ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ – ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Πρόγραμμα Μεταπτυχιακών Σπουδών

«Πληροφορική»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Ανάπτυξη εφαρμογής για την διαχείριση ιατρικών ραντεβού με χρήση τεχνολογιών AWS και Spring Boot.** **Application development for managing medical appointments using AWS and Spring Boot technologies.**
Ονοματεπώνυμο Φοιτητή	**Μπαλτά Ειρήνη**
Πατρώνυμο	**Κωνσταντίνος**
Αριθμός Μητρώου	**ΜΠΠΛ20051**
Επιβλέπων	**Ευθύμιος Αλέπης, Αναπληρωτής Καθηγητής**

Ημερομηνία Παράδοσης **11/2023**

Τριμελής Εξεταστική Επιτροπή

Ευθύμιος Αλέπης Αναπληρωτής Καθηγητής	Μαρία Βίρβου Καθηγήτρια	Κωνσταντίνος Πατσάκης Αναπληρωτής Καθηγητής

Ευχαριστίες

Η παρούσα μεταπτυχιακή διατριβή εκπονήθηκε κατά τη διάρκεια του ακαδημαϊκού έτους 2022-2023 και πραγματοποιήθηκε στα πλαίσια του Μεταπτυχιακού Προγράμματος Σπουδών του τμήματος Πληροφορική, του Πανεπιστημίου Πειραιώς.

Θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα της μεταπτυχιακής μου διατριβής, Αναπλ. Καθηγητή του ΠΑ.ΠΕΙ., Κο Ευθύμιο Αλέπη, για την πολύτιμη καθοδήγηση, υποστήριξη, ανεκτίμητη υπομονή, ανελλιπή κατανόηση και ενθάρρυνση που μου παρείχε σε κάθε στάδιο υλοποίησης της παρούσας διατριβής. Επιθυμώ να του εκφράσω τη βαθιά μου ευγνωμοσύνη για την αμέριστη εμπιστοσύνη που επέδειξε στο πρόσωπό μου. Οι συμβουλές και η συνεχής του παρουσία ήταν καθοριστική για την ολοκλήρωση αυτού του έργου, ενώ μέσα από τα μαθήματά του ανακάλυψα την αγάπη μου για τον προγραμματισμό.

Θερμές ευχαριστίες θέλω να εκφράσω και στα υπόλοιπα μέλη της επιτροπής: την καθηγήτρια του τμήματος Πληροφορικής του Πανεπιστημίου Πειραιώς Κα Μαρία Βίρβου και τον Αναπληρωτή Καθηγητή Κο Κωνσταντίνο Πατσάκη για την πρόθυμη συμμετοχή τους στη κρίση της μεταπτυχιακής αυτής διατριβής.

Ιδιαίτερες ευχαριστίες προς τον αγαπημένο μου αδελφό Δημήτρη, ο οποίος δεν είναι μόνο ένας ενθαρρυντικός υποστηρικτής των ονείρων μου, αλλά και ένας σημαντικός καθοδηγητής στο δύσκολο αυτό ταξίδι με πλούσια εμπειρία και γνώση στον επιστημονικό αυτό κλάδο.

Επίσης ένα πολύ μεγάλο ευχαριστώ προς τους συναδέλφους και φίλους μου, Κάρλε Νεκτάριο - Παναγιώτη και Καραβάτσο Γεώργιο, για την ανεκτίμητη στήριξη που μου προσέφεραν καθ' όλη τη διάρκεια αυτής της απαιτητικής πορείας.

Τέλος, θα ήθελα να ευχαριστήσω τους γονείς μου για όλη την αγάπη και την συμπαράσταση τους καθ' όλη τη διάρκεια των ακαδημαϊκών μου σπουδών.

Η παρούσα μεταπτυχιακή μου διατριβή αφιερώνεται στο βαφτιστήρι και ανιψιό μου Κωνσταντίνο.

Περιεχόμενα

Περίληψη

Η διατριβή αυτή αποτελεί μια μελέτη που επικεντρώνεται στον σχεδιασμό, στην ανάπτυξη και στην υλοποίηση μιας εφαρμογής για την ψηφιακή διαχείριση ιατρικών ραντεβού. Η εφαρμογή αυτή προσφέρει μια σύγχρονη λύση για τη βελτίωση της διαχείρισης των ραντεβού, την ενίσχυση της επικοινωνίας μεταξύ ιατρών και ασθενών, και την ασφαλή αποθήκευση των ιατρικών δεδομένων.

Η διαδικασία σχεδιασμού της εφαρμογής περιλαμβάνει τον καθορισμό των, λειτουργικών και μη λειτουργικών της, απαιτήσεων. Η ανάπτυξη και υλοποίηση της πραγματοποιήθηκαν με τη χρήση διαφορετικών τεχνολογιών που εξασφαλίζουν την αποδοτικότητάς της.

Όσον αφορά το backend της εφαρμογής, ο κώδικας υλοποιήθηκε στη γλώσσα προγραμματισμού Java με τεχνολογίες του Spring (Spring Boot). Η εφαρμογή εκτελείται στις υπηρεσίες της Amazon Web Services(AWS) παρέχοντας ένα αξιόπιστο και επεκτάσιμο περιβάλλον για την λειτουργία της. Συγκριμένα οι υπηρεσίες ECS (Elastic Container Service) και ECR (Elastic Container Registry) συμπληρώνουν την τεχνολογική υποδομή που επιτρέπει την απρόσκοπτη λειτουργία της εφαρμογής στο περιβάλλον της AWS και επιλέχθηκαν ως υπηρεσίες διαχείρισης των Docker containers για τον έλεγχο, την κλιμάκωση και την αποθήκευση της εφαρμογής. Η πλήρως διαχειριζόμενη υπηρεσία βάσης δεδομένων σχεσιακού τύπου, με το όνομα Aurora της AWS, γνωστή και ως RDS (Relational Database Service), επιλέχθηκε για την αποθήκευση και την ανάκτηση των δεδομένων με αποτελεσματικότητα και ασφάλεια. Πρόσθετα χρησιμοποιήθηκε η H2 ως βάση δεδομένων μνήμης για τη διατήρηση των δεδομένων κατά την διάρκεια της ανάπτυξης και δοκιμής της εφαρμογής.

Όσον αφορά το frontend της εφαρμογής, χρησιμοποιήθηκαν τα εργαλεία Thymeleaf (server-side rendering), JavaScript, CSS και HTML έτσι ώστε να μπορούν οι χρήστες να επικοινωνούν και να οργανώνουν τις ιατρικές συναντήσεις με έναν εύχρηστο και φιλικό προς τον χρήστη τρόπο.

Τέλος χρησιμοποιήθηκε η αρχιτεκτονική REST API για την διασύνδεση της εφαρμογής με άλλα συστήματα και η τεχνική ORM (Object-Relational Mapping) για τη διαχείριση και την αλληλεπίδραση με τις βάσεις δεδομένων.

ABSTRACT

This dissertation represents a study that focuses on the design, development, and implementation of an application for digital medical appointment management. This application offers a modern solution for improving appointment management, enhancing communication between doctors and patients, and securely storing medical data.

The application's design process includes defining both its functional and non-functional requirements. The development and implementation were carried out using various technologies to ensure its efficiency.

Regarding the backend of the application, the code was implemented in the Java programming language using Spring technologies (Spring Boot). The application runs on Amazon Web Services (AWS) services, providing a reliable and scalable environment for its operation. Specifically, the ECS (Elastic Container Service) and ECR (Elastic Container Registry) services complement the technological infrastructure that enables the seamless operation of the application in the AWS environment and were chosen as the Docker container management services for control, scalability, and storage of the application. The fully managed relational database service, known as Aurora within AWS, also referred to as RDS (Relational Database Service), was selected for data storage and retrieval, ensuring efficiency and security. Additionally, H2 was used as an in-memory database for data retention during the development and testing of the application.

Regarding the frontend of the application, Thymeleaf (server-side rendering), JavaScript, CSS, and HTML were utilized, allowing users to communicate and organize medical appointments in a user-friendly and intuitive manner.

Finally, the REST API architecture was employed to interface the application with other systems, and Object-Relational Mapping (ORM) techniques were used for database management and interaction.

1. Εισαγωγή

Η τεχνολογία της πληροφορικής έχει διεισδύσει σε κάθε πτυχή της κοινωνίας μας, επανασχεδιάζοντας τον τρόπο με τον οποίο αλληλοεπιδρούμε, εργαζόμαστε και ανταλλάσσουμε πληροφορίες. Στον κλάδο της υγείας, η πρόσβαση σε σύγχρονες τεχνολογίες και υποδομές μπορεί να διαδραματίσει έναν κρίσιμο ρόλο στην βελτίωση της ποιότητας της περίθαλψης, την ενίσχυση της αποτελεσματικότητας και την επίτευξη ολοκληρωμένων λύσεων που εξυπηρετούν τις ανάγκες των ασθενών και των επαγγελματιών υγείας.

Στη σύγχρονη ψηφιακή εποχή, η ανάγκη για αποτελεσματικότερη διαχείριση των ιατρικών ραντεβού είναι επιτακτική και αποτελεί κινητήριο παράγοντα στην ανάπτυξη του ψηφιακού προγραμματισμού των ραντεβού. Οι παραδοσιακές μέθοδοι προ συνεννόησης και διαχείρισης ραντεβού στον τομέα της υγείας είναι χρονοβόρες και ως αποτέλεσμα έχουν την καθυστέρηση της ιατρικής περίθαλψης για τους ασθενείς, σε αντίθεση με τον ψηφιακό προγραμματισμό ραντεβού που μπορεί να βελτιώσει την εμπειρία των ασθενών και των ιατρών, μειώνοντας τον χρόνο αναμονής και βελτιστοποιώντας την διαχείριση του χρόνου των ιατρών.

Η παρούσα διατριβή επικεντρώνεται στην ανάπτυξη μιας προηγμένης εφαρμογής διαδικτυακής καταχώρησης ιατρικών ραντεβού, με τη βοήθεια των υπηρεσιών του Amazon Web Services (AWS). Χρησιμοποιώντας μια ποικιλία προηγμένων τεχνολογιών, εργαλείων και σύγχρονων αρχιτεκτονικών, δημιουργήθηκε ένα ευέλικτο σύστημα που διευκολύνει τη διαδικασία διαχείρισης των ιατρικών ραντεβού, την ανταλλαγή ιατρικού περιεχομένου μεταξύ ιατρών και ασθενών καθώς επίσης βοηθά στην εξοικονόμηση χρόνου και στη βελτίωση της ποιότητας της ιατρικής φροντίδας.

1.1 Στόχος και Σκοπός της Διατριβής

Η χρήση της τεχνολογίας και της πληροφορικής αποτελεί βασική προϋπόθεση για την ανάπτυξη προϊόντων ιατρικού λογισμικού. Η ανάγκη για την ανάπτυξη τέτοιων προϊόντων αποτελεί ένα σημαντικό ζήτημα στον τομέα της υγείας και έχει απασχολήσει εξειδικευμένο επιστημονικό και τεχνικό προσωπικό που διαθέτει εξαιρετικά βαθιά και εκτεταμένη γνώση και εμπειρία στον τομέα των Ιατρικών Πληροφοριακών Συστημάτων.

Η ανάπτυξη λογισμικού για τον ιατρικό τομέα είναι κρίσιμης σημασίας για τη βελτίωση της ποιότητας της υγειονομικής φροντίδας, την αποτελεσματική διαχείριση των ασθενούντων, και την αυξημένη ασφάλεια των ιατρικών δεδομένων.

Ο στόχος της παρούσας μεταπτυχιακής διατριβής είναι η ανάπτυξη και υλοποίηση ενός δυναμικού ψηφιακού συστήματος με σκοπό να δημιουργηθεί ένα εύχρηστο και αξιόπιστο εργαλείο για τη διαχείριση των ιατρικών ραντεβού, την αποτελεσματική επικοινωνία μεταξύ ιατρών και ασθενών, καθώς και την ανάπτυξη μιας αξιόπιστης βάσης δεδομένων και μιας ασφαλούς αρχιτεκτονικής για την αποθήκευση και την ανάκτηση ιατρικών δεδομένων.

1.2 Εταιρικές υλοποιήσεις

Υπάρχουν πολλές εταιρείες παγκοσμίως που διαθέτουν καινοτόμα προϊόντα Ιατρικού λογισμικού και παρέχουν υπηρεσίες Ιατρικής Πληροφορικής. Ορισμένες από αυτές είναι πολύ δημοφιλείς και έχουν καταφέρει να αποκτήσουν ευρεία αποδοχή. Τα πιο γνωστά λογισμικά/ εφαρμογές (Software as a service) σε διεθνές επίπεδο είναι:

- Doctolib: Πρόκειται για ένα δημοφιλές λογισμικό που επιτρέπει την ηλεκτρονική κράτηση ραντεβού με γιατρούς, τη διαχείριση ραντεβού σε ιατρεία και εγκαταστάσεις υγειονομικής περίθαλψης καθώς και τηλεϊατρικές βιντεοσκοπήσεις και επικοινωνία μεταξύ γιατρών και ασθενών στην Ευρώπη. Το λογισμικό αυτό χρησιμοποιήθηκε στο Βερολίνο και τη Γαλλία για τον συντονισμό των ραντεβού για τους εμβολιασμούς κατά του COVID-19.

  Οι τεχνολογίες που χρησιμοποιούνται είναι:

  - Front-end: HTML, CSS, JavaScript, React.js
  - Back-end: Ruby, Rust
  - Βάση Δεδομένων: Redis

- Zocdoc: Πρόκειται για μια δημοφιλή εφαρμογή, η οποία χρησιμοποιείται αποκλειστικά στις ΗΠΑ, που επιτρέπει στους ασθενείς να βρίσκουν και να κάνουν κράτηση ραντεβού αυτοπροσώπως ή τηλεϊατρικής για ιατρική ή οδοντιατρική περίθαλψη. Η πλατφόρμα λειτουργεί επίσης ως βάση δεδομένων αξιολόγησης και σύγκρισης γιατρών και οδοντιάτρων.

  Οι τεχνολογίες που χρησιμοποιούνται είναι:

  - Front-end: HTML, CSS, JavaScript, React.js, TypeScript
  - Back-end: Python, Node.js, Swift, C#, .Net, Golang
  - Cloud: Amazon Web Services (AWS)

- DocPlanner: Πρόκειται για μια διεθνή εφαρμογή με παρουσία σε πολλές χώρες(Πολωνία, Ισπανία, Γερμανία και Ιταλία) που επαναπροσδιορίζει την πρόσβαση στην υγειονομική περίθαλψη.

  Οι τεχνολογίες που χρησιμοποιούνται είναι:

  - Front-end: HTML, CSS, JavaScript, Angular.js, Less
  - Back-end: PHP, C#, Node.js, .Net, Golang

- Βάση Δεδομένων: PostgreSQL, MongoDB, MariaDB
- Cloud: Amazon Web Services (AWS)

- NexHealth: Πρόκειται για μια δημοφιλή εφαρμογή στις ΗΠΑ ηλεκτρονικού μητρώου υγείας (EHR) που παρέχει υπηρεσίες διαχείρισης στον κλάδο της υγειονομικής περίθαλψης.

  Οι τεχνολογίες που χρησιμοποιούνται είναι:

  - Front-end: HTML, CSS, jQuery, Angular.js
  - Back-end: Ruby
  - Cloud: Amazon Web Services (AWS)

- Praktikertjänst: Πρόκειται για μια από τις μεγαλύτερες εφαρμογές ψηφιακής υποστήριξης της υγείας και οδοντιατρικής περίθαλψης στη Σουηδία.

  Οι τεχνολογίες που χρησιμοποιούνται είναι:

  - Front-end: HTML, CSS, JavaScript, React.js
  - Back-end: Java, Spring Boot
  - Βάση Δεδομένων: MySQL
  - Cloud: Microsoft Azure

- Curefit: Πρόκειται για μια δημοφιλή εφαρμογή στην Ινδία που προσφέρει όχι μόνο ψηφιακή υποστήριξη στο τομέα της υγείας αλλά και άλλες υπηρεσίες σε θέματα φυσικής κατάστασης, διατροφής και ψυχικής ευεξίας.

  Οι τεχνολογίες που χρησιμοποιούνται είναι:

  - Front-end: HTML, CSS, JavaScript, Angular
  - Back-end: Java, Spring Boot
  - Βάση Δεδομένων: MongoDB
  - Cloud: Amazon Web Services (AWS)

- **MyChart:** Πρόκειται για μια διαδικτυακή εφαρμογή που χρησιμοποιείται σε διάφορες χώρες και σχετίζεται με την υγειονομική περίθαλψη που επιτρέπει στους ασθενείς να αλληλοεπιδρούν και να επικοινωνούν με τους παρόχους υγειονομικής περίθαλψης (γιατρούς και νοσοκομεία).

  Οι τεχνολογίες που χρησιμοποιούνται είναι:

  - Front-end: HTML, CSS, JavaScript, Angular
  - Back-end: Java, Spring Boot
  - Βάση Δεδομένων: Microsoft SQL Server
  - Cloud: Microsoft Azure

Είναι σημαντικό να αναφερθεί ότι αυτές είναι ορισμένες από τις τεχνολογίες που χρησιμοποιούνται σε αυτές τις εφαρμογές. Οι τεχνολογίες επιλέγονται ανάλογα με τις απαιτήσεις και τις προτιμήσεις της ανάπτυξης της εκάστοτε εφαρμογής. Οι παραπάνω τεχνολογίες αποτελούν μια γενική εικόνα καθώς υπάρχουν επιπλέον τεχνολογίες και συνδυασμοί που χρησιμοποιούνται για την ανάπτυξη των εφαρμογών αυτών.

2. Μεθοδολογία

2.1 Ανάλυση απαιτήσεων

Στο πρώτο στάδιο, πραγματοποιήθηκε μια λεπτομερής ανάλυση των απαιτήσεων του συστήματος. Αυτές οι απαιτήσεις περιλαμβάνουν:

2.1.1 Λειτουργικές απαιτήσεις

Οι λειτουργικές απαιτήσεις περιγράφουν τις λειτουργίες του συστήματος που σχεδιάστηκε. Πρόκειται για μια περιγραφή του τι είναι το σύστημα και πώς λειτουργεί για να ικανοποιήσει τις ανάγκες των χρηστών. Παρέχουν μια σαφή περιγραφή του τρόπου με τον οποίο το σύστημα ανταποκρίνεται σε μια συγκεκριμένη εντολή, τα χαρακτηριστικά και τι περιμένουν οι χρήστες. Συνεπώς οι λειτουργικές απαιτήσεις της εφαρμογής είναι:

1. Διαχείριση ιατρικών ραντεβού:
   1.1. Οι ιατροί και οι ασθενείς πρέπει να μπορούν να δημιουργήσουν λογαριασμό στην εφαρμογή.
   1.2. Οι ασθενείς πρέπει να μπορούν να προγραμματίσουν ραντεβού με ιατρούς.
   1.3. Οι ιατροί πρέπει να μπορούν να διαχειρίζονται τις ώρες και τις ημέρες των ραντεβού τους και να τα ακυρώνουν αν το επιθυμούν.

1.4. Οι ασθενείς πρέπει να μπορούν να βλέπουν τις καταγραφές των ραντεβού τους και τυχόν ακυρώσεις από τους ιατρούς.
1.5. Οι ασθενείς πρέπει να έχουν την επιλογή ακύρωσης των ραντεβού τους.
1.6. Τα ραντεβού πρέπει να έχουν σχετικές πληροφορίες, όπως ημερομηνία, ώρα, ιατρός, ασθενής, λόγο επίσκεψης κλπ.

2. Επικοινωνία με άλλες συσκευές ή προγράμματα μέσω REST APIs:
   2.1. Η εφαρμογή πρέπει να παρέχει REST APIs για την επικοινωνία με άλλες εξωτερικές συσκευές ή προγράμματα.
   2.2. Τα APIs πρέπει να είναι ασφαλείς και να παρέχουν τις απαιτούμενες λειτουργίες για την ανταλλαγή πληροφοριών.

2.1.2 Μη λειτουργικές απαιτήσεις

Οι μη λειτουργικές απαιτήσεις αποτελούν σημαντικό μέρος της ανάπτυξης ενός συστήματος. Βοηθούν να διασφαλιστεί ότι το σύστημα ανταποκρίνεται στις ανάγκες του χρήστη και είναι σε θέση να λειτουργεί όπως προβλέπεται. Συνεπώς οι μη Λειτουργικές απαιτήσεις της εφαρμογής είναι:

- Ασφάλεια: Το σύστημα πρέπει να είναι ασφαλές από μη εξουσιοδοτημένη πρόσβαση.
- Απόδοση: Το σύστημα πρέπει να είναι σε θέση να χειρίζεται τον απαιτούμενο αριθμό χρηστών χωρίς καμία υποβάθμιση στην απόδοση.
- Ευελιξία: Το σύστημα πρέπει να μπορεί να αυξάνει ή να μειώνει ανάλογα με τις ανάγκες.
- Διαθεσιμότητα: Το σύστημα πρέπει να είναι διαθέσιμο όταν χρειάζεται.
- Συντήρηση: Το σύστημα πρέπει να είναι εύκολο στη συντήρηση και ενημέρωση.
- Φορητότητα: Το σύστημα πρέπει να μπορεί να λειτουργεί σε διαφορετικές πλατφόρμες με ελάχιστες αλλαγές.
- Αξιοπιστία: Το σύστημα πρέπει να είναι αξιόπιστο και να ανταποκρίνεται στις απαιτήσεις του χρήστη.
- Ευχρηστία: Το σύστημα πρέπει να είναι εύκολο στη χρήση και κατανοητό.
- Συμβατότητα: Το σύστημα πρέπει να είναι συμβατό με άλλα συστήματα.

Σύμφωνα με τα παραπάνω:

- ✓ Η υπηρεσία ECS σε συνδυασμό με την υπηρεσία RDS συμβάλουν για την κάλυψη της ασφάλειας του συστήματος.
- ✓ Η απόδοση επιτεύχθηκε με την χρήση της υπηρεσίας ECS η οποία έχει τη δυνατότητα να κατανέμει αυτόματα τους χρήστες της εφαρμογής, στους διαθέσιμους πόρους(container instance) με τη βοήθεια της τεχνολογίας load balancer.

- ✓ Η ευελιξία καλύπτεται από την υπηρεσία του ECS.
- ✓ Όσον αφορά τη διαθεσιμότητα του συστήματος, η AWS ως πάροχος υπηρεσιών τεχνολογιών νέφους είναι διαθέσιμη όλο το 24ωρο κατ' επέκταση και η εφαρμογή που εκτελείται μέσω αυτής.
- ✓ Λόγω της σχεδίασης της εφαρμογής είναι εύκολη η επέκτασή της γιατί όλες οι λειτουργίες είναι μέθοδοι, επομένως είναι εύκολη στη συντήρηση.
- ✓ Η φορητότητα επιτεύχθηκε με τη χρήση του εργαλείου container (docker) το οποίο περιλαμβάνει όλα τα πακέτα και τις βιβλιοθήκες που χρειάζεται η εφαρμογή για να εκτελεστεί.
- ✓ Κατά την σχεδίαση έγινε λεπτομερής ανάλυση έτσι ώστε κατά την υλοποίηση να καλυφθούν όλες οι ανάγκες του χρήστη. Συνεπώς καλύπτεται η αξιοπιστία του συστήματος.
- ✓ Η εφαρμογή είναι συμβατή με όλες τις συσκευές των χρηστών (smartphone,pc και tablet) μέσω της τεχνολογίας του Thymeleaf engine και του HTML5.

