ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ – ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Πρόγραμμα Μεταπτυχιακών Σπουδών «Προηγμένα Συστήματα Πληροφορικής – Ανάπτυξη Λογισμικού και Τεχνητής Νοημοσύνης»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	Σύστημα καταγραφής κρουσμάτων COVID-19 COVID-19 outbreak recording System
Ονοματεπώνυμο Φοιτητή	Κωνσταντίνος Βελντές
Πατρώνυμο	Γεώργιος
Αριθμός Μητρώου	ΜΠΣΠ 19007
Επιβλέπων	Ευθύμιος Αλέπης , Αναπληρωτής Καθηγητής

Ημερομηνία Παράδοσης **Οκτώβριος 2022**

Τριμελής Εξεταστική Επιτροπή

Ευθύμιος Αλέπης Αναπληρωτής Καθηγητής	Μαρία Βίρβου Καθηγήτρια	Κωνσταντίνος Πατσάκης Αναπληρωτής Καθηγητής

Περίληψη

Η εφαρμογή φιλοδοξεί να παρουσιάσει στον χρήστη ένα Σύστημα καταγραφής κρουσμάτων COVID-19, στο οποίο νοσηλευτικό και ιατρικό προσωπικό της χώρας θα μπορεί να δει άμεσα πληροφορίες και στατιστικά σχετικά με κρούσματα επιβεβαιωμένα ή μη του COVID-19.

Αναφορικά με τις τεχνολογίες που χρησιμοποιεί η συγκεκριμένη εφαρμογή μπορούμε να πούμε ότι η εφαρμογή διαιρείται σε δυο ξεχωριστά μέρη, τα οποία είναι αυτά του:

- πίσω μέρους της εφαρμογής / Back-end
- εμπρόσθιου μέρους της εφαρμογής / Front-end

Για το Back-end μέρος της εφαρμογής, έγινε χρήση του -βασισμένου σε Java δωρεάν και ανοιχτού κώδικα- Spring Boot framework, ενώ για το Front-end μέρος της εφαρμογής έγινε χρήση της - δωρεάν και ανοιχτού κώδικα βιβλιοθηκης React – βιβλιοθήκης, η οποια βασιζεται στο JavaScript, διαμεσου της οποιας μπρουν να χτιστουν μέσα διασύνδεσης χρήστη, βασιζόμενα σε UI components. Επίσης, γίνεται χρήση μιας δωρεάν και ανοιχτού κώδικα σχεσιακής βάσης δεδομένων της MariaDB. Παράλληλα, για την παραπάνω βάση χρησιμοποιήθηκε το Hibernate ORM και γίνεται δημιουργία και χρήση JSON Web Token.

Για κάθε ενδεχόμενο κρούσμα, σε πρώτη φάση θα καταγράφονται δημογραφικά δεδομένα καθώς και το ΑΜΚΑ του. Η καταγραφή θα γίνεται από το εκάστοτε νοσηλευτικό προσωπικό διαμέσου του συστήματος σε βάση δεδομένων. Επίσης, σε δεύτερο χρόνο το σύστημα θα δίνει την δυνατότητα σε ιατρικό προσωπικό να σημαίνει το κάθε κρούσμα ως επιβεβαιωμένο ή μη.

Η συγκεκριμένη εφαρμογή είναι μια Web εφαρμογή κάτι που σημαίνει ότι θα μπορεί να υπάρχει πρόσβαση από οπουδήποτε, αρκεί ο ενδιαφερόμενος χρήστης να έχει πρόσβαση στο διαδίκτυο και να έχει τους κατάλληλους κωδικούς.

Abstract

The application aims to present the user with a COVID-19 case registration system, in which nursing and medical staff in the country will be able to directly view information and statistics on confirmed or unconfirmed COVID-19 cases.

Regarding the technologies used in this application, we can say that the application is divided into two separate parts, which are:

- the Back end of the application
- the Front end of the application

For the Back-end part of the application, the Java-based free and open-source Spring Boot framework was used, while for the Front-end part of the application, the free and open-source React library was used, which is based on JavaScript, through which user interface tools can be built, based on UI components. Also, a free and open-source relational database of MariaDB is used. In parallel, for the above database, Hibernate ORM is used and JSON Web Token is created and used.

For each potential case, demographic data and its social security number will be recorded in the first phase. The recording will be done by the individual nursing staff through the system in a database. Also, for a second time, the system will enable medical staff to indicate whether each case is confirmed or not.

This application is a Web application which means that it can be accessed from anywhere, as long as the user concerned has access to the Internet and has the appropriate passwords.

Περιεχόμενα

1.Εισαγωγή

1.1.Στόχοι εργασίας

Στο πλαίσιο της εργασίας αυτής θα αναπτυχθεί σύγχρονη διαδικτυακή εφαρμογή, η οποία θα αφορά στην ανάπτυξη συστήματος καταγραφής κρουσμάτων κορονοϊού μέσω του Spring Boot 2 Framework και της React JavaScript library. Επίσης, θα δημιουργηθούν 2ο ρόλοι για διαβαθμισμένους χρήστες, οι οποίοι θα μπορούν, ο μεν πρώτος να καταγράφει πιθανά κρούσματα ενώ ο δεύτερος θα τα σημαίνει ως επιβεβαιωμένα ή όχι κρούσματα κορονοϊού. Επιπλέον, θα υπάρχουν και στατιστικά των όλων κρουσμάτων. Στο πλαίσιό της θα τεκμηριωθεί με τη βοήθεια κειμένων και διαγραμμάτων η διαδικασία καθορισμού απαιτήσεων, σχεδιασμού και ανάπτυξης του αντίστοιχου λογισμικού. Ως εκ τούτου το έργο διαιρείται σε 2 φάσεις.

Οι φάσεις αυτές είναι:
1. Η ανάπτυξη της Backend RESTful εφαρμογής διάμεσου Spring Boot 2.
2. Η ανάπτυξη του Frond end διαμέσου της React JavaScript library.

Θα καταγράψουμε βήμα προς βήμα την διαδικασία της Υλοποίησης, κατά την οποία αναπτύσσεται το λογισμικό καθαυτό και εγκαθίσταται σε περιβάλλον λειτουργίας. Οι φάσεις που προαναφέρθηκαν αποτελούν τα κύρια ορόσημα του έργου. Με την ολοκλήρωσή τους παραδίδονται τα αντίστοιχα κομμάτια λογισμικού τα οποία στην σύνθεσή τους θα αποτελέσουν τον ηλεκτρονικό ιστότοπο.

Τα δεδομένα θα αποθηκεύονται σε μια βάση δεδομένων (MariaDB) και θα χρησιμοποιηθούν διάφορα στοιχεία React τρίτων κατασκευαστών για να γίνει το Frontend πιο φιλικό προς το χρήστη.

1.2. Ορισμός του προβλήματος προς επίλυση

Μια ιατρική δομή στην χώρα επιδιώκει να δημιουργηθεί ένα σύστημα καταγραφής κρουσμάτων κορονοϊού διαμέσου δημιουργίας μιας διαδικτυακή εφαρμογή. Ο ιστότοπος αυτός θα είναι προσιτός στους χρήστες του, ανεξαρτήτως της εξοικείωσής τους στη χρήση των ηλεκτρονικών υπολογιστών.

Το σύστημα θα προσφέρει:

1. Σελίδες προσβάσιμες σε εγγεγραμμένο απλό νοσηλευτικό προσωπικό για απλή καταγραφή του περιστατικού χωρίς να είναι το κρούσμα επιβεβαιωμένο.
2. Σελίδες προσβάσιμες μόνο σε εγγεγραμμένο ιατρικό προσωπικό, το οποίο θα μπορεί να πραγματοποιεί την σήμανση του εκάστοτε περιστατικού ως επιβεβαιωμένο.
3. Εύκολη ένδειξη σε σύνδεσμο για τα υπάρχον στατιστικά των κρουσμάτων από την βάση.

2.Ανασκοπηση πεδίου

2.1 Spring Boot 2

Το Spring Boot είναι ένα σύγχρονο Backend Framework βασισμένο σε Java που διευκολύνει τη δημιουργία αυτόνομων, παραγωγικών εφαρμογών βασισμένων στο Spring, τις οποίες μπορούμε να "τρέξουμε" διάμεσου της Spring Boot και των βιβλιοθηκών τρίτων δίνοντάς μας

την δυνατότητα να ξεκινήσουμε με την ελάχιστη δυνατή προσπάθεια. Οι περισσότερες εφαρμογές Spring Boot χρειάζονται ελάχιστη διαμόρφωση στην Spring.

Χαρακτηριστικά:

- Δημιουργία αυτόνομων εφαρμογών Spring.
- Ενσωμάτωση απευθείας του Tomcat, του Jetty ή το Undertow (δεν χρειάζεται να αναπτύξουμε αρχεία WAR).
- Παροχή εξαρτήσεων "εκκίνησης" τέτοια ώστε να απλοποιείται η όλη ανάπτυξη.
- Αυτόματη διαμόρφωση της Spring και των βιβλιοθηκών 3ων κατασκευαστών όποτε είναι δυνατόν.
- Παροχή λειτουργιών έτοιμων για ανάπτυξη, όπως μετρικές, έλεγχοι υγείας και εξωτερικές ρυθμίσεις παραμέτρων.
- Απολύτως καμία δημιουργία κώδικα και καμία απαίτηση για διαμόρφωση XML.

(“Spring Boot,” n.d.)

2.2 React JavaScript

Η React είναι μια βιβλιοθήκη JavaScript για την κατασκευή διεπαφών χρήστη.

Χαρακτηριστικά:

- Δηλωτική. Η βιβλιοθήκη React καθιστά εύκολη τη δημιουργία διαδραστικών διεπαφών χρήστη. Σχεδιάζοντας απλές προβολές για κάθε κατάσταση της εφαρμογής μας, η React θα ενημερώνει και θα απεικονίζει αποτελεσματικά ακριβώς τα σωστά στοιχεία όταν αλλάζουν τα δεδομένα.
- Component-Based. Δίνει την δυνατότητα κατασκευής ενθυλακωμένων components/συστατικών που διαχειρίζονται τη δική τους κατάσταση και, στη συνέχεια, δίνει την δυνατότητα σύνθεσης για δημιουργία σύνθετων UI.

(“React – A JavaScript library for building user interfaces,” n.d.)

3.Δημιουργια του Backend της εφαρμογής.

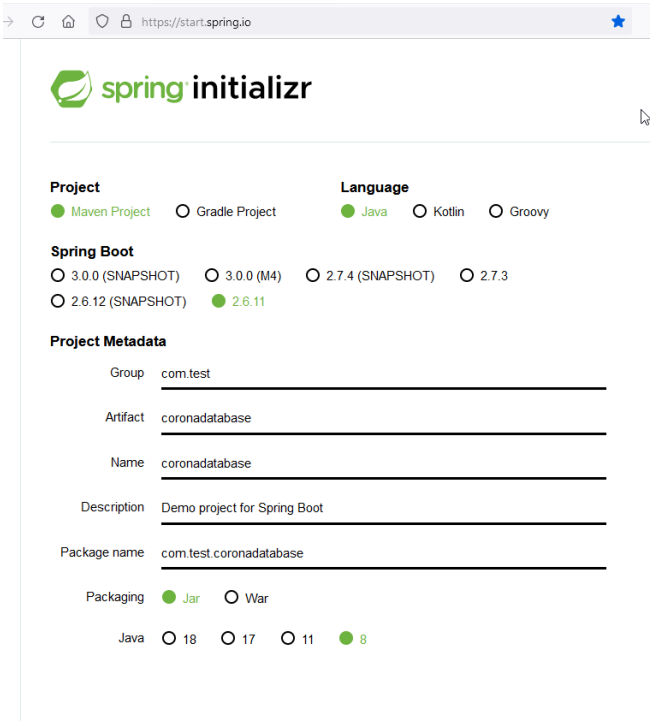
3.1 Τεχνικές απαιτήσεις

Θα χρειαστούμε το Java SDK έκδοση 8 ή νεότερη. Στην παρούσα εργασία χρησιμοποιούμε το Java SDK έκδοση 8 της Amazon (“Amazon Corretto Production-ready distribution of OpenJDK,” n.d.).Επίσης για την παρούσα υλοποίηση έγινε χρήση του Intellij IDE Comunity edition (“Download IntelliJ IDEA: The Capable & Ergonomic Java IDE by JetBrains,” n.d.), καθώς και για βάση δεδομένων θα χρησιμοποιήσουμε την MariaDB Srv. Ver. 10.5. (“MariaDB Foundation - MariaDB.org,” n.d.)

3.2 Δημιουργία του αρχικού Backend Project διαμέσου του Spring Initializr

Το Spring Initializr είναι ένα διαδικτυακό εργαλείο που χρησιμοποιείται για τη δημιουργία έργων Spring Boot. Το Spring Initializr μπορούμε να το βρούμε στη διεύθυνση **https://start.spring.io** (“Spring Initializr,” n.d.)

Συνεπώς, πηγαίνοντας στην παραπάνω διεύθυνση θα δούμε την ακόλουθη οθόνη:



Εικόνα 1

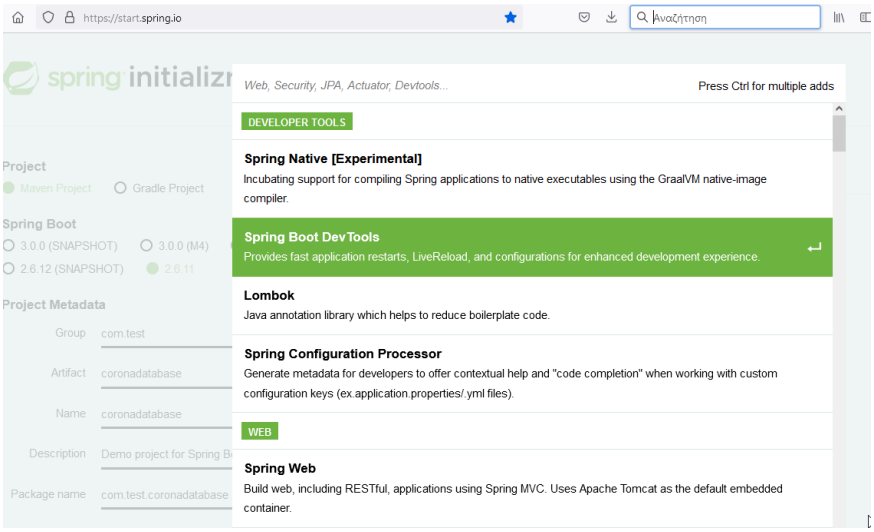
Θα δημιουργήσουμε ένα έργο Maven* με Java και θα επιλέξουμε μια από τις εκδόσεις του Spring Boot.

* Τα βασικά στοιχεία του Maven

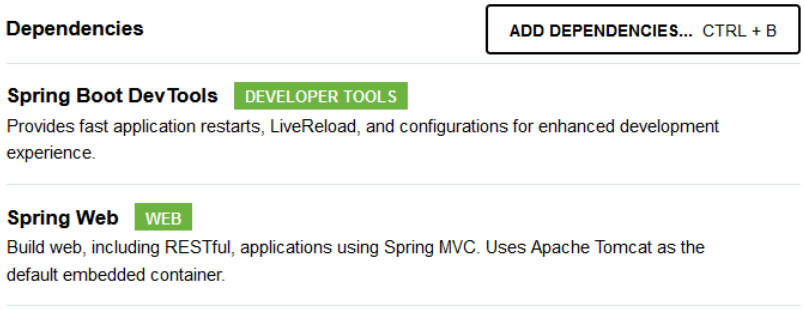
Το Apache Maven είναι ένα εργαλείο διαχείρισης έργων λογισμικού. Η βάση του Maven είναι το μοντέλο αντικειμένων έργου (Project Object Model - POM). Το Maven καθιστά τη διαδικασία ανάπτυξης λογισμικού απλούστερη και ενοποιεί επίσης τη διαδικασία ανάπτυξης. Το POM είναι ένα αρχείο pom.xml που περιέχει βασικές πληροφορίες για το έργο. Υπάρχουν επίσης όλες οι εξαρτήσεις που πρέπει να κατεβάσει το Maven για να μπορέσει να κατασκευάσει το έργο.Το maven διατίθεται ως αυτόνομο λογισμικό , αλλά υπάρχει και ενσωματωμένο στο IntelliJ.

Στο πεδίο Group, θα ορίσουμε το αναγνωριστικό ομάδας, το οποίο μετέπειτα θα γίνει το βασικό πακέτο στο έργο μας Java. Στο πεδίο Artifact, θα ορίσουμε το artifact ID, το οποίο θα είναι και το όνομα του project/έργου μας στο intellij.

Στην ενότητα Dependencies (Εξαρτήσεις), θα επιλέξουμε τους starters/εκκινητές και τις εξαρτήσεις που χρειάζονται στο έργο μας. Το Spring Boot παρέχει πακέτα εκκινητών που απλοποιούν τη διαμόρφωση του Maven. Τα πακέτα εκκινητών της Spring Boot είναι στην πραγματικότητα ένα σύνολο εξαρτήσεων που μπορούμε να συμπεριλάβουμε στο project/έργο. Θα προσθέσουμε στο έργο δύο εξαρτήσεις, τις Web και DevTools, ώστε το αρχικό μας έργο να ξεκινάει από αυτές.



Εικόνα 2

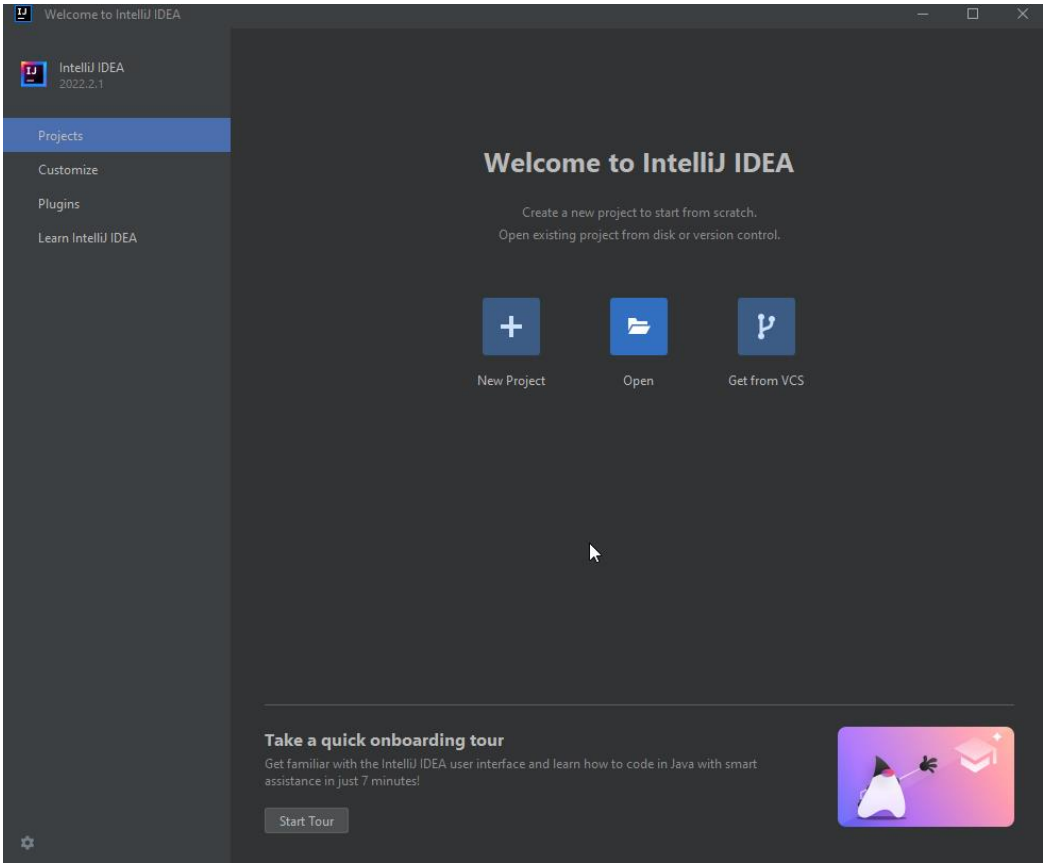


Εικόνα 3

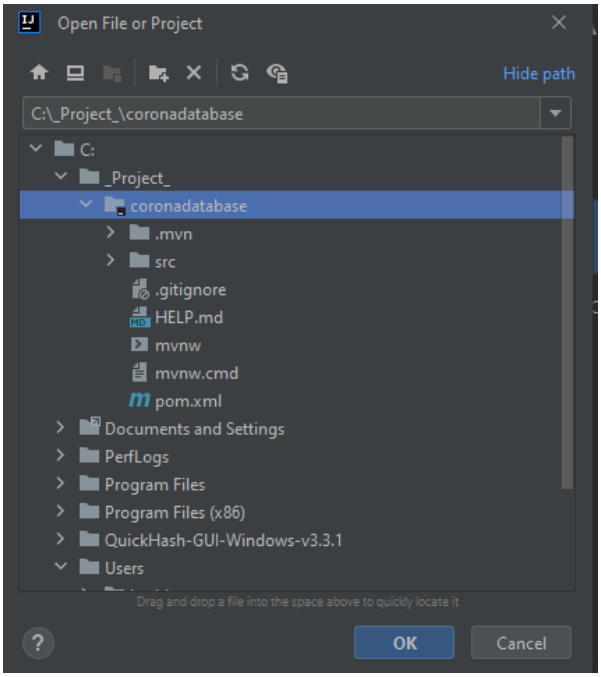
Η εξάρτηση DevTools μας παρέχει τα εργαλεία ανάπτυξης του Spring Boot, τα οποία παρέχουν λειτουργικότητα αυτόματης επανεκκίνησης.
Το πακέτο εκκίνησης web παρέχει ενσωματωμένο τον Tomcat και μας δίνει και Restful δυνατότητες.

Τέλος, κάνουμε κλικ στο κουμπί Generate Project (Δημιουργία έργου), το οποίο δημιουργεί το πακέτο ZIP του project για εμάς.

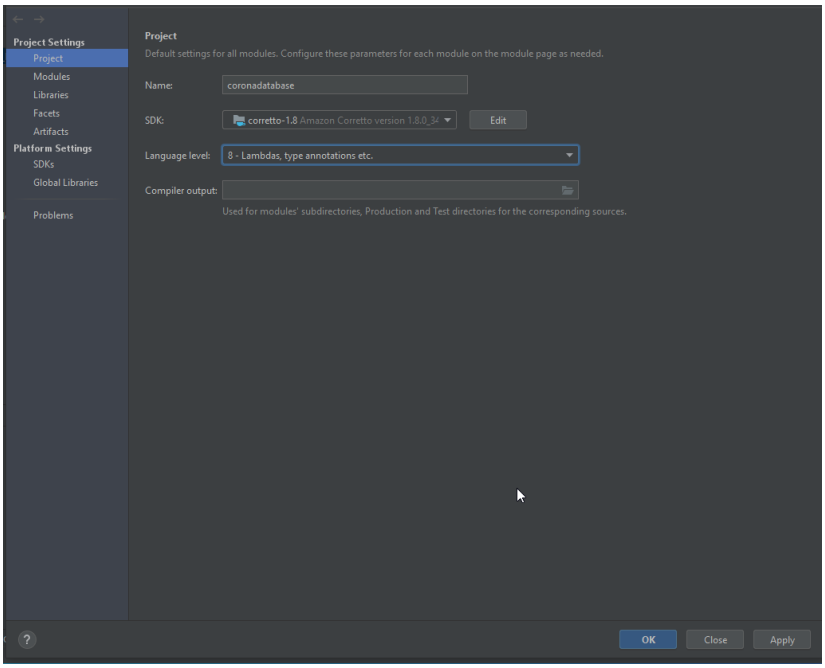
Εν συνέχεια, ανοίγουμε το IntelliJ Idea της JetBrains community edition και αφού αποσυμπιέσουμε το παραπάνω zip file το ανοίγουμε διαμέσου του κουμπιού Open και επιλέγουμε το φάκελο coronadatabase και επιλέγουμε το OK κουμπί.



Εικόνα 4



Εικόνα 5



Εικόνα 6

Επίσης, δεν ξεχνάμε να ρυθμίσουμε το project μας, ώστε να χρησιμοποιεί το Java SDK corretto 8.

3.3 Maven pom.xml αρχείο

Στη ρίζα του έργου μας, υπάρχει το αρχείο **pom.xml**, το οποίο είναι το αρχείο ρυθμίσεων του Maven για το έργο μας. Αν παρατηρήσουμε τις εξαρτήσεις μέσα στο αρχείο, μπορούμε να δούμε ότι υπάρχουν πλέον εξαρτήσεις που επιλέξαμε στη σελίδα Spring Initializr. Υπάρχει επίσης μια εξάρτηση δοκιμής που περιλαμβάνεται αυτόματα χωρίς καμία επιλογή. Στην διάρκεια αυτής της εργασίας θα προσθέσουμε περισσότερη λειτουργικότητα στην εφαρμογή μας προσθέτοντας και τις αντίστοιχες εξαρτήσεις χειροκίνητα στο αρχείο pom.xml:

```
<dependencies>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-devtools</artifactId>
<scope>runtime</scope>
</dependency>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-test</artifactId>
<scope>test</scope>
</dependency>
</dependencies>
```

3.4 Η κύρια κλάση του Spring Boot που δημιουργήθηκε μέχρι στιγμής ως σημείο έναρξης του Backend.

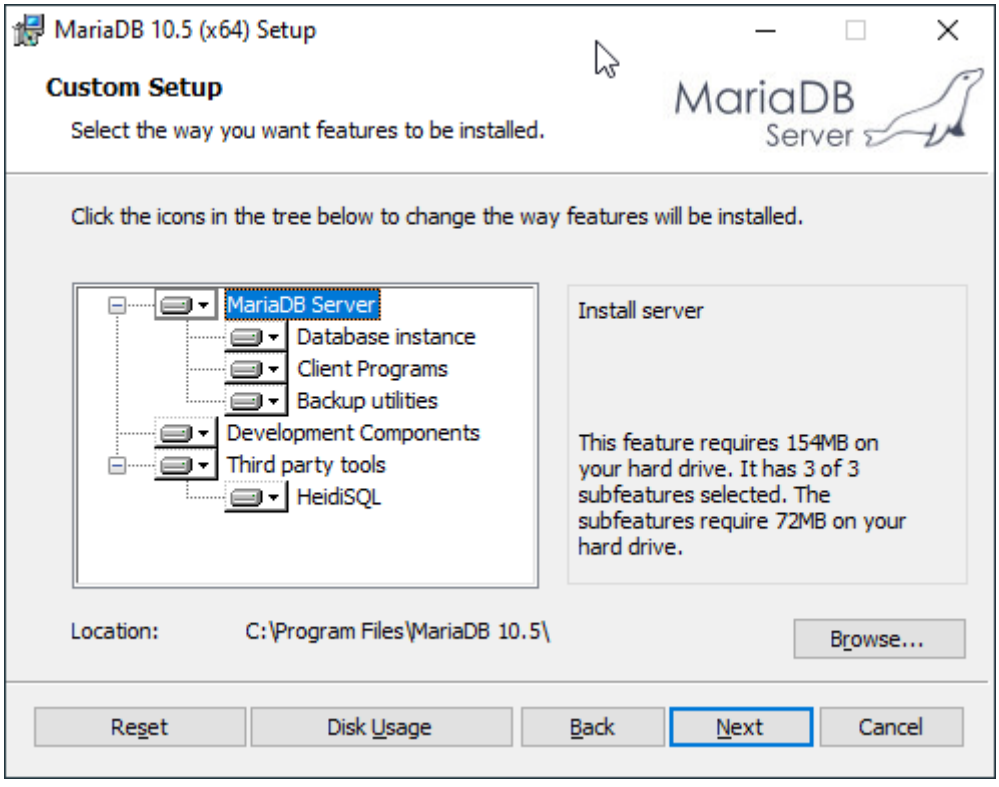
```
package com.test.coronadatabase;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication
public class CoronadatabaseApplication {
  public static void main(String[] args) {
    SpringApplication.run(CoronadatabaseApplication.class, args);
  }
}
```

3.5 Spring Boot & Apache Tomcat

Το Spring Boot χρησιμοποιεί τον ενσωματωμένο Apache Tomcat (http://tomcat.apache.org/) ως διακομιστή εφαρμογών από προεπιλογή. Από προεπιλογή, ο Tomcat εκτελείται στη θύρα 8080.

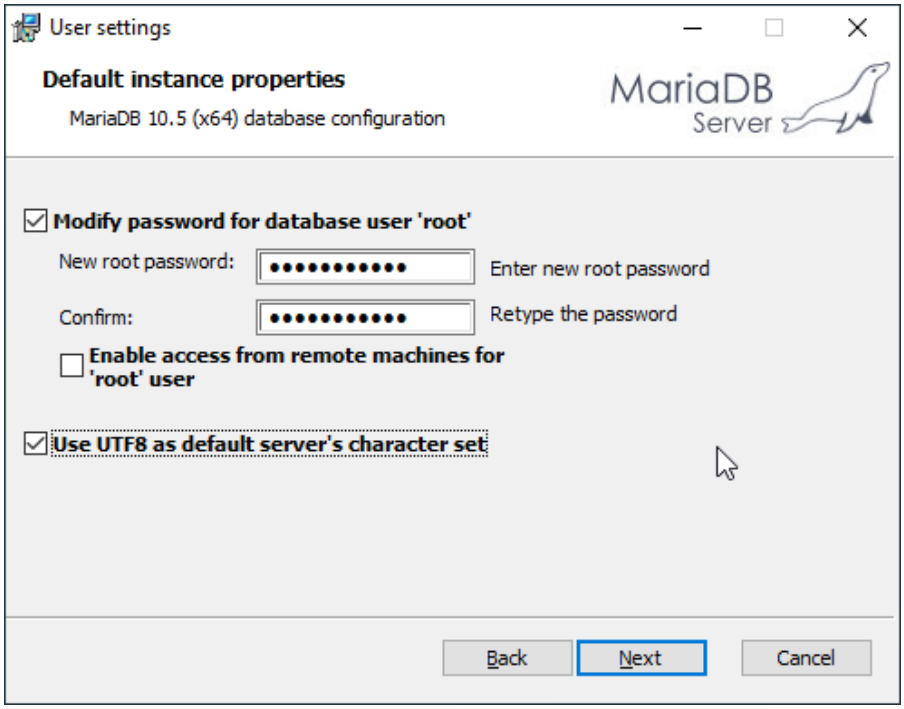
3.6 Εγκατάσταση της MariaDB

MariaDB είναι μια σχεσιακή βάση δεδομένων ανοικτού κώδικα και είναι διαθέσιμη για διάφορα Λειτουργικά Συστήματα.



