ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Ανάπτυξη και Υλοποίηση ERP Συστήματος Μηχανογράφησης Αποθήκης

Του φοιτητή
Χατζημελετίου Δημήτριος
Αρ. Μητρώου: 164776

Επιβλέπων
Μπράτσας Χαράλαμπος
Καθηγητής

Ημερομηνία 29/05/2025

Τίτλος Δ.Ε. ERP σύστημα για τη μηχανογράφηση της αποθήκης
Κωδικός Δ.Ε. 25137
Ονοματεπώνυμο φοιτητή/τών Χατζημελετίου Δημήτριος
Ονοματεπώνυμο εισηγητή Μπράτσας Χαράλαμπος
Ημερομηνία ανάληψης Δ.Ε. 05/03/2025
Ημερομηνία περάτωσης Δ.Ε. 29/05/2025

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 διπλωματική εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

Η έγκριση της διπλωματικής 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.

«Σε όσους με ενθάρρυναν, με ώθησαν όταν έπρεπε, και στάθηκαν δίπλα μου σε αυτή την προσπάθεια»

Πρόλογος

Η ιδέα για την παρούσα πτυχιακή εργασία προέκυψε μέσα από προσωπική εμπειρία, καθώς εργαζόμουν ως αποθηκάριος στην οικογενειακή μας επιχείρηση, η οποία δραστηριοποιείται στον τομέα της χονδρικής πώλησης. Στην καθημερινότητά μας, η διαχείριση της αποθήκης βασιζόταν σε χειρόγραφες σημειώσεις ή υποτυπώδη καταγραφή σε αρχεία Excel. Αυτή η προσέγγιση αποδείχθηκε συχνά μη πρακτική και επιρρεπής σε λάθη, ειδικά κατά την ετοιμασία παραγγελιών ή την καταμέτρηση αποθεμάτων. Η δυσκολία εύρεσης προϊόντων, οι καθυστερήσεις στη συλλογή παραγγελιών και η απουσία αυτοματοποιημένης παρακολούθησης με ώθησαν να αναζητήσω μια πιο λειτουργική και αποδοτική λύση, προσαρμοσμένη στις πραγματικές ανάγκες μιας αποθήκης. Έτσι, σχεδίασα και υλοποίησα μια πρώτη έκδοση (proof of concept) ενός ERP συστήματος μηχανογράφησης αποθήκης, με στόχο τον εκσυγχρονισμό της διαδικασίας και την υποστήριξη των μελών της επιχείρησης στην καθημερινή εργασία τους. Μέσα από την ανάπτυξη αυτής της εφαρμογής, είχα την ευκαιρία να εφαρμόσω στην πράξη τις γνώσεις που απέκτησα κατά τη διάρκεια των σπουδών μου, να εμβαθύνω σε σύγχρονες τεχνολογίες και να συνδυάσω τον ρόλο του φοιτητή με αυτόν του "χρήστη-τελικού αποδέκτη". Η εμπειρία ήταν ιδιαίτερα δημιουργική και ουσιαστική, με την προοπτική στο μέλλον το σύστημα να εξελιχθεί περαιτέρω ώστε να καλύπτει πλήρως τις επιχειρησιακές ανάγκες μιας πραγματικής αποθήκης.

Περίληψη

Η παρούσα πτυχιακή εργασία αφορά την ανάπτυξη ενός πληροφοριακού συστήματος διαχείρισης αποθήκης (Warehouse Management System - WMS), το οποίο υλοποιήθηκε ως full-stack web εφαρμογή με χρήση Node.js, Prisma ORM και SQLite στη μεριά του εξυπηρετητή, και vanilla JavaScript στο frontend, ενσωματώνοντας barcode scanning μέσω Quagga.js.

Το σύστημα επιτρέπει την πλήρη παρακολούθηση της αποθήκης, από την καταγραφή παραλαβών και την τοποθέτηση προϊόντων σε ράφια, μέχρι την αναζήτηση, τη διαχείριση παραγγελιών και τη διενέργεια απογραφών. Ο σχεδιασμός του βασίστηκε σε αρχές ευχρηστίας και επιχειρησιακής λογικής, ώστε να συμβάλει ουσιαστικά στη μείωση του χρόνου αναζήτησης προϊόντων και στην ακρίβεια των αποθεμάτων.Το σύστημα δοκιμάστηκε σε προσομοιωμένο περιβάλλον, με θετικά αποτελέσματα σε επίπεδο λειτουργικότητας, επεκτασιμότητας και απλότητας χρήσης. Η εργασία τεκμηριώνει αναλυτικά την αρχιτεκτονική, τα endpoints, τη βάση δεδομένων και τις τεχνολογίες που χρησιμοποιήθηκαν, ενώ προτείνονται και επεκτάσεις για μελλοντική χρήση.

Design and Implementation of a Warehouse ERP System(WMS)

Chatzimeletiou Dimitrios

Abstract

This thesis presents the design and development of a Warehouse Management System (WMS) implemented as a full-stack web application. The backend is built with Node.js, Prisma ORM, and SQLite, while the frontend is developed in vanilla JavaScript, integrating barcode scanning capabilities using Quagga.js.

The system supports inventory registration, product placement by shelf-row-level structure, search functionality, store restocking orders, active order picking, and inventory counting. It emphasizes usability and operational efficiency, reducing item search time and enhancing stock management accuracy.

It was tested in a simulated warehouse scenario and showed satisfactory performance in handling typical workflows of a small-to-medium business warehouse. The thesis includes detailed documentation of the backend architecture, frontend design, REST API endpoints, and suggestions for future improvements.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω όλους όσοι συνέβαλαν, με τον τρόπο τους, στην ολοκλήρωση αυτής της πτυχιακής εργασίας. Πρώτα απ' όλα, ευχαριστώ τον επιβλέποντα καθηγητή μου, Κ. Χαράλαμπο Μπράτσα καθώς και τους φίλους και συμφοιτητές μου που στάθηκαν δίπλα μου κατά την υλοποίηση αυτής της εργασίας.

Ευχαριστώ θερμά την οικογένειά μου, που με υποστήριξε ψυχολογικά και πρακτικά σε όλη τη διάρκεια των σπουδών μου, αλλά και μέσα από την ίδια την καθημερινότητα της οικογενειακής μας επιχείρησης, μου έδωσε το έναυσμα για το θέμα αυτής της εργασίας. Τέλος, ένα ξεχωριστό ευχαριστώ σε όσους με ενθάρρυναν να συνδυάσω τις τεχνικές μου γνώσεις με την εμπειρία της πραγματικής ζωής, μετατρέποντας μια καθημερινή ανάγκη σε πηγή δημιουργίας.

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Συντομογραφίες

Δ.Ε.	Διπλωματική Εργασία
ΔΙΠΑΕ	Διεθνές Πανεπιστήμιο Ελλάδος
Π.Ε.	Πτυχιακή Εργασία
ERP	Enterprise Resource Planning
WMS	Warehouse Management System
API	Application Programming Interface
CRUD	Create, Read, Update, Delete
ORM	Object-Relational Mapping
M2M	Many-to-Many (πολλοί προς πολλούς)
1:M	One-to-Many (ένας προς πολλούς)
M:1	Many-to-One (πολλοί προς έναν)
UI	User Interface (Διεπαφή Χρήστη)
UX	User Experience (Εμπειρία Χρήστη)
HTTP	HyperText Transfer Protocol
SQL	Structured Query Language
JSON	JavaScript Object Notation
DOM	Document Object Model

Κεφάλαιο 1ο: Εισαγωγή

1.1 Πλαίσιο και σκοπιμότητα του έργου

Η αποθήκη αποτελεί κρίσιμο λειτουργικό πυρήνα για κάθε επιχείρηση που ασχολείται με εμπόριο, διανομή ή χονδρική πώληση. Η ακριβής παρακολούθηση αποθεμάτων, η άμεση εξυπηρέτηση παραγγελιών και η οργάνωση του χώρου αποτελούν βασικούς παράγοντες για την ομαλή λειτουργία και την αποδοτικότητα του συνόλου της επιχείρησης.

Η παρούσα εργασία εστιάζει στον σχεδιασμό και την υλοποίηση ενός **ERP συστήματος μηχανογράφησης αποθήκης**, που σκοπό έχει να οργανώσει, να απλοποιήσει και να ψηφιοποιήσει τις διαδικασίες παραλαβής, αποθήκευσης, αναζήτησης, συλλογής παραγγελιών και απογραφής προϊόντων. Η εφαρμογή προέκυψε από **προσωπική εμπειρία**, καθώς, εργαζόμενος στην αποθήκη της οικογενειακής μας επιχείρησης, εντόπισα τις δυσκολίες που προκύπτουν από τη χρήση χειρόγραφων ή απλοϊκών μέσων καταγραφής (όπως Excel), και τις επιπτώσεις τους στην καθημερινή λειτουργία: καθυστερήσεις, λάθη, διπλοεγγραφές ή παραλείψεις.

Μέσα από αυτήν την εργασία, υλοποιείται ένα **proof of concept σύστημα** με στόχο να λειτουργήσει ως βάση για μελλοντική επέκταση και προσαρμογή στις πραγματικές συνθήκες και ανάγκες μιας αποθήκης.

1.2 Σκοπός, Στόχοι και Παραδοτέα της Έρευνας

Η εργασία αυτή στοχεύει στην ανάπτυξη ενός πληροφοριακού συστήματος διαχείρισης αποθήκης, βασισμένου σε αρχιτεκτονική full-stack web εφαρμογής, με στόχο την υποστήριξη και αυτοματοποίηση βασικών διαδικασιών αποθήκης όπως η καταγραφή προϊόντων, η τοποθέτησή τους στον χώρο, η διαχείριση παραγγελιών και η απογραφή.

Η προσέγγιση του έργου βασίζεται σε σύγχρονες τεχνολογίες web και περιλαμβάνει την υλοποίηση backend με RESTful API και Prisma ORM, frontend με JavaScript και barcode scanning, και βάση δεδομένων SQLite. Το σύστημα επιδιώκει να προσφέρει λειτουργικότητα, ευχρηστία και δυνατότητα επεκτασιμότητας.

Στόχοι:

- **Ανάπτυξη ενός web-based ERP συστήματος για αποθήκες**
  Στόχος είναι η δημιουργία ενός πλήρως λειτουργικού συστήματος που να υποστηρίζει τις βασικές ανάγκες μιας αποθήκης: εισαγωγή προϊόντων, διαχείριση παραγγελιών, αναζήτηση, απογραφή. Το σύστημα σχεδιάζεται ως web εφαρμογή ώστε να είναι προσβάσιμο από οποιαδήποτε συσκευή χωρίς εγκατάσταση λογισμικού.

- **Υλοποίηση RESTful API με Node.js και Express**
  Η διασύνδεση frontend και backend υλοποιείται μέσω API endpoints που ακολουθούν τη REST αρχιτεκτονική. Αυτό εξασφαλίζει διαχωρισμό της επιχειρησιακής λογικής από την παρουσίαση και ευκολία μελλοντικής επέκτασης, π.χ. με mobile εφαρμογή.

- **Δημιουργία βάσης δεδομένων με Prisma ORM και SQLite**
  Η χρήση του Prisma επιτρέπει μοντελοποίηση των οντοτήτων της αποθήκης (π.χ. προϊόν, τοποθεσία, παραγγελία) με ασφάλεια, σαφήνεια και δυνατότητα μελλοντικής μεταφοράς σε

άλλο DBMS. Επιλέχθηκε η SQLite για ευκολία στο deployment και στην ανάπτυξη (development simplicity).

- **Καταγραφή προϊόντων με χρήση barcode scanning**
  Μέσω της βιβλιοθήκης Quagga.js ενσωματώνεται λειτουργία σάρωσης barcodes στην εφαρμογή, ώστε οι χρήστες να μπορούν να καταχωρούν και να εντοπίζουν προϊόντα χωρίς να πληκτρολογούν χειροκίνητα τους κωδικούς. Η τεχνολογία αυτή μειώνει τα λάθη και βελτιώνει την ταχύτητα.

- **Διαχείριση αποθηκευτικού χώρου με ταξινόμηση προϊόντων**
  Τα προϊόντα τοποθετούνται σε ράφια βάσει της λογικής: **διάδρομος – ράφι – επίπεδο**. Η λειτουργία αυτή βοηθά στον χωρικό εντοπισμό των προϊόντων και μειώνει τον χρόνο συλλογής κατά την προετοιμασία παραγγελιών.

- **Δυνατότητα δημιουργίας και συλλογής παραγγελιών**
  Οι χρήστες μπορούν να καταχωρούν αιτήματα παραγγελίας (π.χ. από κατάστημα προς αποθήκη), να βλέπουν τις ενεργές παραγγελίες και να επιβεβαιώνουν ποια προϊόντα συλλέχθηκαν. Το σύστημα υποστηρίζει ενδιάμεσες και τελικές καταστάσεις για παρακολούθηση της ροής.

- **Ενσωμάτωση λειτουργίας απογραφής προϊόντων**
  Η λειτουργία απογραφής δίνει τη δυνατότητα στο προσωπικό να εισάγει τις πραγματικές ποσότητες προϊόντων που βρίσκονται στην αποθήκη. Αυτό επιτρέπει τον εντοπισμό ελλείψεων ή αποκλίσεων και τη διατήρηση σωστών αποθεμάτων.

- **Επεκτασιμότητα και προετοιμασία για παραγωγική χρήση**
  Αν και η παρούσα υλοποίηση αποτελεί proof of concept, έχουν ληφθεί υπόψη οι αρχές καλού σχεδιασμού ώστε να μπορεί το σύστημα να επεκταθεί μελλοντικά. Ενδεικτικά, μπορεί να προστεθεί διαχείριση ρόλων χρηστών, υποστήριξη πολλών αποθηκών, ή λειτουργία μέσω κινητού

1.3 Δομή της πτυχιακής εργασίας

Η παρούσα πτυχιακή εργασία είναι δομημένη με τέτοιο τρόπο ώστε να παρουσιάσει σταδιακά το θεωρητικό πλαίσιο, την τεχνολογική προσέγγιση και την υλοποίηση του πληροφοριακού συστήματος, τεκμηριώνοντας πλήρως τα στάδια ανάλυσης, ανάπτυξης και αξιολόγησης.

Η συγγραφή της πτυχιακής βασίστηκε σε πέντε ενότητες:

- **Κεφάλαιο 1 – Εισαγωγή:** Παρουσιάζεται το πλαίσιο και η σκοπιμότητα του έργου, οι στόχοι και τα παραδοτέα της εργασίας, καθώς και η δομή της συνολικής πτυχιακής.

- **Κεφάλαιο 2 – Θεωρητικό Υπόβαθρο:** Περιλαμβάνει έννοιες σχετικές με ERP συστήματα, αποθηκευτικά πρότυπα, APIs, barcode scanning και σύγχρονες web τεχνολογίες.

- **Κεφάλαιο 3 – Υλοποίηση του Συστήματος:** Αναλύονται τα επιμέρους μέρη του backend και frontend, το σχήμα της βάσης, η αρχιτεκτονική και η ενσωμάτωση των λειτουργιών της αποθήκης.

- **Κεφάλαιο 4 – Δοκιμές και Αξιολόγηση:** Παρουσιάζονται σενάρια χρήσης, τεχνικά αποτελέσματα, δυσκολίες και παρατηρήσεις από τη λειτουργία του συστήματος.

- **Κεφάλαιο 5 – Συμπεράσματα και Μελλοντικές Επεκτάσεις:** Ανακεφαλαιώνονται τα επιτεύγματα και προτείνονται τρόποι επέκτασης της εφαρμογής σε παραγωγικό περιβάλλον.

Βιβλιογραφία και Παραρτήματα: Περιλαμβάνουν τις πηγές που χρησιμοποιήθηκαν και υποστηρικτικό υλικό.

1.4 Επίλογος

Στο παρόν κεφάλαιο παρουσιάστηκε το γενικό πλαίσιο της πτυχιακής εργασίας, τους λόγους που με οδήγησαν στην επιλογή του θέματος, καθώς και τους στόχους και τα παραδοτέα της εργασίας. Μέσα από την προσωπική εμπειρία στον χώρο της αποθήκης, ανέδειξα την ανάγκη για μια πιο οργανωμένη και σύγχρονη λύση. Η συνέχεια της εργασίας εστιάζει στο θεωρητικό υπόβαθρο και τις τεχνολογικές βάσεις που στήριξαν την υλοποίηση της εφαρμογής.

Κεφάλαιο 2ο: Θεωρητικό Υπόβαθρο και Τεχνολογικές Επιλογές

Το παρόν κεφάλαιο παρουσιάζει το θεωρητικό υπόβαθρο που στηρίζει την ανάπτυξη της εφαρμογής, καθώς και τις τεχνολογικές επιλογές που πραγματοποιήθηκαν για την υλοποίησή της. Οι τεχνολογίες επιλέχθηκαν με βάση την απλότητα υλοποίησης, την ευκολία στην εγκατάσταση και τη δυνατότητα μελλοντικής επέκτασης.

2.1 Αρχές Διαχείρισης Αποθηκών και ERP Συστημάτων

Τα ERP (Enterprise Resource Planning) συστήματα αποτελούν βασικό εργαλείο για την ψηφιοποίηση και την αυτοματοποίηση λειτουργιών επιχειρήσεων. Στο πλαίσιο της διαχείρισης αποθήκης, ένα ERP περιλαμβάνει λειτουργίες όπως καταγραφή προϊόντων, διαχείριση αποθεμάτων, δημιουργία παραγγελιών και παρακολούθηση κινήσεων.

Η ανάγκη για αυτοματοποίηση στη διαχείριση αποθηκών προκύπτει από τις καθημερινές δυσκολίες που σχετίζονται με τη φυσική καταγραφή δεδομένων, τον εντοπισμό προϊόντων και την ακριβής απογραφή. Μέσω ενός συστήματος ERP, αυτά τα προβλήματα μπορούν να ελαχιστοποιηθούν, βελτιώνοντας την αποδοτικότητα και μειώνοντας τα σφάλματα.

