



ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ
«Ανάπτυξη Εφαρμογής Συστήματος Διαχείρισης
Παραγγελιών Επιχειρήσεων Εστίασης»



Του φοιτητή
Μακρέβσκι Γκόριαν
Αρ. Μητρώου: 133983

Επιβλέπων
Αντωνίου Ευστάθιος
Καθηγητής

Ημερομηνία 31/8/2026

Τίτλος Π.Ε. Ανάπτυξη Εφαρμογής Συστήματος Διαχείρισης Παραγγελιών Επιχειρήσεων Εστίασης
Κωδικός Π.Ε. 24327

Ονοματεπώνυμο φοιτητή Μακρέβσκι Γκόριαν
Ονοματεπώνυμο εισηγητή Αντωνίου Ευστάθιος
Ημερομηνία ανάληψης Π.Ε. 13/01/2025
Ημερομηνία περάτωσης Π.Ε. 31/8/2026

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

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

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

«Αφιερώνεται στην οικογένειά μου.»

Πρόλογος

Η παρούσα πτυχιακή εργασία είναι το αποτέλεσμα της ακαδημαϊκής και προσωπικής μου πορείας στο πεδίο της ανάπτυξης λογισμικού. Έχοντας ήδη επαγγελματική τριβή με την ανάπτυξη εφαρμογών για κινητά, κύριο κίνητρό μου για την επιλογή του συγκεκριμένου θέματος υπήρξε η επιθυμία να διευρύνω τις γνώσεις μου πέρα από τα στενά όρια της διεπαφής χρήστη.

Στόχος μου ήταν να εμβαθύνω στη σχεδίαση ολοκληρωμένων Full-Stack συστημάτων, συνδυάζοντας προηγμένες αρχιτεκτονικές mobile εφαρμογών με ένα λειτουργικό backend, καθώς και την ενσωμάτωση τεχνολογιών Τεχνητής Νοημοσύνης στη συσκευή με το Google ML Kit.

Η υλοποίηση του συστήματος OrderFlow υπήρξε μια ιδιαίτερα δημιουργική και απαιτητική πρόκληση, μέσω της οποίας κατάφερα όχι μόνο να αποκτήσω πολύτιμες νέες γνώσεις, αλλά και να αναπτύξω μια σφαιρική αντίληψη για τη σχεδίαση συστημάτων υψηλών απαιτήσεων.

Περίληψη

Η παρούσα πτυχιακή εργασία παρουσιάζει τη μελέτη, τη σχεδίαση και την υλοποίηση του συστήματος OrderFlow, μιας ολοκληρωμένης Full-Stack πλατφόρμας διαχείρισης παραγγελιοληψίας για χώρους εστίασης. Σκοπός του συστήματος είναι ο εκσυγχρονισμός και η βελτιστοποίηση της ροής εργασιών μεταξύ σερβιτόρων, παρασκευαστηρίου και διοίκησης.

Αρχιτεκτονικά, το σύστημα βασίζεται σε δομή Monopero και αποτελείται από τρεις εξειδικευμένες εφαρμογές Flutter, οι οποίες επικοινωνούν με ένα κεντρικό REST API αναπτυγμένο σε Java Spring Boot και βάση δεδομένων MariaDB. Η εφαρμογή του διαχειριστή παρέχει πλήρη έλεγχο του καταλόγου και του προσωπικού. Η εφαρμογή της κουζίνας προβάλλει σε πραγματικό χρόνο την κατάσταση των παραγγελιών, ενώ η εφαρμογή του σερβιτόρου εστιάζει στην ταχύτητα καταχώρισης.

Καινοτόμο στοιχείο της εργασίας αποτελεί η ενσωμάτωση μηχανισμού αναγνώρισης χειρόγραφης γραφής (Google ML Kit Digital Ink Recognition) στην εφαρμογή του σερβιτόρου, σε συνδυασμό με έναν αλγόριθμο ανάλυσης συντομογραφιών. Ο μηχανισμός αυτός επιτρέπει την ακαριαία μετατροπή χειρόγραφων σύντομων σημειώσεων σε δομημένες παραγγελίες.

Τα αποτελέσματα επιβεβαιώνουν ότι η προτεινόμενη αρχιτεκτονική εξασφαλίζει υψηλή απόδοση, επεκτασιμότητα και σημαντική μείωση του χρόνου παραγγελιοληψίας, προσφέροντας μια σύγχρονη και εργονομική λύση για τον κλάδο της εστίασης.

«Development of an Order Management System Application for Hospitality Businesses»

«Gorjan Makrevski»

Abstract

This thesis presents the design, analysis, and implementation of OrderFlow. It is a comprehensive full-stack order management platform tailored for hospitality businesses. The primary objective of the system is to streamline operational workflows and enhance communication between waitstaff, kitchen personnel, and management.

Architecturally is structured as a monorepo. The platform comprises three specialized Flutter applications communicating with a centralized REST API built with Java Spring Boot and backed by a MariaDB database. The Admin application enables complete management of products, categories, and staff. The kitchen display application tracks order status in real time; and the Waiter application focuses on ergonomic, rapid order placement.

A key technical innovation of the project is the integration of an on-device handwriting recognition engine (Google ML Kit Digital Ink Recognition). It is combined with a custom Shorthand Parser within the Waiter application. This hybrid mechanism enables the instantaneous conversion of handwritten shorthand notes into structured order items.

The evaluation of the system demonstrates high performance, scalability, and clean code maintainability through state management patterns like BLoC. Ultimately, OrderFlow significantly reduces order placement time, providing a modern and efficient solution for the food service industry.

Ευχαριστίες

Με την εκπόνηση της παρούσας εργασίας ολοκληρώνω τον κύκλο προπτυχιακών σπουδών μου στο διεθνές πανεπιστήμιο της Ελλάδος.

Σε αυτό το σημείο θα ήθελα να εκφράσω τις θερμές μου ευχαριστίες στον επιβλέποντα καθηγητή μου, κύριο Αντωνίου Ευστάθιο, για την ανάθεση του θέματος, την εμπιστοσύνη του και την άψογη συνεργασία που είχαμε κατά τη διάρκεια εκπόνησης του έργου.

Τέλος οφείλω ένα μεγάλο ευχαριστώ στην οικογένειά μου και στα φιλικά μου πρόσωπα για την αμέριστη συμπαράσταση και την υπομονή που έδειξαν σε όλη τη διάρκεια των σπουδών μου στο τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων.

Περιεχόμενα

Πρόλογος.....	4
Περίληψη.....	5
Abstract.....	6
Ευχαριστίες.....	7
Περιεχόμενα.....	8
Συντομογραφίες.....	11
Κεφάλαιο 1ο: Εισαγωγή.....	12
1.1 Εισαγωγή.....	12
1.2 Πλαίσιο και Κίνητρο.....	12
1.3 Σκοπός και Στόχοι της Πτυχιακής Εργασίας.....	13
1.4 Παραδοτέα.....	13
1.5 Δομή του Κειμένου.....	14
Κεφάλαιο 2ο: Θεωρητικό Υπόβαθρο και Σχετικές Τεχνολογίες.....	15
2.1 Εισαγωγή.....	15
2.2 Συστήματα Διαχείρισης Παραγγελιών.....	15
2.3 Τεχνολογίες Ανάπτυξης Διεπαφών Χρήστη.....	16
2.3.1 Η Πλατφόρμα Flutter και η Αρχιτεκτονική Απόδοσης Γραφικών.....	16
2.3.2 Η Γλώσσα Προγραμματισμού Dart.....	16
2.3.3 Διαχείριση Κατάστασης: Το Αρχιτεκτονικό Πρότυπο BLoC.....	17
2.4 Τεχνολογίες Υποδομής.....	17
2.4.1 Η Γλώσσα Java (Έκδοση 21).....	17
2.4.2 Το Πλαίσιο Spring Boot 3.....	18
2.4.3 Η Αρχιτεκτονική REST (Representational State Transfer).....	18
2.5 Βάση Δεδομένων.....	19
2.5.1 Σχεσιακό Μοντέλο και Ιδιότητες ACID.....	19
2.5.2 Τεχνολογία ORM (Object-Relational Mapping).....	19
2.6 Ο κύκλος ζωής των δεδομένων.....	19
Κεφάλαιο 3ο: Ανάλυση Απαιτήσεων και Σχεδίαση Συστήματος.....	21
3.1 Εισαγωγή.....	21
3.2 Ανάλυση Απαιτήσεων και Περιπτώσεις Χρήσης.....	21
3.2.1 Απαιτήσεις Διαχειριστή.....	21
3.2.2 Απαιτήσεις Σερβιτόρου.....	22

3.2.3	Απαιτήσεις Προσωπικού Παρασκευής.....	23
3.3	Σχεδίαση Διεπαφής Χρήστη.....	24
3.3.1	Οπτική Ταυτότητα και Τυπογραφία.....	24
3.3.2	Εργονομία και Πλοήγηση.....	24
3.3.3	Σύνοψη Κεφαλαίου.....	24
Κεφάλαιο 4ο: Υλοποίηση Συστήματος OrderFlow.....		26
4.1	Εισαγωγή.....	26
4.2	Υλοποίηση του Backend Συστήματος.....	26
4.2.1	Αρχιτεκτονική Multi-Module του Backend.....	26
4.2.2	Μοντέλα Δεδομένων - Οντότητες.....	27
4.2.3	Επίπεδο Πρόσβασης Δεδομένων (DAOs).....	31
4.2.4	Επίπεδο Ελεγκτών (REST Controllers).....	33
4.2.5	Διαχείριση Ρυθμίσεων και Σύνδεσης με τη Βάση Δεδομένων.....	34
4.2.6	4.2.6 Δοκιμές Ενσωμάτωσης (Integration Testing).....	35
4.3	Ανάπτυξη του Κοινόχρηστου Πακέτου.....	36
4.4	Υλοποίηση Εφαρμογής Διαχείρισης.....	44
4.4.1	Κεντρικός Ελεγκτής και Polling.....	44
4.4.2	Δυναμική Διεπαφή και Πλοήγηση.....	44
	4.4.3 Σύστημα Διαχείρισης Καταλόγου και Συντομογραφιών.....	44
4.4.3	Σύστημα Διαχείρισης Προσωπικού.....	45
4.4.4	Εποπτεία Παραγγελιών.....	45
4.5	Υλοποίηση Εφαρμογής Εξυπηρέτησης.....	45
4.5.1	Αρχιτεκτονική και Διαχείριση Κατάστασης (State Management).....	45
4.5.2	Διεπαφή Χρήστη και Πλοήγηση.....	46
4.5.3	Παραμετροποίηση Προϊόντων.....	46
4.5.4	Συντακτικός Αναλυτής Συντομογραφιών.....	46
4.5.5	Λογική και Αλγόριθμος Ανάλυσης.....	47
4.5.6	4.5.5 Διαχείριση Ενεργών Παραγγελιών και Ταμεία.....	48
4.6	Υλοποίηση Εφαρμογής Παραγωγής.....	48
4.6.1	Αυτόματη Ανανέωση Δεδομένων.....	49
4.6.2	Διεπαφή Χρήστη Παραγωγής.....	49
4.6.3	Κύκλος Ζωής της Παραγγελίας στην Παραγωγή.....	49
4.6.4	Διαχείριση Αποθεμάτων με Αισιόδοξη Ενημέρωση.....	50
4.7	Επίλογος.....	50
Κεφάλαιο 5ο: Συμπεράσματα και Προτάσεις Βελτίωσης.....		52

5.1	Συνοπτική Αξιολόγηση του Έργου.....	52
5.2	Προβλήματα που αντιμετωπίστηκαν.....	52
5.3	Προτάσεις για Μελλοντικές Επεκτάσεις.....	52
5.4	Επίλογος.....	53
ΒΙΒΛΙΟΓΡΑΦΙΑ.....		54

Συντομογραφίες

Δ.Ε.	Διπλωματική Εργασία
ΔΙΠΑΕ	Διεθνές Πανεπιστήμιο Ελλάδος
Π.Ε.	Πτυχιακή Εργασία
API	Application Programming Interface (Διεπαφή Προγραμματισμού Εφαρμογών)
BLoC	Business Logic Component (Στοιχείο Επιχειρησιακής Λογικής)
CRUD	Create, Read, Update, Delete (Δημιουργία, Ανάγνωση, Ενημέρωση, Διαγραφή)
HTTP	Hypertext Transfer Protocol (Πρωτόκολλο Μεταφοράς Υπερκειμένου)
JSON	JavaScript Object Notation (Μορφοποίηση Αντικειμένων JavaScript)
ORM	Object-Relational Mapping
REST	Representational State Transfer (Αντιπροσωπευτική Μεταφορά Κατάστασης)
SQL	Structured Query Language (Δομημένη Γλώσσα Ερωτημάτων)
UI	User Interface (Διεπαφή Χρήστη)
UX	User Experience (Εμπειρία Χρήστη)

Κεφάλαιο 1ο: Εισαγωγή

1.1 Εισαγωγή

Η ραγδαία εξέλιξη της τεχνολογίας κατά την τελευταία δεκαετία έχει αναδιαμορφώσει ριζικά τον τρόπο λειτουργίας των σύγχρονων επιχειρήσεων. Επιβάλλει τον ψηφιακό μετασχηματισμό σε όλους ανεξαιρέτως τους κλάδους της οικονομίας. Ένας από τους τομείς που βιώνει την πιο έντονη και ταχεία τεχνολογική μετάβαση είναι η βιομηχανία της εστίασης και της φιλοξενίας. Η παραδοσιακή παραγγελιοληψία, η οποία επί δεκαετίες βασιζόταν αποκλειστικά στη χρήση χαρτιού και στυλό, θεωρείται πλέον παρωχημένη, καθώς αδυνατεί να ανταποκριθεί στις σύγχρονες απαιτήσεις για ταχύτητα εξυπηρέτησης, ακρίβεια στην καταγραφή και ελαχιστοποίηση του ανθρώπινου σφάλματος.

Σήμερα, οι επιχειρήσεις εστίασης κάθε μεγέθους, από τα μικρά καφέ της γειτονιάς μέχρι τις μεγάλες αλυσίδες εστιατορίων, στηρίζονται ολοένα και περισσότερο σε ψηφιακά συστήματα σημείου πώλησης (POS ή PDA). Τα συστήματα αυτά αναλαμβάνουν να αυτοματοποιήσουν ολόκληρη τη ροή της πληροφορίας μέσα στην επιχείρηση, μειώνοντας δραματικά τον χρόνο που μεσολαβεί από τη στιγμή που ένας πελάτης κάνει την παραγγελία του μέχρι τη στιγμή που το τελικό προϊόν φτάνει στο τραπέζι του. Ταυτόχρονα, δίνουν τη δυνατότητα στον ιδιοκτήτη να παρακολουθεί σε πραγματικό χρόνο τη λειτουργία του καταστήματος, αποκτώντας πλήρη εικόνα για τα αποθέματα, τις πωλήσεις και την απόδοση του προσωπικού.

Έχοντας αυτά τα δεδομένα, η παρούσα πτυχιακή εργασία παρουσιάζει τον σχεδιασμό και την ανάπτυξη του OrderFlow. Πρόκειται για ένα ολοκληρωμένο, σύγχρονο σύστημα διαχείρισης παραγγελιών, το οποίο δεν περιορίζεται σε μια απλή εφαρμογή καταχώρησης, αλλά προσφέρει μια συνολική λύση: ένα πλήρες οικοσύστημα κατανεμημένο σε τρεις εξειδικευμένες εφαρμογές, απόλυτα προσαρμοσμένες στις ανάγκες και τους ρόλους του κάθε μέλους του προσωπικού μιας επιχείρησης εστίασης.

1.2 Πλαίσιο και Κίνητρο

Το κίνητρο για την υλοποίηση του συγκεκριμένου συστήματος πηγάζει από την άμεση παρατήρηση των πρακτικών προβλημάτων, που αντιμετωπίζουν καθημερινά οι εργαζόμενοι στον κλάδο της εστίασης. Παρότι η αγορά λογισμικού προσφέρει ποικίλες λύσεις, συχνά αυτές πάσχουν από σημαντικά, σχεδιαστικά ελαττώματα, τα οποία δυσχεραίνουν τη χρήση τους. Πολλές φορές ο κατάλογος περιέχει πάρα πολλούς κωδικούς και αν οι πελάτες επιθυμούν προϊόντα από διαφορετικές κατηγορίες η λήψη της παραγγελίας γίνεται πολύ χρονοβόρα.

Ένα από τα συχνότερα προβλήματα είναι η μονολιθική προσέγγιση όπου μια ενιαία, εξαιρετικά πολύπλοκη εφαρμογή, προσπαθεί να εξυπηρετήσει ταυτόχρονα τον σερβιτόρο, τον μάγειρα, τον μπάρμαν και τον διαχειριστή. Αυτό έχει ως αποτέλεσμα τη δημιουργία δύσχρηστων διεπαφών χρήστη, υπερβολικά πολλά κουμπιά ή μενού και πληροφορίες που δεν αφορούν τον εκάστοτε υπάλληλο. Ένας σερβιτόρος ενώ εξυπηρετεί, δεν έχει τον χρόνο να πλοηγηθεί σε πολύπλοκα, πολυεπίπεδα μενού για να εντοπίσει ένα προϊόν. Ούτε ένας μάγειρας, με βρεγμένα ή λερωμένα χέρια, μπορεί να αλληλεπιδρά με μια οθόνη αφής συνέχεια. Η πληροφοριακή υπερφόρτωση μπαίνει εμπόδιο στην ταχύτητα, προκαλεί λάθη και τελικά κοστίζει σε χρόνο και χρήμα στην επιχείρηση.

Βασιζόμενο στις παραπάνω παρατηρήσεις, το κίνητρο της εργασίας είναι η δημιουργία ενός συστήματος που τοποθετεί τον υπάλληλο στο επίκεντρο. Η κεντρική σχεδιαστική φιλοσοφία του OrderFlow είναι η διάσπαση της πολυπλοκότητας: αντί για μία περίπλοκη εφαρμογή αναπτύσσονται

τρεις εύχρηστες εφαρμογές, εξειδικευμένες σε κάθε ρόλο. Η προσέγγιση αυτή μειώνει κάθετα τον χρόνο εκπαίδευσης του νέου προσωπικού, ελαχιστοποιεί τα ανθρώπινα λάθη (τα οποία σε ένα εστιατόριο μεταφράζονται σε λάθος παραγγελίες, χαμένα έσοδα και δυσαρέσκεια πελατών) και επιτρέπει στο προσωπικό να εστιάσει στην εξυπηρέτηση αντί στην ψηφιακή γραφειοκρατία.

1.3 Σκοπός και Στόχοι της Πτυχιακής Εργασίας

Ο κύριος σκοπός της παρούσας πτυχιακής εργασίας είναι η μελέτη, ο σχεδιασμός και η πρακτική υλοποίηση (full-stack development) ενός πλήρους συστήματος παραγγελιοληψίας εστίασης. Για την επίτευξη αυτού του σκοπού, τέθηκαν οι παρακάτω στόχοι:

- Πρώτος στόχος είναι η ανάπτυξη εξειδικευμένων διεπαφών χρήστη. Σχεδιάστηκαν και υλοποιήθηκαν τρεις ξεχωριστές εφαρμογές (Σερβιτόρου, Κουζίνας, Διαχειριστή), καθεμία προσαρμοσμένη στις φυσικές και εργονομικές ανάγκες του αντίστοιχου ρόλου. Η εφαρμογή του σερβιτόρου σχεδιάστηκε για κάθετη χρήση σε κινητό τηλέφωνο (Portrait Mode). Η εφαρμογή της κουζίνας σχεδιάστηκε για οριζόντια χρήση σε tablet (Landscape Mode) και η εφαρμογή του διαχειριστή για χρήση είτε σε tablet είτε σε κινητό.
- Δεύτερος στόχος είναι η δημιουργία ενός μηχανισμού καταχώρησης χειρόγραφων παραγγελιών. Αναπτύχθηκε ένας έξυπνος αλγόριθμος (Αναλυτής συντομογραφιών) ο οποίος επιτρέπει στον σερβιτόρο να καταχωρεί παραγγελίες πληκτρολογώντας σύντομους κωδικούς, αυξάνοντας δραματικά την ταχύτητα εξυπηρέτησης σε σχέση με τη συμβατική πλοήγηση στον κατάλογο.
- Τρίτος στόχος είναι ο συγχρονισμός σε πραγματικό χρόνο. Υλοποιήθηκαν μηχανισμοί (όπως το polling και Optimistic UI Updates) που εξασφαλίζουν ότι τα δεδομένα, οι νέες παραγγελίες και οι ελλείψεις προϊόντων, μεταδίδονται άμεσα χωρίς να απαιτείται ανανέωση ή χειροκίνητη ενέργεια.
- Τέταρτος στόχος είναι η ανάπτυξη κεντρικού συστήματος υποστήριξης (Backend). Δημιουργήθηκε ένας ασφαλής, επεκτάσιμος και κατανεμημένος διακομιστής σε αρχιτεκτονική REST API, ο οποίος διαχειρίζεται την επιχειρησιακή λογική, τον έλεγχο πρόσβασης βάσει ρόλων (RBAC) και την ακεραιότητα των δεδομένων.
- Πέμπτος στόχος είναι η αρχιτεκτονική βελτιστοποίηση κώδικα. Εφαρμόστηκε το μοντέλο Monorepo με κεντρικό κοινόχρηστο πακέτο (order_flow_common) για τη διαχείριση μοντέλων δεδομένων, οπτικής ταυτότητας και επαναχρησιμοποιήσιμων στοιχείων, εξαλείφοντας πλήρως τον διπλό κώδικα μεταξύ των εφαρμογών.

1.4 Παραδοτέα

Ως αποτέλεσμα της υλοποίησης των παραπάνω στόχων, το έργο παραδίδει ένα πλήρες λογισμικό. Τα κύρια παραδοτέα της πτυχιακής εργασίας είναι τα εξής υποσυστήματα:

- Η εφαρμογή order_flow_waiter για κινητά τηλέφωνα, που χρησιμοποιείται αποκλειστικά από τους σερβιτόρους για την καταγραφή παραγγελιών, τη διαχείριση τραπεζιών, την παραμετροποίηση προϊόντων και την έκδοση λογαριασμών.
- Η εφαρμογή order_flow_production για tablet και κινητά. Επιτρέπει στο προσωπικό παραγωγής να λαμβάνει ειδοποιήσεις για νέες παραγγελίες, να επισημαίνει παραγγελίες ως "Σε Εξέλιξη" ή "Ετοιμες" και να ενημερώνει για ελλείψεις αποθεμάτων σε πραγματικό χρόνο.

- Η εφαρμογή `order_flow_admin` για κινητά. Μέσω αυτής, ο ιδιοκτήτης ή ο υπεύθυνος δημιουργεί και διαχειρίζεται τον κατάλογο προϊόντων, διαχειρίζεται τους υπαλλήλους (δημιουργία λογαριασμών, ανάθεση ρόλων, ενεργοποίηση/απενεργοποίηση) και επιβλέπει τη ροή των ενεργών και ολοκληρωμένων παραγγελιών.
- Το πακέτο `order_flow_common` αποτελεί τη κεντρική βιβλιοθήκη κώδικα που τροφοδοτεί τις τρεις παραπάνω εφαρμογές. Περιλαμβάνει τα μοντέλα δεδομένων (built value), τις υπηρεσίες επικοινωνίας με το API (HTTP Services), και τη σχεδιαστική ταυτότητα (Color Palettes, Typography).
- Τέλος, το `order_flow_backend` είναι η κεντρική υποδομή και η σχεσιακή βάση δεδομένων. Ανεπτυγμένο σε Java (Spring Boot), αποτελείται από πολλαπλά υποσυστήματα (modules): το κύριο REST API για την εξυπηρέτηση δεδομένων, έναν ενδιάμεσο κόμβο ασφαλείας (API Gateway) για την πρόσβαση και ένα πακέτο κοινών βοηθητικών εργαλείων (utilities).