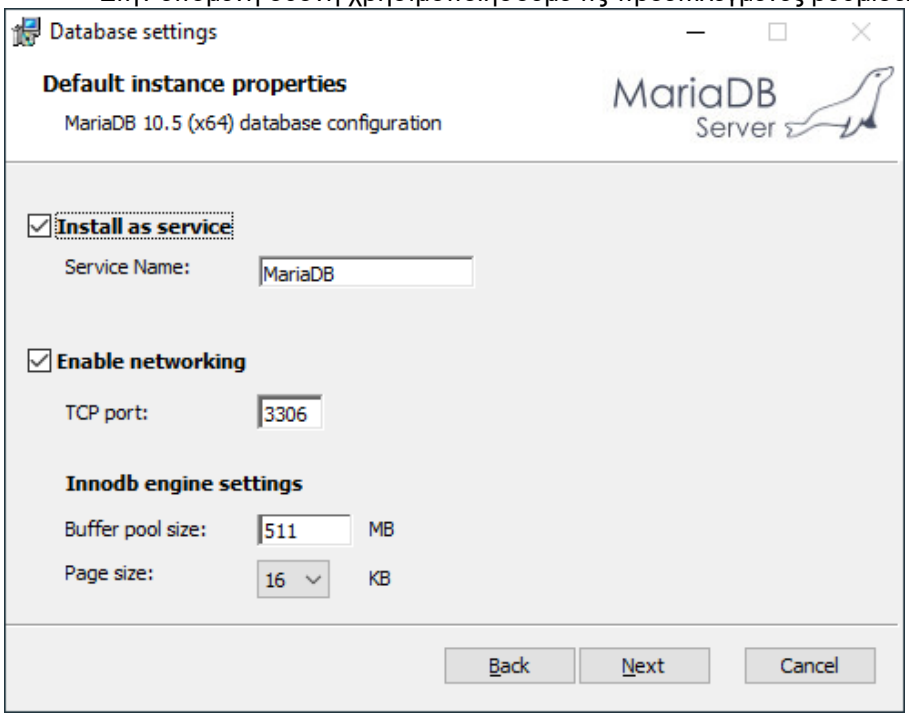
Εικόνα 7

Εν συνεχεία, δίνουμε έναν κωδικό πρόσβασης για τον χρήστη root. Ο κωδικός πρόσβασης είναι απαραίτητος όταν θα συνδεθεί η εφαρμογή μας με τη βάση δεδομένων:



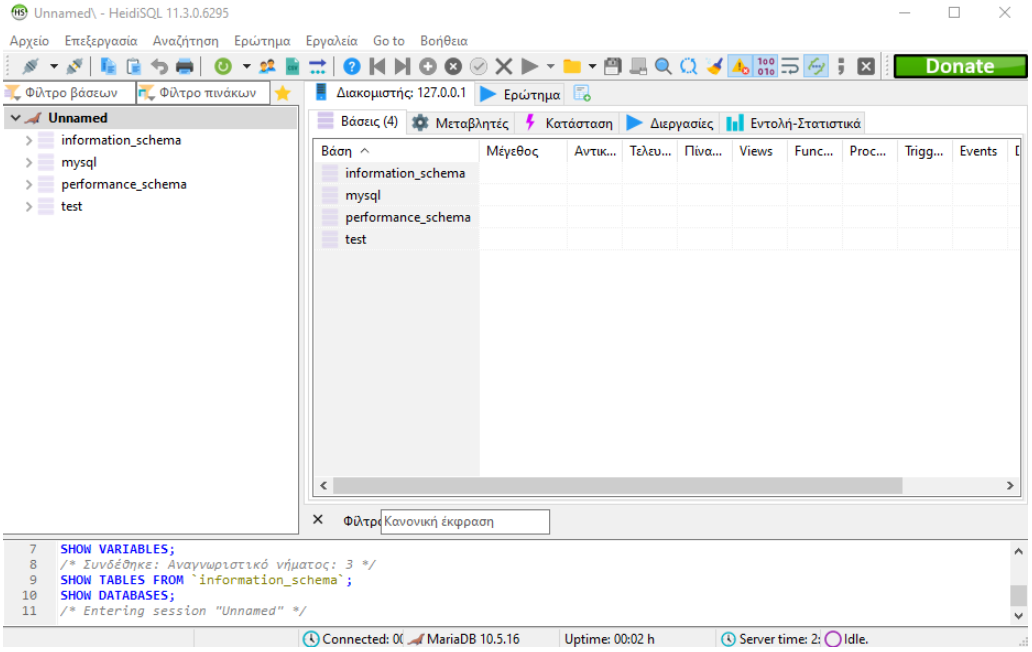
Εικόνα 8

Στην επόμενη οθόνη χρησιμοποιήσουμε τις προεπιλεγμένες ρυθμίσεις:



Εικόνα 9

Τώρα η εγκατάσταση θα ξεκινήσει και η MariaDB θα εγκατασταθεί στον τοπικό μας υπολογιστή. Επίσης, εγκαθιστάται και το HeidiSQL, το οποίο είναι ένα γραφικά εύχρηστο πρόγραμμα-πελάτη βάσης δεδομένων. Θα το χρησιμοποιήσουμε για να προσθέσουμε μια νέα βάση δεδομένων και να κάνουμε ερωτήματα στη βάση δεδομένων μας.



Εικόνα 10

3.7 Spring Boot και έννοιες όπως : ORM , JPA και Hibernate για τη δημιουργία και πρόσβαση σε μια βάση δεδομένων

3.7.1 Οι έννοιες: ORM , JPA, Hibernate

ORM/Object relational mapping

Η αντιστοίχιση αντικειμένου-σχεσιακής απεικόνισης (ORM, O/RM και εργαλείο αντιστοίχισης O/R) στην επιστήμη των υπολογιστών είναι μια τεχνική προγραμματισμού για τη μετατροπή δεδομένων μεταξύ συστημάτων τύπων με χρήση αντικειμενοστραφών γλωσσών προγραμματισμού. Με τον τρόπο αυτό δημιουργείται, στην πραγματικότητα, μια "εικονική βάση δεδομένων αντικειμένων" που μπορεί να χρησιμοποιηθεί μέσα από τη γλώσσα προγραμματισμού.
Στον αντικειμενοστραφή προγραμματισμό, οι εργασίες διαχείρισης δεδομένων ενεργούν σε αντικείμενα. (“Object–relational mapping - Wikipedia,” n.d.)

JPA/Java Persistence API

Το Java Persistence API παρέχει στους προγραμματιστές Java μια δυνατότητα αντιστοίχισης αντικειμένων/σχεσιακών δεδομένων για τη διαχείριση σχεσιακών δεδομένων σε εφαρμογές Java. (“Introduction to the Java Persistence API - The Java EE 6 Tutorial,” n.d.)

Hibernate

Το Hibernate ORM (ή απλά Hibernate) είναι ένα εργαλείο αντιστοίχισης αντικειμένου-σχεσιακής απεικόνισης για τη γλώσσα προγραμματισμού Java. Παρέχει ένα πλαίσιο για την αντιστοίχιση ενός αντικειμενοστραφούς μοντέλου σε μια σχεσιακή βάση δεδομένων. Το Hibernate αντιμετωπίζει τα προβλήματα αναντιστοιχίας εμπλοκής αντικειμένου-σχεσιακής σχέσης αντικαθιστώντας τις άμεσες, μόνιμες προσβάσεις στη βάση δεδομένων με λειτουργίες χειρισμού αντικειμένων υψηλού επιπέδου. Το Hibernate είναι ελεύθερο λογισμικό που διανέμεται υπό την GNU Lesser General Public License 2.1.
Το κύριο χαρακτηριστικό του Hibernate είναι η αντιστοίχιση από κλάσεις Java σε πίνακες βάσης δεδομένων και η αντιστοίχιση από τύπους δεδομένων Java σε τύπους δεδομένων SQL. Το Hibernate παρέχει επίσης δυνατότητες αναζήτησης και ανάκτησης δεδομένων. Δημιουργεί κλήσεις SQL και απαλλάσσει τον προγραμματιστή από το χειροκίνητο χειρισμό και τη μετατροπή αντικειμένων του συνόλου των αποτελεσμάτων. (“Hibernate (framework) - Wikipedia,” n.d.)

3.7.2 Spring Boot Hibernate και H2

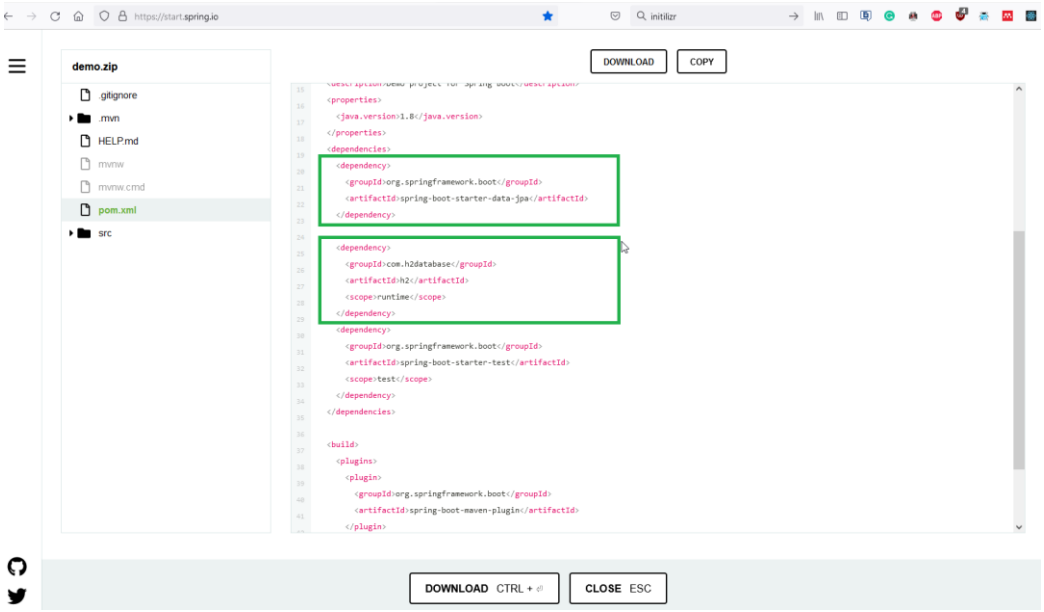
Θα χρησιμοποιήσουμε τη βάση δεδομένων μνήμης: H2* για σκοπούς ανάπτυξης και επίδειξης.

* Η βάση δεδομένων H2
Τα κύρια χαρακτηριστικά της H2 είναι:

- Πολύ γρήγορη και ανοικτού κώδικα, JDBC API
- Ενσωματωμένες λειτουργίες και λειτουργίες διακομιστή- βάσεις δεδομένων στη μνήμη
- Εφαρμογή κονσόλας με βάση το πρόγραμμα περιήγησης
- Μικρό αποτύπωμα: περίπου 2,5 MB μέγεθος αρχείου jar

(“H2 Database Engine,” n.d.)

Για να μπορέσουμε να χρησιμοποιήσουμε την JPA και τη βάση δεδομένων H2, πρέπει να προσθέσουμε τις ακόλουθες εξαρτήσεις στο αρχείο pom.xml:
Τις συγκεκριμένες εξαρτήσεις μπορούμε να τις εξάγουμε μέσα από την σελίδα https://start.spring.io/, εφόσον επιλέξουμε για δημιουργία maven Project Java Version 8 και προσθέσουμε jpa & H2 και δούμε την δομή του pom.xml πατώντας το κουμπί explorer πχ:



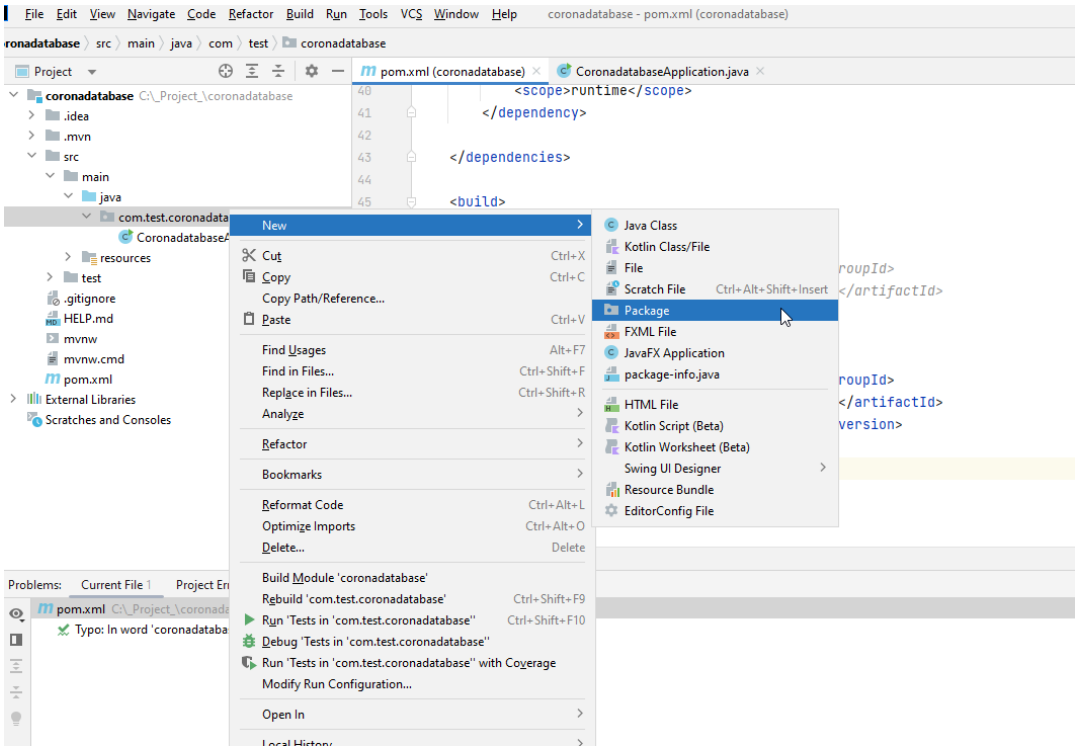
Εικόνα 11

Οπότε προσθέτουμε τα ακόλουθα στο αρχείο pom.xml:

```
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>

<groupId>com.h2database</groupId>
<artifactId>h2</artifactId>
<scope>runtime</scope>
</dependency>
```

Εν συνεχεία, ακολουθούν τα βήματα για τη δημιουργία entity classes/ κλάσεων οντοτήτων για χρήση της H2:

1. Για να δημιουργήσουμε μια entity class/κλάση οντοτήτων στο Spring Boot, θα δημιουργήσουμε αρχικά το δικό μας πακέτο για τις οντότητες/entites. Το πακέτο θα πρέπει να δημιουργηθεί κάτω από το root package με ονομασία domain.



Εικόνα 12

2. Θα ονομάσουμε το πακέτο μας com.test.coronadatabase.domain

3. Στη συνέχεια, δημιουργούμε την entity class. Το όνομα της κλάσης οντότητας είναι Patient.

4. Έπειτα, πάμε στην πακέτο domain στην κλάση Patient και επισημάνουμε την κλάση με την σήμανση @Entity. Αυτό μας δίνεται ως δυνατότητα μέσα από το πακέτο javax.persistence:

```
package com.test.coronadatabase.domain;
import javax.persistence.Entity;

@Entity
public class Patient {
}
```

5. Στο επόμενο βήμα, προσθέτουμε τα πεδία/fields στην κλάση μας. Τα πεδία της κλάσης οντοτήτων αντιστοιχίζονται σε στήλες πίνακα της βάσης δεδομένων και είναι αυτά που θα έχει το σύστημα καταγραφής κρουσμάτων κορονοϊού. Η κλάση οντότητας patient θα πρέπει επίσης να περιέχει ένα μοναδικό αναγνωριστικό που χρησιμοποιείται ως πρωτεύον κλειδί στη βάση δεδομένων:

```
package com.test.coronadatabase.domain;

import javax.persistence.Entity;
```

```
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Patient {
    @Id
    @GeneratedValue(strategy= GenerationType.AUTO)
    private long id;
    private String lastName, firstName, birthDate, city,address, telephone;;
    private int patientAmka, postcode;
    private boolean coronaPatient;
}
```

Το πρωτεύον κλειδί ορίζεται με τη χρήση της σημείωσης @Id.
Ο σχολιασμός @GeneratedValue ορίζει ότι το αναγνωριστικό παράγεται αυτόματα από τη βάση δεδομένων. Μπορούμε επίσης να ορίσουμε τη στρατηγική δημιουργίας του κλειδιού μας. Ο τύπος AUTO σημαίνει ότι αυτός που το παρέχει εν προκειμένου το JPA επιλέγει την καλύτερη στρατηγική για μια συγκεκριμένη βάση δεδομένων. Τα υπόλοιπα πεδία/fields της κλάσης θα είναι οι στήλες του πίνακα Patient που θα δημιουργηθεί στην βάση δεδομένων H2 και μετέπειτα στην MariaDB.

6. Τέλος, προσθέτουμε getters, setters και constructors στην Patient entity κλάση. Δεν χρειαζόμαστε ένα πεδίο ID στον κατασκευαστή λόγω της αυτόματης δημιουργίας ID.
Ο πηγαίος κώδικας των κατασκευαστών της κλάσης οντότητας Patient έχει ως εξής:

```
package com.test.coronadatabase.domain;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Patient {
    @Id
    @GeneratedValue(strategy= GenerationType.AUTO)
    private long id;
    private String lastName, firstName, birthDate, city,address, telephone;
    private int patientAmka, postcode;
    private boolean coronaPatient;

    public Patient() {
    }

    public Patient(String lastName, String firstName, String birthDate, String city, String address,
int patientAmka, int postcode, String telephone, boolean coronaPatient) {
        this.lastName = lastName;
        this.firstName = firstName;
        this.birthDate = birthDate;
        this.city = city;
        this.address = address;
        this.patientAmka = patientAmka;
        this.postcode = postcode;
```

```
        this.telephone = telephone;
        this.coronaPatient = coronaPatient;
    }
```

Ακολουθεί ο πηγαίος κώδικας των getters και setters της entity κλάσης **Patient**:

```
public String getLastName() {
        return lastName;
    }

    public String getFirstName() {
        return firstName;
    }

    public String getBirthDate() {
        return birthDate;
    }

    public String getCity() {
        return city;
    }

    public String getAddress() {
        return address;
    }

    public int getPatientAmka() {
        return patientAmka;
    }

    public int getPostcode() {
        return postcode;
    }

    public String getTelephone() {
        return telephone;
    }

    public boolean isCoronaPatient() {
        return coronaPatient;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public void setBirthDate(String birthDate) {
        this.birthDate = birthDate;
    }
```

```
    public void setCity(String city) {
        this.city = city;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    public void setPatientAmka(int patientAmka) {
        this.patientAmka = patientAmka;
    }

    public void setPostcode(int postcode) {
        this.postcode = postcode;
    }

    public void setTelephone(String telephone) {
        this.telephone = telephone;
    }

    public void setCoronaPatient(boolean coronaPatient) {
        this.coronaPatient = coronaPatient;
    }
    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
    }

}
```

Μετά τα παραπάνω και με την έναρξη της Backend εφαρμογής ο πίνακας με την ονομασία Patient θα δημιουργηθεί. Αυτό μπορούμε να το ελέγξουμε αν προσθέσουμε μια νέα γραμμή στο *αρχείο* ***application.properties,*** η οποία θα μας επιτρέψει να καταγράφουμε τις εντολές SQL στην κονσόλα:

spring.jpa.show-sql=true

Τώρα μπορούμε να δούμε τις δηλώσεις δημιουργίας πίνακα κατά την εκτέλεση της εφαρμογής:

```
2022-08-21 15:35:53.870  INFO 3708 --- [  restartedMain] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
2022-08-21 15:35:54.020  INFO 3708 --- [  restartedMain] o.s.b.a.h2.H2ConsoleAutoConfiguration    : H2 console available at '/h2-console'. Database available at 'jdbc:h2:mem:90f4b196-d5b4-4243-ac
2022-08-21 15:35:55.209  INFO 3708 --- [  restartedMain] o.hibernate.jpa.internal.util.LogHelper  : HHH000204: Processing PersistenceUnitInfo [name: default]
2022-08-21 15:35:55.524  INFO 3708 --- [  restartedMain] org.hibernate.Version                    : HHH000412: Hibernate ORM core version 5.4.27.Final
2022-08-21 15:35:56.256  INFO 3708 --- [  restartedMain] o.hibernate.annotations.common.Version   : HCANN000001: Hibernate Commons Annotations {5.1.2.Final}
2022-08-21 15:35:56.909  INFO 3708 --- [  restartedMain] org.hibernate.dialect.Dialect            : HHH000400: Using dialect: org.hibernate.dialect.H2Dialect
Hibernate: drop table if exists patient CASCADE
Hibernate: drop sequence if exists hibernate_sequence
Hibernate: create sequence hibernate_sequence start with 1 increment by 1
Hibernate: create table patient (id bigint not null, address varchar(255), birth_date varchar(255), city varchar(255), corona_patient boolean not null, first_name varchar(255), last_name varchar(
2022-08-21 15:35:59.694  INFO 3708 --- [  restartedMain] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.int
2022-08-21 15:35:59.724  INFO 3708 --- [  restartedMain] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2022-08-21 15:35:59.816  INFO 3708 --- [  restartedMain] o.s.b.d.a.OptionalLiveReloadServer       : LiveReload server is running on port 35729
2022-08-21 15:36:00.167  WARN 3708 --- [  restartedMain] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may be performed dur
2022-08-21 15:36:00.663  INFO 3708 --- [  restartedMain] o.s.s.concurrent.ThreadPoolTaskExecutor  : Initializing ExecutorService 'applicationTaskExecutor'
2022-08-21 15:36:02.072  INFO 3708 --- [  restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path ''
2022-08-21 15:36:02.119  INFO 3708 --- [  restartedMain] c.t.c.CoronadatabaseApplication          : Started CoronadatabaseApplication in 21.199 seconds (JVM running for 24.117)
```

Εικόνα 13

Η βάση δεδομένων H2 παρέχει μια κονσόλα βασισμένη στο διαδίκτυο που μπορεί να χρησιμοποιηθεί για την εξερεύνηση μιας βάσης δεδομένων και την εκτέλεση δηλώσεων SQL. Για να ενεργοποιήσουμε την κονσόλα, πρέπει να προσθέσουμε τις ακόλουθες γραμμές στο αρχείο **application.properties.**

```
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console
```

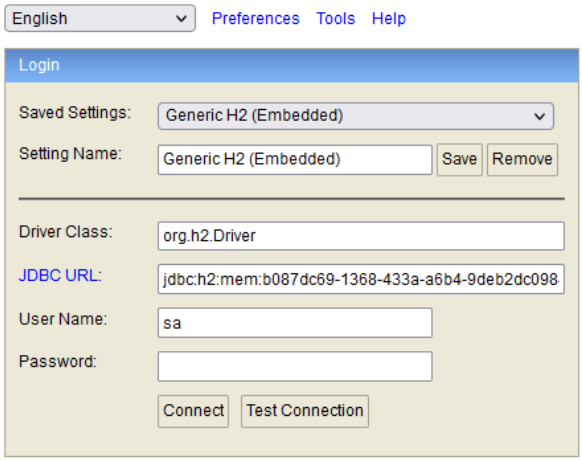
Εικόνα 14

Η πρώτη ρύθμιση ενεργοποιεί την κονσόλα H2 , ενώ η δεύτερη ορίζει το τελικό σημείο που μπορούμε να χρησιμοποιήσουμε για πρόσβαση στην κονσόλα.

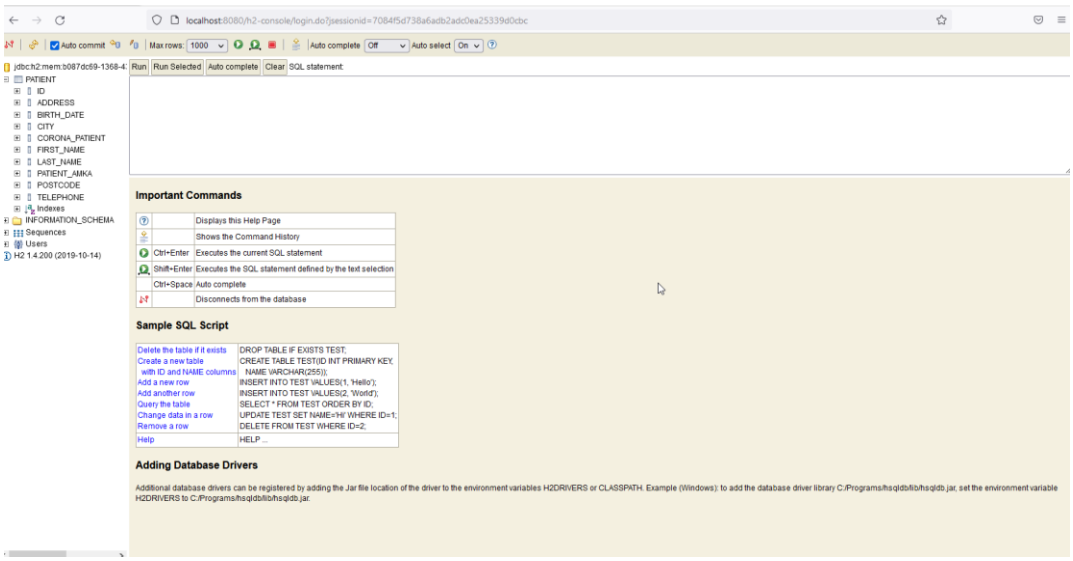
Συνεπώς, τώρα μπορούμε να αποκτήσουμε πρόσβαση στην κονσόλα H2 με πλοήγηση στο **localhost:8080/h2-console** με το πρόγραμμα περιήγησης ιστού.
Χρησιμοποιούμε το αλφαριθμητικό που δίνει η console του IntelliJ *Database available at '**jdbc:h2:mem:b087dc69-1368-433a-a6b4-9deb2dc09847**'* ως διεύθυνση URL JDBC και αφήνουμε το πεδίο Password κενό στο παράθυρο Login και πατάμε το κουμπί Connect (Σύνδεση) για να συνδεθούμε στην βάση, όπως φαίνεται στα ακόλουθα στιγμιότυπα οθόνης:

```
0:21.883  INFO 4324 --- [  restartedMain] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.41]
0:22.340  INFO 4324 --- [  restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
0:22.341  INFO 4324 --- [  restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 6905 ms
0:22.758  INFO 4324 --- [  restartedMain] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
0:23.591  INFO 4324 --- [  restartedMain] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
0:23.627  INFO 4324 --- [  restartedMain] o.s.b.a.h2.H2ConsoleAutoConfiguration    : H2 console available at '/h2-console'. Database available at 'jdbc:h2:mem:b087dc69-1368-433a-a6b4-9deb2dc09847'
0:25.038  INFO 4324 --- [  restartedMain] o.hibernate.jpa.internal.util.LogHelper  : HHH000204: Processing PersistenceUnitInfo [name: default]
0:25.358  INFO 4324 --- [  restartedMain] org.hibernate.Version                    : HHH000412: Hibernate ORM core version 5.4.27.Final
0:26.032  INFO 4324 --- [  restartedMain] o.hibernate.annotations.common.Version   : HCANN000001: Hibernate Commons Annotations {5.1.2.Final}
0:26.760  INFO 4324 --- [  restartedMain] org.hibernate.dialect.Dialect            : HHH000400: Using dialect: org.hibernate.dialect.H2Dialect
 table if exists patient CASCADE
 sequence if exists hibernate_sequence
te sequence hibernate_sequence start with 1 increment by 1
te table patient (id bigint not null, address varchar(255), birth_date varchar(255), city varchar(255), corona_patient boolean not null, first_name varchar(255), last_name varchar(255), patient_a
0:29.665  INFO 4324 --- [  restartedMain] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlat
0:29.697  INFO 4324 --- [  restartedMain] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
0:29.744  INFO 4324 --- [  restartedMain] o.s.b.d.a.OptionalLiveReloadServer       : LiveReload server is running on port 35729
```

Εικόνα 15



Εικόνα 16



Εικόνα 17

Επομένως, τώρα, βλέπουμε τον πίνακα Patient στη βάση δεδομένων H2.

Εν συνεχεία, θα αναφέρουμε την διαδικασία δημιουργίας CRUD repository / αποθετηρίου. Το Spring Boot Data JPA μας παρέχει μια διεπαφή / interface CrudRepository για λειτουργίες CRUD (CREATE, READ, UPDATE,DELETE.) στην βάση δεδομένων.

Θα δημιουργήσουμε τώρα το αποθετήριό μας στο πακέτο domain ως εξής:
1. Αρχικά, δημιουργούμε μια νέα κλάση με όνομα PatientRepository στο πακέτο domain.

```
package com.test.coronadatabase.domain;

import org.springframework.data.repository.CrudRepository;


@RepositoryRestResource
public interface PatientRepository extends CrudRepository<Patient, Long> {

}
```

Το PatientRepository μας επεκτείνει τώρα το interface/διεπαφή CrudRepository της Spring Boot JPA.
Τα ορίσματα τύπου <Patient, Long> ορίζουν ότι πρόκειται για το αποθετήριο της κλάσης οντοτήτων Patient και ότι ο τύπος του πεδίου ID είναι Long.
(“Spring Data - CrudRepository save() Method | Baeldung,” n.d.)

2. Έπειτα, θα προσθέσουμε κάποια δεδομένα επίδειξης στη βάση δεδομένων H2. Για το σκοπό αυτό, θα χρησιμοποιήσουμε τη διεπαφή/interface Spring Boot CommandLineRunner. Η διεπαφή CommandLineRunner μας επιτρέπει να εκτελούμε πρόσθετο κώδικα πριν από την

πλήρη εκκίνηση της εφαρμογής. Ως εκ τούτου, είναι ένα καλό σημείο για να προσθέσουμε δεδομένα επίδειξης στη βάση δεδομένων.
(“Spring Boot CommandLineRunner - running beans with CommandLineRunner,” n.d.)

