνά επιβάλλει την αναζήτηση εναλλακτικών, ελαφρύτερων σχημάτων κρυπτογράφησης [17].

2.2.12 Bluetooth Low Energy (BLE)

Το Bluetooth Low Energy (BLE) αποτελεί μια εξειδικευμένη έκδοση του προτύπου Bluetooth, σχεδιασμένη για εφαρμογές που απαιτούν εξαιρετικά χαμηλή κατανάλωση ενέργειας και μεταδίδουν μικρούς όγκους δεδομένων σε μικρές αποστάσεις [17]. Η τεχνολογία αυτή είναι ιδανική για περιβάλλοντα έξυπνης γεωργίας με υψηλή πυκνότητα αισθητήρων, καθώς επιτρέπει στους κόμβους να παραμένουν σε κατάσταση αναμονής (sleep mode) για το μεγαλύτερο μέρος του χρόνου, καταναλώνοντας ελάχιστη ενέργεια [17].

Παρά τον περιορισμό της εμβέλειας, η βιβλιογραφία προκρίνει τη χρήση του BLE σε υβριδικές τοπολογίες όπου λειτουργεί συμπληρωματικά με τεχνολογίες ευρείας εμβέλειας όπως το LoRa. Η ενσωμάτωση του CoAP πάνω από το BLE επιτρέπει τη δημιουργία αποδοτικών δικτύων όπου πολλαπλοί οικονομικοί αισθητήρες BLE (leaf nodes) συνενώνονται σε έναν κεντρικό κόμβο συγκέντρωσης (aggregator) [18].

Επιπλέον, το BLE λειτουργεί ως μια ασφαλής διεπαφή παραμετροποίησης, επιτρέποντας στο έξυπνο κινητό τηλέφωνο να δράσει ως «διαμεσολαβητής σύζευξης» (pairing mediator) για τη μεταφορά κρίσιμων ρυθμίσεων [19]. Η καθολική υποστήριξη του προτύπου διευκολύνει την άμεση αλληλεπίδραση του παραγωγού με τους αισθητήρες, επιτρέποντας την ασφαλή ανταλλαγή κρυπτογραφικών κλειδιών και διαπιστευτηρίων δικτύου [19, 17].

2.2.13 Κυψελοειδής Τηλεφωνία ως Υποδομή Οπισθόζευξης

Η κυψελοειδής τηλεφωνία αποτελεί την πλέον διαδεδομένη λύση για την ασύρματη οπισθόζευξη σε αγροτικές περιοχές, καθώς προσφέρει ευελιξία ανάπτυξης χωρίς την ανάγκη

για δαπανηρές επίγειες υποδομές καλωδίωσης.

Δίκτυα 4G LTE Το δίκτυο 4G LTE παραμένει η βασική επιλογή για τις τρέχουσες υλοποιήσεις IoT, προσφέροντας μια εξαιρετική ισορροπία μεταξύ κάλυψης, κόστους και ταχύτητας. Με ρυθμούς μετάδοσης που φτάνουν τα 100 Mbps (downlink) και τα 50 Mbps (uplink), το 4G υπερκαλύπτει τις ανάγκες απλής τηλεμετρίας, ενώ επιτρέπει και τη χρήση καμερών χαμηλής ανάλυσης για οπτική επιτήρηση. Η τεχνολογία 4G εξελίχθηκε ειδικά για τη μετάδοση δεδομένων, γεγονός που την καθιστά ικανή να διαχειρίζεται δισεκατομμύρια έξυπνες συσκευές παγκοσμίως.

5G Η τεχνολογία 5G, λειτουργεί από τα 0,6 GHz και πάνω, αξιοποιώντας βασικές τεχνολογίες όπως το massive MIMO, η δικτύωση καθορισμένη από λογισμικό (SDN), η εικονικοποίηση δικτυακών λειτουργιών (Network Function Virtualization - NFV) και η χρήση κυμάτων συχνοτήτων χιλιοστού (millimeter Waves - mmWaves) για την επίτευξη υψηλών ρυθμών δεδομένων, χαμηλής καθυστέρησης και βελτιωμένης συνδεσιμότητας. Το massive MIMO επιτρέπει σημαντική χωρητικότητα δεδομένων μέσω πολυάριθμων κεραιών, ενώ το SDN διαχωρίζει τα επίπεδα ελέγχου και δεδομένων για ευέλικτη διαχείριση του δικτύου. Το NFV εικονικοποιεί τις λειτουργίες του δικτύου, μειώνοντας την εξάρτηση από το υλικό. Το 5G υποστηρίζει επίσης πλήρη αμφίδρομη επικοινωνία, επιτρέποντας την ταυτόχρονη μετάδοση και λήψη στην ίδια συχνότητα, και το Mobile Edge Computing (MEC) για τη μείωση της καθυστέρησης με την επεξεργασία δεδομένων πιο κοντά στους χρήστες. Ο τεμαχισμός του δικτύου (Network Slicing) [20] επιτρέπει τη δημιουργία πολλών εικονικών δικτύων σε κοινή υποδομή, βελτιστοποιώντας την κατανομή των πόρων για διάφορες εφαρμογές.

2.2.14 Ενσύρματη Υποδομή: DSL και Οπτικές Ίνες

Η ενσύρματη σύνδεση παραμένει η "ραχοκοκαλιά" κάθε συστήματος που απαιτεί μέγιστη σταθερότητα και μηδενικό jitter, ειδικά σε εγκαταστάσεις που λειτουργούν ως τοπικοί κόμβοι (Fog/Edge nodes). **DSL** Η τεχνολογία DSL χρησιμοποιεί τις υπάρχουσες τηλεφωνικές γραμμές χαλκού.

- **Περιορισμοί Ταχύτητας:** Οι ταχύτητες σπάνια ξεπερνούν τα 40-100 Mbps και μειώνονται δραματικά με την απόσταση από το κέντρο του παρόχου.
- **Ασυμμετρία:** Ο χαμηλός ρυθμός ανόδου (upload) αποτελεί τροχοπέδη για την αποστολή μεγάλου όγκου δεδομένων από πολλούς αισθητήρες ταυτόχρονα.

- **Ευπάθεια:** Ο χαλκός είναι επιρρεπής σε ηλεκτρομαγνητικές παρεμβολές και καιρικά φαινόμενα, οδηγώντας σε συχνές αποσυνδέσεις σε ακραίες αγροτικές συνθήκες.

Οπτικές Ίνες Οι οπτικές ίνες αποτελούν το μέλλον της αγροτικής συνδεσιμότητας, προσφέροντας συμμετρικές ταχύτητες gigabit και την χαμηλότερη δυνατή καθυστέρηση.

2.3 Διαχείριση Δεδομένων

2.3.1 Σχεσιακές βάσεις δεδομένων

Οι σχεσιακές βάσεις δεδομένων, με χαρακτηριστικά παραδείγματα τη MySQL και την PostgreSQL, είναι συστήματα που οργανώνουν τα δεδομένα σε δομημένους πίνακες οι οποίοι αποτελούνται από στήλες και γραμμές. Οι βάσεις αυτές είναι σχεδιασμένες για την αποθήκευση συσχετισμένων σημείων δεδομένων που αφορούν μια συγκεκριμένη χρονική στιγμή. Η δομή τους επιτρέπει τη διατήρηση αυστηρών κανόνων για το πώς τα δεδομένα συνδέονται μεταξύ τους, διασφαλίζοντας ότι η πληροφορία παραμένει οργανωμένη και προσβάσιμη μέσω σταθερών σχέσεων [21].

Το κύριο πλεονέκτημα των σχεσιακών βάσεων είναι η ικανότητά τους να διασφαλίζουν την ακεραιότητα και τη συνέπεια των δεδομένων, γεγονός που τις καθιστά ιδανικές για συναλλακτικές εφαρμογές. Υποστηρίζουν εξαιρετικά σύνθετα ερωτήματα που περιλαμβάνουν πολλαπλούς πίνακες μέσω λειτουργιών συγχώνευσης και ομαδοποίησης. Είναι ιδιαίτερα αποτελεσματικές σε σενάρια όπου οι σχέσεις μεταξύ διαφορετικών δεδομένων είναι ουσιώδεις και απαιτείται υψηλή ακρίβεια στις αναζητήσεις [21].

Η κύρια αδυναμία τους, όπως επισημαίνεται στη μελέτη, εμφανίζεται κατά τη διαχείριση τεράστιων όγκων δεδομένων με χρονοσήμανση, όπου οι χρόνοι απόκρισης των ερωτημάτων στη MySQL μπορούν να γίνουν υπερβολικά μεγάλοι καθώς το dataset μεγαλώνει. Επιπλέον, οι σχεσιακές βάσεις είναι λιγότερο αποδοτικές στην αποθήκευση, καταλαμβάνοντας σημαντικά περισσότερο χώρο στον δίσκο. Ενδεικτικά, στο πιο απαιτητικό σενάριο δοκιμής, η MySQL κατέλαβε έως και 10 φορές περισσότερο χώρο και ήταν 26 φορές πιο αργή σε σχέση με τις εξειδικευμένες βάσεις χρονοσειρών [21]. Συχνά, για να αντιμετωπιστούν αυτά τα προβλήματα, απαιτείται μείωση της συχνότητας δειγματοληψίας, γεγονός που οδηγεί σε απώλεια πολύτιμης πληροφορίας [21].

2.3.2 Βάσεις δεδομένων χρονοσειρών

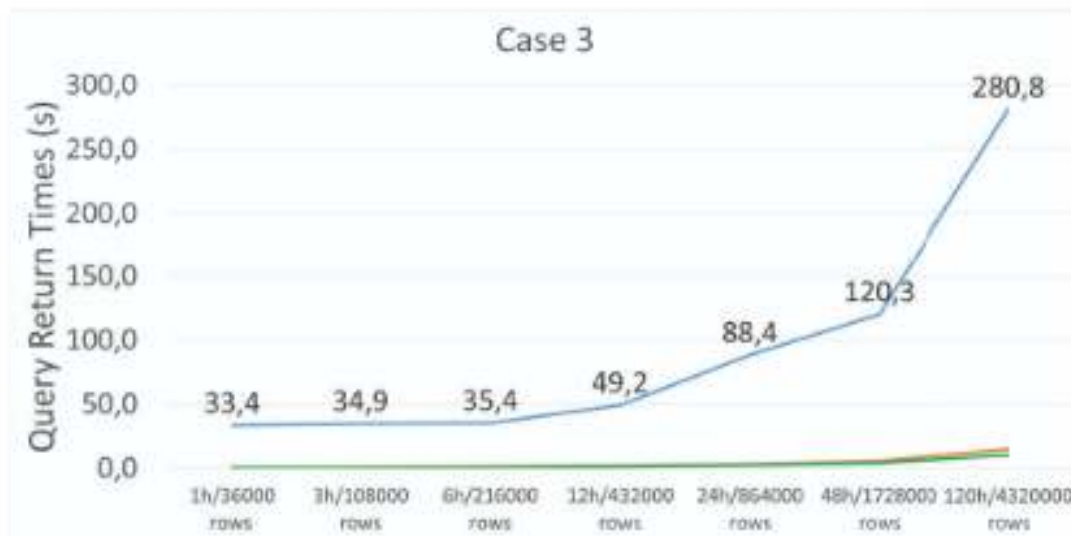
Οι βάσεις δεδομένων χρονοσειρών (Time Series Databases - TSDB), όπως η InfluxDB και η Prometheus, αποτελούν μια εξειδικευμένη κατηγορία συστημάτων NoSQL που σχεδιάστη-

καν για τη διαχείριση δεδομένων με εγγενή χρονική σήμανση. Σε αντίθεση με τις σχεσιακές βάσεις, αποθηκεύουν τα δεδομένα σε χρονολογική σειρά, γεγονός που επιτρέπει την ταχύτατη αναζήτηση και ανάκτηση πληροφοριών που βασίζονται στον χρόνο [21]. Η ανάγκη για τέτοια συστήματα προέκυψε από την τέταρτη βιομηχανική επανάσταση και την εκρηκτική άνοδο του Internet of Things (IoT), όπου δισεκατομμύρια έξυπνες συσκευές παράγουν συνεχείς ροές δεδομένων που χαρακτηρίζονται από τον κανόνα των «4V»: όγκος (volume), ποικιλία (variety), ταχύτητα (velocity) και αξιοπιστία (veracity) [22]. Σε αντίθεση με τις παραδοσιακές σχεσιακές βάσεις (RDBMS), οι οποίες φτάνουν στα όριά τους όταν καλούνται να επεξεργαστούν μαζικά δεδομένα σε πραγματικό χρόνο, οι TSDB αποθηκεύουν την πληροφορία σε χρονολογική σειρά, επιτρέποντας την ταχύτατη ανάκτηση και ανάλυση [21].

Το κύριο πλεονέκτημα των TSDB είναι η ικανότητά τους να εκτελούν λειτουργίες ανάγνωσης και εγγραφής με εξαιρετικά υψηλές ταχύτητες, αποφεύγοντας τις καθυστερήσεις που προκαλεί η ανάγκη ευρετηρίασης (indexing) στις σχεσιακές δομές. Είναι βελτιστοποιημένες για λειτουργίες [21] όπως:

- **Downsampling:** Η μείωση της ανάλυσης παλαιότερων δεδομένων για την εξοικονόμηση αποθηκευτικού χώρου χωρίς την πλήρη διαγραφή της πληροφορίας.
- **Retention Policies:** Η αυτόματη διαγραφή δεδομένων μετά από ένα καθορισμένο χρονικό παράθυρο, διασφαλίζοντας τη βιώσιμη διαχείριση των πόρων.
- **Data Compression:** Προηγμένη συμπίεση δεδομένων που τις καθιστά πολύ πιο αποδοτικές στην αποθήκευση σε σχέση με τις σχεσιακές βάσεις.
- **High Cardinality:** Η αποτελεσματική διαχείριση εκατομμυρίων μοναδικών σειρών δεδομένων, κάτι που καθιστά δυνατή την παρακολούθηση πολύπλοκων διανεμημένων συστημάτων.

Η υψηλή απόδοση των βάσεων δεδομένων χρονοσειρών βασίζεται σε έναν συνδυασμό στηλοειδούς αποθήκευσης και βελτιστοποιημένων μηχανισμών εγγραφής, οι οποίοι διαφοροποιούνται ριζικά από τη γραμμοκεντρική προσέγγιση των σχεσιακών συστημάτων. Συγκεκριμένα, η αποθήκευση των δεδομένων ανά στήλη επιτρέπει την ανάκτηση αποκλειστικά και μόνο των πεδίων που απαιτούνται για ένα ερώτημα, γεγονός που επιταχύνει τη διαδικασία ανάγνωσης κατά περίπου 1,64 φορές, καθώς αποφεύγεται η περιττή επεξεργασία άσχετων πληροφοριών [23].

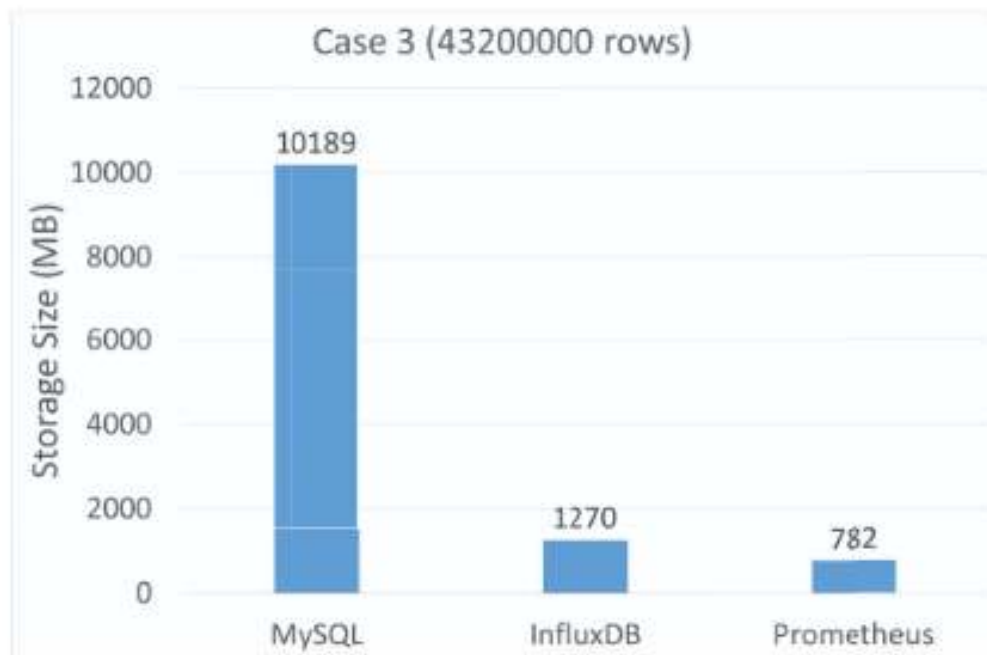


Σχήμα 3: Σύγκριση χρόνου απόκρισης μεταξύ MySQL, InfluxDB και Prometheus [21].

Παράλληλα, οι TSDB επιτυγχάνουν εξαιρετικές ταχύτητες εγγραφής μέσω της ομαδοποίησης των δεδομένων σε δέσμες, μια τεχνική που επιτρέπει την αποστολή πολλαπλών σημείων δεδομένων μέσω μίας μόνο σύνδεσης, μειώνοντας δραστικά το δικτυακό overhead και τον φόρτο του συστήματος. Οι μηχανισμοί αυτοί πλαισιώνονται από προηγμένους αλγορίθμους συμπίεσης που ελαχιστοποιούν τις απαιτήσεις σε εύρος ζώνης και αποθηκευτικό χώρο. Ως αποτέλεσμα αυτής της αρχιτεκτονικής, οι βάσεις χρονοσειρών αποδεικνύονται έως και 10 φορές πιο αποδοτικές στην κατάληψη χώρου στον δίσκο συγκριτικά με τη MySQL, διατηρώντας ταυτόχρονα τη δυνατότητα διαχείρισης μαζικών ροών πληροφορίας σε πραγματικό χρόνο [23].

Η θεωρητική υπεροχή των TSDB επιβεβαιώνεται από πειραματικά δεδομένα, όπου η σύγκριση μεταξύ σχεσιακών βάσεων (MySQL) και εξειδικευμένων λύσεων (Prometheus, InfluxDB) αναδεικνύει την τεράστια διαφορά στην κλιμακωσιμότητα. Σε σενάρια υψηλού φόρτου με εκατομμύρια εγγραφές, η Prometheus εμφάνισε ταχύτητες απόκρισης ερωτημάτων έως και 26 φορές υψηλότερες από τη MySQL [21]. Ενδεικτικά, για την ανάκτηση δεδομένων εύρους 120 ωρών, ο χρόνος απόκρισης της MySQL ανήλθε στα 280,8 δευτερόλεπτα, ενώ η InfluxDB και η Prometheus ολοκλήρωσαν το ίδιο ερώτημα σε μόλις 15,2 και 10,8 δευτερόλεπτα αντίστοιχα [21].

η αποθηκευτική αποδοτικότητα αυξάνεται αναλογικά με τον όγκο των δεδομένων. Ενώ σε μικρά σύνολα δεδομένων η InfluxDB καταλαμβάνει περίπου τον μισό χώρο σε σχέση με τη MySQL, σε περιβάλλοντα μαζικής συλλογής μετρικών η διαφορά αυτή δεκαπλασι-



Σχήμα 4: Σύγκριση αποθηκευτικού χώρου δεδομένων χρονοσειρών και σχεσιακών βάσεων δεδομένων [21].

άζεται, με τη MySQL να απαιτεί 10.189 MB και την Prometheus μόλις 782 MB για το ίδιο σύνολο πληροφοριών [21].

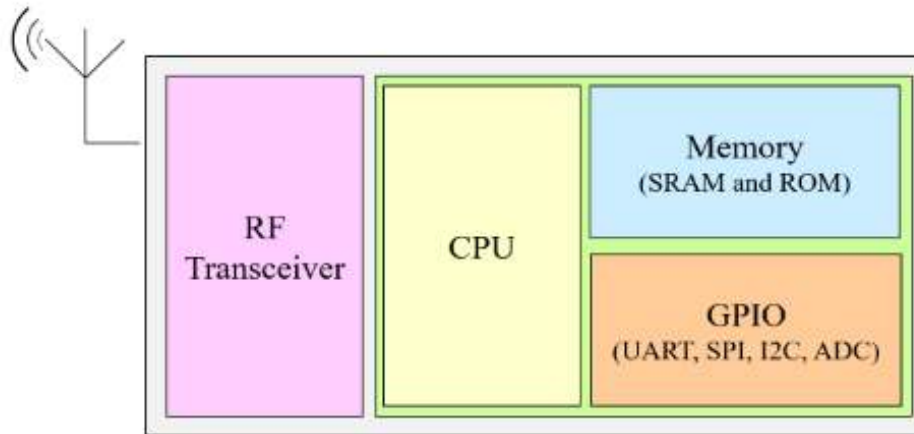
2.4 Υλικό

2.4.1 Μικροελεγκτές και Μονάδες Επεξεργασίας

Ο μικροελεγκτής (Microcontroller Unit - MCU) αποτελεί το κεντρικό δομικό στοιχείο ενός IoT κόμβου, αναλαμβάνοντας τη διαχείριση των αισθητήρων, την τοπική επεξεργασία των δεδομένων και τον έλεγχο των μονάδων επικοινωνίας. Στο σύγχρονο τοπίο του Internet of Things, η τάση μετατοπίζεται από τους απλούς μικροελεγκτές προς την αρχιτεκτονική System-on-Chip (SoC), όπου σε ένα μόνο ολοκληρωμένο κύκλωμα ενσωματώνονται ο επεξεργαστής, η μνήμη και οι ελεγκτές ασύρματης δικτύωσης.

Η Σειρά ESP8266

Παρουσιάστηκε το 2014 και είναι ένα Wi-Fi SoC (System-on-Chip) που ενσωματώνει επεξεργαστή 32-bit RISC. Λειτουργεί στα 2.5V έως 3.6V και διαθέτει λειτουργίες ύπνου (sleep modes) για εξοικονόμηση ενέργειας. Το SoC επικοινωνεί με τα περιφερειακά μέσω τυπικών διεπαφών, όπως UART και SPI. Διατίθεται σε τρεις μορφές: ως γυμνό τσιπ, ως



Σχήμα 5: Δομικά στοιχεία του SoC ESP8266 [24].

έτοιμη μονάδα και ως πλακέτα ανάπτυξης. Οι μονάδες είναι επιφανειακής στήριξης και επιτρέπουν τη σύνδεση στο διαδίκτυο μέσω των προτύπων IEEE 802.11 b/g/n με πιστοποιήσεις WPA/WPA2 [24].

Οι πλακέτες ανάπτυξης παράγονται από διάφορους κατασκευαστές για την ταχεία υλοποίηση έργων IoT.

- ESP-01: Είναι η πιο βασική και οικονομική μονάδα, η οποία κατασκευάστηκε από την Ai-Thinker το 2014. Λόγω του μικρού μεγέθους της, είναι ιδανική για εφαρμογές με περιορισμένο χώρο. Διαθέτει δύο ψηφιακές ακίδες γενικής χρήσης (GPIO0, GPIO2) και δύο ακίδες για σειριακή επικοινωνία (TX/RX), αλλά δεν διαθέτει αναλογική είσοδο.
- ESP-07: Αποτελεί βελτιωμένη έκδοση με περισσότερες λειτουργίες, διαθέτοντας 16 ακίδες, εκ των οποίων οι 11 λειτουργούν ως GPIO. Περιλαμβάνει μία αναλογική είσοδο (0-1V) και έναν σύνδεσμο U.FL που επιτρέπει τη σύνδεση εξωτερικής κεραίας για την ενίσχυση του σήματος Wi-Fi.
- ESP-12: Είναι ιδανική επιλογή όταν απαιτούνται περισσότερες ακίδες για αξιόπιστη συνδεσιμότητα. Οι νεότερες εκδόσεις, όπως το ESP-12E και το ESP-12F, διαθέτουν 11 GPIO, αναλογική είσοδο και διεπαφές SPI. Το ESP-12F διακρίνεται για τον καλύτερο σχεδιασμό κεραίας που προσφέρει μεγαλύτερη εμβέλεια.

Η Σειρά ESP32

Η σειρά ESP32, που κυκλοφόρησε το 2016 ως διάδοχος του ESP8266, προσφέρει σημαντικά αναβαθμισμένες δυνατότητες για πιο απαιτητικά έργα IoT. Είναι ένα υβριδικό σύστημα (SoC) που συνδυάζει Wi-Fi και Bluetooth [24]. Με διπύρνηνο επεξεργαστή 32-bit (Tensilica Xtensa LX series), ενσωματωμένο TCP/IP stack. Η επικοινωνία με τα περιφερειακά γίνεται μέσω κλασικών διεπαφών όπως UART, SPI και I2C.

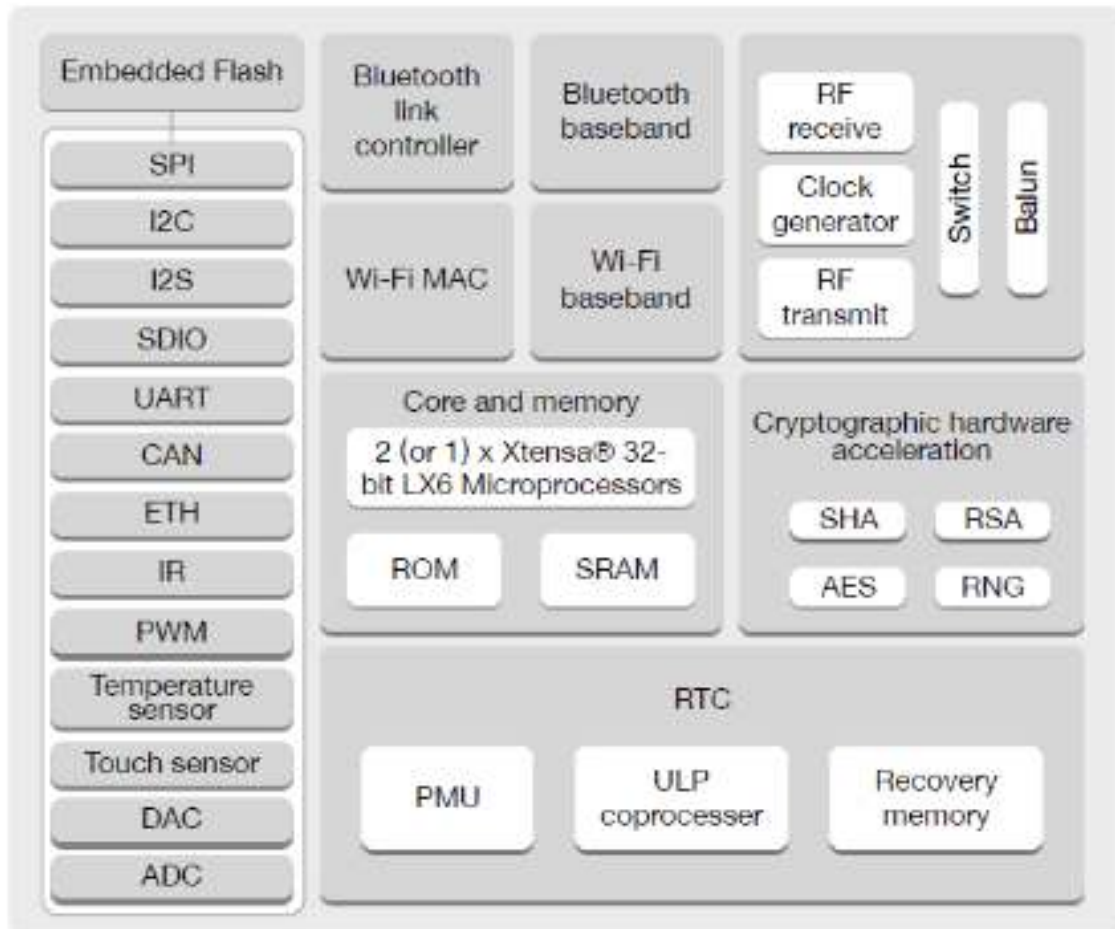
Κατανάλωση Ενέργειας:

- Σε κανονική λειτουργία, απαιτεί 160-260 mA στα 3.3 V.
- Με ενεργά το Wi-Fi και το Bluetooth, η κατανάλωση μπορεί να φτάσει τα 790 mA.
- Για εξοικονόμηση ενέργειας, διαθέτει διάφορες καταστάσεις ύπνου: modem sleep, light sleep, deep sleep και hibernation.

Το ESP32 διατίθεται ως απλό τσιπ, ως έτοιμη μονάδα (module) ή ως πλακέτα ανάπτυξης. Το 2020, η Espressif αναβάθμισε το SoC στην έκδοση ECO V3, η οποία προσφέρει νέο σχεδιασμό στο πυρίτιο, ασφαλή εκκίνηση (secure boot) και κρυπτογράφηση της μνήμης flash. Όλες οι παραλλαγές είναι σε μεγάλο βαθμό συμβατές μεταξύ τους σε επίπεδο κώδικα.

Άλλες παραλλαγές του ESP32 περιλαμβάνουν:

- **ESP32-S2:** Μονοπύρνηνος επεξεργαστής με 43 προγραμματιζόμενα GPIO και μόνο Wi-Fi 4.
- **ESP32-C3:** Μια πιο απλή έκδοση με μονοπύρνηνο επεξεργαστή RISC-V, Wi-Fi 4 και Bluetooth 5 (LE).
- **ESP32-S3:** Διπύρνηνος επεξεργαστής με 44 GPIO, Wi-Fi 4 και Bluetooth 5 (LE). Είναι σχεδιασμένο ειδικά για εφαρμογές Τεχνητής Νοημοσύνης των Πραγμάτων (AIoT)
- **ESP32-C6:** Διαθέτει επεξεργαστή RISC-V και υποστηρίζει Wi-Fi 6 (802.11ax). Προσφέρει μεγαλύτερη εμβέλεια, χαμηλότερη κατανάλωση για συσκευές με μπαταρία και υποστηρίζει τεχνολογίες OFDMA και MU-MIMO για καλύτερη απόδοση σε περιβάλλοντα με πολλές συσκευές.



Σχήμα 6: Λειτουργικά μπλοκ του μικροελεγκτή ESP32 [25].

Οι μονάδες ESP32 προσφέρουν εκτεταμένη ευελιξία και δυνατότητες προσαρμογής κατά την αναθεώρηση του υλικολογισμικού για εφαρμογές ενσωματωμένων συστημάτων. Το ESP32-WROOM-32 αποτελεί μια δημοφιλή μονάδα που μπορεί να χρησιμοποιηθεί σε μια μεγάλη ποικιλία εφαρμογών IoT. Είναι εξοπλισμένο με δυνατότητες Wi-Fi και Bluetooth, τροφοδοτείται από το SoC ESP32-D0WDQ6 και διαθέτει ενσωματωμένη μνήμη SPI Flash μεγέθους 4 MB. Ενσωματώνει ένα πλούσιο σύνολο περιφερειακών μέσω διεπαφών ADC, DAC, SPI, UART, I2S, I2C, PWM, SD και Ethernet. Η μονάδα ESP32-WROOM-32 παράγεται με διαφορετικούς τύπους κεραιών. Ενώ το ESP32-WROOM-32U διαθέτει σύνδεσμο που επιτρέπει τη σύνδεση εξωτερικής κεραίας, οι μονάδες ESP32-WROOM-32D, ESP32-WROOM-32E και ESP32-WROOM-32UE συνοδεύονται από κεραίες τύπου MIFA, PCB και IPEX αντίστοιχα [24].

Το λειτουργικό διάγραμμα του ESP32 αποκαλύπτει μια εξαιρετικά ολοκληρωμένη αρχιτεκτονική System-on-Chip, η οποία έχει σχεδιαστεί για να υποστηρίζει ένα ευρύ φάσμα

εφαρμογών IoT με ελάχιστα εξωτερικά εξαρτήματα, όπως αποτυπώνεται στο σχήμα 6. Στην καρδιά του συστήματος βρίσκονται δύο μικροεπεξεργαστές Xtensa® 32-bit LX6, οι οποίοι λειτουργούν σε συχνότητες από 80 MHz έως 240 MHz και πλαισιώνονται από εσωτερική μνήμη ROM (448 KB) και SRAM (520 KB). Ιδιαίτερη έμφαση δίνεται στο υποσύστημα ασύρματης επικοινωνίας, το οποίο περιλαμβάνει πλήρεις μονάδες Wi-Fi (802.11 b/g/n) και Bluetooth (v4.2 BR/EDR και BLE), καθώς και το απαραίτητο RF front-end με στοιχεία όπως ο διακόπτης (switch), το balun και η γεννήτρια ρολογιού. Εκτός από τη συνδεσιμότητα, η αρχιτεκτονική περιλαμβάνει ισχυρή υποδομή ασφαλείας μέσω υλικού κρυπτογραφικής επιτάχυνσης για αλγορίθμους SHA, RSA, AES και μια γεννήτρια τυχαίων αριθμών (RNG), διασφαλίζοντας την προστασία των δεδομένων χωρίς την επιβάρυνση των κύριων επεξεργασιών. Η ευελιξία του chip ενισχύεται από μια πληθώρα διεπαφών, όπως SPI, I2C, I2S, UART, Ethernet και CAN, που επιτρέπουν τη διασύνδεση με ποικίλα περιφερειακά και αποθηκευτικά μέσα. Τέλος, το SoC ενσωματώνει προηγμένες δυνατότητες αίσθησης, όπως αισθητήρες θερμοκρασίας και αφής, καθώς και μετατροπείς ADC και DAC, ενώ το υποσύστημα RTC με τον συνεπεξεργαστή ULP (Ultra-Low-Power) και τη μνήμη ανάκτησης επιτρέπει στη συσκευή να εκτελεί μετρήσεις ακόμα και σε κατάσταση βαθιάς ύπνωσης (deep sleep) [25].

Arduino

Η πλατφόρμα Arduino αποτελεί σήμερα ένα από τα πιο διαδεδομένα οικοσυστήματα για την ανάπτυξη ενσωματωμένων συστημάτων και ρομποτικών εφαρμογών παγκοσμίως. Η ιστορία της ξεκινά το 2005 στην Ιταλία, ως ένα πανεπιστημιακό ερευνητικό έργο μιας ομάδας πέντε ατόμων, λαμβάνοντας το όνομά της από έναν Ιταλό μεσαιωνικό βασιλιά. Παρόλο που η αρχική παραγωγή περιοριζόταν σε μόλις 200 μονάδες για τις ανάγκες των φοιτητών, η άμεση αποδοχή του εγχειρήματος οδήγησε στην ίδρυση μιας εταιρείας που μετεξελίχθηκε σε έναν από τους μεγαλύτερους κατασκευαστές μικροελεγκτών στον κόσμο [26].

Η ευελιξία του οικοσυστήματος βασίζεται στην πληθώρα των διαθέσιμων πλακετών, όπως οι εκδόσεις Uno, Mega, Nano και Leonardo, οι οποίες προσφέρουν διαφορετικά χαρακτηριστικά ανάλογα με τις απαιτήσεις του έργου σε μνήμη και θύρες εισόδου/εξόδου. Ένα από τα ισχυρότερα πλεονεκτήματα της πλατφόρμας είναι η δυνατότητα επέκτασης μέσω των Arduino Shields, πρόσθετων εξαρτημάτων που τοποθετούνται άμεσα στις ακίδες της κύριας πλακέτας, επιτρέποντας την προσθήκη νέων λειτουργιών χωρίς την ανάγκη

περίπλοκης καλωδίωσης [26].

Η πλέον διαδεδομένη έκδοση, το Arduino Uno, αποτελεί το πρότυπο αναφοράς για την ανάπτυξη πρωτοτύπων, βασιζόμενο στον μικροελεγκτή ATmega328. Διαθέτει 14 ψηφιακές θύρες εισόδου/εξόδου, εκ των οποίων οι 6 μπορούν να λειτουργήσουν ως έξοδοι PWM (Pulse Width Modulation) για τον έλεγχο φορτίων, 6 αναλογικές εισόδους για την ανάγνωση αισθητήρων και κρύσταλλο χρονισμού στα 16 MHz. Για εφαρμογές όπου οι απαιτήσεις σε θύρες είναι αυξημένες, όπως σε ένα κεντρικό σύστημα ελέγχου αγρού με πολλούς ενεργοποιητές, το Arduino Mega 2560 προσφέρει σημαντικά διευρυμένες δυνατότητες με 54 ψηφιακές και 16 αναλογικές θύρες, καθώς και 4 διεπαφές UART για ταυτόχρονη επικοινωνία με πολλαπλά modules [26]. Στον αντίποδα, για λύσεις που απαιτούν ελαχιστοποίηση του μεγέθους, το Arduino Nano ενσωματώνει σχεδόν όλες τις δυνατότητες του Uno σε μια εξαιρετικά μικρή επιφάνεια, καθιστώντας το ιδανικό για ενσωμάτωση σε περιορισμένα περιβλήματα. Παράλληλα, ειδικές εκδόσεις όπως το Arduino Leonardo χρησιμοποιούν τον μικροελεγκτή ATmega32u4, ο οποίος διαθέτει ενσωματωμένο ελεγκτή USB. Αυτό επιτρέπει στην πλακέτα να αναγνωρίζεται από έναν ηλεκτρονικό υπολογιστή ως αυτόνομη συσκευή εισόδου (HID), όπως πληκτρολόγιο ή ποντίκι, διευρύνοντας τις δυνατότητες αλληλεπίδρασης του συστήματος [26].

Raspberry Pi

Το Raspberry Pi αποτελεί μια οικογένεια υπολογιστών που σχεδιάστηκε από το Raspberry Pi Foundation, έναν φιλανθρωπικό οργανισμό με έδρα το Ηνωμένο Βασίλειο, με αρχικό σκοπό την εκπαιδευτική χρήση. Πρόκειται για έναν μικρό και οικονομικό υπολογιστή σχεδιασμένο σε μία πλακέτα, ο οποίος ενσωματώνει όλα τα βασικά εξαρτήματα ενός υπολογιστικού συστήματος σε ένα μοναδικό μικροσίπ (System on Chip - SoC), συμπεριλαμβανομένου του επεξεργαστή (CPU) και της μονάδας επεξεργασίας γραφικών (GPU). Παρόλο που η συσκευή δεν περιλαμβάνει οθόνη, ποντίκι ή πληκτρολόγιο, διαθέτει όλες τις απαραίτητες διεπαφές σύνδεσης, όπως θύρες USB, έξοδο HDMI, διεπαφή GPIO, έξοδο ήχου και υποδοχή κάρτας SD για το λειτουργικό σύστημα και την αποθήκευση δεδομένων [27].

Η εξέλιξη των μοντέλων Raspberry Pi χαρακτηρίζεται από συνεχή βελτίωση της ταχύτητας και των δυνατοτήτων τους:

- **Raspberry Pi Zero & Zero W:** Το Pi Zero είναι το μικρότερο και οικονομικότερο μοντέλο με μονοπύρηνο επεξεργαστή 1GHz και 512MB RAM. Η έκδοση Zero W

(2017) προσθέτει ασύρματη συνδεσιμότητα LAN και Bluetooth.

- **Raspberry Pi 1 & 2:** Τα πρώτα μοντέλα (A+ και B+) αναβαθμίστηκαν το 2014 με 40 ακίδες GPIO και υποδοχή micro SD. Το Raspberry Pi 2 (2015) εισήγαγε τετραπύρηνο επεξεργαστή 900MHz και 1GB RAM.
- **Raspberry Pi 3:** Η τρίτη γενιά (2016-2018) βελτίωσε τις ταχύτητες (1.2GHz - 1.4GHz) και ενσωμάτωσε Wi-Fi και Bluetooth απευθείας στην πλακέτα.
- **Raspberry Pi 4 Model B:** Το πλέον εξελιγμένο μοντέλο που περιγράφεται διαθέτει τετραπύρηνο επεξεργαστή 1.5GHz, επιλογές μνήμης RAM 2GB, 4GB ή 8GB, θύρες USB 3.0, Gigabit Ethernet και υποστήριξη για δύο οθόνες 4K μέσω micro HDMI.

Το επίσημο λειτουργικό σύστημα είναι το Raspbian (βασισμένο στο Linux Debian), το οποίο είναι βελτιστοποιημένο για την αρχιτεκτονική ARM και χρησιμοποιεί το ελαφρύ γραφικό περιβάλλον εργασίας PIXEL.

Προγραμματιζόμενο Λογικό Ελεγκτή (PLC)

Ο Προγραμματιζόμενος Λογικός Ελεγκτής (PLC) αποτελεί το βασικό στοιχείο των συστημάτων αυτοματισμού, το οποίο σχεδιάστηκε αρχικά το 1968 από την General Motors για να αντικαταστήσει τα δύσκαμπτα συστήματα με ρελέ. Η αρχιτεκτονική του βασίζεται σε μια Κεντρική Μονάδα Επεξεργασίας (CPU) που λειτουργεί ως ο "εγκέφαλος" του συστήματος, επεξεργαζόμενη δεδομένα από ψηφιακές και αναλογικές εισόδους (όπως αισθητήρες πίεσης ή στάθμης) για να ελέγξει αντίστοιχες εξόδους στο πεδίο. Τα συστήματα αυτά είναι κατασκευασμένα να αντέχουν σε σκληρές βιομηχανικές συνθήκες, όπως υψηλές θερμοκρασίες και υγρασία, διατηρώντας την αξιοπιστία τους μέσω ειδικών μονάδων τροφοδοσίας και μπαταριών ασφαλείας που προστατεύουν το πρόγραμμα στη μνήμη RAM [28].

Ο προγραμματισμός των PLC εκτελείται μέσω ειδικών εργαλείων, όπως ηλεκτρονικοί υπολογιστές ή συσκευές χειρός, και βασίζεται σε γλώσσες όπως το Διάγραμμα Κλίμακας και τα Συναρτησιακά Μπλοκ. Οι τεχνικές προγραμματισμού διακρίνονται σε γραμμικές, όπου οι εντολές επεξεργάζονται διαδοχικά, και σε δομημένες, όπου οι λειτουργίες οργανώνονται σε ανεξάρτητα μπλοκ για μεγαλύτερη ευελιξία. Στο Ladder Diagram χρησιμοποιούνται διάφοροι τύποι εντολών, όπως εντολές ρελέ (NO/NC επαφές), χρονικά (TON/TOF) και μετρητές (CTU/CTD), καθώς και εντολές αριθμητικών πράξεων και μεταφοράς δεδομένων (GET/PUT) για τον έλεγχο σύνθετων βιομηχανικών διεργασιών [28].

Στη σύγχρονη εποχή, η τεχνολογία των PLC έχει επεκταθεί μέσω της σύνδεσης με προγράμματα SCADA για την εποπτεία και τη συλλογή δεδομένων, καθώς και μέσω της ενσωμάτωσης στο Internet. Αυτό επιτρέπει την τηλεπαρακολούθηση των συστημάτων σε πραγματικό χρόνο, όπου τα δεδομένα αποθηκεύονται σε κεντρικούς διακομιστές (Servers) και γίνονται προσβάσιμα σε απομακρυσμένους χρήστες μέσω Web εφαρμογών. Η εξέλιξη αυτή καθιστά τις διαδικασίες παραγωγής πιο αποδοτικές και ανταγωνιστικές, επιτρέποντας την παρέμβαση στο πρόγραμμα και την ανάπτυξη του συστήματος από οπουδήποτε, μειώνοντας παράλληλα τα ανθρώπινα σφάλματα [28].

STM32

Εισαγωγή και Κατηγοριοποίηση ARM

Οι επεξεργαστές ARM (Acorn RISC Machine) βασίζονται σε αρχιτεκτονική RISC 32-bit ή 64-bit και προτιμώνται για την ικανότητά τους να επιτυγχάνουν υψηλές ταχύτητες με πολύ χαμηλή κατανάλωση ενέργειας σε συσκευές όπως κινητά τηλέφωνα και υπολογιστές. Η αρχιτεκτονική ARM χωρίζεται σε τρεις κύριες κλάσεις ανάλογα με τον τομέα εφαρμογής: **Classic ARM**, **Embedded Cortex (σειρές M και R)** και **Application Cortex (σειρά A)**. Ειδικότερα, η σειρά Cortex-A αναπτύχθηκε για εφαρμογές υψηλών επιδόσεων, η σειρά Cortex-R για έργα πραγματικού χρόνου όπως η διαχείριση κινητήρων στην αυτοκινητοβιομηχανία, ενώ η σειρά Cortex-M εστιάζει σε οικονομικές ηλεκτρονικές συσκευές και εκπαιδευτικές πλακέτες ελέγχου [29].

Η Οικογένεια Cortex-M και η STMicroelectronics

Η οικογένεια επεξεργαστών Cortex-M προτιμάται ευρέως από εταιρείες που αναπτύσσουν ενσωματωμένα συστήματα, με σημαντικότερο εκπρόσωπο την STMicroelectronics, η οποία ιδρύθηκε το 1987 και αποτελεί σήμερα τον μεγαλύτερο Ευρωπαίο κατασκευαστή τσιπ [29]. Το 2007, η εταιρεία ξεκίνησε την παραγωγή του STM32F1, του πρώτου προϊόντος της οικογένειας ελεγκτών STM32 που βασίζεται στον 32-bit επεξεργαστή Cortex-M. Οι ελεγκτές αυτοί είναι ταχύτεροι και φθηνότεροι από πολλούς ελεγκτές 8-bit, προσφέροντας ταυτόχρονα το πλεονέκτημα της εύκολης μετάβασης μεταξύ διαφορετικών μοντέλων της ίδιας οικογένειας χωρίς σημαντικές τροποποιήσεις στο λογισμικό.

Προγραμματισμός και Εργαλεία Ανάπτυξης STM32

Ο προγραμματισμός των πλακετών STM32 μπορεί να γίνει είτε σε γλώσσα χαμηλού επιπέδου (Assembly) είτε σε γλώσσα υψηλού επιπέδου όπως η C [29]. Ενώ οι γλώσσες αυτές

απαιτούν επαρκή γνώση της οργάνωσης της μνήμης και του συνόλου εντολών, το εργαλείο STM32Cube επιτρέπει τη γραφική διαμόρφωση του επεξεργαστή χωρίς να απαιτείται εξειδικευμένη γνώση του υλικού [29]. Παρόλο που υπάρχουν πολλοί συντάκτες και προγράμματα αποσφαλμάτωσης όπως τα Keil, MikroC και IAR, το TrueSTUDIO προτιμάται συχνά καθώς αποτελεί ένα εμπορικά ενισχυμένο περιβάλλον ανάπτυξης C/C++ που υποστηρίζεται επίσημα από την ST.

2.4.2 Αισθητήρες

Αισθητήρας DHT11

Ο αισθητήρας DHT11 αποτελεί μία από τις πιο διαδεδομένες επιλογές στην αγορά για τη μέτρηση περιβαλλοντικών συνθηκών, καθώς συνδυάζει τις λειτουργίες μέτρησης θερμοκρασίας και υγρασίας σε μία ενιαία μονάδα [30]. Λόγω της παραγωγής ψηφιακού σήματος και της ευκολίας υλοποίησής του, χρησιμοποιείται ευρέως σε εφαρμογές χαμηλής πολυπλοκότητας, όπως η παρακολούθηση θερμοκρασίας δωματίου, οι αυτοματισμοί σε θερμοκήπια και τα συστήματα Internet of Things (IoT) [30]. Παρά την ευρεία χρήση του, οι συγκριτικές μελέτες επιδόσεων έναντι άλλων αισθητήρων αναδεικνύουν συγκεκριμένους περιορισμούς στην ακρίβεια και τη σταθερότητα των μετρήσεών του [30].

Σε εργαστηριακές συνθήκες βαθμονόμησης, ο DHT11 παρουσίασε τιμή αβεβαιότητας $0,18^{\circ}\text{C}$ [30]. Η στατιστική ανάλυση έδειξε συντελεστή συσχέτισης 0,999098, ο οποίος αν και θεωρείται υψηλός, παραμένει ο χαμηλότερος μεταξύ των αισθητήρων της ίδιας κατηγορίας (όπως οι DHT22 και DS18B20) [30]. Κατά τη διάρκεια δοκιμών σύγκρισης με πρότυπα θερμόμετρα, ο αισθητήρας εμφάνισε μέσο σφάλμα της τάξης του 3,12% [30]. Ειδικότερα, στις μετρήσεις υπό ελεγχόμενα σημεία αναφοράς (setpoints), παρατηρήθηκε ότι η ακρίβεια μειώνεται σε υψηλότερες θερμοκρασίες, με την αναγκαία τιμή διόρθωσης να φτάνει τους $-1,54^{\circ}\text{C}$ στο setpoint των 40°C [30].

Αισθητήρας DHT22

Ο αισθητήρας DHT22 (γνωστός και ως AM2302) αποτελεί την αναβαθμισμένη έκδοση του DHT11, προσφέροντας διευρυμένο εύρος μετρήσεων και βελτιωμένη ακρίβεια, παραμένοντας παράλληλα μια προσιτή λύση για περιβαλλοντική παρακολούθηση [30]. Όπως και ο προκάτοχός του, ενσωματώνει τη μέτρηση θερμοκρασίας και σχετικής υγρασίας σε ένα ψηφιακό σήμα εξόδου, γεγονός που διευκολύνει τη διασύνδεσή του με μικροελεγκτές χωρίς την ανάγκη εξωτερικών κυκλωμάτων προσαρμογής [30]. Στο πλαίσιο συγκριτικών

μελετών, ο DHT22 κατατάσσεται ως μία ενδιαμέση λύση υψηλότερων προδιαγραφών σε σχέση με τον DHT11, παρουσιάζοντας καλύτερη απόδοση σε ένα ευρύ φάσμα εφαρμογών, από τον έλεγχο υδροπονικών συστημάτων έως την παρακολούθηση δωματίων διακομιστών [30].

Σε εργαστηριακές συνθήκες βαθμονόμησης, ο DHT22 επέδειξε μέσο σφάλμα μέτρησης 1,96%, τιμή σημαντικά βελτιωμένη σε σύγκριση με το 3,12% που παρουσιάζει ο DHT11 [30]. Η αβεβαιότητα μέτρησης (U_{95}) προσδιορίστηκε στους $0,18^{\circ}\text{C}$, ενώ ο συντελεστής συσχέτισης (R^2) με τα πρότυπα όργανα αναφοράς ανήλθε στο εντυπωσιακό 0,999976 [30]. Κατά τη διαδικασία βαθμονόμησης σε διαφορετικά επίπεδα θερμοκρασίας, οι αναγκαίες τιμές διόρθωσης για τον DHT22 βρέθηκαν να είναι $+0,36^{\circ}\text{C}$ για τους 20°C , $+0,46^{\circ}\text{C}$ για τους 30°C και $+0,66^{\circ}\text{C}$ για τους 40°C [30]. Τα δεδομένα αυτά υποδεικνύουν μια σταθερή και προβλέψιμη συμπεριφορά του αισθητήρα σε όλο το εύρος λειτουργίας του [30].

Αισθητήρας DHT20

Ο αισθητήρας DHT20 αποτελεί μία σύγχρονη λύση για τη μέτρηση θερμοκρασίας και σχετικής υγρασίας, η οποία διαφοροποιείται από τους προκατόχους της (DHT11/22) κυρίως λόγω της χρήσης του πρωτοκόλλου επικοινωνίας I2C [31]. Σύμφωνα με τις τεχνικές προδιαγραφές, ο αισθητήρας υποστηρίζει ένα ευρύ φάσμα λειτουργίας, με εύρος μέτρησης θερμοκρασίας από -40 έως $+80^{\circ}\text{C}$ και σχετικής υγρασίας από 0 έως 100% [31]. Η ονομαστική του ακρίβεια ορίζεται στους $\pm 0,5^{\circ}\text{C}$ για τη θερμοκρασία και $\pm 3\%$ για την υγρασία, ενώ η υψηλή του ανάλυση ($0,01^{\circ}\text{C}$ και $0,01\%RH$) επιτρέπει την καταγραφή ακόμη και μικρών διακυμάνσεων στο περιβάλλον [31]. Παρά τις βελτιωμένες προδιαγραφές, η βιβλιογραφική αναζήτηση δείχνει ότι ο DHT20 είναι ακόμη λιγότερο διαδεδομένος στην ερευνητική κοινότητα σε σύγκριση με τα παλαιότερα μοντέλα της σειράς DHT [31].

Η εργαστηριακή αξιολόγηση του αισθητήρα σε ελεγχόμενες συνθήκες κλιματικού θαλάμου ανέδειξε εξαιρετικές επιδόσεις, ιδιαίτερα στον τομέα της θερμοκρασίας αέρα. Συγκεκριμένα, ο DHT20 πέτυχε τα βέλτιστα αποτελέσματα μεταξύ επτά διαφορετικών αισθητήρων χαμηλού κόστους, παρουσιάζοντας συντελεστή προσδιορισμού $R^2 = 0,9999$ και το χαμηλότερο μέσο τετραγωνικό σφάλμα ($RMS E = 0,1836^{\circ}\text{C}$) [31]. Στον τομέα της σχετικής υγρασίας, ο αισθητήρας επέδειξε επίσης γραμμική συμπεριφορά ($R^2 = 0,9951$), αν και παρατηρήθηκαν ορισμένες δυσκολίες στην πλήρη παρακολούθηση των ακραίων τιμών (πολύ υψηλής ή πολύ χαμηλής υγρασίας) σε σύγκριση με τα όργανα αναφοράς [31]. Επιπλέον, η θερμογραφική ανάλυση επιβεβαίωσε ότι ο αισθητήρας δεν παρουσιάζει φαι-

νόμμενα τοπικής υπερθέρμανσης κατά τη λειτουργία του, γεγονός που θα μπορούσε να εισάγει συστηματικά σφάλματα στις μετρήσεις [31].

Από άποψη ενεργειακής διαχείρισης, ο DHT20 κρίνεται ως μία ιδιαίτερα αποδοτική επιλογή για εφαρμογές που βασίζονται σε μπαταρίες. Συμπερασματικά, η υιοθέτηση του DHT20 προσφέρει έναν ιδανικό συνδυασμό υψηλής ακρίβειας στη θερμοκρασία και χαμηλής κατανάλωσης ισχύος. Η χρήση του σε συνδυασμό με απλές μεθόδους γραμμικής παλινδρόμησης για τη διόρθωση των τιμών υγρασίας μπορεί να προσφέρει μετρήσεις υψηλής αξιοπιστίας, εφάμιλλες επαγγελματικού εξοπλισμού, για την αξιολόγηση των περιβαλλοντικών συνθηκών [31].

Αισθητήρες Υγρασίας Εδάφους Βασισμένοι στην Αντίσταση

Οι αισθητήρες υγρασίας τύπου αντίστασης (resistive-type sensors) λειτουργούν μετρώντας την ηλεκτρική αντίσταση μεταξύ δύο ηλεκτροδίων που εισάγονται στο έδαφος [32]. Καθώς η περιεκτικότητα σε νερό μεταβάλλεται, αλλάζει η τιμή της αντίστασης του εδάφους, γεγονός που επηρεάζει το ρεύμα που διέρχεται μεταξύ των ηλεκτροδίων του αισθητήρα [32]. Το σήμα αυτό ενισχύεται και μετατρέπεται συνήθως σε έξοδο τάσης DC, η οποία μπορεί να αντιστοιχιστεί σε ποσοστό υγρασίας μέσω μιας προκαθορισμένης καμπύλης βαθμονόμησης [32].

Αυτή η κατηγορία αισθητήρων αντιπροσωπεύει τη χαμηλότερη δυνατή δαπάνη στην αγορά, με χαρακτηριστικά παραδείγματα όπως ο πειραματικός αισθητήρας MSMS (flat thin mm-sized sensor), ο οποίος κατασκευάζεται με τεχνικές χάραξης χρυσών δίσκων (CDs) και έχει εξαιρετικά χαμηλό κόστος, περίπου \$1 ανά μονάδα. Παράλληλα, η έρευνα έχει στραφεί στη χρήση νανοϋλικών, όπως οι κβαντικές κουκκίδες γραφενίου (GQDs), για τη δημιουργία αισθητήρων αντίστασης με πολύ υψηλή ευαισθησία και χαμηλό κόστος κατασκευής.

Παρά το οικονομικό τους πλεονέκτημα, οι αισθητήρες αγωγιμότητας παρουσιάζουν σημαντικά μειονεκτήματα που περιορίζουν τη χρήση τους σε μακροχρόνιες εφαρμογές. Το κυριότερο πρόβλημα είναι η φθορά των ηλεκτροδίων λόγω του φαινομένου της ηλεκτρόλυσης, καθώς η άμεση επαφή του μετάλλου με το υγρό χώμα οδηγεί σε οξείδωση και διάβρωση [32]. Επιπλέον, η ακρίβεια των μετρήσεων είναι επιρρεπής σε ολίσθηση (drift) λόγω της επίδρασης της θερμοκρασίας, καθιστώντας απαραίτητη τη συνεχή αντιστάθμιση των τιμών κατά τις μετρήσεις πεδίου [32]. Αυτοί οι περιορισμοί έχουν οδηγήσει πολλούς ερευνητές στην επιλογή χωρητικών αισθητήρων ως μια πιο αξιόπιστη εναλλακτική λύση

χαμηλού κόστους [32].

Αισθητήρες Υγρασίας Εδάφους Βασισμένοι στη Χωρητικότητα

Οι αισθητήρες χωρητικότητας αποτελούν μία από τις πιο διαδεδομένες διηλεκτρικές μεθόδους μέτρησης της υγρασίας του εδάφους, βασιζόμενοι στη σημαντική διαφορά μεταξύ της διηλεκτρικής σταθεράς του νερού (περίπου 80) και των υπόλοιπων συστατικών του εδάφους, όπως ο αέρας (περίπου 1) και τα στερεά σωματίδια (περίπου 4) [32]. Η λειτουργία τους στηρίζεται σε ένα κύκλωμα RC όπου ο αισθητήρας και το περιβάλλον έδαφος λειτουργούν ως ένας ισοδύναμος πυκνωτής, στον οποίο εφαρμόζεται ένα σήμα διέγερσης τετραγωνικού κύματος [32]. Καθώς μεταβάλλεται η περιεκτικότητα του εδάφους σε νερό, μεταβάλλεται η διηλεκτρική του σταθερά και κατ' επέκταση η ισοδύναμη χωρητικότητα του αισθητήρα, επηρεάζοντας τις καμπύλες φόρτισης και εκφόρτισης, οι οποίες τελικά μετατρέπονται σε μια ανάλογη έξοδο τάσης [32].

Ένα από τα κύρια πλεονεκτήματα αυτών των αισθητήρων είναι το εξαιρετικά χαμηλό κόστος τους σε σύγκριση με άλλες τεχνολογίες, όπως οι αισθητήρες TDR (Time-Domain Reflectometry) που μπορεί να κοστίζουν έως και \$3000, καθιστώντας τους ιδανικούς για ευρείας κλίμακας εφαρμογές στο *IoT* και την ψηφιακή γεωργία [32]. Συγκεκριμένα μοντέλα χαμηλού κόστους, όπως ο αισθητήρας SKU:SEN0193 της DFRobot, χρησιμοποιούν μια μονωτική επικάλυψη που αποτρέπει την άμεση επαφή των ηλεκτροδίων με το υγρό έδαφος, εξαλείφοντας το πρόβλημα της διάβρωσης λόγω ηλεκτρόλυσης και αυξάνοντας σημαντικά τη διάρκεια ζωής της συσκευής [32]. Επιπλέον, οι αισθητήρες αυτοί χαρακτηρίζονται από χαμηλή κατανάλωση ενέργειας και μπορούν να συνδεθούν απευθείας σε πλακέτες *IoT* για τη λήψη δεδομένων σε πραγματικό χρόνο [32].

Ωστόσο, η ακρίβεια των χωρητικών αισθητήρων επηρεάζεται από διάφορους περιβαλλοντικούς και εγγενείς παράγοντες, με κυριότερους τη θερμοκρασία του εδάφους, την αλατότητα και τον τύπο της εδαφικής υφής [32]. Η έρευνα επισημαίνει ότι οι εργοστασιακές βαθμονομήσεις συχνά δεν επαρκούν για όλες τις εφαρμογές, καθιστώντας αναγκαία την εφαρμογή ειδικών μοντέλων βαθμονόμησης (soil-specific calibration) ανάλογα με τον τύπο του εδάφους [32]. Παράλληλα, η σταθερότητα των μετρήσεων εξαρτάται άμεσα από την ποιότητα της εγκατάστασης και το βάθος τοποθέτησης, καθώς η σωστή συμπίεση του εδάφους γύρω από τον αισθητήρα είναι απαραίτητη προϋπόθεση για την αποφυγή σφαλμάτων κατά τη λειτουργία του [32].

Αισθητήρες Ανακλαστομετρίας Πεδίου Χρόνου (TDR - Time-Domain Reflectometry)

Οι αισθητήρες TDR βασίζονται στη φυσική αρχή ότι η ταχύτητα διάδοσης ενός ηλεκτρομαγνητικού παλμού μεταβάλλεται ανάλογα με τη διηλεκτρική σταθερά του μέσου. Κατά τη λειτουργία τους, μια γεννήτρια εκπέμπει έναν παλμό ο οποίος ταξιδεύει μέσω ομοαξονικού καλωδίου προς την ακίδα του αισθητήρα, όπου εμφανίζεται ασυμφωνία σύνθετης αντίστασης (impedance mismatch). Ο αισθητήρας μετρά τη χρονική διαφορά μεταξύ των ανακλώμενων σημάτων για να προσδιορίσει την υγρασία. Θεωρούνται από τους πλέον ακριβείς αισθητήρες και παρουσιάζουν τη μικρότερη ολίσθηση λόγω θερμοκρασίας (temperature drift), ωστόσο το κόστος τους είναι πολύ υψηλό, φτάνοντας έως και τα \$3000 [32].

Εκτός από την ογκομετρική υγρασία, οι προηγμένοι αισθητήρες TDR (όπως η σειρά CS650 της Campbell Scientific) μπορούν να μετρήσουν ταυτόχρονα τη θερμοκρασία του εδάφους και τη μαζική ηλεκτρική αγωγιμότητα (Bulk EC). Η μέτρηση της αγωγιμότητας είναι ιδιαίτερα χρήσιμη για την εκτίμηση της αλατότητας του εδάφους.

Αισθητήρες Ανακλαστομετρίας Πεδίου Συχνότητας (FDR - Frequency Domain Reflectometry)

Η μέθοδος FDR βασίζεται στη μέτρηση των μεταβολών της συχνότητας που προκαλούνται από τις αλλαγές στη διηλεκτρική σταθερά του εδάφους. Σε αυτό το σύστημα, το έδαφος λειτουργεί ως διηλεκτρικό υλικό και η ακίδα του αισθητήρα λειτουργεί ως πυκνωτής που μαζί με έναν εξωτερικό ταλαντωτή σχηματίζουν ένα κύκλωμα συντονισμού. Η αύξηση της υγρασίας αυξάνει την ισοδύναμη χωρητικότητα, γεγονός που επηρεάζει άμεσα τη συχνότητα συντονισμού (resonant frequency) του αισθητήρα. Οι αισθητήρες FDR προσφέρουν μεγαλύτερη ευελιξία στο σχήμα και το μήκος των ακίδων σε σχέση με τους TDR, αλλά είναι πιο ευαίσθητοι στις μεταβολές της θερμοκρασίας του εδάφους [32].

Αισθητήρες Λόγου Στάσιμων Κυμάτων (SWR - Standing Wave Ratio)

Η τεχνολογία SWR βασίζεται στην αρχή των στάσιμων κυμάτων της θεωρίας μικροκυμάτων. Μια πηγή παράγει ένα σήμα (συνήθως στα 100 MHz) το οποίο, κατά τη μετάδοσή του στην ακίδα, προκαλεί ασυμφωνία σύνθετης αντίστασης. Το προσπίπτον και το ανακλώμενο σήμα υπερτίθενται, σχηματίζοντας ένα στάσιμο κύμα. Η διαφορά τάσης που προκύπτει μεταβάλλεται ανάλογα με τη σύνθετη αντίσταση της ακίδας, η οποία καθορίζεται από τη διηλεκτρική σταθερά του εδάφους. Είναι μια μέθοδος πιο εύκολη στην

υλοποίηση και σημαντικά φθηνότερη από τις άλλες δύο, αν και υστερεί ελαφρώς σε ακρίβεια και δυνατότητα εναλλαγής μεταξύ διαφορετικών αισθητήρων [32].

Παρόλο που η κύρια χρήση τους είναι η υγρασία του εδάφους, η αρχή SWR έχει χρησιμοποιηθεί επιτυχώς και για τη μη καταστροφική μέτρηση της περιεκτικότητας σε νερό στον κορμό των φυτών (plant stem water content). Σε εμπορικό επίπεδο, οι αισθητήρες αυτοί συχνά ενσωματώνονται σε συστήματα που παρακολουθούν τη συνολική τάση του νερού στο έδαφος (soil water tension) [32].

2.5 Επισκόπηση Ερευνητικών Συστημάτων και Πειραματικών Διατάξεων

Στην παρούσα ενότητα πραγματοποιείται κριτική σύνθεση επιλεγμένων ερευνητικών εργασιών που αφορούν την υλοποίηση συστημάτων έξυπνης γεωργίας. Η ανάλυση εστιάζει σε τρεις βασικούς άξονες: την αρχιτεκτονική του υλικού και του λογισμικού, τα πρωτόκολλα επικοινωνίας που υιοθετήθηκαν, και τους τεχνικούς περιορισμούς που αναδείχθηκαν κατά την πειραματική αξιολόγηση. Οι εργασίες οργανώνονται θεματικά, ξεκινώντας από τα απλούστερα αυτόνομα συστήματα τοπικού ελέγχου και καταλήγοντας σε πολύπλοκες αρχιτεκτονικές με δυνατότητες μεγάλης εμβέλειας και ενσωμάτωσης τεχνητής νοημοσύνης. Ο Πίνακας 1 στο τέλος της ενότητας συνοψίζει συγκριτικά τα τεχνικά χαρακτηριστικά κάθε προσέγγισης.

2.5.1 Τοπικά Συστήματα Αυτόνομου Ελέγχου Άρδευσης

Στο απλούστερο επίπεδο σχεδίασης εντοπίζονται υλοποιήσεις που βασίζονται αποκλειστικά σε τοπικό έλεγχο κλειστού βρόχου χωρίς σύνδεση δικτύου. Οι [33] παρουσίασαν ένα σύστημα αυτόματης άρδευσης χαμηλού κόστους βασισμένο στον μικροελεγκτή Arduino UNO και αισθητήρα υγρασίας εδάφους. Ο αλγόριθμος λήψης αποφάσεων ακολουθεί απλή λογική κατωφλίου: η αντλία νερού ενεργοποιείται μέσω μονάδας ρελέ όταν η τιμή υγρασίας υπερβεί ένα σταθερό όριο, ενώ η κατάσταση προβάλλεται σε τοπική οθόνη LCD 16x2. Η πειραματική αξιολόγηση κατέδειξε σημαντική μείωση της σπατάλης νερού σε σχέση με τη χειροκίνητη άρδευση, ωστόσο ο περιορισμένος χαρακτήρας του συστήματος (απουσία απομακρυσμένης πρόσβασης, εξάρτηση από εξωτερική τροφοδοσία) το καθιστά κατάλληλο αποκλειστικά για οικιακή χρήση.

Σε αντίστοιχη κατεύθυνση αλλά με μεγαλύτερη πολυπλοκότητα, οι [34] ανέπτυξαν σύστημα στάγδην άρδευσης ακριβείας σε περιβάλλον θερμοκηπίου για την καλλιέργεια

πιπεριάς cayenne. Η αρχιτεκτονική βασίζεται σε μικροελεγκτή Arduino Mega 2560, ο οποίος διαχειρίζεται ταυτόχρονα 16 αισθητήρες υγρασίας YL-69 και 16 ηλεκτροβάνες μέσω τεσσάρων ρελέ τεσσάρων καναλιών. Ο βρόχος ελέγχου εκτελείται ανά δύο δευτερόλεπτα, στοχεύοντας στη διατήρηση της υγρασίας εντός του εύρους 60%–80%. Τα πειραματικά αποτελέσματα κατέδειξαν επιτυχή ρύθμιση στο 91,7% των γλαστρών, αν και η αστοχία ενός αισθητήρα ανέδειξε την ανάγκη μηχανισμών αυτοδιάγνωσης. Ο κύριος περιορισμός παραμένει η κλιμάκωση, καθώς ο αριθμός αισθητήρων δεσμεύεται από τις αναλογικές εισόδους του μικροελεγκτή.

2.5.2 Συστήματα Συνδεδεμένα με Υπολογιστικό Νέφος μέσω Wi-Fi

Σε υψηλότερο επίπεδο πολυπλοκότητας, αρκετές ερευνητικές ομάδες ενσωμάτωσαν δυνατότητες απομακρυσμένης πρόσβασης μέσω πρωτοκόλλων Wi-Fi και MQTT. Οι [35] σχεδίασαν ένα ολοκληρωμένο σύστημα αυτοματοποίησης άρδευσης center-pivot με ιεραρχική αρχιτεκτονική Master-Slave. Ο κεντρικός ελεγκτής, βασισμένος σε μικροελεγκτή ESP32, συλλέγει δεδομένα από αισθητήρες υγρασίας εδάφους, θερμοκρασίας (DHT22) και βροχής, ενώ η επικοινωνία με το υπολογιστικό νέφος πραγματοποιείται μέσω MQTT broker. Ο αλγόριθμος λήψης αποφάσεων ενσωματώνει χρονοδιαγράμματα άρδευσης ανά καλλιέργεια, ενώ η πρόσβαση των χρηστών διασφαλίζεται μέσω εφαρμογών Flutter (κινητό) και Angular SPA (ιστός). Η χρήση ηλιακής ενέργειας στο πεδίο εξασφαλίζει αυτονομία, αν και η εξάρτηση από σύνδεση Wi-Fi περιορίζει τη χωρική εμβέλεια.

Μια εναλλακτική προσέγγιση στον ίδιο τεχνολογικό χώρο προτάθηκε από τους [36], οι οποίοι αξιοποίησαν Raspberry Pi 4 σε συνδυασμό με το οπτικό περιβάλλον ροών Node-RED για τη δημιουργία ενός συστήματος edge computing. Αισθητήρες DHT11 και υγρασίας εδάφους τροφοδοτούν τον αλγόριθμο αποφάσεων, ο οποίος συνδυάζει πολλαπλά κατώφλια (υγρασία εδάφους < 50%, υγρασία αέρα < 70%, θερμοκρασία > 32°C) και ενσωματώνει δεδομένα από το Open Weather Forecast API για πρόβλεψη βροχόπτωσης. Τα δεδομένα αποθηκεύονται στην πλατφόρμα Favoriot μέσω MQTT, ενώ παρέχεται δυνατότητα χειροκίνητου ελέγχου μέσω dashboard. Η χρήση Raspberry Pi προσφέρει σημαντικά μεγαλύτερη υπολογιστική ισχύ σε σχέση με απλούς μικροελεγκτές, αλλά με αυξημένο ενεργειακό αποτύπωμα και κόστος.

2.5.3 Αυτόνομα Συστήματα Μεγάλης Εμβέλειας με Τεχνολογία LoRa

Για γεωργικές εκτάσεις μεγάλης κλίμακας όπου η κάλυψη Wi-Fi είναι ανέφικτη, η τεχνολογία LoRa αποτελεί τη βασική εναλλακτική. Οι [37] ανέπτυξαν ένα ηλιακά τροφο-

δοτούμενο δίκτυο ασύρματων αισθητήρων (WSN) με τοπολογία αστέρα για τη μέτρηση ογκομετρικής υγρασίας εδάφους (VWC). Κάθε κόμβος βασίζεται σε ESP8266 NodeMCU με εξωτερικό LoRa SX1278 module, χωρητικό αισθητήρα υγρασίας SEN0193 και ηλιακό πάνελ 6V με μπαταρία Li-ion 18650. Η χρήση λειτουργίας deep sleep (κατανάλωση 0,25 mA σε αδράνεια έναντι 70–75 mA σε λειτουργία) εξασφαλίζει θεωρητική αυτονομία 44 ημερών χωρίς ηλιοφάνεια. Η βαθμονόμηση των αισθητήρων μέσω βαρυμετρικής μεθόδου κατέδειξε τυπικό σφάλμα 7–9%, ενώ παρατηρήθηκε ότι η έκθεση στον ήλιο επηρεάζει σημαντικά τους ρυθμούς εξάτμισης μεταξύ κόμβων.

Σε παρόμοια λογική, οι [38] πραγματοποίησαν εκτεταμένη εξάμηνη δοκιμή πεδίου πολλαπλών τύπων κόμβων IoT, αξιολογώντας τόσο ESP32 με Wi-Fi/HTTP όσο και επεξεργαστές ASR605x και STM32WLE5JC με LoRaWAN. Η γκάμα αισθητήρων ήταν ευρεία, καλύπτοντας μετρήσεις περιβάλλοντος (TMP116N, HDC2080, AS7262), εδάφους (Watermark 200SS, χωρητικοί PIM520/SEN0193) και βροχόπτωσης (μηχανικός κάδος HadeX, οπτικός RG-15). Τα δεδομένα αποθηκεύονταν σε Google Sheets μέσω Wi-Fi ή στο δίκτυο The Things Network μέσω LoRaWAN. Η δοκιμή σε ακραίες καιρικές συνθήκες ανέδειξε κρίσιμα πρακτικά ζητήματα: η ξήρανση του εδάφους προκαλούσε απώλεια επαφής των χωρητικών αισθητήρων, η τάση των μπαταριών μεταβαλλόταν αντιστρόφως ανάλογα με τη θερμοκρασία, και κατά την ταυτόχρονη ανάγνωση πολλαπλών αισθητήρων παρατηρούνταν αμοιβαίες παρεμβολές που απαιτούσαν γαλβανική απομόνωση.

2.5.4 Πολυλειτουργικά Συστήματα με Ενσωμάτωση Μηχανικής Μάθησης

Πέρα από τον αυτόματο έλεγχο άρδευσης, ορισμένες ερευνητικές προσεγγίσεις επεκτείνουν το εύρος λειτουργιών κατά τρόπο ολιστικό. Οι [39] πρότειναν ένα σπονδυλωτό μοντέλο βασισμένο σε ESP8266EX που συνδυάζει την προστασία καλλιεργειών από εισβολείς (μέσω αισθητήρα κίνησης PIR και ηχητικών αποτρεπτικών μηχανισμών) με τη μέτρηση περιβαλλοντικών παραμέτρων (DHT11, αισθητήρας υγρασίας εδάφους) και τη λήψη αποφάσεων μέσω αλγορίθμων μηχανικής μάθησης στο υπολογιστικό νέφος. Η αρχιτεκτονική χρησιμοποιεί επικοινωνία πολλαπλών αλμάτων (multi-hop) μέσω Wi-Fi/GSM, ενώ η τροφοδοσία βασίζεται σε μπαταρίες 18650 Li-ion με ρυθμιστή MB102. Η πειραματική αξιολόγηση κατέδειξε ακρίβεια 94–99% για τους αισθητήρες και 97–98% για τους ενεργοποιητές, αν και οι ασταθείς καιρικές συνθήκες και η έλλειψη τεχνικών γνώσεων από τους τελικούς χρήστες παραμένουν ανοιχτές προκλήσεις.

2.5.5 Συγκριτική Αξιολόγηση

Ο Πίνακας 1 συγκεντρώνει τα βασικά τεχνικά χαρακτηριστικά των εξεταζόμενων ερευνητικών υλοποιήσεων. Από τη σύγκριση προκύπτουν ορισμένα κοινά μοτίβα: η πλειονότητα των συστημάτων βασίζεται σε αλγόριθμους κατωφλίου για τη λήψη αποφάσεων άρδευσης, ενώ η επιλογή μεταξύ τοπικού ελέγχου και σύνδεσης στο υπολογιστικό νέφος καθορίζεται κυρίως από τις ανάγκες κλιμάκωσης. Παράλληλα, διαπιστώνεται ότι κανένα από τα εξεταζόμενα συστήματα δεν ενσωματώνει ταυτόχρονα ασφαλή εγγραφή συσκευών, υβριδική αρχιτεκτονική βάσεων δεδομένων και σύγχρονο περιβάλλον οπτικοποίησης — κενό που επιχειρεί να καλύψει η παρούσα εργασία.

Πίνακας 1: Συγκριτικός πίνακας ερευνητικών υλοποιήσεων συστημάτων έξυπνης γεωργίας.

Αναφορά	Υλικό (MCU)	Αισθητήρες	Επικοινωνία	Αρχιτεκτονική	Βασικοί Περιορισμοί
[33]	Arduino UNO (ATmega328)	Υγρασία εδάφους (ωμικός)	Καμία (τοπικός έλεγχος)	Κλειστός βρόχος, οθόνη LCD	Χωρίς απομακρυσμένη πρόσβαση· μόνο οικιακή χρήση
[34]	Arduino Mega 2560	YL-69 × 16	Καμία (ενσύρματος)	Έλεγχος ανά γλάστρα, 16 ηλεκτροβάνες	Μέγ. 16 αισθητήρες ανά MCU· αστοχία αισθητήρων
[35]	ESP32 (Master/Slave)	Υγρασία εδάφους, DHT22, αισθ. βροχής	Wi-Fi, MQTT	Master-Slave, Cloud (MQTT broker), SPA	Εξάρτηση από κάλυψη Wi-Fi
[36]	Raspberry Pi 4	DHT11, υγρασία εδάφους	Wi-Fi, MQTT	Edge computing (Node-RED), Favoriot Cloud	Υψηλή ενεργ. κατανάλωση Raspberry Pi
[37]	ESP8266 NodeMCU V3	Χωρητικός SEN0193	LoRa (SX1278)	WSN τοπολογία αστέρα, ηλιακή τροφοδοσία	Σφάλμα 7–9%· ενεργειακή εξάρτηση από ηλιοφάνεια
[38]	ESP32, ASR605x, STM32WLE5JC	TMP116N, HDC2080, Watermark, PIM520/SEN0193	Wi-Fi (HTTP), LoRaWAN (TTN)	Πολλαπλοί τύποι κόμβων, Google Sheets/TTN	Αποκόλληση αισθητήρων σε ξηρό έδαφος· παρεμβολές
[39]	ESP8266EX	PIR, DHT11, υγρασία εδάφους	Wi-Fi/GSM multi-hop	Σπονδυλωτή (4 modules), ML στο Cloud	Ασταθείς συνθήκες· έλλειψη τεχν. γνώσεων χρηστών

3 Ανάλυση Απαιτήσεων & Προδιαγραφές

3.1 Εισαγωγή και Πεδίο Εφαρμογής

Η παρούσα ενότητα ορίζει το πλαίσιο ανάπτυξης της πλατφόρμας Agropos, μιας ολοκληρωμένης λύσης στον τομέα της Έξυπνης Γεωργίας. Το σύστημα σχεδιάστηκε για να προσφέρει μια ενοποιημένη εμπειρία διαχείρισης γεωργικών δεδομένων, δίνοντας έμφαση στην ασφάλεια, την επεκτασιμότητα και την ευελιξία στη συνδεσιμότητα.

3.1.1 Σκοπός του Συστήματος

Ο κύριος σκοπός του Agropos είναι η γεφύρωση του χάσματος μεταξύ των φυσικών μετρήσεων στο πεδίο και της λήψης αποφάσεων μέσω ψηφιακών εργαλείων. Η πλατφόρμα στοχεύει στην παροχή μιας multi-tenant αρχιτεκτονικής, όπου πολλαπλοί χρήστες μπορούν να διαχειρίζονται ανεξάρτητα τους αγρούς και τις συσκευές τους με πλήρη απομόνωση δεδομένων. Επιπλέον, επιδιώκει την υποστήριξη υβριδικής συνδεσιμότητας, επιτρέποντας σε IoT κόμβους να επικοινωνούν είτε μέσω αστικών δικτύων (Wi-Fi) είτε μέσω τεχνολογιών μεγάλης εμβέλειας (LoRa) για απομακρυσμένες καλλιέργειες.