2.2 Αρχιτεκτονική Web Εφαρμογών και Μοντέλο Client–Server

Η εφαρμογή υλοποιήθηκε ως **web-based πληροφοριακό σύστημα** με αρχιτεκτονική πελάτη–εξυπηρετητή (client-server). Ο **πελάτης (client)** είναι ο browser του χρήστη και επικοινωνεί με τον **εξυπηρετητή (server)** μέσω HTTP αιτήσεων. Αυτή η λογική καθιστά την εφαρμογή προσβάσιμη από πολλαπλές συσκευές χωρίς την ανάγκη εγκατάστασης τοπικού λογισμικού.

2.3 Τεχνολογίες Υλοποίησης Backend

2.3.1 Node.js

Η υλοποίηση του backend πραγματοποιήθηκε με τη χρήση του Node.js, μιας πλατφόρμας που βασίζεται στη JavaScript και προσφέρει δυνατότητα ταυτόχρονης εξυπηρέτησης πολλαπλών αιτήσεων μέσω του event-driven μοντέλου της.

2.3.2 Framework Express.js

Το Express αποτέλεσε το βασικό framework για την ανάπτυξη του REST API. Επιτρέπει την εύκολη δημιουργία routes, τη διαχείριση middleware και την οργάνωση της επιχειρησιακής λογικής σε αρθρωτή μορφή (modular).

2.3.3 Σύστημα ORM Prisma

Το **Prisma** είναι ένα σύγχρονο εργαλείο τύπου Object-Relational Mapping (ORM), το οποίο χρησιμοποιείται για την επικοινωνία με σχεσιακές βάσεις δεδομένων, μέσω καθαρής και ασφαλούς σύνταξης σε JavaScript ή TypeScript. Αντί ο προγραμματιστής

να γράφει απευθείας SQL ερωτήματα, το Prisma δημιουργεί αυτόματα έναν client που βασίζεται στο schema και επιτρέπει την εκτέλεση βασικών λειτουργιών όπως αναζήτηση, καταχώρηση και ενημέρωση δεδομένων (CRUD), με τρόπο πιο εύκολο και αξιόπιστο.

Επιλέχθηκε επειδή είναι εύχρηστο, με πολύ καλή τεκμηρίωση και μεγάλη κοινότητα υποστήριξης. Ενσωματώνεται άψογα σε εφαρμογές Node.js και συνδυάζει την απλότητα ενός ORM με τη δυνατότητα ελέγχου και ευελιξίας που χρειάζεται μια custom υλοποίηση.

2.3.4 Βάση Δεδομένων SQLite

Η βάση δεδομένων της εφαρμογής υλοποιήθηκε με το SQLite, ένα ελαφρύ και ενσωματωμένο σύστημα διαχείρισης βάσεων δεδομένων. Η επιλογή αυτή βασίστηκε στην ανάγκη για απλότητα, ευκολία στη μεταφορά και γρήγορη ρύθμιση χωρίς εξαρτήσεις από εξωτερικούς servers.Το SQLite είναι κατάλληλο για μικρές εφαρμογές ή proof-of-concept έργα, όπως αυτό, όπου η έμφαση δίνεται στην λειτουργικότητα και στην αμεσότητα της ανάπτυξης. Η βάση δεδομένων αποθηκεύεται τοπικά σε ένα αρχείο .db, διευκολύνοντας τις διαδικασίες ανάπτυξης, δοκιμών και παρουσίασης.

2.4 Τεχνολογίες Υλοποίησης Frontend

2.4.1 HTML, CSS και JavaScript

Το frontend της εφαρμογής υλοποιήθηκε με καθαρή HTML, CSS και JavaScript, χωρίς τη χρήση εξωτερικών frameworks, με στόχο την ευκολία εγκατάστασης και τη διατήρηση του έργου απλό και επεκτάσιμο. Η δομή των σελίδων βασίζεται σε index.html, ενώ το main.js διαχειρίζεται τα περισσότερα συμβάντα της εφαρμογής.Η χρήση vanilla JavaScript επέτρεψε τη χειροκίνητη διαχείριση του DOM, την εμφάνιση/απόκρυψη στοιχείων, καθώς και την υλοποίηση της λογικής του σαρωτή barcode, των modals και της επικοινωνίας με το backend API.

2.4.2 Modals και Διεπαφή Χρήστη

Η εφαρμογή περιλαμβάνει modal παραθυράκια, τα οποία ενεργοποιούνται για τη σάρωση barcode, την εμφάνιση μηνυμάτων ή την επιλογή ραφιού. Η υλοποίηση έγινε με JavaScript και CSS, χωρίς τη χρήση έτοιμης βιβλιοθήκης (π.χ. Bootstrap), δίνοντας έτσι περισσότερο έλεγχο στη συμπεριφορά της διεπαφής.Η λογική για τα modals διαχειρίζεται από το αρχείο modal.js, όπου καθορίζεται η ενεργοποίηση/απενεργοποίηση παραθύρων, καθώς και η σύνδεσή τους με συγκεκριμένες λειτουργίες (π.χ. φόρμες ή σάρωση).

2.4.3 Επικοινωνία με Backend – API Calls

Η επικοινωνία με το backend γίνεται μέσω του αρχείου api.js, το οποίο λειτουργεί ως intermediary layer ανάμεσα στη λογική της εφαρμογής και το REST API. Χρησιμοποιούνται fetch() κλήσεις με async/await για λήψη ή αποστολή δεδομένων, ανάλογα με το endpoint.

```
export async function fetchActiveOrders() {
  const response = await fetch(`${apiUrl}/orders?status=pending`, {
    method: 'GET',
    headers,
  });
  if (!response.ok) {
    throw new Error('Network response was not ok');
  }
  return response.json();
}
```

2.4.4 Χρήση της Βιβλιοθήκης Quagga.js για Barcode Scanning

Για την υποστήριξη σάρωσης barcode ενσωματώθηκε η βιβλιοθήκη Quagga.js, η οποία χρησιμοποιεί την κάμερα της συσκευής για να αναγνωρίσει γραμμωτούς κώδικες (EAN, UPC κ.λπ.) σε πραγματικό χρόνο. Η λειτουργία αυτή επιταχύνει σημαντικά τη διαδικασία καταχώρισης ή αναζήτησης προϊόντων και περιορίζει την ανάγκη για χειροκίνητη εισαγωγή κωδικών.

2.5 Αρχές Σχεδίασης RESTful API

Η επικοινωνία μεταξύ του frontend και του backend βασίστηκε στην αρχιτεκτονική REST. Κάθε λειτουργία της αποθήκης υλοποιείται μέσω ξεχωριστού endpoint (π.χ. /products, /orders, /inventory) με χρήση των κατάλληλων HTTP μεθόδων (GET, POST, PATCH, DELETE).

Η επιλογή της RESTful προσέγγισης επέτρεψε:

- την ανεξαρτησία frontend–backend,
- την ευκολία δοκιμών μέσω εργαλείων όπως Postman,
- και τη δυνατότητα μελλοντικής σύνδεσης με άλλες πλατφόρμες ή mobile εφαρμογές

2.6 Πρότυπα Ασφάλειας και Μηχανισμοί Ελέγχου Πρόσβασης

Αν και η εφαρμογή αναπτύχθηκε ως proof of concept, εφαρμόστηκε **βασικός μηχανισμός αυθεντικοποίησης** (Basic Auth) για προστασία του API και των δεδομένων. Οι χρήστες πιστοποιούνται με username και password, τα οποία συγκρίνονται πριν επιτραπεί πρόσβαση σε προστατευμένα endpoints.

2.7 Επίλογος

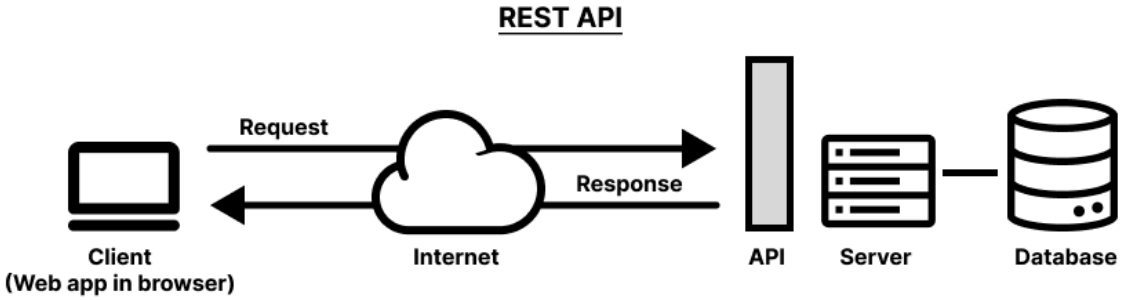
Σε αυτό το κεφάλαιο περιέγραψα τις βασικές τεχνολογίες και έννοιες που χρησιμοποίησα για να χτίσω την εφαρμογή μου. Ήταν σημαντικό για μένα να επιλέξω εργαλεία που καταλαβαίνω καλά και μπορώ να αξιοποιήσω σωστά. Η θεωρητική κατανόηση με βοήθησε να οργανώσω πιο σωστά την υλοποίηση, η οποία παρουσιάζεται στο επόμενο κεφάλαιο.

Κεφάλαιο 3ο: Σχεδιασμός και Υλοποίηση Συστήματος

3.1 Αρχιτεκτονική του Συστήματος

Η εφαρμογή που αναπτύχθηκε βασίζεται σε μια τυπική **αρχιτεκτονική client–server τριών επιπέδων** (*three-tier architecture*), η οποία διαχωρίζει καθαρά τη λογική της παρουσίασης, της επιχειρησιακής λειτουργίας και της διαχείρισης δεδομένων. Αυτή η προσέγγιση προσφέρει ευελιξία, ευκολία στην επεκτασιμότητα και καθαρό διαχωρισμό αρμοδιοτήτων μεταξύ των τμημάτων του συστήματος.

Τα τρία βασικά επίπεδα της αρχιτεκτονικής είναι τα εξής:



Σχήμα 3.1: Παράδειγμα αρχιτεκτονικής client–server τριών επιπέδων

3.1.1 Επίπεδο Παρουσίασης (Frontend)

Αφορά τη διεπαφή του χρήστη και υλοποιήθηκε με **HTML, CSS και JavaScript**. Ο χρήστης μπορεί μέσω browser να εκτελεί λειτουργίες όπως:

- Προσθήκη προϊόντων/Καταχώριση παραλαβών
- Αναζήτηση προϊόντων
- Σάρωση barcodes
- Δημιουργία παραγγελιών
- Συλλογή παραγγελιών
- Απογραφή

Η JavaScript αποτελεί το βασικό εργαλείο διαχείρισης της διεπαφής, καθώς αναλαμβάνει την αλληλεπίδραση του χρήστη με το backend μέσω fetch() αιτήσεων. Μέσα από αυτή τη λογική, η εφαρμογή μπορεί να στέλνει και να λαμβάνει δεδομένα δυναμικά χωρίς ανανέωση της σελίδας. Παράλληλα, διαχειρίζεται τη συμπεριφορά των modals (π.χ. άνοιγμα σαρωτή), την υποβολή φορμών (π.χ. καταχώριση προϊόντων ή παραγγελιών), καθώς και την εμφάνιση ειδοποιήσεων που καθοδηγούν τον χρήστη κατά τη χρήση της εφαρμογής..

3.1.2 Επίπεδο Λογικής Εφαρμογής (Backend – API Layer)

Το backend υλοποιήθηκε με **Node.js** και το framework **Express.js**. Εδώ βρίσκονται οι κύριες λειτουργίες του συστήματος:

- Υποδοχή και επεξεργασία των αιτήσεων από το frontend

- Επιχειρησιακή λογική (services)
- Αυθεντικοποίηση χρηστών
- Επικοινωνία με τη βάση δεδομένων

Το backend διαθέτει ένα RESTful API, το οποίο χρησιμοποιείται από το frontend για όλες τις βασικές λειτουργίες. Περιλαμβάνει endpoints όπως GET /products για εμφάνιση προϊόντων, POST /orders για δημιουργία παραγγελιών και PATCH /products για ενημέρωση προϊόντων. Μέσω αυτών, η εφαρμογή μπορεί να ανταλλάσσει δεδομένα εύκολα και με οργανωμένο τρόπο.

3.1.3 Επίπεδο Δεδομένων (Database Layer)

Το επίπεδο δεδομένων αποτελεί το θεμέλιο του συστήματος, καθώς εκεί αποθηκεύονται όλες οι κρίσιμες πληροφορίες που σχετίζονται με τα προϊόντα, τις τοποθεσίες αποθήκευσης, τις παραγγελίες και τις απογραφές. Η βάση δεδομένων έχει υλοποιηθεί με χρήση **SQLite**, ενώ η διαχείριση των δεδομένων πραγματοποιείται μέσω του **Prisma ORM**. Ο ρόλος του επιπέδου αυτού είναι να εξασφαλίζει τη μόνιμη και οργανωμένη αποθήκευση των δεδομένων, ώστε να μπορούν να ανακτώνται, να ενημερώνονται και να διαγράφονται με ασφάλεια μέσω των λειτουργιών του backend.

Η αναλυτική σχεδίαση της βάσης δεδομένων, καθώς και οι σχέσεις μεταξύ των οντοτήτων που την αποτελούν, παρουσιάζονται αναλυτικά στην επόμενη ενότητα (3.2).

3.2 Σχεδιασμός Βάσης Δεδομένων

Η βάση δεδομένων της εφαρμογής περιλαμβάνει αρκετούς πίνακες (οντότητες), και κάθε ένας από αυτούς χρησιμοποιείται για να καλύψει μια συγκεκριμένη λειτουργία της αποθήκης. Μέσα από τις συνδέσεις μεταξύ τους, μπορούμε να οργανώσουμε τις βασικές διαδικασίες, όπως η τοποθέτηση προϊόντων, οι παραγγελίες και η απογραφή.

Παρακάτω παρουσιάζονται όλοι οι βασικοί πίνακες της βάσης, όπως έχουν οριστεί στο αρχείο schema του Prisma.

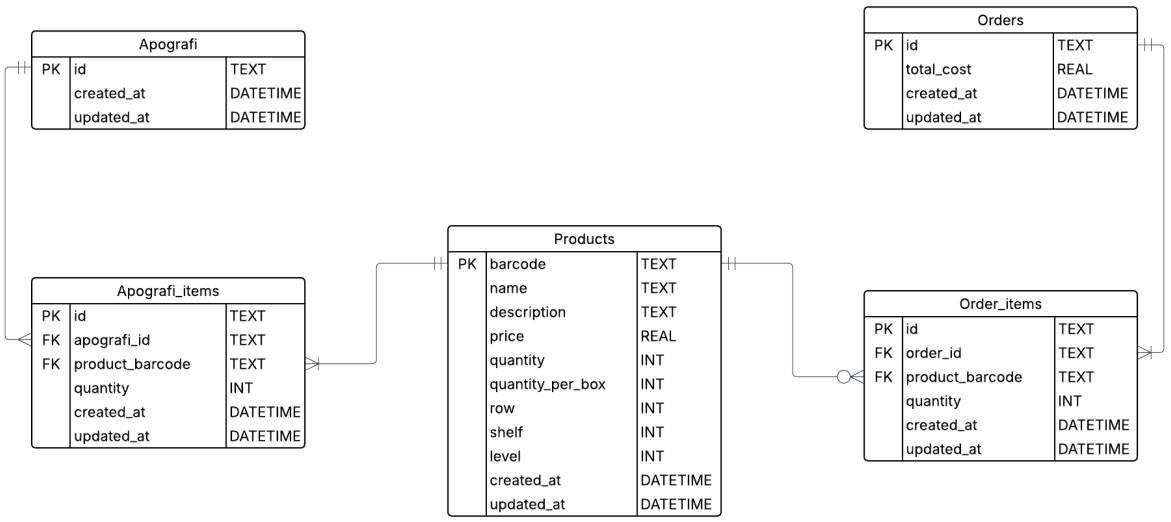
Οντότητες Βάσης Δεδομένων και Σχέσεις				
Α/Α	**Οντότητα**	**Περιγραφή**	**Πεδία**	**Σχέσεις**
1	**Product**	Αντιπροσωπεύει κάθε προϊόν που αποθηκεύεται στην αποθήκη.	barcode, name, description, price, quantity, quantity_per_box, row, shelf, level, created_at, updated_at	Συνδέεται με OrderItems one-to-many - Συνδέεται με ApografiItems (one-to-many)

2	**Order**	Αντιπροσωπεύει μια παραγγελία που δημιουργείται από χρήστη.	id, status, created_at, updated_at	Συνδέεται με OrderItems one-to-many
3	**OrderItem**	Αντιπροσωπεύει κάθε γραμμή προϊόντος μέσα σε μια παραγγελία.	id, orderId, productId, quantity	Συνδέεται με Product many-to-one - Συνδέεται με Order many-to-one
4	**Apografi**	Καταγράφει μια απογραφή προϊόντων σε δεδομένη χρονική στιγμή.	id, created_at	Συνδέεται με ApografiItems one-to-many
5	**ApografiItems**	Αντιπροσωπεύει το αποτέλεσμα καταμέτρησης ενός προϊόντος σε μια απογραφή.	id, apografiId, productId, quantity	Συνδέεται με Product many-to-one - -Συνδέεται με Apografi many-to-one

Πίνακας 3.2.1 – Οντότητες της Βάσης Δεδομένων: Περιγραφή, Πεδία και Σχέσεις

3.2.1 Ανάλυση ER-Diagram

Το τελικό διάγραμμα ER της βάσης μπορείτε να το παρατηρήσετε στο Σχέδιο/Fig. 1. Ο σχεδιασμός του διαγράμματος έγινε με την Web εφαρμογή lucidchart. Το συγκεκριμένο σχέδιο πέρασε από διάφορες φάσεις μέχρι να φτάσει σε αυτό το σημείο και τελειοποιήθηκε στην συγκεκριμένη κατάσταση. Σαν σχέδιο είναι εύκολα επεκτάσιμο και απλοϊκά διαχειρίσιμο σε περίπτωση που χρειάζονται να γίνουν αλλαγές πάνω στην βάση ή σε περίπτωση που χρειάζεται να εκτελέσουμε CRUD διεργασίες πάνω σε αυτή. Η βάση όπως είναι σχεδιασμένη εκτελεί όλες τις ανάγκες της εφαρμογής μας και αποθηκεύει όσο περισσότερη πληροφορία χρειάζεται για την σωστή χρήση της χωρίς να υπάρχουν άσκοπα πεδία.



