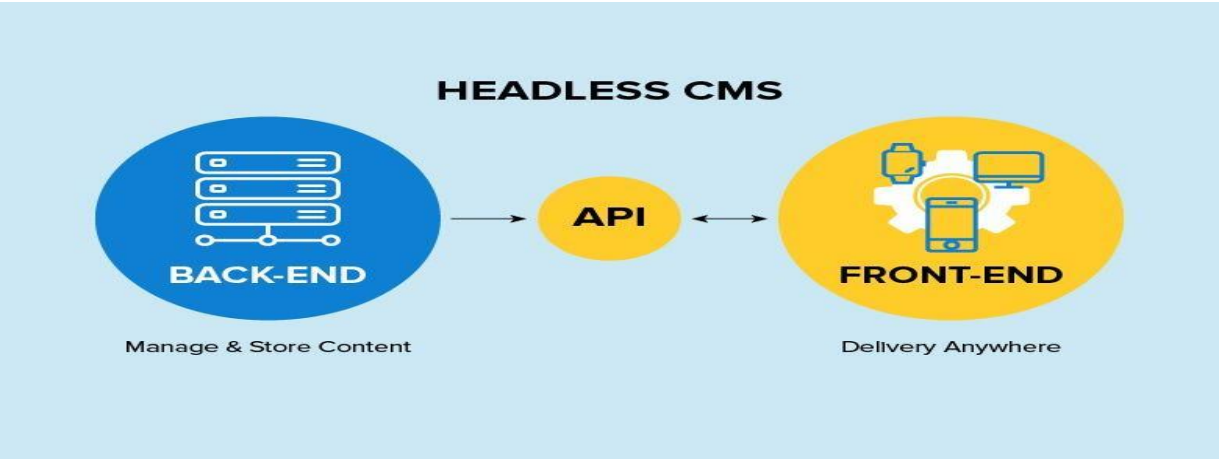
ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

«ΜΕΛΕΤΗ ΚΑΙ ΑΝΑΠΤΥΞΗ HEADLESS E-COMMERCE»



Του φοιτητή
Παπουνίδη Λάζαρου
154586

Επιβλέπων
Δρ. Τσιακμάκης Κυριάκος

Ιούνιος 2025

Μελέτη και ανάπτυξη Headless e-commerce

24225
Λάζαρος Παπουνίδης
Κυριάκος Τσιακμάκης

Ημερομηνία ανάληψης Π.Ε. 16-09-2024
Ημερομηνία περάτωσης Π.Ε. 20-05-2025

Βεβαιώνω ότι είμαι ο συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχω καταγράψει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών, εικόνων και κειμένου, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επιπλέον, βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά, ειδικά ως πτυχιακή εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

Η παρούσα εργασία αποτελεί πνευματική ιδιοκτησία του φοιτητή Παπουνίδη Λάζαρου που την εκπόνησε. Στο πλαίσιο της πολιτικής ανοικτής πρόσβασης, ο συγγραφέας/δημιουργός εκχωρεί στο Διεθνές Πανεπιστήμιο της Ελλάδος άδεια χρήσης του δικαιώματος αναπαραγωγής, δανεισμού, παρουσίασης στο κοινό και ψηφιακής διάχυσης της εργασίας διεθνώς, σε ηλεκτρονική μορφή και σε οποιοδήποτε μέσο, για διδακτικούς και ερευνητικούς σκοπούς, άνευ ανταλλάγματος. Η ανοικτή πρόσβαση στο πλήρες κείμενο της εργασίας, δεν σημαίνει καθ' οιονδήποτε τρόπο παραχώρηση δικαιωμάτων διανοητικής ιδιοκτησίας του συγγραφέα/δημιουργού, ούτε επιτρέπει την αναπαραγωγή, αναδημοσίευση, αντιγραφή, πώληση, εμπορική χρήση, διανομή, έκδοση, μεταφόρτωση (downloading), ανάρτηση (uploading), μετάφραση, τροποποίηση με οποιονδήποτε τρόπο, τμηματικά ή περιληπτικά της εργασίας, χωρίς τη ρητή προηγούμενη έγγραφη συναίνεση του συγγραφέα/δημιουργού.

Η έγκριση της πτυχιακής εργασίας από το Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του Διεθνούς Πανεπιστημίου της Ελλάδος, δεν υποδηλώνει απαραιτήτως και αποδοχή των απόψεων του συγγραφέα, εκ μέρους του Τμήματος.

Πρόλογος

Η παρούσα πτυχιακή εργασία έχει ως στόχο τη μελέτη και ανάπτυξη μιας σύγχρονης headless e-commerce εφαρμογής, αξιοποιώντας τεχνολογίες αιχμής του διαδικτύου. Στο πλαίσιο της εργασίας, υλοποιήθηκε το frontend της εφαρμογής με τη NEXT 14, μια ισχυρή πλατφόρμα ανάπτυξης όπου χωρίζοντας τα αρχεία σε client και server μας δίνει την δυνατότητα να αξιοποιήσουμε στο έπακρο την απόδοση και τις δυνατότητες.

Η διαχείριση του περιεχομένου και των δεδομένων πραγματοποιείται μέσω του CMS WordPress, το οποίο λειτουργεί ως headless backend. Η επικοινωνία μεταξύ frontend και backend επιτεύχθηκε με GraphQL, επιτρέποντας την αποδοτική και στοχευμένη ανάκτηση δεδομένων, ενώ για την υλοποίηση των queries και mutations χρησιμοποιήθηκε το Apollo Client.

Μέσα από αυτή την εργασία εξετάζονται τόσο οι θεωρητικές βάσεις του headless commerce, όσο και οι πρακτικές προκλήσεις και λύσεις που προέκυψαν κατά την ανάπτυξη. Σκοπός είναι να αναδειχθεί η ευελιξία και η δυναμική των headless αρχιτεκτονικών σε σύγχρονες εμπορικές εφαρμογές.

Περίληψη

Η παρούσα πτυχιακή εργασία πραγματεύεται τη μελέτη και ανάπτυξη μιας e-commerce εφαρμογής βασισμένης σε headless αρχιτεκτονική, αξιοποιώντας σύγχρονες τεχνολογίες ανάπτυξης ιστοσελίδων και διαχείρισης περιεχομένου. Η υλοποίηση έγινε με χρήση της NEXT 14 για το frontend, ενώ ως backend επιλέχθηκε το WordPress σε headless μορφή, προσφέροντας τη δυνατότητα αποσύνδεσης της παρουσίασης από τη διαχείριση δεδομένων.

Η επικοινωνία μεταξύ του frontend και του CMS πραγματοποιήθηκε μέσω του πρωτοκόλλου GraphQL, το οποίο προσφέρει ευελιξία, ταχύτητα και ακριβή έλεγχο στην ανάκτηση των δεδομένων. Για τη διαχείριση των ερωτημάτων και των μεταβολών των δεδομένων (queries & mutations), χρησιμοποιήθηκε η βιβλιοθήκη Apollo Client, που διευκολύνει τη σύνδεση της εφαρμογής με το backend και εξασφαλίζει σταθερή και αποδοτική λειτουργία.

Μέσα από τη διαδικασία ανάπτυξης, έγινε εκτενής μελέτη των πλεονεκτημάτων που προσφέρει η headless προσέγγιση, όπως είναι η ευκολία στην προσαρμογή, η δυνατότητα κλιμάκωσης, η ανεξαρτησία του frontend από το backend και η καλύτερη εμπειρία χρήστη. Παράλληλα, εντοπίστηκαν και αντιμετωπίστηκαν πρακτικές προκλήσεις, όπως η διαχείριση της ασφάλειας, η σύνδεση του WordPress με GraphQL μέσω plugins, και η οργάνωση των δεδομένων με τρόπο που να εξυπηρετεί τόσο τον χρήστη όσο και τη λειτουργικότητα της εφαρμογής.

Η εργασία ολοκληρώνεται με παρουσίαση του τελικού προϊόντος, ανάλυση της δομής του κώδικα και αποτίμηση της εμπειρίας από την ανάπτυξη, αναδεικνύοντας την πρακτική αξία των headless συστημάτων στο σύγχρονο ηλεκτρονικό εμπόριο.

«HEADLESS E-COMMERCE STUDY AND DEVELOPMENT»

«Lazaros Papounidis»

Abstract

This thesis focuses on the study and development of a headless e-commerce application, utilizing modern web technologies and a content management system. The frontend of the application was developed using NEXT 14, a powerful React-based framework known for its performance and server-side rendering capabilities. For the backend, WordPress was selected in a headless configuration, allowing for a clear separation between content management and presentation.

The communication between the frontend and the CMS was achieved through GraphQL, a query language that provides efficient and precise data retrieval. To manage the data flow and server interactions, Apollo Client was used, offering a robust and flexible way to handle GraphQL queries and mutations.

Throughout the development process, the benefits of the headless architecture were thoroughly explored, including adaptability, scalability, and the ability to develop frontend and backend independently. At the same time, several practical challenges were encountered and resolved, such as data structuring, integrating GraphQL with WordPress, and ensuring secure and optimized data handling.

The final result is a fully functional e-commerce platform that demonstrates the practical application of headless systems in modern web development. The structure of the code, the integration of technologies, and the performance of the application were carefully analyzed and documented. The project highlights how headless commerce can offer enhanced user experiences and greater flexibility for developers and businesses alike.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω, την οικογένεια μου, τους φίλους. Επίσης θα ήθελα να ευχαριστήσω τον επιβλέποντα της πτυχιακής εργασίας, τον Επίκουρο Καθηγητή κ. Κυριάκο Τσιακμάκη, για τις συμβουλές του οι οποίες υπήρξαν καθοριστικές για την ολοκλήρωση της εργασίας μου.

Περιεχόμενα

Κατάλογος Σχημάτων

Συντομογραφίες

SEO	Search Engine Optimization
NEXT	Next.js
REST	Representational State Transfer
API	Application Programming Interface
HTML	Hypertext Markup Language
CSS	Cascading Style Sheets
CMS	Content Management System
JWT	Json Web Token
SSR	Server Side Rendering
SSG	Static Site Generation
UI	User Interface
DNS	Domain Name System
URL	Uniform Resource Locator
GDPR	General Data Protection Regulation

Κεφάλαιο 1ο: Εισαγωγή

Η παρούσα πτυχιακή εργασία με τίτλο «Μελέτη και ανάπτυξη Headless e-commerce» εξετάζει τη μοντέρνα προσέγγιση του σχεδιασμού και της υλοποίησης ενός ηλεκτρονικού καταστήματος με χρήση headless αρχιτεκτονικής. Επικεντρώνεται σε μια μελέτη των τεχνολογιών που υποστηρίζουν τέτοιο μοντέλο και την πρακτική υλοποίηση με χρήση του WordPress ως συστήματος διαχείρισης περιεχομένου, της GraphQL για τον τρόπο με τον οποίο τα δεδομένα ανακτώνται, του Apollo Client για τη διαχείριση των ερωτημάτων και της NEXT 14 για το frontend. Η επιλογή τους δεν οφείλεται σε τυχαίες καταστάσεις αλλά οφείλεται σε μεγαλύτερη ευελιξία, ταχύτητα και βελτιστοποιημένη εμπειρία χρήστη που απαιτεί η μετάβαση σε τέτοια υποδομή. Αλλά ας πάρουμε τα πράγματα από την αρχή τους για το πως ξεκίνησαν αυτές οι τεχνολογίες.

1.1 Η ιστορία του Παγκόσμιου Ιστού και η μετάβαση στο Web 2.0/3.0

Ο Παγκόσμιος Ιστός, όπως τον γνωρίζουμε σήμερα, ξεκίνησε την πορεία του στις αρχές της δεκαετίας του '90 ως μια σειρά από στατικές ιστοσελίδες [1]. Όσο η τεχνολογία έβαλε μπρος και οι χρήστες ζητούσαν όλο και περισσότερα, το Web άρχισε να μεταμορφώνεται από ένα απλό μέσο πληροφόρησης (Web 1.0) σε ένα δυναμικό και αλληλεπιδραστικό περιβάλλον, γνωστό ως Web 2.0 [2]. Μαζί με αυτή την αλλαγή προέκυψε η εκτεταμένη χρήση δυναμικού περιεχομένου, η αλληλεπίδραση με τον χρήστη και η ενσωμάτωση κοινωνικών λειτουργιών. Σήμερα, με το Web 3.0 να κάνει τα πρώτα του βήματα, βλέπουμε την ανθρώπινη διάσταση, τη μηχανική μάθηση και τις αποκεντρωμένες τεχνολογίες να δημιουργούν ένα Διαδίκτυο που είναι πιο "έξυπνο" και ευέλικτο [2].

Εικόνα 1.1: Διαφορά Web 2.0 με Web 3.0

[https://www.stellardigital.in/blog/difference-between-web-2-0-and-web-3-0]

1.2 Σύγχρονες ανάγκες ανάπτυξης και η εμφάνιση των frontend framework

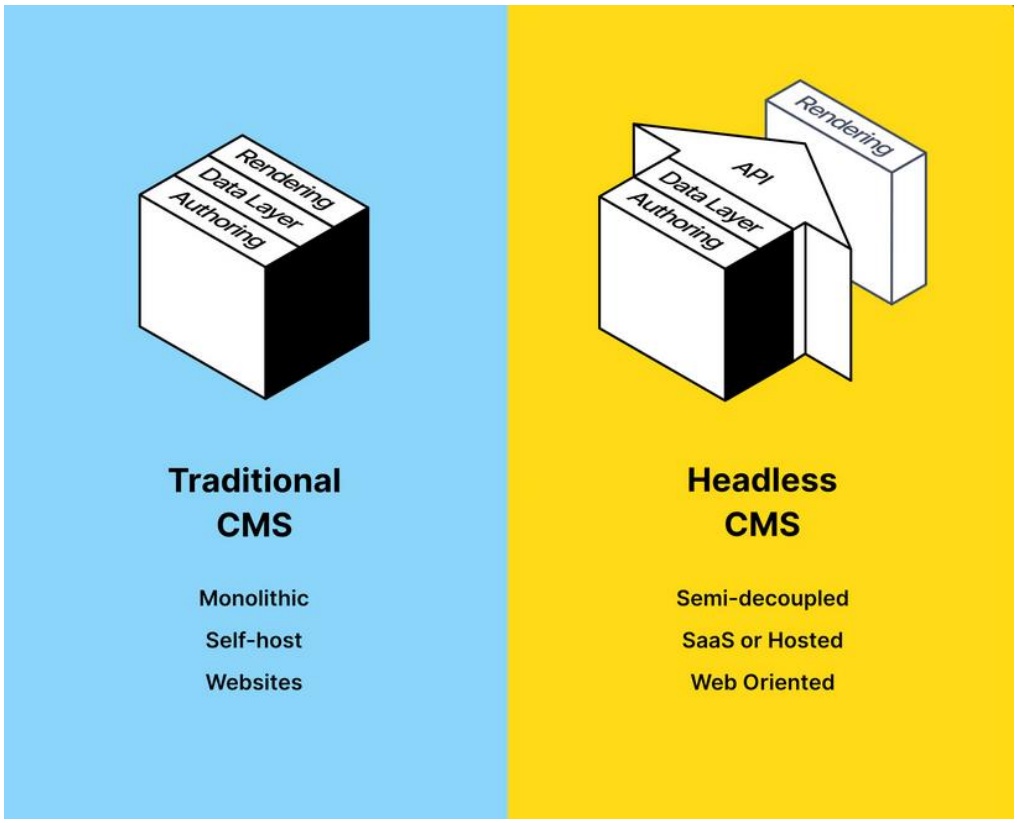
Με την αυξανόμενη πολυπλοκότητα των web εφαρμογών, έγινε γρήγορα εμφανές ότι η "παραδοσιακή" προσέγγιση ανάπτυξης με απλή HTML, CSS και JavaScript δεν επαρκούσε [3]. Έτσι γεννήθηκε η ανάγκη για πιο οργανωμένες, επεκτάσιμες και αποδοτικές λύσεις. Τα frontend frameworks και libraries, όπως η Angular, η Vue.js και φυσικά η React, ήρθαν να καλύψουν αυτό το κενό. Προσφέρουν μια δομημένη αρχιτεκτονική, ενισχύουν την επαναχρησιμοποίηση κώδικα και καθιστούν την ανάπτυξη πιο αποδοτική και διατηρήσιμη. Αυτή η αλλαγή στον τρόπο σκέψης και υλοποίησης επηρέασε καθοριστικά την εξέλιξη των web εφαρμογών.

1.3 Η NEXT και η σημασία του στο σύγχρονο web development

Η Next είναι ένα framework που βασίζεται στη React και εισήχθη για πρώτη φορά το 2016 από την εταιρεία Vercel [4]. Σχεδιάστηκε για να διευκολύνει την ανάπτυξη σύγχρονων web εφαρμογών με έμφαση στην απόδοση, τη βελτιστοποίηση SEO και την εμπειρία χρήστη. Παρέχει σημαντικές δυνατότητες όπως server-side rendering, static site generation, dynamic routing και υποστήριξη για API routes. Με τις νεότερες εκδόσεις της, και ιδιαίτερα με την έκδοση 14, η Next ενσωματώνει βελτιώσεις που καθιστούν την ανάπτυξη πιο ευέλικτη, πιο γρήγορη και πιο ισχυρή. Για εφαρμογές e-commerce, όπως αυτή της παρούσας εργασίας, η Next προσφέρει όλα τα απαραίτητα εργαλεία για τη δημιουργία μιας γρήγορης και πλήρως παραμετροποιήσης εμπειρίας του χρήστη.

1.4 Headless CMS: Έννοια και πλεονεκτήματα

Σε ένα παραδοσιακό σύστημα διαχείρισης περιεχομένου (CMS) όπως το WordPress, το περιεχόμενο και η εμφάνισή του είναι "δεμένα" μεταξύ τους. Δηλαδή, ο τρόπος που γράφεις, αποθηκεύεις και αναδεικνύεις τις πληροφορίες γίνεται μέσα από το ίδιο περιβάλλον, κάτι που περιορίζει την ευελιξία όταν θέλεις να αλλάξεις το frontend ή να αναδείξεις το περιεχόμενο σε διαφορετικές πλατφόρμες (όπως mobile apps ή άλλες συσκευές).



Εικόνα 1.2: Διαφορά του CMS με το Headless CMS

[https://bluewhaleapps.com/blog/headless-vs-traditional-cms-government-private-sector]

Το Headless CMS έρχεται να λύσει αυτό το πρόβλημα. Στην ουσία, "κόβει" το frontend (το λεγόμενο "κεφάλι") από το backend και προσφέρει μόνο τα δεδομένα, συνήθως μέσω API (όπως REST ή GraphQL). Έτσι, οι προγραμματιστές έχουν τη δυνατότητα να δημιουργήσουν την εμφάνιση του site ή της εφαρμογής χρησιμοποιώντας ό,τι τεχνολογία θέλουν — για παράδειγμα React ή NEXT — και να παίρνουν μόνο τα απαραίτητα δεδομένα από το CMS [5].

1.5 Υποδομή και φιλοξενία εφαρμογής

Η ολοκληρωμένη εφαρμογή headless e-commerce φιλοξενείται σε cloud server της Hetzner, ένας αξιόπιστος πάροχος με δυνατότητες υψηλής απόδοσης και ευελιξία στη διαχείριση πόρων. Η χρήση δικού μας server, αντί για κάποιο managed hosting, επιλέχθηκε για μεγαλύτερο έλεγχο, καλύτερη απόδοση και δυνατότητα παραμετροποίησης σύμφωνα με τις ανάγκες της εφαρμογής. Μέσα από αυτή την προσέγγιση έγινε επίσης δυνατή η διαχείριση του περιβάλλοντος ανάπτυξης, της βάσης δεδομένων, του WordPress backend και της NEXT εφαρμογής σε ενιαίο οικοσύστημα.

Επιπλέον, η εφαρμογή έχει συνδεθεί με domain name καταχωρημένο μέσω της πλατφόρμας Namecheap, ενώ το frontend της έχει γίνει deploy στη Vercel, αξιοποιώντας τα πλεονεκτήματα του αυτόματου build που προσφέρει. Έτσι, επιτυγχάνεται γρήγορη απόκριση, ευκολία στη διαχείριση και υψηλή διαθεσιμότητα του frontend κομματιού της εφαρμογής.[6]

1.6 Σκοπός της εργασίας και υλοποίηση

Ο βασικός στόχος της παρούσας εργασίας είναι η δημιουργία μιας σύγχρονης web εφαρμογής ηλεκτρονικού εμπορίου, η οποία υλοποιεί μοντέρνες τεχνολογίες και αρχιτεκτονικές, όπως το Headless CMS και το framework NEXT. Η επιλογή αυτών των εργαλείων δεν έγινε τυχαία· αντιθέτως, βασίστηκε στην ανάγκη για ευελιξία, υψηλή απόδοση, καλύτερη εμπειρία χρήστη και δυνατότητα μελλοντικής επέκτασης.

Πιο συγκεκριμένα, μέσα από αυτή την εργασία επιδιώκεται να καλυφθούν δύο βασικά ζητήματα: αφενός η τεχνική κατανόηση και εφαρμογή σύγχρονων τεχνολογιών ανάπτυξης web εφαρμογών, και αφετέρου η πρακτική δημιουργία μιας headless e-commerce λύσης, η οποία μπορεί να προσαρμοστεί εύκολα σε διάφορα σενάρια χρήσης.

Η εφαρμογή που αναπτύχθηκε έχει τις εξής βασικές δυνατότητες και χαρακτηριστικά:

- Το frontend υλοποιείται με NEXT (v14), ένα ισχυρό framework βασισμένο στη React, το οποίο υποστηρίζει server-side rendering, static site generation και δυναμικό routing.

- Ως σύστημα διαχείρισης περιεχομένου (CMS) χρησιμοποιείται το WordPress, το οποίο λειτουργεί σε headless μορφή, παρέχοντας μόνο τα δεδομένα μέσω REST API, χωρίς να εμπλέκεται στην εμφάνιση.

- Η επικοινωνία μεταξύ WordPress και NEXT γίνεται μέσω REST API, επιτρέποντας την ευελιξία στην παρουσίαση του περιεχομένου και την πλήρη ανεξαρτησία του frontend.

- Η εφαρμογή έχει αναπτυχθεί και φιλοξενείται σε ιδιόκτητο cloud server της Hetzner, με στόχο την πλήρη παραμετροποίηση και τον έλεγχο του περιβάλλοντος φιλοξενίας.

- Το domain της εφαρμογής έχει κατοχυρωθεί μέσω της πλατφόρμας Namecheap, ενώ το frontend έχει γίνει deploy στη Vercel, αξιοποιώντας τα πλεονεκτήματα του αυτόματου build και της ταχύτητας φόρτωσης.

Επιπλέον, έχει υλοποιηθεί σύστημα ασφαλούς ταυτοποίησης χρηστών, βασισμένο στο NextAuth v5, με υποστήριξη για Google OAuth σύνδεση.[7] Μέσω αυτής της υλοποίησης, οι χρήστες μπορούν να συνδέονται στην εφαρμογή χρησιμοποιώντας τον Google λογαριασμό τους, με ασφάλεια και ταχύτητα. Το NextAuth διαχειρίζεται τα authentication sessions και χρησιμοποιεί JWT tokens, που δημιουργούνται και πιστοποιούνται μέσω του JWT Authentication plugin στο WordPress[8]. Με αυτό τον τρόπο, επιτυγχάνεται πλήρης ενοποίηση του authentication flow μεταξύ frontend και backend, χωρίς να χρειάζεται παραδοσιακός μηχανισμός διαχείρισης χρηστών από το WordPress, παραμένοντας πιστοί στη headless προσέγγιση.

Από λειτουργική άποψη, η εφαρμογή υποστηρίζει προβολή προϊόντων, δυναμικό καλάθι αγορών, φιλτράρισμα προϊόντων ανά κατηγορία και σχεδιάζεται να επεκταθεί με login σύστημα, σύστημα παραγγελιών και πληρωμών.

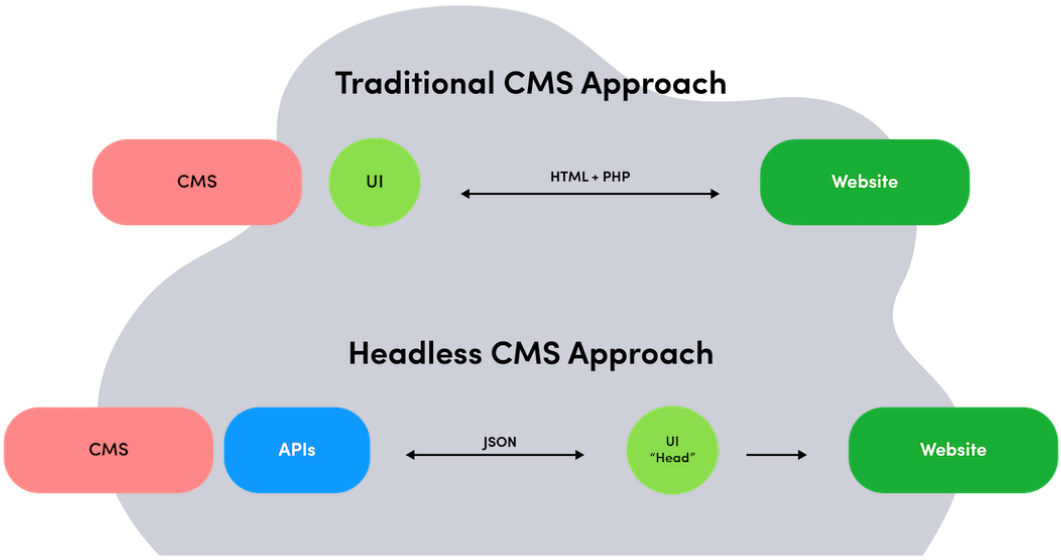
Μέσα από αυτή την υλοποίηση, επιχειρείται να αναδειχθεί πώς η χρήση σύγχρονων εργαλείων μπορεί να προσφέρει έναν πιο ευέλικτο, μοντέρνο και αποδοτικό τρόπο ανάπτυξης εφαρμογών. Παράλληλα, δίνεται η δυνατότητα κατανόησης των αρχών του headless web development, της αρχιτεκτονικής διαχωρισμού περιεχομένου και παρουσίασης, και της αξιοποίησης cloud-based υποδομών.

Τελικός στόχος είναι να παρουσιαστεί μια ολοκληρωμένη λύση e-commerce, η οποία να είναι λειτουργική, επεκτάσιμη και πλήρως ευθυγραμμισμένη με τις απαιτήσεις της σύγχρονης αγοράς ανάπτυξης web εφαρμογών.