3.1.2 Τεχνικό Πεδίο Εφαρμογής

Το πεδίο εφαρμογής της διπλωματικής εργασίας εκτείνεται σε τρία βασικά επίπεδα αρχιτεκτονικής. Στο επίπεδο των IoT κόμβων, πραγματοποιήθηκε η ανάπτυξη firmware για μικροελεγκτές ESP32. Υλοποιήθηκαν μηχανισμοί εξοικονόμησης ενέργειας μέσω της λειτουργίας Deep Sleep και συστήματα τοπικής διαχείρισης ρυθμίσεων στη μνήμη NVS. Παράλληλα, υποστηρίζονται πολλαπλοί τύποι αισθητήρων, όπως θερμοκρασίας, υγρασίας αέρα και εδάφους.

Στο επίπεδο της επικοινωνίας και της δρομολόγησης, το σύστημα αξιοποιεί τα πρωτόκολλα HTTP και MQTT για τη μεταφορά των δεδομένων. Για την κάλυψη των αναγκών σε περιοχές χωρίς δικτυακή υποδομή, αναπτύχθηκε ένας αυτόνομος LoRa Gateway που γεφυρώνει τους LoRa κόμβους με το διαδίκτυο, εξασφαλίζοντας τη διαφανή προώθηση των πακέτων δεδομένων από το πεδίο προς την κεντρική υποδομή.

Τέλος, στο επίπεδο της εφαρμογής και της οπτικοποίησης, αναπτύχθηκε το κεντρικό σύστημα σε περιβάλλον Laravel 12 για τη διαχείριση της επιχειρηματικής λογικής και της αυθεντικοποίησης. Για την αποδοτική διαχείριση του μεγάλου όγκου μετρήσεων χρησιμοποιείται η βάση InfluxDB, η οποία εξειδικεύεται στην αποθήκευση δεδομένων χρο-

νοσειρών. Η διεπαφή του χρήστη υλοποιήθηκε σε Vue.js, παρέχοντας ένα διαδραστικό περιβάλλον με δυνατότητες απεικόνισης της πληροφορίας σε πραγματικό χρόνο μέσω της τεχνολογίας WebSockets.

3.2 Προσωπικότητες Χρηστών

Η σχεδίαση της πλατφόρμας Agropos βασίστηκε στη μεθοδολογία της Σχεδίασης με Κέντρο τον Χρήστη, όπου οι λειτουργικές απαιτήσεις προκύπτουν από τις συγκεκριμένες ανάγκες, τους περιορισμούς και το τεχνικό υπόβαθρο των τελικών χρηστών. Για τον σκοπό αυτό, ορίστηκαν τρεις αντιπροσωπευτικές προσωπικότητες, οι οποίες καλύπτουν το φάσμα των χρηστών της εφαρμογής και δικαιολογούν τις αρχιτεκτονικές επιλογές του συστήματος.

3.2.1 Persona 1: Μαρία (Η Αστική Καλλιεργήτρια)

Η Μαρία αντιπροσωπεύει τον ερασιτέχνη καλλιεργητή που διατηρεί έναν μικρό κήπο σε αστικό περιβάλλον και διαθέτει βασικές γνώσεις τεχνολογίας χωρίς εξειδίκευση στον προγραμματισμό. Η κύρια ανάγκη της είναι μια λύση χαμηλού κόστους που να εγκαθίσταται εύκολα και να αξιοποιεί την υπάρχουσα οικιακή υποδομή Wi-Fi. Λόγω του περιορισμένου τεχνικού υποβάθρου, η συγκεκριμένη χρήστης αδυνατεί να παραμετροποιήσει τον κώδικα της συσκευής για την εισαγωγή διαπιστευτηρίων δικτύου. Αυτή η απαίτηση οδήγησε στην υλοποίηση του Captive Portal για το δυναμικό Wi-Fi provisioning, επιτρέποντας τη ρύθμιση της συσκευής μέσω γραφικού περιβάλλοντος στον browser, καθώς και στη χρήση του πρωτοκόλλου HTTP POST για την άμεση και απλή μεταφορά δεδομένων στο backend.

3.2.2 Persona 2: Νίκος (Ο Επαγγελματίας Παραγωγός)

Ο Νίκος αντιπροσωπεύει τον επαγγελματία αγρότη που διαχειρίζεται μεγάλες εκτάσεις σε απομακρυσμένες περιοχές, όπου δεν υφίσταται κάλυψη Wi-Fi ή σταθερή τροφοδοσία ρεύματος. Οι ανάγκες του εστιάζουν στην αξιοπιστία της επικοινωνίας σε μεγάλες αποστάσεις, στη μέγιστη αυτονομία της μπαταρίας και στην ασφάλεια των δεδομένων του, καθώς η πρόσβαση για συντήρηση είναι δύσκολη. Η persona αυτή υπαγόρευσε την ενσωμάτωση της τεχνολογίας LoRa για τη μετάδοση δεδομένων σε μεγάλες αποστάσεις, την αυστηρή εφαρμογή στρατηγικών Deep Sleep για την εξοικονόμηση ενέργειας, καθώς και την υιοθέτηση κρυπτογράφησης AES-128-CTR με προστασία Anti-Replay για τη διασφάλιση της ακεραιότητας των μετρήσεων κατά τη διαδρομή τους από το πεδίο.

3.2.3 Persona 3: Ερευνητής Γεωπόνος

Ο Ερευνητής αντιπροσωπεύει τον επιστημονικό συνεργάτη που αναλύει τα δεδομένα για την εξαγωγή συμπερασμάτων σχετικά με την απόδοση των καλλιεργειών και τη βελτιστοποίηση των πόρων. Απαιτεί πρόσβαση σε ιστορικά δεδομένα υψηλής ανάλυσης, συγκεντρωτικά στατιστικά στοιχεία και εργαλεία σύγκρισης μετρικών χωρίς καθυστερήσεις στην απόκριση του συστήματος. Η ανάγκη αυτή δικαιολογεί την επιλογή της InfluxDB ως εξειδικευμένης βάσης δεδομένων χρονοσειρών για την αποδοτική διαχείριση μεγάλου όγκου πληροφοριών. Παράλληλα, οδήγησε στην ανάπτυξη της υπηρεσίας SensorTime-SeriesService, η οποία επιτρέπει τον υπολογισμό συγκεντρωτικών στατιστικών, όπως η ελάχιστη, η μέγιστη και η μέση τιμή, παρέχοντας στον ερευνητή την απαραίτητη πληροφορία για την επιστημονική τεκμηρίωση των αποτελεσμάτων του.

3.3 Λειτουργικές Απαιτήσεις

Οι λειτουργικές απαιτήσεις του Agropos περιγράφουν τις θεμελιώδεις υπηρεσίες που το σύστημα οφείλει να εκτελεί για τους χρήστες και τις συνδεδεμένες συσκευές. Στο επίκεντρο της πλατφόρμας βρίσκεται η διαχείριση της πολυενοικίασης, η οποία επιτρέπει τον διαχωρισμό των πόρων ανάλογα με τον εγγεγραμμένο χρήστη, προσφέροντας δυνατότητες δημιουργίας αγρών και συσχέτισης αυτών με συγκεκριμένους IoT κόμβους. Κατά τη δημιουργία ενός αγρού, ο χρήστης ορίζει τα γεωγραφικά του όρια μέσω διαδραστικού χάρτη, με τα δεδομένα να αποθηκεύονται σε μορφή GeoJSON Polygon. Οι χρήστες έχουν τη δυνατότητα να καταχωρούν νέους αισθητήρες στο σύστημα, είτε μέσω χειροκίνητης εισαγωγής είτε μέσω σάρωσης κωδικών ταχείας απόκρισης (QR codes). Κατά την εγγραφή κάθε αισθητήρα, καταχωρούνται οι γεωγραφικές συντεταγμένες τοποθέτησής του (γεωγραφικό πλάτος και μήκος), ώστε κάθε μέτρηση να συσχετίζεται μόνιμα με τη συγκεκριμένη φυσική θέση στην οποία ελήφθη. Επιπλέον, οι χρήστες μπορούν να παρακολουθούν την κατάσταση λειτουργίας των συσκευών τους.

Όσον αφορά την αλληλεπίδραση με το υλικό, το σύστημα υποστηρίζει τη διαδικασία αρχικής παραμετροποίησης των κόμβων μέσω γραφικού περιβάλλοντος, επιτρέποντας τη σύνδεση σε τοπικά δίκτυα Wi-Fi και την αυθεντικοποίηση στο backend για την έκδοση ψηφιακών διαπιστευτηρίων. Οι συσκευές είναι υπεύθυνες για την ανάγνωση των φυσικών μεγεθών από το περιβάλλον και τη δρομολόγησή τους προς την κεντρική υποδομή, επιλέγοντας το κατάλληλο πρωτόκολλο επικοινωνίας ανάλογα με τη διαθεσιμότητα των

υπηρεσιών. Στην περίπτωση των LoRa συσκευών, η λειτουργικότητα επεκτείνεται στη διαχείριση κρυπτογραφημένων πακέτων που προωθούνται μέσω πυλών (gateways), διασφαλίζοντας ότι η πληροφορία θα φτάσει στον προορισμό της ακόμη και σε περιβάλλοντα με περιορισμένη συνδεσιμότητα.

Η παρουσίαση της πληροφορίας αποτελεί την τελική λειτουργική απαίτηση, με το σύστημα να παρέχει εργαλεία οπτικοποίησης των μετρήσεων. Οι χρήστες μπορούν να προβάλλουν την τρέχουσα κατάσταση των καλλιεργειών τους, να λαμβάνουν άμεσες ενημερώσεις για νέες μετρήσεις και να ανακτούν ιστορικά στοιχεία για περαιτέρω ανάλυση. Η πλατφόρμα αναλαμβάνει την επεξεργασία των ακατέργαστων δεδομένων, υπολογίζοντας στατιστικά μεγέθη και τάσεις, ώστε να μετατρέπει τις απλές μετρήσεις σε χρήσιμη πληροφορία για τη λήψη αποφάσεων.

3.4 Μη Λειτουργικές Απαιτήσεις

Σε συνέχεια των παραπάνω λειτουργιών, οι μη λειτουργικές απαιτήσεις θέτουν τα ποιοτικά κριτήρια που διασφαλίζουν την ορθή εκτέλεση των υπηρεσιών υπό πραγματικές συνθήκες. Η ασφάλεια αποτελεί τον κρίσιμότερο παράγοντα, επιβάλλοντας την κρυπτογράφηση της πληροφορίας σε όλα τα επίπεδα μεταφοράς. Ενώ η λειτουργική απαίτηση ορίζει την αποστολή των δεδομένων, η μη λειτουργική προδιαγράφει ότι αυτή πρέπει να γίνεται μέσω ασφαλών καναλιών, χρησιμοποιώντας κρυπτογραφικούς αλγορίθμους AES-128 για το δίκτυο LoRa και ψηφιακά διαπιστευτήρια JWT για το Wi-Fi, ώστε να αποτρέπεται η μη εξουσιοδοτημένη πρόσβαση ή η αλλοίωση των μετρήσεων.

Η απόδοση και η επεκτασιμότητα καθορίζουν την ικανότητα του συστήματος να ανταποκρίνεται σε αυξανόμενο φόρτο εργασίας. Προδιαγράφεται η χρήση εξειδικευμένων τεχνολογιών για τη διαχείριση των δεδομένων χρονοσειρών, έτσι ώστε η ανάκτηση και ο υπολογισμός των στατιστικών να ολοκληρώνονται σε ελάχιστο χρόνο, ανεξάρτητα από τον συνολικό όγκο των αποθηκευμένων εγγραφών. Αυτός ο διαχωρισμός της αρχιτεκτονικής διασφαλίζει ότι η οπτικοποίηση των δεδομένων παραμένει αποκρίσιμη, παρέχοντας στον χρήστη μια αδιάλειπτη εμπειρία χρήσης, ακόμη και όταν το σύστημα εξυπηρετεί ταυτόχρονα εκατοντάδες ενεργούς κόμβους.

Τέλος, η ενεργειακή διαχείριση και η αξιοπιστία αποτελούν τους βασικούς περιορισμούς για τη λειτουργία του υλικού στο πεδίο. Οι συσκευές πρέπει να επιδεικνύουν εξαιρετικά χαμηλή κατανάλωση ισχύος, παραμένοντας σε κατάσταση ύπνου για το μεγαλύτερο

μέρος του χρόνου, προκειμένου να εξασφαλιστεί η μακροχρόνια αυτονομία τους. Παράλληλα, η ανοχή σε σφάλματα δικτύου και η ικανότητα ανάκτησης της κατάστασης μετά από επανεκκίνηση είναι απαραίτητες ιδιότητες, ώστε η συλλογή των δεδομένων να παραμένει αδιάλειπτη χωρίς την ανάγκη φυσικής παρουσίας του χρήστη για τη συντήρηση των συσκευών.

3.5 Σενάρια Εφαρμογής

Η ανάλυση των σεναρίων εφαρμογής επιτρέπει την επικύρωση της αρχιτεκτονικής του Agropos, εξετάζοντας πώς οι λειτουργικές και μη λειτουργικές απαιτήσεις ανταποκρίνονται σε διαφορετικά περιβάλλοντα καλλιέργειας. Τα σενάρια αυτά βασίζονται στις προσωπικότητες των χρηστών που ορίστηκαν προηγουμένως και καταδεικνύουν την ικανότητα του συστήματος να προσαρμόζεται σε ποικίλες τεχνολογικές υποδομές και συνθήκες πεδίου. Συνολικά παρουσιάζονται πέντε σενάρια: η ρύθμιση αγρού και η εγγραφή αισθητήρων με γεωγραφική τοποθεσία, η λειτουργία σε αστικό περιβάλλον με Wi-Fi, η ανάπτυξη σε απομακρυσμένη καλλιέργεια μέσω LoRa και, τέλος, η επιστημονική ανάλυση ιστορικών δεδομένων.

3.5.1 Σενάριο A: Καταχώρηση Αγρού και Εγγραφή Αισθητήρα με Γεωγραφική Τοποθεσία

Στο πρώτο σενάριο αποτυπώνεται η διαδικασία με την οποία ο Νίκος ρυθμίζει για πρώτη φορά την παρακολούθηση της καλλιέργειάς του στην πλατφόρμα. Αφού ολοκληρώσει την εγγραφή του, δημιουργεί έναν νέο αγρό μέσω της διεπαφής της εφαρμογής, ορίζοντας τα γεωγραφικά του όρια απευθείας σε έναν διαδραστικό χάρτη. Τα όρια αποθηκεύονται εσωτερικά σε μορφή GeoJSON Polygon, επιτρέποντας τη μελλοντική χαρτογραφική απεικόνιση της κατανομής των αισθητήρων εντός της καλλιέργειας.

Στη συνέχεια, ο Νίκος τοποθετεί φυσικά τους κόμβους με τους αισθητήρες στο χωράφι και προχωρά στη δήλωσή τους στο σύστημα. Η καταχώρηση της συσκευής πραγματοποιείται είτε μέσω σάρωσης του κωδικού QR που φέρει ο κόμβος είτε μέσω χειροκίνητης εισαγωγής του μοναδικού αναγνωριστικού του (Device ID). Αμέσως μετά, εγγράφει κάθε αισθητήρα που είναι συνδεδεμένος στον κόμβο ξεχωριστά: ανοίγει τη φόρμα εισαγωγής αισθητήρα από το κινητό του τηλέφωνο και σαρώνει τον κωδικό QR που φέρει ο αισθητήρας. Η εφαρμογή αναγνωρίζει αυτόματα τον τύπο μέτρησης του αισθητήρα από τα δεδομένα του QR code και ταυτόχρονα αντλεί τις γεωγραφικές συντεταγμένες της τρέχου-

σας θέσης μέσω του GPS του κινητού, συμπληρώνοντας προκαταβολικά τα αντίστοιχα πεδία της φόρμας. Ο Νίκος επιβεβαιώνει ή διορθώνει τα προσυμπληρωμένα στοιχεία και υποβάλλει τη φόρμα. Οι πληροφορίες αυτές αποθηκεύονται στη σχεσιακή βάση δεδομένων ως η μόνιμη ταυτότητα κάθε αισθητήρα, διασφαλίζοντας ότι κάθε μέτρηση που θα ληφθεί στο μέλλον συνοδεύεται αυτόματα από τη φυσική θέση στην οποία τοποθετήθηκε. Από τη στιγμή ολοκλήρωσης της εγγραφής, ο κόμβος είναι έτοιμος να αυθεντικοποιηθεί και να αρχίσει την αποστολή μετρήσεων που αντιστοιχίζονται αυτόματα στον σωστό αγρό.

3.5.2 Σενάριο Β: Αστικός Κήπος και Συνδεσιμότητα Wi-Fi

Στο δεύτερο σενάριο, η Μαρία επιθυμεί να παρακολουθεί την υγρασία του εδάφους στον κήπο της χρησιμοποιώντας την υπάρχουσα υποδομή Wi-Fi της κατοικίας της. Η διαδικασία ξεκινά με την τροφοδοσία του IoT κόμβου, ο οποίος, αναγνωρίζοντας την έλλειψη διαπιστευτηρίων, ενεργοποιεί αυτόματα ένα τοπικό σημείο πρόσβασης. Η Μαρία συνδέεται στη συσκευή μέσω του κινητού της τηλεφώνου και εισάγει τα στοιχεία του οικιακού δικτύου σε ένα απλό γραφικό περιβάλλον, αποφεύγοντας οποιαδήποτε επαφή με τον κώδικα της συσκευής. Μετά την επανεκκίνηση, ο κόμβος συνδέεται στο διαδίκτυο, εκτελεί αυθεντικοποίηση στο backend και λαμβάνει ένα ψηφιακό token για την εξουσιοδότηση των μελλοντικών μεταδόσεων.

Κατά τη διάρκεια της λειτουργίας, ο αισθητήρας καταγράφει περιοδικά τις περιβαλλοντικές συνθήκες και τις προωθεί στην κεντρική υποδομή. Η πλατφόρμα έχει σχεδιαστεί ώστε να υποστηρίζει πολλαπλούς τρόπους επικοινωνίας, εξασφαλίζοντας ότι η πληροφορία θα φτάσει στη βάση δεδομένων είτε υπάρχει ενεργός μεσίτης μηνυμάτων (MQTT broker) στην εγκατάσταση είτε όχι. Η Μαρία ενημερώνεται σε πραγματικό χρόνο μέσω του Dashboard της εφαρμογής, βλέποντας τις μετρήσεις να ανανεώνονται αυτόματα στην οθόνη της. Η αδιάλειπτη ροή των δεδομένων της επιτρέπει να αντιδρά άμεσα στις ανάγκες των φυτών της, επιβεβαιώνοντας την ευχρηστία και την αποτελεσματικότητα του συστήματος σε οικιακό περιβάλλον.

3.5.3 Σενάριο Γ: Απομακρυσμένη Καλλιέργεια και Δίκτυο LoRa

Στο τρίτο σενάριο, ο Νίκος αναπτύσσει το σύστημα σε έναν απομακρυσμένο ελαιώνα όπου δεν υπάρχει κάλυψη δικτύων Wi-Fi. Εγκαθιστά μια κεντρική πύλη LoRa Gateway σε σημείο με πρόσβαση στο διαδίκτυο και τοποθετεί αυτόνομους κόμβους εντός της καλλιέργειας. Οι συσκευές αυτές παραμένουν σε κατάσταση βαθιάς ύπνωσης για τη μέγιστη εξοικονόμηση ενέργειας, ξυπνώντας μόνο για τη λήψη μετρήσεων. Τα δεδομένα μετατρέπονται

σε συμπαγή δυαδική μορφή, κρυπτογραφούνται για τη διασφάλιση της ιδιωτικότητας και αποστέλλονται μέσω ραδιοκυμάτων μεγάλης εμβέλειας προς την πύλη.

Η πύλη λειτουργεί ως γέφυρα, λαμβάνοντας τα πακέτα και προωθώντας τα αυτούσια στο backend. Στο επίπεδο της εφαρμογής, το σύστημα αναγνωρίζει την ταυτότητα της συσκευής και προχωρά στην αποκρυπτογράφηση και την επικύρωση της ακεραιότητας της πληροφορίας. Ο Νίκος, παρά την απουσία άμεσης σύνδεσης των αισθητήρων στο δίκτυο, απολαμβάνει την ίδια ποιότητα πληροφόρησης με κάθε άλλο χρήστη. Μέσω της ανάλυσης των ιστορικών δεδομένων και των στατιστικών στοιχείων που παράγει η πλατφόρμα, είναι σε θέση να προγραμματίζει τις εργασίες του και να παρακολουθεί την υγεία της καλλιέργειας σε βάθος χρόνου, αποδεικνύοντας τη στιβαρότητα της αρχιτεκτονικής σε απαιτητικές συνθήκες πεδίου.

3.5.4 Σενάριο Δ: Ανάλυση Ιστορικών Δεδομένων από τον Ερευνητή Γεωπόνο

Στο τέταρτο σενάριο, ο Ερευνητής Γεωπόνος χρησιμοποιεί την πλατφόρμα με σκοπό την εξαγωγή στατιστικών συμπερασμάτων για την απόδοση της καλλιέργειας κατά μια συγκεκριμένη καλλιεργητική περίοδο. Χρησιμοποιώντας τα εργαλεία φιλτραρίσματος της εφαρμογής, επιλέγει το επιθυμητό χρονικό εύρος και τους αισθητήρες ενδιαφέροντος. Η πλατφόρμα υποβάλλει τα αντίστοιχα ερωτήματα στην InfluxDB, η οποία επιστρέφει με ελάχιστη καθυστέρηση συγκεντρωτικά αποτελέσματα — ελάχιστη, μέγιστη και μέση τιμή — ανεξάρτητα από τον συνολικό όγκο των αποθηκευμένων εγγραφών.

Ο ερευνητής μπορεί να συγκρίνει την εξέλιξη μιας μετρικής, όπως η υγρασία εδάφους, σε διαφορετικές περιοχές της ίδιας καλλιέργειας. Αξιοποιώντας τις γεωγραφικές συντεταγμένες που καταχωρήθηκαν κατά την εγγραφή των αισθητήρων, μπορεί να εντοπίσει χωρικές ανομοιομορφίες στη διανομή υγρασίας ή θερμοκρασίας εντός του αγρού. Το σενάριο αυτό επιβεβαιώνει ότι ο συνδυασμός εξειδικευμένης βάσης δεδομένων χρονοσειρών με τη γεωαναφορά των αισθητήρων παρέχει ουσιαστικά οφέλη για την επιστημονική ανάλυση, μετατρέποντας τις απλές χρονικές σειρές μετρήσεων σε χωροχρονικά δεδομένα υψηλής αξίας.

3.5.5 Σενάριο Ε: Μετεγκατάσταση Αισθητήρα στο Πεδίο

Στο πέμπτο σενάριο, ο Νίκος αποφασίζει να αναδιατάξει κάποιους αισθητήρες εδάφους μέσα στον ελαιώνα του, επειδή μετά την πρώτη καλλιεργητική περίοδο διαπίστωσε ότι ορισμένοι κόμβοι δεν κάλυπταν αντιπροσωπευτικά περιοχές με διαφορετική υδροφορία. Βγάζει τον κόμβο από την παλιά θέση, τον μεταφέρει στο νέο σημείο και τον τοποθετεί.

Στη συνέχεια, ανοίγει την εφαρμογή στο κινητό του τηλέφωνο — ακολουθώντας ακριβώς την ίδια διαδικασία που εκτέλεσε κατά την πρώτη εγγραφή — και σαρώνει το QR code του αισθητήρα.

Η πλατφόρμα αναγνωρίζει ότι ο αισθητήρας με αυτό το UUID υπάρχει ήδη στη βάση δεδομένων και, αντί να δημιουργήσει νέα εγγραφή, ενημερώνει διαφανώς τις γεωγραφικές του συντεταγμένες με τη θέση που αντλεί αυτόματα το GPS του κινητού. Ο Νίκος επιβεβαιώνει τα προσυμπληρωμένα πεδία και υποβάλλει τη φόρμα. Από τη στιγμή αυτή, όλες οι επόμενες μετρήσεις του κόμβου αντιστοιχίζονται αυτόματα στη νέα γεωγραφική θέση, ενώ το ιστορικό των παλαιών μετρήσεων διατηρείται αναλλοίωτο στην InfluxDB με τις αρχικές συντεταγμένες.

Το σενάριο αυτό αποδεικνύει ότι η διπλή ροή της μεθόδου `scan()` δεν αποτελεί τεχνική λεπτομέρεια, αλλά ουσιαστική λειτουργική απαίτηση: η μετεγκατάσταση αισθητήρα πρέπει να είναι εξίσου απλή με την πρώτη εγκατάστασή του, χωρίς η πρόσβαση σε μενού διαχείρισης ή η χειροκίνητη εισαγωγή συντεταγμένων να αποτελέσουν εμπόδιο για τον αγρότη στο πεδίο.

4 Αρχιτεκτονική Συστήματος

Η πλατφόρμα *Agronos* αντιμετωπίζει ένα κλασικό πρόβλημα IoT συστημάτων: πώς να συγκεντρώνονται δεδομένα από ετερογενείς κόμβους πεδίου, να αποθηκεύονται αποδοτικά και να εξυπηρετούνται πολλαπλοί ανεξάρτητοι χρήστες σε κοινή υποδομή. Το κεφάλαιο αυτό περιγράφει τις αρχιτεκτονικές επιλογές που ελήφθησαν για να αντιμετωπιστούν αυτές οι απαιτήσεις.

4.1 Επισκόπηση Αρχιτεκτονικής

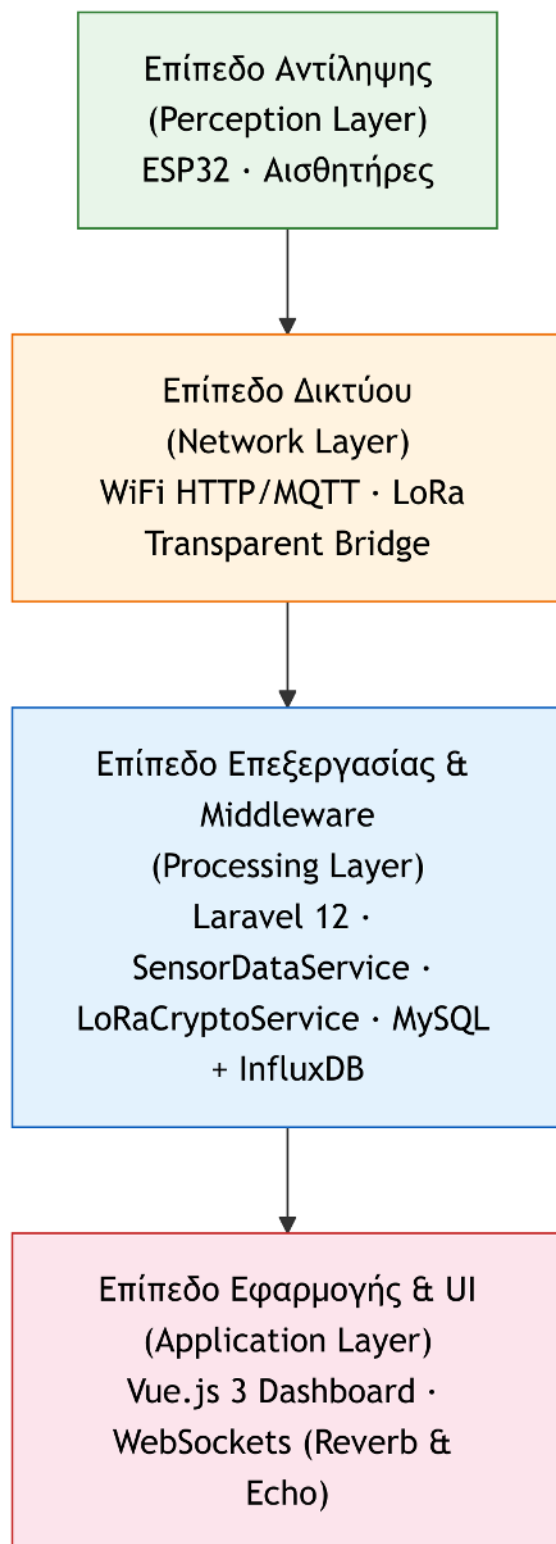
Η πλατφόρμα *Agronos* ακολουθεί ένα **μοντέλο τεσσάρων επιπέδων** (Four-Layer Architecture) [4], το οποίο διαχωρίζει σαφώς τις ευθύνες μεταξύ αντίληψης, δικτύωσης, επεξεργασίας και τελικής εφαρμογής. Στόχος του σχεδιασμού αυτού είναι η χαλαρή σύζευξη (loose coupling), ώστε κάθε στρώμα να μπορεί να εξελίσσεται ανεξάρτητα:

- **Επίπεδο Αντίληψης (Perception Layer):** Περιλαμβάνει τους φυσικούς κόμβους ESP32 και τους αισθητήρες στο πεδίο. Η διαχείριση των μετρήσεων γίνεται μέσω των προτύπων Factory και Strategy, επιτρέποντας την ομοιόμορφη ανάγνωση διαφορετικών τύπων αισθητήρων.
- **Επίπεδο Δικτύου (Network Layer):** Διαχειρίζεται τη μεταφορά δεδομένων μέσω WiFi (HTTP/MQTT) ή LoRa. Το LoRa Gateway λειτουργεί ως διαφανής γέφυρα (transparent bridge): προωθεί κρυπτογραφημένα πακέτα χωρίς πρόσβαση στο περιεχόμενό τους.
- **Επίπεδο Middleware & Υποστήριξης Υπηρεσιών (Service Support Layer):** Αποτελεί το κεντρικό στρώμα επεξεργασίας και ενορχήστρωσης (Laravel 12). Εδώ υλοποιείται η κρυπτογραφική επεξεργασία (LoRaCryptoService), η προστασία από επιθέσεις επανάληψης (Anti-Replay) και η ενορχήστρωση της υβριδικής αποθήκευσης σε MySQL και InfluxDB.
- **Επίπεδο Εφαρμογής & Παρουσίασης (Application Layer):** Αποτελεί τη διεπαφή του χρήστη με το σύστημα (Vue.js 3 SPA (Single-Page Application))¹. Μέσω WebSo-

¹Η υλοποίηση χρησιμοποιεί Inertia.js (@inertiajs/vue3 ^2.0.0), το οποίο δεν αποτελεί παραδοσιακό SPA: η δρομολόγηση σελίδων γίνεται server-side από το Laravel, ενώ το Vue αναλαμβάνει μόνο την απόδοση κάθε σελίδας χωρίς client-side router.

ckets (Laravel Reverb), το επίπεδο αυτό παρέχει real-time οπτικοποίηση των δεδομένων και διαχείριση των αγροτικών πόρων.

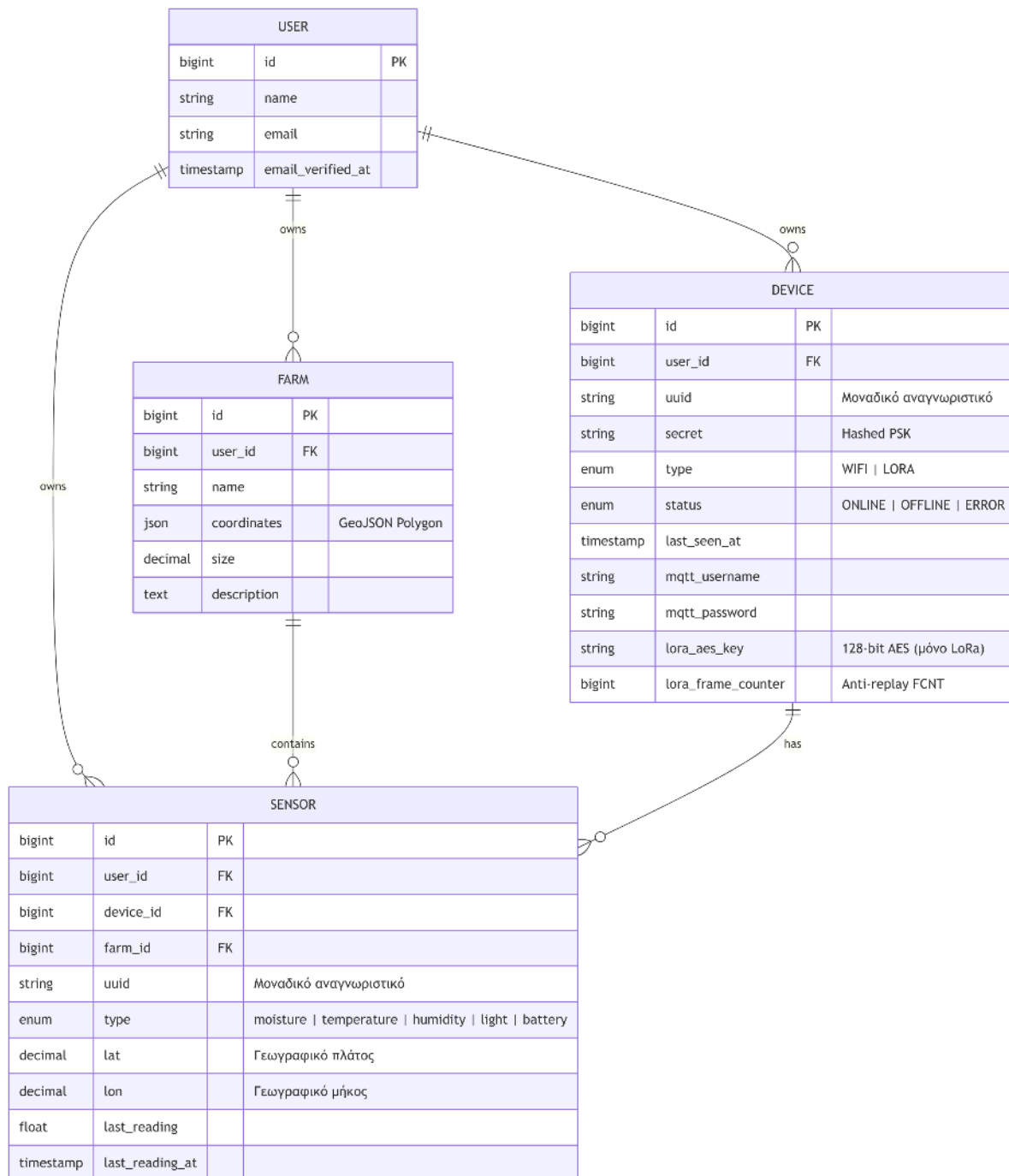
Η συνολική δομή των τεσσάρων επιπέδων απεικονίζεται στο Σχήμα 7.



Σχήμα 7: Μοντέλο τεσσάρων επιπέδων αρχιτεκτονικής της πλατφόρμας *Agronos*

4.2 Μοντέλο Οντοτήτων & Γεωγραφική Σήμανση

Η αρχιτεκτονική του συστήματος *Agropos* βασίζεται σε ένα μοντέλο οντοτήτων που έχει σχεδιαστεί για να αντικατοπτρίζει τις πραγματικές συνθήκες του γεωργικού πεδίου, διαχωρίζοντας τη φυσική υποδομή από τις λογικές μονάδες μέτρησης. Το Σχήμα 8 παρουσιάζει το ER διάγραμμα με τις βασικές οντότητες και τις σχέσεις τους.



Σχήμα 8: ER διάγραμμα με τις βασικές οντότητες και τις σχέσεις τους

4.2.1 Διαχωρισμός Συσκευής και Αισθητήρα

Κεντρική αρχιτεκτονική επιλογή αποτελεί ο σαφής διαχωρισμός μεταξύ των οντοτήτων **Συσκευή** και **Αισθητήρας**:

- **Συσκευή:** Αποτελεί την κεντρική υπολογιστική μονάδα και τον κόμβο επικοινωνίας, αναλαμβάνοντας την αυθεντικοποίηση, τη διασύνδεση με το δίκτυο (WiFi/LoRa) και τη διαχείριση ενέργειας.
- **Αισθητήρας:** Αποτελεί τη μονάδα λήψης δεδομένων, ταυτοποιούμενη από έναν τύπο (π.χ. DHT11, Soil Moisture) και ένα μοναδικό UUID.

4.2.2 Σχέση 1 : N και Γεωγραφική Σήμανση

Το σύστημα υποστηρίζει τη σύνδεση πολλαπλών αισθητήρων σε μία συσκευή (1 : N), προσφέροντας σημαντική βελτιστοποίηση κόστους και επεκτασιμότητα. Σε αντίθεση με συμβατικές προσεγγίσεις, οι **γεωγραφικές συντεταγμένες (Latitude/Longitude)** αποθηκεύονται στο επίπεδο του **Αισθητήρα**.

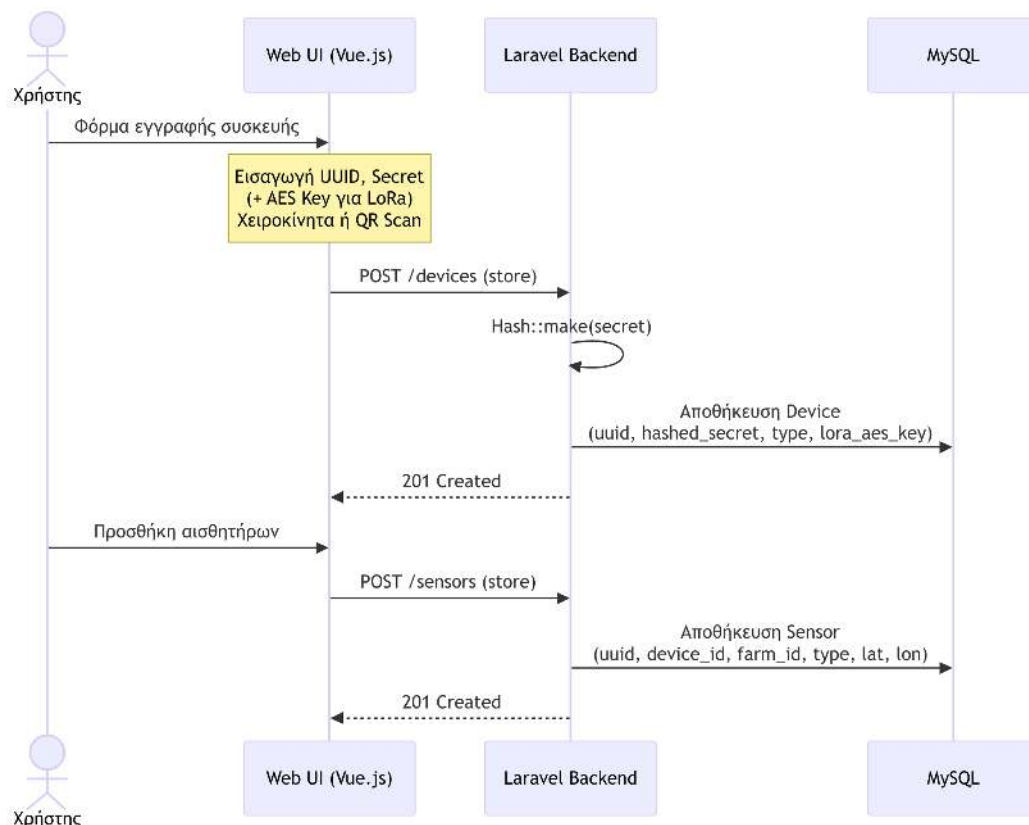
Αυτή η επιλογή επιτρέπει τη μέτρηση διαφορετικών σημείων ενδιαφέροντος με τη χρήση μίας μόνο κεντρικής υπολογιστικής μονάδας, παρέχοντας υψηλή χωρική ανάλυση δεδομένων με μειωμένο κόστος υποδομής. Παράλληλα, επιτυγχάνεται η **λογική αποσύνδεση** του υλικού επικοινωνίας από το σημείο λήψης καθώς η συσκευή λειτουργεί ως ένας κόμβος διαχείρισης ενέργειας και συνδεσιμότητας, ενώ ο αισθητήρας παραμένει η σταθερή μονάδα πληροφορίας.

Σε σενάρια συντήρησης, η αντικατάσταση μιας ελαττωματικής συσκευής δεν επηρεάζει τη γεωγραφική ταυτότητα ή τη συνέχεια των δεδομένων των αισθητήρων στο πεδίο. Μια συσκευή μπορεί επίσης να εξυπηρετεί ταυτόχρονα αισθητήρες **ίδιου τύπου** (π.χ. πολλαπλούς αισθητήρες υγρασίας εδάφους για την κάλυψη διαφορετικών σημείων ενός αγρού) ή **διαφορετικών τύπων** (π.χ. υγρασία εδάφους και θερμοκρασία αέρα) σε διαφορετικά ακριβή σημεία τοποθέτησης.

4.2.3 Προ-κοινοποιημένα Διαπιστευτήρια

Η ασφάλεια και η ταυτοποίηση των κόμβων βασίζονται σε ένα μοντέλο προ-κοινοποιημένων διαπιστευτηρίων (*Pre-shared Credentials*), όπου η ανταλλαγή πληροφοριών μεταξύ Backend και συσκευής προηγείται της ενεργοποίησης στο πεδίο. Συγκεκριμένα, ο χρήστης εισάγει μέσω της διεπαφής ιστού της εφαρμογής —σε μία ενιαία φόρμα— τα μοναδικά

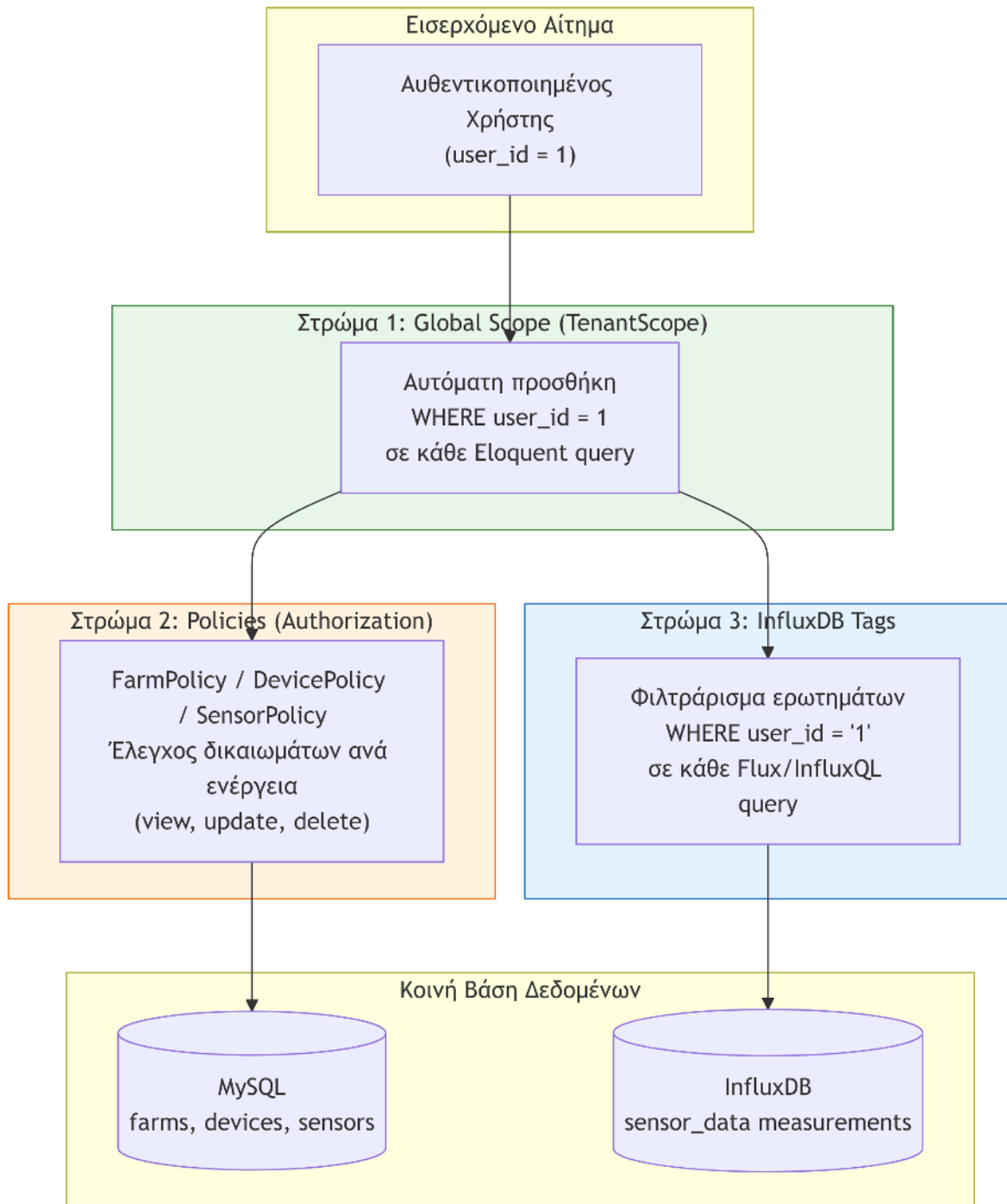
αναγνωριστικά (UUID), τα μυστικά κλειδιά (*Secrets*) των συσκευών, καθώς και το 128-bit AES κλειδί στην περίπτωση των LoRa συσκευών, είτε χειροκίνητα είτε μέσω σάρωσης κώδικα QR. Αυτή η διαδικασία δημιουργεί ένα μοντέλο *Pre-shared Key* (PSK) [19], το οποίο διασφαλίζει ότι το σύστημα αναγνωρίζει και εμπιστεύεται τη συσκευή αμέσως μετά την πρώτη της σύνδεση, επιτρέποντας παράλληλα την end-to-end αποκρυπτογράφηση των δεδομένων χωρίς την ανάγκη περίπλοκων και ενεργοβόρων πρωτοκόλλων ανταλλαγής κλειδιών «στον αέρα». Ο εγγενής περιορισμός του μοντέλου *PSK* έγκειται στην απουσία δυναμικής ανανέωσης κλειδιών (*key rotation*): σε περίπτωση παραβίασης κλειδιού, η ανάκληση και αντικατάστασή του απαιτεί χειροκίνητη επέμβαση μέσω της διεπαφής ιστού. Εναλλακτικά πρωτόκολλα δυναμικής ανταλλαγής κλειδιών, όπως το DTLS ή το TLS-PSK, θα παρείχαν αυτόματη ανανέωση, αλλά θα εισήγαγαν σημαντική ενεργειακή επιβάρυνση στους κόμβους ESP32 και LoRa. Για τις ανάγκες του Agropos σε επίπεδο πρωτοτύπου, ο περιορισμός αυτός κρίθηκε αποδεκτός. Η ακολουθία εγγραφής συσκευής παρουσιάζεται στο Σχήμα 9.



Σχήμα 9: Sequence diagram εγγραφής συσκευής μέσω web UI

4.3 Αρχιτεκτονική Πολυμίσθωσης

Η πλατφόρμα *Agronos* σχεδιάστηκε εξ αρχής ως σύστημα πολυμίσθωσης, όπου πολλοί ανεξάρτητοι χρήστες μοιράζονται την ίδια υποδομή και βάση δεδομένων, διατηρώντας πλήρη απομόνωση μεταξύ των δεδομένων τους. Η αρχιτεκτονική αυτή αποφεύγει το κόστος και την πολυπλοκότητα ξεχωριστών βάσεων δεδομένων ανά χρήστη, ενώ κανένας χρήστης δεν έχει πρόσβαση σε δεδομένα άλλου. Στο Σχήμα 10 παρουσιάζονται τα στρώματα ασφαλείας της αρχιτεκτονικής πολυμίσθωσης.



Σχήμα 10: Διάγραμμα στρωμάτων ασφαλείας multi-tenancy

4.3.1 Στρατηγική Υλοποίησης

Η απομόνωση δεδομένων υλοποιείται στο επίπεδο του ORM (*Object-Relational Mapping*) μέσω του μηχανισμού *Global Scopes* του Laravel. Συγκεκριμένα, ορίζεται ένα *TenantScope* το οποίο εφαρμόζεται αυτόματα σε κάθε ερώτημα βάσης δεδομένων, προσθέτοντας τον περιορισμό `WHERE user_id = <αυθεντικοποιημένος χρήστης>`. Έτσι ο κώδικας δεν χρει-

άζεται να φιλτράρει χειροκίνητα σε κάθε ερώτημα με την απομόνωση γίνεται αυτόματα.

- **Ασφάλεια εκ σχεδιασμού:** Η απομόνωση εφαρμόζεται αυτόματα σε επίπεδο μοντέλου, σε κάθε ερώτημα βάσης δεδομένων.
- **Μηδενική επιβάρυνση ανάπτυξης:** Οι προγραμματιστές γράφουν ερωτήματα χωρίς να φροντίζουν για το φιλτράρισμα ανά χρήστη — το *scope* το αναλαμβάνει αυτόματα.
- **Ελεγχόμενη παράκαμψη:** Για διαχειριστικές λειτουργίες (π.χ. στατιστικά συστήματος), παρέχεται η δυνατότητα παράκαμψης μέσω ρητής κλήσης (`allTenants()`), η οποία απενεργοποιεί το `TenantScope` για το συγκεκριμένο ερώτημα.

Αξίζει να επισημανθεί ότι η απομόνωση μέσω *Global Scopes* λειτουργεί σε επίπεδο εφαρμογής (application-level), και όχι σε επίπεδο βάσης δεδομένων — εναλλακτικές όπως η Row-Level Security (RLS) του PostgreSQL θα παρείχαν ισχυρότερες εγγυήσεις απομόνωσης ανεξάρτητα από τη λογική του ORM. Ωστόσο, η προσέγγιση αυτή επιλέχθηκε για τη συμβατότητά της με το MySQL και την ευκολία ελέγχου (testability) στο πλαίσιο του Laravel. Όπως περιγράφεται στην ακόλουθη ενότητα, ένα δεύτερο επίπεδο Πολιτικών Εξουσιοδότησης (*Policies*) αντισταθμίζει αυτόν τον περιορισμό.

4.3.2 Μηχανισμός Αυτόματης Ανάθεσης

Πέρα από το φιλτράρισμα ανάγνωσης, η αρχιτεκτονική πολυμίσθωσης περιλαμβάνει και μηχανισμό αυτόματης ανάθεσης κατά τη δημιουργία εγγραφών. Κάθε νέα εγγραφή λαμβάνει αυτόματα το `user_id` του αυθεντικοποιημένου χρήστη, εξαλείφοντας τον κίνδυνο δημιουργίας ορφανών εγγραφών ή εσφαλμένης ανάθεσης σε λάθος χρήστη.

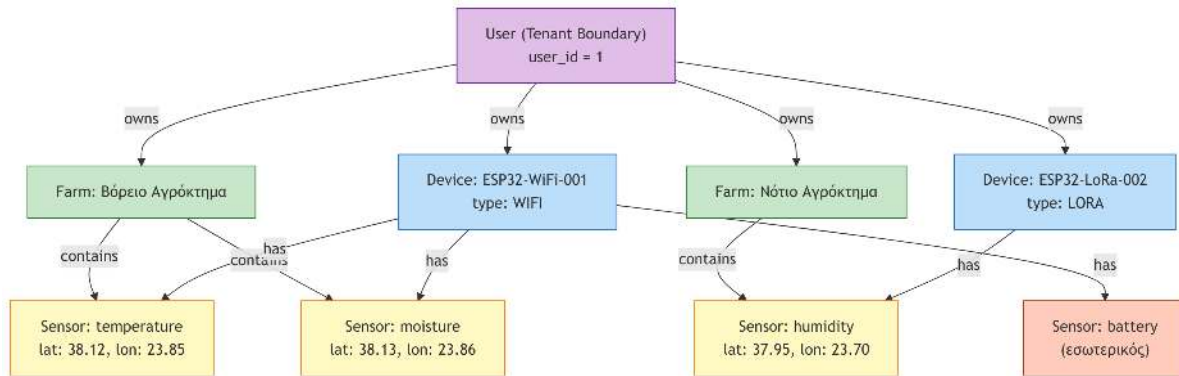
4.3.3 Ιεραρχία Οντοτήτων και Εφαρμογή

Η πολυμίσθωση εφαρμόζεται σε όλες τις βασικές οντότητες του συστήματος, δημιουργώντας μια ιεραρχία απομόνωσης που συνοψίζεται στον Πίνακα 2:

Οντότητα	Πολυμίσθωση	Ρόλος
User	— (ρίζα)	Ο ιδιοκτήτης-μισθωτής
Farm	BelongsToTenant	Αγροτεμάχια του χρήστη
Device	BelongsToTenant	Συσκευές IoT του χρήστη
Sensor	BelongsToTenant	Αισθητήρες του χρήστη

Πίνακας 2: Εφαρμογή πολυμίσθωσης ανά οντότητα

Η ιεραρχία $User \rightarrow Farm \rightarrow Sensor \leftarrow Device$ εξασφαλίζει ότι κάθε αισθητήρας ανήκει ταυτόχρονα σε ένα αγροτεμάχιο (λογική τοποθεσία) και σε μία συσκευή (φυσική υποδομή), ενώ όλες οι οντότητες παραμένουν αυστηρά εντός των ορίων του μισθωτή τους. Το Σχήμα 11 απεικονίζει δένδρικά την ιεραρχία αυτή.



Σχήμα 11: Δένδρικό διάγραμμα ιεραρχίας tenant → entities

4.3.4 Ενίσχυση μέσω Πολιτικών Εξουσιοδότησης

Η απομόνωση μέσω *Global Scopes* συμπληρώνεται από ένα δεύτερο στρώμα ασφαλείας: τις **Πολιτικές Εξουσιοδότησης** (*Policies*) του Laravel. Για κάθε οντότητα (*Farm*, *Device*, *Sensor*) ορίζεται αντίστοιχη πολιτική που ελέγχει αναλυτικά τα δικαιώματα ανά ενέργεια (προβολή, επεξεργασία, διαγραφή). Ο συνδυασμός αυτός προσθέτει ένα ακόμα επίπεδο προστασίας: ακόμη και αν το *Global Scope* παρακαμφθεί για λειτουργικούς λόγους, η πολιτική εξουσιοδότησης αποτρέπει μη εξουσιοδοτημένες ενέργειες.

4.3.5 Απομόνωση στη Βάση Χρονοσειρών

Η απομόνωση δεδομένων επεκτείνεται και στην InfluxDB. Κάθε μέτρηση αισθητήρα αποθηκεύεται με ετικέτα (*tag*) *user_id*, η οποία χρησιμοποιείται ως φίλτρο σε κάθε ερώτημα ανάκτησης χρονοσειρών. Με τον τρόπο αυτό, η αρχή της πολυμίσθωσης διατρέχει ολόκληρο το σύστημα αποθήκευσης — τόσο τη σχεσιακή βάση όσο και τη βάση χρονοσειρών — εξασφαλίζοντας συνεπή απομόνωση σε κάθε επίπεδο.

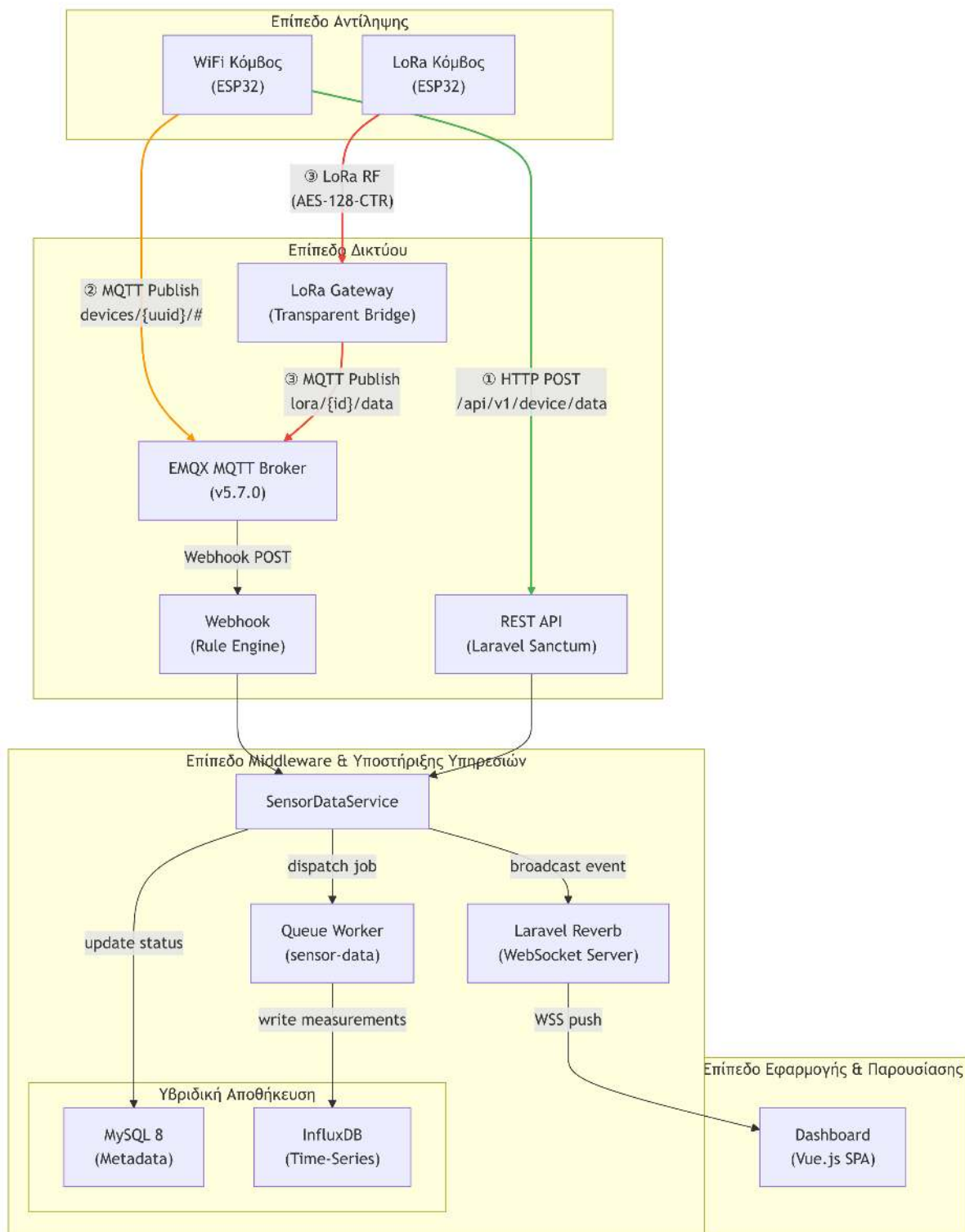
4.4 Επικοινωνιακή Αρχιτεκτονική

Η πλατφόρμα *Agronos* υποστηρίζει τρία διακριτά μονοπάτια επικοινωνίας (*communication paths*) για τη μεταφορά δεδομένων από τους κόμβους πεδίου στο κεντρικό σύστημα. Τα τρία μονοπάτια εξυπηρετούν διαφορετικές ανάγκες, αλλά καταλήγουν στην ίδια υπηρε-

σία επεξεργασίας (SensorDataService), ώστε η επεξεργασία να είναι ενιαία ανεξαρτήτως πρωτοκόλλου:

1. **Μονοπάτι 1 — WiFi → HTTP → Πλατφόρμα:** Σύγχρονη, απευθείας επικοινωνία μέσω REST API.
2. **Μονοπάτι 2 — WiFi → MQTT Broker → Πλατφόρμα:** Ασύγχρονη επικοινωνία μέσω pub/sub πρωτοκόλλου και webhook.
3. **Μονοπάτι 3 — LoRa → Gateway → MQTT Broker → Πλατφόρμα:** Κρυπτογραφημένη μετάδοση μεγάλης εμβέλειας μέσω LoRa, ενδιάμεσου Gateway και MQTT.

Η συνολική εικόνα και των τριών μονοπατιών δεδομένων παρουσιάζεται στο Σχήμα 12.

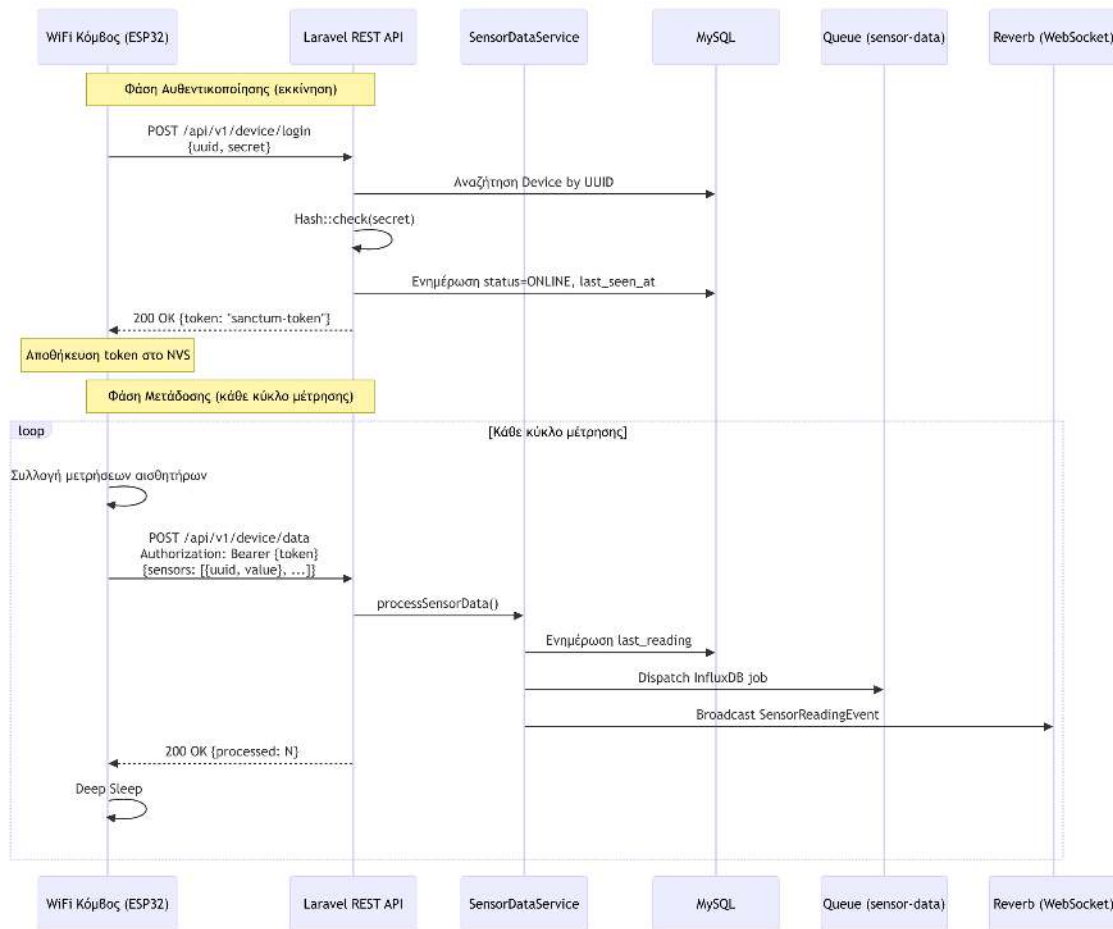


Σχήμα 12: Συνολικό διάγραμμα και των τριών μονοπατιών δεδομένων

4.4.1 Μονοπάτι 1: WiFi → HTTP → Πλατφόρμα

Το πρώτο μονοπάτι αποτελεί τον απλούστερο τρόπο επικοινωνίας. Οι WiFi κόμβοι ακολουθούν ένα σύγχρονο μοντέλο αιτήματος-απάντησης (*request-response*) μέσω πρωτο-

κόλλου HTTP, επικοινωνώντας απευθείας με το REST API του Backend. Η ακολουθία επικοινωνίας αναλύεται στο Σχήμα 13.



Σχήμα 13: Sequence diagram αποκλειστικά για το HTTP μονοπάτι

Φάση Αυθεντικοποίησης.

Κατά την εκκίνηση, ο κόμβος αποστέλλει αίτημα `POST /api/v1/device/login` με το προ-κοινοποιημένο UUID και *secret*. Το Backend επαληθεύει τα διαπιστευτήρια μέσω σύγκρισης κατακερματισμένων τιμών (*hashed comparison*), εκδίδει ένα *Bearer token* μέσω Laravel Sanctum [40, 41], ενημερώνει την κατάσταση της συσκευής σε *ONLINE* και καταγράφει τη χρονοσφραγίδα τελευταίας σύνδεσης (*last_seen_at*). Το token αποθηκεύεται τοπικά στη μνήμη NVS (Non-Volatile Storage) του μικροελεγκτή και χρησιμοποιείται για κάθε επακόλουθο αίτημα μέσω της κεφαλίδας *Authorization*.

Φάση Μετάδοσης εδομένων.

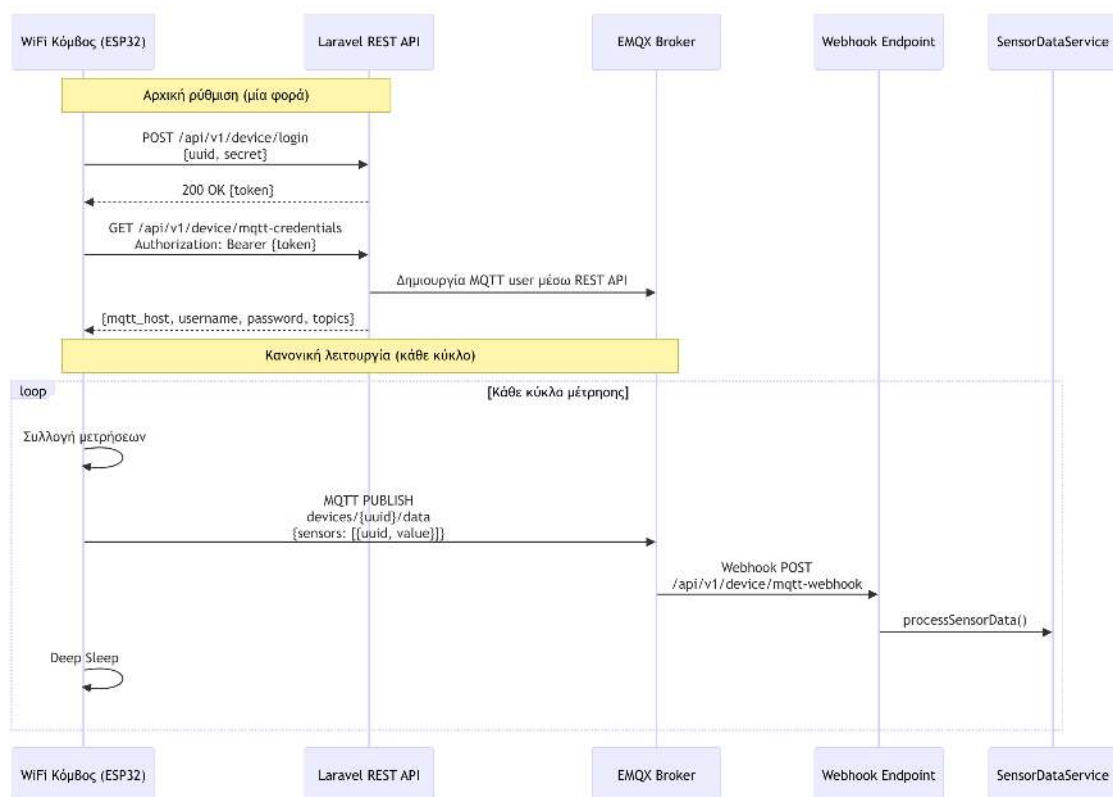
Σε κάθε κύκλο μέτρησης, ο κόμβος συλλέγει τιμές από τους αισθητήρες του και τις αποστέλλει συγκεντρωτικά μέσω αιτήματος `POST /api/v1/device/data`. Το σώμα του

αιτήματος ακολουθεί τη μορφή JSON και περιλαμβάνει έναν πίνακα μετρήσεων, όπου κάθε στοιχείο φέρει το UUID του αισθητήρα και την αντίστοιχη τιμή. Η διαδικασία αυτή επιτρέπει την αποστολή δεδομένων πολλαπλών αισθητήρων σε ένα μόνο αίτημα HTTP, μειώνοντας την κατανάλωση ενέργειας και το δικτυακό overhead.

Μετά τη λήψη, το Backend αντιστοιχίζει κάθε UUID αισθητήρα με την εγγραφή του στη βάση δεδομένων, καλεί την υπηρεσία *SensorDataService* για την επεξεργασία κάθε μέτρησης, και επιστρέφει αθροιστική απάντηση με τον αριθμό των εγγραφών που επεξεργάστηκαν επιτυχώς.

4.4.2 Μονοπάτι 2: WiFi → MQTT Broker → Πλατφόρμα

Το δεύτερο μονοπάτι αξιοποιεί το πρωτόκολλο MQTT [18] για ασύγχρονη επικοινωνία μέσω του μοντέλου δημοσίευσης–εγγραφής (*publish/subscribe*) [17]. Σε αυτή τη λειτουργία, οι WiFi κόμβοι — αντί να αποστέλλουν απευθείας HTTP αιτήματα στο Backend — δημοσιεύουν τα δεδομένα τους στον MQTT broker (EMQX), ο οποίος τα προωθεί στην πλατφόρμα μέσω μηχανισμού *webhook*. Η ακολουθία του μονοπατιού αυτού αποτυπώνεται στο Σχήμα 14.



Σχήμα 14: Sequence diagram για WiFi κόμβο μέσω MQTT

MQTT Broker (EMQX).

Ως κεντρικός μεσολαβητής μηνυμάτων χρησιμοποιείται ο EMQX v5.7.0, ένας broker υψηλής απόδοσης με υποστήριξη πρωτοκόλλου MQTT 5.0. Ο EMQX εκτελείται ως Docker container εντός της υποδομής του συστήματος και παρέχει πολλαπλές θύρες πρόσβασης: MQTT μέσω TCP (1883), MQTT μέσω TLS (8883), MQTT μέσω WebSocket (8083/8084), καθώς και κονσόλα διαχείρισης (18083).

Η ιεραρχία θεμάτων (*topic hierarchy*) για WiFi κόμβους ακολουθεί τη δομή `devices/{device_uuid}/`. Κάθε κόμβος δημοσιεύει τα δεδομένα αισθητήρων στο θέμα `devices/{uuid}/sensors`, το θέμα κατάστασης είναι `devices/{uuid}/status` και οι εντολές λαμβάνονται από `devices/{uuid}/`. Ο έλεγχος πρόσβασης (ACL — Access Control List) διασφαλίζει ότι κάθε συσκευή μπορεί να δημοσιεύσει και να εγγραφεί αποκλειστικά στα θέματα που αντιστοιχούν στο δικό της UUID, αποτρέποντας τη διασταυρούμενη πρόσβαση μεταξύ συσκευών.

Χορήγηση ιαπιστευτηρίων MQTT.

Τα διαπιστευτήρια πρόσβασης στον MQTT broker χορηγούνται δυναμικά. Μετά την επιτυχή αυθεντικοποίηση μέσω HTTP, η συσκευή αιτείται τα MQTT credentials μέσω του *endpoint* `GET /api/v1/device/mqtt-credentials`. Η υπηρεσία `MqttCredentialService` δημιουργεί έναν τυχαίο κωδικό 32 χαρακτήρων hex (παραγόμενο από 16 τυχαία bytes), τον καταχωρεί στον EMQX μέσω του REST API του broker, και αποθηκεύει τα credentials στο μοντέλο της συσκευής. Η απάντηση περιλαμβάνει τη διεύθυνση του broker, τη θύρα, το *username* (UUID συσκευής) και τον *password*, καθώς και τα θέματα στα οποία επιτρέπεται η δημοσίευση.

Μηχανισμός Webhook.

Η γεφύρωση μεταξύ EMQX και Laravel υλοποιείται μέσω του μηχανισμού *Rule Engine* του EMQX. Κάθε μήνυμα που δημοσιεύεται στα θέματα `devices/#` προωθείται αυτόματα ως HTTP POST αίτημα στο *endpoint* `/api/v1/device/mqtt-webhook`. Ο ελεγκτής παραλαμβάνει το μήνυμα, αντιστοιχίζει τη συσκευή μέσω του MQTT username, αποσειριοποιεί το ωφέλιμο φορτίο JSON, και τα προωθεί στο `SensorDataService` για επεξεργασία.

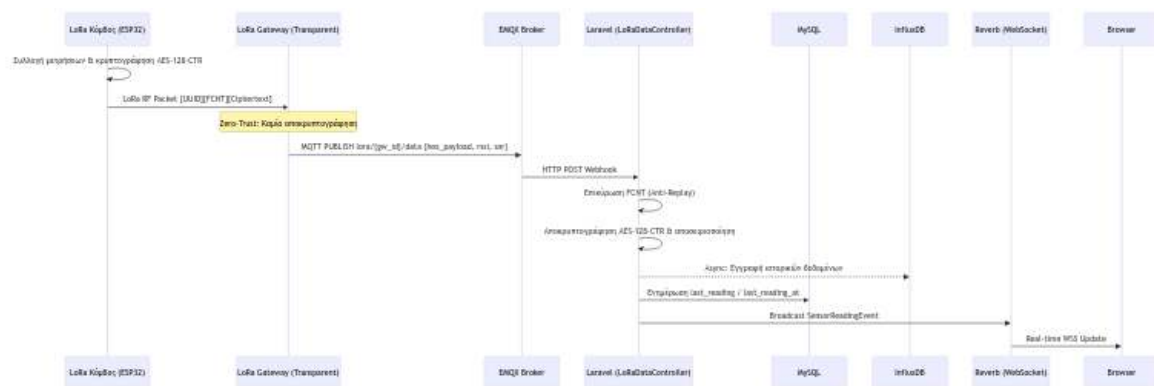
Σύγκριση με Μονοπάτι 1.

Σε σχέση με το HTTP, το MQTT διαφέρει σε δύο σημεία [42]: ο κόμβος δεν αναμένει απάντηση από το Backend προτού εισέλθει σε *Deep Sleep* (ασύγχρονο μοντέλο publish-and-forget), ενώ το *subscribe* επιτρέπει και αμφίδρομη επικοινωνία — η πλατφόρμα μπορεί να

στέλνει εντολές προς τον κόμβο (π.χ. ενεργοποίηση actuators) χωρίς polling.

4.4.3 Μονοπάτι 3: LoRa → Gateway → MQTT Broker → Πλατφόρμα

Το τρίτο μονοπάτι σχεδιάστηκε για περιοχές χωρίς πρόσβαση σε δίκτυο WiFi, αξιοποιώντας τη μεγάλη εμβέλεια του πρωτοκόλλου LoRa (έως αρκετά χιλιόμετρα) σε συνδυασμό με χαμηλή κατανάλωση ενέργειας [9, 43]. Η ροή δεδομένων διέρχεται από τρία στάδια: LoRa κόμβος → αυτόνομος Gateway → EMQX MQTT broker → Backend, όπως απεικονίζεται στο Σχήμα 15.



Σχήμα 15: Sequence diagram πλήρους ροής LoRa

LoRa Gateway.

Ο LoRa Gateway αποτελεί έναν αυτόνομο κόμβο ESP32 που διαθέτει ταυτόχρονα LoRa δέκτη και σύνδεση WiFi. Λειτουργεί ως μια απόλυτα διαφανής γέφυρα, καθώς δεν κατέχει κλειδί κρυπτογράφησης, δεν εξετάζει το περιεχόμενο των πακέτων και δεν εκτελεί καμία λογική επεξεργασίας. Ο ρόλος του περιορίζεται αποκλειστικά στη λήψη κρυπτογραφημένων LoRa πακέτων και την αναμετάδοσή τους αυτούσιων στον EMQX broker μέσω WiFi, στο θέμα `lora/{gateway_id}/data`. Η αποκρυπτογράφηση (AES-128-CTR) και η επικύρωση του Frame Counter εκτελούνται αποκλειστικά στο Backend από την υπηρεσία `LoRaCryptoService`.

Ιεραρχία Θεμάτων LoRa.

Για τα LoRa gateways χρησιμοποιείται ξεχωριστή ιεραρχία θεμάτων (`lora/{gateway_id}/#`), διακριτή από αυτή των WiFi συσκευών. Ο μηχανισμός *Rule Engine* του EMQX παρακολουθεί τα θέματα `lora/#` και προωθεί κάθε μήνυμα ως HTTP POST αίτημα στο `endpoint /api/v1/lora/webhook`. Ο ελεγκτής `LoRaDataController` παραλαμβάνει το μήνυμα, εξαγάγει το UUID της συσκευής, το *frame counter* και το κρυπτογραφημένο ωφέλιμο φορτίο,

και τα προωθεί στην υπηρεσία αποκρυπτογράφησης.

4.4.4 Πρωτόκολλο LoRa — Κρυπτογράφηση και Ακεραιότητα

Η επικοινωνία των LoRa κόμβων υλοποιεί κρυπτογράφηση στο επίπεδο εφαρμογής (*application-layer encryption*) με σκοπό την end-to-end προστασία των δεδομένων. Σε αντίθεση με το πρωτόκολλο LoRaWAN που βασίζεται σε δικτυακή υποδομή τρίτων [9], χρησιμοποιείται ιδιωτικό σχήμα κρυπτογράφησης βασισμένο σε AES-128-CTR.

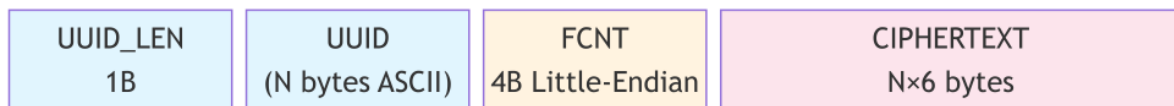
υαδική ομή Πακέτου.

Κάθε LoRa πακέτο ακολουθεί μια συμπαγή δυαδική δομή για τη μείωση του μεγέθους μετάδοσης, όπως παρουσιάζεται στον Πίνακα 3:

Πεδίο	Μέγεθος	Μορφή	Περιγραφή
UUID_LEN	1 byte	uint8	Μήκος UUID συσκευής
UUID	UUID_LEN bytes	ASCII	Αναγνωριστικό συσκευής
FCNT (Frame Counter)	4 bytes	uint32 LE	Μετρητής πλαισίου
CIPHERTEXT	$N \times 6$ bytes	binary	Κρυπτογραφημένες μετρήσεις

Πίνακας 3: Δυαδική δομή πακέτου LoRa

Η γραφική αναπαράσταση της δυαδικής δομής παρουσιάζεται στο Σχήμα 16.



Σχήμα 16: Block diagram δομής πακέτου

Κρυπτογράφηση AES-128-CTR.

Τα δεδομένα μετρήσεων κρυπτογραφούνται με τον αλγόριθμο AES-128 σε λειτουργία CTR (*Counter Mode*). Η λειτουργία CTR επιλέχθηκε διότι δεν απαιτεί *padding* — κρίσιμο πλεονέκτημα για πακέτα μεταβλητού μεγέθους σε IoT περιβάλλοντα. Το κλειδί κρυπτογράφησης (128-bit) είναι μοναδικό ανά συσκευή και αποθηκεύεται ως προ-κοινοποιημένο μυστικό κατά τη φάση εγγραφής. Η ακρίβεια κρυπτογράφησης (*nonce*) κατασκευάζεται ντετερμινιστικά από τρία στοιχεία: τον κατακερματισμό CRC32 του UUID της συσκευής (4 bytes), τον μετρητή πλαισίου (4 bytes) και σταθερή συμπλήρωση μηδενικών (8 bytes), που αποτελούν συνολικά 16 bytes.

Αποκρυπτογράφηση Μετρήσεων.

Μετά την αποκρυπτογράφηση, τα δεδομένα αποσειριοποιούνται σε μεμονωμένες μετρήσεις. Κάθε μέτρηση καταλαμβάνει 6 bytes: τα πρώτα 4 bytes αποτελούν το πρόθεμα ASCII του UUID αισθητήρα και τα υπόλοιπα 2 bytes μια ακέραια τιμή *int16* σε μορφή *little-endian*, κλιμακωμένη κατά 100 (π.χ. η τιμή 2550 αντιστοιχεί σε 25,50 °C). Τα προθέματα αντιστοιχίζονται με τα UUID των αισθητήρων που έχουν καταχωρηθεί στη βάση δεδομένων.

