ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ – ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Πρόγραμμα Μεταπτυχιακών Σπουδών

«ΠΜΣ Πληροφορική»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Αλληλεπίδραση ασθενών με γιατρούς με χρήση εφαρμογής κινητού** **Interaction between patients and doctors using mobile application**
Ονοματεπώνυμο Φοιτητή	**Παναγιώτης Παγώνης**
Πατρώνυμο	**Αθανάσιος**
Αριθμός Μητρώου	**ΜΠΠΛ19040**
Επιβλέπων	**Ευθύμιος Αλέπης, Αναπληρωτής Καθηγητής**

Ημερομηνία Παράδοσης: **Οκτώβριος 2023**

Τριμελής Εξεταστική Επιτροπή

Ευθύμιος Αλέπης Αναπληρωτής Καθηγητής	Μαρία Βίρβου Καθηγήτρια	Σακκόπουλος Ευάγγελος Αναπληρωτής Καθηγητής

Περιεχόμενα

Ευχαριστίες

Η παρούσα μεταπτυχιακή διατριβή εκπονήθηκε στα πλαίσια του ΠΜΣ με τίτλο «Πληροφορική» στο τμήμα Πληροφορικής του Πανεπιστημίου Πειραιά υπό την επίβλεψη του Καθηγητή κ. Ευθύμιου Αλέπη.

Θα ήθελα να ευχαριστήσω τον Καθηγητή κ. Ευθύμιο Αλέπη για την ουσιαστική και αποτελεσματική επιστημονική βοήθεια που μου προσέφερε κατά τη διάρκεια εκπόνησης της μεταπτυχιακής διατριβής. Εκτός από Καθηγητής, αποτέλεσε και Καθοδηγητής για εμένα και τον ευχαριστώ θερμά για την εμπιστοσύνη που μου έδειξε.

Ακόμη θα ήθελα να ευχαριστήσω ξεχωριστά την οικογένειά μου, καθώς και όσους ανθρώπους στάθηκαν δίπλα μου όλον αυτόν τον καιρό. Τους ευχαριστώ και τους ευγνωμονώ για την ηθική, αλλά και πνευματική υποστήριξη που συνεχίζουν να μου παρέχουν μέχρι και σήμερα.

Περίληψη

Ένα συχνό θέμα που αντιμετωπίζει πολύς κόσμος που βρίσκεται σε ανάγκη ιατρικής περίθαλψης, είναι η αναζήτηση και επικοινωνία με εξειδικευμένους γιατρούς. Ο σκοπός της μεταπτυχιακής διατριβής είναι η ανάπτυξη μίας ιατρικής εφαρμογής για κινητά, για τη διευκόλυνση επικοινωνίας μεταξύ γιατρών και ασθενών. Η εφαρμογή υποστηρίζει τη λειτουργία δημιουργίας λογαριασμού χρήστη και αναζήτησης γιατρού από την πλευρά του ασθενή και δημιουργίας λογαριασμού χρήστη από την πλευρά του γιατρού. Η επικοινωνία μεταξύ των δύο πλευρών επιτυγχάνεται με τη δημιουργία επαφών και εν συνεχεία με την ανταλλαγή μηνυμάτων, προσφέροντας στον ασθενή άμεση πρόσβαση σε ιατρική φροντίδα.

Abstract

A very common problem that people face, when in need of medical help, is the search and communication with specialized doctors. The purpose of current master thesis is to produce a mobile medical app, which will facilitate the communication between doctors and patients. The app supports the functionality of creating an account and search for a doctor from patient's point of view, as well as creating an account from doctor's point of view. The communication between two sides is achieved by creating contacts and sending chat messages, providing to the patient an instant access to medical help.

Ενότητα 1 - Εισαγωγή

Σκοπός της παρούσης μεταπτυχιακής διατριβής, είναι η σχεδίαση και η δημιουργία μίας mobile εφαρμογής (app), η οποία θα επιτρέπει την επικοινωνία μεταξύ ασθενών και γιατρών μέσω μίας οθόνης μηνυμάτων (chat screen). Πιο συγκεκριμένα, οι ασθενείς είναι σε θέση να χρησιμοποιήσουν την εφαρμογή, προκειμένου να αναζητήσουν γιατρούς, χρησιμοποιώντας κάποια φίλτρα αναζήτησης, όπως: ονοματεπώνυμο γιατρού, ειδικότητες, απόσταση από την τρέχουσα τοποθεσία του χρήστη. Αφού ο ασθενής επιλέξει τον γιατρό της αρεσκείας του από την οθόνη των αποτελεσμάτων, στη συνέχεια μπορεί να αποστείλει ένα αίτημα υγείας (heal request) προς το γιατρό. Ο γιατρός από τη μεριά του, μπορεί να αποδεχθεί (ή όχι) το αίτημα του ασθενούς. Εφόσον ο γιατρός αποδεχθεί το αίτημα, στη συνέχεια ανοίγει μία οθόνη μηνυμάτων (chat screen), μέσω της οποίας γιατρός και ασθενής μπορούν να επικοινωνήσουν άμεσα, με απώτερο σκοπό τη λύση του προβλήματος υγειάς του ασθενούς.

Προκειμένου οι χρήστες (είτε ασθενείς είτε γιατροί) να μπορέσουν να χρησιμοποιήσουν την εφαρμογή, απαιτείται δημιουργία λογαριασμού και σύνδεση του χρήστη σε αυτή. Κατά την πρώτη είσοδο του χρήστη στην εφαρμογή, θα χρειαστεί να ενημερώσει και το προφίλ του αντιστοίχως.

Η συγκεκριμένη υλοποίηση θα μπορούσε να βρει εφαρμογή προκειμένου να επιλύσει σύγχρονα θέματα στον τομέα της υγείας. Πιο συγκεκριμένα:

- **στην εύρεση γιατρών για άτομα που επισκέπτονται την χώρα μας (Ελλάδα) από το εξωτερικό:** Άτομα που επισκέπτονται την χώρα μας, δεν είναι εξοικειωμένοι με το σύστημα υγείας. Σαν αποτέλεσμα, η εύρεση γιατρών άμεσα αποτελεί δύσκολη υπόθεση. Η εφαρμογή επιτρέπει στον πιθανό ασθενή να έρθει σε γρήγορη επικοινωνία με το γιατρό και ενδεχομένως να κλείσει ραντεβού μέσω αυτής.
- **στην παροχή έξτρα υπηρεσιών σε συμβόλαια ασφαλιστικών εταιρειών που συνεργάζονται με νοσοκομεία για την κάλυψη επισκέψεων σε γιατρούς:** Συνήθως ένας ασφαλιστικό συμβόλαιο ζωής, δίνει τη δυνατότητα δωρεάν επίσκεψης ασθενών σε γιατρούς. Ωστόσο, οι γιατροί που είναι συμβεβλημένοι με το εκάστοτε νοσοκομείο είναι συγκεκριμένοι, δύσκολο να αναγνωρίσεις ποιοι εκτός εάν ρωτήσεις όλους τους διαθέσιμους γιατρούς που υπάρχουν για την ειδικότητα που ενδιαφέρεσαι (συνήθως δεν υπάρχει online σύστημα καταγραφής), ενώ παράλληλα η επικοινωνία μαζί τους δεν είναι άμεση, καθώς λόγο του συμβολαίου και του μεγάλου πλήθους ατόμων που καλύπτουν, χρειάζεται να περιμένει κανείς μεγάλο χρονικό διάστημα για να επικοινωνήσει ξανά μαζί τους. Η εφαρμογή μπορεί να δίνεται στον ασφαλισμένο ασθενή ως έξτρα παροχή στο συμβόλαιό του, δίνοντας λύση στα παραπάνω.
- **στην άνοδο ζήτησης γιατρών ιδιωτών, οι οποίοι έχουν μικρή ή περιορισμένη εμπειρία:** Πιο συγκεκριμένα γιατροί, οι οποίοι έχουν στην κατοχή τους κάποιο δικό τους ιατρείο, ενδεχομένως να δυσκολεύονται στην αρχή να βρουν ασθενείς να επιλέξουν το ιατρείο τους. Η εφαρμογή θα ενισχύσει τη ζήτηση γιατρών, αφού πλέον οι ασθενείς θα μπορούν να έρχονται σε άμεση επικοινωνία μαζί τους.
- **στην άμεση επικοινωνία μεταξύ ασθενών και γιατρών:** το τελευταίο μπορεί να αποτελέσει θετικό και αρνητικό, κυρίως από τη μεριά των γιατρών. Ωστόσο από την πλευρά των ασθενών, η συγκεκριμένη εφαρμογή θα τους διευκολύνει καθώς είναι σε θέση να ρωτήσουν οτιδήποτε χρειαστούν τον ίδιο τον γιατρό, χωρίς κάποιον ενδιάμεσο (γραμματεία νοσοκομείου/ ιατρείου).

Ενότητα 2 - Ανασκόπηση πεδίου

Στην συνέχεια παραθέτουμε κάποιες εφαρμογές συγγενικές με την υλοποίηση της τρέχουσας μεταπτυχιακής διατριβής, οι οποίες στη βάση τους επιτρέπουν την επικοινωνία μεταξύ γιατρών και ασθενών. Πιο συγκεκριμένα:

2.1 Doctor Anytime

Το DoctorAnyTime αποτελεί μία εφαρμογή, η οποία δίνει τη δυνατότητα σε ασθενείς να επισκεφθούν και να κλείσουν ραντεβού με γιατρούς, καθώς και διαγνωστικά κέντρα για εξετάσεις. Η αναζήτηση μπορεί να γίνει είτε με βάση την περιοχή, είτε με βάση την ειδικότητα του γιατρού, είτε με βάση το όνομα του διαγνωστικού κέντρου. Στη συνέχεια ο ασθενής επιλέγει το γιατρό ή το διαγνωστικό κέντρο και προγραμματίζει ένα ραντεβού με βάση τις ανάγκες του. Για τη λειτουργία της εφαρμογής, απαιτείται δημιουργία λογαριασμού και από τη μεριά των ασθενών, καθώς από τη μεριά των γιατρών. Η κάθε πλευρά οφείλει να ενημερώσει τα απαραίτητα στοιχεία, τα οποία γνωστοποιούνται μέσω της πλατφόρμας όταν πρόκειται για ραντεβού.

2.2 Digital Clinic

Το Digital Clinic αποτελεί μια ψηφιακή υπηρεσία τηλεϊατρικής. Οι κάτοικοι που βρίσκονται σε απομακρυσμένες περιοχές, πολυάσχολοι άνθρωποι ή άτομα με δυσκολία στην μετακίνηση έχουν πρόσβαση σε γιατρό μέσω της κινητής τους συσκευής.

Η εφαρμογή προσφέρει στον ασθενή ιατρική εκτίμηση, διάγνωση και ιατρικές οδηγίες μέσω chat, αλλά και τη δυνατότητα προγραμματισμού με φυσική παρουσία στα Θεραπευτήρια του Υγεία και του Metropolitan Hospital. Όλα αυτά παρέχονται από τον όμιλο HHG (Υγεία, Metropolitan Hospital, Μητέρα, Metropolitan General, Λητώ, Creta, InterClinic).

2.3 Medi ON

Το Medi ON πρόκειται για μία καινοτόμα εφαρμογή της Interamerican, αξιοποιώντας την τεχνητή νοημοσύνη και άλλες σύγχρονες τεχνολογίες να:

- να μάθει τα πιθανά αίτια για οποιοδήποτε σύμπτωμα τον απασχολεί
- να γνωρίζει αν πρόκειται για επείγον περιστατικό που απαιτεί άμεση διαχείριση
- να λάβει ιατρικές συμβουλές και πρώτες βοήθειες μέσω chat ή τηλεφώνου από εξειδικευμένο γιατρό
- να βρει τον πλησιέστερο κατάλληλο γιατρό και διαγνωστικό κέντρο από το ευρύ δίκτυο υγείας της **Interamerican**.

Η εφαρμογή είναι διαθέσιμη στο Google Play και στο App Store.

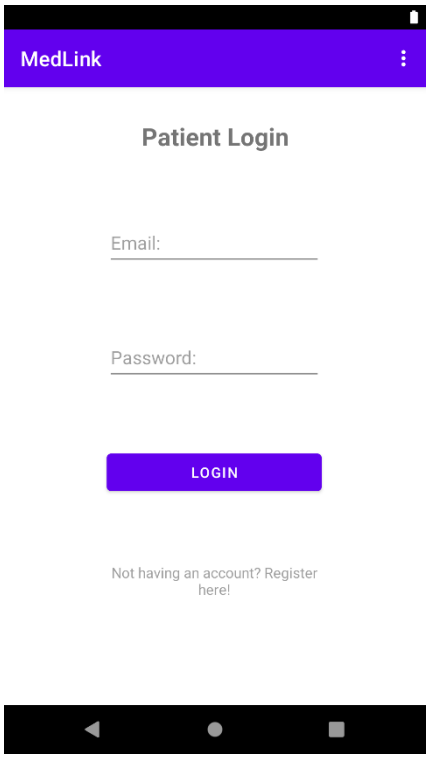
2.4 MediQuo Medical Chat

Το MediQuo Medical Chat σου δίνει τη δυνατότητα να απευθυνθείς σε εξειδικευμένο γιατρό, με τον οποίο μπορείς να επικοινωνήσει άμεσα για το θέμα υγείας που σε απασχολεί. Η εφαρμογή λειτουργεί με chat, μέσω του οποίου ο χρήστης ανταλλάσσει μηνύματα με το γιατρό που επιθυμεί. Αποτελεί μία εφαρμογή, η οποία χρησιμοποιείται ευρέως στο εξωτερικό.

Ενότητα 3 - Παρουσίαση και χρήση της εφαρμογής (User manual)

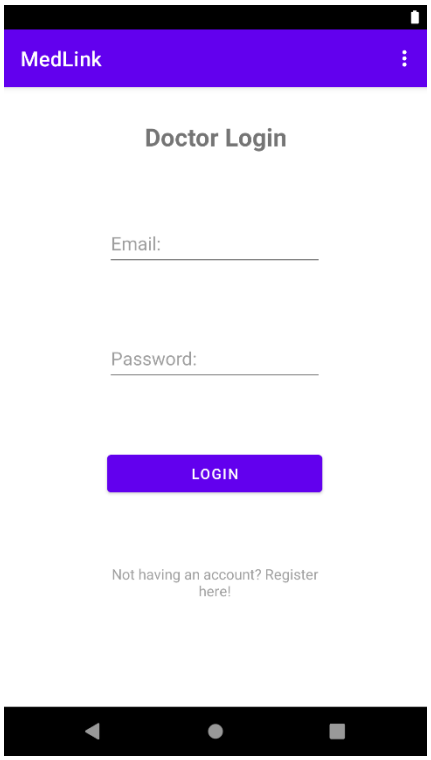
3.1 Οθόνη Σύνδεσης Ασθενή (Patient Login Screen)

Με το που ο χρήστης ξεκινήσει να χρησιμοποιεί την εφαρμογή, περιηγείται σε μία αρχική οθόνη σύνδεσης (login screen) σε αυτή, η οποία έχει ως προεπιλεγμένη επιλογή την ιδιότητα «Patient». Αυτό σημαίνει πως σε περίπτωση που ο χρήστης αποφασίσει να συνδεθεί στην εφαρμογή, θα συνδεθεί με την ιδιότητα του «Ασθενή» σε αυτή. Για την επιτυχημένη σύνδεση σε αυτή, ο χρήστης καλείται να πληκτρολογήσει ένα email, καθώς και ένα κωδικό πρόσβασης. Απαραίτητη προϋπόθεση είναι ο συγκεκριμένος λογαριασμός να υπάρχει. Σε περίπτωση που κάποιο από τα στοιχεία δεν είναι έγκυρα, εμφανίζεται το κατάλληλο μήνυμα στο χρήστη μέσω ενός toast message.



3.2 Οθόνη Σύνδεσης Γιατρού (Doctor Login Screen)

Εάν ο χρήστης επιθυμεί να συνδεθεί με την ιδιότητα του «Doctor», τότε αρκεί να πατήσει στις τρεις τελείες στο πάνω δεξιά μέρος της οθόνης και να επιλέξει «Doctor».

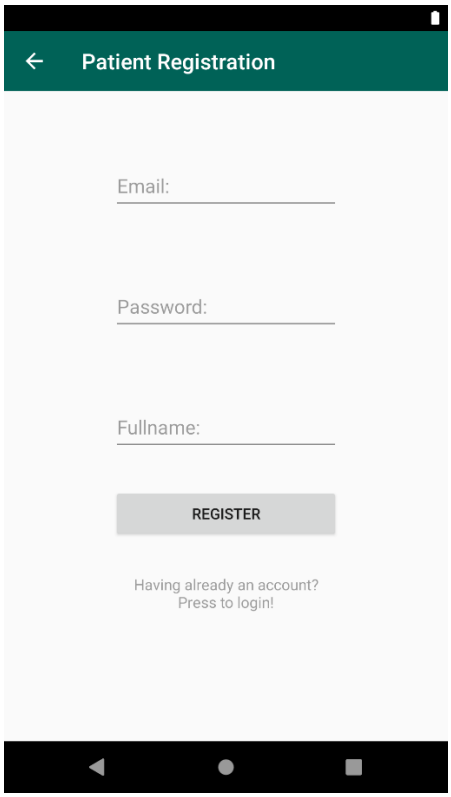


Σε πλήρη αντιστοιχία με την περίπτωση του Patient, ο χρήστης καλείται να πληκτρολογήσει ένα email, καθώς και ένα κωδικό πρόσβασης. Σε περίπτωση που κάποιο από τα στοιχεία δεν είναι έγκυρα, εμφανίζεται το κατάλληλο μήνυμα στο χρήστη μέσω ενός toast message.

Εάν ο χρήστης δεν έχει δημιουργήσει κάποιο λογαριασμό, μπορεί να δημιουργήσει έναν, πατώντας το ακόλουθο κείμενο: «Not having an account? Register here!».

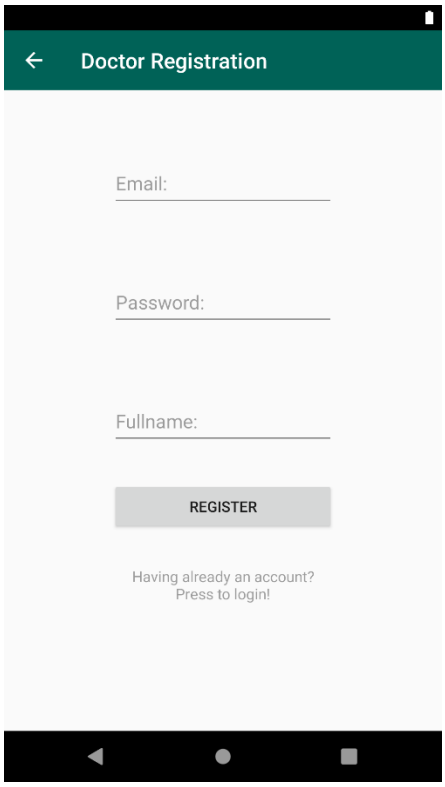
3.3 Οθόνη Δημιουργίας Λογαριασμού Ασθενή (Patient Registration Screen)

Όταν ο χρήστης πατήσει το προαναφερθέν κείμενο, μεταβαίνει σε μία οθόνη δημιουργίας λογαριασμού (registration screen). Στη συγκεκριμένη οθόνη, ο χρήστης μπορεί να εισάγει ένα email, έναν κωδικό πρόσβασης, καθώς και ένα ονοματεπώνυμο. Πατώντας το κουμπί «Register», ο χρήστης πραγματοποιεί έγγραφή στο σύστημα. Σε περίπτωση που κάποιο από τα στοιχεία είναι λάθος, εμφανίζεται το κατάλληλο μήνυμα στο χρήστη μέσω ενός toast message. Εάν ο χρήστης είχε επιλέξει την ιδιότητα του «Patient», θα μεταφερθεί στην οθόνη δημιουργίας λογαριασμού για ασθενή.