1.5 Δομή του Κειμένου

Η παρούσα πτυχιακή εργασία οργανώνεται σε έξι διακριτά κεφάλαια, τα οποία ακολουθούν τη λογική ροή ανάπτυξης ενός έργου λογισμικού, από τη θεωρία στην πράξη:

Το Κεφάλαιο 1 παρουσιάζει το γενικό πλαίσιο, τα κίνητρα, τους στόχους και τα παραδοτέα του συστήματος OrderFlow.

Το Κεφάλαιο 2 παραθέτει το θεωρητικό υπόβαθρο, εξετάζοντας τα σύγχρονα συστήματα POS, αναλύοντας σε βάθος τις τεχνολογίες (Flutter, Dart, Java, Spring Boot, MariaDB) και τα αρχιτεκτονικά πρότυπα (BLoC, REST, ORM) πάνω στα οποία βασίστηκε η υλοποίηση.

Το Κεφάλαιο 3 εστιάζει στην ανάλυση απαιτήσεων του συστήματος, περιγράφοντας αναλυτικά τα σενάρια χρήσης ανά ρόλο, και στον σχεδιασμό της διεπαφής χρήστη (UI/UX Design).

Το Κεφάλαιο 4 αποτελεί τον πυρήνα της υλοποίησης των εφαρμογών, αναλύοντας τον κώδικα, τον αλγόριθμο ανάλυσης συντομογραφιών, τη διαχείριση κατάστασης (State Management), τους μηχανισμούς συγχρονισμού κλπ.

Το Κεφάλαιο 5 περιγράφει την αρχιτεκτονική και την ανάπτυξη του backend, την υλοποίηση του API Gateway, καθώς και τον σχεδιασμό και τη δομή της σχεσιακής βάσης δεδομένων.

Το Κεφάλαιο 6 συνοψίζει τα συμπεράσματα της εργασίας, αξιολογεί το τελικό αποτέλεσμα, καταγράφει τα προβλήματα που αντιμετωπίστηκαν και προτείνει μελλοντικές επεκτάσεις για τη βελτίωση και εξέλιξη του συστήματος.

Κεφάλαιο 2ο: Θεωρητικό Υπόβαθρο και Σχετικές Τεχνολογίες

2.1 Εισαγωγή

Η ανάπτυξη ενός σύγχρονου και ολοκληρωμένου συστήματος διαχείρισης παραγγελιών απαιτεί τη σύνθεση πολλαπλών τεχνολογιών αιχμής, οι οποίες καλούνται να συνεργαστούν αρμονικά. Στο παρόν κεφάλαιο πραγματοποιείται μια εκτενής θεωρητική ανασκόπηση των τεχνολογιών αυτών. Εξετάζεται αρχικά το θεωρητικό πλαίσιο των συστημάτων σημείου πώλησης (POS). Στη συνέχεια, αναλύονται σε βάθος οι μηχανισμοί των σύγχρονων framework ανάπτυξης διεπαφών χρήστη, με έμφαση στο Flutter και τη γλώσσα Dart, καθώς και τα αρχιτεκτονικά πρότυπα διαχείρισης κατάστασης (BLoC Pattern). Ακολούθως, αποδομείται η αρχιτεκτονική του κεντρικού διακομιστή (Backend), εξετάζοντας τη γλώσσα Java, το πλαίσιο Spring Boot και τις αρχές του REST. Τέλος, παρουσιάζεται η θεωρία των σχεσιακών βάσεων δεδομένων και περιγράφεται ο πλήρης κύκλος ζωής μιας πληροφορίας μέσα στο οικοσύστημα του OrderFlow. Στόχος αυτού του κεφαλαίου είναι να παρέχει στον αναγνώστη τα απαραίτητα θεωρητικά θεμέλια για να κατανοήσει πλήρως τις αρχιτεκτονικές αποφάσεις που οδήγησαν στην υλοποίηση του συστήματος στα επόμενα κεφάλαια.

2.2 Συστήματα Διαχείρισης Παραγγελιών

Τα συστήματα σημείου πώλησης (POS) αποτελούν αναπόσπαστο κομμάτι της εμπορικής και επιχειρησιακής λειτουργίας κάθε σύγχρονης επιχείρησης στον κλάδο της λιανικής και, ειδικότερα, της εστίασης. Ιστορικά, τα πρώτα συστήματα POS ήταν οι απλές μηχανικές ταμειακές μηχανές, οι οποίες είχαν ως αποκλειστικό σκοπό την έκδοση αποδείξεων και τη φύλαξη των μετρητών. Η ψηφιακή εξέλιξη ωστόσο μετέτρεψε αυτές τις απλές μηχανές, σε σύνθετα συστήματα, ικανά να διαχειρίζονται αποθέματα, να αναλύουν καταναλωτικά πρότυπα και να παρακολουθούν σε πραγματικό χρόνο την απόδοση του προσωπικού.

Σήμερα, τα Συστήματα Διαχείρισης Παραγγελιών (Order Management Systems) κατηγοριοποιούνται σε δύο μεγάλες αρχιτεκτονικές οικογένειες:

Η πρώτη κατηγορία είναι τα συστήματα τοπικής εγκατάστασης (On-Premise). Σε αυτή την αρχιτεκτονική, ο διακομιστής βρίσκεται φυσικά μέσα στον χώρο του καταστήματος. Τα δεδομένα δεν εγκαταλείπουν ποτέ το τοπικό δίκτυο, γεγονός που προσφέρει υψηλή ταχύτητα (καθώς δεν εξαρτώνται από την ταχύτητα του διαδικτύου) και αυξημένη ασφάλεια. Το μειονέκτημα αυτής της προσέγγισης είναι το υψηλό αρχικό κόστος εξοπλισμού και η ανάγκη για τεχνική συντήρηση του διακομιστή στον χώρο.

Η δεύτερη κατηγορία είναι τα συστήματα νέφους (Cloud-Based). Εδώ, τα δεδομένα φιλοξενούνται σε απομακρυσμένους διακομιστές μέσω υπηρεσιών SaaS (Software as a Service). Το πλεονέκτημα είναι η πρόσβαση από οπουδήποτε, ακόμα και μέσω του προσωπικού κινητού τηλεφώνου του ιδιοκτήτη, καθώς και το πολύ χαμηλό κόστος αρχικής εγκατάστασης. Ωστόσο, η λειτουργικότητα του συστήματος εξαρτάται πλήρως από τη σταθερότητα της σύνδεσης στο διαδίκτυο, κάτι που στον χώρο της εστίασης δεν μπορεί πάντα να θεωρηθεί δεδομένο.

Ένα κρίσιμο υποσύστημα που εμφανίστηκε τα τελευταία χρόνια στον κλάδο της εστίασης είναι το σύστημα ψηφιακής προβολής παραγγελιών στην κουζίνα. Παραδοσιακά, μόλις ο σερβιτόρος καταχωρούσε μια παραγγελία, αυτή εκτυπωνόταν σε ένα κομμάτι θερμικού χαρτιού στον εκτυπωτή της κουζίνας (σαν ticket). Η μέθοδος αυτή, αν και λειτουργική, είναι αναξιόπιστη: τα χαρτιά χάνονται,

καταστρέφονται από υγρασία ή λάδια, μπερδεύεται η σειρά προτεραιότητας τους, ενώ είναι αδύνατη η αναδρομική ανάκτηση ή αναζήτηση μιας παλαιότερης παραγγελίας. Το ψηφιακό σύστημα αντικαθιστά αυτή τη διαδικασία, προβάλλοντας τις παραγγελίες ψηφιακά σε μεγάλες οθόνες, επιτρέποντας τον ακριβή έλεγχο του χρόνου παρασκευής, τη δυναμική αναδιάταξη προτεραιοτήτων και τη χρωματική σήμανση για παραγγελίες που καθυστερούν. Το σύστημα OrderFlow υιοθετεί πλήρως αυτή τη σύγχρονη φιλοσοφία, ψηφιοποιώντας κάθε βήμα της διαδικασίας παραγγελιοληψίας.

2.3 Τεχνολογίες Ανάπτυξης Διεπαφών Χρήστη

Η επιλογή του κατάλληλου πλαισίου ανάπτυξης (framework) για τις frontend εφαρμογές αποτελεί μια από τις πλέον κρίσιμες αποφάσεις στη μηχανική λογισμικού, καθώς καθορίζει τόσο την εμπειρία χρήστη (UX) όσο και το μακροπρόθεσμο κόστος συντήρησης. Στον τομέα της ανάπτυξης εφαρμογών για κινητές συσκευές, οι διαθέσιμες προσεγγίσεις χωρίζονται ιστορικά σε τρεις κατηγορίες:

Η πρώτη είναι το native development, η οποία απαιτεί τη χρήση διαφορετικών γλωσσών προγραμματισμού για κάθε πλατφόρμα (Swift ή Objective-C για iOS, Kotlin ή Java για Android). Προσφέρει τη μέγιστη δυνατή απόδοση και πλήρη πρόσβαση στις δυνατότητες της συσκευής, αλλά διπλασιάζει ουσιαστικά το κόστος ανάπτυξης και συντήρησης, καθώς απαιτούνται δύο εντελώς ξεχωριστές ομάδες προγραμματιστών και δύο βάσεις κώδικα.

Η δεύτερη είναι η υβριδική ανάπτυξη. Τεχνολογίες όπως το Ionic χρησιμοποιούν τεχνολογίες διαδικτύου (HTML, CSS, JavaScript), οι οποίες εκτελούνται μέσα σε έναν ενσωματωμένο περιηγητή (WebView) στη συσκευή. Η προσέγγιση αυτή μειώνει τον χρόνο ανάπτυξης (μία βάση κώδικα για όλες τις πλατφόρμες), αλλά υποφέρει από σοβαρά προβλήματα απόδοσης και αυξημένη κατανάλωση μπαταρίας.

Η τρίτη κατηγορία, στην οποία ανήκει και η τεχνολογία που επέλεξα, είναι το Cross-Platform Development. Εδώ ανήκουν frameworks όπως το React Native και η Flutter, τα οποία επιχειρούν να ισορροπήσουν μεταξύ απόδοσης και ταχύτητας ανάπτυξης.

2.3.1 Η Πλατφόρμα Flutter και η Αρχιτεκτονική Απόδοσης Γραφικών

Για το σύστημα OrderFlow επέλεξα τη Flutter της Google. Η επιλογή αυτή βασίστηκε στη ριζοσπαστική αρχιτεκτονική του Flutter, η οποία διαφέρει θεμελιωδώς από τις ανταγωνιστικές πλατφόρμες. Σε αντίθεση με το React Native, το οποίο χρησιμοποιεί JavaScript Bridge για να μεταφράζει τις εντολές JavaScript στα widgets του λειτουργικού συστήματος, δημιουργώντας καθυστερήσεις στην επικοινωνία, η Flutter ακολουθεί μια εντελώς διαφορετική προσέγγιση. Δεν χρησιμοποιεί καθόλου τα native widgets του Android ή του iOS. Αντιθέτως, διαθέτει τη δική του μηχανή γραφικών, γραμμένη σε C++, η οποία ζωγραφίζει (render) το κάθε pixel της οθόνης ανεξάρτητα. Αυτό εγγυάται πλήρη οπτική ομοιομορφία: η εφαρμογή θα φαίνεται και θα συμπεριφέρεται ακριβώς το ίδιο, είτε εκτελείται σε ένα παλαιότερο Android tablet στην κουζίνα, είτε σε ένα σύγχρονο iPhone στα χέρια του διαχειριστή.

2.3.2 Η Γλώσσα Προγραμματισμού Dart

Η Flutter είναι άρρηκτα συνδεδεμένη με τη γλώσσα προγραμματισμού Dart, η οποία σχεδιάστηκε εξ αρχής από τη Google με γνώμονα τις ανάγκες ανάπτυξης εφαρμογών. Η Dart διαθέτει ένα μοναδικό χαρακτηριστικό που την καθιστά ιδανική για αυτόν τον σκοπό: υποστηρίζει δύο εντελώς διαφορετικούς τρόπους μεταγλώττισης (Compilation).

Κατά τη διάρκεια της ανάπτυξης (Development phase), η Dart χρησιμοποιεί JIT (Just-In-Time) compilation. Αυτό σημαίνει ότι ο κώδικας μεταγλωττίζεται τη στιγμή της εκτέλεσης, παρέχοντας τη λειτουργία Hot Reload. Με το Hot Reload, οι αλλαγές που εφαρμόζει ο προγραμματιστής στον κώδικα εμφανίζονται στην οθόνη της συσκευής σε δευτερόλεπτα, χωρίς να χαθεί η τρέχουσα κατάσταση (state) της εφαρμογής. Αυτό μειώνει δραματικά τον χρόνο ανάπτυξης και δοκιμής.

Αντιθέτως, όταν η εφαρμογή ετοιμάζεται για τελική διανομή (Production release), η Dart μεταγλωττίζεται με AOT (Ahead-Of-Time) compilation.

2.3.3 Διαχείριση Κατάστασης: Το Αρχιτεκτονικό Πρότυπο BLoC

Σε σύνθετες εφαρμογές, η διαχείριση της πληροφορίας (κατάστασης) που μοιράζεται ανάμεσα σε πολλές οθόνες αποτελεί το μεγαλύτερο αρχιτεκτονικό πρόβλημα. Χωρίς μια σοβαρή στρατηγική, ο κώδικας οδηγείται αναπόφευκτα σε αυτό που η βιομηχανία λογισμικού αποκαλεί "Spaghetti Code". Για την αποφυγή αυτού του κινδύνου, το OrderFlow υιοθετεί πλήρως το αρχιτεκτονικό πρότυπο BLoC (Business Logic Component).

Το BLoC βασίζεται στον αντιδραστικό προγραμματισμό. Αντί η διεπαφή χρήστη (UI) να ελέγχει απευθείας τα δεδομένα με διαδοχικές εντολές, το UI απλώς "ακούει" ένα stream. Όταν ο χρήστης εκτελεί μια ενέργεια (π.χ. πατάει μια επιλογή καφέ), η διεπαφή εκπέμπει ένα συμβάν στο BLoC. Το BLoC, ως αποκλειστικός υπεύθυνος της επιχειρησιακής λογικής, εκτελεί τους απαραίτητους υπολογισμούς (έλεγχος διαθεσιμότητας, υπολογισμός τιμής) και εκπέμπει (emit) μια νέα, αμετάβλητη κατάσταση (State). Η διεπαφή, λαμβάνοντας τη νέα κατάσταση, επανασχεδιάζει αυτόματα μόνο τα τμήματα της οθόνης που επηρεάστηκαν. Αυτός ο διαχωρισμός εγγυάται ότι η διεπαφή δεν περιέχει καμία λογική, ενώ η λογική δεν γνωρίζει τίποτα για τη διεπαφή, καθιστώντας τον κώδικα εξαιρετικά συντηρήσιμο.

2.4 Τεχνολογίες Υποδομής

Ενώ οι frontend εφαρμογές εξασφαλίζουν την αλληλεπίδραση με τον χρήστη, το backend αποτελεί τον "εγκέφαλο" του συστήματος. Είναι υπεύθυνο για τη συνοχή και την ασφάλεια των δεδομένων, τον έλεγχο ταυτότητας (Authentication), την επιβολή των επιχειρηματικών κανόνων και τον συγχρονισμό όλων των συσκευών, ακόμα και όταν δέχεται πολλαπλά ταυτόχρονα αιτήματα.

2.4.1 Η Γλώσσα Java (Έκδοση 21)

Η επιλογή της Java ως γλώσσας ανάπτυξης του Backend δεν ήταν τυχαία. Παρά την εμφάνιση νεότερων γλωσσών, η Java παραμένει ο αδιαφιλονίκητος ηγέτης στον χώρο των εταιρικών εφαρμογών (Enterprise Systems). Ένας βασικός λόγος είναι η Εικονική Μηχανή (JVM, Java Virtual Machine), η οποία εκτελεί τον κώδικα (Bytecode) ανεξαρτήτως του λειτουργικού συστήματος (Linux, Windows, macOS), υλοποιώντας την αρχή "Write Once, Run Anywhere".

Ο ισχυρός στατικός έλεγχος τύπων της Java σημαίνει ότι μια μεγάλη κατηγορία κρίσιμων σφαλμάτων εντοπίζεται κατά τον χρόνο μεταγλώττισης (compile time) και όχι κατά την εκτέλεση (runtime), όταν πλέον θα ήταν πολύ αργά. Επίσης η Java ενσωματώνει αυτόματη διαχείριση μνήμης μέσω του garbage collector. Σε αντίθεση με γλώσσες όπως η C++, όπου η χειροκίνητη δέσμευση και αποδέσμευση μνήμης αποτελεί πηγή σοβαρότατων σφαλμάτων (memory leaks), ο garbage collector σαρώνει διαρκώς τη μνήμη, εντοπίζοντας και απελευθερώνοντας αντικείμενα που δεν χρησιμοποιούνται πλέον, χωρίς παρέμβαση του προγραμματιστή.

Η έκδοση 21 της Java, η οποία χρησιμοποιείται στο OrderFlow, αποτελεί έκδοση μακροπρόθεσμης υποστήριξης (Long-Term Support, LTS), εξασφαλίζοντας ενημερώσεις ασφαλείας για αρκετά χρόνια. Εισάγει επίσης σύγχρονες βελτιστοποιήσεις, όπως τα Virtual Threads, τα οποία επιτρέπουν στον διακομιστή να διαχειρίζεται χιλιάδες ταυτόχρονες συνδέσεις με ελάχιστη κατανάλωση μνήμης.

2.4.2 Το Πλαίσιο Spring Boot 3

Από μόνη της η Java παρέχει τα εργαλεία, αλλά η κατασκευή ενός σύγχρονου διακομιστή REST από το μηδέν θα ήταν εξαιρετικά χρονοβόρα. Εδώ έρχεται το framework Spring Boot να δώσει την λύση. Ιστορικά, η ρύθμιση (configuration) μιας εφαρμογής Spring απαιτούσε εκατοντάδες γραμμές αρχείων XML. Το Spring Boot δημιουργήθηκε ακριβώς για να εξαλείψει αυτή την πολυπλοκότητα, εισάγοντας τη φιλοσοφία convention over configuration.

Ο πυρήνας του Spring βασίζεται σε δύο θεμελιώδεις αρχές: το Inversion of Control (IoC, Αντιστροφή Ελέγχου) και το Dependency Injection (DI, Έγχυση Εξαρτήσεων). Σε ένα παραδοσιακό πρόγραμμα, μια κλάση δημιουργεί μόνη της τα αντικείμενα που χρειάζεται μέσω της λέξης-κλειδί "new". Με το IoC, ο έλεγχος αντιστρέφεται: το ίδιο το πλαίσιο (Spring Container) δημιουργεί τα αντικείμενα και τα κάνει inject στην κλάση τη στιγμή που τα χρειάζεται. Αυτό επιτυγχάνει χαλαρή σύζευξη (loose coupling) μεταξύ των εξαρτημάτων, κάνοντας τον κώδικα modular και εύκολο στη δοκιμή (Unit Testing).

Το Spring Boot παρέχει, επιπλέον, ενσωματωμένο web server (Apache Tomcat), ο οποίος ξεκινά αυτόματα χωρίς εξωτερική εγκατάσταση. Μέσω του μηχανισμού Auto-Configuration, το Spring Boot αναγνωρίζει αυτόματα τις βιβλιοθήκες που υπάρχουν στο project (π.χ. αν εντοπίσει βιβλιοθήκη οδηγού MariaDB, ρυθμίζει αυτόματα τη σύνδεση με τη βάση δεδομένων) και εφαρμόζει τις κατάλληλες ρυθμίσεις, εξοικονομώντας στον προγραμματιστή ημέρες παραμετροποίησης.

2.4.3 Η Αρχιτεκτονική REST (Representational State Transfer)

Η αμφίδρομη επικοινωνία μεταξύ των εφαρμογών Flutter (κινητά σερβιτόρων, tablet κουζίνας) και του κεντρικού διακομιστή Spring Boot υλοποιείται μέσω διεπαφής RESTful API. Το REST δεν αποτελεί πρωτόκολλο ούτε συγκεκριμένο λογισμικό. Πρόκειται για ένα σύνολο αρχιτεκτονικών αρχών, οι οποίες εισήχθησαν από τον Roy Fielding στη διδακτορική του διατριβή το 2000 και αποτελούν σήμερα τον de facto κανόνα για τις σύγχρονες υπηρεσίες ιστού.

Ένα RESTful API βασίζεται σε αυστηρούς αρχιτεκτονικούς περιορισμούς. Ο σημαντικότερος από αυτούς είναι η Απουσία Κατάστασης (Statelessness): ο διακομιστής δεν αποθηκεύει κανένα δεδομένο σχετικό με τη συνεδρία (session) του πελάτη μεταξύ δύο αιτημάτων. Κάθε HTTP Request που στέλνει η συσκευή του σερβιτόρου ή το tablet της κουζίνας πρέπει να είναι πλήρως αυτοδύναμο, περιέχοντας όλες τις απαραίτητες πληροφορίες για να εξυπηρετηθεί (π.χ. τα credentials αυθεντικοποίησης). Άλλοι βασικοί περιορισμοί περιλαμβάνουν τον απόλυτο διαχωρισμό Πελάτη-Διακομιστή (Client-Server separation), τη δυνατότητα προσωρινής αποθήκευσης (Cacheable responses) και την ομοιόμορφη διεπαφή (Uniform Interface) μέσω σταθερών URIs και τυποποιημένων μεθόδων HTTP.

Η ανταλλαγή δεδομένων μεταξύ πελάτη και διακομιστή γίνεται αποκλειστικά σε μορφή JSON (JavaScript Object Notation). Η JSON είναι μια ελαφριά, αναγνώσιμη από τον άνθρωπο δομή δεδομένων, η οποία αποτελείται από ζεύγη κλειδιού-τιμής. Η σειριοποίηση (serialization) και αποσειριοποίηση (deserialization) των δεδομένων γίνεται στην πλευρά της Java μέσω της βιβλιοθήκης Gson, ενώ στην πλευρά της Dart μέσω του πακέτου built_value.

2.5 Βάση Δεδομένων

Σε ένα σύστημα εστίασης, η ορθή και ασφαλής αποθήκευση δεδομένων δεν αποτελεί απλώς τεχνική απαίτηση, αλλά θεμελιώδη επιχειρηματική αναγκαιότητα. Η απώλεια ή η αλλοίωση μιας παραγγελίας μεταφράζεται αυτόματα σε οικονομική ζημία και δυσaréσκεια πελατών.

2.5.1 Σχεσιακό Μοντέλο και Ιδιότητες ACID

Παρά την αυξανόμενη δημοτικότητα των βάσεων δεδομένων NoSQL (όπως η MongoDB) σε ορισμένους τομείς, η βιομηχανία της εστίασης και γενικότερα οι εφαρμογές με οικονομικές συναλλαγές απαιτούν Σχεσιακά Συστήματα Διαχείρισης Βάσεων Δεδομένων (RDBMS). Ο κύριος λόγος είναι οι ιδιότητες ACID (Atomicity, Consistency, Isolation, Durability), τις οποίες εγγυώνται τα σχεσιακά συστήματα:

Η Ατομικότητα (Atomicity) σημαίνει ότι μια συναλλαγή (transaction) είτε εκτελείται ολόκληρη με επιτυχία, είτε ακυρώνεται εντελώς. Αν κατά την καταχώρηση μιας παραγγελίας 5 προϊόντων πέσει το ρεύμα μετά το τρίτο, δεν θα αποθηκευτεί "μισή" παραγγελία. Η Συνοχή (Consistency) εξασφαλίζει ότι η βάση μεταβαίνει πάντα από μια έγκυρη κατάσταση σε μια άλλη έγκυρη κατάσταση. Η Απομόνωση (Isolation) εγγυάται ότι ταυτόχρονες συναλλαγές δεν αλληλοεπηρεάζονται. Η Ανθεκτικότητα (Durability) εξασφαλίζει ότι μόλις μια συναλλαγή επιβεβαιωθεί (committed), τα δεδομένα παραμένουν αποθηκευμένα ακόμα και σε περίπτωση αστοχίας του υλικού.

