ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Community Pharmacies

"Διαδικτυακή εφαρμογή για τη διαχείριση φαρμάκων σε κοινωνικά φαρμακεία"

Του φοιτητή

Παπαμαυρουδή Μιλτιάδη

Αρ. Μητρώου: 164723

Επιβλέπων

Ουγιάρογλου Στέφανος,
Επίκουρος Καθηγητής

Θεσσαλονίκη 2023

Τίτλος Π.Ε: "Community Pharmacies,Διαδικτυακή εφαρμογή για τη διαχείριση φαρμάκων σε κοινωνικά φαρμακεία"

Κωδικός Π.Ε: 23278

Ονοματεπώνυμο φοιτητή/τών: Παπαμαυρουδής Μιλτιάδης

Ονοματεπώνυμο εισηγητή: Παπαμαυρουδής Μιλτιάδης

Ημερομηνία ανάληψης Π.Ε: 18-10-2023

Ημερομηνία περάτωσης Π.Ε. …

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

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

Πρόλογος

Η παρούσα πτυχιακή εργασία συντάχθηκε από τον φοιτητή Παπαμαυρουδή Μιλτιάδη υπό την εποπτεία του κ. Ουγιάρογλου Στέφανου. Αφορά την δημιουργία μιας διαδικτυακής εφαρμογής, η οποία επιτρέπει του συνανθρώπους μας που αντιμετωπίζουν οικονομικά προβλήματα να μπορούν να εντοπίζουν κοινωνικά φαρμακεία και έπειτα να έχουν την δυνατότητα κράτησης. Η παρούσα εφαρμογή περιλαμβάνει μια ιστοσελίδα όπου οι χρήστες μπορούν να εντοπίζουν τα κοινωνικά φαρμακεία με χρήση χάρτη, αναζήτησης κοινωνικών φαρμακείων ή προϊόντων. Η διαδικτυακή εφαρμογή έχει υλοποιηθεί κυρίως με το MERN stack που περιλαμβάνει τέσσερις τεχνολογίες MongoDB, ExpressJs, React NodeJs. Στόχος της εργασίας μας, είναι τα κοινωνικά φαρμακεία να γίνουν γνωστά στο ευρύ κοινό με περισσότερες επιλογές απο τις υπάρχων σελίδες οι οποίες δεν εξυπηρετούν πλήρως τις ανάγκες των συμπολιτών μας. Τέλος, ελπίζουμε ότι θα αυξηθεί η ευαισθητοποίηση σε όσους έχουν φάρμακα στο σπίτι τους και δεν τα χρειάζονται.

Περίληψη Πτυχιακής

Πολλοί συμπολίτες μας που είτε είναι ανασφάλιστοι είτε ανήκουν σε ευάλωτες ομάδες αδυνατούν να αγοράσουν από το φαρμακείο τα απαραίτητα φάρμακά τους. Τα κοινωνικά φαρμακεία, τα οποία λειτουργούν στο πλαίσιο δήμων, μη κερδοσκοπικών οργανώσεων, σωματείων, ιδρυμάτων και άλλων φορέων κοινωνικής αλληλεγγύης, συμβάλλουν στη δημιουργία ενός αυτοτροφοδοτούμενο συστήματος δωρεάς και αναδιανομής περισσευούμενων φαρμάκων προσφέροντας δωρεάν φάρμακα. Ωστόσο, μία από τις μεγαλύτερες αδυναμίες των κοινωνικών φαρμακείων είναι ότι δεν διαθέτουν μεγάλη ποικιλία σε φάρμακα και παράλληλα δεν είναι γνωστά στο ευρύ κοινό και η παρουσία τους στο διαδίκτυο είναι εξαιρετικά περιορισμένη. Στα πλαίσια της παρούσας πτυχιακής εργασίας πρόκειται να αναπτυχθεί μια full stack εφαρμογή ιστού με όνομα "Community Pharmacies" η όποια θα επιχειρήσει να λύσει τα παραπάνω προβλήματα. Μέσω του Community Pharmacies θα γίνονται γνωστά τα κοινωνικά φαρμακεία μαζί με τα διαθέσιμα φάρμακα που έχει καθένα από αυτά. Η εφαρμογή θα δίνει τη δυνατότητα σε οποιονδήποτε διαχειριστή κοινωνικού φαρμακείου να καταχωρεί στην εφαρμογή τα στοιχεία του κοινωνικού φαρμακείου και τα φάρμακα τα οποία διαθέτει κάθε χρονική στιγμή. Από την άλλη, οι ενδιαφερόμενοι θα μπορούν να δουν με τη χρήση ενός χάρτη τα διαθέσιμα κοινωνικά φαρμακεία, να αναζητήσουν φάρμακα που χρειάζεται, εφόσον τα βρουν, να καταχωρήσουν παραγγελία/κράτηση και να παραλάβουν τα φάρμακα από το φυσικό κατάστημα.

"Community Pharmacies: Online Application for Medication Management in Social Pharmacies"

<<Miltiadis Papamavroudis>>

Abstract

Many of our fellow citizens, whether uninsured or belonging to vulnerable groups, struggle to purchase their necessary medications from pharmacies. Social pharmacies, operating within the framework of municipalities, non-profit organizations, associations, institutions, and other social solidarity entities, contribute to the creation of a feedback-driven system of donation and redistribution of surplus medications by offering free drugs. However, one of the major weaknesses of social pharmacies is that they lack a wide variety of medications, and at the same time, they are not widely known to the public, with their online presence being extremely limited.As part of this thesis project, a full-stack web application named "Community Pharmacies" will be developed to address the above issues. Through Community Pharmacies, social pharmacies and the available medications at each of them will be made known. The application will allow any administrator of a social pharmacy to enter the details of the social pharmacy and the available medications at any given time into the application.On the other hand, users will be able to view, using a map, the available social pharmacies, search for needed medications, place an order/reservation if they find what they need, and pick up the medications from the physical store.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον κ. Στέφανο Ουγιάρογλου, Επίκουρο καθηγητή του τμήματος Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του Διεθνές Πανεπιστημίου της Ελλάδος, ο οποίος είναι ο επιβλέπων καθηγητής της συγκεκριμένης πτυχιακής εργασίας.

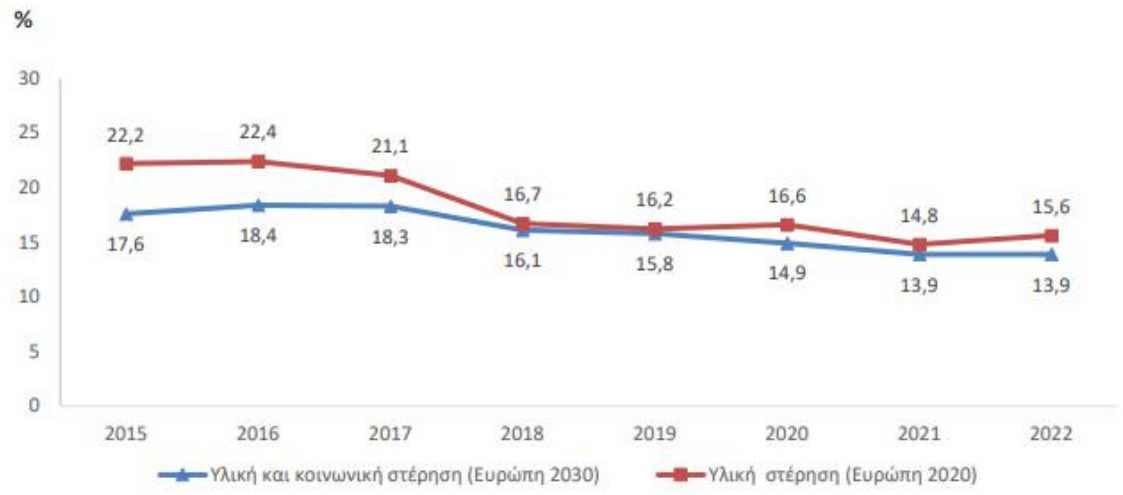
Περιεχόμενα

Κατάλογος Σχημάτων

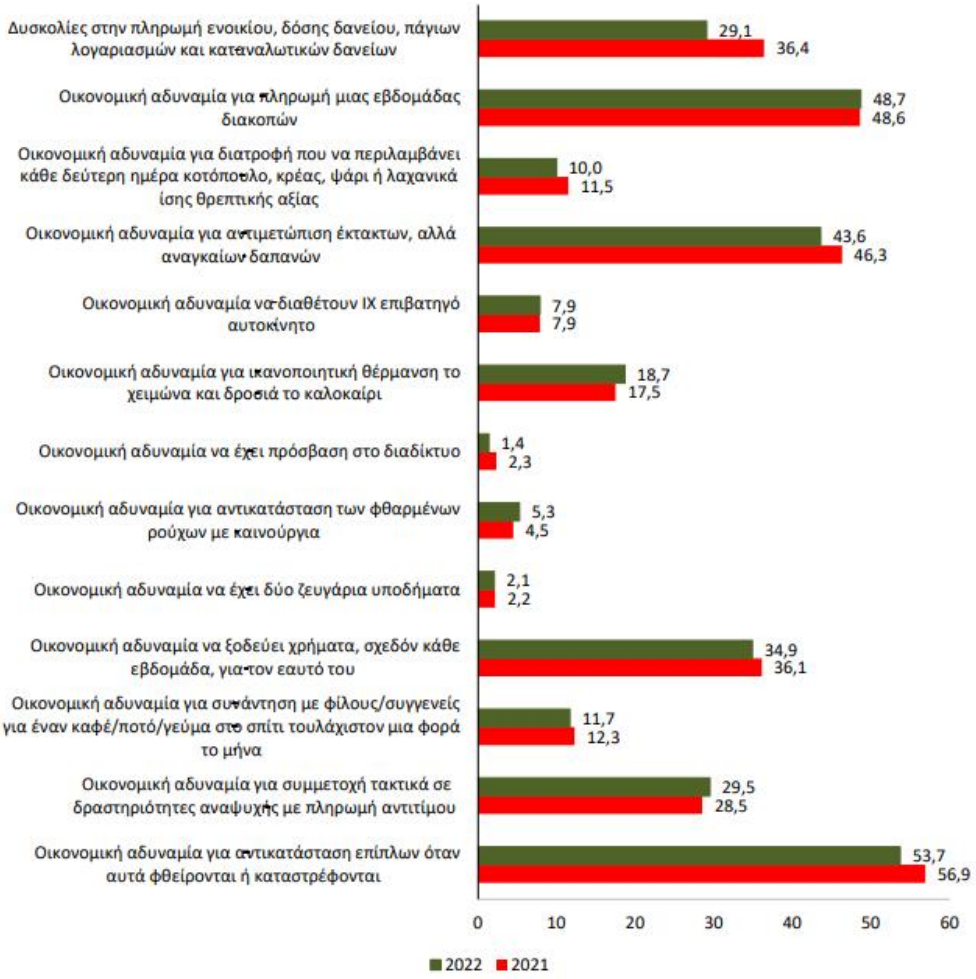
Κεφάλαιο 1ο: Εισαγωγή

1.1 Η ανάγκη των συμπολιτών μας

Σύμφωνα με έρευνα τής ελληνικής αρχής (ΕΛΣΤΑΤ) [1] για το έτος 2021 αναφέρει το 19,6% του ελληνικού πληθυσμού βρίσκεται πολύ πιθανόν σε κίνδυνο φτώχειας και το 13,9% βρίσκονται σε υλική και κοινωνική στέρηση. Οι συνάνθρωποι μας, δυσκολεύονται να καλύψουν βασικές ανάγκες τους, όπως να αγοράσουν τα φάρμακα που χρειάζονται.



Σχήμα 1.1 Ποσοστιαία κατανομή πληθυσμού με υλικές και κοινωνικές στερήσεις και υλικές στερήσεις : 2015 – 2022.



Σχήμα 1.2: Ποσοστιαία κατανομή πληθυσμού με υλικές και κοινωνικές στερήσεις, ανά συνιστώσα στέρησης: 2021 – 2022.

Σύμφωνα με τον ν.4368/2016 και την ΚΥΑ Α3(γ)/ΓΠ/οικ.25132/4-4-2016, θεσπίστηκε [2] για πρώτη φορά το δικαίωμα ελεύθερης πρόσβασης σε όλες τις δημόσιες δομές υγείας για την παροχή νοσηλευτικής και ιατροφαρμακευτικής περίθαλψης σε ανασφάλιστους και σε ευάλωτες κοινωνικές ομάδες.

Συγκεκριμένα τα πλέον πλεονεκτήματα είναι τα εξής:

- Ισοστάθμιση των δικαιωμάτων ασφαλισμένων και μη ασφαλισμένων συνανθρώπων μας.
- Ελεύθερη και δωρεάν πρόσβαση σε πρωτοβάθμιες και δευτεροβάθμιες δημόσιες δομές υγείας, μονάδες ψυχικής υγείας, δομές απεξάρτησης και πανεπιστημιακά νοσοκομεία.
- Πρόληψη και προαγωγή υγείας (π.χ. εμβολιασμοί) χωρίς καμία χρέωση.
- Δωρεάν οδοντιατρική περίθαλψη.
- Δωρεάν παροχές μαιευτικής περίθαλψης και προγραμματισμού τοκετών από τα Δημόσια Νοσοκομεία.
- Χορήγηση φαρμακευτικής αγωγής από τα ιδιωτικά και τα δημόσια φαρμακεία. Μέρος του ανασφάλιστου πληθυσμού δικαιούται μηδενική συμμετοχή στη φαρμακευτική αγωγή.

Παρόλο με τα παραπάνω πλεονεκτήματα που αναφέρουμε, εξακολουθούν τα θέματα της περίθαλψης να μην είναι αρκετά για να καλύψουν τις ανάγκες των συμπολιτών μας. Οι συνάνθρωποι μας που βρίσκονται σε κίνδυνο φτώχειας, μπορεί να έχουν πρόσβαση σε δημόσιες δομές υγείας και να παίρνουν την ανάλογη δωρεάν συνταγογράφηση για το φάρμακο που χρειάζονται, αλλά επειδή η ανάγκη τους δεν αναγνωρίζεται το ίδιο σημαντική σε σύγκριση με τους καρκινοπαθείς ή τους διαβητικούς, πολλές φορές υποχρεούνται να πληρώσουν μεγάλο ποσοστό συμμετοχής ή ολόκληρο το ποσό και σε πολλές περιπτώσεις αδυνατούν.

Κάποιες από τις παραπάνω ευπαθείς ομάδες είναι οι εξής:

- Παιδιά και βρέφη που χρειάζονται ειδικά σκευάσματα όπως ειδικό γάλα, πρωτόγαλα, βιταμίνες, Ω3 κτλ και πολλές φορές δεν συνταγογραφούνται.
- Ζεύγος συνταξιούχων με ένα εισόδημα που βρίσκονται σε κατάσταση φτώχειας και δυσκολεύονται ακόμα περισσότερο να καλύψουν τις βασικές ανάγκες υγείας τους όπως φάρμακα, καθετήρες, υλικά κατακλίσεων κτλ.
- Συνάνθρωποι μας με ποσοστό αναπηρίας που δεν αναγνωρίζεται από το κράτος με αποτέλεσμα να χρειαστεί να πληρώσουν ολόκληρη την τιμή του φαρμάκου.
- Μετανάστες με άδεια παραμονής η οποία έληξε και μέχρι να γίνουν αναγκαίες ενέργειες ανανέωσης της άδειας τους, δεν έχουν το δικαίωμα της δωρεάν περίθαλψης.

1.2 Κοινωνικά φαρμακεία

Σύμφωνα με μια έρευνα της ιστοσελίδας Givmed [3], την οποία θα το εξηγήσουμε στο υποκεφάλαιο 1.3 και το υπουργείου υγείας ανέφερε πως τριάντα τέσσερα εκατομμύρια συσκευασίες φαρμάκων ληγμένων ή ελάχιστα χρησιμοποιημένων καταλήγουν στα απορρίμματα που αυτό κοστολογείται περίπου στο ένα δισεκατομμύριο ευρώ. Ωστόσο, βρέθηκε μια λύση τα τελευταία χρόνια όπου μπορεί να αντιμετωπίσει το παραπάνω φαινόμενο. Με την δημιουργία κοινωνικών φαρμακείων.

Τα κοινωνικά φαρμακεία δημιουργήθηκαν για να διευκολύνουν την πρόσβαση των ευπαθών ομάδων στα φάρμακα που έχουν ανάγκη. Τα φάρμακα, υγειονομικό υλικό και παραφαρμακευτικά προϊόντα δίνονται δωρεάν και συνήθως, κάθε δήμος έχει ένα κατάστημα ή ένα μέρος μιας δημόσιας υπηρεσίας που λειτουργεί ως κοινωνικό φαρμακείο στα πλαίσια μη κερδοσκοπικών οργανώσεων για την εξυπηρέτηση των πολιτών.

Τα κοινωνικά φαρμακεία ανατροφοδοτούνται με διάφορους τρόπους, οι πιο γνωστοί είναι οι εξής:

- Συμπολίτες που έχουν αποθηκεύσει φάρμακα στα σπίτια τους και δεν τα χρειάζονται, ενεργούν ως ενεργοί συνεργάτες παραδίδοντας τα αχρησιμοποίητα φάρμακα στα τοπικά κοινωνικά φαρμακεία.
- Μη κερδοσκοπικές οργανώσεις αναλαμβάνουν τη συγκέντρωση φαρμάκων από διάφορες εκδηλώσεις.
- Τα τοπικά κοινωνικά φαρμακεία λαμβάνουν μικρό ποσοστό στήριξης μέσω προγραμμάτων που παρέχονται από τις δημοτικές αρχές.

1.3 Κίνητρο και Συνεισφορά της εργασίας

Τα κοινωνικά φαρμακεία αντιπροσωπεύουν έναν μηχανισμό αλληλεγγύης, ανταποκρινόμενα στις ανάγκες των ευάλωτων στην κοινωνία. Παρόλα αυτά, δυστυχώς, συχνά παραμένουν εκτός εύρους για το κοινό. Με μια απλή αναζήτηση στο διαδίκτυο για το "κοινωνικό φαρμακείο Ευόσμου", ενδέχεται να εντοπιστεί επιτυχώς το συγκεκριμένο κοινωνικό φαρμακείο. Παρά ταύτα, η παρεχόμενη πληροφορία παραμένει ελάχιστη, περιοριζόμενη συχνά στη μόνη αναφορά της τοποθεσίας, ενώ η διαθεσιμότητα των προϊόντων παραμένει ασαφής.

Μία ιστοσελίδα που προσπαθεί να ανταποκριθεί σε ένα ελάχιστο ποσοστό την ανάγκη των συμπολιτών μας είναι η ιστοσελίδα givmed [3]. Η συγκεκριμένη ιστοσελίδα ομαδοποιεί όλα τα κοινωνικά φαρμακεία παρέχοντας μόνο τις βασικές δυνατότητες όπως προβολή φαρμακείου με την εμφάνιση της ονομασίας της οδού και μερικές φορές τα φάρμακα που έχουν.

Όπως προαναφέραμε, υπάρχουν ιστοσελίδες που εξυπηρετούν κατά κύριο λόγο τους συνανθρώπους μας, αλλά όχι με τον αποτελεσματικότερο τρόπο. Πολλές φορές, οι συνάνθρωποι ενδέχεται να εντοπίσουν ένα φάρμακο σε ένα κοινωνικό φαρμακείο, το οποίο δεν είναι διαθέσιμο την επόμενη μέρα και μερικές φορές η αναζήτηση του κοινωνικού φαρμακείου είναι περίπλοκη διότι δεν υπάρχει στον χάρτη. Αυτός είναι ο λόγος που μας έδωσε το κίνητρο για την υλοποίηση της εφαρμογής.

Η Συνεισφορά της παρούσας διπλωματικής εργασίας είναι η ανάπτυξη μιας διαδικτυακής εφαρμογής προκειμένου να εντοπίζουν οι συμπολίτες μας τα κοινωνικά φαρμακεία εύκολα με χρήση χάρτη και αναζήτησης. Οι συμπολίτες μας θα ενεργούν ως χρήστες/πελάτες της εφαρμογής δημιουργώντας λογαριασμό για να μπορούν να έχουν την δυνατότητας κράτησης ή απλώς ως επισκέπτες για να βλέπουν τα διαθέσιμα φάρμακα. Τέλος, ένας υπεύθυνος κάθε κοινωνικού φαρμακείου θα δημιουργεί έναν λογαριασμό βάζοντας κάποια στοιχεία και έπειτα θα μπορεί να βάζει τα διαθέσιμα φάρμακα ως απόθεμα και να διαχειρίζεται τις ενεργές παραγγελίες.

1.4 Οργάνωση της εργασίας

Στο επόμενο κεφάλαιο θα αναφέρουμε όλες τις τεχνολογίες που έχουν χρησιμοποιηθεί για την υλοποίηση της εφαρμογής αρχίζοντας από το πρόγραμμα συγγραφής και επεξεργασίας κώδικα. Έπειτα, θα αναφέρουμε τις βασικότερες τεχνολογίες για την υλοποίησης της εφαρμογής το MERN και αμέσως μετά τις βιβλιοθήκες που χρησιμοποιήθηκαν.

Στο τρίτο κεφάλαιο, θα αρχίσουμε με κάποια εξήγηση με τις λειτουργικές απαιτήσεις της εφαρμογής μέσω user stories με χρήση διαγραμμάτων. Έπειτα θα προχωρήσουμε στην αρχιτεκτονική του frontend για την παραλαβή προβολή και επεξεργασία δεδομένων και στην εξήγηση της αρχιτεκτονικής του backend για την δημιουργία των API’s, Endpoints και στην επικοινωνία της βάσης δεδομένων. Στο τέλος του τρίτου κεφαλαίου θα αναφερθούμε στην υλοποίησης της εφαρμογής στο frontend και αντίστοιχα στο backend.

Στο τέταρτο κεφάλαιο θα παρουσιάσουμε την λειτουργία της εφαρμογής. Θα κάνουμε δυο σενάρια για να δείξουμε με ποιο τρόπο λειτουργεί η εφαρμογή από την πλευρά του πελάτη και αντίστοιχα για τον διαχειριστή του κοινωνικού φαρμακείου. Θα αναφέρουμε όλα τα απαραίτητα βήματα που πρέπει να γίνουν από την αρχή ως το τέλος.

Τέλος, στο πέμπτο κεφάλαιο, θα παρουσιάσουμε ορισμένα συμπεράσματα που προκύπτουν από την εκπόνηση της πτυχιακής εργασίας. Επιπλέον, θα εξετάσουμε πιθανές επεκτάσεις της εφαρμογής για να επιτύχει μεγαλύτερη ευχρηστία και λειτουργικότητα στο μέλλον.

Κεφάλαιο 2ο: Τεχνολογίες

Στα πλαίσια αυτού του κεφαλαίου θα αναφέρουμε αναλυτικά τις τεχνολογίες που χρησιμοποιήθηκαν για την υλοποίηση της διαδικτυακής εφαρμογής. Ξεκινώντας με τα εργαλεία και έπειτα τις βιβλιοθήκες και τα frameworks.

2.1 Περιβάλλον VS Code

Το Visual Studio Code (VS Code) είναι ένα δωρεάν, ανοιχτού κώδικα περιβάλλον ανάπτυξης που αναπτύχθηκε από τη Microsoft. Είναι ένα πολυλειτουργικό και ελαφρύ περιβάλλον που παρέχει πολλά χαρακτηριστικά για την ανάπτυξη λογισμικού. Τέσσερα από το πιο χαρακτηριστικά πλεονεκτήματα και διαφέρει από τα υπόλοιπα περιβάλλοντα είναι:

- Παρέχει ισχυρό εργαλείο επεξεργασίας κειμένου με υποστήριξη για πολλές γλώσσες προγραμματισμού.
- Υποστηρίζει την ολοκλήρωση με συστήματα ελέγχου έκδοσης όπως το Git, παρέχοντας εύκολο τρόπο για τη διαχείριση και αποθήκευση του κώδικα.
- Υποστηρίζει πολλές Επεκτάσεις (extensions) από την κοινότητα.
- Διαθέτει μια ενεργή και μεγάλη κοινότητα χρηστών και προγραμματιστών.

2.2 HTML, CSS και JavaScript

Τα HTML [4], CSS [5] και JavaScript αποτελούν το θεμέλιο για την ανάπτυξη ιστοσελίδων και web εφαρμογών. Κάθε μία από αυτές τις τεχνολογίες επιτελεί διαφορετικό ρόλο. Η HTML είναι η βασική γλώσσα σήμανσης που χρησιμοποιείται για τη δομή και την οργάνωση του περιεχομένου μιας ιστοσελίδας. Χρησιμοποιεί στοιχεία όπως τίτλους, παραγράφους, εικόνες, συνδέσμους και άλλα για να καθορίσει τη δομή της σελίδας. Η CSS χρησιμοποιείται για την εμφάνισης και μορφοποίησης ιστοσελίδων και συνήθως βρίσκεται σε διαφορετικό κώδικα από την HTML. Τέλος, η JavaScript είναι μια γλώσσα προγραμματισμού που χρησιμοποιείται για την προσθήκη δυναμικής λειτουργικότητας στις ιστοσελίδες. Χρησιμοποιείται για τη διαχείριση συμβάντων, τον έλεγχο του DOM (Document Object Model), και την αλληλεπίδραση με τον χρήστη.

2.3 MERN Stack

Το MERN Stack [6] είναι ένα σύνολο τεχνολογιών που χρησιμοποιούνται για την ανάπτυξη full-stack εφαρμογών. Το MERN αντιπροσωπεύει τα πρώτα γράμματα των βασικών τεχνολογιών MongoDB, ExpressJS, React και NodeJs. Ο λόγος που επιλέχθηκε για την συγκεκριμένη εργασία είναι οι εξής λόγοι. Προσφέρει ολοκληρωμένο περιβάλλον ανάπτυξης που καλύπτει την πλευρά του πελάτη (React), τον εξυπηρετητή (NodeJs και ExpressJs) και τη βάση δεδομένων (MongoDB). Χρησιμοποιεί την JavaScript για όλη την εφαρμογή και την MongoDB ως μια NoSQL βάση όπου την καθιστά πιο εύκολη ως προς την αποθήκευση και διαχείριση των δεδομένων.

2.3.1 React

Είναι μια βιβλιοθήκη ανοικτού κώδικα που δημιουργήθηκε στις 29 Μαΐου του 2013 από την Meta [7] για την αποδοτικότερη και ευέλικτη ανάπτυξη ιστοσελίδων. Το κύριο χαρακτηριστικό της, η ανάπτυξη εφαρμογών μιας σελίδας (Single Page Applications - SPAs). Ο χρήστης πλέον μπορεί να αλληλεπιδρά με την εφαρμογή χωρίς να περιμένει την νέα σελίδα από τον εξυπηρετητή. Επιπλέον, η React έρχεται με μια νέα σύνταξη γλώσσα ονομάζοντας την ως "jsx". Η jsx συνδυάζει τη σύνταξη της JavaScript με στοιχεία που μοιάζουν με HTML, καθιστώντας τον κώδικα πιο ευανάγνωστο, ευέλικτο και κάποιες φορές επαναχρησιμοποιήσιμο, δηλαδή να δημιουργήσεις ένα αρχείο jsx και να το χρησιμοποιήσεις σε δύο και παραπάνω σελίδες. Αρκετές φορές αυτό το αρχείο jsx είναι ένα απομονωμένο, αυτόνομο, και επαναχρησιμοποιήσιμο κομμάτι κώδικα ονομάζεται ως component.

Επιπλέον βασικό χαρακτηριστικό της React είναι το state. Το "state" αναφέρεται στο εσωτερικό δεδομένο που διαχειρίζεται και ελέγχεται από ένα component. Χρησιμοποιούνται στην React για τη διατήρηση της κατάστασης της εφαρμογής και την ανανέωση του UI (User Interface) ανάλογα με αλλαγές στα δεδομένα. Περιέχει τις μεταβλητές και τις τιμές που πρέπει να διατηρηθούν και να ελεγχθούν από το React component. Όταν το state ενημερώνεται, το component βλέπει αυτήν την αλλαγή και επανασχεδιάζει το UI αναλόγως, δημιουργώντας μια αίσθηση δυναμικότητας στην εφαρμογή.

Μία αλλαγή που συναντάμε στην react είναι το Virtual DOM και θεωρείτε ως αντίγραφο του πραγματικού DOM. Χρησιμοποιείτε από την react για την βελτίωση της απόδοση και αποτελεσματικότερη ενημέρωσης της σελίδας, κρίνει και επεμβαίνει όταν είναι αναγκαίο για την ενημέρωση του πραγματικού DOM.

Για παράδειγμα, όταν ο χρήστης αλληλεπιδρά με τη σελίδα π.χ, κάνοντας κλικ σε ένα κουμπί ή συμπληρώνει ένα πεδίο εισαγωγής, αυτό προκαλεί αλλαγές στο πραγματικό DOM. Στην React, δημιουργεί ένα Virtual DOM το οποίο παρακολουθεί της αλλαγές σε ένα αντίγραφο του DOM και όταν ανιχνεύσει κάποια νέα αλλαγή τότε υπολογίζει τις ελάχιστες αλλαγές που απαιτούνται για την ενημέρωση του DOM και στην συνέχεια το εφαρμόζει στο πραγματικό DOM.

Παρακάτω φαίνεται η σύνταξη της React.

```
const MyPharmacy = () => {
  const pharmacyData = useSelector((state) => state.session.pharmacyData) || {};
  const [pharmacyProfileData,setPharmacyProfileData]=useState({})
  useEffect(() => {
    const fetchData = async () => {
     const data = await getPharmacyData(pharmacyData.pharmacy._id);
     setPharmacyProfileData(data)
    };
    if (pharmacyData.pharmacy) {
      fetchData();
```

```
    }
  }, [pharmacyData]);
  return (
    <div className="m-20">
      <Header title="My Pharmacy" />
      <div>
        <div className="rounded-lg p-3 justify-center">
        {pharmacyProfileData.images && pharmacyProfileData.images[0] && (
            <img className="w-72" src={pharmacyProfileData.images[0].url} alt="" />
          )}
        </div>
            <MyPharmacyDetailsTable pharmacyProfileData={pharmacyProfileData} />
        {/* <MyprofileDetailsTable userProfileData={userProfileData} /> */}
      </div>
    </div>
  );
};
```

2.3.2 MongoDB

Η MongoDB [8] είναι ένα σύστημα διαχείρισης βάσεων δεδομένων (DBMS) που ανήκει στην κατηγορία των NoSQL (Not Only SQL) βάσεων δεδομένων. Η βασική ιδέα πίσω από τη MongoDB είναι η αποθήκευση και την επεξεργασία δεδομένων σε μορφή JSON-ειδών καταγραφών, που ονομάζονται εγγραφές ή έγγραφα (documents). Επιπλέον, χρησιμοποιεί ένα μοντέλο αποθήκευσης δεδομένων που ονομάζεται BSON (Binary JSON), το οποίο είναι μια δυαδική μορφή του JSON.