Προστασία Επανάληψης.

Κάθε πακέτο περιλαμβάνει έναν αύξοντα μετρητή πλαισίου (*frame counter*, FCNT), ο οποίος χρησιμοποιείται ως μηχανισμός προστασίας έναντι επιθέσεων επανάληψης (*replay attacks*). Η υπηρεσία LoRaCryptoService εφαρμόζει δύο ελέγχους:

1. **Έλεγχος επανάληψης:** Εάν ο εισερχόμενος FCNT είναι ίσος ή μικρότερος του αποθηκευμένου, το πακέτο απορρίπτεται ως πιθανή επίθεση επανάληψης (κωδικός HTTP 409).
2. **Έλεγχος κενού:** Εάν η διαφορά μεταξύ εισερχόμενου και αποθηκευμένου FCNT υπερβαίνει ένα ρυθμιζόμενο κατώφλι (10.000), το πακέτο σημαίνεται ως ύποπτο, καθώς ενδέχεται η συσκευή να έχει επαναφερθεί σε εργοστασιακές ρυθμίσεις ή να υποβάλλεται σε παραποίηση (κωδικός HTTP 422).

Το κατώφλι των 10.000 επιλέχθηκε εμπειρικά ώστε να απορροφά συνήθεις απώλειες πακέτων σε lossy LoRa περιβάλλοντα (όπου ένας κόμβος μπορεί να μεταδίδει κανονικά για μεγάλο χρονικό διάστημα χωρίς λήψη στο gateway), χωρίς όμως να υποστηρίζεται από πειραματική αξιολόγηση για τις συγκεκριμένες συνθήκες ανάπτυξης. Η βαθμονόμηση αυτής της τιμής για διαφορετικές πυκνότητες κόμβων και αποστάσεις αποτελεί αντικείμενο μελλοντικής εργασίας.

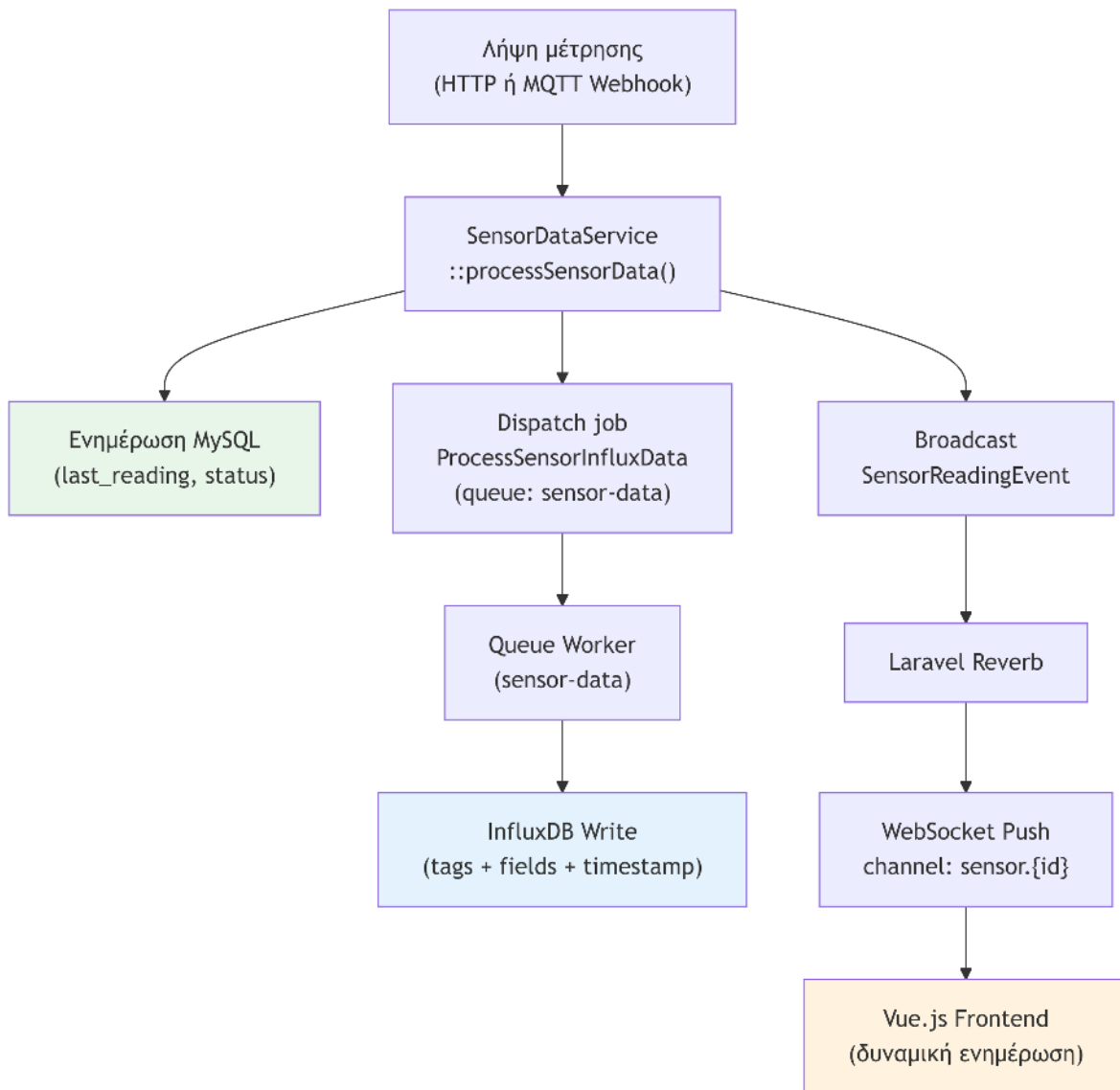
Μετά την επιτυχή επαλήθευση, ο νέος FCNT αποθηκεύεται στο μοντέλο της συσκευής, εξασφαλίζοντας αυστηρά αύξουσα ακολουθία.

4.4.5 Ασύγχρονη Επεξεργασία και Ενημέρωση Πραγματικού Χρόνου

Ανεξαρτήτως μονοπατιού προέλευσης (HTTP ή MQTT/webhook), όλες οι μετρήσεις καταλήγουν στην κεντρική υπηρεσία SensorDataService, η οποία ενορχηστρώνει τρεις διαδικασίες ενέργειες:

1. **Αποθήκευση χρονοσειράς (ασύγχρονη):** Δημιουργείται εργασία (*job*) `ProcessSensorInfluxData` στην ουρά `sensor-data`, η οποία εκτελείται από εξειδικευμένο *queue worker*. Η εργασία εγγράφει τη μέτρηση στην InfluxDB (η αναλυτική δομή δεδομένων παρουσιάζεται στην Ενότητα 4.5.2).
2. **Ενημέρωση κατάστασης (σύγχρονη):** Τα πεδία `last_reading` και `last_reading_at` του αισθητήρα ενημερώνονται στη MySQL, παρέχοντας πρόσβαση στην τελευταία τιμή χωρίς αναζήτηση στην InfluxDB.
3. **Ειδοποίηση πραγματικού χρόνου (ασύγχρονη):** Εκπέμπεται συμβάν `SensorReadingEvent` μέσω του *broadcast driver*, το οποίο προωθείται στον Reverb WebSocket server και αποστέλλεται στα εγγεγραμμένα *clients* μέσω ιδιωτικού καναλιού `sensor.{sensorId}`.

Η ροή των τριών ενεργειών επεξεργασίας απεικονίζεται στο flowchart του Σχήματος 17.



Σχήμα 17: Flowchart επεξεργασίας στο SensorDataService

WebSocket Κανάλια.

Ο WebSocket server (Laravel Reverb) λειτουργεί ως αυτόνομο container εντός της υποδομής Docker. Τα κανάλια επικοινωνίας ορίζονται ως ιδιωτικά (*private channels*), απαιτώντας αυθεντικοποίηση χρήστη. Ο κανόνας εξουσιοδότησης για το κανάλι `sensor.{sensorId}` επαληθεύει ότι ο αισθητήρας ανήκει στον αυθεντικοποιημένο χρήστη, αποτρέποντας τη λήψη ειδοποιήσεων από αισθητήρες τρίτων. Με αυτόν τον τρόπο, η αρχή της πολυμίσθωσης εφαρμόζεται και στο επίπεδο των WebSockets.

Ουρά Εργασιών Αισθητήρων.

Η αρχιτεκτονική χρησιμοποιεί εξειδικευμένο *queue worker* αποκλειστικά για την ουρά *sensor-data*, διαφοροποιημένο από τον γενικό *worker* της εφαρμογής. Ο διαχωρισμός αυτός εξασφαλίζει ότι ο μεγάλος όγκος δεδομένων αισθητήρων δεν μπλοκάρει εργασίες γενικού σκοπού (π.χ. αποστολή email, ειδοποιήσεις), ενώ επιτρέπει την ανεξάρτητη κλιμάκωση μέσω αύξησης των αντίστοιχων *containers*.

4.5 Αρχιτεκτονική Αποθήκευσης Δεδομένων

Η πλατφόρμα *Agronos* υιοθετεί μια αρχιτεκτονική διττής βάσης δεδομένων (*polyglot persistence*), χρησιμοποιώντας δύο εξειδικευμένα συστήματα αποθήκευσης: μια σχεσιακή βάση δεδομένων (MySQL 8) για τα δομημένα δεδομένα εφαρμογής, και μια βάση χρονοσειρών (InfluxDB) για τις μετρήσεις αισθητήρων. Η επιλογή αυτή αντανακλά τις διαφορετικές απαιτήσεις μεταξύ μεταδεδομένων (χαμηλός ρυθμός εγγραφής, σύνθετες σχέσεις) και δεδομένων τηλεμετρίας (υψηλός ρυθμός εγγραφής, χρονικά ερωτήματα εύρους) [22, 21].

4.5.1 Σχεσιακή Βάση Δεδομένων

Η MySQL αποθηκεύει τα δομημένα δεδομένα της εφαρμογής — χρήστες, αγροτεμάχια, συσκευές, αισθητήρες και *tokens* αυθεντικοποίησης — εξασφαλίζοντας ακεραιότητα αναφορών μέσω ξένων κλειδιών (*foreign keys*).

Σχεσιακό Σχήμα.

Η βάση δεδομένων αποτελείται από τους ακόλουθους πίνακες, των οποίων η δομή συνοψίζεται στον Πίνακα 4:

Πίνακας	Σκοπός	Βασικά Πεδία
users	Λογαριασμοί χρηστών	id, name, email, password
farms	Αγροτεμάχια	user_id, name, coordinates (JSON)
devices	Συσκευές IoT	user_id, uuid, secret, type, status, lora_aes_key, lora_frame_counter, signal_strength, battery_level
sensors	Αισθητήρες	device_id, farm_id, uuid, type, lat, lon
personal_access_tokens	Sanctum tokens	Αυθεντικοποίηση συσκευών/χρηστών
jobs	Ουρά εργασιών	Ασύγχρονη επεξεργασία

Πίνακας 4: Πίνακες σχεσιακής βάσης δεδομένων

Αξιοσημείωτες Σχεδιαστικές Επιλογές.

Στη σχεσιακή βάση δεδομένων υπάρχουν δύο αξιοσημείωτες σχεδιαστικές επιλογές:

Γεωχωρικά δεδομένα ως JSON. Τα γεωγραφικά όρια κάθε αγροτεμαχίου αποθηκεύονται στο πεδίο `coordinates` σε μορφή GeoJSON Polygon. Αυτή η επιλογή επιτρέπει την αποθήκευση πολύπλοκων πολυγώνων χωρίς εξάρτηση από γεωχωρικές επεκτάσεις βάσης δεδομένων, ενώ το μοντέλο Farm υπολογίζει δυναμικά το κέντρο και την έκταση του πολυγώνου. Εναλλακτικά, η επέκταση PostGIS (PostgreSQL) θα παρείχε εγγενή χωρική δεικτοδότηση και βελτιωμένη απόδοση σε ερωτήματα γεωγραφικής αναζήτησης μεγάλης κλίμακας. Ωστόσο, κρίθηκε ότι τα πλεονεκτήματα αυτά δεν δικαιολογούνταν για τον αναμενόμενο αριθμό αγροτεμαχίων, ενώ η αποθήκευση σε απλή στήλη JSON εξασφαλίζει φορητότητα (portability) — πλεονέκτημα κατά τις δοκιμές με SQLite. Επιπλέον, η μορφή JSON παρέχει ευελιξία σχήματος (*schema flexibility*): μελλοντικές επεκτάσεις του μοντέλου γεωδεδομένων — όπως η προσθήκη πεδίων υψομέτρου, περιγραφικών ιδιοτήτων ανά ζώνη ή ενσωμάτωση αντικειμένου GeoJSON Feature αντί απλού Polygon — μπορούν να εφαρμοστούν χωρίς αλλαγή τύπου στήλης ή migration σχήματος, κάτι που ένας δεσμευτικός τύπος GEOMETRY του PostGIS δεν θα επέτρεπε εξίσου εύκολα.

Στιγμιότυπο τελευταίας μέτρησης. Κάθε αισθητήρας διατηρεί τα πεδία `last_reading` και `last_reading_at` στη MySQL. Αν και η πλήρης ιστορία αποθηκεύεται στην InfluxDB, η τελευταία τιμή αποθηκεύεται πλεονασματικά στη σχεσιακή βάση ώστε η διεπαφή χρήστη να εμφανίζει τρέχουσες τιμές χωρίς ερώτημα στην InfluxDB — μειώνοντας τη λανθάνουσα καθυστέρηση κατά τη φόρτωση σελίδων.

4.5.2 Βάση Χρονοσειρών

Η InfluxDB αναλαμβάνει αποκλειστικά την αποθήκευση του ιστορικού μετρήσεων, παρέχοντας υψηλή απόδοση εγγραφής, ενσωματωμένες συναρτήσεις συγκέντρωσης και αποδοτική συμπίεση χρονοσειρών [23, 21].

Μοντέλο εδομένων.

Όλες οι μετρήσεις αποθηκεύονται σε ένα ενιαίο *measurement* με όνομα `sensor_measurement`, βελτιστοποιημένο τόσο για εγγραφή όσο και για αναζήτηση, όπως φαίνεται στον Πίνακα 5:

Τα *tags* δεικτοδοτούνται αυτόματα (*indexed*) από την InfluxDB, κάτι που εξυπηρετεί ταυτόχρονα δύο σκοπούς: επιτρέπει ταχεία αναζήτηση και εφαρμόζει απομόνωση μι-

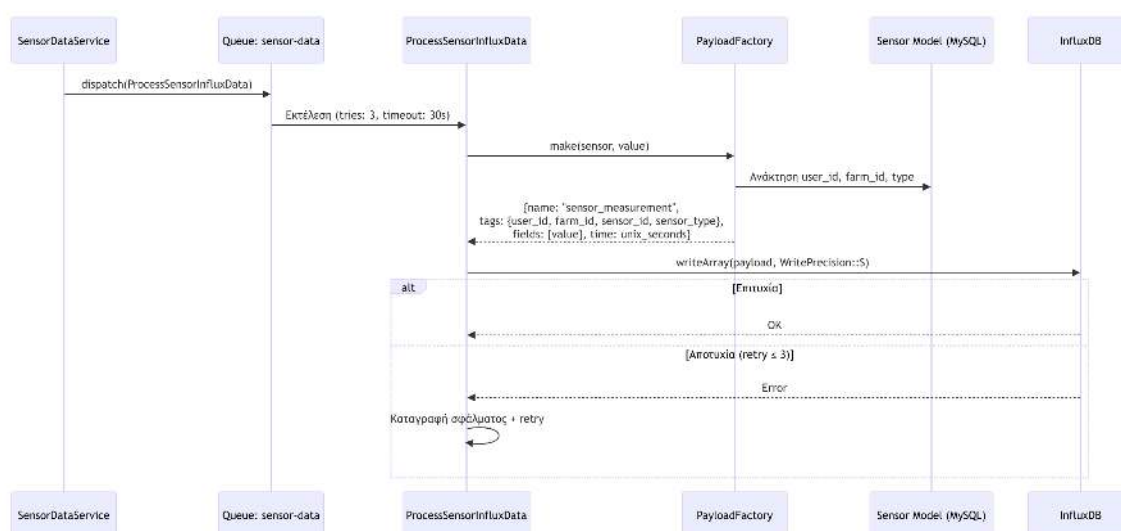
Κατηγορία	Πεδίο	Τύπος	Ρόλος
Tag	user_id	integer	Απομόνωση μισθωτή
Tag	farm_id	integer	Ομαδοποίηση ανά αγρό
Tag	sensor_id	string	Ταυτοποίηση αισθητήρα
Tag	sensor_type	string	Τύπος μέτρησης
Field	value	float	Τιμή μέτρησης
Timestamp	—	Unix seconds	Χρόνος μέτρησης

Πίνακας 5: Δομή δεδομένων InfluxDB

σθωτών μέσω του `user_id` — κάθε ερώτημα φιλτράρεται αυτόματα στις μετρήσεις του τρέχοντος χρήστη.

Μοτίβα Εγγραφής.

Όπως περιγράφηκε στην Ενότητα 4.4, η εγγραφή δεδομένων εκτελείται ασύγχρονα μέσω της εργασίας `ProcessSensorInfluxData`. Η εργασία κατασκευάζει το ωφέλιμο φορτίο μέσω του `SensorMeasurementPayloadFactory`, ο οποίος αντλεί τις ετικέτες (*tags*) από τα σχεσιακά μοντέλα και τις συνδυάζει με την τιμή μέτρησης. Η ακρίβεια χρονοσήμανσης (*timestamp precision*) ορίζεται σε δευτερόλεπτα, επαρκής για τον ρυθμό δειγματοληψίας γεωργικών αισθητήρων. Η ακολουθία της διαδικασίας εγγραφής απεικονίζεται στο Σχήμα 18.



Σχήμα 18: Sequence diagram εγγραφής μέτρησης

Μοτίβα Ανάγνωσης και Συγκέντρωσης.

Η ανάκτηση δεδομένων χρονοσειρών πραγματοποιείται μέσω της γλώσσας ερωτημάτων Flux, με δύο βασικά μοτίβα:

- **Πρόσφατες μετρήσεις:** Ανάκτηση των τελευταίων N μετρήσεων ενός αισθητήρα, ταξινομημένων φθίνουσα κατά χρόνο. Χρησιμοποιείται για πίνακες δεδομένων στη διεπαφή χρήστη.
- **Γραφήματα με συγκέντρωση:** Ανάκτηση μετρήσεων εντός χρονικού εύρους, ομαδοποιημένων σε παράθυρα (*aggregateWindow*) με υπολογισμό μέσου όρου. Το μέγεθος του παραθύρου προσαρμόζεται αυτόματα ανάλογα με το χρονικό εύρος αναζήτησης.

Τα παράθυρα συγκέντρωσης ανά χρονικό εύρος παρουσιάζονται στον Πίνακα 6.

Χρονικό εύρος	Παράθυρο συγκέντρωσης	Σημεία δεδομένων (μέγ.)
1 ώρα	1 λεπτό	60
24 ώρες	15 λεπτά	96
7 ημέρες	1 ώρα	168
30 ημέρες	6 ώρες	120
90 ημέρες	12 ώρες	180

Πίνακας 6: Παράθυρα συγκέντρωσης ανά χρονικό εύρος

Ο μηχανισμός αυτός εξασφαλίζει ότι τα γραφήματα εμφανίζουν πάντα εύλογο αριθμό σημείων δεδομένων (60–180), ανεξαρτήτως του χρονικού εύρους αναζήτησης, αποφεύγοντας τόσο την υπερφόρτωση της διεπαφής όσο και την υπερφόρτωση της βάσης δεδομένων.

Ερωτήματα σε Επίπεδο Αγρού.

Πέρα από μεμονωμένους αισθητήρες, η υπηρεσία *FarmTimeSeriesService* εκτελεί ερωτήματα συγκέντρωσης σε επίπεδο αγροτεμαχίου: ομαδοποιεί όλους τους αισθητήρες ενός αγρού ανά τύπο (*sensor_type*) και υπολογίζει ελάχιστη, μέγιστη και μέση τιμή. Τα αποτελέσματα χρησιμοποιούνται στο ταμπλό του αγρού (*farm dashboard*) για τη συνοπτική απεικόνιση της κατάστασης.

4.5.3 Αιτιολόγηση Διττής Αρχιτεκτονικής

Η επιλογή χρήσης δύο διακριτών βάσεων δεδομένων, αντί μίας ενιαίας, δικαιολογείται από τις θεμελιωδώς διαφορετικές απαιτήσεις κάθε κατηγορίας δεδομένων, όπως αποτυπώνεται συγκριτικά στον Πίνακα 7:

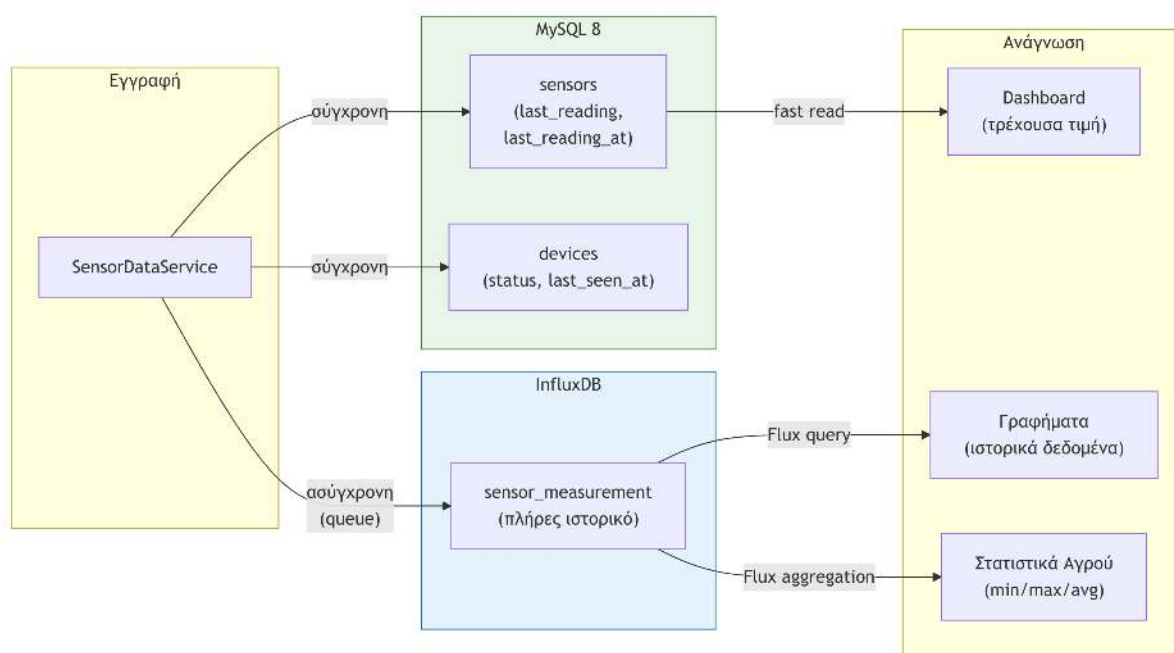
Η MySQL υπερέχει στη διαχείριση σύνθετων σχέσεων (π.χ. $User \rightarrow Farm \rightarrow Sensor$) και στις συναλλαγές ACID, αλλά αδυνατεί να ανταπεξέλθει αποδοτικά στην αποθήκευση εκατομμυρίων μετρήσεων χρονοσειρών, καθώς οι ευρετηριασμοί B-tree δεν βελτιστο-

Χαρακτηριστικό	MySQL	InfluxDB
Ρυθμός εγγραφής	Χαμηλός	Υψηλός
Μοντέλο δεδομένων	Σχεσιακό (JOINS)	Χρονοσειρές
Συναλλαγές ACID	Ναι	Όχι
Συμπίεση	Περιορισμένη	Υψηλή
Συγκέντρωση	Χειρωνακτική	Ενσωματωμένη
Διατήρηση δεδομένων	Χειρωνακτική	Αυτόματη (TTL)
Χρονικά ερωτήματα	$O(n \log n)$	$O(\log n)$

Πίνακας 7: Σύγκριση MySQL και InfluxDB

ποιούνται για χρονικά ερωτήματα εύρους [21]. Αντίστροφα, η InfluxDB προσφέρει υψηλή απόδοση για χρονικά ερωτήματα και ενσωματωμένες συναρτήσεις συγκέντρωσης (aggregateWindow, mean, min, max) [23], αλλά δεν υποστηρίζει σχεσιακές λειτουργίες (JOIN) ούτε συναλλαγές.

Ο συνδυασμός εκμεταλλεύεται τα πλεονεκτήματα κάθε συστήματος: η MySQL παρέχει τα μεταδεδομένα και το πλαίσιο (ποιος αισθητήρας, σε ποιον αγρό, ποιου χρήστη), ενώ η InfluxDB παρέχει την ιστορική απόδοση (τι μέτρησε, πότε, με ποιο ρυθμό μεταβολής). Το Σχήμα 19 απεικονίζει τον τρόπο συνεργασίας των δύο βάσεων.



Σχήμα 19: Πώς συνεργάζονται οι δύο βάσεις

4.6 Σύνοψη

Η αρχιτεκτονική του *Agronos* βασίζεται σε τρεις κεντρικές επιλογές: η πολυμίσθωση υλοποιείται στο επίπεδο ORM μέσω *Global Scopes* αντί ξεχωριστών βάσεων, η κρυπτογράφηση LoRa γίνεται end-to-end στον κόμβο και στο Backend (όχι στο Gateway), και η αποθήκευση χωρίζεται μεταξύ MySQL και InfluxDB ανάλογα με τη φύση των δεδομένων. Τα επόμενα κεφάλαια παρουσιάζουν πώς αυτές οι επιλογές υλοποιούνται στον κώδικα του Backend και του Firmware.

5 Σχεδιασμός και Ανάπτυξη του Κεντρικού Συστήματος Διαχείρισης (Backend) και της Διαδικτυακής Εφαρμογής

Ο πλήρης πηγαίος κώδικας του backend είναι δημοσίως διαθέσιμος στο αποθετήριο GitHub: <https://github.com/gkalogeitonas/agronos>.

Το παρόν κεφάλαιο παρουσιάζει την υλοποίηση του κεντρικού συστήματος διαχείρισης και της διαδικτυακής εφαρμογής. Αρχικά περιγράφονται οι επιλογές τεχνολογίας και η οργάνωση κώδικα, ακολουθεί η ανάλυση του επιπέδου δεδομένων (Eloquent μοντέλα, enums, υβριδική αποθήκευση MySQL + InfluxDB) και το επίπεδο εφαρμογής με τους CRUD controllers και τις σελίδες Vue.js. Στη συνέχεια παρουσιάζεται το REST API των IoT συσκευών, η κεντρική υπηρεσία επεξεργασίας δεδομένων, η επικοινωνία πραγματικού χρόνου μέσω WebSocket και, τέλος, οι μηχανισμοί ασφαλείας της εφαρμογής.

5.1 Οργάνωση Κώδικα και Εργαλεία Ανάπτυξης

Η ανάπτυξη του συστήματος Backend της πλατφόρμας Agronos βασίστηκε σε σύγχρονες αρχές μηχανικής λογισμικού, με κύριο άξονα τον πλήρη διαχωρισμό των ευθυνών μεταξύ της διεπαφής χρήστη και της διαχείρισης των δεδομένων.

5.1.1 Αρχιτεκτονική Βασισμένη σε API

Η σύγχρονη ανάπτυξη διαδικτυακών εφαρμογών χαρακτηρίζεται από μια ευρεία μετάβαση σε αρχιτεκτονικές που βασίζονται σε API, οι οποίες διαχωρίζουν πλήρως τις λειτουργίες του frontend από εκείνες του backend. Αυτός ο διαχωρισμός επιτρέπει την ανάπτυξη συστημάτων που είναι αρθρωτά, κλιμακώσιμα και ιδιαίτερα εύκολα στη συντήρηση [40].

5.1.2 Η Στρατηγική Επιλογή του Laravel

Για την υλοποίηση αυτού του επιπέδου, επιλέχθηκε το πλαίσιο Laravel (έκδοση 12). Το Laravel αναγνωρίζεται σήμερα ως ένα από τα κορυφαία προγραμματιστικά εργαλεία για την ανάπτυξη διαδικτυακών εφαρμογών σε PHP, διακρινόμενο για την κομψή και εκφραστική του σύνταξη. Η επιλογή αυτή κρίθηκε κατάλληλη για έργα της κλίμακας του Agronos, καθώς έρευνες υιοθέτησής του σε Μικρομεσαίες Επιχειρήσεις και συστήματα με περιορισμένους πόρους καταδεικνύουν ότι συμβάλλει στη μείωση κινδύνων και κόστους κατά τον κύκλο ζωής της ανάπτυξης [44]. Αξίζει να σημειωθεί ότι εναλλακτικά πλαίσια — όπως το Symfony για απαιτητικά εταιρικά περιβάλλοντα ή ασύγχρονες λύσεις βασισμένες σε

Node.js για εξαιρετικά υψηλό I/O throughput — θα αποτελούσαν τεχνικά έγκυρες εναλλακτικές. Ωστόσο, αποκλείστηκαν στη συγκεκριμένη περίπτωση, καθώς οι απαιτήσεις κλιμάκωσης του Agropos εμπίπτουν στα δυνατά σημεία του Laravel, ενώ η υφιστάμενη εμπειρία του συγγραφέα με το framework επέτρεψε την εστίαση στα ερευνητικά ζητήματα αντί στην εκμάθηση νέων εργαλείων.

5.1.3 Τεχνικά Χαρακτηριστικά και Υλοποίηση

Σε τεχνικό επίπεδο, το Laravel υλοποιεί αυστηρά το πρότυπο αρχιτεκτονικής Model-View-Controller (MVC), παρέχοντας καθαρό διαχωρισμό μεταξύ της επιχειρησιακής λογικής της πλατφόρμας και των δεδομένων, γεγονός που καθιστά τον κώδικα περισσότερο δομημένο. Παράλληλα, η ενσωμάτωση του Eloquent ORM (Object-Relational Mapping) αυτοματοποιεί και απλοποιεί τη διαχείριση της σχεσιακής βάσης δεδομένων (MySQL). Ο μηχανισμός αυτός εξαλείφει την ανάγκη χειροκίνητης συγγραφής πολύπλοκων ερωτημάτων SQL, επιτρέποντας την ανάπτυξη ενός REST API που είναι εγγενώς αρθρωτό και κλιμακώσιμο [40].

Τέλος, όσον αφορά την ασφάλεια του συστήματος, το framework παρέχει ενσωματωμένες, προηγμένες λειτουργίες για κρυπτογράφηση, διαχείριση κωδικών πρόσβασης και αυστηρή επικύρωση (validation) των εισερχόμενων δεδομένων. Οι μηχανισμοί αυτοί ενισχύθηκαν περαιτέρω με τη χρήση του πακέτου Laravel Sanctum, το οποίο παρέχει στρατηγικές αυθεντικοποίησης βασισμένες σε tokens, αποτελώντας τον βασικό πυλώνα ασφαλείας τόσο για την πιστοποίηση των IoT συσκευών όσο και για την προστασία του SPA frontend [41].

5.1.4 Ανάπτυξη Διεπαφής Χρήστη και Ενοποίηση

Το επίπεδο Εφαρμογής και Παρουσίασης της πλατφόρμας υλοποιήθηκε μέσω του πλαισίου Vue.js (έκδοση 3), δημιουργώντας μια Εφαρμογή Μονοσέλιδου (SPA). Η επιλογή του Vue 3, και συγκεκριμένα της αρχιτεκτονικής του Composition API, επιτρέπει τη δημιουργία εξαιρετικά επαναχρησιμοποιήσιμων και αρθρωτών τμημάτων κώδικα (components). Το σύστημα δυναμικής αντίδρασης (reactivity) του πλαισίου εξυπηρετεί την ανάπτυξη σύγχρονων πινάκων ελέγχου IoT, καθώς διασφαλίζει ότι η διεπαφή του τελικού χρήστη ενημερώνεται ακαριαία κατά την έλευση νέων τηλεμετρικών δεδομένων, προσφέροντας οπτικοποίηση σε πραγματικό χρόνο. Παρόμοιες δυνατότητες παρέχουν και εναλλακτικά πλαίσια, όπως το React που διαθέτει ευρύτερο οικοσύστημα, ή το Svelte που προσφέρει μικρότερο bundle size. Η επιλογή του Vue 3 υπαγορεύτηκε τόσο από την ομαλή ενσωμάτωσή

του με το Inertia.js στο οικοσύστημα του Laravel, όσο και από την υφιστάμενη εξοικείωση του συγγραφέα με αυτό.

Ωστόσο, η παραδοσιακή προσέγγιση ανάπτυξης ενός αυτόνομου SPA συχνά εισάγει σημαντική πολυπλοκότητα. Συνήθως απαιτεί τη διαχείριση ενός ξεχωριστού συστήματος δρομολόγησης στον πελάτη, τη διαχείριση κατάσταση και την ανάπτυξη ενός δευτερεύοντος API, αποκλειστικά για την τροφοδοσία του γραφικού περιβάλλοντος. Για την επίλυση αυτού του αρχιτεκτονικού προβλήματος, ενσωματώθηκε η τεχνολογία Inertia.js (έκδοση 2) ως η γέφυρα επικοινωνίας μεταξύ του συστήματος Laravel και του Vue.js. Το Inertia.js τροποποιεί σημαντικά το πρότυπο επικοινωνίας, καθώς περιορίζει την ανάγκη για ξεχωριστό client-side router και επιτρέπει στο Laravel να διατηρήσει τον πλήρη έλεγχο της δρομολόγησης (server-driven routing). Πρακτικά, αυτό σημαίνει ότι το σύνολο των διαδρομών της εφαρμογής ορίζεται σε ένα μόνο σημείο — το αρχείο `routes/web.php` του Laravel — εξαλείφοντας τον κίνδυνο ασυνέπειας μεταξύ server-side και client-side routing και μειώνοντας σημαντικά τη διαχειριστική επιβάρυνση. Ο περιορισμός αυτής της προσέγγισης έγκειται στη μειωμένη ευελιξία για αμιγώς client-side μεταβάσεις χωρίς επικοινωνία με τον εξυπηρετητή, λειτουργία που θα απαιτούσε πλήρες αυτόνομο SPA. Για τις ανάγκες της πλατφόρμας Agronos, ο περιορισμός αυτός κρίθηκε αποδεκτός.

5.2 Επίπεδο Δεδομένων

Το επίπεδο δεδομένων της πλατφόρμας Agronos βασίζεται στο Eloquent ORM του Laravel, το οποίο αντιστοιχίζει κάθε πίνακα της σχεσιακής βάσης σε μία κλάση PHP (Model). Η ενότητα αυτή περιγράφει τα τρία βασικά μοντέλα, τα Enums που ορίζουν τις επιτρεπτές τιμές κατάστασης και τύπου, καθώς και τη στρατηγική υβριδικής αποθήκευσης σε MySQL και InfluxDB.

5.2.1 Eloquent Μοντέλα και Σχέσεις

Οι τρεις βασικές οντότητες του συστήματος — Farm, Device και Sensor — αντιστοιχούν σε ισάριθμα Eloquent μοντέλα. Κάθε μοντέλο ενσωματώνει το trait `BelongsToTenant`, υλοποιώντας αυτόματα την απομόνωση δεδομένων πολυμίσθωσης που περιγράφηκε στην ενότητα 4.3.

Farm — Αγρός.

Αναπαριστά ένα αγροτεμάχιο του χρήστη. Τα βασικά πεδία περιλαμβάνουν το όνομα, την τοποθεσία, το μέγεθος σε τετραγωνικά μέτρα, μια προαιρετική περιγραφή και τις γεωγρα-

φικές συντεταγμένες σε μορφή GeoJSON polygon. Οι συντεταγμένες αποθηκεύονται ως JSON στη βάση και μετατρέπονται μέσω του μηχανισμού cast.

```
1 class Farm extends Model
2 {
3     use BelongsToTenant, HasFactory;
4
5     protected $casts = [
6         'coordinates' => 'array',
7     ];
8
9     protected $appends = ['center'];
10
11     public function user(): BelongsTo {...}
12
13     public function getAreaAttribute(){...}
14
15     public function getCenterAttribute(){...}
16
17     public function sensors(){ ... }
18 }
```

Listing 1: Σύντομη προδιαγραφή του μοντέλου Farm

Μέθοδοι του μοντέλου Farm.

Οι βοηθητικές μέθοδοι του μοντέλου υποστηρίζουν λειτουργίες γεωχωρικών υπολογισμών και σχέσεων:

- **getAreaAttribute():** Υπολογίζει το εμβαδόν (σε τετραγωνικά μέτρα) του πολυγώνου που βρίσκεται στο `coordinates` (GeoJSON). Η υλοποίηση χρησιμοποιεί τον αλγόριθμο Shoelace/γεωμετρικό άθροισμα και επιστρέφει αριθμό κινητής υποδιαστολής.
- **getCenterAttribute():** Επιστρέφει το centroid του πολυγώνου ως πίνακα [`'lat'` => ..., `'lng'` => ...] (ή `null` αν δεν υπάρχουν συντεταγμένες). Το πεδίο `center` προστίθεται αυτόματα στις σειριαλοποιήσεις μέσω του `$appends`.

- `sensors()`: Σχέση `hasMany` προς το `Sensor`, που επιτρέπει ανάκτηση, φιλτράρισμα και eager loading των αισθητήρων του αγρού.
- `user()`: Σχέση `belongsTo` με τον `User` — συνδέει τον αγρό με τον ιδιοκτήτη του.

Οι μέθοδοι που δουλεύουν με γεωγραφικά δεδομένα βασίζονται στο `coordinates` που έχει δηλωθεί στο `$casts` ως `array`, επομένως οι υπολογισμοί γίνονται σε PHP πίνακες.

Η σχέση `hasMany` με τον `Sensor` επιτρέπει την ανάκτηση όλων των αισθητήρων που ανήκουν σε ένα αγροτεμάχιο.

Ο πλήρης ορισμός του μοντέλου παρατίθεται στο Παράρτημα Α' (Listing 69).

Device — Συσκευή.

Η συσκευή είναι ο κόμβος επικοινωνίας στο πεδίο. Σε αντίθεση με τα υπόλοιπα μοντέλα, η κλάση `Device` κληρονομεί από `Authenticatable` αντί για `Model` και χρησιμοποιεί το `trait HasApiTokens`, καθώς οι συσκευές αυθεντικοποιούνται ανεξάρτητα μέσω `Laravel Sanctum`:

Τα πεδία περιλαμβάνουν το μοναδικό αναγνωριστικό (`uuid`), το μυστικό κλειδί (`secret`), τον τύπο (`type`), την κατάσταση (`status`), καθώς και πεδία για LoRa: τον μετρητή πλαισίου (`lora_frame_counter`) και το κλειδί κρυπτογράφησης (`lora_aes_key`). Τα ευαίσθητα πεδία `secret` και `lora_aes_key` δηλώνονται στον πίνακα `$hidden`, ώστε να μην συμπεριλαμβάνονται σε JSON αποκρίσεις.

```
1 class Device extends Authenticatable
2 {
3     use BelongsToTenant, HasApiTokens, HasFactory;
4
5     protected $hidden = [
6         'secret',
7         'lora_aes_key',
8     ];
9
10    protected $casts = [
11        'status' => DeviceStatus::class,
12    ];
13
14    public function user(): BelongsTo {...}
```

```
15
16     public function sensors() { ... }
17
18     public function isLoRa(): bool {...}
19 }
```

Listing 2: Σύντομη προδιαγραφή του μοντέλου Device

Μέθοδοι του μοντέλου Device.

Η κλάση Device επεκτείνει τον μηχανισμό αυθεντικοποίησης (Authenticatable) και, με το HasApiTokens, παρέχει τη δυνατότητα έκδοσης API tokens μέσω Laravel Sanctum. Σημαντικά σημεία:

- Ευαίσθητα πεδία: `secret` και `lora_aes_key` δηλώνονται στο `$hidden` για να αποκλείονται από σειριαλοποιήσεις.
- `user()`: Σχέση `belongsTo` με τον ιδιοκτήτη (χρήστη).
- `sensors()`: Σχέση `hasMany` που επιστρέφει τους αισθητήρες της συσκευής και υποστηρίζει `eager loading`.
- `isLoRa()`: Βοηθητική μέθοδος που ελέγχει αν ο τύπος της συσκευής είναι LoRa.

Ο πλήρης ορισμός του μοντέλου παρατίθεται στο Παράρτημα Α' (Listing 70).

Sensor — Αισθητήρας.

Αναπαριστά μια μονάδα λήψης δεδομένων. Κάθε αισθητήρας συνδέεται ταυτόχρονα με μία συσκευή (`belongsTo Device`) και έναν αγρό (`belongsTo Farm`), υλοποιώντας τη σχέση που περιγράφηκε στο κεφάλαιο αρχιτεκτονικής. Τα πεδία `lat` και `lon` αποθηκεύουν τη γεωγραφική θέση του αισθητήρα (και όχι της συσκευής), ενώ τα πεδία `last_reading` και `last_reading_at` λειτουργούν ως τοπική cache της πιο πρόσφατης μέτρησης, αποφεύγοντας κλήσεις στην InfluxDB για απλή εμφάνιση τρέχουσας τιμής.

```
1 class Sensor extends Model
2 {
3     use BelongsToTenant, HasFactory;
4
5     public function device(): BelongsTo {...}
```

```
6
7   public function farm(): BelongsTo {...}
8
9   public function user(): BelongsTo {...}
10
11  public function scopeDeviceOnline(Builder $query): Builder
12      {...}
13
14  public function scopeFarmRelevant(Builder $query): Builder
15      {...}
16 }
```

Listing 3: Σύντομη προδιαγραφή του μοντέλου Sensor

Μέθοδοι του μοντέλου Sensor.

Το μοντέλο Sensor παρέχει βασικές σχέσεις και scopes που απλοποιούν την ανάκτηση και το φιλτράρισμα των αισθητήρων:

- **device():** Σχέση belongsTo με την Device — χρησιμοποιείται για έλεγχο κατάστασης και πρόσβαση σε πεδία της συσκευής.
- **farm():** Σχέση belongsTo με τον Farm — συνδέει τον αισθητήρα με το αγροτεμάχιο στο οποίο ανήκει.
- **user():** Σχέση belongsTo με τον User — διευκολύνει ελέγχους εξουσιοδότησης και πολυμίσθωσης.
- **scopeDeviceOnline():** Scope που φιλτράρει αισθητήρες των οποίων η συσκευή είναι ONLINE, χρήσιμο για πίνακες συστημάτων ή ειδοποιήσεις σε πραγματικό χρόνο.
- **scopeFarmRelevant():** Scope που αποκλείει εσωτερικούς/τεχνικούς τύπους (π.χ. BATTERY) και επιστρέφει μόνο γεωπονικά δεδομένα για προβολές αγρού.

Επιπλέον, τα πεδία `last_reading` και `last_reading_at` χρησιμοποιούνται ως τοπική cache της τελευταίας μέτρησης για γρήγορη εμφάνιση — οι ιστορικές χρονοσειρές ανακτώνται από την `SensorTimeSeriesService`.

Ο πλήρης ορισμός του μοντέλου παρατίθεται στο Παράρτημα Α' (Listing 71).

***Trait* πολυμίσθωσης.**

Το `BelongsToTenant` ενσωματώνει δύο μηχανισμούς: (α) κατά την εκκίνηση (boot), εφαρμόζει το `TenantScope` ως Global Scope, και (β) κατά τη δημιουργία (creating), αναθέτει αυτόματα το `user_id`. Παρέχει επίσης τις βοηθητικές μεθόδους `allTenants()` και `forTenant($userId)` για ελεγχόμενη παράκαμψη:

```
1  protected static function bootBelongsToTenant()
2  {
3      static::addGlobalScope(new TenantScope);
4
5      static::creating(function (Model $model) {
6          if (! $model->user_id && Auth::check()) {
7              $model->user_id = Auth::id();
8          }
9      });
10 }
11
12 public static function allTenants()
13 {
14     return static::withoutGlobalScope(TenantScope::class);
15 }
```

Listing 4: Απόσπασμα του trait `BelongsToTenant`

5.2.2 Enums: Τύποι και Καταστάσεις

Η πλατφόρμα αξιοποιεί τα PHP 8.1 Backed Enums για να ορίσει σύνολα τιμών με πλήρη τυποποίηση. Τα τρία βασικά enums είναι:

DeviceType Τύπος επικοινωνίας συσκευής.

DeviceStatus Κύκλος ζωής συσκευής.

SensorType Είδος αισθητήρα.

Ιδιαίτερο ενδιαφέρον παρουσιάζει το `SensorType`, το οποίο — πέρα από τις τιμές — παρέχει μεθόδους για τη μονάδα μέτρησης κάθε τύπου και τον διαχωρισμό γεωπονικών από τεχνικούς αισθητήρες:

```
1 public function unit(): string
2 {
3     return match ($this) {
4         self::MOISTURE    => '%',
5         self::TEMPERATURE => 'C',
6         self::HUMIDITY    => '%',
7         self::LIGHT       => 'lux',
8         self::BATTERY     => '%',
9         self::OTHER       => '',
10    };
11 }
12
13 public static function farmRelevantValues(): array
14 {
15     return array_values(
16         array_filter(self::values(), fn ($v) => $v !== self::
17             BATTERY->value)
18     );
19 }
```

Listing 5: Μέθοδοι του SensorType enum

Η μέθοδος `farmRelevantValues()` αποκλείει τον τύπο `BATTERY` από τις σελίδες αγρού, καθώς πρόκειται για τεχνικό δεδομένο της συσκευής και όχι για γεωπονική μέτρηση. Η ένδειξη μπαταρίας εμφανίζεται αποκλειστικά στη σελίδα λεπτομερειών της συσκευής.

Επιπλέον, ορίζεται το `TimeRange` enum που τυποποιεί τα χρονικά εύρη ανάκτησης δεδομένων και αντιστοιχίζει κάθε εύρος σε κατάλληλο παράθυρο συγκέντρωσης (aggregate window) για ερωτήματα στην InfluxDB:

```
1 enum TimeRange: string
2 {
3     case HOUR    = '-1h';
4     case DAY     = '-24h';
5     case WEEK    = '-7d';
6     case MONTH   = '-30d';
```

```
7      case QUARTER = '-90d';
8
9      public function aggregateWindow(): string
10     {
11         return match ($this) {
12             self::HOUR      => '1m',
13             self::DAY       => '15m',
14             self::WEEK      => '1h',
15             self::MONTH     => '6h',
16             self::QUARTER  => '12h',
17         };
18     }
19 }
```

Listing 6: Αντιστοίχιση χρονικού εύρους σε aggregate window

Η αντιστοίχιση αυτή διασφαλίζει ότι τα δεδομένα γραφημάτων είναι πάντα σε διαχειρίσιμο μέγεθος: για παράδειγμα, σε εύρος 7 ημερών, οι μετρήσεις ομαδοποιούνται ανά ώρα αντί να εμφανίζονται χιλιάδες μεμονωμένα σημεία.

5.2.3 Υβριδική Αποθήκευση: MySQL + InfluxDB

Η αρχιτεκτονική αποθήκευσης ακολουθεί μία υβριδική προσέγγιση, αξιοποιώντας τα πλεονεκτήματα κάθε βάσης:

- **MySQL:** Αποθηκεύει τα δομημένα δεδομένα (χρήστες, αγρούς, συσκευές, αισθητήρες) και τις μεταξύ τους σχέσεις. Κάθε αισθητήρας διατηρεί τοπικά τα πεδία `last_reading` και `last_reading_at` ως cache της πιο πρόσφατης μέτρησης, ώστε η εμφάνιση συνοπτικών δεδομένων να μην απαιτεί πρόσβαση στη βάση χρονοσειρών.
- **InfluxDB:** Αποθηκεύει το πλήρες ιστορικό μετρήσεων. Κάθε σημείο (data point) φέρει τη μέτρηση `sensor_measurement` με ετικέτες (tags) `user_id`, `farm_id`, `sensor_id`, `sensor_type` και πεδίο (field) `value`. Η δομή αυτή επιτρέπει αποδοτική ομαδοποίηση και φιλτράρισμα σε πολλαπλά επίπεδα. Η επιλογή εξειδικευμένης βάσης χρονοσειρών για το ιστορικό αισθητήρων αιτιολογείται από εμπειρικές αξιολογήσεις σε περιβάλλοντα IoT, οι οποίες αναδεικνύουν πλεονεκτήματα ρυθμού εγγραφής, συ-

μπίεσης και εκτέλεσης χρονικών ερωτημάτων σε σύγκριση με σχεσιακές βάσεις δεδομένων [23, 45].

Υπηρεσία Χρονοσειρών.

Η κλάση `SensorTimeSeriesService` ενθυλακώνει τα ερωτήματα Flux προς την InfluxDB και παρέχει τρεις βασικές μεθόδους για την ανάκτηση και παρουσίαση χρονοσειρικών δεδομένων:

- `recentReadings($sensorId, $range)` — Επιστρέφει τις πιο πρόσφατες μετρήσεις ως πίνακα `[{time, value}]`, κατάλληλο για λίστες "recent" ή γρήγορες προβολές.
- `stats($sensorId, $range)` — Υπολογίζει στατιστικά (`min, max, avg, count`) στο δοθέν χρονικό εύρος. Η μέθοδος υλοποιεί "προοδευτική εύρεση" (progressive widening) ώστε να επιστραφούν στατιστικά ακόμα και όταν τα δεδομένα είναι αραιά.
- `chartReadings($sensorId, $range)` — Επιστρέφει ομαδοποιημένα σημεία για γραφήματα στη μορφή `[timestamp_ms, value]`, χρησιμοποιώντας `aggregateWindow` ανάλογα με το `TimeRange` για να περιορίσει τον αριθμό σημείων.

Η μέθοδος `stats()` υλοποιεί **προοδευτική εύρεση** (progressive widening): εάν δεν υπάρχουν δεδομένα στο 24ωρο, αναζητά αυτόματα σε εύρος 7 ημερών και στη συνέχεια 30 ημερών, ώστε ο χρήστης να βλέπει πάντα στατιστικά ακόμη και για αισθητήρες που αποστέλλουν μετρήσεις αραιά.

Η μέθοδος `chartReadings()` αξιοποιεί τη συνάρτηση `aggregateWindow` της Flux για ομαδοποίηση μετρήσεων, χρησιμοποιώντας το παράθυρο που ορίζει το `TimeRange` enum (Λίστα 6). Ένα τυπικό ερώτημα Flux για γράφημα 24 ωρών:

```
1 from(bucket: "agronos")
2   |> range(start: -24h)
3   |> filter(fn: (r) => r._measurement == "sensor_measurement"
4     and r.sensor_id == "42" and r._field == "value")
5   |> aggregateWindow(every: 15m, fn: mean, createEmpty: false)
6   |> sort(columns: ["_time"])
```

Listing 7: Ερώτημα Flux για ομαδοποιημένες μετρήσεις γραφήματος

Ο πλήρης κώδικας των υπηρεσιών `SensorTimeSeriesService` και `InfluxDBService` παρατίθεται στο Παράρτημα Α' (Listings Α'.3).

5.3 Επίπεδο Εφαρμογής και Παρουσίασης

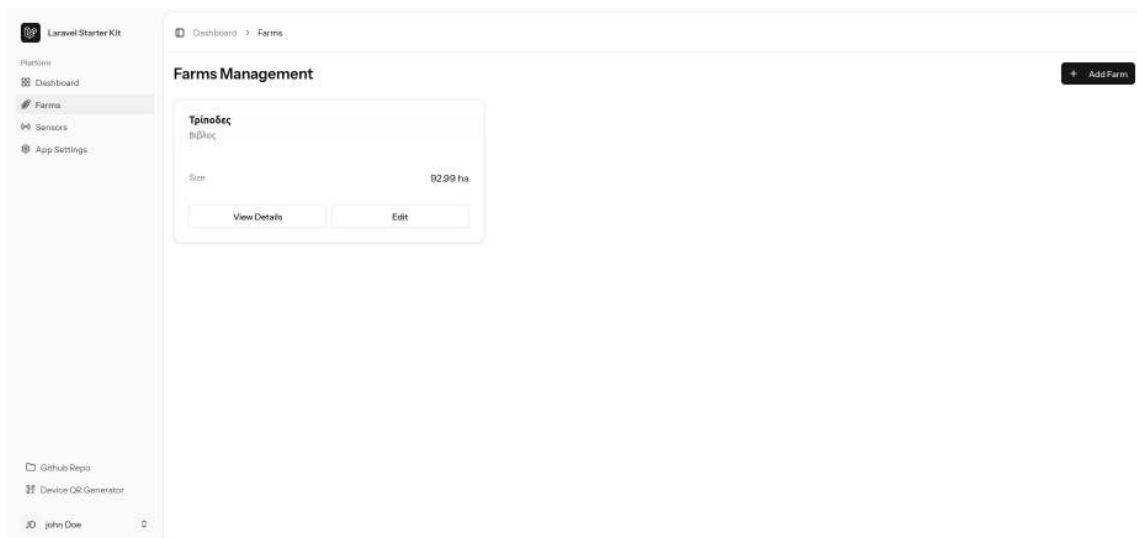
Το επίπεδο εφαρμογής παρέχει στον τελικό χρήστη μια ολοκληρωμένη διεπαφή για τη διαχείριση αγρών, συσκευών και αισθητήρων. Η υλοποίηση ακολουθεί το πρότυπο CRUD (Create, Read, Update, Delete) μέσω αντίστοιχων controllers του Laravel, οι οποίοι τροφοδοτούν σελίδες Vue.js μέσω Inertia.js. Σε κάθε λειτουργία εφαρμόζονται Authorization Policies για τον έλεγχο πρόσβασης και Form Requests για την επικύρωση δεδομένων, εξασφαλίζοντας ασφάλεια και ακεραιότητα σε επίπεδο εφαρμογής.

Οι ενότητες που ακολουθούν περιγράφουν τη λειτουργικότητα κάθε οντότητας, με έμφαση στα κρίσιμα τμήματα των controllers. Ο πλήρης κώδικας παρατίθεται στο Παράρτημα Α΄.

5.3.1 Διαχείριση Αγρών

Ο FarmController υλοποιεί τις τυπικές λειτουργίες CRUD για τα αγροτεμάχια: λίστα αγρών, δημιουργία, προβολή λεπτομερειών, επεξεργασία και διαγραφή. Κάθε ενέργεια ελέγχεται μέσω του FarmPolicy, ενώ η επικύρωση των δεδομένων εισόδου γίνεται μέσω του FarmRequest.

Η σελίδα λίστας αγρών (Farms Index) εμφανίζει τα αγροτεμάχια του χρήστη με βασικές πληροφορίες σε μορφή καρτών (Σχήμα 20).



Σχήμα 20: Σελίδα λίστας αγρών (Farms Index).

Σελίδα λεπτομερειών αγρού.

Η σελίδα Farm Show αποτελεί τον κεντρικό πίνακα ελέγχου κάθε αγροτεμαχίου. Η μέθοδος `show()` του `FarmController` συγκεντρώνει δεδομένα από δύο πηγές: τη σχεσιακή βάση για τα μεταδεδομένα και την `InfluxDB` για τα χρονοσειριακά στατιστικά.

Τα δεδομένα που φορτώνονται από τη σχεσιακή βάση περιλαμβάνουν: τη λίστα αισθητήρων με φιλτράρισμα μέσω `farmRelevant()` (αποκλεισμός τεχνικών τύπων όπως `BATTERY`), τον συνολικό αριθμό αισθητήρων, πλήθος ανά τύπο και τον μέσο όρο τελευταίας μέτρησης ανά τύπο:

```
1 public function show(Farm $farm)
2 {
3     $this->authorize('view', $farm);
4     $sensors = $farm->sensors()->farmRelevant()->get();
5
6     $totalSensors = $farm->sensors()->farmRelevant()->count();
7     $sensorTypeStats = $farm->sensors()
8         ->farmRelevant()
9         ->selectRaw('type, COUNT(*) as count')
10        ->whereNotNull('type')
11        ->groupBy('type')
12        ->pluck('count', 'type')
13        ->toArray();
14
15     $lastAvgByType = $farm->sensors()
16         ->farmRelevant()
17         ->deviceOnline()
18         ->selectRaw('type, AVG(last_reading) as avg_last_reading')
19         ->whereNotNull('type')
20         ->whereNotNull('last_reading')
21         ->groupBy('type')
22         ->pluck('avg_last_reading', 'type')
23         ->mapWithKeys(fn ($v, $k) => [$k => $v !== null ? (float) $v
24             : null])
25         ->toArray();
```

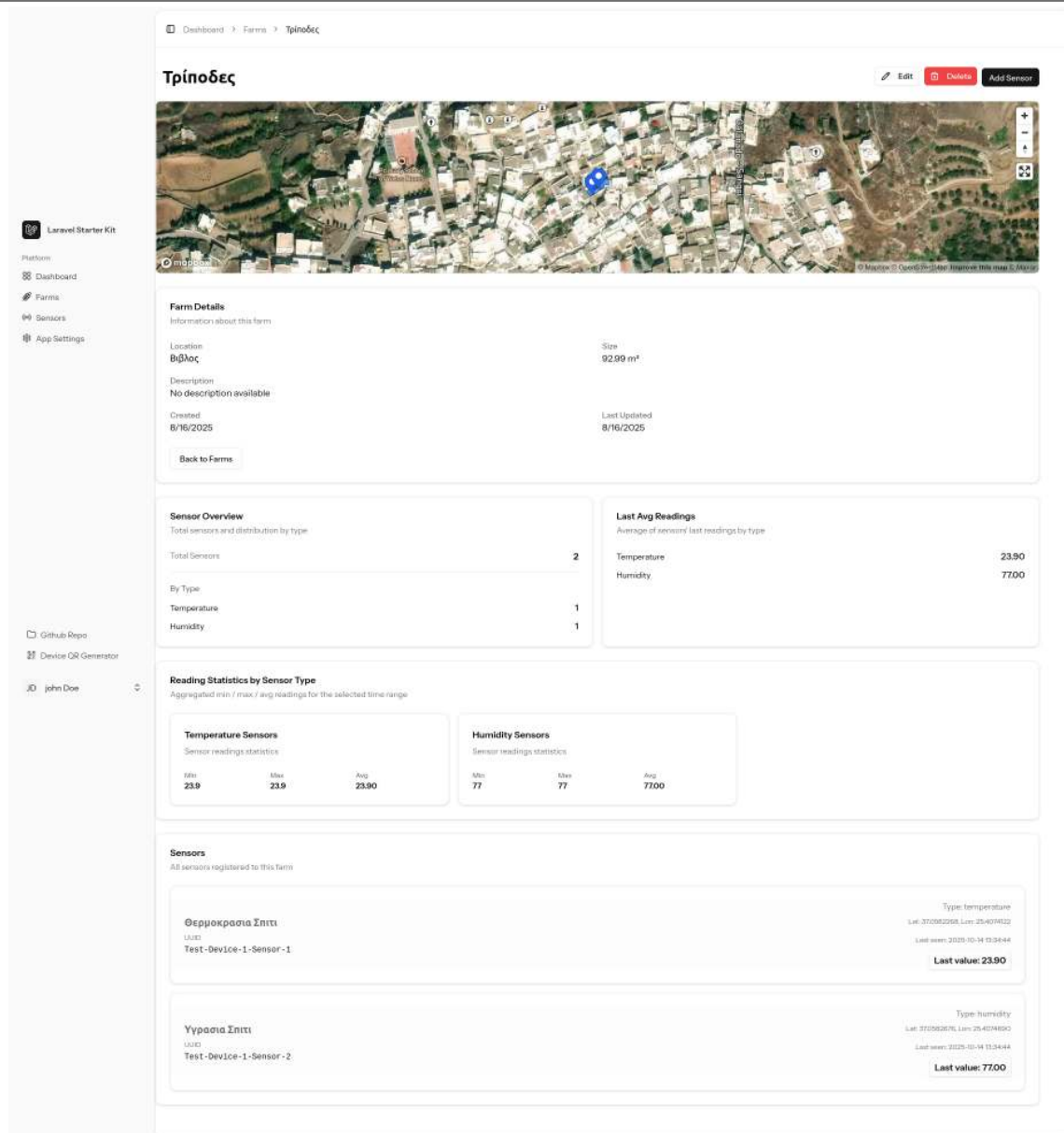
```
25 // ...  
26 }
```

Listing 8: Υπολογισμός στατιστικών αγρού — απόσπασμα FarmController::show()

Τα στατιστικά χρονοσειρών (min, max, avg ανά τύπο αισθητήρα) φορτώνονται μέσω deferred props (βλ. ενότητα 5.3.4) ώστε η αρχική σελίδα να εμφανίζεται ταχύτατα.

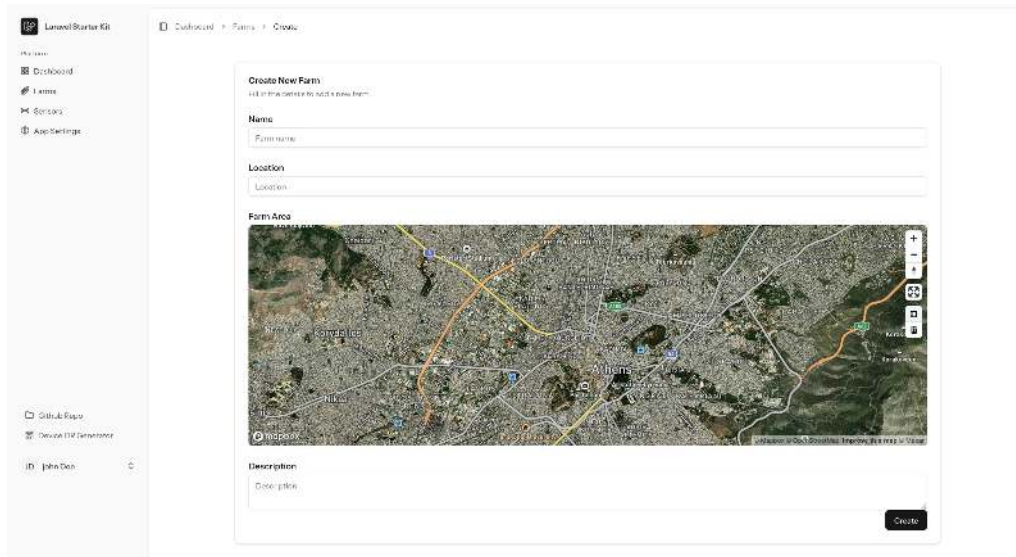
Η σελίδα παρουσιάζει (Σχήμα 21):

- Κάρτες συνοπτικών στατιστικών ανά τύπο αισθητήρα (min, max, avg).
- Διαδραστικό χάρτη με τη θέση του αγρού (πολύγωνο) και τους αισθητήρες ως σημεία.
- Λίστα αισθητήρων (SensorCard) με τελευταία μέτρηση, τύπο, μονάδα μέτρησης και κατάσταση.



Σχήμα 21: Σελίδα λεπτομερειών αγρού με κάρτες στατιστικών, χάρτη και λίστα αισθητήρων.

Η δημιουργία και η επεξεργασία αγρού γίνονται μέσω φόρμας (Σχήμα 22) με πεδία ονόματος, τοποθεσίας, μεγέθους, περιγραφής και γεωγραφικών συντεταγμένων (σχεδίαση πολυγώνου στον χάρτη).

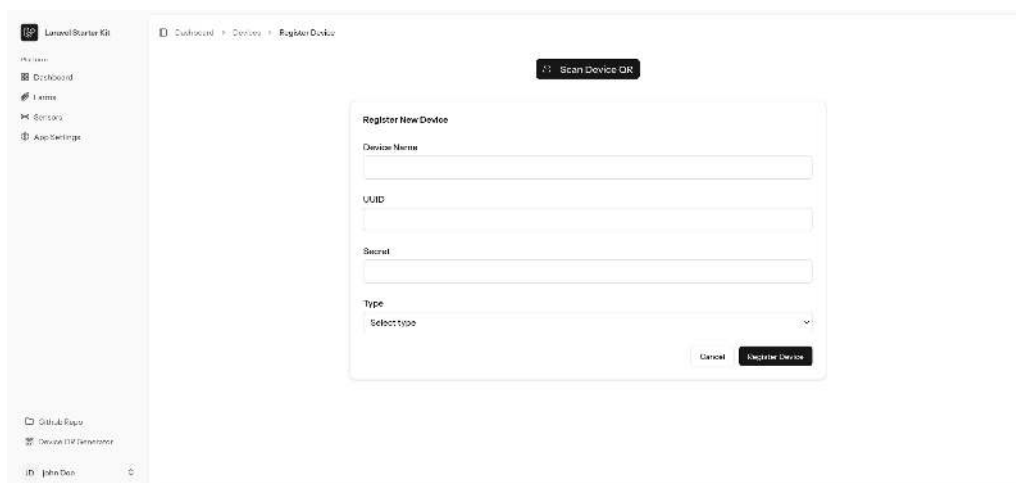


Σχήμα 22: Φόρμα δημιουργίας αγρού.

Ο πλήρης κώδικας του FarmController παρατίθεται στο Παράρτημα Α΄ (Listing 75).

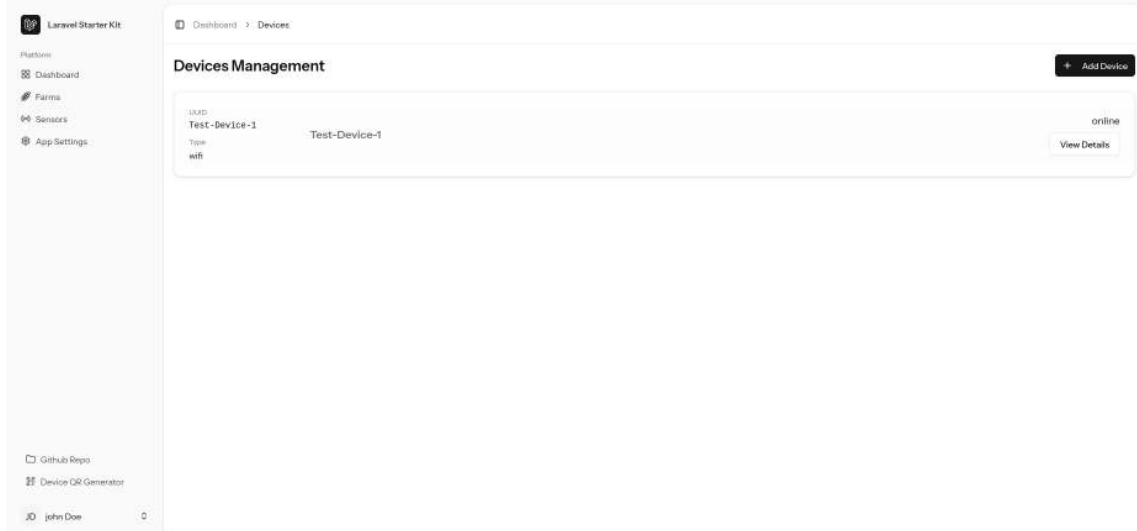
5.3.2 Διαχείριση Συσκευών

Ο DeviceController υλοποιεί τη διαχείριση κύκλου ζωής των IoT συσκευών. Η εγγραφή μιας νέας συσκευής γίνεται μέσω φόρμας (Σχήμα 23), στην οποία ο χρήστης εισάγει το UUID και το secret — είτε χειροκίνητα, είτε μέσω σάρωσης του QR code που συνοδεύει τη φυσική συσκευή.



Σχήμα 23: Φόρμα εγγραφής συσκευής μέσω σάρωσης QR code ή χειροκίνητης εισαγωγής.

Η λίστα συσκευών (Devices Index) εμφανίζει τις καταχωρημένες συσκευές του χρήστη με τον τύπο, την κατάσταση (registered, online, offline) και βασικές επιλογές διαχείρισης (Σχήμα 24).



Σχήμα 24: Σελίδα λίστας συσκευών (Devices Index).

Σελίδα λεπτομερειών συσκευής.

Η σελίδα Device Show (Σχήμα 25) εμφανίζει τα χαρακτηριστικά της συσκευής, ένδειξη στάθμης μπαταρίας, τη λίστα αισθητήρων που ανήκουν στη συσκευή και τα διαπιστευτήρια MQTT (εφόσον έχουν εκδοθεί).

Η ανάκτηση της στάθμης μπαταρίας υλοποιείται μέσω μιας ιδιωτικής μεθόδου που ελέγχει πρώτα το πεδίο `battery_level` της ίδιας της συσκευής και, αν αυτό είναι κενό, αναζητά τον αισθητήρα τύπου `BATTERY`. Η προσέγγιση αυτή αξιοποιεί την υπάρχουσα υποδομή αισθητήρων χωρίς να απαιτεί ειδικό μηχανισμό στη ροή κατανάλωσης δεδομένων:

```
1 private function getBatteryLevel(Device $device, $sensors): ?  
   float  
2 {  
3     if ($device->battery_level !== null) {  
4         return is_numeric($device->battery_level) ? (float)  
           $device->battery_level : null;  
5     }  
6  
7     $batterySensor = $sensors->firstWhere('type', SensorType::  
           BATTERY->value);  
8     if ($batterySensor && $batterySensor->last_reading !== null) {
```



```
9      return is_numeric($batterySensor→last_reading)
10          ? (float) $batterySensor→last_reading : null;
11    }
12
13    return null;
14  }
```

Listing 9: Ανάκτηση στάθμης μπαταρίας — DeviceController

Τα ιστορικά δεδομένα μπαταρίας φορτώνονται επίσης ασύγχρονα μέσω deferred props, ώστε η σελίδα να μην καθυστερεί λόγω κλήσεων στην InfluxDB.

The screenshot displays the configuration page for a device named 'Test-Device-2' (firebeetle-2-esp32-c8). The device is online with a battery level of 79.6%. It lists MQTT credentials (username: Test-Device-2, password: *****), three sensors (humidity, temperature, and battery level), and a battery status section showing the current level (79.6%) and recent readings.

Device Information:

- UUID: Test-Device-2
- Type: firebeetle-2-esp32-c8
- WiFi: [redacted]
- Created At: 2026-01-04T17:47:42.000000Z
- Last Seen: 2026-02-01 18:40:06
- Status: online
- Battery: 79.6%

MQTT Credentials:

- Username: Test-Device-2
- Password: *****

Sensors:

- Υγρασία firebeetle DHT20**
Type: humidity
Lat: 37.0562676, Lon: 25.4074690
Last seen: 2026-02-01 20:40:06
Last value: 69.79
- Θερμοκρασία firebeetle**
Type: temperature
Lat: 37.0562676, Lon: 25.4074690
Last seen: 2026-02-01 20:40:06
Last value: 17.23
- Device 1 Battery**
Type: battery
Lat: 37.0562676, Lon: 25.4074690
Last seen: 2026-02-01 20:40:06
Last value: 79.60

Battery:

- Device battery level and recent readings
- Current: 79.6%
- Recent readings (last 5):
 - 2026-02-01 18:40:06 — 79.6
 - 2026-02-01 18:36:56 — 79.6
 - 2026-02-01 18:33:47 — 79.6
 - 2026-02-01 18:27:27 — 79.6
 - 2026-02-01 18:24:18 — 78.8

[Back to List](#)

Σχήμα 25: Σελίδα λεπτομερειών συσκευής με ένδειξη μπαταρίας, λίστα αισθητήρων και MQTT διαπιστευτήρια.

ιαπιστευτήρια MQTT.

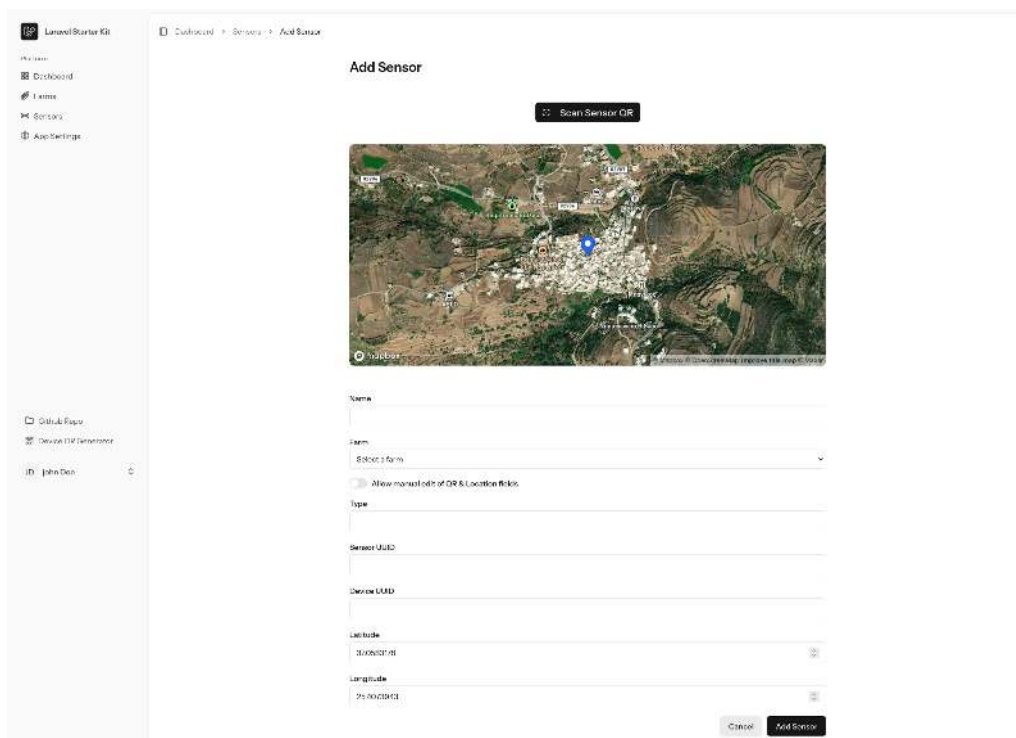
Η μέθοδος `createMqttCredentials()` εκδίδει τα MQTT διαπιστευτήρια μιας συσκευής μέσω της υπηρεσίας `MqttCredentialService`. Τα διαπιστευτήρια εμφανίζονται στη σελίδα της συσκευής (Σχήμα 25) για τη διευκόλυνση της ρύθμισης παραμέτρων.

Ο πλήρης κώδικας του `DeviceController` παρατίθεται στο Παράρτημα Α' (Listing 76).

5.3.3 Διαχείριση Αισθητήρων

Ο `SensorController` υλοποιεί τη διαχείριση αισθητήρων, με δύο τρόπους εγγραφής: (α) χειροκίνητη εισαγωγή μέσω φόρμας και (β) σάρωση QR code. Η δεύτερη μέθοδος αυτοματοποιεί τη ροή εγγραφής στο πεδίο, καθώς το QR code κάθε αισθητήρα περιέχει τα πεδία `uuid` (αισθητήρα), `device_uuid` (συσκευής) και `type`, εξαλείφοντας τη χειροκίνητη εισαγωγή.

Η φόρμα εγγραφής (Σχήμα 26) παρέχει πεδία ονόματος, αγρού, τύπου, UUID αισθητήρα/συσκευής και γεωγραφικών συντεταγμένων. Εφόσον η φόρμα ανοίγει από τη σελίδα ενός συγκεκριμένου αγρού, ο αγρός προεπιλέγεται αυτόματα. Επιπλέον, όταν η φόρμα προσπελαστεί από συσκευή που διαθέτει GPS (π.χ. κινητό τηλέφωνο), τα πεδία γεωγραφικού πλάτους και μήκους συμπληρώνονται αυτόματα από τις τρέχουσες συντεταγμένες της συσκευής, εξαλείφοντας την ανάγκη χειροκίνητης εισαγωγής τους.



Σχήμα 26: Φόρμα δημιουργίας αισθητήρα με δυνατότητα σάρωσης QR code.

Ροή σάρωσης QR code — δημιουργία ή μετεγκατάσταση.

Η μέθοδος `scan()` σχεδιάστηκε σκόπιμα με διπλή ροή, για να εξυπηρετεί δύο διαφορετικές ανάγκες με την **ίδια ακριβώς** διαδικασία από πλευράς χρήστη:

- **Πρώτη εγγραφή:** Ο αισθητήρας δεν υπάρχει ακόμα στο σύστημα — η σάρωση τον

δημιουργεί, καταχωρώντας τα στοιχεία του QR code μαζί με τις τρέχουσες γεωγραφικές συντεταγμένες από το GPS του κινητού.

- **Μετεγκατάσταση:** Ο αισθητήρας υπάρχει ήδη και ο αγρότης τον μετακινεί σε νέα θέση στο χωράφι. Αρκεί να βγάλει τον κόμβο, να τον τοποθετήσει αλλού και να σαρώσει ξανά το QR code του από το νέο σημείο — η εφαρμογή αναγνωρίζει το uuid, αντιλαμβάνεται ότι η εγγραφή υπάρχει και ενημερώνει αθόρυβα τις γεωγραφικές συντεταγμένες — αντλώντας τες από το GPS της συσκευής του χρήστη — χωρίς να απαιτείται καμία άλλη ενέργεια από τον χρήστη.

Η αρχιτεκτονική αυτή αποφεύγει την ανάγκη ξεχωριστής «φόρμας μετεγκατάστασης», ελαχιστοποιώντας την τριβή για τον αγρότη στο πεδίο (βλ. Σενάριο Ε στο Κεφάλαιο 3). Σε κάθε περίπτωση, η μέθοδος επαληθεύει ότι η συσκευή ανήκει στον ίδιο χρήστη μέσω του DevicePolicy:

```
1 public function scan(ScanSensorRequest $request)
2 {
3     $validated = $request->validated();
4
5     $device = Device::where('uuid', $validated['device_uuid'])->
        firstOrFail();
6     $sensor = Sensor::where('uuid', $validated['uuid'])->first();
7     $this->authorize('view', $device);
8
9     if ($sensor) {
10         $sensor->update(
11             collect($validated)->except(['device_uuid', 'uuid'])->
                toArray()
12         );
13         return redirect()->route('sensors.index');
14     } else {
15         $this->authorize('create', Sensor::class);
16         $sensor = Sensor::create(
17             collect($validated)->except('device_uuid')->toArray()
18             + [
```

```
19         'user_id' => $request->user()->id,  
20         'device_id' => $device->id,  
21     ]  
22 );  
23     return redirect()->route('sensors.index');  
24 }  
25 }
```

Listing 10: Μέθοδος σάρωσης αισθητήρα — SensorController::scan()

Σελίδα λεπτομερειών αισθητήρα.

