



Σχολή Θετικών Επιστημών και Τεχνολογίας
Μεταπτυχιακή Εξειδίκευση στα Πληροφοριακά Συστήματα

Διπλωματική Εργασία

Σχεδιασμός και Ανάπτυξη Web-Based Συστήματος Παρακολούθησης
και Διαχείρισης Δικτυακών Συσκευών με Ενοποιημένο Dashboard

Παναγιώτης Δοδόπουλος

Επιβλέπων καθηγητής: Παπαμιχαήλ Γεώργιος

Πάτρα, Ιούνιος 2026

Η παρούσα εργασία αποτελεί πνευματική ιδιοκτησία του φοιτητή («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 Ανάπτυξη Web-Based Συστήματος Παρακολούθησης και Διαχείρισης Δικτυακών Συσκευών με Ενοποιημένο Dashboard

Παναγιώτης Δοδόπουλος

Επιτροπή Επίβλεψης Διπλωματικής Εργασίας

Επιβλέπων Καθηγητής:

Παπαμιχαήλ Γεώργιος

Συν-Επιβλέπων Καθηγητής:

Καραπιτέρης Δημήτριος

Πάτρα, Ιούνιος 2026

*Θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα καθηγητή μου, Δρ. Γεώργιο Παπαμιχαήλ,
για την πολύτιμη καθοδήγηση και την υποστήριξή του καθ' όλη τη διάρκεια εκπόνησης της
παρούσας εργασίας.*

*Ένα ξεχωριστό ευχαριστώ οφείλω στη σύζυγό μου, Ελεάννα, για την αμέριστη συμπαράσταση
και υπομονή της. Τέλος, αφιερώνω την εργασία αυτή στο νεογέννητο παιδί μας, τη μεγαλύτερη
πηγή έμπνευσης και χαράς.*

Περίληψη

Η παρούσα διπλωματική εργασία αφορά στον σχεδιασμό και την υλοποίηση ενός διαδικτυακού συστήματος για την παρακολούθηση και τη διαχείριση δικτυακών συσκευών. Σκοπός της εργασίας είναι η ανάπτυξη μιας ενιαίας πλατφόρμας, μέσω της οποίας είναι δυνατή η κεντρική εποπτεία δικτυακών υποδομών και η υποστήριξη βασικών λειτουργιών διαχείρισης δικτύου. Το σύστημα απευθύνεται σε χρήστες με αρμοδιότητες διαχείρισης πληροφοριακών και δικτυακών υποδομών.

Η εφαρμογή υποστηρίζει την καταχώρηση και οργάνωση δικτυακών συσκευών, όπως δρομολογητές, μεταγωγείς και εξυπηρετητές, καθώς και την παρακολούθηση της κατάστασής τους μέσω βασικών δεικτών λειτουργίας. Παρέχεται η δυνατότητα ελέγχου της διαθεσιμότητας των συσκευών και της συλλογής δεδομένων που σχετίζονται με τη λειτουργία τους, με σκοπό την έγκαιρη αναγνώριση προβλημάτων. Η παρουσίαση των πληροφοριών πραγματοποιείται μέσω ενός κεντρικού πίνακα ελέγχου, ο οποίος διευκολύνει την επισκόπηση της συνολικής κατάστασης του δικτύου.

Στο πλαίσιο της εργασίας πραγματοποιήθηκε ανάλυση των απαιτήσεων του συστήματος και σχεδιασμός της αρχιτεκτονικής του. Ακολούθησε ο σχεδιασμός της βάσης δεδομένων και η υλοποίηση της εφαρμογής, αξιοποιώντας σύγχρονες τεχνολογίες διαδικτυακής ανάπτυξης. Ιδιαίτερη μέριμνα δόθηκε σε θέματα ασφάλειας και ελέγχου πρόσβασης, μέσω της υποστήριξης πολλαπλών χρηστών με διαφορετικά επίπεδα δικαιωμάτων.

Τέλος, πραγματοποιήθηκαν δοκιμές λειτουργικότητας με σκοπό την αξιολόγηση της ορθής λειτουργίας του συστήματος. Τα αποτελέσματα των δοκιμών δείχνουν ότι η εφαρμογή ανταποκρίνεται στις βασικές απαιτήσεις παρακολούθησης και διαχείρισης δικτυακών συσκευών και μπορεί να αποτελέσει μια αξιόπιστη βάση για μελλοντικές επεκτάσεις και βελτιώσεις.

Λέξεις – Κλειδιά

Παρακολούθηση Δικτυακών Συσκευών, Παρακολούθηση σε Πραγματικό Χρόνο, Διαχείριση Δικτυακής Υποδομής, Σύστημα Διαχείρισης Δικτύου

Design and Development of a Web-Based Network Device Monitoring and Management System with a Unified Dashboard

Panagiotis Dodopoulos

Abstract

This master's thesis focuses on the design and implementation of a web-based system for monitoring and managing network devices. The objective of the work is the development of a unified platform that enables centralized oversight of network infrastructures and supports basic network management functions. The system is intended for users responsible for the administration of information and network infrastructures.

The application supports the registration and organization of network devices, such as routers, switches, and servers, as well as the monitoring of their status through key performance indicators. It provides the ability to check device availability and collect operational data, aiming for the timely detection of potential issues. Information is presented via a centralized dashboard, facilitating an overview of the overall network status.

As part of the work, a requirements analysis and system architecture design were carried out. This was followed by the database design and the implementation of the application, utilizing modern web development technologies. Special attention was given to security and access control, supporting multiple users with different levels of permissions.

Finally, functionality tests were conducted to evaluate the correct operation of the system. The test results demonstrate that the application meets the basic requirements for network device monitoring and management and can serve as a reliable foundation for future extensions and improvements.

Keywords

Network Device Monitoring, Real-Time Monitoring, IT Infrastructure Management

Περιεχόμενα

Περίληψη	7
Abstract	8
Περιεχόμενα	9
Κατάλογος Εικόνων / Σχημάτων	12
Κατάλογος Πινάκων	13
Συντομογραφίες & Ακρωνύμια	14
1. Εισαγωγή	1
1.1 Παρουσίαση Προβλήματος	2
1.2 Βασικές Έννοιες	3
1.3 Στόχοι Εργασίας	4
1.4 Δομή Εργασίας	5
2. Θεωρητικό Υπόβαθρο και Τεχνολογίες Υλοποίησης	7
2.1 Συστήματα Παρακολούθησης Δικτυακών Συσκευών	7
2.2 Τεχνικές και Πρωτόκολλα Παρακολούθησης Δικτύου	8
2.2.1 SNMP (Simple Network Management Protocol)	8
2.2.2 ICMP (Internet Control Message Protocol)	9
2.2.3 Syslog	10
2.2.4 NetFlow / sFlow	11
2.3 Υφιστάμενες Λύσεις	12
2.3.1 Nagios	12
2.3.2 Zabbix	14
2.3.3 PRTG Network Monitor	15
2.4 Τεχνολογίες Υλοποίησης της Εφαρμογής	17
2.4.1 Backend - Python και FastAPI	18
2.4.2 Frontend- React.js	19
2.4.3 Βάση Δεδομένων - PostgreSQL	20
2.4.4 Real Time Monitoring - WebSockets	21
2.4.5 Ασύγχρονες Διεργασίες Υποβάθρου - Python Asyncio και Redis	22
2.4.6 Περιβάλλον Ανάπτυξης Εφαρμογής - Visual Studio Code	23
2.4.7 Version Control - Git και Github	23
2.4.8 Containerization - Docker	24
2.4.9 API Testing - Postman	25
3. Απαιτήσεις και Σχεδιασμός	26
3.1 Λειτουργικές Απαιτήσεις	26
3.2 Μη Λειτουργικές Απαιτήσεις	30
3.3 Περιπτώσεις Χρήσης	33
4. Ανάπτυξη της Εφαρμογής	38

4.1 Δομή και Αρχιτεκτονική του Frontend	38
4.1.1 Οργάνωση και Δομή του Κώδικα (src/)	39
4.1.2 Layouts, Routing και Έλεγχος Πρόσβασης (Route Guards)	40
4.1.3 Δυναμικό Θέμα (Dynamic Theming & index.css)	40
4.1.4 Frontend Σελίδες και Λειτουργικότητα	41
4.2 Backend και Μοντελοποίηση Δεδομένων	44
4.2.1 Βάση Δεδομένων και Object-Relational Mapping (SQLAlchemy)	44
4.2.2 Μοντελοποίηση Οντοτήτων (SQLAlchemy Models)	45
4.3 Μηχανισμός Παρακολούθησης (Monitoring Engine) και Real-time Επικοινωνία	45
4.3.1 Ασύγχρονος Scheduler	45
4.3.2 Διεργασία Polling ανά Συσκευή	46
4.3.3 Υπολογισμός Ρυθμού Κυκλοφορίας (Stateful Traffic Rate)	47
4.3.4 Διασύνδεση Redis Pub/Sub και WebSockets	48
4.3.5 Διαχείριση Alerting και email	49
4.4 REST API και Ασφάλεια	50
4.4.1 Δομή API Routers	50
4.4.2 Αυθεντικοποίηση JWT (JSON Web Tokens) και RBAC	51
4.4.3 Κρυπτογράφηση Credentials στη Βάση Δεδομένων	52
4.5 Containerization και Ενορχήστρωση με Docker	53
4.5.1 Dockerfiles	53
4.5.2 Docker Compose και Παραμετροποίηση (.env)	53
5. Λειτουργικότητα και Διεπαφή Χρήστη	55
5.1 Είσοδος Χρήστη	55
5.2 Επαναφορά Κωδικού Πρόσβασης	57
5.3 Κεντρική Σελίδα Παρακολούθησης	60
5.3.1 KPI Widgets	61
5.3.2 Δυναμικά Γραφήματα (Chart.js)	62
5.3.3 Live Alerts Pane	62
5.4 Inventory και Διαχείριση Συσκευών	62
5.4.1 Πίνακας Inventory	63
5.4.2 Προσθήκη και Επεξεργασία Συσκευής	63
5.4.2 Bulk Import και Data Export	64
5.5 Απομακρυσμένη Διαχείριση και Κονσόλα Ενεργειών	64
5.6 Διαχείριση Χρηστών και Ρόλων	65
5.7 Καταγραφές Ασφαλείας (Audit Logs)	66
5.8 Ρυθμίσεις Συστήματος (Settings)	67
6. Δοκιμές και Αξιολόγηση	69
6.1 Λειτουργικός Έλεγχος και Δοκιμές Ενοποίησης (Functional & Integration Testing)	69
6.1.1 Δοκιμές REST API με χρήση Postman	69
6.1.2 Δοκιμές Σεναρίων End-to-End (E2E Data Flow)	70
6.2 Αξιολόγηση Απόδοσης και Κλιμακωσιμότητας (Performance & Scalability Analysis)	70
6.2.1 Scaling Test του Monitoring Engine	70

6.2.2 Response Time του REST API	71
6.3 Έλεγχος και Επαλήθευση Ασφαλείας (Security Audit)	71
6.3.1 Επαλήθευση Κρυπτογράφησης Credentials στη Βάση	71
6.3.2 Έλεγχος Role-Based Access Control (RBAC)	72
6.4 Αξιολόγηση Χρηστικότητας από Χρήστες	72
6.4.1 Μεθοδολογία	72
6.4.2 Προφίλ Συμμετεχόντων	73
6.4.3 Ποσοτικά Αποτελέσματα	73
6.4.4 Ποιοτικά Αποτελέσματα	74
6.4.5 Συμπεράσματα της Αξιολόγησης	74
6.5 Συμπεράσματα Δοκιμών και Τεχνικές Βελτιώσεις	75
7. Συμπεράσματα και Μελλοντικές Κατευθύνσεις	75
7.1 Αποτίμηση και Συμπεράσματα	75
7.2 Μελλοντικές Κατευθύνσεις και Επεκτάσεις	76
Παράρτημα Α: Swagger API Documentation	80

Κατάλογος Εικόνων / Σχημάτων

Εικόνα 1 Το WEB UI του Nagios	13
Εικόνα 2 Το WEB UI του Zabbix	15
Εικόνα 3 Το WEB UI του PRTG Network Monitor	17
Εικόνα 4 Δομή καταλόγων και αρχείων του Frontend (React.js)	39
Εικόνα 5 Παράδειγμα της εφαρμογής σε Dark Mode	41
Εικόνα 6 Η διεπαφή του dashboard με τα widgets πραγματικού χρόνου και τα γραφήματα τάσεων	41
Εικόνα 7 Η σελίδα διαχείρισης συσκευών με τον πίνακα inventory και τις επιλογές import/export	42
Εικόνα 8 Η σελίδα διαχείρισης χρηστών για τους διαχειριστές της εφαρμογής	42
Εικόνα 9 Η σελίδα ρυθμίσεων με τους range sliders για τα CPU, Memory και Latency thresholds	43
Εικόνα 10 Δομή καταλόγων και αρχείων του Backend (Python)	44
Εικόνα 11 Πηγαίος κώδικας του ασύγχρονου Scheduler	46
Εικόνα 12 Πηγαίος κώδικας για ICMP polling	47
Εικόνα 13 Η ασύγχρονη συνάρτηση subscribe_and_forward της κλάσης RedisService	49
Εικόνα 14 Ο πηγαίος κώδικας της αποστολής ηλεκτρονικών μηνυμάτων	50
Εικόνα 15 Η τεκμηρίωση των REST API endpoints μέσω του Swagger UI	51
Εικόνα 16 Ο πηγαίος κώδικας του αρχείου backend/services/encryptor.py	52
Εικόνα 17 Περιεχόμενα του αρχείου docker-compose.yml	54
Εικόνα 18 Η σελίδα εισόδου (Login) του συστήματος	56
Εικόνα 19 Μήνυμα λανθασμένου email/κωδικού πρόσβασης	57
Εικόνα 20 Η φόρμα αιτήματος επαναφοράς κωδικού	58
Εικόνα 21 Το ηλεκτρονικό μήνυμα με τον σύνδεσμο αλλαγής του κωδικού πρόσβασης	59
Εικόνα 22 Η σελίδα ορισμού νέου κωδικού πρόσβασης	60
Εικόνα 23 Η κεντρική σελίδα του dashboard	61
Εικόνα 24 Η σελίδα διαχείρισης συσκευών	62
Εικόνα 25 Το παράθυρο προσθήκης νέας συσκευής	63
Εικόνα 26 Το παράθυρο προσθήκης credentials για συσκευές	64
Εικόνα 27 Το μενού ενεργειών και η κονσόλα απομακρυσμένης διαχείρισης	65
Εικόνα 28 Η οθόνη διαχείρισης χρηστών	66
Εικόνα 29 Ο πίνακας Audit Logs του συστήματος	67
Εικόνα 30 Η σελίδα ρυθμίσεων (Settings) με τους range sliders και τη λειτουργία validation	68
Σχήμα 1 Διάγραμμα Περιπτώσεων Χρήσης ανά Ρόλο Χρήστη	34
Σχήμα 2 Διάγραμμα Περίπτωσης Χρήσης για Προσθήκη Νέας Συσκευής	34
Σχήμα 3 Διάγραμμα Περίπτωσης Χρήσης για Παρακολούθηση Συσκευής σε Πραγματικό Χρόνο	35
Σχήμα 4 Διάγραμμα Περίπτωσης Χρήσης για Ιστορικά Δεδομένα	36
Σχήμα 5 Διάγραμμα Περίπτωσης Χρήσης Χρηστών και Δικαιωμάτων	37

Κατάλογος Πινάκων

Πίνακας 1 Χρόνος ολοκλήρωσης κύκλου polling σε συνάρτηση με τον αριθμό των συσκευών	71
Πίνακας 2 Δημογραφικά στοιχεία και υπόβαθρο των συμμετεχόντων στην αξιολόγηση χρηστικότητας	73
Πίνακας 1 Επιμέρους και μέση βαθμολογία SUS των συμμετεχόντων	73

Συντομογραφίες & Ακρωνύμια

API	Application Programming Interface
REST	Representational State Transfer
HTTP	HyperText Transfer Protocol
JWT	JSON Web Token
JSON	JavaScript Object Notation
SPA	Single Page Application
UI	User Interface
UX	User Experience
ORM	Object Relational Mapping
SQL	Structured Query Language
CSS	Cascading Style Sheets
IDE	Integrated Development Environment
SMTP	Simple Mail Transfer Protocol
SNMP	Simple Network Management Protocol
ICMP	Internet Control Message Protocol
SSH	Secure Shell
MIB	Management Information Base
IoT	Internet of Things
IP	Internet Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
CPU	Central Processing Unit
RAM	Random Access Memory
RBAC	Role-Based Access Control
AES	Advanced Encryption Standard
PBKDF2	Password-Based Key Derivation Function 2
TLS	Transport Layer Security
CRUD	Create, Read, Update, Delete
DBMS	Database Management System

1. Εισαγωγή

Η αποτελεσματική παρακολούθηση και διαχείριση των δικτυακών υποδομών αποτελεί σήμερα μια από τις βασικές προκλήσεις για οργανισμούς και επιχειρήσεις, καθώς η διαθεσιμότητα, η απόδοση και η ασφάλεια των δικτύων επηρεάζουν άμεσα τη λειτουργία των πληροφοριακών συστημάτων. Η ανάγκη για κεντρική εποπτεία και έγκαιρη αναγνώριση προβλημάτων καθιστά απαραίτητα τα σύγχρονα εργαλεία παρακολούθησης και διαχείρισης δικτυακών συσκευών.

Η συνεχής αύξηση της χρήσης δικτυακών συσκευών, συμπεριλαμβανομένων των routers, switches, servers και IoT συσκευών, έχει δημιουργήσει την ανάγκη για ολοκληρωμένα συστήματα που να συνδυάζουν λειτουργίες καταγραφής, παρακολούθησης και ειδοποιήσεων. Η υιοθέτηση web-based λύσεων επιτρέπει την πρόσβαση σε πραγματικό χρόνο, ανεξαρτήτως τοποθεσίας, και διευκολύνει τη διαχείριση πολυχρηστικών περιβαλλόντων, διασφαλίζοντας παράλληλα την ασφάλεια και την ορθή κατανομή των δικαιωμάτων πρόσβασης.

Η παρούσα εργασία στοχεύει στην ανάπτυξη μιας ενιαίας διαδικτυακής πλατφόρμας για την παρακολούθηση και διαχείριση δικτυακών συσκευών μέσω ενός ενοποιημένου dashboard, υποστηρίζοντας βασικές λειτουργίες όπως η συλλογή μετρικών απόδοσης, η ανίχνευση αλλαγών κατάστασης και η αποστολή ειδοποιήσεων στους χρήστες. Παράλληλα, εξετάζονται θέματα όπως η σχεδίαση της βάσης δεδομένων, η αρχιτεκτονική του συστήματος και η αξιολόγηση της απόδοσής του, με στόχο την εξασφάλιση ενός λειτουργικού και επεκτάσιμου συστήματος.

Το κεφάλαιο αυτό θέτει τα θεμέλια για την κατανόηση της σημασίας των συστημάτων παρακολούθησης και διαχείρισης δικτύου, αναδεικνύοντας τον τρόπο με τον οποίο μπορούν να συμβάλουν στην αποτελεσματική λειτουργία των δικτυακών υποδομών και στην υποστήριξη των επιχειρησιακών διαδικασιών.

1.1 Παρουσίαση Προβλήματος

Σήμερα, η αποτελεσματική λειτουργία των οργανισμών εξαρτάται σε μεγάλο βαθμό από την ομαλή λειτουργία των δικτυακών υποδομών τους. Η διαθεσιμότητα, η αξιοπιστία και η

απόδοση των δικτυακών συσκευών επηρεάζουν άμεσα τις πληροφοριακές υπηρεσίες, τη διαχείριση δεδομένων και τη συνολική απόδοση των επιχειρησιακών διαδικασιών. Η διαχείριση αυτών των υποδομών περιλαμβάνει κρίσιμες λειτουργίες, όπως η παρακολούθηση της κατάστασης των συσκευών, η συλλογή μετρικών απόδοσης, η ανίχνευση δυσλειτουργιών, η διαχείριση ειδοποιήσεων και η ασφάλεια πρόσβασης των χρηστών.

Παρά τη διαθεσιμότητα τεχνολογικών εργαλείων, πολλές οργανώσεις συνεχίζουν να χρησιμοποιούν αποσπασματικές ή χειροκίνητες λύσεις. Τέτοιες πρακτικές μπορεί να οδηγήσουν σε καθυστερήσεις στην αναγνώριση προβλημάτων, περιορισμένη δυνατότητα πρόληψης δυσλειτουργιών, μειωμένη αποδοτικότητα προσωπικού και δυσκολία στη συνολική οργάνωση των δικτύων.

Συγκεκριμένα, τα βασικά προβλήματα που αντιμετωπίζουν οι οργανισμοί περιλαμβάνουν:

- **Παρακολούθηση κατάστασης συσκευών:** Χωρίς αυτοματοποιημένα εργαλεία, η αναγνώριση offline ή προβληματικών συσκευών είναι αργή, με αποτέλεσμα ενδεχόμενες διακοπές υπηρεσιών.
- **Συλλογή και ανάλυση μετρικών:** Η περιορισμένη πληροφόρηση για τη χρήση CPU, μνήμης και δικτυακής κίνησης δυσχεραίνει την τεκμηριωμένη λήψη αποφάσεων και την προληπτική συντήρηση.
- **Διαχείριση ειδοποιήσεων:** Οι χειροκίνητες διαδικασίες καθυστερούν την ενημέρωση των υπευθύνων και μπορεί να οδηγήσουν σε λανθασμένη αντιμετώπιση συμβάντων.
- **Ασφάλεια και έλεγχος πρόσβασης:** Η ανεπαρκής διαχείριση χρηστών και δικαιωμάτων ενέχει κίνδυνο μη εξουσιοδοτημένων τροποποιήσεων ή πρόσβασης σε ευαίσθητα δεδομένα.

Η κατανόηση αυτών των προκλήσεων αναδεικνύει την ανάγκη για οργανωμένη προσέγγιση στη διαχείριση των δικτυακών υποδομών και τονίζει τις δυσκολίες που αντιμετωπίζουν οι οργανισμοί στην καθημερινή λειτουργία τους, αποτελώντας τη βάση για την περαιτέρω ανάλυση και σχεδιασμό λύσεων στα επόμενα κεφάλαια.

1.2 Βασικές Έννοιες

Για την κατανόηση των θεμάτων που πραγματεύεται η παρούσα διπλωματική εργασία, κρίνεται απαραίτητη η εισαγωγή ορισμένων βασικών εννοιών και όρων που σχετίζονται με την παρακολούθηση και διαχείριση δικτυακών υποδομών.

Δικτυακές Συσκευές (Network Devices): Ο όρος περιλαμβάνει συσκευές που χρησιμοποιούνται για τη διασύνδεση, διαχείριση και μετάδοση δεδομένων σε ένα δίκτυο. Στην παρούσα εργασία, οι κύριες συσκευές που εξετάζονται είναι οι δρομολογητές (routers), οι μεταγωγείς (switches), οι εξυπηρετητές (servers) και οι συσκευές IoT, οι οποίες παρέχουν κρίσιμες λειτουργίες για την επικοινωνία και την επεξεργασία πληροφοριών στο δίκτυο.

Web-Based Σύστημα: Πρόκειται για εφαρμογή που εκτελείται μέσω ενός διαδικτυακού περιηγητή και παρέχει πρόσβαση σε λειτουργίες και δεδομένα χωρίς την ανάγκη τοπικής εγκατάστασης λογισμικού. Η επιλογή web-based προσέγγισης επιτρέπει την ευελιξία, την κεντρική διαχείριση και τη δυνατότητα πρόσβασης από οποιοδήποτε σημείο και συσκευή.

Real-Time Monitoring: Η παρακολούθηση σε πραγματικό χρόνο αναφέρεται στην συνεχή καταγραφή και ενημέρωση της κατάστασης των συσκευών και των δικτυακών υπηρεσιών, επιτρέποντας την άμεση αναγνώριση προβλημάτων και την έγκαιρη παρέμβαση για την αποφυγή δυσλειτουργιών.

Dashboard: Ένα κεντρικό περιβάλλον απεικόνισης πληροφοριών, όπου παρουσιάζονται συνοπτικά δεδομένα και μετρικές απόδοσης των δικτυακών συσκευών. Το dashboard μπορεί να περιλαμβάνει γραφήματα, ειδοποιήσεις, πίνακες κατάστασης και άλλα στοιχεία που διευκολύνουν την παρακολούθηση και λήψη αποφάσεων.

SNMP (Simple Network Management Protocol): Πρόκειται για ένα πρωτόκολλο που χρησιμοποιείται για τη συλλογή και διαχείριση πληροφοριών σχετικά με τις συσκευές ενός δικτύου. Επιτρέπει την ανάγνωση και καταγραφή μετρικών, καθώς και την εκτέλεση βασικών εντολών διαχείρισης απομακρυσμένα.

Alerting και Notifications: Μηχανισμοί που ειδοποιούν τους υπεύθυνους διαχείρισης όταν εντοπίζονται προβλήματα ή παραβιάζονται προκαθορισμένα όρια λειτουργίας, ώστε να ληφθούν άμεσα διορθωτικά μέτρα.

Η κατανόηση των παραπάνω εννοιών αποτελεί προαπαιτούμενο για τη σωστή ερμηνεία των λειτουργιών του συστήματος και τη μελέτη των διαδικασιών που περιγράφονται στα επόμενα κεφάλαια της εργασίας.

1.3 Στόχοι Εργασίας

Η παρούσα διπλωματική εργασία στοχεύει στη σχεδίαση και ανάπτυξη μιας web-based εφαρμογής παρακολούθησης και διαχείρισης δικτυακών συσκευών με ενοποιημένο dashboard. Οι στόχοι της εργασίας συνοψίζονται ως εξής:

- **Εγγραφή και Διαχείριση Συσκευών:** Καταχώρηση δικτυακών συσκευών με λεπτομερή στοιχεία (διεύθυνση IP, διεύθυνση MAC, τύπος συσκευής, τοποθεσία, κατασκευαστής, μοντέλο). Κατηγοριοποίηση συσκευών ανά δίκτυο, τμήμα ή λειτουργικό ρόλο. Διαχείριση κωδικών πρόσβασης και πρωτοκόλλων επικοινωνίας (SNMP, SSH, HTTP/HTTPS). Ομαδοποίηση συσκευών για μαζικές ενέργειες και οργάνωση.
- **Real-time Monitoring και Status Tracking:** Συνεχής παρακολούθηση της διαθεσιμότητας συσκευών μέσω ping, SNMP polling ή άλλων πρωτοκόλλων. Έλεγχος βασικών παραμέτρων όπως uptime, CPU usage, memory utilization, network traffic. Ανίχνευση αλλαγών κατάστασης (online/offline) και καταγραφή διαθεσιμότητας. Visual indicators για γρήγορη αναγνώριση προβλημάτων.
- **Dashboard και Visualization:** Κεντρικό dashboard με συνοπτική εικόνα όλων των συσκευών και της κατάστασής τους. Interactive charts και graphs για την απεικόνιση μετρικών απόδοσης. Customizable widgets που επιτρέπουν στους χρήστες να προσαρμόσουν το dashboard στις ανάγκες τους.
- **Alerting και Notification System:** Δημιουργία rules και thresholds για αυτοματοποιημένες ειδοποιήσεις. Πολλαπλά κανάλια ειδοποιήσεων (email, web notifications, webhook integrations). Escalation policies για διαβαθμισμένες ειδοποιήσεις βάσει σοβαρότητας. Alert acknowledgment και tracking για παρακολούθηση επίλυσης προβλημάτων.
- **Ιστορικά Δεδομένα και Reporting:** Αποθήκευση ιστορικών δεδομένων για trend analysis και capacity planning. Δημιουργία αναφορών διαθεσιμότητας, απόδοσης και συμβάντων. Εξαγωγή δεδομένων σε διάφορες μορφές (CSV, JSON). Scheduled reports που αποστέλλονται αυτόματα σε stakeholders.

- **User Management και Security:** Multi-user support με διαφορετικά επίπεδα πρόσβασης (admin, operator, viewer). Role-based access control για περιορισμό πρόσβασης σε συγκεκριμένες συσκευές ή λειτουργίες. Secure authentication και authorization mechanisms. Audit logging για παρακολούθηση ενεργειών χρηστών.
- **API και Integrations:** RESTful API για ενσωμάτωση με άλλα συστήματα και automation tools. Webhook support για real-time notifications σε εξωτερικά συστήματα. Integration με δημοφιλείς monitoring solutions (Nagios, Zabbix, PRTG). Bulk operations API για μαζική διαχείριση συσκευών.
- **Configuration Management:** Backup και restore configurations δικτυακών συσκευών. Configuration change detection και version control. Template-based configuration deployment. Compliance checking για security και organizational policies.

Συνολικά, η εργασία επιδιώκει την ανάπτυξη ενός ολιστικού και επεκτάσιμου συστήματος, ικανό να υποστηρίξει τη διαχείριση δικτυακών υποδομών, να βελτιώσει την απόδοση και να διευκολύνει τη λήψη τεκμηριωμένων αποφάσεων από διαχειριστές και IT professionals.

1.4 Δομή Εργασίας

Η διπλωματική εργασία έχει σχεδιαστεί ώστε να καλύπτει ολοκληρωμένα όλα τα στάδια ανάπτυξης ενός web-based συστήματος παρακολούθησης και διαχείρισης δικτυακών συσκευών, από τη θεωρητική τεκμηρίωση μέχρι την υλοποίηση και αξιολόγηση. Το έργο οργανώνεται σε επτά κεφάλαια, τα οποία αλληλοσυνδέονται και προσφέρουν μια σαφή εικόνα της μεθοδολογίας, των επιλογών και των αποτελεσμάτων της εργασίας.

Κεφάλαιο 1 – Εισαγωγή

Παρουσιάζεται το πλαίσιο της εργασίας, η αναγκαιότητα ανάπτυξης του συστήματος και οι στόχοι που επιδιώκονται. Περιλαμβάνονται οι βασικές έννοιες, η διατύπωση του προβλήματος και η παρουσίαση των επιδιωκόμενων λειτουργιών του συστήματος.

Κεφάλαιο 2 – Θεωρητικό Υπόβαθρο και Τεχνολογίες Υλοποίησης

Παρουσιάζονται οι βασικές έννοιες και αρχές των συστημάτων παρακολούθησης δικτυακών συσκευών, καθώς και τα πρωτόκολλα και οι τεχνικές που χρησιμοποιούνται για τη συλλογή και απεικόνιση δεδομένων. Παράλληλα, γίνεται αναφορά σε υφιστάμενες λύσεις, ενώ παρουσιάζονται οι τεχνολογίες που επιλέχθηκαν για την υλοποίηση της εφαρμογής και αιτιολογούνται οι σχετικές επιλογές.

Κεφάλαιο 3 – Απαιτήσεις και Σχεδιασμός

Καταγράφονται οι λειτουργικές και μη λειτουργικές απαιτήσεις της εφαρμογής, περιγράφονται οι χρήστες και τα σενάρια χρήσης τους. Παρουσιάζεται ο σχεδιασμός της αρχιτεκτονικής του συστήματος και της βάσης δεδομένων, καθώς και η λογική οργάνωσης των δεδομένων και των δικτυακών οντοτήτων.

Κεφάλαιο 4 – Ανάπτυξη της Εφαρμογής

Αναλύεται η διαδικασία υλοποίησης, οι τεχνολογίες front-end και back-end, η διαχείριση δεδομένων και η ενσωμάτωση των βασικών λειτουργιών, όπως real-time monitoring, ειδοποιήσεις και dashboards. Παρουσιάζονται τα διαγράμματα ροής και τα αρχιτεκτονικά μοντέλα.

Κεφάλαιο 5 – Λειτουργικότητα και Διεπαφή Χρήστη

Περιγράφονται οι δυνατότητες της εφαρμογής από την πλευρά του χρήστη. Αναλύονται τα dashboards, οι διεπαφές παρακολούθησης, η διαχείριση συσκευών, ειδοποιήσεων και αναφορών, καθώς και η εμπειρία χρήσης.

Κεφάλαιο 6 – Δοκιμές και Αξιολόγηση

Καταγράφονται οι δοκιμές λειτουργικότητας και απόδοσης, η ανάλυση αποτελεσμάτων, η αξιολόγηση της σταθερότητας και η εμπειρία χρήσης. Επισημαίνονται τα δυνατά σημεία του συστήματος και προτείνονται βελτιώσεις.

Κεφάλαιο 7 – Συμπεράσματα και Μελλοντικές Κατευθύνσεις

Συνοψίζονται τα αποτελέσματα της εργασίας και αξιολογείται η επίτευξη των στόχων. Παρουσιάζονται προτάσεις για μελλοντική εξέλιξη, όπως η ενσωμάτωση τεχνητής νοημοσύνης για ανίχνευση ανωμαλιών, predictive analytics, cloud monitoring και mobile εφαρμογές για παρακολούθηση σε πραγματικό χρόνο.

Η οργανωμένη αυτή δομή επιτρέπει στον αναγνώστη να παρακολουθήσει τη μελέτη από τη θεωρία στην πράξη, κατανοώντας ταυτόχρονα τις τεχνικές επιλογές και τα αποτελέσματα της ανάπτυξης του συστήματος.

2. Θεωρητικό Υπόβαθρο και Τεχνολογίες Υλοποίησης

Στο παρόν κεφάλαιο παρουσιάζονται οι βασικές θεωρητικές έννοιες που σχετίζονται με τα συστήματα παρακολούθησης και διαχείρισης δικτυακών συσκευών, καθώς και οι κύριες τεχνικές που χρησιμοποιούνται για τη συλλογή και επεξεργασία δεδομένων δικτύου. Αναλύονται τα πρωτόκολλα και οι μηχανισμοί που υποστηρίζουν την παρακολούθηση της κατάστασης και της απόδοσης των δικτυακών υποδομών. Παράλληλα, γίνεται αναφορά σε υφιστάμενες λύσεις network monitoring, με στόχο την κατανόηση των δυνατοτήτων και των περιορισμών τους. Τέλος, παρουσιάζονται οι τεχνολογίες που επιλέχθηκαν για την υλοποίηση της εφαρμογής και αιτιολογούνται οι σχετικές επιλογές, λαμβάνοντας υπόψη την απόδοση, την επεκτασιμότητα και τις απαιτήσεις ενός σύγχρονου web-based συστήματος.

2.1 Συστήματα Παρακολούθησης Δικτυακών Συσκευών

Τα συστήματα παρακολούθησης δικτυακών συσκευών (Network Device Monitoring Systems) αποτελούν βασικό τμήμα της σύγχρονης διαχείρισης πληροφοριακών και τηλεπικοινωνιακών υποδομών. Μέσω αυτών, καθίσταται δυνατή η συνεχής εποπτεία της κατάστασης, της διαθεσιμότητας και της απόδοσης των συσκευών που συνθέτουν ένα δίκτυο, όπως δρομολογητές, μεταγωγείς, εξυπηρετητές και συσκευές Internet of Things (IoT).

Η ανάγκη για τέτοιου είδους συστήματα προκύπτει από την αυξανόμενη πολυπλοκότητα των δικτυακών υποδομών και την απαίτηση για υψηλά επίπεδα αξιοπιστίας και διαθεσιμότητας. Η έγκαιρη ανίχνευση δυσλειτουργιών, η πρόληψη πιθανών βλαβών και η άμεση απόκριση σε περιστατικά αποτελούν κρίσιμους παράγοντες για τη διασφάλιση της απρόσκοπτης λειτουργίας των υπηρεσιών που βασίζονται στο δίκτυο.

Στα συστήματα παρακολούθησης συλλέγονται δεδομένα που αφορούν τόσο τη βασική κατάσταση των συσκευών (online/offline) όσο και μετρικές απόδοσης, όπως ο χρόνος λειτουργίας (uptime), η χρήση επεξεργαστή και μνήμης, καθώς και η κίνηση δικτύου. Τα δεδομένα αυτά αποθηκεύονται και επεξεργάζονται, επιτρέποντας την εξαγωγή συμπερασμάτων σχετικά με τη συμπεριφορά του δικτύου σε βάθος χρόνου.

Παράλληλα, παρέχεται η δυνατότητα ορισμού μηχανισμών ειδοποίησης, μέσω των οποίων ενημερώνονται οι διαχειριστές σε περίπτωση υπέρβασης προκαθορισμένων ορίων ή αλλαγής της κατάστασης μιας συσκευής. Με τον τρόπο αυτό, επιτυγχάνεται η έγκαιρη αντιμετώπιση προβλημάτων, πριν αυτά επηρεάσουν τη συνολική λειτουργία του συστήματος.

Συνολικά, τα συστήματα παρακολούθησης δικτυακών συσκευών συμβάλλουν στη βελτιστοποίηση της διαχείρισης των πόρων, στη μείωση του χρόνου αποκατάστασης βλαβών και στη λήψη τεκμηριωμένων αποφάσεων, βασισμένων σε πραγματικά δεδομένα λειτουργίας του δικτύου.

2.2 Τεχνικές και Πρωτόκολλα Παρακολούθησης Δικτύου

Η παρακολούθηση δικτυακών συσκευών βασίζεται σε διάφορες τεχνικές και πρωτόκολλα, τα οποία επιτρέπουν τη συλλογή δεδομένων για τη διαθεσιμότητα, την απόδοση και τη λειτουργία των δικτυακών υποδομών. Η κατανόηση αυτών των πρωτοκόλλων είναι κρίσιμη για την ανάπτυξη ενός αποτελεσματικού συστήματος network monitoring.

2.2.1 SNMP (Simple Network Management Protocol)

Το SNMP είναι το πιο διαδεδομένο πρωτόκολλο για τη διαχείριση και παρακολούθηση δικτυακών συσκευών, όπως routers, switches, firewalls, servers και IoT συσκευές. Αποτελεί βασικό εργαλείο για τη συλλογή πληροφοριών σχετικά με την κατάσταση και τις επιδόσεις ενός δικτύου, ενώ επιτρέπει και την απομακρυσμένη διαχείριση των συσκευών.