3.4 Οθόνη Δημιουργίας Λογαριασμού Γιατρού (Doctor Registration Screen)

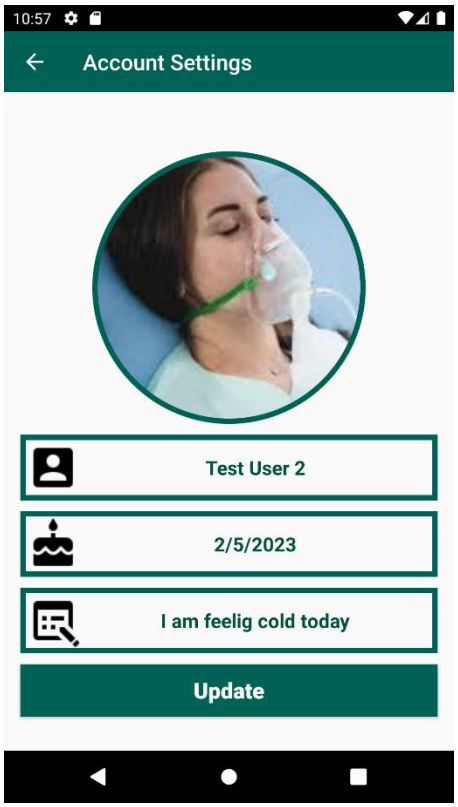
Εάν ο χρήστης έχει επιλέξει την ιδιότητα του «Doctor», θα μεταφερθεί στην οθόνη δημιουργίας λογαριασμού για γιατρό.



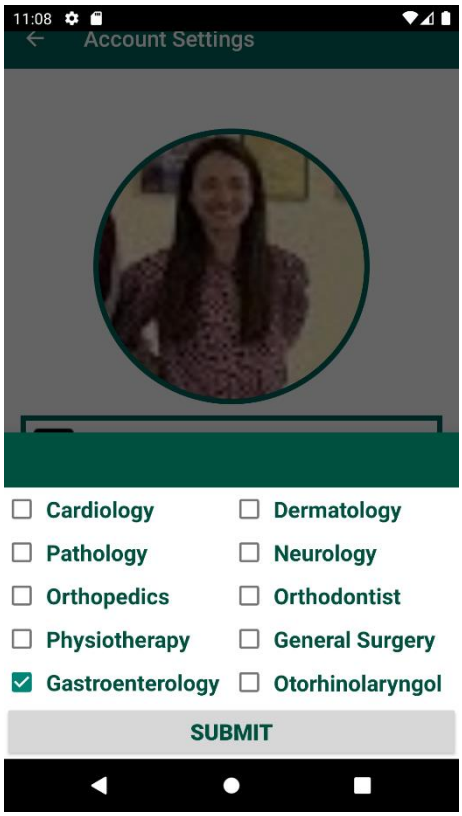
3.5 Οθόνη Ρύθμισης Λογαριασμού (Account Settings Screen)

Όταν ο χρήστης καταφέρει να συνδεθεί είτε μέσω της οθόνης σύνδεσης (login screen) είτε μέσω της οθόνης δημιουργίας λογαριασμού, μεταφέρεται στην κεντρική οθόνη της εφαρμογής. Εάν είναι η πρώτη φορά που ο χρήστης συνδέεται στην εφαρμογή, μεταφέρεται αυτομάτα σε μία οθόνη, προκειμένου να ορίσει τα στοιχειά του, τα οποία είναι απαραίτητα για το chat. Τα συγκεκριμένα στοιχεία είναι απαραίτητα, προκειμένου όταν υπάρξει αίτημα από τη μια μεριά προς την άλλη ή ξεκινήσει μια επικοινωνία μεταξύ τους, αυτά να γνωστοποιούνται διευκολύνοντας την επικοινωνία μεταξύ γιατρού και ασθενή. Πιο συγκεκριμένα:

- Όταν ένας ασθενής μεταφέρεται στην οθόνη συμπλήρωσης στοιχείων (account settings), τότε καλείται να συμπληρώσει μία εικόνα προφίλ, το ονοματεπώνυμό του (το οποίο είναι προ συμπληρωμένο, εφόσον είχε εισαχθεί στο βήμα δημιουργίας λογαριασμού), την ημερομηνία γέννησής του, καθώς και το status του. Το τελευταίο αποτελεί ένα κείμενο, μέσω του οποίου ο χρήστης προϊδεάζει τον ενδιαφερόμενο για την κατάσταση την οποία βρίσκεται. Για παράδειγμα ένας ασθενής, ο οποίος αναζητά άμεσα οδοντίατρο, θα μπορούσε να αναφέρει την εξής κατάσταση «I have severe tooth pain!». Με αυτό τον τρόπο ο γιατρός προ ιδεάζεται σχετικά με το πιθανό αίτημα που πρόκειται να δεχθεί από τον ασθενή. Όταν ο χρήστης πατήσει πάνω στην εικόνα προφίλ του, ανοίγει ο φάκελος με τις φωτογραφίες, μέσω του οποίου ο χρήστης επιλέγει την φωτογραφία που επιθυμεί να εισάγει και για την οποία έχει τη δυνατότητα περικοπής (crop). Προκειμένου ο χρήστης να εισάγει την ημερομηνία γέννησής του, ένα ημερολόγιο εμφανίζεται πατώντας το text field «Date of Birth», προκειμένου να εισάγει την ημερομηνία γέννησής του. **Σε αυτό το σημείο είναι απαραίτητο να αναφέρουμε ότι για να συνεχίσει ο χρήστης να περιηγείται στην εφαρμογή, πρέπει να συμπληρώσει την ημερομηνία γέννησης και την εικόνα προφίλ του. Αν δεν το κάνει αυτό, δε μπορεί να συνεχίσει την πλοήγησή του εντός της εφαρμογής.** Για την αποθήκευση της ημερομηνίας γέννησης και του status, απαιτείται το πάτημα του κουμπιού «Update». Για την αποθήκευση νέας εικόνας προφίλ, αρκεί η επιτυχημένη αλλαγή εικόνας.

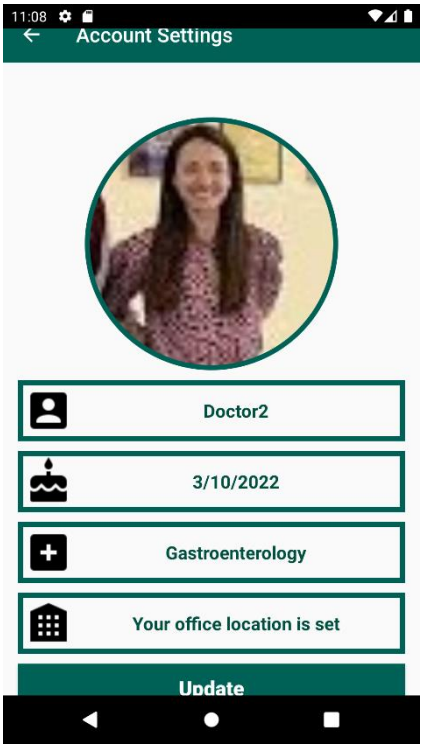


- Όταν ένας γιατρός μεταφέρεται στην οθόνη συμπλήρωσης στοιχείων (account settings), τότε καλείται να συμπληρώσει μια εικόνα προφίλ, το ονοματεπώνυμό του (το οποίο είναι προ συμπληρωμένο, εφόσον έχει εισαχθεί στο βήμα δημιουργίας λογαριασμού), την ημερομηνία γέννησής του, την ειδικότητά του, καθώς και την τοποθεσία του ιατρείου στο οποίο βρίσκεται. **Σε αυτό το σημείο είναι απαραίτητο να αναφέρουμε ότι για να συνεχίσει ο χρήστης να περιηγείται στην εφαρμογή, πρέπει να συμπληρώσει την ημερομηνία γέννησης, την εικόνα προφίλ του, καθώς και την ειδικότητά του. Αν δεν το κάνει αυτό, δε μπορεί να συνεχίσει την πλοήγησή του εντός της εφαρμογής.** Όταν ο χρήστης πατήσει πάνω στην εικόνα προφίλ του, ανοίγει ο φάκελος με τις φωτογραφίες, μέσω του οποίου ο χρήστης επιλέγει την φωτογραφία που επιθυμεί να εισάγει και για την οποία έχει τη δυνατότητα περικοπής (crop). Προκειμένου ο χρήστης να εισάγει την ημερομηνία γέννησής του, ένα ημερολόγιο εμφανίζεται πατώντας το text field «Date of Birth», προκειμένου να εισάγει την ημερομηνία γέννησής του. Στη συνέχεια ο χρήστης καλείται να εισάγει μία ειδικότητα. Για να το πετύχει αυτό, εμφανίζεται ένα bottom sheet dialog με διάφορες ειδικότητες. Ο χρήστης επιλέγει μία εξ αυτών πατώντας στο αντίστοιχο checkbox.

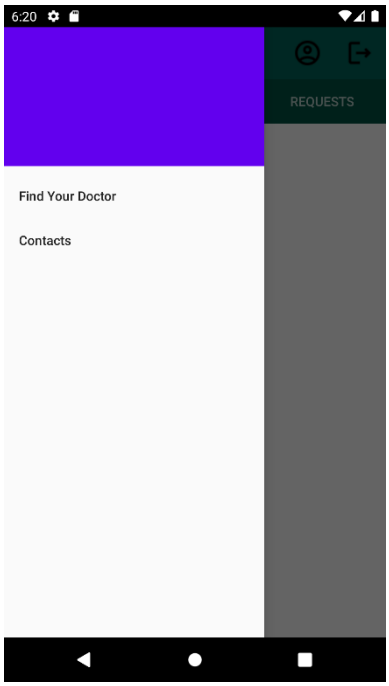


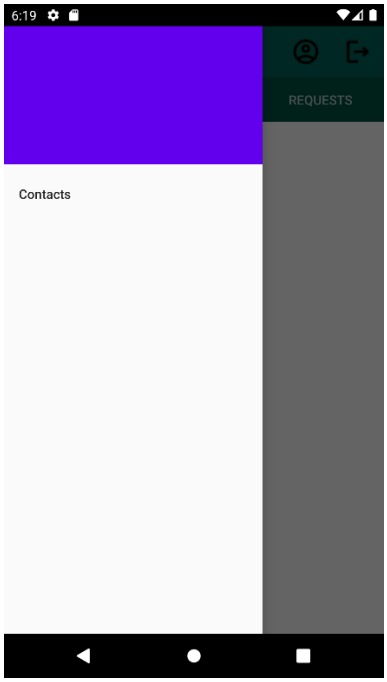
Στη συνέχεια, ο χρήστης μπορεί να πατήσει πάνω στο text field office location, μέσω του οποίου περιηγείται σε μία καινούρια οθόνη, προκειμένου να εισάγει την τοποθεσία του ιατρείου του. Πιο συγκεκριμένα, πατώντας παρατεταμένα το marker στον χάρτη, ο marker ενεργοποιείται και είναι έτοιμος για μετακίνηση εντός του χάρτη. Πατώντας «Confirm Office Location», αποθηκεύεται προσωρινά η τοποθεσία του ιατρείου του χρήστη.

Το τελικό βήμα για την αποθήκευση των στοιχείων του χρήστη, απαιτεί το πάτημα του κουμπιού «Update», το οποίο αποθηκεύει: την ημερομηνία γέννησης, την ειδικότητα, καθώς και την τοποθεσία του ιατρείου του χρήστη. Για την αποθήκευση νέας εικόνας προφίλ, αρκεί η επιτυχημένη αλλαγή εικόνας.



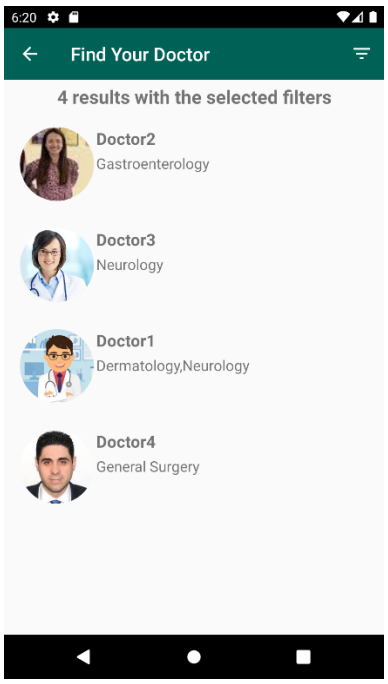
Αφότου ο χρήστης, ανεξαρτήτως ιδιότητας (Patient ή Doctor), ολοκληρώσει με την ενημέρωση των στοιχείων (account settings), μπορεί πλέον να συνεχίσει με την πλοήγησή του στο υπόλοιπο της εφαρμογής. Και αυτό είναι το σημείο στο οποίο η λειτουργικότητα μεταξύ ασθενή και γιατρού διαφέρει. Πιο συγκεκριμένα, ένας ασθενής μπορεί να αναζητήσει ένα γιατρό και **όχι το ανάποδο.** Για να δούμε τη διαφορά ανάμεσα στη λειτουργικότητα των δύο ιδιοτήτων, αρκεί μία ματιά στο slide menu που ενεργοποιείται πατώντας το burger button της εφαρμογής. Όπως παρατηρούμε στην περίπτωση του ασθενούς, το slide menu υποστηρίζει δύο λειτουργίες: **Find Your Doctor, Contacts**, ενώ στην περίπτωση του γιατρού υπάρχει μόνο η δεύτερη.





3.6 Οθόνη Εύρεσης του Δικού σου Γιατρού (Find Your Doctor Screen)

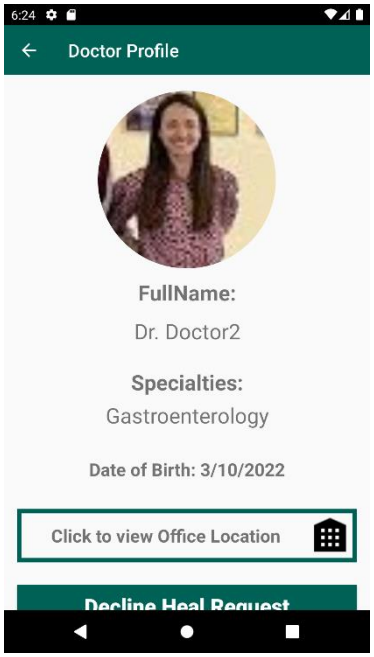
Όταν ο ασθενής πατήσει πάνω στην επιλογή Find Your Doctor, μεταβαίνει σε μία καινούρια οθόνη αποτελεσμάτων με όλους τους διαθέσιμους γιατρούς. Στο πάνω μέρος την οθόνης εμφανίζεται το πλήθος των αποτελεσμάτων αναζήτησης, ενώ στο κέντρο της οθόνης εμφανίζονται τα διάφορα αποτελέσματα.



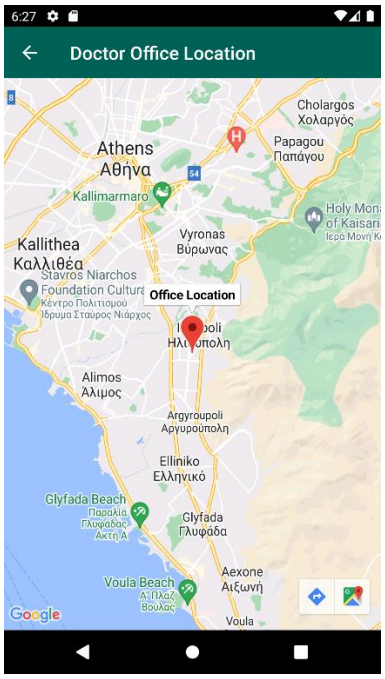
Ο ασθενής καλείται να πατήσει σε όποιο γιατρό τον ενδιαφέρει και με τον οποίο επιθυμεί να έλθει σε επικοινωνία. Ας υποθέσουμε ότι ο ασθενής επιλέγει τον πρώτο γιατρό στην λίστα αποτελεσμάτων.

Πατώντας πάνω σε αυτόν, μεταβαίνει σε μία καινούρια οθόνη, η οποία περιγράφει αναλυτικά τα διάφορα στοιχεία του γιατρού.

3.7 Οθόνη Στοιχείων Γιατρού (Find Doctor Profile Screen)



Τα στοιχεία τα οποία αναγράφονται, είναι πρακτικά τα στοιχεία τα οποία είχε εισάγει ο γιατρός στο βήμα account settings που περιεγράφηκε νωρίτερα. Παρατηρούμε λοιπόν ότι στην οθόνη φαίνεται: η εικόνα προφίλ του γιατρού, το ονοματεπώνυμό του, η ειδικότητά του, η ημερομηνία γέννησής του, καθώς και ένα text field στο οποίο μπορεί να πατήσει για να δει την τοποθεσία του ιατρείου του.



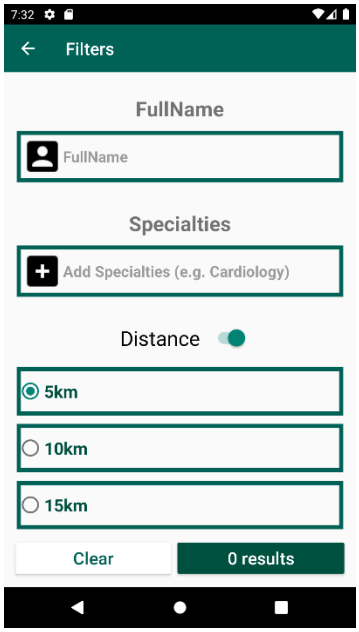
Στο κάτω μέρος της οθόνης υπάρχει ένα πράσινο κουμπί. Το κουμπί αυτό αλλάζει περιεχόμενο, με βάση την κατάσταση επικοινωνίας μεταξύ γιατρού και ασθενή. Προκειμένου να ανοίξει ο δίαυλος

επικοινωνίας μεταξύ των δύο, το αρχικό κείμενο του κουμπιού περιέχει το λεκτικό «Send Heal Request». Εάν ο χρήστης το πατήσει, στέλνεται αυτομάτως ένα αίτημα προς το γιατρό. Το κουμπί με τη σειρά του αλλάζει λεκτικό και γίνεται «Decline Heal Request». Εάν ο χρήστης το πατήσει ξανά, τότε ακυρώνεται το προηγούμενο αίτημα προς το γιατρό. Με αυτό τον τρόπο λοιπόν μπορεί ένας ασθενής να επικοινωνήσει με ένα γιατρό, προκειμένου να έλθει σε επικοινωνία.