Για τον σκοπό αυτό επελέγη η MariaDB, μια ανοιχτού κώδικα, υψηλής απόδοσης σχεσιακή βάση δεδομένων. Η MariaDB δημιουργήθηκε ως fork της MySQL από τον αρχικό δημιουργό της MySQL, Michael "Monty" Widenius, και θεωρείται πλήρως συμβατή με αυτήν, προσφέροντας παράλληλα βελτιώσεις στον μηχανισμό αποθήκευσης (InnoDB storage engine) και στην απόδοση ερωτημάτων.

2.5.2 Τεχνολογία ORM (Object-Relational Mapping)

Η διασύνδεση της Java με την βάση δεδομένων MariaDB υλοποιείται με την τεχνολογία ORM (Object-Relational Mapping). Στο OrderFlow, χρησιμοποιείται το πρότυπο JPA (Jakarta Persistence API) σε συνδυασμό με την υλοποίησή του από το Hibernate.

Το ORM αναλαμβάνει να αντιστοιχίσει αυτόματα τις κλάσεις της Java σε πίνακες (tables) της βάσης δεδομένων, τις μεταβλητές σε στήλες (columns) και τα αντικείμενα σε εγγραφές (rows). Ο προγραμματιστής δεν χρειάζεται να γράφει απευθείας εντολές SQL (όπως INSERT INTO ή SELECT). Απλώς, χειρίζεται τα δεδομένα ως αντικείμενα κώδικα, καλώντας μεθόδους (π.χ. repository.save(order)). Το Spring Data JPA μεταφράζει αυτόματα τις κλήσεις αυτές σε

βελτιστοποιημένα ερωτήματα SQL, αποτρέποντας ταυτόχρονα επιθέσεις τύπου SQL Injection, καθώς τα δεδομένα περνούν πάντα μέσω παραμετρικών ερωτημάτων (parameterized queries).

2.6 Ο κύκλος ζωής των δεδομένων

Για την ολοκληρωμένη κατανόηση του τρόπου συνεργασίας όλων των παραπάνω τεχνολογιών, είναι σημαντικό να γίνει κατανοητή η πλήρης διαδρομή μιας πληροφορίας μέσα στο οικοσύστημα του OrderFlow, από τη στιγμή που ο σερβιτόρος αγγίζει την οθόνη, μέχρι τη στιγμή που η παραγγελία εμφανίζεται στο tablet της κουζίνας.

Το πρώτο βήμα είναι η δημιουργία στο επίπεδο του UI. Ο σερβιτόρος πατάει την οθόνη αφής του κινητού για να προσθέσει ένα προϊόν. Το Flutter Widget στέλνει ένα Event στο αρμόδιο Cubit (BLoC),

Εισαγωγή

το οποίο δημιουργεί στη μνήμη (RAM) της συσκευής το αντικείμενο της παραγγελίας, ως ένα αμετάβλητο (immutable) αντικείμενο Dart μέσω του `built_value`.

Το δεύτερο βήμα είναι το `serialization`. Τα δεδομένα του αντικειμένου μετατρέπονται σε κείμενο μορφής JSON για μεταφορά μέσω δικτύου.

Το τρίτο βήμα είναι η μετάδοση μέσω δικτύου. Μέσω της βιβλιοθήκης Dio, αποστέλλεται ένα αίτημα HTTP POST μέσω του τοπικού ασύρματου δικτύου (Wi-Fi) του καταστήματος, μεταφέροντας το JSON ως `body` του αιτήματος.

Στο τέταρτο βήμα, η υποδοχή, ο ενσωματωμένος διακομιστής Apache Tomcat του Spring Boot, ο οποίος ακούει στη θύρα 8080, παραλαμβάνει το αίτημα. Ο `controller` αποκωδικοποιεί το JSON μέσω της βιβλιοθήκης Gson, μετατρέποντάς το πίσω σε αντικείμενο Java.

Το πέμπτο βήμα είναι η αποθήκευση. Το αντικείμενο προωθείται στο Service Layer, το οποίο κάνει και `validation` και στη συνέχεια καλεί το Spring Data JPA Repository. Το Hibernate παράγει αυτόματα τις εντολές INSERT INTO και η MariaDB τις εκτελεί, αποθηκεύοντας οριστικά τα δεδομένα στην βάση.

Στο έκτο και τελευταίο βήμα, την κατανάλωση, η εφαρμογή της κουζίνας εκτελεί τον δικό της κύκλο. Μέσω του μηχανισμού `polling`, ενημερώνεται για την νέα εγγραφή στη βάση δεδομένων. Ο διακομιστής απαντά με τη νέα παραγγελία σε μορφή JSON. Η Dart αποκωδικοποιεί τα δεδομένα, το Cubit εκπέμπει μια νέα κατάσταση, και η οθόνη του tablet ανανεώνεται αυτόματα, εμφανίζοντας τη νέα παραγγελία στον υπάλληλο.

Αυτός ο σύνθετος κύκλος εκτελείται σε χιλιοστά του δευτερολέπτου, επιτρέποντας στην επιχείρηση να λειτουργεί ομαλά, γρήγορα και με απόλυτη ακρίβεια ακόμα και στις πιο πιεστικές ώρες αιχμής.

Κεφάλαιο 3ο: Ανάλυση Απαιτήσεων και Σχεδίαση Συστήματος

3.1 Εισαγωγή

Στα προηγούμενα κεφάλαια εξετάστηκε το θεωρητικό και τεχνολογικό υπόβαθρο, πάνω στο οποίο βασίζεται η σχεδίαση του συστήματος. Στο παρόν κεφάλαιο, θα εστιάσουμε στο πρακτικό κομμάτι. Θα αναλυθεί λεπτομερώς ο λειτουργικός και οπτικός σχεδιασμός του συστήματος παραγγελιοληψίας Order Flow. Η μετάβαση από τη σύλληψη της αρχικής ιδέας στην τελική υλοποίηση προϋποθέτει μια αυστηρή μεθοδολογία σχεδιασμού, προκειμένου να αποφευχθούν προγραμματιστικά σφάλματα που θα κόστιζαν ακριβά σε χρόνο και πόρους.

Ο χώρος της εστίασης αποτελεί ένα από τα πλέον απαιτητικά εργασιακά περιβάλλοντα. Χαρακτηρίζεται από έντονη πίεση χρόνου, υψηλό φόρτο εργασίας, θόρυβο και μηδενική ανοχή στην καθυστέρηση. Με βάση αυτά τα δεδομένα, ο σχεδιασμός του συστήματος απαιτεί μια πελατοκεντρική προσέγγιση. Πρέπει να μελετηθούν διεξοδικά οι ξεχωριστές ανάγκες του κάθε υπαλλήλου (Σερβιτόρος, Προσωπικό Παραγωγής, Διαχειριστής). Έπειτα πρέπει να δημιουργηθούν τα κατάλληλα εργαλεία για τον καθένα.

Στις υποενότητες που ακολουθούν, καταγράφονται αναλυτικά οι λειτουργικές και μη λειτουργικές απαιτήσεις του συστήματος ανά ρόλο, ενώ στη συνέχεια αναλύεται ο σχεδιασμός της διεπαφής χρήστη (UI/UX). Ο συγκεκριμένος σχεδιασμός επιλέχθηκε για να εξασφαλίσει την γρήγορη και ξεκούραστη χρήση των εφαρμογών από το προσωπικό.

3.2 Ανάλυση Απαιτήσεων και Περιπτώσεις Χρήσης

Ο καθορισμός των απαιτήσεων αποτελεί ένα από τα σημαντικότερα στάδια του κύκλου ζωής ανάπτυξης λογισμικού. Στο πλαίσιο του συστήματος OrderFlow, η ανάλυση απαιτήσεων δεν περιορίστηκε μόνο στη βασική καταγραφή των λειτουργιών (CRUD operations), αλλά έλαβε σοβαρά υπόψη τις ιδιαιτερότητες, την πίεση χρόνου και την ποικιλομορφία στον τρόπο λειτουργίας ενός καταστήματος εστίασης.

Λόγω της επιλογής να δημιουργηθούν τρεις διακριτές εφαρμογές, η ανάλυση των απαιτήσεων κατηγοριοποιήθηκε αυστηρά με βάση τον ρόλο του εκάστοτε τελικού χρήστη. Με αυτόν τον τρόπο, διασφαλίζεται ότι κάθε υποσύστημα θα παρέχει αποκλειστικά και μόνο τα εργαλεία που χρειάζεται ο συγκεκριμένος υπάλληλος. Επιπλέον αποφεύγουμε την οπτική υπερφόρτωση της διεπαφής και αποκλείουμε την πιθανότητα ένας χρήστης να αποκτήσει πρόσβαση σε δεδομένα εκτός της δικαιοδοσίας του.

3.2.1 Απαιτήσεις Διαχειριστή

Ο ρόλος του διαχειριστή (ιδιοκτήτης ή υπεύθυνος βάρδιας) απαιτεί πλήρη εποπτεία του καταστήματος και πρόσβαση στα στοιχεία σύνδεσης των χρηστών. Είναι επίσης υπεύθυνος για την αρχική καταχώρηση του καταλόγου. Αργότερα καλείται να διαχειριστεί τον κατάλογο και τη διαθεσιμότητα των προϊόντων.

Λειτουργικές Απαιτήσεις:

- Διαχείριση Προσωπικού: Το σύστημα πρέπει να επιτρέπει στον διαχειριστή να προσθέτει νέους υπαλλήλους στο σύστημα, να τους αναθέτει συγκεκριμένους ρόλους (Σερβιτόρος, Παραγωγή, Διαχειριστής) και να λαμβάνει τα μοναδικά διαπιστευτήριά τους (Username/Password). Πρέπει επίσης να υποστηρίζεται η προσωρινή απενεργοποίηση λογαριασμών αντί για την οριστική

διαγραφή τους, καθώς στον κλάδο της εστίασης η εποχιακή εργασία είναι σύνηθες φαινόμενο και η διατήρηση του ιστορικού του υπαλλήλου κρίνεται απαραίτητη.

- **Δυναμική Διαχείριση Καταλόγου:** Ο διαχειριστής πρέπει να μπορεί να δημιουργεί, να επεξεργάζεται και να διαγράφει κατηγορίες και προϊόντα. Κρίσιμη απαίτηση είναι η υποστήριξη δυναμικών ομάδων επιλογών (Option Groups) ανά προϊόν. Κάθε προϊόν (π.χ. Καφές) πρέπει να μπορεί να συνοδεύεται από λίστες επιλογών (π.χ. Είδος Ζάχαρης, Είδος Γάλακτος).
- **Διαχείριση Συντομογραφιών:** Κατά τη δημιουργία του καταλόγου, το σύστημα πρέπει να επιτρέπει την αντιστοίχιση μοναδικών συντομογραφιών για κάθε προϊόν και επιλογή, ώστε να τροφοδοτείται σωστά ο αλγόριθμος του αναλυτή συντομογραφιών που θα χρησιμοποιήσουν οι σερβιτόροι.
- **Εποπτεία Παραγγελιών:** Πρέπει να διαθέτει μια συγκεντρωτική οθόνη όπου ο διαχειριστής θα βλέπει σε πραγματικό χρόνο τις ενεργές παραγγελίες και το ιστορικό όσων έχουν ολοκληρωθεί.

Μη Λειτουργικές Απαιτήσεις:

Η διεπαφή πρέπει να είναι βελτιστοποιημένη για οθόνες μεσαίου ή μικρού μεγέθους (Tablet και κινητά), καθώς η διαχείριση ενός πολύπλοκου καταλόγου απαιτεί αρκετό χώρο, όμως η εφαρμογή θα τρέχει σε φορητές συσκευές.

Περίπτωση Χρήσης 1 (Use Case): Δημιουργία Νέου Προϊόντος

Ο διαχειριστής επιλέγει την προσθήκη νέου προϊόντος.

Το σύστημα ζητάει το όνομα, την τιμή, την κατηγορία και τη συντομογραφία (π.χ. "FRED").

Ο διαχειριστής προσθέτει μια προαιρετική Ομάδα Επιλογών (π.χ. "Ζάχαρη") με τις επιλογές "Γλυκός (GL)", "Μέτριος (ME)", "Σκέτος (SK)".

Το σύστημα επικυρώνει τα δεδομένα και ενημερώνει τη βάση δεδομένων. Ο κατάλογος των σερβιτόρων ενημερώνεται άμεσα.

3.2.2 Απαιτήσεις Σερβιτόρου

Ο σερβιτόρος είναι ο χρήστης που δέχεται τη μεγαλύτερη πίεση χρόνου. Η εφαρμογή του πρέπει να λειτουργεί αξιόπιστα, και να μειώνει τα περιθώρια σφάλματος κατά την καταγραφή της παραγγελίας, ειδικά σε ώρες αιχμής.

Λειτουργικές Απαιτήσεις:

- **Πλοήγηση στον Κατάλογο:** Ο χρήστης πρέπει να μπορεί να πλοηγηθεί γρήγορα στις διαθέσιμες κατηγορίες. Προϊόντα που βρίσκονται σε έλλειψη πρέπει να εμφανίζονται μεν, αλλά να είναι απενεργοποιημένα, ώστε να μην επιτρέπεται η επιλογή τους.
- **Παραμετροποίηση Προϊόντος:** Όταν επιλέγεται ένα προϊόν, το σύστημα πρέπει να εμφανίζει άμεσα τις υποχρεωτικές και προαιρετικές επιλογές, καθώς και πεδίο για ελεύθερο κείμενο (ειδικές παρατηρήσεις πελάτη).
- **Διαχείριση Πληρωμών:** Ο σερβιτόρος πρέπει να έχει πρόσβαση στις ενεργές παραγγελίες ανά τραπέζι, ώστε να μπορεί να εκδώσει τον συνολικό λογαριασμό. Οι παραγγελίες ενός τραπεζιού πρέπει να ομαδοποιούνται κατά την πληρωμή ώστε να βγει συνολικός λογαριασμός.

Μη Λειτουργικές Απαιτήσεις:

Η διεπαφή πρέπει να διαθέτει μεγάλα και ευδιάκριτα πλήκτρα, ώστε να είναι εύκολη η χρήση με το ένα χέρι εν κινήσει, σε οθόνες κινητών.

Περίπτωση Χρήσης 2 (Use Case): Λήψη Παραγγελίας

Ο σερβιτόρος προσεγγίζει το Τραπέζι 4 και ανοίγει την εφαρμογή.

Επιλέγει την κατηγορία "Καφέδες" και το προϊόν "Freddo Espresso".

Επιλέγει μέσα από το αναδυόμενο παράθυρο (Bottom Sheet) "Μέτριος", "Με μαύρη ζάχαρη" και προσθέτει παρατήρηση "Χωρίς καλαμάκι".

Προσθέτει τα είδη στο καλάθι, εισάγει τον αριθμό "4" στο πεδίο του τραπεζιού και πατάει αποστολή.

Η παραγγελία καταγράφεται στη βάση δεδομένων και μεταφέρεται αυτόματα στην οθόνη της κουζίνας.

3.2.3 Απαιτήσεις Προσωπικού Παρασκευής

Το προσωπικό παραγωγής (κουζίνα, bar) λειτουργεί σε ένα περιβάλλον με θόρυβο και έντονη κινητικότητα, όπου η άμεση επικοινωνία με τους σερβιτόρους είναι ζωτικής σημασίας για τη γρήγορη εξυπηρέτηση των πελατών.

Λειτουργικές Απαιτήσεις:

- Αυτόματη λήψη και ειδοποίηση για νέες παραγγελίες: Η εφαρμογή πρέπει να λαμβάνει αυτόματα τις νέες παραγγελίες τη στιγμή ακριβώς που τις καταχωρεί ο σερβιτόρος. Για την αποφυγή απωλειών λόγω θορύβου, απαιτείται η παραγωγή οπτικοακουστικής ειδοποίησης σε κάθε νέα εισερχόμενη παραγγελία.
- Διαχείριση Κύκλου Ζωής Παραγγελίας: Ο υπάλληλος πρέπει να μπορεί να αλλάξει την κατάσταση της παραγγελίας από "Ενεργή" σε "Αναγνωσμένη" και τελικά σε "Ετοιμη", ενημερώνοντας το κεντρικό σύστημα ώστε να την παραλάβει ο σερβιτόρος.
- Διαχείριση Αποθεμάτων σε Πραγματικό Χρόνο: Εάν ένα υλικό τελειώσει κατά τη διάρκεια της βάρδιας, το προσωπικό παραγωγής πρέπει να μπορεί να απενεργοποιήσει το προϊόν επιτόπου από μια ειδική καρτέλα, ώστε να ενημερωθούν άμεσα όλα τα κινητά των σερβιτόρων και να αποφευχθούν άκυρες παραγγελίες.

Μη Λειτουργικές Απαιτήσεις:

- Η εφαρμογή πρέπει να λειτουργεί αδιάλειπτα ως μόνιμη οθόνη (Always-On Display) σε μορφή Tablet, τοποθετημένη σε βάση.
- Οι οπτικές ενδείξεις (χρώματα, μεγέθη γραμματοσειράς) πρέπει να είναι εξαιρετικά έντονες. Οι παραγγελίες πρέπει να είναι ευανάγνωστες από απόσταση τουλάχιστον 1-2 μέτρων, επιτρέποντας στο προσωπικό να διαβάζει τα εισερχόμενα, χωρίς να χρειάζεται να αγγίζει συνεχώς την οθόνη με λερωμένα ή βρεγμένα χέρια.

Περίπτωση Χρήσης 3 (Use Case): Διαχείριση Έλλειψης Προϊόντος (Out of Stock)

Ο Barista διαπιστώνει ότι τελείωσε το φρέσκο γάλα και δεν μπορεί να φτιάξει άλλο Carruccino.

Μεταβαίνει στην καρτέλα "Διαθεσιμότητα" της εφαρμογής Παραγωγής.

Εντοπίζει το προϊόν "Carruccino" και κλείνει τον διακόπτη διαθεσιμότητας.

Το σύστημα ενημερώνει τη βάση δεδομένων μέσω REST API.

Αυτόματα, σε όλα τα κινητά των σερβιτόρων το συγκεκριμένο προϊόν γίνεται γκρι (disabled) και δεν μπορούν πλέον να το προσθέσουν σε νέες παραγγελίες, γλιτώνοντας χρόνο και παρεξηγήσεις με τους πελάτες.

3.3 Σχεδίαση Διεπαφής Χρήστη

Σε ένα περιβάλλον εστίασης, ο χρόνος εκπαίδευσης του νέου προσωπικού είναι ελάχιστος και η πίεση κατά τις ώρες αιχμής μεγάλη. Οπότε, η σχεδιαστική φιλοσοφία του OrderFlow εστίασε στη δημιουργία μιας μινιμαλιστικής, κατανοητής και άκρως λειτουργικής διεπαφής χρήστη, αφαιρώντας κάθε περιττή οπτική πληροφορία. Η διεπαφή προσφέρει ομαλά animations, σαφή οπτική ιεραρχία και σύγχρονη αισθητική.

3.3.1 Οπτική Ταυτότητα και Τυπογραφία

Για τη διασφάλιση της οπτικής συνοχής και στις τρεις εφαρμογές του συστήματος, η τυπογραφία δεν ορίστηκε ξεχωριστά σε κάθε οθόνη, αλλά συγκεντρωτικά μέσω της κοινής βιβλιοθήκης `order_flow_common`. Επιλέχθηκε η γραμματοσειρά Ubuntu (μέσω του πακέτου `google_fonts`), η οποία προσφέρει εξαιρετική αναγνωσιμότητα ακόμα και σε μικρές οθόνες κινητών. Η διανομή των στυλ κειμένου (Text Styles) γίνεται δυναμικά σε ολόκληρο το δέντρο των Widgets μέσω της κλάσης `FontsProvider`.

3.3.2 Εργονομία και Πλοήγηση

Η ροή της πλοήγησης προσαρμόστηκε στις φυσικές ανάγκες του κάθε ρόλου:

- Χρήση με το ένα χέρι: Η εφαρμογή του σερβιτόρου σχεδιάστηκε για κάθετη χρήση (Portrait Mode) σε κινητά τηλέφωνα. Τα κουμπιά ενεργειών (π.χ. προσθήκη στο καλάθι) είναι μεγάλα για να πατιούνται εύκολα με τον αντίχειρα. Για την παραμετροποίηση της παραγγελίας χρησιμοποιούνται Bottom Sheets (αναδυόμενα παράθυρα από το κάτω μέρος της οθόνης).
- Οπτική Διαύγεια: Η εφαρμογή της κουζίνας έχει σχεδιαστεί για χρήση σε Tablet (Landscape Mode). Εδώ χρησιμοποιείται το `DefaultTabController` για γρήγορη, οριζόντια εναλλαγή καρτελών (ενεργές, ιστορικό). Η διεπαφή χρησιμοποιεί το επαναχρησιμοποιήσιμο widget `OrderCard`, το οποίο προβάλλει τα είδη της παραγγελίας με μεγάλα γράμματα και έντονες αντιθέσεις, ώστε ο υπάλληλος να μπορεί να διαβάζει την παραγγελία από απόσταση.
- Έξυπνα Πλήκτρα: Στην εφαρμογή διαχείρισης, ο χώρος της οθόνης εξοικονομείται με τη χρήση ενός δυναμικού Floating Action Button. Αντί να γεμίζει η οθόνη με κουμπιά "Προσθήκη Χρήστη" ή "Προσθήκη Προϊόντος", το σύστημα "διαβάζει" σε ποια καρτέλα βρίσκεται ο διαχειριστής (`state.currentTabIndex`) και προσαρμόζει αυτόματα τη λειτουργία και το εικονίδιο του μοναδικού κουμπιού.

3.3.3 Σύνοψη Κεφαλαίου

Ολοκληρώνοντας την ανάλυση του παρόντος κεφαλαίου, έχει πλέον καθοριστεί πλήρως ο λειτουργικός και σχεδιαστικός σκελετός, πάνω στον οποίο οικοδομήθηκε το σύστημα OrderFlow. Η συλλογή και

καταγραφή των απαιτήσεων ανέδειξε την αναγκαιότητα για τη δημιουργία τριών διακριτών εφαρμογών, οι οποίες εξυπηρετούν στοχευμένα τους διαφορετικούς ρόλους μιας επιχείρησης εστίασης.

Παράλληλα, ο προσεκτικός σχεδιασμός της διεπαφής (UI/UX) έλαβε υπόψη τις ιδιαίτερες συνθήκες εργασίας. Με την προσεκτική επιλογή τυπογραφίας και την εργονομική σχεδίαση των διεπαφών, εξασφαλίστηκε μια σύγχρονη εμπειρία χρήσης, ικανή να ανταπεξέλθει πραγματικό κόσμο.

Με τα θεμέλια του σχεδιασμού και των απαιτήσεων να έχουν τεθεί ξεκάθαρα, στο επόμενο κεφάλαιο θα παρουσιαστεί η πρακτική υλοποίηση του συστήματος. Εκεί θα αναλυθούν οι τεχνικές προκλήσεις, η αρχιτεκτονική του κώδικα και οι αλγόριθμοι που αναπτύχθηκαν για να ανταποκριθούν στον σχεδιασμό που περιγράφηκε παραπάνω.

