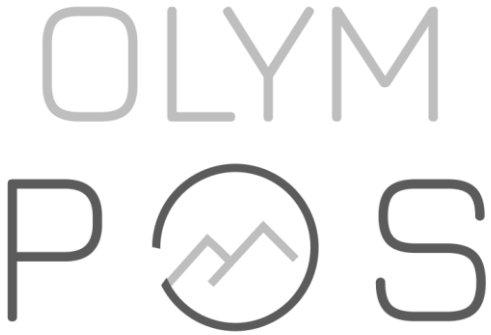
ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

«Ανάπτυξη συστήματος διαχείρισης παραγγελιών εστίασης με τεχνολογίες web»



Της φοιτήτριας
Ρηγούτσου Λεονάρδας
Αρ. Μητρώου: 154531

Επιβλέπων
Ουγιάρογλου Στέφανος
Επ. Καθηγητής

09 Σεπτεμβρίου 2023

Τίτλος Δ.Ε: Ανάπτυξη συστήματος διαχείρισης παραγγελιών εστίασης με τεχνολογίες web
Κωδικός Δ.Ε 22250
Ονοματεπώνυμο φοιτήτριας: Ρηγούτσου Λεονάρδα
Ονοματεπώνυμο εισηγητή: Ουγιάρογλου Στέφανος
Ημερομηνία ανάληψης Δ.Ε. : 10-10-2022
Ημερομηνία περάτωσης Δ.Ε. : 05-10-2023

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

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

«Στους γονείς μου»

Πρόλογος

Η παρούσα διπλωματική εργασία συντάχθηκε και αναπτύχθηκε από την φοιτήτρια Ρηγούτσου Λεονάρδα υπό την εποπτεία του καθηγητή κ. Στέφανου Ουγιάρογλου. Εστιάζει στην ανάπτυξη μιας ολοκληρωμένης διαδικτυακής εφαρμογής που απευθύνεται για τον τομέα της εστίασης. Παρατηρώντας τις προκλήσεις που αντιμετωπίζει ο τομέας της εστίασης, τέθηκαν στόχοι που βοηθούν στην επίλυση των προβλημάτων και διευκολύνουν την καθημερινότητα όλων των άμεσα αλλά και έμμεσα συσχετιζόμενων. Στόχος της παρούσας εφαρμογής, αποτελεί η δημιουργία ενός συστήματος παραγγελιοληψίας και διαχείρισης αυτών, γνωστό και ως POS (Point of Sale - Σημείο Πώλησης), χρησιμοποιώντας σύγχρονες τεχνολογίες web. Αναλαμβάνοντας την ανάπτυξη ενός τέτοιου συστήματος, με όφελος τη γνώση και εξοικείωση με διάφορες τεχνολογίες, οδηγηθήκαμε στη συνεισφορά της επίλυσης των προβλημάτων αρκετών επιχειρήσεων παρέχοντας τους την εφαρμογή λογισμικού και στη διεύρυνση των γνώσεων μου αντιμετωπίζοντας διάφορες προκλήσεις.

Περίληψη

Καθώς η βιομηχανία της εστίασης αντιμετωπίζει διαρκώς αυξανόμενες ανάγκες και προκλήσεις, οι πελάτες αναζητούν ολοένα πιο αποτελεσματικές λύσεις για την ικανοποίηση των αναγκών τους. Η πληθώρα πλεονεκτημάτων των τεχνολογιών του παγκόσμιου ιστού, όπως η συμβατότητα με όλες τις συσκευές, η διαθεσιμότητα των υπηρεσιών και η άμεση ενημέρωση του λογισμικού σε συνδυασμό με το χαμηλό κόστος ανάπτυξης και συντήρησης αποτέλεσαν την ιδέα για την ανάπτυξη μίας εφαρμογής που αξιοποιεί τις βασικές αυτές αρχές. Η παρούσα διπλωματική εργασία παρουσιάζει ένα σύστημα παραγγελιοληψίας, γνωστό και ως Point of Sale (POS) το οποίο δημιουργήθηκε για τον χώρο της εστίασης, χρησιμοποιώντας σύγχρονες τεχνολογίες web. Το παρόν σύστημα σχεδιάστηκε για να βελτιστοποιήσει τόσο τη διαδικασία της παραγγελιοληψίας, όσο και τη διαχείριση της εκάστοτε επιχείρησης. Λαμβάνοντας υπόψη τις μεταβαλλόμενες απαιτήσεις της αγοράς, η παρούσα εφαρμογή διευκολύνει το προσωπικό να είναι πιο αποδοτικό, μειώνοντας τον χρόνο αναμονής των πελατών και διαχειρίζοντας τη διαδικασία των παραγγελιών σε κάθε στάδιο. Η παρούσα εργασία αποτελείται από δύο τμήματα. Στο πρώτο τμήμα θα γίνει μία σύντομη αναφορά στα συστήματα παραγγελιοληψίας. Στη συνέχεια θα αναφερθούν οι τεχνολογίες που χρησιμοποιήθηκαν για τη σχεδίαση και υλοποίηση της εφαρμογής. Στο δεύτερο τμήμα ακολουθεί μία παρουσίαση της εφαρμογής που υλοποιήθηκε, με όλες τις δυνατότητες που μπορεί να προσφέρει.

«Development of a Web-based Restaurant Order Management System»

«Rigoutsou Leonarda»

Abstract

As the restaurant industry faces ever-increasing needs and challenges, customers are seeking more effective solutions to meet their demands. The multitude of benefits from global web technologies, such as compatibility across all devices, service availability, and real-time software updates, combined with low development and maintenance costs, have given rise to the idea of developing an application that leverages these fundamental principles. This thesis presents an order-taking system, known as Point Of Sale (POS) that was created for the restaurant space, utilizing modern web technologies. The current system was designed to optimize both the order-taking process and the management of each business. Considering the evolving market requirements, this application facilitates staff efficiency, reducing customer waiting time, and managing the order process at every stage. This work consists of two sections. The first section provides a brief overview of order-taking systems. Following that, the technologies used for the design and implementation of the application are discussed. In the second section, there is a presentation of the implemented application, highlighting all the capabilities it can offer.

Ευχαριστίες

Ξεκινώντας, θα ήθελα να ευχαριστήσω την οικογένεια μου, η οποία με στήριξε από την αρχή της εκπόνησης της διπλωματικής μου εργασίας έως και το τέλος αυτής. Η κατανόηση, η στήριξη και η υπομονή που μου έδωσαν αποτέλεσαν σημαντικά στοιχεία για την επιτυχή ολοκλήρωση της εργασίας μου.

Στην συνέχεια, θα ήθελα να ευχαριστήσω τον καθηγητή μου, κ. Στέφανο Ουγιάρογλου για την εμπιστοσύνη του σε εμένα και την καθοδήγησή του σε όλη την ανάπτυξη της εργασίας, καθιστώντας ομαλή όλη την διαδικασία από τις χρήσιμες συμβουλές του για την διαχείριση του χρόνου έως και τον ξεκάθαρο προσδιορισμό τον στόχων της εργασίας αυτής.

Τέλος, η συμπαράσταση όλων των φίλων και συναδέλφων για την πίστη τους σε εμένα, αποτέλεσε σημαντικό παράγοντα για την ολοκλήρωση αυτής την εργασίας.

Περιεχόμενα

Πρόλογος……. ii
Περίληψη……. iii
Abstract…… iv
Ευχαριστίες……. v
Περιεχόμενα…… vi
Κατάλογος Σχημάτων viii
Συντομογραφίες ix
Κεφάλαιο 1ο: Εισαγωγή 1
1.1 Συστήματα παραγγελιοληψίας 1
1.2 ΕΛΛΑΚ και Συστήματα παραγγελιοληψίας 1
1.3 Κίνητρο 2
1.4 Συνεισφορά 3
1.5 Οργάνωση εργασίας 3
Κεφάλαιο 2ο: Τεχνολογίες 5
2.1 Node.js 5
2.2 Express.js 5
2.3 Postgres/Sequelize 6
2.4 Typescript 7
2.5 Angular 7
2.6 Βιβλιοθήκες 9
2.6.1 bcrypt.js 9
2.6.2 body-parser 9
2.6.3 cors 9
2.6.4 jsonwebtoken 9
2.6.5 Multer 10
2.6.6 socket.io 10
2.6.7 Angular Material 11
2.6.8 Rxjs 11
2.6.9 socket.io-client 13
2.6.10 jwt-decode 13
2.7 Εργαλεία 13
2.7.1 Figma 13

2.7.2 Visual Studio Code 13
2.7.3 ElephantSQL 14
2.7.4 PG Admin 14
2.7.5 Git 14
2.7.6 Swagger 15
Κεφάλαιο 3ο: Σχεδίαση και Υλοποίηση της εφαρμογής 16
3.1 Λειτουργικές απαιτήσεις 16
3.2 Αρχιτεκτονική της εφαρμογής 21
3.3 Σχεδίαση της Βάσης Δεδομένων 23
3.4 Αυθεντικοποίηση 25
3.5 Υλοποίηση του Backend 27
3.6 Υλοποίηση του Frontend 30
3.7 Η εφαρμογή στο GitHub 37
Κεφάλαιο 4ο: Παρουσίαση της εφαρμογής 38
4.1 Διαχειριστής 38
4.2 Παραγγελιολήπτης 44
4.3 Παραγγελιοδέκτης 46
Κεφάλαιο 5ο: Συμπεράσματα και Μελλοντικές επεκτάσεις 47
5.1 Συμπεράσματα 47
5.2 Μελλοντικές Επεκτάσεις 47
ΒΙΒΛΙΟΓΡΑΦΙΑ 48
ΠΑΡΑΡΤΗΜΑ Α: ΚΩΔΙΚΑΣ BACKEND 50
ΠΑΡΑΡΤΗΜΑ Β: ΚΩΔΙΚΑΣ FRONTEND 61
ΠΑΡΑΡΤΗΜΑ Γ: ΒΑΣΗ ΔΕΔΟΜΕΝΩΝ 78

Κατάλογος Σχημάτων

Σχήμα 3.1: Διάγραμμα περιπτώσεων χρήσης συστήματος παραγγελιοληψίας..... 20
Σχήμα 3.2: Διάγραμμα ροής συστήματος παραγγελιοληψίας..... 21
Σχήμα 3.3: Διάγραμμα αρχιτεκτονικής συστήματος..... 22
Σχήμα 3.4: Διάγραμμα Οντοτήτων-Συσχετίσεων της βάσης δεδομένων του συστήματος..... 25
Σχήμα 3.5: Διάγραμμα ροής πληροφορίας αυθεντικοποίησης..... 27
Σχήμα 3.6: Υλοποίηση όλων των endpoint του API στο Swagger..... 28
Σχήμα 3.7: Δυνατότητα επεξεργασίας των endpoints για λειτουργίες CRUD..... 29
Σχήμα 3.8: Η εφαρμογή στο GitHub..... 37
Σχήμα 4.1: Κεντρική σελίδα διαχειριστή με 6 λειτουργίες (/admin)..... 38
Σχήμα 4.2: Προβολή κλειστών και ακυρωμένων παραγγελιών (/admin/reports)..... 39
Σχήμα 4.3: Δημιουργία και επεξεργασία αναπαράστασης τραπεζιών (/admin/map)..... 39
Σχήμα 4.4: Προβολή και διαχείριση των υπαλλήλων (/admin/employees)..... 40
Σχήμα 4.5: Πλαίσιο επεξεργασίας στοιχείων υπαλλήλου..... 40
Σχήμα 4.6: Προβολή και διαχείριση κατηγοριών και προϊόντων (/admin/editmenu)..... 41
Σχήμα 4.7: Πλαίσιο επεξεργασίας κατηγορίας και προϊόντος..... 42
Σχήμα 4.8: Προβολή και διαχείριση κρατήσεων (/admin/reservations)..... 42
Σχήμα 4.9: Πλαίσιο επεξεργασίας στοιχείων κράτησης..... 43
Σχήμα 4.10: Δυνατότητα φόρτωσης εικονιδίων για τροποποίηση της εφαρμογής (/admin/images)..... 43
Σχήμα 4.11: Κεντρική σελίδα παραγγελιολήπτη με εμφάνιση κατάστασης τραπεζιού (/server/map)..... 44
Σχήμα 4.12: Σελίδα επεξεργασίας παραγγελιών (/server/order?table={tableNumber})..... 45
Σχήμα 4.13: Κεντρική σελίδα παραγγελιοδεκτών (/cooker/orders)..... 46

Συντομογραφίες

Δ.Ε.	Διπλωματική Εργασία
ΔΙΠΑΕ	Διεθνές Πανεπιστήμιο Ελλάδος
Π.Ε.	Πτυχιακή Εργασία
ΕΛΛΑΚ	Ελεύθερο Λογισμικό Λογισμικού Ανοικτού Κώδικα
WEB	World Wide Web
HTTP	Hypertext Transfer Protocol
URL	Uniform Resource Locator
API	Application Programming Interface
SPA	Single Page Application
CSS	Cascading Style Sheets
HTML	Hypertext Markup Language
REST	REpresentational State Transfer
POS	Point of Sale
ORM	Object Relational Mapping
CRUD	Create Read Update Delete
DOM	Document Object Model
JSON	JavaScript Object Notation
CORS	Cross Origin Resource Sharing
HMAC	Hash-based message authentication code
TCP	Transfer Control Protocol
IETF	Internet Engineering Task Force
RFC	Requests for Comments
YAML	Yet Another Markup Language

Κεφάλαιο 1ο: Εισαγωγή

1.1 Συστήματα παραγγελιοληψίας

Ένα σύστημα παραγγελιοληψίας, γνωστό και ως σημείο πώλησης (Point Of Sale - POS) αποτελεί μία τεχνολογική λύση που χρησιμοποιείται στις επιχειρήσεις για την υποβολή παραγγελιών αλλά και τη διαχείριση και πληρωμή αυτών.

Στις μέρες μας ένα σύστημα παραγγελιοληψίας μπορεί να αξιοποιηθεί κυρίως στην εστίαση, αλλά και σε διάφορα καταστήματα λιανικού και χονδρικού εμπορίου. Λόγω της πληθώρας πλεονεκτημάτων που μπορεί να προσφέρει, μπορεί να θεωρεί ως απαραίτητο εργαλείο για την εύρυθμη λειτουργία μιας επιχείρησης. Σημαντικό λόγο για τη χρήση ενός τέτοιου συστήματος σε μία τέτοια επιχείρηση αποτελεί η ταχύτητα των παραγγελιών καθώς αποστέλλονται σε άμεσο χρόνο πράγμα που σημαίνει ότι βελτιώνει την εξυπηρέτηση των πελατών αλλά διευκολύνει και τον ίδιο τον χρήστη/εργαζόμενο. Καθώς τα σύγχρονα συστήματα παραγγελιών εξυπηρετούν διαφόρους χρήστες και όχι μόνο τον παραγγελιολήπτη και παραγγελιοδέκτη, έχουν ενσωματώσει διάφορες λειτουργίες και μπορούν να εξυπηρετήσουν ρόλους όπως του διαχειριστή. Πιο συγκεκριμένα, οι παραγγελίες μπορούν να αποθηκεύονται στο ιστορικό, τα προϊόντα του καταλόγου να δημιουργούνται και να τροποποιούνται στον κατάλογο, να παρέχεται η δυνατότητα δημιουργίας κρατήσεων, να καταχωρούνται στο σύστημα οι χρήστες και πολλές ακόμα λειτουργίες. Η πληθώρα των πλεονεκτημάτων αποτελεί απάντηση για μία επιχείρηση που θέτει το ερώτημα για τη χρήση ενός τέτοιου συστήματος.

Τέτοια συστήματα μπορούν να υλοποιηθούν είτε σαν εφαρμογές που λειτουργούν μέσω του παγκόσμιου ιστού (world wide web) και αποτελούν την πλειοψηφία, χωρίς να απαιτείται η λήψη ή εγκατάσταση επιπλέον λογισμικού στη συσκευή, είτε σαν εφαρμογές υπολογιστή που απαιτείται εγκατάσταση τοπικά στη συσκευή και εκτελείται απευθείας από το λειτουργικό σύστημα είτε εφαρμογές φορητών συσκευών που εκτελούνται στο λειτουργικό σύστημα που χρησιμοποιούν, όπως το iOS και το Android.

1.2 ΕΛΛΑΚ και Συστήματα παραγγελιοληψίας

Ο όρος “Ελεύθερο Λογισμικό και Λογισμικό Ανοικτού Κώδικα” αναφέρεται σε λογισμικό το οποίο διανέμεται με τρόπο που επιτρέπει στους χρήστες την ελεύθερη πρόσβασή τους. Με τον χαρακτηρισμό “Ελεύθερο Λογισμικό” δεν εννοείται στην τιμή του, δηλαδή ότι παρέχεται δωρεάν αλλά ότι μπορεί να χρησιμοποιηθεί, μελετηθεί, τροποποιηθεί, εκτελεστεί και διανεμηθεί από τους χρήστες. Προκειμένου να χαρακτηριστεί ένα “Λογισμικό Ανοιχτού κώδικα” δεν αρκεί να παρέχεται πρόσβαση στον πηγαίο κώδικα αλλά πρέπει να πληροί συγκεκριμένες προϋποθέσεις όπως ότι η διανομή πραγματοποιείται δωρεάν και το λογισμικό να συνοδεύεται με την αρχική άδεια εκμετάλλευσης, ότι ο πηγαίος κώδικας διατίθεται μαζί με το λογισμικό είτε υπάρχει εύκολη πρόσβαση σε αυτόν και είναι εφικτή η τροποποίησή του.

Σκοπός αυτού το μοντέλου αποτελεί η ενδυνάμωση του τελικού χρήστη παρέχοντας του αρκετά οφέλη όπως είναι τα παρακάτω:

- Μεγάλη κοινότητα ανάπτυξης και υποστήριξης του λογισμικού
- Ασφάλεια καθώς ο καθένας μπορεί να μελετήσει τον πηγαίο κώδικα της εφαρμογής και να επιβεβαιώσει τις λειτουργίες του

- Απόλυτα νόμιμο και συνήθως χωρίς κόστος ενώ τείνει να έχει και μικρότερο κόστος συντήρησης
- Η χρήση του μπορεί να γίνει τόσο από ιδιώτες όσο και από δημόσιους οργανισμούς και υπηρεσίες
- Δίνει τη δυνατότητα στον χρήστη να προσαρμόσει το λογισμικό ανάλογα με τις ανάγκες του, δίνοντας του τεράστια ευελιξία

Η νοοτροπία του διαμοιρασμού του πηγαίου κώδικα ξεκίνησε από μερικούς προγραμματιστές στις ΗΠΑ το 1983, λόγω της ανάγκης για την εξέλιξη του υπάρχοντος λογισμικού. Με τον διαμοιρασμό του κώδικα επιτυγχάνεται η βελτίωση του και η διόρθωση των δυσλειτουργιών από τα μέλη της κοινότητας.

Στη σύγχρονη εποχή, καθώς οι ανάγκες για τη χρήση συστημάτων παραγγελιοληψίας ολοένα και αυξάνονται, έχουν δημιουργηθεί συστήματα τα οποία είναι βασισμένα στη φιλοσοφία του λογισμικού ανοιχτού κώδικα και εξελίσσονται σημαντικά. Πιο συγκεκριμένα, οι εφαρμογές λογισμικού συστημάτων παραγγελιοληψίας που χαρακτηρίζονται ελεύθερου και ανοικτού κώδικα εξασφαλίζουν την αποτελεσματική και ομαλή λειτουργία της επιχείρησης καθώς επιτρέπουν στους επιχειρηματίες να προσαρμόσουν το λογισμικό σύμφωνα με τις ανάγκες και τις απαιτήσεις τους.

Ορισμένα παραδείγματα εφαρμογών λογισμικού POS ανοιχτού κώδικα αποτελούν τα παρακάτω:

- uniCenta, ευέλικτο σύστημα με δυνατότητες προσαρμογής και υποστήριξη διαφόρων περιφερειακών συσκευών
- Loyverse POS, κατάλληλο για μικρές επιχειρήσεις για τον τομέα της εστίασης το οποίο προσφέρει μεγάλη ευκολία στη διαχείριση παραγγελιών και πληρωμών
- Floreant POS, απλό στη λειτουργικότητα για παραγγελίες και τιμολόγηση με κύρια εφαρμογή στον τομέα της εστίασης, για καφετέριες και εστιατόρια
- Odoo POS, ευέλικτο και πλήρες POS σύστημα με πληθώρα δυνατοτήτων όπως είναι η παραγγελίες, η τιμολόγηση, η διαχείριση αποθεμάτων και πολλά άλλα
- Openbravo POS, προηγμένη επιλογή με εκτεταμένες δυνατότητες διαχείρισης αποθεμάτων, παραγγελιών και τιμολόγησης

1.3 Κίνητρο

Βασικό έναυσμα για την δημιουργία της εφαρμογής αποτελεί η επιθυμία για την δημιουργία ενός ελεύθερου και ανοικτού κώδικα λογισμικό το οποίο να εξυπηρετεί τις ανάγκες για ένα ολοκληρωμένο σύστημα παραγγελιοληψίας, που παρέχει δυνατότητες εξατομίκευσης για την κάθε επιχείρηση. Πιο συγκεκριμένα, κίνητρο αποτέλεσε η επιχείρηση της οικογένειας μου που δραστηριοποιείται στον τομέα της εστίασης και κατ' επέκταση η κάθε επιχείρηση που επιθυμεί να κάνει χρήση αυτής της εφαρμογής. Παρατηρώντας τις δυσκολίες που αντιμετωπίζουν τέτοιου είδους επιχειρήσεις, η ανάπτυξη της παρούσας διαδικτυακής εφαρμογής καλείται να διευκολύνει την όλη διαδικασία για την διαχείριση των παραγγελιών της εκάστοτε επιχείρησης. Εστιάζοντας στις ανάγκες και απαιτήσεις των μικρομεσαίων επιχειρήσεων εστίασης, με κύριο γνώμονα την προσαρμοστικότητα και την ευκολία στην λειτουργία διανέμεται η διαδικτυακή εφαρμογή προς χρήση.

1.4 Συνεισφορά

Η συγκεκριμένη διαδικτυακή εφαρμογή, όπως θα αναλυθεί και στα παρακάτω κεφάλαια, επικεντρώνεται στη διαδικασία της παραγγελιοληψίας από το πρώτο στάδιο έως και το τελευταίο. Πιο συγκεκριμένα, ο παραγγελιολήπτης είναι υπεύθυνος τόσο για τη λήψη της παραγγελίας όσο για την παρακολούθηση της και τέλος το κλείσιμο της, αφού ολοκληρωθεί. Το διαδραστικό αυτό σύστημα επιτρέπει την άμεση επικοινωνία του παραγγελιολήπτη με τον παραγγελιοδέκτη και αντίστροφα, μέσα από σύγχρονες τεχνολογίες web. Παρακάτω παρουσιάζεται ο κάθε ρόλος του συστήματος με τις αντίστοιχες λειτουργίες που μπορούν να εκτελέσουν.

Κύριος ρόλος της εφαρμογής αποτελεί ο παραγγελιολήπτης, ο οποίος αντιστοιχεί στον σερβιτόρο της επιχείρησης που δημιουργεί τις παραγγελίες. Ο παραγγελιολήπτης έχει τη δυνατότητα επιλογής τραπεζιού για την καταχώρηση της παραγγελίας. Στη συνέχεια, προσθέτει την ποσότητα και τα προϊόντα της παραγγελίας και την αποστέλλει άμεσα στον παραγγελιοδέκτη που αντιστοιχεί στην παραγωγή της επιχείρησης που λαμβάνει τις παραγγελίες προς ετοιμασία.

Ο παραγγελιοδέκτης, λαμβάνει άμεσα την παραγγελία για την εκκίνηση της ετοιμασίας αυτής. Έχοντας δυνατότητα επιβεβαίωσης και ακύρωσης της παραγγελίας μπορεί να επικοινωνήσει με αυτές τις δύο ενέργειες με τον παραγγελιολήπτη για τη σχετική ενημέρωση της κατάστασης της παραγγελίας.

Σημαντικό ρόλο με πληθώρα δυνατοτήτων αποτελεί και ο τρίτος ρόλος τους συστήματος, ο διαχειριστής. Ο διαχειριστής εκτελεί σημαντικές λειτουργίες όπως είναι προσθήκη κατηγοριών και προϊόντων στον κατάλογο, η δημιουργία κρατήσεων, η προβολή όλων των παραγγελιών που ολοκληρώθηκαν ή ακυρώθηκαν ώστε να γνωρίζει περισσότερες πληροφορίες για το ιστορικό των παραγγελιών, η δημιουργία των τραπεζιών με τη δυνατότητα καταχώρησής τους σε συγκεκριμένες θέσεις με δυναμικό τρόπο και τέλος η δυνατότητα αποθήκευσης εικονιδίων για τα τραπέζια και το λογότυπο ώστε να τροποποιηθεί για την εκάστοτε επιχείρηση.

1.5 Οργάνωση εργασίας

Στο τρέχων κεφάλαιο έγινε μία σύντομη αναφορά για τα συστήματα παραγγελιοληψίας. Επιπλέον, αναφέρεται ο όρος του ελεύθερου λογισμικού και λογισμικού ανοικτού κώδικα καθώς και ορισμένες εφαρμογές παραγγελιοληψίας στην εστίαση που αποτελούν λογισμικά ανοιχτού κώδικα με τα κύρια χαρακτηριστικά της κάθε εφαρμογής και τα πλεονεκτήματα τους.

Στο δεύτερο κεφάλαιο ακολουθεί η επεξήγηση των τεχνολογιών που χρησιμοποιήθηκαν για την ανάπτυξη της εφαρμογής. Πιο συγκεκριμένα, το NodeJS για την υλοποίηση του backend με τη βοήθεια του ExpressJS για την υλοποίηση του API, η PostgreSQL για τη σχεσιακή βάση δεδομένων με χρήση της ORM βιβλιοθήκης Sequelize και τέλος το framework Angular με χρήση της TypeScript για την υλοποίηση του web client. Επιπλέον, θα παρουσιαστούν οι βιβλιοθήκες που χρησιμοποιήθηκαν τόσο στο frontend όσο και στο backend και θα αναφερθούν τα βασικά χαρακτηριστικά και ο τρόπος λειτουργίας της κάθε τεχνολογίας και βιβλιοθήκης. Στο τέλος του κεφαλαίου, θα παρουσιαστούν και τα εργαλεία που χρησιμοποιήθηκαν για την ανάπτυξη της εφαρμογής.

Στο τρίτο κεφάλαιο, εστιάζεται στη σχεδίαση και η υλοποίηση της εφαρμογής. Θα γίνει αναφορά στις λειτουργικές απαιτήσεις του συστήματος, στη συνέχεια αναλύεται η αρχιτεκτονική της εφαρμογής και

μετά επεξηγείται η σχεδίαση της βάσης δεδομένων. Έπειτα ακολουθεί η ανάλυση της αυθεντικοποίησης στην εφαρμογή και τέλος η επεξήγηση του κώδικα, τόσο για τον server όσο και για την web εφαρμογή. Γίνεται επίσης και μια σύντομη αναφορά για τον διαμοιρασμό της εφαρμογής μέσω του GitHub, μιας πλατφόρμας αποθετηρίων ανοιχτού κώδικα.

Στο τέταρτο κεφάλαιο, παρουσιάζεται η εφαρμογή. Για το ολοκληρωμένο σύστημα το οποίο δημιουργήθηκε, γίνεται τελική παρουσίαση των δυνατοτήτων του από την πλευρά του χρήστη και αναλύεται περαιτέρω η λειτουργικότητά του για τον κάθε ρόλο. Σε αυτό το σημείο η εφαρμογή εξετάζεται σαν ένα εργαλείο για την επίλυση των προβλημάτων που καλείται να λύσει και των προκλήσεων που προκύπτουν στην επικοινωνία μεταξύ των ρόλων του συστήματος.

Τέλος, στο πέμπτο κεφάλαιο αναφέρονται τα συμπεράσματα από την εκπόνηση της διπλωματικής αυτής εργασίας. Κλείνοντας, θα προταθούν ορισμένες ιδέες για μελλοντική επέκταση του συστήματος καθιστώντας το πιο εύχρηστο με ακόμα περισσότερες λειτουργίες.

Κεφάλαιο 2ο: Τεχνολογίες

Στο παρόν κεφάλαιο παρουσιάζονται οι τεχνολογίες και τα εργαλεία που χρησιμοποιήθηκαν για την υλοποίηση της εφαρμογής. Θα δοθεί μία σύντομη περιγραφή για την κάθε τεχνολογία. Οι τεχνολογίες περιλαμβάνουν τις γλώσσες προγραμματισμού, τη βάση δεδομένων και τη βιβλιοθήκη που συνδέεται με αυτή, το framework της web εφαρμογής, τις βιβλιοθήκες που εξυπηρετούν διάφορες λειτουργίες τόσο στο backend όσο και στο frontend, καθώς επίσης και τα εργαλεία που βοήθησαν στην ανάπτυξη της εφαρμογής όπως το περιβάλλον ανάπτυξης, η σελίδα σχεδίασης της διεπαφής και τα εργαλεία που βοήθησαν στην τεκμηρίωση και τη δοκιμή του κώδικα.

2.1 Node.js

Για την υλοποίηση του backend χρησιμοποιήθηκε το Node.js. Το Node.js είναι ένα περιβάλλον εκτέλεσης της γλώσσας JavaScript το οποίο επιτρέπει την ανάπτυξη εφαρμογών στην πλευρά του server πέρα από την τυπική χρήση της γλώσσας για προγραμματισμό διεπαφών σε περιηγητές. Το Node.js αξιοποιεί το V8 engine το οποίο είναι μια ανοιχτού κώδικα μηχανή εκτέλεσης της JavaScript η οποία μεταγλωττίζει σε πραγματικό χρόνο (just-in-time-compilation) [8] πριν την εκτέλεση της. Σε συνδυασμό με μια ακόμη βιβλιοθήκη της C++, την libuv καθίσταται δυνατή η εκτέλεση της JavaScript εκτός του browser, δίνοντας της πρόσβαση σε πόρους του τοπικού συστήματος.

Βασικό χαρακτηριστικό του Node.js αποτελεί η χρήση της ασύγχρονης αρχιτεκτονικής που επιτυγχάνεται μέσω non-blocking events. Με τον όρο non-blocking εννοείται ότι οι εντολές εκτελούνται παράλληλα σε ένα μόνο thread, το οποίο έχει ως αποτέλεσμα την ταχύτερη πρόσβαση στους πόρους του συστήματος και την εξυπηρέτηση περισσότερων REST αιτημάτων στον server. Ένα event στην περίπτωση μιας web εφαρμογής αντιπροσωπεύει ένα αίτημα στον server το οποίο εκείνος χειρίζεται και επιστρέφει την απάντηση στον χρήστη. Η αρχιτεκτονική της JavaScript βασίζεται στο event loop το οποίο είναι υπεύθυνο για τον έλεγχο των εισερχόμενων events και την εκτέλεσή τους χωρίς να δεσμεύει περιττούς πόρους και χωρίς την παρεμπόδιση της ολοκλήρωσης άλλων διεργασιών. Αξίζει να σημειωθεί ότι με το NodeJS καθίσταται δυνατή η ανάπτυξη ολοκληρωμένων εφαρμογών με μία μόνο γλώσσα και κατ' επέκταση η χρήση κοινών βιβλιοθηκών και εργαλείων.

2.2 Express.js

Το ExpressJS είναι ένα framework βασισμένο στο NodeJS το οποίο διευκολύνει την ανάπτυξη web εφαρμογών και APIs (Application Programming Interfaces). Βασική χρήση του αποτελεί η δρομολόγηση των αιτημάτων (routing) και η παροχή ενδιάμεσων ενεργειών (middlewares) όπως η αυθεντικοποίηση (authentication) και η έγκριση πρόσβασης των χρηστών (authorization). Μία ακόμα χρήση του είναι το σερβίρισμα στατικών αρχείων, όπως εικόνες και αρχεία ιστοσελίδων.

Για την δρομολόγηση των αιτημάτων (routing) το ExpressJS αντιστοιχίζει την διεύθυνση του αιτήματος με ένα middleware το οποίο εκτελεί μία μέθοδο και επιστρέφει την απάντηση με το αντίστοιχο κωδικό HTTP ή προωθεί το αίτημα σε ένα άλλο middleware. Η διεύθυνση μπορεί να περιλαμβάνει και δυναμικές τιμές οι οποίες εξάγονται και μπορούν να χρησιμοποιηθούν σαν ορίσματα στην μέθοδο που εκτελείται. Η αντιστοίχιση της διεύθυνσης περιλαμβάνει και το HTTP method (GET, POST, PUT, DELETE) με το οποίο έχει γίνει το αίτημα.