Κεφάλαιο 2ο: Ανάλυση και σχεδίαση της εφαρμογής

Στο προηγούμενο κεφάλαιο παρουσιάστηκαν οι βασικές τεχνολογίες και η φιλοσοφία που διέπει την ανάπτυξη σύγχρονων web εφαρμογών, με έμφαση στο Headless CMS, τη NEXT και την αρχιτεκτονική διαχωρισμού frontend και backend. Σε αυτό το δεύτερο κεφάλαιο, θα εστιάσουμε στην ανάλυση της αρχιτεκτονικής της εφαρμογής που αναπτύχθηκε στο πλαίσιο αυτής της εργασίας.

Θα περιγράψουμε τον τρόπο με τον οποίο συνδυάζονται τα επιμέρους τεχνολογικά κομμάτια από τη δομή του frontend και του backend, έως την επικοινωνία τους μέσω API, τη διαχείριση αυθεντικοποίησης και τον τρόπο φιλοξενίας. Η επιλογή κάθε εργαλείου και υπηρεσίας έγινε με γνώμονα τη λειτουργικότητα, την απόδοση και τη μελλοντική επεκτασιμότητα του συστήματος. Σκοπός του κεφαλαίου είναι να αποτυπώσει με σαφήνεια πώς υλοποιήθηκε η συνολική λύση και να δώσει μια πλήρη εικόνα της αρχιτεκτονικής της εφαρμογής.



Εικόνα 2.1: Αρχιτεκτονική διαφορά του CMS και του Headless CMS

[https://buttercms.com/blog/understanding-headless-architecture]

2.1 Αρχιτεκτονική συστήματος

Η εφαρμογή που αναπτύχθηκε βασίζεται σε μια σύγχρονη αρχιτεκτονική τύπου headless, η οποία διαχωρίζει πλήρως την παρουσίαση του περιεχομένου (frontend) από τη διαχείρισή του (backend). Ο βασικός στόχος αυτής της προσέγγισης είναι να προσφέρει μεγαλύτερη ευελιξία, καλύτερη απόδοση,

ανεξαρτησία μεταξύ των τεχνολογιών που χρησιμοποιούνται και δυνατότητα εύκολης επέκτασης στο μέλλον. Η συγκεκριμένη αρχιτεκτονική ακολουθεί τη λογική της αποκέντρωσης των ρόλων, επιτρέποντας σε κάθε μέρος του συστήματος να κάνει αυτό που γνωρίζει καλύτερα.

Το frontend της εφαρμογής έχει υλοποιηθεί με τη NEXT, ένα δημοφιλές framework βασισμένο στη React, το οποίο προσφέρει εργαλεία για βελτιστοποίηση της απόδοσης, υποστήριξη για SSR, SSG και ευέλικτο routing. Από την άλλη πλευρά, το backend στηρίζεται σε μια headless εγκατάσταση του WordPress. Το WordPress εδώ χρησιμοποιείται καθαρά για την εισαγωγή, αποθήκευση και διαχείριση του περιεχομένου, χωρίς να εμπλέκεται καθόλου στην προβολή του, καθώς αυτό είναι αποκλειστική ευθύνη της NEXT.

Η επικοινωνία μεταξύ των δύο μερών γίνεται μέσω REST API, με το WordPress να προσφέρει τα δεδομένα σε μορφή JSON και η NEXT να αναλαμβάνει την απόδοσή τους με τον τρόπο που εμείς ορίζουμε. Αυτός ο τρόπος επιτρέπει μεγαλύτερο έλεγχο στον προγραμματιστή και μεγαλύτερη ελευθερία στον σχεδιασμό του frontend, ανεξάρτητα από το πώς λειτουργεί ή είναι δομημένο το backend. Παράλληλα, η εφαρμογή κάνει χρήση είτε του SSR είτε του SSG ανάλογα με τις ανάγκες κάθε σελίδας, ώστε να επιτυγχάνει τη μέγιστη απόδοση και καλύτερη εμπειρία χρήστη, χωρίς να θυσιάζει την ταχύτητα ή την ευελιξία.

Ένα άλλο σημαντικό κομμάτι της αρχιτεκτονικής είναι η διαχείριση της αυθεντικοποίησης των χρηστών. Για τον σκοπό αυτό χρησιμοποιείται το NextAuth.js (έκδοση 5), το οποίο παρέχει υποστήριξη για authentication μέσω OAuth παρόχων – στην περίπτωσή μας, με χρήση του Google OAuth 2.0. Η διαδικασία ταυτοποίησης ολοκληρώνεται με τη χρήση JWT (JSON Web Tokens), τα οποία διαχειρίζεται η NextAuth και μπορούν να αξιοποιηθούν τόσο από το frontend όσο και από το backend για έλεγχο ταυτότητας. Στο backend έχει εγκατασταθεί και το κατάλληλο plugin για JWT authentication στο WordPress, έτσι ώστε το API να μπορεί να αναγνωρίσει επαληθευμένους χρήστες και να δίνει πρόσβαση σε προστατευμένα δεδομένα.

Η υποδομή φιλοξενίας έχει δομηθεί με στόχο την πλήρη αυτονομία και έλεγχο. Το backend (WordPress, MariaDB, PHP και Nginx) είναι εγκατεστημένο σε cloud server της Hetzner, που προσφέρει ισχυρές δυνατότητες σε σχέση με την απόδοση, την παραμετροποίηση και την ασφάλεια. Με αυτόν τον τρόπο, ο προγραμματιστής έχει την πλήρη διαχείριση του περιβάλλοντος, των ρυθμίσεων και της ασφάλειας του backend συστήματος[9]. Αντίστοιχα, το frontend κομμάτι της εφαρμογής έχει αναπτυχθεί και γίνει deploy στη Vercel, η οποία προσφέρει σύγχρονες δυνατότητες για συνεχή ενσωμάτωση, βελτιστοποίηση των build, caching και δυναμική απόδοση των σελίδων.

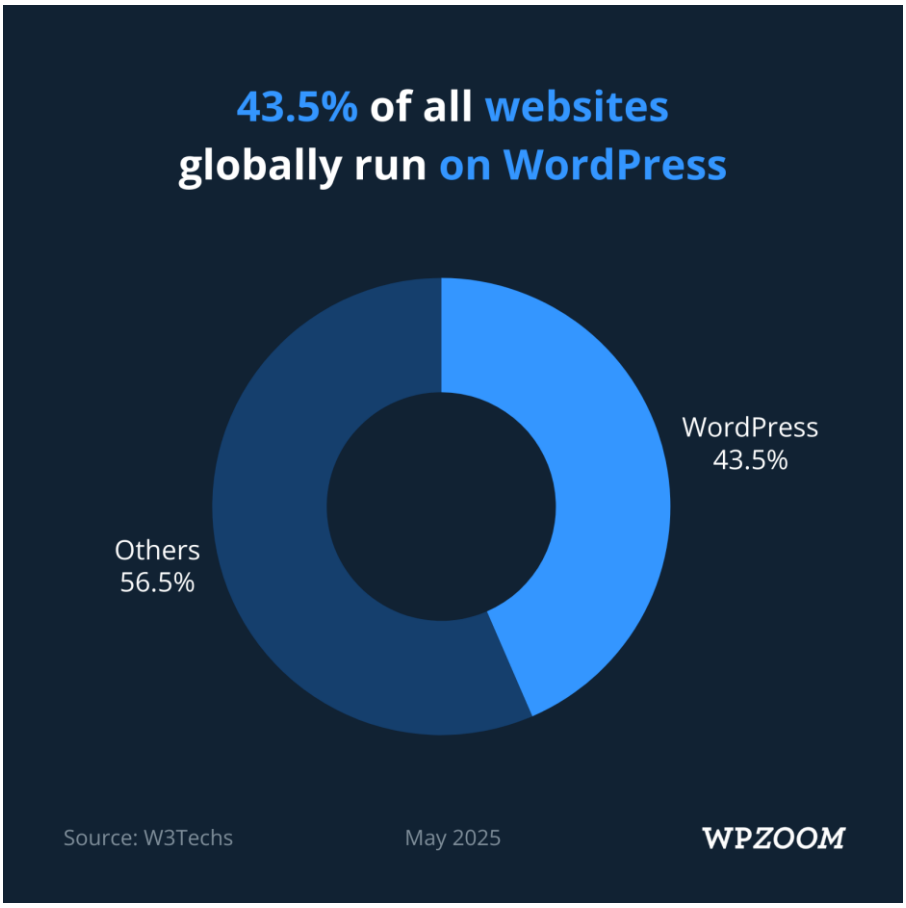
Τέλος, η σύνδεση με το domain της εφαρμογής γίνεται μέσω Namecheap, ενώ η Vercel χειρίζεται τη δημόσια πρόσβαση στο frontend. Έτσι, ολόκληρη η αρχιτεκτονική παραμένει διαχωρισμένη, ευέλικτη και απόλυτα προσαρμοσμένη στις ανάγκες μιας μοντέρνας headless e-commerce εφαρμογής.

2.2 Επιλογή τεχνολογιών

Η επιλογή τεχνολογιών για την ανάπτυξη της συγκεκριμένης headless e-commerce εφαρμογής δεν έγινε τυχαία, αλλά στηρίχθηκε σε συγκεκριμένα κριτήρια, όπως η απόδοση, η ευελιξία και η δυνατότητα μελλοντικής επέκτασης.

Η επιλογή της NEXT ως κύριο framework για το frontend βασίστηκε στην ανάγκη για γρήγορο, δυναμικό και καλά βελτιστοποιημένο rendering. Η έκδοση 14 της NEXT ήταν η πιο κατάλληλη, καθώς φέρνει σημαντικές βελτιώσεις τόσο σε επίπεδο server components όσο και στη γενικότερη διαχείριση των routes και των data fetching λειτουργιών.[10] Επιπλέον, παρέχει για dynamic rendering, static site generation και εύκολη διασύνδεση με API, κάτι που είναι κρίσιμο για μια headless προσέγγιση. Η δυνατότητα να ορίζονται server actions, να διαχειρίζεσαι cookies χωρίς να υποβαθμίζεται το rendering σε dynamic, καθώς και η σταδιακή υλοποίηση του App Router ως standard routing σύστημα, ενίσχυσαν την απόφαση να χρησιμοποιηθεί μία από τις πιο πρόσφατες εκδόσεις της NEXT.

Το WordPress χρησιμοποιήθηκε ως Headless CMS, κυρίως λόγω των πολλών έτοιμων τεχνολογιών του και της μεγάλης ποικιλίας εργαλείων και plugins που προσφέρει, με πιο χαρακτηριστικό το WooCommerce για τη διαχείριση των προϊόντων και των παραγγελιών. Η λογική του headless WordPress επιτρέπει τη χρήση του μόνο ως back-end, χωρίς να δεσμεύει την παρουσίαση ή την αρχιτεκτονική του frontend. Έτσι, το περιεχόμενο διαχειρίζεται από το WordPress, αλλά παρουσιάζεται και καταναλώνεται από τη NEXT, προσφέροντας πλήρη ευελιξία στον σχεδιασμό της εμπειρίας χρήστη. Επίσης στατιστικές μελέτες έχουν δείξει ότι το CMS του WordPress χρησιμοποιείται παγκοσμίως στις ιστοσελίδες με 43.5% (εικόνα 2.2).



Εικόνα 2.2: Ποσοστό ιστοσελίδων που χρησιμοποιούν WordPress

[https://www.wpzoom.com/blog/wordpress-statistics]

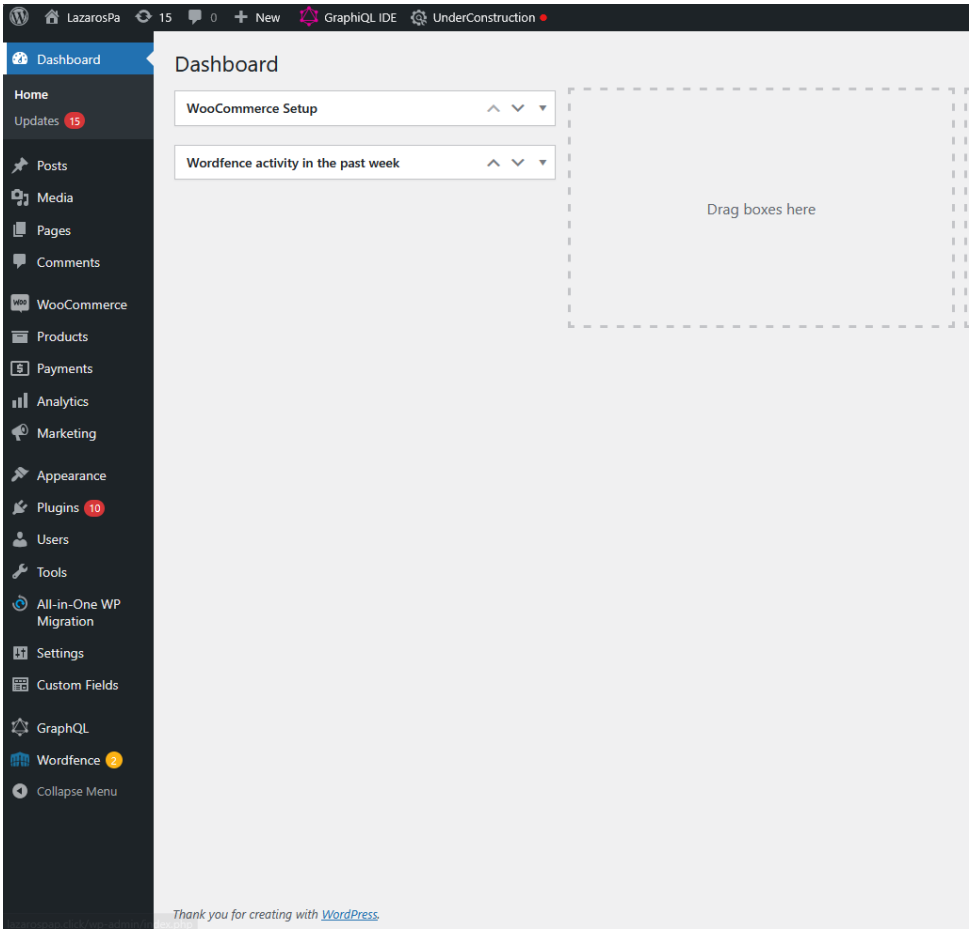
Για την επικοινωνία μεταξύ frontend και backend, χρησιμοποιήθηκαν δύο διαφορετικές τεχνολογίες. Συγκεκριμένα, αξιοποιήθηκε το REST API του WooCommerce για την υλοποίηση του checkout και των παραγγελιών, καθώς αυτή είναι η προεπιλεγμένη και πλήρως υποστηριζόμενη λύση από την πλευρά του WordPress. Η αξιοπιστία του REST, η ώριμη τεκμηρίωση και η σταθερότητά του για λειτουργίες κρίσιμες, όπως η αποστολή παραγγελιών, το καθιστούν ιδανική επιλογή.[11]

Παράλληλα, για την προβολή και κατανάλωση των δεδομένων του καταλόγου προϊόντων, των φίλτρων και των κατηγοριών, χρησιμοποιήθηκε το GraphQL σε συνδυασμό με τον Apollo Client. Το GraphQL προσφέρει την ευελιξία να ζητούνται ακριβώς τα δεδομένα που χρειάζονται, μειώνοντας το βάρος στο δίκτυο και βελτιώνοντας την απόδοση. Ο Apollo Client υποστηρίζει caching, optimistic UI updates και διαχείριση των local states, στοιχεία που προσφέρουν πιο διαδραστική και ομαλή εμπειρία στον τελικό χρήστη.[12]

Το περιβάλλον φιλοξενίας χωρίστηκε σε δύο μέρη. Το frontend έχει γίνει deploy στη Vercel, την επίσημη πλατφόρμα της NEXT, η οποία παρέχει αυτόματα build, preview deployments και

βελτιστοποιήσεις απόδοσης χωρίς επιπλέον ρυθμίσεις. Η Vercel ενδείκνυται για εφαρμογές NEXT, ειδικά όταν απαιτείται συχνό deployment και σταθερή απόδοση.

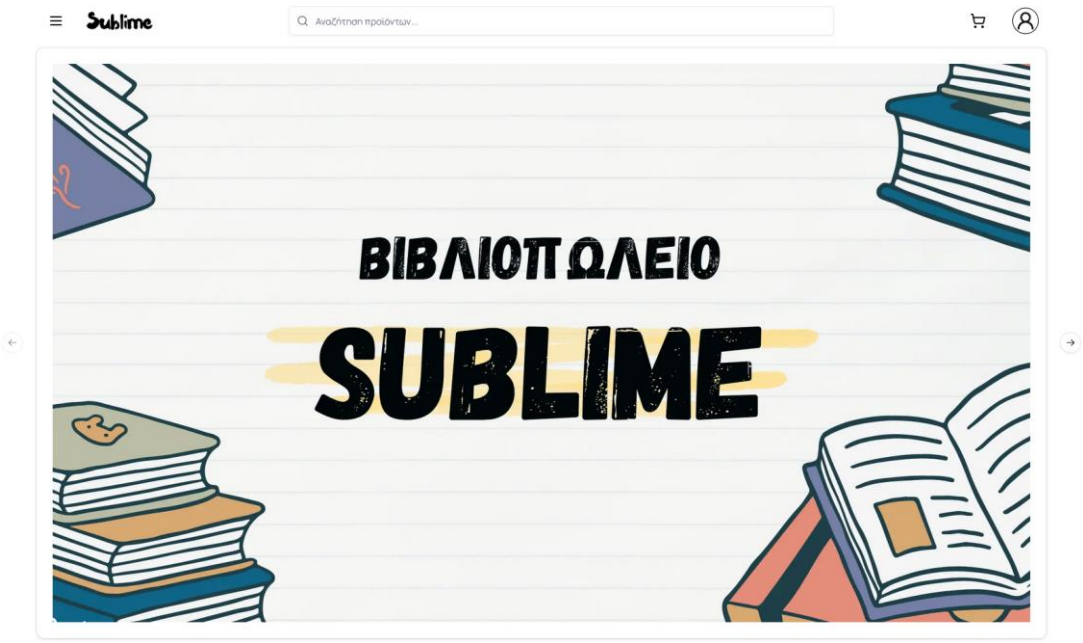
Από την άλλη πλευρά, το backend, δηλαδή το WordPress μαζί με τη βάση δεδομένων και το WooCommerce, φιλοξενείται σε ιδιόκτητο cloud server της Hetzner. Η επιλογή του Hetzner αντί κάποιου managed hosting provider έγινε με βάση την ανάγκη για μεγαλύτερο έλεγχο, ευελιξία παραμετροποίησης, αλλά και χαμηλότερο λειτουργικό κόστος. Μέσα από αυτή την προσέγγιση υπήρξε η δυνατότητα να στηθεί ακριβώς το περιβάλλον που απαιτείται, να γίνουν βελτιστοποιήσεις σε επίπεδο nginx, PHP και caching, και να υπάρχει πλήρης έλεγχος σε επίπεδο διαχείρισης και ασφάλειας. Το διαχειριστικό κομμάτι του WordPress σύμφωνα με την (εικόνα 2.3) εμφανίζεται με αυτό τον τρόπο δίνοντας την ευκολία διαχείρισης των plugins, της GraphQl καθώς και πολλά άλλα προνόμια διαχειρήσης.



Εικόνα 2.3 Πίνακας ελέγχου του WordPress

Η υλοποίηση του authentication συστήματος βασίζεται στο NextAuth v5, την πλέον σύγχρονη έκδοση της γνωστής βιβλιοθήκης ταυτοποίησης για NEXT. Επιλέχθηκε η μέθοδος ταυτοποίησης μέσω Google OAuth, καθώς παρέχει ασφάλεια, ταχύτητα και ευχρηστία για τον τελικό χρήστη. Παράλληλα, υλοποιήθηκε διασύνδεση με το WordPress backend μέσω JWT authentication plugin, έτσι ώστε τα δεδομένα του χρήστη να μπορούν να διασταυρώνονται και με το backend όταν χρειάζεται, διατηρώντας παράλληλα την ανεξαρτησία του frontend.

Οι τεχνολογίες που επιλέχθηκαν προσφέρουν πληθώρα πλεονεκτημάτων, όπως ταχύτητα, επεκτασιμότητα και καλύτερη εμπειρία χρήστη. Ωστόσο, δεν λείπουν και οι προκλήσεις: η διαχείριση ενός custom headless περιβάλλοντος απαιτεί καλή τεχνογνωσία, ιδιαίτερη προσοχή σε θέματα ασφαλείας, και περισσότερη προσπάθεια στην αρχική εγκατάσταση. .Παρ' όλα αυτά, το τελικό αποτέλεσμα δικαιολογεί την πολυπλοκότητα, προσφέροντας ένα σύγχρονο και στιβαρό τεχνολογικό θεμέλιο (εικόνα 2.4).



Εικόνα 2.4: Αρχική εμφάνιση της ιστοσελίδας

2.3 Υποδομή WordPress και επιλεγμένα plugins

Στο backend, το WordPress λειτουργεί ως η βασική πλατφόρμα διαχείρισης περιεχομένου (CMS), η οποία όμως έχει προσαρμοστεί έτσι ώστε να λειτουργεί αποκλειστικά headless. Για να καταστεί αυτό εφικτό, αξιοποιήθηκε μια σειρά από plugins που προσφέρουν τόσο λειτουργικότητα όσο και τη δυνατότητα διασύνδεσης με το frontend μέσω APIs. Τα βασικά plugins που χρησιμοποιήθηκαν είναι τα εξής:

- **Advanced Custom Fields (ACF)**: Το plugin αυτό χρησιμοποιείται για να εμπλουτίσουμε τις προεπιλεγμένες δυνατότητες του WordPress με custom fields, επιτρέποντάς μας να προσθέσουμε πιο δομημένα και σύνθετα δεδομένα στα προϊόντα και στις σελίδες. Είναι απαραίτητο όταν χρειαζόμαστε π.χ. custom labels, εσωτερικούς κωδικούς, media galleries κ.ά.

- **JWT Authentication for WP-API**: Αυτό το plugin χρησιμοποιήθηκε για να διασφαλίσουμε ασφαλή επικοινωνία μεταξύ του frontend και του WordPress backend. Μέσω JWT, μπορούμε να κάνουμε authenticated requests στο WordPress API, π.χ. για να δημιουργήσουμε παραγγελίες ή να αποθηκεύσουμε δεδομένα που απαιτούν login.

- **WooCommerce**: Αποτελεί την καρδιά της e-commerce λειτουργικότητας. Παρότι το frontend υλοποιήθηκε εξ ολοκλήρου σε NEXT, το WooCommerce παραμένει υπεύθυνο για τη διαχείριση προϊόντων, καλαθιών, πληρωμών και παραγγελιών, μέσω του REST API του.

- **WPGraphQL**: Είναι το βασικό plugin που επιτρέπει την πρόσβαση σε δεδομένα του WordPress μέσω GraphQL queries. Χάρη σε αυτό, μπορούμε να αντλούμε δυναμικά περιεχόμενο όπως προϊόντα, σελίδες, κατηγορίες, φίλτρα κ.ά., σε πραγματικό χρόνο από το frontend, χωρίς να φορτώνουμε το σύστημα με δεδομένα που δε χρειαζόμαστε.

- **WPGraphQL for Advanced Custom Fields**: Επεκτείνει το WPGraphQL ώστε να υποστηρίζει και τα custom fields που έχουν δημιουργηθεί με το ACF. Έτσι μπορούμε να διαβάζουμε τα ειδικά πεδία των προϊόντων ή των σελίδων μέσω GraphQL, με τον ίδιο τρόπο που διαβάζουμε τα υπόλοιπα δεδομένα.

- **WPGraphQL WooCommerce (WooGraphQL)**: Πρόκειται για επέκταση του WPGraphQL που φέρνει τη λειτουργικότητα του WooCommerce (προϊόντα, παραλλαγές, καρότσι, κ.λπ.) μέσα στον κόσμο του GraphQL. Χωρίς αυτό το plugin, η πλειοψηφία των δεδομένων του WooCommerce θα ήταν προσβάσιμη μόνο μέσω REST API.

- **WPS Hide Login**: Πρόκειται για ένα απλό αλλά σημαντικό plugin που επιτρέπει την αλλαγή της default διαδρομής σύνδεσης (/wp-admin) σε μια custom, αυξάνοντας την ασφάλεια του backend και μειώνοντας την έκθεση σε brute-force επιθέσεις. Αυτό το plugin βοηθάει μόνο για το διαχειριστικό του WordPress όπου μπορεί να προσπαθήσει κάποιος να εισβάλει.

Όλα αυτά τα plugins συνεργάζονται μεταξύ τους ώστε να μετατρέψουν το WordPress σε ένα ευέλικτο, πλήρως headless backend, ικανό να υποστηρίξει μια μοντέρνα, δυναμική εφαρμογή γραμμένη εξ ολοκλήρου στο frontend με NEXT. Η σωστή επιλογή και ρύθμιση των εργαλείων

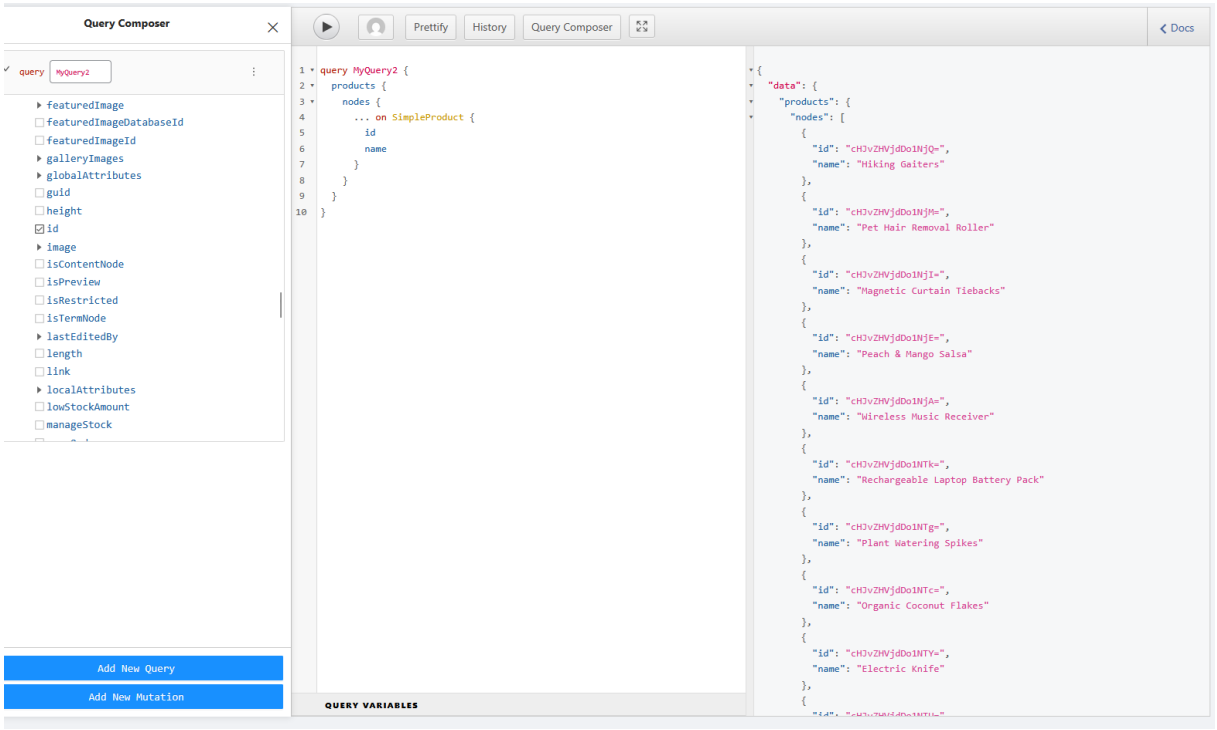
αυτών ήταν καθοριστική για την επιτυχία της αρχιτεκτονικής αυτής. Αυτά είναι τα βασικά plugins που μας είναι χρήσιμα για να μπορούμε γεφυρώσουμε την NEXT με το WordPress, διότι όλα τα υπόλοιπα πέρα από την διαχείριση των παραγγελιών σχεδιάζονται και υλοποιούνται προγραμματιστικά.