Η διεπαφή CommandLineRunner βρίσκεται μέσα στην main class/ κύρια κλάση:

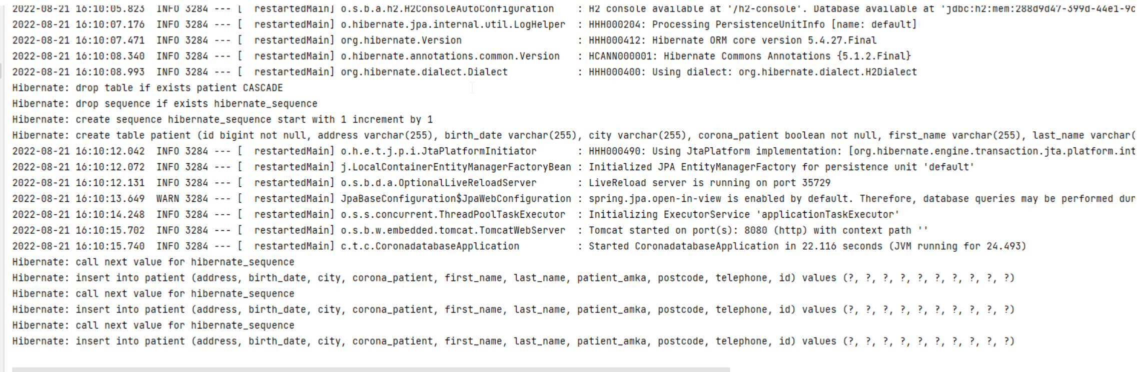
```
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;

@SpringBootApplication
public class CoronadatabaseApplication {

public static void main(String[] args) {
SpringApplication.run(CoronadatabaseApplication.class, args);
}

@Bean

CommandLineRunner runner(){
return args -> {

};
}
}
```

3. Στη συνέχεια, πρέπει να εισάγουμε το patient repository/αποθετήριο ασθενών μας στην κύρια κλάση για να μπορούμε να αποθηκεύουμε νέα αντικείμενα ασθενών στη βάση δεδομένων. Μια σημείωση @Autowired χρησιμοποιείται για να ενεργοποιηθεί η dependency injection/ έγχυση εξάρτησης. Η έγχυση εξαρτήσεων μας επιτρέπει να περάσουμε εξαρτήσεις σε ένα αντικείμενο. Αφού έχουμε εγχύσει την κλάση αποθετηρίου, μπορούμε να χρησιμοποιήσουμε τις μεθόδους CRUD που παρέχει. Το ακόλουθο παράδειγμα κώδικα φανερώνει πώς να εισάγουμε μερικούς ασθενείς στη βάση δεδομένων:

```
package com.test.coronadatabase;

import com.test.coronadatabase.domain.Patient;
import com.test.coronadatabase.domain.PatientRepository;
import com.test.coronadatabase.domain.User;
import com.test.coronadatabase.domain.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;

@SpringBootApplication

public class CoronadatabaseApplication {
        @Autowired
        private PatientRepository repository;
```

```
	public static void main(String[] args) {

		SpringApplication.run(CoronadatabaseApplication.class, args);
	}
	@Bean
	CommandLineRunner runner(){
		return args -> {
			// Save demo data to database
			repository.save(new Patient("LastName1","FirstName1","28/9/1975",

	"Athens","SomePlace1",1111111,18344,"2109784456",false));
			repository.save(new Patient("LastName2","FirstName2","28/9/1980",

	"Athens","SomePlace2",1111456,18380,"2109784456",false));
			repository.save(new Patient("LastName3","FirstName3","28/9/2000",

	"Athens","SomePlace3",1111111,18344,"2109784456",false));
		};
	}
}
```

Οι εντολές Insert εμφανίζονται στην κονσόλα του IntelliJ μετά την εκτέλεση της εφαρμογής:



```
2022-08-21 16:10:05.823  INFO 3284 --- [  restartedMain] o.s.b.a.h2.H2ConsoleAutoConfiguration    : H2 console available at '/h2-console'. Database available at 'jdbc:h2:mem:288d9d47-399d-44e1-9c
2022-08-21 16:10:07.176  INFO 3284 --- [  restartedMain] o.hibernate.jpa.internal.util.LogHelper  : HHH000204: Processing PersistenceUnitInfo [name: default]
2022-08-21 16:10:07.471  INFO 3284 --- [  restartedMain] org.hibernate.Version                    : HHH000412: Hibernate ORM core version 5.4.27.Final
2022-08-21 16:10:08.340  INFO 3284 --- [  restartedMain] o.hibernate.annotations.common.Version   : HCANN000001: Hibernate Commons Annotations {5.1.2.Final}
2022-08-21 16:10:08.993  INFO 3284 --- [  restartedMain] org.hibernate.dialect.Dialect            : HHH000400: Using dialect: org.hibernate.dialect.H2Dialect
Hibernate: drop table if exists patient CASCADE
Hibernate: drop sequence if exists hibernate_sequence
Hibernate: create sequence hibernate_sequence start with 1 increment by 1
Hibernate: create table patient (id bigint not null, address varchar(255), birth_date varchar(255), city varchar(255), corona_patient boolean not null, first_name varchar(255), last_name varchar(
2022-08-21 16:10:12.042  INFO 3284 --- [  restartedMain] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.int
2022-08-21 16:10:12.072  INFO 3284 --- [  restartedMain] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2022-08-21 16:10:12.131  INFO 3284 --- [  restartedMain] o.s.b.d.a.OptionalLiveReloadServer       : LiveReload server is running on port 35729
2022-08-21 16:10:13.649  WARN 3284 --- [  restartedMain] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may be performed dur
2022-08-21 16:10:14.248  INFO 3284 --- [  restartedMain] o.s.s.concurrent.ThreadPoolTaskExecutor  : Initializing ExecutorService 'applicationTaskExecutor'
2022-08-21 16:10:15.702  INFO 3284 --- [  restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path ''
2022-08-21 16:10:15.740  INFO 3284 --- [  restartedMain] c.t.c.CoronadatabaseApplication          : Started CoronadatabaseApplication in 22.116 seconds (JVM running for 24.493)
Hibernate: call next value for hibernate_sequence
Hibernate: insert into patient (address, birth_date, city, corona_patient, first_name, last_name, patient_amka, postcode, telephone, id) values (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
Hibernate: call next value for hibernate_sequence
Hibernate: insert into patient (address, birth_date, city, corona_patient, first_name, last_name, patient_amka, postcode, telephone, id) values (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
Hibernate: call next value for hibernate_sequence
Hibernate: insert into patient (address, birth_date, city, corona_patient, first_name, last_name, patient_amka, postcode, telephone, id) values (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
```

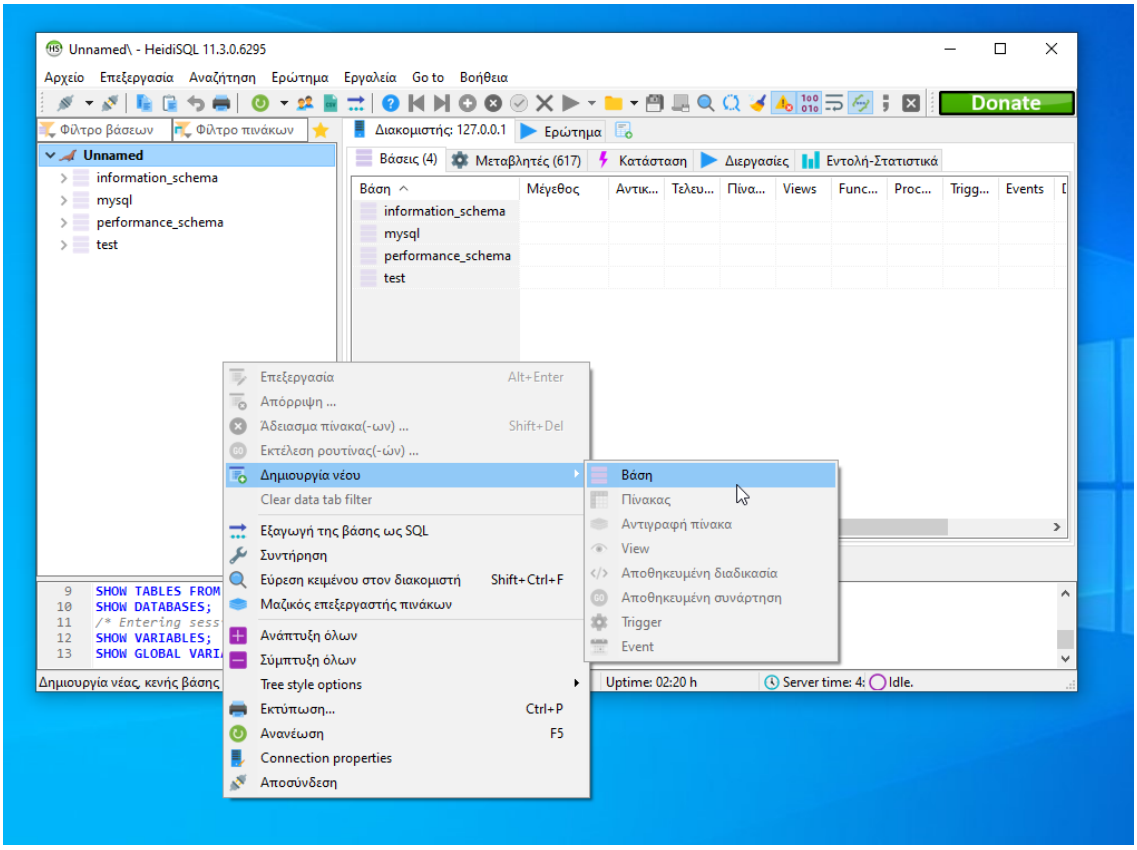
Εικόνα 18

Μπορούμε, επίσης, να χρησιμοποιήσουμε την κονσόλα H2 για να ανακτήσουμε ασθενείς από τη βάση δεδομένων, διαμέσου ερωτημάτων.

3.7.3 Spring Boot Hibernate και MariaDB

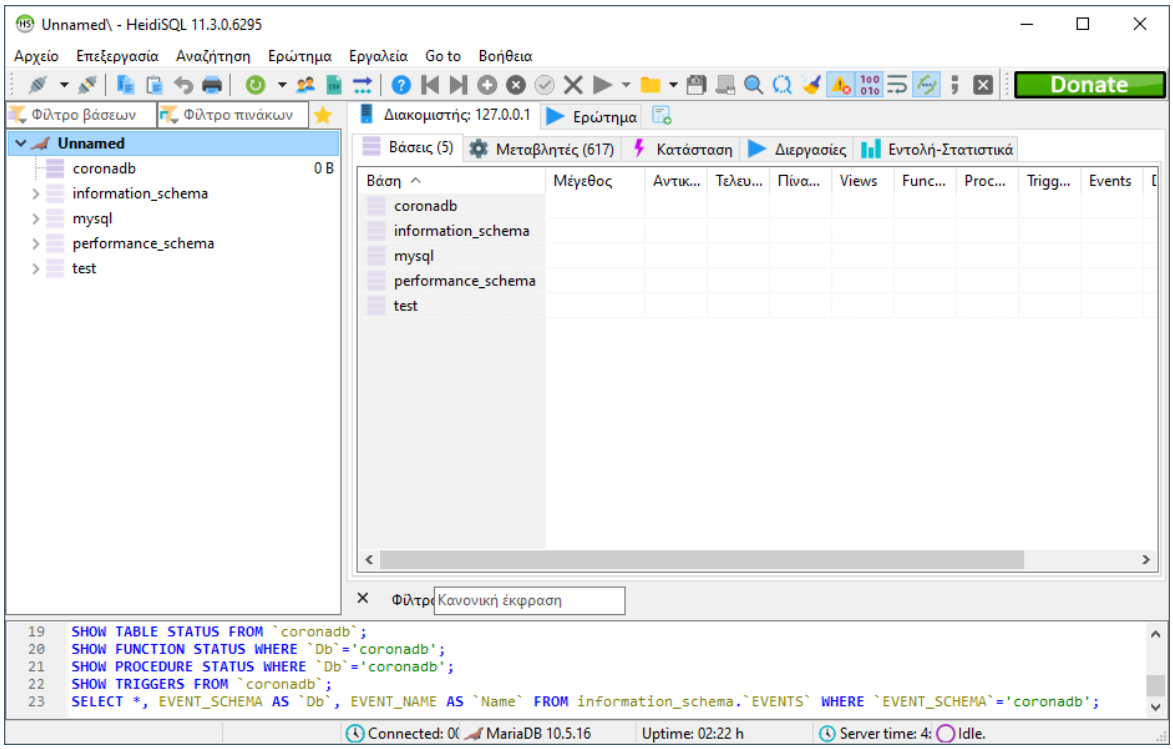
Εν συνεχεία, θα αλλάξουμε τη βάση δεδομένων από H2 σε MariaDB.
Οι πίνακες της βάσης δεδομένων εξακολουθούν να δημιουργούνται αυτόματα από το JPA.
Πριν τρέξουμε την εφαρμογή μας, πρέπει να δημιουργήσουμε μια βάση δεδομένων για αυτήν. Η βάση δεδομένων μπορεί να δημιουργηθεί με τη χρήση του HeidiSQL το οποίο ανοίγουμε και:

1. Κάνουμε δεξί κλικ με το ποντίκι μας μέσα στη λίστα με τις βάσεις δεδομένων και
2. Στη συνέχεια, επιλέγουμε Δημιουργία νέας / Βάση δεδομένων:



Εικόνα 19

3. Ονοματίζουμε την βάση δεδομένων μας **coronadb**.



Εικόνα 20

4. Στην εφαρμογή προσθέτουμε μια εξάρτηση MariaDB στο αρχείο pom.xml και αφαιρούμε το H2 dependency που δεν χρειαζόμαστε πλέον:

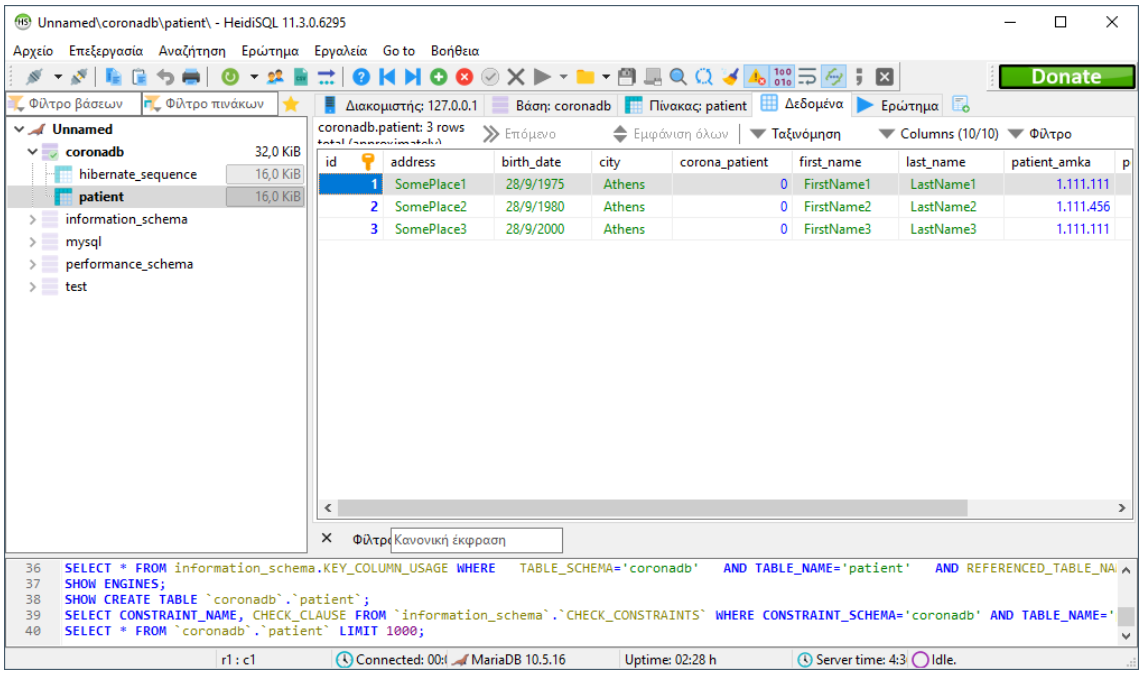
```
<dependency>
<groupId>org.mariadb.jdbc</groupId>
<artifactId>mariadb-java-client</artifactId>
</dependency>
```

5.Στο αρχείο application.properties, θα ορίσουμε τώρα τη σύνδεση με τη βάση δεδομένων. Αρχικά, θα οριστεί το url της βάσης δεδομένων, το όνομα χρήστη, ο κωδικός πρόσβασης και η κλάση του προγράμματος οδήγησης της βάσης δεδομένων.
Η ρύθμιση spring.jpa.generate-ddl ορίζει αν η JPA θα πρέπει να αρχικοποιήσει τη βάση δεδομένων (true/false).
Η ρύθμιση spring.jpa.hibernate.ddl-auto, ορίζει τη συμπεριφορά της αρχικοποίησης της βάσης δεδομένων:

```
spring.datasource.url=jdbc:mariadb://localhost:3306/coronadb
spring.datasource.username=root
spring.datasource.password=your passwd
spring.datasource.driver-class-name=org.mariadb.jdbc.Driver

spring.jpa.generate-ddl=true
spring.jpa.hibernate.ddl-auto=create-drop
```

(“Configuring Spring Boot for MariaDB - Spring Framework Guru,” n.d.)

6.Μετά την εκτέλεση της εφαρμογής, θα πρέπει να βλέπουμε τα τεστ δεδομένα του νεοδημιουργηθεντος πίνακα patient μέσα από την βάση coronadb στη MariaDB.



Εικόνα 21

3.8 Δημιουργώντας μια RESTful υπηρεσία ιστού με την Spring Boot

3.8.1 Τι είναι το REST

Το Representational State Transfer (REST) είναι ένα αρχιτεκτονικό στυλ λογισμικού που περιγράφει μια ομοιόμορφη διεπαφή μεταξύ φυσικά ξεχωριστών στοιχείων, συχνά μέσω του Διαδικτύου σε μια αρχιτεκτονική πελάτη-εξυπηρετητή.

Το REST ορίζει τέσσερις περιορισμούς διεπαφής:

- Προσδιορισμός των πόρων
- Χειρισμός πόρων
- Αυτοπεριγραφικά μηνύματα
- Υπερμέσα ως η μηχανή της κατάστασης της εφαρμογής

Το REST έχει χρησιμοποιηθεί σε ολόκληρη τη βιομηχανία λογισμικού και είναι ευρέως αποδεκτό ως ένα σύνολο κατευθυντήριων γραμμών για τη δημιουργία αξιόπιστων APIs εφαρμογών ιστού. Ένα web API που υπακούει στους περιορισμούς REST περιγράφεται ανεπίσημα ως RESTful. Σε γενικές γραμμές, τα RESTful web APIs βασίζονται σε μεθόδους HTTP όπως οι GET, POST, PUT και DELETE. Τα αιτήματα HTTP χρησιμοποιούνται για την πρόσβαση σε δεδομένα ή πόρους στην εφαρμογή ιστού μέσω κωδικοποιημένων παραμέτρων URL, οι οποίες γενικά μορφοποιούνται είτε ως JSON, είτε ως XML για τη μετάδοση των δεδομένων.

Οι "πόροι ιστού" ορίστηκαν για πρώτη φορά στον Παγκόσμιο Ιστό ως έγγραφα ή αρχεία που αναγνωρίζονται από τις διευθύνσεις URL τους. Σήμερα, ο ορισμός είναι πολύ πιο γενικός και αφηρημένος και περιλαμβάνει κάθε πράγμα, οντότητα ή ενέργεια που συνδέεται με το Διαδίκτυο, το τοπικό δίκτυο ή τη συσκευή. Κάθε συσκευή στο Διαδίκτυο διαθέτει ένα URI ή Uniform

Resource Identifier. Σε μια RESTful υπηρεσία ιστού, οι αιτήσεις που γίνονται στο URI ενός πόρου προκαλούν μια απάντηση με ένα ωφέλιμο φορτίο διαμορφωμένο σε HTML, XML, JSON ή κάποια άλλη μορφή. Το πιο συνηθισμένο πρωτόκολλο για αυτά τα αιτήματα και τις απαντήσεις είναι το HTTP, το οποίο παρέχει λειτουργίες (μεθόδους HTTP) όπως GET, POST, PUT, PATCH και DELETE. Με τη χρήση ενός stateless πρωτοκόλλου και τυποποιημένων λειτουργιών, τα RESTful συστήματα στοχεύουν στη γρήγορη απόδοση, την αξιοπιστία και τη δυνατότητα ανάπτυξης με την επαναχρησιμοποίηση στοιχείων που μπορούν να διαχειριστούν και να ενημερωθούν χωρίς να επηρεάσουν το σύστημα ως σύνολο, ακόμη και κατά τη διάρκεια της λειτουργίας του. (“Representational state transfer - Wikipedia,” n.d.)

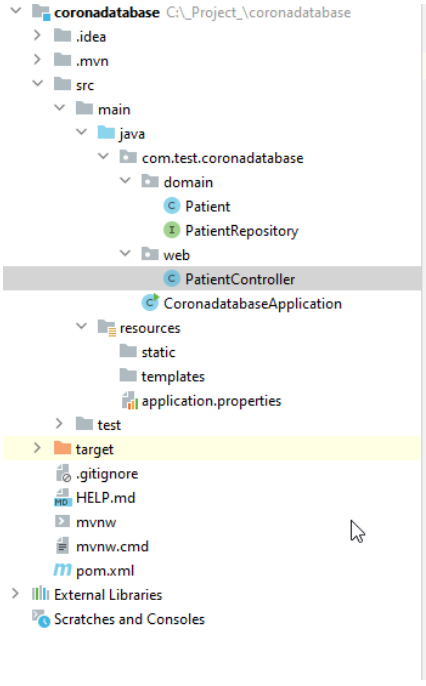
3.8.2 Δημιουργία μιας RESTful υπηρεσίας ιστού

Στο Spring Boot, όλα τα αιτήματα HTTP διαχειρίζονται από controller classes/ κλάσεις ελεγκτών, συνεπώς, προκειμένου να δημιουργήσουμε μια RESTful υπηρεσία ιστού, πρέπει πρώτα να δημιουργήσουμε μια controller class/κλάση ελεγκτή.
(“Tutorial | Building REST services with Spring,” n.d.)

Θα δημιουργήσουμε ένα νέο πακέτο για τον controller/ελεγκτή:
1. Θα ονομάσουμε το νέο πακέτο package com.test.coronadatabase.web

2. Δημιουργούμε μια νέα controller class/ κλάση ελεγκτή στο ένα νέο πακέτο web και την ονομάζουμε PatientController:

3. Τώρα, η δομή του έργου μας θα μοιάζει με το ακόλουθο στιγμιότυπο οθόνης:



Εικόνα 22

4. Στη συνέχεια, ανοίγουμε την συγκεκριμένη controller class/κλάση του ελεγκτή και προσθέτουμε το σχόλιο @RestController πριν από τον ορισμό της κλάσης. Ο σχολιασμός @RestController προσδιορίζει ότι αυτή η κλάση θα είναι ο ελεγκτής για την RESTful υπηρεσία ιστού:

```
package com.test.coronadatabase.web;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class PatientController {
}
```

5.Έπειτα, προσθέτουμε μια νέα μέθοδο/ getPetients() μέσα στην κλάση του ελεγκτή. Η μέθοδος είναι σχολιασμένη με το σχόλιο @RequestMapping, το οποίο ορίζει το τελικό σημείο στο οποίο αντιστοιχίζεται η μέθοδος. Σε παρακάτω κώδικα ουσιαστικά, όταν ένας χρήστης πλοηγείται στο τελικό σημείο /patients εκτελείται η μέθοδος getPetients():

```
package com.test.coronadatabase.web;

import com.test.coronadatabase.domain.Patient;
import com.test.coronadatabase.domain.PatientRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class PatientController {

@RequestMapping("/patients")
   public Iterable<Patient> getPetients() {

   }
}
```

Η μέθοδος getPetients() επιστρέφει όλα τα αντικείμενα ασθενών, τα οποία στη συνέχεια διαμορφώνονται σε αντικείμενα JSON από τη βιβλιοθήκη Jackson. Από προεπιλογή, η @RequestMapping χειρίζεται όλες τις αιτήσεις με τη μέθοδο HTTP (GET, PUT, POST και άλλες).

6.Για να μπορούμε να επιστρέψουμε ασθενείς από τη βάση δεδομένων, πρέπει να εισάγουμε το PatientRepository στον ελεγκτή. Στη συνέχεια, μπορούμε να χρησιμοποιήσουμε τη μέθοδο findAll() που παρέχει το αποθετήριο για να φέρουμε όλους τους ασθενείς.

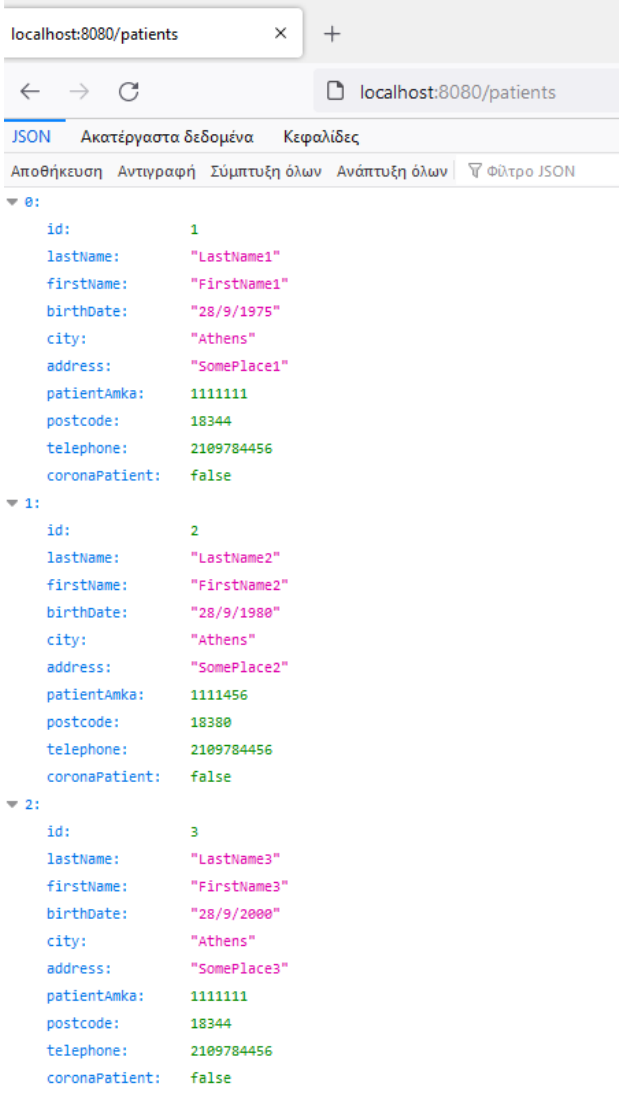
Ο παρακάτω πηγαίος κώδικας δείχνει τον κώδικα του ελεγκτή:

```
package com.test.coronadatabase.web;

import com.test.coronadatabase.domain.Patient;
import com.test.coronadatabase.domain.PatientRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class PatientController {

@Autowired
   private PatientRepository repository;

   @RequestMapping("/patients")
```

```
    public Iterable<Patient> getPetients() {
        return repository.findAll();
    }
}
```

7. Ακολούθως, διαμέσου ενός φυλλομετρητή πηγαίνουμε στη διεύθυνση localhost:8080/patients .

8. Και βλέπουμε ότι λαμβάνουμε όλους τους τεστ ασθενείς από τη βάση δεδομένων σε μορφή JSON, όπως φαίνεται στο ακόλουθο στιγμιότυπο οθόνης:



Εικόνα 23

3.8.3 Κάνοντας Χρήση του Spring Data REST

Το Spring Data REST αποτελεί μέρος του έργου Spring Data. Προσφέρει έναν εύκολο και γρήγορο τρόπο για την υλοποίηση RESTful υπηρεσιών ιστού με την Spring.
(“Spring Data REST,” n.d.)

Για να ξεκινήσουμε να χρησιμοποιούμε το Spring Data REST, πρέπει να προσθέσουμε την ακόλουθη εξάρτηση στο αρχείο pom.xml:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-data-rest</artifactId>
</dependency>
```

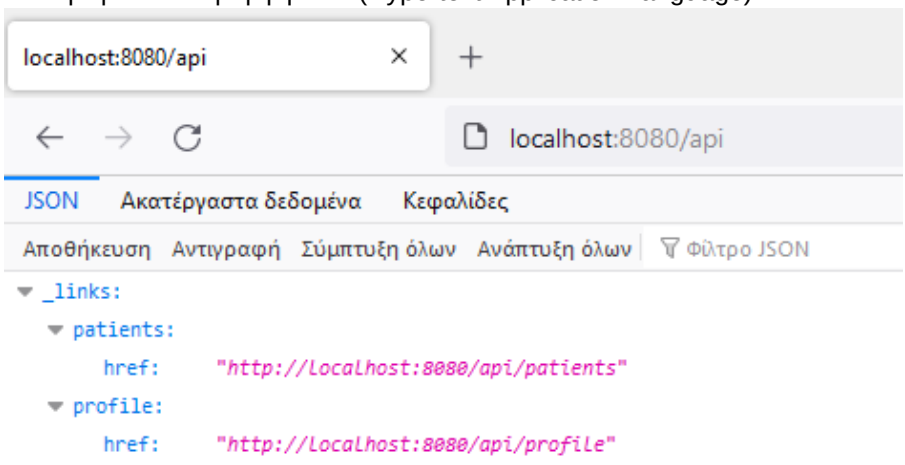
Από προεπιλογή, το Spring Data REST βρίσκει όλα τα δημόσια αποθετήρια/public repositories από την εφαρμογή και δημιουργεί αυτόματα RESTful υπηρεσίες ιστού για τις οντότητές μας.

Μπορούμε να ορίσουμε το τελικό σημείο της υπηρεσίας στο application.properties της εφαρμογής, ως εξής:

