

Σχολή Θετικών Επιστημών & Τεχνολογίας (ΣΘΕΤ)

Μεταπτυχιακή Εξειδίκευση στα Πληροφοριακά
Συστήματα(MSc)

Πτυχιακή / Διπλωματική Εργασία

**Σχεδιασμός, υλοποίηση και αξιολόγηση ενός ηλεκτρονικού
καταστήματος με τεχνολογίες αιχμής**

Θεόδωρος Αγγελόπουλος

Επιβλέπων καθηγητής: Δημήτριος Αμανατίδης

Πάτρα, Ιούνιος 2026

Η παρούσα εργασία αποτελεί πνευματική ιδιοκτησία του φοιτητή Αγγελόπουλου Θεόδωρου 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 Ηλεκτρονικού Καταστήματος με
Τεχνολογίες Αιχμής

Θεόδωρος Αγγελόπουλος

Επιτροπή Επίβλεψης Πτυχιακής / Διπλωματικής Εργασίας

Επιβλέπων Καθηγητής:

Δημήτριος Αμανατίδης

Μέλος ΣΕΠ

Σχολή Θετικών Επιστημών και
Τεχνολογίας

Ελληνικό Ανοικτό Πανεπιστήμιο

Συν-Επιβλέπων Καθηγητής:

Δημήτριος Καλλές

Καθηγητής

Σχολή Θετικών Επιστημών και
Τεχνολογίας

Ελληνικό Ανοικτό Πανεπιστήμιο

Πάτρα, Ιούνιος 2026

«Ιδιαίτερες ευχαριστίες οφείλω στην κόρη μου Βασιλική, η οποία αποτέλεσε για μένα ανεξάντλητη πηγή δύναμης, υπομονής και έμπνευσης καθ' όλη τη διάρκεια της εκπόνησης της παρούσας διπλωματικής εργασίας. Η αγάπη και η παρουσία της μου υπενθύμιζαν καθημερινά την αξία της επιμονής, της προσπάθειας και της ολοκλήρωσης των στόχων μας. Σε εκείνη αφιερώνω αυτή την εργασία, με την ευχή να ακολουθεί πάντοτε τα όνειρά της με πίστη, θάρρος και αποφασιστικότητα.»

Περίληψη

Η ραγδαία ανάπτυξη του ηλεκτρονικού εμπορίου και η αυξανόμενη ανάγκη για αποδοτικές, επεκτάσιμες και φιλικές προς τον χρήστη εφαρμογές έχουν οδηγήσει στην υιοθέτηση νέων αρχιτεκτονικών σχεδιασμού πληροφοριακών συστημάτων. Τα παραδοσιακά συστήματα διαχείρισης περιεχομένου, στα οποία το επίπεδο παρουσίασης και το επίπεδο διαχείρισης δεδομένων είναι στενά συνδεδεμένα, παρουσιάζουν περιορισμούς ως προς την απόδοση, την ευελιξία και τη δυνατότητα προσαρμογής σε σύγχρονες τεχνολογικές απαιτήσεις.

Η παρούσα διπλωματική εργασία επικεντρώνεται στον σχεδιασμό, την υλοποίηση και την αξιολόγηση ενός σύγχρονου ηλεκτρονικού καταστήματος βασισμένου σε headless αρχιτεκτονική, με χρήση του Strapi ως συστήματος διαχείρισης περιεχομένου και του Next.js ως Progressive Web Application (PWA) στο επίπεδο παρουσίασης. Η προτεινόμενη αρχιτεκτονική διαχωρίζει πλήρως το backend από το frontend, επιτρέποντας την παροχή δεδομένων μέσω προγραμματιστικών διεπαφών (APIs) και την ανάπτυξη ενός ευέλικτου και υψηλής απόδοσης περιβάλλοντος χρήστη. Επιπλέον, η πλατφόρμα επεκτάθηκε πειραματικά με λειτουργία προτάσεων προϊόντων, A/B testing, δοκιμαστική διασύνδεση πληρωμών μέσω Stripe, evaluation toolkit και ενδεικτική serverless/MACH αρχιτεκτονική. Στο πλαίσιο της εργασίας πραγματοποιείται ανάλυση των λειτουργικών και μη λειτουργικών απαιτήσεων ενός συστήματος ηλεκτρονικού εμπορίου, ακολουθούμενη από τον αρχιτεκτονικό σχεδιασμό και την υλοποίηση του backend με το Strapi και του frontend με το Next.js, αξιοποιώντας χαρακτηριστικά Progressive Web Applications, όπως η offline λειτουργία και η βελτιστοποίηση απόδοσης. Η εφαρμογή αξιολογείται με βάση μετρικές απόδοσης, ασφάλειας και εμπειρίας χρήστη, ενώ τα αποτελέσματα συγκρίνονται με τις αντίστοιχες δυνατότητες παραδοσιακών προσεγγίσεων ηλεκτρονικού εμπορίου.

Τα αποτελέσματα της μελέτης καταδεικνύουν ότι ο συνδυασμός headless CMS και PWA μπορεί να προσφέρει σημαντικά πλεονεκτήματα ως προς την απόδοση, την επεκτασιμότητα και τη συνολική εμπειρία χρήστη, καθιστώντας την προτεινόμενη λύση ιδιαίτερα κατάλληλη για σύγχρονες εφαρμογές ηλεκτρονικού εμπορίου.

Λέξεις – Κλειδιά

- Headless CMS
- Strapi
- Ηλεκτρονικό Εμπόριο
- Progressive Web Applications (PWA)
- Next.js
- Αρχιτεκτονική Ιστού

Design and Development of an E-Commerce Application Using Cutting-Edge Technologies

Theodoros Angelopoulos

Abstract

The rapid growth of electronic commerce and the increasing demand for high-performance, scalable and user-friendly web applications have led to the adoption of new architectural approaches in the design of modern information systems. Traditional content management systems, in which the presentation layer and the data management layer are tightly coupled, often present limitations in terms of performance, flexibility and adaptability to contemporary technological requirements.

This thesis focuses on the design, implementation and evaluation of a modern e-commerce application based on a headless architecture, using Strapi as the content management system and Next.js as a Progressive Web Application (PWA) on the presentation layer. The proposed architecture fully separates the backend from the frontend, enabling data delivery through application programming interfaces (APIs) and allowing the development of a flexible, high-performance user interface. In addition, the platform was experimentally extended with a product recommendation mechanism, A/B testing, a test Stripe payment integration, an evaluation toolkit, and an indicative serverless/MACH architecture.

Within the scope of this work, the functional and non-functional requirements of an e-commerce system are analyzed, followed by the architectural design and the implementation of the backend with Strapi and the frontend with Next.js, leveraging key PWA features such as offline functionality and performance optimization. The developed application is evaluated in terms of performance, security and user experience, while the results are compared with those of traditional e-commerce approaches.

The findings of this study indicate that the combination of a headless CMS and a Progressive Web Application can provide significant advantages in performance, scalability and overall user experience, making the proposed solution particularly suitable for modern e-commerce applications.

Keywords

- Headless CMS
- Strapi
- E-commerce
- Progressive Web Applications (PWA)
- Next.js
- Web Architecture

Περιεχόμενα

Περίληψη.....	2
Abstract	4
Περιεχόμενα	6
Κατάλογος Εικόνων	7
Κατάλογος Πινάκων	8
Συνοτομογραφίες & Ακρωνύμια.....	9
1. Εισαγωγή.....	10
1.1. Αντικείμενο και σκοπός της εργασίας	10
1.2. Κίνητρο και σημασία της εργασίας.....	11
1.3. Στόχοι της εργασίας	12
1.4. Μεθοδολογία.....	13
1.5. Δομή της εργασίας	13
2. Θεωρητικό Υπόβαθρο Και Βιβλιογραφική Ανασκόπηση	15
2.1. Συστήματα Διαχείρισης Περιεχομένου (Content Management Systems)	15
2.2. Παραδοσιακά CMS στο ηλεκτρονικό εμπόριο	16
2.3. Συστήματα Διαχείρισης Περιεχομένου χωρίς Ενσωματωμένο Επίπεδο Παρουσίασης (Headless CMS)	17
2.4. Το Strapi ως Headless CMS.....	18
2.5. Προοδευτικές Διαδικτυακές Εφαρμογές (Progressive Web Applications – PWA)	19
2.6. Συνδυασμός Headless CMS και PWA στο ηλεκτρονικό εμπόριο	19
2.7. Βιβλιογραφική ανασκόπηση	19
2.7.1 Σύγχρονες τάσεις στο ηλεκτρονικό εμπόριο και στα headless CMS.....	20
2.7.2 Progressive Web Applications και εμπειρία χρήστη στο e-commerce.....	20
2.7.3 Τεχνητή νοημοσύνη, recommender systems και πειραματική βελτιστοποίηση UX.....	21
2.7.4 Ερευνητικό κενό και συνεισφορά της εργασίας	21
2.8. Συμπεράσματα κεφαλαίου	22
3. Ανάλυση Απαιτήσεων και Σχεδιασμός Συστήματος	23
3.1. Ανάλυση προβλήματος	23
3.2. Λειτουργικές απαιτήσεις.....	24
3.3. Μη λειτουργικές απαιτήσεις	25
3.4. Αντιστοίχιση απαιτήσεων με υλοποίηση και αξιολόγηση.....	27
3.5. Αρχιτεκτονικός σχεδιασμός του συστήματος	30
3.6. Σχεδιασμός δεδομένων (Data Model).....	31
3.7. Σχεδιασμός διεπαφών και ροή δεδομένων	32
3.8. Συμπεράσματα κεφαλαίου	33
4. Υλοποίηση Backend Συστηματος Με Strapi	34
4.1. Εισαγωγή.....	34
4.2. Εγκατάσταση και βασικές ρυθμίσεις Strapi.....	35
4.3. Σχεδίαση και Δημιουργία Content Types	35
4.4. Καταχώρηση Περιεχομένου μέσω Content Manager	38
4.5. Σχέσεις μεταξύ Content Types.....	40
4.6. Διαχείριση χρηστών και δικαιωμάτων πρόσβασης.....	41
4.7. Παροχή δεδομένων μέσω API	44
4.8. Ανάπτυξη και φιλοξενία του backend.....	44

4.9. Συμπεράσματα κεφαλαίου	44
5. Υλοποίηση Frontend Εφαρμογής Με Next.js Ως Progressive Web Application	45
5.1. Εισαγωγή	45
5.2. Επιλογή και ρύθμιση του Next.js	46
5.3. Επικοινωνία Frontend με το Strapi Backend	46
5.4. Υλοποίηση βασικών λειτουργιών ηλεκτρονικού καταστήματος	47
5.4.1 Η εφαρμογή υλοποιήθηκε με χρήση του App Router του Next.js	47
5.4.2 Ανάκτηση προϊόντων	47
5.4.3 Δυναμική σελίδα προϊόντος	48
5.4.4 Καλάθι αγορών	48
5.4.5 Checkout	49
5.4.6 Login και Register	50
5.4.7 Προβολή ιστορικού παραγγελιών	50
5.4.8 Προβολή κατάστασης παραγγελίας	51
5.5. Υλοποίηση χαρακτηριστικών Progressive Web Application	52
5.6. Βελτιστοποίηση απόδοσης και εμπειρίας χρήστη	52
5.7. Ασφάλεια στο επίπεδο του Frontend	52
5.8. Ανάπτυξη και φιλοξενία του Frontend	53
5.9. Συμπεράσματα κεφαλαίου	53
6. Πειραματικές Επεκτάσεις, Πληρωμές και Αξιολόγηση της Πλατφόρμας	55
6.1. Πειραματικός AI Recommender και A/B Testing	55
6.2. Μετρική αξιολόγηση απόδοσης και ασφάλειας	56
6.3. Συγκριτική αξιολόγηση με παραδοσιακό CMS WooCommerce	58
6.4. Ενδεικτική serverless και composable αρχιτεκτονική	59
6.5. Αντιστοίχιση με αρχές MACH	60
6.6. Υλοποίηση δοκιμαστικού συστήματος πληρωμών με Stripe	61
6.7. Οφέλη της προτεινόμενης αρχιτεκτονικής	62
6.8. Συμπεράσματα κεφαλαίου	64
7. Βιβλιογραφία	64
8. Παράρτημα Α: Βασικός Κώδικας Υλοποίησης	67
9. Παράρτημα Β: «Front-End (Next.js)»	71
10. Παράρτημα Γ – Κώδικας Πιστοποίησης Χρηστών	82
11. Παράρτημα Δ: Πειραματικές Επεκτάσεις και Evaluation Toolkit	84

Κατάλογος Εικόνων

Εικόνα 1. Ροή αρχιτεκτονικής συστήματος ηλεκτρονικού εμπορίου	13
Εικόνα 2. Αρχιτεκτονική Παραδοσιακού CMS και Headless CMS	16
Εικόνα 3. Αρχιτεκτονική Headless CMS	18
Εικόνα 4. Διάγραμμα Οντοτήτων-Συσχετίσεων του συστήματος	32
Εικόνα 5. Η οντότητα του προϊόντος (product)	36
Εικόνα 6. Η οντότητα κατηγορίας (category)	37
Εικόνα 7. Η οντότητα παραγγελίας (order)	38
Εικόνα 8. Καταχώρηση κατηγοριών προϊόντων στο Strapi Content Manager	39
Εικόνα 9. Καταχωρημένα προϊόντα στο Strapi Content Manager	39

Εικόνα 10. Καταχωρημένες παραγγελίες στο Strapi Content Manager	40
Εικόνα 11. Η σχέση μεταξύ χρήστη (User) και παραγγελίας (Order).....	41
Εικόνα 12. Ρόλοι χρηστών.....	42
Εικόνα 13. Stripe checkout.....	50
Εικόνα 14. Ιστορικό και κατάσταση παραγγελιών εκάστοτε χρήστη στο front end.....	51
Εικόνα 15. Μετρικές UX	56
Εικόνα 16. Μετρική αξιολόγηση απόδοσης και ασφάλειας	57

Κατάλογος Πινάκων

Πίνακας 1 Στοιχεία πειραματικού recommender και A/B testing	56
Πίνακας 2. Συνοπτικά αποτελέσματα HTTP σύγκρισης Next.js/Strapi και WooCommerce	58
Πίνακας 3. Ποιοτική σύγκριση της προτεινόμενης headless πλατφόρμας με WooCommerce	59
Πίνακας 4 Ενδεικτικά serverless services	60
Πίνακας 5 Αντιστοίχιση της πλατφόρμας με τις αρχές MACH	61

Συντομογραφίες & Ακρωνύμια

API	Application Programming Interface
CMS	Content Management System
CRUD	Create, Read, Update, Delete
DB	Database
ERD	Entity Relationship Diagram
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
JSON	JavaScript Object Notation
JWT	JSON Web Token
PWA	Progressive Web Application
REST	Representational State Transfer
SPA	Single Page Application
SSR	Server-Side Rendering
SSG	Static Site Generation
TTFB	Time To First Byte
UI	User Interface
UX	User Experience
AI	Artificial Intelligence
CTR	Click-Through Rate
MACH	Microservices, API-first, Cloud-native, Headless
TTFB	Time To First Byte
UX	User Experience
JWT	JSON Web Token

1. Εισαγωγή

Η ανάπτυξη σύγχρονων εφαρμογών ηλεκτρονικού εμπορίου απαιτεί αρχιτεκτονικές που να μπορούν να ανταποκριθούν σε αυξημένες απαιτήσεις απόδοσης, ασφάλειας, επεκτασιμότητας και εμπειρίας χρήστη. Οι παραδοσιακές μονολιθικές προσεγγίσεις, στις οποίες το επίπεδο παρουσίασης και το επίπεδο διαχείρισης δεδομένων είναι στενά συνδεδεμένα, συχνά περιορίζουν την ευελιξία και δυσχεραίνουν την ενσωμάτωση νέων τεχνολογιών και υπηρεσιών (Newman, 2021).

Η υιοθέτηση αρχιτεκτονικών διαχωρισμένου επιπέδου παρουσίασης (headless architectures) και αρχιτεκτονικών με προτεραιότητα στις προγραμματιστικές διεπαφές (API-first architectures) επιτρέπει τον σαφή διαχωρισμό του οπίσθιου επιπέδου (backend) από το επίπεδο παρουσίασης (frontend). Παράλληλα, παρέχει τη δυνατότητα ανεξάρτητης ανάπτυξης, συντήρησης και επέκτασης των επιμέρους τμημάτων του συστήματος. Στο πλαίσιο της παρούσας εργασίας, ο διαχωρισμός αυτός υλοποιείται με τη χρήση του Strapi για τη διαχείριση των δεδομένων και της επιχειρησιακής λογικής και του Next.js για την ανάπτυξη μιας προοδευτικής διαδικτυακής εφαρμογής (Progressive Web Application – PWA).

Η προτεινόμενη λύση αξιοποιείται για την υλοποίηση ενός λειτουργικού ηλεκτρονικού καταστήματος, το οποίο περιλαμβάνει διαχείριση προϊόντων και κατηγοριών, πιστοποίηση χρηστών, καλάθι αγορών, διαχείριση παραγγελιών, δοκιμαστική ενσωμάτωση πληρωμών και πειραματικές λειτουργίες εξατομίκευσης και αξιολόγησης. Παράλληλα, διερευνάται η δυνατότητα μελλοντικής μετάβασης σε αρχιτεκτονική χωρίς διαχείριση εξυπηρετητών (serverless architecture), σε συνθέσιμη αρχιτεκτονική (composable architecture) και σε αρχιτεκτονική που ακολουθεί τις αρχές MACH.

1.1. Αντικείμενο και σκοπός της εργασίας

Η ραγδαία εξέλιξη των τεχνολογιών διαδικτύου και η αυξανόμενη ζήτηση για αποδοτικές, επεκτάσιμες και ασφαλείς εφαρμογές ηλεκτρονικού εμπορίου έχουν οδηγήσει σε σημαντικές αλλαγές στον τρόπο σχεδιασμού και υλοποίησης σύγχρονων πληροφοριακών συστημάτων. Τα παραδοσιακά συστήματα διαχείρισης περιεχομένου (Content Management Systems – CMS), τα οποία συνδυάζουν σε μία ενιαία αρχιτεκτονική τόσο το

επίπεδο παρουσίασης (frontend) όσο και το επίπεδο διαχείρισης δεδομένων (backend), παρουσιάζουν περιορισμούς ως προς την απόδοση, την επεκτασιμότητα και την ευελιξία προσαρμογής σε σύγχρονες απαιτήσεις.

Στο πλαίσιο αυτό, η αρχιτεκτονική των headless CMS αποτελεί μία σύγχρονη προσέγγιση, η οποία διαχωρίζει πλήρως το σύστημα διαχείρισης περιεχομένου από το επίπεδο παρουσίασης, παρέχοντας τα δεδομένα μέσω προγραμματιστικών διεπαφών (APIs) (Strapi, 2025). Η προσέγγιση αυτή επιτρέπει την ανάπτυξη ανεξάρτητων επιπέδων παρουσίασης (frontend) εφαρμογών, οι οποίες μπορούν να αξιοποιούν σύγχρονες τεχνολογίες ιστού με στόχο τη βελτίωση της απόδοσης και της εμπειρίας χρήστη.

Σκοπός της παρούσας διπλωματικής εργασίας είναι ο σχεδιασμός, η υλοποίηση και η αξιολόγηση ενός ηλεκτρονικού καταστήματος βασισμένου σε αρχιτεκτονική διαχωρισμένου επιπέδου παρουσίασης (headless architecture), με χρήση του Strapi ως συστήματος διαχείρισης περιεχομένου και του Next.js για την ανάπτυξη του επιπέδου παρουσίασης ως προοδευτικής διαδικτυακής εφαρμογής (Progressive Web Application – PWA) (Google, 2024). Η εργασία στοχεύει στη διερεύνηση των πλεονεκτημάτων που προσφέρει ο συνδυασμός headless CMS και PWA σε σύγκριση με τις παραδοσιακές προσεγγίσεις υλοποίησης εφαρμογών ηλεκτρονικού εμπορίου (e-commerce).

1.2. Κίνητρο και σημασία της εργασίας

Η συνεχής αύξηση των ηλεκτρονικών αγορών και η ανάγκη για άμεση, αξιόπιστη και φιλική προς τον χρήστη εμπειρία καθιστούν απαραίτητη τη χρήση τεχνολογιών που μπορούν να ανταποκριθούν σε υψηλά φορτία, πολλαπλές συσκευές και απαιτήσεις χαμηλού χρόνου απόκρισης. Οι χρήστες αναμένουν πλέον εφαρμογές που να φορτώνουν γρήγορα, να λειτουργούν απρόσκοπτα σε κινητές συσκευές και να προσφέρουν εμπειρία αντίστοιχη με εγγενείς εφαρμογές (native apps).

Τα Progressive Web Applications συνδυάζουν τα πλεονεκτήματα των web εφαρμογών με χαρακτηριστικά εγγενών εφαρμογών, όπως η δυνατότητα offline λειτουργίας, η εγκατάσταση στην αρχική οθόνη της συσκευής και η βελτιστοποιημένη απόδοση. Σε συνδυασμό με ένα headless CMS όπως το Strapi, το οποίο παρέχει ευέλικτη και ασφαλή διαχείριση δεδομένων μέσω API, προκύπτει μία σύγχρονη και επεκτάσιμη αρχιτεκτονική κατάλληλη για εφαρμογές ηλεκτρονικού εμπορίου.

Το κίνητρο για την εκπόνηση της παρούσας εργασίας προκύπτει από την ανάγκη πρακτικής μελέτης και αξιολόγησης της συγκεκριμένης αρχιτεκτονικής προσέγγισης σε ένα ολοκληρωμένο σενάριο ηλεκτρονικού εμπορίου. Ιδιαίτερο ενδιαφέρον παρουσιάζει η συνδυαστική αξιοποίηση ενός headless CMS και μιας Progressive Web Application, καθώς επιτρέπει να εξεταστούν στην πράξη η απόδοση, η επεκτασιμότητα, η ασφάλεια και η εμπειρία χρήστη της προτεινόμενης λύσης.

1.3. Στόχοι της εργασίας

Οι βασικοί στόχοι της παρούσας διπλωματικής εργασίας συνοψίζονται ως εξής:

- Ο σχεδιασμός ενός σύγχρονου ηλεκτρονικού καταστήματος με βάση τις αρχές της headless αρχιτεκτονικής.
- Η υλοποίηση backend συστήματος διαχείρισης περιεχομένου με χρήση του Strapi.
- Η ανάπτυξη frontend εφαρμογής με χρήση του Next.js, αξιοποιώντας χαρακτηριστικά Progressive Web Application.
- Η διασύνδεση frontend και backend μέσω ασφαλών προγραμματιστικών διεπαφών (REST ή GraphQL APIs).
- Η αξιολόγηση της απόδοσης, της ασφάλειας και της εμπειρίας χρήστη της υλοποιημένης εφαρμογής.
- Η σύγκριση της προτεινόμενης λύσης με παραδοσιακές προσεγγίσεις ηλεκτρονικού εμπορίου.
- Η πειραματική ενσωμάτωση μηχανισμού προτάσεων προϊόντων με χρήση συνεργατικού φιλτραρίσματος.
- Η αξιολόγηση διαφορετικών παραλλαγών παρουσίασης προτάσεων μέσω δοκιμή δύο παραλλαγών (**A/B testing**).
- Η δοκιμαστική διασύνδεση της διαδικασίας ολοκλήρωσης αγοράς (**checkout**) με σύστημα ηλεκτρονικών πληρωμών μέσω Stripe.
- Η διερεύνηση δυνατότητας μελλοντικής μετάβασης σε serverless και composable/MACH αρχιτεκτονική.

1.4. Μεθοδολογία

Για την επίτευξη των στόχων της εργασίας ακολουθείται συγκεκριμένη μεθοδολογία, η οποία περιλαμβάνει αρχικά τη μελέτη και ανάλυση της σχετικής βιβλιογραφίας στον τομέα των συστημάτων διαχείρισης περιεχομένου, των headless αρχιτεκτονικών και των Progressive Web Applications. Στη συνέχεια πραγματοποιείται ανάλυση απαιτήσεων για τον καθορισμό των λειτουργικών και μη λειτουργικών χαρακτηριστικών της εφαρμογής ηλεκτρονικού εμπορίου.

Ακολουθεί ο αρχιτεκτονικός σχεδιασμός του συστήματος και η υλοποίηση του οπίσθιου επιπέδου (backend) με το Strapi και του επιπέδου παρουσίασης (frontend) με το Next.js (Εικόνα 1. Ροή αρχιτεκτονικής συστήματος ηλεκτρονικού εμπορίου). Η εργασία ολοκληρώνεται με την αξιολόγηση της εφαρμογής μέσω μετρήσεων απόδοσης, ασφάλειας και χρηστικότητας, καθώς και με την εξαγωγή συμπερασμάτων και προτάσεων για μελλοντική επέκταση.



Εικόνα 1. Ροή αρχιτεκτονικής συστήματος ηλεκτρονικού εμπορίου

Σε επόμενο στάδιο ενσωματώθηκαν πειραματικές λειτουργίες προτάσεων προϊόντων και A/B testing, δοκιμαστική ροή ηλεκτρονικών πληρωμών και ανεξάρτητο σύνολο εργαλείων αξιολόγησης. Η αξιολόγηση περιλαμβάνει μετρήσεις απόδοσης, ελέγχους βασικών μηχανισμών ασφάλειας και συλλογή μετρικών αλληλεπίδρασης του χρήστη. Τέλος, εξετάστηκε η δυνατότητα απομόνωσης επιμέρους λειτουργιών σε ανεξάρτητες υπηρεσίες, σύμφωνα με τις αρχές μιας συνθέσιμης και MACH προσανατολισμένης αρχιτεκτονικής.

1.5. Δομή της εργασίας

Η παρούσα διπλωματική εργασία οργανώνεται σε επτά κεφάλαια. Στο πρώτο κεφάλαιο παρουσιάζονται το αντικείμενο, ο σκοπός, το κίνητρο, οι στόχοι και η μεθοδολογία της εργασίας. Στο δεύτερο κεφάλαιο αναλύεται το θεωρητικό υπόβαθρο και η σχετική

βιβλιογραφία που αφορά τα συστήματα διαχείρισης περιεχομένου, τις headless αρχιτεκτονικές, τα Progressive Web Applications και τον συνδυασμό τους στο πλαίσιο του ηλεκτρονικού εμπορίου. Το τρίτο κεφάλαιο επικεντρώνεται στην ανάλυση των λειτουργικών και μη λειτουργικών απαιτήσεων, καθώς και στον αρχιτεκτονικό σχεδιασμό και το μοντέλο δεδομένων του συστήματος.

Στο τέταρτο κεφάλαιο παρουσιάζεται η υλοποίηση του backend συστήματος με χρήση του Strapi. Ειδικότερα, περιγράφονται η εγκατάσταση και παραμετροποίηση του Strapi, η δημιουργία των Content Types, η καταχώρηση περιεχομένου μέσω του Content Manager, οι σχέσεις μεταξύ των οντοτήτων, η διαχείριση χρηστών και δικαιωμάτων πρόσβασης, καθώς και η παροχή δεδομένων μέσω API. Στο πέμπτο κεφάλαιο αναλύεται η υλοποίηση του frontend με χρήση του Next.js ως Progressive Web Application, με έμφαση στην επικοινωνία με το Strapi backend, την ανάκτηση προϊόντων, τις δυναμικές σελίδες προϊόντων, το καλάθι αγορών, το checkout, τη σύνδεση και εγγραφή χρηστών, καθώς και την προβολή ιστορικού και κατάστασης παραγγελιών.

Στο έκτο κεφάλαιο παρουσιάζονται οι πειραματικές επεκτάσεις και η αξιολόγηση της πλατφόρμας. Συγκεκριμένα, περιγράφονται ο πειραματικός μηχανισμός προτάσεων προϊόντων με A/B testing, η μετρική αξιολόγηση απόδοσης και ασφάλειας, η ενδεικτική serverless και composable αρχιτεκτονική, η αντιστοίχιση της υλοποίησης με τις αρχές MACH, καθώς και η δοκιμαστική διασύνδεση πληρωμών μέσω Stripe. Τέλος, στο έβδομο κεφάλαιο παρατίθεται η βιβλιογραφία, ενώ στα παραρτήματα παρουσιάζεται ο βασικός κώδικας υλοποίησης του backend, του frontend, της πιστοποίησης χρηστών και των πειραματικών επεκτάσεων.

2. Θεωρητικό Υπόβαθρο Και Βιβλιογραφική Ανασκόπηση

Στο παρόν κεφάλαιο παρουσιάζεται το θεωρητικό υπόβαθρο που υποστηρίζει τον σχεδιασμό και την υλοποίηση της εφαρμογής ηλεκτρονικού εμπορίου. Αρχικά εξετάζονται τα Συστήματα Διαχείρισης Περιεχομένου και οι βασικές αρχές λειτουργίας των παραδοσιακών CMS, με έμφαση στα πλεονεκτήματα και στους περιορισμούς που εμφανίζουν σε σύγχρονα περιβάλλοντα ηλεκτρονικού εμπορίου.

Στη συνέχεια αναλύεται η έννοια των headless CMS και ο διαχωρισμός του επιπέδου διαχείρισης δεδομένων από το επίπεδο παρουσίασης. Ιδιαίτερη αναφορά γίνεται στο Strapi ως headless και API-first σύστημα διαχείρισης περιεχομένου, καθώς και στις δυνατότητες που παρέχει για τη δημιουργία προσαρμοσμένων δομών δεδομένων, τη διαχείριση χρηστών και την παροχή περιεχομένου μέσω προγραμματιστικών διεπαφών.

Παράλληλα, παρουσιάζονται οι Progressive Web Applications και τα βασικά χαρακτηριστικά τους, όπως η βελτιωμένη απόδοση, η προσαρμογή σε κινητές συσκευές, η δυνατότητα προσωρινής αποθήκευσης δεδομένων και η προσέγγιση εμπειρίας εγγενούς εφαρμογής. Εξετάζεται επίσης ο συνδυασμός headless CMS και PWA ως μία σύγχρονη αρχιτεκτονική προσέγγιση για την ανάπτυξη ευέλικτων, επεκτάσιμων και υψηλής απόδοσης εφαρμογών ηλεκτρονικού εμπορίου.

Τέλος, μέσω της βιβλιογραφικής ανασκόπησης διερευνώνται οι σχετικές αρχιτεκτονικές τάσεις και οι τεχνολογικές προσεγγίσεις που έχουν προταθεί στη διεθνή βιβλιογραφία. Η ανάλυση αυτή δημιουργεί το θεωρητικό πλαίσιο πάνω στο οποίο βασίζονται η ανάλυση απαιτήσεων, ο αρχιτεκτονικός σχεδιασμός και η πρακτική υλοποίηση της πλατφόρμας που παρουσιάζονται στα επόμενα κεφάλαια.

