Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής

Πρόγραμμα Μεταπτυχιακών Σπουδών

«Πληροφορική»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Web Σύστημα Παραγγελιοληψίας σε Πραγματικό Χρόνο για Συσκευές Android με την χρήση .Net και React.js** **Real Time Web Ordering System for Android Devices using .Net and React.js**
Ονοματεπώνυμο Φοιτητή	**Μαρία Σπανού**
Πατρώνυμο	**Γεράσιμος**
Αριθμός Μητρώου	**ΜΠΠΛ 17050**
Επιβλέπων	**Ευθύμιος Αλέπης, Αναπληρωτής Καθηγητής**

Ημερομηνία Παράδοσης **Ιανουάριος 2021**

Τριμελής Εξεταστική Επιτροπή

Ευθύμιος Αλέπης, Αναπληρωτής καθηγητής	Μαρία Βίρβου, Καθηγήτρια	Κωνσταντίνος Πατσάκης, Αναπληρωτής καθηγητής

ΠΕΡΙΕΧΟΜΕΝΑ

ΠΕΡΙΛΗΨΗ

ΕΛΛΗΝΙΚΑ

Στόχος της διπλωματικής είναι η δημιουργία ενός συστήματος - μιας εφαρμογής που να διευκολύνει ανθρώπους με κινητικές δυσκολίες στην καθημερινή τους ζωή και πιο συγκεκριμένα κατά την παραγγελία από καταστήματα (για φαΐ, καφέ). Η εργασία αποτελείται από δύο μέρη, τη μεριά του καταναλωτή (android εφαρμογή) και το σύστημα από τη μεριά του καταστήματος. Ο καταναλωτής μέσω της android εφαρμογής (γραμμένη σε java) περνάει τα προσωπικά του στοιχεία (όπως είναι η φυσική διεύθυνση, η διεύθυνση ηλεκτρονικού ταχυδρομείου κοκ), επιλέγει από τα κοντινά σε αυτόν καταστήματα τα προϊόντα που επιθυμεί να παραγγείλει και ενημερώνεται για την εξέλιξη της παραγγελίας του. Ο διαχειριστής του συστήματος του καταστήματος, που είναι διαθέσιμο μέσω ενός web browser (γραμμένο σε javascript και react.js), προσθέτει και ενημερώνει τα προϊόντα του, δέχεται παραγγελίες και ενημερώνει τον κάθε πελάτη για την κατάσταση της παραγγελίας του. Με αυτό τον τρόπο, ο πελάτης έχει τον χρόνο να προετοιμαστεί για την παραλαβή της παραγγελίας.

ΑΓΓΛΙΚΑ

The aim of this thesis is to create a system - an application that facilitates people with mobility difficulties in their daily lives and more specifically when ordering from stores (ie food, coffee). The project consists of two parts, the consumer side (android application) and the system on the store side. The consumer through the android application (written in java) inserts his personal data (such as physical address, e-mail address etc.), selects from the nearby stores the products he wishes to order and is informed about the progress of the order. The administrator of the store, which is available through a web browser (written in javascript and react.js), adds and updates its products, accepts orders and informs each customer about the status of his order. This way, the customer has time to prepare for the receipt of the order.

ΕΙΣΑΓΩΓΗ

Το θέμα της διπλωματικής εργασίας είναι ένα σύστημα - μια εφαρμογή παραγγελιοληψίας στο χώρο της εστίασης (εστιατόρια και καφέ) με στόχο να εξυπηρετήσει ανθρώπους που έχουν κινητικές δυσκολίες.

Η εργασία αποτελείται από δύο μέρη, την android εφαρμογή από τη μεριά του καταναλωτή και το σύστημα από τη μεριά του καταστήματος. Όσον αφορά στο πρώτο κομμάτι, τη μεριά του καταναλωτή, είχε εκπονηθεί εργασία στα πλαίσια του μαθήματος της Ιατρικής Πληροφορικής. Πρόκειται για μια εφαρμογή android στην οποία ο χρήστης περνάει τα στοιχεία του, μπορεί να παραγγείλει τα προϊόντα που επιθυμεί από τα κοντινά του καταστήματα εστίασης. Ως διπλωματική εργασία έχει υλοποιηθεί το σύστημα για το κατάστημα (μέσω ενός web browser), καθώς επίσης προστέθηκαν κάποιες περαιτέρω λειτουργίες στην android εφαρμογή. Ο διαχειριστής του συστήματος του καταστήματος δέχεται παραγγελίες και ενημερώνει τον κάθε πελάτη για την εξέλιξη της παραγγελίας του.

Για την εκπόνηση της εργασίας έγινε έρευνα κατά την οποία διαπιστώθηκε ότι το ποσοστό ατόμων με αναπηρία αφορά στο 15%[1] του παγκόσμιου πληθυσμού. Ποσοστό που μεταφράζεται περίπου σε 1 δισεκατομμύριο άτομα. Κατά την αρχική έρευνα δεν είχε εντοπιστεί εφαρμογή που να έχει υλοποιηθεί με σκοπό την εξυπηρέτηση ανθρώπων με μειωμένες κινητικές ικανότητες, όσον αφορά στον τομέα της παραγγελίας στο χώρο της εστίασης.

Ένας άνθρωπος με μειωμένες κινητικές ικανότητες (όπως για παράδειγμα κάποιου είδους σκλήρυνσης κατά πλάκας) διευκολύνεται εάν είναι μεγαλύτερα τα μεγέθη των actions στο κινητό (ή και στο tablet) ώστε να μη δυσκολεύεται να επιλέξει κάποια από τις επιλογές (buttons) που εμφανίζονται ή να μην επιλέγει λάθος και μετά να χρειάζεται να κάνει περισσότερες κινήσεις. Επιπλέον, βοηθάει η χρήση φωνητικών εντολών. Για παράδειγμα το notification που λαμβάνει για την εξέλιξη της παραγγελίας αν είναι φωνητική εντολή τον διευκολύνει από το να χρειάζεται να ανοίξει πάλι το κινητό του και να επιλέξει το μήνυμα να το διαβάσει.

Ένα από τα πιο σημαντικά θέματα είναι οτι ο πελάτης χρειάζεται περισσότερο χρόνο, όπως για παράδειγμα χρόνο από την στιγμή που θα χτυπήσει το κουδούνι μέχρι να φτάσει στην πόρτα να ανοίξει. Για να διευκολυνθεί ο χρήστης-πελάτης σε αυτό το τομέα, ο χρήστης του συστήματος ενημερώνει τον πελάτη για την εξέλιξη της παραγγελίας. Δηλαδή, μόλις λάβει την παραγγελία ο χρήστης του συστήματος ενημερώνει για τον χρόνο που υπολογίζει ότι χρειάζεται για τη διεκπεραίωση της, καθώς επίσης ενημερώνει εκ νέου μόλις γίνει η αποστολή από το κατάστημα, ώστε ο πελάτης να προετοιμαστεί.

Η εργασία αποτελείται από την παρουσίαση της λειτουργία της android εφαρμογής και του συστήματος μέσω ενός εγχειριδίου χρήστη (user manual). Επιπλέον, από την αρχιτεκτονική του συστήματος, δηλαδή εργαλεία που χρησιμοποιήθηκαν, οι γλώσσες που είναι γραμμένη, οι βάσεις δεδομένων, διαγράμματα UML, σχολιασμός του κώδικα. Τέλος, υπάρχουν κάποιες πιθανές μελλοντικές επεκτάσεις που θα μπορούσαν να πραγματοποιηθούν καθώς και τα συμπεράσματα.

[1] Σύμφωνα με https://www.worldbank.org/en/topic/disability

ΑΝΑΣΚΟΠΗΣΗ ΠΕΔΙΟΥ

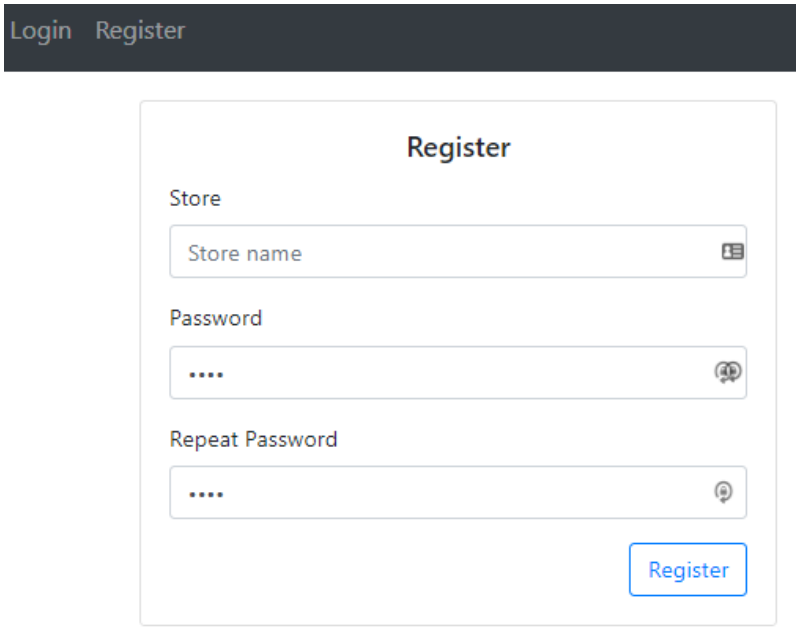
Εφαρμογές αντίστοιχες, στην Ελλάδα ή και στο εξωτερικό, εώς και αρχές του 2020 δεν έχουν εντοπιστεί. Υπάρχουν εφαρμογές παραγγελιοληψίας στον χώρο της εστίασης (φαγητού και καφέ), αλλά δεν έχουν υλοποιηθεί με στόχο την εξυπηρέτηση ανθρώπων με ειδικές ανάγκες. Έχουν εντοπιστεί εφαρμογές που στοχεύουν σε ανθρώπους με ειδικές ανάγκες (συγκεκριμένα με κινητικές δυσκολίες), αλλά όχι στο τομέα της εστίασης. Έχουν υλοποιηθεί με στόχο κυρίως την προσβασιμότητα σε μέρη, παραδείγματος χάρη είναι καταγεγραμμένα ποια μέρη έχουν ειδική ράμπα. Τέτοιες εφαρμογές που χρησιμοποιούνται στο εξωτερικό είναι η ***wheelmap*** και η **wheelMate**. Μια ελληνική εφαρμογή είναι η **SYROS AccessPoint** η οποία παρουσιάζει όλες τις καταγεγραμμένες θέσεις στάθμευσης για ΑμεΑ, όλες τις ράμπες της Ερμούπολης και των παραθαλάσσιων οικισμών και όλες τις προσβάσιμες παραλίες του νησιού. Ακόμη, υπάρχουν εφαρμογές που στοχεύουν να βοηθήσουν ανθρώπους που πέφτουν όπως είναι η Fall Detection η οποία εντοπίζει κάποια πιθανή πτώση και στέλνει μήνυμα στις επαφές που έχει δηλώσει ο χρήστης ώστε να λάβει την απαραίτητη βοήθεια. Επιπλέον, έχουν εντοπιστεί εργαλεία για πιο εύκολη διαχείριση του κινητού που στοχεύουν και σε ανθρώπους με περιορισμένες επιδεξιότητες. Κάποιες από αυτές τις εφαρμογές - εργαλεία είναι το **Assistive Touch for Android**. Μέσω ενός πάνελ που εμφανίζεται στην οθόνη του (android) κινητού χρήστη μέσω του οποίου μπορεί να χρησιμοποιήσει άμεσα και γρήγορα κάποιες από τις εφαρμογές του (όσες έχει επιλέξει). Τέλος, η SwiftKey είναι πληκτρολόγιο το οποίο προσαρμόζεται στις συνήθειες του χρήστη, δηλαδή μαθαίνει τις προτιμήσεις του χρήστη, προτείνει και διορθώνει τα λάθη κατά την πληκτρολόγηση.

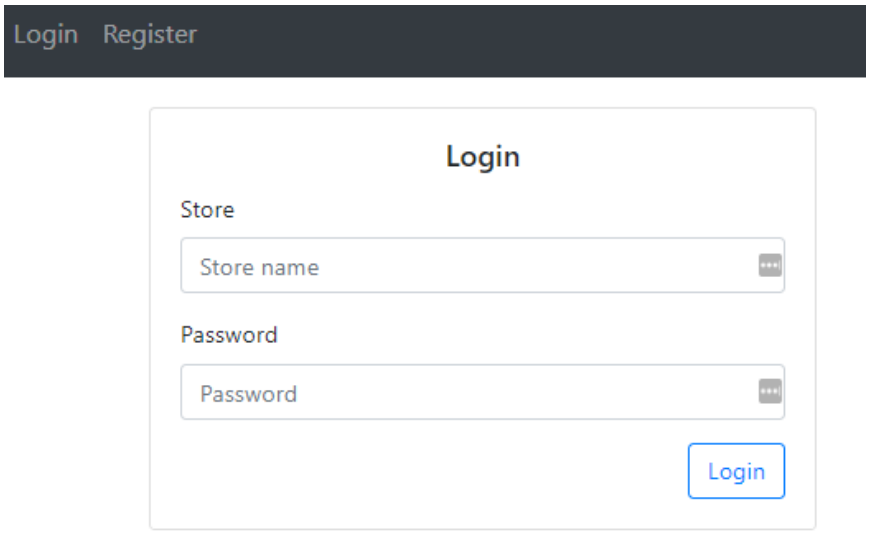
ΠΑΡΟΥΣΙΑΣΗ ΛΕΙΤΟΥΡΓΙΑΣ

Βασικές πληροφορίες καταστήματος

Εγγραφή/Σύνδεση Χρήστη (Register/Login)

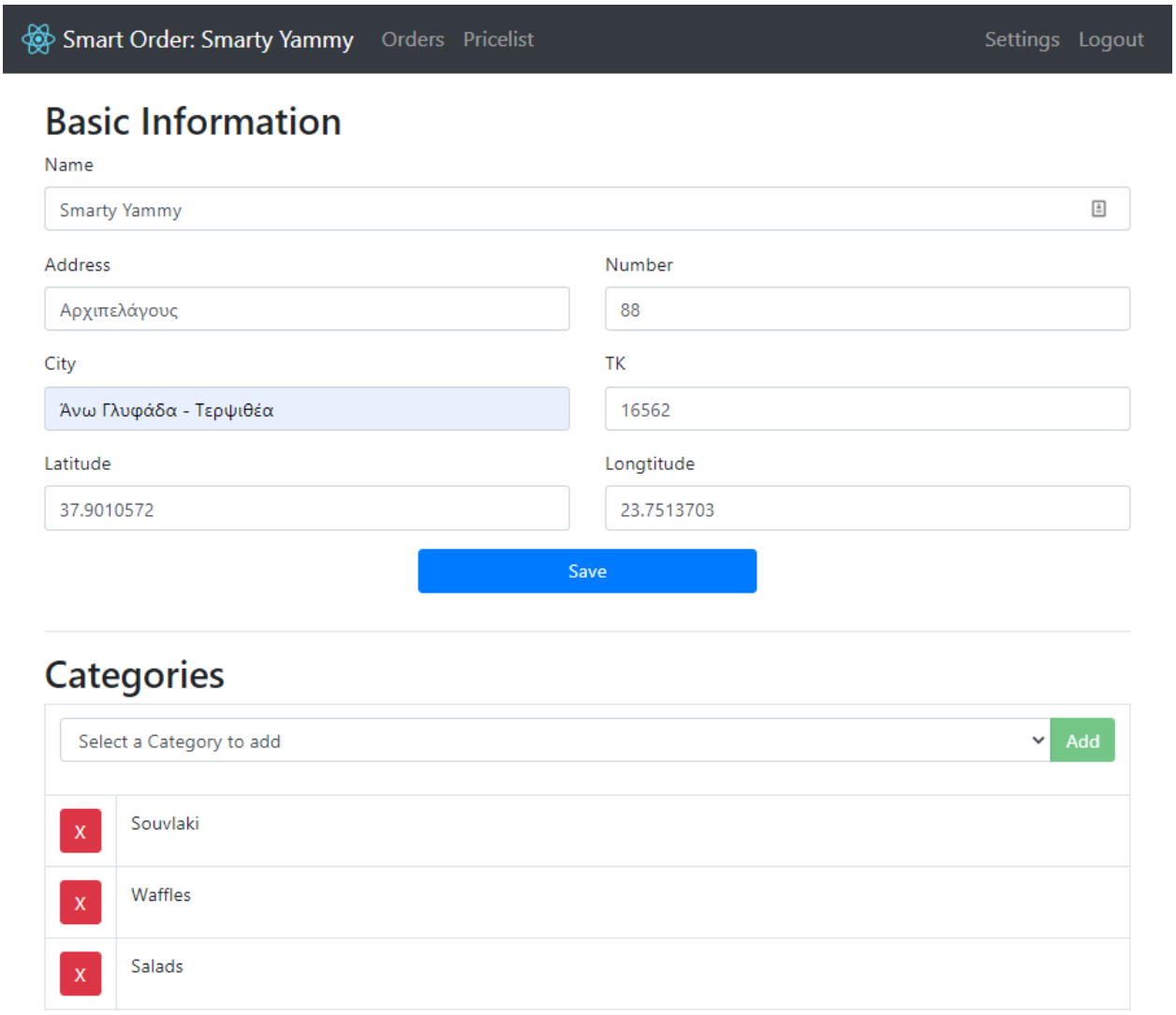
Ο χρήστης πληκτρολογεί τα απαραίτητα στοιχεία για νέα εγγραφή, ή αν έχει ήδη λογαριασμό για να συνδεθεί κι επιλέγει το πλήκτρο *Register* ή *Login* αντίστοιχα.