Συνεχίζοντας με τις υπόλοιπες λειτουργίες της εφαρμογής, ο χρήστης μπορεί να επιλέξει κάποια φίλτρα στην αναζήτησή του. Για να το κάνει αυτό, αρκεί να πατήσει στο πάνω δεξιό μέρος της οθόνης του το κουμπί των φίλτρων και αυτομάτως πλοηγείτε στην αντίστοιχη οθόνη. Τα φίλτρα που μπορεί να ορίσει ο χρήστης, στο σύνολό τους είναι τρία:

- Ονοματεπώνυμο (FullName): Ο χρήστης ξεκινάει να πληκτρολογεί κάποιους χαρακτήρες και αν υπάρξει ταύτιση με το ονοματεπώνυμο κάποιου γιατρού, η τελική αναζήτηση θα περιέχει μόνο αυτό.
- Ειδικότητες (Specialties): Σε πλήρη αντιστοιχία με τις ειδικότητες στο βήμα Account Settings για γιατρούς, ο ασθενής επιλέγει την ειδικότητα που τον ενδιαφέρει μέσα από ένα σύνολο επιλογών που εμφανίζονται μέσω ενός bottom sheet dialog μενού.
- Απόσταση (Distance): Αφού πρώτα ο ασθενής δώσει άδεια στην εφαρμογή να αποκτήσει πρόσβαση στην τρέχουσα τοποθεσία του, εκείνος μπορεί να αναζητήσει γιατρούς σε μία απόσταση 5, 10, 15km από το τρέχων στίγμα.

Για κάθε επιλογή που κάνει ο χρήστης ή πληκτρολόγηση χαρακτήρα, πραγματοποιείται εκ νέου μια αναζήτηση γιατρών, το αποτέλεσμα της οποίας εμφανίζεται στο κάτω δεξιά πράσινο κουμπί. Με αυτόν τον τρόπο, ο χρήστης είναι πάντα ενήμερος σχετικά με το πλήθος των αποτελεσμάτων και προ ιδεάζεται σχετικά με το τι να περιμένει, αν αποφασίσει να πατήσει το κουμπί της αναζήτησης. Τέλος ο χρήστης έχει τη δυνατότητα καθαρισμού των φίλτρων, που οδηγεί σε νέα αναζήτηση χωρίς φίλτρα.



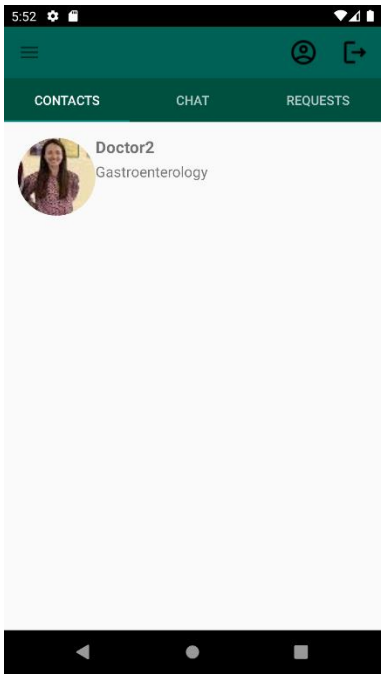
3.8 Οθόνη Κύριων Επαφών (Main Contacts Screen)

Η συγκεκριμένη οθόνη αποτελεί κοινή είτε ο χρήστης έχει την ιδιότητα του Ασθενή, είτε εκείνη του Γιατρού. Πιο συγκεκριμένα, η οθόνη απαρτίζεται από 3 τμήματα (tabs): Επαφές (Contacts), Αιτήματα (Requests) και Chat. Στο πάνω αριστερά μέρος υπάρχει ένα κουμπί (burger button), το οποίο μόλις πατηθεί οδηγεί στην εμφάνιση μίας slide καρτέλας μενού από αριστερά. Όπως έχουμε ήδη αναφέρει, εδώ είναι ένα σημείο στο οποίο διαφοροποιείται η λειτουργικότητα μεταξύ των 2 ιδιοτήτων. Στην

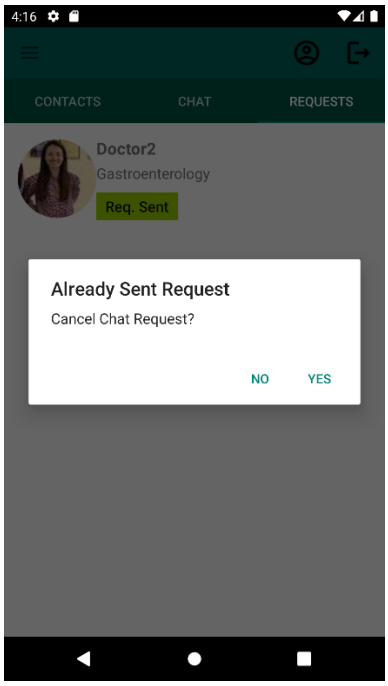
περίπτωση του ασθενή υπάρχουν 2 επιλογές: Εύρεση Γιατρού (Find Your Doctor) και Επαφές (Contacts), ενώ στην περίπτωση του Ασθενούς υπάρχει μόνο το δεύτερο.

3.9 Οθόνη Αιτημάτων Ασθενούς (Patient Tab Requests Screen)

Σε περίπτωση που ο ασθενής έχει κάνει αίτημα σε κάποιο γιατρό, στο συγκεκριμένο tab εμφανίζεται η καρτέλα του γιατρού.

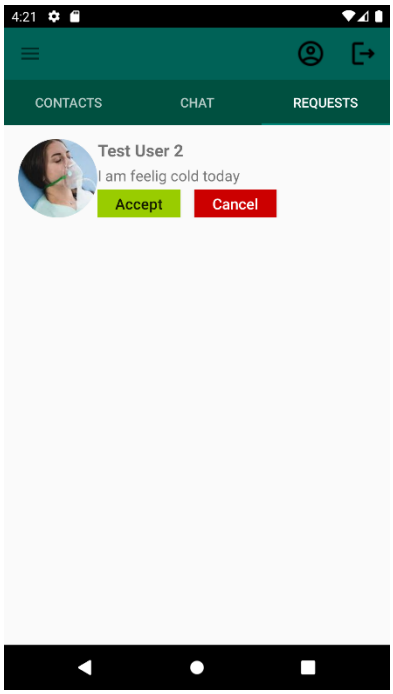


Πιο συγκεκριμένα, εμφανίζεται το ονοματεπώνυμο του γιατρού, καθώς και η ειδικότητά του. Το συγκεκριμένο τμήμα περιέχει μία λίστα με όλα τα αιτήματα που βρίσκονται υπό αναμονή – όλα τα αιτήματα προς γιατρούς που έχει πραγματοποιήσει ο ασθενής. Ο ασθενής πατώντας πάνω στην καρτέλα του γιατρού, του εμφανίζεται ένα νέο αναδυόμενο παράθυρο (pop up window), το οποίο τον ρωτάει αν επιθυμεί να ακυρώσει το Αίτημα Υγείας (Cancel Heal Request). Αν πατήσει «Ναι», το αίτημα ακυρώνεται κι η αντίστοιχη καρτέλα γιατρού διαγράφεται από τη λίστα. Σε περίπτωση που πατήσει «Όχι», το αναδυόμενο παράθυρο εξαφανίζεται.



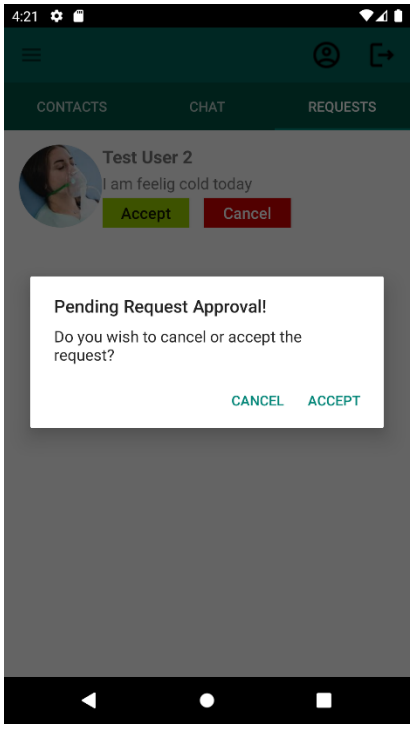
3.10 Οθόνη Αιτημάτων Γιατρού (Doctor Tab Requests Screen)

Σε περίπτωση που ο γιατρός έχει δεχθεί κάποιο αίτημα από ασθενή, στο συγκεκριμένο tab εμφανίζεται η καρτέλα του ασθενούς.



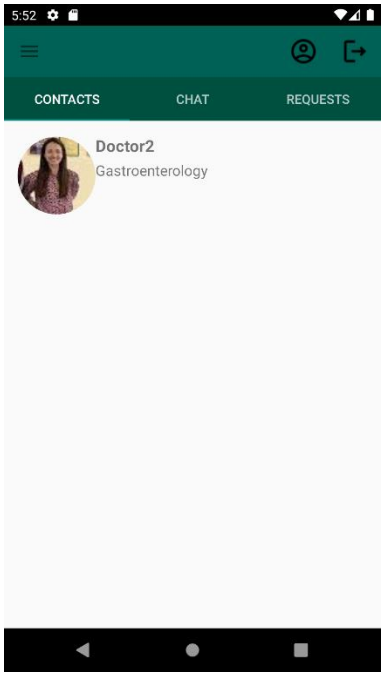
Πιο συγκεκριμένα, εμφανίζεται το ονοματεπώνυμο του ασθενούς, καθώς και η κατάστασή του(status), αν έχει εισαγάγει κάποια. Το συγκεκριμένο τμήμα περιέχει μία λίστα με όλα τα αιτήματα που βρίσκονται υπό αναμονή – όλα των ασθενών που έχουν γίνει προς το συγκεκριμένο γιατρό. Ο γιατρός πατώντας πάνω στην καρτέλα του ασθενούς, του εμφανίζεται ένα νέο αναδυόμενο παράθυρο (pop up window), το οποίο τον ρωτάει αν επιθυμεί να πραγματοποιήσει μία από τις ακόλουθες δύο κινήσεις: να δεχθεί το

Αίτημα Υγείας (Accept Heal Request), να ακυρώσει το Αίτημα Υγείας (Cancel Heal Request). Αν πατήσει «Αποδοχή», το αίτημα γίνεται δεκτό κι η αντίστοιχη καρτέλα ασθενούς διαγράφεται από τη λίστα. Σε περίπτωση που πατήσει «Ακύρωση», το αίτημα δεν γίνεται αποδεκτό και η αντίστοιχη καρτέλα ασθενούς διαγράφεται από τη λίστα.



3.11 Οθόνη Επαφών Ασθενούς (Patient Tab Contacts Screen)

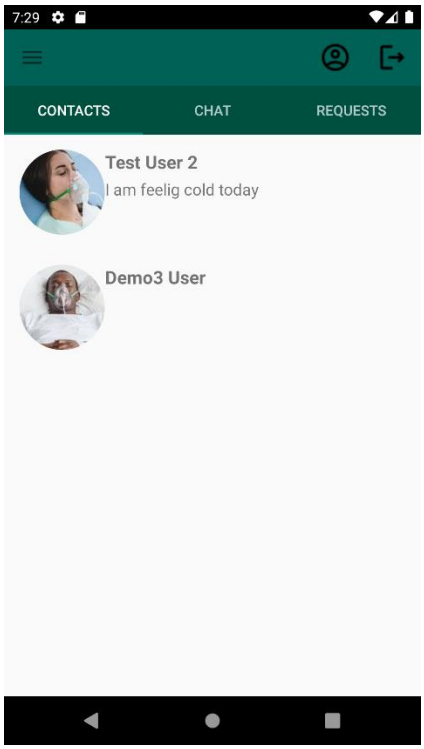
Σε περίπτωση που το αίτημα κάποιου ασθενούς γίνει αποδεκτό από κάποιον γιατρό, τότε οι αντίστοιχες καρτέλες γιατρών εμφανίζονται σε μία λίστα.



Πιο συγκεκριμένα, για κάθε καρτέλα εμφανίζεται το ονοματεπώνυμο του γιατρού μαζί με την ειδικότητά του. Η συγκεκριμένη λίστα, απλά ενημερώνει το χρήστη σχετικά με τις διαθέσιμες επαφές που έχει.

3.12 Οθόνη Επαφών Γιατρού (Doctor Tab Contacts Screen)

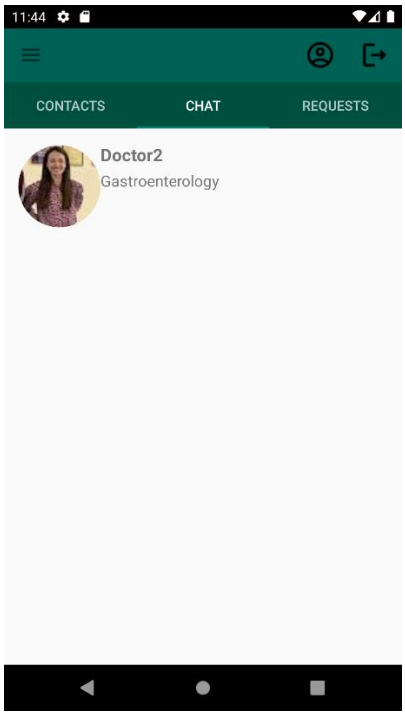
Σε περίπτωση που το αίτημα κάποιου ασθενούς γίνει αποδεκτό από κάποιον γιατρό, τότε οι αντίστοιχες καρτέλες ασθενών εμφανίζονται σε μία λίστα.



Πιο συγκεκριμένα, για κάθε καρτέλα εμφανίζεται το ονοματεπώνυμο του ασθενούς μαζί με την κατάστασή (status) του αν υπάρχει. Η συγκεκριμένη λίστα, απλά ενημερώνει το χρήστη σχετικά με τις διαθέσιμες επαφές που έχει.

3.13 Οθόνη Μηνυμάτων Ασθενούς (Patient Tab Chat Screen)

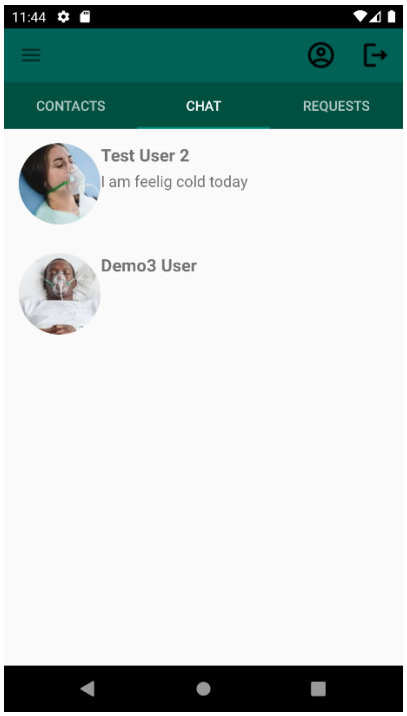
Σε περίπτωση που το αίτημα κάποιου ασθενούς γίνει αποδεκτό από κάποιον γιατρό, τότε οι αντίστοιχες καρτέλες γιατρών εμφανίζονται σε μία λίστα.



Πιο συγκεκριμένα, για κάθε καρτέλα εμφανίζεται το ονοματεπώνυμο του γιατρού μαζί με την ειδικότητά του. Η συγκεκριμένη λίστα, απλά ενημερώνει το χρήστη σχετικά με τις διαθέσιμες επαφές που έχει, με τις οποίες έχει τη δυνατότητα να ανταλλάξει ένα μήνυμα. Ο ασθενής μπορεί να πατήσει πάνω σε μία καρτέλα, το οποίο θα τον οδηγήσει σε μία καινούρια οθόνη ανταλλαγής μηνυμάτων.

3.14 Οθόνη Μηνυμάτων Γιατρού (Doctor Tab Chat Screen)

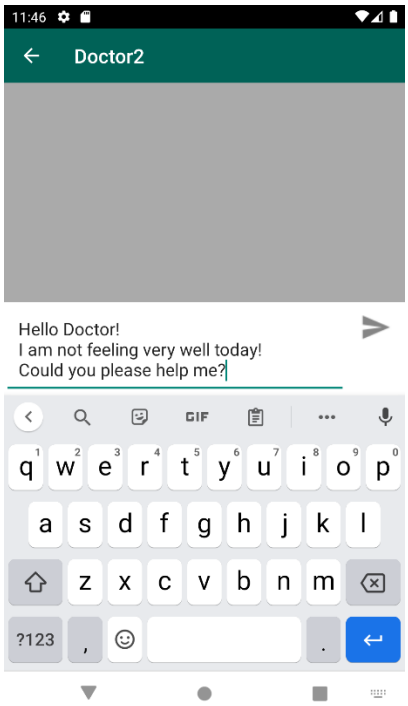
Σε περίπτωση που το αίτημα κάποιου ασθενούς γίνει αποδεκτό από κάποιον γιατρό, τότε οι αντίστοιχες καρτέλες ασθενών εμφανίζονται σε μία λίστα.



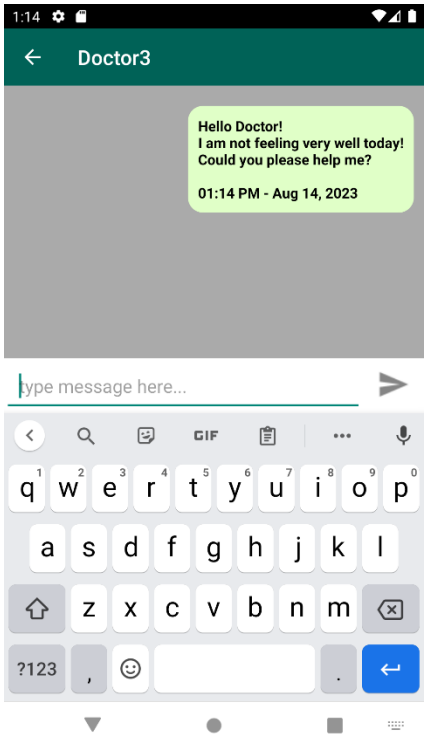
Πιο συγκεκριμένα, για κάθε καρτέλα εμφανίζεται το ονοματεπώνυμο του ασθενούς μαζί με την κατάστασή (status) του αν υπάρχει. Η συγκεκριμένη λίστα, απλά ενημερώνει το χρήστη σχετικά με τις διαθέσιμες επαφές που έχει, με τις οποίες έχει τη δυνατότητα να ανταλλάξει ένα μήνυμα. Ο ασθενής μπορεί να πατήσει πάνω σε μία καρτέλα, το οποίο θα τον οδηγήσει σε μία καινούρια οθόνη ανταλλαγής μηνυμάτων.