Στην σελίδα Sensor Show (Σχήμα 27) έχουμε τον βασικό πίνακα ελέγχου κάθε αισθητήρα. Στη μέθοδο show(), τα δεδομένα φορτώνονται σε δύο φάσεις:

1. **Αρχική φόρτωση:** Τα μεταδεδομένα (όνομα, τύπος, αγρός, συσκευή, τελευταία μέτρηση) από τη σχεσιακή βάση, μέσω του SensorResource.
2. **Ασύγχρονη φόρτωση (deferred):** Πρόσφατες μετρήσεις, στατιστικά και δεδομένα γραφήματος από την InfluxDB.

```
1 public function show(Sensor $sensor)  
2 {  
3     $this->authorize('view', $sensor);  
4     $sensor->load(['farm', 'device']);  
5  
6     $ts = app(SensorTimeSeriesService::class);  
7     $range = request()->query('range', TimeRange::default()->value)  
8  
9     return Inertia::render('Sensors/Show', [  
10         'sensor' => (new SensorResource($sensor))->flat(request()),  
11         'rangeOptions' => TimeRange::options(),  
12         'selectedRange' => $range,  
13         'recentReadings' => Inertia::defer(  
14             fn () => $ts->recentReadings($sensor->id, $range, 20)  
15         ),  
16     ]  
17 );
```

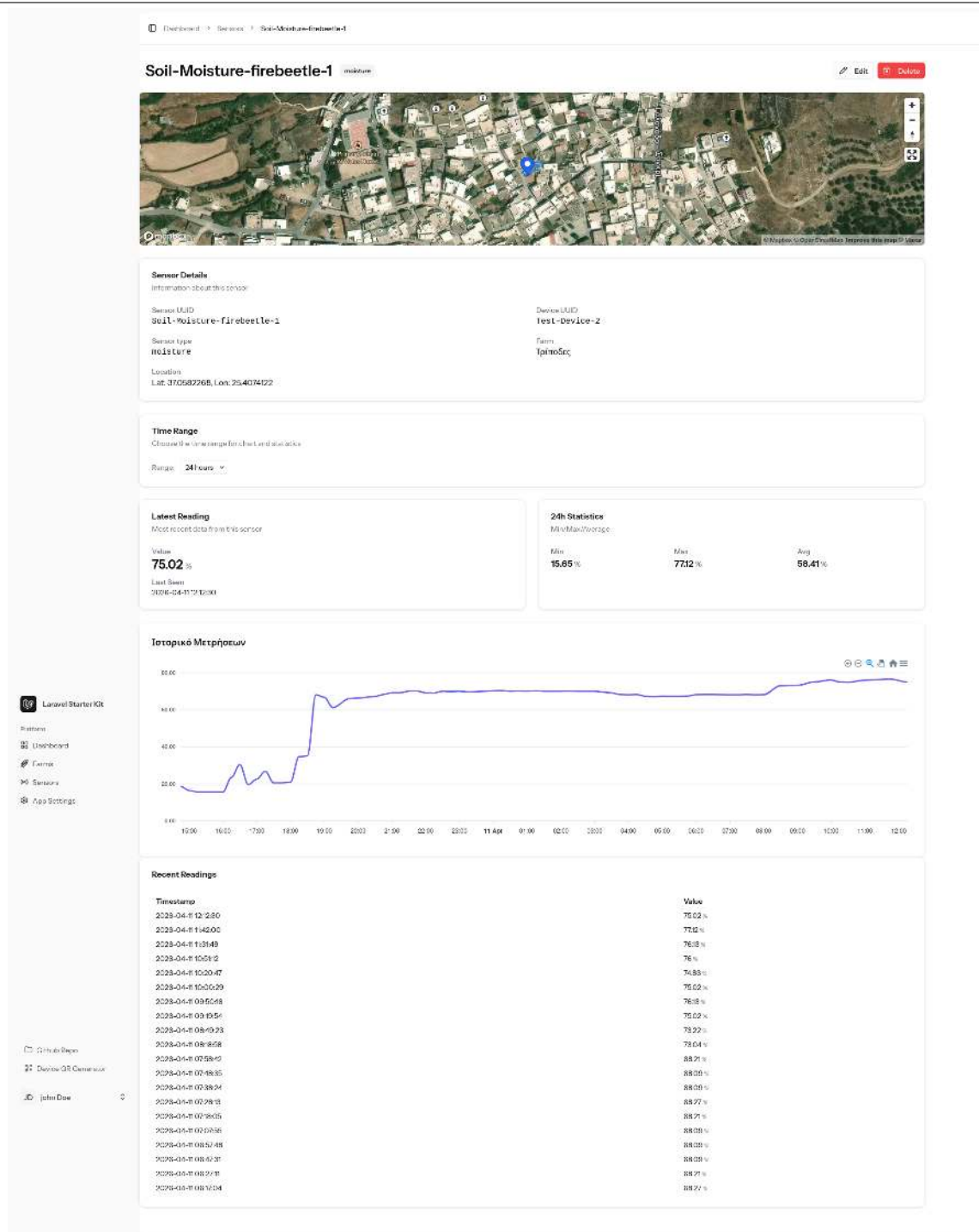
```
16      'stats' => Inertia::defer(  
17          fn () => $ts->stats($sensor->id, $range)  
18      ),  
19      'chartData' => Inertia::defer(  
20          fn () => $ts->chartReadings($sensor->id, $range)  
21      ),  
22  ]);  
23 }
```

Listing 11: Φόρτωση σελίδας αισθητήρα — `SensorController::show()`

Η σελίδα παρουσιάζει:

- Τα μεταδεδομένα του αισθητήρα (τύπος, μονάδα, αγρός, συσκευή).
- Κάρτες στατιστικών (min, max, avg) για το επιλεγμένο χρονικό εύρος.
- Διαδραστικό γράφημα (ApexCharts) με ομαδοποιημένες μετρήσεις, με δυνατότητα εναλλαγής χρονικού εύρους (1 ώρα, 24 ώρες, 7 ημέρες κ.λπ.).
- Πίνακα πρόσφατων μετρήσεων.

Το γράφημα τροφοδοτείται από τη μέθοδο `chartReadings()` της `SensorTimeSeriesService`, η οποία εφαρμόζει `aggregateWindow` ανάλογα με το επιλεγμένο εύρος (Λίστα 6), ώστε ο αριθμός σημείων να παραμένει διαχειρίσιμος.



Σχήμα 27: Σελίδα λεπτομερειών αισθητήρα με γράφημα μετρήσεων, στατιστικά και πρόσφατες τιμές.

Ο πλήρης κώδικας του `SensorController` παρατίθεται στο Παράρτημα Α' (Listing 77).

5.3.4 Βελτιστοποίηση Απόδοσης — Deferred Props (Inertia v2)

Η φόρτωση δεδομένων χρονοσειρών από την InfluxDB μπορεί να εισαγάγει σημαντική καθυστέρηση, ιδιαίτερα σε σελίδες με πολλαπλούς αισθητήρες ή μεγάλα χρονικά εύρη. Για

να εξασφαλιστεί ταχύτατη αρχική εμφάνιση, υιοθετήθηκε η τεχνική Deferred Props του Inertia.js v2: η αρχική σελίδα αποδίδεται αμέσως με τα δεδομένα της σχεσιακής βάσης, ενώ τα χρονοσειριακά δεδομένα φορτώνονται ασύγχρονα και ενημερώνουν τη διεπαφή μόλις είναι διαθέσιμα.

Η χρήση γίνεται μέσω της μεθόδου `Inertia::defer()`, η οποία δέχεται ένα closure που αξιολογείται σε δεύτερο HTTP αίτημα, χωρίς ο χρήστης να αντιλαμβάνεται καθυστέρηση:

```
1 return Inertia::render('Farms/Show', [  
2   'farm' => $farm,  
3   'sensors' => $sensors,  
4   'sensorDbStats' => $sensorDbStats,  
5   'timeSeriesStats' => Inertia::defer(  
6     fn () => $ts->farmStats($farm, '-24h')  
7   ),  
8 ]);
```

Listing 12: Χρήση `Inertia::defer()` στον `FarmController`

Η ίδια τεχνική εφαρμόζεται σε τρεις σελίδες:

Farm Show: Τα στατιστικά χρονοσειρών φορτώνονται ασύγχρονα.

Device Show: Τα ιστορικά μπαταρίας φορτώνονται ασύγχρονα.

Sensor Show: Τα δεδομένα γραφήματος, τα στατιστικά και οι πρόσφατες μετρήσεις φορτώνονται ασύγχρονα.

Κατά τη φάση ασύγχρονης φόρτωσης, η διεπαφή εμφανίζει — στοιχεία κράτησης θέσης με παλμικό εφέ (pulsing animation) — στη θέση των δεδομένων που εκκρεμούν. Μόλις ολοκληρωθεί η φόρτωση, η σελίδα ενημερώνεται αυτόματα χωρίς refresh, αξιοποιώντας τη δυναμική αντίδραση (reactivity) του Vue.js.

Η προσέγγιση αυτή προσφέρει δύο βασικά πλεονεκτήματα: πρώτον, εξασφαλίζει μικρότερο χρόνο αρχικής εμφάνισης, επιτρέποντας στον χρήστη να δει αμέσως τα βασικά δεδομένα, και δεύτερον, προσφέρει καλύτερη αντίληψη της απόδοσης, καθώς το σχετικό εφέ τον ενημερώνει ότι επίκειται η φόρτωση επιπλέον δεδομένων

5.4 REST API Συσκευών

Η πλατφόρμα παρέχει ένα σύνολο REST API endpoints αποκλειστικά σχεδιασμένων για την επικοινωνία με τις IoT συσκευές. Τα endpoints ακολουθούν εκδοσιοποίηση (/api/v1/) και ομαδοποιούνται ανά λειτουργία: αυθεντικοποίηση, αποστολή δεδομένων, χορήγηση MQTT διαπιστευτηρίων και λήψη LoRa δεδομένων. Η αρχιτεκτονική των μονοπατιών επικοινωνίας περιγράφηκε αναλυτικά στην ενότητα 4.4 — η ενότητα αυτή εστιάζει στην υλοποίηση Laravel κάθε endpoint.

Ο Πίνακας 8 συνοψίζει τα διαθέσιμα endpoints:

Μέθοδος	URI	Controller	Περιγραφή
POST	/api/v1/device/login	DeviceAuthController	Αυθεντικοποίηση συσκευής
POST	/api/v1/device/data	DeviceDataController	Λήψη μετρήσεων (HTTP)
GET	/api/v1/device/mqtt-credentials	DeviceAuthController	Χορήγηση MQTT διαπιστευτηρίων
POST	/api/v1/device/mqtt-webhook	DeviceDataController	Webhook από EMQX
POST	/api/v1/lora/webhook	LoRaDataController	Webhook για LoRa δεδομένα

Πίνακας 8: Πίνακας REST API endpoints συσκευών

Τα endpoints που απαιτούν αυθεντικοποίηση προστατεύονται μέσω του middleware `auth:sanctum`. Τα δύο webhook endpoints δεν απαιτούν Sanctum token, καθώς καλούνται από τον EMQX Rule Engine — η ταυτοποίηση της συσκευής γίνεται μέσω του `username` (UUID συσκευής) που περιέχεται στο σώμα του αιτήματος.

5.4.1 Αυθεντικοποίηση Συσκευών (DeviceAuthController)

Η μέθοδος `login()` υλοποιεί την αρχική αυθεντικοποίηση κάθε συσκευής. Η συσκευή αποστέλλει το UUID και το `secret` μέσω POST· το Backend αναζητά τη συσκευή στη βάση, επαληθεύει το `secret` και σε περίπτωση επιτυχίας, εκδίδει Bearer token μέσω Laravel Sanctum:

```
1 public function login(Request $request)
2 {
3     $request->validate([
4         'uuid' => 'required|string',
```

```
5         'secret' => 'required|string',
6     ]);
7
8     $device = Device::where('uuid', $request->uuid)->first();
9
10    if (! $device || ! Hash::check($request->secret, $device->
        secret)) {
11        return response()->json(['message' => 'Invalid credentials'
            ], 401);
12    }
13
14    $device->update([
15        'status' => DeviceStatus::ONLINE,
16        'last_seen_at' => now(),
17    ]);
18
19    $token = $device->createToken('device-token')->plainTextToken;
20    return response()->json(['token' => $token]);
21 }
```

Listing 13: Αυθεντικοποίηση συσκευής — DeviceAuthController::login()

Κατά την επιτυχή αυθεντικοποίηση, η κατάσταση της συσκευής ενημερώνεται σε ONLINE και καταγράφεται η χρονοσφραγίδα τελευταίας σύνδεσης. Το secret αποθηκεύεται στη βάση ως bcrypt hash (μέσω Hash::make() κατά την εγγραφή), ώστε ακόμη και σε περίπτωση παραβίασης της βάσης, τα διαπιστευτήρια να παραμένουν προστατευμένα.

Ο πλήρης κώδικας του DeviceAuthController παρατίθεται στο Παράρτημα Α' (Listing 78).

5.4.2 Λήψη Δεδομένων μέσω HTTP (DeviceDataController)

Η μέθοδος store() αποτελεί το κύριο endpoint λήψης μετρήσεων από WiFi συσκευές μέσω απευθείας HTTP. Η συσκευή αποστέλλει ένα JSON σώμα με πίνακα μετρήσεων, όπου κάθε στοιχείο περιέχει το uuid του αισθητήρα και την value:

```
1 {
2     "sensors": [
```

```
3      { "uuid": "sens-temp-01", "value": 25.4 },  
4      { "uuid": "sens-hum-01", "value": 62.1 }  
5  ]  
6 }
```

Listing 14: Παράδειγμα payload αποστολής δεδομένων

Η μέθοδος αξιοποιεί το `DeviceDataRequest` για επικύρωση και προωθεί τα δεδομένα στην `SensorDataService`:

```
1 public function store(DeviceDataRequest $request,  
    SensorDataService $sensorDataService)  
2 {  
3     $device = $request→user();  
4     $device→update([  
5         'status' ⇒ DeviceStatus::ONLINE,  
6         'last_seen_at' ⇒ now(),  
7     ]);  
8     $sensorPayloads = $request→validated()['sensors'];  
9     $response = $sensorDataService→processSensorData($device,  
        $sensorPayloads);  
10  
11     return response()→json($response, 200);  
12 }
```

Listing 15: Λήψη δεδομένων — `DeviceDataController::store()`

MQTT Webhook.

Η ίδια κλάση περιέχει τη μέθοδο `mqttBrokerWebhook()`, η οποία εξυπηρετεί το δεύτερο μονοπάτι επικοινωνίας (WiFi → MQTT → Backend). Ο EMQX Rule Engine προωθεί κάθε μήνυμα MQTT ως HTTP POST στο `/api/v1/device/mqtt-webhook`. Η μέθοδος εξαγάγει το `username` (UUID συσκευής) και το `payload` (JSON μετρήσεων) από το σώμα του αιτήματος, εντοπίζει τη συσκευή στη βάση και καλεί την `SensorDataService` — ακολουθώντας ακριβώς την ίδια ροή με το HTTP endpoint:

```
1 public function mqttBrokerWebhook(Request $request,  
    SensorDataService $sensorDataService)
```

```
2      {
3          $device = $request['username'];
4          $payload = $request['payload'];
5          // Find device by uuid (assuming username is the device
           UUID)
6          $device = Device::where('uuid', $request['username'])→
           first();
7          if (! $device) {
8              return response()→json(['message' ⇒ 'Device not found
           '], 404);
9          }
10         // // Update device status and last seen
11         $device→update([
12             'status' ⇒ DeviceStatus::ONLINE,
13             'last_seen_at' ⇒ now(),
14         ]);
15         $sensorPayloads = $payload ? json_decode($payload, true)
           : [];
16         $sensorPayloads = $sensorPayloads['sensors'] ?? [];
17         $response = $sensorDataService→processSensorData($device,
           $sensorPayloads);
18
19         return response()→json(['message' ⇒ 'Webhook received'],
           200);
20     }
```

Listing 16: MQTT Webhook — DeviceDataController::mqttBrokerWebhook()

Ο πλήρης κώδικας του DeviceDataController παρατίθεται στο Παράρτημα Α' (Listing 79).

5.4.3 Διαπιστευτήρια MQTT (MqttCredentialService)

Η χορήγηση MQTT διαπιστευτηρίων πραγματοποιείται μέσω του endpoint GET /api/v1/device/mqtt- το οποίο απαιτεί αυθεντικοποίηση Sanctum. Η λογική ενθυλακώνεται στην υπηρεσία MqttCredentialService, η οποία λειτουργεί ως εξής: εάν η συσκευή διαθέτει ήδη διαπιστευτήρια, τα επιστρέφει· αλλιώς, δημιουργεί νέα. Η διαδικασία δημιουργίας περιλαμ-

βάνει τρία βήματα:

1. Παραγωγή τυχαίου κωδικού 16 bytes (`bin2hex(random_bytes(16))`).
2. Καταχώρηση του χρήστη στον EMQX broker μέσω του REST API του (`EmqxService::createUser`).
3. Αποθήκευση των πεδίων `mqtt_username` (= UUID συσκευής) και `mqtt_password` στο μοντέλο Device.

```
1 public function createCredentials(Device $device): array
2 {
3     if ($device->mqtt_username && $device->mqtt_password) {
4         return [
5             'mqtt_broker_url' => config('services.emqx.url', '/'),
6             'username' => $device->mqtt_username,
7             'password' => $device->mqtt_password,
8             'created' => false,
9         ];
10    }
11
12    $username = $device->uuid;
13    $password = bin2hex(random_bytes(16));
14
15    $emqx = app(EmqxService::class);
16    $result = $emqx->createUser($username, $password);
17
18    $device->update([
19        'mqtt_username' => $username,
20        'mqtt_password' => $password,
21    ]);
22
23    return [
24        'mqtt_broker_url' => config('services.emqx.url', '/'),
25        'username' => $username,
26        'password' => $password,
27        'created' => true,
```

```
28     ];  
29 }
```

Listing 17: Δημιουργία MQTT διαπιστευτηρίων — απόσπασμα `MqttCredentialService`

Η απάντηση περιλαμβάνει τη διεύθυνση του broker, τα διαπιστευτήρια και τη σημαία `created` που υποδεικνύει αν δημιουργήθηκαν νέα ή επιστράφηκαν υπάρχοντα. Σε περίπτωση αδυναμίας επικοινωνίας με τον EMQX, η μέθοδος επιστρέφει σφάλμα `emqx_unavailable` χωρίς να αποθηκεύσει μερικά δεδομένα.

Ο πλήρης κώδικας της `MqttCredentialService` παρατίθεται στο Παράρτημα Α' (Listing 81).

5.4.4 LoRa Webhook (LoRaDataController)

Το endpoint `POST /api/v1/lora/webhook` εξυπηρετεί το τρίτο μονοπάτι επικοινωνίας. Ο EMQX Rule Engine προωθεί τα μηνύματα από τα LoRa gateways (topic `lora/+/data`) ως HTTP POST. Ο `LoRaDataController` υλοποιεί μια πολύπλοκη αλυσίδα επεξεργασίας, η οποία περιγράφηκε αρχιτεκτονικά στην ενότητα 4.4:

1. **Αποκωδικοποίηση EMQX envelope:** Εξαγωγή του εσωτερικού JSON payload μέσω `LoRaWebhookRequest` (Form Request).
2. **Δεύτερη επικύρωση:** Εφαρμογή `Validator::make()` στο εσωτερικό JSON (`gateway_id`, `raw_payload`, `rsi`, `snr`).
3. **Ανάλυση δυαδικού πακέτου:** Αποκωδικοποίηση `hex` → `binary`, εξαγωγή UUID, frame counter και κρυπτογραφημένου φορτίου σύμφωνα με τη δομή του Πίνακα 3.
4. **Εντοπισμός συσκευής:** Αναζήτηση με `allTenants()` (παράκαμψη tenant scope) φιλτράροντας κατά UUID και τύπο LORA.
5. **Αντι-επανάληψη:** Κλήση `LoRaCryptoService::validateFrameCounter()` — σε περίπτωση αποτυχίας, επιστρέφεται κωδικός 409 (replay) ή 422 (gap).
6. **Αποκρυπτογράφηση:** Κλήση `LoRaCryptoService::decrypt()` (AES-128-CTR).
7. **Αποσειριοποίηση:** Κλήση `LoRaCryptoService::deserialize()` — αντιστοίχιση 4-χαρακτηρων UUID prefix σε εγγεγραμμένους αισθητήρες.

8. **Κλήση `SensorDataService`:** Τα αποκρυπτογραφημένα δεδομένα τροφοδοτούνται στην ίδια υπηρεσία που χρησιμοποιούν οι WiFi συσκευές.

Τα κρίσιμα βήματα εξαγωγής και εντοπισμού παρουσιάζονται στο ακόλουθο απόσπασμα:

```
1 // Decode hex → binary, extract UUID, frame counter, ciphertext
2 $rawBytes = hex2bin($validated['raw_payload']);
3 $uuidLen = ord($rawBytes[0]);
4 $headerLen = 1 + $uuidLen + 4;
5
6 $uuid = substr($rawBytes, 1, $uuidLen);
7 $fcnt = unpack('V', substr($rawBytes, 1 + $uuidLen, 4))[1];
8 $ciphertext = base64_encode(substr($rawBytes, $headerLen));
9
10 // Look up the LoRa device by its embedded UUID
11 $device = Device::allTenants()
12     →where('uuid', $uuid)
13     →where('type', DeviceType::LORA→value)
14     →first();
```

Listing 18: Ανάλυση δυαδικού πακέτου και εντοπισμός συσκευής — απόσπασμα `LoRaDataController`

Μετά την αποκρυπτογράφηση, η αντιστοίχιση μετρήσεων σε αισθητήρες γίνεται μέσω σύγκρισης των πρώτων 4 χαρακτήρων του UUID κάθε εγγεγραμμένου αισθητήρα με τα prefix που περιέχονται στο αποκρυπτογραφημένο πακέτο:

```
1 $sensors = $device→sensors()→get();
2 $sensorPayloads = [];
3 foreach ($sensors as $sensor) {
4     $prefix = substr($sensor→uuid, 0, 4);
5     if (array_key_exists($prefix, $readings)) {
6         $sensorPayloads[] = [
7             'uuid' ⇒ $sensor→uuid,
8             'value' ⇒ $readings[$prefix],
9         ];
10    }
```

```
10     }  
11 }  
12 $response = $sensorDataService->processSensorData($device ,  
    $sensorPayloads);
```

Listing 19: Αντιστοίχιση μετρήσεων σε αισθητήρες μέσω UUID prefix

Αξίζει να σημειωθεί ότι η κλήση `processSensorData()` είναι κοινή για όλα τα μονοπάτια, γεγονός που σημαίνει ότι οι εγγραφές InfluxDB, οι ενημερώσεις μοντέλου και τα WebSocket broadcasts λειτουργούν αυτόματα και για LoRa δεδομένα χωρίς επιπλέον κώδικα.

Ο πλήρης κώδικας του `LoRaDataController` παρατίθεται στο Παράρτημα Α' (Listing 80), ενώ η `LoRaCryptoService` στο Listing 83.

5.4.5 Form Requests και Επικύρωση

Η επικύρωση δεδομένων εισόδου στα API endpoints γίνεται μέσω κλάσεων Form Request, οι οποίες τυποποιούν τους κανόνες επικύρωσης ανεξάρτητα από τον controller. Η πλατφόρμα ορίζει δύο βασικά Form Requests για τα API συσκευών:

DeviceDataRequest.

Επικυρώνει τα δεδομένα μετρήσεων που αποστέλλονται μέσω HTTP:

```
1 public function rules(): array  
2 {  
3     return [  
4         'sensors' => ['required', 'array', 'min:1'],  
5         'sensors.*.uuid' => ['required', 'string', 'max:64'],  
6         'sensors.*.value' => ['required', 'numeric'],  
7     ];  
8 }
```

Listing 20: Κανόνες επικύρωσης — DeviceDataRequest

LoRaWebhookRequest.

Επικυρώνει το EMQX envelope που φέρει τα LoRa δεδομένα, υποστηρίζοντας δύο μορφές (wrapped και direct):

```
1 public function rules(): array
```

```
2 {  
3     return [  
4         'request' => ['sometimes', 'array'],  
5         'request.username' => ['required_with:request', 'string'],  
6         'request.payload' => ['required_with:request', 'string'],  
7         'username' => ['required_without:request', 'string'],  
8         'payload' => ['required_without:request', 'string'],  
9     ];  
10 }
```

Listing 21: Κανόνες επικύρωσης — LoRaWebhookRequest

Η διπλή δομή κανόνων (`required_with` / `required_without`) εξασφαλίζει συμβατότητα με διαφορετικές ρυθμίσεις του EMQX Rule Engine, ο οποίος μπορεί να αποστέλλεται δεδομένα είτε σε wrapper αντικείμενο (`request. {}`) είτε απευθείας στο ριζικό επίπεδο.

5.5 Κεντρική Υπηρεσία Δεδομένων

Στην Ενότητα 4.4 του Κεφαλαίου 4 παρουσιάστηκε η αρχιτεκτονική επικοινωνίας και τα τρία μονοπάτια εισόδου δεδομένων (WiFi HTTP, MQTT webhook, LoRa webhook). Ανεξαρτήτως μονοπατιού, κάθε μέτρηση καταλήγει στην κλάση `SensorDataService`, η οποία και είναι το μοναδικό σημείο εισόδου (`single entry point`) για την επεξεργασία δεδομένων αισθητήρων. Η ενοποιημένη αυτή αρχιτεκτονική, η οποία απεικονίζεται στο Σχήμα 18, εξασφαλίζει ομοιόμορφη συμπεριφορά σε ολόκληρο το σύστημα — εγγραφή χρονοσειρών, ενημέρωση σχεσιακών μοντέλων και εκπομπή γεγονότων πραγματικού χρόνου — χωρίς αντιγραφή λογικής μεταξύ `controllers`.

5.5.1 Η μέθοδος `processSensorData()`

Η μέθοδος `processSensorData()` δέχεται ένα μοντέλο συσκευής και έναν πίνακα ωφέλιμων φορτίων αισθητήρων (ζεύγη `uuid/value`) και εκτελεί τα ακόλουθα βήματα:

1. **Αντιστοίχιση αισθητήρων:** Εφαρμογή `allTenants()` ώστε η αναζήτηση να γίνεται ανεξαρτήτως `tenant scope`, φιλτράροντας κατά `device_id` και `uuid`. Τα αποτελέσματα δεικτοδοτούνται κατά UUID μέσω της `keyBy()`.
2. **Ασύγχρονη εγγραφή InfluxDB:** Για κάθε αναγνωρισμένο αισθητήρα, αποστέλλεται εργασία `ProcessSensorInfluxData` στην ουρά `sensor-data`.

3. **Ενημέρωση σχεσιακού μοντέλου:** Τα πεδία `last_reading` και `last_reading_at` ενημερώνονται σύγχρονα στην MySQL, παρέχοντας άμεση πρόσβαση στην τελευταία τιμή χωρίς ερώτημα στην InfluxDB.
4. **Εκπομπή γεγονότος:** Κλήση `event(new SensorReadingEvent(...))` για broadcast σε ιδιωτικό κανάλι WebSocket (αναλύεται στην Ενότητα 5.6).
5. **Αναφορά σφαλμάτων:** Τα UUID που δεν αντιστοιχίστηκαν σε εγγεγραμμένο αισθητήρα επιστρέφονται στον κλήσαντα ελεγκτή ως `missing_uuids`, επιτρέποντας διαγνωστικά στο επίπεδο API.

```
1 public function processSensorData($device, array $sensorPayloads)
2     : array
3     {
4         $sensors = Sensor::allTenants()
5             →where('device_id', $device→id)
6             →whereIn('uuid', collect($sensorPayloads)→pluck('uuid')→
7                 all())
8             →get()
9             →keyBy('uuid');
10
11         foreach ($sensorPayloads as $sensor) {
12             $sensorModel = $sensors→get($sensor['uuid']);
13             if (! $sensorModel) { $missingUuids[] = $sensor['uuid'];
14                 continue; }
15
16             // 1. Async InfluxDB write via queue
17             ProcessSensorInfluxData::dispatch($sensorModel, (float)
18                 $sensor['value'], time());
19
20             // 2. Sync last-reading update in MySQL
21             $sensorModel→last_reading = $sensor['value'];
22             $sensorModel→last_reading_at = now();
23             $sensorModel→save();
24         }
25     }
```

```
21 // 3. WebSocket broadcast
22 event(new SensorReadingEvent($sensorModel→id, [
23     'value' ⇒ $sensorModel→last_reading,
24     'time' ⇒ $sensorModel→last_reading_at→
25         toDateTimeString(),
26 ]));
27 }
28 return ['message' ⇒ 'Data received.'] +
29     (count($missingUuids ?? []) ? ['missing_uuids' ⇒
30         $missingUuids] : []);
31 }
```

Listing 22: Κεντρική μέθοδος επεξεργασίας δεδομένων — SensorDataService

Η χρήση `allTenants()` είναι σκόπιμη: τα API requests από συσκευές δεν ανήκουν σε χρήση session, επομένως ο global scope πολυμίσθωσης (§4.3) παρακάμπτεται ρητά, ενώ η απομόνωση εξασφαλίζεται μέσω του `device_id` — η συσκευή έχει ήδη αυθεντικοποιηθεί και ανήκει σε συγκεκριμένο tenant.

5.5.2 Εργασία Ουράς

Η εγγραφή μετρήσεων στην InfluxDB εκτελείται ασύγχρονα μέσω εργασίας ουράς (queued job), ώστε ο χρόνος απόκρισης του API να μην εξαρτάται από τη διαθεσιμότητα ή τη καθυστέρηση της βάσης χρονοσειρών. Η κλάση `ProcessSensorInfluxData` υλοποιεί τη διεπαφή `ShouldQueue` και δρομολογείται στην αποκλειστική ουρά `sensor-data`:

```
1 class ProcessSensorInfluxData implements ShouldQueue
2 {
3     use InteractsWithQueue, Queueable, SerializesModels;
4
5     public int $tries = 3;
6     public int $timeout = 30;
7
8     public function __construct(
9         public Sensor $sensor,
10         public float $value,
11         public int $timestamp
```

```
12     ) {
13         $this->onQueue('sensor-data');
14     }
15
16     public function handle(InfluxDBService $influx): void
17     {
18         $payload = SensorMeasurementPayloadFactory::make($this->
19             sensor, $this->value);
19         if ($this->timestamp) {
20             $payload['time'] = $this->timestamp;
21         }
22         $influx->writeArray($payload);
23     }
24 }
```

Listing 23: Εργασία ασύγχρονης εγγραφής InfluxDB

Μηχανισμός αξιοπιστίας.

Η ιδιότητα `$tries = 3` ορίζει μέγιστο αριθμό τριών επαναλήψεων σε περίπτωση αποτυχίας, ενώ το `$timeout = 30` (δευτερόλεπτα) αποτρέπει την αιώνια αναμονή λόγω δικτυακού προβλήματος. Σε οριστική αποτυχία (εξάντληση των 3 προσπαθειών), η μέθοδος `failed()` καταγράφει αναλυτικά μηνύματα στο σύστημα καταγραφής (logging), ώστε να είναι δυνατή η διάγνωση αργότερα.

5.5.3 Κατασκευή Ωφέλιμου Φορτίου

Πριν από κάθε εγγραφή, η στατική μέθοδος `SensorMeasurementPayloadFactory::make()` μετατρέπει τη μέτρηση σε δομή συμβατή με την InfluxDB:

```
1 public static function make(Sensor $sensorModel, $value): array
2 {
3     return [
4         'name'    => 'sensor_measurement',
5         'tags'    => [
6             'user_id'    => $sensorModel->user_id,
7             'farm_id'    => $sensorModel->farm_id,
8             'sensor_id'  => $sensorModel->id,
```

```
9         'sensor_type' => $sensorModel->type ,
10     ],
11     'fields' => ['value' => (float) $value],
12     'time'   => time() ,
13 ];
14 }
```

Listing 24: Κατασκευή ωφέλιμου φορτίου — SensorMeasurementPayloadFactory

Τα πεδία tags δεικτοδοτούνται αυτόματα από την InfluxDB και εξυπηρετούν διπλό σκοπό: αφενός διευκολύνουν τα ερωτήματα φιλτραρίσματος (*sensor_id*, *sensor_type*), αφετέρου εφαρμόζουν την απομόνωση πολυμίσθωσης μέσω *user_id* και *farm_id*, όπως περιγράφηκε στον Πίνακα 5. Η τιμή μετατρέπεται πάντα σε *float* ώστε να αποφευχθεί σύγκρουση τύπου στήλης (*integer/float column conflict*) στην InfluxDB, πρόβλημα που εμφανίζεται όταν η πρώτη εγγραφή δημιουργεί στήλη ακέραιου τύπου και επόμενες εγγραφές αποστέλλουν δεκαδικές τιμές.

5.5.4 Υπηρεσία Σύνδεσης InfluxDB

Η κλάση *InfluxDBService* ενθυλακώνει τη σύνδεση με τον InfluxDB server μέσω της επίσημης βιβλιοθήκης *influxdb2-php*. Η αρχικοποίηση γίνεται στον κατασκευαστή, διαβάζοντας τα απαιτούμενα στοιχεία (*url*, *token*, *bucket*, *org*) από τις μεταβλητές ρύθμισης (`config('services.influxdb.*')`):

```
1 public function writeArray(array $data, $precision =
    WritePrecision::S)
2 {
3     $writeApi = $this->client->createWriteApi();
4     $writeApi->write($data, $precision, $this->bucket, $this->org);
5 }
```

Listing 25: Εγγραφή array στην InfluxDB

Η μέθοδος *writeArray()* καλείται από τον job *ProcessSensorInfluxData* και δέχεται τη δομή *name/tags/fields/time* που κατασκευάζει ο *PayloadFactory*. Η ακρίβεια χρονοσήμανσης (*precision*) ορίζεται σε δευτερόλεπτα (*WritePrecision::S*), επαρκής για τον ρυθμό δειγματοληψίας γεωργικών αισθητήρων. Η υπηρεσία παρέχει επίσης τη μέθοδο *queryPipeline()*, η οποία χρησιμοποιείται από τις υπηρεσίες χρονοσειρών για ερω-

τήματα Flux — προσαρτά αυτόματα τον bucket στο ερώτημα, μειώνοντας τον κίνδυνο σφαλμάτων ρύθμισης.

Ο πλήρης κώδικας της `SensorDataService` παρατίθεται στο Παράρτημα Α' (Listing 82), η εργασία ουράς `ProcessSensorInfluxData` στο Listing 84, ο `SensorMeasurementPayloadFactory` στο Listing 85 και η `InfluxDBService` στο Listing 86.

5.6 Επικοινωνία Πραγματικού Χρόνου

Στην Ενότητα 4.4 του Κεφαλαίου 4 περιγράφηκε η αρχιτεκτονική ειδοποιήσεων πραγματικού χρόνου σε επίπεδο σχεδίασης: κάθε μέτρηση που περνά από τη `SensorDataService` (§5.5) εκπέμπεται ως γεγονός σε ιδιωτικό κανάλι `WebSocket` μέσω του `Laravel Reverb`. Στο σημείο αυτό περιγράφεται η υλοποίηση αυτής της αρχιτεκτονικής, από τον ορισμό του γεγονότος στον server έως την κατανάλωσή του στα `Vue components` του client.

5.6.1 Γεγονός `SensorReadingEvent`

Η κλάση `SensorReadingEvent` υλοποιεί τη διεπαφή `ShouldBroadcast`, γεγονός που σηματοδοτεί στο `Laravel` ότι πρέπει να προωθηθεί στον broadcast driver (στην περίπτωσή μας, τον `Reverb WebSocket server`):

```
1 class SensorReadingEvent implements ShouldBroadcast
2 {
3     use Dispatchable, InteractsWithSockets, SerializesModels;
4
5     public function __construct(
6         public int $sensorId,
7         public array $payload = []
8     ) {}
9
10    public function broadcastOn(): PrivateChannel
11    {
12        return new PrivateChannel("sensor.{$this->sensorId}");
13    }
14
15    public function broadcastWith(): array
16    {
17        return $this->payload;
```

```
18     }  
19 }
```

Listing 26: Γεγονός εκπομπής μέτρησης αισθητήρα

5.6.2 Εξουσιοδότηση Καναλιών

Η πρόσβαση στα ιδιωτικά κανάλια ελέγχεται στο αρχείο `routes/channels.php`. Κατά τη σύνδεση, ο Reverb server εκτελεί HTTP callback στο Laravel για επαλήθευση της εξουσιοδότησης:

```
1 Broadcast::channel('sensor.{sensorId}', function ($user,  
    $sensorId) {  
2     $sensor = Sensor::find($sensorId);  
3     if (! $sensor) {  
4         return false;  
5     }  
6     // Allow if the authenticated user owns the sensor  
7     return (int) $sensor->user_id === (int) $user->id;  
8 });
```

Listing 27: Εξουσιοδότηση ιδιωτικού καναλιού αισθητήρα

Ο κανόνας εξουσιοδότησης εφαρμόζει τη λογική πολυμίσθωσης (§4.3) στο επίπεδο WebSocket: μόνο ο ιδιοκτήτης του αισθητήρα μπορεί να εγγραφεί στο αντίστοιχο κανάλι. Η αναζήτηση `Sensor::find()` γίνεται μέσω του κανονικού global scope, χωρίς `allTenants()`, καθώς εδώ υπάρχει αυθεντικοποιημένος χρήστης (session-based) και ο tenant scope λειτουργεί κανονικά.

5.6.3 Αρχικοποίηση Laravel Echo και Reverb

Στην πλευρά του client, η σύνδεση με τον WebSocket server αρχικοποιείται στο σημείο εκκίνησης της εφαρμογής (`app.ts`) μέσω του πακέτου `@laravel/echo-vue`:

```
1 import { configureEcho } from '@laravel/echo-vue';  
2  
3 configureEcho({  
4     broadcaster: 'reverb',  
5 });
```

Listing 28: Αρχικοποίηση Echo στο σημείο εκκίνησης

Η ρύθμιση `broadcaster`: `'reverb'` ενεργοποιεί τον Reverb driver, ο οποίος αυτόματα χρησιμοποιεί τις μεταβλητές περιβάλλοντος `VITE_REVERB_*` (διεύθυνση, θύρα, κλειδί εφαρμογής) που ορίζονται στο `.env`. Η αρχικοποίηση εκτελείται μία φορά κατά τη φόρτωση της εφαρμογής, ενώ κάθε component μπορεί στη συνέχεια να εγγραφεί σε κανάλια μέσω του composable `useEcho()`.

5.6.4 Εγγραφή Components σε Κανάλια

Τα Vue components εγγράφονται σε ιδιωτικά κανάλια μέσω του composable `useEcho()`, που παρέχεται από το πακέτο `@laravel/echo-vue`. Η εγγραφή είναι δηλωτική (declarative) — ενεργοποιείται κατά τη δημιουργία του component και απελευθερώνεται αυτόματα κατά την καταστροφή του, αποφεύγοντας διαρροές μνήμης.

SensorCard.vue — *Κάρτα αισθητήρα στη λίστα.*

Κάθε κάρτα διατηρεί τοπικές αντιδραστικές μεταβλητές (`ref()`) για την τελευταία τιμή και χρονοσήμανση, οι οποίες αρχικοποιούνται από τα props και ενημερώνονται σε πραγματικό χρόνο:

```
1 const lastReading = ref(props.sensor.last_reading ?? null);
2 const lastSeen = ref(props.sensor.last_reading_at ?? null);
3
4 // Sync if parent props change (e.g. Inertia page reload)
5 watch(() => props.sensor.last_reading, (v) => { lastReading.value =
    v ?? null; });
6 watch(() => props.sensor.last_reading_at, (v) => { lastSeen.value =
    v ?? null; });
7
8 // Subscribe to private sensor channel
9 useEcho('sensor.${props.sensor.id}', 'SensorReadingEvent', (
    payload) => {
10     if (payload?.value !== undefined) lastReading.value = payload
        .value;
11     if (payload?.time) lastSeen.value = payload.time;
12 });
```

Listing 29: Εγγραφή σε κανάλι — SensorCard.vue

Η χρήση `watch()` επιτρέπει στα τοπικά `ref()` να συγχρονίζονται με αλλαγές props (π.χ. κατά επαναφόρτωση σελίδας μέσω `Inertia`), ενώ η `useEcho()` ενημερώνει τις ίδιες μεταβλητές χωρίς καμία αλληλεπίδραση χρήστη.

Sensors/Show.vue — Σελίδα λεπτομερειών αισθητήρα.

Στη σελίδα λεπτομερειών, η λογική real-time είναι πιο σύνθετη, καθώς εκτός από την ενημέρωση της τελευταίας τιμής, η νέα μέτρηση εισάγεται στον πίνακα πρόσφατων μετρήσεων:

```
1 const recentReadings = ref(props.recentReadings ?? []);
2
3 useEcho('sensor.${sensor.value.id}', 'SensorReadingEvent', (
4   payload) => {
5     if (payload.time && payload.value !== undefined) {
6       // Prepend to recent readings table (cap at 50 entries)
7       const arr = recentReadings.value.slice();
8       arr.unshift({ time: payload.time, value: payload.value })
9       ;
10      if (arr.length > 50) arr.pop();
11      recentReadings.value = arr;
12
13      // Update sensor last reading
14      sensor.value.last_reading = payload.value;
15      sensor.value.last_reading_at = payload.time;
16    }
17  });
```

Listing 30: Ενσωμάτωση real-time μετρήσεων — Sensors/Show.vue

Η εισαγωγή γίνεται στην αρχή του πίνακα (`unshift`), ενώ ο περιορισμός σε 50 εγγραφές (`pop`) αποτρέπει τη συσσώρευση μνήμης σε μακροχρόνιες συνεδρίες. Η ενημέρωση του αντιδραστικού `sensor.value` προκαλεί αυτόματη επανασχεδίαση (`re-render`) της κάρτας τελευταίας μέτρησης.

Τα δεδομένα γραφήματος (chart data) δεν ενημερώνονται μέσω WebSocket — φορτώνονται ως deferred prop (§5.3.4) κατά τη φόρτωση της σελίδας και περιέχουν ήδη συγκεντρωτικά δεδομένα (aggregated values) από την InfluxDB. Η προσθήκη μεμονωμένων μη συγκεντρωτικών σημείων στο γράφημα θα αλλοίωνε την κλίμακα των δεδομένων.

5.7 Ασφάλεια Εφαρμογής

Η ασφάλεια της πλατφόρμας Agronos βασίζεται σε τρεις αλληλοσυμπληρούμενους μηχανισμούς: **πολιτικές εξουσιοδότησης** (Authorization Policies) που ελέγχουν την πρόσβαση σε πόρους, **κλάσεις επικύρωσης** (Form Requests) που εξασφαλίζουν την ακεραιότητα των δεδομένων εισόδου, και **αυθεντικοποίηση μέσω tokens** (Laravel Sanctum) για τις συσκευές IoT. Η παρούσα ενότητα παρουσιάζει την υλοποίηση αυτών των μηχανισμών, ενώ η αρχιτεκτονική ασφάλειας σε επίπεδο επικοινωνίας (κρυπτογράφηση LoRa, anti-replay) καλύφθηκε στην Ενότητα 5.4.4.

5.7.1 Πολιτικές Εξουσιοδότησης

Το Laravel παρέχει τον μηχανισμό Policies — κλάσεις που αντιστοιχίζονται σε μοντέλα Eloquent και ορίζουν ποιες ενέργειες επιτρέπονται σε κάθε χρήστη. Η πλατφόρμα ορίζει τρεις πολιτικές: FarmPolicy, DevicePolicy και SensorPolicy, οι οποίες ακολουθούν κοινή δομή.

Μοτίβο ιδιοκτησίας.

Κάθε πολιτική εφαρμόζει τον ίδιο θεμελιώδη κανόνα: ο αυθεντικοποιημένος χρήστης επιτρέπεται να προσπελάσει έναν πόρο **μόνο εάν τον κατέχει** (σύγκριση user_id). Οι ενέργειες viewAny και create επιτρέπονται πάντα, καθώς δεν αφορούν συγκεκριμένη εγγραφή:

```
1 class FarmPolicy
2 {
3     use HandlesAuthorization;
4
5     public function viewAny(User $user): bool { return true; }
6
7     public function view(User $user, Farm $farm): bool
8     {
9         return $user->id === $farm->user_id;
```

```
10     }
11
12     public function update(User $user, Farm $farm): bool
13     {
14         return $user->id === $farm->user_id;
15     }
16
17     public function delete(User $user, Farm $farm): bool
18     {
19         return $user->id === $farm->user_id;
20     }
21 }
```

Listing 31: Πολιτική εξουσιοδότησης — FarmPolicy (απόσπασμα)

Οι DevicePolicy και SensorPolicy ακολουθούν πανομοιότυπη δομή, διαφοροποιούμενες μόνο στον τύπο μοντέλου.

Κλήση στους controllers.

Κάθε ελεγκτής χρησιμοποιεί το trait `AuthorizesRequests` και καλεί τη μέθοδο `$this->authorize()` πριν από κάθε ενέργεια. Σε περίπτωση αποτυχίας, το Laravel διακόπτει αυτόματα την εκτέλεση με κωδικό HTTP 403:

```
1 class FarmController extends Controller
2 {
3     use AuthorizesRequests;
4
5     public function index()
6     {
7         $this->authorize('viewAny', Farm::class);
8         // ...
9     }
10
11     public function show(Farm $farm)
12     {
13         $this->authorize('view', $farm);
```

```
14      // ...
15  }
16
17  public function destroy(Farm $farm)
18  {
19      $this->authorize('delete', $farm);
20      // ...
21  }
22 }
```

Listing 32: Κλήσεις εξουσιοδότησης στον FarmController (απόσπασμα)

Σχέση με πολυμίσθωση.

Οι πολιτικές λειτουργούν συμπληρωματικά με τον μηχανισμό πολυμίσθωσης (§4.3): ο TenantScope φιλτράρει αυτόματα τα ερωτήματα Eloquent ώστε να επιστρέφονται μόνο εγγραφές του τρέχοντος χρήστη, ενώ η Policy χρησιμεύει ως **δεύτερη γραμμή άμυνας** — ακόμη και αν μια εγγραφή αποκαλυφθεί μέσω route model binding, η πολιτική θα απορρίψει την πρόσβαση εάν ο χρήστης δεν είναι ο ιδιοκτήτης.

5.7.2 Επικύρωση Δεδομένων Εισόδου

Η επικύρωση δεδομένων στους ελεγκτές web εκτελείται μέσω κλάσεων Form Request, οι οποίες τυποποιούν τους κανόνες ανεξάρτητα από τη λογική του ελεγκτή. Η πλατφόρμα χρησιμοποιεί δύο μοτίβα, ανάλογα με την πολυπλοκότητα του πόρου.

Μοτίβο 1: Ενιαία κλάση.

Για πόρους με απλούς κανόνες, χρησιμοποιείται μία κλάση Form Request τόσο για τη δημιουργία όσο και για την ενημέρωση. Χαρακτηριστικό παράδειγμα είναι η FarmRequest:

```
1  class FarmRequest extends FormRequest
2  {
3      public function rules(): array
4      {
5          return [
6              'name'          => ['required', 'string', 'max:255'],
7              'location'      => ['required', 'string', 'max:255'],
8              'size'          => ['nullable', 'numeric', 'min:0'],
```



```
9         'coordinates' => ['nullable', 'array'],
10        'description' => ['nullable', 'string'],
11    ];
12    }
13 }
```

Listing 33: Κανόνες επικύρωσης — FarmRequest

Μοτίβο 2: Αφηρημένη βασική κλάση με εξειδικεύσεις.

Για πόρους με διαφορετικούς κανόνες ανά ενέργεια, χρησιμοποιείται μια αφηρημένη κλάση που ορίζει κοινούς κανόνες (`commonRules()`) και εξειδικευμένες υποκλάσεις:

```
1  abstract class SensorRequest extends FormRequest
2  {
3      protected function commonRules(): array
4      {
5          return [
6              'name'      => ['nullable', 'string', 'max:255'],
7              'farm_id'   => ['nullable', 'exists:farms,id'],
8              'lat'       => ['nullable', 'numeric', 'between:-90,90'],
9              'lon'       => ['nullable', 'numeric', 'between:-180,180'],
10             'type'      => ['nullable', Rule::in(SensorType::values()),],
11         ];
12     }
13 }
14
15 class StoreSensorRequest extends SensorRequest
16 {
17     public function rules(): array
18     {
19         return array_merge($this->commonRules(), [
20             'device_uuid' => ['required', 'exists:devices,uuid'],
```

```
21         'uuid'           ⇒ ['required', 'string', 'unique:  
                               sensors,uuid'],  
22     ] );  
23 }  
24 }
```

Listing 34: Ιεραρχία επικύρωσης αισθητήρων — αφηρημένη βάση και StoreSensorRequest

Η `StoreSensorRequest` απαιτεί `device_uuid` και μοναδικό `uuid`, ενώ η αντίστοιχη `UpdateSensorRequest` χρησιμοποιεί μόνο τους κοινούς κανόνες. Μια τρίτη υποκλάση, `ScanSensorRequest`, εξυπηρετεί τη ροή σάρωσης QR code (§5.3.3).

Επικύρωση εγγραφής συσκευής.

Η `RegisterDeviceRequest` αξιοποιεί δυναμικούς κανόνες βάσει τύπου συσκευής: το πεδίο `lora_aes_key` είναι υποχρεωτικό μόνο όταν `type = lora`, μέσω του κανόνα `required_if`:

```
1 public function rules(): array  
2 {  
3     return [  
4         'uuid'           ⇒ 'required|string|unique:devices,uuid',  
5         'secret'         ⇒ 'required|string',  
6         'type'           ⇒ 'required|string|in:' . implode(',',  
                               DeviceType::values()),  
7         'lora_aes_key' ⇒ 'nullable|string|size:32|required_if:  
                               type,lora',  
8     ] ;  
9 }
```

Listing 35: Δυναμικοί κανόνες επικύρωσης — RegisterDeviceRequest

Η κλάση αυτή υπερκαλύπτει επίσης τη μέθοδο `failedValidation()` ώστε σε αποτυχία να επιστρέφει δομημένη απάντηση JSON (κωδικός 422), αντί της συνήθους ανακατεύθυνσης web — αναγκαία προσαρμογή, καθώς ο πελάτης σε αυτή την περίπτωση είναι ενσωματωμένη συσκευή και όχι φυλλομετρητής.

5.7.3 Σύνοψη Μηχανισμών Ασφαλείας

Ο Πίνακας 9 συνοψίζει τα επίπεδα ασφαλείας που εφαρμόζονται σε κάθε κατηγορία αιτήματος:

Κατηγορία αιτήματος	Auth	Policy	Validation	Tenant Scope
Web (χρήστης)	Session	✓	Form Request	✓
API (συσκευή)	Sanctum	—	Form Request	<code>allTenants()</code>
MQTT webhook	MQTT Credentials	—	Validator	<code>allTenants()</code>
LoRa webhook	Shared secret + AES	—	Validator	<code>allTenants()</code>
WebSocket	Session	—	—	<code>channels.php</code>

Πίνακας 9: Επίπεδα ασφαλείας ανά κατηγορία αιτήματος

Η αρχιτεκτονική εφαρμόζει την αρχή της *άμυνας σε βάθος* (defense in depth): κανένα μεμονωμένο επίπεδο δεν αρκεί για πλήρη ασφάλεια, αλλά ο συνδυασμός αυθεντικοποίησης, εξουσιοδότησης, επικύρωσης και αυτόματου φιλτραρίσματος πολυμίσθωσης παρέχει ολοκληρωμένη κάλυψη σε όλα τα μονοπάτια εισόδου δεδομένων.

5.8 Σύνοψη

Το παρόν κεφάλαιο κάλυψε την πλήρη υλοποίηση του backend και της διαδικτυακής εφαρμογής της πλατφόρμας Agronos. Επιλέχθηκε το Laravel ως κεντρικό πλαίσιο ανάπτυξης και το Inertia.js ως γέφυρα ενσωμάτωσης με το Vue.js frontend, εξαλείφοντας την πολυπλοκότητα ενός ανεξάρτητου SPA.

Το επίπεδο δεδομένων βασίζεται στα Eloquent μοντέλα Farm, Device και Sensor, ενισχυμένα με το trait `BelongsToTenant` για αυτόματη απομόνωση πολυμίσθωσης, και αξιοποιεί υβριδική αποθήκευση σε MySQL (δομημένα δεδομένα) και InfluxDB (χρονοσειρές μετρήσεων). Η κεντρική υπηρεσία `SensorDataService` αποτελεί το μοναδικό σημείο εισόδου για δεδομένα αισθητήρων από όλα τα μονοπάτια εισόδου (WiFi HTTP, MQTT webhook, LoRa webhook), εξασφαλίζοντας ομοιόμορφη εγγραφή στην InfluxDB, ενημέρωση της σχεσιακής βάσης και εκπομπή γεγονότων πραγματικού χρόνου μέσω WebSocket.

Η ασφάλεια υλοποιείται σε πολλαπλά επίπεδα: αυθεντικοποίηση μέσω Laravel Sanctum για τις IoT συσκευές, πολιτικές εξουσιοδότησης (Policies) για τους χρήστες, επικύρωση εισόδου μέσω Form Requests και κρυπτογραφική προστασία AES-128-CTR για τα LoRa δεδομένα. Το επόμενο κεφάλαιο παρουσιάζει την υλοποίηση του firmware των IoT κόμβων.

6 Υλοποίηση Υλικολογισμικού

Ο πλήρης πηγαίος κώδικας του *firmware* είναι δημοσίως διαθέσιμος στο αποθετήριο *GitHub*: <https://github.com/gkalogeitonas/Agronos-iot-device>.

Το παρόν κεφάλαιο παρουσιάζει την υλοποίηση του υλικολογισμικού των κόμβων της πλατφόρμας *Agronos*. Εδώ υλοποιούμε την αρχιτεκτονική που περιγράφεται στο Κεφάλαιο 4. Παρουσιάζονται οι επιλογές πλατφόρμας, το περιβάλλον ανάπτυξης, τα βασικά σχεδιαστικά πρότυπα και οι τρεις παραλλαγές του *firmware* (WiFi, LoRa, Gateway).

6.1 Εισαγωγή και Επιλογή Τεχνολογιών

Το υλικολογισμικό (*firmware*) εκτελείται στους κόμβους πεδίου της πλατφόρμας *Agronos*. Είναι υπεύθυνο για την ανάγνωση των αισθητήρων, τη σύνδεση στο δίκτυο, την αυθεντικοποίηση με το *backend* και την αξιόπιστη αποστολή δεδομένων. Η σωστή αρχιτεκτονική σχεδίαση του *firmware* είναι κρίσιμη, καθώς οι κόμβοι λειτουργούν αυτόνομα στο πεδίο με περιορισμένους υπολογιστικούς πόρους και ενέργεια.

6.1.1 Πλατφόρμα Υλικού: ESP32

Ως μικροελεγκτής επιλέχθηκε η οικογένεια ESP32 της Espressif Systems. Η πλατφόρμα υποστηρίζει ενσωματωμένη συνδεσιμότητα WiFi και Bluetooth, διαθέτει επαρκή υπολογιστική ισχύ (διπύρηνος Xtensa LX6 στα 240 MHz ή RISC-V στα 160 MHz) και παρέχει πλούσιο σύνολο περιφερειακών (ADC, I2C, SPI, GPIO).

Η πλατφόρμα *Agronos* υποστηρίζει τρεις διαφορετικές πλακέτες:

- **ESP32 DevKit**: Η κλασική αναπτυξιακή πλακέτα, κατάλληλη για πρωτοτυποποίηση και δοκιμές.
- **DFRobot FireBeetle 2 ESP32-C6**: Πλακέτα βασισμένη σε αρχιτεκτονική RISC-V με διπλό πυρήνα — έναν υψηλής απόδοσης (HP) και έναν χαμηλής κατανάλωσης (LP) [46] — ενσωματωμένο κύκλωμα διαχείρισης ενέργειας (CN3165), υποδοχή ηλιακού πάνελ και μπαταρίας Li-Ion. Ο ESP32-C6 καταναλώνει μόλις 7 μ A σε Deep Sleep σύμφωνα με το *datasheet* [46], ενώ σε επίπεδο πλακέτας η συνολική κατανάλωση ανέρχεται σε περίπου 16 μ A λόγω των περιφερειακών κυκλωμάτων της πλακέτας (voltage regulator, USB bridge κ.λπ.).
- **TTGO LoRa32 V2.1**: Πλακέτα με ενσωματωμένο LoRa πομποδέκτη (SX1276, 868 MHz),

κατάλληλη για κόμβους μεγάλης εμβέλειας χωρίς WiFi.

6.1.2 Περιβάλλον Ανάπτυξης: PlatformIO

Η ανάπτυξη του firmware γίνεται μέσω του PlatformIO, ενός ανοικτού συστήματος αυτοματοποίησης μεταγλώττισης (build system) για ενσωματωμένα συστήματα. Επιλέχθηκε έναντι του Arduino IDE για δύο λόγους που είναι κρίσιμοι στο συγκεκριμένο project:

1. **Δηλωτική διαχείριση εξαρτήσεων:** Οι βιβλιοθήκες ορίζονται στο αρχείο `platformio.ini` με πλήρη πινακοποίηση εκδόσεων (version pinning), εξασφαλίζοντας αναπαραγωγή builds.
2. **Υποστήριξη πολλαπλών στόχων (multi-target build):** Από την ίδια βάση κώδικα, ο προγραμματιστής μπορεί να μεταγλώττισει για διαφορετικές πλακέτες. Κάθε στόχος (environment) ορίζει τη δική του πλακέτα, τα build flags και τα φίλτρα αρχείων πηγαίου κώδικα.

```
1 [env]
2 framework = arduino
3 monitor_speed = 115200
4 lib_deps =
5     adafruit/Adafruit Unified Sensor@^1.1.5
6     adafruit/DHT sensor library@^1.4.5
7     bblanchon/ArduinoJson@^6.19.5
8     knolleary/PubSubClient@^2.8
9     robtillaart/DHT20 @ ^0.3.1
10
11 [env:esp32dev]
12 platform = espressif32
13 board = esp32dev
14 src_filter =
15     +<*>
16     -<main_lora.cpp>
17     -<lora_payload.cpp>
18     -<lora_crypto.cpp>
19     -<lora_fcmt.cpp>
```

```
20     -<lora_radio.cpp>
21
22 [env:esp32-c6-devkitc-1]
23 platform = espressif32
24 board = esp32-c6-devkitc-1
25 build_flags =
26     -D ARDUINO_USB_MODE=1
27     -D ARDUINO_USB_CDC_ON_BOOT=1
28 src_filter =
29     +<*>
30     -<main_lora.cpp>
31     -<lora_payload.cpp>
32     -<lora_crypto.cpp>
33     -<lora_fcmt.cpp>
34     -<lora_radio.cpp>
35
36 [env:ttgo-lora32-v21]
37 platform = espressif32
38 board = ttgo-lora32-v21
39 build_flags =
40     -D LORA_NODE=1
41 lib_deps =
42     ${env.lib_deps}
43     sandeepmistry/LoRa@^0.8.0
44 src_filter =
45     +<*>
46     -<main.cpp>
47     -<wifi_portal.cpp>
48     -<auth.cpp>
49     -<data_sender.cpp>
50     -<mqtt_client.cpp>
51
52 [env:ttgo-lora32-v21-wifi]
53 platform = espressif32
```

```
54 board = ttgo-lora32-v21
55 src_filter =
56     +<*>
57     -<main_lora.cpp>
58     -<lora_payload.cpp>
59     -<lora_crypto.cpp>
60     -<lora_fcmt.cpp>
61     -<lora_radio.cpp>
```

Listing 36: Απόσπασμα του platformio.ini — κοινές ρυθμίσεις και τρεις στόχοι μεταγλώττισης

Η χρήση φίλτρων πηγαίου κώδικα (`src_filter`) αντί οδηγιών προεπεξεργαστή (`#ifdef`) αποκλείει τη μεταγλώττιση των WiFi-specific αρχείων στον στόχο LoRa και αντίστροφα.

6.1.3 Γλώσσα Προγραμματισμού

Το firmware αναπτύσσεται σε C++ με το πλαίσιο Arduino. Η επιλογή της C++ — σε σχέση με την καθαρή C — επιτρέπει τη χρήση αντικειμενοστραφών σχεδιαστικών προτύπων (κληρονομικότητα, πολυμορφισμός, templates) που αναβαθμίζουν σημαντικά την επεκτασιμότητα του κώδικα.

6.2 Αρχιτεκτονική Λογισμικού

Η σχεδίαση του υλικολογισμικού των IoT κόμβων της πλατφόρμας Agropos βασίστηκε στις αρχές της αντικειμενοστραφούς προγραμματισμού. Η υιοθέτηση καθιερωμένων σχεδιαστικών προτύπων κρίθηκε επιβεβλημένη για την εξασφάλιση της ευελιξίας και της εύκολης επεκτασιμότητας του συστήματος.

6.2.1 Επίπεδο Αφαίρεσης Αισθητήρων

Για τη διαχείριση της ετερογένειας των φυσικών αισθητήρων (π.χ. ψηφιακοί αισθητήρες θερμοκρασίας, αναλογικοί αισθητήρες υγρασίας εδάφους), εφαρμόστηκε το Strategy Pattern [47]. Η κεντρική ιδέα αυτού του προτύπου είναι ο ορισμός μιας κοινής διεπαφής για μια οικογένεια αλγορίθμων ή, στην προκειμένη περίπτωση, συσκευών.

Στο παρόν σύστημα, όλοι οι αισθητήρες κληρονομούν από την αφηρημένη βασική κλάση `SensorBase`, η οποία επιβάλλει την υλοποίηση της μεθόδου ανάγνωσης (`read()`). Αυτή η προσέγγιση (Sensor Abstraction Layer) αποσυνδέει (decouples) τη λογική ελέγχου

και μετάδοσης δεδομένων από τις ιδιαιτερότητες του κάθε hardware. Ο κεντρικός βρόχος εκτέλεσης (main loop) της συσκευής αλληλεπιδρά ομοιόμορφα με οποιονδήποτε τύπο αισθητήρα, αγνοώντας το υποκείμενο πρωτόκολλο επικοινωνίας (π.χ. I2C, SPI, Analog), επιτρέποντας την απρόσκοπτη κλήση της μεθόδου ανάγνωσης.

Στο Listing 37 παρουσιάζεται η δομή περιγραφής αισθητήρα `SensorConfig` και η αφηρημένη βασική κλάση `SensorBase`. Κάθε αισθητήρας περιγράφεται δηλωτικά μέσω ενός `SensorConfig` (τύπος, pin, UUID), ενώ η κλάση `SensorBase` ορίζει δύο αμιγώς ει-
κονικές μεθόδους: `uuid()` για την αναγνώριση και `read()` για την ανάγνωση τιμής.

```
1 struct SensorConfig {  
2     const char* type;           // e.g. "DHT11_TEMP", "  
    SoilMoistureSensor"  
3     int pin;                   // GPIO pin or -1 if not applicable  
4     const char* uuid;          // sensor UUID used by the server  
5     const char* displayName;   // human-readable name  
6 };  
7  
8 class SensorBase {  
9 public:  
10     virtual ~SensorBase() = default;  
11     virtual const char* uuid() const = 0;  
12     virtual bool read(float &out) = 0;  
13 };
```

Listing 37: Η δομή `SensorConfig` και η αφηρημένη κλάση `SensorBase` (sensor.h)

Στο Listing 38 φαίνεται ένα συγκεκριμένο παράδειγμα υλοποίησης: ο αισθητήρας θερμοκρασίας DHT11. Η κλάση κληρονομεί από τη `SensorBase` και υλοποιεί τη μέθοδο `read()` χρησιμοποιώντας τη βιβλιοθήκη DHT. Η τελευταία γραμμή — η κλήση `registerSensorFactory()` — αποτελεί τον μηχανισμό αυτο-καταχώρησης που αναλύεται στη συνέχεια.

```
1 class DHT11TemperatureReader : public SensorBase {  
2 public:  
3     DHT11TemperatureReader(int pin, const char* uid)  
4     : pin_(pin), uuid_(uid), dht_(getSharedDht(pin)) {}
```

```
5
6     const char* uuid() const override { return uuid_; }
7
8     bool read(float &out) override {
9         if (!dht_) return false;
10        float t = dht_→readTemperature();
11        if (isnan(t)) return false;
12        out = t;
13        return true;
14    }
15 private:
16     int pin_;
17     const char* uuid_;
18     DHT* dht_;
19 };
20
21 // Self-registration: executed at static initialization time
22 static bool _reg = registerSensorFactory(
23     "DHT11TemperatureReader",
24     create_sensor_impl<DHT11TemperatureReader>);
```

Listing 38: Υλοποίηση αισθητήρα DHT11 θερμοκρασίας με αυτο-καταχώρηση (dht11_temp.cpp)

6.2.2 Δυναμική Δημιουργία Αντικειμένων

Η αρχικοποίηση και δημιουργία των αντικειμένων των αισθητήρων υλοποιείται μέσω του Factory Pattern [47]. Αντί τα αντικείμενα των αισθητήρων να δημιουργούνται στατικά (hardcoded) στον κύριο κώδικα, η πλατφόρμα χρησιμοποιεί έναν κεντρικό μηχανισμό (SensorFactory) που τα δημιουργεί δυναμικά κατά την εκκίνηση της συσκευής.

Η λειτουργία βασίζεται σε μια κεντρική δομή ρυθμίσεων (config.h), όπου ορίζονται οι τύποι των αισθητήρων, τα συνδεδεμένα GPIO pins και τα μοναδικά τους αναγνωριστικά (UUIDs). Μέσω ενός μηχανισμού καταγραφής (registry mechanism), κάθε νέα κλάση αισθητήρα αυτο-καταχωρείται στο Factory. Αυτό σημαίνει ότι η προσθήκη υποστήριξης για έναν νέο αισθητήρα (π.χ. έναν αναλογικό αισθητήρα υγρασίας εδάφους) απαιτεί μόνο τη

δημιουργία της νέας κλάσης και την εγγραφή της στο registry, διατηρώντας τον κεντρικό κώδικα της εφαρμογής (factory) απόλυτα ανέπαφο.

Στο Listing 39 παρουσιάζεται ο πυρήνας του μηχανισμού. Η συνάρτηση `registerSensorFactory()` αποθηκεύει ζεύγη (όνομα, συνάρτηση δημιουργίας) σε ένα στατικό registry. Η `createSensors()` διατρέχει τον πίνακα `SENSOR_CONFIGS` και, για κάθε εγγραφή, αναζητά τον κατάλληλο δημιουργό (creator) στο registry.

Ο πλήρης κώδικας παρατίθεται στο Παράρτημα Β'.2.

```
1 static std::vector<std::pair<const char*, CreatorFunc>>& registry
   () {
2     static std::vector<std::pair<const char*, CreatorFunc>> r;
3     return r;
4 }
5
6 bool registerSensorFactory(const char* name, CreatorFunc creator)
   {
7     registry().emplace_back(name, creator);
8     return true;
9 }
10
11 std::vector<std::unique_ptr<SensorBase>> createSensors() {
12     std::vector<std::unique_ptr<SensorBase>> vec;
13     vec.reserve(SENSOR_CONFIG_COUNT);
14     for (size_t i = 0; i < SENSOR_CONFIG_COUNT; ++i) {
15         const SensorConfig &cfg = SENSOR_CONFIGS[i];
16         SensorBase* inst = createSensorByType(cfg.type, cfg);
17         if (!inst) continue;
18         vec.emplace_back(inst);
19     }
20     return vec;
21 }
```

Listing 39: Μηχανισμός registry και δημιουργία αισθητήρων (sensor_factory.cpp)

Στο Listing 40 φαίνεται πώς ο χρήστης ορίζει δηλωτικά τους αισθητήρες μιας συγκεκρι-

κριμένης συσκευής στο αρχείο `device_profile.h`. Για την προσθήκη ενός νέου αισθητήρα αρκεί μία νέα γραμμή στον πίνακα — χωρίς καμία τροποποίηση στον κώδικα του `factory`.

```
1 constexpr SensorConfig SENSOR_CONFIGS[] = {  
2     { "DHT11TemperatureReader", 21, "Sensor-UUID-1", "Temperature"  
3       },  
4     { "DHT11HumidityReader",    21, "Sensor-UUID-2", "Humidity"  
5       },  
6     { "SoilMoistureSensor",      32, "Sensor-UUID-3", "Soil  
7       Moisture"},  
8   };  
9  
10 constexpr size_t SENSOR_CONFIG_COUNT =  
11     sizeof(SENSOR_CONFIGS) / sizeof(SENSOR_CONFIGS[0]);
```

Listing 40: Δηλωτικός ορισμός αισθητήρων ανά συσκευή (`device_profile.h`)

6.2.3 Αρθρωτή Σχεδίαση και Διαχωρισμός Ευθυνών

Το υλικολογισμικό έχει διαχωριστεί σε διακριτά, ανεξάρτητα υποσυστήματα, εφαρμόζοντας την αρχή του διαχωρισμού ευθυνών (Separation of Concerns). Τα βασικά συστατικά περιλαμβάνουν:

- Τη διαχείριση της μόνιμης μνήμης (Storage / NVS).
- Την αυθεντικοποίηση (AuthManager).
- Τη δικτύωση και το Captive Portal (WifiPortal).
- Τη λογική κατασκευής και δρομολόγησης πακέτων (DataSender).

6.3 Υλοποίηση Αισθητήρων

Η ενότητα 6.2 παρουσίασε την αρχιτεκτονική της διεπαφής `SensorBase` και τον μηχανισμό αυτο-καταχώρησης εργοστασίου. Η παρούσα ενότητα περιγράφει τις συγκεκριμένες υλοποιήσεις αισθητήρων που είναι διαθέσιμες στην πλατφόρμα• όλες ακολουθούν το ίδιο μοτίβο Strategy (κληρονομιά από `SensorBase`, υλοποίηση της `read()`, αυτο-καταχώρηση στο εργοστάσιο). Ο Πίνακας 10 συγκεντρώνει τα κύρια χαρακτηριστικά τους. Ο πλήρης κώδικας κάθε υλοποίησης παρατίθεται στο Παράρτημα Β΄.3.

Πίνακας 10: Συνοπτικός πίνακας υλοποιημένων αισθητήρων

Κλάση	Αισθητήρας	Δίαυλος	Μετρούμενο μέγεθος
DHT11TemperatureReader	DHT11	1-Wire GPIO	Θερμοκρασία (°C)
DHT11HumidityReader	DHT11	1-Wire GPIO	Σχετική υγρασία (%)
DHT20TemperatureReader	DHT20	I ² C	Θερμοκρασία (°C)
DHT20HumidityReader	DHT20	I ² C	Σχετική υγρασία (%)
SoilMoistureSensor	SEN0193	ADC	Εδαφική υγρασία (%)
BatteryLevelSensor	Εσωτερικό ADC	ADC	Στάθμη μπαταρίας (%)
SimulatedSensor	—	—	Προσομοίωση (αύξων αριθμός)

6.3.1 Κοινοχρησία Στιγμοτύπων Υλικού

Μια βασική πρόκληση προκύπτει όταν πολλαπλοί αισθητήρες μοιράζονται τον ίδιο φυσικό δίαυλο. Για παράδειγμα, ο αισθητήρας DHT11 παρέχει ταυτόχρονα θερμοκρασία και υγρασία μέσω ενός μόνο GPIO pin. Αν κάθε κλάση δημιουργούσε δικό της αντικείμενο DHT, οι διαδοχικές κλήσεις `read()` θα προκαλούσαν συγκρούσεις στον δίαυλο και τιμές NaN.

Η λύση είναι ένα απλό registry κοινόχρηστων στιγμοτύπων, υλοποιημένο ως συνδεδεμένη λίστα με αναζήτηση κατά pin:

```

1 DHT* getSharedDht(int pin) {
2     struct Entry { int pin; DHT* dht; Entry* next; };
3     static Entry* head = nullptr;
4     // Search for existing instance on this pin
5     for (Entry* e = head; e; e = e->next)
6         if (e->pin == pin) return e->dht;
7     // First request for this pin: create and register
8     Entry* ne = (Entry*)malloc(sizeof(Entry));
9     ne->pin = pin;
10    ne->dht = new DHT(pin, DHT11);
11    ne->dht->begin();
12    ne->next = head;
13    head = ne;
14    return ne->dht;

```

Listing 41: Shared DHT instance registry (dht_shared.cpp)

Με τον μηχανισμό αυτό, οι κλάσεις `DHT11TemperatureReader` και `DHT11HumidityReader` αποκτούν στον κατασκευαστή τους έναν δείκτη στο ίδιο αντικείμενο DHT μέσω `getSharedDht(pin)`, αποφεύγοντας συγκρούσεις.

Το ίδιο πρόβλημα εμφανίζεται και στον DHT20, ο οποίος επικοινωνεί μέσω I^2C . Εκεί η λύση είναι ανάλογη: μια συνάρτηση `getSharedDHT20()` επιστρέφει ένα μοναδικό στιγμιοτύπο `DHT20`, ενώ μια προστασία χρονισμού (`readSharedDHT20()`) εξασφαλίζει ότι η πραγματική ανάγνωση I^2C γίνεται το πολύ μία φορά ανά 2 δευτερόλεπτα (ελάχιστο διάστημα που απαιτεί ο DHT20).

6.3.2 Αισθητήρας Εδαφικής Υγρασίας

Ο χωρητικός αισθητήρας εδαφικής υγρασίας DFRobot SEN0193 διαβάζει αναλογική τάση μέσω ADC. Η τιμή εξαρτάται αντίστροφα από την υγρασία: υψηλή τιμή ADC σημαίνει ξηρό έδαφος (χαμηλή χωρητικότητα), ενώ χαμηλή τιμή σημαίνει υγρό έδαφος. Η μετατροπή σε ποσοστό βασίζεται σε δύο σταθερές βαθμονόμησης που προσδιορίζονται εμπειρικά:

```
1  const int airValue    = SOIL_MOISTURE_AIR_VALUE;    // 2941 (dry)
2  const int waterValue = SOIL_MOISTURE_WATER_VALUE; // 1324 (wet)
3
4  // Clamp to calibrated range
5  if (rawValue > airValue)    rawValue = airValue;
6  if (rawValue < waterValue) rawValue = waterValue;
7
8  // 100% = wet, 0% = dry (inverted scale)
9  float moisturePercent =
10      100.0f * (float)(airValue - rawValue)
11      / (float)(airValue - waterValue);
```

Listing 42: Υπολογισμός ποσοστού εδαφικής υγρασίας (soil_moisture.cpp)

Οι σταθερές `SOIL_MOISTURE_AIR_VALUE` και `SOIL_MOISTURE_WATER_VALUE` ορίζονται στο `config.h` και πρέπει να προσδιοριστούν με βαθμονόμηση για κάθε αισθητήρα: καταγραφή της τιμής ADC στον αέρα (ξηρός) και βυθισμένο σε νερό (υγρός). Η μέτρηση βασίζεται σε μέσο όρο 10 δειγμάτων για σταθερότητα.

6.3.3 Προσομοιωμένος Αισθητήρας

Για δοκιμές χωρίς φυσικό υλικό, η κλάση `SimulatedSensor` επιστρέφει μονοτονικά αυξανόμενες τιμές (αρχική τιμή 20,0 με βήμα +0,13 ανά κλήση). Αυτός ο αισθητήρας επιτρέπει τον έλεγχο ολόκληρης της αλυσίδας — από το εργοστάσιο μέχρι το backend — χωρίς να απαιτείται φυσική συνδεσιμότητα αισθητήρων. Δηλώνεται στο `device_profile.h` ως "SIMULATED" και αγνοεί το όρισμα `pin`.

6.4 Υλοποίηση WiFi Κόμβων

Η παραλλαγή WiFi αποτελεί την κύρια υλοποίηση κόμβου της πλατφόρμας *Agronos*. Ο κόμβος αξιοποιεί τη σύνδεση WiFi του ESP32 για απευθείας επικοινωνία με το backend, τόσο μέσω HTTP όσο και μέσω MQTT. Η ενότητα αυτή αναλύει τον μηχανισμό αρχικής ρύθμισης (provisioning), τη διαχείριση ρυθμίσεων σε χρόνο εκτέλεσης, και τη ροή αυθεντικοποίησης και αποστολής δεδομένων. Ο πλήρης κώδικας παρατίθεται στο Παράρτημα Β΄.4.

6.4.1 Μηχανισμός Provisioning: Enhanced Captive Portal

Ένα κεντρικό πρόβλημα στην ανάπτυξη κόμβων IoT είναι η αρχική ρύθμιση (provisioning) των δικτυακών παραμέτρων. Η ενσωμάτωση (hardcoding) των διαπιστευτηρίων WiFi στον πηγαίο κώδικα δημιουργεί σοβαρά ζητήματα ασφαλείας και πρακτικότητας — κάθε αλλαγή δικτύου θα απαιτούσε εκ νέου μεταγλώττιση και φόρτωση (reflashing). Η πλατφόρμα *Agronos* επιλύει αυτό μέσω ενός Enhanced Captive Portal που λειτουργεί απευθείας στον μικροελεγκτή.

Αρχιτεκτονική του Portal.

Το Captive Portal συνδυάζει τρεις τεχνολογίες σε ένα ενοποιημένο σύστημα:

1. **WiFi Access Point:** Ο ESP32 εκκινεί σε λειτουργία AP+STA (`WIFI_AP_STA`), δημιουργώντας δικό του δίκτυο (`ESP_Config`) στη σταθερή διεύθυνση `192.168.4.1`, ενώ παράλληλα μπορεί να συνδεθεί σε υπάρχον δίκτυο.
2. **HTTP Web Server:** Ένας ενσωματωμένος HTTP server εξυπηρετεί τη δυναμικά παραγόμενη σελίδα HTML με τη φόρμα ρυθμίσεων.

```
1 void WifiPortal::start() {  
2     WiFi.mode(WIFI_AP_STA);
```



```
3   WiFi.softAPConfig(apIP, apIP, IPAddress(255,255,255,0));
4   WiFi.softAP(ssid, pass);
5   scanNetworks(); // Scan for available WiFi networks
6
7   dnsServer.start(53, "*", apIP); // DNS hijacking
8   webServer.on("/", HTTP_GET, ...);
9   webServer.on("/save", HTTP_POST, ...);
10
11  // Platform-specific captive portal detection
12  webServer.on("/generate_204", HTTP_GET, []() { /* 204 */ });
13  webServer.on("/hotspot-detect.html", HTTP_GET, ...);
14  webServer.on("/ncsi.txt", HTTP_GET, ...);
15  webServer.onNotFound(...); // Fallback to portal page
16  webServer.begin();
17 }
```

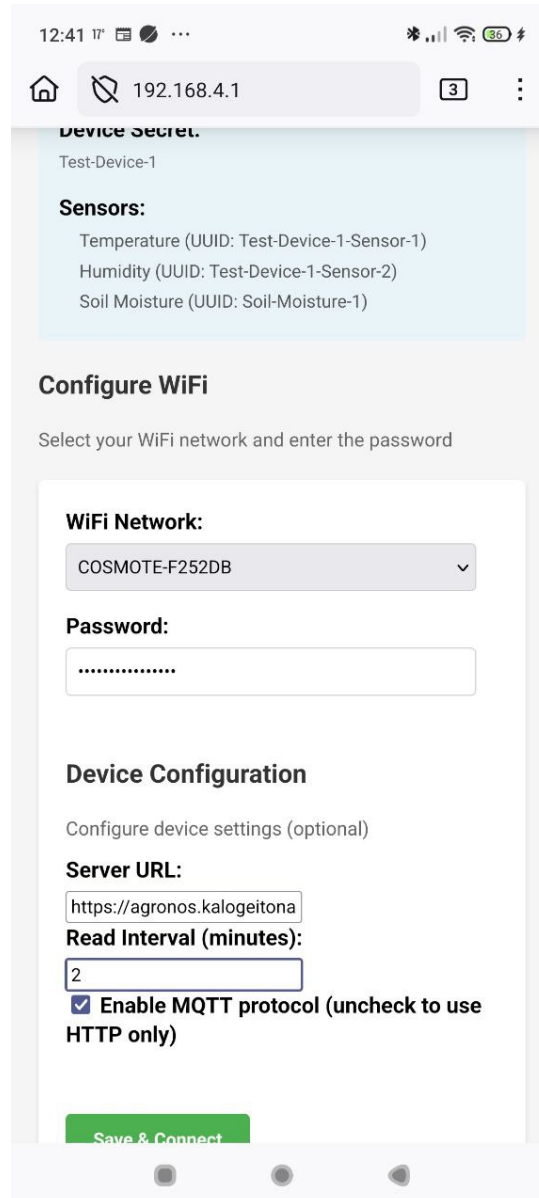
Listing 43: Εκκίνηση του Captive Portal — DNS, HTTP routes και αυτόματη ανίχνευση

Σάρωση ικτύων και δυναμική Φόρμα.