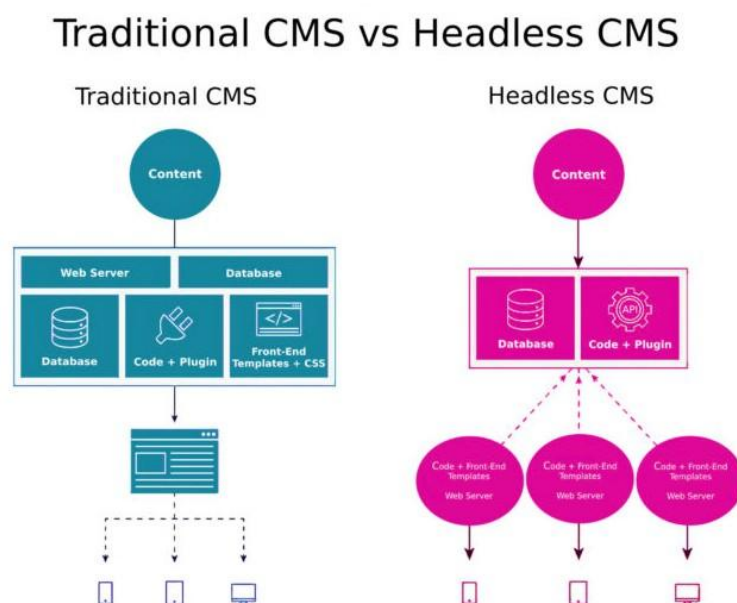
2.1. Συστήματα Διαχείρισης Περιεχομένου (Content Management Systems)

Τα Συστήματα Διαχείρισης Περιεχομένου (Content Management Systems – CMS) αποτελούν λογισμικά συστήματα τα οποία επιτρέπουν τη δημιουργία, διαχείριση και δημοσίευση ψηφιακού περιεχομένου χωρίς να απαιτείται εξειδικευμένη γνώση προγραμματισμού από τον τελικό χρήστη. Τα CMS χρησιμοποιούνται ευρέως για την ανάπτυξη ιστοσελίδων, διαδικτυακών εφαρμογών και ηλεκτρονικών καταστημάτων, καθώς

παρέχουν εργαλεία για τη διαχείριση κειμένου, πολυμέσων, χρηστών και ροών εργασίας. (Barker, 2016).

Στα παραδοσιακά CMS, το σύστημα διαχείρισης περιεχομένου και το επίπεδο παρουσίασης είναι άρρηκτα συνδεδεμένα σε μία ενιαία αρχιτεκτονική. Το CMS είναι υπεύθυνο τόσο για την αποθήκευση και επεξεργασία των δεδομένων όσο και για την τελική απόδοσή τους στον χρήστη μέσω προκαθορισμένων προτύπων παρουσίασης (templates). Αν και αυτή η προσέγγιση διευκολύνει την αρχική ανάπτυξη, παρουσιάζει περιορισμούς όσον αφορά την επεκτασιμότητα, την απόδοση και την προσαρμογή σε σύγχρονες τεχνολογικές απαιτήσεις (Barker, 2016).

Στην Εικόνα 2. Αρχιτεκτονική Παραδοσιακού CMS και Headless CMS παρουσιάζεται η βασική αρχιτεκτονική διαφορά μεταξύ ενός παραδοσιακού CMS και ενός headless CMS. Στην πρώτη περίπτωση, το σύστημα διαχείρισης περιεχομένου είναι υπεύθυνο τόσο για την αποθήκευση όσο και για την παρουσίαση των δεδομένων, ενώ στη δεύτερη περίπτωση το CMS λειτουργεί αποκλειστικά ως backend υπηρεσία, παρέχοντας το περιεχόμενο μέσω προγραμματιστικών διεπαφών (APIs).



Εικόνα 2. Αρχιτεκτονική Παραδοσιακού CMS και Headless CMS

2.2. Παραδοσιακά CMS στο ηλεκτρονικό εμπόριο

Στον τομέα του ηλεκτρονικού εμπορίου, τα παραδοσιακά CMS χρησιμοποιούνται συχνά σε συνδυασμό με εξειδικευμένα πρόσθετα (plugins) που παρέχουν λειτουργικότητες όπως διαχείριση προϊόντων, παραγγελιών και πληρωμών. Παρόλο που οι λύσεις αυτές είναι ώριμες και ευρέως διαδεδομένες, παρουσιάζουν μειονεκτήματα όπως αυξημένο φόρτο στο

frontend, δυσκολία στην υποστήριξη πολλαπλών καναλιών διάθεσης (web, mobile, IoT) και περιορισμένες δυνατότητες βελτιστοποίησης της απόδοσης (**Barker, 2016· Laudon & Traver, 2020**).

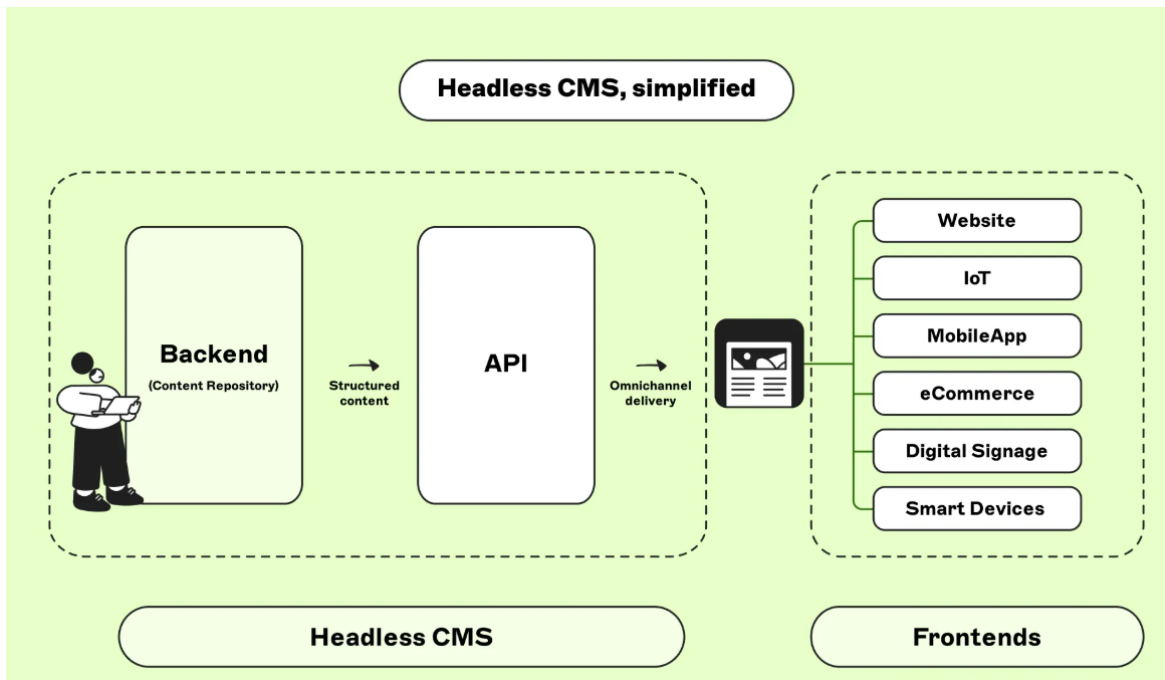
Επιπλέον, η στενή σύζευξη backend και frontend καθιστά δύσκολη την υιοθέτηση σύγχρονων πλαισίων ανάπτυξης του επιπέδου παρουσίασης (frontend frameworks) και τεχνικών, όπως η στατική παραγωγή σελίδων (Static Site Generation), η επαυξημένη προσωρινή αποθήκευση (caching) και η βελτιστοποίηση Core Web Vitals, τα οποία είναι κρίσιμα για την επιτυχία ενός σύγχρονου ηλεκτρονικού καταστήματος.

2.3. Συστήματα Διαχείρισης Περιεχομένου χωρίς Ενσωματωμένο Επίπεδο Παρουσίασης (Headless CMS)

Η έννοια του συστήματος διαχείρισης περιεχομένου χωρίς ενσωματωμένο επίπεδο παρουσίασης (headless CMS) εισάγει έναν διαφορετικό αρχιτεκτονικό σχεδιασμό, στον οποίο το CMS λειτουργεί αποκλειστικά ως υπηρεσία οπίσθιου επιπέδου (backend service). Το headless CMS δεν είναι υπεύθυνο για την παρουσίαση του περιεχομένου, αλλά παρέχει τα δεδομένα μέσω προγραμματιστικών διεπαφών, συνήθως REST ή GraphQL (**Barker, 2016· Strapi, 2025**).

Ο διαχωρισμός αυτός επιτρέπει την ανεξάρτητη ανάπτυξη του frontend με χρήση σύγχρονων τεχνολογιών, προσφέροντας αυξημένη ευελιξία, καλύτερη απόδοση και δυνατότητα διάθεσης του ίδιου περιεχομένου σε πολλαπλά κανάλια. Στο πλαίσιο του ηλεκτρονικού εμπορίου, η headless προσέγγιση επιτρέπει τη δημιουργία ταχύτερων και πιο φιλικών προς τον χρήστη εφαρμογών, οι οποίες μπορούν να προσαρμοστούν ευκολότερα στις ανάγκες της αγοράς και σε διαφορετικά κανάλια διάθεσης περιεχομένου (**Barker, 2016· Strapi, 2025**).

Στην Εικόνα 3. Αρχιτεκτονική Headless CMS παρουσιάζεται η βασική αρχιτεκτονική ενός headless CMS. Το σύστημα διαχείρισης περιεχομένου λειτουργεί αποκλειστικά ως backend υπηρεσία, παρέχοντας δεδομένα μέσω προγραμματιστικών διεπαφών (APIs), ενώ η παρουσίαση του περιεχομένου αναλαμβάνεται από ανεξάρτητες frontend εφαρμογές. Ο διαχωρισμός αυτός επιτρέπει αυξημένη ευελιξία, επεκτασιμότητα και πολυκαναλική διάθεση του περιεχομένου



Εικόνα 3. Αρχιτεκτονική Headless CMS
Πηγή: Storyblok, “Headless CMS Explained”.

2.4. Το Strapi ως Headless CMS

Το Strapi αποτελεί ένα ανοιχτού κώδικα headless CMS, το οποίο βασίζεται σε αρχιτεκτονική με προτεραιότητα στις προγραμματιστικές διεπαφές (API-first) και επιτρέπει τη γρήγορη ανάπτυξη backend συστημάτων για σύγχρονες web εφαρμογές. Παρέχει γραφικό περιβάλλον διαχείρισης για τη δημιουργία τύπων περιεχομένου (Content Types), τον ορισμό σχέσεων μεταξύ δεδομένων και τη διαχείριση χρηστών και δικαιωμάτων πρόσβασης.

Ένα από τα βασικά πλεονεκτήματα του Strapi είναι η ευελιξία του ως προς τη δομή των δεδομένων, καθώς επιτρέπει τη δημιουργία προσαρμοσμένων σχημάτων που καλύπτουν τις ανάγκες εφαρμογών ηλεκτρονικού εμπορίου, όπως προϊόντα, κατηγορίες, παραγγελίες και χρήστες. Παράλληλα, υποστηρίζει τόσο REST όσο και GraphQL APIs, διευκολύνοντας την επικοινωνία με σύγχρονες frontend εφαρμογές.

Η δυνατότητα αυτο-φιλοξενίας (self-hosting) του Strapi επιτρέπει πλήρη έλεγχο του backend, γεγονός ιδιαίτερα σημαντικό για εφαρμογές που διαχειρίζονται ευαίσθητα δεδομένα χρηστών και συναλλαγών. Επιπλέον, η αρχιτεκτονική του Strapi το καθιστά κατάλληλο για ενσωμάτωση σε αρχιτεκτονική χωρίς διαχείριση εξυπηρετητών (serverless) και συνθέσιμες υποδομές (composable architecture).

2.5. Προοδευτικές Διαδικτυακές Εφαρμογές (Progressive Web Applications – PWA)

Οι προοδευτικές διαδικτυακές εφαρμογές (Progressive Web Applications – PWA) αποτελούν προσέγγιση ανάπτυξης διαδικτυακών εφαρμογών που αξιοποιεί τεχνολογίες του ιστού, παρέχοντας παράλληλα δυνατότητες που συναντώνται σε εγγενείς εφαρμογές. Οι PWA μπορούν να αξιοποιούν μηχανισμούς εκτέλεσης στο παρασκήνιο (service workers) και αρχεία δήλωσης εφαρμογής ιστού (web app manifests), ώστε να υποστηρίζουν εγκατάσταση στη συσκευή, προσωρινή αποθήκευση πόρων και λειτουργία σε συνθήκες περιορισμένης ή μη διαθέσιμης σύνδεσης (Google, 2024· Biørn-Hansen et al., 2017).

Στον τομέα του ηλεκτρονικού εμπορίου, οι PWA μπορούν να συμβάλουν σημαντικά στη βελτίωση της εμπειρίας χρήστη, μειώνοντας τους χρόνους φόρτωσης και αυξάνοντας την αξιοπιστία της εφαρμογής ακόμη και σε περιβάλλοντα με περιορισμένη συνδεσιμότητα. Παράλληλα, η χρήση τεχνικών όπως η προσωρινή αποθήκευση δεδομένων και η βελτιστοποίηση των πόρων οδηγεί σε καλύτερες επιδόσεις σύμφωνα με τις μετρήσεις Core Web Vitals (Biørn-Hansen et al., 2017· Malavolta et al., 2017).

2.6. Συνδυασμός Headless CMS και PWA στο ηλεκτρονικό εμπόριο

Ο συνδυασμός ενός headless CMS, όπως το Strapi, με ένα frontend βασισμένο σε PWA τεχνολογίες δημιουργεί μία ισχυρή και ευέλικτη αρχιτεκτονική για εφαρμογές ηλεκτρονικού εμπορίου. Το backend αναλαμβάνει αποκλειστικά τη διαχείριση και παροχή δεδομένων, ενώ το frontend επικεντρώνεται στην απόδοση, την εμπειρία χρήστη και τη διαδραστικότητα.

Η αρχιτεκτονική αυτή επιτρέπει την ανεξάρτητη εξέλιξη των δύο επιπέδων, διευκολύνοντας τη συντήρηση και την επεκτασιμότητα του συστήματος. Επιπλέον, προσφέρει τη δυνατότητα ενσωμάτωσης με σύγχρονες τεχνολογίες υπολογιστικού νέφους (cloud) και αρχιτεκτονικές χωρίς διαχείριση εξυπηρετητών (serverless), ενισχύοντας την ανθεκτικότητα και την απόδοση της εφαρμογής (Barker, 2016· Biørn-Hansen et al., 2017· Strapi, 2025).

2.7. Βιβλιογραφική ανασκόπηση

Η βιβλιογραφική ανασκόπηση οργανώνεται γύρω από τέσσερις άξονες που συνδέονται άμεσα με την υλοποίηση της εργασίας: σύγχρονες headless και composable αρχιτεκτονικές, Progressive Web Applications, τεχνικές εξατομίκευσης με recommender systems και

πειραματική αξιολόγηση της εμπειρίας χρήστη. Η επιλογή αυτή επιτρέπει τη σύνδεση της θεωρίας με τις λειτουργίες που αναπτύχθηκαν στο πρωτότυπο ηλεκτρονικό κατάστημα.

2.7.1 Σύγχρονες τάσεις στο ηλεκτρονικό εμπόριο και στα headless CMS

Η πρόσφατη βιβλιογραφία δείχνει ότι το ηλεκτρονικό εμπόριο μετατοπίζεται από μονολιθικές λύσεις, όπου η διαχείριση περιεχομένου, η παρουσίαση και οι εμπορικές λειτουργίες συνυπάρχουν στο ίδιο σύστημα, προς πιο αποσυνδεδεμένες αρχιτεκτονικές. Οι headless CMS προσεγγίσεις διαχωρίζουν το επίπεδο διαχείρισης περιεχομένου από το επίπεδο παρουσίασης και επιτρέπουν την κατανάλωση δεδομένων μέσω APIs. Η επιλογή αυτή συνδέεται με καλύτερη ευελιξία, δυνατότητα πολυκαναλικής διάθεσης περιεχομένου και ανεξάρτητη εξέλιξη frontend και backend (Hassan & Eassa, 2024· MACH Alliance, n.d.· Strapi, n.d.).

Σε σχέση με ένα παραδοσιακό CMS, η headless προσέγγιση δεν είναι από μόνη της πανάκεια. Απαιτεί περισσότερη αρχική τεχνική οργάνωση, σαφή σχεδιασμό APIs και ευθύνη για τη σύνθεση επιμέρους υπηρεσιών. Ωστόσο, σε εφαρμογές όπου ζητούνται απόδοση, επεκτασιμότητα, ανεξάρτητη ανάπτυξη λειτουργιών και δυνατότητα πειραματισμού, η αποσύνδεση των επιπέδων αποτελεί τεκμηριωμένη αρχιτεκτονική επιλογή. Στο πλαίσιο της παρούσας εργασίας, η χρήση Strapi ως headless backend και Next.js ως frontend υλοποιεί ακριβώς αυτόν τον διαχωρισμό.

2.7.2 Progressive Web Applications και εμπειρία χρήστη στο e-commerce

Οι Progressive Web Applications αξιοποιούνται όλο και περισσότερο σε εμπορικές εφαρμογές, επειδή συνδυάζουν χαρακτηριστικά ιστοσελίδας και εφαρμογής, όπως εγκατάσταση στη συσκευή, caching, ταχύτερη επαναφόρτωση πόρων και καλύτερη εμπειρία σε κινητές συσκευές. Πρόσφατες μελέτες για PWA σε περιβάλλοντα ηλεκτρονικού εμπορίου συνδέουν τη συγκεκριμένη τεχνολογία με βελτίωση της εμπλοκής του χρήστη, του χρόνου παραμονής και της αντιλαμβανόμενης ευχρηστίας, όταν συνοδεύεται από κατάλληλο σχεδιασμό διεπαφής και μετρική αξιολόγηση (Hidayat et al., 2023· Putro et al., 2025· Vrhovac et al., 2023).

Η παρούσα υλοποίηση αξιοποιεί την PWA λογική συμπληρωματικά προς την headless αρχιτεκτονική. Έτσι, το frontend δεν λειτουργεί απλώς ως στατική βιτρίνα προϊόντων, αλλά ως ανεξάρτητο επίπεδο εμπειρίας χρήστη, στο οποίο μπορούν να εφαρμοστούν τεχνικές βελτιστοποίησης απόδοσης, προσωρινής αποθήκευσης και UX αξιολόγησης. Αυτή η

σύνδεση είναι κρίσιμη, διότι στο e-commerce η τεχνική απόδοση επηρεάζει άμεσα την εμπιστοσύνη, την παραμονή και την πιθανότητα ολοκλήρωσης αγοράς.

2.7.3 Τεχνητή νοημοσύνη, recommender systems και πειραματική βελτιστοποίηση UX

Τα συστήματα προτάσεων αποτελούν βασικό μηχανισμό εξατομίκευσης στο ηλεκτρονικό εμπόριο. Η πρόσφατη έρευνα αναδεικνύει τεχνικές collaborative filtering, hybrid approaches και AI-driven recommendation pipelines ως τρόπους βελτίωσης της σχετικότητας των προτεινόμενων προϊόντων και της εμπειρίας χρήστη (Aljunid et al., 2025· Necula & Păvăloaia, 2023· Valencia-Arias et al., 2024). Παράλληλα, η βιβλιογραφία επισημαίνει ότι η αξία τέτοιων τεχνικών πρέπει να εξετάζεται πειραματικά, με βάση δεδομένα αλληλεπίδρασης και μετρικές απόδοσης.

Στην παρούσα εργασία, ο πειραματικός AI recommender, το interaction tracking και το A/B testing δεν αντιμετωπίζονται ως πλήρως παραγωγικό σύστημα τεχνητής νοημοσύνης, αλλά ως απόδειξη αρχής για το πώς μια headless/composable πλατφόρμα μπορεί να επεκταθεί με ανεξάρτητες υπηρεσίες βελτιστοποίησης. Με αυτόν τον τρόπο, η εργασία συνδέει την πρακτική υλοποίηση με τη βιβλιογραφία περί εξατομίκευσης, UX αξιολόγησης και συνεχούς βελτίωσης ψηφιακών εμπορικών εφαρμογών (Zimmermann & Auinger, 2023).

2.7.4 Ερευνητικό κενό και συνεισφορά της εργασίας

Από τη βιβλιογραφία προκύπτει ότι υπάρχουν αρκετές μελέτες για CMS, PWA, recommender systems και UX αξιολόγηση, όμως συχνά εξετάζονται ως ξεχωριστά τεχνικά αντικείμενα. Το κενό που επιχειρεί να καλύψει η παρούσα εργασία βρίσκεται στη συνδυαστική αξιολόγηση αυτών των τεχνολογιών μέσα από ένα ενιαίο, λειτουργικό πρωτότυπο ηλεκτρονικού καταστήματος. Η εργασία δεν περιορίζεται στη θεωρητική σύγκριση τεχνολογιών, αλλά συνδέει την αρχιτεκτονική επιλογή με συγκεκριμένα υλοποίηση, λειτουργικές απαιτήσεις, μη λειτουργικές απαιτήσεις και μετρήσεις αξιολόγησης.

Η συνεισφορά της εργασίας εντοπίζεται στη σχεδίαση και υλοποίηση μιας πλατφόρμας που συνδυάζει Strapi, Next.js, PWA χαρακτηριστικά, δοκιμαστική πληρωμή μέσω Stripe, AI-based recommendations, A/B testing, serverless-style services και evaluation toolkit. Με τον τρόπο αυτό, η εργασία προσφέρει ένα πρακτικό παράδειγμα για το πώς μπορεί να

αξιολογηθεί μια σύγχρονη headless/composable αρχιτεκτονική έναντι πιο παραδοσιακών CMS προσεγγίσεων ως προς επεκτασιμότητα, απόδοση, ασφάλεια και εμπειρία χρήστη.

2.8. Συμπεράσματα κεφαλαίου

Στο παρόν κεφάλαιο παρουσιάστηκε το θεωρητικό υπόβαθρο που σχετίζεται με τα συστήματα διαχείρισης περιεχομένου, τις headless αρχιτεκτονικές και τις Progressive Web Applications. Ιδιαίτερη έμφαση δόθηκε στο Strapi ως headless CMS και στη σημασία του συνδυασμού του με PWA τεχνολογίες για την ανάπτυξη σύγχρονων εφαρμογών ηλεκτρονικού εμπορίου. Τα στοιχεία αυτά αποτελούν τη βάση για την ανάλυση απαιτήσεων και τον σχεδιασμό του συστήματος που παρουσιάζεται στο επόμενο κεφάλαιο.

3. Ανάλυση Απαιτήσεων και Σχεδιασμός Συστήματος

Στο παρόν κεφάλαιο παρουσιάζεται η ανάλυση των απαιτήσεων και ο σχεδιασμός της προτεινόμενης πλατφόρμας ηλεκτρονικού εμπορίου. Στόχος είναι να προσδιοριστούν με σαφήνεια οι βασικές λειτουργίες που πρέπει να υποστηρίζει το σύστημα, καθώς και τα ποιοτικά χαρακτηριστικά που είναι απαραίτητα για την αξιόπιστη, ασφαλή και αποδοτική λειτουργία του. Η συστηματική καταγραφή και τεκμηρίωση των απαιτήσεων αποτελεί βασικό μέρος της μηχανικής απαιτήσεων και υποστηρίζει τόσο τον σχεδιασμό όσο και την επαλήθευση του τελικού συστήματος (ISO/IEC/IEEE, 2018).

Αρχικά εξετάζεται το πρόβλημα που καλείται να αντιμετωπίσει η εφαρμογή και αποτυπώνονται οι ανάγκες των τελικών χρηστών και των διαχειριστών. Στη συνέχεια αναλύονται οι λειτουργικές απαιτήσεις, οι οποίες περιλαμβάνουν τη διαχείριση προϊόντων και κατηγοριών, την εγγραφή και πιστοποίηση χρηστών, τη λειτουργία καλαθιού αγορών, τη δημιουργία παραγγελιών και την παρακολούθηση της κατάστασής τους. Παράλληλα, προσδιορίζονται οι μη λειτουργικές απαιτήσεις που αφορούν την απόδοση, την ασφάλεια, την επεκτασιμότητα, τη διαθεσιμότητα και την εμπειρία χρήστη.

Ιδιαίτερη έμφαση δίνεται στον αρχιτεκτονικό σχεδιασμό του συστήματος, ο οποίος βασίζεται στον διαχωρισμό του backend από το frontend. Το Strapi χρησιμοποιείται για τη διαχείριση των δεδομένων, των χρηστών, των δικαιωμάτων πρόσβασης και των προγραμματιστικών διεπαφών, ενώ το Next.js αναλαμβάνει την παρουσίαση και την αλληλεπίδραση με τον χρήστη. Η επικοινωνία μεταξύ των δύο επιπέδων πραγματοποιείται μέσω APIs, σύμφωνα με τις αρχές της headless και API-first αρχιτεκτονικής.

Στο κεφάλαιο παρουσιάζεται επίσης το μοντέλο δεδομένων της εφαρμογής, με έμφαση στις βασικές οντότητες Product, Category, Order και User και στις μεταξύ τους σχέσεις. Τέλος, αναλύεται η ροή δεδομένων μεταξύ των επιμέρους τμημάτων της πλατφόρμας και τεκμηριώνεται ο τρόπος με τον οποίο ο σχεδιασμός αυτός υποστηρίζει τη συντηρησιμότητα, την ασφάλεια και τη μελλοντική επέκταση του συστήματος.

3.1. Ανάλυση προβλήματος

Η ανάπτυξη ενός σύγχρονου ηλεκτρονικού καταστήματος απαιτεί την υιοθέτηση αρχιτεκτονικών και τεχνολογιών που μπορούν να ανταποκριθούν στις αυξημένες απαιτήσεις απόδοσης, επεκτασιμότητας και εμπειρίας χρήστη. Οι μονολιθικές υλοποιήσεις μπορούν να παρουσιάσουν περιορισμούς ως προς την ανεξάρτητη εξέλιξη των επιμέρους

επιπέδων, ιδιαίτερα όταν αυξάνονται οι λειτουργικές απαιτήσεις και οι εξωτερικές διασυνδέσεις.

Το πρόβλημα που καλείται να αντιμετωπίσει η παρούσα εργασία αφορά τον σχεδιασμό και την υλοποίηση ενός ηλεκτρονικού καταστήματος το οποίο να είναι σε θέση να εξυπηρετεί αποτελεσματικά τους χρήστες, ανεξαρτήτως συσκευής, παρέχοντας υψηλή απόδοση και αξιόπιστη λειτουργία. Παράλληλα, το σύστημα θα πρέπει να προσφέρει ευελιξία στη διαχείριση περιεχομένου και δυνατότητα μελλοντικής επέκτασης χωρίς σημαντικές τροποποιήσεις στην αρχιτεκτονική του.

3.2. Λειτουργικές απαιτήσεις

Οι λειτουργικές απαιτήσεις περιγράφουν τις υπηρεσίες και τις ενέργειες που πρέπει να υποστηρίζει το ηλεκτρονικό κατάστημα. Οι απαιτήσεις προσδιορίστηκαν με βάση τους βασικούς ρόλους του συστήματος, δηλαδή τον επισκέπτη, τον εγγεγραμμένο χρήστη και τον διαχειριστή. Η καταγραφή τους με μοναδικούς κωδικούς επιτρέπει την αντιστοίχισή τους με τις λειτουργίες που υλοποιούνται και με τους αντίστοιχους ελέγχους αποδοχής.

Κωδικός	Λειτουργική απαίτηση	Ρόλος
FR-01	Το σύστημα πρέπει να επιτρέπει στον διαχειριστή τη δημιουργία, τροποποίηση, δημοσίευση και διαγραφή προϊόντων.	Διαχειριστής
FR-02	Το σύστημα πρέπει να υποστηρίζει τη δημιουργία και διαχείριση κατηγοριών προϊόντων.	Διαχειριστής
FR-03	Ο επισκέπτης πρέπει να μπορεί να περιηγείται στον κατάλογο και να προβάλλει τις αναλυτικές πληροφορίες κάθε προϊόντος.	Επισκέπτης
FR-04	Το σύστημα πρέπει να υποστηρίζει εγγραφή και σύνδεση χρηστών μέσω μηχανισμού αυθεντικοποίησης (authentication).	Επισκέπτης/Χρήστης

FR-05	Ο χρήστης πρέπει να μπορεί να προσθέτει και να αφαιρεί προϊόντα από το καλάθι αγορών και να βλέπει το συνολικό κόστος.	Χρήστης
FR-06	Ο συνδεδεμένος χρήστης πρέπει να μπορεί να ολοκληρώνει την αγορά (checkout) και να δημιουργεί παραγγελία.	Χρήστης
FR-07	Το σύστημα πρέπει να υποστηρίζει δοκιμαστική ηλεκτρονική πληρωμή μέσω Stripe και να ενημερώνει την κατάσταση της παραγγελίας μετά την επιβεβαίωσή της.	Χρήστης
FR-08	Ο χρήστης πρέπει να μπορεί να προβάλλει το ιστορικό και την κατάσταση των δικών του παραγγελιών.	Χρήστης
FR-09	Το σύστημα πρέπει να περιορίζει την πρόσβαση κάθε χρήστη αποκλειστικά στις δικές του παραγγελίες.	Χρήστης
FR-10	Το σύστημα πρέπει να παράγει πειραματικές προτάσεις προϊόντων με βάση τις καταγεγραμμένες αλληλεπιδράσεις ή εναλλακτικούς μηχανισμούς προτάσεων.	Χρήστης
FR-11	Το σύστημα πρέπει να καταγράφει συμβάντα αλληλεπίδρασης, όπως προβολή προϊόντος, προσθήκη στο καλάθι, αγορά και επιλογή πρότασης.	Σύστημα

3.3. Μη λειτουργικές απαιτήσεις

Οι μη λειτουργικές απαιτήσεις περιγράφουν τα ποιοτικά χαρακτηριστικά και τους περιορισμούς λειτουργίας της πλατφόρμας. Για τον προσδιορισμό τους λαμβάνονται υπόψη χαρακτηριστικά ποιότητας όπως η αποδοτικότητα απόδοσης, η ασφάλεια, η αξιοπιστία, η συντηρησιμότητα και η δυνατότητα επέκτασης (ISO/IEC, 2023).

Η ασφάλεια αποτελεί επίσης κρίσιμη απαίτηση, καθώς το σύστημα διαχειρίζεται δεδομένα χρηστών και παραγγελιών. Η επικοινωνία μεταξύ frontend και backend θα πρέπει να πραγματοποιείται μέσω ασφαλών πρωτοκόλλων και μηχανισμών αυθεντικοποίησης. Επιπλέον, το σύστημα θα πρέπει να είναι επεκτάσιμο, επιτρέποντας την προσθήκη νέων λειτουργιών χωρίς σημαντικές αλλαγές στον υπάρχοντα κώδικα.

Κωδικός	Μη λειτουργική απαίτηση	Κριτήριο αξιολόγησης
NFR-01	Η εφαρμογή πρέπει να παρέχει αποδεκτούς χρόνους απόκρισης για τις βασικές σελίδες και τα τελικά σημεία API.	Μέτρηση TTFB και καθυστέρησης API (latency).
NFR-02	Το επίπεδο παρουσίασης πρέπει να βελτιστοποιείται με βάση τις βασικές μετρικές εμπειρίας ιστού (Core Web Vitals).	Έλεγχος LCP, INP και CLS μέσω κατάλληλου εργαλείου.
NFR-03	Η επικοινωνία σε περιβάλλον παραγωγής πρέπει να πραγματοποιείται μέσω HTTPS.	Επιβεβαίωση χρήσης TLS/HTTPS.
NFR-04	Τα προστατευμένα τελικά σημεία API πρέπει να απορρίπτουν αιτήματα χωρίς έγκυρο διακριτικό JWT.	Έλεγχος πρόσβασης χωρίς ή με μη έγκυρο JWT.
NFR-05	Οι χρήστες πρέπει να έχουν πρόσβαση μόνο στους πόρους για τους οποίους έχουν εξουσιοδοτηθεί.	Έλεγχος ρόλων και ιδιοκτησίας παραγγελίας.
NFR-06	Τα μυστικά κλειδιά και οι ευαίσθητες μεταβλητές πρέπει να αποθηκεύονται μόνο στο backend μέσω μεταβλητών περιβάλλοντος.	Έλεγχος ότι δεν εκτίθενται στον κώδικα του client.
NFR-07	Η αρχιτεκτονική πρέπει να επιτρέπει την ανεξάρτητη συντήρηση και εξέλιξη του backend και του frontend.	Διακριτά έργα, ρυθμίσεις και κύκλοι ανάπτυξης.