Το SNMP βασίζεται σε τρεις βασικούς ρόλους:

- **Manager (Διαχειριστής):**

Το σύστημα ή ο server που συλλέγει δεδομένα από τις συσκευές του δικτύου, αναλύει τις πληροφορίες και δημιουργεί αναφορές ή ειδοποιήσεις.

- **Agent (Πράκτορας):**

Λογισμικό εγκατεστημένο σε κάθε συσκευή που παρακολουθεί συγκεκριμένα δεδομένα, απαντά σε αιτήματα του manager και στέλνει ειδοποιήσεις όταν προκύπτουν σημαντικά γεγονότα.

- **MIB (Management Information Base):**

Μια βάση δεδομένων που περιγράφει τις πληροφορίες που μπορούν να συλλεχθούν από τη συσκευή. Περιλαμβάνει αντικείμενα όπως CPU usage, memory usage, network interfaces, errors, uptime, θερμοκρασία και πολλά άλλα.

Οι τύποι των SNMP μηνυμάτων είναι οι εξής:

- **GET:** Ο manager ζητάει συγκεκριμένες πληροφορίες από τον agent.
- **GETNEXT / GETBULK:** Χρησιμοποιούνται για ανάκτηση μεγάλων ή συνεχόμενων συνόλων δεδομένων.

- **SET:** Ο manager στέλνει εντολή στον agent για αλλαγή παραμέτρων, όπως ενεργοποίηση ή απενεργοποίηση διεπαφής.
- **TRAP:** Ο agent στέλνει ειδοποίηση στον manager αυτόματα, π.χ. σε περίπτωση αποτυχίας σύνδεσης, υπερφόρτωσης ή αστοχίας υλικού.
- **INFORM:** Παρόμοιο με το TRAP αλλά απαιτεί επιβεβαίωση από τον manager, εξασφαλίζοντας αξιόπιστη παράδοση της ειδοποίησης.

Το SNMP προσφέρει σημαντικά πλεονεκτήματα που το καθιστούν ένα από τα πιο δημοφιλή πρωτόκολλα παρακολούθησης δικτύου. Υποστηρίζεται από σχεδόν όλες τις δικτυακές συσκευές, παρέχοντας δυνατότητα απομακρυσμένης διαχείρισης και παρακολούθησης σε πραγματικό χρόνο. Μέσω των MIBs, οι διαχειριστές μπορούν να παρακολουθούν ένα ευρύ φάσμα παραμέτρων και να επεκτείνουν τη συλλογή δεδομένων χωρίς αλλαγές στο πρωτόκολλο. Επιπλέον, οι αυτοματοποιημένες ειδοποιήσεις μέσω TRAP ή INFORM επιτρέπουν την έγκαιρη ανίχνευση προβλημάτων και τη γρήγορη αντίδραση από τους υπεύθυνους του δικτύου.

Παρά τα πλεονεκτήματά του, το SNMP παρουσιάζει και ορισμένους περιορισμούς. Οι παλαιότερες εκδόσεις (v1, v2) χρησιμοποιούν απλή αυθεντικοποίηση με κοινό κλειδί, γεγονός που δημιουργεί ζητήματα ασφάλειας, τα οποία αντιμετωπίζονται μόνο με την υιοθέτηση του SNMP v3. Επιπλέον, η συνεχής συλλογή δεδομένων μπορεί να επιβαρύνει το δίκτυο, ιδιαίτερα σε μεγάλα περιβάλλοντα. Η κατανόηση και η σωστή χρήση των MIBs απαιτεί εξειδικευμένη γνώση, ενώ η συχνότητα polling πρέπει να ρυθμιστεί προσεκτικά ώστε να παρέχεται ενημερωμένη εικόνα χωρίς υπερφόρτωση.

2.2.2 ICMP (Internet Control Message Protocol)

Το ICMP είναι ένα πρωτόκολλο που χρησιμοποιείται κυρίως για την παρακολούθηση της διαθεσιμότητας και της συνδεσιμότητας των συσκευών δικτύου. Το πιο γνωστό παράδειγμα είναι η εντολή ping, η οποία στέλνει μηνύματα Echo Request σε έναν κόμβο και περιμένει Echo Reply, επιβεβαιώνοντας ότι η συσκευή είναι online. Επιπλέον, χρησιμοποιείται από εργαλεία όπως το traceroute, για τον εντοπισμό της διαδρομής που ακολουθούν τα πακέτα σε ένα δίκτυο. Το ICMP επιτρέπει τη γρήγορη ανίχνευση προβλημάτων σύνδεσης και παρέχει βασικά στοιχεία για τη μέτρηση καθυστέρησης (latency) και απώλειας πακέτων.

Τα πλεονεκτήματα του ICMP είναι ότι είναι ελαφρύ, εύκολο στη χρήση και προσφέρει άμεση πληροφόρηση σχετικά με τη διαθεσιμότητα των κόμβων και την κατάσταση του δικτύου σε πραγματικό χρόνο. Η χρήση του δεν απαιτεί ιδιαίτερη ρύθμιση ή εγκατάσταση, ενώ μπορεί να συνδυαστεί με άλλα εργαλεία για τη δημιουργία πιο ολοκληρωμένων συστημάτων παρακολούθησης. Παράλληλα, επιτρέπει στους διαχειριστές να εντοπίζουν γρήγορα προβλήματα συνδεσιμότητας και να παρακολουθούν βασικούς δείκτες απόδοσης.

Ωστόσο, το ICMP έχει και περιορισμούς. Δεν παρέχει πληροφορίες για την απόδοση ή την κατάσταση των πόρων μιας συσκευής, όπως CPU ή μνήμη, περιορίζοντας τη χρήση του σε βασικό έλεγχο διαθεσιμότητας. Επιπλέον, τα μηνύματα ICMP μπορούν να μπλοκαριστούν από firewalls ή πολιτικές ασφαλείας, με αποτέλεσμα η παρακολούθηση να μην είναι αξιόπιστη σε όλα τα περιβάλλοντα. Τέλος, η συχνή χρήση του σε μεγάλα δίκτυα μπορεί να δημιουργήσει επιπλέον κίνηση, αν και ο φόρτος που δημιουργείται συνήθως είναι χαμηλός σε σχέση με άλλα πρωτόκολλα.

2.2.3 Syslog

Το Syslog είναι ένα πρωτόκολλο που χρησιμοποιείται για τη συλλογή, αποθήκευση και αναφορά μηνυμάτων καταγραφής (log messages) από δικτυακές συσκευές, servers, εφαρμογές και συστήματα ασφαλείας. Παρέχει έναν τυποποιημένο τρόπο για τη μετάδοση πληροφοριών σχετικά με γεγονότα συστήματος, σφάλματα, προειδοποιήσεις και λειτουργικές ενέργειες. Τα μηνύματα Syslog περιλαμβάνουν συνήθως πληροφορίες όπως timestamp, severity level (π.χ. error, warning, info), host, εφαρμογή ή διεργασία και την περιγραφή του γεγονότος, επιτρέποντας συγκεντρωτική παρακολούθηση και ανάλυση. Χρησιμοποιείται ευρέως για auditing, troubleshooting και monitoring σε πραγματικό χρόνο.

Τα πλεονεκτήματα του Syslog περιλαμβάνουν την τυποποίηση της καταγραφής γεγονότων και την κεντρική συλλογή τους, που διευκολύνει την ανάλυση και τη διαχείριση μεγάλου όγκου δεδομένων. Επιτρέπει στους διαχειριστές να παρακολουθούν τη λειτουργία συσκευών και εφαρμογών σε ένα ενιαίο σημείο, ενώ διευκολύνει την ανίχνευση προβλημάτων και την εφαρμογή πολιτικών ασφαλείας. Το Syslog μπορεί να χρησιμοποιηθεί σε συνδυασμό με εργαλεία ανάλυσης log ή SIEM (Security Information and Event Management) για την

εξαγωγή στατιστικών, την παρακολούθηση ασυνήθιστων δραστηριοτήτων και την παραγωγή ειδοποιήσεων.

Παρά τα πλεονεκτήματά του, το Syslog παρουσιάζει ορισμένα μειονεκτήματα. Δεν παρέχει μηχανισμό επιβεβαίωσης παράδοσης των μηνυμάτων, γεγονός που μπορεί να οδηγήσει σε απώλεια δεδομένων αν η δικτυακή σύνδεση διακοπεί. Επιπλέον, τα μηνύματα δεν είναι πάντα τυποποιημένα πλήρως ανάμεσα σε διαφορετικούς κατασκευαστές, κάτι που απαιτεί προσαρμογή κατά την ανάλυση των logs. Τέλος, η συλλογή μεγάλου όγκου μηνυμάτων σε πραγματικό χρόνο μπορεί να επιβαρύνει το δίκτυο ή τον server καταγραφής, αν δεν υλοποιηθεί κατάλληλη στρατηγική φιλτραρίσματος και αρχειοθέτησης.

2.2.4 NetFlow / sFlow

Το NetFlow (αναπτυγμένο από τη Cisco) και το sFlow (sampling Flow) είναι πρωτόκολλα που χρησιμοποιούνται για τη συλλογή πληροφοριών σχετικά με την κυκλοφορία δικτύου. Επιτρέπουν στους διαχειριστές να παρακολουθούν τα πακέτα δεδομένων που διέρχονται από τις συσκευές δικτύου, καταγράφοντας πληροφορίες όπως προέλευση και προορισμό IP, πρωτόκολλο, αριθμούς ports, και όγκο δεδομένων. Το NetFlow συγκεντρώνει πλήρη στατιστικά για flows, ενώ το sFlow χρησιμοποιεί sampling, δηλαδή συλλέγει δεδομένα μόνο από δείγμα της κυκλοφορίας, μειώνοντας το φόρτο στο δίκτυο. Και τα δύο πρωτόκολλα παρέχουν πολύτιμες πληροφορίες για traffic analysis, capacity planning και εντοπισμό προβλημάτων.

Τα πλεονεκτήματα του NetFlow / sFlow είναι η δυνατότητα λεπτομερούς παρακολούθησης της κίνησης σε πραγματικό χρόνο και η υποστήριξη ανάλυσης μοτίβων δικτύου. Επιτρέπουν την ανίχνευση συμφόρησης, την ανάλυση χρηστών και εφαρμογών, και τη βελτιστοποίηση της διαχείρισης bandwidth. Η δυνατότητα εξαγωγής δεδομένων σε κεντρικά εργαλεία παρακολούθησης (π.χ. dashboards) επιτρέπει στους διαχειριστές να έχουν συνολική εικόνα της δικτυακής κατάστασης και να λαμβάνουν αποφάσεις βασισμένες σε μετρήσιμα δεδομένα.

Παρά τα πλεονεκτήματα, υπάρχουν και περιορισμοί. Το NetFlow απαιτεί σημαντικούς πόρους στις συσκευές για πλήρη συλλογή δεδομένων, ενώ το sFlow μειώνει το φόρτο αλλά χάνει λεπτομέρειες λόγω sampling. Επιπλέον, η ανάλυση μεγάλου όγκου δεδομένων μπορεί να απαιτεί ισχυρό hardware ή εξειδικευμένο λογισμικό. Τέλος, η παρακολούθηση traffic σε πραγματικό χρόνο απαιτεί σωστό σχεδιασμό του δικτύου και του συστήματος συλλογής, διαφορετικά υπάρχει κίνδυνος καθυστερήσεων ή απώλειας δεδομένων.

2.3 Υφιστάμενες Λύσεις

Τα κυριότερα συστήματα παρακολούθησης και διαχείρισης δικτυακών υποδομών, όπως το Nagios, το Zabbix και το PRTG Network Monitor, μελετήθηκαν για την κατανόηση των δυνατοτήτων και περιορισμών τους. Στόχος της ανάλυσης ήταν η αναγνώριση βέλτιστων πρακτικών και η επισήμανση σημείων όπου μπορούν να εισαχθούν βελτιώσεις σε μια νέα εφαρμογή.

Τα εργαλεία αυτά προσφέρουν παρακολούθηση πολλαπλών τύπων συσκευών, συλλογή μετρικών απόδοσης και ειδοποιήσεις για προβλήματα. Παράλληλα, εμφανίζουν περιορισμούς όπως η σχετικά περίπλοκη εγκατάσταση, η περιορισμένη φιλικότητα προς τον χρήστη και η έλλειψη πλήρως διαδραστικών, ενοποιημένων dashboards.

Τα συμπεράσματα από τη μελέτη των υπάρχοντων λύσεων καθόρισαν τον σχεδιασμό της εφαρμογής, εξασφαλίζοντας την ενσωμάτωση βέλτιστων πρακτικών.

2.3.1 Nagios

Το Nagios είναι ένα από τα πιο γνωστά συστήματα παρακολούθησης δικτύων και υπηρεσιών. Σχεδιάστηκε για να παρέχει συνεχή εποπτεία των συσκευών, των υπηρεσιών και των εφαρμογών ενός δικτύου. Μέσω του Nagios, οι διαχειριστές μπορούν να παρακολουθούν διαθεσιμότητα servers, routers, switches, εφαρμογές και άλλες υποδομές, ενώ ενημερώνονται σε πραγματικό χρόνο μέσω ειδοποιήσεων όταν ανιχνεύονται προβλήματα ή ασυνήθιστη συμπεριφορά. Το Nagios χρησιμοποιεί plugins για τη συλλογή δεδομένων και την εκτέλεση ελέγχων, κάτι που επιτρέπει μεγάλη ευελιξία και επεκτασιμότητα.

Τα βασικά χαρακτηριστικά του περιλαμβάνουν:

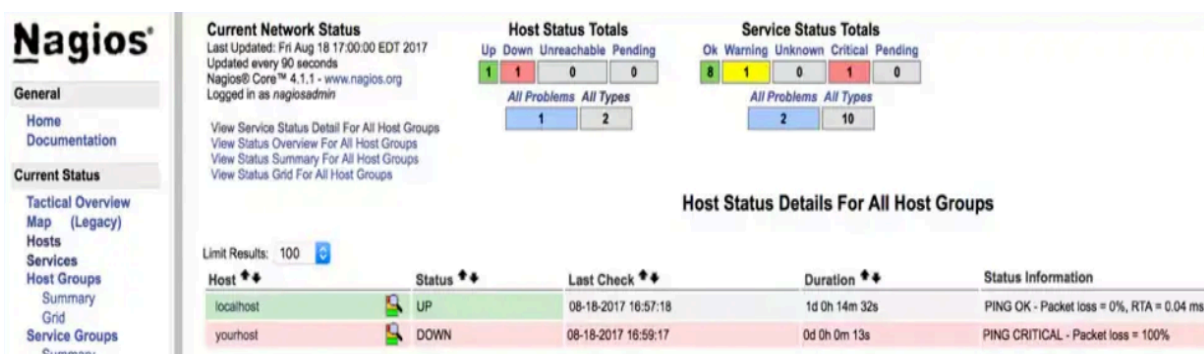
- **Monitoring υποδομών:** Έλεγχος uptime, CPU, μνήμης, χρήσης δίσκου, και δικτυακής κυκλοφορίας.
- **Alerting:** Αυτόματες ειδοποιήσεις μέσω email ή SMS όταν προκύπτουν προβλήματα ή όταν ξεπερνιούνται thresholds.
- **Plugins και επεκτασιμότητα:** Χιλιάδες διαθέσιμα plugins για κάθε τύπο συσκευής ή υπηρεσίας, ενώ οι χρήστες μπορούν να δημιουργούν και δικά τους.
- **Reporting και ιστορικά δεδομένα:** Παρακολούθηση τάσεων, logs και εκθέσεις για capacity planning και αναφορά συμβάντων.

Πλεονεκτήματα:

- Ευέλικτο και επεκτάσιμο, με μεγάλο οικοσύστημα plugins.
- Σταθερότητα και αξιοπιστία, ευρέως χρησιμοποιούμενο σε επιχειρησιακά περιβάλλοντα.
- Υποστήριξη κοινοτήτων και πλούσια τεκμηρίωση.

Μειονεκτήματα:

- Η αρχική ρύθμιση μπορεί να είναι πολύπλοκη, ιδιαίτερα για νέους χρήστες.
- Το interface θεωρείται παρωχημένο σε σχέση με πιο σύγχρονα web-based dashboards.
- Δεν υποστηρίζει εγγενώς την πραγματικού χρόνου visualisation με interactive graphs, κάτι που απαιτεί πρόσθετα εργαλεία ή πρόσθετο configuration.



Εικόνα 1 Το WEB UI του Nagios

2.3.2 Zabbix

Το Zabbix αποτελεί ένα ολοκληρωμένο σύστημα παρακολούθησης δικτύων, servers, εφαρμογών και cloud υποδομών. Σχεδιάστηκε για να παρέχει συνεχή εποπτεία, ειδοποιήσεις και αναφορές σε πραγματικό χρόνο, ενώ υποστηρίζει μεγάλα δίκτυα με εκατοντάδες ή και χιλιάδες συσκευές. Η αρχιτεκτονική του βασίζεται σε έναν κεντρικό server που συλλέγει δεδομένα μέσω agent ή agentless μεθόδων (π.χ. SNMP, IPMI, JMX), τα αποθηκεύει σε βάση δεδομένων και τα παρουσιάζει μέσω web interface με dashboards και γραφήματα.

Τα βασικά χαρακτηριστικά του περιλαμβάνουν:

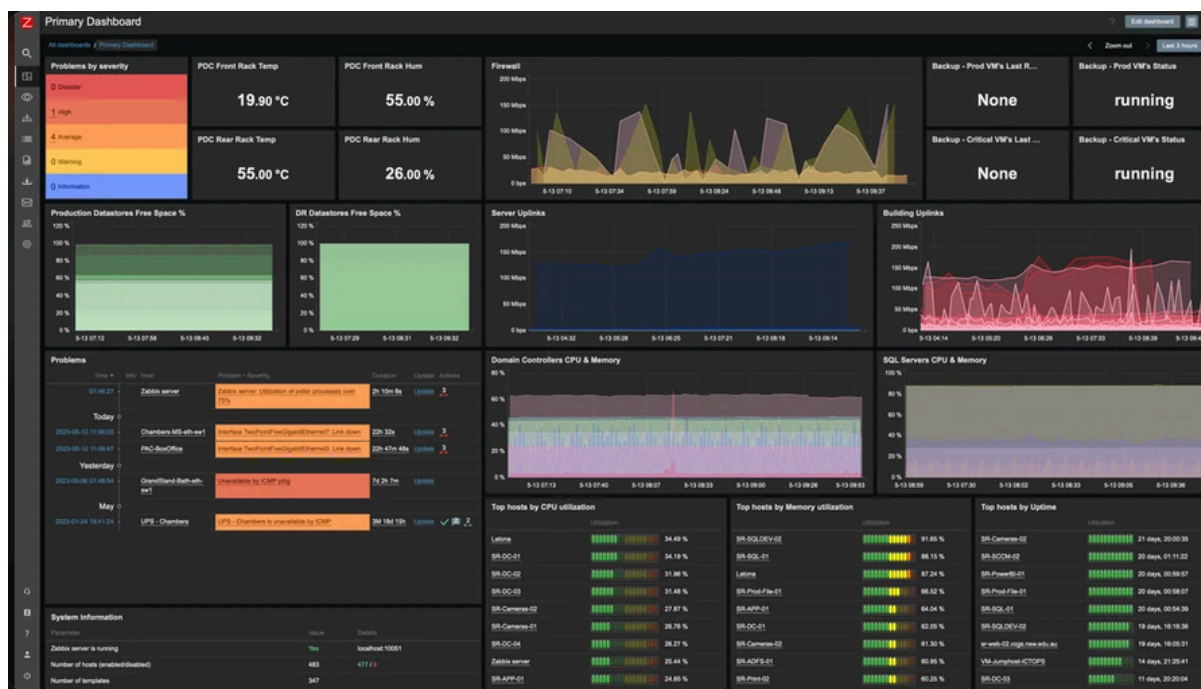
- **Monitoring σε πραγματικό χρόνο:** Συλλογή μετρικών όπως CPU, μνήμη, δίσκος, δικτυακή κίνηση, υπηρεσίες και εφαρμογές.
- **Alerting και escalation:** Ορισμός κανόνων ειδοποίησης με thresholds και αυτοματοποιημένες ενέργειες σε περίπτωση προβλημάτων.
- **Ιστορικά δεδομένα και trend analysis:** Αποθήκευση μετρικών για αναλύσεις τάσεων, capacity planning και ανίχνευση ανωμαλιών.
- **Ολοκληρωμένα dashboards:** Διαδραστικά γραφήματα, charts και widgets που παρέχουν συνολική εικόνα της κατάστασης του δικτύου.
- **Agent και agentless monitoring:** Δυνατότητα συλλογής δεδομένων με χρήση agent ή χωρίς, ανάλογα με τον τύπο της συσκευής.

Πλεονεκτήματα:

- Παρέχει πλήρη εικόνα της υποδομής και προηγμένα analytics.
- Υποστηρίζει μεγάλη κλίμακα και πολλαπλά είδη συσκευών και πρωτοκόλλων.
- Διαδραστικά dashboards με υψηλή παραμετροποίηση.

Μειονεκτήματα:

- Απαιτεί σημαντική αρχική ρύθμιση και εκπαίδευση για τους διαχειριστές.
- Η διαχείριση μεγάλου όγκου δεδομένων μπορεί να απαιτεί επιπλέον resources και βελτιστοποίηση της βάσης δεδομένων.
- Παρά την πληρότητα, το user interface μπορεί να φαίνεται πολύπλοκο σε νέους χρήστες.



Εικόνα 2 Το WEB UI του Zabbix

2.3.3 PRTG Network Monitor

Το PRTG Network Monitor είναι ένα εργαλείο παρακολούθησης δικτύου που επικεντρώνεται στην ευκολία χρήσης και την ταχεία εγκατάσταση. Προσφέρει πλήρη εσοπτεία της δικτυακής υποδομής, συμπεριλαμβανομένων servers, switches, routers, εφαρμογών και υπηρεσιών cloud. Η συλλογή δεδομένων γίνεται μέσω πολλών πρωτοκόλλων, όπως SNMP, WMI, HTTP, Ping, NetFlow, ενώ οι χρήστες μπορούν να παρακολουθούν την απόδοση των συσκευών σε πραγματικό χρόνο μέσω διαδραστικών dashboards.

Τα κύρια χαρακτηριστικά του περιλαμβάνουν:

- **Monitoring πολλαπλών συσκευών:** Συνεχής έλεγχος της κατάστασης hardware, λογισμικού και υπηρεσιών.
- **Alerting και ειδοποιήσεις:** Αυτόματες ειδοποιήσεις μέσω email, SMS ή push notifications όταν ανιχνεύονται προβλήματα ή υπερβάσεις ορίων.
- **Visualisation και dashboards:** Προσφέρει γραφήματα, χάρτες και dashboards που δίνουν πλήρη εικόνα της υποδομής και της απόδοσης.

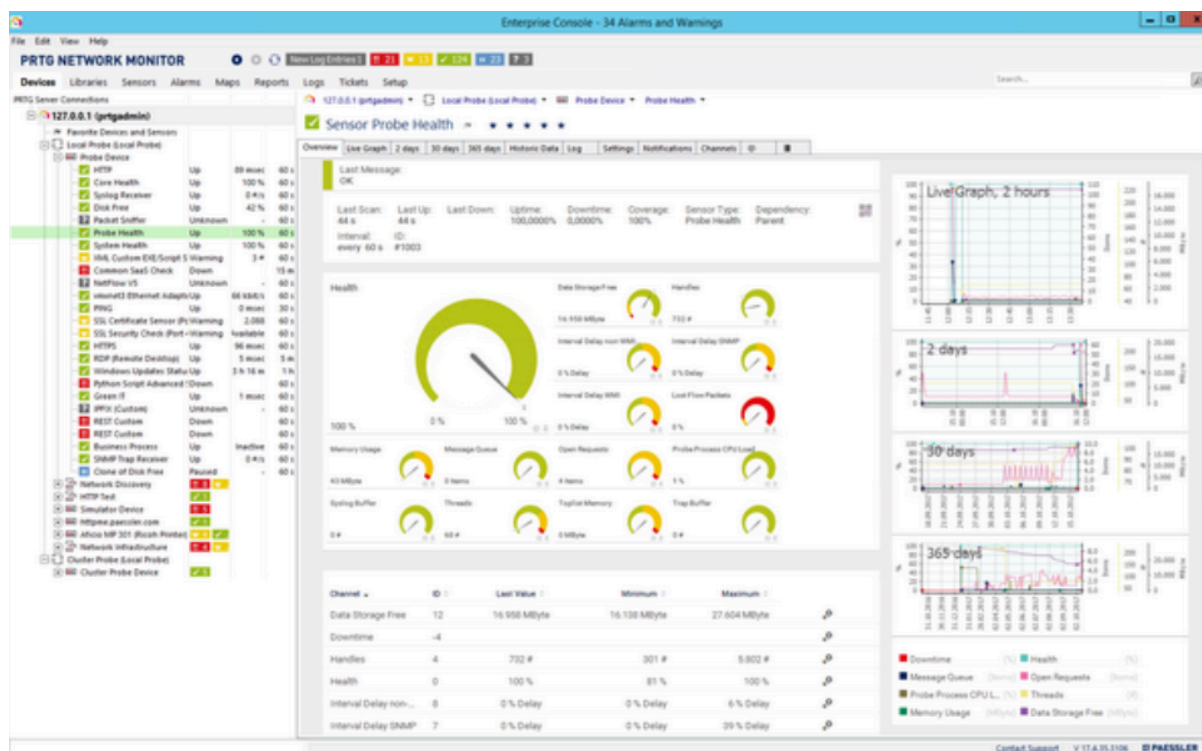
- **Reporting και ιστορικά δεδομένα:** Συλλογή και ανάλυση δεδομένων για αναφορές, πρόβλεψη προβλημάτων και capacity planning.
- **Εύκολη εγκατάσταση και χρήση:** Απευθύνεται σε μικρές και μεσαίες επιχειρήσεις, με φιλικό περιβάλλον χρήστη και γρήγορη παραμετροποίηση.

Πλεονεκτήματα:

- Γρήγορη εγκατάσταση και εύκολη διαχείριση, χωρίς εξειδικευμένες γνώσεις.
- Πλήρης εποπτεία και διαδραστική οπτικοποίηση δεδομένων.
- Υποστήριξη πολλών πρωτοκόλλων και ευελιξία στην παραμετροποίηση ειδοποιήσεων.

Μειονεκτήματα:

- Μπορεί να γίνει δαπανηρό για μεγάλα δίκτυα λόγω του μοντέλου αδειοδότησης ανά αισθητήρα.
- Λιγότερο κατάλληλο για πολύπλοκες, μεγάλες υποδομές σε σχέση με εργαλεία όπως Zabbix.
- Παρά την απλότητα, οι δυνατότητες παραμετροποίησης είναι περιορισμένες συγκριτικά με open-source λύσεις.



Εικόνα 3 Το WEB UI του PRTG Network Monitor

2.4 Τεχνολογίες Υλοποίησης της Εφαρμογής

Για την υλοποίηση του προτεινόμενου συστήματος παρακολούθησης και διαχείρισης δικτυακών συσκευών επιλέχθηκε ένα σύνολο σύγχρονων τεχνολογιών, οι οποίες υποστηρίζουν την ανάπτυξη web-based εφαρμογών με έμφαση στη διαθεσιμότητα, την επεκτασιμότητα και την απόδοση σε πραγματικό χρόνο. Η επιλογή των τεχνολογιών βασίστηκε τόσο στις λειτουργικές απαιτήσεις του συστήματος όσο και στις σύγχρονες πρακτικές ανάπτυξης λογισμικού στον τομέα των πληροφοριακών συστημάτων και της διαχείρισης δικτύων.

Η αρχιτεκτονική της εφαρμογής ακολουθεί τη λογική του διαχωρισμού σε επίπεδα (backend, frontend και βάση δεδομένων), διευκολύνοντας τη συντήρηση, την επεκτασιμότητα και τη μελλοντική εξέλιξη του συστήματος. Παράλληλα, αξιοποιούνται τεχνολογίες που υποστηρίζουν ασύγχρονη επεξεργασία, επικοινωνία σε πραγματικό χρόνο και αυτοματοποίηση εργασιών, στοιχεία κρίσιμα για ένα σύστημα παρακολούθησης δικτυακών υποδομών.

Στις επόμενες υποενότητες παρουσιάζονται αναλυτικά οι τεχνολογίες που χρησιμοποιούνται, καθώς και ο ρόλος τους στο συνολικό σύστημα.

2.4.1 Backend - Python και FastAPI

Η υλοποίηση του backend του συστήματος βασίζεται στη γλώσσα προγραμματισμού Python, η οποία αποτελεί μία από τις πλέον διαδεδομένες επιλογές για την ανάπτυξη πληροφοριακών συστημάτων, εφαρμογών δικτύου και εργαλείων αυτοματοποίησης. Η ευρεία χρήση της Python στον τομέα της διαχείρισης δικτυακών υποδομών οφείλεται τόσο στη συντακτική της απλότητα όσο και στη διαθεσιμότητα εξειδικευμένων βιβλιοθηκών που υποστηρίζουν πρωτόκολλα και μηχανισμούς παρακολούθησης δικτύων, όπως το SNMP και το SSH.

Στο πλαίσιο της παρούσας εφαρμογής, η Python επιλέγεται ως κεντρική γλώσσα υλοποίησης της επιχειρησιακής λογικής, καθώς επιτρέπει την εύκολη ενσωμάτωση μηχανισμών επικοινωνίας με δικτυακές συσκευές, την υλοποίηση αυτοματοποιημένων ελέγχων διαθεσιμότητας και την επεξεργασία δεδομένων απόδοσης. Επιπλέον, η δυναμική φύση της γλώσσας και η υποστήριξη αντικειμενοστραφούς προγραμματισμού διευκολύνουν τη μοντελοποίηση των βασικών οντοτήτων του συστήματος, όπως οι χρήστες, οι δικτυακές συσκευές και οι μετρήσεις λειτουργίας.

Για την ανάπτυξη του διαδικτυακού επιπέδου του backend επιλέχθηκε το πλαίσιο FastAPI, ένα σύγχρονο web framework για τη δημιουργία RESTful υπηρεσιών υψηλής απόδοσης. Το FastAPI βασίζεται στη χρήση ασύγχρονου προγραμματισμού (asynchronous programming) και αξιοποιεί τον μηχανισμό `async/await` της Python, επιτρέποντας την ταυτόχρονη επεξεργασία πολλαπλών αιτημάτων χωρίς την ανάγκη δημιουργίας πολλαπλών νημάτων εκτέλεσης. Το χαρακτηριστικό αυτό είναι ιδιαίτερα σημαντικό σε εφαρμογές παρακολούθησης δικτύων, όπου απαιτείται συχνή επικοινωνία με μεγάλο αριθμό συσκευών και ταυτόχρονη εξυπηρέτηση πολλών χρηστών.

Η ασύγχρονη αρχιτεκτονική του FastAPI επιτρέπει στο σύστημα να πραγματοποιεί ελέγχους διαθεσιμότητας, συλλογή μετρικών και ανταλλαγή δεδομένων σε πραγματικό χρόνο, χωρίς να επηρεάζεται η συνολική απόδοση της εφαρμογής. Με τον τρόπο αυτό, διασφαλίζεται ότι οι λειτουργίες παρακολούθησης εκτελούνται αποδοτικά, ακόμη και σε περιβάλλοντα με αυξημένο φόρτο εργασίας.

Ένα ακόμη σημαντικό πλεονέκτημα του FastAPI είναι η εγγενής υποστήριξη για την ανάπτυξη καλά δομημένων RESTful APIs. Μέσω των endpoints του backend υλοποιείται η επικοινωνία μεταξύ του frontend και της βάσης δεδομένων, καθώς και η διαχείριση κρίσιμων λειτουργιών, όπως η εγγραφή και τροποποίηση δικτυακών συσκευών, η ανάκτηση δεδομένων κατάστασης και η διαχείριση χρηστών και δικαιωμάτων πρόσβασης. Ο σαφής διαχωρισμός των endpoints από την επιχειρησιακή λογική συμβάλλει στη δημιουργία ενός καθαρού και επεκτάσιμου κώδικα.

Το FastAPI παρέχει επίσης ενσωματωμένη υποστήριξη για την αυτόματη παραγωγή τεκμηρίωσης των APIs μέσω των προτύπων OpenAPI και Swagger. Η δυνατότητα αυτή διευκολύνει σημαντικά τη διαδικασία δοκιμών, την κατανόηση της λειτουργίας του συστήματος και τη μελλοντική του συντήρηση. Παράλληλα, επιτρέπει τη χρήση εργαλείων όπως το Postman για τον έλεγχο της ορθής λειτουργίας των API endpoints.

Τέλος, η επιλογή του συνδυασμού Python και FastAPI συμβάλλει στη δημιουργία ενός backend που χαρακτηρίζεται από ευελιξία, υψηλή απόδοση και δυνατότητα κλιμάκωσης. Η

αρχιτεκτονική αυτή επιτρέπει την εύκολη ενσωμάτωση επιπλέον λειτουργιών, όπως μηχανισμοί ειδοποιήσεων, background εργασίες και υποστήριξη real-time επικοινωνίας, καθιστώντας το σύστημα κατάλληλο για μελλοντικές επεκτάσεις και βελτιώσεις.

2.4.2 Frontend- React.js

Το επίπεδο παρουσίασης της εφαρμογής υλοποιείται με τη βιβλιοθήκη React.js, η οποία αποτελεί μία από τις πιο διαδεδομένες τεχνολογίες για την ανάπτυξη σύγχρονων διαδικτυακών διεπαφών χρήστη. Η React.js βασίζεται στη φιλοσοφία της ανάπτυξης εφαρμογών μέσω επαναχρησιμοποιήσιμων components, επιτρέποντας τη δημιουργία modular, επεκτάσιμων και ευέλικτων διεπαφών, στοιχείο ιδιαίτερα σημαντικό για εφαρμογές με σύνθετη λειτουργικότητα, όπως τα συστήματα παρακολούθησης δικτύων.

Στο πλαίσιο της παρούσας εφαρμογής, η χρήση της React.js επιτρέπει τη δυναμική απεικόνιση μεγάλου όγκου δεδομένων που προέρχονται από το backend, όπως η κατάσταση των δικτυακών συσκευών, οι μετρικές απόδοσης και οι ειδοποιήσεις. Μέσω του μηχανισμού του Virtual DOM, η React ενημερώνει επιλεκτικά μόνο τα τμήματα της διεπαφής που μεταβάλλονται, βελτιώνοντας σημαντικά την απόδοση και εξασφαλίζοντας ομαλή εμπειρία χρήστη ακόμη και σε περιβάλλοντα με συχνές ανανεώσεις δεδομένων.

Η αρχιτεκτονική της εφαρμογής στο frontend βασίζεται σε διακριτά components, τα οποία αντιστοιχούν σε βασικές λειτουργικές ενότητες του συστήματος, όπως το κεντρικό dashboard, οι λίστες δικτυακών συσκευών, οι σελίδες λεπτομερειών συσκευών και οι φόρμες διαχείρισης χρηστών. Ο διαχωρισμός αυτός διευκολύνει τη συντήρηση του κώδικα και επιτρέπει την ανεξάρτητη εξέλιξη κάθε τμήματος της διεπαφής χωρίς να επηρεάζεται η συνολική λειτουργία της εφαρμογής.

Ιδιαίτερη έμφαση δίνεται στη δημιουργία ενός ενοποιημένου dashboard, μέσω του οποίου ο χρήστης αποκτά συνολική εικόνα της κατάστασης του δικτύου. Η React.js επιτρέπει την άμεση ενημέρωση των στοιχείων του dashboard, όπως δείκτες διαθεσιμότητας, γραφήματα απόδοσης και ειδοποιήσεις, χωρίς την ανάγκη ανανέωσης της σελίδας. Η δυνατότητα αυτή είναι κρίσιμη για εφαρμογές παρακολούθησης σε πραγματικό χρόνο, όπου η έγκαιρη πληροφόρηση παίζει καθοριστικό ρόλο.

Παράλληλα, η React.js διευκολύνει την ενσωμάτωση βιβλιοθηκών οπτικοποίησης δεδομένων, όπως οι Chart.js και D3.js, οι οποίες χρησιμοποιούνται για την απεικόνιση μετρικών απόδοσης και ιστορικών δεδομένων. Μέσω διαδραστικών γραφημάτων και διαγραμμάτων, ο χρήστης μπορεί να αναλύει τη συμπεριφορά των δικτυακών συσκευών και να εντοπίζει τάσεις ή ανωμαλίες στη λειτουργία τους.

Τέλος, η επιλογή της React.js υποστηρίζει την ανάπτυξη responsive διεπαφών, διασφαλίζοντας τη σωστή λειτουργία της εφαρμογής σε διαφορετικές συσκευές και μεγέθη οθόνης. Η ευελιξία αυτή καθιστά το σύστημα προσβάσιμο τόσο από σταθερούς υπολογιστές όσο και από φορητές συσκευές, ενισχύοντας τη χρηστικότητα και τη λειτουργικότητά του σε πραγματικά περιβάλλοντα χρήσης.

2.4.3 Βάση Δεδομένων - PostgreSQL

Για την αποθήκευση και διαχείριση των δεδομένων της εφαρμογής επιλέχθηκε το σύστημα διαχείρισης σχεσιακών βάσεων δεδομένων PostgreSQL. Η PostgreSQL αποτελεί μία ώριμη, αξιόπιστη και ιδιαίτερα ισχυρή λύση, η οποία χρησιμοποιείται ευρέως σε πληροφοριακά συστήματα με αυξημένες απαιτήσεις σε ακεραιότητα δεδομένων, ασφάλεια και απόδοση. Η επιλογή της κρίνεται κατάλληλη για την παρούσα εφαρμογή, καθώς υποστηρίζει πολύπλοκες δομές δεδομένων και σύνθετες σχέσεις μεταξύ οντοτήτων, χαρακτηριστικά απαραίτητα σε ένα σύστημα διαχείρισης δικτυακών συσκευών.