3.15 Οθόνη Ανταλλαγής Μηνυμάτων Ασθενούς (Patient Chat Messages Screen)

Η τρέχουσα οθόνη είναι κοινή ως προς την λειτουργικότητά της, ανεξαρτήτως ιδιότητας χρήστη (Ασθενής ή Γιατρός). Όταν ο ασθενής εισάγεται στη συγκεκριμένη οθόνη, στο πάνω μέρος εμφανίζεται το όνομα του γιατρού με το οποίο θα επικοινωνήσει. Στο πάνω δεξιά μέρος υπάρχει ένα «Πίσω Κουμπί» (Up Button), με το οποίο μπορεί να κατευθυνθεί στην προηγούμενη οθόνη. Στο κάτω μέρος της οθόνης υπάρχει ένα πεδίο εισαγωγής κειμένου (edit text). Όταν ο ασθενής πατήσει πάνω σε αυτό, εμφανίζεται ένα πληκτρολόγιο κινητού. Ο ασθενής έχει την ευκαιρία με αυτό τον τρόπο, να πληκτρολογήσει ένα μήνυμα προς το γιατρό, το οποίο είναι έτοιμο προς αποστολή.

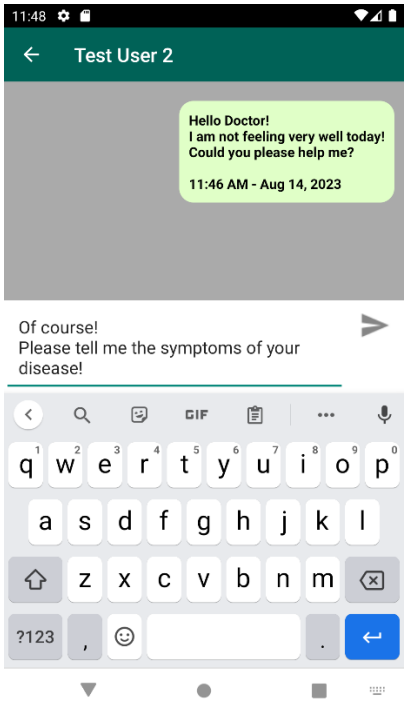


Στο δεξιά μέρος του πεδίου εισαγωγής κειμένου, υπάρχει ένα βέλος με το οποίο ο ασθενής αποστέλλει το μήνυμα που επιθυμεί προς τον γιατρό. Όταν πατήσει το κουμπί, το μήνυμα αναγράφεται πάνω στην οθόνη μαζί με την ημερομηνία και την ώρα αποστολής του κειμένου.

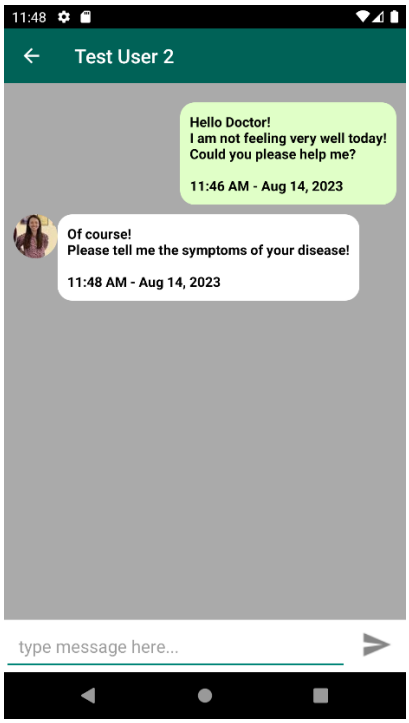


3.16 Οθόνη Ανταλλαγής Μηνυμάτων Γιατρού (Doctor Chat Messages Screen)

Η τρέχουσα οθόνη είναι κοινή ως προς την λειτουργικότητά της, ανεξαρτήτως ιδιότητας χρήστη (Ασθενής ή Γιατρός). Όταν ο γιατρός εισάγεται στη συγκεκριμένη οθόνη, στο πάνω μέρος εμφανίζεται το όνομα του ασθενούς με το οποίο θα επικοινωνήσει. Στο πάνω δεξιά μέρος υπάρχει ένα «Πίσω Κουμπί» (Up Button), με το οποίο μπορεί να κατευθυνθεί στην προηγούμενη οθόνη. Στο κάτω μέρος της οθόνης υπάρχει ένα πεδίο εισαγωγής κειμένου (edit text). Όταν ο γιατρός πατήσει πάνω σε αυτό, εμφανίζεται ένα πληκτρολόγιο κινητού. Ο γιατρός έχει την ευκαιρία με αυτό τον τρόπο, να πληκτρολογήσει ένα μήνυμα προς το γιατρό, το οποίο είναι έτοιμο προς αποστολή.



Στο δεξιά μέρος του πεδίου εισαγωγής κειμένου, υπάρχει ένα βέλος με το οποίο ο γιατρός αποστέλλει το μήνυμα που επιθυμεί προς τον ασθενή. Όταν πατήσει το κουμπί, το μήνυμα αναγράφεται πάνω στην οθόνη μαζί με την ημερομηνία και την ώρα αποστολής του κειμένου.



Ενότητα 4 - Αρχιτεκτονική Συστήματος

Προκειμένου να κατανοήσουμε την αρχιτεκτονική του συστήματος που χρησιμοποιήθηκε στα πλαίσια της παρούσης μεταπτυχιακής διατριβής, είναι σημαντικό να κατανοήσουμε το πως έχει δομηθεί το όλο project. Πιο συγκεκριμένα, το project απαρτίζεται από δύο κομμάτια:

1. μία εφαρμογή κινητού (mobile app) και
2. ένα τοπικό service

Σχετικά με το πρώτο, η εφαρμογή κινητού είναι γραμμένη σε γλώσσα προγραμματισμού Kotlin, η υλοποίησή της έγινε μέσω του IDE Android Studio και η αρχιτεκτονική που έχει επιλεχθεί για την υλοποίησή της είναι αυτή του MVVM (Model-View-ViewModel). Η εφαρμογή σε επίπεδο κώδικα έχει ακολουθήσει την τεχνική του modularization (σπάσιμο κώδικα σε modules, όπου το καθένα περιέχει κώδικα που επαφίεται εννοιολογικά σε συγκεκριμένη ενότητα).

Σχετικά με το δεύτερο, για το τοπικό service έχει ακολουθηθεί η αρχιτεκτονική του Spring Boot, η υλοποίησή του έγινε χρησιμοποιώντας το IDE IntelliJ Ultimate, ενώ η γλώσσα προγραμματισμού είναι Java.

Ο τελικός στόχος του project, είναι να αποτυπωθεί η τελική πληροφορία στο χρήστη (πληροφορία για γιατρούς, ασθενείς, ανταλλαγή μηνυμάτων). Η πληροφορία αυτή πρέπει να είναι αποθηκευμένη κάπου και να ανανεώνεται δυναμικά, όταν χρειάζεται. Για το λόγο αυτό χρησιμοποιούνται δύο βάσεις δεδομένων: μία σχεσιακή και μία μη σχεσιακή ΒΔ. Πιο συγκεκριμένα:

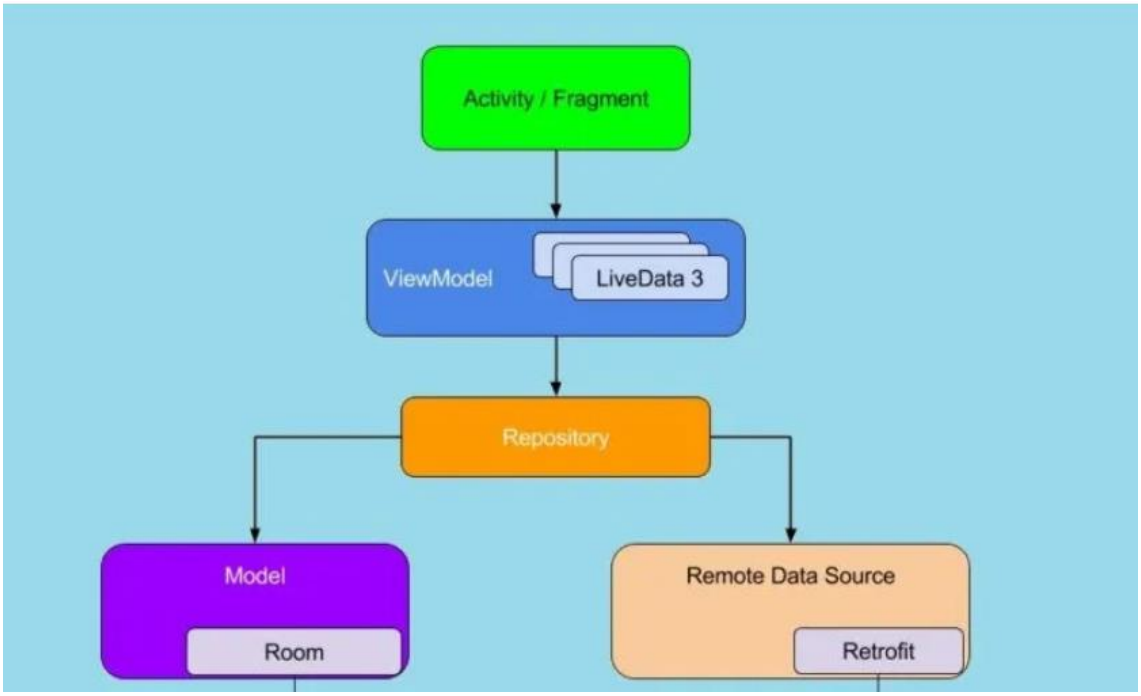
Το τοπικό service επικοινωνεί με μία σχεσιακή ΒΔ, η οποία είναι υπεύθυνη να κρατά τις βασικές πληροφορίες του χρήστη. Για την υλοποίησή της χρησιμοποιήθηκε μία ΒΔ τύπου MySQL, ενώ το λογισμικό που χρησιμοποιήθηκε για την υλοποίηση αυτού είναι το XAMPP.

Η εφαρμογή κινητού επικοινωνεί με μία μη σχεσιακή ΒΔ, η οποία είναι υπεύθυνη να κρατά τις πληροφορίες που χρειάζεται ο κάθε χρήστης για την τελική επικοινωνία μέσω chat. Για την υλοποίησή της χρησιμοποιήθηκε το λογισμικό της Firebase (Firebase Realtime Database και Firebase Storage). Έχοντας αναφέρει εισαγωγικά τις τεχνολογίες που χρησιμοποιήθηκαν, ακολουθεί ανάλυση αυτών σε βάθος:

4.1 Αρχιτεκτονική MVVM (Model-View-ViewModel)

Υπάρχουν πολλές διαφορετικές αρχιτεκτονικές που μπορούν να χρησιμοποιήσουν οι προγραμματιστές για να δημιουργήσουν τις εφαρμογές τους. Μια τέτοια αρχιτεκτονική είναι το MVVM (Model-View-ViewModel), το οποίο έχει αποκτήσει δημοτικότητα με τα χρόνια λόγω των πολλών πλεονεκτημάτων του. Στη παρούσα ενότητα εξηγούμε: τι είναι το MVVM, τα πλεονεκτήματα και τα μειονεκτήματά του.

4.1.1 Τι είναι η αρχιτεκτονική MVVM



Το MVVM είναι μια αρχιτεκτονική που διαχωρίζει τη διεπαφή χρήστη (UI) μιας εφαρμογής από την επιχειρηματική της λογική (business logic). Αποτελείται από τρία κύρια στοιχεία: Model, View και ViewModel.

Model: Το μοντέλο αντιπροσωπεύει τα δεδομένα και την επιχειρηματική λογική (business logic) της εφαρμογής. Ανακτά δεδομένα από διάφορες πηγές και τα προετοιμάζει για παρουσίαση στον χρήστη.

View: Η προβολή είναι το στοιχείο διεπαφής χρήστη της εφαρμογής (UI). Εμφανίζει τα δεδομένα στον χρήστη και του επιτρέπει να αλληλοεπιδράσει με αυτά.

ViewModel: Το ViewModel λειτουργεί ως ενδιάμεσος μεταξύ της προβολής (View) και του μοντέλου (Model). Επεξεργάζεται τα δεδομένα των χρηστών από την Προβολή (View) και ενημερώνει το Μοντέλο (Model) ανάλογα. Ενημερώνει επίσης την προβολή με τα πιο πρόσφατα δεδομένα από το μοντέλο.

4.1.2 Πλεονεκτήματα του MVVM

Separation of Concerns: Το MVVM διαχωρίζει τις ευθύνες του UI, της επιχειρηματικής λογικής και των δεδομένων. Αυτό διευκολύνει τη διατήρηση και την τροποποίηση του codebase.

Testability: Το MVVM διευκολύνει τη σύνταξη δοκιμών μονάδων για τα επίπεδα επιχειρηματικής λογικής και δεδομένων. Το ViewModel μπορεί να δοκιμαστεί ανεξάρτητα από τη διεπαφή χρήστη και το μοντέλο μπορεί να ελεγχθεί ανεξάρτητα τόσο από το UI όσο και από το ViewModel.

Flexibility: Το MVVM επιτρέπει στους προγραμματιστές να τροποποιούν τη διεπαφή χρήστη χωρίς να επηρεάζουν την επιχειρηματική λογική ή τα επίπεδα δεδομένων. Αυτό διευκολύνει την πραγματοποίηση αλλαγών στη διεπαφή χρήστη χωρίς διακοπή της εφαρμογής.

Reusability: Το MVVM επιτρέπει στους προγραμματιστές να επαναχρησιμοποιούν κώδικα σε πολλές προβολές. Αυτό διευκολύνει τη διατήρηση μιας συνεπούς εμφάνισης και αίσθησης σε διαφορετικές οθόνες της εφαρμογής.

4.1.3 Μειονεκτήματα του MVVM

Complexity: Το MVVM μπορεί να είναι πιο περίπλοκο από άλλες αρχιτεκτονικές, ειδικά για μικρά έργα. Απαιτεί μια σταθερή κατανόηση data binding, observables και άλλων εννοιών.

Steep learning curve: Οι προγραμματιστές που είναι νέοι στο MVVM μπορεί να δυσκολεύονται να ενημερωθούν με την αρχιτεκτονική. Απαιτεί διαφορετικό τρόπο σκέψης σε σύγκριση με άλλες αρχιτεκτονικές.

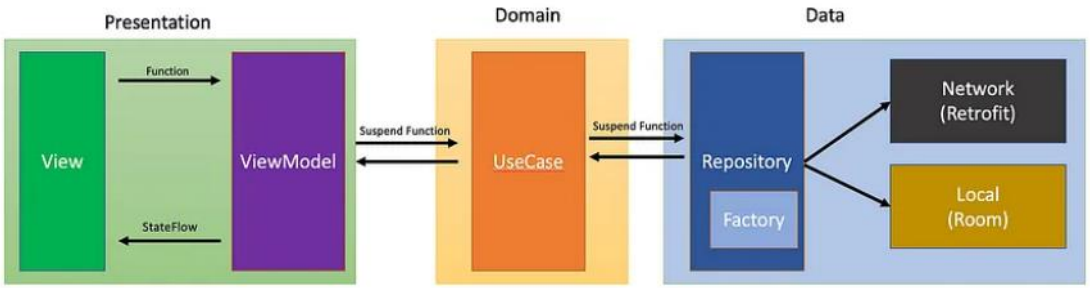
Performance overhead: Το MVVM μπορεί να έχει επιβάρυνση απόδοσης λόγω της διαδικασίας δέσμευσης δεδομένων. Αυτό μπορεί να μετριαστεί με καλές πρακτικές σχεδιασμού και τεχνικές βελτιστοποίησης.

4.1.4 Επίπεδα (Layers) της αρχιτεκτονικής MVVM

Όπως αναφέρθηκε ήδη, η αρχιτεκτονική του MVVM στοχεύει στο να διαχωρίσει την προβολή (view – UI) και την επιχειρηματική λογική (business logic). Εφαρμόζοντας τις αρχές του Clean Code, οι προγραμματιστές μπορούν να διαχωρίσουν αποτελεσματικά αυτές τις ανησυχίες, με αποτέλεσμα ένα καθαρό, ισχυρό και εύκολα συντηρήσιμο έργο. Ωστόσο, ένα μειονέκτημα του MVVM είναι ότι σε μεγαλύτερα έργα, το ViewModel συχνά επιβαρύνεται με υπερβολική επιχειρηματική λογική. Σε τέτοια σενάρια, η χρήση του MVVM παράλληλα με τις πρακτικές Clean Code μπορεί να αποδειχθεί ανεκτίμητη, καθώς επιτρέπει τον διαχωρισμό των ευθυνών, προάγει τη σταθερότητα του έργου και διευκολύνει την αβίαστη εξέλιξη και συντήρηση.

Το έργο χωρίζεται σε τρία επίπεδα:

1. Δεδομένα (Data)
2. Τομέα (Domain)
3. Παρουσίαση (Presentation)



Data Layer

Όπως υποδηλώνει το όνομα, περιέχει τα δεδομένα μας. Τα API interfaces, databases, Repository Implementation βρίσκονται σε αυτό το επίπεδο.

Το επίπεδο δεδομένων στην καθαρή αρχιτεκτονική αναλαμβάνει το ρόλο του χειρισμού της ροής δεδομένων και της ανάκτησης πληροφοριών από διάφορες πηγές όπως βάσεις δεδομένων, υπηρεσίες web και αρχεία. Διαχειρίζεται περαιτέρω τη διατήρηση των δεδομένων και διευκολύνει την πρόσβαση στα δεδομένα. Επιπλέον, το επίπεδο δεδομένων χρησιμεύει ως interface για απρόσκοπτη αλληλεπίδραση μεταξύ των άλλων επιπέδων της εφαρμογής και των δεδομένων. Λειτουργεί ως abstraction layer, προστατεύοντας τα δεδομένα από την άμεση αλληλεπίδραση με την υπόλοιπη εφαρμογή.

Domain Layer

Τα μοντέλα μας (οντότητες που περιέχουν δεδομένα) και τα Repository interfaces περιλαμβάνονται σε αυτό το επίπεδο. Ταυτόχρονα, η επιχειρηματική λογική του έργου διατηρείται σε αυτό το επίπεδο μαζί με τα Use cases.

Το επίπεδο τομέα (domain layer) χρησιμεύει ως ο κόμβος για την επιχειρηματική λογική, που περιλαμβάνει εργασίες όπως η μετατροπή δεδομένων, το φιλτράρισμα, η συγχώνευση και η ταξινόμηση. Σκοπός του είναι να μετατρέψει τα ακατέργαστα δεδομένα που λαμβάνονται από το επίπεδο δεδομένων σε μια επεξεργασμένη μορφή που είναι βολική και διαχειρίσιμη για το επίπεδο παρουσίασης (presentation layer).