NFR-08	Η διεπαφή πρέπει να προσαρμόζεται σε διαφορετικά μεγέθη οθόνης και κινητές συσκευές.	Έλεγχος responsive σχεδίασης.
NFR-09	Το σύστημα πρέπει να επιτρέπει την προσθήκη νέων υπηρεσιών χωρίς ριζική αλλαγή του βασικού πυρήνα.	Ανεξάρτητη ενσωμάτωση recommender, analytics και πληρωμών.

3.4. Αντιστοίχιση απαιτήσεων με υλοποίηση και αξιολόγηση

Για να τεκμηριωθεί ότι οι απαιτήσεις της πλατφόρμας δεν παραμένουν σε θεωρητικό επίπεδο, οι λειτουργικές απαιτήσεις αντιστοιχίζονται με συγκεκριμένα σημεία υλοποίησης και οι μη λειτουργικές απαιτήσεις με μεθόδους αξιολόγησης ή μετρικές. Η αντιστοίχιση αυτή λειτουργεί ως πίνακας ιχνηλασιμότητας μεταξύ ανάλυσης απαιτήσεων, υλοποίησης και αξιολόγησης.

Αντιστοίχιση λειτουργικών απαιτήσεων με σημεία υλοποίησης

Απαίτηση	Σημείο υλοποίησης	Τεκμήριο στην εργασία
FR-01	Strapi Content Manager και Product content type για δημιουργία, ενημέρωση, δημοσίευση και διαγραφή προϊόντων.	Κεφ. 4: Σχεδίαση Content Types και Καταχώρηση Περιεχομένου.
FR-02	Category content type και συσχέτιση προϊόντων με κατηγορίες.	Κεφ. 4: Σχέσεις μεταξύ Content Types.
FR-03	Σελίδα καταλόγου προϊόντων και δυναμική σελίδα προϊόντος στο Next.js frontend.	Κεφ. 5: Ανάκτηση προϊόντων και Δυναμική σελίδα προϊόντος.
FR-04	Ροές register/login με JWT authentication μέσω Strapi Users & Permissions.	Κεφ. 4: Διαχείριση χρηστών και Κεφ. 5: Login και Register.

FR-05	Client-side cart store, προσθήκη/αφαίρεση προϊόντων και υπολογισμός συνολικού κόστους.	Κεφ. 5: Καλάθι αγορών.
FR-06	Checkout flow και δημιουργία παραγγελίας στο Strapi backend.	Κεφ. 5: Checkout και Κεφ. 6: Stripe payment flow.
FR-07	Stripe PaymentIntents σε δοκιμαστικό περιβάλλον και ενημέρωση κατάστασης παραγγελίας μετά την επιβεβαίωση.	Κεφ. 6: Υλοποίηση δοκιμαστικού συστήματος πληρωμών με Stripe.
FR-08	Σελίδες ιστορικού και κατάστασης παραγγελιών ανά χρήστη.	Κεφ. 5: Προβολή ιστορικού παραγγελιών και κατάσταση παραγγελίας.
FR-09	Έλεγχος JWT, ownership παραγγελιών και περιορισμός πρόσβασης στα API endpoints.	Κεφ. 4: Ρόλοι και δικαιώματα πρόσβασης.
FR-10	Πειραματικό recommendation endpoint και frontend panel με προτεινόμενα προϊόντα.	Κεφ. 6: AI Recommender και A/B Testing.
FR-11	Interaction tracking για product view, add to cart, purchase και recommendation events.	Κεφ. 6: Interaction tracking και μετρικές UX.

Αντιστοίχιση μη λειτουργικών απαιτήσεων με μετρικές αξιολόγησης

Απαίτηση	Τρόπος αξιολόγησης	Σχετικό τεκμήριο/μέτρηση
NFR-01	HTTP latency και TTFB μέσω evaluation toolkit.	Κεφ. 6: Μετρική αξιολόγηση και σύγκριση με WooCommerce.
NFR-02	Lighthouse/Core Web Vitals όπου είναι διαθέσιμα και αξιολόγηση frontend performance.	Κεφ. 6: Μετρική αξιολόγηση απόδοσης.
NFR-03	Έλεγχος δημόσιας πρόσβασης μέσω HTTPS στα παραγωγικά endpoints.	Online εγκατάσταση shop.tagg.gr και woo.tagg.gr.
NFR-04	Δοκιμή προστατευμένων API endpoints χωρίς ή με μη έγκυρο JWT.	Κεφ. 4: Users & Permissions και API permissions.
NFR-05	Έλεγχος ownership σε παραγγελίες και προβολή μόνο των παραγγελιών του χρήστη.	Κεφ. 5: Ιστορικό και κατάσταση παραγγελιών.
NFR-06	Χρήση backend env variables για Stripe secret key και μη έκθεση μυστικών στον client.	Κεφ. 6: Stripe PaymentIntents και μεταβλητές περιβάλλοντος.
NFR-07	Διακριτά έργα backend/frontend και ανεξάρτητοι κύκλοι ανάπτυξης.	Κεφ. 3 και Κεφ. 6: headless/composable αρχιτεκτονική.
NFR-08	Έλεγχος responsive διεπαφής σε σελίδες	Κεφ. 5: UI/UX και PWA χαρακτηριστικά.

	προϊόντων, cart, checkout και evaluation.	
NFR-09	Ενσωμάτωση νέων υπηρεσιών recommender, analytics, checkout και serverless-style endpoints.	Κεφ. 6: AI Recommender, A/B Testing και serverless services.

Με τον τρόπο αυτό, οι απαιτήσεις συνδέονται άμεσα με τα κεφάλαια υλοποίησης και αξιολόγησης. Ιδιαίτερα για τις μη λειτουργικές απαιτήσεις, η ύπαρξη του evaluation toolkit και της συγκριτικής μέτρησης με WooCommerce επιτρέπει την παρουσίαση τεκμηριωμένων αποτελεσμάτων αντί για απλή θεωρητική περιγραφή.

3.5. Αρχιτεκτονικός σχεδιασμός του συστήματος

Ο αρχιτεκτονικός σχεδιασμός της πλατφόρμας βασίζεται στον διαχωρισμό του οπίσθιου επιπέδου (backend) από το επίπεδο παρουσίασης (frontend). Το Strapi λειτουργεί ως κεντρικό σύστημα διαχείρισης δεδομένων, χρηστών, δικαιωμάτων πρόσβασης και επιχειρησιακής λογικής, ενώ το Next.js αναλαμβάνει την παρουσίαση του περιεχομένου και την αλληλεπίδραση με τον χρήστη.

Η επικοινωνία μεταξύ των δύο επιπέδων πραγματοποιείται μέσω προγραμματιστικής διεπαφής REST (REST API), με ανταλλαγή δεδομένων σε μορφή JSON. Η επιλογή αυτή επιτρέπει την ανεξάρτητη ανάπτυξη, συντήρηση και φιλοξενία των δύο επιπέδων και ακολουθεί τη βασική λογική της αρχιτεκτονικής χωρίς ενσωματωμένο επίπεδο παρουσίασης (headless architecture) (Fielding, 2000).

Ο διαχωρισμός των επιπέδων διευκολύνει την προσθήκη νέων λειτουργιών, όπως μηχανισμών προτάσεων, υπηρεσιών πληρωμών, αναλυτικών δεδομένων χρήσης (analytics) και υπηρεσιών χωρίς διαχείριση εξυπηρετητών (serverless services), χωρίς να απαιτείται ανασχεδιασμός ολόκληρης της εφαρμογής.

3.6. Σχεδιασμός δεδομένων (Data Model)

Ο σχεδιασμός των δεδομένων αποτελεί βασικό στοιχείο της αρχιτεκτονικής του συστήματος. Οι κύριες οντότητες που εντοπίζονται στο σύστημα είναι τα προϊόντα, οι κατηγορίες, οι χρήστες και οι παραγγελίες. Κάθε προϊόν συνδέεται με μία ή περισσότερες κατηγορίες, ενώ οι παραγγελίες περιλαμβάνουν ένα σύνολο προϊόντων και σχετίζονται με συγκεκριμένο χρήστη.

Η Εικόνα 4. Διάγραμμα Οντοτήτων-Συσχετίσεων του συστήματος απεικονίζει το Διάγραμμα Οντοτήτων-Συσχετίσεων (Entity Relationship Diagram – ERD) του συστήματος ηλεκτρονικού καταστήματος. Το διάγραμμα παρουσιάζει τις βασικές οντότητες του συστήματος, τα κύρια χαρακτηριστικά τους, καθώς και τις μεταξύ τους σχέσεις, όπως αυτές προκύπτουν από την ανάλυση απαιτήσεων.

Οι βασικές οντότητες του συστήματος είναι οι Προϊόν (**Product**), Κατηγορία (**Category**), Παραγγελία (**Order**) και Χρήστης (**User**). Η οντότητα Product περιλαμβάνει πληροφορίες που αφορούν τα προϊόντα του καταστήματος, όπως το αναγνωριστικό τους, το όνομα, την τιμή, την περιγραφή και την εικόνα. Η οντότητα Category χρησιμοποιείται για την ομαδοποίηση των προϊόντων και επιτρέπει τη δημιουργία δομημένου καταλόγου.

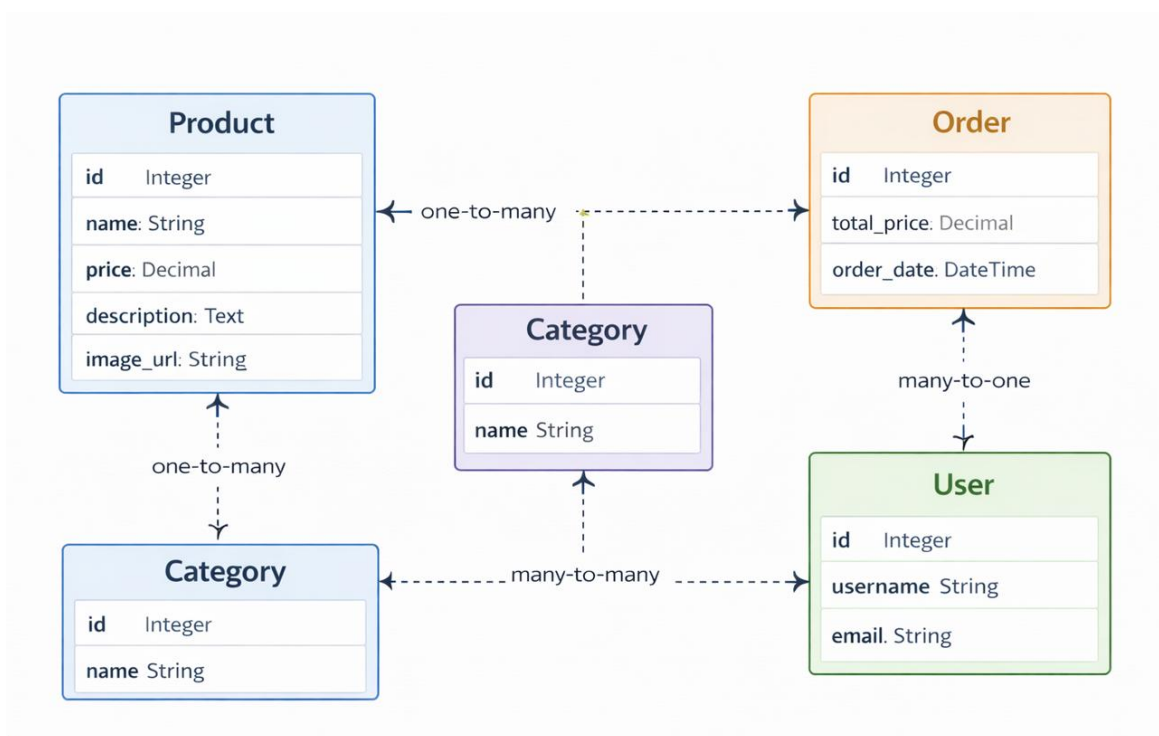
Μεταξύ των οντοτήτων Product και Category υφίσταται σχέση τύπου πολλά-προς-πολλά (many-to-many), καθώς ένα προϊόν μπορεί να ανήκει σε περισσότερες από μία κατηγορίες και κάθε κατηγορία μπορεί να περιλαμβάνει πολλαπλά προϊόντα. Η σχέση αυτή υποστηρίζεται από το Strapi μέσω μηχανισμών συσχέτισης περιεχομένου.

Η οντότητα Order αναπαριστά τις παραγγελίες που πραγματοποιούνται από τους χρήστες και περιλαμβάνει στοιχεία όπως το συνολικό κόστος και την ημερομηνία παραγγελίας. Κάθε παραγγελία συσχετίζεται με έναν χρήστη μέσω σχέσης τύπου πολλά-προς-ένα (many-to-one), καθώς ένας χρήστης μπορεί να πραγματοποιήσει πολλαπλές παραγγελίες, ενώ κάθε παραγγελία ανήκει σε έναν μόνο χρήστη.

Η οντότητα User αντιπροσωπεύει τους εγγεγραμμένους χρήστες του συστήματος και περιλαμβάνει βασικά στοιχεία ταυτοποίησης, όπως όνομα χρήστη και ηλεκτρονική διεύθυνση. Η διαχείριση των χρηστών υλοποιείται μέσω των ενσωματωμένων μηχανισμών αυθεντικοποίησης και εξουσιοδότησης του Strapi.

Το παρόν ER Diagram αποτυπώνει τη βασική δομή των δεδομένων του συστήματος και αποτελεί τη βάση για την υλοποίηση των Content Types στο Strapi, καθώς και για την

παραγωγή των αντίστοιχων API endpoints που χρησιμοποιούνται από το frontend της εφαρμογής.



Εικόνα 4. Διάγραμμα Οντοτήτων-Συσχετίσεων του συστήματος

Το διάγραμμα οντοτήτων–συσχετίσεων που παρουσιάζεται στην Εικόνα 4 παρέχει μία σαφή απεικόνιση της δομής των δεδομένων του συστήματος και διασφαλίζει τη συνοχή και την επεκτασιμότητα της αρχιτεκτονικής του ηλεκτρονικού καταστήματος.

Το μοντέλο δεδομένων σχεδιάζεται με τέτοιο τρόπο ώστε να υποστηρίζει επεκτάσεις και τροποποιήσεις στο μέλλον, διατηρώντας παράλληλα τη συνοχή και την ακεραιότητα των δεδομένων. Η χρήση του Strapi διευκολύνει τον ορισμό σχέσεων μεταξύ οντοτήτων και την αυτόματη παραγωγή αντίστοιχων API endpoints.

3.7. Σχεδιασμός διεπαφών και ροή δεδομένων

Η ροή δεδομένων μεταξύ του frontend και του backend βασίζεται στο μοντέλο αιτήματος–απόκρισης (request–response). Για τα δημόσια δεδομένα, όπως τα προϊόντα και οι κατηγορίες, το frontend αποστέλλει αιτήματα HTTP προς τα αντίστοιχα τελικά σημεία του Strapi και λαμβάνει αποκρίσεις σε μορφή JSON.

Για τις προστατευμένες λειτουργίες, όπως η δημιουργία και η ανάκτηση παραγγελιών, το αίτημα περιλαμβάνει διακριτικό JWT στην κεφαλίδα εξουσιοδότησης (Authorization

header). Το backend ελέγχει την αυθεντικοποίηση και την εξουσιοδότηση του χρήστη πριν από την εκτέλεση της ενέργειας.

Κατά τη διαδικασία ολοκλήρωσης αγοράς, το frontend αποστέλλει τα στοιχεία του καλαθιού στο backend. Το backend επανυπολογίζει το συνολικό ποσό, δημιουργεί τη σχετική ροή πληρωμής και, μετά την επιτυχή επιβεβαίωση, καταχωρίζει την παραγγελία με την αντίστοιχη κατάσταση.

Η χρήση σαφώς καθορισμένων τελικών σημείων API και η ανεξαρτησία των δύο επιπέδων διευκολύνουν τη συντήρηση, τη δοκιμή και τη μελλοντική επέκταση του συστήματος.

3.8. Συμπεράσματα κεφαλαίου

Στο παρόν κεφάλαιο καταγράφηκαν οι λειτουργικές και μη λειτουργικές απαιτήσεις της πλατφόρμας και παρουσιάστηκε ο αρχιτεκτονικός σχεδιασμός της. Προσδιορίστηκαν οι βασικοί ρόλοι και οι λειτουργίες του συστήματος, καθώς και τα χαρακτηριστικά ποιότητας που αφορούν την απόδοση, την ασφάλεια, τη συντηρησιμότητα και την επεκτασιμότητα.

Παράλληλα, αποτυπώθηκαν το μοντέλο δεδομένων και οι βασικές ροές επικοινωνίας μεταξύ του Strapi backend και του Next.js frontend. Ο διαχωρισμός των δύο επιπέδων και η επικοινωνία μέσω REST API δημιουργούν τη βάση για μία αρθρωτή εφαρμογή, η οποία μπορεί να επεκταθεί με νέες υπηρεσίες χωρίς να απαιτείται ριζική μεταβολή της βασικής αρχιτεκτονικής. Στο επόμενο κεφάλαιο παρουσιάζεται η υλοποίηση του backend με χρήση του Strapi.

4. Υλοποίηση Backend Συστήματος Με Strapi

Στο παρόν κεφάλαιο παρουσιάζεται αναλυτικά η υλοποίηση του backend της εφαρμογής ηλεκτρονικού εμπορίου με χρήση του Strapi. Το backend αποτελεί το βασικό επίπεδο διαχείρισης δεδομένων και επιχειρησιακής λογικής της πλατφόρμας και είναι υπεύθυνο για την αποθήκευση και οργάνωση των προϊόντων, των κατηγοριών, των χρηστών και των παραγγελιών.

Αρχικά περιγράφονται η εγκατάσταση και οι βασικές ρυθμίσεις του Strapi, καθώς και η διαμόρφωση του περιβάλλοντος ανάπτυξης. Στη συνέχεια παρουσιάζεται η σχεδίαση και δημιουργία των Content Types που αντιστοιχούν στις βασικές οντότητες του συστήματος, μαζί με τα πεδία και τις μεταξύ τους σχέσεις. Ιδιαίτερη αναφορά γίνεται στην καταχώρηση δοκιμαστικού περιεχομένου μέσω του Content Manager, ώστε να ελεγχθεί η λειτουργικότητα των δομών δεδομένων και η ορθή τροφοδότηση του frontend.

Παράλληλα, αναλύεται η διαχείριση χρηστών, ρόλων και δικαιωμάτων πρόσβασης μέσω του μηχανισμού Users & Permissions του Strapi. Παρουσιάζονται επίσης οι μηχανισμοί αυθεντικοποίησης με JSON Web Tokens, τα προστατευμένα endpoints και οι προσαρμοσμένοι controllers που υλοποιήθηκαν για τη διαχείριση παραγγελιών και την ασφαλή πρόσβαση στα δεδομένα κάθε χρήστη.

Τέλος, εξετάζεται η παροχή δεδομένων μέσω REST API και η διασύνδεση του backend με το Next.js frontend. Η υλοποίηση αυτή επιτρέπει τον σαφή διαχωρισμό της επιχειρησιακής λογικής από το επίπεδο παρουσίασης και δημιουργεί μία ευέλικτη βάση για την ενσωμάτωση επιπλέον λειτουργιών, όπως πληρωμές, προτάσεις προϊόντων, analytics και ανεξάρτητες serverless υπηρεσίες.

4.1. Εισαγωγή

Για την υλοποίηση του εξυπηρετητή της εφαρμογής επιλέχθηκε το **Strapi**, ένα σύγχρονο headless CMS ανοικτού κώδικα, το οποίο παρέχει δυνατότητες διαχείρισης περιεχομένου, δημιουργίας REST API, ελέγχου πρόσβασης και μηχανισμούς πιστοποίησης χρηστών. Η επιλογή του Strapi έγινε λόγω της ευελιξίας του στην ανάπτυξη API-first εφαρμογών, της εύκολης παραμετροποίησης των content types και της άμεσης συνεργασίας του με σύγχρονες τεχνολογίες front-end.

Η αρχιτεκτονική του back-end βασίστηκε στη λογική ότι το Strapi λειτουργεί ως κεντρικός κόμβος διαχείρισης δεδομένων και επιχειρησιακής λογικής, ενώ το front-end καταναλώνει

τα δεδομένα μέσω HTTP requests. Με τον τρόπο αυτό επιτυγχάνεται σαφής διαχωρισμός μεταξύ presentation layer και business logic, στοιχείο που αποτελεί βασικό χαρακτηριστικό της headless προσέγγισης.

4.2. Εγκατάσταση και βασικές ρυθμίσεις Strapi

Η εγκατάσταση του Strapi πραγματοποιήθηκε σε περιβάλλον Node.js, χρησιμοποιώντας το επίσημο εργαλείο δημιουργίας έργου. Κατά τη διαδικασία εγκατάστασης επιλέχθηκε βάση δεδομένων κατάλληλη για ανάπτυξη και δοκιμές, ενώ η αρχιτεκτονική του συστήματος επιτρέπει την εύκολη μετάβαση σε πιο ισχυρές βάσεις δεδομένων σε περιβάλλον παραγωγής.

Η ανάπτυξη του back-end ξεκίνησε με τη δημιουργία ενός Strapi project σε τοπικό περιβάλλον ανάπτυξης. Το σύστημα αρχικά ρυθμίστηκε ώστε να λειτουργεί με τις προεπιλεγμένες δυνατότητες του Strapi, ενώ στη συνέχεια επεκτάθηκε με νέα content types και custom controllers. Μετά την αρχική εγκατάσταση, δημιουργήθηκε λογαριασμός διαχειριστή μέσω του admin panel, από όπου πραγματοποιήθηκε το μεγαλύτερο μέρος της παραμετροποίησης του συστήματος.

Το περιβάλλον διαχείρισης του Strapi χρησιμοποιήθηκε για:

- τη δημιουργία των συλλογών δεδομένων,
- τον ορισμό σχέσεων μεταξύ οντοτήτων,
- τη ρύθμιση των ρόλων χρηστών,
- την ενεργοποίηση ή απενεργοποίηση δικαιωμάτων πρόσβασης σε endpoints.

4.3. Σχεδίαση και Δημιουργία Content Types

Η λογική δομή του back-end βασίστηκε σε τρεις κύριες οντότητες: **προϊόντα**, **κατηγορίες** και **παραγγελίες**. Παράλληλα αξιοποιήθηκε το ενσωματωμένο σύστημα χρηστών του plugin **Users & Permissions** του Strapi για τη διαχείριση των τελικών χρηστών του ηλεκτρονικού καταστήματος.

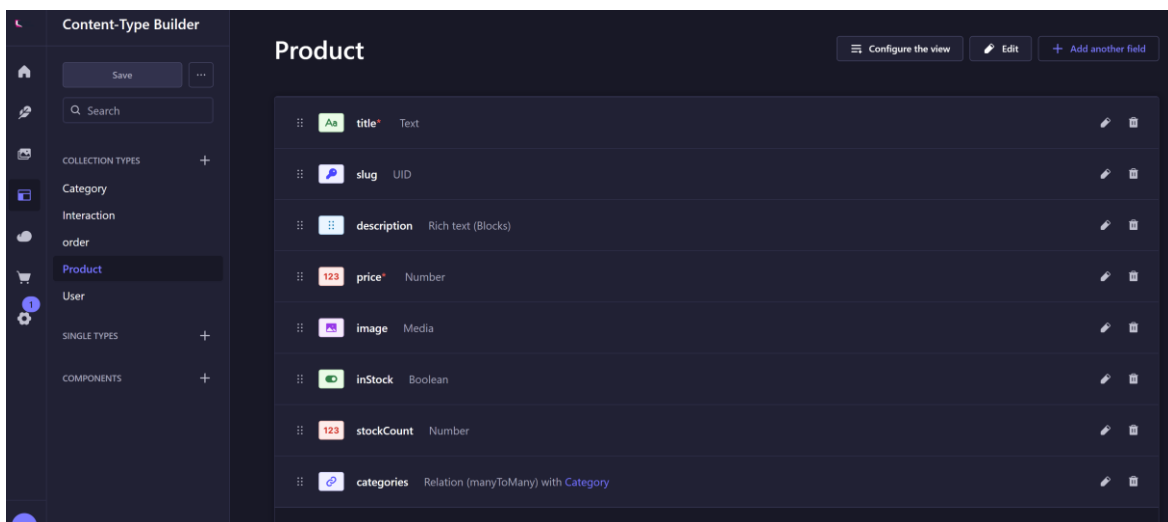
Το Content Type Product περιλαμβάνει πεδία για την αποθήκευση πληροφοριών σχετικά με τα προϊόντα του καταστήματος, όπως όνομα, τιμή, περιγραφή και εικόνες. Το Content Type Category χρησιμοποιείται για την κατηγοριοποίηση των προϊόντων, επιτρέποντας τη δημιουργία ιεραρχικής δομής περιεχομένου.

Προϊόν (Product)

Η οντότητα του προϊόντος περιλαμβάνει βασικά χαρακτηριστικά που απαιτούνται για την παρουσίαση και διάθεση προϊόντων σε ένα ηλεκτρονικό κατάστημα. Ειδικότερα, για κάθε προϊόν ορίστηκαν πεδία όπως:

- τίτλος,
- slug,
- περιγραφή,
- τιμή,
- διαθεσιμότητα,
- απόθεμα,
- εικόνα,
- συσχέτιση με κατηγορία.

Η χρήση του πεδίου **slug** κρίθηκε ιδιαίτερα σημαντική, καθώς επιτρέπει τη δημιουργία φιλικών URL και τη δυναμική φόρτωση σελίδων προϊόντων στο front-end.



Εικόνα 5. Η οντότητα του προϊόντος (product)

Κατηγορία (Category)

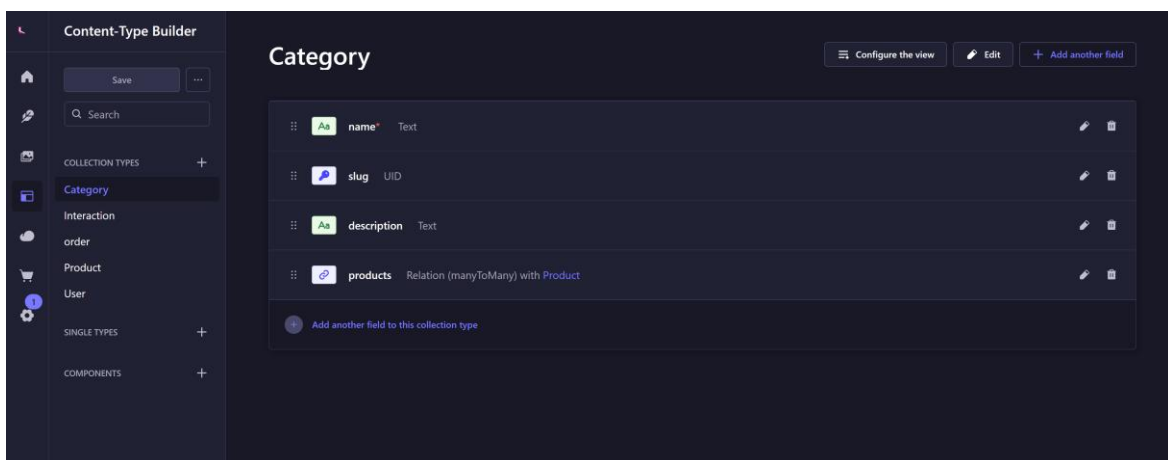
Η οντότητα κατηγορίας χρησιμοποιήθηκε για την ομαδοποίηση προϊόντων. Για κάθε κατηγορία καταχωρίζονται:

- όνομα,
- slug,

- περιγραφή.

Η σχέση μεταξύ κατηγορίας και προϊόντων επιτρέπει την ευκολότερη πλοήγηση του χρήστη, καθώς και τη μελλοντική επέκταση της εφαρμογής με φίλτρα ή category pages.

Για τη διαχείριση των παραγγελιών ορίστηκε το Content Type Order, το οποίο περιλαμβάνει πληροφορίες όπως το συνολικό κόστος και την ημερομηνία δημιουργίας της παραγγελίας. Οι χρήστες του συστήματος διαχειρίζονται μέσω των ενσωματωμένων μηχανισμών του Strapi, οι οποίοι παρέχουν έτοιμη λειτουργικότητα για αυθεντικοποίηση και εξουσιοδότηση.



Εικόνα 6. Η οντότητα κατηγορίας (category)

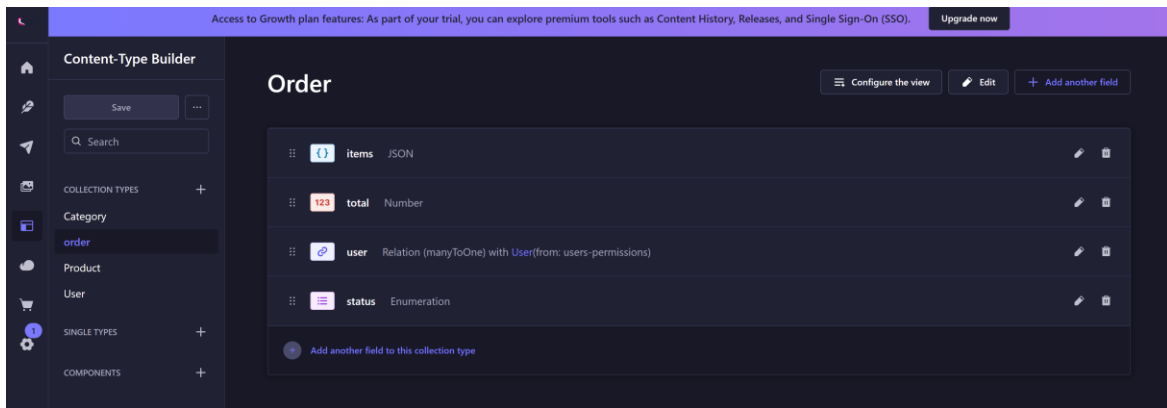
Παραγγελία (Order)

Η οντότητα παραγγελίας υλοποιήθηκε ώστε να αποθηκεύονται:

- τα προϊόντα της παραγγελίας σε μορφή JSON,
- το συνολικό ποσό,
- η κατάσταση της παραγγελίας,
- ο χρήστης στον οποίο ανήκει η παραγγελία.

Για το πεδίο κατάστασης χρησιμοποιήθηκε enumeration με τιμές όπως:

- pending,
- paid,
- cancelled.