Η σχεσιακή φύση της PostgreSQL διευκολύνει τη μοντελοποίηση των βασικών οντοτήτων του συστήματος, όπως οι χρήστες, οι ρόλοι, οι δικτυακές συσκευές και οι συσχετίσεις μεταξύ τους. Μέσω κατάλληλων πινάκων και σχέσεων, διασφαλίζεται η ακεραιότητα των δεδομένων και η αποφυγή πλεονασμών, ενώ παράλληλα επιτρέπεται η αποδοτική ανάκτηση πληροφορίας μέσω σύνθετων ερωτημάτων (queries).

Στη βάση δεδομένων αποθηκεύονται δεδομένα που αφορούν:

- στοιχεία χρηστών και μηχανισμούς αυθεντικοποίησης,
- ρόλους και επίπεδα πρόσβασης (role-based access control),
- καταχωρημένες δικτυακές συσκευές με τα βασικά χαρακτηριστικά τους,
- ρυθμίσεις επικοινωνίας και παρακολούθησης,
- ιστορικά δεδομένα κατάστασης και λειτουργίας.

Η PostgreSQL προσφέρει προηγμένους μηχανισμούς ακεραιότητας, όπως primary keys, foreign keys και constraints, οι οποίοι διασφαλίζουν τη συνέπεια των δεδομένων ακόμη και σε περιπτώσεις ταυτόχρονων προσβάσεων. Οι μηχανισμοί αυτοί είναι ιδιαίτερα σημαντικοί σε περιβάλλοντα όπου πολλοί χρήστες αλληλεπιδρούν ταυτόχρονα με το σύστημα και πραγματοποιούνται συχνές εγγραφές και ενημερώσεις δεδομένων.

Επιπλέον, η PostgreSQL υποστηρίζει αποδοτικούς μηχανισμούς ευρετηρίασης (indexing), οι οποίοι συμβάλλουν στη βελτίωση της απόδοσης κατά την ανάκτηση δεδομένων. Η δυνατότητα αυτή είναι κρίσιμη για τη λειτουργία του dashboard, όπου απαιτείται άμεση πρόσβαση σε δεδομένα κατάστασης και ιστορικές μετρήσεις. Μέσω της σωστής σχεδίασης των ευρετηρίων, επιτυγχάνεται ταχεία απόκριση ακόμη και σε αυξημένο όγκο δεδομένων.

Ένα ακόμη πλεονέκτημα της PostgreSQL είναι η υποστήριξη συναλλαγών (transactions) και η συμμόρφωση με το πρότυπο ACID (Atomicity, Consistency, Isolation, Durability). Η ιδιότητα αυτή διασφαλίζει ότι οι λειτουργίες εισαγωγής, ενημέρωσης και διαγραφής

δεδομένων εκτελούνται με ασφάλεια και αξιοπιστία, μειώνοντας τον κίνδυνο απώλειας ή αλλοίωσης πληροφορίας.

Τέλος, η PostgreSQL ενσωματώνεται ομαλά με το backend της εφαρμογής μέσω της Python, αξιοποιώντας κατάλληλα εργαλεία και βιβλιοθήκες διασύνδεσης. Η ευελιξία αυτή επιτρέπει την περαιτέρω επέκταση της βάσης δεδομένων στο μέλλον, όπως την αποθήκευση επιπλέον ιστορικών δεδομένων ή τη βελτιστοποίηση των μηχανισμών ανάλυσης, χωρίς να απαιτούνται ριζικές αλλαγές στην αρχιτεκτονική του συστήματος.

2.4.4 Real Time Monitoring - WebSockets

Για την υλοποίηση της επικοινωνίας σε πραγματικό χρόνο μεταξύ του backend και του frontend της εφαρμογής επιλέχθηκε η χρήση της τεχνολογίας WebSockets. Τα WebSockets επιτρέπουν τη δημιουργία μιας μόνιμης, αμφίδρομης σύνδεσης μεταξύ client και server, σε αντίθεση με το παραδοσιακό μοντέλο HTTP, το οποίο βασίζεται σε αιτήματα και απαντήσεις. Η συγκεκριμένη προσέγγιση κρίνεται απαραίτητη σε ένα σύστημα παρακολούθησης δικτυακών συσκευών, όπου η άμεση ενημέρωση της κατάστασης αποτελεί βασική λειτουργική απαίτηση.

Μέσω των WebSockets, το backend έχει τη δυνατότητα να αποστέλλει δεδομένα στο frontend τη στιγμή που αυτά παράγονται ή μεταβάλλονται, χωρίς να απαιτείται συνεχής αποστολή αιτημάτων από την πλευρά του χρήστη (polling). Με τον τρόπο αυτό, μειώνεται σημαντικά ο φόρτος του δικτύου και βελτιώνεται η συνολική απόδοση της εφαρμογής, ιδιαίτερα σε περιβάλλοντα με μεγάλο αριθμό συσκευών και συχνές μεταβολές κατάστασης.

Στο πλαίσιο της εφαρμογής, τα WebSockets χρησιμοποιούνται για την άμεση ενημέρωση του dashboard σχετικά με την κατάσταση των δικτυακών συσκευών, όπως αλλαγές διαθεσιμότητας (online/offline), μεταβολές σε βασικές μετρικές απόδοσης (CPU, μνήμη, δικτυακή κίνηση) και την εμφάνιση ειδοποιήσεων. Η πληροφορία αυτή προβάλλεται σε πραγματικό χρόνο μέσω γραφημάτων και οπτικών ενδείξεων, επιτρέποντας στους χρήστες να αποκτούν άμεση εικόνα της κατάστασης του δικτύου.

Η ενσωμάτωση των WebSockets με το FastAPI πραγματοποιείται μέσω εξειδικευμένων endpoints, τα οποία διαχειρίζονται τις ενεργές συνδέσεις και την αποστολή δεδομένων προς τους συνδεδεμένους clients. Στην πλευρά του frontend, η React.js αξιοποιεί τους WebSocket clients για τη λήψη και δυναμική απεικόνιση των δεδομένων, χωρίς την ανάγκη ανανέωσης της σελίδας. Η αρχιτεκτονική αυτή συμβάλλει στην παροχή μιας ομαλής και άμεσης εμπειρίας χρήστη.

Επιπλέον, η χρήση WebSockets επιτρέπει την υλοποίηση πιο προηγμένων λειτουργιών, όπως η αποστολή στοχευμένων ενημερώσεων σε συγκεκριμένους χρήστες ή ομάδες χρηστών, καθώς και η υποστήριξη μηχανισμών ειδοποίησης με βάση τη σοβαρότητα των συμβάντων. Η δυνατότητα αυτή ενισχύει την ευελιξία και την επεκτασιμότητα του συστήματος, καθιστώντας το κατάλληλο για μελλοντικές επεκτάσεις και αυξημένες απαιτήσεις παρακολούθησης.

2.4.5 Ασύγχρονες Διεργασίες Υποβάθρου - Python Asyncio και Redis

Για την υλοποίηση λειτουργιών που απαιτούν εκτέλεση στο υπόβαθρο και δεν πρέπει να επιβαρύνουν τον κύριο κύκλο λειτουργίας της εφαρμογής (request-response cycle), επιλέχθηκε η χρήση των ασύγχρονων δυνατοτήτων της βιβλιοθήκης Asyncio της Python σε συνδυασμό με το Redis. Η συγκεκριμένη προσέγγιση επιτρέπει την ασύγχρονη και παράλληλη επεξεργασία εργασιών, συμβάλλοντας καθοριστικά στη διατήρηση της υψηλής απόκρισης και της σταθερότητας του συστήματος, ειδικά κατά την εκτέλεση χρονοβόρων ή επαναλαμβανόμενων διαδικασιών.

Στο πλαίσιο ενός συστήματος παρακολούθησης δικτυακών συσκευών, πολλές λειτουργίες, όπως ο περιοδικός έλεγχος διαθεσιμότητας των συσκευών (ping) και η συλλογή μετρικών απόδοσης μέσω SNMP ή SSH, είναι χρονοβόρες και εξαρτώνται από την απόκριση του δικτύου. Εάν οι διαδικασίες αυτές εκτελούνταν συγχρονισμένα κατά την υποβολή αιτημάτων από τους χρήστες, θα προκαλούσαν σημαντικές καθυστερήσεις στη διεπαφή (frontend) και πιθανά σφάλματα λήξης χρόνου (timeouts). Η μεταφορά αυτών των λειτουργιών στο υπόβαθρο εξασφαλίζει ότι το API παραμένει άμεσα διαθέσιμο και ομαλό στη χρήση.

Η διαχείριση των εργασιών υποβάθρου βασίζεται σε έναν ενσωματωμένο ασύγχρονο χρονοπρογραμματιστή (scheduler), ο οποίος εκκινεί μαζί με την εφαρμογή και εκτελείται συνεχώς στο παρασκήνιο. Ο χρονοπρογραμματιστής ανακτά δυναμικά τις ρυθμίσεις του συστήματος για να καθορίσει τη συχνότητα ελέγχου των συσκευών, προσφέροντας ευελιξία στους διαχειριστές. Σε κάθε κύκλο ελέγχου, οι εργασίες για όλες τις ενεργές συσκευές εκτελούνται παράλληλα και όχι διαδοχικά, επιτρέποντας στο σύστημα να κλιμακώνεται αποδοτικά καθώς αυξάνεται ο αριθμός των παρακολουθούμενων κόμβων.

Παράλληλα, το Redis διαδραματίζει διπλό ρόλο στην υποστήριξη αυτών των διεργασιών. Αφενός, λειτουργεί ως message broker για τη διανομή των αποτελεσμάτων σε πραγματικό χρόνο, μεταφέροντας τις ενημερώσεις των συσκευών από το υπόβαθρο στους συνδεδεμένους clients μέσω WebSockets. Αφετέρου, χρησιμοποιείται ως μηχανισμός προσωρινής αποθήκευσης (caching) για τη διατήρηση των απαραίτητων προηγούμενων τιμών (όπως οι μετρητές δικτυακής κίνησης), οι οποίες απαιτούνται για τον υπολογισμό μετρικών σε πραγματικό χρόνο, μειώνοντας έτσι την ανάγκη για συνεχείς αναγνώσεις και εγγραφές στη σχεσιακή βάση δεδομένων.

Η επιλογή της αξιοποίησης των εγγενών (native) ασύγχρονων δυνατοτήτων της Python (Asyncio) έναντι εξωτερικών, πιο σύνθετων συστημάτων διαχείρισης ουράς εργασιών (όπως το Celery), προσφέρει σημαντικά πλεονεκτήματα για το συγκεκριμένο έργο. Απλοποιεί την αρχιτεκτονική του συστήματος, μειώνει τις απαιτήσεις σε υπολογιστικούς πόρους (καθώς δεν απαιτούνται επιπλέον containers ή workers) και διευκολύνει τη συντήρηση και την

εγκατάσταση της εφαρμογής, διασφαλίζοντας παράλληλα τη βέλτιστη απόδοση και την ομαλή λειτουργία της.

2.4.6 Περιβάλλον Ανάπτυξης Εφαρμογής - Visual Studio Code

Το Visual Studio Code (VS Code) αποτελεί το κύριο περιβάλλον ανάπτυξης που επιλέχθηκε για την υλοποίηση της εφαρμογής. Πρόκειται για ένα ελαφρύ, αλλά εξαιρετικά ισχυρό εργαλείο προγραμματισμού, το οποίο υποστηρίζει πλήρως τις γλώσσες Python και JavaScript, απαραίτητες για το backend (FastAPI) και το frontend (React.js) αντίστοιχα. Η αρχιτεκτονική του VS Code βασίζεται σε μονάδες (modules) και προσθέματα (extensions), τα οποία μπορούν να εγκατασταθούν για την επέκταση των δυνατοτήτων του IDE, καλύπτοντας ανάγκες debugging, linting, testing, version control και αυτοματοποίησης εργασιών.

Ένα από τα σημαντικότερα χαρακτηριστικά του VS Code είναι η έξυπνη συμπλήρωση κώδικα (IntelliSense), η οποία παρέχει προτάσεις για μεταβλητές, συναρτήσεις και modules με βάση τον τρέχοντα κώδικα, μειώνοντας σημαντικά τα πιθανά σφάλματα και επιταχύνοντας την ανάπτυξη. Η ενσωμάτωση του debugging επιτρέπει τη ρύθμιση breakpoints, την παρακολούθηση μεταβλητών σε πραγματικό χρόνο και την εκτέλεση του κώδικα βήμα-βήμα, διευκολύνοντας τον εντοπισμό και την επίλυση προβλημάτων.

Η δυνατότητα ενσωμάτωσης με σύστημα ελέγχου έκδοσης Git επιτρέπει την παρακολούθηση αλλαγών, τη δημιουργία branches για την ανάπτυξη νέων λειτουργιών και την ασφαλή συγχώνευση κώδικα μέσω pull requests. Επιπλέον, το ενσωματωμένο terminal προσφέρει τη δυνατότητα εκτέλεσης εντολών χωρίς την ανάγκη εξωτερικών εφαρμογών, διευκολύνοντας την καθημερινή διαχείριση του έργου.

Το VS Code υποστηρίζει επίσης multi-root workspaces, επιτρέποντας την ταυτόχρονη εργασία σε πολλαπλά έργα ή modules της εφαρμογής, γεγονός που είναι ιδιαίτερα χρήσιμο σε σύνθετα συστήματα όπως αυτό της παρακολούθησης δικτυακών συσκευών. Η διαχείριση των extensions για linting, formatting και testing εξασφαλίζει ότι η ποιότητα του κώδικα διατηρείται υψηλή, ενώ η υποστήριξη για snippets, templates και code refactoring ενισχύει την παραγωγικότητα των προγραμματιστών.

Τέλος, η δυνατότητα παραμετροποίησης του περιβάλλοντος εργασίας, η υποστήριξη για live share sessions για συνεργατική ανάπτυξη και η ευρεία κοινότητα χρηστών και διαθέσιμων extensions καθιστούν το VS Code ένα ιδανικό εργαλείο για την ανάπτυξη σύγχρονων, πολυεπίπεδων εφαρμογών με υψηλές απαιτήσεις σε αξιοπιστία, απόδοση και ευχρηστία.

2.4.7 Version Control - Git και Github

Το Git είναι ένα κατακευματισμένο σύστημα ελέγχου έκδοσης (version control system) που χρησιμοποιείται για την παρακολούθηση των αλλαγών στον πηγαίο κώδικα της εφαρμογής. Κάθε αλλαγή στον κώδικα καταγράφεται με μοναδικό αναγνωριστικό (commit),

επιτρέποντας την αναλυτική παρακολούθηση της εξέλιξης του έργου, την ανάκτηση προηγούμενων εκδόσεων και την αποφυγή μόνιμης απώλειας δεδομένων. Η λειτουργία των branches επιτρέπει την ανάπτυξη νέων λειτουργιών ή τη διόρθωση σφαλμάτων ανεξάρτητα από τον κύριο κώδικα, ενώ τα merges επιτρέπουν την ομαλή ενσωμάτωση των αλλαγών σε έναν κοινό κλάδο, διασφαλίζοντας την ομαλή συνεργασία μεταξύ των προγραμματιστών.

Το GitHub επεκτείνει τις δυνατότητες του Git, προσφέροντας απομακρυσμένη αποθήκευση του έργου και εργαλεία για την οργάνωση και διαχείριση της συνεργασίας μεταξύ των μελών της ομάδας. Μέσω της πλατφόρμας, δημιουργούνται pull requests, τα οποία διευκολύνουν την αναθεώρηση και επικύρωση του κώδικα πριν την ενσωμάτωσή του στον κύριο κλάδο, μειώνοντας τον κίνδυνο σφαλμάτων και βελτιώνοντας την ποιότητα του λογισμικού. Παράλληλα, το GitHub υποστηρίζει issue tracking για την καταγραφή και παρακολούθηση σφαλμάτων ή νέων απαιτήσεων, καθώς και projects και boards για τη διαχείριση εργασιών, εξασφαλίζοντας σαφήνεια και οργάνωση σε κάθε στάδιο της ανάπτυξης.

Επιπλέον, η ενσωμάτωση με CI/CD pipelines μέσω GitHub Actions επιτρέπει την αυτοματοποίηση διαδικασιών όπως unit testing, builds και deployment, διασφαλίζοντας ότι οι αλλαγές στον κώδικα παράγουν σταθερές και αξιόπιστες εκδόσεις της εφαρμογής. Η δυνατότητα συνεργασίας σε πραγματικό χρόνο, η ιστορική παρακολούθηση του έργου και η ευκολία πρόσβασης από οποιαδήποτε τοποθεσία καθιστούν το Git σε συνδυασμό με το GitHub ένα απαραίτητο εργαλείο για τη διαχείριση μεγάλων και σύνθετων έργων λογισμικού, όπως ένα σύστημα παρακολούθησης και διαχείρισης δικτυακών συσκευών.

2.4.8 Containerization - Docker

Το Docker είναι μια πλατφόρμα που επιτρέπει τη δημιουργία, ανάπτυξη και εκτέλεση εφαρμογών μέσα σε containers, τα οποία λειτουργούν ως ελαφριά, απομονωμένα περιβάλλοντα. Κάθε container περιέχει όλα τα απαραίτητα στοιχεία για την εκτέλεση της εφαρμογής: κώδικα, βιβλιοθήκες, εξαρτήσεις και ρυθμίσεις, εξασφαλίζοντας ότι η εφαρμογή τρέχει με τον ίδιο τρόπο ανεξάρτητα από το περιβάλλον στο οποίο αναπτύσσεται (ανάπτυξη, δοκιμές ή παραγωγή). Αυτή η απομόνωση μειώνει σημαντικά τα προβλήματα συμβατότητας που συχνά εμφανίζονται σε παραδοσιακές μεθόδους ανάπτυξης.

Στην υλοποίηση της εφαρμογής παρακολούθησης και διαχείρισης δικτυακών συσκευών, το Docker χρησιμοποιείται για την απομόνωση των κύριων στοιχείων του συστήματος, όπως το backend με FastAPI, το frontend με React.js, τη βάση δεδομένων PostgreSQL, το Redis για caching και task queue, καθώς και τους Asyncio workers που εκτελούν background tasks. Κάθε component εκτελείται σε ξεχωριστό container, επιτρέποντας την ανεξάρτητη ανάπτυξη, δοκιμή και κλιμάκωση.

Η χρήση του Docker Compose διευκολύνει τη συγκρότηση των containers σε ένα ολοκληρωμένο σύστημα, καθορίζοντας τις σχέσεις και τις εξαρτήσεις μεταξύ τους. Μέσω του Compose, η εκκίνηση όλων των απαραίτητων υπηρεσιών γίνεται με μία μόνο εντολή, μειώνοντας σημαντικά το χρόνο ανάπτυξης και το κόστος ρυθμίσεων. Επιπλέον, η containerization επιτρέπει την εύκολη αναπαραγωγή του περιβάλλοντος εργασίας,

διευκολύνοντας τη συνεργασία μεταξύ προγραμματιστών και διασφαλίζοντας ότι η εφαρμογή θα τρέχει με σταθερή απόδοση σε οποιοδήποτε σύστημα ή server.

Επιπλέον, το Docker ενισχύει την ασφάλεια και τη διαχείριση πόρων. Η απομόνωση των containers περιορίζει την επιρροή ενός τυχόν σφάλματος σε μία υπηρεσία, χωρίς να επηρεάζεται ολόκληρο το σύστημα. Η φορητότητα, η δυνατότητα κλιμάκωσης και η συμβατότητα με cloud περιβάλλοντα καθιστούν το Docker εργαλείο κλειδί για την ανάπτυξη σύγχρονων web-based εφαρμογών υψηλής απόδοσης και αξιοπιστίας.

2.4.9 API Testing - Postman

Το Postman αποτελεί ένα ολοκληρωμένο εργαλείο που χρησιμοποιείται για τη δημιουργία, δοκιμή και διαχείριση RESTful APIs, τα οποία αποτελούν βασικό κομμάτι της εφαρμογής web-based παρακολούθησης και διαχείρισης δικτυακών συσκευών. Μέσω του Postman, μπορούν να ελεγχθούν οι λειτουργίες του backend, όπως οι ενέργειες δημιουργίας, ανάγνωσης, ενημέρωσης και διαγραφής (CRUD) δεδομένων για χρήστες, συσκευές ή ρυθμίσεις συστήματος, χωρίς να απαιτείται η εκτέλεση της πλήρους frontend εφαρμογής.

Το εργαλείο επιτρέπει τη δημιουργία συλλογών (collections) εντολών API, οι οποίες μπορούν να οργανωθούν με τρόπο που να αναπαριστούν διαφορετικά σενάρια χρήσης του συστήματος. Μέσω αυτών των συλλογών, γίνεται δυνατή η αυτοματοποίηση των δοκιμών (automated testing), η επαναληπτική εκτέλεση σε διαφορετικά περιβάλλοντα (development, staging, production) και η διασφάλιση της ορθής λειτουργίας των endpoints σε όλες τις φάσεις ανάπτυξης. Επιπλέον, το Postman παρέχει δυνατότητες logging, debugging και παρακολούθησης αιτήσεων και απαντήσεων, επιτρέποντας την ανάλυση του χρόνου απόκρισης, την επαλήθευση της δομής των δεδομένων και την απομόνωση τυχόν σφαλμάτων.

Μέσω του Postman, μπορεί να πραγματοποιηθεί έλεγχος τόσο για βασικές λειτουργίες όσο και για πιο σύνθετες διαδικασίες, όπως η διαχείριση authentication tokens, η επικύρωση δικαιωμάτων πρόσβασης, καθώς και η διαχείριση παραμέτρων και φίλτρων αναζήτησης σε πραγματικά σενάρια χρήσης. Η δυνατότητα δημιουργίας mock servers δίνει τη δυνατότητα προσομοίωσης endpoints πριν αυτά υλοποιηθούν πλήρως, διευκολύνοντας το frontend development παράλληλα με το backend.

Η χρήση του Postman βελτιώνει σημαντικά την αξιοπιστία, την ποιότητα και την ταχύτητα ανάπτυξης της εφαρμογής, καθώς επιτρέπει στον προγραμματιστή να εντοπίζει σφάλματα και να επαληθεύει τη σωστή λειτουργία του API σε κάθε στάδιο, προτού οι αλλαγές ενσωματωθούν στο κύριο σύστημα.

3. Απαιτήσεις και Σχεδιασμός

Στο κεφάλαιο αυτό παρουσιάζονται οι απαιτήσεις και ο σχεδιασμός του συστήματος παρακολούθησης και διαχείρισης δικτυακών συσκευών. Καταγράφονται οι λειτουργικές και μη λειτουργικές απαιτήσεις που πρέπει να καλύπτει η εφαρμογή, προσδιορίζονται οι χρήστες και οι ρόλοι τους, καθώς και τα σενάρια χρήσης τους. Παράλληλα, γίνεται ανάλυση της αρχιτεκτονικής του συστήματος, περιγράφοντας τη δομή του backend, frontend, της βάσης δεδομένων και των υπηρεσιών υποβάθρου. Τέλος, παρουσιάζεται η λογική οργάνωση των δεδομένων και η μοντελοποίηση των δικτυακών οντοτήτων, θέτοντας τη βάση για την υλοποίηση και τη λειτουργία του συστήματος.

3.1 Λειτουργικές Απαιτήσεις

Οι λειτουργικές απαιτήσεις καθορίζουν τις βασικές δυνατότητες και λειτουργίες που πρέπει να παρέχει το σύστημα για την ορθή και αποτελεσματική διαχείριση των δικτυακών συσκευών. Οι απαιτήσεις αυτές περιγράφουν με σαφήνεια τις διεπαφές χρήστη, τις ενέργειες που μπορούν να εκτελεστούν από τους διαχειριστές και τους τελικούς χρήστες, καθώς και την αλληλεπίδραση του συστήματος με τα δεδομένα και τις συσκευές. Μέσω της σαφούς καταγραφής των λειτουργικών απαιτήσεων εξασφαλίζεται η πληρότητα της εφαρμογής, η βελτιστοποίηση της εμπειρίας των χρηστών και η ομαλή λειτουργία του συστήματος σε καθημερινές συνθήκες χρήσης.

Διαχείριση Συσκευών

- Το σύστημα θα πρέπει να επιτρέπει στον admin ή/και operator να καταχωρεί νέες δικτυακές συσκευές με πλήρη στοιχεία, όπως IP address, MAC address, τύπο συσκευής, τοποθεσία, κατασκευαστή και μοντέλο.
- Το σύστημα θα πρέπει να επιτρέπει την κατηγοριοποίηση των συσκευών ανά τύπο (router, switch, server, IoT device), ανά δίκτυο ή ανά τμήμα της επιχείρησης.
- Το σύστημα θα πρέπει να επιτρέπει την επεξεργασία των στοιχείων μιας συσκευής, ώστε να ενημερώνονται πληροφορίες όπως IP, MAC, τοποθεσία ή ρόλος συσκευής.
- Το σύστημα θα πρέπει να επιτρέπει τη διαγραφή μιας συσκευής, με ασφαλή επιβεβαίωση για την αποφυγή τυχαίας διαγραφής.
- Το σύστημα θα πρέπει να επιτρέπει την ομαδοποίηση συσκευών για μαζικές ενέργειες, όπως αλλαγή ρυθμίσεων, reboot ή ενημέρωση firmware.
- Το σύστημα θα πρέπει να υποστηρίζει τη διαχείριση credentials και πρωτοκόλλων επικοινωνίας, όπως SNMP, SSH και HTTP/HTTPS, με ασφαλή αποθήκευση κωδικών

και κλειδιών.

- Το σύστημα θα πρέπει να εμφανίζει λίστα όλων των καταχωρημένων συσκευών σε πίνακα, με δυνατότητα φιλτραρίσματος, ταξινόμησης και αναζήτησης με βάση οποιοδήποτε πεδίο.
- Το σύστημα θα πρέπει να παρέχει πληροφορίες κατάστασης σε πραγματικό χρόνο για κάθε συσκευή (online/offline, uptime, CPU/memory usage, network traffic).
- Το σύστημα θα πρέπει να επιτρέπει την επισήμανση συσκευών με προτεραιότητα ή κατάσταση (π.χ. κρίσιμη, προτεραιότητα χαμηλή) ώστε να διευκολύνεται η διαχείριση των πιο σημαντικών στοιχείων.
- Το σύστημα θα πρέπει να επιτρέπει την προβολή ιστορικού αλλαγών για κάθε συσκευή, καταγράφοντας ποιος χρήστης έκανε την αλλαγή, τι αλλαγή έγινε και πότε.
- Το σύστημα θα πρέπει να υποστηρίζει bulk import/export συσκευών μέσω αρχείων CSV ή JSON, για εύκολη αρχικοποίηση ή ενημέρωση του συστήματος.
- Το σύστημα θα πρέπει να παρέχει δυνατότητα δημιουργίας report ανά συσκευή ή ομάδα συσκευών, περιλαμβάνοντας κατάσταση, uptime, μετρικές απόδοσης και alerts.

Real-time Παρακολούθηση και Ειδοποιήσεις

- Το σύστημα θα πρέπει να παρακολουθεί σε πραγματικό χρόνο τη διαθεσιμότητα όλων των δικτυακών συσκευών μέσω πρωτοκόλλων όπως SNMP, ICMP και SSH.
- Το σύστημα θα πρέπει να εμφανίζει σε dashboard την κατάσταση κάθε συσκευής (online, offline) με χρωματική απεικόνιση και εύκολα κατανοητά εικονίδια.
- Το σύστημα θα πρέπει να καταγράφει μετρήσεις απόδοσης για κάθε συσκευή, όπως CPU usage, memory utilization, network traffic, latency και uptime.
- Το σύστημα θα πρέπει να εντοπίζει και να ενημερώνει άμεσα για αλλαγές κατάστασης ή ανωμαλίες στις συσκευές.
- Το σύστημα θα πρέπει να επιτρέπει στον διαχειριστή να δημιουργεί κανόνες ειδοποίησης (alerts) βάσει thresholds για κρίσιμες μετρικές, όπως υπερβολική χρήση CPU ή χαμηλή διαθεσιμότητα.
- Το σύστημα θα πρέπει να υποστηρίζει πολλαπλά κανάλια ειδοποιήσεων, όπως email, web notifications και webhook integrations με άλλα εργαλεία.
- Το σύστημα θα πρέπει να παρέχει δυνατότητα διαβαθμισμένων ειδοποιήσεων (escalation policies), ώστε πιο σοβαρά προβλήματα να ειδοποιούνται άμεσα σε

περισσότερα επίπεδα χρήστη.

- Το σύστημα θα πρέπει να διατηρεί ιστορικό alerts και συμβάντων για αναλύσεις, audit και μελλοντική βελτιστοποίηση.
- Το σύστημα θα πρέπει να επιτρέπει στον διαχειριστή να προσαρμόζει τη συχνότητα παρακολούθησης και polling ανά τύπο συσκευής ή δίκτυο.
- Το σύστημα θα πρέπει να εμφανίζει γραφήματα με ιστορικά δεδομένα, τάσεις απόδοσης και πληρότητα συσκευών, ώστε να διευκολύνεται η λήψη αποφάσεων.
- Το σύστημα θα πρέπει να υποστηρίζει την αυτοματοποίηση ενεργειών μετά από alert, όπως επανεκκίνηση συσκευής ή ειδοποίηση τρίτου εργαλείου διαχείρισης.

Διαχείριση Χρηστών και Δικαιωμάτων

- Το σύστημα θα πρέπει να υποστηρίζει πολλαπλούς χρήστες με διαφορετικά επίπεδα πρόσβασης (admin, operator, viewer).
- Το σύστημα θα πρέπει να επιτρέπει στον διαχειριστή να δημιουργεί νέους χρήστες με συγκεκριμένο ρόλο και προσαρμοσμένα δικαιώματα.
- Το σύστημα θα πρέπει να επιτρέπει την επεξεργασία στοιχείων χρήστη, όπως όνομα, επώνυμο, email, τηλέφωνο, ρόλο και κατάσταση λογαριασμού (ενεργός/ανενεργός).
- Το σύστημα θα πρέπει να παρέχει δυνατότητα προσωρινής ή μόνιμης απενεργοποίησης χρήστη.
- Το σύστημα θα πρέπει να υποστηρίζει δυνατότητες ανάθεσης δικαιωμάτων πρόσβασης σε συγκεκριμένες συσκευές, τμήματα δικτύου ή λειτουργίες του dashboard.
- Το σύστημα θα πρέπει να καταγράφει όλες τις ενέργειες των χρηστών για λόγους audit και ασφάλειας.
- Το σύστημα θα πρέπει να υποστηρίζει μηχανισμούς ασφαλούς authentication και authorization, όπως password hashing και token-based authentication (JWT).
- Το σύστημα θα πρέπει να επιτρέπει στους διαχειριστές να επαναφέρουν κωδικούς χρηστών ή να δημιουργούν προσωρινά credentials για συγκεκριμένες ενέργειες.
- Το σύστημα θα πρέπει να υποστηρίζει τη δημιουργία ομάδων χρηστών για ευκολότερη διαχείριση δικαιωμάτων σε μαζικό επίπεδο.
- Το σύστημα θα πρέπει να επιτρέπει την ειδοποίηση των χρηστών για αλλαγές στα δικαιώματα ή για σημαντικά γεγονότα ασφαλείας.

Αρχιτεκτονική Δεδομένων και Διαχείριση Βάσης Δεδομένων

- Το σύστημα θα πρέπει να χρησιμοποιεί σχεσιακή βάση δεδομένων (PostgreSQL) για την αποθήκευση χρηστών, συσκευών και ιστορικών δεδομένων.
- Το σύστημα θα πρέπει να υποστηρίζει ομαδοποίηση και κατηγοριοποίηση των συσκευών ανά τύπο, τοποθεσία, δίκτυο και λειτουργικό ρόλο.
- Το σύστημα θα πρέπει να επιτρέπει τη διαχείριση credentials και πρωτοκόλλων επικοινωνίας (SNMP, SSH, HTTP/HTTPS) για κάθε συσκευή.
- Το σύστημα θα πρέπει να υποστηρίζει ιστορικότητα δεδομένων, ώστε να είναι δυνατή η ανάλυση τάσεων απόδοσης, uptime και incidents σε βάθος χρόνου.
- Το σύστημα θα πρέπει να παρέχει δυνατότητα δημιουργίας backup και restore της βάσης δεδομένων.
- Το σύστημα θα πρέπει να επιτρέπει εύκολη ανάκτηση και αναζήτηση δεδομένων με βάση πολλαπλά κριτήρια, όπως όνομα συσκευής, τύπος, IP, κατάσταση ή ημερομηνία καταγραφής.
- Το σύστημα θα πρέπει να επιτρέπει την παρακολούθηση σε πραγματικό χρόνο των αλλαγών στα δεδομένα, όπως η προσθήκη νέων συσκευών ή η ενημέρωση των στοιχείων χρηστών.
- Το σύστημα θα πρέπει να υποστηρίζει την εξαγωγή δεδομένων σε δημοφιλείς μορφές (CSV, JSON) για αναφορές και ανάλυση.
- Το σύστημα θα πρέπει να εξασφαλίζει την ακεραιότητα των δεδομένων και την αποφυγή συγκρούσεων κατά την ταυτόχρονη πρόσβαση πολλών χρηστών.
- Το σύστημα θα πρέπει να επιτρέπει την εύκολη επεκτασιμότητα της βάσης δεδομένων για προσθήκη νέων τύπων συσκευών, επιπλέον πεδίων ή νέων λειτουργιών στο μέλλον.

Λειτουργικό Dashboard και Παρακολούθηση Συσκευών

- Το σύστημα θα πρέπει να παρέχει ένα κεντρικό dashboard που θα εμφανίζει συνοπτική εικόνα όλων των καταχωρημένων συσκευών και της κατάστασής τους σε πραγματικό χρόνο.
- Το dashboard θα πρέπει να επιτρέπει την προβολή μετρικών απόδοσης ανά συσκευή, όπως uptime, CPU usage, memory utilization, network traffic και latency.

- Το σύστημα θα πρέπει να εμφανίζει visual indicators (π.χ. χρώματα, icons) για γρήγορη αναγνώριση προβλημάτων ή offline συσκευών.
- Το dashboard θα πρέπει να παρέχει δυνατότητα φιλτραρίσματος συσκευών με βάση τύπο, τοποθεσία, κατάσταση ή δίκτυο.
- Το σύστημα θα πρέπει να επιτρέπει την προβολή ιστορικών δεδομένων και τάσεων απόδοσης σε γραφήματα (line charts, bar charts).
- Το σύστημα θα πρέπει να επιτρέπει στον διαχειριστή να ορίζει κανόνες ειδοποιήσεων (thresholds) για κρίσιμες παραμέτρους κάθε συσκευής.
- Το σύστημα θα πρέπει να παρέχει ειδοποιήσεις σε πολλαπλά κανάλια (email, web notifications, webhook) σε περίπτωση υπέρβασης των ορίων ή ανίχνευσης σφαλμάτων.
- Το dashboard θα πρέπει να επιτρέπει τη δυναμική ενημέρωση των δεδομένων σε πραγματικό χρόνο μέσω WebSockets, χωρίς την ανάγκη ανανέωσης της σελίδας.
- Το σύστημα θα πρέπει να επιτρέπει την άμεση μετάβαση από κάθε συσκευή στο λεπτομερές ιστορικό της, με αναλυτικά logs, alerts και στατιστικά.
- Το σύστημα θα πρέπει να παρέχει επιλογές export των δεδομένων σε μορφές CSV ή JSON για αναφορές και ανάλυση.

3.2 Μη Λειτουργικές Απαιτήσεις

Οι μη λειτουργικές απαιτήσεις καθορίζουν τα ποιοτικά χαρακτηριστικά του συστήματος και περιγράφουν τα κριτήρια με βάση τα οποία θα αξιολογηθεί η συνολική του απόδοση, αξιοπιστία και καταλληλότητα για χρήση σε πραγματικές επιχειρησιακές συνθήκες. Σε αντίθεση με τις λειτουργικές απαιτήσεις, οι οποίες επικεντρώνονται στις συγκεκριμένες δυνατότητες που πρέπει να προσφέρει η εφαρμογή, οι μη λειτουργικές απαιτήσεις προσδιορίζουν τον τρόπο με τον οποίο το σύστημα οφείλει να λειτουργεί, τις προδιαγραφές που πρέπει να πληρεί και τα πρότυπα που πρέπει να ακολουθεί.

Απόδοση & Κλιμάκωση

- Το σύστημα θα πρέπει να υποστηρίζει ταυτόχρονη χρήση από πολλαπλούς χρήστες χωρίς σημαντική υποβάθμιση των επιδόσεων.
- Ο χρόνος απόκρισης των κύριων λειτουργιών (όπως προβολή συσκευών, φόρτωση dashboard, εμφάνιση μετρικών) δεν θα πρέπει να υπερβαίνει τα 3 δευτερόλεπτα υπό κανονικές συνθήκες φόρτου.

- Το σύστημα θα πρέπει να μπορεί να υποστηρίξει τουλάχιστον 100 καταγεγραμμένες συσκευές χωρίς ανάγκη αλλαγής στην αρχιτεκτονική ή ανασχεδιασμό του backend.
- Η διαδικασία συλλογής δεδομένων (polling) θα πρέπει να εκτελείται σε καθορισμένα intervals (10–60 δευτερόλεπτα) χωρίς να επηρεάζει τη διαθεσιμότητα του UI.
- Η υποδομή θα πρέπει να υποστηρίζει μελλοντική κλιμάκωση είτε κατακόρυφη (ενίσχυση πόρων server) είτε οριζόντια (πολλαπλά instances με load balancing).