Κεφάλαιο 4ο: Υλοποίηση Συστήματος OrderFlow

4.1 Εισαγωγή

Στο παρόν κεφάλαιο αναλύεται εκτενώς, ο συνολικός σχεδιασμός και η υλοποίηση ολόκληρου του συστήματος OrderFlow, καλύπτοντας τόσο τις mobile εφαρμογές (Frontend), όσο και την υποδομή του συστήματος (Backend). Αντί να γραφτεί σε μία ενιαία και ογκώδη εφαρμογή για όλους τους χρήστες, ακολουθήθηκε η αρχιτεκτονική του διαχωρισμού των αρμοδιοτήτων.

Αποτέλεσμα αυτής της προσέγγισης είναι η δημιουργία ενός πλήρους και πολυεπίπεδου οικοσυστήματος (monorepo). Το monorepo αποτελείται από τρεις εφαρμογές. Καθεμία εκ των οποίων είναι αυστηρά προσαρμοσμένη στις ανάγκες ενός συγκεκριμένου ρόλου, του διαχειριστή, της εξυπηρέτησης και της παραγωγής. Τις τρεις εφαρμογές έρχεται να συμπληρώσει, μια κοινή βιβλιοθήκη κώδικα η οποία λειτουργεί ως "γέφυρα". Παρέχει τα κοινά μοντέλα και εργαλεία προς όλα τα υποσυστήματα. Φυσικά το σύστημα δεν θα μπορούσε να λειτουργήσει χωρίς την κεντρική υποδομή, το REST API και την σχεσιακή βάση δεδομένων.

Όλες οι mobile εφαρμογές αναπτύχθηκαν αξιοποιώντας το framework Flutter της Google. Η επιλογή της Flutter βασίστηκε στην ικανότητά της να παράγει native εφαρμογές για πολλαπλές πλατφόρμες (Android, iOS).

Μια από τις μεγαλύτερες τεχνικές προκλήσεις στη σχεδίαση τριών ξεχωριστών εφαρμογών, που έχουν όμως πολλά κοινά σημεία, είναι να αποφύγουμε να επαναλαμβάνουμε τον ίδιο κώδικα (αρχή προγραμματισμού DRY Don't Repeat Yourself). Οπότε ο κώδικας που χρειάζεται σε πάνω από μία εφαρμογή γράφτηκε στο πακέτο `order_flow_common`.

Το πακέτο `common` είναι ο πυρήνας (core library) ολόκληρου του Frontend. Περιέχει τα μοντέλα δεδομένων, τα οποία δημιουργούνται με την βιβλιοθήκη `built_value` εξασφαλίζοντας την αμεταβλητότητα των δεδομένων. Παράλληλα, το πακέτο φιλοξενεί τον μηχανισμό επικοινωνίας με το REST API, και την χρωματική ταυτότητα των εφαρμογών.

4.2 Υλοποίηση του Backend Συστήματος

4.2.1 Αρχιτεκτονική Multi-Module του Backend

Η υλοποίηση του backend βασίζεται στη γλώσσα προγραμματισμού Java και συγκεκριμένα στην έκδοση 21. Επίσης βασίζεται στο framework Spring Boot. Αντί να επιλέξω μονολιθική αρχιτεκτονική όπου ο κώδικας θα βρισκόταν σε ένα φάκελο, επέλεξα να χωρίσω τον κώδικα σε πολλαπλές ενότητες (Multi-Module Architecture). Για την αυτοματοποίηση του build, του συστήματος επέλεξα να χρησιμοποιήσω το εργαλείο Apache Maven

Στη ρίζα των φακέλων βρίσκεται ένα κεντρικό αρχείο ρυθμίσεων (`pom.xml`). Είναι απαραίτητο για τη λειτουργία της maven. Η κυριότερη λειτουργία του, είναι η διαχείριση των εκδόσεων των βιβλιοθηκών που χρησιμοποιούν τα modules. Έτσι ξέρουμε ότι όλα θα έχουν τις ίδιες εκδόσεις.

Η εφαρμογή οργάνωνεται ως multi-module Maven project[6] με τρεις διακριτές υπο-ενότητες (modules), εφαρμόζοντας την αρχή του διαχωρισμού των αρμοδιοτήτων (Separation of Concerns). Οι ενότητες είναι οι εξής:

- `estiasi-rest`: Αποτελεί τον πυρήνα του backend (Core REST API) και τρέχει στην θύρα 8080. Αυτή η ενότητα περιέχει όλη τη βασική επιχειρησιακή λογική (Business Logic) της εφαρμογής.

Περιλαμβάνει τα μοντέλα δεδομένων (JPA Entities), τα αντικείμενα πρόσβασης δεδομένων (DAOs) για την επικοινωνία με τη MariaDB, και τους controllers που απαντούν στα αιτήματα των mobile εφαρμογών.

- **estiasi-utils:** Αποτελεί ένα βοηθητικό module το οποίο αναλαμβάνει την διαχείριση των ρυθμίσεων της εφαρμογής. Ουσιαστικά αποτελείται από δύο κλάσεις. Η πρώτη είναι υπεύθυνη για την ανάγνωση εξωτερικών αρχείων ρυθμίσεων (όπως το config.ini). Έτσι μπορούμε να έχουμε διαφορετικές ρυθμίσεις, ανάλογα τον server που θέλουμε να χρησιμοποιήσουμε, χωρίς να χρειάζεται να κάνουμε αλλαγές στον κώδικα. Η δεύτερη κλάση αναλαμβάνει την δημιουργία, την αρχικοποίηση και την σύνδεση με τη βάση δεδομένων, αξιοποιώντας τις ρυθμίσεις από την προηγούμενη κλάση.
- **estiasi-test:** Είναι μία ενότητα αποκλειστικά για να δοκιμάσω την λειτουργία του REST API, και περιέχει κώδικα για τις δοκιμές. Είναι σε ξεχωριστό module για να είμαι σίγουρος ότι δεν θα ενσωματωθεί κατά λάθος, κώδικας δοκιμών στο τελικό εκτελέσιμο αρχείο (.jar) της εφαρμογής. Κάτι τέτοιο θα μεγάλωνε τα εκτελέσιμα αρχεία και ενδεχομένως να δημιουργούσε προβλήματα ασφάλειας.

4.2.2 Μοντέλα Δεδομένων - Οντότητες

Στον φάκελο data του backend, βρίσκονται οι κλάσεις που ονομάζονται οντότητες (Entities). Αυτές οι κλάσεις είναι απαραίτητες για την επικοινωνία του συστήματος με τη σχεσιακή βάση δεδομένων MariaDB. Μέσω ειδικών annotations (όπως το @Entity και το @Table), κάθε κλάση "αντιστοιχίζεται" (γίνεται mapped) αυτόματα σε έναν συγκεκριμένο πίνακα της βάσης, και κάθε πεδίο (field) σε μία στήλη.

Πολύπλοκες συνδέσεις της JPA, όπως το @OneToMany, είτε οδηγούν σε περίπλοκα join, είτε σε πολλαπλά ερωτήματα SQL. Για να εξοικονομήσω πόρους προτίμησα το annotation @Transient. Αυτό σημαίνει ότι κάποια πεδία δημιουργούνται μόνο στη μνήμη της Java, προσωρινά και προορίζονται αποκλειστικά για να σταλούν στο κινητό ως απάντηση JSON, χωρίς όμως να αποθηκεύονται απευθείας στη βάση δεδομένων.

Ας δούμε αναλυτικά την κάθε οντότητα και τα πεδία της:

Μοντέλο EstiasiUser (EstiasiUser.java) Αναπαριστά έναν υπάλληλο της επιχείρησης. Χρησιμοποιείται για την ταυτοποίηση, το login και τον έλεγχο πρόσβασης με βάση τους ρόλους του προσωπικού. Αποτελείται από τα παρακάτω πεδία:

- **id (Integer):** Το μοναδικό αναγνωριστικό του υπαλλήλου.
- **username (String):** Το όνομα χρήστη για τη σύνδεση στο σύστημα.
- **password (String):** Ο κωδικός πρόσβασης του υπαλλήλου.
- **firstName (String):** Το όνομα του υπαλλήλου.
- **lastName (String):** Το επώνυμο του υπαλλήλου.
- **typeId (Integer):** Ένας ακέραιος αριθμός που καθορίζει τον ρόλο του χρήστη (1 για διαχειριστή, 2 για σερβιτόρο, 3 για το προσωπικό της παραγωγής).
- **active (Boolean):** Μας δείχνει αν ο λογαριασμός είναι ενεργός ή αν έχει κλειδωθεί προσωρινά.
- **deleted (Boolean):** Μας δείχνει αν ο λογαριασμός έχει διαγραφεί. Αναγκαστικά κάνουμε soft delete ώστε να μην καταστραφεί το ιστορικό των παραγγελιών που είχε πάρει στο παρελθόν.

Μοντέλο Product (Product.java) Αναπαριστά ένα προϊόν και αποτελείται από τα παρακάτω πεδία:

- id (Integer): Το μοναδικό αναγνωριστικό του προϊόντος.
- name (String): Το όνομα του προϊόντος (π.χ. "Φραπέ").
- description (String): Μια σύντομη περιγραφή των συστατικών του.
- price (Double): Αρχική τιμή του προϊόντος. Δεν περιλαμβάνει επιλογές, τροποποιήσεις, έξτρα υλικά κλπ.
- imageUrl (String): Η διεύθυνση (URL) της φωτογραφίας του προϊόντος.
- available (Boolean): Δείκτης που δείχνει αν το προϊόν είναι διαθέσιμο ή αν έχει εξαντληθεί (out of stock).
- abbreviation (String): Συντομογραφία που χρησιμοποιεί το σύστημα για να αποκωδικοποιήσει τις χειρόγραφες παραγγελίες.

Μοντέλο MenuCategory (MenuCategory.java) Αναπαριστά μια κατηγορία του καταλόγου (π.χ. Καφέδες, Ποτά, Επιδόρπια). Αποτελείται από τα παρακάτω πεδία:

- id (Integer): Το μοναδικό αναγνωριστικό της κατηγορίας.
- name (String): Ο τίτλος της κατηγορίας.
- imageUrl (String): Η διεύθυνση (URL) της φωτογραφίας, της κατηγορίας.
- disabled (Boolean): Δείκτης που καθορίζει αν η κατηγορία έχει απενεργοποιηθεί προσωρινά.
- items (List): Μια λίστα (@Transient) που γεμίζει δυναμικά κατά την εκτέλεση με τα προϊόντα που ανήκουν στην συγκεκριμένη κατηγορία.

Μοντέλο MenuItem (MenuItem.java) Αναπαριστά τη σύνδεση ενός προϊόντος με μια κατηγορία. Έχει τα εξής πεδία:

- id (Integer): Το μοναδικό αναγνωριστικό του MenuItem.
- categoryId (Integer): Το μοναδικό αναγνωριστικό της κατηγορίας στην οποία ανήκει.
- productId (Integer): Το μοναδικό αναγνωριστικό του προϊόντος.
- cost (Double): Αρχική τιμή.
- available (Boolean): Δείκτης που μας δείχνει αν το προϊόν είναι διαθέσιμο.
- product (Product): Ένα αντικείμενο (@Transient) που γεμίζει από τον Controller με τα πλήρη στοιχεία του προϊόντος (όνομα, εικόνα).
- itemOption (List): Λίστα (@Transient) με όλες τις επιλογές (π.χ. Ζάχαρη, Γάλα) που συνδέονται με αυτό το προϊόν.

Μοντέλο ItemOption (ItemOption.java) Αναπαριστά μια ομάδα επιλογών (Option Group) για ένα συγκεκριμένο προϊόν (π.χ. "Επιλογή Ζάχαρης").

- id (Integer): Το αναγνωριστικό της ομάδας.
- description (String): Το όνομα της ομάδας (π.χ. "Είδος ζάχαρης").
- menuItemId (Integer): Μοναδικό αναγνωριστικό του MenuItem που σχετίζεται με το συγκεκριμένο προϊόν.
- isMultipleChoice (Boolean): Δείκτης που καθορίζει αν το συγκεκριμένο πεδίο είναι πολλαπλής επιλογής.
- values (List): Μια λίστα (@Transient) που περιέχει τις διαθέσιμες επιλογές της συγκεκριμένης προτίμησης.

Μοντέλο ItemOptionValue (ItemOptionValue.java) Αναπαριστά μία συγκεκριμένη επιλογή για την προτίμηση του πελάτη. Π.χ. αν η προτίμηση είναι είδος ζάχαρης, μία επιλογή μπορεί να είναι η λευκή ζάχαρη. Αποτελείται από τα παρακάτω πεδία:

- id (Integer): Το αναγνωριστικό της επιλογής.
- value (String): Το όνομα της επιλογής (π.χ. "Λευκή ζάχαρη" ή "Ζαχαρίνη").
- extraCost (Double): Το επιπλέον κόστος, αν υπάρχει (π.χ. +0.50€ για γάλα αμυγδάλου).
- itemOptionId (Integer): Το αναγνωριστικό της ομάδας στην οποία ανήκει.

Μοντέλο Order (Order.java) Αναπαριστά μια παραγγελία, ακριβώς όπως αποθηκεύεται στον πίνακα estiasi_order της βάσης δεδομένων.

- id (Integer): Το μοναδικό αναγνωριστικό της παραγγελίας.
- table (Integer): Ο αριθμός του τραπεζιού στο οποίο πρέπει να πάει συγκεκριμένη παραγγελία.
- orderId (Integer): Ένας ακέραιος (1 έως 4) που δείχνει την κατάσταση της παραγγελίας αυτή τη στιγμή.
- status (OrderStatus): Ένα αντικείμενο (@Transient) που υπολογίζει δυναμικά το όνομα της κατάστασης, με βάση τον παραπάνω ακέραιο αριθμό.
- items (List): Μια λίστα (@Transient) με όλα τα προϊόντα που έχει μέσα η παραγγελία.

Μοντέλο OrderItem (OrderItem.java) Αναπαριστά το προϊόν μέσα σε μια παραγγελία, δηλαδή το προϊόν μαζί με τις προτιμήσεις και τις επιλογές του πελάτη.

- id (Integer): Το αναγνωριστικό του συγκεκριμένου παραμετροποιημένου προϊόντος.
- quantity (Integer): Η συνολική ποσότητα (π.χ. 3 ίδιοι καφέδες).
- orderId (Integer): Το αναγνωριστικό της παραγγελίας στην οποία ανήκει.
- menuItemId (Integer): Το αναγνωριστικό του MenuItem από το οποίο προήλθε.
- name (String): Το όνομα του προϊόντος. Αποθηκεύεται ανεξάρτητα για να διασφαλίσουμε την ακεραιότητα των δεδομένων. Αν στο μέλλον το προϊόν σβηστεί από τον κατάλογο, η παραγγελία και η απόδειξη δεν πρέπει να χαλάσουν ή να εμφανίσουν κενό.
- price (Double): Η τελική τιμή του προϊόντος τη στιγμή της παραγγελίας. Εδώ έχουμε συνυπολογίσει και τυχόν παραμετροποίηση του προϊόντος για παράδειγμα έξτρα υλικά.
- note (String): Επιπλέον παρατηρήσεις ή αιτήματα του πελάτη, που δεν υπάρχουν στο σύστημα σαν επιλογές.
- selectedOptionsJson (String): Το σύστημα αποθηκεύει τις προτιμήσεις του πελάτη απευθείας σε μορφή JSON μέσα στη βάση. Αυτό γίνεται προκειμένου να γλιτώσει το σύστημα από πολύπλοκα JOINS, κερδίζοντας σε ταχύτητα κατά την ανάκτηση της παραγγελίας από την βάση.
- selectedOptions (List): Μια δυναμική λίστα (@Transient) που περιέχει τα δεδομένα της προηγούμενης στήλης αλλά parsed.

Για να γίνει πιο κατανοητός ο τρόπος με τον οποίο απέφυγα τα περίπλοκα ερωτήματα της JPA, παραθέτω ένα απόσπασμα από το μοντέλο OrderItem. Όπως φαίνεται στον παρακάτω κώδικα, η λίστα με τα επιλεγμένα υλικά (selectedOptions) μαρκάρεται με το annotation @Transient ώστε να αγνοηθεί

Κεφάλαιο 4

από τη βάση, ενώ τα δεδομένα αποθηκεύονται πρακτικά ως απλό κείμενο στο πεδίο `selectedOptionsJson`.

`@Entity`

`@Table(name = "order_item")`

`public class OrderItem implements Serializable {`

`@Id`

`@GeneratedValue(strategy = GenerationType.IDENTITY)`

`@Column(name = "order_item_id")`

`private Integer id;`

`@Column(name = "order_quantity")`

`private Integer quantity;`

`@Column(name = "order_id")`

`private Integer orderId;`

`@Column(name = "menu_item_id")`

`private Integer menuItemId;`

`@Column(name = "item_name")`

`private String name;`

`@Column(name = "item_price")`

`private Double price;`

`@Column(name = "selected_options", columnDefinition = "TEXT")`

`private String selectedOptionsJson;`

`@Column(name = "item_note", columnDefinition = "TEXT")`

`private String note;`

`@Transient`

`private MenuItem menuItem;`

`@Transient`

`private List<OrderItemOption> options;`

`@Transient`

`private List<Object> selectedOptions;`

`// ... παραλείπονται οι μέθοδοι get/set για συντομία`

`}`

Μοντέλο OrderStatus (OrderStatus.java) Αποθηκεύει τα ονόματα των καταστάσεων που μπορεί να έχει μια παραγγελία.

- id (Integer): Το αναγνωριστικό της κατάστασης.
- status (String): Το όνομα της κατάστασης.

4.2.3 Επίπεδο Πρόσβασης Δεδομένων (DAOs)

Το επίπεδο Πρόσβασης Δεδομένων (Data Access Objects - DAO) στο backend έχει υλοποιηθεί χρησιμοποιώντας το framework Spring Data JPA. Το Spring Data JPA είναι ένα εργαλείο που αυτοματοποιεί την επικοινωνία με τη σχεσιακή βάση δεδομένων (MariaDB) μέσω του Hibernate ORM. Αντί να γράψω SQL ερωτήματα για κάθε πίνακα, υλοποίησα διεπαφές (interfaces) που κληρονομούν την κλάση JpaRepository. Αυτό δίνει αυτόματα σε κάθε DAO όλες τις βασικές λειτουργίες CRUD (Create, Read, Update, Delete) έτοιμες προς χρήση[8]. Όπου χρειάζεται κάτι πιο περίπλοκο, χρησιμοποιείται η γλώσσα JPQL (Java Persistence Query Language) μέσω του annotation @Query, και μου επιτρέπει να αναζητήσω με βάση τη λογική των αντικειμένων της Java και όχι τους πίνακες της βάσης δεδομένων.

Η δομή των repositories του συστήματος αποτελείται από τα παρακάτω 10 αρχεία:

EstiasiUserRepository (EstiasiUserRepository.java) Αναπαριστά το επίπεδο πρόσβασης για τους λογαριασμούς του προσωπικού. Εκτός από τις βασικές λειτουργίες, περιέχει το query getUserByCredentials(). Το query αυτό δεν ελέγχει μόνο αν το username και το password είναι σωστά, αλλά φιλτράρει δυναμικά (active = true AND deleted = false) διασφαλίζοντας ότι διαγραφμένοι ή ανενεργοί υπάλληλοι δεν θα έχουν πρόσβαση στην εφαρμογή.

Παραθέτω ένα σύντομο παράδειγμα repository:

```
public interface EstiasiUserRepository extends JpaRepository<EstiasiUser, Integer> {

    @Query("SELECT e FROM EstiasiUser e WHERE e.username = :username AND e.password = :password AND e.active = true AND e.deleted = false")

    EstiasiUser getUserByCredentials(@Param("username") String username, @Param("password") String password);

}
```

MenuCategoryRepository (MenuCategoryRepository.java) Αναπαριστά την επικοινωνία με τον πίνακα των κατηγοριών του μενού. Δεν περιέχει περίπλοκα ερωτήματα, καθώς αξιοποιεί τις ενσωματωμένες

μεθόδους του `JpaRepository` (όπως `findAll()` και `findById()`) για την ανάκτηση ή δημιουργία των κατηγοριών (π.χ. Καφέδες, Ποτά, Επιδόρπια).

`ProductRepository` (`ProductRepository.java`) Αναπαριστά την επικοινωνία με τον πίνακα των προϊόντων. Όπως συμβαίνει και με τις κατηγορίες, κληρονομεί αυτόματα όλες τις μεθόδους από το `Spring Data JPA`, επιτρέποντας τη μαζική διαχείριση (αποθήκευση, διαγραφή, εύρεση) του καταλόγου προϊόντων, χωρίς να χρειάζεται να γράψω κώδικα SQL.

`MenuItemRepository` (`MenuItemRepository.java`) Αναπαριστά την σύνδεση ανάμεσα στα προϊόντα και τις κατηγορίες. Περιέχει δύο πολύ σημαντικά query: την `findByMenuCategoryId()`, που επιστρέφει όλα τα προϊόντα μιας συγκεκριμένης κατηγορίας, και το `findActiveByMenuCategoryId()`, που λειτουργεί ως δυναμικό φίλτρο, επιστρέφοντας αυστηρά μόνο τα προϊόντα που είναι διαθέσιμα για παραγγελία τη δεδομένη στιγμή, και ανήκουν στη συγκεκριμένη κατηγορία.

`ItemOptionRepository` (`ItemOptionRepository.java`) Χρησιμοποιείται για την ανάκτηση των Ομάδων Επιλογών (Option Groups, π.χ. Ζάχαρη, Γάλα) ενός συγκεκριμένου προϊόντος. Παρέχει την `findByMenuItemId()`, η οποία δέχεται ως παράμετρο το αναγνωριστικό του προϊόντος και επιστρέφει ως λίστα (List) όλες τις ομάδες επιλογών που συνδέονται με αυτό.

`ItemOptionValueRepository` (`ItemOptionValueRepository.java`) Αναλαμβάνει τη διαχείριση των συγκεκριμένων επιλογών (π.χ. Μέτριος, Γλυκός). Η μέθοδος `findByItemOptionId()` αναζητά και επιστρέφει όλες τις υπο-επιλογές (μαζί με την πιθανή έξτρα χρέωσή τους), που ανήκουν σε μια συγκεκριμένη ομάδα (Option Group), ολοκληρώνοντας έτσι την δενδρική δομή του καταλόγου.

`OrderRepository` (`OrderRepository.java`) Αναπαριστά τη διαχείριση μιας παραγγελίας (π.χ. ημερομηνία, τραπέζι, σημειώσεις, κατάσταση). Αξιοποιεί αποκλειστικά την ενσωματωμένη υλοποίηση του `JpaRepository`, καθώς η σύνθετη λογική (ταξινόμηση και φιλτράρισμα ενεργών παραγγελιών) διεκπεραιώνεται σε υψηλότερο επίπεδο (στους `Controllers`) και όχι απευθείας στη βάση δεδομένων.

`OrderStatusRepository` (`OrderStatusRepository.java`) Διαχειρίζεται τις πιθανές καταστάσεις που μπορεί να λάβει μια παραγγελία (1: Pending, 2: Seen, 3: Ready, 4: Paid). Χρησιμοποιείται κυρίως ως "λεξικό" της βάσης δεδομένων, για να επικυρώσει (validate) ότι μια αλλαγή κατάστασης που ζητείται από την κουζίνα ή τον σερβιτόρο αντιστοιχεί σε έναν υπαρκτό και έγκυρο ακέραιο αριθμό. Έκρινα ότι αυτό είναι απαραίτητο ώστε να μπορώ να επεκτείνω μελλοντικά το σύστημα με παραπάνω καταστάσεις παραγγελιών.