2.2 Ανάλυση και σχεδίαση με UML

Το εργαλείο που χρησιμοποιήθηκε για την καλύτερη κατανόηση και ανάπτυξη του λογισμικού συστήματος και βοήθησε στην επιτυχή υλοποίηση της εφαρμογής "EKB HealthEngine", είναι τα διαγράμματα UML.

Τα διαγράμματα UML (Unified Modeling Language) είναι ένα σύνολο γλωσσών και γραφικών σημάτων που χρησιμοποιούνται στον κόσμο της πληροφορικής και της ανάπτυξης λογισμικού για την αναπαράσταση, τον σχεδιασμό, την τεκμηρίωση και την ανάλυση συστημάτων. Παρέχουν ένα κοινό γλωσσικό πλαίσιο που μπορεί να κατανοηθεί από τους προγραμματιστές, τους αναλυτές, τους σχεδιαστές και άλλους εμπλεκόμενους σε ένα έργο λογισμικού.

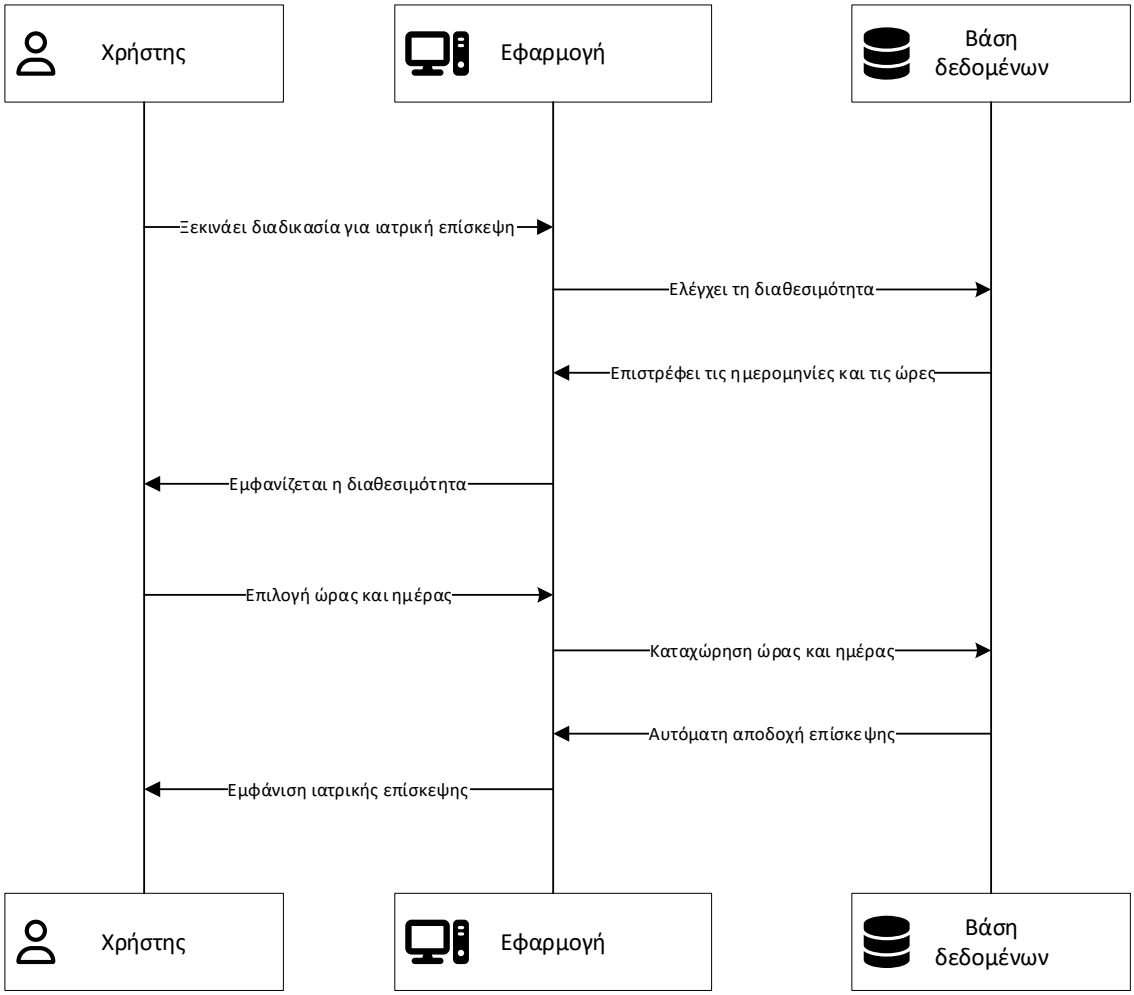
Οι βασικές συνιστώσες της εφαρμογής "EKB HealthEngine" περιλαμβάνουν τους χρήστες (ιατρούς και ασθενείς), τις λειτουργίες που μπορούν να εκτελέσουν, τις διεπαφές και τις σχέσεις μεταξύ τους.

Με βάση την ανάλυση απαιτήσεων της εφαρμογής, αναπτύχθηκαν τα κατάλληλα διαγράμματα UML που θα αναπαραστήσουν τη δομή και τη λειτουργία της εφαρμογής. Αυτά είναι:

- διαγράμματα περίπτωσης χρήσης(use case diagram)
- δραστηριοτήτων (activity diagram)
- και ακολουθίας (sequence diagram)

2.2.1 Διάγραμμα δραστηριοτήτων (activity diagram)

Το διάγραμμα δραστηριοτήτων είναι μια ειδική κατάσταση διαγράμματος στο οποίο όλες (ή τουλάχιστον οι περισσότερες) μεταβάσεις προκαλούνται από την ολοκλήρωση των δράσεων των πηγαίων καταστάσεων. Σκοπός του διαγράμματος αυτού είναι να εστιάσει σε ροές οι οποίες προκαλούνται από την εξωτερική επεξεργασία. Χρησιμοποιείται στις περιπτώσεις όπου όλα ή τα περισσότερα συμβάντα αναπαριστούν την ολοκλήρωση εσωτερικά παραγόμενων δράσεων ,δηλαδή την διαδικαστική ροή ελέγχου. Ακολουθεί ένας γράφος δραστηριοτήτων.

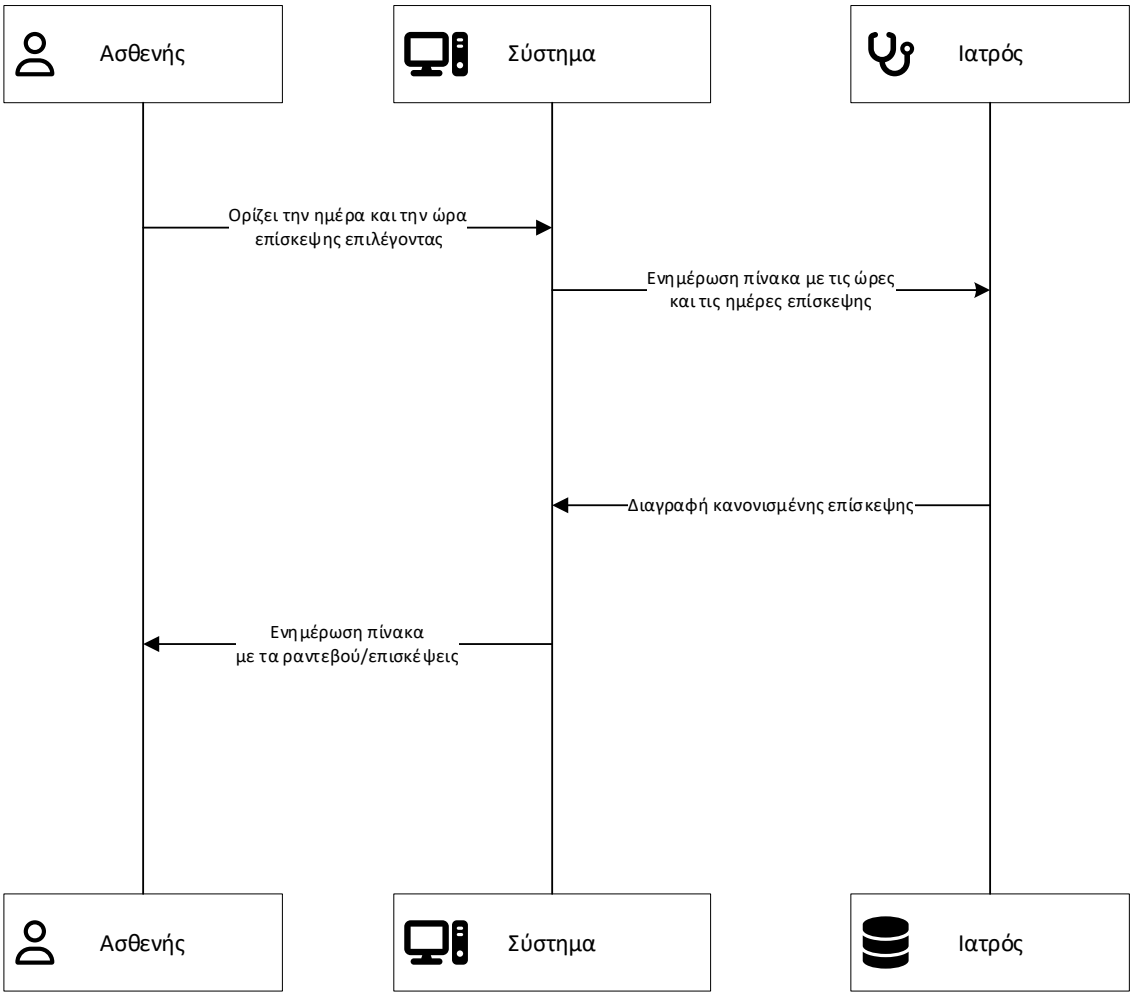


Διάγραμμα 1: Διάγραμμα δραστηριοτήτων (Activity Diagram).

2.2.2 Διάγραμμα περίπτωσης χρήσης (use case diagram)

Οι περιπτώσεις χρήσης είναι ένα σύνολο σεναρίων που συνδέονται με ένα συγκεκριμένο σκοπό του χρήστη. Στην πραγματικότητα το use case είναι οι λειτουργίες του συστήματος οι οποίες παριστάνονται με ένα πιο οργανωμένο τρόπο. Τα διαγράμματα Περίπτωσης Χρήσης στοχεύουν στο να:

1. Καθοριστούν και να περιγραφούν οι λειτουργικές απαιτήσεις του συστήματος
2. Δίνουν μια σαφή και συνεπή περιγραφή για το τι θα πρέπει να κάνει το σύστημα
3. Παρέχουν την ικανότητα να εντοπίζονται οι λειτουργικές απαιτήσεις μέσα στις κλάσεις και τις λειτουργίες του συστήματος.



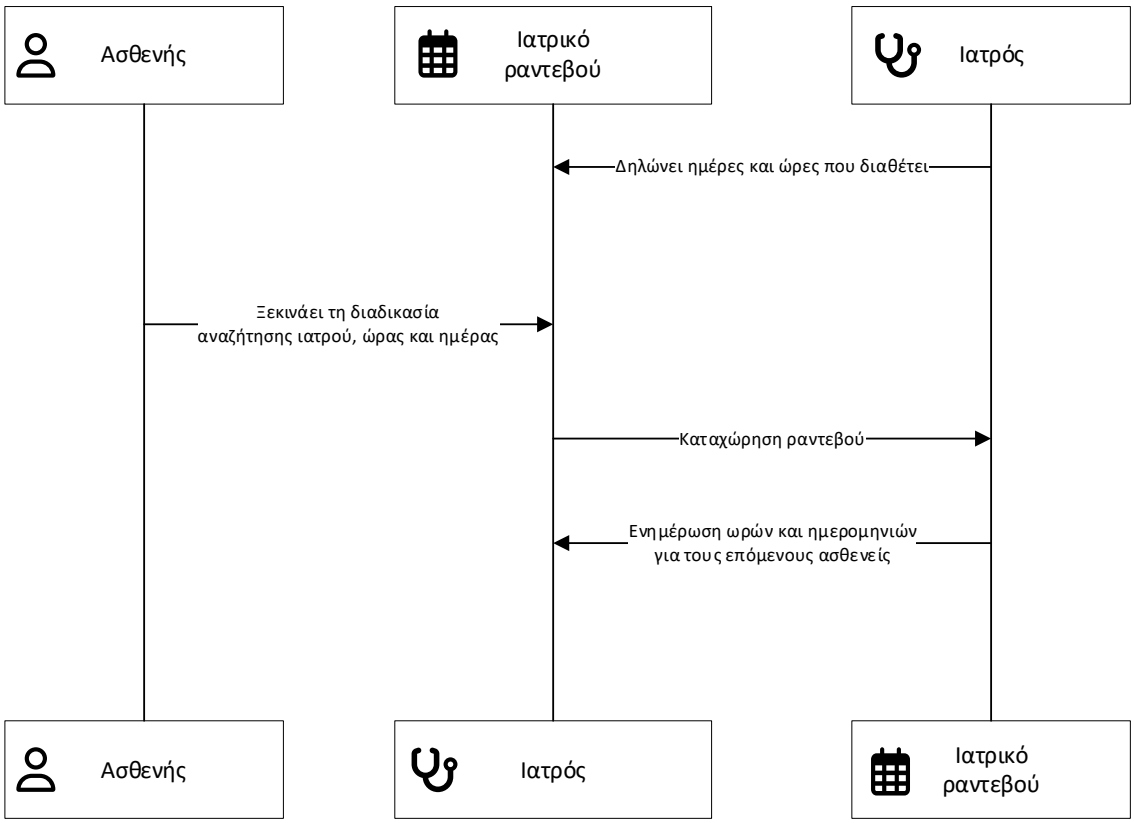
Διάγραμμα 2: Διάγραμμα περίπτωσης χρήσης (Use case Diagram).

2.2.3 Διαγράμματα ακολουθίας (Sequence Diagrams)

Ένα διάγραμμα ακολουθίας αναπαριστά την αλληλεπίδραση, μία σειρά από μηνύματα ανάμεσα σε ταξινομητές (classifiers) μέσα σε μία συνεργασία ώστε να πραγματοποιηθεί μία επιθυμητή πράξη ή αποτέλεσμα. Ένα διάγραμμα ακολουθίας έχει δύο διαστάσεις:

1. την κάθετη διάσταση που αναπαριστά το χρόνο
2. και την οριζόντια διάσταση που αναπαριστά τα διάφορα αντικείμενα.

Κανονικά ο χρόνος προχωρά προς τα κάτω. Αν είναι επιθυμητό οι διαστάσεις μπορούν να αλλάξουν θέση.



Διάγραμμα 3: Διάγραμμα ακολουθίας (Sequence diagram).

2.2 Σχεδίαση αρχιτεκτονικής

Μετά την ανάλυση των απαιτήσεων, πραγματοποιήθηκε η σχεδίαση της αρχιτεκτονικής του συστήματος. Επιλέχθηκε να χρησιμοποιηθεί το μονολιθικό μοντέλο MVC (Model-View-Controller), προσφέροντας έτσι μια πιο οργανωμένη και ευκολότερη συντήρηση του κώδικα. Πρόκειται για ένα σχεδιαστικό πρότυπο που χρησιμοποιείται ευρέως στην ανάπτυξη λογισμικού, ιδιαίτερα σε εφαρμογές που έχουν γραφικό περιβάλλον χρήστη, για τον διαχωρισμό της εφαρμογής στο λογικό επίπεδο (model layer), στο επίπεδο της επεξεργασίας (controller layer) και στο επίπεδο της παρουσίασης (view layer) των δεδομένων της εφαρμογής.

Αναλύοντας τα τρία κύρια συστατικά του μοντέλου MVC:

- Μοντέλο (Model):

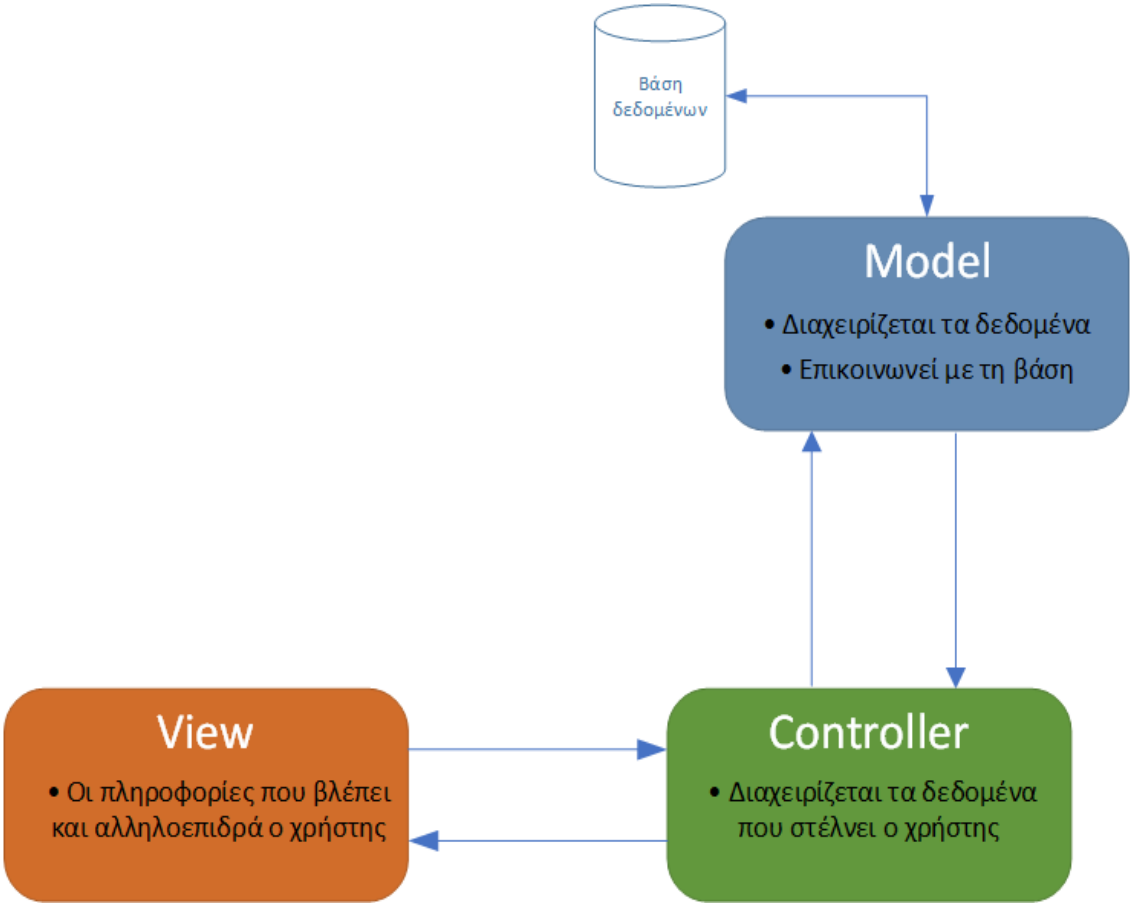
Το μοντέλο αναπαριστά τα δεδομένα και τη λογική της εφαρμογής. Περιλαμβάνει τις δομές δεδομένων και τις βάσεις δεδομένων, που απαιτούνται για την επεξεργασία και τη διαχείριση των δεδομένων. Το μοντέλο δεν έχει άμεση επαφή με τον χρήστη και έτσι ο χρήστης δεν έχει άμεση αντίληψη της υλοποίησης του μοντέλου.

- Προβολή (View):

Η προβολή αναλαμβάνει τον ρόλο της απεικόνισης των δεδομένων που παρέχονται από το μοντέλο στον χρήστη. Αυτή είναι η διεπαφή του χρήστη (UI) που βλέπει και αλληλεπιδρά με την εφαρμογή. Οι προβολές απεικονίζουν τα δεδομένα με τρόπο κατανοητό και επικεντρώνονται στην παρουσίαση.

- Ελεγκτής (Controller):

Ο ελεγκτής λειτουργεί ως ενδιάμεσος μεταξύ της προβολής (που βλέπει και αλληλεπιδρά με τον χρήστη) και του μοντέλου (που διαχειρίζεται τα δεδομένα της εφαρμογής) δηλαδή διαχειρίζεται τις εντολές που λαμβάνει από τον χρήστη και επικοινωνεί με το μοντέλο για να εκτελέσει τις αντίστοιχες εργασίες.



Σχεδιάγραμμα 1: Παρουσίαση μοντέλου MVC (Model-View-Controller).

2.3 Τεχνολογίες

Για την υλοποίηση της εφαρμογής, χρησιμοποιήθηκαν διάφορες τεχνολογίες έτσι ώστε να είναι πλήρης, να εκτελείται χωρίς διακοπές και έτοιμη για να χρησιμοποιηθεί από τους ασθενής και τους ιατρούς.

2.3.1 Java και Spring Boot

Η εφαρμογή αναπτύχθηκε με τη χρήση της γλώσσας προγραμματισμού Java και των εργαλείων Spring Boot.

Η Java επιλέχθηκε ως γλώσσα προγραμματισμού για την υλοποίηση της εφαρμογής λόγω της ευρείας αποδοτικότητας, της αξιοπιστίας και της δυνατότητας κλιμάκωσης που παρέχει. Επίσης είναι μια αντικειμενοστραφής γλώσσα που χρησιμοποιείται ευρέως στην ανάπτυξη εμπορικών εφαρμογών λόγω της ασφάλειας, της σταθερότητας και της ευελιξίας της.

Το Spring Boot είναι ένα ανοιχτού κώδικα framework που βασίζεται στη γλώσσα Java και χρησιμοποιείται για την ανάπτυξη γρήγορων και εύκολων εφαρμογών. Το Spring Boot παρέχει ένα ευέλικτο και παραγωγικό περιβάλλον ανάπτυξης, με αυτόματη διαμόρφωση και διαχείριση των βασικών λειτουργιών της εφαρμογής. Αυτό επιτρέπει στους προγραμματιστές να επικεντρωθούν στην ανάπτυξη της λογικής της εφαρμογής, από το να ασχολούνται με την περίπλοκη διαμόρφωση και ρύθμιση της υποδομής.

Ο λόγος που χρησιμοποιήθηκε το Spring Boot είναι η απλότητα και η ταχύτητα που παρέχει στην ανάπτυξη της εφαρμογής, καθώς και η δυνατότητα επέκτασης που παρέχει με τις αρχές σχεδιασμού REST.

2.3.2 REST APIs

Για την ανάπτυξη των διεπαφών προγραμματισμού εφαρμογής (APIs), χρησιμοποιήθηκαν τα REST (Representational State Transfer) APIs. Ο λόγος της χρήσης των REST APIs είναι ότι παρέχουν ένα σύνολο από κανόνες και πρωτόκολλα για την ανταλλαγή δεδομένων μεταξύ εφαρμογών. Χρησιμοποιώντας τα REST APIs, οι ασθενείς μπορούν να επικοινωνούν με την εφαρμογή, να αναζητούν και να κλείνουν ραντεβού, ενώ οι ιατροί μπορούν να διαχειρίζονται τα ραντεβού και να αλληλεπιδρούν με τους ασθενείς τους.