Ο όρος NoSQL [9] αναφέρεται σε μια κατηγορία βάσεων δεδομένων που διαφέρει από τις παραδοσιακές σχεσιακές βάσεις δεδομένων SQL. Οι βάσεις δεδομένων NoSQL σχεδιάστηκαν για να ανταποκριθούν σε συγκεκριμένες ανάγκες και περιπτώσεις χρήσης, όπως η αποθήκευση μεγάλου όγκου δεδομένων, η αναγκαιότητα γρήγορης ανάκτησης δεδομένων ή η δυνατότητα ευελιξίας στο σχήμα των δεδομένων.

Οι βασικές διαφορές μεταξύ SQL και NoSQL είναι οι εξής. Σχετικά με το σχήμα δεδομένων, σε μια σχεσιακή βάση δεδομένων, ακολουθείται μια προκαθορισμένη δομή με πίνακες, στήλες και γραμμές. Αντίθετα, σε μια μη σχεσιακή βάση δεδομένων, δεν υπάρχει προκαθορισμένο σχήμα και μπορεί να αλλάξει δυναμικά ανάλογα με τα δεδομένα που αποθηκεύονται. Όσον αφορά τη γλώσσα ερωτημάτων, η SQL χρησιμοποιείται για την εκτέλεση ερωτημάτων SELECT, UPDATE, DELETE κτλ, ενώ η NoSQL χρησιμοποιεί διαφορετικές γλώσσες ερωτημάτων ανάλογα με τη βάση δεδομένων. Τέλος, όσον αφορά την κλιμακωσιμότητα, η SQL χρησιμοποιείται κυρίως για οριζόντια κλιμακωσιμότητα (sharding) για μεγάλους όγκους δεδομένων, ενώ η NoSQL μπορεί να υποστηρίξει τόσο οριζόντια όσο και κατακόρυφη κλιμακωσιμότητα.

2.3.3 NodeJS

Το Node.js [10] είναι ένα περιβάλλον εκτέλεσης (runtime) που αναπτύχθηκε από τον Ryan Dahl. Επιτρέπει στους προγραμματιστές να χρησιμοποιούν JavaScript κώδικα στην πλευρά του διακομιστή (server-side), προσφέροντας υψηλή απόδοση. Τα βασικά χαρακτηριστικά του περιλαμβάνουν την

ασύγχρονη εκτέλεση (Asynchronous Execution), η οποία υποστηρίζει ασύγχρονο προγραμματισμό για αποτελεσματική διαχείριση ι/ο και εργασιών χωρίς αναμονή. Επιπλέον, προσφέρει ευελιξία για την ανάπτυξη διακομιστών, εφαρμογών εντοπισμού σφαλμάτων, API, real-time εφαρμογών και άλλων, ενώ χρησιμοποιεί τη γλώσσα προγραμματισμού JavaScript τόσο για το frontend όσο και για το backend. Επιπλέον, είναι κατάλληλο για εφαρμογές που απαιτούν επιδόσεις σε πραγματικό χρόνο και διαθέτει τη δυνατότητα εγκατάστασης πακέτων μέσω του npm (Node Package Manager), ενός συστήματος διαχείρισης πακέτων.

2.3.4 ExpressJs

Το Express.js [11] είναι ένα ελαφρύ, ευέλικτο πλαίσιο εφαρμογής (web application framework) για τη δημιουργία server-side εφαρμογών με χρήση της γλώσσας προγραμματισμού JavaScript και του περιβάλλοντος εκτέλεσης Node.js και αποτελειται ως ένα από τα πιο δημοφιλή frameworks για τη δημιουργία API. Ορισμένες βασικές λειτουργίες του Express.js περιλαμβάνουν τον καθορισμό και την διαχείριση των διαδρομών (Routes), τον χειρισμό αιτημάτων (requests) και απαντήσεων (responses) και τον καθορισμό μέσων (middleware) για επεξεργασία αιτημάτων.

2.4 Βιβλιοθήκες και Frameworks

Προηγουμένως περιγράψαμε εκτενώς τις βασικές τεχνολογίες που αποτελούν το θεμέλιο της εργασίας μας. Ωστόσο, η υλοποίηση της εφαρμογής απαιτούσε περισσότερα από απλή χρήση των βασικών τεχνολογιών. Επιλέξαμε και ενσωματώσαμε διάφορα frameworks και βιβλιοθήκες, προσδίδοντας έτσι επιπλέον δυνατότητες και ευελιξία της εφαρμογής μας.

2.4.1 Tailwind CSS

Το Tailwind CSS [12] είναι ένα πλαίσιο (framework) CSS που σχεδιάστηκε για τη δημιουργία προσαρμοσμένων, αποτελεσματικών,πολύ γρήγορων και επαναχρησιμοποιήσιμων στυλ (styles). Μια βασική διαφορά που διακρίνετε με την παραδοσιακή CSS μπορείτε να δείτε στο παρακάτω παράδειγμα.

```
.button {
   padding: 10px 20px;
   background-color: #3498db;
   color: #fff;
   border-radius: 5px;
   cursor: pointer;
 }
```

Εδώ είναι το ίδιο κουμπί με τα ίδια χαρακτηριστικά σε tailwind CSS

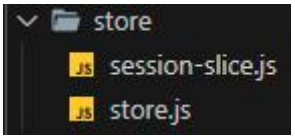
```
<button className="p-2 bg-blue-500 text-white rounded cursor-pointer">Click me</button>
```

2.4.2 Mapbox

Το Mapbox [13] είναι μια εταιρεία που παρέχει υπηρεσίες χαρτογράφησης και το Mapbox Platform είναι ένα σύνολο εργαλείων και υπηρεσιών που επιτρέπουν στους προγραμματιστές να ενσωματώνουν χάρτες σε εφαρμογές και ιστοσελίδες. Προτιμήθηκε από το google maps διότι παρέχει δωρεάν πακέτο μέχρι κάποιο περιορισμένο αριθμό requests και δεν απαιτείται προσθήκη πιστωτικής κάρτας.

2.4.3 React Redux

Η React Redux είναι ένα state management framework [14] και ο σκοπός του είναι να διευκολύνει τη διαχείριση της κατάστασης (state) της εφαρμογής.Στις μεγάλες εφαρμογές, η διαχείριση της κατάστασης, μπορεί να γίνει δύσκολη λόγω των μεγάλων αριθμό components. Η React Redux έρχεται να λύσει αυτό το πρόβλημα παρέχοντας μια κεντρική αποθήκη (store) όπου μπορούν να αποθηκευτούν τα δεδομένα της εφαρμογής και έπειτα να ενημερώνουμε και λαμβάνουμε συγκεκριμένες καταστάσεις με την βοήθεια συναρτήσεων που παρέχει η React Redux.



Σχήμα 2.1 : React Redux store φάκελος

2.4.4 React Router

Το React Router [15] είναι μια βιβλιοθήκη που παρέχει δυνατότητα πλοήγησης και διαχείρισης διαδρομών σε μια εφαρμογή React. Τα κύρια χαρακτηριστικά του περιλαμβάνουν τη δυνατότητα πλοήγησης μέσω των Components <Link> και <NavLink>, τα οποία επιτρέπουν τη δημιουργία συνδέσμων πλοήγησης μεταξύ διαφορετικών σελίδων της εφαρμογής. Επίσης, διαχειρίζεται τις διαδρομές της εφαρμογής (Routing), επιτρέποντας τον καθορισμό ποιου component πρέπει να αποδείξει για κάθε διαδρομή. Έχει επίσης τη δυνατότητα παραμετροποίησης των διαδρομών (Route Parameters), επιτρέποντας τη δημιουργία δυναμικών σελίδων, και υποστηρίζει την εμφώλευση διαδρομών (Nested Routing), επιτρέποντας την εμφάνιση συγκεκριμένων διαδρομών της εφαρμογής μέσα σε άλλες διαδρομές.

2.4.5 Axios

Το Axios [16] είναι μια βιβλιοθήκη της Javascript και χρησιμοποιείται για την διεκπεραίωση HTTP αιτημάτων σε φυλλομετρητές (browsers), εξυπηρετητές (servers) και ανάκτηση δεδομένων από API ή άλλους εξωτερικούς πόρους. Τα αιτήματα που συμπεριλαμβάνει είναι τα GET, POST, PUT, και DELETE.

Παρακάτω βλέπουμε ένα αίτημα όπου παίρνουμε πληροφορίες για ένα συγκεκριμένο φαρμακείο με βάση το ID του.

```
import axios from "axios";
```

```
const getPharmacyData = async (id) => {
   try {
     const config = { headers: { Accept: "application/json" } };
     const res = await axios.get(`/pharmacies/${id}`, config);
     return res.data;
   } catch (e) {
     return "Seems to be an error";
   }
 };
```

2.4.6 Routee API

Είναι API της AMD Telecom [17]. Με αυτά τα API's μπορούμε να φτιάξουμε SMS μηνύματα επιλέγοντας ένα σώμα μηνύματος, ένα όνομα για αποστολέα και τηλέφωνο προορισμού.

Παρακάτω βλέπουμε και ένα αίτημα δημιουργίας sms μηνύματος προς έναν παραλήπτη το οποίο μπορούμε να το βρούμε στο αρχείο routee.js.

```
const sendSMStoUser=(userPhone,username)=>{
  const userPhoneInt= parseInt(userPhone, 10);
  const options = {
       method: 'POST',
       url: 'https://connect.routee.net/sms',
       headers: {
         accept: 'application/json',
         'Content-Type': 'application/json',
         Authorization: `Bearer RouteeToken`
       },
       data: {
         flash: false,
         ttl: 4320,
         transcode: false,
         urlShortener: {urlValidity: '2592000'},
         body: `Γειά σου ${username}. Η παραγγελία σου έχει καταχωρηθεί με επιτυχία!`,
         to: userPhoneInt,
         from: 'Pharmacies'
       }
     };
     axios
       .request(options)
       .then(function (response) {
         console.log(response.data);
       })
       .catch(function (error) {
         console.error(error);
```

```
    });
}
```

2.4.7 Multer και Multer storage Cloudinary

Το Multer [18] είναι ένα middleware για το Express.js που χρησιμοποιείται σε φόρμες ανεβάσματος αρχείων και φωτογραφιών μέσω αιτημάτων POST. Παρέχει τη δυνατότητα να αλλάξουμε τα ονόματα των αρχείων, να επιλέξετε τους επιτρεπτούς τύπους αρχείων και να ορίσουμε ένα όριο μεγέθους.

Στην προκειμένη περίπτωση όπως θα δούμε παρακάτω, θέλουμε να ανεβάσουμε σε μία πλατφόρμα αποθήκευσης (cloud), φωτογραφίες. Όποτε ιδανικά θα επιθυμούσαμε να ορίσουμε και να ονομάσουμε τον φάκελο που θα αποθηκεύονται οι εικόνες. Η βιβλιοθήκη Multer storage Cloudinary μπορεί να μας δώσει την λύση.

Ο παρακάτω κώδικας μας δείχνει τους επιτρεπόμενους τύπους αρχείων και το όνομα του φακέλου που θα αποθηκεύονται οι φωτογραφίες.

```
const cloudinary = require('cloudinary').v2;
const { CloudinaryStorage } = require('multer-storage-cloudinary');

cloudinary.config({
   cloud_name: process.env.CLOUDINARY_CLOUD_NAME,
   api_key: process.env.CLOUDINARY_KEY,
   api_secret: process.env.CLOUDINARY_SECRET
})

const storage= new CloudinaryStorage({
   cloudinary,
   params:{
      folder:"CommunityPharamacy",
      allowedFormats:["jpeg","png","jpg"]
   }
})

module.exports={
   cloudinary,
   storage}
```

2.4.8 Cloudinary

Το Cloudinary [19] είναι μια υπηρεσία cloud που παρέχει λύσεις για τη διαχείριση και την παροχή ψηφιακού περιεχομένου, κυρίως για αρχεία πολυμέσων όπως εικόνες και βίντεο. Προσφέρει

δυνατότητες όπως αποθήκευση, επεξεργασία, διαχείριση και παράδοση περιεχομένου μέσω του cloud. Εμείς θα χρησιμοποιήσουμε την υπηρεσία κυρίως για αποθήκευση φωτογραφιών.

2.4.9 Mongoose

Το Mongoose [20] είναι μια βιβλιοθήκη ODM (Object-Document Mapper) της MongoDB στο Node.js όπου επιτρέπει τον προγραμματιστή να χρησιμοποιεί JavaScript για να αλληλεπιδρά με τη βάση δεδομένων. Οι βασικές λειτουργίες του Mongoose περιλαμβάνουν τον καθορισμό του σχήματος, τη μοντελοποίηση των δεδομένων, την εύκολη σύνδεση με τη βάση και τέλος επεξεργασία και δημιουργία ερώτησης των δεδομένων.

2.10 Nodemailer

Το Nodemailer [21] είναι μια βιβλιοθήκη για το Node.js που χρησιμοποιείται για την εύκολη αποστολή email και υποστηρίζει τα πρωτόκολλα SMTP, IMAP και POP3. Παρέχει δημιουργία email με HTML κώδικα και αποστολή συνημμένων αρχείων. Στην συγκεκριμένη εργασία θα το ενσωματώσουμε για την αυθεντικοποίηση του χρήστη και για την επαναφορά κωδικού πρόσβασης. Για να μπορέσουμε να χρησιμοποιήσουμε την υπηρεσία SMTP, θα πρέπει να ορίσουμε πρώτα ένα domain ως αποστολέα.

Οπότε προβήκαμε σε ένα gmail λογαριασμό, κάναμε τις απαραίτητες ρυθμίσεις για να πάρουμε τα ανάλογα κλειδιά (credentials) για να τα τοποθετήσουμε στον παρακάτω κώδικα. Με το παρακάτω κώδικα δημιουργούμε ένα SMTP email για επαναφορά κωδικού πρόσβασης.

```
const sendResetPasswordMail= async(name,email,token)=>{
    try{
        const transporter=nodemailer.createTransport({
            host:"smtp.gmail.com",
            port: 587,
            secure: false,
            requireTLS:true,
            auth:{
                user:config.emailUser,
                pass: config.emailPassword
            }
        });
        const mailOptions={
            from:config.emailUser,
            to:email,
            subject:"Community Pharmacies Reset Password !",
            html:"<p> Hi "+name+ `! Please Click here to <a
href=http://localhost:3000/resetpassword?token=${token}"> Reset</a> Password.`
        }

        transporter.sendMail(mailOptions,function(error,info){
            if(error){
                console.log(error)
            }else{
```

```
            console.log("Email has been Sent- ",info.response);
        }
    })
}catch(error){
    console.log(error.message)
}}
```

2.4.11 Passport

Το Passport [22] αποτελεί μια βιβλιοθήκη αυθεντικοποίησης για το Node.js [8], χρήσιμο για την διαχείριση της διαδικασίας σύνδεσης (login). Αυτή η βιβλιοθήκη είναι επεκτάσιμη, παρέχοντας τη δυνατότητα επέκτασης μέσω στρατηγικών (strategies).

Συγκεκριμένα, επιλέχθηκε η στρατηγική Passport Local Mongoose. Αυτή η στρατηγική διατηρεί τη βασική λειτουργικότητα της διαδικασίας σύνδεσης και επεκτείνει την λειτουργικότητα προσφέροντας πρόσθετες δυνατότητες.Η βιβλιοθήκη αναλαμβάνει τη διαδικασία κρυπτογράφησης του κωδικού όταν ένας χρήστης δημιουργεί έναν λογαριασμό και διαχειρίζεται ολόκληρη τη διαδικασία αυθεντικοποίησης όταν ένας χρήστης προσπαθεί να συνδεθεί στην εφαρμογή. Παρακάτω, θα δούμε σε ποιο σημείο και πώς χρησιμοποιήθηκε η συγκεκριμένη λειτουργικότητα.

2.4.12 React icons

Είναι μια αρκετά απλή βιβλιοθήκη η οποία περιέχει μια αρκετή μεγάλη γκάμα εικονιδίων τα οποία είναι ταξινομημένα στην ιστοσελίδα και χρησιμοποιούνται σε μορφή component. Για να την εγκαταστήσουμε χρησιμοποιήσουμε την παρακάτω εντολή στο terminal του Node JS.

npm install react-icons

Για να μπορέσουμε να εισάγουμε την βιβλιοθήκη, εισάγουμε τον παρακάτω κώδικα.

import { MdOutlineCancel } from "react-icons/md";

το MdOutlineCancel είναι το αντικείμενο του εικονιδίου και το react-icons/md είναι η διαδρομή που βρίσκεται το εικονίδιο. Παρακάτω θα δούμε ότι άλλα εικονίδια βρίσκονται σε διαφορετική διαδρομή.

Για να μπορέσουμε να χρησιμοποιήσουμε το icon απλά καλούμε το αντικείμενο εκεί που επιθυμούμε ως <MdOutlineCancel />.

2.4.13 React Hot Toast

Το react hot toast είναι μια αρκετά απλή βιβλιοθήκη που χρησιμοποιείται όταν θέλουμε να εμφανίζουμε ειδοποιήσεις. Από μόνο του έχει κάποια έτοιμα components που απλά πληκτρολογείς το μήνυμα που θέλεις να εμφανίζεται, σε πιο σημείο της οθόνης (στην μέση, γωνία κτλ) και αν είναι για μήνυμα λάθους ή επιτυχίας. Χρησιμοποιήθηκε αρκετά στην εφαρμογής ώστε να έχει experience ο χρήστης για κάθε υποβολή παραγγελίας, είσοδο λογαριασμού κτλ.

Για να το εγκαταστήσουμε,πληκτρολογούμε στο terminal του NodeJs

npm install react-hot-toast

Έπειτα, τοποθετείς το component με τα χαρακτηριστικά που επιθυμούμε

```
<Toaster
  position="bottom-right"
  reverseOrder={false}
/>
```

Τέλος, το μήνυμα που θα γίνετε trigger στην συνάρτηση της javaScript.

```
toast.success('Successfully toasted!')
```

Κεφάλαιο 3ο: Υλοποίηση Του Community Pharmacies

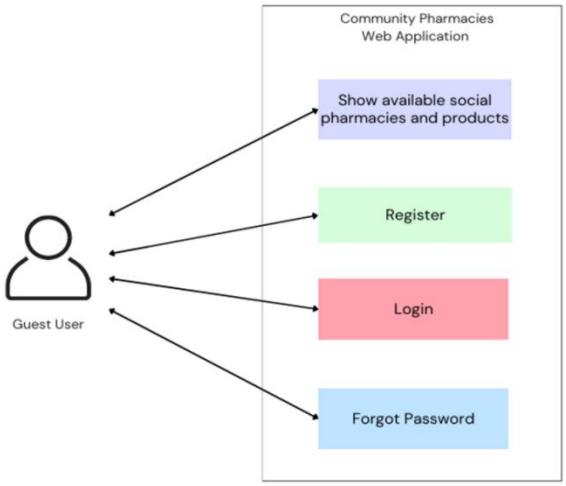
Στα πλαίσια αυτού του κεφαλαίου, θα ξεκινήσουμε με την σχεδίαση υλοποίησης της εφαρμογής μέσω user stories, την περιγραφή της αρχιτεκτονικής ενώ στη συνέχεια θα εξετάσουμε την υλοποίηση της. Επιπλέον, θα προχωρήσουμε στη ανάλυση των βιβλιοθηκών και των τεχνολογιών που έλαβαν καθοριστικό ρόλο στην ολοκλήρωση της εφαρμογής. Τέλος, θα παρουσιάσουμε τα Endpoints της εφαρμογής, εξετάζοντας τον έλεγχο αυθεντικοποίησης και τη βάση δεδομένων.

3.1 Σχεδίαση και Λειτουργικές απαιτήσεις

Η λειτουργικές απαιτήσεις αναφέρονται σε συγκεκριμένες λειτουργίες ή χαρακτηριστικά που πρέπει να παρέχει ένα σύστημα προκειμένου να ικανοποιήσει τις ανάγκες του χρήστη. Οι λειτουργικές απαιτήσεις καθορίζουν το τι πρέπει να γίνει, δηλαδή τις επιθυμητές ενέργειες ή αποτελέσματα που αναμένονται από το σύστημα.

Η ιστορία χρήστη (User Stories) είναι ένας τρόπος να περιγράψουμε μια λειτουργική απαίτηση από την πλευρά του χρήστη. Ένα User Story περιλαμβάνει τρία βασικά στοιχεία, τίτλο, περιγραφή και κριτήρια αποδοχής. Παρακάτω θα περιγράψουμε τα user stories για όλων των ειδών χρήστες.

3.1.1 User story: Επιλογές ως μη συνδεδεμένος ή μη εγγεγραμμένος χρήστης.



Σχήμα 3.1.1: Μη συνδεδεμένος ή μη εγγεγραμμένος χρήστης user story.

Σενάριο 1 - Περιήγηση στο σύστημα:

Περιγραφή: Ο μη συνδεδεμένος χρήστης θέλει να περιηγηθεί στο σύστημα βλέποντας τα διαθέσιμα κοινωνικά φαρμακεία και τα προϊόντα που υπάρχουν.

Κριτήρια Αποδοχής:

- Υπάρχει μια αρχική σελίδα που εμφανίζει όλα τα διαθέσιμα κοινωνικά φαρμακεία και την επιλογή για να εμφανίσει όλα τα προϊόντα.
- Πατώντας πάνω σε ένα κοινωνικό φαρμακείο ή προϊόντα, βλέπει και περισσότερες λεπτομέρειες για το κοινωνικό φαρμακείο.

Σενάριο 2 - Είσοδος:

Περιγραφή: Ο μη συνδεδεμένος αλλά εγγεγραμμένος χρήστης θέλει να κάνει είσοδο στο σύστημα.

Κριτήρια Αποδοχής:

- Υπάρχει φόρμα εισόδου που περιλαμβάνει πεδία για username και κωδικό πρόσβασης.

Σενάριο 3 - Εγγραφή:

Περιγραφή: Ο μη εγγεγραμμένος χρήστης θέλει να εγγραφεί στο σύστημα.

Κριτήρια Αποδοχής:

- Υπάρχει σύνδεσμος Register που τον καθοδηγεί σε μια φόρμα εγγραφής.
- Η φόρμα περιλαμβάνει απαραίτητα πεδία και κουμπί "Register".
- Αφού δημιουργήσει τον λογαριασμό και συνδεθεί στο σύστημα, πρέπει να πατήσει το κουμπί verify ώστε να επαληθεύσει τον λογαριασμό του.
- Θα λάβει ένα email στο ηλεκτρονικό του ταχυδρομείο όπου πρέπει να πατήσει τον σύνδεσμο για να επαληθευτεί ο λογαριασμός του.

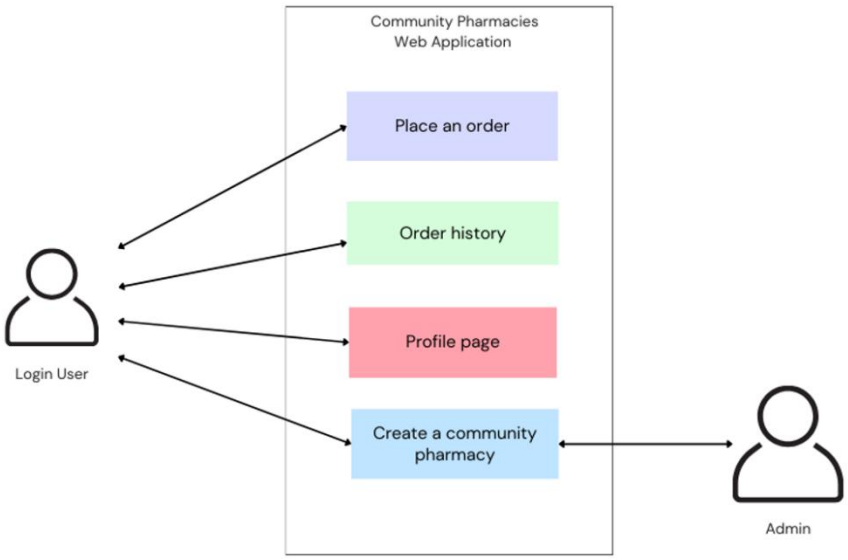
Σενάριο 4 - Επαναφορά Κωδικού:

Περιγραφή: Ο μη συνδεδεμένος χρήστης είναι εγγεγραμμένος και θέλει να επαναφέρει τον κωδικό πρόσβασης του.

Κριτήρια Αποδοχής:

- Υπάρχει σύνδεσμος Forgot password που τον καθοδηγεί σε μια φόρμα επαναφοράς κωδικού.
- Η φόρμα περιλαμβάνει πεδίο για την εισαγωγή του email και κουμπί "Αποστολή".
- Ο χρήστης θα λάβει ένα email που θα πρέπει να πατήσει τον σύνδεσμο.
- Αφού πατήσει τον σύνδεσμο θα πρέπει να πληκτρολογήσει τον νέο κωδικό πρόσβασης.

3.1.2 User story: Επιλογές ως συνδεδεμένος χρήστης.



Σχήμα 3.1.2: Συνδεδεμένος χρήστης user story.

Σενάριο 1 - Δημιουργία κράτησης/παραγγελίας:

Περιγραφή: Ο συνδεδεμένος χρήστης θέλει να περιηγηθεί στο σύστημα βλέποντας τα διαθέσιμα κοινωνικά φαρμακεία και τα προϊόντα που υπάρχουν και θέλει να κάνει κράτηση μερικά προϊόντα.

Κριτήρια Αποδοχής:

- Υπάρχει μια αρχική σελίδα που εμφανίζει όλα τα διαθέσιμα κοινωνικά φαρμακεία και την επιλογή για να εμφανίσει όλα τα προϊόντα.
- Πατώντας πάνω σε ένα κοινωνικό φαρμακείο ή προϊόντα, βλέπει και περισσότερες λεπτομέρειες για το κοινωνικό φαρμακείο.
- Επιλέγει αριθμό προϊόντων και πατάει order ώστε να κάνει κράτηση τα προϊόντα και να τα παραλάβει από το κατάστημα.
- Υπάρχει σελίδα που βλέπει ο χρήστης όλες τις ενεργές του κρατήσεις/παραγγελίες.

Σενάριο 2 - Ιστορικό κρατήσεων/παραγγελιών:

Περιγραφή: Ο συνδεδεμένος χρήστης θέλει να βλέπει το ιστορικό κρατήσεων/παραγγελιών του και να έχει την επιλογή να επιλέξει το προϊόν να το κάνει παραγγελία ξανά.

Κριτήρια Αποδοχής:

- Υπάρχει σελίδα που περιλαμβάνει όλες τις κρατήσεις/παραγγελίες που έχουν γίνει με επιτυχία ή έχουν απορριφθεί.
- Έχει την δυνατότητα να επιλέξει από τις προηγούμενες παραγγελίες/κρατήσεις το προϊόν και να τον ανακατευθύνει μέσα στο σύστημα ώστε να το ξανά παραγγείλει.

Σενάριο 3 - Προβολή προφίλ:

Περιγραφή: Ο συνδεδεμένος χρήστης θέλει να δει τα στοιχεία λογαριασμού του.

Κριτήρια Αποδοχής:

- Υπάρχει σελίδα που περιλαμβάνει τα στοιχεία λογαριασμού του.

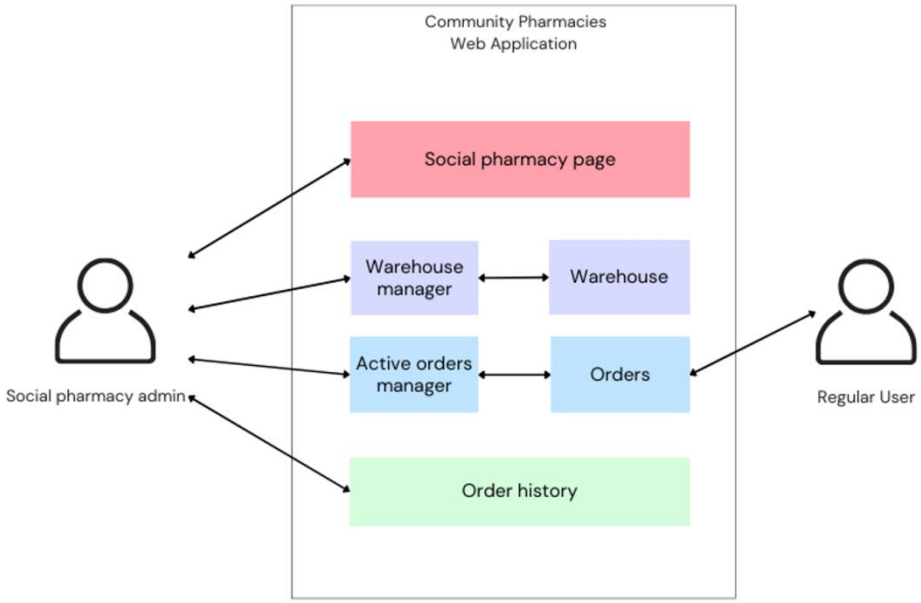
Σενάριο 4 - Δημιουργία/καταχώρηση κοινωνικού φαρμακείου :

Περιγραφή: Ο συνδεδεμένος χρήστης θέλει να δημιουργήσει/καταχωρήσει το κοινωνικό φαρμακείο στο σύστημα ως διαχειριστής του κοινωνικού φαρμακείου.

Κριτήρια Αποδοχής:

- Υπάρχει σελίδα που περιέχει την φόρμα για δημιουργία/καταχώρηση κοινωνικού φαρμακείου που πρέπει να συμπληρώσει κάποια πεδία όπως όνομα κοινωνικού φαρμακείου, διεύθυνση, εικόνα,σε ποιο σημείο βρίσκεται στον χάρτη κτλ.
- Έπειτα από την υποβολή του, θα πρέπει να εγκριθεί από τον διαχειριστή του συστήματος.

3.1.3 User story: Επιλογές ως διαχειριστής κοινωνικού φαρμακείου.



Σχήμα 3.1.3: Διαχειριστής κοινωνικού φαρμακείου user story.

Σενάριο 1 - Προβολή προφίλ κοινωνικού φαρμακείου:

Περιγραφή: Ο διαχειριστής κοινωνικού φαρμακείου θέλει να δει τα στοιχεία του κοινωνικού φαρμακείου του.

Κριτήρια Αποδοχής:

- Υπάρχει σελίδα που περιλαμβάνει τα στοιχεία κοινωνικού φαρμακείου του.

Σενάριο 2 - Προσθήκη προϊόντων κοινωνικού φαρμακείου:

Περιγραφή: Ο διαχειριστής κοινωνικού φαρμακείου θέλει να προσθέσει προϊόντα στο σύστημα.

Κριτήρια Αποδοχής:

- Υπάρχει σελίδα που μπορεί να προσθέσει προϊόντα. Πρέπει να συμπληρώσει όνομα προϊόντος,κατασκευαστή, αν απαιτεί συνταγή, ποσότητα κτλ.

Σενάριο 3 - Διαχείριση των υπάρχων προϊόντων:

Περιγραφή: Ο διαχειριστής κοινωνικού φαρμακείου θέλει να προσθέσει ποσότητα σε κάποια υπάρχων προϊόντα στο σύστημα και θέλει να αφαιρέσει κάποια άλλα προϊόντα από το σύστημα.