`OrderItemRepository` (`OrderItemRepository.java`) Αναλαμβάνει να διαχειριστεί τα προϊόντα μέσα σε μία συγκεκριμένη παραγγελία (λειτουργεί σαν καλάθι). Η συνάρτηση `findByOrderId()` εκτελεί ένα JPQL ερώτημα που δέχεται ως όρισμα το ID της παραγγελίας και επιστρέφει όλα τα προϊόντα που περιέχει, μαζί με την ποσότητα και τις σημειώσεις του καθενός.

`OrderItemOptionRepository` (`OrderItemOptionRepository.java`) Αναπαριστά τις συγκεκριμένες επιλογές που έκανε ο πελάτης πάνω στο εκάστοτε προϊόν. Μέσω της μεθόδου `findByOrderItemId()`, το σύστημα ανακτά τις προτιμήσεις (π.χ. χωρίς πάγο, με στέβια) που συνδέονται με ένα συγκεκριμένο προϊόν (`OrderItem`) μιας συγκεκριμένης παραγγελίας.

4.2.4 Επίπεδο Ελεγκτών (REST Controllers)

Το επίπεδο των Ελεγκτών (Controllers) αποτελεί τον πυρήνα του backend. Είναι το σημείο όπου τα HTTP αιτήματα (requests) από τις τρεις mobile εφαρμογές παραλαμβάνονται, αναλύονται, και προωθούνται στην κατάλληλη επιχειρησιακή λογική. Στο Spring Boot, η υλοποίηση αυτού του επιπέδου γίνεται μέσω της χρήσης του annotation `@RestController`. Αυτό δηλώνει αυτόματα ότι κάθε μέθοδος της κλάσης δεν θα επιστρέψει κώδικα HTML, αλλά θα βάλει τα δεδομένα απευθείας στο σώμα (body) της απάντησης (HTTP Response), σε μορφή JSON.

Η διασύνδεση των ελεγκτών με το επίπεδο πρόσβασης δεδομένων (τα DAOs που αναλύθηκαν προηγουμένως) επιτυγχάνεται μέσω της αρχιτεκτονικής Dependency Injection χρησιμοποιώντας το annotation `@Autowired`[8]. Οι ελεγκτές αντιστοιχίζουν τις βασικές λειτουργίες διαχείρισης πόρων (CRUD) στα ρήματα του πρωτοκόλλου HTTP:

- GET: Για την ανάκτηση πληροφοριών (π.χ. `@GetMapping("/order")`).
- POST: Για τη δημιουργία νέων εγγράφων (π.χ. `@PostMapping("/order")`).
- PUT: Για την πλήρη ανανέωση υφιστάμενων εγγράφων (π.χ. `@PutMapping("/order")`).
- DELETE: Για τη διαγραφή πόρων (π.χ. `@DeleteMapping("/order/{id}")`).
-

Ένα πλεονέκτημα των ελεγκτών του συστήματος είναι η χρήση κανονικών εκφράσεων (Regular Expressions), όπως η `Pattern.compile("^\\d+$")`, για την επικύρωση (validation) των παραμέτρων της διαδρομής (`@PathVariable`). Αυτό διασφαλίζει αυστηρά ότι τα αναγνωριστικά (IDs) που στέλνουν οι τρεις εφαρμογές, είναι ακέραιοι αριθμοί πριν εκτελεστεί οποιοδήποτε ερώτημα στη βάση, προστατεύοντας το σύστημα από κακόβουλες επιθέσεις (όπως το SQL Injection).

Στο σύστημα έχουν υλοποιηθεί 11 συνολικά Controllers. Ενώ οι περισσότεροι (όπως οι `ProductController`, `ItemOptionController`) λειτουργούν ως τυπικές διεπαφές CRUD, υπάρχουν τρεις ελεγκτές που ενσωματώνουν ιδιαίτερα προηγμένη λογική και αξίζουν ειδικής ανάλυσης:

4.2.4.1 MenuController: Αρχιτεκτονική Ενοποιημένου Σημείου (Aggregate Endpoint Pattern)

Ο `MenuController` Χρειάστηκε πολύ προσεκτικό σχεδιασμό για να έχουμε καλή απόδοση. Σε ένα κακό API, η εφαρμογή της εξυπηρέτησης θα έπρεπε να κάνει πολλαπλές ξεχωριστές κλήσεις: μία για να πάρει τις κατηγορίες, μετά μία για κάθε κατηγορία ώστε να πάρει τα προϊόντα, και έπειτα μία για κάθε προϊόν ώστε να πάρει τα υλικά. Αυτό είναι γνωστό στη βιβλιογραφία ως το πρόβλημα των N+1 ερωτημάτων (N+1 Queries Problem)[7]. Αντί για αυτό, το σύστημά εδώ έχει ένα μοναδικό, ενοποιημένο σημείο πρόσβασης: το `GET /menu`. Η μέθοδος `getMenu()` αναλαμβάνει όλο τον υπολογιστικό φόρτο, της συναρμολόγησης. Καλεί διαδοχικά τα απαραίτητα repositories, συνθέτει την δενδρική δομή (Κατηγορίες -> Προϊόντα -> Ομάδες Επιλογών -> Τελικές Επιλογές) και επιστρέφει ένα ολοκληρωμένο, εμφωλευμένο (nested) JSON. Έτσι, το κινητό του σερβιτόρου χρειάζεται να κάνει μόνο ένα HTTP Request στην αρχή της βάρδιας για να κατεβάσει ολόκληρο τον κατάλογο.

4.2.4.2 OrderController: Δυναμική Φόρτωση Δεδομένων και Serialization με Gson

Ο `OrderController` διαχειρίζεται τις παραγγελίες. Η πιο σύνθετη λειτουργία του, είναι στη βοηθητική μέθοδο `getOrderDetails()`. Επειδή η σχεσιακή βάση δεδομένων αποθηκεύει μόνο κωδικούς (IDs), η μέθοδος αυτή παίρνει μια άδεια παραγγελία και την συμπληρώνει με τα πραγματικά αντικείμενα (π.χ. βρίσκει το προϊόν από το ID) στα πεδία που είναι χαρακτηρισμένα ως `@Transient`. Ιδιαίτερο ενδιαφέρον έχει ο τρόπος που διαχειρίζεται τις επιλογές του πελάτη (`selectedOptions`). Όπως ανέφερα στα μοντέλα,

στη βάση αποθηκεύονται ως ένα απλό κείμενο JSON. Ο OrderController χρησιμοποιεί τη βιβλιοθήκη Google Gson[9]. Κατά τη δημιουργία της παραγγελίας (POST), χρησιμοποιεί την gson.toJson() για να μετατρέψει τη λίστα των επιλογών σε κείμενο (Serialization). Αντίστροφα, κατά την ανάκτηση (GET), χρησιμοποιεί την gson.fromJson() σε συνδυασμό με το TokenType για να αποκωδικοποιήσει το κείμενο πίσω σε πλήρη αντικείμενα Java (Deserialization), πριν αυτά καταλήξουν στο κινητό κάποιου υπάλληλου.

4.2.4.3 EstiasiUserController: Ταυτοποίηση μέσω DTO

Ο EstiasiUserController διαχειρίζεται τους λογαριασμούς και την πρόσβαση του προσωπικού στις τρεις εφαρμογές. Η κύρια λειτουργία του είναι το endpoint ταυτοποίησης (POST /estiasi-user/login). Αντί να λαμβάνει ολόκληρο τον χρήστη από το αίτημα, χρησιμοποιεί μια εξειδικευμένη κλάση, ένα αντικείμενο Μεταφοράς Δεδομένων (DTO - Data Transfer Object)[1], το Credentials, το οποίο περιέχει μόνο το username και το password. Στη συνέχεια, περνάει αυτά τα δεδομένα στο ειδικό ερώτημα της βάσης για να ελέγξει την ορθότητά τους. Αν ο υπάλληλος βρεθεί και είναι ενεργός, ο ελεγκτής επιστρέφει πίσω στο κινητό όλα τα στοιχεία του (συμπεριλαμβανομένου του ρόλου του), επιτρέποντας στην εφαρμογή να φορτώσει τις σχετικές οθόνες.

4.2.5 Διαχείριση Ρυθμίσεων και Σύνδεσης με τη Βάση Δεδομένων

Συνήθως, όταν φτιάχνουμε μια εφαρμογή με το Spring Boot, αφήνουμε το ίδιο το framework να διαχειριστεί τις ρυθμίσεις της βάσης, μέσα από το έτοιμο αρχείο application.properties. Στη δική μας περίπτωση όμως, παρατήρησα ότι ο έτοιμος μηχανισμός του Spring δεν μας παρέχει μεγάλη ευελιξία για τον τρόπο που θέλαμε να στήσουμε τον server. Για αυτόν τον λόγο, επέλεξα μια πιο άμεση λύση, φτιάχνοντας το ξεχωριστό module estiasi-utils. Ουσιαστικά, αυτό το module στηρίζεται σε δύο κλάσεις: την ConfigurationReader, η οποία διαβάζει τα στοιχεία πολύ πιο απλά από ένα εξωτερικό αρχείο config.ini, και την DBConnectionConfiguration, η οποία παίρνει αυτά τα στοιχεία και αναλαμβάνει τη σύνδεση με τη MariaDB.

4.2.5.1 Μηχανισμός Ανάγνωσης Ρυθμίσεων (ConfigurationReader)

Η κλάση ConfigurationReader είναι υπεύθυνη για την ανάγνωση εξωτερικών αρχείων ρυθμίσεων. Αυτό μας δίνει την δυνατότητα να έχουμε τις ρυθμίσεις σε εξωτερικό αρχείο και όχι μέσα στον κώδικα, πράγμα που σημαίνει ότι αν θελήσουμε να τις αλλάξουμε αργότερα δεν χρειάζεται να ξανακάνουμε build και να παράξουμε νέο .jar. Για την υλοποίηση αυτής της λειτουργίας, χρησιμοποιείται η κλάση FileReader της Java, από το πακέτο java.io, η οποία ανοίγει ένα ρεύμα ανάγνωσης (stream) προς το αρχείο. Στη συνέχεια, τα δεδομένα φορτώνονται στο αντικείμενο Properties του πακέτου java.util, το οποίο αποκωδικοποιεί αυτόματα τα ζεύγη κλειδιού-τιμής αγνοώντας τυχόν κενά.

4.2.5.2 Προγραμματιστική Σύνδεση με την Βάση Δεδομένων (DBConnectionConfiguration)

Η κλάση DBConnectionConfiguration αναλαμβάνει να συνδυάσει τις εξωτερικές ρυθμίσεις του .ini με τον μηχανισμό δεδομένων του Spring Boot. Χρησιμοποιώντας το annotation @Configuration, δείχνει

στο Spring Framework ότι πρόκειται για κλάση που διαχειρίζεται το ίδιο το Spring και η οποία πρέπει να εκτελεστεί κατά την εκκίνηση του συστήματος.

Μέσω της μεθόδου `getDataSource()`, η οποία έχει το annotation `@Bean`, κατασκευάζεται δυναμικά το αντικείμενο `DataSource` που χρειάζεται για τη σύνδεση με τη MariaDB. Η μέθοδος καλεί αρχικά τον `ConfigurationReader` για να διαβάσει τις τέσσερις κρίσιμες παραμέτρους σύνδεσης:

- `driver.name`: Το όνομα του JDBC driver που διαχειρίζεται το πρωτόκολλο επικοινωνίας (π.χ. `org.mariadb.jdbc.Driver`).
- `db.host.url`: Το τελικό URL της βάσης δεδομένων στο δίκτυο.
- `db.username`: Το όνομα χρήστη (username) της βάσης.
- `db.password`: Τον κωδικό πρόσβασης.

Στη συνέχεια, η μέθοδος χρησιμοποιεί την κλάση `DataSourceBuilder` (υλοποιώντας ουσιαστικά το σχεδιαστικό πρότυπο `Builder Pattern`)[11] για να "χτίσει" το αντικείμενο της σύνδεσης. Τέλος, το επιστρέφει στο Spring Context, δίνοντας στο σύστημα πλήρη έλεγχο πάνω στον τρόπο που αρχικοποιείται η σύνδεση με τη βάση δεδομένων.

4.2.6 Δοκιμές Ενσωμάτωσης (Integration Testing)

Ένα ολοκληρωμένο και έτοιμο για παραγωγή σύστημα, απαιτεί έλεγχο ποιότητας πριν την τελική του διάθεση. Για το σκοπό αυτό, δημιουργήθηκε το αυτόνομο module `estiasi-test`. Η κλάση `TestRest` προσομοιώνει πραγματικά σενάρια χρήσης (End-to-End testing). Εκτελεί πραγματικά HTTP αιτήματα (requests) προς τη θύρα 8080 (εκεί που τρέχει το κεντρικό API), ελέγχοντας παράλληλα αν τα δεδομένα γράφονται σωστά στη βάση δεδομένων (MariaDB).

Η διαδικασία ελέγχου χωρίζεται σε τρία βασικά στάδια:

- **Αρχικοποίηση και Τροφοδότηση Δεδομένων (Database Seeding)**: Μέσω της μεθόδου `test()`, το σύστημα εκτελεί διαδοχικά POST αιτήματα για να δημιουργήσει έναν βασικό, δοκιμαστικό κατάλογο. Συγκεκριμένα, δημιουργεί Προϊόντα (π.χ. "Frappe", "Espresso"), τα εισάγει στην κατηγορία "Καφέδες", και δημιουργεί διαδοχικά τις απαραίτητες Επιλογές (π.χ. "Είδος Ζάχαρης"). Αυτή η διαδικασία δεν αποτελεί μόνο έναν έλεγχο του API (ότι γίνονται εγγραφές), αλλά λειτουργεί και ως μηχανισμός τροφοδότησης (seeding). Έτσι έχοντας ήδη έναν έτοιμο κατάλογο, μπορώ να κάνω δοκιμές και στις τρεις εφαρμογές, χωρίς να απαιτείται η χειροκίνητη εισαγωγή δεδομένων.
- **Έλεγχος Ενοποιημένου Σημείου (Aggregate Endpoint Testing)**: Η μέθοδος `test2()` επαληθεύει ότι το περίπλοκο `Aggregate Endpoint` του καταλόγου (`MenuController`) λειτουργεί σωστά. Στέλνει ένα απλό GET αίτημα στο `/menu` και ελέγχει αν τα πλήρη δεδομένα (Κατηγορίες, μαζί με τα εμφωλευμένα προϊόντα και τις τελικές επιλογές) έρχονται σωστά. Παράλληλα, επαληθεύει ότι τα δεδομένα μετατρέπονται σωστά σε δομή JSON από τη βιβλιοθήκη `Gson`, διασφαλίζοντας ότι η εφαρμογή εξυπηρέτησης, θα λάβει ένα ορθά δομημένο δέντρο πληροφοριών.
- **Έλεγχος Διαχείρισης Προσωπικού και Ταυτοποίησης**: Οι μέθοδοι `test3()` και `test4()` ασχολούνται αποκλειστικά με το κρίσιμο κομμάτι της ταυτοποίησης. Αρχικά, δημιουργούνται δοκιμαστικοί υπάλληλοι με διαφορετικά δικαιώματα και καταστάσεις (π.χ. ένας ενεργός

σερβιτόρος, ένας απολυμένος/ανενεργός, ένας διεγγραμμένος λογαριασμός). Έπειτα, το σύστημα προσπαθεί να εκτελέσει λειτουργία login (POST /login) δοκιμάζοντας όλα τα πιθανά σενάρια. Μέσω αυτού του ελέγχου, διασφαλίζεται ότι το API απορρίπτει επιτυχώς όσους λογαριασμούς δεν πληρούν τις αυστηρές προϋποθέσεις (active = true και deleted = false).

Η αρχιτεκτονική επιλογή του να τοποθετηθούν αυτές οι δοκιμές σε ένα εντελώς ξεχωριστό module (estiasi-test) είναι απολύτως σκόπιμη. Εξασφαλίζει ότι ο δοκιμαστικός κώδικας και τα mock δεδομένα δεν θα ενσωματωθούν ποτέ στο τελικό, production εκτελέσιμο αρχείο (.jar) του Server.

4.3 Ανάπτυξη του Κοινόχρηστου Πακέτου

Το σύστημα παραγγελιοληψίας αποτελείται από τρεις διακριτές εφαρμογές, την εφαρμογή του διαχειριστή, την εφαρμογή της εξυπηρέτησης, και την εφαρμογή παραγωγής.

Και οι τρεις εφαρμογές έχουν κάποιες κοινές ανάγκες, χρησιμοποιούν τις ίδιες γραμματοσειρές, κοινά χρώματα και όλες έχουν την ανάγκη να ανακτούν δεδομένα από τη βάση δεδομένων και φυσικά χρειάζονται τα ίδια μοντέλα. Για παράδειγμα όλες χρησιμοποιούν το μοντέλο “προϊόν”.

Προκειμένου να μειώσω τον κίνδυνο λαθών και να αποφύγω τον διπλότυπο ή και τριπλότυπο κώδικα χρειάστηκε να αναπτύξω ένα κοινό πακέτο dart που το ονόμασα common. Εκείνο λειτουργεί ως μια βιβλιοθήκη για να μπορούν οι τρεις εφαρμογές να παίρνουν τα μοντέλα, τις σταθερές και τις μεθόδους που χρειάζονται. Η συγκεκριμένη αρχιτεκτονική επιταχύνει πολύ την ανάπτυξη καθώς τα σημαντικά και χρονοβόρα τμήματα του κώδικα όπως η σύνδεση με το back end, οι ηχητικές ειδοποιήσεις και η ανάκτηση δεδομένων χρειάζεται να γραφτούν και να δοκιμαστούν μόνο μία φορά.

Το πακέτο common αποτελείται κυρίως από τρεις φακέλους. Τον φάκελο με τις σταθερές(constants), τον φάκελο δεδομένων(data) και τον φάκελο με τους providers. Ο φάκελος constants περιέχει δύο αρχεία, τα colors.dart και fonts.dart. Μέσα στο colors.dart έχουμε κωδικούς για κάθε χρώμα της χρωματικής ταυτότητας της εφαρμογής. Με αυτόν τον τρόπο αν αποφασίσουμε να αλλάξουμε μία απόχρωση, η αλλαγή θα γίνει σε ένα σημείο και όχι σε όλα τα σημεία που χρησιμοποιείται το συγκεκριμένο χρώμα. Π.χ. αν θελήσουμε πιο σκούρο γκρι φόντο. Μέσα στο fonts.dart έχω κωδικοποιήσει πολλές παραλλαγές και στυλ της γραμματοσειράς ubuntu. Έτσι έχω διαθέσιμες γραμματοσειρές με διαφορετικό μέγεθος και βάρος για κάθε πιθανή χρήση μέσα στις τρεις εφαρμογές. Αυτό σημαίνει ότι γράφω μία φορά το πλήρες στυλ και μετά μπορώ να το χρησιμοποιώ μόνο με το όνομά του σαν συντόμευση.

Στο πακέτο common υπάρχουν δύο providers, ο ColorProvider και ο FontsProvider, που υλοποιήθηκαν με τη χρήση ChangeNotifier της Flutter. Είναι ο μηχανισμός με τον οποίο οι τρεις εφαρμογές αποκτούν πρόσβαση στις σταθερές που έχουμε ορίσει για τα χρώματα και τις γραμματοσειρές. Λειτουργούν δηλαδή σαν κεντρικές αποθήκες δεδομένων που μοιράζουν αυτά τα δεδομένα όχι μόνο στις τρεις εφαρμογές αλλά και σε κάθε μία από τις σελίδες-διεπαφές που υπάρχουν μέσα σε αυτές.

Μέσα στον φάκελο data έχουμε τον υποφάκελο models. Περιέχει τα μοντέλα από όλες τις βασικές οντότητες του συστήματος π.χ. Προϊόντα, παραγγελίες, επιλογές και άλλα. Ο φάκελος models είναι ίσως ο πιο σημαντικός καθώς διασφαλίζει ότι και οι τρεις εφαρμογές, διαβάζουν και επεξεργάζονται τα δεδομένα με τον ίδιο ακριβώς τρόπο.

Για την υλοποίηση των παραπάνω μοντέλων αποφάσισα να χρησιμοποιήσω την βιβλιοθήκη `built_value` [4] της Dart. Η `built_value` έχει συγκεκριμένα πλεονεκτήματα που επιλύουν δύο συχνά προβλήματα στην ανάπτυξη περίπλοκων συστημάτων.

Το πρώτο πλεονέκτημα είναι ότι μας παρέχει αμετάβλητα αντικείμενα. Στις περισσότερες εφαρμογές τα δεδομένα μεταφέρονται διαδοχικά από την μία οθόνη στην άλλη. Π.χ. από τον κατάλογο του σερβιτόρου, στην οθόνη τροποποίησης παραγγελίας. Αν τα αντικείμενα μπορούσαν να μεταβάλλονται θα υπήρχε ο κίνδυνος ένα κομμάτι του κώδικα να αλλάξει κατά λάθος την τιμή ή την ποσότητα μιας παραγγελίας, προκαλώντας σφάλματα που είναι δύσκολο να εντοπιστούν. Η συγκεκριμένη βιβλιοθήκη επιβάλλει τα αντικείμενα να είναι αμετάβλητα, και αν χρειάζεται να κάνουμε κάποια αλλαγή, μας δίνει την δυνατότητα να κάνουμε ένα καινούργιο αντικείμενο χρησιμοποιώντας μία μέθοδο `builder`.

Η επικοινωνία των `mobile` εφαρμογών του συστήματος με το `backend` γίνεται μέσω `rest api` και η ανταλλαγή δεδομένων γίνεται με την μορφή `JSON`. Εδώ εμφανίζεται το δεύτερο σημαντικό πλεονέκτημα του `built_value`, η αυτόματη κωδικοποίηση και αποκωδικοποίηση, από και προς, την μορφή `JSON`. Μέσω του `built_value`, αυτή η διαδικασία αυτοματοποιείται πλήρως. Σε συνδυασμό με το `build_runner` της Dart, ο κώδικας για τον χειρισμό των `JSON` δεδομένων (`serialization / deserialization`) παράγεται αυτόματα. Έτσι επιταχύνουμε την ανάπτυξη και μειώνουμε τον κίνδυνο για ανθρώπινα λάθη π.χ. τυπογραφικά στα ονόματα πεδίων.

Παρακάτω θα παρουσιάσω συνοπτικά τα μοντέλα δεδομένων που αναπτύχθηκαν, ώστε να εξηγήσω την πρακτική χρήση τους, στις τρεις εφαρμογές καθώς και την δομή των δεδομένων που μεταφέρουν.

1. Μοντέλο `Category(category.dart)`

Αναπαριστά μία κατηγορία του καταλόγου, για παράδειγμα ποτά, καφέδες, επιδόρπια κλπ. Χρησιμοποιείται για την οργάνωση και την ομαδοποίηση προϊόντων. Ουσιαστικά επιτρέπει στον σερβιτόρο να επιλέγει με ευκολία προϊόντα από τον κατάλογο. Αποτελείται από τα παρακάτω πεδία:

- `id (String)`: Μοναδικό αναγνωριστικό της κατηγορίας.
- `name (String)`: Το όνομα της κατηγορίας.
- `imageUrl (String)`: Η διεύθυνση URL της εικόνας της κατηγορίας.
- `isDisabled (boolean)`: Ένας δείκτης που μας επιτρέπει να απενεργοποιήσουμε προσωρινά μία κατηγορία από τον κατάλογο. Τότε αυτή εμφανίζεται γκριζαρισμένη και το προσωπικό δεν μπορεί να επιλέξει προϊόντα από αυτή την κατηγορία.