Ασφάλεια

- Όλες οι επικοινωνίες μεταξύ frontend και backend θα πρέπει να πραγματοποιούνται μέσω HTTPS για την αποφυγή υποκλοπής δεδομένων.
- Τα credentials των συσκευών (π.χ. SNMP community strings, SSH passwords/keys) θα πρέπει να αποθηκεύονται κρυπτογραφημένα στη βάση δεδομένων με χρήση κατάλληλων αλγορίθμων (π.χ. AES-256 ή ισοδύναμο).
- Οι κωδικοί χρηστών θα πρέπει να αποθηκεύονται με password hashing (π.χ. bcrypt, Argon2).
- Η πρόσβαση στα δεδομένα θα πρέπει να ελέγχεται με μηχανισμούς authentication & authorization (JWT tokens).
- Το σύστημα θα πρέπει να υποστηρίζει role-based access control (RBAC) και, εφόσον απαιτηθεί, multi-factor authentication.
- Θα πρέπει να εφαρμόζονται μηχανισμοί προστασίας από συνηθισμένες επιθέσεις web εφαρμογών (SQL injection, XSS, CSRF, brute-force).
- Το σύστημα θα πρέπει να καταγράφει ύποπτες δραστηριότητες και να ειδοποιεί τον διαχειριστή για πιθανές παραβιάσεις ασφαλείας.

Διαθεσιμότητα & Αξιοπιστία

- Το σύστημα θα πρέπει να είναι διαθέσιμο τουλάχιστον στο 99% του χρόνου για τους χρήστες.
- Η αρχιτεκτονική θα πρέπει να επιτρέπει την εκτέλεση background εργασιών (monitoring, alerts, polling) ανεξάρτητα από τη χρήση του UI.
- Η αποτυχία μεμονωμένων υπηρεσιών (π.χ. Asyncio worker ή WebSocket) δεν θα πρέπει να καθιστά μη λειτουργική όλη την εφαρμογή.

- Το σύστημα θα πρέπει να υποστηρίζει διαδικασίες ανάκτησης από σφάλματα (fault tolerance) και επανεκκίνηση services χωρίς απώλεια δεδομένων.

Συντηρησιμότητα & Επεκτασιμότητα

- Ο πηγαίος κώδικας θα πρέπει να είναι οργανωμένος με βάση αρχές σύγχρονης ανάπτυξης (modular architecture, separation of concerns).
- Η εφαρμογή θα πρέπει να υποστηρίζει εύκολη προσθήκη νέων λειτουργιών χωρίς ανακατασκευή του υπάρχοντος κώδικα.
- Θα πρέπει να χρησιμοποιείται version control (Git/GitHub) με τεκμηριωμένη στρατηγική branches (feature branches, reviews, pull requests).
- Ο κώδικας θα πρέπει να συνοδεύεται από σχόλια και τεχνική τεκμηρίωση για την ευκολότερη παρακολούθηση και αλλαγή του.
- Η χρήση containers (Docker) θα πρέπει να καθιστά εύκολη την ανάπτυξη σε νέες υποδομές ή cloud providers.

Χρηστικότητα & Εμπειρία Χρήστη

- Η διεπαφή χρήστη θα πρέπει να είναι φιλική, κατανοητή και λειτουργική για χρήστες με βασικές τεχνικές γνώσεις.
- Όλες οι κύριες λειτουργίες (π.χ. προσθήκη συσκευής, δημιουργία κανόνων ειδοποίησης, προβολή dashboard) θα πρέπει να είναι προσβάσιμες σε λιγότερα από τρία βήματα.
- Θα πρέπει να χρησιμοποιούνται διεθνώς αναγνωρίσιμα πρότυπα UI/UX (χρωματική κωδικοποίηση, εικονίδια κατάσταση, tooltips).
- Η εφαρμογή θα πρέπει να υποστηρίζει responsive σχεδιασμό για χρήση σε υπολογιστές και tablets.
- Οι ειδοποιήσεις θα πρέπει να παρέχονται με τρόπο μη ενοχλητικό, αλλά ορατό και κατανοητό από τον χρήστη.

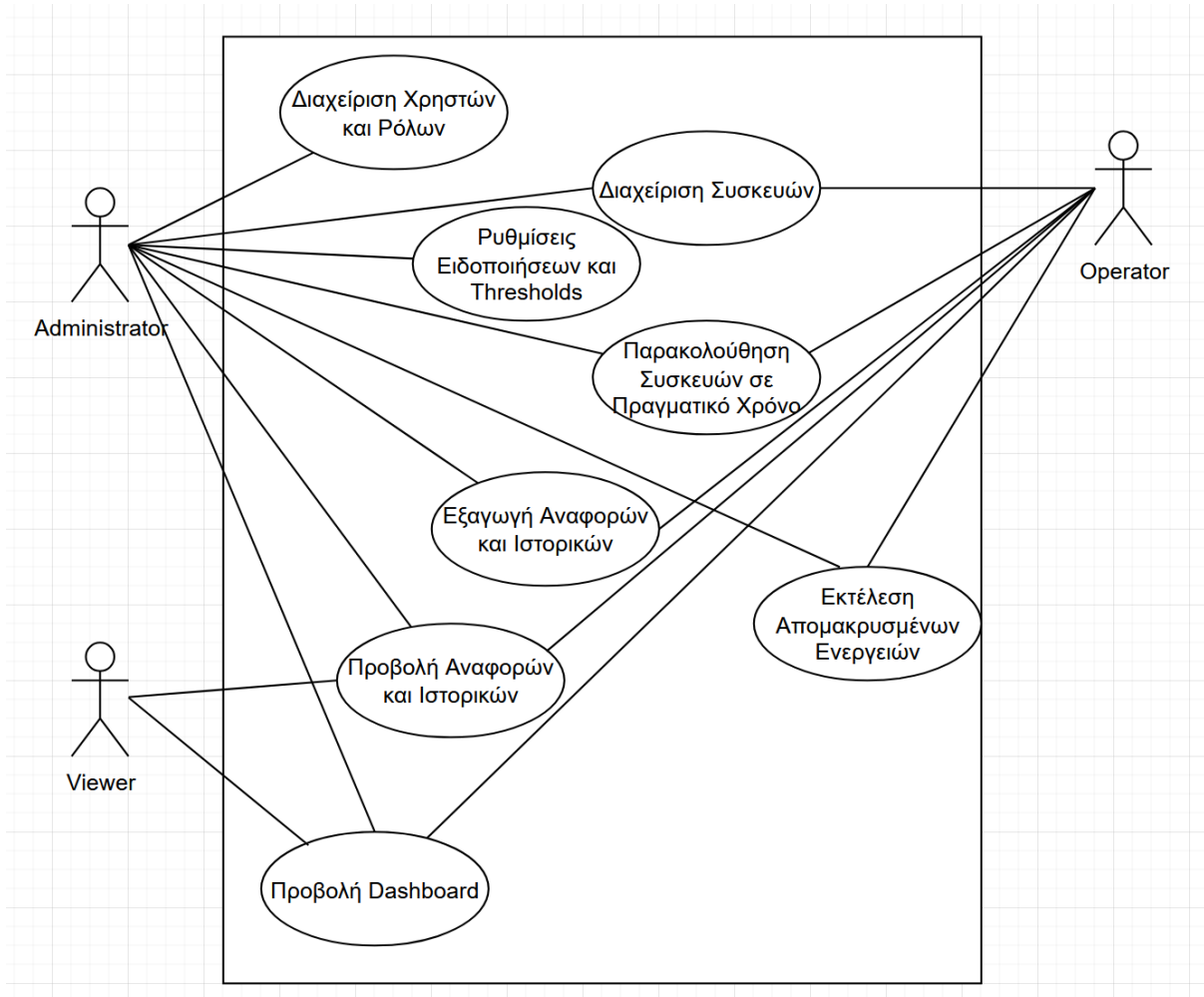
Συμβατότητα & Διαλειτουργικότητα

- Το σύστημα θα πρέπει να λειτουργεί σε όλους τους σύγχρονους browsers (Chrome, Firefox, Edge, Safari).
- Θα πρέπει να υποστηρίζονται συσκευές από διαφορετικούς κατασκευαστές (Cisco, Mikrotik, Ubiquiti, HP, Dell, IoT συσκευές κ.λπ.), εφόσον ακολουθούν κοινά πρότυπα πρωτοκόλλων SNMP/SSH/ICMP.
- Το backend θα πρέπει να είναι ανεξάρτητο από το υπόβαθρο λειτουργικού συστήματος (Linux/Windows), υπό την προϋπόθεση ότι υπάρχουν διαθέσιμα τα απαραίτητα dependencies.
- Το σύστημα θα πρέπει να υποστηρίζει ενσωμάτωση με τρίτες υπηρεσίες μέσω APIs ή webhooks, επιτρέποντας μελλοντικές επεκτάσεις.

3.3 Περιπτώσεις Χρήσης

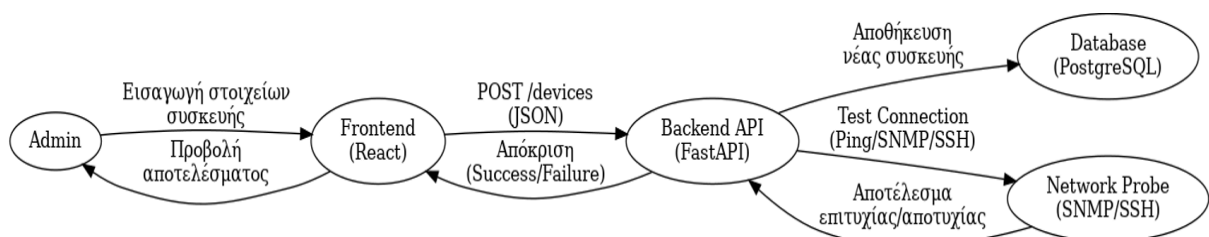
Στην εφαρμογή ορίζονται τρεις βασικοί ρόλοι χρηστών:

- **Διαχειριστής (Administrator):** Πλήρης πρόσβαση σε όλα τα δεδομένα, τις ρυθμίσεις και τις λειτουργίες του συστήματος.
- **Χειριστής (Operator):** Διαχείριση και παρακολούθηση συσκευών, αλλά με περιορισμένη πρόσβαση στις ρυθμίσεις συστήματος και στους χρήστες.
- **Παρατηρητής (Viewer):** Πρόσβαση σε δεδομένα παρακολούθησης, χωρίς δυνατότητα τροποποίησης ή επέμβασης.



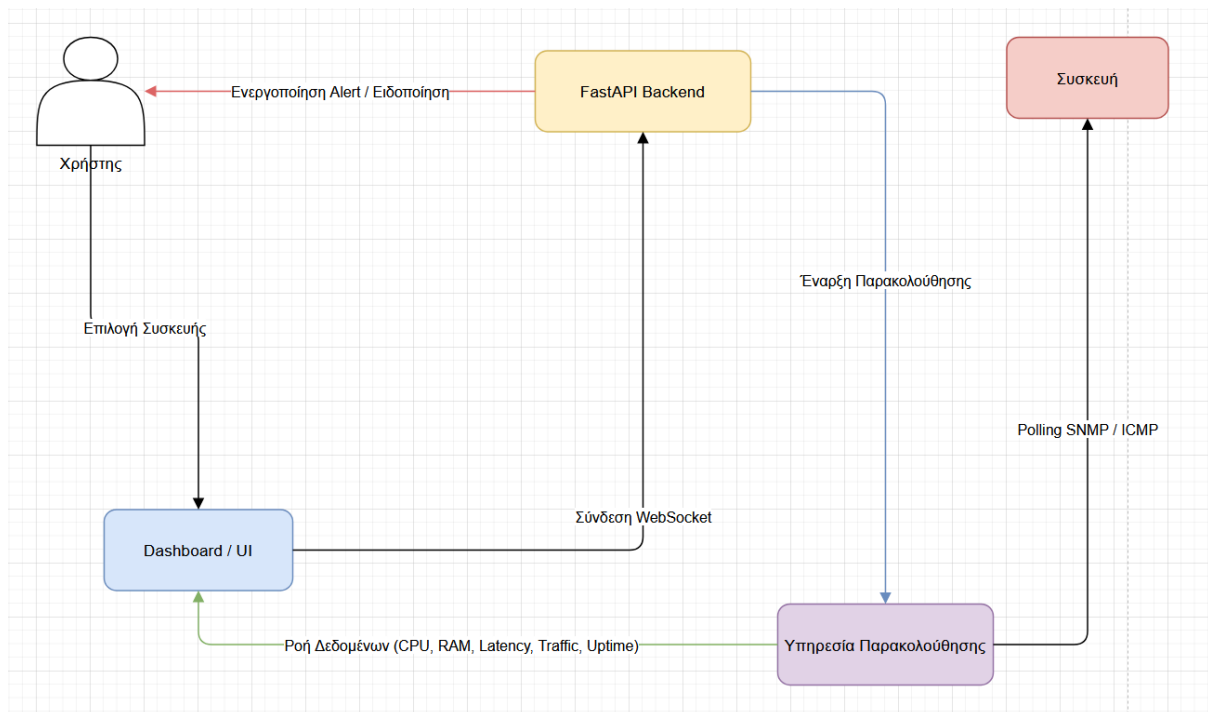
Σχήμα 1 Διάγραμμα Περιπτώσεων Χρήσης ανά Ρόλο Χρήστη

Το διάγραμμα περίπτωσης χρήσης για την προσθήκη νέας συσκευής παρουσιάζει τη διαδικασία καταχώρησης μιας συσκευής στο σύστημα από τον διαχειριστή. Ο χρήστης εισάγει τα στοιχεία της συσκευής μέσω της διεπαφής React και υποβάλλει το αίτημα στο backend (FastAPI), όπου πραγματοποιείται έλεγχος εγκυρότητας και αποθήκευση των δεδομένων στη βάση PostgreSQL. Στη συνέχεια, το backend εκτελεί αρχικό έλεγχο συνδεσιμότητας μέσω πρωτοκόλλων όπως SNMP ή ping για να διαπιστώσει την κατάσταση της συσκευής. Τέλος, το σύστημα επιστρέφει ενημερωμένη απάντηση στο frontend, το οποίο εμφανίζει το αποτέλεσμα και ενημερώνει το dashboard.



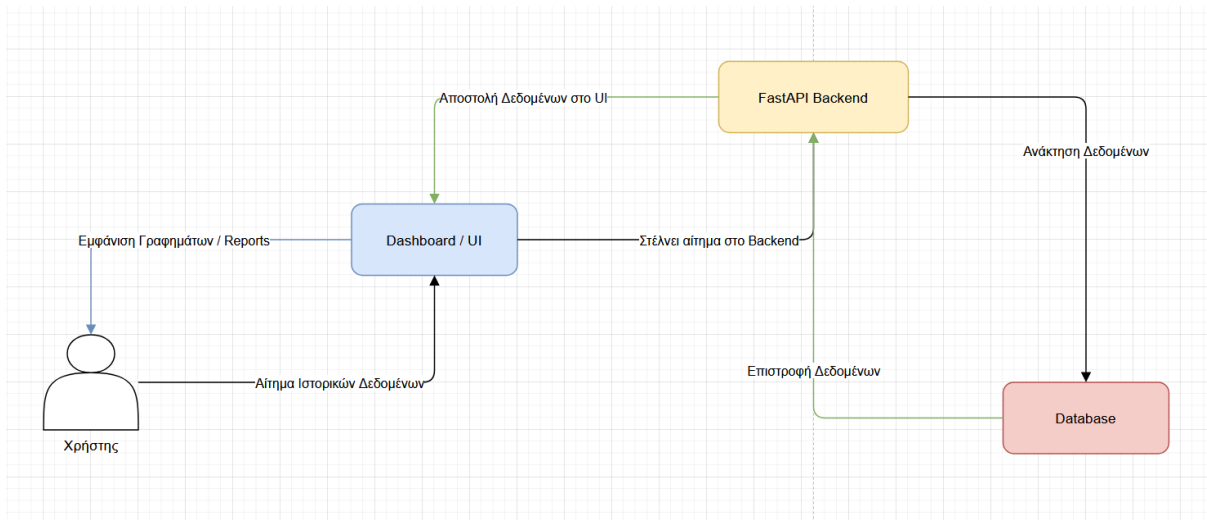
Σχήμα 2 Διάγραμμα Περίπτωσης Χρήσης για Προσθήκη Νέας Συσκευής

Το διάγραμμα περίπτωσης χρήσης για την παρακολούθηση συσκευής σε πραγματικό χρόνο αφορά τη συνεχή συλλογή και εμφάνιση κρίσιμων μετρικών από μία καταγεγραμμένη δικτυακή συσκευή. Ο χρήστης επιλέγει συσκευή από το dashboard και η εφαρμογή εγκαθιστά WebSocket σύνδεση με το backend. Το σύστημα πραγματοποιεί επαναλαμβανόμενα αιτήματα SNMP/ICMP για ανάκτηση δεδομένων (CPU, RAM, latency, uptime, traffic) και μεταδίδει τις τιμές άμεσα στο UI με τη μορφή γεγονότων streaming. Η ροή δεδομένων ενημερώνει δυναμικά γραφήματα, ενδείξεις κατάστασης και οπτικούς δείκτες. Επιπλέον, πραγματοποιείται έλεγχος κανόνων ειδοποίησης (thresholds), ενεργοποίηση alerts και αποστολή ενημερώσεων στον χρήστη.



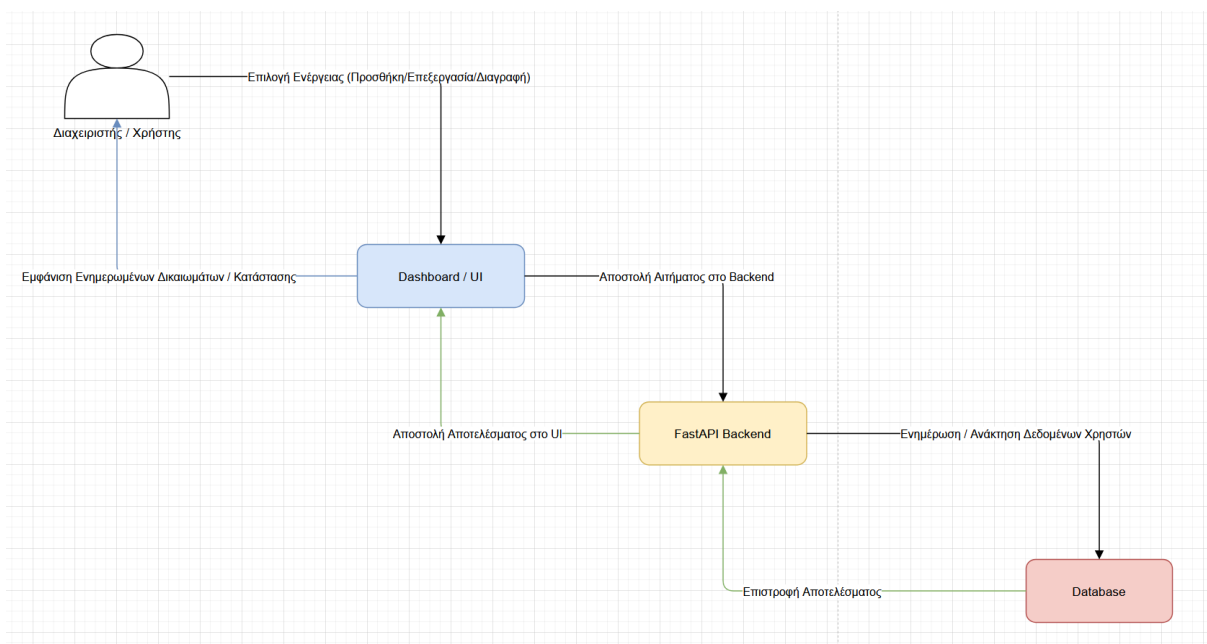
Σχήμα 3 Διάγραμμα Περίπτωσης Χρήσης για Παρακολούθηση Συσκευής σε Πραγματικό Χρόνο

Το διάγραμμα περίπτωσης χρήσης για ιστορικά δεδομένα επιτρέπει στον χρήστη να ανακτά, να προβάλλει και να αναλύει προηγούμενες μετρήσεις των συσκευών του. Μέσω του dashboard, ο χρήστης επιλέγει χρονικό διάστημα ή φίλτρα για τις μετρήσεις (CPU, RAM, latency, traffic, uptime) και το UI στέλνει αίτημα στο backend. Το backend ανακτά τα δεδομένα από τη βάση δεδομένων και τα επιστρέφει στο UI, όπου εμφανίζονται δυναμικά σε γραφήματα, πίνακες ή αναφορές.



Σχήμα 4 Διάγραμμα Περίπτωσης Χρήσης για Ιστορικά Δεδομένα

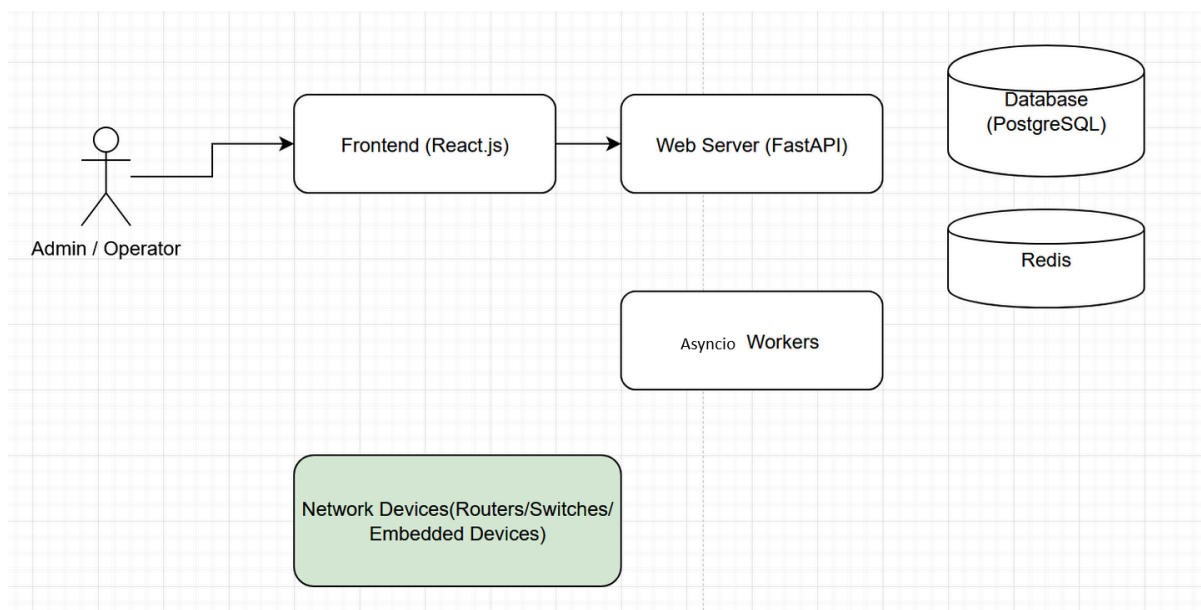
Το διάγραμμα περίπτωσης χρήσης για την διαχείριση χρηστών και δικαιωμάτων αφορά τη δημιουργία, επεξεργασία και διαγραφή λογαριασμών χρηστών, καθώς και την εκχώρηση κατάλληλων δικαιωμάτων πρόσβασης στο σύστημα. Ο διαχειριστής μπορεί να προσθέτει νέους χρήστες, να ορίζει ρόλους (π.χ. admin, viewer, operator) και να τροποποιεί τα δικαιώματα ανάλογα με τις ανάγκες. Το backend επεξεργάζεται τις αλλαγές στη βάση δεδομένων και ενημερώνει το UI ώστε οι διαθέσιμες λειτουργίες και η ορατότητα στοιχείων να αντιστοιχούν στα δικαιώματα κάθε χρήστη, διασφαλίζοντας ασφαλή και ελεγχόμενη πρόσβαση στο σύστημα.



Σχήμα 5 Διάγραμμα Περίπτωσης Χρήσης Χρηστών και Δικαιωμάτων

Το παρακάτω διάγραμμα απεικονίζει την υποδομή της εφαρμογής και τη ροή δεδομένων μεταξύ των επιμέρους στοιχείων. Συνοπτικά:

- Ο Admin/Operator αντιπροσωπεύει τον χρήστη που διαχειρίζεται και παρακολουθεί τις συσκευές. Αλληλεπιδρά με το Frontend (React.js), που αποτελεί την διεπαφή χρήστη.
- Το frontend επικοινωνεί με τον Web Server (FastAPI), ο οποίος χειρίζεται τα αιτήματα, την επεξεργασία δεδομένων και την επικοινωνία με τη βάση δεδομένων.
- Η Database (PostgreSQL) αποθηκεύει όλες τις κρίσιμες πληροφορίες για χρήστες, συσκευές, ιστορικά δεδομένα και ρυθμίσεις του συστήματος.
- Το Redis χρησιμοποιείται για caching δεδομένων και για την υποστήριξη ουράς εργασιών των Asyncio Workers, βελτιώνοντας την απόδοση και την ταχύτητα.
- Οι Asyncio Workers εκτελούν background tasks, όπως επαναλαμβανόμενους ελέγχους συσκευών, συλλογή μετρικών και αποστολή ειδοποιήσεων.
- Τα Network Devices (routers, switches, embedded devices) αντιπροσωπεύουν τις συσκευές που παρακολουθούνται και διαχειρίζονται από το σύστημα. Η επικοινωνία με αυτές γίνεται μέσω πρωτοκόλλων όπως SNMP ή SSH.



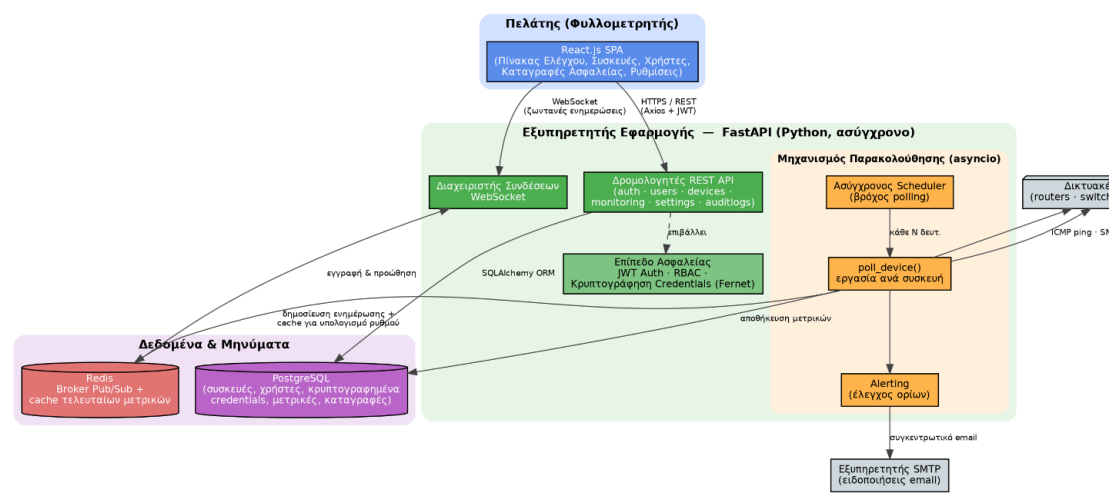
Σχήμα 6 Διάγραμμα Τοπολογίας Δικτύου

4. Ανάπτυξη της Εφαρμογής

Στο κεφάλαιο αυτό παρουσιάζεται αναλυτικά η διαδικασία ανάπτυξης, η δομή του πηγαίου κώδικα και η αρχιτεκτονική του web-based συστήματος παρακολούθησης και διαχείρισης δικτυακών συσκευών. Η εφαρμογή έχει σχεδιαστεί και αναπτυχθεί με βάση τη σύγχρονη φιλοσοφία του διαχωρισμού των επιπέδων (layered architecture) και της πλήρους αποσύνδεσης (decoupling) μεταξύ του επιπέδου παρουσίασης (frontend) και του επιπέδου επιχειρησιακής λογικής (backend).

Συγκεκριμένα, αναλύονται:

- Η δομή και η διάρθρωση του frontend επιπέδου (React.js, Routing, Layouts, Context Providers, CSS Styling).
- Η αρχιτεκτονική του backend (Python, FastAPI, SQLAlchemy ORM, Pydantic schemas) και η μοντελοποίηση των δεδομένων.
- Ο μηχανισμός παρακολούθησης (Monitoring Engine) με την υποστήριξη των πρωτοκόλλων ICMP, SNMP και SSH, καθώς και η ασύγχρονη εκτέλεση background διεργασιών.
- Η επικοινωνία σε πραγματικό χρόνο μέσω της ενοποίησης του Redis Pub/Sub και των WebSockets.
- Η ασφάλεια του συστήματος (JWT authentication, password complexity policies, κρυπτογράφηση credentials με AES-256/Fernet).
- Η κοντεϊνερποίηση (containerization) και η ενορχήστρωση των υπηρεσιών με Docker και Docker Compose.



Εικόνα 4 Αρχιτεκτονική υψηλού επιπέδου του συστήματος

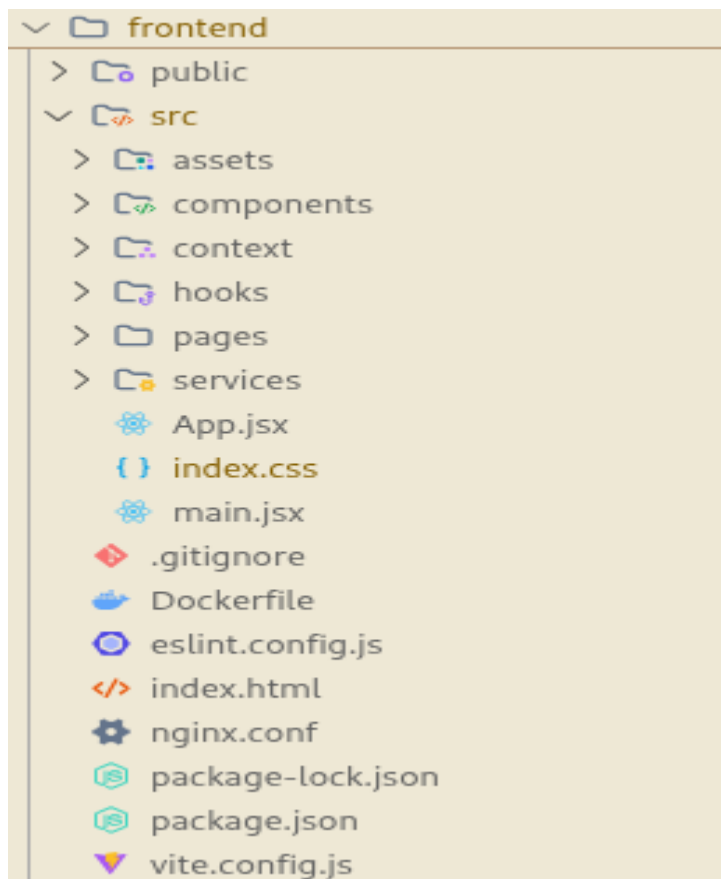
4.1 Δομή και Αρχιτεκτονική του Frontend

Το frontend επίπεδο της εφαρμογής αναπτύχθηκε ως μια εφαρμογή μιας σελίδας (Single Page Application - SPA) με τη χρήση της βιβλιοθήκης React.js. Η προσέγγιση αυτή εξασφαλίζει

εξαιρετικά γρήγορη απόκριση και ομαλή πλοήγηση, καθώς η φόρτωση των δεδομένων και οι αλλαγές στη διεπαφή πραγματοποιούνται δυναμικά μέσω ασύγχρονων κλήσεων API (Axios), χωρίς να απαιτείται η πλήρης ανανέωση (page reload) της σελίδας από τον browser.

4.1.1 Οργάνωση και Δομή του Κώδικα (src/)

Ο πηγαίος κώδικας του frontend βρίσκεται στον κατάλογο /frontend/src και ακολουθεί μια αυστηρά modular δομή:



Εικόνα 4 Δομή καταλόγων και αρχείων του Frontend (React.js)

- **assets/**: Φιλοξενεί στατικά αρχεία, όπως εικόνες και το λογότυπο της εφαρμογής.
- **components/**: Περιλαμβάνει επαναχρησιμοποιήσιμα δομικά στοιχεία της διεπαφής. Το σημαντικότερο component είναι το Layout Layout.jsx, το οποίο ορίζει την κοινή γραφική δομή της εφαρμογής (Sidebar πλοήγησης, Header, κουμπί αλλαγής θέματος και ενδείξεις κατάστασης χρήστη).
- **context/**: Περιέχει τους Context Providers της React για τη διαχείριση της καθολικής κατάστασης (global state). Συγκεκριμένα:
 - **AuthContext.jsx**: Διαχειρίζεται την κατάσταση σύνδεσης (authentication state), αποθηκεύει το JSON Web Token (JWT) στο LocalStorage και εκθέτει πληροφορίες για τον τρέχοντα χρήστη και τον ρόλο του.

- **ThemeContext.jsx**: Διαχειρίζεται τη δυναμική εναλλαγή του θέματος (Light/Dark mode) και αποθηκεύει την προτίμηση του χρήστη στο LocalStorage.
- **hooks/**: Custom React hooks (π.χ. useAuth.js) που διευκολύνουν την πρόσβαση στα contexts χωρίς την ανάγκη επανάληψης boilerplate κώδικα.
- **pages/**: Περιέχει τα views των επιμέρους σελίδων της εφαρμογής:
 - **Login.jsx / Login.css**: Η σελίδα αυθεντικοποίησης των χρηστών.
 - **Dashboard.jsx**: Η κεντρική κονσόλα παρακολούθησης σε πραγματικό χρόνο.
 - **Devices.jsx**: Η σελίδα διαχείρισης του inventory των συσκευών.
 - **Users.jsx**: Η σελίδα διαχείρισης χρηστών (προσβάσιμη μόνο από Admin).
 - **AuditLogs.jsx**: Η σελίδα προβολής των καταγραφών ασφαλείας και ενεργειών.
 - **Settings.jsx**: Η σελίδα ρυθμίσεων συστήματος (range sliders κ.λπ.) και προφίλ.
- **services/**: Περιέχει την κλάση επικοινωνίας με το API services/api.js, η οποία ρυθμίζει το base URL και ενσωματώνει Axios interceptors για την αυτόματη προσθήκη του JWT token στο Authorization header κάθε αιτήματος, καθώς και τη διαχείριση των σφαλμάτων (π.χ. 401 Unauthorized).

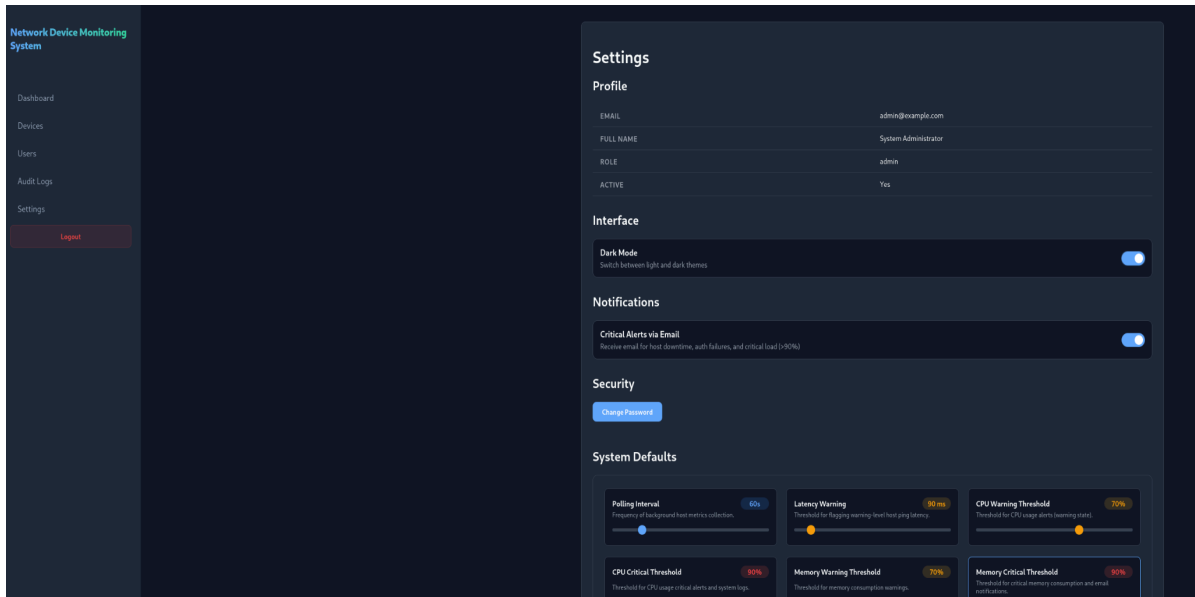
4.1.2 Layouts, Routing και Έλεγχος Πρόσβασης (Route Guards)

Η διαχείριση των διαδρομών (routing) γίνεται μέσω του react-router-dom στο αρχείο frontend/src/App.jsx. Για την προστασία των endpoints του frontend από μη εξουσιοδοτημένους χρήστες, εφαρμόζεται μια δομή Route Guards με βάση τον ρόλο (Role-Based Access Control - RBAC):

1. **Δημόσιες Διαδρομές (Public Routes)**: Η σελίδα Login και η σελίδα ResetPassword είναι προσβάσιμες μόνο από μη συνδεδεμένους χρήστες. Αν ένας συνδεδεμένος χρήστης προσπαθήσει να εισέλθει σε αυτές, ανακατευθύνεται αυτόματα στο Dashboard.
2. **Προστατευμένες Διαδρομές (Protected Routes)**: Σελίδες όπως το Dashboard, η λίστα συσκευών (Devices) και οι ρυθμίσεις προφίλ απαιτούν ενεργή σύνδεση. Αν ένας μη συνδεδεμένος χρήστης επιχειρήσει να τις επισκεφθεί, ανακατευθύνεται στη σελίδα Login.
3. **Διαδρομές Διαχειριστή (Admin-Only Routes)**: Σελίδες όπως η διαχείριση χρηστών (Users), η προβολή των Audit Logs και η αλλαγή των συστημικών defaults (System Settings) απαιτούν ρόλο admin. Ο έλεγχος γίνεται δυναμικά: αν ο ρόλος στο αποκωδικοποιημένο JWT δεν είναι admin, η πρόσβαση αποκλείεται και εμφανίζεται μήνυμα σφάλματος.