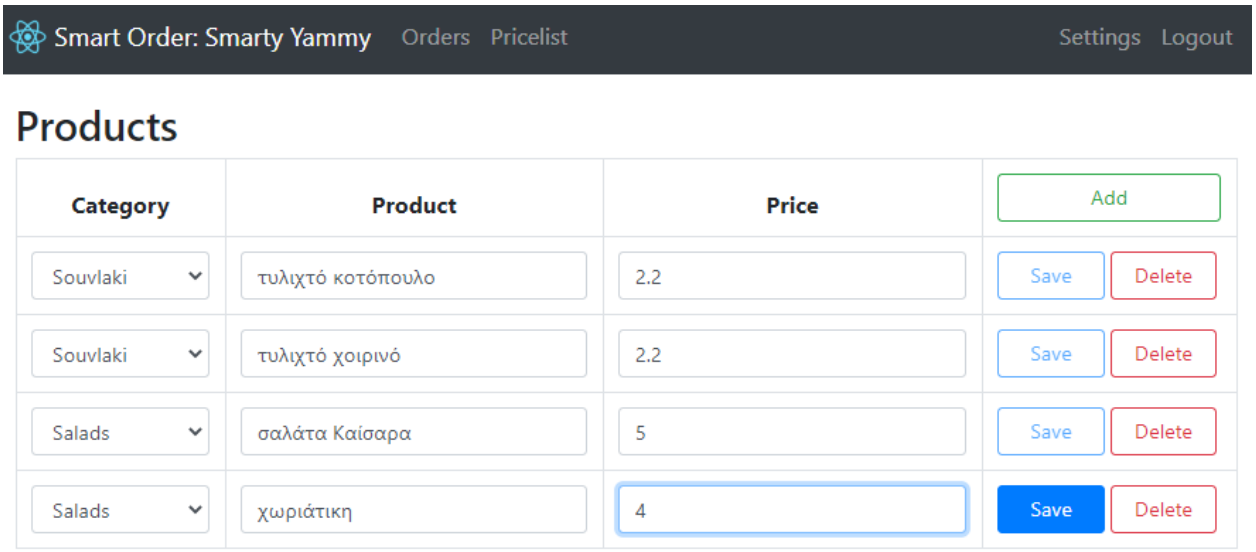
Ρυθμίσεις (Settings)

Από το Menu επιλέγει το *Settings* (τις ρυθμίσεις), συμπληρώνει τα στοιχεία του καταστήματος (διεύθυνση, περιοχή) ώστε να εμφανίζεται το κατάστημα σε όσους πελάτες βρίσκονται σε κοντινή απόσταση (εντός 10km) και επιλέγει το *Save* για να αποθηκευτούν τα στοιχεία. Ύστερα επιλέγει τις κατηγορίες που διαθέτει το κατάστημα, προσθέτει κάθε κατηγορία επιλέγοντας το πλήκτρο *Add*. Αν έχει κάνει κάποιο λάθος, με το πλήκτρο X μπορεί να διαγράψει την κατηγορία.



Τιμοκατάλογος (Price list)

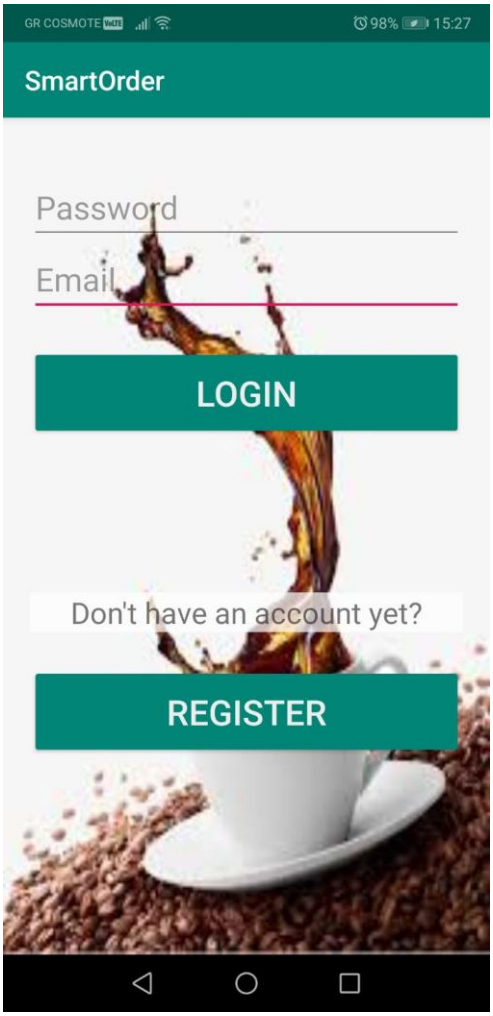
Από το Menu επιλέγει το *Pricelist*. Για κάθε κατηγορία που διαθέτει το κατάστημα, προσθέτει όσα προϊόντα επιθυμεί (επιλέγοντας το πλήκτρο *Add*) πληκτρολογεί την αντίστοιχη τιμή και επιλέγει το *Save* για να αποθηκεύσει το κάθε προϊόν και το *Delete* αν θέλει να διαγράψει κάποιο προϊόν.



Βασικές πληροφορίες κινητού

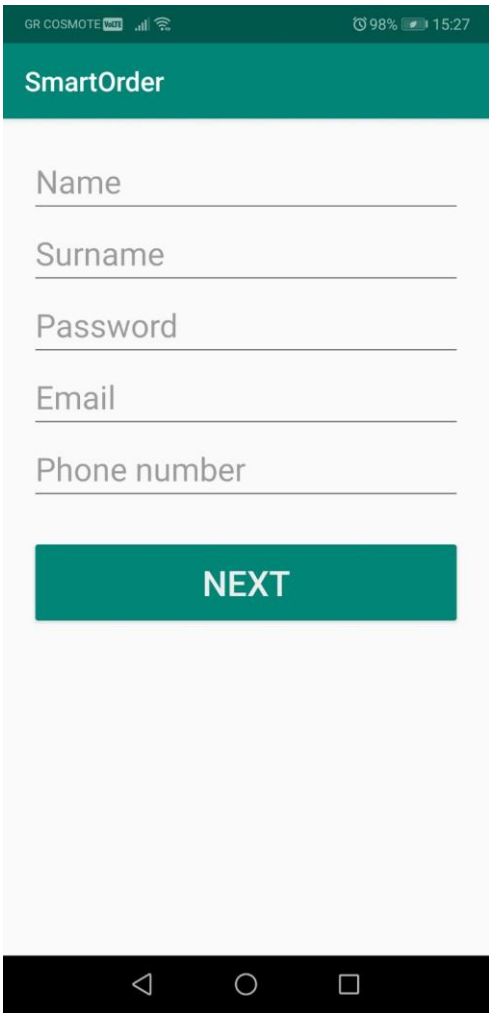
Σύνδεση (Login)

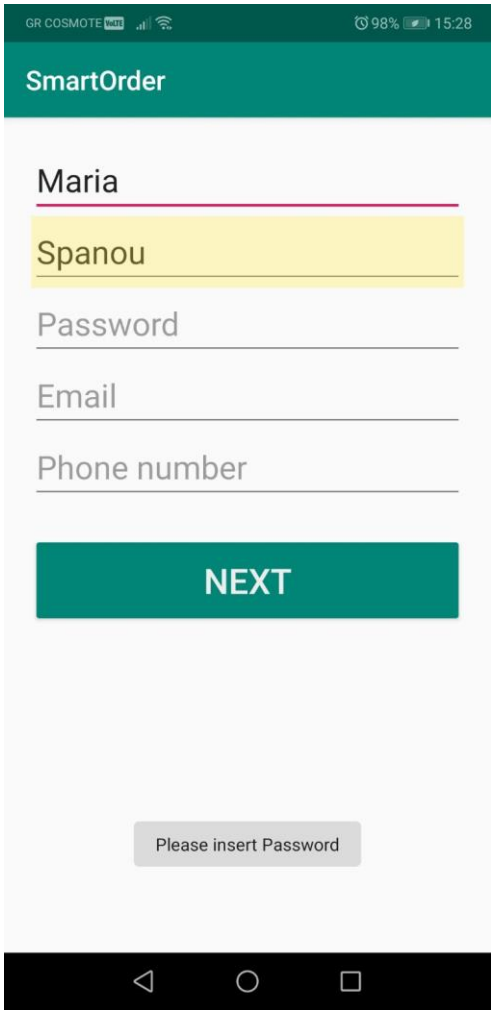
Μόλις ανοίγει για πρώτη φορά η εφαρμογή ***SmartOrder*** εμφανίζεται η επιλογή να κάνει ο χρήστης νέα εγγραφή, ή αν έχει ήδη λογαριασμό να βάλει τα στοιχεία του και να συνδεθεί.



Εγγραφή νέου χρήστη (Register)

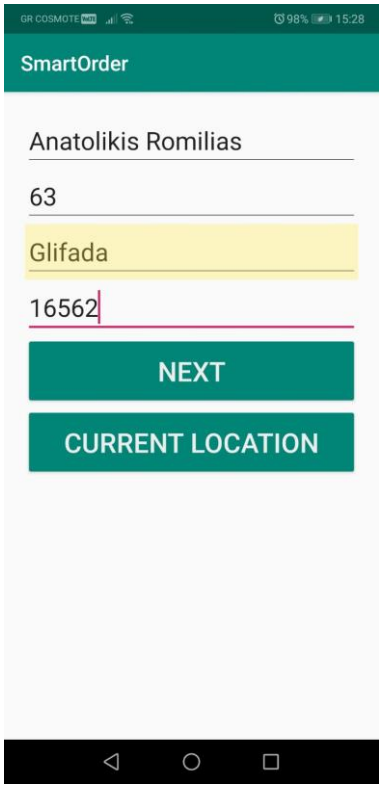
Ο χρήστης πληκτρολογεί τα στοιχεία του και πατάει το πλήκτρο NEXT. Αν δε συμπληρώσει κάποιο πεδίο, εμφανίζεται μήνυμα που τον ενημερώνει ότι πρέπει να συμπληρώσει όλα τα στοιχεία του.

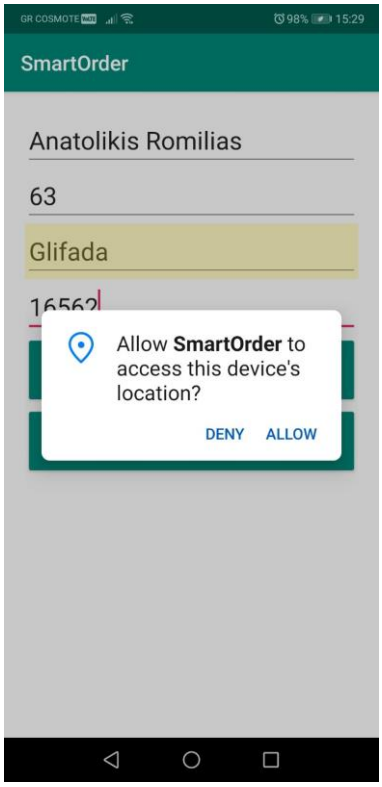


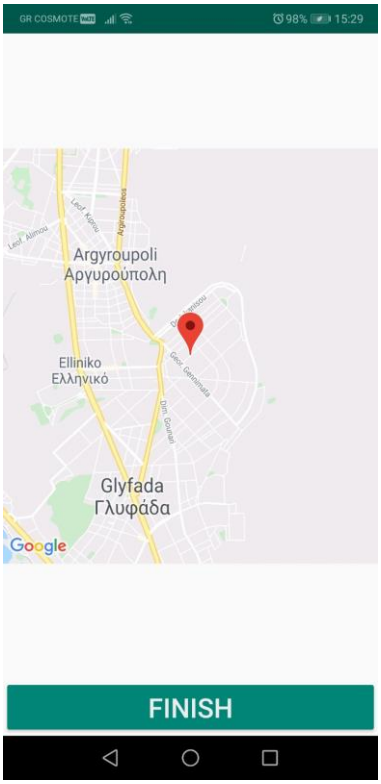


Διεύθυνση

Ο χρήστης μπορεί να δηλώσει τη διεύθυνση του είτε συμπληρώνοντας τα πεδία και να πατήσει το NEXT, είτε να πατήσει το CURRENT LOCATION οπου τότε εμφανίζεται παράθυρο που του ζητάει να επιτρέψει στην εφαρμογή να έχει πρόσβαση στην τοποθεσία του και συμπληρώνονται αυτόματα τα στοιχεία της διεύθυνσης.