Σχήμα 3.2.1: Διάγραμμα Οντοτήτων και Συσχετίσεων της Βάσης (ER-Diagram)

Στις ενότητες που ακολουθούν γίνεται αναλυτική παρουσίαση των βασικών πινάκων και των σχέσεων που περιλαμβάνονται στο παραπάνω διάγραμμα.

3.2.2 Δομή και Συσχετίσεις Οντοτήτων

Η παρούσα ενότητα παρουσιάζει τη δομή της βάσης δεδομένων της εφαρμογής, όπως αυτή διαμορφώθηκε μετά την ανάλυση απαιτήσεων και τον εννοιολογικό σχεδιασμό. Περιγράφονται οι βασικές οντότητες του συστήματος, οι μεταξύ τους σχέσεις, καθώς και η μεθοδολογία που ακολουθήθηκε για την ομαλή και αποδοτική αποθήκευση των δεδομένων. Το διάγραμμα οντοτήτων-συσχετίσεων (ER-Diagram) απεικονίζει γραφικά τη λογική σύνδεση των πινάκων της βάσης.

3.2.2.1 Ανάλυση Απαιτήσεων & Λογικός Σχεδιασμός

Για τον σχεδιασμό της βάσης δεδομένων χρειάστηκε πρώτα να γίνει μία αναγνώριση των απαιτήσεων της εφαρμογής. Δεδομένου ότι θέλουμε να έχουμε ένα σύστημα διαχείρισης και επίβλεψης προϊόντων για απογραφή και πωλήσεις έπρεπε αρχικά να γίνει μία πρώτη διερεύνηση. Θέλουμε να έχουμε καταγεγραμμένα τα προϊόντα σε ένα σύστημα, τις απογραφές που κάνουμε για ολόκληρα τα προϊόντα αλλά και να έχουμε μία γενική εικόνα των παραγγελιών που γίνονται από τους πελάτες. Έτσι σε πρώτο στάδιο έρχεται η ιδέα για την δημιουργία πινάκων που αντιπροσωπεύουν σε σχεσιακή μορφή τις απογραφές (πίνακας Apografi), τα προϊόντα (πίνακας Products) και τις παραγγελίες (πίνακας Orders).

3.2.2.2 Περιγραφή Πίνακα Products

Ο πίνακας products περιέχει τα απαραίτητα πεδία για την καταγραφή των προιόντων. Μέσα σε αυτά συμπεριλαμβάνονται το barcode σε μορφή κειμένουν, το όνομα και η περιγραφή των προιότνων αλλά και πεδία που έχουν να κάνουν με το κόστος και την ποσότητα που υπάρχει σε κάθε προιόν. Τέλος υπάρχουν υποστηρικτικά πεδία που αναφέρονται στην τοποθεσία της εκάστοτε αποθήκης που βρίσκεται το κάθε προιόν (row, shelf, level) αλλά και στην διαχείριση έκδοσης της κάθε γραμμής που υπάρχει σε κάθε βάση με την χρήση πεδίων created_at and updated_at.

Τα πεδία created_at και updated_at έχουν μεγάλη χρησιμότητα στις βάσεις δεδομένων διότι βοηθούν στην ανάγνωση της βάσης από το ανθρώπινο μάτι, στην διαχείριση των δεδομένων όπως και στην υποστήριξη ανάλυσης τους. Βοηθάει στον συγχρονισμό με πολλαπλά συστήματα αλλά και στην καταγραφή των αλλαγών τους και στην διαχείριση προβλημάτων (conflicts) που μπορεί να δημιουργούν κατά την διάρκεια ζωής μίας εφαρμογής.

3.2.2.3 Αρχική Θεώρηση Σχέσεων – Πρόκληση M:N

Μέσα στα συγκεκριμένους πίνακες θέλουμε να έχουμε τα απαραίτητα πεδία για να μπορέσει να γίνει ένας καλός σχεδιασμός της βάσης δεδομένων για τις ανάγκες της εφαρμογής. Δυστυχώς όμως σε αυτό το στάδιο της βάσης μπορούμε να παρατηρήσουμε ότι οι συνδέσεις που μπορούμε να σκεφτούμε μεταξύ των πεδίων είναι πολλές-προς-πολλές (many-to-many). Τα προϊόντα μπορούν να περιέχονται σε πολλές απογραφές σε περίπτωση που δεν γίνουν πωλήσεις ενώ η κάθε απογραφή περιέχει πολλά προϊόντα. Αντίστοιχα σε μία παραγγελία υπάρχουν παραπάνω από ένα προϊόν και το κάθε προϊόν μπορεί να υπάρξει σε πολλές παραγγελίες.

Οι σχέσεις πινάκων επιπέδου many-to-many (M2M) αναφέρονται σε σχέσεις όπου πολλές γραμμές ενός πίνακα πρακτικά συσχετίζονται με πολλές άλλες γραμμές ενός πίνακα. Η χρήση της συγκεκριμένης μεθοδολογίας στον σχεδιασμό βάσεων ενώ έχει τους λόγους ύπαρξής της σε διάφορες περιπτώσεις, στην συγκεκριμένη περίπτωση δημιουργεί προβλήματα κατά την διάρκεια της υλοποίησης της εφαρμογής αλλά και της διαχείρισης των δεδομένων της. Κάποια από τα αρνητικά της χρήσης M2M για την δημιουργία και τον σχεδιασμό βάσεων είναι η αυξημένη πολυπλοκότητα λόγω των επαναλήψεων και των πολλαπλών καταγραφών στην βάση. Η χρήση M2M συσχετίζει πίνακες με έμμεσο τρόπο κάτι που σαν γενικός κανόνας γίνεται προσπάθεια αποφυγής.

3.2.2.4 Κανονικοποίηση μέσω Πινάκων Γέφυρας

Το συγκεκριμένο πρόβλημα όμως έχει τρόπο αντιμετώπισης. Πιο συγκεκριμένα, προτείνεται η χρήση βοηθητικών πινάκων διακλάδωσης που σαν κύριο στόχο έχουν να εκτελέσουν τον ρόλο «Γέφυρας» μεταξύ των σχετιζόμενων πινάκων. Εξ' ορισμού οι M2M σχέσεις χρήζουν δύσκολης διαχείρισης, η χρήση πινάκων διακλάδωσης βοηθάνε να λύσουν αυτό το πρόβλημα, κανονικοποιούν την βάση αλλά και επιτρέπουν την ύπαρξη σχέσεων μεταξύ αυτών των πινάκων.

Στο παράδειγμα της εφαρμογής, δημιουργούμε δύο πίνακες, έναν που θα συνδέσει τα προϊόντα με τις απογραφές και έναν που θα συνδέσει τις παραγγελίες με τα προϊόντα. Οι πίνακες ονομάζονται "apografi_items" και "order_items" και εκτελούν το έργο της γέφυρας με τους πίνακες της απογραφής και των παραγγελιών αντίστοιχα. Ο τρόπος που αποφασίστηκαν οι σχέσεις μεταξύ τους έγινε με τις εξής καταφατικές προτάσεις.

- Μία απογραφή περιέχει πολλά αντικείμενα της απογραφής, ενώ το κάθε αντικείμενο απογραφής εμπεριέχεται σε μία και μόνο μία απογραφή. Έτσι δημιουργείται η σχέση των πινάκων "Apografi" και "Apografi_items" μεταξύ τους One-To-Many (1:M).
- Το κάθε αντικείμενο απογραφής αναφέρεται πάντα σε ένα προϊόν, ενώ το κάθε προϊόν μπορεί να υπάρξει σε παραπάνω από μία απογραφές. Έτσι αποφασίζουμε την σχέση των πινάκων "Apografi_items" και "products" Many-To-One (M:1)
- Το κάθε προϊόν μπορεί να εμπεριέχεται σε πολλές εγγραφές αντικειμένων παραγγελίας, και η κάθε εγγραφή αντικειμένων παραγγελίας μπορεί να αναφέρεται σε μόνο ένα αντικείμενο. Έτσι αποφασίζουμε την σχέση των πινάκων "products" και "order_items" One-To-Many(1:M). Σημαντικό να αναφερθεί πως υπάρχει πιθανότητα σε αυτήν την περίπτωση να μην υπάρξει καμία παραγγελία για το Χ προϊόν. Οπότε ορίζουμε την Many σχέση ως Καμία, μία ή πολλές.
- Η κάθε εγγραφή αντικειμένων παραγγελίας ανήκει πάντα σε μία και μόνο μία παραγγελία, ενώ η κάθε παραγγελία μπορεί να έχει τουλάχιστον μία ή και πολλές εγγραφές αντικειμένων παραγγελίας. Έτσι αποφασίζουμε την σχέση των πινάκων "order_items" και "orders" Many-To-One(M:1)

3.2.2.5 Κλειδιά και Πεδία Συσχέτισης

Για την υγιή λειτουργία της βάσης επιλέχθηκαν στα πεδία της "apografi_items" το κύριο κλειδί "id" σε μορφή κειμένου και τα ξένα κλειδιά "apografi_id" και "product_barcode" που αντιστοιχίζονται στα κύρια κλειδιά των πινάκων "apografi" και "Products" αντιστοίχως. Επιπλέον στο "apografi_items" έχουμε το πεδίο quantity που αναφέρεται στην ποσότητα των αντικειμένων που απογράφηκαν για το εκάστοτε αντικείμενο για την αντίστοιχη απογραφή ενώ έχουμε και τα πεδία "created_at" και "updated_at" που η ύπαρξη τους έχει δικαιολογηθεί πιο πάνω. Αντίστοιχα στον

πίνακα “order_items” έχουμε το κύριο κλειδί “id” και τα ξένα κλειδιά “order_id” & “product_barcode” που αντιστοιχίζονται με τα κύρια κλειδιά των πινάκων “products” και “orders” αντίστοιχα. Ο πίνακας ακολουθεί την ίδια λογική με τον “apografi_items” όπου εμπεριέχει τα πεδία quantity, created_at και updated_at για τους αντίστοιχους λόγους.

3.2.3 Απόσπασμα Prisma Schema

Η δομή της βάσης δεδομένων έχει υλοποιηθεί μέσω του εργαλείου Prisma ORM, το οποίο επιτρέπει την περιγραφή των μοντέλων της βάσης μέσα από ένα αρχείο .prisma. Κάθε οντότητα δηλώνεται ρητά με τα πεδία της και τις σχέσεις της με άλλες οντότητες.

Παράδειγμα 1 – Many-to-One :Το παρακάτω απόσπασμα παρουσιάζει την οντότητα Product και τη σύνδεσή της με την οντότητα Shelf η οποία δεν υπάρχει ως ξεχωριστό μοντέλο στο αρχείο schema.prisma.txt. Αντί για αυτό, η λογική της θέσης αποθήκευσης (διάδρομος, ράφι, επίπεδο) υλοποιείται απευθείας ως πεδία μέσα στον πίνακα Products(many-to-one):

```
model Products {
  barcode String @unique @id
  name String
  description String?
  quantity Int
  quantity_per_box Int @default(1)
  row Int?
  shelf Int?
  level Int?
  created_at DateTime @default(now())
  updated_at DateTime @updatedAt
  part_of_order OrderItems[]
  part_of_apografi ApografiItems[]

  @@map("products")
  @@unique([row, shelf, level], name: "unique_row_shelf_level")
}
```

Παράδειγμα 2 – One-to-Many- Orders → OrderItems : Κάθε παραγγελία (Orders) μπορεί να έχει πολλά OrderItems, και κάθε OrderItem ανήκει σε μία παραγγελία.

```
model Orders {
  id String @id @default(uuid())
  created_at DateTime @default(now())
  updated_at DateTime @updatedAt
  order_items OrderItems[]
  status String @default("pending")

  @@map("orders")
}
```

```
model OrderItems {
  id String @id @default(uuid())
  order_id String
  product_barcode String
  quantity Int
  created_at DateTime @default(now())
  updated_at DateTime @updatedAt

  order Orders @relation(fields: [order_id], references: [id])
  product Products @relation(fields: [product_barcode], references: [barcode])

  @@map("order_items")
}
```

Παράδειγμα 3 – One-to-One ή One-to-Many – ApografiItems → Products: Κάθε γραμμή απογραφής (ApografiItems) συνδέεται με ένα συγκεκριμένο προϊόν από τον πίνακα Products. Αντίστοιχα, ένα προϊόν μπορεί να εμφανίζεται σε πολλές απογραφές. Αυτό σημαίνει ότι έχουμε μια σχέση τύπου one-to-many, όπου ένα προϊόν σχετίζεται με πολλές εγγραφές απογραφής. Η σύνδεση γίνεται μέσω του πεδίου productId που λειτουργεί ως ξένο κλειδί.

```
model Apografi {
  id String @id @default(uuid())
  created_at DateTime @default(now())
  updated_at DateTime @updatedAt
  apografi_items ApografiItems[]

  @@map("apografi")
}

model ApografiItems {
  id String @id @default(uuid())
  apografi_id String
  product_barcode String
  quantity Int
  created_at DateTime @default(now())
  updated_at DateTime @updatedAt

  apografi Apografi @relation(fields: [apografi_id], references: [id])
  product Products @relation(fields: [product_barcode], references: [barcode])

  @@map("apografi_items")
}
model Products {
Παράδειγμα 1
}
```

3.2.4 Δημιουργία της Βάσης με Prisma Migrate

Για τη δημιουργία και διαχείριση της βάσης δεδομένων χρησιμοποιήθηκε το εργαλείο Prisma Migrate, το οποίο αποτελεί μέρος του Prisma ORM. Το εργαλείο αυτό επιτρέπει την αυτόματη δημιουργία και τροποποίηση της βάσης, βασισμένο στο περιεχόμενο του αρχείου schema.prisma.

Αντί να γράφονται χειροκίνητα SQL εντολές, το Prisma δημιουργεί migration αρχεία και εφαρμόζει τις αλλαγές στη βάση με μία μόνο εντολή.

Εντολή για αρχική δημιουργία της βάσης:

```
npx prisma migrate dev --name init
```

Με αυτή την εντολή:

- Δημιουργείται το αρχείο migration (SQL)
- Εφαρμόζεται το schema στην SQLite βάση (dev.db)
- Δημιουργείται ο Prisma Client για χρήση στον κώδικα backend

Παραγόμενο αρχείο .sql για οντότητα Products - orders - order_items:

```
CREATE TABLE "products" (
    "barcode" TEXT NOT NULL PRIMARY KEY,
    "name" TEXT NOT NULL,
    "description" TEXT,
    "price" REAL NOT NULL,
    "quantity" INTEGER NOT NULL,
    "quantity_per_box" INTEGER NOT NULL DEFAULT 1,
    "row" INTEGER,
    "shelf" INTEGER,
    "level" INTEGER,
    "created_at" DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    "updated_at" DATETIME NOT NULL
);


CREATE TABLE "orders" (
    "id" TEXT NOT NULL PRIMARY KEY,
    "total_cost" REAL NOT NULL,
    "created_at" DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
```

```
  "updated_at" DATETIME NOT NULL
);


CREATE TABLE "order_items" (
  "id" TEXT NOT NULL PRIMARY KEY,
  "order_id" TEXT NOT NULL,
  "product_barcode" TEXT NOT NULL,
  "quantity" INTEGER NOT NULL,
  "created_at" DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
  "updated_at" DATETIME NOT NULL,
  CONSTRAINT "order_items_order_id_fkey" FOREIGN KEY ("order_id") REFERENCES "orders"
("id") ON DELETE RESTRICT ON UPDATE CASCADE,
  CONSTRAINT "order_items_product_barcode_fkey" FOREIGN KEY ("product_barcode")
REFERENCES "products" ("barcode") ON DELETE RESTRICT ON UPDATE CASCADE
);
```

3.2.5 Παράδειγμα Query με Prisma Client

Παράδειγμα: createApografi()

Η μέθοδος createApografi είναι υπεύθυνη για τη δημιουργία μιας νέας απογραφής στην αποθήκη. Θα αναλυθεί αναλυτικά σε επόμενη ενότητα, όμως σε αυτό το σημείο μας ενδιαφέρει να δούμε πώς χρησιμοποιεί τον Prisma Client για να αλληλεπιδράσει με τη βάση.

Η κλήση περιλαμβάνει τρεις τύπους Prisma queries, τα οποία εκτελούνται μέσα σε συναλλαγή (prisma.$transaction) για ασφάλεια και ατομικότητα:

createApografi()		
Α/Α	**Εντολή**	**Περιγραφή**
1	prisma.products.findMany()	Ανάκτηση των προϊόντων που υπάρχουν στη βάση
2	prisma.products.update()	Ενημέρωση της ποσότητας κάθε προϊόντος (μία φορά ανά προϊόν)
3	prisma.apografi.create() (με nested apografi_items.create)	Δημιουργία εγγραφής απογραφής και των αντίστοιχων προϊόντων

Πίνακας 3.2.6 – createApografi: Περιγραφή, Εντολή.

prisma.products.findMany()	
Χρησιμοποιείται για να ανακτηθούν όλα τα προϊόντα από τη βάση δεδομένων που αντιστοιχούν στα barcodes που στάλθηκαν από τη φόρμα απογραφής. Εξασφαλίζει ότι όλα τα προϊόντα υπάρχουν πριν συνεχιστεί η καταχώριση	
Prisma Client	`const products = await prisma.products.findMany({ where: { barcode: { in: barcodes, }, },});`
SQL	`SELECT * FROM products WHERE barcode IN ('123456', '789012', '345678');`

Πίνακας 3.2.6.1 – prisma.products.findMany().

prisma.products.update()	
Ενημερώνει την ποσότητα αποθέματος ενός προϊόντος με βάση τα δεδομένα της απογραφής. Αν η ποσότητα είναι 0, καθαρίζει και τη θέση του προϊόντος στην αποθήκη (διάδρομος, ράφι, επίπεδο).	
Prisma Client	`await prisma.products.update({ where: { barcode: item.barcode }, data: { quantity: updatedStock, row: updatedStock === 0 ? null : product.row, shelf: updatedStock === 0 ? null : product.shelf, level: updatedStock === 0 ? null : product.level, }, })`
SQL	`UPDATE products SET quantity = 8, row = 2, // or NULL shelf = 1, // or NULL level = 3 // or NULL WHERE barcode = '123456';`

Πίνακας 3.2.6.2 – prisma.products.update().

prisma.apografi.create()	
Δημιουργεί μία νέα εγγραφή απογραφής (apografi) και ταυτόχρονα καταχωρεί όλες τις γραμμές προϊόντων (apografi_items) που σχετίζονται με αυτή. Κάθε προϊόν συνδέεται με βάση το barcode του.	