2.4 Αρχιτεκτονική συστήματος και σύνδεση Frontend - Backend

Η αρχιτεκτονική της εφαρμογής βασίστηκε σε μια καθαρά headless προσέγγιση, όπου το frontend (NEXT) λειτουργεί ανεξάρτητα από το backend (WordPress). Ο διαχωρισμός αυτός επέτρεψε τη μεγαλύτερη ευελιξία στην ανάπτυξη, τη βελτίωση της απόδοσης και τη δυνατότητα ελέγχου κάθε πτυχής του τρόπου με τον οποίο παρουσιάζονται και διαχειρίζονται τα δεδομένα.

Η επικοινωνία μεταξύ του NEXT και του WordPress backend επιτυγχάνεται με δύο τρόπους: μέσω REST API και μέσω GraphQL. Το REST API χρησιμοποιήθηκε κυρίως για λειτουργίες που σχετίζονται με το WooCommerce, όπως για παράδειγμα η ολοκλήρωση της παραγγελίας (checkout), ενώ η GraphQL αξιοποιήθηκε για την ανάκτηση περιεχομένου και δυναμικών πεδίων από custom post types και προϊόντα. Ο συνδυασμός των δύο αυτών μεθόδων παρείχε την απαραίτητη ισορροπία μεταξύ απόδοσης και ευελιξίας, επιτρέποντας στον client (NEXT) να ζητά ακριβώς τα δεδομένα που χρειάζεται.

Για την επικοινωνία μέσω GraphQL, χρησιμοποιήθηκε ο Apollo Client, ένα δημοφιλές εργαλείο διαχείρισης state και δεδομένων στον κόσμο του React. Μέσω αυτού, δημιουργήθηκαν queries που "χτυπούν" το WPGraphQL API του WordPress για να αντλήσουν δυναμικά δεδομένα, όπως κατηγορίες, προϊόντα και περιγραφές. Ταυτόχρονα, το REST API αξιοποιήθηκε για την δημιουργία παραγγελίας διότι δεν υπάρχει επιλογή στην GraphQL να περιμένει πληρωμή με κάρτα. Οπότε έγινε ένα API route που εξασφάλισε την πληρωμή και με κάρτα. Με το GraphQL playground μπορούμε να δοκιμάζουμε όλες τις πληροφορίες που μας δίνει το WordPress για να μπορέσουμε να τις αντλήσουμε με το Apollo για την εφαρμογή μας.



Εικόνα 2.5: GraphQL playground για queries και mutations

Η authentication αρχιτεκτονική αποτελεί ένα ακόμα σημαντικό κομμάτι της διασύνδεσης. Η εφαρμογή χρησιμοποιεί το NextAuth v5, σε συνδυασμό με Google OAuth για login και το plugin JWT Authentication for WP-API στο WordPress. Αυτό επιτρέπει στον χρήστη να συνδέεται με τον Google λογαριασμό του στο frontend και, στη συνέχεια, να πραγματοποιείται ασφαλής επικοινωνία με το WordPress backend μέσω JWT tokens. Το token αυτό επισυνάπτεται στα αιτήματα (requests) προς προστατευμένα endpoints, επιτρέποντας τη διασφάλιση των δικαιωμάτων πρόσβασης.

Συνολικά, η σύνδεση frontend και backend επιτεύχθηκε με τρόπο modular, καθαρό και ευέλικτο. Κάθε πλευρά του συστήματος λειτουργεί αυτόνομα αλλά συνεργατικά, δημιουργώντας μια υποδομή που μπορεί να επεκταθεί εύκολα, να προσαρμοστεί σε νέες ανάγκες και να προσφέρει μια ομαλή και ασφαλή εμπειρία χρήσης.

2.5 Δημιουργία Hetzner Cloud

Σε αυτό το κεφάλαιο περιγράφεται η πλήρης διαδικασία εγκατάστασης ενός WordPress περιβάλλοντος σε cloud server της Hetzner, με λειτουργικό σύστημα Ubuntu, χρήση του Nginx ως web server, MariaDB ως σύστημα διαχείρισης βάσης δεδομένων, και phpMyAdmin ως γραφικό περιβάλλον για διαχείριση της βάσης.

Βήμα 1: Δημιουργία Server στο Hetzner Cloud

1. Πραγματοποιούμε είσοδο στο https://console.hetzner.cloud ή σε WSL(Windows Subsystem for Linux), το οποίο μας δίνει τη δυνατότητα να εκτελείτε ένα σύστημα αρχείων Linux, μαζί με εργαλεία γραμμής εντολών Linux και εφαρμογές απευθείας στα Windows, παράλληλα με την παραδοσιακή επιφάνεια εργασίας και τις εφαρμογές των Windows.

2. Δημιουργούμε νέο project και στη συνέχεια έναν νέο server:

- Image: Ubuntu 22.04 LTS
- Location: Ευρώπη ή κοντά στον τελικό χρήστη
- -Type: CX21 ή μεγαλύτερο
- -SSH Key: Προσθήκη ή δημιουργία νέου

Βήμα 2: Σύνδεση στον Server μέσω SSH

```
ssh root@your_server_ip
```

Συνδεόμαστε απομακρυσμένα στον server μέσω πρωτοκόλλου SSH, χρησιμοποιώντας το username root και τη δημόσια IP του Hetzner server. Πρόκειται για το βασικό βήμα διαχείρισης του server από τοπικό υπολογιστή.

Βήμα 3: Ενημέρωση Συστήματος

```
apt update && apt upgrade -y
```

Ενημερώνουμε τη λίστα των πακέτων και αναβαθμίζουμε το σύστημα στην τελευταία διαθέσιμη έκδοση. Είναι σημαντικό για λόγους ασφάλειας και σταθερότητας.

Βήμα 4: Εγκατάσταση Nginx

```
apt install nginx -y

systemctl enable nginx

systemctl start nginx
```

Εγκαθίσταται ο web server Nginx, ενεργοποιείται ώστε να ξεκινά αυτόματα με κάθε επανεκκίνηση (enable) και στη συνέχεια ξεκινά η υπηρεσία (start). Ο Nginx θα χειρίζεται όλα τα αιτήματα HTTP για το WordPress.

Βήμα 5: Εγκατάσταση MariaDB

```
apt install mariadb-server -y

systemctl enable mariadb

systemctl start mariadb

mysql_secure_installation
```

Η MariaDB είναι η βάση δεδομένων που θα χρησιμοποιήσει το WordPress. Μετά την εγκατάσταση, ενεργοποιείται η υπηρεσία και με την εντολή **mysql_secure_installation** προσθέτουμε βασική ασφάλεια (όπως root password, απενεργοποίηση ανώνυμων χρηστών).

Βήμα 6: Δημιουργία Βάσης Δεδομένων

```
mysql -u root -p

CREATE DATABASE wordpress_db DEFAULT CHARACTER SET utf8 COLLATE utf8_unicode_ci;

CREATE USER 'wordpress_user'@'localhost' IDENTIFIED BY 'your_password';

GRANT ALL PRIVILEGES ON wordpress_db.* TO 'wordpress_user'@'localhost';

FLUSH PRIVILEGES;

EXIT;
```

Δημιουργείται μια νέα βάση δεδομένων π.χ.(wordpress_db) και ένας χρήστης π.χ.(wordpress_user) με δικαιώματα πλήρους πρόσβασης. Αυτά τα στοιχεία θα χρησιμοποιηθούν στο αρχείο wp-config.php για τη σύνδεση του WordPress με τη βάση δεδομένων.

Βήμα 7: Εγκατάσταση PHP & Modules

```
apt install php-fpm php-mysql php-curl php-gd php-mbstring php-xml php-xmlrpc php-soap php-intl php-zip -y
```

Εγκαθίστανται η PHP και όλα τα modules που απαιτούνται από το WordPress για λειτουργίες όπως επεξεργασία εικόνων, φόρμες, σύνδεση με βάσεις δεδομένων, SOAP, XML, και διαχείριση συμπιεσμένων αρχείων.

Βήμα 8: Κατέβασμα και Εγκατάσταση WordPress

cd /var/www/

```
wget https://wordpress.org/latest.tar.gz

tar -xvzf latest.tar.gz

chown -R www-data:www-data wordpress

chmod -R 755 wordpress
```

Κατεβάζουμε την τελευταία έκδοση του WordPress από την επίσημη σελίδα, την αποσυμπιέζουμε και ρυθμίζουμε σωστά τα δικαιώματα ώστε ο web server να μπορεί να έχει πρόσβαση στα αρχεία.

Βήμα 9: Δημιουργία Virtual Host (Nginx)

nano /etc/nginx/sites-available/wordpress

```
server {
    listen 80;
    server_name your_domain.com;
    root /var/www/wordpress;
    index index.php index.html;
    location / {
        try_files $uri $uri/ /index.php?$args;
    }
    location ~ \.php$ {
        include snippets/fastcgi-php.conf;
        fastcgi_pass unix:/var/run/php/php7.1-fpm.sock;
    }
    location ~ /\.ht {
        deny all;
```

```
    }

}

ln -s /etc/nginx/sites-available/wordpress /etc/nginx/sites-enabled/

nginx -t

systemctl reload nginx
```

Ορίζουμε έναν εικονικό host (virtual host) για να δηλώσουμε ποιο domain εξυπηρετεί ο Nginx και για ποια αρχεία. Ορίζεται η διαδρομή προς τα αρχεία του WordPress, η υποστήριξη για PHP και η ασφάλεια για αρχεία .ht. Ενεργοποιούμε τον virtual host και ελέγχουμε τη σωστή σύνταξη του αρχείου με nginx -t. Τέλος, γίνεται επανεκκίνηση της υπηρεσίας Nginx για να εφαρμοστούν οι αλλαγές.

Βήμα 10: Εγκατάσταση WordPress μέσω UI

```
http://your_server_ip
```

Ανοίγοντας το browser και πληκτρολογώντας http://your_server_ip, ο χρήστης οδηγείται στο γραφικό περιβάλλον εγκατάστασης του WordPress. Εκεί ολοκληρώνεται η διαμόρφωση (όνομα site, admin account, κ.λπ.).

Βήμα 11: Εγκατάσταση phpMyAdmin

```
apt install phpmyadmin -y

ln -s /usr/share/phpmyadmin /var/www/wordpress/phpmyadmin

http://{your_server_ip}/phpmyadmin
```

Εγκαθίσταται το phpMyAdmin, ένα γραφικό περιβάλλον για διαχείριση της βάσης δεδομένων. Ο σύνδεσμος επιτρέπει την πρόσβαση μέσω του URL: http://your_server_ip/phpmyadmin.

Βήμα 12: SSL με Let's Encrypt

```
apt install certbot python3-certbot-nginx -y

certbot --nginx -d your_domain.com -d www.{your_domain.com}
```

Τέλος, προστίθεται δωρεάν SSL πιστοποιητικό με χρήση του Let's Encrypt, μέσω του εργαλείου Certbot. Η εντολή αυτόματα ρυθμίζει τον Nginx ώστε η σελίδα να λειτουργεί με HTTPS.

Με τα παραπάνω βήματα δημιουργήσαμε ένα πλήρως λειτουργικό περιβάλλον φιλοξενίας WordPress πάνω σε Hetzner Cloud.

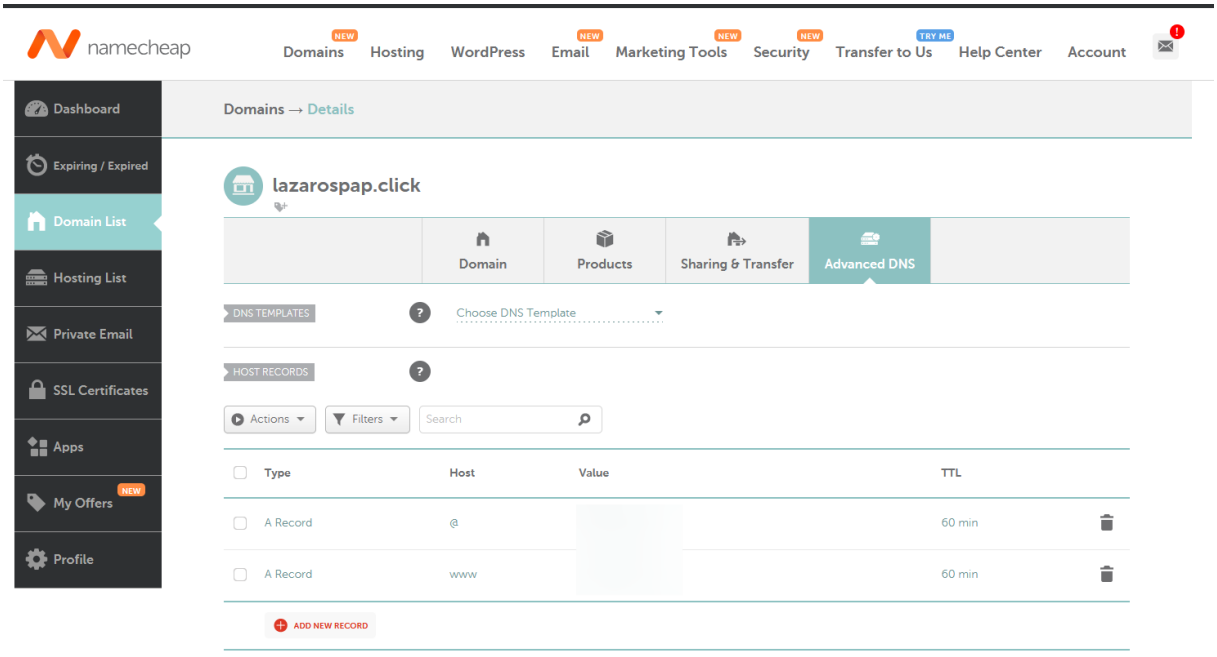
2.6 Υποδομή φιλοξενίας

Επιπλέον, αγοράστηκε domain name (lazarospap.click) για τη φιλοξενία της εφαρμογής και πραγματοποιήθηκε η κατάλληλη ρύθμιση των DNS εγγραφών του. Συγκεκριμένα, μέσω του πίνακα διαχείρισης DNS του παρόχου (Advanced DNS Settings), προστέθηκαν δύο εγγραφές τύπου A Record:

- Μία εγγραφή A με όνομα @, που αντιστοιχεί στο κύριο domain (π.χ example.com), στην δική μας περίπτωση έχουμε βάλει την IP του Server μας.

- Μία δεύτερη A εγγραφή για το www, ώστε και το www.example.com να οδηγεί στον ίδιο server.

Και οι δύο εγγραφές A διαμορφώθηκαν (Εικόνα 2.5) ώστε να δρομολογούνται προς τη δημόσια IP διεύθυνση του Hetzner Cloud Server. Με τον τρόπο αυτό, διασφαλίστηκε ότι κάθε επίσκεψη στο domain (είτε με είτε χωρίς το www) καταλήγει σωστά στην υποδομή φιλοξενίας της εφαρμογής. Οι ρυθμίσεις αυτές πραγματοποιήθηκαν βάσει των οδηγιών της Hetzner.



Εικόνα 2.6: Διαχείριση DNS στο Namecheap

Η πρώτη φάση της υλοποίησης της εφαρμογής ξεκίνησε με τη δημιουργία ενός καθαρού boilerplate project με χρήση του NEXT στην έκδοση 14. Για αυτόν τον σκοπό, χρησιμοποιήθηκε η επίσημη εντολή **npx create-next-app@latest**, μέσα από την οποία δημιουργήθηκε ένας βασικός φάκελος έργου με όλα τα απαραίτητα αρχεία για να ξεκινήσει η ανάπτυξη. Από την αρχή επιλέχθηκε η χρήση του App Router, καθώς προσφέρει καλύτερη οργάνωση και λειτουργικότητα σε μοντέρνα NEXT εφαρμογές.

Μετά την αρχικοποίηση του project, αφαιρέθηκε το default περιεχόμενο και δημιουργήθηκε η βασική δομή φακέλων και αρχείων. Εγκαταστάθηκε το Tailwind CSS ακολουθώντας τις οδηγίες από την επίσημη τεκμηρίωση, ώστε να διαμορφωθεί ένα ευέλικτο και γρήγορο σύστημα styling.

Αφού τέθηκε σε λειτουργία το βασικό περιβάλλον, το επόμενο βήμα ήταν η σύνδεση με το backend. Για την επικοινωνία με το WordPress (που λειτουργεί ως headless CMS με WooCommerce), επιλέχθηκε ο Apollo Client. Το συγκεκριμένο εργαλείο υποστηρίζει GraphQL queries με έναν σύγχρονο και developer-friendly τρόπο.

Ο Apollo Client εγκαταστάθηκε με την εντολή **npm install @apollo/client** και στη συνέχεια δημιουργήθηκε το αρχείο **lib/apolloClient.js** για χρήση σε server components, καθώς και το **lib/ApolloWrapper.js** για χρήση στον client[13]. Ο διαχωρισμός αυτός επιτρέπει τη σωστή υποστήριξη του App Router architecture, δίνοντας τη δυνατότητα χρήσης Apollo queries τόσο στον server όσο και στον client. Επιπλέον, προστέθηκε υποστήριξη για το WooCommerce session cookie, ώστε οι χρήστες να μπορούν να διατηρούν το καλάθι τους.

Παρακάτω παρατίθεται ο αντίστοιχος κώδικας:

lib/apolloClient.js

```
import { ApolloClient, InMemoryCache, HttpLink } from "@apollo/client";

import { registerApolloClient } from "@apollo/experimental-nextjs-app-support";



export const { getClient, query, PreloadQuery } = registerApolloClient(() => {

  return new ApolloClient({

    cache: new InMemoryCache(),

    link: new HttpLink({

      uri: process.env.NEXT_PUBLIC_WORDPRESS_API_URL,

    }),

    defaultOptions: {
```

```
    query: {
      fetchPolicy: "no-cache", // Force fresh fetch
    },
  },
 });
});
```

Ο συγκεκριμένος κώδικας ρυθμίζει τον Apollo Client για χρήση σε μια εφαρμογή Next.js 14 με App Router, ώστε να επικοινωνεί με ένα WordPress backend μέσω GraphQL. Χρησιμοποιεί τη συνάρτηση registerApolloClient από την πειραματική υποστήριξη του Apollo για Next.js, επιτρέποντας τη χρήση του client τόσο σε server όσο και σε client components.

Ο client συνδέεται με το WordPress API μέσω του HttpLink, ενώ η πολιτική fetchPolicy: "no-cache" εξασφαλίζει ότι τα queries φέρνουν πάντα φρέσκα δεδομένα, παρακάμπτοντας το τοπικό cache. Τέλος, επιστρέφονται χρήσιμες βοηθητικές μέθοδοι (getClient, query, PreloadQuery) για την ευέλικτη χρήση του client στην εφαρμογή.

lib/ApolloWrapper.js

```
"use client";
import { ApolloLink, HttpLink } from "@apollo/client";
import {
  ApolloNextAppProvider,
  ApolloClient,
  InMemoryCache,
} from "@apollo/experimental-nextjs-app-support"; // Experimental Apollo support for NEXT App Router
// Define the GraphQL HTTP link to your WordPress (WooCommerce) backend
const httpLink = new HttpLink({
  uri: process.env.NEXT_PUBLIC_WORDPRESS_API_URL, // URL of the WordPress GraphQL API
  fetchOptions: { cache: "no-store" }, // Disable fetch-level caching to always get fresh data
  useGETForQueries: true, // Use GET requests for queries (helps with caching/proxies)
});
// Middleware to attach WooCommerce session token to each outgoing request (if available in cookies)
```

```
const middlewareLink = new ApolloLink((operation, forward) => {
  if (typeof window === "undefined") return forward(operation); // Exit early on the server
  const wooSession = getCookie("woo-session"); // Get the session token from browser cookies
  // If a session exists, attach it as a custom header
  if (wooSession) {
    operation.setContext(({ headers = {} }) => ({
      headers: {
        ...headers,
        "woocommerce-session": `Session ${wooSession}`, // Custom WooCommerce session header
      },
    }));
  }
  return forward(operation); // Pass the operation to the next link
});
// Afterware to capture any updated session token from the response headers and update cookies accordingly
const afterwareLink = new ApolloLink((operation, forward) => {
  return forward(operation).map((response) => {
    if (typeof window === "undefined") return response; // Exit early on the server
    const context = operation.getContext(); // Access response context
  const sessionToken = context.response.headers.get("woocommerce-session"); // Read WooCommerce session from response
    // If the token is "false", clear the cookie
    if (sessionToken) {
      if (sessionToken === "false") {
        setCookie("woo-session", "", { maxAge: -1 }); // Expire the cookie
      } else if (getCookie("woo-session") !== sessionToken) {
        setCookie("woo-session", sessionToken); // Update the cookie with the new session
      }
    }
```

```
    return response;
  });
});
```

Αυτό το αρχείο υλοποιεί τη ρύθμιση του Apollo Client για χρήση στον client side μιας εφαρμογής Next.js 14, αξιοποιώντας την πειραματική υποστήριξη για το App Router. Ο βασικός του στόχος είναι να διαχειρίζεται σωστά τις WooCommerce session πληροφορίες μέσω cookies, ώστε να εξασφαλίζεται μια συνεπής εμπειρία χρήστη στο καλάθι και τις παραγγελίες.

Αρχικά, ορίζεται ένα HttpLink που δείχνει στο WordPress GraphQL API, με cache: "no-store" για να αποφεύγεται η αποθήκευση δεδομένων από προηγούμενα queries. Στη συνέχεια, προστίθεται ένα middleware που ελέγχει αν υπάρχει cookie με το session token του WooCommerce (woo-session). Αν υπάρχει, το ενσωματώνει στα headers.

Επιπλέον, με το afterware link, γίνεται έλεγχος της απόκρισης και καταγράφεται οποιοδήποτε νέο ή ενημερωμένο session token επιστρέφει ο server. Αυτό το token είτε αποθηκεύεται ξανά στο cookie (σε περίπτωση αλλαγής), είτε αφαιρείται αν ο server επιστρέψει "false".

Η συνολική λογική διασφαλίζει ότι η WooCommerce session του χρήστη παραμένει συγχρονισμένη με τον client, επιτρέποντας λειτουργίες όπως διατήρηση του καλαθιού, ενημέρωση ποσοτήτων και προβολή παραγγελιών με ασφάλεια και αξιοπιστία.

Κεφάλαιο 3ο: Υλοποίηση εφαρμογής

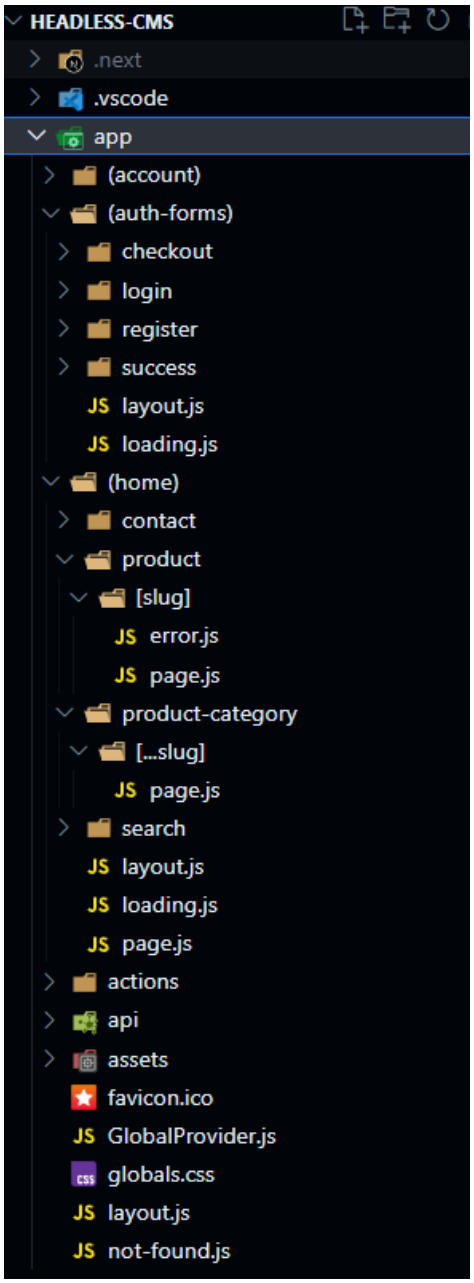
Στο κεφάλαιο αυτό παρουσιάζεται η αναλυτική υλοποίηση της εφαρμογής, η οποία βασίζεται στο framework NEXT με χρήση του App router για την οργάνωση των routes και των layouts. Περιγράφεται η αρχική δομή του έργου, η χρήση route groups για την απομόνωση διαφορετικών περιοχών της εφαρμογής, καθώς και η δυναμική δημιουργία σελίδων μέσω παραμετρικών routes.

Επιπλέον, περιγράφεται η ενσωμάτωση εργαλείων όπως ο Apollo Client για επικοινωνία με WordPress μέσω GraphQL, και η διασύνδεση με το backend για την ανάκτηση περιεχομένου, προϊόντων και κατηγοριών. Η εφαρμογή έχει σχεδιαστεί με γνώμονα την επεκτασιμότητα και την επαναχρησιμοποίηση κοινών στοιχείων UI μέσω κατάλληλης αρχιτεκτονικής και οργάνωσης των φακέλων.

Στα υποκεφάλαια που ακολουθούν, γίνεται λεπτομερής ανάλυση της δομής των routes, του τρόπου φόρτωσης δεδομένων, της διαχείρισης κατάστασης μέσω Apollo, και άλλων βασικών τεχνικών στοιχείων που συνθέτουν την εφαρμογή.