```
spring.data.rest.basePath=/api
```

Συνεπως, τώρα, μπορούμε να αποκτήσουμε πρόσβαση στην υπηρεσία ιστού RESTful πηγαίνοντας στην διεύθυνση **localhost:8080/api**.

Ως αποτέλεσμα, με την κλήση του root endpoint της υπηρεσίας διάμεσου του Spring Data REST, 1)επιστρέφονται οι πόροι που είναι διαθέσιμοι και 2) τα δεδομένα JSON επιστρέφονται σε μορφή HAL (Hypertext Application Language).

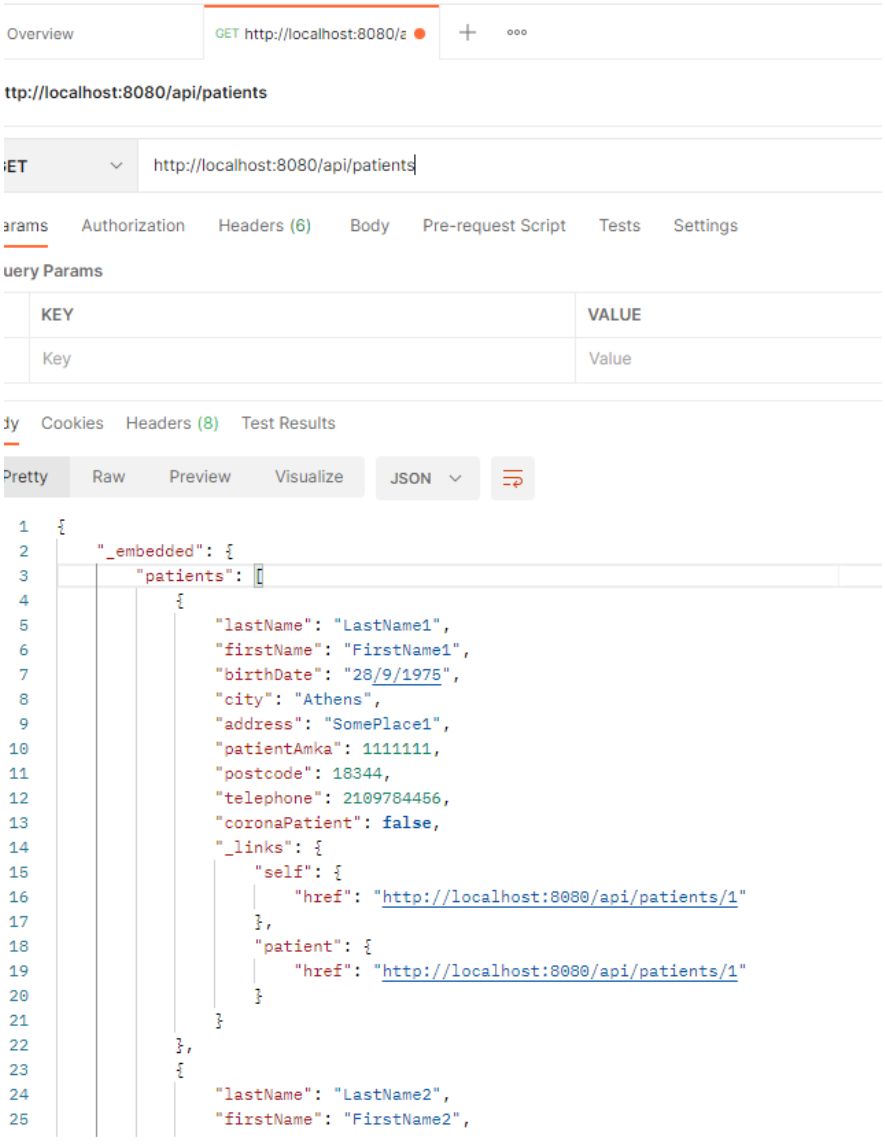


Εικόνα 24

Βλέπουμε ότι υπάρχουν σύνδεσμοι προς τις υπηρεσίες οντοτήτων ασθενών. Το όνομα της διαδρομής της υπηρεσίας Spring Data REST προκύπτει από το όνομα της οντότητας. Το όνομα θα είναι στη συνέχεια στον πληθυντικό αριθμό και χωρίς κεφαλαίο γράμμα. Για παράδειγμα, το όνομα της διαδρομής υπηρεσίας της οντότητας Patient θα ονομάζεται patients.

Υπάρχουν πολλά βοηθητικά εργαλεία διαθέσιμα για τον έλεγχο και την κατανάλωση RESTful υπηρεσιών ιστού. Θα κάνουμε χρήση του προγράμματος Postman. Το Postman μπορεί να αποκτηθεί ως εφαρμογή επιφάνειας εργασίας ή ως πρόσθετο πρόγραμμα περιήγησης. (“Postman API Platform | Sign Up for Free,” n.d.)

Οπότε, τώρα, κάνοντας μια αίτηση στο τελικό σημείο patients, http://localhost:8080/api/patients και χρησιμοποιώντας τη μέθοδο GET, θα λάβουμε μια λίστα με όλους τους ασθενείς, όπως φαίνεται στο ακόλουθο στιγμιότυπο οθόνης:



Εικόνα 25

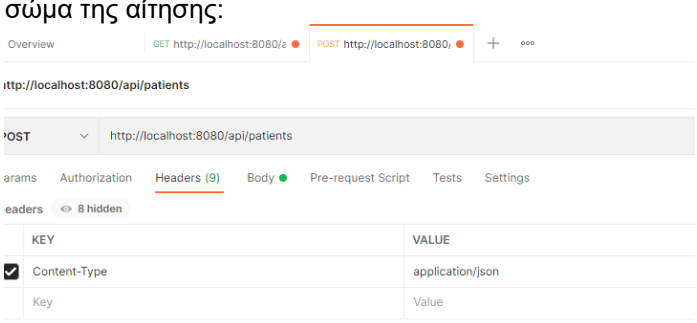
Όλοι οι ασθενείς έχουν επίσης το χαρακτηριστικό "_links", το οποίο είναι μια συλλογή από συνδέσμους, και με αυτούς μπορούμε να αποκτήσουμε πρόσβαση στον ίδιο τον ασθενή. Η διαδρομή είναι http://localhost:8080/api/patients/{id}.

Η υπηρεσία REST της Spring Data παρέχει όλες τις λειτουργίες CRUD. Ο παρακάτω πίνακας δείχνει ποιες μεθόδους HTTP μπορούμε να χρησιμοποιήσουμε για διάφορες λειτουργίες CRUD:

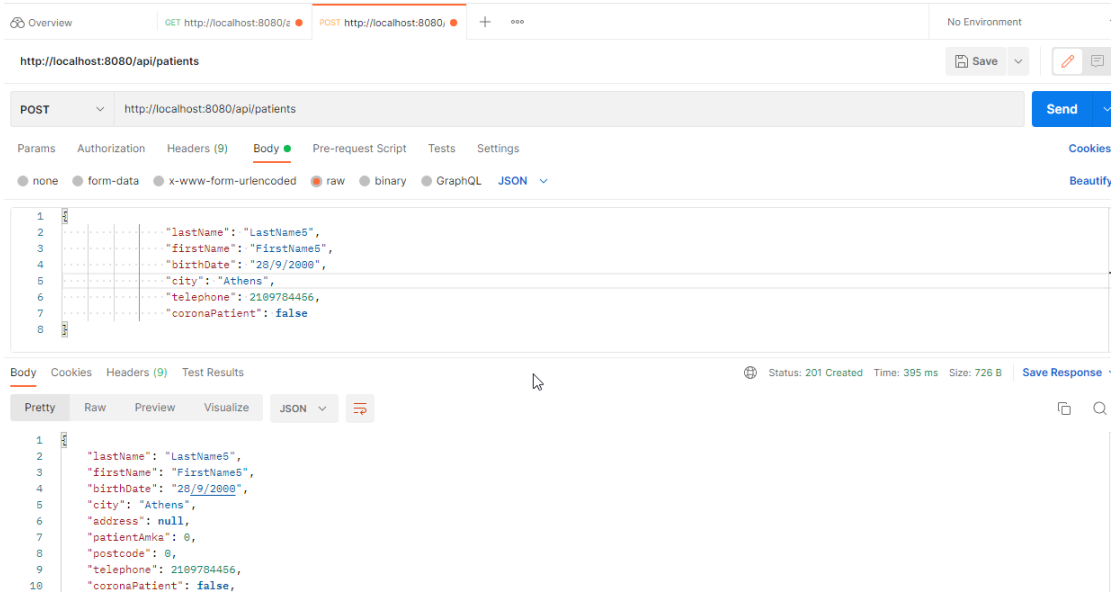
Μέθοδος HTTP	**CRUD**
GET	Ανάγνωση
POST	Δημιουργία
put/patch	Ενημέρωση
DELETE	Διαγραφή

Όταν θέλουμε να προσθέσουμε ένα νέο ασθενή στη βάση δεδομένων, πρέπει να χρησιμοποιήσουμε τη μέθοδο **POST** και ο σύνδεσμος είναι http://localhost:8080/api/patients

Η κεφαλίδα/header στο POSTMAN πρέπει να περιέχει το πεδίο Content-Type με την τιμή Content-Type:application/json, ενώ το αντικείμενο του νέου ασθενή θα ενσωματωθεί στο σώμα της αίτησης:



Εικόνα 26



Εικόνα 27

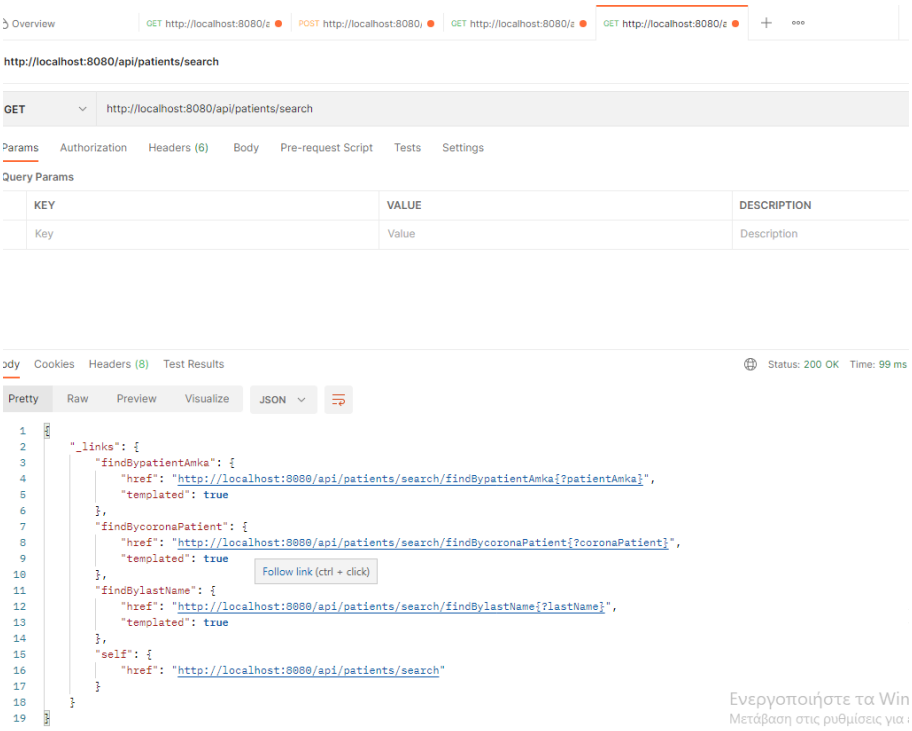
Η απάντηση θα στείλει πίσω ένα αντικείμενο νεοδημιουργημένου ασθενή. Τώρα, αν κάνουμε και πάλι μια αίτηση GET στη διαδρομή http://localhost:8080/api/patients, μπορούμε να δούμε ότι ο νέος ασθενής υπάρχει στη βάση δεδομένων.

Για να ενημερώσουμε οντότητες, πρέπει να χρησιμοποιήσουμε τη μέθοδο PATCH και το σύνδεσμο προς τον ασθενή που θέλουμε να ενημερώσουμε. (http://localhost:8080/api/patients/{id}).

Για να συμπεριλάβουμε τα ερωτήματα, πρέπει να προσθέσουμε το σχόλιο @RepositoryRestResource στην repository class/κλάση του αποθετηρίου. Οι παράμετροι των ερωτημάτων σχολιάζονται με τον σχολιασμό @Param. Ο παρακάτω πηγαίος κώδικας δείχνει το PatientRepository μας με αυτές τις επισημάνσεις:

```
package com.test.coronadatabase.domain;
import java.util.List;
import org.springframework.data.repository.CrudRepository;
import org.springframework.data.repository.query.Param;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;

public interface PatientRepository extends CrudRepository<Patient, Long> {
    // Fetch Patients by patientAmka
    List<Patient> findBypatientAmka(@Param("patientAmka") String patientAmka);
    // Fetch Patients by lastName
    List<Patient> findBylastName(@Param("lastName") String lastName);
    // Fetch Patients by coronaPatient
    List<Patient> findBycoronaPatient(@Param("coronaPatient") String coronaPatient);
}
```

Τώρα, όταν κάνουμε μια αίτηση GET στη διαδρομή http://localhost:8080/api/patients, μπορούμε να δούμε ότι υπάρχει ένα νέο τελικό σημείο με όνομα /search. Η κλήση της διαδρομής http://localhost:8080/api/patients/search επιστρέφει την ακόλουθη απάντηση:



Εικόνα 28

Από την απόκριση, μπορούμε να δούμε ότι και τα δύο ερωτήματα είναι πλέον διαθέσιμα στην υπηρεσία μας. Από την παραπάνω εικόνα φαίνεται καθαρά πώς μπορούμε να αντλήσουμε ασθενείς ανά ΑΜΚΑ, ανά επίθετο/lastname, είτε πρόκειται για ασθενή με κορονοϊό είτε όχι.

3.9 Ασφαλίζοντας την Backend εφαρμογή

3.9.1 Spring και Ασφάλεια

Το Spring Security είναι ένα ισχυρό και εξαιρετικά παραμετροποιήσιμο framework ελέγχου ταυτότητας και πρόσβασης. Αποτελεί το de facto πρότυπο για την εξασφάλιση εφαρμογών που βασίζονται στην Spring. Το Spring Security είναι ένα framework που επικεντρώνεται στην παροχή τόσο ελέγχου ταυτότητας όσο και εξουσιοδότησης σε εφαρμογές Java. Όπως όλα τα έργα της Spring, η πραγματική δύναμη του Spring Security βρίσκεται στο πόσο εύκολα μπορεί να επεκταθεί για να ικανοποιήσει τις εκάστοτε απαιτήσεις.

Χαρακτηριστικά:

- Ολοκληρωμένη και επεκτάσιμη υποστήριξη τόσο για έλεγχο ταυτότητας όσο και για εξουσιοδότηση
- Προστασία από επιθέσεις όπως session fixation, clickjacking, cross site request forgery, κ.λ.π.
- Ενσωμάτωση API Servlet
- Προαιρετική ενσωμάτωση με Spring Web MVC

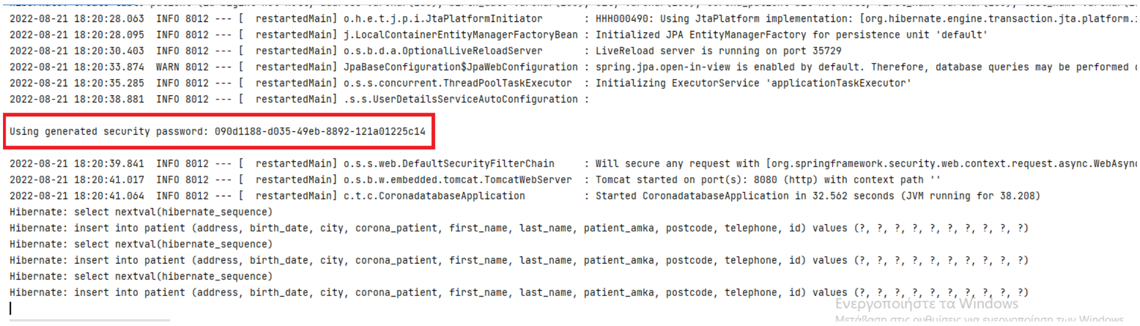
(“Spring Security,” n.d.)

3.9.2 Ρύθμιση της εφαρμογής ώστε να υποστηρίζει την Spring Security

Μπορούμε να συμπεριλάβουμε το Spring Security στην εφαρμογή μας προσθέτοντας την ακόλουθη εξάρτηση στο αρχείο pom.xml του Maven:

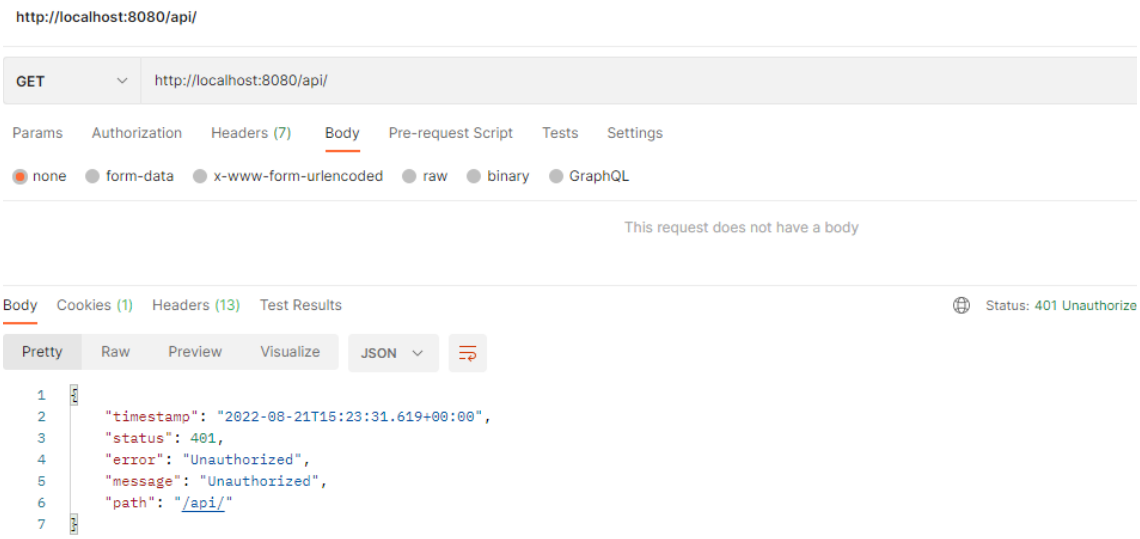
```
<dependency>
	<groupId>org.springframework.boot</groupId>
	<artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

Όταν ξεκινάμε την εφαρμογή , βλέπουμε από την κονσόλα ότι η Spring Security έχει δημιουργήσει έναν χρήστη στη μνήμη με το όνομα χρήστη user. Ο κωδικός πρόσβασης του χρήστη μπορεί να φανεί στην έξοδο της κονσόλας:



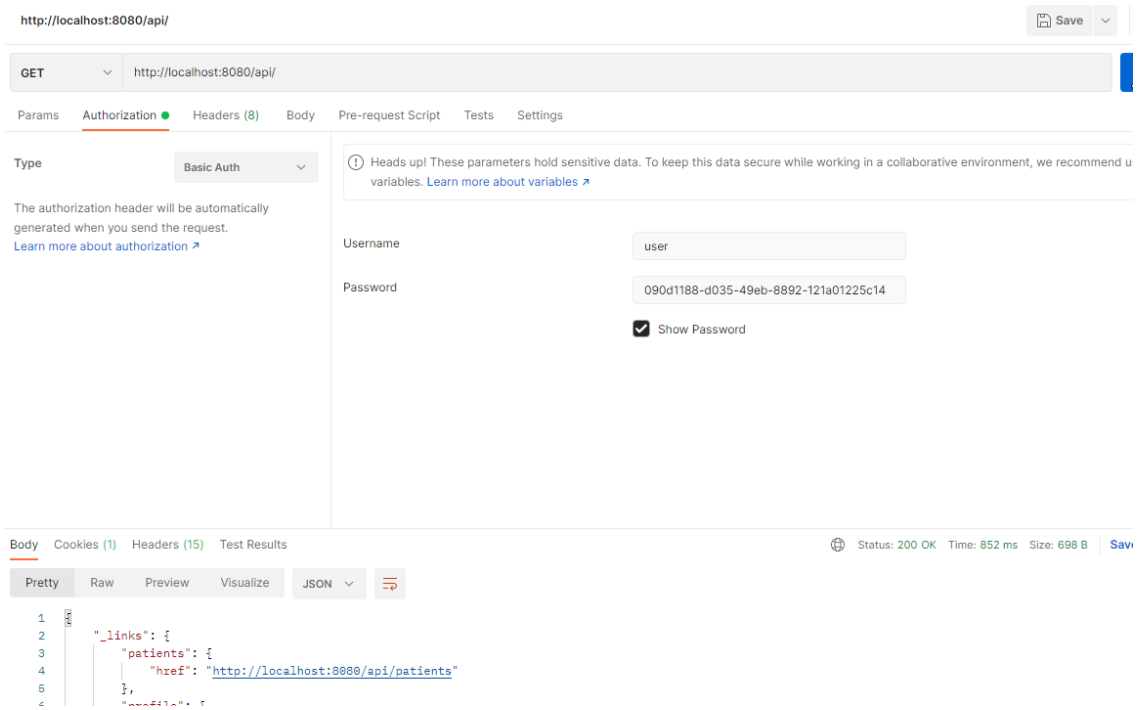
Εικόνα 29

Αν κάνουμε μια αίτηση GET στο τελικό σημείο του API , θα δούμε ότι είναι πλέον ασφαλές και θα λάβουμε ένα σφάλμα 401 Unauthorized:



Εικόνα 30

Για να μπορέσουμε να κάνουμε μια επιτυχημένη αίτηση GET, πρέπει να χρησιμοποιήσουμε basic auth/ βασικό έλεγχο ταυτότητας. Το παρακάτω στιγμιότυπο οθόνης δείχνει πώς να το κάνουμε αυτό με το Postman. Με τον έλεγχο ταυτότητας, μπορούμε να δούμε ότι η κατάσταση είναι 200 OK και ότι η απάντηση έχει σταλεί:



Εικόνα 31

Για να ρυθμίσουμε τον τρόπο συμπεριφοράς του Spring Security, πρέπει να προσθέσουμε μια νέα κλάση η οποία θα επεκτείνει το WebSecurityConfigurerAdapter. (“Introduction to Java Config for Spring Security | Baeldung,” n.d.)

Δημιουργούμε μια νέα κλάση με όνομα SecurityConfig στο πακέτο root της εφαρμογής . Ο παρακάτω πηγαίος κώδικας δείχνει τη δομή της class SecurityConfig. Οι επισημειώσεις @Configuration και @EnableWebSecurity απενεργοποιούν την προεπιλεγμένη διαμόρφωση ασφάλειας ιστού και μπορούμε να ορίσουμε τη δική μας διαμόρφωση σε αυτή την κλάση. Μέσα στη μέθοδο configure(HttpSecurity http), μπορούμε να ορίσουμε ποια τελικά σημεία της εφαρμογής μας είναι ασφαλή και ποια όχι. Ουσιαστικά, δεν χρειαζόμαστε ακόμη αυτή τη μέθοδο, επειδή μπορούμε να χρησιμοποιήσουμε τις προεπιλεγμένες ρυθμίσεις όπου όλα τα τελικά σημεία /endpoints είναι ασφαλή:

```
package com.test.coronadatabase;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapt
er;
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
```

```
    }
}
```

Για να αποθηκεύσουμε χρήστες στη βάση δεδομένων, πρέπει να δημιουργήσουμε μια user entity class/κλάση οντότητας χρήστη και ένα repository/αποθετήριο, όπως κάναμε προηγουμένως με την Patient class και την PatientRepository. Οι κωδικοί πρόσβασης δεν πρέπει να αποθηκεύονται στη βάση δεδομένων σε μορφή απλού κειμένου. Το Spring Security παρέχει πολλαπλούς αλγόριθμους ασφαλείας, όπως ο Bcrypt*.

* Το BCrypt υλοποιεί κατακερματισμό κωδικών πρόσβασης τύπου Blowfish του OpenBSD χρησιμοποιώντας το σχήμα που περιγράφεται στο "A Future-Adaptable Password Scheme" των Niels Provos και David Mazieres.
Αυτό το σύστημα κατακερματισμού κωδικών πρόσβασης προσπαθεί να αποτρέψει την off-line διάσπαση κωδικών πρόσβασης χρησιμοποιώντας έναν υπολογιστικά εντατικό αλγόριθμο κατακερματισμού, βασισμένο στον κρυπτογράφο Blowfish του Bruce Schneier. Ο συντελεστής εργασίας του αλγορίθμου είναι παραμετροποιημένος, ώστε να μπορεί να αυξηθεί καθώς οι υπολογιστές γίνονται ταχύτεροι.
(“BCrypt (spring-security-docs 5.7.3 API),” n.d.)

1. Δημιουργούμε μια νέα κλάση με όνομα User στο πακέτο domain.
2. Σημειώνουμε στην κλάση User το σχόλιο @Entity. Προσθέτουμε τα πεδία της κλάσης id, username, password και role. Τέλος, προσθέτουμε τους constructors, getters και setters και ορίζουμε όλα τα πεδία να είναι nullable και καθορίζουμε ότι το όνομα χρήστη πρέπει να είναι μοναδικό, χρησιμοποιώντας τον σχολιασμό @Column.

```
package com.test.coronadatabase.domain;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
@Entity
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(nullable = false, updatable = false)
    private Long id;
    @Column(nullable = false, unique = true)
    private String username;
    @Column(nullable = false)
    private String password;
    @Column(nullable = false)
    private String role;
    public User() {
    }
    public User(String username, String password, String role) {
        super();
        this.username = username;
        this.password = password;
        this.role = role;
    }
    public Long getId() {
        return id;
```

```
    }
    public String getUsername() {
        return username;
    }
    public String getPassword() {
        return password;
    }
    public String getRole() {
        return role;
    }
    public void setId(Long id) {
        this.id = id;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public void setPassword(String password) {
        this.password = password;
    }
    public void setRole(String role) {
        this.role = role;
    }
}
```

3. Δημιουργούμε ένα νέο Interface με όνομα UserRepository στο πακέτο domain

4. Ο πηγαίος κώδικας της UserRepository είναι παρόμοιος με αυτόν που φτιάξαμε στο patientRepository, αλλά υπάρχει μία μέθοδος ερωτήματος, η findByUsername, που χρειαζόμαστε για τα βήματα που ακολουθούν.

```
package com.test.coronadatabase.domain;
import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;
@Repository
public interface UserRepository extends CrudRepository<User, Long> {
    User findByUsername(String username);
}
```

5. Στη συνέχεια, θα δημιουργήσουμε μια κλάση που θα υλοποιεί τη διεπαφή UserDetailsService interface που παρέχεται από την Spring Security.
Η Spring Security το χρησιμοποιεί για την αυθεντικοποίηση και την εξουσιοδότηση των χρηστών.(“Spring Security: Database-backed UserDetailsService | Baeldung,” n.d.)

Δημιουργούμε ένα νέο πακέτο στο ριζικό πακέτο με το όνομα : service:com.test.coronadatabase.service

6. Δημιουργούμε μια νέα κλάση με όνομα *UserDetailServiceImpl* στο *πακέτο service* που μόλις δημιουργήσαμε.

7. Πρέπει να εισάγουμε την κλάση UserRepository στην κλάση UserDetailServiceImpl, επειδή αυτή χρειάζεται για να φέρουμε τον χρήστη από τη βάση δεδομένων όταν η Spring

Security χειρίζεται τον έλεγχο ταυτότητας. Η μέθοδος loadByUsername *επιστρέφει το αντικείμενο UserDetails*, το οποίο απαιτείται για τον έλεγχο ταυτότητας. Ακολουθεί ο πηγαίος κώδικας της UserDetailServiceImpl.java:

```
package com.test.coronadatabase.service;
import com.test.coronadatabase.domain.User;
import com.test.coronadatabase.domain.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.authority.AuthorityUtils;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;
@Service
public class UserDetailServiceImpl implements UserDetailsService {
   @Autowired
   private UserRepository repository;
   @Override
   public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException
   {
      User currentUser = repository.findByUsername(username);
      UserDetails user = new org.springframework.security.core
            .userdetails.User(username, currentUser.getPassword()
            , true, true, true, true,
            AuthorityUtils.createAuthorityList(currentUser.getRole()));
      return user;
   }
}
```

8. Στην SecurityConfig /κλάση διαμόρφωσης της ασφάλειας, πρέπει να ορίσουμε ότι η Spring Security θα πρέπει να χρησιμοποιεί χρήστες από τη βάση.
Προσθέτουμε μια νέα μέθοδο configureGlobal για να ενεργοποιήσουμε τους χρήστες από τη βάση δεδομένων. Δεν πρέπει ποτέ να αποθηκεύουμε τον κωδικό πρόσβασης ως απλό κείμενο στη βάση δεδομένων. Επομένως, θα ορίσουμε έναν αλγόριθμο κατακερματισμού κωδικού πρόσβασης στη μέθοδο configureGlobal. Σε αυτό το παράδειγμα, χρησιμοποιούμε τον αλγόριθμο BCrypt. Αυτό μπορεί εύκολα να υλοποιηθεί με την Spring Security BCryptPasswordEncoder κλάση. Ακολουθεί ο πηγαίος κώδικας της SecurityConfig.java.
Τώρα, ο κωδικός πρόσβασης πρέπει να κατακερματιστεί χρησιμοποιώντας το BCrypt πριν αποθηκευτεί στη βάση δεδομένων:

```
package com.test.coronadatabase;
import com.test.coronadatabase.service.UserDetailServiceImpl;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerB
uilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapt
```

```
er;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    @Autowired
    private UserDetailServiceImpl userDetailsService;
    @Override
    protected void configure(HttpSecurity http) throws Exception {
    }
    @Autowired
    public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(userDetailsService)
                .passwordEncoder(new BCryptPasswordEncoder());
    }
}
```

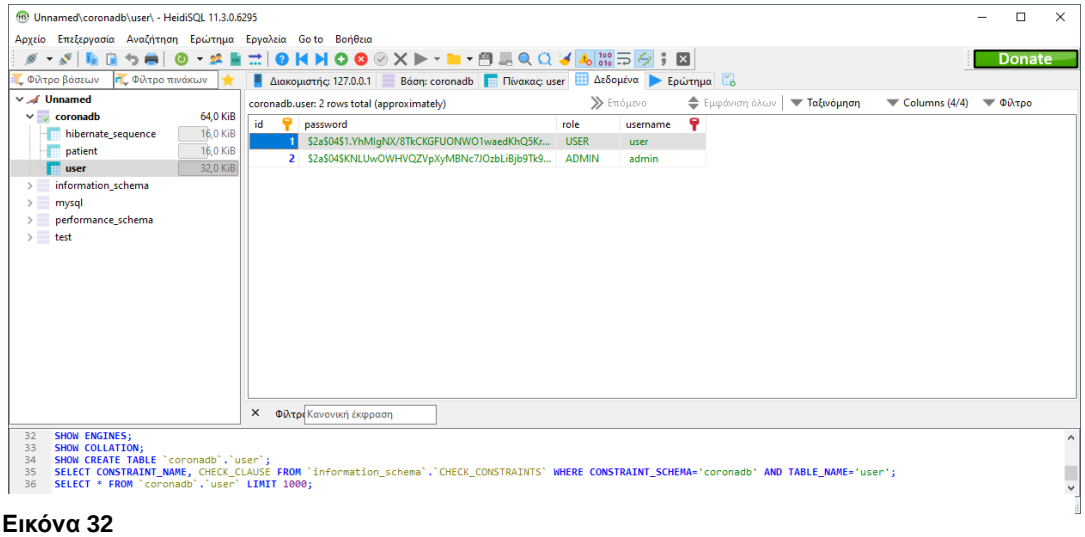
9. Τέλος, αποθηκεύουμε μερικούς δοκιμαστικούς χρήστες στη βάση δεδομένων στο CommandLineRunner μας. Ανοίγουμε το αρχείο CoronadatabaseApplication.java και προσθέτουμε τον ακόλουθο κώδικα στην αρχή της κλάσης για να εγχύσετε το UserRepository στην κύρια κλάση:

```
@Autowired
private UserRepository urepository;
```

10.Αποθηκεύουμε τους χρήστες στη βάση δεδομένων με κατακερματισμένους κωδικούς πρόσβασης. Μπορούμε να χρησιμοποιήσουμε οποιοδήποτε υπολογιστικό πρόγραμμα BCrypt στο διαδίκτυο για να το κάνουμε αυτό:

```
@Bean
CommandLineRunner runner(){
        return args -> {

                repository.save(new Patient("LastName1","FirstName1","28/9/1975",
                                "Athens","SomePlace1",1111111,18344,2109784456,false));
                repository.save(new Patient("LastName2","FirstName2","28/9/1980",
                                "Athens","SomePlace2",1111456,18380,2109784456,false));
                repository.save(new Patient("LastName3","FirstName3","28/9/2000",
                                "Athens","SomePlace3",1111111,18344,2109784456,false));


                urepository.save(new User("user",

        "$2a$04$1.YhMIgNX/8TkCKGFUONWO1waedKhQ5KrnB30fl0Q01QKqmzLf.Zi",
                                "USER"));

                urepository.save(new User("admin",

        "$2a$04$KNLUwOWHVQZVpXyMBNc7JOzbLiBjb9Tk9bP7KNcPI12ICuvzXQQKG",
                                "ADMIN"));
        };
}
```

Αφού εκτελέσουμε την εφαρμογή μας, θα δούμε ότι υπάρχει πλέον ένας πίνακας χρηστών στη βάση δεδομένων και ότι έχουν αποθηκευτεί δύο εγγραφές χρηστών:



Εικόνα 32

Αν προσπαθήσουμε να στείλουμε ένα αίτημα GET στο τελικό σημείο /api χωρίς έλεγχο ταυτότητας θα λάβουμε ένα σφάλμα 401 Unauthorized. Θα πρέπει να πραγματοποιήσουμε έλεγχο ταυτότητας για να μπορέσουμε να στείλουμε ένα επιτυχημένο αίτημα. Η διαφορά, σε σύγκριση με το προηγούμενο παράδειγμα, είναι ότι χρησιμοποιούμε τους χρήστες από τη βάση δεδομένων για τον έλεγχο ταυτότητας.

3.10 Προσθήκη δυνατότητας JWT στην Backend εφαρμογή

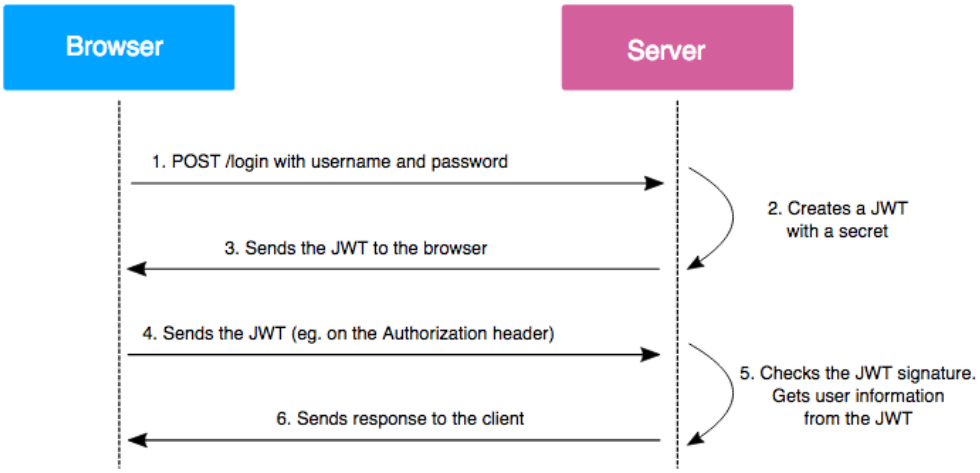
Στα προηγούμενα αναφέραμε τον τρόπο χρήσης του βασικού ελέγχου ταυτότητας με την υπηρεσία ιστού RESTful. Αυτή η μέθοδος δεν μπορεί να χρησιμοποιηθεί όταν αναπτύσσουμε το δικό μας frontend με το React, οπότε θα χρησιμοποιήσουμε αντί αυτού τον έλεγχο ταυτότητας *JSON Web Token (JWT)*.

3.10.1 Τι είναι ένα JWT Token

Το JSON Web Token (JWT) είναι ένα ανοικτό πρότυπο (RFC 7519) που ορίζει έναν συμπαγή και αυτοτελή τρόπο για την ασφαλή μετάδοση πληροφοριών μεταξύ των μερών ως αντικείμενο JSON. Οι πληροφορίες αυτές μπορούν να επαληθευτούν και να είναι αξιόπιστες επειδή είναι ψηφιακά υπογεγραμμένες. Τα JWT μπορούν να υπογραφούν χρησιμοποιώντας ένα μυστικό (με τον αλγόριθμο HMAC) ή ένα ζεύγος δημόσιου/ιδιωτικού κλειδιού χρησιμοποιώντας RSA ή ECDSA.

Αν και τα JWT μπορούν να κρυπτογραφηθούν για να παρέχουν επίσης μυστικότητα μεταξύ των μερών.(“JSON Web Token Introduction - jwt.io,” n.d.)