Prisma Client	`const apografi = await prisma.apografi.create({`
	`  data: {`
	`  apografi_items: {`
	`  create: data.items.map((item) => ({`
	`  product: {`
	`  connect: {`
	`  barcode: item.barcode,`
	`  },`
	`  },`
	`  quantity: item.quantity,`
	`  })), },}`

SQL	`INSERT INTO apografi (created_at)` `VALUES (CURRENT_TIMESTAMP);`
	`INSERT INTO apografi_items (apografi_id, product_id, quantity)` `VALUES (1, 5, 10);`

Πίνακας 3.2.6.2 – prisma.apografi.create().

3.2.6 Τοποθέτηση Προϊόντων και SQL Triggers

Η τοποθέτηση των προϊόντων στην αποθήκη γίνεται με βάση τη λογική: διάδρομος – ράφι – επίπεδο, τα οποία αποθηκεύονται ως πεδία στον πίνακα Products. Το σύστημα δεν χρησιμοποιεί ξεχωριστό πίνακα για τις θέσεις (όπως π.χ. Shelf), αλλά αντιπροσωπεύει κάθε θέση με συνδυασμό των πεδίων row, shelf και level.

Παρακάτω παρουσιάζεται το απόσπασμα του Prisma schema που ορίζει τη μοναδικότητα θέσης:

```
model Products {
  ...
  row   Int?
  shelf Int?
  level Int?

  @@unique([row, shelf, level], name: "unique_row_shelf_level")
}
```

Μηχανισμός Ακεραιότητας:

Ο περιορισμός @@unique([row, shelf, level]) : διασφαλίζει ότι δεν μπορεί να καταχωρηθεί πάνω από ένα προϊόν στην ίδια θέση αποθήκευσης. Αν γίνει προσπάθεια εισαγωγής δεύτερου προϊόντος με τα ίδια στοιχεία θέσης, η βάση θα απορρίψει την καταχώριση.

Εκτός από τον περιορισμό μοναδικότητας @@unique([row, shelf, level]), έχουν υλοποιηθεί δύο SQL triggers για επιπλέον έλεγχο ακεραιότητας:

- Trigger 1: enforce_product_location (BEFORE INSERT)

```
CREATE TRIGGER enforce_product_location
BEFORE INSERT ON "products"
FOR EACH ROW
WHEN (
  (NEW."row" IS NOT NULL AND NEW."shelf" IS NOT NULL AND NEW."level" IS NOT NULL)
  OR
  (NEW."row" IS NULL AND NEW."shelf" IS NULL AND NEW."level" IS NULL)
) IS FALSE
BEGIN
  SELECT RAISE(ABORT, 'Invalid product location: all fields must be either NULL or non-NULL');
END;
```

- Trigger 2: enforce_product_location_update (BEFORE UPDATE)

```
CREATE TRIGGER enforce_product_location_update
BEFORE UPDATE ON "products"
FOR EACH ROW
WHEN (
  (NEW."row" IS NOT NULL AND NEW."shelf" IS NOT NULL AND NEW."level" IS NOT NULL)
  OR
  (NEW."row" IS NULL AND NEW."shelf" IS NULL AND NEW."level" IS NULL)
) IS FALSE
BEGIN
  SELECT RAISE(ABORT, 'Invalid product location: all fields must be either NULL or non-NULL');
END;
```

Οι παραπάνω triggers δεν δημιουργούνται αυτόματα από το Prisma. Αντίθετα, προστέθηκαν χειροκίνητα στο αρχείο migration.sql ώστε να διασφαλίζουν ότι δεν μπορεί να εισαχθεί ή να ενημερωθεί προϊόν με μερικώς συμπληρωμένη θέση (π.χ. μόνο row χωρίς shelf/level). Αν κάποια από τις παραπάνω συνθήκες παραβιαστεί, η βάση ακυρώνει τη λειτουργία και επιστρέφει σχετικό μήνυμα σφάλματος.

3.3 Ανάπτυξη Backend

3.3.1 Δομή REST API και Endpoints

Ο server εκθέτει τα endpoints μέσω Express.js, με χρήση του Prisma για πρόσβαση στη βάση δεδομένων. Τα δεδομένα διακινούνται σε μορφή JSON και εξυπηρετούνται μέσω fetch() από το frontend.

Βασικά χαρακτηριστικά REST API:

- Κάθε λειτουργία του backend αντιστοιχεί σε μία HTTP route (endpoint)
- Χρησιμοποιούνται τυπικά HTTP methods:
    - GET για ανάγνωση
    - POST για δημιουργία
    - PATCH για ενημέρωση
    - DELETE για διαγραφή (αν και δεν χρησιμοποιείται εκτενώς εδώ)
- Τα δεδομένα ανταλλάσσονται σε μορφή JSON

Πίνακας API endpoints			
Endpoint (Method & Path)	Περιγραφή	Headers	
GET /	Έλεγχος λειτουργίας API – επιστρέφει 'ok'	None Required	
GET /products	Επιστρέφει όλα τα προϊόντα με δυνατότητα φιλτραρίσματος	None Required	Prisma .findMany() inline
GET /products/:barcode	Επιστρέφει προϊόν βάσει barcode	None Required	Prisma .findUnique() inline
POST /products	Δημιουργεί νέο προϊόν	Authorization (Basic Auth)	createProduct()
PATCH /products/:barcode	Ενημερώνει στοιχεία προϊόντος	Authorization (Basic Auth)	updateProduct()
DELETE /products/:barcode	Διαγράφει προϊόν	Authorization (Basic Auth)	deleteProduct()
POST /orders	Δημιουργεί νέα παραγγελία	Authorization (Basic Auth)	createOrder()
POST /orders/:id/complete	Ολοκληρώνει	None Required	completeOrder()

	συγκεκριμένη παραγγελία		
GET /orders	Επιστρέφει όλες τις παραγγελίες με φίλτρα	None Required	Prisma .findMany() inline
GET /orders/:id	Επιστρέφει συγκεκριμένη παραγγελία	Authorization (Basic Auth)	Prisma .findUnique() inline
POST /apografes	Δημιουργεί νέα απογραφή	None Required	createApografi()
GET /apografes	Επιστρέφει λίστα απογραφών με δυνατότητα φιλτραρίσματος	None Required	Prisma .findMany() inline
GET /apografes/:id	Επιστρέφει συγκεκριμένη απογραφή	None Required	Prisma .findUnique() inline

Πίνακας 3.3.1 – Πίνακας API endpoints.

1.POST/orders

Το συγκεκριμένο endpoint χρησιμοποιείται για τη δημιουργία νέας παραγγελίας. Η παραγγελία αποτελείται από έναν ή περισσότερους κωδικούς προϊόντων (barcodes) και τις αντίστοιχες ποσότητες. Κατά την κλήση του endpoint, δημιουργείται εγγραφή στον πίνακα orders και οι αντίστοιχες εγγραφές στον πίνακα order_items, οι οποίες συνδέουν τα προϊόντα με την παραγγελία.

Request Body:

```
{
 "items": [
   {
    "barcode": "1234567890123",
    "quantity": 3
   },
   {
    "barcode": "2345678901234",
    "quantity": 2
   }
 ]
}
```

Response Body :

- **id**: Το μοναδικό αναγνωριστικό της νέας παραγγελίας
- **created_at / updated_at**: Η χρονική σήμανση δημιουργίας/ενημέρωσης
- **order_items**[]: Λίστα με τα προϊόντα που περιέχει η παραγγελία
- **product_id / order_id**: IDs από τη βάση
- **quantity**: Η ζητούμενη ποσότητα
- **product**{}: Εμφωλευμένο αντικείμενο με πληροφορίες προϊόντος (barcode, name)
- **product.barcode**: Ο μοναδικός κωδικός του προϊόντος (EAN).
- **product.name**: Η ονομασία του προϊόντος όπως έχει καταχωρηθεί.


```
{
 "id": "a1b2c3d4",
 "created_at": "2024-05-25T12:34:56.000Z",
 "updated_at": "2024-05-25T12:34:56.000Z",
 "order_items": [
  {
   "id": "item1",
   "quantity": 2,
   "product_id": "123",
   "order_id": "a1b2c3d4",
   "product": {
    "barcode": "1234567890123",
    "name": "Παντόφλα"
   }
  },
  {
   "id": "item2",
   "quantity": 1,
   "product_id": "234",
   "order_id": "a1b2c3d4",
   "product": {
    "barcode": "2345678901234",
    "name": "Πιτζάμα"
   }
  }
 ]
}
```


Συμπληρωματική περιγραφή:

Κάθε εγγραφή order_items συνδέεται με ένα προϊόν από τον πίνακα products. Τα δεδομένα του προϊόντος (όπως το name, barcode, quantity_per_box) περιλαμβάνονται στη response ως nested αντικείμενο.

Η παραγγελία φέρει status = pending μέχρι να ολοκληρωθεί από το /orders/:id/complete.

Σφάλματα / Έλεγχοι:

- Αν δεν βρεθεί κάποιο barcode στη βάση:

  400 Bad Request – Unknown barcode: 000000

- Αν το Authorization header λείπει ή είναι λάθος:

  401 Unauthorized

2.POST /products

Το endpoint POST /products χρησιμοποιείται για να προστεθεί ένα νέο προϊόν στην αποθήκη. Η προσθήκη μπορεί να γίνει είτε κατά την αρχική παραλαβή, είτε όταν εισάγεται για πρώτη φορά ένας νέος κωδικός στο σύστημα.Είναι σημαντικό το προϊόν να συνοδεύεται από πληροφορίες όπως το όνομα, η περιγραφή και η τιμή του, αλλά και από τη φυσική του θέση στην αποθήκη — δηλαδή τον διάδρομο, το ράφι και το επίπεδο στο οποίο τοποθετείται.Αυτή η πληροφορία είναι κρίσιμη, γιατί βοηθάει σημαντικά στην εύρεση του προϊόντος στο μέλλον και μειώνει τον χρόνο αναζήτησης, ειδικά σε μεγάλους χώρους με πολλά είδη.

Request Body:

- **barcode**: Μοναδικός κωδικός EAN του προϊόντος.
- **name**: Όνομα/τύπος του προϊόντος.
- **description**: Προαιρετική περιγραφή.
- **price**: Τιμή τεμαχίου.
- **quantity**: Συνολικά διαθέσιμα τεμάχια στην αποθήκη.
- **quantity_per_box**: Πόσα τεμάχια περιλαμβάνονται ανά κιβώτιο.
- **row / shelf / level**: Θέση αποθήκευσης (διάδρομος – ράφι – επίπεδο).

```
{
 "barcode": "1234567890123",
 "name": "Καρέκλα ",
 "description": "Καρεκλάκι Camping",
 "price": 15.50,
 "quantity": 30,
 "quantity_per_box": 5,
 "row": 1,
 "shelf": A,
 "level": 3
}
```

Response Body:

- **barcode**: Ο μοναδικός κωδικός του προϊόντος.
- **name**: Το όνομα του προϊόντος.
- **description**: Προαιρετική περιγραφή.

- **price**: Η τιμή μονάδας.
- **quantity**: Η αρχική διαθέσιμη ποσότητα.
- **quantity_per_box**: Ο αριθμός τεμαχίων ανά κιβώτιο.
- **row**: Ο αριθμός του διαδρόμου στον οποίο βρίσκεται το προϊόν μέσα στην αποθήκη. Συνήθως αριθμητική τιμή (π.χ. 1, 2, 3)
- **shelf**: Το γράμμα που δηλώνει τη στήλη ραφιού (π.χ. A, B, C). Συχνά χρησιμοποιείται για οριζόντια διάταξη ραφιών στον ίδιο διάδρομο.
- **level**: Το επίπεδο (ύψος) στο οποίο έχει τοποθετηθεί το προϊόν, με τιμές 1 (χαμηλό), 2 (μεσαίο), ή 3 (ψηλό).
- **created_at / updated_at**: Χρονικές σφραγίδες καταχώρισης και τελευταίας ενημέρωσης.


```
{
 "barcode": "1234567890123",
 "name": "Παντόφλα",
 "description": "Χειμερινή με επένδυση",
 "price": 12.5,
 "quantity": 20,
 "quantity_per_box": 10,
 "row": 2,
 "shelf": 3,
 "level": 1,
 "created_at": "2024-05-25T12:00:00.000Z",
 "updated_at": "2024-05-25T12:00:00.000Z"
}
```


Σφάλματα / Έλεγχοι:

- Το authorization header λείπει ή είναι λάθος:

  401 Unauthorized – Missing or invalid credentials

- Πρόβλημα με τη σύνδεση στη βάση:

  500 Internal Server Error – Database connection failed

- Δεν υπάρχουν προϊόντα στη βάση:

  200 OK – [](επιστρέφει άδεια λίστα)

3.GET /products

Ανακτά όλα τα προϊόντα που είναι καταχωρημένα στη βάση δεδομένων. Χρησιμοποιείται για την εμφάνιση της λίστας προϊόντων στο περιβάλλον διαχείρισης, καθώς και για λειτουργίες όπως αναζήτηση και τοποθέτηση.

Response Body :

Η απάντηση επιστρέφει μια λίστα από όλα τα προϊόντα που είναι αποθηκευμένα στη βάση. Κάθε στοιχείο της λίστας αντιστοιχεί σε ένα προϊόν και περιλαμβάνει τα πεδία που αναφέραμε και παραπάνω.

Σφάλματα / Έλεγχοι:

- Το authorization header λείπει ή είναι λάθος:

  401 Unauthorized – Missing or invalid credentials

- Πρόβλημα με τη σύνδεση στη βάση:

  500 Internal Server Error – Database connection failed

- Δεν υπάρχουν προϊόντα στη βάση:

  200 OK – [](επιστρέφει άδεια λίστα)

4.GET /apografes/:id

Το συγκεκριμένο endpoint χρησιμοποιείται για την προβολή μιας συγκεκριμένης απογραφής, με βάση το id της. Η απογραφή περιλαμβάνει πληροφορίες για την ημερομηνία καταγραφής και τη λίστα προϊόντων που μετρήθηκαν, μαζί με τις αντίστοιχες ποσότητες.

Η δυνατότητα αυτή είναι χρήσιμη όταν χρειάζεται να ελέγξουμε τα αποτελέσματα προηγούμενων απογραφών, να εντοπίσουμε τυχόν διαφορές ή να ελέγξουμε την ορθότητα των στοιχείων που έχουν καταγραφεί.

Response Body:

- **id**: Το μοναδικό αναγνωριστικό της απογραφής.
- **created_at**: Η ημερομηνία και ώρα δημιουργίας της απογραφής.
- **apografi_items[]**: Η λίστα προϊόντων που καταμετρήθηκαν.
- **product{}**: Πληροφορίες προϊόντος (barcode, name).
- **product.barcode**: Ο μοναδικός κωδικός του προϊόντος (EAN
- **product.name**: Η ονομασία του προϊόντος όπως έχει καταχωρηθεί.
- **quantity**: Ποσότητα που μετρήθηκε για κάθε προϊόν.


```
{
 "id": 17,
 "created_at": "2024-05-25T12:00:00.000Z",
 "apografi_items": [
  {
   "id": 1,
   "product": {
    "barcode": "1234567890123",
    "name": "Παντόφλα"
   },
   "quantity": 15
```

```
  },
  {
   "id": 2,
   "product": {
    "barcode": "2345678901234",
    "name": "Πιτζάμα"
   },
   "quantity": 7
  }
```

Σφάλματα / Έλεγχοι:

- Αν η απογραφή δεν υπάρχει:

  404 Not Found – Apografi not found

- Αν δεν σταλεί σωστό authorization header:

  401 Unauthorized

5.POST /apografes

Το endpoint POST /apografes χρησιμοποιείται για την καταγραφή μιας νέας απογραφής στην αποθήκη. Ο χρήστης (συνήθως ο αποθηκάριος) καταχωρεί τις ποσότητες που μετρήθηκαν για κάθε προϊόν. Το σύστημα δημιουργεί μία εγγραφή στον πίνακα apografi και τις επιμέρους γραμμές απογραφής (apografi_items), που συνδέονται με τα αντίστοιχα προϊόντα.

Η απογραφή πραγματοποιείται συνήθως περιοδικά ή όταν υπάρχουν αμφιβολίες για την ακρίβεια του αποθέματος.