2. Μοντέλο `Product(product.dart)`

Αναπαριστά ένα συγκεκριμένο προϊόν του καταλόγου. Περιέχει όλα τα χαρακτηριστικά του προϊόντος και μία λίστα από ομάδες επιλογών, για να μπορούν να αποθηκευτούν οι προτιμήσεις του πελάτη π.χ. με ζάχαρη, χωρίς πάγο κλπ. Έχει τα παρακάτω πεδία:

- `id (String)`: Το μοναδικό αναγνωριστικό του προϊόντος.
- `menuItemId (String)`: Συνδέει το προϊόν με ένα συγκεκριμένο αντικείμενο στον κατάλογο.
- `categoryId (String)`: Το μοναδικό αναγνωριστικό της κατηγορίας στην οποία ανήκει το προϊόν.
- `name (String)`: Το όνομα του προϊόντος.
- `description (String)`: Μία σύντομη περιγραφή του προϊόντος.
- `price (double)`: Η αρχική τιμή του προϊόντος.
- `imageUrl (String)`: Μία διεύθυνση URL για την φωτογραφία του προϊόντος.
- `isAvailable (boolean)`: Ένας δείκτης που μας επιτρέπει να θέσουμε το προϊόν εκτός διαθεσιμότητας. Τότε το προσωπικό δεν μπορεί να το επιλέξει από τον κατάλογο.
- `optionGroups (List of OptionGroup)`: Μία λίστα από αντικείμενα `optionGroups` που περιέχει όλες τις διαθέσιμες επιλογές που υπάρχουν για το συγκεκριμένο προϊόν.

- `abbreviation(String?)`: Είναι η συντομογραφία που γράφει ο σερβιτόρος, για να μπορέσει η εφαρμογή να αποκωδικοποιήσει τις χειρόγραφες παραγγελίες και να εντοπίσει το συγκεκριμένο προϊόν.

Καθώς το μοντέλο `Product` το συναντάμε σχεδόν παντού στην εφαρμογή και είναι το σημαντικότερο, παραθέτω ένα μικρό παράδειγμα κώδικα της υλοποίησης του. Έτσι θα έχουμε καλύτερη κατανόηση της αρχιτεκτονικής που βασίζεται στο `built_value`, τόσο για το συγκεκριμένο μοντέλο όσο και για τα υπόλοιπα.

```
abstract class Product implements Built<Product, ProductBuilder> {
    String get id;
    String? get menuItemId;
    String get categoryId;
    String get name;
    String get description;
    double get price;
    String get imageUrl;
    bool get isAvailable;
    BuiltList<OptionGroup> get optionGroups;
    String? get abbreviation;

    Product._();
    factory Product([void Function(ProductBuilder) updates]) = _$Product;

    static void _initializeBuilder(ProductBuilder b) => b
        ..imageUrl = ""
        ..isAvailable = true;
}
```

3. Μοντέλα `OptionGroup` και `OptionChoice`(`option_group.dart`)

Μέσα στο ίδιο αρχείο έχουμε δύο μοντέλα που είναι απαραίτητα για να αποθηκεύονται οι προτιμήσεις των πελατών, για συγκεκριμένα προϊόντα. Το μοντέλο `OptionGroup` αναπαριστά μία προτίμηση, για παράδειγμα το μέγεθος, ενώ το μοντέλο `OptionChoice` αναπαριστά τις συγκεκριμένες επιλογές που είναι διαθέσιμες για αυτή την προτίμηση. Για παράδειγμα μικρό, μεσαίο, η μεγάλο ποτήρι.

Το μοντέλο `OptionGroup` έχει τα εξής πεδία:

- `name (String)`: Το όνομα της προτίμησης για παράδειγμα ζάχαρη.
- `choices (BuiltList of OptionChoice)`: Μία λίστα με τις διαθέσιμες επιλογές για παράδειγμα λευκή, καστανή, η στέβια. Οι επιλογές αποθηκεύονται σαν αντικείμενα `OptionChoice`.
- `isMultipleChoice (boolean)`: Είναι ένας δείκτης που καθορίζει το αν μία επιλογή είναι πολλαπλής επιλογής. Την χρειαζόμαστε σε περίπτωση που ένας πελάτης θέλει να προσθέσει πολλαπλά υλικά.

- `abbreviation` (String, optional): Είναι η συντομογραφία που γράφει ο σερβιτόρος, για να μπορέσει η εφαρμογή να αποκωδικοποιήσει τις χειρόγραφες παραγγελίες και να εντοπίσει την συγκεκριμένη προτίμηση.

Το μοντέλο `OptionChoice` έχει τα εξής πεδία:

- `name` (String): Το όνομα της επιλογής.
- `extraCost` (double): Ορισμένες επιλογές μπορούν προαιρετικά να έχουν επιπλέον κόστος.
- `abbreviation` (String, optional): Είναι η συντομογραφία που γράφει ο σερβιτόρος, για να μπορέσει η εφαρμογή να αποκωδικοποιήσει τις χειρόγραφες παραγγελίες και να εντοπίσει την συγκεκριμένη επιλογή.

4. Μοντέλα `Order` και `OrderItem`(`order.dart`)

Έχουμε σε κοινό αρχείο το μοντέλο της παραγγελίας και το μοντέλο `OrderItem`, για τα προϊόντα που υπάρχουν στην παραγγελία. Το μοντέλο της παραγγελίας περιλαμβάνει όλα τα χαρακτηριστικά μιας παραγγελίας, όπως ποια προϊόντα επιλέχθηκαν, με ποιες επιλογές, την κατάσταση της παραγγελίας, το κόστος της κλπ. Το `OrderItem` αναπαριστά ένα προϊόν που περιλαμβάνεται σε μία παραγγελία, μαζί όμως με τις προτιμήσεις και τις επιλογές του πελάτη.

Το μοντέλο `Order` έχει τα παρακάτω πεδία:

- `id` (String, optional): Μοναδικό αναγνωριστικό της παραγγελίας, δημιουργείται από το backend, όταν η εφαρμογή του σερβιτόρου στείλει μία νέα παραγγελία.
- `tableNumber` (String): Το νούμερο του τραπεζιού στο οποίο αντιστοιχεί η συγκεκριμένη παραγγελία. Ένα τραπέζι φυσικά μπορεί να έχει παραπάνω από μία παραγγελία.
- `items` (List of `OrderItem`): Είναι μία λίστα από `OrderItem` που αναπαριστούν τα προϊόντα μιας παραγγελίας.
- `notes` (String): Σημειώσεις που αφορούν πράγματα που μπορεί να ζητήσουν οι πελάτες σχετικά με την παραγγελία. Αφορά κυρίως τροποποιήσεις για τις οποίες δεν υπάρχει επιλογή στο μενού για παράδειγμα να αφαιρεθεί κάποιο αλλεργιογόνο συστατικό.
- `status` (String): Η κατάσταση της παραγγελίας μπορεί να είναι είτε ολοκληρωμένη, είτε σε διαδικασία παραγωγής, η έτοιμη προς παράδοση.
- `isSeen` (boolean): Είναι ένας δείκτης για το προσωπικό παραγωγής και μας δείχνει αν το προσωπικό έχει δει την παραγγελία ή όχι.
- `isReady` (boolean): Δείκτης που καθορίζει αν μία παραγγελία είναι έτοιμη για παράδοση στο πελάτη ή όχι.
- `isPaid` (boolean): Δείκτης που καθορίζει αν μία παραγγελία έχει πληρωθεί.
- `totalAmount` (double, computed): Το συνολικό κόστος παραγγελίας. Υπολογίζεται δυναμικά και λαμβάνει υπόψη τα προϊόντα, τις τροποποιήσεις, τις επιλογές και τις ποσότητες.

Το μοντέλο `OrderItem` έχει τα παρακάτω πεδία:

- `name` (String): Όνομα προϊόντος.
- `quantity` (int): Ποσότητα.
- `price` (double): Αρχική τιμή. Η τιμή του προϊόντος πριν συνυπολογιστούν οποιεσδήποτε τροποποιήσεις που θέλει να κάνει ο πελάτης.

- `note` (String, optional): Σημειώσεις που αφορούν πράγματα που μπορεί να ζητήσουν οι πελάτες σχετικά με το προϊόν. Αφορά κυρίως τροποποιήσεις για τις οποίες δεν υπάρχει επιλογή στο μενού για παράδειγμα να αφαιρεθεί κάποιο αλλεργιογόνο συστατικό.
- `selectedOptions` (List of `SelectedOption`): Μία λίστα από `SelectedOption` που αναπαριστούν τις τροποποιήσεις και τις επιλογές που έχει ζητήσει ο πελάτης.
- `total` (double, computed): Είναι η τελική τιμή του προϊόντος, έχοντας συνυπολογίσει τυχόν τροποποιήσεις και προσθήκες που ζήτησαν οι πελάτες.

5. Μοντέλο `SelectedOption(selected_option.dart)`

Πρόκειται για σχετικά μικρό μοντέλο που υπάρχει για να μας διευκολύνει στην τήρηση ιστορικού. Αναπαριστά μία αλλαγή που ζήτησε ο πελάτης και το επιπλέον κόστος της. Αποτελείται από τα παρακάτω πεδία:

- `groupName` (String): Το όνομα της προτίμησης στην οποία ανήκει η παρούσα επιλογή.
- `choiceName` (String): Όνομα επιλογής.
- `extraCost` (double): Το κόστος που πρέπει να προστεθεί στο προϊόν.

6. Μοντέλο `StaffUser(staff_user.dart)`

Το μοντέλο `StaffUser` αναπαριστά τα προσωπικά στοιχεία των υπαλλήλων, τα στοιχεία αυθεντικοποίησης, και τον ρόλο του εκάστοτε υπαλλήλου. Αποτελείται από τα παρακάτω πεδία:

- `id` (String): Το μοναδικό αναγνωριστικό του υπαλλήλου.
- `username` (String): Το όνομα χρήστη που χρησιμοποιείται για την σύνδεση στην εφαρμογή.
- `password` (String): Ο κωδικός πρόσβασης του χρήστη.
- `firstName` (String): Το όνομα του υπαλλήλου.
- `lastName` (String): Το επώνυμο του υπαλλήλου.
- `email` (String): Η διεύθυνση ηλεκτρονικού ταχυδρομείου του υπαλλήλου.
- `role` (`StaffRole Enum`): Είναι ο ρόλος του υπαλλήλου (διαχειριστής, σερβιτόρος ή κουζίνα) και καθορίζει σε ποιες εφαρμογές έχει δικαίωμα εισόδου.
- `isActive` (boolean): Δείκτης που επιτρέπει την προσωρινή απενεργοποίηση της πρόσβασης του υπαλλήλου στο σύστημα, χωρίς όμως να διαγράφεται.
- `fullName` (String): Δυναμικό πεδίο που επιστρέφει αυτόματα το πλήρες ονοματεπώνυμο (όνομα και επώνυμο) για ευκολία στην εμφάνιση.

Μέσα στο πακέτο `common` υπάρχει επίσης ο φάκελος `services`. Εκεί συγκεντρώνουμε τον κώδικα που είναι υπεύθυνος για την επικοινωνία των `mobile` εφαρμογών με το `backend`. Αντί τα `API calls` να γίνονται μέσα στον κώδικα των `Widgets`, έχουν συγκεντρωθεί σε συγκεκριμένες κλάσεις (`Services`). Αυτή η αρχιτεκτονική (`Service-Repository Pattern`) κάνει τον κώδικα πιο ευανάγνωστο, ασφαλή και εύκολο στη συντήρηση.

Ο φάκελος `services` περιέχει έξι αρχεία, το καθένα με διαφορετικό ρόλο:

1. `ServiceRepository (service_repository.dart)` Είναι το κεντρικό αποθετήριο (`repository`) και των τριών εφαρμογών. Αρχικοποιείται μόνο μία φορά, κατά την εκκίνηση της εκάστοτε εφαρμογής (στο `main.dart`) και αρχικοποιεί όλα τα υπόλοιπα `services`. Δημιουργεί το `hook useRepo()`, το οποίο δίνει τη δυνατότητα, σε οποιαδήποτε οθόνη της εφαρμογής να ζητήσει δεδομένα άμεσα και εύκολα. Ουσιαστικά όλες οι οθόνες αποκτούν πρόσβαση στα δεδομένα των επιμέρους `services` μέσω ενός ενιαίου

μηχανισμού. Η συγκεκριμένη αρχιτεκτονική βασίζεται στην οργάνωση των λειτουργιών της εφαρμογής μέσω ενός Service layer και στην απομόνωση της πρόσβασης στα δεδομένα μέσω repository [1].

2. `HttpService` (`http_service.dart`) Είναι μια κλάση που αξιοποιεί την βιβλιοθήκη Dio[3], η οποία είναι υπεύθυνη για την αποστολή των HTTP αιτημάτων στο REST API. Μας βοηθά στο να απλοποιούμε τον κώδικα των υπόλοιπων services. Μας παρέχει τις συναρτήσεις `parseResponseMap` και `parseResponseList` που αναλαμβάνουν να μετατρέψουν το JSON που έρχεται από το backend, σε Lists και Maps.

3. `ProductService` (`product_service.dart`) Από εδώ περνάνε όλα τα δεδομένα που έχουν να κάνουν με τα προϊόντα και τον κατάλογο. Παίρνει από το backend τις κατηγορίες, τα προϊόντα και τις προτιμήσεις (options). Κάνει τις κλήσεις δημιουργίας, διαγραφής ή ενημέρωσης ενός προϊόντος, ενημέρωσης διαθεσιμότητας κλπ.

Καθώς το `ProductService` αποτελείται από σχεδόν 500 γραμμές κώδικα, είναι ίσως το πιο κομβικό service της εφαρμογής. Όχι μόνο από άποψη έκτασης αλλά και πολυπλοκότητας. Η πολυπλοκότητά του προκύπτει από τον τρόπο που διαχειρίζεται τα δεδομένα. Πιο συγκεκριμένα, οι εργασίες που εκτελεί είναι παρακάτω:

Ανάκτηση και αποκωδικοποίηση του καταλόγου (`fetchMenu`). Η μέθοδος `fetchMenu()` είναι υπεύθυνη για να κατεβάσει ολόκληρο τον κατάλογο του καταστήματος στην εφαρμογή του σερβιτόρου. Αντί να κάνει ξεχωριστές κλήσεις για κάθε προϊόν και κατηγορία, κάτι που θα καθυστερούσε σημαντικά το σύστημα, χτυπάει ένα κεντρικό endpoint (`/menu`) του REST API. Τα δεδομένα έρχονται σε μορφή JSON όμως η πληροφορία είναι εμφωλευμένη πολλαπλές φορές.[2] Συγκεκριμένα έχει τη μορφή: Κατηγορίες -> Προϊόντα -> Ομάδες Επιλογών (`OptionGroups`) -> Συγκεκριμένες Επιλογές (`Choices`). Ο κώδικας του service διαβάζει το JSON και δημιουργεί τα αντίστοιχα Dart μοντέλα που περιγράψαμε νωρίτερα.

Για να σιγουρευτούμε ότι δεν θα προβάλλουμε διπλότυπα προϊόντα, ο κώδικας περιέχει και δύο Set, το `categoryIdsSet` και το `productIdsSet`. Έτσι μπορεί να ελέγξει για διπλότυπα id ή ονόματα, τόσο για τα προϊόντα όσο και για τις κατηγορίες.

Δημιουργία προϊόντος. Η δημιουργία ενός προϊόντος μέσω της `createProduct()` δεν είναι ένα απλό HTTP POST request, επειδή το backend χρησιμοποιεί σχεσιακή βάση δεδομένων. Όταν ο διαχειριστής φτιάχνει ένα νέο καφέ με επιλογές για ζάχαρη και γάλα, το `ProductService` εκτελεί πολλαπλές κλήσεις στο REST API:

- Δημιουργεί το προϊόν και περιμένει τον server να του στείλει το `productId`.
- Δημιουργεί το `MenuItem`, ώστε το προϊόν να αποκτήσει κατηγορία και τιμή. Έπειτα περιμένει τον server να του στείλει το `menuItemId`.
- Για κάθε μία από τις ομάδες επιλογών (π.χ. Ζάχαρη, Γάλα) κάνει ξεχωριστά POST requests, περιμένοντας τα `itemOptionId` σαν απάντηση.
- Για κάθε μία από τις επιλογές (π.χ. Μέτριος, Σκέτος) κάνει ξεχωριστές κλήσεις και τις αποθηκεύει συνδέοντάς τες στην ομάδα που ανήκει η κάθε μία.

- Όλη αυτή η διαδικασία εκτελείται ασύγχρονα (Future) και αν οποιοδήποτε βήμα αποτύχει, το σύστημα το αντιλαμβάνεται στην try-catch, και επιστρέφει null ώστε το UI να δείξει το αντίστοιχο μήνυμα λάθους.

Διαδικασία Ενημέρωσης. Η διαδικασία τροποποίησης ενός υπάρχοντος προϊόντος είναι τεχνικά πιο δύσκολη από τη δημιουργία ενός νέου. Αν ο διαχειριστής αλλάξει τις επιλογές ενός προϊόντος, η μέθοδος `updateProduct()` πρώτα ενημερώνει τα βασικά στοιχεία. Και μετά για να είμαστε σίγουροι ότι δεν θα μείνουν λάθος ή παλιά δεδομένα στη βάση (π.χ. επιλογές που αφαιρέθηκαν), το service ζητάει όλες τις παλιές επιλογές που σχετίζονται με αυτό το προϊόν και τις διαγράφει μαζί με DELETE requests, προτού δημιουργήσει από την αρχή τις νέες επιλογές.

Ασφαλής Διαχείριση Τύπων Δεδομένων. Επειδή η Dart είναι "strongly typed", θα προκύψει σφάλμα αν λάβουμε διαφορετικό τύπο δεδομένων από αυτόν που περιμένουμε, το service έχει συναρτήσεις όπως η `_parseBool()` που διαβάζει τα δεδομένα και άσχετα με το αν το backend έστειλε την απάντηση, σε μορφή string ή με τη μορφή ακέραιου αριθμού 0 ή 1, μετατρέπει την τιμή σε boolean.

4. `OrderService` (`order_service.dart`) Διαχειρίζεται όλες τις διεργασίες που αφορούν παραγγελίες. Το χρησιμοποιεί η εφαρμογή εξυπηρέτησης για να στείλει μια νέα παραγγελία στην παραγωγή. Η παραγωγή για να ενημερώνεται συνεχώς για τις ενεργές παραγγελίες και να ειδοποιεί την εφαρμογή εξυπηρέτησης όταν ένα προϊόν είναι έτοιμο, αλλάζοντας την κατάσταση της παραγγελίας. Ο διαχειριστής το χρειάζεται για να βλέπει το ιστορικό.

Η λειτουργία αυτού του service αναλύεται στις παρακάτω λειτουργίες:

Ανάκτηση και Διαχωρισμός Παραγγελιών. Αφού το service λάβει τα δεδομένα αποκωδικοποιεί το JSON και δημιουργεί Dart αντικείμενα. Μετά χωρίζει τη λίστα με τις παραγγελίες, στα δύο, τις ενεργές παραγγελίες και τις παραγγελίες του ιστορικού. `activeOrders`: Οι τρέχουσες παραγγελίες που βρίσκονται σε εξέλιξη. `historyOrders`: Οι παραγγελίες που έχουν ήδη πληρωθεί και κλείσει. Το ιστορικό ταξινομείται πως και η πιο πρόσφατες παραγγελίες να εμφανίζονται στο πάνω μέρος της οθόνης. Αυτό διευκολύνει το προσωπικό στο να επιλύει τυχόν προβλήματα εξυπηρέτησης πελατών π.χ. λανθασμένες παραγγελίες.

Μηχανισμός Προσωρινής Μνήμης (Cache) Οι εφαρμογές εξυπηρέτησης και παραγωγής κάνουν "polling" (ελέγχουν για νέα δεδομένα κάθε 5 δευτερόλεπτα) για να εμφανίσουν τις νέες παραγγελίες. Αυτό θα μπορούσε να μας οδηγήσει σε πτώση της απόδοσης και κατανάλωση μπαταρίας στο κινητό, καθώς οι παραγγελίες μπορούν να είναι πολλές ακόμη και χιλιάδες. Για το λόγο αυτό, το service υλοποιεί έναν έξυπνο μηχανισμό προσωρινής μνήμης, χρησιμοποιώντας ένα λεξικό (`Map<String, Order> _orderCache`). Αν μια παραγγελία υπάρχει ήδη στην `_orderCache`, το σύστημα δεν την ξαναφτιάχνει. Απλώς συγκρίνει και ενημερώνει την κατάστασή της (π.χ. από Pending σε Ready), εξοικονομώντας πόρους συστήματος κάτι που είναι πάρα πολύ σημαντικό μιας και εδώ έχουμε να κάνουμε με κινητές συσκευές, είτε tablet είτε κινητά που καλούνται να μείνουν ανοιχτά για πάρα πολλές ώρες.

Αποστολή και Δημιουργία Παραγγελίας (`createOrder`) Η αποστολή μιας παραγγελίας είναι η αντίστροφη διαδικασία. Το service παίρνει το αντικείμενο από το ψηφιακό καλάθι του σερβιτόρου και το κωδικοποιεί σε ένα αντικείμενο JSON που είναι κατανοητό από το backend. Μετατρέπει την

κατάσταση της παραγγελίας σε ακέραιους αριθμούς (1 = Εκκρεμεί, 2 = Διαβάστηκε, 3 = Έτοιμη, 4 = Πληρώθηκε). Τέλος, ενσωματώνει για κάθε προϊόν τις προτιμήσεις και τις επιλογές των πελατών με τρόπο που είναι κατανοητό από το backend.

Διαχείριση Αλλαγών (updateOrder & replaceOrder) Ένα από τα μεγαλύτερα κίνητρα για να χρησιμοποιήσει μία επιχείρηση ψηφιακό σύστημα παραγγελιοληψίας, είναι η εύκολη διαχείριση αλλαγών στις παραγγελίες (π.χ. όταν ο πελάτης προσθέτει ένα υλικό λίγα δευτερόλεπτα αφού ο σερβιτόρος στείλει την παραγγελία). Το service έχει την μέθοδο replaceOrder(), που οποία λύνει το πρόβλημα εύκολα, διαγράφει την παλιά παραγγελία μέσω της deleteOrder, και καταχωρεί την τροποποιημένη ως μια ολοκαίνουρια παραγγελία. Αυτό σημαίνει ότι η παραγγελία θα έχει καινούργιο μοναδικό αναγνωριστικό παραγγελίας και κατάσταση της θα είναι ξανά σε εκκρεμότητα. Έτσι διασφαλίζουμε ότι το προσωπικό παραγωγής θα ενημερωθεί εκ νέου για την παραγγελία ώστε να μη χάσει την αλλαγή.

Ασφαλής Εξαγωγή Δεδομένων (Extraction Helpers) Λόγω της σχεσιακής βάσης δεδομένων του backend, η δομή του JSON μπορεί να διαφέρει ελαφρώς ανάλογα με το πώς αντλήθηκαν τα δεδομένα (π.χ. αν το όνομα του προϊόντος βρίσκεται στο κλειδί name, στο itemName, ή μέσα στο menuItem). Για να το λύσω αυτό, το αρχείο περιέχει τις συναρτήσεις (_extractItemName, _extractItemPrice, _extractSelectedOptions). Αυτές λειτουργούν ως δικλίδες ασφαλείας, ελέγχοντας όλες τις πιθανές δομές του JSON. Έτσι είμαι βέβαιος ότι η εφαρμογή θα βρει την σωστή τιμή, δεν θα κρασάρει, και δεν θα εμφανίσει ποτέ κενές διεπαφές στον χρήστη, ότι δεδομένα και λάβει.