Το ακόλουθο διάγραμμα φανερώνει την κύρια ιδέα της διαδικασίας ελέγχου ταυτότητας JWT:



Εικόνα 33

Source: (“JWT tokens and security - working principles and use cases,” n.d.)

Μετά τον επιτυχή έλεγχο ταυτότητας, τα αιτήματα που αποστέλλονται από τον χρήστη θα πρέπει πάντα να περιέχουν το διακριτικό JWT που ελήφθη κατά τον έλεγχο ταυτότητας.

3.10.2.Προσθηκη βιβλιοθήκης Java Jwt στο pom.xml

Χρησιμοποιούμε τη βιβλιοθήκη ***Java JWT*** *, η οποία είναι η βιβλιοθήκη JWT για Java και Android.* (“GitHub - jwtk/jjwt: Java JWT: JSON Web Token for Java and Android,” n.d.)
Ως εκ τούτου, πρέπει να προσθέσουμε την ακόλουθη εξάρτηση στο αρχείο pom.xml. Η βιβλιοθήκη JWT χρησιμοποιείται για τη δημιουργία και την ανάλυση των JWT:

```
<dependency>
        <groupId>io.jsonwebtoken</groupId>
        <artifactId>jjwt</artifactId>
        <version>0.9.1</version>
</dependency>
```

(“Spring Boot Security + JWT Hello World Example | JavaInUse,” n.d.)

3.10.3 Τα παρακάτω βήματα δείχνουν πώς να ενεργοποιήσουμε τον έλεγχο ταυτότητας JWT στο Backend.

1.Δημιουργούμε μια νέα κλάση με όνομα *AuthenticationService* στο πακέτο service. Στην αρχή της κλάσης, θα ορίσουμε μερικές σταθερές-το EXPIRATIONTIME που ορίζει το χρόνο λήξης του token σε χιλιοστά του δευτερολέπτου, ενώ το SIGNINGKEY είναι ένα κλειδί υπογραφής συγκεκριμένου αλγορίθμου που χρησιμοποιείται για την ψηφιακή υπογραφή του JWT. Θα πρέπει να χρησιμοποιήσουμε μια κωδικοποιημένη με base64 συμβολοσειρά. Το PREFIX ορίζει το πρόθεμα του token και συνήθως χρησιμοποιείται το σχήμα Bearer. Η μέθοδος addToken δημιουργεί το token και το προσθέτει στην επικεφαλίδα Authorization της αίτησης. Το κλειδί υπογραφής κωδικοποιείται χρησιμοποιώντας τον αλγόριθμο SHA-512. Η μέθοδος προσθέτει επίσης Access-Control-Expose- Headers στην επικεφαλίδα με την τιμή Authorization. Αυτό είναι απαραίτητο επειδή δεν είμαστε σε θέση να έχουμε πρόσβαση στην επικεφαλίδα Authorization μέσω ενός frontend JavaScript από προεπιλογή.(“Multitenancy With Spring Data JPA | Baeldung,” n.d.)

Η μέθοδος getAuthentication παίρνει το token από την επικεφαλίδα Authorization της απάντησης χρησιμοποιώντας την parser() που παρέχεται από τη βιβλιοθήκη jjwt.(“io.jsonwebtoken.Jwts.parser java code examples | Tabnine,” n.d.)

```
package com.test.coronadatabase.service;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.util.Date;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;

import static java.util.Collections.emptyList;

public class AuthenticationService {
    static final long EXPIRATIONTIME = 864_000_00; // 1 day in milliseconds
    static final String SIGNINGKEY = "SecretKey";
    static final String PREFIX = "Bearer";

    // Add token to Authorization header
    static public void addToken(HttpServletResponse res, String username) {
        String JwtToken = Jwts.builder().setSubject(username)
                .setExpiration(new Date(System.currentTimeMillis()
                        + EXPIRATIONTIME))
                .signWith(SignatureAlgorithm.HS512, SIGNINGKEY)
                .compact();
        res.addHeader("Authorization", PREFIX + " " + JwtToken);
        res.addHeader("Access-Control-Expose-Headers", "Authorization");
    }

    // Get token from Authorization header
    static public Authentication getAuthentication(HttpServletRequest request) {
        String token = request.getHeader("Authorization");
        if (token != null) {
            String user = Jwts.parser()
                    .setSigningKey(SIGNINGKEY)
                    .parseClaimsJws(token.replace(PREFIX, ""))
                    .getBody()
                    .getSubject();

            if (user != null)
                return new UsernamePasswordAuthenticationToken(user, null,
                        emptyList());
        }
        return null;
    }
}
```

2.Στη συνέχεια, θα προσθέσουμε μια νέα απλή κλάση POJO για να κρατάμε τα διαπιστευτήρια για τον έλεγχο ταυτότητας.
Φτιάχνουμε μια νέα κλάση με όνομα *AccountCredentials* στο πακέτο domain. Η κλάση έχει δύο πεδία, το όνομα χρήστη και τον κωδικό πρόσβασης. Αυτή η κλάση δεν έχει τον σχολιασμό @Entity επειδή δεν χρειάζεται να αποθηκεύσουμε τα διαπιστευτήρια στη βάση δεδομένων:

```
package com.test.coronadatabase.domain;
//Μια απλη POJO class για να κραταει τα credentials για το authentication.
public class AccountCredentials {
    private String username;
    private String password;
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
}
```

3.Θα χρησιμοποιήσουμε κλάσεις φίλτρων για την είσοδο και τον έλεγχο ταυτότητας. Δημιουργούμε μια νέα κλάση με το όνομα LoginFilter στο root package που χειρίζεται τις αιτήσεις POST στο τελικό σημείο /login. Η κλάση LoginFilter επεκτείνει τη διεπαφή AbstractAuthenticationProcessingFilter της Spring Security, η οποία απαιτεί να ορίσουμε την ιδιότητα authenticationManager. Η αυθεντικοποίηση πραγματοποιείται από τη μέθοδο attemptAuthentication. Εάν ο έλεγχος ταυτότητας είναι επιτυχής, εκτελείται η μέθοδος succesfulAuthentication. Αυτή η μέθοδος θα καλέσει στη συνέχεια τη μέθοδο addToken της κλάσης υπηρεσίας μας και το token θα προστεθεί στην επικεφαλίδα Authorization.(“java - Redirect in result of AbstractAuthenticationProcessingFilter - Stack Overflow,” n.d.)

```
package com.test.coronadatabase;

import java.io.IOException;
import java.util.Collections;
import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.test.coronadatabase.domain.AccountCredentials;
import com.test.coronadatabase.service.AuthenticationService;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.AuthenticationException;
import org.springframework.security.web.authentication.AbstractAuthenticationProcessingFilter;
import org.springframework.security.web.util.matcher.AntPathRequestMatcher;
import com.fasterxml.jackson.databind.ObjectMapper;


public class LoginFilter extends AbstractAuthenticationProcessingFilter {
    public LoginFilter(String url, AuthenticationManager authManager) {
        super(new AntPathRequestMatcher(url));
        setAuthenticationManager(authManager);
    }
```

```
    @Override
    public Authentication attemptAuthentication(
            HttpServletRequest req, HttpServletResponse res)
            throws AuthenticationException, IOException, ServletException {
        AccountCredentials creds = new ObjectMapper()
                .readValue(req.getInputStream(), AccountCredentials.class);
        return getAuthenticationManager().authenticate(
                new UsernamePasswordAuthenticationToken(
                        creds.getUsername(),
                        creds.getPassword(),
                        Collections.emptyList()
                )
        );
    }
    @Override
    protected void successfulAuthentication(
            HttpServletRequest req,
            HttpServletResponse res, FilterChain chain,
            Authentication auth) throws IOException, ServletException {
        AuthenticationService.addToken(res, auth.getName());
    }

}
```

4.Δημιουργήστε μια νέα κλάση με όνομα AuthenticationFilter στο πακέτο root. Η κλάση επεκτείνει την GenericFilterBean, η οποία είναι μια γενική υπερκλάση για κάθε τύπο φίλτρου. Αυτή η κλάση θα χειρίζεται τον έλεγχο ταυτότητας σε όλα τα άλλα τελικά σημεία, εκτός από το /login. Το AuthenticationFilter χρησιμοποιεί τη μέθοδο getAuthentication από την κλάση της υπηρεσίας μας για να πάρει ένα token από την κεφαλίδα Authorization της αίτησης.
(“Spring Security custom authentication filter using Java Config - Stack Overflow,” n.d.)

```
package com.test.coronadatabase;
//Κλάση η οποία διαχειρίζεστε τα authentication Post  αιτήματα για τα endpoints εκτός του /login.
// Class that handles Post authentication requests for endpoints other than / login.
import com.test.coronadatabase.service.AuthenticationService;
import org.springframework.web.filter.GenericFilterBean;
import java.io.IOException;
import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletRequest;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;

public class AuthenticationFilter extends GenericFilterBean {
    @Override
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain
filterChain) throws IOException, ServletException {
        Authentication authentication =
AuthenticationService.getAuthentication((HttpServletRequest)request);
        SecurityContextHolder.getContext().
                setAuthentication(authentication);
        filterChain.doFilter(request, response);
    }
```

```
}
```

5. Εν συνεχεία θα πρέπει να κάνουμε αλλαγές στη μέθοδο configure της κλάσης μας SecurityConfig. Εκεί, ορίζουμε ότι το αίτημα με τη μέθοδο POST προς το τελικό σημείο /login επιτρέπεται χωρίς έλεγχο ταυτότητας και ότι τα αιτήματα προς όλα τα άλλα τελικά σημεία απαιτούν έλεγχο ταυτότητας. Ορίζουμε επίσης τα φίλτρα που θα χρησιμοποιηθούν στο /login και στα άλλα τελικά σημεία χρησιμοποιώντας τη μέθοδο addFilterBefore:

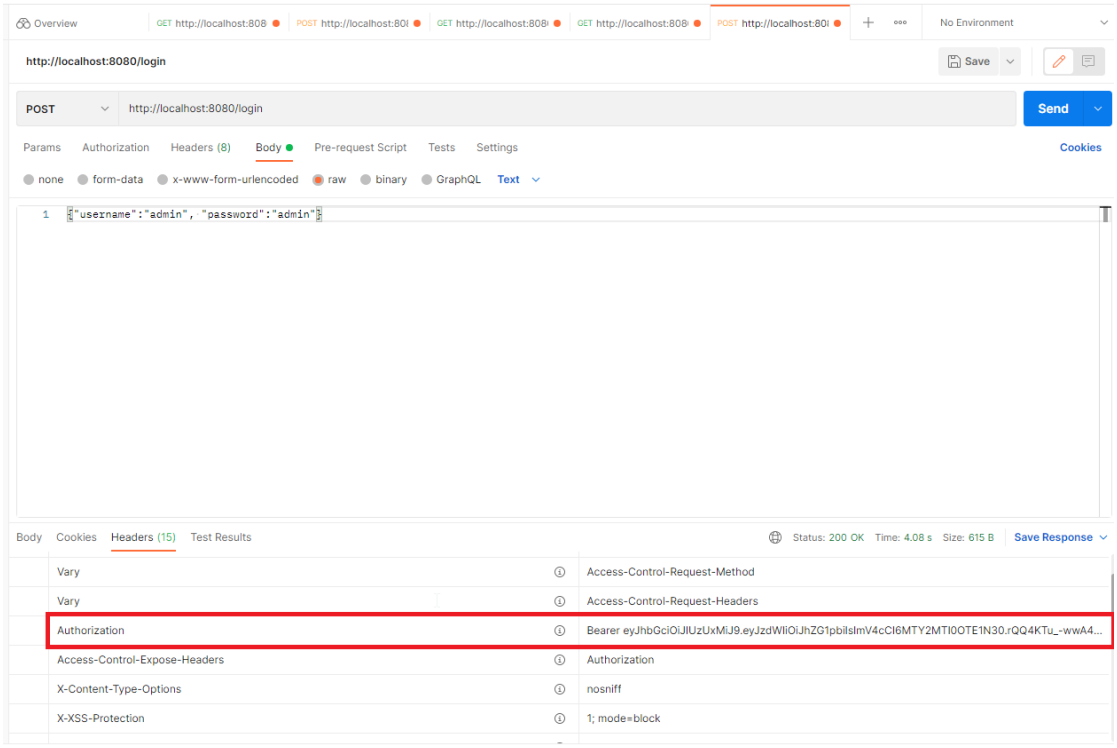
```
//SecurityConfig.java
@Override //Μέθοδος από την οποία ορίζουμε ποια endpoints είναι ασφαλή και ποια όχι.
protected void configure(HttpSecurity http) throws Exception {
   //http.csrf().disable().cors().and().authorizeRequests().anyRequest().permitAll();

   http.csrf().disable().cors().and().authorizeRequests()
         .antMatchers(HttpMethod.POST, "/login").permitAll()
         .anyRequest().authenticated() //Ορίζουμε ότι η POST μέθοδο που αιτείτε το /Login
endpoint επιτρέπετε χωρίς authentication &
                              // όλα τα αλλά αιτήματα για endpoint πέραν του Login θα απαιτούν
authentication.
         .and()
         // Filter for the api/login requests
         .addFilterBefore(new LoginFilter("/login", authenticationManager()),
               UsernamePasswordAuthenticationFilter.class)
         // Filter for other requests to check JWT in header
         .addFilterBefore(new AuthenticationFilter(),
               UsernamePasswordAuthenticationFilter.class);
}
```

6. Θα προσθέσουμε, επίσης, ένα φίλτρο CORS (συντομογραφία για Cross-Origin Resource Sharing) στην SecurityConfig/ κλάση διαμόρφωσης ασφαλείας μας. Αυτό είναι απαραίτητο για το frontend, το οποίο στέλνει αιτήσεις από την άλλη προέλευση. Το φίλτρο CORS υποκλέπτει τα αιτήματα και αν αυτά αναγνωρίζονται ως cross-origin, προσθέτει τις κατάλληλες επικεφαλίδες στο αίτημα. Για το σκοπό αυτό, θα χρησιμοποιήσουμε τη διεπαφή CorsConfigurationSource της Spring Security. Σε αυτό το παράδειγμα, θα επιτρέψουμε όλες τις μεθόδους HTTP και τις επικεφαλίδες (χρησιμοποιώντας "*").(“20. CORS,” n.d.)

```
// SecurityConfig.java
@Bean
CorsConfigurationSource corsConfigurationSource() {
   UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
   CorsConfiguration config = new CorsConfiguration();
   //config.setAllowedOrigins(Arrays.asList("*"));
   config.setAllowedOrigins(Arrays.asList("http://localhost:3000"));
   config.setAllowedMethods(Arrays.asList("*"));
   config.setAllowedHeaders(Arrays.asList("*"));
   //config.setAllowCredentials(false);
   config.setAllowCredentials(true);
   config.applyPermitDefaultValues();
```

```
    source.registerCorsConfiguration("/**", config);
    return source;
}
```

Τώρα, όταν εκτελέσουμε την εφαρμογή, μπορούμε να καλέσουμε το τελικό σημείο /login (http://localhost:8080/login) με τη μέθοδο POST αφου στο body του βάλουμε το ακολουθο: {"username":"admin", "password":"admin"} , οπότε σε περίπτωση επιτυχούς σύνδεσης, θα λάβουμε ένα JWT token στην επικεφαλίδα Authorization:



Εικόνα 34

Μετά από μια επιτυχημένη είσοδο, μπορούμε να καλέσουμε τα άλλα τερματικά σημεία υπηρεσιών RESTful στέλνοντας το JWT token που λάβαμε από την είσοδο στην επικεφαλίδα Authorization.

Πηγή για τη δημιουργία του κώδικα της ασφάλειας: (“Hands-On Full Stack Development with Spring Boot 2 and React: Build modern ... - Juha Hinkula - Google Books,” n.d.)

4.Δημιουργία της Frontend εφαρμογής και ένωσή της με το Backend

4.1 Τεχνικές απαιτήσεις

Απαιτείται αφενός η εφαρμογή Spring Boot που δημιουργήσαμε στα προηγούμενα ,αφετέρου να είναι εγκατεστημένος ο Node JS LTS.

4.2 Προετοιμασία του Backend της Spring Boot

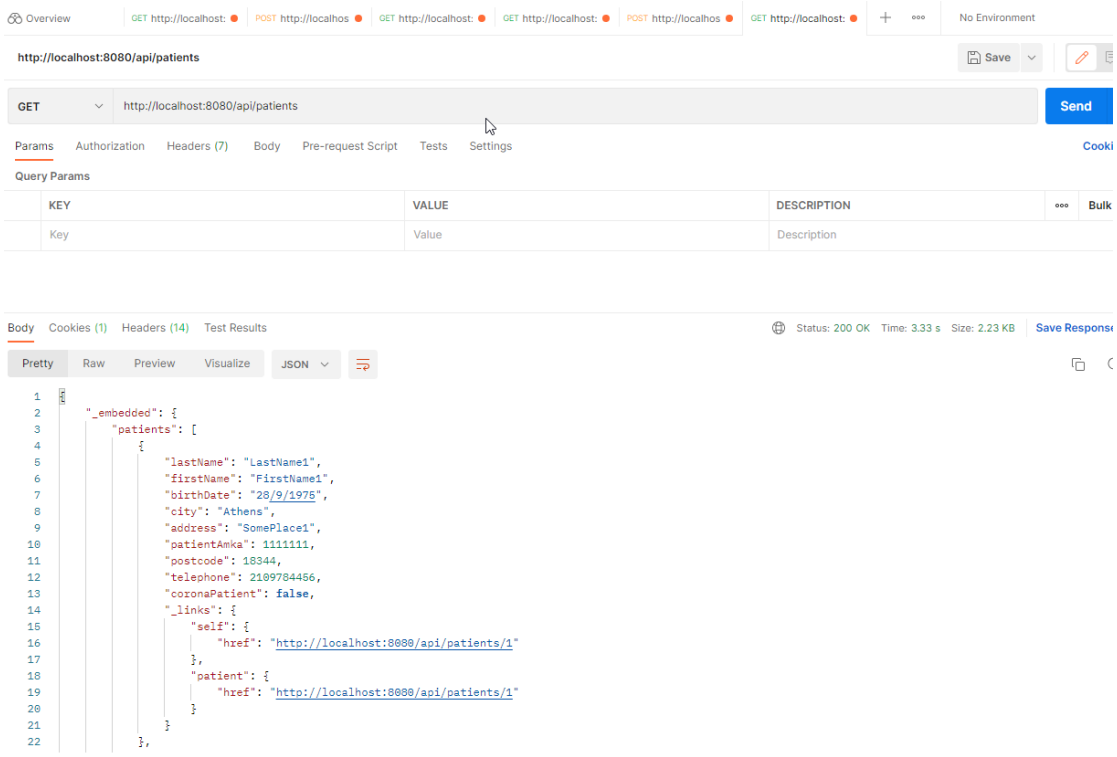
Θα ξεκινήσουμε την ανάπτυξη του frontend αφού α) πρώτα απασφαλίσουμε το backend μας ώστε να υλοποιήσουμε όλες τις λειτουργίες CRUD στο μπροστά μέρος της εφαρμογής/FrondEnd και β) στη συνεχεία αφού ενεργοποιήσουμε την ασφάλεια στο backend μας κάνοντας εκείνες τις τροποποιήσεις που απαιτούνται ώστε τελικά να υλοποιήσουμε τον έλεγχο ταυτότητας/authentication.

4.2.1 Απασφαλίζοντας το Backend

Ανοίγουμε το αρχείο ***SecurityConfig.java*** που ορίζει τη διαμόρφωση του Spring Security και σχολιάζουμε προσωρινά την τρέχουσα διαμόρφωση ώστε να δώσουμε σε όλους πρόσβαση σε όλα τα τελικά σημεία.

```
@Override
protected void configure(HttpSecurity http) throws Exception {
// Add this row to allow access to all endpoints
http.csrf().disable().cors().and().authorizeRequests().anyRequest().permitAll();

/* Σχολιάζουμε αυτό το κομμάτι κώδικα
http.csrf().disable().cors().and().authorizeRequests()
.antMatchers(HttpMethod.POST, "/login").permitAll()
.anyRequest().authenticated()
.and()
// Filter for the api/login requests
.addFilterBefore(new LoginFilter("/login", authenticationManager()),
UsernamePasswordAuthenticationFilter.class)
// Filter for other requests to check JWT in header
.addFilterBefore(new AuthenticationFilter(),
UsernamePasswordAuthenticationFilter.class);
*/
}
```

Τώρα, αν εκτελέσουμε το Backend και δοκιμάσουμε το τελικό σημείο / endpoint ***http:/localhost:8080/api/patients*** με το Postman, θα πρέπει να λάβουμε όλους τους ασθενείς στην απόκριση, όπως φαίνεται στο ακόλουθο στιγμιότυπο οθόνης:



Εικόνα 35

4.3 Δημιουργία του React project / έργου για το Frontend

1. Ανοίγουμε το PowerShell και δημιουργούμε μια νέα εφαρμογή React πληκτρολογώντας την ακόλουθη εντολή στον φάκελο του ΛΣ που επιθυμούμε:

```
npx create-react-app patientfront
```

2. Εγκαθιστούμε τη βιβλιοθήκη συστατικών Material-UI πληκτρολογώντας την ακόλουθη εντολή στο ριζικό φάκελο του έργου:

```
npm install @material-ui/core
```

3. Εκτελούμε την εφαρμογή πληκτρολογώντας την ακόλουθη εντολή στον ριζικό φάκελο του έργου:

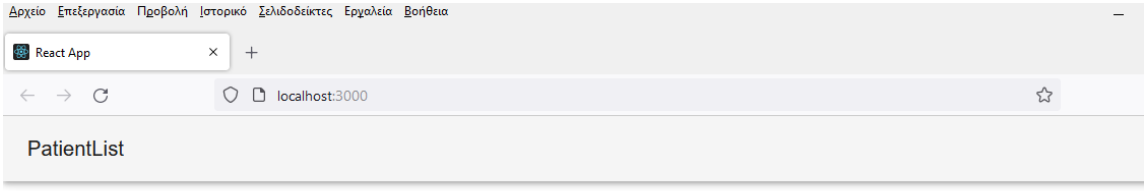
```
npm start
```

4. Ανοίγουμε το φάκελο **src** με το VS Code, αφαιρούμε τυχόν περιττό κώδικα και χρησιμοποιούμε το Material- UI Appbar στο αρχείο **App.js** για να αποκτήσουμε τη γραμμή εργαλείων για την εφαρμογή. Μετά τις τροποποιήσεις, ο πηγαίος κώδικας του αρχείου App.js θα πρέπει να εμφανίζεται ως εξής:

```
import React from 'react';
import './App.css';
import AppBar from '@material-ui/core/AppBar';
import Toolbar from '@material-ui/core/Toolbar';
import Typography from '@material-ui/core/Typography';
```

```
function App() {
return (
<div className="App">
<AppBar position="static" color="default">
<Toolbar>
<Typography variant="h6" color="inherit">
PatientList
</Typography>
</Toolbar>
</AppBar>
</div>
);
}

export default App;
```

Οπότε, τώρα το σημείο εκκίνησης του Frontend θα έχει την ακόλουθη μορφή:



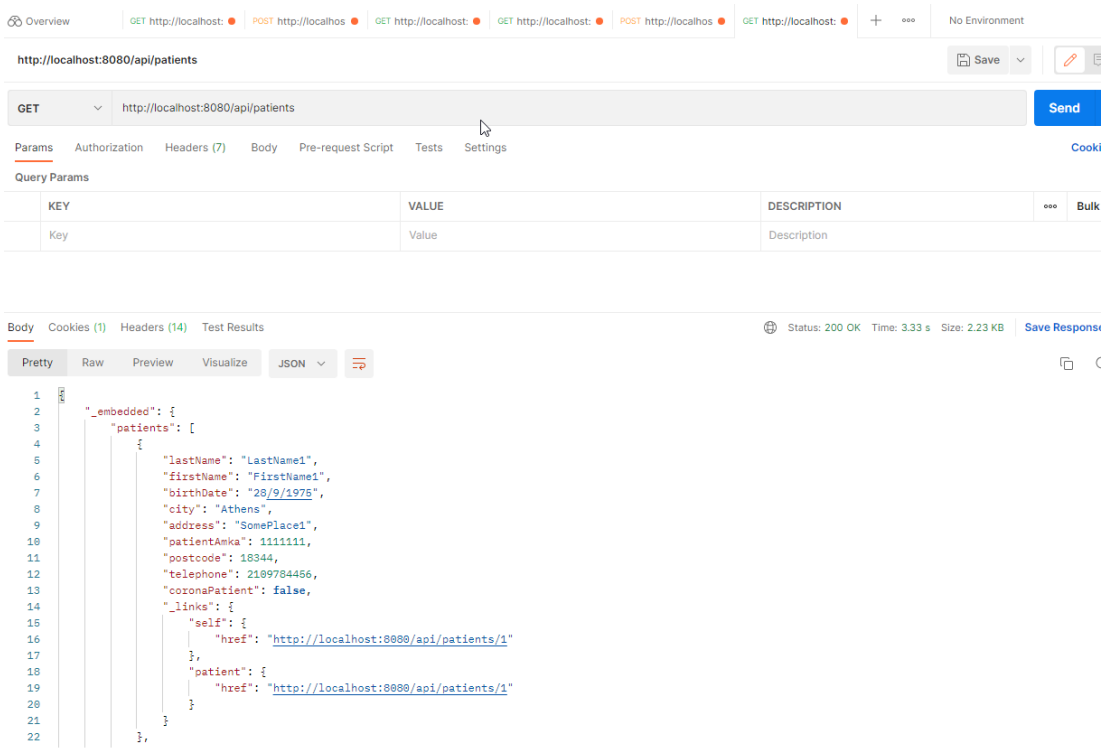
Εικόνα 36

4.4 Προσθήκη λειτουργιών CRUD στο FrontEnd

4.4.1 Δημιουργία της list page/σελίδας λίστας ασθενών

Σε πρώτη φάση, θα δημιουργήσουμε τη σελίδα λίστας για τους ασθενείς με δυνατότητες σελιδοποίησης, φιλτραρίσματος και ταξινόμησης. Εκτελούμε το Backend. Οι ασθενείς μπορούν να ανακτηθούν στέλνοντας το αίτημα GET στη διεύθυνση URL http://localhost:8080/api/patients

Τώρα, αν επιθεωρήσουμε τα δεδομένα JSON από την απόκριση θα δούμε ότι ο πίνακας των ασθενών μπορεί να βρεθεί στον κόμβο _embedded.patients των δεδομένων της απόκρισης JSON:



Εικόνα 37

Αφού ξέρουμε πώς να αντλούμε τους ασθενείς από το backend, είμαστε έτοιμοι να υλοποιήσουμε τη σελίδα λίστας για την εμφάνιση των ασθενών. Τα παρακάτω βήματα περιγράφουν το παραπάνω:

1.Ανοίγουμε την εφαρμογή της React patientfront με το VS Code .

2. Επειδή η εφαρμογή θα έχει πολλά components /συστατικά, δημιουργούμε έναν φάκελο για αυτά με το όνομα components στο φάκελο src.

3.Και δημιουργούμε ένα νέο αρχείο με όνομα Patientlist.js στον παραπάνω φάκελο(components).

4.Ανοίγουμε το αρχείο και γράφουμε τον βασικό κώδικα του component/συστατικού, ως εξής:

```
import React, { Component } from 'react';

class Patientlist extends Component {

render() {
return (
<div></div>
);
}
}
```

```
export default Patientlist;
```

5.Χρειαζόμαστε μια state / κατάσταση για τους ασθενείς που ανακτώνται από το REST API. Επομένως, πρέπει να προσθέσουμε τον κατασκευαστή και να ορίσουμε μια state / κατάσταση τύπου ενός πίνακα:

```
class Patientlist extends Component {
  constructor(props) {
    super(props);
    this.state = { patients: [] };
  }
```

6.Εκτελούμε την ανάκτηση στη μέθοδο κύκλου ζωής componentDidMount(). Οι ασθενείς από τα δεδομένα της απόκρισης JSON θα αποθηκευτούν στην κατάσταση, που ονομάζεται patients:

```
componentDidMount() {
  fetch('http://localhost:8080/api/patients')
 .then((response) => response.json())
      .then((responseData) => {
        this.setState({
          patients: responseData._embedded.patients,
        });
      })
      .catch((err) => console.error(err));
  };
```

7.Κανουμε χρήση της συνάρτηση map για να μετατρέψουμε τα αντικείμενα patient/ασθενών σε γραμμές πίνακα στη μέθοδο render() και προσθέτουμε το στοιχείο πίνακα:

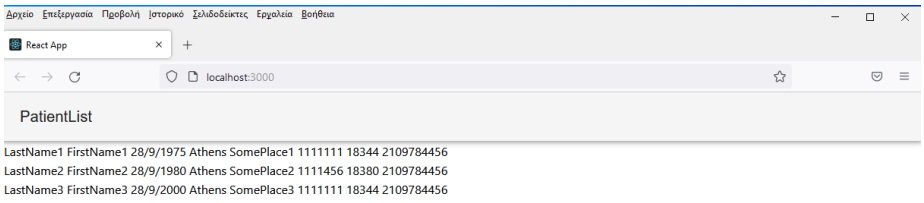
```
render() {
const tableRows = this.state.patients.map((patient, index) =>
<tr key={index}>
<td>{patient.lastName}</td>
<td>{patient.firstName}</td>
<td>{patient.birthDate}</td>
<td>{patient.city}</td>
<td>{patient.address}</td>
<td>{patient.patientAmka}</td>
<td>{patient.postcode}</td>
<td>{patient.telephone}</td>
</tr>
);

return (
<div className="App">
<table>
<tbody>{tableRows}</tbody>
</table>
```

```
</div>
);
}
```

8.Στη συνέχεια, θα πρέπει να εισάγουμε και να αποδώσουμε το στοιχείο Patientlist / component στο αρχείο App.js. Στο αρχείο App.js, προσθέτουμε τη δήλωση import κι έπειτα το στοιχείο Patientlist στη δήλωση return:

```
import React from 'react';
import './App.css';
import AppBar from '@material-ui/core/AppBar';
import Toolbar from '@material-ui/core/Toolbar';
import Typography from '@material-ui/core/Typography';
import Patientlist from './components/Patientlist';
function App() {
return (
<div className="App">
<AppBar position="static" color="default">
<Toolbar>
<Typography variant="h6" color="inherit">
Patientlist
</Typography>
</Toolbar>
</AppBar>
<Patientlist />
</div>
);
}

export default App;
```

Τώρα, αν ξεκινήσουμε την εφαρμογή React (αφού έχουμε προηγουμένως εκκινήσει την Spring Boot application) με την εντολή npm start, θα πρέπει να δούμε την ακόλουθη σελίδα:



Εικόνα 38

Η διεύθυνση URL του διακομιστή μπορεί να επαναλαμβάνεται πολλές φορές όταν δημιουργούμε περισσότερες λειτουργίες CRUD. Επομένως, είναι προτιμότερο να την ορίσουμε ως σταθερά. Στη συνέχεια, όταν αλλάζει η τιμή της διεύθυνσης URL, πρέπει να την τροποποιούμε σε ένα σημείο.