Κριτήρια Αποδοχής:

- Υπάρχει σελίδα που μπορεί να διαχειριστεί τα υπάρχων προϊόντα. Μπορεί να προσθέσει ή αφαιρέσει αντίστοιχα μαζικά αριθμό προϊόντων στα υπάρχοντα προϊόντα.
- Στην λίστα με τα υπάρχων προϊόντα υπάρχει κουμπί διαγραφή για να διαγράψει το επιλεγμένο προϊόν.

Σενάριο 4 - Διαχείριση κρατήσεων.

Περιγραφή: Ο διαχειριστής κοινωνικού φαρμακείου θέλει να βλέπει τις ενεργές παραγγελίες και να τις διαχειρίζεται.

Κριτήρια Αποδοχής:

- Υπάρχει σελίδα που μπορεί να βλέπει τις ενεργές παραγγελίες.
- Έχει την επιλογή να τις απορρίπτει ή να της ορίζει ως επιτυχημένες όταν παραλάβει την παραγγελία ο χρήστης από το φυσικό κατάστημα.

Σενάριο 5 - Ιστορικό κρατήσεων/παραγγελιών:

Περιγραφή: Ο διαχειριστής κοινωνικού φαρμακείου θέλει να βλέπει το ιστορικό κρατήσεων/παραγγελιών.

Κριτήρια Αποδοχής:

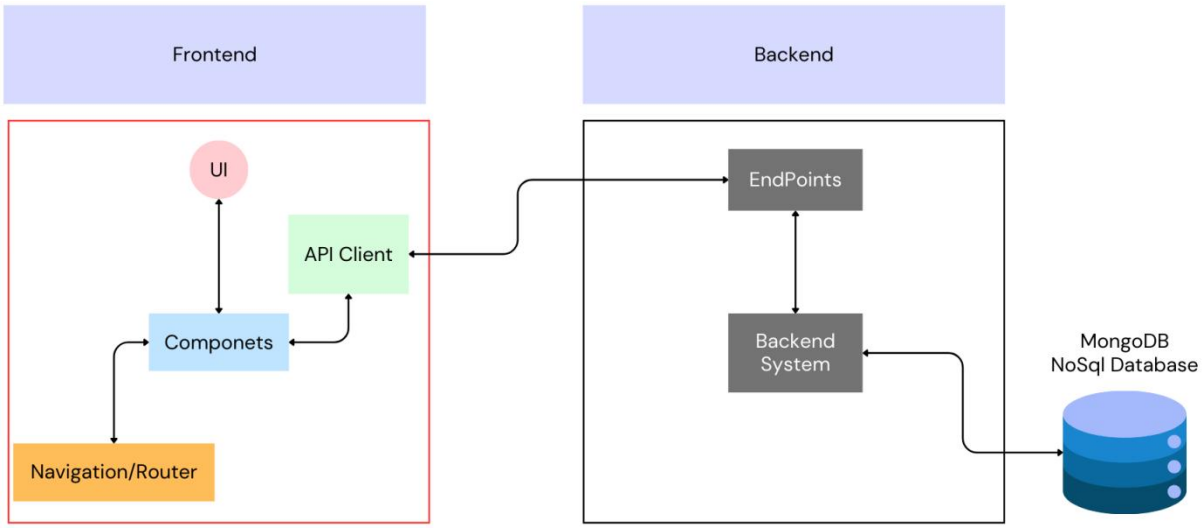
- Υπάρχει σελίδα που περιλαμβάνει όλες τις κρατήσεις/παραγγελίες που έχουν γίνει με επιτυχία ή έχουν απορριφθεί.
- Παρέχει ημερομηνία, ώρα και το όνομα του πελάτη/χρήστη που έχει κάνει την παραγγελία.

3.2 Αρχιτεκτονική εφαρμογής

Η αρχιτεκτονική περιγράφει τον σχεδιασμό της εφαρμογής. Θα περιγραφεί σε δύο υποκεφάλαια τόσο για το frontend όσο και για το backend.

Από την πλευρά του frontend, περιγράφει την εμπειρία του χρήστη της εφαρμογής, ενώ η ανάπτυξη backend αναφέρετε στην παροχή πρόσβασης στα δεδομένα, τις υπηρεσίες, λειτουργίες και άλλα υπάρχοντα συστήματα που κάνουν την εφαρμογή λειτουργική.

3.2.1 Αρχιτεκτονική Frontend



Σχήμα 3.2.1: Αρχιτεκτονική Frontend.

API Client

Κάθε component ή σελίδα που παρέχει λειτουργικότητα ή απαιτεί αλληλεπίδραση με τον εξυπηρετητή, εκτελεί την αντίστοιχη κλήση προς το endpoint, χρησιμοποιώντας το Axios όπως έχουμε αναλύσει σε προηγούμενο κεφάλαιο.Ας δούμε ένα παράδειγμα για να γίνει κατανοητό.

Το αρχείο pharmacy.jsx λειτουργεί ως κύρια σελίδα. Ο χρήστης έχει τη δυνατότητα να προβάλει τα διαθέσιμα κοινωνικά φαρμακεία ή τα διαθέσιμα προϊόντα, ανάλογα με την επιλογή του. Για να επιτευχθεί αυτό, πραγματοποιούνται δύο κλήσεις προς τα αντίστοιχα endpoints του API, με σκοπό τη λήψη των αντίστοιχων απαντήσεων. Παρακάτω παρατίθενται οι αντίστοιχες συναρτήσεις:

```
import axios from "axios";
const allPharmaciesFunc = async () => {
    try {
      const response = await axios.get("/pharmacies");
```

```
      setAllPharmacies(response.data);
    } catch (e) {
      return "Seems to be an error";
    }
  };

  const allProductsFunc = async () => {
    try {
      const response = await axios.get("/products");
      setAllProducts(response.data);
      setGroupedProducts(response.data);
    } catch (e) {
      return "Seems to be an error";
    }
  };
```

Η συνάρτηση allPharmaciesFunc χρησιμοποιείται για να ανακτήσει και να αποθηκεύσει όλα τα κοινωνικά φαρμακεία, ενώ η συνάρτηση allProductsFunc χρησιμοποιείται για τα διαθέσιμα προϊόντα, και οι δύο χρησιμοποιούν τη μέθοδο GET. Σε αυτήν την προσέγγιση, δεν απαιτείται κάποια παράμετρος για να επιστραφεί κάτι συγκεκριμένο, όπως ένα κοινωνικό φαρμακείο με βάση το ID. Οι κλήσεις αυτές γίνονται κάθε φορά που ο χρήστης μεταβαίνει στη σελίδα pharmacy.jsx.

Στην περίπτωση που ο χρήστης επιλέξει και πατήσει ένα κοινωνικό φαρμακείο για να δει περισσότερες λεπτομέρειες, τότε γίνεται μια API GET κλήση που θα επιστρέψει πληροφορίες για το συγκεκριμένο κοινωνικό φαρμακείο βάσει του ID του. Η συγκεκριμένη συνάρτηση βρίσκεται στο showPharamacy.jsx.

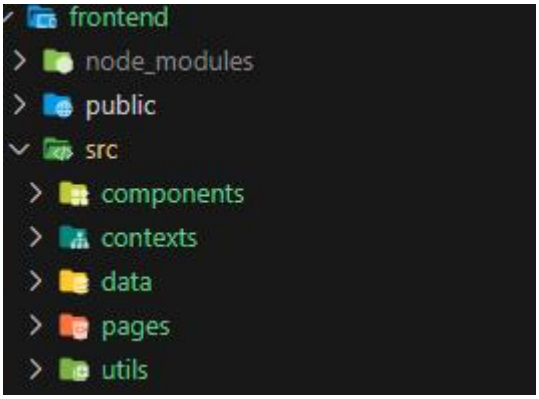
```
 const fetchPharmacies = async () => {
    try {
      const config = { headers: { Accept: "application/json" } };
      console.log("Sent request");
      const response = await axios.get(`/pharmacies/${id}`, config);
      console.log(response);
      setPharmacyData(response.data);
    } catch (e) {
      navigate("/nonepage");
    }
  };
  fetchPharmacies();

}
```

Πρέπει να αναφέρουμε ότι οι κλήσεις προς το API δεν πραγματοποιούνται πάντα, αλλά εξαρτώνται από τα δικαιώματα του χρήστη. Είναι εμφανές ότι ένας μη εξουσιοδοτημένος χρήστης ή ένας επισκέπτης δεν θα πραγματοποιήσει αντίστοιχες κλήσεις με έναν εξουσιοδοτημένο χρήστη. Για περισσότερες πληροφορίες σχετικά με την εξουσιοδότηση θα δούμε παρακάτω.

Ωστόσο, είναι σημαντικό να σημειώσουμε ότι όταν ένας χρήστης συνδέεται επιτυχώς στην εφαρμογή, παίρνουμε ως απάντηση κάποιες πληροφορίες και τις αποθηκεύουμε τοπικά. Αυτές οι πληροφορίες περιλαμβάνουν το ID του χρήστη, την επαλήθευση του χρήστη, κλπ. Με βάση αυτές τις πληροφορίες, παρέχουμε και την ανάλογη πρόσβαση.

Components

Τα components βρίσκονται ανάμεσα στο κέντρο της αρχιτεκτονικής και συνδέεται άμεσα με όλα τα υπόλοιπα τμήματα. Ανάλογα ποιο component έχει ο χρήστης στην οθόνη του, κάνει μια κλήση προς το API και αποθηκεύσει τοπικά την απάντηση και την εμφανίζει. Τα components είναι διαθέσιμα στο φάκελο pages και components.



Σχήμα 3.2.2: Φάκελο components και pages

Οι κύριες διαφορές μεταξύ των δύο είναι ότι το pages αποτελείται ως τις σελίδες του συστήματος, μπορεί να είναι η σελίδα login, να είναι η κύρια σελίδα, σελίδα με τις πληροφορίες του χρήστη κτλ. Ενώ τα components περιέχουν τα κομμάτια των pages, όπου είναι πίνακες, κουμπιά κτλ. Η κίνηση αυτή έχει το πλεονέκτημα ότι σπάμε τις σελίδες σε μικρότερα κομμάτια και έτσι είναι πιο ευανάγνωστο όπως έχουμε αναφέρει σε προηγούμενο κεφάλαιο.

Navigation/ Router

Όπως έχουμε αναφέρει σε προηγούμενο κεφάλαιο, παρέχει δυνατότητα πλοήγησης και διαχείρισης διαδρομών (routes) σε μια εφαρμογή React. Έχει άμεση σχέση με τα components, ανάλογα που θα πλοηγηθεί και πατήσει ανάλογο κουμπί ο χρήστης, εμφανίζεται ανάλογο page μαζί με τα components που του αντιστοιχεί.

Το component sidebar.jsx είναι το sidebar που περιέχει το λογότυπο της εφαρμογής και τα κουμπιά πλοήγησης όπως πληροφορίες, παραγγελίες κεντρική σελίδα κτλ.

Παρακάτω βλέπουμε το αρχείο sidebar.jsx

```
import { HiDatabase, FiEdit} from "react-icons/hi";

import {  AiOutlineShoppingCart,} from "react-icons/ai";
```

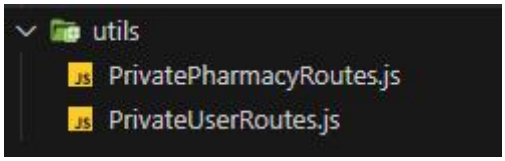
```
function Sidebar() {
 const activeLink =
   "flex items-center gap-5 pl-4 pt-3 pb-2.5 rounded-lg text-sm m-2 text-light-gray-700 dark:text-
black  bg-green-300 hover:bg-green-300 m-2 ";

  const normalLink =
   "flex items-center gap-5 pl-4 pt-3 pb-2.5 rounded-lg text-sm m-2 text-gray-700 dark:text-gray-200
dark:hover:text-black hover:bg-green-100 m-2 ";

return(

<NavLink
          to="/mypharmacy"
         onClick={handleCloseSideBar}
         className={({ isActive }) =>
         isActive ? activeLink : normalLink } >
         <AiOutlineShoppingCart />
          <span className="capitalize">My Pharmacy</span>
 </NavLink>
<NavLink
          to={`mywarehouse/${linkWarehouse}`}
         onClick={handleCloseSideBar}
         className={({ isActive }) =>
         isActive ? activeLink : normalLink  } >
        <HiDatabase />
        <span className="capitalize">My warehouse</span>
 </NavLink>
 <NavLink
          to="/historypharmacy"
         onClick={handleCloseSideBar}
         className={({ isActive }) =>
         isActive ? activeLink : normalLink}>
         <FiEdit />
         <span className="capitalize">History pharmacy orders</span>
</NavLink>
)
export default Sidebar;
```

Το navlink είναι αυτό που θα περιηγηθεί τον χρήστη στο αντίστοιχο url. Έπειτα την περιήγηση, με την λειτουργία του className ={{isActive}) θα αλλάξει δυναμικά το css με την μεταβλητή activeLink απο normalLink ώστε να φαίνεται πως έχει επιλέξει την συγκεκριμένη σελίδα.

Αφού ο χρήστης έχει πλοηγηθεί στην συγκεκριμένη σελίδα γίνεται αντιστοίχιση το url με το component που θα εμφανιστεί. Η εμφάνιση του κάθε component γίνεται στο αρχείο index.js όπως βλέπουμε παρακάτω.

```
ReactDOM.render(
  <Provider store={store}>
    <ContextProvider>
      <Router>
        <Routes>
          <Route path="/" element={<App />}>
            <Route path="pharmacies" element={<Pharmacies />} />
            <Route path="" element={<Pharmacies />} />
            <Route path="pharmacies/:id" element={<ShowPharmacy />} />
            <Route element={<PrivateUserRoutes />}>
              <Route path="orders" element={<Orders />} />
              <Route path="myprofile" element={<MyProfile />} />
              <Route path="newpharmacy" element={<NewPharmacy />} />
              <Route path="historyuser" element={<HistoryUser />} />
            </Route>

            <Route element={<PrivatePharmacyRoutes />}>
              <Route path="mypharmacy" element={<MyPharmacy />} />
              <Route path="mywarehouse/:id" element={<MyWarehouse />} />
              <Route path="pharmacyorders" element={<OrdersPharmacy />} />
              <Route path="historypharmacy" element={<HistoryPharmacy />} />
            </Route>
          </Route>

          <Route path="/resetpassword" element={<ResetPassword />} />
          <Route path="/register" element={<Register />} />
          <Route path="/forgot" element={<Forgot />} />
          <Route path="/login" element={<Login />} />
          <Route path="/verifieduserpage" element={<VerifiedUserPage />} />
          <Route path="/*" element={<NonePage />} />
        </Routes>
      </Router>
    </ContextProvider>
  </Provider>,
  document.getElementById("root"));
```

Παραπάνω βλέπουμε κάποια routes τα οποία είναι εμφωλευμένα. Ο λόγος αυτός είναι διότι κάποιες διαδρομές είναι προσβάσιμες μόνο αν ισχύει ο παραπάνω όρος. Στο route <Route element={<PrivateUserRoutes />}> ο χρήστης θα έχει πρόσβαση και θα μπορεί να περιηγηθεί στις διαδρομές που αποτελείται αν είναι συνδεδεμένος στην εφαρμογή, σε κάθε άλλη περίπτωση, απλά θα ανακατευθυνθεί πίσω στην αρχική σελίδα. Η λειτουργικότητα των ιδιωτικών (private) routes βρίσκονται μέσα στον φάκελο utils.



Σχήμα 3.2.3: Φάκελος Με τις περιπτώσεις διαδρομών

Ας δούμε με περισσότερες λεπτομέρειες τι ακριβώς κάνει τι κάνει το αρχείο PrivateUserRoutes.js

```
import React, { useEffect } from 'react'
import { Navigate,Outlet } from 'react-router-dom';

const  PrivateUserRoutes=()=> {
   const isVerified = localStorage.getItem("isVerified");
   useEffect(()=>{
   },[])
   return (
     isVerified=="false" ||  isVerified==null ? <Navigate to="/" /> : <Outlet />
  )
}
export default PrivateUserRoutes
```

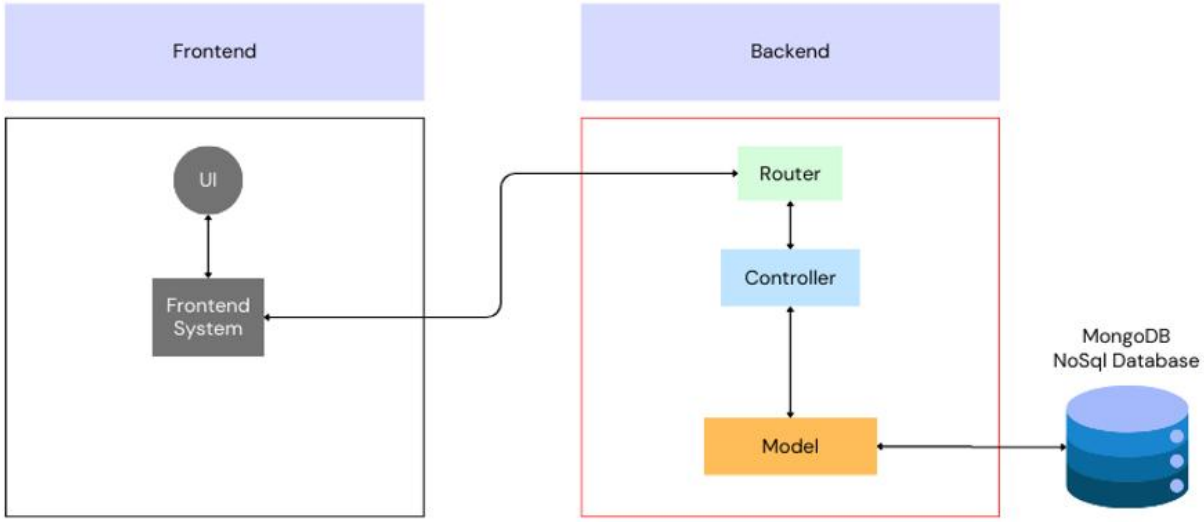
Το αρχείο αρχικά κάνει import τις απαραίτητες βιβλιοθήκες του react router dom ώστε να χρησιμοποιήσουμε τις συναρτήσεις του. Ελέγχει από τις τοπικές μεταβλητές του browser αν ο χρήστης είναι verified και αν ισχύει αυτή η συνθήκη τότε περνάει στην εμφάνιση της διαδρομής.

Ισως παρατηρήσατε ένα component το <Outlet/>, αναφορικά χρησιμοποιείται για να αντιστοιχεί στις περιπτώσεις που υπάρχουν γονικές διαδρομές. Στο παραπάνω κώδικα το route <Route path="/" element={<App />} εμφωλεύει σχεδόν όλες τις διαδρομές εκτός από την σελίδα για είσοδο της εφαρμογής, δημιουργία λογαριασμού κτλ.

Επιπλέον το αρχείο PrivatePharmacyRoutes.js έχει την ίδια λογική αλλά αντί να ελέγχει αν ο χρήστη, ελέγχει αν το κοινωνικό φαρμακείο είναι επαληθευμένο.

Τέλος το route <Route path="/*" element={<NonePage />} />, ορίζει πως για οποιαδήποτε άλλη διαδρομή που δεν υπάρχει στην εφαρμογή, σε ανακατευθύνει σε ένα 404 page.

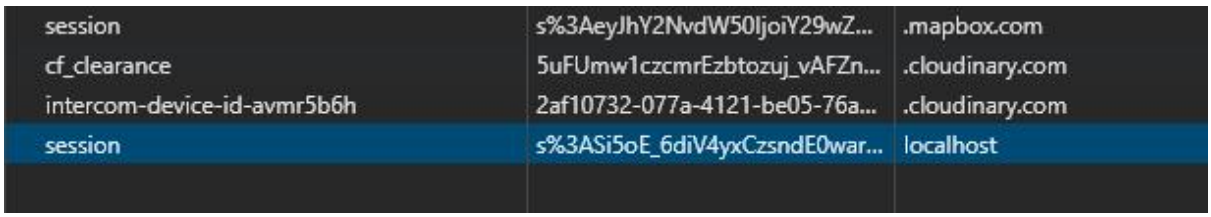
3.2.2 Αρχιτεκτονική Backend



Σχήμα 3.2.4: Αρχιτεκτονική Backend.

Ένα πολύ βασικό στοιχείο στην αρχιτεκτονική του backend είναι οι συνεδρίες (sessions). Στον κόσμο του web development, τα sessions αναφέρονται σε μια μοναδική διάρκεια συνεργασίας μεταξύ του server και ενός client (συνήθως έναν browser). Κατά τη διάρκεια μιας συνεδρίας, δεδομένα μπορούν να διατηρούνται και να μοιράζονται μεταξύ του server και του client.

Το backend είναι υπεύθυνο για τη διαχείριση των δεδομένων του χρήστη, συμπεριλαμβανομένων των sessions. Όταν ένας χρήστης κάνει αίτηση σε έναν server, ο server δημιουργεί ένα session και του αναθέτει ένα μοναδικό αναγνωριστικό. Στη συνέχεια, αυτό το αναγνωριστικό μπορεί να χρησιμοποιηθεί για να αναγνωριστεί ο χρήστης και να διατηρηθεί η κατάσταση της συνεδρίας μεταξύ διαφορετικών αιτημάτων προς τον server.



Σχήμα 3.2.5: Session ID στα developer tools του google browser.

Στον server.js βρίσκεται ο παρακάτω κώδικας που δημιουργεί το session.

```
const MongoStore = require('connect-mongo');
const app = express()
const dbUrl=MONGO_DB_URL;
```

```
const secret="Thisshouldbebettersecret"
const store =new MongoStore({
  mongoUrl: dbUrl,
  secret,
  touchAfter: 24*60*60,
  authenticate:false
});

app.use(session({
  store,
  name: 'session',
  secret,
  resave: true,
  saveUninitialized: true,
  cookie: {
    httpOnly: true,
    expires: Date.now() + 1000 * 60 * 60 * 24 * 7,
    maxAge: 1000 * 60 * 60 * 24 * 7
  }
}));
```

Ο παραπάνω κώδικας χρησιμοποιεί το MongoStore για να αποθηκεύσει τα sessions των χρηστών σε μια βάση δεδομένων MongoDB. Παρακάτω γίνεται η εξήγηση των ρυθμίσεων.

mongoUrl: Η διεύθυνση της βάσης δεδομένων MongoDB.

secret: Ένα μυστικό που χρησιμοποιείται για ασφαλή υπογραφή των sessions.

touchAfter: Η περίοδος σε δευτερόλεπτα μετά την οποία θα ανανεώνεται η σήμανση του session.

authenticate: Αναφέρει εάν απαιτείται πιστοποίηση για ενέργειες που αφορούν τα sessions.

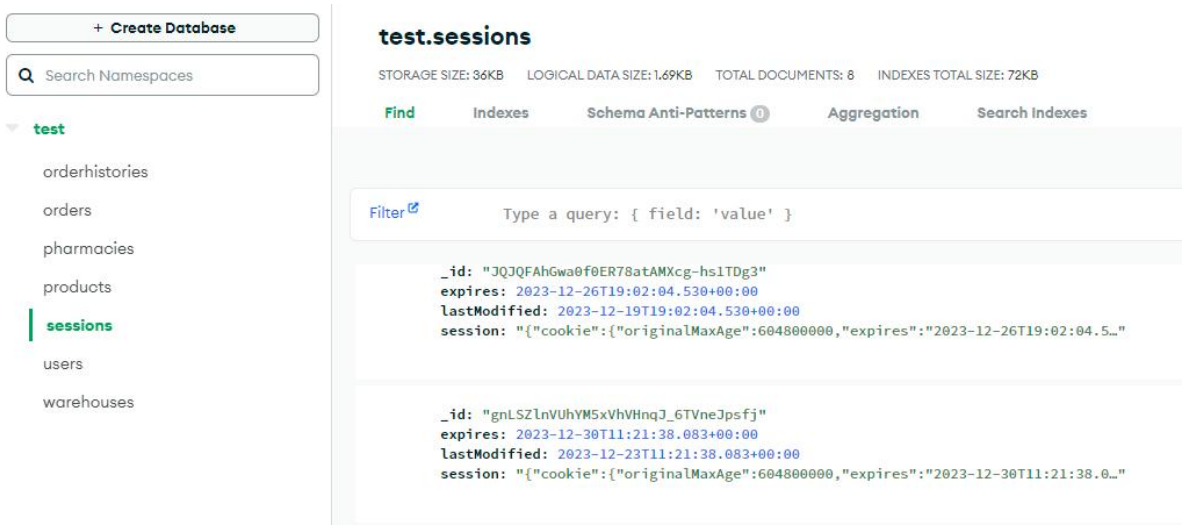
name: Το όνομα του cookie που χρησιμοποιείται για την αποθήκευση του Session ID στον browser του χρήστη.

cookie: Ρυθμίσεις για το cookie που περιέχει το Session ID.

httpOnly: Εάν το cookie είναι προσβάσιμο μόνο μέσω HTTP και δεν μπορεί να τροποποιηθεί μέσω JavaScript, προσθέτοντας ασφάλεια.

expires: Η ημερομηνία λήξης του cookie.

maxAge: Η μέγιστη διάρκεια ζωής του cookie σε χιλιοστά του δευτερολέπτου.



Σχήμα 3.2.6: Sessions στο MongoDB Atlas Database.

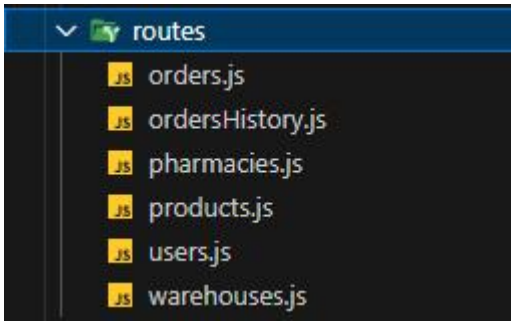
Παραδείγματα με τα sessions υλοποιούνται παρακάτω στο υπό-κεφάλαιο των controllers.

Router

Στο προηγούμενο κεφάλαιο αναφερθήκαμε πως το Axios είναι μια βιβλιοθήκη που χρησιμοποιείται στον client, συνήθως σε συνδυασμό με μια βιβλιοθήκη προβολής (π.χ., React), για να κάνει HTTP αιτήματα προς τον server και να λαμβάνει τα δεδομένα που επιστρέφονται.Τώρα θα αναφέρουμε πώς ανταποκρίνεται στα αιτήματα που λαμβάνει. Τα Routers (Οι δρομολογητές) είναι στο πλαίσιο του ExpressJS για οργάνωση και διαχείριση διαδρομών.Κάθε διαφορετική διαδρομή αντιστοιχεί σε μια διαφορετική λειτουργία ή πόρο της εφαρμογής.

Οι διαδρομές (routes) σε μια full-stack εφαρμογή συνήθως οργανώνονται σε αρχεία καλούμενα routes. Αυτά τα αρχεία λειτουργούν ως δρομολογητές, παρέχοντας μια διαδρομή για κάθε συγκεκριμένη λειτουργία της εφαρμογής. Για παράδειγμα, ένα αρχείο με το όνομα user.js μπορεί να περιέχει όλες τις διαδρομές που σχετίζονται με τους χρήστες, όπως το login, το register, κλπ. Το αρχείο pharmacy.js θα μπορούσε να περιέχει τις διαδρομές που αφορούν το φαρμακείο, όπως η δημιουργία του, η εμφάνιση συγκεκριμένων κοινωνικών φαρμακείων κλπ.

Αυτός ο οργανισμός επιτρέπει μια φυσική διαχωρισμένη δομή, καθιστώντας τον κώδικα πιο ευανάγνωστο και εύχρηστο. Κάθε αρχείο routes είναι υπεύθυνο για τις διαδρομές που σχετίζονται με μια συγκεκριμένη λειτουργία της εφαρμογής, επιτρέποντας έτσι την ευκολότερη συντήρηση και επέκταση του κώδικα. Σε αυτό το πλαίσιο, κάθε αρχείο routes λειτουργεί ως μια λογική μονάδα που είναι υπεύθυνη για την υλοποίηση των συγκεκριμένων λειτουργιών της εφαρμογής, βοηθώντας στην επιτυχή διαχείριση και οργάνωση του κώδικα.



Σχήμα 3.2.7: Φάκελος routes που περιέχει τις διαδρομές

Ας δούμε καλύτερα των διαδρομών των routes και συγκεκριμένα το user.js.

```
const express = require("express");
const router = express.Router();
const users = require("../controllers/users")

router.get("/userData/:id",users.userData)
router.post("/register", users.register);
router.get("/getSessionData",users.getSessionData)
router.post("/login",users.login)
router.get("/logout",users.logout)
router.post("/forgot",users.forgot)
router.get("/retrieveResetUserData/:token",users.retrieveResetUserData)
router.post("/resetPassword",users.resetPassword)
router.post("/verifiedUser",users.verifiedUser)
router.get("/verifyUserfunction/:token",users.verifyUserfunction)

module.exports = router
```

Αρχικά δημιουργούμε δύο σταθερές express και router ώστε να αναφερθούμε στο Express.js framework και στο router που αντιπροσωπεύει το Express.Js. Επιπλέον βλέπουμε ότι δημιουργείται ακόμα μια σταθερά με όνομα users που αναφέρεται σε ένα αρχείο users.js που βρίσκεται στο φάκελο controllers. Για να έχουμε μια εικόνα τι επρόκειτο αυτό το αρχείο, περιέχει την λογική που αντιστοιχεί σε κάθε διαδρομή.

Τέλος, η γραμμή module.exports = router; δηλώνει ότι το αντικείμενο Router που ορίζεται στη μεταβλητή router θα είναι διαθέσιμο για χρήση από άλλα αρχεία ή modules. Συγκεκριμένα στο αρχείο server.js.