4.1.3 Δυναμικό Θέμα (Dynamic Theming & index.css)

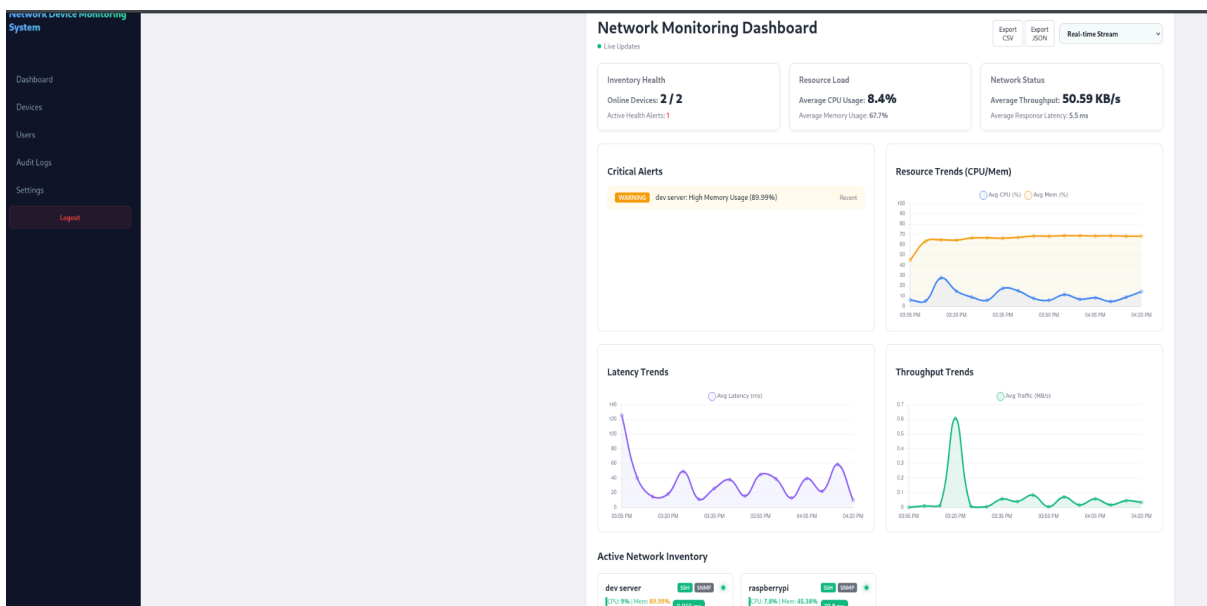
Η εφαρμογή υποστηρίζει πλήρως Light και Dark mode. Η υλοποίηση βασίζεται σε CSS variables στο αρχείο /frontend/src/index.css. Όταν ο χρήστης αλλάζει το θέμα μέσω του διακόπτη (switch) στο Layout, το ThemeContext ενημερώνει το attribute data-theme στο root στοιχείο <html> (document.documentElement.setAttribute('data-theme', theme)). Τα CSS variables (χρώματα φόντου, κειμένου, borders, εισαγωγών) αλλάζουν αυτόματα τιμές, προσφέροντας μια ομαλή μετάβαση χωρίς τρεμόπαιγμα.



Εικόνα 5 Παράδειγμα της εφαρμογής σε Dark Mode

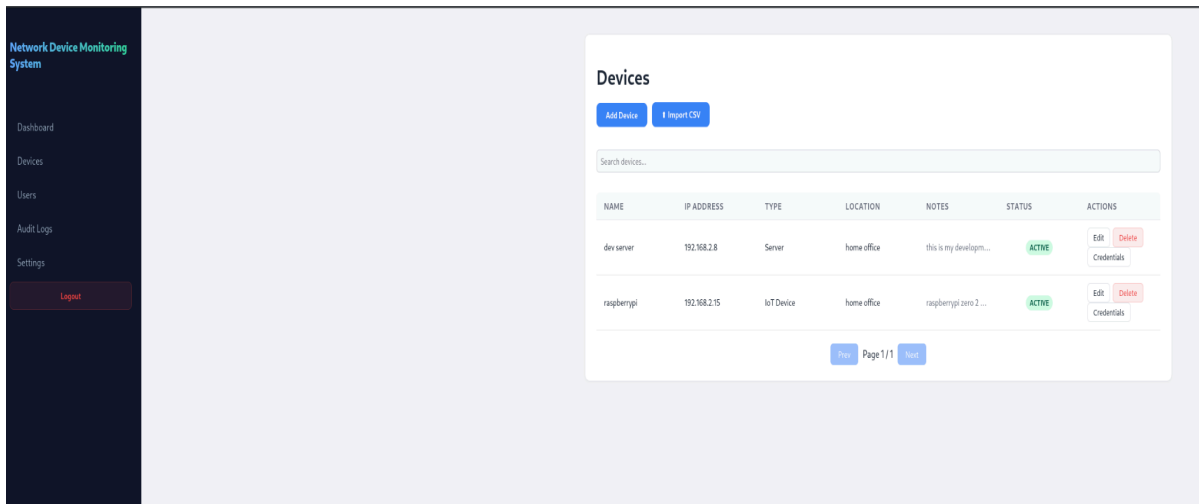
4.1.4 Frontend Σελίδες και Λειτουργικότητα

- **Dashboard /frontend/src/pages/Dashboard.jsx:** Αποτελεί την κεντρική οθόνη του συστήματος. Περιλαμβάνει KPI widgets με συγκεντρωτικά στοιχεία (σύνολο συσκευών, online/offline, critical alerts), διαγράμματα Chart.js για την ιστορική εξέλιξη των μετρικών, και έναν πίνακα ειδοποιήσεων σε πραγματικό χρόνο.



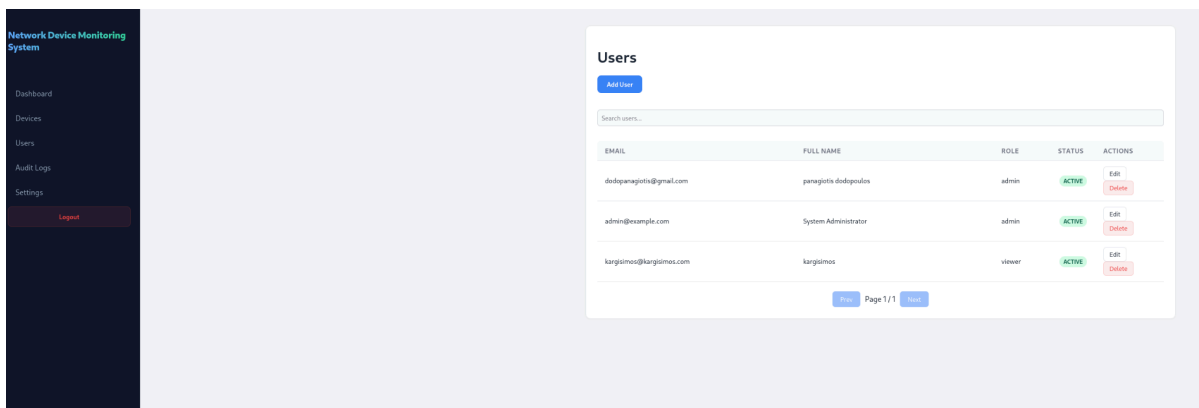
Εικόνα 6 Η διεπαφή του dashboard με τα widgets πραγματικού χρόνου και τα γραφήματα τάσεων

- **Devices /frontend/src/pages/Devices.jsx:** Επιτρέπει την προβολή, αναζήτηση, φιλτράρισμα και διαχείριση των συσκευών. Υποστηρίζει μαζική εισαγωγή συσκευών μέσω CSV αρχείων και εξαγωγή ιστορικών μετρικών σε CSV/JSON.



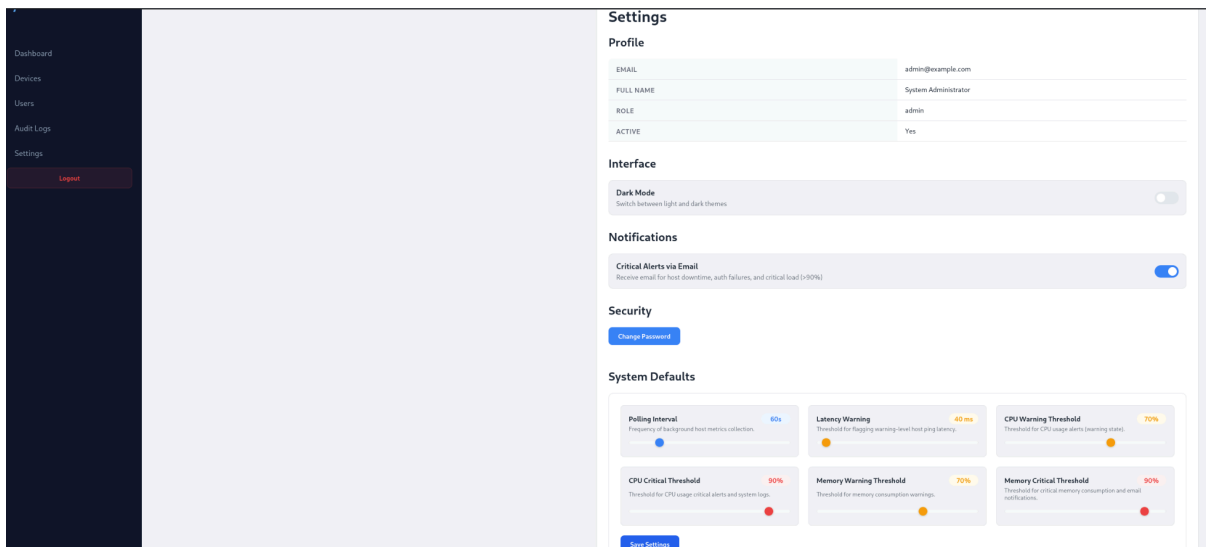
Εικόνα 7 Η σελίδα διαχείρισης συσκευών με τον πίνακα inventory και τις επιλογές import/export

- **Users /frontend/src/pages/Users.jsx:** Παρέχει στον διαχειριστή τη δυνατότητα διαχείρισης των χρηστών (δημιουργία, επεξεργασία ρόλου, ενεργοποίηση/απενεργοποίηση).



Εικόνα 8 Η σελίδα διαχείρισης χρηστών για τους διαχειριστές της εφαρμογής

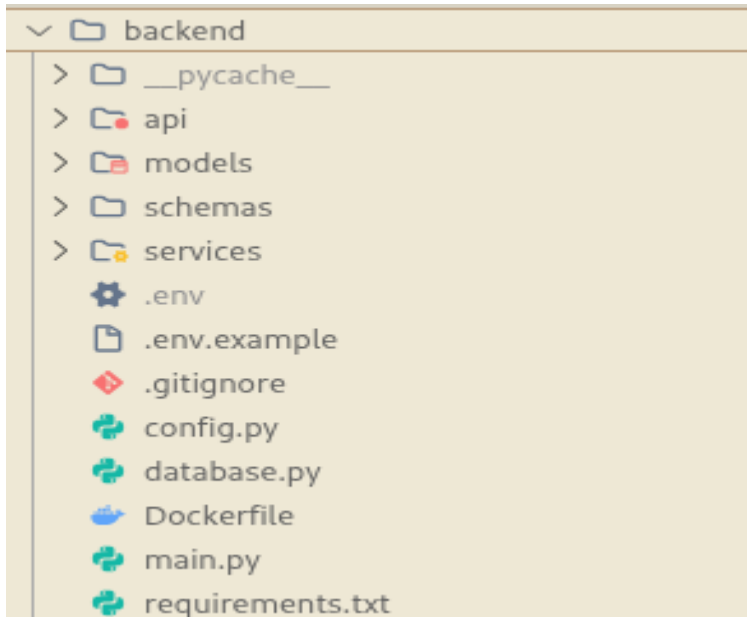
- **Settings /frontend/src/pages/Settings.jsx:** Επιτρέπει στους διαχειριστές να προσαρμόζουν τις προεπιλογές του συστήματος. Για την αναβάθμιση της εμπειρίας χρήσης, τα πεδία εισαγωγής αντικαταστάθηκαν με range sliders (bars) με χρωματική κωδικοποίηση (μπλε για το polling, πορτοκαλί για warning, κόκκινο για critical). Επιπλέον, ενσωματώθηκε client-side validation που εμποδίζει την αποθήκευση αν τα warning thresholds είναι μεγαλύτερα ή ίσα με τα critical thresholds, προειδοποιώντας οπτικά τον χρήστη.



Εικόνα 9 Η σελίδα ρυθμίσεων με τους range sliders για τα CPU, Memory και Latency thresholds

4.2 Backend και Μοντελοποίηση Δεδομένων

Το backend της εφαρμογής αναπτύχθηκε με Python και το framework FastAPI. Η επιλογή αυτή έγινε λόγω της εξαιρετικής ταχύτητας εκτέλεσης του FastAPI, της αυτόματης δημιουργίας Swagger documentation, και της εγγενούς υποστήριξης ασύγχρονων λειτουργιών (async/await), η οποία επιτρέπει τη διαχείριση πολλαπλών I/O-bound εργασιών (όπως η επικοινωνία με δικτυακές συσκευές) με ελάχιστους πόρους συστήματος.



Εικόνα 10 Δομή καταλόγων και αρχείων του Backend (Python)

4.2.1 Βάση Δεδομένων και Object-Relational Mapping (SQLAlchemy)

Για την αποθήκευση των δεδομένων χρησιμοποιείται PostgreSQL. Η επικοινωνία και ο χειρισμός της βάσης γίνονται μέσω του SQLAlchemy ORM. Στο αρχείο `/backend/database.py` ορίζεται η σύνδεση με τη βάση, το engine και το SessionLocal για τη δημιουργία database sessions.

Η αρχικοποίηση του σχήματος της βάσης δεδομένων (schema initialization) γίνεται αυτόματα κατά την εκκίνηση του server στο `/backend/main.py` με την εντολή:

```
Base.metadata.create_all(bind=engine)
```

Αυτό διασφαλίζει ότι οι πίνακες, οι σχέσεις (foreign keys) και τα indexes δημιουργούνται αυτόματα στη βάση δεδομένων σύμφωνα με τον ορισμό των Python κλάσεων (Models).

4.2.2 Μοντελοποίηση Οντοτήτων (SQLAlchemy Models)

Στον κατάλογο `/backend/models` ορίζονται τα εξής σχήματα:

- **User** `/backend/models/user.py`: Αναπαριστά τους χρήστες. Περιέχει τα πεδία `id`, `full_name`, `email` (μοναδικό ευρετήριο), `hashed_password` (αποθηκεύεται με bcrypt hashing), `role` (Enum: `admin`, `operator`, `viewer`), `is_active` (boolean για προσωρινή απενεργοποίηση λογαριασμού) και `receive_email_notifications`.
- **Device** `/backend/models/device.py`: Αναπαριστά τις συσκευές. Περιλαμβάνει πεδία για στατικά στοιχεία (`name`, `ip_address`, `type`, `model`, `location`, `notes`) και πεδία για την αποθήκευση της τελευταίας κατάστασης κατά το monitoring (`last_status`,

last_latency, last_cpu, last_memory, last_traffic, last_polled, last_error, failing_protocols). Συνδέεται με σχέση one-to-many με τα metrics και τα credentials.

- **DeviceCredential /backend/models/device.py**: Αποθηκεύει credentials ανά συσκευή. Περιέχει device_id (foreign key), type (ssh ή snmp), username, password και community_string. Τα πεδία password και community_string κρυπτογραφούνται πριν αποθηκευτούν.
- **DeviceMetric /backend/models/device.py**: Καταγράφει τις ιστορικές μετρήσεις. Περιέχει device_id, timestamp (UTC datetime), cpu_usage, memory_usage, traffic (Mbps) και latency (ms).
- **SystemSettings /backend/models/system_settings.py**: Αποθηκεύει τις global ρυθμίσεις του συστήματος, όπως το polling_interval (προεπιλογή 60s) και τα warning/critical thresholds για CPU, Memory και Latency.
- **AuditLog /backend/models/auditlog.py**: Καταγράφει τις ενέργειες των χρηστών. Περιλαμβάνει timestamp, action (π.χ. "user_login", "device_reboot"), actor_email (ποιος έκανε την ενέργεια), target_type και target_name.

4.3 Μηχανισμός Παρακολούθησης (Monitoring Engine) και Real-time Επικοινωνία

Ο μηχανισμός παρακολούθησης (Monitoring Engine) λειτουργεί ως αυτόνομη background υπηρεσία ενσωματωμένη στο FastAPI, εκμεταλλευόμενη τον ασύγχρονο βρόχο γεγονότων (event loop) της Python.

4.3.1 Ασύγχρονος Scheduler

Στο αρχείο /backend/services/scheduler.py ορίζεται η συνάρτηση start_scheduler(). Η διεργασία αυτή εκτελείται σε ένα ατέρμονο loop:

- Καλεί τη συνάρτηση run_monitoring_cycle() για την έναρξη του polling.
- Ανοίγει ένα database session, διαβάζει την τρέχουσα τιμή του polling_interval από τον πίνακα system_settings και κλείνει το session.
- Εκτελεί await asyncio.sleep(polling_interval) για να αναστείλει την εκτέλεσή της μέχρι τον επόμενο κύκλο.

Η προσέγγιση αυτή επιτρέπει στους διαχειριστές να αλλάζουν το polling interval δυναμικά από το UI, και το backend να προσαρμόζει αυτόματα τη συχνότητα ελέγχου στον επόμενο κύκλο.

```
backend > services > scheduler.py
1  import asyncio
2  from services.monitors import run_monitoring_cycle
3  from database import SessionLocal
4  from models.system_settings import SystemSettings
5  from config import settings
6
7  async def start_scheduler():
8      while True:
9          try:
10             await run_monitoring_cycle()
11         except Exception:
12             pass
13
14         # Fetch interval from db
15         db = SessionLocal()
16         try:
17             sys_settings = db.query(SystemSettings).first()
18             polling_interval = sys_settings.polling_interval if sys_settings else 60
19         finally:
20             db.close()
21
22         await asyncio.sleep(polling_interval)
23
24  def register_scheduler():
25      loop = asyncio.get_event_loop()
26      loop.create_task(start_scheduler())
27
```

Εικόνα 11 Πηγαίος κώδικας του ασύγχρονου Scheduler

4.3.2 Διαργασία Polling ανά Συσκευή

Στο αρχείο /backend/services/monitors.py υλοποιείται η λογική συλλογής μετρικών. Η συνάρτηση `run_monitoring_cycle()` ανακτά όλες τις ενεργές συσκευές από τη βάση δεδομένων και εκκινεί την `poll_device(device_id)` για κάθε συσκευή παράλληλα χρησιμοποιώντας την `asyncio.gather()`. Αυτό επιτρέπει τη σύγχρονη παρακολούθηση δεκάδων συσκευών χωρίς καθυστερήσεις.

Η `poll_device` εκτελεί τους εξής ελέγχους:

1. **Έλεγχος ICMP (Ping):** Εκτελείται η ασύγχρονη subprocess κλήση:

```
ping -c 1 -W 2 <IP>
```

μέσω της `asyncio.create_subprocess_exec`. Αν η εντολή επιστρέψει κωδικό 0, η συσκευή θεωρείται online και καταγράφεται ο χρόνος απόκρισης (latency) σε ms από το stdout.

2. **Συλλογή μέσω SNMP:** Αν η συσκευή είναι online και διαθέτει SNMP credentials, εκτελείται ασύγχρονο SNMP GET request μέσω της βιβλιοθήκης `pysnmp` για συγκεκριμένα OIDs.

3. **Συλλογή μέσω SSH:** Αν η συσκευή διαθέτει SSH credentials (π.χ. Linux servers), το σύστημα συνδέεται ασύγχρονα μέσω της βιβλιοθήκης `paramiko` και εκτελεί CLI εντολές:
 - Για CPU και RAM:

```
top -bn1 | grep 'Cpu(s)' | awk '{print $2 + $4}' και free | grep Mem | awk '{print $3/$2 * 100}'
```

- Για Network Traffic:

```
cat /proc/net/dev | tail -n +3 | awk ' {rx+= $2; tx+= $10} END {print rx+tx}'
```

που επιστρέφει το άθροισμα των ληφθέντων και απεσταλμένων bytes.

```
backend > services >  monitors.py >  ping_device
1  import asyncio
2  import re
3  import json
4  import time
5  from datetime import datetime
6
7  import paramiko
8  from pysnmp.hlapi.asyncio import *
9  from sqlalchemy.orm import Session
10
11 from models.device import Device, DeviceMetric, DeviceCredential
12 from database import SessionLocal
13 from services.alerting import check_for_alerts, notify_users
14 from services.encryptor import decrypt_value
15 from services.websocket_manager import manager
16 from services.redis.service import redis_service
17 from services.auditlogger import log_action
18
19 async def ping_device(ip: str):
20     """ICMP check supporting both IP and Hostnames"""
21     try:
22         proc = await asyncio.create_subprocess_exec(
23             'ping', '-c', '1', '-W', '2', ip,
24             stdout=asyncio.subprocess.PIPE,
25             stderr=asyncio.subprocess.PIPE
26         )
27         stdout, stderr = await proc.communicate()
28
29         if proc.returncode == 0:
30             match = re.search(r'time=([\d.]+)', stdout.decode())
31             return True, float(match.group(1)) if match else 0.0
32
33         return False, None
34     except Exception:
35         return False, None
36
```

Εικόνα 12 Πηγαίος κώδικας για ICMP polling

4.3.3 Υπολογισμός Ρυθμού Κυκλοφορίας (Stateful Traffic Rate)

Επειδή οι μετρήσεις SNMP και SSH επιστρέφουν αθροιστικούς μετρητές bytes (counters) που αυξάνονται συνεχώς, ο υπολογισμός του τρέχοντος ρυθμού (throughput σε MB/s) απαιτεί σύγκριση με την προηγούμενη κατάσταση.

Το σύστημα αποθηκεύει στο Redis (ως key-value με expiration) την προηγούμενη τιμή bytes και τη χρονική σήμανση της μέτρησης. Κατά τον επόμενο έλεγχο, υπολογίζεται η διαφορά των bytes, $\Delta(\text{bytes})$, και η διαφορά του χρόνου, $\Delta(t)$. Ο ρυθμός υπολογίζεται ως εξής:

$$\text{Traffic (MB/s)} = \Delta(\text{bytes}) / (1.048.576 * \Delta(t))$$

Αν δεν υπάρχει προηγούμενη τιμή στο Redis, η μέτρηση αρχικοποιείται και εμφανίζει 0.0 MB/s.

4.3.4 Διασύνδεση Redis Pub/Sub και WebSockets

Για τη μετάδοση των δεδομένων πραγματικού χρόνου στο Dashboard:

1. Αμέσως μόλις η poll_device ολοκληρώσει τη λήψη μετρικών, δημιουργεί ένα JSON payload και το δημοσιεύει (Publish) σε ένα κανάλι Redis με όνομα device_updates.
2. Η δημοσίευση γίνεται ασύγχρονα. Για να αποτραπεί το πάγωμα (blocking) του single-threaded event loop της Python από την επικοινωνία με τον Redis client, χρησιμοποιείται ο thread executor της asyncio (loop.run_in_executor(None, self.redis_client.publish, ...)).
3. Στο /backend/services/redis_service.py, η ασύγχρονη μέθοδος subscribe_and_forward() είναι εγγεγραμμένη (Subscribe) στο ίδιο κανάλι. Για την προστασία από υπερβολική κατανάλωση CPU, η μέθοδος χρησιμοποιεί non-blocking polling (pubsub.get_message) με timeout 0.1 δευτερολέπτων και μικρό sleep (await asyncio.sleep(0.01)).
4. Μόλις ληφθεί ένα μήνυμα από το Redis, αυτό προωθείται άμεσα στον Connection Manager των WebSockets /backend/services/websocket_manager.py, ο οποίος εκτελεί broadcast του payload σε όλες τις ενεργές συνδέσεις WebSocket του frontend.

```
backend > services > redis_service.py > ...
9 class RedisService:
65
66     async def subscribe_and_forward(self):
67         """Subscribe to Redis channel and forward messages to WebSocket clients"""
68         try:
69             if not self.redis_client:
70                 self.connect()
71
72             self.pubsub = self.redis_client.pubsub()
73             self.pubsub.subscribe(self.channel_name)
74             print(f"Redis subscriber listening on channel '{self.channel_name}'")
75
76             # Listen for messages in a non-blocking way
77             while True:
78                 try:
79                     # Run get_message in executor to avoid blocking
80                     loop = asyncio.get_event_loop()
81                     message = await loop.run_in_executor(
82                         None,
83                         self.pubsub.get_message,
84                         True, # ignore_subscribe_messages
85                         0.1 # timeout
86                     )
87
88                     if message and message["type"] == "message":
89                         try:
90                             data = json.loads(message["data"])
91                             await manager.broadcast(data)
92                             print(f"Forwarded update to {len(manager.active_connections)} WebSocket client(s)")
93                         except json.JSONDecodeError as e:
94                             print(f"Invalid JSON in Redis message: {e}")
95                         except Exception as e:
96                             print(f"Error forwarding to WebSocket: {e}")
97
98                     # Small delay to prevent CPU spinning
99                     await asyncio.sleep(0.01)
100
101                 except Exception as e:
102                     print(f"Error in message loop: {e}")
103                     await asyncio.sleep(1)
104
105             except asyncio.CancelledError:
106                 print("Redis subscriber task cancelled")
107                 raise
108             except Exception as e:
109                 print(f"Redis subscriber error: {e}")
110                 # Attempt reconnection after delay
111                 await asyncio.sleep(5)
112                 await self.subscribe_and_forward()
113
114
115 redis_service = RedisService(host=settings.REDIS_HOST, port=settings.REDIS_PORT)
116 | Ctrl+L for Agent
```

Εικόνα 13 Η ασύγχρονη συνάρτηση subscribe_and_forward της κλάσης RedisService

4.3.5 Διαχείριση Alerting και email

Μετά το polling κάθε συσκευής, εκτελείται η check_for_alerts(db, device). Το σύστημα συγκρίνει τις νέες μετρικές (CPU, Memory, Latency, Status) με τα thresholds που ορίζονται στα SystemSettings. Αν κάποια τιμή ξεπερνά τα critical thresholds ή αν η συσκευή είναι offline, δημιουργείται ένα alert.

Για την αποφυγή alert fatigue, τα alerts συγκεντρώνονται σε μια λίστα κατά τη διάρκεια του monitored κύκλου. Στο τέλος του κύκλου, η συνάρτηση run_monitoring_cycle() συνθέτει ένα ενιαίο, συγκεντρωτικό email με όλα τα alerts που προέκυψαν και το αποστέλλει μέσω του /backend/services/email_service.py (χρησιμοποιώντας SMTP με TLS/SSL) στους εγγεγραμμένους χρήστες.

```
backend > services > email_service.py > ...
1  import smtplib
2  import ssl
3  import asyncio
4  from email.message import EmailMessage
5  from config import settings
6
7  def send_email_sync(subject: str, content: str, to_emails: list[str], is_html: bool = False):
8      if not settings.SMTP_USER or not settings.SMTP_PASSWORD:
9          print("SMTP credentials not configured. Skipping email.")
10         return
11
12         msg = EmailMessage()
13         msg["Subject"] = subject
14         msg["From"] = settings.EMAIL_FROM
15         msg["To"] = ", ".join(to_emails)
16
17         if is_html:
18             msg.set_content("This is an HTML email. Please use an email client that supports HTML.")
19             msg.add_alternative(content, subtype='html')
20         else:
21             msg.set_content(content)
22
23         context = ssl.create_default_context()
24
25         try:
26             with smtplib.SMTP(settings.SMTP_HOST, settings.SMTP_PORT) as server:
27                 server.ehlo()
28                 server.starttls(context=context)
29                 server.ehlo()
30                 server.login(settings.SMTP_USER, settings.SMTP_PASSWORD)
31                 server.send_message(msg)
32         except Exception as e:
33             print(f"ERROR: Failed to send email to {to_emails}: {e}")
34
35     async def send_email(subject: str, content: str, to_emails: list[str], is_html: bool = False):
36         await asyncio.to_thread(send_email_sync, subject, content, to_emails, is_html)
37     | Ctrl+L for Agent
```

Εικόνα 14 Ο πηγαίος κώδικας της αποστολής ηλεκτρονικών μηνυμάτων

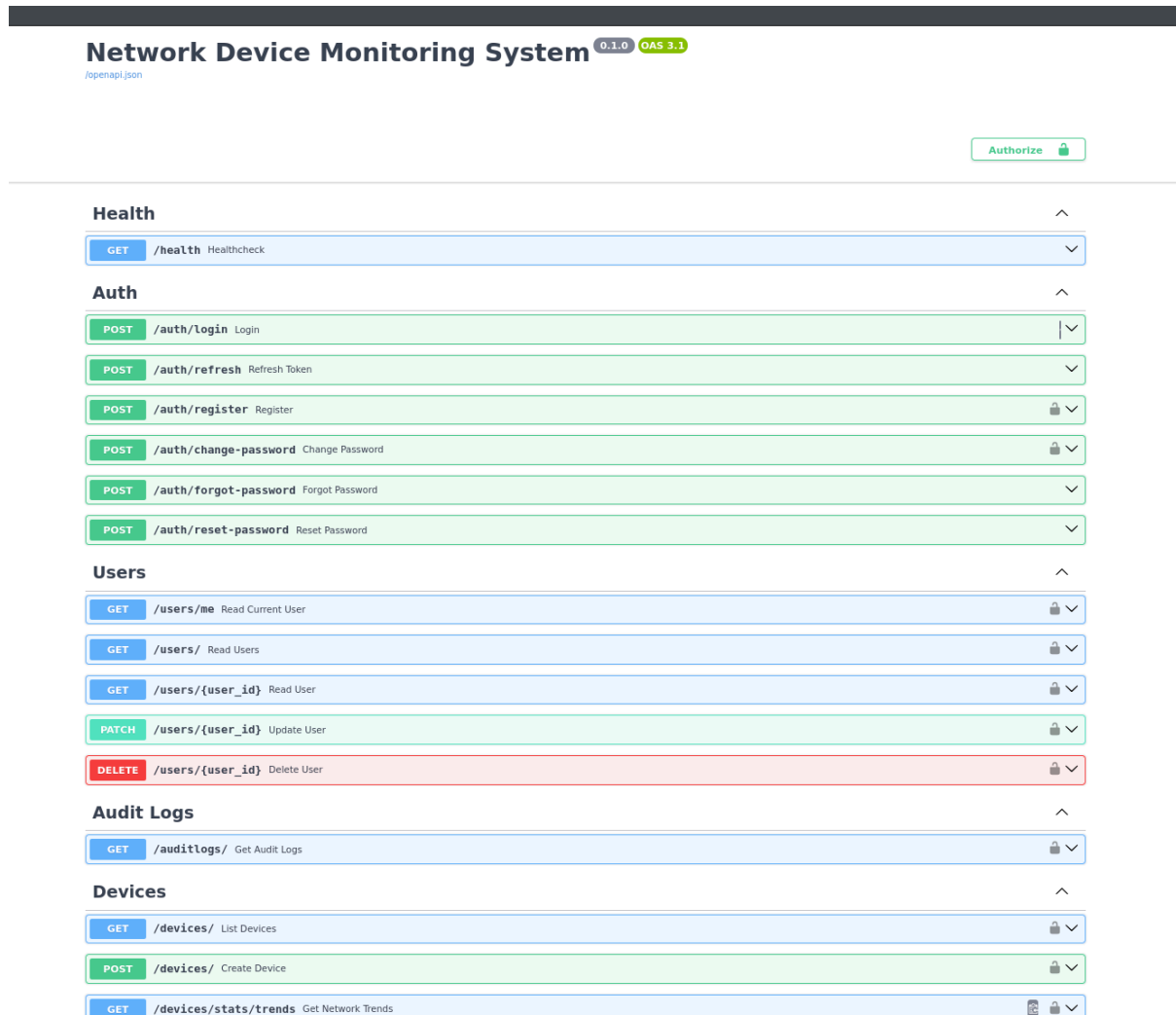
4.4 REST API και Ασφάλεια

Η επικοινωνία μεταξύ frontend και backend πραγματοποιείται μέσω REST API endpoints που προστατεύονται με σύγχρονα πρωτόκολλα ασφαλείας.

4.4.1 Δομή API Routers

Τα endpoints είναι οργανωμένα σε ξεχωριστά αρχεία (routers) στον κατάλογο /backend/api:

- **auth.py:** Endpoints για login, logout και έκδοση JWT tokens.
- **users.py:** Endpoints για διαχείριση χρηστών, αλλαγή κωδικού και προτιμήσεων ειδοποιήσεων.
- **devices.py:** Endpoints για προσθήκη, επεξεργασία, διαγραφή συσκευών, bulk import (CSV) και exports δεδομένων.
- **settings_api.py:** Endpoints για την ανάκτηση και αποθήκευση των συστημικών defaults.
- **auditlogs.py:** Endpoints για την ανάκτηση των logs δραστηριότητας.
- **websocket.py:** Endpoint για την αποδοχή και διατήρηση των WebSocket συνδέσεων.



Εικόνα 15 Η τεκμηρίωση των REST API endpoints μέσω του Swagger UI

4.4.2 Αυθεντικοποίηση JWT (JSON Web Tokens) και RBAC

Η ασφάλεια των endpoints βασίζεται σε JWT tokens. Κατά το login, το backend εκδίδει ένα token υπογεγραμμένο με τον αλγόριθμο HS256 και ένα secret key. Το token περιέχει claims όπως το email του χρήστη, τον ρόλο του (role) και την ημερομηνία λήξης (exp).

Κάθε προστατευμένο endpoint του FastAPI χρησιμοποιεί dependencies που ελέγχουν το Authorization header (Bearer <token>). Αν το token έχει λήξει ή η υπογραφή είναι άκυρη, επιστρέφεται 401 Unauthorized.

Επιπλέον, εφαρμόζεται Role-Based Access Control (RBAC). Για παράδειγμα, το endpoint για την ενημέρωση των System Settings χρησιμοποιεί το dependency:

```
current_user = get_current_active_user(role="admin")
```

Αν ο συνδεδεμένος χρήστης είναι Operator ή Viewer, το FastAPI επιστρέφει αυτόματα 403 Forbidden, εμποδίζοντας την ενέργεια σε επίπεδο backend.

4.4.3 Κρυπτογράφηση Credentials στη Βάση Δεδομένων

Για την ασφάλεια των credentials των συσκευών, στο αρχείο `/backend/services/encryptor.py` υλοποιείται συμμετρική κρυπτογράφηση AES με χρήση του Fernet από τη βιβλιοθήκη `cryptography`.

Για τη διασφάλιση της μέγιστης προστασίας, το κλειδί κρυπτογράφησης παράγεται δυναμικά (key derivation) μέσω της συνάρτησης `_derive_key` η οποία εκτελεί SHA-256 hashing στη μεταβλητή περιβάλλοντος `DEVICE_CREDENTIAL_SECRET_KEY` και στη συνέχεια κωδικοποιεί το αποτέλεσμα σε URL-safe Base64:

```
digest = hashlib.sha256(secret.encode()).digest()
return base64.urlsafe_b64encode(digest)
```

Όταν ένας Operator προσθέτει credentials για SSH ή SNMP, αυτά κρυπτογραφούνται πριν αποθηκευτούν στους πίνακες της PostgreSQL. Η αποκρυπτογράφηση γίνεται αποκλειστικά στη μνήμη του server κατά το runtime του polling, εξασφαλίζοντας ότι οι κωδικοί δεν εκτίθενται ποτέ σε απλό κείμενο (plaintext) στη βάση δεδομένων.

```
backend > services > encryptor.py > ...
1  import base64
2  import hashlib
3  from cryptography.fernet import Fernet
4  from config import settings
5
6
7  def _derive_key(secret: str) -> bytes:
8      """
9      Derive a stable Fernet key from an environment secret.
10     """
11     digest = hashlib.sha256(secret.encode()).digest()
12     return base64.urlsafe_b64encode(digest)
13
14
15  _fernet = Fernet(_derive_key(settings.DEVICE_CREDENTIAL_SECRET_KEY))
16
17
18  def encrypt_value(value: str | None) -> str | None:
19     """
20     Encrypt a sensitive value for storage.
21     Treat None, empty strings, or whitespace-only strings as None.
22     """
23     if not value or not str(value).strip():
24         return None
25     return _fernet.encrypt(str(value).encode()).decode()
26
27
28  def decrypt_value(value: str | None) -> str | None:
29     """
30     Decrypt a sensitive value for runtime usage.
31     """
32     if value is None:
33         return None
34     return _fernet.decrypt(value.encode()).decode()
35
```

Εικόνα 16 Ο πηγαίος κώδικας του αρχείου `backend/services/encryptor.py`

4.5 Containerization και Ενορχήστρωση με Docker

Για τη διασφάλιση της φορητότητας (portability) και της ευκολίας ανάπτυξης, ολόκληρη η εφαρμογή είναι κοντεϊνερποιημένη (containerized) με τη χρήση του Docker.

4.5.1 Dockerfiles

- **Backend Dockerfile (/backend/Dockerfile):**

Βασίζεται στην επίσημη ελαφριά εικόνα python:3.10-slim.

Αντιγράφει το requirements.txt και εγκαθιστά τις βιβλιοθήκες (FastAPI, SQLAlchemy, PySNMP, Paramiko, cryptography κ.λπ.).

Αντιγράφει τον κώδικα της εφαρμογής και εκκινεί τον unicorn server στη θύρα 8000.

- **Frontend Dockerfile (/frontend/Dockerfile):**

Χρησιμοποιεί multi-stage build.

Φάση 1 (Build): Χρησιμοποιεί την εικόνα node:18 για την εγκατάσταση των npm πακέτων και τη δημιουργία του βελτιστοποιημένου production build (npm run build).

Φάση 2 (Serve): Αντιγράφει τα static assets (HTML/JS/CSS) στην εικόνα nginx:alpine και ρυθμίζει τον Nginx να σερβίρει την εφαρμογή στη θύρα 80, διαχειριζόμενος σωστά τα routes της React SPA.