Request Body:

- items[]: Λίστα με τα προϊόντα που απογράφηκαν.
- barcode: Ο μοναδικός κωδικός του προϊόντος
- quantity: Η ποσότητα που μετρήθηκε κατά την απογραφή

```
{
 "items": [
  {
   "barcode": "1234567890123",
   "quantity": 15
  },
  {
   "barcode": "2345678901234",
   "quantity": 7
  }
 ]
}
```

Response Body:

- **id**: Αναγνωριστικό απογραφής
- **created_at**: Ημερομηνία και ώρα δημιουργίας.
- **apografi_items[]**: Τα προϊόντα που συμμετείχαν στην απογραφή
- **product.barcode / name**: Πληροφορίες για το κάθε προϊόν.
- **quantity**: Η μετρημένη ποσότητα.

```
{
 "id": 42,
 "created_at": "2024-05-25T14:10:00.000Z",
 "apografi_items": [
   {
    "product": {
     "barcode": "1234567890123",
     "name": "Παντόφλα"
    },
    "quantity": 15
   },
   {
    "product": {
     "barcode": "2345678901234",
     "name": "Πιτζάμα"
    },
    "quantity": 7
   }
 ]
```

Συμπληρωματική περιγραφή:

Η ενέργεια εκτελείται μέσω $transaction, ώστε αν αποτύχει κάποιο υπο-βήμα (π.χ. ενημέρωση προϊόντος), να ακυρωθεί ολόκληρη η καταχώριση και να διατηρηθεί η ακεραιότητα των δεδομένων.

Σφάλματα / Έλεγχοι:

- To authorization header λείπει

  401 Unauthorized

- Κάποιο barcode δεν υπάρχει

  400 Bad Request – Unknown barcode: 000000

- quantity μη έγκυρη (π.χ. αρνητικό)

  400 Bad Request – Invalid quantity

- To request body είναι κενό ή δεν έχει items

  400 Bad Request – Missing items

6.PATCH /products/:barcode

Το endpoint PATCH /products/:barcode χρησιμοποιείται για την ενημέρωση στοιχείων ενός προϊόντος που υπάρχει ήδη στη βάση. Χρησιμοποιείται κυρίως όταν αλλάζει η ποσότητα αποθέματος, η θέση τοποθέτησης (row, shelf, level), ή και άλλα χαρακτηριστικά όπως η τιμή ή η περιγραφή.

Η συγκεκριμένη λειτουργία είναι κρίσιμη κατά την παραλαβή νέου stock, κατά τη διαδικασία απογραφής ή όταν γίνεται αλλαγή διάταξης στην αποθήκη.

Request Body:

- **quantity:** Η νέα διαθέσιμη ποσότητα.
- **row**: Ο αριθμός του διαδρόμου στον οποίο βρίσκεται το προϊόν μέσα στην αποθήκη. Συνήθως αριθμητική τιμή (π.χ. 1, 2, 3)
- **shelf**: Το γράμμα που δηλώνει τη στήλη ραφιού (π.χ. A, B, C). Συχνά χρησιμοποιείται για οριζόντια διάταξη ραφιών στον ίδιο διάδρομο.
- **level**: Το επίπεδο (ύψος) στο οποίο έχει τοποθετηθεί το προϊόν, με τιμές 1 (χαμηλό), 2 (μεσαίο), ή 3 (ψηλό).


```
{
  "barcode": "1234567890123",
  "name": "Παντόφλα",
  "quantity": 12,
  "row": 2,
  "shelf": 1,
  "level": 3,
  "updated_at": "2024-05-25T14:45:00.000Z"
}
```


Από τα Endpoints στην Επιχειρησιακή Λογική

Όλα τα endpoints του API είναι προστατευμένα με Basic Authentication, ώστε μόνο εξουσιοδοτημένοι χρήστες να μπορούν να κάνουν ενέργειες όπως η καταχώριση παραγγελίας ή απογραφής. Ο έλεγχος αυτός γίνεται αυτόματα μέσα από middleware στον Express server.

Η λειτουργία που κρύβεται πίσω από κάθε endpoint — δηλαδή το τι ακριβώς γίνεται όταν γίνεται μια αίτηση — δεν είναι γραμμένη απευθείας στα routes. Αντί για αυτό, έχει οργανωθεί στο αρχείο services.ts, ώστε ο κώδικας να είναι πιο καθαρός και ευανάγνωστος. Εκεί χρησιμοποιείται ο Prisma Client για να διαβάζει και να ενημερώνει τη βάση δεδομένων.

Αυτός ο διαχωρισμός σε "επίπεδα" (routes, services, database) κάνει τον κώδικα πιο εύκολο στη συντήρηση και ιδανικό για να επεκταθεί στο μέλλον.

3.3.2 Επιχειρησιακή Λογική (Services)

Για την καλύτερη οργάνωση και συντηρησιμότητα του κώδικα, η επιχειρησιακή λογική της εφαρμογής έχει μεταφερθεί από τα routes σε ξεχωριστό αρχείο: το services.ts. Εκεί υλοποιούνται οι βασικές λειτουργίες της αποθήκης, όπως:

Ανάλυση Μεθόδων services.ts	
Συνάρτηση	Περιγραφή
createProduct	Δημιουργεί νέο προϊόν. Ελέγχει για ύπαρξη barcode πριν την εισαγωγή.
updateProduct	Ενημερώνει τα στοιχεία προϊόντος βάσει barcode.
deleteProduct	Ενημερώνει τα στοιχεία προϊόντος βάσει barcode.
createOrder	Δημιουργεί νέα παραγγελία και τις γραμμές order_items. Εκτελείται με $transaction.
completeOrder	Ολοκληρώνει παραγγελία και αφαιρεί ποσότητες από τα αντίστοιχα προϊόντα.
createApografi	Καταγράφει απογραφή, ενημερώνει ποσότητες και δημιουργεί γραμμές apografi_items.

Σχήμα 3.3.2.1:Περιγραφή της συνάρτησης Πίνακας βασικών μεθόδων του αρχείου

Όλες οι μέθοδοι που σχετίζονται με παραγγελίες και απογραφές (3 τελευταίες) περιλαμβάνουν λογική ελέγχου αποθέματος και χρήση nested Prisma queries. Οι createOrder και createApografi εκτελούνται μέσα από prisma.$transaction() για ασφαλή εγγραφή δεδομένων.

3.3.2.1 Ανάλυση Μεθόδων services.ts

1.Συνάρτηση: createProduct()

Η συνάρτηση createProduct είναι υπεύθυνη για την προσθήκη ενός νέου προϊόντος στο σύστημα. Καλείται κατά την παραλαβή ή όταν εισάγεται νέος κωδικός στο απόθεμα για πρώτη φορά.

Η μέθοδος λαμβάνει ως είσοδο ένα αντικείμενο που περιέχει όλα τα απαραίτητα χαρακτηριστικά του προϊόντος. Επιπλέον, πριν αποθηκευτεί το προϊόν στη βάση, πραγματοποιείται έλεγχος για το αν ήδη υπάρχει προϊόν με ίδιο barcode, ώστε να αποφευχθεί διπλοεγγραφή.

Η συνάρτηση χρησιμοποιείται από το endpoint POST /products, το οποίο είναι προσβάσιμο μόνο από εξουσιοδοτημένους χρήστες.

createProduct()			
Ενέργειες στη βάση	create στον πίνακα products, με έλεγχο μοναδικότητας barcode		
Endpoint	POST /products		
Παράμετροι εισόδου Request – Body	name	string	mandatory
	barcode	string	mandatory
	description	string	optional
	price	number	mandatory
	quantity	integer	mandatory
	quantity_per_box	integer	mandatory
	row	integer	optional
	shelf	integer	optional
	level	integer	optiona
Επιστρεφόμενο Αποτέλεσμα Response – Body	Το προϊόν όπως καταχωρήθηκε στη βάση, με όλα τα πεδία και χρονικές σφραγίδες created_at, updated_at		
	barcode	string	
	name	string	
	description	string	
	price	number	
	quantity	integer	
	quantity_per_box	integer	
	row	integer	
	shelf	integer	
	level	integer	
	created_at	string (ISO date)	
	updated_at	string (ISO date)	

Σχήμα 3.3.2.1.1:Περιγραφή της συνάρτησης createProduct() Request, Response , Endpoint

2.Συνάρτηση: updateProduct()

Η updateProduct χρησιμοποιείται για την ενημέρωση των στοιχείων ενός ήδη καταχωρημένου προϊόντος. Καλείται όταν χρειάζεται να αλλάξουν δεδομένα όπως η ποσότητα, η θέση αποθήκευσης (διάδρομος, ράφι, επίπεδο), η περιγραφή ή η τιμή ενός προϊόντος.

Η συνάρτηση εντοπίζει το προϊόν βάσει του barcode και τροποποιεί τα πεδία που έχουν δοθεί στο αίτημα. Εάν η νέα ποσότητα είναι μηδενική, μπορεί να καθαρίσει και τη θέση αποθήκευσης (θέτοντάς την σε null), ώστε να αποτυπώνεται σωστά ότι το προϊόν δεν βρίσκεται σε ράφι.

Η updateProduct εξυπηρετεί το endpoint PATCH /products/:barcode και αποτελεί βασική λειτουργία για τη διαχείριση αποθέματος.

updateProduct()			
Ενέργειες στη βάση	Εκτελεί update στον πίνακα products, εντοπίζοντας το προϊόν βάσει barcode		
Endpoint	PATCH /products/:barcode		
Παράμετροι εισόδου Request – Body	Το barcode από την URL και όποια πεδία προς ενημέρωση quantity, description, row, price, row, shelf, level, quantity_per_box		
	quantity	integer	optional
	price	number	optional
	description	string	optional
	quantity_per_box	integer	optional
	row	integer	optional
	shelf	integer	optional
	level	integer	optiona
Επιστρεφόμενο Αποτέλεσμα Response – Body	Το προϊόν όπως καταχωρήθηκε στη βάση, με όλα τα πεδία και χρονικές σφραγίδες created_at, updated_at		
	barcode	string	

Σχήμα 3.3.2.1.2:Περιγραφή της συνάρτησης updateProduct() Request, Response , Endpoint

3.Συνάρτηση: deleteProduct()

Η συνάρτηση deleteProduct χρησιμοποιείται για τη διαγραφή ενός προϊόντος από το σύστημα. Καλείται όταν ένα προϊόν δεν χρειάζεται πλέον να παρακολουθείται, είτε επειδή καταργήθηκε από το απόθεμα, είτε επειδή εισήχθη λανθασμένα.

Η λειτουργία εκτελείται με βάση το barcode του προϊόντος. Αν το προϊόν βρεθεί, αφαιρείται από τον πίνακα products. Αν δεν εντοπιστεί, επιστρέφεται κατάλληλο σφάλμα.

Η deleteProduct αντιστοιχεί στο endpoint DELETE /products/:barcode.

deleteProduct()		
Ενέργειες στη βάση	Εκτελεί delete από τον πίνακα products, με βάση το barcode	
Endpoint	DELETE /products/:barcode	
Παράμετροι εισόδου Request – Body	Δεν απαιτείται σώμα (body). Το barcode παρέχεται μέσω του URL.	
Επιστρεφόμενο Αποτέλεσμα Response – Body	Επιβεβαίωση διαγραφής (συνήθως 204 No Content) ή μήνυμα σφάλματος αν δεν βρεθεί το προϊόν	
	barcode	string

Σχήμα 3.3.2.1.3:Περιγραφή της συνάρτησης deleteProduct() Request, Response , Endpoint

4.Συνάρτηση: createOrder()

Η συνάρτηση createOrder είναι υπεύθυνη για τη δημιουργία νέας παραγγελίας στο σύστημα. Λαμβάνει ως είσοδο έναν πίνακα αντικειμένων (προϊόντα με ποσότητες), ελέγχει την εγκυρότητα των barcodes, δημιουργεί μια νέα εγγραφή στον πίνακα orders και τις αντίστοιχες εγγραφές στον πίνακα order_items. Η διαδικασία εκτελείται μέσα από συναλλαγή (transaction) ώστε να διασφαλίζεται η ακεραιότητα των δεδομένων.

```
export const createOrder = async (data: CreateOrderInput) =>
  prisma.$transaction(async (prisma) => {
   Παράδειγμα Prisma Θα Γίνει ανάλυση σε επόμενο ενότητα
}
    const order = await prisma.orders.create({
      data: {
        order_items: {
          create: data.items.map((item) => ({
            product: {
              connect: {
                barcode: item.barcode,
              },
            },
            quantity: item.quantity,
          })),
```

createOrder()	
Ενέργειες στη βάση	Δημιουργία εγγραφής στον πίνακα orders και αντίστοιχες εγγραφές στον order_items
Endpoint	POST /orders
Παράμετροι εισόδου Request – Body	Πίνακας items[], με κάθε στοιχείο να περιέχει barcode προϊόντος και quantity

	items	array[]	mandatory
	barcode	string	mandatory
	quantity	string	mandatory
Επιστρεφόμενο Αποτέλεσμα Response – Body	barcode	string	

Σχήμα 3.3.2.1.4:Περιγραφή της συνάρτησης createOrder() Request, Response , Endpoint

5.Συνάρτηση: completeOrder()

Η συνάρτηση completeOrder καλείται όταν ολοκληρώνεται η διαδικασία συλλογής μιας παραγγελίας. Ο κύριος ρόλος της είναι να αλλάξει την κατάσταση της παραγγελίας σε "completed" και να αφαιρέσει απόθεμα από τα προϊόντα που περιλαμβάνει.

Η λειτουργία αυτή είναι κρίσιμη για την ορθή παρακολούθηση του αποθέματος, αφού με την ολοκλήρωση της παραγγελίας η αποθήκη πρέπει να ενημερώνει τη διαθεσιμότητα των προϊόντων. Όλα τα updates εκτελούνται μέσα από μία συναλλαγή για να διασφαλιστεί η ακεραιότητα των δεδομένων.

Η συνάρτηση εξυπηρετεί το endpoint POST /orders/:id/complete.

completeOrder()		
Ενέργειες στη βάση	update στον πίνακα orders για να αλλάξει η κατάσταση, και update σε products για να αφαιρεθούν τα τεμάχια	
Endpoint	POST /orders/:id/complete	
Παράμετροι εισόδου Request – Body	Δεν απαιτείται σώμα (body). Η ενέργεια βασίζεται στο id της παραγγελίας μέσω URL.	
Επιστρεφόμενο Αποτέλεσμα	barcode	string

Σχήμα 3.3.2.1.5:Περιγραφή της συνάρτησης completeOrder() Request, Response , Endpoint

6.Συνάρτηση: createApografi()

Η createApografi είναι η συνάρτηση που καταγράφει μία νέα απογραφή στην αποθήκη. Χρησιμοποιείται όταν ο χρήστης θέλει να καταμετρήσει τα φυσικά αποθέματα και να ενημερώσει το σύστημα με τις νέες ποσότητες.

Η λειτουργία περιλαμβάνει τη δημιουργία μιας νέας εγγραφής στον πίνακα apografi, την αποθήκευση των γραμμών απογραφής στον πίνακα apografi_items και την ενημέρωση των ποσοτήτων στο products. Όλα τα βήματα εκτελούνται μέσα από $transaction ώστε να διασφαλιστεί ότι είτε θα αποθηκευτούν όλα σωστά είτε τίποτα, σε περίπτωση σφάλματος.

Η συνάρτηση εξυπηρετεί το endpoint POST /apografes.

createApografi()			
Ενέργειες στη βάση	create στον πίνακα apografi, create στις αντίστοιχες εγγραφές στον apografi_items, και update στον πίνακα products για να καταχωρηθούν οι μετρημένες ποσότητες. Όλα εκτελούνται μέσω $transaction.		
Endpoint	POST /apografes		
Παράμετροι εισόδου Request – Body	Πίνακας items[], με κάθε στοιχείο να περιέχει barcode προϊόντος και quantity		
	items	array[]	mandatory
	barcode	string	mandatory
	quantity	string	mandatory
Επιστρεφόμενο Αποτέλεσμα Response – Body	barcode	string	

Σχήμα 3.3.2.1.6:Περιγραφή της συνάρτησης createApografi() Request, Response , Endpoint

3.3.3 Authentication και Προστασία API

Για την προστασία των λειτουργιών του backend, η εφαρμογή χρησιμοποιεί μηχανισμό Basic Authentication. Κάθε φορά που ο χρήστης επιχειρεί να προσπελάσει κάποιο endpoint, γίνεται έλεγχος διαπιστευτηρίων (όνομα χρήστη και κωδικός) μέσω ενός middleware, το οποίο εφαρμόζεται στο σύνολο των routes.

Ο έλεγχος αυτός έχει υλοποιηθεί στο αρχείο index.ts, χρησιμοποιώντας τη βιβλιοθήκη basic-auth. Τα διαπιστευτήρια συγκρίνονται με μεταβλητές περιβάλλοντος που έχουν οριστεί στο αρχείο .env.

Απόσπασμα Middleware Ελέγχου Πρόσβασης

```
const authMiddleware = (req, res, next) => {
  const user = basicAuth(req);
  const validUser = user &&
    user.name === process.env.ADMIN_NAME &&
    user.pass === process.env.PASSWORD;
  if (validUser) {
    return next();
  }
  throw new UnauthorizedError("Invalid credentials");
};
app.use(authMiddleware);
```

Το middleware αυτό εκτελείται για όλα τα endpoints που δηλώνονται μετά την εντολή app.use(authMiddleware);, διασφαλίζοντας ότι οι λειτουργίες εισαγωγής, ενημέρωσης και διαγραφής δεδομένων είναι προσβάσιμες μόνο από εξουσιοδοτημένους χρήστες.

Σε περίπτωση μη έγκυρων στοιχείων, η εφαρμογή επιστρέφει σφάλμα 401 Unauthorized μέσω της custom κλάσης UnauthorizedError.

3.3.4 Error Handling και Ασφάλεια

Η εφαρμογή περιλαμβάνει μηχανισμούς για να μπορεί να αντιμετωπίζει σφάλματα που μπορεί να προκύψουν, είτε από τον χρήστη είτε από τη λειτουργία του backend. Στόχος είναι το σύστημα να μην "σπάει", αλλά να απαντά με κατανοητά μηνύματα και κατάλληλο κωδικό κατάστασης (status code), όπως 400 ή 404.