Στο αρχείο server.js, το τμήμα που αναφέρεται αποκλειστικά στον router, περιλαμβάνεται ο παρακάτω κώδικας:

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

const productsRoutes=require("./routes/products")
const pharmaciesRoutes=require("./routes/pharmacies")
const warehousesRoutes=require("./routes/warehouses")
const usersRoutes=require("./routes/users")
const ordersRoutes=require("./routes/orders")
const orderHistoryRoutes=require("./routes/ordersHistory")

app.use("/products",productsRoutes)
app.use("/pharmacies",pharmaciesRoutes)
app.use("/warehouses",warehousesRoutes)
app.use("/users",usersRoutes)
app.use("/orders",ordersRoutes)
app.use("/ordershistory",orderHistoryRoutes)
```

Στο πρώτο μέρος, με το const app = express(), δημιουργούμε μια νέα εφαρμογή Express και αποθηκεύουμε το αντικείμενο στη μεταβλητή app. Η μεταβλητή app θα χρησιμοποιηθεί για να διαχειρίζεται τον server και στο const express = require('express'), φορτώνουμε το module express ώστε να χρησιμοποιήσουμε το framework.

Στο δεύτερο μέρος, δημιουργήθηκαν διάφοροι δρομολογητές (routes) για τις διάφορες λειτουργίες της εφαρμογής μας πχ το productsRoutes είναι δρομολογητής για τις λειτουργίες που σχετίζονται με προϊόντα, το pharmaciesRoutes είναι δρομολογητής για τις λειτουργίες που σχετίζονται με τα κοινωνικά φαρμακεία κτλ.

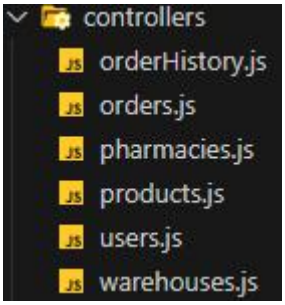
Στο τρίτο μέρος, χρησιμοποιούμε τη μέθοδο app.use() για να ορίσουμε πώς θα δρομολογούνται οι αιτήσεις για κάθε δρομολογητή:Όταν η αίτηση αναφέρεται στον /products, ο δρομολογητής productsRoutes αναλαμβάνει τον χειρισμό της. Όταν η αίτηση αναφέρεται στο /pharmacies, ο δρομολογητής pharmaciesRoutes αναλαμβάνει τον χειρισμό της και ούτω καθεξής για τους υπόλοιπους δρομολογητές.

Με αυτόν τον τρόπο, έχουμε οργανώσει τη λογική των διαφορετικών τμημάτων της εφαρμογής μας σε διαφορετικούς δρομολογητές, καθιστώντας τον κώδικα πιο δομημένο και ευανάγνωστο.

Controller

Προηγουμένως, αναφέρουμε ότι τα controllers περιλαμβάνουν τη λογική που αντιστοιχεί σε κάθε διαδρομή. Τώρα, θα προχωρήσουμε σε μια πιο λεπτομερή ανάλυση του ρόλου και των λειτουργιών τους.

Στο πλαίσιο μιας web εφαρμογής, τα controllers αντιπροσωπεύουν το "νου" της εφαρμογής. Είναι υπεύθυνα για τον χειρισμό των λειτουργικών πτυχών των αιτημάτων των χρηστών. Κυρίως, τα controllers δέχονται τα αιτήματα (requests) από τα routes, εκτελούν την κατάλληλη λογική επεξεργασίας και στη συνέχεια επιστρέφουν την κατάλληλη απόκριση (response).



Σχήμα 3.2.8: Ο Φάκελος controllers

Ας δούμε μια δομή ενός controller και συγκεκριμένα που αφορά το κοινωνικό φαρμακείο.

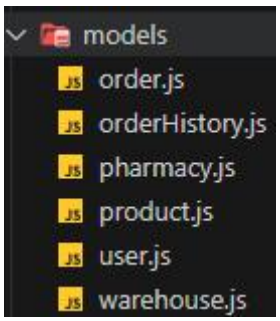
```
const Pharmacy = require("../models/pharmacy");

module.exports.index = async (req, res) => {
  try {
    const pharmacy = await Pharmacy.find({ isVerified: true });
    res.send(pharmacy);
  } catch (error) {
    res.status(500).send("An error occurred while fetching verified pharmacies.");
  }
};
```

Αρχικά, εισάγετε το μοντέλο (model) Pharmacy από το αρχείο pharmacy.js που βρίσκεται στον υποφάκελο models, τα οποία μοντέλα θα εξηγηθούν αμέσος παρακάτω. Έπειτα, ορίζουμε μια συνάρτηση index στο module.exports, η οποία είναι διαθέσιμη σε άλλο μέρος του κώδικά, πιο συγκεκριμένα στο router του pharmacy. Η συνάρτηση αυτή χρησιμοποιεί το async για να υποστηρίξει την ασύγχρονη εκτέλεση για να επιστρέψει το αποτέλεσμα όλων το κοινωνικών φαρμακείων τις εφαρμογής τα οποία όμως είναι επιβεβαιωμένα από τον διαχειριστή της εφαρμογής. Σε περίπτωση σφάλματος, θα επιστρέψει ένα κατάλληλο μήνυμα σφάλματος και έναν κωδικό κατάστασης HTTP 500.

Παραπάνω αναφερθήκαμε στο asyc. Το async χρησιμοποιείται σε συνδυασμό με το await για τον χειρισμό ασύγχρονου κώδικα. Ο όρος ασύγχρονος σημαίνει ότι ο κώδικας δεν εκτελείται σε μια συγκεκριμένη σειρά από πάνω προς τα κάτω, αλλά μπορεί να εκτελεστεί καθώς οι ασύγχρονες εργασίες ολοκληρώνονται. Όταν μια συνάρτηση δηλώνεται με το async, σημαίνει ότι η συνάρτηση θα επιστρέψει πάντα ένα promise. Η async συνάρτηση χρησιμοποιεί το await μέσα στο σώμα της ώστε να περιμένει την εκτέλεση του promise. Η χρήση του await καθιστά την ασύγχρονη λειτουργία του κώδικα πιο ευανάγνωστη και εύκολη στην κατανόηση. Στην παραπάνω συνάρτηση,χρησιμοποιήσαμε το await ώστε να περιμένει την ολοκλήρωση της ασύγχρονης λειτουργίας Pharmacy.find({ isVerified: true }).

Model

Έγινε παραπάνω μια μικρή αναφορά στα models αλλά είναι καλό να εξηγηθούν ακόμα περαιτέρω. Τα "models" σε μια εφαρμογή που χρησιμοποιεί MongoDB και Mongoose αναφέρονται στον τρόπο που καθορίζουμε τη δομή και τις λειτουργίες των δεδομένων που θα αποθηκεύονται στη βάση δεδομένων. Σχεδιάζονται για να αντιπροσωπεύουν τους τύπους δεδομένων, να καθορίζουν τους κανόνες εγκυρότητας, και να παρέχουν μεθόδους για την αλληλεπίδραση με τα δεδομένα αυτά. Όλα τα models βρίσκονται οργανωμένα στον φάκελο models.



Σχήμα 3.2.9: Ο Φάκελος Models.

Ας δούμε αναλυτικότερα το user model.

```
const mongoose = require('mongoose');
const passportLocalMongoose = require('passport-local-mongoose');
const Schema=mongoose.Schema;

const UserSchema=new Schema({
   firstName:String,
   lastName:String,
   mobile:Number,
   email:{
      type: String,
      require: true,
      unique:true
   },
   token:{
      type:String,
      default:""
   },
   isVerified:{
      type:Boolean,
      default:false
   },
   pharmacy: {
      type: Schema.Types.ObjectId,
      ref: 'Pharmacy',
```

```
    },
});

UserSchema.plugin(passportLocalMongoose);
module.exports= mongoose.model("User",UserSchema);
```

Αρχικά εισάγουμε τις απαραίτητες βιβλιοθήκες την mongoose και την passportLocalMongoose και δημιουργούμε το schema με τις απαραίτητες μεταβλητές. Παρατηρούμε ότι κάποιες μεταβλητές έχουμε ορίσει τύπο δεδομένων και προεπιλεγμένο value.

Στο πεδίο pharmacy, αποθηκεύουμε ένα Object ID που αντιστοιχεί σε ένα άλλο model με το όνομα Pharmacy. Αυτό είναι γνωστό ως σύνδεση μεταξύ models στην Mongoose και χρησιμοποιείται για να δημιουργήσουμε μια σχέση μεταξύ των δύο "models". Το User μπορεί να συσχετιστεί με ένα Pharmacy μέσω του Object ID του. Αυτό μπορεί να χρησιμοποιηθεί για να δείξει ποιο κοινωνικό φαρμακείο έχει/διαχειρίζεται ο χρήστης χρησιμοποιώντας το .populate().

Τέλος, το UserSchema.plugin(passportLocalMongoose) απλός χρησιμοποιεί το plugin passportLocalMongoose του Mongoose για να προσθέσει λειτουργικότητα στο "model" του χρήστη (User). Η λειτουργικότητα που χρησιμοποιήθηκε είναι η αυθεντικοποίηση του χρήστη με την χρήση του passport.js όπου αφήνουμε την βιβλιοθήκη την αποθήκευση του username και του κρυπτογραφημένου κωδικού. Σχετικά με την αυθεντικοποιήση θα μιλήσουμε περισσότερο στο υπό - κεφάλαιο 3.4.2.

Για να δούμε πως ακριβώς η σχέση μεταξύ των δύο model, ας δούμε και το model pharmacy.

```
const mongoose = require('mongoose');
const Schema=mongoose.Schema;


const ImageSchema=new Schema({
   url:String,
   filename:String
})

const PharmacySchema=new Schema({
   name:String,
   streetAddress:String,
   streetNumber:Number,
   phoneNumber:Number,
   city:String,
   area:String,
   postalCode:Number,
   email:String,
   coordinates:{
      type:[Object]

   },
   isVerified:{
```

```
    type:Boolean,
    default:false
  },
  images:[ImageSchema],
  warehouse: {
    type: Schema.Types.ObjectId,
    ref: 'Warehouse',
   },

})

module.exports=mongoose.model("Pharmacy",PharmacySchema)
```

Αρχικά, φορτώνουμε τις απαραίτητες βιβλιοθήκες. Στη συνέχεια, ορίζουμε ένα ImageSchema που αντιπροσωπεύει την εικόνα του κοινωνικού φαρμακείου και έχει δύο πεδία, το url που είναι αποθηκευμένη η εικόνα στο cloudinary και σε ποιο αρχείο θα αποθηκευτεί.

Στη συνέχεια, ορίσαμε το βασικό PharmacySchema, το οποίο περιέχει πληροφορίες σχετικά με ένα κοινωνικό φαρμακείο, όπως το όνομα, η διεύθυνση, το τηλέφωνο, κλπ. Επίσης, αναφερόμαστε σε άλλο μοντέλο, όπως το Warehouse, χρησιμοποιώντας τα Schema.Types.ObjectId. Το οποίο με την σειρά του περιέχει την αποθήκη των φαρμακείων με τα ονόματα των φαρμάκων, κατασκευαστή, ποσότητα και αν χρειάζεται συνταγή ή όχι.

3.3 Υλοποίηση του frontend

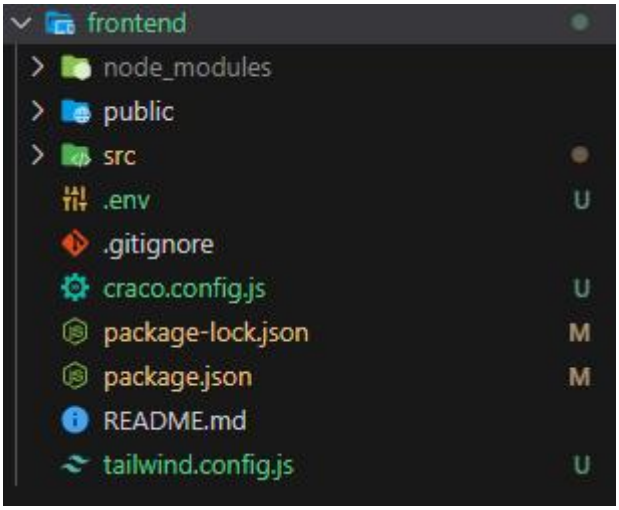
Στα πλαίσια του συγκεκριμένου υπό-κεφαλαίου θα εξηγήσουμε αναλυτικά πως έχουν υλοποιηθεί όλα τα βασικά στοιχεία από τα οποία αποτελείται το frontend. Για να διαχωρίσουμε την λειτουργικότητα του frontend με του backend, φτιάχνουμε δύο φακέλου frontend και backend και κατευθυνόμαστε μέσα στον φάκελο frontend.

Για την δημιουργία του project, πληκτρολογούμε στο terminal με την εντολή :

npm install create-react-app thesis_app

Το αποτέλεσμα εκτέλεσης της πάνω εντολής είναι η δημιουργία ενός φακέλου με όνομα thesis_app το οποίο με την σειρά του δημιουργεί κάποια αρχεία όπως το public, src και κάποια configurations.

Σχήμα 3.3.1: Οι δύο Φάκελοι frontend και backend



Σχήμα 3.3.2: Ο Φάκελος frontend.

Μερικές αναφορές για τα αρχεία που δημιουργήθηκαν είναι:

src: Αυτός είναι ο φάκελος όπου θα αναπτύξουμε τον κώδικα της εφαρμογής μας.
public/index.html: Το κύριο HTML αρχείο της εφαρμογής.
package.json: Το αρχείο που περιέχει τις εξαρτήσεις (dependencies) της εφαρμογής και τις ρυθμίσεις.
node_modules: Αυτός είναι ο φάκελος που περιέχει όλα τα πακέτα (packages) που χρειάζονται για το project. Τα πακέτα αυτά δεν περιλαμβάνονται, αλλά εγκαθίστανται τοπικά με βάση το package.json.

Για να μπορούμε να τρέξουμε το project της react σε local host, χρειάζεται να πάμε στο terminal, να κατευθυνθούμε στον φάκελο που είναι το frontend project και να γράψουμε την εντολή:

npm start

Προτού ξεκινήσουμε, θα αναφέρουμε δυο εργαλεία που παρέχονται μαζί με την React και ονομάζονται React Context και hooks. το React Context αφορά το state όπου μας επιτρέπει να μοιράζουμε άμεσα δεδομένα και να είναι προσβάσιμα σε όλα τα components αποφεύγοντας να χρησιμοποιήσουμε τα props κατά βάση της ιεραρχίας των components (από το component πατέρα σε παιδί).

Το αρχείο βρίσκεται στον φάκελο context με όνομα ContextProvider.js

```
import { createContext, useContext, useState } from "react";
const StateContext = createContext();
export const ContextProvider = ({ children }) => {
  const [activeMenu, setActiveMenu] = useState(true);
  const [isClicked, setIsClicked] = useState(initialState);
  const [screenSize, setScreenSize] = useState(undefined)
  const [selectedProduct, setSelectedProduct] = useState(null);

  const handleProductClick = (productName) => {
     setSelectedProduct(productName);
   };
  const getProductName=()=>{
```

```
  return selectedProduct
 }
 const getToggleTrue=()=>{
  return true
 }

  return (
    <StateContext.Provider
      value={{
        activeMenu,
        setActiveMenu,
        isClicked,
        setIsClicked,
        handleClick,
        screenSize,
        setScreenSize,
        selectedProduct,
        handleProductClick,
        getProductName,
        getToggleTrue
      }}
    >
      {children}
    </StateContext.Provider>
  );

};
export const useStateContext = () => useContext(StateContext);
```

Στα πλαίσια του React context πρέπει να αναφέρουμε δύο έννοιες το provider και το context. Ο provider αναφέρεται σε ένα στοιχείο που παρέχει τις καταστάσεις ή τα δεδομένα σε όλα τα παιδιά του που βρίσκονται μέσα σε αυτό και το context είναι ένα μηχανισμός που επιτρέπει τη μετάδοση δεδομένων από ένα στοιχείο σε όλα τα παιδιά του. Στην δική μας περίπτωση καλύψαμε με τον provider όλο το project στο αρχείο index.js. Τα επιπλέον πεδία του context εξηγούνται παρακάτω:

StateContext: Δημιουργείται το React context με τη χρήση της συνάρτησης createContext. Αυτό το context θα χρησιμοποιηθεί για να μοιράζεται το state μεταξύ διαφορετικών components.

activeMenu: Μια μεταβλητή που χρησιμοποιείται για τον έλεγχο της ενεργοποίησης/απενεργοποίησης του μενού.

isClicked: Ένα αντικείμενο που χρησιμοποιείται για την αποθήκευση διαφόρων καταστάσεων με κλικ.

screenSize: Μια μεταβλητή που κρατάει το μέγεθος της οθόνης.

selectedProduct: Μια μεταβλητή που δείχνει το επιλεγμένο προϊόν.

handleProductClick: Μια συνάρτηση που χρησιμοποιείται για τον χειρισμό του κλικ σε ένα προϊόν.

getProductName: Μια συνάρτηση που επιστρέφει το όνομα του επιλεγμένου προϊόντος.

getToggleTrue: Μια συνάρτηση που επιστρέφει την τιμή true.

Τα hooks από την άλλη, παρέχουν έναν εύκολο τρόπο οργάνωσης του κώδικα, διαχείρισης της κατάστασης και εκτέλεσης λειτουργιών που σχετίζονται με τον κύκλο ζωής του component.

Παρακάτω βρίσκονται τα βασικότερα hooks που έχουμε χρησιμοποιήσει :

useContext: Το useContext χρησιμοποιείται για την πρόσβαση και καταχώρηση τιμών ενός React context.

useState: Το useState χρησιμοποιείται για τη δήλωση μιας μεταβλητής κατάστασης (state) σε ένα component. Επιστρέφει έναν πίνακα με δύο στοιχεία. Την τρέχουσα τιμή της state και μια συνάρτηση για την τροποποίησή της.

useEffect: Το useEffect χρησιμοποιείται για τη διαχείριση side effects σε ένα component. Στην ουσία, εκτελείτε κάθε φορά αναπαράστασης ενός component. Το όρισμα [] είναι ένας πίνακας εξαρτήσεων (dependencies) και καθορίζει πότε θα εκτελεστεί το useEffect. Αν ο πίνακας είναι κενός ([]), το useEffect θα εκτελεστεί μόνο μια φορά κατά την αρχικοποίηση του component. Αν περιλαμβάνει εξαρτήσεις, το useEffect θα εκτελεστεί κάθε φορά που μία από τις εξαρτήσεις αλλάξει. Στην περίπτωση μας, το render εξαρτάτε από μια συνάρτηση με όνομα getProductName. Το getProduct name.

3.3.1 Pharmacies

Μέσα στον φάκελο src/pages υπάρχει το αρχείο pharmacies.jsx. Το pharmacies page είναι η πρώτη σελίδα που απεικονίζεται κατά την εκκίνηση της εφαρμογής. Πριν αναφερθούμε σε αυτό που ακριβώς κάνει, θα εξετάσουμε τα βασικά components από τα οποία αποτελείται.

SearchBar

Αρχικά αποτελείτε από ένα SearchBar όπου η υλοποίηση του γίνεται στον φάκελο components με όνομα searchbar.jsx και η λειτουργικότητα είναι να επιστρέφει τα προϊόντα ανάλογα με το όνομα τους ή την δραστική τους ουσία.

```
import React, { useEffect, useState,useRef } from "react";
import axios from "axios";
import { useStateContext } from "../contexts/ContextProvider";

function SearchBar({ onSearchResults,onQuery }) {
  const [searchQuery, setSearchQuery] = useState("");
  const {getProductName,handleProductClick}=useStateContext();
  const handleSearch = async (e) => {
    e.preventDefault();
    try {
      const response = await axios.get(`/products/searchproducts?q=${searchQuery}`);
      onSearchResults(response.data);
      onQuery(searchQuery)
    } catch (error) {
      console.error(error);
    }
  };
  useEffect(()=>{
    if (getProductName() !== null) {
```

```
        setSearchQuery(getProductName());
    handleProductClick(null)
   }
  },[getProductName])
  return (
   <>
   <div className="flex justify-center">
   <form  className= " w-1/2" onSubmit={handleSearch}>
   <label htmlFor="default-search" className="mb-2 text-sm font-medium text-gray-900 sr-only
dark:text-white">Search</label>
   <div className="relative">
     <div className="absolute inset-y-0 left-0 flex items-center pl-3 pointer-events-none">
       <svg className="w-4 h-4 text-gray-500 dark:text-gray-400" aria-hidden="true"
xmlns="http://www.w3.org/2000/svg" fill="none" viewBox="0 0 20 20">
          <path stroke="currentColor" strokeLinecap="round" strokeLinejoin="round"
strokeWidth="2" d="m19 19-4-4m0-7A7 7 0 1 1 1 8a7 7 0 0 1 14 0Z"/>
       </svg>
     </div>
     <input  value={searchQuery}  onChange={(e) => setSearchQuery(e.target.value)} type="search"
id="default-search" className="block w-full p-4 pl-10 text-sm text-gray-900 border border-gray-300
rounded-lg bg-gray-50 focus:ring-green-500 focus:border-green-500 dark:bg-gray-700 dark:border-
gray-600 dark:placeholder-gray-400 dark:text-white dark:focus:ring-green-500 dark:focus:border-
green-500" placeholder="Search for products or active substance.."/>
     <button type="submit" className="text-white absolute right-2.5 bottom-2.5 bg-green-400
hover:bg-green-600 focus:ring-4 focus:outline-none focus:ring-green-300 font-medium rounded-lg
text-sm px-4 py-2 dark:bg-green-600 dark:hover:bg-green-700 dark:focus:ring-green-
800">Search</button>
   </div>
</form>
   </div>
   </>
  );
}
export default SearchBar;
```

Ο χρήστης αφού πατήσει το κουμπί search, εκτελεί την συνάρτηση handleSearch με την μεταβλητή searchQuery που περιέχει το input του χρήστη και κάνει μια κλήση προς το API έτσι ώστε να φιλτράρει και να επιστρέψει το αποτέλεσμα. Τέλος, το αποτέλεσμα επιστρέφεται στο αρχικό Parent component με δυο μεταβλητές, την onSearchResult που περιέχει το αποτέλεσμα και την onQuery που περιέχει την λέξη που αναζήτησε ο χρήστης.

Επιπλέον, στο useEffect, υπάρχει έλεγχος για το αν η μεταβλητή που προέρχεται από τη συνάρτηση getProductName, η οποία λαμβάνεται από το useContext, δεν είναι κενή. Σε αυτή την περίπτωση, η μεταβλητή searchQuery ορίζεται με την τιμή της, προκειμένου να εμφανίζεται στο πεδίο εισαγωγής (input). Στη συνέχεια, με τη χρήση της συνάρτησης handleProductClick, η μεταβλητή getProductName αρχικοποιείται σε null.

Ο λόγος που γίνεται αυτός ο έλεγχος, όπως εξηγήσαμε στο κεφάλαιο 3.1.2, είναι ότι ο χρήστης έχει τη δυνατότητα, από το ιστορικό κρατήσεων/παραγγελιών, να επιλέξει ένα προϊόν από τις προηγούμενες παραγγελίες/κρατήσεις και να τον ανακατευθύνει στο σύστημα ώστε να το ξανά παραγγείλει. Ουσιαστικά, από τη σελίδα ιστορικού παραγγελιών χρήστη, θέλουμε να αποθηκεύσουμε τη μεταβλητή με το όνομα του προϊόντος και να το εμφανίσουμε στο input του searchBar. Αφού το

εμφανίσουμε στο searchBar, χρειαστήκαμε έναν έλεγχο ώστε να μην τρέχει κάθε φορά αναζήτηση για αυτό το όνομα του προϊόντος, και έτσι αποφασίσαμε να κάνουμε έλεγχο κάθε φορά στη μεταβλητή της συνάρτησης, αν είναι κενή, υποδηλώνοντας ότι κανείς έχει επιλέξει προϊόν για παραγγελία.

PharmacySearchBar

Πέρα από το component του searchBar.jsx που περιγράψαμε προηγουμένως, η εφαρμογή περιλαμβάνει και ένα δεύτερο search Bar με το όνομα PharmacySearchBar.jsx. Στην ουσία, αυτό το search Bar επιτρέπει στον χρήστη να κάνει αναζήτηση για ονόματα κοινωνικών φαρμακείων ή περιοχές. Είναι σημαντικό να σημειωθεί ότι τα δύο αυτά search Bar δεν είναι ορατά παράλληλα στον χρήστη, αλλά αλλάζουν ανάλογα με ένα κουμπί toggle. Ο λόγος ύπαρξης και των δύο είναι ότι το searchBar.jsx και το PharmacySearchBar.jsx κάνουν διαφορετικές κλήσεις προς το API. Αξίζει να σημειωθεί ότι το αρχείο PharmacySearchBar.jsx δεν παρουσιάζει αισθητικές διαφορές ή λειτουργικές από το searchBar.jsx, εκτός από την κλήση προς το API,του μηνύματος στο placeholder και της απουσίας της λειτουργικότητας του useEffect.

Pharmacies

Μετά από την περιγραφή των βασικών components που αποτελούν το αρχείο pharmacies.jsx, ήρθε η στιγμή να προχωρήσουμε στην εξήγησή του. Λόγω του μεγάλου μεγέθους του αρχείου, θα διαχωρίσουμε την εξήγησή του σε μικρότερα κομμάτια.

```
import { React, useEffect, useState } from "react";
import axios from "axios";
import HeadMap from "../components/HeadMap";
import Header from "../components/Header";
import CardTemplate from "../components/CardTemplate";
import { useSelector } from "react-redux";
import SearchBar from "../components/SearchBar";
import AlertPharmacy from "../components/alertComponents/AlertPharmacy";
import AlertUser from "../components/alertComponents/AlertUser";
import { useStateContext } from "../contexts/ContextProvider";
import ProductsTable from "../components/ProductsTable";
import PharmacySearchbar from "../components/PharmacySearchbar";
```

Αρχικά, εισάγουμε τα απαραίτητα components και βιβλιοθήκες που χρειαζόμαστε, αναφέροντας έκαστο από αυτά.

HeadMap: Ένα component που περιλαμβάνει όλη τη λειτουργικότητα του κύριου χάρτη.
Header: Ένα component που καλείται σχεδόν σε όλες τις σελίδες της εφαρμογής. Δεν περιέχει παρά μόνο λίγο κώδικα του Tailwind CSS και HTML.
CardTemplate: Ένα component που χρησιμοποιείται για την εμφάνιση των στοιχείων κάθε κοινωνικού φαρμακείου σε ολόκληρη την εφαρμογή.
ProductsTable: Ένα component που χρησιμοποιείται για την εμφάνιση μ όλων των προϊόντων στην εφαρμογή μέσο ενός table.
AlertUser και AlertPharmacy: Δύο απλά components που περιέχουν μηνύματα ειδοποίησης για να ενημερώνουν τον χρήστη ότι δεν έχει επαληθευτεί ο λογαριασμός του ή το κοινωνικό του φαρμακείο

αντίστοιχα.Επιπλέον το AlertUser περιέχει ένα κουμπί ώστε να γίνει μια κλήση προς το API για αποστολή email.
userSelector: Μια συνάρτηση από τη βιβλιοθήκη react-redux που κρατά τα δεδομένα του συνδεδεμένου χρήστη.
Header: Ένα component που περνάμε το όνομα της σελίδας, περιέχει tailwind για την σχεδίαση του.

Έπειτα ορίσαμε μερικές μεταβλητές.

```
const userData = useSelector((state) => state.session.userData);
const pharmacyData = useSelector((state) => state.session.pharmacyData);
const [allPharmacies, setAllPharmacies] = useState();
const [allProducts, setAllProducts] = useState();
const [toggle, setToggle] = useState(false);
const {getProductName}=useStateContext();
const [searchResults, setSearchResults] = useState([]);
const [query, setQuery] = useState("");
const [searchResultsPharmacy, setSearchResultsPharmacy] = useState([]);
const [queryPharmacy, setQueryPharmacy] = useState("");
```

Τα userData και pharmacyData χρησιμοποιούν τη συνάρτηση useSelector για να επιλέξουν τα δεδομένα χρήστη και διαχειριστή κοινωνικού φαρμακείου από το Redux store και τα αποθηκεύουν στις μεταβλητές userData και pharmacyData αντίστοιχα.

Τα allPharmacies και allProducts καθορίζουν μεταβλητές κατάστασης και αποθηκεύουν τα αποτελέσματα των κλήσεων των API για την αποθήκευση όλων των κοινωνικών φαρμακείων και προϊόντων αντίστοιχα.

Το searchResults και query αποθηκεύουν το αποτέλεσμα της αναζήτησης των προϊόντων από το searchBar.jsx και το όνομα του προϊόντος που αναζητήθηκε.

Το searchResultsPharmacy και queryPharmacy αποθηκεύουν το αποτέλεσμα της αναζήτησης των φαρμακείων από το PharmacySearchBar.jsx και το όνομα του φαρμακείου που αναζητήθηκε.

Το Toggle είναι μια μεταβλητή κατάστασης που αλλάζει σε true ή false ανάλογα με την προηγούμενη κατάσταση του.

```
const handleToggle = () => {
   setToggle((prevToggle) => !prevToggle);
 };
 const handleSearchResults = (results) => {
   setSearchResults(results);
 };
 const handleCharacters = (chars) => {
   setQuery(chars);
 };
 const handleSearchResultsPharmacy = (results) => {
   const pharmacy=results.pharmacies
```

```
  setSearchResultsPharmacy(pharmacy);
 };
 const handleCharactersPharmacy = (chars) => {
  setQueryPharmacy(chars);
 };
```

Το handleSearchResults και setSearchResults είναι δύο συναρτήσεις που παίρνουν τα δεδομένα από το searchbar.jsx και τα αποθηκεύουν στο pharmacies.jsx. Αντίστοιχη λογική το handleSearchResultsPharmacy και handleCharactersPharmacy. Η συνάρτηση HandleToggle απλά μεταβάλει από true σε false.

```
 const getDisplayedMapPharmacies = () => {
   if (!queryPharmacy) {
        return allPharmacies;
      } else if (searchResultsPharmacy.length > 0) {
        return searchResultsPharmacy;
      } else if (searchResultsPharmacy.length === 0 && queryPharmacy) {
        return [];
      }
 };
 const getDisplayedMapProducts = () => {
   const extractedPharmacyData = searchResults.map((item) => {
    return item.pharmacies.map((pharmacy) => pharmacy.pharmacyData);
   });
   if (!query) {
    return allPharmacies;
   } else if (searchResults.length > 0) {
    return extractedPharmacyData.flat();
   } else if (searchResults.length === 0 && query) {
    return [];
   }
   extractedPharmacyData.map((item) => {
    return item;
   });
 };
const getDisplayedProducts = () => {
   if (!query) {
    return allProducts;
   } else if (searchResults.length > 0) {
    return searchResults;
   } else if (searchResults.length < 0 && query) {
    return [];
   }
 };
 const displayedProducts = getDisplayedProducts();
 const displayedMapProducts = getDisplayedMapProducts();
 const displayedMapPharmacies = getDisplayedMapPharmacies();
```

Η συνάρτηση getDisplayedMapProducts αναλαμβάνει να εξάγει τα δεδομένα των κοινωνικών φαρμακείων από τα αποτελέσματα της αναζήτησης. Αν δεν έχει πραγματοποιηθεί αναζήτηση, επιστρέφει όλα τα κοινωνικά φαρμακεία. Αν υπάρχουν αποτελέσματα αναζήτησης, επιστρέφει τα δεδομένα των κοινωνικών φαρμακείων που προέκυψαν από την αναζήτηση. Σε περίπτωση που δεν υπάρχουν αποτελέσματα αναζήτησης και υπάρχει η μεταβλητή query (δηλαδή ο χρήστης προσπάθησε να κάνει αναζήτηση), επιστρέφει έναν κενό πίνακα. Τα αποτελέσματα θα τα χρειαστούμε γιατί περιέχουν συντεταγμένες ώστε να βάλουμε και αντίστοιχα markers στους χάρτες.

Η συνάρτηση getDisplayedMapProducts έχει παρόμοια λειτουργικότητα με την getDisplayedProducts, αλλά αναλαμβάνει την αναζήτηση για τα προϊόντα, στο backend συσχετίζουμε κάθε προϊόν που αναζητούμε με το αντίστοιχο κοινωνικό φαρμακείο που ανήκει. Κάθε προϊόν μπορεί να ανήκει σε περισσότερα από ένα κοινωνικά φαρμακεία. Στο response της API κλήσης, τα δεδομένα που επιστρέφονται περιλαμβάνουν ένα αντικείμενο με το όνομα pharmacyData, το οποίο περιέχει τα στοιχεία του φαρμακείου που σχετίζεται με το συγκεκριμένο προϊόν. Με την συνάρτηση item.pharmacies.map((pharmacy) => pharmacy.pharmacyData); κρατάμε μόνο τα στοιχεία του που αφορά το κοινωνικά φαρμακεία. Τέλος, οι τρεις παρακάτω συναρτήσεις οριστικοποιούνται σε αντίστοιχες μεταβλητές.