2.3.3 Thymeleaf

Για την άμεση επαφή των χρηστών με την εφαρμογή επιλέχθηκε το εργαλείο Thymeleaf. Το Thymeleaf είναι ένα παραγωγικό και ευέλικτο template engine που επιτρέπει τη συνεργασία με το Spring Boot framework για την χειρισμό της δυναμικής απεικόνισης περιεχομένου στη διεπαφή χρήστη. Το Thymeleaf επιτρέπει την άριστη ένταξη με το backend και το frontend, δίνοντας τη δυνατότητα άμεσης πρόσβασης σε αντικείμενα Java και σε κομμάτια Spring για τη σύνδεση με τον χρήστη. Αυτή η συμβατότητα βοηθά στην επίτευξη συνοχής στην ανάπτυξη της εφαρμογής, καθώς οι προγραμματιστές μπορούν να εργαστούν σε ένα ολοκληρωμένο περιβάλλον ανάπτυξης. Επιπρόσθετα παρέχει έναν ευανάγνωστο και φιλικό προς τον προγραμματιστή συντακτικό κώδικα το οποίο είναι πλήρως συμβατό και με τη συγγραφή της HTML, ενώ ταυτόχρονα παρέχει τη δυνατότητα χρήσης εργαλείων προγραμματισμού όπως είναι οι επαναλήψεις (loops) και οι συνθήκες (conditionals).

2.3.4 H2 Database

Για τις δοκιμές και την ανάπτυξη ενός τοπικού περιβάλλοντος, χρησιμοποιήθηκε η βάση δεδομένων H2. Η H2 είναι μια ελαφριά, ενσωματωμένη βάση δεδομένων που μπορεί να τρέξει εντός της εφαρμογής Java. Με την χρήση της H2, δόθηκε η δυνατότητα να δημιουργηθούν και να διαχειριστούν οι απαραίτητοι πίνακες δεδομένων της εφαρμογής.

2.3.5 Docker container

Προκειμένου να επιβεβαιωθεί πως η εφαρμογή μπορεί να εκτελεστεί σε οποιοδήποτε περιβάλλον χωρίς την ανάγκη για προηγούμενη εγκατάσταση και ρύθμιση των απαιτούμενων εξαρτήσεων, επιλέχθηκε η εργαλειοθήκη Docker για την δημιουργία δύο container από τα οποία το ένα θα χρησιμοποιούσε τη βάση δεδομένων H2 για τις τοπικές δοκιμές, ενώ το δεύτερο θα είχε συνδεδεμένη την βάση δεδομένων Aurora της AWS και πιο συγκεκριμένα την έκδοση που είναι συμβατή με την MySQL.

Πρόσθετα, η χρήση των Docker containers επιτρέπει την ευκολότερη και αποτελεσματικότερη διαχείριση της εφαρμογής και των πόρων της. Κάθε container εκτελείται απομονωμένα από τα υπόλοιπα, επιτρέποντας έτσι την παράλληλη εκτέλεση πολλαπλών αντιγράφων της εφαρμογής σε διάφορα περιβάλλοντα. Αυτό είναι ιδιαίτερα χρήσιμο για την ανάπτυξη, τον έλεγχο και την αποσφαλμάτωση της εφαρμογής σε διάφορες συνθήκες χωρίς να επηρεάζεται η σταθερότητα του παραγωγικού περιβάλλοντος.

2.3.6 Υπηρεσίες AWS (Amazon Web Services)

Το AWS Aurora, το Elastic Container Registry και το Elastic Container Service επιλέχθηκαν για την υλοποίηση της εφαρμογής λόγω των πλεονεκτημάτων που προσφέρουν. Το AWS Aurora είναι μια διαχειριζόμενη υπηρεσία βάσης δεδομένων που προσφέρει υψηλή απόδοση, αυξημένη ανθεκτικότητα και αυτόματη κλιμάκωση, επιτρέποντας την αποθήκευση και την ανάκτηση δεδομένων με ασφάλεια και αποτελεσματικότητα.

Το Elastic Container Registry (ECR) είναι ένας πλήρως διαχειριζόμενος χώρος αποθήκευσης εικόνων Docker που επιτρέπει την εύκολη αποθήκευση, διαχείριση και διανομή των εικόνων των εφαρμογών που δημιουργήθηκαν. Παρέχει υψηλή διαθεσιμότητα, ασφάλεια και κλιμακωσιμότητα, καθιστώντας την ιδανική λύση για την αποθήκευση και κοινή χρήση αυτών των Docker εικόνων.

Το Elastic Container Service (ECS) είναι μια πλήρως διαχειριζόμενη υπηρεσία που επιτρέπει την εκτέλεση και την διαχείριση εφαρμογών σε περιβάλλοντα βασισμένα σε containers. Με το ECS, καθίσταται δυνατή η διαμόρφωση και η κλιμάκωση των containers, ενώ ταυτόχρονα διαχειρίζεται τον κύκλο ζωής των εφαρμογών/ containers. Παρέχει αυτόματη ενσωμάτωση με άλλες υπηρεσίες του AWS και επιτρέπει την ευελιξία και την κλιμάκωση των εφαρμογών ανάλογα με τις ανάγκες.

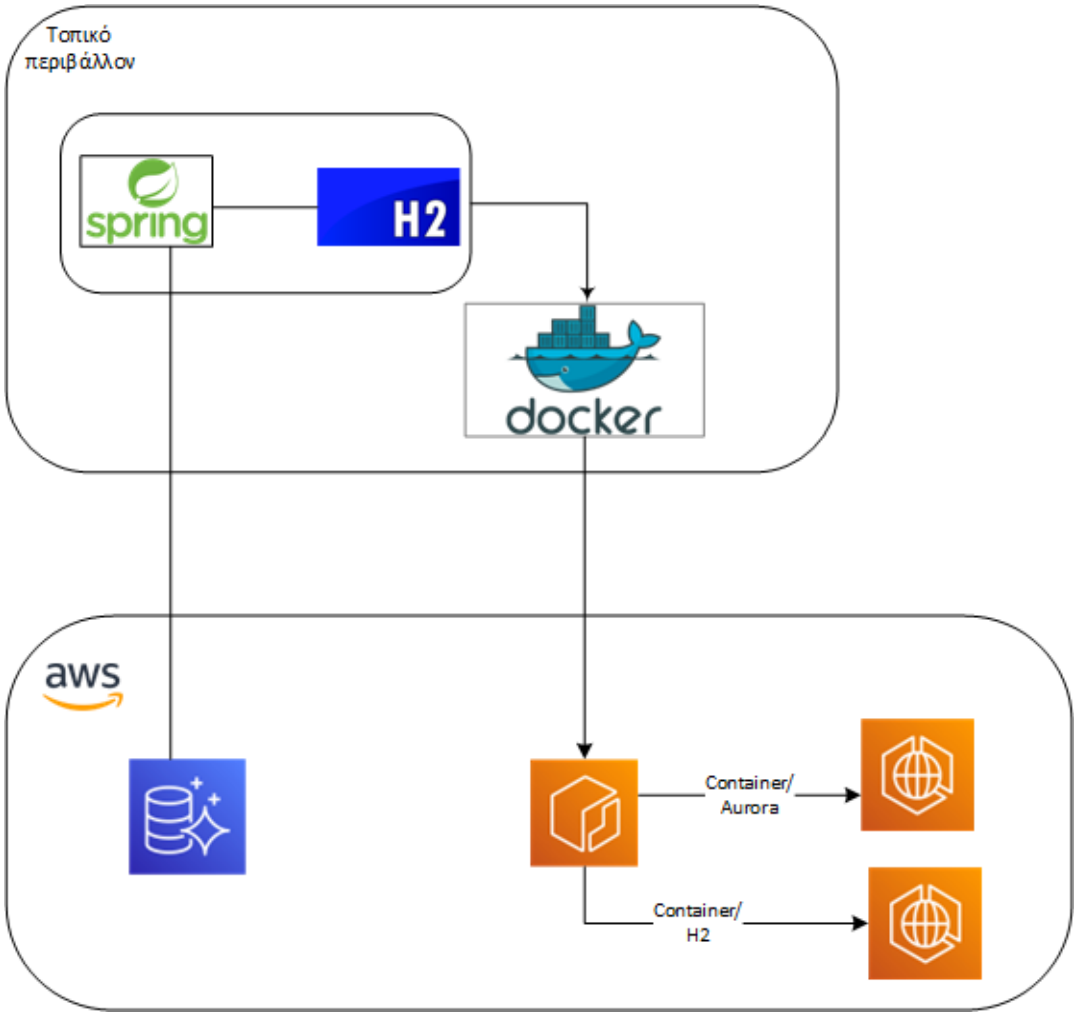
Ο λόγος που χρησιμοποιήθηκαν τα παραπάνω εργαλεία είναι η αξιοποίηση των πλεονεκτημάτων του cloud computing και η διευκόλυνση της διαδικασίας ανάπτυξης, εκτέλεσης και διαχείρισης της εφαρμογής. Η χρήση τους επιτρέπει την αποκλειστική επικέντρωση στην ανάπτυξη της εφαρμογής, ενώ αναλαμβάνουν την φροντίδα για την υποδομή και τις λειτουργικές πτυχές της.

2.4 Αρχιτεκτονική

Η αρχιτεκτονική της εφαρμογής, όπως προαναφέρθηκε, βασίστηκε σε ένα μονολιθικό μοντέλο. Στο μονολιθικό μοντέλο, όλα τα στοιχεία της εφαρμογής εκτελούνται σε έναν μοναδικό διακομιστή. Οι συναρτήσεις της εφαρμογής υλοποιούνται ως μέθοδοι εντός του ίδιου του κώδικα. Αυτό επιτρέπει την απλή ανάπτυξη, τη συντήρηση και την επέκταση της εφαρμογής.

Το μονολιθικό μοντέλο επιλέχθηκε έτσι ώστε η υλοποίηση της εφαρμογής να είναι ακριβής στις λειτουργίες και να μην έχει περιττά επίπεδα πολυπλοκότητας έτσι ώστε να είναι συντηρήσιμη. Ωστόσο, το σύστημα είναι ευέλικτο και είναι εφικτή η μετάβαση σε μια πιο διαμορφωμένη αρχιτεκτονική, όπως η αρχιτεκτονική των μικρών υπηρεσιών (microservices) αν το μέγεθος και οι απαιτήσεις του συστήματος αυξηθούν στο μέλλον.

Με αυτόν τον τρόπο, η εφαρμογή ακολουθεί μια μεθοδολογία ανάπτυξης που περιλαμβάνει ανάλυση απαιτήσεων, σχεδίαση αρχιτεκτονικής, υλοποίηση και δοκιμή. Η τεχνολογία Spring Boot, η H2 Database, τα REST APIs, οι υπηρεσίες AWS και η μονολιθική αρχιτεκτονική χρησιμοποιήθηκαν για την ανάπτυξη και την εκτέλεση της εφαρμογής.

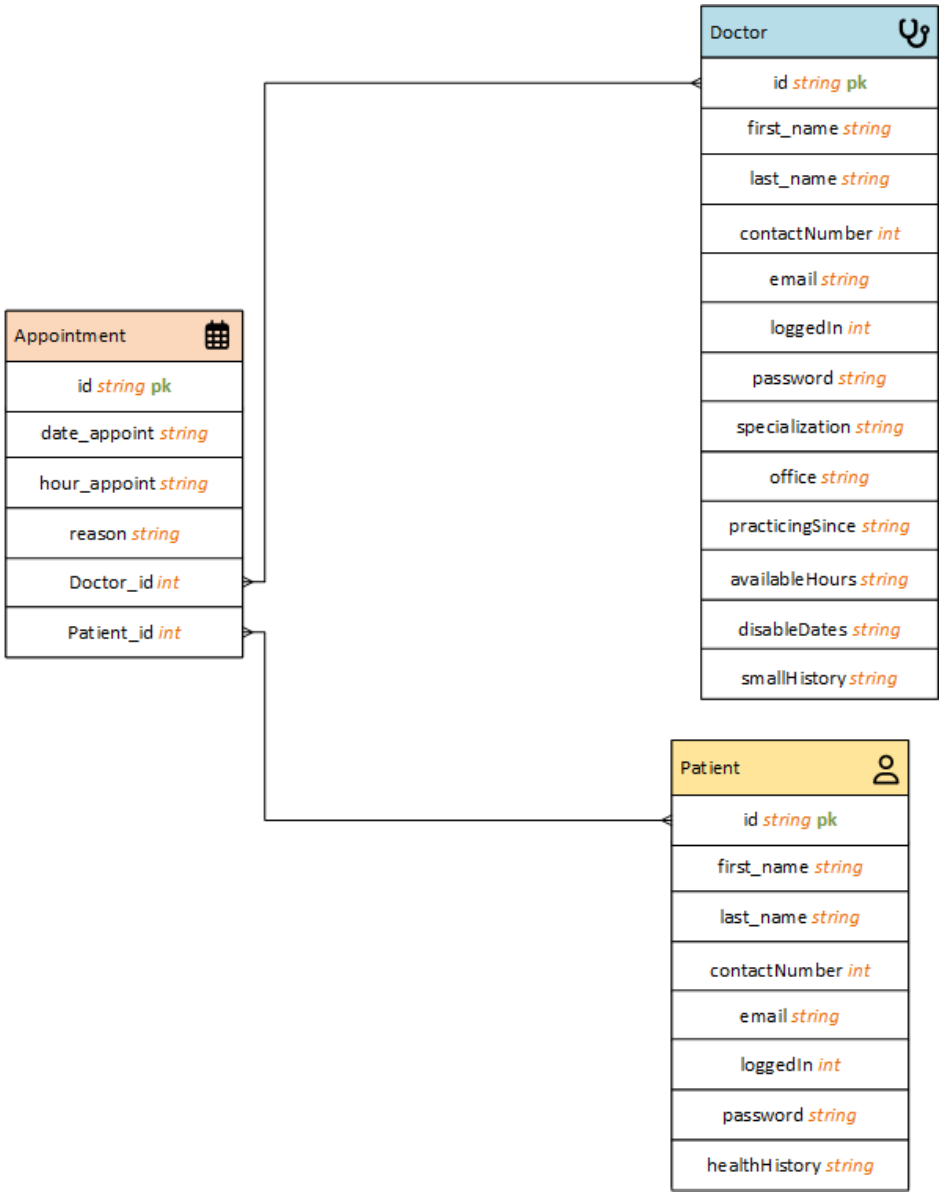


Σχεδιάγραμμα 2: Αρχιτεκτονική της εφαρμογής.

3. Υλοποίηση

3.1 Βάση δεδομένων

Σχεδιάζοντας την εφαρμογή δόθηκε ιδιαίτερη έμφαση στην ασφάλεια, επεκτασιμότητα και αξιοπιστία της εφαρμογής. Έτσι η βάση δεδομένων της εφαρμογής έχει τη παρακάτω μορφή:



Σχεδιάγραμμα 3: Πίνακες βάσης δεδομένων.

Το παραπάνω σχεσιακό μοντέλο υλοποιήθηκε με στόχο την αποθήκευση και πρόσβαση των ιατρών και των ασθενών στο σύστημα των ραντεβού. Από το σχεδιάγραμμα φαίνεται πως οι ασθενείς της βάσης μπορούν να έχουν από κανένα μέχρι πολλαπλά ραντεβού, όπως και οι ιατροί. Η σύνδεση των ιατρών και των ασθενών έγινε μέσω του πίνακα των ραντεβού.

Σημαντική έμφαση δίνεται στη κρυπτογράφηση των κωδικών πρόσβαση με το εργαλείο Base64 το οποίο παίρνει μια σειρά από γράμματα, αριθμούς και σύμβολα και κάθε τρία bytes (24 bits) δεδομένων που λαμβάνει τα αναπαριστά από τέσσερις χαρακτήρες μορφής ASCII.

Εξίσου σημαντική ήταν και η χρήση της τιμής *loggedin* για λόγους απλότητας και ασφάλειας. Με αυτό το τρόπο εξακριβώθηκε αν οι χρήστες οι είναι συνδεδεμένοι ή όχι, ανεξάρτητα από το αν ήταν ιατροί ή ασθενείς.

3.2 Περιβάλλον λογισμικού

Για την ανάπτυξη του λογισμικού χρησιμοποιήθηκε το IntelliJ IDEA, ένα περιβάλλον ανάπτυξης λογισμικού (IDE) της JetBrains και το λειτουργικό σύστημα των Windows 10 της Microsoft.

Παρακάτω γίνεται παρουσίαση των πακέτων που γράφτηκαν και χρησιμοποιήθηκαν για την υλοποίηση της εφαρμογής στο περιβάλλον του Intellij IDEA.

3.2.1 Model

Το Model είναι ένα πακέτο κλάσεων το οποίο δημιουργήθηκε, σύμφωνα με το μοντέλο MVC, προκειμένου να εφαρμοστεί η μεθοδολογία επικοινωνίας ORM (Objective Relational Mapping) . Ο λόγος που έγινε αυτή η υλοποίηση είναι για να υπάρχει η ελευθερία χρήσης οποιασδήποτε σχεσιακής βάσης δεδομένων, είτε είναι η MySQL είτε η H2 είτε οποιαδήποτε άλλη. Πιο αναλυτικά, μέσω του εργαλείου διαχείρισης πακέτων Maven και του αρχείου application.properties, το οποίο δημιουργήθηκε αυτόματα από το οικοσύστημα του spring boot, συνδέθηκε η βάση δεδομένων, με την υπόλοιπη εφαρμογή και μέσω αυτού και του ORM ήταν δυνατή η "αυτόματη" δημιουργία πινάκων στη βάση δεδομένων.

3.2.2 Repositories

Τα Repositories είναι ένα πακέτο κλάσεων, συγκεκριμένα τύπου interfaces, τα οποία έχουν ως σκοπό να ελέγξουν τη σωστή λήψη δεδομένων από τη κεντρική βάση δεδομένων. Έτσι δημιουργήθηκαν αυτές οι κλάσεις και αποτέλεσαν σημαντικό κομμάτι της εφαρμογής γιατί μέσα από αυτά δόθηκε η δυνατότητα να οριστούν και να εκτελεστούν μέθοδοι οι οποίες αλληλοεπιδρούν με τη βάση δεδομένων χωρίς να χρησιμοποιηθούν αποκλειστικά συνθήκες SQL, οι οποίες θα μπορούσαν σε βάθος χρόνου να αποτελέσουν ‘’αδυναμίες’’ στην ασφάλεια της εφαρμογής.

3.2.3 Services

Κάθε εφαρμογή έχει κάποιες λειτουργίες με τις οποίες επεξεργάζεται τα δεδομένα της και αλληλεπιδρά με τη βάση δεδομένων. Για αυτόν τον λόγο δημιουργήθηκε το πακέτο Services το οποίο αποτέλεσε ένα σύνολο μεθόδων οι οποίες επεξεργάστηκαν τα δεδομένα που εισήχθησαν από το πακέτο Controllers και συνδέθηκαν έμμεσα και άμεσα με τα υπόλοιπα πακέτα.

3.2.3 Controllers

Σύμφωνα με το μοντέλο MVC, ο Controller είναι μία σύνθεση από κλάσεις και μεθόδους όπου μέσα από αυτές ο χρήστης ή μια άλλη εφαρμογή αλληλεπιδρά με την παρούσα εφαρμογή μέσα από τις μεθόδους επικοινωνίας HTTP (Hypertext Transfer Protocol) . Μετά την ολοκλήρωση της αλληλεπίδρασης ο Controller λαμβάνει τα δεδομένα και τα παρέχει στα Services, Repositories, Interfaces ή οτιδήποτε άλλο έχει δυνατότητα να επικοινωνήσει και στη συνέχεια επιστρέφει ένα αποτέλεσμα στο View, το οποίο στη περίπτωση αυτής της εφαρμογής είναι το Thymeleaf ή επιστρέφει μια απάντηση σε μορφή JSON.

3.3 Μεταφορά στο Νέφος

Η μετατροπή της συγκεκριμένης εφαρμογής σε container και το ανέβασμα στο νέφος του AWS είναι ένα σημαντικό βήμα που επιτρέπει την εκμετάλλευση των πλεονεκτημάτων της μοντέρνας αρχιτεκτονικής και του cloud computing. Η χρήση containers, όπως το Docker, επιτρέπει να πακεταριστεί η εφαρμογή με όλες τις εξαρτήσεις της και να εκτελεστεί σε διαφορετικά περιβάλλοντα, εξασφαλίζοντας την ομαλή λειτουργία της ανεξάρτητα από τον υπολογιστικό πόρο που διατίθεται.

Τα βασικοί βήματα για τη μετατροπή της εφαρμογής σε container είναι:

- Προετοιμασία του Dockerfile: Το Dockerfile είναι ένα αρχείο κειμένου που περιγράφει τα βήματα για το πώς να δημιουργηθεί το container. Σε αυτό το στάδιο, καθορίστηκαν οι βάσεις (base images) που χρησιμοποιήθηκαν, εγκαταστάθηκαν οι απαραίτητες εξαρτήσεις και αντιγράφηκαν τα απαραίτητα αρχεία για την εκτέλεση της εφαρμογής.

- Υλοποίηση του container: Με τη χρήση του Docker CLI (Command Line Interface), δημιουργήθηκε το container από το Dockerfile. Αυτό δημιουργεί ένα εικονικό περιβάλλον που περιλαμβάνει την εφαρμογή και όλες τις απαραίτητες εξαρτήσεις.

- Δοκιμή του container τοπικά: Αφού δημιουργήθηκε το container, εκτελέστηκε τοπικά στον υπολογιστή και διαπιστώθηκε ότι η εφαρμογή λειτουργεί σωστά σε αυτό το περιβάλλον.

- Ανέβασμα στο AWS ECR (Elastic Container Registry): Με τη χρήση του Docker CLI, μεταφέρθηκε το container που υλοποιήθηκε, στο ECR ώστε να είναι διαθέσιμο στο νέφος του AWS.

- Εκτέλεση του container στο AWS ECS (Elastic Container Service): Εδώ, ορίστηκαν οι παράμετροι του cluster, του task definition και της service που χρειάζεται η εφαρμογή, και η υπηρεσία τοποθετεί το container στους κατάλληλους υπολογιστικούς πόρους.

Μετά από αυτή τη διαδικασία, η εφαρμογή είναι πλέον διαθέσιμη στο νέφος του AWS και μπορεί να εκτελεστεί και να κλιμακωθεί (scale-up) εάν χρειαστεί.

4. Παρουσίαση της υλοποίησης

4.1 Παρουσίαση κεντρικής υπηρεσίας και λειτουργιών

Η διεπαφή του χρήστη (UI - User Interface) αποτελεί τον σημαντικό και πρώτο τρόπο με τον οποίο οι χρήστες αλληλοεπιδρούν με μια εφαρμογή. Είναι η πύλη που τους επιτρέπει να εκτελούν λειτουργίες, να διαχειρίζονται πληροφορίες και να αλληλοεπιδρούν με τα δεδομένα της εφαρμογής. Αναλαμβάνει τον ρόλο της γέφυρας μεταξύ του ανθρώπινου χρήστη και της τεχνολογίας, παρέχοντας έναν τρόπο να αξιοποιήσουν τη λειτουργικότητα της εφαρμογής.