Κατά την εκκίνηση, το portal εκτελεί σάρωση (`WiFi.scanNetworks()`) των διαθέσιμων δικτύων, και τα παρουσιάζει σε dropdown menu. Η δυναμική φόρμα (`generateHtmlPage()`) περιλαμβάνει:

- **Επιλογή δικτύου WiFi:** Dropdown με τα σαρωμένα SSID — αποφεύγονται λάθη πληκτρολόγησης.
- **Κωδικός πρόσβασης:** Πεδίο εισαγωγής τύπου password.
- **Πληροφορίες συσκευής:** UUID, μυστικό κλειδί και λίστα αισθητήρων με τα UUID τους.
- **Ρυθμίσεις λειτουργίας:** Server URL (προ-συμπληρωμένο), διάστημα ανάγνωσης σε λεπτά, ενεργοποίηση/απενεργοποίηση MQTT.

Τα πεδία παρουσιάζονται προ-συμπληρωμένα (pre-populated) με τις τρέχουσες τιμές, είτε από το NVS είτε από τις προεπιλογές του `config.h`, ώστε ο χρήστης να χρειάζεται να τροποποιήσει μόνο ό,τι θέλει να αλλάξει.



Σχήμα 28: Στιγμιότυπο του Captive Portal σε κινητή συσκευή — φόρμα επιλογής WiFi δικτύου και ρύθμισης παραμέτρων συσκευής.

Ροή Αποθήκευσης και Επανεκκίνηση.

Όταν ο χρήστης υποβάλει τη φόρμα (POST στο /save), η μέθοδος onSave() εξάγει τις παραμέτρους και τις αποθηκεύει ατομικά:

```
1 void WifiPortal::onSave() {
2     String ssidArg = webServer.arg("ssid");
3     String passArg = webServer.arg("pass");
4 }
```

```
5     storage.setWifiCreds(ssidArg, passArg); // NVS namespace "
        wifi"
6
7     DeviceConfig newConfig;
8     newConfig.baseUrl = webServer.arg("base_url");
9     newConfig.readIntervalMs = readIntervalArg.toInt() * 60000;
10    newConfig.mqttEnabled = webServer.hasArg("mqtt_enabled");
11
12    storage.saveConfig(newConfig); // NVS namespace "config"
13    ESP.restart();                // Apply new settings
14 }
```

Listing 44: Αποθήκευση ρυθμίσεων από τη φόρμα του Captive Portal

Μετά την αποθήκευση, η συσκευή επανεκκινεί αυτόματα. Στην επόμενη εκκίνηση, ο κώδικας ανιχνεύει τα αποθηκευμένα διαπιστευτήρια και συνδέεται αυτόματα στο WiFi, παρακάμπτοντας το portal.

Λογική Ενεργοποίησης.

Το Captive Portal **δεν** εκκινεί σε κάθε εκκίνηση. Η λογική στη `setup()` ελέγχει πρώτα αν υπάρχουν αποθηκευμένα WiFi credentials και εάν η σύνδεση πετυχαίνει (`tryAutoConnect()`). Μόνο σε αποτυχία ξεκινά το portal. Αντίστοιχα, αν η σύνδεση WiFi επιτευχθεί ενώ το portal τρέχει, τερματίζεται αυτόματα:

```
1 void WifiPortal::handle() {
2     if (!running) return;
3     dnsServer.processNextRequest();
4     webServer.handleClient();
5     if (WiFi.status() == WL_CONNECTED) {
6         stop(); // Portal no longer needed
7     }
8 }
```

Listing 45: Αυτόματος τερματισμός του portal κατά τη σύνδεση WiFi

6.4.2 Ροή Εκκίνησης και Αυθεντικοποίηση

Η ροή εκκίνησης (boot sequence) ενός WiFi κόμβου ακολουθεί μια αυστηρά καθορισμένη σειρά βημάτων, σχεδιασμένη ώστε κάθε φάση να εκτελείται μόνο αν η προηγούμενη ολοκληρώθηκε επιτυχώς:

1. **Αρχικοποίηση υλικού:** Ρύθμιση σειριακής θύρας, GPIO pin κουμπιού, έλεγχος για μακρά πίεση κουμπιού (factory reset).
2. **Φόρτωση ρυθμίσεων:** Η Storage λαμβάνει τις προεπιλογές (loadDefaults) και παρέχει τις αποθηκευμένες ή εναλλακτικές τιμές.
3. **Κατασκευή υποσυστημάτων:** Δημιουργία WifiPortal, AuthManager, DataSender, MqttClient μέσω Dependency Injection.
4. **Δημιουργία αισθητήρων:** Η createSensors() διαβάζει τον πίνακα SENSOR_CONFIGS και παράγει τα αντικείμενα μέσω του factory.
5. **Σύνδεση WiFi:** Προσπάθεια αυτόματης σύνδεσης (tryAutoConnect). Σε αποτυχία → εκκίνηση Captive Portal.
6. **Χορήγηση διαπιστευτηρίων (provisioning):** Μονοφασική εκτέλεση κατά την πρώτη σύνδεση.

Χορήγηση διαπιστευτηρίων (One-Time Provisioning).

Η φάση αυτή εκτελείται μόνο μία φορά μετά τη σύνδεση WiFi και είναι η γέφυρα μεταξύ της φυσικής συσκευής και του backend:

```
1 static void oneTimeProvisioning() {  
2     if (WiFi.status() != WL_CONNECTED) return;  
3  
4     // Step 1: Ensure JWT token exists  
5     String token = storage.getToken();  
6     if (token.length() == 0) {  
7         auth->tryAuthenticateOnce(); // POST /api/v1/device/login  
8         token = storage.getToken();  
9     }  
10 }
```

```
11     if (!mqttEnabled) return;
12
13     // Step 2: Fetch MQTT credentials if needed
14     if (!auth→hasMqttCredentials() && token.length() > 0) {
15         auth→fetchMqttCredentials(); // GET /api/v1/device/mqtt-
16                                     credentials
17     }
18
19     // Step 3: Establish initial MQTT connection
20     if (auth→hasMqttCredentials()) {
21         mqttClient→connect();
22     }
```

Listing 46: Μονοφασική χορήγηση διαπιστευτηρίων κατά την εκκίνηση

Η ροή αυτή αντικατοπτρίζει τα τρία στάδια αυθεντικοποίησης που περιγράφηκαν στην Ενότητα 4.4: (1) απόκτηση JWT μέσω HTTP, (2) ανάκτηση διαπιστευτηρίων MQTT με Bearer token, (3) σύνδεση στον MQTT broker. Κάθε βήμα ελέγχεται ανεξάρτητα — αν η συσκευή ήδη διαθέτει token ή MQTT credentials (π.χ. από προηγούμενη εκκίνηση), παραλείπεται η αντίστοιχη φάση.

6.4.3 Δρομολόγηση Δεδομένων: MQTT-First με HTTP Fallback

Η κλάση `DataSender` υλοποιεί μια στρατηγική διπλού πρωτοκόλλου, όπου η επικοινωνία μέσω HTTP λειτουργεί ως εναλλακτικός μηχανισμός υποχώρησης (fallback). Η συμπεριφορά αυτή ενεργοποιείται κυρίως στην περίπτωση που ο χρήστης έχει επιλέξει τη χρήση MQTT στις ρυθμίσεις, αλλά η συσκευή δεν έχει λάβει ακόμη τα απαραίτητα διαπιστευτήρια για τον MQTT broker από την πλατφόρμα:

```
1  bool DataSender::sendReadings(const SensorReading* readings,
2                                size_t count) {
3      // 1. Build JSON payload once (shared by both protocols)
4      DynamicJsonDocument doc(capacity);
5      JSONArray sensorsArr = doc.createNestedArray("sensors");
6      for (size_t i = 0; i < count; ++i) {
7          JsonObject obj = sensorsArr.createNestedObject();
```

```
8         obj["uuid"] = readings[i].uuid;
9         obj["value"] = roundf(readings[i].value * 100) / 100;
10    }
11    String payload;
12    serializeJson(doc, payload);
13
14    // 2. Try MQTT first
15    if (MQTT_ENABLED && mqttClient && storage.hasMqttCredentials
16        ()) {
17        if (!mqttClient->isConnected())
18            mqttClient->connect();    // Ephemeral connect
19        if (mqttClient->publishSensorDataPayload(payload.c_str()))
20            return true;              // Success -- skip HTTP
21    }
22
23    // 3. Fallback to HTTP
24    String token = storage.getToken();
25    return postJson(payload, token);  // POST /api/v1/device/data
26 }
```

Listing 47: Στρατηγική MQTT-first με HTTP fallback στον DataSender

Ενιαίο JSON Payload.

Η δομή του JSON είναι ταυτόσημη ανεξαρτήτως πρωτοκόλλου:

```
1 {
2   "sensors": [
3     { "uuid": "DHT20-Temp-1", "value": 23.45 },
4     { "uuid": "Soil-1",       "value": 67.20 }
5   ]
6 }
```

Listing 48: Δομή JSON payload αισθητήρων

Με αυτήν την ταύτιση πετυχαίνουμε ότι το backend επεξεργάζεται τα δεδομένα με τον ίδιο τρόπο, ανεξάρτητα αν προέρχονται από τον MQTT webhook ή απευθείας μέσω HTTP

POST.

Εφήμερη Σύνδεση MQTT (Ephemeral Connect).

Σε αντίθεση με την τυπική χρήση MQTT όπου ο πελάτης διατηρεί μόνιμη σύνδεση (persistent connection), ο κόμβος ακολουθεί στρατηγική εφήμερης σύνδεσης: συνδέεται στον broker μόνο για τη διάρκεια του publish και αποσυνδέεται αμέσως μετά. Η επιλογή αυτή είναι σκόπιμη, καθώς ο κόμβος εισέρχεται σε Deep Sleep μετά την αποστολή — μια μόνιμη σύνδεση θα τερματιζόταν ούτως ή άλλως κατά την αναστολή λειτουργίας.

Υποσύστημα MQTT (MqttClient).

Η κλάση MqttClient διαχειρίζεται τη σύνδεση με τον EMQX broker μέσω της βιβλιοθήκης PubSubClient. Τα βασικά χαρακτηριστικά:

- **Δομή θεμάτων (topic hierarchy):** Η δημοσίευση γίνεται στο topic `devices/{uuid}/sensors`, σύμφωνα με τους κανόνες ACL του broker (Ενότητα 4.4).
- **Δυναμικά διαπιστευτήρια:** Δεν χρησιμοποιούνται hardcoded στοιχεία σύνδεσης — τα server, username και password ανακτώνται από τον Storage κατά τη σύνδεση.
- **Αποκλειστική δημοσίευση:** Ο κόμβος δεν εγγράφεται σε κανένα topic (publish-only), ελαχιστοποιώντας την κατανάλωση πόρων.

```
1  bool MqttClient::connect() {
2      if (!loadCredentials()) return false; // From NVS
3      if (mqttClient.connected()) return true;
4
5      mqttClient.setServer(credentials.server.c_str(),
6                          AGRONOS_MQTT_PORT);
7      String clientId = String("agronos-") + deviceUuid;
8
9      return mqttClient.connect(
10         clientId.c_str(),
11         credentials.username.c_str(),
12         credentials.password.c_str(),
13         nullptr, 0, false, nullptr,
14         AGRONOS_MQTT_CLEAN_SESSION // true: no session state
```

```
15     );  
16 }
```

Listing 49: Σύνδεση στον MQTT broker με δυναμικά διαπιστευτήρια

6.4.4 Κύριος Βρόχος Εκτέλεσης (Main Loop)

Μετά την ολοκλήρωση της εκκίνησης, ο κύριος βρόχος `loop()` ακολουθεί μια απλή και ενεργειακά αποδοτική λογική:

```
1  void loop() {  
2      portal->handle();  // DNS + HTTP if portal is active  
3  
4      if (WiFi.status() != WL_CONNECTED) return;  
5  
6      auth->loop();  // Periodic re-auth if token is missing  
7  
8      // Rate-limited send attempt  
9      if (millis() - lastSendAttempt < SEND_RETRY_BACKOFF_MS)  
10         return;  
11  
12     lastSendAttempt = millis();  
13  
14     if (sendMeasurements()) {  
15         // Success: configure timer + GPIO wakeup, enter Deep  
16         // Sleep  
17         mqttClient->disconnect();  
18         WiFi.disconnect(true);  
19         WiFi.mode(WIFI_OFF);  
20  
21         // Full platform-specific wakeup setup: see Listing~\ref{  
22         // lst:wifi_deep_sleep}  
23         esp_sleep_enable_timer_wakeup(sleep_us);  // Wake after  
24         readIntervalMs  
25         esp_deep_sleep_start();  
26     }  
27     // Failure: retry after SEND_RETRY_BACKOFF_MS (10 sec)
```


Listing 50: Κύριος βρόχος εκτέλεσης WiFi κόμβου (απλοποιημένος — πλήρης υλοποίηση Deep Sleep στο Listing 66)

Η λογική αυτή ακολουθεί το μοντέλο *μέτρηση* → *αποστολή* → *ύπνος*: ο κόμβος παραμένει ενεργός μόνο για τη διάρκεια που απαιτείται η σύνδεση, η ανάγνωση αισθητήρων και η αποστολή δεδομένων. Σε αποτυχία αποστολής, εφαρμόζεται backoff 10 δευτερολέπτων πριν τη νέα προσπάθεια, αποφεύγοντας υπερφόρτωση του δικτύου ή του backend.

Η συνάρτηση `sendMeasurements()` διαβάζει διαδοχικά κάθε αισθητήρα μέσω της πολυμορφικής κλήσης `sensors[i]->read(value)`, στρογγυλοποιεί τις τιμές σε δύο δεκαδικά ψηφία και τις προωθεί στον `DataSender` για αποστολή μέσω MQTT ή HTTP.

6.5 Υλοποίηση LoRa Κόμβων

Η παραλλαγή LoRa είναι η δεύτερη υλοποίηση κόμβου της πλατφόρμας *Agronos*, σχεδιασμένη για περιοχές χωρίς κάλυψη WiFi. Σε αντίθεση με τον WiFi κόμβο, ο κόμβος LoRa δεν αυθεντικοποιείται απευθείας στο backend ούτε χρησιμοποιεί πρωτόκολλα HTTP ή MQTT. Αντ' αυτού, σειριοποιεί τις μετρήσεις σε δυαδική μορφή, τις κρυπτογραφεί με AES-128-CTR και τις μεταδίδει μέσω ραδιοσυχνότητας LoRa στο Gateway, το οποίο αναλαμβάνει την προώθηση στο Internet. Η επιλογή του πομποδέκτη SX1276 βασίζεται στη διαμόρφωση LoRa® ως ultra-long range spread spectrum communication, η οποία παρέχει υψηλή ανοσία σε παρεμβολές (high interference immunity) και ευαισθησία λήψης έως και -148 dBm [48]. Τα χαρακτηριστικά αυτά καθιστούν εφικτή την αξιόπιστη επικοινωνία σε μεγάλες αποστάσεις, ακόμη και όταν παρεμβάλλονται εμπόδια όπως δέντρα, ελαιώνες ή κτίρια. Η ενότητα αυτή αναλύει τη ροή εκτέλεσης, τη δυαδική σειριοποίηση, την κρυπτογραφία και τη διαχείριση του μετρητή πλαισίου. Ο πλήρης κώδικας παρατίθεται στο Παράρτημα Β'.5.

6.5.1 Αρχιτεκτονική και Διαφοροποίηση από WiFi

Η αρχιτεκτονική του κόμβου LoRa βασίζεται στην ίδια αρθρωτή σχεδίαση με τον κόμβο WiFi (Ενότητα 6.2), αλλά αντικαθιστά πλήρως το στρώμα δικτύωσης. Τα κοινά υποσυστήματα — αφαίρεση αισθητήρων (`SensorBase`), εργοστάσιο δημιουργίας (`SensorFactory`), μόνιμη αποθήκευση (`Storage`) — επαναχρησιμοποιούνται χωρίς τροποποίηση. Τα υποσυστήματα ειδικά για WiFi (`WifiPortal`, `AuthManager`, `DataSender`, `MqttClient`) αντι-

καθίστανται από τέσσερα νέα:

- **lora_payload**: Δυναδική σειριοποίηση μετρήσεων σε συμπαγή μορφή.
- **lora_crypto**: Κρυπτογράφηση AES-128-CTR με ντετερμινιστική κατασκευή nonce.
- **lora_fcnt**: Διαχείριση μονοτονικού μετρητή πλαισίου (frame counter).
- **lora_radio**: Οδηγός (driver) του ραδιοπομπού SX1276.

Ο Πίνακας 11 συνοψίζει την επαναχρησιμοποίηση κώδικα μεταξύ των δύο παραλλαγών.

Υποσύστημα	WiFi	LoRa
sensor.h / SensorBase	✓	✓
sensor_factory.cpp	✓	✓
Υλοποιήσεις αισθητήρων (DHT11, κ.λπ.)	✓	✓
storage.cpp (NVS)	✓	✓
wifi_portal.cpp	✓	—
auth.cpp / data_sender.cpp	✓	—
mqtt_client.cpp	✓	—
lora_*.cpp	—	✓

Πίνακας 11: Επαναχρησιμοποίηση υποσυστημάτων μεταξύ WiFi και LoRa παραλλαγών

Ο διαχωρισμός επιτυγχάνεται μέσω φίλτρων πηγαίου κώδικα (`src_filter`) στο `platformio.ini` (Listing 36), αντί οδηγιών προεπεξεργαστή. Τα WiFi-specific αρχεία δεν μεταγλωττίζονται καν στον στόχο LoRa και αντίστροφα. Η μόνη οδηγία προεπεξεργαστή που χρησιμοποιείται είναι η `-D LORA_NODE=1`, η οποία ενεργοποιεί τις LoRa-specific σταθερές στο `config.h` (ακροδέκτες SPI, ρυθμίσεις ραδιοσυχνότητας, σταθερές μετρητή πλαισίου).

6.5.2 Ροή Εκτέλεσης

Σε αντίθεση με τον WiFi κόμβο που χρησιμοποιεί `loop()` με μηχανισμό backoff (Ενότητα 6.4.4), ο κόμβος LoRa ακολουθεί μοντέλο **μίας εκτέλεσης** (one-shot): η πλήρης λογική εκτελείται εντός της `setup()` και η `loop()` παραμένει κενή, καθώς η συσκευή εισέρχεται πάντα σε Deep Sleep μετά την ολοκλήρωση. Η ροή αυτή ελαχιστοποιεί τον χρόνο ενεργού λειτουργίας και κατά συνέπεια την κατανάλωση ενέργειας:

1. **Αρχικοποίηση υλικού:** Ρύθμιση σειριακής θύρας, ενεργοποίηση LED, έλεγχος για μακρά πίεση κουμπιού (factory reset).
2. **Αρχικοποίηση μετρητή πλαισίου:** Η `fcntInit()` ανακτά τον μετρητή από τη μνήμη RTC (αφύπνιση) ή τον NVS (cold boot).
3. **Αρχικοποίηση ραδιοσυχνότητας:** Η `loraRadioInit()` ρυθμίζει τον SX1276 μέσω SPI.
4. **Δημιουργία αισθητήρων:** Η `createSensors()` (κοινή με WiFi) διαβάζει τον πίνακα `SENSOR_CONFIGS`.
5. **Ανάγνωση μετρήσεων:** Πολυμορφική κλήση `sensors[i]->read(value)` για κάθε αισθητήρα.
6. **Δυναδική σειριοποίηση:** Η `serializeReadings()` μετατρέπει τις μετρήσεις σε $N \times 6$ bytes.
7. **Κρυπτογράφηση:** Κατασκευή nonce, κρυπτογράφηση AES-128-CTR.
8. **Μετάδοση:** Η `loraTransmit()` αποστέλλει το πακέτο μέσω ραδιοσυχνότητας.
9. **Deep Sleep:** Αναστολή λειτουργίας για διάστημα `SENSORS_READ_INTERVAL_MS`.

Σε περίπτωση αποτυχίας σε οποιοδήποτε βήμα (αρχικοποίηση ραδιοσυχνότητας, σειριοποίηση, κρυπτογράφηση), ο κόμβος εισέρχεται σε Deep Sleep χωρίς μετάδοση, αποφεύγοντας τη σπατάλη ενέργειας. Στο Listing 51 παρουσιάζονται τα κρίσιμα βήματα της ροής εκτέλεσης.

```
1 void setup() {  
2     Serial.begin(115200);  
3  
4     // 1. Button factory reset check  
5     pinMode(BUTTON_PIN, INPUT_PULLUP);  
6     checkButtonReset();  
7  
8     // 2. Frame counter: RTC warm wake or NVS cold boot  
9     fcntInit(storage);
```

```
10
11 // 3. LoRa radio initialization
12 if (!loraRadioInit()) {
13     esp_deep_sleep_start(); // Fatal: sleep and retry later
14 }
15
16 // 4. Create sensors (shared factory pattern)
17 sensors = createSensors();
18
19 // 5. Read all sensors
20 std::vector<SensorReading> readings;
21 for (auto& s : sensors) {
22     float value;
23     if (s->read(value))
24         readings.push_back({ s->uuid(), roundf(value*100)/100
25                               });
26 }
27
28 if (readings.empty()) { loraRadioSleep();
29     esp_deep_sleep_start(); }
30
31 // 6. Serialize to binary (N x 6 bytes)
32 uint8_t plaintext[MAX_PAYLOAD_SIZE];
33 size_t payloadLen = serializeReadings(readings.data(),
34                                       readings.size(), plaintext, sizeof(
35                                           plaintext));
36
37 // 7. Encrypt with AES-128-CTR
38 uint32_t fcnt = fcntNext(storage);
39 uint8_t nonce[16];
40 buildNonce(DEFAULT_UUID, fcnt, nonce);
41 uint8_t ciphertext[MAX_PAYLOAD_SIZE];
42 encryptPayload(plaintext, payloadLen, LORA_AES_KEY,
43               nonce, ciphertext);
```

```

41 // 8. Transmit via LoRa radio
42 loraTransmit(DEFAULT_UUID, fcmt, ciphertext, payloadLen);
43
44 // 9. Deep sleep
45 loraRadioSleep();
46 esp_sleep_enable_timer_wakeup(
47     (uint64_t)SENSORS_READ_INTERVAL_MS * 1000ULL);
48 esp_deep_sleep_start();
49 }
50
51 void loop() {} // Never reached

```

Listing 51: Ροή εκτέλεσης κόμβου LoRa — βασικά βήματα από τη `setup()` (`main_lora.cpp`)

6.5.3 Δυαδική Σειριοποίηση (Binary Payload)

Σε αντίθεση με τον WiFi κόμβο που χρησιμοποιεί JSON (Ενότητα 6.4.3), ο κόμβος LoRa σειριοποιεί τις μετρήσεις σε συμπαγή δυαδική μορφή, ώστε να ελαχιστοποιηθεί το μέγεθος μετάδοσης μέσω ραδιοσυχνότητας. Κάθε αισθητήρας αναπαρίσταται σε ακριβώς 6 bytes:

Bytes	Περιεχόμενο	Μορφή
0–3	Πρόθεμα UUID αισθητήρα	4 χαρακτήρες ASCII
4–5	Τιμή $\times 100$	int16_t Little-Endian

Πίνακας 12: Δυαδική δομή μέτρησης ανά αισθητήρα (6 bytes)

Για παράδειγμα, ένας αισθητήρας με UUID "DHT2-Temp-1" και τιμή 23,45 σειριοποιείται ως: bytes 0–3 = 44 48 54 32 (ASCII: "DHT2") και bytes 4–5 = 29 09 (η τιμή $2345 = 23,45 \times 100$ σε Little-Endian). Η δομή αυτή είναι πανομοιότυπη με αυτή που αναμένει η `LoRaCryptoService::deserialize()` στο backend, εξασφαλίζοντας πλήρη συμβατότητα μεταξύ firmware και διακομιστή.

```

1 size_t serializeReadings(const SensorReading* readings, size_t
    count,
2                             uint8_t* outBuf, size_t bufSize) {
3     size_t needed = count * 6; // 6 bytes per sensor
4     if (!readings || count == 0 || bufSize < needed) return 0;

```

```
5
6   for (size_t i = 0; i < count; ++i) {
7       uint8_t* chunk = outBuf + (i * 6);
8
9       // Bytes 0-3: first 4 ASCII chars of UUID
10      size_t uuidLen = strlen(readings[i].uuid);
11      for (size_t j = 0; j < 4; ++j)
12          chunk[j] = (j < uuidLen)
13              ? static_cast<uint8_t>(readings[i].uuid[j]) : 0;
14
15      // Bytes 4-5: value * 100 as int16_t (native LE on ESP32)
16      int16_t scaled = static_cast<int16_t>(
17          readings[i].value * 100.0f);
18      memcpy(chunk + 4, &scaled, sizeof(int16_t));
19  }
20  return needed;
21 }
```

Listing 52: Δυναδική σειριοποίηση μετρήσεων αισθητήρων (lor_payload.cpp)

Η χρήση `memcpy` για την αποθήκευση της τιμής `int16_t` εκμεταλλεύεται το γεγονός ότι ο ESP32 είναι εγγενώς Little-Endian, αποφεύγοντας χειροκίνητη αντιστροφή bytes.

6.5.4 Κρυπτογράφηση AES-128-CTR

Για την ασφάλεια του κόμβου LoRa επιλέχθηκε κρυπτογράφηση στο επίπεδο εφαρμογής με AES-128-CTR (Counter Mode). Η CTR δεν απαιτεί padding — το κρυπτοκείμενο έχει ακριβώς το ίδιο μήκος με το αρχικό κείμενο — και λειτουργεί ως κρυπτογράφηση ροής (stream cipher), χαρακτηριστικά ιδανικά για IoT εφαρμογές με δεδομένα μικρού μεγέθους.

Κατασκευή Nonce.

Το 16-byte nonce κατασκευάζεται ντετερμινιστικά, χωρίς να μεταδίδεται στον αέρα. Η δομή του πρέπει να είναι πανομοιότυπη μεταξύ firmware και backend:

Bytes	Περιεχόμενο	Μορφή
0–3	CRC32(UUID)	uint32 Little-Endian
4–7	Μετρητής πλαισίου (fcnt)	uint32 Little-Endian
8–15	Σταθερή συμπλήρωση	0x00

Πίνακας 13: Δομή 16-byte nonce για AES-128-CTR

Η χρήση CRC32 του UUID (αντί του ίδιου του UUID) δίνει πάντα σταθερό μήκος 4 bytes, ανεξαρτήτως του μήκους του αναγνωριστικού. Η υλοποίηση χρησιμοποιεί τον ίδιο πολυωνυμικό αλγόριθμο (0xEDB88320) με τη PHP `crc32()`, ώστε firmware και backend να παράγουν ακριβώς το ίδιο nonce:

```

1  uint32_t uuidHash(const char* uuid) {
2      uint32_t crc = 0xFFFFFFFF;
3      while (*uuid) {
4          crc ^= (uint8_t)*uuid++;
5          for (int j = 0; j < 8; ++j)
6              crc = (crc >> 1) ^ (0xEDB88320 & -(crc & 1));
7      }
8      return crc ^ 0xFFFFFFFF;
9  }
10
11 void buildNonce(const char* uuid, uint32_t fcnt,
12                uint8_t nonce[16]) {
13     memset(nonce, 0, 16);
14     uint32_t hash = uuidHash(uuid);
15     memcpy(nonce, &hash, 4); // Bytes 0-3: CRC32(UUID) LE
16     memcpy(nonce + 4, &fcnt, 4); // Bytes 4-7: fcnt LE
17 }

```

Listing 53: Κατασκευή ντετερμινιστικού nonce για AES-128-CTR (lora_crypto.cpp)

Κρυπτογράφηση μέσω mbedTLS.

Η κρυπτογράφηση εκτελείται μέσω της βιβλιοθήκης mbedTLS, η οποία παρέχεται ενσωματωμένη στο πλαίσιο Arduino για ESP32 και αξιοποιεί τον επιταχυντή υλικού AES (hardware accelerator) του ESP32:

```
1  bool encryptPayload(const uint8_t* plaintext, size_t len,  
2                      const uint8_t key[16],  
3                      const uint8_t nonce[16],  
4                      uint8_t* ciphertext) {  
5      mbedtls_aes_context ctx;  
6      mbedtls_aes_init(&ctx);  
7      mbedtls_aes_setkey_enc(&ctx, key, 128);  
8  
9      uint8_t nonceCounter[16];  
10     memcpy(nonceCounter, nonce, 16);  
11     uint8_t streamBlock[16] = {0};  
12     size_t ncOffset = 0;  
13  
14     int ret = mbedtls_aes_crypt_ctr(&ctx, len, &ncOffset,  
15                                     nonceCounter, streamBlock, plaintext, ciphertext)  
16                                     ;  
17     mbedtls_aes_free(&ctx);  
18     return (ret == 0);  
19 }
```

Listing 54: Κρυπτογράφηση AES-128-CTR με επιτάχυνση υλικού (lora_crypto.cpp)

Η προσέγγιση αυτή παρέχει σημαντικά πλεονεκτήματα, καθώς αξιοποιεί τον επιταχυντή υλικού AES του κλασικού ESP32 [49] για μέγιστη ταχύτητα και βασίζεται σε μια καθιερωμένη κρυπτογραφική βιβλιοθήκη χωρίς ιδιοκατασκευές. Επιπλέον, συνεπάγεται μηδενικές επιπλέον εξαρτήσεις, αφού η mbedtls είναι ήδη ενσωματωμένη στο πλαίσιο Arduino για ESP32.

6.5.5 Διαχείριση Μετρητή Πλαισίου (Frame Counter)

Ο μετρητής πλαισίου (frame counter, fcnt) είναι μονοτονικά αυξανόμενος αριθμός: ενσωματώνεται στο nonce (Πίνακας 13) ώστε κάθε μετάδοση να κρυπτογραφείται με μοναδικό nonce, και επαληθεύεται από το backend ως μέτρο έναντι επιθέσεων επανάληψης (replay attacks). Η υλοποίησή του πρέπει να εξισορροπήσει δύο αντικρουόμενες ανάγκες: **ακρίβεια** (να μην επαναχρησιμοποιηθεί ποτέ η ίδια τιμή) και **προστασία μνήμης flash**

(να μην γράφεται στον NVS σε κάθε μετάδοση, αποφεύγοντας τη φθορά).

Υβριδική Μνήμη RTC + NVS.

Η λύση βασίζεται σε υβριδική αποθήκευση: η μνήμη RTC (RTC_DATA_ATTR) διατηρεί τον μετρητή κατά τη διάρκεια Deep Sleep, ενώ ο NVS παρέχει αντίγραφο ασφαλείας για περιπτώσεις απώλειας ρεύματος:

```
1  RTC_DATA_ATTR static uint32_t rtcFcnt = 0;
2  RTC_DATA_ATTR static bool    rtcInitialized = false;
3  RTC_DATA_ATTR static uint32_t txSinceLastSave = 0;
4
5  void fcntInit(Storage& storage) {
6      if (rtcInitialized) {
7          // Deep sleep wake: RTC value is valid
8          return;
9      }
10     // Cold boot: read NVS + safety gap
11     uint32_t nvsFcnt = storage.getLoraFcnt();
12     rtcFcnt = nvsFcnt + FCNT_COLD_BOOT_GAP; // +100
13     txSinceLastSave = 0;
14     rtcInitialized = true;
15     storage.setLoraFcnt(rtcFcnt);
16 }
17
18 uint32_t fcntNext(Storage& storage) {
19     rtcFcnt++;
20     txSinceLastSave++;
21     // Lazy save: write NVS every 100 transmissions
22     if (txSinceLastSave >= FCNT_NVS_SAVE_INTERVAL) {
23         storage.setLoraFcnt(rtcFcnt);
24         txSinceLastSave = 0;
25     }
26     return rtcFcnt;
27 }
```

Listing 55: Υβριδική διαχείριση μετρητή πλαισίου (lora_fcnt.cpp)

Ο Πίνακας 14 συνοψίζει τη συμπεριφορά σε κάθε σενάριο εκκίνησης:

Σενάριο	Συμπεριφορά
Deep Sleep αφύπνιση	Η <code>rtcInitialized</code> είναι <code>true</code> — η τιμή RTC είναι έγκυρη, καμία προσπάθεια NVS.
Cold Boot (απώλεια ρεύματος)	Ανάκτηση από NVS, προσθήκη <code>FCNT_COLD_BOOT_GAP = 100</code> , αποθήκευση στο NVS.
Κάθε 100 μεταδόσεις	Lazy save: αποθήκευση τρέχουσας τιμής RTC στο NVS.

Πίνακας 14: Σενάρια εκκίνησης και συμπεριφορά μετρητή πλαισίου

Με το “άλμα” +100 σε cold boot, ακόμη και αν η τελευταία εγγραφή στον NVS ήταν 99 μεταδόσεις πριν από την απώλεια ρεύματος, ο κόμβος επανεκκινεί με μετρητή **μεγαλύτερο** από τον τελευταίο μεταδοθέντα. Αυτό είναι ασφαλές διότι το backend αποδέχεται άλματα μετρητή πλαισίου έως `MAX_FCNT_GAP = 10.000`.

6.5.6 Οδηγός Ραδιοσυχνότητας (LoRa Radio Driver)

Ο οδηγός ραδιοσυχνότητας αφαιρεί τις λεπτομέρειες του ελεγκτή SX1276 πίσω από τρεις συναρτήσεις: αρχικοποίηση, μετάδοση και αναστολή λειτουργίας. Η αρχικοποίηση ρυθμίζει τον δίαυλο SPI και τις παραμέτρους μετάδοσης:

```

1  bool loraRadioInit() {
2      SPI.begin(LORA_SCK_PIN, LORA_MISO_PIN,
3              LORA_MOSI_PIN, LORA_SS_PIN);
4      LoRa.setPins(LORA_SS_PIN, LORA_RST_PIN, LORA_DIO0_PIN);
5
6      if (!LoRa.begin(LORA_FREQUENCY)) return false; // 868 MHz
7
8      LoRa.setSpreadingFactor(LORA_SF); // SF10
9      LoRa.setSignalBandwidth(LORA_BW); // 125 kHz
10     LoRa.setCodingRate4(LORA_CR); // 4/5
11     LoRa.setSyncWord(LORA_SYNC_WORD); // 0x12
12     LoRa.setTxPower(LORA_TX_POWER); // 20 dBm
13     return true;

```

Listing 56: Αρχικοποίηση ραδιοπομπού LoRa (lora_radio.cpp)

Οι παράμετροι ραδιοσυχνότητας (Spreading Factor 10, εύρος ζώνης 125 kHz, Coding Rate 4/5) προσφέρουν συμβιβασμό μεταξύ εμβέλειας και ρυθμού μετάδοσης. Σύμφωνα με τη διαμόρφωση του LoRa modem στο SX1276 [48], η αύξηση του Spreading Factor βελτιώνει την ευαισθησία του δέκτη και την ικανότητα λήψης σημάτων κάτω από το επίπεδο θορύβου, αλλά αυξάνει αντίστοιχα τον χρόνο εκπομπής (Time-on-Air). Η επιλογή SF10 με εύρος ζώνης 125 kHz και Coding Rate 4/5 και πιστεύουμε ότι είναι ένα πρακτικό σημείο ισορροπίας για αγροτικό περιβάλλον: προσφέρει εμβέλεια αρκετών χιλιομέτρων χωρίς υπερβολική επιβάρυνση στον χρόνο μετάδοσης, ενώ η ισχύς εκπομπής 20 dBm αποτελεί το μέγιστο επιτρεπόμενο για τη ζώνη 868 MHz στην Ευρώπη. Η λέξη συγχρονισμού 0x12 σηματοδοτεί ιδιωτικό δίκτυο, αποτρέποντας τη λήψη πακέτων από δημόσια δίκτυα LoRa-WAN (που χρησιμοποιούν 0x34).

ομή Πακέτου Μετάδοσης.

Η `loraTransmit()` κατασκευάζει και αποστέλλει το πακέτο σε δυαδική μορφή. Η δομή αυτή — η οποία αναλύθηκε στην Ενότητα 4.4 — επιτρέπει στο backend να αναγνωρίσει τη συσκευή, να ανακατασκευάσει το nonce και να αποκρυπτογραφήσει τις μετρήσεις:

```
1 bool loraTransmit(const char* uuid, uint32_t fcnt,
2                  const uint8_t* ciphertext, size_t cipherLen) {
3     uint8_t uuidLen = (uint8_t)strlen(uuid);
4     uint8_t fcntBytes[4];
5     memcpy(fcntBytes, &fcnt, 4); // Native Little-Endian
6
7     LoRa.beginPacket();
8     LoRa.write(uuidLen);           // 1B: UUID
9     LoRa.write((const uint8_t*)uuid, uuidLen); // NB: UUID (
10    LoRa.write(fcntBytes, 4);       // 4B: fcnt (LE
11    LoRa.write(ciphertext, cipherLen); // NB:
12    }
```

```
12   LoRa.endPacket(true); // Blocking: wait for TX complete
13   return true;
14 }
```

Listing 57: Μετάδοση πακέτου LoRa (lora_radio.cpp)

6.5.7 Προφίλ Συσκευής LoRa

Η παραμετροποίηση ενός κόμβου LoRa ακολουθεί το ίδιο πρότυπο `device_profile.h` (Ενότητα 6.2), με μία επιπλέον ρύθμιση: το 128-bit κλειδί AES που είναι μοναδικό ανά συσκευή. Η αναγνώριση της συσκευής γίνεται μέσω του υπάρχοντος `DEFAULT_UUID` — δεν απαιτείται ξεχωριστό αναγνωριστικό LoRa. Το UUID ενσωματώνεται στην επικεφαλίδα κάθε πακέτου και χρησιμοποιείται (μέσω CRC32 hash) για την κατασκευή του nonce.

```
1  constexpr const char* DEFAULT_UUID    = "Test-LoRa-Battery";
2  constexpr const char* DEFAULT_SECRET = "Test-LoRa-Battery";
3
4  constexpr SensorConfig SENSOR_CONFIGS[] = {
5      { "BatteryLevelSensor", 35, "Lora-Battery-Level-1",
6        "Battery Level" },
7  };
8  constexpr size_t SENSOR_CONFIG_COUNT =
9      sizeof(SENSOR_CONFIGS) / sizeof(SENSOR_CONFIGS[0]);
10
11 // 128-bit AES key (matches backend lora_aes_key column)
12 constexpr uint8_t LORA_AES_KEY[16] = {
13     0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
14     0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F
15 };
```

Listing 58: Προφίλ συσκευής LoRa (device_profile.h)

Η σχεδίαση αυτή μας επιτρέπει ότι η προσθήκη ενός νέου κόμβου LoRa απαιτεί μόνο τη δημιουργία του αρχείου `device_profile.h` με το κατάλληλο UUID, τη λίστα αισθητήρων και το κλειδί AES — ακολουθώντας ακριβώς την ίδια διαδικασία τριών βημάτων (3-step process) με τον κόμβο WiFi, με τη μόνη προσθήκη του κλειδιού κρυπτογράφησης.

6.6 Υλοποίηση LoRa Gateway (Bridge Layer)

Ο πλήρης πηγαίος κώδικας του Lora Gateway είναι δημοσίως διαθέσιμος στο αποθετήριο GitHub: <https://github.com/gkalogeitonas/ESP32-LoRa-to-Internet-Gateway>.

Το LoRa Gateway αποτελεί τον ενδιάμεσο κρίκο μεταξύ των κόμβων LoRa και του backend. Λειτουργεί ως **ελαφρύ transport-layer bridge**: λαμβάνει αδιαφανή κρυπτογραφημένα πακέτα LoRa, τα εμπλουτίζει με ραδιομεταδεδομένα (RSSI, SNR) και τα προωθεί αυτούσια στο Internet μέσω MQTT ή HTTP, χωρίς να αποκρυπτογραφεί το περιεχόμενό τους. Αυτό διασφαλίζει ότι ακόμη και αν το Gateway παραβιαστεί, τα δεδομένα παραμένουν ασφαλή και μη αναγνώσιμα.

Το Gateway αναπτύχθηκε ως ανεξάρτητο project², ξεχωριστό από τον κώδικα των κόμβων, ώστε να λειτουργεί ως γενικό, επαναχρησιμοποιήσιμο εργαλείο. Η επιλογή αυτή — αντί ενός πλήρους LoRaWAN concentrator — μειώνει δραστικά το κόστος υλικού και τη λειτουργική πολυπλοκότητα, καθιστώντας τη λύση ιδιαίτερα κατάλληλη για μικρής έως μεσαίας κλίμακας αγροτικές εφαρμογές. Ο πλήρης κώδικας του Gateway παρατίθεται στο Παράρτημα ??.

6.6.1 Πλατφόρμα Υλικού

Το Gateway στοχεύει αποκλειστικά την πλακέτα **TTGO LoRa32 v2.1** — την ίδια πλακέτα που χρησιμοποιείται και για τους κόμβους LoRa — η οποία ενσωματώνεται:

- ESP32-D0WDQ6 MCU με WiFi
- Πομποδέκτη SX1276 LoRa (868 MHz)
- Οθόνη OLED SSD1306 (128×64) μέσω I2C
- Είσοδο μέτρησης μπαταρίας μέσω GPIO35
- Υποστήριξη τροφοδοσίας LiPo

Ο συνδυασμός ταυτόχρονης ύπαρξης δέκτη LoRa και WiFi στην ίδια πλακέτα, μαζί με την ενσωματωμένη οθόνη, καθιστά τη συσκευή ιδανική για ρόλο gateway χωρίς επιπλέον υλικό. Οι βιβλιοθήκες που χρησιμοποιούνται ορίζονται δηλωτικά στο `platformio.ini`:

²Αποθετήριο: <https://github.com/gkalogeitonas/ESP32-LoRa-to-Internet-Gateway>

```

1 lib_deps =
2   sandeepmistry/LoRa@^0.8.0
3   thingpulse/ESP8266 and ESP32 OLED driver for SSD1306@^4.4.0
4   mathieucarbou/ESP Async WebServer@^3.0.6
5   mathieucarbou/AsyncTCP@^3.3.2
6   bblanchon/ArduinoJson@^7.3.0
7   knolleary/PubSubClient@^2.8

```

Listing 59: Εξαρτήσεις PlatformIO του LoRa Gateway

6.6.2 Αρχιτεκτονική Firmware

Το firmware του Gateway οργανώνεται σε έξι αυτόνομα υποσυστήματα, καθένα υλοποιημένο σε ξεχωριστό ζεύγος αρχείων (.h/.cpp):

Υποσύστημα	Ευθύνη
config	Φόρτωση/αποθήκευση ρυθμίσεων (LittleFS JSON)
display	Οδηγός οθόνης OLED SSD1306
battery	Παρακολούθηση μπαταρίας μέσω ADC
lora_radio	Λήψη πακέτων LoRa (SX1276)
wifi_manager	Σύνδεση STA ή δημιουργία AP
web_server	Captive Portal για ρύθμιση μέσω web
forwarder	Προώθηση δεδομένων μέσω MQTT ή HTTP

Πίνακας 15: Υποσυστήματα του LoRa Gateway

Η εκκίνηση (setup()) ακολουθεί αυστηρά καθορισμένη σειρά αρχικοποίησης, όπου κάθε υποσύστημα εξαρτάται από τα προηγούμενα:

```

1 void setup() {
2   Serial.begin(115200);
3
4   configInit();           // 1. Load config from LittleFS
5   displayInit();          // 2. OLED splash screen
6   batteryInit();          // 3. ADC setup
7   loraInit(gConfig);       // 4. SX1276 radio in receive mode
8   wifiInit(gConfig);       // 5. STA connect or AP fallback

```

```
9    wifiScanSync();           // 6. Scan available networks
10   webServerInit();          // 7. Async web server + captive
    portal
11   forwarderInit(gConfig); // 8. MQTT or HTTP client
12 }
```

Listing 60: Σειρά αρχικοποίησης του LoRa Gateway (main.cpp)

6.6.3 Κύριος Βρόχος Εκτέλεσης

Σε αντίθεση με τον κόμβο LoRa που λειτουργεί σε μοντέλο one-shot (Ενότητα 6.5.2), το Gateway παραμένει συνεχώς ενεργό. Ο κύριος βρόχος `loop()` ακολουθεί cooperative αρχιτεκτονική χωρίς RTOS tasks, όπου οι επιμέρους λειτουργίες εκτελούνται περιοδικά μέσω χρονοδρομολόγησης βάσει `millis()`:

```
1  void loop() {
2      unsigned long now = millis();
3
4      // Handle config change flags from web server
5      if (flagWifiChanged) { wifiInit(gConfig); }
6      if (flagLoraChanged) { loraApplySettings(gConfig); }
7      if (flagForwardChanged) { forwarderUpdateConfig(gConfig); }
8      if (flagRestartRequested) { ESP.restart(); }
9
10     // 1. Battery check (every 30s)
11     if (now - lastBatteryCheck >= 30000) {
12         currentBatteryPercent = batteryReadPercent();
13         if (batteryIsCritical() && loraActive) {
14             loraDisable(); // Protect LiPo from deep discharge
15             loraActive = false;
16         }
17     }
18
19     // 2. WiFi management (reconnect if needed)
20     wifiLoop();
21 }
```

```
22 // 3. LoRa packet polling
23 if (loraActive) {
24     LoRaPacket pkt;
25     if (loraPoll(pkt)) {
26         totalPacketsReceived++;
27         if (wifiIsConnected()) {
28             if (forwarderSend(pkt, gConfig))
29                 totalPacketsForwarded++;
30         }
31     }
32 }
33
34 // 4. MQTT keepalive (every 100ms)
35 if (gConfig.use_mqtt && wifiIsConnected())
36     forwarderLoop();
37
38 // 5. Display update (every 1s)
39 if (now - lastDisplayUpdate >= 1000)
40     displayUpdate(/* WiFi, battery, RSSI, counters */);
41 }
```

Listing 61: Κύριος βρόχος του Gateway (main.cpp)

Με αυτόν τον τρόπο, η λήψη πακέτων LoRa δεν εμποδίζεται (blocked) από τις υπόλοιπες λειτουργίες, καθώς το `loraPoll()` εκτελείται σε κάθε iteration του βρόχου.

6.6.4 Διαχείριση Ρυθμίσεων (LittleFS)

Σε αντίθεση με τους κόμβους LoRa/WiFi που χρησιμοποιούν NVS (key-value store), το Gateway αποθηκεύει τις ρυθμίσεις του σε αρχείο `/config.json` στο σύστημα αρχείων LittleFS. Η δομή `GatewayConfig` αποτελεί το μοναδικό σημείο αλήθειας για τη runtime κατάσταση:

```
1 struct GatewayConfig {
2     char wifi_ssid[33];           // WiFi SSID
3     char wifi_pass[65];          // WiFi password
4 }
```



```
5    long    lora_frequency;    // 868 MHz
6    int     lora_sf;           // Spreading Factor (7-12)
7    long    lora_bandwidth;    // 125/250/500 kHz
8    int     lora_coding_rate;  // 4/5 to 4/8
9    uint8_t lora_sync_word;    // 0x12 (private network)
10
11    bool use_mqtt;             // true=MQTT, false=HTTP
12
13    char mqtt_broker[65];      // MQTT broker address
14    int  mqtt_port;            // default: 1883
15    char mqtt_topic[65];      // e.g. "lora/{gw_id}/data"
16    char mqtt_user[33];
17    char mqtt_pass[65];
18
19    char http_url[129];        // HTTP POST endpoint
20    char gateway_id[33];       // Gateway identifier
21 };
```

Listing 62: Δομή ρυθμίσεων Gateway (config.h)

Κατά την εκκίνηση, η `configInit()` προσαρτά (mount) το LittleFS και φορτώνει το `config.json`. Αν δεν υπάρχει αρχείο (πρώτη εκκίνηση), δημιουργούνται προεπιλεγμένες ρυθμίσεις. Η σειριοποίηση γίνεται μέσω `ArduinoJson`, εξασφαλίζοντας ένα ευανάγνωστο και εύκολα τροποποιήσιμο μορφότυπο.

6.6.5 Captive Portal και Web-Based Παραμετροποίηση

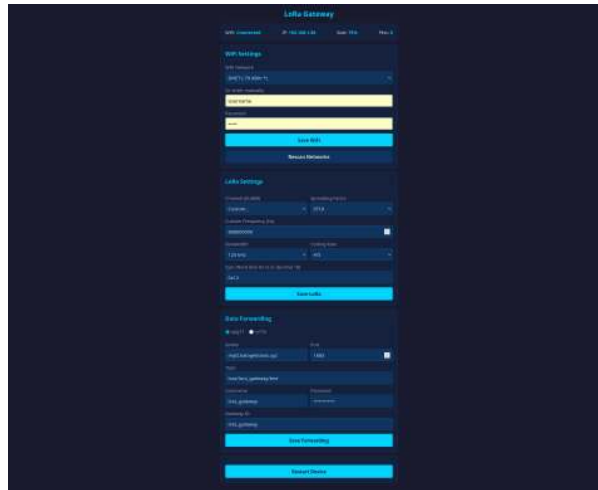
Το Gateway διαθέτει ενσωματωμένη διεπαφή ιστού (web interface) για επιτόπια ρύθμιση χωρίς ανα-μεταγλώττιση. Αν δεν υπάρχουν έγκυρα WiFi credentials, η συσκευή δημιουργεί αυτόματα Access Point με SSID `LoRa-GW-Setup` και ενεργοποιεί DNS wildcard redirection, ώστε κάθε αίτημα DNS να κατευθύνεται στη σελίδα ρύθμισης.

Η διεπαφή περιλαμβάνει τρεις ενότητες:

1. **Ρυθμίσεις WiFi:** Επιλογή δικτύου από λίστα σαρωμένων SSID ή χειροκίνητη εισαγωγή, κωδικός πρόσβασης, επανασάρωση.
2. **Ρυθμίσεις LoRa:** Επιλογή από 8 προρυθμισμένα κανάλια EU868 ή εισαγωγή προ-

σαρμοσμένης συχνότητας, Spreading Factor (7–12), εύρος ζώνης (125/250/500 kHz), Coding Rate (4/5 έως 4/8), λέξη συγχρονισμού.

3. **Προώθηση δεδομένων:** Επιλογή μεταξύ MQTT και HTTP, ρυθμίσεις broker/topic/credentials ή URL αντίστοιχα, αναγνωριστικό gateway.



Σχήμα 29: Διεπαφή ρύθμισης του LoRa Gateway — ρυθμίσεις WiFi, LoRa και προώθησης δεδομένων.

Ασύγχρονος Μηχανισμός Ενημέρωσης.

Ο διακομιστής ιστού βασίζεται στην ασύγχρονη βιβλιοθήκη ESPAsyncWebServer. Κατά την υποβολή μιας φόρμας, ο handler **δεν εφαρμόζει** άμεσα τις αλλαγές. Αντ' αυτού, αποθηκεύει τη νέα ρύθμιση στο LittleFS και θέτει σημαίες (flags) — `flagWifiChanged`, `flagLoraChanged`, `flagForwardChanged` — τις οποίες ο κύριος βρόχος `loop()` ελέγχει και εκτελεί τις κατάλληλες ενέργειες (Listing 61). Η αρχιτεκτονική αυτή αποφεύγει blocking εργασίες εντός ασύγχρονων callbacks.

6.6.6 Λήψη και Προώθηση Πακέτων

Ο πυρήνας λειτουργίας του Gateway είναι η λήψη πακέτων LoRa και η μετατροπή τους σε JSON για προώθηση μέσω Internet. Ο οδηγός ραδιοσυχνότητας λειτουργεί αποκλειστικά σε λειτουργία λήψης (`LoRa.receive()`), σε αντίθεση με τον κόμβο LoRa που λειτουργεί μόνο σε λειτουργία εκπομπής.

ομή *LoRaPacket*.

Κάθε ληφθέν πακέτο αποθηκεύεται σε δομή `LoRaPacket` που περιλαμβάνει τα ωμά bytes (`raw[]`), μέγεθος, μεταδεδομένα (RSSI, SNR), ρυθμίσεις ραδιοσυχνότητας και τοπική

χρονοσφραγίδα (millis()).

Κατασκευή JSON Payload.

Ο forwarder μετατρέπει κάθε LoRaPacket σε JSON, κωδικοποιώντας τα ωμά bytes ως hex string (raw_payload). Το κρυπτογραφημένο περιεχόμενο μεταφέρεται αυτούσιο, χωρίς αποσύνθεση ή ερμηνεία:

```
1 static String buildPayload(const LoRaPacket &pkt,
2                             const GatewayConfig &cfg) {
3     JsonDocument doc;
4     doc["gateway_id"]      = cfg.gateway_id;
5     doc["timestamp"]       = pkt.timestamp;
6     doc["rssi"]            = pkt.rssi;
7     doc["snr"]             = pkt.snr;
8     doc["spreading_factor"] = pkt.sf;
9     doc["frequency"]       = pkt.frequency;
10    doc["packet_size"]      = pkt.size;
11
12    // Raw binary payload as hex string
13    String hexPayload;
14    hexPayload.reserve(pkt.size * 2);
15    for (int i = 0; i < pkt.size; i++) {
16        char buf[3];
17        snprintf(buf, sizeof(buf), "%02X", pkt.raw[i]);
18        hexPayload += buf;
19    }
20    doc["raw_payload"] = hexPayload;
21
22    String output;
23    serializeJson(doc, output);
24    return output;
25 }
```

Listing 63: Κατασκευή JSON payload από ληφθέν πακέτο LoRa (forwarder.cpp)

Η χρήση hex string αντί Base64 ή δυαδικής μεταφοράς επιλέχθηκε για μέγιστη αναγνω-

σιμότητα κατά τον εντοπισμό σφαλμάτων (debugging), ενώ ταυτόχρονα έχουμε αλλοίωση μεταφορά δυαδικού κρυπτοκείμενου μέσω JSON.

Λειτουργία MQTT.

Σε λειτουργία MQTT (`use_mqtt = true`), το Gateway συνδέεται στον EMQX broker μέσω PubSubClient και δημοσιεύει στο θέμα `lorawan/{gateway_id}/data`. Ο μηχανισμός Rule Engine του EMQX αναλαμβάνει την αυτόματη προώθηση (webhook) στο endpoint `/api/v1/lorawan/webhook` του backend.

Λειτουργία HTTP.

Σε λειτουργία HTTP (`use_mqtt = false`), το ίδιο JSON αποστέλλεται απευθείας μέσω `HTTPClient::POST()` στο ρυθμισμένο endpoint. Η λειτουργία αυτή είναι χρήσιμη για εργαστηριακές δοκιμές χωρίς broker ή για απευθείας σύνδεση με το backend webhook.

```
1 bool forwarderSend(const LoRaPacket &pkt,
2                   const GatewayConfig &cfg) {
3     String payload = buildPayload(pkt, cfg);
4
5     if (cfg.use_mqtt) {
6         return mqttSend(payload, cfg); // PubSubClient
7     } else {
8         return httpSend(payload, cfg); // HTTPClient POST
9     }
10 }
```

Listing 64: Διπλή λειτουργία προώθησης: MQTT ή HTTP (forwarder.cpp)

6.6.7 Οθόνη OLED και Επιτόπια Διάγνωση

Η ενσωματωμένη οθόνη OLED 128×64 μας παρέχει ένα εργαλείο επιτόπιας διάγνωσης σε αγροτικό περιβάλλον, όπου δεν υπάρχει πάντα πρόσβαση σε σειριακό τερματικό (serial monitor). Η σελίδα dashboard εμφανίζει σε πραγματικό χρόνο:

- Κατάσταση δικτύου: εικονίδιο WiFi, διεύθυνση IP ή ένδειξη AP Mode.
- Ποσοστό μπαταρίας (αριθμητικά και γραφικά), με ένδειξη κρίσιμου επιπέδου.
- Μεταδεδομένα τελευταίου πακέτου: RSSI και Spreading Factor.

- Μετρητές: αριθμός ληφθέντων (Rx) και προωθημένων (Fwd) πακέτων.
- Ρυθμιζόμενο κανάλι και SF λειτουργίας.



Σχήμα 30: Οθόνη OLED του LoRa Gateway κατά τη λειτουργία — ένδειξη WiFi, μπαταρίας, SF/RSSI και μετρητών Rx/Fwd.

6.6.8 Προστασία Μπαταρίας

Η παρακολούθηση μπαταρίας γίνεται μέσω ADC στο GPIO35, αξιοποιώντας τον ενσωματωμένο διαιρέτη τάσης (100K/100K) της TTGO πλακέτας. Η μετατροπή τάσης σε ποσοστό βασίζεται σε τμηματικά γραμμική παρεμβολή (piecewise linear interpolation) επάνω στην καμπύλη εκφόρτισης LiPo (14 σημείων, από 3,0 V = 0% έως 4,2 V = 100%):

```
1 static const float BATT_V[] = {
2     3.00, 3.50, 3.60, 3.70, 3.75, 3.80, 3.85,
3     3.90, 3.95, 4.00, 4.05, 4.10, 4.15, 4.20 };
4 static const int BATT_P[] = {
5     0, 5, 10, 20, 30, 40, 50,
6     60, 70, 80, 85, 90, 95, 100 };
7
8 float batteryReadVoltage() {
9     uint32_t sum = 0;
```

```
10     for (int i = 0; i < 16; i++)
11         sum += analogRead(BATTERY_PIN);
12     float adcAvg = sum / 16.0f;
13     // TTGO: 100K/100K voltage divider on GPIO35
14     return (adcAvg / 4095.0f) * 3.3f * 2.0f;
15 }
```

Listing 65: Μέτρηση μπαταρίας LiPo με τμηματική παρεμβολή (battery.cpp)

Όταν το ποσοστό μπαταρίας πέσει κάτω από 5%, καλείται η `loraDisable()` η οποία θέτει τον πομποδέκτη LoRa σε κατάσταση ύπνου. Με αυτόν τον τρόπο, η συσκευή διατηρεί τη σύνδεση WiFi και τη διεπαφή ιστού (για πρόσβαση εξ αποστάσεως), ενώ προστατεύει το στοιχείο LiPo από βαθιά εκφόρτιση (deep discharge). Η κατάσταση αυτή εμφανίζεται στην οθόνη OLED ως LoRa OFF (low batt).

6.7 Διαχείριση Ενέργειας και Διεπαφή Χρήστη

Η αυτόνομη λειτουργία κόμβων στο πεδίο, συχνά με τροφοδοσία από μπαταρία LiPo ή ηλιακό πάνελ, καθιστά τη διαχείριση ενέργειας κεντρικό σχεδιαστικό ζήτημα. Παράλληλα, η εγκατάσταση στο πεδίο απαιτεί φιλική διεπαφή χρήστη (UI) για παραμετροποίηση και διάγνωση. Η παρούσα ενότητα συγκεντρώνει τους μηχανισμούς ενεργειακής διαχείρισης, παρακολούθησης μπαταρίας, επαναφοράς εργοστασιακών ρυθμίσεων και αλληλεπίδρασης με τον χρήστη, που διατρέχουν εγκάρσια τις τρεις παραλλαγές firmware.

6.7.1 Deep Sleep και Μοντέλα Ενεργειακού Κύκλου

Οι κόμβοι πεδίου ακολουθούν τον κύκλο *μέτρηση* → *αποστολή* → *ύπνος*, ο οποίος περιγράφηκε στην ενότητα 6.4.4. Κατά τη φάση Deep Sleep, ο ESP32-C6 απενεργοποιεί τον κύριο επεξεργαστή και τα περιφερειακά, διατηρώντας μόνο τη μνήμη RTC και τον χρονοδιακόπτη αφύπνισης, με κατανάλωση μόλις 7 μ A [46]. Μετά την επιτυχή αποστολή δεδομένων, ο κόμβος αποσυνδέεται κανονικά (graceful shutdown) από τον MQTT broker και απενεργοποιεί πλήρως τον πομποδέκτη WiFi πριν εισέλθει σε Deep Sleep:

```
1 // Turn off WiFi cleanly to speed shutdown
2 if (mqttEnabled) {
3     mqttClient->disconnect();
4 }
5 WiFi.disconnect(true);
```

```
6  WiFi.mode(WIFI_OFF);
7  delay(50);
8
9  // Enable wakeup from button (LOW = pressed)
10 #if defined(CONFIG_IDF_TARGET_ESP32)
11  esp_sleep_enable_ext0_wakeup((gpio_num_t)BUTTON_PIN, 0);
12 #elif defined(CONFIG_IDF_TARGET_ESP32C6)
13  esp_deep_sleep_enable_gpio_wakeup(
14      1ULL << BUTTON_PIN, ESP_GPIO_WAKEUP_GPIO_LOW);
15 #else
16  esp_sleep_enable_ext1_wakeup(
17      1ULL << BUTTON_PIN, ESP_EXT1_WAKEUP_ANY_LOW);
18 #endif
19 // Also enable timer wakeup
20 esp_sleep_enable_timer_wakeup(sleep_us);
21 esp_deep_sleep_start();
```

Listing 66: Είσοδος σε Deep Sleep με διπλή αφύπνιση (main.cpp)

Η υλοποίηση χρησιμοποιεί **διπλή πηγή αφύπνισης**: χρονοδιακόπτη (timer_wakeup) για τον κανονικό κύκλο μέτρησης, και εξωτερικό πλήκτρο (GPIO wakeup) για χειροκίνητη αφύπνιση ή επαναφορά εργοστασιακών ρυθμίσεων (§6.7.3). Ο μηχανισμός αφύπνισης μέσω πλήκτρου δηλώνεται με μεταγλώττιση υπό συνθήκη (#if defined), καθώς κάθε παραλλαγή ESP32 χρησιμοποιεί διαφορετικό API:

- **ESP32 κλασικό:** `esp_sleep_enable_ext0_wakeup()` — αφύπνιση από μεμονωμένο RTC GPIO.
- **ESP32-C6 (RISC-V):** `esp_deep_sleep_enable_gpio_wakeup()` — ειδικό API βάθους ύπνου για αυθαίρετα GPIO. Η αρχιτεκτονική του ESP32-C6 δεν υποστηρίζει ext0/ext1, αλλά παρέχει αντίστοιχη λειτουργία μέσω της μονάδας διαχείρισης ενέργειας (PMU) [46].
- **Λοιπές παραλλαγές (S2, S3, C3):** `esp_sleep_enable_ext1_wakeup()` — αφύπνιση από μάσκα GPIO με λογική ANY_LOW.

Ενεργειακή Αποδοτικότητα μέσω Σχεδίασης.

Σε όλες τις περιπτώσεις, η ενεργειακή αποδοτικότητα επιτυγχάνεται κυρίως μέσω *αρχιτεκτονικής σχεδίασης* και όχι μέσω επιθετικής βελτιστοποίησης χαμηλού επιπέδου. Η ελαχιστοποίηση του χρόνου ενεργού κύκλου (duty cycle) — σύνδεση, μέτρηση, αποστολή, ύπνος — σχεδιαστική ώστε να μειώνει την κατανάλωση ενέργειας. Το διαμορφώσιμο διάστημα μέτρησης (SENSORS_READ_INTERVAL_MS, προεπιλογή 10 λεπτά) επιτρέπει στον χρήστη να προσαρμόσει τη συχνότητα αποστολής στις ανάγκες της εφαρμογής.

6.7.2 Παρακολούθηση Στάθμης Μπαταρίας

Η παρακολούθηση της στάθμης μπαταρίας είναι αναπόσπαστο μέρος τόσο του firmware των κόμβων όσο και της διαδικτυακής εφαρμογής (backend). Υλοποιείται ως κανονικός αισθητήρας μέσω του μηχανισμού εγγραφής εργοστασίου (ενότητα 6.2), με τη δήλωση `BatteryLevelSensor` στο `device_profile.h`.

Υλοποίηση στον *FireBeetle 2 ESP32-C6*.

Η κλάση `BatteryLevelSensor` αξιοποιεί τον ενσωματωμένο διαιρέτη τάσης (αναλογία 2:1) του *FireBeetle* και τη λειτουργία `analogReadMilliVolts()` για βελτιωμένη ακρίβεια. Η μέτρηση βασίζεται σε μέσο όρο 10 δειγμάτων:

```
1 constexpr int NUM_SAMPLES = 10;
2 long sumMv = 0;
3 for (int i = 0; i < NUM_SAMPLES; ++i) {
4     int mv = analogReadMilliVolts(adc_pin_);
5     sumMv += mv;
6     delay(10);
7 }
8 int avgMv = (int)(sumMv / NUM_SAMPLES);
9 // Built-in voltage divider: actual = avgMv * 2
10 float batteryPercent = voltageToBatteryPercent(avgMv * 2);
```

Listing 67: Μέτρηση μπαταρίας LiPo στον *FireBeetle C6* (`battery_level.cpp`)

Η μετατροπή τάσης σε ποσοστό πραγματοποιείται μέσω τμηματικά γραμμικής προσέγγισης (piecewise linear approximation) της καμπύλης εκφόρτισης LiPo:

- 4,2 V – 4,0 V → 100% – 90% (ταχεία πτώση κορυφής)

- $4,0\text{ V} - 3,7\text{ V} \rightarrow 90\% - 50\%$ (σταδιακή μείωση)
- $3,7\text{ V} - 3,4\text{ V} \rightarrow 50\% - 20\%$ (μέτρια μείωση)
- $3,4\text{ V} - 3,0\text{ V} \rightarrow 20\% - 0\%$ (ταχεία πτώση βάσης)

Η τιμή αποστέλλεται στο backend ως κανονική μέτρηση αισθητήρα (τύπου battery), επιτρέποντας την απομακρυσμένη παρακολούθηση μέσω της διαδικτυακής εφαρμογής. Στο Σχήμα 31 φαίνεται η σελίδα συσκευής στην πλατφόρμα, η οποία εμφανίζει το ποσοστό μπαταρίας (79,6%), τα MQTT credentials, οι αισθητήρες με τις τελευταίες τιμές, και ιστορικό μετρήσεων μπαταρίας.

Devices > firebeetle-2-esp32-c6

UID: Test-Device-2 online
Type: firebeetle-2-esp32-c6
wifi Battery: 79.6%

Created At: 2026-01-04T17:47:42.000000Z Last Seen: 2026-02-01 18:40:06

MQTT Credentials Show

Username: Test-Device-2 Password: *****

Sensors
All sensors registered to this device

Υγρασία firebeetle DHT20 Type: humidity
UID: DHT20-Hum-1 Lab: 370582676, Lot: 25-4074630
Last seen: 2026-02-01 18:40:06
Last value: 69.79

Θερμοκρασία firebeetle Type: temperature
UID: DHT20-Temp-1 Lab: 370582676, Lot: 25-4074630
Last seen: 2026-02-01 18:40:06
Last value: 17.23

Device1 Battery Type: battery
UID: Battery-Level-1 Lab: 370582676, Lot: 25-4074630
Last seen: 2026-02-01 18:40:06
Last value: 79.60

Battery
Device battery level and recent readings

Current: **79.6%**

Recent readings (last 5):

- 2026-02-01 18:40:06 — 79.6
- 2026-02-01 18:36:56 — 79.6
- 2026-02-01 18:33:47 — 79.6
- 2026-02-01 18:27:27 — 79.6
- 2026-02-01 18:24:18 — 78.8

[Back to List](#)

Σχήμα 31: Σελίδα συσκευής στην πλατφόρμα Agronos — εμφανίζεται η στάθμη μπαταρίας (79,6%), τα MQTT credentials, οι αισθητήρες με τις τελευταίες τιμές, και ιστορικό μετρήσεων μπαταρίας.

6.7.3 Μηχανισμός Επαναφοράς Εργοστασιακών Ρυθμίσεων

Καθώς οι κόμβοι εγκαθίστανται σε δυσπρόσιτα σημεία, απαιτείται ένας μηχανισμός επαναφοράς (factory reset) που δεν προϋποθέτει σύνδεση υπολογιστή ή επαναφόρτωση firmware. Αυτό επιτυγχάνεται μέσω **παρατεταμένου πατήματος** του ενσωματωμένου πλήκτρου κατά την εκκίνηση:

```
1 static void checkButtonReset() {  
2     if (digitalRead(BUTTON_PIN) == LOW) {  
3         unsigned long pressStart = millis();  
4         while (digitalRead(BUTTON_PIN) == LOW) {  
5             if (millis() - pressStart >= BUTTON_LONG_PRESS_MS) {  
6                 storage.clearAll(); // Wipe WiFi, JWT, MQTT  
5                 creds  
7                 delay(1000);  
8                 ESP.restart();  
9             }  
10            delay(100);  
11        }  
12    }  
13 }
```

Listing 68: Μηχανισμός επαναφοράς εργοστασιακών ρυθμίσεων (main.cpp / main_lora.cpp)

Ο μηχανισμός εκτελείται πρώιμα στη `setup()`, πριν από οποιαδήποτε σύνδεση δικτύου, και λειτουργεί ως εξής:

1. Κατά την εκκίνηση, το firmware ελέγχει αν το πλήκτρο είναι πατημένο (LOW).
2. Αν ναι, μετρά τη χρονική διάρκεια κράτησης.
3. Αν η διάρκεια υπερβεί τα `BUTTON_LONG_PRESS_MS` (10 δευτερόλεπτα), καλεί `storage.clearAll()` που διαγράφει:
 - Διαπιστευτήρια WiFi (SSID / κωδικός)
 - JWT token αυθεντικοποίησης

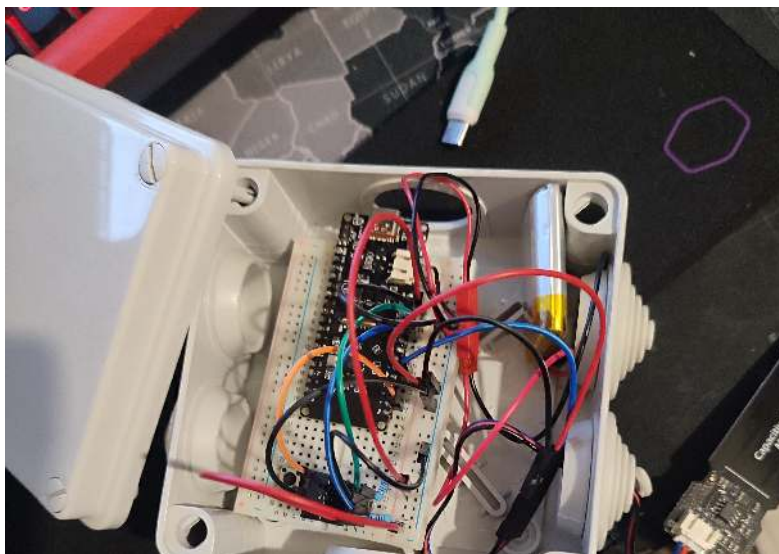
- Διαπιστευτήρια MQTT (username / password)

4. Η συσκευή επανεκκινεί (`ESP.restart()`) και, χωρίς αποθηκευμένα διαπιστευτήρια, εκκινεί αυτόματα το Captive Portal (ενότητα 6.4.1) για νέα παραμετροποίηση.

Το πλήκτρο `BUTTON_PIN` διαφέρει ανά πλακέτα (`GPIO 0` στον `TTGO LoRa32`, `GPIO 7` στον `ESP32-C6`, `GPIO 14` στον κλασικό `ESP32`) και ορίζεται κεντρικά στο `config.h` μέσω μεταγλώττισης υπό συνθήκη. Η ίδια συνάρτηση `checkButtonReset()` χρησιμοποιείται τόσο στο `main.cpp` (WiFi) όσο και στο `main_lora.cpp` (LoRa).

6.7.4 Φυσική Εγκατάσταση

Για την εγκατάσταση στο πεδίο, οι κόμβοι WiFi τοποθετούνται σε αδιάβροχο ηλεκτρολογικό κουτί προστασίας `IP65`. Στο εσωτερικό στερεώνονται η αναπτυξιακή πλακέτα (`FireBeetle 2 ESP32-C6`), η πλακέτα δοκιμών (`breadboard`) για τη σύνδεση αισθητήρων, και η μπαταρία `LiPo`. Τα καλώδια αισθητήρων (εδαφικής υγρασίας, θερμοκρασίας/υγρασίας αέρα) περνούν μέσα από τα στυπιοθλίπτες (`cable glands`) του κουτιού.



Σχήμα 32: Κόμβος WiFi σε αδιάβροχο κουτί `IP65` — `FireBeetle 2 ESP32-C6` με μπαταρία `LiPo`, `breadboard` και αισθητήρες.

Σύνοψη

Στο κεφάλαιο αυτό τεκμηριώθηκε η υλοποίηση του υλικολογισμικού της πλατφόρμας *Agropos* σε τρεις παραλλαγές. Ο κόμβος WiFi είναι η κύρια παραλλαγή ανάπτυξης: συνδυάζει τον αυτόματο μηχανισμό παραμετροποίησης μέσω Captive Portal, ταυτοποίηση

με JWT και αξιόπιστη αποστολή δεδομένων μέσω MQTT με εναλλακτική επιστροφή σε HTTP. Ο κόμβος LoRa στοχεύει σε αγροτικές εγκαταστάσεις μεγάλης εμβέλειας χωρίς υποδομή WiFi: το ελαφρύ πρωτόκολλο μηνυμάτων με κρυπτογράφηση AES-128-CTR και ο μετρητής πλαισίων εξασφαλίζουν εμπιστευτικότητα και προστασία από επαναληπτικές επιθέσεις. Το Gateway μεταφράζει τα LoRa πακέτα σε μορφή κατανοητή από το backend, αναλαμβάνοντας την προώθησή τους. Η κοινή αρχιτεκτονική — επίπεδο αφαίρεσης αισθητήρων, factory αυτο-καταχώρησης, αρθρωτά υποσυστήματα και διαχείριση ενέργειας μέσω Deep Sleep — διαπερνά και τις τρεις παραλλαγές, περιορίζοντας την επανάληψη κώδικα και διευκολύνοντας μελλοντικές επεκτάσεις.

7 Δοκιμές, Μετρήσεις & Αξιολόγηση

Το παρόν κεφάλαιο παρουσιάζει την πειραματική αξιολόγηση της πλατφόρμας Agropos σε δύο επίπεδα: τεχνικό και χρηστικό. Στο τεχνικό επίπεδο μετράται η ενεργειακή αυτονομία των κόμβων πεδίου υπό διαφορετικές διαμορφώσεις αισθητήρων και πρωτοκόλλων επικοινωνίας. Στο χρηστικό επίπεδο αξιολογείται η ευχρηστία των βασικών ροών της εφαρμογής μέσω πιλοτικής μελέτης.

7.1 Μεθοδολογία Πειραμάτων

Για την αξιολόγηση της πλατφόρμας Agropos σε πραγματικές συνθήκες πεδίου, σχεδιάστηκε και εκτελέστηκε μια σειρά πειραμάτων με κύριο άξονα την ενεργειακή αυτονομία του κόμβου. Κάθε πείραμα μεταβάλλει μία ή λίγες παραμέτρους ως προς ένα κοινό σημείο αναφοράς (baseline), ώστε τα αποτελέσματα να είναι συγκρίσιμα μεταξύ τους.

7.1.1 Πειράματα Κόμβου WiFi/MQTT

Πείραμα 1: Αυτονομία Μπαταρίας — Πλήρης Εξοπλισμός

Σκοπός.

Μέτρηση της ενεργειακής αυτονομίας του κόμβου σε πραγματικές συνθήκες αγρού, με ενεργοποιημένους όλους τους αισθητήρες και αποστολή δεδομένων μέσω MQTT, υπό την εποπτεία του χρήστη.

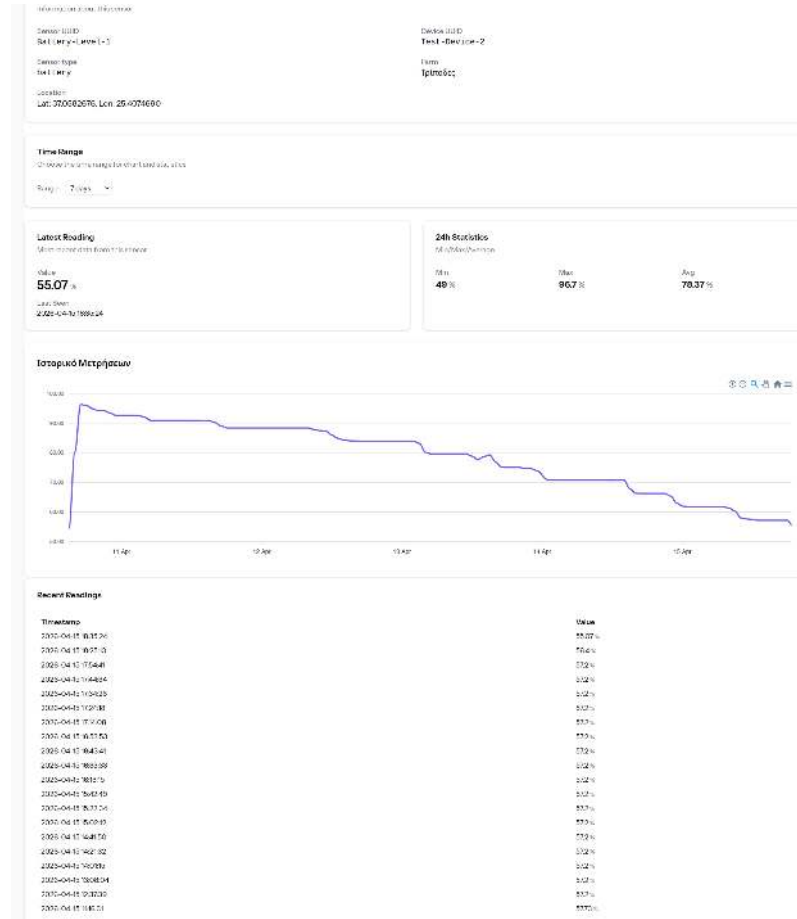
Εξοπλισμός & Παράμετροι.

Πίνακας 16: Παράμετροι Πειράματος 1.

Παράμετρος	Τιμή
Πλακέτα	DFRobot FireBeetle 2 ESP32-C6
Πρωτόκολλο	MQTT
Περίοδος αποστολής	10 min
Αισθητήρες	DHT20 + Αισθητήρας υγρασίας εδάφους
Δείγματα μ.ό. ανά μέτρηση	10 (καθυστέρηση 10 ms μεταξύ δειγμάτων)
Μπαταρία	LiPo 1200 mAh
Διάρκεια	5 ημέρες (10–15/04/2026)

Αποτελέσματα.

Μετά από πέντε ημέρες συνεχούς λειτουργίας, η στάθμη της μπαταρίας μειώθηκε στο **55,07 %**. Η πτώση ήταν σχεδόν γραμμική, όπως φαίνεται στο Σχήμα 33.



Σχήμα 33: Πτώση ποσοστού μπαταρίας κατά τη διάρκεια 5 ημερών (Πείραμα 1).

Πείραμα 2: Αυτονομία Μπαταρίας — Συσκευή χωρίς αισθητήρες

Σκοπός.

Μέτρηση της ενεργειακής κατανάλωσης του κόμβου όταν λειτουργεί χωρίς συνδεδεμένους αισθητήρες, ώστε να απομονωθεί η κατανάλωση της ίδιας της συσκευής και του μηχανισμού επικοινωνίας και να εκτιμηθεί η συμβολή των αισθητήρων στη συνολική αποφόρτιση.

Εξοπλισμός & Παράμετροι.

Πίνακας 17: Παράμετροι Πειράματος 2.

Παράμετρος	Τιμή
Πλακέτα	DFRobot FireBeetle 2 ESP32-C6
Πρωτόκολλο	MQTT
Περίοδος αποστολής	10 min
Αισθητήρες	Κανένας
Μπαταρία	LiPo 1200 mAh
Διάρκεια	5 ημέρες (15–20/04/2026)

Αποτελέσματα.

Μετά από πέντε ημέρες συνεχούς λειτουργίας χωρίς αισθητήρες, το επίπεδο της μπαταρίας καταγράφηκε στο **89,73 %**.



Σχήμα 34: Πτώση ποσοστού μπαταρίας κατά τη διάρκεια 5 ημερών χωρίς αισθητήρες (Πείραμα 2).

Παρατηρήσεις.