Presentation Layer

Το επίπεδο παρουσίασης χρησιμεύει ως η διεπαφή μεταξύ του χρήστη και της εφαρμογής, διαχειρίζεται τα δεδομένα εισαγωγής χρήστη και εμφανίζει τα αντίστοιχα αποτελέσματα. Αυτό το επίπεδο συνήθως αποτελείται από μια γραφική διεπαφή χρήστη, όπως μια ιστοσελίδα, μια εφαρμογή για κινητά ή μια εφαρμογή για υπολογιστές. Στην αρχιτεκτονική MVVM, το επίπεδο παρουσίασης περιλαμβάνει αντικείμενα view και view model. Τα Views (Activities, Fragments) θα πρέπει να προσπαθούν να παραμείνουν απλά και χωρίς επιχειρηματική λογική. Εάν υπάρχει κάποια επιχειρηματική λογική εντός των views, είναι απαραίτητο να η υλοποίηση να αλλάξει.

4.2 Αρχιτεκτονική Spring Boot

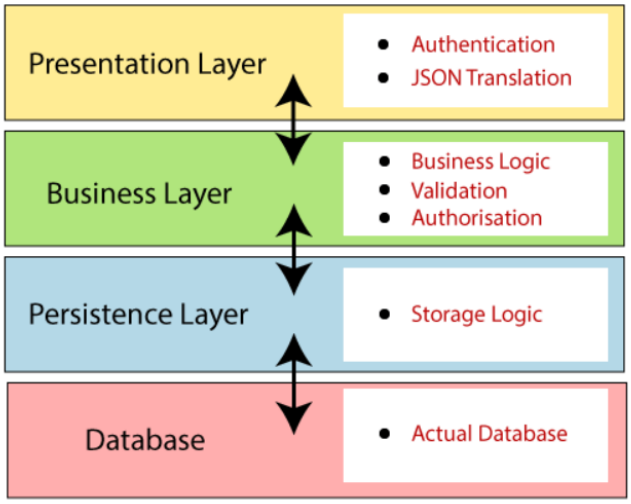
Το Spring Boot είναι μια ενότητα του Spring Framework. Χρησιμοποιείται για τη δημιουργία αυτόνομων εφαρμογών που βασίζονται σε ελατήρια ποιότητας παραγωγής με ελάχιστη προσπάθεια. Αναπτύχθηκε πάνω από το βασικό Spring Framework.

4.2.1 Επίπεδα (Layers) της αρχιτεκτονικής Spring Boot

Το Spring Boot ακολουθεί μια πολυεπίπεδη αρχιτεκτονική στην οποία κάθε επίπεδο επικοινωνεί με το επίπεδο ακριβώς κάτω ή πάνω από αυτό (ιεραρχική δομή).

Προτού κατανοήσουμε την Αρχιτεκτονική Spring Boot, πρέπει να γνωρίζουμε τα διαφορετικά επίπεδα και τις κατηγορίες που υπάρχουν σε αυτήν. Υπάρχουν τέσσερα επίπεδα στο Spring Boot είναι τα εξής:

1. Επίπεδο παρουσίασης (Presentation Layer)
2. Επιχειρηματικό Επίπεδο (Business Layer)
3. Επίπεδο επιμονής (Persistence Layer)
4. Επίπεδο βάσης δεδομένων (Database Layer)



Presentation Layer

Το επίπεδο παρουσίασης χειρίζεται τα αιτήματα HTTP, μεταφράζει την παράμετρο JSON σε αντικείμενο και πιστοποιεί την ταυτότητα του αιτήματος και το μεταφέρει στο επιχειρηματικό επίπεδο. Εν ολίγοις, αποτελείται από προβολές, δηλ. τμήμα frontend.

Business Layer

Το επιχειρηματικό επίπεδο χειρίζεται όλη την επιχειρηματική λογική. Αποτελείται από κλάσεις υπηρεσιών και χρησιμοποιεί υπηρεσίες που παρέχονται από επίπεδα πρόσβασης δεδομένων. Εκτελεί επίσης εξουσιοδότηση και επικύρωση.

Persistence Layer

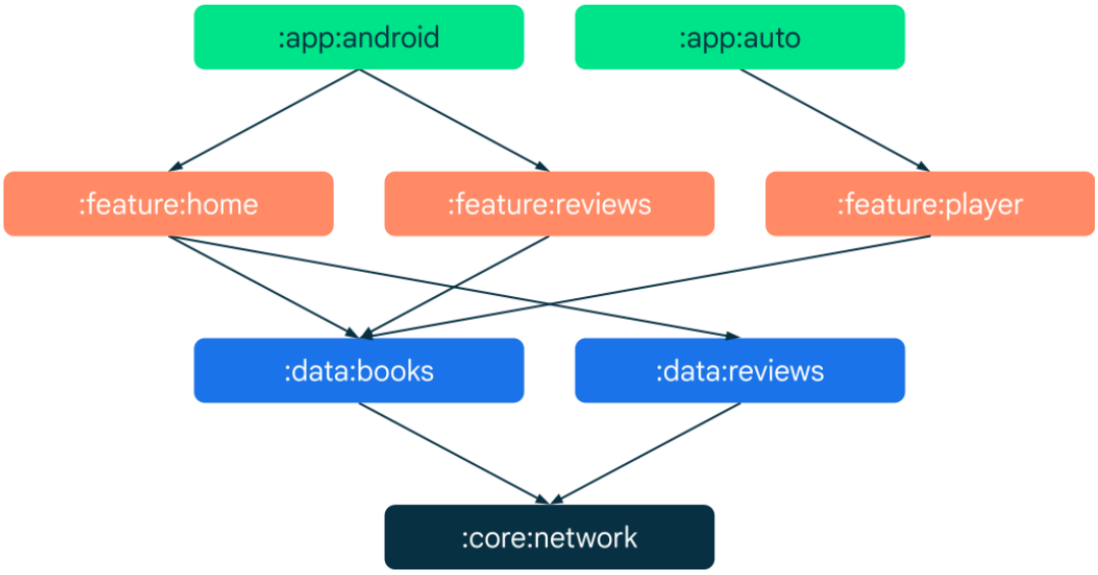
Το επίπεδο επιμονής περιέχει όλη τη λογική αποθήκευσης και μεταφράζει επιχειρηματικά αντικείμενα από και σε σειρές βάσης δεδομένων.

Database Layer

Στο επίπεδο βάσης δεδομένων, εκτελούνται λειτουργίες CRUD (δημιουργία, ανάκτηση, ενημέρωση, διαγραφή).

4.3 Modularization

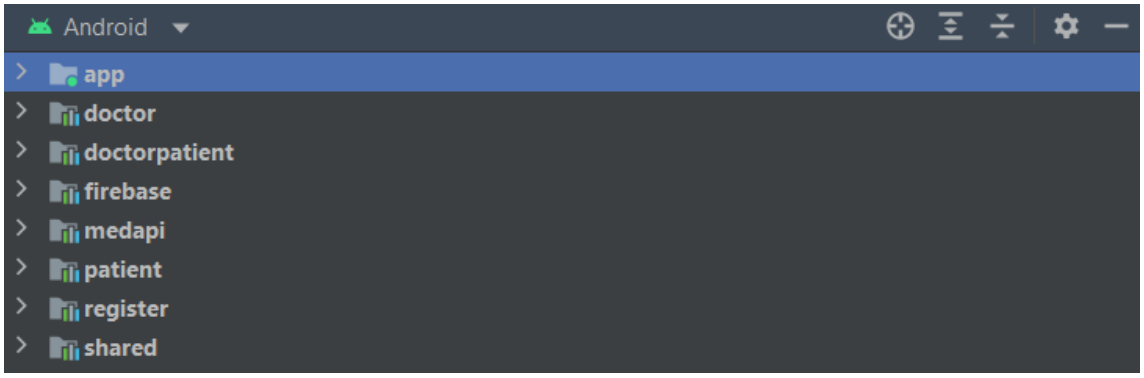
Είναι ο τρόπος με τον οποίο μπορείς να οργανώσεις το codebase σε μικρότερα κομμάτια ανεξάρτητα μεταξύ τους. Το κάθε κομμάτι ονομάζεται module. Το καθένα από αυτά εξυπηρετεί ένα δικό του σκοπό. Είναι ένας τρόπος να παίρνεις ένα πρόβλημα και να το αναλύει σε μικρότερα κομμάτια, με αποτέλεσμα να μειώνεται η πολυπλοκότητα σχεδίασης μία εφαρμογής και η αποτροπή από τη δημιουργία ενός μονολιθικού project.



Στα πλαίσια της μεταπτυχιακής διατριβής χρησιμοποιήθηκε η τεχνική του modularization, προκειμένου ο κώδικας της εφαρμογής να σπάσει σε κομμάτια, ώστε να είναι πιο εύκολα διαχειρίσιμος. Πιο συγκεκριμένα, ο κώδικας έχει χωριστεί με τον εξής τρόπο στα ακόλουθα modules:

1. app (περιέχει τη λογική του login)
2. doctor (περιέχει τη λογική – οθόνες που είναι αποκλειστικές για το γιατρό)
3. doctorpatient (περιέχει κοινή λογική για γιατρό και ασθενή)
4. patient (περιέχει τη λογική – οθόνες που είναι αποκλειστικές για το ασθενή)
5. firebase (περιέχει τη λογική σχετικά με τα request στη Firebase)
6. medapi (περιέχει τη λογική σχετικά με τα request στο τοπικό server)
7. register (περιέχει τη λογική με την εγγραφή λογαριασμού χρήστη)

8. shared (περιέχει λογική που διαμοιράζεται στα παραπάνω modules)



4.4 Βάσεις Δεδομένων

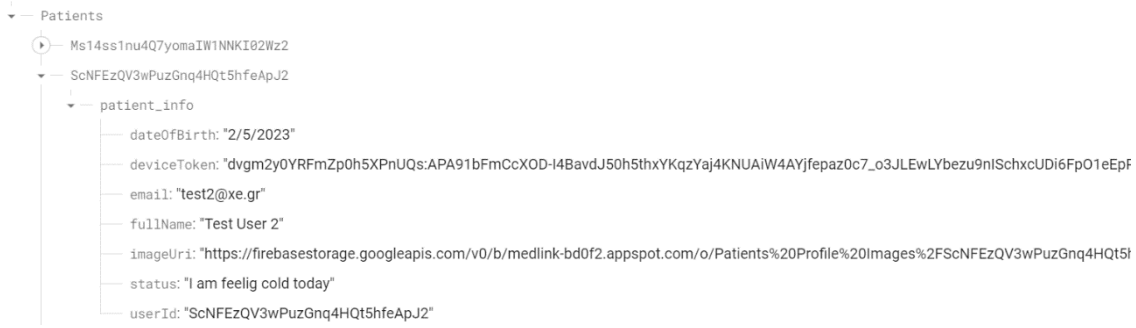
Για την υλοποίηση του τρέχοντος project απαιτούνται δύο βάσεις δεδομένων: μία μη σχεσιακή ΒΔ και μία σχεσιακή.

4.4.1 Μη σχεσιακή ΒΔ

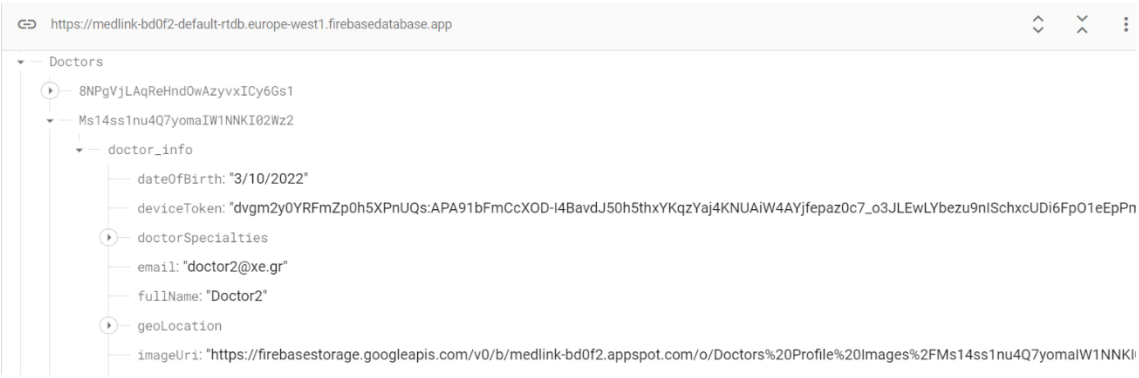
Εκμεταλλευόμενοι το λογισμικό της Firebase και πιο συγκεκριμένα αυτό της Realtime Database, έχουμε τη δυνατότητα να δημιουργήσουμε μία μη σχεσιακή ΒΔ. Όπως έχουμε ήδη αναφέρει, η πληροφορία που αποθηκεύεται στη συγκεκριμένη βάση, είναι τέτοια έτσι ώστε να μπορέσει στο τέλος να υλοποιηθεί η ανταλλαγή μηνυμάτων μεταξύ γιατρού και ασθενή. Σε αυτήν, ο τρόπος με τον οποίο αποθηκεύονται τα δεδομένα είναι ο εξής:

Αρχικά έχουμε κάποιου βασικούς κόμβους (πατέρες), οι οποίοι είναι υπεύθυνοι να αποθηκεύουν τις πληροφορίες για τους Ασθενείς και τους Γιατρούς. Πιο συγκεκριμένα:

Ένας **Ασθενής** αποθηκεύεται ως κόμβος παιδί κάτω από τον κόμβο πατέρα με όνομα «Patients» με κάποιο child id. Αυτό με τη σειρά του έχει ένα κόμβο παιδί με όνομα «patient_info», το οποίο περιέχει τις βασικές πληροφορίες για τα account settings του ασθενή, δηλαδή: dateOfBirth, deviceToken, email, fullName, imageUri (url στο Firebase Storage όπου αποθηκεύονται οι φωτογραφίες προφίλ χρήστη), userId.

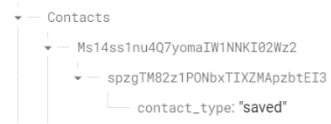


Ένας **Γιατρός** αποθηκεύεται ως κόμβος παιδί κάτω από τον κόμβο πατέρα με όνομα «Doctors» με κάποιο child id. Αυτό με τη σειρά του έχει ένα κόμβο παιδί με όνομα «doctor_info», το οποίο περιέχει τις βασικές πληροφορίες για τα account settings του γιατρού, δηλαδή: dateOfBirth, deviceToken, doctorSpecialties, email, fullName, geolocation, imageUri (url στο Firebase Storage όπου αποθηκεύονται οι φωτογραφίες προφίλ χρήστη), userId.

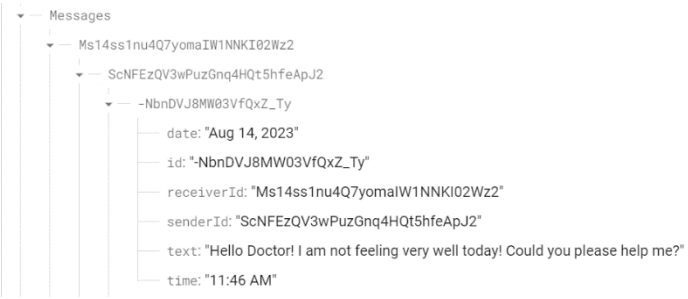


Στη συνέχεια έχουμε τα **Αιτήματα Υγείας** (Heal Requests), τα οποία λειτουργούν ως εξής: Κάθε φορά που κάνει ένα αίτημα ένας ασθενής σε ένα γιατρό, δημιουργούνται δύο κόμβοι παιδιά, κάτω από τον κόμβο πατέρα με όνομα «Heal Requests». Το πρώτο περιέχει το id του γιατρού και κάτω από αυτό έχει ως παιδί το id του ασθενή με ένα κλειδί request_type, ενώ το δεύτερο παιδί έχει το αντίστροφο. Ανάλογα με την τιμή του request_type (sent/ received), γίνεται αντιληπτό για το ποιος έχει κάνει το αίτημα και ποιος το έχει δεχθεί. Εφόσον στα πλαίσια της εφαρμογής μόνο ο ασθενής μπορεί να κάνει request σε γιατρό, το request_type του ασθενή θα είναι πάντα sent και του γιατρού θα είναι πάντα received.

Στη συνέχεια έχουμε τις **Επαφές** (Contacts), οι οποίες λειτουργούν ως εξής: Κάθε φορά που κάνει ένα αίτημα ένας ασθενής σε ένα γιατρό και γίνεται αποδεκτό, τότε διαγράφονται οι ανάλογοι κόμβοι από το Heal Requests και δημιουργούνται οι αντίστοιχοι δύο κόμβοι κάτω από τον κόμβο πατέρα «Contacts». Ο πρώτος περιέχει το id του γιατρού και κάτω από αυτό έχει ως παιδί το id του ασθενή με ένα κλειδί contact_type, ενώ ο δεύτερος έχει το αντίστροφο.



Στο τέλος έχουμε τα **Μηνύματα** (Messages), τα οποία λειτουργούν ως εξής: Κάθε φορά που ανταλλάσσεται κάποιο μήνυμα μεταξύ γιατρού και ασθενή, τότε δημιουργούνται οι αντίστοιχοι δύο κόμβοι κάτω από τον κόμβο πατέρα «Messages». Ο πρώτος περιέχει το id του γιατρού και κάτω από αυτό έχει ως παιδί το id του ασθενή με ένα κλειδί το οποίο περιέχει ως τιμή ένα μήνυμα με τιμές για: data, id μηνύματος, receiverId, senderId, text, time. Ο δεύτερος περιέχει ανάλογες πληροφορίες, μόνο που ο κόμβος του γιατρού έχει ως παιδί το κόμβο του ασθενή.



4.4.2 Σχεσιακή ΒΔ

Εκμεταλλευόμενοι το λογισμικό του XAMPP, δημιουργούμε μία σχεσιακή ΒΔ, η οποία αποσκοπεί στην αποθήκευση δεδομένων ασθενή και γιατρού. Πιο συγκεκριμένα:

Έχουμε τον **Πίνακα patient**, ο οποίος περιέχει τις εξής πληροφορίες: patient_id (primary key), email, fullName. Ο συγκεκριμένος πίνακας **έχει σχέση 1 προς 1** με τον **Πίνακα patient_details**, ο οποίος περιέχει τις εξής πληροφορίες: id, fk_patient_id (ξένο κλειδί στο πρωτεύον κλειδί του Πίνακα patient), address, birth_date, phone, email_of_contact, personal_info.

Έχουμε τον **Πίνακα doctor**, ο οποίος περιέχει τις εξής πληροφορίες: doctor_id (primary key), email, fullName. Ο συγκεκριμένος πίνακας **έχει σχέση 1 προς 1** με τον **Πίνακα doctor_details**, ο οποίος περιέχει τις εξής πληροφορίες: id, fk_doctor_id (ξένο κλειδί στο πρωτεύον κλειδί του Πίνακα doctor), description, birth_date, phone, email_of_contact.