Ας δημιουργήσουμε ένα νέο αρχείο, το **constants.js**, στο ριζικό φάκελο της εφαρμογής μας:

1. Ανοίγουμε το αρχείο στον επεξεργαστή και προσθέτουμε την ακόλουθη γραμμή στο αρχείο:

```
export const SERVER_URL = 'http://localhost:8080/'
```

2.Στη συνέχεια, θα την εισάγουμε στο αρχείο μας Patientlist.js και θα τη χρησιμοποιήσουμε στη μέθοδο fetch:

```
//Patientlist.js
import {SERVER_URL} from '../constants.js'

// Κάνουμε χρήση της εισηγμένης σταθεράς μέσα στην  fetch method
fetch(SERVER_URL + 'api/patients')
```

1.Οπότε, μετά τα παραπάνω, ο πηγαίος κώδικας του αρχείου Patientlist.js θα πρέπει να μοιάζει με τον ακόλουθο:

```
import React, { Component } from 'react';
import {SERVER_URL} from '../constants.js'

class Patientlist extends Component {

    constructor(props) {
      super(props);
      this.state = { patients: [] };
    }

    componentDidMount() {
      fetch(SERVER_URL + 'api/patients')
      .then((response) => response.json())
          .then((responseData) => {
            this.setState({
              patients: responseData._embedded.patients,
            });
          })
          .catch((err) => console.error(err));
      };

     render() {
      const tableRows = this.state.patients.map((patient, index) =>
      <tr key={index}>
      <td>{patient.lastName}</td>
```

```
        <td>{patient.firstName}</td>
        <td>{patient.birthDate}</td>
        <td>{patient.city}</td>
        <td>{patient.address}</td>
        <td>{patient.patientAmka}</td>
        <td>{patient.postcode}</td>
        <td>{patient.telephone}</td>
        </tr>
        );

        return (
        <div className="App">
        <table>
        <tbody>{tableRows}</tbody>
        </table>
        </div>
        );
        }
}
export default Patientlist;
```

Τώρα, θα χρησιμοποιήσουμε το React Table για να αποκτήσουμε τις δυνατότητες σελιδοποίησης, φιλτραρίσματος και ταξινόμησης out of the box. Σταματάμε τον διακομιστή ανάπτυξης πατώντας Ctrl + C στο τερματικό και πληκτρολογούμε την ακόλουθη εντολή για να εγκαταστήσουμε το React Table.

```
npm install react-table-6
```

Στη συνέχεια, εισάγουμε το react-table και το φύλλο στυλ στο αρχείο Patientlist.js:

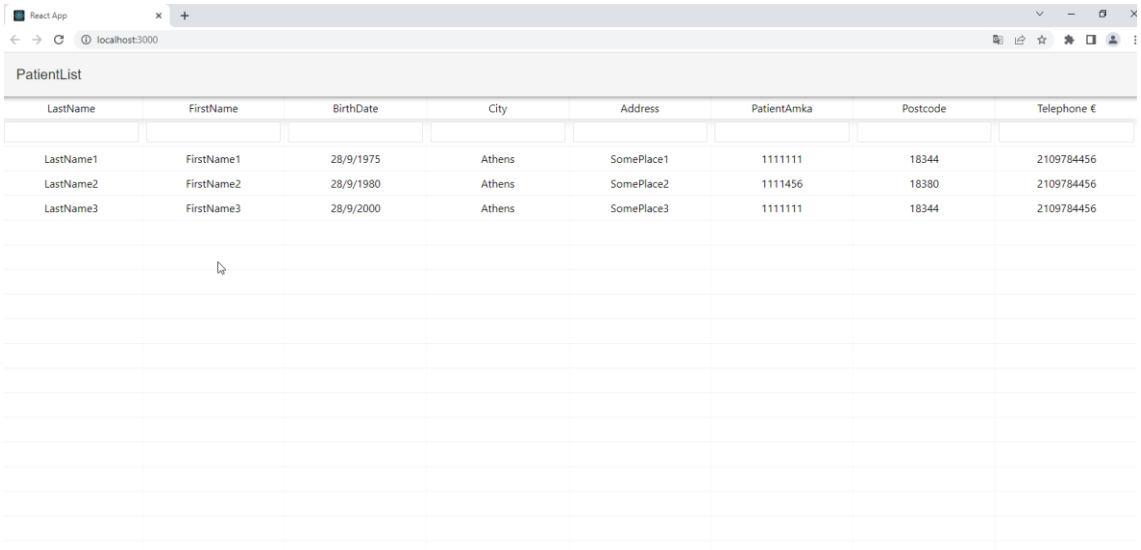
```
import ReactTable from "react-table-6";
import "react-table-6/react-table.css"
```

Έπειτα, αφαιρούμε τα table και tableRows από τη μέθοδο render(). Οπότε, τώρα το this.state.patients, περιέχει τους ανακτημένους ασθενείς. Πρέπει, επίσης, να ορίσουμε τις στήλες του πίνακα, όπου accessor είναι το πεδίο του αντικειμένου patient και header είναι το κείμενο της επικεφαλίδας. Για να ενεργοποιήσουμε το φιλτράρισμα, ορίζουμε το prop filterable του πίνακα σε true.

```
      render() {
        const columns = [{
        Header: 'LastName',
        accessor: 'lastName'
        }, {
        Header: 'FirstName',
        accessor: 'firstName',
        }, {
        Header: 'BirthDate',
        accessor: 'birthDate',
```

```
}, {
Header: 'City',
accessor: 'city'
}, {
Header: 'Address',
accessor: 'address',
}, {
Header: 'PatientAmka',
accessor: 'patientAmka',
}, {
Header: 'Postcode',
accessor: 'postcode',
}, {
Header: 'Telephone €',
accessor: 'telephone',
},]

return (
<div className="App">
<ReactTable data={this.state.patients} columns={columns}
filterable={true}/>
</div>
);
}
```

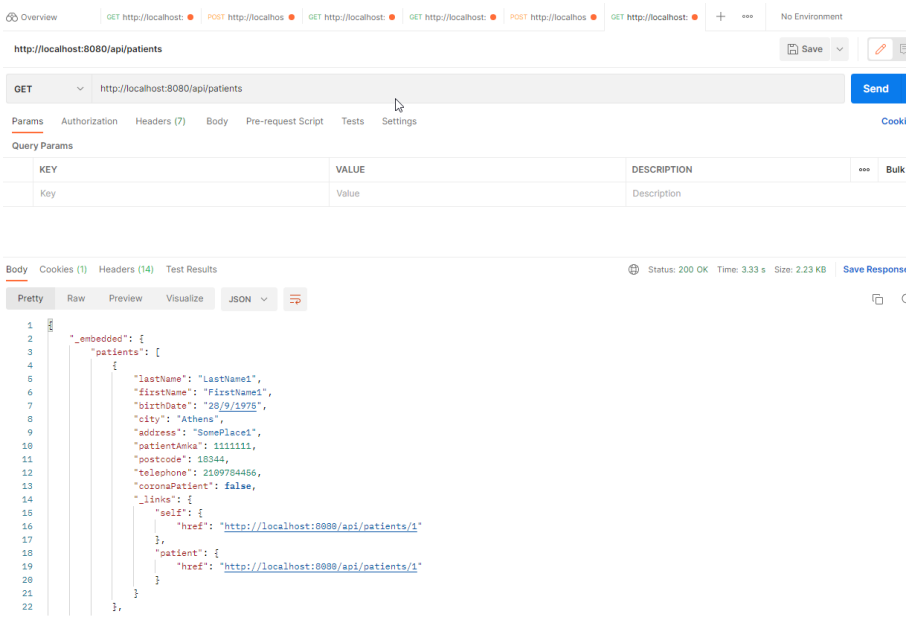
Με το συστατικό React Table, η σελίδα της λίστας μοιάζει με την ακόλουθη:



Εικόνα 39

4.4.2 Η λειτουργικότητα διαγραφής και τα βήματα για την δημιουργία της

Τα στοιχεία μπορούν να διαγραφούν από τη βάση δεδομένων στέλνοντας το αίτημα της μεθόδου DELETE στο τελικό σημείο http://localhost:8080/api/patients/[patientid]. Αν εξετάσουμε τα δεδομένα απόκρισης JSON, μπορούμε να δούμε ότι κάθε ασθενής περιέχει έναν σύνδεσμο προς τον εαυτό του και ότι μπορεί να γίνει πρόσβαση σε αυτόν από τον κόμβο _links.self.href, όπως φαίνεται στο ακόλουθο στιγμιότυπο:



Εικόνα 40

1. Εδώ, θα δημιουργήσουμε ένα κουμπί για κάθε γραμμή του πίνακα. Ο accessor του κουμπιού θα είναι ο κόμβος _links.self.href, τον οποίο μπορούμε να χρησιμοποιήσουμε για να καλέσουμε τη συνάρτηση delete που θα δημιουργήσουμε. Αλλά πρώτα, προσθέτουμε μια νέα στήλη στον πίνακα χρησιμοποιώντας την Cell για την απόδοση του κουμπιού, όπως βλέπουμε στον παρακάτω πηγαίο κώδικα. Επίσης, δεν ενεργοποιήσουμε την ταξινόμηση και το φιλτράρισμα για τη στήλη του κουμπιού, τα θέτουμε σε false. Το κουμπί καλεί τη συνάρτηση onDelClick όταν πατηθεί και στέλνει ως όρισμα έναν σύνδεσμο στον ασθενή:

```
const columns = [{
                Header: 'LastName',
                    accessor: 'lastName'
                    }, {
                    Header: 'FirstName',
                    accessor: 'firstName',
                    }, {
                    Header: 'BirthDate',
                    accessor: 'birthDate',
                    }, {
                    Header: 'City',
                    accessor: 'city'
                    }, {
                    Header: 'Address',
```

```
                    accessor: 'address',
                    }, {
                    Header: 'PatientAmka',
                    accessor: 'patientAmka',
                    }, {
                    Header: 'Postcode',
                    accessor: 'postcode',
                    }, {
                    Header: 'Telephone €',
                    accessor: 'telephone',
                    },
                {
                id: 'delbutton',
                sortable: false,
                filterable: false,
                width: 100,
                accessor: '_links.self.href',
                Cell: ({value}) => (<button onClick={()=>{this.onDelClick(value)}}>Delete</button>)
                },
];
```

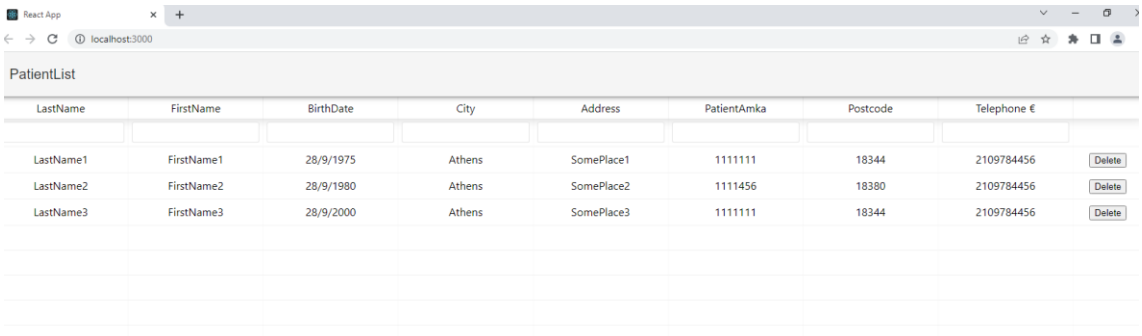
2.Υλοποιούμε τη συνάρτηση onDelClick. Όμως, πρώτα αυτό που κάνουμε είναι να βγάλουμε τη συνάρτηση fetchPatients από τη μέθοδο componentDidMount(). Αυτό είναι απαραίτητο, επειδή θέλουμε επίσης να καλέσουμε τη συνάρτηση fetchPatients μετά τη διαγραφή του ασθενή , προκειμένου να εμφανίσουμε μια ενημερωμένη λίστα με ασθενείς στο χρήστη. Δημιουργούμε μια νέα συνάρτηση με όνομα fetchPatients() και αντιγράφουμε τον κώδικα από τη μέθοδο componentDidMount() σε μια νέα συνάρτηση. Στη συνέχεια, θα καλέσουμε τη συνάρτηση fetchPatients() από τη συνάρτηση componentDidMount() για την ανάκτηση ασθενών:

```
componentDidMount() {
   this.fetchPatients();
 }

 fetchPatients = () => {
   console.log("FETCH");
   fetch(SERVER_URL + "api/patients")
     .then((response) => response.json())
     .then((responseData) => {
       this.setState({
         patients: responseData._embedded.patients,
       });
     })
     .catch((err) => console.error(err));
 };
```

3.Εν συνεχεία υλοποιούμε τη συνάρτηση onDelClick. Στέλνουμε το αίτημα DELETE σε έναν σύνδεσμο ασθενών και όταν το αίτημα DELETE επιτύχει, ανανεώνουμε τη σελίδα της λίστας καλώντας τη συνάρτηση fetchPatients():

```
// Delete Patient
onDelClick = (link) => {
fetch(link, {method: 'DELETE'})
.then(res => this.fetchPatients())
.catch(err => console.error(err))
}
```

Οπότε, τώρα όταν ξεκινάμε την εφαρμογή ,το frontend θα πρέπει να μοιάζει με το ακόλουθο στιγμιότυπο οθόνης και ο ασθενής θα εξαφανίζεται από τη λίστα όταν πατηθεί το κουμπί Delete:



Εικόνα 41

Επίσης, θα πρέπει να δώσουμε στο χρήστη κάποια ανατροφοδότηση σε περίπτωση επιτυχούς διαγραφής ή αν υπάρχουν σφάλματα. Οπότε υλοποιούμε ένα μήνυμα ενημέρωσης για να δείξουμε την κατάσταση της διαγραφής. Για το σκοπό αυτό χρησιμοποιούμε το συστατικό react-toastify (https://github.com/fkhadra/react-toastify) το οποίο και εγκαθιστούμε πληκτρολογώντας την ακόλουθη εντολή:

```
npm install react-toastify
```

Εν συνεχεία ανοίγουμε το αρχείο Patientlist.js στον Vs Code. Και κανουμε import / εισάγωγη τα:[1]ToastContainer, toast και [2]το css του, ώστε να μπορέσουμε να αρχίσουμε να χρησιμοποιούμε το react-toastify.

```
import { ToastContainer, toast } from "react-toastify";
import "react-toastify/dist/ReactToastify.css";
```

Το ToastContainer είναι ένα component / συστατικό για την εμφάνιση των μηνυμάτων το οποίο μας παρέχει feedback και θα πρέπει να βρίσκεται μέσα στη μέθοδο render(). Στο ToastContainer, μπορούμε να ορίσουμε τη διάρκεια του μηνύματος σε χιλιοστά του δευτερολέπτου χρησιμοποιώντας την ιδιότητα autoClose. Εν συνεχεία, προσθέτουμε το συστατικό ToastContainer μέσα στη δήλωση return της μεθόδου render(), αμέσως μετά το ReactTable:

```
return (
```

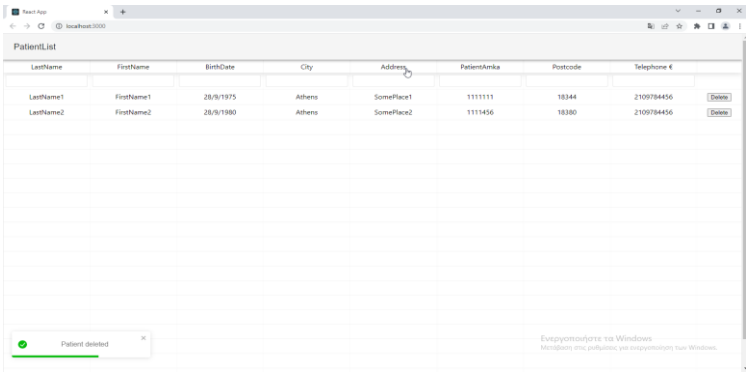
```
<div className="App">
<ReactTable data={this.state.patients} columns={columns}
filterable={true}/>
<ToastContainer autoClose={1500} />
</div>
);
```

Στη συνέχεια, καλούμε τη μέθοδο toast μέσα από την συνάρτηση onDelClick() για να εμφανίσουμε το μήνυμα ανατροφοδοτησης. Το μήνυμα επιτυχίας εμφανίζεται όταν η διαγραφή είναι επιτυχής, ενώ το μήνυμα σφάλματος εμφανίζεται σε περίπτωση σφάλματος:

```
// Delete patient
  onDelClick = (link) => {
    if (window.confirm("Are you sure to delete?")) {
      fetch(link, {
        method: "DELETE",
      })
        .then((res) => {
          toast.success("Patient deleted", {
            position: toast.POSITION.BOTTOM_LEFT,
          });
          this.fetchPatients();
        })
        .catch((err) => {
          toast.error("Error when deleting", {
            position: toast.POSITION.BOTTOM_LEFT,
          });
          console.error(err);
        });
    }
  };
```

Οπότε τώρα, θα εμφανίζεται το μήνυμα ανατροφοδότησης όταν διαγράφεται ασθενής, όπως φαίνεται στο ακόλουθο στιγμιότυπο οθόνης.

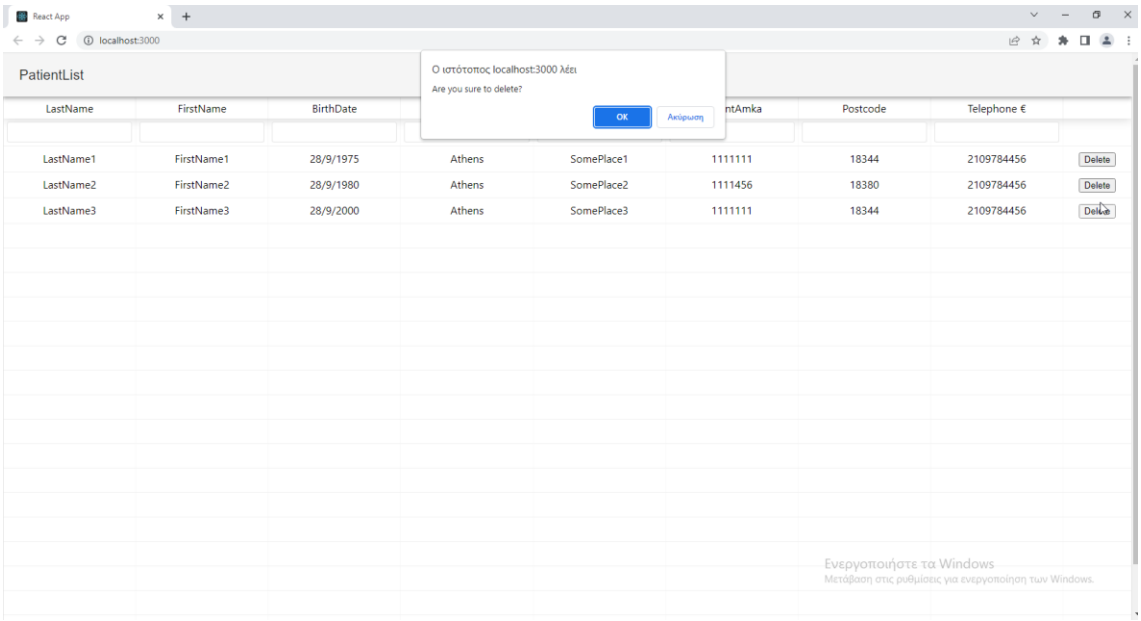
Για να καλύψουμε το ενδεχόμενο της τυχαίας διαγραφής του ασθενή, δημιουργούμε παράθυρο επιβεβαίωσης μετά το πάτημα του κουμπιού διαγραφής. Αυτό θα το υλοποιήσουμε χρησιμοποιώντας τη μέθοδο confirm του αντικειμένου window. Προσθέτουμε το confirm στη μέθοδο onDelClick:



Εικόνα 42

```
// Delete patient
  onDelClick = (link) => {
    if (window.confirm("Are you sure to delete?")) {
      fetch(link, {
        method: "DELETE",
      })
        .then((res) => {
          toast.success("Patient deleted", {
            position: toast.POSITION.BOTTOM_LEFT,
          });
          this.fetchPatients();
        })
        .catch((err) => {
          toast.error("Error when deleting", {
            position: toast.POSITION.BOTTOM_LEFT,
          });
          console.error(err);
        });
    }
};
```

Οπότε, τώρα, αφού πατηθεί το κουμπί Delete, θα ανοίξει το παράθυρο επιβεβαίωσης και ο ασθενής θα διαγραφεί μόνο αν πατήσουμε το κουμπί OK:



Εικόνα 43

4.4.3 Η λειτουργικότητα προσθήκης και τα βήματα για την λειτουργικότητα προσθήκης

Το επόμενο βήμα είναι η δημιουργία μιας λειτουργικότητας προσθήκης για το frontend. Αυτό το υλοποιούμε χρησιμοποιώντας τον διαλογικό διάλογο Material-UI* modal. Αυτό που κάνουμε είναι να προσθέσουμε ένα νέο κουμπί με όνομα New Patient (Νέος Ασθενής) στη διεπαφή χρήστη, το οποίο να ανοίγει τη διαδραστική φόρμα όταν πατηθεί. Η modal φόρμα περιέχει όλα τα πεδία που απαιτούνται για την αποθήκευση του ασθενή, καθώς και το κουμπί για την αποθήκευση και την ακύρωση.

*Η βιβλιοθήκη συστατικών Material-UI έχει ήδη προστεθεί στην εφαρμογή frontend από πριν.

1. Δημιουργούμε ένα νέο αρχείο με όνομα AddPatient.js στο φάκελο components και γράφουμε στο αρχείο κάποιο βασικό κώδικα κλάσης συνάρτησης, όπως φαίνεται παρακάτω και κάνουμε και εισαγωγή του συστατικού Material-UI Dialog:

```
import React from 'react';
import Dialog from '@material-ui/core/Dialog';
import DialogActions from '@material-ui/core/DialogActions';
import DialogContent from '@material-ui/core/DialogContent';
import DialogContentText from '@material-ui/core/DialogContentText';
import DialogTitle from '@material-ui/core/DialogTitle';

const AddPatient = (props) => {
return (
<div>
</div>
);
};

export default AddPatient;
```

2. Εισάγουμε μια state / κατάσταση τύπου αντικειμένου που περιέχει όλα τα πεδία του patient / ασθενή χρησιμοποιώντας το useState hook. Για τον διάλογο, χρειαζόμαστε επίσης μια κατάσταση τύπου Boolean για να ορίσουμε την ορατότητα της φόρμας διαλόγου:

```
import React, { useState } from "react";
import Dialog from "@material-ui/core/Dialog";
import DialogActions from "@material-ui/core/DialogActions";
import DialogContent from "@material-ui/core/DialogContent";
import DialogTitle from "@material-ui/core/DialogTitle";

const AddPatient = (props) => {
   const [open, setOpen] = useState(false);
   const [patient, setPatient] = useState({
     lastName: "",
     firstName: "",
     birthDate: "",
     city: "",
     address: "",
     patientAmka: "",
```

```
    postcode: "",
    telephone: "",
    coronaPatient: "",
  });

  return (
    <div>
    </div>
    );
    };
export default AddPatient;
```

3.Προσθέτουμε μια φόρμα διαλόγου μέσα στη δήλωση επιστροφής. Η φόρμα περιέχει το στοιχείο Dialog Material-UI με τα κουμπιά και τα πεδία εισαγωγής που απαιτούνται για τη συλλογή των δεδομένων του ασθενή. Το κουμπί που ανοίγει το modal παράθυρο, το οποίο θα εμφανίζεται στη σελίδα με τη λίστα ασθενών, πρέπει να βρίσκεται εκτός του συστατικού Dialog. Όλα τα πεδία εισόδου πρέπει να έχουν το χαρακτηριστικό name με μια τιμή που είναι ίδια με το όνομα της state / κατάστασης, στην οποία θα αποθηκευτεί η τιμή. Τα πεδία εισόδου έχουν επίσης το χαρακτηριστικό onChange χειριστή, ο οποίος αποθηκεύει την τιμή στην κατάσταση με την κλήση της συνάρτησης handleChange. Οι συναρτήσεις handleClose και handleOpen ορίζουν την τιμή της κατάστασης open, η οποία επηρεάζει την ορατότητα της modal φόρμας:

```
  const [open, setOpen] = useState(false);
  const [patient, setPatient] = useState({
    lastName: "",
    firstName: "",
    birthDate: "",
    city: "",
    address: "",
    patientAmka: "",
    postcode: "",
    telephone: "",
    coronaPatient: "",
  });

 // Open the modal form
 const handleClickOpen = () => {
 setOpen(true);
 };

 // Close the modal form
 const handleClose = () => {
 setOpen(false);
 };

  const handleChange = (event) => {
    setPatient({ ...patient, [event.target.name]: event.target.value });
```

```
  };

return (
<div>
<button style={{margin: 10}} onClick={handleClickOpen}>New Patient</button>
<Dialog open={open} onClose={handleClose}>
<DialogTitle>New Patient</DialogTitle>
<DialogContent>
<input type="text" placeholder="LastName" name="lastName"
value={patient.lastName} onChange={handleChange}/><br/>
<input type="text" placeholder="FirstName" name="firstName"
value={patient.firstName} onChange={handleChange}/><br/>
<input type="text" placeholder="BirthDate" name="birthDate"
value={patient.birthDate} onChange={handleChange}/><br/>
<input type="text" placeholder="City" name="city"
value={patient.city} onChange={handleChange}/><br/>
<input type="text" placeholder="Address" name="address"
value={patient.address} onChange={handleChange}/><br/>
<input type="text" placeholder="PatientAmka" name="patientAmka"
value={patient.patientAmka} onChange={handleChange}/><br/>
<input type="text" placeholder="Postcode" name="postcode"
value={patient.postcode} onChange={handleChange}/><br/>
<input type="text" placeholder="Telephone" name="telephone"
value={patient.telephone} onChange={handleChange}/><br/>
</DialogContent>
<DialogActions>
<button onClick={handleClose}>Cancel</button>
<button onClick={handleClose}>Save</button>
</DialogActions>
</Dialog>
</div>
);
```

4. Υλοποιούμε τη συνάρτηση addPatient στο αρχείο Patientlist.js, η οποία θα στείλει το αίτημα POST στο τελικό σημείο api/patients του backend. Η αίτηση θα περιλαμβάνει το νέο αντικείμενο patient μέσα στο σώμα και την επικεφαλίδα /header 'Content-Type': 'application/json'. Η επικεφαλίδα /header είναι απαραίτητη επειδή το αντικείμενο patient μετατρέπεται σε μορφή JSON χρησιμοποιώντας τη μέθοδο JSON.stringify():

```
 // Add new Patient
 addPatient(patient) {
  fetch(SERVER_URL + "api/patients", {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
```

```
    },
    body: JSON.stringify(patient),
  })
    .then((res) => this.fetchPatients())
    .catch((err) => console.error(err));
}
```

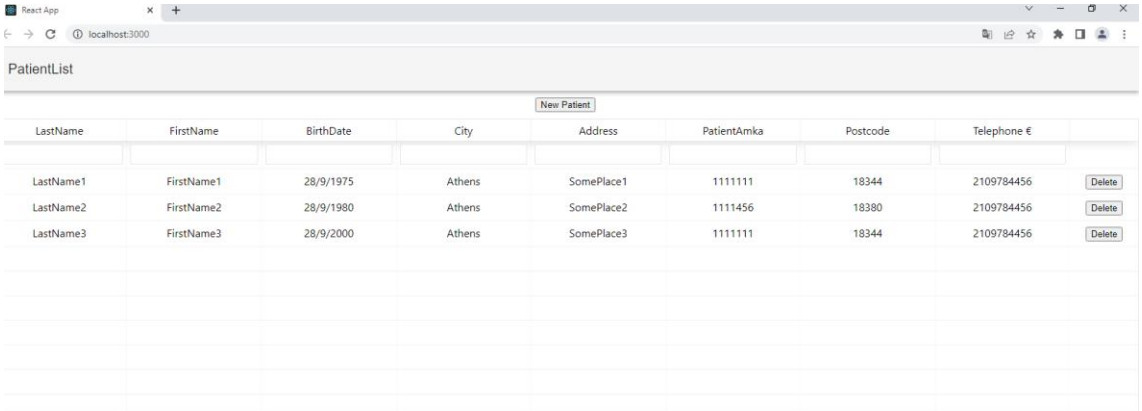
5.Εισάγουμε το στοιχείο AddPatient στο αρχείο Patientlist.js:

```
import AddPatient from "./AddPatient";
```

6.Προσθέτουμε το συστατικό AddPatient στη μέθοδο render() του αρχείου Patientlist.js και περνάμε τις συναρτήσεις addPatient και fetchPatients ως props στο συστατικό AddPatient. Αυτό μας επιτρέπει να καλέσουμε αυτές τις συναρτήσεις από το συστατικό AddPatient. Τώρα, η δήλωση επιστροφής του αρχείου Patientlist.js θα πρέπει να εμφανίζεται ως εξής:

```
// Patientlist.js
return (
        <div className="App">
        <AddPatient addPatient={this.addPatient} fetchPatients={this.fetchPatients } />
        <ReactTable data={this.state.patients} columns={columns}
        filterable={true} pageSize={10}/>
        <ToastContainer autoClose={1500}/>
        </div>
        );
```

7. Αν ξεκινήσουμε τώρα την εφαρμογή frontend, θα πρέπει να μοιάζει με την ακόλουθη, και αν πατήσουμε το κουμπί New Patient, θα πρέπει να ανοίξει η modal φόρμα:

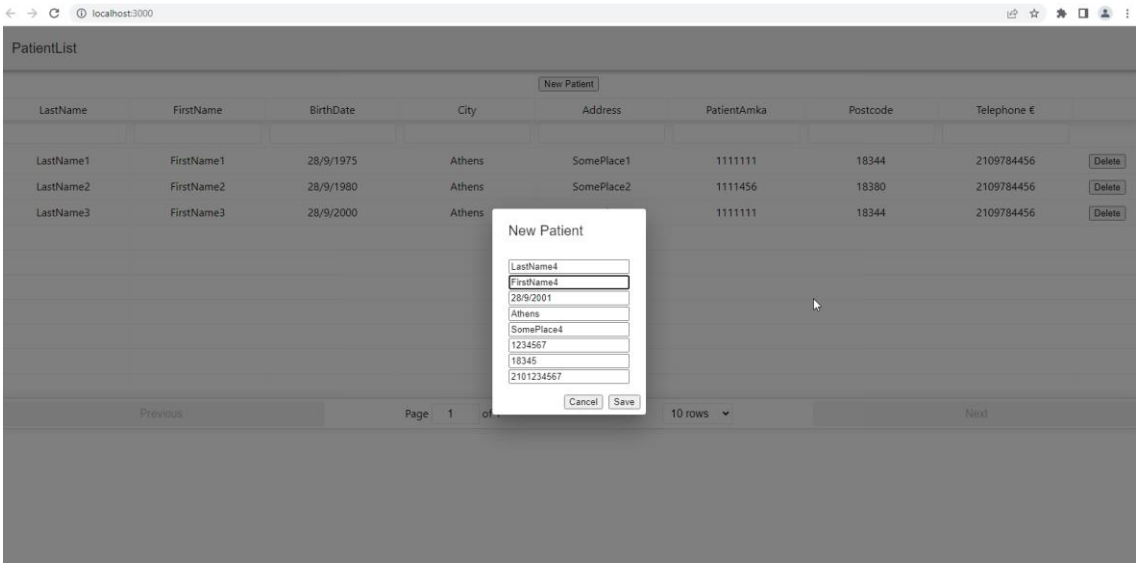


Εικόνα 44

8. Δημιουργούμε μια συνάρτηση με όνομα handleSave στο αρχείο AddPatient.js. Η συνάρτηση handleSave καλεί τη συνάρτηση addPatient, η οποία μπορεί να προσπελαστεί με τη χρήση props, και της περνάει το αντικείμενο κατάστασης του ασθενη. Τέλος, η modal φόρμα κλείνει και η λίστα ασθενών ενημερώνεται:

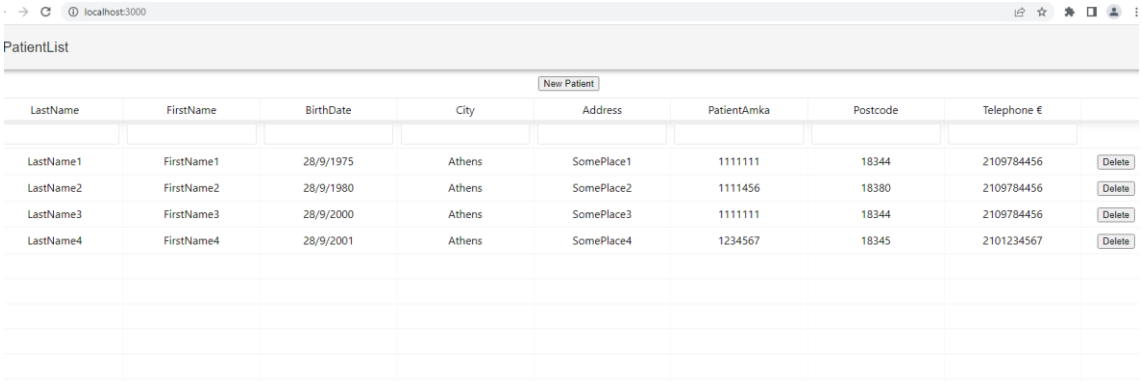
```
// Save patient and close modal form
 const handleSave = () => {
   props.addPatient(patient);
   handleClose();
 };
```

9. Τέλος, πρέπει αλλάζουμε το onClick του κουμπιού, ώστε να καλεί τη συνάρτηση handleSave, onClick= {handleSave}. Τώρα, μπορούμε να ανοίξουμε τη modal φόρμα πατώντας το κουμπί New Patient. Στη συνέχεια, μπορούμε να συμπληρώσουμε τη φόρμα με δεδομένα και να πατήσουμε το κουμπί Save.