Ένα middleware περιλαμβάνει τρία ορίσματα τα οποία είναι τα δεδομένα του αιτήματος, τα δεδομένα της απάντησης και ένα callback function το οποίο παραπέμπει στο επόμενο στην σειρά middleware. Με τον όρο callback function εννοείται μία συνάρτηση η οποία δίνεται ως όρισμα με σκοπό την εκτέλεσή της σε μελλοντικό χρόνο και απαιτείται το αποτέλεσμά της μετά την εκτέλεση μιας λειτουργίας. Στην περίπτωση διακοπής της εκτέλεσης ενός middleware επιστρέφονται τα δεδομένα της απάντησης τα οποία δημιουργούνται δυναμικά κατά την εκτέλεση και αναβάλλεται η κλήση των επόμενων middlewares. Τα δεδομένα της απάντησης περιέχουν headers, το HTTP response code και τα δεδομένα της απάντησης.

Το express προσφέρει ένα αντικείμενο που δημιουργείται κατά την εκκίνηση του server και περιλαμβάνει πληροφορίες για το API που εξυπηρετείται και σε αυτό το αντικείμενο επισυνάπτονται πριν την εκκίνηση του server όλα τα ενδιάμεσα συστατικά που εκτελούν τις λειτουργίες του server. Η εκκίνηση του server ορίζεται με την μέθοδο listen, η οποία τον καθιστά ενεργό και δέχεται σαν παράμετρο την πόρτα (port) στην οποία περιμένει να δεχτεί τα αιτήματα. Πριν την κλήση της listen, παρατίθενται τα middlewares τα οποία αντιστοιχούν τα μονοπάτια των εισερχόμενων URL με συναρτήσεις που εκτελούν τον κώδικα του API. Η αντιστοίχιση γίνεται με την σειρά που δηλώνονται τα middlewares πριν εκκινηθεί ο server.

Για την δήλωση ενός καινούργιου endpoint του API αρχικά δίνεται σαν όρισμα σε μια συνάρτηση του app το μονοπάτι του και έπειτα η συνάρτηση έχει την ονομασία του HTTP method με το οποίο γίνεται το αντίστοιχο αίτημα. Το express δίνει την δυνατότητα της ομαδοποίησης αρκετών endpoint μέσα σε ένα αρχείο και την δήλωση ενός προθέματος που αντιστοιχίζει όλα τα ομαδοποιημένα endpoints μετά από αυτό. Μια κοινή πρακτική στα RESTful APIs είναι κάθε endpoint να περιέχει στο μονοπάτι του το /api. Με αυτόν τον τρόπο είναι πιο εύκολη η δήλωση των endpoints που αφορούν τον ίδιο πόρο μέσα στο ίδιο αρχείο χωρίς να επηρεάζεται η δομή της εφαρμογής. Για την χρήση των ομαδοποιημένων endpoints, δίνεται ως όρισμα στην μέθοδο use το πρόθεμα με ένα αντικείμενο το οποίο περιλαμβάνει τα middlewares που εκτελούνται με την αντιστοίχηση του κάθε endpoint.

Όσον αφορά το σερβίρισμα στατικών αρχείων μπορεί να οριστεί με την βοήθεια του ExpressJS η τοποθεσία στην μνήμη του server όπου είναι αποθηκευμένα και ανατίθεται μία διεύθυνση σε αυτόν τον προορισμό με την οποία ο χρήστης μπορεί να τα ανακτήσει.

2.3 Postgres/Sequelize

Για την αποθήκευση και διαχείριση των δεδομένων της εφαρμογής χρησιμοποιείται η ανοιχτού κώδικα σχεσιακή βάση δεδομένων PostgreSQL. Η υψηλή απόδοση της, σε συνδυασμό την ευελιξία της και τα μοντέρνα χαρακτηριστικά της την καθιστούν ιδανική επιλογή για μία λύση διαχείρισης δομημένων δεδομένων. Ως σχεσιακή βάση παρέχει την δική της δομημένη γλώσσα ερωτημάτων (Structured Query Language, SQL)

Για την επικοινωνία της βάσης δεδομένων με την εφαρμογή χρησιμοποιείται το Sequelize το οποίο είναι μία ORM (Object Relational Mapping) βιβλιοθήκη της JavaScript που παρέχει την δυνατότητα της σύνδεσης με την βάση και την μοντελοποίηση των σχεσιακών πινάκων σε αντικείμενα της γλώσσας. Μέσω του Sequelize επιτυγχάνεται η εκτέλεση των CRUD (Create, Read, Update, Delete) λειτουργιών στα δεδομένα μέσω κώδικα καθώς δεν είναι απαραίτητη η συγγραφή ερωτημάτων σε SQL.

Το Sequelize με την βοήθεια της βιβλιοθήκης pg εγκαθιδρύει μια σύνδεση με την PostgreSQL και λαμβάνοντας τα δεδομένα των ερωτημάτων ως ορίσματα αναλαμβάνει την εκτέλεση των ερωτημάτων

στην βάση και επιστρέφει τα αποτελέσματα ξανά σαν αντικείμενα. Με τη χρήση του Sequelize είναι δυνατή η αλλαγή της βάσης που χρησιμοποιείται χωρίς να υπάρχουν αλλαγές στο κώδικα καθώς τα ερωτήματα μπορούν να μεταφραστούν σε SQL για παραπάνω από μια γνωστές βάσεις δεδομένων.

Εκτός από την σύνδεση στη βάση και την εκτέλεση των queries, το Sequelize δίνει την δυνατότητα της περιγραφής και της δημιουργίας του schema της βάσης και των συσχετίσεων των πινάκων, καθώς ακόμα και την αλλαγή του μετά τη δημιουργία της βάσης. Η δημιουργία των πινάκων ξεκινάει με την περιγραφή των στηλών και των κλειδιών του πίνακα και κατόπιν μπορούν να δηλωθούν οι συσχετίσεις με τα ξένα κλειδιά και τον τρόπο με τον οποίο συνδέονται (One-to-One, One-to-Many, Many-to-Many). Στην περίπτωση της συσχέτισης Many-to-Many το Sequelize δημιουργεί αυτόματα τον ενδιάμεσο πίνακα και μπορούν να δηλωθούν τα πεδία του που δεν είναι ξένα κλειδιά όπως και στους υπόλοιπους πίνακες.

Για την αλλαγή του schema, το Sequelize προσφέρει έναν εύκολο τρόπο περιγραφής των αλλαγών και γίνεται κατά την επόμενη σύνδεση με τη βάση χωρίς να δημιουργεί συγκρούσεις ή απώλειες δεδομένων. Τα νεοεισαχθέντα πεδία του πίνακα μπορούν να πάρουν προκαθορισμένες τιμές κατά την εισαγωγή τους και οι αλλαγές μπορούν να καταγραφούν με την χρονική σειρά με την οποία συνέβησαν έτσι ώστε να είναι δυνατή η αναίρεση των αλλαγών σε μια προηγούμενη επιθυμητή κατάσταση. [12]

2.4 Typescript

Η Typescript είναι μια ανοιχτού κώδικα γλώσσα προγραμματισμού υψηλού επιπέδου η οποία προσθέτει την δυνατότητα στατικών τύπων στην γλώσσα JavaScript. Για την εκτέλεσή της ο κώδικας μεταφράζεται σε JavaScript οπότε μπορεί να θεωρηθεί και σαν ένα υποσύνολο της εφόσον ακολουθεί τους ίδιους κανόνες της γλώσσας με την προσθήκη επιπλέον χαρακτηριστικών. Η χρήση της κάνει πιο εύκολη την εύρεση συντακτικών λαθών και σφαλμάτων τύπων πριν την εκτέλεση του κώδικα, κάτι το οποίο δεν είναι εφικτό με την JavaScript. Εκτός από την χρήση των στατικών τύπων, η Typescript παρέχει και αρκετά χαρακτηριστικά που συναντώνται σε άλλες αντικειμενοστραφείς γλώσσες προγραμματισμού, όπως είναι τα interfaces, τα annotations, τα enumerations και τα generics, τα οποία επεκτείνουν τα ήδη υπάρχοντα αντικειμενοστραφή χαρακτηριστικά της γλώσσας. Η Typescript χρησιμοποιείται σε αρκετά ανοιχτού κώδικα εργαλεία για την υλοποίηση τους με την Angular να είναι ένα από αυτά.

2.5 Angular

Η Angular (Angular 2.0 και μεταγενέστερες εκδόσεις) είναι ένα πλαίσιο εφαρμογών σελίδας ιστού (web framework) βασισμένο στην γλώσσα Typescript, το οποίο χρησιμοποιείται για την δημιουργία εφαρμογών μίας σελίδας (Single Page Application). Ένα SPA φορτώνεται μια φορά στον φυλλομετρητή και όλες οι αλλαγές στα στοιχεία της σελίδας και η αλληλεπίδραση με τον server γίνονται στο παρασκήνιο δυναμικά, χωρίς να γίνει επαναφόρτωση της σελίδας.

Η εφαρμογή διασπάται σε μικρότερα συστατικά (components), το καθένα από τα οποία αντιστοιχεί σε μια μικρή λειτουργία της εφαρμογής, απλουστεύοντας έτσι τις διεργασίες και παρέχοντας αυτονομία στα υποσυστήματα της εφαρμογής. Για την δυναμική εμφάνιση των components χρειάζεται ένας μηχανισμός για την αντιστοίχιση τους με μια διεύθυνση από την οποία μπορούν να εισαχθούν

μεταβλητές είτε στο μονοπάτι της διεύθυνσης (path variables) είτε σαν επιπλέον παράμετροι στο τέλος της διεύθυνσης (query parameters). Για την εμφάνιση των components χρησιμοποιούνται templates γραμμένα σε HTML (Hyper Text Media Language) σε συνδυασμό με CSS (Cascading Style Sheets).

Η Angular περιγράφει τα templates σε ένα αρχείο με επέκταση .html και τα styles με επέκταση .css ενώ ο κώδικας της Typescript ο οποίος χειρίζεται τα δεδομένα και τα χορηγεί στο template περιγράφεται μέσα σε μία μόνο κλάση σε ένα αρχείο με επέκταση .ts. Τα τρία αυτά αρχεία αποθηκεύονται σε έναν ξεχωριστό φάκελο ο οποίος περιγράφει το component σαν ένα στοιχείο της εφαρμογής. Τα components μπορούν να ομαδοποιηθούν σε πακέτα γνωστά και ως modules δημιουργώντας με αυτόν τον τρόπο επαναχρησιμοποιούμενες βιβλιοθήκες από τα μικρότερα στοιχεία της εφαρμογής.

Για την σύνδεση των δεδομένων με τα στοιχεία που τα εμφανίζουν, αλλά και για την επιστροφή των events της Javascript που προκύπτουν από την αλληλεπίδραση του χρήστη, η Angular παρέχει δύο λύσεις γνωστές ως data binding και event binding. Το data binding είναι η χρήση των δεδομένων σαν μεταβλητές μέσα στα HTML στοιχεία του template ενώ το event binding είναι η κλήση μιας μεθόδου η οποία εκτελεί μια λειτουργία κάθε φορά που προκύπτει ένα συμβάν από την αλληλεπίδραση του χρήστη με την εφαρμογή. Ο συνδυασμός αυτών των δύο τεχνικών επιτρέπει την αμφίδρομη συσχέτιση των δεδομένων και των γραφικών στοιχείων της εφαρμογής χωρίς την περαιτέρω ανάπτυξη κώδικα για την μεταξύ τους επικοινωνία.

Για την αποστολή αιτημάτων στον server και την διαχείριση της κατάστασης των δεδομένων της εφαρμογής η Angular έχει μια ξεχωριστή κατηγορία κλάσεων οι οποίες αποκαλούνται υπηρεσίες (services). Σε συνδυασμό με την βιβλιοθήκη της Angular που εξασφαλίζει την ανάκτηση δεδομένων από τον server, οι υπηρεσίες παρέχουν στα επιμέρους συστατικά τα δεδομένα για την δυναμική απόκριση της εφαρμογής. Για την χρήση τους η Angular παρέχει την δυνατότητα του dependency injection, της δυναμικής φόρτωσης άλλων κλάσεων οι οποίες παρέχουν τα απαραίτητα δεδομένα για την λειτουργία της εφαρμογής μέσα στην κλάση του κάθε συστατικού. Με αυτόν τον τρόπο επιτυγχάνεται η ροή των δεδομένων από τον server στα επιμέρους στοιχεία χωρίς να χρειάζεται η απ΄ απευθείας επικοινωνία του κάθε στοιχείου ξεχωριστά με τον server αφού η υπηρεσία παρέχει τα ήδη υπάρχοντα δεδομένα σαν ενδιάμεσος.

Ένα ακόμα στοιχείο της Angular το οποίο χρησιμοποιείται αρκετά και από το ίδιο το framework αλλά και στην εφαρμογή είναι το directive. Ένα directive προστίθεται σαν ένα όρισμα σε ένα στοιχείο HTML ή ένα component της εφαρμογής και του δίνει μια ιδιότητα η οποία υλοποιείται στην κλάση του directive και μπορεί είτε να παραποιήσει το γραφικό στοιχείο είτε να υλοποιήσει μια μέθοδο που όταν κληθεί εκτελεί μια λειτουργία πάνω στο συγκεκριμένο στοιχείο ή component της Angular. Μερικά από τα πιο βασικά directives που παρέχει η Angular είναι το *ngIf και το *ngFor. Ο αστερίσκος πριν το directive δηλώνει ότι το συγκεκριμένο directive αλλάζει την μορφή του HTML δυναμικά προσθέτοντας ή αφαιρώντας elements από το DOM. Το ngIf περιέχει μια λογική συνθήκη η οποία καθορίζει αν θα εμφανιστεί ή όχι ένα element ενώ το ngFor δίνει την δυνατότητα της προσπέλασης μιας λίστας και της χρήσης των στοιχείων της ως δυναμικές παραμέτρους μέσα στα elements που περιλαμβάνει το συγκεκριμένο element.

Πέρα από τα βασικά χαρακτηριστικά της Angular έχουν υλοποιηθεί και βιβλιοθήκες οι οποίες προσθέτουν λειτουργίες που επιτρέπουν την επαναχρησιμοποίηση τόσο των γραφικών στοιχείων όσο

και της λειτουργικότητας της εφαρμογής, διευκολύνοντας έτσι την ανάπτυξη μεγάλων εφαρμογών με πολλά δυναμικά στοιχεία και την μείωση λαθών από ήδη υπάρχουσες λειτουργίες.

2.6 Βιβλιοθήκες

Για την υλοποίηση της παρούσας εργασίας χρησιμοποιήθηκαν αρκετές βιβλιοθήκες ανοιχτού κώδικα οι οποίες διευκόλυναν την ανάπτυξη κάποιων χαρακτηριστικών της εφαρμογής και μείωσαν αρκετό τον απαιτούμενο κώδικα για ορισμένες συμπληρωματικές λειτουργίες. Οι βιβλιοθήκες που παρουσιάζονται είναι όλες βασισμένες στην JavaScript και στην Typescript για τον κώδικα του backend και του frontend αντίστοιχα.

2.6.1 bcrypt.js

Το bcrypt.js είναι μια βιβλιοθήκη της JavaScript η οποία παρέχει μια υλοποίηση της συνάρτησης κατακερματισμού bcrypt, η οποία είναι χρήσιμη για την αποθήκευση ευαίσθητων δεδομένων στην βάση, όπως οι κωδικοί των χρηστών. Για τον κατακερματισμό των δεδομένων δίνεται ένα τυχαίο αλφαριθμητικό, γνωστό στην κρυπτογραφία ως Salt ή εναλλακτικά δίνεται ένας αριθμός ο οποίος δηλώνει το μέγεθος ενός Salt που θα δημιουργηθεί αυτόματα από την βιβλιοθήκη. Το αποτέλεσμα της συνάρτησης είναι ένα αλφαριθμητικό το οποίο περιλαμβάνει κωδικοποιημένα τον αλγόριθμο του κατακερματισμού, το Salt και το κατακερματισμένο δεδομένο.

2.6.2 body-parser

Το body-parser είναι μια βιβλιοθήκη της JavaScript η οποία παρέχει ενδιάμεσο λογισμικό (middleware) για την εξαγωγή των δεδομένων από το σώμα ενός HTTP αιτήματος στον server. Η χρήση του είναι πολύ απλή καθώς δηλώνεται κατά την αρχικοποίηση του διακομιστή η μορφή του σώματος που θα εξαχθεί και όταν προκύψει ένα αίτημα εκτελείται η αντίστοιχη μέθοδος και προωθεί τα δεδομένα σαν ένα JavaScript αντικείμενο. Υπάρχουν τέσσερις μορφές body που υποστηρίζονται από την βιβλιοθήκη, το JSON body, το raw body, το Text body και το URL-encoded.

2.6.3 cors

Το cors είναι μια βιβλιοθήκη της JavaScript η οποία ενεργοποιεί την δυνατότητα της κοινής χρήσης πόρων διαφορετικής προέλευσης (Cross-Origin Resource Sharing, CORS) στον server. Με την ενεργοποίηση του cors ο server μπορεί να στείλει απαντήσεις σε αιτήματα από διαφορετικές διευθύνσεις, το οποίο τον καθιστά διαθέσιμο στο διαδίκτυο με την δυνατότητα ελέγχου και περιορισμού συγκεκριμένων διευθύνσεων και την απαίτηση credentials για την απάντηση των ερωτημάτων. Πριν ο server απαντήσει σε ένα ερώτημα, δέχεται ένα OPTIONS αίτημα το οποίο περιλαμβάνει την κεφαλίδα Origin η οποία δηλώνει το domain του application από το οποίο έγινε το αίτημα και αν υποστηρίζεται, ο server απαντάει με την κεφαλίδα Access-Control-Allow-Origin για να ξεκινήσει η αποστολή αιτημάτων. Η βιβλιοθήκη αναλαμβάνει αυτή τη διαδικασία με τη δήλωση των κεφαλίδων κατά την αρχικοποίηση του server ελαχιστοποιώντας έτσι την προσθήκη των κεφαλίδων σε κάθε απάντηση ερωτήματος.

2.6.4 jsonwebtoken

Το jsonwebtoken είναι μια βιβλιοθήκη της Javascript υπεύθυνη για την δημιουργία και την επαλήθευση JSON Web Tokens (JWTs). Ένα jwt είναι ένα αλφαριθμητικό το οποίο περιέχει κρυπτογραφημένες κάποιες πληροφορίες και χρησιμοποιείται στα Single Page Application για την

αυθεντικοποίηση του χρήστη και για την διαχείριση των συνεδριών της εφαρμογής. Στα κρυπτογραφημένα δεδομένα περιέχεται:

- ο αλγόριθμος που χρησιμοποιήθηκε
- ο πάροχος του token
- μια χρονοσφραγίδα που ορίζει την ημερομηνία λήξης της εγκυρότητας του token πέρα από την οποία το token δεν θεωρείται αξιοποιήσιμο
- τα δεδομένα που μπορούν να οριστούν από τον δημιουργό της εφαρμογής τα οποία χρησιμοποιούνται για την αναγνώριση ενός χρήστη ή την αποστολή παραμέτρων μεταξύ των εφαρμογών.

Για την κωδικοποίηση των δεδομένων χρησιμοποιείται ένα κρυφό κλειδί, συνήθως ένα μεγάλο αλφαριθμητικό, το οποίο παραμένει αποθηκευμένο στον server και εισάγεται σε μια συνάρτηση που παρέχει η βιβλιοθήκη για την δημιουργία του token. Το token χορηγείται σε κάθε ερώτημα προς τον server μέσα στην κεφαλίδα Authorization και επαληθεύεται από τον server για να μπορέσει να συνεχιστεί η ολοκλήρωση του αιτήματος. Εάν το token έχει δημιουργηθεί από τον ίδιο server με το κρυφό κλειδί, τότε η διαδικασία της επαλήθευσης είναι επιτυχής και θεωρείται ότι το token έχει χορηγηθεί από τον server. Εάν έχει περάσει η ημερομηνία εγκυρότητας του token τότε η επαλήθευση αποτυγχάνει.

Το JWT αποτελείται από τρία κρυπτογραφημένα μέρη. Το πρώτο είναι οι επικεφαλίδες του, το δεύτερο περιλαμβάνει το σώμα των δεδομένων της εφαρμογής και το τρίτο την υπογραφή του token που επαληθεύεται από τον πάροχο του token. Οι αλγόριθμοι που χρησιμοποιούντα για την κρυπτογράφηση των token είναι ο HMAC με την χρήση της συνάρτησης κατακερματισμού SHA-256 και η RSA υπογραφή πάλι με την χρήση της συνάρτησης SHA-256. Οι κωδικοποιήσεις των αλγορίθμων είναι HS256 και RS256 αντίστοιχα.

2.6.5 Multer

Το multer είναι μια βιβλιοθήκη της JavaScript η οποία παρέχει την δυνατότητα εξαγωγής δεδομένων από ένα HTTP αίτημα που περιέχει αρκετά μέρη (multipart data). Συνήθως αυτά τα αιτήματα περιλαμβάνουν αρχεία τα οποία χωρίζονται σε διαφορετικά πακέτα και χρειάζεται να γίνει η συναρμολόγηση τους στο server. Το multer παρέχει ένα ενδιάμεσο λογισμικό το οποίο αναλαμβάνει την εξαγωγή ενός ή πολλών αρχείων από το HTTP αίτημα και παρέχει χρήσιμες πληροφορίες όπως είναι ο τύπος του, το μέγεθος του ή κωδικοποίησή του στη μορφή ενός JavaScript αντικειμένου. Δίνεται επίσης η δυνατότητα να επιλεγεί η τοποθεσία που θα αποθηκευτεί το αρχείο, αν πρόκειται για αποθήκευση στο χώρο του server ή προσωρινά στη μνήμη, ενώ μπορούν να υλοποιηθούν και φίλτρα που ελέγχουν το αρχείο πριν την αποθήκευση του.

2.6.6 socket.io

Το socket.io είναι μια βιβλιοθήκη ανοιχτού κώδικα βασισμένη στην JavaScript η οποία δίνει την δυνατότητα της υλοποίησης μιας εφαρμογής με την υλοποίηση WebSockets για την αμφίδρομη επικοινωνία μεταξύ του server και του browser πάνω από μια σύνδεση TCP. Το πρωτόκολλο που αξιοποιεί το WebSocket έχει τυποποιηθεί από το IETF το 2011 με RFC 6455 [21]. Το socket.io είναι η βιβλιοθήκη που χρησιμοποιεί ο server για να ανοίξει ένα socket το οποίο περιμένει αιτήματα σύνδεσης από τους browsers που ανοίγουν την εφαρμογή. Όπως και πολλές ακόμη υλοποιήσεις σε JavaScript έτσι και το socket.io λειτουργεί με βάση τα events τα οποία ενεργοποιούνται σε κάποιο function και χαρακτηρίζονται από ένα συγκεκριμένο όνομα. Όταν κληθεί ένα event, η βιβλιοθήκη

στέλνει ένα μήνυμα στο κανάλι που έχει ανοίξει με τον κάθε browser και όταν λάβει ένα μήνυμα με ένα όνομα ενεργοποιεί και το αντίστοιχο συμβάν. Η χρήση του socket.io αντικαθιστά πλήρως την επικοινωνία μέσω HTTP αιτημάτων και δίνει την δυνατότητα της αποστολής μηνυμάτων από τον server στο browser αμφίδρομα, πράγμα που δεν το επιτρέπει η αρχιτεκτονική των REST APIs.

2.6.7 Angular Material

Το Material Design είναι μια βιβλιοθήκη ανοιχτού κώδικα της Google η οποία παρέχει στοιχεία για τον σχεδιασμό γραφικών με έναν ενοποιημένο και τεκμηριωμένο τρόπο και επιτρέπει την υλοποίηση τους μέσω των πιο γνωστών framework και βιβλιοθηκών ανάπτυξης διεπαφών χρήστη για web εφαρμογές. Η βιβλιοθήκη περιέχει έτοιμα συστατικά διεπαφών τα οποία μπορούν να χρησιμοποιηθούν ανεξάρτητα και διευκολύνουν την ανάπτυξη των εφαρμογών καθώς δεν απαιτείται ο προγραμματισμός αυτών εκ νέου για κάθε εφαρμογή.

Η Angular, ένα από τα πιο διαδεδομένα frameworks web εφαρμογών έχει την δική της υλοποίηση για τα Material στοιχεία και η βιβλιοθήκη αυτή είναι γνωστή ως Angular Material. Πρόκειται για κώδικα γραμμένο σε Typescript που υλοποιεί τα στοιχεία αυτά σαν μια ενιαία βιβλιοθήκη έτοιμη προς χρήση σε ένα Angular project. Τα συστατικά της Angular Material αντικαθιστούν εξ ολοκλήρου τα HTML στοιχεία για την αρχιτεκτονική και τον προγραμματισμό της εφαρμογής και είναι δομημένα σαν αυτόνομα Modules της Angular σύμφωνα με την κάθε κατηγορία.

Μερικά από τα χρήσιμα συστατικά της βιβλιοθήκης είναι τα Material Input Fields τα οποία έχουν ενσωματώσει και λειτουργίες έλεγχου εγκυρότητας (validation) των δεδομένων που εισάγονται στις φόρμες, παραμετροποίηση των buttons με χρώματα και διακόσμηση προκαθορισμένη από τον χρήστη καθώς επίσης και έτοιμες αυτόνομες φόρμες (modals) με ευέλικτο χειρισμό των απαντήσεων του χρήστη και δυναμική διαχείριση των παραμέτρων που χορηγούνται για κάθε διαφορετική χρήση.

Η χρήση της Angular Material για την ανάπτυξη των γραφικών στοιχείων της εφαρμογής την καθιστά γρήγορη με την επαναχρησιμοποίηση των συστατικών της, ομοιόμορφη ως προς την αισθητική παρουσίαση και την αλληλεπίδραση του χρήστη καθώς οι λειτουργίες είναι κοινές σε όλα τα επιμέρους συστατικά και αποφυγή λαθών καθώς παρέχονται έτοιμες μέθοδοι για συχνές λειτουργίες που έχουν ήδη αναπτυχθεί και δοκιμαστεί από τους προγραμματιστές της βιβλιοθήκης, δίνοντας έτσι την δυνατότητα στην ενασχόληση μόνο με την υλοποίηση του business logic της εφαρμογής.

2.6.8 Rxjs

Η βιβλιοθήκη rxjs της Typescript είναι μια υλοποίηση της βιβλιοθήκης ανοιχτού κώδικα ReactiveX η οποία παρέχει δυνατότητες ασύγχρονου προγραμματισμού εφαρμογών με την χρήση παρατηρήσιμων ροών δεδομένων (Observable streams). Ένα Observable stream είναι ένα σύνολο δεδομένων το οποίο παρατηρείται συνεχώς για τον εντοπισμό αλλαγών που προκύπτουν στα δεδομένα κατά την ροή εκτέλεσης, επιτρέποντας έτσι στην εφαρμογή να χειρίζεται τις αλλαγές δυναμικά και ασύγχρονα χωρίς την ανάγκη ειδικών μεθόδων για κάθε ξεχωριστό σύνολο δεδομένων.

Η αρχιτεκτονική της rxjs περιλαμβάνει δύο βασικά αντικείμενα και αρκετές υλοποιημένες μεθόδους οι οποίες καλούνται ασύγχρονα όταν συμβαίνουν αλλαγές στα δεδομένα και εκτελούν λειτουργίες που αντιστοιχούν στις μεθόδους των λιστών που παρέχει και η JavaScript. Τα δύο αντικείμενα είναι ο Observer και το Observable τα οποία σε συνδυασμό μεταξύ τους δημιουργούν παρατηρήσιμα αντικείμενα που χειρίζονται τα δεδομένα της εφαρμογής.

Ένας Observer ακολουθεί την γνωστή try-catch-finally ροή εκτέλεσης που παρέχεται στις περισσότερες γλώσσες προγραμματισμού με την παραλλαγή στην ονομασία τους ως next-error-complete. Στο σώμα του next παρέχεται το επόμενο σε σειρά στοιχείο της παρατηρήσιμης λίστας και εκτελείται κάποια αλλαγή στα δεδομένα ή προωθούνται σε κάποιο εξωτερικό αντικείμενο της εφαρμογής. Εάν προκύψει κάποιο error η ροή εκτέλεσης ξεκινάει μέσα στο σώμα του error του Observer ενώ στο τέλος κάθε περίπτωσης εκτελείται ο κώδικας που περιέχει το σώμα του complete κατ' αντιστοιχία με το finally.

Το παρατηρήσιμο αντικείμενο Observable παρέχει μια μέθοδο που λέγεται subscribe και είναι η εφαρμογή ενός Observer κατά τη ροή εκτέλεσης. Μέσα στην subscribe περιγράφεται το σύνολο των μεθόδων next-error-complete και η κλήση αυτής της μεθόδου επιστρέφει ένα αποτέλεσμα το οποίο καλείται Subscription. Ο κύκλος αυτής της εκτέλεσης ολοκληρώνεται συνήθως με την καταστροφή των στοιχείων που περιλαμβάνουν αυτά τα αντικείμενα και την αποδέσμευση των πόρων που αξιοποιούν με την κλήση της μεθόδου unsubscribe στο αντικείμενο Subscription.

Ένα χρήσιμο αντικείμενο της rxjs είναι το BehaviorSubject το οποίο περιλαμβάνει ένα αντικείμενο οποιουδήποτε τύπου ορισμένο από τον χρήστη και του δίνει την δυνατότητα της παρατήρησης από έναν Observer. Για την εφαρμογή του Observer καλείται η μέθοδος subscribe ενώ για την αλλαγή της τιμής του καλείται η μέθοδος next με όρισμα την νέα τιμή του αντικειμένου. Όταν καλείται η next ξεκινάει και η εκτέλεση του σώματος της next μέσα σε κάθε Observer που παρατηρεί αυτό το αντικείμενο. Με αυτόν τον τρόπο επιτυγχάνεται η ασύγχρονη ενημέρωση των τιμών που έχουν συσχετιστεί με την τιμή του Subject.

Οι συναρτήσεις που παρέχει η rxjs εμφωλεύονται πριν την εκτέλεση του subscribe και μπορούν να πραγματοποιήσουν διάφορες λειτουργίες χωρίς να χάνεται η ασυγχρονικότητα της εκτέλεσης. Για την εμφώλευση των συναρτήσεων καλείται η συνάρτηση pipe του παρατηρήσιμου αντικειμένου η οποία διοχετεύει τα δεδομένα σε κάποια συνάρτηση πριν την κλήση της subscribe. Μέσα στην συνάρτηση pipe λοιπόν καλείται κάποια συνάρτηση που εκτελεί μια συγκεκριμένη λειτουργία ή ακολουθία λειτουργιών στα δεδομένα και το αποτέλεσμα της συνάρτησης χορηγείται στην next του Observer.

Μερικές χρήσιμες συναρτήσεις που έχουν χρησιμοποιηθεί και στην υλοποίηση της παρούσας εφαρμογής είναι η map, η debounceTime και η distinctUntilChanged. Η map πολύ απλά εφαρμόζει μια άλλη συνάρτηση σε κάθε ένα από τα δεδομένα που παρατηρούνται, εκτελεί δηλαδή προκαθορισμένο κώδικα για κάθε ένα από τα στοιχεία της λίστας. Οι debounceTime και distinctUntilChanged συνδυάζονται για την ασύγχρονη αλλαγή μια τιμής που προέρχεται από ένα πεδίο της εφαρμογής με σκοπό να αποθηκεύσει την συνολική τιμή της εισόδου αντί για τους μεμονωμένους χαρακτήρες της. Στην περίπτωση εισαγωγής μιας αριθμητικής τιμής από τον χρήστη, με την debounceTime ορίζεται ο χρόνος ο οποίος μεσολαβεί από το ξεκίνημα της πληκτρολόγησης της τιμής μέχρι το διάβασμά της από τον παρατηρητή. Η distinctUntilChanged είναι μια δήλωση ότι η εκτέλεση θα γίνει μόνο όταν τα δεδομένα έχουν δεχτεί κάποια αλλαγή, στην συγκεκριμένη περίπτωση τα δεδομένα εισόδου που είναι κενά πριν την αλλαγή της τιμής από τον χρήστη.

Η rxjs και η ReactiveX γενικότερα είναι μια υλοποίηση με αρκετές δυνατότητες για ασύγχρονο προγραμματισμό και αποτελεί μια εξαιρετική λύση για την διαχείριση της κατάστασης των δεδομένων τόσο στην web εφαρμογή όσο και στον κώδικα του backend καθώς υπάρχουν υλοποιήσεις για αρκετές γλώσσες προγραμματισμού.