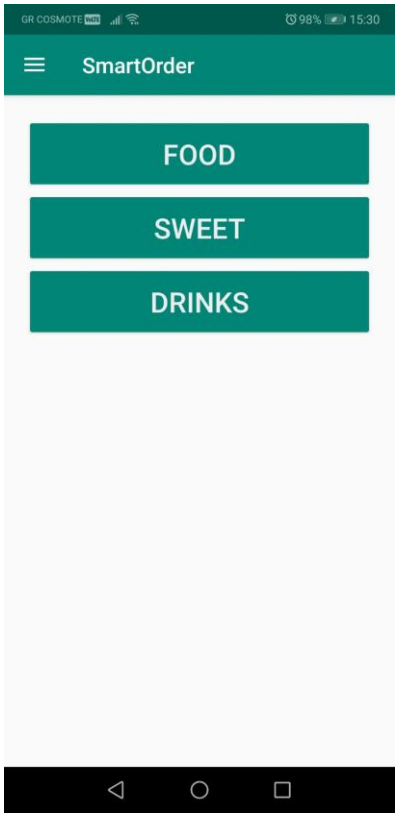


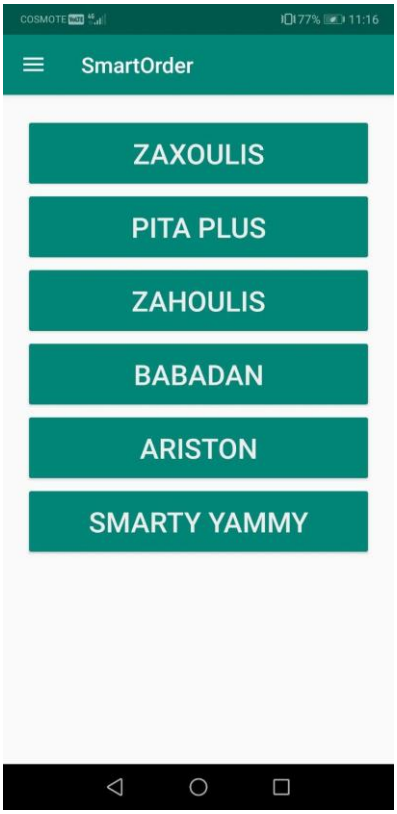


Flow Παραγγελίας

Κατηγορίες

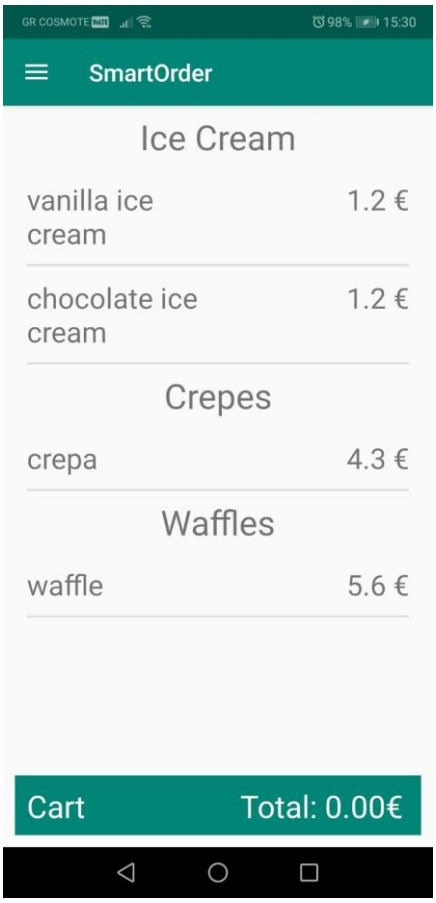
Αφού συνδεθεί ο χρήστης, του εμφανίζονται οι κατηγορίες. (Αριστερή εικόνα). Εφόσον επιλέξει κατηγορία (πχ food) εμφανίζονται οι υποκατηγορίες. (Μεσαία εικόνα). Κατόπιν εμφανίζονται τα καταστήματα που μπορεί να παραγγείλει (καταστήματα που βρίσκονται σε απόσταση μικρότερη των 10Km από τη διεύθυνση που έχει δηλώσει). (Δεξιά εικόνα)

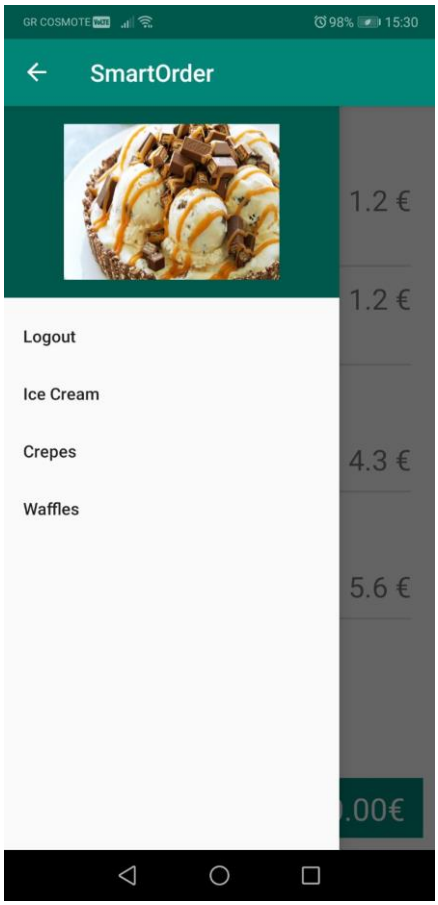




Μενού

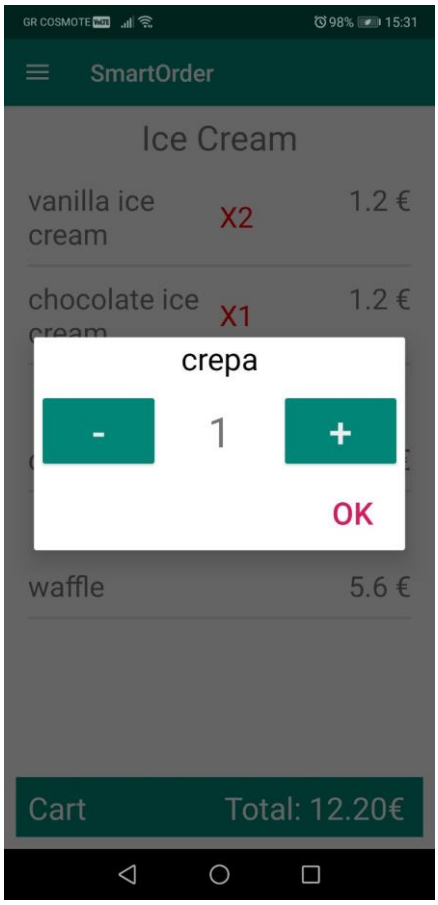
Αφού επιλέξει κατάστημα, εμφανίζεται το μενού. (Αριστερή εικόνα). Αν υπάρχουν πολλές κατηγορίες, για να διευκολυνθεί ο χρήστης (και να μην χρειάζεται να διαβάσει όλα τα προϊόντα), μπορεί να επιλέξει το πλήκτρο με τις 3 γραμμές πάνω αριστερά ώστε να επιλέξει πιο εύκολα κατηγορία που θέλει. (Δεξιά εικόνα). Γίνεται ανακατεύθυνση στην επιλεγμένη κατηγορία.

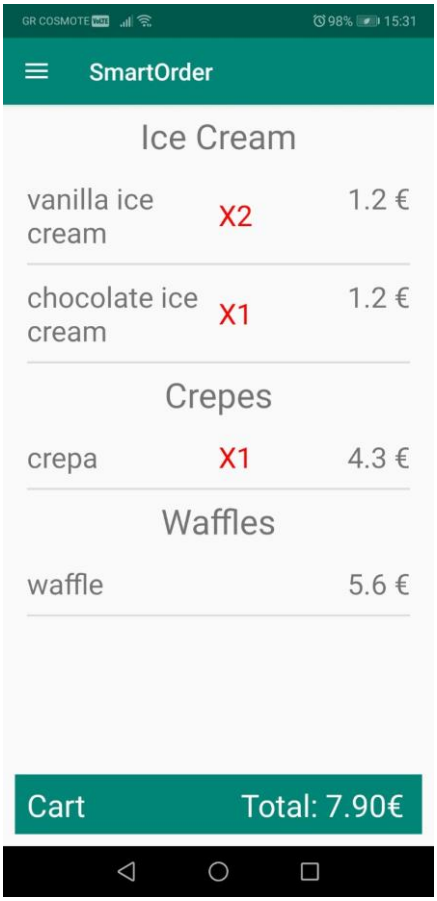




Επιλογή Προϊόντος

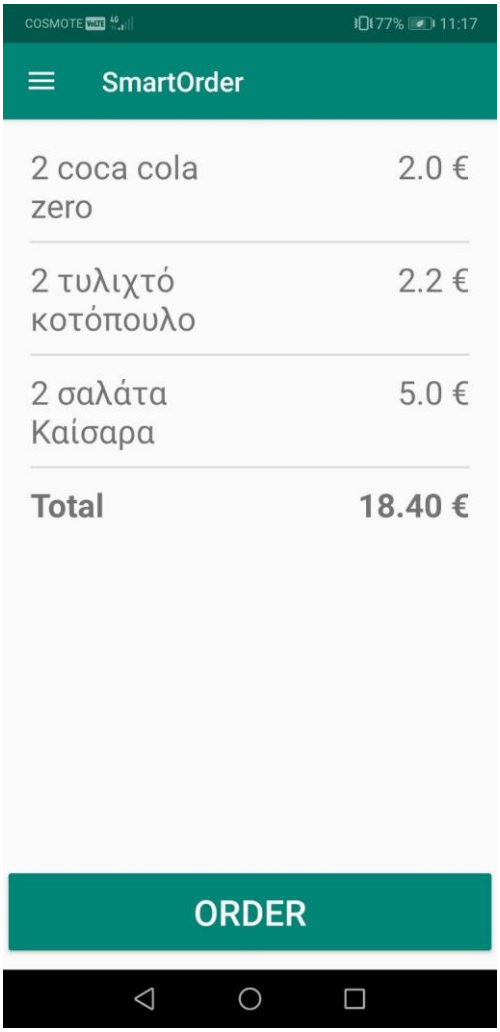
Με κλικ σε ένα συγκεκριμένο προϊόν, εμφανίζεται παράθυρο όπου ο χρήστης επιλέγει την ποσότητα που επιθυμεί. (Αριστερή εικόνα). Οι κόκκινοι χαρακτήρες δείχνουν την ποσότητα κάθε προϊόντος και στο τέλος εμφανίζεται το συνολικό ποσό πληρωμής. (Δεξιά εικόνα). Αν το επιλέξει ο χρήστης ανακατευθύνεται στην παραγγελία.

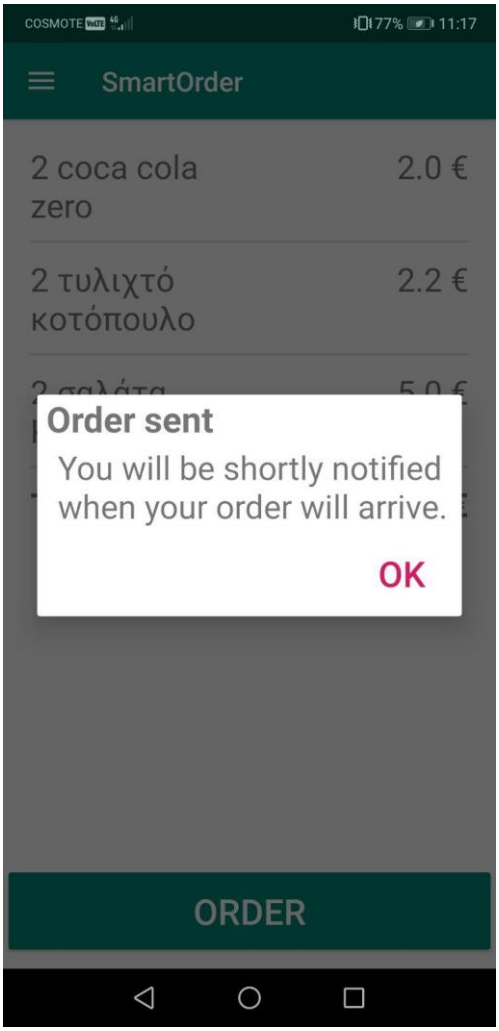




Ολοκλήρωση Παραγγελίας

Εμφανίζονται τα προϊόντα που έχει επιλέξει ο χρήστης, η ποσότητα από το κάθε ένα, η τιμή του και το συνολικό ποσό πληρωμής. Για την ολοκλήρωση της παραγγελίας επιλέγει το ORDER. (Αριστερή εικόνα) Εμφανίζεται μήνυμα που ενημερώνει τον χρήστη πως θα ενημερωθεί για την εξέλιξη της παραγγελίας του. (Δεξιά εικόνα)





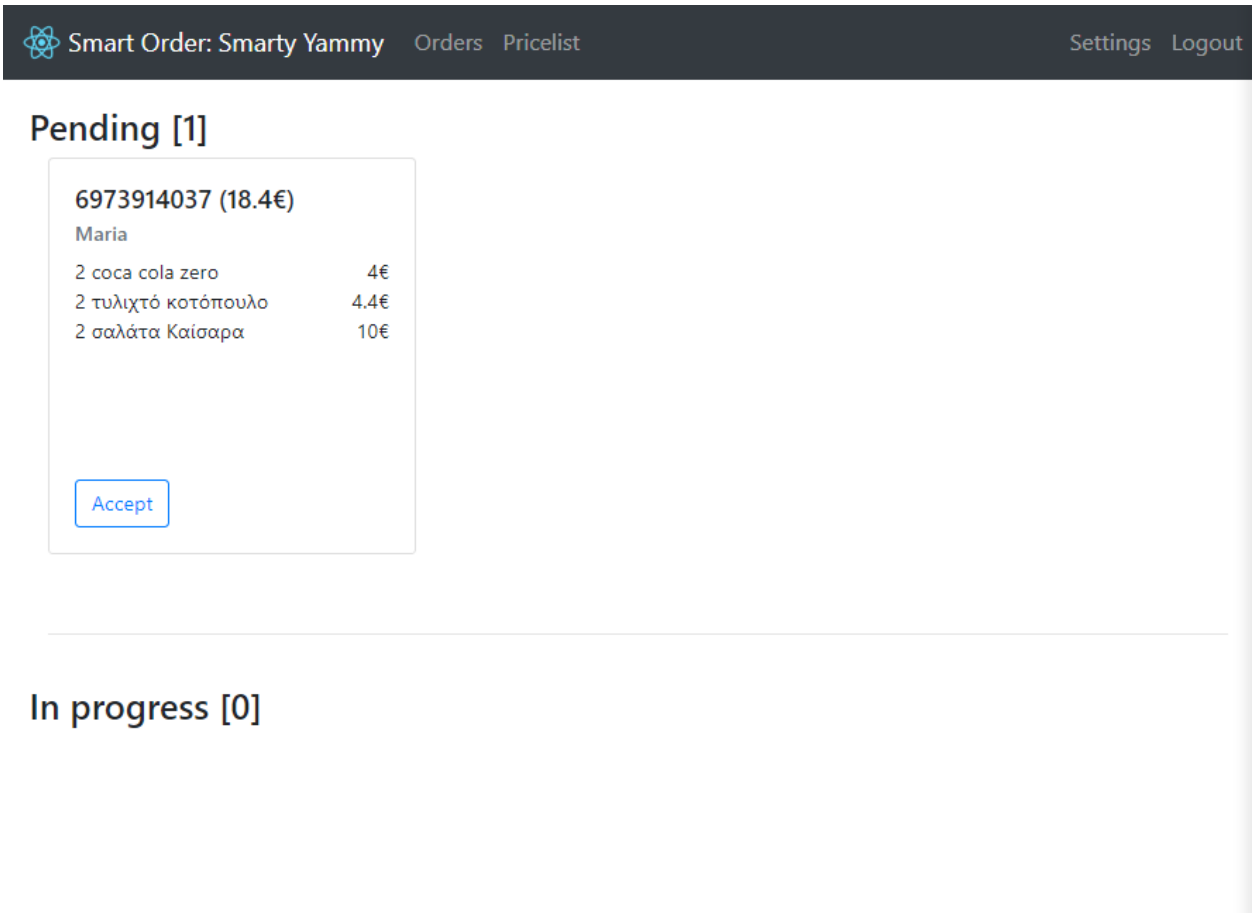
Παραγγελίες (Orders) Καταστήματος

Pending

Παραγγελίες που έχουν σταλεί από πελάτες αλλά δεν εχει δει ο χρήστης του καταστήματος

In progress

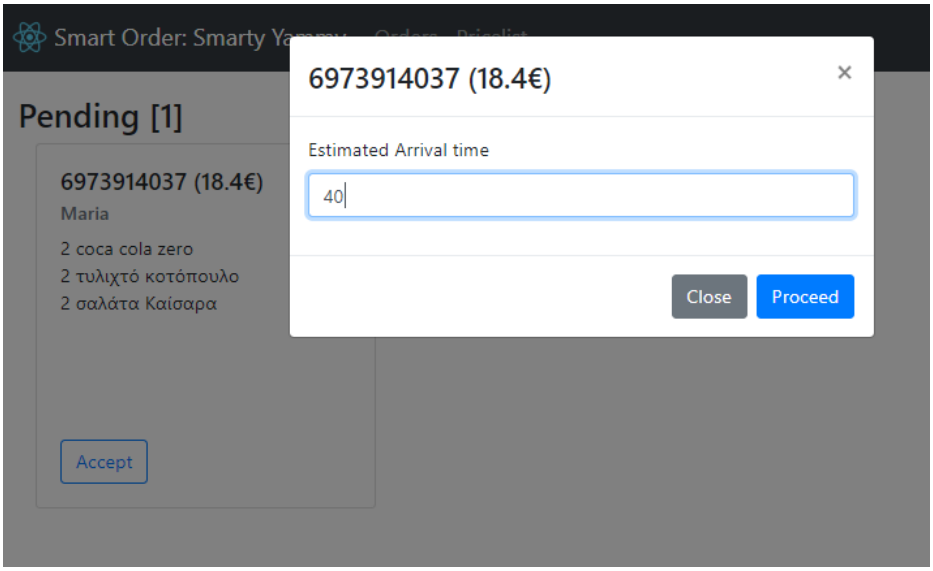
Παραγγελίες που έχει αποδεχτεί ο χρήστης του καταστήματος και βρίσκονται σε εξέλιξη.