3.1 Δομή εφαρμογής και routing

Η εφαρμογή έχει δομηθεί με βάση το **App Router** του NEXT 14, αξιοποιώντας **route groups** για τη διαχείριση διαφορετικών layouts, καθώς και nested dynamic routes για προβολή περιεχομένου (π.χ. προϊόντων και κατηγοριών)[14]. Στην παρακάτω (εικόνα 3.1) φαίνεται η δομή του φακέλου **app/**, ο οποίος αποτελεί τον πυρήνα του routing και της σύνθεσης περιεχομένου της εφαρμογής:



Εικόνα 3.1: Δομή φακέλου app

- **layout.js:** καθορίζει το layout μιας περιοχής. Στην συγκεκριμένη περίπτωση υπάρχουν και άλλα layouts μεσα στα route groups όπως το (account), αυτό γίνεται επειδή θέλουμε να έχουμε διαφορετική δομή μέσα σε εκείνα τα αρχεία όπως να μην σχεδιάσουμε το μενού της ιστοσελίδας.

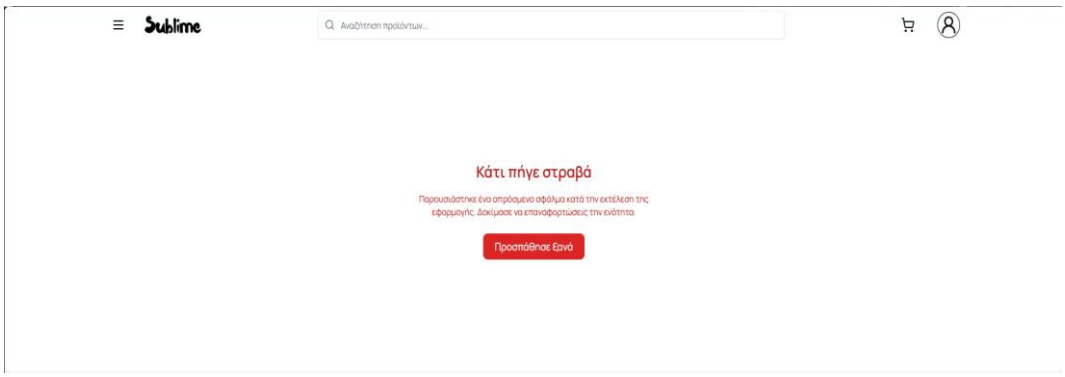
- **page.js:** καθορίζει το περιεχόμενο της σελίδας.

- **loading.js** για χειρισμό κατάστασης. Το αρχείο loading.js χρησιμοποιείται σε ένα route (ή route group) για να εμφανίσει προσωρινό περιεχόμενο όσο φορτώνονται δεδομένα ή components ασύγχρονα.

  Τοποθετείται μέσα σε έναν φάκελο app/route-name/ ή layout/ και ενεργοποιείται αυτόματα όταν υπάρχει καθυστέρηση στη φόρτωση δεδομένων από server components ή dynamic imports.

```
export default function Loading() {
 return (
  <div className="flex items-center justify-center h-screen">
   <div className="border-gray-300 h-20 w-20 animate-spin rounded-full border-8 border-t-blue-600"></div>
  </div>
 );
}
```

- **error.js:** Είναι προαιρετικό και εμφανίζεται μόνο όταν καθυστερεί το αρχικό page.js ή το layout.js που περιέχει async στοιχεία. Μπορεί να χρησιμοποιηθεί για εμφάνιση spinner, skeleton UI, placeholder μηνύματα κ.λπ. Το αρχείο error.js επιτρέπει να χειριστούμε σφάλματα που συμβαίνουν μέσα σε ένα συγκεκριμένο route ή nested route tree. Είναι σαν ένα custom error boundary (εικόνα 3.2).



Εικόνα 3.2: Μήνυμα σφάλματος

```
"use client"; // Error components must be Client Components

import { useEffect } from "react";
```

```
export default function Error({ error, reset }) {

 useEffect(() => {

  // Log the error to an error reporting service

  console.error(error);

 }, [error]);

 return (

  <div className="flex flex-col items-center justify-center min-h-[50vh] px-4 py-8 text-red-700 rounded-xl shadow-md">

   <h2 className="text-2xl font-semibold mb-4">Κάτι πήγε στραβά</h2>

   <p className="text-sm mb-6 max-w-md text-center">

    Παρουσιάστηκε ένα απρόσμενο σφάλμα κατά την εκτέλεση της εφαρμογής.

    Δοκίμασε να επαναφορτώσεις την ενότητα.

   </p>

   <button

    onClick={reset}

    className="px-5 py-2 bg-red-600 hover:bg-red-700 text-white font-medium rounded-lg transition-colors duration-200"

   >

    Προσπάθησε ξανά

   </button>

  </div>

 );

}
```

- **not-found.js:** Το αρχείο not-found.js πρέπει να βρίσκεται μέσα στον φάκελο /app ή σε οποιοδήποτε route group/subpath για να εμφανίζει το 404 μήνυμα σφάλματος όταν δεν υπάρχει αυτό που ψάχνουμε.

Εικόνα 3.3: Εμφάνιση του 404

```
import { TransitionLink } from "@/utils/TransitionLink";

import AuthLayout from "./(auth-forms)/layout";


export default function NotFound() {

 return (

  <AuthLayout>

   <div className="flex items-center min-h-screen px-4 py-12 sm:px-6 md:px-8 lg:px-12 xl:px-16">

    <div className="w-full space-y-6 text-center">

     <div className="space-y-3">

      <h1 className="text-4xl font-bold tracking-tighter sm:text-9xl animate-bounce">

       404

      </h1>
```

```
          <p className="text-gray-500">
            Ουπς! Η σελίδα που αναζητάτε δεν υπάρχει.
          </p>
        </div>
        <TransitionLink
          href="/"
          className="inline-flex h-10 items-center rounded-md
        bg-gray-900 px-8 text-sm font-medium text-gray-50 shadow
        transition-colors hover:bg-gray-900/90 focus-visible:outline-none
        focus-visible:ring-1 focus-visible:ring-gray-950 disabled:pointer-events-none
        disabled:opacity-50 dark:bg-gray-50 dark:text-gray-900 dark:hover:bg-gray-50/90 dark:focus-visible:ring-gray-300"
          prefetch={false}
        >
          Επιστροφή στην αρχική
        </TransitionLink>
      </div>
    </div>
  </AuthLayout>
 );
}
```

Αυτό το component υλοποιεί μια προσαρμοσμένη σελίδα 404 για την εφαρμογή — δηλαδή τη σελίδα που εμφανίζεται όταν ο χρήστης προσπαθεί να επισκεφθεί μια διαδρομή (route) που δεν υπάρχει.

Τα route groups χρησιμοποιούνται για την οργάνωση του περιεχομένου και τον ορισμό διαφορετικών layouts χωρίς να επηρεάζεται η δομή των URLs.

Επιπλέον, χρησιμοποιούνται δυναμικά routes όπως:

- **[slug]:** για προβολή ενός μόνο πόρου με βάση την παράμετρο.

- **[...slug]:** για catch-all routing, χρήσιμο όταν δεν γνωρίζουμε εκ των προτέρων το βάθος του path.

Για παράδειγμα:

/products/[slug] → εμφανίζει συγκεκριμένο προϊόν.

/product-category/[...slug] → μπορεί να αντιστοιχεί σε /product-category/grafiki-yli/gomes.

Το product-category είναι το προκαθορισμένο από το WordPress path που μπορούμε να βάλουμε. Επίσης μπορεί να γίνει αλλαγή αυτού του path.

3.2 Δημιουργία αρχικής σελίδας

Η αρχική σελίδα της εφαρμογής αποτελεί το σημείο εισόδου για τον επισκέπτη και περιλαμβάνει δυναμικά στοιχεία που φορτώνονται ασύγχρονα. Για τον λόγο αυτό, αξιοποιείται η δυνατότητα του NEXT να χειρίζεται async components μέσω της React Suspense.

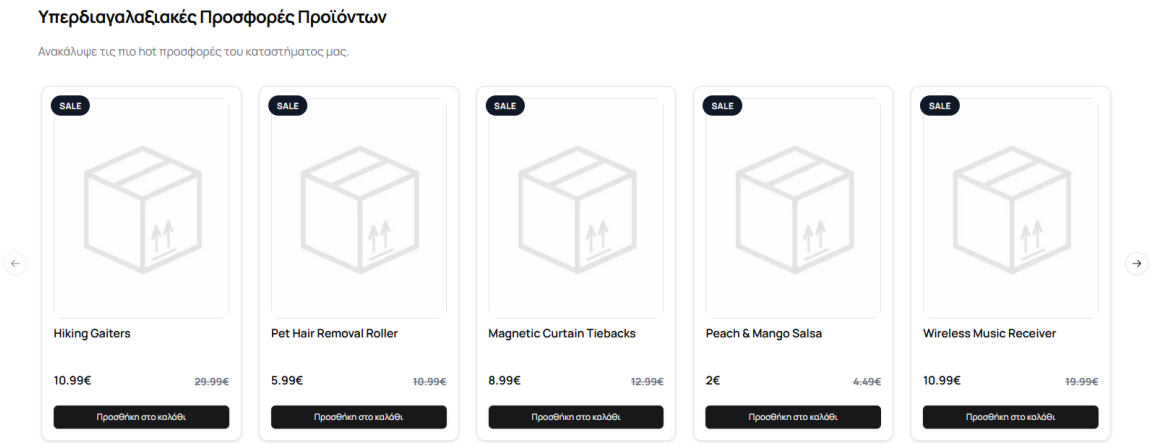
```
import { HeroSlider } from "@/components/Sliders/HeroSlider";
import { SalesSlider } from "@/components/Sliders/SalesSlider";
import { BestSellers } from "@/components/Sliders/BestSellers";
import { Suspense } from "react";
import { CarouselSkeleton } from "@/components/Skeletons/Sliders/CarouselSkeleton";
import { SalesSliderSkeleton } from "@/components/Skeletons/Sliders/SalesSliderSkeleton";
import { BestSellersSkeleton } from "@/components/Skeletons/Sliders/BestSellersSkeleton";
import Featured from "@/components/Sliders/Featured";

export default async function Home() {
 return (
  <div className="container mx-auto">
   <Suspense fallback={<CarouselSkeleton />}>
    <HeroSlider />
   </Suspense>
   <Suspense fallback={<SalesSliderSkeleton />}>
    <SalesSlider />
   </Suspense>
   <Suspense fallback={<BestSellersSkeleton />}>
```

```
      <BestSellers />
    </Suspense>
    <Featured />
  </div>
 );
}
```

Αυτός ο κώδικας αποτελεί τη βασική υλοποίηση της αρχικής σελίδας της εφαρμογής και είναι γραμμένος ως server component στη NEXT. Η αρχιτεκτονική βασίζεται στη χρήση του <Suspense> από τη React, το οποίο επιτρέπει την καθυστέρηση της απόδοσης (rendering) ενός component μέχρι να φορτωθούν τα απαραίτητα δεδομένα ή ο ίδιος ο component.

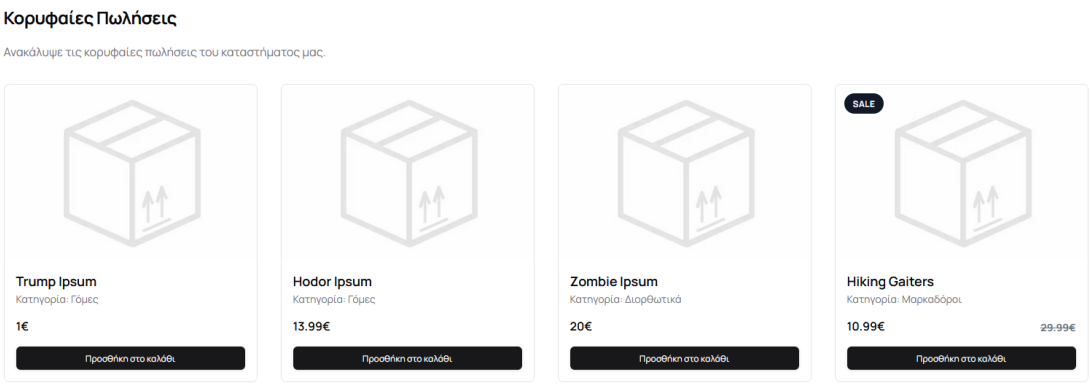
Η σελίδα Home περιέχει 4 κύρια sections:

- HeroSlider: το αρχικό καρουζέλ
- SalesSlider: προϊόντα σε προσφορά



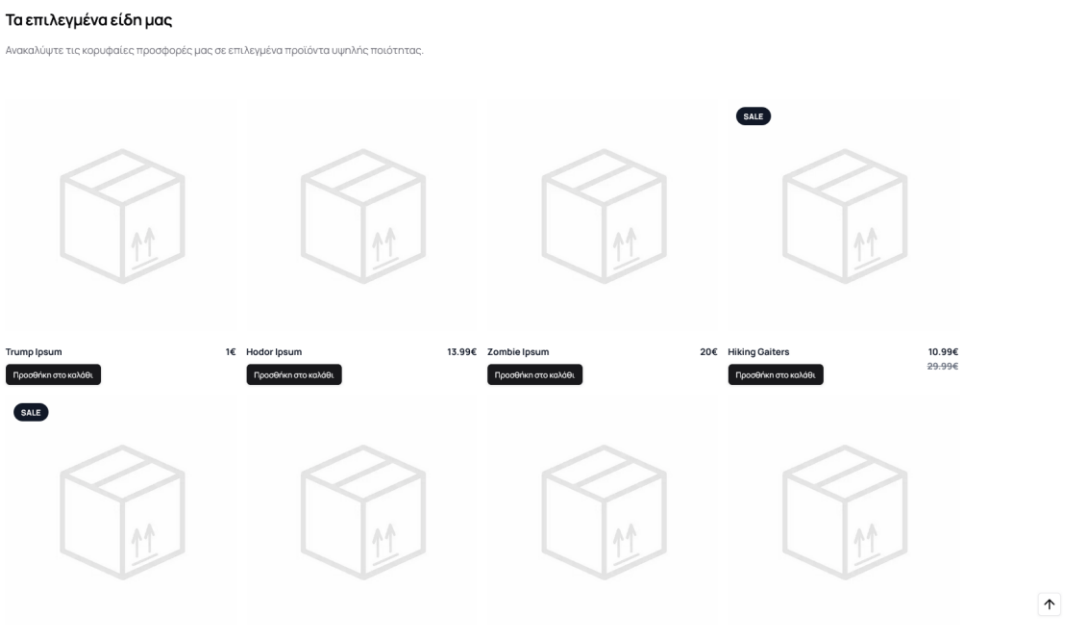
Εικόνα 3.4: Το αρχείο SalesSlider

- BestSellers: τα πιο δημοφιλή προϊόντα



Εικόνα 3.5: Το αρχείο BestSellers

- Featured: επιλεγμένα προϊόντα



Εικόνα 3.6: Το αρχείο Featured

Όλα τα sections εκτός του Featured περιβάλλονται από Suspense, ώστε να εμφανίζεται ένα skeleton UI κατά τη διάρκεια της φόρτωσης των δεδομένων. Τα skeletons έχουν φτιαχτεί έτσι ώστε να αντανακλούν παρομοίως την σχεδίαση του component που θα εμφανίσει.

Το Suspense είναι μια λειτουργία του React (και υποστηρίζεται πλήρως από το App Router του NEXT 14) που μας επιτρέπει να περιμένουμε την απόκριση ενός ασύγχρονου component και να εμφανίζουμε ένα placeholder.[15]

Στα παραπάνω θα δούμε τα εξής:

- Αναμονή φόρτωσης των δεδομένων (αν το SalesSlider είναι ένα async Server Component ή περιέχει dynamic import).

- Skeleton/Placeholder εμφάνιση κατά τη διάρκεια της αναμονής (βελτιώνει την εμπειρία χρήστη).

- Αυτόματη μετάβαση στο κανονικό component μόλις ολοκληρωθεί η φόρτωση.

```
import { Skeleton } from "@/components/ui/skeleton";

export async function CarouselSkeleton() {

 return (

  <div className="w-full">

   <Skeleton className="h-[90vh] w-full rounded-md" />

  </div>

 );

}
```

3.3 Δημιουργία μενού

Το Header της εφαρμογής αποτελεί έναν από τους βασικότερους άξονες πλοήγησης για τον χρήστη, προσφέροντας άμεση πρόσβαση σε κρίσιμες λειτουργίες όπως η αναζήτηση προϊόντων, το καλάθι αγορών και η σελίδα προφίλ. Έχει σχεδιαστεί με σκοπό να εξυπηρετεί διαφορετικούς τύπους χρηστών, προσαρμόζοντας δυναμικά το περιεχόμενό του ανάλογα με το αν ο χρήστης είναι συνδεδεμένος ή όχι. Για παράδειγμα, όταν ο χρήστης είναι συνδεδεμένος, εμφανίζονται επιλογές που σχετίζονται με τις παραγγελίες του ή την αποσύνδεση, ενώ σε αντίθετη περίπτωση προβάλλονται επιλογές για σύνδεση ή εγγραφή.

Ιδιαίτερη έμφαση δόθηκε στην ευχρηστία και την προσβασιμότητα, ώστε κάθε επισκέπτης να μπορεί να περιηγηθεί εύκολα στην εφαρμογή ανεξαρτήτως συσκευής ή επιπέδου εμπειρίας. Το Header, όπως και όλο το υπόλοιπο περιβάλλον της εφαρμογής, έχει υλοποιηθεί με responsive design λογική, προσαρμοζόμενο ομαλά τόσο σε desktop όσο και σε κινητά ή tablets. Έτσι εξασφαλίζεται μια συνεπής και φιλική εμπειρία χρήσης σε κάθε μέγεθος οθόνης.

Αξίζει επίσης να αναφερθεί ότι όλα τα components της εφαρμογής, συμπεριλαμβανομένου του Header έχουν δημιουργηθεί με βάση τη βιβλιοθήκη shadcn/ui, η οποία συνδυάζει λειτουργικότητα και αισθητική[16]. Η χρήση αυτής της βιβλιοθήκης εξασφάλισε ένα αρμονικό και μοντέρνο UI/UX, ενώ επέτρεψε την ανάπτυξη επαναχρησιμοποιήσιμων στοιχείων, τα οποία διατηρούν συνοχή σε όλη την εφαρμογή. Αυτό διευκόλυνε σημαντικά την υλοποίηση ενός καλά δομημένου περιβάλλοντος χρήστη με σταθερά standards σε εμφάνιση και συμπεριφορά.

```
import { ShoppingCartIcon } from "./ShoppingCartIcon";
import { SearchInput } from "../Search/SearchInput";
import { NavBarScroll } from "./NavBarScroll";
import { UserAvatar } from "./UserAvatar";
import { Suspense } from "react";
import { SearchInputSkeleton } from "../Skeletons/SearchBar/SearchInputSkeleton";
import MainNav from "./MainNav";

export const Navbar = async () => {
 return (
  <NavBarScroll>
   <div className="container mx-auto flex items-center justify-between h-14 px-4">
    <div className="flex items-center space-x-4">
     <MainNav />
    </div>
    <div className="flex-1 flex justify-center">
     <Suspense fallback={<SearchInputSkeleton />}>
      <SearchInput placeholder="Αναζήτηση προϊόντων..." />
     </Suspense>
    </div>
    <div className="flex items-center">
     <ShoppingCartIcon
      className="hidden md:block mr-10"
      badgeCounter="hidden md:block"
     />
     <UserAvatar />
    </div>
   </div>
  </NavBarScroll>
 );
```

```
};
```

Αυτό το component αποτελεί την υλοποίηση της μπάρας πλοήγησης (Navbar) της εφαρμογής και επιστρέφει ένα λειτουργικό header.

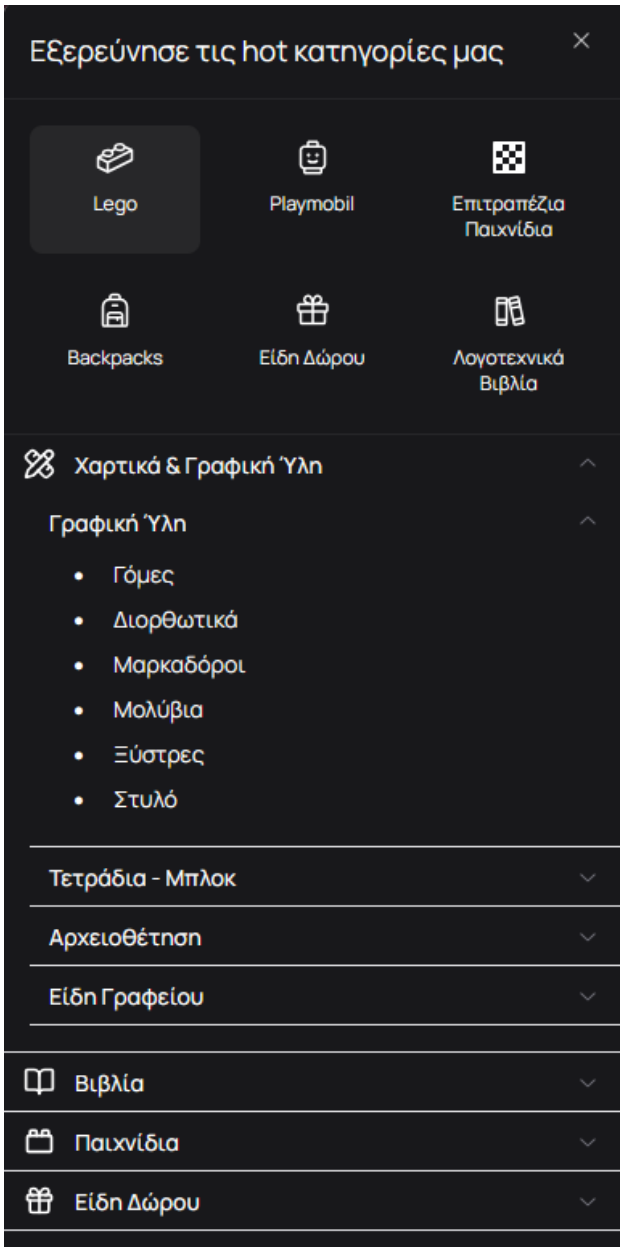
Η μπάρα χωρίζεται σε τρία κύρια μέρη:

- Στα αριστερά, μέσω του MainNav, εμφανίζεται το βασικό μενού πλοήγησης της σελίδας.

- Στο κέντρο, τοποθετείται η μπάρα αναζήτησης SearchInput, η οποία φορτώνεται μέσα από Suspense με skeleton placeholder (SearchInputSkeleton) για ομαλότερη εμπειρία κατά τη φόρτωση.

- Στα δεξιά, προβάλλονται δύο βασικά εργαλεία για τον χρήστη: το ShoppingCartIcon, το οποίο εμφανίζει το εικονίδιο του καλαθιού με ένδειξη ποσότητας προϊόντων (badge), και το UserAvatar, που εμφανίζει την του χρήστη ή εικονίδιο σύνδεσης, ανάλογα με το αν έχει συνδεθεί.

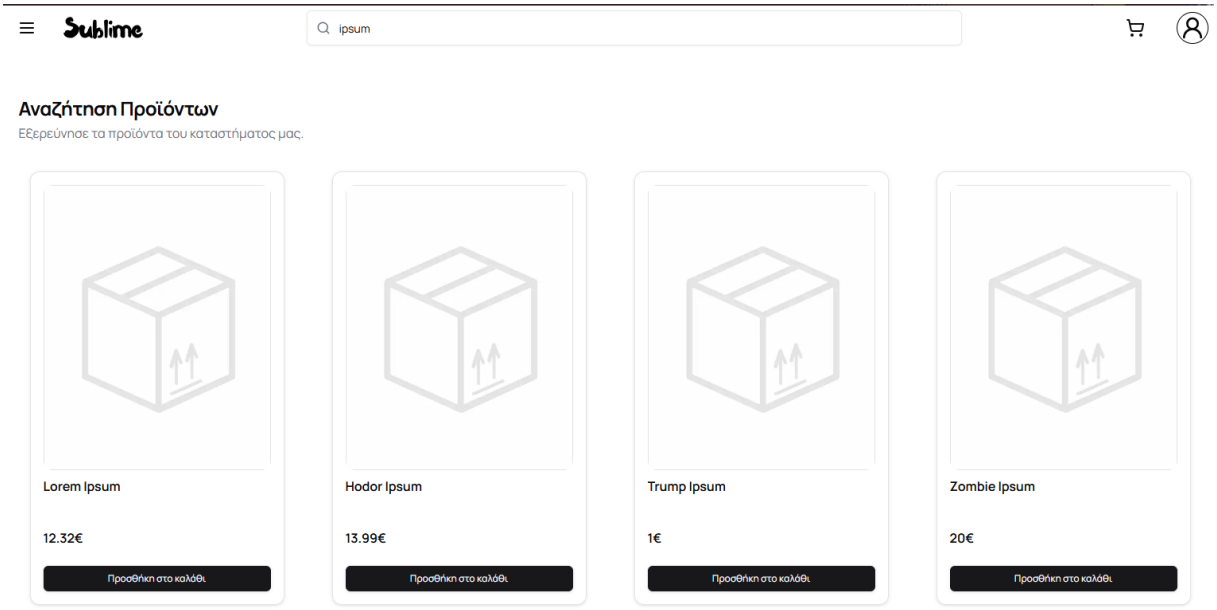
Η συνολική διάταξη είναι responsive, προσαρμοσμένη για μικρότερες συσκευές.



Εικόνα 3.7: Εμφάνιση του μενού

Η λειτουργία αναζήτησης στην εφαρμογή αξιοποιεί τις δυνατότητες του App Router του NEXT 14, επιτρέποντας server-side rendering (SSR) με χρήση των παραμέτρων URL (searchParams). Αυτός ο

σχεδιασμός προσφέρει σημαντικά πλεονεκτήματα σε επίπεδο SEO, ταχύτητας και διαμοιρασμού περιεχομένου.[17]



Εικόνα 3.8: Αναζήτηση προϊόντων

Πώς λειτουργεί η αναζήτηση με βάση το App Router

Στη NEXT η αναζήτηση υλοποιείται με τον εξής τρόπο:

- **Client-side:** Ο χρήστης πληκτρολογεί στο πεδίο αναζήτησης.
- **URL Update:** Η εφαρμογή ενημερώνει το URL με την παράμετρο αναζήτησης (?q=...) χρησιμοποιώντας τα hooks useRouter() και useSearchParams() από το next/navigation.
- **Server-side Rendering:** Το page.js (Server Component) λαμβάνει την παράμετρο searchParams και εκτελεί αναζήτηση στον server (π.χ., μέσω GraphQL ή REST API).
- **Απόδοση αποτελεσμάτων:** Η σελίδα φορτώνει με τα νέα δεδομένα, διατηρώντας ταυτόχρονα την κατάσταση στο URL.

Πλεονεκτήματα αυτής της προσέγγισης:

- **SEO-friendly:** Τα αποτελέσματα αναζήτησης αποδίδονται στον server, καθιστώντας τα προσβάσιμα από τις μηχανές αναζήτησης.