2.6.9 socket.io-client

Το socket.io-client είναι μια βιβλιοθήκη βασισμένη στην Typescript, η οποία παρέχει τις μεθόδους για την σύνδεση μιας web εφαρμογής με έναν server που αξιοποιεί την μέθοδο των WebSockets. Παρέχει μια μέθοδο σύνδεσης με τον server δίνοντας σαν όρισμα το URL του και εγκαθιστά μια σύνδεση η οποία αποθηκεύεται σε ένα αντικείμενο για μελλοντική αναφορά. Στην web εφαρμογή δημιουργούνται μέθοδοι οι οποίες καλούν συγκεκριμένα συμβάντα τα οποία στέλνουν μηνύματα στο server ενώ για την λήψη μηνυμάτων από τον server χρησιμοποιούνται αντίστοιχα events τα οποία παρατηρούνται για την αποδοχή ενός μηνύματος από τον server και την προώθηση των δεδομένων στην εφαρμογή. Το socket.io-client παρέχει έτοιμες μεθόδους που αντιστοιχούν στην υλοποίηση του socket.io εξαλείφοντας έτσι την ανάγκη δημιουργίας και διαχείρισης την σύνδεσης από τον ίδιο τον προγραμματιστή.

2.6.10 jwt-decode

Το jwt-decode είναι μια βιβλιοθήκη που επιτρέπει την αποκωδικοποίηση των Json Web Tokens (JWTs) με σκοπό την χρήση τους στην εφαρμογή. Χρησιμοποιείται από την Web εφαρμογή για να εξάγει τα χρήσιμα δεδομένα από το token που έχουν αποσταλεί από τον server και συνήθως αφορούν πληροφορίες για τον χρήστη όπως ο ρόλος του, το αναγνωριστικό του ή η εφαρμογή στην οποία του δίνεται πρόσβαση μέσω του συγκεκριμένου token.

2.7 Εργαλεία

Σε αυτό το υποκεφάλαιο θα παρουσιαστούν τα εργαλεία που χρησιμοποιήθηκαν για την σχεδίαση, την ανάπτυξη, την δοκιμή, την τεκμηρίωση και την αποθήκευση του κώδικα της εφαρμογής. Περιλαμβάνει εργαλεία σχεδίασης διεπαφών, επεξεργαστές κειμένου κώδικα και σχετικών προσθηκών, την πλατφόρμα εξυπηρετητή της βάσης δεδομένων και την πλατφόρμα αποθήκευσης και ελέγχου έκδοσης του κώδικα.

2.7.1 Figma

Το Figma είναι μια διαδικτυακή πλατφόρμα σχεδίασης διεπαφών χρήστη με πληθώρα εικονιδίων και προσθηκών για την δημιουργία γραφικών συστατικών και παρουσίασης σελίδων. Ένα από τα βασικά χαρακτηριστικά του είναι η χρήση διανυσματικών γραφικών και η συλλογική επεξεργασία τους από αρκετούς χρήστες. Το Figma χρησιμοποιείται για την σχεδίαση πρωτοτύπων διεπαφών και την βελτίωση της εμπειρίας του χρήστη πάνω στην ίδια περιοχή με εξαιρετική ευκολία στην δημιουργία συστατικών στοιχείων και την ομαδοποίησή τους για την προσομοίωση ολοκληρωμένων γραφικών. Παρέχει επίσης την δυνατότητα δημιουργίας διαγραμμάτων για την ανάλυση των περιπτώσεων χρήσης μιας εφαρμογής και της αρχιτεκτονικής συστημάτων και μέσω εικόνων που παρέχονται από τον χρήστη.

2.7.2 Visual Studio Code

Το Visual Studio Code (VSCode) είναι ένας επεξεργαστής κειμένου ανοιχτού κώδικα που έχει αναπτυχθεί από την Microsoft βασισμένος στο ευρέως διαδεδομένο περιβάλλον ανάπτυξης κώδικα της Microsoft Visual Studio. Παρέχει μεταξύ άλλων προσθήκες για την αναγνώριση των περισσότερων γνωστών γλωσσών προγραμματισμού καθώς και εργαλεία αποσφαλμάτωσης και

ενσωμάτωσης φυλλομετρητών στον ίδιο τον επεξεργαστή. Το VSCode είναι ένα αρκετά διαδεδομένο πρόγραμμα ανάπτυξης κώδικα καθώς είναι ιδιαίτερα ελαφρύ κατά την εκτέλεσή του, έχει φιλική διεπαφή προς τον χρήστη και υποστηρίζει αρκετά εξωτερικά εργαλεία για άλλες πλατφόρμες όπως το GitHub για τον έλεγχο έκδοσης του κώδικα ή επεκτάσεις για την Angular και το Node.js καθώς επίσης και ενσωμάτωση γραμμής εντολών.

2.7.3 ElephantSQL

Το ElephantSQL είναι μια πλατφόρμα εξυπηρέτησης της ανοιχτού κώδικα βάσης δεδομένων PostgreSQL σαν μια υπηρεσία (Database-as-a-Service). Με τον όρο υπηρεσία εννοείται ότι η χρήση της βάσης είναι δυνατή μέσω της πλατφόρμας παρέχοντας μια σύνδεση σε αυτή και πραγματοποιώντας τα SQL queries απομακρυσμένα με την υπηρεσία να τα δέχεται σαν αιτήματα πάνω από την τρέχουσα σύνδεση. Με την είσοδο του χρήστη στην πλατφόρμα είναι δυνατή η είσοδος σε ένα διαμοιραζόμενο στιγμιότυπο της PostgreSQL και η σύνδεση με μια ξεχωριστή βάση στην οποία δίνεται πρόσβαση μέσω ενός URL και ενός κωδικού που έχει ορίσει ο χρήστης.

Το ElephantSQL παρέχει την πρόσβαση σε μια βάση δωρεάν για την αποθήκευση δεδομένων έως 20 Megabyte ενώ για την επέκταση του χώρου αποθήκευσης ή της δημιουργίας μεμονωμένων στιγμιότυπων της βάσης για κάθε χρήστη υπάρχουν και κλιμακούμενες χρεώσεις της υπηρεσίας. Για την παρούσα υλοποίηση, χρησιμοποιήθηκε η συγκεκριμένη υπηρεσία για την ευέλικτη ανάπτυξη της εφαρμογής με την χρήση της βάσης εκτός του τοπικού μηχανήματος και με στόχο την δοκιμή της με μια απομακρυσμένη βάση προσομοιώνοντας έτσι και το εύρος ζώνης της χρήσης της εφαρμογής στο διαδίκτυο.

2.7.4 PG Admin

Το PGAdmin είναι μια ανοιχτού κώδικα εφαρμογή διαχείρισης της βάσης δεδομένων PostgreSQL που χρησιμοποιείται για την ανάπτυξη βάσεων δεδομένων και εφαρμογών βασισμένων στην PostgreSQL. Μερικά από τα κύρια χαρακτηριστικά της είναι ότι μπορεί να συνδεθεί σε απομακρυσμένες βάσεις, να εκτελέσει queries και να διαχειριστεί τα δεδομένα των πινάκων με φιλικό προς τον χρήστη τρόπο. Επίσης, παρέχει και την δυνατότητα εξαγωγής διαγραμμάτων των συσχετίσεων των πινάκων με διαδραστικό τρόπο καθώς και την διαχείριση των χρηστών που έχουν πρόσβαση στη βάση.

2.7.5 Git

Το Git είναι ένα εργαλείο ελέγχου έκδοσης κώδικα το οποίο αναπτύχθηκε από τον Linus Torvalds, δημιουργό του πυρήνα του Linux, ενός από τα πιο διάσημα λειτουργικά συστήματα. Το Git αποθηκεύει τον κώδικα σαν ένα κατανεμημένο δέντρο πάνω στο οποίο αρκετοί χρήστες μπορούν να κάνουν αλλαγές ενώ δημιουργούνται αντίγραφα του κώδικα από τον βασικό κορμό και αποθηκεύονται σαν ξεχωριστά κλαδιά του δέντρου (branches). Οι μεμονωμένες αλλαγές του κάθε χρήστη καταγράφονται και αποθηκεύονται στο δέντρο υπό την μορφή ενός commit.

Ένα commit αντιπροσωπεύει ένα σύνολο αλλαγών τα οποία διακρίνονται με ένα ξεχωριστό αναγνωριστικό. Σε συνδυασμό με ένα σύστημα που φιλοξενεί τον κώδικα σαν ένα αποθετήριο (repository) πραγματοποιείται η ευέλικτη επεξεργασία του κώδικα συνδυάζοντας τα commits σε ομάδες και αντιπροσωπεύουν μια συγκεκριμένη έκδοση του κώδικα. Τα branches χρησιμοποιούνται για να αντιγράψουν την τρέχουσα κατάσταση του κώδικα και να απομονώσουν τις αλλαγές του κάθε χρήστη για να αποφευχθούν οι συγκρούσεις (conflicts), δηλαδή οι αλλαγές στις ίδιες γραμμές κώδικα στα ίδια αρχεία από πολλαπλούς χρήστες.

Όταν χρειαστεί μια ενσωμάτωση των commits των διαφορετικών χρηστών μπορεί να γίνει merge δύο ή περισσότερων branches ενώ η τρέχουσα κατάσταση αποθηκεύεται σε ένα branch που καλείται master. Για τον συγχρονισμό των αλλαγών στο αποθετήριο από το τοπικό μηχάνημα ενός χρήστη αντιγράφονται οι αλλαγές και αποστέλλονται στο remote branch μέσω μιας διαδικασίας που ονομάζεται push. Με την διαδικασία του push, τα commits του κάθε χρήστη και το branch στο οποίο πραγματοποιούνται αποθηκεύονται στο repository και μπορούν να ελεγχθούν ή να ενσωματωθούν με τις αλλαγές των υπόλοιπων χρηστών.

Για την ενσωμάτωση αλλαγών άλλων χρηστών τοπικά, ένας χρήστης εκτελεί την λειτουργία του git που ονομάζεται pull και είναι ο συγχρονισμός των τοπικών αλλαγών σε ένα branch με τις αλλαγές που βρίσκονται αποθηκευμένες στο repository. Ένα branch το οποίο δεν υπάρχει τοπικά μπορεί να ανακτηθεί ανά πάσα στιγμή με την λειτουργία fetch. Το fetch πρακτικά λαμβάνει τις πληροφορίες για όλα τα branches τα οποία διαθέτει το αποθετήριο και δεν υπάρχουν στο τοπικό μηχάνημα του χρήστη. Για την εναλλαγή των branches χρησιμοποιείται η λειτουργία γνωστή ως checkout. Με το checkout εμφανίζονται στον χρήστη τα commits τα οποία περιγράφουν το κάθε branch ξεχωριστά. Με τον συνδυασμό των παραπάνω λειτουργιών επιτυγχάνεται η διαχείριση των αλλαγών του κώδικα από τους χρήστες και η συλλογική επεξεργασία του.

Το git είναι ένα σύγχρονο εργαλείο διαχείρισης έκδοσης κώδικα και υπάρχουν αρκετές πλατφόρμες που υποστηρίζουν την αποθήκευση αποθετηρίων με βάση αυτό. Μια γνωστή πλατφόρμα είναι το GitHub, στο οποίο οι χρήστες αποθηκεύουν τις αλλαγές του κώδικα σε συνδυασμό με την παροχή διαπιστευτηρίων για τους χρήστες με σκοπό την ασφάλεια της πρόσβαση στο αποθετήριο και τον έλεγχο των λειτουργιών των χρηστών από τους διαχειριστές της ομάδας που αναπτύσσει τον κώδικα. Επειδή το git είναι ένα εργαλείο κονσόλας που ελέγχει απομακρυσμένα ένα αποθετήριο, έχουν αναπτυχθεί αρκετά εργαλεία με στόχο την γραφική αναπαράσταση των branches και την ευέλικτη εκτέλεση των λειτουργιών του από τους προγραμματιστές.

2.7.6 Swagger

Το Swagger είναι ένα εργαλείο ανοιχτού κώδικα το οποίο χρησιμοποιείται για την τεκμηρίωση και την δοκιμή των RESTful APIs. Το εργαλείο βασίζεται στην μεθοδολογία προσδιορισμού OpenAPI με την οποία περιγράφεται ένα REST API με την μορφή αρχείου YAML ή JSON με βάση τις HTTP μεθόδους (GET, POST, PUT, DELETE) και τα endpoints που παρέχονται και των HTTP απαντήσεων που μπορούν να δοθούν. Το Swagger δημιουργεί ένα γραφικό περιβάλλον web με το σύνολο των endpoints σαν λίστα, ενώ δίνει την δυνατότητα αποστολής αιτημάτων σε αυτά με την χρήση των κατάλληλων κεφαλίδων και παραμέτρων. Για την καλύτερη σχεδίαση και εκτέλεση του API μπορούν να μοντελοποιηθούν ως αντικείμενα οι πίνακες της βάσης και να δοθούν ως ορίσματα στο σώμα κάθε αιτήματος.

Κεφάλαιο 3ο: Σχεδίαση και Υλοποίηση της εφαρμογής

Στο παρόν κεφάλαιο, θα δοθεί μία αναλυτική περιγραφή για την σχεδίαση και την υλοποίηση του ολοκληρωμένου συστήματος που αναπτύχθηκε. Πιο συγκεκριμένα, ξεκινώντας με τις λειτουργικές απαιτήσεις, οι οποίες περιγράφουν την συμπεριφορά του συστήματος, θα γίνει ο προσδιορισμός των απαιτήσεων για όλη την λειτουργικότητα. Στην συνέχεια, θα εξηγηθεί η αρχιτεκτονική της εφαρμογής και των επιμέρους τμημάτων του λογισμικού. Θα παρουσιαστούν τόσο η δομή του συστήματος όσο και οι σχέσεις μεταξύ των δομικών στοιχείων της εφαρμογής. Ακολουθεί η παρουσίαση της βάσης δεδομένων που σχεδιάστηκε και υλοποιήθηκε για την αποθήκευση των δεδομένων και την άντληση αυτών κατά την χρήση της εφαρμογής. Θα παρουσιαστεί επίσης και ο τρόπος με τον οποίο το σύστημα ταυτοποιεί τους εξουσιοδοτημένους χρήστες για την είσοδο τους σε αυτό και εξασφαλίζει την ασφάλεια της εφαρμογής. Τέλος, θα αναλυθεί όλη η υλοποίηση για το μοντέλο πελάτη και εξυπηρετητή που εφαρμόστηκαν και θα αναφερθεί και η αποθήκευση της στην διαδικτυακή πλατφόρμα του GitHub..

3.1 Λειτουργικές απαιτήσεις

Στο παρόν υποκεφάλαιο θα γίνει μια ανάλυση των λειτουργικών απαιτήσεων της εφαρμογής και μια σύνοψη για την υπηρεσία που παρέχεται συνολικά από την παρούσα υλοποίηση. Για την ολοκληρωμένη ανάλυση του είναι σημαντικό αρχικά να προσδιορίσουμε τι είναι μια απαίτηση από το σύστημα, ποια είναι τα επιμέρους στοιχεία του, τι σκοπούς εκπληρώνουν αυτά τα στοιχεία και πως επιτυγχάνεται η αλληλεπίδραση με τους χρήστες του συστήματος για την λύση του προβλήματος. Σε συνδυασμό με κάποια διαγράμματα που αναπαριστούν την αρχιτεκτονική του συστήματος και τη ροή της πληροφορίας θα δοθεί μια ολοκληρωμένη ανάλυση της εφαρμογής.

Ο ορισμός για μια απαίτηση στα πληροφοριακά συστήματα είναι μια πληροφορία η οποία περιγράφει τι πρέπει να κάνει ένα σύστημα, δηλαδή ποιες διεργασίες πρέπει να εκτελεστούν προκειμένου να υπάρχει η λύση σε ένα πρόβλημα. Οι απαιτήσεις μπορεί να είναι είτε λειτουργικές και να αφορούν τις υπηρεσίες που παρέχει το σύστημα είτε ποιοτικές και να περιγράφουν τις τεχνικές λεπτομέρειες που εξασφαλίζουν την απόκριση του συστήματος χρονικά, την διαχείριση των πόρων και την διαθεσιμότητά του. Οι λειτουργικές απαιτήσεις είναι το πρώτο στάδιο της ανάλυσης και είναι απαραίτητο για να τεθούν τα όρια της υλοποίησης της εφαρμογής αφού οριστούν με ακρίβεια οι επιθυμητές ενέργειες που πρέπει να εκτελεί το σύστημα για να καταστεί λειτουργικό.

Οι λειτουργικές απαιτήσεις μπορούν να χωριστούν σε κατηγορίες για την ανάλυσή τους με βάση διαφορετικές πτυχές του συστήματος. Συνοπτικά χωρίζονται ως εξής:

- Οι είσοδοι που δέχεται το σύστημα, τα δεδομένα δηλαδή που παρέχει ο χρήστης.
- Οι έξοδοι του συστήματος, δηλαδή τα αποτελέσματα που παράγει από τις ενέργειες που εκτελεί και η εμφάνισή τους σε μορφή κατανοητή από τον χρήστη.
- Τα δεδομένα που αποθηκεύονται στο ίδιο το σύστημα και μπορούν να χρησιμοποιηθούν από άλλα συστήματα. Εδώ δεν περιλαμβάνονται δεδομένα που χρησιμοποιούνται από το ίδιο το σύστημα, όπως οι μεταβλητές του περιβάλλοντος ή κάποια τεχνική πληροφορία της υλοποίησης.
- Οι υπολογισμοί που πρέπει να εκτελέσει το σύστημα. Πρόκειται για τις ενέργειες που γίνονται αντιληπτές από τους χρήστες χωρίς τις τεχνικές λεπτομέρειες της υλοποίησης.
- Ο χρονισμός και ο συγχρονισμός των δεδομένων και των υπολογισμών. Η συγκεκριμένη λειτουργική απαίτηση αφορά τον τρόπο με τον οποίο η διαθεσιμότητα των δεδομένων

επιτρέπει την σωστή εκτέλεση των υπολογισμών με σκοπό την εξαγωγή σωστών αποτελεσμάτων.

Με βάση αυτές τις κατηγορίες λειτουργικών απαιτήσεων μπορούν να οριστούν οι απαιτήσεις της παρούσας εφαρμογής και η σύνδεση μεταξύ των συστατικών μερών για την επεξήγηση των υπολογισμών και του συγχρονισμού.

Οι είσοδοι στο παρόν σύστημα παραγγελιοληψίας είναι τα δεδομένα των παραγγελιών που εισάγονται από τον παραγγελιολήπτη κατά την δημιουργία της παραγγελίας και τα δεδομένα του εστιατορίου και η οργάνωση της επιχείρησης από την πλευρά του διαχειριστή, όπως τα στοιχεία των υπαλλήλων, η οργάνωση των τραπεζιών και το λογότυπο της εφαρμογής. Ως είσοδος στο σύστημα θεωρούνται επίσης και οι ενέργειες των χρηστών που αλλάζουν την κατάσταση της παραγγελιάς όπως είναι η ακύρωση ή το κλείσιμό της.

Ως έξοδοι του συστήματος μπορούν να οριστούν τα αποτελέσματα του συνόλου των παραγγελιών που εμφανίζονται αναλυτικά κατά το κλείσιμο της ημέρας με τα αναλυτικά ποσά των παραγγελιών και η εμφάνιση της κάθε παραγγελίας ξεχωριστά, σε κάθε στάδιο της εξέλιξής της.

Τα δεδομένα που αποθηκεύονται είναι τα αντικείμενα που περιγράφονται από τους πίνακες της βάσης και περιλαμβάνουν όλα τα στοιχεία που περιγράφουν τα δεδομένα της εφαρμογής και μπορούν να αποθηκευτούν και να χρησιμοποιηθούν από το σύστημα, να εμφανιστούν στο χρήστη ή να συλλεχθούν για την δημιουργία μιας αναφοράς αν πρόκειται για τα δεδομένα των παραγγελιών ή των υπαλλήλων. Τα δεδομένα μοντελοποιούνται με βάση το φυσικό αντικείμενο το οποίο περιγράφουν για να γίνει πιο εύκολη η διαχείρισή του από το σύστημα τόσο σαν αντικείμενο της γλώσσας προγραμματισμού όσο και ως μια λογική οντότητα που αποτελεί μέρος του συστήματος.

Το βασικό δεδομένο της εφαρμογής πάνω στο οποίο εκτελούνται οι περισσότερες λειτουργίες της είναι αυτό της παραγγελίας (Order). Η παραγγελία περιέχει πληροφορίες για τον υπάλληλο ο οποίος πραγματοποίησε την λήψη της, την ημερομηνία και ώρα της καταγραφής της στο σύστημα, το συνολικό ποσό των προϊόντων της παραγγελίας, οι οδηγίες προς τον παραγγελιοδέκτη και η τρέχουσα κατάστασή της. Τα προϊόντα της παραγγελίας έχουν επίσης ένα σύνολο από χαρακτηριστικά ανάλογα με το είδος τους ενώ μαζί αποθηκεύεται και η ποσότητα του κάθε προϊόντος και η τρέχουσα κατάστασή του. Η τρέχουσα κατάσταση των παραγγελιών και των προϊόντων είναι μια προκαθορισμένη κατάσταση και μπορεί να είναι μια από τις ακόλουθες: Νέα, Ανοιχτή, Ακυρωμένη, Έτοιμη και Κλειστή. Οι ονομασίες των καταστάσεων δεν αποθηκεύονται από το σύστημα στην βάση αλλά δημιουργούνται και αποθηκεύονται κατά την εκτέλεση του προγράμματος σε ένα στατικό αντικείμενο στην μνήμη της εφαρμογής.

Ένα άλλο δεδομένο της εφαρμογής είναι το προϊόν, όχι σαν μέρος της παραγγελίας αλλά σαν ένα περιεχόμενο του μενού της επιχείρησης. Τα προϊόντα αποθηκεύονται με τον τίτλο τους, την τιμή πώλησης, τα συστατικά ή το περιεχόμενό τους και μια ένδειξη ότι είναι διαθέσιμα την τρέχουσα στιγμή στον κατάλογο. Επίσης για την καλύτερη διαχείριση τους κατηγοριοποιούνται σε διακριτές και ορισμένες από τον χρήστη κατηγορίες με το μοντέλο της κατηγορίας να περιλαμβάνει τον τύπο του προϊόντος, μια χαρακτηριστική ονομασία και μια ένδειξη για την διαθεσιμότητα ολόκληρης της κατηγορίας.

Για την οργάνωση και την διαχείριση των τραπεζιών του εστιατορίου καθώς και των κρατήσεων είναι απαραίτητη και η αποθήκευσή τους στη βάση για μετέπειτα αναφορά. Ένα τραπέζι διαθέτει αριθμό θέσεων, ένα αναγνωριστικό για τους υπαλλήλους και τον διαχειριστή και την τοποθεσία του στον

χώρο του εστιατορίου που μεταφράζεται στην τοποθεσία του πάνω στην δισδιάστατη αναπαράσταση του χώρου της εστίασης. Το μοντέλο της κράτησης παρέχει πληροφορίες για την ημερομηνία της κράτησης και ένα όνομα καταχώρησης.

Για την διαχείριση των χρηστών και την πιστοποίηση τους από τον εξυπηρετητή κατά την εκτέλεση της εφαρμογής χρειάζεται και ένα μοντέλο του χρήστη με όλες τις απαραίτητες πληροφορίες που εισάγονται είτε από τον διαχειριστή κατά την δημιουργία ενός νέου χρήστη αλλά και από το ίδιο το σύστημα κατά την κρυπτογράφηση των κωδικών. Ένας χρήστης έχει ένα χαρακτηριστικό όνομα χρήστη (username), διατίθεται επίσης το όνομα και το επίθετο του, ο ρόλος του στο σύστημα, η ημερομηνία πρόσληψης του και ο κωδικός που χρησιμοποιεί για την είσοδό του στις υπηρεσίες, ο οποίος αποθηκεύεται κωδικοποιημένος για λόγους ασφαλείας.

Οι περισσότερες λειτουργίες του συστήματος εκτελούνται από τους χρήστες με βάση τον ρόλο τους ως υπάλληλοι της επιχείρησης. Στις λειτουργικές απαιτήσεις περιλαμβάνονται όλες οι λειτουργίες των υπαλλήλων για κάθε ρόλο και κάθε περίπτωση ξεχωριστά ενώ η πρόσβαση των χρηστών στα δεδομένα περιορίζεται σε κάθε ενέργεια από τον ίδιο ρόλο. Συνοπτικά οι ρόλοι είναι τρεις, αυτοί του παραγγελιολήπτη σερβιτόρου (server), του παραγγελιοδέκτη στην παραγωγή (cook) και του διαχειριστή του συστήματος (admin). Αξίζει να σημειωθεί εδώ ότι ο διαχειριστής περιορίζεται στις λειτουργίες του συστήματος στις οποίες έχει πρόσβαση και δεν αναλαμβάνει και ρόλους διαχείρισης της υποδομής του συστήματος όπως η διαχείριση της βάσης ή η φιλοξενία της εφαρμογής.

Ως σερβιτόρος, ο χρήστης έχει πρόσβαση στο σύνολο των τραπεζιών της επιχείρησης σε μια διαδραστική αναπαράσταση απ’ όπου μπορεί να επιλέξει ένα τραπέζι για να επεξεργαστεί την κατάσταση μιας παραγγελίας. Με την επιλογή ενός τραπεζιού του δίνεται η δυνατότητα εμφάνισης του μενού του εστιατορίου με βάση τις κατηγορίες στις οποίες έχει οργανωθεί και μόνο των στοιχείων που είναι διαθέσιμα την χρονική στιγμή της λήψης της παραγγελίας. Ο σερβιτόρος μπορεί να επιλέξει τα προϊόντα από τον κατάλογο και όταν η παραγγελία είναι έτοιμη μπορεί να την καταχωρήσει στο σύστημα για να προχωρήσει στο επόμενο στάδιο της προετοιμασίας.

Άλλες δυνατότητες που δίνονται στο σερβιτόρο είναι η διαγραφή προϊόντων από την λίστα της παραγγελίας, η προσθήκη ειδικών οδηγιών για την προετοιμασία της και η επιλογή των προϊόντων με βάση τα ιδιαίτερα χαρακτηριστικά τους. Ο σερβιτόρος είναι υπεύθυνος αποκλειστικά για την δημιουργία ή την ακύρωση μιας παραγγελίας ενώ με την αποχώρηση του πελάτη μπορεί να καταχωρήσει την παραγγελία στο σύστημα ως κλειστή. Η διαφορά μια κλειστής παραγγελίας από μια έτοιμη ή ανοιχτή παραγγελία είναι ότι ο σερβιτόρος δεν έχει πλέον πρόσβαση στα δεδομένα της παραγγελίας και δεν μπορεί να την τροποποιήσει με κανέναν τρόπο.

Ο παραγγελιοδέκτης έχει πρόσβαση σε μία μόνο σελίδα της εφαρμογής στην οποία εμφανίζονται όλες οι ανοιχτές παραγγελίες, οι παραγγελίες δηλαδή που έχουν προϊόντα που δεν έχουν ετοιμαστεί από την παραγωγή. Οι δύο ενέργειες στις οποίες μπορεί να προβεί ένας παραγγελιοδέκτης είναι οι ακύρωση ή η καταχώρηση μιας παραγγελίας ως έτοιμη. Αυτή είναι η μόνη αρμοδιότητα που δίνεται σε έναν παραγγελιοδέκτη από το σύστημα ενώ κατά την εμφάνιση της παραγγελίας δεν εμφανίζεται η τιμή των προϊόντων ή η συνολική τιμή της παραγγελίας.

Σε αντίθεση με τους άλλους δύο ρόλους του συστήματος ο διαχειριστής έχει την δυνατότητα να επεξεργαστεί δεδομένα του συστήματος μέσω έξι διαφορετικών συνόλων ενεργειών. Παρόλο που έχει πρόσβαση στα δεδομένα, ο διαχειριστής δεν μπορεί να ανοίξει τις σελίδες των υπόλοιπων ρόλων για λόγους διαφάνειας της χρήσης της εφαρμογής. Με αυτόν τον τρόπο ένας διαχειριστής δεν μπορεί να

τροποποιήσει τα δεδομένα ή να δημιουργήσει παραγγελίες ή να παρέμβει με οποιονδήποτε τρόπο στην διαδικασία εξέλιξης της παραγγελίας.

Μια βασική ενέργεια που απολαμβάνει ένας διαχειριστής είναι η εμφάνιση των παραγγελιών που έχουν κλείσει με το συνολικό ποσό τους καθώς και την εμφάνιση του συνολικού ποσού που εισπράχθηκε από τις παραγγελίες για μια δεδομένη ημέρα. Μπορεί να περιηγηθεί σε διαφορετικές μέρες και να συγκρίνει τα ποσά των παραγγελιών ή να ελέγξει αν κάποια παραγγελία έχει ακυρωθεί.

Μια άλλη ενέργεια του χρήστη σαν διαχειριστή είναι η προσθήκη νέων και η διαγραφή ήδη καταχωρημένων τραπεζιών στο σύστημα. Ο διαχειριστής μπορεί κατά την δημιουργία να εισάγει το αναγνωριστικό του τραπεζιού και τον αριθμό των θέσεων του. Η επιλογή της τοποθεσίας του τραπεζιού στην απεικόνιση του εστιατορίου ορίζεται κατά την δημιουργία του και δεν είναι δυνατή η τροποποίησή της. Σε περίπτωση αλλαγής της τοποθεσίας του ο διαχειριστής μπορεί να το διαγράψει εξ ολοκλήρου και να το τοποθετήσει εκ νέου στην επιθυμητή τοποθεσία.

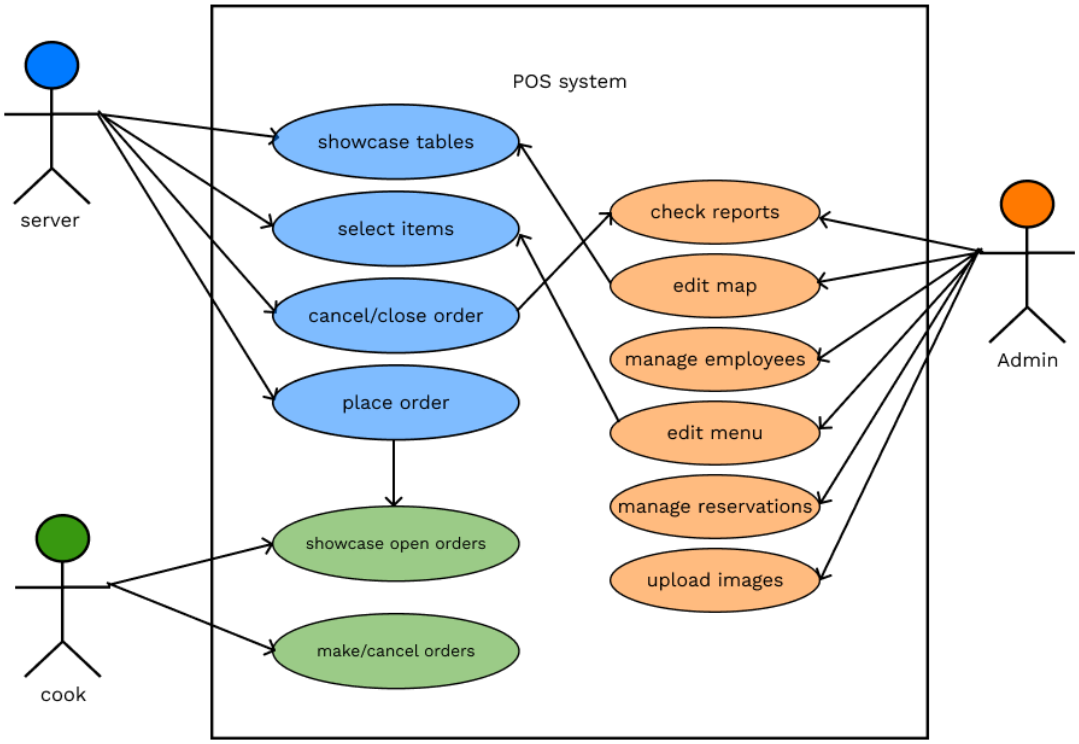
Για την διαχείριση των χρηστών του συστήματος παρέχεται στον διαχειριστή μια σελίδα στην οποία φαίνονται όλοι οι ενεργοί χρήστες του συστήματος με τις πληροφορίες τους εκτός από τον κωδικό και επιτρέπεται η προσθήκη ενός νέου χρήστη, η τροποποίηση των στοιχείων του ή η διαγραφή του από την βάση της εφαρμογής. Η αλλαγή του κωδικού του χρήστη είναι εφικτή χωρίς να φαίνεται ο παλιός κωδικός απλά τροποποιώντας τα στοιχεία του και εισάγοντας έναν νέο κωδικό ο οποίος θα αντικαταστήσει αυτόματα τον παλιό.