```
useEffect(() => {
  const allPharmaciesFunc = async () => {
    try {
      const response = await axios.get("/pharmacies");
      setAllPharmacies(response.data);
    } catch (e) {
      return "Seems to be an error";
    }
  };
  const allProductsFunc = async () => {
    try {
      const response = await axios.get("/products");
      setAllProducts(response.data);
        } catch (e) {
      return "Seems to be an error";
    }
  };
  allProductsFunc();
  allPharmaciesFunc();
}, []);
```

Η useEffect συνάρτηση δεν κάνει τίποτα παρά μόνο να χτυπάει στα αντίστοιχα Endpoints για να επιστρέφει όλα τα κοινωνικά φαρμακεία και όλα τα προϊόντα της κατά την εκκίνηση της εφαρμογής και τα αποθηκεύει στις αντίστοιχες μεταβλητές κατάστασης setAllProducts και setAllPharmacies.

```
return (
  <div className="mt-2">
    {userData?.isVerified === false && <AlertUser userData={userData} />}
    {pharmacyData?.pharmacy?.isVerified === false && <AlertPharmacy />}
```

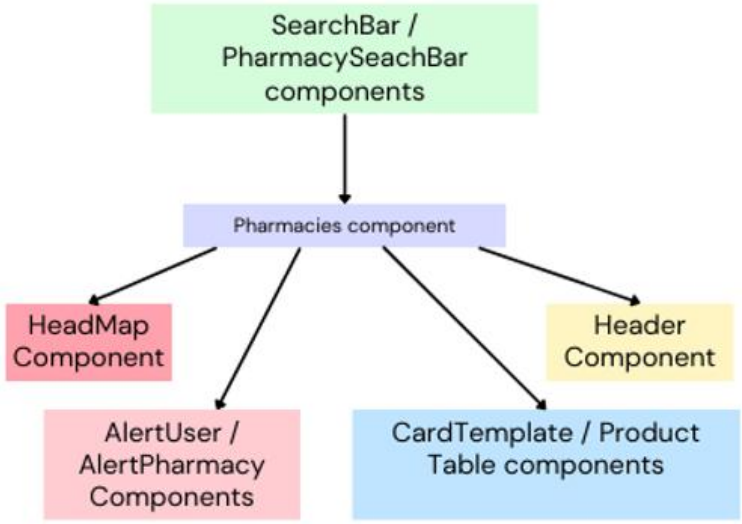
```
      {toggle ? (  <HeadMap allPharmacies={displayedMapProducts} />):(  <HeadMap
        allPharmacies={displayedMapPharmacies} />)}
    <div className="ml-10">
     <Header title="All Pharmacies" />
    </div>
    {toggle ? (
     <SearchBar
      onSearchResults={handleSearchResults}
      onQuery={handleCharacters}
     />
    ) : (
     <PharmacySearchbar
      onSearchResults={handleSearchResultsPharmacy}
      onQuery={handleCharactersPharmacy}
     />
    )}
    <button
     onClick={handleToggle}
          className="m-10 text-white bottom-2.5 bg-green-400 hover:bg-green-600 focus:ring-4
focus:outline-none focus:ring-green-300 font-medium rounded-lg text-sm px-4 py-2"
    >
     {toggle ? "Toggle to Pharmacies" : "Toggle to Products"}
    </button>
    {toggle ? (
       <div className="flex justify-center"> <ProductsTable allProducts={displayedProducts} />
</div>
    ) : (
     <CardTemplate allPharmacies={displayedMapPharmacies} />
    )}
   </div>
  );
}
export default Pharmacies;
```

Εδώ είναι το τελευταίο μέρος της σελίδας pharmacies και ίσως είναι και το πιο ενδιαφέρον. Αρχικά, ελέγχουμε αν τα στοιχεία του συνδεδεμένου χρήστη δεν έχουν επαληθευτεί από το Redux store. Αν είναι έτσι, εμφανίζεται το αντίστοιχο μήνυμα που περιλαμβάνει το component AlertUser. Όπως αναφέραμε συνοπτικά, το AlertUser περιέχει ένα μήνυμα και ένα κουμπί για να κάνει κλήση προς το API για την αποστολή μηνύματος επαλήθευσης email. Για να υποδείξουμε σε ποιον χρήστη θα γίνει αυτή η αποστολή email, χρησιμοποιήσαμε το userId μέσα στην API κλήση ώστε να κάνει την ανάλογη ανάθεση.

Για να μπορέσουμε να χρησιμοποιήσουμε το userId του χρήστη στο AlertUser component, περνάμε τα στοιχεία του συνδεδεμένου χρήστη στο component. Από την άλλη πλευρά, η ίδια λογική ισχύει και για το AlertPharmacyComponent, το οποίο περιέχει μόνο ένα μήνυμα. Στη συνέχεια, βλέπουμε το toggle. Αν το toggle είναι false, θα εμφανιστεί το searchBar και το component productTable που αφορά τα προϊόντα. Ενώ αν είναι true, θα εμφανίσει το pharmacySearchBar και το component CardTemplate που αφορά τα κοινωνικά φαρμακεία.

Το toggle κάνει επιπλέον έλεγχο που αφορά τον χάρτη. Στην περίπτωση που το toggle είναι false και εμφανίζονται τα components που αφορούν τα προϊόντα, περνά τα δεδομένα που αποθηκεύσαμε στη μεταβλητή displayMapProducts ώστε να εμφανίσει τα αντίστοιχα markers που ανήκουν σε κάθε κοινωνικό φαρμακείο που διαθέτει τα προϊόντα. Αντίστοιχα, στην περίπτωση που το toggle είναι true και εμφανίζονται τα components που αφορούν τα κοινωνικά φαρμακεία, περνά τη μεταβλητή displayMapPharmacies που περιέχει τα στοιχεία των κοινωνικών φαρμακείων .

Στην περίπτωση των search bar components, λαμβάνουμε τα δεδομένα των αποτελεσμάτων αναζήτησης από εκεί. Στη συνέχεια, πραγματοποιούμε τον ανάλογο έλεγχο στο component pharmacies, και όσον αφορά το CardTemplate και το ProductsTable, περνάμε τα δεδομένα για να διαχειριστούμε την εμφάνιση και τη λειτουργικότητα αντίστοιχα.



Σχήμα 3.3.3 Συσχέτιση σελίδας pharmacies με τα υπόλοιπα components.

3.3.2 HeadMap

Στο προηγούμενο υπό-κεφάλαιο, αναφερθήκαμε στον βασικό χάρτη της εφαρμογής και τον τρόπο με τον οποίο περνούσαμε δεδομένα σε αυτό το component από το pharmacies component, προκειμένου να εμφανίσουμε τα αντίστοιχα markers. Για την εμφάνιση των markers, απαιτούνται γεωγραφικό μήκος και πλάτος (longitude και latitude). Από όλα τα δεδομένα που λαμβάνει αυτό το component, θα απομονώσουμε ορισμένα για τη χρήση τους. Παρακάτω παρατίθεται ο αντίστοιχος κώδικας.

Αρχικά εισάγουμε τις βιβλιοθήκες του MapBox και μας επιστρέφει κάποιες συναρτήσεις / components.

import ReactMapGL, { Marker, Popup } from "react-map-gl";

Το ReactMapGL είναι ο χάρτης, το Marker είναι τα markers στον χάρτη και το popup ένα λευκό popup με κάποιο μήνυμα όταν πατήσει ο χρήστης πάνω στο marker.

```
const [selectedMarker, setSelectedMarker] = useState(null);
const [showPopup, setShowPopup] = useState(false);

<ReactMapGL
      mapboxAccessToken=MapBoxToken

      initialViewState={{
        longitude: 24.9464951342173,
        latitude: 38.5397368408087,
        zoom: 4.6,
      }}
          mapStyle="mapbox://styles/mapbox/streets-v9"
    >
      {allPharmacies ? (
       allPharmacies.map((item) => (
         <div
          key={item._id}
          onClick={() => {
           setSelectedMarker(item);
           setShowPopup(true);
          }}
         >
          <Marker
           key={item._id}
           longitude={item.coordinates[0].lng}
           latitude={item.coordinates[0].lat}
           onClick={() => {
            setSelectedMarker(item);
            setShowPopup(true);
           }}
          ></Marker>
         </div>
       ))
      ) : (
       <h1>Loading..</h1>
      )}

      {showPopup && selectedMarker && (
       <Popup
         longitude={selectedMarker.coordinates[0].lng}
         latitude={selectedMarker.coordinates[0].lat}
         closeButton={true}
         closeOnClick={false}
         onClose={() => setShowPopup(false)}
         anchor="bottom"
         offset={{ bottom: [0, -20] }}
       >
```

```
          <div>
            <p className="font-semibold">{selectedMarker.name}</p>
            <p>{selectedMarker.streetAddress},{selectedMarker.streetNumber}</p>
            <p className=" cursor-pointer underline text-blue-600" onClick={() =>
navigate(`/pharmacies/${selectedMarker._id}`)}> View</p>
          </div>
        </Popup>
      )}
    </ReactMapGL>
```

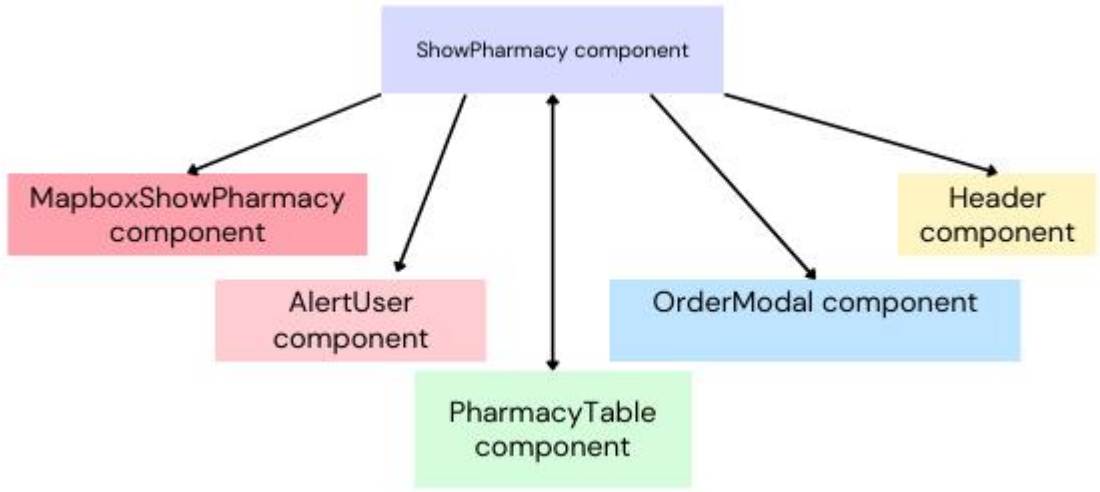
Οι επιλογές που ορίστηκαν στο χάρτη είναι οι εξής.

initialViewState: Ορίζει ποιο σε πιο γεωγραφικό μήκος,πλάτος και ύψος θα παρουσιάζει ο χάρτης.
mapStyle: Ορίζει το είδος και το στυλ του χάρτη το οποίο παρέχετε από την ιστοσελίδα του Mapbox.

Μετά από τη λήψη των δεδομένων από τη σελίδα pharmacies και την αποθήκευσή τους στη μεταβλητή allPharmacies, χρησιμοποιούμε τη μέθοδο map για να δημιουργήσουμε έναν marker για κάθε αντικείμενο (κοινωνικό φαρμακείο) που περιέχεται στο allPharmacies. Έτσι, για κάθε κοινωνικό φαρμακείο, δημιουργείται ένα νέο marker και παρέχεται λειτουργικότητα σε περίπτωση που γίνει κλικ πάνω του. Στην περίπτωση του popup, γίνεται έλεγχος για το αν έχει πατηθεί το marker με τιμή true και αν η μεταβλητή selectedMarker περιέχει δεδομένα. Η selectedMarker περιέχει τα δεδομένα του επιλεγμένου marker και τα μεταβιβάζει στο component "popup", που περιέχει κώδικα για την εμφάνιση των δεδομένων. Επιπλέον, το component περιλαμβάνει ένα κουμπί view, το οποίο κατευθύνει τον χρήστη σε μια σελίδα showPharmacy. Αυτή η σελίδα θα εξηγηθεί στο επόμενο υπό - κεφάλαιο.

3.3.3 ShowPharmacy

Η σελίδα showPharamacy περιέχει δυναμικά τις λεπτομέρειες ενός κοινωνικού φαρμακείου, τα διαθέσιμα προϊόντα που έχει, δυνατότητα κράτησης/παραγγελίας των προϊόντων αυτόν και έναν χάρτη ο οποίος δείχνει γεωγραφικά που βρίσκεται. Οι λειτουργίες αυτής της σελίδας, χωρίστηκαν σε κάποια components όπως φαίνεται παρακάτω στο σχήμα 3.3.3.



Σχήμα 3.3.4 Συσχέτιση σελίδας showPharmacy με τα άλλα components.

Αρχικά, θα πρέπει να εξετάσουμε τα component με τα οποία το showPharmacy έρχεται σε άμεση επαφή. Το AlertUser και το Header διαθέτουν ακριβώς την ίδια λειτουργικότητα, όπως αναλύεται στο κεφάλαιο 3.3.1. Αντίστοιχα, το MapboxShowPharmacy component μοιάζει λειτουργικά με το HeadMap component, όπως περιγράφεται στο κεφάλαιο 3.3.2. Η μόνη διαφορά είναι ότι, αντί να εμφανίζουμε όλα τα marker για όλα τα κοινωνικά φαρμακεία, παρουσιάζουμε μόνο ένα.Ένα επιπλέον component που προστίθεται είναι το OrderModal. Αυτό το component περιέχει ένα μήνυμα που επιβεβαιώνει με δύο επιλογές, Confirm ή Cancel, την οριστικοποίηση της κράτησης.

PharmacyTable

Όσον αφορά το pharmacyTable, έχει αμφίδρομη αλληλεπίδραση με τα στοιχεία του showPharmacy και βρίσκεται στο αρχείο components/pharmaciesComponenets. Λαμβάνει δεδομένα από το showPharmacy σχετικά με τα προϊόντα, προκειμένου να τα διαχειριστεί και να τα παρουσιάσει εκεί. Παράλληλα, αποστέλλει δεδομένα που αφορούν έναν μετρητή, ο οποίος αναφέρεται στην ποσότητα των προϊόντων που έχουν επιλεγεί.

function PharmacyTable({ products, counters, setCounters }) {}

Στην δημιουργία του component βάλαμε 3 παραμέτρους, το products που περιέχει όλα τα στοιχεία των προϊόντων, το counters που είναι ο μετρητής και η setCounters που είναι η συνάρτηση ώστε να αναθέτει νέα τιμή για την μεταβλητή κατάσταση counters,

```
const getBadgeColor = (quantity) => {
  if (quantity <= 10) return "bg-red-500 text-white";
  if (quantity <= 20) return "bg-yellow-500 text-black";
  return "bg-green-500 text-white text-sm";
};
const getBadgeText = (quantity) => {
  if (quantity <= 10) return "Low";
  if (quantity <= 20) return "Mid";
  return "High";
};
```

Από όλα τα στοιχεία των προϊόντων, εξάγαμε το πεδίο quantity, το οποίο χρησιμοποιούμε ως μέτρο για την ποσότητα των προϊόντων. Δεν επιθυμούμε την εμφάνιση της πραγματικής ποσότητας των προϊόντων λόγω ευαίσθητης πληροφορίας. Για τον λόγο αυτό, προβαίνουμε σε ορισμένους ελέγχους, ώστε να εμφανίζεται κατάλληλο μήνυμα και χρώμα.Συγκεκριμένα, αν η ποσότητα των προϊόντων είναι κάτω από δέκα, το σύστημα εμφανίζει τη λέξη Low και χρησιμοποιεί ένα κόκκινο χρώμα. Εάν η ποσότητα είναι ανάμεσα σε δέκα και είκοσι, το μήνυμα είναι Mid και το χρώμα κίτρινο. Τέλος, εάν η ποσότητα είναι πάνω από είκοσι, εμφανίζεται το μήνυμα High με ένα πράσινο χρώμα.

```
const handleIncrement = (productId) => {
  setCounters((prevCounters) => ({
    ...prevCounters,
    [productId]: (prevCounters[productId] || 0) + 1,
  }));
};
const handleDecrement = (productId) => {
  if (counters[productId] > 0) {
    setCounters((prevCounters) => {
```

```
      const updatedCounters = { ...prevCounters };
      updatedCounters[productId] = prevCounters[productId] - 1;

      if (updatedCounters[productId] === 0) {
        delete updatedCounters[productId];
      }
      return updatedCounters;
    });
  }
};
```

Κάθε προϊόν στον πίνακα παρουσιάζει δύο κουμπιά, ένα θετικό (+) και ένα αρνητικό (-), που επιτρέπουν την προσθήκη και τη μείωση της ποσότητας του προϊόντος αντίστοιχα. Για κάθε προϊόν, η συνάρτηση handleIncrement λαμβάνει ως παράμετρο το id του προϊόντος και αυξάνει την ποσότητα του συγκεκριμένου προϊόντος κατά ένα. Χρησιμοποιεί τη συνάρτηση setCounters για να ενημερώσει τον κατάλογο των μετρητών, διατηρώντας τις προηγούμενες τιμές των μετρητών και προσθέτοντας 1 στην ποσότητα του συγκεκριμένου προϊόντος.Η συνάρτηση handleDecrement λαμβάνει επίσης ως παράμετρο το id του προϊόντος και ελέγχει εάν η ποσότητα του συγκεκριμένου προϊόντος είναι μεγαλύτερη από 0 πριν προβεί σε μείωση. Αν η ποσότητα είναι μεγαλύτερη από 0, μειώνει την ποσότητα κατά ένα. Χρησιμοποιεί τη συνάρτηση setCounters για να ενημερώσει τον κατάλογο των μετρητών, αφαιρώντας 1 από την ποσότητα του προϊόντος. Επιπλέον, αν η ενημερωμένη ποσότητα είναι μηδέν, το προϊόν αφαιρείται εξ ολοκλήρου από τον κατάλογο των μετρητών. Τέλος την μεταβλητή την περνάμε στην σελίδα showPharmacies όπου και θα χρειαστεί.

```
 <thead className="text-xs text-gray-700 uppercase bg-green-50 dark:bg-gray-700 dark:text-gray-400">
      <tr>
        <th scope="col" className="px-6 py-3">
          Product name
        </th>
        <th scope="col" className="px-6 py-3">
          Quantity
        </th>
        <th scope="col" className="px-6 py-3">
          Manufacturer
        </th>
        <th scope="col" className="px-6 py-3">
          Prescription
        </th>
        <th scope="col" className="px-6 py-3">
          Choose Quantity
        </th>
      </tr>
    </thead>
```

Το thead δεν είναι τίποτα παρά μια κεφαλίδα ενός πίνακα, όπου στην πρώτη γραμμή δηλώνουμε τα ονόματα των στηλών.

```
    <tbody>
      {Array.isArray(products) && products.length > 0 ? (
        products.map((product, index) => (
          <tr  key={index}
            className="bg-white border-b dark:bg-gray-800 dark:border-gray-700"   >
```

```
            <th  scope="row"
              className="px-6 py-4 font-medium text-gray-900 whitespace-nowrap dark:text-white"  >
              {product.product.name}   </th>
            <td className="px-6 py-4">
              <span
                className={`inline-block px-4 py-2 rounded-full text-sm ${getBadgeColor(
           product.quantity  )}`}    >
                {getBadgeText(product.quantity)}    </span>  </td>
            <td className="px-6 py-4">{product.product.manufacturer}</td>
            <td className="px-6 py-4">{product.product.prescription}</td>
            <td className="px-6 py-4">
              <div className="flex items-center">
                <button
                  className="rounded-full w-8 h-8 bg-white text-gray-500 hover:text-gray-700 border
          border-gray-300 hover:border-gray-500 flex items-center justify-center cursor-pointer"
                  onClick={() => handleDecrement(product.product._id)}
                >  -  </button> <input>  disabled
                  className="bg-gray-300 w-8 rounded-lg text-black-700 font-bold text-center mx-2"
                  value={counters[product.product._id] || ""}  />
                <button
                  className="rounded-full w-8 h-8 bg-white text-gray-500 hover:text-gray-700 border
          border-gray-300 hover:border-gray-500 flex items-center justify-center cursor-pointer"
                  onClick={() => handleIncrement(product.product._id)}
                >
                  +  </button> </div></td> </tr>  ))) : ( <tr><td colSpan="4">No products available</td>
          </tr>
        )}
</tbody>
```

Το {Array.isArray(products) && products.length > 0 ? ...} ελέγχει αν η μεταβλητή products είναι πίνακας (Array) και έχει μήκος μεγαλύτερο από το μηδέν. Αν αυτή η συνθήκη ισχύει, τότε εκτελείται το μπλοκ που περιλαμβάνει τον κώδικα για τη δημιουργία γραμμών προϊόντων. Η μέθοδος map χρησιμοποιείται για να επεξεργαστεί κάθε στοιχείο του πίνακα products και να δημιουργήσει μια γραμμή για κάθε προϊόν. Επιπλέον, όπως αναφέραμε προηγουμένως κάθε προϊόν, παρέχεται ένα κουτί εικονίδιο με + και - για την αύξηση και μείωση της ποσότητας. Αυτά τα κουμπιά είναι ενσωματωμένα σε ένα div και διαχειρίζονται από τις συναρτήσεις handleIncrement και handleDecrement περνώντας το id του συγκεκριμένου προϊόντος που εξηγήθηκαν παραπάνω. Τέλος, αν η μεταβλητή products δεν είναι πίνακας ή έχει μήκος μηδέν, τότε εμφανίζεται ένα μήνυμα No products available.

ShowPharmacy

Αφού εξηγήσαμε τα παραπάνω components που αποτελούνται στο showPharmacy, ήρθε η στιγμή να προχωρήσουμε στην εξήγησή του. Λόγω του μεγάλου μεγέθους του αρχείου, θα διαχωρίσουμε την εξήγησή του σε μικρότερα κομμάτια.

```
 const sendMessage = async () => {
   try {
     const config = { headers: { Accept: "application/json" } };
     const response = await axios.post(
       `/orders/sendsms`,
       { userId: userData.userId, pharmacyId: id },
       config
     );
```

```
    const userPhone = `+30${response.data.userPhone}`;
    const pharmacyPhone = `+30${response.data.pharmacyPhone}`;
    const username = userData.username;
    sendSMStoUser(userPhone, username);
    sendSMStoPharmacy(pharmacyPhone);
   } catch {}  };
```

Η συνάρτηση sendMessage χρησιμοποιεί τα ονόματα και τα κινητά τηλέφωνα του συνδεδεμένου χρήστη και του διαχειριστή του κοινωνικού φαρμακείου προκειμένου να αποστείλει ένα αντίστοιχο SMS μήνυμα για την καταχώρηση της νέας παραγγελίας. Η διαδικασία αρχίζει με την κλήση του API μέσω του Endpoint /orders/sendsms, όπου περνάμε ως παραμέτρους το id του συνδεδεμένου χρήστη και το ID του φαρμακείου. Αυτό το Endpoint επιστρέφει τα αντίστοιχα τηλέφωνα.

Τα δύο τηλέφωνα που λαμβάνουμε, τα μορφοποιούμε προσθέτοντας το +30 μπροστά, προκειμένου να υποδηλώσουμε ότι αναφέρονται σε ελληνικά νούμερα. Τέλος, χρησιμοποιούμε τις συναρτήσεις SendSMStoUser και SendSMStoPharmacy, περνώντας τις αντίστοιχες παραμέτρους. Αυτές οι δύο συναρτήσεις γίνονται import από το αρχείο routee.js, το οποίο περιλαμβάνει τη λειτουργικότητα για την παραγωγή των SMS. Η δομή του API αναφέρθηκε στο υπό - κεφάλαιο 2.4.6.

```
const createOrder = async () => {
   try {
     const config = { headers: { Accept: "application/json" } };
     let userId_key = "userId";
     const userId = localStorage.getItem(userId_key);
     const products = Object.entries(counters).map(
       ([productId, quantity]) => ({
         productId,
         quantity,
       })
     );
     sendMessage();
     const response = await axios.post( `/orders/${id}/createOrder`,{ userId, products }, config
     );
     setSubmitProduct(true);
     setCounters({});
     toast.success("Successfully orders!");
     navigate("/orders");
   } catch (e) {
     toast.success("Failed order!");
     return "Seems to be an error";
   }
 };
```

Η συνάρτηση createOrder χρησιμοποιείται για τη δημιουργία μιας νέας παραγγελίας ή κράτησης. Αρχικά, χρησιμοποιούμε τη μέθοδο Object.entries(counters) για να μετατρέψουμε το αντικείμενο counters σε έναν πίνακα, όπου κάθε στοιχείο είναι ένα ζεύγος id προϊόντος - ποσότητας. Η μεταβλητή counters προήλθε από το pharmacyTable component, όπως περιγράφηκε παραπάνω.

Στη συνέχεια, χρησιμοποιούμε το Endpoint /orders/${id}/createOrder και περνάμε ως παράμετρο το id του συνδεδεμένου χρήστη και τη μεταβλητή products που περιέχει τα ζευγάρια id προϊόντος - ποσότητας. Η δυναμική μεταβλητή id του endpoint ,αναγνωρίζει το κοινωνικό φαρμακείο προς το οποίο απευθύνεται η κράτηση.Όταν η παραγγελία ολοκληρώνεται με επιτυχία, εμφανίζεται ένα toast με ένα μήνυμα επιτυχίας· σε διαφορετική περίπτωση, εμφανίζεται ένα μήνυμα αποτυχίας. Τέλος, χρησιμοποιώντας τη συνάρτηση navigate, ο χρήστης ανακατευθύνεται στη σελίδα με όνομα orders.

```
import { useParams, useNavigate } from "react-router-dom";

 const { id } = useParams();
```

Χρησιμοποιώντας τη συνάρτηση useParams από τη βιβλιοθήκη react-router-dom, πήραμε δυναμικά το id του κοινωνικού φαρμακείου από το url . Αυτή η συνάρτηση μας επέτρεψε να αντιστοιχίσουμε τη μεταβλητή id με την τιμή που βρίσκεται στο url, προσφέροντάς μας έτσι τη δυνατότητα να χρησιμοποιούμε δυναμικά το id του κοινωνικού φαρμακείου στη συνέχεια.

```
useEffect(() => {
   const fetchPharmacies = async () => {
    try {
      const config = { headers: { Accept: "application/json" } };
      console.log("Sent request");
      const response = await axios.get(`/pharmacies/${id}`, config);
      console.log(response);
      setPharmacyData(response.data);
    } catch (e) {
      navigate("/nonepage");
    }
   };
   fetchPharmacies();
   setSubmitProduct(false);
 }, [id, navigate, setPharmacyData]);

 const orderSubmitHandler = () => {
   setModalStatus(true);
 };
 const closeOrderModal = () => {
   setModalStatus(false);
 };
```

Όπως έχουμε αναφέρει , η useEffect λειτουργεί ως το πρώτο component που εκτελείται κατά το rendering της σελίδας. Κατά την εκτέλεσή της, αρχικοποιεί το πρώτο αίτημα προς τον εξυπηρετητή (endpoint) στο pharmacies/${id}, προκειμένου να ανακτήσει όλες τις σχετικές πληροφορίες για το κοινωνικό φαρμακείο. Αυτές οι πληροφορίες περιλαμβάνουν το όνομα του κοινωνικού φαρμακείου, τη διεύθυνση, τον αριθμό τηλεφώνου και τη λίστα των προϊόντων που προσφέρει.
Παράλληλα, υπάρχει έλεγχος διαδρομής που αντιμετωπίζει τυχόν προσπάθειες πρόσβασης σε μια διαδρομή για ένα κοινωνικό φαρμακείο που δεν υπάρχει. Σε αυτήν την περίπτωση, ο χρήστης ανακατευθύνεται αυτόματα σε μια σελίδα 404.

Η συνάρτηση orderSubmitHandler εκτελείται όταν ο χρήστης πατάει το κουμπί Order. Με το πάτημα του κουμπιού, η μεταβλητή modalStatus ρυθμίζεται σε true, εμφανίζοντας ένα ανάλογο μήνυμα επιβεβαίωσης στον χρήστη, ρωτώντας τον αν επιθυμεί να εκτελέσει την παραγγελία.Η συνάρτηση closeOrderModal εκτελείται για να κλείσει το μήνυμα όταν ο χρήστης πατάει το "cancel" ή το "submit" για την παραγγελία.

```
 <OrderModal
      closeModal={closeOrderModal}
      modalStatus={modalStatus}
      createOrder={createOrder}
    />
```

Τέλος, αφού υλοποιήσαμε τη λειτουργικότητα των συναρτήσεων createOrder και closeOrderModal, τις περνάμε στο orderModal. Επιπλέον, περνάμε και τη μεταβλητή modalStatus, καθώς ανάλογα με την τιμή της (αν είναι true ή false), θα εμφανιστεί το μήνυμα.

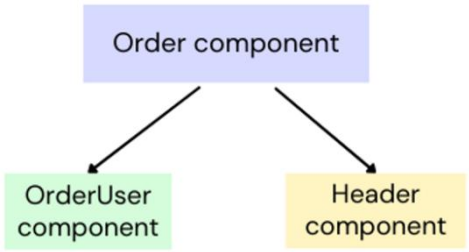
Στο orderModal component συναντάμε των παρακάτω κώδικα.

```
 const handleConfirm = () => {
   createOrder();
   closeModal();
 };

<div>
        <button
            type="button"
            className="inline-flex w-full justify-center rounded-md bg-green-600 px-3 py-2 text-sm
                font-semibold text-white shadow-sm"
            onClick={handleConfirm} / >
            Confirm
          </button>
          <button
            type="button"
              className="mt-3 inline-flex w-full justify-center rounded-md bg-white px-3 py-2 text-
                sm font-semibold text-gray-900 shadow-sm"
            onClick={closeModal}
                ref={cancelButtonRef} >
            Cancel
          </button>
  </div>
```

Η συνάρτηση handleConfirm εκτελείται όταν ο χρήστης πατάει το confirm. Η εκτέλεση της συνάρτησης,περιγράφηκε προηγουμένως. Αν ο χρήστης πατήσει το cancel, το modal απλώς κλείνει χωρίς να πραγματοποιείται κάποια επιπλέον ενέργεια.

3.3.4 Orders

Η σελίδα orders περιέχει τις ενεργές παραγγελίες του χρήστη.



Σχήμα 3.3.5 Συσχέτιση σελίδας Orders με τα άλλα components.

Όπως παρατηρούμε στο Σχήμα 3.3.5, η σελίδα orders αποτελείται από λίγα components που έρχονται σε άμεση επαφή. Το orderUser component αντιστοιχεί ουσιαστικά σε έναν πίνακα, στον οποίο αποστέλλουμε το response που λαμβάνουμε από το αντίστοιχο API. Αυτός ο πίνακας χρησιμεύει για την εμφάνιση και διαχείριση των δεδομένων.Παρακάτω, παραθέτεται ο κώδικας.