- **Διαμοιρασμός URL:** Οι χρήστες μπορούν να μοιραστούν ή να αποθηκεύσουν URLs με συγκεκριμένα queries.

- **Απλότητα:** Δεν απαιτείται επιπλέον client-side state management.

- **Βελτιωμένη απόδοση:** Η αναζήτηση εκτελείται στον server, μειώνοντας το φορτίο στον client.

```
import { query } from "@/lib/apolloClient";
import { SEARCH_PRODUCTS_QUERY } from "@/queries/searchProducts";
import { Suspense } from "react";
import ProductCard from "@/components/ProductCard/ProductCard";
import { ProductCardSkeleton } from "@/components/Skeletons/Products/ProductCardSkeleton";

// Async component for product fetching
async function ProductList({ search }) {
  const { data } = await query({
    query: SEARCH_PRODUCTS_QUERY,
    variables: { search }, // Pass the search term to the query
  });
  const products = data.products.nodes;
  if (products.length === 0) {
    return (
      <p className="text-center text-muted-foreground">
        Δεν βρέθηκαν προϊόντα που να ταιριάζουν με την αναζήτησή σας.
      </p>
    );
  }
  return (
    <div className="grid grid-cols-2 sm:grid-cols-2 md:grid-cols-3 lg:grid-cols-4 gap-6 md:gap-8">
      {products.map((product, index) => (
        <div key={index} className="md:basis-1/2 lg:basis-1/5">
          <div className="p-1 grid justify-center">
            <ProductCard product={product} index={index} />
          </div>
        </div>
      ))}
```

```
    </div>
  );
}
export default function Page({ searchParams }) {
  const querySearch = searchParams?.query || "";
  const keyString = `search=${querySearch}`; // Unique key for Suspense
  return (
    <>
      <section className="py-12 bg-background min-h-[50vh]">
        <div className="container px-4 md:px-6 mx-auto">
          <div className="grid gap-6 md:gap-8">
            <div className="grid gap-1">
              <h1 className="text-2xl font-bold tracking-tight">
                Αναζήτηση Προϊόντων
              </h1>
              <p className="text-muted-foreground">
                Εξερεύνησε τα προϊόντα του καταστήματος μας.
              </p>
            </div>

            {!querySearch && (
              <p className="text-center text-muted-foreground">
                Παρακαλώ πληκτρολογήστε κάτι για να ξεκινήσετε την αναζήτηση.
              </p>
            )}

            {querySearch && (
              <Suspense key={keyString} fallback={<ProductCardSkeleton />}>
                {/* Render the async ProductList component */}
                <ProductList search={querySearch} />
              </Suspense>
            )}
          </div>
        </div>
      </section>
    </>
  );
```

```
}
```

Ο παραπάνω κώδικας αφορά την υλοποίηση της σελίδας αναζήτησης προϊόντων στην εφαρμογή. Το component Page λαμβάνει τις παραμέτρους αναζήτησης από το searchParams, και εάν υπάρχει κάποιο query string (search=query), ενεργοποιείται η απόδοση αποτελεσμάτων.

Αν ο χρήστης δεν έχει εισάγει κάποια τιμή, εμφανίζεται ένα ενημερωτικό μήνυμα που του ζητάει να ξεκινήσει την αναζήτηση. Όταν υπάρχει τιμή στο query, ο ProductList component (που είναι ασύγχρονος) εκτελεί ένα GraphQL query μέσω του Apollo Client για να ανακτήσει προϊόντα που ταιριάζουν με τον όρο αναζήτησης.

Η χρήση του Suspense εξασφαλίζει καλύτερη εμπειρία χρήστη, εμφανίζοντας προσωρινά ένα skeleton loading UI (ProductCardSkeleton) όσο φορτώνονται τα προϊόντα. Αν δεν βρεθούν σχετικά προϊόντα,

εμφανίζεται ένα απλό μήνυμα “Δεν βρέθηκαν προϊόντα”. Αν υπάρχουν αποτελέσματα, εμφανίζονται μέσω του ProductCard component.

Πρόκειται για μια δομή που αξιοποιεί τις δυνατότητες του App Router στη NEXT, με async components και caching συνδυασμένο με SSR αποδόσεις περιεχομένου.

Στην συνέχεια για το σύστημα καλαθιού αγορών είναι ένα πλήρως λειτουργικό και αποδοτικό module για e-commerce εφαρμογές σε NEXT. Έχει σχεδιαστεί με γνώμονα την απόκριση και τη βελτιστοποίηση της εμπειρίας χρήστη και περιλαμβάνει:

- Context-based state management. Η χρήση του CartContext με useReducer επιτρέπει την ευέλικτη και scalable διαχείριση του καλαθιού. Τα δεδομένα αποθηκεύονται στο localStorage για διατήρηση μεταξύ των sessions.

- Live ενημέρωση ποσοτήτων με debounce. Οι αλλαγές στην ποσότητα των προϊόντων εφαρμόζονται άμεσα στο UI (optimistic UI) και αποστέλλονται στον server με καθυστέρηση μέσω debounced GraphQL mutation, μειώνοντας τον αριθμό των requests.

- Οπτική ειδοποίηση πλήθους προϊόντων (badge). Το εικονίδιο του καλαθιού εμφανίζει ένα δυναμικό badge με το πλήθος των αντικειμένων.

- Πλαϊνό panel (sheet) με responsive design. Το καλάθι εμφανίζεται ως Sheet με scrollable περιεχόμενο, συνολικό κόστος, αφαιρούμενα προϊόντα και CTA για checkout.

- Αφαίρεση προϊόντος με ασφάλεια. Υποστηρίζεται η πλήρης αφαίρεση προϊόντος από το καλάθι με ενημέρωση του context και του localStorage.

Εικόνα 3.9: Προβολή καλαθιού προϊόντων

Τεχνική Υλοποίηση του καλαθιού με το hook του useReducer για όλες τις περιπτώσεις που έχουμε.

Διαχείριση Καλαθιού (Reducer):

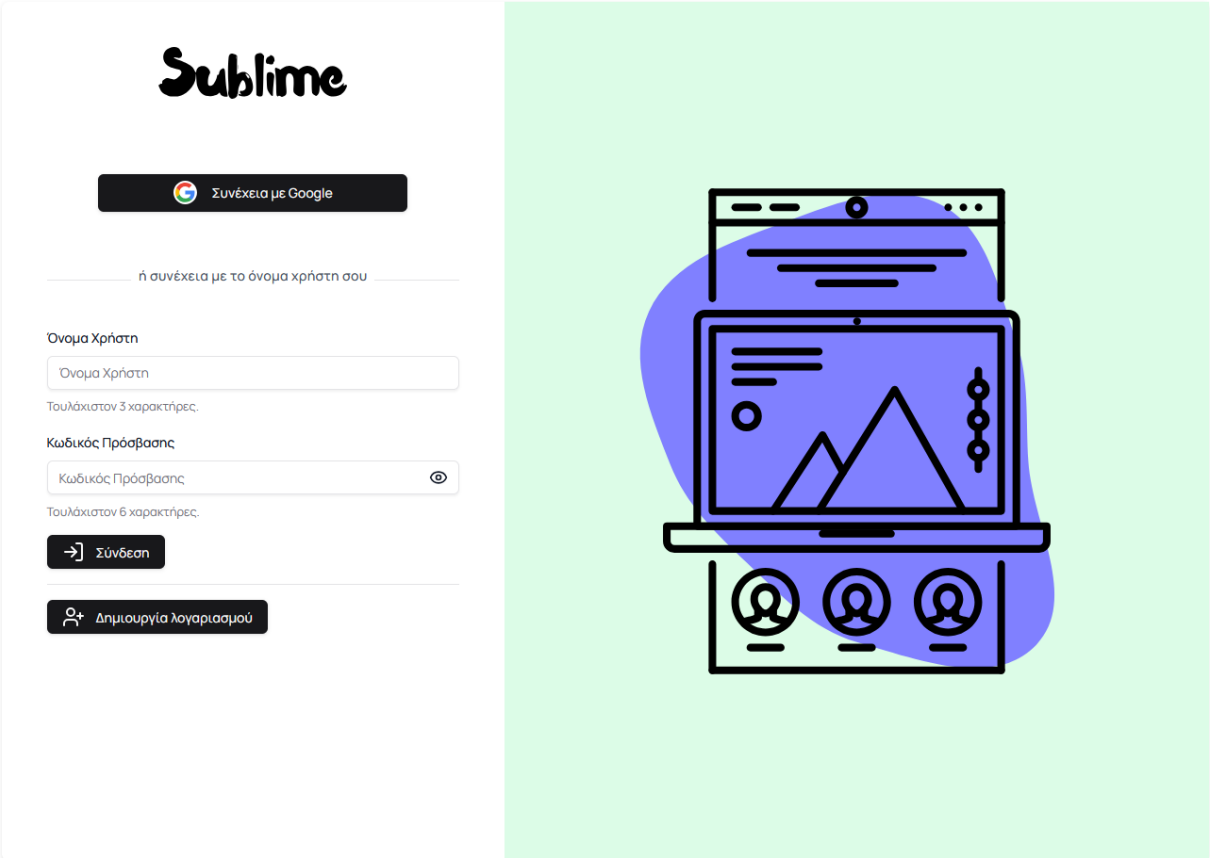
- **ADD_ITEM:** Προσθήκη ή ενημέρωση υπάρχοντος προϊόντος με αύξηση ποσότητας.
- **UPDATE_ITEM:** Ενημέρωση ποσότητας ή μοναδικού key προϊόντος.
- **REMOVE_ITEM:** Αφαίρεση προϊόντος με βάση το key.
- **SET_CART:** Αρχική φόρτωση από localStorage.

Product Quantity Controls: Πλήκτρα "+" και "-" ενημερώνουν την ποσότητα προϊόντων και πυροδοτούν debounced mutation (useDebouncedCallback) προς το backend με χρήση Apollo GraphQL.

GraphQL Mutation: Το UPDATE_ITEM_QUANTITY mutation εκτελείται με optimistic response για απρόσκοπτη εμπειρία χρήστη, δηλαδή το request ακόμα τρέχει από πίσω ενώ ο χρήστης βλέπει ότι

γίνει το αίτημα για να προστεθεί στο καλάθι. Αυτό βοηθάει στην ταχύτητα της ιστοσελίδας και την καλύτερη εμπειρία του χρήστη.

Τέλος στην εφαρμογή έχει υλοποιηθεί ένα διαδραστικό μενού χρήστη, το οποίο προσαρμόζεται δυναμικά ανάλογα με το αν ο επισκέπτης είναι συνδεδεμένος ή όχι. Στην περίπτωση που ο χρήστης δεν έχει πραγματοποιήσει σύνδεση, εμφανίζεται ένα εικονίδιο προφίλ, το οποίο λειτουργεί ως σύνδεσμος και οδηγεί στη σελίδα εισόδου. Με αυτόν τον τρόπο, η πρόσβαση στη διαδικασία σύνδεσης παραμένει απλή και προσβάσιμη από το βασικό μενού πλοήγησης.

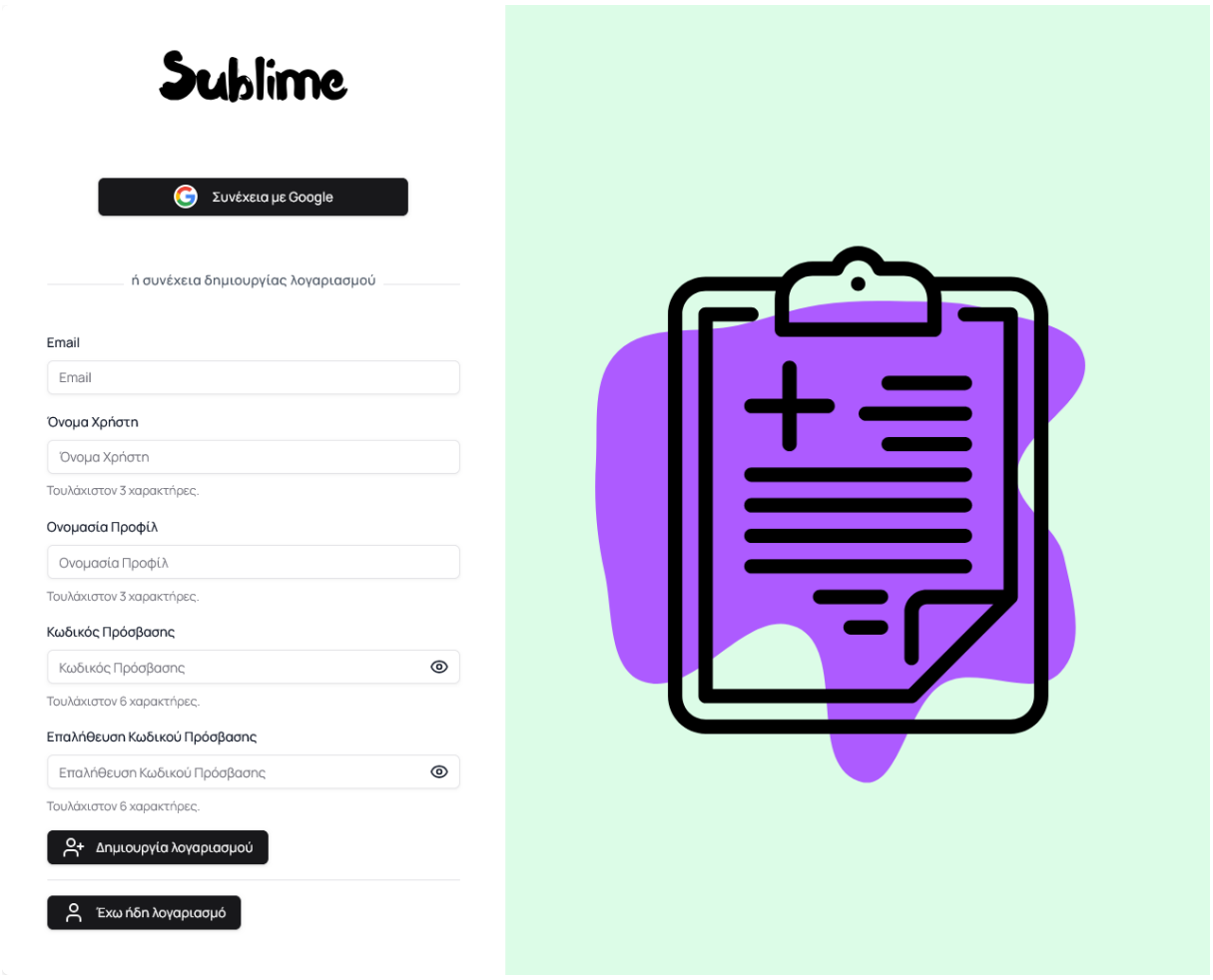


Εικόνα 3.10: Φόρμα σύνδεσης χρήστη

Στο σύστημα εγγραφής χρηστών της εφαρμογής έχει υλοποιηθεί μια απλή και ασφαλής φόρμα εγγραφής, η οποία περιλαμβάνει τα απαραίτητα πεδία: διεύθυνση email, όνομα χρήστη, ονομασία προφίλ (π.χ. πλήρες όνομα), κωδικό πρόσβασης και επαλήθευση κωδικού πρόσβασης. Για την επικύρωση των τιμών που εισάγει ο χρήστης, χρησιμοποιείται η βιβλιοθήκη Zod, η οποία παρέχει έναν σαφή και ευέλικτο τρόπο ορισμού σχημάτων (schemas) και ελέγχου εγκυρότητας δεδομένων στον

client. Στον παρακάτω κώδικα είναι ένα validation από τα πολλά που έχουμε για την σύνδεση του χρήστη με την εφαρμογή μας, χρησιμοποιώντας την Zod βιβλιοθήκη.[18]

```
// Schema for the signIn

export const SignInFormSchema = z.object({

  username: z

    .string({ required_error: "Το όνομα χρήστη είναι υποχρεωτικό" })

    .nonempty({ message: "Παρακαλώ συμπληρώστε το όνομα χρήστη σας" })

    .min(3, "Το όνομα χρήστη θα πρέπει να είναι τουλάχιστον 3 χαρακτήρες")

    .max(20, "Το όνομα χρήστη θα πρέπει να είναι εώς 20 χαρακτήρες")

    .regex(/^\S+$/, "Το όνομα χρήστη δεν μπορεί να περιέχει κενά"),

  password: z

    .string()

    .nonempty({ message: "Παρακαλώ συμπληρώστε το κωδικό πρόσβαση σας" })

    .min(6, "Ο κωδικός θα πρέπει να είναι τουλάχιστον 6 χαρακτήρες")

    .max(20, "Ο κωδικός θα πρέπει να είναι εώς 20 χαρακτήρες"),

});
```



Εικόνα 3.11: Φόρμα εγγραφής χρήστη

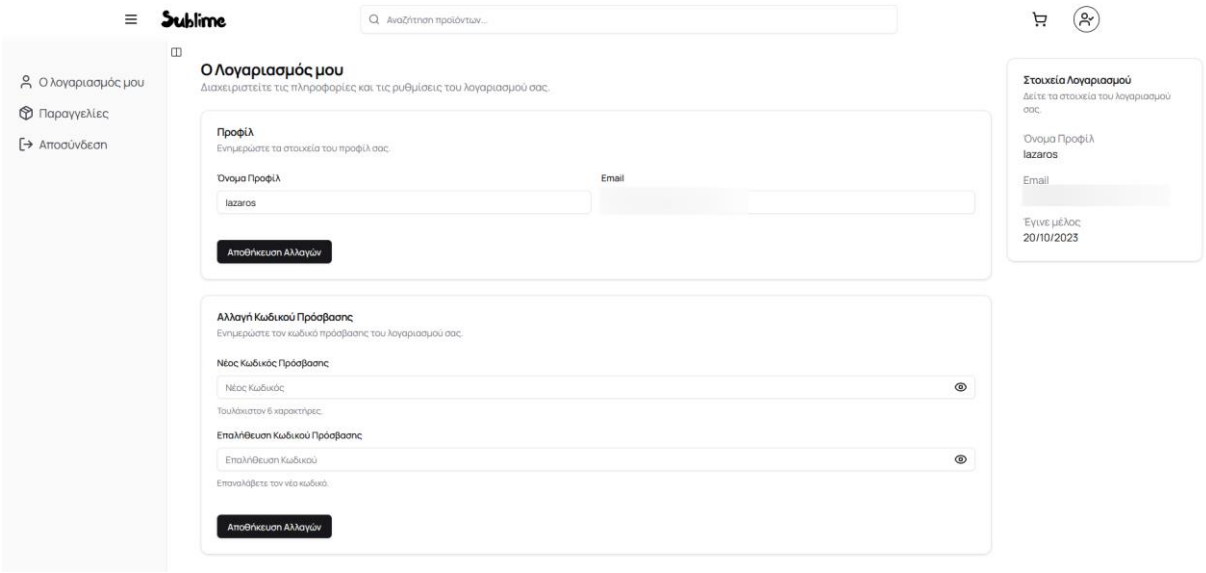
Κάθε πεδίο υπόκειται σε συγκεκριμένους κανόνες επικύρωσης (validation rules). Για παράδειγμα:

- Το email πρέπει να είναι σε έγκυρη μορφή.
- Το όνομα χρήστη πρέπει να έχει έναν ελάχιστο αριθμό χαρακτήρων και να μην περιέχει απαγορευμένους χαρακτήρες.
- Η ονομασία προφίλ είναι το όνομα του προφίλ που θα εμφανίζεται.
- Ο κωδικός πρόσβασης απαιτείται να είναι αρκετά ασφαλής, με ελάχιστο αριθμό χαρακτήρων και πιθανόν σύνθετους κανόνες (π.χ. κεφαλαία, σύμβολα κ.λπ.).

- Η επαλήθευση κωδικού πρόσβασης συγκρίνεται με τον αρχικό κωδικό για να διασφαλιστεί ότι δεν υπάρχουν λάθη κατά την καταχώρηση.

Σε περίπτωση που κάποια από τις εισόδους αποτύχει στην επικύρωση, εμφανίζονται κατάλληλα μηνύματα λάθους στον χρήστη πριν την αποστολή των δεδομένων στον διακομιστή. Με αυτόν τον τρόπο διασφαλίζεται μια καλύτερη εμπειρία χρήσης, μειώνονται τα περιττά αιτήματα προς τον server και αυξάνεται η συνολική ασφάλεια της διαδικασίας εγγραφής.

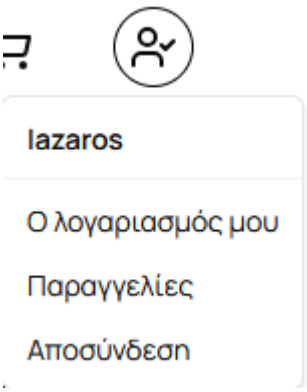
Αντίθετα, όταν ο χρήστης έχει συνδεθεί, στη θέση του γενικού εικονιδίου εμφανίζεται η προσωπική του εικόνα προφίλ, εφόσον αυτή είναι διαθέσιμη. Παράλληλα, ενεργοποιείται ένα αναδυόμενο μενού που παρέχει εξατομικευμένες επιλογές. Συγκεκριμένα, ο χρήστης μπορεί να δει το όνομά του, να μεταβεί στη σελίδα διαχείρισης του λογαριασμού του, να ελέγξει τις παραγγελίες του, καθώς και να επιλέξει την αποσύνδεση από την πλατφόρμα.



Εικόνα 3.12: Διαχείριση του λογαριασμού ενός συνδεδεμένου χρήστη

Η προσέγγιση αυτή ενισχύει την εμπειρία χρήστη, προσφέροντας μια καθαρή και ευχάριστη αλληλεπίδραση με το περιβάλλον της εφαρμογής. Επιπλέον, ενσωματώνει με ομαλό τρόπο βασικές

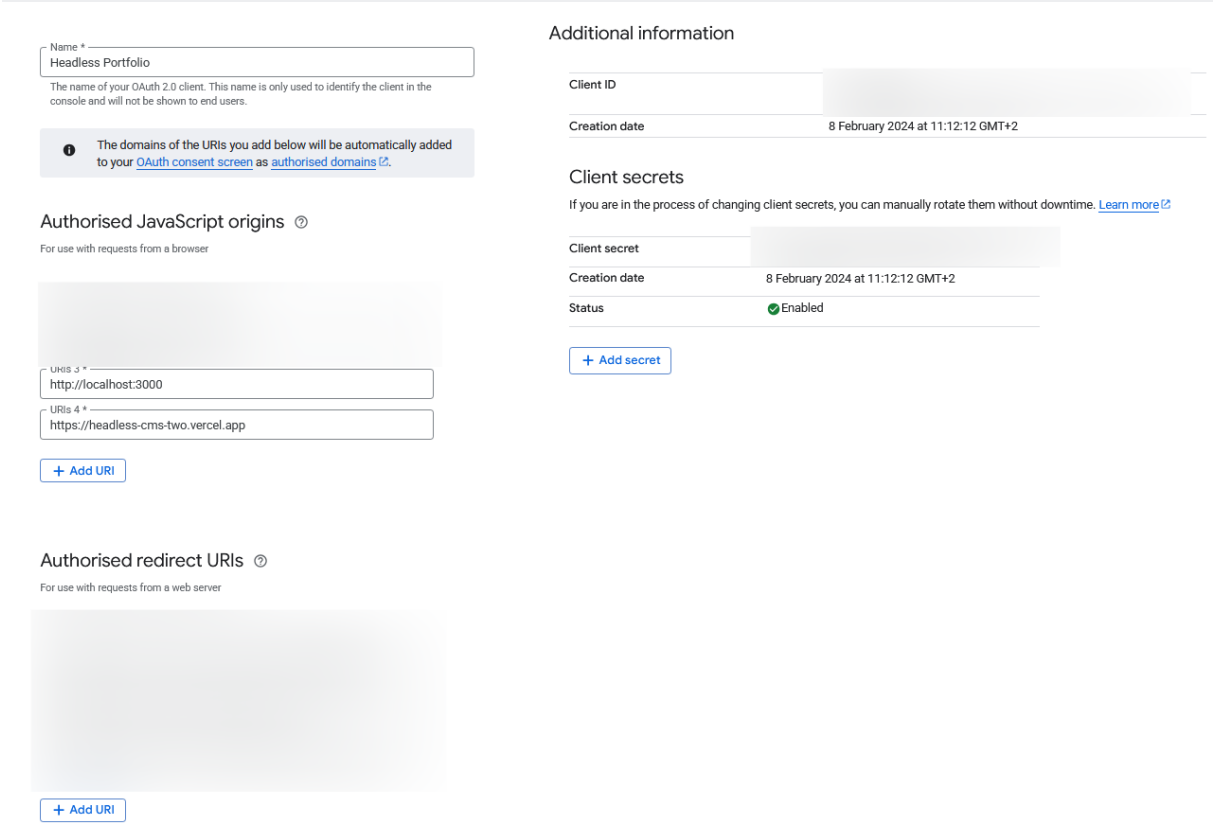
λειτουργίες αυθεντικοποίησης, επιτρέποντας την εύκολη διαχείριση του λογαριασμού μέσα από το κεντρικό περιβάλλον πλοήγησης.



Εικόνα 3.13: Λίστα επιλογών του χρήστη

Επιπλέον, στην εφαρμογή έχει ενσωματωθεί λειτουργία αυθεντικοποίησης μέσω Google OAuth, προσφέροντας μια εναλλακτική και ιδιαίτερα φιλική προς τον χρήστη μέθοδο σύνδεσης. Με αυτή τη δυνατότητα, οι χρήστες μπορούν να συνδεθούν στην πλατφόρμα χρησιμοποιώντας τον υπάρχοντα

λογαριασμό τους στη Google, χωρίς την ανάγκη δημιουργίας νέου λογαριασμού ή εισαγωγής κωδικών πρόσβασης.



Εικόνα 3.14: Google Cloud Console εγκατάσταση για την είσοδο με Google OAuth

Η ενσωμάτωση της αυθεντικοποίησης μέσω Google ενισχύει την ασφάλεια της εφαρμογής, καθώς αξιοποιεί τα πρωτόκολλα ασφαλείας της Google, ενώ ταυτόχρονα απλοποιεί τη διαδικασία εισόδου, μειώνοντας σημαντικά την τριβή για τον τελικό χρήστη. Με την πρώτη επιτυχημένη σύνδεση μέσω Google, η εφαρμογή δημιουργεί αυτόματα τον απαραίτητο λογαριασμό στη βάση δεδομένων και αποθηκεύει τα βασικά στοιχεία του χρήστη, όπως όνομα, email και εικόνα προφίλ.

Για να ενεργοποιηθεί η σύνδεση μέσω Google OAuth, απαιτείται η δημιουργία ενός project στην κονσόλα προγραμματιστών της Google (Google Cloud Console). Στο πλαίσιο αυτού του project, γίνεται ενεργοποίηση του OAuth 2.0 API και δημιουργία ενός OAuth Client. Κατά τη διαδικασία αυτή, παρέχεται ένα μοναδικό Client ID και ένα αντίστοιχο Client Secret, τα οποία χρησιμοποιούνται από την εφαρμογή για την επικοινωνία με τους διακομιστές της Google.