Ειδικοί τύποι σφαλμάτων

Έχουν δημιουργηθεί πέντε βασικοί τύποι σφαλμάτων στο αρχείο error.ts , δύο από αυτά:

```
export class UnauthorizedError extends ApiError {
  constructor(message: string) {
    super(message, 401);
  }
}
export class NotFoundError extends ApiError {
  constructor(message: string) {
    super(message, 404);
  }
}
```

Αυτά τα σφάλματα χρησιμοποιούνται σε όλη την εφαρμογή. Για παράδειγμα, όταν κάποιος ζητά προϊόν που δεν υπάρχει, εμφανίζεται:

```
throw new NotFoundError("Product not found")
```

Ή όταν τα στοιχεία σύνδεσης είναι λάθος:

```
throw new UnauthorizedError("Invalid credentials");
```

Στο τέλος του index.ts υπάρχει μια συνάρτηση που "πιάνει" όλα τα σφάλματα που γίνονται στο backend και απαντά στον χρήστη με το κατάλληλο μήνυμα:

```
app.use(errorHandler);
```

Αντί η εφαρμογή να "κρασάρει" ή να επιστρέφει ακατανόητα τεχνικά σφάλματα, απαντά με ήρεμο και ξεκάθαρο τρόπο, τόσο για τον χρήστη όσο και για τον προγραμματιστή.

Η σωστή διαχείριση σφαλμάτων είναι σημαντική γιατί:

- Αποτρέπει την εμφάνιση επικίνδυνων ή ανεξέλεγκτων μηνυμάτων στον τελικό χρήστη,
- Διευκολύνει τον εντοπισμό και την επίλυση προβλημάτων,
- Προστατεύει την εφαρμογή από κατάρρευση ή απώλεια δεδομένων.

Με αυτόν τον τρόπο, το σύστημα γίνεται πιο σταθερό, ασφαλές και αξιόπιστο — έτοιμο να λειτουργήσει σε πραγματικές συνθήκες.

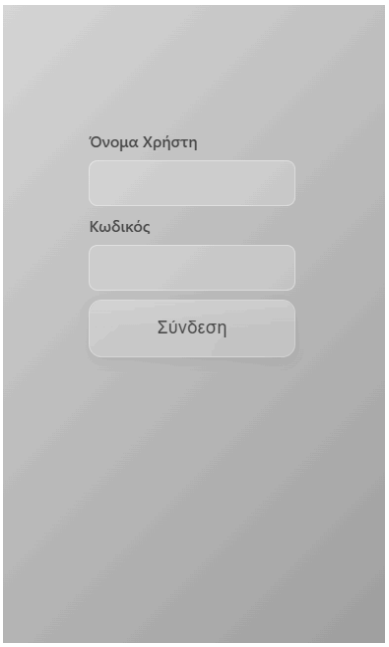
3.4 Ανάπτυξη Frontend

Αφού παρουσιάστηκε η αρχιτεκτονική και η λειτουργικότητα του backend, σε αυτή την ενότητα επικεντρωνόμαστε στο frontend της εφαρμογής. Το περιβάλλον χρήστη αποτελεί το κύριο μέσο αλληλεπίδρασης του αποθηκάριου με το σύστημα, και είναι υπεύθυνο για την εισαγωγή δεδομένων, την εκτέλεση βασικών ενεργειών και την οπτική καθοδήγηση.

3.4.1 Διεπαφή Χρήστη (UI), Login και Modals

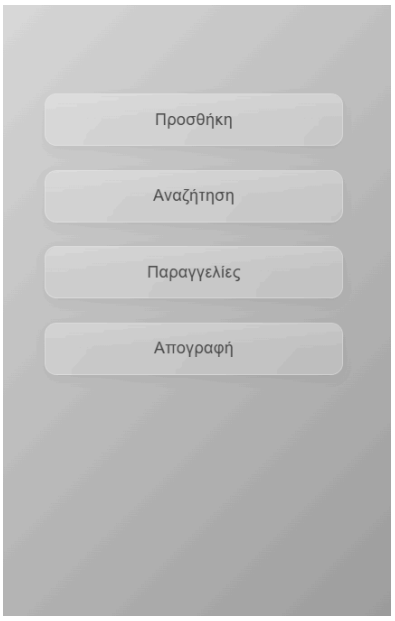
Η διεπαφή της εφαρμογής είναι σχεδιασμένη έτσι ώστε να είναι απλή και ξεκάθαρη, ειδικά για τον αποθηκάριο που τη χρησιμοποιεί καθημερινά. Από τη σύνδεση μέχρι το βασικό μενού και τα παράθυρα λειτουργιών (modals), η φιλοσοφία ήταν να κρατήσουμε τα πράγματα όσο πιο λιτά γίνεται, χωρίς περιττές πληροφορίες ή περίπλοκα μενού.

Με το που ανοίγει η εφαρμογή, ο χρήστης βλέπει τη φόρμα σύνδεσης και συμπληρώνει το username και το password. Η σύνδεση γίνεται με έναν βασικό αλλά λειτουργικό μηχανισμό ελέγχου (basic authentication), που ελέγχει τα στοιχεία και επιτρέπει την πρόσβαση μόνο αν είναι σωστά.



Σχήμα 3.4.1α: Φόρμα σύνδεσης (login screen)

Αμέσως μετά τη σύνδεση, εμφανίζεται το **κεντρικό μενού** με τέσσερις βασικές επιλογές:

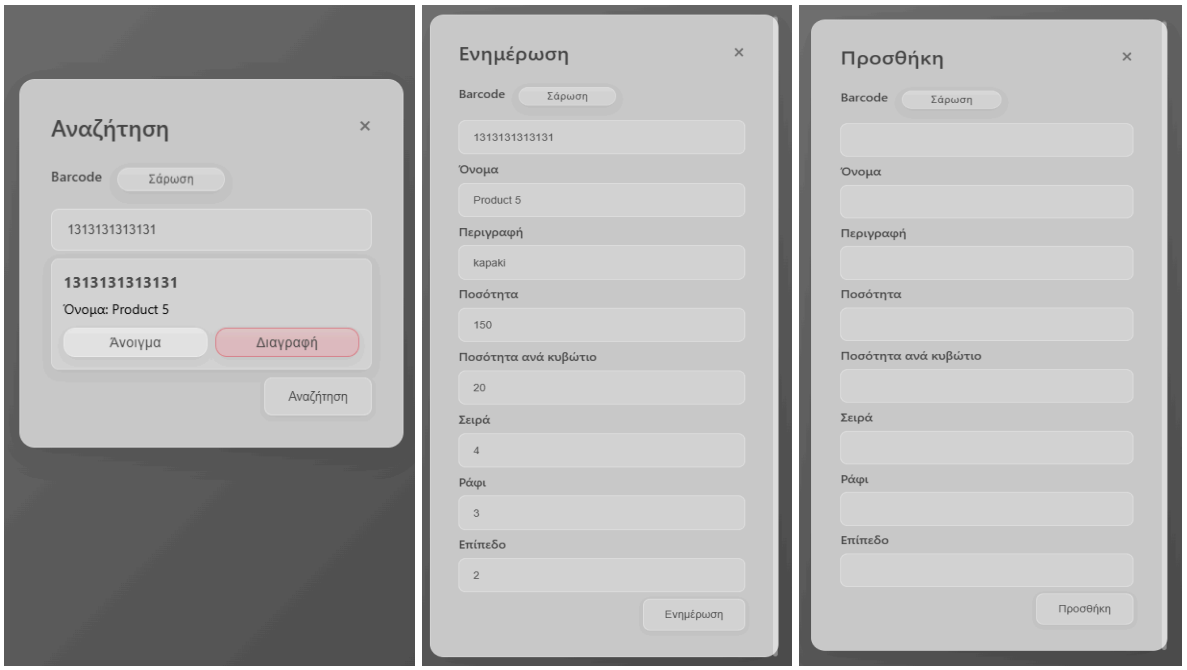


Σχήμα 3.4.1β: Κεντρικό μενού με βασικές λειτουργίες

Η αρχική οθόνη περιλαμβάνει κουμπιά για τις βασικές λειτουργίες: καταχώρηση προϊόντων, αναζήτηση, παραγγελίες και απογραφή. Κάθε ένα από τα κουμπιά ενεργοποιεί ένα modal – δηλαδή ένα εμφανιζόμενο παράθυρο – που επιτρέπει τη λειτουργία χωρίς να αλλάζει η σελίδα. Αυτό επιταχύνει τη ροή εργασιών και μειώνει τις άσκοπες πλοηγήσεις.

Για παράδειγμα, όταν πατηθεί το κουμπί **Αναζήτηση**, ανοίγει ένα modal που επιτρέπει στον χρήστη να αναζητήσει προϊόν είτε με σάρωση barcode είτε με πληκτρολόγηση του ονόματος. Μόλις βρεθεί το προϊόν, εμφανίζονται στην οθόνη οι βασικές του πληροφορίες και δίνεται η δυνατότητα για τρεις ενέργειες:

- **Άνοιγμα**: Τον μεταφέρει σε νέο modal, όπου προβάλλονται όλες οι πληροφορίες του προϊόντος (όνομα, ποσότητα, θέση, περιγραφή). Από εκεί μπορεί είτε να δει τα στοιχεία είτε να τα τροποποιήσε.
- **Ενημέρωση**: Είναι ουσιαστικά συνέχεια του "Άνοιγμα", αφού μέσα από το ίδιο modal δίνεται δυνατότητα για αλλαγές και αποθήκευση.
- **Διαγραφή**: Αν χρειάζεται, μπορεί να διαγράψει εντελώς την εγγραφή του προϊόντος.



Σχήμα 3.4.2: Modal παραθύρων αναζήτησης, ενημέρωσης και προσθήκης προϊόντος

Αντίστοιχα, με την επιλογή **Προσθήκη**, εμφανίζεται modal όπου ο χρήστης μπορεί να σκανάρει το ή να εισάγει χειροκίνητα το barcode ενός προϊόντος και να συμπληρώσει τα στοιχεία του.

Στην παραπάνω ενότητα παρουσιάστηκαν βασικά σημεία του UI, όπως η **οθόνη σύνδεσης**, το **βασικό μενού** και το **παράθυρο αναζήτησης**, **ενημέρωσης προϊόντος** και το modal της **Προσθήκης**. Στόχος ήταν να δούμε πώς είναι οργανωμένη η εφαρμογή και τι επιλογές έχει ο χρήστης μπροστά του.

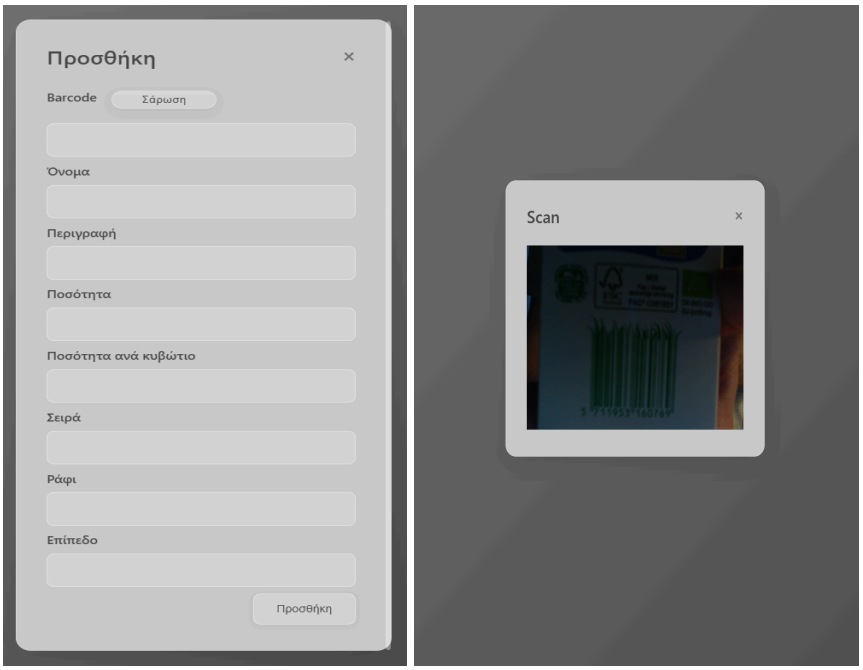
Σε επόμενη ενότητα 3.5 Ενσωμάτωση Κύριων Λειτουργιών θα δούμε πιο αναλυτικά πώς λειτουργεί κάθε βασική διαδικασία, βήμα–βήμα. Εκεί θα εξηγήσουμε πώς ο χρήστης αλληλεπιδρά με την εφαρμογή, τι συμβαίνει στο παρασκήνιο και πώς γίνονται οι κλήσεις προς το backend.

3.4.2 Σάρωση Barcode και Ενημέρωση Προϊόντων

Μία από τις βασικότερες λειτουργίες του frontend είναι η σάρωση barcode με κάμερα, ώστε ο χρήστης να μπορεί να εντοπίζει ή να καταχωρεί προϊόντα χωρίς να πληκτρολογεί τον κωδικό χειροκίνητα. Η λειτουργία αυτή είναι διαθέσιμη τόσο κατά την προσθήκη νέου προϊόντος, όσο και κατά την ενημέρωση υπάρχοντος.

Η σάρωση υλοποιείται με τη χρήση της βιβλιοθήκης **Quagga.js**, η οποία αξιοποιεί την κάμερα της συσκευής για να εντοπίσει και να αναγνωρίσει γραμμωτούς κώδικες (barcodes). Η ενσωμάτωση έχει γίνει στο αρχείο **barcode-scanner.js**, ενώ η ενεργοποίηση του scanner γίνεται μέσω κουμπιού μέσα στο modal.

Όπως είδαμε και στο παράθυρο προσθήκης προϊόντος (modal), υπάρχει η επιλογή να σαρώσουμε barcode με κάμερα αντί να γράψει ο χρήστης χειροκίνητα τον αριθμό. Αυτή η δυνατότητα προσφέρει ταχύτητα και ακρίβεια, ειδικά σε περιβάλλοντα όπως η αποθήκη, όπου τα προϊόντα διαθέτουν barcode πάνω στη συσκευασία.



Σχήμα 3.4.3: Φόρμα προσθήκης προϊόντος και ενεργοποίηση κάμερας για σάρωση barcode

Όταν ο χρήστης πατήσει "Σάρωση", ανοίγει η κάμερα της συσκευής και η εφαρμογή περιμένει να εστιάσει πάνω στο barcode. Αν η ανάγνωση είναι επιτυχής, το πεδίο του barcode στη φόρμα συμπληρώνεται αυτόματα.

Η λειτουργία σάρωσης βασίζεται στη βιβλιοθήκη Quagga.js, η οποία έχει ενσωματωθεί στο αρχείο barcode-scanner.js. Το modal περιέχει κουμπί ενεργοποίησης της σάρωσης, το οποίο καλεί τη σχετική συνάρτηση και εμφανίζει το live feed της κάμερας.

Η σάρωση barcode μπορεί να χρησιμοποιηθεί:

- Κατά την προσθήκη νέου προϊόντος, ώστε ο κωδικός να καταγραφεί με ένα scan.
- Κατά την αναζήτηση προϊόντος, για να εντοπιστεί άμεσα το είδος.
- Κατά την απογραφή, όπου η σάρωση επιταχύνει την καταμέτρηση χωρίς πληκτρολόγηση.

Η διαδικασία είναι απλή, λειτουργεί τόσο σε υπολογιστή (με κάμερα) όσο και σε κινητό, και μειώνει σημαντικά τα λάθη που θα μπορούσαν να προκύψουν από τη χειροκίνητη πληκτρολόγηση.

Παρακάτω παρουσιάζεται, σε τρία βασικά βήματα, ο τρόπος με τον οποίο έχει υλοποιηθεί η λειτουργία σάρωσης στο frontend, χρησιμοποιώντας τη βιβλιοθήκη **Quagga.js**.

Απόσπασμα κώδικα ενεργοποίησης σάρωσης barcode, αρχικοποίηση του σαρωτή:

```
Quagga.init(
```

```
{
  inputStream: {
    type: "LiveStream",
```

```
    target: document.querySelector("#scanner"),
    constraints: {
      width: 480,
      height: 320,
      facingMode: "environment",
    },},
   decoder: {
    readers: ["ean_reader"],
   }, },
  function (err) {
   if (err) {
    console.error(err);
    return;
   }
   Quagga.start();
  }
);
```

➔ Η Quagga.init() ετοιμάζει την κάμερα για live streaming και ορίζει τον τύπο barcode που θα διαβάζει (EAN). Αν όλα πάνε καλά, ξεκινάει η σάρωση με Quagga.start().

Ανίχνευση επιτυχούς barcode:

```
Quagga.onDetected(function (result) {
 const code = result.codeResult.code;
 if (code) {
  Quagga.stop();
  onDetected(code);
 }})
```

➔ Όταν εντοπιστεί έγκυρος barcode, η onDetected() καλείται. Ο σαρωτής σταματάει αυτόματα και στέλνει τον κωδικό πίσω στη φόρμα.

Κλήση της συνάρτησης από το modal:

```
startScanner((barcode) => {
 document.getElementById("barcode-input").value = barcode;
});
```

➔ Από το modal, η startScanner() καλείται και μόλις διαβαστεί ο κωδικός, γεμίζει αυτόματα το πεδίο barcode-input με την τιμή.

3.4.3 Επικοινωνία με Backend

Κάθε φορά που ο χρήστης εκτελεί μια ενέργεια μέσω της εφαρμογής (όπως προσθήκη προϊόντος, αναζήτηση ή απογραφή), γίνεται επικοινωνία με το backend μέσω HTTP αιτημάτων. Αυτό επιτυγχάνεται με χρήση της fetch() της JavaScript, μέσα από το αρχείο api.js.

Πριν σταλεί οποιοδήποτε αίτημα, προστίθεται στο header το πεδίο **Authorization** με τα στοιχεία του χρήστη, τα οποία έχουν αποθηκευτεί στο localStorage από τη φόρμα σύνδεσης.