Για την τροποποίηση του καταλόγου των προϊόντων του εστιατορίου παρέχεται στον διαχειριστή μια ξεχωριστή σελίδα στην οποία εμφανίζεται το μενού με τις κατηγορίες και τα προϊόντα όπως έχουν οριστεί από τον ίδιο τον χρήστη. Τα προϊόντα ανήκουν όλα σε μια και μόνο κατηγορία οπότε για την προσθήκη ενός προϊόντος απαιτείται πρώτα η δημιουργία μιας νέας κατηγορίας. Ο χρήστης μπορεί να ορίσει μια χαρακτηριστική ονομασία για την κατηγορία και έναν τύπο κατηγορίας, όπου ο κάθε τύπος δίνει την δυνατότητα της επιλογής διαφορετικών χαρακτηριστικών στο προϊόν κατά την καταχώρηση της παραγγελίας. Τα χαρακτηριστικά αυτά είναι προκαθορισμένα από το σύστημα με τον διαχειριστή να μπορεί να επιλέξει μόνο κάποια κατηγορία χαρακτηριστικών. Είναι δυνατή επίσης η σήμανση μιας κατηγορίας ως διαθέσιμη για την εμφάνιση της στο μενού.

Εφόσον υπάρχει μια κατηγορία, ο διαχειριστής μπορεί να προσθέσει ένα νέο προϊόν με ένα χαρακτηριστικό όνομα, την τιμή που θα εμφανίζεται στον κατάλογο, μια κατηγορία που έχει ήδη δημιουργηθεί από τον ίδιο, τα συστατικά του προϊόντος και την σήμανση του προϊόντος ως διαθέσιμο στον κατάλογο. Ο διαχειριστής μπορεί να τροποποιήσει οποιοδήποτε από αυτά τα πεδία ή να διαγράψει εξ ολοκλήρου το προϊόν.

Το σύστημα παρέχει επίσης την δυνατότητα της κράτησης ενός τραπεζιού για κάποιον πελάτη. Σε αυτή τη λειτουργία ο διαχειριστής είναι υπεύθυνος για τον έλεγχο των κρατήσεων καθώς μπορεί να καταχωρήσει μια νέα κράτηση με το όνομα του πελάτη, την ημερομηνία της κράτησης και το τραπέζι το οποίο θα παραχωρηθεί στον πελάτη. Οι κρατήσεις καταχωρούνται και εμφανίζονται με βάση την ημέρα για την οποία γίνεται η επιθυμητή κράτηση του τραπεζιού. Είναι δυνατή η τροποποίηση όλων των πεδίων της κράτησης ενώ μπορεί και να διαγραφεί εξ ολοκλήρου.

Σχετικά με την εμφάνιση της εφαρμογής και την παραμετροποίηση της για κάθε εστιατόριο παρέχεται η δυνατότητα αλλαγής του λογοτύπου το οποίο εμφανίζεται στην αρχική σελίδα της εφαρμογής καθώς επίσης και του εικονιδίου που αναπαριστά ένα τραπέζι στην απεικόνιση του εστιατορίου. Αυτό επιτυγχάνεται με το ανέβασμα των δύο αυτών εικόνων ενώ η αλλαγή τους γίνεται με το ανέβασμα μιας καινούργιας εικόνας και την αυτόματη αντικατάσταση της παλιάς από το σύστημα.



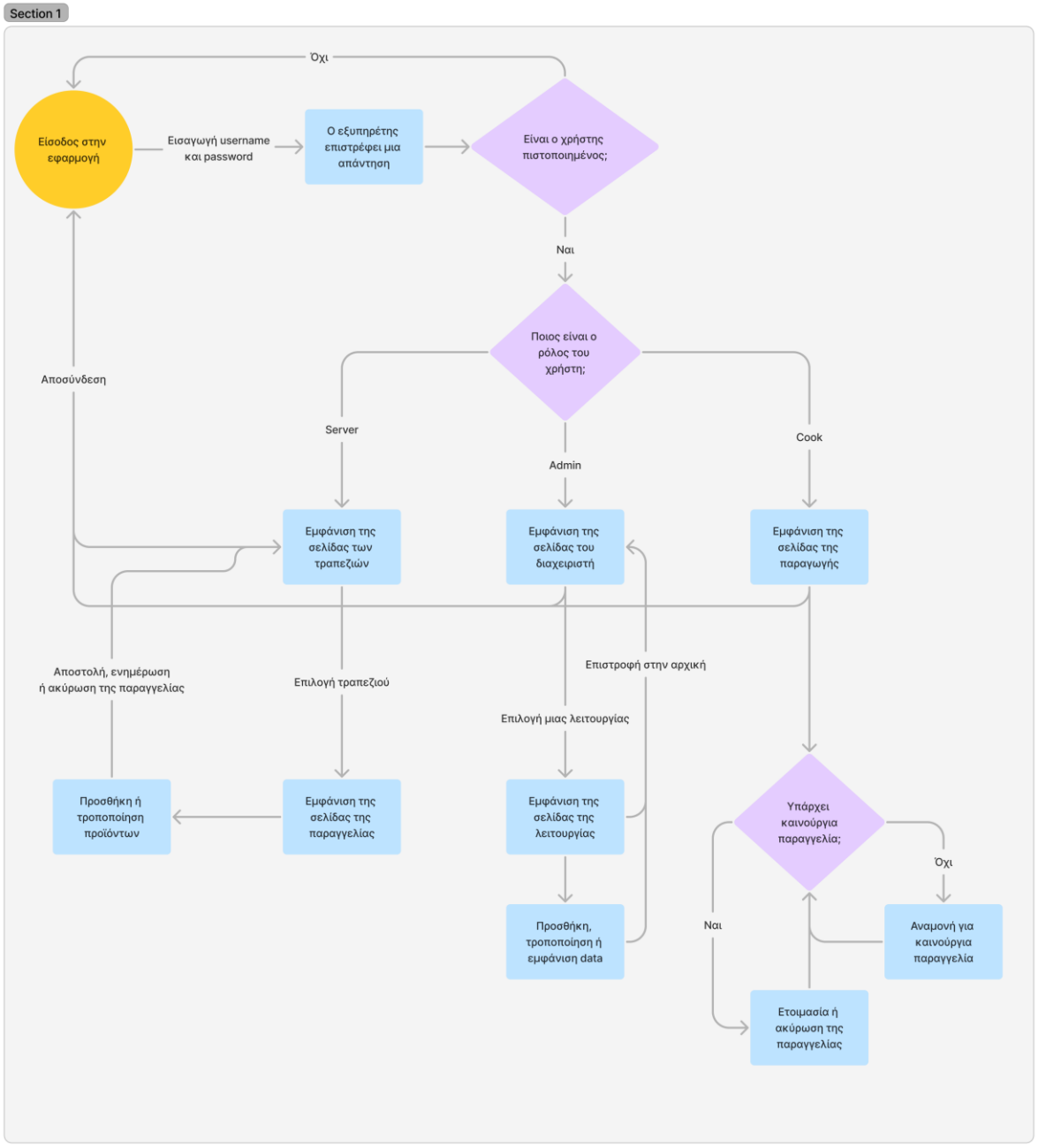
Σχήμα 3.1: Διάγραμμα περιπτώσεων χρήσης συστήματος παραγγελιοληψίας

Για την τελευταία κατηγορία λειτουργικών απαιτήσεων που αφορά τον συγχρονισμό των δεδομένων είναι σημαντικό να αναφερθεί ότι το σύστημα πρέπει να αποθηκεύει την κατάσταση των παραγγελιών σε κάθε χρονική στιγμή έτσι ώστε όταν εμφανίζονται ή τροποποιούνται από τους χρήστες, τα δεδομένα να έχουν τις σωστές τιμές και να είναι δυνατές οι αλλαγές χωρίς να προκύπτουν λογικά λάθη ή η απρόσμενη κατάρρευση του συστήματος λόγω εσφαλμένων δεδομένων. Στην περίπτωση του παρόντος συστήματος παραγγελιοληψίας, τα δεδομένα που πρέπει να ληφθούν υπόψη ως προς την διαθεσιμότητά τους σε κάθε χρονική στιγμή για κάθε εμπλεκόμενο χρήστη είναι οι πληροφορίες των κατηγοριών και των προϊόντων του καταλόγου, τα δεδομένα της κάθε παραγγελίας, και τα τραπέζια του εστιατορίου και η απεικόνιση τους.

Για την εξασφάλιση της σωστής διαχείρισης των παραγγελιών, λαμβάνεται υπόψη το πεδίο της κατάστασης της παραγγελίας καθώς με βάση αυτό οι χρήστες σε κάθε σελίδα μπορούν αρχικά να ανακτήσουν και να εμφανίσουν τα δεδομένα της όπως επίσης και άλλων χρήσιμων πληροφοριών για την παραγγελία όπως είναι το τραπέζι στο οποίο έχει καταχωρηθεί. Το σύστημα έχει σχεδιαστεί να αντλεί την τελευταία αλλαγή της κάθε παραγγελίας από τον server και τον προώθησή της σε όλα τα συμβαλλόμενα συστατικά της εφαρμογής καθώς και στους υπόλοιπους χρήστες που είναι συνδεδεμένοι στην εφαρμογή έτσι ώστε να μην υπάρχουν ασυνέπειες ακόμη και αν υπάρχει κάποιο πρόβλημα με την διαθεσιμότητα των δεδομένων λόγω τεχνικού σφάλματος.

Η κατάσταση του καταλόγου των προϊόντων ενημερώνεται κάθε φορά επίσης από την βάση με την έναρξη της καταχώρησης μιας νέας παραγγελίας, ενώ τα τραπέζια που απεικονίζονται δημιουργούνται δυναμικά κάθε φορά που ο παραγγελιολήπτης επιστρέφει στην αρχική σελίδα του. Με αυτόν τον τρόπο ακόμη και αν υπάρχουν αλλαγές στο κατάλογο ή στην απεικόνιση των τραπεζιών δεν μπορεί να

επηρεαστεί η κατάσταση των παραγγελιών αλλά μόνο η διαθεσιμότητα των δεδομένων που στην περίπτωση σφάλματος εμποδίζεται και η χρήση της εφαρμογής.



Σχήμα 3.2: Διάγραμμα ροής συστήματος παραγγελιοληψίας

3.2 Αρχιτεκτονική της εφαρμογής

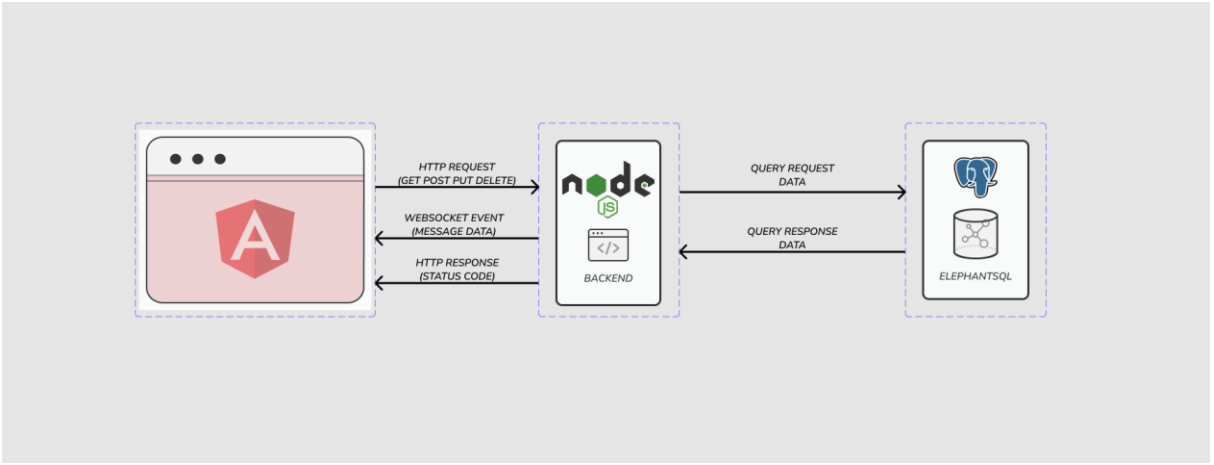
Η αρχιτεκτονική της εφαρμογής περιλαμβάνει όλα τα συστατικά στοιχεία που την αποτελούν μαζί με την διασύνδεσή τους για την σωστή λειτουργία της. Η εφαρμογή ακολουθεί το μοντέλο του πελάτη-εξυπερετητή (client-server) ενώ η web εφαρμογή είναι υλοποιημένη με το μοντέλο του Single Page

Application, μιας εφαρμογής που λειτουργεί χωρίς την ανανέωση της σελίδας από τον χρήστη. Ο server είναι υπεύθυνος για την φιλοξενία των στατικών αρχείων της εφαρμογής, τα HTML, CSS και Javascript αρχεία και παρέχει επίσης ένα RESTful API από το οποίο διατίθενται οι πόροι της εφαρμογής, δηλαδή τα δεδομένα που είναι αποθηκευμένα στην βάση. Επίσης ο server διαθέτει και τα στατικά αρχεία που αποθηκεύει ο χρήστης, όπως είναι οι εικόνες για το λογότυπο και τα τραπέζια του εστιατορίου.

Ο server επικοινωνεί με την βάση δεδομένων μέσω διαδικτύου εφόσον η βάση παρέχεται σαν υπηρεσία από το ElephantSQL. Για την σύνδεση παρέχεται από την υπηρεσία ένα URL το οποίο περιλαμβάνει έναν κωδικό για την πιστοποίηση του χρήση και το όνομα του στιγμιότυπου της βάσης που συνδέεται η εφαρμογή. Το URL της βάσης είναι στατικό και παρέχεται στην εφαρμογή σαν παράμετρος κατά την εκκίνηση του server. Το Sequelize πραγματοποιεί την σύνδεση με την βάση μέσω του URL και αποθηκεύει ένα σύνολο από συνδέσεις (connection pool) για την αποστολή των αιτημάτων στην βάση.

Για την αμφίδρομη επικοινωνία του server με την εφαρμογή, η οποία έχει επιτευχθεί με την μέθοδο των WebSockets, κάθε φορά που ο χρήστης επισκέπτεται την σελίδα, ανοίγει μια TCP σύνδεση του browser και του server μέσω της οποίας ανταλλάσουν Javascript αντικείμενα σαν μηνύματα. Με το κλείσιμο της σελίδας η σύνδεση αυτή τερματίζεται. Ο server όπως ακριβώς και με το API, όταν ξεκινάει, ανοίγει ένα WebSocket το οποίο περιμένει να δεχτεί μια σύνδεση του χρήστη και η αποστολή των μηνυμάτων μπορεί ασύγχρονα με την εκτέλεση ενός συμβάντος να πραγματοποιηθεί είτε στον server είτε στον client.

Η web εφαρμογή εμφανίζεται στον browser όταν ο χρήστης επισκέπτεται το URL του server και κάθε αίτημα στο server για την ανάκτηση των δεδομένων γίνεται στο παρασκήνιο χωρίς να επηρεάζεται η ροή της εφαρμογής ή να χρειάζεται εναλλαγή μεταξύ άλλων σελίδων. Για την εμφάνιση διαφορετικών σελίδων στην εφαρμογή, η Angular παρέχει το δικό της μηχανισμό για να αντιστοιχεί την κάθε σελίδα με ένα διαφορετικό URL, ο οποίος δίνει το πλεονέκτημα ότι μπορεί να προσομοιωθεί το redirection στις σελίδες της εφαρμογής και να διευκολυνθεί έτσι η μετάβαση μεταξύ τους σαν να γινόταν ξεχωριστά αιτήματα στο server.



Σχήμα 3.3: Διάγραμμα αρχιτεκτονικής συστήματος

3.3 Σχεδίαση της Βάσης Δεδομένων

Η σχεσιακή βάση δεδομένων του συστήματος αποτελείται από επτά πίνακες που θα περιγραφούν παρακάτω.

Στον πίνακα users αποθηκεύονται όλοι οι χρήστες του συστήματος. Κάθε ένας από αυτούς έχει ένα μοναδικό αναγνωριστικό κωδικό, το urseId. Το userId είναι ένας αυτοαυξανόμενος ακέραιος αριθμός που ξεχωρίζει μοναδικά την καταχώρηση του χρήστη στην βάση και αποτελεί πρωτεύον κλειδί του πίνακα. Επιπλέον, κάθε χρήστης έχει ένα username και password τα οποία είναι συμβολοσειρές των 255 χαρακτήρων. Τα δύο αυτά πεδία χρησιμοποιούνται και για την σύνδεση του στο σύστημα. Πέραν των πεδίων αυτών, ο χρήστης μπορεί να προσθέσει και το πραγματικό του όνομα και επώνυμο στα πεδία firstName και lastName αντίστοιχα, τα οποία είναι κι αυτά συμβολοσειρές των 255 χαρακτήρων. Χρήσιμη πληροφορία για την επιχείρηση αποτελεί το πεδίο για την ημερομηνία πρόσληψης του υπαλλήλου, το hireDate, το οποίο είναι μία χρονοσφραγίδα, δηλαδή μία κωδικοποιημένη απεικόνιση της ημερομηνίας και της ώρας. Τέλος, ο ρόλος του υπαλλήλου στην επιχείρηση αποθηκεύεται στην στήλη role η οποία είναι συμβολοσειρά των 255 χαρακτήρων και παρέχεται από το object roles στο backend.

Στον πίνακα orders αποθηκεύονται όλες οι παραγγελίες που έχουν καταχωρηθεί στο σύστημα. Κάθε παραγγελία έχει ένα μοναδικό αναγνωριστικό κωδικό, το orderId. Το orderId είναι ένας αυτοαυξανόμενος ακέραιος αριθμός που ξεχωρίζει μοναδικά την εκάστοτε παραγγελία στην βάση και αποτελεί πρωτεύον κλειδί του πίνακα. Επιπλέον, με την στήλη serverId δηλώνεται ο σερβιτόρος της παραγγελίας. Το πεδίο serverId συνδέει τον πίνακα orders και τον users, με αντιστοιχία του userId και ορίζεται ως ξένο κλειδί. Ξένο κλειδί αποτελεί και η στήλη tableNum που ορίζει τον αριθμό τραπεζιού της παραγγελίας με έναν ακέραιο αριθμό και συνδέει τον πίνακα orders με τον πίνακα tables. Σημαντικές πληροφορίες για την παραγγελία αποτελεί και η στήλη orderDate, το οποίο είναι μία χρονοσφραγίδα, δηλαδή μία κωδικοποιημένη απεικόνιση της ημερομηνίας και της ώρας. Για την κατάσταση της παραγγελίας χρησιμοποιείται η στήλη state, μία συμβολοσειρά των 255 χαρακτήρων και παρέχεται από το object states στο backed.

Στον πίνακα tables αποθηκεύονται τα τραπέζια που έχει η εφαρμογή τα οποία έχει δημιουργήσει ο διαχειριστής της εφαρμογής. Κάθε τραπέζι έχει ένα μοναδικό αναγνωριστικό το οποίο είναι πρωτεύον κλειδί στην βάση , το tableNum . Ο αριθμός του τραπεζιού είναι ένας ακέραιος αριθμός που δίνεται σαν όρισμα από τον διαχειριστή της εφαρμογής. Με το πεδίο seats περιγράφεται ο αριθμός καθισμάτων κάθε τραπεζιού και είναι ακέραιος αριθμός. Τέλος, με τις στήλες locationX και locationY αναπαριστάται η ακριβής τοποθεσία του εκάστοτε τραπεζιού. Αποτελούν δύο ακέραιες τιμές οι οποίες αντιπροσωπεύουν τον άξονα X και άξονα Y.

Στον πίνακα reservations αποθηκεύονται όλες οι κρατήσεις της επιχείρησης. Πιο συγκεκριμένα, κάθε κράτηση αποτελείται από ένα μοναδικό αναγνωριστικό κωδικό, το reservationId. Το reservationId είναι ένας αυτοαυξανόμενος ακέραιος αριθμός που ξεχωρίζει μοναδικά την εκάστοτε κράτηση και αποτελεί το πρωτεύον κλειδί στην βάση. Επιπλέον, κάθε κράτηση έχει ένα όνομα το οποίο αποθηκεύεται στο πεδίο reservationName, το οποίο είναι μία συμβολοσειρά των 255 χαρακτήρων. Η ημερομηνία της κράτησης αποθηκεύεται στο πεδίο reservationDate, το οποίο είναι μία χρονοσφραγίδα που περιέχει την ημερομηνία και την ώρα της κράτησης. Τέλος, η πληροφορία που περιέχει τον αριθμό του τραπεζιού που ανήκει η κράτηση αποθηκεύεται στην στήλη tableNum, η οποία είναι ένας

ακέραιος αριθμός και αποτελεί ξένο κλειδί καθώς συνδέει τον πίνακα των κρατήσεων με τον πίνακα των τραπεζιών.

Στον πίνακα categories αποθηκεύονται οι κατηγορίες των προϊόντων του καταλόγου. Κάθε κατηγορία είναι μοναδική και ξεχωρίζεται από την στήλη categoryId, το οποίο είναι ένας αυτοαυξανόμενος ακέραιος αριθμός και αποτελεί το πρωτεύον κλειδί στην βάση. Επιπλέον, για την κάθε κατηγορία ορίζεται ένα όνομα που αποθηκεύεται στην στήλη name και αποτελεί μία συμβολοσειρά των 255 χαρακτήρων. Ο διαχειριστής έχει την επιλογή του ορισμού της διαθεσιμότητας της κατηγορίας, το οποίο αποθηκεύεται στην στήλη isAvailable, το οποίο είναι μία λογική μεταβλητή. Τέλος, τα χαρακτηριστικά των προϊόντων για τις διάφορες προσθήκες και τροποποιήσεις που μπορούν να δεχτούν, ομαδοποιούνται ανά κατηγορία και αποθηκεύονται στην στήλη qualifierType, το οποίο αποτελεί μία συμβολοσειρά των 255 χαρακτήρων.

Στον πίνακα items αποθηκεύονται τα προϊόντα του καταλόγου της εκάστοτε επιχείρησης. Κάθε προϊόν είναι μοναδικό και αποθηκεύεται στην βάση ο μοναδικός ακέραιος αυτοαυξανόμενος αριθμός. Επιπλέον, για τον κάθε αριθμό ορίζεται ένα όνομα που το περιγράφει, το title το οποίο αποτελεί μία συμβολοσειρά των 255 χαρακτήρων. Η τιμή από το κάθε προϊόν αποθηκεύεται στην στήλη price ως ακέραιος αριθμός. Όπως και οι κατηγορίες προϊόντων ο διαχειριστής μπορεί να ορίσει την διαθεσιμότητα του προϊόντος στην στήλη isAvailable, η οποία είναι μία λογική μεταβλητή. Κάθε προϊόν αποτελείται από τα συστατικά του, τα οποία περιγράφονται στην στήλη ingredients, που είναι μία συμβολοσειρά των 255 χαρακτήρων. Τέλος, κάθε προϊόν ανήκει σε μία κατηγορία η οποία αποθηκεύεται στην στήλη categoryId. Η στήλη categoryId είναι ακέραιος αριθμός και αποτελεί ξένο κλειδί καθώς συνδέει τον πίνακα items με τον πίνακα categories.

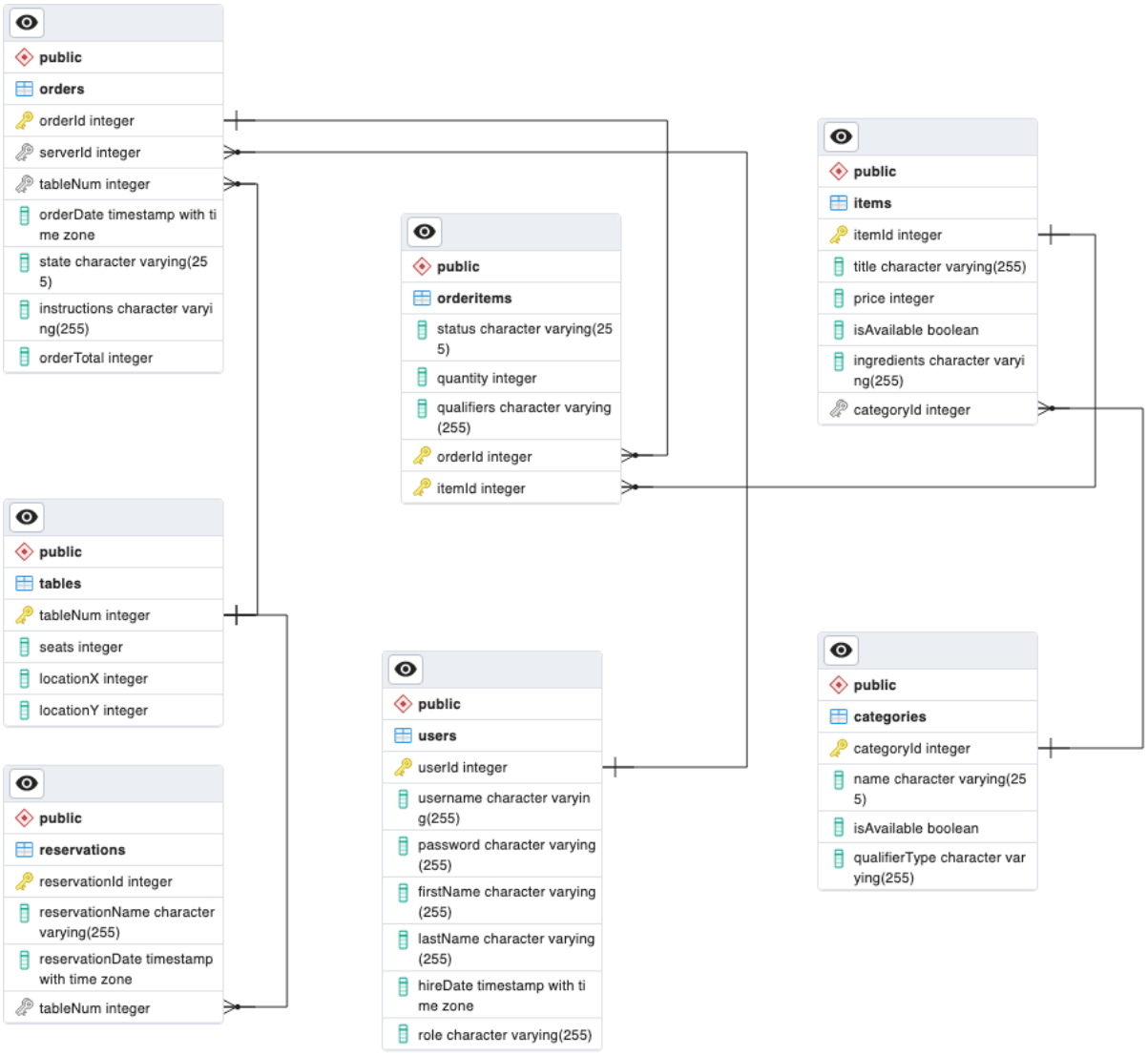
Τέλος, στον πίνακα orderItems αποθηκεύονται τα προϊόντα που υπάρχουν σε μία παραγγελία. Πιο συγκεκριμένα, ο πίνακας αυτών συνδέει τον πίνακα orders με τον πίνακα items. Ο συνδυασμός των δύο ξένων κλειδιών, του orderId και itemId αποτελεί το πρωτεύον κλειδί στον πίνακα. Επιπλέον, κάθε παραγγελία έχει την πληροφορία της κατάστασης. Η κατάσταση της παραγγελίας αποθηκεύεται στην στήλη status σαν συμβολοσειρά των 255 χαρακτήρων. Τέλος, η ποσότητα για το κάθε προϊόν αποθηκεύεται στην στήλη quantity σαν ακέραιος αριθμός και τα χαρακτηριστικά του κάθε προϊόντος στην στήλη qualifiers η οποία αποτελεί μία συμβολοσειρά 255 χαρακτήρων.

Παρακάτω περιγράφονται οι συσχετίσεις τον πινάκων, σύμφωνα με τα κλειδιά (πρωτεύοντα και ξένα):

- Το table users με το orders έχουν συσχέτιση ένα προς πολλά (One-to-Many) μέσω του πρωτεύοντος κλειδιού userId του users και του ξένου κλειδιού serverId του orders καθώς ένα order ανήκει σε έναν μόνο user ενώ ένας user μπορεί να αναλάβει πολλά orders.
- Το table orders με το tables έχουν συσχέτιση ένα προς πολλά μέσω του πρωτεύοντος κλειδιού tableNum του tables και του ξένου κλειδιού tableNum του orders. Αυτό ισχύει καθώς ένα order καταχωρείται σε ένα μόνο table ενώ σε ένα table θέλουμε να αναθέσουμε παραπάνω από ένα orders.
- Το table reservations έχει συσχέτιση επίσης ένα προς πολλά με το tables μέσω του πρωτεύοντος κλειδιού tableNum του tables και του ξένου κλειδιού tableNum του reservations και επειδή μια ξεχωριστή κράτηση γίνεται σε ένα ακριβώς τραπέζι, ενώ ένα τραπέζι μπορεί να έχει πολλές κρατήσεις.
- Το order σχετίζεται με το item με σχέση πολλά προς πολλά (Many-to-Many) μέσω του table orderitems και η σύνδεση τους γίνεται με την βοήθεια των ξένων κλειδιών orderId και itemId του orderitems. Αυτό είναι επιθυμητό επειδή ένα order μπορεί να περιέχει αρκετά items ενώ ένα item μπορεί να συμπεριληφθεί σε πολλά orders.

- Τέλος το table items με το categories έχουν συσχέτιση ένα προς πολλά μέσω του πρωτεύοντος κλειδιού categoryId του categories και του ξένου κλειδιού categoryId του items καθώς ένα item ανήκει σε ένα μόνο category ενώ ένα category περιλαμβάνει πολλά items.

Στο παρακάτω διάγραμμα μοντέλο-συσχετίσεων (Entity Relationship Diagram - ERD) μπορούμε να διακρίνουμε τον κάθε πίνακα με τις αντίστοιχες στήλες του, καθώς και την μεταξύ τους σχέση. Η διακλάδωση με τις τρείς γραμμές συμβολίζει την σχέση πολλά από την μεριά του table που έρχεται η γραμμή ενώ ο σταυρός τη συσχέτιση του ενός.



Σχήμα 3.4: Διάγραμμα Οντοτήτων-Συσχετίσεων της βάσης δεδομένων του συστήματος

3.4 Αυθεντικοποίηση

Για την εξασφάλιση ότι η εφαρμογή μπορεί να χρησιμοποιηθεί αποκλειστικά από τους χρήστες που έχουν πιστοποιηθεί από το σύστημα και έχουν τον κατάλληλο ρόλο έχει σχεδιαστεί και υλοποιηθεί ένα σύστημα αυθεντικοποίησης με την μέθοδο του Bearer token. Με αυτόν τον όρο εννοείται ότι

χρησιμοποιείται ένα μοναδικό αλφαριθμητικό στοιχείο το οποίο είναι αποκλειστικά υπεύθυνο για την πρόσβαση του χρήστη στο σύστημα. Για να επιτευχθεί η πρόσβαση ο χρήστης πρέπει αρχικά να έχει καταχωρηθεί στο σύστημα από τον διαχειριστή και να έχει στην κατοχή του ένα χαρακτηριστικό όνομα (username) και έναν κωδικό (password).

Η δημιουργία ενός session στο σύστημα δεν είναι εφικτή καθώς η συγκεκριμένη αρχιτεκτονική δεν αποθηκεύει πληροφορίες για την σύνδεση του χρήστη στην εφαρμογή. Αντί για την δημιουργία session αποστέλλεται στον browser το token το οποίο καθίσταται ανενεργό έπειτα από μια προκαθορισμένη ώρα από την δημιουργία του και αυτό αντιστοιχεί στην λήξη της συνεδρίας οπότε ο χρήστης θα πρέπει να πραγματοποιήσει ξανά μια σύνδεση με τα διαπιστευτήριά του. Χωρίς την χρήση του token η πρόσβαση στους πόρους του συστήματος δεν είναι δυνατή και η μόνη ενέργεια στην οποία μπορεί να προβεί ο χρήστης είναι να σύνδεση εκ νέου.

Το token είναι ένα JSON Web Token το οποίο περιέχει πληροφορίες του χρήστη απαραίτητες για τη λειτουργία της εφαρμογής, όπως ο ρόλος του χρήστη και το αναγνωριστικό του ενώ αναφέρεται και ο πάροχος του token, ένα αναγνωριστικό της ίδιας της εφαρμογής. Το token κωδικοποιείται με ένα μοναδικό αλφαριθμητικό το οποίο με κρυπτογραφικούς όρους είναι ένα ιδιωτικό κλειδί και αποθηκεύεται στο server. Είναι ιδιαίτερα σημαντικό το συγκεκριμένο κλειδί να παραμείνει κρυφό για την εξασφάλιση της πρόσβασης στο σύστημα μόνο από χρήστες που έχουν ήδη πραγματοποιήσει μια σύνδεση με τα διαπιστευτήριά τους καθώς με την γνώση του κλειδιού μπορούν να πραγματοποιηθούν μη εξουσιοδοτημένες προσπάθειες για την ανάκτηση των πόρων του συστήματος.