Εικόνα 45

Η σελίδα της λίστας ανανεώνεται και ο νέος ασθενής μπορεί πλέον να εμφανιστεί:



Εικόνα 46

4.4.4 Η λειτουργικότητα επεξεργασίας και τα βήματα για την λειτουργικότητα επεξεργασίας

Θα υλοποιήσουμε τη λειτουργικότητα επεξεργασίας προσθέτοντας το κουμπί Επεξεργασία σε κάθε γραμμή του πίνακα. Όταν πατηθεί το κουμπί επεξεργασίας γραμμής, ανοίγει η modal φόρμα, όπου ο χρήστης μπορεί να επεξεργαστεί τον υπάρχοντα ασθενή και τέλος να αποθηκεύσει τις αλλαγές του:

1. Αρχικά, θα δημιουργήσουμε έναν σκελετό του συστατικού EditPatient, το οποίο θα είναι η φόρμα για την επεξεργασία ενός υπάρχοντος ασθενή. Δημιουργούμε ένα νέο αρχείο με όνομα EditPatient.js στο φάκελο components. Ο κώδικας του συστατικού EditPatient είναι παρόμοιος με το συστατικό AddPatient , αλλά προς το παρόν, στη συνάρτηση handleSave, θα πρέπει να καλέσουμε τη συνάρτηση update που θα υλοποιήσουμε αργότερα:

```
import React, { useState } from "react";
import Dialog from "@material-ui/core/Dialog";
import DialogActions from "@material-ui/core/DialogActions";
import DialogContent from "@material-ui/core/DialogContent";
import DialogTitle from "@material-ui/core/DialogTitle";

const EditPatient = (props) => {
  const [open, setOpen] = useState(false);
  const [patient, setPatient] = useState({
    lastName: "",
    firstName: "",
    birthDate: "",
    city: "",
    address: "",
    patientAmka: "",
    postcode: "",
    telephone: "",
    coronaPatient: "",
  });

 // Open the modal form
```

```
const handleClickOpen = () => {
setOpen(true);
};

// Close the modal form
const handleClose = () => {
setOpen(false);
};

 const handleChange = (event) => {
    setPatient({ ...patient, [event.target.name]: event.target.value });
  };


  // Update patient and close modal form
  const handleSave = () => {

};


return (
<div>
<button style={{margin: 10}} onClick={handleClickOpen}>Edit</button>
<Dialog open={open} onClose={handleClose}>
<DialogTitle>Edit patient</DialogTitle>
<DialogContent>
<input type="text" placeholder="LastName" name="lastName"
value={patient.lastName} onChange={handleChange}/><br/>
<input type="text" placeholder="FirstName" name="firstName"
value={patient.firstName} onChange={handleChange}/><br/>
<input type="text" placeholder="BirthDate" name="birthDate"
value={patient.birthDate} onChange={handleChange}/><br/>
<input type="text" placeholder="City" name="city"
value={patient.city} onChange={handleChange}/><br/>
<input type="text" placeholder="Address" name="address"
value={patient.address} onChange={handleChange}/><br/>
<input type="text" placeholder="PatientAmka" name="patientAmka"
value={patient.patientAmka} onChange={handleChange}/><br/>
<input type="text" placeholder="Postcode" name="postcode"
value={patient.postcode} onChange={handleChange}/><br/>
<input type="text" placeholder="Telephone" name="telephone"
value={patient.telephone} onChange={handleChange}/><br/>
</DialogContent>
<DialogActions>
```

```
  <button onClick={handleClose}>Cancel</button>
  <button onClick={handleSave}>Save</button>
  </DialogActions>
  </Dialog>
  </div>
  );
}

export default EditPatient;
```

2. Για να ενημερώσουμε τα δεδομένα του ασθενή θα πρέπει να στείλουμε το αίτημα PUT στη διεύθυνση URL http://localhost:8080/api/patients/[patientid]. Ο σύνδεσμος θα είναι ο ίδιος όπως και για τη λειτουργία διαγραφής. Η αίτηση περιέχει το ενημερωμένο αντικείμενο του patient/ασθενή μέσα στο σώμα και την επικεφαλίδα 'Content-Type': 'application/json' που είχαμε στη λειτουργικότητα προσθήκης. Δημιουργούμε μια νέα συνάρτηση με όνομα updatePatient στο αρχείο Patientlist.js. Ο πηγαίος κώδικας της συνάρτησης φαίνεται στο παρακάτω απόσπασμα κώδικα. Η συνάρτηση λαμβάνει δύο ορίσματα, το αντικείμενο του updated patient / ενημερωμένου ασθενή και τη διεύθυνση URL του αιτήματος. Μετά από μια επιτυχημένη ενημέρωση, θα εμφανίσουμε ένα μήνυμα πρότασης στον χρήστη:

```
// Patientlist.js file
 // Update patient
updatePatient(patient, link) {
  fetch(link,
  { method: 'PUT',
  headers: {
  'Content-Type': 'application/json',
  },
  body: JSON.stringify(patient)
  })
  .then(res => {
  toast.success("Changes saved", {
  position: toast.POSITION.BOTTOM_LEFT
  });
  this.fetchPatients();
  })
  .catch(err =>
  toast.error("Error when saving", {
  position: toast.POSITION.BOTTOM_LEFT
  })
  )
  }
```

3.Στη συνέχεια, θα εισάγουμε το στοιχείο EditPatient στο στοιχείο PatientList, ώστε να είμαστε σε θέση να το εμφανίσουμε στη λίστα ασθενών. Προσθέτουμε την ακόλουθη εισαγωγή στο αρχείο PatientList.js:

```
import Editpatient from './EditPatient';
```

4. Εν συνεχεία προσθέτουμε το στοιχείο EditPatient στις στήλες του πίνακα με τον ίδιο τρόπο που κάναμε με τη λειτουργία διαγραφής. Τώρα, το συστατικό EditPatient αποδίδεται στα κελιά του πίνακα και εμφανίζει μόνο το κουμπί Edit (Επεξεργασία). Αυτό συμβαίνει επειδή η modal φόρμα δεν είναι ορατή πριν πατηθεί το κουμπί. Όταν ο χρήστης πατήσει το κουμπί Edit, θέτει την τιμή open state σε true στο στοιχείο EditPatient και εμφανίζεται η modal φόρμα. Ουσιαστικά περνάμε τέσσερα props /στηρίγματα στο συστατικό EditPatient. Το πρώτο props είναι το row, το οποίο περιέχει όλες τις τιμές της γραμμής ως αντικείμενο (αντικείμενο patient). Το δεύτερο props είναι το value, το οποίο ορίζεται ως _links.href.self, το οποίο θα είναι η διεύθυνση URL του patient που χρειαζόμαστε στην αίτηση. Το τρίτο είναι η συνάρτηση updatePatient, την οποία πρέπει να καλέσουμε από το στοιχείο EditPatient για να μπορέσουμε να αποθηκεύσουμε τις αλλαγές. Η τελευταία είναι η συνάρτηση fetchPatients, η οποία χρησιμοποιείται για την ανανέωση της λίστας ασθενών μετά από μια ενημέρωση:

```
const columns = [{
                Header: 'LastName',
                    accessor: 'lastName'
                    }, {
                    Header: 'FirstName',
                    accessor: 'firstName',
                    }, {
                    Header: 'BirthDate',
                    accessor: 'birthDate',
                    }, {
                    Header: 'City',
                    accessor: 'city'
                    }, {
                    Header: 'Address',
                    accessor: 'address',
                    }, {
                    Header: 'PatientAmka',
                    accessor: 'patientAmka',
                    }, {
                    Header: 'Postcode',
                    accessor: 'postcode',
                    }, {
                    Header: 'Telephone €',
                    accessor: 'telephone',
                    },
            {
              Header: "Edit",
              sortable: false,
              filterable: false,
              width: 100,
              accessor: "_links.self.href",
              Cell: ({ value, row }) => (
              <EditPatient
               patient={row}
```

```
            link={value}
            updatePatient={this.updatePatient}
            fetchPatients={this.fetchPatients}
            />
            ),
          },
           {
          id: 'delbutton',
          sortable: false,
          filterable: false,
          width: 100,
          accessor: '_links.self.href',
          Cell: ({value}) => (<button onClick={()=>{this.onDelClick(value)}}>Delete</button>)
          },
          ];
```

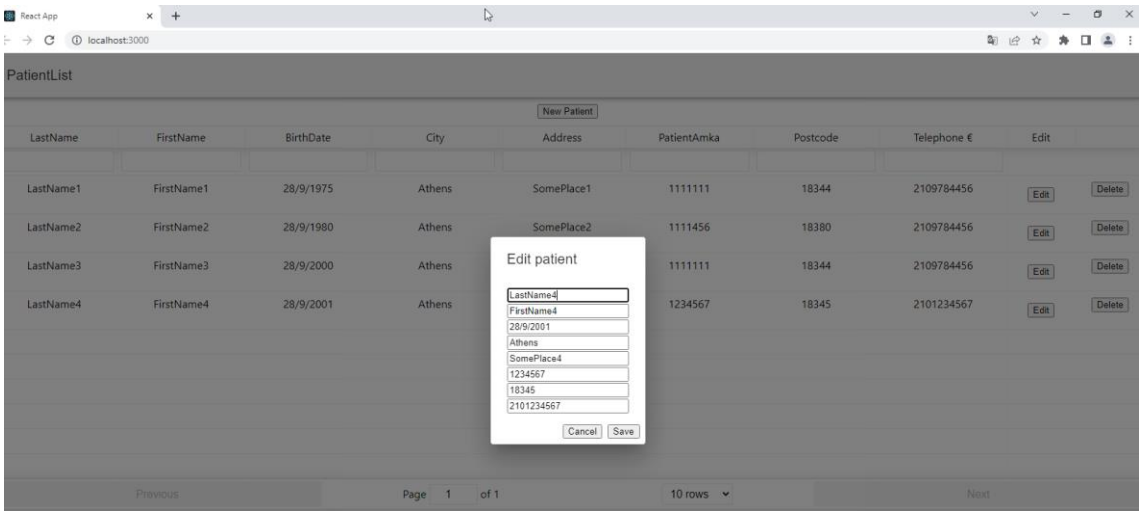
5. Στη συνέχεια, θα πραγματοποιήσουμε τις τελικές τροποποιήσεις στο αρχείο EditPatient.js. Παίρνουμε τον προς επεξεργασία patient από τα props/στηρίγματα των patient, τα οποία χρησιμοποιούμε για να συμπληρώσουμε τη φόρμα με τις υπάρχουσες τιμές των patient. Αλλάζουμε τη συνάρτηση handleClickOpen στο αρχείο EditPatient.js. Τώρα, όταν ανοίγει η φόρμα, η κατάσταση του patient/ασθενή ενημερώνεται με τις τιμές από τα props του patient /ασθενή:

```
const handleClickOpen = () => {
   setPatient({
     lastName: props.patient.lastName,
     firstName: props.patient.firstName,
     birthDate: props.patient.birthDate,
     city: props.patient.city,
     address: props.patient.address,
     patientAmka: props.patient.patientAmka,
     postcode: props.patient.postcode,
     telephone: props.patient.telephone,
     coronaPatient: props.patient._original.coronaPatient,
   });
   setOpen(true);
 };
```

6.Τέλος, θα αλλάξουμε τη συνάρτηση handleSave και θα καλέσουμε τη συνάρτηση updatePatient χρησιμοποιώντας props:

```
// Update patient and close modal form
 const handleSave = () => {
   props.updatePatient(patient, props.link);
   handleClose();
 };
```

7.Αν πατήσουμε το κουμπί Edit (Επεξεργασία) στον πίνακα, ανοίγει η modal φόρμα επεξεργασίας και εμφανίζεται ο ασθενής της συγκεκριμένης γραμμής. Οι ενημερωμένες τιμές αποθηκεύονται στη βάση δεδομένων όταν πατήσουμε το κουμπί Save:



Εικόνα 47

4.4 Άλλες λειτουργικότητες για το FrontEnd

Μια λειτουργία που θα υλοποιήσουμε επίσης, είναι η εξαγωγή των δεδομένων με τιμές διαχωρισμένες με κόμμα (CSV). Υπάρχει ένα πακέτο που ονομάζεται react-csv (https://github.com/abdennour/react-csv) που μπορεί να χρησιμοποιηθεί για την εξαγωγή ενός πίνακα δεδομένων στο αρχείο CSV.

Πληκτρολογούμε την ακόλουθη εντολή για να εγκαταστήσουμε το react-csv:

```
npm install react-csv
```

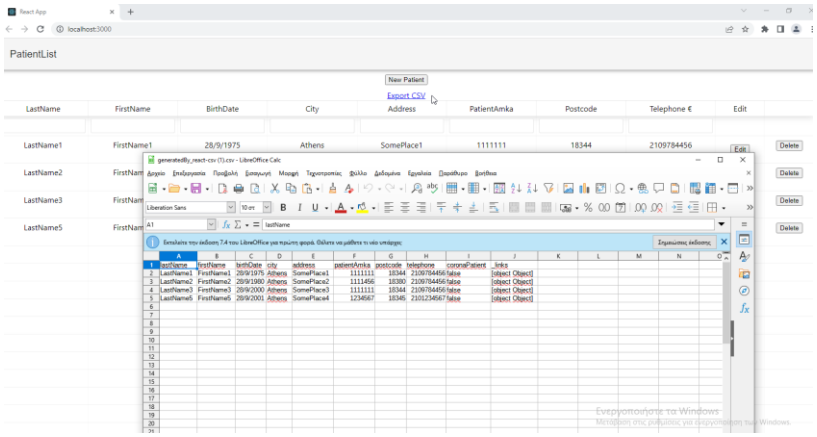
Το πακέτο react-csv περιέχει δύο συστατικά: CSVLink και CSVDownload. Θα χρησιμοποιήσουμε το πρώτο στην εφαρμογή μας, οπότε προσθέτουμε την ακόλουθη εισαγωγή στο αρχείο Patientlist.js:

```
import { CSVLink } from 'react-csv';
```

Το συστατικό CSVLink λαμβάνει τα prop data, το οποίο περιέχει τον πίνακα δεδομένων που θα εξαχθεί στο αρχείο CSV. Προσθέτουμε το συστατικό CSVLink μέσα στη δήλωση επιστροφής στη μέθοδο render().
Η τιμή του prop data θα είναι τώρα this.state.patients:

```
return (
        <div className="App">
        <AddPatient addPatient={this.addPatient } fetchPatients={this.fetchPatients} />
        <CSVLink data={this.state.patients} separator=";">Export CSV</CSVLink>
        <ReactTable data={this.state.patients} columns={columns}
        filterable={true}/>
        <ToastContainer autoClose={1500} />
        </div>
        );
```

Ανοίγουμε την εφαρμογή στο πρόγραμμα περιήγησης και θα δούμε τον σύνδεσμο Export CSV στην εφαρμογή μας. Αν πατήσουμε το σύνδεσμο, θα λάβουμε τα δεδομένα σε μορφή αρχείου csv:



Εικόνα 48

4.6 Διαμόρφωση του frontend με το React Material-UI

4.6.1 Χρήση του Button component του Material-UI

Θα αλλάξουμε πρώτα όλα τα κουμπιά, ώστε να χρησιμοποιούν το συστατικό Material-UI Button. Εισάγουμε το στοιχείο Button στο αρχείο AddPatient.js:

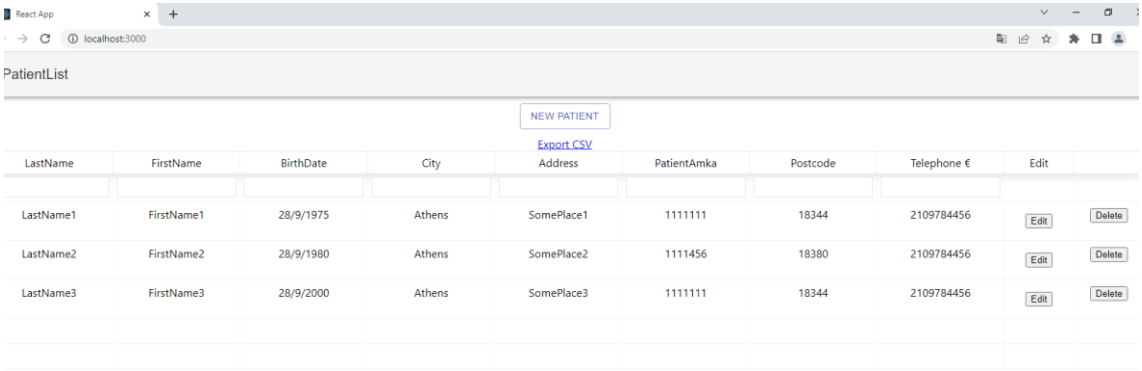
```
// AddPatient.js
import Button from '@material-ui/core/Button';
```

Στη συνέχεια, θα αλλάξουμε τα κουμπιά ώστε να χρησιμοποιούν το στοιχείο Button.Ο παρακάτω κώδικας δείχνει το συστατικό AddPatient:

```
 return (
   <div>
   <Button variant="outlined" color="primary" style={{margin: 10}}
onClick={handleClickOpen}>New Patient</Button>
   <Dialog open={open} onClose={handleClose}>
   <DialogTitle>New Patient</DialogTitle>
   <DialogContent>
   <input type="text" placeholder="LastName" name="lastName"
   value={patient.lastName} onChange={handleChange}/><br/>
   <input type="text" placeholder="FirstName" name="firstName"
   value={patient.firstName} onChange={handleChange}/><br/>
   <input type="text" placeholder="BirthDate" name="birthDate"
   value={patient.birthDate} onChange={handleChange}/><br/>
   <input type="text" placeholder="City" name="city"
   value={patient.city} onChange={handleChange}/><br/>
   <input type="text" placeholder="Address" name="address"
   value={patient.address} onChange={handleChange}/><br/>
   <input type="text" placeholder="PatientAmka" name="patientAmka"
   value={patient.patientAmka} onChange={handleChange}/><br/>
   <input type="text" placeholder="Postcode" name="postcode"
   value={patient.postcode} onChange={handleChange}/><br/>
   <input type="text" placeholder="Telephone" name="telephone"
   value={patient.telephone} onChange={handleChange}/><br/>
   </DialogContent>
   <DialogActions>
   <Button color="secondary" onClick={handleClose}>Cancel</Button>
   <Button color="primary" onClick={handleSave}>Save</Button>
   </DialogActions>
   </Dialog>
   </div>
```

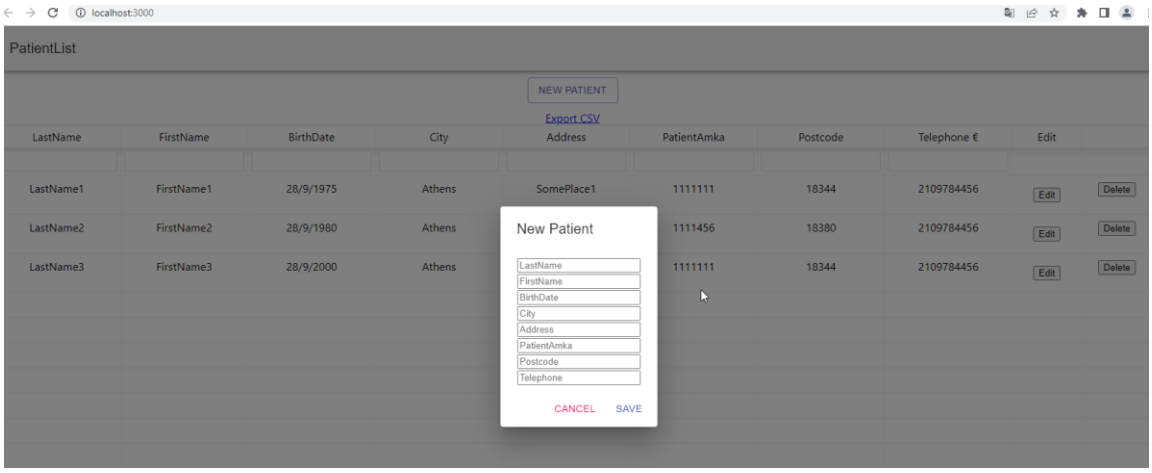
```
);
```

Τώρα, το κουμπί της σελίδας λίστας ασθενών θα πρέπει να μοιάζει με το ακόλουθο:



Εικόνα 49

Και τα κουμπιά της modal φόρμας θα πρέπει να μοιάζουν με τα ακόλουθα:



Εικόνα 50

Θα πρέπει επίσης να αλλάξουμε και τα κουμπιά στο συστατικό EditPatient component. Το κουμπί που ανοίγει τη modal φόρμα είναι το κουμπί Edit, το οποίο φαίνεται στον πίνακα. Επομένως, χρησιμοποιούμε το κουμπί χωρίς περιθώρια και ρυθμίζουμε το μέγεθός του σε μικρό. Ακολουθεί ο πηγαίος κώδικας του συστατικού EditPatient:

```
return (
    <div>
    <Button color="primary" size="small" onClick={handleClickOpen}>Edit</Button>
    <Dialog open={open} onClose={handleClose}>
    <DialogTitle>Edit patient</DialogTitle>
    <DialogContent>
    <input type="text" placeholder="LastName" name="lastName"
    value={patient.lastName} onChange={handleChange}/><br/>
    <input type="text" placeholder="FirstName" name="firstName"
```

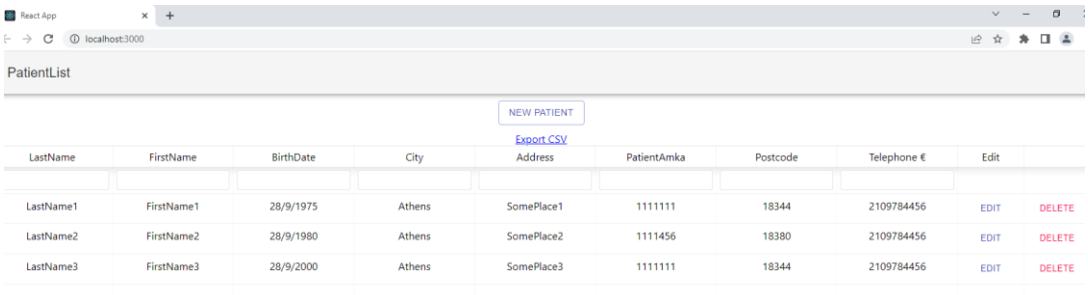
```
value={patient.firstName} onChange={handleChange}/><br/>
<input type="text" placeholder="BirthDate" name="birthDate"
value={patient.birthDate} onChange={handleChange}/><br/>
<input type="text" placeholder="City" name="city"
value={patient.city} onChange={handleChange}/><br/>
<input type="text" placeholder="Address" name="address"
value={patient.address} onChange={handleChange}/><br/>
<input type="text" placeholder="PatientAmka" name="patientAmka"
value={patient.patientAmka} onChange={handleChange}/><br/>
<input type="text" placeholder="Postcode" name="postcode"
value={patient.postcode} onChange={handleChange}/><br/>
<input type="text" placeholder="Telephone" name="telephone"
value={patient.telephone} onChange={handleChange}/><br/>
</DialogContent>
<DialogActions>
<Button color="secondary" onClick={handleClose}>Cancel</Button>
<Button color="primary" onClick={handleSave}>Save</Button>
</DialogActions>
</Dialog>
</div>
);
```

Χρησιμοποιούμε επίσης το συστατικό Button στον πίνακα ασθενών και ορίζουμε το μέγεθος του κουμπιού ως μικρό για το κουμπί Delete στο αρχείο patientlist.js

```
// Patientlist.js render() method
const columns = [{
                Header: 'LastName',
                    accessor: 'lastName'
                    }, {
                    Header: 'FirstName',
                    accessor: 'firstName',
                    }, {
                    Header: 'BirthDate',
                    accessor: 'birthDate',
                    }, {
                    Header: 'City',
                    accessor: 'city'
                    }, {
                    Header: 'Address',
                    accessor: 'address',
                    }, {
                    Header: 'PatientAmka',
                    accessor: 'patientAmka',
                    }, {
```

```
            Header: 'Postcode',
            accessor: 'postcode',
            }, {
            Header: 'Telephone €',
            accessor: 'telephone',
            },
        {
          Header: "Edit",
          sortable: false,
          filterable: false,
          width: 100,
          accessor: "_links.self.href",
          Cell: ({ value, row }) => (
          <EditPatient
           patient={row}
          link={value}
          updatePatient={this.updatePatient}
          fetchPatients={this.fetchPatients}
          />
          ),
        },
         {
        id: 'delbutton',
        sortable: false,
        filterable: false,
        width: 100,
        accessor: '_links.self.href',
        /* Cell: ({value}) => (<button onClick={()=>{this.onDelClick(value)}}>Delete</button>)
        }, */
        Cell: ({value}) => (<Button size="small" color="secondary"
        onClick={()=>{this.onDelClick(value)}}>Delete</Button>)},
        ];
```

Τώρα, ο πίνακας θα πρέπει να μοιάζει με τα ακόλουθα:



Εικόνα 51

4.7 Χρήση του Grid component

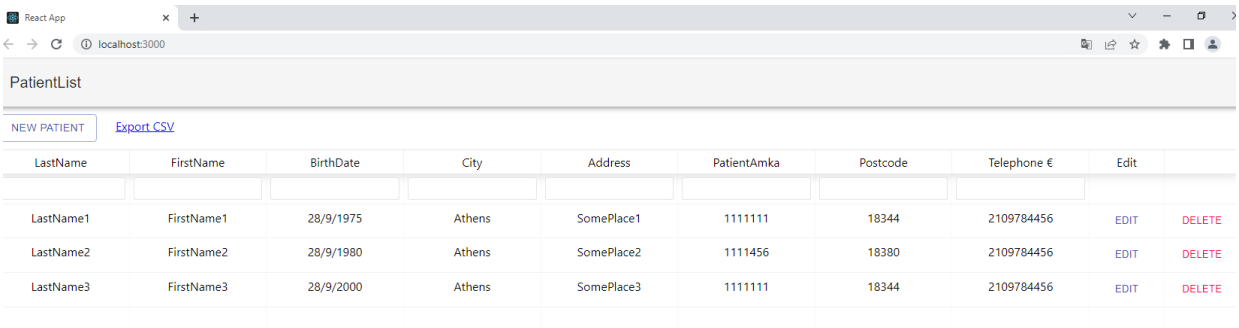
Το Material-UI παρέχει ένα συστατικό Grid που μπορεί να χρησιμοποιηθεί για να αποκτήσει η εφαρμογή μας React μια διάταξη πλέγματος. Θα χρησιμοποιήσουμε το Grid για να πάρουμε το κουμπί New Item και το σύνδεσμο Export CSV στην ίδια γραμμή.

Προσθέτουμε το ακόλουθο import στο αρχείο Patientlist.js για να εισάχθει το συστατικό Grid:

```
import Grid from '@material-ui/core/Grid';
```

Στη συνέχεια, ενσωματώνουμε το AddPatient και το CSVLink στα στοιχεία Grid. Υπάρχουν δύο τύποι συστατικών Grid - ένας περιέκτης /container και ένα στοιχείο /item. Τα AddPatient και CSVLink είναι ενσωματωμένα μέσα στα στοιχεία Grid τύπου item. Στη συνέχεια, και τα δύο στοιχεία Grid τύπου item ενσωματώνονται στο στοιχείο Grid τύπου container:

```
//Patientlist.js render() method
 return (
            <div className="App">
              <Grid container>
                <Grid item>
                  <AddPatient addPatient={this.addPatient } fetchPatients={this.fetchPatients } />
                </Grid>
                <Grid item style={{padding: 15}}>
                  <CSVLink data={this.state.patients} separator=";">Export CSV</CSVLink>
                </Grid>
              </Grid>
              <ReactTable data={this.state.patients} columns={columns}
                filterable={true}/>
              <ToastContainer autoClose={1500} />
            </div>
          );
```



Εικόνα 52

Τώρα, η εφαρμογή θα πρέπει να μοιάζει ως ακολούθως:

Το κουμπί και ο σύνδεσμος CSV τοποθετούνται σε μία γραμμή πάνω αριστερά.

4.8 Χρήση των TextField components / στοιχείων

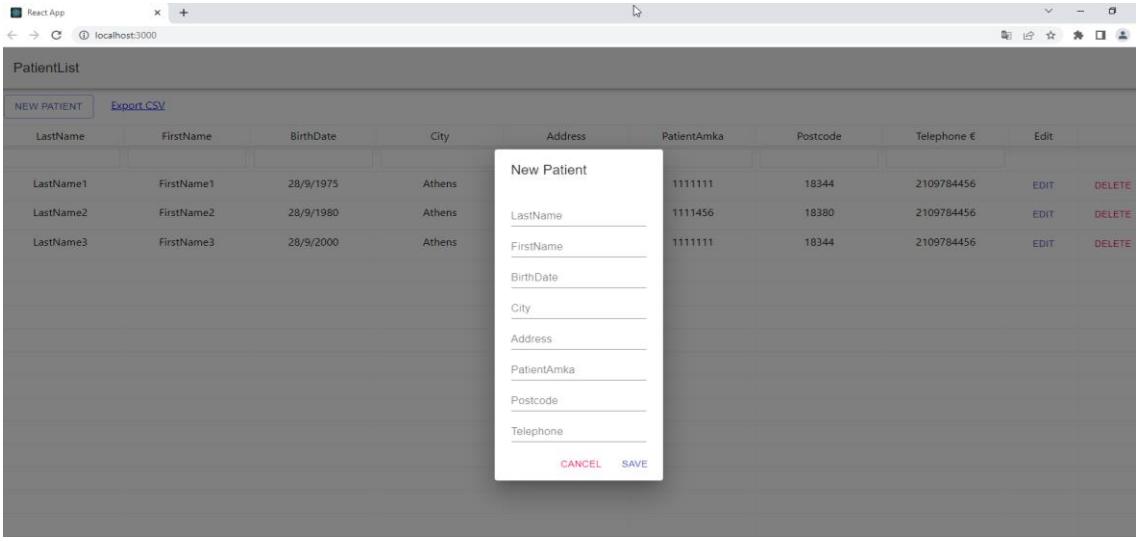
Στο σημείο αυτό θα αλλάξουμε την είσοδο κειμένου στη modal φόρμα χρησιμοποιώντας το συστατικό Material-UI TextField. Προσθέτουμε την ακόλουθη δήλωση import στα αρχεία AddPatient.js και EditPatient.js:

```
import TextField from '@material-ui/core/TextField';
```

Στη συνέχεια, αλλάζουμε την input /είσοδο στα στοιχεία σε TextField στις φόρμες προσθήκης και επεξεργασίας. Χρησιμοποιούμε τα props label για να ορίσουμε τις ετικέτες των στοιχείων TextField. Το πρώτο στοιχείο TextField περιέχει τα props autoFocus και η είσοδος θα εστιάζεται σε αυτό το πεδίο:

```
return (
    <div>
    <Button variant="outlined" color="primary" style={{margin: 10}}
onClick={handleClickOpen}>New Patient</Button>
    <Dialog open={open} onClose={handleClose}>
    <DialogTitle>New Patient</DialogTitle>
    <DialogContent>
    <TextField autoFocus fullWidth label="LastName"
name="lastName"  value={patient.lastName} onChange={handleChange}/><br/>
    <TextField fullWidth  label="FirstName" name="firstName" value={patient.firstName}
onChange={handleChange}/><br/>
    <TextField fullWidth  label="BirthDate" name="birthDate" value={patient.birthDate}
onChange={handleChange}/><br/>
    <TextField fullWidth  label="City" name="city"          value={patient.city}
onChange={handleChange}/><br/>
    <TextField fullWidth  label="Address" name="address"     value={patient.address}
onChange={handleChange}/><br/>
    <TextField fullWidth  label="PatientAmka" name="patientAmka" value={patient.patientAmka}
onChange={handleChange}/><br/>
    <TextField fullWidth  label="Postcode" name="postcode"    value={patient.postcode}
onChange={handleChange}/><br/>
    <TextField fullWidth  label="Telephone" name="telephone"  value={patient.telephone}
onChange={handleChange}/><br/>
    </DialogContent>
    <DialogActions>
    <Button color="secondary" onClick={handleClose}>Cancel</Button>
    <Button color="primary" onClick={handleSave}>Save</Button>
    </DialogActions>
    </Dialog>
    </div>
    );
```

Μετά τις τροποποιήσεις, η modal φόρμα θα πρέπει να έχει την ακόλουθη μορφή:



Εικόνα 53

4.8 Ασφάλιση της όλης εφαρμογής

4.8.1 Ασφάλιση του Backend και τα βήματα ασφάλισης του Backend

Έχουμε υλοποιήσει λειτουργίες CRUD στο frontend ,χρησιμοποιώντας ένα μη ασφαλές backend. Εν συνεχεία θα ενεργοποιήσουμε την ασφάλεια για το backend.