Παράδειγμα προσθήκης προϊόντος createProduct():

```
export async function createProduct(product) {
  const response = await fetch(`${API_BASE}/products`, {
    method: "POST",
    headers: getHeaders(),
    body: JSON.stringify(product),
  });
  return response.json();
}
```

Η συνάρτηση getHeaders() προσθέτει το Authorization header με τα στοιχεία του χρήστη:

```
function getHeaders() {
  const username = localStorage.getItem("username");
  const password = localStorage.getItem("password");
  return {
    "Content-Type": "application/json",
    Authorization: "Basic " + btoa(`${username}:${password}`),
  };
}
```

Αντίστοιχα, για την ενημέρωση προϊόντος (μέσω PATCH) ή την ολοκλήρωση παραγγελίας, καλούνται οι αντίστοιχες συναρτήσεις του api.js, πάντα με την ίδια λογική.

```
export async function updateProduct(barcode, product) {
  const response = await fetch(`${API_BASE}/products/${barcode}`, {
    method: "PATCH",
    headers: getHeaders(),
    body: JSON.stringify(product),
  });
  return response.json();
}
```

Όλες οι αποκρίσεις του server χειρίζονται σε κάθε κλήση, ώστε να εμφανιστεί κατάλληλο μήνυμα στο χρήστη (επιτυχία ή αποτυχία), μέσω alerts ή console logs.

Όπως φαίνεται, όλες οι βασικές ενέργειες του χρήστη — από την προσθήκη και ενημέρωση προϊόντων μέχρι τη δημιουργία παραγγελίας — υλοποιούνται μέσα από απλές κλήσεις προς το backend, με τη χρήση της fetch() και των κατάλληλων headers. Στην επόμενη ενότητα (3.5) θα δούμε αναλυτικά πώς αυτές οι λειτουργίες εφαρμόζονται στην πράξη, μέσα από πραγματικά παραδείγματα χρήσης της εφαρμογής.

3.5 Ενσωμάτωση Κύριων Λειτουργιών

Σε αυτό το σημείο παρουσιάζεται πώς ο χρήστης (π.χ. ο αποθηκάριος) αξιοποιεί το σύστημα στην καθημερινή του εργασία. Θα δούμε τα τέσσερα βασικά σενάρια χρήσης της εφαρμογής:

- Παραλαβή προϊόντων και τοποθέτηση σε ράφι
- Αναζήτηση και διαχείριση προϊόντων
- Δημιουργία και ολοκλήρωση παραγγελίας
- Απογραφή προϊόντων

Σχήμα 3.5.1: Βασικό μενού

Κάθε λειτουργία περιγράφεται με **βήματα**, **screenshots** από το UI και (όπου χρειάζεται) **σχόλια κώδικα ή API κλήσεις**.

3.5.1 Παραλαβή και Τοποθέτηση Προϊόντων

Η παραλαβή αφορά την προσθήκη νέων προϊόντων που φτάνουν στην αποθήκη. Μέσα από το κουμπί «Προσθήκη», ο χρήστης μπορεί εύκολα να τα καταχωρήσει στο σύστημα και να δηλώσει πού ακριβώς θα τοποθετηθούν.

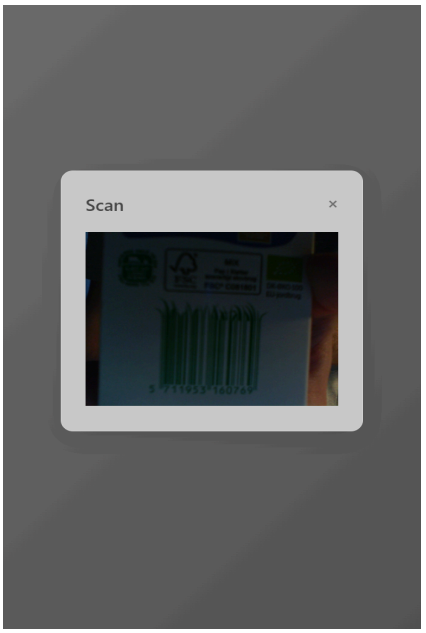
➔ **Βήμα 1 – Επιλογή "Προσθήκη"**

Αφού πατήσει το κουμπί, ανοίγει ένα παράθυρο (modal) με μια απλή φόρμα. Εκεί έχει δύο επιλογές: είτε να πληκτρολογήσει το barcode χειροκίνητα, είτε να χρησιμοποιήσει την κάμερα για να το σαρώσει αυτόματα.

Σχήμα 3.5.2: Modal προσθήκης προϊόντος

➔ **Βήμα 2 – Σάρωση ή καταχώρηση barcode**

Ο χρήστης μπορεί να επιλέξει «Σάρωση», ώστε να ενεργοποιηθεί η κάμερα της συσκευής. Η σάρωση υλοποιείται με τη βιβλιοθήκη Quagga.js και, όπως περιγράφηκε αναλυτικά στην ενότητα 3.4.2, επιτρέπει την αυτόματη αναγνώριση του barcode και τη συμπλήρωσή του στο αντίστοιχο πεδίο της φόρμας.



Σχήμα 3.5.3: Ενεργοποίηση κάμερας για σάρωση barcode

➔ **Βήμα 3 – Συμπλήρωση στοιχείων προϊόντος**

Αφού συμπληρωθεί το barcode, ο χρήστης εισάγει:

- Το όνομα του προϊόντος
- Μια περιγραφή
- Την ποσότητα κιβωτίων
- Την ποσότητα τεμαχίων ανα κιβώτιο
- Τη θέση αποθήκευσης (διάδρομος, ράφι, επίπεδο)

➔ **Βήμα 4 – Υποβολή και αποθήκευση**

Με το κουμπί «Προσθήκη»,γίνεται χρήση της κλήσης createProduct() και η εφαρμογή στέλνει τα δεδομένα στο backend με **POST** αίτημα στο endpoint **POST /products**.

Κλήση API από api.js:

```
export async function createProduct(product) {
  const response = await fetch(`${API_BASE}/products`, {
    method: "POST",
    headers: getHeaders(),
    body: JSON.stringify(product),
  });
  return response.json();
}
```

Αν όλα πάνε καλά, εμφανίζεται επιβεβαίωση. Σε περίπτωση λάθους (π.χ. διπλό barcode), εμφανίζεται σχετικό μήνυμα:

«Το barcode που εισάγατε υπάρχει ήδη.»

3.5.2 Αναζήτηση και Ενημέρωση Προϊόντων

Η δυνατότητα αναζήτησης προϊόντων είναι ιδιαίτερα χρήσιμη για την εύρεση και τη διαχείριση εγγραφών που έχουν ήδη καταχωρηθεί. Μέσα από το κουμπί «Αναζήτηση» στο κεντρικό μενού, ο χρήστης μπορεί εύκολα να βρει προϊόντα με βάση το barcode ή το όνομά τους.

➔ **Βήμα 1 – Πάτημα στο κουμπί «Αναζήτηση»**

Από την αρχική οθόνη της εφαρμογής, ο χρήστης επιλέγει «Αναζήτηση» για να ανοίξει το σχετικό modal. Εκεί έχει τη δυνατότητα είτε να πληκτρολογήσει μέρος του ονόματος, είτε να σαρώσει το barcode για πιο γρήγορη εύρεση.

Μόλις ο χρήστης εισάγει κριτήρια αναζήτησης και πατήσει «Αναζήτηση», η εφαρμογή καλεί την fetchProduct() όπου στέλνει **GET** αίτημα προς το backend στο endpoint **GET /products/:barcode**, περνώντας τα δεδομένα ως query parameters. Στην backend μεριά, καλείται η **findMany()** της Prisma, η οποία επιστρέφει μοναδικό προϊόν από τον πίνακα products με βάση το barcode.

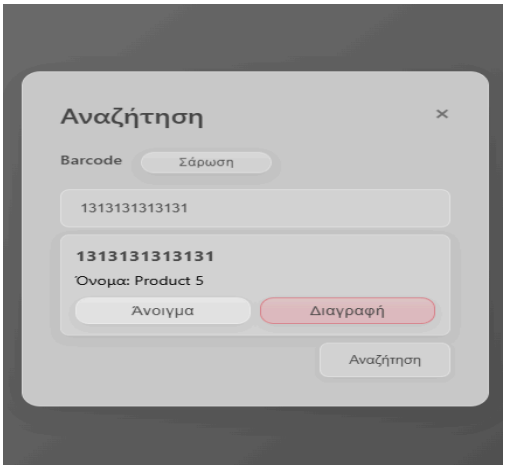
Σχήμα 3.5.4: Modal αναζήτησης προϊόντος

```
export async function fetchProduct(productId) {
  const response = await fetch(`${apiUrl}/products/${productId}`, {
    method: 'GET',
    headers,
  });
  if (!response.ok) {
    throw new Error('Network response was not ok');
  }
  return response.json();
}
```

➔ **Βήμα 2 – Προβολή αποτελέσματος και επιλογές**

Μόλις βρεθεί το προϊόν, εμφανίζεται σε λίστα με τις βασικές του πληροφορίες. Ο χρήστης έχει τρεις επιλογές:

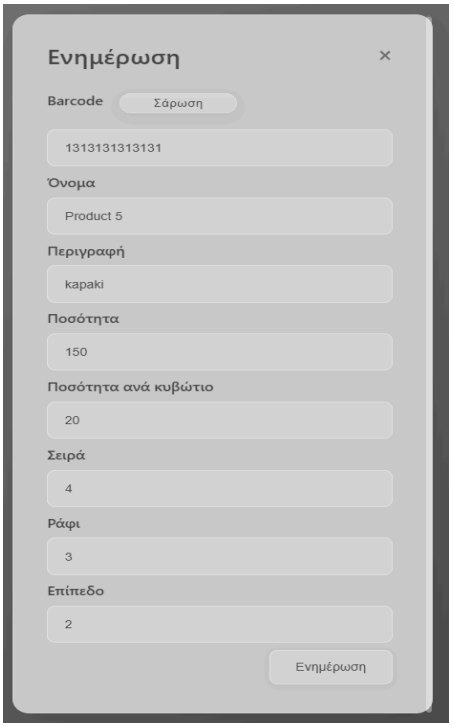
- Άνοιγμα → εμφανίζει όλα τα στοιχεία του προϊόντος σε νέο modal.
- Ενημέρωση → επιτρέπει την τροποποίηση των στοιχείων (όνομα, ποσότητα, θέση κ.λπ.).
- Διαγραφή → αφαιρεί το προϊόν από τη βάση.



Σχήμα 3.5.5: Προβολή αποτελέσματος & Κουμπιά «Άνοιγμα», «Ενημέρωση», «Διαγραφή»

➔ **Βήμα 3 – Προβολή στοιχείων και Ενημέρωση (προαιρετικά)**

Όταν πατηθεί το Άνοιγμα, ανοίγει ένα νέο παράθυρο με πλήρη καρτέλα του προϊόντος. Από εκεί ο χρήστης μπορεί είτε να ενημερώσει τα στοιχεία του, είτε απλώς να τα δει.



Σχήμα 3.5.5: Προβολή πλήρης καρτέλα του προϊόντος ή «Ενημέρωση»

Αν επιλεγεί η τροποποίηση, ο χρήστης αλλάζει τα πεδία και πατάει «Αποθήκευση». Αυτό ενεργοποιεί ένα **PATCH** αίτημα στο backend, μέσω της συνάρτησης updateProduct(barcode, product) από το αρχείο api.js, η οποία καλεί το endpoint **PATCH /products/:barcode**, το οποίο με τη σειρά του συνδέεται με τη λειτουργία **updateProduct()** που βρίσκεται στο services.ts.

```
export async function updateProduct(barcode, product) {
  const response = await fetch(`${API_BASE}/products/${barcode}`, {
    method: "PATCH",
    headers: getHeaders(),
    body: JSON.stringify(product),
  });
  return response.json();
}
```

➔ **Βήμα 4 – Διαγραφή (προαιρετικά)**

Ο χρήστης έχει και τη δυνατότητα να διαγράψει το προϊόν. Όταν πατήσει «Διαγραφή», γίνεται DELETE αίτημα στο backend μέσω της fetch() από το frontend, που καλεί το endpoint **DELETE /products/:barcode**. Η εντολή αυτή αντιστοιχεί στη λειτουργία **deleteProduct()** στο αρχείο services.ts.

```
export async function deleteProduct(barcode) {
  const response = await fetch(`${apiUrl}/products/${barcode}`, {
    method: 'DELETE',
    headers,
  });
  return response.json();
}
```

Με την επιτυχή ολοκλήρωση της διαγραφής, το προϊόν αφαιρείται από τη βάση και η λίστα ενημερώνεται.

Η δυνατότητα αναζήτησης και διαχείρισης προϊόντων κάνει την καθημερινή εργασία στην αποθήκη πολύ πιο γρήγορη και εύκολη. Ο χρήστης δεν χρειάζεται να γνωρίζει τον ακριβή κωδικό, αφού μπορεί να αναζητήσει και με βάση το όνομα ή να σκανάρει τον γραμμωτό κώδικα.

3.5.3 Δημιουργία και Ολοκλήρωση Παραγγελίας

Η λειτουργία παραγγελίας επιτρέπει στα καταστήματα να στέλνουν αιτήματα για επαναπρομήθεια και στον αποθηκάριο να οργανώνει τη συλλογή των προϊόντων. Η διαδικασία περιλαμβάνει τη δημιουργία της παραγγελίας, τη συγκέντρωση των ζητούμενων προϊόντων και την ολοκλήρωσή της μέσα από την εφαρμογή.

Όλα γίνονται πιο εύκολα και γρήγορα, αφού ο αποθηκάριος γνωρίζει ήδη πού βρίσκεται κάθε προϊόν στην αποθήκη — σε ποιο ράφι, σε ποιο επίπεδο και σε ποιο διάδρομο. Έτσι, εξοικονομεί χρόνο, αποφεύγει μπλέξιμο και μπορεί να διεκπεραιώσει παραγγελίες πιο αποτελεσματικά.

➔ **Βήμα 1 – Νέα παραγγελία**

Ο χρήστης πατά το κουμπί «Παραγγελία» στο μενού και επιλέγει «Νέα παραγγελία». Ανοίγει modal όπου μπορεί να προσθέσει προϊόντα που ζητά το κατάστημα.

Σχήμα 3.5.6: Επιλογή νέας παραγγελίας

Αφού συμπληρωθεί η λίστα προϊόντων, πατάει «Ολοκλήρωση Παραγγελίας». Αυτό καλεί τη συνάρτηση **createOrder(order)** από το αρχείο api.js, η οποία στέλνει **POST** αίτημα στο endpoint **POST /orders**, το οποίο οδηγεί στην εκτέλεση της **createOrder()** στο services.ts, όπου γίνεται η εισαγωγή της παραγγελίας και των πληροφοριών της στη βάση.

```
export async function createOrder(order) {
  const response = await fetch(`${API_BASE}/orders`, {
    method: 'POST',
    headers: getHeaders(),
    body: JSON.stringify(order),
  });
  return response.json();
}
```

➔ **Βήμα 2 – Ενεργές παραγγελίες**

Μόλις δημιουργηθεί η παραγγελία, ο αποθηκάριος μπορεί να τη δει στη λίστα με τις ενεργές παραγγελίες. Από εκεί επιλέγει μία για να ξεκινήσει τη συλλογή προϊόντων.

Σχήμα 3.5.7: Ενεργές παραγγελίες και επιλογή παραγγελίας προς συλλογή

➔ Βήμα 3 – Συλλογή προϊόντων και ολοκλήρωση

Μέσα στην παραγγελία εμφανίζεται η λίστα με τα ζητούμενα προϊόντα και οι ποσότητες. Ο χρήστης συλλέγει τα προϊόντα και τα επιβεβαιώνει. Όταν είναι έτοιμος, πατά «Ολοκλήρωση».

Η ολοκλήρωση καλεί τη συνάρτηση **completeOrder(orderId)** από το αρχείο api.js, η οποία στέλνει **POST** αίτημα στο endpoint **POST/orders/:id/complete**. Από εκεί εκτελείται η **completeOrder()** στο services.ts, όπου η παραγγελία σημειώνεται ως ολοκληρωμένη και αφαιρούνται τα τεμάχια από τα αποθέματα.

```
export async function completeOrder(id) {
  const response = await fetch(`${API_BASE}/orders/${id}/complete`, {
    method: "POST",
    headers: getHeaders(),
  });
  return response.json();
}
```

Η λειτουργία παραγγελίας επιτρέπει την οργανωμένη εκτέλεση αιτημάτων από τα καταστήματα και διασφαλίζει ότι η αποθήκη μένει πάντα ενημερωμένη με τις σωστές ποσότητες.

3.5.4 Απογραφή

Η απογραφή είναι μια από τις πιο κρίσιμες λειτουργίες για μια αποθήκη, καθώς επιτρέπει στον υπεύθυνο να καταγράψει τις πραγματικές ποσότητες των προϊόντων που βρίσκονται στα ράφια. Μέσω της εφαρμογής, η διαδικασία αυτή γίνεται με απλό και γρήγορο τρόπο, μέσα από τη σάρωση barcode και την εισαγωγή των μετρημένων τεμαχίων.

Βήμα 1 – Επιλογή της λειτουργίας «Απογραφή»

Σχήμα 3.5.18: Άνοιγμα modal απογραφής – αρχική κατάσταση

Βήμα 2 – Σάρωση προϊόντος

Με το πάτημα του κουμπιού σάρωσης, ενεργοποιείται η κάμερα μέσω της βιβλιοθήκης Quagga.js. Ο χρήστης σαρώνει τον γραμμωτό κώδικα του προϊόντος και, αν το barcode αναγνωριστεί, η εφαρμογή φέρνει αυτόματα τα στοιχεία του προϊόντος από τη βάση.

Σχήμα 3.5.12: Σαρωμένο προϊόν και εμφάνιση στοιχείων για απογραφή

Βήμα 3 – Καταγραφή φυσικής ποσότητας

Αφού εμφανιστούν τα στοιχεία του προϊόντος (barcode και όνομα), ο χρήστης εισάγει τον αριθμό των τεμαχίων που βρήκε στο ράφι. Έπειτα πατά «Ενημέρωση», για να καταχωρηθεί η απογραφή στο σύστημα.