4.5.2 Docker Compose και Παραμετροποίηση (.env)

Η ενορχήστρωση των επιμέρους containers ορίζεται στο αρχείο docker-compose.yml στη ρίζα του έργου. Ορίζει τέσσερις υπηρεσίες:

1. **db**: Postgres database. Χρησιμοποιεί μόνιμο volume (postgres_data) για την επιμονή των δεδομένων.
2. **redis**: Redis server για caching και pub/sub streaming.
3. **api**: FastAPI backend. Εξαρτάται από τις υπηρεσίες db και redis.
4. **frontend**: Nginx web server που εκθέτει την εφαρμογή στο host σύστημα (port mapping 5173:80).

```
docker-compose.yml
1  version: "3.9"
2
3  services:
4    api:
5      build: ./backend
6      container_name: fastapi_app
7      command: uvicorn main:app --host 0.0.0.0 --port 8000 --reload
8      ports:
9        - "8000:8000"
10     env_file:
11       - ./backend/.env
12     depends_on:
13       - db
14       - redis
15     volumes:
16       - ./backend:/app
17
18     db:
19       image: postgres:15
20       container_name: postgres_db
21       restart: always
22       env_file:
23         - ./backend/.env
24       ports:
25         - "5432:5432"
26       volumes:
27         - postgres_data:/var/lib/postgresql/data
28
29     redis:
30       image: redis:7
31       container_name: redis_cache
32       restart: always
33       ports:
34         - "6379:6379"
35
36     frontend:
37       build:
38         context: ./frontend
39       container_name: frontend_app
40       ports:
41         - "5173:80"
42       depends_on:
43         - api
44
45
46     volumes:
47       postgres_data:
48
```

Εικόνα 17 Περιεχόμενα του αρχείου docker-compose.yml

Όλη η παραμετροποίηση του συστήματος (credentials βάσης, secret keys για JWT, κλειδιά κρυπτογράφησης credentials και ρυθμίσεις SMTP για την αποστολή emails) αποθηκεύεται στο αρχείο .env του backend. Το Docker Compose εισάγει αυτόματα αυτές τις μεταβλητές στα container environments, εξασφαλίζοντας την τήρηση των βέλτιστων πρακτικών ασφαλείας.

5. Λειτουργικότητα και Διεπαφή Χρήστη

Στο κεφάλαιο αυτό παρουσιάζεται αναλυτικά η λειτουργία και η διεπαφή χρήστη (User Interface - UI) του αναπτυγμένου συστήματος παρακολούθησης και διαχείρισης δικτυακών συσκευών. Η παρουσίαση γίνεται μέσα από την οπτική γωνία των διαφορετικών χρηστών της εφαρμογής, εστιάζοντας στη χρηστικότητα (usability), στη φιλικότητα του περιβάλλοντος πλοήγησης (UX) και στον τρόπο με τον οποίο το σύστημα ανταποκρίνεται στις επιχειρησιακές ανάγκες διαχείρισης δικτύων.

Η ανάλυση ακολουθεί τις βασικές σελίδες και ροές εργασίας της εφαρμογής:

1. Τη διαδικασία εισόδου στο σύστημα (Login Page) και την ασφαλή επαλήθευση στοιχείων.
2. Τη ροή επαναφοράς ξεχασμένου κωδικού πρόσβασης (Password Reset).
3. Την κεντρική κονσόλα παρακολούθησης (Dashboard) με τα widgets και τα δυναμικά γραφήματα πραγματικού χρόνου.
4. Το inventory και τη διαχείριση συσκευών (Devices View) με τις επιλογές φιλτραρίσματος, bulk import και exports.
5. Την εκτέλεση απομακρυσμένων ενεργειών (SSH Actions) όπως reboots και service restarts.
6. Τη διαχείριση των χρηστών και των δικαιωμάτων τους (Users Administration).
7. Την προβολή των καταγραφών ασφαλείας και ενεργειών (Audit Logs).
8. Την παραμετροποίηση των προτιμήσεων προφίλ, notifications και των συστημικών defaults (Settings).

5.1 Είσοδος Χρήστη

Η σελίδα εισόδου (Login Page) αποτελεί την πύλη εισόδου των χρηστών στην εφαρμογή και έχει σχεδιαστεί με γνώμονα την απλότητα και την ασφάλεια. Ο χρήστης καλείται να εισάγει τη διεύθυνση ηλεκτρονικού ταχυδρομείου (email) και τον κωδικό πρόσβασης (password).



Εικόνα 18 Η σελίδα εισόδου (Login) του συστήματος

Το σύστημα πραγματοποιεί επικαιροποίηση των στοιχείων τόσο σε επίπεδο frontend (έγκυρη μορφή email) όσο και σε επίπεδο backend:

- Σε περίπτωση εισαγωγής κενών ή λανθασμένων στοιχείων μορφής, εμφανίζονται άμεσα κόκκινα μηνύματα σφάλματος κάτω από τα αντίστοιχα πεδία.
- Σε περίπτωση αποτυχίας αυθεντικοποίησης λόγω λανθασμένων διαπιστευτηρίων, εμφανίζεται ένα toast notification με το μήνυμα "Invalid email or password".
- Με την επιτυχή είσοδο, ο χρήστης λαμβάνει το JWT token και ανακατευθύνεται αυτόματα στην κεντρική σελίδα του συστήματος (Dashboard).

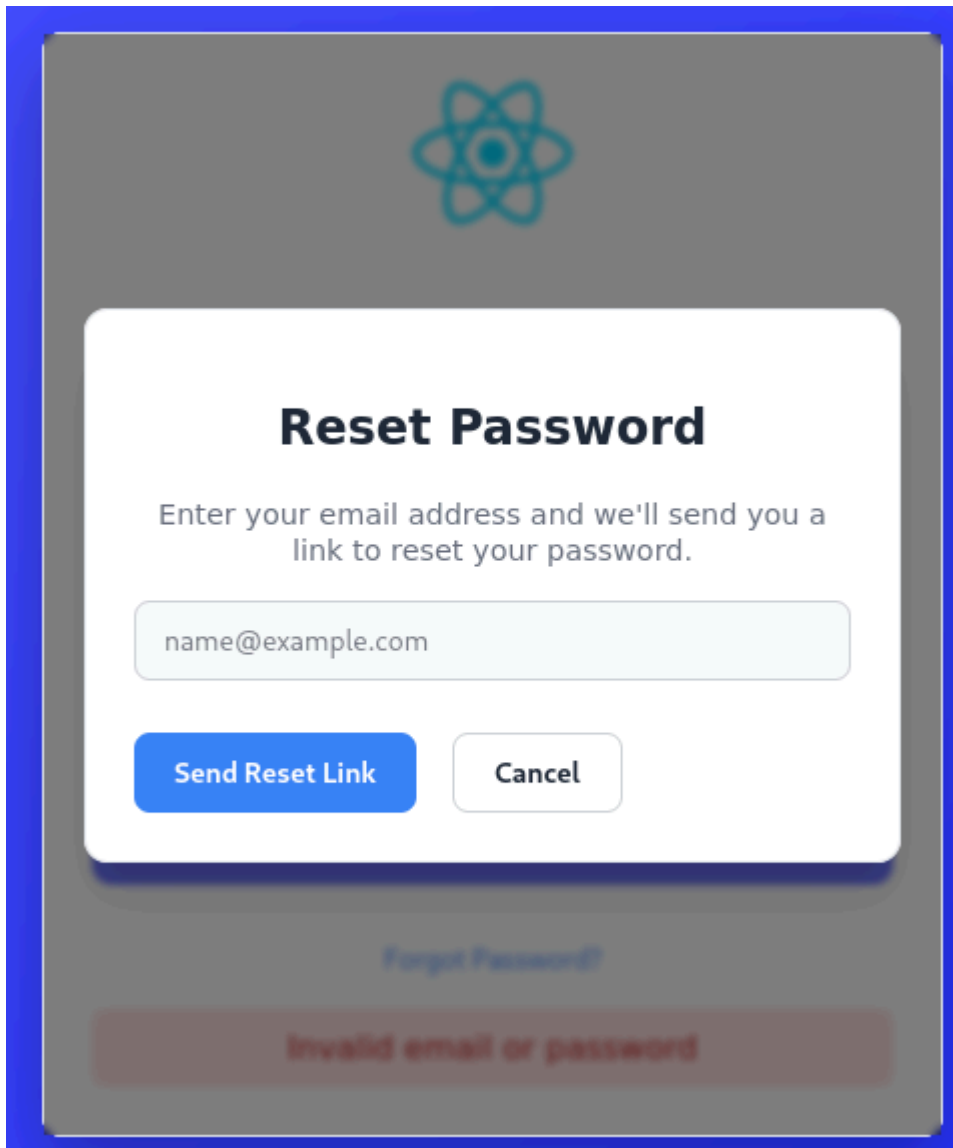


Εικόνα 19 Μήνυμα λανθασμένου email/κωδικού πρόσβασης

5.2 Επαναφορά Κωδικού Πρόσβασης

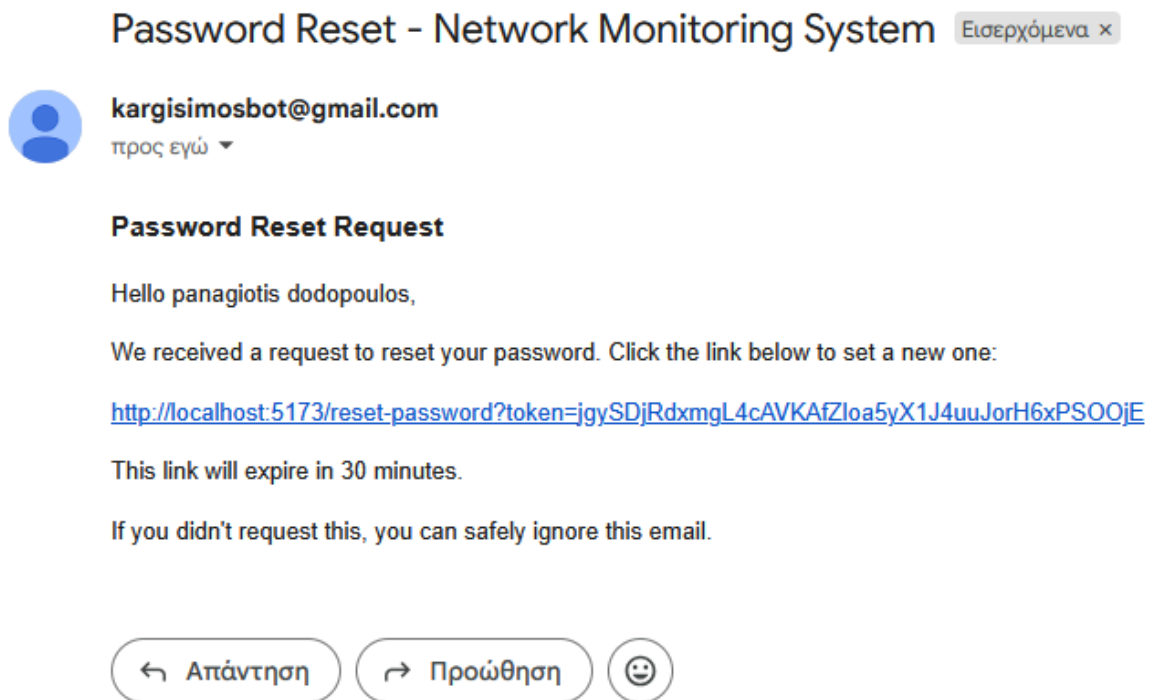
Σε περίπτωση που ένας χρήστης ξεχάσει τον κωδικό πρόσβασής του, η εφαρμογή παρέχει μια ασφαλή ροή ανάκτησης (Password Reset):

- Από τη σελίδα Login, ο χρήστης επιλέγει τον σύνδεσμο "Forgot Password", ο οποίος ανοίγει μια φόρμα εισαγωγής email.



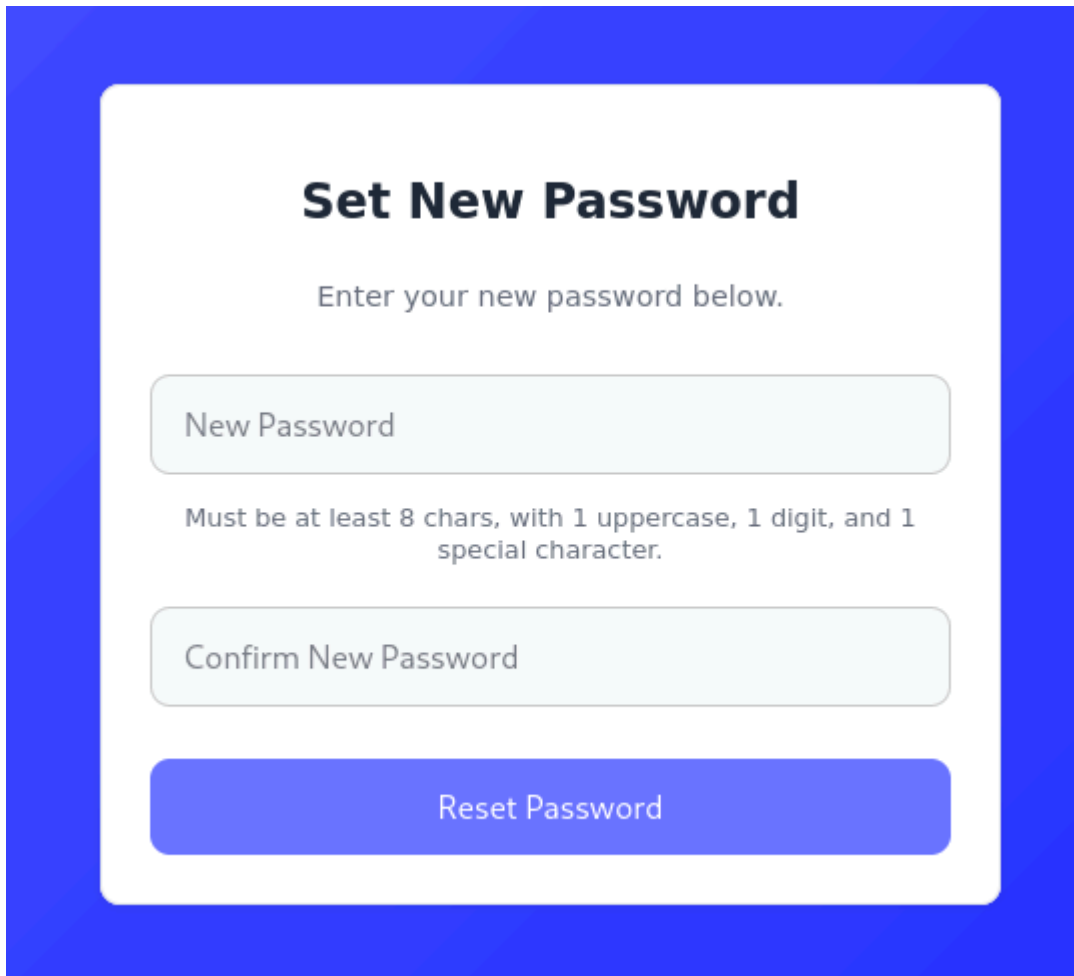
Εικόνα 20 Η φόρμα αιτήματος επαναφοράς κωδικού

- Εισάγοντας το email του, το σύστημα ελέγχει αν ο λογαριασμός υπάρχει στη βάση δεδομένων και, εφόσον υπάρχει, αποστέλλει ένα email που περιέχει έναν μοναδικό, χρονικά περιορισμένο σύνδεσμο επαναφοράς (reset token).



Εικόνα 21 Το ηλεκτρονικό μήνυμα με τον σύνδεσμο αλλαγής του κωδικού πρόσβασης

- Πατώντας τον σύνδεσμο, ο χρήστης μεταφέρεται σε μια ειδική σελίδα όπου εισάγει τον νέο κωδικό πρόσβασης. Η σελίδα επιβάλλει την ισχύουσα πολιτική ασφαλείας (τουλάχιστον 8 χαρακτήρες, 1 κεφαλαίο γράμμα, 1 ψηφίο και 1 ειδικό χαρακτήρα) και εμφανίζει βοηθητικό κείμενο για τη συμπλήρωση των απαιτήσεων.



Set New Password

Enter your new password below.

New Password

Must be at least 8 chars, with 1 uppercase, 1 digit, and 1 special character.

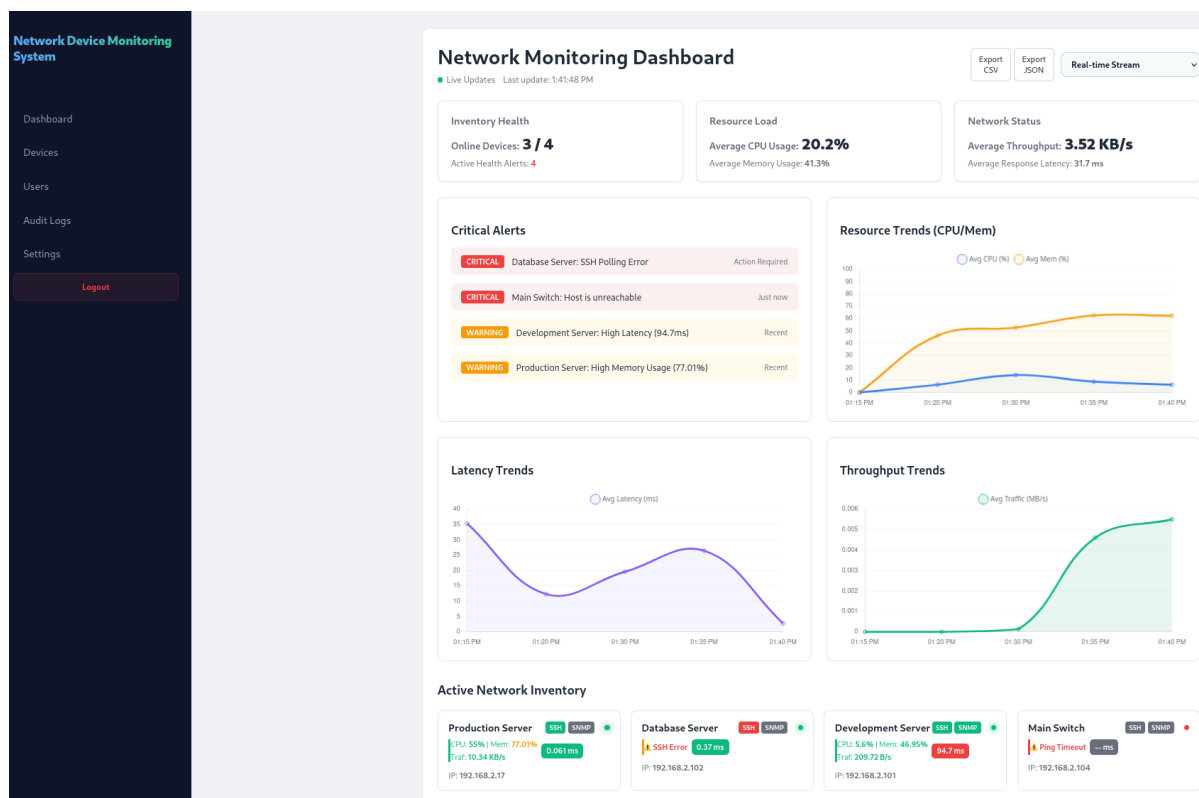
Confirm New Password

Reset Password

Εικόνα 22 Η σελίδα ορισμού νέου κωδικού πρόσβασης

5.3 Κεντρική Σελίδα Παρακολούθησης

Το Dashboard αποτελεί την κεντρική οθόνη του συστήματος, προσφέροντας στους διαχειριστές και τους operators μια σφαιρική και άμεση εικόνα της κατάστασης του δικτύου σε πραγματικό χρόνο.



Εικόνα 23 Η κεντρική σελίδα του dashboard

Η σελίδα αποτελείται από τρεις βασικές ενότητες:

1. KPI Widgets
2. Δυναμικά Γραφήματα
3. Live Alerts Pane

5.3.1 KPI Widgets

Στο επάνω μέρος προβάλλονται τέσσερα widgets που συνοψίζουν την κατάσταση του δικτύου:

- **Total Devices (Σύνολο Συσκευών):** Εμφανίζει τον αριθμό των καταχωρημένων συσκευών.
- **Online/Offline Devices:** Εμφανίζει τον αριθμό των ενεργών και ανενεργών συσκευών με χρωματική κωδικοποίηση (πράσινο για online, κόκκινο για offline).
- **Average Latency (Μέσος Χρόνος Απόκρισης):** Δείχνει τον μέσο χρόνο latency σε ms για όλες τις online συσκευές.
- **Active Alerts (Ενεργά Alerts):** Εμφανίζει το πλήθος των κρίσιμων προειδοποιήσεων που εκκρεμούν.

5.3.2 Δυναμικά Γραφήματα (Chart.js)

Στο κέντρο της σελίδας προβάλλονται διαγράμματα (line charts) που απεικονίζουν τις ιστορικές τάσεις απόδοσης:

- **CPU Load Trend (%)**: Παρακολουθεί τη μέση χρήση επεξεργαστή.
- **Memory Usage Trend (%)**: Παρακολουθεί τη χρήση μνήμης RAM.
- **Network Throughput (Mbps)**: Παρακολουθεί τον ρυθμό διακίνησης δεδομένων.

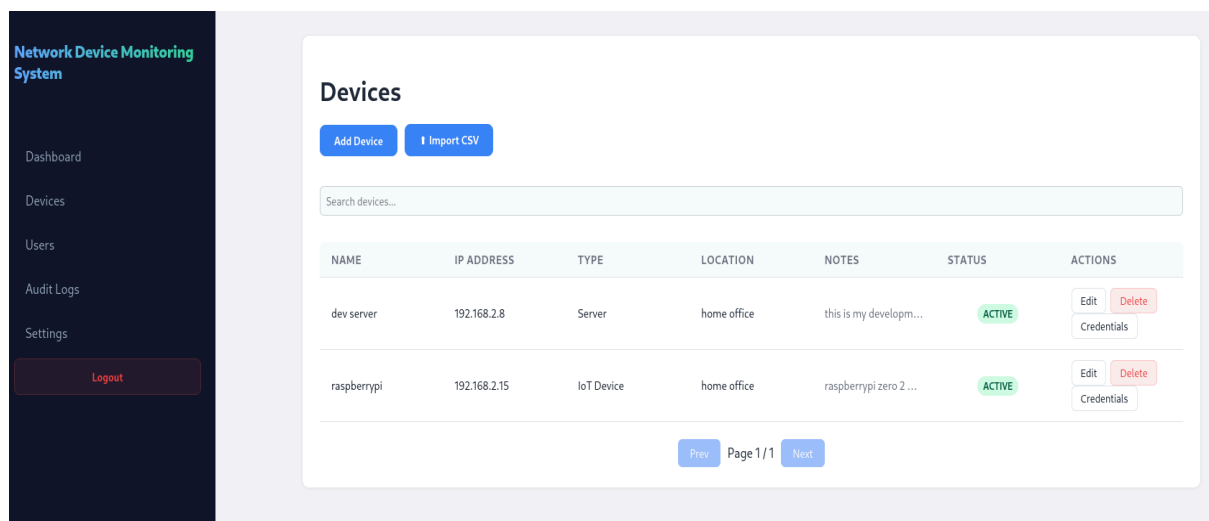
Τα γραφήματα υποστηρίζουν φιλτράρισμα χρονικού εύρους (24 ώρες, 7 ημέρες, 30 ημέρες) και ενημερώνονται δυναμικά σε πραγματικό χρόνο (χωρίς ανανέωση της σελίδας) μέσω WebSockets, μόλις ολοκληρωθεί ένας monitoring κύκλος στο backend.

5.3.3 Live Alerts Pane

Στη δεξιά ή κάτω πλευρά του Dashboard βρίσκεται ένα πάνελ με τις τρέχουσες ενεργές ειδοποιήσεις. Κάθε ειδοποίηση εμφανίζει το όνομα της συσκευής, το είδος του alert (π.χ. "CPU utilization exceeded warning threshold"), τη σοβαρότητα (Warning/Critical) και το timestamp της καταγραφής, επιτρέποντας στους μηχανικούς δικτύου να εντοπίζουν άμεσα τις δυσλειτουργίες.

5.4 Inventory και Διαχείριση Συσκευών

Η σελίδα Devices αποτελεί το inventory των δικτυακών στοιχείων του συστήματος, επιτρέποντας στους Operator και Admin να διαχειρίζονται τις συσκευές.



Εικόνα 24 Η σελίδα διαχείρισης συσκευών

5.4.1 Πίνακας Inventory

Όλες οι καταχωρημένες συσκευές εμφανίζονται σε έναν διαδραστικό πίνακα με δυνατότητες σελιδοποίησης (pagination), ταξινόμησης ανά στήλη και αναζήτησης/φιλτραρίσματος βάσει ονόματος, IP διεύθυνσης, τοποθεσίας ή τύπου συσκευής. Κάθε γραμμή εμφανίζει:

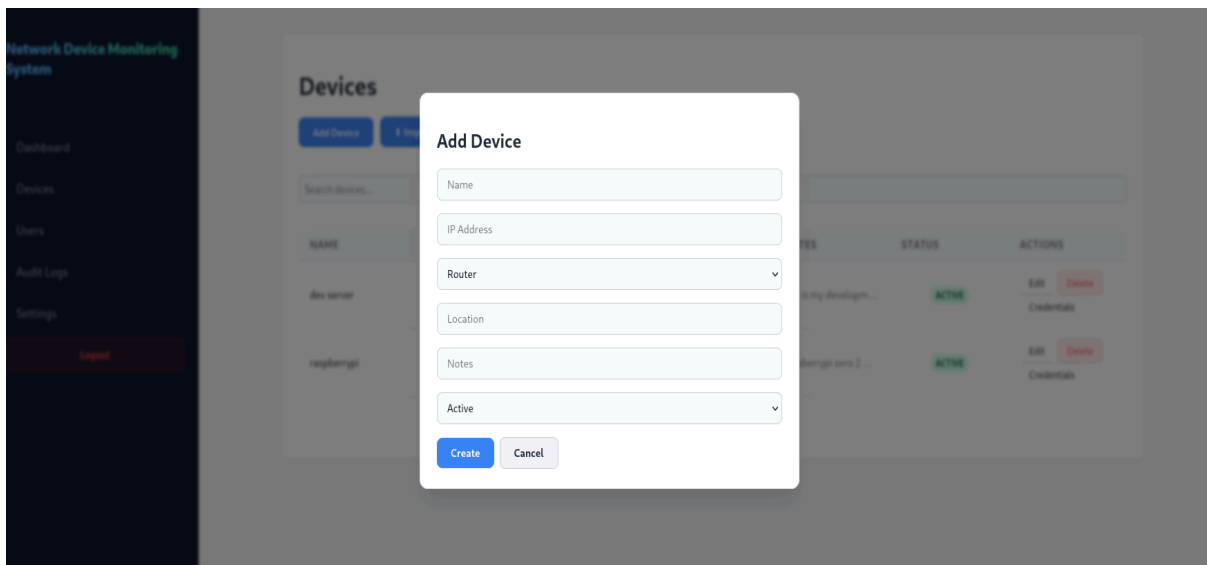
- Το όνομα και την IP διεύθυνση της συσκευής.
- Έναν visual indicator της κατάστασης (πράσινη κουκίδα για online, κόκκινη για offline).
- Τις τρέχουσες τιμές CPU, Memory, Latency και Traffic.
- Τα failing protocols (π.χ. αν το ping λειτουργεί αλλά το SNMP έχει auth error).
- Ένα κουμπί μενού ενεργειών.

5.4.2 Προσθήκη και Επεξεργασία Συσκευής

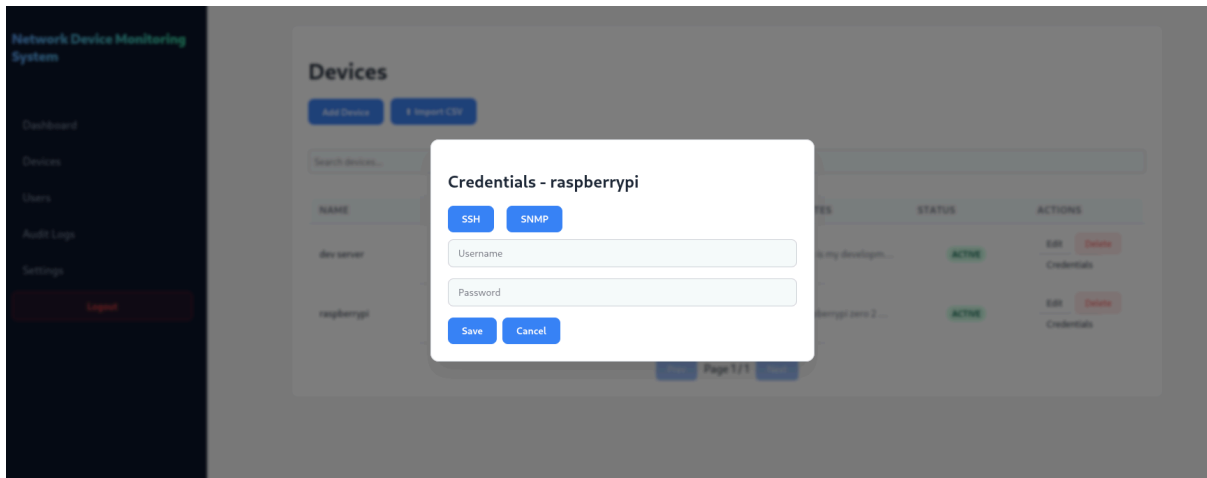
Πατώντας το κουμπί "Add Device", ανοίγει ένα modal παράθυρο που ζητά τα στατικά στοιχεία της συσκευής (Name, IP, Type, Location, Notes) και προαιρετικά τα Credentials:

- **SSH Credentials:** Username και Password για Linux servers.
- **SNMP Credentials:** Community String (π.χ. public) για routers/switches.

Τα credentials κρυπτογραφούνται αυτόματα κατά την υποβολή.



Εικόνα 25 Το παράθυρο προσθήκης νέας συσκευής



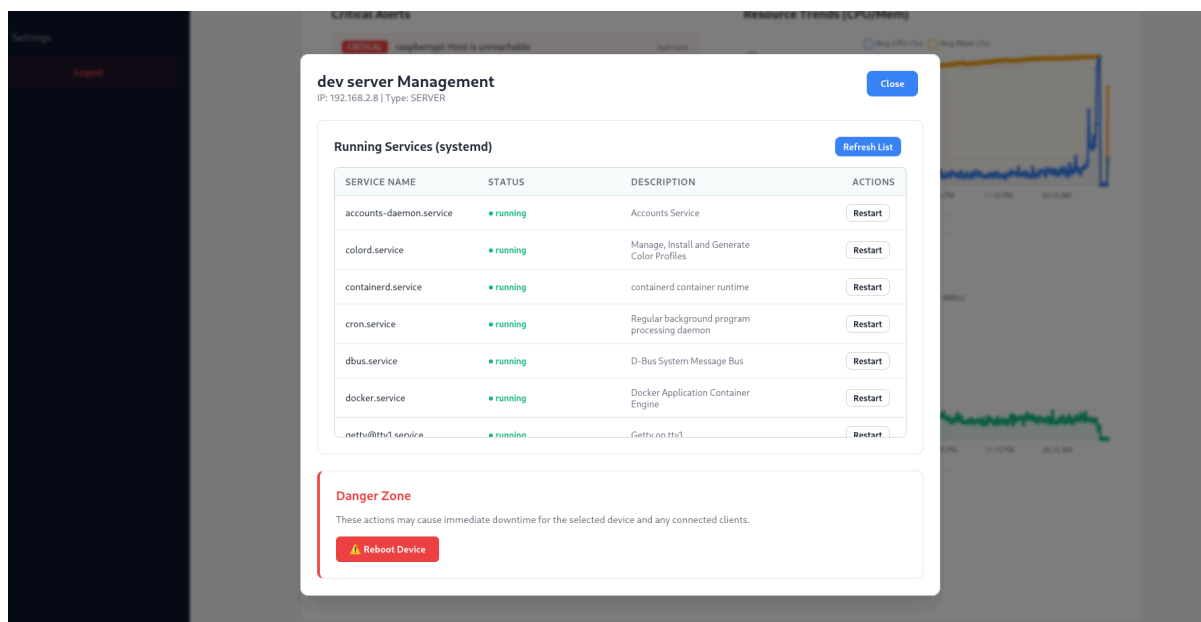
Εικόνα 26 Το παράθυρο προσθήκης credentials για συσκευές

5.4.2 Bulk Import και Data Export

Για την εύκολη αρχικοποίηση μεγάλων δικτύων, παρέχεται η δυνατότητα "Bulk Import" μέσω αρχείου CSV. Το σύστημα αναλύει το αρχείο, επικυρώνει τις IP διευθύνσεις και εισάγει μαζικά τις συσκευές. Επιπλέον, παρέχονται επιλογές "Export" για τη λήψη του ιστορικού των μετρήσεων σε μορφή CSV ή JSON για εξωτερική ανάλυση.

5.5 Απομακρυσμένη Διαχείριση και Κονσόλα Ενεργειών

Μία από τις πιο προηγμένες λειτουργίες της εφαρμογής είναι η δυνατότητα ενεργητικής διαχείρισης των συσκευών απευθείας από το UI (διαθέσιμη μόνο σε Operators και Admins), εφόσον έχουν ρυθμιστεί έγκυρα SSH credentials.



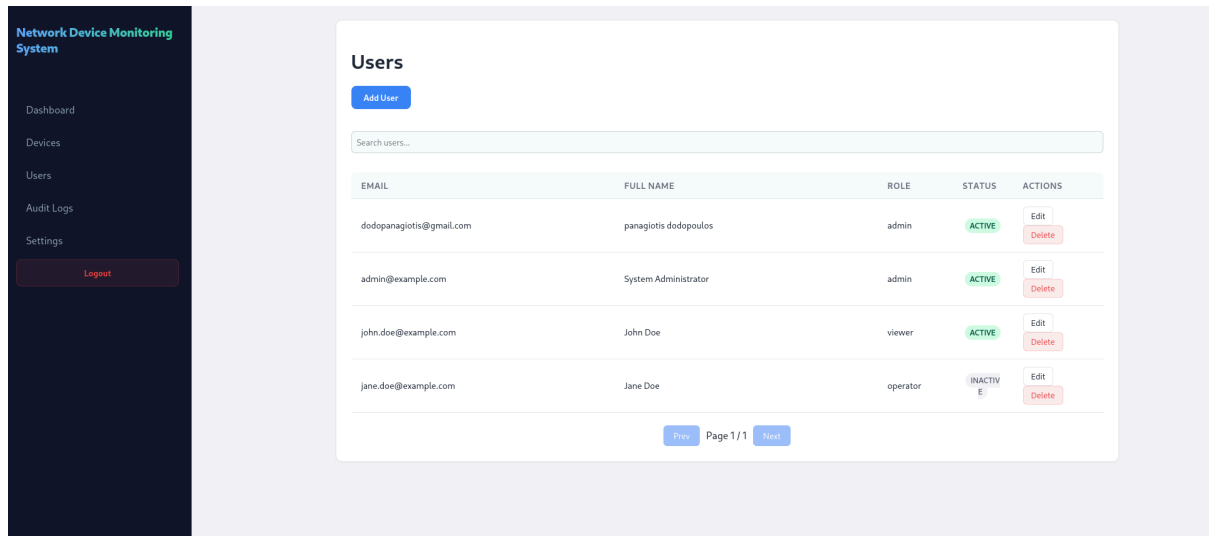
Εικόνα 27 Το μενού ενεργειών και η κονσόλα απομακρυσμένης διαχείρισης

Στο μενού ενεργειών κάθε συσκευής, παρέχονται οι εξής δυνατότητες:

- Reboot Device (Επανεκκίνηση): Αποστέλλει ασύγχρονη εντολή reboot (sudo reboot) στη συσκευή μέσω SSH. Το σύστημα ζητά επιβεβαίωση από τον χρήστη πριν την εκτέλεση και καταγράφει την ενέργεια στο Audit Log.
- Restart Service (Επανεκκίνηση Υπηρεσίας): Ανοίγει ένα modal παράθυρο όπου ο διαχειριστής εισάγει το όνομα μιας systemd υπηρεσίας (π.χ. nginx, apache2, docker) και εκτελεί την εντολή επανεκκίνησης (sudo systemctl restart <service>). Το backend επιστρέφει το αποτέλεσμα της εκτέλεσης (success/error) το οποίο εμφανίζεται στην οθόνη του χρήστη.

5.6 Διαχείριση Χρηστών και Ρόλων

Η σελίδα Users (προσβάσιμη μόνο από Admin) επιτρέπει τον έλεγχο των χρηστών που έχουν πρόσβαση στο σύστημα.



Εικόνα 28 Η οθόνη διαχείρισης χρηστών

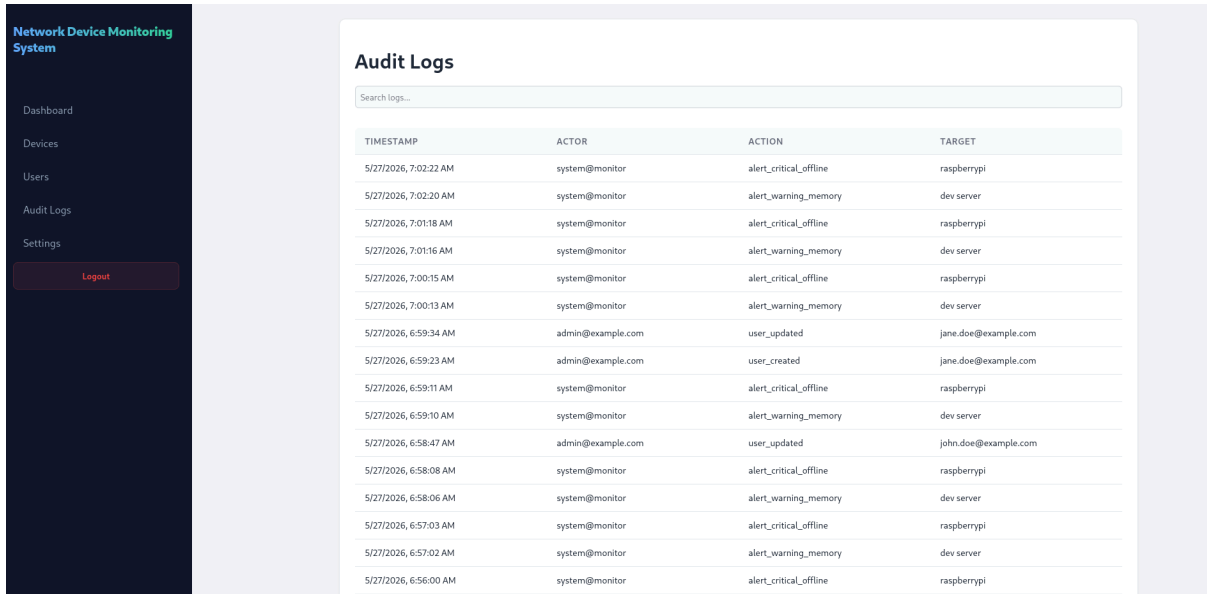
Ο διαχειριστής μπορεί να:

- Προβάλλει τη λίστα των χρηστών, τα στοιχεία επικοινωνίας και τους ρόλους τους.
- Αλλάξει τον ρόλο ενός χρήστη (Admin, Operator, Viewer) μέσω ενός dropdown επιλογέα.
- Ενεργοποιήσει ή απενεργοποιήσει προσωρινά έναν λογαριασμό (toggle active state).