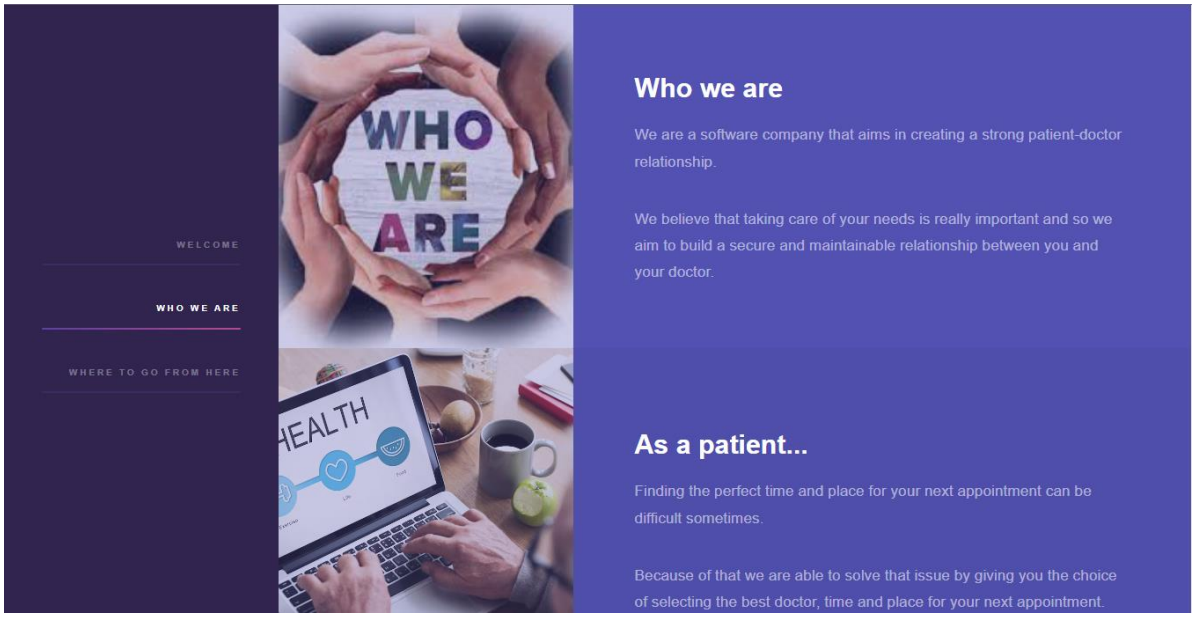
Κατά τη διάρκεια αυτής της παρουσίασης βλέπουμε πώς οι χρήστες αλληλοεπιδρούν με την εφαρμογή, πώς πραγματοποιούν ενέργειες και πώς αυτές οι ενέργειες επηρεάζουν τη λειτουργία της.

Σε αυτήν την παρουσίαση, εξετάζεται ο τρόπος με τον οποίο η διεπαφή του χρήστη σχεδιάστηκε για να είναι χρήσιμη, ευανάγνωστη και φιλική προς τον χρήστη. Βλέπουμε επίσης ποιες λειτουργίες και δυνατότητες προσφέρει στους χρήστες και πώς αυτές συμβάλλουν στον επιτυχημένο σκοπό της εφαρμογής. Αναφορικά με την έναρξη της εφαρμογής δημιουργούνται αυτόματα καταγραφές στη βάση δεδομένων μέσα από τη Java.

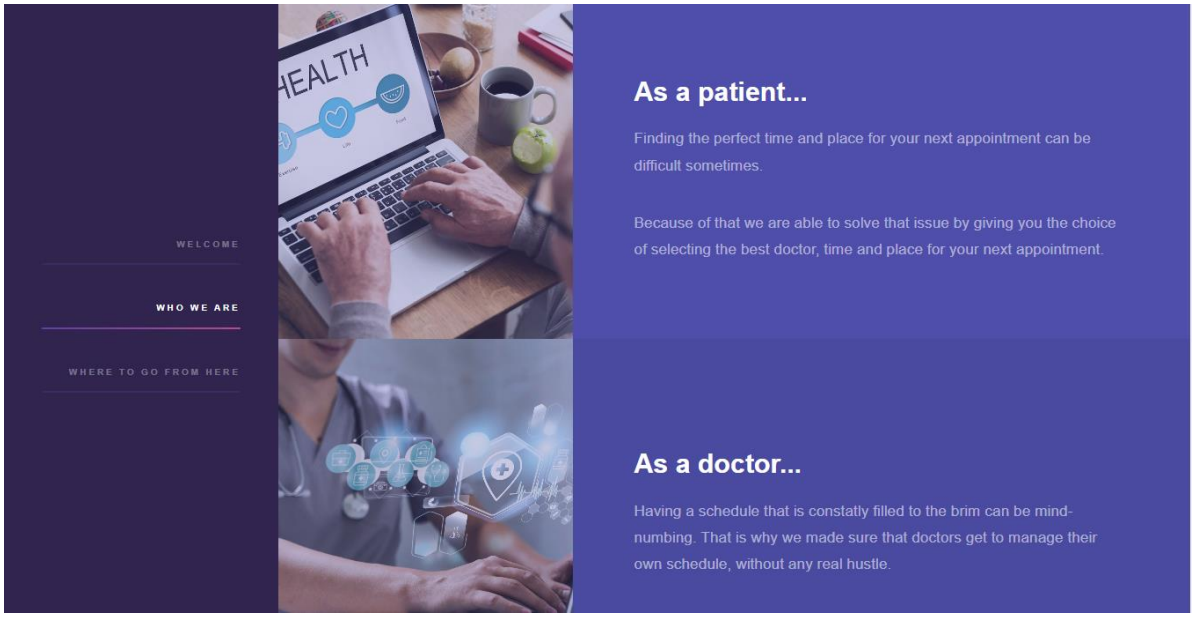
4.1.1 Αρχική σελίδα της εφαρμογής

Παρουσιάζεται η αρχική σελίδα της εφαρμογής. Υπάρχει ένα μενού επιλογών για την πλοήγηση στη σελίδα. Ο επισκέπτης της σελίδας, κάνοντας τις αντίστοιχες επιλογές από το μενού, μπορεί να μεταβεί σε περιοχές που αφορούν το καλωσόρισμα του στη σελίδα (*Welcome*), εισαγωγικές πληροφορίες για την εφαρμογή (*Who we are*) και έχει τη δυνατότητα να δημιουργήσει λογαριασμό (*Where to go from here*).

Εικόνα 1: Αρχική σελίδα της εφαρμογής.



Εικόνα 2: Πλοήγηση στην σελίδα της εφαρμογής.

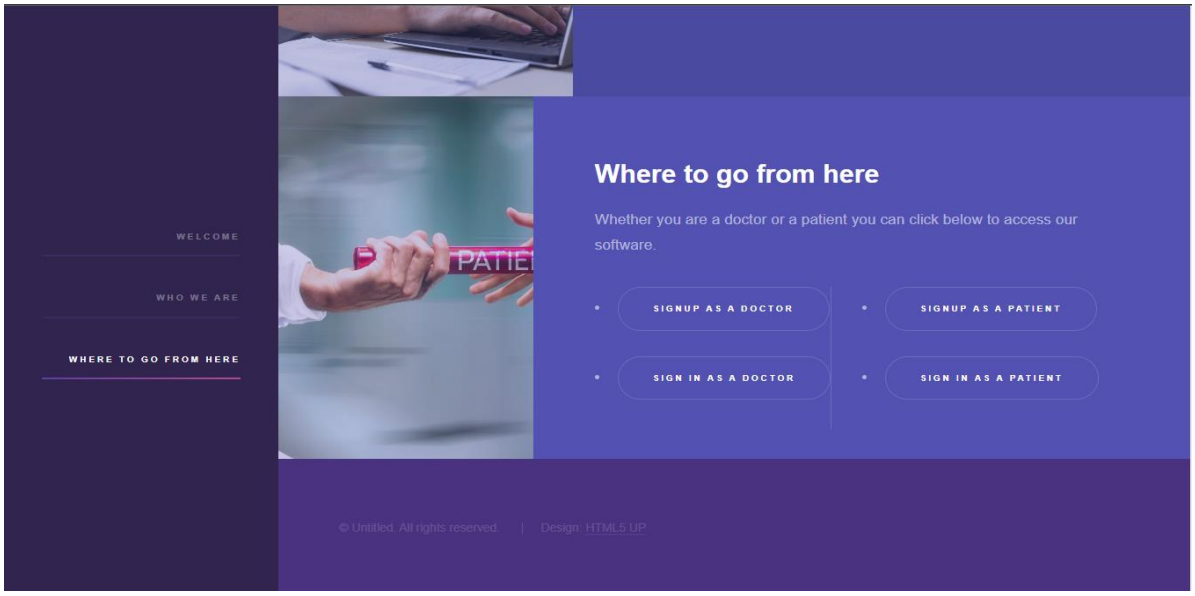


Εικόνα 3: Πλοήγηση στην σελίδα της εφαρμογής.

4.2 Είσοδος Χρήστη στην εφαρμογή

Στην εφαρμογή έχουν πρόσβαση δύο ειδών χρήστες, οι ιατροί και οι ασθενείς, οι οποίοι έχουν τη δυνατότητα να δημιουργήσουν τα αντίστοιχα προφίλ τους, να έχουν πρόσβαση στις καταγραφές των ραντεβού που θα δημιουργηθούν για τον εκάστοτε σκοπό και να μοιράζονται πληροφορίες ιατρικού περιεχομένου.

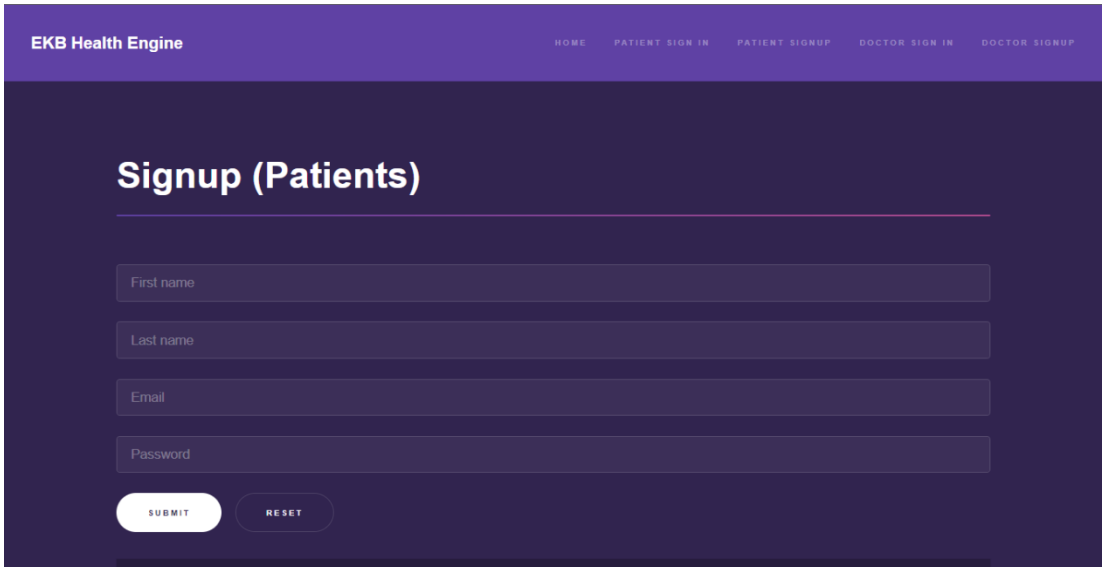
Η διαδικασία δημιουργίας και σύνδεσης λογαριασμού είναι ίδια και στον ιατρό αλλά και στον ασθενή. Οι διαφορές ξεκινούν μετά τη δημιουργία και τη σύνδεση στον εκάστοτε λογαριασμό.



Εικόνα 4: Είσοδος χρήστη στην εφαρμογή.

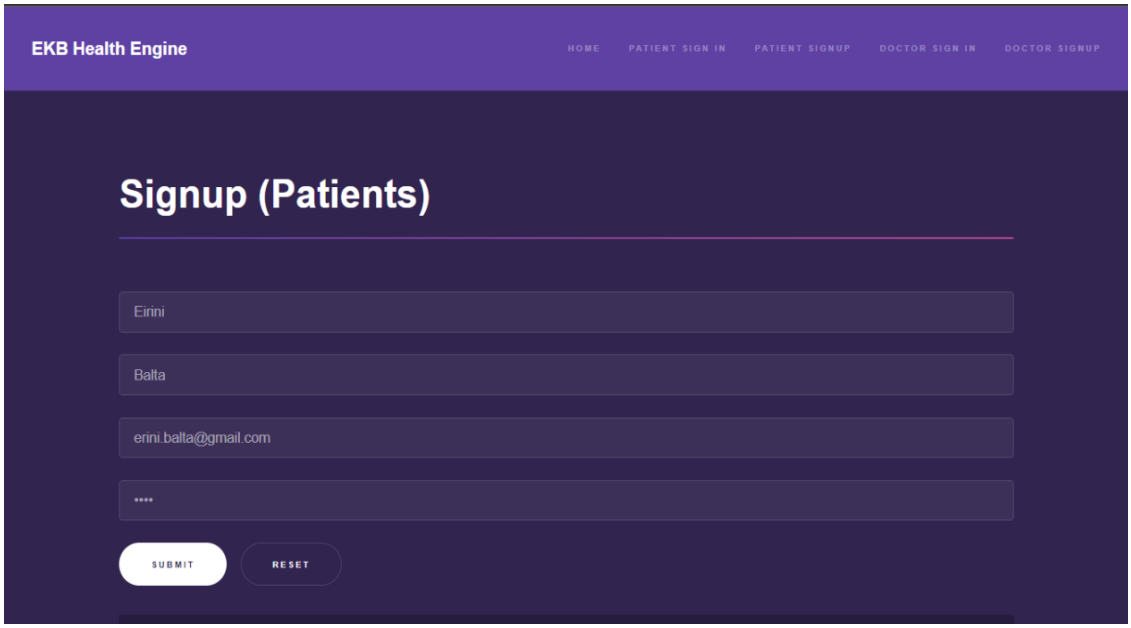
4.2.1 Εγγραφή ασθενή στην εφαρμογή

Ο χρήστης κάνοντας την επιλογή *''Sign up as a patient''* μπορεί να δημιουργήσει λογαριασμό ως ασθενής και του εμφανίζεται η παρακάτω οθόνη.



Εικόνα 5: Δημιουργία λογαριασμού ως ασθενή στην εφαρμογή.

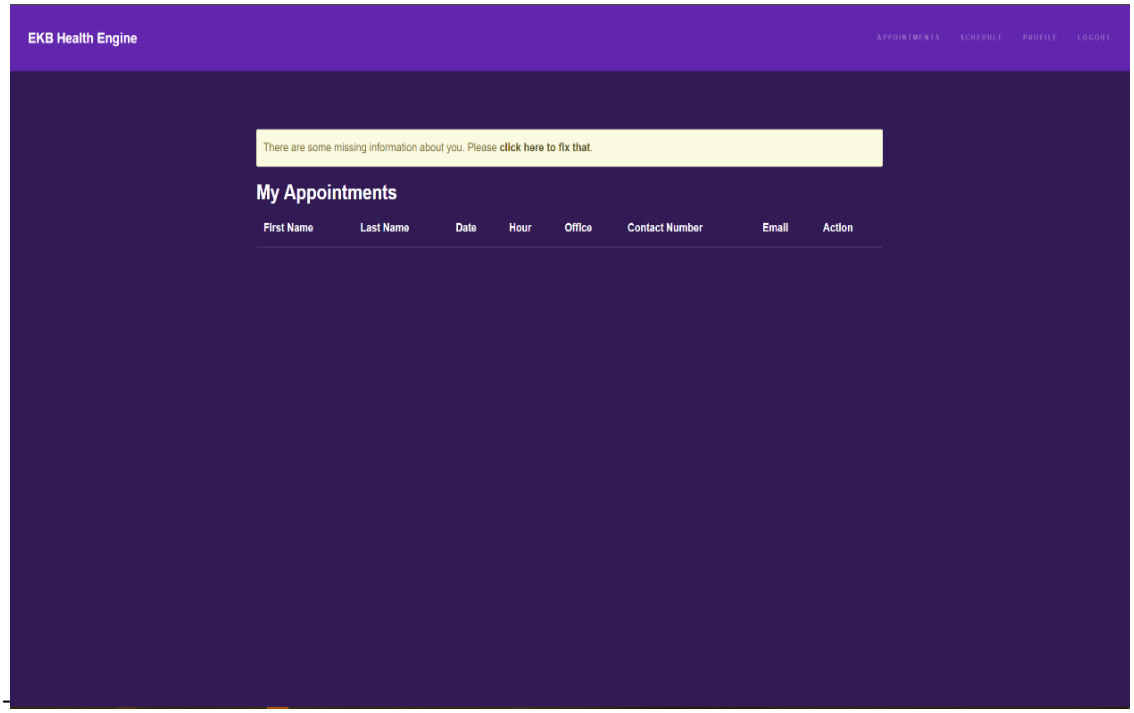
Ο χρήστης πρέπει να συμπληρώσει με τα προσωπικά του στοιχεία τα κατάλληλα πεδία που εμφανίζονται και κάνοντας την επιλογή ‘’*Submit*’’ δημιουργεί το προφίλ του.



Εικόνα 6: Καταχώρηση στοιχείων χρήστη για την δημιουργία προφίλ ασθενή.

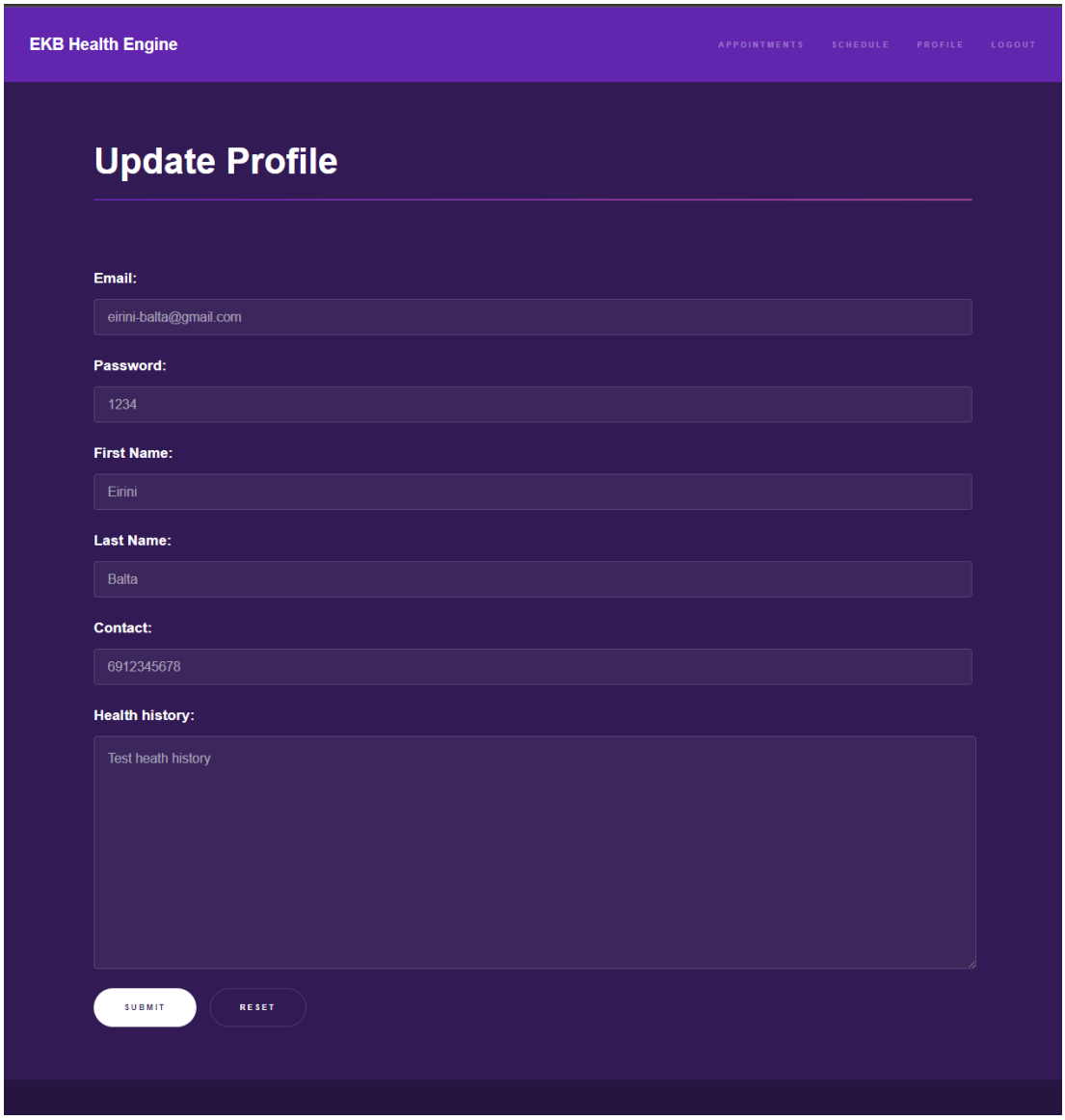
Με την επιτυχή δημιουργία του λογαριασμού του χρήστη ως ασθενής, ο ασθενής-χρήστης μεταφέρεται στην αρχική οθόνη του ασθενή (οι στήλες του πίνακα των ραντεβού είναι: *First Name, Last Name, Date, Hour, Office,Contact Number, Email και Action)*.Εμφανίζεται ένα μήνυμα *‘’There are some missing information about you.**Please click here to fix that’’***, το οποίο θα μείνει στην αρχική οθόνη μέχρι ο ασθενής να το επιλέξει και να συμπληρώσει το προφίλ του με περαιτέρω πληροφορίες, έτσι ώστε ο ιατρός να είναι σε θέση να λάβει τις πληροφορίες που χρειάζεται. Μέχρι να συμπληρώσει το προφίλ του, ο ασθενής δεν μπορεί να κλείσει κάποιο ιατρικό ραντεβού.

Ο χρήστης έχει επίσης και τη δυνατότητα να περιηγηθεί στη πλατφόρμα (*appointments, schedule, profile*) όπως και να αποσυνδεθεί (*logout*) από το λογαριασμό του, από το μενού που βρίσκεται στην επάνω δεξιά μεριά της οθόνης .



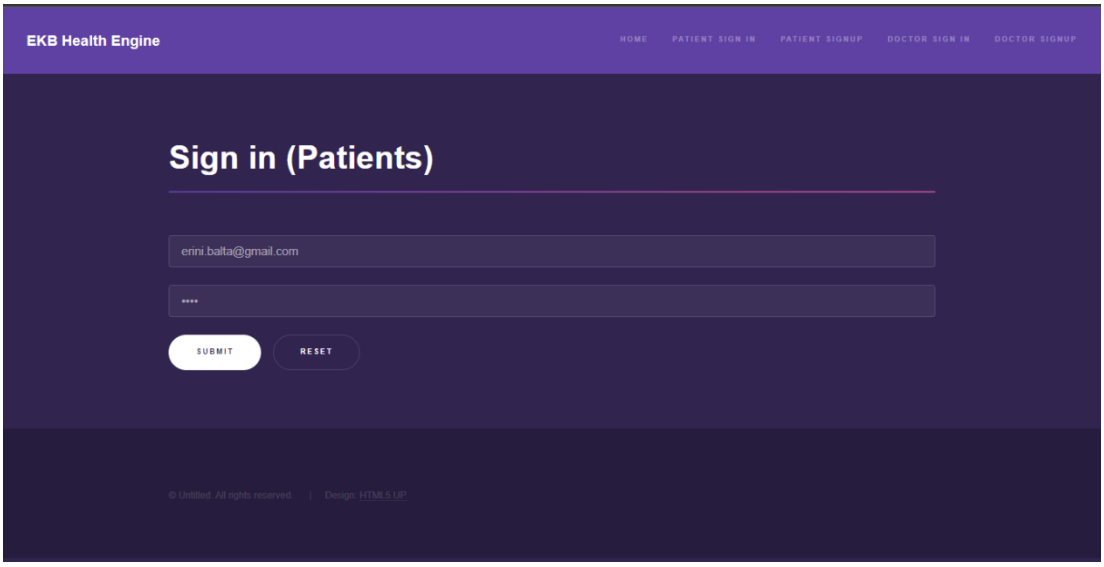
Εικόνα 7: Αρχική οθόνη προφίλ ασθενή και εμφάνιση ειδοποίησης για το προφίλ.