Στη συνέχεια παραθέτουμε τον τρόπο με τον οποίο εισάγονται εγγραφές στους συγκεκριμένους πίνακες χρησιμοποιώντας το λογισμικό Postman. Πιο συγκεκριμένα:

Θέτουμε σε λειτουργία το τοπικό service και στη συνέχεια «χτυπάμε» τα ανάλογα endpoints για την εισαγωγή εγγραφών στον πίνακα της σχεσιακής ΒΔ. Η παρακάτω εικόνα μας δείχνει τα διάφορα endpoints που υποστηρίζονται σχετικά με τις λειτουργικότητες του ασθενή. Η εικόνα αποτελεί στιγμιότυπο του αρχείου PatientRestController. Οι λειτουργικότητες που υποστηρίζονται είναι:

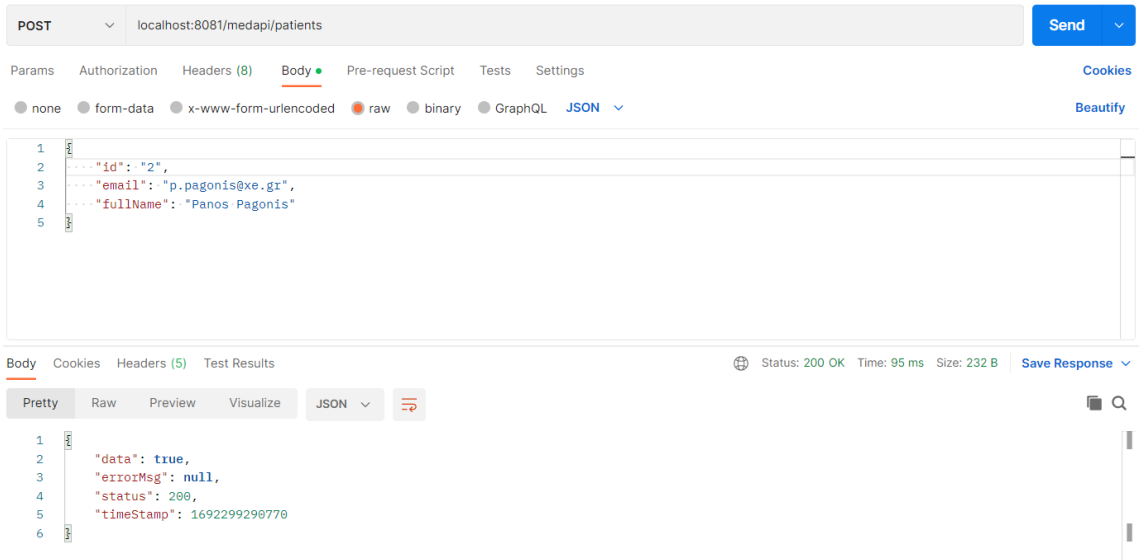
1. savePatient (POST HTTP Request) – αποθήκευση ασθενούς
2. createPatientDetails (POST HTTP Request) - δημιουργία πληροφοριών ασθενούς
3. getPatientDetails (GET HTTP Request) – προβολή πληροφοριών ασθενούς
4. updatePatientDetails (PUT HTTP Request) – αλλαγή πληροφοριών ασθενούς

Το αρχείο ακολουθεί πιστά την αρχιτεκτονική Spring Boot, το οποίο σημαίνει ότι ο Rest Controller με τη σειρά του επικοινωνεί με ένα Service, το οποίο με τη σειρά του επικοινωνεί με ένα Repository, το οποίο επικοινωνεί με τη σχεσιακή ΒΔ ορίζοντας κάποια μοντέλα.

```java
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/medapi")
public class PatientRestController {

    @Autowired
    private final PatientService patientService;

    public PatientRestController(PatientService patientService) { this.patientService = patientService; }

    @PostMapping("/patients")
    public MedApiResponse<Boolean> savePatient(@RequestBody Patient patient) {
        return patientService.savePatient(patient);
    }

    @GetMapping("/patients/{patientId}/details")
    public MedApiResponse<PatientDetailsDTO> getPatientDetails(@PathVariable String patientId) {
        return patientService.getPatientDetails(patientId);
    }

    @PostMapping("/patients/{patientId}/details")
    public MedApiResponse<Boolean> createPatientDetails(@PathVariable String patientId, @RequestBody PatientDetails patientDetails) {
        return patientService.createPatientDetails(patientId, patientDetails);
    }

    @PutMapping("/patients/{patientId}/details")
    public MedApiResponse<Boolean> updatePatientDetails(@PathVariable String patientId, @RequestBody PatientDetails patientDetails) {
        return patientService.updatePatientDetails(patientId, patientDetails);
    }
}
```

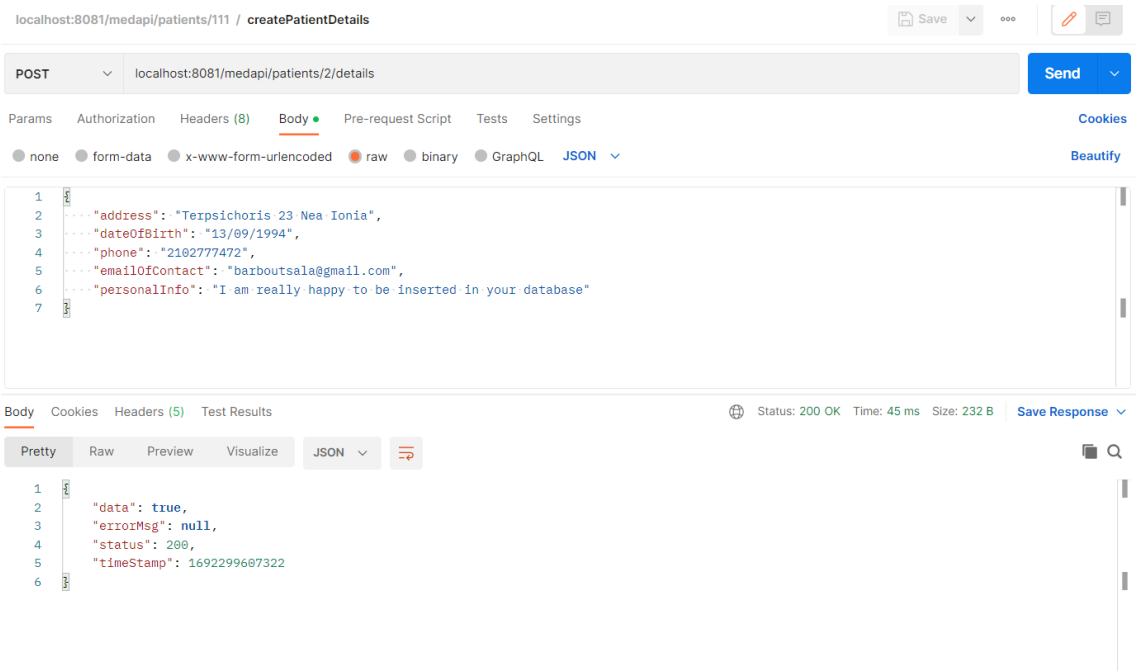
1. savePatient (POST HTTP Request) – αποθήκευση ασθενούς

Για την αποθήκευση ασθενούς στη σχεσιακή ΒΔ, χρησιμοποιούμε το λογισμικό Postman. Πιο συγκεκριμένα χτυπάμε το endpoint: **localhost:8081/medapi/patient** και περνάμε στο σώμα του request, ένα κατάλληλο JSON object. Η παρακάτω εικόνα μας δείχνει μία επιτυχημένη εγγραφή χρήστη, το οποίο επιστρέφει ένα response τύπου JSON object.



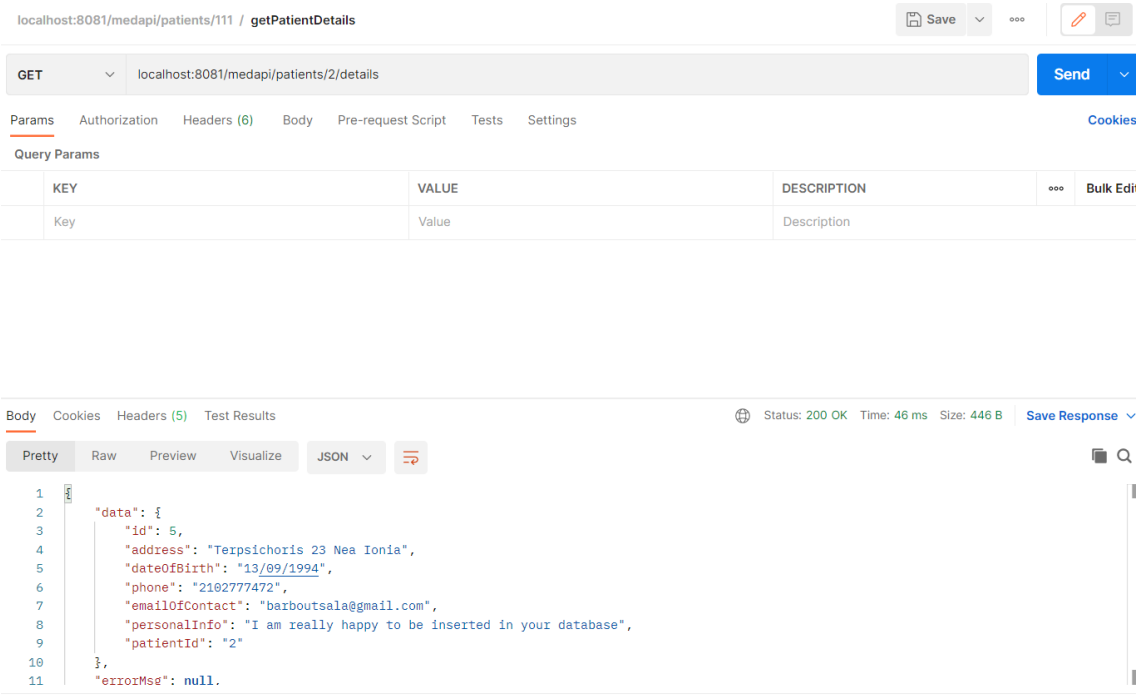
2. createPatientDetails (POST HTTP Request) - δημιουργία πληροφοριών ασθενούς

Για την αποθήκευση πληροφοριών ασθενούς στη σχεσιακή ΒΔ, χρησιμοποιούμε το λογισμικό Postman. Πιο συγκεκριμένα χτυπάμε το endpoint: **localhost:8081/medapi/patients/2/details** (2 είναι το id του ασθενούς που αποθηκεύτηκε προηγουμένως) και περνάμε στο σώμα του request, ένα κατάλληλο JSON object. Η παρακάτω εικόνα μας δείχνει μία επιτυχημένη εγγραφή πληροφοριών χρήστη, το οποίο επιστρέφει ένα response τύπου JSON object.



3. getPatientDetails (GET HTTP Request) – προβολή πληροφοριών ασθενούς

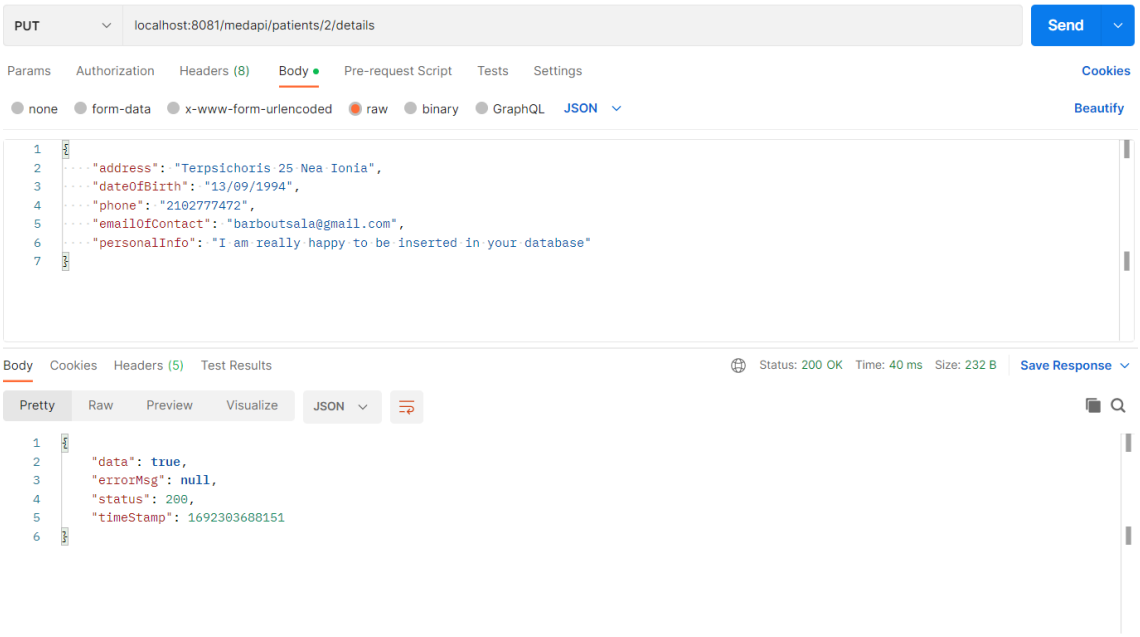
Για την προβολή πληροφοριών ασθενούς στη σχεσιακή ΒΔ, χρησιμοποιούμε το λογισμικό Postman. Πιο συγκεκριμένα χτυπάμε το endpoint: **localhost:8081/medapi/patients/2/details** (2 είναι το id του ασθενούς που αποθηκεύτηκε προηγουμένως). Η παρακάτω εικόνα μας δείχνει μία επιτυχημένη προβολή πληροφοριών χρήστη, το οποίο επιστρέφει ένα response τύπου JSON object.



4. updatePatientDetails (PUT HTTP Request) – αλλαγή πληροφοριών ασθενούς

Για την αλλαγή πληροφοριών ασθενούς στη σχεσιακή ΒΔ, χρησιμοποιούμε το λογισμικό Postman. Πιο συγκεκριμένα χτυπάμε το endpoint: **localhost:8081/medapi/patients/2/details** (2 είναι το id του ασθενούς που αποθηκεύτηκε προηγουμένως) και περνάμε στο σώμα του request, ένα κατάλληλο JSON

object. Η παρακάτω εικόνα μας δείχνει μία επιτυχημένη προβολή πληροφοριών χρήστη, το οποίο επιστρέφει ένα response τύπου JSON object.



Παρόμοια διαδικασία ακολουθείται και για την εισαγωγή εγγραφών στους πίνακες γιατρών. Η παρακάτω εικόνα μας δείχνει τα διάφορα endpoints που υποστηρίζονται σχετικά με τις λειτουργικότητες του γιατρού. Η εικόνα αποτελεί στιγμιότυπο του αρχείου DoctorRestController. Οι λειτουργικότητες που υποστηρίζονται είναι:

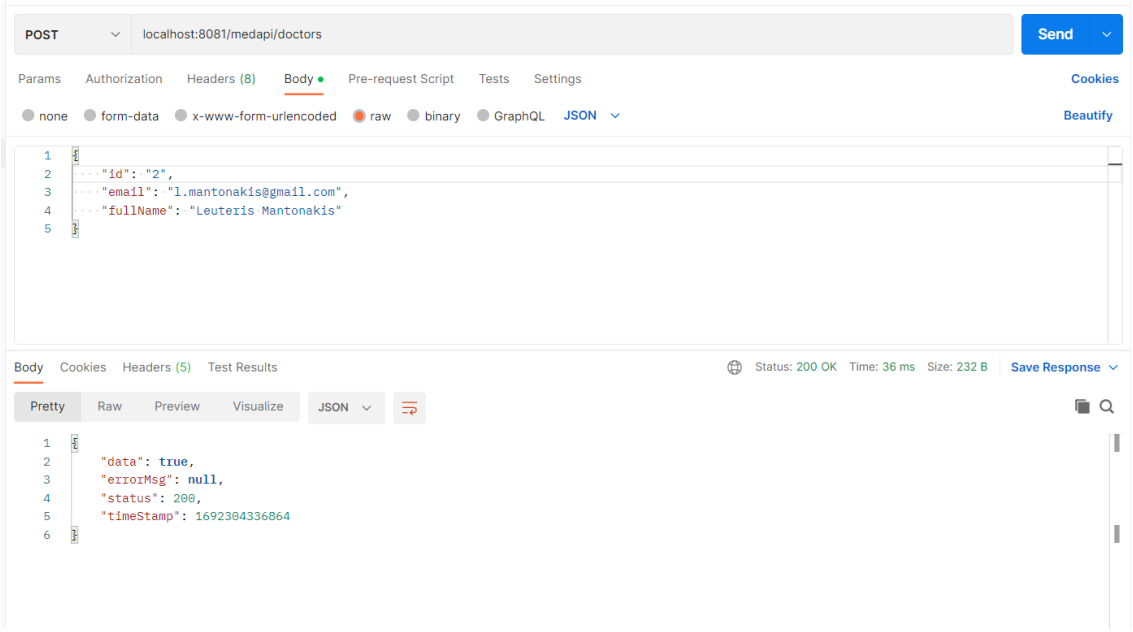
1. saveDoctor (POST HTTP Request) – αποθήκευση γιατρού
2. createDoctorDetails (POST HTTP Request) - δημιουργία πληροφοριών γιατρού
3. getDoctorDetails (GET HTTP Request) – προβολή πληροφοριών γιατρού
4. updateDoctorDetails (PUT HTTP Request) – αλλαγή πληροφοριών γιατρού

Το αρχείο ακολουθεί πιστά την αρχιτεκτονική Spring Boot, το οποίο σημαίνει ότι ο Rest Controller με τη σειρά του επικοινωνεί με ένα Service, το οποίο με τη σειρά του επικοινωνεί με ένα Repository, το οποίο επικοινωνεί με τη σχεσιακή ΒΔ ορίζοντας κάποια μοντέλα.

```java
DoctorRestController.java
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/medapi")
public class DoctorRestController {

    @Autowired
    private final DoctorService doctorService;

    public DoctorRestController(DoctorService doctorService) {
        this.doctorService = doctorService;
    }

    @PostMapping("/doctors")
    public MedApiResponse<Boolean> saveDoctor(@RequestBody Doctor doctor) { return doctorService.saveDoctor(doctor); }

    @GetMapping("/doctors/{doctorId}/details")
    public MedApiResponse<DoctorDetailsDTO> getDoctorDetails(@PathVariable String doctorId) {
        return doctorService.getDoctorDetails(doctorId);
    }

    @PostMapping("/doctors/{doctorId}/details")
    public MedApiResponse<Boolean> createDoctorDetails(@PathVariable String doctorId, @RequestBody DoctorDetails doctorDetails) {
        return doctorService.createDoctorDetails(doctorId, doctorDetails);
    }

    @PutMapping("/doctors/{doctorId}/details")
    public MedApiResponse<Boolean> updateDoctorDetails(@PathVariable String doctorId, @RequestBody DoctorDetails doctorDetails) {
        return doctorService.updateDoctorDetails(doctorId, doctorDetails);
    }
}
```