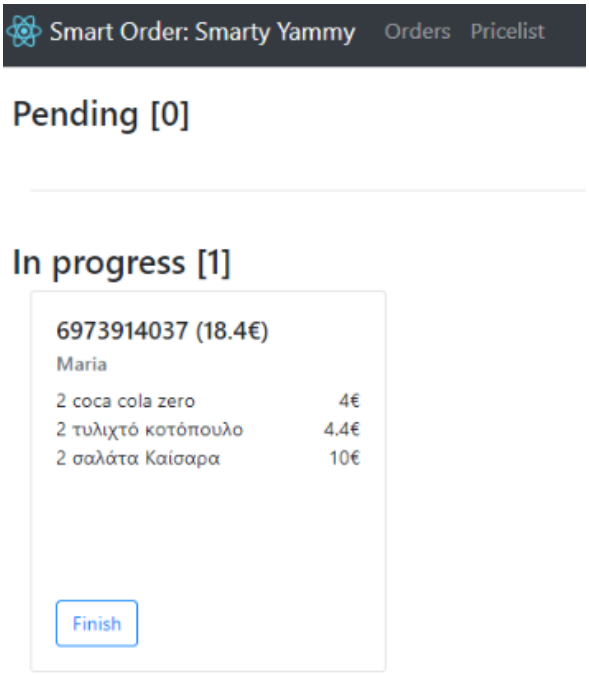


Στην κάθε παραγγελία φαίνεται το τηλέφωνο του πελάτη, τα προϊόντα που έχει επιλέξει καθώς και η συνολική τιμή της παραγγελίας. Μόλις επιλέξει ο χρήστης το πλήκτρο *Accept* εμφανίζεται στην οθόνη το παρακάτω Pop-up παράθυρο ώστε να πληκτρολογήσει τον χρόνο που χρειάζεται για την ολοκλήρωση της παραγγελίας.

Από *Pending* σε *In Progress*

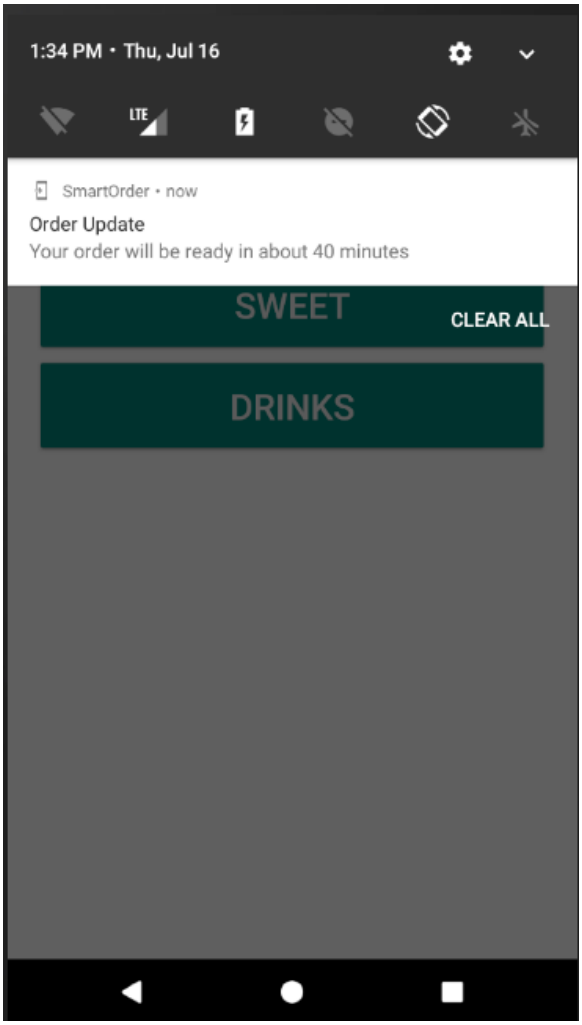
Ο χρήστης του καταστήματος ενημερώνει τον πελάτη (αποστολή notification) σε πόση ώρα περίπου θα φτάσει η παραγγελία του. Μόλις επιλέξει το πλήκτρο *Proceed* η παραγγελία πλέον βρίσκεται σε εξέλιξη (In progress).





Notification πελάτη

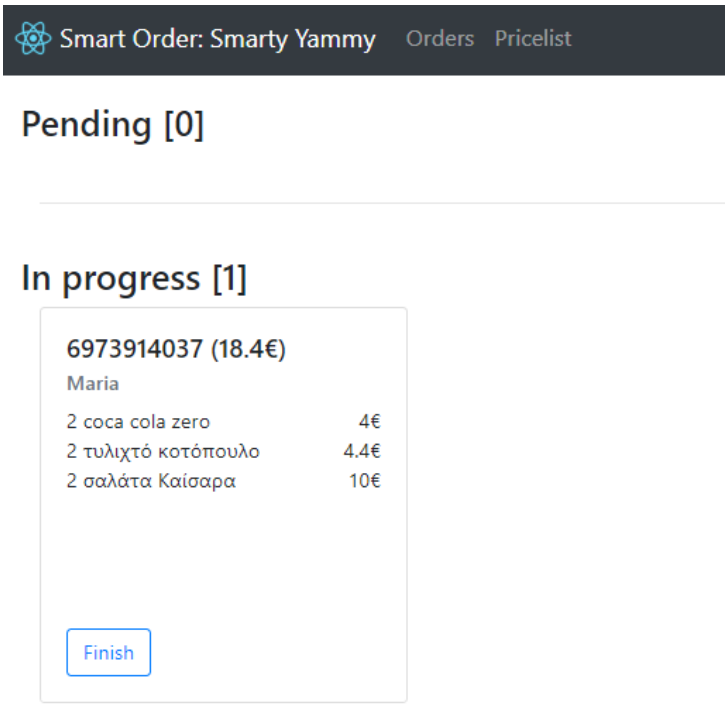
Ο πελάτης λαμβάνει notification με τον χρόνο άφιξης της παραγγελίας. Το μήνυμα είναι και ηχητικό, ώστε να μη χρειάζεται να ανοίξει το κινητό του για να το διαβάσει.

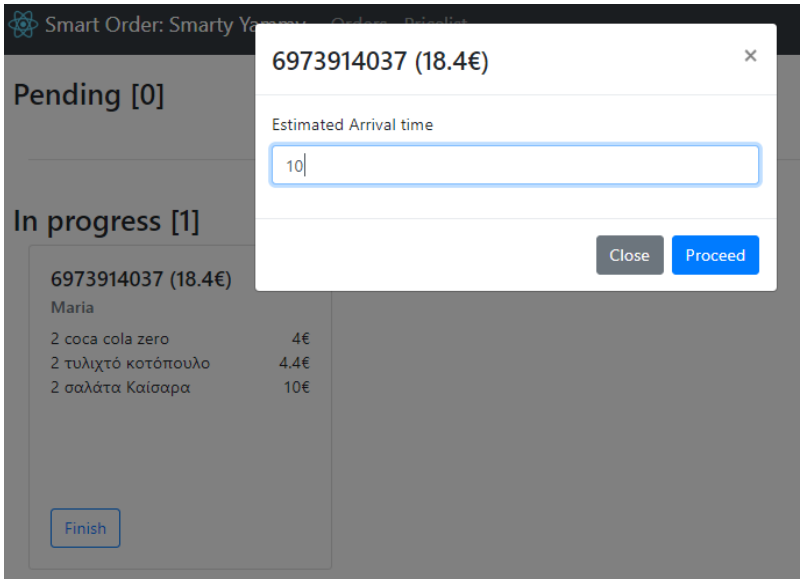


Ολοκλήρωση παραγγελίας

Από *In progress σε Αποστολή*

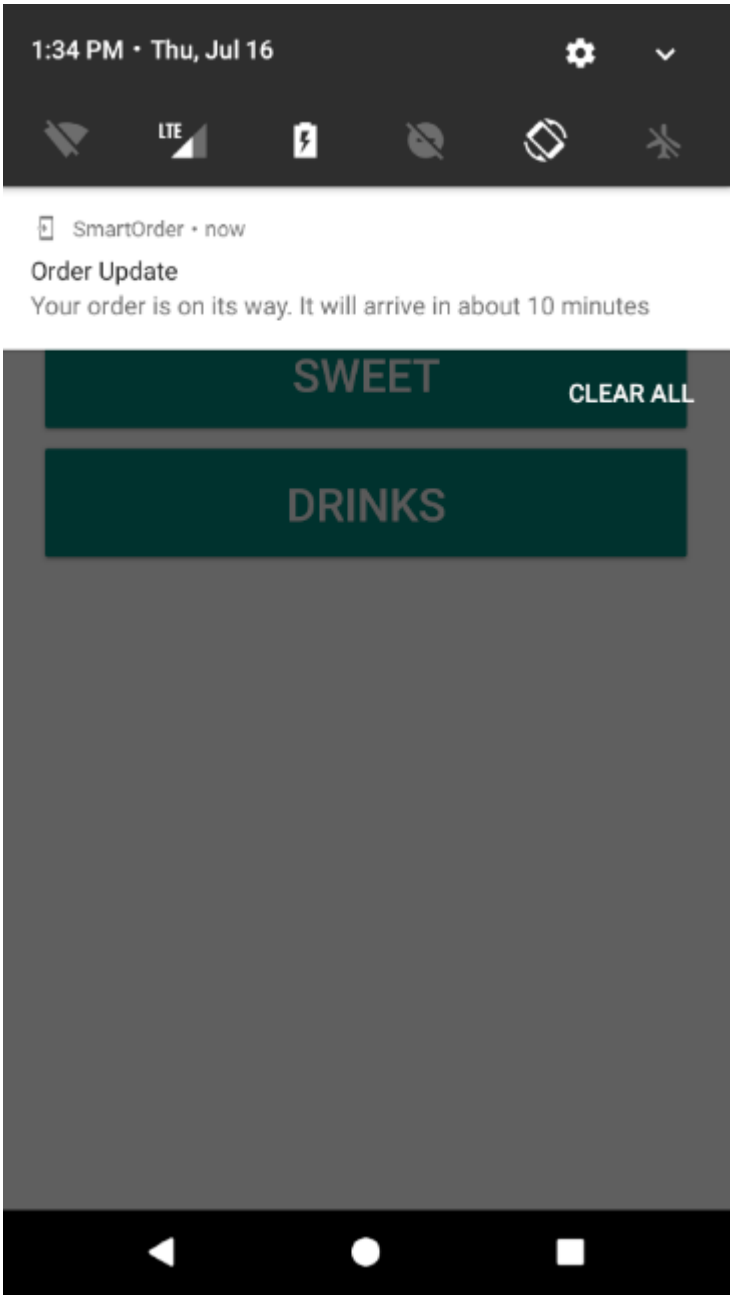
Μόλις ολοκληρωθεί η παραγγελία και είναι σε κατάσταση προς αποστολή, ο χρήστης επιλέγει το πλήκτρο *Finish* ώστε να εμφανιστεί το παρακάτω pop-up παράθυρο και να ενημερώσει τον πελάτη ότι η παραγγελία του βρίσκεται προς αποστολή και σε πόση ώρα περίπου θα φτάσει (αποστολή notification σε πελάτη).





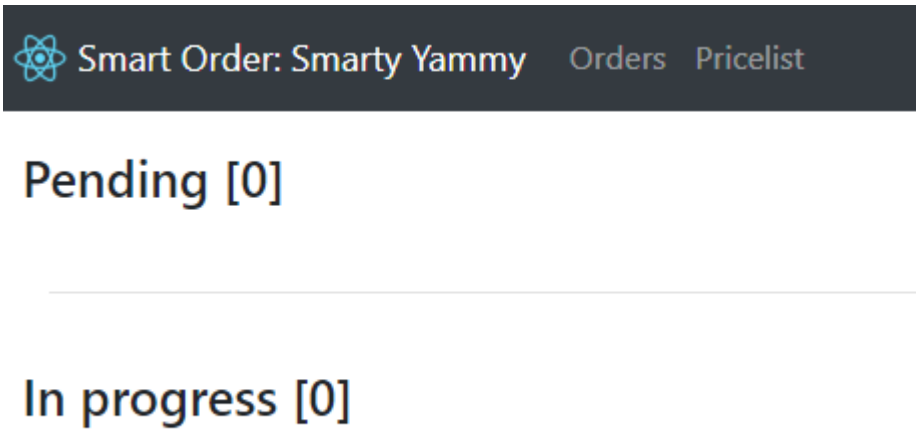
Notification πελάτη

Μόλις ο χρήστης του συστήματος του καταστήματος επιλέξει το πλήκτρο *Proceed*, ο πελάτης λαμβάνει notification οτι η παραγγελία έχει αποσταλεί. Το μήνυμα είναι και ηχητικό, ώστε να μη χρειάζεται να ανοίξει το κινητό του για να το διαβάσει.