Όταν ο χρήστης επιλέξει να πατήσει πάνω στην ειδοποίηση, θα οδηγηθεί στην οθόνη του προσωπικού του προφίλ και θα πρέπει να καταχωρήσει συμπληρωματικά στοιχεία στα κατάλληλα πεδία της φόρμας, η οποία θα παρέχει στους ιατρούς σημαντικές πληροφορίες όπως ιατρικό ιστορικό ή πληροφορίες σχετικά με την συμπτωματολογία που παρουσιάζει καθώς και τα στοιχεία επικοινωνίας του.



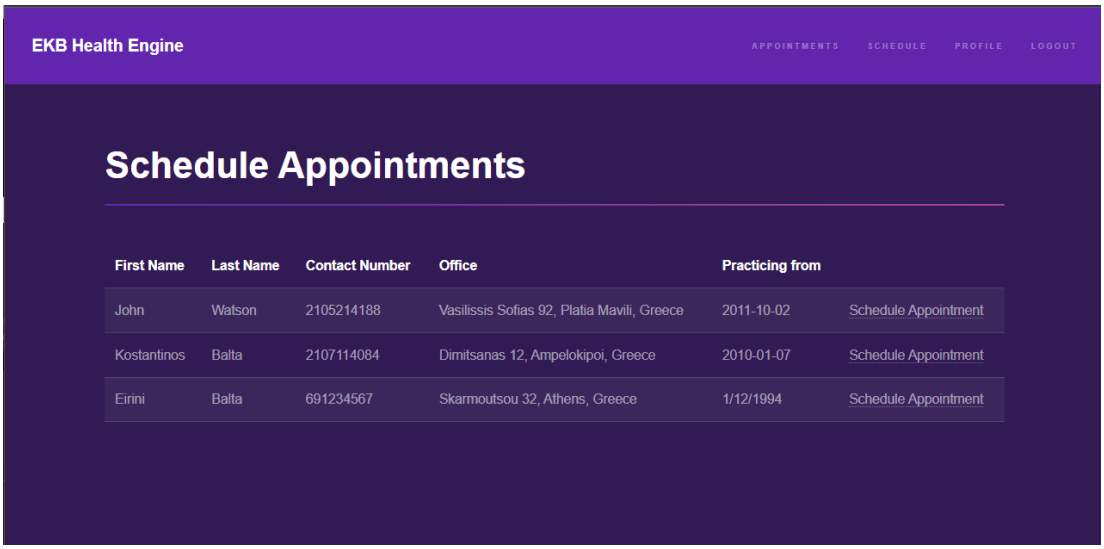
Εικόνα 8: Ανανέωση προφίλ ασθενή με συμπληρωματικά στοιχεία.

Ο χρήστης κάνοντας την επιλογή ‘’*Submit*’’ ανανεώνει το προφίλ του και αποσυνδέεται αυτόματα έτσι ώστε να ενημερωθεί κατάλληλα η βάση δεδομένων. Θα πρέπει να συνδεθεί ξανά στην πλατφόρμα για να κλείσει το ιατρικό ραντεβού του.



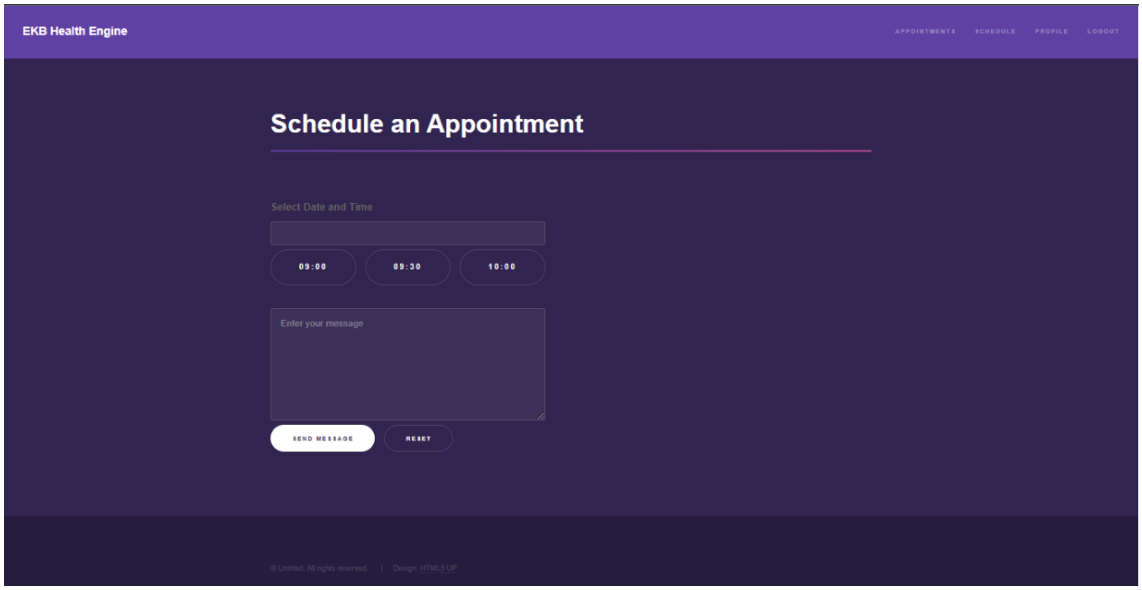
Εικόνα 9: Σύνδεση στο προφίλ ασθενή.

Όταν ο χρήστης συνδεθεί ξανά έχει τη δυνατότητα να επιλέξει το ‘’*Schedule”* από το μενού, που βρίσκεται στην πάνω δεξιά μεριά της οθόνης. Κάνοντας αυτή την επιλογή θα οδηγηθεί στη παρακάτω οθόνη και μπορεί να επιλέξει τον ιατρό που θέλει να κλείσει ραντεβού.

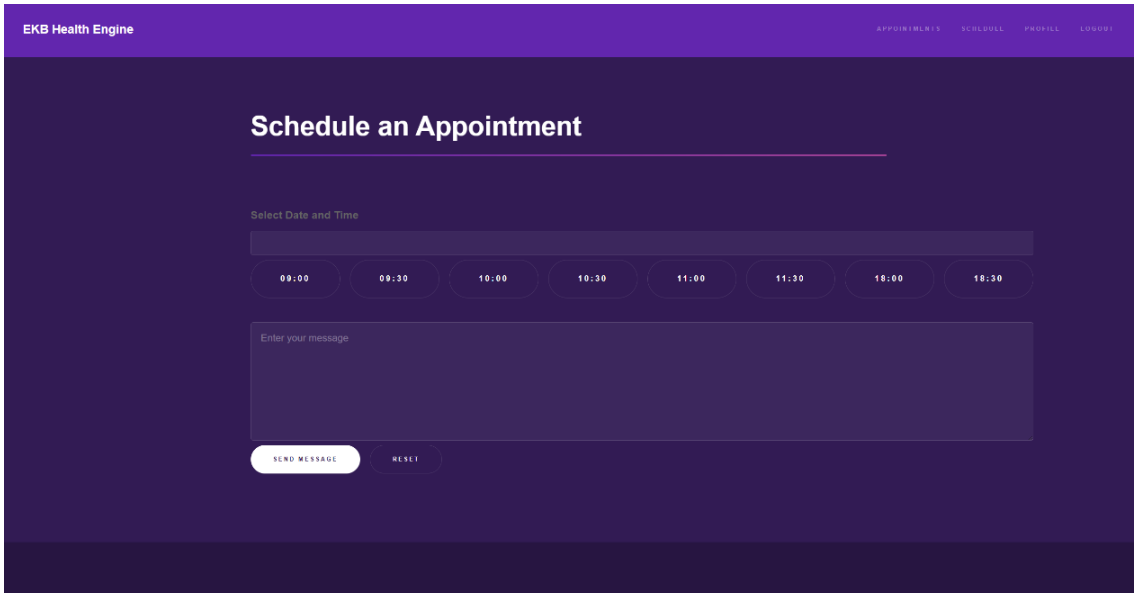


Εικόνα 10: Επιλογή Schedule - εμφάνιση Ιατρών.

Δίπλα από κάθε Ιατρό έχει την επιλογή ‘’*Schedule Appointment”* επιλέγοντας την οδηγείται στις επόμενες οθόνες.



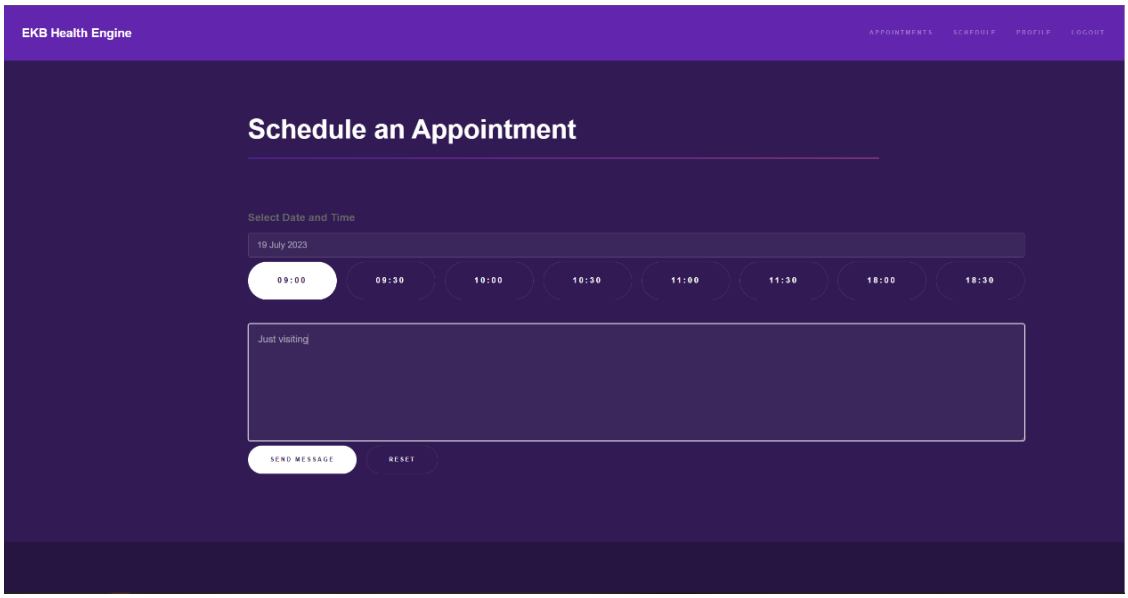
Εικόνα 11: Επιλογή Schedule Appointment στον πρώτο Ιατρό που εμφανίζεται στη λίστα.



Εικόνα 12: Επιλογή Schedule Appointment στον δεύτερο Ιατρό που εμφανίζεται στη λίστα.

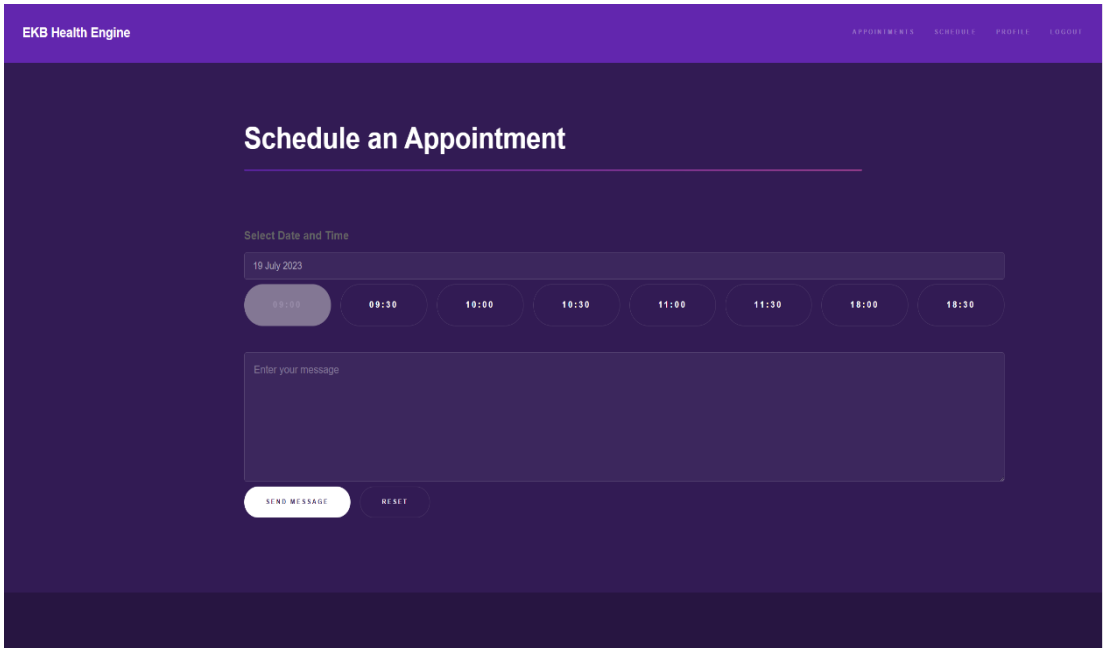
Ο ασθενής μπορεί να επιλέξει την ημερομηνία, την ώρα και να γράψει ένα μήνυμα αναφέροντας τον λόγο για τον οποίο κλείνει το ραντεβού με τον ιατρό και αυτόματα ενημερώνεται το ημερολόγιο του εκάστοτε ιατρού. Η ημερομηνία που θα επιλέξει από το ημερολόγιο αναγράφεται στο

αντίστοιχο πεδίο. Τα κουμπιά που αναγράφονται οι ώρες έχουν χρώμα μωβ και όταν επιλέξει ο ασθενής την ώρα του ραντεβού που θέλει να κλείσει, το κουμπί γίνεται άσπρο.



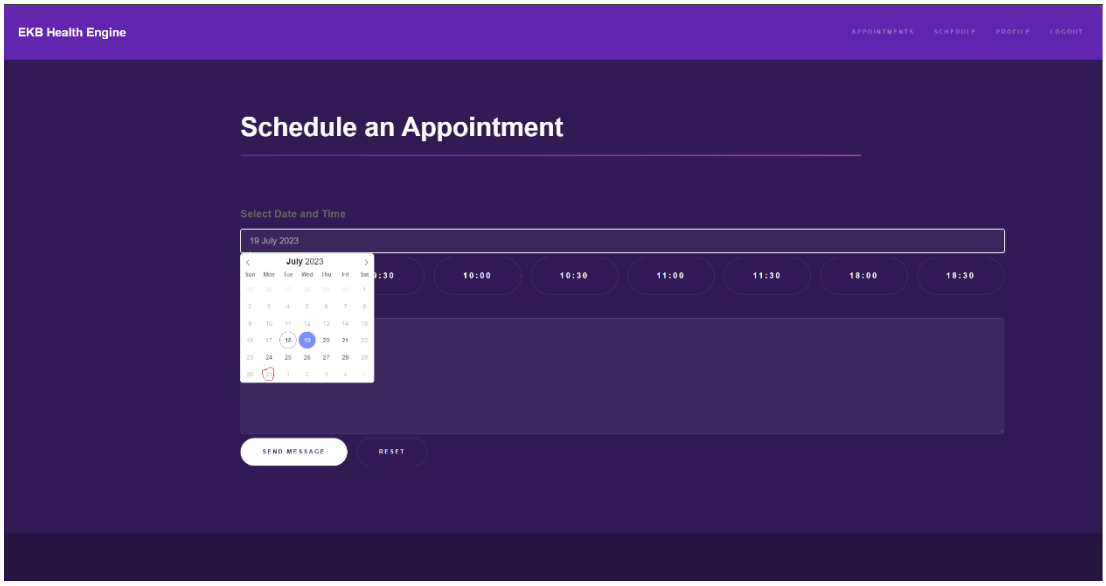
Εικόνα 13: Επιλογή ημερομηνίας και ώρας για το ιατρικό ραντεβού.

Ο ασθενής δε μπορεί να επιλέξει ημερομηνία και ώρα που υπάρχει ήδη δεσμευμένο ραντεβού ή ημερομηνία και ώρα για την οποία ο ιατρός δεν είναι διαθέσιμος. Οι μη διαθέσιμες ώρες ραντεβού εμφανίζονται με γκρι χρώμα (κουμπί) και δεν μπορούν τεχνικά να επιλεχθούν. Επίσης έχει δηλωθεί πως τα ραντεβού μπορούν να γίνουν από Δευτέρα έως Παρασκευή και το ωράριο μπορεί να αλλάξει από ιατρό σε ιατρό.



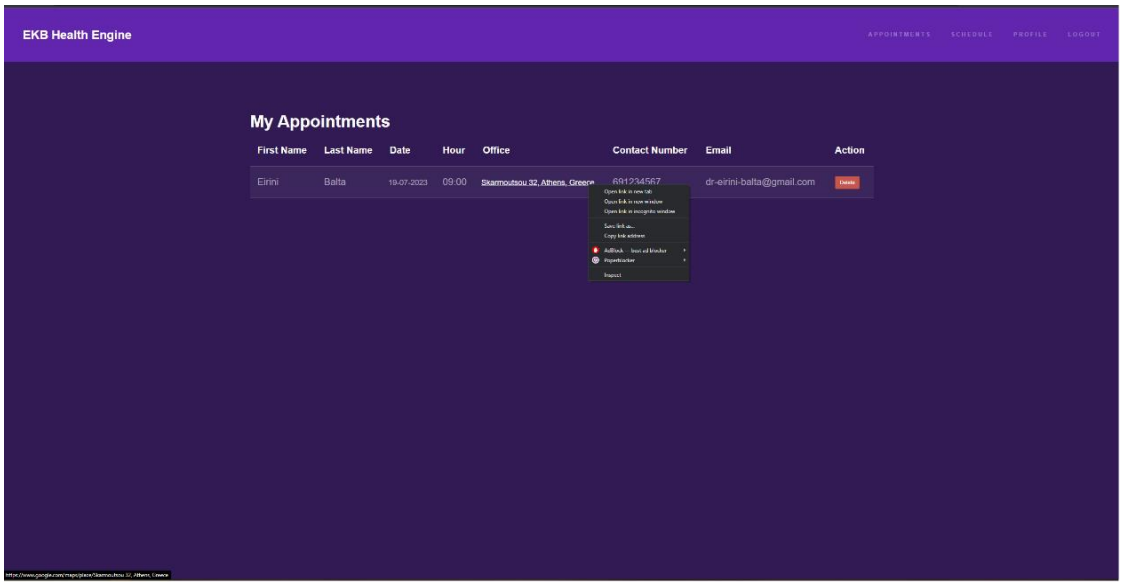
Εικόνα 14: Επιλογή ώρας για το ιατρικό ραντεβού.

Επιπρόσθετα, ο ασθενής δεν έχει δυνατότητα τεχνικά να επιλέξει ραντεβού σε ημερομηνίες που έχουν προηγηθεί από την ημέρα που πραγματοποιεί την καταχώρηση του ραντεβού.

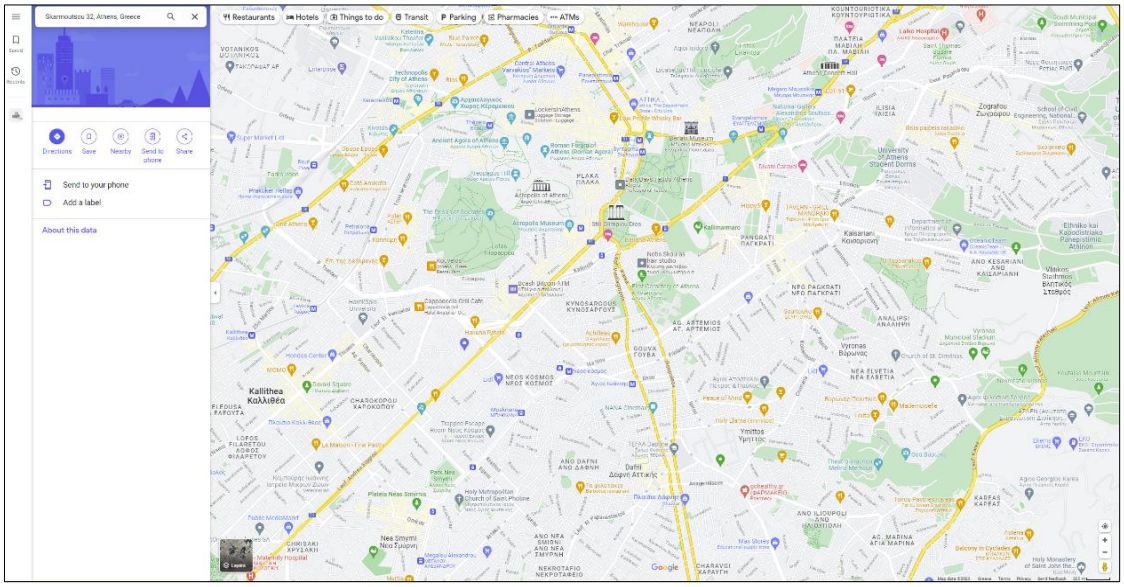


Εικόνα 15: Επιλογή ημερομηνίας για το ιατρικό ραντεβού.

Ο ασθενής έχει την δυνατότητα να δει την τοποθεσία του ιατρείου που θα πραγματοποιήσει το ραντεβού του, με τη βοήθεια του google maps. Στη στήλη *"Office"* εμφανίζονται με hyperlinks οι διευθύνσεις των ιατρείων.

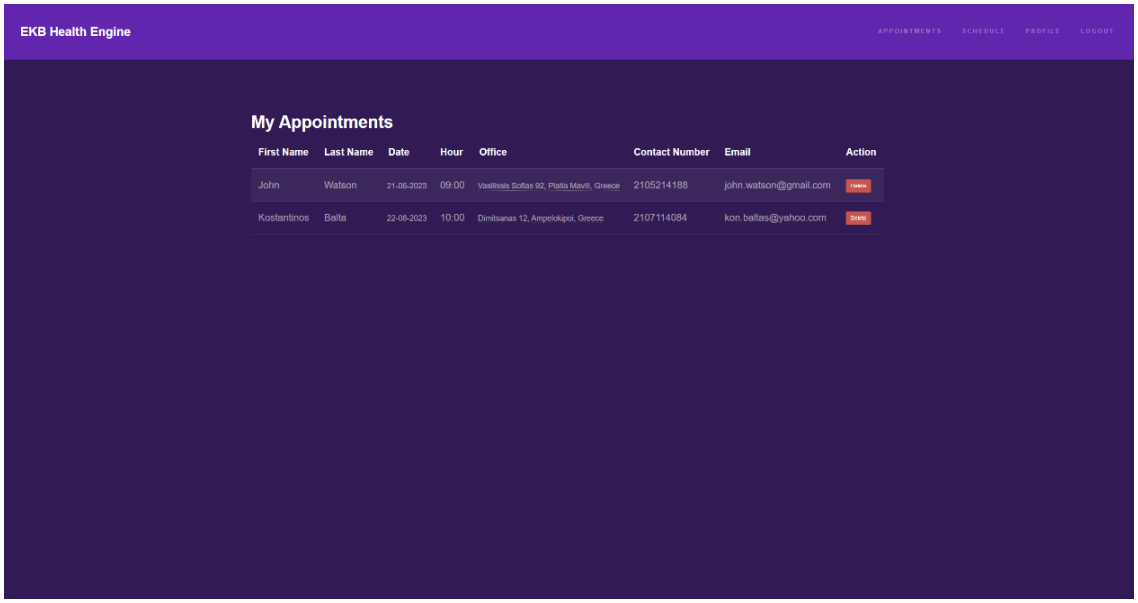


Εικόνα 16: Εμφάνιση διευθύνσεων ιατρείων με hyperlinks.