Οι αλλαγές εφαρμόζονται άμεσα: αν ένας χρήστης απενεργοποιηθεί ή υποβαθμιστεί ο ρόλος του, το JWT token του καθίσταται άκυρο για τις αντίστοιχες προστατευμένες λειτουργίες κατά την επόμενη κλήση API.

5.7 Καταγραφές Ασφαλείας (Audit Logs)

Η σελίδα Audit Logs παρέχει ένα πλήρες ιστορικό ιχνηλασιμότητας όλων των κρίσιμων ενεργειών που πραγματοποιήθηκαν στο σύστημα, διασφαλίζοντας τη συμμόρφωση με τα πρότυπα ασφαλείας.



TIMESTAMP	ACTOR	ACTION	TARGET
5/27/2026, 7:02:22 AM	system@monitor	alert_critical_offline	raspberrypi
5/27/2026, 7:02:20 AM	system@monitor	alert_warning_memory	dev server
5/27/2026, 7:01:18 AM	system@monitor	alert_critical_offline	raspberrypi
5/27/2026, 7:01:16 AM	system@monitor	alert_warning_memory	dev server
5/27/2026, 7:00:15 AM	system@monitor	alert_critical_offline	raspberrypi
5/27/2026, 7:00:13 AM	system@monitor	alert_warning_memory	dev server
5/27/2026, 6:59:34 AM	admin@example.com	user_updated	jane.doe@example.com
5/27/2026, 6:59:23 AM	admin@example.com	user_created	jane.doe@example.com
5/27/2026, 6:59:11 AM	system@monitor	alert_critical_offline	raspberrypi
5/27/2026, 6:59:10 AM	system@monitor	alert_warning_memory	dev server
5/27/2026, 6:58:47 AM	admin@example.com	user_updated	john.doe@example.com
5/27/2026, 6:58:08 AM	system@monitor	alert_critical_offline	raspberrypi
5/27/2026, 6:58:06 AM	system@monitor	alert_warning_memory	dev server
5/27/2026, 6:57:03 AM	system@monitor	alert_critical_offline	raspberrypi
5/27/2026, 6:57:02 AM	system@monitor	alert_warning_memory	dev server
5/27/2026, 6:56:00 AM	system@monitor	alert_critical_offline	raspberrypi

Εικόνα 29 Ο πίνακας Audit Logs του συστήματος

Ο πίνακας εμφανίζει:

- Το Timestamp της ενέργειας.
- Τον τύπο της ενέργειας (Action) (π.χ. "change_settings", "reboot_device", "user_login").
- Το email του χρήστη που πραγματοποίησε την ενέργεια (Actor).
- Τον τύπο και το όνομα της συσκευής-στόχου (Target).

Ο διαχειριστής μπορεί να κάνει αναζήτηση βάσει email ή ενέργειας και να φιλτράρει τα logs με βάση τη χρονική περίοδο, διευκολύνοντας τον εντοπισμό σφαλμάτων ή ύποπτης δραστηριότητας.

5.8 Ρυθμίσεις Συστήματος (Settings)

Η σελίδα Settings παρέχει επιλογές παραμετροποίησης του συστήματος, διαχωρισμένες σε ενότητες:

- Profile: Εμφανίζει τα στοιχεία του συνδεδεμένου χρήστη και παρέχει ένα modal παράθυρο για την αλλαγή του κωδικού πρόσβασης (Change Password).
- Interface: Περιλαμβάνει τον διακόπτη (switch) για την ενεργοποίηση/απενεργοποίηση του Dark Mode.
- Notifications: Επιτρέπει στον χρήστη να επιλέξει αν επιθυμεί να λαμβάνει συγκεντρωτικά email ειδοποιήσεων για κρίσιμα συμβάντα (Critical Alerts via Email).
- System Defaults (Admin Only): Επιτρέπει τη διαχείριση των defaults του monitoring engine.
 - Polling Interval: Range slider από 10 έως 300 δευτερόλεπτα.
 - Latency Warning: Range slider από 10 έως 1000 ms.
 - CPU Warning / Memory Warning: Range sliders από 10% έως 100%.

- CPU Critical / Memory Critical: Range sliders από 10% έως 100%.
- Constraint Logic: Αν ο διαχειριστής σύρει το Warning threshold σε τιμή μεγαλύτερη ή ίση από το Critical threshold, εμφανίζεται προειδοποιητικό μήνυμα "Warning thresholds must be strictly less than critical thresholds", η γραμμή του slider κοκκινίζει και το κουμπί "Save Settings" απενεργοποιείται αυτόματα, διασφαλίζοντας τη λογική συνοχή των ειδοποιήσεων.

Settings

Profile

EMAIL	admin@example.com
FULL NAME	System Administrator
ROLE	admin
ACTIVE	Yes

Interface

Dark Mode
Switch between light and dark themes ☐

Notifications

Critical Alerts via Email
Receive email for host downtime, auth failures, and critical load (>90%) ☒

Security

[Change Password](#)

System Defaults

Polling Interval
Frequency of background host metrics collection. **60s**

Latency Warning
Threshold for flagging warning-level host ping latency. **40 ms**

CPU Warning Threshold
Threshold for CPU usage alerts (warning state). **70%**

Εικόνα 30 Η σελίδα ρυθμίσεων (Settings) με τους range sliders και τη λειτουργία validation

6. Δοκιμές και Αξιολόγηση

Στο κεφάλαιο αυτό παρουσιάζεται αναλυτικά η διαδικασία δοκιμών και η τεχνική αξιολόγηση του αναπτυγμένου συστήματος παρακολούθησης και διαχείρισης δικτυακών συσκευών. Η αξιολόγηση αυτή επικεντρώνεται στον έλεγχο της λειτουργικής ορθότητας (functional correctness), της ασφάλειας (security audit) και της απόδοσης (performance analysis) της εφαρμογής σε συνθήκες κλιμάκωσης. Οι δοκιμές πραγματοποιήθηκαν σε όλα τα επίπεδα της αρχιτεκτονικής (frontend, backend, βάση δεδομένων, monitoring engine) για να διασφαλιστεί ότι το σύστημα ανταποκρίνεται πλήρως στις λειτουργικές και μη λειτουργικές απαιτήσεις που τέθηκαν στο Κεφάλαιο 3, χωρίς την ανάγκη εξωτερικών συμμετεχόντων, εστιάζοντας στην τεχνική επαλήθευση των προδιαγραφών.

Συγκεκριμένα, αναλύονται:

1. Ο λειτουργικός έλεγχος και οι δοκιμές σεναρίων ροής δεδομένων (End-to-End Testing) στο σύστημα.
2. Η αξιολόγηση της απόδοσης και της κλιμακωσιμότητας του ασύγχρονου μηχανισμού παρακολούθησης (Monitoring Engine).
3. Ο έλεγχος και η επαλήθευση των μηχανισμών ασφάλειας, αυθεντικοποίησης (JWT) και εξουσιοδότησης (RBAC).
4. Τα συμπεράσματα των δοκιμών και τα τεχνικά σημεία προς μελλοντική βελτίωση.

6.1 Λειτουργικός Έλεγχος και Δοκιμές Ενοποίησης (Functional & Integration Testing)

Ο λειτουργικός έλεγχος σχεδιάστηκε για να επιβεβαιώσει ότι κάθε μεμονωμένο component της layered αρχιτεκτονικής λειτουργεί σωστά και συνεργάζεται ομαλά με τα υπόλοιπα.

6.1.1 Δοκιμές REST API με χρήση Postman

Για την επαλήθευση των REST endpoints του backend, δημιουργήθηκε μια ολοκληρωμένη συλλογή δοκιμών (Test Collection) στο Postman. Μέσω αυτής, ελέγχθηκαν συστηματικά:

- Endpoints Αυθεντικοποίησης: Δοκιμές για την έκδοση JWT tokens κατά το Login, έλεγχος λήξης του token και επαλήθευση της ροής Request Password Reset με αποστολή email.
- CRUD Λειτουργίες Inventory: Έλεγχος προσθήκης, ανάκτησης, ενημέρωσης και διαγραφής συσκευών. Επαληθεύτηκε ότι οι εισαγωγές πεδίων (π.χ. IP διευθύνσεις) ελέγχονται αυστηρά από τα Pydantic schemas.
- Bulk Import: Έλεγχος της επεξεργασίας του αρχείου test_devices.csv από το backend, διασφαλίζοντας την επιτυχή μαζική εισαγωγή συσκευών στη βάση δεδομένων.

6.1.2 Δοκιμές Σεναρίων End-to-End (E2E Data Flow)

Πραγματοποιήθηκε έλεγχος της πλήρους ροής δεδομένων στο σύστημα μέσω συγκεκριμένων σεναρίων:

- Ροή Παρακολούθησης και Real-time Ενημέρωσης:
 1. Ο scheduler εκκινεί έναν κύκλο παρακολούθησης.
 2. Οι background monitors συλλέγουν δεδομένα (ICMP ping, SNMP CPU/RAM, SSH stats).
 3. Οι μετρήσεις αποθηκεύονται στον πίνακα `device_metrics` της PostgreSQL.
 4. Οι μετρήσεις δημοσιεύονται (Publish) στο Redis κανάλι `device_updates`.
 5. Το WebSocket router λαμβάνει το μήνυμα και εκτελεί broadcast στο React frontend.
 6. Τα widgets και τα line charts (Chart.js) στο Dashboard ενημερώνονται δυναμικά.
- Ροή Ειδοποιήσεων (Alerting Flow):
 1. Προσομοιώθηκε η απώλεια σύνδεσης μιας συσκευής (offline) και η υπέρβαση ορίων (CPU utilization > 90%).
 2. Το backend ανίχνευσε την υπέρβαση βάσει των ορίων του πίνακα `system_settings`.
 3. Δημιουργήθηκαν εγγραφές alert στο UI (Live Alerts Pane) σε πραγματικό χρόνο.
 4. Στο τέλος του monitored κύκλου, συντέθηκε και απεστάλη ένα συγκεντρωτικό email ειδοποίησης μέσω SMTP στους χρήστες που είχαν ενεργοποιημένη τη σχετική επιλογή.
- Ροή Απομακρυσμένων Ενεργειών (Remote Execution Flow):
 1. Από το UI επιλέχθηκε η επανεκκίνηση της υπηρεσίας Nginx σε έναν εικονικό Linux server.
 2. Το backend έλαβε το αίτημα, ανακτώρησε τα κρυπτογραφημένα SSH credentials από τη βάση δεδομένων, τα αποκρυπτογράφησε στη μνήμη και εκτέλεσε την εντολή `sudo systemctl restart nginx` μέσω Paramiko.
 3. Η υπηρεσία επανεκκινήθηκε επιτυχώς και το αποτέλεσμα επιστράφηκε στο UI, ενώ η ενέργεια καταγράφηκε αυτόματα στον πίνακα `audit_logs`.

6.2 Αξιολόγηση Απόδοσης και Κλιμακωσιμότητας (Performance & Scalability Analysis)

Η αξιολόγηση της απόδοσης επικεντρώθηκε στην ικανότητα του συστήματος να διαχειρίζεται μεγάλο όγκο συσκευών και υψηλή συχνότητα polling χωρίς να επιβαρύνεται η απόκριση του REST API.

6.2.1 Scaling Test του Monitoring Engine

Το κρίσιμο σημείο απόδοσης είναι ο χρόνος ολοκλήρωσης ενός κύκλου polling. Εάν οι συσκευές ελέγχονταν σειριακά (synchronous polling), ο χρόνος θα αυξανόταν γραμμικά:

$$T_{total} = t_1 + t_2 + \dots + t_N$$

Όπου N ο αριθμός των συσκευών και t_i ο χρόνος επικοινωνίας με κάθε συσκευή (π.χ. 2 δευτερόλεπτα timeout σε περίπτωση offline συσκευής). Για 50 συσκευές, ένας σειριακός

έλεγχος θα απαιτούσε έως και 100 δευτερόλεπτα, καθιστώντας αδύνατη την παρακολούθηση με polling interval 60s.

Με τη χρήση της βιβλιοθήκης `asyncio` και της μεθόδου `asyncio.gather()`, το backend εκτελεί το polling όλων των συσκευών παράλληλα (asynchronous I/O). Πραγματοποιήθηκε δοκιμή κλιμάκωσης με προσομοίωση εικονικών συσκευών (mock nodes):

Αριθμός Συσκευών	Πρωτόκολλα Ελέγχου	Χρόνος Ολοκλήρωσης Κύκλου (sec)
10	ICMP + SNMP	1,15
25	ICMP + SNMP + SSH	2,05
50	ICMP + SNMP + SSH	2,12
100	ICMP + SNMP + SSH	2,24

Πίνακας 1 Χρόνος ολοκλήρωσης κύκλου polling σε συνάρτηση με τον αριθμό των συσκευών

Τα αποτελέσματα δείχνουν ότι ο χρόνος ολοκλήρωσης παραμένει σχεδόν σταθερός (λίγο πάνω από τα 2 δευτερόλεπτα, που αποτελεί το default timeout του ping), αποδεικνύοντας την εξαιρετική κλιμακωσιμότητα του ασύγχρονου scheduler.

6.2.2 Response Time του REST API

Μέσα από τη χρήση του Postman Runner, μετρήθηκε ο χρόνος απόκρισης του REST API υπό φορτίο. Τα endpoints ανάκτησης δεδομένων (όπως το `/api/devices`) εμφάνισαν μέσο χρόνο απόκρισης μικρότερο από 45ms για 100 ταυτόχρονες αιτήσεις. Αυτό οφείλεται στο γεγονός ότι το FastAPI εκτελείται σε asynchronous event loop και οι βαριές εργασίες επικοινωνίας με τις συσκευές έχουν απομονωθεί πλήρως από τα API routes, εκτελούμενες στο background.

6.3 Έλεγχος και Επαλήθευση Ασφαλείας (Security Audit)

Η ασφάλεια του συστήματος επαληθεύτηκε μέσω στοχευμένων ελέγχων για τη διασφάλιση της εμπιστευτικότητας, της ακεραιότητας και της ελεγχόμενης πρόσβασης.

6.3.1 Επαλήθευση Κρυπτογράφησης Credentials στη Βάση

Πραγματοποιήθηκε απευθείας έλεγχος στον πίνακα `device_credentials` της PostgreSQL. Εκτελώντας SQL ερώτημα `'SELECT password, community_string FROM device_credentials'`, επιβεβαιώθηκε ότι οι κωδικοί πρόσβασης SSH και τα SNMP community strings είναι αποθηκευμένα ως κρυπτογραφημένα ciphertexts (συμμετρική κρυπτογράφηση

Fernet AES-256) και όχι σε plaintext μορφή. Η αποκρυπτογράφηση πραγματοποιείται αποκλειστικά στη μνήμη του server κατά τη διάρκεια του polling.

6.3.2 Έλεγχος Role-Based Access Control (RBAC)

Ελέγχθηκε η συμπεριφορά του backend και του frontend σε προσπάθειες μη εξουσιοδοτημένης πρόσβασης:

- **Δοκιμή API Bypass:** Στάλθηκε POST αίτημα για επανεκκίνηση συσκευής στο endpoint `/api/devices/{id}/reboot` χρησιμοποιώντας JWT token ενός χρήστη με ρόλο viewer. Το backend απέρριψε άμεσα το αίτημα επιστρέφοντας *HTTP Status 403 Forbidden*, επιβεβαιώνοντας την ορθή λειτουργία του middleware εξουσιοδότησης.
- **Δοκιμή Frontend Route Guard:** Επιχειρήθηκε απευθείας πληκτρολόγηση της διεύθυνσης `/users` στον browser από λογαριασμό με ρόλο operator. Το frontend, μέσω του Route Guard στο `/frontend/src/App.jsx`, εντόπισε ότι ο ρόλος δεν είναι admin, απέκλεισε την πρόσβαση και ανακατεύθυνε τον χρήστη στο Dashboard.

6.4 Αξιολόγηση Χρηστικότητας από Χρήστες

Πέραν των τεχνικών δοκιμών που παρουσιάστηκαν στις προηγούμενες ενότητες, πραγματοποιήθηκε και αξιολόγηση της χρηστικότητας (usability) του συστήματος με τη συμμετοχή πραγματικών χρηστών. Στόχος της διαδικασίας ήταν η αποτίμηση της ευκολίας εκμάθησης και χρήσης της εφαρμογής, καθώς και ο εντοπισμός σημείων που χρήζουν βελτίωσης από την οπτική του τελικού χρήστη, στοιχεία που δεν μπορούν να προκύψουν αποκλειστικά από αυτοματοποιημένους τεχνικούς ελέγχους.

6.4.1 Μεθοδολογία

Για την αξιολόγηση επιλέχθηκε η μέθοδος της δοκιμής χρηστικότητας βάσει σεναρίου (scenario-based usability testing) σε συνδυασμό με το τυποποιημένο ερωτηματολόγιο System Usability Scale (SUS) του Brooke (1996). Το SUS αποτελεί ένα ευρέως αναγνωρισμένο και επιστημονικά επικυρωμένο εργαλείο μέτρησης της αντιλαμβανόμενης χρηστικότητας, το οποίο αποτελείται από δέκα προτάσεις βαθμολογούμενες σε πενταβάθμια κλίμακα (1 = «Διαφωνώ απόλυτα» έως 5 = «Συμφωνώ απόλυτα»), με εναλλασσόμενη θετική και αρνητική διατύπωση.

Στη μελέτη συμμετείχαν πέντε (5) χρήστες. Το μέγεθος του δείγματος είναι σύμφωνο με την καθιερωμένη πρακτική στις μελέτες χρηστικότητας, καθώς σύμφωνα με τον Nielsen ένα δείγμα πέντε χρηστών επαρκεί για τον εντοπισμό της πλειονότητας των προβλημάτων χρηστικότητας μιας διεπαφής. Οι συμμετέχοντες προέρχονταν τόσο από ακαδημαϊκό όσο και από επαγγελματικό υπόβαθρο της πληροφορικής, ώστε να καλυφθεί ένα εύρος εμπειρίας με εργαλεία παρακολούθησης δικτύου.

Η διαδικασία για κάθε συμμετέχοντα περιλάμβανε τα εξής στάδια: αρχικά, ο χρήστης κλήθηκε να ολοκληρώσει ένα προκαθορισμένο σύνολο αντιπροσωπευτικών εργασιών στο πλήρως λειτουργικό σύστημα, το οποίο εκτελούνταν σε περιβάλλον Docker. Οι εργασίες

περιλάμβαναν: είσοδο στο σύστημα, ανάγνωση της κατάστασης του δικτύου από τον κεντρικό πίνακα ελέγχου, προσθήκη και επεξεργασία δικτυακής συσκευής, τροποποίηση των ορίων (thresholds) στις ρυθμίσεις, διαχείριση χρηστών και αναζήτηση εγγραφών στις καταγραφές ασφαλείας (audit logs). Μετά την ολοκλήρωση των εργασιών, κάθε συμμετέχων συμπλήρωσε ανώνυμα το ερωτηματολόγιο SUS καθώς και τρεις ανοιχτές ερωτήσεις ποιοτικού χαρακτήρα.

6.4.2 Προφίλ Συμμετεχόντων

Συμμετέχων	Επαγγελματικό Υπόβαθρο	Εμπειρία με εργαλεία Monitoring	Ηλικιακή Ομάδα
Σ1	Φοιτητής Πληροφορικής	Καμία	18-25
Σ2	Φοιτητής Πληροφορικής	Περιορισμένη	26-35
Σ3	Διαχειριστής Δικτύου	Εκτεταμένη	36-45
Σ4	Προγραμματιστής	Μέτρια	26-35
Σ5	Μηχανικός Λογισμικού	Περιορισμένη	26-35

Πίνακας 2 Δημογραφικά στοιχεία και υπόβαθρο των συμμετεχόντων στην αξιολόγηση χρηστικότητας

6.4.3 Ποσοτικά Αποτελέσματα

Η βαθμολόγηση του SUS πραγματοποιείται ως εξής: για τις προτάσεις με περιττό αριθμό (θετική διατύπωση) η συνεισφορά κάθε απάντησης είναι η βαθμολογία μείον ένα, ενώ για τις προτάσεις με άρτιο αριθμό (αρνητική διατύπωση) η συνεισφορά είναι το πέντε μείον τη βαθμολογία. Το άθροισμα των δέκα τιμών πολλαπλασιάζεται επί 2,5, παράγοντας μια τελική βαθμολογία στην κλίμακα 0–100. Στον παρακάτω πίνακα παρουσιάζονται οι επιμέρους βαθμολογίες ανά συμμετέχοντα.

Συμμετέχων	Σ1	Σ2	Σ3	Σ4	Σ5	M.O.
Βαθμολογία SUS	77.5	87.5	85.0	70.0	87.5	81.5

Πίνακας 3 Επιμέρους και μέση βαθμολογία SUS των συμμετεχόντων

Η μέση βαθμολογία SUS ανήλθε σε 81,5 (τυπική απόκλιση 7,62, εύρος 70,0–87,5). Δεδομένου ότι ο μέσος όρος αναφοράς για το SUS στη βιβλιογραφία είναι περίπου 68, η επιτευχθείσα βαθμολογία τοποθετεί την εφαρμογή σημαντικά πάνω από τον μέσο όρο. Σύμφωνα με την ερμηνευτική κλίμακα των Bangor et al., μια βαθμολογία άνω του 80 αντιστοιχεί στον χαρακτηρισμό «Excellent» (βαθμός A) και σε ποσοστημόριο κοντά στο 90%. Αξίζει να σημειωθεί ότι ακόμη και ο χρήστης χωρίς προηγούμενη εμπειρία με αντίστοιχα εργαλεία (Σ1) απέδωσε βαθμολογία 77,5, γεγονός που επιβεβαιώνει τη χαμηλή καμπύλη εκμάθησης του συστήματος.

6.4.4 Ποιοτικά Αποτελέσματα

Οι απαντήσεις στις ανοιχτές ερωτήσεις παρείχαν χρήσιμη ποιοτική ανατροφοδότηση, η οποία συνοψίζεται ως εξής:

- Θετικά σημεία: Οι συμμετέχοντες αξιολόγησαν θετικά την ευκολία χρήσης (αναφέρθηκε από δύο χρήστες) και τη σχεδίαση της διεπαφής (UI). Επιπλέον, επισημάνθηκαν η ασφαλής αποθήκευση των ευαίσθητων δεδομένων (κρυπτογραφημένα credentials) και οι ειδοποιήσεις σε πραγματικό χρόνο (real-time notifications) ως ιδιαίτερα ισχυρά χαρακτηριστικά.
- Δυσκολίες: Οι τρεις από τους πέντε συμμετέχοντες δεν ανέφεραν καμία δυσκολία κατά τη χρήση. Τα δύο σημεία τριβής που εντοπίστηκαν αφορούσαν τη σωστή καταχώρηση των credentials μιας συσκευής και τη διαμόρφωση των ρυθμίσεων SMTP για τις ειδοποιήσεις μέσω email, διαδικασίες με αυξημένη τεχνική πολυπλοκότητα.
- Προτάσεις βελτίωσης: Οι προτάσεις των χρηστών περιλάμβαναν την προσθήκη περισσότερων γραφημάτων και επιπλέον λειτουργιών, την υποστήριξη αυθεντικοποίησης SSH με ιδιωτικό κλειδί (private key) αντί για συνδυασμό ονόματος χρήστη/κωδικού, την απλούστευση της διαδικασίας ρύθμισης SMTP, καθώς και την ενσωμάτωση λειτουργιών τεχνητής νοημοσύνης για προηγμένη ανάλυση δεδομένων (AI analytics).

6.4.5 Συμπεράσματα της Αξιολόγησης

Συνολικά, η αξιολόγηση από χρήστες επιβεβαίωσε ότι το σύστημα διαθέτει υψηλό βαθμό χρηστικότητας και χαμηλή καμπύλη εκμάθησης, με μέση βαθμολογία SUS 81,5 να κατατάσσει την εφαρμογή στην κατηγορία «Excellent». Τα ποιοτικά σχόλια ανέδειξαν ως κύρια θετικά στοιχεία την απλότητα της διεπαφής, την ασφάλεια και τις ειδοποιήσεις πραγματικού χρόνου, ενώ τα σημεία βελτίωσης (απλούστευση ρύθμισης SMTP και credentials, εμπλουτισμός γραφημάτων, υποστήριξη SSH key authentication, AI analytics) είναι απολύτως συμβατά με τις μελλοντικές κατευθύνσεις που παρουσιάζονται στην ενότητα 7.2. Παρά το περιορισμένο μέγεθος του δείγματος, το οποίο αποτελεί και βασικό περιορισμό της παρούσας αξιολόγησης, τα αποτελέσματα παρέχουν ισχυρές ενδείξεις για την αποδοχή και τη λειτουργική αρτιότητα του συστήματος από την πλευρά των τελικών χρηστών.

6.5 Συμπεράσματα Δοκιμών και Τεχνικές Βελτιώσεις

Οι τεχνικές δοκιμές επιβεβαίωσαν ότι το σύστημα είναι σταθερό, εξαιρετικά γρήγορο στην επικοινωνία του με τις δικτυακές συσκευές και ασφαλές. Η χρήση ασύγχρονου προγραμματισμού (FastAPI + asyncio) και η ενοποίηση του Redis Pub/Sub με WebSockets

προσφέρει εξαιρετική απόδοση σε πραγματικό χρόνο με ελάχιστη κατανάλωση πόρων συστήματος.

Ωστόσο, κατά τη διαδικασία των δοκιμών εντοπίστηκαν ορισμένα σημεία που επιδέχονται τεχνική βελτίωση:

- **Δημιουργία Unit Tests με pytest:** Προτείνεται η ανάπτυξη μιας σουίτας αυτοματοποιημένων unit tests για τις υπηρεσίες κρυπτογράφησης `/backend/services/encryptor.py` και υπολογισμού του `traffic rate`.
- **Βελτιστοποίηση Ευρετηρίων (Indexes) στη Βάση:** Για περιβάλλοντα με εκατοντάδες συσκευές, η συσσώρευση ιστορικών δεδομένων στον πίνακα `device_metrics` μπορεί να επιβραδύνει τα queries των γραφημάτων. Προτείνεται η δημιουργία σύνθετων ευρετηρίων (composite indexes) στα πεδία `device_id`, `timestamp`.
- **Υποστήριξη SNMP Bulk GET:** Για συσκευές με μεγάλο αριθμό interfaces, η χρήση απλών SNMP GET requests μπορεί να αντικατασταθεί με SNMP BULK GET, μειώνοντας ακόμη περισσότερο τα δίκτυακά πακέτα.

7. Συμπεράσματα και Μελλοντικές Κατευθύνσεις

Το κεφάλαιο αυτό αποσκοπεί στην τεχνική και λειτουργική αποτίμηση των αποτελεσμάτων της παρούσας διπλωματικής εργασίας, καθώς και στην παρουσίαση των κατευθύνσεων για τη μελλοντική εξέλιξη και επέκταση του συστήματος. Μέσα από τη σχεδίαση, την ανάπτυξη και τη δοκιμαστική αξιολόγηση της web-based πλατφόρμας παρακολούθησης και διαχείρισης δικτυακών συσκευών, επιβεβαιώθηκε η δυνατότητα δημιουργίας ενός σύγχρονου, ασφαλούς, εξαιρετικά αποδοτικού και εύχρηστου εργαλείου, προσαρμοσμένου στις ανάγκες διαχειριστών δικτύου και μηχανικών συστημάτων. Στην ενότητα που ακολουθεί γίνεται ανασκόπηση των στόχων που επιτεύχθηκαν και αναλύονται συγκεκριμένες προτάσεις για τη λειτουργική και τεχνολογική αναβάθμιση της εφαρμογής.

7.1 Αποτίμηση και Συμπεράσματα

Η ολοκλήρωση της ανάπτυξης και της αξιολόγησης του συστήματος οδήγησε σε σημαντικά συμπεράσματα σχετικά με την αρχιτεκτονική του και τον βαθμό επίτευξης των αρχικών προδιαγραφών:

- **Αποτελεσματικότητα Ασύγχρονης Σχεδίασης:** Η επιλογή του FastAPI σε συνδυασμό με την ασύγχρονη βιβλιοθήκη `asyncio` (`asyncio.gather()`) αποδείχθηκε εξαιρετικά επιτυχής. Η παράλληλη εκτέλεση του polling (ICMP ping, SNMP, SSH) απέτρεψε τις γραμμικές καθυστερήσεις που χαρακτηρίζουν τα σειριακά συστήματα, διατηρώντας τον χρόνο ολοκλήρωσης του κύκλου σταθερό (κοντά στα 2 δευτερόλεπτα) ακόμη και για 100 συσκευές. Επιπλέον, η απομόνωση των βαριών background διεργασιών από τα REST routes επέτρεψε στο API να ανταποκρίνεται σε λιγότερο από 45ms υπό φορτίο.
- **Real-time Ενημέρωση και Ροή Δεδομένων:** Η ενοποίηση του Redis Pub/Sub με τις συνδέσεις WebSockets επέτρεψε την άμεση αναμετάδοση των νέων μετρικών απόδοσης στο React frontend. Το Dashboard ενημερώνεται δυναμικά χωρίς την

ανάγκη χειροκίνητης ανανέωσης, προσφέροντας μια εξαιρετική εμπειρία χρήσης (UX) και άμεση εποπτεία της κατάστασης του δικτύου.

- **Ενισχυμένη Ασφάλεια:** Ο έλεγχος ασφαλείας επιβεβαίωσε την αποτελεσματικότητα των μηχανισμών προστασίας. Τα credentials των συσκευών (SSH κωδικοί και SNMP community strings) αποθηκεύονται στη βάση δεδομένων αποκλειστικά ως κρυπτογραφημένα ciphertexts (Fernet AES-256 με key derivation από το .env), ενώ η πρόσβαση στα API endpoints και τις frontend σελίδες ελέγχεται αυστηρά μέσω JWT tokens και Role-Based Access Control (RBAC).
- **Χρηστικότητα και Λειτουργική Σχεδίαση:** Η αξιολόγηση χρηστικότητας που διενεργήθηκε με πραγματικούς χρήστες (Ενότητα 6.4) επιβεβαίωσε ότι η εφαρμογή διαθέτει πολύ χαμηλή καμπύλη εκμάθησης, με μέση βαθμολογία SUS 81,5 («Excellent»). Λειτουργίες όπως η μαζική εισαγωγή (Bulk Import) μέσω CSV, οι range sliders με ενσωματωμένο constraint validation στα Settings, και η αποστολή συγκεντρωτικών email ειδοποιήσεων ανά monitoring cycle συνέβαλαν καθοριστικά στην αποτροπή λαθών και στη μείωση του alert fatigue για τους διαχειριστές.

Συνολικά, η εργασία πέτυχε τον σχεδιασμό και την υλοποίηση ενός πλήρους, λειτουργικού και κλιμακώσιμου συστήματος διαχείρισης δικτύου, το οποίο μπορεί να υποστηρίξει ρεαλιστικές επιχειρησιακές ανάγκες με ελάχιστη κατανάλωση πόρων.

7.2 Μελλοντικές Κατευθύνσεις και Επεκτάσεις

Παρά την αρτιότητα της τρέχουσας έκδοσης, η διαδικασία των δοκιμών ανέδειξε ορισμένες σημαντικές προοπτικές για τη μελλοντική βελτίωση και επέκταση της εφαρμογής:

- **Ανάπτυξη Διαδραστικού Χάρτη Τοπολογίας (Network Topology Map):** Προτείνεται η προσθήκη μιας σελίδας οπτικοποίησης της δικτυακής τοπολογίας (π.χ. με χρήση της βιβλιοθήκης Vis.js ή D3.js). Η γραφική απεικόνιση των συνδέσεων μεταξύ routers, switches και servers θα διευκολύνει τους διαχειριστές στον άμεσο εντοπισμό του κρίσιμου σημείου αστοχίας (root cause analysis) σε περίπτωση πολλαπλών offline συσκευών.
- **Αυτοματοποιημένη Ανακάλυψη Συσκευών (Auto-Discovery):** Αντί για τη χειροκίνητη εισαγωγή ή το bulk import μέσω CSV, μπορεί να αναπτυχθεί ένας background μηχανισμός σάρωσης (subnet scanning) που θα εκτελεί ICMP sweeps ή SNMP sweeps σε συγκεκριμένα IP ranges για τον αυτόματο εντοπισμό και την καταχώρηση νέων συσκευών στο inventory.
- **Υποστήριξη SNMP v3:** Για τη χρήση της εφαρμογής σε περιβάλλοντα με αυστηρές προδιαγραφές ασφαλείας, κρίνεται αναγκαία η αναβάθμιση του monitoring engine ώστε να υποστηρίζει το πρωτόκολλο SNMP v3, το οποίο παρέχει αυθεντικοποίηση χρηστών και κρυπτογράφηση των πακέτων διαχείρισης, αποφεύγοντας τη μετάδοση plaintext community strings.
- **Push Notifications και Mobile Integrations:** Προτείνεται η ενσωμάτωση εναλλακτικών καναλιών ειδοποίησης, όπως η αποστολή push notifications σε κινητές συσκευές μέσω του Web Push API ή η δημιουργία integrations (webhooks) με δημοφιλείς πλατφόρμες επικοινωνίας (π.χ. Telegram bots, Slack channels) για την άμεση ενημέρωση των μηχανικών εκτός γραφείου.

- **Προβλέψεις και Ανίχνευση Ανωμαλιών μέσω Μηχανικής Μάθησης (AI/ML):** Μία σημαντική επέκταση αποτελεί η ενσωμάτωση ελαφριών μοντέλων μηχανικής μάθησης (π.χ. Prophet ή LSTM νευρωνικά δίκτυα). Τα μοντέλα αυτά θα αναλύουν τις ιστορικές μετρήσεις απόδοσης για να προβλέψουν μελλοντική εξάντληση πόρων (π.χ. πότε η μνήμη ενός server θα ξεπεράσει το 90%) ή να ανιχνεύσουν ανώμαλες μεταβολές στην κίνηση του δικτύου (anomaly detection).
- **Αρχιτεκτονική SaaS και Πολυενοικία (Multi-Tenancy):** Για την εμπορική αξιοποίηση της εφαρμογής, προτείνεται η μετατροπή της σε multi-tenant Cloud SaaS. Αυτό θα επιτρέπει σε διαφορετικούς οργανισμούς να χρησιμοποιούν την ίδια κεντρική εγκατάσταση της εφαρμογής, διαχειριζόμενοι ο καθένας το δικό του απομονωμένο inventory συσκευών και χρηστών.

Βιβλιογραφία

Ακολουθούν οι βιβλιογραφικές αναφορές (πηγές) της Εργασίας.