Εικόνα 7. Η οντότητα παραγγελίας (order)

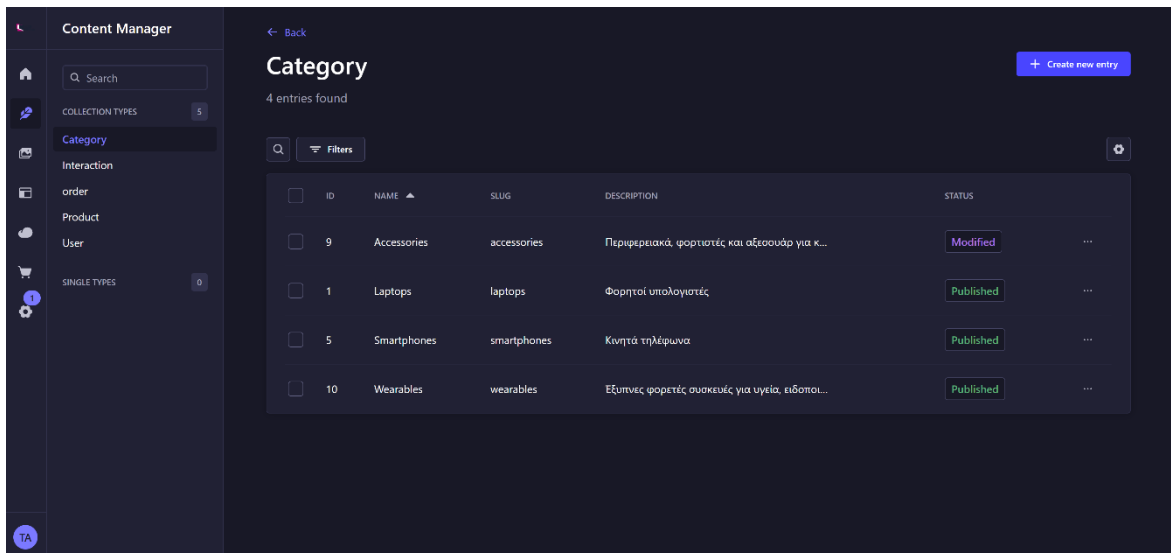
Με αυτόν τον τρόπο υποστηρίζεται βασική ροή διαχείρισης παραγγελιών και δημιουργείται υποδομή για μελλοντική ενσωμάτωση συστημάτων πληρωμών ή διαχείρισης logistics.

Αφού ολοκληρώθηκε ο ορισμός των Content Types, το επόμενο βήμα ήταν η καταχώρηση δοκιμαστικών δεδομένων μέσω του Content Manager, ώστε να ελεγχθεί η πρακτική λειτουργία του backend και η διασύνδεσή του με το frontend. (Η αναλυτική δομή του Content Type Order, τα πεδία του και η σχέση του με την οντότητα χρήστη παρουσιάζονται στον κώδικα του Παραρτήματος Α.1.)

4.4. Καταχώρηση Περιεχομένου μέσω Content Manager

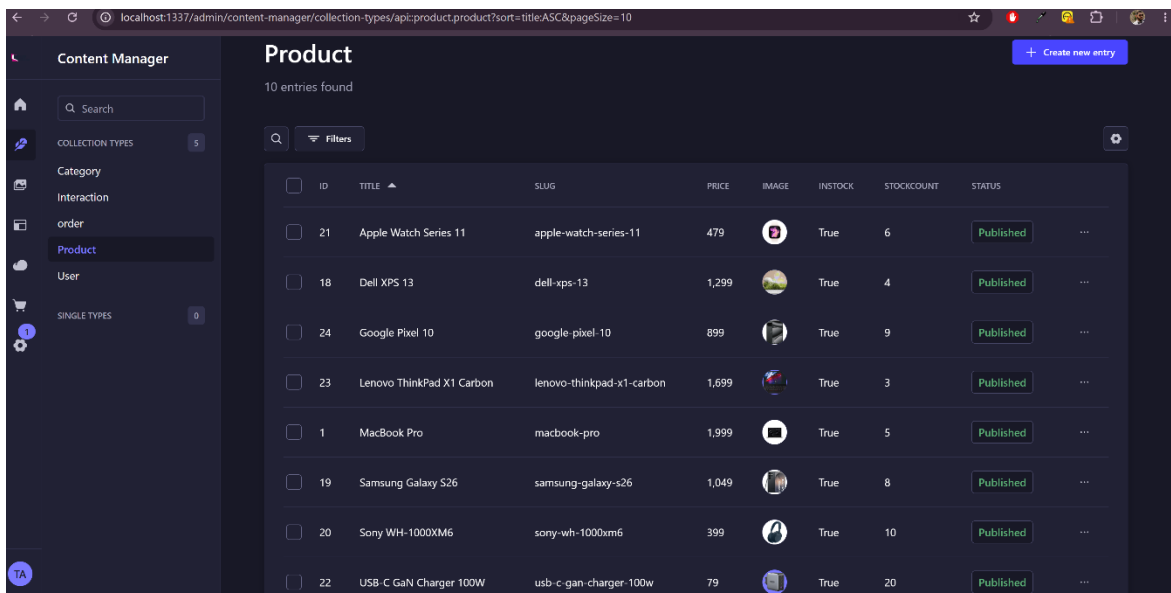
Μετά τη δημιουργία των βασικών Content Types μέσω του Content-Type Builder του Strapi, ακολούθησε η καταχώρηση πραγματικού περιεχομένου μέσω του Content Manager. Η διαδικασία αυτή ήταν απαραίτητη, καθώς επέτρεψε τη δημιουργία δοκιμαστικών δεδομένων για την τροφοδότηση του frontend, την επαλήθευση της ορθής λειτουργίας των σχέσεων μεταξύ οντοτήτων και τη δοκιμή των API endpoints που χρησιμοποιεί η εφαρμογή.

Αρχικά καταχωρήθηκαν οι βασικές κατηγορίες προϊόντων, οι οποίες χρησιμοποιούνται για την ομαδοποίηση και φιλτράρισμα του καταλόγου. Όπως παρουσιάζεται στην Εικόνα 8, στο Content Manager δημιουργήθηκαν ενδεικτικές κατηγορίες όπως Laptops, Smartphones, Wearables και Accessories. Κάθε κατηγορία περιλαμβάνει όνομα, slug και σύντομη περιγραφή, ώστε να μπορεί να αξιοποιηθεί τόσο στην παρουσίαση των προϊόντων όσο και στη δυναμική δρομολόγηση του frontend.



Εικόνα 8. Καταχώρηση κατηγοριών προϊόντων στο Strapi Content Manager

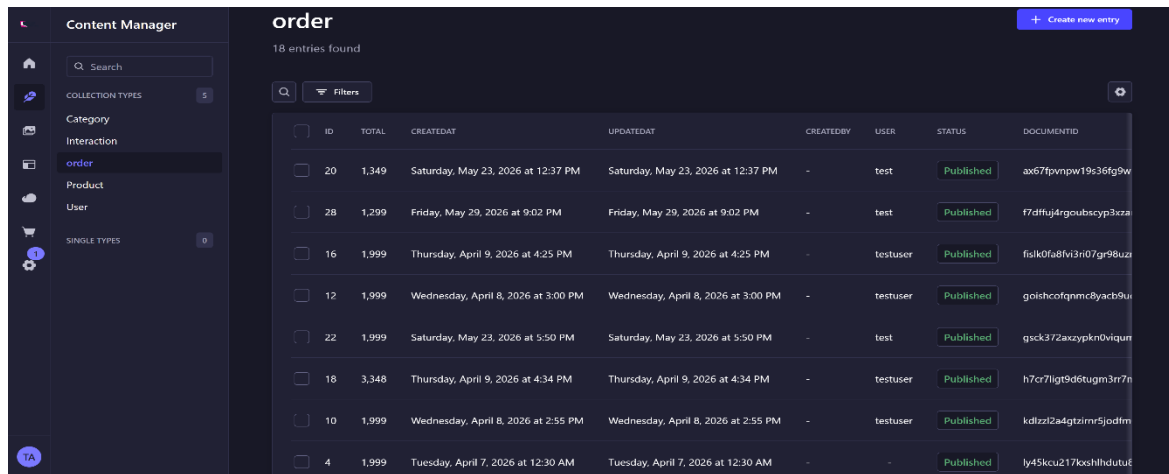
Στη συνέχεια πραγματοποιήθηκε η καταχώρηση προϊόντων, τα οποία αποτελούν το βασικό περιεχόμενο του ηλεκτρονικού καταστήματος. Όπως φαίνεται στην Εικόνα 9, κάθε προϊόν περιλαμβάνει τίτλο, slug, τιμή, εικόνα, διαθεσιμότητα και απόθεμα. Τα δεδομένα αυτά χρησιμοποιούνται από το Next.js frontend για την προβολή της λίστας προϊόντων, τη δημιουργία δυναμικών σελίδων προϊόντος και τη λειτουργία του καλαθιού αγορών.



Εικόνα 9. Καταχωρημένα προϊόντα στο Strapi Content Manager

Τέλος, μέσω του Content Manager είναι δυνατή και η προβολή των παραγγελιών που έχουν δημιουργηθεί από τους χρήστες της εφαρμογής. Στην Εικόνα 10 παρουσιάζεται η λίστα των καταχωρημένων παραγγελιών, όπου εμφανίζονται στοιχεία όπως το συνολικό ποσό, η

ημερομηνία δημιουργίας, ο χρήστης στον οποίο ανήκει η παραγγελία και η κατάσταση δημοσίευσης.



ID	TOTAL	CREATEDAT	UPDATEDAT	CREATEDBY	USER	STATUS	DOCUMENTID
20	1,349	Saturday, May 23, 2026 at 12:37 PM	Saturday, May 23, 2026 at 12:37 PM	-	test	Published	ax67fnpvnpw19s36f9w
28	1,299	Friday, May 29, 2026 at 9:02 PM	Friday, May 29, 2026 at 9:02 PM	-	test	Published	f7dfuj4rgoubscyp3zza
16	1,999	Thursday, April 9, 2026 at 4:25 PM	Thursday, April 9, 2026 at 4:25 PM	-	testuser	Published	fslk0fa8fvi3ri07gr98uza
12	1,999	Wednesday, April 8, 2026 at 3:00 PM	Wednesday, April 8, 2026 at 3:00 PM	-	testuser	Published	goishcofqnmcb9acb9u
22	1,999	Saturday, May 23, 2026 at 5:50 PM	Saturday, May 23, 2026 at 5:50 PM	-	test	Published	gsck37Zaxzypkndviqu
18	3,348	Thursday, April 9, 2026 at 4:34 PM	Thursday, April 9, 2026 at 4:34 PM	-	testuser	Published	h7α7lgt9d6tugm3rr7n
10	1,999	Wednesday, April 8, 2026 at 2:55 PM	Wednesday, April 8, 2026 at 2:55 PM	-	testuser	Published	kdtzzl2adgtzrmv5jodfm
4	1,999	Tuesday, April 7, 2026 at 12:30 AM	Tuesday, April 7, 2026 at 12:30 AM	-	-	Published	ly45kcu217koshldutuf

Εικόνα 10. Καταχωρημένες παραγγελίες στο Strapi Content Manager

Η ύπαρξη των παραγγελιών στο περιβάλλον διαχείρισης επιβεβαιώνει ότι το backend μπορεί να αποθηκεύει και να οργανώνει δεδομένα που προκύπτουν από τη λειτουργία checkout της εφαρμογής.

Η καταχώρηση περιεχομένου στο Strapi δεν περιορίζεται στη δοκιμή του backend, αλλά αποτελεί βασικό βήμα για την ολοκλήρωση της συνολικής ροής της εφαρμογής. Τα δεδομένα που δημιουργούνται στον Content Manager διατίθενται μέσω REST API στο frontend, επιτρέποντας τη δυναμική εμφάνιση προϊόντων, κατηγοριών και παραγγελιών. Με αυτόν τον τρόπο επιβεβαιώνεται η λειτουργική σύνδεση μεταξύ Strapi και Next.js, καθώς και η πρακτική αξιοποίηση των Content Types που δημιουργήθηκαν στο προηγούμενο στάδιο.

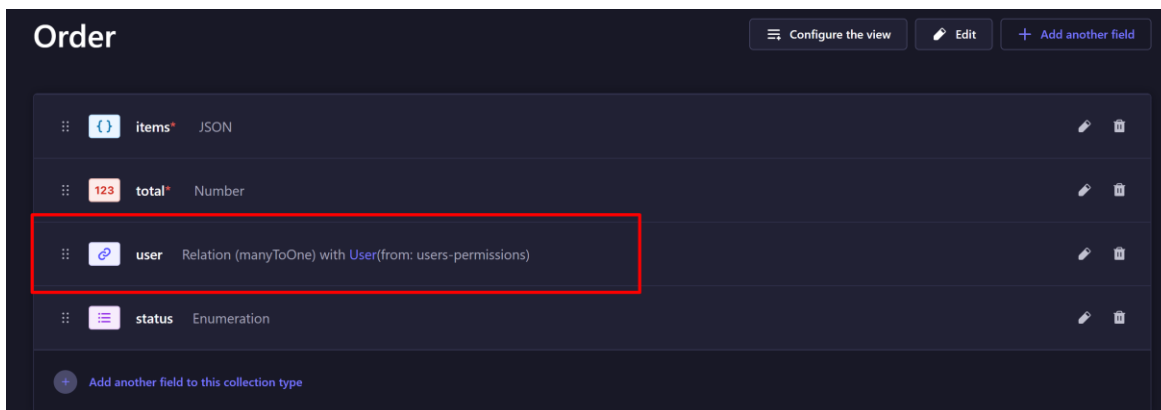
4.5. Σχέσεις μεταξύ Content Types

Το Strapi υποστηρίζει τον ορισμό σχέσεων μεταξύ Content Types, επιτρέποντας τη δημιουργία σύνθετων δομών δεδομένων. Στο πλαίσιο της παρούσας υλοποίησης, ορίστηκαν σχέσεις μεταξύ προϊόντων και κατηγοριών, καθώς και μεταξύ χρηστών και παραγγελιών.

Η σχέση μεταξύ Product και Category υλοποιήθηκε ως σχέση πολλά-προς-πολλά, επιτρέποντας σε κάθε προϊόν να ανήκει σε περισσότερες από μία κατηγορίες. Αντίστοιχα, η σχέση μεταξύ User και Order υλοποιήθηκε ως σχέση ένα-προς-πολλά, καθώς ένας χρήστης μπορεί να πραγματοποιήσει πολλαπλές παραγγελίες. (Η υλοποίηση της σχέσης

μεταξύ παραγγελίας και χρήστη αποτυπώνεται στο schema της οντότητας Order, το οποίο παρατίθεται στο Παράρτημα Α.1.)

Ο ορισμός των σχέσεων πραγματοποιήθηκε μέσω του γραφικού περιβάλλοντος του Strapi, γεγονός που απλοποιεί τη διαδικασία ανάπτυξης και διασφαλίζει τη συνέπεια των δεδομένων.



Εικόνα 11. Η σχέση μεταξύ χρήστη (User) και παραγγελίας (Order)

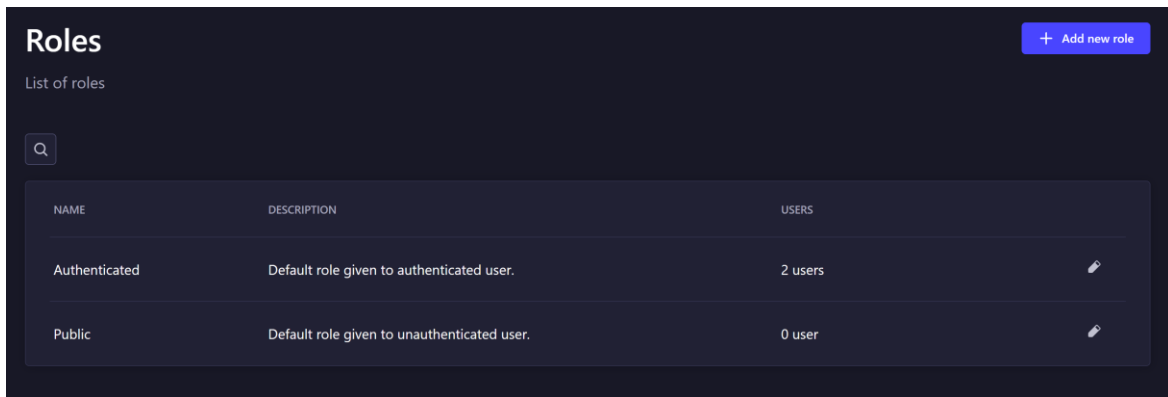
4.6. Διαχείριση χρηστών και δικαιωμάτων πρόσβασης

Η ασφάλεια του backend συστήματος επιτυγχάνεται μέσω του μηχανισμού Roles and Permissions που παρέχει το Strapi. Ο μηχανισμός αυτός επιτρέπει τον καθορισμό διαφορετικών ρόλων χρηστών και τον έλεγχο πρόσβασης σε συγκεκριμένα API endpoints.

Στην παρούσα υλοποίηση ορίστηκαν ρόλοι για διαχειριστές και απλούς χρήστες. Οι διαχειριστές έχουν πλήρη πρόσβαση στη διαχείριση περιεχομένου και παραγγελιών, ενώ οι απλοί χρήστες έχουν περιορισμένη πρόσβαση, σύμφωνα με τις απαιτήσεις του συστήματος.

Η αυθεντικοποίηση των χρηστών πραγματοποιείται με χρήση JSON Web Tokens (JWT), τα οποία διασφαλίζουν την ασφαλή επικοινωνία μεταξύ frontend και backend.

- **Public**, για μη πιστοποιημένους χρήστες,
- **Authenticated**, για πιστοποιημένους χρήστες.



Εικόνα 12. Ρόλοι χρηστών

Για τον ρόλο **Public** επιτράπηκε πρόσβαση μόνο σε λειτουργίες ανάγνωσης προϊόντων και κατηγοριών, όπως:

- ανάκτηση λίστας προϊόντων,
- ανάκτηση μεμονωμένου προϊόντος,
- ανάκτηση κατηγοριών.

Για τον ρόλο **Authenticated** ενεργοποιήθηκαν επιπλέον λειτουργίες όπως:

- δημιουργία παραγγελίας,
- ανάκτηση των προσωπικών παραγγελιών του χρήστη.

Με τον τρόπο αυτό διασφαλίστηκε ότι μόνο πιστοποιημένοι χρήστες μπορούν να ολοκληρώσουν αγορές και να έχουν πρόσβαση στο ιστορικό των παραγγελιών τους.

4.6.1. Υλοποίηση authentication

Για τη διαδικασία αυθεντικοποίησης χρησιμοποιήθηκαν τα built-in endpoints του Strapi:

- για εγγραφή χρήστη,
- για σύνδεση χρήστη,
- για παραγωγή JWT token.

Μετά την επιτυχή σύνδεση, το σύστημα επιστρέφει JWT token, το οποίο αποθηκεύεται στο front-end και αποστέλλεται στα προστατευμένα requests μέσω του header Authorization. Η χρήση JWT επιτρέπει ασφαλή επικοινωνία μεταξύ client και server χωρίς την ανάγκη διατήρησης session state στον εξυπηρετητή.

4.6.2. Custom endpoint για προστατευμένες παραγγελίες

Παρότι το Strapi παρέχει αυτόματα endpoints για κάθε content type, στην περίπτωση των παραγγελιών κρίθηκε αναγκαία η υλοποίηση custom endpoint ώστε κάθε χρήστης να έχει πρόσβαση μόνο στις δικές του παραγγελίες.

Για τον σκοπό αυτό δημιουργήθηκε custom route και custom controller action, τα οποία αξιοποιούν το `ctx.state.user` για να εντοπίσουν τον authenticated χρήστη και να επιστρέψουν μόνο τις παραγγελίες που σχετίζονται με το δικό του αναγνωριστικό. Με αυτόν τον τρόπο υλοποιήθηκε πραγματικός έλεγχος πρόσβασης σε επίπεδο backend και αποφεύχθηκε το σοβαρό πρόβλημα να είναι ορατές όλες οι παραγγελίες σε οποιονδήποτε πιστοποιημένο χρήστη.

Η συγκεκριμένη επιλογή αποτελεί ιδιαίτερα σημαντικό στοιχείο της υλοποίησης, καθώς εισάγει ασφαλή authorization logic σε μια εφαρμογή ηλεκτρονικού εμπορίου. (Ο προσαρμοσμένος controller και η αντίστοιχη διαδρομή για την ανάκτηση αποκλειστικά των παραγγελιών του πιστοποιημένου χρήστη παρατίθενται στα Παραρτήματα Α.3 και Α.4.)

4.6.3. Δημιουργία παραγγελίας

Η δημιουργία παραγγελίας πραγματοποιείται μέσω προστατευμένου endpoint, στο οποίο αποστέλλονται:

- τα προϊόντα του καλαθιού,
- η ποσότητα κάθε προϊόντος,
- η τιμή κάθε προϊόντος,
- το συνολικό ποσό.

Στο backend ο authenticated χρήστης ανακτάται από το token και η παραγγελία αποθηκεύεται συνδεδεμένη με τον σωστό λογαριασμό χρήστη. Η επιχειρησιακή αυτή λογική εξασφαλίζει ότι ο client δεν μπορεί να δηλώσει αυθαίρετα διαφορετικό ιδιοκτήτη παραγγελίας. (Ο βασικός controller δημιουργίας παραγγελίας και ο αντίστοιχος router του Content Type Order παρουσιάζονται στα Παραρτήματα Α.2 και Α.5.)

4.6.4. Διαχείριση κατάστασης παραγγελίας

Για την καλύτερη παρακολούθηση της ροής αγοράς, στην οντότητα παραγγελίας προστέθηκε πεδίο κατάστασης. Το πεδίο αυτό επιτρέπει:

- αρχική αποθήκευση της παραγγελίας ως `pending`,
- μελλοντική ενημέρωση σε `paid`,

- πιθανή μετάβαση σε `cancelled`.

Το συγκεκριμένο πεδίο βελτιώνει τόσο τη λειτουργικότητα της εφαρμογής όσο και την αναγνωσιμότητα των δεδομένων στο περιβάλλον διαχείρισης του Strapi.

4.7. Παροχή δεδομένων μέσω API

Το Strapi παρέχει αυτόματα REST και GraphQL APIs για κάθε Content Type που δημιουργείται. Στο πλαίσιο της παρούσας εργασίας, αξιοποιήθηκαν τα REST APIs για την επικοινωνία με το frontend της εφαρμογής.

Μέσω των API endpoints, το frontend μπορεί να ανακτά δεδομένα προϊόντων και κατηγοριών, να δημιουργεί και να διαχειρίζεται παραγγελίες, καθώς και να εκτελεί λειτουργίες αυθεντικοποίησης χρηστών. Η χρήση τυποποιημένων μορφών ανταλλαγής δεδομένων, όπως το JSON, διευκολύνει την ενσωμάτωση του backend με το Next.js frontend. (Η υλοποίηση των βασικών routes και controllers του Strapi που υποστηρίζουν την παροχή δεδομένων μέσω API παρουσιάζεται στο Παράρτημα Α.)

4.8. Ανάπτυξη και φιλοξενία του backend

Η αρχιτεκτονική του Strapi επιτρέπει την εύκολη ανάπτυξη του backend σε περιβάλλοντα παραγωγής. Το σύστημα μπορεί να φιλοξενηθεί σε σύγχρονες cloud υποδομές, παρέχοντας υψηλή διαθεσιμότητα και επεκτασιμότητα.

Κατά την ανάπτυξη λαμβάνονται υπόψη παράγοντες όπως η ασφάλεια της βάσης δεδομένων, η διαχείριση μεταβλητών περιβάλλοντος και η χρήση ασφαλών πρωτοκόλλων επικοινωνίας. Η δυνατότητα αυτο-φιλοξενίας του Strapi προσφέρει πλήρη έλεγχο στο backend και ευελιξία στην επιλογή υποδομής.

4.9. Συμπεράσματα κεφαλαίου

Στο παρόν κεφάλαιο παρουσιάστηκε η υλοποίηση του backend συστήματος του ηλεκτρονικού καταστήματος με χρήση του Strapi. Αναλύθηκαν η εγκατάσταση, ο ορισμός Content Types, η διαχείριση σχέσεων και χρηστών, καθώς και η παροχή δεδομένων μέσω API. Η υλοποίηση αυτή θέτει τις βάσεις για την ανάπτυξη του frontend της εφαρμογής, το οποίο παρουσιάζεται στο επόμενο κεφάλαιο.

5. Υλοποίηση Frontend Εφαρμογής Με Next.Js Ως Progressive Web Application

Στο παρόν κεφάλαιο παρουσιάζεται η υλοποίηση του frontend της πλατφόρμας ηλεκτρονικού εμπορίου με χρήση του Next.js. Το frontend αποτελεί το επίπεδο παρουσίασης και αλληλεπίδρασης του χρήστη με την εφαρμογή και είναι υπεύθυνο για την προβολή των προϊόντων, την πλοήγηση, τη διαχείριση του καλαθιού αγορών, την ολοκλήρωση παραγγελιών και την πρόσβαση στις λειτουργίες του λογαριασμού χρήστη.

Η εφαρμογή αναπτύχθηκε ως ανεξάρτητο επίπεδο παρουσίασης, το οποίο επικοινωνεί με το Strapi backend μέσω προγραμματιστικών διεπαφών REST. Η προσέγγιση αυτή επιτρέπει τον σαφή διαχωρισμό της διεπαφής χρήστη από τη διαχείριση των δεδομένων και την επιχειρησιακή λογική, ενισχύοντας τη συντηρησιμότητα και τη δυνατότητα ανεξάρτητης εξέλιξης των δύο επιπέδων.

Αρχικά παρουσιάζονται η επιλογή και η βασική ρύθμιση του Next.js, καθώς και η οργάνωση της εφαρμογής με χρήση του App Router. Στη συνέχεια αναλύεται η επικοινωνία με το Strapi API και η υλοποίηση των βασικών λειτουργιών του ηλεκτρονικού καταστήματος, όπως η ανάκτηση και παρουσίαση προϊόντων, οι δυναμικές σελίδες προϊόντων, το καλάθι αγορών, η διαδικασία checkout, η σύνδεση και εγγραφή χρηστών και η προβολή του ιστορικού και της κατάστασης των παραγγελιών.

Παράλληλα, εξετάζονται τα χαρακτηριστικά Progressive Web Application που ενσωματώθηκαν στην εφαρμογή, καθώς και οι τεχνικές βελτιστοποίησης απόδοσης, ασφάλειας και εμπειρίας χρήστη. Ιδιαίτερη έμφαση δίνεται στη διαχείριση των προστατευμένων αιτημάτων, στη χρήση JWT για την αυθεντικοποίηση των χρηστών και στην ασφαλή επικοινωνία με το backend.

Η υλοποίηση του frontend ολοκληρώνει τη βασική λειτουργική ροή της πλατφόρμας και δημιουργεί το περιβάλλον μέσα από το οποίο ο χρήστης αλληλεπιδρά με τα δεδομένα και τις υπηρεσίες του συστήματος. Παράλληλα, η ανεξάρτητη αρχιτεκτονική του επιτρέπει την ενσωμάτωση επιπλέον λειτουργιών, όπως προτάσεις προϊόντων, A/B testing, πληρωμές μέσω Stripe και διασύνδεση με serverless υπηρεσίες.

5.1. Εισαγωγή

Στο παρόν κεφάλαιο παρουσιάζεται η υλοποίηση του frontend της εφαρμογής ηλεκτρονικού καταστήματος με χρήση του Next.js, το οποίο αξιοποιείται ως Progressive

Web Application (PWA). Το frontend αποτελεί το επίπεδο αλληλεπίδρασης του χρήστη με το σύστημα και είναι υπεύθυνο για την παρουσίαση του περιεχομένου, την πλοήγηση και την παροχή μίας αποδοτικής και φιλικής εμπειρίας χρήστη.

Η επιλογή του Next.js βασίστηκε στις δυνατότητες που προσφέρει για υλοποίηση σύγχρονων web εφαρμογών υψηλής απόδοσης, όπως η υποστήριξη server-side rendering, static site generation και προηγμένων μηχανισμών βελτιστοποίησης.

Το front-end σχεδιάστηκε ώστε να λειτουργεί ως ανεξάρτητη εφαρμογή παρουσίασης, η οποία ανακτά δεδομένα από το Strapi μέσω REST API. Η συνολική λογική του συστήματος ακολουθεί τον διαχωρισμό:

- Strapi για δεδομένα και επιχειρησιακή λογική,
- Next.js για παρουσίαση, πλοήγηση και αλληλεπίδραση χρήστη.

5.2. Επιλογή και ρύθμιση του Next.js

Το Next.js αποτελεί ένα σύγχρονο framework βασισμένο στη βιβλιοθήκη React και επιτρέπει την ανάπτυξη εφαρμογών με βελτιστοποιημένη απόδοση και βελτιωμένη εμπειρία χρήστη. Η αρχική ρύθμιση του έργου πραγματοποιήθηκε με τη χρήση των επίσημων εργαλείων δημιουργίας εφαρμογών, ενώ η δομή του έργου οργανώθηκε σύμφωνα με τις βέλτιστες πρακτικές του framework.

Η αρχιτεκτονική του frontend σχεδιάστηκε με γνώμονα την ευκολία συντήρησης και επέκτασης, επιτρέποντας τη σαφή διάκριση μεταξύ σελίδων, επαναχρησιμοποιήσιμων components και βοηθητικών συναρτήσεων για την επικοινωνία με το backend.

5.3. Επικοινωνία Frontend με το Strapi Backend

Η επικοινωνία μεταξύ του frontend και του backend πραγματοποιείται μέσω των REST APIs που παρέχει το Strapi. Το Next.js αποστέλλει αιτήματα για την ανάκτηση δεδομένων προϊόντων, κατηγοριών και παραγγελιών, καθώς και για τη διαχείριση της αυθεντικοποίησης των χρηστών.

Τα δεδομένα ανταλλάσσονται σε μορφή JSON, επιτρέποντας την εύκολη επεξεργασία και παρουσίασή τους στο περιβάλλον χρήστη. Για τη βελτιστοποίηση της απόδοσης, αξιοποιούνται τεχνικές όπως η προσωρινή αποθήκευση δεδομένων και η

επαναχρησιμοποίηση αιτημάτων, όπου αυτό είναι εφικτό. (Η βοηθητική συνάρτηση επικοινωνίας με το Strapi API, καθώς και η διαχείριση των διευθύνσεων των πολυμέσων, παρουσιάζονται στο Παράρτημα Β.1.)

5.4. Υλοποίηση βασικών λειτουργιών ηλεκτρονικού καταστήματος

Το frontend της εφαρμογής υλοποιεί τις βασικές λειτουργίες ενός σύγχρονου ηλεκτρονικού καταστήματος. Οι χρήστες μπορούν να περιηγηθούν στον κατάλογο προϊόντων, να προβάλουν αναλυτικές πληροφορίες για κάθε προϊόν και να πραγματοποιήσουν αναζητήσεις βάσει κατηγοριών.

Η λειτουργία καλαθιού αγορών υλοποιείται στο frontend, επιτρέποντας την προσθήκη και αφαίρεση προϊόντων, καθώς και την προβολή του συνολικού κόστους της παραγγελίας. Η κατάσταση του καλαθιού διατηρείται τοπικά, ώστε να διασφαλίζεται η ομαλή εμπειρία χρήστη ακόμη και σε περιπτώσεις προσωρινής απώλειας σύνδεσης.

5.4.1 Η εφαρμογή υλοποιήθηκε με χρήση του App Router του Next.js.

Η βασική δομή περιλαμβάνει σελίδες για:

- αρχική προβολή προϊόντων,
- δυναμική σελίδα μεμονωμένου προϊόντος,
- καλάθι,
- checkout,
- σύνδεση και εγγραφή,
- ιστορικό παραγγελιών

Η προσέγγιση αυτή επέτρεψε καλύτερο διαχωρισμό λειτουργιών και πιο καθαρή ανάπτυξη κάθε σελίδας.