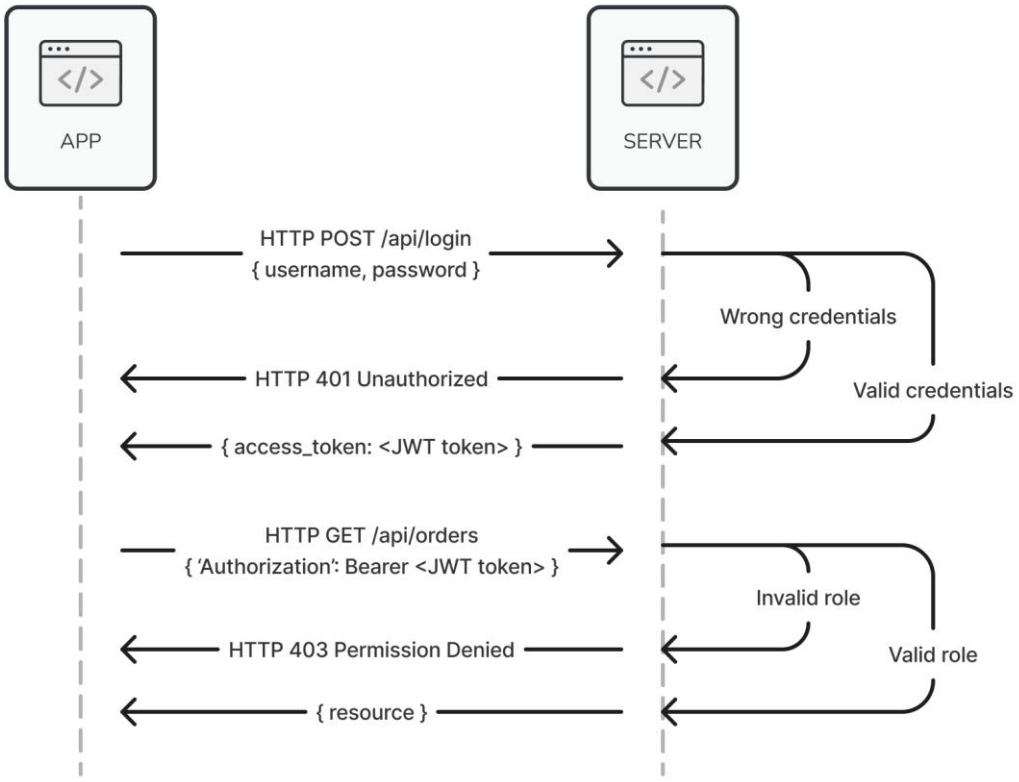
Για την πιστοποίηση του χρήστη από το σύστημα και την πρόσβασή του στους πόρους και στις αντίστοιχες σελίδες πραγματοποιείται μια χειραψία (handshake) τριών δρόμων μεταξύ του browser του χρήστη και του εξυπηρετητή. Στο πρώτο στάδιο ο χρήστης στέλνει με ένα HTTP POST αίτημα τα διαπιστευτήριά του τα οποία ελέγχονται από τον server και αν ο χρήστης είναι πιστοποιημένος, είναι δηλαδή καταχωρημένος στο σύστημα, επιστρέφεται το token σαν απάντηση στο αίτημα αυτό. Έπειτα η εφαρμογή προσθέτει το token στην κεφαλίδα Authorization κάθε επόμενου αιτήματος, ο server επικυρώνει ότι το token έχει χορηγηθεί από τον ίδιο και ότι ο χρήστης είναι πιστοποιημένος και επιστρέφει σαν απάντηση τον αιτούμενο πόρο.

Η πιστοποίηση του χρήστη (Authentication) γίνεται με τον έλεγχο της υπογραφής του token και την εύρεση του χρήστη στη βάση δεδομένων. Εάν ο χρήστης είναι καταχωρημένος, η διαδικασία της απάντησης στο αίτημα συνεχίζεται κανονικά. Στην περίπτωση που τα στοιχεία του χρήστη δεν είναι αποθηκευμένα στη βάση επιστρέφεται μια απάντηση με ένα μήνυμα και κωδικό σφάλματος 401 Unauthorized στην εφαρμογή. Στην περίπτωση που το token δεν έχει υπογραφεί με το κρυφό κλειδί που διαθέτει μόνο ο server, επιστρέφεται ξανά το ίδιο σφάλμα και πρόκειται για μια μη εξουσιοδοτημένη προσπάθεια σύνδεσης στο σύστημα.

Η πιστοποίηση των δικαιωμάτων του χρήστη (Authorization) γίνεται με την εξαγωγή των δεδομένων από το token και την εύρεση του username του χρήστη στη βάση. Αν ο συγκεκριμένος ρόλος έχει το δικαίωμα να ανακτήσει τον πόρο που περιγράφει το αίτημα επιστρέφεται η απάντηση. Σε κάθε άλλη περίπτωση επιστρέφεται ένα μήνυμα σφάλματος με τον κωδικό 403 Permission Denied για να δηλώσει ότι ο χρήστης είναι καταχωρημένος αλλά δεν έχει την δυνατότητα ανάκτησης του πόρου που ζητήθηκε.

Έπειτα από την ολοκλήρωση της χρήσης της εφαρμογής η αποσύνδεση του χρήστη γίνεται με την διαγραφή του token από τον browser και την δήλωση του token ως ανενεργό, έτσι ώστε οποιαδήποτε χρήση του μετά την αποσύνδεση να μην μπορεί να επιστρέψει μια απάντηση. Με αυτόν τον τρόπο

εξομοιώνεται και η λήξη του session με τον σύστημα καθώς είναι η μόνη πληροφορία που παρέχεται από τον server για την σύνδεση του συγκεκριμένου χρήστη στο σύστημα.



Σχήμα 3.5: Διάγραμμα ροής πληροφορίας αυθεντικοποίησης

3.5 Υλοποίηση του Backend

Η υλοποίηση του backend βασίζεται σε μεγάλο βαθμό στο Express.js ενώ περιλαμβάνει και την υλοποίηση των WebSockets. Για την ενεργοποίηση του API το express παρέχει όλα τα απαραίτητα εργαλεία με το Node.js να αναλαμβάνει μόνο την εκτέλεση του κώδικα σαν ένα περιβάλλον εκτέλεσης. Οι βιβλιοθήκες του Node.js που χρησιμοποιήθηκαν αφορούν μόνο την επικοινωνία με το σύστημα αρχείων του server για την αποθήκευση και τον διαμοιρασμό των αρχείων και για την δημιουργία ενός HTTP server ο οποίος αξιοποιείται από το socket.io για την εγκαθίδρυση μιας σύνδεσης μέσω των WebSockets.

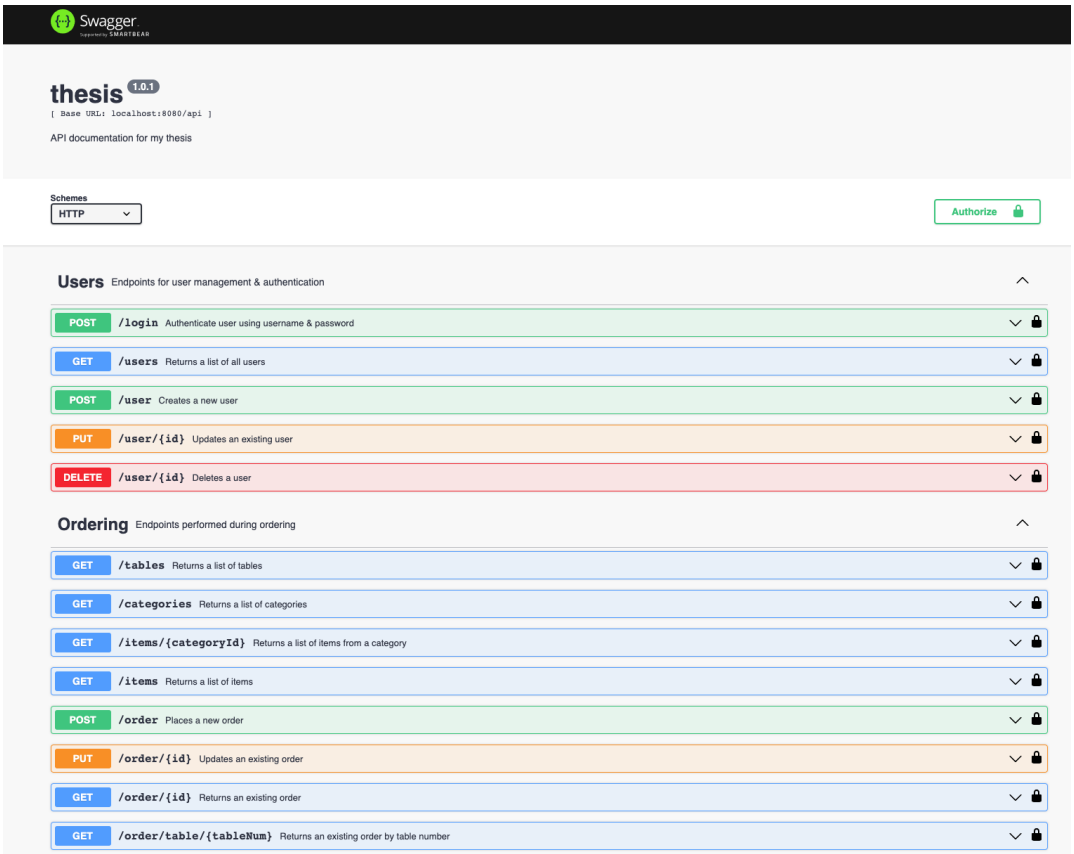
Ο κώδικας του backend χωρίζεται σε μερικούς φακέλους που περιέχουν τα βασικά συστατικά του στοιχεία, ένα αρχείο που περιγράφει το ίδιο το express αντικείμενο που χειρίζεται την εφαρμογή και μερικά αρχεία παραμετροποίησης και χειρισμού των βιβλιοθηκών. Οι βιβλιοθήκες αποθηκεύονται στον φάκελο node_modules μαζί με τις εξωτερικές βιβλιοθήκες που ανακτώνται από το αποθετήριο του node. Το εργαλείο που χειρίζεται τις βιβλιοθήκες λέγεται npm (node package manager). Το αρχείο που περιγράφει τον server και παραπέμπει στα υπόλοιπα αρχεία που εκτελούνται τα middlewares της εφαρμογής λέγεται server.js

Τα endpoints της εφαρμογής για κάθε πόρο αποθηκεύονται σε ξεχωριστά αρχεία στον φάκελο routes. Κάθε ένα από αυτά τα αρχεία εξάγει ένα αντικείμενο στο server.js με τα middlewares του express που

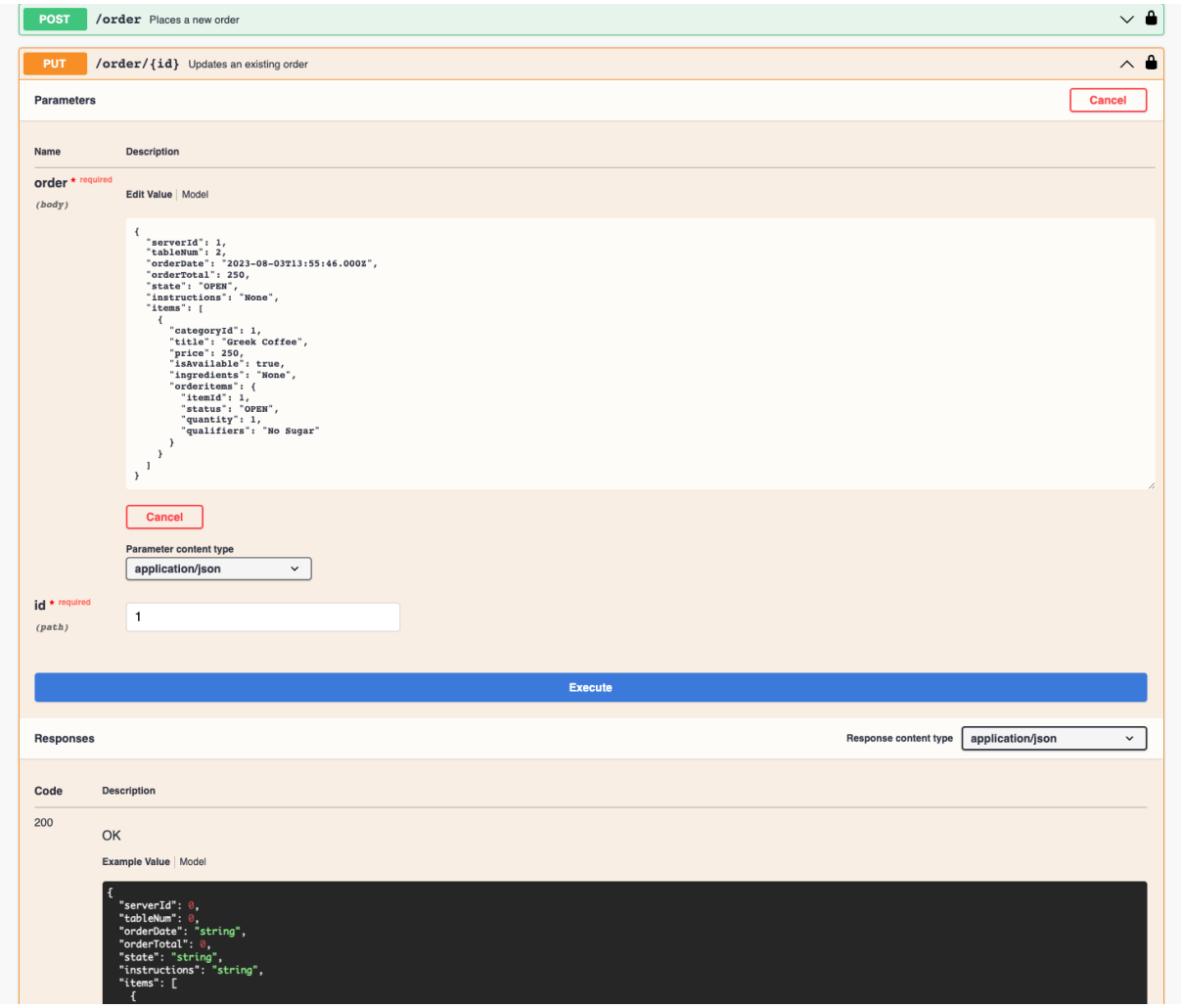
καλούν με την σειρά τους το καθένα από ένα ακόμη middleware που αναλαμβάνει την εκτέλεση του κώδικα και καλείται controllers. Οι controllers αποθηκεύονται σε ένα ξεχωριστό αρχείο για κάθε πόρο στον φάκελο controllers. Ένας controller είναι το τελικό σημείο εκτέλεσης της ροής του κώδικα για κάθε αίτημα πριν την αποστολή της απάντησης το οποίο δηλώνεται με την επιστροφή του αντικειμένου res (response).

Μέσα στο αρχείο ενός controller έχουν υλοποιηθεί οι μέθοδοι που εκτελούν τις CRUD λειτουργίες στην βάση με την βοήθεια του Sequelize. Για την πρόσβαση στην βάση υπάρχει ένα μοντέλο για κάθε πίνακα το οποίο περιγράφει σαν αντικείμενο τις στήλες του. Τα αντικείμενα αυτά αποθηκεύονται μαζί με την περιγραφή των συσχετίσεων των πινάκων στο αρχείο database.js μέσα στο φάκελο util. Επίσης στο φάκελο του database.js δηλώνονται και οι συσχετίσεις των πινάκων με την κλήση των μεθόδων hasOne, hasMany, belongsToOne και belongsToMany του Sequelize.

Η εκτέλεση του controller περιλαμβάνει αρχικά την εξαγωγή των παραμέτρων από το HTTP αίτημα, την ανάκτηση των δεδομένων από την βάση, τον έλεγχο των δεδομένων και σε περίπτωση που δεν υπάρχουν την επιστροφή του κατάλληλου μηνύματος με τον κωδικό 404 ενώ στην περίπτωση επιτυχίας εύρεσης των κατάλληλων δεδομένων επιστρέφονται με τον κωδικό 200 και τερματίζεται η επεξεργασία του αιτήματος. Εάν προκύψει κάποιο σφάλμα κατά την επεξεργασία του αιτήματος καλείται η μέθοδος next που παραπέμπει στην εκτέλεση του επόμενου κατά σειρά middleware και δίνεται σαν όρισμα το αντικείμενο του σφάλματος που προέκυψε.



Σχήμα 3.6: Υλοποίηση όλων των endpoint του API στο Swagger



Σχήμα 3.7: Δυνατότητα επεξεργασίας των endpoints για λειτουργίες CRUD

Για την διαχείριση των σφαλμάτων του server δηλώνεται τελευταίο ένα middleware το οποίο περιλαμβάνει ένα ακόμη όρισμα το οποίο είναι το αντικείμενο που περιγράφει ένα σφάλμα. Η εμφώλευση του συγκεκριμένου middleware μπορεί να γίνει από οποιοδήποτε σημείο του κώδικα με την δήλωση του error σαν όρισμα όπως αναφέρεται στην προηγούμενη παράγραφο. Η εκτέλεση της μεθόδου για κάθε error στέλνει σαν απάντηση στον browser το μήνυμα του σφάλματος με τον κωδικό 500 και αποτρέπει έτσι την κατάρρευση του server από αναπάντεχα σφάλματα κατά την εκτέλεση.

Πριν την εκτέλεση κάθε controller εμφωλεύονται σαν ορίσματα δύο ακόμη middlewares τα οποία είναι υπεύθυνα για την αυθεντικοποίηση και την πιστοποίηση των δικαιωμάτων του χρήστη για τον κάθε ρόλο. Τα middlewares αποθηκεύονται στο φάκελο middlewares στο αρχείο authMiddleware.js

και γίνονται import στο κάθε route αρχείο για την προστασία όλων των endpoints του API. Μετά την δήλωση του μονοπατιού ενός endpoint το πρώτο middleware που καλείται είναι το authenticate το οποίο ελέγχει την ταυτότητα του JWT token και βρίσκει τον χρήστη στην βάση για να επιβεβαιώσει τα στοιχεία του. Εάν προκύψει κάποιο σφάλμα διακόπτεται η ροή του και επιστρέφεται στον browser ο κωδικός 401 Unauthorized με το κατάλληλο μήνυμα. Με την επιτυχή πιστοποίηση του χρήστη καλείται η μέθοδος next και εκτελείται το επόμενο σε σειρά middleware.

Για την πιστοποίηση του ρόλου του χρήστη υπάρχει το middleware authorize το οποίο δέχεται σαν όρισμα μια λίστα με τους ρόλους του συστήματος που τους επιτρέπεται να έχουν πρόσβαση στον πόρο που διαμοιράζει το συγκεκριμένο endpoint. Εφόσον ο χρήστης διαθέτει έναν από τους απαιτούμενους ρόλους εκτελείται η next και ξεκινάει η εκτέλεση του controller. Σε κάθε άλλη περίπτωση επιστρέφεται στον browser ένα μήνυμα που δηλώνει ότι ο χρήστης δεν έχει δικαίωμα χρήσης του πόρου με τον κωδικό 403 Permission Denied.

Για την υλοποίηση των WebSockets υπάρχει ένα αρχείο socket.js στο φάκελο util που περιλαμβάνει την δήλωση ενός http server ο οποίος δίνεται σαν όρισμα στην αρχικοποίηση του socket και έπειτα δηλώνεται ότι όταν προκύψει ένα event στο server με την ονομασία connection θα προστεθεί το id της σύνδεσης, που αντιστοιχεί με τον browser ενός χρήστη που ανοίγει επισκέπτεται την σελίδα της εφαρμογής. Στο ίδιο αρχείο παρέχεται και μια μέθοδος η οποία δέχεται ως ορίσματα την ονομασία του event και ένα μήνυμα τα οποία με την βοήθεια της σύνδεσης του socket στέλνει στον browser.

Η αποστολή ενός μηνύματος μέσω WebSockets εξυπηρετεί τον σκοπό της ενημέρωσης των υπόλοιπων χρηστών και συγκεκριμένα των σερβιτόρων για την αλλαγή της κατάστασης μιας παραγγελίας από τον μάγειρα. Συγκεκριμένα όταν γίνεται ένα αίτημα στο endpoint /api/order/:orderId με την μέθοδο put, πραγματοποιείται δηλαδή μια αλλαγή σε μια παραγγελία με αναγνωριστικό orderId, στέλνεται ένα μήνυμα στο event ‘order’ το οποίο εισακούει η web εφαρμογή και του παραδίδει την αλλαγή στην κατάσταση της παραγγελίας, πράγμα αδύνατο να υλοποιηθεί μόνο με την χρήση HTTP αιτημάτων.

3.6 Υλοποίηση του Frontend

Η υλοποίηση του frontend έχει οργανωθεί σε φακέλους με βάση τα συστατικά της Angular και τις κοινές πρακτικές που εφαρμόζονται με την χρήση του συγκεκριμένου framework. Στο project υπάρχει ένας φάκελος src μέσα στον οποίον υπάρχουν όλα τα αρχεία του κώδικα της εφαρμογής ενώ στο εξωτερικό φάκελο web περιλαμβάνονται τα αρχεία παραμετροποίησης της εφαρμογής, τα node_modules με τις βιβλιοθήκες και τα αρχεία που χρειάζονται για την εκτέλεση του development server και τα αρχεία που περιγράφουν και παραμετροποιούν την Typescript στην οποία βασίζεται όλο το project.

Ο φάκελος src περιλαμβάνει δύο ξεχωριστούς φακέλους και τα αρχεία index.html, styles.css, main.ts και favicon.ico. Το main.ts ενσωματώνει το AppModule το οποίο είναι η συλλογή των components της εφαρμογής τα οποία θα γίνουν render στο browser. Ο ένας από τους δύο φακέλους στο src, ονομάζεται assets και περιλαμβάνει τα εικονίδια της εφαρμογής, δηλαδή τα στατικά αρχεία που φορτώνονται στον browser κατά την λήψη της εφαρμογής. Ο άλλος φάκελος ονομάζεται app και περιλαμβάνει όλα τα components της εφαρμογής.

Η ομαδοποίηση των components μέσα στο app γίνεται ως εξής:

- components: περιλαμβάνει όλους τους φακέλους με τα βασικά components της εφαρμογής όπως τα βασικά κουμπιά και οι φόρμες και μερικά επαναχρησιμοποιούμενα components που εμφανίζονται αρκετά όπως το μενού των προϊόντων, η αναπαράσταση της παραγγελίας και τα εικονίδια των τραπεζιών. Τα ονόματα των φακέλων είναι ίδια με τις κλάσεις των components που περιέχονται σε αυτά ενώ κάθε φάκελος περιέχει τα τρία αρχεία που χαρακτηρίζουν ένα Angular component, ένα .component.ts το οποίο περιέχει την λογική και τις μεθόδους του component, ένα .component.html το οποίο αντιπροσωπεύει το HTML template της εφαρμογής και ένα .component.css το οποίο περιέχει τις κλάσεις του CSS για την διακόσμηση του component. Κάθε component χαρακτηρίζεται από το annotation της Angular @Component στο οποίο δηλώνονται τα αρχεία του template και του styling καθώς επίσης και ο selector του component ο οποίος προστίθεται μέσα σε tags για να καλέσει το component μέσα σε ένα template.
- directives: εδώ περιλαμβάνεται ένα μόνο αρχείο το οποίο περιγράφει το directive που δηλώνει το σημείο που εισάγονται τα τραπέζια δυναμικά στην αναπαράσταση του εστιατορίου. Το directive αυτό ονομάζεται insertion point και εισάγεται σε ένα div το οποίο περιλαμβάνει τα τραπέζια στην σελίδα του διαχειριστή. Αυτό επιτρέπει στο μηχανισμό της Angular που δημιουργεί δυναμικά components να γνωρίζει το σημείο που θα εμφανίσει κάθε καινούργιο component. Το αρχείο έχει την κατάληξη .directive.ts σαν πρακτική της Angular.
- models: σε αυτόν τον φάκελο αποθηκεύονται τα αρχεία που περιέχουν τα models της εφαρμογής σαν objects όπως επιστρέφονται από το backend και τα enumerations που περιέχουν τις ονομασίες των states του order και του mode στο οποίο ανοίγει το order στα διαφορετικά pages. Τα models είναι μια πρακτική η οποία επιτρέπει την επαναχρησιμοποίηση των αντικειμένων της εφαρμογής και την απαλοιφή λαθών που σχετίζονται με τους τύπους των δεδομένων που επιστρέφει ο server και χρησιμοποιεί η εφαρμογή. Η ονομασία των αρχείων είναι το όνομα του model και το .model.ts ως επίθεμα.
- pages: στο φάκελο pages υπάρχουν τα components που αντιπροσωπεύουν τις ξεχωριστές σελίδες της εφαρμογής. Κάθε σελίδα έχει ονομασία με βάση τον ρόλο του χρήστη από τον οποίο αξιοποιείται και το περιεχόμενο της εφαρμογής. Η πρώτη λέξη είναι ο ρόλος, η δεύτερη είναι η λειτουργία και η τρίτη η λέξη page. Έτσι έχουμε δύο σελίδες για τον χρήστη για την εμφάνιση της αναπαράστασης του εστιατορίου και της σελίδας της παραγγελιοληψίας, μια σελίδα για τον παραγγελιοδέκτη και 7 σελίδες για τον διαχειριστή, μια για την κεντρική σελίδα και μια για κάθε λειτουργία που του παρέχεται. Υπάρχει επίσης μια ακόμη σελίδα που ονομάζεται login-page και αντιπροσωπεύει το landing page, δηλαδή την πρώτη σελίδα που εμφανίζεται στο χρήστη όταν δεν έχει συνδεθεί στη εφαρμογή.
- services: εδώ βρίσκονται τα services της Angular για κάθε αντικείμενο που περιγράφεται σαν model και για το authentication, τα οποία περιλαμβάνουν υλοποιήσεις για τις κλήσεις σε κάθε ένα από τα endpoint της εφαρμογής. Κάθε κλάση μέσα σε ένα αρχείο χαρακτηρίζεται από το annotation @Injectable το οποίο υποδηλώνει ότι μπορεί να εισαχθεί δυναμικά από τον constructor κάποιας άλλης κλάσης σαν όρισμα. Τα services κατηγοριοποιούνται με βάση το model και η ονομασία τους αποτελείται από το όνομα του model και το .service.ts ως επίθεμα.

Τα μονοπάτια που παρέχει η εφαρμογή για να εμφανιστούν τα pages είναι χωρισμένα όπως τα pages με βάση τον ρόλο και την αντίστοιχη λειτουργία που εκτελεί το page που εμφανίζεται. Στο αρχείο app.module.ts δηλώνονται τα μονοπάτια μέσα σε μια λίστα ονόματι appRoutes με τα ορίσματα το μονοπάτι ως αλφαριθμητικό, το component το οποίο εμφανίζεται με την επίσκεψη του συγκεκριμένου μονοπατιού και ένα ακόμη όρισμα για την προστασία του page. Το αντικείμενο για την προστασία λέγεται AuthGuard και περιλαμβάνει μια μέθοδο που ελέγχει αν ο χρήστης έχει κάνει login στην εφαρμογή, με τον έλεγχο δηλαδή του token που έχει σταλεί από τον server.

Τα μονοπάτια της εφαρμογής μαζί με τα components που εμφανίζουν είναι τα εξής:

- /login: το αρχικό μονοπάτι της εφαρμογής, το οποίο είναι και το μόνο που εμφανίζεται στην περίπτωση που δεν υπάρχει κανένα μονοπάτι στο URL που επισκέπτεται ο χρήστης και εμφανίζει το LoginPageComponent από το login-page.
- /admin: η αρχική σελίδα του διαχειριστή η οποία εμφανίζει το AdminPageComponent και περιλαμβάνει έξι εικονίδια που παραπέμπουν στις έξι δυνατές λειτουργίες του χρήστη ως διαχειριστή
- /admin/employees: η σελίδα που δίνει την δυνατότητα της διαχείρισης των χρηστών του συστήματος και εμφανίζει το AdminEmployeePageComponent.
- /admin/reports: η σελίδα που εμφανίζει συσσωρευμένες τις παραγγελίες του συστήματος μέσω του AdminReportsPageComponent.
- /admin/map: η σελίδα που εμφανίζει το AdminMapPageComponent και παρέχει την λειτουργία της διαχείρισης της αναπαράστασης του εστιατορίου στον διαχειριστή.
- /admin/editmenu: η σελίδα δίνει την δυνατότητα επεξεργασίας του καταλόγου του εστιατορίου και εμφανίζει το AdminEditmenuPageComponent.
- /admin/reservations: η σελίδα για την διαχείριση των κρατήσεων των τραπεζιών του εστιατορίου η οποία εμφανίζει το AdminReservationsPageComponent.
- /admin/images: η σελίδα για το ανέβασμα στον server των εικόνων της εφαρμογής η οποία εμφανίζει το AdminImagesPageComponent.
- /server/map: η αρχική σελίδα του χρήστη που συνδέεται ως σερβιτόρος και εμφανίζει το ServerMapPageComponent με την αναπαράσταση του εστιατορίου και των τραπεζιών με την μορφή που έχουν οριστεί από τον διαχειριστή.
- /server/order: η σελίδα την οποία ανοίγει ο σερβιτόρος με την επιλογή ενός τραπεζιού και εμφανίζει το ServerOrderPageComponent με τον κατάλογο του εστιατορίου και την προεπισκόπηση της παραγγελίας
- /cooker/orders: η σελίδα του παραγγελιοδέκτη ο οποίος μπορεί να δει και να ετοιμάσει ή να ακυρώσει τις παραγγελίες. Εμφανίζεται μέσω του CookerOrdersPageComponent.
- Στο τέλος της λίστας ορίζεται ένα μονοπάτι wildcard το οποίο χρησιμεύει στην περίπτωση που δεν αντιστοιχηθεί κανένα από τα παραπάνω να παραπέμπει στην σελίδα του login.

Η σελίδα του login εμφανίζεται με το άνοιγμα της σελίδας από τον χρήστη στην περίπτωση που ο χρήστης δεν έχει πραγματοποιήσει ακόμη την είσοδο του στο σύστημα. Εμφανίζεται το λογότυπο του εστιατορίου με ένα πεδίου για το username, ένα πεδίο για το password και ένα κουπί Login για την αποστολή του αιτήματος της σύνδεσης. Το component LoginPageComponent χειρίζεται την εισαγωγή των στοιχείων του χρήστη μέσω μιας φόρμας η οποία αποθηκεύει προσωρινά τις τιμές που εισάγει ο χρήστης και τις προωθεί στο login method του LoginService για να σταλεί τελικά το αίτημα στο server. Στην περίπτωση απάντησης με σφάλμα, το μήνυμα του σφάλματος εμφανίζεται στον χρήση σε ένα κόκκινο πλαίσιο ενώ τα πεδία εισαγωγής έχουν κόκκινο περίγραμμα στην περίπτωση που ο χρήστης δεν τα συμπληρώσει πριν την αποστολή του αιτήματος.

Το token που επιστρέφεται με την επιτυχία της σύνδεσης του χρήστη αποθηκεύεται στο browser αυτούσιο αλλά πραγματοποιείται και η αποκωδικοποίησή του για την αποθήκευση των χρήσιμων πληροφοριών που περιέχει για την μετέπειτα επαναχρησιμοποίηση τους. Με την αποκωδικοποίηση του ελέγχεται ο ρόλος και εμφανίζεται η αντίστοιχη κεντρική σελίδα με την χρήση του method navigate που παρέχει το αντικείμενο Router της Angular. Το token χρησιμοποιείται σε κάθε HTTP αίτημα ενώ με την εκτέλεση του logout method διαγράφεται πλήρως από τον browser για να πραγματοποιηθεί η λήξη της συνεδρίας.

Το βασικό component της εφαρμογής είναι το TicketComponent. Σε αυτό δίνεται σαν όρισμα μια παραγγελία η οποία αποθηκεύεται σε αυτό και εμφανίζονται τα στοιχεία της παραγγελίας στα οποία περιλαμβάνονται ο αναγνωριστικός αριθμός της, η ημερομηνία της δημιουργίας της, ο υπάλληλος που

την καταχώρισε, το τραπέζι στο οποίο ανήκει και τα προϊόντα τα οποία αντιστοιχούν σε αυτή. Επίσης ως όρισμα mode δίνεται μια τιμή που δηλώνει την μορφή της εμφάνισης της παραγγελίας η οποία περιλαμβάνει είτε το τελικό ποσό είτε τα κουμπιά που αλλάζουν την κατάστασή της. Οι καταστάσεις αυτές ονομάζονται αντίστοιχα Total και Buttons και αποθηκεύονται στο ticket-mode.model.ts σαν ένα enumeration. Τα προϊόντα της παραγγελίας εμφανίζονται με την ονομασία, τα χαρακτηριστικά που έχει επιλέξει ο χρήστης, την ποσότητα των ξεχωριστών προϊόντων και της τιμής τους. Δίπλα ακριβώς εμφανίζεται ένα κουμπί για την διαγραφή του προϊόντος από την παραγγελία και για τον έλεγχο της εμφάνισης του χρησιμοποιείται ακόμη ένα όρισμα που ονομάζεται editMode το οποίο όταν είναι true επιτρέπει την διαγραφή προϊόντων και την προσθήκη οδηγιών στην παραγγελία.