5. AdminService (admin_service.dart) Χρησιμοποιείται αποκλειστικά από την εφαρμογή της διαχείρισης. Επικοινωνεί με το Admin Gateway του backend. Διαχειρίζεται την δημιουργία νέων λογαριασμών προσωπικού, τις αναθέσεις ρόλων, και τον έλεγχο των στοιχείων σύνδεσης(κωδικός χρήστη και όνομα χρήστη)

Οι κύριες λειτουργίες του είναι:

- Διαχείριση Προσωπικού. Το service παρέχει το πλήρες εύρος CRUD (Create, Read, Update, Delete) για τους λογαριασμούς των υπαλλήλων.
- Ασφάλεια και Έλεγχος Πρόσβασης (RBAC - Role Based Access Control) Κατά τη δημιουργία (createStaffUser) ή την επεξεργασία (updateStaffUser) ενός υπαλλήλου, το service διασφαλίζει ότι ο υπάλληλος έχει συγκεκριμένο ρόλο. Επιπλέον, η μέθοδος δημιουργίας κλειδώνει το πεδίο deleted στην τιμή false, αποτρέποντας την λανθασμένη καταχώρηση ενός "διαγραμμένου" υπαλλήλου εξ αρχής. Κάθε υπάλληλος μπορεί να συνδεθεί μόνο στην εφαρμογή που ταιριάζει στον ρόλο του, για παράδειγμα ένας σερβιτόρος μπορεί να συνδεθεί μόνο στην εφαρμογή εξυπηρέτησης.

6. SoundService (sound_service.dart) Σε αντίθεση με τα υπόλοιπα services εδώ εξυπηρετείται μόνο μία λειτουργία που είναι η αναπαραγωγή ήχου για την ειδοποίηση του προσωπικού.

Χρησιμοποιεί την βιβλιοθήκη audioplayers[5] για να παίξει μία ηχητική ειδοποίηση (bell.wav). Καλείται κυρίως από τις εφαρμογές παραγωγής και εξυπηρέτησης για να δώσει ένα άμεσο ηχητικό ερέθισμα στο προσωπικό, μόλις αλλάξει η κατάσταση μίας παραγγελίας, ή δημιουργηθεί μια νέα παραγγελία. Έτσι μπορεί να εργάζεται κανονικά, χωρίς να χρειάζεται συνεχώς να κοιτάει τις οθόνες.

4.4 Υλοποίηση Εφαρμογής Διαχείρισης

Η εφαρμογή διαχείρισης είναι το κέντρο ελέγχου της επιχείρησης. Απευθύνεται αποκλειστικά στους ιδιοκτήτες και τους υπεύθυνους καταστημάτων. Με την χρήση της εφαρμογής, η διοίκηση έχει την πλήρη εποπτεία της ροής των παραγγελιών. Έχει δικαιώματα προσθήκης, επεξεργασίας και διαγραφής (CRUD Operations) στον κατάλογο των προϊόντων και στο εγγεγραμμένο προσωπικό. Επίσης μπορεί να μαρκάρει προϊόντα ως μη διαθέσιμα.

4.4.1 Κεντρικός Ελεγκτής και Polling

Όπως και στα υπόλοιπα υποσυστήματα, η διαχείριση της κατάστασης υλοποιείται μέσω του BLoC/Cubit. Η κλάση HomeCubit αναλαμβάνει ρόλο κεντρικής διαχείρισης. Καθώς ένας υπεύθυνος χρειάζεται να βλέπει την τρέχουσα εικόνα του καταστήματος, ενσωματώθηκε και εδώ ένας μηχανισμός Auto-Polling.

Με τη χρήση του Timer.periodic της Dart [16], το σύστημα επικοινωνεί ασύγχρονα με το REST API κάθε 5 δευτερόλεπτα. Με αυτόν τον ρυθμό καλούνται οι loadMenuData() και loadOrders(), και ανανεώνουν τα δεδομένα. Έτσι το tablet ή το κινητό του manager, προβάλλει πάντα τη σωστή πληροφορία. Κατά το κλείσιμο του Cubit (μέθοδος close()), γίνεται ασφαλής ακύρωση του Timer (_pollTimer?.cancel()).

4.4.2 Δυναμική Διεπαφή και Πλοήγηση

Η εφαρμογή έχει σχεδιαστεί για οθόνες μικρού ή μεσαίου μεγέθους (tablet και κινητά). Η κεντρική οθόνη στο πάνω μέρος της, έχει τρεις καρτέλες. Μία για τον κατάλογο και τα προϊόντα, μία για το προσωπικό. Και τέλος μία για τις παραγγελίες.

Ένα από τα πιο ενδιαφέροντα χαρακτηριστικά της διεπαφής είναι το έξυπνο, δυναμικό κουμπί ενεργειών FloatingActionButton. Αντί να υπάρχουν στατικά κουμπιά διάσπαρτα στην οθόνη, το κουμπί αυτό ξέρει ποια καρτέλα είναι ανοιχτή μέσω του state.currentTabIndex. Ανάλογα με την καρτέλα που είναι ανοιχτή δείχνει το κατάλληλο εικονίδιο και εκτελεί την αντίστοιχη λειτουργία. Έτσι εξοικονομεί χώρο και προσφέρει εύκολη πλοήγηση.

4.4.3 Σύστημα Διαχείρισης Καταλόγου και Συντομογραφιών

Στην καρτέλα των προϊόντων, ο διαχειριστής έχει την πλήρη ευχέρεια να δημιουργήσει δυναμικά το μενού. Αυτό το σημείο της εφαρμογής είναι εξαιρετικά σημαντικό, καθώς ένα προϊόν (π.χ. Καφές Espresso) δεν διαθέτει απλώς ένα όνομα και μια τιμή, αλλά συνοδεύεται από δυναμικές Ομάδες Επιλογών (Option Groups), όπως "Ζάχαρη" ή "Είδος Γάλακτος".

Για την υλοποίηση της προσθήκης προϊόντος, το πλαίσιο διαλόγου (_showProductDialog) περιβάλλεται από ένα StatefulWidget. Έχει δηλαδή τη δυνατότητα να διατηρεί την τοπική κατάσταση των δεδομένων, του ξεχωριστά από την οθόνη του Cubit. Με αυτό τον τρόπο μειώνουμε τα rebuild της κεντρικής οθόνης.

Το πιο καινοτόμο στοιχείο αυτού του διαλόγου, είναι η καταχώρηση των Συντομογραφιών (Abbreviations). Μέσα στη φόρμα δημιουργίας του προϊόντος, ο διαχειριστής μπορεί να ορίσει (μέσω του abbrevController) τον μοναδικό κωδικό που αντιστοιχεί στο εκάστοτε προϊόν (π.χ. "FRED" για Freddo Espresso).

Η διαδικασία αυτή δεν περιορίζεται μόνο στο βασικό προϊόν, αλλά περιλαμβάνει και τις επιλογές. Ο διαχειριστής, όταν δημιουργεί μια ομάδα επιλογών (π.χ. Ζάχαρη), καταχωρεί ξεχωριστές

συντομογραφίες για κάθε πιθανή επιλογή (π.χ. "SK" για Σκέτο, "ME" για Μέτριο, "OL" για με Ολίγη κλπ). Φυσικά καταχωρείται μία συντομογραφία και την ίδια την επιλογή (π.χ. Z για ποσότητα ζάχαρης). Αυτά τα δεδομένα, μετατρέπονται από τη βιβλιοθήκη `built_value` σε ένα αντικείμενο `Product` και γράφονται στη βάση δεδομένων.

4.4.3 Σύστημα Διαχείρισης Προσωπικού

Το σύστημα επιτρέπει την πρόσβαση με βάση τον ρόλο του κάθε χρήστη. Στην καρτέλα `Staff`, ο διαχειριστής βλέπει τη λίστα με τους υπαλλήλους (`StaffUser`). Κατά τη δημιουργία νέου χρήστη (`_showAddUserDialog`), ο διαχειριστής συμπληρώνει τα στοιχεία επικοινωνίας και αναθέτει έναν ρόλο στον υπάλληλο.

Μετά την δημιουργία ενός νέου χρήστη στο `backend`, ενεργοποιείται η μέθοδος `_showCredentialsPopup()`. Εμφανίζεται ένα αναδυόμενο παράθυρο, το οποίο δεν κλείνει τυχαία εάν ο χρήστης πατήσει έξω από αυτό. Κλείνει μόνο αν πατήσει το κουμπί. Το παράθυρο αυτό προβάλλει κεντρικά το `Username` και το `Password` του νέου υπαλλήλου. Ενημερώνει τον υπεύθυνο ότι πρέπει να δώσει αυτά τα στοιχεία στον υπάλληλο. Ο διαχειριστής έχει την δυνατότητα να απενεργοποιεί προσωρινά λογαριασμούς υπαλλήλων. Σε επιχειρήσεις εστίασης είναι πολύ συχνό οι εργαζόμενοι να αποχωρούν για λίγο και να επιστρέφουν αργότερα ή να εργάζονται εποχιακά.

4.4.4 Εποπτεία Παραγγελιών

Τέλος, η εφαρμογή διαχείρισης δίνει τη δυνατότητα στον υπεύθυνο να ενημερώνεται για τις παραγγελίες. Η καρτέλα των παραγγελιών χωρίζεται και αυτή σε δύο μέρη. Το πρώτο δείχνει το ιστορικό που είναι η παραγγελίες που έχουν ήδη ολοκληρωθεί και το άλλο οι παραγγελίες σε εξέλιξη.

4.5 Υλοποίηση Εφαρμογής Εξυπηρέτησης

Η εφαρμογή του σερβιτόρου αποτελεί το πιο νευραλγικό σημείο επαφής του προσωπικού και της επιχείρησης με το σύστημα. Σχεδιάστηκε με στόχο την ευκολία, την ταχύτητα και την ελαχιστοποίηση των λαθών, κατά τη λήψη των παραγγελιών. Η υλοποίηση βασίζεται στο `framework Flutter` [12] της Google, το οποίο επιτρέπει τη δημιουργία γρήγορων, `native`, `cross platform` εφαρμογών.

4.5.1 Αρχιτεκτονική και Διαχείριση Κατάστασης (State Management)

Για τη διαχείριση της κατάστασης της εφαρμογής (`State Management`), απέφυγα την χρήση του κλασικού `setState()`, καθώς σε πολύπλοκες εφαρμογές οδηγεί σε κώδικα που είναι δύσκολο να συντηρηθεί και προκαλεί περιττά `rebuidls` της διεπαφής. Επέλεξα την σύγχρονη αρχιτεκτονική `BLoC` (`Business Logic Component`) μέσω της βιβλιοθήκης `flutter_bloc` [13], και πιο συγκεκριμένα το πρότυπο `Cubit`.

Κάθε οθόνη έχει το δικό της `Cubit` (π.χ. `HomeCubit`, `AllOrdersCubit`). Το `Cubit` μέσω του `ServiceRepository` επικοινωνεί με το `REST API`, λαμβάνει και επεξεργάζεται τα δεδομένα, και μετά εκπέμπει (`emit`) μια νέα, κατάσταση (`State`). Η διεπαφή χρήστη βλέπει αυτές τις αλλαγές μέσω του `widget BlocBuilder` και ανανεώνει δυναμικά μόνο τα σημεία της οθόνης που πραγματικά άλλαξαν. Με αυτόν τον διαχωρισμό, η οπτική αναπαράσταση της εφαρμογής (`UI`) είναι εντελώς αποκομμένη από την επιχειρησιακή λογική.

4.5.2 Διεπαφή Χρήστη και Πλοήγηση

Για την πλοήγηση μεταξύ των οθονών επιλέχθηκε η βιβλιοθήκη `go_router` [14], η οποία προσφέρει δρομολόγηση βασισμένη σε URLs. Η κύρια οθόνη του σερβιτόρου (Home Page) αποτελείται από δύο βασικά τμήματα:

- Οριζόντια Λίστα Κατηγοριών: Μια μπάρα (slider) στην κορυφή της οθόνης που επιτρέπει την γρήγορη προσπέλαση του καταλόγου. Οι απενεργοποιημένες κατηγορίες εμφανίζονται αυτόματα γκριζαρισμένες προειδοποιώντας τον χρήστη.
- Λίστα Προϊόντων: Ο χώρος όπου εμφανίζονται τα προϊόντα της επιλεγμένης κατηγορίας.

Όταν ο σερβιτόρος προσθέτει προϊόντα από εδώ, αυτά αποθηκεύονται προσωρινά στη μνήμη δημιουργώντας ουσιαστικά το καλάθι της παραγγελίας.

4.5.3 Παραμετροποίηση Προϊόντων

Η διαδικασία επιλογής και παραμετροποίησης ενός προϊόντος (π.χ. προσθήκη ζάχαρης, γάλακτος, επιλογή μεγέθους), είναι βασική λειτουργία της εφαρμογής, και είναι απαραίτητη σε κάθε παραγγελία. Για να επιλέξει και ένα παραμετροποιήσει προϊόντα ο σερβιτόρος δεν οδηγείται σε νέες οθόνες, αντί αυτού το σύστημα εμφανίζει παράθυρα διαλόγου στο το κάτω μέρος της οθόνης, καθιστώντας εύκολο το χειρισμό της εφαρμογής με το ένα χέρι.

Σε αυτά τα widgets, επειδή η πληροφορία είναι καθαρά τοπική (π.χ. ποιό είδος ζάχαρης επιλέχθηκε) και δεν αφορά την υπόλοιπη εφαρμογή, δεν κρίθηκε απαραίτητη η δημιουργία ξεχωριστού Cubit. Επέλεξα την βιβλιοθήκη `flutter_hooks` [15]. Τα Hooks μάς επιτρέπουν να διαχειριζόμαστε την τοπική κατάσταση των widgets π.χ. ποσότητα, με ελάχιστο και καθαρό κώδικα, χωρίς να χρειάζεται να μετατρέψουμε το widget σε `StatefulWidget`. Μόλις ο σερβιτόρος ολοκληρώσει τις επιλογές του, το τελικό προϊόν (μαζί με τις επιλογές και την τελική τιμή) μπαίνει στο κεντρικό καλάθι.

4.5.4 Συντακτικός Αναλυτής Συντομογραφιών

Η μετάβαση από τον παραδοσιακό τρόπο εστίασης (μπλοκάκι και στυλό) στα ψηφιακά συστήματα συχνά αποτυγχάνει λόγω της αντίστασης του έμπειρου προσωπικού. Ένας παλιός σερβιτόρος έχει εκπαιδευτεί να καταγράφει μια πολύπλοκη παραγγελία στο μπλοκάκι του γράφοντας απλώς συντομογραφίες όπως για παράδειγμα ένα νούμερο σε κύκλο για τον αριθμό του τραπεζιού και κωδικούς όπως `fred.` σκ. για τα προϊόντα. Όσο καλά σχεδιασμένο κι αν είναι ένα ψηφιακό μενού, το να αναγκαστεί ο σερβιτόρος να ψάξει σε κατηγορίες και να πατήσει πέντε διαφορετικά κουμπιά στην οθόνη του κινητού, κατά τη διάρκεια της ώρας αιχμής, δημιουργεί καθυστερήσεις και εκνευρισμό. Ειδικά σε περιπτώσεις όπου στην ίδια παρέα οι πελάτες επιθυμούν προϊόντα από διαφορετικές κατηγορίες το μενού υστερεί σημαντικά. Αυτό αναγκάζει τον υπάλληλο να επιλέξει κατηγορία, μετά να ψάχνει το προϊόν που του ζητήθηκε να το παραμετροποιεί και ύστερα να βγαίνει και να επαναλαμβάνει την διαδικασία για κάθε προϊόν διαφορετικής κατηγορίας.

Σε αυτό το θεμελιώδες πρόβλημα εμπειρίας χρήστη έρχεται να μας δώσει τη λύση το πιο καινοτόμο χαρακτηριστικό του συστήματος `OrderFlow`: ο Συντακτικός Αναλυτής Συντομογραφιών. Στόχος αυτού του μηχανισμού είναι να γεφυρώσει το χάσμα, επιτρέποντας στους έμπειρους χρήστες να διατηρήσουν την ταχύτητα και τη νοητική τους ρουτίνα γράφοντας με γραφίδα ή το δάχτυλο τους παλιούς κωδικούς

του καταστήματος, ενώ το σύστημα αναλαμβάνει να μεταφράσει το χειρόγραφο σε δομημένα δεδομένα και στην συνέχεια να αντιστοιχίσει αυτά τα δεδομένα σε υπαρκτά προϊόντα.

Η πολυπλοκότητα αυτού του προβλήματος έγκειται στο γεγονός ότι ο κατάλογος είναι δυναμικός (αντλείται από τη βάση δεδομένων). Αυτό σημαίνει ότι η κάθε επιχείρηση μπορεί να καταχωρήσει ότι προϊόντα θέλει στον κατάλογο. Και παράλληλα η κάθε επιχείρηση έχει τις δικές της κωδικοποιημένες συντομογραφίες που αντιστοιχούν στα προϊόντα της. Συνεπώς, δεν ήταν δυνατή η χρήση στατικών κανονικών εκφράσεων (hardcoded) για την εύρεση των προϊόντων. Το σύστημα σχεδιάστηκε ώστε να αναλύει το κείμενο δυναμικά συγκρίνοντάς το με τα εκάστοτε διαθέσιμα προϊόντα και τις συντομογραφίες που έχει ορίσει ο διαχειριστής. Ουσιαστικά δίνουμε τη δυνατότητα σε κάθε κατάστημα να ορίζει συντομογραφίες για κάθε προϊόν, και για κάθε μια επιλογή του προϊόντος.

Για να μπορέσει το σύστημα να αντιληφθεί και να επεξεργαστεί τις χειρόγραφες συντομογραφίες, πρέπει πρώτα να μπορεί να διαβάσει. Με την τεχνολογία Αναγνώρισης Ψηφιακής Μελάνης (Digital Ink Recognition) του ενσωματωμένου μοντέλου μηχανικής μάθησης Google ML Kit [10]. Η διαδικασία ξεκινά όταν ο υπάλληλος σχεδιάζει τους χαρακτήρες της παραγγελίας πάνω σε έναν ειδικό ψηφιακό καμβά στην οθόνη της εφαρμογής, χρησιμοποιώντας είτε μια γραφίδα είτε απλώς το δάχτυλό του. Το μοντέλο του Google ML Kit αναλαμβάνει στο παρασκήνιο να αναλύσει αυτή την πληροφορία και να αποκρυπτογραφήσει τα σχήματα, μετατρέποντας το "ψηφιακό μελάνι" σε ψηφιακό κείμενο. Αυτή η παραγόμενη συμβολοσειρά είναι ουσιαστικά η ακατέργαστη ψηφιακή αναπαράσταση του χειρογράφου, η οποία αμέσως μετά τροφοδοτείται στον αλγόριθμο συντακτικής ανάλυσης ώστε να ταυτοποιηθούν τα προϊόντα και οι επιλογές τους.

4.5.5 Λογική και Αλγόριθμος Ανάλυσης

Ο σερβιτόρος μπορεί να γράψει “χειρόγραφα” μία παραγγελία. Ο αλγόριθμος του Parser αναλαμβάνει την πλήρη αποκωδικοποίηση της παραγγελίας μέσω ενός επαναληπτικού βρόχου (while-loop) και διαδοχικών σταδίων:

- Εξαγωγή Αναγνωριστικού Τραπεζιού: Μέσω της έκφρασης `RegExp(r'ts*(\d+)')`, ο αναλυτής σαρώνει το κείμενο για να εντοπίσει το γράμμα 't' ακολουθούμενο από αριθμό. Ανάμεσα τους μπορεί να υπάρχουν 0 ή περισσότερα κενά. Ο αριθμός μπορεί να έχει ένα ψηφίο ή πολλά. Μόλις εντοπιστεί, ο αριθμός του τραπεζιού αποθηκεύεται στη μεταβλητή `tableNumber` και το αντίστοιχο κομμάτι κειμένου αφαιρείται από το αρχικό κείμενο, αποτρέποντας έτσι τυχόν σύγχυση κατά την επακόλουθη αναζήτηση των προϊόντων.
- Καθαρισμός Δεδομένων: Ο κώδικας αφαιρεί όλα τα κενά, τις τελείες, τα κόμματα και τις αλλαγές γραμμής με την κανονική έκφραση `RegExp(r'[\s\.,\n]')` και μετατρέπει τους χαρακτήρες σε πεζά. Αυτό το βήμα είναι κρίσιμο, καθώς διασφαλίζει ότι τυπογραφικά λάθη ή διαφορετικά στυλ γραφής του προσωπικού (π.χ. "FR.ZSK", "fredzmt" ή "fredcap zol"), θα μετατραπούν σε μια κοινή, ομοιόμορφη συμβολοσειρά.
- Ταύτιση Προϊόντων: Ο αλγόριθμος μπαίνει στη μέθοδο `_parseNextItem`. Για την εύρεση του προϊόντος, συγκρίνει διαδοχικά την αρχή του εναπομείναντος κειμένου (χρησιμοποιώντας τη μέθοδο `startsWith`) με τις καταχωρημένες συντομογραφίες (abbreviations) όλων των προϊόντων. Εδώ υλοποιείται η τεχνική Greedy Match: ο αλγόριθμος καταγράφει όλες τις πιθανές ταυτίσεις, αλλά επιλέγει τελικά εκείνη με το μεγαλύτερο μήκος χαρακτήρων. Έτσι, αποτρέπονται τα σφάλματα (π.χ. αν υπάρχει συντομογραφία "FR" για τον Φραπέ και "FRED"

για Freddo Espresso, το σύστημα θα αντιστοιχίσει το σωστό προϊόν χωρίς να επιλέξει την πρώτη συντόμευση που θα βρει).

- **Δυναμική Εξαγωγή Επιλογών :** Αφού επιλεγεί το σωστό προϊόν, ο αλγόριθμος διαβάσει δυναμικά τις ομάδες επιλογών (π.χ. Ζάχαρη, Γάλα) που συνδέονται αποκλειστικά με αυτό. Ο αλγόριθμος δεν αναζητά απλώς εάν η συντομογραφία μιας επιλογής (π.χ. "ολ" για με Ολίγη) υπάρχει κάπου ελεύθερα μέσα στο κείμενο. Αντίθετα, ελέγχει αυστηρά μέσω της μεθόδου `startsWith()`, αν το κείμενο που έχει περισσέψει ξεκινάει με την συντομογραφία της συγκεκριμένης επιλογής. Η κατεύθυνση ανάγνωσης (από αριστερά προς τα δεξιά) διασφαλίζει ότι δεν θα διαβάσει κατά λάθος υλικά που προορίζονταν για άλλο προϊόν. Εφόσον υπάρξει ταύτιση, δημιουργείται το αντικείμενο `SelectedOption` και τέλος, οι χαρακτήρες που χρησιμοποιήσαμε για να ταυτοποιήσουμε το προϊόν και τις επιλογές του, κόβονται από το κείμενο (μέσω της συνάρτησης `substring()`).
- **Εξαγωγή Ποσοτήτων:** Στο τελευταίο βήμα ο αλγόριθμος αναζητά τον πολλαπλασιαστή ποσότητας μέσω του προθέματος `x` (όπως `x2`). Αν το κείμενο που έμεινε ξεκινάει με `x`, ακολουθούμενο από αριθμό, τότε καταγράφεται η ποσότητα και οι συγκεκριμένοι χαρακτήρες αφαιρούνται από την συμβολοσειρά της παραγγελίας.
- **Διαχείριση Σφαλμάτων:** Σε περίπτωση που ο αλγόριθμος συναντήσει χαρακτήρες που δεν αντιστοιχούν πουθενά, διαθέτει μηχανισμό ασφαλείας: κόβει τον πρώτο ακατανόητο χαρακτήρα (`cleanInput.substring(1)`) και συνεχίζει, αποτρέποντας την πρόκληση ατέρμονου βρόχου (`infinite loop`) και την κατάρρευση της εφαρμογής.

Ο Συντακτικός Αναλυτής Συντομογραφιών αποτελεί εξαιρετικό παράδειγμα του πώς οι τεχνικές της Επιστήμης Υπολογιστών μπορούν να επιλύσουν πρακτικά, καθημερινά προβλήματα, καθιστώντας την ψηφιακή μετάβαση της επιχείρησης πολύ πιο ομαλή για το προσωπικό.