5.4.2 Ανάκτηση προϊόντων

Η λίστα των προϊόντων υλοποιήθηκε μέσω fetch requests προς το Strapi REST API. Για την ανάκτηση των σχετικών πεδίων και μέσων χρησιμοποιήθηκε κατάλληλο query string με populate, ώστε η απόκριση του API να περιλαμβάνει εικόνες και σχέσεις με κατηγορίες.

Τα προϊόντα αποδόθηκαν σε μορφή grid με πληροφορίες όπως:

- τίτλος
- τιμή
- εικόνα
- διαθεσιμότητα

Με αυτόν τον τρόπο επιτεύχθηκε καθαρή και λειτουργική σελίδα λίστας προϊόντων. (Ο κώδικας ανάκτησης και απόδοσης της λίστας προϊόντων από το Strapi API παρατίθεται στο Παράρτημα Β.2.)

5.4.3 Δυναμική σελίδα προϊόντος

Για την προβολή αναλυτικών στοιχείων κάθε προϊόντος χρησιμοποιήθηκε δυναμικό route βάσει του slug. Η σελίδα προϊόντος ανακτά το προϊόν που αντιστοιχεί στο συγκεκριμένο slug και εμφανίζει:

- όνομα προϊόντος
- κατηγορία
- τιμή
- εικόνα
- κατάσταση αποθέματος

Η χρήση δυναμικών routes συνέβαλε τόσο στη βελτιωμένη εμπειρία χρήστη όσο και στη δημιουργία καθαρών URL. (Η υλοποίηση της δυναμικής σελίδας προϊόντος, η οποία ανακτά τα δεδομένα βάσει του πεδίου slug, παρουσιάζεται στο Παράρτημα Β.3.)

5.4.4 Καλάθι αγορών

Η λειτουργία καλαθιού υλοποιήθηκε στην πλευρά του client με χρήση του localStorage. Το καλάθι επιτρέπει:

- προσθήκη προϊόντων
- αύξηση/μείωση ποσότητας
- αφαίρεση προϊόντος
- καθαρισμό καλαθιού
- υπολογισμό συνολικού ποσού

Η επιλογή του `localStorage` κρίθηκε επαρκής για το επίπεδο της παρούσας εφαρμογής, καθώς επιτρέπει απλή και αποδοτική διατήρηση της κατάστασης του καλαθιού μεταξύ διαφορετικών σελίδων. (Η λογική διαχείρισης του καλαθιού αγορών και της προσωρινής αποθήκευσης των επιλεγμένων προϊόντων παρατίθεται στο Παράρτημα Β.4.)

5.4.5 Checkout

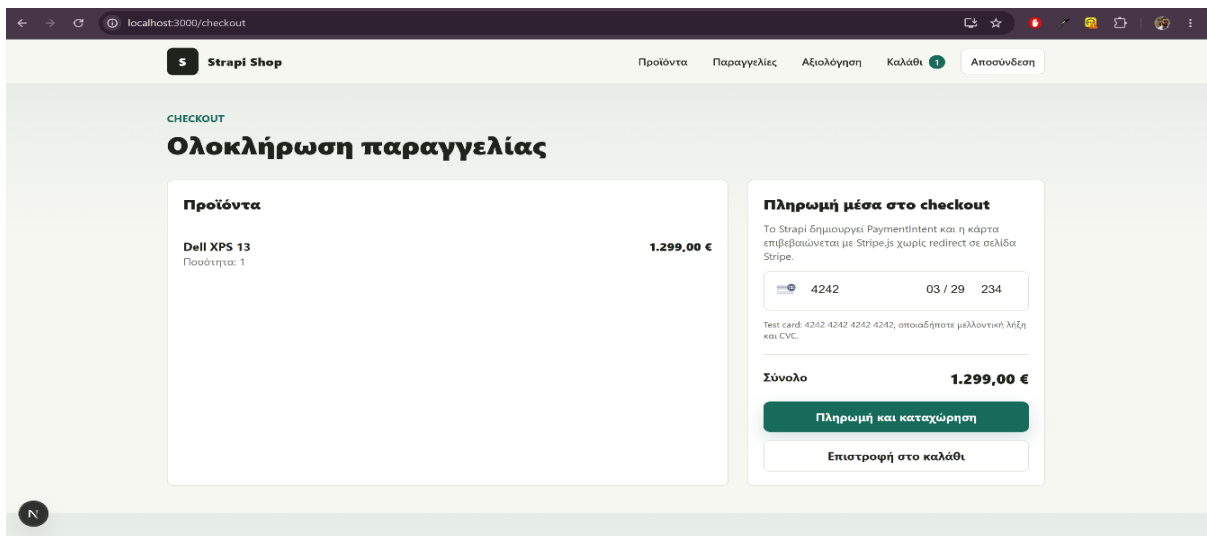
Η σελίδα checkout υλοποιήθηκε ώστε να εμφανίζει τα προϊόντα του καλαθιού και να αποστέλλει την παραγγελία στο Strapi μόνο όταν ο χρήστης είναι πιστοποιημένος. Για τον σκοπό αυτό χρησιμοποιείται το αποθηκευμένο JWT token. Αν ο χρήστης δεν είναι συνδεδεμένος, το front-end τον ενημερώνει ότι απαιτείται login.

Κατά την ολοκλήρωση της παραγγελίας:

1. ανακτώνται τα στοιχεία του καλαθιού
2. υπολογίζεται το τελικό ποσό
3. αποστέλλεται POST request στο backend
4. το Strapi αποθηκεύει την παραγγελία
5. το καλάθι αδειάζει

Η συγκεκριμένη ροή προσομοιώνει βασική λειτουργία πραγματικού ηλεκτρονικού καταστήματος.

Η λειτουργία checkout επεκτάθηκε με δοκιμαστική διασύνδεση πληρωμών μέσω Stripe. Η υλοποίηση βασίζεται σε embedded ροή πληρωμής, όπου ο χρήστης εισάγει τα στοιχεία κάρτας μέσα στη σελίδα ολοκλήρωσης παραγγελίας, ενώ το Strapi backend αναλαμβάνει τη δημιουργία και επιβεβαίωση του `PaymentIntent`. Η αναλυτική περιγραφή της ροής πληρωμής παρουσιάζεται στην Ενότητα 6.5. (Ο κώδικας της σελίδας checkout και της αποστολής των δεδομένων της παραγγελίας προς το backend παρουσιάζεται στο Παράρτημα Β.5.)



Εικόνα 13. Stripe checkout

5.4.6 Login και Register

Υλοποιήθηκαν δύο ξεχωριστές σελίδες για:

- σύνδεση χρήστη (Ο κώδικας της σελίδας σύνδεσης χρήστη και της αποθήκευσης του JWT token παρουσιάζεται στο Παράρτημα Γ.1.)
- εγγραφή νέου χρήστη (Ο κώδικας της σελίδας εγγραφής νέου χρήστη και της επικοινωνίας με το αντίστοιχο endpoint του Strapi παρατίθεται στο Παράρτημα Γ.2.)

Το frontend αποστέλλει τα διαπιστευτήρια σύνδεσης (credentials) στα αντίστοιχα endpoints του Strapi και αποθηκεύει το διακριτικό JWT (JWT token) και τα βασικά στοιχεία του χρήστη. Η αποθήκευση αυτή επιτρέπει τη μετέπειτα πραγματοποίηση προστατευμένων αιτημάτων (requests), όπως η δημιουργία παραγγελίας και η ανάκτηση ιστορικού αγορών.

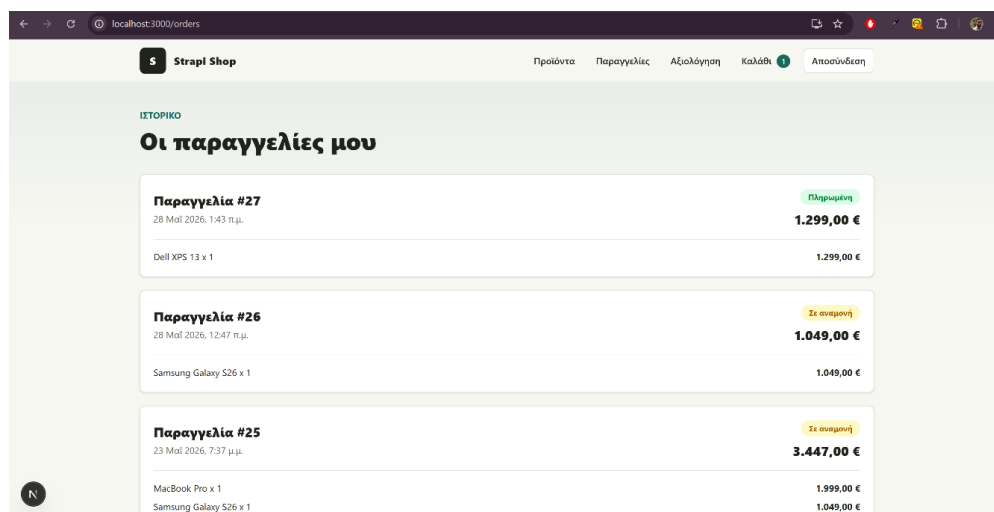
5.4.7 Προβολή ιστορικού παραγγελιών

Η σελίδα “Οι παραγγελίες μου” υλοποιήθηκε ως προστατευμένη σελίδα front-end, η οποία αποστέλλει αίτημα στο ειδικό endpoint του backend για ανάκτηση μόνο των παραγγελιών του τρέχοντος χρήστη. Η σελίδα περιλαμβάνει:

- loading state
- error state

- empty state
- λίστα παραγγελιών με ημερομηνία, σύνολο και προϊόντα

Με αυτόν τον τρόπο βελτιώθηκε αισθητά η εμπειρία χρήστη, καθώς κάθε πελάτης μπορεί να έχει άμεση εικόνα του ιστορικού των αγορών του. (Η υλοποίηση της προστατευμένης σελίδας ιστορικού παραγγελιών και η απόδοση των στοιχείων κάθε παραγγελίας παρουσιάζονται στο Παράρτημα Β.6.)



Εικόνα 14. Ιστορικό και κατάσταση παραγγελιών εκάστοτε χρήστη στο front end

5.4.8 Προβολή κατάστασης παραγγελίας

Στη σελίδα ιστορικού παραγγελιών ενσωματώθηκε και η απεικόνιση της κατάστασης κάθε παραγγελίας μέσω visual badge. Για παράδειγμα:

- οι παραγγελίες σε αναμονή εμφανίζονται με κίτρινο
- οι πληρωμένες με πράσινο,
- οι ακυρωμένες με κόκκινο.

Η επιλογή αυτή ενισχύει τη χρηστικότητα του συστήματος και καθιστά πιο εύκολη την παρακολούθηση της πορείας μιας παραγγελίας. (Η λογική εμφάνισης της κατάστασης κάθε παραγγελίας ως πληρωμένης, σε αναμονή ή ακυρωμένης παρουσιάζεται στο Παράρτημα Β.6.)

5.5. Υλοποίηση χαρακτηριστικών Progressive Web Application

Η εφαρμογή αναπτύχθηκε ως Progressive Web Application, αξιοποιώντας χαρακτηριστικά που βελτιώνουν σημαντικά την εμπειρία χρήστη. Η χρήση service workers επιτρέπει την προσωρινή αποθήκευση βασικών πόρων της εφαρμογής, εξασφαλίζοντας ταχύτερους χρόνους φόρτωσης και βασική offline λειτουργία.

Επιπλέον, η εφαρμογή υποστηρίζει εγκατάσταση στην αρχική οθόνη της συσκευής του χρήστη μέσω web manifest, προσφέροντας εμπειρία παρόμοια με εγγενείς εφαρμογές. Τα χαρακτηριστικά αυτά καθιστούν την εφαρμογή ιδιαίτερα κατάλληλη για χρήση σε κινητές συσκευές.

Κατά την ανάπτυξη της εφαρμογής δόθηκε ιδιαίτερη προσοχή στον τρόπο φόρτωσης εικόνων προϊόντων από το Strapi. Αρχικά προέκυψαν ζητήματα συμβατότητας με τη βελτιστοποίηση εικόνων του Next.js, τα οποία επιλύθηκαν μέσω κατάλληλης προσαρμογής του τρόπου εμφάνισης των media. Η εμπειρία αυτή ανέδειξε την ανάγκη σωστής ρύθμισης της επικοινωνίας μεταξύ headless CMS και front-end framework, ιδιαίτερα όταν χρησιμοποιούνται δυναμικά media URLs.

5.6. Βελτιστοποίηση απόδοσης και εμπειρίας χρήστη

Η απόδοση της εφαρμογής αποτελεί βασικό παράγοντα επιτυχίας ενός ηλεκτρονικού καταστήματος. Στο πλαίσιο της παρούσας υλοποίησης, αξιοποιήθηκαν οι δυνατότητες του Next.js για βελτιστοποίηση φόρτωσης σελίδων και πόρων.

Η χρήση τεχνικών όπως η στατική παραγωγή σελίδων και η βελτιστοποιημένη διαχείριση εικόνων συμβάλλει στη μείωση των χρόνων απόκρισης και στη βελτίωση των μετρικών Core Web Vitals. Παράλληλα, η σχεδίαση της διεπαφής χρήστη πραγματοποιήθηκε με έμφαση στην απλότητα και τη σαφήνεια, ενισχύοντας τη χρηστικότητα της εφαρμογής.

5.7. Ασφάλεια στο επίπεδο του Frontend

Η ασφάλεια στο frontend αποτελεί σημαντικό στοιχείο του συνολικού συστήματος. Η εφαρμογή διαχειρίζεται τα δεδομένα αυθεντικοποίησης με προσοχή, αποφεύγοντας την αποθήκευση ευαίσθητων πληροφοριών σε μη ασφαλείς περιοχές.

Η επικοινωνία με το backend πραγματοποιείται αποκλειστικά μέσω ασφαλών συνδέσεων, ενώ οι μηχανισμοί ελέγχου πρόσβασης υλοποιούνται σε συνεργασία με το backend σύστημα του Strapi. Η προσέγγιση αυτή διασφαλίζει την προστασία των δεδομένων των χρηστών και τη συνολική αξιοπιστία της εφαρμογής.

5.8. Ανάπτυξη και φιλοξενία του Frontend

Το frontend της εφαρμογής μπορεί να αναπτυχθεί και να φιλοξενηθεί σε σύγχρονες πλατφόρμες φιλοξενίας που υποστηρίζουν εφαρμογές Next.js. Η διαδικασία ανάπτυξης περιλαμβάνει τη δημιουργία περιβάλλοντος παραγωγής, τη διαχείριση μεταβλητών περιβάλλοντος και τη σύνδεση με το backend σύστημα.

Η ανεξαρτησία του frontend από το backend επιτρέπει την ευέλικτη ανάπτυξη και κλιμάκωση του συστήματος, ανάλογα με τις ανάγκες χρήσης και τον φόρτο της εφαρμογής.

Κατά την υλοποίηση αντιμετωπίστηκαν αρκετές πρακτικές δυσκολίες, όπως:

- σφάλματα συσχετίσεων μεταξύ content types,
- προβλήματα συγχρονισμού των generated types στο Strapi,
- ασυμβατότητες ανάμεσα σε schema αλλαγές και cached metadata,
- προβλήματα με dynamic routes όταν δεν είχαν οριστεί σωστά τα slugs των προϊόντων,
- προσαρμογές στη διαχείριση εικόνων.

Η επίλυση των παραπάνω προβλημάτων οδήγησε σε βαθύτερη κατανόηση της λειτουργίας ενός headless συστήματος και ανέδειξε τη σημασία της σωστής σχεδίασης του data model από τα πρώτα στάδια της ανάπτυξης.

5.9. Συμπεράσματα κεφαλαίου

Στο παρόν κεφάλαιο παρουσιάστηκε η υλοποίηση του frontend της εφαρμογής ηλεκτρονικού καταστήματος με χρήση του Next.js ως Progressive Web Application. Αναλύθηκαν οι βασικές λειτουργίες, τα χαρακτηριστικά PWA, οι τεχνικές βελτιστοποίησης απόδοσης και οι μηχανισμοί ασφάλειας. Η υλοποίηση αυτή, σε συνδυασμό με το backend σύστημα που παρουσιάστηκε στο προηγούμενο κεφάλαιο, ολοκληρώνει την ανάπτυξη του προτεινόμενου συστήματος ηλεκτρονικού εμπορίου.

Η υλοποίηση του συστήματος απέδειξε ότι η συνδυαστική χρήση Strapi και Next.js αποτελεί αποτελεσματική λύση για την ανάπτυξη σύγχρονου ηλεκτρονικού καταστήματος. Το back-end παρέχει δομημένη διαχείριση περιεχομένου, ασφαλή μηχανισμό πιστοποίησης και δυνατότητα επέκτασης με custom logic. Το front-end προσφέρει γρήγορη, ευέλικτη και φιλική εμπειρία χρήστη, αξιοποιώντας τις δυνατότητες του React οικοσυστήματος.

Η αρχιτεκτονική αυτή επιτρέπει:

- ευκολότερη συντήρηση του συστήματος,
- καθαρό διαχωρισμό αρμοδιοτήτων,
- καλύτερη επεκτασιμότητα,
- δυνατότητα μελλοντικής ενσωμάτωσης πρόσθετων υπηρεσιών όπως online πληρωμές, ειδοποιήσεις e-mail και αναλυτικά dashboards.

Συνεπώς, η επιλεγμένη προσέγγιση κρίνεται κατάλληλη όχι μόνο για πειραματική ή ακαδημαϊκή χρήση, αλλά και ως βάση για πραγματική εφαρμογή ηλεκτρονικού εμπορίου.

6. Πειραματικές Επεκτάσεις, Πληρωμές και Αξιολόγηση της Πλατφόρμας

Στο παρόν κεφάλαιο παρουσιάζονται οι πειραματικές επεκτάσεις που προστέθηκαν στην πλατφόρμα ηλεκτρονικού εμπορίου, με στόχο τη διερεύνηση πιο σύγχρονων τεχνικών εξατομίκευσης, αξιολόγησης εμπειρίας χρήστη και αρχιτεκτονικής επεκτασιμότητας. Οι επεκτάσεις αυτές περιλαμβάνουν την υλοποίηση πειραματικού μηχανισμού προτάσεων προϊόντων, την ενσωμάτωση A/B testing, τη δημιουργία ανεξάρτητου evaluation toolkit για μετρήσεις απόδοσης και ασφάλειας, καθώς και την ενδεικτική αποτύπωση μιας serverless και composable αρχιτεκτονικής σύμφωνα με τις αρχές MACH

6.1. Πειραματικός AI Recommender και A/B Testing

Στο πλαίσιο της επέκτασης της λειτουργικότητας της πλατφόρμας προστέθηκε πειραματική λειτουργία προτάσεων προϊόντων. Η λειτουργία αυτή βασίζεται σε τεχνικές item-based και session-based collaborative filtering, με στόχο την παραγωγή σχετικών προτάσεων με βάση τις αλληλεπιδράσεις του χρήστη με το σύστημα.

Ο μηχανισμός προτάσεων αξιοποιεί δεδομένα αλληλεπίδρασης, όπως προβολές προϊόντων, προσθήκες στο καλάθι και αγορές. Όταν δεν υπάρχουν επαρκή δεδομένα αλληλεπίδρασης, το σύστημα εφαρμόζει εναλλακτικό μηχανισμό fallback, προτείνοντας προϊόντα ίδιας κατηγορίας, δημοφιλή προϊόντα ή πρόσφατα καταχωρισμένα προϊόντα. Με αυτόν τον τρόπο διασφαλίζεται ότι η ενότητα προτάσεων μπορεί να λειτουργεί ακόμη και σε περιπτώσεις περιορισμένων δεδομένων.

Παράλληλα, ενσωματώθηκε μηχανισμός A/B testing για την αξιολόγηση διαφορετικών παραλλαγών παρουσίασης των προτεινόμενων προϊόντων. Η ανάθεση παραλλαγής A ή B πραγματοποιείται ανά session μέσω browser storage, ώστε ο ίδιος χρήστης να βλέπει σταθερά την ίδια παραλλαγή κατά τη διάρκεια της περιήγησής του. Η λειτουργία αυτή επιτρέπει τη συλλογή πειραματικών δεδομένων UX και τη σύγκριση της αποτελεσματικότητας διαφορετικών τρόπων εμφάνισης των προτεινόμενων προϊόντων.

Στοιχείο	Περιγραφή
Interaction tracking	Καταγραφή view, add_to_cart, purchase, recommendation_impression και recommendation_click event
A/B variant	Ανάθεση παραλλαγής A ή B ανά session μέσω browser storage

Recommendation endpoint	Νέο endpoint /api/recommendations για παραγωγή προτάσεων προϊόντων.
Evaluation metrics	Υπολογισμός CTR προτάσεων, add-to-cart rate και conversion rate ανά variant.

Πίνακας 1 Στοιχεία πειραματικού recommender και A/B testing

A/B TESTING						
Μετρικές recommender UX						
Τα δεδομένα προέρχονται από interactions που γράφονται στο Strapi.						
Variant	Sessions	Views	Add to cart	Purchases	CTR προτάσεων	Conversion
A	1	1	0 (0.0%)	0	0.0%	0.0%
B	2	12	6 (300.0%)	4	0.0%	200.0%

Εικόνα 15. Μετρικές UX

Η λειτουργία αυτή ενισχύει τη δυνατότητα αξιολόγησης της εμπειρίας χρήστη, καθώς επιτρέπει τη συσχέτιση των προτάσεων προϊόντων με πραγματικές ενέργειες των χρηστών. Επιπλέον, παρέχει βάση για μελλοντική ενσωμάτωση πιο σύνθετων αλγορίθμων προτάσεων, αξιοποιώντας μεγαλύτερο όγκο δεδομένων αλληλεπίδρασης. (Ο κώδικας του recommendation endpoint, τα βάρη των αλληλεπιδράσεων και η λογική collaborative filtering παρουσιάζονται στο Παράρτημα Δ.1, ενώ η καταγραφή των interaction events και η ανάθεση παραλλαγής A/B παρουσιάζονται στο Παράρτημα Δ.2.)

6.2. Μετρική αξιολόγηση απόδοσης και ασφάλειας

Για την ποσοτική αξιολόγηση της πλατφόρμας προστέθηκε ανεξάρτητο evaluation toolkit, το οποίο εκτελεί επαναλαμβανόμενες μετρήσεις και παράγει αναφορές σε μορφή JSON και Markdown. Οι μετρήσεις αυτές προβάλλονται και στη σελίδα /evaluation του frontend, ώστε να είναι δυνατή η άμεση επισκόπηση των αποτελεσμάτων αξιολόγησης.

Το evaluation toolkit χρησιμοποιείται για τη μέτρηση κρίσιμων χαρακτηριστικών απόδοσης και ασφάλειας της εφαρμογής. Συγκεκριμένα, πραγματοποιείται μέτρηση Time-to-First-Byte (TTFB) για βασικές σελίδες του Next.js frontend, καθώς και μέτρηση latency για Strapi API endpoints, όπως products, recommendations και interaction summary. Παράλληλα, εκτελείται έλεγχος βασικών HTTP security headers τόσο για το frontend όσο και για το Strapi API.

Ιδιαίτερη σημασία δίνεται και στον έλεγχο προστατευμένων endpoints. Για τον σκοπό αυτό, το toolkit ελέγχει τη συμπεριφορά του endpoint παραγγελιών όταν δεν παρέχεται JWT

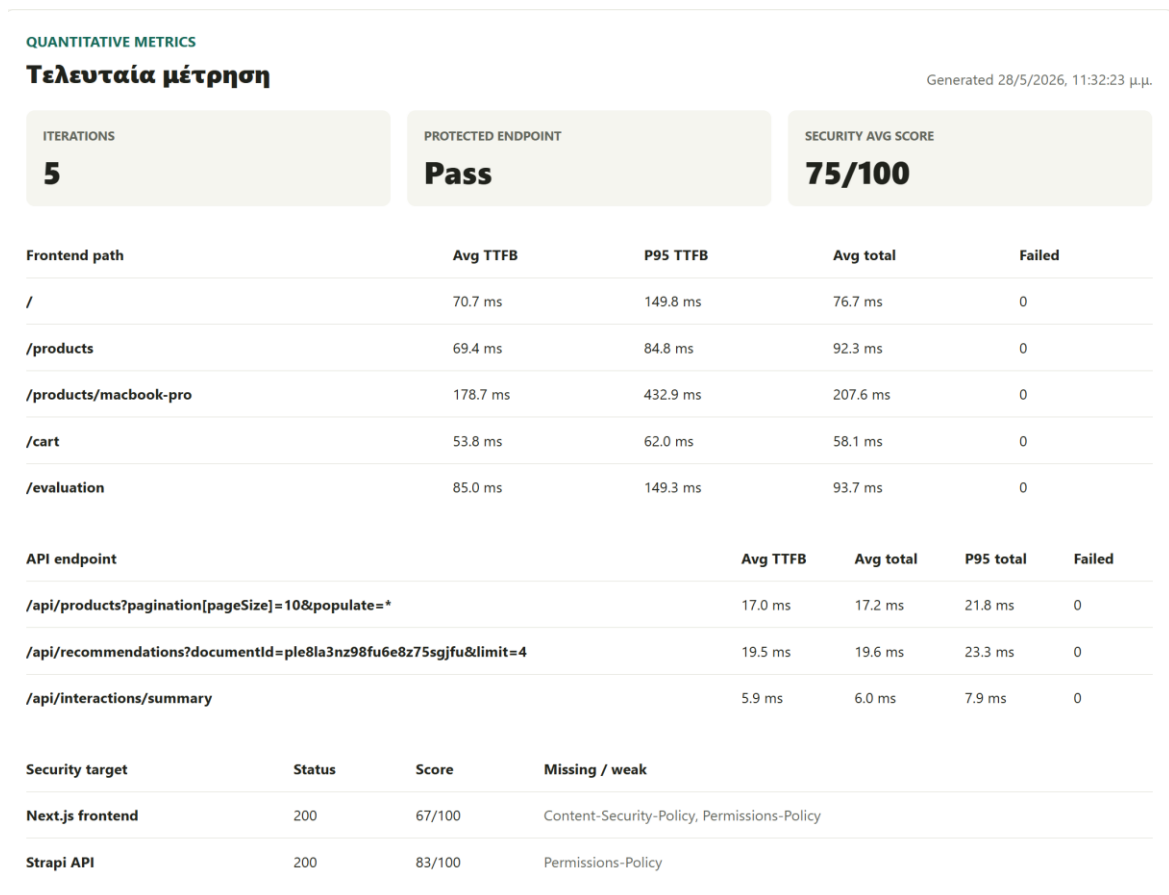
token, ώστε να επιβεβαιωθεί ότι η πρόσβαση σε προστατευμένους πόρους περιορίζεται μόνο σε πιστοποιημένους χρήστες.

Η εκτέλεση των μετρήσεων πραγματοποιείται με την εντολή:

npm run evaluate

Τα αποτελέσματα αποθηκεύονται στα ακόλουθα αρχεία:

evaluation/reports/latest.json
evaluation/reports/summary.md



Εικόνα 16. Μετρική αξιολόγηση απόδοσης και ασφάλειας

Η προσθήκη του evaluation toolkit επιτρέπει τη συστηματική και επαναλήψιμη αξιολόγηση της πλατφόρμας. Επιπλέον, παρέχει τη δυνατότητα παραγωγής συγκριτικής μελέτης έναντι παραδοσιακού monolithic CMS, συμβάλλοντας στην τεκμηρίωση των πλεονεκτημάτων της headless προσέγγισης ως προς την απόδοση, την ασφάλεια και την επεκτασιμότητα. (Η υλοποίηση του evaluation toolkit, οι μετρήσεις TTFB και latency, καθώς και η παραγωγή των αναφορών JSON και Markdown παρουσιάζονται στο Παράρτημα Δ.3.)

6.3. Συγκριτική αξιολόγηση με παραδοσιακό CMS WooCommerce

Στο πλαίσιο της αξιολόγησης της προτεινόμενης αρχιτεκτονικής πραγματοποιήθηκε ενδεικτική συγκριτική μέτρηση μεταξύ της υλοποιημένης πλατφόρμας και μιας εγκατάστασης WooCommerce, η οποία είναι διαθέσιμη στο woo.tagg.gr. Η σύγκριση αυτή συνδέεται με την αρχική απαίτηση της εργασίας για αξιολόγηση εναλλακτικών προσεγγίσεων ηλεκτρονικού εμπορίου με βάση κριτήρια όπως η απόδοση, η επεκτασιμότητα, η ασφάλεια και η δυνατότητα συντήρησης.

Η προτεινόμενη πλατφόρμα βασίζεται σε headless αρχιτεκτονική, με Strapi ως backend και Next.js ως ανεξάρτητο frontend. Αντίθετα, το WooCommerce αποτελεί παραδοσιακή CMS-based λύση, καθώς λειτουργεί ως πρόσθετο του WordPress και ενσωματώνει τη διαχείριση περιεχομένου, την εμπορική λογική και την παρουσίαση στο ίδιο οικοσύστημα. Η σύγκριση δεν αποσκοπεί στην απόλυτη κατάταξη των δύο λύσεων, αλλά στην ανάδειξη των αρχιτεκτονικών διαφορών και των επιπτώσεών τους σε πρακτικές μετρικές.

Για τη συγκριτική αξιολόγηση χρησιμοποιήθηκε το evaluation toolkit της εργασίας. Για κάθε σελίδα πραγματοποιήθηκε ένα warm-up request που εξαιρέθηκε από τα αποτελέσματα και στη συνέχεια 10 δείγματα ανά URL, χωρίς cookies, χωρίς ταυτόχρονα αιτήματα και με ακολουθία εναλλαγής μεταξύ των δύο πλατφορμών. Τα αποτελέσματα δημοσιεύονται στη σελίδα shop.tagg.gr/evaluation και μπορούν να εξαχθούν σε μορφή CSV και JSON.

Σελίδα	Next median	WooCommerce median	Λόγος	Μέγεθος HTML
Αρχική	203 ms	942 ms	4.6x	11 KB / 166 KB
Κατάλογος	347 ms	1031 ms	3.0x	56 KB / 261 KB
Προϊόν	477 ms	933 ms	2.0x	32 KB / 138 KB
Άδαιο καλάθι	204 ms	940 ms	4.6x	7 KB / 264 KB

Πίνακας 2. Συνοπτικά αποτελέσματα HTTP σύγκρισης Next.js/Strapi και WooCommerce

Τα αποτελέσματα δείχνουν ότι, στις συγκεκριμένες ενδεικτικές μετρήσεις, η headless υλοποίηση εμφανίζει χαμηλότερο διάμεσο χρόνο παραλαβής HTML σε όλες τις εξεταζόμενες σελίδες. Ενδεικτικά, στην αρχική σελίδα η διάμεση τιμή ήταν περίπου 203 ms για την πλατφόρμα Next.js/Strapi έναντι περίπου 942 ms για το WooCommerce, ενώ στον κατάλογο προϊόντων ήταν περίπου 347 ms έναντι 1031 ms αντίστοιχα.

Η αξιολόγηση πρέπει να ερμηνεύεται ως ενδεικτική και όχι ως πλήρες benchmark παραγωγικού φορτίου. Δεν περιλαμβάνει σύνθετα σενάρια ταυτόχρονων χρηστών, πλήρη φόρτωση εικόνων και JavaScript από πραγματικό browser, checkout συναλλαγές ή δοκιμές κλιμάκωσης υπό διαφορετικά επίπεδα traffic. Επιπλέον, οι δύο εγκαταστάσεις έχουν

διαφορετικά προϊόντα, εικόνες, templates, plugins και ενδεχομένως διαφορετικές ρυθμίσεις cache.