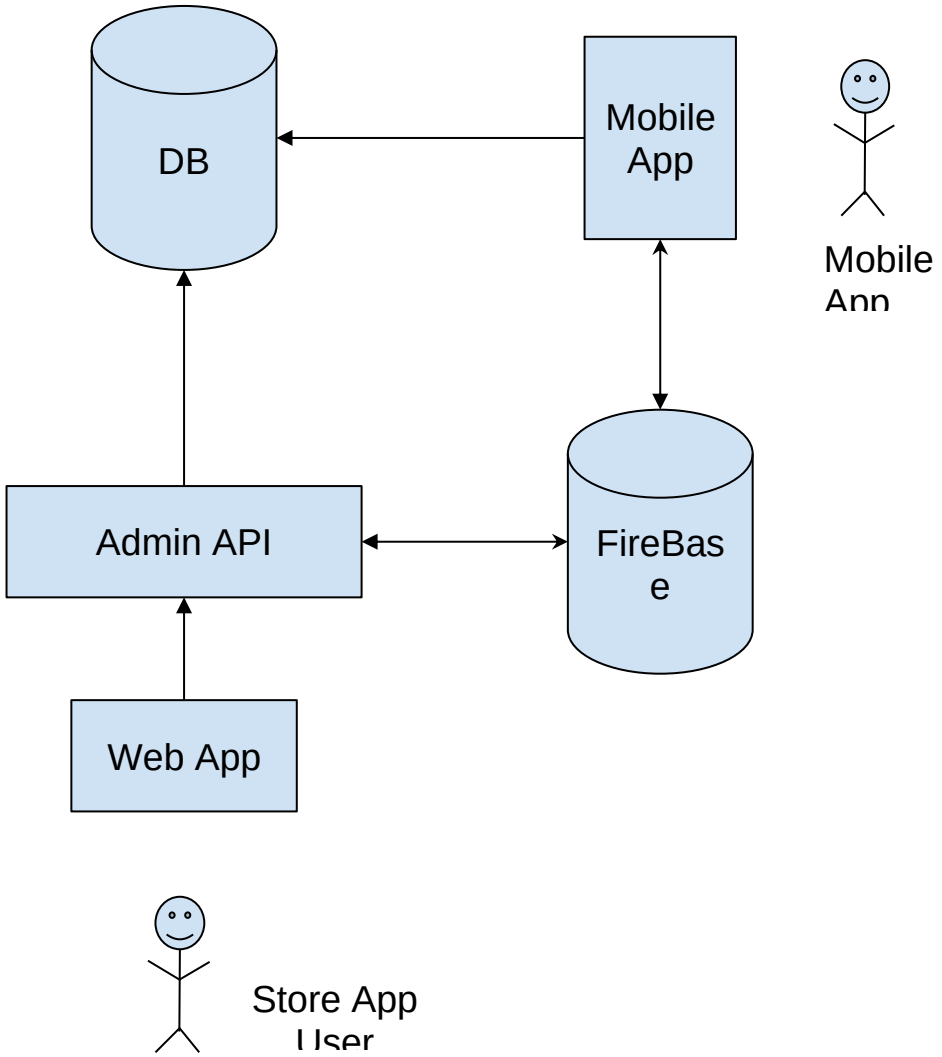
Παραγγελίες καταστήματος

Η παραγγελία έχει ολοκληρωθεί. Το κατάστημα δεν έχει άλλη παραγγελία ούτε *Pending* ούτε *In progress*.



ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΣΥΣΤΗΜΑΤΟΣ

Αρχιτεκτονικό Διάγραμμα



Mobile App

Η Native Android εφαρμογή είναι γραμμένη σε Java (Version 8) και έχει υλοποιηθεί μέσω του Android Studio. Εκπονήθηκε στα πλαίσια της εργασίας για το μάθημα *Ιατρική Πληροφορική*. Έχουν γίνει κάποιες προσθήκες οι οποίες θα αναλυθούν παρακάτω.

Web App

Πρόκειται για την εφαρμογή που χρησιμοποιεί ο user του καταστήματος ώστε να διαχειρίζεται τις παραγγελίες του. Η web εφαρμογή είναι γραμμένη σε HTML - Javascript χρησιμοποιώντας το Framework React.js

Admin API

Το back-end για τη διαχείριση του καταστήματος είναι γραμμένο σε C# .Net Core 3.1. Πρόκειται για ένα rest API, ώστε το (store) UI να επικοινωνεί με τη Βάση.

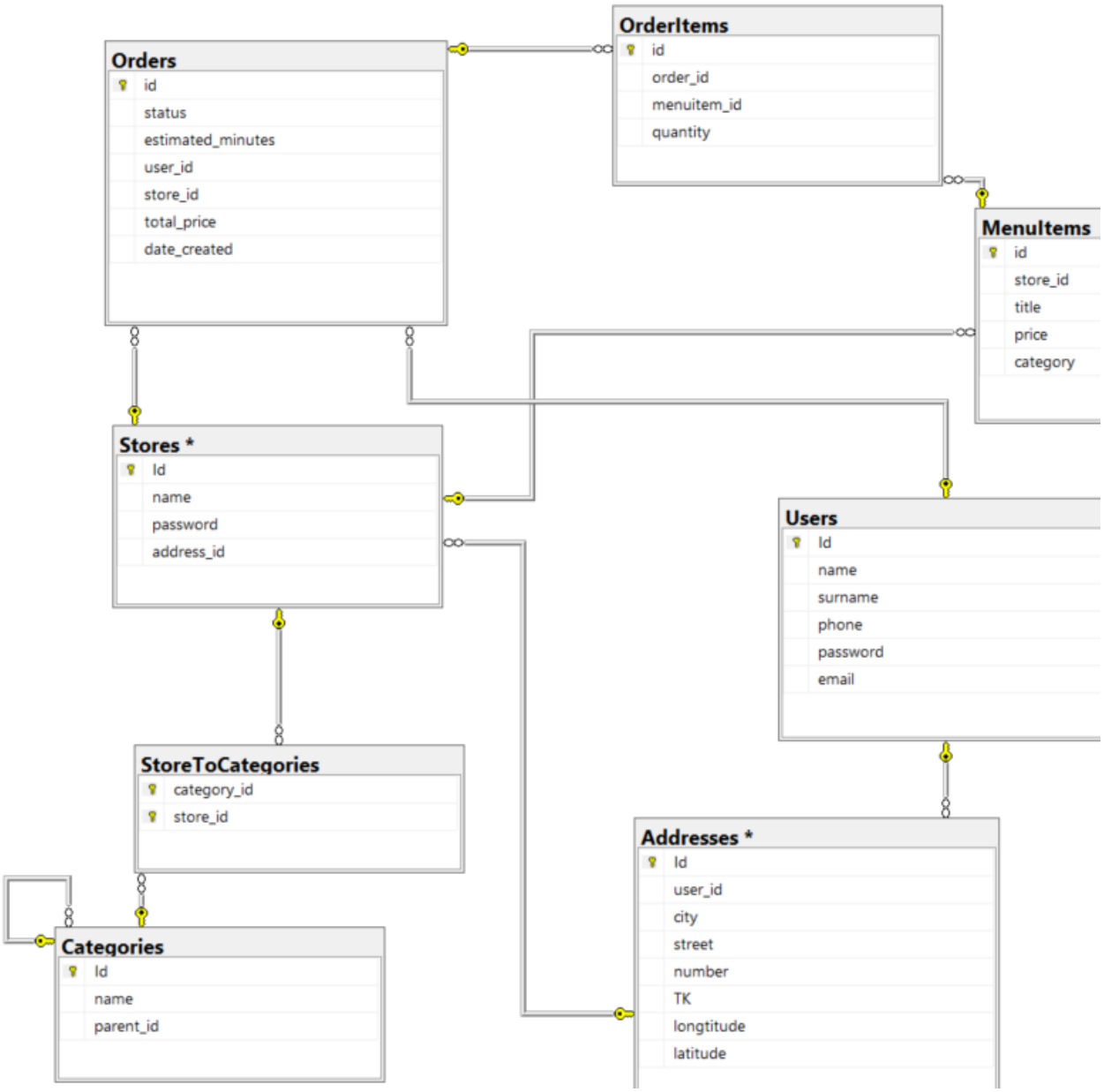
FireBase

Χρησιμοποιείται ώστε να στέλνονται real time notifications στους users (πχ σε πόση ώρα θα είναι έτοιμη η παραγγελία)

DataBase

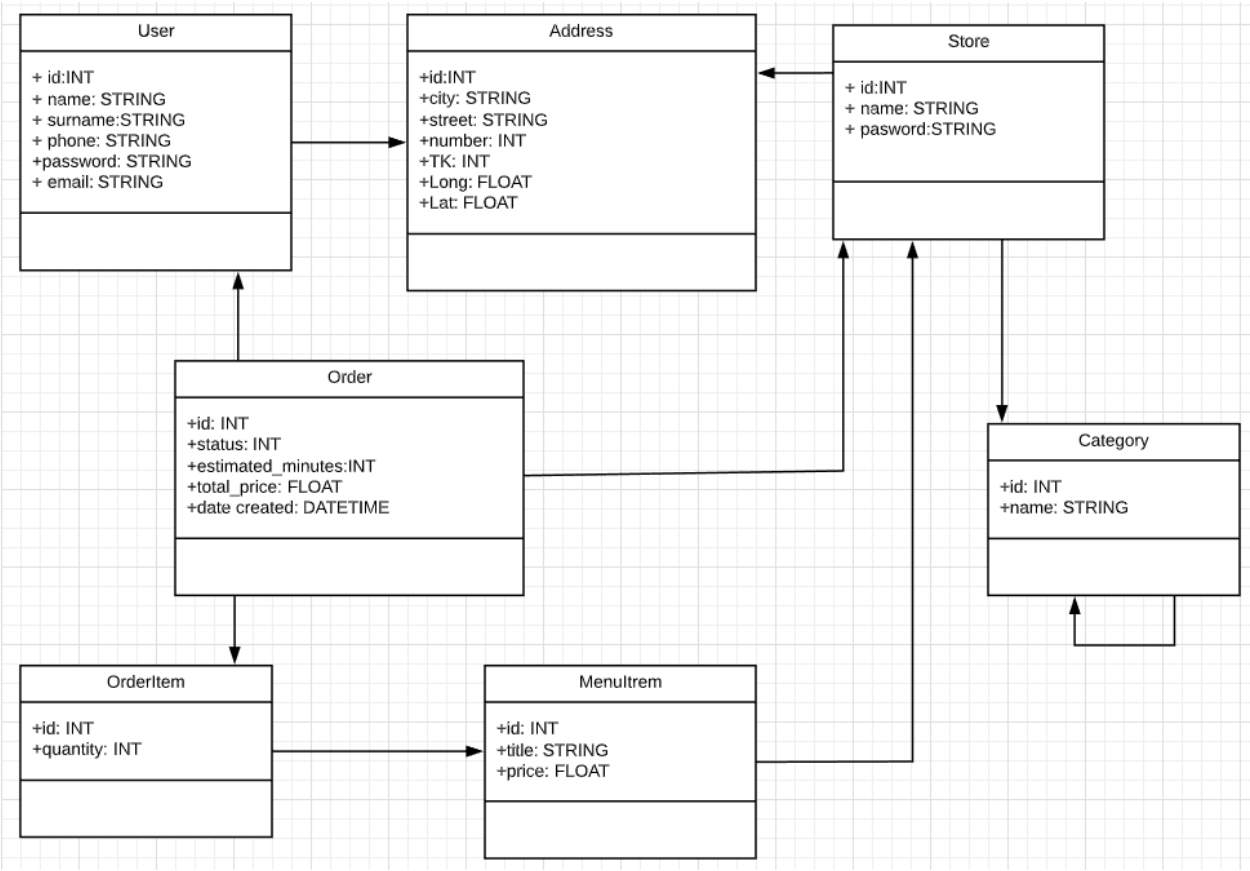
Για την αποθήκευση των δεδομένων των καταστημάτων και των παραγγελιών χρησιμοποιείται η βάση δεδομένων Microsoft SQL (MSSQL). Τόσο η Android εφαρμογή όσο και το admin API επικοινωνούν με αυτή τη βάση ώστε να έχουν τα ίδια δεδομένα.

Σχήμα Βάσης Δεδομένων



UML Class Diagram

Το διάγραμμα τάξεων είναι ο συνηθέστερος τύπος διαγράμματος για την τεκμηρίωση του λογισμικού. Περιέχει κατηγορίες, παράλληλα με τα χαρακτηριστικά τους και τις συμπεριφορές τους. Η σχέση μεταξύ των διαφορετικών τάξεων (που αντιπροσωπεύεται από μια γραμμή σύνδεσης), αποτελεί ένα διάγραμμα τάξεων



Εργαλεία που χρησιμοποιήθηκαν

- Android Studio
- Visual Studio Code
- Microsoft SQL Server Management Studio
- Npm

Παρουσίαση Κώδικα

Εφαρμογή Android

Η βασική εφαρμογή Android, που χρησιμοποιείται από τον πελάτη, εκπονήθηκε στα πλαίσια της εργασίας της *Ιατρικής Πληροφορικής*.

Base Activity

Έχει δημιουργηθεί ένα basic Activity το οποίο κάνουν extend όλα τα άλλα activities και περιέχει κοινή λογική που χρειάζονται τα περισσότερα activities

```
//Base Activity που περιέχει κοινή λογική για τα περισσότερα Activities
/* Side Menu
// * Order Updates/ Notification
public class BasicActivity extends AppCompatActivity {

    private DrawerLayout dl;
    private NavigationView nv;
    private ActionBarDrawerToggle drawerToggle;
```

Για παράδειγμα, το Side Menu υπάρχει σε όλες της activities ώστε να μπορεί ο χρήστης να κάνει αποσύνδεση όπου κι αν βρίσκεται.

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    //Δημιουργία Side Menu
    nv = findViewById(R.id.nv);
    dl = findViewById(R.id.activity_menu);
    drawerToggle = new ActionBarDrawerToggle( activity: this, dl, "Sign in or register", "SmartOrder");
    dl.addDrawerListener(drawerToggle);
    drawerToggle.syncState();
    getSupportActionBar().setDisplayHomeAsUpEnabled(true);

    nv.setNavigationItemSelectedListener((item) → {
            int id = item.getItemId();
            switch (id) {
                case R.id.logoutItem:
                    logout();
                    return true;
            }

            return false;
    });
```

Helper Activity

Περιέχει βοηθητικές συναρτήσεις που χρειάζονται διάφορα activities. Για παράδειγμα, αποθηκεύεται στα Preferences της εφαρμογής ο συνδεδεμένος χρήστης, ώστε να μη χρειάζεται να κάνει Login κάθε φορά που ανοίγει την εφαρμογή.

```
//Αποθηκεύεται στα Preferences της εφαρμογής ο συνδεδεμένος χρήστης
//ώστε να μη χρειάζεται να κάνει Login κάθε φορά που ανοίγει την εφαρμογή
public static void setCurrentUser(Activity activity, int userId) {
    SharedPreferences.Editor editor = activity.getSharedPreferences( name: "SmartOrderPrefs", MODE_PRIVATE).edit();
    editor.putInt("userId", userId);
    editor.apply();
}

//επιστρέφει τον χρήστη από τα Preferences ώστε στη συνέχεια να γίνει αυτόματο Login
public static int getCurrentUser(Activity activity) {
    SharedPreferences preferences = activity.getSharedPreferences( name: "SmartOrderPrefs", MODE_PRIVATE);
    return preferences.getInt( key: "userId",  defValue: 0);
}
```

Οι static μεταβλητές που αποθηκεύονται κάποια δεδομένα ώστε να περνάνε στο επόμενο activity. Δεν χρησιμοποιήθηκαν οι putExtra, διότι ήθελα να περάσω ολόκληρα αντικείμενα και όχι απλούς τύπους

```
//είναι οι static μεταβλητές που αποθηκεύονται κάποια δεδομένα ώστε να περνάνε στο επόμενο activity
//δεν χρησιμοποίησα τις putExtra διότι ήθελα να περάσω ολόκληρα αντικείμενα και όχι απλούς τύπους
public static void setCurrentOrder(Order order) { activeOrder = order; }

public static Order getCurrentOrder() { return activeOrder; }

public static void setEditingAddress(Address address) { editingAddress = address; }

public static Address getEditingAddress() { return editingAddress; }
```

Login Activity

Ελέγχεται αν ο συγκεκριμένος χρήστης έχει ξανασυνδεθεί, ώστε να μεταφερθεί κατευθείαν στις κατηγορίες.