Η κεντρική σελίδα του διαχειριστή είναι αρκετά απλή καθώς περιέχει μόνο έξι κουμπιά και το καθένα εμφανίζει τη σελίδα της αντίστοιχης λειτουργίας. Σε αυτή τη σελίδα όπως και σε κάθε κεντρική σε σελίδα (ServerMapPageComponent, CookerOrdersPageComponent) υπάρχει επίσης ένα κουμπί το οποίο εκτελεί την αποσύνδεση του χρήστη από την εφαρμογή και την εμφάνιση της σελίδας του login. Με την πλοήγηση του χρήστη σε κάποια άλλη σελίδα δεν είναι δυνατή εάν δεν πραγματοποιηθεί ξανά αποστολή της φόρμας του login.

Η σελίδα της διαχείρισης των χρηστών περιλαμβάνει μια λίστα στην οποία εμφανίζονται δυναμικά όλοι οι χρήστες της εφαρμογής με τα στοιχεία του ονόματος τους, του επιθέτου τους, του ρόλου, του username και της ημερομηνίας πρόσληψής τους. Για την ανάκτηση των δεδομένων των χρηστών εκτελείται η μέθοδος getUsers του user service και το αποτέλεσμα το οποίο είναι μια λίστα διατρέχεται μέσω του directive *ngFor της Angular και προσθέτει τα στοιχεία στα αντίστοιχα πεδία της λίστας. Η λίστα είναι ένα HTML table το οποίο έχει δομηθεί με τις κεφαλίδες που περιέχουν την περιγραφή της στήλης και την γραμμή που δημιουργείται για κάθε μια εγγραφή της λίστας των χρηστών και στοιχειοθετεί τα δεδομένα κάτω από την σωστή κεφαλίδα.

Για την διαχείριση των χρηστών στην σελίδα υπάρχουν δύο κουμπιά δίπλα σε κάθε εγγραφή, για την επεξεργασία των δεδομένων και την διαγραφή του χρήστη από το σύστημα. Με την επιλογή της επεξεργασίας ανοίγει μια φόρμα μέσα σε ένα modal που περιέχει τα ήδη υπάρχοντα στοιχεία του χρήστη και με την επιλογή του κουμπιού Update στέλνεται στο server το αντικείμενο με τα νέα στοιχεία του χρήστη μέσω της μεθόδου updateById του user service. Το id που δίνεται σαν όρισμα στο HTTP αίτημα λαμβάνεται από τα δεδομένα των χρηστών που έχει ήδη η λίστα με τους χρήστες. Για την διαγραφή του χρήστη αντίστοιχα χρησιμοποιείται η μέθοδος deleteById του user service. Κάτω από την λίστα των χρηστών εμφανίζεται ένα ακόμη κουμπί με label New Employee το οποίο ανοίγει την ίδια φόρμα με το update για την δημιουργία ενός νέου χρήστη. Για την αποστολή του νέου χρήστη στο server χρησιμοποιείται η μέθοδος createUser και ο χρήστης αποθηκεύεται στη λίστα των χρηστών που είναι αποθηκευμένη στο AdminEmployeePageComponent για να εμφανιστεί στην λίστα με τους υπόλοιπους χρήστες.

Το component που χρησιμοποιείται για την εμφάνιση του modal είναι το ίδιο και στις δύο περιπτώσεις και ονομάζεται AdminEmployeeFormComponent. Μέσα στο component χρησιμοποιείται η βιβλιοθήκη angular-material μέσω της οποία εμφωλεύεται ένα αντικείμενο MatDialogRef το οποίο δημιουργεί δυναμικά το modal. Για την παραμετροποίηση του modal δίνεται σαν όρισμα κατά τη δημιουργία του μια λογική τιμή με ονομασία editMode η οποία είναι δηλωμένη true στο update και false στο create. Η τιμή αυτή ελέγχει την εμφάνιση του τίτλου του modal αλλά κυρίως την επιλογή της μεθόδου που θα εκτελεστεί για να στείλει τα δεδομένα στον server, η updateUserById ή η createUser αντίστοιχα. Επίσης για τον χειρισμό των δεδομένων που εισάγει ο χρήστης χρησιμοποιείται μια φόρμα με όλα τα στοιχεία που εμφανίζονται και μπορεί να αλλάξει ο χρήστης.

Η σελίδα των reports κατά την δημιουργία της ανακτά τις παραγγελίες του συστήματος και εμφανίζει εκείνες που είναι καταχωρημένες ως κλειστές ή ακυρωμένες για την εμφάνιση τους από τον χρήστη και μόνο αυτές που έχουν πραγματοποιηθεί μια συγκεκριμένη μέρα. Με το αρχικό άνοιγμα της σελίδας έχει οριστεί η σημερινή ημερομηνία ως προεπιλογή και υπάρχουν δύο κουμπιά που επιτρέπουν την περιήγηση στην ημέρες. Δίπλα από την λίστα των παραγγελιών εμφανίζεται το TicketComponent με όρισμα για την έλεγχο του mode δηλωμένο ως TicketMode.Total το οποίο εμφανίζει το τελικό ποσό της παραγγελίας. Με την επιλογή μιας παραγγελίας από την λίστα δίνεται σαν όρισμα στο προεπισκόπηση της παραγγελίας δίπλα και εμφανίζονται τα στοιχεία της.

Για την διαχείριση και την δημιουργία της αναπαράστασης των τραπεζιών του εστιατορίου υπάρχει ένα πλαίσιο στο AdminMapPageComponent στο οποίο τοποθετούνται τα τραπέζια και ένα κουμπί για την δημιουργία ενός νέου τραπεζιού στο map. Πατώντας το κουμπί καλείται η μέθοδος createDragableIcon η οποία δημιουργεί ένα καινούργιο DragableIconComponent. Το συγκεκριμένο component υλοποιεί από την βιβλιοθήκη drag-drop το directive cdkDrag το οποίο δηλώνει ότι το component μπορεί να συρθεί στην επιφάνεια του map για την τοποθέτηση σε ένα σημείο που επιθυμεί ο διαχειριστής.

Με την τοποθέτηση ενός τραπεζιού εμφανίζεται ένα κουμπί για την διαγραφή του και ένα πεδίο για την εισαγωγή του αναγνωριστικού του τραπεζίου. Η εμφάνιση των δύο αυτών στοιχείων ελέγχεται από την λογική τιμή dropped η οποία γίνεται true όταν τελειώσει το μετακίνηση του dragable στην επιφάνεια του map. Με την εισαγωγή του αναγνωριστικού καλείται η μέθοδος createTable ενώ έχει κρατηθεί η τοποθεσία του τραπεζιού ως η απόσταση από την αρχή της σελίδας με τις συντεταγμένες x και y να αντιπροσωπεύουν τα πεδία left και top του component αντίστοιχα.

Η εμφάνιση των τραπεζιών που έχουν ήδη καταχωρηθεί γίνεται καλώντας την μέθοδο getTables του table service κατά την εμφάνιση της σελίδας και μόλις ανακτηθούν από τον server τα δεδομένα των τραπεζιών εκτελείται η μέθοδος placeTables η οποία δημιουργεί ένα καινούργιο PlacedDragableIconComponent για κάθε τραπέζι δυναμικά και με την χρήση των συντεταγμένων ως το top και το left πεδίο εμφανίζονται στην ίδια θέση που δημιουργήθηκαν. Η ίδια διαδικασία ακολουθείται και για την εμφάνιση των τραπεζιών στον σερβιτόρο με την χρήση ενός απλού εικονιδίου.

Για την δημιουργία και την επεξεργασία του καταλόγου του εστιατορίου εμφανίζονται στο AdminEditMenuPageComponent δύο κουμπιά, ένα για την δημιουργία μιας κατηγορίας και ένα για την δημιουργία ενός προϊόντος. Και στις δύο περιπτώσεις εμφανίζεται ένα modal με τις φόρμες AdminCategoryFormComponent και AdminItemFormComponent. Για την κατηγορία ο χρήστης μπορεί να επιλέξει ένα αναγνωριστικό όνομα, ένα πεδίο isAvailable για τον έλεγχο της διαθεσιμότητας του και έναν τύπο χαρακτηριστικών δηλαδή μια προεπιλεγμένη κατηγορία προϊόντων η οποία μπορεί να είναι είτε coffee είτε drinks. Για το προϊόν η φόρμα περιέχει ένα πεδίο για το αναγνωριστικό όνομα, μια κατηγορία από τις ήδη υπάρχουσες, την τιμή του, τα συστατικά και το πεδίο της διαθεσιμότητας isAvailable. Η εμφάνιση του καταλόγου γίνεται δυναμικά με την εμφώλευση των προϊόντων που περιέχει η κάθε κατηγορία σαν μια λίστα κουμπιών κάτω από τα κουμπιά των κατηγοριών ενώ μέσα σε κάθε κουμπί κατηγορίας ή προϊόντος εμφανίζονται και δύο κουμπιά για την επεξεργασία ή την διαγραφή τους.

Η σελίδα των reservations ακολουθεί το ίδιο μοτίβο με αυτό της σελίδας των χρηστών στην σχεδίασή της με την εμφάνιση των reservations σε μια λίστα με το όνομα της κράτησης, την ημερομηνία και το τραπέζι το οποίο κρατήθηκε. Η κάθε κράτηση μπορεί να επεξεργασθεί ή να διαγραφεί εξ ολοκλήρου από την βάση με την κλήση των μεθόδων updateReservation και deleteReservation του reservation service τα οποία καλούνται από δύο κουμπιά που εμφανίζονται για κάθε κράτηση ξεχωριστά. Όπως και με τους χρήστες υπάρχει ένα ακόμα κουμπί το οποίο εμφανίζει το component AdminReservationFormComponent το οποίο περιέχει την φόρμα στην οποία ο διαχειριστής δημιουργεί μια νέα κράτηση ή επεξεργάζεται μια ήδη υπάρχουσα.

Η τελευταία σελίδα του διαχειριστή εμφανίζεται μέσω του AdminImagesComponent και περιέχει δύο κουμπιά για την αποθήκευση των εικόνων των τραπεζιών και του λογότυπου του εστιατορίου. Με την επιλογή του κάθε κουμπιού εμφανίζεται ένα παράθυρο με την διαχείριση των αρχείων του υπολογιστή του χρήστη και μπορεί να επιλέξει μια εικόνα την οποία με την επιλογή του κουμπιού Upload Image καλεί την μέθοδο uploadImage του image service και στέλνει το αρχείο στο server. Η προεπισκόπηση των εικόνων γίνεται με την χρήση των στατικών URL που παρέχει ο server για την ανάκτηση τους. Οι εικόνες βρίσκονται στο μονοπάτι του server images και έχουν την ονομασία table.png και logo.png αντίστοιχα.

Η κεντρική σελίδα του χρήστη ως σερβιτόρου εμφανίζει το σύνολο των τραπεζιών ως TableButtonComponent που έχει δημιουργηθεί από τον διαχειριστή με τα τραπέζια. Επίσης ανακτάται και η παραγγελία που έχει καταχωρημένη το κάθε τραπέζι για την εμφάνιση της κατάστασης της στο σερβιτόρο. Με την επιλογή του κάθε τραπεζιού καλείται η μέθοδος του component του κουμπιού onTableClick η οποία εκτελεί το navigate δίνοντας ως όρισμα το αναγνωριστικό του τραπεζιού που επιλέχθηκε. Στην συγκεκριμένη σελίδα γίνεται ένα subscribe στο Observable που επιστρέφει η μέθοδος getOrderFromSocket η οποία δέχεται μια παραγγελία από τον server όταν αλλάξει η κατάσταση της και αναβαθμίζεται εκείνη τη στιγμή το αντικείμενο της συγκεκριμένης παραγγελίας.

Η σελίδα στην οποία παραπέμπει το κάθε κουμπί ενός τραπεζιού εμφανίζεται μέσω του ServerOrderPageComponent και περιέχει τα στοιχεία που είναι απαραίτητα για την διαχείριση μιας παραγγελίας από τον σερβιτόρο. Κατά την δημιουργία της με την χρήση του αντικειμένου ActivatedRoute της Angular το οποίο περιέχει τα δεδομένα του μονοπατιού της σελίδας εξάγεται το αναγνωριστικό του τραπεζιού ανακτώνται τα δεδομένα της παραγγελίας.

Στην σελίδα καλούνται τα OrderMenuComponent και TicketComponent για την εμφάνιση του καταλόγου και της παραγγελίας στον χρήστη. Εμφανίζονται επίσης οι πληροφορίες του τραπεζιού και το αναγνωριστικό της παραγγελίας ενώ αν η παραγγελία που έχει ανακτηθεί είναι καινούργια εμφανίζεται ένα μήνυμα που δηλώνει ότι πρόκειται για την δημιουργία νέας παραγγελίας. Υπάρχουν επίσης τέσσερα κουμπιά τα οποία αφορούν την προσθήκη οδηγιών στο ticket, την ακύρωση της παραγγελίας, το κλείσιμό της ή την δημιουργία της. Στην περίπτωση που υπάρχει παραγγελία το κουμπί της δημιουργίας αντικαθιστάται με το κουμπί της ενημέρωσης της. Τα κουμπιά που αφορούν το order καλούν αντίστοιχα τις μεθόδους onCancel, onClose, onUpdate και onPlace όπου οι τρείς πρώτες καλούν την updateOrder του order service και αλλάζουν το πεδίο state της παραγγελίας αντίστοιχα σε Cancelled, Closed και Made ενώ η onPlace καλεί την μέθοδο createOrder του service με την αλλαγή της κατάστασης σε Made.

Η εμφάνιση του καταλόγου διαφέρει από την εμφάνιση στην σελίδα του διαχειριστή, με τα κουμπιά εδώ να υλοποιούνται με το OrderCategoryButtonComponent για την κατηγορία και το ItemCategoryButtonComponent για το προϊόν. Το κουμπί της κατηγορίας εμφανίζει μόνο το όνομα

και μόνο αν η κατηγορία είναι διαθέσιμη με το πέρασμα του πεδίου isAvailable στην ngIf. Με την επιλογή μιας κατηγορίας εμφανίζεται μια λίστα με τα κουμπιά των προϊόντων τα οποία με την επιλογή τους εμφανίζουν ένα πλαίσιο με τα ξεχωριστά χαρακτηριστικά βάσει της κατηγορίας τους. Η εμφάνιση των χαρακτηριστικών είναι δυνατή με την συνθήκη στο ngIf του πλαισίου της ύπαρξης του αντίστοιχου qualifierType ενώ για κάθε qualifierType εμφανίζεται ένα διαφορετικό πλαίσιο.

Με την επιλογή των qualifiers και την επιλογή του κουμπιού DONE ή την επιλογή του κουμπιού του προϊόντος σε περίπτωση που δεν υπάρχουν qualifiers γίνεται emit το event itemClicked το οποίο ενημερώνει το OrderMenuComponent ότι πατήθηκε το συγκεκριμένο κουμπί και καλείται η μέθοδος onItemClicked η οποία λαμβάνει τα στοιχεία των qualifiers αν υπάρχουν, εισάγει το προϊόν σε μία λίστα και προωθεί την λίστα στο παρατηρήσιμο αντικείμενο order του order service το οποίο χρησιμοποιείται σαν ενδιάμεση αποθήκευση της κατάστασης της παραγγελίας για να το προωθήσει στο ticket. Στην συγκεκριμένη περίπτωση επειδή υπάρχει διαστρωμάτωση των components είναι απαραίτητο να χρησιμοποιηθεί το service καθώς η κατάσταση της παραγγελίας χρησιμοποιείται και από το ServerOrderPageComponent και από το TicketComponent οπότε γίνεται και στα δύο αυτά components αυτόματα η ενημέρωσή τους.

Το TicketComponent κατά την δημιουργία του λαμβάνει ως ορίσματα το αντικείμενο της παραγγελίας που δημιουργήθηκε, την μεταβλητή editMode με τιμή true για την επεξεργασία της παραγγελίας με διαγραφή προϊόντων και εισαγωγή οδηγιών, την μεταβλητή ticketMode με την τιμή Total για την εμφάνιση του ποσού της παραγγελίας, την τιμή showInstructions η οποία είναι true εάν πατηθεί το κουμπί εισαγωγής οδηγιών και εμφανίζει το textarea των οδηγιών στο ticket και τον αριθμό του τραπεζιού της παραγγελίας. Το ticket όπως έχει ήδη αναφερθεί δέχεται τα προϊόντα από το order service κάνοντας subscribe στο BehaviorSubject order που περιέχει ενώ κατά την διαγραφή ενός προϊόντος από την λίστα ακολουθείται η ίδια διαδικασία με το order που στέλνεται στο server για δημιουργία ή αναβάθμιση να περιέχει την τιμή από το order του service. Για την οπτικοποίηση της κατάστασης του order μέσα στο ticket χρησιμοποιείται η τιμή της Angular ngStyle στην οποία δίνεται ένα σύνολο από πεδία διακόσμησης CSS το οποίο αλλάζει δυναμικά το περίγραμμα του ticket.

Με την εισαγωγή κάθε νέου προϊόντος, η τιμή του αθροίζεται στο τελικό αποτέλεσμα του orderTotal που κρατάει το order στο orderService ενώ κατά την διαγραφή του εάν η ποσότητα είναι μεγαλύτερη από ένα αφαιρείται το αντίστοιχο πόσο και ένα τεμάχιο ενώ αν υπάρχει ένα μόνο προϊόν αφαιρείται ολόκληρο από την λίστα. Στην εισαγωγή και την διαγραφή προϊόντων λαμβάνεται υπ’ όψιν και η ύπαρξη των qualifiers καθώς όταν προστίθεται ή αφαιρείται ένα προϊόν με διαφορετικά χαρακτηριστικά δεν πρέπει επηρεάζει τα υπόλοιπα προϊόντα με το ίδιο αναγνωριστικό στην λίστα. Η υλοποίηση αυτής της λογικής γίνεται μέσα στην onItemClicked του OrderMenuComponent με τον έλεγχο των qualifiers των ήδη καταχωρημένων προϊόντων στην λίστα και την αύξηση της ποσότητας ή την προσθήκη νέου προϊόντος αντίστοιχα.

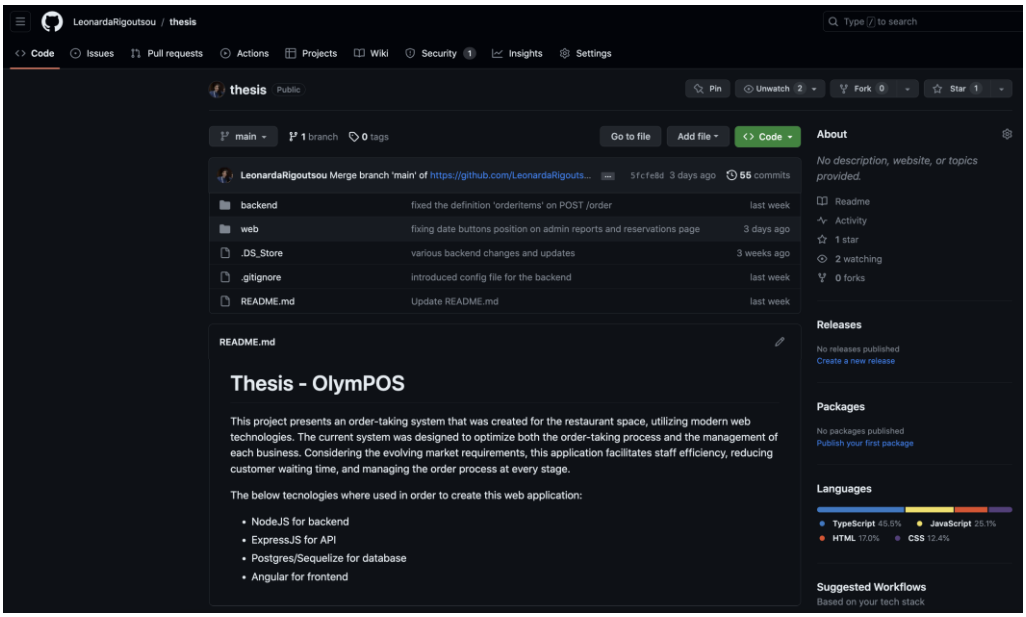
Η τελευταία σελίδα της εφαρμογής είναι η διαχείριση των παραγγελιών από τον παραγγελιοδέκτη. Με την δημιουργία του CookerOrdersPageComponent ανακτώνται οι παραγγελίες και πριν εμφανιστούν φιλτράρονται για να εμφανιστούν μόνο οι ανοιχτές παραγγελίες που υπάρχουν την συγκεκριμένη χρονική στιγμή. Επίσης καλείται η μέθοδος subscribe στην μέθοδο getOrderFromSocket για να δεχτεί μια παραγγελία που στέλνεται από τον server κάθε φορά που αλλάζει η κατάστασή της και να την εισάγει στις λίστα με τις ήδη υπάρχουσες παραγγελίες.

Όταν ανακτηθούν οι παραγγελίες δημιουργείται δυναμικά μέσω της χρήσης του ngFor στην λίστα που έχουν αποθηκευτεί για κάθε μια παραγγελία, ένα TicketComponent στο οποίο δίνεται σαν όρισμα στο

πεδίο order η αντίστοιχη παραγγελία από την λίστα. Παρέχεται επίσης το αναγνωριστικό του τραπεζιού, η μεταβλητή ticketMode με τιμή Buttons για την εμφάνιση των δύο κουμπιών που χειρίζονται την κατάσταση της παραγγελίας, η τιμή true στο πεδίο showInstructions για την εμφάνιση των οδηγιών και η τιμή false στο πεδίο editMode για τον περιορισμό της εκούσιας ή ακούσιας αλλαγής των προϊόντων της παραγγελίας.

3.7 Η εφαρμογή στο GitHub

Η υλοποίηση της συγκεκριμένης εφαρμογής βρίσκεται σε ένα αποθετήριο στο GitHub. Μέσα στο αποθετήριο υπάρχει ένας φάκελος ο οποίος περιέχει την υλοποίηση του server, με όνομα backend και ένας φάκελος με όνομα web, ο οποίος περιέχει την υλοποίηση της Angular εφαρμογής. Για την απόκρυψη των ευαίσθητων δεδομένων της εφαρμογής, όπως είναι το URL της βάσης δεδομένων και το κρυφό κλειδί που χρησιμοποιείται για την κρυπτογράφηση του JWT token χρησιμοποιείται στην εφαρμογή το αρχείο .env το οποίο παρέχει αυτές τις πληροφορίες ως παραμέτρους κατά την εκκίνηση του server. Το συγκεκριμένο αρχείο δεν υπάρχει στο αποθετήριο και είναι απαραίτητη η δημιουργία του για την εκκίνηση της εφαρμογής από κάποιον χρήστη. Το μαρκάρισμα του συγκεκριμένου αρχείου ως μη διαμοιράσιμο περιγράφεται στο αρχείο .gitignore το οποίο αποτελεί κοινή πρακτική στα αποθετήρια που χρησιμοποιούν το git.



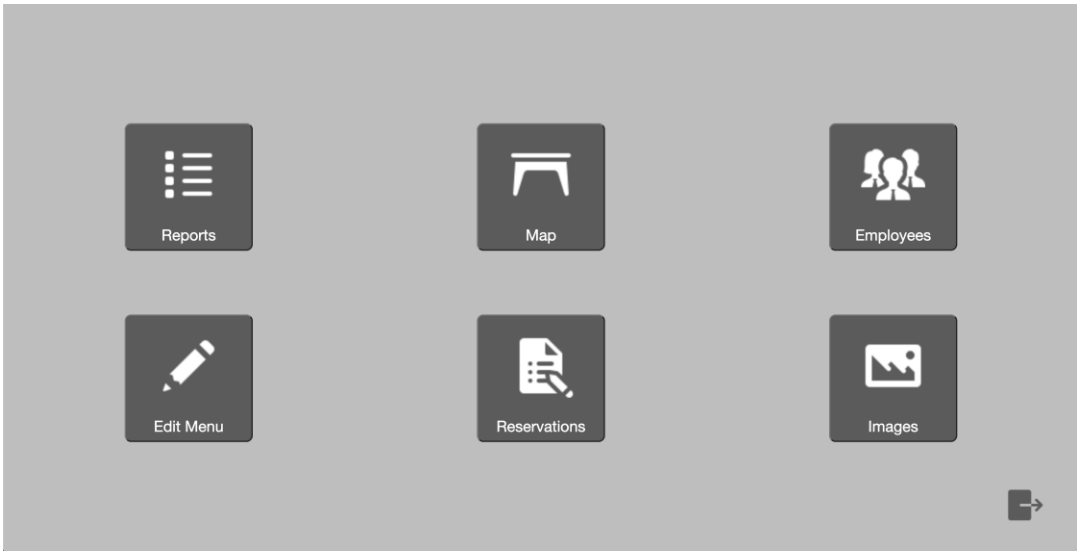
Σχήμα 3.8: Η εφαρμογή στο GitHub

Κεφάλαιο 4ο: Παρουσίαση της εφαρμογής

Στο παρόν κεφάλαιο, θα παρουσιαστεί η εφαρμογή κατηγοριοποιημένη με βάση της σελίδες που βλέπει και έχει πρόσβαση κάθε ρόλος στο σύστημα. Η κεντρική σελίδα της εφαρμογής πριν την αυθεντικοποίηση των χρηστών είναι κοινή για κάθε χρήστη και δίνει μόνο την δυνατότητα εισαγωγής μόνο του username και password για την αυθεντικοποίηση του χρήστη στο σύστημα. Κάθε χρήστης που έχει δικαίωμα σύνδεσης στην εφαρμογή μπορεί να εισάγει τα στοιχεία του και να ανακατευθυνθεί στην αντίστοιχη κεντρική σελίδα, ανάλογα με τον ρόλο του, που θα αναλυθεί στα παρακάτω κεφάλαια.

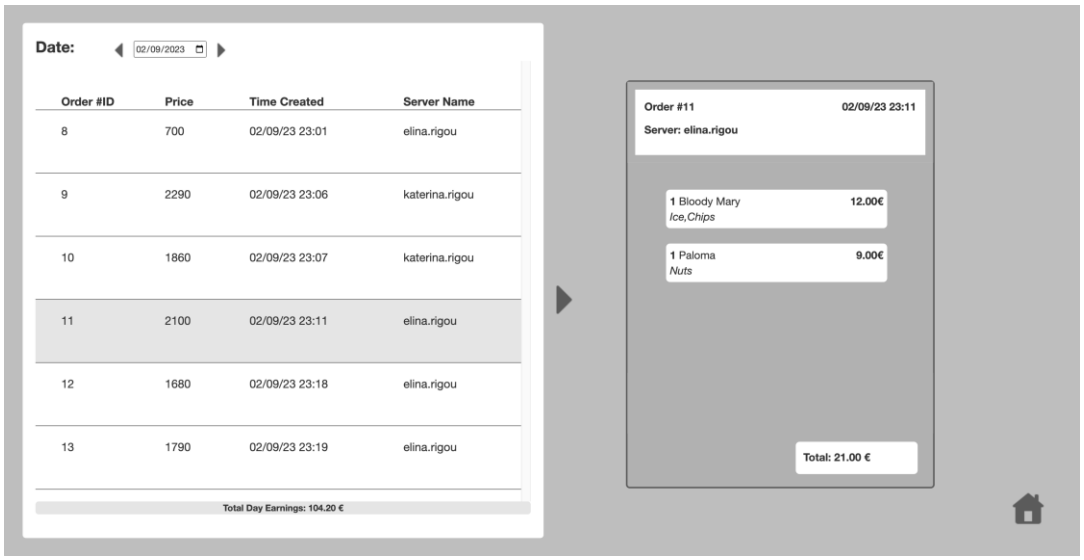
4.1 Διαχειριστής

Η κεντρική σελίδα του διαχειριστή παρέχει την έξι κουμπιά τα οποία παραπέμπουν στις έξι δυνατότητες που έχει ο διαχειριστής.



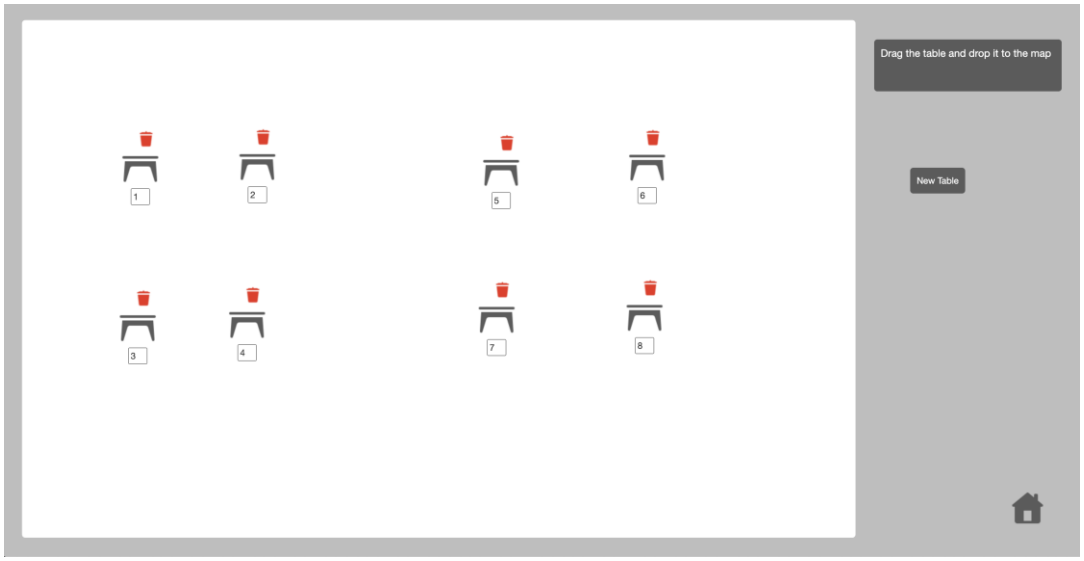
Σχήμα 4.1: Κεντρική σελίδα διαχειριστή με 6 λειτουργίες (/admin)

Στην πρώτη λειτουργία, με τίτλο “Reports”, ο διαχειριστής μπορεί να περιηγηθεί στις παραγγελίες που έχουν ταξινομηθεί ανά ημέρα οι οποίες έχουν καταχωρηθεί ως κλειστές (πληρωμένες) ή ακυρωμένες. Με την επιλογή της κάθε παραγγελίες φορτώνονται όλα τα στοιχεία που διαθέτει, όπως είναι τα προϊόντα, οι ποσότητα, οι τιμές των προϊόντων και το σύνολο της παραγγελίας καθώς και το νούμερο, η ώρα και ο σερβιτόρος που την κατέγραψε. Ο διαχειριστής μπορεί να δει το συνολικό κέρδος ανά ημέρα στο τέλος της λίστας καθώς και να επιλέξει συγκεκριμένη ημερομηνία σε οποιαδήποτε μήνα με την χρήση ενός πεδίου ημερομηνίας.



Σχήμα 4.2: Προβολή κλειστών και ακυρωμένων παραγγελιών (/admin/reports)

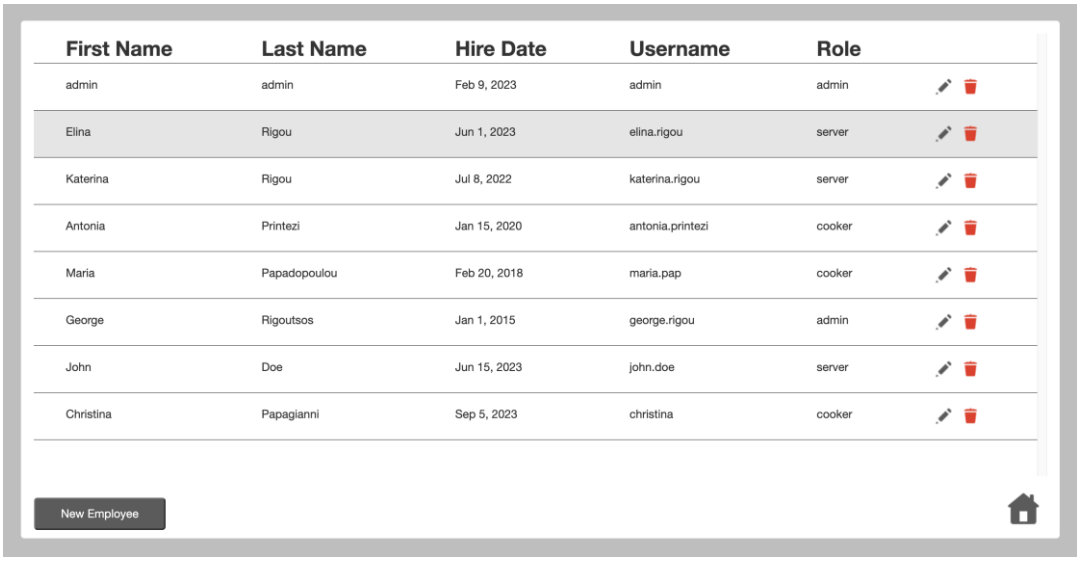
Στην δεύτερη λειτουργία, με τίτλο “Map”, ο διαχειριστής μπορεί να δημιουργήσει την αναπαράσταση του εστιατορίου ως μία απεικόνιση των τραπεζιών και να ορίσει τα αναγνωριστικά του. Πατώντας το κουμπί “New Table” εμφανίζεται ένα τραπέζι, το οποίο σέρνοντας το μπορεί να το τοποθετήσει σε οποιοδήποτε σημείο του λευκού πλαισίου. Με την τοποθέτηση ενός τραπεζιού στο map ο διαχειριστής δεν μπορεί να τροποποιήσει την τοποθεσία του, παρά μόνο να το διαγράψει εξ ολοκλήρου και να σύρει ένα νέο τραπέζι στην τοποθεσία που επιθυμεί.