Κριτήριο	Strapi + Next.js	WooCommerce
Αρχιτεκτονική	Headless διαχωρισμός frontend και backend	Παραδοσιακό CMS βασισμένο σε WordPress plugin
Απόδοση	Βελτιστοποίηση frontend, API και caching	Εξαρτάται από theme, plugins, cache και server setup
Επεκτασιμότητα	Ευκολότερη προσθήκη ανεξάρτητων υπηρεσιών	Συχνά βασίζεται σε plugins και WordPress hooks
Συντήρηση	Διακριτά επίπεδα και καθαρότερος διαχωρισμός ευθυνών	Πιθανή αύξηση πολυπλοκότητας λόγω plugins
Ασφάλεια	API-based έλεγχος και μικρότερη έκθεση του frontend	Ωριμο οικοσύστημα αλλά αυξημένη ανάγκη ενημερώσεων
Πειραματικές επεκτάσεις	Recommender, A/B testing, serverless services	Συχνά εξαρτάται από plugins ή custom WordPress ανάπτυξη

Πίνακας 3. Ποιοτική σύγκριση της προτεινόμενης headless πλατφόρμας με WooCommerce

Συνεπώς, η σύγκριση αναδεικνύει ότι το βασικό όφελος της προτεινόμενης αρχιτεκτονικής δεν περιορίζεται μόνο σε μεμονωμένες μετρικές απόδοσης. Το σημαντικότερο κέρδος είναι η δυνατότητα ανεξάρτητης εξέλιξης, επέκτασης και αξιολόγησης των επιμέρους υποσυστημάτων. Η αρχιτεκτονική Strapi και Next.js επιτρέπει στην πλατφόρμα να λειτουργεί ως composable σύστημα, στο οποίο νέες λειτουργίες μπορούν να προστεθούν σταδιακά και να αξιολογηθούν με μετρήσιμα κριτήρια.

6.4. Ενδεικτική serverless και composable αρχιτεκτονική

Ως απόδειξη δυνατότητας μελλοντικής μετάβασης σε composable και MACH αρχιτεκτονική, δημιουργήθηκε φάκελος serverless με ανεξάρτητα function-style services. Τα services αυτά μπορούν να λειτουργήσουν τοπικά μέσω adapter, αλλά η δομή τους είναι συμβατή με πιθανή μελλοντική μεταφορά σε AWS Lambda, Azure Functions ή αντίστοιχα serverless περιβάλλοντα.

Microservice	Ρόλος	Endpoint τοπικού adapter
recommendations	Αυτόνομη υπηρεσία προτάσεων προϊόντων	GET /recommendations
analytics	Αυτόνομη καταγραφή interaction events	POST /track

Checkout	Αυτόνομη ενορχήστρωση checkout με JWT forwarding	POST /checkout
----------	--	----------------

Πίνακας 4 Ενδεικτικά serverless services

Το frontend υποστηρίζει δύο τρόπους λειτουργίας. Αν υπάρχουν οι αντίστοιχες μεταβλητές περιβάλλοντος, χρησιμοποιεί τα serverless endpoints. Αν δεν υπάρχουν, συνεχίζει να χρησιμοποιεί τα υπάρχοντα Strapi endpoints, διατηρώντας πλήρη συμβατότητα με την αρχική υλοποίηση.

Οι βασικές μεταβλητές περιβάλλοντος είναι οι εξής:

NEXT_PUBLIC_RECOMMENDATIONS_URL=<http://127.0.0.1:8787/recommendations>
NEXT_PUBLIC_ANALYTICS_URL=<http://127.0.0.1:8787/track>
NEXT_PUBLIC_CHECKOUT_URL=<http://127.0.0.1:8787/checkout>

Η προσέγγιση αυτή δείχνει ότι επιμέρους λειτουργίες της πλατφόρμας μπορούν να απομονωθούν από τον βασικό κορμό της εφαρμογής και να εξελιχθούν ανεξάρτητα. Με αυτόν τον τρόπο ενισχύεται η επεκτασιμότητα, η συντηρησιμότητα και η δυνατότητα μελλοντικής υιοθέτησης cloud-native υποδομών. (Οι function-style υπηρεσίες recommendations, analytics και checkout, καθώς και ο τοπικός adapter εκτέλεσής τους, παρουσιάζονται στο Παράρτημα Δ.4.)

6.5. Αντιστοίχιση με αρχές MACH

Η πειραματική επέκταση της πλατφόρμας επιτρέπει την ενδεικτική αντιστοίχιση της υλοποίησης με τις βασικές αρχές της MACH αρχιτεκτονικής. Το ακρωνύμιο MACH αναφέρεται σε Microservices, API-first, Cloud-native και Headless αρχιτεκτονική προσέγγιση.

Αρχή MACH	Υλοποίηση στην πλατφόρμα
Microservices	Οι λειτουργίες recommender, analytics και checkout απομονώθηκαν ως ανεξάρτητα services.
API-first	Η επικοινωνία γίνεται μέσω JSON/HTTP endpoints με σαφή contracts.
Cloud-native	Οι handlers έχουν μορφή συμβατή με serverless functions και μπορούν να μεταφερθούν σε cloud runtime.

Headless	To Strapi παραμένει headless CMS, ενώ το Next.js παραμένει ανεξάρτητο frontend/PWA.
----------	---

Πίνακας 5 Αντιστοίχιση της πλατφόρμας με τις αρχές MACH

Η παραπάνω αντιστοίχιση δείχνει ότι, παρότι η βασική υλοποίηση της εργασίας βασίζεται σε Strapi και Next.js, η αρχιτεκτονική μπορεί να επεκταθεί προς πιο σύγχρονες composable κατευθύνσεις. Η απομόνωση επιμέρους λειτουργιών, όπως οι προτάσεις προϊόντων, η καταγραφή analytics και η διαδικασία checkout, επιτρέπει τη σταδιακή μετάβαση από μία κλασική headless υλοποίηση σε ένα πιο ευέλικτο οικοσύστημα ανεξάρτητων υπηρεσιών. (Η πρακτική απομόνωση των επιμέρους υπηρεσιών, στην οποία βασίζεται η αντιστοίχιση με τις αρχές MACH, τεκμηριώνεται από τον κώδικα του Παραρτήματος Δ.4.)

6.6. Υλοποίηση δοκιμαστικού συστήματος πληρωμών με Stripe

Στο πλαίσιο της επέκτασης της πλατφόρμας ηλεκτρονικού εμπορίου, υλοποιήθηκε ενδεικτική διασύνδεση με σύστημα ηλεκτρονικών πληρωμών σε δοκιμαστικό περιβάλλον, αξιοποιώντας το Stripe. Η επιλογή του Stripe έγινε επειδή παρέχει πλήρες test mode, υποστήριξη για ασφαλή διαχείριση πληρωμών μέσω PaymentIntents, καθώς και δυνατότητα ενσωμάτωσης της φόρμας κάρτας απευθείας στο περιβάλλον της εφαρμογής.

Η τελική υλοποίηση βασίστηκε σε ενσωματωμένη ροή πληρωμής (embedded payment flow) και όχι σε ανακατεύθυνση του χρήστη σε εξωτερική σελίδα Stripe Checkout. Συγκεκριμένα, το frontend εμφανίζει πεδίο εισαγωγής κάρτας μέσα στη σελίδα ολοκλήρωσης παραγγελίας, ενώ το Strapi backend αναλαμβάνει τη δημιουργία του PaymentIntent μέσω του μυστικού Stripe API key. Με αυτόν τον τρόπο, το μυστικό κλειδί δεν εκτίθεται ποτέ στον client, ενώ τα στοιχεία της κάρτας δεν αποθηκεύονται ούτε επεξεργάζονται από την εφαρμογή, αλλά διαχειρίζονται από το Stripe.js.

Η ροή πληρωμής υλοποιήθηκε ως εξής: αρχικά ο χρήστης προσθέτει προϊόντα στο καλάθι και μεταβαίνει στη σελίδα checkout. Στη συνέχεια, το frontend καλεί ειδικό endpoint του Strapi για τη δημιουργία PaymentIntent, με βάση τα προϊόντα του καλαθιού και το συνολικό ποσό. Το Strapi επανυπολογίζει server-side την αξία της παραγγελίας, ώστε να μη βασίζεται αποκλειστικά σε τιμές που αποστέλλονται από τον client. Έπειτα, το frontend χρησιμοποιεί το client secret που επιστρέφεται από το Strapi και επιβεβαιώνει την πληρωμή μέσω Stripe.js. Μετά την επιτυχημένη επιβεβαίωση, καλείται δεύτερο endpoint του Strapi, το

οποίο ελέγχει την κατάσταση του `PaymentIntent` και δημιουργεί την παραγγελία με κατάσταση `paid`.

Για την υποστήριξη της λειτουργίας δημιουργήθηκαν δύο custom Strapi endpoints: ένα για τη δημιουργία του `PaymentIntent` και ένα για την οριστικοποίηση της πληρωμένης παραγγελίας. Τα endpoints προστατεύονται με αυθεντικοποίηση και είναι διαθέσιμα μόνο σε συνδεδεμένους χρήστες. Επιπλέον, προστέθηκαν τα αντίστοιχα δικαιώματα πρόσβασης στο `authenticated role` του Strapi, ώστε η ροή πληρωμής να μπορεί να εκτελείται μόνο από νόμιμους χρήστες της πλατφόρμας. (Ο κώδικας δημιουργίας του Stripe `PaymentIntent`, της επιβεβαίωσης της πληρωμής και της οριστικοποίησης της παραγγελίας με κατάσταση `paid` παρουσιάζεται στο Παράρτημα Δ.5.)

Η υλοποίηση λειτουργεί σε `test mode`, χρησιμοποιώντας δοκιμαστικά Stripe API keys και test card numbers, χωρίς πραγματική χρέωση. Για λόγους ασφάλειας, το `publishable key` χρησιμοποιείται μόνο στο frontend, ενώ το `secret key` παραμένει αποκλειστικά στο Strapi backend. Η προσέγγιση αυτή ακολουθεί σύγχρονες πρακτικές ασφαλούς ολοκλήρωσης πληρωμών, καθώς η εφαρμογή δεν αποθηκεύει αριθμούς καρτών, CVC ή άλλα ευαίσθητα στοιχεία πληρωμής.

Η προσθήκη του Stripe ενισχύει τον ρεαλισμό της πλατφόρμας, καθώς μετατρέπει τη διαδικασία checkout από απλή δοκιμαστική καταχώρηση παραγγελίας σε ολοκληρωμένη ροή ηλεκτρονικής πληρωμής. Παράλληλα, αποτελεί χρήσιμο παράδειγμα για το πώς ένα headless CMS, όπως το Strapi, μπορεί να λειτουργήσει ως backend orchestration layer για εξωτερικές υπηρεσίες πληρωμών, διατηρώντας τον διαχωρισμό μεταξύ frontend, backend και τρίτων παρόχων υπηρεσιών. (Η ενσωμάτωση της φόρμας πληρωμής στο frontend και η χρήση του `client secret` μέσω `Stripe.js` παρατίθενται επίσης στο Παράρτημα Δ.5.)

6.7. Οφέλη της προτεινόμενης αρχιτεκτονικής

Η αρχιτεκτονική που υλοποιήθηκε στην παρούσα εργασία προσφέρει σημαντικά πλεονεκτήματα σε σχέση με μία πιο παραδοσιακή, μονολιθική προσέγγιση ηλεκτρονικού εμπορίου. Ο βασικός σχεδιασμός στηρίζεται στον διαχωρισμό του frontend από το backend, με το Strapi να λειτουργεί ως headless CMS και το Next.js να αναλαμβάνει το επίπεδο παρουσίασης και την εμπειρία χρήστη. Ο διαχωρισμός αυτός επιτρέπει την ανεξάρτητη εξέλιξη των δύο επιπέδων και διευκολύνει τη μελλοντική προσθήκη νέων λειτουργιών.

Ένα βασικό όφελος είναι η επεκτασιμότητα. Η πλατφόρμα δεν περιορίζεται σε έναν ενιαίο κώδικα που περιλαμβάνει όλες τις λειτουργίες, αλλά μπορεί να επεκταθεί με ανεξάρτητες υπηρεσίες, όπως ο μηχανισμός προτάσεων προϊόντων, το interaction tracking, το A/B testing, η αξιολόγηση απόδοσης και η ροή πληρωμών. Με αυτόν τον τρόπο, κάθε λειτουργία μπορεί να αναπτυχθεί, να συντηρηθεί ή να αντικατασταθεί χωρίς να απαιτείται πλήρης αναδόμηση του συστήματος.

Επιπλέον, η αρχιτεκτονική ενισχύει την ασφάλεια της εφαρμογής. Χαρακτηριστικό παράδειγμα αποτελεί η ενσωμάτωση του Stripe μέσω PaymentIntents, όπου το frontend χρησιμοποιεί μόνο publishable key, ενώ το secret key παραμένει αποκλειστικά στο Strapi backend. Παράλληλα, η τελική δημιουργία της παραγγελίας πραγματοποιείται server-side, αφού προηγηθεί έλεγχος της κατάστασης της πληρωμής. Έτσι, η εφαρμογή δεν διαχειρίζεται απευθείας ευαίσθητα στοιχεία κάρτας και δεν βασίζεται αποκλειστικά σε δεδομένα που αποστέλλονται από τον client.

Σημαντικό πλεονέκτημα αποτελεί επίσης η δυνατότητα ποσοτικής αξιολόγησης της πλατφόρμας. Μέσω του evaluation toolkit μπορούν να μετρηθούν χρόνοι απόκρισης, Time-to-First-Byte, latency και βασικά security headers. Παράλληλα, μέσω του interaction tracking και του A/B testing μπορούν να συλλεχθούν δεδομένα συμπεριφοράς χρηστών, όπως προβολές προϊόντων, προσθήκες στο καλάθι, clicks σε προτεινόμενα προϊόντα και αγορές. Τα δεδομένα αυτά καθιστούν δυνατή τη συστηματική αξιολόγηση τόσο της τεχνικής απόδοσης όσο και της εμπειρίας χρήστη.

Τέλος, η ενδεικτική υλοποίηση serverless function-style services δείχνει ότι η πλατφόρμα μπορεί να εξελιχθεί προς composable και MACH αρχιτεκτονική. Λειτουργίες όπως recommendations, analytics και checkout μπορούν μελλοντικά να απομονωθούν σε ανεξάρτητα services και να κλιμακωθούν ξεχωριστά. Με αυτόν τον τρόπο, η υλοποίηση δεν αποτελεί απλώς ένα ηλεκτρονικό κατάστημα, αλλά μία βάση για ένα σύγχρονο, ευέλικτο και επεκτάσιμο οικοσύστημα ηλεκτρονικού εμπορίου.

Συνολικά, με την προτεινόμενη αρχιτεκτονική κερδίζουμε διαχωρισμό ευθυνών, καλύτερη συντηρησιμότητα, αυξημένη ασφάλεια, δυνατότητα ανεξάρτητης κλιμάκωσης επιμέρους λειτουργιών και μετρήσιμη αξιολόγηση της πλατφόρμας. Τα χαρακτηριστικά αυτά καθιστούν τη λύση πιο κοντά στις σύγχρονες πρακτικές headless, composable και cloud-native e-commerce συστημάτων.

6.8. Συμπεράσματα κεφαλαίου

Στο παρόν κεφάλαιο παρουσιάστηκαν οι πειραματικές επεκτάσεις της πλατφόρμας ηλεκτρονικού εμπορίου. Η ενσωμάτωση μηχανισμού προτάσεων προϊόντων και A/B testing επιτρέπει τη συλλογή δεδομένων αλληλεπίδρασης και την αξιολόγηση διαφορετικών τρόπων παρουσίασης περιεχομένου. Παράλληλα, το evaluation toolkit παρέχει επαναλήψιμη μετρική αξιολόγηση απόδοσης και ασφάλειας, ενώ η ενδεικτική serverless και composable αρχιτεκτονική αποτυπώνει τη δυνατότητα μελλοντικής επέκτασης της πλατφόρμας σύμφωνα με τις αρχές MACH.

Οι επεκτάσεις αυτές δεν μεταβάλλουν τον βασικό στόχο της εργασίας, αλλά ενισχύουν την τεχνική της πληρότητα, καθώς δείχνουν πώς μία headless PWA εφαρμογή ηλεκτρονικού εμπορίου μπορεί να εξελιχθεί προς πιο προηγμένες, εξατομικευμένες και επεκτάσιμες αρχιτεκτονικές λύσεις.

7. Βιβλιογραφία

Ακολουθούν οι βιβλιογραφικές αναφορές (πηγές) της Εργασίας.

- Aljunid, M. F., Huchaiah, M. D., Hooshmand, M. K., Ali, W. A., Shetty, A. M., & Alzoubah, S. Q. (2025). A collaborative filtering recommender systems: Survey. *Neurocomputing*, 617, 128718. <https://doi.org/10.1016/j.neucom.2024.128718>
- Almahmood, R. J. K., & Tekerek, A. (2022). Issues and solutions in deep learning-enabled recommendation systems within the e-commerce field. *Applied Sciences*, 12(21), 11256. <https://doi.org/10.3390/app122111256>
- Attar, R. W., Almusharraf, A., Alfawaz, A., & Hajli, N. (2022). New trends in e-commerce research: Linking social commerce and sharing commerce: A systematic literature review. *Sustainability*, 14(23), 16024. <https://doi.org/10.3390/su142316024>
- Barker, D. (2016). *Web content management: Systems, features, and best practices*. O'Reilly Media.
- Bass, L., Clements, P., & Kazman, R. (2013). *Software architecture in practice* (3rd ed.). Addison-Wesley.
- Bawack, R. E., Wamba, S. F., Carillo, K. D. A., & Akter, S. (2022). Artificial intelligence in e-commerce: A bibliometric study and literature review. *Electronic Markets*, 32, 297-338. <https://doi.org/10.1007/s12525-022-00537-z>

- Biørn-Hansen, A., Majchrzak, T. A., & Grønli, T.-M. (2017). Progressive Web Apps: The possible web-native unifier for mobile development. In Proceedings of the 13th International Conference on Web Information Systems and Technologies (pp. 344-351). SCITEPRESS.
- Fielding, R. T. (2000). Architectural styles and the design of network-based software architectures [Doctoral dissertation, University of California, Irvine].
- Google. (n.d.). Progressive Web Apps. web.dev. Retrieved September 25, 2026, from <https://web.dev/explore/progressive-web-apps>
- Google. (n.d.). Web Vitals. web.dev. Retrieved September 25, 2026, from <https://web.dev/vitals/>
- Hassan, L. (2023). Major and minor issues affecting the user experience on e-commerce websites. *International Journal of Advanced and Applied Sciences*, 10(10), 208-221. <https://doi.org/10.21833/ijaas.2023.10.023>
- Hassan, S., & Eassa, A. M. (2024). Comparing performance between headless CMS and traditional CMS. *International Journal of Business Information Systems*. <https://doi.org/10.1504/IJBIS.2024.10066810>
- Hidayat, R., Hasbi, H. F., Putri, D. M., Ningrum, S. W., Supriani, N., & Farida, M. (2023). Rancangan user interface untuk meningkatkan performa pada e-commerce menggunakan Progressive Web Apps (PWA). *Technology and Informatics Insight Journal*, 2(2), 70-82. <https://doi.org/10.32639/tij.v2i2.781>
- ISO/IEC. (2023). Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - Product quality model (ISO/IEC 25010:2023). International Organization for Standardization.
- ISO/IEC/IEEE. (2018). Systems and software engineering - Life cycle processes - Requirements engineering (ISO/IEC/IEEE 29148:2018). International Organization for Standardization.
- Islek, I., & Oguducu, S. G. (2022). A hierarchical recommendation system for e-commerce using online user reviews. *Electronic Commerce Research and Applications*, 52, 101131. <https://doi.org/10.1016/j.elerap.2022.101131>
- Laudon, K. C., & Traver, C. G. (2020). *E-commerce 2020-2021: Business, technology and society* (16th ed.). Pearson.
- MACH Alliance. (n.d.). MACH principles. Retrieved September 25, 2026, from <https://machalliance.org/mach-principles>
- Malavolta, I., Procaccianti, G., Noorland, P., & Vukmirović, P. (2017). Assessing the impact of service workers on the energy efficiency of Progressive Web Apps. In 2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems. IEEE.
- Necula, S.-C., & Păvăloaia, V.-D. (2023). AI-driven recommendations: A systematic review of the state of the art in e-commerce. *Applied Sciences*, 13(9), 5531. <https://doi.org/10.3390/app13095531>

- Newman, S. (2021). *Building microservices: Designing fine-grained systems* (2nd ed.). O'Reilly Media.
- Next.js. (n.d.). App Router documentation. Retrieved September 25, 2026, from <https://nextjs.org/docs/app>
- Next.js. (n.d.). Progressive Web Applications with Next.js. Retrieved September 25, 2026, from <https://nextjs.org/docs/app/guides/progressive-web-apps>
- OWASP Foundation. (2023). OWASP API Security Top 10 2023. <https://owasp.org/www-project-api-security/>
- Pautasso, C., Zimmermann, O., & Leymann, F. (2008). RESTful web services vs. Big web services: Making the right architectural decision. In *Proceedings of the 17th International Conference on World Wide Web* (pp. 805-814). ACM.
- PCI Security Standards Council. (n.d.). PCI DSS. Retrieved September 25, 2026, from <https://www.pcisecuritystandards.org/standards/pci-dss/>
- Pressman, R. S., & Maxim, B. R. (2019). *Software engineering: A practitioner's approach* (9th ed.). McGraw-Hill.
- Putro, D. P., Suprianto, J., & Suryani, P. E. (2025). Dampak Progressive Web Application (PWA) terhadap peningkatan engagement dan dwell time pada aplikasi e-commerce. *IDEALIS*, 8(1), 84-91. <https://doi.org/10.36080/idealism.v8i1.3369>
- Shopify. (n.d.). Hydrogen: Shopify's headless commerce framework. Retrieved September 25, 2026, from <https://hydrogen.shopify.dev/>
- Sommerville, I. (2016). *Software engineering* (10th ed.). Pearson.
- Strapi. (n.d.). Strapi 5 documentation. Retrieved September 25, 2026, from <https://docs.strapi.io/>
- Stripe. (n.d.). The Payment Intents API. Retrieved September 25, 2026, from <https://docs.stripe.com/payments/payment-intents>
- Trinh, T., Nguyen, V.-H., Nguyen, N., & Nguyen, D.-N. (2025). Product collaborative filtering based recommendation systems for large-scale e-commerce. *International Journal of Information Management Data Insights*, 5(1), 100322. <https://doi.org/10.1016/j.jjime.2025.100322>
- Valencia-Arias, A., Uribe-Bedoya, H., González-Ruiz, J. D., Sánchez Santos, G., Chapañan Ramírez, E., & Martínez Rojas, E. (2024). Artificial intelligence and recommender systems in e-commerce: Trends and research agenda. *Intelligent Systems with Applications*, 24, 200435. <https://doi.org/10.1016/j.iswa.2024.200435>
- Vrhovac, V., Vasić, S., Milisavljević, S., Dudić, B., Štarchoň, P., & Žižakov, M. (2023). Measuring e-commerce user experience in the last-mile delivery. *Mathematics*, 11(6), 1482. <https://doi.org/10.3390/math11061482>
- WooCommerce. (n.d.). WooCommerce REST API documentation. Retrieved September 25, 2026, from <https://woocommerce.com/document/woocommerce-rest-api/>
- Zimmermann, R., & Auinger, A. (2023). Developing a conversion rate optimization framework for digital retailers: Case study. *Journal of Marketing Analytics*, 11, 233-243. <https://doi.org/10.1057/s41270-022-00161-y>

Στα παρόν παραρτήματα παρατίθενται αντιπροσωπευτικά αποσπάσματα του πηγαίου κώδικα που χρησιμοποιήθηκαν για την υλοποίηση της εφαρμογής. Τα αποσπάσματα καλύπτουν τόσο το back-end μέρος του συστήματος, το οποίο αναπτύχθηκε με Strapi, όσο και το front-end μέρος, το οποίο αναπτύχθηκε με Next.js. Η παρουσίαση του κώδικα αποσκοπεί στην τεκμηρίωση της υλοποίησης και στη διευκόλυνση της αναπαραγωγής της προτεινόμενης λύσης.

8. Παράρτημα Α: Βασικός Κώδικας Υλοποίησης

A.1 Back-End – Schema παραγγελίας στο Strapi

Αρχείο:src/api/order/content-types/order/schema.json

Σκοπός: Ορισμός της οντότητας παραγγελίας, των πεδίων της και της σχέσης με τον χρήστη.

```
1. {
2.   "kind": "collectionType",
3.   "collectionName": "orders",
4.   "info": {
5.     "singularName": "order",
6.     "pluralName": "orders",
7.     "displayName": "order"
8.   },
9.   "options": {
10.    "draftAndPublish": true
11.  },
12.  "pluginOptions": {},
13.  "attributes": {
14.    "items": {
15.      "type": "json"
16.    },
17.    "total": {
18.      "type": "integer"
19.    },
20.    "user": {
21.      "type": "relation",
22.      "relation": "manyToOne",
23.      "target": "plugin::users-permissions.user",
24.      "inversedBy": "orders"
25.    },
26.    "status": {
27.      "type": "enumeration",
28.      "enum": ["pending", "paid", "cancelled"],
29.      "default": "pending"
30.    }
31.  }
32. }
33.
```

A.2 Back-End – Επέκταση μοντέλου χρήστη

Αρχείο: src/extensions/users-permissions/content-types/user/schema.json

Σκοπός: Αντίστροφη σχέση χρήστη-παραγγελιών.

```
1. {
2.   "collectionName": "up_users",
3.   "info": {
4.     "singularName": "user",
5.     "pluralName": "users",
6.     "displayName": "User"
7.   },
8.   "options": {
9.     "draftAndPublish": false
10.  },
11.  "attributes": {
12.    "username": {
13.      "type": "string",
14.      "minLength": 3,
15.      "unique": true,
16.      "configurable": false,
```

```

17.     "required": true
18. },
19. "email": {
20.     "type": "email",
21.     "minLength": 6,
22.     "configurable": false,
23.     "required": true
24. },
25. "provider": {
26.     "type": "string",
27.     "configurable": false
28. },
29. "password": {
30.     "type": "password",
31.     "minLength": 6,
32.     "configurable": false,
33.     "private": true,
34.     "searchable": false
35. },
36. "resetPasswordToken": {
37.     "type": "string",
38.     "configurable": false,
39.     "private": true,
40.     "searchable": false
41. },
42. "confirmationToken": {
43.     "type": "string",
44.     "configurable": false,
45.     "private": true,
46.     "searchable": false
47. },
48. "confirmed": {
49.     "type": "boolean",
50.     "default": false,
51.     "configurable": false
52. },
53. "blocked": {
54.     "type": "boolean",
55.     "default": false,
56.     "configurable": false
57. },
58. "role": {
59.     "type": "relation",
60.     "relation": "manyToOne",
61.     "target": "plugin::users-permissions.role",
62.     "inversedBy": "users",
63.     "configurable": false
64. },
65. "orders": {
66.     "type": "relation",
67.     "relation": "oneToMany",
68.     "target": "api::order.order",
69.     "mappedBy": "user"
70. }
71. }
72. }
73.

```

A.3 Back-End – Controller παραγγελιών

Αρχείο: src/api/order/controllers/order.ts

Σκοπός: Δημιουργία παραγγελίας και επιστροφή μόνο των παραγγελιών του συνδεδεμένου χρήστη.

```
1. import { factories } from '@strapi/strapi'
2.
3. export default factories.createCoreController('api::order.order', ({ strapi }) => ({
4.   async create(ctx) {
5.     const user = ctx.state.user
6.
7.     if (!user) {
8.       return ctx.unauthorized('You must be logged in.')
9.     }
10.
11.     const { items, total } = ctx.request.body.data || {}
12.
13.     if (!items || !Array.isArray(items) || typeof total !== 'number') {
14.       return ctx.badRequest('Invalid order payload.')
15.     }
16.
17.     const order = await strapi.entityService.create('api::order.order', {
18.       data: {
19.         items,
20.         total,
21.         user: user.id,
22.         status: 'pending',
23.       } as any,
24.       populate: ['user'] as any,
25.     } as any)
26.
27.     return this.transformResponse(order)
28.   },
29.
30.   async myOrders(ctx) {
31.     const user = ctx.state.user
32.
33.     if (!user) {
34.       return ctx.unauthorized('You must be logged in.')
35.     }
36.
37.     const orders = await strapi.entityService.findMany('api::order.order', {
38.       filters: {
39.         user: {
40.           id: {
41.             $eq: user.id,
42.           },
43.         },
44.       } as any,
45.       sort: { createdAt: 'desc' },
46.       populate: ['user'] as any,
47.     } as any)
48.
49.     return this.transformResponse(orders)
50.   },
51. })})
```

A.4 Back-End – Custom route προστατευμένων παραγγελιών

Αρχείο: src/api/order/routes/custom-order.ts

Σκοπός: Ορισμός προστατευμένου endpoint για το ιστορικό παραγγελιών.

```
1. export default {
2.   routes: [
```

```

3.    {
4.      method: 'GET',
5.      path: '/orders/my',
6.      handler: 'order.myOrders',
7.      config: {
8.        auth: {},
9.      },
10.    },
11.  ],
12. }

```

A.5 Back-End – Βασικό router παραγγελιών

Αρχείο:src/api/order/routes/order.ts

Σκοπός: Ενεργοποίηση default router του content type.

```

1. import { factories } from '@strapi/strapi'
2.
3. export default factories.createCoreRouter('api::order.order')

```

9. Παράρτημα Β: «Front-End (Next.js)»

Αρχείο:lib/strapi.ts

Σκοπός: Βοηθητικές συναρτήσεις για κλήσεις προς το Strapi API.

```

1. import qs from 'qs'
2. import type { Product } from './types'
3.
4. const STRAPI_URL = process.env.NEXT_PUBLIC_STRAPI_URL!
5.
6. export function getStrapiMedia(url?: string | null) {
7.   if (!url) return null
8.   if (url.startsWith('http')) return url
9.   return `${STRAPI_URL}${url}`
10. }
11.
12. export async function fetchStrapi<T>({
13.   path: string,
14.   query?: Record<string, unknown>,
15.   options: RequestInit = {}
16. }): Promise<T> {
17.   const queryString = query
18.     ? qs.stringify(query, { encodeValuesOnly: true })
19.     : ''
20.
21.   const url = `${STRAPI_URL}${path}${queryString ? `?${queryString}` : ''}`
22.
23.   const res = await fetch(url, {
24.     ...options,
25.     headers: {
26.       'Content-Type': 'application/json',
27.       ...(options.headers || {}),
28.     },
29.     next: { revalidate: 60 },
30.   })
31.

```

```

32.   if (!res.ok) {
33.     let details = ''
34.     try {
35.       const err = await res.json()
36.       details = err?.error?.message
37.       ? ` - ${err.error.message}`
38.       : ` - ${JSON.stringify(err)}`
39.     } catch {
40.       details = ''
41.     }
42.
43.     throw new Error(
44.       `Strapi request failed: ${res.status} ${res.statusText}${details}`
45.     )
46.   }
47.
48.   return res.json()
49. }
50.
51. type StrapiListResponse<T> = {
52.   data: T[]
53.   meta: {
54.     pagination?: {
55.       page: number
56.       pageSize: number
57.       pageCount: number
58.       total: number
59.     }
60.   }
61. }
62.
63. export async function getProducts() {
64.   return fetchStrapi<StrapiListResponse<Product>>('/api/products', {
65.     populate: '*',
66.     sort: ['createdAt:desc'],
67.     pagination: {
68.       page: 1,
69.       pageSize: 24,
70.     },
71.   })
72. }
73.
74. export async function getProductBySlug(slug: string) {
75.   return fetchStrapi<StrapiListResponse<Product>>('/api/products', {
76.     filters: {
77.       slug: {
78.         $eq: slug,
79.       },
80.     },
81.     populate: '*',
82.   })
83. }
84.