```
//Ελέγχει αν ο χρήστης έχει ξαναμπεί
private void autoLogin() {
    int userId = Helper.getCurrentUser( activity: this);
    if (userId <= 0) {
        return;
    }

    //ελέγχει αν το id υπάρχει στη βάση
    User user = repo.getUser(userId);
    if (user == null) {
        return;
    }

    Intent intent = new Intent(getApplicationContext(), HomeActivity.class);
    startActivity(intent);
}
```

```
public void loginBtn(View view){

    User user = repo.getUser(loginEmailTxt.getText().toString(), loginPasswordTxt.getText().toString());

    if (user == null){
        Toast.makeText( context: this,  text: "Wrong User", Toast.LENGTH_SHORT).show();
        return;
    }
    //καλει την setCurrentUser ώστε να θυμάται μόλις ξανανοίξει η εφαρμογή
    // ποιος χρήστης έχει κάνει Login για να μπει αυτόματα
    Helper.setCurrentUser( activity: this, user.getId());

    Intent intent = new Intent(getApplicationContext(), HomeActivity.class);
    startActivity(intent);
}
```

Register Activity

Γίνεται έλεγχος των στοιχείων (να έχουν συμπληρωθεί όλα τα πεδία και το τηλέφωνο να είναι μεταξύ κάποιων συγκεκριμένων χαρακτήρων) και δημιουργείται νέος χρήστης.

```
//Δημιουργία νέου χρήστη αφού ελέγξει ότι έχουν συμπληρωθεί όλα τα πεδία
User u = new User();
u.setName(nameTxt.getText().toString());
u.setSurname(surnameTxt.getText().toString());
u.setPassword(passwordTxt.getText().toString());
u.setEmail(emailTxt.getText().toString());
u.setPhone(phoneTxt.getText().toString());

if (nameTxt.getText().toString().isEmpty()){
    Toast.makeText( context: this,  text: "Please insert Name", Toast.LENGTH_LONG).show();
    return;
}

if (surnameTxt.getText().toString().isEmpty()){
    Toast.makeText( context: this,  text: "Please insert Surname", Toast.LENGTH_LONG).show()
    return;
```

```
//ελέγχει αν ειναι έγκυρο το email
public static boolean isValidEmail(CharSequence target){
    if (TextUtils.isEmpty(target)){
        return false;
    }else {
        return Patterns.EMAIL_ADDRESS.matcher(target).matches();
    }
}
//ελέγχει αν ειναι έγκυρο το τηλέφωνο
public boolean isValidPhone(String phone){
    if (phone.length() < 6 || phone.length() > 13) {
        Toast.makeText( context: this,  text: "Phone number must have 6 to 13 digits", Toast.LENGTH_LONG).sho
        return false;
    }
    return true;
}
```

Home Activity (Categories)

Για κάθε κατηγορία (πχ φαΐ, γλυκό) δημιουργείται ένα button. Το κάθε ένα ανακατευθύνει τον χρήση στις υποκατηγορίες της επιλεγμένης κατηγορίας.

```
//για κάθε βασική κατηγορία δημιουργείται ενα button,
//το οποίο μας οδηγεί σε επόμενο activity με τις εκάστοτε υποκατηγορίες
private void createCategories() {
    LinearLayout layout = findViewById(R.id.mainMenuLayout);

    for (final Category cat: repo.getRootCategories()){
        Button btn = new Button( context: this);
        btn.setText(cat.getName());
        btn.setTextSize(26);
        //btn.setTextColor();
        btn.setPadding( left: 0,  top: 50,  right: 0,  bottom: 50);
        layout.addView(btn);

        btn.setOnClickListener((v) → {
                Intent intent = new Intent(getApplicationContext(), SubcategoriesActivity.class);
                intent.putExtra( name: "catId", cat.getId());
                startActivity(intent);
        });
```

Stores Activity

Υπολογίζονται τα καταστήματα που βρίσκονται σε απόσταση μικρότερη των 10 χιλιομέτρων από τη διεύθυνση του χρήστη

```
//υπολογισμός απόστασης μεταξύ χρήστη και καταστήματος
for (final Store store: repo.getStores(getIntent().getIntExtra( name: "subCatId",  defaultValue: 0)
    double dist = calculateDistance(
            store.getAddress().getUserLong(), store.getAddress().getUserLat(),
            user.getAddress().getUserLong(), user.getAddress().getUserLat()
    );

    //αν ειναι μεγαλύτερο απο 10Km απόσταση δεν το επιστρέφει
    if (dist > 10000) {
        continue;
    }
```

Menu Activity

Στο SideMenu προστίθεται η κάθε κατηγορία που προσφέρει το συγκεκριμένο κατάστημα ώστε να διευκολυνθεί ο χρήστης και να μη χρειάζεται να διαβάσει όλο το μενού.

```
//Προσθέτουμε στο side menu τις κατηγορίες προϊόντων που προσφέρει το κατάστημα
//ώστε να επιλέξει κατευθείαν ο χρήστης κατηγορία (πχ σαλάτες)
ArrayList<String> categories = new ArrayList<>();
for (com.studentsunipi.spanou.smartorder.model.MenuItem mi : menuItems) {
    if (categories.contains(mi.getCategory()) == false) {
        categories.add(mi.getCategory());
    }
}

for (final String category : categories) {
    nv.getMenu().add(category).setOnMenuItemClickListener((item) → {
            focusOnView(categorySections.get(category));
            dl.closeDrawers();
            return true;
    });
    createCategorySection(category);
}
```

Για κάθε κατηγορία εμφανίζεται μια ενότητα που περιέχει τον τίτλο και τα προϊόντα (πχ Σαλάτα περιέχει του καίσαρα, χωριάτικη κλπ)

```
//Για κάθε κατηγορία προϊόντος εμφανίζεται μια ενότητα που περιέχει
//τον τίτλο της κατηγορίας και τα προϊόντα
private void createCategorySection(String categoryName) {
    LinearLayout categoryLayout = new LinearLayout( context: this);
    categoryLayout.setOrientation(LinearLayout.VERTICAL);
    categoryLayout.setId(categoryName.hashCode());
    categorySections.put(categoryName, categoryLayout);

    mainLayout.addView(categoryLayout);

    TextView categoryTitle = new TextView( context: this);
    LinearLayout.LayoutParams categoryParams = new LinearLayout.LayoutParams(
            LinearLayout.LayoutParams.MATCH_PARENT,
            LinearLayout.LayoutParams.MATCH_PARENT,
             weight: 1
    );
    categoryParams.setMargins( left: 0,  top: 20,  right: 0,  bottom: 20);
    categoryTitle.setLayoutParams(categoryParams);
    categoryTitle.setText(categoryName);
    categoryTitle.setTextSize(29);
    categoryTitle.setGravity(Gravity.CENTER);
    categoryLayout.addView(categoryTitle);

    for (final com.studentsunipi.spanou.smartorder.model.MenuItem menuItem: menuItems){
        if (menuItem.getCategory().equals(categoryName) == false) {
            continue;
        }

        createMenuItemSection(categoryLayout, menuItem);
    }
}
```

Για κάθε προϊόν (φαΐ, αναψυκτικό) δημιουργείται μια γραμμή με τα απαραίτητα στοιχεία (όνομα, τιμή).

```
//εμφανίζει μια γραμμή για κάθε προϊόν
private void createMenuItemSection(LinearLayout parent, final com.studentsunipi.sp
    //line
    LinearLayout layout = new LinearLayout( context: this);
    layout.setOrientation(LinearLayout.HORIZONTAL);
    LinearLayout.LayoutParams containerParams = new LinearLayout.LayoutParams(
            LinearLayout.LayoutParams.MATCH_PARENT,
            LinearLayout.LayoutParams.MATCH_PARENT
    );
    containerParams.setMargins( left: 0,  top: 30,  right: 0,  bottom: 30);
    layout.setLayoutParams(containerParams);
    parent.addView(layout);

    //product title label
    TextView label = new TextView( context: this);
    label.setLayoutParams(new LinearLayout.LayoutParams(
         width: 0,
        LinearLayout.LayoutParams.MATCH_PARENT,
         weight: 2.0f
    ));
    label.setText(menuItem.getName());
    label.setTextSize(24);
    layout.addView(label);
```

Στην επιλογή συγκεκριμένου προϊόντος, εμφανίζεται παράθυρο ώστε ο χρήστης να επιλέξει ποσότητα που επιθυμεί.

```
//στο κλικ της γραμμής εμφανίζεται ένα PopUp όπου ο χρήστης επιλέγει ποσότητα
layout.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if (orderItem.getQuantity() == 0) {
            orderItem.setQuantity(1);
        }
        createMenuItemPopup(menuItem, orderItem);
    }
});
```

Στο Popup υπάρχουν τα πλήκτρα αύξησης και μείωσης ποσότητας, textview για να φαίνεται η ποσότητα.

```
//buttons
final Button addBtn = new Button( context: MenuActivity.this);
addBtn.setLayoutParams(new LinearLayout.LayoutParams(
        LinearLayout.LayoutParams.MATCH_PARENT,
        LinearLayout.LayoutParams.MATCH_PARENT,
         weight: 1.0f
));
final Button removeBtn = new Button( context: MenuActivity.this);
removeBtn.setLayoutParams(new LinearLayout.LayoutParams(
        LinearLayout.LayoutParams.MATCH_PARENT,
        LinearLayout.LayoutParams.MATCH_PARENT,
         weight: 1.0f
));
final TextView contLbl = new TextView( context: MenuActivity.this);
contLbl.setLayoutParams(new LinearLayout.LayoutParams(
        LinearLayout.LayoutParams.MATCH_PARENT,
        LinearLayout.LayoutParams.MATCH_PARENT,
         weight: 1.0f
));
```

```
addBtn.setOnClickListener((v) -> {
        orderItem.setQuantity(orderItem.getQuantity() + 1);
        contLbl.setText(orderItem.getQuantity() + "");
        removeBtn.setEnabled(true);
});

removeBtn.setOnClickListener((v) -> {
        if (orderItem.getQuantity() == 0) {
            return;
        } else if (orderItem.getQuantity() == 1) {
            removeBtn.setEnabled(false);
        }

        orderItem.setQuantity(orderItem.getQuantity() - 1);
        contLbl.setText(orderItem.getQuantity() + "");
});
```

```java
@Override
public void onClick(DialogInterface dialog, int which) {
    OrderItem existingOrderItem = order.getOrderItem(menuItem);

    TextView quantityLbl = findViewById(getQuantityLabelId(menuItem));

    if (orderItem.getQuantity() == 0) {
        quantityLbl.setText("");
        if (existingOrderItem != null) {
            order.getOrderItems().remove(existingOrderItem);
        }
    } else {
        quantityLbl.setText("X" + orderItem.getQuantity());
        if (existingOrderItem != null) {
            existingOrderItem.setQuantity(orderItem.getQuantity());
        } else {
            order.addOrderItem(menuItem, orderItem.getQuantity());
        }
    }

    updateTotalPrice();
}
```

Ανακατεύθυνση (Scroll) σε συγκεκριμένη κατηγορία (που επέλεξε ο χρήστης). Μοναδικό id για κάθε προϊόν, και τέλος ενημέρωση συνολικής τιμής.

```java
//σκρολάρει σε συγκεκριμένο σημείο (view) της οθόνης
private final void focusOnView(final View item){
    scrollView.post(() -> { scrollView.scrollTo( x: 0, item.getTop()); });
}
//δίνεται μοναδικό id σε κάθε menu item για να βρούμε το view Που ανήκει σε κάθε menuItem
private int getQuantityLabelId(com.studentsunipi.spanou.smartorder.model.MenuItem menuItem) {
    return 1000 + menuItem.getId();
}

//ανανεώνει τη συνολική τιμής της παραγγελίας και το εμφανίζει
private void updateTotalPrice() {
    order.updateTotalPrice();
    TextView tv = findViewById(R.id.totalPriceTxtV);
    String roundedPrice = String.format("%.2f", order.getTotalPrice());
    tv.setText("Total: " + roundedPrice + "€");
}
```

Order Activity

Για κάθε προϊόν που έχει επιλεχθεί δημιουργείται μια γραμμή, που περιλαμβάνει το όνομα του, την ποσότητα και την τιμή. Τέλος, φαίνεται το συνολικό (τελικό) ποσό πληρωμής.

```
//για καθε item που έχει παραγγείλει εμφανίζει μια γραμμή
// με την περιγραφή την ποσότητα και την τιμή
private void drawOrderItems() {
    LinearLayout parent = findViewById(R.id.orderLayout);
    for (OrderItem item : order.getOrderItems()) {
        drawItem(parent, left: item.getQuantity() + " " + item.getMenuItem().getName(),
                right: item.getMenuItem().getPrice() + " €", bold: false);
        parent.addView(new MyDivider( context: this));
    }

    String roundedPrice = String.format("%.2f", order.getTotalPrice());

    //εμφανίζει μια γραμμή με το συνολικό ποσό της παραγγελίας
    drawItem(parent, left: "Total", right: roundedPrice + " €", bold: true);
}
```

Αποθήκευση της παραγγελίας στη Βάση Δεδομένων

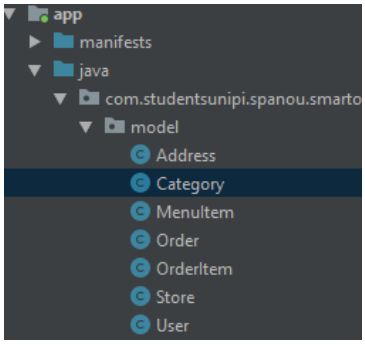
```
public void placeOrder(View view) {
    //αποθηκεύει στη Βάση την παραγγελία
    repo.saveOrder(order, Helper.getCurrentUser( activity: this), storeId);

    //εμφανίζει μήνυμα ενημέρωσης παραγγελίας
    AlertDialog.Builder builder = new AlertDialog.Builder( context: OrderActivity.this);
```

Repository

Περιέχει μεθόδους για να μιλάμε με τη βάση δεδομένων και να μετατρέπουμε java instances σε Insert & Update SQL statements και το αντίστροφο, δηλαδή sql select statements σε java instances. Ενδεικτική μέθοδος του repository, βλέπει στη βάση αν υπάρχει ο συγκεκριμένος χρήστης (Mail & password) και τον επιστρέφει.

```java
//βλέπει στη βάση αν υπάρχει χρήστης με το συγκεκριμένο email & password και αν υπάρχει τον επιστρέφει
public User getUser(String email, String password) {
    Connection conn = connect();
    User user = null;

    try (Statement stmt = conn.createStatement()) {
        //το ResultSet περιέχει τις εγγραφές(γραμμές) που επέστρεψε το query στη Βάση
        ResultSet resultat = stmt.executeQuery(
                sql: "SELECT * FROM Users where email = '" + email + "' AND password = '" + password + "';");

        //αν υπάρχει έστω μία, φέρε τη
        if (resultat.next()) {
            user = parseUser(resultat);
        }

        resultat.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }

    return user;
}
```

Χρησιμοποιείται η βάση Microsoft SQL (MSSQL). Όλες οι κλάσεις για τους πίνακες που υπάρχουν στην βάση



Σύνδεση με τη firebase

Για την εκπόνηση της διπλωματικής, προστέθηκαν η σύνδεση με τη firebase, live notifications καθώς και οι φωνητικές εντολές (text to speech).