1. saveDoctor (POST HTTP Request) – αποθήκευση γιατρού

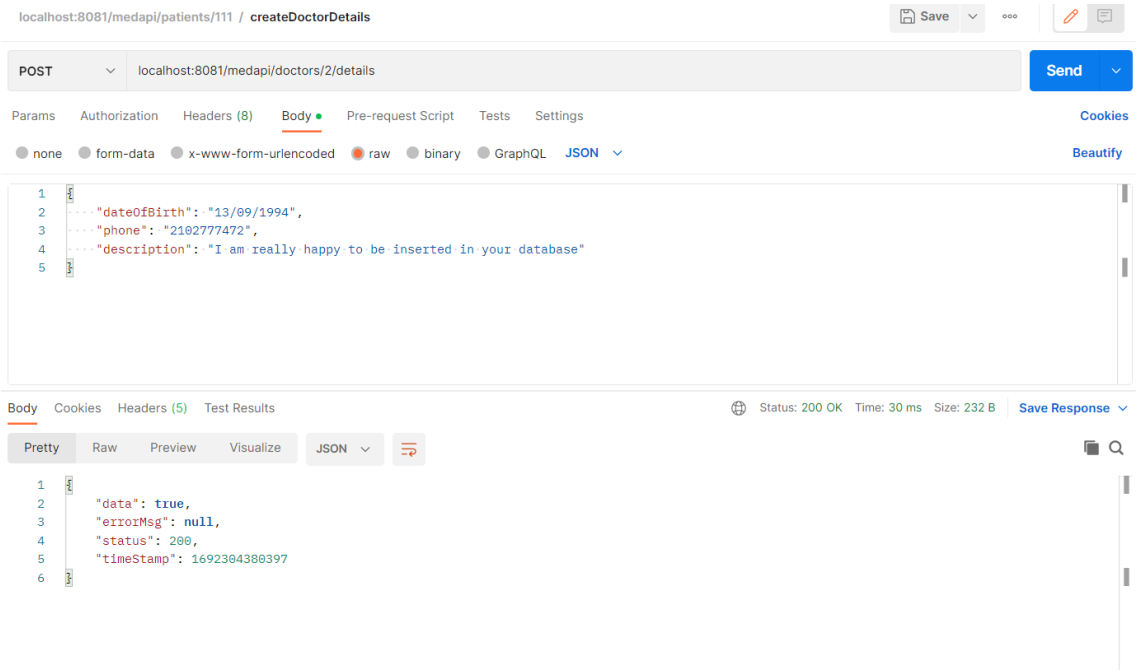
Για την αποθήκευση γιατρού στη σχεσιακή ΒΔ, χρησιμοποιούμε το λογισμικό Postman. Πιο συγκεκριμένα χτυπάμε το endpoint: **localhost:8081/medapi/doctor** και περνάμε στο σώμα του request, ένα κατάλληλο JSON object. Η παρακάτω εικόνα μας δείχνει μία επιτυχημένη εγγραφή χρήστη, το οποίο επιστρέφει ένα response τύπου JSON object.



2. createDoctorDetails (POST HTTP Request) - δημιουργία πληροφοριών γιατρού

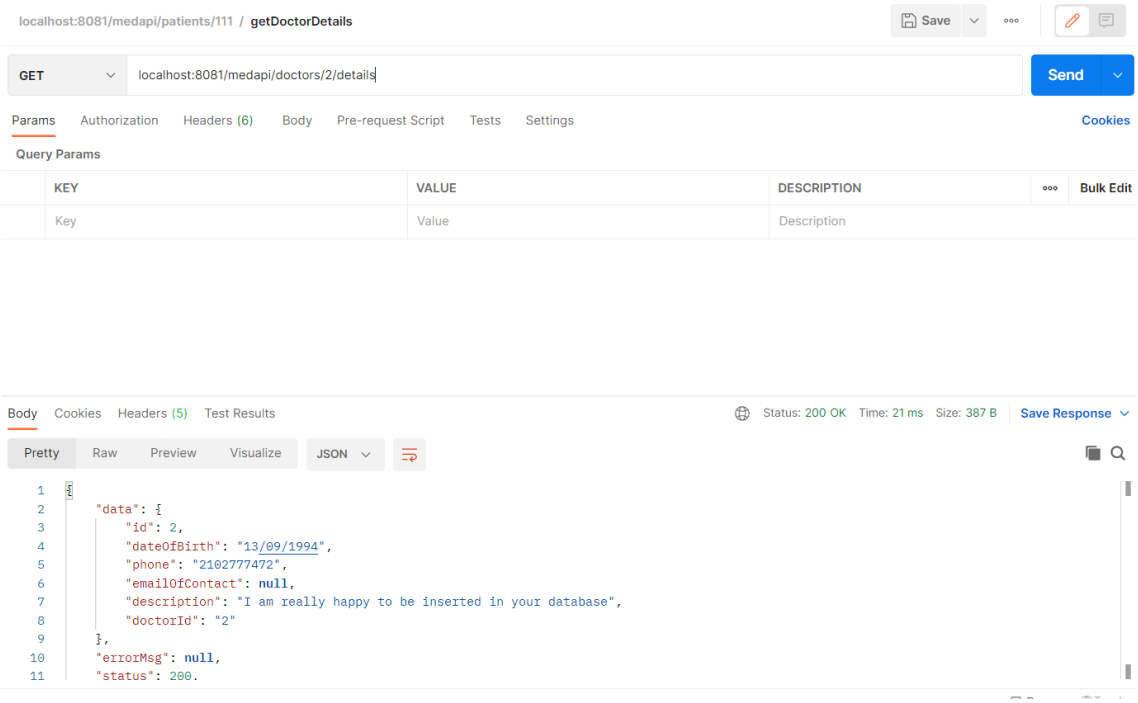
Για την αποθήκευση πληροφοριών γιατρού στη σχεσιακή ΒΔ, χρησιμοποιούμε το λογισμικό Postman. Πιο συγκεκριμένα χτυπάμε το endpoint: **localhost:8081/medapi/doctors/2/details** (2 είναι το id του γιατρού που αποθηκεύτηκε προηγουμένως) και περνάμε στο σώμα του request, ένα κατάλληλο JSON

object. Η παρακάτω εικόνα μας δείχνει μία επιτυχημένη εγγραφή πληροφοριών χρήστη, το οποίο επιστρέφει ένα response τύπου JSON object.



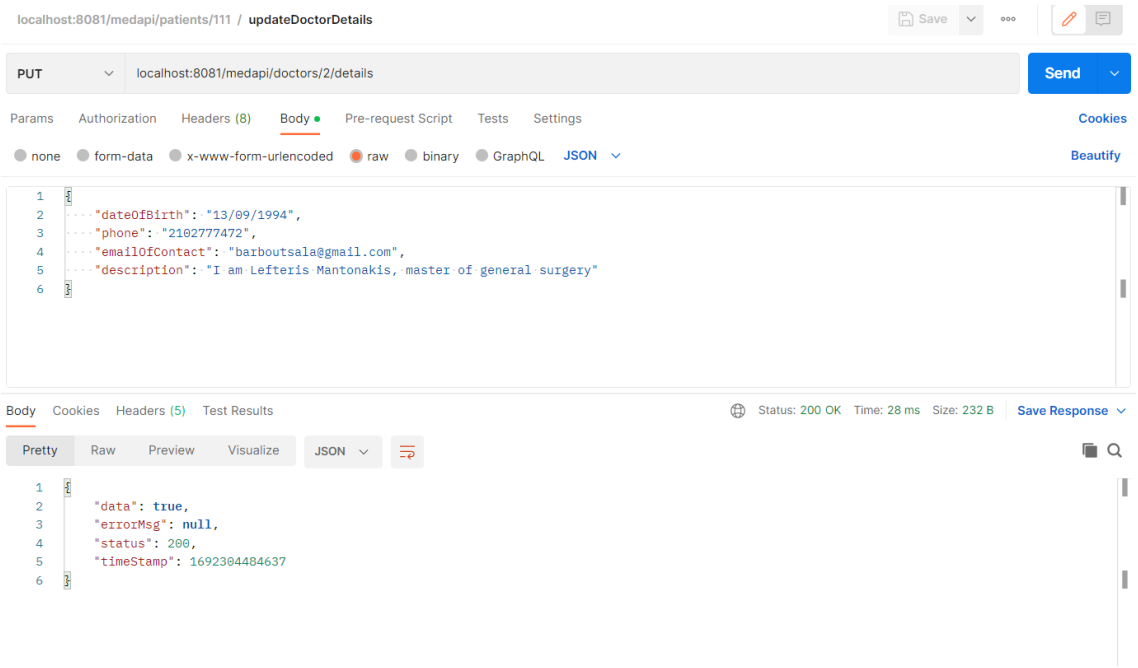
3. getDoctorDetails (GET HTTP Request) – προβολή πληροφοριών γιατρού

Για την προβολή πληροφοριών γιατρού στη σχεσιακή ΒΔ, χρησιμοποιούμε το λογισμικό Postman. Πιο συγκεκριμένα χτυπάμε το endpoint: **localhost:8081/medapi/doctors/2/details** (2 είναι το id του γιατρού που αποθηκεύτηκε προηγουμένως). Η παρακάτω εικόνα μας δείχνει μία επιτυχημένη προβολή πληροφοριών χρήστη, το οποίο επιστρέφει ένα response τύπου JSON object.



4. updateDoctorDetails (PUT HTTP Request) – αλλαγή πληροφοριών γιατρού

Για την αλλαγή πληροφοριών γιατρού στη σχεσιακή ΒΔ, χρησιμοποιούμε το λογισμικό Postman. Πιο συγκεκριμένα χτυπάμε το endpoint: **localhost:8081/medapi/doctors/2/details** (2 είναι το id του γιατρού που αποθηκεύτηκε προηγουμένως) και περνάμε στο σώμα του request, ένα κατάλληλο JSON object. Η παρακάτω εικόνα μας δείχνει μία επιτυχημένη προβολή πληροφοριών χρήστη, το οποίο επιστρέφει ένα response τύπου JSON object.

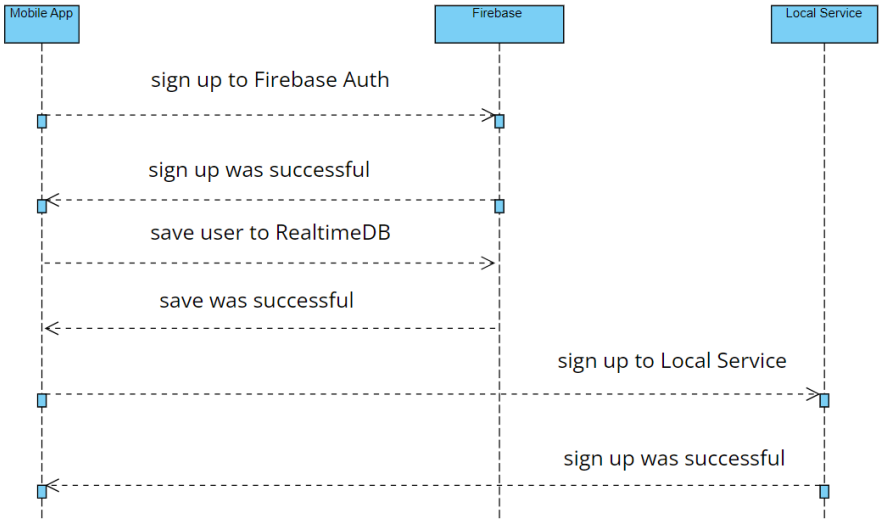


4.5 Ανάλυση τριών βασικών workflows της εφαρμογής (Registration, Login, Search Doctor)

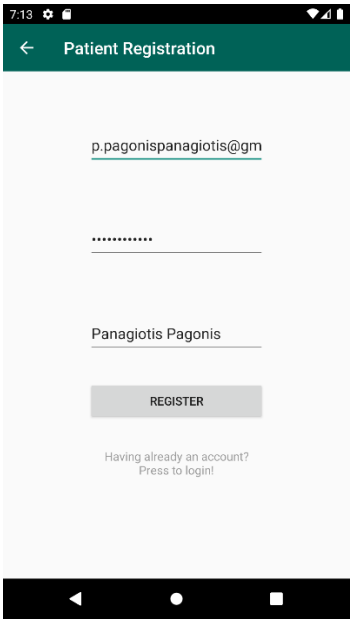
Έχοντας αναλύσει τα βασικά δομικά στοιχεία της αρχιτεκτονικής που επιλέχθηκε στα πλαίσια της παρούσης μεταπτυχιακής διατριβής, στη συνέχεια αναλύονται τρία βασικά workflows της εφαρμογής.

4.5.1 Εγγραφή Χρήστη

Ως πρώτο workflow περιγράφεται αυτό της δημιουργίας λογαριασμού χρήστη μέσω της εφαρμογής. Πιο συγκεκριμένα ο χρήστης (στην προκειμένη περίπτωση ένας Ασθενής), έχει πληκτρολογήσει τα απαραίτητα στοιχεία στα κατάλληλα παιδία και επιλέγει το κουμπί «Register». Αυτό που θα συμβεί, είναι πως σε πρώτη φάση, η εφαρμογή θα προσπαθήσει να κάνει register στο σύστημα Firebase Authentication. Το Firebase Authentication αποτελεί ένα σύστημα, στο οποίο μπορούν να αποθηκευτούν email χρηστών χρησιμοποιώντας κάποιον πάροχο (provider), όπως Google, Facebook, κλπ. Στην προκειμένη περίπτωση επιλέγουμε ως τρόπο το Email And Password (δεν χρησιμοποιείται κάποιος provider). Αφότου ολοκληρωθεί επιτυχώς η σύνδεση του χρήστη στο σύστημα, ξεκινά η δεύτερη φάση με ένα request στη βάση που περιέχεται στο σύστημα Firebase Realtime Database. Εφόσον η αποθήκευση του χρήστη ολοκληρωθεί με επιτυχία, φθάνουμε στην τρίτη και τελευταία φάση, όπου ένα request στέλνεται από την εφαρμογή προς το τοπικό service. Όταν η αποθήκευση του χρήστη πραγματοποιηθεί στη ΒΔ με την οποία επικοινωνεί το τοπικό service, τότε έχουμε μία επιτυχημένη εγγραφή χρήστη στη βάση. Όλα τα παραπάνω περιγράφονται με το ακόλουθο **Sequence Diagram**:



Σε επίπεδο κώδικα, οι παραπάνω ενέργειες μεταφράζονται ως εξής: Αρχικά όλη η πληροφορία που αναγράφεται στην οθόνη του χρήστη (email, password, fullName) είναι αποθηκευμένη σε ένα ViewModel.



Όταν ο χρήστης πατήσει το κουμπί «Register», ξεκινά η εκτέλεση ενός request με όνομα **registerUser().** Ο ακόλουθος κώδικα βρίσκεται μέσα στο αρχείο **PatientRegisterActivity.**

```
private fun setupRegisterButtonListener() = ui.run {  this: ActivityPatientRegisterBinding
    registerBtn.let {  it: Button
        registerBtn.setOnClickListener {  it: View!
            viewModel.registerUser()
        }
    }
}
```

Πατώντας επομένως το κουμπί Register, εκτελείται η συνάρτηση **registerUser()** του ViewModel, η οποία εκτελεί δύο ενέργειες: αποθηκεύει τον χρήστη στη Firebase και αποθηκεύει τον χρήστη στο τοπικό server. Αυτό φαίνεται και στην παρακάτω εικόνα του αρχείου **RegisterViewModel**. Για την ολοκλήρωση των διαδικασιών που μόλις περιγράφτηκαν, χρησιμοποιούνται τα κατάλληλα useCases (**firebaseRegisterUseCase**, **medApiUseCase**).

```
override fun registerUser() {
    viewModelScope.launch { this: CoroutineScope
        registerInputsStateValue = verifyInputs()
        if (registerInputsStateValue == VALID_INPUTS) {
            try {
                val firebaseSavedUser = firebaseRegisterUseCase.saveUserInFirebase(
                    email = emailValue ?: "",
                    password = pwdValue ?: "",
                    fullName = fullNameValue ?: "",
                )

                saveUserInMedApi(user = firebaseSavedUser)

                registerUserValue = Status.success(
                    RegisterResult(
                        id = firebaseSavedUser?.userId,
                        deviceToken = firebaseSavedUser?.deviceToken,
                    )
                )
            } catch (exception: Throwable) {
                when {
                    exception.isFirebaseError -> deleteUserFromFirebaseAuth()
                    else -> deleteUserFromFirebaseAuthAndDB()
                }
                registerUserValue = Status.failure(exception)
            } finally {
                registerInputsStateValue = DEFAULT_REGISTER_INPUT_STATE
            }
        }
    }
```

Το αποτέλεσμα της όλης διαδικασία αποθηκεύεται σε ένα live data με όνομα registerUserValue. Το αποτέλεσμα αυτού ανιχνεύεται (observe) από το UI, προκειμένου να εκτελεστεί το κατάλληλο action.

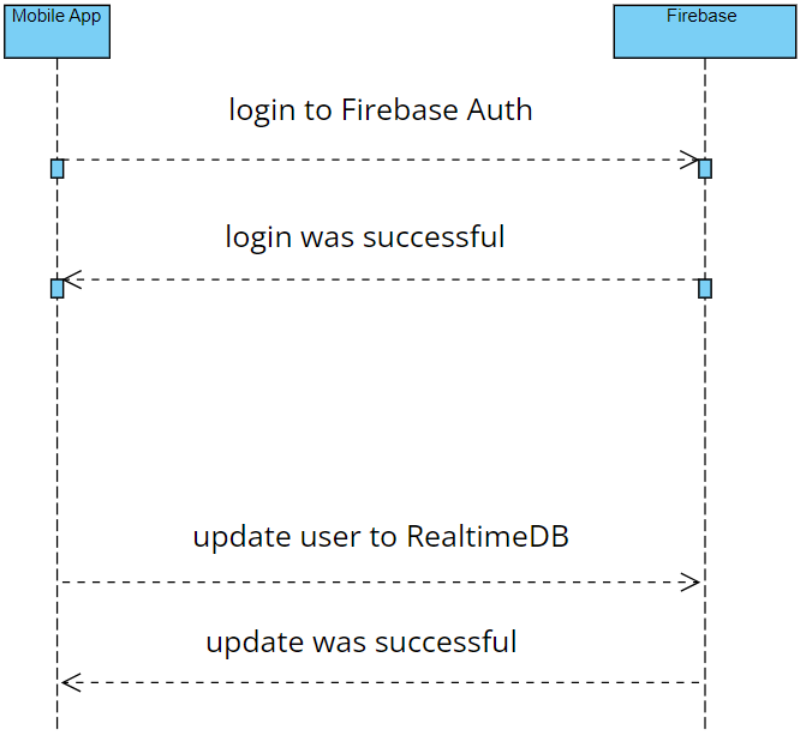
```
override val registerInputsState: LiveData<RegisterInputState?>
    get() = savedState.getLiveData(
        REGISTER_INPUT_STATE,
        DEFAULT_REGISTER_INPUT_STATE
    )
```

Στην περίπτωση της εφαρμογής, αν η διαδικασία είναι επιτυχής, τότε επιστρέφουμε ένα θετικό αποτέλεσμα (**RESULT_OK**) που θα οδηγήσει στο να ανακατευθυνθεί ο χρήστης στη κύρια οθόνη της εφαρμογής. Σε διαφορετική περίπτωση εμφανίζεται ένα toast message με το κατάλληλο μήνυμα λάθους.