```
import React, { useEffect, useState } from "react";
import { Header } from "../components";
import OrderUserTable from "../components/OrderUserComponent/OrderUserTable";
import axios from "axios";
```

Εισάγουμε τις απαραίτητες βιβλιοθήκες και components.

```
 const [allOrders, setAllOrders] = useState({});

 useEffect(() => {
  const fetchPharmacyOrders = async () => {
   try {
    let userId_key = "userId";
    const userId = localStorage.getItem(userId_key);
    const config = { headers: { Accept: "application/json" } };
    const response = await axios.get(
     `/orders/showUserOrders/${userId}`,
     config
    );
    const data = response.data;
    setAllOrders(data);
   } catch (e) {
    return "Seems to be an error";
   }
  };
  fetchPharmacyOrders();
 }, []);
```

Στην useEffect, γίνεται η κλήση προς το API με endpoint το /orders/showUserOrders/${userId} βάζοντας ως παράμετρο το userId και επιστρέφει σαν response όλες τις παραγγελίες / κρατήσεις του συνδεδεμένου χρήστη και τις αποθηκεύει στην allOrders χρησιμοποιώντας την συνάρτηση κατάστασης setAllOrders.

```
 return (
  <div className="m-20">
   <div>
    <Header title="My orders" />
   </div>
   <OrderUserTable allOrders={allOrders} />
   <Toaster
    className="scale-125"
    position="bottom-right"
```

```
      reverseOrder={true}
    />
   </div>
  );
}

export default Orders;
```

Τέλος, περνάμε την μεταβλητή που περιέχει όλες τις παραγγελίες στο component orderUserTable όπου αναλαμβάνει την εμφάνιση των παραγγελιών σε ένα table. Το component έχει παρόμοια λογική με το αντίστοιχο pharmacyTable component όπου εξηγήσαμε στο υπό - κεφάλαιο 3.3.2 .

3.3.5 History User

Η σελίδα "History User" λειτουργεί ως ιστορικό παραγγελιών για τον συνδεδεμένο χρήστη και βρίσκεται στο αρχείο pages. Όπως αναφέρθηκε στο υπο-κεφάλαιο 3.1.2, αυτή η σελίδα περιλαμβάνει μια επιλογή όπου ο χρήστης μπορεί να επιλέξει και να ξαναπαραγγείλει/κάνει κράτηση ενός προϊόντος που έχει αγοράσει στο παρελθόν. Το αρχείο αυτό είναι αρκετά απλό, διότι περιλαμβάνει μόνο το component header, ενώ κυρίως περιέχει έναν πίνακα που χρησιμεύει για την εμφάνιση των αποτελεσμάτων από το response του αντίστοιχου API.

```
import React, { useEffect, useState } from "react";
import { useStateContext } from "../contexts/ContextProvider";

 const userData = useSelector((state) => state.session.userData) || {};
 const [userHistoryOrders, setUserHistoryOrders] = useState([]);
 const {handleProductClick} =useStateContext();
 const navigate = useNavigate();

 useEffect(() => {
   const fetchData = async () => {
     const data = await getUserHistoryOrders(userData.userId);
     setUserHistoryOrders(data.orders || []);
   };
   if (userData.userId) {
     fetchData();
   }
  }, [userData.userId]);
```

Η κλήση προς το API φαίνεται παραπάνω όπου αποθηκεύει το αποτέλεσμα στην μεταβλητή κατάστασης με την χρήση της συνάρτησης setUserHistoryOrders.

```
  const handleClick=(productName)=>{
    handleProductClick(productName)
    navigate("/")
  }
```

Η συνάρτηση handleClick λαμβάνει το όνομα του προϊόντος που πατήθηκε. Στη συνέχεια, αποθηκεύει αυτό το όνομα στο Redux store χρησιμοποιώντας τη συνάρτηση handleProductClick, όπως έχει αναφερθεί στο υπο-κεφάλαιο 3.3.1. Τέλος, η συνάρτηση ανακατευθύνει τον χρήστη στην αρχική σελίδα, όπου το αντικείμενο που πατήθηκε εμφανίζεται στη γραμμή αναζήτησης.

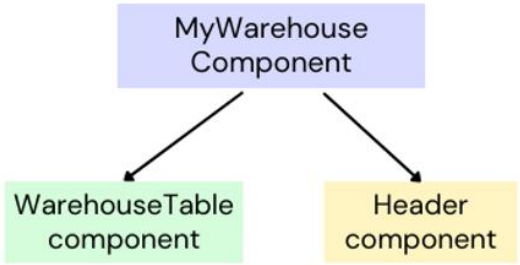
Τέλος, στο παρακάτω component, χρησιμοποιείται η συνάρτηση map για να εμφανίσει όλα τα προϊόντα από τη μεταβλητή userHistoryOrders. Επιπλέον, για τη μετατροπή του format της ώρας, χρησιμοποιείται η συνάρτηση newDate. Τέλος, το όνομα χρησιμοποιείται στο κουμπί Search ως παράμετρος για τη συνάρτηση handleClick.

```
<tbody>

        userHistoryOrders.map((order) => (
         <tr
          key={order._id}       >
          <td >  {order.pharmacy.name}   </td>
                <td ><span ${getBadgeColor(
              order.status )}`} >  {order.status} </span> </td>
          <td>
            {new Date(order.orderDateStart).toLocaleString()}
          </td>
          <td>
            {new Date(order.orderDateFinish).toLocaleString()}
          </td>
          <td >
            <ul>
              {order.products.map((product, index) => (
                <li key={index}>
                  {product.product[0]} x {product.quantity}
                </li>
              ))}
            </ul>
          </td>
          <td >
            <ul>
              {order.products.map((product, index) => (
                <li key={index}>
                  <button
                    onClick={() => handleClick(product.product[0])}
                                    >
                    Search
                  </button>
                </li>
              ))}
            </ul> </td>  </tr>)
</tbody>
```

3.3.6 My warehouse

Στα προηγούμενα υπό - κεφάλαια αναφερθήκαμε σε σελίδες που κυρίως περιηγείται ο συνδεδεμένος χρήστης που κυρίως ενεργεί ως πελάτης. Τώρα θα εξηγήσουμε μια σελίδα που περιηγείται ο διαχειριστής κοινωνικού φαρμακείου. Η σελίδα αυτή ονομάζεται My warehouse και βρίσκεται στον φάκελο pages.

Η My warehouse είναι η σελίδα όπου ο διαχειριστής κοινωνικού καταχωρεί προϊόντα, τα διαχειρίζεται και βλέπει όλο το stock του όπως εξηγήθηκε στο υπό - κεφάλαιο 3.1.3. Όπως βλέπουμε και στο σχήμα 3.3.6 έχει άμεση συσχέτιση με δύο components, το warehouseTable που περιέχει έναν πίνακα για εμφάνιση των προϊόντων και το header που ο τίτλος της σελίδας.



Σχήμα 3.3.6 Συσχέτιση σελίδας MyWarehouse με τα άλλα components

```
const [allProducts, setAllProducts] = useState();
const [productQuantity, setProductQuantity] = useState({});
const [submitProduct, setSubmitProduct] = useState(false); //Handling the refresh
const [productData, setProductData] = useState({

 const fetchWarehouse = async () => {
   try {
     const config = { headers: { Accept: "application/json" } };
     const response = await axios.get(`/warehouses/${id}`, config);
     setAllProducts(response.data);
   } catch (e) {
     return "Seems to be an error";
   }
 };
```

Η συνάρτηση αυτή επιστρέφει όλα τα προϊόντα του κοινωνικού φαρμακείου και τα αποθηκεύει στη μεταβλητή κατάστασης allProducts.

```
 const createProduct = async (newProduct) => {
   try {
     const config = { headers: { Accept: "application/json" } };
     const response = await axios.post(
       `/warehouses/${id}/addProduct`,
       newProduct,
       config
```

```
    );
    if (response.data.message === "Product added successfully.") {
      toast.success("Product added successfully!");
      return response.data;
    }
  } catch (e) {
    console.error("Error while creating the product:", e);
    toast.error("Product already exists!");
    return "Seems to be an error";
  }
};
const deleteProduct = async (productId) => {
  try {
    const config = { headers: { Accept: "application/json" } };
    const response = await axios.delete(
      `/warehouses/${id}/deleteProduct/${productId}`,
      config
    );
    console.log(response.data);
    return response.data;
  } catch (e) {
    return "Seems to be an error";
  }
};
```

Η createProduct είναι μια συνάρτηση κάνει μια κλήση προς το API, όπου δημιουργεί ένα νέο προϊόν στην αποθήκη του κοινωνικού φαρμακείου. Αν δημιουργηθεί με επιτυχία εμφανίζει ένα toast με αντίστοιχο μήνυμα. Σε περίπτωση που υπάρχει ήδη, εμφανίζει ένα αντίστοιχο μήνυμα αποτυχίας. Αντίστροφη λειτουργικότητα έχει η συνάρτηση deleteProduct όπου διαγράφει το προϊόν σύμφωνα με το id του.

```
const addQuantityProduct = () => {
console.log(productQuantity);
editWarehouseProduct(productQuantity, "add");
toast.success("Successfully product updated!");
};
const deleteQuantityProduct = () => {
console.log(productQuantity);
editWarehouseProduct(productQuantity, "delete");
toast.success("Successfully product updated!");
};
```

Οι δύο παραπάνω συναρτήσεις χρησιμοποιούνται για τη διαχείριση προϊόντων στην αποθήκη. Σε κάθε περίπτωση, περνάμε τον αριθμό των προϊόντων και τον τύπο της πράξης που θέλουμε να εκτελεστεί ως παράμετρο στη συνάρτηση editWarehouseProduct. Η editWarehouseProduct αναλαμβάνει την επεξεργασία των προϊόντων στην αποθήκη, ανάλογα με τον τύπο πράξης που της δίνεται.

```
const editWarehouseProduct = async (productQuantity, operation) => {
   try {
     const config = { headers: { Accept: "application/json" } };
     const response = await axios.put(
       `/warehouses/${id}/editWarehouseProduct`,
       { productQuantity, operation },
       config
     );
     console.log(response.data);
     return response.data;
   } catch (e) {
     return "Seems to be an error";
   }
 };
```

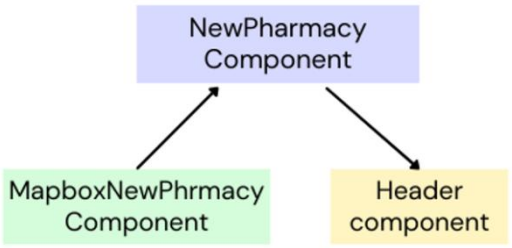
Η συνάρτηση editWarehouseProduct δεν κάνει τίποτα πέρα από μια κλήση στο endpoint /warehouses/${id}/editWarehouseProduct και στο σώμα του τα δύο πεδία που εξηγήσαμε παραπάνω. Έτσι, παραχωρεί την διαχείριση στο backend.

```
 const handleSubmit = async (event) => {
   event.preventDefault();
   const newProduct = {
     name: productData.name,
     quantity: productData.quantity,
     manufacturer: productData.manufacturer,
     activeIngredient:productData.activeIngredient,
     prescription: productData.prescription,
   };
   createProduct(newProduct);
   };
```

Η συνάρτηση handleSubmit δημιουργεί ένα νέο αντικείμενο newProduct, το οποίο περιέχει τα στοιχεία όπως το όνομα του προϊόντος, η ποσότητα, ο κατασκευαστής και άλλα. Στη συνέχεια, καλεί τη συνάρτηση createProduct που έγινε αναφορά παραπάνω για να προσθέσει το νέο προϊόν. Τέλος, μηδενίζει τα στοιχεία της φόρμας που υποβλήθηκαν προηγουμένως.

3.3.7 NewPharmacy

Η σελίδα New Pharmacy είναι για την δημιουργία / καταχώρηση του κοινωνικού φαρμακείου και βρίσκεται στο φάκελο pages/pharmacies. Δεν περιέχει τίποτα πέρα από πεδία καταχώρησης, ένα header και ένα χάρτη που δίνει την δυνατότητα να βάλει το marker σε γεωγραφικό επίπεδό του κοινωνικού φαρμακείου. Προτού δούμε τι περιέχει η σελίδα newPharmacy,ας εξετάσουμε το mapboxNewPharmacy.



Σχήμα 3.3.7 Συσχέτιση σελίδας NewPharmacy με τα άλλα components.

MapboxNewPharmacy

Το MapboxNewPharmacy component περιέχει όλη την λειτουργικότητα που αφορά το κλίκ του χρήστη πάνω στον χάρτη.

```
const [newPlace, setNewPlace] = useState(null);
 const markPoint = (e) => {
   const coordinates = e.lngLat;
   setNewPlace({ lng: coordinates.lng, lat: coordinates.lat });
   props.coordinatesHandler(coordinates)
 };
```

Η συνάρτηση markPoint ανιχνεύει το κλικ του χρήστη πάνω στον χάρτη και αποθηκεύει το γεωγραφικό πλάτος σε μήκος ως αντικείμενο. Με την συνάρτηση props.coordinatesHandler περνάμε την πληροφορία στο γονικό component .

```
    <div className="">
    <ReactMapGL
      onClick={markPoint}
      mapboxAccessToken={
        "mapToken"
      }    style={{ width: 900, height: 400 }}
      mapStyle="mapbox://styles/mapbox/streets-v9"
      transitionDuration="200"
    >
      {!newPlace ? null : (
              <Marker latitude={newPlace.lat} longitude={newPlace.lng}></Marker>
      )}
    </ReactMapGL>
</div>
```

Η markPoint εκτελείτε πάνω στον χάρτη του ReactMapGL. Τέλος αφού εκτελεστεί, εμφανίζει το αποτέλεσμα του κλικ πάνω στον χάρτη με την συντεταγμένες που λάβαμε ως marker με την χρήση του component <Marker latitude={newPlace.lat} longitude={newPlace.lng}></Marker>.

NewPharmacy

Η σελίδα newPharmacy περιέχει το κουμπί για την δημιουργία του νέου κοινωνικού φαρμακείου άλλα και την συνάρτηση που εκτελεί την κλήση προς το ανάλογο API.

```
const handleSubmit = async (event) => {
  event.preventDefault();
  const formData = new FormData();
  formData.append("image", fileData);
  formData.append("name", pharmacyData.name);
  formData.append("phoneNumber", pharmacyData.phoneNumber);
  formData.append("email", pharmacyData.email);
  formData.append("user", localStorage.getItem(userId_key));
  formData.append("streetAddress", pharmacyData.streetAddress);
  formData.append("streetNumber", pharmacyData.streetNumber);
  formData.append("city", pharmacyData.city);
  formData.append("area", pharmacyData.area);
  formData.append("postalCode", pharmacyData.postalCode);
  formData.append("coordinates", JSON.stringify(coordinates));
  console.log(formData);
  const result = await postPharmacy(formData);
  toast.success("Successfully pharmacy created!");
  navigate("/");
   };

const postPharmacy = async (formData) => {
  try {
   const config = { headers: { Accept: "application/json" } };
   const res = await axios.post("/pharmacies/create", formData, config);
   return res.data;
  } catch (e) {
   return "Seems to be an error";
  }
 };
```

Όταν γίνει κλικ το κουμπί create new pharmacy εκτελείτε η συνάρτηση handleSubmit όπου παίρνει όλα τα δεδομένα που μαζεύτηκαν από τα πεδία, τα δημιουργεί σε μία μορφή αντικειμένου και έπειτα περνάει το αντικείμενο αυτό ως παράμετρο στην συνάρτηση postPharmacy.

3.3.8 Order Pharmacy

Η σελίδα order pharmacy όπως εξηγήθηκε στο υπό - κεφάλαιο 3.1.3, ο διαχειριστής του κοινωνικού φαρμακείου πρέπει να έχει μια σελίδα την οποία πρέπει να βλέπει τις ενεργές παραγγελίες του και να έχει την δυνατότητα να τις απορρίπτει ή άπλα να τις ορίζει σαν ολοκληρωμένες όταν τις παραδώσει στον τελικό χρήστη συνάνθρωπό μας. Η λειτουργικότητα είναι παρόμοια με την σελίδα του απλού χρήστη όπως εξετάσαμε στο υπό - κεφάλαιο 3.3.4. Απλά, εδώ η σελίδα περιέχει δύο παραπάνω συναρτήσεις και φυσικά διαφορετικό endpoints.

```
const fetchPharmacyOrders = async () => {
    try {
     let userId_key  ="userId";
     const userId = localStorage.getItem(userId_key);
     const response = await axios.get(`/orders/showPharmacyOrders/${userId}`,userId);
     const data=response.data;
     setAllOrders(data)
    } catch (e) {
      return "Seems to be an error";
    }
  };
```

Η συνάρτηση fetchPharmacyOrders εκτελείτε μέσα στην useEffect και ανακτώνται όλες οι ενεργές παραγγελίες του κοινωνικού φαρμακείου με βάση το id του διαχειριστή.

```
const deleteOrderAndReturnProducts=async(orderId)=>{
    try {
          const response = await axios.get(`/orders/deleteOrderAndReturnProducts/${orderId}`);
           toast.success("Successfully order discard!");
    } catch (e) {
      return "Seems to be an error";
    }
  }
  const doneOrderProducts=async(orderId)=>{
    try {
          const response = await axios.get(`/orders/doneOrderProducts/${orderId}`);
          toast.success("Successfully order!");
    } catch (e) {
      return "Seems to be an error";
    }
  }
```

Οι παραπάνω δυο συναρτήσεις εκτελούνται σε περίπτωση εκτέλεση της παραγγελίας ή απόρριψη σύμφωνα με το id της παραγγελιάς.

3.3.9 Sidebar

Το sidebar είναι στο αριστερό μέρος της εφαρμογής και περιέχει τα περιεχόμενα και τις σελίδες που περιηγείται ο χρήστης της εφαρμογής και το αρχείο βρίσκεται στον φάκελο components με όνομα sidebar.jsx. Το sidebar επιπλέον έχει την λειτουργικότητα ελέγχου όπου εμφανίζει ανάλογες επιλογές όταν αφορούν μη συνδεδεμένο χρήστη, για συνδεδεμένο χρήστη κτλ.

```
const pharmacyData = useSelector((state) => state.session.pharmacyData) || {};
const userData = useSelector((state) => state.session.userData);
```

Για να μπορέσουμε να κάνουμε αυτόν τον έλεγχο, θα χρειαστούμε τις τιμές που αφορούν το id και την τιμή αν ο χρήστης είναι verified όπου τα έχουμε αποθηκευμένα μέσα στο react redux store.

```
{pharmacyData &&
        pharmacyData.pharmacy &&
        pharmacyData.pharmacy.isVerified==false &&
        pharmacyData.pharmacy._id != null && (
         <>
           <p
           className="text-gray-400 m-3 mt-4 uppercase" >
           Pharmacy
         </p> <NavLink
           onClick={handleCloseSideBar}
           className={disableLink  }>
           <AiOutlineShoppingCart />
           <span className="capitalize">My Pharmacy</span>
         </NavLink>
         <NavLink
           onClick={handleCloseSideBar}
           className={disableLink  } >
           <HiDatabase />
           <span className="capitalize">My warehouse</span>
         </NavLink>
         <NavLink
           onClick={handleCloseSideBar}
           className={disableLink }
         > <FiEdit />
           <span className="capitalize">Pharmacy Orders</span>
         </NavLink>
        </>
       )}
```

Στον παραπάνω κώδικα γίνεται έλεγχος για την ύπαρξη δεδομένων που αφορούν το κοινωνικό φαρμακείο και εάν είναι επαληθευμένο (verified). Αν υπάρχουν αυτά τα δεδομένα, τότε εμφανίζονται τα αντίστοιχα navlinks που επιτρέπουν στον διαχειριστή να περιηγηθεί στις διαθέσιμες σελίδες. Σε άλλη περίπτωση, θεωρείτε ότι είναι απλός χρήστης και εμφανίζει τα ανάλογα navlinks.

3.4 Υλοποίηση του backend

Στο προηγούμενο κεφάλαιο αναλύσαμε τις σελίδες, τα components και τις συναρτήσεις που αφορούν για την υλοποίηση της εφαρμογής στο frontend. Ήρθε η ώρα να αναλύσουμε και στο επίπεδο του backend. Για την λειτουργία του server, θα χρειαστεί σε ένα διαφορετικό terminal παράλληλα με το frontend και να φτιάξουμε ένα αρχείο server.js.

Για να μπορεί να εκτελεστεί ως server θα πρέπει να υπάρχει το παρακάτω κομμάτι κώδικας.

```
const app = express()
const port = 8000
app.listen(port, () => {
  console.log(`Example app listening on port ${port}`)
})
```

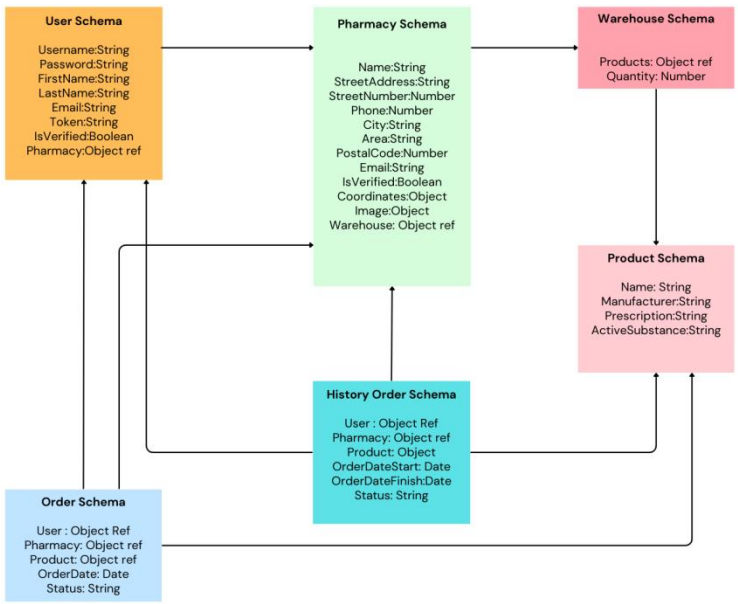
Για να μπορούμε να τρέξουμε τον σέρβερ, θα πρέπει να πληκτρολογήσουμε την εντολή:

```
node server.js
```

3.4.1 Υλοποίηση σχημάτων

Στο υπο-κεφάλαιο 3.2.2 αναλύσαμε τα μοντέλα και τη δημιουργία τους στη βάση δεδομένων, χρησιμοποιώντας τη βιβλιοθήκη Mongoose. Εξηγήσαμε πώς αυτά τα μοντέλα αντιπροσωπεύουν τη λογική των δεδομένων σε μια NoSQL βάση δεδομένων MongoDB. Για να καταστήσουμε πιο κατανοητή την έννοια της αναφοράς μεταξύ δύο σχημάτων, παρουσιάσαμε τη σχέση μεταξύ των χρηστών και των κοινωνικών φαρμακείων και τα πεδία που αποτελεί.

Το παρακάτω σχήμα αναδεικνύει τη βάση δεδομένων της εφαρμογής και περιγράφει τον τρόπο με τον οποίο γίνεται η αλληλεπίδραση μεταξύ τους.



Σχήμα 3.4.1 Βάση δεδομένων της εφαρμογής

Όλα τα σχήματα της βάσης βρίσκονται στον φάκελο models όπως είδαμε στο σχήμα 3.2.9.

Warehouse

```
const mongoose = require('mongoose');
const Pharmacy =require("./pharmacy")
const Schema=mongoose.Schema;
const WarehouseSchema = new Schema({
 pharmacy: {
   type: Schema.Types.ObjectId,
   ref: 'Pharmacy',
 },
 products: [
   {
     product: {
       type: Schema.Types.ObjectId,
       ref: 'Product',
     },
     quantity: Number
   },
 ],
});
module.exports =  mongoose.model('Warehouse', WarehouseSchema);
```

Στο σχήμα της αποθήκης του φαρμακείου, υπάρχουν 2 κύρια μοντέλα που συνδέονται μεταξύ τους, το μοντέλο Pharmacy και το μοντέλο Product. Το μοντέλο Warehouse περιλαμβάνει δύο κύρια πεδία. Πρώτον, το πεδίο pharmacy αναφέρεται στο μοντέλο Pharmacy χρησιμοποιώντας το ObjectID ως αναφορά. Δεύτερον, το πεδίο products είναι ένας πίνακας αντικειμένων, καθένα από τα οποία αναφέρεται στο μοντέλο Product με το ObjectID ως αναφορά, ενώ παράλληλα περιέχει το πεδίο quantity που συμβολίζει την ποσότητα του προϊόντος.

Product

```
const mongoose=require("mongoose");
const Schema= mongoose.Schema;
const ProductSchema = new Schema({
   name: {
     type: String,
     required: true,
   },
   manufacturer: {
     type: String,
     required: true,
   },
   prescription:{
     type: String,
     required: true,
   },
```

```
  activeIngredient:{
    type:String,
    required: true,
  },
});

module.exports=mongoose.model("Product",ProductSchema);
```

Το μοντέλο Product ορίζει τη δομή δεδομένων για ένα προϊόν στη βάση δεδομένων. Αποτελείται από τα εξής πεδία. Ένα πεδίο name που αναπαριστά το όνομα του προϊόντος, το Manufacturer που δηλώνει τον κατασκευαστή του προϊόντος, το prescription που υποδεικνύει εάν το προϊόν απαιτεί συνταγή ή όχι και το activeIngredient που περιέχει το την δραστική ουσία του προϊόντος.

Order

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;
const OrderSchema = new Schema({
 user: {
  type: Schema.Types.ObjectId,
  ref: 'User',
  required: true,
 },
 pharmacy: {
  type: Schema.Types.ObjectId,
  ref: 'Pharmacy',
  required: true,
 },
 products: [
  {
   product: {
    type: Schema.Types.ObjectId,
    ref: 'Product',
    required: true,
   },
   quantity: {
    type: Number,
    required: true,
   },
  },
 ],
 orderDate: {
  type: Date,
  default: Date.now,
 },
 status: {
```

```
    type: String,
    enum: ['pending', 'success', 'canceled'],
    default: 'pending',
  },
});

module.exports = mongoose.model('Order', OrderSchema);
```

Το μοντέλο Order καθορίζει τη δομή μιας ενεργής παραγγελίας στο σύστημα. Κάθε παραγγελία αντιστοιχεί σε έναν συγκεκριμένο χρήστη (user) και κοινωνικό φαρμακείο (pharmacy). Η λεπτομέρεια των προϊόντων που περιλαμβάνονται στην παραγγελία αποθηκεύεται στο πεδίο product, το οποίο είναι ένας πίνακας αντικειμένων. Κάθε αντικείμενο περιλαμβάνει το product, που αναφέρεται στο μοντέλο Product χρησιμοποιώντας το ObjectID ως αναφορά, καθώς και την ποσότητα του συγκεκριμένου προϊόντος στην παραγγελία.

Το πεδίο orderDate καθορίζει την ημερομηνία καταχώρησης της παραγγελίας και έχει ως προεπιλεγμένη τιμή την τρέχουσα ημερομηνία. Τέλος, το πεδίο status καθορίζει την κατάσταση της παραγγελίας και μπορεί να παίρνει μία από τις τρεις τιμές: pending, success ή canceled. Η προεπιλεγμένη κατάσταση είναι pending.

OrderHistory

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;
const OrderHistorySchema = new Schema({
  order: {
    type: Schema.Types.ObjectId,
    ref: 'Order',
    required: true,
  },
  user:{
    type: Schema.Types.ObjectId,
    ref: 'User',
  },
  pharmacy:{
    type: Schema.Types.ObjectId,
    ref:"Pharmacy"
  },
  status: {
    type: String,
    enum: ['pending', 'success', 'canceled'],
    default: 'pending',
  },
  orderDateFinish: {
    type: Date,
    default: Date.now,
  },
  orderDateStart: {
    type: Date,
    default: null,
  },
```

```
 products: [{
 product: {
      type:[Object]
    },
    quantity: {
      type: Number,
      required: true,
    },
  },
 ],
});

module.exports = mongoose.model('OrderHistory', OrderHistorySchema);
```

Το μοντέλο OrderHistory παρακολουθεί το ιστορικό ενός παραγγελίας στο σύστημα. Κάθε εγγραφή στο ιστορικό αντιστοιχεί σε μια συγκεκριμένη παραγγελία order και συνδέεται με τον αντίστοιχο χρήστη και κοινωνικό φαρμακείο μέσω των αντίστοιχων ObjectIDs. Το πεδίο status καθορίζει την κατάσταση ιστορικού της παραγγελίας , με τις πιθανές τιμές να είναι , success ή canceled.

Τα πεδία orderDateFinish και orderDateStart καθορίζουν αντίστοιχα την ημερομηνία ολοκλήρωσης και έναρξης της παραγγελίας. Η ημερομηνία ολοκλήρωσης έχει ως προεπιλεγμένη τιμή την τρέχουσα ημερομηνία, ενώ η ημερομηνία έναρξης αρχικά ορίζεται ως null.Το πεδίο products αποθηκεύει τα προϊόντα που περιλαμβάνονται στο ιστορικό παραγγελίας . Κάθε αντικείμενο περιλαμβάνει το product, που είναι τύπου Object, και την αντίστοιχη ποσότητα quantity του προϊόντος.

3.4.2 Αυθεντικοποίηση

Σε πολλά σημεία κατά την υλοποίηση του frontend στο υπό - κεφάλαιο 3.3 έχουμε αναφέρει την πρόταση <<ο συνδεδεμένος χρήστης>> ή αντίστοιχα <<ο διαχειριστής του κοινωνικού φαρμακείου>>. Για να μπορέσουμε να αναφέρουμε τις παραπάνω αυτές προτάσεις, έχουν υλοποιηθεί κάποιες λειτουργίες και έλεγχοι. Στην ουσία, σκοπός της αυθεντικοποίησης είναι να διασφαλιστεί ότι μόνο εξουσιοδοτημένοι χρήστες έχουν πρόσβαση σε συγκεκριμένες σελίδες ή υπηρεσίες.

Στην περίπτωση μας, ο κώδικας της αυθεντικοποιήσης υλοποιήθηκε στον φάκελο controller με όνομα user.js.