```
//Connection with FireBase for notifications
User user = Helper.getRepository().getUser(Helper.getCurrentUser( activity: this));

if (user == null) {
    return;
}

FirebaseDatabase database = FirebaseDatabase.getInstance();
DatabaseReference pendingOrdersRef = database.getReference( path: "pending_orders");
DatabaseReference finishedOrdersRef = database.getReference( path: "finished_orders");

pendingOrdersRef.addValueEventListener(new ValueEventListener() {
    @Override
    public void onDataChange(DataSnapshot dataSnapshot) {
        for (DataSnapshot item: dataSnapshot.getChildren()) {
            Long value = item.getValue(Long.class);
            int orderId = Integer.parseInt(item.getKey());
            if (_pendingOrders.contains(orderId)) {
                String message = "Your order will be ready in about " + value + " minutes";
                showNotification(message);
                tts.speak(message);
                _toBeFinishedOrders.add(orderId);
                _pendingOrders.removeIf(o -> o == orderId);
                item.getRef().removeValue();
                break;
            }
        }
    }

    @Override
    public void onCancelled(DatabaseError error) {
        // Failed to read value
        Log.w( tag: "FIREBASE",  msg: "Failed to read value.", error.toException());
    }
});
```

```
finishedOrdersRef.addValueEventListener(new ValueEventListener() {
    @Override
    public void onDataChange(DataSnapshot dataSnapshot) {
        for (DataSnapshot item: dataSnapshot.getChildren()) {
            Long value = item.getValue(Long.class);
            int orderId = Integer.parseInt(item.getKey());
            if (_toBeFinishedOrders.contains(orderId)) {
                String message = "Your order is on its way. It will arrive in about " + value + " minutes";
                showNotification(message);
                tts.speak(message);
                _toBeFinishedOrders.removeIf(o -> o == orderId);
                item.getRef().removeValue();
                break;
            }
        }
    }

    @Override
    public void onCancelled(DatabaseError error) {
        // Failed to read value
        Log.w( tag: "FIREBASE",  msg: "Failed to read value.", error.toException());
    }
});
```

Εφαρμογή Καταστήματος

Η web εφαρμογή που χρησιμοποιεί ο user του καταστήματος ώστε να διαχειρίζεται τις παραγγελίες του, να ενημερώνει τα προϊόντα του και να ενημερώνει τον κάθε πελάτη για την κατάσταση της παραγγελίας του.
Αποτελείται από δύο μέρη, το backend το οποίο είναι γραμμένο σε .net Core και το frontend που είναι γραμμένο σε HTML - Javascript χρησιμοποιώντας το Framework React.js.

Backend

Startup

Η κλάση startup χρησιμοποιείται από το .net Core για να κανει initialize τον server

```
2 references
public IConfiguration Configuration { get; }

// This method gets called by the runtime. Use this method to add services to the container.
0 references
public void ConfigureServices(IServiceCollection services)
{
    //Σύνδεση με Βάση Δεδομένων
    services.AddDbContext<DB_A25A8E_orderspapiContext>(opt =>
       opt.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

    //Επίτρεψε cross origin requests
    //για να μπορεί το front-end να χτυπάει τον server από άλλο domain name
    services.AddCors(options =>
            {
                options.AddPolicy("all", builder => builder
                    .AllowAnyOrigin()
                    .AllowAnyMethod()
                    .AllowAnyHeader()
                );
            });

    services.AddControllers();
}
```

Controllers

Store Controller

Για τη διαχείριση του κάθε καταστήματος έχει δημιουργηθεί ο store controller που περιέχει διάφορα actions που εξυπηρετούν διαφορετικούς σκοπούς, όπως είναι η εγγραφή νέου καταστήματος ή η προσθήκη προϊόντων κά

Εγγραφή νέου καταστήματος

Γίνεται έλεγχος των στοιχείων (έλεγχος password, αν το κατάστημα υπάρχει ήδη) και δημιουργείται το νέο κατάστημα.

```
//register ένα νέο κατάστημα
[HttpPost("Register")]
0 references
public string Register(RegisterDTO data)
{
    //Ensure passwords are same
    if (data.Password != data.RepeatPassword)
    {
        return "Passwords do not match";
    }

    //Ensure password is long enough
    if (data.Password == null || data.Password.Length < 3)
    {
        return "Password too short";
    }

    //Ensure store does not exist
    var store = _context.Stores.FirstOrDefault(s => s.Name == data.StoreName);
    if (store != null)
    {
        return "Store already exists";
    }

    //create store
    var nextId = 1;
    var latestStore = _context.Stores.OrderByDescending(s => s.Id).FirstOrDefault();
    if (latestStore != null)
    {
        nextId = latestStore.Id + 1;
    }
    var newStore = new Stores
    {
        Name = data.StoreName,
        Password = data.Password,
        Id = nextId
    };

    //Add store to DB
    _context.Stores.Add(newStore);
    _context.SaveChanges();

    return newStore.Id.ToString();
}
```

Login καταστήματος

Γίνεται έλεγχος ποιο κατάστημα κάνει login.

```
// Έλεγχος ποιο κατάστημα κάνει Login
[HttpGet("Authenticate/{username}/{password}")]
0 references
public int Authenticate(string username, string password)
{
    var store = _context.Stores.FirstOrDefault(s => s.Name == username && s.Password == password);

    //αν δε βρεθεί κατάστημα, επέστρεψε "0" (για να εμφανιστεί κατάλληλο μήνυμα)
    if (store == null)
    {
        return 0;
    }

    return store.Id;
}
```

Επιστρέφεται το κατάστημα από τη βάση

```
// επιστρέφει τα στοιχεία ενός καταστήματος (settings) ώστε να τα διαχειριστεί ο user
[HttpGet("Info/{storeId}")]
1 reference
public StoreDTO Info(int storeId)
{
    // φέρνει το κατάστημα απο τη βάση
    var store = _context.Stores.First(s => s.Id == storeId);
    //δημιουργεί το DTO (data transfer object)
    var storeDto = new StoreDTO
    {
        Id = store.Id,
        Name = store.Name
    };
    //προσθέτει στο DTO τα στοιχεία του καταστήματος
    if (store.AddressId != null)
    {
        var address = _context.Addresses.First(a => a.Id == store.AddressId);
        storeDto.Address = address.Street;
        storeDto.Lat = address.Latitude;
        storeDto.Long = address.Longtitude;
        storeDto.AddressNumber = address.Number;
        storeDto.TK = address.Tk;
        storeDto.City = address.City;
    }

    return storeDto;
}
```

Η *updateInfo* φέρνει το κατάστημα από τη βάση, αν δεν έχει στοιχεία (διεύθυνση) τα ενημερώνει

```
// αποθηκεύει τα στοιχεία του καστήματος
[HttpPost("UpdateInfo")]
0 references
public StoreDTO UpdateInfo(StoreDTO data)
{
    //φέρνει το κατάστημα απο τη βάση
    var store = _context.Stores.First(s => s.Id == data.Id);
    //ενημερώνει τη βάση με αυτό πληκτρολογησε ο χρήστης
    store.Name = data.Name;

    //φέρνει απο τη βάση τη διεύθυνση, αλλιώς (αν δεν εχει) δημιουργεί νέο address
    Addresses address = null;
    if (store.AddressId != null)
    {
        address = _context.Addresses.First(a => a.Id == store.AddressId);
    }
    else
    {
        address = new Addresses();
        address.City = "";
        address.Number = "";
        address.Tk = "";
        var latestAddress = _context.Addresses.OrderByDescending(s => s.Id).FirstOrDefault();
        store.AddressId = latestAddress.Id + 1;
        _context.Addresses.Add(address);
    }

    //γίνεται update η διεύθυνση με τα νέα στοιχεία που στέλνει ο client
    address.Street = data.Address;
    address.Longtitude = data.Long;
    address.Latitude = data.Lat;

    address.Number = data.AddressNumber;
    address.Tk = data.TK;
    address.City = data.City;
    //σώζει στη βάση
    _context.SaveChanges();

    return Info(data.Id);
}
```

Η συνάρτηση *categories/{storeId}* επιστρέφει τις κατηγορίες του συγκεκριμένου καταστήματος

```
//Συνάρτηση που φέρνει τις κατηγορίες του καταστήματος (αυτές που έχει ήδη επιλέξει)
[HttpGet("Categories/{storeId}")]
2 references
public IEnumerable<CategoryDTO> Categories(int storeId)
{
    var storeCategories = _context.StoreToCategories
        .Where(s => s.StoreId == storeId)
        .Select(s => s.CategoryId)
        .ToArray();

    return _context.Categories
        .Where(m => storeCategories.Contains(m.Id))
        .Select(s => new CategoryDTO
        {
            Id = s.Id,
            ParentId = s.ParentId,
            Name = s.Name
        });
}
```

Προσθήκη και αφαίρεση κατηγορίας σε συγκεκριμένο κατάστημα

```
//προσθέτει την επιλεγμένη κατηγορία στο store
[HttpGet("Categories/Add/{categoryId}/{storeId}")]
0 references
public IEnumerable<CategoryDTO> AddCategory(int categoryId, int storeId)
{
    var categoryToAdd = new StoreToCategories();
    categoryToAdd.CategoryId = categoryId;
    categoryToAdd.StoreId = storeId;

    _context.StoreToCategories.Add(categoryToAdd);
    _context.SaveChanges();

    return Categories(storeId);
}
//αφαιρεί την επιλεγμένη κατηγορία απο το κατάστημα
[HttpGet("Categories/Remove/{categoryId}/{storeId}")]
0 references
public IEnumerable<CategoryDTO> RemoveCategory(int categoryId, int storeId)
{
    var catToRemove = _context.StoreToCategories.First(c => c.CategoryId == categoryId && c.StoreId == storeId);

    if (catToRemove != null)
    {
        _context.StoreToCategories.Remove(catToRemove);
        _context.SaveChanges();
    }

    return Categories(storeId);
}
```

Η συνάρτηση *products/{storeId}* επιστρέφει τα προϊόντα του καταστήματος. Γίνεται προσθήκη και ενημέρωση προϊόντος με τις *Products/Add Products/Update* αντίστοιχα

```
//φέρνει από τη βάση τα προϊόντα του συγκεκριμένου καταστήματος
[HttpGet("Products/{storeId}")]
0 references
public IEnumerable<MenuItemDTO> Products(int storeId)
{
    return _context.MenuItems
        .Where(m => m.StoreId == storeId)
        .OrderBy(m => m.Category)
        .Select(s => new MenuItemDTO
        {
            Id = s.Id,
            StoreId = s.StoreId,
            Title = s.Title,
            CategoryId = int.Parse(s.Category),
            Price = s.Price
        });
}
//προσθέτει προϊόν για το συγκεκριμένο κατάστημα
[HttpPost("Products/Add")]
0 references
public int AddProduct(MenuItemDTO product)
{
    var productToAdd = new MenuItems();
    productToAdd.StoreId = product.StoreId;
    productToAdd.Title = product.Title;
    productToAdd.Category = product.CategoryId.ToString();
    productToAdd.Price = product.Price;
    _context.MenuItems.Add(productToAdd);
    _context.SaveChanges();
    return productToAdd.Id;
}
// φέρνει το προϊόν απο τη βάση και το αλλάζει
[HttpPost("Products/Update")]
0 references
public void UpdateProduct(MenuItemDTO product)
{
    var productToUpdate = _context.MenuItems.First(p => p.Id == product.Id);
    productToUpdate.StoreId = product.StoreId;
    productToUpdate.Title = product.Title;
    productToUpdate.Category = product.CategoryId.ToString();
    productToUpdate.Price = product.Price;
    _context.SaveChanges();
}
```

Order Controller

Για τη διαχείριση των παραγγελιών έχει δημιουργηθεί ο order controller που περιέχει τα παρακάτω actions

HasNewOrders

Καλείται το action HasNewOrders κάθε δύο δευτερόλεπτα. Συγκρίνει το πλήθος των pending orders και αν υπάρχει διαφορά, τότε καλείται η GetPendingOrders, η οποία φέρνει τις παραγγελίες

```
[HttpGet("HasNew/{storeId}/{clientCount}")]
0 references
public bool HasNewOrders(int storeId, int clientCount)
{
    //φέρνει απο τη βάση το πλήθος των pending orders του καταστήματος
    var dbOrdersCount = _context.Orders.Where(o => o.Status == 1 && o.StoreId == storeId).Count();
    //αν ειναι διαφορετικές απο αυτές που γνωρίζει ο client, σημαίνει ότι προστέθηκε νέα παραγγελία
    return clientCount != dbOrdersCount;
}

// φέρνει τις pending orders
[HttpGet("Pending/{storeId}")]
0 references
public IEnumerable<OrderDTO> GetPendingOrders(int storeId)
{
    return GetOrders(1, storeId);
}

// φέρνει τις in progress orders
[HttpGet("InProgress/{storeId}")]
0 references
public IEnumerable<OrderDTO> GetInProgressOrders(int storeId)
{
    return GetOrders(2, storeId);
}
```

AcceptOrder

Καλείται από τον client όταν ένα μαγαζί κάνει accept μια παραγγελία. Φέρνει από τη βάση την παραγγελία, σημειώνεται ανάλογα το status (pending, finished), ενημερώνεται ο χρόνος ολοκλήρωσης της παραγγελίας. Στέλνεται στη firebase και ενημερώνεται ο αριθμός της παραγγελίας και ο χρόνος άφιξης.

```
// Controller action που θα καλεστει απο τον client όταν ένα μαγαζί κάνει accept μια παραγγελία
[HttpGet("Accept/{orderId}/{minutes}")]
0 references
public async Task AcceptOrder(int orderId, int minutes)
{
    // φέρνουμε απο την βαση την παραγγελία με το id που ζητήθηκε
    var order = _context.Orders.First(o => o.Id == orderId);
    var firebaseTable = "";
    // ανάλογα το status της παραγγελίας μαρκάρεται είτε ως finished είτε ως pending
    // επίσης βρίσκουμε το table της firebase που πρέπει να ενημερωθεί ώστε να σταλεί αντίστοιχο Notification στον πελάτη
    if (order.Status == 2)
    {
        firebaseTable = "finished_orders";
        order.Status = 3;
    }
    else
    {
        firebaseTable = "pending_orders";
        order.Status = 2;
    }
    //ενημερώνονται τα λεπτά που θα ολοκληρωθεί η παραγγελία και αποθηκεύεται στη βάση δεδομένων
    order.EstimatedMinutes = minutes;
    _context.SaveChanges();

    //Συνδεση με τη firebase
    var firebaseClient = new FirebaseClient(
        "https://smartorder-f831d.firebaseio.com/",
        new FirebaseOptions
        {
            AuthTokenAsyncFactory = () => Task.FromResult("ZyRTXq7TCUVeJvxZw8RYNlgo1qBgk1Q0x9IHz7AM")
        });
    //στέλνεται στη firebase ο αριθμός της παραγγελίας και ο χρόνος άφιξης σε λεπτά
    await firebaseClient
        .Child(firebaseTable)
        .PutAsync("{ \"" + orderId + "\": " + minutes + " }");
```

GetOrders

Επιστρέφει τις παραγγελίες του καταστήματος ανάλογα το status που ζητήθηκε

```
private IEnumerable<OrderDTO> GetOrders(int status, int storeId)
{
    // φέρνει από τη βάση τις παραγγελίες του συγκεκριμένου καταστήματος για το συγκεκριμένο status
    var orders = _context.Orders.Where(o => o.Status == status && o.StoreId == storeId);

    //μετρέπονται τα αντικείμενα της βάσης στα αντικείμενα που περιμένει ο client (orderDTO)
    return orders.Select(o => new OrderDTO
    {
        Id = o.Id,
        DateCreated = o.DateCreated,
        Status = o.Status,
        Store = o.StoreId,
        TotalPrice = o.TotalPrice,
        User = new UserDTO
        {
            Email = o.User.Email,
            Name = o.User.Name,
            Phone = o.User.Phone
        },
        OrderItems = o.OrderItems.Select(i => new OrderItemDTO
        {
            Quantity = i.Quantity,
            Menuitem = new MenuItemDTO
            {
                Id = i.Menuitem.Id,
                Price = i.Menuitem.Price,
                Title = i.Menuitem.Title
            }
        })
    });
```

Frontend

Χρησιμοποιεί το framework React.js για να κάνει bind το μοντέλο με το UI (html). Μέσω ajax request καλεί τα controller actions του backend ώστε να αλληλεπιδράσει με τη βάση δεδομένων (MSSQL)

Index.js

Κάνει render το app component στο root του html