Το Client ID είναι δημόσιο και χρησιμοποιείται για την αναγνώριση της εφαρμογής προς την Google, ενώ το Client Secret παραμένει ιδιωτικό και λειτουργεί ως ένας μηχανισμός επαλήθευσης της ταυτότητας της εφαρμογής κατά την ανταλλαγή κωδικών πρόσβασης. Αυτά τα στοιχεία είναι

απαραίτητα για τη διασφάλιση της ασφάλειας και της ορθής λειτουργίας της διαδικασίας αυθεντικοποίησης.

Η υλοποίηση αυτή εξασφαλίζει μία πιο ολοκληρωμένη και σύγχρονη εμπειρία αυθεντικοποίησης, καθιστώντας την εφαρμογή προσβάσιμη και λειτουργική για ένα ευρύτερο κοινό, ενώ παράλληλα διατηρεί υψηλά πρότυπα ασφάλειας και εργονομίας.

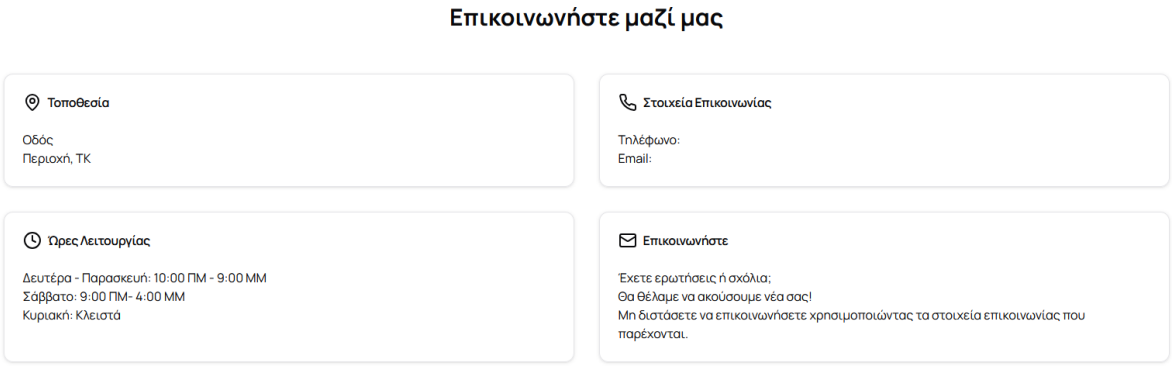
3.4 Υποσελίδα

Στο κάτω μέρος της ιστοσελίδας (footer) έχει προστεθεί ένα σταθερό μενού πλοήγησης που περιλαμβάνει συνδέσμους προς βασικές σελίδες, όπως η Πολιτική Απορρήτου και η Επικοινωνία. Οι σελίδες αυτές είναι διαθέσιμες σε όλα τα υποσελίδια της πλατφόρμας, προσφέροντας εύκολη πρόσβαση στον χρήστη ανεξαρτήτως της τοποθεσίας του στον ιστότοπο.

Εικόνα 3.15: Υποσελίδα της εφαρμογής

Η σελίδα Πολιτική Απορρήτου έχει δημιουργηθεί μέσω της δωρεάν υπηρεσίας του ιστότοπου FreePrivacyPolicy.com, παρέχοντας ένα πλήρως διαμορφωμένο και νομικά τεκμηριωμένο κείμενο, το οποίο καλύπτει κρίσιμα ζητήματα σχετικά με τη συλλογή και επεξεργασία προσωπικών δεδομένων, τη χρήση cookies, τις δυνατότητες επικοινωνίας, καθώς και τα δικαιώματα του χρήστη σύμφωνα με τον Γενικό Κανονισμό Προστασίας Δεδομένων (GDPR).

Η σελίδα επικοινωνίας προσφέρει μία απλή σελίδα πληροφοριών του καταστήματος για να μπορέσει να επικοινωνήσει ο χρήστης με την επιχείρηση.



Εικόνα 3.16: Σελίδα επικοινωνίας καταστήματος

3.5 Σελίδα προϊόντων ανά κατηγορία

Η σελίδα Product Category αποτελεί βασικό μέρος της εμπειρίας περιήγησης στην πλατφόρμα και έχει σχεδιαστεί ώστε να εμφανίζει όλα τα προϊόντα που ανήκουν σε μία συγκεκριμένη κατηγορία, όπως π.χ. “Γόμες”, “Στυλό”, κ.λπ. Η κατηγοριοποίηση των προϊόντων γίνεται δυναμικά, με βάση το περιεχόμενο που διαχειρίζεται ο διαχειριστής μέσω του headless CMS (WordPress), από όπου προέρχονται τα δεδομένα μέσω GraphQL queries.

Κατά την είσοδο του χρήστη σε μία σελίδα κατηγορίας, γίνεται ανάκτηση των προϊόντων που σχετίζονται με αυτή, και εμφανίζονται με τη μορφή καρτών προϊόντων. Κάθε κάρτα περιλαμβάνει πληροφορίες όπως η εικόνα του προϊόντος, η τιμή, ο τίτλος και, εφόσον υπάρχουν, κάποια προσφορά του προϊόντος.

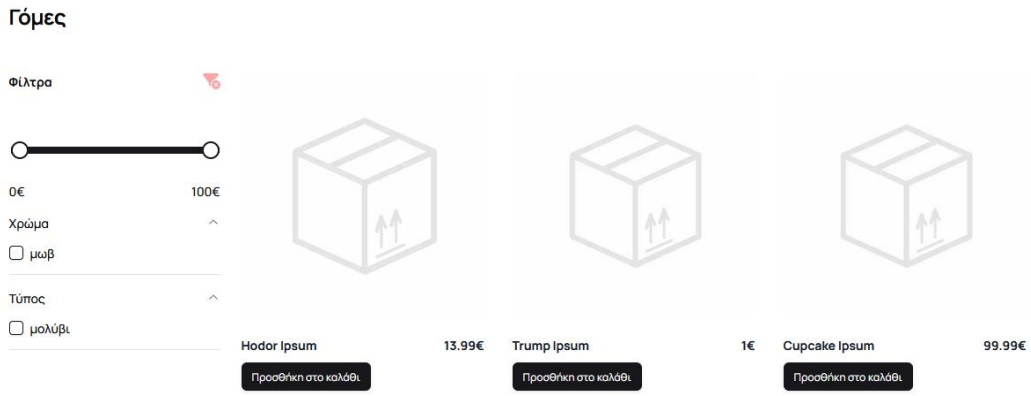
Για τη βελτίωση της εμπειρίας χρήστη και της δυνατότητας αναζήτησης, έχουν ενσωματωθεί μηχανισμοί φιλτραρίσματος (filters). Οι επιλογές φίλτρων εμφανίζονται συνήθως στην αριστερή ή επάνω πλευρά της σελίδας και περιλαμβάνουν:

- **Εύρος τιμής**: Παρέχεται δυνατότητα καθορισμού ελάχιστης και μέγιστης τιμής με slider ή πεδία αριθμών, επιτρέποντας στους χρήστες να περιορίσουν τα αποτελέσματα ανάλογα με τον προϋπολογισμό τους.

- **Τύπος προϊόντος**: Δίνεται η δυνατότητα επιλογής ανάμεσα σε διαφορετικούς τύπους προϊόντων.

- **Χρώμα**: Ο χρήστης μπορεί να φιλτράρει τα προϊόντα βάσει διαθέσιμων χρωμάτων, προσφέροντας πιο στοχευμένα αποτελέσματα.

- **Άλλες ιδιότητες**: Ανάλογα με τα δεδομένα του CMS, ενδέχεται να προστεθούν επιπλέον φίλτρα.



Εικόνα 3.17: Φίλτρα προϊόντων ανά κατηγορία

Η λειτουργία των φίλτρων έχει υλοποιηθεί με τρόπο ώστε να μην απαιτείται επαναφόρτωση της σελίδας. Αντιθέτως, οι επιλογές φιλτραρίσματος προκαλούν δυναμική ανάκτηση των προϊόντων που ικανοποιούν τα αντίστοιχα κριτήρια (μέσω client-side αιτημάτων σε GraphQL endpoint), επιτυγχάνοντας μια πιο διαδραστική και ταχεία εμπειρία για τον επισκέπτη.

Η σελίδα των προϊόντων ανά κατηγορία έχει σχεδιαστεί με γνώμονα τόσο την αισθητική όσο και τη χρηστικότητα, με στόχο να βοηθήσει τον χρήστη να ανακαλύψει εύκολα αυτό που ψάχνει, χωρίς να χαθεί σε ένα μεγάλο εύρος αποτελεσμάτων.

Ο παραπάνω κώδικας υλοποιεί την κεντρική λειτουργία της σελίδας Product Category, δηλαδή της προβολής προϊόντων που ανήκουν σε συγκεκριμένη κατηγορία, με υποστήριξη φιλτραρίσματος. Η συνάρτηση ProductCategory δέχεται ως παραμέτρους το slug της τρέχουσας κατηγορίας και τα searchParams, δηλαδή τις παραμέτρους φίλτρων που περιλαμβάνονται στο URL.

```
/**
 * Renders the product category page with filters and a product catalog.
 *
 * @param {Object} props - Component props.
 * @param {string} props.currentSlug - The slug of the current product category.
```

```
 * @param {Object} props.searchParams - Search parameters from the URL (e.g., filters, price range).
 * @returns {JSX.Element} The rendered product category section.
 *
 * This component:
 * - Fetches category details, filtered product attributes, and all product prices.
 * - Computes absolute min and max prices for use in price range filters.
 * - Displays a mobile and desktop filter UI.
 * - Renders a paginated product catalog.
 */
export const dynamic = "force-dynamic";
export async function ProductCategory({ currentSlug, searchParams }) {
  const [attribute, attributeTerm] =
    Object.entries(searchParams || {})[0] || [];
  const minPriceParam = searchParams?.min_price;
  const maxPriceParam = searchParams?.max_price;
  const minPrice = minPriceParam ? parseFloat(minPriceParam) : undefined;
  const maxPrice = maxPriceParam ? parseFloat(maxPriceParam) : undefined;
  /** Get only the title of the product category since you can't get it elsewhere. */
  const { data } = await query({
    query: PRODUCT_CATEGORY,
    variables: { id: currentSlug },
  });
  /** This is are the products from the product catalog that we manipulate
   *  with the attributes. */
  const { data: productAttributes } = await query({
    query: PRODCUTS_ATTRIBUTES,
    variables: {
      category: currentSlug,
```

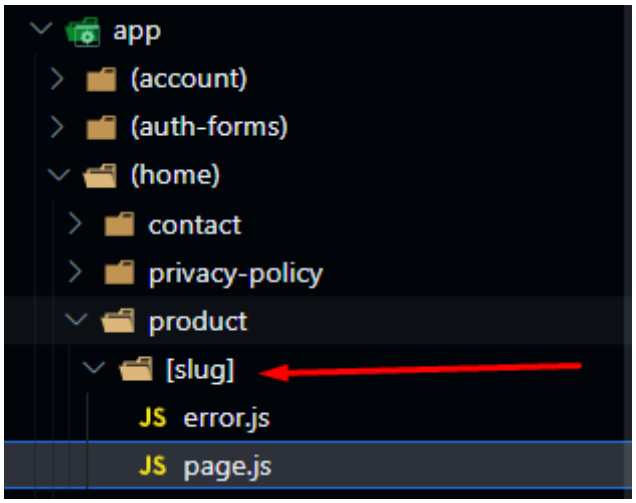
```
    attribute,
    attributeTerm,
    minPrice,
    maxPrice,
    first: 1000,
    after: null,
  },
});
/** Get all the prices from the products */
const { data: priceData } = await query({
  query: ALL_PRODUCT_PRICES,
  variables: { category: currentSlug },
});
/** Map through all the products prices. */
const allPrices =
  priceData?.products?.nodes?.map((p) => parseFloat(p.price)) || [];
/** Find the minPrice and the maxPrice from all the products prices. */
const absoluteMinPrice = Math.min(...allPrices);
const absoluteMaxPrice = Math.max(...allPrices);
```

Αρχικά, γίνεται ανάκτηση των στοιχείων της επιλεγμένης κατηγορίας με χρήση GraphQL query, καθώς και των διαθέσιμων προϊόντων, βάσει φίλτρων όπως είδος χαρακτηριστικού, εύρος τιμής, κ.λπ. Παράλληλα, πραγματοποιείται ξεχωριστό query για την ανάκτηση όλων των τιμών των προϊόντων της συγκεκριμένης κατηγορίας, ώστε να προσδιοριστούν δυναμικά τα απόλυτα ελάχιστη και μέγιστη τιμή, οι οποίες χρησιμοποιούνται στο εύρος τιμής.

Η σελίδα υποστηρίζει τόσο σε υπολογιστή όσο και σε μικρότερες συσκευές περιβάλλον, με διαφορετική απεικόνιση του UI των φίλτρων ανά συσκευή. Τα προϊόντα εμφανίζονται με δυνατότητα σελιδοποίησης (pagination), ενώ τα φίλτρα εφαρμόζονται σε πραγματικό χρόνο μέσω του searchParams, χωρίς να απαιτείται ανανέωση της σελίδας. Με αυτόν τον τρόπο εξασφαλίζεται μια διαδραστική και ομαλή εμπειρία χρήσης κατά την αναζήτηση προϊόντων.

3.6 Σελίδα μοναδικού προϊόντος

Η σελίδα προβολής ενός μοναδικού προϊόντος αποτελεί βασικό σημείο στην εμπειρία του χρήστη σε ένα ηλεκτρονικό κατάστημα. Η υλοποίηση αυτής της σελίδας στο παρόν έργο περιλαμβάνει όλες τις απαραίτητες πληροφορίες και λειτουργίες που σχετίζονται με ένα συγκεκριμένο προϊόν.



Εικόνα 3.18: Δυναμική διαδρομή σελίδας για το SingleProduct

Επίσης στον παρακάτω κώδικα είναι ένα δείγμα ενός query για τον SingleProduct για να πάρουμε τις τιμές από GraphQL.

```
import gql from "graphql-tag";

export const SINGLE_ITEM_QUERY = gql`

  query SINGLE_ITEM_QUERY($id: ID!) {

    product(id: $id, idType: SLUG) {

      name

      databaseId

      description(format: RAW)

      reviewCount

      productCategories {

        nodes {

          name
```

```
  }
}
... on SimpleProduct {
  id: databaseId # Same with the category so it can understand the id and not the string
  name
  price(format: RAW)
  salePrice(format: RAW)
  regularPrice(format: RAW)
  onSale
  stockStatus
  commentCount
  comments(first: 10) {
    nodes {
      date
      content
      author {
        node {
          name
          avatar {
            url
          }
        }
      }
    }
  }
  galleryImages {
    nodes {
      sourceUrl
```

```
      }
     }
    }
    image {
     altText
     sourceUrl
    }
   }
  }
`;
```

Το SINGLE_ITEM_QUERY είναι ένα GraphQL query που ζητά αναλυτικά στοιχεία για ένα συγκεκριμένο προϊόν βάσει του slug του. Επιστρέφει πληροφορίες όπως όνομα, τιμές, περιγραφή, κατάσταση αποθέματος, εικόνες, κατηγορίες και τα τελευταία σχόλια με λεπτομέρειες για τον συγγραφέα και την ημερομηνία. Χρησιμοποιείται για την εμφάνιση της σελίδας ενός μεμονωμένου προϊόντος.

Πιο αναλυτικά, η σελίδα περιλαμβάνει:

- **Κεντρική εικόνα** του προϊόντος, η οποία προβάλλεται σε μεγάλο μέγεθος για να εστιάσει την προσοχή του χρήστη.

- **Gallery εικόνων**: Αν το προϊόν διαθέτει περισσότερες από μία εικόνες, τότε εμφανίζεται και σχετικό carousel ή slider για την εναλλαγή τους, προσφέροντας καλύτερη οπτική αναπαράσταση του προϊόντος.

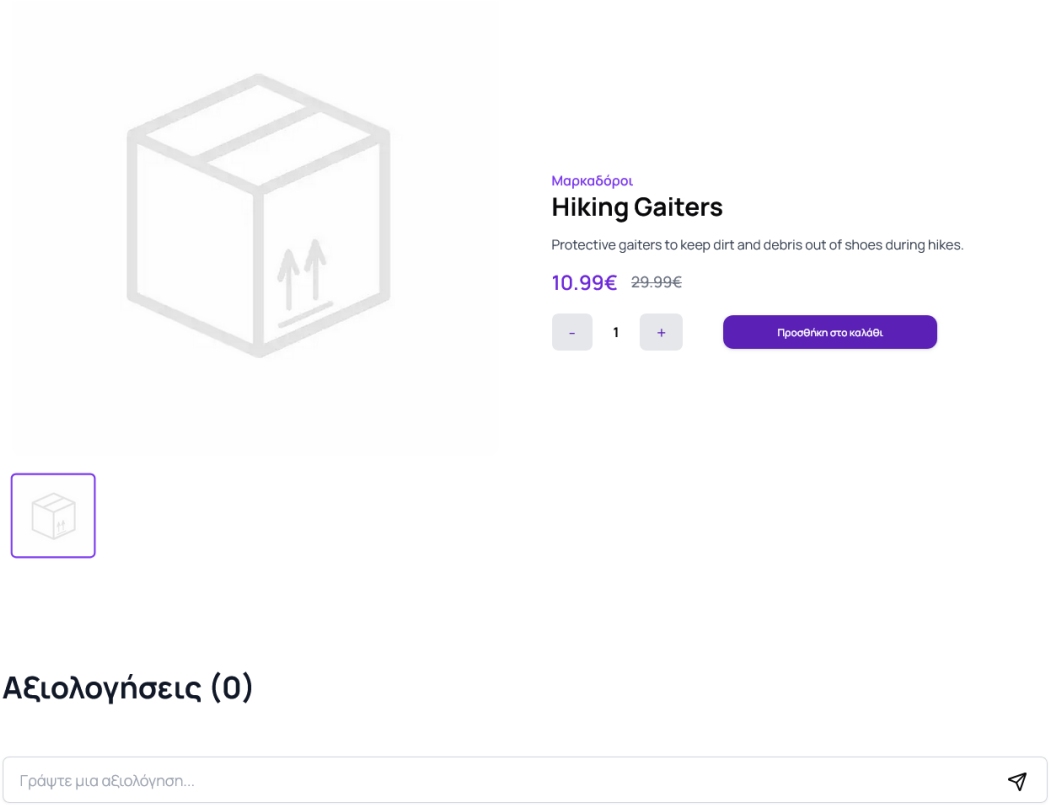
- **Περιγραφή** του προϊόντος με αναλυτικά χαρακτηριστικά, πληροφορίες για τα υλικά, τον τρόπο χρήσης ή άλλες σχετικές λεπτομέρειες.

- **Σχόλια** χρηστών: Οι χρήστες μπορούν να δουν σχόλια από άλλους πελάτες, τα οποία εμφανίζονται ταξινομημένα, συνήθως από τα πιο πρόσφατα προς τα παλαιότερα.

- **Προσθήκη στο καλάθι**: Παρέχεται η δυνατότητα προσθήκης του προϊόντος στο καλάθι αγορών, μέσω κατάλληλου κουμπιού, ώστε να διευκολυνθεί η διαδικασία αγοράς καθώς και

την ποσότητα που θέλει ο χρήστης να βάλει.

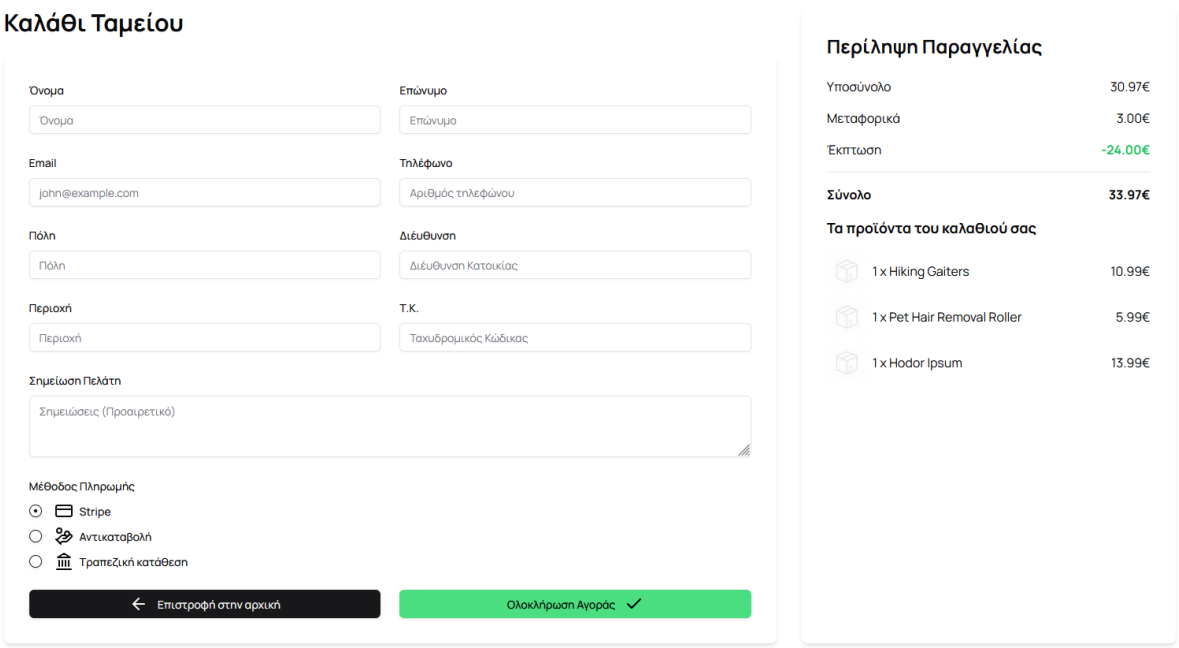
- **Δυνατότητα υποβολής σχολίου**: Αν ο επισκέπτης είναι συνδεδεμένος με τον λογαριασμό του, εμφανίζεται ένα πεδίο μέσω του οποίου μπορεί να προσθέσει το δικό του σχόλιο.



Εικόνα 3.19: Προβολή μοναδικού προϊόντος

3.7 Ολοκλήρωση αγοράς

Η σελίδα ολοκλήρωσης αγοράς αποτελεί το τελικό βήμα της αγοραστικής εμπειρίας του χρήστη, όπου συμπληρώνει μια φόρμα με τα απαραίτητα προσωπικά και στοιχεία αποστολής, καθώς και τη μέθοδο πληρωμής. Η φόρμα είναι υλοποιημένη ως React component και αξιοποιεί τη βιβλιοθήκη react-hook-form σε συνδυασμό με το Zod για έλεγχο εγκυρότητας (validation) των πεδίων.



Εικόνα 3.20: Φόρμα ολοκλήρωσης παραγγελίας

- **Υλοποίηση Πληρωμών**
  - **Stripe (Ηλεκτρονική πληρωμή):**
    Όταν ο χρήστης επιλέξει την επιλογή Stripe, τότε:
    - Δημιουργείται πρώτα η παραγγελία με fetch("/api/create_order"), η οποία στέλνει τα στοιχεία του χρήστη και το καλάθι.
    - Στη συνέχεια, με το fetch("/api/checkout_sessions"), δημιουργείται ένα session με το REST API του WooCommerce ώστε να συνδεθεί με την υπηρεσία Stripe.
    - Ο χρήστης ανακατευθύνεται αυτόματα στο Stripe Checkout μέσω του URL που επιστρέφεται.

```
let orderData;

    if (CheckoutFormSchema.paymentMethod === "stripe") {

     // 1. Request a Stripe Checkout session

     // const result = await handleCheckout(CheckoutFormSchema);

     orderData = getCreateOrderData(CheckoutFormSchema, cart);
```

```
    const orderResponse = await fetch("/api/create_order", {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify(orderData), // Send form data
    });
    const orderInfo = await orderResponse.json();


    const response = await fetch("/api/checkout_sessions", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({
        email: CheckoutFormSchema.email,
        products: cart,
        orderId: orderInfo.orderId,
      }),
    });
    const session = await response.json();
    if (session.url) {
      // 2. Redirect user to Stripe Checkout
      window.location.href = session.url;
      return;
    } else {
      throw new Error("Failed to create Stripe checkout session.");
    }
  }
```

Αυτό το απόσπασμα κώδικα χειρίζεται την περίπτωση όπου ο χρήστης επιλέγει ως μέθοδο πληρωμής τη Stripe. Αρχικά, δημιουργεί τα δεδομένα της παραγγελίας (orderData) και τα στέλνει σε ένα backend endpoint (/api/create_order) για να δημιουργηθεί η αντίστοιχη παραγγελία στο WooCommerce. Έπειτα, στέλνει αίτημα στο /api/checkout_sessions με τα προϊόντα, το email του χρήστη και το ID της παραγγελίας, ώστε να δημιουργηθεί ένα Stripe Checkout session. Αν η διαδικασία είναι επιτυχής, γίνεται αυτόματη ανακατεύθυνση του χρήστη στο περιβάλλον πληρωμής της Stripe.

- **Αντικαταβολή και Τραπεζική Κατάθεση:**
  Για αυτές τις επιλογές, η παραγγελία ολοκληρώνεται άμεσα μέσω handleCheckout() με GraphQL mutation στο backend του WooCommerce. Η παραγγελία αποθηκεύεται και ο χρήστης μεταφέρεται στη σελίδα επιβεβαίωσης με orderId.

```
// 3. Handle other payment methods normally
    const result = await handleCheckout(CheckoutFormSchema);
    if (result?.success) {
      setformSuccess("Η παραγγελία σας ολοκληρώθηκε");
      localStorage.removeItem("cart");
      window.location.href = `/success?orderId=${result?.data?.checkout?.order?.databaseId}`;
    } else {
      setGlobalError(result.message);
    }
  } catch (error) {
    console.error("Error processing checkout", error);
    setGlobalError("Αποτυχία πληρωμής. Παρακαλώ προσπαθήστε ξανά.");
  }
```

Επιπλέον, σε περίπτωση επιτυχούς ολοκλήρωσης της πληρωμής, εμφανίζεται σχετική επιβεβαίωση στον χρήστη και το καλάθι αδειάζει αυτόματα, ώστε να ολοκληρωθεί σωστά η ροή της παραγγελίας. Αντιθέτως, σε περίπτωση αποτυχίας, εμφανίζεται κατάλληλο μήνυμα σφάλματος που ενημερώνει τον χρήστη για το πρόβλημα που προέκυψε. Όλα τα δεδομένα που σχετίζονται με τη διαδικασία πληρωμής και την παραγγελία ελέγχονται αυστηρά μέσω validation schema που έχει οριστεί με χρήση της βιβλιοθήκης Zod, διασφαλίζοντας ότι η πληροφορία που υποβάλλεται είναι σωστά δομημένη και έγκυρη πριν προχωρήσει οποιαδήποτε ενέργεια.