```
const User = require("../models/user");
const passport = require("passport");

module.exports.register = async (req, res, next) => {
   const { email, username, password, firstName, lastName, mobile } = req.body;
   const user = new User({ email, username, firstName, lastName, mobile });
   const registeredUser = await User.register(user, password);
};


module.exports.login = (req, res, next) => {
  passport.authenticate("local", (err, user) => {
    if (err) {
      return next(err);
    }
```

```
    if (!user) {
      return res.status(401).json({ message: "Authentication failed" });
    }
    req.logIn(user, (err) => {
      if (err) {

        return next(err);
      }else{
        req.session.userId = user._id;
        req.session.username = user.username;
        req.session.isVerified=user.isVerified;
        return res.json({ message: "Login successful" });
      }
    });
  })(req, res, next);
};

module.exports.userData = async (req, res) => {
  try {
    const id = req.params.id;
    const user = await User.findById(id);
    if (!user) {
      return res.status(404).json({ error: 'User not found' });
    }
    res.json(user);
  } catch (error) {
    res.status(500).json({ error: 'Internal server error' });
  }
};
```

Αρχικά, γίνεται η εισαγωγή απαραίτητων modules και στοιχείων όπως το User από τον φάκελο models και το passport για την υλοποίηση της ταυτοποίησης.

Στο module register, υλοποιεί τη λειτουργία εγγραφής νέου χρήστη. Παίρνει από το σώμα (body) του request τα στοιχεία και τα αποθηκεύει στην βάση . Ίσως η παραπάνω γραμμή const user = new User({ email, username, firstName, lastName, mobile }) δεν είναι πολύ οικία. Όπως αναφέραμε παραπάνω, η mongoose χρησιμοποιείται για αλληλεπίδραση με την βάση και επεξεργασία και δημιουργία ερώτησης των δεδομένων.Η γραμμή αυτή, δημιουργεί ενός νέου αντικειμένου τύπου User βάση ενός σχήματος που έχει οριστεί με τη χρήση του Mongoose.

Στο module login, υλοποιεί τη λειτουργία σύνδεσης χρήστη στην εφαρμογή χρησιμοποιώντας το passport όπως έχουμε αναφέρει στο υπό - κεφάλαιο 2.4.12. Το passport.authenticate("local", (err, user) => {...}) χρησιμοποιεί το passport για την ταυτοποίηση του χρήστη με την στρατηγική local. Η στρατηγική local, σημαίνει ότι ο χρήστης ταυτοποιείται με βάση το όνομα χρήστη και τον κωδικό πρόσβασης του, τα οποία παρέχονται στο σώμα του αιτήματος (req.body). Η callback λαμβάνει δύο παραμέτρους, err για σφάλματα κατά τη διάρκεια της ταυτοποίησης και user για τον ταυτοποιημένο. Αν ο χρήστης βάλει λάθος στοιχεία, επιστρέφεται ένα μήνυμα λάθους. Αν η ταυτοποίηση είναι επιτυχής, με τη χρήση της μεθόδου req.logIn(user, callback) τα δεδομένα του χρήστη όπως το user._id, το user.username και το user.isVerified αποθηκεύονται στη συνεδρία (session).

Στο module του logout, υλοποιεί τη λειτουργία αποσύνδεσης χρήστη στην εφαρμογή χρησιμοποιώντας το Passport.To req.logout((err) => {...}). Χρησιμοποιεί το passport για να εκτελέσει τη λειτουργία αποσύνδεσης (logout). Η μέθοδος req.logout() παρέχεται από το Passport και αποσυνδέει τον χρήστη από την τρέχουσα συνεδρία. Ο κώδικας εκτελεί την αποσύνδεση και παρέχει

μια συνάρτηση ως callback για να χειριστεί τυχόν σφάλματα που μπορεί να προκύψουν κατά την διαδικασία αποσύνδεσης και επιστρέφει ένα μήνυμα σφάλματος. Εάν η αποσύνδεση είναι επιτυχής και δεν υπάρχει σφάλμα, επιστρέφεται ένα μήνυμα επιτυχίας στον χρήστη.

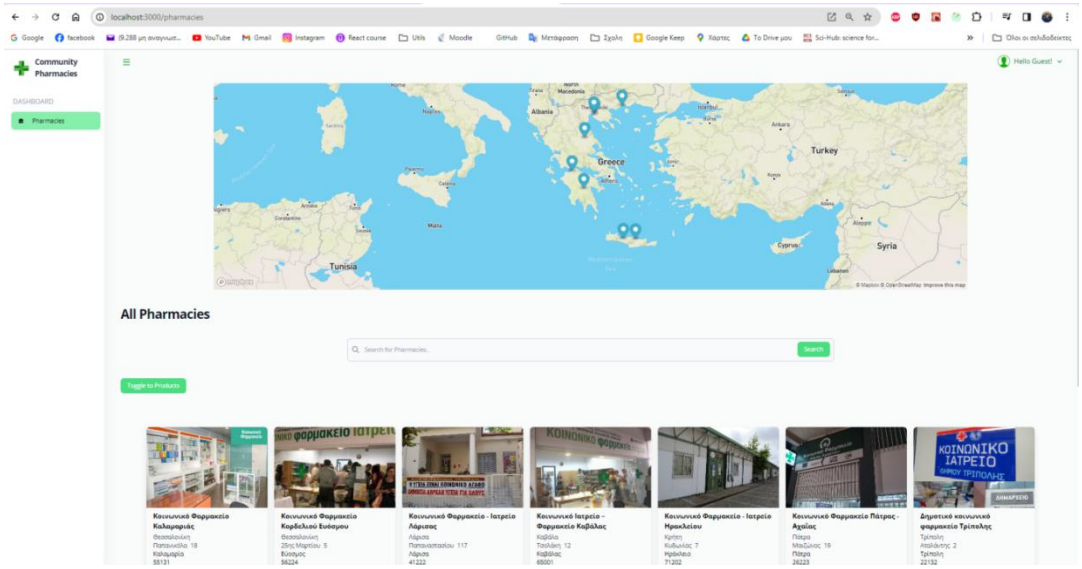
Στο module userData, υλοποιεί τη λειτουργία για την ανάκτηση δεδομένων χρήστη βάσει του ID. Σε πολλές περιπτώσεις εμφανίζουμε τα στοιχεία του χρήστη ή τα χρησιμοποιήσουμε όταν θέλουμε να κάνουμε μια ενέργεια όπως μια παραγγελία.

Κεφάλαιο 4ο: Η εφαρμογή Community pharmacies

Σε αυτό το κεφάλαιο θα προσομοιώσουμε τα τρία σενάρια χρήσης της εφαρμογής. Το πρώτο σενάριο θα είναι ένα απο την πλευρά του πελάτη. Θα δούμε όλα τα βήματα που πρέπει να κάνει ο πελάτης στην εφαρμογή όπως δημιουργία λογαριασμού, αυθεντικοποίηση λογαριασμού, είσοδο στην λογαριασμό και τέλος την δημιουργία παραγγελία/ καταχώρηση προιόντων. Το δεύτερο σενάριο θα είναι τα βήματα που πρέπει να κάνει ο διαχειρηστής κοινωνικού όπως καταχώρηση κοινωνικού φαρμακείου, καταχώρηση προιόντων και τέλος απόρυψη παραγγελίων ή να δηλώσει ως επιτυχιμένη παραγγελία. Τέλος, ένα σενάριο για ανάκτηση κωδικού πρόσβασης.

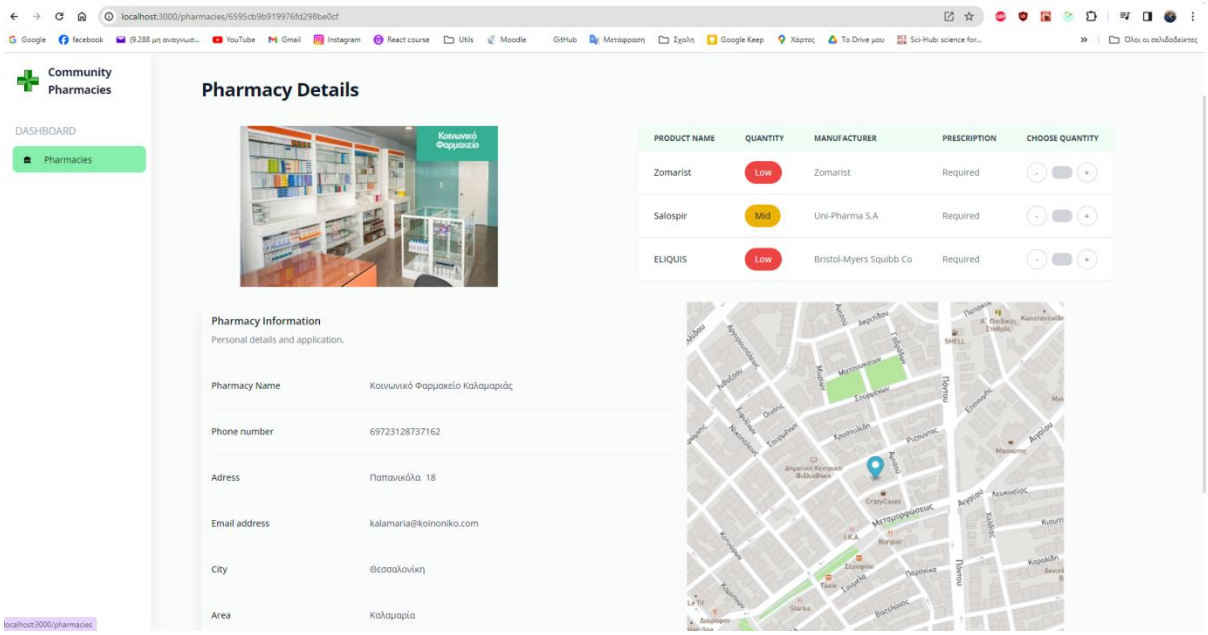
4.1 Σενάριο χρήστη

Ο χρήστης χρησιμοποιεί ένα λίνκ για να αναζητήσει το site στον browser, στην συγκεκριμένη περίπτωση τρέχει στον τοπικό σέρβερ χρησιμοποιόντας το link http://localhost:3000/pharmacies . Η πρώτη σελίδα εμφανίζεται, φαίνεται στο σχήμα 4.1.1.



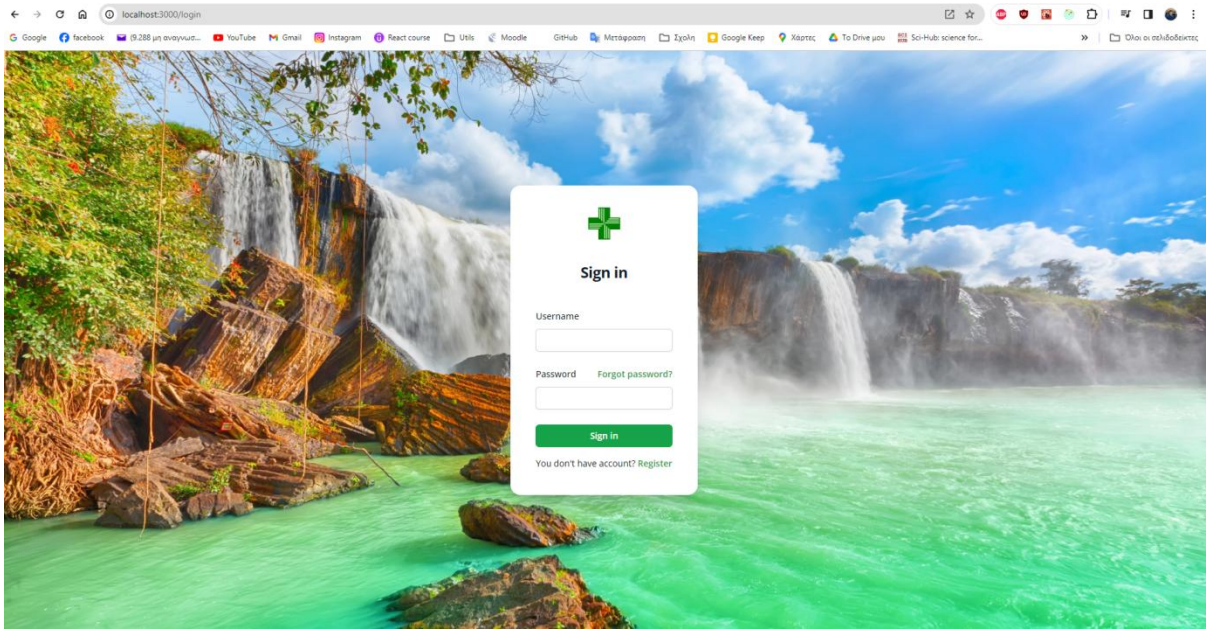
Σχήμα 4.1.1: Pharmacies page.

Σε αυτήν τη σελίδα, μπορεί ο μη συνδεδεμένος χρήστης να δει τα διαθέσιμα κοινωνικά φαρμακεία και την τοποθεσία τους στον χάρτη. Επιπλέον, έχει τη δυνατότητα να επιλέξετε ένα από τα κοινωνικά φαρμακεία και να δει περισσότερες λεπτομέρειες , όπως φαίνεται στο σχήμα 4.1.2. Επίσης, μπορεί να πραγματοποιήσει αναζήτηση για ένα κοινωνικό φαρμακείο με βάση το όνομά του, την πόλη όπου βρίσκεται ή να κάνετε αναζήτηση με το όνομα του φαρμάκου.

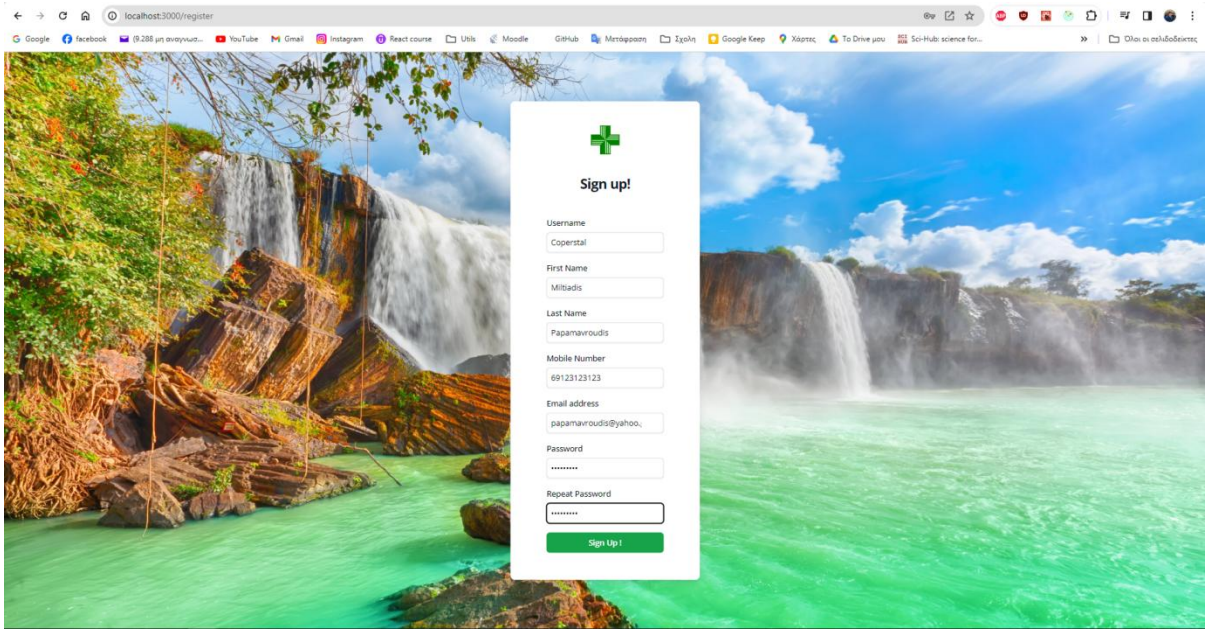


Σχήμα 4.1.2: Pharmacy details page.

Ο χρήστης στην προκείμενη περίπτωση δεν είναι συνδεδεμένος και δεν έχει λογαριασμό, όποτε θα πρέπει να κατευθυνθεί επάνω δεξιά ώστε να πατήσει το login και να επιλέξει την επιλογή register. Όπως φαίνεται παρακάτω στον σχήμα 4.1.3 και 4.1.4.

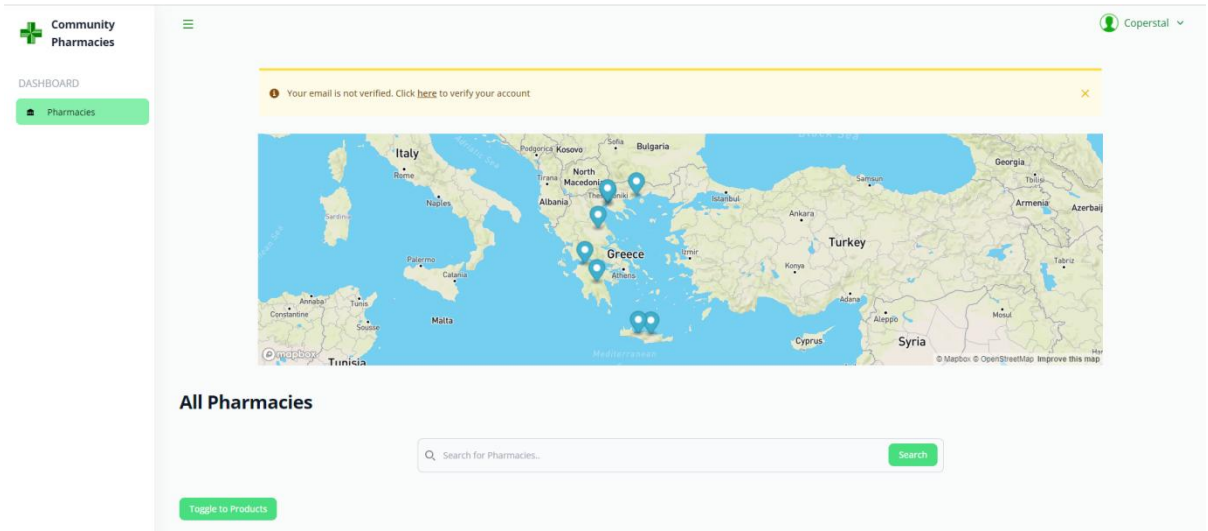


Σχήμα 4.1.3: Login page.



Σχήμα 4.1.4: Register page.

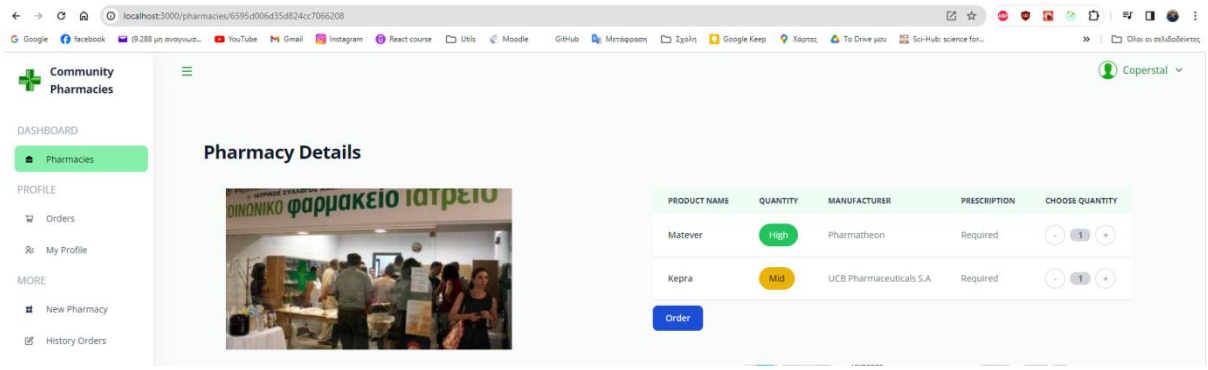
Όταν ο χρήστης συμπληρώσει όλα τα πεδία, θα ανακατευθυνθεί στη σελίδα σύνδεσης προκειμένου να συνδεθεί με τον νέο του λογαριασμό. Στη συνέχεια, εφόσον πραγματοποιήσει τη σύνδεση, θα παρατηρήσει ότι το email του δεν έχει επαληθευτεί. Για να επιβεβαιώσει το email του, θα χρειαστεί να πατήσει το here, ώστε να λάβει ένα email στον φάκελο εισερχομένων του. Ο σύνδεσμος στο email θα τον οδηγήσει πίσω στην εφαρμογή, κατόπιν του οποίου θα έχει πλήρη δικαιώματα πρόσβασης.



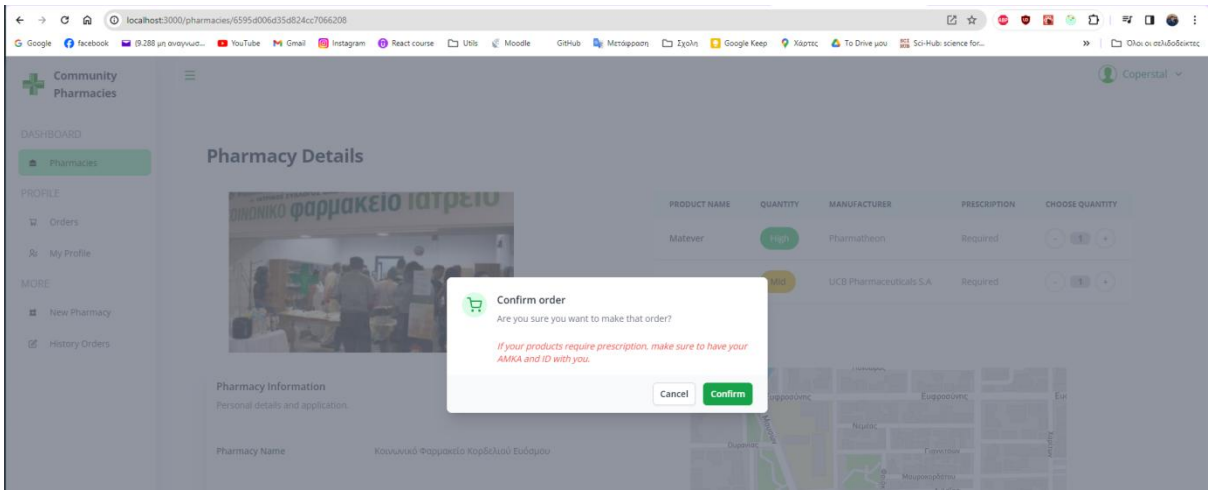
Σχήμα 4.1.5: Μήνυμα επαλήθευσης λογαριασμού χρήστη.

Σχήμα 4.1.6: Email επαλήθευσης .

Αφού ο χρήστης πλέον έχει τα ανάλογα δικαιώματα, μπορεί να επιλέξει ένα κοινωνικό φαρμακείο και να επιλέξει κάποια προϊόντα και να πατήσει το κουμπί order και έπειτα το μήνυμα επιβεβαίωσης όπως φαίνεται στο σχήμα 4.1.7 και 4.1.8.

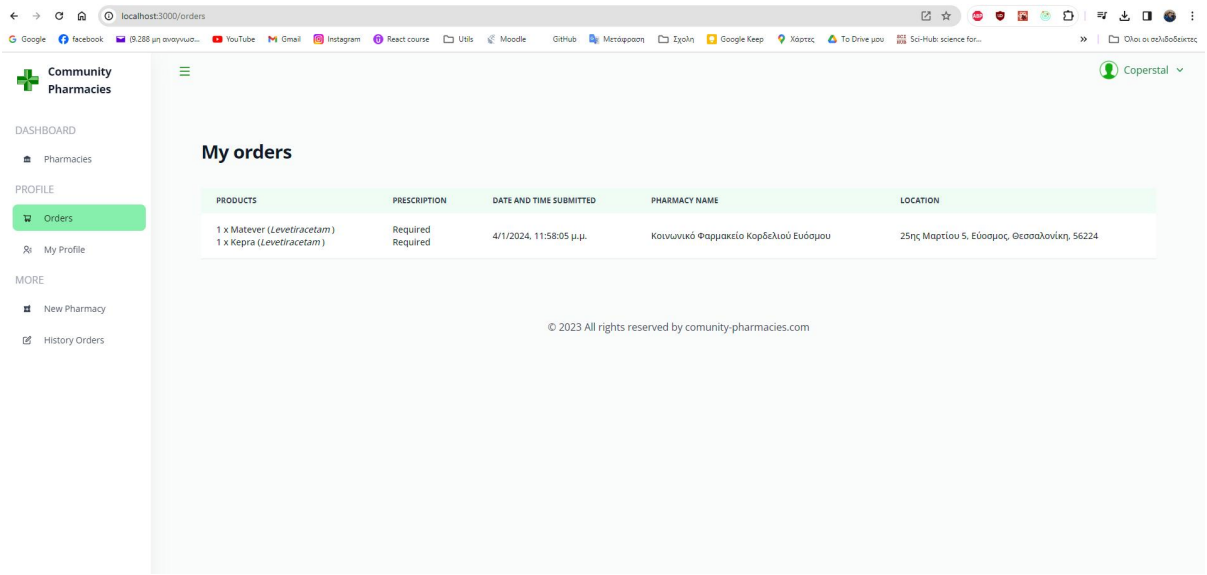


Σχήμα 4.1.7: Επιλογή προϊόντος .



Σχήμα 4.1.8: Επιβεβαίωση παραγγελίας.

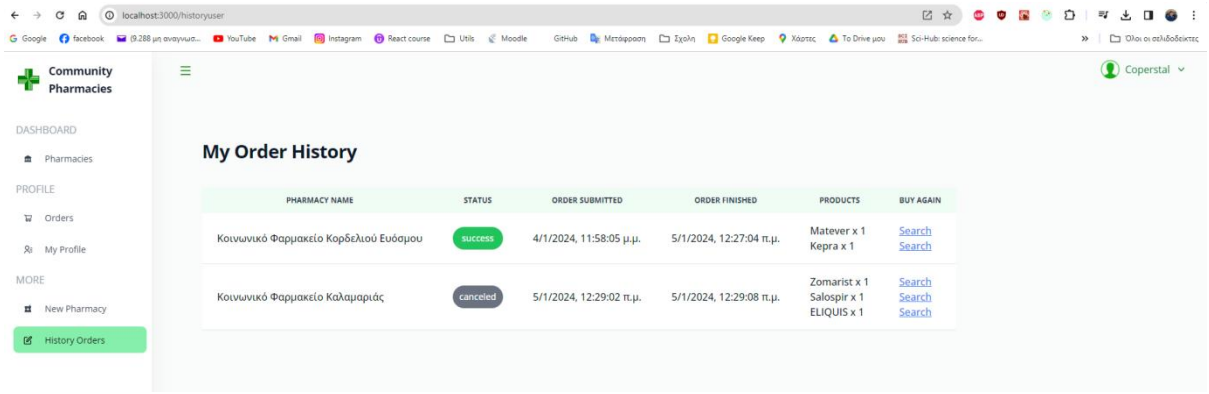
Τέλος ο χρήστης, θα ανακατευθυνθεί στην σελίδα my orders όπου βλέπει την ενεργή του κράτηση / παραγγελία όπως φαίνεται στο σχήμα 4.1.9 ενώ παράλληλα θα λάβει και ένα SMS ότι έγινε με επιτυχία όπως φαίνεται στο σχήμα 4.1.10.



Σχήμα 4.1.9: My orders page.

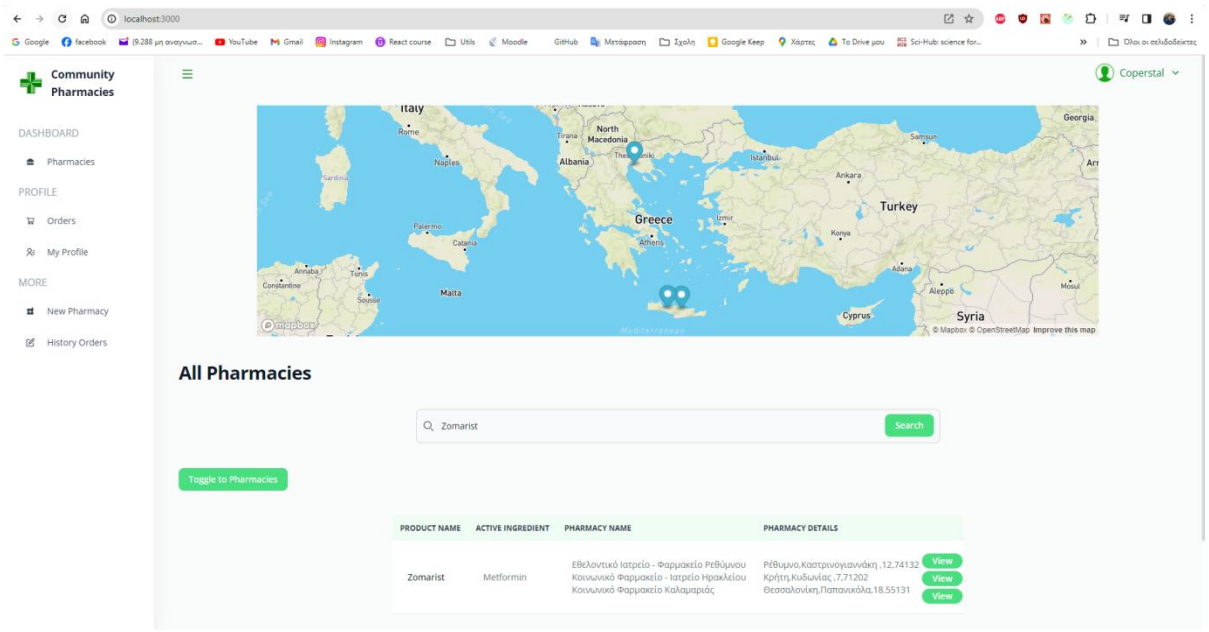
Σχήμα 4.1.10: SMS επιτυχίας καταχώρησης παραγγελίας χρήστη.

Ο χρήστης τώρα έχει τη δυνατότητα να παραλάβει την παραγγελία του από το κοινωνικό φαρμακείο, εκτός εάν απορριφθεί από τον διαχειριστή. Μόλις παραλάβει την παραγγελία του από το κοινωνικό φαρμακείο, η ενεργή παραγγελία του θα μεταφερθεί στη σελίδα History orders με το αντίστοιχο status όπως φαίνεται στο σχήμα 4.1.11.



Σχήμα 4.1.11: My order history page.

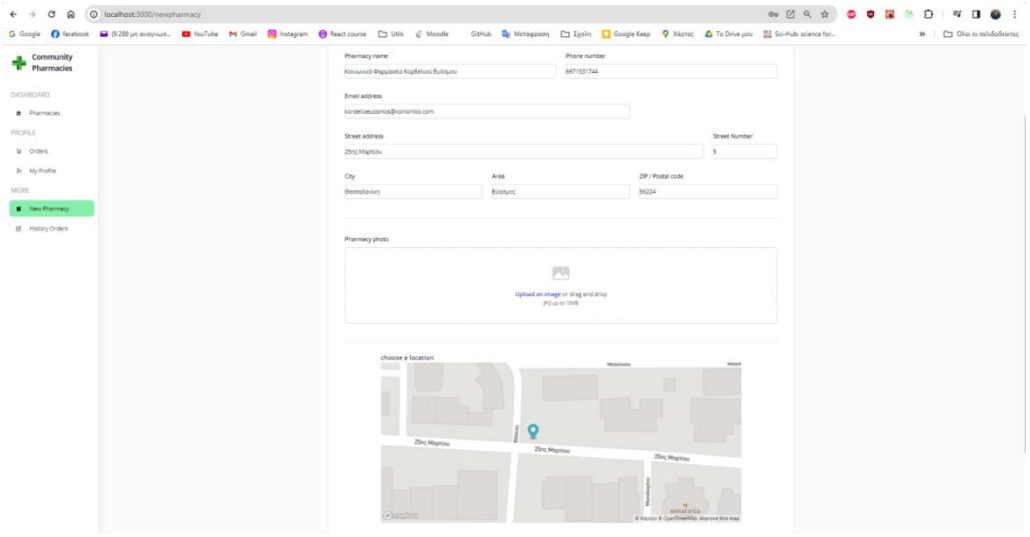
Ο χρήστης αν επιθυμεί μπορεί να πατήσει στον πίνακα στο πεδίο search , ώστε να τον ανακατευθύνει και να εμφανίσει τα ανάλογα κοινωνικά φαρμακεία που έχουν αυτό το προϊόν όπως φαίνεται στο σχήμα 4.1.12 για να κάνει ξανά παραγγελία του προϊόντος.



Σχήμα 4.1.12: Εμφάνιση προϊόντος με βάση το ιστορικό παραγγελίας του.

4.2 Σενάριο διαχειριστή κοινωνικού φαρμακείου

Ο χρήστης που επιθυμεί να καταχωρήσει το κοινωνικό φαρμακείο του στην εφαρμογή θα πρέπει να ακολουθήσει τα ίδια βήματα με έναν κλασσικό χρήστη της εφαρμογής. Αυτό σημαίνει ότι πρέπει να χρησιμοποιήσει τον σύνδεσμο http://localhost:3000/pharmacies, να δημιουργήσει έναν νέο λογαριασμό, να συνδεθεί στην εφαρμογή, και στη συνέχεια να επιβεβαιώσει τον λογαριασμό του μέσω του απεσταλμένου email, χρησιμοποιώντας τον κατάλληλο σύνδεσμο. Έπειτα, θα πρέπει να επιλέξει από το αριστερό sidebar την σελίδα new pharmacy Σε αυτή την σελίδα όπως φαίνεται στο σχήμα 4.2.1 , του εμφανίζεται μια φόρμα στην οποία βάζει τα στοιχεία του κοινωνικού φαρμακείου όπως όνομα, τηλέφωνο, ένα marker την τοποθεσία του κοινωνικού φαρμακείου κτλ.

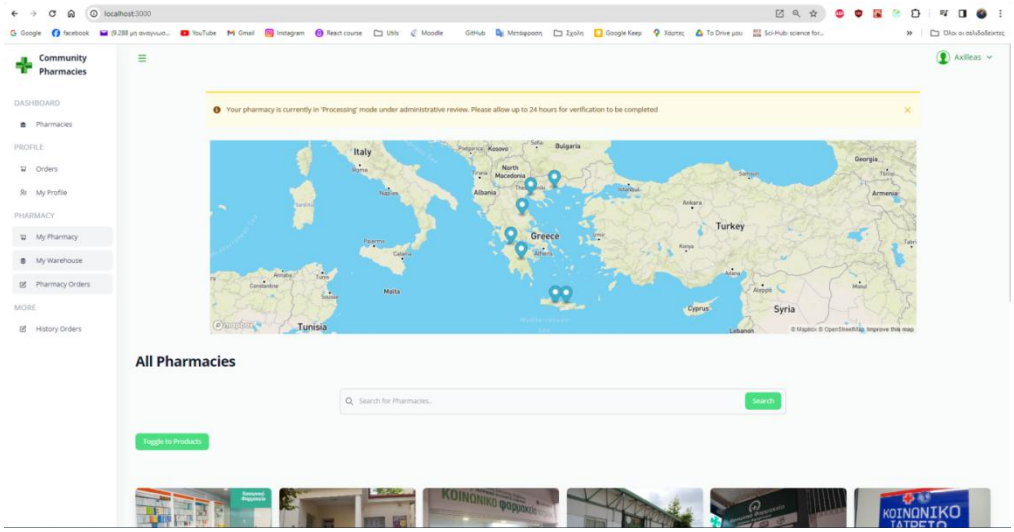


Σχήμα 4.2.1: New pharmacy page.

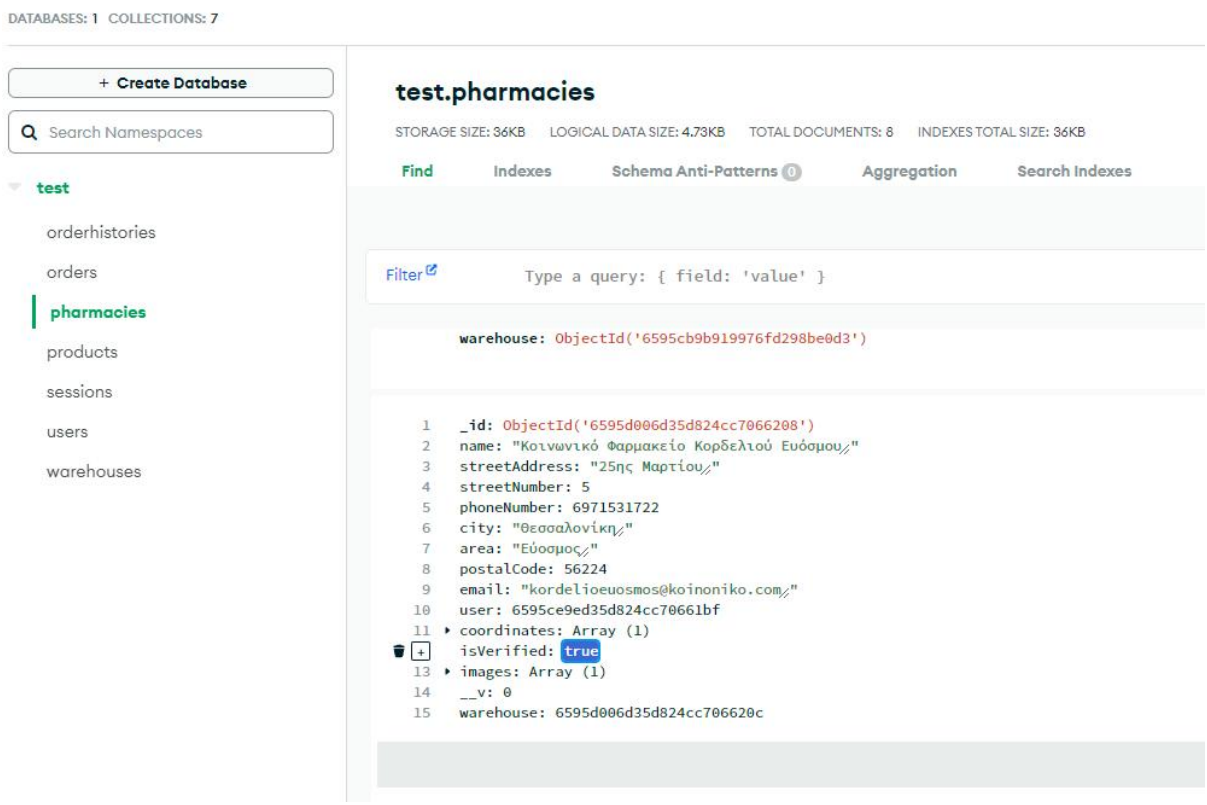
Αφού καταχωρηθεί το κοινωνικό φαρμακείο, όπως φαίνεται στο σχήμα 4.2.3, θα πρέπει να εγκριθεί από τον διαχειριστή της εφαρμογής. Η διαδικασία αυτή πραγματοποιείται με σκοπό την αποφυγή οποιασδήποτε κακόβουλης ενέργειας ή spam. Επιπλέον, όπως φαίνεται στο ίδιο σχήμα, υπάρχουν τρεις νέες σελίδες στο αριστερό sidebar: My Pharmacy, My Warehouse, και Pharmacy Orders, οι οποίες παραμένουν μη προσβάσιμες μέχρι να λάβουν την απαραίτητη έγκριση από τον διαχειριστή.

Στην περίπτωση του διαχειριστή της εφαρμογής, απαιτείται απλά να συνδεθεί στη βάση δεδομένων, να ελέγξει το νέο κοινωνικό φαρμακείο και, εάν τα στοιχεία καταχώρησης φαίνονται έγκυρα και αληθή, να τα εγκρίνει. Στη συνέχεια, θα τροποποιήσει τη μεταβλητή από isVerified:false σε true, όπως φαίνεται στο σχήμα 4.2.4.