```

B.2 Front-End – Authentication helper

Αρχείο: lib/auth.ts

Σκοπός: Login, register, διαχείριση token και χρήση.

```

1. const STRAPI_URL = process.env.NEXT_PUBLIC_STRAPI_URL!
2.
3. const TOKEN_KEY = 'token'
4. const USER_KEY = 'user'
5.
6. export async function login(identifier: string, password: string) {

```

```

7.   const res = await fetch(`${STRAPI_URL}/api/auth/local`, {
8.     method: 'POST',
9.     headers: {
10.      'Content-Type': 'application/json',
11.    },
12.    body: JSON.stringify({
13.      identifier,
14.      password,
15.    }),
16.  })
17.
18.   const data = await res.json()
19.
20.   if (!res.ok) {
21.     throw new Error(data.error?.message || 'Login failed')
22.   }
23.
24.   return data
25. }
26.
27. export async function register(
28.   username: string,
29.   email: string,
30.   password: string
31. ) {
32.   const res = await fetch(`${STRAPI_URL}/api/auth/local/register`, {
33.     method: 'POST',
34.     headers: {
35.      'Content-Type': 'application/json',
36.    },
37.    body: JSON.stringify({
38.      username,
39.      email,
40.      password,
41.    }),
42.  })
43.
44.   const data = await res.json()
45.
46.   if (!res.ok) {
47.     throw new Error(data.error?.message || 'Register failed')
48.   }
49.
50.   return data
51. }
52.
53. export function getToken() {
54.   if (typeof window === 'undefined') return null
55.   return localStorage.getItem(TOKEN_KEY)
56. }
57.
58. export function setToken(token: string) {
59.   if (typeof window === 'undefined') return
60.   localStorage.setItem(TOKEN_KEY, token)
61. }
62.
63. export function removeToken() {
64.   if (typeof window === 'undefined') return
65.   localStorage.removeItem(TOKEN_KEY)
66. }
67.
68. export function getStoredUser() {
69.   if (typeof window === 'undefined') return null
70.
71.   try {
72.     const raw = localStorage.getItem(USER_KEY)
73.     return raw ? JSON.parse(raw) : null
74.   } catch {
75.     return null
76.   }

```

```

77. }
78.
79. export function setStoredUser(user: unknown) {
80.   if (typeof window === 'undefined') return
81.   localStorage.setItem(USER_KEY, JSON.stringify(user))
82. }
83.
84. export function logout() {
85.   if (typeof window === 'undefined') return
86.   localStorage.removeItem(TOKEN_KEY)
87.   localStorage.removeItem(USER_KEY)
88. }
89.

```

B.3 Front-End – Διαχείριση καλαθιού

Αρχείο: lib/cart.ts

Σκοπός: Τοπική αποθήκευση και διαχείριση καλαθιού αγορών.

```

1. export type CartItem = {
2.   id: number
3.   slug: string
4.   title: string
5.   price: number
6.   quantity: number
7.   image?: string | null
8. }
9.
10. const CART_KEY = 'shop_cart'
11.
12. export function getCart(): CartItem[] {
13.   if (typeof window === 'undefined') return []
14.
15.   try {
16.     const raw = localStorage.getItem(CART_KEY)
17.     return raw ? JSON.parse(raw) : []
18.   } catch {
19.     return []
20.   }
21. }
22.
23. export function saveCart(items: CartItem[]) {
24.   if (typeof window === 'undefined') return
25.   localStorage.setItem(CART_KEY, JSON.stringify(items))
26. }
27.
28. export function addToCart(item: CartItem) {
29.   const cart = getCart()
30.   const existing = cart.find((p) => p.id === item.id)
31.
32.   if (existing) {
33.     existing.quantity += item.quantity
34.   } else {
35.     cart.push(item)
36.   }
37.
38.   saveCart(cart)
39. }
40.
41. export function updateCartQuantity(id: number, quantity: number) {
42.   const cart = getCart().map((item) =>
43.     item.id === id ? { ...item, quantity } : item
44.   )
45.
46.   saveCart(cart.filter((item) => item.quantity > 0))
47. }
48.
49. export function removeFromCart(id: number) {

```

```

50.   const cart = getCart().filter((item) => item.id !== id)
51.   saveCart(cart)
52. }
53.
54. export function clearCart() {
55.   if (typeof window === 'undefined') return
56.   localStorage.removeItem(CART_KEY)
57. }
58.

```

B.4 Front-End – Σελίδα λίστας προϊόντων

Αρχείο: app/products/page.tsx

```

1. import Link from 'next/link'
2. import { getProducts, getStrapiMedia } from '@lib/strapi'
3.
4. export default async function ProductsPage() {
5.   const data = await getProducts()
6.   const products = data.data
7.
8.   return (
9.     <main className="max-w-6xl mx-auto px-4 py-10">
10.      <h1 className="text-3xl font-bold mb-6">Products</h1>
11.
12.      <div className="grid gap-6 sm:grid-cols-2 lg:grid-cols-3">
13.        {products.map((product: any) => {
14.          const imageUrl = getStrapiMedia(product.image?.url)
15.          const href = product.slug ? `/products/${product.slug}` : '#'
16.
17.          return (
18.            <Link
19.              key={product.id}
20.              href={href}
21.              className="block border rounded-2xl p-4 shadow-sm hover:shadow-md
transition"
22.            >
23.              {imageUrl ? (
24.                <img
25.                  src={imageUrl}
26.                  alt={product.title}
27.                  className="w-full h-52 object-cover rounded-xl mb-3"
28.                />
29.              ) : (
30.                <div className="w-full h-52 rounded-xl bg-gray-100 mb-3" />
31.              )}
32.
33.              <h2 className="text-lg font-semibold">{product.title}</h2>
34.              <p className="mt-2 font-bold">{product.price} €</p>
35.            </Link>
36.          )
37.        })}
38.      </div>
39.    </main>
40.  )
41. }

```

B.5 Front-End – Δυναμική σελίδα προϊόντος

Αρχείο: app/products/[slug]/page.tsx

```

1. import { notFound } from 'next/navigation'
2. import AddToCartButton from '@components/AddToCartButton'
3. import { getProductBySlug, getStrapiMedia } from '@lib/strapi'
4.
5. type Props = {
6.   params: Promise<{ slug: string }>

```

```

7. }
8.
9. export default async function ProductPage({ params }: Props) {
10.   const { slug } = await params
11.   const response = await getProductBySlug(slug)
12.   const product = response.data[0]
13.
14.   if (!product) notFound()
15.
16.   const imageUrl = getStrapiMedia(product.image?.url)
17.
18.   return (
19.     <main className="max-w-5xl mx-auto px-4 py-10">
20.       <div className="grid gap-8 md:grid-cols-2">
21.         <div>
22.           {imageUrl ? (
23.             <img
24.               src={imageUrl}
25.               alt={product.image?.alternativeText || product.title}
26.               className="w-full rounded-2xl object-cover"
27.             />
28.           ) : (
29.             <div className="w-full aspect-square bg-gray-100 rounded-2xl" />
30.           )}
31.         </div>
32.
33.         <div>
34.           <p className="text-sm text-gray-500 mb-2">
35.             {product.categories?.[0]?.name || 'Χωρίς κατηγορία'}
36.           </p>
37.
38.           <h1 className="text-3xl font-bold mb-4">{product.title}</h1>
39.
40.           <p className="text-2xl font-semibold mb-6">
41.             {product.price.toFixed(2)} €
42.           </p>
43.
44.           <div className="mb-6">
45.             {product.inStock ? (
46.               <span className="text-green-700">Σε απόθεμα</span>
47.             ) : (
48.               <span className="text-red-700">Εκτός αποθέματος</span>
49.             )}
50.           </div>
51.
52.           <AddToCartButton
53.             id={product.id}
54.             slug={product.slug}
55.             title={product.title}
56.             price={product.price}
57.             image={imageUrl}
58.           />
59.         </div>
60.       </div>
61.     </main>
62.   )
63. }

```

B.6 Front-End – Προσθήκη προϊόντος στο καλάθι

Αρχείο: components/AddToCartButton.tsx

```

1. 'use client'
2.
3. import { addToCart } from '@/lib/cart'
4.
5. type Props = {
6.   id: number
7.   slug: string
8.   title: string

```

```

9.   price: number
10.  image?: string | null
11. }
12.
13. export default function AddToCartButton(props: Props) {
14.   return (
15.     <button
16.       onClick={() => {
17.         addToCart({
18.           id: props.id,
19.           slug: props.slug,
20.           title: props.title,
21.           price: props.price,
22.           quantity: 1,
23.           image: props.image || null,
24.         })
25.         alert('Το προϊόν προστέθηκε στο καλάθι')
26.       }}
27.       className="rounded-xl bg-black text-white px-5 py-3"
28.     >
29.       Προσθήκη στο καλάθι
30.     </button>
31.   )
32. }
33.

```

B.7 Front-End – Checkout

Αρχείο: app/checkout/page.tsx

```

1. 'use client'
2.
3. import { useEffect, useState } from 'react'
4. import Link from 'next/link'
5. import { getCart, clearCart, type CartItem } from '@lib/cart'
6. import { getToken } from '@lib/auth'
7.
8. export default function CheckoutPage() {
9.   const [items, setItems] = useState<CartItem[]>([])
10.  const [loading, setLoading] = useState(false)
11.  const [success, setSuccess] = useState(false)
12.
13.  useEffect(() => {
14.    const cart = getCart()
15.    setItems(cart)
16.  }, [])
17.
18.  const total = items.reduce(
19.    (sum, item) => sum + item.price * item.quantity,
20.    0
21.  )
22.
23.  async function handleCheckout() {
24.    const token = getToken()
25.
26.    if (!token) {
27.      alert('Πρέπει να κάνεις login')
28.      return
29.    }
30.
31.    setLoading(true)
32.
33.    try {
34.      const res = await fetch(
35.        `${process.env.NEXT_PUBLIC_STRAPI_URL}/api/orders`,
36.        {
37.          method: 'POST',
38.          headers: {
39.            'Content-Type': 'application/json',
40.            Authorization: `Bearer ${token}`,

```

```

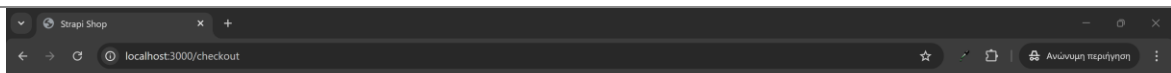
41.         },
42.         body: JSON.stringify({
43.             data: {
44.                 items,
45.                 total,
46.             },
47.         })),
48.     }
49. )
50.
51. if (!res.ok) {
52.     throw new Error('Order failed')
53. }
54.
55. clearCart()
56. setItems([])
57. setSuccess(true)
58. } catch (err) {
59.     console.error(err)
60.     alert('Κάτι πήγε λάθος')
61. } finally {
62.     setLoading(false)
63. }
64. }
65.
66. if (items.length === 0 && !success) {
67.     return (
68.         <main className="max-w-4xl mx-auto px-4 py-10">
69.             <h1 className="text-2xl font-bold mb-4">Καλάθι</h1>
70.             <p>Το καλάθι είναι άδειο.</p>
71.             <Link href="/products" className="text-blue-500">
72.                 Πήγαινε στα προϊόντα
73.             </Link>
74.         </main>
75.     )
76. }
77.
78. if (success) {
79.     return (
80.         <main className="max-w-4xl mx-auto px-4 py-10">
81.             <h1 className="text-2xl font-bold mb-4">Ολοκληρώθηκε 🎉</h1>
82.             <p>Η παραγγελία καταχωρήθηκε!</p>
83.             <Link href="/" className="text-blue-500">
84.                 Επιστροφή στην αρχική
85.             </Link>
86.         </main>
87.     )
88. }
89.
90. return (
91.     <main className="max-w-4xl mx-auto px-4 py-10">
92.         <h1 className="text-2xl font-bold mb-6">Checkout</h1>
93.
94.         <div className="space-y-4">
95.             {items.map((item) => (
96.                 <div
97.                     key={item.id}
98.                     className="flex justify-between border p-4 rounded-xl"
99.                 >
100.                    <div>
101.                        <h2 className="font-semibold">{item.title}</h2>
102.                        <p>Ποσότητα: {item.quantity}</p>
103.                    </div>
104.                    <p>{item.price * item.quantity} €</p>
105.                </div>
106.            ))}
107.        </div>
108.
109.        <div className="mt-6 text-right">

```

```

110.     <p className="text-xl font-bold">Σύνολο: {total} €</p>
111.
112.     <button
113.       onClick={handleCheckout}
114.       disabled={loading}
115.       className="mt-4 bg-black text-white px-6 py-3 rounded-xl"
116.     >
117.       {loading ? '...' : 'Ολοκλήρωση Παραγγελίας'}
118.     </button>
119.   </div>
120. </main>
121. )
122. }

```



Αρχική Προϊόντα Καλάθι Σύνδεση Εγγραφή Παραγγελίες

Checkout

iphone 17 pro
Ποσότητα: 1

1349 €

MacBook Pro
Ποσότητα: 1

1999 €

Σύνολο: 3348 €

Ολοκλήρωση Παραγγελίας

B.8 Front-End – Σελίδα ιστορικού παραγγελιών

Αρχείο: app/orders/page.tsx

```

1. 'use client'
2.
3. import Link from 'next/link'
4. import { useEffect, useState } from 'react'
5. import { getToken } from '@lib/auth'
6.
7. type OrderItem = {
8.   id?: number
9.   title?: string
10.  quantity: number
11.  price: number
12. }
13.
14. type Order = {
15.   id: number
16.   items: OrderItem[]
17.   total: number
18.   createdAt?: string
19.   status?: 'pending' | 'paid' | 'cancelled'
20. }
21.
22. export default function OrdersPage() {
23.   const [orders, setOrders] = useState<Order[]>([])
24.   const [loading, setLoading] = useState(true)
25.   const [error, setError] = useState('')
26.
27.   useEffect(() => {
28.     async function loadOrders() {
29.       try {
30.         const token = getToken()
31.
32.         if (!token) {
33.           setError('Πρέπει να κάνεις σύνδεση για να δεις τις παραγγελίες σου.')
34.           setLoading(false)
35.           return
36.         }
37.

```

```

38.     const res = await fetch(
39.       `${process.env.NEXT_PUBLIC_STRAPI_URL}/api/orders/my`,
40.       {
41.         method: 'GET',
42.         headers: {
43.           Authorization: `Bearer ${token}`,
44.         },
45.         cache: 'no-store',
46.       }
47.     )
48.
49.     const data = await res.json()
50.
51.     if (!res.ok) {
52.       throw new Error(data.error?.message || 'Failed to fetch orders')
53.     }
54.
55.     setOrders(data.data || [])
56.   } catch (err: any) {
57.     setError(err.message || 'Κάτι πήγε λάθος')
58.   } finally {
59.     setLoading(false)
60.   }
61. }
62.
63.   loadOrders()
64. }, [])
65.
66. if (loading) {
67.   return (
68.     <main className="max-w-5xl mx-auto px-4 py-10">
69.       <h1 className="text-3xl font-bold mb-6">Οι παραγγελίες μου</h1>
70.       <p>Φόρτωση...</p>
71.     </main>
72.   )
73. }
74.
75. if (error) {
76.   return (
77.     <main className="max-w-5xl mx-auto px-4 py-10">
78.       <h1 className="text-3xl font-bold mb-6">Οι παραγγελίες μου</h1>
79.       <p className="text-red-600 mb-4">{error}</p>
80.       <Link href="/login" className="underline">
81.         Σύνδεση
82.       </Link>
83.     </main>
84.   )
85. }
86.
87. if (orders.length === 0) {
88.   return (
89.     <main className="max-w-5xl mx-auto px-4 py-10">
90.       <h1 className="text-3xl font-bold mb-6">Οι παραγγελίες μου</h1>
91.       <p>Δεν υπάρχουν παραγγελίες ακόμη.</p>
92.       <Link href="/products" className="underline">
93.         Δες προϊόντα
94.       </Link>
95.     </main>
96.   )
97. }
98.
99. return (
100.  <main className="max-w-5xl mx-auto px-4 py-10">
101.    <h1 className="text-3xl font-bold mb-6">Οι παραγγελίες μου</h1>
102.
103.    <div className="space-y-6">
104.      {orders.map((order) => {
105.        const status = order.status || 'pending'
106.
107.        return (

```

```

108. <div key={order.id} className="border rounded-2xl p-5 shadow-sm">
109.   <div className="flex justify-between items-center mb-4">
110.     <h2 className="text-lg font-semibold">
111.       Παράγγελία #{order.id}
112.     </h2>
113.     <span className="font-bold text-lg">
114.       {order.total} €
115.     </span>
116.   </div>
117.
118.   {order.createdAt && (
119.     <p className="text-sm text-gray-500 mb-2">
120.       {new Date(order.createdAt).toLocaleString('el-GR')}
121.     </p>
122.   )}
123.
124.   <div className="mb-4">
125.     <span
126.       className={`inline-block rounded-full px-3 py-1 text-xs font-semibold
127.         ${
128.           status === 'paid'
129.             ? 'bg-green-100 text-green-700'
130.             : status === 'pending'
131.               ? 'bg-yellow-100 text-yellow-700'
132.               : 'bg-red-100 text-red-700'
133.         }`>
134.       >
135.         {status === 'paid'
136.           ? 'Πληρωμένη'
137.           : status === 'pending'
138.             ? 'Σε αναμονή'
139.             : 'Ακυρωμένη'}
140.     </span>
141.   </div>
142.
143.   <div className="space-y-2">
144.     {order.items?.map((item, index) => (
145.       <div
146.         key={index}
147.         className="flex justify-between border-b pb-2 text-sm"
148.       >
149.         <span>
150.           {item.title || 'Προϊόν'} × {item.quantity}
151.         </span>
152.         <span>
153.           {item.price * item.quantity} €
154.         </span>
155.       </div>
156.     ))}
157.   </div>
158. )
159.   )}
160. </div>
161. </main>
162. )
163. }
164.

```

10. Παράρτημα Γ – Κώδικας Πιστοποίησης Χρηστών

Γ.1 Front-End – Σελίδα σύνδεσης

Αρχείο: app/login/page.tsx

Σκοπός: Σύνδεση χρήστη μέσω Strapi και αποθήκευση JWT token.

```
1. 'use client'
2.
3. import { useState } from 'react'
4. import { login, setToken, setStoredUser } from '@lib/auth'
5. import { useRouter } from 'next/navigation'
6.
7. export default function LoginPage() {
8.   const router = useRouter()
9.
10.  const [identifier, setIdentifier] = useState('')
11.  const [password, setPassword] = useState('')
12.  const [error, setError] = useState('')
13.  const [loading, setLoading] = useState(false)
14.
15.  async function handleLogin(e: React.FormEvent) {
16.    e.preventDefault()
17.    setLoading(true)
18.    setError('')
19.
20.    try {
21.      const data = await login(identifier, password)
22.
23.      setToken(data.jwt)
24.      setStoredUser(data.user)
25.
26.      router.push('/')
27.    } catch (err: any) {
28.      setError(err.message)
29.    } finally {
30.      setLoading(false)
31.    }
32.  }
33.
34.  return (
35.    <main className="max-w-md mx-auto px-4 py-10">
36.      <h1 className="text-2xl font-bold mb-6">Σύνδεση</h1>
37.
38.      <form onSubmit={handleLogin} className="space-y-4">
39.        <input
40.          type="text"
41.          placeholder="Email ή Username"
42.          value={identifier}
43.          onChange={(e) => setIdentifier(e.target.value)}
44.          className="w-full border p-3 rounded-xl"
45.        />
46.
47.        <input
48.          type="password"
49.          placeholder="Κωδικός"
50.          value={password}
51.          onChange={(e) => setPassword(e.target.value)}
52.          className="w-full border p-3 rounded-xl"
53.        />
54.
55.        {error && <p className="text-red-600">{error}</p>}
56.
57.        <button
58.          type="submit"
59.          disabled={loading}
60.          className="w-full bg-black text-white py-3 rounded-xl">
```

```

61.         >
62.         {loading ? '...' : 'Σύνδεση'}
63.     </button>
64. </form>
65. </main>
66. )
67. }
68.

```

Γ.2 Front-End – Σελίδα εγγραφής

Αρχείο: app/register/page.tsx

Σκοπός: Δημιουργία νέου λογαριασμού χρήστη και αποθήκευση JWT token.

```

1. 'use client'
2.
3. import { useState } from 'react'
4. import { register, setToken, setStoredUser } from '@lib/auth'
5. import { useRouter } from 'next/navigation'
6.
7. export default function RegisterPage() {
8.   const router = useRouter()
9.
10.  const [username, setUsername] = useState('')
11.  const [email, setEmail] = useState('')
12.  const [password, setPassword] = useState('')
13.  const [error, setError] = useState('')
14.  const [loading, setLoading] = useState(false)
15.
16.  async function handleRegister(e: React.FormEvent) {
17.    e.preventDefault()
18.    setLoading(true)
19.    setError('')
20.
21.    try {
22.      const data = await register(username, email, password)
23.
24.      setToken(data.jwt)
25.      setStoredUser(data.user)
26.
27.      router.push('/')
28.    } catch (err: any) {
29.      setError(err.message)
30.    } finally {
31.      setLoading(false)
32.    }
33.  }
34.
35.  return (
36.    <main className="max-w-md mx-auto px-4 py-10">
37.      <h1 className="text-2xl font-bold mb-6">Εγγραφή</h1>
38.
39.      <form onSubmit={handleRegister} className="space-y-4">
40.        <input
41.          type="text"
42.          placeholder="Username"
43.          value={username}
44.          onChange={(e) => setUsername(e.target.value)}
45.          className="w-full border p-3 rounded-xl"
46.        />
47.
48.        <input
49.          type="email"
50.          placeholder="Email"
51.          value={email}
52.          onChange={(e) => setEmail(e.target.value)}
53.          className="w-full border p-3 rounded-xl"
54.        />
55.
56.        <input

```

```

57.         type="password"
58.         placeholder="Κωδικός"
59.         value={password}
60.         onChange={(e) => setPassword(e.target.value)}
61.         className="w-full border p-3 rounded-xl"
62.     />
63.
64.     {error && <p className="text-red-600">{error}</p>}
65.
66.     <button
67.         type="submit"
68.         disabled={loading}
69.         className="w-full bg-black text-white py-3 rounded-xl"
70.     >
71.         {loading ? '...' : 'Εγγραφή'}
72.     </button>
73. </form>
74. </main>
75. )
76. }
77.

```

11. Παράρτημα Δ: Πειραματικές Επεκτάσεις και Evaluation Toolkit

Δ1.Κώδικας AI Recommender

Strapi route για recommendations

```

1. // src/api/recommendation/routes/recommendation.ts
2.
3. export default {
4.   routes: [
5.     {
6.       method: 'GET',
7.       path: '/recommendations',
8.       handler: 'recommendation.find',
9.       config: {
10.        auth: false,
11.      },
12.    },
13.  ],
14. }
15.
16.

```

Βάρη αλληλεπιδράσεων χρήστη

```

1. // src/api/recommendation/controllers/recommendation.ts
2.
3. const WEIGHTS: Record<string, number> = {
4.   purchase: 5,
5.   add_to_cart: 3,
6.   recommendation_click: 2,
7.   view: 1,
8.   recommendation_impression: 0.25,
9. }
10.

```

Collaborative filtering με βάση κοινά sessions

```

1. // src/api/recommendation/controllers/recommendation.ts
2.
3. async function collaborativeCandidates(strapi: any, product: any, limit: number) {

```

```

4.   const relatedSessions = await strapi.entityService.findMany(
5.     'api::interaction.interaction',
6.     {
7.       filters: {
8.         product: { id: product.id },
9.         eventType: {
10.           $in: ['view', 'add_to_cart', 'purchase', 'recommendation_click'],
11.         },
12.       },
13.       fields: ['sessionId'],
14.       limit: 1000,
15.     }
16.   )
17.
18.   const sessions = Array.from(
19.     new Set(
20.       relatedSessions
21.         .map((interaction: any) => interaction.sessionId)
22.         .filter(Boolean)
23.     )
24.   )
25.
26.   if (sessions.length === 0) return []
27.
28.   const interactions = await strapi.entityService.findMany(
29.     'api::interaction.interaction',
30.     {
31.       filters: {
32.         sessionId: { $in: sessions },
33.         product: { id: { $ne: product.id } },
34.       },
35.       fields: ['eventType'],
36.       populate: ['product'],
37.       limit: 3000,
38.     }
39.   )
40.
41.   const scores = new Map<number, number>()
42.
43.   for (const interaction of interactions as any[]) {
44.     const id = interaction.product?.id
45.     if (!id) continue
46.
47.     scores.set(
48.       id,
49.       (scores.get(id) || 0) + (WEIGHTS[interaction.eventType] || 0)
50.     )
51.   }
52.
53.   return Array.from(scores.entries())
54.     .sort((a, b) => b[1] - a[1])
55.     .slice(0, limit)
56.     .map(([id]) => id)
57. }
58.

```

Fallback με βάση κατηγορία προϊόντος

```

1. // src/api/recommendation/controllers/recommendation.ts
2.
3. async function categoryFallback(strapi: any, product: any, limit: number) {
4.   const categoryIds = (product.categories || []).map(
5.     (category: any) => category.id
6.   )
7.
8.   if (categoryIds.length === 0) return []
9.
10.  const products = await strapi.documents('api::product.product').findMany({
11.    status: 'published',

```

```

12.   filters: {
13.     id: { $ne: product.id },
14.     inStock: true,
15.     categories: { id: { $in: categoryIds } },
16.   },
17.   populate: ['categories', 'image'],
18.   sort: { createdAt: 'desc' },
19.   limit: 50,
20. })
21.
22. return products.slice(0, limit)
23. }
24.

```

Fallback με βάση δημοφιλία

```

1. // src/api/recommendation/controllers/recommendation.ts
2.
3. async function popularFallback(
4.   strapi: any,
5.   product: any,
6.   limit: number,
7.   exclude: number[]
8. ) {
9.   const interactions = await strapi.entityService.findMany(
10.    'api::interaction.interaction',
11.    {
12.      filters: {
13.        product: { id: { $notIn: [product.id, ...exclude] } },
14.      },
15.      fields: ['eventType'],
16.      populate: ['product'],
17.      limit: 3000,
18.    }
19.  )
20.
21.  const scores = new Map<number, number>()
22.
23.  for (const interaction of interactions as any[]) {
24.    const id = interaction.product?.id
25.    if (!id) continue
26.
27.    scores.set(
28.      id,
29.      (scores.get(id) || 0) + (WEIGHTS[interaction.eventType] || 0)
30.    )
31.  }
32.
33.  const ids = Array.from(scores.entries())
34.    .sort((a, b) => b[1] - a[1])
35.    .slice(0, limit)
36.    .map(([id]) => id)
37.
38.  return fetchProducts(strapi, ids, limit)
39. }
40.

```

Κεντρικό endpoint recommendation

```

1. // src/api/recommendation/controllers/recommendation.ts
2.
3. export default {
4.   async find(ctx: any) {
5.     const limit = parseLimit(ctx.query.limit)
6.     const product = await findTargetProduct(strapi, ctx.query)
7.
8.     if (!product) {
9.       return ctx.badRequest('Product was not found.')
10.    }

```

```

11.
12.   const collaborativeIds = await collaborativeCandidates(strapi, product, limit)
13.   const collaborativeProducts = await fetchProducts(
14.     strapi,
15.     collaborativeIds,
16.     limit
17.   )
18.
19.   const categoryProducts = await categoryFallback(
20.     strapi,
21.     product,
22.     limit - collaborativeProducts.length
23.   )
24.
25.   const categoryIds = categoryProducts.map((item: any) => item.id)
26.
27.   const popularProducts = await popularFallback(
28.     strapi,
29.     product,
30.     limit - collaborativeProducts.length - categoryProducts.length,
31.     [...collaborativeIds, ...categoryIds]
32.   )
33.
34.   const usedIds = [
35.     ...collaborativeProducts.map((item: any) => item.id),
36.     ...categoryProducts.map((item: any) => item.id),
37.     ...popularProducts.map((item: any) => item.id),
38.   ]
39.
40.   const latestProducts = await latestFallback(
41.     strapi,
42.     product,
43.     limit - usedIds.length,
44.     usedIds
45.   )
46.
47.   const seen = new Set<number>()
48.
49.   const data = [
50.     ...collaborativeProducts.map((item: any) => ({
51.       ...item,
52.       recommendationReason: 'collaborative',
53.     })),
54.     ...categoryProducts.map((item: any) => ({
55.       ...item,
56.       recommendationReason: 'category',
57.     })),
58.     ...popularProducts.map((item: any) => ({
59.       ...item,
60.       recommendationReason: 'popular',
61.     })),
62.     ...latestProducts.map((item: any) => ({
63.       ...item,
64.       recommendationReason: 'latest',
65.     })),
66.   ]
67.   .filter((item: any) => {
68.     if (seen.has(item.id)) return false
69.     seen.add(item.id)
70.     return true
71.   })
72.   .slice(0, limit)
73.
74.   ctx.body = {
75.     data,
76.     meta: {
77.       algorithm:
78.         'item-session collaborative filtering with category/popularity fallback',
79.     },
80.   }

```

```
81.   },
82. }
83.
```

Καταγραφή user interactions

```
1. // src/api/interaction/routes/interaction.ts
2.
3. export default {
4.   routes: [
5.     {
6.       method: 'POST',
7.       path: '/interactions/track',
8.       handler: 'interaction.track',
9.       config: {
10.        auth: false,
11.      },
12.    },
13.    {
14.      method: 'GET',
15.      path: '/interactions/summary',
16.      handler: 'interaction.summary',
17.      config: {
18.        auth: false,
19.      },
20.    },
21.  ],
22. }
23.
```

```
1// lib/analytics.ts
2.
3. export function trackInteraction(payload: TrackPayload) {
4.   if (typeof window === 'undefined') return
5.   if (!payload.productId && !payload.documentId) return
6.
7.   const body = JSON.stringify({
8.     data: {
9.       ...payload,
10.      sessionId: getSessionId(),
11.      variant: getAbVariant(),
12.    },
13.  })
14.
15.   const url = getAnalyticsUrl()
16.
17.   if (navigator.sendBeacon) {
18.     const blob = new Blob([body], { type: 'application/json' })
19.     navigator.sendBeacon(url, blob)
20.     return
21.   }
22.
23.   fetch(url, {
24.     method: 'POST',
25.     headers: {
26.       'Content-Type': 'application/json',
27.     },
28.     body,
29.     keepalive: true,
30.   }).catch(() => {})
31. }
32.
```

Frontend φόρτωση προτεινόμενων προϊόντων

```
1. // components/RecommendedProducts.tsx
2.
3. async function getRecommendations(productId: number, documentId?: string) {
4.   try {
```

```

5.     const response = await fetchStrapi<RecommendationResponse>(
6.       '/api/recommendations',
7.       {
8.         productId,
9.         documentId,
10.        limit: 4,
11.      }
12.    )
13.
14.    return response.data
15.  } catch {
16.    return []
17.  }
18. }
19.
20. export default async function RecommendedProducts({
21.   productId,
22.   documentId,
23. }: Props) {
24.   const products = await getRecommendations(productId, documentId)
25.
26.   if (!products.length) return null
27.
28.   return (
29.     <RecommendationPanel
30.       productId={productId}
31.       products={products}
32.     />
33.   )
34. }
35.