Η υλοποίηση του συστήματος πληρωμών στην εφαρμογή βασίζεται στην υπηρεσία Stripe και πραγματοποιείται μέσω ενός webhook μηχανισμού. Πιο συγκεκριμένα, οι συναλλαγές γίνονται προς το παρόν σε δοκιμαστικό περιβάλλον (sandbox) και εξομοιώνονται σε τοπική ανάπτυξη (localhost), επιτρέποντας την ελεγχόμενη αξιολόγηση της ροής πληρωμής πριν μεταφερθεί σε παραγωγικό περιβάλλον.

Κατά την έναρξη της πληρωμής, ο χρήστης μεταφέρεται σε ένα Stripe Checkout session, το οποίο δημιουργείται δυναμικά μέσω ενός POST endpoint. Σε αυτό το session αποστέλλονται βασικές πληροφορίες, όπως η διεύθυνση email του πελάτη, τα προϊόντα του καλαθιού, και το μοναδικό αναγνωριστικό της παραγγελίας από το WooCommerce.

```
export async function POST(req) {
 const rawBody = await req.text();
 const signature = headers().get("Stripe-Signature");
 let event;
 try {
  event = stripe.webhooks.constructEvent(
   rawBody,
   signature,
   process.env.STRIPE_WEBHOOK_SECRET
  );
 } catch (error) {
  console.error("Webhook signature verification failed:", error.message);
  return new NextResponse("Invalid signature", { status: 400 });
 }
 const session = event.data.object;
 if (event.type === "checkout.session.completed") {
  const orderId = session.metadata?.order_id;
  if (orderId) {
   try {
    await updateWooOrderStatus(orderId, "processing", session.id);
```

```
    } catch (error) {
      console.error("Failed to update WooCommerce order:", error);
    }
  } else {
    console.warn("No order ID found in session metadata.");
  }
 }
 return new NextResponse("ok", { status: 200 });
}
```

Η επεξεργασία της συναλλαγής παρακολουθείται μέσω ενός webhook που ακούει για το γεγονός checkout.session.completed. Μόλις η Stripe ενημερώσει για την επιτυχή ολοκλήρωση, γίνεται επαλήθευση του αιτήματος χρησιμοποιώντας την υπογραφή της Stripe και το μυστικό κλειδί που έχει αποθηκευτεί στο αρχείο περιβάλλοντος. Αν η υπογραφή είναι έγκυρη, ανακτάται το order_id από τα metadata και ενημερώνεται η παραγγελία στο WooCommerce ώστε να χαρακτηριστεί ως "processing", δηλαδή σε φάση επεξεργασίας.

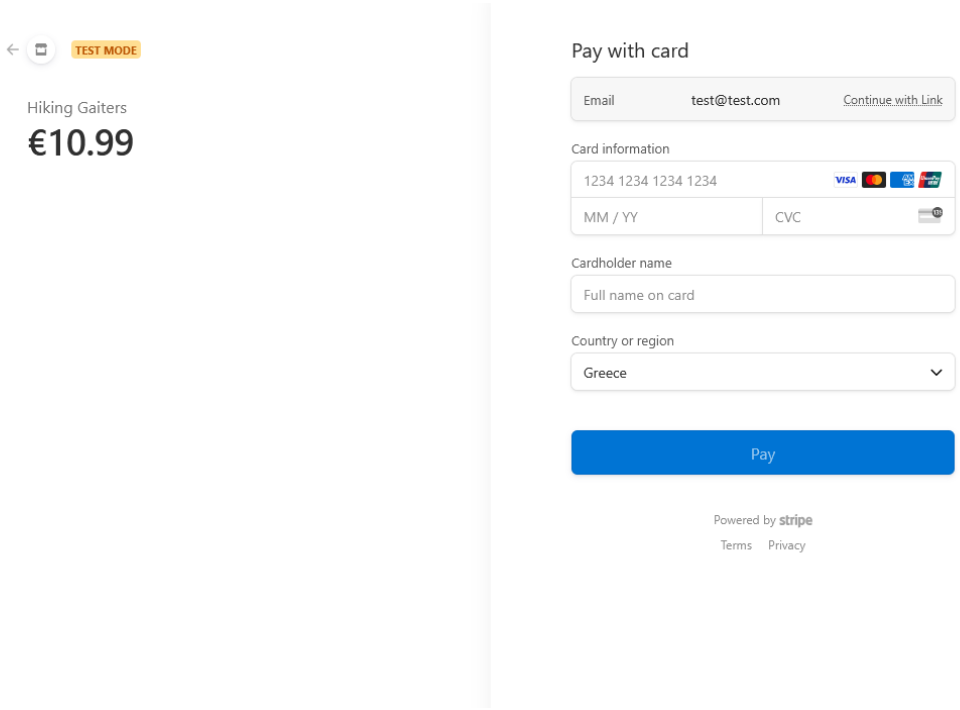
Παρακάτω παρουσιάζεται συνοπτικά η διαδικασία:

1. Ο χρήστης επιλέγει πληρωμή, και καλείται ένα backend endpoint που δημιουργεί ένα checkout session στη Stripe.

2. Αν όλα πάνε καλά, επιστρέφεται το URL της Stripe και ο χρήστης ανακατευθύνεται για την πληρωμή.

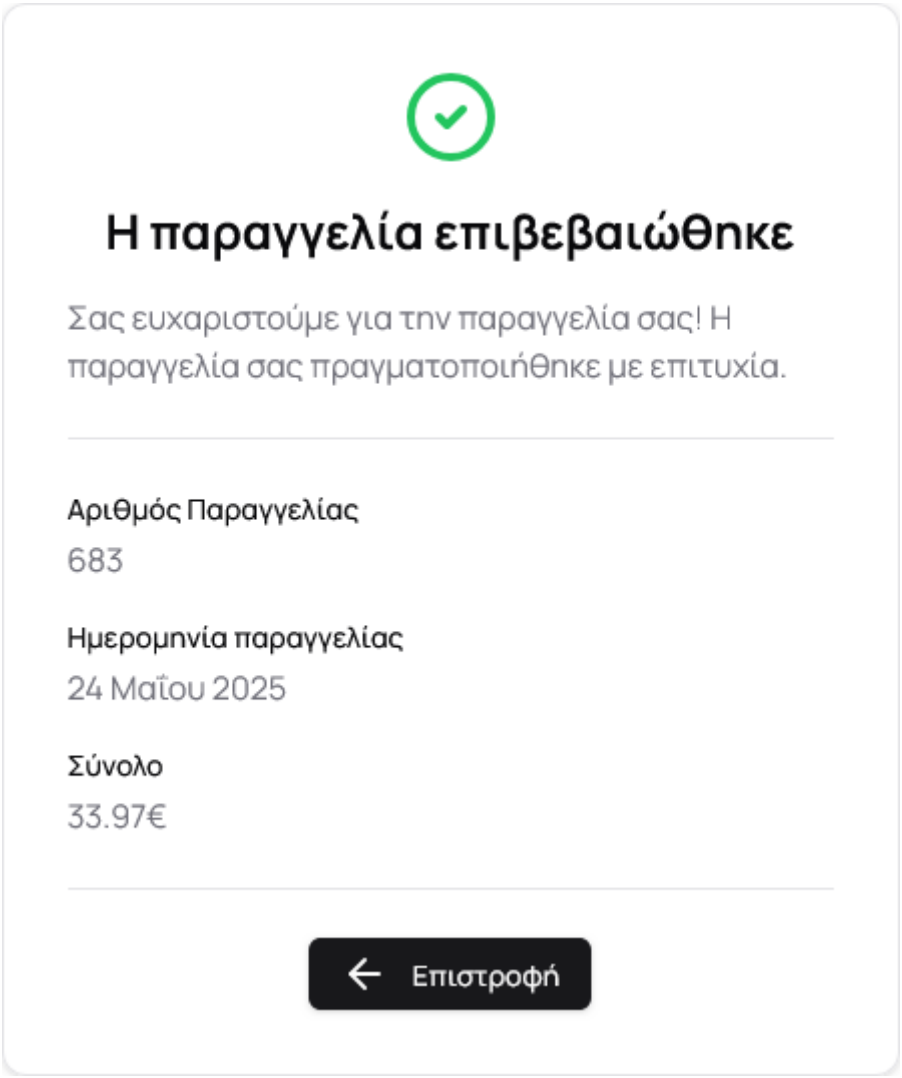
3. Μετά την επιτυχία, η Stripe στέλνει webhook στον server.

4. Ο server επαληθεύει το event και ενημερώνει το WooCommerce με το κατάλληλο status για την παραγγελία.



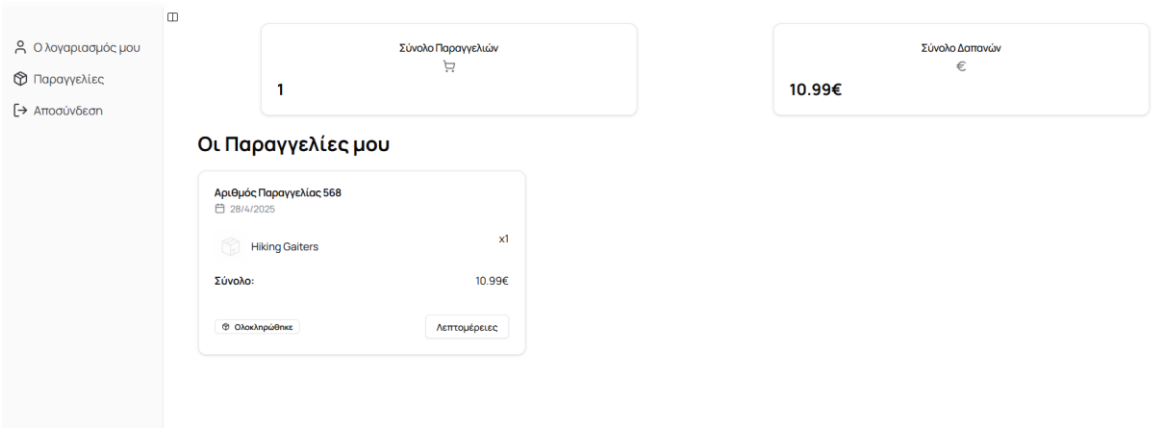
Εικόνα 3.21: Ασφαλές περιβάλλον ηλεκτρονικής πληρωμής Stripe

Σε περίπτωση επιτυχίας, ο χρήστης βλέπει μήνυμα επιβεβαίωσης και το καλάθι του αδειάζει αυτόματα. Αν υπάρξει κάποιο σφάλμα, εμφανίζεται σχετικό μήνυμα αποτυχίας στην οθόνη, προκειμένου ο χρήστης να ενημερωθεί άμεσα.



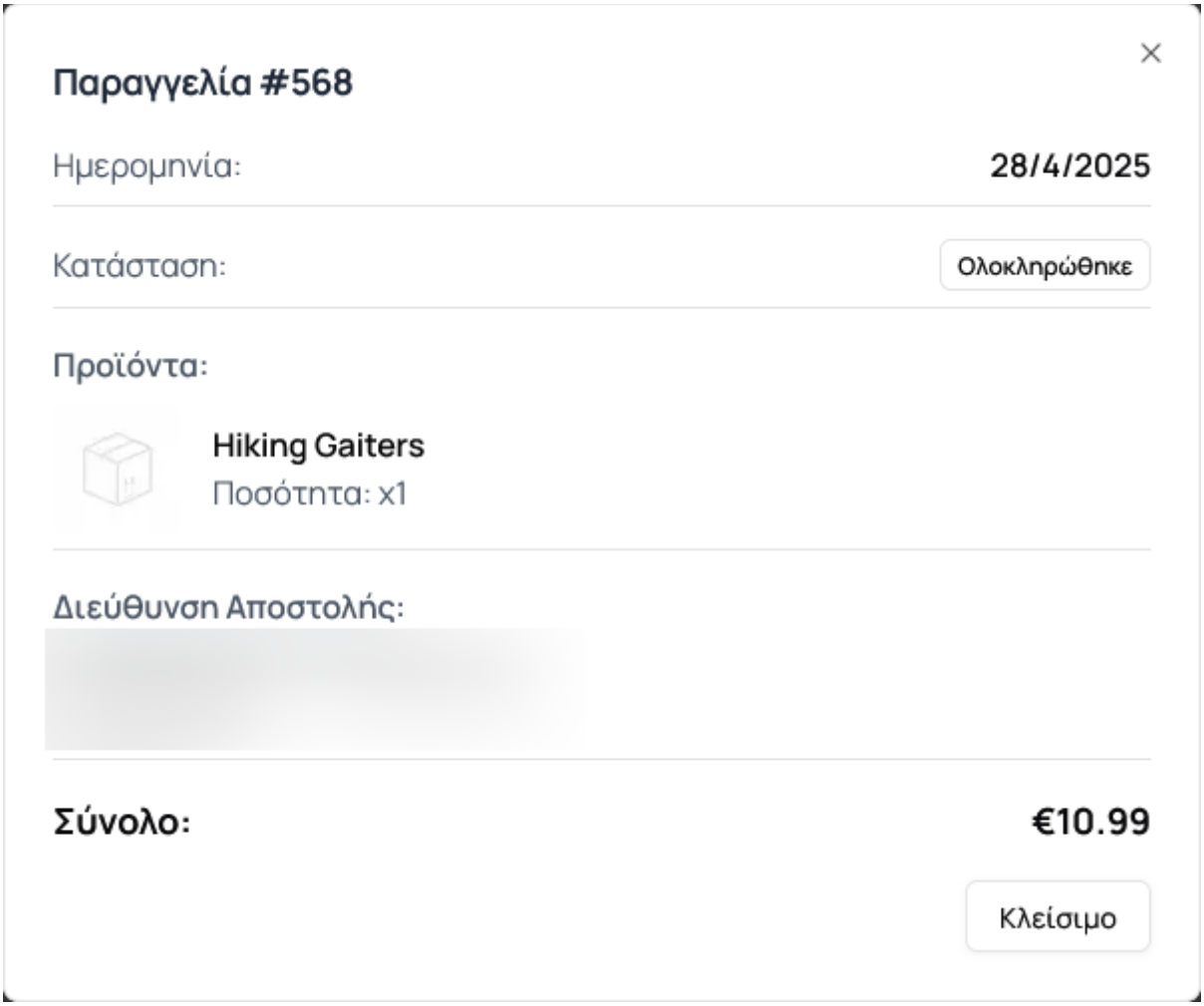
Εικόνα 3.22: Μήνυμα επιβεβαίωσης παραγγελίας

Με την επιτυχή ολοκλήρωση μιας παραγγελίας, το σύστημα ενημερώνει τον χρήστη μέσω κατάλληλου μηνύματος, το οποίο τον διαβεβαιώνει ότι η διαδικασία καταχώρισης πραγματοποιήθηκε χωρίς σφάλματα. Πέραν του βασικού μηνύματος επιβεβαίωσης, παρέχεται και επιπλέον πληροφόρηση, με στόχο τη βελτίωση της εμπειρίας χρήσης και την ενίσχυση της διαφάνειας στην παρακολούθηση των αγορών.



Εικόνα 3.23: Οι παραγγελίες του εγγεγραμμένου χρήστη

Πιο συγκεκριμένα, στην περίπτωση που ο χρήστης έχει προβεί στην παραγγελία όντας συνδεδεμένος στον προσωπικό του λογαριασμό, του δίνεται η δυνατότητα να μεταβεί στη σελίδα "Οι Παραγγελίες μου".



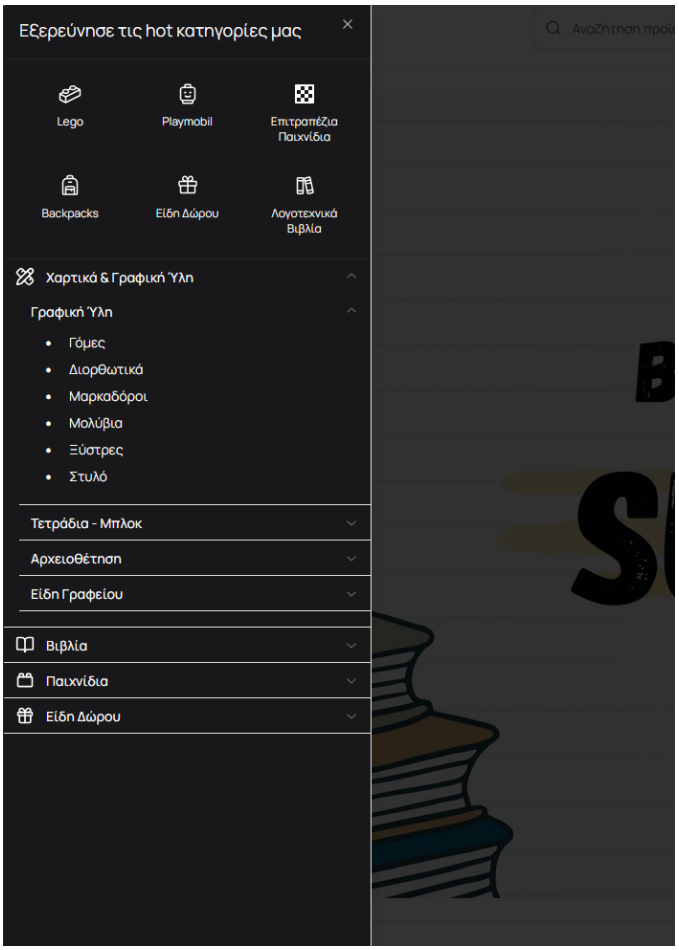
Εικόνα 3.24: Λεπτομέρειες παραγγελίας

Κεφάλαιο 4ο: Παρουσίαση και Λειτουργία της Εφαρμογής

Σε αυτό το κεφάλαιο περιγράφεται αναλυτικά η λειτουργία της εφαρμογής που υλοποιήθηκε, εστιάζοντας τόσο στη ροή που ακολουθεί ο χρήστης, όσο και στις τεχνολογίες που υποστηρίζουν τη διαχείριση και παρουσίαση των δεδομένων.

4.1 Αρχική σελίδα

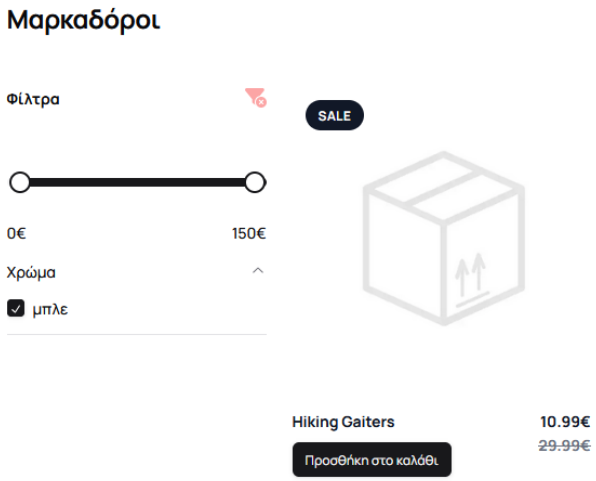
Όταν ο χρήστης επισκέπτεται για πρώτη φορά την εφαρμογή, αντικρίζει την αρχική σελίδα η οποία έχει σχεδιαστεί με τρόπο φιλικό και ευχάριστο. Το μενού πλοήγησης στο πάνω μέρος επιτρέπει τη γρήγορη πρόσβαση σε βασικές ενότητες όπως τα προϊόντα, το καλάθι και η φόρμα επικοινωνίας, ενώ δίνεται η δυνατότητα εγγραφής ή σύνδεσης σε προσωπικό λογαριασμό. Στο κεντρικό μέρος της σελίδας εμφανίζονται επιλεγμένα προϊόντα, τα οποία παρουσιάζονται με φωτογραφίες, τίτλους, σύντομες περιγραφές και την τιμή τους. Η γενική εμπειρία έχει σχεδιαστεί ώστε ο χρήστης να μπορεί άμεσα να περιηγηθεί στο κατάστημα, χωρίς να χαθεί ή να χρειάζεται ιδιαίτερη εξοικείωση.



Εικόνα 4.1: Επιλογές του μενού στην αρχική οθόνη

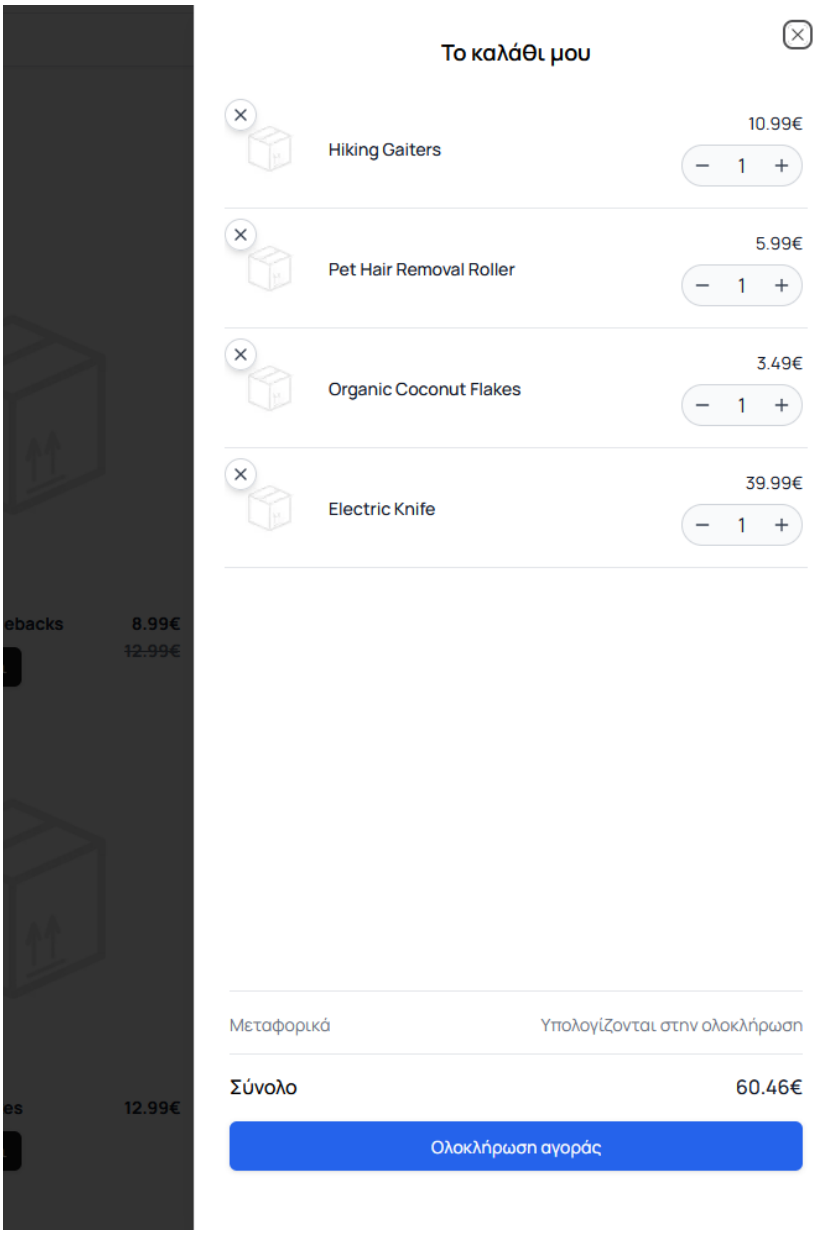
4.2 Περιήγηση σε προϊόντα και λειτουργία του καλαθιού

Περιηγούμενος στο κατάστημα, ο χρήστης έχει τη δυνατότητα να εξερευνήσει όλα τα διαθέσιμα προϊόντα ανά κατηγορία, με φιλτράρισμα ανά τιμή ή άλλες παραμέτρους.



Εικόνα 4.2: Κατηγορία προϊόντων με επιλεγμένο φίλτρο

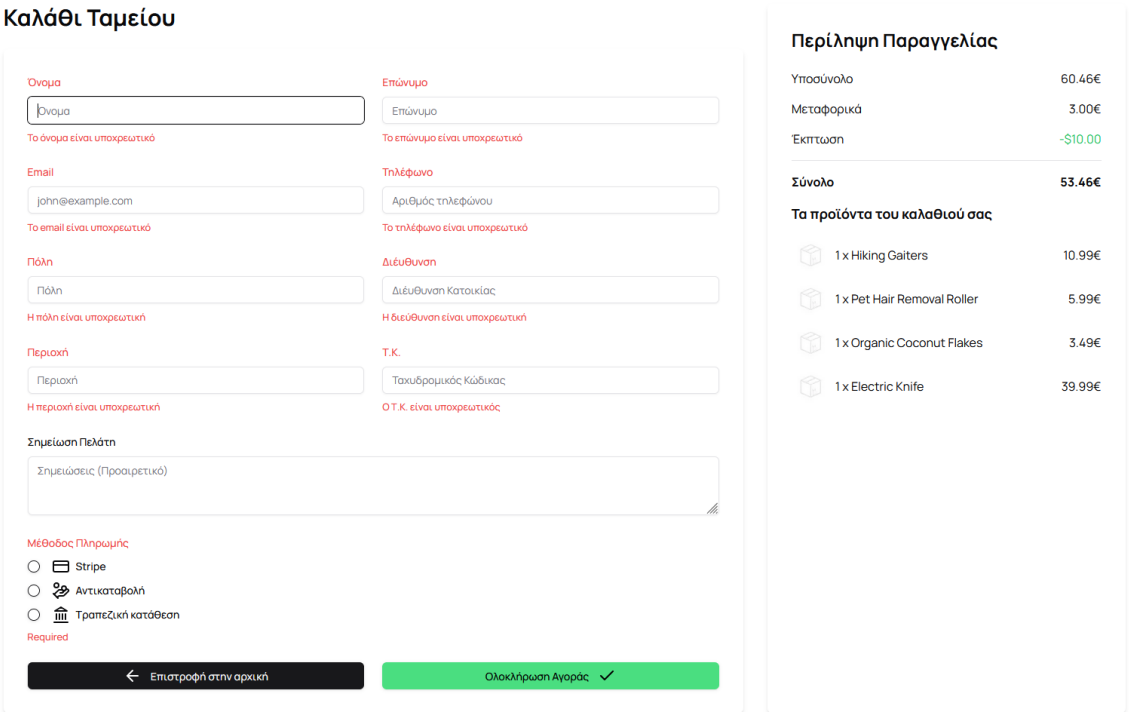
Πατώντας σε ένα προϊόν, μεταφέρεται σε μια ειδική σελίδα όπου παρουσιάζονται περισσότερες λεπτομέρειες. Εκεί ο χρήστης μπορεί να προσθέσει το προϊόν στο καλάθι του. Η λειτουργία του καλαθιού είναι διαδραστική και απλή. Ο χρήστης μπορεί να δει όλα τα είδη που έχει προσθέσει, να αλλάξει ποσότητες ή να αφαιρέσει προϊόντα, ενώ παράλληλα ενημερώνεται αυτόματα το συνολικό κόστος, χωρίς να χρειάζεται ανανέωση της σελίδας.