Βήμα 4 – Υποβολή στο backend

Η απογραφή καταχωρείται με τη χρήση της συνάρτησης createApografi(input) από το api.js, η οποία στέλνει POST αίτημα στο **endpoint POST /apografes**. Από εκεί καλείται η συνάρτηση **createApografi()** στο services.ts, η οποία καταγράφει τη νέα απογραφή

Η απογραφή είναι πολύ χρήσιμη για να βλέπει ο αποθηκάριος τι υπάρχει πραγματικά στα ράφια. Αν κάτι λείπει ή έχει αλλάξει, μπορεί να το διορθώσει και να ενημερώσει σωστά τη βάση. Έτσι η αποθήκη παραμένει οργανωμένη και ακριβής.Επίσης, η απογραφή είναι σημαντική στο τέλος της χρονιάς, γιατί βοηθά την επιχείρηση να ξέρει τι ακριβώς διαθέτει όταν ξεκινάει η νέα χρονιά. Είναι ένα απαραίτητο βήμα για να υπάρχει σωστή εικόνα του αποθέματος.

3.6 Επίλογος

Στην Ενότητα 3 παρουσιάστηκε αναλυτικά η ανάπτυξη του συστήματος, από τη σχεδίαση της αρχιτεκτονικής μέχρι την υλοποίηση του backend και του frontend. Είδαμε πώς δομήθηκε το REST API, πώς υλοποιήθηκαν οι βασικές υπηρεσίες με Prisma και Express.js, και πώς διασφαλίστηκε η ασφάλεια και η σταθερότητα του backend μέσω authentication και error handling.Στη συνέχεια, εξετάσαμε τη λειτουργικότητα της διεπαφής χρήστη και την ενσωμάτωση σημαντικών λειτουργιών όπως η προσθήκη, αναζήτηση και ενημέρωση προϊόντων, καθώς και η δημιουργία και ολοκλήρωση παραγγελιών. Όλες αυτές οι ενότητες δείχνουν πώς το σύστημα υποστηρίζει με πρακτικό και αποδοτικό τρόπο τις καθημερινές ανάγκες μιας αποθήκης.

Στην επόμενη ενότητα θα δούμε πώς δοκιμάστηκε το σύστημα, ποια σενάρια αξιολογήθηκαν και ποιες βελτιώσεις προέκυψαν μέσα από την πρακτική του χρήση.

Κεφάλαιο 4ο: Δοκιμές και Αξιολόγηση

Σε αυτό το κεφάλαιο παρουσιάζονται οι δοκιμές που έγιναν για να ελεγχθεί αν η εφαρμογή λειτουργεί σωστά και είναι εύχρηστη στην πράξη. Μέσα από αυτές τις δοκιμές διαπιστώθηκε αν ο αποθηκάριος μπορεί να κάνει άνετα τις βασικές εργασίες και αν όλα τα βασικά σημεία της εφαρμογής δουλεύουν χωρίς προβλήματα.

Εκτός από το τεχνικό κομμάτι, δόθηκε έμφαση και στο πόσο εύκολη είναι η χρήση της εφαρμογής από τον τελικό χρήστη, χωρίς πολύπλοκα μενού ή περιττές διαδικασίες.

4.1 Περιβάλλον και σενάρια δοκιμών

Οι δοκιμές έγιναν σε τοπικό υπολογιστή και σε κινητό τηλέφωνο, ώστε να δούμε πώς λειτουργεί η εφαρμογή σε διαφορετικές συσκευές. Χρησιμοποιήθηκε ο Google Chrome για το frontend και ο Node.js server για το backend.

Η σάρωση με κάμερα δοκιμάστηκε σε κινητό Android & IOS για να διαπιστωθεί αν μπορεί να διαβάσει σωστά τα barcodes στην πράξη, ακόμα και σε μέτριο φωτισμό.

Τα βασικά που δοκιμάστηκαν:

- Είσοδος στην εφαρμογή με σωστά και λάθος στοιχεία
- Προσθήκη νέου προϊόντος με και χωρίς σάρωση
- Αναζήτηση προϊόντος με όνομα ή barcode
- Ενημέρωση ή διαγραφή προϊόντος
- Δημιουργία παραγγελίας και επιβεβαίωση ολοκλήρωσης
- Μείωση αποθεμάτων μετά από παραγγελία
- Καταγραφή ποσοτήτων την απογραφή
- Έλεγχος αν εμφανίζονται σωστά τα μηνύματα λάθους ή επιβεβαίωσης

Κάθε λειτουργία ελέγχθηκε για να φανεί αν:

- Δουλεύει σωστά από την αρχή ως το τέλος
- Ενημερώνει σωστά τη βάση
- Δίνει σαφή ενημέρωση στον χρήστη για το τι έγινε

4.2 Τεχνικά ζητήματα που προέκυψαν

Κατά τη διάρκεια της ανάπτυξης και των δοκιμών εμφανίστηκαν κάποια μικρά τεχνικά προβλήματα, τα οποία αντιμετωπίστηκαν σταδιακά. Αυτά τα ζητήματα είναι φυσιολογικά σε κάθε project και βοήθησαν να βελτιωθεί ακόμα περισσότερο η εφαρμογή.

Μερικά παραδείγματα:

- Δυσκολία στη σάρωση barcode σε χαμηλό φωτισμό:

  Σε κάποιες περιπτώσεις, ειδικά σε χώρους με κακό φωτισμό ή με κακή κάμερα, η σάρωση με Quagga.js δεν λειτουργούσε καλά. Λύθηκε με πιο προσεκτική σκόπευση και χρήση φυσικού φωτός.

- Καταχώριση με άδεια πεδία:

  Στην αρχή, η φόρμα προσθήκης προϊόντος επέτρεπε την αποστολή χωρίς όλα τα στοιχεία. Προστέθηκαν έλεγχοι ώστε να ζητούνται υποχρεωτικά πεδία (π.χ. barcode, όνομα, ποσότητα).

- Αναζήτηση με λάθος barcode:

  Όταν ο χρήστης έβαζε λάθος ή ανύπαρκτο barcode, δεν υπήρχε σαφές μήνυμα. Προστέθηκε ενημέρωση τύπου “Το προϊόν δεν βρέθηκε”.

- Modal που δεν φαινόταν σωστά σε κινητά:

  Κάποια παράθυρα (modals) δεν εμφανίζονταν σωστά σε μικρές οθόνες. Έγιναν διορθώσεις ώστε να είναι πιο ευέλικτα και να προσαρμόζονται στο μέγεθος της οθόνης.

- Ολοκλήρωση παραγγελίας χωρίς συλλογή όλων των προϊόντων:

  Αν ο χρήστης πάταγε "Ολοκλήρωση" χωρίς να έχει περάσει όλα τα προϊόντα, δεν υπήρχε έλεγχος. Προστέθηκε σχετική προειδοποίηση στο backend.

Αν και πρόκειται για απλά θέματα, η επίλυσή τους βελτίωσε σημαντικά την εμπειρία του χρήστη και έκανε την εφαρμογή πιο σταθερή.

4.3 Αξιολόγηση Χρηστικότητας και Λειτουργικότητας

Η εφαρμογή φτιάχτηκε για να είναι όσο γίνεται πιο απλή και κατανοητή για τον αποθηκάριο. Οι βασικές λειτουργίες είναι συγκεντρωμένες στο κεντρικό μενού, και κάθε ενέργεια (όπως προσθήκη, αναζήτηση ή παραγγελία) γίνεται με λίγα και ξεκάθαρα βήματα.

Κατά τη χρήση της εφαρμογής, φάνηκε ότι:

- Οι οθόνες είναι καθαρές και εύκολες στην πλοήγηση.
- Η σάρωση barcode γλιτώνει χρόνο σε σχέση με τη χειροκίνητη εισαγωγή.
- Τα μηνύματα που εμφανίζονται βοηθούν τον χρήστη να καταλάβει τι έκανε σωστά ή λάθος.
- Η διαδικασία συλλογής παραγγελίας είναι απλή και καθοδηγούμενη.
- Η απογραφή γίνεται πιο γρήγορα και χωρίς χαρτιά ή μπλοκάκια.

Γενικά, η εφαρμογή βοηθάει στην καθημερινή δουλειά της αποθήκης, μειώνει τα λάθη και κάνει τη διαδικασία πιο οργανωμένη.

4.4 Επίλογος

Μέσα από τις δοκιμές που έγιναν, φάνηκε ότι η εφαρμογή ανταποκρίνεται σωστά στις βασικές λειτουργίες και εξυπηρετεί αποτελεσματικά τον σκοπό της. Παρά τα μικρά τεχνικά ζητήματα που προέκυψαν, η συνολική εμπειρία χρήσης ήταν θετική. Ο αποθηκάριος μπορεί να κάνει τις καθημερινές του εργασίες γρήγορα και χωρίς περιττή δυσκολία, ενώ η εφαρμογή παραμένει σταθερή και εύχρηστη.

Τα αποτελέσματα αυτών των δοκιμών έδωσαν και χρήσιμες ιδέες για μελλοντικές βελτιώσεις, οι οποίες θα παρουσιαστούν στο επόμενο κεφάλαιο.

Κεφάλαιο 5ο: Συμπεράσματα και Μελλοντικές Προεκτάσεις

Η ανάπτυξη της εφαρμογής ολοκληρώθηκε με επιτυχία και καλύπτει βασικές ανάγκες μιας αποθήκης. Σε αυτό το κεφάλαιο γίνεται μια σύντομη ανακεφαλαίωση της υλοποίησης, παρουσιάζονται ιδέες για επεκτάσεις στο μέλλον, καθώς και παρατηρήσεις που προέκυψαν μέσα από τη χρήση και τις δοκιμές.

5.1 Ανακεφαλαίωση υλοποίησης

Η εφαρμογή που δημιουργήθηκε καλύπτει όλες τις βασικές ανάγκες μιας αποθήκης: προσθήκη νέων προϊόντων, αναζήτηση, ενημέρωση, παραγγελίες και απογραφή.Ο χρήστης μπορεί να κάνει τις εργασίες του εύκολα και γρήγορα, χωρίς να χρειάζεται να θυμάται πολλά βήματα ή να γράφει χειροκίνητα αριθμούς. Η χρήση του barcode κάνει τη διαδικασία ακόμα πιο άμεση.

Η συνολική λειτουργία της εφαρμογής είναι σταθερή και πρακτική. Ο συνδυασμός των τεχνολογιών (Node.js για το backend, JavaScript για το frontend και Prisma για τη βάση) λειτούργησε καλά και έκανε την ανάπτυξη πιο απλή και οργανωμένη.

5.2 Πιθανές επεκτάσεις(π.χ. mobile app, roles, γραφήματα)

1.Χρήση εξειδικευμένης συσκευής αποθήκης (με ενσωματωμένο scanner)

Αν στο μέλλον προχωρήσει η μεταφορά της εφαρμογής σε native mobile μορφή (π.χ. Android), τότε η χρήση επαγγελματικής συσκευής αποθήκης με λειτουργικό σύστημα Android θα βοηθήσει ακόμα περισσότερο. Θα επιτρέψει καλύτερη απόδοση, πρόσβαση στο φακό, σάρωση, ειδοποιήσεις

Αντί για απλή κάμερα κινητού ή υπολογιστή, μπορεί να χρησιμοποιηθεί ειδική φορητή συσκευή αποθήκης — δηλαδή ένα «robust» PDA scanner (σαν κινητό), που είναι σχεδιασμένο για βιομηχανικό περιβάλλον. Τέτοιες συσκευές διαθέτουν:

- **Ενσωματωμένο laser scanner** για πιο γρήγορη και ακριβή σάρωση
- **Προστασία από πτώσεις** και δύσκολες συνθήκες
- **Φακό, GPS, Wi-Fi** και άλλες λειτουργίες ειδικά για αποθήκες.

Αν στο μέλλον η εφαρμογή μεταφερθεί σε **mobile περιβάλλον**, τότε μια τέτοια συσκευή μπορεί να αξιοποιηθεί πλήρως. Η χρήση τέτοιας τεχνολογίας συνδυάζει την ευκολία του κινητού με την αξιοπιστία του επαγγελματικού εξοπλισμού.

2.Βελτίωση της τεχνολογίας σάρωσης

Αν προστεθεί εξωτερικός barcode scanner (π.χ. με καλώδιο ή bluetooth), τότε δεν χρειάζεται να αλλάξει η τεχνολογία σάρωσης στην εφαρμογή. Ο σαρωτής «πληκτρολογεί» αυτόματα τον κωδικό στο πεδίο και δεν απαιτείται κάμερα ή ειδική βιβλιοθήκη όπως το Quagga.js. Άρα, η βελτίωση της σάρωσης μέσω κάμερας δεν είναι απαραίτητη σε αυτό το σενάριο.

3.Γραφική απεικόνιση αποθήκης και έξυπνη καθοδήγηση

Αν προστεθεί οπτικός χάρτης της αποθήκης, ο αποθηκάριος θα μπορούσε να δει «πού να πάει» σε κατάσταση παραγγελίας ή απογραφής. Θα γλιτώνει χρόνο και θα αποφεύγει λάθη.

4.Ρόλοι και διαβαθμισμένα δικαιώματα

Ο αποθηκάριος να έχει πρόσβαση μόνο σε λειτουργίες διαχείρισης προϊόντων, ενώ ο υπεύθυνος αποθήκης να μπορεί να δει απογραφές, να εγκρίνει διαγραφές, κλπ.

5.Στατιστικά και γραφήματα

Π.χ. πόσα προϊόντα υπάρχουν ανά ράφι, ποιες παραγγελίες εκκρεμούν, μέσος χρόνος παραλαβής κλπ. Όλα αυτά θα μπορούσαν να βοηθήσουν τη διοίκηση να έχει πιο καθαρή εικόνα.

6.Εμφάνιση ιστορικού ενεργειών

Να υπάρχει καταγραφή για το πότε προστέθηκε ή τροποποιήθηκε ένα προϊόν, πότε έγινε παραγγελία, ή απογραφή.

7.Αυτόματες ειδοποιήσεις

Να εμφανίζεται προειδοποίηση όταν κάποιο προϊόν φτάνει κάτω από ένα όριο (π.χ. «προσοχή: χαμηλό απόθεμα»), ώστε να δημιουργείται αυτόματα παραγγελία.

8.Εξαγωγή σε Excel ή PDF

Η δυνατότητα εξαγωγής λιστών (όπως προϊόντα, παραγγελίες ή απογραφές) σε αρχείο Excel ή PDF μπορεί να βοηθήσει τον υπεύθυνο στην καθημερινή του οργάνωση.

5.3 Προτάσεις για βελτιώσεις

1.Επιβεβαίωση σε κρίσιμες ενέργειες

Όταν ο χρήστης διαγράφει ή ολοκληρώνει παραγγελία, να εμφανίζεται ένα popup για επιβεβαίωση.

2.Ένδειξη φόρτωσης (loading)

Όταν γίνεται κάποια κλήση στο backend, θα μπορούσε να εμφανίζεται ένα «παρακαλώ περιμένετε…».

3.Πιο καθαρά μηνύματα σφαλμάτων

Σε περιπτώσεις όπως διπλό barcode ή λάθος πεδία, το σύστημα να εμφανίζει πιο κατανοητά μηνύματα στον χρήστη.

4.Έξυπνη επιλογή τοποθέτησης (ράφι, διάδρομος, επίπεδο)

Μια ακόμη βελτίωση αφορά τον τρόπο με τον οποίο επιλέγεται η θέση τοποθέτησης των προϊόντων (διάδρομος, ράφι, επίπεδο). Αντί να εισάγονται οι τιμές χειροκίνητα, τα πεδία μπορούν να γίνουν drop-down λίστες, ώστε ο χρήστης να διαλέγει πιο γρήγορα και με ακρίβεια. Παράλληλα, το σύστημα θα μπορούσε να εμφανίζει μόνο τις διαθέσιμες θέσεις. Για παράδειγμα, αν ένα ράφι έχει ήδη καλυφθεί πλήρως (και τα 3 του επίπεδα έχουν χρησιμοποιηθεί), τότε αυτό το ράφι δεν θα εμφανίζεται στη λίστα επιλογών. Το ίδιο μπορεί να ισχύει και για τους διαδρόμους. Έτσι αποφεύγονται λάθη, η διαδικασία γίνεται πιο γρήγορη και ο αποθηκάριος βλέπει μόνο τις πραγματικά ελεύθερες θέσεις.

5.4 Επίλογος

Η εφαρμογή που αναπτύχθηκε καταφέρνει να καλύψει με απλό και πρακτικό τρόπο τις βασικές ανάγκες μιας αποθήκης. Ο χρήστης μπορεί να κάνει τις βασικές του εργασίες εύκολα, ενώ το σύστημα τον καθοδηγεί και τον βοηθά να μην κάνει λάθη.

Μέσα από τη χρήση της, προέκυψαν και ιδέες για το πώς μπορεί να γίνει ακόμα καλύτερη στο μέλλον, με πιο έξυπνη διαχείριση θέσεων, πιο γρήγορη σάρωση, στατιστικά και νέες δυνατότητες.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Wikipedia, “Warehouse management system,” [Online].

Available: https://en.wikipedia.org/wiki/Warehouse_management_system

[2] Katana MRP, “Barcode Inventory System,” [Online].

Available: https://katanamrp.com/barcode-inventory-system/

[3] Novacura, “Barcode WMS: Best Practices,” [Online].

Available: https://www.novacura.com/barcode-wms-best-practices/

[4] QuaggaJS, “Barcode Scanner Library,” [Online].

Available: https://serratus.github.io/quaggaJS/

[5] Prisma, “Prisma ORM Documentation,” [Online].

Available: https://www.prisma.io/docs/

[6] Node.js, “Official Node.js Documentation,” [Online].

Available: https://nodejs.org/en/docs

[7] Express.js, “Express - Node.js Web Application Framework,” [Online].

Available: https://expressjs.com/

[6] Node.js, “Official SQLite Documentation,” [Online].

Available: https://www.sqlite.org/docs.html

[6] Quagga.js, “Quagga.js Barcode Scanner Library” [Online].

Available:https://serratus.github.io/quaggaJS/