4.5.6 4.5.5 Διαχείριση Ενεργών Παραγγελιών και Ταμεία

Πέρα από την καταχώρηση νέων παραγγελιών, ο σερβιτόρος έχει πρόσβαση στην οθόνη όλων των παραγγελιών (`AllOrdersPage`). Μέσω του `AllOrdersCubit`, η εφαρμογή ανακτά τις παραγγελίες και τις χωρίζει σε δύο καρτέλες: τις ενεργές παραγγελίες και το ιστορικό.

Μια πολύ σημαντική λειτουργία αυτού του υποσυστήματος είναι η "Έκδοση Απόδειξης" (`receipt_bottom_sheet.dart`). Όταν μια παρέα ζητήσει τον λογαριασμό, η εφαρμογή ομαδοποιεί αυτόματα όλες τις ξεχωριστές παραγγελίες που έχουν εκκρεμότητα και αντιστοιχούν στο συγκεκριμένο τραπέζι. Υπολογίζει το τελικό συνολικό ποσό και παρέχει το κουμπί "Σήμανση ως Πληρωμένο", το οποίο ενημερώνει το backend, κλείνει τον κύκλο ζωής της παραγγελίας, και την μεταφέρει στο ιστορικό.

4.6 Υλοποίηση Εφαρμογής Παραγωγής

Η εφαρμογή παραγωγής απευθύνεται αποκλειστικά στο προσωπικό παρασκευής του καταστήματος (όπως οι `barista`, οι `μπάρμαν` και οι `μάγειρες`). Οργανώνει την εκτέλεση των παραγγελιών της επιχείρησης. Κύριος στόχος της εφαρμογής είναι η άμεση ειδοποίηση του προσωπικού για τις νέες παραγγελίες και η ενημέρωση του σταδίου προετοιμασίας τους σε πραγματικό χρόνο. Παρέχει επίσης διαχείριση της διαθεσιμότητας του καταλόγου σε περιπτώσεις έλλειψης αποθεμάτων. Όπως και η

εφαρμογή του σερβιτόρου, είναι αναπτυγμένη με βάση το framework Flutter, αξιοποιώντας στο έπακρο τις δυνατότητες του State Management.

4.6.1 Αυτόματη Ανανέωση Δεδομένων

Το εργασιακό περιβάλλον μιας κουζίνας χαρακτηρίζεται από γρήγορους ρυθμούς. Πράγμα που σημαίνει ότι το προσωπικό συχνά δεν έχει την δυνατότητα ή τον χρόνο, να αλληλεπιδρά συνεχώς με μια οθόνη για να ελέγξει αν ήρθε νέα παραγγελία. Οπότε η εφαρμογή σχεδιάστηκε με την δυνατότητα για συνεχή, αυτόματο συγχρονισμό δεδομένων με το backend, χωρίς καμία παρέμβαση ή ενέργεια από τον χρήστη.

Για τη διαχείριση της κατάστασης της κεντρικής οθόνης, υλοποιήθηκε η κλάση HomeCubit. Μέσα στο Cubit [13], δημιουργήθηκε ένας μηχανισμός polling. Χρησιμοποιώντας την κλάση Timer.periodic [16] της γλώσσας Dart, το σύστημα εκτελεί ασύγχρονα ερωτήματα στο REST API σε τακτά χρονικά διαστήματα (κάθε 5 δευτερόλεπτα), ζητώντας τα πιο πρόσφατα δεδομένα.

Ο αλγόριθμος του συστήματος συγκρίνει τα μοναδικά αναγνωριστικά (IDs) των νέων παραγγελιών που λαμβάνει, με ένα τοπικό σύνολο ήδη γνωστών παραγγελιών (_knownOrderIds). Εάν βρεθεί καινούργια παραγγελία, η εφαρμογή ενεργοποιεί αυτόματα μια δυνατή ηχητική ειδοποίηση μέσω του ηχείου της συσκευής. Έτσι εξασφαλίζεται η άμεση κινητοποίηση του προσωπικού, ανεξάρτητα από το αν κοιτάνε την οθόνη. Για την προστασία του συστήματος από διαρροές μνήμης (memory leaks), κατά το κλείσιμο της οθόνης το timer ακυρώνεται αυτόματα.

4.6.2 Διεπαφή Χρήστη Παραγωγής

Το ιδανικό θα ήταν η εφαρμογή παραγωγής να τρέχει σε tablet. Για να αξιοποιήσω μέγιστα τον διαθέσιμο χώρο στην οθόνη, οργάνωσα τα δεδομένα σε τρεις καρτέλες. Το σύστημα αυτό υλοποιείται μέσω του DefaultTabController, επιτρέποντας την εύκολη εναλλαγή των δεδομένων χωρίς την φόρτωση νέων σελίδων. Οι καρτέλες είναι οι παρακάτω:

Ενεργές Παραγγελίες: Η βασική οθόνη της παραγωγής. Εδώ φαίνονται οι παραγγελίες που εκκρεμούν, ταξινομημένες χρονολογικά. Επίσης η διεπαφή υποστηρίζει "Pull to Refresh" (σύρσιμο προς τα κάτω) για άμεση, χειροκίνητη ανανέωση δεδομένων. Αυτό μας είναι χρήσιμο αν το προσωπικό δεν θέλει να περιμένει τον επόμενο κύκλο του polling ή συμβεί κάποια αστοχία στο δίκτυο.

Ιστορικό: Δείχνει όλες τις παραγγελίες που έχουν ήδη ετοιμαστεί ή πληρωθεί. Ο αλγόριθμος δημιουργεί αυτή τη λίστα συνενώνοντας (merge) τις ενεργές παραγγελίες που μόλις σημάνθηκαν ως έτοιμες, μαζί με το ιστορικό του backend. Η καρτέλα αυτή είναι χρήσιμη για την επαλήθευση λαθών από τους μάγειρες.

Διαχείριση Διαθεσιμότητας: Προβάλλει ολόκληρο τον κατάλογο, όμως η μόνη ενέργεια που μπορεί να κάνει το προσωπικό είναι η ενεργοποίηση και η απενεργοποίηση του προϊόντος. Ομαδοποιεί τα προϊόντα ανά κατηγορία, επιτρέποντας τη γρήγορη εύρεση και απενεργοποίηση μεμονωμένων προϊόντων (ή και ολόκληρων κατηγοριών), εφόσον υπάρξει έλλειψη σε κάποια πρώτη ύλη.

4.6.3 Κύκλος Ζωής της Παραγγελίας στην Παραγωγή

Η αλληλεπίδραση του προσωπικού παραγωγής με την παραγγελία, γίνεται από το widget OrderCard. Κάθε κάρτα διαθέτει κουμπιά για την αλλαγή της εσωτερικής κατάστασης της παραγγελίας. Ο κύκλος ζωής υλοποιείται σε δύο διακριτά στάδια εξέλιξης, τα οποία τροποποιούν αντίστοιχα πεδία στη βάση δεδομένων:

Σήμανση ως Αναγνωσμένη: Με το πάτημα αυτού του κουμπιού, η εφαρμογή ενημερώνει το backend αλλάζοντας το πεδίο `isSeen` σε αληθές (`true`). Η χρήση αυτής της λειτουργίας είναι προαιρετική καθώς μία παραγγελία μπορεί να καταγραφεί ως έτοιμη ή ακόμα και πληρωμένη, χωρίς ποτέ να πατηθεί αυτό το κουμπί.

Σήμανση ως Έτοιμη: Καταγράφει την παραγγελία ως ολοκληρωμένη από πλευράς παραγωγής. Η εφαρμογή αφαιρεί την παραγγελία από την καρτέλα των ενεργών παραγγελιών, εφαρμόζοντας ένα φίλτρο, και την μεταφέρει στην αρχή του τοπικού ιστορικού.

4.6.4 Διαχείριση Αποθεμάτων με Αισιόδοξη Ενημέρωση

Μία από τις πιο κρίσιμες και ευαίσθητες στον χρόνο λειτουργίες της εφαρμογής, είναι η δυνατότητα του προσωπικού να επισημάνει άμεσα ένα προϊόν ως "Μη Διαθέσιμο". Μέσω ενός διακόπτη το προϊόν σταματάει να είναι διαθέσιμο στους σερβιτόρους. Για να διασφαλιστεί η ομαλή εμπειρία χρήσης, υλοποιήθηκε το σχεδιαστικό μοτίβο της αισιόδοξης ενημέρωσης (Optimistic UI Update).

Βάσει αυτής της λογικής, όταν ο χρήστης πατήσει τον διακόπτη διαθεσιμότητας, η εφαρμογή εντοπίζει το προϊόν (με την χρήση της `firstWhere`), δημιουργεί ένα νέο αμετάβλητο αντικείμενο με αντίθετη κατάσταση διαθεσιμότητας. Μετά ενημερώνει την τοπική κατάσταση (`state`) του `Cubit`. Η οθόνη δείχνει πλέον το προϊόν ως "Μη διαθέσιμο", χωρίς να περιμένει την οριστική απάντηση του backend.

Στο παρασκήνιο, η εφαρμογή αποστέλλει το αντίστοιχο HTTP Request. Εάν για οποιονδήποτε λόγο η αποθήκευση στη βάση δεδομένων αποτύχει, ο αλγόριθμος μπαίνει σε διαδικασία χειρισμού σφαλμάτων. Εκεί αναστρέφει την τοπική κατάσταση στην αρχική της μορφή ακυρώνοντας την "αισιόδοξη" ενημέρωση. Τέλος ενημερώνει τον χρήστη για το σφάλμα μέσω ενός αναδυόμενου μηνύματος (`SnackBar`). Αυτό το μοτίβο εξαλείφει τις οπτικές καθυστερήσεις κατά την πλοήγηση και προσδίδει στην εφαρμογή μια αίσθηση ταχύτητας και αξιοπιστίας.

4.7 Επίλογος

Σε αυτό το σημείο θέλω να συνοψίσω την τεχνική υλοποίηση του συστήματος παραγγελιοληψίας. Παρατηρούμε πως η επιλογή δημιουργίας τριών ξεχωριστών εφαρμογών απέδωσε άριστα. Η βιομηχανία της εστίασης χαρακτηρίζεται από τεράστια πίεση χρόνου και μικρή ανοχή στα λάθη κατά τις ώρες αιχμής. Το σύστημα που υλοποιήθηκε λύνει πρακτικά αυτά τα προβλήματα, εξασφαλίζοντας ότι η πληροφορία της παραγγελίας μεταφέρεται γρήγορα και αξιόπιστα από την οθόνη του σερβιτόρου, στην οθόνη της κουζίνας, χωρίς καθυστερήσεις.

Αρχικά, η παραγγελία δημιουργείται στην εφαρμογή του σερβιτόρου, όπου η εισαγωγή των δεδομένων έχει επιταχυνθεί δραματικά χάρη στον πρωτοποριακό Συντακτικό Αναλυτή Συντομογραφιών. Αντί ο υπάλληλος να χάνει χρόνο μπεινοβγαίνοντας σε ατελείωτα μενού, περνάει στο σύστημα τις παραγγελίες του με ελάχιστο κόπο.

Σε δεύτερο χρόνο, η παραγγελία μεταδίδεται ακαριαία στην εφαρμογή της παραγωγής. Εκεί, ο μηχανισμός `rolling` εξασφαλίζει ότι καμία παραγγελία δεν θα χαθεί, ενώ η οπτική και ηχητική ειδοποίηση βοηθά στην άμεση εκτέλεσή της.

Τέλος, σε όλη τη διάρκεια της διαδικασίας, η εφαρμογή της διαχείρισης λειτουργεί ως επόπτης του συστήματος, προσφέροντας στον υπεύθυνο πλήρη εποπτεία. Ο διαχειριστής παρακολουθεί την ροή σε πραγματικό χρόνο και επεμβαίνει στον κατάλογο. Έχει φυσικά την ελευθερία να γράφει το λεξιλόγιο του Συντακτικού Αναλυτή Συντομογραφιών. Με αυτό τον τρόπο κάθε επιχείρηση μπορεί να έχει το δικό της κώδικα επικοινωνίας.

Εν κατακλείδι, ο συνδυασμός του BLoC/Cubit για τη διαχείριση της κατάστασης με τα σύγχρονα χαρακτηριστικά της γλώσσας Dart, συνέβαλε στη δημιουργία ενός οικοσυστήματος που δεν είναι απλώς λειτουργικό, αλλά είναι εύκολο στην συντήρηση, επεκτάσιμο (scalable) και φιλικό προς τον τελικό χρήστη.

Κεφάλαιο 5ο: Συμπεράσματα και Προτάσεις Βελτίωσης

5.1 Συνοπτική Αξιολόγηση του Έργου

Η παρούσα πτυχιακή εργασία είχε ως πρωταρχικό στόχο τον σχεδιασμό και την ανάπτυξη ενός ολοκληρωμένου, σύγχρονου συστήματος παραγγελιοληψίας. Ενός συστήματος ικανού να ανταπεξέλθει στις υψηλές απαιτήσεις του κλάδου της εστίασης. Κοιτώντας αναδρομικά την πορεία του έργου, ο στόχος αυτός επιτεύχθηκε στο ακέραιο.

Η δομή του Monopero επέτρεψε τον διαχωρισμό των αρμοδιοτήτων, και επιτάχυνε τη συγγραφή του κώδικα σημαντικά. Παράλληλα, το backend σε Java Spring Boot απέδειξε την ικανότητά του να διαχειρίζεται τα δεδομένα με ασφάλεια και ταχύτητα. Καινοτομίες όπως ο αλγόριθμος ανάλυσης συντομογραφιών μετέτρεψαν το OrderFlow από ένα απλό ακαδημαϊκό project σε ένα εργαλείο με την δυνατότητα να λειτουργήσει σε πραγματικές συνθήκες.

5.2 Προβλήματα που αντιμετωπίστηκαν

Η υλοποίηση ενός πλήρους συστήματος (Full-Stack) από το μηδέν συνοδεύτηκε από σημαντικές τεχνικές προκλήσεις, οι οποίες απαιτούσαν προσεκτικό σχεδιασμό και συνεχή έρευνα.

Στα αρχικά στάδια της ανάπτυξης, η διαχείριση της πληροφορίας (π.χ. το καλάθι του σερβιτόρου) αποδείχθηκε δύσκολη με απλές μεθόδους. Η υιοθέτηση του προτύπου BLoC/Cubit είχε μια αρχική δυσκολία εκμάθησης, αλλά τελικά έλυσε τα προβλήματα συγχρονισμού των οθονών.

Μια από τις σημαντικότερες τεχνικές προκλήσεις κατά την ανάπτυξη του συστήματος αφορούσε την ενσωμάτωση του συντακτικού αναλυτή συντομογραφιών. Καθώς τα προεκπαιδευμένα μοντέλα μηχανικής μάθησης είναι βελτιστοποιημένα να αναγνωρίζουν υπαρκτές λέξεις, δυσκολεύονταν σημαντικά στην ερμηνεία των συντομογραφιών του καταλόγου. Επιπρόσθετα, η απόπειρα χρήσης ειδικών συμβόλων ως διαχωριστικά, όπως το κόμμα (,) ή το σύμβολο της πρόσθεσης (+), αποδείχθηκε κακή ιδέα. Ο αλγόριθμος συχνά τα παρερμήνευε ως γράμματα του αλφαβήτου (π.χ. αναγνώριση του + ως T). Για την οριστική αντιμετώπιση του ζητήματος, σχεδιάστηκε και υλοποιήθηκε ένας αλγόριθμος συντακτικής ανάλυσης. Ο αλγόριθμος αυτός εξαλείφει την ανάγκη ύπαρξης διαχωριστικών ή κενών, σαρώνοντας την ακατέργαστη συμβολοσειρά σειριακά (από αριστερά προς τα δεξιά). Στην συνέχεια εξάγει δυναμικά τα προϊόντα, τις επιλογές και τις ποσότητές τους βάσει του διαθέσιμου καταλόγου. Η προσέγγιση αυτή κατέστησε την καταχώρηση πολλαπλών ειδών ανθεκτική στα λάθη του μοντέλου, προσφέροντας μια ευχάριστη και απρόσκοπτη εμπειρία στον χρήστη.

5.3 Προτάσεις για Μελλοντικές Επεκτάσεις

Παρότι το σύστημα OrderFlow είναι πλήρως λειτουργικό, η τεχνολογική εξέλιξη αφήνει πάντα περιθώρια για περαιτέρω βελτιώσεις και προσθήκες νέων χαρακτηριστικών. Μερικές προτάσεις για τη μελλοντική επέκταση του έργου είναι οι εξής:

Υποστήριξη Λειτουργίας Χωρίς Δίκτυο: Τα καταστήματα εστίασης συχνά αντιμετωπίζουν προβλήματα με το Wi-Fi. Μια μελλοντική έκδοση θα μπορούσε να αποθηκεύει προσωρινά τις παραγγελίες σε μια τοπική βάση SQLite στη συσκευή του σερβιτόρου, και να τις συγχρονίζει αυτόματα με το Backend μόλις επανέλθει η σύνδεση.

Ελαχιστοποίηση κατανάλωσης ενέργειας: Επί του παρόντος, η εφαρμογή της παραγωγής βασίζεται στο polling. Η αντικατάσταση αυτού του μηχανισμού με WebSockets, θα έστελνε τις παραγγελίες στην κουζίνα ακαριαία, μειώνοντας δραστικά τον φόρτο και την κατανάλωση μπαταρίας στα tablet.

Στατιστικά και Ανάλυση Δεδομένων : Η εφαρμογή του διαχειριστή μπορεί να εμπλουτιστεί με γραφήματα που θα παρουσιάζουν αναλυτικά στοιχεία, όπως τα προϊόντα με τις περισσότερες πωλήσεις και τις ώρες αιχμής του καταστήματος.

Διασύνδεση με Θερμικούς Εκτυπωτές: Η δυνατότητα άμεσης εκτύπωσης αποδείξεων και δελτίων παραγγελίας σε θερμικούς εκτυπωτές (μέσω Bluetooth ή δικτύου) θα καθιστούσε το σύστημα πολύ πιο λειτουργικό.

Dark Theme: Το χαρακτηριστικό αυτό κρίνεται ιδιαίτερα σημαντικό για περιβάλλοντα εστίασης με χαμηλό φωτισμό (όπως νυχτερινά κέντρα ή μπαρ), καθώς η χρήση μιας φωτεινής οθόνης κουράζει τα μάτια του προσωπικού και μπορεί να ενοχλήσει τους πελάτες. Η μελλοντική υλοποίηση θα αποθηκεύει την προτίμηση του χρήστη τοπικά (μέσω SharedPreferences) και θα προσαρμόζει αυτόματα την παλέτα χρωμάτων.

Αναζήτηση Προϊόντων στον Κατάλογο: Καθώς ένας κατάλογος εστιατορίου μπορεί να μεγαλώσει και να περιλαμβάνει εκατοντάδες κωδικούς, η χειροκίνητη πλοήγηση μέσω των κατηγοριών ενδέχεται να καθυστερεί τον υπάλληλο. Μια σημαντική μελλοντική προσθήκη είναι η ενσωμάτωση μιας μπάρας αναζήτησης.

5.4 Επίλογος

Η εκπόνηση της παρούσας πτυχιακής εργασίας αποτέλεσε μια πολυδιάστατη πρόκληση, καθώς απαίτησε τη σύνθεση γνώσεων από διαφορετικά πεδία της πληροφορικής. Από τον σχεδιασμό διεπαφών χρήστη (UI/UX) και τον προγραμματισμό φορητών συσκευών, μέχρι την ανάπτυξη backend και τη διαχείριση βάσεων δεδομένων.

Το τελικό αποτέλεσμα, το σύστημα OrderFlow, αποδεικνύει στην πράξη ότι η σύγχρονη μηχανική λογισμικού δεν αφορά απλώς τη συγγραφή κώδικα, αλλά την επίλυση πραγματικών προβλημάτων. Η χρήση προχωρημένων αρχιτεκτονικών προτύπων (όπως το Monorepo, το BLoC Pattern και η επικοινωνία μέσω REST API), διασφάλισε ότι το σύστημα δεν είναι απλώς ένα στατικό ακαδημαϊκό παραδοτέο, αλλά μια λειτουργική, επεκτάσιμη και πραγματικά αξιοποιήσιμη πλατφόρμα.

Μέσα από αυτό το ταξίδι, επιβεβαιώθηκε ότι η επιτυχία ενός συστήματος λογισμικού στην εστίαση δεν κρίνεται μόνο από την ορθότητα του αλγορίθμου, αλλά κυρίως από την ταχύτητα απόκρισης και την ικανότητά του να κρύβει την πολυπλοκότητά του από τον τελικό χρήστη. Το OrderFlow πετυχαίνει αυτόν ακριβώς τον στόχο, θέτοντας γερά θεμέλια για τη συνεχή εξέλιξη και προσαρμογή του στις μελλοντικές ανάγκες της κοινωνίας.

ΒΙΒΛΙΟΓΡΑΦΙΑ

Παραθέτω την σχετική βιβλιογραφία:

Βιβλία

- [1] M. Fowler, Patterns of Enterprise Application Architecture. Addison-Wesley Professional, 2002.
- [7] C. Bauer and G. King, Java Persistence with Hibernate, 2nd ed. Shelter Island, NY: Manning Publications, 2015
- [8] C. Walls, Spring in Action, 6th ed. Shelter Island, NY: Manning Publications, 2022.
- [11] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software. Reading, MA: Addison-Wesley, 1994.

Application Note

Πατέντες

Data Sheet

Internet Site

- [2] T. Bray, Ed., "The application/json Media Type for JavaScript Object Notation (JSON)," Internet Engineering Task Force (IETF), RFC 8259, 2017. [Online]. Available: <https://www.rfc-editor.org/info/rfc8259/>
- [3] Flutter, "Dio: A powerful HTTP client for Dart," Dart Packages (Pub.dev), 2026. [Online]. Available: <https://pub.dev/packages/dio>
- [4] Google, "built_value: Value types with builders, Dart classes as enums, and serialization," Dart Packages (Pub.dev), 2026. [Online]. Available: https://pub.dev/packages/built_value
- [5] Blue Fire, "audioplayers: A Flutter plugin to play multiple audio files simultaneously," Dart Packages (Pub.dev), 2026. [Online]. Available: <https://pub.dev/packages/audioplayers>
- [6] Apache Software Foundation, "Maven – POM Reference: Aggregation", maven.apache.org. [Online]. Διαθέσιμο: <https://maven.apache.org/pom.html>
- [9] Google, "Gson User Guide", github.com/google/gson. [Online]. Διαθέσιμο: <https://github.com/google/gson/blob/master/UserGuide.md>.
- [12] Google, "Flutter - Build apps for any screen," Flutter.dev, 2026. [Online]. Available: <https://flutter.dev/>
- [13] F. Angelov, "bloc: A predictable state management library that helps implement the BLoC design pattern," Dart Packages (Pub.dev), 2026. [Online]. Available: https://pub.dev/packages/flutter_bloc
- [14] Flutter, "go_router: A declarative routing package for Flutter," Dart Packages (Pub.dev), 2026. [Online]. Available: https://pub.dev/packages/go_router

- [15] R. Roussel, "flutter_hooks: A Flutter implementation of React hooks," Dart Packages (Pub.dev), 2026. [Online]. Available: https://pub.dev/packages/flutter_hooks
- [16] Google, "Dart API Reference: Timer class - dart:async library," Dart.dev. [Online]. Available: <https://api.dart.dev/stable/dart-async/Timer-class.html>. [Accessed: Sep. 2026].
- [10] Google, "Digital Ink Recognition," Google ML Kit. Διαθέσιμο: <https://developers.google.com/ml-kit/vision/digital-ink-recognition>. [Πρόσβαση: 6 Σεπτεμβρίου 2026].

Paper in Conference Proceedings

Journal Articles