Εικόνα 17: Εμφάνιση τοποθεσίας ιατρείου με τη βοήθεια του google maps.

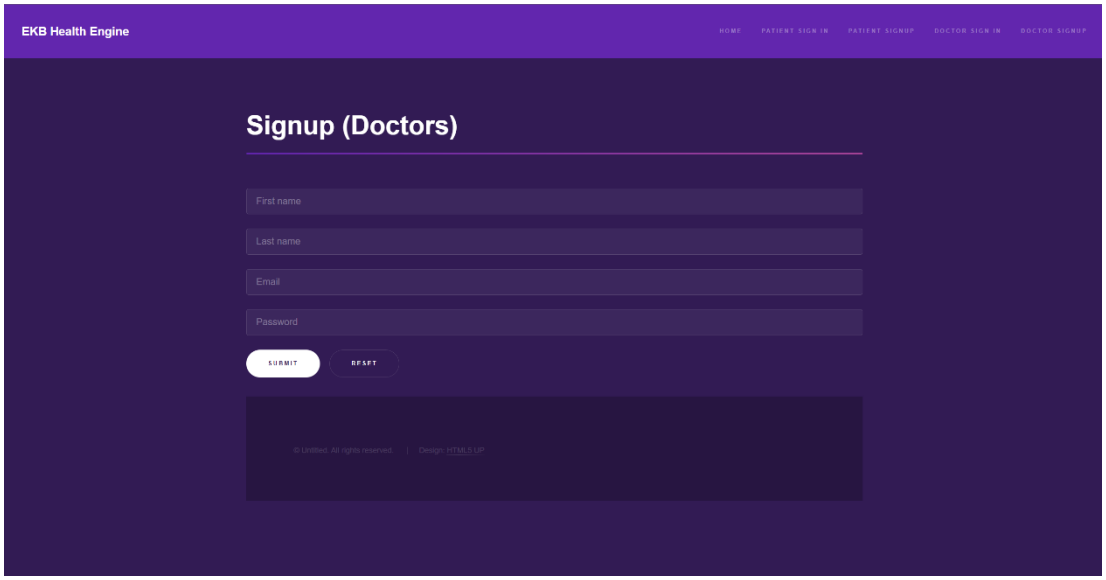
Τέλος ο ασθενής έχει και τη δυνατότητα να προσθέσει κ άλλο ραντεβού ή να διαγράψει κάποιο από αυτά που ήδη έχει στη λίστα επίσκεψης.



Εικόνα 18: Εμφάνιση delete για τα ραντεβού.

4.2.2 Εγγραφή γιατρού στην εφαρμογή

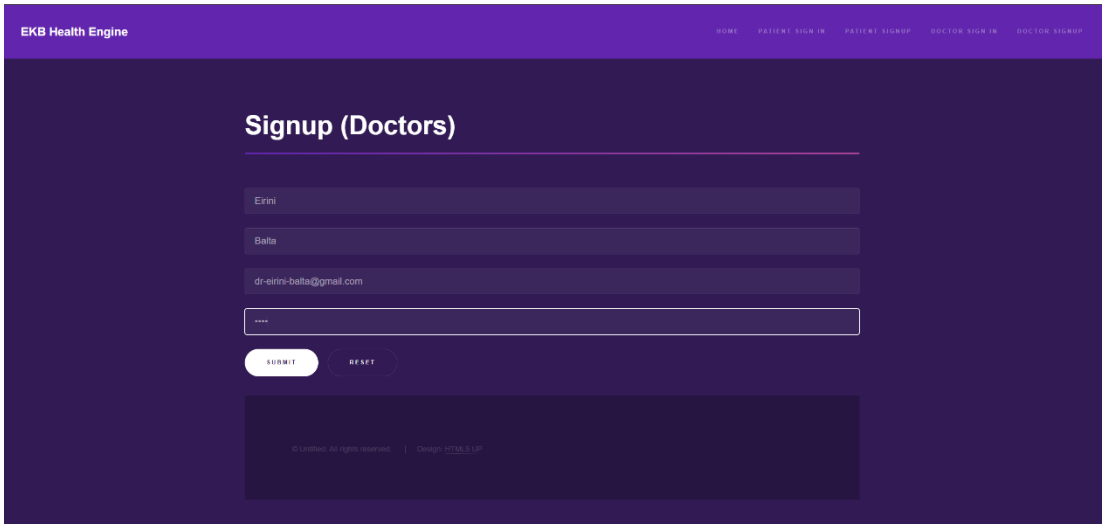
Ο χρήστης κάνοντας την επιλογή *''Sign up as a doctor''* μπορεί να δημιουργήσει λογαριασμό ως γιατρός και του εμφανίζεται η παρακάτω οθόνη.



Εικόνα 19: Δημιουργία λογαριασμού ως γιατρός στην εφαρμογή.

Η φόρμα εγγραφής των ιατρών είναι ίδια με εκείνη των ασθενών, ωστόσο μετά τη δημιουργία του λογαριασμού τους οι δυνατότητές των ιατρών είναι διαφορετικές από αυτές των ασθενών.

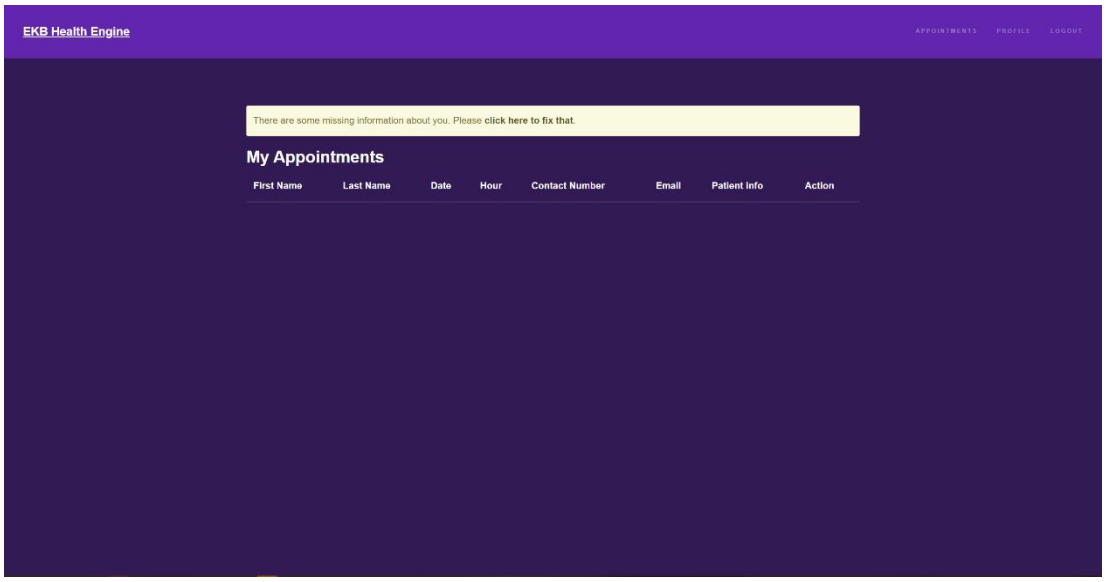
Ο χρήστης πρέπει να συμπληρώσει με τα προσωπικά του στοιχεία τα κατάλληλα πεδία που εμφανίζονται και κάνοντας την επιλογή ‘’*Submit*’’ δημιουργεί το προφίλ του.



Εικόνα 20: Καταχώρηση στοιχείων χρήστη για την δημιουργία προφίλ γιατρού.

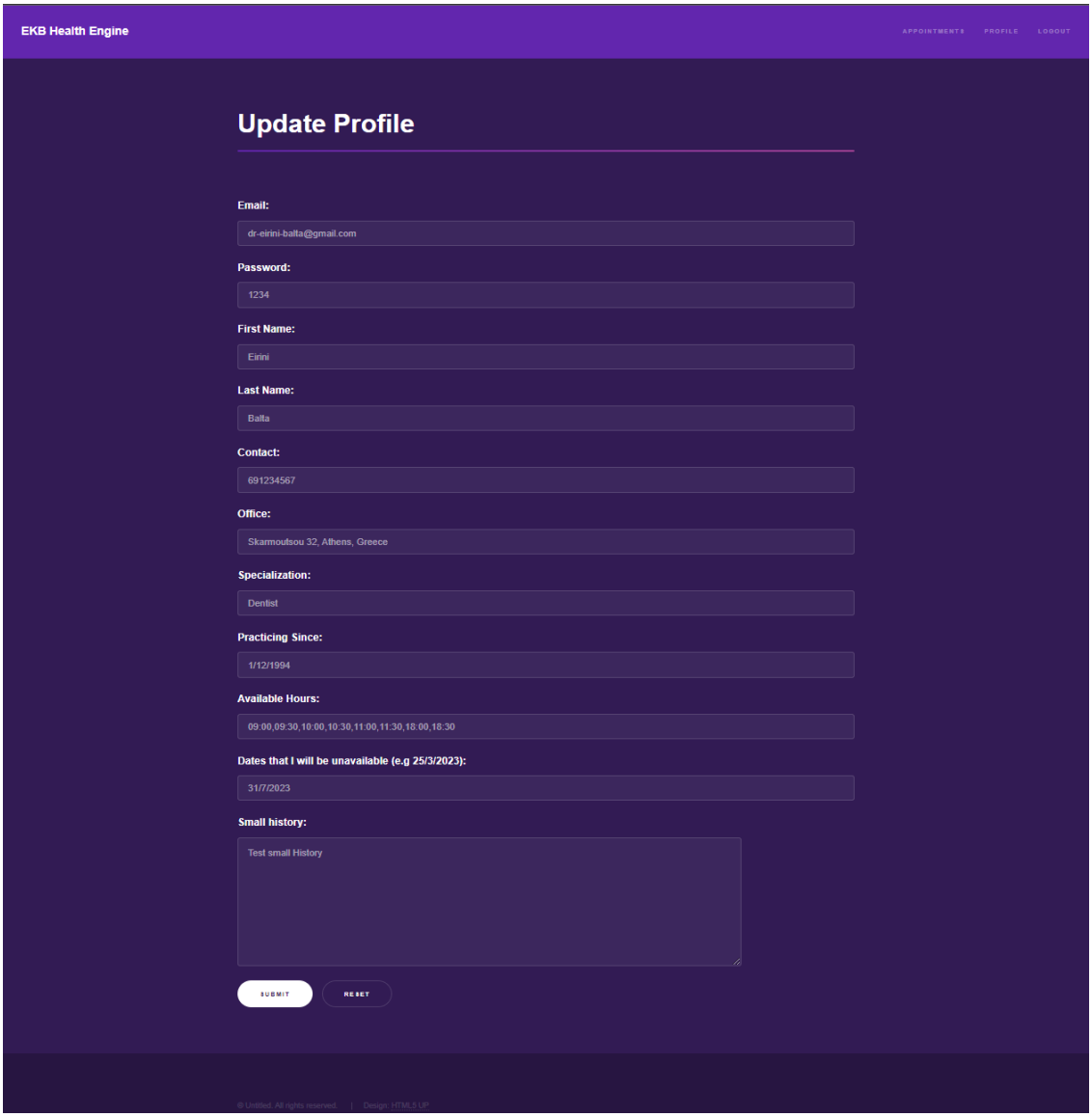
Με την επιτυχή δημιουργία του λογαριασμού του χρήστη ως γιατρός, ο γιατρός-χρήστης μεταφέρεται στην αρχική οθόνη του ιατρού. Ο πίνακας των ραντεβού διαφέρει από αυτήν των ασθενών (οι στήλες του πίνακα είναι: *First Name, Last Name, Date, Hour, Contact Number, Email, Patient info και Action)*. Εμφανίζεται ένα μήνυμα *''There are some missing information about you.**Please click here to fix that''***, το οποίο θα μείνει στην αρχική οθόνη μέχρι ο ιατρός να το επιλέξει και να συμπληρώσει το προφίλ του με περαιτέρω πληροφορίες, έτσι ώστε ο ασθενής να είναι σε θέση να βρει τις πληροφορίες που χρειάζεται. Μέχρι να συμπληρώσει το προφίλ του ο ιατρός, ο ασθενής δεν μπορεί να κλείσει κάποιο ιατρικό ραντεβού.

Ο χρήστης έχει επίσης και τη δυνατότητα να περιηγηθεί στη πλατφόρμα (*appointments, profile*) όπως και να αποσυνδεθεί (*logout*) από το λογαριασμό του, από το μενού που βρίσκεται στην επάνω δεξιά μεριά της οθόνης .



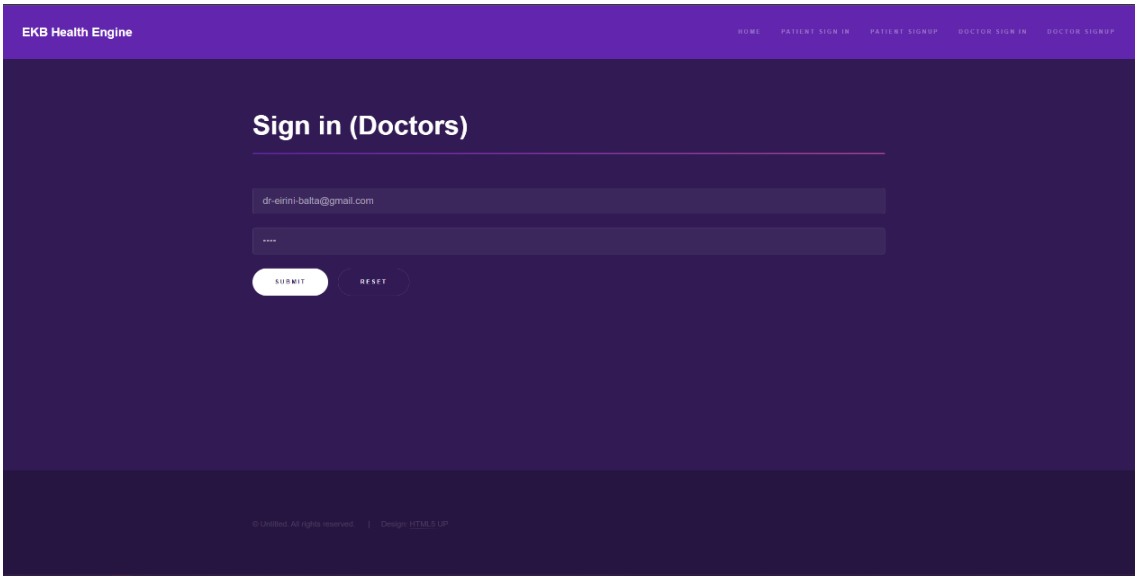
Εικόνα 21: Αρχική οθόνη προφίλ ιατρού και εμφάνιση ειδοποίησης για το προφίλ.

Όταν ο χρήστης επιλέξει να πατήσει πάνω στην ειδοποίηση, θα οδηγηθεί στην οθόνη του προσωπικού του προφίλ και θα πρέπει να καταχωρήσει συμπληρωματικά στοιχεία στα κατάλληλα πεδία της φόρμας, η οποία θα παρέχει στους ασθενείς σημαντικές πληροφορίες σχετικά με τις ώρες επισκεψιμότητας καθώς και την τοποθεσία του ιατρείου, το τηλέφωνο επικοινωνίας και άλλα στοιχεία του ιατρού που απαιτούνται για την δημιουργία του ραντεβού τους.



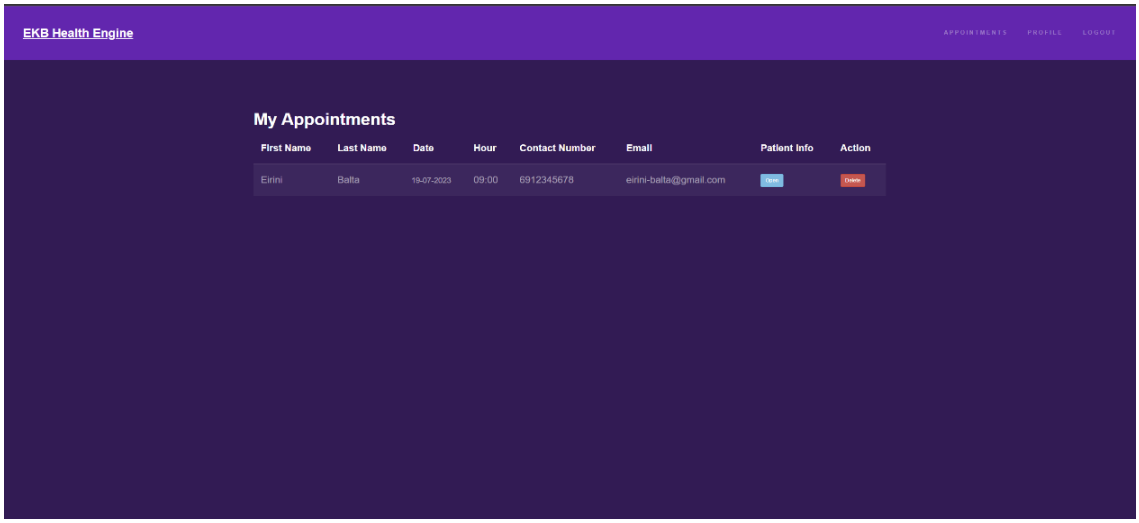
Εικόνα 22: Ανανέωση προφίλ ιατρού με συμπληρωματικά στοιχεία.

Ο χρήστης κάνοντας την επιλογή ‘’*Submit*’’ ανανεώνει το προφίλ του και αποσυνδέεται αυτόματα έτσι ώστε να ενημερωθεί κατάλληλα η βάση δεδομένων. Θα πρέπει να συνδεθεί ξανά στην πλατφόρμα για δει τα ιατρικά ραντεβού που έχουν καταχωρηθεί στη λίστα του.



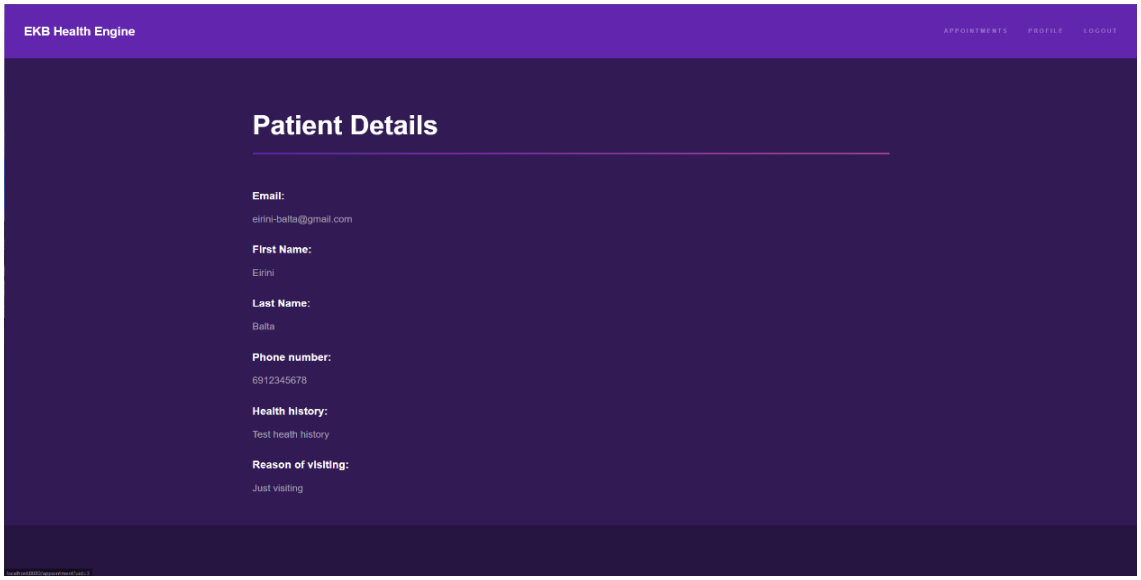
Εικόνα 23: Σύνδεση στο προφίλ ιατρού.

Όταν ο ασθενής επιλέξει τον ιατρό που θέλει να επισκεφτεί, η λίστα του ιατρού ανανεώνεται αυτόματα. Επομένως όταν ο ιατρός-χρήστης συνδεθεί ξανά έχει τη δυνατότητα να επιλέξει το *''Appointments''* από το μενού, που βρίσκεται στην πάνω δεξιά μεριά της οθόνης. Κάνοντας αυτή την επιλογή θα οδηγηθεί στη παρακάτω οθόνη που έχει την δυνατότητα να δει στοιχεία για την επίσκεψη του ασθενή καθώς επίσης μπορεί να διαγράψει το καταχωρημένο ραντεβού αν δεν επιθυμεί ή δεν μπορεί να το πραγματοποιήσει.



Εικόνα 24: Εμφάνιση ραντεβού Ιατρών.

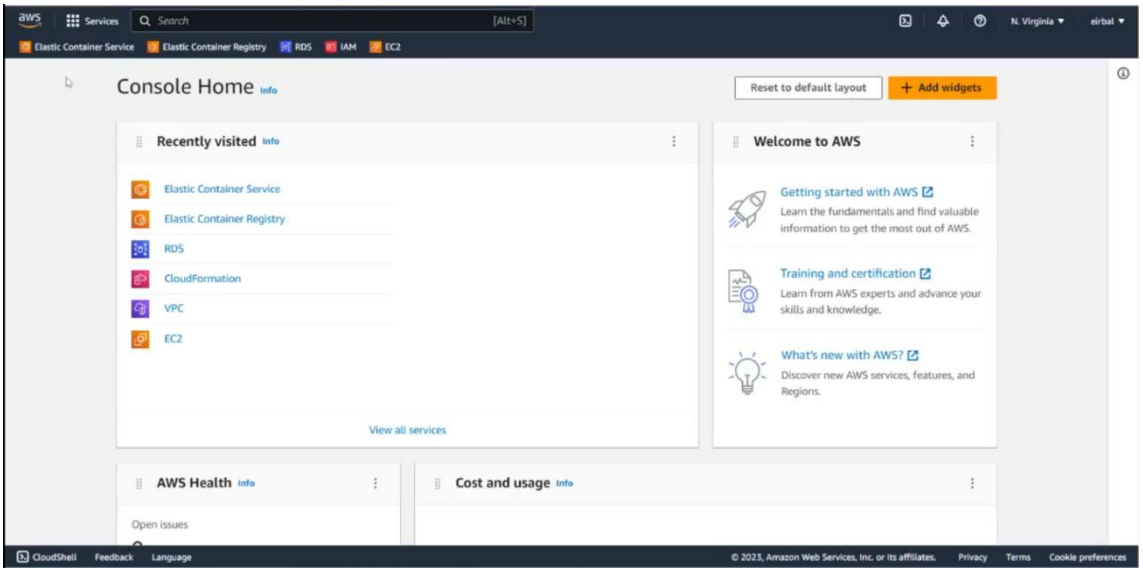
Ο ιατρός χρήστης κάνοντας την επιλογή *''Open''* στη στήλη του πίνακα *''Patient info''* μεταβαίνει στην παρακάτω οθόνη που έχει την δυνατότητα να δει στοιχεία για την επίσκεψη του ασθενή όπως το όνομα του, το τηλέφωνό του, ένα μικρό ιατρικό ιστορικό καθώς και τον λόγο επίσκεψης του ή την συμπτωματολογία που παρουσιάζει.



Εικόνα 25: Εμφάνιση πληροφοριών ασθενή που έχει κλείσει ραντεβού.