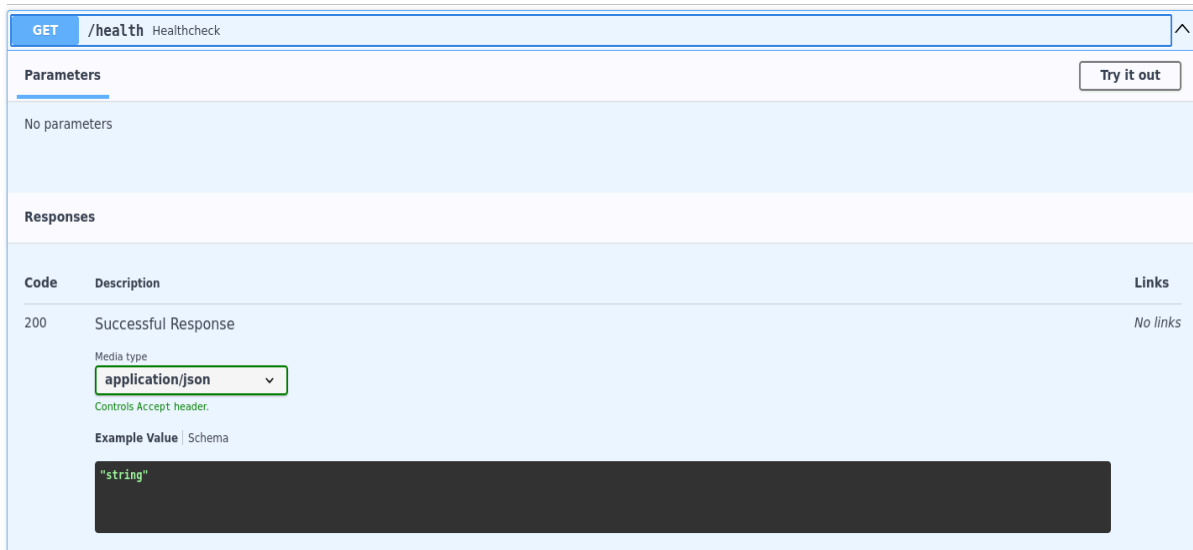
- FastAPI Documentation. Sebastián Ramírez. <https://fastapi.tiangolo.com/>
- React – A JavaScript library for building user interfaces. Meta Platforms, Inc. <https://reactjs.org/>
- PostgreSQL: The world's most advanced open source relational database. PostgreSQL Global Development Group. <https://www.postgresql.org/>
- Redis – In-memory data structure store. Redis Labs, Inc. <https://redis.io/>
- Docker – Enterprise container platform. Docker, Inc. <https://www.docker.com/>
- Python Language Reference, version 3.11. Python Software Foundation. <https://docs.python.org/3/reference/>
- TypeScript – JavaScript with syntax for types. Microsoft. <https://www.typescriptlang.org/>
- Pydantic – Data validation using Python type annotations. Samuel Colvin. <https://pydantic-docs.helpmanual.io/>
- SQLAlchemy – The Python SQL Toolkit and Object Relational Mapper. Mike Bayer. <https://www.sqlalchemy.org/>
- Uvicorn – Lightning-fast ASGI server. Tom Christie. <https://www.uvicorn.org/>
- Gunicorn – Python WSGI HTTP Server for UNIX. Benoît Chesaux. <https://gunicorn.org/>
- Massé, M. (2011). REST API Design Rulebook. O'Reilly Media.
- Bowman, J. S., Emerson, S. L., & Darnovsky, M. (2001). The Practical SQL Handbook: Using SQL Variants. Addison-Wesley Professional.
- Elmasri, R., & Navathe, S. B. (2015). Fundamentals of Database Systems. Pearson.
- Sinha, K. (2016). Structured Query Language For All RDBMS And PL/SQL. BPB Publications.
- Fowler, M. (2003). Data Transfer Object. Patterns of Enterprise Application Architecture.
- Fowler, M. (2002). Patterns of Enterprise Application Architecture. Addison-Wesley Professional.
- Bassett, L. (2015). Introduction to JavaScript Object Notation: A To-the-Point Guide to JSON. O'Reilly Media.
- Keller, S. (2016). JSON Book: Easy Learning of JavaScript Standard Object Notation. CreateSpace Independent Publishing.
- pytest – Powerful testing framework for Python. <https://docs.pytest.org/>
- Git – Distributed version control system. <https://git-scm.com/>
- GitHub – Platform for collaborative software development. <https://github.com/>
- Agile Manifesto – Principles of Agile Software Development. <https://agilemanifesto.org/>
- UML – Unified Modeling Language. OMG. <https://www.omg.org/spec/UML/>
- SNMP – Simple Network Management Protocol (RFC 3411-3418). <https://www.rfc-editor.org/rfc/rfc3411>
- WebSockets – Full-duplex communication over a single TCP socket. https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API
- JSON Web Token (JWT) – Open standard for secure token exchange. <https://jwt.io/>
- Docker Compose – Define and run multi-container Docker applications. <https://docs.docker.com/compose/>
- CI/CD – Continuous Integration/Continuous Deployment concepts. <https://www.atlassian.com/continuous-delivery>

Kubernetes – Production-grade container orchestration. <https://kubernetes.io/>
Apache Kafka – Distributed event streaming platform. <https://kafka.apache.org/>
OpenAPI Specification – Standard for describing RESTful APIs.
<https://swagger.io/specification/>
GraphQL – Data query and manipulation language for APIs. <https://graphql.org/>
Kleppmann, M. (2017). Designing Data-Intensive Applications. O'Reilly Media.
Martin, R. C. (2017). Clean Architecture: A Craftsman's Guide to Software Structure and Design. Prentice Hall.
Jain, R. (1991). The Art of Computer Systems Performance

Παράρτημα Α: Swagger API Documentation

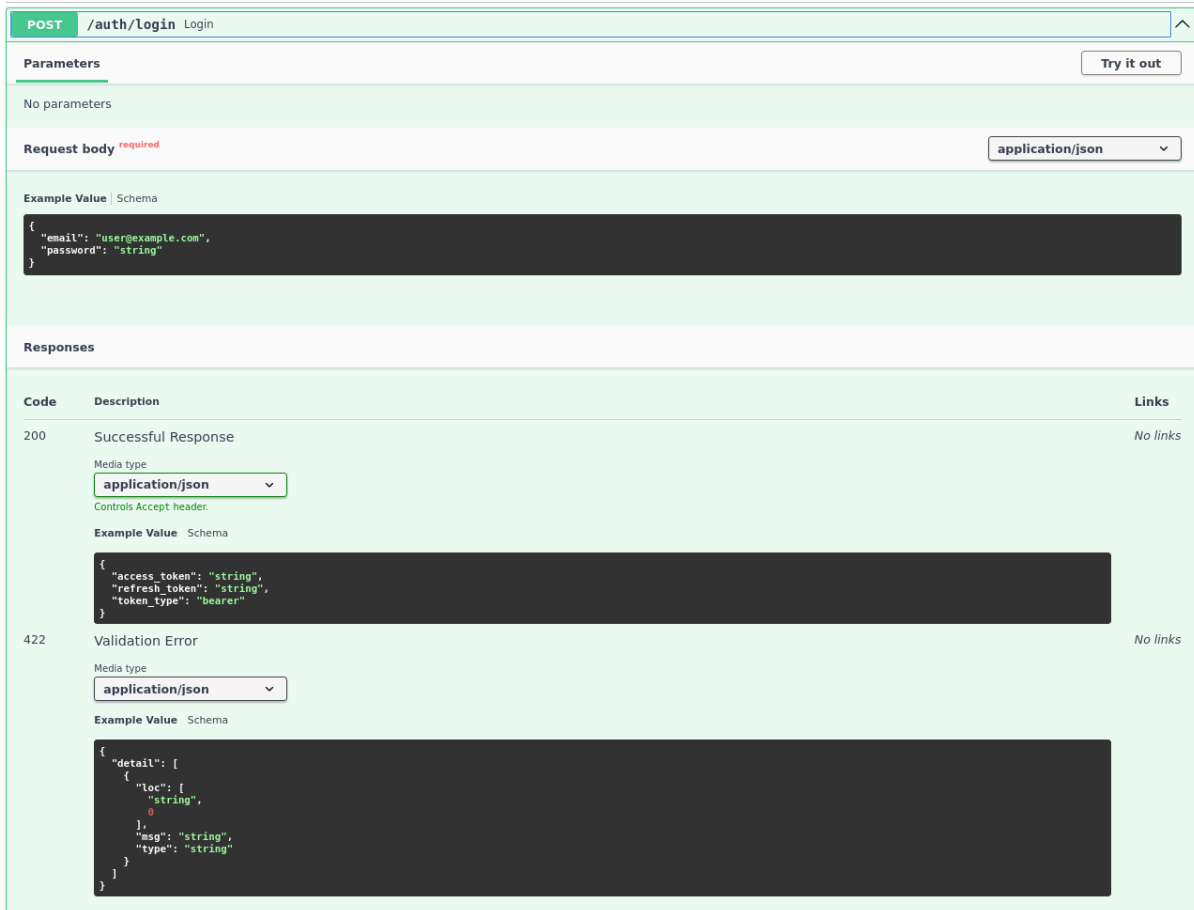
Η εφαρμογή παρέχει αυτόματα προδιαγραφή OpenAPI (Swagger) μέσω του FastAPI.

Health endpoint:



The screenshot shows the Swagger UI for the `GET /health` endpoint, labeled "Healthcheck". The "Parameters" section indicates "No parameters". The "Responses" section shows a 200 status code for a "Successful Response". A dropdown menu for "Media type" is set to "application/json". Below this, there is a section for "Example Value" and "Schema" which displays the string `"string"`.

Auth endpoints:



The image shows a Swagger UI interface for the `POST /auth/login` endpoint. The interface is divided into several sections: **Parameters** (No parameters), **Request body** (required, `application/json`), **Example Value** (Schema), and **Responses**.

Request body (required) `application/json`

Example Value | Schema

```
{
  "email": "user@example.com",
  "password": "string"
}
```

Responses

Code	Description	Links
200	Successful Response Media type: <code>application/json</code> Controls Accept header. Example Value Schema <pre>{ "access_token": "string", "refresh_token": "string", "token_type": "bearer" }</pre>	No links
422	Validation Error Media type: <code>application/json</code> Example Value Schema <pre>{ "detail": [{ "loc": ["string", 0], "msg": "string", "type": "string" }] }</pre>	No links

POST /auth/refresh Refresh Token

Parameters Try it out

No parameters

Request body **required** application/json

Example Value | Schema

```
{
  "additionalProp1": {}
}
```

Responses

Code	Description	Links
200	Successful Response	No links
Media type application/json Controls Accept header. Example Value Schema <pre>{ "access_token": "string", "refresh_token": "string", "token_type": "bearer" }</pre>		
422	Validation Error	No links
Media type application/json Example Value Schema <pre>{ "detail": [{ "loc": ["string", 0], "msg": "string", "type": "string" }] }</pre>		

POST /auth/register Register  

Parameters Try it out

No parameters

Request body *required* application/json

Example Value | **Schema**

```
{
  "email": "user@example.com",
  "password": "string",
  "full_name": "string",
  "role": "admin"
}
```

Responses

Code	Description	Links
200	Successful Response	No links
Media type: application/json Controls Accept header. Example Value Schema <pre>{ "email": "user@example.com", "password": "string", "full_name": "string", "role": "admin" }</pre>		
422	Validation Error	No links
Media type: application/json Example Value Schema <pre>{ "detail": [{ "loc": ["string", 0], "msg": "string", "type": "string" }] }</pre>		

POST /auth/change-password Change Password  

Parameters Try it out

No parameters

Request body *required* application/json

Example Value | **Schema**

```
{
  "current_password": "string",
  "new_password": "string"
}
```

Responses

Code	Description	Links
204	Successful Response	No links
422	Validation Error	No links
Media type: application/json Example Value Schema <pre>{ "detail": [{ "loc": ["string", 0], "msg": "string", "type": "string" }] }</pre>		

POST /auth/forgot-password Forgot Password

Parameters Try it out

No parameters

Request body **required** application/json

Example Value | Schema

```
{
  "email": "user@example.com"
}
```

Responses

Code	Description	Links
200	Successful Response	No links
Media type: application/json Controls Accept header.		
Example Value Schema		
<pre>"string"</pre>		
422	Validation Error	No links
Media type: application/json		
Example Value Schema		
<pre>{ "detail": [{ "loc": ["string", "string", 0], "msg": "string", "type": "string" }] }</pre>		

POST /auth/reset-password Reset Password

Parameters Try it out

No parameters

Request body **required** application/json

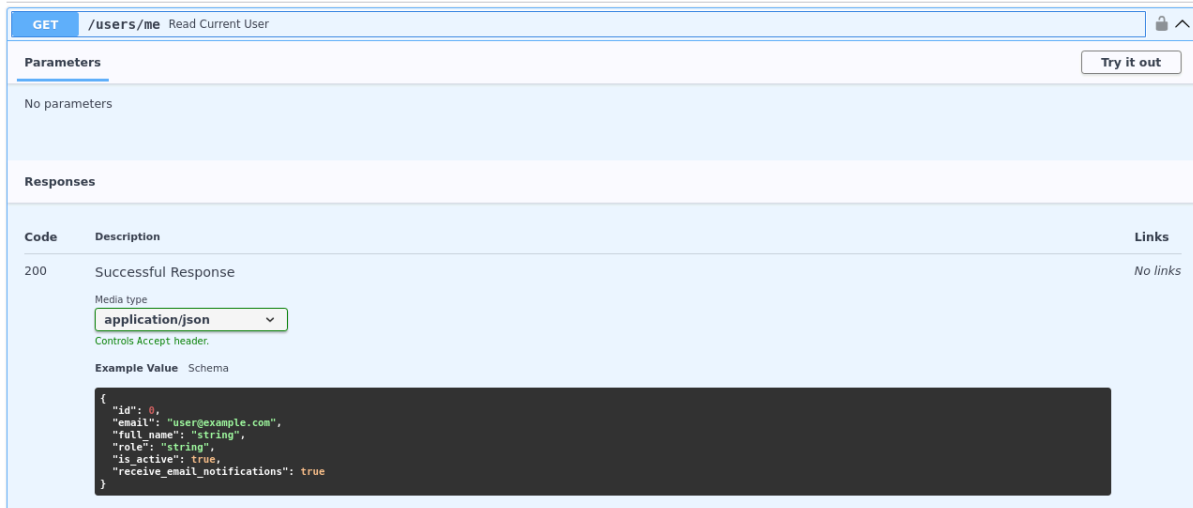
Example Value | Schema

```
{
  "token": "string",
  "new_password": "string"
}
```

Responses

Code	Description	Links
204	Successful Response	No links
422	Validation Error	No links
Media type: application/json		
Example Value Schema		
<pre>{ "detail": [{ "loc": ["string", "string", 0], "msg": "string", "type": "string" }] }</pre>		

Users endpoints:



GET /users/me Read Current User

Parameters

No parameters

Responses

Code	Description	Links
200	Successful Response	No links

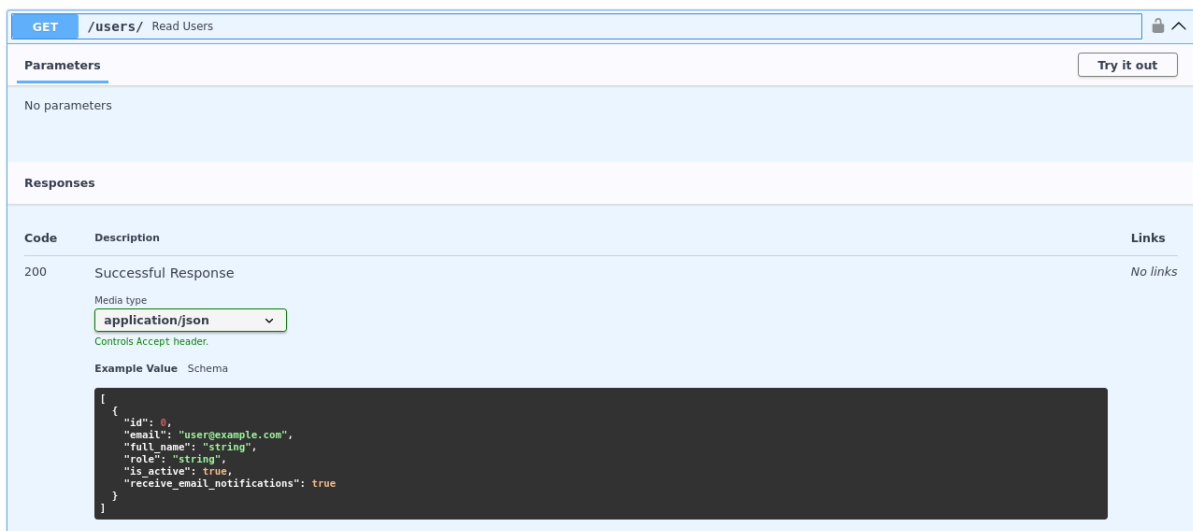
Media type

application/json

Controls Accept header.

Example Value Schema

```
{  "id": 0,  "email": "user@example.com",  "full_name": "string",  "role": "string",  "is_active": true,  "receive_email_notifications": true}
```



GET /users/ Read Users

Parameters

No parameters

Responses

Code	Description	Links
200	Successful Response	No links

Media type

application/json

Controls Accept header.

Example Value Schema

```
{  "id": 0,  "email": "user@example.com",  "full_name": "string",  "role": "string",  "is_active": true,  "receive_email_notifications": true}
```

GET
/users/{user_id}
Read User

Parameters
Try it out

Name	Description
user_id required integer (path)	user_id

Responses

Code	Description	Links
200	Successful Response Media type application/json Controls Accept header. Example Value Schema <pre> { "id": 0, "email": "user@example.com", "full_name": "string", "role": "string", "is_active": true, "receive_email_notifications": true } </pre>	No links
422	Validation Error Media type application/json Example Value Schema <pre> { "detail": [{ "loc": ["string", 0], "msg": "string", "type": "string" }] } </pre>	No links

PATCH /users/{user_id} Update User

Try it out

Name	Description
user_id * required integer (path)	user_id

Request body required

application/json

Example Value

Schema

```
{
  "full_name": "string",
  "email": "string",
  "role": "admin",
  "is_active": true,
  "receive_email_notifications": true
}
```

Responses

Code	Description	Links
200	Successful Response	No links
Media type application/json Controls Accept header. Example Value Schema <pre>{ "id": 0, "email": "user@example.com", "full_name": "string", "role": "string", "is_active": true, "receive_email_notifications": true }</pre>		
422	Validation Error	No links
Media type application/json Example Value Schema <pre>{ "detail": [{ "loc": ["string", 0], "msg": "string", "type": "string" }] }</pre>		

DELETE /users/{user_id} Delete User  

Parameters Try it out

Name	Description
user_id * required integer (path)	user_id

Responses

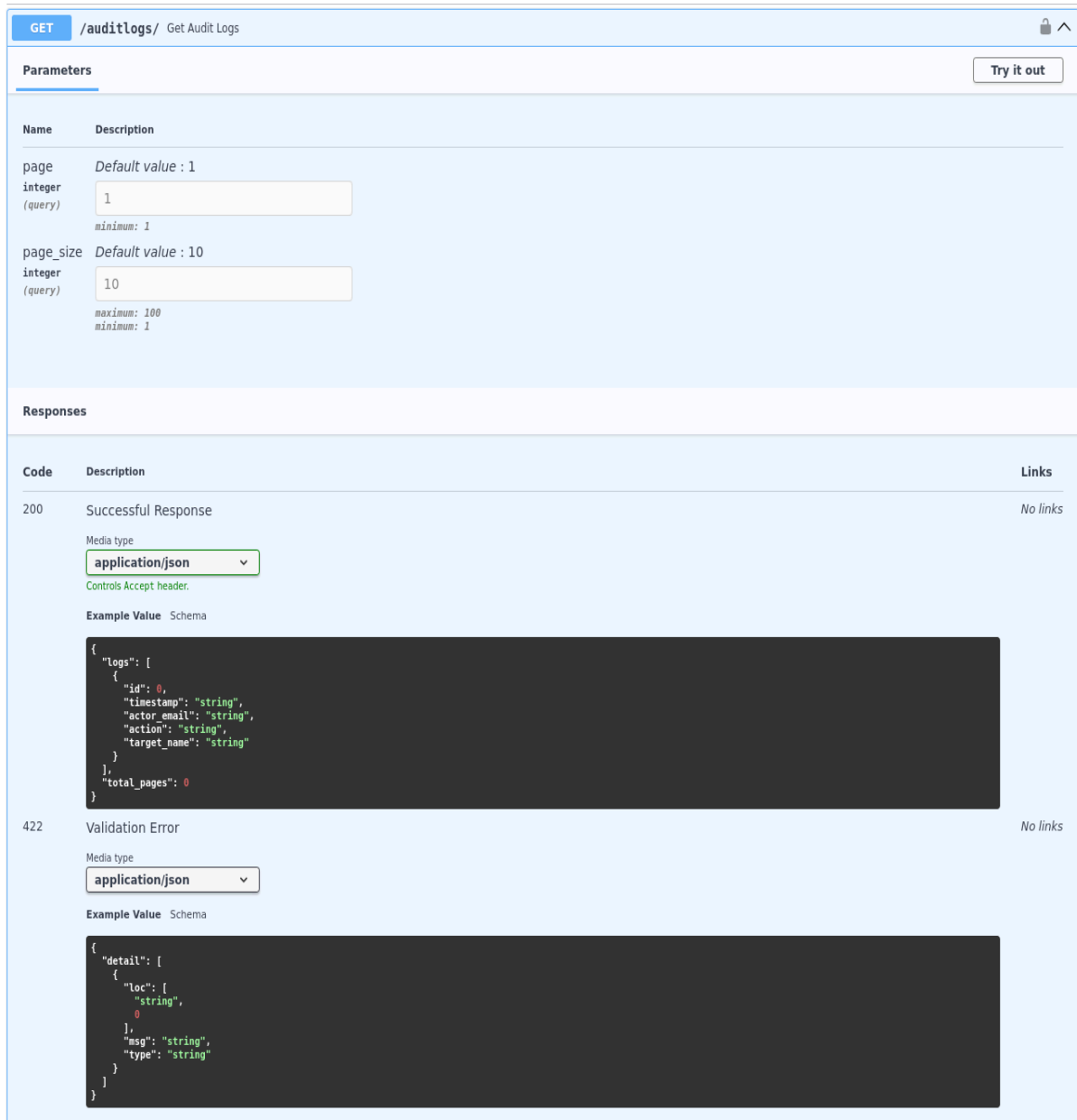
Code	Description	Links
204	Successful Response	No links
422	Validation Error	No links

Media type

Example Value Schema

```
{
  "detail": [
    {
      "loc": [
        "string",
        0
      ],
      "msg": "string",
      "type": "string"
    }
  ]
}
```

Audit Logs endpoint:



GET /auditlogs/ Get Audit Logs

Parameters

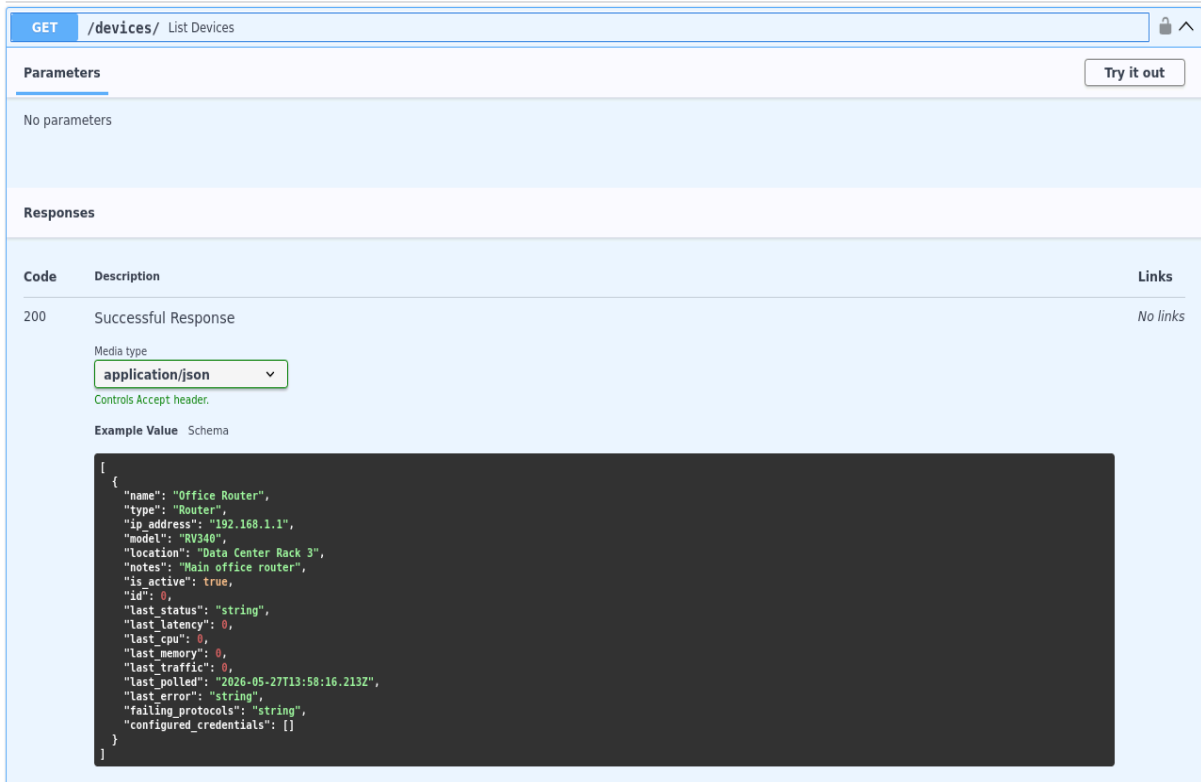
Try it out

Name	Description
page integer (query)	Default value : 1 minimum: 1
page_size integer (query)	Default value : 10 maximum: 100 minimum: 1

Responses

Code	Description	Links
200	Successful Response	No links
Media type: application/json		
Controls Accept header.		
Example Value Schema		
<pre>{ "logs": [{ "id": 0, "timestamp": "string", "actor_email": "string", "action": "string", "target_name": "string" }], "total_pages": 0}</pre>		
422	Validation Error	No links
Media type: application/json		
Example Value Schema		
<pre>{ "detail": [{ "loc": ["string", 0], "msg": "string", "type": "string" }]}</pre>		

Devices endpoints:



GET /devices/ List Devices

Parameters [Try it out](#)

No parameters

Responses

Code	Description	Links
200	Successful Response	No links

Media type:

Controls Accept header.

Example Value Schema

```
{
  "name": "Office Router",
  "type": "Router",
  "ip_address": "192.168.1.1",
  "model": "RV340",
  "location": "Data Center Rack 3",
  "notes": "Main office router",
  "is_active": true,
  "id": 0,
  "last_status": "string",
  "last_latency": 0,
  "last_cpu": 0,
  "last_memory": 0,
  "last_traffic": 0,
  "last_polled": "2026-05-27T13:58:16.213Z",
  "last_error": "string",
  "failing_protocols": "string",
  "configured_credentials": []
}
```

POST

/devices/ Create Device

Parameters

Try it out

No parameters

Request body required

application/json

Example Value | Schema

```
{
  "name": "Office Router",
  "type": "Router",
  "ip_address": "192.168.1.1",
  "model": "RV340",
  "location": "Data Center Rack 3",
  "notes": "Main office router",
  "is_active": true
}
```

Responses

Code	Description	Links
201	Successful Response Media type application/json Controls Accept header. Example Value Schema <pre>{ "name": "Office Router", "type": "Router", "ip_address": "192.168.1.1", "model": "RV340", "location": "Data Center Rack 3", "notes": "Main office router", "is_active": true, "id": 0, "last_status": "string", "last_latency": 0, "last_cpu": 0, "last_memory": 0, "last_traffic": 0, "last_polled": "2026-05-27T13:58:28.764Z", "last_error": "string", "failing_protocols": "string", "configured_credentials": [] }</pre>	

 No links || 422 | Validation Error Media type application/json Example Value | Schema ``` { "detail": [{ "loc": ["string", 0], "msg": "string", "type": "string" }] } ``` | No links |

GET

/devices/stats/trends

Get Network Trends

🔒 ⬆

Get aggregated network trends over a specific timeframe. Supported ranges: "24h", "7d", "30d"

Parameters

Try it out

Name	Description
range string (query)	Default value : 24h 24h

Responses

Code	Description	Links
200	Successful Response Media type application/json Controls Accept header. Example Value Schema <pre>{ "timestamp": "2026-05-27T13:59:11.704Z", "avg_cpu": 0, "avg_memory": 0, "avg_latency": 0, "avg_traffic": 0 }</pre>	

PATCH /devices/{device_id} Update Device

Parameters Try it out

Name	Description
device_id required	
integer (path)	device_id

Request body required application/json

Example Value | Schema

```
{
  "name": "string",
  "type": "Router",
  "ip_address": "string",
  "model": "string",
  "location": "string",
  "notes": "string",
  "is_active": true
}
```

Responses

Code	Description	Links
200	Successful Response	No links
Media type: application/json		
Controls Accept header.		
Example Value Schema		
<pre>{ "name": "Office Router", "type": "Router", "ip_address": "192.168.1.1", "model": "RV340", "location": "Data Center Rack 3", "notes": "Main office router", "is_active": true, "id": 0, "last_status": "string", "last_latency": 0, "last_cpu": 0, "last_memory": 0, "last_traffic": 0, "last_polled": "2026-05-27T13:59:26.337Z", "last_error": "string", "failing_protocols": "string", "configured_credentials": [] }</pre>		
422	Validation Error	No links
Media type: application/json		
Example Value Schema		
<pre>{ "detail": [{ "loc": ["string", 0], "msg": "string", "type": "string" }] }</pre>		

DELETE /devices/{device_id} Delete Device

Parameters Try it out

Name	Description
device_id * required integer (path)	device_id

Responses

Code	Description	Links
204	Successful Response	No links
422	Validation Error	No links

Media type

Example Value | Schema

```
{
  "detail": [
    {
      "loc": [
        "string",
        0
      ],
      "msg": "string",
      "type": "string"
    }
  ]
}
```

GET /devices/export Export Devices

Parameters Try it out

Name	Description
format string (query)	Default value : csv CSV

Responses

Code	Description	Links
200	Successful Response	No links
422	Validation Error	No links

Media type

Controls Accept header.

Example Value | Schema

```
"string"
```

Media type

Example Value | Schema

```
{
  "detail": [
    {
      "loc": [
        "string",
        0
      ],
      "msg": "string",
      "type": "string"
    }
  ]
}
```

POST /devices/import Import Devices

Bulk import devices from a CSV file. Required CSV columns: name, ip_address, type Optional columns: location, notes, is_active Returns a summary of imported and failed rows.

Parameters Try it out

No parameters

Request body required multipart/form-data

file required
string(binary)

Responses

Code	Description	Links
200	Successful Response Media type application/json Controls Accept header. Example Value Schema "string"	No links
422	Validation Error Media type application/json Example Value Schema <pre>{ "detail": [{ "loc": ["string", 0], "msg": "string", "type": "string" }] }</pre>	No links

GET

/devices/{device_id}/history/export

Export Device History

Parameters

Try it out

Name	Description
device_id required integer (path)	device_id
format string (query)	Default value : csv csv
range string (query)	Default value : 24h 24h

Responses

Code	Description	Links
200	Successful Response Media type application/json Controls Accept header. Example Value Schema "string"	No links
422	Validation Error Media type application/json Example Value Schema <pre>{ "detail": [{ "loc": ["string", 0], "msg": "string", "type": "string" }] }</pre>	No links

POST

/devices/{device_id}/credentials

Create Or Update Credential



Parameters

Try it out

Name	Description
device_id * required	device_id
integer	
(path)	

Request body required

application/json

Example Value

Schema

```
{
  "type": "ssh",
  "username": "string",
  "password": "string",
  "community_string": "string"
}
```

Responses

Code	Description	Links
200	Successful Response	No links
	Media type application/json Controls Accept header.	
	Example Value Schema	
	<pre>{ "type": "ssh", "device_id": 0 }</pre>	
422	Validation Error	No links
	Media type application/json	
	Example Value Schema	
	<pre>{ "detail": [{ "loc": ["string", 0], "msg": "string", "type": "string" }] }</pre>	

GET

/devices/{device_id}/history

Get Device History

Try it out

Name	Description
device_id required	
integer (path)	device_id
range	Default value : 24h
string (query)	24h
limit	Default value : 100
integer (query)	100

Responses

Code	Description	Links
200	Successful Response	No links
Media type application/json Controls Accept header.		
Example Value Schema		
<pre>{ "id": 0, "device_id": 0, "timestamp": "2026-05-27T14:01:27.418Z", "cpu_usage": 0, "memory_usage": 0, "traffic": 0, "latency": 0 }</pre>		
422	Validation Error	No links
Media type application/json		
Example Value Schema		
<pre>{ "detail": [{ "loc": ["string", 0], "msg": "string", "type": "string" }] }</pre>		

GET

/devices/{device_id}/services

List Device Services

🔒 ⬆

Parameters

Try it out

Name	Description
device_id ★ required	
integer (path)	device_id

Responses

Code	Description	Links
200	Successful Response	No links
Media type application/json Controls Accept header. Example Value Schema "string"		
422	Validation Error	No links
Media type application/json Example Value Schema <pre>{ "detail": [{ "loc": ["string", 0], "msg": "string", "type": "string" }] }</pre>		

POST

/devices/{device_id}/services/{service_name}/restart

Process Restart Service

🔒 ⬆

Parameters

Try it out

Name	Description
device_id * required integer (path)	device_id
service_name * required string (path)	service_name

Responses

Code	Description	Links
200	Successful Response Media type application/json Controls Accept header. Example Value Schema "string"	No links
422	Validation Error Media type application/json Example Value Schema <pre>{ "detail": [{ "loc": ["string", 0], "msg": "string", "type": "string" }] }</pre>	No links

POST /devices/{device_id}/reboot Process Reboot Device

Parameters

Try it out

Name	Description
device_id * required	
integer (path)	device_id

Responses

Code	Description	Links
200	Successful Response	No links
Media type: application/json		
Controls Accept header.		
Example Value	Schema	
<pre>"string"</pre>		
422	Validation Error	No links
Media type: application/json		
Controls Accept header.		
Example Value	Schema	
<pre>{ "detail": [{ "loc": ["string", 0], "msg": "string", "type": "string" }] }</pre>		

Settings endpoints:

GET /settings/ Get Settings

Parameters

Try it out

No parameters

Responses

Code	Description	Links
200	Successful Response	No links
Media type: application/json		
Controls Accept header.		
Example Value	Schema	
<pre>{ "polling_interval": 0, "cpu_warning_threshold": 0, "cpu_critical_threshold": 0, "memory_warning_threshold": 0, "memory_critical_threshold": 0, "latency_warning_threshold": 0, "id": 0 }</pre>		

PATCH /settings/ Update Settings

Parameters Try it out

No parameters

Request body required application/json

Example Value | Schema

```
{
  "polling_interval": 0,
  "cpu_warning_threshold": 0,
  "cpu_critical_threshold": 0,
  "memory_warning_threshold": 0,
  "memory_critical_threshold": 0,
  "latency_warning_threshold": 0
}
```

Responses

Code	Description	Links
200	Successful Response	No links
Media type application/json Controls Accept header.		
Example Value Schema <pre>{ "polling_interval": 0, "cpu_warning_threshold": 0, "cpu_critical_threshold": 0, "memory_warning_threshold": 0, "memory_critical_threshold": 0, "latency_warning_threshold": 0, "id": 0 }</pre>		
422	Validation Error	No links
Media type application/json		
Example Value Schema <pre>{ "detail": [{ "loc": ["string", 0], "msg": "string", "type": "string" }] }</pre>		

Schemas:

AuditLogOut Collapse all object

```
id* integer
timestamp* string
actor_email* string
action* string
target_name* > Expand all (string | null)
```

AuditLogPage Collapse all object

```
logs* > Expand all array<object>
total_pages* integer
```

Body_import_devices_devices_import_post Collapse all object

```
file* string binary
```

ChangePasswordRequest ^ Collapse all object

```
current_password* string
new_password* string
```

CredentialCreate ^ Collapse all object

```
type* > Expand all string
username > Expand all (string | null)
password > Expand all (string | null)
community_string > Expand all (string | null)
```

CredentialOut ^ Collapse all object

```
type* > Expand all string
device_id* integer
```

DeviceCreate ^ Collapse all object

```
name* > Expand all string
type* > Expand all string
ip_address* > Expand all string ipvanaddress
model > Expand all (string | null)
location > Expand all (string | null)
notes > Expand all (string | null)
is_active > Expand all (boolean | null)
```

DeviceOut ^ Collapse all object

```
name* > Expand all string
type* > Expand all string
ip_address* > Expand all string ipvanaddress
model > Expand all (string | null)
location > Expand all (string | null)
notes > Expand all (string | null)
is_active > Expand all (boolean | null)
id* integer
last_status* string
last_latency > Expand all (number | null)
last_cpu > Expand all (number | null)
last_memory > Expand all (number | null)
last_traffic > Expand all (number | null)
last_polled > Expand all (string | null)
last_error > Expand all (string | null)
failing_protocols > Expand all (string | null)
configured_credentials > Expand all array<string>
```

DeviceUpdate ^ Collapse all object

```
name > Expand all (string | null)
type > Expand all (string | null)
ip_address > Expand all (string | null)
model > Expand all (string | null)
location > Expand all (string | null)
notes > Expand all (string | null)
is_active > Expand all (boolean | null)
```

ForgotPasswordRequest ^ Collapse all object

```
email* string email
```

HTTPValidationError ^ Collapse all object

```
detail > Expand all array<object>
```

```
MetricOut ^ Collapse all object
  id* integer
  device_id* integer
  timestamp* string date-time
  cpu_usage > Expand all (number | null)
  memory_usage > Expand all (number | null)
  traffic > Expand all (number | null)
  latency > Expand all (number | null)
```

```
ResetPasswordRequest ^ Collapse all object
  token* string
  new_password* string
```

```
SystemSettingsResponse ^ Collapse all object
  polling_interval* integer
  cpu_warning_threshold* number
  cpu_critical_threshold* number
  memory_warning_threshold* number
  memory_critical_threshold* number
  latency_warning_threshold* number
  id* integer
```

```
SystemSettingsUpdate ^ Collapse all object
  polling_interval > Expand all (integer | null)
  cpu_warning_threshold > Expand all (number | null)
  cpu_critical_threshold > Expand all (number | null)
  memory_warning_threshold > Expand all (number | null)
  memory_critical_threshold > Expand all (number | null)
  latency_warning_threshold > Expand all (number | null)
```

```
Token ^ Collapse all object
  access_token* string
  refresh_token* string
  token_type > Expand all string
```

```
TrendOut ^ Collapse all object
  timestamp* string date-time
  avg_cpu* number
  avg_memory* number
  avg_latency* number
  avg_traffic* number
```

```
UserCreate ^ Collapse all object
  email* string email
  password* string
  full_name > Expand all (string | null)
  role* > Expand all string
```

```
UserLogin ^ Collapse all object
  email* string email
  password* string
```

```
UserOut ^ Collapse all object
  id* integer
  email* string email
  full_name* > Expand all (string | null)
  role* string
  is_active* boolean
  receive_email_notifications* boolean
```

```
UserUpdate ^ Collapse all object
  full_name > Expand all (string | null)
  email > Expand all (string | null)
  role > Expand all (string | null)
  is_active > Expand all (boolean | null)
  receive_email_notifications > Expand all (boolean | null)
```

```
ValidationError ^ Collapse all object
  loc* > Expand all array<(string | integer)>
  msg* string
  type* string
```

Υπεύθυνη Δήλωση Συγγραφέα:

Δηλώνω ρητά ότι, σύμφωνα με το άρθρο 8 του Ν.1599/1986, η παρούσα εργασία αποτελεί αποκλειστικά προϊόν προσωπικής μου εργασίας, δεν προσβάλλει κάθε μορφής δικαιώματα διανοητικής ιδιοκτησίας, προσωπικότητας και προσωπικών δεδομένων τρίτων, δεν περιέχει έργα/εισφορές τρίτων για τα οποία απαιτείται άδεια των δημιουργών/δικαιούχων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον και πληρούν τους κανόνες της επιστημονικής παράθεσης.