Σχήμα 4.3: Δημιουργία και επεξεργασία αναπαράστασης τραπεζιών (/admin/map)

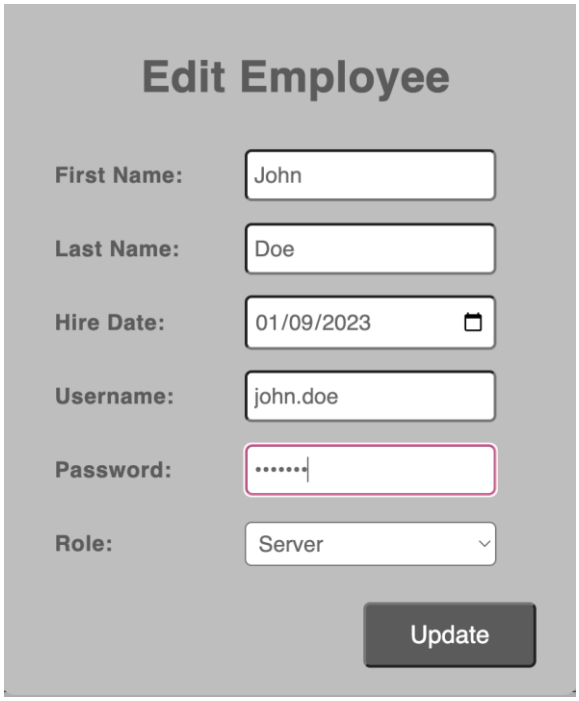
Στην τρίτη λειτουργία, με τίτλο “Employees”, ο διαχειριστής μπορεί να δει πληροφορίες για τον κάθε υπάλληλο της επιχείρησης, καθώς και να εισάγει έναν νέο υπάλληλο. Πατώντας το “New Employee” εμφανίζεται πλαίσιο για την εισαγωγή στοιχείων του υπαλλήλου, όπως είναι το ονοματεπώνυμο, η

ημερομηνία πρόσληψης, το username και password που θα χρησιμοποιεί και ο ρόλος που θα έχει. Μόλις πατήσει “Create”, ο υπάλληλος καταχωρείται στο σύστημα.



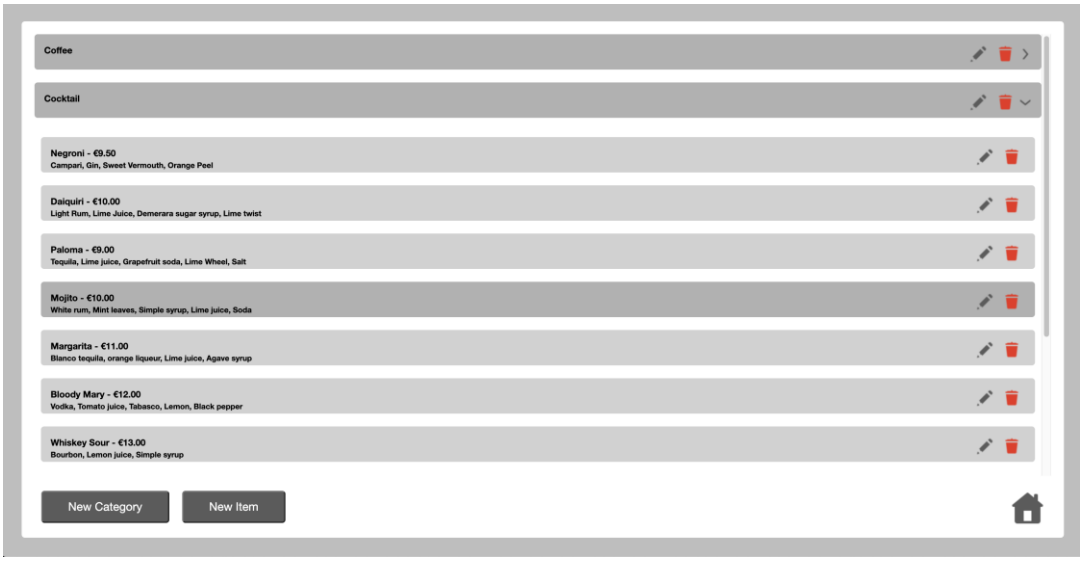
Σχήμα 4.4: Προβολή και διαχείριση των υπαλλήλων (/admin/employees)

Στα δεξιά από κάθε υπάλληλο δίνεται η δυνατότητα τροποποίησης των στοιχείων του μέσω από ένα πλαίσιο, παρόμοιο με αυτό της εισαγωγής. Δίνεται επίσης η δυνατότητα διαγραφής αυτού από το εικονίδιο του κόκκινου κάδου.



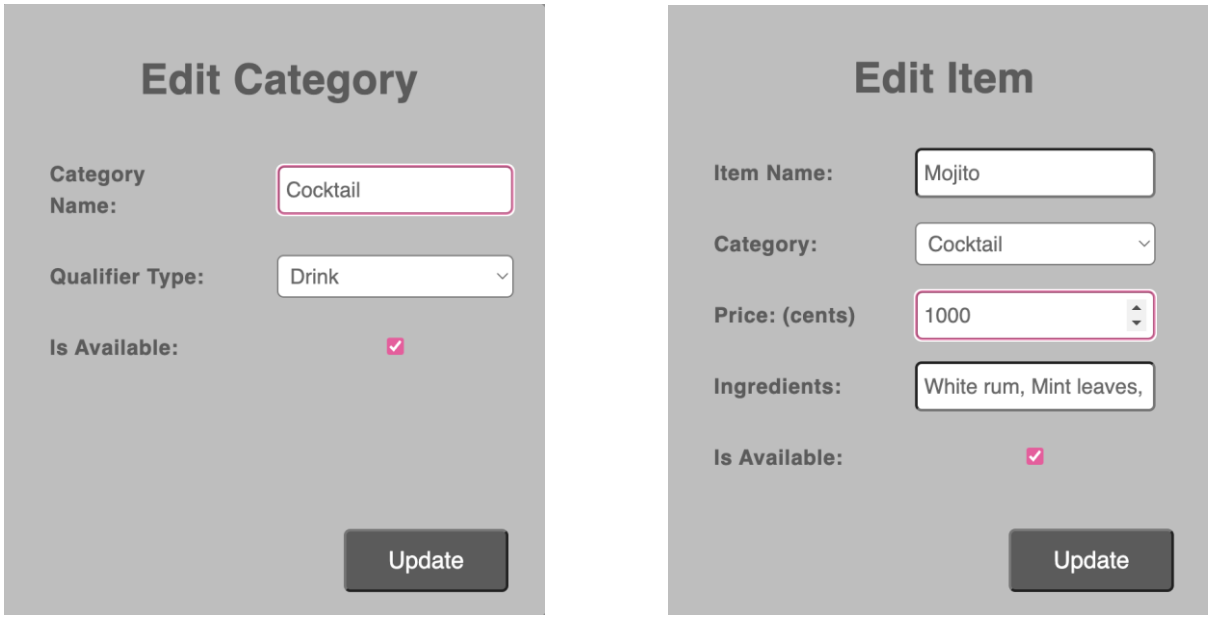
Σχήμα 4.5: Πλαίσιο επεξεργασίας στοιχείων υπαλλήλου

Στην τέταρτη λειτουργία με τίτλο “Edit Menu”, ο διαχειριστής μπορεί να δημιουργήσει τις κατηγορίες των προϊόντων και τα προϊόντα του καταλόγου που θα υπάρχουν στην επιχείρηση. Από τα κουμπιά “New Category” και “New Item” εμφανίζεται το πλαίσιο για την εισαγωγή στοιχείων της εκάστοτε κατηγορίας και του εκάστοτε προϊόντος. Πιο συγκεκριμένα, η κάθε κατηγορία αποτελείται από το όνομα, των τύπο των qualifier που θα έχει, το οποίο δεν είναι υποχρεωτικό, και την διαθεσιμότητα της. Αντίστοιχα, για κάθε προϊόν, ορίζεται το όνομα του, η κατηγορία στην οποία ανήκει, η τιμή του, τα υλικά που αποτελείται και η διαθεσιμότητα του. Η διαθεσιμότητα και στις δύο περιπτώσεις ορίζει το αν θα εμφανίζεται η κατηγορία ή το προϊόν στον κατάλογο του σερβιτόρου.



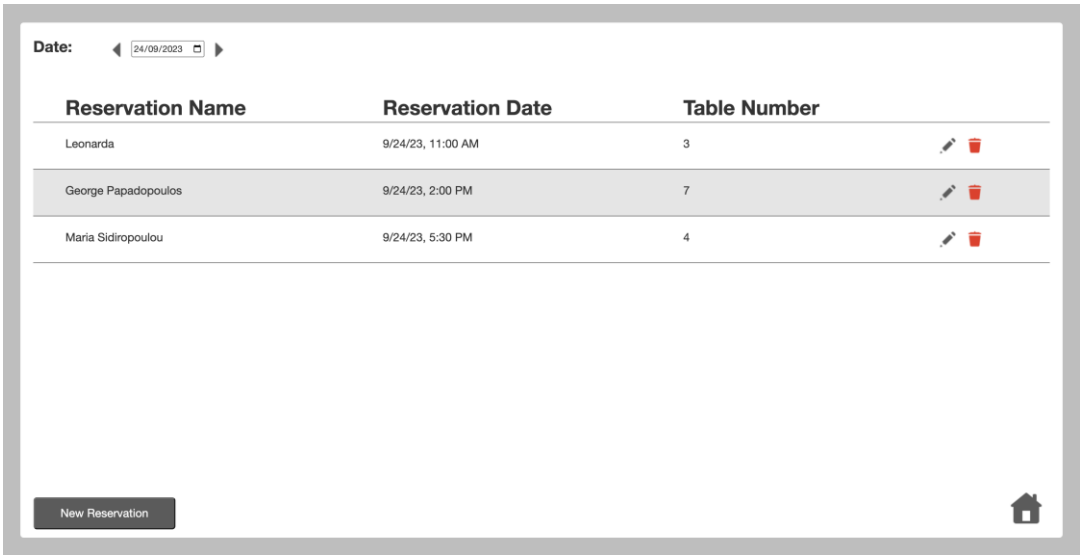
Σχήμα 4.6: Προβολή και διαχείριση κατηγοριών και προϊόντων (/admin/editmenu)

Όπως και στις άλλες λειτουργίες, στα δεξιά από κάθε κατηγορία ή προϊόν δίνεται η δυνατότητα επεξεργασίας και διαγραφής του.



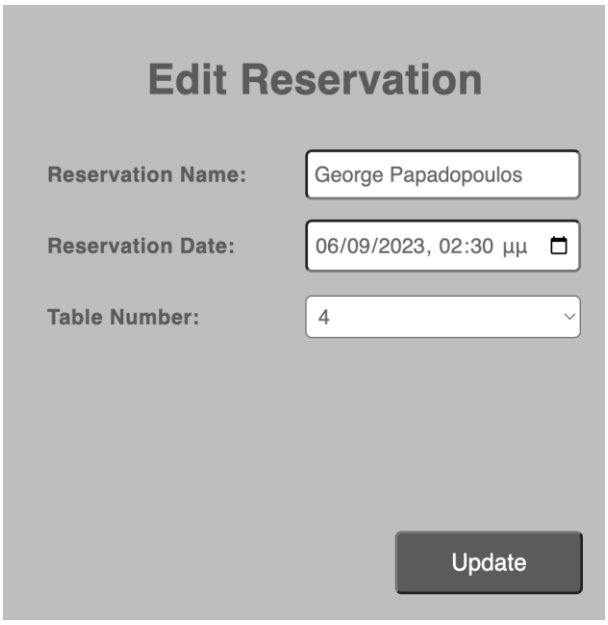
Σχήμα 4.7: Πλαίσιο επεξεργασίας κατηγορίας και προϊόντος

Στην πέμπτη λειτουργία, με τίτλο “Reservations”, ο διαχειριστής μπορεί να δει και να διαχειριστεί τις κρατήσεις της επιχείρησης. Πιο συγκεκριμένα, εμφανίζεται μία φιλτραρισμένη λίστα με τις κρατήσεις που αντιστοιχούν για την ημερομηνία που έχει επιλεχθεί. Η κάθε κράτηση αποτελείται από το όνομα κράτησης, την ημερομηνία κράτησης και τον αριθμό τραπεζιού που ανήκει.



Σχήμα 4.8: Προβολή και διαχείριση κρατήσεων (/admin/reservations)

Ο διαχειριστής μπορεί να εισάγει μία κράτηση μέσω του πλαισίου που εμφανίζεται όταν πατήσει το κουμπί “New Reservation”. Στα δεξιά από κάθε κράτηση δίνεται η δυνατότητα τροποποίησης της αλλά και διαγραφή αυτής.



Σχήμα 4.9: Πλαίσιο επεξεργασίας στοιχείων κράτησης

Στην έκτη λειτουργία και τελευταία, με τίτλο “Images”, ο διαχειριστής έχει την δυνατότητα να ανεβάσει δύο εικονίδια. Το πρώτο αφορά το λογότυπο που επιθυμεί να φαίνεται στην κεντρική σελίδα, στην είσοδο της εφαρμογής. Συνήθως αυτό είναι το εικονίδιο της εκάστοτε επιχείρησης. Το δεύτερο εικονίδιο αφορά την εικόνα που επιθυμεί να εμφανίζεται σαν τραπέζι, το οποίο εμφανίζεται στον σερβιτόρο και στην σελίδα του διαχειριστή που δημιουργεί την αναπαράσταση με την διάταξη των τραπεζιών.

Σχήμα 4.10: Δυνατότητα φόρτωσης εικονιδίων για τροποποίηση της εφαρμογής (/admin/images)

4.2 Παραγγελιολήπτης

Η κεντρική σελίδα του παραγγελιολήπτη που δίνεται στο ρόλο του σερβιτόρου είναι η σελίδα που εμφανίζει τα τραπέζια της επιχείρησης με την δομή που έχει οριστεί από τον διαχειριστή και εμφανίζουν τον αριθμό του τραπεζιού και το την κατάσταση της παραγγελίας στο συγκεκριμένο τραπέζι. Ο σερβιτόρος από την συγκεκριμένη σελίδα μπορεί μόνο να επιλέξει ένα τραπέζι για να μεταφερθεί στην σελίδα επεξεργασίας της παραγγελίας του και να αποσυνδεθεί από τον εφαρμογή.

Σχήμα 4.11: Κεντρική σελίδα παραγγελιολήπτη με εμφάνιση κατάστασης τραπεζιού (/server/map)

Όταν ο σερβιτόρος μεταφερθεί στην σελίδα επεξεργασίας της εκάστοτε παραγγελίας, το πάνω μέρος της σελίδας εμφανίζεται είτε ο αριθμός της παραγγελίας είτε το New Order το οποίο υποδηλώνει ότι ξεκινάει μία νέα παραγγελία και το νούμερο του τραπεζιού το οποίο επέλεξε.

Στο αριστερό τμήμα της σελίδας, εμφανίζονται όλες οι διαθέσιμες κατηγορίες. όταν ο σερβιτόρος επιλέξει μία κατηγορία, εμφανίζονται όλα τα διαθέσιμα προϊόντα που υπάρχουν για την αντίστοιχη κατηγορία. Όταν ένα κατηγορία έχει qualifiers ο σερβιτόρος μπορεί να επεκτείνει το προϊόν και να επιλέξει τα αντίστοιχα χαρακτηριστικά. Σε περίπτωση που δεν έχει οριστεί κάποιος qualifier για κάποια κατηγορία, όλα τα προϊόντα προστίθενται άμεσα στην παραγγελία.

Στο δεξί τμήμα της σελίδας, εμφανίζεται η προεπισκόπηση της παραγγελίας. Ο σερβιτόρος μπορεί να δει την ημερομηνία με την ώρα που ξεκίνησε η παραγγελία, ποιός σερβιτόρος ξεκίνησε αυτή την παραγγελία καθώς και το νούμερο της συγκεκριμένης παραγγελίας. Στα αριστερά εμφανίζεται η ποσότητα του κάθε προϊόντος και στα δεξιά εμφανίζεται η τιμή του και η δυνατότητα διαγραφής του. Όταν ο σερβιτόρος επιλέξει να διαγράψει ένα προϊόν, το προϊόν διαγράφεται ανά τεμάχιο. Στο κάτω μέρος της παραγγελίας εμφανίζεται το τελικό σύνολο.

Κάτω από την παραγγελία, εμφανίζονται οι παρακάτω τέσσερις επιλογές:

- Add Instructions
- Cancel order
- Close order
- Place order

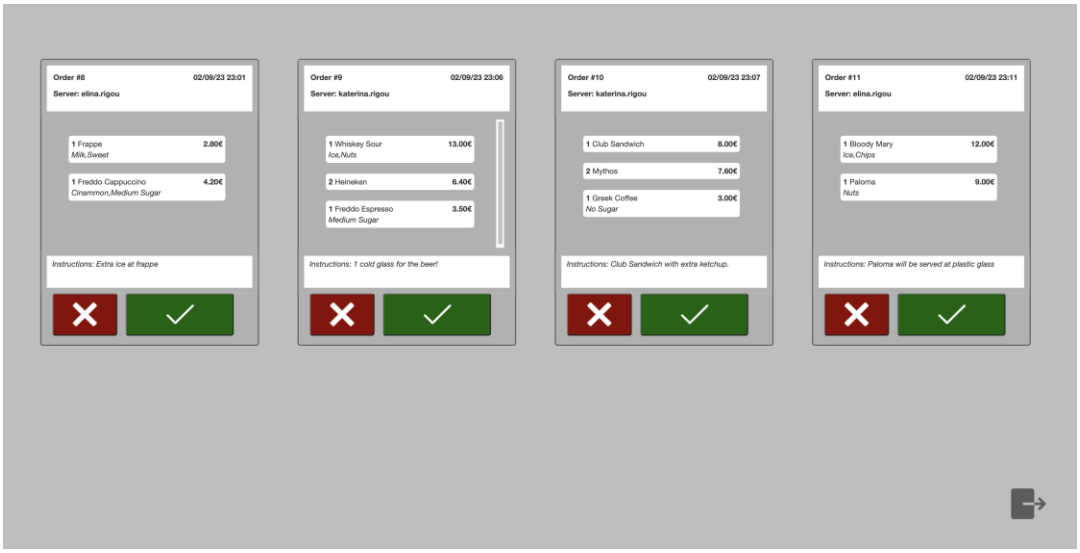
Ο σερβιτόρος έχει την επιλογή να πληκτρολογήσει οδηγίες για την παραγγελία είτε κάποιο σχόλιο που θέλει σχετικά με την παραγγελία (Add Instructions). Στην συνέχεια, ο σερβιτόρος μπορεί να στείλει άμεσα την παραγγελία στην παραγωγή (Place order). Οι υπάλληλοι με ρόλο cooker θα δουν την παραγγελία και θα ξεκινήσουν άμεσα την προετοιμασία αυτής. Επιπλέον, ο σερβιτόρος μπορεί να ακυρώσει την παραγγελία (Cancel order), ενημερώνοντας την παραγωγή για την αλλαγή της κατάστασης της παραγγελίας, ώστε να σταματήσουν την προετοιμασία της. Τέλος, αφού πληρωθεί η παραγγελία, ο σερβιτόρος κλείνει την παραγγελία (Close order). Με το κλείσιμο της παραγγελίας ελευθερώνεται το τραπέζι για νέα παραγγελία και μαρκάρεται στο σύστημα σαν κλειστή.

Σχήμα 4.12: Σελίδα επεξεργασίας παραγγελιών (/server/order?table={tableNumber})

4.3 Παραγγελιοδέκτης

Η κεντρική σελίδα του παραγγελιοδέκτη που δίνεται στον ρόλο της παραγωγής, συνήθως είναι κουζίνα στον χώρο της εστίασης, είναι η σελίδα που εμφανίζονται όλες οι παραγγελίες που έχουν αποσταλεί από τον σερβιτόρο και δεν έχουν ακυρωθεί ή κλείσει. Όλες οι παραγγελίες εμφανίζονται σε ένα πλέγμα με την σειρά που έχουν σταλεί.

Οι παραγγελιοδέκτες, έχουν δύο δυνατότητες, είτε να ακυρώσουν την παραγγελία είτε να ετοιμάσουν την παραγγελία. Όταν επιλέξουν μία από τις δύο επιλογές, ο σερβιτόρος μπορεί να δει άμεσα την κατάσταση της παραγγελίας όπως αναφέρθηκε παραπάνω, και η παραγγελία εξαφανίζεται από την οθόνη, καθώς είτε ετοιμάστηκε, είτε ακυρώθηκε.



Σχήμα 4.13: Κεντρική σελίδα παραγγελιοδεκτών (/cooker/orders)

Κεφάλαιο 5ο: Συμπεράσματα και Μελλοντικές επεκτάσεις

5.1 Συμπεράσματα

Στην παρούσα έρευνα και υλοποίηση είδαμε με την αναγκαιότητα του λογισμικού στην σύγχρονη εποχή. Αξιοποιώντας σύγχρονες τεχνολογίες web, δημιουργήθηκε η εφαρμογή του συστήματος παραγγελιοληψίας που παρέχει πληθώρα δυνατοτήτων για διάφορους ρόλους χρηστών, όπως παρουσιάστηκε και στα παραπάνω κεφάλαια. Πιο συγκεκριμένα, παρουσιάστηκε η ενσωμάτωση ενός ολοκληρωμένου συστήματος POS στον χώρο της εστίασης και αναλύθηκε το πώς αυτό συμβάλλει στην βελτίωση της διαχείρισης των παραγγελιών από τους υπαλλήλους και κατ' επέκταση την καλύτερη εξυπηρέτηση των πελατών. Για τους διαχειριστές του συστήματος η εφαρμογή εξασφάλισε μία λύση για την ολοκληρωμένη εικόνα της επιχείρησης παρέχοντας τους την δυνατότητα της απλοποιημένης επεξεργασίας και ανάγνωσης των δεδομένων των υπάλληλων και των παραγγελιών που πραγματοποιούνται με απώτερο στόχο την εύρυθμη λειτουργία της επιχείρησης.

5.2 Μελλοντικές Επεκτάσεις

Μία χρήσιμη αναβάθμιση στην συγκεκριμένη εφαρμογή θα αποτελούσε η παροχής της δυνατότητας της αυτοεξυπηρετούμενης παραγγελιοληψίας (Self-Ordering) με την προβολή του καταλόγου μέσω της σάρωσης ενός QR code και την αποστολή της παραγγελίας χωρίς την ανάγκη ενός σερβιτόρου. Με αυτόν τον τρόπο θα εξυπηρετούνταν περισσότεροι πελάτες και θα αποδέσμευσε σημαντικό χρόνο από τις υπόλοιπες αρμοδιότητες του υπαλλήλου. Μία ακόμα σημαντική αναβάθμιση θα ήταν η ενσωμάτωση στο σύστημα παραγγελιοληψίας ενός αυτόνομου συστήματος πληρωμών για την διευκόλυνση της πληρωμής από τους πελάτες και την αποφυγή λαθών των υπαλλήλων με την χρήση εξωτερικών τερματικών ή ταμειακών μηχανών.

Για την διαχείριση περισσότερων εστιατορίων θα μπορούσε να αναπτυχθεί η δυνατότητα επικοινωνίας των ξεχωριστών συστημάτων τους με ένα κεντρικό σύστημα διαχείρισης στο οποίο θα συγκεντρώνονται οι παραγγελίες κάθε εστιατορίου και θα διευκολύνεται η αναβάθμιση ενός ενιαίου καταλόγου που μπορεί να διαμοιράζονται μεταξύ τους. Επιπλέον, η λειτουργία της εφαρμογής μπορεί να χωριστεί σε μικρότερα και αυτόνομα υποσυστήματα τα οποία θα αποστέλλουν σε καθημερινή βάση τα στοιχεία των παραγγελιών και των εσόδων παρέχοντας έτσι την συνολική διαχείριση μία επιχείρησης που αποτελείται από πολλά καταστήματα.

Τέλος, για την τεχνική αναβάθμιση της εφαρμογής μπορεί να μεταφερθεί η εξυπηρέτηση της και η διαχείριση της βάσης σε μία κοινή υποδομή στο cloud. Το όφελος μιας τέτοιας ενέργειας θα ήταν η εξασφάλιση της διαθεσιμότητας των υπηρεσιών στους υπαλλήλους, η κλιμάκωση της εφαρμογής χωρίς την απαίτηση αναδημιουργίας της και η παροχή ασφάλειας στα δεδομένα της εφαρμογής από κινδύνους απωλειών καθώς και επιθέσεων με την χρήση διπλοτύπων σε περισσότερες τοποθεσίες.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Timothy Christian Lethbridge, Robert Laganiere, *Object-Oriented Software Engineering: Practical Software Development Using UML and Java*. Belmont, McGraw-Hill Publishing Company, 2004.

[2] Ian Sommerville, Software Engineering (8th Edition). England: Pearson Education Limited, 2007.

[3] Μ. Γιακουμάκης, Ν. Διαμαντίδης, Τεχνολογία Λογισμικού. Αθήνα: Εκδόσεις Σταμούλη Α.Ε., 2009.

[4] Raghu Ramakrishnan, Johannes Gehrke, Database Management Systems. New York: McGraw-Hill Higher Education, 2003.

[5] Andrew M. St. Laurent, Understanding Open Source and Free Software Licensing. USA: O'Reilly Media, 2004.

[6] "Ελεύθερο Λογισμικό και Λογισμικό Ανοιχτού Κώδικα" https://iglezakis.gr/wp-content/uploads/2016/04/OiAdiesLogismikouAniktouKodika.pdf Accessed: Sep 8, 2023

[7] "Point of Sale Systems for Table Restaurants" https://digitalcommons.fiu.edu/cgi/viewcontent.cgi?article=1179&context=hospitalityreview Accessed: Sep 10, 2023

[8] "NodeJS cross-platform runtime environment" https://nodejs.org/en/about Accessed: Aug 25, 2023

[9] "V8 Javascript Engine" https://v8.dev/blog/ignition-interpreter Accessed: Aug 26, 2023

[10] "Express library documentation" https://expressjs.com/en/4x/api.html Accessed: Aug 30, 2023

[11] "PostgreSQL documentation" https://www.postgresql.org/about/ Accessed: Aug 4, 2023

[12] "Sequelize Node ORM" https://sequelize.org/docs/v6/other-topics/migrations/ Accessed: Aug 10, 2023

[13] "TypeScript documentation" https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html Accessed: Sep 3, 2023

[14] "Angular framework documentation" https://angular.io/docs Accessed: Sep 6, 2023

[15] "Bcrypt library documentation" https://www.npmjs.com/package/bcryptjs Accessed: Sep 2, 2023

[16] "Bcrypt Algorithm" https://cvsweb.openbsd.org/cgi-bin/cvsweb/src/lib/libc/crypt/bcrypt.c?rev=1.1&content-type=text/x-cvsweb-markup Accessed: Sep 2, 2023

[17] "Body-parser library" https://www.npmjs.com/package/body-parser Accessed: Aug 29, 2023

[18] "Cross-Origin Resource Sharing documentation" https://www.npmjs.com/package/cors Accessed: Aug 30, 2023

[19] "Cross-Origin Resource Sharing documentation" https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS Accessed: Aug 30, 2023

[20] "JSON web token" https://www.npmjs.com/package/jsonwebtoken Accessed: Aug 7, 2023

[21] "RFC for JSON Web Token" https://datatracker.ietf.org/doc/html/rfc7519 Accessed: Aug 7, 2023

[22] “Multer library for uploading files” https://www.npmjs.com/package/multer Accessed: Aug 29, 2023

[23] ”Socket.io library” https://socket.io/docs/v4/ Accessed: Aug 28, 2023

[24] “RFC The WebSocket Protocol” https://datatracker.ietf.org/doc/html/rfc6455 Accessed: Aug 28, 2023

[25] “Angular Material documentation” https://material.angular.io/guide/getting-started Accessed: Aug 11, 2023

[26] “RxJS documentation” https://rxjs.dev/guide/overview Accessed: Aug 27, 2023

[27] “Socket.io for client documentation” https://www.npmjs.com/package/socket.io-client Accessed: Aug 30, 2023

[28] ”JWT decode library” https://www.npmjs.com/package/jwt-decode Accessed: Sep 4, 2023

[29] “Figma design tool” https://en.wikipedia.org/wiki/Figma_(software) Accessed: Sep 2, 2023

[30] “VSCode editor” https://code.visualstudio.com/docs Accessed: Sep 3, 2023

[31] “ElephantSQL database hosting service” https://www.elephantsql.com/docs/index.html Accessed: Sep 18, 2023

[32] “PGAdmin development platform for PostgreSQL” https://www.pgadmin.org/development/ Accessed: Aug 6, 2023