4.2 Παρουσίαση εργαλείων Amazon Web Services

Για την χρήση των υπηρεσιών του AWS, δημιουργήθηκε ένας λογαριασμός στη πλατφόρμα του AWS . Μετά τη δημιουργία του λογαριασμού ο διαχειριστής αυτού του λογαριασμού βλέπει τη παρακάτω οθόνη.



Εικόνα 26: Αρχική σελίδα του AWS.

Μετά την εγγραφή, πραγματοποιήθηκε η πλήρης εγκατάσταση του εργαλείου AWS cli και σε συνδυασμό με το εργαλείο Docker CLI, που εγκαταστάθηκε μαζί με το Docker Engine, ο διαχειριστής απέκτησε τη δυνατότητα να διαχειριστεί και να αναρτήσει τα containers του στις υπηρεσίες του AWS.

Για τη δημιουργία (build) και την ανάρτηση της εφαρμογής, με όλες τις απαραίτητες βιβλιοθήκες και εξαρτήσεις της (dependencies), προκειμένου να εκτελείται ντετερμινιστικά από όλα τα συστήματα, χρησιμοποιήθηκε ένα αρχείο με το όνομα *Dockerfile*, το οποίο δημιουργήθηκε στον ίδιο φάκελο με την εφαρμογή και περιέχει τις παρακάτω γραμμές εντολών:

```
FROM openjdk:17-alpine
COPY target/doctorappoint-1.0.2-SNAPSHOT.jar doctorappoint-1.0.2-SNAPSHOT.jar
ENTRYPOINT ["java","-jar","/doctorappoint-1.0.2-SNAPSHOT.jar"]
```

Μετά την εκτέλεση των παραπάνω εντολών έγινε έλεγχος των λειτουργιών της εφαρμογής και στη συνέχεια εκτελέστηκαν οι παρακάτω εντολές έτσι ώστε να αναρτηθεί σωστά η εφαρμογή στην υπηρεσία του ECR (Elastic Container Registry) του AWS:

```
docker build -t ekb-healthengine .
docker run -d --publish 8080:8080 ekb-healthengine
```

Αναφορικά οι παραπάνω εντολές εκτελέστηκαν σύμφωνα με το εγχειρίδιο χρήσης του AWS cli. Σε αυτό το εγχειρίδιο υπάρχουν αναλυτικές οδηγίες για το τι είναι το [*region*] και το [*aws_account_id*] και για λόγους ασφάλειας δεν αναγράφεται η πλήρη εντολή.

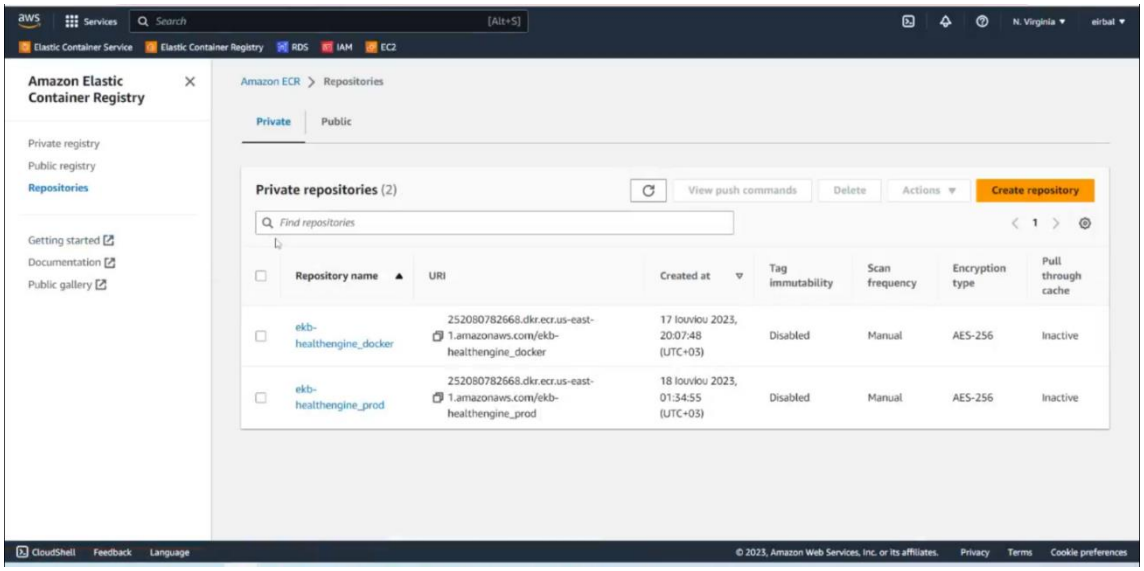
Στη συνέχεια δημιουργήθηκαν δύο containers. Αυτά αποτελούν τα δύο περιβάλλοντα, το περιβάλλον παραγωγής (production), όπου θα βλέπουν οι χρήστες της εφαρμογής, και το περιβάλλον ανάπτυξης/δοκιμών (docker), το οποίο θα βλέπουν και θα δοκιμάζουν μόνο οι προγραμματιστές. Στο περιβάλλον development συνδέθηκε η βάση δεδομένων H2 και στο περιβάλλον production συνδέθηκε η βάση δεδομένων Aurora της υπηρεσίας RDS (Relational Database Services) της AWS.

Η διαφοροποίηση στα containers έγινε από το αρχείο *application.properties* , ωστόσο τα βήματα για την δημιουργία και ανάρτηση της εφαρμογής είναι πανομοιότυπα και η παρακάτω εντολή εκτελέστηκε δύο φορές με τη μόνη διαφορά ότι στις εντολές, το όνομα του *repository-name* αλλάζει από *ekb-healthengine_docker* σε *ekb-healthengine_prod* :

```
aws ecr get-login-password --region [region] | docker login --username AWS --password-stdin [aws_account_id].dkr.ecr.[region].amazonaws.com
```

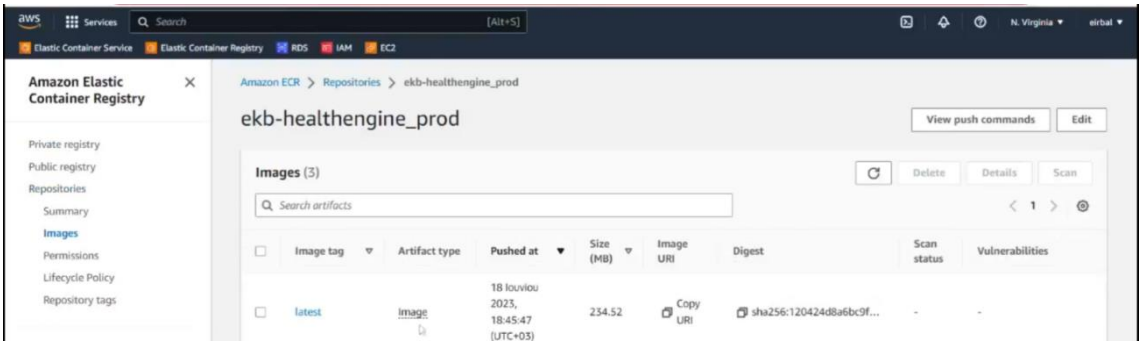
Με αποτέλεσμα μετά τις εντολές, ο διαχειριστής να βλέπει στο ECR όλες τις αλλαγές που έκανε από τη γραμμή εντολών.

```
aws ecr create-repository --repository-name ekb-healthengine_docker
aws ecr create-repository --repository-name ekb-healthengine_prod
```



Εικόνα 27: Κεντρική σελίδα ECR με ανεβασμένα τα containers.

Μεταβαίνοντας σε ένα από τα repositories φαίνονται τα στοιχεία για τα containers.



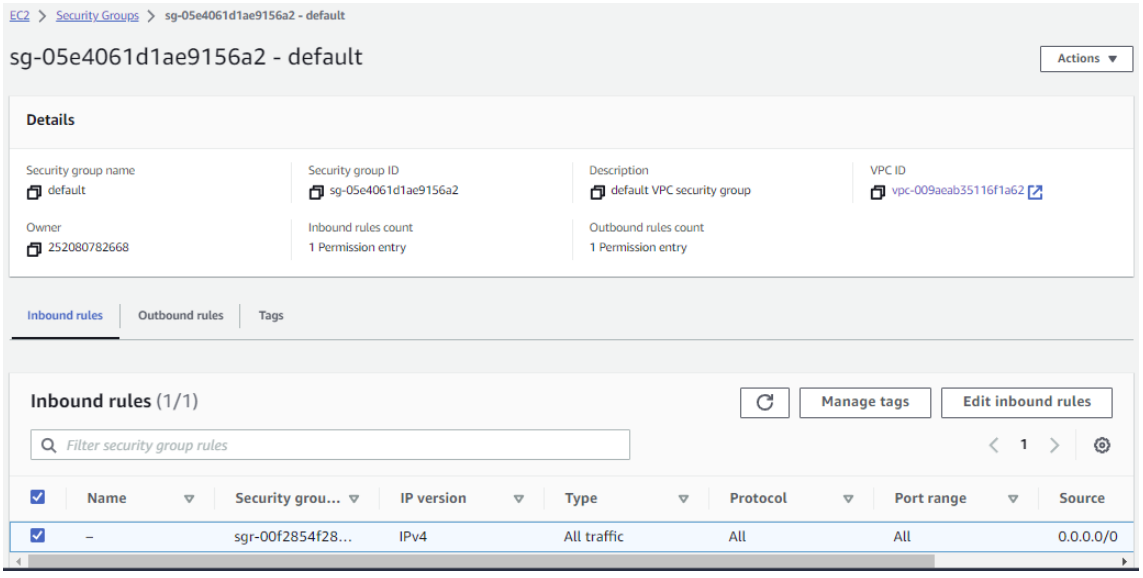
Εικόνα 28: Repository του ekb-healthengine_prod.

Στη συνέχεια δημιουργήθηκε ένα αρχείο με το όνομα task-definition.json . Αυτό το αρχείο, σύμφωνα με το εγχειρίδιο του AWS, περιέχει όλες τις απαραίτητες ρυθμίσεις με τις οποίες το ECS (Elastic Container Services) διαχειρίζεται τα containers. Παρακάτω βρίσκεται το αρχείο που χρησιμοποίησε ο διαχειριστής:

```
{
   "family": "ekb-prod",
   "containerDefinitions": [
      {
         "name": "ekb-prd",
         "image": "[aws_account_id].dkr.ecr.[region].amazonaws.com/ekb-healthengine_prod:latest",
         "cpu": 1024,
         "portMappings": [
            {
               "name": "ekb-prod-5000-tcp",
               "containerPort": 5000,
               "hostPort": 5000,
               "protocol": "tcp",
               "appProtocol": "http"
            }
         ],
         "essential": true,
         "environment": [],
         "environmentFiles": [],
         "mountPoints": [],
         "volumesFrom": [],
         "ulimits": [],
         "logConfiguration": {
```

```
            "logDriver": "awslogs",
            "options": {
              "awslogs-create-group": "true",
              "awslogs-group": "/ecs/prod-service-5000",
              "awslogs-region": "us-east-1",
              "awslogs-stream-prefix": "ecs"
            }
          }
        }
    ],
    "taskRoleArn": "arn:aws:iam:: [aws_account_id]:role/ecsTaskExecutionRole",
    "executionRoleArn": "arn:aws:iam:: [aws_account_id]:role/ecsTaskExecutionRole",
    "networkMode": "awsvpc",
    "requiresCompatibilities": [
      "FARGATE"
    ],
    "cpu": "1024",
    "memory": "3072",
    "runtimePlatform": {
      "cpuArchitecture": "X86_64",
      "operatingSystemFamily": "LINUX"
    }
}
```

Μετά τη σύνταξη του αρχείου, δημιουργήθηκε μια ομάδα ρυθμίσεων δικτύου (security group) με σκοπό να ανοίξει η εφαρμογή στο ευρύ κοινό:



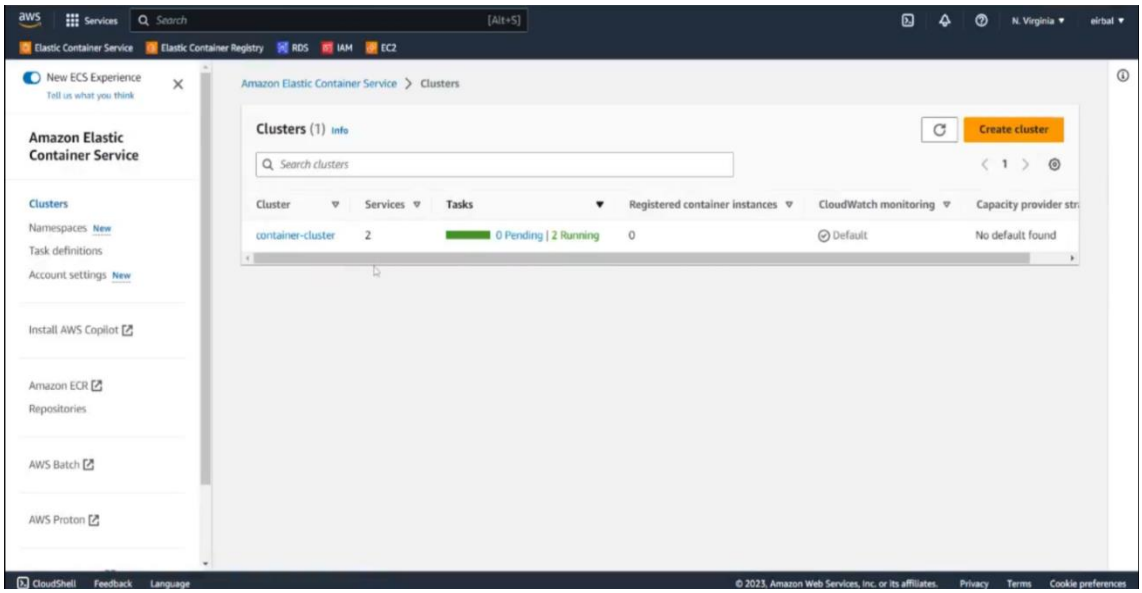
Εικόνα 29: Security group για το ECS.

Στη συνέχεια εκτελέστηκαν οι παρακάτω εντολές και έτσι ξεκίνησε το ECS να ομαδοποιεί και να διαχειρίζεται τα containers σαν συστάδα (cluster):

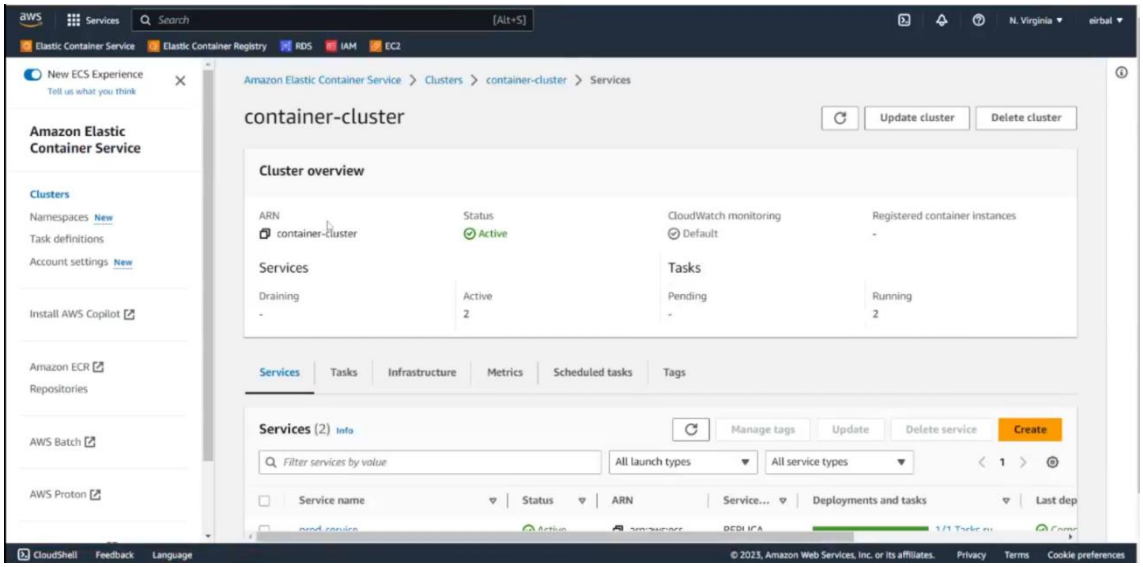
```
aws ecs register-task-definition --cli-input-json file://task-definition.json

aws ecs create-service --cluster container-cluster --service-name container-service --task-definition ekb-prod:1 --desired-count 1 --launch-type "FARGATE" --network-configuration "awsvpcConfiguration={subnets=[subnet-069fe47b4707dea86], securityGroups=[sg-05e4061d1ae9156a2], assignPublicIp=ENABLED}"
```

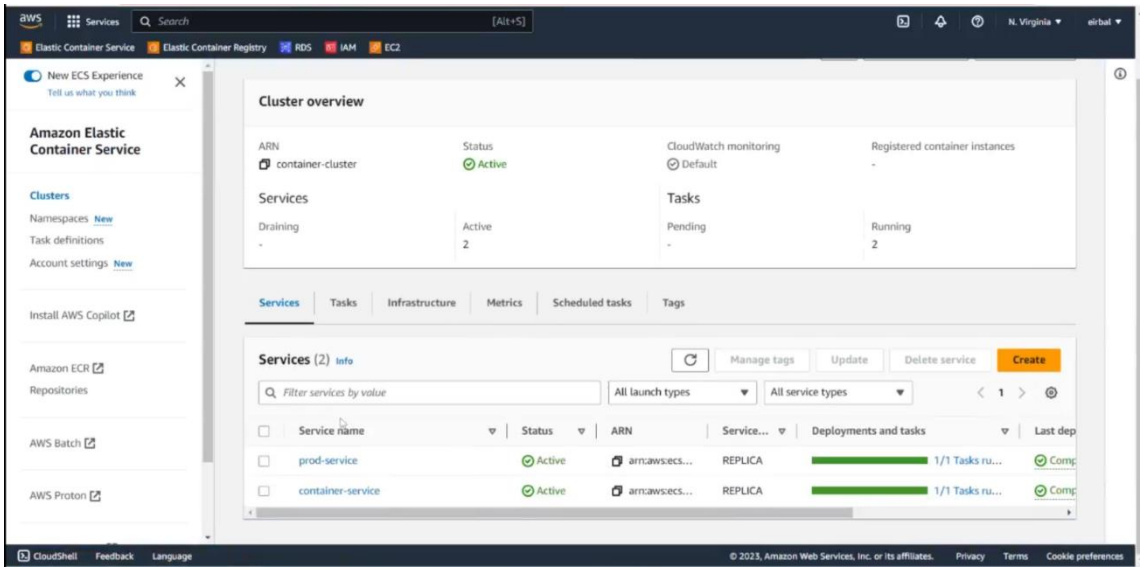
Μετά την εκτέλεση των εντολών, φαίνονται οι αλλαγές στο ECS του AWS και μέσα από τις οθόνες φαίνεται πως τρέχουν με επιτυχία τα containers.



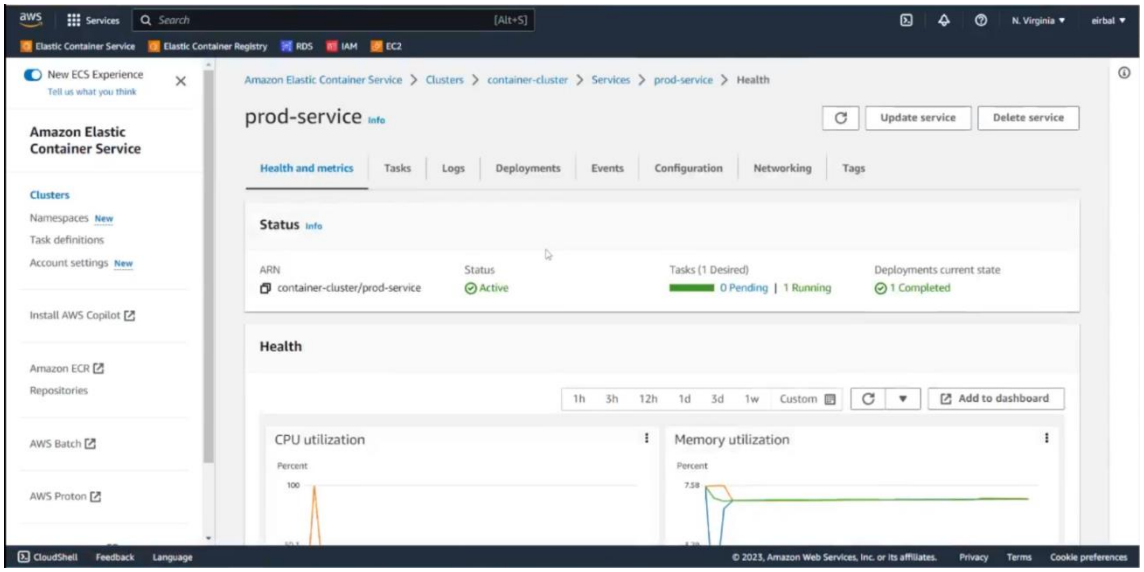
Εικόνα 30: Κύριο cluster του ECS.



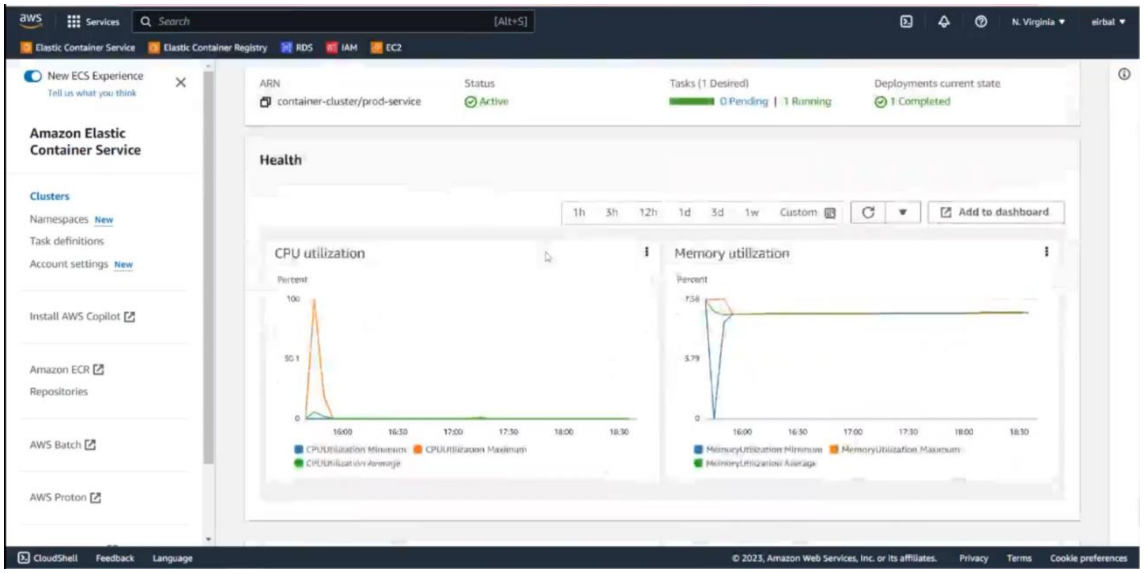
Εικόνα 31: Services του κύριου cluster (μέρος 1).