Το αποτέλεσμα αυτό δείχνει σημαντικά μικρότερη κατανάλωση σε σχέση με το Πείραμα 1, όπου για την ίδια διάρκεια και με πλήρη εξοπλισμό η στάθμη είχε μειωθεί στο **55,07 %**. Η μεγάλη διαφορά σε σχέση με το Πείραμα 1 υποδεικνύει ότι σημαντικό μέρος της συνολικής κατανάλωσης οφείλεται στους αισθητήρες και στη διαδικασία λήψης μετρήσεων, και

όχι μόνο στην πλακέτα ή στη μετάδοση μέσω MQTT. Επομένως, το Πείραμα 2 λειτουργεί ως σημείο αναφοράς για την αποτίμηση της ενεργειακής επιβάρυνσης που εισάγει ο εξοπλισμός πεδίου.

Πείραμα 3: Μεμονωμένος Αισθητήρας DHT20

Σκοπός.

Απομόνωση της ενεργειακής συμβολής του ψηφιακού αισθητήρα θερμοκρασίας και υγρασίας DHT20, ώστε να αξιολογηθεί αν η ενσωμάτωσή του επιφέρει μετρήσιμη αύξηση της κατανάλωσης σε σχέση με το σημείο αναφοράς (Πείραμα 2 — κόμβος χωρίς αισθητήρες).

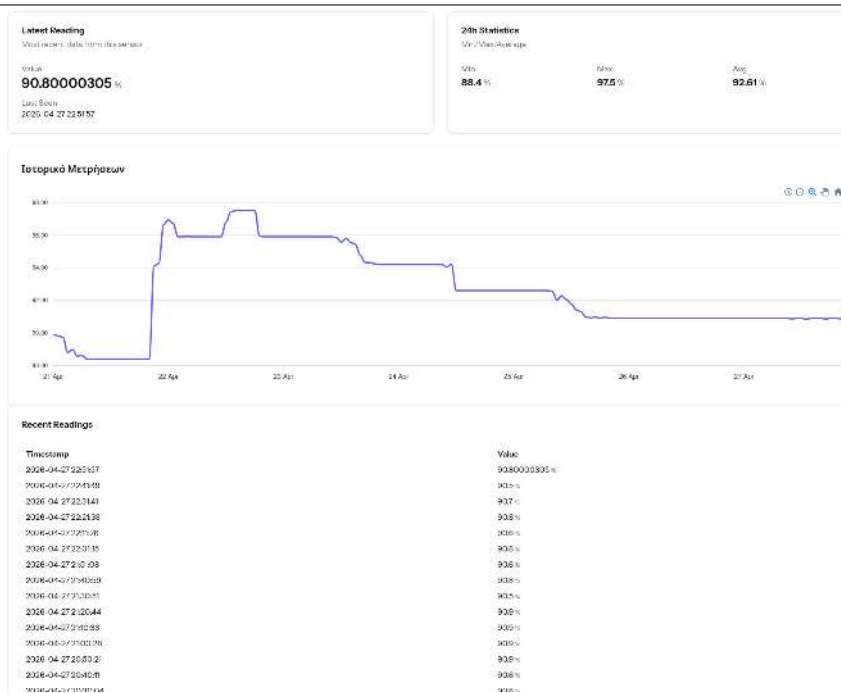
Εξοπλισμός & Παράμετροι.

Πίνακας 18: Παράμετροι Πειράματος 3.

Παράμετρος	Τιμή
Πλακέτα	DFRobot FireBeetle 2 ESP32-C6
Πρωτόκολλο	MQTT
Περίοδος αποστολής	10 min
Αισθητήρες	Μόνο DHT20
Δείγματα μ.ό. ανά μέτρηση	10 (καθυστέρηση 10 ms μεταξύ δειγμάτων)
Μπαταρία	LiPo 1200 mAh
Διάρκεια	5 ημέρες (22–27/04/2026)

Αποτελέσματα.

Μετά από 5 ημέρες συνεχούς λειτουργίας, η στάθμη της μπαταρίας καταγράφηκε στο **90,80 %**.



Σχήμα 35: Πτώση ποσοστού μπαταρίας κατά τη διάρκεια 5 ημερών με μεμονωμένο αισθητήρα DHT20 (Πείραμα 3).

Παρατηρήσεις.

Συγκρίνοντας το αποτέλεσμα αυτό με τη μέτρηση του Πειράματος 2 (89,73 %), όπου η συσκευή λειτούργησε χωρίς συνδεδεμένους αισθητήρες, παρατηρείται ότι η τελική στάθμη της μπαταρίας παραμένει στα ίδια πρακτικά επίπεδα. Η ελάχιστη απόκλιση μεταξύ των δύο μετρήσεων (η οποία εμπίπτει στα όρια ανοχής και πιθανότατα οφείλεται σε φυσιολογικές διακυμάνσεις του κυκλώματος μέτρησης ADC, σε επιδράσεις της θερμοκρασίας, ή σε ελαφρώς διαφορετική αρχική τάση κατά την πλήρη φόρτιση) υποδεικνύει ότι ο ψηφιακός αισθητήρας DHT20 έχει εξαιρετικά χαμηλές ενεργειακές απαιτήσεις. Από τα παραπάνω προκύπτει το συμπέρασμα ότι η ενσωμάτωση και η περιοδική λήψη μετρήσεων από τον εν λόγω αισθητήρα δεν επιφέρει καμία αξιοσημείωτη περαιτέρω ενεργειακή επιβάρυνση στον κόμβο σε σχέση με τη βασική κατανάλωση της σκέτης συσκευής.

Πείραμα 4: Μεμονωμένος Αισθητήρας Υγρασίας Εδάφους

Σκοπός.

Απομόνωση της ενεργειακής συμβολής του αναλογικού αισθητήρα υγρασίας εδάφους, ώστε να προσδιοριστεί ποιος από τους δύο αισθητήρες αποτελεί την κύρια πηγή κατανάλωσης ενέργειας στον πλήρως εξοπλισμένο κόμβο (Πείραμα 1).

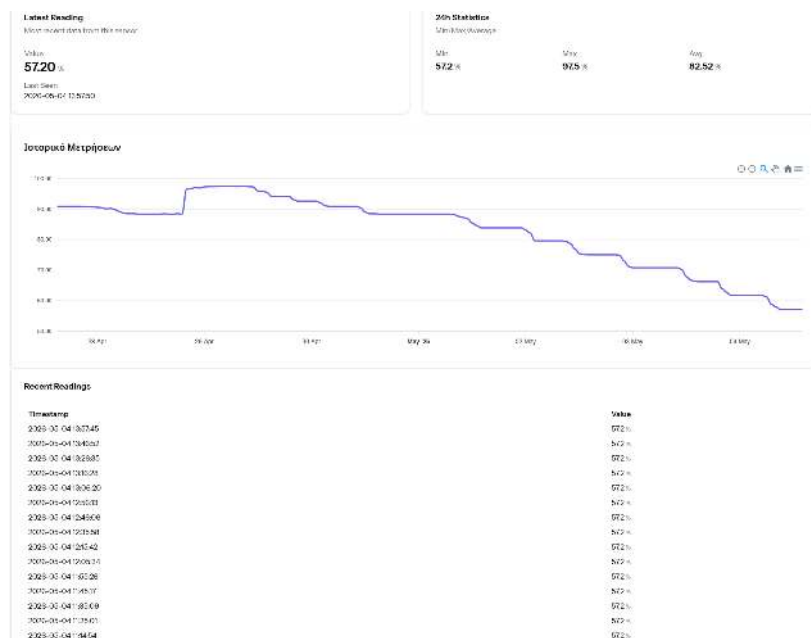
Εξοπλισμός & Παράμετροι.

Πίνακας 19: Παράμετροι Πειράματος 4.

Παράμετρος	Τιμή
Πλακέτα	DFRobot FireBeetle 2 ESP32-C6
Πρωτόκολλο	MQTT
Περίοδος αποστολής	10 min
Αισθητήρες	Μόνο αισθητήρας υγρασίας εδάφους
Δείγματα μ.ό. ανά μέτρηση	10 (καθυστέρηση 10 ms μεταξύ δειγμάτων)
Μπαταρία	LiPo 1200 mAh
Διάρκεια	5 ημέρες (29/04–04/05/2026)

Αποτελέσματα.

Μετά από 5 ημέρες συνεχούς λειτουργίας, η στάθμη της μπαταρίας καταγράφηκε στο **57,20 %**.



Σχήμα 36: Πτώση ποσοστού μπαταρίας κατά τη διάρκεια 5 ημερών με μεμονωμένο αισθητήρα υγρασίας εδάφους (Πείραμα 4).

Παρατηρήσεις.

Συγκρίνοντας το αποτέλεσμα αυτό (57,20 %) με τις προηγούμενες μετρήσεις, καθίσταται σαφές ότι η κύρια πηγή ενεργειακής κατανάλωσης στον κόμβο προέρχεται από τον αναλογικό αισθητήρα υγρασίας εδάφους. Η τελική στάθμη της μπαταρίας βρίσκεται πολύ κοντά σε αυτή του Πειράματος 1 (55,07 %), όπου λειτουργούσε το σύνολο του εξοπλισμού. Αντίθετα, στα Πειράματα 2 (συσκευή χωρίς αισθητήρες) και 3 (συσκευή μόνο με τον ψηφιακό DHT20), η μπαταρία διατηρήθηκε σε επίπεδα περίπου 90 %. Η αυξημένη αυτή κατανάλωση εξηγείται από τη φύση της μέτρησης: ο αισθητήρας υγρασίας εδάφους απαιτεί ενεργοποίηση του μετατροπέα αναλογικού-σε-ψηφιακό (ADC) του μικροελεγκτή, καθώς και τη λήψη πολλαπλών δειγμάτων με χρονικές καθυστερήσεις για τον υπολογισμό του μέσου όρου, αντλώντας σημαντικά μεγαλύτερο ρεύμα κατά τον χρόνο που παραμένει ενεργός ο κόμβος.

7.1.2 Συγκριτική Αξιολόγηση Πρωτοκόλλων Επικοινωνίας

Στόχος της ενότητας αυτής είναι η σύγκριση της ενεργειακής κατανάλωσης τριών διαφορετικών πρωτοκόλλων επικοινωνίας: LoRa, WiFi/MQTT και WiFi/HTTP. Για να εξαιρεθεί η επίδραση διαφορών στο κύκλωμα, χρησιμοποιείται το ίδιο υλικό σε όλα τα πειράματα — η πλακέτα TTGO LoRa32 v2.1 868MHz — χωρίς συνδεδεμένους αισθητήρες. Έτσι, η ανεξάρτητη μεταβλητή είναι αποκλειστικά το πρωτόκολλο επικοινωνίας, ενώ η εξαρτημένη μεταβλητή είναι η ενεργειακή κατανάλωση που εκφράζεται ως πτώση στάθμης μπαταρίας σε διάστημα 5 ημερών.

Πείραμα B1: Αποστολή μέσω LoRa

Σκοπός.

Μέτρηση της ενεργειακής κατανάλωσης της πλακέτας TTGO LoRa32 v2.1 868MHz κατά τη λειτουργία της με το πρωτόκολλο LoRa, χωρίς συνδεδεμένους αισθητήρες. Το πείραμα αποτελεί το σημείο αναφοράς (baseline) για τη σύγκριση με τα πρωτόκολλα WiFi/MQTT (Πείραμα B2) και WiFi/HTTP (Πείραμα B3).

Εξοπλισμός & Παράμετροι.

Πίνακας 20: Παράμετροι Πειράματος B1.

Παράμετρος	Τιμή
Πλακέτα	TTGO LoRa32 v2.1 868MHz
Πρωτόκολλο	LoRa
Αισθητήρες	Κανέναν
Διάρκεια	5 ημέρες

Αποτελέσματα.

Μετά από πέντε ημέρες συνεχούς λειτουργίας, η στάθμη της μπαταρίας καταγράφηκε στο 96,7 %.

Παρατηρήσεις.

Το αποτέλεσμα αυτό δείχνει πολύ μικρή αποφόρτιση, γεγονός που αναδεικνύει το LoRa ως το πιο αποδοτικό ενεργειακά πρωτόκολλο από τα τρία που εξετάζονται. Η μέτρηση αυτή αποτελεί τη γραμμή βάσης (baseline) για τη συγκριτική αξιολόγηση των Πειραμάτων B2 και B3.

Πείραμα B2: Αποστολή μέσω WiFi/MQTT**Σκοπός.**

Μέτρηση και αξιολόγηση της ενεργειακής κατανάλωσης της πλακέτας TTGO LoRa32 v2.1 868MHz όταν λειτουργεί με το υλικολογισμικό (firmware) των κόμβων WiFi και επικοινωνεί μέσω του πρωτοκόλλου MQTT. Το πείραμα αυτό λειτουργεί συμπληρωματικά με το Πείραμα B1, αξιοποιώντας το ίδιο ακριβώς υλικό, αλλάζοντας μόνο το πρωτόκολλο επικοινωνίας. Στόχος είναι η άμεση και δίκαιη συγκριτική αποτίμηση της ενεργειακής επιβάρυνσης του WiFi/MQTT έναντι του LoRa (Πείραμα B1), χωρίς να επηρεάζονται τα αποτελέσματα από διαφορές στο κύκλωμα της πλακέτας.

Εξοπλισμός & Παράμετροι.

Πίνακας 21: Παράμετροι Πειράματος B2.

Παράμετρος	Τιμή
Πλακέτα	TTGO LoRa32 v2.1 868MHz
Πρωτόκολλο	MQTT (μέσω WiFi)
Περίοδος αποστολής	10 min
Αισθητήρες	Κανένας
Διάρκεια	5 ημέρες (30/04–05/05/2026)

Αποτελέσματα.

Μετά από 5 ημέρες συνεχούς λειτουργίας, η στάθμη της μπαταρίας καταγράφηκε στο **92,70 %**.

Παρατηρήσεις.

Συγκρίνοντας το αποτέλεσμα αυτό (92,70 %) με το αντίστοιχο του Πειράματος B1 (96,7 %), όπου χρησιμοποιήθηκε ακριβώς το ίδιο υλικό (πλακέτα TTGO LoRa32) αλλά με επικοινωνία LoRa, προκύπτει ξεκάθαρα η ενεργειακή διαφορά των δύο πρωτοκόλλων. Η σύνδεση και μετάδοση μέσω του υποσυστήματος WiFi προκάλεσε μεγαλύτερη αποφόρτιση της μπαταρίας (περίπου 4 ποσοστιαίες μονάδες επιπλέον σε διάστημα 5 ημερών), επιβεβαιώνοντας ότι η μετάδοση WiFi/MQTT είναι πιο ενεργοβόρα διαδικασία από τη μετάδοση LoRa.

Πείραμα B3: Αποστολή μέσω WiFi/HTTP**Σκοπός.**

Μέτρηση και αξιολόγηση της ενεργειακής κατανάλωσης της πλακέτας TTGO LoRa32 v2.1 868MHz όταν λειτουργεί με το υλικολογισμικό (firmware) των κόμβων WiFi και επικοινωνεί μέσω του πρωτοκόλλου HTTP. Το πείραμα αυτό λειτουργεί συμπληρωματικά με το Πείραμα B2, αξιοποιώντας το ίδιο ακριβώς υλικό, αλλάζοντας μόνο το πρωτόκολλο επικοινωνίας. Στόχος είναι η άμεση και δίκαιη συγκριτική αποτίμηση της ενεργειακής επιβάρυνσης του WiFi/HTTP έναντι του WiFi/MQTT (Πείραμα B2) και του LoRa (Πείραμα B1), χωρίς να επηρεάζονται τα αποτελέσματα από διαφορές στο κύκλωμα της πλακέτας.

Εξοπλισμός & Παράμετροι.

Πίνακας 22: Παράμετροι Πειράματος B3.

Παράμετρος	Τιμή
Πλακέτα	TTGO LoRa32 v2.1 868MHz
Πρωτόκολλο	HTTP (μέσω WiFi)
Περίοδος αποστολής	10 min
Αισθητήρες	Κανένας
Διάρκεια	5 ημέρες (06–11/05/2026)

Αποτελέσματα.

Μετά από 5 ημέρες συνεχούς λειτουργίας, η στάθμη της μπαταρίας καταγράφηκε στο **91,70 %**.

Παρατηρήσεις.

Συγκρίνοντας το αποτέλεσμα αυτό (91,70 %) με τα αντίστοιχα των Πειραμάτων B1 (96,7 %) και B2 (92,70 %), προκύπτει ότι η μετάδοση μέσω HTTP επιφέρει ελαφρώς μεγαλύτερη ενεργειακή κατανάλωση σε σχέση με το MQTT, αλλά και σημαντικά μεγαλύτερη σε σχέση με το LoRa.

7.1.3 Σύνοψη Αποτελεσμάτων Πειραμάτων

Ο Πίνακας 23 συγκεντρώνει τα αποτελέσματα όλων των πειραμάτων, επιτρέποντας την άμεση σύγκριση μεταξύ διαφορετικών διαμορφώσεων αισθητήρων και πρωτοκόλλων επικοινωνίας.

Πίνακας 23: Συγκριτικά αποτελέσματα όλων των πειραμάτων (διάρκεια 5 ημερών ανά πείραμα).

Πείραμα	Πλακέτα	Πρωτόκολλο	Αισθητήρες	Τελική στάθμη	Πτώση
1	FireBeetle	MQTT	DHT20 + Υγρασία εδάφους	55,07 %	44,93 %
2	FireBeetle	MQTT	Κανένας	89,73 %	10,27 %
3	FireBeetle	MQTT	Μόνο DHT20	90,80 %	9,20 %
4	FireBeetle	MQTT	Μόνο υγρασία εδάφους	57,20 %	42,80 %
B1	TTGO LoRa32	LoRa	Κανένας	96,70 %	3,30 %
B2	TTGO LoRa32	MQTT	Κανένας	92,70 %	7,30 %
B3	TTGO LoRa32	HTTP	Κανένας	91,70 %	8,30 %

Από τα συγκεντρωτικά αποτελέσματα προκύπτουν δύο βασικά συμπεράσματα. Πρώτον, ο αναλογικός αισθητήρας υγρασίας εδάφους είναι η κυρίαρχη πηγή ενεργειακής κατανάλωσης στον κόμβο FireBeetle: η παρουσία του (Πείραμα 4) οδηγεί σε πτώση 42,80 %, έναντι 9,20 % του DHT20 (Πείραμα 3) και 10,27 % χωρίς αισθητήρες (Πείραμα 2). Δεύτερον, μεταξύ των τριών πρωτοκόλλων επικοινωνίας που δοκιμάστηκαν υπό ίδιες συνθήκες υλικού (πλακέτα TTGO LoRa32, χωρίς αισθητήρες), το LoRa εμφανίζεται ως το πλέον ενεργειακά αποδοτικό (πτώση 3,30 %), ακολουθούμενο από το WiFi/MQTT (7,30 %) και το WiFi/HTTP (8,30 %).

7.2 Πιλοτική Μελέτη Χρηστικότητας

Εκτός από τις τεχνικές μετρήσεις που αφορούν την κατανάλωση ενέργειας, την αξιοπιστία επικοινωνίας και τη συμπεριφορά των κόμβων πεδίου, κρίθηκε απαραίτητη και η αξιολόγηση της πλατφόρμας από την πλευρά του τελικού χρήστη. Για τον σκοπό αυτό σχεδιάστηκε μία **Πιλοτική Μελέτη Χρηστικότητας** (Pilot Usability Study), η οποία βασίζεται σε καθοδηγούμενα user flows και σε δομημένο ερωτηματολόγιο καταγραφής χρόνου, αντιλαμβανόμενης δυσκολίας και ποιοτικών παρατηρήσεων.

Στο πεδίο της Αλληλεπίδρασης Ανθρώπου-Υπολογιστή (Human-Computer Interaction,

HCI), έχει τεκμηριωθεί εμπειρικά ότι η δοκιμή με πέντε (5) χρήστες αρκεί για να εντοπιστεί έως και το 85% των προβλημάτων χρηστικότητας ενός συστήματος [50]. Βάσει αυτής της αρχής, η παρούσα μελέτη χρησιμοποιεί πέντε συμμετέχοντες, καθιστώντας τη στατιστικά επαρκή για ένα πιλοτικό στάδιο αξιολόγησης.

Η συγκεκριμένη ενότητα αποσκοπεί στην αποτίμηση του βαθμού ευχρηστίας των βασικών λειτουργιών της πλατφόρμας Agropos, με έμφαση στις ροές που σχετίζονται με την αρχική χρήση του συστήματος, τη δημιουργία αγροτεμαχίου και την εγγραφή συσκευών και αισθητήρων. Με αυτόν τον τρόπο, η αξιολόγηση δεν περιορίζεται μόνο στην τεχνική ορθότητα της υλοποίησης, αλλά επεκτείνεται και στην εμπειρία χρήσης του συστήματος.

7.2.1 Στόχος και Σχεδιασμός Αξιολόγησης

Η αξιολόγηση χρηστικότητας σχεδιάστηκε ως δοκιμή εκτέλεσης συγκεκριμένων εργασιών από χρήστες, οι οποίοι ακολουθούν έναν σύντομο οδηγό βημάτων και στη συνέχεια συμπληρώνουν ερωτηματολόγιο. Η διαδικασία στοχεύει να απαντήσει στα εξής ερωτήματα:

- κατά πόσο οι βασικές λειτουργίες της πλατφόρμας είναι εύκολα εντοπίσιμες και κατανοητές,
- πόσο χρόνο απαιτεί η ολοκλήρωση κάθε βασικής ροής χρήσης,
- σε ποια σημεία οι χρήστες αντιμετωπίζουν δυσκολία ή σύγχυση,
- κατά πόσο λειτουργούν αξιόπιστα οι μηχανισμοί εγγραφής μέσω QR.

Η προσέγγιση αυτή είναι κατάλληλη για το Agropos, επειδή η πλατφόρμα απευθύνεται σε χρήστες που χρειάζεται να εκτελούν σαφείς, πρακτικές ενέργειες, όπως δημιουργία λογαριασμού, δημιουργία farm και συσχέτιση φυσικών συσκευών με το πληροφοριακό σύστημα.

Προφίλ Συμμετεχόντων.

Οι συμμετέχοντες στη μελέτη είναι πέντε (5) άτομα ηλικίας 25–45 ετών με πανεπιστημιακή εκπαίδευση. Το προφίλ αυτό επελέγη ως αντιπροσωπευτικό του τελικού χρήστη της πλατφόρμας, ο οποίος αναμένεται να διαθέτει βασική εξοικείωση με ψηφιακές υπηρεσίες και να ενδιαφέρεται για τη διαχείριση αγροτικών εκμεταλλεύσεων με τεχνολογικά μέσα.

7.2.2 Διαδικασία Διεξαγωγής

Σε κάθε δοκιμαστή παρέχεται αρχικά ο οδηγός ροών χρήσης, ο οποίος περιγράφει βήμα προς βήμα τις ενέργειες που πρέπει να εκτελεστούν στην πλατφόρμα. Ο δοκιμαστής ακολουθεί τις ροές με τη σειρά που δίνονται, χωρίς επιπλέον τεχνική καθοδήγηση πέρα από τις γραπτές οδηγίες, ώστε να προσεγγίζεται ένα ρεαλιστικό σενάριο πρώτης χρήσης.

Για κάθε εργασία καταγράφονται ο χρόνος ολοκλήρωσης και η υποκειμενική δυσκολία σε κλίμακα από 1 έως 5, όπου η τιμή 5 αντιστοιχεί σε πολύ εύκολη εκτέλεση και η τιμή 1 σε πολύ δύσκολη. Μετά την ολοκλήρωση όλων των εργασιών, ο χρήστης απαντά σε σύντομες ερωτήσεις ανοικτού και κλειστού τύπου σχετικά με την ευκολία εντοπισμού λειτουργιών, την αξιοπιστία της σάρωσης QR και τα σημεία στα οποία παρατηρήθηκε σύγχυση.

Ο Πίνακας 24 συνοψίζει τη μεθοδολογία της αξιολόγησης χρηστικότητας.

Πίνακας 24: Δομή της αξιολόγησης χρηστικότητας μέσω ερωτηματολογίου.

Στοιχείο	Περιγραφή
Μέθοδος	Δοκιμή εκτέλεσης εργασιών με συμπλήρωση ερωτηματολογίου
Υλικό δοκιμής	Οδηγός user flows και έντυπο καταγραφής απαντήσεων
Τύπος μετρήσεων	Ποσοτικές και ποιοτικές μετρήσεις
Ποσοτικές μεταβλητές	Χρόνος ολοκλήρωσης, βαθμός δυσκολίας
Ποιοτικές μεταβλητές	Σχόλια χρηστών, σημεία σύγχυσης, παρατηρήσεις για σφάλματα
Κλίμακα δυσκολίας	5 = πολύ εύκολο, 1 = πολύ δύσκολο

7.2.3 Εργασίες των Δοκιμαστών

Οι εργασίες που ζητούνται από τους συμμετέχοντες επιλέχθηκαν ώστε να καλύπτουν το βασικό μονοπάτι χρήσης της πλατφόρμας από την πρώτη είσοδο μέχρι την ολοκλήρωση της εγγραφής υποδομών πεδίου. Οι εργασίες αυτές παρουσιάζονται στον Πίνακα 25.

Πίνακας 25: Εργασίες που εκτελούν οι δοκιμαστές κατά την αξιολόγηση χρηστικότητας.

A/A	Εργασία
1	Δημιουργία νέου λογαριασμού χρήστη
2	Σύνδεση στην πλατφόρμα
3	Δημιουργία νέου farm και ορισμός περιοχής στον χάρτη
4	Καταχώριση συσκευής με χειροκίνητη εισαγωγή στοιχείων
5	Καταχώριση συσκευής μέσω σάρωσης QR
6	Καταχώριση αισθητήρα μέσω σάρωσης QR
7	Καταχώριση αισθητήρα με χειροκίνητη εισαγωγή στοιχείων

Μαζί με τον οδηγό βημάτων, σε κάθε δοκιμαστή δόθηκε έντυπο με τα στοιχεία της δοκιμαστικής συσκευής και του αισθητήρα, το οποίο περιλαμβάνει τα QR codes εγγραφής. Το έντυπο αυτό αναπαράγεται στο Σχήμα 37.



Σχήμα 37: Έντυπο QR codes συσκευής και αισθητήρα που δόθηκε στους δοκιμαστές.

Οι παραπάνω εργασίες εστιάζουν στα σημεία της εφαρμογής όπου ο χρήστης αλληλεπιδρά με περισσότερα πεδία εισόδου, με χάρτη ή με δεδομένα που προέρχονται από φυσικές ετικέτες QR. Κατά συνέπεια, είναι κατάλληλες για την ανάδειξη προβλημάτων πλοήγησης, ασάφειας οδηγιών ή σφαλμάτων στη διασύνδεση ανθρώπου-συστήματος.

7.2.4 Μετρικές και Κριτήρια Αξιολόγησης

Οι μετρήσεις που συλλέγονται από το ερωτηματολόγιο διακρίνονται σε ποσοτικές και ποιοτικές:

- **Χρόνος ολοκλήρωσης εργασίας:** χρησιμοποιείται ως ένδειξη του βαθμού ευχρηστίας και της ευκολίας πλοήγησης.
- **Αντιλαμβανόμενη δυσκολία:** αποτυπώνει τη subjective εμπειρία του χρήστη για κάθε ροή.
- **Επιτυχής ολοκλήρωση εργασίας:** δείχνει αν ο χρήστης κατάφερε να ολοκληρώσει τη ροή χωρίς εξωτερική παρέμβαση.
- **Σχόλια και ανοικτές παρατηρήσεις:** αναδεικνύουν ειδικά σημεία σύγχυσης, μη αναμενόμενα μηνύματα σφάλματος ή ελλείψεις στην ονοματοδοσία και στην οργάνωση του UI.

Ιδιαίτερη βαρύτητα δίνεται στις ροές εγγραφής συσκευής και αισθητήρα, επειδή αυτές συνδυάζουν πληροφορία από την εφαρμογή, το υλικό πεδίου και τους μηχανισμούς αναγνώρισης μέσω QR. Επομένως, τυχόν αστοχίες σε αυτές τις ροές επηρεάζουν άμεσα την αντιλαμβανόμενη ποιότητα και την πρακτική αξιοποίηση του συστήματος.

7.2.5 Παρουσίαση και Σύνοψη Αποτελεσμάτων

Μετά τη συλλογή των απαντήσεων, τα αποτελέσματα μπορούν να παρουσιαστούν συγκριτικά ανά εργασία, με χρήση μέσου χρόνου ολοκλήρωσης, μέσου βαθμού δυσκολίας και σύντομης σύνοψης των κυριότερων σχολίων. Ο Πίνακας 26 δίνει μία ενδεικτική μορφή παρουσίασης των αποτελεσμάτων.

Πίνακας 26: Σύνοψη ποσοτικών μετρήσεων ανά εργασία.

Εργασία	Μέσος χρόνος	Μέση δυσκολία	Ποσοστό επιτυχίας
Δημιουργία λογαριασμού	36"	5	100%
Σύνδεση	20"	5	100%
Δημιουργία farm	177"	3,3	100%
Προσθήκη συσκευής (χειροκίνητα)	70"	4,6	100%
Προσθήκη συσκευής (QR)	25"	4,6	100%
Προσθήκη αισθητήρα (QR)	27,6"	5	100%
Προσθήκη αισθητήρα (χειροκίνητα)	70"	2	30%

7.2.6 Αποτελέσματα και Ανάλυση Παρατηρήσεων

Μετά την ολοκλήρωση της πιλοτικής δοκιμής, τα δεδομένα από τα έντυπα καταγραφής και οι προφορικές παρατηρήσεις των χρηστών ομαδοποιήθηκαν. Παρόλο που το σύστημα κρίθηκε λειτουργικό, αναδείχθηκαν κρίσιμα σημεία «τριβής» (UX friction points) που αφορούν τη διασύνδεση και τη ροή πληροφορίας.

Ανάλυση ανά Εργασία και Προβλήματα Χρηστικότητας.

Οι κυριότερες παρατηρήσεις των χρηστών συνοψίζονται παρακάτω, ταξινομημένες σύμφωνα με τις αρχές ευχρηστίας:

- **Δημιουργία Farm (Task 3):** Παρατηρήθηκε βραχεία σύγχυση (1–2 δευτερόλεπτα) κατά τον τερματισμό του πολυγώνου στον χάρτη. Οι χρήστες δεν ήταν άμεσα σίγουροι για τη ενεργεια που απαιτείται για το κλείσιμο του σχήματος, αν και η λειτουργία εντοπίστηκε γρήγορα μέσω δοκιμής.
- **Προσθήκη Αισθητήρα μέσω QR (Task 5-6):**
 - Δοκιμαστής επιχείρησε να σαρώσει αισθητήρα χωρίς να έχει εγγράψει προηγουμένως τη συσκευή (node) στην οποία ανήκει. Το σύστημα απέρριψε την ενέργεια χωρίς να παρέχει επεξηγηματικό μήνυμα σφάλματος, προκαλώντας σύγχυση.
 - Σε περιπτώσεις απενεργοποιημένου GPS, οι συντεταγμένες δεν συμπληρώθηκαν αυτόματα και οι χρήστες αναγκάστηκαν να τις εισάγουν χειροκίνητα, διαδικασία που κρίθηκε επίπονη.
- **Χειροκίνητη Προσθήκη Αισθητήρα (Task 7):**
 - Χρήστης εισήγαγε κατά λάθος το UUID ενός ήδη υπάρχοντος αισθητήρα. Το σύστημα προχώρησε σε ενημέρωση (edit) της υπάρχουσας εγγραφής αντί για δημιουργία νέας, χωρίς όμως να ειδοποιήσει τον χρήστη για αυτή την αυτόματη αλλαγή κατάστασης.
 - Το συγκεκριμένο task είχε το χαμηλότερο ποσοστό επιτυχίας (30%), καθώς η πλειονότητα των χρηστών δεν γνώριζε ποιες τιμές πρέπει να εισαχθούν στα πεδία των γεωγραφικών συντεταγμένων.

Σύνοψη Ποιοτικών Ευρημάτων.

Ο Πίνακας 27 κωδικοποιεί τα προβλήματα που εντοπίστηκαν.

Πίνακας 27: Κατηγοριοποίηση προβλημάτων χρηστικότητας.

Πρόβλημα	Σύμπτωμα	Αρχή Ευχρηστίας
Απουσία Feedback	Μη εμφάνιση μηνύματος κατά την αποτυχία σύνδεσης αισθητήρα με μη εγγεγραμμένη συσκευή.	Visibility of System Status
Ασάφεια Δεδομένων	Δυσκολία κατανόησης και εισαγωγής γεωγραφικών συντεταγμένων με το χέρι.	Match between System and Real World
Αθέλητη Επεξεργασία	Αυτόματη ενημέρωση (edit) εγγραφής σε περίπτωση διπλότυπου ID χωρίς επιβεβαίωση.	Error Prevention

7.2.7 Προτάσεις Βελτίωσης

Με βάση τα παραπάνω ευρήματα, η μελλοντική εξέλιξη της πλατφόρμας Agronos θα πρέπει να εστιάσει στις εξής παρεμβάσεις:

1. **Βελτίωση Μηνυμάτων Σφάλματος:** Αντικατάσταση των γενικών μηνυμάτων με εξειδικευμένες οδηγίες (π.χ. «Παρακαλώ εγγράψτε πρώτα τη συσκευή πύλης πριν τη σάρωση του αισθητήρα»).
2. **Οπτική Επιλογή Σημείου:** Αντικατάσταση των πεδίων κειμένου για τις συντεταγμένες με δυνατότητα επιλογής σημείου απευθείας πάνω στον χάρτη (οπτικός επιλογέας τοποθεσίας).
3. **Στιγμιαία Μηνύματα Ενημέρωσης:** Εμφάνιση σύντομων αναδυόμενων μηνυμάτων (flash messages) μετά την ολοκλήρωση μιας ενέργειας, προκειμένου να ενημερώνεται ξεκάθαρα ο χρήστης για το ποια ενέργεια μόλις πραγματοποιήθηκε (π.χ. «Επιτυχής ενημέρωση υπάρχοντος αισθητήρα»).

8 Συμπέρασμα και Μελλοντικές Επεκτάσεις

8.1 Συμπεράσματα

Η παρούσα διπλωματική εργασία είχε ως κεντρικό στόχο τον σχεδιασμό και την υλοποίηση μιας ολοκληρωμένης, επεκτάσιμης και ασφαλούς πλατφόρμας IoT για την Έξυπνη Γεωργία, η οποία να γεφυρώνει το χάσμα μεταξύ της συλλογής δεδομένων πεδίου και της λήψης στρατηγικών αποφάσεων. Οι τέσσερις επιμέρους στόχοι — πολυπρωτοκολλική αρχιτεκτονική, ασφάλεια δεδομένων, αποδοτική διαχείριση χρονοσειρών και απλοποιημένη εγκατάσταση — επιτεύχθηκαν και αξιολογήθηκαν μέσα από πειραματικές μετρήσεις και πιλοτική μελέτη χρηστικότητας.

Αναφορικά με τη **πολυπρωτοκολλική αρχιτεκτονική**, αναπτύχθηκαν δύο ανεξάρτητοι τύποι κόμβων πεδίου: ο Wi-Fi κόμβος, κατάλληλος για αστικά περιβάλλοντα με υπάρχουσα υποδομή δικτύου, και ο LoRa κόμβος, σχεδιασμένος για τη γεφύρωση των μεγάλων αποστάσεων που χαρακτηρίζουν τις αγροτικές εκτάσεις. Και οι δύο τύποι ενσωματώνονται απρόσκοπτα στο ίδιο backend, αποδεικνύοντας την αρχιτεκτονική ευελιξία της πλατφόρμας.

Αναφορικά με την **ασφάλεια δεδομένων**, εφαρμόστηκε υβριδικό μοντέλο ανά πρωτόκολλο: κρυπτογράφηση TLS/HTTPS για τους Wi-Fi κόμβους και κρυπτογράφηση AES-128-CTR από άκρο σε άκρο για τους LoRa κόμβους, εξασφαλίζοντας ότι τα δεδομένα παραμένουν αδιάβαστα για οποιονδήποτε ενδιάμεσο κόμβο δικτύου.

Αναφορικά με τη **διαχείριση χρονοσειρών**, η ενσωμάτωση της InfluxDB επέτρεψε την αποδοτική αποθήκευση και οπτικοποίηση ιστορικών δεδομένων μέσω δυναμικού dashboard, μετατρέποντας τις ακατέργαστες μετρήσεις σε κατανοητή πληροφορία για τον παραγωγό.

Μέσα από την πειραματική αξιολόγηση, αναδείχθηκε ότι η ενεργειακή βιωσιμότητα των κόμβων πεδίου εξαρτάται άμεσα από τη διαχείριση των περιφερειακών αισθητήρων. Συγκεκριμένα, οι δοκιμές ανέδειξαν τον αισθητήρα υγρασίας εδάφους ως το πλέον ενεργοβόρο στοιχείο, μειώνοντας την αυτονομία κατά περίπου 35%. Αντίθετα, η επικοινωνία μέσω πρωτοκόλλου LoRa επέδειξε εντυπωσιακή ενεργειακή σταθερότητα, με μόλις 3,3% αποφόρτιση σε διάστημα πέντε ημερών, καθιστώντας την ως τη βέλτιστη λύση για απομακρυσμένες καλλιέργειες χωρίς υφιστάμενες υποδομές δικτύου.

Αναφορικά με την **απλοποίηση της εγκατάστασης**, η πιλοτική μελέτη χρηστικότητας κατέδειξε ότι η επιτυχία των συστημάτων Ag-Tech κρίνεται στον βαθμό απλοποίησης των διαδικασιών παραμετροποίησης (provisioning). Οι μηχανισμοί σάρωσης QR μετατρέπουν τη σύνθετη εγγραφή υλικού σε μια απλή αλληλουχία κινήσεων, μειώνοντας δραστικά τον τεχνικό φραγμό εισόδου για τον μη εξειδικευμένο χρήστη. Ο πρωτεύων τρόπος εισαγωγής, μέσω σάρωσης QR και αυτόματης ανάκτησης γεωγραφικών συντεταγμένων από το GPS του κινητού, κρίθηκε εύχρηστος από τους συμμετέχοντες. Ωστόσο, ο εναλλακτικός τρόπος χειροκίνητης εισαγωγής συντεταγμένων δημιούργησε σύγχυση, αναδεικνύοντας την ανάγκη αντικατάστασής του με ένα οπτικό εργαλείο επιλογής τοποθεσίας στον χάρτη.

Συμπερασματικά, το σύστημα Agropos τεκμηριώνει ότι ο συνδυασμός ασφαλούς αρχιτεκτονικής και ενεργειακά αποδοτικής επικοινωνίας μπορεί να αποτελεί ένα βιώσιμο θεμέλιο για τη μετάβαση στη γεωργία ακριβείας, υπό την προϋπόθεση περαιτέρω αξιολόγησης και προσαρμογής σε ποικίλα περιβάλλοντα.

8.2 Επεκτάσεις

8.2.1 Υλοποίηση Δικτύου Πλέγματος (LoRa Mesh) για Γεωγραφικά Διεσπαρμένες Ιδιοκτησίες

Η μορφολογία και η δομή της αγροτικής ιδιοκτησίας στην ελληνική ύπαιθρο χαρακτηρίζεται κατά κύριο λόγο από μικρό κλήρο και έντονη γεωγραφική διασπορά. Ένας αγρότης συχνά διαχειρίζεται πολλαπλά, μη συνορεύοντα αγροτεμάχια. Η ανάπτυξη του προσαρμοσμένου LoRa Gateway της πλατφόρμας Agropos σχεδιάστηκε ακριβώς με γνώμονα τη δραστική μείωση του κόστους εγκατάστασης έναντι των ακριβών εμπορικών LoRaWAN concentrators, προσφέροντας μια οικονομικά βιώσιμη λύση για τη συλλογή δεδομένων.

Παρόλα αυτά, ένας θεμελιώδης περιορισμός της υφιστάμενης αρχιτεκτονικής είναι η απαίτηση για ενεργή σύνδεση στο Διαδίκτυο (backhaul network) στο σημείο τοποθέτησης του κάθε Gateway. Σε περιπτώσεις όπου τα απομακρυσμένα αγροτεμάχια βρίσκονται εκτός της εμβέλειας ενός κεντρικού LoRa Gateway (π.χ. εκτός της ακτίνας των 2–5 χιλιομέτρων), η κάλυψή τους θα απαιτούσε την εγκατάσταση επιπλέον κόμβων πύλης, οι οποίοι θα έπρεπε υποχρεωτικά να εξοπλιστούν με modem κυψελοειδούς τηλεφωνίας (βλ. Ενότητα 2.2.13) για την προώθηση των δεδομένων στο cloud. Η προσέγγιση αυτή εισάγει σημαντικό λειτουργικό κόστος, λόγω των παγίων συνδρομών δικτύου για κάθε ξεχωριστό

απομακρυσμένο αγροτεμάχιο, και αναιρεί εν μέρει το πλεονέκτημα της χαμηλού κόστους αρχιτεκτονικής.

Μια πιθανή μελλοντική κατεύθυνση είναι η επέκταση του πρωτοκόλλου επικοινωνίας ώστε να υποστηρίζει τοπολογία Δικτύου Πλέγματος (Mesh Network) σε επίπεδο πυλών (Gateway-to-Gateway routing), μια επιλογή που φέρει ορισμένα ανοιχτά ζητήματα (π.χ. διαχείριση δρομολόγησης, κόστη υλοποίησης, εγγυήσεις ασφάλειας). Αντί κάθε Gateway να απαιτεί άμεση σύνδεση στο Διαδίκτυο, τα ενδιάμεσα Gateways θα μπορούν να λειτουργούν ως κόμβοι αναμετάδοσης (relay nodes).

Σε αυτή την αρχιτεκτονική, αξιοποιώντας την τεχνολογία LoRa Mesh (βλ. Ενότητα 2.2.3), τα πακέτα δεδομένων θα δρομολογούνται πολυ-αλματικά (multi-hop) από το ένα Gateway στο άλλο, μέχρι να καταλήξουν στον τελικό κόμβο εξόδου (Border Router / Sink Node), ο οποίος θα είναι ο μοναδικός που θα διαθέτει σύνδεση στο Διαδίκτυο (π.χ. στο σπίτι του παραγωγού ή μέσω μιας μοναδικής σύνδεσης 4G/5G).

Η υλοποίηση αυτής της προσέγγισης θα προσφέρει τα εξής στρατηγικά πλεονεκτήματα:

- **Εκμνησίση Λειτουργικού Κόστους (OPEX):** Απαλείφεται η ανάγκη για πολλές κάρτες SIM και συμβόλαια κινητής τηλεφωνίας.
- **Αυξημένη Εμβέλεια και Ανθεκτικότητα (Resilience):** Η αυτο-οργάνωση του πλέγματος επιτρέπει την παράκαμψη φυσικών εμποδίων μέσω εναλλακτικών μονοπατιών δρομολόγησης.
- **Προστασία Δεδομένων :** Καθώς η κρυπτογράφηση AES-128-CTR εφαρμόζεται σε επίπεδο ακραίου κόμβου (end-node), τα ενδιάμεσα Gateways του mesh δικτύου αναμεταδίδουν αποκλειστικά τα αδιαφανή (κρυπτογραφημένα) payloads, εγγυώμενα την ιδιωτικότητα και την ακεραιότητα των δεδομένων.

8.2.2 Πλήρης Ενσωμάτωση Πρωτοκόλλου LoRaWAN και Χρήση Πολυκαναλικών Πυλών (LoRa Concentrators)

Η τρέχουσα υλοποίηση του LoRa Gateway της πλατφόρμας Agronos βασίζεται σε έναν μονοκαναλικό πομποδέκτη (single-channel transceiver, π.χ. SX1276), ο οποίος ακούει σε μία συγκεκριμένη συχνότητα και με έναν προκαθορισμένο Παράγοντα Διασποράς (Spreading Factor – SF) τη φορά. Αν και αυτή η προσέγγιση είναι εξαιρετικά αποδοτική από πλευράς

κόστους για μικρής και μεσαίας κλίμακας εγκαταστάσεις, παρουσιάζει εγγενείς περιορισμούς χωρητικότητας. Καθώς ο αριθμός των αισθητήρων (node density) εντός της ίδιας περιοχής κάλυψης αυξάνεται, το απλό μοντέλο πρόσβασης στο μέσο (pure ALOHA) που χρησιμοποιείται, οδηγεί αναπόφευκτα σε αυξημένο ρυθμό συγκρούσεων πακέτων (packet collisions) και σε κορεσμό του καναλιού.

Για περιπτώσεις μεγάλης κλίμακας (π.χ. εκτεταμένες συνεταιριστικές καλλιέργειες, μεγάλα θερμοκήπια ή έξυπνα χωριά), καθίσταται επιβεβλημένη η μετάβαση στο βιομηχανικό πρότυπο LoRaWAN (βλ. Ενότητα 2.2.2) και η αντικατάσταση του απλού Gateway από συσκευές εξοπλισμένες με LoRa Concentrator (π.χ. βασισμένες σε chipsets SX1302/SX1303).

Μια τέτοια αναβάθμιση θα προσφέρει θεμελιώδη αρχιτεκτονικά και λειτουργικά πλεονεκτήματα στην πλατφόρμα Agronos:

- **Ταυτόχρονη Πολυκαναλική Λήψη:** Ένας LoRa concentrator έχει την ικανότητα να αποδιαμορφώνει ταυτόχρονα πακέτα σε πολλαπλά κανάλια (συνήθως 8 ή περισσότερα) και σε πολλαπλούς διαφορετικούς Παράγοντες Διασποράς (SF). Αυτό πολλαπλασιάζει δραματικά τη χωρητικότητα του δικτύου, επιτρέποντας την απρόσκοπτη συνύπαρξη χιλιάδων αισθητήρων κάτω από την ίδια πύλη.
- **Προσαρμοστικός Ρυθμός Δεδομένων (Adaptive Data Rate – ADR):** Η πλήρης ενσωμάτωση του LoRaWAN MAC layer θα επιτρέψει στο δίκτυο να ελέγχει δυναμικά την ισχύ εκπομπής (TX power) και τον Παράγοντα Διασποράς (SF) κάθε κόμβου ξεχωριστά. Οι κόμβοι που βρίσκονται πιο κοντά στην πύλη θα εκπέμπουν με μικρότερο SF, εξοικονομώντας μπαταρία και μειώνοντας το χρόνο δέσμευσης του ραδιοφωνικού φάσματος (Time on Air).
- **Διαλειτουργικότητα (Interoperability):** Η υιοθέτηση του LoRaWAN θα αποδεσμεύσει την πλατφόρμα Agronos από το custom firmware των κόμβων. Το σύστημα θα μπορεί να δέχεται (out-of-the-box) οποιονδήποτε εμπορικό, πιστοποιημένο LoRaWAN αισθητήρα της αγοράς (COTS – Commercial Off-The-Shelf), διευρύνοντας τις εμπορικές προοπτικές του έργου.

8.2.3 Ενσωμάτωση Ενεργοποιητών (Actuators) και Συστημάτων Αυτοματισμού

Στα πλαίσια της παρούσας διπλωματικής εργασίας, η ανάπτυξη της πλατφόρμας Agronos επικεντρώθηκε αποκλειστικά στη μονόδρομη συλλογή, ασφαλή μετάδοση, αποθήκευση

και οπτικοποίηση τηλεμετρικών δεδομένων. Ως εκ τούτου, δεν υλοποιήθηκε το υποσύστημα των ενεργοποιητών (actuators). Ωστόσο, για να αναδειχθούν οι πραγματικές δυνατότητες και η προστιθέμενη αξία ενός ολοκληρωμένου συστήματος Έξυπνης Γεωργίας (Smart Agriculture), η μετάβαση από την απλή παρακολούθηση (monitoring) στον ενεργό έλεγχο (active control) κρίνεται επιβεβλημένη.

Η προσθήκη ενεργοποιητών στο επίπεδο του πεδίου —όπως για παράδειγμα ηλεκτροβάνες για τον έλεγχο του δικτύου άρδευσης, διακόπτες ρελέ (relays) για αντλίες νερού ή συστήματα ελέγχου κλίματος σε θερμοκήπια— θα επιτρέψει τη δημιουργία ενός συστήματος κλειστού βρόχου (closed-loop system). Η λειτουργία των ενεργοποιητών προτείνεται να αναπτυχθεί σε δύο διακριτά επίπεδα αλληλεπίδρασης:

Απομακρυσμένος Χειροκίνητος Έλεγχος (Remote Manual Control): Ο χρήστης, μέσω του διαδικτυακού περιβάλλοντος (dashboard) της πλατφόρμας, θα έχει τη δυνατότητα να ενεργοποιεί και να απενεργοποιεί χειροκίνητα τους μηχανισμούς στον αγρό του (π.χ. άνοιγμα/κλείσιμο παροχής νερού), παρακάμπτοντας την ανάγκη φυσικής παρουσίας.

Αυτόνομη Λειτουργία και Κανόνες Αυτοματισμού (Rule-based Automation): Ο πυρήνας της αναβάθμισης έγκειται στην υλοποίηση μιας Μηχανής Κανόνων (Rule Engine) στο backend της εφαρμογής. Ο παραγωγός θα μπορεί να ορίζει λογικές συνθήκες τύπου “if-this-then-that”. Για παράδειγμα, το σύστημα θα μπορεί να αξιολογεί συνεχώς τις τιμές του αισθητήρα υγρασίας εδάφους (Soil Moisture) και, εάν το ποσοστό πέσει κάτω από ένα κρίσιμο όριο (π.χ. $< 30\%$), να δίνει αυτόματα εντολή έναρξης του ποτίσματος, διακόπτοντάς το μόλις επιτευχθεί το βέλτιστο επίπεδο.

Αρχιτεκτονική Εφαρμογή: Τεχνικά, η ενοποίηση των ενεργοποιητών μπορεί να αξιοποιήσει την ήδη υπάρχουσα υποδομή του MQTT Broker (EMQX) (βλ. Ενότητα 2.2.11). Σε αντίθεση με τη λειτουργία συλλογής δεδομένων, όπου οι συσκευές εκπέμπουν (publish), στην περίπτωση των ενεργοποιητών οι IoT κόμβοι θα πρέπει να εγγράφονται (subscribe) σε αποκλειστικά command topics (π.χ. `devices/{uuid}/commands`). Το Laravel backend, μέσω του ενσωματωμένου MQTT client, θα δημοσιεύει τα μηνύματα εντολών σε πραγματικό χρόνο προς τις συσκευές, διασφαλίζοντας χαμηλή καθυστέρηση (low latency) και επιβεβαίωση λήψης (μέσω QoS levels) κατά την εκτέλεση κρίσιμων γεωργικών εργασιών.

8.2.4 Ενσωμάτωση Τεχνολογίας NB-IoT για Απομακρυσμένους Ενεργοποιητές (Actuators)

Σε συνέχεια της προτεινόμενης υλοποίησης συστημάτων αυτοματισμού (ενεργοποιητών), προκύπτει μια σημαντική αρχιτεκτονική πρόκληση: τι συμβαίνει όταν ο εξοπλισμός ελέγ-

χου (π.χ. μια κεντρική ηλεκτροβάννα δικτύου άρδευσης ή το απομακρυσμένο αντλιοστάσιο μιας γεώτρησης) βρίσκεται σε πλήρη γεωγραφική απομόνωση από το υπόλοιπο δίκτυο αισθητήρων του αγροτεμαχίου;

Σε τέτοιες περιπτώσεις, η ανάπτυξη μιας αποκλειστικής πύλης LoRa (LoRa Gateway) μόνο για την εξυπηρέτηση ενός και μόνο ενεργοποιητή καθίσταται οικονομικά μη αποδοτική. Επιπλέον, το πρωτόκολλο LoRa (βλ. Ενότητα 2.2.1), ενώ αριστεύει στην ασύγχρονη και χαμηλής κατανάλωσης αποστολή δεδομένων προς το διαδίκτυο (uplink), παρουσιάζει εγγενείς περιορισμούς στην αμφίδρομη επικοινωνία πραγματικού χρόνου. Η λήψη εντολών ελέγχου (downlink) υφίσταται σημαντικό λανθάνοντα χρόνο (latency) για να εξοικονομηθεί μπαταρία, εκτός εάν οι κόμβοι ρυθμιστούν σε συνεχή λειτουργία λήψης (LoRaWAN Class C), γεγονός που αυξάνει δραματικά τις ενεργειακές απαιτήσεις.

Ως βέλτιστη μελλοντική επέκταση για αυτές τις ειδικές περιπτώσεις, προτείνεται η εγγενής υποστήριξη συσκευών που αξιοποιούν το πρωτόκολλο Narrowband IoT (NB-IoT) (βλ. Ενότητα 2.2.6). Το NB-IoT είναι ένα εξειδικευμένο Κυψελοειδές Δίκτυο Ευρείας Περιοχής Χαμηλής Ισχύος (Cellular LPWAN), το οποίο αξιοποιεί την υφιστάμενη εθνική υποδομή της κινητής τηλεφωνίας (4G/5G).

Η στρατηγική ενσωμάτωση του NB-IoT προσφέρει τρία κρίσιμα πλεονεκτήματα για τον απομακρυσμένο έλεγχο μηχανισμών:

- **Άμεση Συνδεσιμότητα (Direct-to-Cloud):** Ο κόμβος NB-IoT επικοινωνεί απευθείας με τον σταθμό βάσης του παρόχου δικτύου κινητής τηλεφωνίας, εξαλείφοντας πλήρως την ανάγκη για ενδιάμεσο τοπικό Gateway και μειώνοντας τα σημεία πιθανής βλάβης (points of failure).
- **Υψηλή Ποιότητα Υπηρεσίας (QoS) σε Αδειοδοτημένο Φάσμα:** Αντίθετα με το LoRa που λειτουργεί σε ελεύθερες συχνότητες (ISM bands), το NB-IoT λειτουργεί σε αδειοδοτημένο φάσμα. Αυτό σημαίνει ότι δεν υπόκειται σε αυστηρούς περιορισμούς χρόνου εκπομπής (duty cycle limits) και δεν επηρεάζεται από παρεμβολές γειτονικών δικτύων. Συνεπώς, εξασφαλίζει ότι οι κρίσιμες εντολές ενεργοποίησης/απενεργοποίησης θα παραδίδονται στον μηχανισμό άμεσα και με απόλυτη αξιοπιστία, κάτι κρίσιμο για υποδομές όπως οι αντλίες νερού.
- **Βαθιά Διεσδυτικότητα (Deep Coverage):** Το σήμα του NB-IoT έχει σχεδιαστεί για να παρέχει εξαιρετική ικανότητα διείσδυσης (+20 dB link budget σε σχέση με το

GPRS), καθιστώντας το ιδανικό για ενεργοποιητές που είναι θαμμένοι σε υπόγεια φρεάτια άρδευσης ή εντός μεταλλικών αντλιοστασίων, όπου το κοινό σήμα υποβαθμίζεται.

8.2.5 Αντικατάσταση Σύγχρονου Webhook με Ασύγχρονη Αρχιτεκτονική Go MQTT Client / Redis Queue / Laravel Queue Workers

Περιγραφή του Υφιστάμενου Περιορισμού

Στην τρέχουσα υλοποίηση, η διαδρομή εισόδου των τηλεμετρικών δεδομένων ακολουθεί ένα σύγχρονο, σφιχτά συζευγμένο (tightly coupled) μοντέλο: ο EMQX Rule Engine προωθεί κάθε εισερχόμενο μήνυμα MQTT ως αίτημα HTTP POST (webhook) στο αντίστοιχο endpoint του Laravel backend (βλ. § 5.4.3). Η προσέγγιση αυτή είναι επαρκής για μικρό αριθμό αισθητήρων, παρουσιάζει όμως ένα δομικό σημείο κορεσμού (bottleneck) καθώς η κλίμακα του δικτύου αυξάνεται: η επεξεργαστική ικανότητα του συστήματος ορίζεται αποκλειστικά από τον αριθμό των PHP-FPM workers του web server που μπορούν να εξυπηρετήσουν ταυτόχρονα HTTP αιτήματα. Εφόσον ο ρυθμός άφιξης MQTT μηνυμάτων υπερβεί τη διαθέσιμη δεξαμενή workers, ο EMQX θα αρχίσει να δέχεται αρνητικές αποκρίσεις (HTTP 5xx), με αποτέλεσμα την απώλεια δεδομένων.

Η Προτεινόμενη Αρχιτεκτονική

Ως σημαντική μελλοντική αναβάθμιση της υποδομής επεξεργασίας δεδομένων, προτείνεται η αντικατάσταση του σύγχρονου webhook μοντέλου με μια τριεπίπεδη ασύγχρονη αρχιτεκτονική (asynchronous pipeline), αποτελούμενη από τα ακόλουθα δομικά στοιχεία:

1. **Go MQTT Client (Επίπεδο Κατανάλωσης MQTT):** Ένας ελαφρύς (lightweight), υψηλής απόδοσης daemon γραμμένος στη γλώσσα Go αναλαμβάνει τον ρόλο του MQTT subscriber, παρακολουθώντας ενεργά τα κανάλια τηλεμετρίας των συσκευών (π.χ. devices/+/telemetry). Η γλώσσα Go επιλέγεται σκοπίμως για τον ρόλο αυτό: το μοντέλο ταυτόχρονης εκτέλεσης με goroutines επιτρέπει στον daemon να διαχειρίζεται χιλιάδες ταυτόχρονες ροές εισερχόμενων μηνυμάτων με αμελητέα κατανάλωση μνήμης και χωρίς την εισαγωγή HTTP επικεφαλίδας (overhead). Σε αντίθεση με το webhook μοντέλο, ο Go daemon δεν αναμένει επιβεβαίωση επεξεργασίας από το backend· αφού ελέγξει τη βαθμωτή έγκυρότητα (sanity check) του ωφέλιμου φορτίου (payload), το ωθεί (push) άμεσα στην ουρά του Redis.

2. **Redis Queue (Επίπεδο Αποθήκευσης Ενδιάμεσης Κατάστασης):** Η Redis LIST δομή δεδομένων λειτουργεί ως ελαστική, ανθεκτική στις αιχμές φόρτου (load-spike resilient) ενδιάμεση ουρά (buffer). Κάθε εισερχόμενο μήνυμα αισθητήρα σειριοποιείται σε μορφή JSON και εισάγεται στην ουρά στη μορφή που απαιτεί το σύστημα ουρών του Laravel. Η Redis εξασφαλίζει ότι τα δεδομένα δεν χάνονται ακόμη και σε περίπτωση προσωρινής αδιαθεσίας του backend, καθώς τα μηνύματα παραμένουν αποθηκευμένα στη δομή LIST έως ότου καταναλωθούν.
3. **Laravel Queue Workers (Επίπεδο Επεξεργασίας):** Ένας ή περισσότεροι PHP worker processes εκτελούν ασύγχρονα την ίδια επεξεργαστική λογική που σήμερα εκτελείται σύγχρονα εντός του HTTP request lifecycle: αποκρυπτογράφηση, επικύρωση, αποθήκευση στην InfluxDB και ενημέρωση cache. Ο αριθμός των workers μπορεί να κλιμακωθεί οριζόντια (horizontal scaling) ανεξάρτητα από τον web server, ακόμη και σε διαφορετικά virtual machines ή containers.

Πλεονεκτήματα της Αρχιτεκτονικής

Αποσύζευξη Ρυθμού Εισόδου από Ρυθμό Επεξεργασίας : Αιχμές στον ρυθμό άφιξης μηνυμάτων (π.χ. ταυτόχρονη αναφορά εκατοντάδων κόμβων μετά από διακοπή δικτύου) απορροφώνται από την Redis ουρά, χωρίς να κατακλύζεται ο PHP application server.

Χωρίς Data Loss σε Αστοχία Backend: Εφόσον το Laravel είναι προσωρινά μη διαθέσιμο (π.χ. ενημέρωση λογισμικού), ο Go daemon συνεχίζει να συλλέγει μηνύματα και να τα εισάγει στην Redis ουρά· μόλις οι workers επανεκκινήσουν, η επεξεργασία επαναλαμβάνεται αυτόματα.

8.2.6 Δυναμική Βαθμονόμηση Αισθητήρων μέσω Captive Portal

Στην παρούσα υλοποίηση του υλικολογισμικού των κόμβων IoT, η βαθμονόμηση των αναλογικών αισθητήρων, και συγκεκριμένα του χωρητικού αισθητήρα υγρασίας εδάφους, βασίζεται σε σταθερές τιμές ορισμένες κατά τον χρόνο μεταγλώττισης. Συγκεκριμένα, οι ακραίες τιμές του μετατροπέα αναλογικού σήματος σε ψηφιακό (ADC) για την κατάσταση πλήρους ξηρασίας (αέρας) και πλήρους υγρασίας (νερό) είναι προκαθορισμένες στο αρχείο ρυθμίσεων της συσκευής.

Αν και αυτή η προσέγγιση είναι λειτουργική σε εργαστηριακό επίπεδο, παρουσιάζει σημαντικούς περιορισμούς σε πραγματικές συνθήκες. Οι αναλογικοί αισθητήρες παρουσιάζουν εγγενείς αποκλίσεις υλικού από παρτίδα σε παρτίδα. Υπό την τρέχουσα αρχι-

τεκτονική, η επαναβαθμονόμηση ενός αισθητήρα απαιτεί επέμβαση στον πηγαίο κώδικα και εκ νέου προγραμματισμό του μικροελεγκτή, μια διαδικασία μη ρεαλιστική για τον μέσο παραγωγό-χρήστη.

Μια πιθανή μελλοντική βελτίωση είναι η υλοποίηση ενός μηχανισμού **Δυναμικής Βαθμονόμησης (Runtime Dynamic Calibration)**, αξιοποιώντας την ήδη υπάρχουσα υποδομή του ενσωματωμένου διακομιστή ιστού (Captive Portal) της συσκευής. Ωστόσο, παραμένουν ανοιχτά ζητήματα σχετικά με την ασφάλεια της διεπαφής βαθμονόμησης και την αξιοπιστία της διαδικασίας σε ποικίλες περιβαλλοντικές συνθήκες. Η αρχιτεκτονική της προτεινόμενης λύσης δομείται στα εξής βήματα:

Προτείνεται η προσθήκη διαδραστικής διεπαφής βαθμονόμησης στο Captive Portal. Η διαδικασία θα είναι δύο σημείων: ο χρήστης θα επιλέγει «Καθορισμός Ξηρού (Set Dry)» με τον αισθητήρα καθαρό στον αέρα και «Καθορισμός Υγρού (Set Wet)» με τον αισθητήρα βυθισμένο στο νερό, αποθηκεύοντας τις αντίστοιχες τιμές αναφοράς. Οι νέες τιμές θα αποθηκεύονται στη μη-πτητική μνήμη (NVS) του ESP32. Κατά την εκκίνηση, η κλάση του αισθητήρα θα ελέγχει την NVS και, εάν υπάρχουν αποθηκευμένες τιμές βαθμονόμησης, θα τις χρησιμοποιεί· σε αντίθετη περίπτωση θα επιστρέφει στις προεπιλεγμένες εργοστασιακές σταθερές.

Η υλοποίηση αυτής της αρχιτεκτονικής προσδίδει σημαντική προστιθέμενη αξία, καθώς εξαλείφει την ανάγκη για τεχνική υποστήριξη πεδίου (field maintenance), διασφαλίζει τη μακροχρόνια ακρίβεια των μετρήσεων και προσφέρει μια επαγγελματικού επιπέδου «Plug-and-Play» εμπειρία στον τελικό χρήστη.

8.2.7 Ενοποιημένο Σύστημα Παραμετροποίησης μέσω Captive Portal και για τους Κόμβους LoRa

Μία ουσιαστική κατεύθυνση για την περαιτέρω εξέλιξη της πλατφόρμας αποτελεί η ενσωμάτωση της λειτουργικότητας του Captive Portal και στους κόμβους επικοινωνίας LoRa, δημιουργώντας έτσι μια ομοιόμορφη εμπειρία διαχείρισης για το σύνολο του εξοπλισμού. Παρόλο που οι συσκευές LoRa είναι σχεδιασμένες με γνώμονα τη χαμηλή ενεργειακή κατανάλωση, η χρήση μικροελεγκτών της οικογένειας ESP32 προσφέρει τη δυνατότητα αξιοποίησης του ενσωματωμένου υποσυστήματος WiFi για περιορισμένο χρονικό διάστημα. Η υλοποίηση αυτή θα επιτρέπει στη συσκευή, κατά την αρχική της ενεργοποίηση ή μετά από εντολή του χρήστη, να λειτουργεί προσωρινά ως σημείο πρόσβασης, παρέχοντας μια διεπαφή παραμετροποίησης παρόμοια με αυτή που αναπτύχθηκε για το LoRa

Gateway.

Μέσω αυτής της δυναμικής προσέγγισης, ο παραγωγός θα αποκτά την ικανότητα να ορίζει κρίσιμες παραμέτρους της ραδιοεπικοινωνίας απευθείας στο πεδίο, χωρίς την ανάγκη σύνδεσης της συσκευής σε ηλεκτρονικό υπολογιστή ή την επαναμεταγλώττιση του κώδικα. Συγκεκριμένα, η διεπαφή θα προσφέρει τη δυνατότητα ρύθμισης των καναλιών συχνοτήτων, του Παράγοντα Διασποράς (Spreading Factor) και του Coding Rate, επιτρέποντας στον χρήστη να προσαρμόζει τη λειτουργία του κόμβου ανάλογα με τις τοπικές συνθήκες κάλυψης ή τις απαιτήσεις του δικτύου.

Συνολικά, η πλατφόρμα Agropos αποτελεί ένα λειτουργικό και τεκμηριωμένο βήμα προς τη γεωργία ακριβείας προσιτής κλίμακας. Οι μελλοντικές επεκτάσεις που περιγράφηκαν στην παρούσα ενότητα δεν αποτελούν αδυναμίες του συστήματος, αλλά φυσικές κατευθύνσεις εξέλιξης που θεμελιώνονται στη σταθερή αρχιτεκτονική βάση που αναπτύχθηκε. Η εργασία αποδεικνύει ότι η υλοποίηση ολοκληρωμένων Ag-Tech λύσεων δεν απαιτεί εμπορικές υποδομές υψηλού κόστους, αλλά μια προσεκτικά σχεδιασμένη αρχιτεκτονική που θέτει στο επίκεντρο την ασφάλεια, την ενεργειακή αποδοτικότητα και τη χρηστικότητα.

Α' Παράρτημα Α: Κώδικας Backend

Το παράρτημα αυτό περιλαμβάνει τα σημαντικότερα τμήματα κώδικα του backend της πλατφόρμας *Agropos*. Οι κλάσεις που παρουσιάζονται εδώ είναι οι κεντρικές υπηρεσίες και οι controllers που συνδέουν το REST API, τη ροή δεδομένων και τη χρονική αποθήκευση.

Α'.1 Eloquent Models και Trait πολυμίσθωσης

```
1 <?php
2
3 namespace App\Models;
4
5 use App\Traits\BelongsToTenant;
6 use Illuminate\Database\Eloquent\Factories\HasFactory;
7 use Illuminate\Database\Eloquent\Model;
8 use Illuminate\Database\Eloquent\Relations\BelongsTo;
9
10 class Farm extends Model
11 {
12     use BelongsToTenant, HasFactory;
13
14     /**
15      * The attributes that are mass assignable.
16      *
17      * @var array<int, string>
18      */
19     protected $fillable = [
20         'user_id',
21         'name',
22         'location',
23         'size',
24         'coordinates',
25         'description',
```



```
26     ];
27
28     protected $casts = [
29         'coordinates' => 'array',
30     ];
31
32     protected $appends = ['center'];
33
34     /**
35      * Get the user that owns the farm.
36      */
37     public function user(): BelongsTo
38     {
39         return $this->belongsTo(User::class);
40     }
41
42     public function getAreaAttribute()
43     {
44         if (! $this->coordinates || ! isset($this->coordinates['
45             coordinates'][0])) {
46             return 0;
47         }
48
49         // Calculate area using turf.js port or geometry library
50         return $this->calculatePolygonArea($this->coordinates['
51             coordinates'][0]);
52     }
53
54     public function getCenterAttribute()
55     {
56         if (! $this->coordinates || ! isset($this->coordinates['
57             coordinates'][0])) {
58             return null;
59         }
60     }
```

```
57     $coords = $this->coordinates['coordinates'][0];
58     $count = count($coords);
59     if ($count === 0) {
60         return null;
61     }
62     $sum = array_reduce($coords, function ($acc, $p) {
63         return [$acc[0] + $p[0], $acc[1] + $p[1]];
64     }, [0, 0]);
65
66     return [
67         'lng' => $sum[0] / $count,
68         'lat' => $sum[1] / $count,
69     ];
70 }
71
72 public function sensors()
73 {
74     return $this->hasMany(Sensor::class);
75 }
76 }
```

Listing 69: Πλήρης ορισμός του μοντέλου Farm

```
1 <?php
2
3 namespace App\Models;
4
5 use App\Enums\DeviceStatus;
6 use App\Enums\DeviceType;
7 use App\Traits\BelongsToTenant;
8 use Illuminate\Database\Eloquent\Factories\HasFactory;
9 use Illuminate\Foundation\Auth\User as Authenticatable;
10 use Laravel\Sanctum\HasApiTokens;
11
12 class Device extends Authenticatable
```

```
13 {  
14     use BelongsToTenant, HasApiTokens, HasFactory;  
15  
16     protected $fillable = [  
17         'user_id',  
18         'name',  
19         'uuid',  
20         'secret',  
21         'type',  
22         'status',  
23         'last_seen_at',  
24         'battery_level',  
25         'signal_strength',  
26         'mqtt_username',  
27         'mqtt_password',  
28         'lora_frame_counter',  
29         'lora_aes_key',  
30     ];  
31  
32     protected $hidden = [  
33         'secret',  
34         'lora_aes_key',  
35     ];  
36  
37     protected $casts = [  
38         'status' => DeviceStatus::class,  
39     ];  
40  
41     public function user()  
42     {  
43         return $this->belongsTo(User::class);  
44     }  
45  
46     public function sensors()
```

```
47     {
48         // Return a simplified array of sensors for frontend
49         // consumption
50         return $this->hasMany(Sensor::class);
51     }
52
53     public function isLoRa(): bool
54     {
55         return $this->type === DeviceType::LORA->value;
56     }
57 }
```

Listing 70: Πλήρης ορισμός του μοντέλου Device

```
1 <?php
2
3 namespace App\Models;
4
5 use App\Enums\DeviceStatus;
6 use App\Enums\SensorType;
7 use App\Traits\BelongsToTenant;
8 use Illuminate\Database\Eloquent\Builder;
9 use Illuminate\Database\Eloquent\Factories\HasFactory;
10 use Illuminate\Database\Eloquent\Model;
11 use Illuminate\Database\Eloquent\Relations\BelongsTo;
12
13 class Sensor extends Model
14 {
15     /** @use HasFactory<\Database\Factories\SensorFactory> */
16     use BelongsToTenant, HasFactory;
17
18     protected $fillable = [
19         'user_id',
20         'device_id',
21         'farm_id',
```

```
22         'crop_id',
23         'name',
24         'uuid',
25         'type',
26         'lat',
27         'lon',
28     ];
29
30     public function device(): BelongsTo
31     {
32         return $this->belongsTo(Device::class);
33     }
34
35     public function farm(): BelongsTo
36     {
37         return $this->belongsTo(Farm::class);
38     }
39
40     public function user(): BelongsTo
41     {
42         return $this->belongsTo(User::class);
43     }
44
45     /**
46      * Scope to filter sensors whose related device is ONLINE.
47      */
48     public function scopeDeviceOnline(Builder $query): Builder
49     {
50         return $query->whereHas('device', function (Builder $q) {
51             $q->where('status', DeviceStatus::ONLINE->value);
52         });
53     }
54
55     /**
```

```
56      * Scope to return sensors that are relevant for farm-facing
      * UIs.
57      * Excludes internal-only sensor types (e.g. battery).
58      */
59      public function scopeFarmRelevant(Builder $query): Builder
60      {
61          return $query->whereIn('type', SensorType::
              farmRelevantValues());
62      }
63  }
```

Listing 71: Πλήρης ορισμός του μοντέλου Sensor

```
1  <?php
2
3  namespace App\Traits;
4
5  use App\Scopes\TenantScope;
6  use Illuminate\Database\Eloquent\Model;
7  use Illuminate\Support\Facades\Auth;
8
9  trait BelongsToTenant
10 {
11     /**
12      * The "booted" method of the trait.
13      *
14      * @return void
15      */
16     protected static function bootBelongsToTenant()
17     {
18         static::addGlobalScope(new TenantScope);
19
20         // Automatically set user_id on model creation if not
           explicitly set
21         static::creating(function (Model $model) {
```

```
22         if (! $model->user_id && Auth::check()) {
23             $model->user_id = Auth::id();
24         }
25     });
26 }
27
28 /**
29  * Get all models without tenant scope applied.
30  *
31  * @return \Illuminate\Database\Eloquent\Builder
32  */
33 public static function allTenants()
34 {
35     return static::withoutGlobalScope(TenantScope::class);
36 }
37
38 /**
39  * Get models for a specific tenant.
40  *
41  * @param int $userId
42  * @return \Illuminate\Database\Eloquent\Builder
43  */
44 public static function forTenant($userId)
45 {
46     return static::withoutGlobalScope(TenantScope::class)→
47         where('user_id', $userId);
48 }
49
50 /**
51  * Define a relationship to the user.
52  *
53  * @return \Illuminate\Database\Eloquent\Relations\BelongsTo
54  */
55 public function user()
```

```
55     {  
56         return $this->belongsTo(config('auth.providers.users.model'  
57             '));  
58     }  
59 }
```

Listing 72: Trait BelongsToTenant για αυτόματη πολυμισθωμένη απομόνωση

Α.2 Enums τύπων και χρονικών εύρων

```
1  <?php  
2  
3  namespace App\Enums;  
4  
5  enum TimeRange: string  
6  {  
7      case HOUR = '-1h';  
8      case DAY = '-24h';  
9      case WEEK = '-7d';  
10     case MONTH = '-30d';  
11     case QUARTER = '-90d';  
12  
13     public function label(): string  
14     {  
15         return match ($this) {  
16             self::HOUR => '1 hour',  
17             self::DAY => '24 hours',  
18             self::WEEK => '7 days',  
19             self::MONTH => '30 days',  
20             self::QUARTER => '90 days',  
21         };  
22     }  
23  
24     /**
```



```
25      * Aggregate window to use for this range when querying time-
26      series (InfluxDB 'aggregateWindow' every value).
27
28      */
29      public function aggregateWindow(): string
30      {
31          return match ($this) {
32              self::HOUR => '1m',
33              self::DAY => '15m',
34              self::WEEK => '1h',
35              self::MONTH => '6h',
36              self::QUARTER => '12h',
37          };
38      }
39
40      public static function options(): array
41      {
42          $out = [];
43          foreach (self::cases() as $case) {
44              $out[] = ['value' => $case->value, 'label' => $case->
45                  label()];
46          }
47
48          return $out;
49      }
50
51      public static function default(): self
52      {
53          return self::DAY;
54      }
55  }
```

Listing 73: Enum TimeRange με αντιστοίχιση aggregate window

Α.3 Υπηρεσία Χρονοσειρών

```
1 <?php
2
3 namespace App\Services\TimeSeries;
4
5 use App\Services\InfluxDBService;
6 use Carbon\Carbon;
7
8 class SensorTimeSeriesService
9 {
10     public function __construct(private readonly InfluxDBService
11         $influx) {}
12
13     // Round numeric values consistently for timeseries output
14     private function roundValue(mixed $val, int $decimals = 2):
15         mixed
16     {
17         if ($val === null) {
18             return null;
19         }
20         if (is_numeric($val)) {
21             return round((float) $val, $decimals);
22         }
23         return $val;
24     }
25
26     /**
27      * Get recent readings for a sensor, flattened to [{time,
28      *     value}]
29      */
30     public function recentReadings(int $sensorId, string $range =
31         '-7d', int $limit = 10): array
32     {
33         $sensorIdStr = (string) $sensorId;
```

```
31     $pipeline = <<<FLUX
32         |> range(start: {$range})
33         |> filter(fn: (r) => r._measurement == "
           sensor_measurement" and r.sensor_id == "{
           $sensorIdStr}" and r._field == "value")
34         |> sort(columns: ["_time"], desc: true)
35         |> limit(n: {$limit})
36     FLUX;
37     try {
38         $result = $this->influx->queryPipeline($pipeline);
39     } catch (\Throwable $e) {
40         return [];
41     }
42     $out = [];
43     foreach ($result as $table) {
44         $records = $table->records ?? [];
45         foreach ($records as $rec) {
46             // Normalize time to match DB format: "Y-m-d H:i:
               s"
47             $time = method_exists($rec, 'getTime') ? $rec->
               getTime() : ($rec->_time ?? ($rec['time'] ??
               null));
48             if ($time instanceof \DateTimeInterface) {
49                 $time = $time->format('Y-m-d H:i:s');
50             } elseif (is_string($time)) {
51                 try {
52                     $dt = new \DateTime($time);
53                     $time = $dt->format('Y-m-d H:i:s');
54                 } catch (\Exception $e) {
55                     // leave original string if parsing fails
56                 }
57             }
58             $rawValue = method_exists($rec, 'getValue') ?
               $rec->getValue() : ($rec->_value ?? ($rec['value
```

```
        ']' ?? null));  
59         $value = $this->roundValue($rawValue, 2);  
60  
61         $out[] = [  
62             'time' => $time,  
63             'value' => $value,  
64         ];  
65     }  
66 }  
67  
68     return $out;  
69 }  
70  
71 /**  
72  * Compute min/max/avg/count for a sensor over a range.  
73  */  
74 public function stats(int $sensorId, string $range = '-24h'):  
    array  
75 {  
76     $sensorIdStr = (string) $sensorId;  
77     $base = <<<FLUX  
78         |> range(start: {$range})  
79         |> filter(fn: (r) => r._measurement == "  
            sensor_measurement" and r.sensor_id == "{$  
            $sensorIdStr}" and r._field == "value")  
80         FLUX;  
81  
82     $stats = ['min' => null, 'max' => null, 'avg' => null, '  
        count' => 0];  
83  
84     try {  
85         // min  
86         $minRes = $this->influx->queryPipeline($base."<br>|> min  
            ()");
```

```
87         foreach ($minRes as $t) {
88             foreach (($t→records ?? []) as $rec) {
89                 $val = method_exists($rec, 'getValue') ? $rec
90                     →getValue() : ($rec→_value ?? ($rec['value
91                         '] ?? null));
92                 if ($val !== null) {
93                     $stats['min'] = $this→roundValue($val, 2)
94                     ;
95                     break 2;
96                 }
97             }
98         }
99
100         // max
101         $maxRes = $this→influx→queryPipeline($base."\n|> max
102             (");
103         foreach ($maxRes as $t) {
104             foreach (($t→records ?? []) as $rec) {
105                 $val = method_exists($rec, 'getValue') ? $rec
106                     →getValue() : ($rec→_value ?? ($rec['value
107                         '] ?? null));
108                 if ($val !== null) {
109                     $stats['max'] = $this→roundValue($val, 2)
110                     ;
111                     break 2;
112                 }
113             }
114         }
115
116         // mean
117         $meanRes = $this→influx→queryPipeline($base."\n|>
118             mean(");
119         foreach ($meanRes as $t) {
120             foreach (($t→records ?? []) as $rec) {
121                 $val = method_exists($rec, 'getValue') ? $rec
122                     →getValue() : ($rec→_value ?? ($rec['value
```

```
        ']' ?? null));  
112         if ($val !== null) {  
113             $stats['avg'] = $this->roundValue($val, 2)  
114             ;  
115             break 2;  
116         }  
117     }  
118     // count  
119     $countRes = $this->influx->queryPipeline($base."<n|>  
        count()");  
120     foreach ($countRes as $t) {  
121         foreach (($t->records ?? []) as $rec) {  
122             $val = method_exists($rec, 'getValue') ? $rec  
                ->getValue() : ($rec->_value ?? ($rec['value  
                ']' ?? null));  
123             if ($val !== null) {  
124                 $stats['count'] = (int) $val;  
125                 break 2;  
126             }  
127         }  
128     }  
129 } catch (\Throwable $e) {  
130     // ignore and return defaults  
131 }  
132  
133 // Progressive widening if nothing found - use TimeRange  
    enum values  
134 if (($stats['count'] ?? 0) === 0) {  
135     if ($range === \App\Enums\TimeRange::DAY->value) {  
136         return $this->stats($sensorId, \App\Enums\  
            TimeRange::WEEK->value);  
137     }  
138     if ($range === \App\Enums\TimeRange::WEEK->value) {
```

```
139         return $this->stats($sensorId, \App\Enums\  
            TimeRange::MONTH->value);  
140     }  
141 }  
142  
143     return $stats;  
144 }  
145  
146 public function chartReadings(int $sensorId, string $range =  
    '-24h'): array  
147 {  
148     $sensorIdStr = (string) $sensorId;  
149  
150     // Determine aggregate window based on requested range  
        via TimeRange enum  
151     $every = '15m';  
152     try {  
153         $tr = \App\Enums\TimeRange::tryFrom($range);  
154         if ($tr !== null) {  
155             $every = $tr->aggregateWindow();  
156         }  
157     } catch (\Throwable) {  
158         // fallback to default  
159     }  
160  
161     $pipeline = <<<FLUX  
162         |> range(start: {$range})  
163         |> filter(fn: (r) => r._measurement == "  
            sensor_measurement" and r.sensor_id == "{  
            $sensorIdStr}" and r._field == "value")  
164         |> aggregateWindow(every: {$every}, fn: mean,  
            createEmpty: false)  
165         |> sort(columns: ["_time"])  
166     FLUX;
```

```
167
168     try {
169         $result = $this->influx->queryPipeline($pipeline);
170     } catch (\Throwable $e) {
171         return [];
172     }
173
174     $out = [];
175     foreach ($result as $table) {
176         $records = $table->records ?? [];
177         foreach ($records as $rec) {
178             // Extract time (support getTime() if provided)
179             $time = method_exists($rec, 'getTime') ? $rec->
                getTime() : ($rec->_time ?? ($rec['time'] ??
                null));
180
181             if ($time instanceof \DateTimeInterface) {
182                 $ms = (int) ($time->getTimestamp() * 1000);
183             } elseif (is_string($time) || is_numeric($time))
184             {
185                 try {
186                     $ms = (int) (Carbon::parse($time)->
                        timestamp * 1000);
187                 } catch (\Throwable $e) {
188                     // skip unparseable times
189                     continue;
190                 }
191             } else {
192                 // unknown time format, skip
193                 continue;
194             }
195
196             $rawValue = method_exists($rec, 'getValue') ?
                $rec->getValue() : ($rec->_value ?? ($rec['value
```



```
        ']' ?? null));  
196         if ($rawValue === null) {  
197             continue;  
198         }  
199  
200         $value = $this->roundValue($rawValue, 2);  
201  
202         $out[] = [$ms, $value];  
203     }  
204 }  
205  
206 // Ensure ascending order by timestamp  
207 usort($out, fn ($a, $b) => $a[0] <=> $b[0]);  
208  
209 return $out;  
210 }  
211 }
```