Μετά την κατάλληλη έγκριση, ο χρήστης της εφαρμογής θα αποκτήσει τον ρόλο του διαχειριστή κοινωνικού φαρμακείου. Αυτός ο χρήστης θα έχει πλέον πρόσβαση στις προαναφερθείσες σελίδες και, φυσικά, το κοινωνικό φαρμακείο θα εμφανίζεται στο χάρτη και στο σύστημα για όλους τους χρήστες που επιθυμούν να πραγματοποιήσουν κρατήσεις ή παραγγελίες σε αυτό.

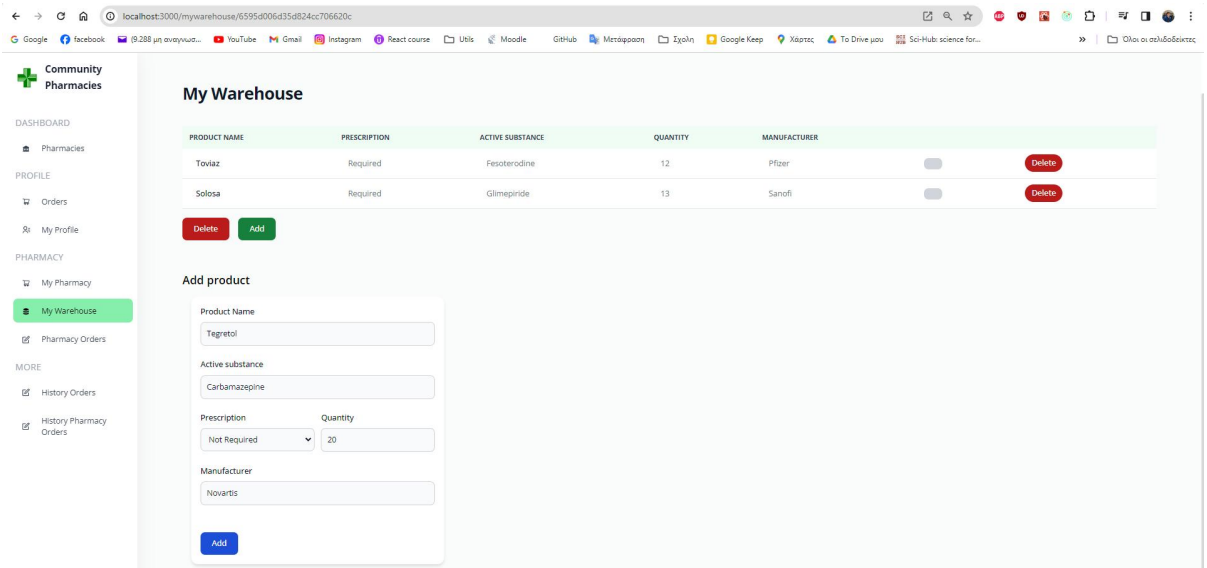


Σχήμα 4.2.3: Μήνυμα επαλήθευσης κοινωνικού φαρμακείου.



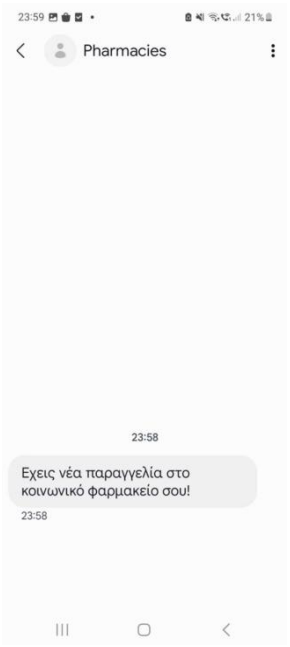
Σχήμα 4.2.4: Διαδικασία επαλήθευσης κοινωνικού φαρμακείου από τον διαχειριστή της εφαρμογής.

Έπειτα, ο διαχειριστής του κοινωνικού φαρμακείου θα πρέπει να καταχωρήσει τα προϊόντα που κατέχει στο σύστημα. Για να το επιτύχει αυτό, θα πρέπει να πλοηγηθεί στη σελίδα My Warehouse, όπως αναφέρεται στο σχήμα 4.2.5. Στη σελίδα My Warehouse, θα βρει διαθέσιμες επιλογές για την καταχώρηση νέων προϊόντων, διαγραφή ή προσθήκη ή μείωση της ποσότητας των υπαρχόντων καταχωρημένων προϊόντων.



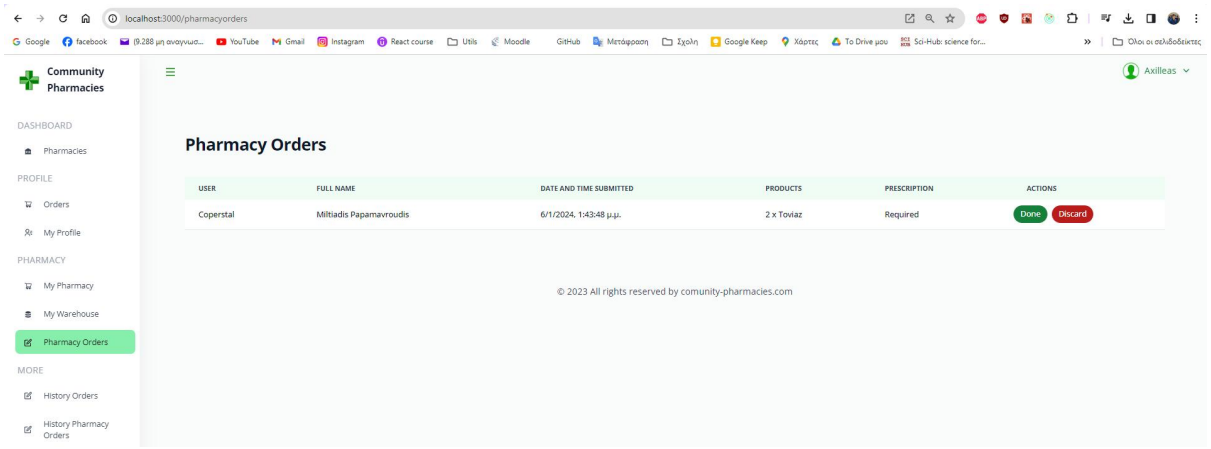
Σχήμα 4.2.5:My warehouse page.

Ο διαχειριστής του κοινωνικού φαρμακείου αφού έχει καταχωρήσει όλα τα προϊόντα του, πλέον μπορεί να περιμένει να δεχτεί κάποια παραγγελία. Όταν δεχτεί μια νέα παραγγελία θα του έρθει ένα αυτοματοποιημένο SMS όπως φαίνεται στο σχήμα 4.2.6 έτσι ώστε να τον ειδοποιήσει για να συνδεθεί στο σύστημα και να κάνει την ανάλογη ενέργεια.



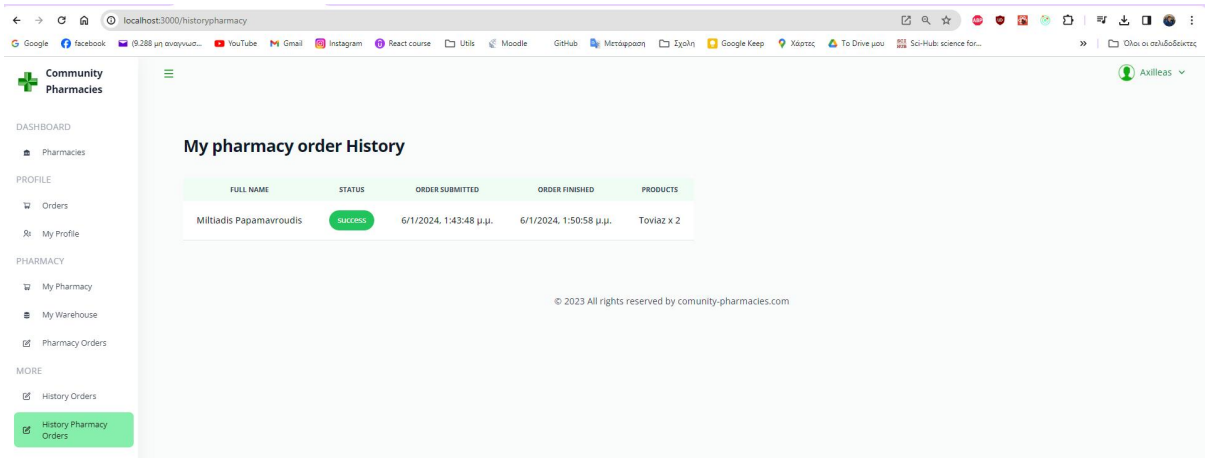
Σχήμα 4.2.6: SMS ενημέρωσης νέας παραγγελίας.

Αμέσως μετά, ο διαχειριστής του κοινωνικού φαρμακείου, αν επιθυμεί, μπορεί να πλοηγηθεί στη σελίδα Pharmacy Orders ώστε να δει τη νέα παραγγελία. Αν επιθυμεί, μπορεί να την αφήσει ως ανοιχτή, επιτρέποντας στον χρήστη να την παραλάβει, και έπειτα να την επισημάνει ως ολοκληρωμένη με το κουμπί Done, ώστε να ολοκληρωθεί η παραγγελία. Σε διαφορετική περίπτωση, μπορεί να την απορρίψει πατώντας το κουμπί Discard.



Σχήμα 4.2.7: Pharmacy order page.

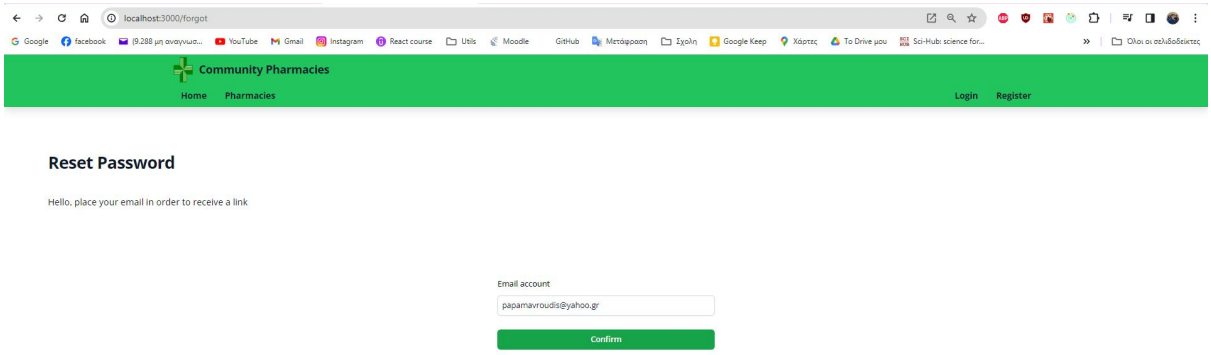
Τέλος, ο διαχειριστής μπορεί να πλοηγηθεί σε όλες τις κλειστές παραγγελίες που έχουν γίνει (απορρίψιμες και ολοκληρωμένες) στη σελίδα History Pharmacy Orders, όπως φαίνεται στο σχήμα 4.2.8.



Σχήμα 4.2.8: History pharmacy order page.

4.3 Ανάκτηση κωδικού πρόσβασης

Σε περίπτωση που κάποιος χρήστης της εφαρμογής έχει ξεχάσει τον κωδικό πρόσβασης του, στη σελίδα login που παρουσιάζεται στο σχήμα 4.1.3, θα πρέπει να επιλέξει την επιλογή Forgot Password και να εισάγει το email με το οποίο είχε δημιουργήσει τον λογαριασμό του. Στη συνέχεια, θα πρέπει να πατήσει το κουμπί Confirm, όπως παρουσιάζεται στο σχήμα 4.3.1.

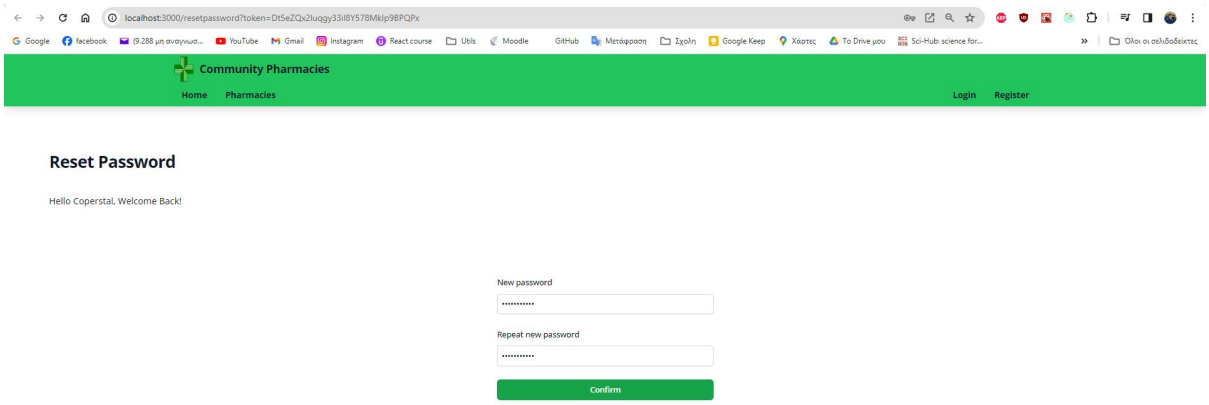


Σχήμα 4.3.1: Ανάκτηση νέου κωδικού πρόσβασης.

Έτσι, στη συνέχεια, θα λάβει ένα email που περιέχει έναν σύνδεσμο, όπως απεικονίζεται στο σχήμα 4.3.2. Ο χρήστης θα πρέπει να κάνει κλικ σε αυτόν τον σύνδεσμο.

Σχήμα 4.3.2: Σύνδεσμος για δημιουργία νέου κωδικού πρόσβασης.

Με τον παραπάνω σύνδεσμο, τον ανακατευθύνει πίσω στην σελίδα όπου πρέπει να εισάγει τον νέο κωδικό πρόσβασης δύο φορές και να πατήσει confirm όπως φαίνεται στο σχήμα 4.3.3.



Σχήμα 4.3.3: Καταχώρηση νέου κωδικού πρόσβασης.

Κεφάλαιο 5ο: Συμπεράσματα και Μελλοντικές επεκτάσεις

Στο πλαίσιο της παρούσας πτυχιακής εργασίας, έχουμε αναδείξει τα οικονομικά προβλήματα που αντιμετωπίζουν πολλοί συμπολίτες μας, είτε είναι ανασφάλιστοι είτε ανήκουν σε ευάλωτες ομάδες, με αποτέλεσμα να αδυνατούν να προμηθευτούν τα απαραίτητα φάρμακά τους από το φαρμακείο. Για την αντιμετώπιση αυτού του προβλήματος, βρήκαμε μια λύση, η οποία περιλαμβάνει την ομαδοποίηση των κοινωνικών φαρμακείων σε μια κεντροκεντροποιημένη εφαρμογή. Στο πλαίσιο αυτό, ο απλός χρήστης του διαδικτύου δεν χρειάζεται πλέον να πραγματοποιεί διάφορες αναζητήσεις στο διαδίκτυο, ελπίζοντας να εντοπίσει ένα κοινωνικό φαρμακείο. Τέλος, επισημαίνουμε ότι, παρά τις διάφορες ιστοσελίδες που μπορούν να εξυπηρετήσουν έναν συμπολίτη μας, ορισμένες φορές δεν παρέχουν τις απαραίτητες πληροφορίες σχετικά με το κοινωνικό φαρμακείο, τα διαθέσιμα φάρμακα ή προϊόντα, καθώς και τη δυνατότητα κράτησης. Η προτεινόμενη εφαρμογή αποτελεί έναν πιο ολοκληρωμένο και αποτελεσματικό τρόπο επίλυσης αυτού του ζητήματος, εξασφαλίζοντας ολοκληρωμένες πληροφορίες και υπηρεσίες για τους χρήστες. Επιπλέον, αναφέραμε και αναλύσαμε τις κύριες τεχνολογίες που χρησιμοποιήθηκαν για την ανάπτυξη της εφαρμογής, γνωστές ως MERN. Επίσης, παρουσιάσαμε ορισμένες βιβλιοθήκες που επιλέχθηκαν για τον σχεδιασμό και τη διαμόρφωση της εφαρμογής, όπως η Tailwind CSS για τη μορφοποίηση, η React Router DOM για τη δημιουργία των διαδρομών, καθώς και η Passport.js για την αυθεντικοποίηση του χρήστη. Επιπλέον, στην υλοποίηση της εφαρμογής συμμετείχαν και άλλες βιβλιοθήκες που, ενώ δεν διαδραμάτισαν κρίσιμο ρόλο, έπαιξαν σημαντικό μέρος στην συνολική λειτουργία του συστήματος. Ανάμεσά τους περιλαμβάνονται η χρήση του Mapbox για τη χαρτογράφηση, το Axios για τις HTTP αιτήσεις, η Cloudinary για τη διαχείριση των εικόνων και το Nodemailer για την υλοποίηση της λειτουργίας αποστολής email.

Στο τρίτο κεφάλαιο, αναφέραμε την αρχιτεκτονική της εφαρμογής, καλύπτοντας τόσο το frontend όσο και το backend. Παρουσιάσαμε τα σημαντικότερα χαρακτηριστικά και τις δομές που χρησιμοποιήθηκαν στην υλοποίηση της. Στη συνέχεια, παρουσιάσαμε μερικά user stories για τους χρήστες της εφαρμογής, δίνοντας ένα ενδεικτικό επισκόπηση των λειτουργιών που παρέχει η εφαρμογή σε διάφορα σενάρια. Στο τέλος του τρίτου κεφαλαίου, προχωρήσαμε στην υλοποίηση της εφαρμογής, αναλύοντας τη λογική πίσω από την κατασκευή της και πώς τα διάφορα κομμάτια της εφαρμογής συνεργάζονται μεταξύ τους. Στο τέταρτο κεφάλαιο, παρουσιάσαμε μερικά σενάρια χρήστη της εφαρμογής, χρησιμοποιώντας διάφορα στιγμιότυπα από την εφαρμογή.

Όπως έχουμε επισημάνει, αναπτύξαμε μια πλήρη εφαρμογή web, η οποία διαθέτει υποστηριξιμότητα και ανταποκριστικότητα (responsiveness) στο περιβάλλον του web. Φυσικά, η εφαρμογή μας είναι σχεδιασμένη να λειτουργεί και σε κινητές συσκευές. Ωστόσο, ως μελλοντική επέκταση, θα ήταν εξαιρετικό να αναπτύξουμε μια εφαρμογή ειδικά σχεδιασμένη για κινητές συσκευές, υποστηρίζοντας τόσο το περιβάλλον Android όσο και το iOS. Αυτή η επέκταση θα ενισχύσει την προσβασιμότητα και την ευχρηστία της εφαρμογής μας, προσφέροντας πλήρη κάλυψη σε όλους τους χρήστες.

ΒΙΒΛΙΟΓΡΑΦΊΑ

[1] Ελληνική δημοκρατία, Ελληνική στατιστική αρχή,"Έρευνα Εισοδήματος και Συνθηκών Διαβίωσης των Νοικοκυριών: Έτος 2022(Περίοδος αναφοράς εισοδήματος: Έτος 2021)",Πειραιάς, 8 Μαΐου 2023,διαθέσιμο

[2] Ελληνική Δημοκρατία, υπουργείο υγείας, πρόσβαση των Ανασφάλιστων στο Δημόσιο Σύστημα Υγείας,24/06/2016,διαθέσιμο:https://www.moh.gov.gr/articles/health/anaptyksh-monadwn-ygeias/3999-prosbash-twn-anasfalistwn-sto-dhmosio-systhma-ygeias.

[3] Givmed,Το πρόβλημα Ποιο πρόβλημα λύνει το GIVMED; Ενημερώσου και ανάλαβε δράση,Έρευνα Εισοδήματος και Συνθηκών Διαβίωσης των Νοικοκυριών 2019 διαθέσιμο: https://givmed.org/el/prosbasi-farmaka-problima/

[4] What is HTML Hypetext Markup Language Basic Explained, Astari S, Aug25 2023, Available: https://www.hostinger.com/tutorials/what-is-html

[5] Introduction to CSS, Faizan Parvez, May 11,2023 Available: https://www.mygreatlearning.com/blog/css-tutorial/

[6] Medium, Ilustration about Mern stack, March 21 2021, available:https://medium.com/techiepedia/what-exactly-a-mern-stack-is-60c304bffbe4

[7] Simplilearn, The Best guide to know what is react,October 5 2023, available: https://www.simplilearn.com/tutorials/reactjs-tutorial/what-is-reactjs

[8] MongoDB, Mern stack explain, available : https://www.mongodb.com/mern-stack

[9] SQL vs NoSQL: The Differences Explained + When to Use Each, Coursera itself, 29th November 2023 Available: https://www.coursera.org/articles/nosql-vs-sql

[10] What is exactly NodeJs? Explained for Beginners, freeCodeCamp, Benjamin Semah,December 5 2022, Available: https://www.freecodecamp.org/news/what-is-node-js/

[11] What is express Js?, Codecademy Team, Available: https://www.codecademy.com/article/what-is-express-js

[12] Introduction to Tailwind CSS, GeekForGeeks, Available: https://www.geeksforgeeks.org/introduction-to-tailwind-css/

[13] Medium,mapbox explained Oct 20,2022, https://blakebhowe.medium.com/introduction-2b278b438adb

[14] React & Redux - An Introduction, Maximilian schwarzmüller Sep 16,2016 https://academind.com/tutorials/react-and-redux-an-introduction

[15] React Router Quick Start Guide, Medium, Blessing Mba Aug 10 2023,Available https://medium.com/@blessingmba3/introduction-to-react-router-da64380ca778

[16] Axios, by Waweru Mwaura, Making HTTP Reguest with Axios, available: https://circleci.com/blog/making-http-requests-with-axios/#:~:text=Axios%20is%20a%20promise%2Dbased,PUT%2FPATCH%20%2C%20and%20DELETE%20.

[17] Routee API, send a single sms, Available : https://docs.routee.net/reference/send-single-sms

[18] Multer,npm, May 30, 2022, Available: https://www.npmjs.com/package/multer

[19] Medium, Using cloudinary as an Aleternative for uploading images to Database, by Nikhil Bhatnagar, Apr 18, 2021, Available: https://medium.com/geekculture/using-cloudinary-as-an-alternative-for-uploading-images-to-database-e786899e9d3e

[20] What is mongoose?, FreeCodeCamp,Monib Bormon, Mar 4 2022 ,Available: https://monib-bormon.medium.com/what-is-mongoose-c1bc3031cc08

[21] Nodemailer Tutorial:Sending an email using nodemailer 101,Available:https://www.turing.com/kb/comprehensive-guide-to-sending-an-email-using-nodemailer

[22] Passport js, by Brad Dayley and Brendan Dayley. Available: https://www.passportjs.org/concepts/authentication/middleware/