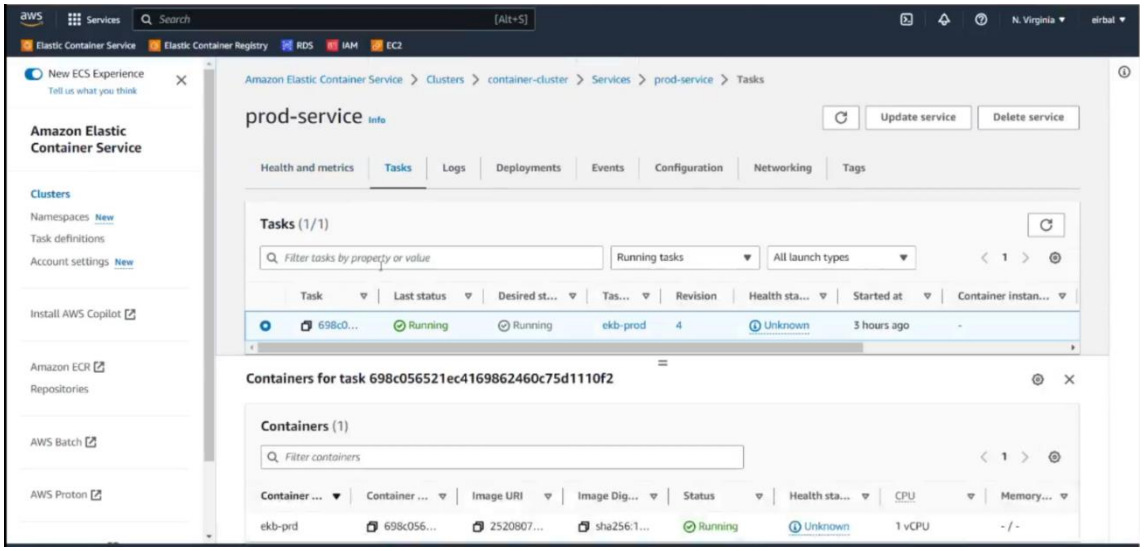
Εικόνα 32: Services του κύριου cluster (μέρος 2).



Εικόνα 33: Διαγράμματα και μετρήσεις για το prod-service (μέρος 1).



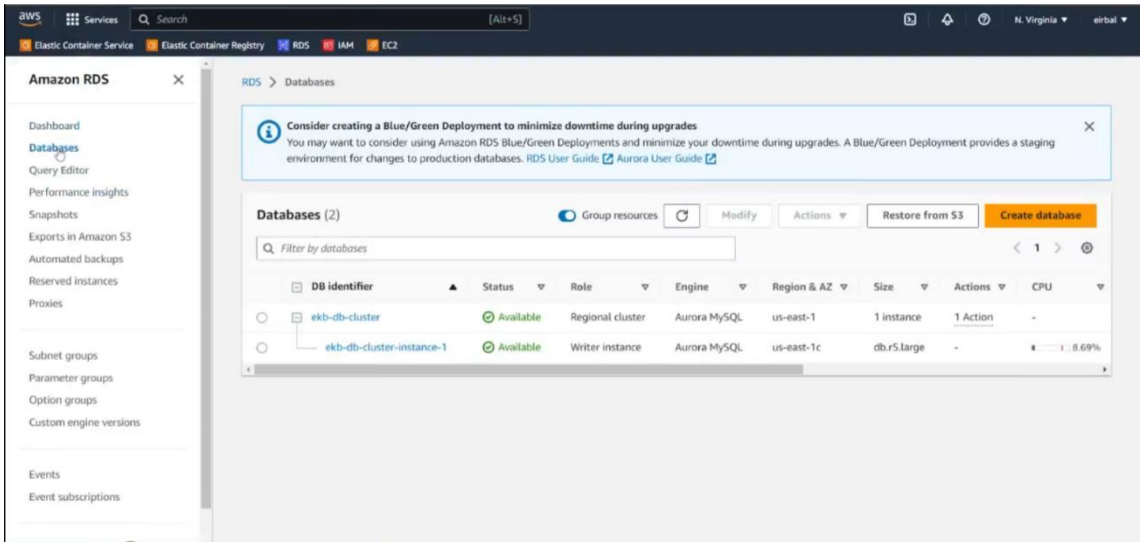
Εικόνα 34: Διαγράμματα και μετρήσεις για το prod-service (μέρος 2).



Εικόνα 35: Κατάσταση container παραγωγής (prod-service).

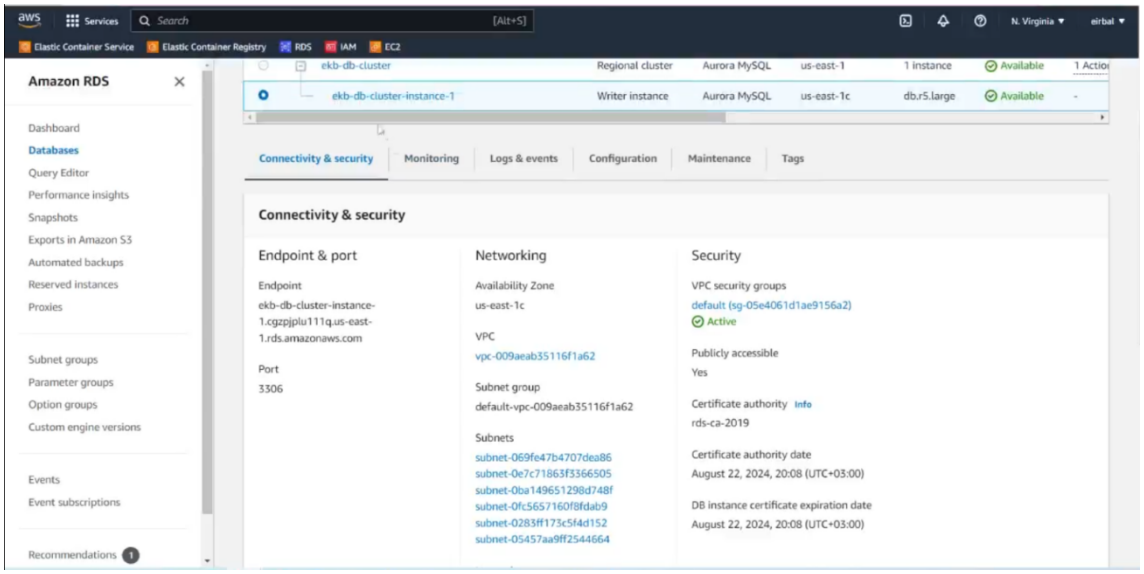
Όσον αφορά τη βάση δεδομένων Aurora του AWS, δημιουργήθηκε πριν το ECR και δεν πραγματοποιήθηκε με γραμμές εντολών αλλά έγινε από τις επιλογές του AWS, όπως αυτό αναγράφεται από το εγχειρίδιο του AWS. Αυτό έγινε επειδή χρησιμοποιήθηκε το μοντέλο του ORM, οπότε δεν υπήρχε λόγος να εκτελεστούν εντολές για τη δημιουργία των πινάκων και εισαγωγή δεδομένων στη βάση.

Στην παρακάτω εικόνα φαίνεται το όνομα της βάσης όπως και άλλα στοιχεία, με τα οποία έγινε η σύνδεση μέσω του Spring Boot και του MySQL workbench.



Εικόνα 36: Κεντρική οθόνης της παραγωγικής βάσης δεδομένων στο RDS.

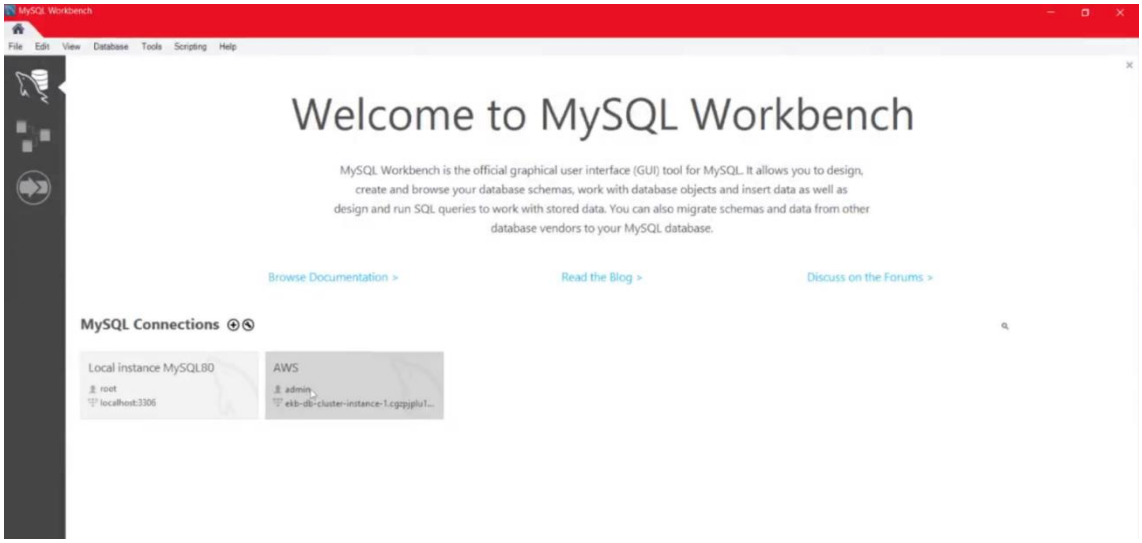
Πιο συγκεκριμένα, επιλέγοντας το *Writer instance* του *Regional cluster* εμφανίζεται η παρακάτω οθόνη. Εδώ η πιο σημαντική πληροφορία που παρέχεται είναι το *Endpoint* το οποίο δίνει τη δυνατότητα επικοινωνίας με τη βάση.



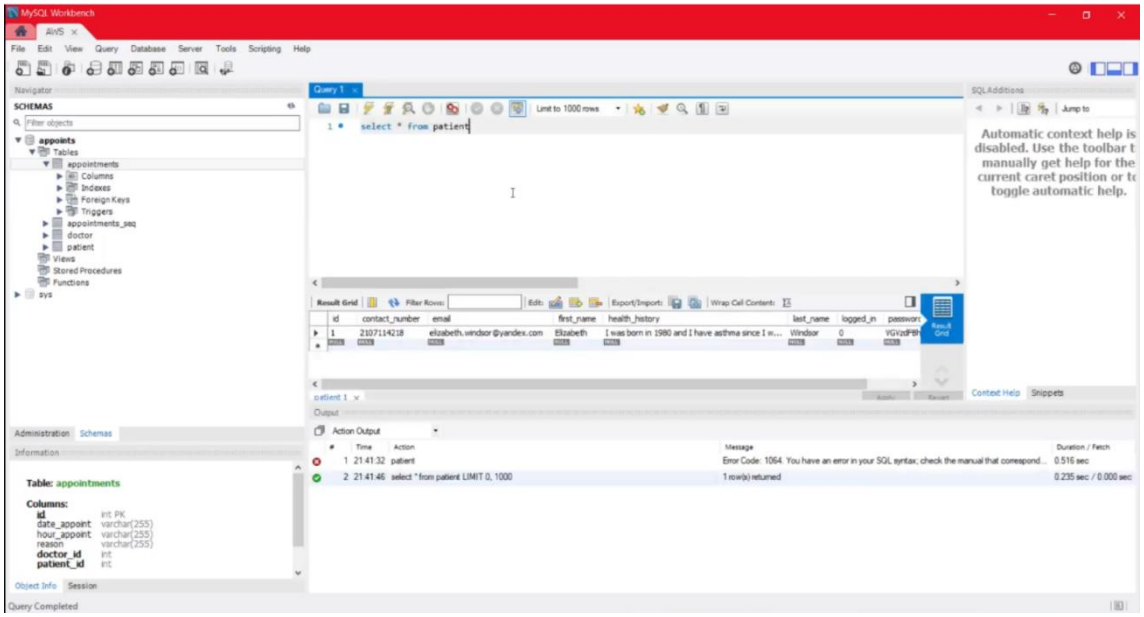
Εικόνα 37: Πληροφορίες του writer-instance της παραγωγικής βάσης δεδομένων στο RDS.

Για την ανάγνωση των πινάκων και των στοιχείων της βάσης χρησιμοποιήθηκε το εργαλείο *MySQL Workbench* . Η επιλογή αυτού του εργαλείου έγινε διότι η βάση δεδομένων της Aurora

είναι συμβατή με την έκδοση του MySQL Workbench (MySQL 5.x.x) που ήταν ήδη εγκατεστημένη στον υπολογιστή.



Εικόνα 38: Κεντρική σελίδα του MySQL Workbench.



Εικόνα 39: Εμφάνιση καταγραφών του πίνακα patients μέσα από το MySQL Workbench.

5. Συμπεράσματα

Η διατριβή αυτή αποτελεί μια μελέτη που αναδεικνύει τη σημασία της τεχνολογίας στο τομέα της υγείας και παρουσιάζει μια εφαρμογή που σχεδιάστηκε και υλοποιήθηκε, με στόχο να προσφέρει μια σύγχρονη προσέγγιση στη διαχείριση των ιατρικών ραντεβού. Αυτή η εφαρμογή αποτελεί ένα σημαντικό εργαλείο που συνδυάζει προχωρημένες τεχνολογικές λύσεις και αυξάνει την ευελιξία των ασθενών και των ιατρών.

Στο πλαίσιο αυτό, η δημιουργία της ψηφιακής πλατφόρμας για τη διαχείριση ιατρικών ραντεβού αποσκοπεί στη βελτίωση της αποτελεσματικότητας και της αποδοτικότητας της διαδικασίας, ενώ ταυτόχρονα προάγει την ενίσχυση της επικοινωνίας μεταξύ ιατρών και ασθενών. Αυτό επιτυγχάνεται μέσω της ανταλλαγής πληροφοριών και ενημερώσεων σχετικά με τα ιατρικά ραντεβού και την κατάσταση υγείας των ασθενών.

Επιπρόσθετα, η χρήση των τεχνολογιών νέφους προσθέτουν ένα επιπλέον επίπεδο πολυπλοκότητας στην υλοποίηση της εφαρμογής, ωστόσο οι παροχές που προσφέρουν καλύπτουν ένα ευρύ φάσμα προβλημάτων που συναντώνται σε κοινότυπες μεθόδους παροχής υποδομών, όπως είναι η προσβασιμότητα από τους χρήστες και ο χώρος αποθήκευσης δεδομένων.

Τέλος, η εφαρμογή διασφαλίζει ένα ασφαλές περιβάλλον για την αποθήκευση και τη διαχείριση ιατρικών δεδομένων, ενισχύοντας την ακεραιότητα των πληροφοριών αυτών.

Όλα τα παραπάνω επιτεύχθηκαν με τη μελέτη παρόμοιων εφαρμογών σε άλλες χώρες, η οποία βοήθησε στην κατανόηση των βέλτιστων πρακτικών που εφαρμόστηκαν σε αυτές και συνέβαλε στην ανάπτυξη της παρούσας εφαρμογής.

Συνοψίζοντας, η τεχνολογική υποδομή που χρησιμοποιήθηκε καθώς και η αρχιτεκτονική που επιλέχθηκε, συνδυάστηκαν με σκοπό τη δημιουργία της παρούσας εφαρμογής που όχι μόνον ανταποκρίνεται στις σύγχρονες απαιτήσεις του τομέα της υγείας αλλά επίσης εγγυάται την αξιοπιστία και την αποδοτικότητά της.

5.1 Μελλοντικές επεκτάσεις

Η παρούσα εφαρμογή αποτελεί ήδη μια αξιόπιστη λύση για την ψηφιακή διαχείριση ραντεβού στον τομέα της υγείας, αλλά υπάρχει ακόμα χώρος για ανάπτυξη και βελτίωση όπως συνήθως συμβαίνει με όλα τα τεχνολογικά προϊόντα.

Ως μελλοντικές επεκτάσεις της εφαρμογής μπορεί να είναι:

- Ενσωμάτωση περισσότερων λειτουργιών: Μπορούν να επεκταθούν οι δυνατότητες της εφαρμογής προσθέτοντας λειτουργίες που να υποστηρίζουν τις τηλεϊατρικές συνεδρίες μεταξύ ιατρών και ασθενών, διαχείριση Φαρμάκων, ηλεκτρονική συνταγογράφηση κ.ά.

- Ενσωμάτωση μηνυμάτων,email και ειδοποιήσεων: Μπορούν να προστεθούν συστήματα αυτοματοποιημένης αποστολής μηνυμάτων, email και ειδοποιήσεων για υπενθύμιση των ραντεβού τους.
- Ενσωμάτωση μεγαλύτερης ανάλυσης δεδομένων: Μπορούν να χρησιμοποιηθούν εργαλεία Business Intelligence και Data Analytics για να αναλυθούν τα δεδομένα των ραντεβού και να βελτιωθεί η απόδοση του συστήματος.
- Επέκταση λειτουργιών με τη προσθήκη τεχνητής νοημοσύνης (AI) και Μηχανικής Μάθησης (Machine Learning): Το AI και το Machine Learning μπορούν να βελτιώσουν τη διαγνωστική ακρίβεια, να προβλέψουν την ανάγκη για συγκεκριμένες θεραπείες και να προσφέρουν εξατομικευμένες προτάσεις υγείας.

Με την επέκταση αυτών των δυνατοτήτων, η εφαρμογή θα μπορεί να καλύψει ευρύτερα φάσματα χρηστών και να παρέχει μια ακόμα πιο αποτελεσματική λύση για τον κλάδο της υγείας.

6. Βιβλιογραφία

1. Smith, J., & Johnson, A. (2020). Appointment Scheduling Systems in Healthcare. New York, NY: Publisher.
2. Brown, L. (2019). Improving Patient Access: Innovative Approaches to Appointment Scheduling. Chicago, IL: Publisher.
3. Anderson, R., & Davis, M. (2018). Managing Patient Appointments: Best Practices for Healthcare Organizations. Boston, MA: Publisher.
4. Wilson, K. (2017). Effective Appointment Management in Healthcare: Strategies and Tools. San Francisco, CA: Publisher.
5. Thompson, E. (2016). The Role of Technology in Appointment Scheduling for Healthcare. London, UK: Publisher.
6. Garcia, M., & Rodriguez, S. (2015). A Comparative Study of Appointment Scheduling Models in Healthcare. Journal of Healthcare Management, 40(3), 78-85.
7. Johnson, R., & Smith, K. (2014). Implementing an Electronic Appointment Scheduling System in a Hospital Setting. Journal of Medical Practice Management, 30(2), 45-52.
8. Davis, M., & Brown, L. (2013). Patient Satisfaction with Appointment Scheduling Systems in Primary Care. Journal of Healthcare Quality, 35(4), 67-74.
9. Wilson, K. (2012). The Impact of Appointment Scheduling Systems on Patient Wait Times. Healthcare Management Review, 37(1), 24-31.
10. Thompson, E., & Garcia, M. (2011). The Role of Patient Preferences in Appointment Scheduling. Journal of Healthcare Administration, 36(2), 56-63.
11. Smith, J., & Johnson, A. (2010). The Use of Technology in Appointment Scheduling: A Review of the Literature. Journal of Healthcare Information Management, 27(4), 18-25.
12. Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design patterns: elements of reusable object-oriented software*. Pearson Deutschland GmbH.
13. Brown, L., & Davis, M. (2009). Appointment Scheduling in Healthcare: Current Practices and Future Trends. Journal of Medical Systems, 33(5), 36-42.
14. Wilson, K., & Thompson, E. (2008). Appointment Scheduling Systems: A Comparison of Manual and Electronic Methods. Journal of Healthcare Technology, 25(3), 54-61.
15. Garcia, M., & Rodriguez, S. (2007). The Impact of Appointment Scheduling on Patient Flow in Healthcare Organizations. Journal of Healthcare Operations Management, 22(1), 12-19.
16. Johnson, R., & Smith, K. (2006). The Role of Appointment Scheduling Systems in Improving Patient Access to Healthcare. Journal of Healthcare Management, 32(2), 45-52.
17. Davis, M., & Brown, L. (2005). Patient Preferences for Appointment Scheduling: A Qualitative Study. Journal of Healthcare Quality, 30(4), 67-74.
18. Wilson, K. (2004). The Effectiveness of Appointment Scheduling Systems in Reducing Patient Wait Times. Healthcare Management Review, 29(1), 24-31.
19. Thompson, E., & Garcia, M. (2003). The Impact of Appointment Scheduling on Patient Satisfaction. Journal of Healthcare Administration, 28(2), 56-63.
20. Smith, J., & Johnson, A. (2002). The Role of Technology in Appointment Scheduling: A Review of the Literature. Journal of Healthcare Information Management, 17(4), 18-25.
21. Brown, L., & Davis, M. (2001). Appointment Scheduling in Healthcare: Current Practices and Future Trends. Journal of Medical Systems, 23(5), 36-42.

22. Wilson, K., & Thompson, E. (2000). Appointment Scheduling Systems: A Comparison of Manual and Electronic Methods. Journal of Healthcare Technology, 15(3), 54-61.
23. Hassel, L., Hunt, E. B., Hoog, J., de Mul, M., & Stein, K. F. (2015). Application of cloud computing in healthcare. BMC Medical Informatics and Decision Making, 15(1), 17.
24. Liu, S., & Ma, Q. (2017). How does patient-provider portal access improve appointment adherence? An analysis with a panel data of over 2.3 million appointments. BMC Medical Informatics and Decision Making, 17(1), 1-10.
25. Erdogan, S. A., Krupski, T. L., & Lobo, J. M. (2018). Optimization of telemedicine appointments in rural areas. Service Science, 10(3), 261-276.
26. O'Leary, D. E. (2017). Cloud computing and health care. The Communications of the Association for Information Systems, 41(1), 16.
27. Šesto, S., & Simović, A. (2020). Amazon ECS in Action. Manning Publications.
28. Nickoloff, J. (2016). Docker in Action. Manning Publications.
29. Walls, C. (2016). Spring Boot in Action. Manning Publications.
30. Newman, S. (2015). Building Microservices: Designing Fine-Grained Systems. O'Reilly Media.
31. Newman, S. (2019). Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith. O'Reilly Media.
32. Long, J. (2019). AWS Certified Developer - Associate Guide: Your one-stop solution to passing the AWS developer's certification. Packt Publishing.
33. Garg, R. (2021). Mastering AWS Aurora: An Expert Guide to Building and Scaling High-Performance Databases on the AWS Cloud. Packt Publishing.
34. Sharma, A. (2021). Spring Boot 2 Recipes: A Problem-Solution Approach. Apress.
35. Zaki, M., & Rashu, R. (2020). Cloud computing in healthcare. Virtual and Mobile Healthcare: Breakthroughs in Research and Practice, 1(1), 724-740.
36. Liu, L. & Stroulia, E. (2011). Improving healthcare appointment scheduling system performance. Procedia Computer Science, 4(1), 297-306.
37. Niknamfar, A. H., & Shetaban, S. M. (2018). Investigation of the effect of cloud computing in development of health care industry with mentioning the offering services. International Journal of Engineering & Technology, 7(2), 752-757.
38. Weider, K., & Blichfeldt, B. S. (2013). Enforced standards in appointment scheduling: Case study of a plastic surgery outpatient clinic. Scandinavian Journal of Caring Sciences, 27(3), 643-649.
39. Ahmed, E., & Ahmed, A. U. (2016). A comparative study of mobile cloud computing for appointment scheduling in healthcare. Advances in Social and Occupational Ergonomics, 487(1), 373-382.
40. Altuwairiqi, M., Goodwin, R., & Roudsari, A. (2020). E-booking Health Services: Knowledge Discovery from Online NHS Choices and Patient Access Data. Springer International Publishing.
41. https://stackshare.io/doctolib/doctolib
42. https://stackshare.io/zocdoc/zocdoc
43. https://stackshare.io/docplanner-tech/docplanner
44. https://stackshare.io/nexhealth/nexhealth