```

Ο AI recommender υλοποιείται ως υβριδικός μηχανισμός προτάσεων. Η βασική τεχνική είναι item-session collaborative filtering, όπου αναζητούνται sessions χρηστών που αλληλεπιδράσαν με το τρέχον προϊόν και στη συνέχεια βαθμολογούνται άλλα προϊόντα που εμφανίστηκαν στα ίδια sessions. Οι αλληλεπιδράσεις έχουν διαφορετικά βάρη, με μεγαλύτερη σημασία σε αγορές και προσθήκες στο καλάθι. Αν δεν υπάρχουν επαρκή δεδομένα, το σύστημα χρησιμοποιεί fallback με βάση την κατηγορία, τη δημοφιλία και τα νεότερα προϊόντα. Έτσι, το recommender λειτουργεί ακόμη και σε περιπτώσεις cold start.

Δ2.Υλοποίηση Stripe PaymentIntents

Αρχείο: src/api/order/routes/custom-order.ts

Strapi custom routes

```

1. export default {
2.   routes: [
3.     {
4.       method: 'POST',
5.       path: '/payments/create-intent',
6.       handler: 'order.createPaymentIntent',
7.       config: {
8.         auth: {},
9.       },
10.    },
11.    {
12.      method: 'POST',
13.      path: '/payments/finalize',
14.      handler: 'order.finalizePayment',
15.      config: {
16.        auth: {},
17.      },

```

```

18.   },
19. ],
20. }
21.

```

Δημιουργία PaymentIntent στο Strapi

```

1. async createPaymentIntent(ctx) {
2.   const user = ctx.state.user
3.
4.   if (!user) {
5.     return ctx.unauthorized('You must be logged in.')
6.   }
7.
8.   try {
9.     const { items, total } = await buildOrderFromCart(
10.      strapi,
11.      ctx.request.body.data?.items
12.    )
13.
14.    const params = new URLSearchParams()
15.
16.    params.append('amount', String(Math.round(total * 100)))
17.    params.append('currency', 'eur')
18.    params.append('automatic_payment_methods[enabled]', 'true')
19.    params.append('metadata[user_id]', String(user.id))
20.    params.append('metadata[source]', 'strapi_embedded_checkout')
21.    params.append('metadata[item_count]', String(items.length))
22.
23.    const paymentIntent = await stripeRequest('payment_intents', params)
24.
25.    return {
26.      clientSecret: paymentIntent.client_secret,
27.      paymentIntentId: paymentIntent.id,
28.      total,
29.    }
30.  } catch (error) {
31.    if (error instanceof OrderPayloadError) {
32.      return ctx.badRequest(error.message)
33.    }
34.
35.    throw error
36.  }
37. }
38.

```

Οριστικοποίηση πληρωμής και δημιουργία paid order

```

1. // src/api/order/controllers/order.ts
2.
3. async finalizePayment(ctx) {
4.   const user = ctx.state.user
5.   const paymentIntentId = ctx.request.body.data?.paymentIntentId
6.
7.   if (!user) {
8.     return ctx.unauthorized('You must be logged in.')
9.   }
10.
11.   if (typeof paymentIntentId !== 'string' || !paymentIntentId.startsWith('pi_')) {
12.     return ctx.badRequest('Invalid payment intent.')
13.   }
14.
15.   try {
16.     const paymentIntent = await stripeRequest(
17.       `payment_intents/${encodeURIComponent(paymentIntentId)}`
18.     )
19.
20.     if (paymentIntent.status !== 'succeeded') {
21.       return ctx.badRequest('Payment has not succeeded.')

```

```

22.     }
23.
24.     const { items, total } = await buildOrderFromCart(
25.       strapi,
26.       ctx.request.body.data?.items
27.     )
28.
29.     const expectedAmount = Math.round(total * 100)
30.
31.     if (paymentIntent.amount !== expectedAmount || paymentIntent.currency !== 'eur') {
32.       return ctx.badRequest('Payment amount does not match the cart.')
33.     }
34.
35.     const order = await createOrder(strapi, user.id, items, total, 'paid')
36.
37.     return this.transformResponse(order)
38.   } catch (error) {
39.     if (error instanceof OrderPayloadError) {
40.       return ctx.badRequest(error.message)
41.     }
42.
43.     throw error
44.   }
45. }

```

Frontend confirmation με Stripe.js

```

1. // app/checkout/page.tsx
2.
3. async function handleEmbeddedStripePayment(token: string) {
4.   const stripe = stripeRef.current
5.   const card = cardRef.current
6.
7.   if (!stripe || !card || !cardReady) {
8.     throw new Error('Το πεδίο κάρτας δεν είναι ακόμη έτοιμο.')
9.   }
10.
11.   const intentRes = await fetch(getPaymentIntentUrl(), {
12.     method: 'POST',
13.     headers: {
14.       'Content-Type': 'application/json',
15.       Authorization: `Bearer ${token}`,
16.     },
17.     body: JSON.stringify({
18.       data: {
19.         items,
20.       },
21.     }),
22.   })
23.
24.   const intentData = await intentRes.json()
25.
26.   if (!intentRes.ok || !intentData?.clientSecret || !intentData.paymentIntentId) {
27.     throw new Error('Το Strapi δεν δημιούργησε PaymentIntent.')
28.   }
29.
30.   const confirmation = await stripe.confirmCardPayment(intentData.clientSecret, {
31.     payment_method: {
32.       card,
33.     },
34.   })
35.
36.   if (confirmation.error) {
37.     throw new Error(confirmation.error.message || 'Η πληρωμή απορρίφθηκε.')
38.   }
39.
40.   if (confirmation.paymentIntent?.status !== 'succeeded') {
41.     throw new Error('Η πληρωμή δεν ολοκληρώθηκε.')

```

```

42.   }
43.
44.   const finalizeRes = await fetch(getPaymentFinalizeUrl(), {
45.     method: 'POST',
46.     headers: {
47.       'Content-Type': 'application/json',
48.       Authorization: `Bearer ${token}`,
49.     },
50.     body: JSON.stringify({
51.       data: {
52.         items,
53.         paymentIntentId: intentData.paymentIntentId,
54.       },
55.     }),
56.   })
57.
58.   if (!finalizeRes.ok) {
59.     throw new Error('Η paid παραγγελία δεν αποθηκεύτηκε στο Strapi.')
60.   }
61. }
62.

```

Μεταβλητές περιβάλλοντος

```

1. # Frontend
2. NEXT_PUBLIC_PAYMENT_PROVIDER=stripe
3. NEXT_PUBLIC_STRIPE_PUBLISHABLE_KEY=pk_test_51LUxIyK9CqJKN5RekA2lvvaLUHpANGHK24LDcfEKCv40Cje
  EVNuTnQ6yUn6ipkw9IMKBCUoOWmXzycxs0bJYBNE700EeszGpNT
4.
5. # Strapi backend
6. STRIPE_SECRET_KEY=sk_test_51LUxIyK9CqJKN5ReMhBsL0g5Mh1Lfd7TdQsQXN2dTiusP77602YISBdcgBfGz0bX
  x9EntUi56k8ZPz9T5RqhL2K200WFQbJvp0
7.

```

Δ3. Κώδικας Interaction Tracking και A/B Testing

Strapi routes για interaction tracking

```

1. // src/api/interaction/routes/interaction.ts
2.
3. export default {
4.   routes: [
5.     {
6.       method: 'POST',
7.       path: '/interactions/track',
8.       handler: 'interaction.track',
9.       config: {
10.        auth: false,
11.      },
12.    },
13.    {
14.      method: 'GET',
15.      path: '/interactions/summary',
16.      handler: 'interaction.summary',
17.      config: {
18.        auth: false,
19.      },
20.    },
21.  ],
22. }

```

Τύποι interaction events

```

1. // src/api/interaction/controllers/interaction.ts
2.
3. const EVENT_TYPES = new Set([
4.   'view',
5.   'add_to_cart',
6.   'purchase',
7.   'recommendation_impression',
8.   'recommendation_click',
9. ])
10.
11. const VARIANTS = new Set(['A', 'B'])
12.

```

Καταγραφή interaction στο Strapi

```

1. // src/api/interaction/controllers/interaction.ts
2.
3. async track(ctx) {
4.   if (isRateLimited(ctx)) {
5.     return ctx.tooManyRequests('Too many analytics events.')
6.   }
7.
8.   const input = normalizeInput(ctx.request.body)
9.   const eventType =
10.     typeof input.eventType === 'string' ? input.eventType : ''
11.
12.   const sessionId =
13.     typeof input.sessionId === 'string' ? input.sessionId.trim() : ''
14.
15.   const variant =
16.     typeof input.variant === 'string' && VARIANTS.has(input.variant)
17.     ? input.variant
18.     : 'A'
19.
20.   if (!EVENT_TYPES.has(eventType)) {
21.     return ctx.badRequest('Invalid interaction event type.')
22.   }
23.
24.   if (sessionId.length < 8 || sessionId.length > 120) {
25.     return ctx.badRequest('Invalid session id.')
26.   }
27.
28.   const product = await findProduct(
29.     strapi,
30.     input.productId,
31.     input.documentId
32.   )
33.
34.   if (!product) {
35.     return ctx.badRequest('Product was not found.')
36.   }
37.
38.   const data = {
39.     eventType,
40.     sessionId,
41.     variant,
42.     source:
43.       typeof input.source === 'string'
44.       ? input.source.slice(0, 80)
45.       : null,
46.     product: product.id,
47.     user: ctx.state.user?.id || null,
48.     metadata:
49.       input.metadata && typeof input.metadata === 'object'
50.       ? input.metadata
51.       : null,
52.   }
53.

```

```

54.   const interaction = await strapi.entityService.create(
55.     'api::interaction.interaction',
56.     {
57.       data,
58.     }
59.   )
60.
61.   return this.transformResponse(interaction)
62. }
63.

```

Frontend αποστολή interaction event

```

1. // lib/analytics.ts
2.
3. export type InteractionEvent =
4.   | 'view'
5.   | 'add_to_cart'
6.   | 'purchase'
7.   | 'recommendation_impression'
8.   | 'recommendation_click'
9.
10. type TrackPayload = {
11.   eventType: InteractionEvent
12.   productId?: number
13.   documentId?: string
14.   source?: string
15.   metadata?: Record<string, unknown>
16. }
17.
18. export function trackInteraction(payload: TrackPayload) {
19.   if (typeof window === 'undefined') return
20.   if (!payload.productId && !payload.documentId) return
21.
22.   const body = JSON.stringify({
23.     data: {
24.       ...payload,
25.       sessionId: getSessionId(),
26.       variant: getAbVariant(),
27.     },
28.   })
29.
30.   const url = getAnalyticsUrl()
31.
32.   if (navigator.sendBeacon) {
33.     const blob = new Blob([body], { type: 'application/json' })
34.     navigator.sendBeacon(url, blob)
35.     return
36.   }
37.
38.   fetch(url, {
39.     method: 'POST',
40.     headers: {
41.       'Content-Type': 'application/json',
42.     },
43.     body,
44.     keepalive: true,
45.   }).catch(() => {})
46. }
47.

```

Καταγραφή recommendation impressions και clicks

```

1. // components/RecommendationPanel.tsx
2.
3. useEffect(() => {

```

```

4.   const currentVariant = getAbVariant()
5.
6.   window.setTimeout(() => setVariant(currentVariant), 0)
7.
8.   for (const product of products) {
9.     trackInteraction({
10.      eventType: 'recommendation_impression',
11.      productId: product.id,
12.      documentId: product.documentId,
13.      source: `recommendations_variant_${currentVariant}`,
14.      metadata: {
15.        targetProductId: productId,
16.        reason: product.recommendationReason,
17.      },
18.    })
19.  }
20. }, [productId, products])
21.

```

```

1. // components/RecommendationPanel.tsx
2.
3. <Link
4.   href={` /products/${product.slug}`}
5.   onClick={() =>
6.     trackInteraction({
7.       eventType: 'recommendation_click',
8.       productId: product.id,
9.       documentId: product.documentId,
10.      source: `recommendations_variant_${variant}`,
11.      metadata: {
12.        targetProductId: productId,
13.        reason: product.recommendationReason,
14.      },
15.    })
16.  }
17. >
18.   /* Product card */
19. </Link>
20.

```

Το interaction tracking καταγράφει βασικές ενέργειες χρήστη, όπως προβολή προϊόντος, προσθήκη στο καλάθι, αγορά, εμφάνιση προτεινόμενου προϊόντος και κλικ σε πρόταση. Κάθε interaction συνδέεται με session id και A/B variant, ώστε να είναι δυνατή η ανάλυση της συμπεριφοράς ανά πειραματική εκδοχή του UI.

Το A/B testing υλοποιείται client-side με τυχαία ανάθεση του χρήστη σε variant A ή B, η οποία αποθηκεύεται στο localStorage. Με αυτόν τον τρόπο ο ίδιος χρήστης βλέπει σταθερά την ίδια εκδοχή κατά τη διάρκεια της χρήσης της εφαρμογής. Τα aggregated metrics υπολογίζονται στο Strapi και περιλαμβάνουν conversion rate, add-to-cart rate και recommendation click-through rate.

Δ4. Evaluation Toolkit

Αρχείο ρυθμίσεων αξιολόγησης

```
1. // evaluation/config.json
2.
3. {
4.   "baseUrl": "http://127.0.0.1:3000",
5.   "strapiUrl": "http://127.0.0.1:1337",
6.   "pages": [
7.     "/",
8.     "/products",
9.     "/cart",
10.    "/checkout",
11.    "/orders",
12.    "/evaluation"
13.  ],
14.  "lighthouse": {
15.    "enabled": true,
16.    "categories": [
17.      "performance",
18.      "accessibility",
19.      "best-practices",
20.      "seo"
21.    ]
22.  },
23.  "security": {
24.    "headers": [
25.      "content-security-policy",
26.      "x-frame-options",
27.      "x-content-type-options",
28.      "referrer-policy"
29.    ]
30.  }
31. }
32.
```

Κεντρικό script αξιολόγησης

```
1. // evaluation/run-evaluation.mjs
2.
3. import fs from 'node:fs/promises'
4. import path from 'node:path'
5. import { performance } from 'node:perf_hooks'
6.
7. const configPath = path.join(process.cwd(), 'evaluation', 'config.json')
8. const outputDir = path.join(process.cwd(), 'evaluation', 'reports')
9.
10. async function readConfig() {
11.   const raw = await fs.readFile(configPath, 'utf8')
12.   return JSON.parse(raw)
13. }
14.
15. async function measureHttp(url) {
16.   const start = performance.now()
17.
18.   try {
19.     const response = await fetch(url, {
20.       redirect: 'manual',
21.       cache: 'no-store',
22.     })
23.
24.     const firstByteMs = Math.round(performance.now() - start)
25.     const bodyStart = performance.now()
26.     await response.arrayBuffer()
27.     const totalMs = Math.round(performance.now() - start)
28.
29.     return {
```

```

30.     url,
31.     status: response.status,
32.     ok: response.ok,
33.     ttfbMs: firstByteMs,
34.     totalMs,
35.     downloadMs: Math.round(performance.now() - bodyStart),
36.     headers: Object.fromEntries(response.headers.entries()),
37.   }
38. } catch (error) {
39.   return {
40.     url,
41.     status: 0,
42.     ok: false,
43.     error: error instanceof Error ? error.message : 'Unknown error',
44.   }
45. }
46. }
47.

```

Έλεγχος security headers

```

1. // evaluation/run-evaluation.mjs
2.
3. function evaluateSecurityHeaders(headers, expectedHeaders) {
4.   return expectedHeaders.map((header) => {
5.     const value = headers[header] || null
6.
7.     return {
8.       header,
9.       present: Boolean(value),
10.      value,
11.    }
12.  })
13. }
14.
15. function scoreSecurityHeaders(results) {
16.   const total = results.length
17.   const present = results.filter((item) => item.present).length
18.
19.   return total > 0 ? Math.round((present / total) * 100) : 0
20. }
21.

```

Μέτρηση frontend pages

```

1. // evaluation/run-evaluation.mjs
2.
3. async function evaluatePages(config) {
4.   const results = []
5.
6.   for (const page of config.pages) {
7.     const url = new URL(page, config.baseUrl).toString()
8.     const measurement = await measureHttp(url)
9.
10.    const securityHeaders = evaluateSecurityHeaders(
11.      measurement.headers || {},
12.      config.security.headers
13.    )
14.
15.    results.push({
16.      page,
17.      url,
18.      status: measurement.status,
19.      ok: measurement.ok,
20.      ttfbMs: measurement.ttfbMs,
21.      totalMs: measurement.totalMs,

```

```

22.     securityHeaders,
23.     securityScore: scoreSecurityHeaders(securityHeaders),
24.   })
25. }
26.
27. return results
28. }
29.

```

Έλεγχος Strapi backend

```

1. // evaluation/run-evaluation.mjs
2.
3. async function evaluateBackend(config) {
4.   const endpoints = [
5.     '/api/products',
6.     '/api/categories',
7.     '/api/recommendations',
8.   ]
9.
10.  const results = []
11.
12.  for (const endpoint of endpoints) {
13.    const url = new URL(endpoint, config.strapiUrl).toString()
14.    const measurement = await measureHttp(url)
15.
16.    results.push({
17.      endpoint,
18.      url,
19.      status: measurement.status,
20.      ok: measurement.ok,
21.      ttfbMs: measurement.ttfbMs,
22.      totalMs: measurement.totalMs,
23.    })
24.  }
25.
26.  return results
27. }
28.

```

Δημιουργία report

```

1. // evaluation/run-evaluation.mjs
2.
3. async function writeReport(report) {
4.   await fs.mkdir(outputDir, { recursive: true })
5.
6.   const latestPath = path.join(outputDir, 'latest.json')
7.   const summaryPath = path.join(outputDir, 'summary.md')
8.
9.   await fs.writeFile(latestPath, JSON.stringify(report, null, 2), 'utf8')
10.
11.  const pageRows = report.frontendPages
12.    .map(
13.      (page) =>
14.        `| ${page.page} | ${page.status} | ${page.ttfbMs}ms | ${page.totalMs}ms |
15.        ${page.securityScore}% |`
16.    )
17.    .join('\n')
18.
19.  const backendRows = report.backendEndpoints
20.    .map(
21.      (endpoint) =>
22.        `| ${endpoint.endpoint} | ${endpoint.status} | ${endpoint.ttfbMs}ms |
23.        ${endpoint.totalMs}ms |`
24.    )
25.
26.  const backendSummary = backendRows.join('\n')
27.
28.  const reportSummary = `${pageRows}\n\n${backendSummary}`
29.
30.  await fs.writeFile(summaryPath, reportSummary, 'utf8')
31.
32.  return { latestPath, summaryPath }
33. }
34.

```

```

23.     .join('\n')
24.
25.     const markdown = `# Evaluation Report
26.
27.     Generated: ${report.generatedAt}
28.
29.     ## Frontend Pages
30.
31.     | Page | Status | TTFB | Total | Security |
32.     | --- | ---: | ---: | ---: | ---: |
33.     ${pageRows}
34.
35.     ## Backend Endpoints
36.
37.     | Endpoint | Status | TTFB | Total |
38.     | --- | ---: | ---: | ---: |
39.     ${backendRows}
40. `
41.
42.     await fs.writeFile(summaryPath, markdown, 'utf8')
43. }
44.

```

Προβολή αποτελεσμάτων στο frontend

```

1. // app/evaluation/page.tsx
2.
3. async function getEvaluationReport() {
4.   try {
5.     const reportPath = path.join(
6.       process.cwd(),
7.       'evaluation',
8.       'reports',
9.       'latest.json'
10.    )
11.
12.    const raw = await fs.readFile(reportPath, 'utf8')
13.    return JSON.parse(raw)
14.  } catch {
15.    return null
16.  }
17. }
18.
19. export default async function EvaluationPage() {
20.   const report = await getEvaluationReport()
21.
22.   return (
23.     <main className="shell py-10">
24.       <h1 className="text-4xl font-black">Αξιολόγηση πλατφόρμας</h1>
25.
26.       {report ? (
27.         <section>
28.           <h2>Τελευταία μετρική αξιολόγηση</h2>
29.           <p>Generated: {report.generatedAt}</p>
30.         </section>
31.       ) : (
32.         <p>Δεν υπάρχει διαθέσιμο evaluation report ακόμη.</p>
33.       )}
34.     </main>
35.   )
36. }

```

Το Evaluation Toolkit υλοποιήθηκε ως βοηθητικό εργαλείο μέτρησης της πλατφόρμας. Εκτελεί HTTP μετρήσεις σε βασικές σελίδες του frontend και endpoints του Strapi backend, καταγράφοντας status code, Time-to-First-Byte, συνολικό χρόνο απόκρισης και βασικούς

security headers. Τα αποτελέσματα αποθηκεύονται σε JSON και Markdown report, ώστε να μπορούν να χρησιμοποιηθούν τόσο για τεχνική τεκμηρίωση όσο και για συγκριτική αξιολόγηση της πλατφόρμας έναντι παραδοσιακών CMS.

Δ5. Serverless Function-Style Services

Serverless recommendations handler

```
1. // serverless/recommendations/handler.mjs
2.
3. import { jsonResponse, getQuery } from '../shared/response.mjs'
4.
5. const STRAPI_URL = process.env.STRAPI_URL || 'http://127.0.0.1:1337'
6.
7. export async function handler(event) {
8.   const query = getQuery(event)
9.   const productId = query.productId || query.id
10.  const documentId = query.documentId
11.  const limit = query.limit || 4
12.
13.  const params = new URLSearchParams()
14.
15.  if (productId) params.set('productId', productId)
16.  if (documentId) params.set('documentId', documentId)
17.  params.set('limit', limit)
18.
19.  const response = await fetch(
20.    `${STRAPI_URL}/api/recommendations?${params.toString()}`
21.  )
22.
23.  const data = await response.json().catch(() => null)
24.
25.  if (!response.ok) {
26.    return jsonResponse(response.status, {
27.      error: data?.error?.message || 'Recommendations service failed.',
28.      service: 'recommendations',
29.      architecture: 'serverless-microservice',
30.    })
31.  }
32.
33.  return jsonResponse(200, {
34.    ...data,
35.    service: 'recommendations',
36.    architecture: 'serverless-microservice',
37.  })
38. }
39.
```

Serverless analytics handler

```
1. // serverless/analytics/handler.mjs
2.
3. import { jsonResponse, parseJsonBody } from '../shared/response.mjs'
4.
5. const STRAPI_URL = process.env.STRAPI_URL || 'http://127.0.0.1:1337'
6.
7. export async function handler(event) {
8.   const payload = parseJsonBody(event)
9.
10.  const response = await fetch(`${STRAPI_URL}/api/interactions/track`, {
11.    method: 'POST',
12.    headers: {
```

```

13.     'Content-Type': 'application/json',
14.   },
15.   body: JSON.stringify(payload),
16. })
17.
18. const data = await response.json().catch(() => null)
19.
20. if (!response.ok) {
21.   return jsonResponse(response.status, {
22.     error: data?.error?.message || 'Analytics tracking failed.',
23.     service: 'analytics',
24.     architecture: 'serverless-microservice',
25.   })
26. }
27.
28. return jsonResponse(200, {
29.   ...data,
30.   service: 'analytics',
31.   architecture: 'serverless-microservice',
32. })
33. }
34.

```

Serverless checkout handler

```

1. // serverless/checkout/handler.mjs
2.
3. import { jsonResponse, parseJsonBody } from '../shared/response.mjs'
4.
5. const STRAPI_URL = process.env.STRAPI_URL || 'http://127.0.0.1:1337'
6.
7. export async function handler(event) {
8.   const payload = parseJsonBody(event)
9.   const authorization = event.headers?.authorization || event.headers?.Authorization
10.
11.   if (!authorization) {
12.     return jsonResponse(401, {
13.       error: 'Missing authorization token.',
14.       service: 'checkout',
15.       architecture: 'serverless-microservice',
16.     })
17.   }
18.
19.   const response = await fetch(`${STRAPI_URL}/api/orders`, {
20.     method: 'POST',
21.     headers: {
22.       'Content-Type': 'application/json',
23.       Authorization: authorization,
24.     },
25.     body: JSON.stringify(payload),
26.   })
27.
28.   const data = await response.json().catch(() => null)
29.
30.   if (!response.ok) {
31.     return jsonResponse(response.status, {
32.       error: data?.error?.message || 'Checkout service failed.',
33.       service: 'checkout',
34.       architecture: 'serverless-microservice',
35.     })
36.   }
37.
38.   return jsonResponse(200, {
39.     ...data,
40.     service: 'checkout',
41.     architecture: 'serverless-microservice',
42.   })
43. }

```

Local serverless adapter

```

1. // serverless/local-dev.mjs
2.
3. import http from 'node:http'
4. import { handler as recommendations } from './recommendations/handler.mjs'
5. import { handler as analytics } from './analytics/handler.mjs'
6. import { handler as checkout } from './checkout/handler.mjs'
7.
8. const PORT = Number(process.env.SERVERLESS_PORT || 8787)
9.
10. const routes = {
11.   'GET /recommendations': recommendations,
12.   'POST /track': analytics,
13.   'POST /checkout': checkout,
14. }
15.
16. function readBody(request) {
17.   return new Promise((resolve) => {
18.     let body = ''
19.
20.     request.on('data', (chunk) => {
21.       body += chunk
22.     })
23.
24.     request.on('end', () => {
25.       resolve(body)
26.     })
27.   })
28. }
29.
30. function toLambdaEvent(request, body) {
31.   const url = new URL(request.url, `http://${request.headers.host}`)
32.
33.   return {
34.     httpMethod: request.method,
35.     path: url.pathname,
36.     headers: request.headers,
37.     queryStringParameters: Object.fromEntries(url.searchParams.entries()),
38.     body,
39.   }
40. }
41.
42. const server = http.createServer(async (request, response) => {
43.   const url = new URL(request.url, `http://${request.headers.host}`)
44.   const routeKey = `${request.method} ${url.pathname}`
45.   const routeHandler = routes[routeKey]
46.
47.   if (!routeHandler) {
48.     response.writeHead(404, { 'Content-Type': 'application/json' })
49.     response.end(JSON.stringify({ error: 'Route was not found.' }))
50.     return
51.   }
52.
53.   const body = await readBody(request)
54.   const result = await routeHandler(toLambdaEvent(request, body))
55.
56.   response.writeHead(result.statusCode, result.headers)
57.   response.end(result.body)
58. })
59.
60. server.listen(PORT, () => {
61.   console.log(`Serverless demo listening on http://127.0.0.1:${PORT}`)
62. })
63.

```

Τα Serverless Function-Style Services υλοποιήθηκαν ως ενδεικτική μετάβαση της πλατφόρμας προς composable και MACH αρχιτεκτονική. Οι λειτουργίες recommendation, analytics tracking και checkout απομονώθηκαν σε ανεξάρτητους handlers που ακολουθούν λογική τύπου AWS Lambda ή Azure Functions. Κάθε handler δέχεται ένα event, επικοινωνεί με το Strapi μέσω HTTP API και επιστρέφει τυποποιημένη JSON απάντηση.

Η προσέγγιση αυτή επιτρέπει τη μελλοντική ανεξάρτητη ανάπτυξη, κλιμάκωση και αντικατάσταση επιμέρους υπηρεσιών χωρίς να απαιτείται πλήρης αναδόμηση της εφαρμογής. Το Strapi παραμένει το headless CMS και κεντρικό content backend, ενώ λειτουργίες όπως recommendations, analytics και checkout μπορούν να εκτελεστούν ως αυτόνομες cloud-native υπηρεσίες.

Δ6. Μεταβλητές Περιβάλλοντος για Serverless Endpoints

Frontend environment variables

```
1. 1. # Βασικό URL του Strapi backend
2. NEXT_PUBLIC_STRAPI_URL=http://127.0.0.1:1337
3.
4. # Προαιρετικό serverless / composable mode
5. # Αν οι παρακάτω μεταβλητές μείνουν κενές ή σχολιασμένες,
6. # το frontend χρησιμοποιεί απευθείας τα Strapi endpoints.
7.
8. NEXT_PUBLIC_RECOMMENDATIONS_URL=http://127.0.0.1:8787/recommendations
9. NEXT_PUBLIC_ANALYTICS_URL=http://127.0.0.1:8787/track
10. NEXT_PUBLIC_CHECKOUT_URL=http://127.0.0.1:8787/checkout
```

Επεξήγηση μεταβλητών

```
1. NEXT_PUBLIC_STRAPI_URL
2. Ορίζει τη διεύθυνση του Strapi backend. Χρησιμοποιείται ως βασικό fallback
3. για προϊόντα, κατηγορίες, παραγγελίες, recommendations και analytics.
4.
5. NEXT_PUBLIC_RECOMMENDATIONS_URL
6. Ορίζει ανεξάρτητο endpoint για την υπηρεσία προτάσεων προϊόντων.
7. Όταν υπάρχει, το frontend καλεί το serverless recommendations service
8. αντί για το Strapi endpoint /api/recommendations.
9.
10. NEXT_PUBLIC_ANALYTICS_URL
11. Ορίζει ανεξάρτητο endpoint για την καταγραφή interaction events.
12. Όταν υπάρχει, τα view, add_to_cart, purchase και recommendation events
13. στέλνονται στο serverless analytics service.
14.
15. NEXT_PUBLIC_CHECKOUT_URL
16. Ορίζει ανεξάρτητο endpoint για τη ροή checkout.
17. Όταν υπάρχει, η δημιουργία παραγγελίας μπορεί να περάσει από serverless
18. checkout orchestration service αντί να καλείται απευθείας το Strapi.
19.
```

Παράδειγμα .env.local για κανονική Strapi λειτουργία

```
1. 1. NEXT_PUBLIC_STRAPI_URL=http://127.0.0.1:1337
2.
```

```
3. # Τα serverless endpoints παραμένουν ανενεργά
4. # NEXT_PUBLIC_RECOMMENDATIONS_URL=
5. # NEXT_PUBLIC_ANALYTICS_URL=
6. # NEXT_PUBLIC_CHECKOUT_URL=
7.
```

Παράδειγμα .env.local για serverless demo λειτουργία

```
1. NEXT_PUBLIC_STRAPI_URL=http://127.0.0.1:1337
2.
3. NEXT_PUBLIC_RECOMMENDATIONS_URL=http://127.0.0.1:8787/recommendations
4. NEXT_PUBLIC_ANALYTICS_URL=http://127.0.0.1:8787/track
5. NEXT_PUBLIC_CHECKOUT_URL=http://127.0.0.1:8787/checkout
6.
```

Οι μεταβλητές περιβάλλοντος επιτρέπουν στο frontend να λειτουργεί είτε με απευθείας σύνδεση στο Strapi είτε με ενδιάμεσα serverless services. Η προσέγγιση αυτή κάνει την αρχιτεκτονική πιο ευέλικτη, επειδή κάθε λειτουργία μπορεί να μεταφερθεί σταδιακά σε ανεξάρτητο microservice χωρίς αλλαγές στον βασικό κώδικα του frontend. Αν δεν οριστεί κάποιο serverless endpoint, η εφαρμογή χρησιμοποιεί αυτόματα το αντίστοιχο Strapi fallback endpoint.

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.