ΠΑΡΑΡΤΗΜΑ Α: ΚΩΔΙΚΑΣ BACKEND

backend/server.js

```
1  const express = require('express');
2  const swagger = require('swagger-ui-express');
3  const bodyParser = require('body-parser');
4  const cors = require('cors');
5  const multer = require('multer');
6  const path = require('path');
7  const { app, server } = require('./util/socket');
8
9  const authroutes = require('./routes/authroutes');
10 const userroutes = require('./routes/userroutes');
11 const tableroutes = require('./routes/tableroutes');
12 const categoryroutes = require('./routes/categoryroutes');
13 const itemroutes = require('./routes/itemroutes');
14 const orderroutes = require('./routes/orderroutes');
15 const reservationroutes = require('./routes/reservationroutes');
16 const imageroutes = require('./routes/imageroutes');
17
18 const db = require('./util/database');
19
20 const swaggerConfig = require('./swagger.json');
21
22 app.use(cors());
23 app.use(bodyParser.json());
24 app.use('/', express.static(path.join(__dirname, 'dist')));
25 app.use('/api-docs', swagger.serve, swagger.setup(swaggerConfig));
26 app.use('/api', authroutes);
27 app.use('/api', userroutes);
28 app.use('/api', tableroutes);
29 app.use('/api', categoryroutes);
30 app.use('/api', itemroutes);
31 app.use('/api', orderroutes);
32 app.use('/api', reservationroutes);
33 app.use('/api', imageroutes);
34 app.use('/images', express.static(path.join(__dirname, 'images')));
35
36
37 //error handling
38 app.use((err, req, res, next) => {
39     console.log(err);
40     if (!err.statusCode) {
41         res.status(500).json({
42             message: "Internal Server Error"
43         })
44         return;
45     }
46     res.status(err.statusCode).json({
47         message: err.message
48     });
49 });
50
51 db.sequelize.authenticate().then(() => {
52     console.log('Connection has been established successfully.');
53 }).catch((error) => {
54     console.error('Unable to connect to the database: ', error);
55 });
56
```

```
57 db.sequelize
58     .sync()
59     // .sync({ alter: true })
60     .then(result => {
61         server.listen(8080);
62     })
63     .catch(err => {
64         console.log(err);
65     })
66
```

backend/controllers/authcontroller.js

```
const express = require('express');
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');
const dotenv = require('dotenv');

dotenv.config();
const SECRET = process.env.SECRET;

const db = require('../util/database');

dotenv.config();

const login = (req, res, next) => {
    const { username, password } = req.body;
    let loadedUser;

    if (!username) {
        return res.status(400).json({ message: 'Username does not exist.' });
    }

    if (!password) {
        return res.status(400).json({ message: 'Password does not exist.' });
    }

    db.User.findAll({
        where: {
            username: username
        }
    }).then(users => {
        if (users.length === 0) {
            const error = new Error('User not found.');
            error.statusCode = 404;
            throw error;
        }
        return users[0];
    }).then(user => {
        loadedUser = user;
        return bcrypt.compare(password, user.password);
    }).then(isEqual => {
        if (!isEqual) {
            const error = new Error('Incorrect password.');
            error.statusCode = 404;
            throw error;
        }
        const token = jwt.sign({
            username: loadedUser.username,
            userId: loadedUser.userId.toString(),
            role: loadedUser.role
        },
            SECRET
        );
        res.status(200).json({ token: token, userId: loadedUser.userId.toString()
});
    }).catch(err => {
        next(err);
    });
```

```
}

module.exports = login;
```

backend/controllers/usercontroller.js

```
const db = require('../util/database');
const bcrypt = require('bcryptjs');

const getUsers = (req, res, next) => {
    db.User.findAll({
        order: [
            ['userId', 'ASC']
        ],
        attributes: { exclude: ['password'] }
    }).then(users => {
        if (users.length === 0) {
            const error = new Error('User not found.');
            error.statusCode = 404;
            throw error;
        }

        res.status(200).json({ users });
    }).catch(err => {
        next(err);
    });
};

const createUser = (req, res, next) => {
    const { username, password, firstName, lastName, hireDate, role } = req.body;

    let hashedpassword;
    bcrypt.hash(password, 10)
        .then(password => {
            hashedpassword = password;

            db.User.findOne({
                where: {
                    username: username
                }
            }).then(user => {
                if (user) {
                    const error = new Error('User already exists.');
                    error.statusCode = 409;
                    throw error;
                }

                db.User.create({
                    username: username,
                    password: hashedpassword,
                    firstName: firstName,
                    lastName: lastName,
                    hireDate: hireDate,
                    role: role
                }).then(user => {
                    res.status(200).json({ user });
                }).catch(err => {
                    next(err);
                });
            }).catch(err => {
                next(err);
            });
```

```
        });
};

const deleteUser = (req, res, next) => {
    const userId = req.params.id;
    if (!Number.isInteger(+userId)) {
        const error = new Error('User Id is not an integer number.');
        error.statusCode = 400;
        throw error;
    }
    db.User.findOne({
        where: {
            userId: userId
        }
    }).then(userFound => {
        db.User.destroy({
            where: {
                userId: userId
            }
        }).then(user => {
            if (!user) {
                const error = new Error('User not found.');
                error.statusCode = 404;
                throw error;
            }

            res.status(200).json(userFound);
        }).catch(err => {
            next(err);
        });

    }).catch(err => {
        next(err);
    });
};

const updateUser = (req, res, next) => {
    const { password } = req.body;
    const userId = req.params.id;
    bcrypt.hash(password, 10)
        .then(hashedpassword => {
            req.body.password = hashedpassword;

            if (!Number.isInteger(+userId)) {
                const error = new Error('User Id is not an integer number.');
                error.statusCode = 400;
                throw error;
            }
            db.User.update({ ...req.body }, {
                where: {
                    userId: userId
                },
                returning: true
            }).then(updatedUser => {
                res.status(200).json(updatedUser);
            }).catch(err => {
                next(err);
            });
        }).catch(err => {
            next(err);
        })
};


exports.getUsers = getUsers;
exports.createUser = createUser;
exports.deleteUser = deleteUser;
exports.updateUser = updateUser;
```

backend/controllers/categorycontroller.js

```
const db = require('../util/database');

const getCategories = async (req, res, next) => {

    try {
        const categories = await db.Category.findAll({
            order: [
                ['categoryId', 'ASC']
            ]
        });

        if (categories.length === 0) {
            const error = new Error('Category not found.');
            error.statusCode = 404;
            throw error;
        }

        res.status(200).json({ categories });
    } catch (err) {
        next(err);
    };
};

const createCategory = async (req, res, next) => {
    const { name, isAvailable } = req.body;

    try {
        const category = await db.Category.findOne({
            where: {
                name: name
            }
        });

        if (category) {
            const error = new Error('Category already exists.');
            error.statusCode = 409;
            throw error;
        }

        const newCategory = await db.Category.create({
            name: name,
            isAvailable: !!isAvailable ? isAvailable : false,
        });

        if (!newCategory) {
            const error = new Error('Could not create category.');
            error.statusCode = 409;
            throw error;
        }

        res.status(200).json({ category: newCategory });
    } catch (error) {
        return next(error);
    }
};

```

```
57  const deleteCategory = async (req, res, next) => {
58      const categoryId = req.params.id;
59
60      if (!Number.isInteger(+categoryId)) {
61          const error = new Error('Category Id is not an integer number.');
62          error.statusCode = 400;
63          throw error;
64      }
65
66      try {
67          const categoryFound = await db.Category.findOne({
68              where: {
69                  categoryId: categoryId
70              }
71          });
72
73          if (!categoryFound) {
74              const error = new Error('Category not found.');
75              error.statusCode = 404;
76              throw error;
77          }
78
79          const categoryDestroyed = await db.Category.destroy({
80              where: {
81                  categoryId: categoryId
82              }
83          });
84
85          if (!categoryDestroyed) {
86              const error = new Error('Could not delete category');
87              error.statusCode = 409;
88              throw error;
89          }
90
91          res.status(200).json({ category: categoryFound });
92      } catch (err) {
93          next(err);
94      };
95  };
96
97  const updateCategory = async (req, res, next) => {
98      const categoryId = req.params.id;
99
100     if (!Number.isInteger(+categoryId)) {
101         const error = new Error('Category Id is not an integer number');
102         error.statusCode = 400;
103         throw error;
104     }
105
106     try {
107
108         const updatedCategory = await db.Category.update({ ...req.body }, {
109             where: {
110                 categoryId: categoryId
111             },
112             returning: true
113         });
114
115         if (!updatedCategory) {
116             const error = new Error('Could not update category');
```

```
            error.statusCode = 409;
            throw error;
        }

        res.status(200).json({ category: updatedCategory });
    } catch (err) {
        next(err);
    };
};


exports.getCategories = getCategories;
exports.createCategory = createCategory;
exports.deleteCategory = deleteCategory;
exports.updateCategory = updateCategory;
```

backend/middleware/authmiddleware.js

```
const jwt = require('jsonwebtoken');
const db = require('../util/database');
const dotenv = require('dotenv');
dotenv.config();
const SECRET = process.env.SECRET;

const authenticate = (req, res, next) => {
    const authHeader = req.get('Authorization');
    if (!authHeader) {
        const error = new Error('Authorization header is missing.');
        error.statusCode = 401;
        throw error;
    }

    const token = authHeader.split(' ')[1];
    if (!token) {
        const error = new Error('Token not found.');
        error.statusCode = 400;
        throw error;
    }

    let decodedToken = jwt.decode(token);
    if (!decodedToken) {
        const error = new Error('Token is not JWT.');
        error.statusCode = 400;
        throw error;
    }

    let verifiedToken;
    try {
        verifiedToken = jwt.verify(token, SECRET);
    } catch (err) {
        throw err;
    }
    if (!verifiedToken) {
        const error = new Error('Invalid token.');
        error.statusCode = 400;
        throw error;
    }
    req.token = verifiedToken;
    next();
}
const authorize = (roles) => {
    return (req, res, next) => {
        for (let i = 0; i <= roles.length; i++) {
            if (req.token.role === roles[i]) {
                return next();
            }
        }
        res.status(403).json({ message: 'Unauthorized user.' });
    }
}
exports.authenticate = authenticate;
exports.authorize = authorize;
```

backend/routes/authroutes.js

```
const express = require('express');

const login = require('../controllers/authcontroller');

const router = express.Router();

router.post('/login', login);

module.exports = router;
```

backend/routes/userroutes.js

```
const express = require('express');

const { authenticate, authorize } = require('../middleware/authmiddleware');
const userController = require('../controllers/usercontroller');

const router = express.Router();

router.get('/users', authenticate, authorize(['admin']), userController.getUsers);

router.post('/user', authenticate, authorize(['admin']),
userController.createUser);

router.delete('/user/:id', authenticate, authorize(['admin']),
userController.deleteUser);

router.put('/user/:id', authenticate, authorize(['admin']),
userController.updateUser);

module.exports = router;
```

backend/routes/categoryroutes.js

```
const express = require('express');
const db = require('../util/database');

const { authenticate, authorize } = require('../middleware/authmiddleware');
const categoryController = require('../controllers/categorycontroller');

const router = express.Router();

router.get('/categories', authenticate, authorize(['server', 'admin']),
categoryController.getCategories);

router.post('/category', authenticate, authorize(['admin']),
categoryController.createCategory);

router.delete('/category/:id', authenticate, authorize(['admin']),
categoryController.deleteCategory);

router.put('/category/:id', authenticate, authorize(['admin']),
categoryController.updateCategory);

module.exports = router;
```

ΠΑΡΑΡΤΗΜΑ Β: ΚΩΔΙΚΑΣ FRONTEND

web/src/app/components/ticket/ticket.component.ts

```
1  import { Component, Input, OnInit, OnDestroy, ViewChild, ElementRef } from
   '@angular/core';
2  import { Subscription } from 'rxjs';
3  import { Item } from 'src/app/models/item.model';
4  import { Order } from 'src/app/models/order.model';
5  import { Status } from 'src/app/models/status.model';
6  import { TicketMode } from 'src/app/models/ticket-mode.model';
7  import { OrderService } from 'src/app/services/order.service';
8  import { UserService } from 'src/app/services/user.service';
9
10 @Component({
11   selector: 'ticket',
12   templateUrl: './ticket.component.html',
13   styleUrls: ['./ticket.component.css']
14 })
15 export class TicketComponent implements OnInit, OnDestroy {
16   @Input() showInstructions: boolean = false;
17   @Input() mode: TicketMode;
18   @Input() tableNumber: number;
19   @Input() order: Order;
20   @Input() editMode: boolean;
21   newItem: Item;
22   items: Item[];
23   TicketMode = TicketMode;
24   Status = Status;
25   label: string;
26   server: string;
27   role: string;
28   @ViewChild('instructions') instructions: ElementRef;
29   orderSubscription: Subscription
30
31   constructor(private orderService: OrderService, private userService:
   UserService) { }
32
33   ngOnInit() {
34     if (!this.order) {
35       this.orderSubscription = this.orderService.order.subscribe({
36         next: (order) => {
37           this.order = order;
38           this.editMode = this.order.state === "NEW" || this.order.state ===
   "OPEN" || this.order.state === "MADE";
39         },
40         error: (error) => {
41           console.log(error);
42         },
43         complete: () => {
44           this.server = this.order.username;
45         }
46       });
47     }
48   }
49
50   ngOnDestroy(): void {
51     if (!this.order) {
52       this.orderSubscription.unsubscribe();
53     }
54   }
```

```
55
56    isMode(ticketMode: string): boolean {
57      this.label = this.mode;
58      return this.mode == ticketMode;
59    }
60
61    setInstructions(): void {
62      this.order.instructions = this.instructions.nativeElement.value;
63      this.orderService.order.next(this.order);
64    }
65
66    getTicketStyle(): any {
67      if (this.order == null) {
68        return {
69          'border-color': '#5b5b5b',
70          'border-width': '2px'
71        };
72      }
73
74      if (this.order.state === Status.MADE) {
75        return {
76          'border-color': 'darkgreen',
77          'border-width': '3px'
78        }
79      }
80
81      if (this.order.state === Status.CANCELLED) {
82        return {
83          'border-color': 'darkred',
84          'border-width': '3px'
85        }
86      }
87    }
88
89    getItemStyle(item: Item): any {
90      if (item.orderitems?.status === Status.OPEN) {
91        return {
92          'border-color': '#5b5b5b',
93          'border-width': '2px'
94        };
95      }
96
97      if (item.orderitems?.status === Status.MADE) {
98        return {
99          'border-color': 'darkgreen',
100         'border-width': '3px'
101       }
102     }
103
104     if (item.orderitems?.status === Status.CANCELLED) {
105       return {
106         'border-color': 'darkred',
107         'border-width': '3px'
108       };
109     }
110   }
111
112   cancelOrder(event: any) {
113     this.order.state = Status.CANCELLED;
114     this.orderService.updateOrder(this.order.orderId, this.order);
```

```
115     }
116
117     makeOrder(event: any) {
118       this.order.state = Status.MADE;
119       this.orderService.updateOrder(this.order.orderId, this.order);
120     }
121
122     onDeleteItem(removedItem: Item) {
123       const index = this.order.items.findIndex(item => {
124         return item.itemId === removedItem.itemId;
125       });
126       if (removedItem.orderitems.quantity > 1) {
127         removedItem.orderitems.quantity -= 1;
128         this.order.orderTotal -= removedItem.price;
129       } else {
130         this.order.items.splice(index, 1);
131         this.order.orderTotal -= removedItem.price;
132       }
133       this.orderService.order.next(this.order);
134     }
135
136 }
137
```

web/src/app/components/ticket/ticket.component.html

```
<div class="ticket" [ngStyle]="getTicketStyle()">
    <div class="ticket-header">
        <div>
            <p *ngIf="order?.orderId === 0">New Order</p>
            <p *ngIf="!order || order?.orderId !== 0">Order #{{order.orderId}}</p>
            <p>Server: {{ order.username }}</p>
        </div>
        <div>
            <p>{{ order.orderDate | date: 'dd/MM/yy HH:mm'}}</p>
        </div>
    </div>
    <div class="ticket-item-list">
        <ul>
            <li class="ticket-list-item" *ngFor="let item of order?.items"
[ngStyle]="getItemStyle(item)">
                <div class="ticket-item">
                    <div class="item">
                        <div><b>{{ item?.orderitems?.quantity }}</b>
{{item?.title}}</div>
                        <b>{{ (!!item?.price ? item.price *
item.orderitems.quantity / 100 : 0) | number: '1.2-2'}}€</b>
                    </div>
                    <div class="qualifiers">{{ item?.orderitems?.qualifiers }}
</div>
                </div>
                <div class="delete-icon">
                    <button *ngIf="editMode || order.state === 'NEW'" class="item-
image-button"
                        (click)="onDeleteItem(item)"><img
src="../../../assets/deletebutton.png" /></button>
                </div>
            </li>
        </ul>
    </div>
    <textarea #instructions type="text" class="ticket-instructions"
*ngIf="showInstructions" maxlength="150"
        [disabled]="!editMode && !isMode('TOTAL')" placeholder="Instructions:"
[value]="order.instructions"
        (input)="setInstructions()"></textarea>
    <div class="ticket-instructions-empty" *ngIf="!(showInstructions)"></div>
    <div class="ticket-total" *ngIf="isMode('TOTAL')">
        Total: {{ (!!order ? order.orderTotal / 100 : 0) | number: '1.2-2'}} €
    </div>
    <div class="ticket-buttons" *ngIf="isMode('BUTTONS')">
        <button class="cancel-button" (click)="cancelOrder($event)">
            <img src="../../../assets/closebutton.png" />
        </button>
        <button class="made-button" (click)="makeOrder($event)">
            <img src="../../../assets/madebutton.png" />
        </button>
    </div>
</div>
```

web/src/app/components/ticket/ticket.component.css

```css
.ticket {
    background-color: #b1b1b1;
    border-color: #5b5b5b;
    border-style: solid;
    border-width: 2px;
    width: 400px;
    height: 530px;
    border-radius: 5px;
}

.ticket-header {
    width: 100%;
    height: 20%;
    border: 10px solid #b1b1b1;
    background-color: white
    display: flex;
    justify-content: space-between;
    padding: 3%;
    font-weight: bold;
}

.ticket-item-list {
    height: 45%;
    overflow-y: auto;
    margin-top: 5px;
    margin-bottom: 5px;
    padding: 5%;
    position: relative;
    right: 5%
}

.ticket-list-item {
    display: flex;
    justify-content: space-between;
}

.item-image-button {
    border: none;
    background-color: transparent;
}

.item-image-button:hover {
    transform: scale(0.9);
}

.delete-icon {
    margin-top: 5px;
    padding: 5px;
}

.ticket-item {
    background-color: white
    border-radius: 5px;
    padding: 5px;
    width: 100%;
    margin: 10px;
```

```
}

.item {
    display: flex;
    justify-content: space-between;
}

.item-price {
    padding-left: 9rem;
}

.ticket-label {
    position: relative;
    /* left: 55%; */
    width: 95%;
    height: 15%;
    margin: 10px;
    top: 3%;
    font-size: 50px;
    text-align: center;
}


.ticket-instructions {
    width: 100%;
    height: 15%;
    border: 10px solid #b1b1b1;
    background-color: white
    resize: none;
    display: flex;
    justify-content: space-between;
    padding: 5px 10px;
    font-style: italic;
    position: relative;
}

.ticket-instructions-empty {
    width: 100%;
    height: 15%;
    border: 10px solid #b1b1b1;
    background-color: #b1b1b1;
    display: flex;
    justify-content: space-between;
    padding: 3%;
    font-style: italic;
    position: relative;
    top: 5%
}

.ticket-total {
    width: 40%;
    height: 8%;
    background-color: white
    position: relative;
    left: 55%;
    top: 7%;
    border-radius: 5px;
    padding: 10px;
    font-weight: bold;
}
```

```
117
118 .ticket-buttons {
119     height: 15%;
120     display: flex;
121     position: relative;
122     left: 20px;
123 }
124
125 .cancel-button {
126     width: 30%;
127     margin-right: 15px;
128     background-color: darkred
129     border-radius: 5px;
130 }
131
132 .made-button {
133     width: 50%;
134     background-color: darkgreen
135     border-radius: 5px;
136 }
137
138 .qualifiers {
139     font-style: italic;
140 }
```

web/src/app/pages/server-order-page/server-order-page.component.ts

```
import { Component, OnDestroy, OnInit } from '@angular/core';
import { ActivatedRoute, Router } from '@angular/router';
import { BehaviorSubject, Subscription } from 'rxjs';
import { Order } from 'src/app/models/order.model';
import { Status } from 'src/app/models/status.model';
import { TicketMode } from 'src/app/models/ticket-mode.model';
import { AuthService } from 'src/app/services/auth.service';
import { OrderService } from 'src/app/services/order.service';
import { UserService } from 'src/app/services/user.service';

@Component({
  selector: 'server-order-page',
  templateUrl: './server-order-page.component.html',
  styleUrls: ['./server-order-page.component.css']
})
export class ServerOrderPageComponent implements OnInit, OnDestroy {
  tableNumber: number;
  total: number;
  showInstructions: BehaviorSubject<boolean> = new BehaviorSubject(false);
  instructionsLabel: string = 'Add Instructions';
  editMode: boolean;

  order: Order;
  ticketMode: TicketMode = TicketMode.TOTAL;
  paramsSubscription: Subscription;
  orderSubscription: Subscription;

  constructor(private route: ActivatedRoute, private router: Router, private userService: UserService, protected orderService: OrderService) { }

  ngOnInit() {
    this.paramsSubscription = this.route.queryParams.subscribe(params => {
      this.tableNumber = params['table'];
    });
    this.orderService.order.subscribe(order => {
      this.order = order;
    });
    this.orderSubscription = this.orderService.getOrderByTableNum(this.tableNumber).subscribe({
      next: (result) => {
        if (result.order.orderId === 0) {
          result.order.serverId = +(localStorage.getItem('userId') as string)
          result.order.username = localStorage.getItem('username') as string;
        }
        this.order = result.order;
        this.editMode = result.order?.state === "OPEN" || result.order?.state === "MADE";
      },
      error: (error) => {
        console.log(error);
      },
      complete: () => {
        this.orderService.order.next(this.order);
      }
    });
  }
```

```
54
55    ngOnDestroy(): void {
56      this.paramsSubscription.unsubscribe();
57      this.orderSubscription.unsubscribe();
58    }
59
60    instructionsHandler() {
61      this.instructionsLabel = this.showInstructions.getValue() ? 'Add Instructions'
    : 'Clear Instructions';
62      this.showInstructions.next(!this.showInstructions.getValue());
63    }
64
65    onCancel() {
66      this.router.navigate(['/server/map']);
67      this.order.state = Status.CANCELLED;
68      this.orderService.updateOrder(this.orderService.order.getValue().orderId,
    this.order);
69    }
70
71    onClose() {
72      this.router.navigate(['/server/map']);
73      this.order.state = Status.CLOSED;
74      this.orderService.updateOrder(this.orderService.order.getValue().orderId,
    this.order);
75    }
76
77    onUpdate() {
78      this.router.navigate(['/server/map']);
79      this.order.state = Status.OPEN;
80      this.orderService.updateOrder(this.orderService.order.getValue().orderId,
    this.order);
81    }
82
83    onPlace() {
84      this.router.navigate(['/server/map']);
85      this.order.state = Status.OPEN;
86      this.orderService.createOrder(this.order);
87    }
88  }
89
```

web/src/app/pages/server-order-page/server-order-page.component.html

```
<div class="order-menu-page">
    <div class="order-title">
        <p *ngIf="this.orderService.order.getValue().orderId !== 0">Order #{{ this.orderService.order.getValue().orderId
            }} Table Number: {{ tableNumber }}</p>
        <p *ngIf="this.orderService.order.getValue().orderId === 0">New Order Table Number: {{ tableNumber }}</p>
    </div>
    <div class="order-menu-components">
        <div class="order-menu-wrapper">
            <order-menu>
            </order-menu>
        </div>
        <div class="order-ticket-actions">
            <ticket [order]="order" [editMode]="editMode" [mode]="ticketMode"
                [showInstructions]="showInstructions.getValue()" [tableNumber]="tableNumber">
            </ticket>
            <div class="buttons-group">
                <div class="buttons-group-row">
                    <button class="order-action-button" (click)="instructionsHandler()">{{ instructionsLabel }}</button>
                    <button class="order-action-button" (click)="onCancel()">Cancel order</button>
                </div>
                <div class="buttons-group-row">
                    <button class="order-action-button" (click)="onClose()">Close order</button>

                    <button *ngIf="!editMode" class="order-action-button" (click)="onPlace()">Place order</button>
                    <button *ngIf="editMode" class="order-action-button" (click)="onUpdate()">Update order</button>
                </div>
            </div>
        </div>
    </div>
</div>
<icon-button class="home-button" [img]="'../../../assets/homebutton.png'" routerLink="/server/map">
</icon-button>
```

web/src/app/pages/server-order-page/server-order-page.component.css

```css
.order-title {
    font-weight: bold;
    font-size: x-large;
}

.home-button {
    border: none;
    background-color: #BEBEBE;
}

.order-menu-components {
    display: flex;
    margin-top: 3%;
}

.order-menu-wrapper {
    margin-right: 20%;
}

.buttons-group {
    margin-top: 10%;
}

.buttons-group-row {
    display: flex;
    justify-content: space-between;
    margin-bottom: 5%;
}

.order-action-button {
    width: 45%;
    height: 20%;
    padding: 5%;
    background-color: white
    border-radius: 5px;
    font-weight: bolder;
}
```

web/src/app/services/order.service.ts

```
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { BehaviorSubject } from 'rxjs';
import { TableService } from './table.service';
import { io } from 'socket.io-client';
import { Order } from '../models/order.model';
import { Status } from '../models/status.model';


@Injectable({
    providedIn: 'root'
})
export class OrderService {
    fetchedOrder: BehaviorSubject<Order> = new BehaviorSubject<Order>({} as Order);
    order: BehaviorSubject<Order> = new BehaviorSubject<Order>({} as Order);
    orders: BehaviorSubject<Order[]> = new BehaviorSubject<Order[]>([]);

    socket = io('localhost:8080');

    getOrderFromSocket() {
        this.socket.on('order', response => {
            this.fetchedOrder.next(response.order);
        });

        return this.fetchedOrder.asObservable();
    }

    constructor(private http: HttpClient, private tableService: TableService) { }

    getOrderByTableNum(tableId: number) {
        return this.http.get<{ order: Order }>
('http://localhost:8080/api/order/table/' + tableId, {
            "headers": {
                'Authorization': 'Bearer ' + localStorage.getItem('token')
            }
        });
    }

    getOrders() {
        this.http.get<{ fetchedOrders: Order[] }>
('http://localhost:8080/api/orders', {
            'headers': {
                'Authorization': 'Bearer ' + localStorage.getItem('token')
            }
        }).subscribe({
            next: (value) => {
                this.orders.next(value.fetchedOrders);
            },
            error: (error) => {
                console.log(error);
            },
            complete: () => { }
        })
    }

    createOrder(order: Order) {
        console.log(order);
```

```
56          this.http.post<Order>('http://localhost:8080/api/order',
this.order.getValue(), {
57              "headers": {
58                  'Authorization': 'Bearer ' + localStorage.getItem('token')
59              }
60          }).subscribe({
61              next: (value) => {
62                  this.order.next(value);
63              },
64              error: (error) => {
65                  console.log(error);
66              },
67              complete: () => { }
68          })
69      }
70
71      updateOrder(orderId: number, order: Order) {
72          this.http.put<{ order: Order }>('http://localhost:8080/api/order/' +
orderId, order, {
73              "headers": {
74                  'Authorization': 'Bearer ' + localStorage.getItem('token')
75              }
76          }).subscribe({
77              next: (result) => {
78                  this.order.next(result.order);
79              },
80              error: (error) => {
81                  console.log(error);
82              },
83              complete: () => {
84                  if (this.order.getValue()?.state === Status.CLOSED ||
this.order.getValue()?.state === Status.CANCELLED) {
85                      this.tableService.closeOrder(this.order.getValue().tableNum);
86                  }
87              }
88          })
89      }
90
91  }
92
```

web/src/app/services/auth-guard.service.ts

```ts
import { Injectable } from "@angular/core";
import { ActivatedRouteSnapshot, CanActivate, Router, RouterStateSnapshot, UrlTree } from "@angular/router";
import { Observable } from "rxjs";
import { AuthService } from "./auth.service";

@Injectable({
    providedIn: 'root'
})
export class AuthGuardService implements CanActivate {

    constructor(private authService: AuthService, private router: Router) { }

    canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): boolean | UrlTree | Observable<boolean | UrlTree> | Promise<boolean | UrlTree> {
        if (localStorage.getItem('token') !== null) {
            return true;
        } else {
            this.router.navigate(['/login']);

            return false;
        }
    }
}
```

web/src/app/services/auth.service.ts

```
import { Injectable, OnDestroy, OnInit } from '@angular/core';
import { Router } from '@angular/router';
import jwt_decode from 'jwt-decode';
import { BehaviorSubject, Subject, Subscription } from 'rxjs';

interface JwtPayload {
  username: string,
  role: string,
  userId: number
}

@Injectable({
  providedIn: 'root'
})
export class AuthService {

  constructor(private router: Router) { }

  async login(username: string, password: string): Promise<string> {
    let errorMessage: string = "";
    try {
      const response: Response = await fetch('http://localhost:8080/api/login', {
        method: 'POST',
        headers: {
          'Accept': 'application/json',
          'Content-Type': 'application/json'
        },
        body: JSON.stringify({
          username: username,
          password: password
        })
      });

      if (response.status === 404 || response.status === 400 || response.status === 500) {
        await response.json().then(body => { errorMessage = body?.message });
      } else if (response.status === 200) {
        await response.json().then(body => {
          const jwt = jwt_decode<JwtPayload>(body.token);
          localStorage.setItem('username', jwt.username);
          localStorage.setItem('userId', body.userId);
          localStorage.setItem('role', jwt.role);

          if (!jwt.role) {
            return;
          }

          localStorage.setItem('token', body.token);

          if (jwt.role === 'admin') {
            this.router.navigate(['/admin']);
          }

          if (jwt.role === 'server') {
            this.router.navigate(['/server/map']);
          }
```

```
56
57            if (jwt.role === 'cooker') {
58              this.router.navigate(['/cooker/orders']);
59            }
60          });
61        }
62      } catch (err) {
63        console.log(err);
64      }
65
66      return errorMessage;
67    }
68
69    logout() {
70      localStorage.removeItem('token');
71      this.router.navigate(['/login']);
72    }
73  }
74
```

web/src/app/models/order.model.ts

```
1  import { Item } from "./item.model";
2  import { Status } from "./status.model";
3
4  export interface Order {
5      orderId: number;
6      serverId: number,
7      tableNum: number,
8      orderDate: string,
9      orderTotal: number,
10     state: Status,
11     instructions: string,
12     items: Item[],
13     username: string
14 }
```

ΠΑΡΑΡΤΗΜΑ Γ: ΒΑΣΗ ΔΕΔΟΜΕΝΩΝ

backend/util/database.js

```
const { Sequelize } = require('sequelize');
const dotenv = require('dotenv');

dotenv.config();
const DB_HOST = process.env.DB_HOST;

const sequelize = new Sequelize(DB_HOST, {
    pool: {
        max: 4,
        min: 0,
        idle: 10000
    },
});

const Category = sequelize.define('categories', {
    categoryId: {
        type: Sequelize.INTEGER,
        allowNull: false,
        autoIncrement: true,
        primaryKey: true,
        unique: true
    },
    name: {
        type: Sequelize.STRING,
        allowNull: false
    },
    qualifierType: {
        type: Sequelize.STRING,
        allowNull: true
    },
    isAvailable: {
        type: Sequelize.BOOLEAN,
        allowNull: false
    }
}, { "timestamps": false });

const Order = sequelize.define('orders', {
    orderId: {
        type: Sequelize.INTEGER,
        autoIncrement: true,
        allowNull: false,
        primaryKey: true,
        unique: true
    },
    serverId: {
        type: Sequelize.INTEGER,
        allowNull: false
    },
    tableNum: {
        type: Sequelize.INTEGER,
        allowNull: false
    },
    orderDate: {
        type: Sequelize.DATE,
        allowNull: false
    },
```

```
    orderTotal: {
        type: Sequelize.INTEGER,
        allowNull: true
    },
    state: {
        type: Sequelize.STRING,
        allowNull: false
    },
    instructions: {
        type: Sequelize.STRING,
        allowNull: true
    }
}, { "timestamps": false });

const Item = sequelize.define('items', {
    itemId: {
        type: Sequelize.INTEGER,
        allowNull: false,
        autoIncrement: true,
        primaryKey: true,
        unique: true
    },
    title: {
        type: Sequelize.STRING,
        allowNull: false
    },
    price: {
        type: Sequelize.INTEGER,
        allowNull: false
    },
    isAvailable: {
        type: Sequelize.BOOLEAN,
        allowNull: false,
    },
    ingredients: {
        type: Sequelize.STRING,
        allowNull: false
    }
}, { "timestamps": false });

const OrderItem = sequelize.define('orderitems', {
    status: {
        type: Sequelize.STRING,
        allowNull: false
    },
    quantity: {
        type: Sequelize.INTEGER,
        allowNull: false
    },
    qualifiers: {
        type: Sequelize.STRING,
        allowNull: true
    }
}, { "timestamps": false });

const Reservation = sequelize.define('reservations', {
    reservationId: {
        type: Sequelize.INTEGER,
        autoIncrement: true,
        allowNull: false,
```

```
        primaryKey: true,
        unique: true
    },
    reservationName: {
        type: Sequelize.STRING,
        allowNull: false
    },
    reservationDate: {
        type: Sequelize.DATE,
        allowNull: false
    }
}, { "timestamps": false });

const Table = sequelize.define('tables', {
    tableNum: {
        type: Sequelize.INTEGER,
        allowNull: false,
        primaryKey: true,
        unique: true
    },
    seats: {
        type: Sequelize.INTEGER,
        allowNull: true
    },
    locationX: {
        type: Sequelize.INTEGER,
        allowNull: false
    },
    locationY: {
        type: Sequelize.INTEGER,
        allowNull: false
    }
}, { "timestamps": false });

const User = sequelize.define('users', {
    userId: {
        type: Sequelize.INTEGER,
        autoIncrement: true,
        allowNull: false,
        primaryKey: true,
        unique: true
    },
    username: {
        type: Sequelize.STRING,
        allowNull: false
    },
    password: {
        type: Sequelize.STRING,
        allowNull: false
    },
    firstName: {
        type: Sequelize.STRING,
        allowNull: false
    },
    lastName: {
        type: Sequelize.STRING,
        allowNull: false
    },
    hireDate: {
        type: Sequelize.DATE,
```

```
        allowNull: false
    },
    role: {
        type: Sequelize.STRING,
        allowNull: false
    }
}, { "timestamps": false });

//Describing associations with foreign keys
User.hasOne(Order, {
    foreignKey: "serverId"
});

Order.belongsTo(Table, {
    foreignKey: "tableNum"
});

Reservation.belongsTo(Table, {
    foreignKey: "tableNum"
});

Order.belongsToMany(Item, {
    through: OrderItem,
    foreignKey: 'orderId'
});
Item.belongsToMany(Order, {
    through: OrderItem,
    foreignKey: 'itemId'
});

Item.belongsTo(Category, {
    foreignKey: "categoryId"
});

const states = {
    new: "NEW",
    open: "OPEN",
    cancelled: "CANCELLED",
    made: "MADE",
    closed: "CLOSED"
};

const roles = {
    admin: "admin",
    server: "server",
    cooker: "cooker"
};

const qualifierTypes = {
    coffee: "coffee",
    drink: "drink"
};


exports.Category = Category;
exports.Order = Order;
exports.Item = Item;
exports.OrderItem = OrderItem;
exports.Table = Table;
exports.User = User;
```

```
237 exports.Reservation = Reservation;
238
239 exports.states = states;
240 exports.roles = roles;
241
242 exports.sequelize = sequelize;
```