Listing 74: Πλήρης ορισμός της SensorTimeSeriesService

Α.4 Controllers Εφαρμογής

```
1 <?php  
2  
3 namespace App\Http\Controllers;  
4  
5 use App\Http\Requests\FarmRequest;  
6 use App\Models\Farm;  
7 use App\Services\TimeSeries\FarmTimeSeriesService;  
8 use Illuminate\Foundation\Auth\Access\AuthorizesRequests;  
9 use Illuminate\Support\Facades\Auth;  
10 use Inertia\Inertia;  
11  
12 class FarmController extends Controller  
13 {
```

```
14     use AuthorizesRequests;
15
16     public function __construct() {}
17
18     /**
19      * Display a listing of the resource.
20      */
21     public function index()
22     {
23         $this->authorize('viewAny', Farm::class);
24         $farms = Farm::where('user_id', Auth::id())->get();
25
26         return Inertia::render('Farms/Index', [
27             'farms' => $farms,
28         ]);
29     }
30
31     /**
32      * Show the form for creating a new resource.
33      */
34     public function create()
35     {
36         return Inertia::render('Farms/Create');
37     }
38
39     /**
40      * Store a newly created resource in storage.
41      */
42     public function store(FarmRequest $request)
43     {
44         $validated = $request->validated();
45         $validated['user_id'] = Auth::id();
46
47         Farm::create($validated);
```

```
48
49     return redirect()→route('farms.index')
50         →with('success', 'Farm created successfully.');
```

51 }

52

53 */***

54 ** Display the specified resource.*

55 **/*

56 **public function** show(Farm \$farm)

57 {

58 \$this→authorize('view', \$farm);

59 \$sensors = \$farm→sensors()→farmRelevant()→get();

60

61 *// Calculate sensor statistics from database (exclude*

62 *internal-only types)*

63 \$totalSensors = \$farm→sensors()→farmRelevant()→count();

64 \$sensorTypeStats = \$farm→sensors()

65 →farmRelevant()

66 →selectRaw('type, COUNT(*) as count')

67 →whereNotNull('type')

68 →groupBy('type')

69 →pluck('count', 'type')

70 →toArray();

71 *// Calculate average of last reading per sensor type (*

72 *using sensors.last_reading)*

73 *// Only include sensors whose device is currently ONLINE*

74 \$lastAvgByType = \$farm→sensors()

75 →farmRelevant()

76 →deviceOnline()

77 →selectRaw('type, AVG(last_reading) as

78 avg_last_reading')

79 →whereNotNull('type')

80 →whereNotNull('last_reading')

```
79         →groupBy('type')
80         →pluck('avg_last_reading', 'type')
81         →mapWithKeys(function ($v, $k) {
82             // cast to float and keep keys as type
83             return [$k => $v !== null ? (float) $v : null];
84         })→toArray();
85
86         // Prepare time series service for deferred evaluation (
87         // do not call it now)
88
89         $ts = app(FarmTimeSeriesService::class);
90
91         // Split statistics so frontend can defer / lazy-load /
92         // poll them independently.
93
94         $sensorDbStats = [
95             'totalSensors' => $totalSensors,
96             'sensorTypeStats' => $sensorTypeStats,
97             'lastAvgByType' => $lastAvgByType,
98         ];
99
100         return Inertia::render('Farms/Show', [
101             'farm' => $farm,
102             'sensors' => $sensors,
103             'sensorDbStats' => $sensorDbStats,
104             // defer heavy/time-series work to a follow-up
105             // request so initial page is fast
106             'timeSeriesStats' => Inertia::defer(fn () => $ts→
107                 farmStats($farm, '-24h')),
108         ]);
109     }
110
111     /**
112      * Show the form for editing the specified resource.
113      */
114     public function edit(Farm $farm)
```

```
109     {
110         $this->authorize('update', $farm);
111
112         return Inertia::render('Farms/Edit', [
113             'farm' => $farm,
114         ]);
115     }
116
117     /**
118      * Update the specified resource in storage.
119      */
120     public function update(FarmRequest $request, Farm $farm)
121     {
122         $this->authorize('update', $farm);
123         $farm->update($request->validated());
124
125         return redirect()->route('farms.show', $farm)
126             ->with('success', 'Farm updated successfully.');
```

```
127     }
128
129     /**
130      * Remove the specified resource from storage.
131      */
132     public function destroy(Farm $farm)
133     {
134         $this->authorize('delete', $farm);
135         $farm->delete();
136
137         return redirect()->route('farms.index')
138             ->with('success', 'Farm deleted successfully.');
```

```
139     }
140 }
```

Listing 75: Πλήρης ορισμός του FarmController

```
1  <?php
2
3  namespace App\Http\Controllers;
4
5  use App\Enums\DeviceType;
6  use App\Enums\SensorType;
7  use App\Http\Requests\RegisterDeviceRequest;
8  use App\Models\Device;
9  use App\Services\MqttCredentialService;
10 use App\Services\TimeSeries\SensorTimeSeriesService;
11 use Illuminate\Foundation\Auth\Access\AuthorizesRequests;
12 use Illuminate\Http\Request;
13 use Illuminate\Support\Facades\Hash;
14 use Inertia\Inertia;
15
16 class DeviceController extends Controller
17 {
18     use AuthorizesRequests;
19
20     /**
21      * Display a listing of the resource.
22      */
23     public function index()
24     {
25         $devices = Device::all();
26
27         return Inertia::render('Devices/Index', [
28             'devices' => $devices,
29         ]);
30     }
31
32     /**
33      * Show the form for creating a new resource.
34      */
```

```
35 public function create()
36 {
37     return Inertia::render('Devices/Create', [
38         'deviceTypes' => collect(DeviceType::cases())->map(fn (
39             $type) => [
40                 'label' => DeviceType::labels()[$type->value],
41                 'value' => $type->value,
42             ]->values(),
43     ]);
44 }
45 /**
46  * Store a newly created resource in storage.
47  */
48 public function store(RegisterDeviceRequest $request)
49 {
50     $validated = $request->validated();
51
52     $device = Device::create([
53         'user_id' => $request->user()->id,
54         'uuid' => $validated['uuid'],
55         'secret' => Hash::make($validated['secret']), //
56         'name' => $validated['name'],
57         'type' => $validated['type'],
58         'status' => 'registered',
59         'lor_aes_key' => $validated['lor_aes_key'] ?? null,
60     ]);
61
62     return redirect()->route('devices.index')->with('success',
63         'Device registered successfully');
64 }
65 /**
```

```
66      * Display the specified resource.
67      */
68      public function show(Device $device)
69      {
70          $this->authorize('view', $device);
71
72          $sensors = $device->sensors()->get();
73
74          // Prefer device-level battery_level if present,
75          // otherwise use battery sensor last_reading
76          $batteryReading = $this->getBatteryLevel($device, $sensors
77              );
78
79          $ts = app(SensorTimeSeriesService::class);
80
81          // Prepare deferred battery time-series (only if a
82          // battery sensor exists)
83          $batterySensorId = optional($sensors->firstWhere('type',
84              SensorType::BATTERY->value))->id;
85
86          return Inertia::render('Devices/Show', [
87              'device' => $device,
88              'sensors' => $sensors,
89              'batteryReading' => $batteryReading,
90              'batteryTimeSeries' => Inertia::defer(fn () =>
91                  $batterySensorId ? $ts->recentReadings((int)
92                      $batterySensorId, '-24h', 100) : []),
93          ]);
94      }
95
96      /**
97       * Ensure MQTT credentials exist for the given device and
98       * return them via flash.
99       */
```



```
93     public function createMqttCredentials(Device $device)
94     {
95         $this->authorize('update', $device);
96
97         $service = app(MqttCredentialService::class);
98         $result = $service->createCredentials($device);
99
100        return redirect()->route('devices.show', $device)->with('
            mqtt_credentials', $result);
101    }
102
103    /**
104     * Show the form for editing the specified resource.
105     */
106    public function edit(string $id)
107    {
108        //
109    }
110
111    /**
112     * Update the specified resource in storage.
113     */
114    public function update(Request $request, string $id)
115    {
116        //
117    }
118
119    /**
120     * Remove the specified resource from storage.
121     */
122    public function destroy(Device $device)
123    {
124        $this->authorize('delete', $device);
125        $device->delete();
```

```
126
127     return redirect()→route('devices.index');
128 }
129
130 private function getBatteryLevel(Device $device, $sensors): ?
131     float
132 {
133     if ($device→battery_level !== null) {
134         return is_numeric($device→battery_level) ? (float)
135             $device→battery_level : null;
136     }
137
138     $batterySensor = $sensors→firstWhere('type', SensorType::
139         BATTERY→value);
140     if ($batterySensor && $batterySensor→last_reading !==
141         null) {
142         return is_numeric($batterySensor→last_reading) ? (
143             float) $batterySensor→last_reading : null;
144     }
145
146     return null;
147 }
```

Listing 76: Πλήρης ορισμός του DeviceController

```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use App\Enums\SensorType;
6 use App\Http\Requests\SensorRequest\ScanSensorRequest;
7 use App\Http\Requests\SensorRequest\StoreSensorRequest;
8 use App\Http\Requests\SensorRequest\UpdateSensorRequest;
```

```
9  use App\Http\Resources\SensorResource;
10 use App\Models\Farm;
11 use App\Models\Sensor;
12 use App\Services\TimeSeries\SensorTimeSeriesService;
13 use App\Enums\TimeRange;
14 use Illuminate\Foundation\Auth\Access\AuthorizesRequests;
15 use Inertia\Inertia;
16
17 class SensorController extends Controller
18 {
19     use AuthorizesRequests;
20
21     /**
22      * Display a listing of the resource.
23      */
24     public function index()
25     {
26         $this->authorize('viewAny', Sensor::class);
27         $sensors = request()->user()->sensors;
28         $farms = request()->user()->farms;
29
30         return Inertia::render('Sensors/Index', [
31             'sensors' => $sensors,
32             'farms' => $farms,
33         ]);
34     }
35
36     /**
37      * Show the form for creating a new resource.
38      */
39     public function create(?Farm $farm = null)
40     {
41         $this->authorize('create', Sensor::class);
42         $farms = request()->user()->farms;
```

```
43
44     return Inertia::render('Sensors/Create', [
45         'farms' => $farms,
46         'selectedFarm' => $farm,
47         'SensorTypes' => SensorType::values(),
48     ]);
49 }
50
51 /**
52  * Store a newly created resource in storage.
53  */
54 public function store(StoreSensorRequest $request)
55 {
56     $this->authorize('create', Sensor::class);
57     $validated = $request->validated();
58
59     $device = \App\Models\Device::where('uuid', $validated['
60         device_uuid'])
61         ->firstOrFail();
62
63     $this->authorize('view', $device); // This uses
64         DevicePolicy
65
66     $sensor = Sensor::create(
67         collect($validated)
68             ->except('device_uuid')
69             ->toArray()
70         + [
71             'user_id' => $request->user()->id,
72             'device_id' => $device->id,
73         ]
74     );
75
76     return redirect()->route('sensors.index');
```

```
75
76  /**
77   * Display the specified resource.
78   */
79  public function show(Sensor $sensor)
80  {
81      $this->authorize('view', $sensor);
82      $sensor->load(['farm', 'device']);
83
84      // Prepare time-series service; defer heavy/time-series
85      // work so initial page is fast
86      $ts = app(SensorTimeSeriesService::class);
87
88      // Respect optional ?range= query param; fall back to
89      // default
90      $range = request()->query('range', TimeRange::default()->
91      value);
92
93      return Inertia::render('Sensors/Show', [
94          'sensor' => (new SensorResource($sensor))->flat(request
95          ()),
96          // expose available ranges and which is selected
97          'rangeOptions' => TimeRange::options(),
98          'selectedRange' => $range,
99          // defer recent readings and stats (resolved
100          // asynchronously by Inertia)
101          'recentReadings' => Inertia::defer(fn () => $ts->
102          recentReadings($sensor->id, $range, 20)),
103          'stats' => Inertia::defer(fn () => $ts->stats($sensor->id
104          , $range)),
105          'chartData' => Inertia::defer(fn () => $ts->
106          chartReadings($sensor->id, $range)), // For chart,
107          // we want all readings in range
108      ]);
```

```
100     }
101
102     /**
103      * Show the form for editing the specified resource.
104      */
105     public function edit(Sensor $sensor)
106     {
107         $this->authorize('update', $sensor);
108         $sensor->load(['farm', 'device']);
109         $farms = request()->user()->farms;
110
111         return Inertia::render('Sensors/Edit', [
112             'sensor' => $sensor,
113             'farms' => $farms,
114         ]);
115     }
116
117     /**
118      * Update the specified resource in storage.
119      */
120     public function update(UpdateSensorRequest $request, Sensor
121         $sensor)
122     {
123         $this->authorize('update', $sensor);
124         $validated = $request->validated();
125
126         $sensor->update($validated);
127
128         return redirect()->route('sensors.show', $sensor->id);
129     }
130
131     /**
132      * Remove the specified resource from storage.
```

```
133     public function destroy(Sensor $sensor)
134     {
135         $this->authorize('delete', $sensor);
136         $sensor->delete();
137
138         return redirect()->route('sensors.index');
139     }
140
141     /**
142      * Scan or add a sensor by QR code (uuid). If the sensor
143      * exists, update it; otherwise, create it.
144      */
145     public function scan(ScanSensorRequest $request)
146     {
147         $validated = $request->validated();
148
149         $device = \App\Models\Device::where('uuid', $validated['
150             device_uuid'])->firstOrFail();
151
152         $sensor = Sensor::where('uuid', $validated['uuid'])->first
153             ();
154
155         $this->authorize('view', $device); // This uses
156             DevicePolicy
157
158         if ($sensor) {
159             // Update sensor, but never update device_id or
160             user_id from scan
161
162             $sensor->update(collect($validated)->except(['
163                 device_uuid', 'uuid'])->toArray());
164
165             return redirect()->route('sensors.index');
166         } else {
167             // Create sensor, ensure user can create
168
169             $this->authorize('create', Sensor::class);
170         }
171     }
```

```
161         $sensor = Sensor::create(  
162             collect($validated)  
163             →except('device_uuid')  
164             →toArray()  
165         + [  
166             'user_id' => $request→user()→id,  
167             'device_id' => $device→id,  
168         ]  
169     );  
170  
171     return redirect()→route('sensors.index');  
172 }  
173 }  
174 }
```

Listing 77: Πλήρης ορισμός του SensorController

A.5 REST API Controllers

```
1 <?php  
2  
3 namespace App\Http\Controllers\Api\V1;  
4  
5 use App\Enums\DeviceStatus;  
6 use App\Http\Controllers\Controller;  
7 use App\Models\Device;  
8 use App\Services\MqttCredentialService;  
9 use Illuminate\Http\Request;  
10 use Illuminate\Support\Facades\Hash;  
11  
12 class DeviceAuthController extends Controller  
13 {  
14     public function login(Request $request)  
15     {  
16         $request→validate([
```



```
17         'uuid' => 'required|string',
18         'secret' => 'required|string',
19     ]);
20
21     $device = Device::where('uuid', $request->uuid)->first();
22
23     if (! $device || ! Hash::check($request->secret, $device->
24         secret)) {
25         return response()->json(['message' => 'Invalid
26             credentials'], 401);
27     }
28
29     // Update status to ONLINE on successful auth
30     $device->update([
31         'status' => DeviceStatus::ONLINE,
32         'last_seen_at' => now(),
33     ]);
34
35     $token = $device->createToken('device-token')->
36         plainTextToken;
37
38     return response()->json(['token' => $token]);
39 }
40
41 // Device can call this after authenticating with the token
42 // to get mqtt credentials
43 public function provisionMqttCredentials(Request $request,
44     MqttCredentialService $svc)
45 {
46     $device = $request->user();
47
48     $creds = $svc->createCredentials($device);
49
50     if (isset($creds['error'])) {
```

```
46         return response()→json($creds , 503);  
47     }  
48  
49     return response()→json($creds , 200);  
50 }  
51 }
```

Listing 78: Πλήρης ορισμός του DeviceAuthController

```
1  <?php  
2  
3  namespace App\Http\Controllers\Api\V1;  
4  
5  use App\Enums\DeviceStatus;  
6  use App\Http\Controllers\Controller;  
7  use App\Http\Requests\Api\V1\DeviceDataRequest;  
8  use App\Models\Device;  
9  use App\Services\SensorDataService;  
10 use Illuminate\Http\Request;  
11  
12 class DeviceDataController extends Controller  
13 {  
14     public function store(DeviceDataRequest $request,  
15         SensorDataService $sensorDataService)  
16     {  
17         $device = $request→user();  
18         // Update device status and last seen  
19         $device→update([  
20             'status' ⇒ DeviceStatus::ONLINE,  
21             'last_seen_at' ⇒ now(),  
22         ]);  
23         $sensorPayloads = $request→validated()['sensors'];  
24         $response = $sensorDataService→processSensorData($device,  
25             $sensorPayloads);  
26     }  
27 }
```

```
25         return response()→json($response, 200);
26     }
27
28     public function mqttBrokerWebhook(Request $request,
29         SensorDataService $sensorDataService)
30     {
31         $device = $request['username'];
32         $payload = $request['payload'];
33         // Find device by uuid (assuming username is the device
34         // UUID)
35         $device = Device::where('uuid', $request['username'])→
36             first();
37         if (! $device) {
38             return response()→json(['message' ⇒ 'Device not found
39                 '], 404);
40         }
41         // // Update device status and last seen
42         $device→update([
43             'status' ⇒ DeviceStatus::ONLINE,
44             'last_seen_at' ⇒ now(),
45         ]);
46         $sensorPayloads = $payload ? json_decode($payload, true)
47             : [];
48         $sensorPayloads = $sensorPayloads['sensors'] ?? [];
49         $response = $sensorDataService→processSensorData($device,
50             $sensorPayloads);
51
52         return response()→json(['message' ⇒ 'Webhook received'],
53             200);
54     }
55 }
```

Listing 79: Πλήρης ορισμός του DeviceDataController

```
2
3 namespace App\Http\Controllers\Api\V1;
4
5 use App\Enums\DeviceStatus;
6 use App\Enums\DeviceType;
7 use App\Exceptions\LoRaFrameCounterGapException;
8 use App\Exceptions\LoRaReplayException;
9 use App\Http\Controllers\Controller;
10 use App\Http\Requests\Api\V1\LoRaWebhookRequest;
11 use App\Models\Device;
12 use App\Services\LoRaCryptoService;
13 use App\Services\SensorDataService;
14 use Illuminate\Support\Facades\Log;
15 use Illuminate\Support\Facades\Validator;
16
17 class LoRaDataController extends Controller
18 {
19     public function webhook(
20         LoRaWebhookRequest $request,
21         LoRaCryptoService $crypto,
22         SensorDataService $sensorDataService,
23     ) {
24
25         // Log raw incoming request for debugging
26         Log::info('Received LoRa webhook', $request->all());
27
28         // Decode the gateway JSON from the EMQX envelope.
29         // Some EMQX configs send a wrapped 'request' object;
30         // others send the fields at the top level.
31
32         $validated = $request->validated();
33
34         $payloadJson = data_get($validated, 'request.payload',
35             data_get($validated, 'payload'));
36
37         if (! is_string($payloadJson)) {
```

```
34         return response()→json(['message' ⇒ 'Missing or
           invalid inner payload.'], 422);
35     }
36
37     $gatewayPayload = json_decode($payloadJson, true);
38     if (! is_array($gatewayPayload)) {
39         return response()→json(['message' ⇒ 'Invalid inner
           payload JSON.'], 422);
40     }
41
42     // Validate the gateway JSON structure
43     $innerValidator = Validator::make($gatewayPayload, [
44         'gateway_id' ⇒ ['required', 'string', 'max:255'],
45         'raw_payload' ⇒ ['required', 'string', 'regex:/^[0-9A
           -Fa-f]+$/'],
46         'rssi' ⇒ ['nullable', 'integer'],
47         'snr' ⇒ ['nullable', 'numeric'],
48     ]);
49
50     if ($innerValidator→fails()) {
51         return response()→json([
52             'message' ⇒ 'Invalid LoRa payload.',
53             'errors' ⇒ $innerValidator→errors(),
54         ], 422);
55     }
56
57     $validated = $innerValidator→validated();
58
59     // Decode hex → binary and extract UUID, frame counter,
           and ciphertext
60     // Layout: [1B uuid_len] [uuid_len B: UUID ASCII] [4B
           fcnt LE] [encrypted payload: N×6B]
61     $rawBytes = hex2bin($validated['raw_payload']);
62     if ($rawBytes === false || strlen($rawBytes) < 12) {
```

```
63         return response()→json(['message' ⇒ 'Invalid
        raw_payload structure.'], 422);
64     }
65
66     $uuidLen = ord($rawBytes[0]);
67     $headerLen = 1 + $uuidLen + 4; // 1B uuid_len + UUID + 4B
        fcnt
68
69     if ($uuidLen < 1 || strlen($rawBytes) < $headerLen + 6 ||
        (strlen($rawBytes) - $headerLen) % 6 !== 0) {
70         return response()→json(['message' ⇒ 'Invalid
        raw_payload structure.'], 422);
71     }
72
73     $uuid = substr($rawBytes, 1, $uuidLen);
74     $fcnt = unpack('V', substr($rawBytes, 1 + $uuidLen, 4))
        [1];
75     $ciphertext = base64_encode(substr($rawBytes, $headerLen)
        );
76
77     // log the incoming webhook for debugging
78     Log::info('Received LoRa webhook', [
79         'username' ⇒ data_get($validated, 'request.username',
        data_get($validated, 'username')),
80         'gateway_payload' ⇒ $validated,
81     ]);
82
83     // log the extracted values for debugging
84     Log::debug('Extracted LoRa payload components', [
85         'uuid' ⇒ $uuid,
86         'frame_counter' ⇒ $fcnt,
87         'ciphertext_length' ⇒ strlen($ciphertext),
88     ]);
89
```

```
90      // Look up the LoRa device by its embedded UUID
91      $device = Device::allTenants()
92          →where('uuid', $uuid)
93          →where('type', DeviceType::LORA→value)
94          →first();
95
96      if (! $device) {
97          return response()→json(['message' ⇒ 'LoRa device not
98              found.'], 404);
99
100      // Validate frame counter (anti-replay)
101      try {
102          $crypto→validateFrameCounter($device, $fcnt);
103      } catch (LoRaReplayException $e) {
104          Log::warning('LoRa replay attack blocked', [
105              'device_id' ⇒ $device→id,
106              'incoming_fcnt' ⇒ $fcnt,
107              'stored_fcnt' ⇒ $device→lora_frame_counter,
108          ]);
109
110          return response()→json(['message' ⇒ 'Replay detected.
111              '], 409);
112      } catch (LoRaFrameCounterGapException $e) {
113          Log::warning('LoRa frame counter gap exceeded', [
114              'device_id' ⇒ $device→id,
115              'incoming_fcnt' ⇒ $fcnt,
116              'stored_fcnt' ⇒ $device→lora_frame_counter,
117          ]);
118
119          return response()→json(['message' ⇒ 'Frame counter
120              gap exceeded.'], 422);
121      }
```

```
121      // Decrypt the payload
122      try {
123          $rawBinaryDecrypted = $crypto->decrypt($device, $fcnt,
124              $ciphertext);
125      } catch (\Throwable $e) {
126          Log::error('LoRa decryption failed', [
127              'device_id' => $device->id,
128              'error' => $e->getMessage(),
129          ]);
130
131          return response()->json(['message' => 'Decryption
132              failed.'], 422);
133      }
134
135      // Deserialize binary into sensor readings
136      try {
137          $readings = $crypto->deserialize($rawBinaryDecrypted);
138      } catch (\InvalidArgumentException $e) {
139          Log::error('LoRa deserialization failed', [
140              'device_id' => $device->id,
141              'error' => $e->getMessage(),
142          ]);
143
144          return response()->json(['message' => 'Payload
145              deserialization failed.'], 422);
146      }
147
148      // Update device status only; gateway metrics are not
149      // stored on device
150      $device->update([
151          'status' => DeviceStatus::ONLINE,
152          'last_seen_at' => now(),
153      ]);
```



```
151      // Map deserialized readings to the device's pre-
      registered sensors by UUID prefix
152      $sensors = $device->sensors()->get();
153      $sensorPayloads = [];
154      foreach ($sensors as $sensor) {
155          $prefix = substr($sensor->uuid, 0, 4);
156          if (array_key_exists($prefix, $readings)) {
157              $sensorPayloads[] = [
158                  'uuid' => $sensor->uuid,
159                  'value' => $readings[$prefix],
160              ];
161          }
162      }
163
164      $response = $sensorDataService->processSensorData($device,
          $sensorPayloads);
165
166      return response()->json(['message' => 'LoRa data processed.
          ']' + $response, 200);
167  }
168 }
```

Listing 80: Πλήρης ορισμός του LoRaDataController

Α.6 Υπηρεσίες API

```
1 <?php
2
3 namespace App\Services;
4
5 use App\Models\Device;
6 use Illuminate\Support\Facades\Log;
7
8 class MqttCredentialService
9 {
```

```
10  /**
11   * Ensure the device has mqtt credentials. Returns ['username
12   *   ' => ..., 'password' => ..., 'created' => bool]
13   */
14  public function createCredentials(Device $device): array
15  {
16      if ($device->mqtt_username && $device->mqtt_password) {
17          return [
18              'mqtt_broker_url' => config('services.emqx.url', '
19              /'),
20              'username' => $device->mqtt_username,
21              'password' => $device->mqtt_password,
22              'created' => false,
23          ];
24      }
25
26      $username = $device->uuid;
27      $password = bin2hex(random_bytes(16));
28
29      // Resolve EmqxService from the container so it can be
30      // mocked in tests
31      $emqx = app(EmqxService::class);
32
33      try {
34          $result = $emqx->createUser($username, $password);
35      } catch (\Throwable $e) {
36          Log::error('MQTT credential creation failed: EMQX
37          service error', [
38              'device_id' => $device->id,
39              'device_uuid' => $device->uuid,
40              'error' => $e->getMessage(),
41          ]);
42      }
```

```
39      // Broker unreachable or other error: do not create
      credentials
40      return [
41          'mqtt_broker_url' => config('services.emqx.url', '
              /'),
42          'created' => false,
43          'error' => 'emqx_unavailable',
44      ];
45  }
46
47  // EmqxService returns an array with 'status' on failure;
      treat that as failure
48  if (is_array($result) && array_key_exists('status',
      $result)) {
49      Log::error('MQTT credential creation failed: EMQX
          returned error response', [
50          'device_id' => $device->id,
51          'device_uuid' => $device->uuid,
52          'emqx_response' => $result,
53      ]);
54
55      return [
56          'mqtt_broker_url' => config('services.emqx.url', '
              /'),
57          'created' => false,
58          'error' => 'emqx_unavailable',
59      ];
60  }
61
62  $device->update([
63      'mqtt_username' => $username,
64      'mqtt_password' => $password,
65  ]);
66
```

```
67         return [  
68             'mqtt_broker_url' => config('services.emqx.url', '/'),  
69             'username' => $username,  
70             'password' => $password,  
71             'created' => true,  
72         ];  
73     }  
74 }
```

Listing 81: Πλήρης ορισμός της MqttCredentialService

```
1  <?php  
2  
3  namespace App\Services;  
4  
5  use App\Events\SensorReadingEvent;  
6  use App\Jobs\ProcessSensorInfluxData;  
7  use App\Models\Sensor;  
8  
9  class SensorDataService  
10 {  
11     public function processSensorData($device, array  
12         $sensorPayloads): array  
13     {  
14         $uuids = collect($sensorPayloads)→pluck('uuid')→all();  
15         $sensors = Sensor::allTenants()  
16             →where('device_id', $device→id)  
17             →whereIn('uuid', $uuids)  
18             →get()  
19             →keyBy('uuid');  
20         $missingUuids = [];  
21         $writtenCount = 0;  
22         foreach ($sensorPayloads as $sensor) {  
23             $sensorModel = $sensors→get($sensor['uuid']);  
24             if (!$sensorModel) {
```

```
24         $missingUids[] = $sensor['uuid'];
25
26         continue;
27     }
28
29     // Dispatch job to write to InfluxDB asynchronously
30     ProcessSensorInfluxData::dispatch($sensorModel, (
31         float) $sensor['value'], time());
32
33     $sensorModel->last_reading = $sensor['value'];
34     $sensorModel->last_reading_at = now();
35     $sensorModel->save();
36     // Broadcast the new reading to any listeners (
37     private per-sensor channel)
38     event(new SensorReadingEvent(
39         $sensorModel->id,
40         [
41             'value' => $sensorModel->last_reading,
42             'time' => optional($sensorModel->
43                 last_reading_at)->toDateTimeString() ?: (
44                 string) $sensorModel->last_reading_at ?:
45                 now()->toDateTimeString(),
46         ]
47     ));
48     $writtenCount++;
49 }
50
51 $response = ['message' => 'Data received.'];
52 if (count($missingUids) > 0) {
53     $response['missing_uids'] = $missingUids;
54 }
```

53 }

Listing 82: Πλήρης ορισμός της SensorDataService

```
1 <?php
2
3 namespace App\Services;
4
5 use App\Exceptions\LoRaFrameCounterGapException;
6 use App\Exceptions\LoRaReplayException;
7 use App\Models\Device;
8
9 class LoRaCryptoService
10 {
11     public const MAX_FCNT_GAP = 10_000;
12
13     /**
14      * Validate the incoming frame counter against the device's
15      * stored counter.
16      *
17      * Updates the device's counter on success.
18      *
19      * @throws LoRaReplayException
20      * @throws LoRaFrameCounterGapException
21      */
22     public function validateFrameCounter(Device $device, int
23         $incomingFcnt): void
24     {
25         $storedFcnt = (int) $device->lora_frame_counter;
26
27         if ($incomingFcnt <= $storedFcnt) {
28             throw new LoRaReplayException($incomingFcnt,
29                 $storedFcnt);
30         }
31
32         $gap = $incomingFcnt - $storedFcnt;
```

```
29         if ($gap > self::MAX_FCNT_GAP) {
30             throw new LoRaFrameCounterGapException($gap, self::
                MAX_FCNT_GAP);
31         }
32
33         $device->update(['lora_frame_counter' => $incomingFcnt]);
34     }
35
36     /**
37      * Decrypt a Base64-encoded AES-128-CTR ciphertext using the
        device's LoRa key.
38      *
39      * Nonce layout (16 bytes):
40      *     [4-byte CRC32(device.uuid) LE] [4-byte frame counter LE]
        [8-byte zero padding]
41      */
42     public function decrypt(Device $device, int $fcnt, string
        $base64Ciphertext): string
43     {
44         $ciphertext = base64_decode($base64Ciphertext, true);
45         if ($ciphertext === false) {
46             throw new \InvalidArgumentException('Invalid Base64
                ciphertext.');
```

```
56         .pack('V', $fcnt)                                // 4 bytes:
               frame counter (little-endian uint32)
57         .str_repeat("\x00", 8);                          // 8 bytes:
               zero padding
58
59         $plaintext = openssl_decrypt(
60             $ciphertext,
61             'aes-128-ctr',
62             $key,
63             OPENSSL_RAW_DATA | OPENSSL_ZERO_PADDING,
64             $nonce
65         );
66
67         if ($plaintext === false) {
68             throw new \RuntimeException('Decryption failed for
               device ' . $device->uuid);
69         }
70
71         return $plaintext;
72     }
73
74     /**
75      * Deserialize a variable-length binary payload into sensor
       readings.
76
77      * Layout:  $N \times 6$ -byte records, where each record is:
78      *   Bytes 0-3: first 4 characters of the sensor UUID (ASCII)
79      *   Bytes 4-5: sensor value (int16 LE, scaled  $\times 100$ )
80      *
81      * @return array<string, float> Keyed by 4-char UUID prefix,
       value divided by 100
82      */
83     public function deserialize(string $binary): array
84     {
```



```
85     $len = strlen($binary);
86     if ($len < 6 || $len % 6 !== 0) {
87         throw new \InvalidArgumentException(
88             "Binary payload length must be a non-zero
89             multiple of 6 bytes, got {$len}."
90         );
91     }
92
93     $readings = [];
94     $count = $len / 6;
95     for ($i = 0; $i < $count; $i++) {
96         $chunk = substr($binary, $i * 6, 6);
97         $prefix = substr($chunk, 0, 4);
98         $raw = unpack('v1v', substr($chunk, 4, 2))['v'];
99         // Convert unsigned int16 to signed int16 (values can
100         // be negative)
101         if ($raw >= 0x8000) {
102             $raw -= 0x10000;
103         }
104         $readings[$prefix] = round($raw / 100, 2);
105     }
106
107     return $readings;
108 }
```

Listing 83: Πλήρης ορισμός της LoRaCryptoService

Α.7 Υπηρεσία Δεδομένων και Ουρά Εργασιών

```
1  <?php
2
3  namespace App\Jobs;
4
5  use App\Models\Sensor;
```

```
6 use App\Services\InfluxDBService;
7 use App\Services\SensorMeasurementPayloadFactory;
8 use Illuminate\Contracts\Queue\ShouldQueue;
9 use Illuminate\Foundation\Queue\Queueable;
10 use Illuminate\Queue\InteractsWithQueue;
11 use Illuminate\Queue\SerializesModels;
12 use Illuminate\Support\Facades\Log;
13
14 class ProcessSensorInfluxData implements ShouldQueue
15 {
16     use InteractsWithQueue, Queueable, SerializesModels;
17
18     /**
19      * The number of times the job may be attempted.
20      */
21     public int $tries = 3;
22
23     /**
24      * The maximum number of seconds the job can run before
25      * timing out.
26      */
27     public int $timeout = 30;
28
29     /**
30      * Create a new job instance.
31      */
32     public function __construct(
33         public Sensor $sensor,
34         public float $value,
35         public int $timestamp
36     ) {
37         // Set queue name for time-series data processing
38         $this->onQueue('sensor-data');
```

```
39
40     /**
41      * Execute the job.
42      */
43     public function handle(InfluxDBService $influx): void
44     {
45         try {
46             $payload = SensorMeasurementPayloadFactory::make(
47                 $this->sensor, $this->value);
48
49             // Override timestamp if provided (for historical
50             // data)
51             if ($this->timestamp) {
52                 $payload['time'] = $this->timestamp;
53             }
54
55             $influx->writeArray($payload);
56         } catch (\Exception $e) {
57             // Log the error for monitoring
58             Log::error('Failed to write sensor data to InfluxDB',
59                 [
60                     'sensor_id' => $this->sensor->id,
61                     'value' => $this->value,
62                     'error' => $e->getMessage(),
63                 ]);
64
65             // Re-throw to trigger job retry mechanism
66             throw $e;
67         }
68     }
69
70     /**
71      * Handle a job failure.
72      */
```

```
70     public function failed(\Throwable $exception): void
71     {
72         Log::error('Sensor InfluxDB job failed permanently', [
73             'sensor_id' => $this->sensor->id,
74             'value' => $this->value,
75             'exception' => $exception->getMessage(),
76         ]);
77     }
78 }
```

Listing 84: Πλήρης ορισμός του ProcessSensorInfluxData job

```
1  <?php
2
3  namespace App\Services;
4
5  use App\Models\Sensor;
6
7  class SensorMeasurementPayloadFactory
8  {
9      /**
10       * Build the payload array for InfluxDB writeArray.
11       *
12       * @param float|int $value
13       */
14     public static function make(Sensor $sensorModel, $value):
15         array
16     {
17         return [
18             'name' => 'sensor_measurement',
19             'tags' => [
20                 'user_id' => $sensorModel->user_id,
21                 'farm_id' => $sensorModel->farm_id,
22                 'sensor_id' => $sensorModel->id,
23                 'sensor_type' => $sensorModel->type,
```

```
23         ],
24         'fields' => [
25             // cast to float to avoid integer/float column
26             // type conflicts in InfluxDB
27             'value' => (float) $value,
28         ],
29         // use integer seconds to match precision=s
30         'time' => time(), // use server time
31     ];
32 }
```

Listing 85: Πλήρης ορισμός του SensorMeasurementPayloadFactory

```
1 <?php
2
3 namespace App\Services;
4
5 use InfluxDB2\Client;
6 use InfluxDB2\Model\WritePrecision;
7 use InfluxDB2\Point;
8
9 class InfluxDBService
10 {
11     protected $client;
12
13     protected $bucket;
14
15     protected $org;
16
17     public function __construct()
18     {
19         $this->client = new Client([
20             'url' => config('services.influxdb.url'),
21             'token' => config('services.influxdb.token'),
```

```
22         });
23         $this->bucket = config('services.influxdb.bucket');
24         $this->org = config('services.influxdb.org');
25     }
26
27     /**
28      * Expose bucket for consumers that might need it (read-only)
29      */
30     public function bucket(): string
31     {
32         return (string) $this->bucket;
33     }
34
35     /**
36      * Expose org for consumers that might need it (read-only)
37      */
38     public function org(): string
39     {
40         return (string) $this->org;
41     }
42
43     /**
44      * Helper to run a Flux pipeline by automatically prefixing
45      * the bucket.
46      *
47      * Provide only the pipeline starting from '|> range(...)'
48      *
49      * Example:
50      *     queryPipeline("|> range(start: -1h) |> filter(fn: (r) =>
51      *         r._measurement == \"m\")")
52      */
53     public function queryPipeline(string $pipeline)
54     {
55         $flux = 'from(bucket: "'.$this->bucket.'")\n'.ltrim(
56             $pipeline);
```

```
53
54     return $this->query($flux);
55 }
56
57 /**
58  * Write data using Line Protocol string
59  */
60 public function writeLineProtocol(string $data, $precision =
    WritePrecision::S)
61 {
62     $writeApi = $this->client->createWriteApi();
63     $writeApi->write($data, $precision, $this->bucket, $this->
        org);
64 }
65
66 /**
67  * Write data using a Point object
68  */
69 public function writePoint(Point $point, $precision =
    WritePrecision::S)
70 {
71     $writeApi = $this->client->createWriteApi();
72     $writeApi->write($point, $precision, $this->bucket, $this->
        org);
73 }
74
75 /**
76  * Write data using an array structure
77  */
78 public function writeArray(array $data, $precision =
    WritePrecision::S)
79 {
80     $writeApi = $this->client->createWriteApi();
81     $writeApi->write($data, $precision, $this->bucket, $this->
```

```
        org);  
  
82    }  
  
83  
  
84    /**  
85     * Properly close the client connection  
86     */  
87    public function close()  
88    {  
89        $this->client->close();  
90    }  
  
91  
  
92    /**  
93     * Run a Flux query and return the result.  
94     * Use this for queries that return data.  
95     */  
96    // Note: Flux queries should be properly formatted with 'from  
97        (bucket: ...)' included.  
98    // If you need to prepend the bucket, use queryPipeline()  
99        instead.  
100    // This method is for direct queries that already include the  
101        bucket.  
102  
103    public function query(string $flux)  
104    {  
105        $queryApi = $this->client->createQueryApi();  
  
        return $queryApi->query($flux, $this->org);  
    }  
}
```

Listing 86: Πλήρης ορισμός της InfluxDBService

Β' Παράρτημα Β: Κώδικας Firmware

Το παράρτημα αυτό περιλαμβάνει τον πλήρη πηγαίο κώδικα του firmware της πλατφόρμας *Agronos*: τους κόμβους πεδίου (WiFi και LoRa) και το LoRa Gateway. Τα αρχεία

παρατίθενται ομαδοποιημένα κατά λειτουργικό υποσύστημα.

Β.1 Ρύθμιση Μεταγλώττισης

```
1 ; PlatformIO Project Configuration File
2 ;
3 ; Build options: build flags, source filter
4 ; Upload options: custom upload port, speed and extra flags
5 ; Library options: dependencies, extra library storages
6 ; Advanced options: extra scripting
7 ;
8 ; Please visit documentation for the other options and examples
9 ; https://docs.platformio.org/page/projectconf.html
10
11 [env]
12 framework = arduino
13 monitor_speed = 115200
14 lib_deps =
15     adafruit/Adafruit Unified Sensor@^1.1.5
16     adafruit/DHT sensor library@^1.4.5
17     bblanchon/ArduinoJson@^6.19.5
18     knolleary/PubSubClient@^2.8
19     robtillaart/DHT20 @ ^0.3.1
20
21 [env:esp32dev]
22 platform = espressif32
23 board = esp32dev
24 src_filter =
25     +<*>
26     -<main_lora.cpp>
27     -<lora_payload.cpp>
28     -<lora_crypto.cpp>
29     -<lora_fcmt.cpp>
30     -<lora_radio.cpp>
31
```

```
32 [env:esp32-c6-devkitc-1]
33 platform = espressif32
34 board = esp32-c6-devkitc-1
35 build_flags =
36     -D ARDUINO_USB_MODE=1
37     -D ARDUINO_USB_CDC_ON_BOOT=1
38 src_filter =
39     +<*>
40     -<main_lora.cpp>
41     -<lora_payload.cpp>
42     -<lora_crypto.cpp>
43     -<lora_fcmt.cpp>
44     -<lora_radio.cpp>
45
46 [env:ttgo-lora32-v21]
47 platform = espressif32
48 board = ttgo-lora32-v21
49 build_flags =
50     -D LORA_NODE=1
51 lib_deps =
52     ${env.lib_deps}
53     sandeepmistry/LoRa@^0.8.0
54 src_filter =
55     +<*>
56     -<main.cpp>
57     -<wifi_portal.cpp>
58     -<auth.cpp>
59     -<data_sender.cpp>
60     -<mqtt_client.cpp>
61
62 [env:ttgo-lora32-v21-wifi]
63 platform = espressif32
64 board = ttgo-lora32-v21
65 src_filter =
```

```
66     +<*>
67     -<main_lora.cpp>
68     -<lora_payload.cpp>
69     -<lora_crypto.cpp>
70     -<lora_fcmt.cpp>
71     -<lora_radio.cpp>
```

Listing 87: platformio.ini — ρυθμίσεις μεταγλώττισης και build targets

Β.2 Διεπαφή Αισθητήρων και Εργοστάσιο

```
1  #pragma once
2
3  #include <stddef.h>
4  #include <vector>
5  #include <memory>
6
7  // Sensor descriptor used by the sensor factory
8  struct SensorConfig {
9      const char* type;           // e.g. "DHT11_TEMP", "DHT11_HUM", "
          DS18B20", "SIMULATED"
10     int pin;                   // pin or -1 if not applicable
11     const char* uuid;          // sensor uuid used by the server
12     const char* displayName;    // human-readable name for UI
          display (e.g. "Temperature", "Humidity")
13 };
14
15 class SensorBase {
16 public:
17     virtual ~SensorBase() = default;
18     virtual const char* uuid() const = 0;
19     // read a single float value from the sensor, return true on
          success
20     virtual bool read(float &out) = 0;
21 };
```

```
22
23 // Creator function type used by the registry
24 using CreatorFunc = SensorBase*(*)(const SensorConfig& cfg);
25
26 // Registration API: sensor implementation files should call this
    (via a static
27 // registrar) to register themselves. This lets the factory
    create instances
28 // without modifying the factory code when new sensor classes are
    added.
29 bool registerSensorFactory(const char* name, CreatorFunc creator)
    ;
30
31 // Create a sensor instance by name using the registered creators
    . Returns
32 // nullptr if no creator is found.
33 SensorBase* createSensorByType(const char* name, const
    SensorConfig& cfg);
34
35 // Factory helpers (implemented in src/sensor_factory.cpp)
36 std::vector<std::unique_ptr<SensorBase>> createSensors();
```

Listing 88: Ορισμός διεπαφής SensorBase και δομής SensorConfig (sensor.h)

```
1 #pragma once
2
3 #include "sensor.h"
4
5 // Templated creator helper to be used by each sensor
    implementation when registering
6 // with the factory. Requires sensor classes exposing a
    constructor (int pin, const char* uuid)
7 // For sensors that don't match that signature, define a small
    wrapper in the cpp file.
8
```

```
9  template <typename T>
10  SensorBase* create_sensor_impl(const SensorConfig& cfg) {
11      return new T(cfg.pin, cfg.uuid);
12  }
13
14  // Specialization for sensors that don't accept pin+uuid can be
    provided in their own files.
```

Listing 89: Πρότυπο δημιουργίας αισθητήρων (sensor_creator.h)

```
1  #include "sensor.h"
2  #include "config.h"
3  #include <Arduino.h>
4  #include <cstring>
5  #include <vector>
6  #include <memory>
7  #include <algorithm>
8
9  // Registry using a vector of pairs to avoid manual malloc and
    linked list
10 static std::vector<std::pair<const char*, CreatorFunc>>& registry
    () {
11     static std::vector<std::pair<const char*, CreatorFunc>> r;
12     return r;
13 }
14
15 bool registerSensorFactory(const char* name, CreatorFunc creator)
    {
16     registry().emplace_back(name, creator);
17     return true;
18 }
19
20 SensorBase* createSensorByType(const char* name, const
    SensorConfig& cfg) {
21     for (auto &p : registry()) {
```

```
22         if (strcmp(p.first, name) == 0) return p.second(cfg);
23     }
24     return nullptr;
25 }
26
27 std::vector<std::unique_ptr<SensorBase>> createSensors() {
28     std::vector<std::unique_ptr<SensorBase>> vec;
29     vec.reserve(SENSOR_CONFIG_COUNT);
30
31     for (size_t i = 0; i < SENSOR_CONFIG_COUNT; ++i) {
32         const SensorConfig &cfg = SENSOR_CONFIGS[i];
33         SensorBase* inst = createSensorByType(cfg.type, cfg);
34         if (!inst) {
35             // Unknown type → fallback to registered SIMULATED
36             creator if present
37             inst = createSensorByType("SIMULATED", cfg);
38             if (!inst) {
39                 // skip sensor if we cannot create it
40                 continue;
41             }
42         }
43         vec.emplace_back(inst);
44     }
45     return vec;
46 }
```

Listing 90: Registry εργοστασίου και δημιουργία αισθητήρων (sensor_factory.cpp)

Β.3 Υλοποιήσεις Αισθητήρων

```
1 #pragma once
2
3 #include <DHT.h>
4
5 // Return a DHT* shared per-pin. Caller does not own the pointer.
```

```
6 DHT* getSharedDht(int pin);
```

Listing 91: Κοινόχρηστο στιγμιότυπο DHT ανά GPIO pin (dht_shared.h)

```
1 #include "dht_shared.h"
2 #include <Arduino.h>
3 #include <stdlib.h>
4
5 DHT* getSharedDht(int pin) {
6     struct Entry { int pin; DHT* dht; Entry* next; };
7     static Entry* head = nullptr;
8     for (Entry* e = head; e != nullptr; e = e->next) {
9         if (e->pin == pin) return e->dht;
10    }
11    Entry* ne = (Entry*)malloc(sizeof(Entry));
12    if (!ne) return nullptr;
13    ne->pin = pin;
14    ne->dht = new DHT(pin, DHT11);
15    ne->dht->begin();
16    ne->next = head;
17    head = ne;
18    return ne->dht;
19 }
```

Listing 92: Υλοποίηση shared DHT registry (dht_shared.cpp)

```
1 #include "sensor.h"
2 #include "config.h"
3 #include "dht_shared.h"
4 #include "sensor_creator.h"
5 #include <Arduino.h>
6
7 class DHT11TemperatureReader : public SensorBase {
8 public:
9     DHT11TemperatureReader(int pin, const char* uid)
10     : pin_(pin), uuid_(uid), dht_(getSharedDht(pin)) {}
```

```
11     const char* uuid() const override { return uuid_; }
12     bool read(float &out) override {
13         if (!dht_) return false;
14         float t = dht_>readTemperature();
15         if (isnan(t)) return false;
16         out = t;
17         return true;
18     }
19 private:
20     int pin_;
21     const char* uuid_;
22     DHT* dht_;
23 };
24
25 static bool _reg = registerSensorFactory("DHT11TemperatureReader"
    , create_sensor_impl<DHT11TemperatureReader>);
```

Listing 93: Αισθητήρας θερμοκρασίας DHT11 (dht11_temp.cpp)

```
1 #include "sensor.h"
2 #include "config.h"
3 #include "dht_shared.h"
4 #include "sensor_creator.h"
5 #include <Arduino.h>
6
7 class DHT11HumidityReader : public SensorBase {
8 public:
9     DHT11HumidityReader(int pin, const char* uid)
10     : pin_(pin), uuid_(uid), dht_(getSharedDht(pin)) {}
11     const char* uuid() const override { return uuid_; }
12     bool read(float &out) override {
13         if (!dht_) return false;
14         float h = dht_>readHumidity();
15         if (isnan(h)) return false;
16         out = h;
```



```
17         return true;
18     }
19 private:
20     int pin_;
21     const char* uuid_;
22     DHT* dht_;
23 };
24
25 static bool _reg = registerSensorFactory("DHT11HumidityReader",
    create_sensor_impl<DHT11HumidityReader>);
```

Listing 94: Αισθητήρας υγρασίας DHT11 (dht11_hum.cpp)

```
1 #include "sensor.h"
2 #include "config.h"
3 #include "sensor_creator.h"
4 #include <Arduino.h>
5 #include <DHT20.h>
6 #include <Wire.h>
7
8 /**
9  * DHT20 Temperature and Humidity Sensor (I2C)
10  * - Uses robtillaart/DHT20 library
11  * - Default I2C address is 0x38
12  * - Multiple readers share the same DHT20 instance to avoid I2C
    conflicts
13  */
14
15 static DHT20* getSharedDHT20() {
16     static DHT20* dht = nullptr;
17     if (!dht) {
18         // Initialize I2C with pins defined in config.h
19         Wire.begin(I2C_SDA, I2C_SCL);
20         dht = new DHT20(&Wire);
21         if (!dht->begin()) {
```

```
22         Serial.println("DHT20: Failed to initialize sensor!");
23         ;
24         return nullptr;
25     }
26     Serial.println("DHT20: Sensor initialized");
27     return dht;
28 }
29
30 // Guarded read wrapper: only performs I2C read if 2+ seconds
31 // have passed
32 // Both temperature and humidity readers call this, but only the
33 // first triggers the actual read
34 static bool readSharedDHT20() {
35     static unsigned long lastReadTime = 0;
36     DHT20* dht = getSharedDHT20();
37     if (!dht) return false;
38
39     unsigned long now = millis();
40     const unsigned long READ_INTERVAL = 2000; // DHT20 requires 2
41     // seconds between reads
42
43     // Only perform an actual read if 2+ seconds have passed
44     if (now - lastReadTime >= READ_INTERVAL) {
45         int status = dht->read();
46         if (status != DHT20_OK) {
47             Serial.print("DHT20 Read Error: ");
48             Serial.println(status);
49             return false;
50         }
51         lastReadTime = now;
52     }
53
54     return true;
55 }
```

```
52 }  
53  
54 class DHT20TemperatureReader : public SensorBase {  
55 public:  
56     DHT20TemperatureReader(int pin, const char* uid)  
57     : uuid_(uid), dht_(getSharedDHT20()) {}  
58  
59     const char* uuid() const override { return uuid_; }  
60  
61     bool read(float &out) override {  
62         if (!dht_) return false;  
63  
64         // Use guarded read (only performs I2C transaction once  
65         // per 2 seconds)  
66         if (!readSharedDHT20()) return false;  
67  
68         out = dht_→getTemperature();  
69         return true;  
70     }  
71 private:  
72     const char* uuid_;  
73     DHT20* dht_;  
74 };  
75  
76 class DHT20HumidityReader : public SensorBase {  
77 public:  
78     DHT20HumidityReader(int pin, const char* uid)  
79     : uuid_(uid), dht_(getSharedDHT20()) {}  
80  
81     const char* uuid() const override { return uuid_; }  
82  
83     bool read(float &out) override {  
84         if (!dht_) return false;
```

```
85
86     // Use guarded read (only performs I2C transaction once
87     // per 2 seconds)
88     if (!readSharedDHT20()) return false;
89
90     out = dht_→getHumidity();
91     return true;
92 }
93 private:
94     const char* uuid_;
95     DHT20* dht_;
96 };
97
98 // Register the factory creators
99 static bool _reg_temp = registerSensorFactory("
100     DHT20TemperatureReader", create_sensor_impl<
101     DHT20TemperatureReader>);
102 static bool _reg_hum = registerSensorFactory("DHT20HumidityReader
103     ", create_sensor_impl<DHT20HumidityReader>);
```

Listing 95: Αισθητήρες θερμοκρασίας και υγρασίας DHT20 (dht20_sensors.cpp)

```
1 #include "sensor.h"
2 #include "config.h"
3 #include "sensor_creator.h"
4 #include <Arduino.h>
5
6 /**
7  * DFRobot Capacitive Soil Moisture Sensor (SEN0193)
8  * - Analog output: 0-3V
9  * - Connected to GPIO32 (ADC1_CH4)
10  * - Requires calibration (air value = dry, water value = wet)
11  * - Returns moisture percentage: 0% (dry) to 100% (wet)
12  */
```

```
13 class SoilMoistureSensor : public SensorBase {
14 public:
15     SoilMoistureSensor(int pin, const char* uid)
16     : pin_(pin), uuid_(uid) {}
17
18     const char* uuid() const override { return uuid_; }
19
20     bool read(float &out) override {
21         // Average multiple readings for stability
22         const int numSamples = 10;
23         long sum = 0;
24
25         for (int i = 0; i < numSamples; i++) {
26             sum += analogRead(pin_);
27             delay(10);
28         }
29
30         int rawValue = sum / numSamples;
31
32         // Calibration values (MUST be determined empirically)
33         // These are estimates scaled from Arduino 10-bit to
34         ESP32 12-bit ADC
35         // To calibrate:
36         // 1. Expose sensor to air, note the raw value → update
37         airValue
38         // 2. Submerge in water (to limit line), note raw value
39         → update waterValue
40         const int airValue = SOIL_MOISTURE_AIR_VALUE;    // Dry
41         soil (high value, low capacitance)
42         const int waterValue = SOIL_MOISTURE_WATER_VALUE; // Wet
43         soil (low value, high capacitance)
44
45         // Clamp to calibrated range
46         if (rawValue > airValue) rawValue = airValue;
```

```
42     if (rawValue < waterValue) rawValue = waterValue;
43
44     // Convert to percentage: 100% = wet (waterValue), 0% =
45         dry (airValue)
46
47     float moisturePercent = 100.0f * (float)(airValue -
48         rawValue) / (float)(airValue - waterValue);
49
50
51     out = moisturePercent;
52     return true;
53 }
54
55 private:
56     int pin_;
57     const char* uuid_;
58 };
59
60 // Auto-register with factory using standard creator template
61 static bool _reg = registerSensorFactory("SoilMoistureSensor",
62     create_sensor_impl<SoilMoistureSensor>);
```

Listing 96: Αισθητήρας εδαφικής υγρασίας (soil_moisture.cpp)

```
1  #include "sensor.h"
2  #include "config.h"
3  #include "sensor_creator.h"
4  #include <Arduino.h>
5
6  /**
7   * DFRobot FireBeetle 2 ESP32-C6 Battery Level Sensor
8   * - Reads battery voltage via ADC0 (GPIO pin 0)
9   * - Built-in voltage divider (2:1 ratio) on the board
10  * - 12-bit ADC resolution (0-4096)
11  * - Returns battery percentage based on LiPo discharge curve
12  *
13  * Battery Specifications:
```

```
14  * - Type: Polymer Lithium Ion Battery (single-cell LiPo)
15  * - Nominal Voltage: 3.7V
16  * - Capacity: 1200mAh
17  *
18  * Voltage ranges:
19  * - 4.2V = 100% (fully charged)
20  * - 3.7V = ~50% (nominal voltage)
21  * - 3.0V = 0% (cutoff voltage)
22  */
23  class BatteryLevelSensor : public SensorBase {
24  public:
25      BatteryLevelSensor(int pin, const char* uid)
26      : adc_pin_(pin), uuid_(uid) {
27          analogReadResolution(12); // Set 12-bit resolution
28                                     (0-4096)
29      }
30
31      const char* uuid() const override { return uuid_; }
32
33      bool read(float &out) override {
34          analogReadResolution(12);
35
36          constexpr int NUM_SAMPLES = 10;
37          long sumMv = 0;
38          int lastRaw = 0;
39          for (int i = 0; i < NUM_SAMPLES; ++i) {
40              lastRaw = analogRead(adc_pin_);
41              int mv = analogReadMilliVolts(adc_pin_);
42              sumMv += mv;
43              delay(10);
44          }
45
46          int avgMv = (int)(sumMv / NUM_SAMPLES);
```

```
47      // print out averaged values:
48      Serial.print("ADC analog value (last) = ");
49      Serial.println(lastRaw);
50      Serial.print("ADC average millivolts = ");
51      Serial.print(avgMv);
52      Serial.println(" mV");
53      Serial.print("BAT millivolts value = ");
54      Serial.print(avgMv * 2);
55      Serial.println(" mV");
56      Serial.println("-----");
57
58      float batteryPercent = voltageToBatteryPercent(avgMv * 2)
59          ;
60      out = batteryPercent;
61      return true;
62  }
63  private:
64      int adc_pin_;
65      const char* uuid_;
66
67      /**
68       * Convert battery voltage to percentage using LiPo discharge
69       * curve
70       * Based on typical single-cell LiPo (3.7V nominal) discharge
71       * characteristics
72       */
73
74      float voltageToBatteryPercent(float millivolts) const {
75          const float voltage = millivolts / 1000.0f; // Convert to
76              volts
77
78          // LiPo voltage-to-percentage mapping (approximate)
79          // These values provide a reasonable linear approximation
80          const float VOLTAGE_MAX = 4.2f; // 100% - fully charged
```



```
77     const float VOLTAGE_MIN = 3.0f;  // 0% - cutoff voltage
78
79     // Clamp to valid range
80     if (voltage >= VOLTAGE_MAX) return 100.0f;
81     if (voltage <= VOLTAGE_MIN) return 0.0f;
82
83     // For more accuracy, use a piecewise linear
84     // approximation of the LiPo discharge curve:
85     // 4.2V-4.0V: 100%-90% (rapid drop at top)
86     // 4.0V-3.7V: 90%-50% (gradual decline)
87     // 3.7V-3.4V: 50%-20% (moderate decline)
88     // 3.4V-3.0V: 20%-0% (rapid drop at bottom)
89
90     if (voltage >= 4.0f) {
91         // 100% to 90%: 4.2V to 4.0V
92         return 90.0f + ((voltage - 4.0f) / 0.2f) * 10.0f;
93     } else if (voltage >= 3.7f) {
94         // 90% to 50%: 4.0V to 3.7V
95         return 50.0f + ((voltage - 3.7f) / 0.3f) * 40.0f;
96     } else if (voltage >= 3.4f) {
97         // 50% to 20%: 3.7V to 3.4V
98         return 20.0f + ((voltage - 3.4f) / 0.3f) * 30.0f;
99     } else {
100         // 20% to 0%: 3.4V to 3.0V
101         return ((voltage - 3.0f) / 0.4f) * 20.0f;
102     }
103 };
104
105 // Auto-register with factory using standard creator template
106 static bool _reg = registerSensorFactory("BatteryLevelSensor",
    create_sensor_impl<BatteryLevelSensor>);
```

Listing 97: Αισθητήρας στάθμης μπαταρίας (battery_level.cpp)

```
1 #include "sensor.h"
2 #include "config.h"
3 #include "sensor_creator.h"
4
5 class SimulatedSensor : public SensorBase {
6 public:
7     SimulatedSensor(int pin_unused, const char* uid) : uuid_(uid)
8         {}
9     const char* uuid() const override { return uuid_; }
10    bool read(float &out) override {
11        static float v = 20.0f;
12        v += 0.13f;
13        out = v;
14        return true;
15    }
16 private:
17     const char* uuid_;
18 };
19 static bool _reg = registerSensorFactory("SIMULATED",
20     create_sensor_impl<SimulatedSensor>);
```

Listing 98: Προσομοιωμένος αισθητήρας για δοκιμές (simulated.cpp)

```
1 #pragma once
2
3 #include "sensor.h" // SensorConfig is defined in the sensor API
4     header
5 //#include "Test-Device-2.h" // Device-specific credentials +
6     sensor list
7 #include "Test-Device-LoRa-Battery.h" // Device-specific
8     credentials + sensor list
9
10 // Network / server
11 constexpr const char* BASE_URL = "https://agronos."
```

```
kalogeitonas.xyz";

9

10 // Captive portal AP
11 constexpr const char* AP_SSID = "ESP_Config";
12 constexpr const char* AP_PASS = ""; // optional
13
14 // Button configuration
15 constexpr unsigned long BUTTON_LONG_PRESS_MS = 10000; // 10
    seconds to trigger reset
16
17 #ifdef LORA_NODE
18 // On TTGO LoRa32 V2.1, GPIO 14 is used by LoRa RST.
19 // Use GPIO 0 (BOOT button) for factory reset.
20 constexpr int BUTTON_PIN = 0;
21 #elif defined(CONFIG_IDF_TARGET_ESP32C6)
22 // On ESP32-C6, deep sleep wakeup is supported only on GPIO 0-7 (
    RTC/VDDPST1 pins).
23 // GPIO 14 is not supported for deep sleep wakeup on C6.
24 // We select GPIO 7 (D11 on FireBeetle 2 C6) as the button pin.
25 constexpr int BUTTON_PIN = 7;
26 #else
27 // On standard ESP32, GPIO 14 is an RTC pin and supports wakeup.
28 constexpr int BUTTON_PIN = 14;
29 #endif
30
31 #if defined(CONFIG_IDF_TARGET_ESP32C6)
32 constexpr int I2C_SDA = 19;
33 constexpr int I2C_SCL = 20;
34 #else
35 constexpr int I2C_SDA = 21;
36 constexpr int I2C_SCL = 22;
37 #endif
38
39 // DHT sensor
```

```
40 constexpr int DHT11_PIN = 21;
41
42 // non-blocking read interval
43 constexpr unsigned long SENSORS_READ_INTERVAL_MS = 10 * 60 *
    1000; // 10 minutes
44
45 // Auth manager
46 constexpr unsigned long AUTH_RETRY_INTERVAL_MS = 30000;
47
48
49 // ===== MQTT CONFIGURATION =====
50 // Enable/Disable MQTT protocol (set to false to use HTTP only)
51 constexpr bool MQTT_ENABLED = true;
52
53 // MQTT connection settings
54 constexpr uint16_t AGRONOS_MQTT_PORT = 1883; // 8883
    for TLS, 1883 for plain
55 constexpr bool AGRONOS_MQTT_USE_TLS = false; // Use
    secure connection
56 constexpr uint16_t AGRONOS_MQTT_KEEPALIVE = 60; // Keep-
    alive interval (seconds)
57 constexpr uint16_t AGRONOS_MQTT_RECONNECT_DELAY = 5000; //
    Reconnection delay (ms)
58 constexpr bool AGRONOS_MQTT_CLEAN_SESSION = true; // Start
    with clean session
59
60 // MQTT topics (will be formatted with device UUID)
61 // Use topics matching broker ACL: devices/<username>/# so EMQX
    allows publishes
62 constexpr const char* AGRONOS_MQTT_TOPIC_DATA = "devices/%s/
    sensors";
63 constexpr const char* AGRONOS_MQTT_TOPIC_STATUS = "devices/%s/
    status";
64 constexpr const char* AGRONOS_MQTT_TOPIC_COMMAND = "devices/%s/
```

```
        commands";

65

66 // MQTT QoS levels
67 constexpr int AGRONOS_MQTT_QOS_DATA = 1;      // At least once
        for sensor data
68 constexpr int AGRONOS_MQTT_QOS_STATUS = 0;    // Fire and forget
        for status
69
70 // NOTE: MQTT server, username, and password are fetched
        dynamically
71 // from the backend using the JWT token after HTTP authentication
        .
72 // =====
73
74 // Soil moisture sensor calibration (SEN0193)
75 constexpr int SOIL_MOISTURE_AIR_VALUE = 2941;  // Dry
76 constexpr int SOIL_MOISTURE_WATER_VALUE = 1324; // Wet
77
78 // ===== LoRa CONFIGURATION (TTGO LoRa32 V2.1)
        =====
79 #ifdef LORA_NODE
80
81 // LoRa SPI pins (TTGO LoRa32 V2.1)
82 constexpr int LORA_SCK_PIN    = 5;
83 constexpr int LORA_MISO_PIN   = 19;
84 constexpr int LORA_MOSI_PIN   = 27;
85 constexpr int LORA_SS_PIN     = 18;
86 constexpr int LORA_RST_PIN    = 14;
87 constexpr int LORA_DIO0_PIN   = 26;
88
89 // LoRa radio settings (must match gateway defaults)
90 constexpr long  LORA_FREQUENCY = 868000000;  // 868 MHz (Europe
        )
91 constexpr int   LORA_SF          = 10;      // Spreading
```

```
Factor 10
92 constexpr long LORA_BW = 125000; // 125 kHz
    bandwidth
93 constexpr int LORA_CR = 5; // Coding Rate
    4/5
94 constexpr uint8_t LORA_SYNC_WORD = 0x12; // Private
    network sync word
95 constexpr int LORA_TX_POWER = 20; // 20 dBm
    transmit power
96
97 // Frame counter management
98 constexpr uint32_t FCNT_NVS_SAVE_INTERVAL = 100; // Save to NVS
    every N transmissions
99 constexpr uint32_t FCNT_COLD_BOOT_GAP = 100; // Jump on cold
    boot (< backend MAX_FCNT_GAP=100000)
100
101 // TTGO LoRa32 V2.1 misc pins
102 constexpr int LORA_LED_PIN = 25; // Built-in LED
103 constexpr int LORA_BATTERY_PIN = 35; // Battery ADC (ADC1_CH7)
104
105 #endif // LORA_NODE
106 //
    =====
```

Listing 99: Κεντρικές σταθερές ρύθμισης (config.h)

```
1 #pragma once
2
3 #include "sensor.h" // SensorConfig is defined in the sensor API
    header
4
5 // === Device Profile (example) ===
6 // This file combines device credentials and sensor list for a
    single device.
```

```
7 // Copy this file to 'device_profile.h' and update values for
   each device.
8 // The 'device_profile.h' should be gitignored on per-device
   repositories.
9
10 // Device credentials
11 constexpr const char* DEFAULT_UUID = "Test-Device-1";
12 constexpr const char* DEFAULT_SECRET = "Test-Device-1";
13
14 // Sensor configuration
15 constexpr SensorConfig SENSOR_CONFIGS[] = {
16     { "DHT11TemperatureReader", 21, "Test-Device-1-Sensor-1", "
       Temperature" },
17     { "DHT11HumidityReader", 21, "Test-Device-1-Sensor-2", "
       Humidity" },
18     { "SoilMoistureSensor", 32, "Soil-Moisture-1", "Soil Moisture
       " },
19     { "DHT20TemperatureReader", -1, "DHT20-Temp-1", "DHT20
       Temperature" },
20     { "DHT20HumidityReader", -1, "DHT20-Hum-1", "DHT20 Humidity"
       }
21 };
22 constexpr size_t SENSOR_CONFIG_COUNT = sizeof(SENSOR_CONFIGS) /
   sizeof(SENSOR_CONFIGS[0]);
23
24 // === LoRa Configuration (only used when LORA_NODE is defined)
   ===
25 // No LORA_DEVICE_ID needed - the firmware uses DEFAULT_UUID for
   device
26 // identification in the LoRa packet and for nonce construction (
   via CRC32).
27
28 // 128-bit AES key shared with the backend's 'lora_aes_key'
   column (hex string in DB).
```

```
29 // Example: hex "000102030405060708090a0b0c0d0e0f" → byte array  
    below.  
30 constexpr uint8_t LORA_AES_KEY[16] = {  
31     0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,  
32     0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F  
33 };
```

Listing 100: Παράδειγμα προφίλ συσκευής (device_profile.h.example)

```
1  #pragma once  
2  #include <Preferences.h>  
3  #include <Arduino.h>  
4  
5  struct MqttCredentials {  
6      String server;  
7      String username;  
8      String password;  
9      bool isValid;  
10 };  
11  
12 struct DeviceConfig {  
13     String baseUrl;  
14     unsigned long readIntervalMs;  
15     bool mqttEnabled;  
16 };  
17  
18 class Storage {  
19 public:  
20     Storage();  
21  
22     // Wifi credentials  
23     // Returns true if credentials exist  
24     bool getWifiCreds(String &ssid, String &pass);  
25     void setWifiCreds(const String &ssid, const String &pass);  
26
```



```
27 // Auth token
28 String getToken();
29 void setToken(const String &token);
30
31 // MQTT credentials
32 bool getMqttCredentials(MqttCredentials &creds);
33 void setMqttCredentials(const String &server, const String &
    username, const String &password);
34 void clearMqttCredentials();
35 bool hasMqttCredentials();
36
37 // Device configuration - Dependency Injection
38 Storage* loadDefaults(const DeviceConfig& defaults);
39
40 // Device configuration - Transparent Getters
41 String getBaseUrl();
42 unsigned long getReadIntervalMs();
43 bool getMqttEnabled();
44
45 // Device configuration - Individual Setters
46 void setBaseUrl(const String &url);
47 void setReadIntervalMs(unsigned long ms);
48 void setMqttEnabled(bool enabled);
49
50 // Device configuration - Atomic Setter
51 void saveConfig(const DeviceConfig& cfg);
52
53 // LoRa frame counter persistence
54 uint32_t getLoraFcnt();
55 void setLoraFcnt(uint32_t fcnt);
56
57 // Clear stored credentials and token
58 void clearAll();
59
```

```
60 private:
61     Preferences prefs;
62
63     // Config cache and defaults
64     DeviceConfig _cache;
65     DeviceConfig _defaults;
66     bool _configLoaded = false;
67
68     // Lazy-loading helper
69     void ensureConfigLoaded();
70 };
```

Listing 101: Διεπαφή μόνιμης αποθήκευσης NVS (storage.h)

```
1  #include "storage.h"
2
3  Storage::Storage() {
4      // nothing
5  }
6
7  bool Storage::getWifiCreds(String &ssid, String &pass) {
8      prefs.begin("wifi", false);
9      ssid = prefs.getString("ssid", "");
10     pass = prefs.getString("pass", "");
11     prefs.end();
12     return ssid.length() > 0;
13 }
14
15 void Storage::setWifiCreds(const String &ssid, const String &pass
    ) {
16     prefs.begin("wifi", false);
17     String oldS = prefs.getString("ssid", "");
18     String oldP = prefs.getString("pass", "");
19     if (ssid != oldS || pass != oldP) {
20         prefs.putString("ssid", ssid);
```

```
21     prefs.putString("pass", pass);
22 }
23 prefs.end();
24 }
25
26 String Storage::getToken() {
27     prefs.begin("auth", false);
28     String t = prefs.getString("token", "");
29     prefs.end();
30     return t;
31 }
32
33 void Storage::setToken(const String &token) {
34     prefs.begin("auth", false);
35     String old = prefs.getString("token", "");
36     if (token != old) prefs.putString("token", token);
37     prefs.end();
38 }
39
40 bool Storage::getMqttCredentials(MqttCredentials &creds) {
41     prefs.begin("mqtt", false);
42     creds.server = prefs.getString("server", "");
43     creds.username = prefs.getString("username", "");
44     creds.password = prefs.getString("password", "");
45     prefs.end();
46
47     creds.isValid = (creds.server.length() > 0 &&
48                     creds.username.length() > 0 &&
49                     creds.password.length() > 0);
50     return creds.isValid;
51 }
52
53 void Storage::setMqttCredentials(const String &server, const
    String &username, const String &password) {
```

```
54     prefs.begin("mqtt", false);
55     prefs.putString("server", server);
56     prefs.putString("username", username);
57     prefs.putString("password", password);
58     prefs.end();
59     Serial.println("MQTT credentials saved to storage");
60 }
61
62 void Storage::clearMqttCredentials() {
63     prefs.begin("mqtt", false);
64     prefs.clear();
65     prefs.end();
66     Serial.println("MQTT credentials cleared");
67 }
68
69 bool Storage::hasMqttCredentials() {
70     MqttCredentials creds;
71     return getMqttCredentials(creds);
72 }
73
74 // ===== Device Configuration (Config Struct
75 //      Pattern) =====
76
77 Storage* Storage::loadDefaults(const DeviceConfig& defaults) {
78     _defaults = defaults;
79     return this;
80 }
81
82 void Storage::ensureConfigLoaded() {
83     if (_configLoaded) return;
84
85     prefs.begin("config", true); // read-only mode
86
87     // Load all config values with defaults as fallback
```

```
87     _cache.baseUrl = prefs.getString("base_url", _defaults.baseUrl)
88     ;
89
90     // Handle mqttEnabled with isKey check for proper default
91     if (prefs.isKey("mqtt_enabled")) {
92         _cache.mqttEnabled = prefs.getBool("mqtt_enabled", _defaults.
93             mqttEnabled);
94     } else {
95         _cache.mqttEnabled = _defaults.mqttEnabled;
96     }
97
98     prefs.end();
99     _configLoaded = true;
100 }
101
102 String Storage::getBaseUrl() {
103     ensureConfigLoaded();
104     return _cache.baseUrl;
105 }
106
107 unsigned long Storage::getReadIntervalMs() {
108     ensureConfigLoaded();
109     return _cache.readIntervalMs;
110 }
111
112 bool Storage::getMqttEnabled() {
113     ensureConfigLoaded();
114     return _cache.mqttEnabled;
115 }
116
117 void Storage::saveConfig(const DeviceConfig& cfg) {
118     ensureConfigLoaded(); // Ensure cache is populated
```

```
118
119     prefs.begin("config", false);
120
121     // Only write changed fields to prevent Flash wear
122     if (cfg.baseUrl != _cache.baseUrl) {
123         prefs.putString("base_url", cfg.baseUrl);
124     }
125
126     if (cfg.readIntervalMs != _cache.readIntervalMs) {
127         prefs.putULong("interval_ms", cfg.readIntervalMs);
128     }
129
130     if (cfg.mqttEnabled != _cache.mqttEnabled) {
131         prefs.putBool("mqtt_enabled", cfg.mqttEnabled);
132     }
133
134     prefs.end();
135
136     // Update cache to match new values
137     _cache = cfg;
138 }
139
140 void Storage::setBaseUrl(const String &url) {
141     DeviceConfig cfg = _cache;
142     cfg.baseUrl = url;
143     saveConfig(cfg);
144 }
145
146 void Storage::setReadIntervalMs(unsigned long ms) {
147     DeviceConfig cfg = _cache;
148     cfg.readIntervalMs = ms;
149     saveConfig(cfg);
150 }
151
```

```
152 void Storage::setMqttEnabled(bool enabled) {
153     DeviceConfig cfg = _cache;
154     cfg.mqttEnabled = enabled;
155     saveConfig(cfg);
156 }
157
158 uint32_t Storage::getLoraFcnt() {
159     prefs.begin("lora", true);
160     uint32_t fcnt = prefs.getULong("fcnt", 0);
161     prefs.end();
162     return fcnt;
163 }
164
165 void Storage::setLoraFcnt(uint32_t fcnt) {
166     prefs.begin("lora", false);
167     prefs.putULong("fcnt", fcnt);
168     prefs.end();
169 }
170
171 void Storage::clearAll() {
172     prefs.begin("wifi", false);
173     prefs.clear();
174     prefs.end();
175     prefs.begin("auth", false);
176     prefs.clear();
177     prefs.end();
178     prefs.begin("mqtt", false);
179     prefs.clear();
180     prefs.end();
181     prefs.begin("config", false);
182     prefs.clear();
183     prefs.end();
184     prefs.begin("lora", false);
185     prefs.clear();
```

```
186     prefs.end();  
187  
188     // Invalidate cache since NVS was cleared  
189     _configLoaded = false;  
190 }
```

Listing 102: Υλοποίηση μόνιμης αποθήκευσης NVS (storage.cpp)