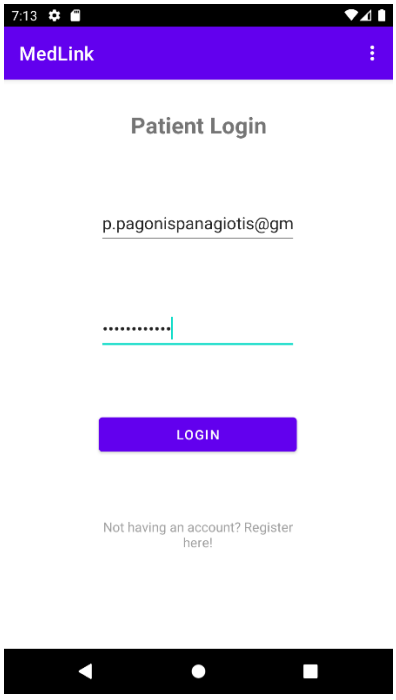
```
observe(viewModel.registerUser) { status ->
    status?.onSuccess { registerResult ->
        val userId = registerResult.id
        val deviceToken = registerResult.deviceToken
        if (userId?.isNotEmpty() == true && deviceToken?.isNotEmpty() == true) {
            setUserSharedPreferences(
                id = userId,
                deviceToken = deviceToken,
            )
            showToast(messageId = "Account created successfully")
            returnWithResult(RESULT_OK)
        } else {
            showToast(messageId = "Oops, something went wrong! Please try again!")
            returnWithResult(RESULT_CANCELED)
        }
    }?.onFailure {  it: Throwable
        val errorMsg = it.message
        if (it.isFirebaseError && !errorMsg.isNullOrEmpty())
            showToast(message = errorMsg)
        else
            showToast(messageId = "Oops, something went wrong! Please try again!")
    }
}
```

4.5.2 Σύνδεση Χρήστη

Ως δεύτερο workflow περιγράφεται αυτό της σύνδεσης λογαριασμού χρήστη μέσω της εφαρμογής. Πιο συγκεκριμένα ο χρήστης (στην προκειμένη περίπτωση ένας Ασθενής), έχει πληκτρολογήσει τα απαραίτητα στοιχεία στα κατάλληλα παιδία και επιλέγει το κουμπί «Login». Αυτό που θα συμβεί, είναι πως σε πρώτη φάση, η εφαρμογή θα προσπαθήσει να κάνει sign in στο σύστημα Firebase Authentication. Το Firebase Authentication αποτελεί ένα σύστημα, στο οποίο μπορούν να αποθηκευτούν email χρηστών χρησιμοποιώντας κάποιον πάροχο (provider), όπως Google, Facebook, κλπ. Στην προκειμένη περίπτωση επιλέγουμε ως τρόπο το Email And Password (δεν χρησιμοποιείται κάποιος provider). Αφότου ολοκληρωθεί επιτυχώς η σύνδεση του χρήστη στο σύστημα, ξεκινά η δεύτερη φάση με ένα request στη βάση που περιέχεται στο σύστημα Firebase Realtime Database. Με αυτόν τον τρόπο ανανεώνονται κάποια στοιχεία του χρήστη στη ΒΔ, όπως το **deviceToken**. Όλα τα παραπάνω περιγράφονται με το ακόλουθο **Sequence Diagram**:



Σε επίπεδο κώδικα, οι παραπάνω ενέργειες μεταφράζονται ως εξής: Αρχικά όλη η πληροφορία που αναγράφεται στην οθόνη του χρήστη (email, password) είναι αποθηκευμένη σε ένα ViewModel.



Όταν ο χρήστης πατήσει το κουμπί «Login», ξεκινά η εκτέλεση ενός request με όνομα **loginWithEmailAndPwd().** Ο ακόλουθος κώδικα βρίσκεται μέσα στο αρχείο **LoginActivity.**

```
loginBtn.setOnClickListener {  it: View!
    viewModel.loginWithEmailAndPwd(
        email = emailTxt.text.toString(),
        password = pwdTxt.text.toString(),
    )
}
```

Πατώντας επομένως το κουμπί Login, εκτελείται η συνάρτηση **loginWithEmailAndPwd()** του ViewModel, η οποία εκτελεί δύο ενέργειες: συνδέει το χρήστη στο Firebase Authentication και ανανεώνει τα στοιχεία του χρήστη στη Firebase. Αυτό φαίνεται και στην παρακάτω εικόνα του αρχείου **LoginViewModel**. Για την ολοκλήρωση των διαδικασιών που μόλις περιγράφτηκαν, χρησιμοποιούνται τα κατάλληλα useCases (**loginUseCase**).

```
override fun loginWithEmailAndPwd(
    email: String,
    password: String,
) {
    viewModelScope.launch {  this: CoroutineScope
        loginInputsStateValue = validateLoginInputsState(
            email = email,
            password = password,
        )
        if (loginInputsStateValue == VALID_INPUTS) {
            loginResultValue = try {
                val result = loginUseCase.loginWithEmailAndPwd(
                    email = email,
                    password = password,
                )
                Status.success(result)
            } catch (exception: Throwable) {
                Status.failure(exception)
            } finally {
                loginInputsStateValue = DEFAULT_LOGIN_INPUTS_STATE
            }
        }
    }
}
```

Το αποτέλεσμα της όλης διαδικασία αποθηκεύεται σε ένα live data με όνομα **loginResult**. Το αποτέλεσμα αυτού ανιχνεύεται (observe) από το UI, προκειμένου να εκτελεστεί το κατάλληλο action.

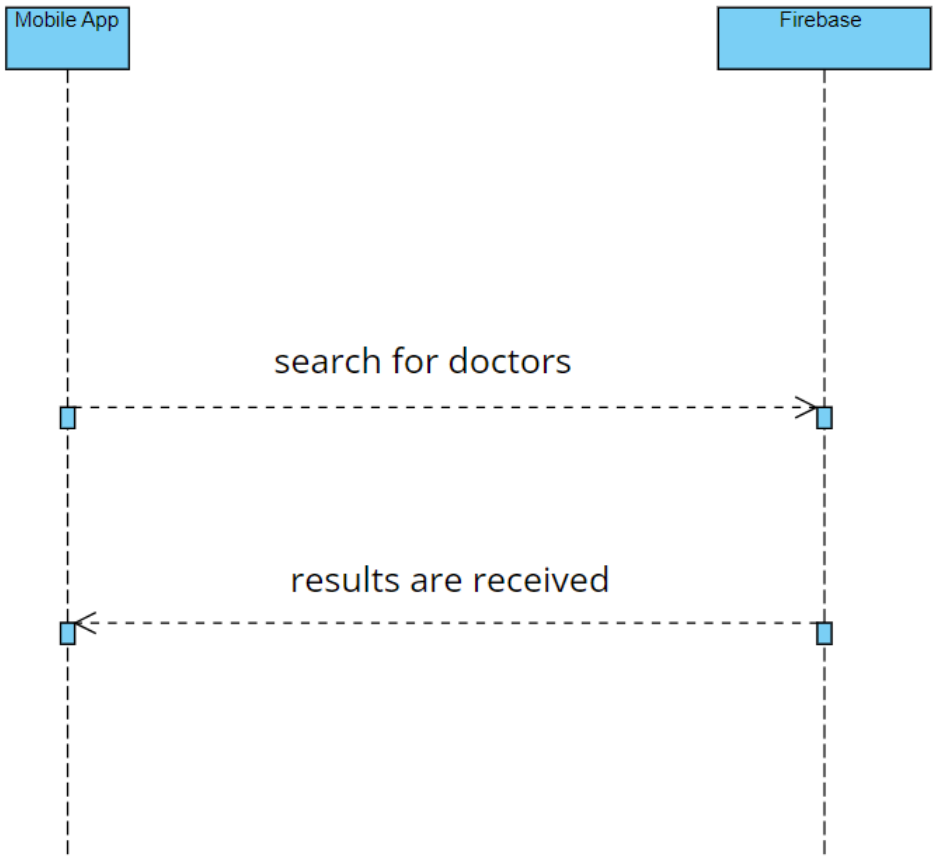
```
override val loginResult: LiveData<Status<LoginResult>?>
    get() = savedState.getLiveData(
        LOGIN_RESULT,
        DEFAULT_LOGIN_RESULT,
    )
```

Στην περίπτωση της εφαρμογής, αν η διαδικασία είναι επιτυχής, τότε ο χρήστης ανακατευθύνεται στην κύρια οθόνη της εφαρμογής. Σε διαφορετική περίπτωση εμφανίζεται ένα toast message με το κατάλληλο μήνυμα λάθους.

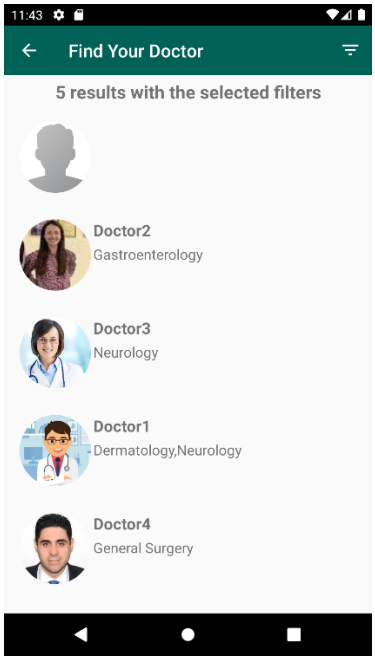
```
observe(viewModel.loginResult) { loginResult ->
    loginResult
        ?.onSuccess {  it: LoginResult
            val userId = it.id
            val deviceToken = it.deviceToken
            if (!userId.isNullOrEmpty() && !deviceToken.isNullOrEmpty()) {
                showToast("You logged in successfully")
                setUserSharedPreferences(
                    id = userId,
                    deviceToken = deviceToken,
                )
                if (userIsPatient)
                    goToPatientContactsActivity()
                else if (userIsDoctor)
                    goToDoctorContactsActivity()
            } else
                showToast(messageId = "Oops, something went wrong! Please try again!")
        }?.onFailure {  it: Throwable
            showToast(messageId = "Oops, something went wrong! Please try again!")
        }
}
```

4.5.3 Αναζήτηση Γιατρού

Ως τρίτο και τελευταίο workflow περιγράφεται αυτό αναζήτησης γιατρού μέσω της εφαρμογής. Πιο συγκεκριμένα ο χρήστης (στην προκειμένη περίπτωση ένας Ασθενής), έχει ορίσει κάποια φίλτρα μέσω της αντίστοιχης οθόνης και πατάει το κουμπί της Αναζήτησης. Αυτό που θα συμβεί, είναι ένα request στη ΒΔ της Firebase στην οθόνη αποτελεσμάτων και αφού επιστραφούν τα αποτελέσματα. Επομένως κάθε φορά που ο χρήστης επισκέπτεται την οθόνη αποτελεσμάτων, πραγματοποιείται ένα request προς την αντίστοιχη ΒΔ της Firebase. Όλα τα παραπάνω περιγράφονται με το ακόλουθο **Sequence Diagram**:



Σε επίπεδο κώδικα, οι παραπάνω ενέργειες μεταφράζονται ως εξής: Αρχικά όλη η πληροφορία για τα φίλτρα του χρήστη (fullName, distance, specialties) είναι αποθηκευμένη σε ένα ViewModel.



Όταν ο χρήστης πατήσει το κουμπί της Αναζήτησης, ξεκινά η εκτέλεση ενός request με όνομα **searchForDoctors().** Ο ακόλουθος κώδικα βρίσκεται μέσα στο αρχείο **FindDoctorSearchResultsScreen.**

```
private fun searchForDoctors() {
    viewModel.searchForDoctors()
}
```

Πατώντας επομένως το κουμπί της , εκτελείται η συνάρτηση **searchForDoctors()** του ViewModel, η οποία εκτελεί ένα request στη μη σχεσιακή ΒΔ της Firebase, χρησιμοποιώντας τα ανάλογα φίλτρα. Αυτό φαίνεται και στην παρακάτω εικόνα του αρχείου **DoctorFiltersSearchViewModel**. Για την ολοκλήρωση των διαδικασιών που μόλις περιγράφτηκαν, χρησιμοποιούνται τα κατάλληλα useCases.

```
override fun searchForDoctors() {
    viewModelScope.launch { this: CoroutineScope
        doctorListValue = useCase.searchForMultipleNodesWithFilters(
            fullName = fullNameValue,
            specialties = doctorSpecialtiesValue,
            distance = selectedDistanceFilterValue?.distance?.value,
            patientGeoLocation = selectedDistanceFilterValue?.userLocation,
        )
    }
}
```

Το αποτέλεσμα της όλης διαδικασία αποθηκεύεται σε ένα live data με όνομα **doctorList**. Το αποτέλεσμα αυτού ανιχνεύεται (observe) από το UI, προκειμένου να εκτελεστεί το κατάλληλο action.

```
override val doctorList: LiveData<Status<List<FirebaseDoctorUI>>?>
    get() = savedState.getLiveData(
        DOCTOR_LIST,
        DEFAULT_DOCTOR_LIST,
    )
```

Στην περίπτωση της εφαρμογής, αν η διαδικασία είναι επιτυχής, τότε η λίστα γεμίζει με αποτελέσματα και προβάλλεται στον χρήστη. Σε διαφορετική περίπτωση εμφανίζεται ένα toast message με το κατάλληλο μήνυμα λάθους.

```
private fun observeAll(lifecycleOwner: LifecycleOwner?) = lifecycleOwner?.apply {
    observe(viewModel.doctorList) { status ->
        status
            ?.onSuccess { doctorList ->
                adapter.updateDataSource(doctorList = doctorList)
                updateTitle(results = doctorList.size)
            }?.onFailure { it: Throwable
                showToast("Something went wrong! Please try again")
            }
    }
}
```

Ενότητα 5 - Συμπεράσματα και μελλοντικές επεκτάσεις

Το παρών κείμενο προσφέρεται για συμπεράσματα και μελλοντικές επεκτάσεις σχετικά με τη μεταπτυχιακή διατριβή. Πιο συγκεκριμένα:

Η χρησιμοποίηση μίας ιατρικής εφαρμογής ανταλλαγής μηνυμάτων (chat) υπό κατάλληλες προϋποθέσεις, διευκολύνει την επικοινωνία μεταξύ ασθενούς και γιατρού. Οι προϋποθέσεις αυτές έχουν αναλυθεί στην Εισαγωγή. Από την άλλη πλευρά, μία τέτοια εφαρμογή ελλοχεύει αρκετούς κινδύνους, αν η χρήση του δεν γίνει με σωστό τρόπο. Πολλές φορές ένας ασθενής μπορεί να κατακλύζεται από το φόβο μίας ασθένειας, με αποτέλεσμα να επικοινωνήσει περισσότερο απ' όσο χρειάζεται με το γιατρό. Με αυτό τον τρόπο η πληροφορία που ανταλλάσσεται μεταξύ γιατρού και ασθενή μετατρέπεται σε "θόρυβο", ειδικότερα αν ο γιατρός χρειάζεται να διαχειριστεί πολλά περιστατικά.

Η μεταπτυχιακή διατριβή προσφέρεται και για μελλοντική επέκταση του project, προσθέτοντας τις ακόλουθες λειτουργίες:

1. push notifications (ειδοποιήσει κάθε φορά που ένας χρήστης δέχεται κάποιο καινούριο μήνυμα)
2. δυνατότητα κλήσης μεταξύ γιατρού και ασθενή
3. αποστολή φωτογραφιών μεταξύ γιατρού και ασθενή
4. αξιολόγηση γιατρού από πλευράς ασθενή
5. κλείσιμο ραντεβού
6. ημερολόγιο ραντεβού
7. προσθήκη καρτέλας περαιτέρω πληροφοριών γιατρού (βιογραφικό, εμπειρία, κλπ.)

Βιβλιογραφία

https://play.google.com/store/apps/details?id=com.mediquo.main&hl=en_US
https://www.doctoranytime.gr/
https://www.digitalclinic.gr/el/
https://www.interamerican.gr/idiotes/proionta-ypiresies/ygeia/medion-app
https://developer.android.com/topic/modularization
https://www.javatpoint.com/spring-boot-architecture
https://developer.android.com/topic/architecture
https://www.geeksforgeeks.org/mvvm-model-view-viewmodel-architecture-pattern-in-android/

Μεταπτυχιακή Διατριβή

- Το κείμενο να είναι γραμμένο σε font size 10pt με μονό διάστιχο (single spacing) και απόσταση παραγράφων 3pt (after).
- Η πρώτη παράγραφος κάθε ενότητας να μην έχει εσοχή πρώτης γραμμής ενώ οι επόμενες να έχουν εσοχή πρώτης γραμμής.
- Τα περιθώρια σελίδας να είναι 3cm και στις τέσσερεις πλευρές (πάνω, κάτω, αριστερά, δεξιά).
- Τα Headings να είναι όλα με font **Arial Black** και όχι Bold. Το **Heading 1 να είναι 12pt**, το **Heading 2 να είναι 11pt**, το **Heading 3 να είναι 10pt**. Να μην χρησιμοποιείτε Heading 4 και πέρα.
- Να μην αφήνετε κενές γραμμές πριν ή μετά από τα headings και κάθε επίπεδο heading να απέχει 18pt before και 6pt after.
- Οι λεζάντες (captions) στα σχήματα και τους πίνακες να είναι αριστερά στοιχισμένες και να είναι **Arial bold 9pt**.
- Σε κάθε σελίδα να υπάρχει footer (Arial 8pt) με τον τίτλο της διατριβής στα αριστερά. Στο footer επίσης να υπάρχει αρίθμηση σελίδας στα δεξιά και πάλι με font Arial 8pt.
- Σε κάθε σελίδα να υπάρχει header (Arial 8pt) με το όνομα του φοιτητή στα δεξιά και το λεκτικό «Μεταπτυχιακή Διατριβή» **(MSc Thesis)** στα αριστερά.
- Τα header και footer να απέχουν από τα άκρα του χαρτιού 2.5cm (στο Page Setup).
- Η διατριβή να περιέχει απαραίτητα:
    - Περίληψη **(Abstract)** σε χωριστή σελίδα (μισή σελίδα Ελληνικά και μισή στα Αγγλικά).
    - Εισαγωγή **(Introduction)** – Σύντομη Περιγραφή Προβλήματος/Αντικειμένου **(Short description of the problem/subject)** (μέχρι 3 σελίδες).
    - ....
    - Συμπεράσματα **(Conclusions)** – Περίληψη **(Summary)**
    - Βιβλιογραφία **(Bibliography)**