Εικόνα 4.3: Προβολή καλαθιού

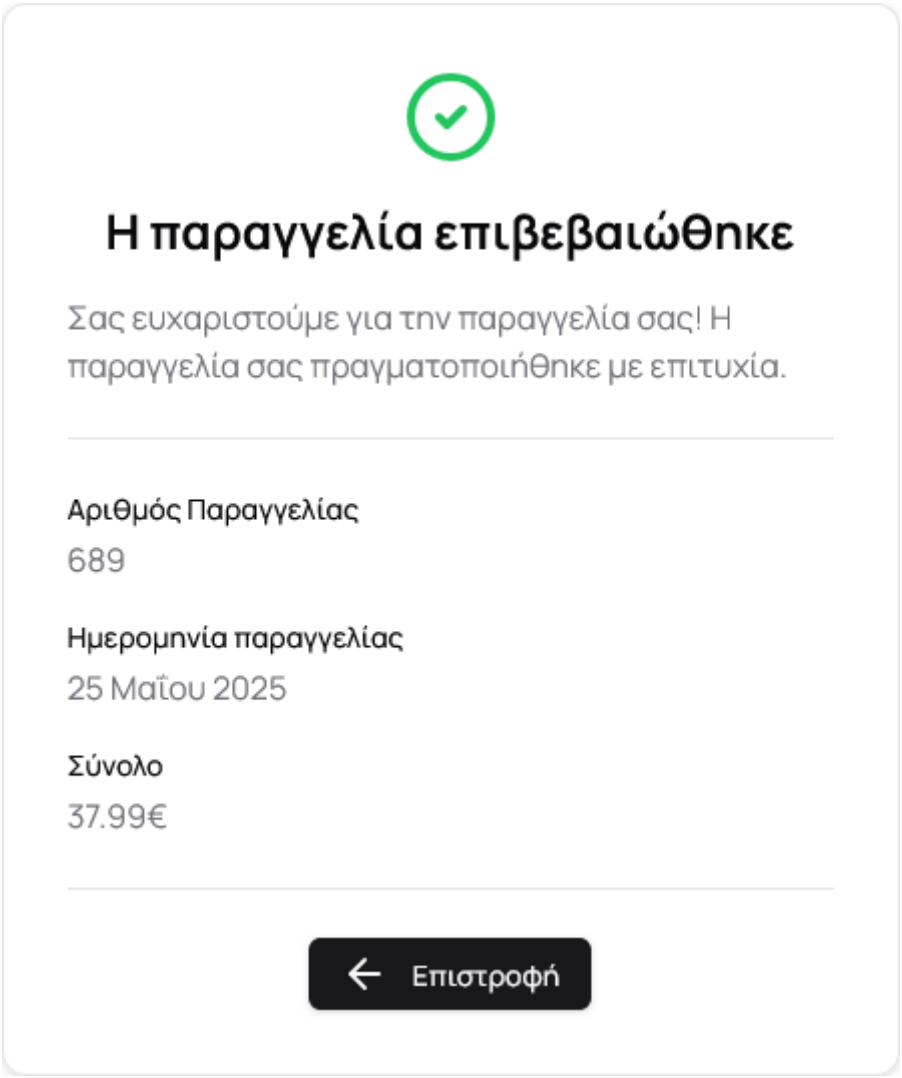
4.3 Ολοκλήρωση παραγγελίας

Όταν ο χρήστης αποφασίσει να προχωρήσει στην αγορά, οδηγείται στο στάδιο της παραγγελίας. Εκεί εισάγει τα προσωπικά του στοιχεία, όπως ονοματεπώνυμο, διεύθυνση αποστολής και στοιχεία επικοινωνίας.



Εικόνα 4.4: Ολοκλήρωση αγοράς με μηνύματα λάθους

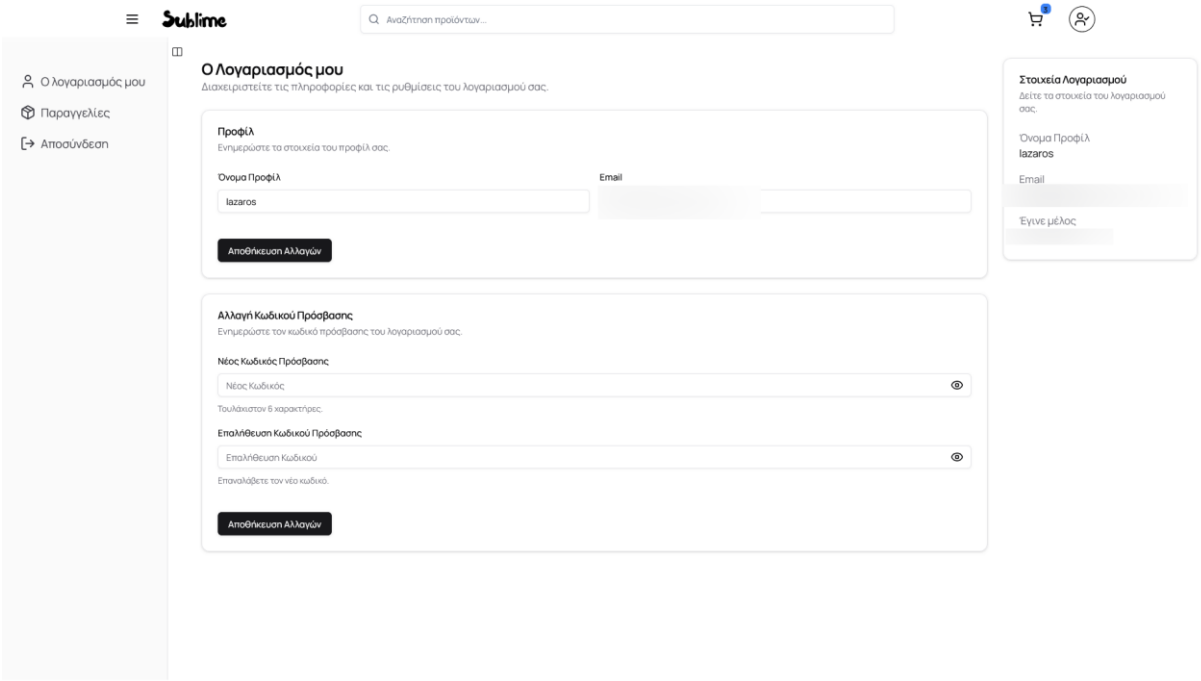
Η εφαρμογή προσφέρει πολλαπλές επιλογές πληρωμής, ανάλογα με τις ανάγκες του πελάτη. Ο χρήστης μπορεί να επιλέξει αντικαταβολή, κατάθεση σε τραπεζικό λογαριασμό ή πληρωμή με κάρτα μέσω του ασφαλούς συστήματος Stripe. Μετά την ολοκλήρωση της πληρωμής ή της υποβολής της παραγγελίας, εμφανίζεται ένα μήνυμα επιβεβαίωσης, ενώ σε περίπτωση που ο χρήστης είναι συνδεδεμένος, η παραγγελία του αποθηκεύεται στο προσωπικό του ιστορικό για μελλοντική αναφορά.



Εικόνα 4.5: Επιτυχής μήνυμα δημιουργίας παραγγελίας

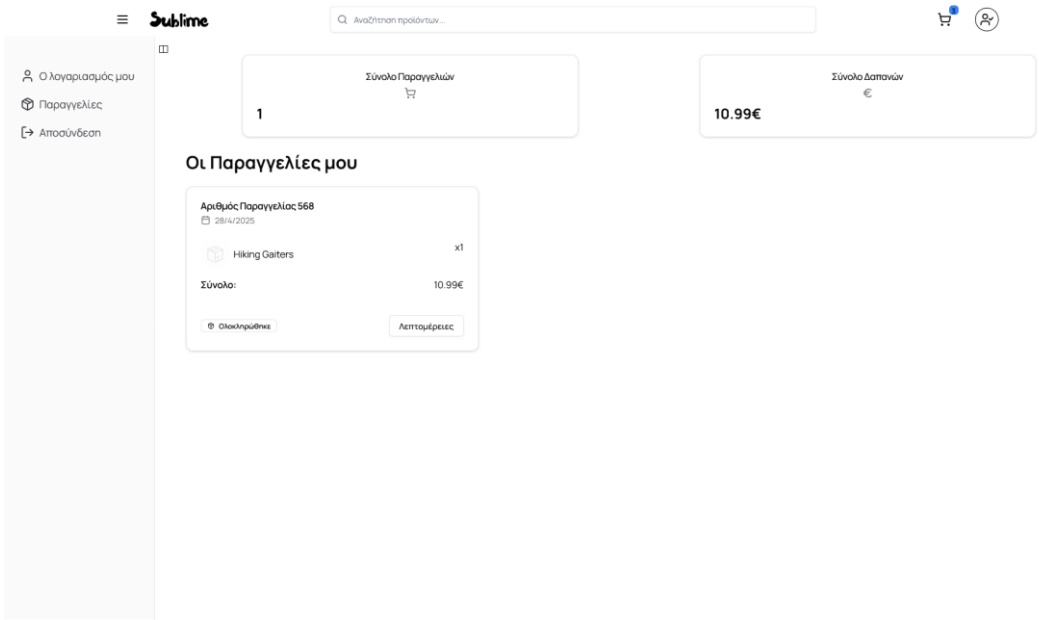
4.4 Προσωπικός λογαριασμός και ιστορικό παραγγελιών

Οι εγγεγραμμένοι χρήστες της εφαρμογής έχουν πρόσβαση σε έναν ειδικά διαμορφωμένο χώρο όπου μπορούν να δουν και να διαχειριστούν τις παραγγελίες τους. Εφόσον εισέλθουν μέσα στην εφαρμογή μπορούν να δουν τα στοιχεία του λογαριασμού τους και τους δίνεται επίσης η δυνατότητα να αλλάξουν κωδικό πρόσβασης, email και όνομα προφίλ.



Εικόνα 4.6: Αλλαγές προσωπικών πληροφοριών συνδεδεμένου χρήστη

Ενώ εάν μεταβεί στις παραγγελίες θα εμφανίζονται με χρονολογική σειρά όλες οι προηγούμενες αγορές τους, μαζί με λεπτομέρειες όπως η ημερομηνία καταχώρησης, τα προϊόντα που περιλαμβάνει η παραγγελία, το συνολικό κόστος και η κατάσταση αποστολής της. Η δυνατότητα αυτή προσφέρει στον χρήστη έναν εύκολο τρόπο να παρακολουθεί το ιστορικό των αγορών του και να ενημερώνεται για την πορεία κάθε παραγγελίας. Εφόσον υπάρξει ενημέρωση από το σύστημα ή από τον διαχειριστή, αυτή αποτυπώνεται σε πραγματικό χρόνο μέσα στον λογαριασμό του χρήστη.



Εικόνα 4.7: Παραγγελίες του συνδεδεμένου χρήστη

Κεφάλαιο 5ο: Συμπεράσματα και προτάσεις βελτιώσεις εφαρμογής

Η υλοποίηση της εφαρμογής αυτής αποτέλεσε μια ουσιαστική εμπειρία τόσο σε τεχνικό όσο και σε σχεδιαστικό επίπεδο. Ο βασικός στόχος ήταν να δημιουργηθεί ένα λειτουργικό και φιλικό προς τον χρήστη περιβάλλον, μέσα από το οποίο ο χρήστης θα μπορούσε να περιηγηθεί στα προϊόντα, να δημιουργήσει το καλάθι του και να ολοκληρώσει μια παραγγελία εύκολα και με ασφάλεια. Η διαδικασία από την επιλογή των προϊόντων μέχρι και την πληρωμή μέσω Stripe ή άλλων τρόπων, όπως η αντικαταβολή ή η τραπεζική κατάθεση, σχεδιάστηκε με τρόπο ώστε να είναι απλή και ξεκάθαρη.

Ιδιαίτερη έμφαση δόθηκε στην εμπειρία του χρήστη μετά την αγορά. Μέσα από τον λογαριασμό του, μπορεί να παρακολουθεί όλες τις προηγούμενες παραγγελίες του, γεγονός που ενισχύει την αίσθηση αξιοπιστίας και ελέγχου. Το κομμάτι αυτό απαιτούσε προσεκτική διαχείριση, ειδικά ως προς την εμφάνιση των σωστών δεδομένων μόνο στον συνδεδεμένο χρήστη, κάτι που επιτεύχθηκε με κατάλληλη χρήση session cookies και backend ελέγχου.

Η ανάπτυξη της εφαρμογής δεν έγινε χωρίς δυσκολίες, διότι όταν δημιουργείται ένας Headless ιστότοπος θα πρέπει όλα να γίνουν από την αρχή, ξεκινώντας από την σχεδίαση μέχρι και την ασφάλεια της ιστοσελίδας. Η επίλυσή τους οδήγησε σε βαθύτερη κατανόηση του πώς πρέπει να σχεδιάζονται τέτοιου είδους συστήματα ώστε να είναι και λειτουργικά και ασφαλή.

Παρότι το τελικό αποτέλεσμα είναι ικανοποιητικό και πλήρως λειτουργικό, υπάρχουν αρκετές δυνατότητες βελτίωσης και επέκτασης της εφαρμογής στο μέλλον. Μία επίλυση θα ήταν να μπορούσαν να προστεθούν λειτουργίες όπως wishlist για τους χρήστες, αξιολογήσεις προϊόντων και ειδοποιήσεις μέσω email

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] CERN, “A short history of the Web,” [Online]. Available: https://home.web.cern.ch/science/computing/birth-web/short-history-web

[2] Simplilearn, “Web 1.0, 2.0, 3.0, & 4.0: A Detailed Guide,” [Online]. Available: https://www.simplilearn.com/what-is-web-1-0-web-2-0-and-web-3-0-with-their-difference-article

[3] W3Schools, “JavaScript History,” [Online]. Available: https://www.w3schools.com/js/js_history.asp

[4] Wikipedia, “NEXT,” [Online]. Available: https://en.wikipedia.org/wiki/NEXT

[5] ButterCMS, “Pros and Cons of Implementing a Headless CMS,” [Online]. Available: https://buttercms.com/blog/headless-cms-advantages/

[6] Vercel, “Deploying to Vercel,” [Online]. Available: https://vercel.com/docs/deployments

[7] Auth.js, “Installing Auth.js,” [Online]. Available: https://authjs.dev/getting-started/installation?framework=NEXT

[8] JWT Authentication for WP REST API, “Installation JWT Authentication,” [Online]. Available: https://wordpress.org/plugins/jwt-authentication-for-wp-rest-api

[9] Hetzner, “Creating a server,” [Online]. Available: https://docs.hetzner.com/cloud/servers/getting-started/creating-a-server

[10] Next.js, “Installation,” [Online]. Available: https://nextjs.org/docs/14/getting-started/installation

[11] Woocommerce, “Rest Api Introduction,” [Online]. Available: https://woocommerce.github.io/woocommerce-rest-api-docs/#introduction

[12] Apollo, “Apollo Docs” [Online]. Available: https://www.apollographql.com/docs/

[13] Medium, “Setup Apollo Client with Next.js 14,” [Online]. Available: https://abdulrehmandev.medium.com/setup-apollo-client-with-next-js-14-42c25c38aea8

[14] Next.js , “Route Groups,” [Online]. Available: https://nextjs.org/docs/app/api-reference/file-conventions/route-groups

[15] Next.js , “Loading UI and Streaming,” [Online]. Available: https://nextjs.org/docs/14/app/building-your-application/routing/loading-ui-and-streaming

[16] Shadcn/ui , “Installation Next.js,” [Online]. Available: https://ui.shadcn.com/docs/installation/next

[17] Next.js, “Installation Next.js,” [Online]. Available: https://nextjs.org/learn/dashboard-app/adding-search-and-pagination

[18] Zod, “Zod Documentation,” [Online]. Available: https://zod.dev

ΠΑΡΑΡΤΗΜΑ Α

TransitionLink

```
'use client';
import { useRouter } from 'next/navigation';
import Link from 'next/link';

function sleep(ms) {
  return new Promise((resolve) => setTimeout(resolve, ms));
}

export function TransitionLink({ children, href, ...props }) {
  const router = useRouter();
  const handleTransition = async (e) => {
    e.preventDefault();
    const body = document.querySelector('body');
    body?.classList.add('page-transition');
    await sleep(500);
    router.push(href);
    await sleep(500);
    body?.classList.remove('page-transition');
  };
  return (
    <Link href={href} onClick={handleTransition} {...props}>
      {children}
    </Link>
  );
}
```

LoadMore

```
"use client";

import { useEffect, useState } from "react";
import { useInView } from "react-intersection-observer";
import { Loader2 } from "lucide-react";
import { fetchMoreProducts } from "@/app/actions/productActions";
import { ProductBoxList } from "../ProductCatalog/ProductBoxList";
```

```
/**
 * @description
 * The `LoadMore` component implements infinite scrolling to fetch and render
 * additional products dynamically when the user scrolls near the bottom of the page.
 * It utilizes the Intersection Observer API via `useInView` to detect when
 * the user reaches the loading trigger and then fetches more products using `fetchMoreProducts`.
 *
 * @param {string} categoryId - The ID of the product category to fetch more products from.
 * @param {string | null} endCursor - The cursor indicating the last fetched product for pagination.
 *
 * @features
 * - Uses `useState` to manage loaded products and pagination state.
 * - Triggers fetching when the user scrolls to the `ref` element.
 * - Displays a loading spinner (`Loader2`) while fetching additional products.
 * - Appends new products to the existing list to maintain a continuous scrolling experience.
 */

export function LoadMore({ categoryId, endCursor }) {
  const { ref, inView } = useInView();
  const [data, setData] = useState([]);
  const [nextCursor, setNextCursor] = useState(endCursor);
  const [hasNextPage, setHasNextPage] = useState(true);

  useEffect(() => {
    if (inView && hasNextPage) {
      fetchMoreProducts(categoryId, nextCursor).then((res) => {
        setData([...data, ...res.products]);
        setNextCursor(res.endCursor);
        setHasNextPage(res.hasNextPage);
      });
    }
  }, [categoryId, data, hasNextPage, inView, nextCursor]);

  return (
    <>
      {data.map((product, index) => (
        <div key={product.id} className="md:basis-1/2 lg:basis-1/5">
```

```
        <div className="p-1 grid justify-center">
          <ProductBoxList product={product} index={index} />
        </div>
      </div>
    ))}
    <div
      ref={ref}
      className="flex justify-center items-center w-full col-span-full mt-10"
    >
      {hasNextPage && <Loader2 className="animate-spin w-14 h-14" />}
    </div>
  </>
 );
}
```

ScrollButton

```
"use client";

import { ArrowUpIcon } from "lucide-react";
import { Button } from "../ui/button";
import { AnimatePresence, motion } from "framer-motion";
import { useEffect, useState } from "react";

/**
 * ScrollButton Component
 *
 * This component renders a scroll-to-top button that appears when the user scrolls
 * down more than 100 pixels on the page. The button is animated using Framer Motion
 * and only visible on screen sizes `sm` and above.
 *
 * Features:
 * - Smooth scroll to top when clicked
 * - Hidden on small screens (mobile)
 * - Animated appearance/disappearance with Framer Motion
 */

export const goToTop = () => {
```

```
  document.documentElement.scrollTo({
    top: 0,
    behavior: "smooth",
  });
};

export default function ScrollButton() {
  const [scrollPosition, setScrollPosition] = useState(0);

  useEffect(() => {
    const updatePosition = () => {
      setScrollPosition(window.pageYOffset);
    };

    window.addEventListener("scroll", updatePosition);

    return () => window.removeEventListener("scroll", updatePosition);
  }, []);

  return (
    <AnimatePresence>
      {scrollPosition > 100 && (
        <motion.div
          onClick={goToTop}
          className="fixed bottom-10 right-10 z-50"
          initial={{ y: 100, opacity: 0 }}
          animate={{ y: 0, opacity: 1, transition: { duration: 0.6 } }}
          exit={{ y: 100, opacity: 0, transition: { duration: 0.6 } }}
          whileHover={{ scale: 1.2 }}
          whileTap={{ scale: 1 }}
        >
          <Button
            variant="outline"
            size="icon"
            className="hidden sm:flex justify-center items-center"
          >
            <ArrowUpIcon className="h-6 w-6" />
            <span className="sr-only">Scroll to top</span>
```

```
        </Button>
      </motion.div>
    )}
  </AnimatePresence>
 );
}
```

DialogOrder

```
import { Button } from "@/components/ui/button";
import {
 Dialog,
 DialogContent,
 DialogFooter,
 DialogHeader,
 DialogTitle,
 DialogTrigger,
 DialogClose,
} from "@/components/ui/dialog";
import { Badge } from "@/components/ui/badge";
import Image from "next/image";
import { getOrderStatusInfo } from "@/utils/orderStatus";

export function DialogOrder({ order }) {
 return (
  <Dialog>
   <DialogTrigger asChild>
    <Button variant="outline">Λεπτομέρειες</Button>
   </DialogTrigger>
   <DialogContent className="sm:max-w-[600px] p-6 rounded-lg shadow-lg max-h-[80vh] overflow-y-auto">
    <DialogHeader>
     <DialogTitle className="text-lg font-bold text-gray-800">
      Παραγγελία #{order?.orderNumber}
     </DialogTitle>
    </DialogHeader>
    <div className="space-y-4">
     <div className="flex justify-between items-center border-b pb-2">
      <span className="text-gray-600">Ημερομηνία:</span>
```

```
  <span className="font-semibold">
    {new Date(order?.date).toLocaleDateString("el-GR")}
  </span>
 </div>
 <div className="flex justify-between items-center border-b pb-2">
  <span className="text-gray-600">Κατάσταση:</span>
  <Badge variant="outline" className="px-2 py-1">
    {getOrderStatusInfo(order?.status).label}
  </Badge>
 </div>
 <div className="border-b pb-2">
  <span className="text-gray-600 font-semibold block mb-2">
    Προϊόντα:
  </span>
  <div className="space-y-3 max-h-[300px] overflow-y-auto">
    {order?.lineItems?.nodes?.map((item, index) => (
      <div key={index} className="flex items-center gap-4">
        <Image
          src={item.product.node.image?.sourceUrl}
          alt={item.product.node.name}
          className="w-16 h-16 rounded-md object-cover"
          width={64}
          height={64}
        />
        <div>
          <p className="font-semibold">{item?.product?.node?.name}</p>
          <p className="text-gray-600">Ποσότητα: x{item?.quantity}</p>
        </div>
      </div>
    ))}
  </div>
 </div>
 {order?.shipping && (
  <div className="border-b pb-2">
    <span className="text-gray-600 font-semibold block mb-2">
      Διεύθυνση Αποστολής:
    </span>
    <p>
```

```
            {order?.shipping?.address1}, {order?.shipping?.city}
          </p>
          <p>
            {order?.shipping?.state}, {order?.shipping?.postcode}
          </p>
        </div>
      )}
      <div className="flex justify-between items-center text-lg font-bold">
        <span>Σύνολο:</span>
        <span>€{parseFloat(order?.total).toFixed(2)}</span>
      </div>
    </div>
    <DialogFooter>
      <DialogClose asChild>
        <Button variant="outline">Κλείσιμο</Button>
      </DialogClose>
    </DialogFooter>
  </DialogContent>
 </Dialog>
);
}
```

PriceRange

```
"use client";

import { useEffect, useState } from "react";

import * as SliderPrimitive from "@radix-ui/react-slider";

import { cn } from "@/lib/utils";

import { useSearchParams, useRouter, usePathname } from "next/navigation";


export function PriceRange({ maxPrice, minPrice, resetPriceRange }) {

 const searchParams = useSearchParams();

 const { replace } = useRouter();
```

```
const pathname = usePathname();

const min = parseFloat(searchParams.get("min_price")) || minPrice;

const max = parseFloat(searchParams.get("max_price")) || maxPrice;

const [priceRange, setPriceRange] = useState([min, max]);

const [internalMax] = useState(max);

const [internalMin] = useState(min);

useEffect(() => {

  if (resetPriceRange) {

    setPriceRange([min, max]); // Reset min and max on reset

  }

}, [resetPriceRange, max, min]);

const handlePriceChange = (values) => {

  setPriceRange(values);

};

// Function to round price to nearest multiple of 5

const roundToNearestFive = (price) => {

  return Math.round(price / 5) * 5;

};

// Format price with currency symbol

const formatPrice = (price) => {

  const roundedPrice = roundToNearestFive(price);

  return `${roundedPrice}€`;

};

/** Send the updated price range to the URL when the user releases the mouse (onPointerUp) */

const handlePointerUp = () => {

  const params = new URLSearchParams(searchParams);

  params.set("min_price", priceRange[0]);

  params.set("max_price", priceRange[1]);
```

```
  replace(`${pathname}?${params.toString()}`);
 };
 return (
  <div className="space-y-6">
   <div className="pt-6 pb-2">
    {/* Custom implementation of range slider with two thumbs */}
    <SliderPrimitive.Root
     className="relative flex w-full touch-none select-none items-center"
     defaultValue={priceRange}
     value={priceRange}
     onValueChange={handlePriceChange}
     min={roundToNearestFive(internalMin)}
     max={roundToNearestFive(internalMax)} // Use internal max value to avoid resetting
     step={5}
     aria-label="Price Range"
     onPointerUp={handlePointerUp}
    >
     <SliderPrimitive.Track className="relative h-2 w-full grow overflow-hidden rounded-full bg-secondary">
      <SliderPrimitive.Range className="absolute h-full bg-primary" />
     </SliderPrimitive.Track>
     {/* First thumb (min price) */}
     <SliderPrimitive.Thumb
      className={cn(
       "block h-5 w-5 rounded-full border-2 border-primary bg-background ring-offset-background transition-colors focus-visible:outline-none focus-visible:ring-2 focus-visible:ring-ring focus-visible:ring-offset-2 disabled:pointer-events-none disabled:opacity-50"
      )}
     />
     {/* Second thumb (max price) */}
```

```
        <SliderPrimitive.Thumb

          className={cn(

            "block h-5 w-5 rounded-full border-2 border-primary bg-background ring-offset-background transition-colors focus-visible:outline-none focus-visible:ring-2 focus-visible:ring-ring focus-visible:ring-offset-2 disabled:pointer-events-none disabled:opacity-50"

          )}

        />

      </SliderPrimitive.Root>

    </div>

    <div className="flex justify-between items-center">

      <div className="text-sm">

        <div className="font-medium">{formatPrice(priceRange[0])}</div>

      </div>

      <div className="text-sm text-right">

        <div className="font-medium">{formatPrice(priceRange[1])}</div>

      </div>

    </div>

  </div>

 );

}
```