Β.4 Κόμβος WiFi

```
1  /*  
2   * This sketch demonstrates how to scan WiFi networks.  
3   * The API is almost the same as with the WiFi Shield library,  
4   * the most obvious difference being the different file you need  
5     to include:  
6   */  
7  #include <WiFi.h>  
8  #include "wifi_portal.h"  
9  #include "storage.h"  
10 #include "auth.h"  
11 #include <HTTPClient.h>  
12 #include "config.h"  
13 #include "data_sender.h"  
14 #include <math.h>  
15  
16 #include "sensor.h"  
17 #include <esp_sleep.h>  
18  
19 #include "mqtt_client.h"  
20  
21 const char* apSsid = AP_SSID;  
22 const char* apPass = AP_PASS; // optional  
23  
24 // Replace global Preferences with Storage instance  
25 Storage storage;
```



```
25
26 // Runtime configuration variables (loaded from storage in setup)
27 String baseUrl;
28 bool mqttEnabled;
29 unsigned long readIntervalMs;
30
31 // These will be constructed after loading config
32 WifiPortal* portal = nullptr;
33 AuthManager* auth = nullptr;
34 DataSender* sender = nullptr;
35 MqttClient* mqttClient = nullptr;
36
37 // (timing handled by deep sleep across boots)
38
39 // Backoff for failed send attempts while connected (ms)
40 constexpr unsigned long SEND_RETRY_BACKOFF_MS = 10UL * 1000UL; //
    30s
41 static unsigned long lastSendAttempt = 0;
42
43 // Sensor instances created from config
44 static std::vector<std::unique_ptr<SensorBase>> sensors;
45
46 // Forward declaration for one-time provisioning function
47 static void oneTimeProvisioning();
48
49 // Check if button is held for more than 10 seconds to reset all
    storage
50 static void checkButtonReset() {
51     if (digitalRead(BUTTON_PIN) == LOW) {
52         Serial.println("Button pressed, checking for long press
            ...");
53         unsigned long pressStart = millis();
54
55         while (digitalRead(BUTTON_PIN) == LOW) {
```

```
56         if (millis() - pressStart >= BUTTON_LONG_PRESS_MS) {
57             Serial.println("Long press detected! Wiping all
                    storage data...");
58             storage.clearAll();
59             Serial.println("Storage cleared. Restarting
                    device...");
60             delay(1000);
61             ESP.restart();
62         }
63         delay(100);
64     }
65     Serial.println("Button released before 10 seconds");
66 }
67 }
68
69 void tryAutoConnect() {
70     String ssid, pass;
71     if (!storage.getWifiCreds(ssid, pass)) return;
72     if (ssid.length() == 0) return;
73     WiFi.mode(WIFI_STA);
74     WiFi.begin(ssid.c_str(), pass.c_str());
75     unsigned long start = millis();
76     while (millis() - start < 10000) {
77         if (WiFi.status() == WL_CONNECTED) {
78             Serial.println("Connected to saved WiFi");
79             return;
80         }
81         delay(200);
82     }
83     Serial.println("Failed to connect to saved WiFi");
84     //storage.setWifiCreds("", ""); // clear invalid creds
85     Serial.println("Starting portal due to failed WiFi connection")
            ;
86 }
```

```
87
88 void setup()
89 {
90     Serial.begin(115200);
91
92     // Button pin setup
93     pinMode(BUTTON_PIN, INPUT_PULLUP);
94
95     // Check for long press to reset storage
96     checkButtonReset();
97
98     // Initialize storage defaults from config.h
99     DeviceConfig defaults = {
100         .baseUrl = BASE_URL,
101         .readIntervalMs = SENSORS_READ_INTERVAL_MS,
102         .mqttEnabled = MQTT_ENABLED
103     };
104     storage.loadDefaults(defaults);
105
106     // Load device configuration from storage (uses defaults if
107         not set)
108     baseUrl = storage.getBaseUrl();
109     readIntervalMs = storage.getReadIntervalMs();
110     mqttEnabled = storage.getMqttEnabled();
111
112     Serial.println("Device Configuration:");
113     Serial.print("  Base URL: "); Serial.println(baseUrl);
114     Serial.print("  MQTT Enabled: "); Serial.println(mqttEnabled
115         ? "Yes" : "No");
116     Serial.print("  Read Interval: "); Serial.print(
117         readIntervalMs / 1000); Serial.println(" seconds");
118
119     // Now construct objects with loaded configuration
```

```
117     portal = new WifiPortal(storage, apSsid, apPass, DEFAULT_UUID
118         , DEFAULT_SECRET,
119             SENSOR_CONFIGS, SENSOR_CONFIG_COUNT,
120             baseUrl.c_str(), mqttEnabled,
121             readIntervalMs);
122     auth = new AuthManager(storage, baseUrl.c_str(), DEFAULT_UUID
123         , DEFAULT_SECRET, AUTH_RETRY_INTERVAL_MS);
124     sender = new DataSender(storage, baseUrl.c_str());
125     mqttClient = new MqttClient(storage, DEFAULT_UUID);
126
127     // create sensors from config
128     sensors = createSensors();
129
130     tryAutoConnect();
131     if (WiFi.status() != WL_CONNECTED) {
132         portal->start();
133     } else {
134         Serial.print("IP: "); Serial.println(WiFi.localIP());
135
136         // Print saved authentication token if it exists
137         String token = storage.getToken();
138         if (token.length() > 0) {
139             Serial.print("Auth token: ");
140             Serial.println(token);
141         } else {
142             Serial.println("No auth token saved");
143         }
144     }
145
146     // Perform one-time provisioning (auth/mqtt credentials/
147         connect)
148     oneTimeProvisioning();
149
150     // Link MQTT client to DataSender for MQTT support at runtime
```

```
147     if (mqttEnabled) {
148         sender->setMqttClient(mqttClient);
149     }
150
151     //print current mqtt credentials
152     MqttCredentials creds;
153     if (storage.getMqttCredentials(creds)) {
154         // Serial.println("Saved MQTT credentials:");
155         // Serial.print("  Server: "); Serial.println(creds.
156             server);
157         // Serial.print("  Username: "); Serial.println(creds.
158             username);
159         // Serial.print("  Password: "); Serial.println(creds.
160             password);
161     } else {
162         // Serial.println("No saved MQTT credentials");
163     }
164     // storage.clearMqttCredentials(); // TESTING: always clear
165     MQTT creds on boot
166 }
167
168 // One-time provisioning: perform auth, fetch MQTT credentials
169 and attempt initial MQTT connect
170 static void oneTimeProvisioning() {
171     if (WiFi.status() != WL_CONNECTED) return;
172
173     // Ensure we have a JWT token (try once synchronously)
174     String token = storage.getToken();
175     if (token.length() == 0) {
176         Serial.println("No token saved, attempting immediate
177             authentication...");
178         auth->tryAuthenticateOnce();
179         token = storage.getToken();
180     }
181 }
```

```
174         if (token.length() > 0) Serial.println("Authentication
           succeeded and token was saved");
175     }
176
177     if (!mqttEnabled) return;
178
179     // If we don't have MQTT credentials but we do have a token,
       fetch them once
180     if (!auth→hasMqttCredentials() && token.length() > 0) {
181         Serial.println("Fetching MQTT credentials (one-time)");
182         if (auth→fetchMqttCredentials()) {
183             Serial.println("MQTT credentials fetched and stored")
               ;
184         } else {
185             Serial.println("Failed to fetch MQTT credentials in
               setup");
186         }
187     }
188
189     // Try a single MQTT connect attempt if credentials are
       available
190     if (auth→hasMqttCredentials()) {
191         Serial.println("Attempting initial MQTT connect from
               setup");
192         if (mqttClient→connect()) {
193             Serial.println("Initial MQTT connect succeeded");
194         } else {
195             Serial.println("Initial MQTT connect failed (will
               retry in loop)");
196         }
197     }
198 }
199
200 // New helper: read sensors and send measurements
```

```
201 static bool sendMeasurements() {
202     // Read sensors into a vector (no manual malloc/free)
203     std::vector<SensorReading> readings;
204     readings.reserve(sensors.size());
205
206     for (size_t i = 0; i < sensors.size(); ++i) {
207         float value;
208         if (sensors[i]→read(value)) {
209             float rounded = roundf(value * 100.0f) / 100.0f; // 2
210                             decimal places
211             readings.push_back({ sensors[i]→uuid(), rounded });
212             Serial.print("Sensor ");
213             Serial.print(sensors[i]→uuid());
214             Serial.print(" = ");
215             Serial.println(rounded);
216         } else {
217             Serial.print("Failed to read from sensor ");
218             Serial.println(sensors[i]→uuid());
219         }
220     }
221
222     if (readings.empty()) {
223         return false;
224     }
225
226     bool ok = sender→sendReadings(readings.data(), readings.size
227                                     ());
228     Serial.print("Data send result: "); Serial.println(ok ? "OK"
229                                                         : "FAILED");
230     return ok;
231 }
```

```
230 void loop()
231 {
```

```
232     portal->handle();
233
234     if (WiFi.status() != WL_CONNECTED) {
235         // Not connected: skip auth.loop() and sendMeasurements()
236         //Serial.println("WiFi not connected, skipping auth and
                send");
237         return;
238     }
239
240
241     // Let auth manager handle periodic auth attempts when needed
242     auth->loop();
243
244     // MQTT runtime processing disabled; initial connect happens
        in setup() only.
245
246     // Attempt send immediately when connected; device will deep-
        sleep on success.
247     // Rate-limit attempts when sends fail to avoid hammering the
        server.
248     unsigned long now = millis();
249     if (now - lastSendAttempt < SEND_RETRY_BACKOFF_MS) {
250         // Wait before trying again
251         return;
252     }
253
254     // Record this attempt time; on failure this enforces the
        backoff interval.
255     lastSendAttempt = now;
256
257     // Call extracted function
258     bool sent = sendMeasurements();
259     if (sent) {
```



```
260      // reset lastSendAttempt to avoid delaying next cycle
      // after wake
261      lastSendAttempt = 0;
262      // Enter deep sleep for the configured interval (
      // milliseconds → microseconds)
263      uint64_t sleep_us = (uint64_t)readIntervalMs * 1000ULL;
264      Serial.print("Measurements sent, entering deep sleep for
      ms: ");
265      Serial.println(readIntervalMs);
266
267      // Turn off WiFi cleanly to speed shutdown
268      if (mqttEnabled) {
269          mqttClient->disconnect();
270      }
271      WiFi.disconnect(true);
272      WiFi.mode(WIFI_OFF);
273      delay(50);
274
275      // Enable wakeup from button (LOW = pressed)
276      #if defined(CONFIG_IDF_TARGET_ESP32)
277          esp_sleep_enable_ext0_wakeup((gpio_num_t)BUTTON_PIN, 0);
278      #elif defined(CONFIG_IDF_TARGET_ESP32C6)
279          // On ESP32-C6, EXT1 wakeup only supports RTC GPIOs (0-7)
280          .
281          // For other pins (like GPIO 14), we use the GPIO deep
      // sleep wakeup.
282          esp_deep_sleep_enable_gpio_wakeup(1ULL << BUTTON_PIN,
          ESP_GPIO_WAKEUP_GPIO_LOW);
283      #else
284          // Other variants (S2, S3, C3) typically use ext1 for
      // GPIO wakeup
285          esp_sleep_enable_ext1_wakeup(1ULL << BUTTON_PIN,
          ESP_EXT1_WAKEUP_ANY_LOW);
286      #endif
```

```
286         // Also enable timer wakeup
287         esp_sleep_enable_timer_wakeup(sleep_us);
288         esp_deep_sleep_start();
289     }
290 }
```

Listing 103: Κύριο σημείο εισόδου WiFi κόμβου (main.cpp)

```
1  #pragma once
2  #include <DNSServer.h>
3  #include <WebServer.h>
4  #include "storage.h"
5  #include "sensor.h"
6  #include <Arduino.h>
7  #include <vector>
8
9  class WifiPortal {
10 public:
11     WifiPortal(Storage &storage, const char* apSsid = "ESP_Config",
12               const char* apPass = "",
13               const char* deviceId = "", const char*
14                 deviceSecret = "", const SensorConfig*
15                 sensorConfigs = nullptr, size_t sensorCount = 0,
16               const char* baseUrl = "", bool mqttEnabled = true,
17               unsigned long readIntervalMs = 120000);
18
19     void start();
20     void stop();
21     void handle();
22     bool isRunning() const { return running; }
23
24 private:
25     Storage &storage;
26     DNSServer dnsServer;
27     WebServer webServer;
28     const char* ssid;
```

```
24  const char* pass;  
25  IPAddress apIP;  
26  bool running;  
27  String indexPage;  
28  std::vector<String> availableNetworks;  
29  const char* deviceUuid;  
30  const char* deviceSecret;  
31  const SensorConfig* sensorConfigs;  
32  size_t sensorCount;  
33  const char* baseUrl;  
34  bool mqttEnabled;  
35  unsigned long readIntervalMs;  
36  
37  void scanNetworks();  
38  String generateHtmlPage();  
39  void onRoot();  
40  void onSave();  
41  };
```

Listing 104: Διεπαφή Captive Portal (wifi_portal.h)

```
1  #include "wifi_portal.h"  
2  #include <WiFi.h>  
3  
4  WifiPortal::WifiPortal(Storage &storage, const char* apSsid,  
5      const char* apPass,  
6      const char* deviceUuid, const char*  
7          deviceSecret, const SensorConfig*  
8          sensorConfigs, size_t sensorCount,  
9      const char* baseUrl, bool mqttEnabled,  
10     unsigned long readIntervalMs)  
11  : storage(storage), webServer(80), ssid(apSsid), pass(apPass),  
12     apIP(192,168,4,1), running(false),  
13     deviceUuid(deviceUuid), deviceSecret(deviceSecret),  
14     sensorConfigs(sensorConfigs), sensorCount(sensorCount),
```

```
9     baseUrl(baseUrl), mqttEnabled(mqttEnabled), readIntervalMs(  
10         readIntervalMs)  
11 {  
12     // indexPage will be generated dynamically in generateHtmlPage  
13     ()  
14 }  
15  
16 void WifiPortal::onRoot() {  
17     String html = generateHtmlPage();  
18     webServer.send(200, "text/html", html);  
19 }  
20  
21 void WifiPortal::scanNetworks() {  
22     availableNetworks.clear();  
23     Serial.println("Scanning for available WiFi networks...");  
24  
25     // Perform WiFi scan (set to async=true to non-blocking)  
26     int numNetworks = WiFi.scanNetworks(false, false);  
27  
28     if (numNetworks == 0) {  
29         Serial.println("No networks found");  
30         return;  
31     }  
32  
33     Serial.print("Found ");  
34     Serial.print(numNetworks);  
35     Serial.println(" networks");  
36  
37     // Collect unique network names (remove duplicates)  
38     for (int i = 0; i < numNetworks; ++i) {  
39         String networkName = WiFi.SSID(i);  
40  
41         // Skip empty SSIDs  
42         if (networkName.length() == 0) continue;
```

```
41
42     // Check if we already have this network in our list
43     bool isDuplicate = false;
44     for (const auto& existing : availableNetworks) {
45         if (existing == networkName) {
46             isDuplicate = true;
47             break;
48         }
49     }
50
51     if (!isDuplicate) {
52         availableNetworks.push_back(networkName);
53         Serial.print("  - ");
54         Serial.print(networkName);
55         Serial.print(" (RSSI: ");
56         Serial.print(WiFi.RSSI(i));
57         Serial.println(")");
58     }
59 }
60
61 WiFi.scanDelete(); // Free memory used by scan
62 }
63
64 String WifiPortal::generateHtmlPage() {
65     String html = R"rawliteral(
66     <!DOCTYPE html>
67     <html>
68     <head>
69         <meta name="viewport" content="width=device-width, initial-
70             scale=1">
71     <style>
72         body { font-family: Arial, sans-serif; margin: 20px;
73             background-color: #f5f5f5; }
74         h3 { color: #333; }
```

```
73     .device-info { background-color: #e8f4f8; padding: 15px;
       border-radius: 4px; margin-bottom: 20px; }
74     .device-info h4 { margin-top: 0; color: #1976d2; }
75     .device-uuid { font-size: 12px; color: #666; word-break:
       break-all; margin: 5px 0; }
76     .sensors-list { margin-left: 15px; font-size: 13px; }
77     .sensor-item { margin: 5px 0; color: #555; }
78     form { margin-top: 20px; background-color: white; padding:
       20px; border-radius: 4px; box-shadow: 0 2px 4px rgba
       (0,0,0,0.1); }
79     label { display: block; margin-bottom: 5px; font-weight:
       bold; }
80     select, input[type="password"] { width: 100%; max-width:
       300px; padding: 8px; margin-bottom: 15px; border: 1px
       solid #ccc; border-radius: 4px; box-sizing: border-box;
       }
81     input[type="submit"] { padding: 10px 20px; background-color
       : #4CAF50; color: white; border: none; border-radius: 4
       px; cursor: pointer; font-weight: bold; }
82     input[type="submit"]:hover { background-color: #45a049; }
83     .info { color: #666; font-size: 14px; margin-bottom: 10px;
       }
84     </style>
85     </head>
86     <body>
87         <div class="device-info">
88             <h4>Device Information</h4>
89             <div>
90                 <strong>Device UUID:</strong>
91                 <div class="device-uuid">)<rawliteral">
92
93     html += deviceUuid ? deviceUuid : "Not configured";
94
95     html += R"<rawliteral(</div>
```

```
96         </div>
97
98         <div style="margin-top: 10px;">
99             <strong>Device Secret:</strong>
100             <div class="device-uuid">)rawliteral";
101
102     html += deviceSecret ? deviceSecret : "Not configured";
103
104     html += R"rawliteral(</div>
105         </div>
106
107         <div style="margin-top: 15px;">
108             <strong>Sensors:</strong>
109             <div class="sensors-list">)rawliteral";
110
111     // Add sensor information
112     if (sensorConfigs && sensorCount > 0) {
113         for (size_t i = 0; i < sensorCount; ++i) {
114             html += "<div class=\"sensor-item\">";
115             html += sensorConfigs[i].displayName;
116             html += " (UUID: ";
117             html += sensorConfigs[i].uuid;
118             html += ")</div>";
119         }
120     } else {
121         html += "<div class=\"sensor-item\">No sensors configured</div>";
122     }
123
124     html += R"rawliteral(
125         </div>
126     </div>
127 </div>
128
```

```
129     <h3>Configure WiFi</h3>
130     <p class="info">Select your WiFi network and enter the
        password</p>
131     <form action="/save" method="POST">
132         <label for="ssid">WiFi Network:</label>
133         <select name="ssid" id="ssid" required>
134             <option value="">-- Select a network --</option>
135     </select>
136
137     // Add available networks to the dropdown
138     for (const auto& network : availableNetworks) {
139         html += "        <option value=\"";
140         html += network;
141         html += "\">";
142         html += network;
143         html += "</option>\n";
144     }
145
146     html += R"rawliteral(
147         </select>
148         <label for="pass">Password:</label>
149         <input type="password" name="pass" id="pass" placeholder="
        Enter WiFi password">
150
151         <h3 style="margin-top: 30px; color: #333;">Device
        Configuration</h3>
152         <p class="info">Configure device settings (optional)</p>
153
154         <label for="base_url">Server URL:</label>
155         <input type="text" name="base_url" id="base_url"
        placeholder="https://example.com" value="">rawliteral";
156
157     // Pre-populate base URL
158     html += baseUrl ? baseUrl : "";
```



```
159
160     html += R"rawliteral(">
161
162         <label for="read_interval_minutes">Read Interval (minutes)
163             :</label>
164
165         <input type="number" name="read_interval_minutes" id="
166             read_interval_minutes" min="1" value=")>rawliteral";
167
168     // Pre-populate read interval in minutes
169     html += String(readIntervalMs / 60000);
170
171     html += R"rawliteral(">
172
173     <label>
174         <input type="checkbox" name="mqtt_enabled" id="
175             mqtt_enabled" value="on">rawliteral";
176
177     // Pre-populate MQTT enabled checkbox
178     if (mqttEnabled) {
179         html += " checked";
180     }
181
182     html += R"rawliteral(>
183         Enable MQTT protocol (uncheck to use HTTP only)
184     </label>
185
186     <br><br>
187     <input type="submit" value="Save & Connect">
188     </form>
189     </body>
190     </html>
191     )rawliteral";
192
193     return html;
```

```
190 }  
191  
192 void WifiPortal::onSave() {  
193     String ssidArg = webServer.arg("ssid");  
194     String passArg = webServer.arg("pass");  
195  
196     // Extract device config parameters  
197     String baseUrlArg = webServer.arg("base_url");  
198     String readIntervalArg = webServer.arg("read_interval_minutes")  
199     ;  
200  
201     bool mqttEnabledArg = webServer.hasArg("mqtt_enabled");  
202  
203     if (ssidArg.length() > 0) {  
204         // Save WiFi credentials  
205         storage.setWifiCreds(ssidArg, passArg);  
206  
207         // Build device configuration struct with new values  
208         DeviceConfig newConfig;  
209         newConfig.baseUrl = (baseUrlArg.length() > 0) ? baseUrlArg :  
210             storage.getBaseUrl();  
211  
212         if (readIntervalArg.length() > 0) {  
213             unsigned long intervalMinutes = readIntervalArg.toInt();  
214             newConfig.readIntervalMs = intervalMinutes * 60 * 1000;  
215         } else {  
216             newConfig.readIntervalMs = storage.getReadIntervalMs();  
217         }  
218  
219         newConfig.mqttEnabled = mqttEnabledArg;  
220  
221         // Save all config in one atomic operation  
222         storage.saveConfig(newConfig);  
223  
224         webServer.send(200, "text/html", "<h3>Saved. Rebooting...</h3>");  
225     }  
226 }
```

```
        >");  
222     delay(5000);  
223     ESP.restart();  
224 } else {  
225     webServer.send(400, "text/plain", "SSID required");  
226 }  
227 }  
228  
229 void WifiPortal::start() {  
230     WiFi.mode(WIFI_AP_STA);  
231     WiFi.softAPConfig(apIP, apIP, IPAddress(255,255,255,0));  
232     WiFi.softAP(ssid, pass);  
233  
234     // Scan for available networks before starting the server  
235     scanNetworks();  
236  
237     dnsServer.start(53, "*", apIP);  
238     // Main portal page and save endpoint  
239     webServer.on("/", HTTP_GET, std::bind(&WifiPortal::onRoot, this  
240         ));  
241     webServer.on("/save", HTTP_POST, std::bind(&WifiPortal::onSave,  
242         this));  
243  
244     // Common captive-portal checks used by Android / iOS / Windows  
245     // / macOS  
246     // Android uses /generate_204 (expecting a 204 No Content)  
247     webServer.on("/generate_204", HTTP_GET, [this]() {  
248         webServer.send(204, "text/plain", "");  
249     });  
250     // Some platforms query for a small file or redirect; reply  
251     // with the portal page  
252     webServer.on("/hotspot-detect.html", HTTP_GET, std::bind(&  
253         WifiPortal::onRoot, this));  
254     webServer.on("/ncsi.txt", HTTP_GET, std::bind(&WifiPortal::
```

```
        onRoot, this));  
250     webServer.on("/connectivity-check.html", HTTP_GET, std::bind(&  
        WifiPortal::onRoot, this));  
  
251  
252     // Favicon requests (silence them)  
253     webServer.on("/favicon.ico", HTTP_GET, [this]() {  
254         webServer.send(204, "image/x-icon", "");  
255     });  
256  
257     // Fallback: serve the portal page for any unknown request (  
        prevents the WebServer "not found" log)  
258     webServer.onNotFound(std::bind(&WifiPortal::onRoot, this));  
259     webServer.begin();  
260     running = true;  
261     Serial.println("AP started, captive portal running");  
262 }  
263  
264 void WifiPortal::stop() {  
265     if (!running) return;  
266     WiFi.softAPdisconnect(true);  
267     dnsServer.stop();  
268     webServer.stop();  
269     running = false;  
270     Serial.println("Portal stopped");  
271 }  
272  
273 void WifiPortal::handle() {  
274     if (!running) return;  
275     dnsServer.processNextRequest();  
276     webServer.handleClient();  
277     // If the device becomes connected while portal is running,  
        stop the portal  
278     if (WiFi.status() == WL_CONNECTED) {  
279         Serial.println("WiFi connected while portal running -
```

```
        stopping_portal");  
280     stop();  
281 }  
282 }
```

Listing 105: Υλοποίηση Captive Portal (wifi_portal.cpp)

```
1  #pragma once  
2  #include <Arduino.h>  
3  #include "storage.h"  
4  
5  class AuthManager {  
6  public:  
7      AuthManager(Storage &storage, const char* baseUrl, const char*  
          uuid, const char* secret, unsigned long retryIntervalMs =  
          30000);  
8      void begin();  
9      // Try authenticate once immediately (blocking network call)  
10     bool tryAuthenticateOnce();  
11     // Call from main loop to perform periodic attempts when needed  
12     void loop();  
13     String getSavedToken();  
14  
15     // MQTT credential provisioning  
16     bool fetchMqttCredentials();  
17     bool hasMqttCredentials();  
18  
19 private:  
20     Storage &storage;  
21     const char* baseUrl;  
22     const char* uuid;  
23     const char* secret;  
24     unsigned long retryIntervalMs;  
25     unsigned long lastAttempt;  
26
```

```
27  bool performAuthRequest();  
28  bool performMqttCredentialsRequest();  
29  };
```

Listing 106: Διεπαφή αυθεντικοποίησης (auth.h)

```
1  #include "auth.h"  
2  #include <HTTPClient.h>  
3  #include <WiFi.h>  
4  #include <ArduinoJson.h>  
5  
6  AuthManager::AuthManager(Storage &storage, const char* baseUrl,  
7    const char* uuid, const char* secret, unsigned long  
8    retryIntervalMs)  
9  : storage(storage), baseUrl(baseUrl), uuid(uuid), secret(secret),  
10    retryIntervalMs(retryIntervalMs), lastAttempt(0) {}  
11  
12  
13 void AuthManager::begin() {  
14     // nothing for now  
15 }  
16  
17 String AuthManager::getSavedToken() {  
18     return storage.getToken();  
19 }  
20  
21 bool AuthManager::performAuthRequest() {  
22     if (WiFi.status() != WL_CONNECTED) {  
23         Serial.println("Not connected to WiFi, skipping  
24             authentication");  
25         return false;  
26     }  
27  
28     HTTPClient http;  
29     String url = String(baseUrl) + "/api/v1/device/login";  
30     Serial.print("Authenticating to: "); Serial.println(url);
```

```
26
27 http.begin(url);
28 http.addHeader("Content-Type", "application/json");
29
30 String payload = String("{\"uuid\":\"" + uuid + "\", \"secret
    \":\"" + secret + "\"}");
31 int httpCode = http.POST(payload);
32
33 if (httpCode > 0) {
34     Serial.print("HTTP response code: "); Serial.println(httpCode
        );
35     String resp = http.getString();
36     Serial.print("Response: "); Serial.println(resp);
37
38     // Use ArduinoJson to parse response safely
39     // Allocate a document with a reasonable size for expected
        responses
40     const size_t capacity = 1024;
41     DynamicJsonDocument doc(capacity);
42     DeserializationError err = deserializeJson(doc, resp);
43     if (err) {
44         Serial.print("Failed to parse JSON response: "); Serial.
            println(err.c_str());
45         http.end();
46         return false;
47     }
48
49     // Successful HTTP codes (2xx) - try to extract token
50     if (httpCode >= 200 && httpCode < 300) {
51         if (doc.containsKey("token")) {
52             const char* token = doc["token"];
53             if (token != nullptr) {
54                 storage.setToken(String(token));
55                 http.end();
```

```
56         return true;
57     }
58 }
59 // If no token field, try to print a message if present
60 if (doc.containsKey("message")) {
61     const char* message = doc["message"];
62     if (message) {
63         Serial.print("Server message: "); Serial.println(
            message);
64     }
65 }
66
67 http.end();
68 return false;
69 }
70
71 // Non-2xx responses - print message if available
72 if (doc.containsKey("message")) {
73     const char* message = doc["message"];
74     if (message) {
75         Serial.print("Server message: "); Serial.println(message)
            ;
76     }
77 }
78
79 http.end();
80 return false;
81 } else {
82     Serial.print("Request failed, error: "); Serial.println(http.
        errorToString(httpCode).c_str());
83     http.end();
84     return false;
85 }
86 }
```



```
87
88 bool AuthManager::tryAuthenticateOnce() {
89     lastAttempt = millis();
90     return performAuthRequest();
91 }
92
93 void AuthManager::loop() {
94     if (WiFi.status() != WL_CONNECTED) return;
95     String token = storage.getToken();
96     if (token.length() > 0) return; // already have token
97     unsigned long now = millis();
98     if (now - lastAttempt >= retryIntervalMs) {
99         lastAttempt = now;
100         performAuthRequest();
101     }
102 }
103
104 bool AuthManager::hasMqttCredentials() {
105     return storage.hasMqttCredentials();
106 }
107
108 bool AuthManager::performMqttCredentialsRequest() {
109     if (WiFi.status() != WL_CONNECTED) {
110         Serial.println("WiFi not connected, cannot fetch MQTT
111             credentials");
112         return false;
113     }
114     String token = storage.getToken();
115     if (token.length() == 0) {
116         Serial.println("No JWT token available for MQTT provisioning"
117             );
118         return false;
119     }
120 }
```

```
119
120 HTTPClient http;
121 String url = String(baseUrl) + "/api/v1/device/mqtt-credentials
    ";
122 Serial.print("Fetching MQTT credentials from: "); Serial.
    println(url);
123
124 http.begin(url);
125 http.addHeader("Content-Type", "application/json");
126 http.addHeader("Authorization", String("Bearer ") + token);
127
128 int httpCode = http.GET();
129
130 if (httpCode > 0) {
131     Serial.print("HTTP response code: "); Serial.println(httpCode
        );
132     String response = http.getString();
133     Serial.print("Response: "); Serial.println(response);
134
135     if (httpCode >= 200 && httpCode < 300) {
136         // Parse JSON response
137         const size_t capacity = JSON_OBJECT_SIZE(10) + 512;
138         DynamicJsonDocument doc(capacity);
139         DeserializationError err = deserializeJson(doc, response);
140
141         if (err) {
142             Serial.print("Failed to parse MQTT credentials JSON: ");
143             Serial.println(err.c_str());
144             http.end();
145             return false;
146         }
147
148         // Extract credentials
149         if (!doc.containsKey("mqtt_broker_url") ||
```

```
150         !doc.containsKey("username") ||
151         !doc.containsKey("password")) {
152         Serial.println("Missing MQTT credential fields in
           response");
153         http.end();
154         return false;
155     }
156
157     String server = doc["mqtt_broker_url"].as<String>();
158     String username = doc["username"].as<String>();
159     String password = doc["password"].as<String>();
160
161     if (server.length() == 0 || username.length() == 0 ||
        password.length() == 0) {
162         Serial.println("Empty MQTT credential fields");
163         http.end();
164         return false;
165     }
166
167     // Store credentials
168     storage.setMqttCredentials(server, username, password);
169
170     Serial.println("MQTT credentials obtained successfully:");
171     Serial.print("  Server: "); Serial.println(server);
172     Serial.print("  Username: "); Serial.println(username);
173     Serial.println("  Password: "); Serial.println(password);
174     // Print actual password
175
176     http.end();
177     return true;
178 } else {
179     Serial.println("Failed to fetch MQTT credentials - non-2xx
           response");
180     http.end();
```

```
180     return false;
181 }
182 } else {
183     Serial.print("HTTP request failed: ");
184     Serial.println(http.errorToString(httpCode).c_str());
185     http.end();
186     return false;
187 }
188 }
189
190 bool AuthManager::fetchMqttCredentials() {
191     return performMqttCredentialsRequest();
192 }
```

Listing 107: Υλοποίηση αυθεντικοποίησης JWT (auth.cpp)

```
1 #pragma once
2
3 #include <Arduino.h>
4 #include "storage.h"
5
6 // Forward declaration
7 class MqttClient;
8
9 struct SensorReading {
10     const char* uuid;
11     float value;
12 };
13
14 class DataSender {
15 public:
16     DataSender(Storage &storage, const char* baseUrl);
17
18     // Set MQTT client for MQTT support (optional)
19     void setMqttClient(MqttClient* client);
```

```
20
21 // Send an array of SensorReading { uuid, value }
22 bool sendReadings(const SensorReading* readings, size_t count
    );
23
24 // Send values paired with explicit UUIDs
25 bool sendValuesWithUuids(const char* uuids[], const float*
    values, size_t count);
26
27 // Backwards-compatible helper that uses SENSOR_UUIDS from
    config.h
28 bool sendValues(const float* values, size_t count);
29
30 private:
31     Storage &storage;
32     const char* baseUrl;
33     MqttClient* mqttClient; // Optional MQTT client
34
35     String buildUrl() const;
36     bool postJson(const String &json, const String &token);
37 };
```

Listing 108: Διεπαφή αποστολής δεδομένων (data_sender.h)

```
1 #include "data_sender.h"
2 #include <HTTPClient.h>
3 #include <WiFi.h>
4 #include <ArduinoJson.h>
5 #include "config.h"
6 #include "mqtt_client.h"
7
8 DataSender::DataSender(Storage &storage, const char* baseUrl)
9 : storage(storage), baseUrl(baseUrl), mqttClient(nullptr) {}
10
11 void DataSender::setMqttClient(MqttClient* client) {
```

```
12     mqttClient = client;
13     Serial.println("MQTT client linked to DataSender");
14 }
15
16 String DataSender::buildUrl() const {
17     return String(baseUrl) + "/api/v1/device/data";
18 }
19
20 bool DataSender::postJson(const String &json, const String &token
    ) {
21     if (WiFi.status() != WL_CONNECTED) {
22         Serial.println("WiFi not connected, cannot send data");
23         return false;
24     }
25
26     HTTPClient http;
27     String url = buildUrl();
28     Serial.print("Posting data to: "); Serial.println(url);
29
30     http.begin(url);
31     http.addHeader("Content-Type", "application/json");
32     http.addHeader("Authorization", String("Bearer ") + token);
33
34     int code = http.POST(json);
35     Serial.print("HTTP code: "); Serial.println(code);
36     String resp = http.getString();
37     Serial.print("Response: "); Serial.println(resp);
38
39     http.end();
40     return (code >= 200 && code < 300);
41 }
42
43 bool DataSender::sendReadings(const SensorReading* readings,
    size_t count) {
```

```
44     if (count == 0 || readings == nullptr) return false;
45
46     // Build JSON payload once (used by MQTT or HTTP) - same
47     // shape as previous HTTP payload
48     const size_t capacity = JSON_ARRAY_SIZE(count) + count *
49                             JSON_OBJECT_SIZE(2) + 512;
50     DynamicJsonDocument doc(capacity);
51     JsonArray sensorsArr = doc.createNestedArray("sensors");
52     // Round values to a fixed number of decimal places before
53     // serializing
54     constexpr int VALUE_PRECISION = 2; // number of decimal
55     // places
56     const float factor = powf(10.0f, VALUE_PRECISION);
57     for (size_t i = 0; i < count; ++i) {
58         JsonObject obj = sensorsArr.createNestedObject();
59         obj["uuid"] = readings[i].uuid;
60         float rounded = roundf(readings[i].value * factor) /
61                             factor;
62         obj["value"] = rounded;
63     }
64
65     String payload;
66     serializeJson(doc, payload);
67
68     // Try MQTT first if enabled and credentials are available (
69     // runtime check)
70     if (MQTT_ENABLED && mqttClient != nullptr && storage.
71         hasMqttCredentials()) {
72         Serial.println("Attempting to send data via MQTT...");
73
74         // Attempt a local connect only for this publish if not
75         // already connected
76         bool connectedNow = false;
77         bool alreadyConnected = mqttClient->isConnected();
```

```
70     if (!alreadyConnected) {
71         connectedNow = mqttClient->connect();
72     }
73
74     bool success = false;
75     if (alreadyConnected || connectedNow) {
76         // give the MQTT/TCP stack a moment to settle after
77         // connect
78         if (connectedNow) mqttClient->process();
79
80         // publish payload created above
81         success = mqttClient->publishSensorDataPayload(payload
82             .c_str());
83         if (success) {
84             Serial.println("Data sent successfully via MQTT");
85             ;
86             if (connectedNow) mqttClient->disconnect();
87             return true;
88         } else {
89             Serial.println("MQTT publish failed, falling back
90                 to HTTP");
91         }
92     } else {
93         Serial.println("MQTT connection failed, falling back
94             to HTTP");
95     }
96
97     if (connectedNow && !success) mqttClient->disconnect();
98
99     // Fallback to HTTP (or use HTTP if MQTT is not enabled/
100     // configured)
101     Serial.println("Sending data via HTTP...");
```



```
98     String token = storage.getToken();
99     if (token.length() == 0) {
100         Serial.println("No auth token available");
101         return false;
102     }
103
104     bool result = postJson(payload, token);
105
106     if (result) {
107         Serial.println("Data sent successfully via HTTP");
108     } else {
109         Serial.println("HTTP send failed");
110     }
111
112     return result;
113 }
114
115 bool DataSender::sendValuesWithUuids(const char* uuids[], const
float* values, size_t count) {
116     if (!uuids || !values || count == 0) return false;
117     // build temporary readings on stack
118     SensorReading tmp[count];
119     for (size_t i = 0; i < count; ++i) {
120         tmp[i].uuid = uuids[i];
121         tmp[i].value = values[i];
122     }
123     return sendReadings(tmp, count);
124 }
125
126 bool DataSender::sendValues(const float* values, size_t count) {
127     if (!values || count == 0) return false;
128     if (count > SENSOR_CONFIG_COUNT) {
129         Serial.println("sendValues: count exceeds
SENSOR_CONFIG_COUNT");
```

```
130         return false;
131     }
132
133     // Build UUID array from SENSOR_CONFIGS (backwards-compatible
134     behavior)
135     const char** uuids = (const char**)malloc(sizeof(const char*)
136         * count);
137     if (!uuids) return false;
138     for (size_t i = 0; i < count; ++i) {
139         uuids[i] = SENSOR_CONFIGS[i].uuid;
140     }
141     bool result = sendValuesWithUuids(uuids, values, count);
142     free(uuids);
143     return result;
144 }
```

Listing 109: Υλοποίηση MQTT-first με HTTP fallback (data_sender.cpp)

```
1  #pragma once
2  #include <Arduino.h>
3  #include <WiFi.h>
4  #include <PubSubClient.h>
5  #include <WiFiClientSecure.h>
6  #include "storage.h"
7  #include "data_sender.h" // For SensorReading struct
8
9  class MqttClient {
10 public:
11     MqttClient(Storage &storage, const char* deviceUuid);
12
13     // Connection management
14     bool connect();
15     bool isConnected();
16     void disconnect();
17     // Process pending MQTT events once (call after connect to
```

```
        stabilize)

18     void process();
19
20     // Publish sensor data payload (JSON string built by
        DataSender)
21     bool publishSensorDataPayload(const char* payload);
22
23     // Publish device status
24     bool publishStatus(const char* status);
25
26 private:
27     Storage &storage;
28     WiFiClient wifiClient;
29     WiFiClientSecure secureClient;
30     PubSubClient mqttClient;
31
32     const char* deviceUuid;
33     MqttCredentials credentials;
34     bool credentialsLoaded;
35
36     String buildTopic(const char* topicTemplate) const;
37     bool loadCredentials();
38
39     // Callback for incoming messages
40     static void mqttCallback(char* topic, byte* payload, unsigned
        int length);
41 };
```

Listing 110: Διεπαφή πελάτη MQTT (mqtt_client.h)

```
1 #include "mqtt_client.h"
2 #include <ArduinoJson.h>
3 #include "config.h"
4
5 MqttClient::MqttClient(Storage &storage, const char* deviceUuid)
```

```
6 : storage(storage),
7   deviceId(deviceId),
8   credentialsLoaded(false) {
9
10  // Initialize MQTT client with appropriate WiFi client
11  #ifdef AGRONOS_MQTT_USE_TLS
12  if (AGRONOS_MQTT_USE_TLS) {
13      secureClient.setInsecure(); // For testing; use proper
14                                  certificates in production
15      mqttClient.setClient(secureClient);
16  } else {
17      mqttClient.setClient(wifiClient);
18  }
19  #endif
20
21  // No callback/subscriptions needed for this device (publish-
22  // only)
23  mqttClient.setKeepAlive(AGRONOS_MQTT_KEEPAIVE);
24 }
25
26 bool MqttClient::loadCredentials() {
27     if (credentialsLoaded && credentials.isValid) {
28         return true;
29     }
30
31     credentialsLoaded = storage.getMqttCredentials(credentials);
32     return credentialsLoaded && credentials.isValid;
33 }
34
35 String MqttClient::buildTopic(const char* topicTemplate) const {
36     char topic[128];
37     snprintf(topic, sizeof(topic), topicTemplate, deviceId);
```

```
38     return String(topic);
39 }
40
41 bool MqttClient::connect() {
42     if (!loadCredentials()) {
43         Serial.println("Cannot connect to MQTT: No credentials
44             available");
45         return false;
46     }
47
48     if (mqttClient.connected()) {
49         return true;
50     }
51
52     // Set MQTT server
53     mqttClient.setServer(credentials.server.c_str(),
54         AGRONOS_MQTT_PORT);
55
56     // Generate client ID
57     String clientId = String("agronos-") + deviceUuid;
58
59     Serial.print("Connecting to MQTT broker: ");
60     Serial.print(credentials.server);
61     Serial.print(":");
62     Serial.println(AGRONOS_MQTT_PORT);
63
64     // Attempt connection
65     bool connected = mqttClient.connect(
66         clientId.c_str(),
67         credentials.username.c_str(),
68         credentials.password.c_str(),
69         nullptr, // willTopic
70         0,       // willQos
71         false,   // willRetain
```

```
70     nullptr, // willMessage
71     AGRONOS_MQTT_CLEAN_SESSION
72 );
73
74 if (connected) {
75     Serial.println("MQTT connected successfully");
76
77     // No subscription needed for publish-only device
78
79     return true;
80 } else {
81     Serial.print("MQTT connection failed, state: ");
82     Serial.println(mqttClient.state());
83     return false;
84 }
85 }
86
87 bool MqttClient::isConnected() {
88     return mqttClient.connected();
89 }
90 void MqttClient::disconnect() {
91     if (mqttClient.connected()) {
92         mqttClient.disconnect();
93         Serial.println("MQTT disconnected");
94     }
95 }
96
97 void MqttClient::process() {
98     // Run a single pass of the underlying client to let the TCP/
99     // MQTT stack settle
100
101     if (mqttClient.connected()) {
102         mqttClient.loop();
103         delay(50);
104     }
```

```
103 }  
104  
105 bool MqttClient::publishSensorDataPayload(const char* payload) {  
106     if (!isConnected()) {  
107         Serial.println("MQTT not connected, cannot publish sensor  
108             data");  
109         return false;  
110     }  
111     if (!payload) {  
112         Serial.println("Empty payload passed to  
113             publishSensorDataPayload");  
114         return false;  
115     }  
116     String topic = buildTopic(AGRONOS_MQTT_TOPIC_DATA);  
117     Serial.print("Publishing to MQTT topic: "); Serial.println(  
118         topic);  
119     Serial.print("Payload: "); Serial.println(payload);  
120     bool published = mqttClient.publish(topic.c_str(), payload,  
121         false);  
122     if (published) {  
123         Serial.println("MQTT publish successful");  
124     } else {  
125         Serial.println("MQTT publish failed");  
126     }  
127  
128     return published;  
129 }  
130  
131 bool MqttClient::publishStatus(const char* status) {  
132     if (!isConnected()) {
```

```
133         return false;
134     }
135
136     String topic = buildTopic(AGRONOS_MQTT_TOPIC_STATUS);
137     return mqttClient.publish(topic.c_str(), status, false);
138 }
139
140 // mqttCallback removed: device is publish-only and does not
    subscribe to topics
```

Listing 111: Υλοποίηση πελάτη MQTT (mqtt_client.cpp)

Β.5 Κόμβος LoRa

```
1 // LoRa-only entry point for TTGO LoRa32 V2.1
2 // Reads sensors, serializes to binary, encrypts with AES-128-CTR
3 // ,
4 // transmits via LoRa radio, then enters deep sleep.
5 // WiFi/MQTT/HTTP are not used - the LoRa Gateway handles
6 // internet forwarding.
7
8 #include <Arduino.h>
9 #include <esp_sleep.h>
10 #include <cmath>
11
12 #include "config.h"
13 #include "storage.h"
14 #include "sensor.h"
15 #include "data_sender.h" // for SensorReading struct
16 #include "lora_payload.h"
17 #include "lora_crypto.h"
18 #include "lora_fcnt.h"
19 #include "lora_radio.h"
```



```
19 // Max sensors - 256-byte LoRa packets: (256 - 8 header) / 6 = 41
    sensors_max
20 static constexpr size_t MAX_PAYLOAD_SIZE = 246;
21
22 Storage storage;
23
24 static void printHex(const char* label, const uint8_t* data,
    size_t len) {
25     Serial.print(label);
26     Serial.print(": ");
27     for (size_t i = 0; i < len; ++i) {
28         if (data[i] < 0x10) Serial.print('0');
29         Serial.print(data[i], HEX);
30     }
31     Serial.println();
32 }
33
34 // Sensor instances created via factory
35 static std::vector<std::unique_ptr<SensorBase>> sensors;
36
37 // Check if button is held for long press to reset all storage
38 static void checkButtonReset() {
39     if (digitalRead(BUTTON_PIN) == LOW) {
40         Serial.println("Button pressed, checking for long press
            ...");
41         unsigned long pressStart = millis();
42
43         while (digitalRead(BUTTON_PIN) == LOW) {
44             if (millis() - pressStart >= BUTTON_LONG_PRESS_MS) {
45                 Serial.println("Long press detected! Wiping all
                    storage data...");
46                 storage.clearAll();
47                 Serial.println("Storage cleared. Restarting
                    device...");
```

```
48         delay(1000);
49         ESP.restart();
50     }
51     delay(100);
52 }
53 Serial.println("Button released before threshold");
54 }
55 }
56
57 void setup() {
58     Serial.begin(115200);
59     delay(2000);
60     Serial.println("=====");
61     Serial.println(" Agronos LoRa Node");
62     Serial.println("=====");
63
64     // Button setup & factory reset check
65     pinMode(BUTTON_PIN, INPUT_PULLUP);
66     checkButtonReset();
67
68     // LED indicator
69     pinMode(LORA_LED_PIN, OUTPUT);
70     digitalWrite(LORA_LED_PIN, HIGH); // LED on during active
71                                     cycle
72
73     // 1. Initialize frame counter (RTC or NVS cold-boot recovery
74     )
75     fcntInit(storage);
76
77     // 2. Initialize LoRa radio
78     if (!loraRadioInit()) {
79         Serial.println("FATAL: LoRa radio init failed. Sleeping
80 ...");
81     }
```

```
78         esp_sleep_enable_timer_wakeup((uint64_t)
           SENSORS_READ_INTERVAL_MS * 1000ULL);
79         esp_deep_sleep_start();
80     }
81
82     // 3. Create sensors from config (reuses factory pattern)
83     sensors = createSensors();
84     Serial.printf("Created %u sensors\n", sensors.size());
85
86     // 4. Read all sensors
87     std::vector<SensorReading> readings;
88     readings.reserve(sensors.size());
89
90     for (size_t i = 0; i < sensors.size(); ++i) {
91         float value;
92         if (sensors[i]→read(value)) {
93             float rounded = roundf(value * 100.0f) / 100.0f;
94             readings.push_back({ sensors[i]→uuid(), rounded });
95             Serial.printf("  %s = %.2f\n", sensors[i]→uuid(),
                           rounded);
96         } else {
97             Serial.printf("  %s = FAILED\n", sensors[i]→uuid());
98         }
99     }
100
101     if (readings.empty()) {
102         Serial.println("No sensor readings. Sleeping...");
103         loraRadioSleep();
104         digitalWrite(LORA_LED_PIN, LOW);
105         esp_sleep_enable_timer_wakeup((uint64_t)
           SENSORS_READ_INTERVAL_MS * 1000ULL);
106         esp_deep_sleep_start();
107     }
108
```

```
109 // 5. Serialize to binary payload (N × 6 bytes)
110 uint8_t plaintext[MAX_PAYLOAD_SIZE];
111 size_t payloadLen = serializeReadings(readings.data(),
112                                     readings.size(),
113                                     plaintext, sizeof(
114                                         plaintext));
115
116 if (payloadLen == 0) {
117     Serial.println("Serialization failed. Sleeping...");
118     loraRadioSleep();
119     digitalWrite(LORA_LED_PIN, LOW);
120     esp_sleep_enable_timer_wakeup((uint64_t)
121                                  SENSORS_READ_INTERVAL_MS * 1000ULL);
122     esp_deep_sleep_start();
123 }
124
125 Serial.printf("Payload: %u bytes (%u sensors)\n", payloadLen,
126               readings.size());
127
128 printHex("Raw payload", plaintext, payloadLen);
129
130
131 // 6. Get next frame counter
132 uint32_t fcnt = fcntNext(storage);
133
134
135 // 7. Build nonce and encrypt
136 uint8_t nonce[16];
137 buildNonce(DEFAULT_UUID, fcnt, nonce);
138
139
140 uint8_t ciphertext[MAX_PAYLOAD_SIZE];
141 if (!encryptPayload(plaintext, payloadLen, LORA_AES_KEY,
142                    nonce, ciphertext)) {
143     Serial.println("Encryption failed. Sleeping...");
144     loraRadioSleep();
145     digitalWrite(LORA_LED_PIN, LOW);
146     esp_sleep_enable_timer_wakeup((uint64_t)
147                                  SENSORS_READ_INTERVAL_MS * 1000ULL);
148     esp_deep_sleep_start();
149 }
```

```
137     }
138
139     printHex("Encrypted payload", ciphertext, payloadLen);
140
141     // Print all transited data for backend test purposes
142     Serial.printf("Transmit meta: uuid=%s, fcnt=%u, payloadLen=%u\n",
143                  DEFAULT_UUID, fcnt, payloadLen);
144
145     printHex("Transmit ciphertext", ciphertext, payloadLen);
146
147     // 8. Transmit via LoRa
148     if (loraTransmit(DEFAULT_UUID, fcnt, ciphertext, payloadLen))
149     {
150         Serial.printf("TX success: uuid=%s fcnt=%u\n",
151                      DEFAULT_UUID, fcnt);
152     } else {
153         Serial.println("TX failed");
154     }
155
156     // 9. Prepare for deep sleep
157     loraRadioSleep();
158     digitalWrite(LORA_LED_PIN, LOW);
159
160     uint64_t sleepUs = (uint64_t)SENSORS_READ_INTERVAL_MS * 1000
161                       ULL;
162     Serial.printf("Entering deep sleep for %lu ms\n",
163                  SENSORS_READ_INTERVAL_MS);
164
165     esp_sleep_enable_timer_wakeup(sleepUs);
166     esp_deep_sleep_start();
167 }
168
169 void loop() {
170     // Never reached - device always deep-sleeps after setup()
171 }
```

Listing 112: Κύριο σημείο εισόδου LoRa κόμβου (main_lora.cpp)

```
1 #pragma once
2
3 #include <stdint.h>
4 #include <stddef.h>
5
6 struct SensorReading; // forward declaration (defined in
   data_sender.h)
7
8 // Serialize sensor readings into binary format: N × 6 bytes
9 // Per sensor: [4 bytes UUID prefix (ASCII)] [2 bytes value as
   int16_t LE (value × 100)]
10 // Returns number of bytes written, or 0 on error.
11 size_t serializeReadings(const SensorReading* readings, size_t
   count,
12                          uint8_t* outBuf, size_t bufSize);
```

Listing 113: Διεπαφή δυαδικής σειριοποίησης (lora_payload.h)

```
1 #include "lora_payload.h"
2 #include "data_sender.h"
3 #include <cstring>
4
5 // Binary payload format per sensor (6 bytes):
6 // Bytes 0-3: First 4 ASCII characters of the sensor UUID
7 // Bytes 4-5: int16_t Little-Endian of (value * 100)
8 //
9 // Total payload size = count * 6 bytes.
10 // Must match Laravel LoRaCryptoService::deserialize() on the
   backend.
11
12 static constexpr size_t CHUNK_SIZE = 6;
13
```

```
14 size_t serializeReadings(const SensorReading* readings, size_t
    count,
15                          uint8_t* outBuf, size_t bufSize) {
16     size_t needed = count * CHUNK_SIZE;
17     if (!readings || count == 0 || !outBuf || bufSize < needed) {
18         return 0;
19     }
20
21     for (size_t i = 0; i < count; ++i) {
22         uint8_t* chunk = outBuf + (i * CHUNK_SIZE);
23
24         // Bytes 0-3: first 4 chars of UUID, zero-padded if
            shorter
25         size_t uuidLen = strlen(readings[i].uuid);
26         for (size_t j = 0; j < 4; ++j) {
27             chunk[j] = (j < uuidLen) ? static_cast<uint8_t>(
                readings[i].uuid[j]) : 0;
28         }
29
30         // Bytes 4-5: value * 100 as int16_t in Little-Endian
            // ESP32 is natively LE so memcpy works directly
31         int16_t scaled = static_cast<int16_t>(readings[i].value *
            100.0f);
32         memcpy(chunk + 4, &scaled, sizeof(int16_t));
33     }
34
35     return needed;
36 }
37 }
```

Listing 114: Υλοποίηση δυαδικής σειριοποίησης (lora_payload.cpp)

```
1 #pragma once
2
3 #include <stdint.h>
4 #include <stddef.h>
```

```
5
6 // Compute a 4-byte hash of the UUID string for nonce
   construction.
7 // Uses standard CRC32 (same polynomial as PHP's crc32() function
   ).
8 uint32_t uuidHash(const char* uuid);
9
10 // Build a deterministic 16-byte nonce for AES-128-CTR.
11 // Layout: [4B CRC32(UUID) LE] [4B fcnt LE] [8B zeros]
12 void buildNonce(const char* uuid, uint32_t fcnt, uint8_t nonce
   [16]);
13
14 // Encrypt (or decrypt) a payload using AES-128-CTR with hardware
   acceleration.
15 // Ciphertext will have exactly the same length as plaintext (no
   padding).
16 // Returns true on success.
17 bool encryptPayload(const uint8_t* plaintext, size_t len,
18                    const uint8_t key[16], const uint8_t nonce
19                    [16],
                    uint8_t* ciphertext);
```

Listing 115: Διεπαφή κρυπτογράφησης AES-128-CTR (lora_crypto.h)

```
1 #include "lora_crypto.h"
2 #include <cstring>
3 #include <mbedtls/aes.h>
4
5 // Standard CRC32 (polynomial 0xEDB88320, same as PHP crc32() and
   zlib).
6 // Produces identical output to PHP's crc32() for the same input
   string.
7 uint32_t uuidHash(const char* uuid) {
8     uint32_t crc = 0xFFFFFFFF;
9     while (*uuid) {
```



```
10     crc ^= (uint8_t)*uuid++;
11     for (int j = 0; j < 8; ++j) {
12         crc = (crc >> 1) ^ (0xEDB88320 & ~(crc & 1));
13     }
14 }
15 return crc ^ 0xFFFFFFFF;
16 }
17
18 // Build a deterministic 16-byte nonce for AES-128-CTR.
19 // Layout must match Laravel LoRaCryptoService::decrypt():
20 // Bytes 0-3: CRC32(UUID) in Little-Endian
21 // Bytes 4-7: fcnt in Little-Endian
22 // Bytes 8-15: zeros
23 void buildNonce(const char* uuid, uint32_t fcnt, uint8_t nonce
24 [16]) {
25     memset(nonce, 0, 16);
26     uint32_t hash = uuidHash(uuid);
27     // ESP32 is natively Little-Endian, so memcpy produces LE
28     // bytes
29     memcpy(nonce, &hash, 4);
30     memcpy(nonce + 4, &fcnt, 4);
31 }
32
33 // AES-128-CTR encryption using ESP32's hardware-accelerated
34 // mbedtls.
35 // CTR mode: ciphertext length == plaintext length, no padding
36 // needed.
37 bool encryptPayload(const uint8_t* plaintext, size_t len,
38                     const uint8_t key[16], const uint8_t nonce
39                     [16],
40                     uint8_t* ciphertext) {
41     if (!plaintext || len == 0 || !key || !nonce || !ciphertext)
42     {
43         return false;
44     }
45 }
```

```
38     }
39
40     mbedtls_aes_context ctx;
41     mbedtls_aes_init(&ctx);
42
43     int ret = mbedtls_aes_setkey_enc(&ctx, key, 128);
44     if (ret != 0) {
45         mbedtls_aes_free(&ctx);
46         return false;
47     }
48
49     // mbedtls_aes_crypt_ctr modifies nonce_counter and
50     // stream_block internally,
51     // so we need mutable copies
52     uint8_t nonceCounter[16];
53     memcpy(nonceCounter, nonce, 16);
54     uint8_t streamBlock[16] = {0};
55     size_t ncOffset = 0;
56
57     ret = mbedtls_aes_crypt_ctr(&ctx, len,
58                                &ncOffset, nonceCounter,
59                                streamBlock,
60                                plaintext, ciphertext);
61
62     mbedtls_aes_free(&ctx);
63     return (ret == 0);
64 }
```

Listing 116: Υλοποίηση κρυπτογράφησης AES-128-CTR (lora_crypto.cpp)

```
1  #pragma once
2
3  #include <stdint.h>
4
5  class Storage; // forward declaration
```

```
6
7 // Initialize the frame counter subsystem.
8 // On cold boot: reads NVS, adds FCNT_COLD_BOOT_GAP, stores in
  RTC.
9 // On deep-sleep wake: RTC value is already valid.
10 void fcntInit(Storage& storage);
11
12 // Increment the frame counter and return the new value.
13 // Lazy-saves to NVS every FCNT_NVS_SAVE_INTERVAL transmissions.
14 uint32_t fcntNext(Storage& storage);
```

Listing 117: Διεπαφή μετρητή πλαισίου (lora_fcnt.h)

```
1 #include "lora_fcnt.h"
2 #include "storage.h"
3 #include "config.h"
4 #include <Arduino.h>
5 #include <esp_sleep.h>
6
7 // Frame counter in RTC memory - survives deep sleep, lost on
  power cycle
8 RTC_DATA_ATTR static uint32_t rtcFcnt = 0;
9 RTC_DATA_ATTR static bool rtcInitialized = false;
10
11 // Transmission counter for lazy NVS saves
12 RTC_DATA_ATTR static uint32_t txSinceLastSave = 0;
13
14 void fcntInit(Storage& storage) {
15     if (rtcInitialized) {
16         // Waking from deep sleep: RTC value is valid, nothing to
          do
17         Serial.printf("[FCNT] Deep-sleep wake, fcnt=%u\n",
            rtcFcnt);
18         return;
19     }
```

```
20
21 // Cold boot: read last saved value from NVS and add safety
    gap
22 uint32_t nvsFcnt = storage.getLoraFcnt();
23 rtcFcnt = nvsFcnt + FCNT_COLD_BOOT_GAP;
24 txSinceLastSave = 0;
25 rtcInitialized = true;
26
27 // Persist the jumped value immediately so the next cold boot
    starts even higher
28 storage.setLoraFcnt(rtcFcnt);
29
30 Serial.printf("[FCNT] Cold boot: NVS=%u, starting at %u (gap
    =%u)\n",
31               nvsFcnt, rtcFcnt, FCNT_COLD_BOOT_GAP);
32 }
33
34 uint32_t fcntNext(Storage& storage) {
35     rtcFcnt++;
36     txSinceLastSave++;
37
38     // Lazy save: write to NVS only every N transmissions to
        reduce flash wear
39     if (txSinceLastSave >= FCNT_NVS_SAVE_INTERVAL) {
40         storage.setLoraFcnt(rtcFcnt);
41         txSinceLastSave = 0;
42         Serial.printf("[FCNT] Saved to NVS: %u\n", rtcFcnt);
43     }
44
45     return rtcFcnt;
46 }
```

Listing 118: Υλοποίηση μετρητή πλαισίου (lora_fcnc.cpp)

```
1 #pragma once
```

```
2
3 #include <stdint.h>
4 #include <stddef.h>
5
6 // Initialize the LoRa radio (SPI + LoRa library) with settings
  from config.h.
7 // Returns true on success.
8 bool loraRadioInit();
9
10 // Transmit a complete LoRa packet: [1B uuid_len][UUID bytes][4B
   fcmt LE][ciphertext]
11 // Returns true on success.
12 bool loraTransmit(const char* uuid, uint32_t fcmt,
13                  const uint8_t* ciphertext, size_t cipherLen);
14
15 // Put LoRa radio into sleep mode (call before deep sleep to save
   power).
16 void loraRadioSleep();
```

Listing 119: Διεπαφή οδηγού ραδιοσυχνότητας LoRa (lora_radio.h)

```
1 #include "lora_radio.h"
2 #include "config.h"
3 #include <Arduino.h>
4 #include <SPI.h>
5 #include <LoRa.h>
6 #include <cstring>
7
8 bool loraRadioInit() {
9     SPI.begin(LORA_SCK_PIN, LORA_MISO_PIN, LORA_MOSI_PIN,
10              LORA_SS_PIN);
11     LoRa.setPins(LORA_SS_PIN, LORA_RST_PIN, LORA_DIO0_PIN);
12
13     if (!LoRa.begin(LORA_FREQUENCY)) {
14         Serial.println("[LoRa] Radio init failed");
15     }
```

```
14         return false;
15     }
16
17     LoRa.setSpreadingFactor(LORA_SF);
18     LoRa.setSignalBandwidth(LORA_BW);
19     LoRa.setCodingRate4(LORA_CR);
20     LoRa.setSyncWord(LORA_SYNC_WORD);
21     LoRa.setTxPower(LORA_TX_POWER);
22
23     Serial.printf("[LoRa] Radio init OK: freq=%ld SF=%d BW=%ld CR
24                    =4/%d SW=0x%02X TX=%ddBm\n",
25                    LORA_FREQUENCY, LORA_SF, LORA_BW, LORA_CR,
26                    LORA_SYNC_WORD, LORA_TX_POWER);
27
28     return true;
29 }
30
31 bool loraTransmit(const char* uuid, uint32_t fcnt,
32                  const uint8_t* ciphertext, size_t cipherLen) {
33     if (!uuid || !ciphertext || cipherLen == 0) return false;
34
35     uint8_t uuidLen = (uint8_t)strlen(uuid);
36
37     // Build packet: [1B uuid_len][UUID bytes][4B fcnt LE][
38     //                ciphertext]
39     uint8_t fcntBytes[4];
40     memcpy(fcntBytes, &fcnt, 4); // ESP32 is natively LE
41
42     LoRa.beginPacket();
43     LoRa.write(uuidLen);
44     LoRa.write((const uint8_t*)uuid, uuidLen);
45     LoRa.write(fcntBytes, 4);
46     LoRa.write(ciphertext, cipherLen);
47     LoRa.endPacket(true); // true = blocking, wait for TX
48                             complete
```

```
44
45     Serial.printf("[LoRa] TX: uuid=%s fcnt=%u payload=%u bytes (
        total=%u)\n",
46                     uuid, fcnt, cipherLen, 1 + uuidLen + 4 +
                        cipherLen);
47     return true;
48 }
49
50 void loraRadioSleep() {
51     LoRa.sleep();
52     Serial.println("[LoRa] Radio entering sleep");
53 }
```

Listing 120: Υλοποίηση οδηγού ραδιοσυχνότητας LoRa (lora_radio.cpp)

References

- [1] Gurpreet Singh and Jaspreet Singh. “Transformative Potential of IoT for Developing Smart Agriculture System: A Systematic Review”. In: *2023 4th International Conference on Communication, Computing and Industry 6.0 (C2I6)*. 2023 4th International Conference on Communication, Computing and Industry 6.0 (C2I6). Bangalore, India: IEEE, Dec. 15, 2023, pp. 1–6. ISBN: 979-8-3503-2732-8. DOI: 10.1109/C2I659362.2023.10430789. URL: <https://ieeexplore.ieee.org/document/10430789/> (visited on 03/15/2026).
- [2] António Monteiro, Sérgio Santos, and Pedro Gonçalves. “Precision Agriculture for Crop and Livestock Farming—Brief Review”. In: *Animals* 11.8 (Aug. 9, 2021), p. 2345. ISSN: 2076-2615. DOI: 10.3390/ani11082345. URL: <https://www.mdpi.com/2076-2615/11/8/2345> (visited on 03/15/2026).
- [3] Dr. Rashmi Sharma, Vishal Mishra, and Suryansh Srivastava. “Enhancing Crop Yields through IoT-Enabled Precision Agriculture”. In: *2023 International Conference on Disruptive Technologies (ICDT)*. 2023 International Conference on Disruptive Technologies (ICDT). Greater Noida, India: IEEE, May 11, 2023, pp. 279–283. ISBN: 979-8-3503-2388-7. DOI: 10.1109/ICDT57929.2023.10151422. URL: <https://ieeexplore.ieee.org/document/10151422/> (visited on 03/15/2026).
- [4] Muhammad Ayaz et al. “Internet-of-Things (IoT)-Based Smart Agriculture: Toward Making the Fields Talk”. In: *IEEE Access* 7 (2019), pp. 129551–129583. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2932609. URL: <https://ieeexplore.ieee.org/document/8784034/> (visited on 03/15/2026).
- [5] Md. Najmul Mowla et al. “Internet of Things and Wireless Sensor Networks for Smart Agriculture Applications: A Survey”. In: *IEEE Access* 11 (2023), pp. 145813–145852. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2023.3346299. URL: <https://ieeexplore.ieee.org/document/10371307/> (visited on 03/15/2026).
- [6] Harleen Kaur, Anand Kr Shukla, and Harpreet Singh. “Review of IoT Technologies used in Agriculture”. In: *2022 2nd International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE)*. 2022 2nd International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE).

- Greater Noida, India: IEEE, Apr. 28, 2022, pp. 1007–1011. ISBN: 978-1-6654-3789-9. DOI: 10.1109/ICACITE53722.2022.9823520. URL: <https://ieeexplore.ieee.org/document/9823520/> (visited on 03/15/2026).
- [7] Partha Pratim Ray. “Internet of things for smart agriculture: Technologies, practices and future direction”. In: *Journal of Ambient Intelligence and Smart Environments* 9.4 (2017), pp. 395–420.
- [8] E. M. B. M. Karunathilake et al. “The Path to Smart Farming: Innovations and Opportunities in Precision Agriculture”. In: *Agriculture* 13.8 (Aug. 11, 2023), p. 1593. ISSN: 2077-0472. DOI: 10.3390/agriculture13081593. URL: <https://www.mdpi.com/2077-0472/13/8/1593> (visited on 03/15/2026).
- [9] Panagiotis Gkotsiopoulos, Dimitrios Zorbas, and Christos Douligeris. “Performance Determinants in LoRa Networks: A Literature Review”. In: *IEEE Communications Surveys & Tutorials* 23.3 (2021), pp. 1721–1758. ISSN: 1553-877X, 2373-745X. DOI: 10.1109/COMST.2021.3090409. URL: <https://ieeexplore.ieee.org/document/9463416/> (visited on 03/14/2026).
- [10] Jeferson Rodrigues Cotrim and João Henrique Kleinschmidt. “LoRaWAN mesh networks: A review and classification of multihop communication”. In: *Sensors* 20.15 (2020), p. 4273.
- [11] Dixin Wu and Jörg Liebeherr. “A low-cost low-power LoRa mesh network for large-scale environmental sensing”. In: *IEEE Internet of Things Journal* 10.19 (2023), pp. 16700–16714.
- [12] Le Tian et al. “Wi-Fi HaLow for the Internet of Things: An up-to-date survey on IEEE 802.11ah research”. In: *Journal of Network and Computer Applications* 182 (May 2021), p. 103036. ISSN: 10848045. DOI: 10.1016/j.jnca.2021.103036. URL: <https://linkinghub.elsevier.com/retrieve/pii/S108480452100062X> (visited on 03/15/2026).
- [13] Emmanuel M. Migabo, Karim D. Djouani, and Anish M. Kurien. “The Narrowband Internet of Things (NB-IoT) Resources Management Performance State of Art, Challenges, and Opportunities”. In: *IEEE Access* 8 (2020), pp. 97658–97675. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.2995938. URL: <https://ieeexplore.ieee.org/document/9097268/> (visited on 03/14/2026).

- [14] Esad Kadusic et al. “The factors for choosing among NB-IoT, LoRaWAN, and Sigfox radio communication technologies for IoT networking”. In: *2022 International Conference on Connected Systems & Intelligence (CSI)*. 2022 International Conference on Connected Systems & Intelligence (CSI). Trivandrum, India: IEEE, Aug. 31, 2022, pp. 1–5. ISBN: 978-1-6654-5815-3. DOI: 10.1109/CSI54720.2022.9923999. URL: <https://ieeexplore.ieee.org/document/9923999/> (visited on 03/14/2026).
- [15] Murugaperumal Krishnamoorthy et al. “A design and development of the smart forest alert monitoring system using IoT”. In: *Journal of sensors* 2023.1 (2023), p. 8063524.
- [16] Heru Nurwarsito and Rizal Dwi Christian. “River water pollutant level monitoring system using websocket protocol and LoRa communication module”. In: *2021 2nd International Conference on ICT for Rural Development (IC-ICTRuDev)*. IEEE. 2021, pp. 1–6.
- [17] Dimitrios Glaroudis, Athanasios Iossifides, and Periklis Chatzimisios. “Survey, comparison and research challenges of IoT application protocols for smart farming”. In: *Computer Networks* 168 (Feb. 2020), p. 107037. ISSN: 13891286. DOI: 10.1016/j.comnet.2019.107037. URL: <https://linkinghub.elsevier.com/retrieve/pii/S1389128619306942> (visited on 03/14/2026).
- [18] Fatma Hmissi and Sofiane Ouni. “A Survey on Application Layer Protocols for IoT Networks”. In: (2022).
- [19] Mikhail Fomichev et al. “Survey and Systematization of Secure Device Pairing”. In: *IEEE Communications Surveys & Tutorials* 20.1 (2018), pp. 517–550. ISSN: 1553-877X. DOI: 10.1109/COMST.2017.2748278. URL: <http://ieeexplore.ieee.org/document/8023969/> (visited on 03/15/2026).
- [20] Fatima Salahdine, Tao Han, and Ning Zhang. “5G, 6G, and Beyond: Recent advances and future challenges”. In: *Annals of Telecommunications* 78.9 (2023), pp. 525–549.
- [21] Alpar Türkoğlu et al. “Comparison Between Time Series and Relational Databases”. In: *2024 9th International Conference on Computer Science and Engineering (UBMK)*. 2024 9th International Conference on Computer Science and Engineering (UBMK). Antalya, Türkiye: IEEE, Oct. 26, 2024, pp. 1174–1177. ISBN: 979-8-3503-6588-7. DOI: 10.1109/UBMK63289.2024.10773532. URL: <https://ieeexplore.ieee.org/document/10773532/> (visited on 03/15/2026).

- [22] Vladislav Rudakov, Merembayev Timur, and Amirgaliyev Yedilkhan. “Comparison of Time Series Databases”. In: *2023 17th International Conference on Electronics Computer and Computation (ICECCO)*. 2023 17th International Conference on Electronics Computer and Computation (ICECCO). Kaskelen, Kazakhstan: IEEE, June 1, 2023, pp. 1–4. ISBN: 979-8-3503-3961-1. DOI: 10.1109/ICECCO58239.2023.10147153. URL: <https://ieeexplore.ieee.org/document/10147153/> (visited on 03/15/2026).
- [23] Eko Simanjuntak, Nico Surantha, and Sani Muhamad Isa. “Evaluation of time-series database on microservice architecture for health monitoring system”. In: *2022 International Symposium on Electronics and Smart Devices (ISESD)*. 2022 International Symposium on Electronics and Smart Devices (ISESD). Bandung, Indonesia: IEEE, Nov. 8, 2022, pp. 1–6. ISBN: 979-8-3503-9660-7. DOI: 10.1109/ISESD56103.2022.9980618. URL: <https://ieeexplore.ieee.org/document/9980618/> (visited on 03/14/2026).
- [24] SOY Hakkı. “ESP8266 and ESP32 series of SoC microcontrollers”. In: *Programmable Smart Microcontroller Cards* 110 (2021).
- [25] Alexander Maier, Andrew Sharp, and Yuriy Vagapov. “Comparative analysis and practical implementation of the ESP32 microcontroller module for the internet of things”. In: *2017 Internet Technologies and Applications (ITA)*. IEEE. 2017, pp. 143–148.
- [26] Resul Butuner and Yusuf Uzun. “Arduino microcontroller card”. In: *Program. Smart Microcontroller Cards* (2021), pp. 1–18.
- [27] Yusuf UZUN and Ozgur DUNDAR. “Raspberry Pi and Models”. In: *Programmable Smart Microcontroller Cards* 34 (2021).
- [28] Ozgur DUNDAR and Sabri KOCER. “Programmable Logic Controllers (PLC)”. In: *Programmable Smart Microcontroller Cards* (2021), p. 50.
- [29] M. H. Ibrahim. “STM32”. In: *Programmable Smart Microcontroller Cards*. Ed. by S. Kocer, O. Dunder, and R. Butuner. ISRES Publishing, 2021, pp. 67–81.
- [30] David Yulizar et al. “Performance analysis comparison of DHT11, DHT22 and DS18B20 as temperature measurement”. In: *Proceedings of the 2nd International Conference on Science Education and Sciences 2022 (ICSES 2022)*. Vol. 8. Springer Nature. 2023, p. 37.

- [31] Francesco Salamone et al. “Low-cost thermohygrometers to assess thermal comfort in the built environment: A laboratory evaluation of their measurement performance”. In: *Buildings* 12.5 (2022), p. 579.
- [32] Limin Yu et al. “Review of research progress on soil moisture sensor technology”. In: *International Journal of Agricultural and Biological Engineering* 14.4 (2021), pp. 32–42.
- [33] Divya Dhatri P V S et al. “A Low-Cost Arduino based Automatic Irrigation System using Soil Moisture Sensor: Design and Analysis”. In: *2019 2nd International Conference on Signal Processing and Communication (ICSPPC)*. 2019 2nd International Conference on Signal Processing and Communication (ICSPPC). Coimbatore, India: IEEE, Mar. 2019, pp. 104–108. ISBN: 978-1-7281-1849-9. DOI: 10.1109/ICSPPC46172.2019.8976483. URL: <https://ieeexplore.ieee.org/document/8976483/> (visited on 03/15/2026).
- [34] Karlisa Priandana and Ramadina Al-Fatihah Wahyu. “Development of Automatic Plant Irrigation System using Soil Moisture Sensors for Precision Agriculture of Chili”. In: *2020 International Conference on Smart Technology and Applications (ICoSTA)*. 2020 International Conference on Smart Technology and Applications (ICoSTA). Surabaya, Indonesia: IEEE, Feb. 2020, pp. 1–4. ISBN: 978-1-7281-3081-1 978-1-7281-3083-5. DOI: 10.1109/ICoSTA48221.2020.1570615896. URL: <https://ieeexplore.ieee.org/document/9079292/> (visited on 03/15/2026).
- [35] Mo'men M Badreldeen et al. “IoT based Smart Irrigation System”. In: 3.1 (2022).
- [36] Nuradin Mohamed Abdikadir et al. “Smart Irrigation System”. In: *International Journal of Electrical and Electronics Engineering* 10.8 (Sept. 1, 2023), pp. 224–234. ISSN: 23488379. DOI: 10.14445/23488379/IJEEE-V10I8P122. URL: <https://www.internationaljournalssrg.org/IJEEE/paper-details?Id=560> (visited on 03/15/2026).
- [37] Aisel Louie G. Agulto, Jericho B. Casañares, and Michael C. Pacis. “Solar-Powered Soil Moisture Detection and Monitoring Using Capacitive Sensor Probe and Calibrated by Gravimetric Method”. In: *2024 16th International Conference on Computer and Automation Engineering (ICCAE)*. 2024 16th International Conference on Computer and Automation Engineering (ICCAE). Melbourne, Australia: IEEE, Mar. 14, 2024, pp. 720–

724. ISBN: 979-8-3503-7005-8. DOI: 10.1109/ICCAE59995.2024.10569213. URL: <https://ieeexplore.ieee.org/document/10569213/> (visited on 03/15/2026).
- [38] Doru Cornei and Cristian Fosalau. “Using IoT in Smart Agriculture: Study about Practical Realizations and Testing in a Real Environment”. In: *2022 International Conference and Exposition on Electrical And Power Engineering (EPE)*. 2022 International Conference and Exposition on Electrical And Power Engineering (EPE). Iasi, Romania: IEEE, Oct. 20, 2022, pp. 013–018. ISBN: 978-1-6654-8994-2. DOI: 10.1109/EPE56121.2022.9959823. URL: <https://ieeexplore.ieee.org/document/9959823/> (visited on 03/15/2026).
- [39] Bhuvan Gour, Preetam Suman, and Sasmita Padhy. “A new Paradigm of Smart Agriculture”. In: *2024 First International Conference on Data, Computation and Communication (ICDCC)*. 2024 First International Conference on Data, Computation and Communication (ICDCC). Sehore, India: IEEE, Nov. 29, 2024, pp. 137–141. ISBN: 979-8-3315-3295-6. DOI: 10.1109/ICDCC62744.2024.10961885. URL: <https://ieeexplore.ieee.org/document/10961885/> (visited on 03/15/2026).
- [40] Fauzan Prasetyo Eka Putra et al. “Trends and Best Practices in API-Based Web Development Using Laravel and React”. In: *Brilliance: Research of Artificial Intelligence* 5.1 (June 16, 2025), pp. 223–233. ISSN: 2807-9035. DOI: 10.47709/brilliance.v5i1.5971. URL: <https://jurnal.itscience.org/index.php/brilliance/article/view/5971> (visited on 03/15/2026).
- [41] Osama M.A. AL-atraqchi. “A Proposed Model for Build a Secure Restful API to Connect between Server Side and Mobile Application Using Laravel Framework with Flutter Toolkits”. In: *Cihan University-Erbil Scientific Journal* 6.2 (Aug. 20, 2022), pp. 28–35. ISSN: 2707-6377, 2519-6979. DOI: 10.24086/cuesj.v6n2y2022.pp28–35. URL: <https://journals.cihanuniversity.edu.iq/index.php/cuesj/article/view/575> (visited on 03/15/2026).
- [42] Praveen Kumar Donta et al. “Survey on recent advances in IoT application layer protocols and machine learning scope for research directions”. In: *Digital Communications and Networks* 8.5 (Oct. 2022), pp. 727–744. ISSN: 23528648. DOI: 10.1016/j.dcan.2021.10.004. URL: <https://linkinghub.elsevier.com/retrieve/pii/S2352864821000845> (visited on 03/15/2026).

- [43] Naga Srinivasarao Chilamkurthy et al. “Low-Power Wide-Area Networks: A Broad Overview of Its Different Aspects”. In: *IEEE Access* 10 (2022), pp. 81926–81959. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2022.3196182. URL: <https://ieeexplore.ieee.org/document/9848798/> (visited on 03/15/2026).
- [44] Mahyar Amini et al. “MAHAMGOSTAR.COM AS A CASE STUDY FOR ADOPTION OF LARAVEL FRAMEWORK AS THE BEST PROGRAMMING TOOLS FOR PHP BASED WEB DEVELOPMENT FOR SMALL AND MEDIUM ENTERPRISES”. In: (2021).
- [45] Piyush Tripathi, Mahdi H. Miraz, and Snigdha Joshi. “Comparative Analysis of MongoDB and InfluxDB for Time Series Data Management in IoT Environments: A Study on Performance, Scalability, and Concurrency”. In: *2023 International Conference on Computing, Networking, Telecommunications & Engineering Sciences Applications (CoNTESA)*. 2023 International Conference on Computing, Networking, Telecommunications & Engineering Sciences Applications (CoNTESA). Zagreb, Croatia: IEEE, Dec. 14, 2023, pp. 39–42. ISBN: 979-8-3503-8275-4. DOI: 10.1109/CoNTESA61248.2023.10384962. URL: <https://ieeexplore.ieee.org/document/10384962/> (visited on 03/14/2026).
- [46] Espressif Systems. *ESP32-C6 Series Datasheet*. Version 1.5. Espressif Systems. 2024. URL: https://www.espressif.com/sites/default/files/documentation/esp32-c6_datasheet_en.pdf (visited on 04/18/2026).
- [47] Erich Gamma et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. Boston, MA: Addison-Wesley, 1994.
- [48] Semtech Corporation. *SX1276/77/78/79 - 137 MHz to 1020 MHz Low Power Long Range Transceiver Datasheet*. Rev. 7. Semtech Corporation. 2020. URL: <https://www.semtech.com/products/wireless-rf/lora-core/sx1276> (visited on 04/18/2026).
- [49] Espressif Systems. *ESP32 Series Datasheet*. Version 1.0. Espressif Systems. 2016. URL: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf (visited on 04/18/2026).
- [50] Jakob Nielsen and Thomas K. Landauer. “A mathematical model of the finding of usability problems”. In: *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems*. CHI '93. Amsterdam, The Netherlands: Associ-

ation for Computing Machinery, 1993, pp. 206–213. ISBN: 0897915755. doi: 10.1145/
169059.169166. URL: <https://doi.org/10.1145/169059.169166>.