```
JS index.js
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import * as serviceWorker from './serviceWorker';

//Κάνει render το app component στο root του html
ReactDOM.render(<App />, document.getElementById('root'));

serviceWorker.unregister();
```

App.js

Είναι το root component που κάνει render τα σκελετό (πχ μενού) της εφαρμογής και τα εσωτερικά components

```
//φέρνει τα στοιχεία του καταστήματος απο το backend(από το API)
getStoreInfo(storeId) {
  fetch(window.$baseUrl + '/Store/Info/' + storeId)
    .then(res => res.json())
    .then((data) => {
      this.setState({ store: data });
    })
    .catch(console.log);
}
```

Επιστρέφεται το html του συγκεκριμένου component και ζωγραφίζεται το μενού

```
//επιστρέφει το HTML του συγκεκριμένου component
//ζωγραφίζει ένα μενού και στο container κάνει render ένα component με βάση το Url του browser
render() {
  return (
    <div>
      <Router>
        <Navbar bg="dark" variant="dark" style={{ 'margin-bottom': "20px", fontSize:"large" }}>
          <Link className="navbar-brand" to="/">
            <img
              alt="" src="/logo192.png" width="30" height="30" className="d-inline-block align-top"
            />{' '}
            {this.state.isAuthenticated === true ? 'Smart Order: ' + this.state.store?.name : 'Smart Order'}
          </Link>
          {this.getMenu()}
        </Navbar>

        <Container>
          <Route exact path="/" component={Home} />
          <Route exact path="/login" component={Login} />
          <Route exact path="/register" component={Register} />
          <PrivateRoute path="/store" component={Store} />
          <PrivateRoute path="/products" component={Products} />
          <PrivateRoute path="/orders" component={Orders} />
        </Container>
      </Router>
    </div >
  );
}
```

getMenu

Εμφανίζει το κατάλληλο μενού αναλόγως αν έχει γίνει σύνδεση ή όχι

```
//συνάρτηση που φτιάχνει το μενού (ανάλογα με το αν εισαι logged in)
getMenu() {
  if (this.state.isAuthenticated === true) {
    return [
      <Nav className="mr-auto">
        <Link className="nav-link" to="/orders">Orders</Link>
        <Link className="nav-link" to="/products">Pricelist</Link>
      </Nav>,
      <Nav className="pull-right">
        <Link className="nav-link" to="/store">Settings</Link>
        <Link className="nav-link" onClick={() => this.logout()}>Logout</Link>
      </Nav>
    ];
  } else {
    return (
      <Nav className="mr-auto">
        <Link className="nav-link" to="/login">Login</Link>
        <Link className="nav-link" to="/register">Register</Link>
      </Nav>
    );
```

Authentication Service

Αντικείμενο για τη διαχείριση της διαδικασίας του authentication. Χτυπάει το controller action του register για την εγγραφή ενός νέου καταστήματος. Αν ο server δεν επιστρέψει κάτι, σημαίνει ότι το register απέτυχε, αλλιώς αποθηκεύεται το id του καταστήματος σε μια μεταβλητή και στο Local storage ώστε να ειναι διαθεσιμο σε περιπτωση που η σελίδα γίνει refresh

```
//χτυπάει το controller action του register για την εγγραφή ενός νέου καταστήματος
register(data, cb) {
    fetch(window.$baseUrl + '/Store/Register', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(data)
    })
        .then(res => res.text())
        .then((data) => {
            // Αν ο server δεν επιστρέψει κάτι, το register απέτυχε
            if (isNaN(data)) {
                cb(false, data);
            } else { //αλλιώς αποθηκεύεται το id του καταστήματος σε μια μεταβλητή
                //και στο Local storage ώστε να ειναι διαθεσιμο σε περιπτωση που η σελίδα γίνει refresh
                var storeId = parseInt(data);
                AuthService.storeId = storeId;
                localStorage.setItem("storeId", AuthService.storeId);
                AuthService.loginCb(AuthService.storeId);
                cb(true);
            }
        })
        .catch(console.log);
},
```

Χτυπάει το controller action του Authenticate για τον έλεγχο των στοιχείων του χρήστη

```
//χτυπάει το controller action του Authenticate για τον έλεγχος των στοιχείων του χρήστη
authenticate(username, password, cb) {
    fetch(window.$baseUrl + '/Store/Authenticate/' + username + "/" + password)
        .then(res => res.json())
        .then((data) => {
            //Αν ο server επιστρέψει κάτι, ο έλεγχος πέτυχε και επιστρέφονται το id του καταστήματος
            if (data > 0) {
                AuthService.storeId = data;
                localStorage.setItem("storeId", AuthService.storeId);
                AuthService.loginCb(AuthService.storeId);
                cb(true);
            } else { //αλλιώς απέτυχε
                cb(false);
            }
        })
        .catch(console.log);
},
```

Login

Login Αυτόματα

Ελέγχεται αν ο συγκεκριμένος χρήστης υπάρχει ώστε να γίνει αυτόματα Login

```
//καλείται μόλις φορτώσει η σελίδα και ελέγχει αν υπάρχει στο Local storage το id του καταστήματος
// αν υπάρχει καλεί το controller action του Login ώστε να γίνει login αυτόματα ο χρήστης
autoLogin() {
    var data = localStorage.getItem("storeId");
    if (data == null) {
        return;
    }

    var storeId = parseInt(data);
    if (isNaN(storeId)) {
        return;
    }

    AuthService.storeId = storeId;
    AuthService.loginCb(AuthService.storeId);
},
```

Private route

Ελέγχεται αν ο χρήστης είναι συνδεδεμένος, αλλιώς γίνεται ανακατεύθυνση στη login

```
//component το οποίο ελέγχει εάν ο χρήστης είναι συνδεδεμένος
//και σε περίπτωση που δεν είναι κάνει ανακατεύθυνση στη login
const PrivateRoute = ({ component: Component, ...rest }) => (
    <Route {...rest} render={(props) => (
        AuthService.isAuthenticated() === true
        ? <Component {...props} />
        : <Redirect to='/login' />
    )} />
  )

export default PrivateRoute;
```

Store

Είναι ένα component για τη διαχείριση των στοιχείων του καταστήματος (όνομα και διεύθυνση, κατηγορίες προϊόντων και τιμοκατάλογος)

```
//καλεί το backend για να φέρει τα στοιχεία του καταστήματος
loadData() {
    fetch(window.$baseUrl + '/Store/Info/' + this.state.storeId)
        .then(res => res.json())
        .then((data) => {
            this.setState({
                store: {
                    name: data.name,
                    id: data.id,
                    address: data.address,
                    addressNumber: data.addressNumber,
                    tk: data.tk,
                    city: data.city,
                    lat: data.lat,
                    long: data.long
                }
            });
        })
        .catch(console.log);

    fetch(window.$baseUrl + '/Store/Categories/' + this.state.storeId)
        .then(res => res.json())
        .then((data) => {
            this.setState({ categories: [...data] });
            this.fetchAllCategories();
        })
        .catch(console.log);
}
```

Επιστρέφονται όλες τις κατηγορίες προϊόντων

```
//φέρνει όλες τις κατηγορίες προϊόντων
fetchAllCategories() {
    fetch(window.$baseUrl + '/Data/Categories')
        .then(res => res.json())
        .then((data) => {
            var notUsedCategories = data.filter(item => this.state.categories.find(c => c.id == item.id) == null);
            this.setState({ allCategories: [...notUsedCategories] });
        })
        .catch(console.log);
}
```

Διαχείριση των αλλαγών που γίνονται στο UI ενημερώνοντας το μοντέλο

```
// διαχειρίζεται τις αλλαγές που γίνονται στο UI ενημερώνοντας το μοντέλο
handleChange(type, event) {
    var newValue = event.target.value;

    if (type === 'name') {
        this.setState({ store: { ...this.state.store, name: newValue } });
    } else if (type === 'address') {
        this.setState({ store: { ...this.state.store, address: newValue } });
    } else if (type === 'lat') {
        this.setState({ store: { ...this.state.store, lat: newValue } });
    } else if (type === 'long') {
        this.setState({ store: { ...this.state.store, long: newValue } });
    } else if (type === 'category') {
        this.setState({ selectedCategory: newValue });
    } else if (type === 'addressNumber') {
        this.setState({ store: { ...this.state.store, addressNumber: newValue } });
    } else if (type === 'tk') {
        this.setState({ store: { ...this.state.store, tk: newValue } });
    } else if (type === 'city') {
        this.setState({ store: { ...this.state.store, city: newValue } });
    }

    this.setState({ isDirty: true });
}
```

Σώζονται τα στοιχεία του καταστήματος

```
//καλεί το backend για να σώσει τα στοιχεία του καταστήματος
saveName() {
    this.state.store.lat = parseFloat(this.state.store.lat);
    this.state.store.long = parseFloat(this.state.store.long);

    fetch(window.$baseUrl + '/Store/UpdateInfo', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(this.state.store)
    })
        .then(res => res.json())
        .then((data) => {
            this.setState({ isDirty: false });
        })
        .catch(console.log);
}
```

Διαδικασίες για προσθήκη και αφαίρεση κατηγοριών στο κατάστημα

```
addCategory() {
    fetch(window.$baseUrl + '/Store/Categories/Add/' + this.state.selectedCategory + '/' + this.state.storeId)
        .then(res => res.json())
        .then((data) => {
            this.setState({ selectedCategory: null, categories: [...data] });
            this.fetchAllCategories();
        })
        .catch(console.log);
}

removeCategory(catId) {
    fetch(window.$baseUrl + '/Store/Categories/Remove/' + catId + '/' + this.state.storeId)
        .then(res => res.json())
        .then((data) => {
            this.setState({ selectedCategory: null, categories: [...data] });
            this.fetchAllCategories();
        })
        .catch(console.log);
}
```

Order

Επιστρέφει τις παραγγελίες και τις βάζει στο state του component

```
//φέρνει απο τα APIs τις παραγγελίες και τις βάζει στο state του component
getOrders() {
    fetch(window.$baseUrl + '/Order/Pending/' + this.state.storeId)
        .then(res => res.json())
        .then((data) => {
            this.setState({ pendingOrders: [...data] });
            this.startTimer();
        })
        .catch(console.log);

    fetch(window.$baseUrl + '/Order/InProgress/' + this.state.storeId)
        .then(res => res.json())
        .then((data) => {
            this.setState({ inProgressOrders: [...data] });
        })
        .catch(console.log);
}

//εκκινεί έναν timer για να ελέγχει αν υπάρχουν νέες παραγγελίες
startTimer() {
    this.intervalId = setTimeout(() => {
        this.checkForNew();
    }, 2000);
}
```

Αν υπάρχει νέα παραγγελία δείχνει τα Notifications και ξαναζητάει τις παραγγελίες, ενώ αν δεν έχει έρθει παραγγελία ενεργοποιεί ξανά τον timer

```
//συνάρτηση που ρωτάει το API αν υπάρχει νέα παραγγελία
//αν υπάρχει δείχνει notification και ξαναζητάει τις παραγγελίες
//αν δεν εχει έρθει παραγγελία, ενεργοποιεί ξανά τον timer
checkForNew() {
    fetch(window.$baseUrl + '/Order/HasNew/' + this.state.storeId + "/" + this.state.pendingOrders.length)
        .then(res => res.json())
        .then((data) => {
            if (data === true) {
                this.n.show();
                this.getOrders();
            } else {
                this.startTimer();
            }
        })
        .catch(console.log);
}
```

ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΙ ΜΕΛΛΟΝΤΙΚΕΣ ΕΠΕΚΤΑΣΕΙΣ

Η δημιουργία εφαρμογών- συστημάτων που είναι εξειδικευμένα σε τομείς που δεν είμαστε συνηθισμένοι να χρησιμοποιούμε στην καθημερινότητά μας είναι δύσκολη διαδικασία. Είναι πολύτιμη η συμβολή των ειδικών που έχουν άμεση επικοινωνία με όσους χρήζουν της συγκεκριμένης ανάγκης στην οποία στοχεύει η εκάστοτε εφαρμογή - σύστημα ή ακόμη και η άμεση επαφή με τους ίδιους τους ανθρώπους στους οποίους στοχεύει η εφαρμογή. Δεν αρκούν μόνο υποθέσεις για να υλοποιηθούν τέτοιου είδους συστήματα. Χρειάζεται ειδική διαχείριση και μεγάλη έρευνα.

Θα μπορούσαν να προστεθούν στην android εφαρμογή περισσότερες λειτουργίες φωνητικής διαχείρισης (speech to text & text to speech) ώστε να είναι ακόμη λιγότερες οι κινήσεις που έχει να πραγματοποιήσει ο πελάτης. Δηλαδή, η παραγγελία να γίνεται κατόπιν φωνητικών εντολών του χρήστη. Επιπλέον, θα μπορούσε το σύστημα να επεκταθεί πέρα από τα καταστήματα φαγητών και καφέ αλλά και σε παραγγελίες προϊόντων όπως για παράδειγμα καταστήματα ένδυσης, υπόδησης, supermarket κά. Ακόμη, θα μπορούσε να ενταχθεί - να ενσωματωθεί σε ήδη υπάρχουσες γνωστές εφαρμογές παραγγελιοληψίας, να είναι δηλαδή ως μια επιλογή στο μενού της εφαρμογής για όσους το επιθυμούν. Το Live tracking θα βοηθούσε τον χρήστη ώστε να βλέπει ακριβώς που βρίσκεται η παραγγελία του. Επίσης, η ηλεκτρονική πληρωμή είναι μια επέκταση που θα βοηθούσε όλους τους χρήστες και των δύο εφαρμογών. Τέλος, την εφαρμογή του καταστήματος θα μπορούσε να προστεθεί μια ενότητα με τα στατιστικά στοιχεία, ώστε να γνωρίζει ανά μήνα και έτος ο ιδιοκτήτης του καταστήματος ποια προϊόντα πουλάει περισσότερο, ώστε να βελτιστοποιήσει το κέρδος του και τη διαδικασία προετοιμασίας των παραγγελιών.

ΒΙΒΛΙΟΓΡΑΦΙΑ

Assistive Touch for Android. Διαθέσιμο στο:
https://play.google.com/store/apps/details?id=com.easytouch.assistivetouch&hl=en_US
Τελευταία ανασκόπηση 03.07.2020

Disabled world. Διαθέσιμο στο:
https://www.disabled-world.com/
Τελευταία ανασκόπηση 13.07.2020

syros_accesspoint. Διαθέσιμο στο:
https://play.google.com/store/apps/details?fbclid=IwAR05eRXLIdLFrXaOnOArVexc5ZnYmpp770qV6_dWsLQ3UryinjxZxsECU9Q&id=com.janrus27.syros_accesspoint
Τελευταία ανασκόπηση 03.07.2020

SwiftKey. Διαθέσιμο στο:
https://play.google.com/store/apps/details?id=com.touchtype.swiftkey&referrer=utm_source%3Dwebsite%26utm_medium%3Dsk%26utm_campaign%3Dfooter-menu
Τελευταία ανασκόπηση 03.07.2020

wheelmap. Διαθέσιμο στο:
https://play.google.com/store/apps/details?id=org.wheelmap.android.online&hl=en
Τελευταία ανασκόπηση 03.07.2020

wheelmate. Διαθέσιμο στο:
https://play.google.com/store/apps/details?id=com.novasa.wheelmate&hl=en_GB
Τελευταία ανασκόπηση 03.07.2020

worldbank. Διαθέσιμο στο:
https://www.worldbank.org/en/topic/disability
Τελευταία ανασκόπηση 03.07.2020