1. Ανοίγουμε το backend project και πάμε στο αρχείο SecurityConfig.java.Έχουμε σχολιάσει την ασφάλεια και έχουμε επιτρέψει σε όλους την πρόσβαση σε όλα τα τελικά σημεία. Τώρα, μπορούμε να αφαιρέσουμε αυτή τη γραμμή. Πλέον, η μέθοδος configure του αρχείου SecurityConfig .java θα πρέπει να μοιάζει με την ακόλουθη:

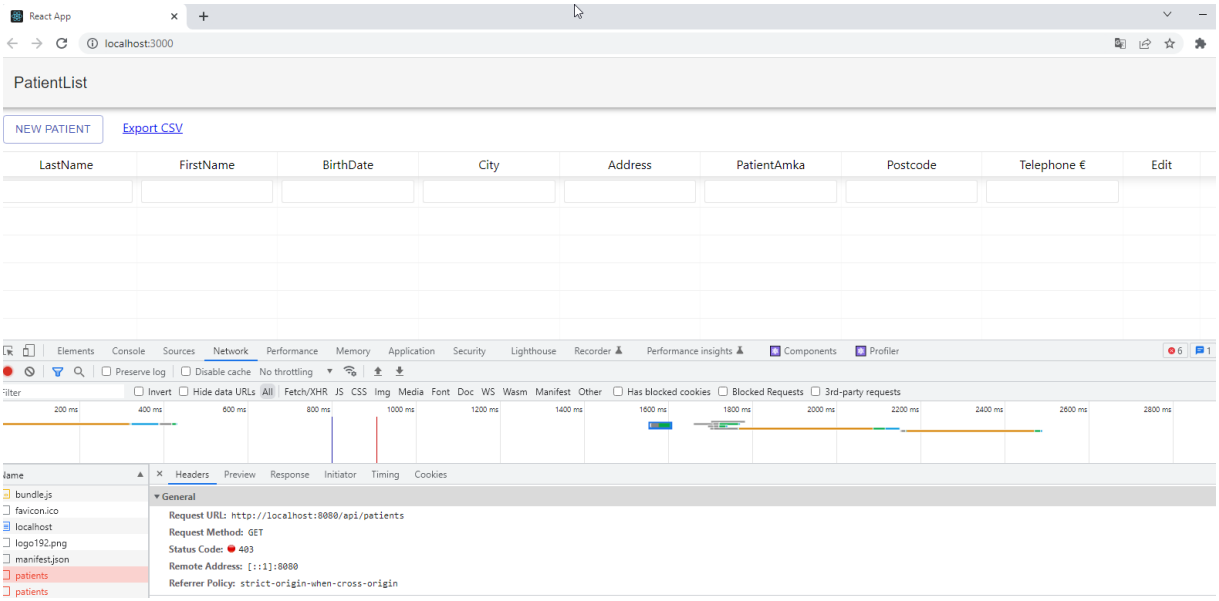
```
@Override
protected void configure(HttpSecurity http) throws Exception {
   // Add this row to allow access to all endpoints
//http.csrf().disable().cors().and().authorizeRequests().anyRequest().permitAll();
   http.csrf().disable().cors().and().authorizeRequests()
         .antMatchers(HttpMethod.POST, "/login").permitAll()
         .anyRequest().authenticated()
         .and()
         .addFilterBefore(new LoginFilter("/login", authenticationManager()),
               UsernamePasswordAuthenticationFilter.class)
         .addFilterBefore(new AuthenticationFilter(),
               UsernamePasswordAuthenticationFilter.class);
}
```

Στο σημείο αυτό θα δοκιμάσουμε να δούμε τι συμβαίνει με το Front End έχοντας ασφαλίσει το backend.

2. Εκτελούμε το backend πατώντας το κουμπί Run στο Itellij και ελέγουμε από την προβολή Console ότι η εφαρμογή ξεκινάει σωστά.

3. Εκτελούμε το frontend πληκτρολογώντας την εντολή npm start, μέσα από το σωστό φάκελό του Front End react project, στο Terminal και το πρόγραμμα περιήγησης θα πρέπει να ανοίξει στη διεύθυνση localhost:3000.

4. Θα πρέπει τώρα να δουμε ότι η σελίδα της λίστας και ο πίνακας είναι άδειοι. Αν ανοίξουμε τα εργαλεία ανάπτυξης του φυλλομετρητή, θα παρατηρήσουμε ότι η αίτηση καταλήγει σε σφάλμα HTTP 403 Forbidden.



Εικόνα 54

4.8.2 Ασφάλιση του Frontend και βήματα δημιουργίας του Login component

Ο έλεγχος ταυτότητας υλοποιήθηκε στο backend με τη χρήση JWT.
Δημιουργήσαμε τον έλεγχο ταυτότητας JWT και το Login ως τελικό σημείο /login endpoint επιτρέπει την πρόσβαση σε όλους χωρίς έλεγχο ταυτότητας.

Οπότε θα πρέπει να δημιουργήσουμε μια σελίδα Login στο Frontend η οποία θα είναι υπεύθυνη να ΄΄μιλά΄΄ με το backend, ώστε να λάβει το token. Μετά από αυτό, το token θα συμπεριλαμβάνεται σε όλες τις αιτήσεις που στέλνουμε στο backend από το frontend.

Ας δημιουργήσουμε πρώτα ένα στοιχείο σύνδεσης που ζητάει διαπιστευτήρια από τον χρήστη για να πάρει ένα token από το backend.

1. Δημιουργούμε ένα νέο αρχείο, με όνομα Login.js, στο φάκελο components.

2. Ανοίγουμε τον VS Code editor και προσθέτουμε τον ακόλουθο βασικό κώδικα στο συστατικό login. Εισάγουμε επίσης το SERVER_URL, επειδή απαιτείται σε ένα αίτημα σύνδεσης προς το Backend

```
import React, { useState } from 'react';
import {SERVER_URL} from '../constants.js';

const Login = () => {
return (
<div>
</div>
);
```

```
}

export default Login;
```

3.Χρειαζόμαστε τρεις state values/τιμές κατάστασης για authentication/ τον έλεγχο ταυτότητας, δύο για τα διαπιστευτήρια ((username/όνομα χρήστη και password/κωδικός πρόσβασης) και μία τιμή Boolean για να υποδεικνύει την κατάσταση του ελέγχου ταυτότητας. Η προεπιλεγμένη τιμή της κατάστασης ελέγχου ταυτότητας είναι false. Εισάγουμε states/καταστάσεις χρησιμοποιώντας τη συνάρτηση useState:

```
const [user, setUser] = useState({username: '', password: ''})
const [isAuthenticated, setAuth] = useState(false);
```

4. Στη διεπαφή χρήστη, θα χρησιμοποιήσουμε τη βιβλιοθήκη συστατικών Material-UI, όπως κάναμε και με την υπόλοιπη διεπαφή χρήστη. Χρειαζόμαστε στοιχεία πεδίου κειμένου για τα διαπιστευτήρια και ένα κουμπί για την κλήση μιας λειτουργίας σύνδεσης. Προσθέτουμε τις εισαγωγές για τα συστατικά στο αρχείο login.js:

```
import TextField from '@material-ui/core/TextField';
import Button from '@material-ui/core/Button';
```

5. Όποτε τώρα στο return statement./ δήλωση επιστροφής, προσθέτουμε δύο στοιχεία TextField, ένα για το όνομα χρήστη και ένα για τον κωδικό πρόσβασης καθώς και ένα συστατικό Button, αφού το χρειάζεται για την κλήση της συνάρτησης login.

```
return (
    <div>
    <TextField name="username"
    label="Username" onChange={handleChange} /><br/>
    <TextField type="password" name="password"
    label="Password" onChange={handleChange} /><br/><br/>
    <Button variant="outlined" color="primary"
    onClick={login}>
    Login
    </Button>
    </div>
    );
```

6. Υλοποιούμε το handleChange για τα στοιχεία TextField, προκειμένου να αποθηκεύονται οι πληκτρολογημένες τιμές στις καταστάσεις:

```
const handleChange = (event) => {
  setUser({ ...user, [event.target.name]: event.target.value });
 };
```

7. Όπως στα προηγούμενα, η σύνδεση με το backend γίνεται με την κλήση του τελικού σημείου /login χρησιμοποιώντας τη μέθοδο POST και στέλνοντας το αντικείμενο χρήστη μέσα στο σώμα, εάν η πιστοποίηση είναι επιτυχής, λαμβάνουμε ένα token σε μια επικεφαλίδα Authorization της απάντησης.

Στη συνέχεια, θα αποθηκεύσουμε το token στο session storage και θα θέσουμε την τιμή της κατάστασης isAuthenticated σε true. Η αποθήκευση συνόδου είναι παρόμοια με την τοπική αποθήκευση, αλλά διαγράφεται όταν τελειώνει μια σύνοδος στην εκάστοτε σελίδα. Όταν η τιμή της κατάστασης isAuthenticated αλλάζει, η διεπαφή χρήστη εμφανίζεται και παρουσιάζεται εκ νέου:

```
const login = () => {
      fetch(SERVER_URL + 'login', {
      method: 'POST',
      body: JSON.stringify(user)
      })
      .then(res => {
      const jwtToken = res.headers.get('Authorization');
      if (jwtToken !== null) {
      sessionStorage.setItem("jwt", jwtToken);
      setAuth(true);
      }
      })
      .catch(err => console.error(err))
      }
```

8.Μπορούμε να υλοποιήσουμε την απόδοση υπό όρους, η οποία απεικονίζει το στοιχείο Login εάν η κατάσταση isAuthenticated είναι false, ή το στοιχείο Patientlist εάν η κατάσταση isAuthenticated είναι true. Όμως θα πρέπει πρώτα να εισάγουμε το συστατικό Patientlist στο συστατικό Login:

```
import Patientlist from "./Patientlist";
```

Στη συνέχεια, πρέπει να υλοποιήσουμε τις ακόλουθες αλλαγές στη δήλωση επιστροφής:

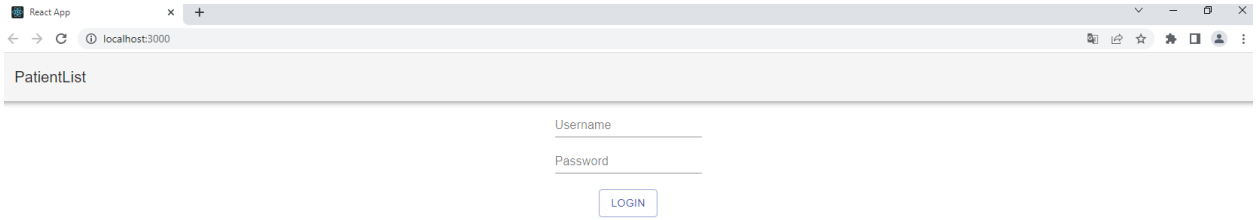
```
  if (isAuthenticated === true) {
        return <Patientlist />;
      } else {
        return (
          <div>
            <TextField name="username" label="Username" onChange={handleChange} />
            <br />
            <TextField
              type="password"
              name="password"
              label="Password"
              onChange={handleChange}
            />
            <br />
            <br />
            <Button variant="outlined" color="primary" onClick={login}>
              Login
```

```
            </Button>
            <ToastContainer autoClose={1500} />
          </div>
        );
      }
    };
```

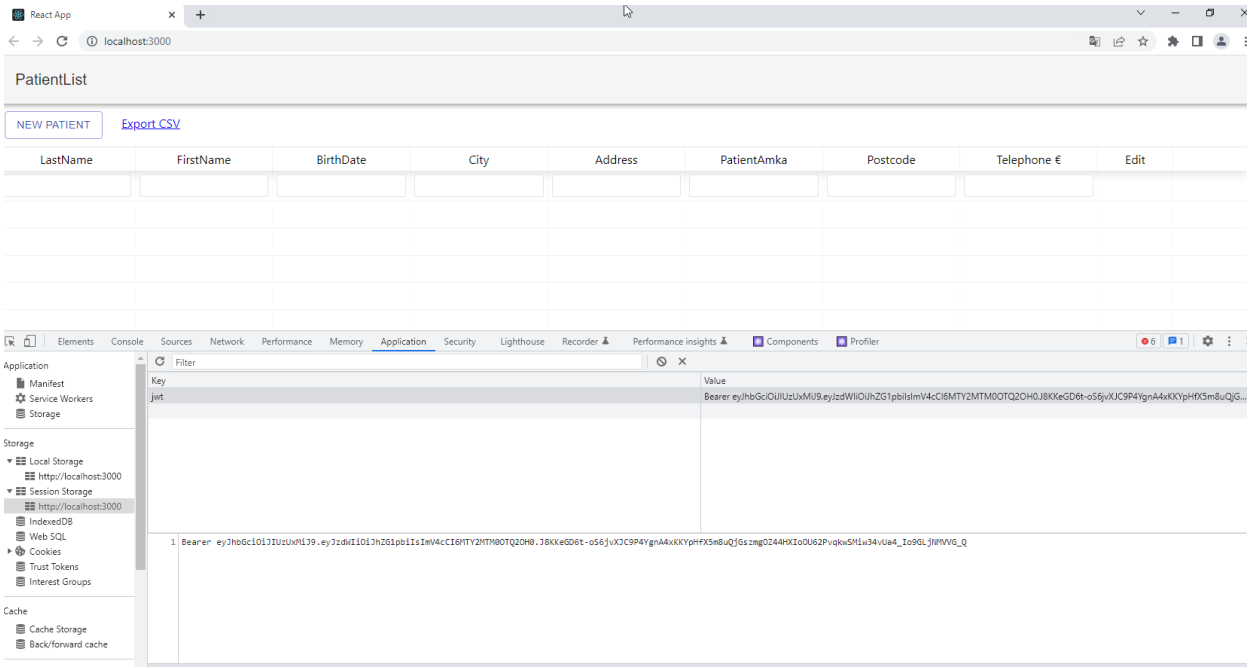
9.Για να εμφανίσουμε τη φόρμα σύνδεσης, πρέπει να αποδώσουμε στο αρχείο App.js το συστατικό Login αντί για το συστατικό Patientlist:

```
// App.js
import React from 'react';
import './App.css';
import AppBar from '@material-ui/core/AppBar';
import Toolbar from '@material-ui/core/Toolbar';
import Typography from '@material-ui/core/Typography';
import Login from './components/Login';

function App() {
return (
<div className="App">
<AppBar position="static" color="default">
<Toolbar>
<Typography variant="h6" color="inherit">
PatientList
</Typography>
</Toolbar>
</AppBar>
<Login />
</div>
);
}

export default App;

export default App;
```

Τώρα, όταν τρέχουν το frontend και το backend, το frontend θα πρέπει να μοιάζει με το ακόλουθο στιγμιότυπο οθόνης:
Εάν συνδεθούμε χρησιμοποιώντας τα διαπιστευτήρια admin/admin η user/user, θα πρέπει να δούμε τη σελίδα **PatientList**.
Επίσης αν ανοίξουμε τα εργαλεία προγραμματιστή, θα ανακαλύψουμε ότι το token έχει πλέον αποθηκευτεί στο session storage/ αποθήκευση συνεδρίας:



Εικόνα 55



Εικόνα 56

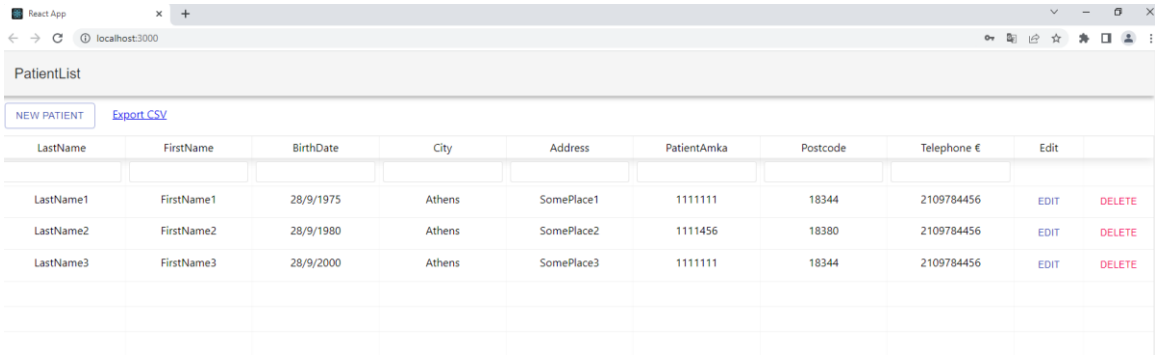
Η λίστα ασθενών είναι ακόμα άδεια, αλλά αυτό είναι σωστό, επειδή δεν έχουμε συμπεριλάβει ακόμα το token στην αίτηση.

Ακολουθούν τα βήματα για την άντληση ασθενών από το FrontEnd διαμέσου JWT από το Backend:

1. Ανοίγουμε το αρχείο **Patientlist.js** στην προβολή VS Code editor. Για να αντλήσουμε τους ασθενείς πρέπει πρώτα να διαβάσουμε το token από το session storage και στη συνέχεια να προσθέσουμε την επικεφαλίδα Authorization με την τιμή του token στην αίτηση.

```
// PAtientlist.js
// Fetch all patients
 // Fetch all patients
  fetchPatients = () => {
    // Read the token from the session storage
    // and include it to Authorization header
    const token = sessionStorage.getItem("jwt");
    fetch(SERVER_URL + "api/patients", {
      headers: { Authorization: token },
    })
      .then((response) => response.json())
      .then((responseData) => {
        this.setState({
          patients: responseData._embedded.patients,
        });
      })
      .catch((err) => console.error(err));
  };
```

2. Αν συνδεθούμε τώρα στο frontend θα πρέπει να δούμε τη λίστα ασθενών συμπληρωμένη με τους test ασθενείς από τη βάση δεδομένων:



Εικόνα 57

Όλες οι άλλες λειτουργίες CRUD απαιτούν την ίδια τροποποίηση για να λειτουργήσουν σωστά. Ο πηγαίος κώδικας της συνάρτησης delete εμφανίζεται ως εξής, μετά τις τροποποιήσεις:

```
// Delete patient
  onDelClick = (link) => {
    if (window.confirm("Are you sure to delete?")) {
      const token = sessionStorage.getItem("jwt");
      fetch(link, {
        method: "DELETE",
        headers: { Authorization: token },
      })
        .then((res) => {
          toast.success("Patient deleted", {
            position: toast.POSITION.BOTTOM_LEFT,
          });
          this.fetchPatients();
        })
        .catch((err) => {
          toast.error("Error when deleting", {
            position: toast.POSITION.BOTTOM_LEFT,
          });
          console.error(err);
        });
    }
  };
```

Ο πηγαίος κώδικας της *συνάρτησης* add εμφανίζεται ως εξής, μετά τις τροποποιήσεις:

```
// Add new patient
  addPatient(patient) {
    const token = sessionStorage.getItem("jwt");
    fetch(SERVER_URL + "api/patients", {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
        Authorization: token,
      },
      body: JSON.stringify(patient),
    })
      .then((res) => this.fetchPatients())
      .catch((err) => console.error(err));
  }
```

Και ο πηγαίος κώδικας της συνάρτησης update μοιάζει ως εξής:

```
// Update patient
  updatePatient(patient, link) {
    const token = sessionStorage.getItem("jwt");
   fetch(link, {
     method: "PUT",
     headers: {
       "Content-Type": "application/json",
       Authorization: token,
     },
     body: JSON.stringify(patient),
   })
     .then((res) => {
       toast.success("Changes saved", {
         position: toast.POSITION.BOTTOM_LEFT,
       });
       this.fetchPatients();
     })
     .catch((err) => {
       console.log("err==================", err)
       toast.error("Error when saving", {
         position: toast.POSITION.BOTTOM_LEFT,
       });
     });
  }
```

Οπότε τώρα όλες οι λειτουργίες CRUD θα λειτουργούν, αφού συνδεθούμε στην εφαρμογή.

Ακολουθούν τα βήματα για την προσθήκη του react-toastify component στο αρχείο Login.js.

Για να είναι εναρμονιζμενη όλη η εφαρμογή με τα μηνύματα feedback μετα από μια ενέργεια μας, θα πρέπει όταν υπάρξει περίπτωση αποτυχίας του ελέγχου ταυτότητας κατά το Login να υπάρχει η δυνατότητα να εμφανίζεται στον τελικό χρήστη. Όποτε χρησιμοποιούμε το συστατικό react-toastify για την εμφάνιση του μηνύματος:

1. Προσθέτουμε την ακόλουθη εισαγωγή στο αρχείο Login.js:

```
import { ToastContainer, toast } from 'react-toastify';
import 'react-toastify/dist/ReactToastify.css';
```

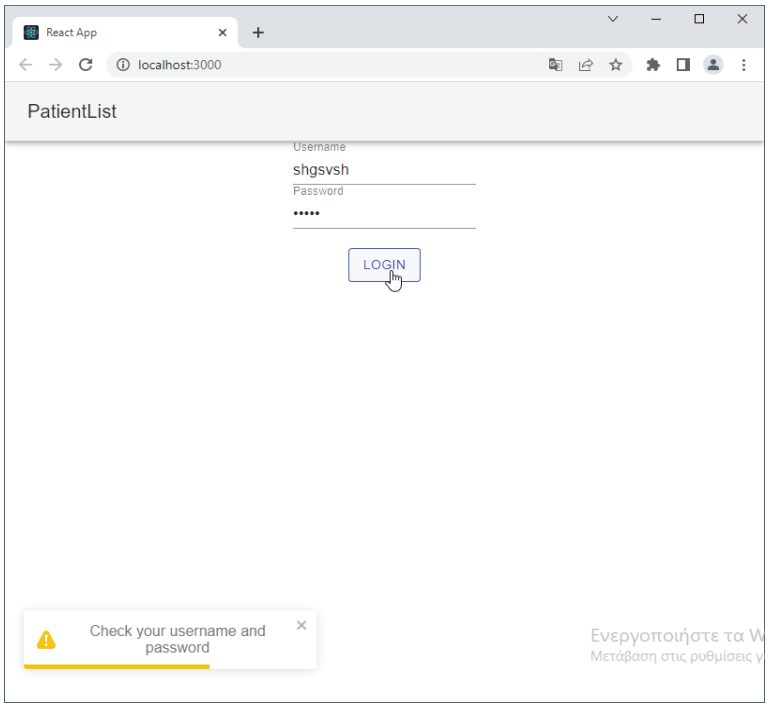
2. Προσθέτουμε το ToastContainer στη μέθοδο render():

```
<ToastContainer autoClose={1500} />
```

3. Και εμφανίζουμε ένα μήνυμα feedback αν ο έλεγχος ταυτότητας αποτύχει:

```
const login = () => {
  fetch(SERVER_URL + "login", {
    method: "POST",
    body: JSON.stringify(user),
  })
    .then((res) => {
      const jwtToken = res.headers.get("Authorization");
      if (jwtToken !== null) {
        sessionStorage.setItem("jwt", jwtToken);
        sessionStorage.setItem("username", user.username);
        setAuth(true);
      } else {
        toast.warn("Check your username and password", {
          position: toast.POSITION.BOTTOM_LEFT,
        });
      }
    })
    .catch((err) => console.error(err));
};
```

Αν τώρα συνδεθούμε με λάθος διαπιστευτήρια, θα δούμε το ακόλουθο toast message/μήνυμα ανάδρασης:



Εικόνα 58

4.8.3 Επιπλέον αλλαγές στο FrontEnd ώστε να υποστηρίζει i)Logout, ii)Διαβαθμιζόμενο Checkbox για τα επιβεβαιωμένα κρούσματα και iii)Στατιστικά

Η συγκεκριμένη λειτουργικότητα αφορά στο αρχείο App.js.

1.Τροποποιούμε το πρώτο import του αρχείου σε

```
import React, { useEffect, useState } from "react";
```

2.Προσθέτουμε το ακόλουθο:

```
const [jwtToken, setJwtToken] = useState("");
```

3.Και δημιουργούμε και την ακόλουθη συνάρτηση ώστε να μπορέσουμε να πάρουμε το JWT Token από το session storage:

```
useEffect(() => {
   if (sessionStorage.getItem("jwt")) {
     setJwtToken(sessionStorage.getItem("jwt"));
   }
 }, [sessionStorage.getItem("jwt")]);
```

4.Επισης δημιουργούμε μια συνάρτηση για το Logout/αποσύνδεση. Ουσιαστικά αφαιρούμε το jwt Token & το username από την αποθήκευση συνεδρίας /session storage:

```
const handleLogout = () => {
   sessionStorage.removeItem("jwt");
   sessionStorage.removeItem("username");
   window.location.reload();
 };
```

5.Η παραπάνω συνάρτηση θα κληθεί μέσα από μια condition μέσα στο αρχείο App.js στο return του δηλαδή:

```
return (
   <div className="App">
     <AppBar position="static" color="default">
       <Toolbar style={{ justifyContent: "space-between" }}>
         <Typography variant="h6" color="inherit">
           PatientList
         </Typography>
         {jwtToken ? (
           <Typography
             variant="h6"
             color="inherit"
             style={{ cursor: "pointer" }}
             onClick={handleLogout}
           >
             Logout
           </Typography>
```

```
        ) : null}
       </Toolbar>
     </AppBar>
     <Login />
   </div>
  );
}
```

[2]Στο αρχείο AddPatient.js προσθέτουμε τα

1) "@material-ui/core/Checkbox¨

```
import Checkbox from "@material-ui/core/Checkbox";
```

2) Την ακόλουθη σταθερά ώστε να έχουμε το username:

```
const username = sessionStorage.getItem("username");
```

3) Αν υπάρχει username = admin

```
{username === "admin" ? <span>
          Corona Affected? <Checkbox name="coronaPatient" onChange={(e) =>
setPatient({ ...patient, "coronaPatient": e.target.checked })} />
        </span> : null}
```

[3]Στο αρχείο Login.js προσθέτουμε τα

1)

```
import React, { useEffect, useState } from "react";
```

2)

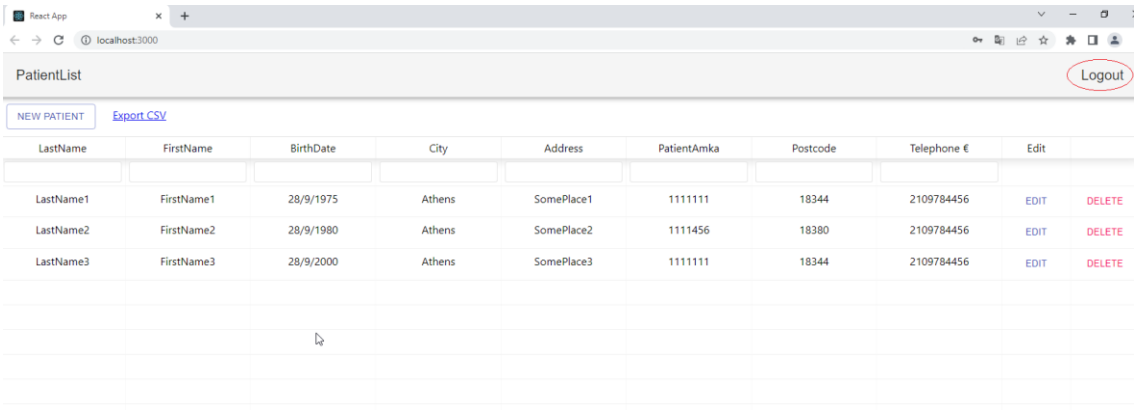
```
useEffect(() => {
   if (sessionStorage.getItem("jwt")) setAuth(true);
 }, []);
```

3)προσθέτουμε το window.location.reload() στον παρακάτω κώδικα

```
const login = () => {
   fetch(SERVER_URL + "login", {
     method: "POST",
     body: JSON.stringify(user),
   })
     .then((res) => {
       const jwtToken = res.headers.get("Authorization");
       if (jwtToken !== null) {
         sessionStorage.setItem("jwt", jwtToken);
         sessionStorage.setItem("username", user.username);
         setAuth(true);
         window.location.reload();
       } else {
```

```
        toast.warn("Check your username and password", {
          position: toast.POSITION.BOTTOM_LEFT,
        });
      }
    })
    .catch((err) => console.error(err));
};
```

Οπότε μετα τις παραπάνω αλλαγές όταν θα κάνουμε Login θα εμφανιστεί η ακόλουθη όψη, η οποία θα εμπεριέχει και ένα κουμπί πάνω δεξιά με όνομα Logout:



Εικόνα 59

το οποίο όταν το πατάμε μας κάνει logout.

[4]EditPatient.js
Εδώ κάνουμε αντίστοιχες ενέργειες με το AddPAtient.js

1)Προσθέτουμε το "@material-ui/core/Checkbox"

```
import Checkbox from "@material-ui/core/Checkbox";
```

2)

```
const username = sessionStorage.getItem("username");
```

3)

```
{username === "admin" ? <span>
        Corona Affected? <Checkbox name="coronaPatient" onChange={(e) =>
setPatient({ ...patient, "coronaPatient": e.target.checked })} checked={patient.coronaPatient} />
      </span> : null}
```

4) Προσθέτουμε τα επιπλέοντα αρχεία CoronaPatient.js & Stats.js:

1) CoronaPatient.js , είναι υπεύθυνο για το checkbox

```
import React, { useState } from "react";
import Checkbox from "@material-ui/core/Checkbox";
import { Button } from "@material-ui/core";
```

```
const CoronaPatient = (props) => {
  const [corona, setCorona] = useState(undefined);
//Παιρνουμε ολα τα δεδομενα του ασθενη
  const [patient, setPatient] = useState({
    lastName: props.patient.lastName,
    firstName: props.patient.firstName,
    birthDate: props.patient.birthDate,
    city: props.patient.city,
    address: props.patient.address,
    patientAmka: props.patient.patientAmka,
    postcode: props.patient.postcode,
    telephone: props.patient.telephone,
    coronaPatient: props.patient._original.coronaPatient,
  });

  const coronaPatientConfrimation = (e) => {
    setCorona(e.target.checked);

    setPatient({ ...patient, [e.target.name]: e.target.checked });
  };

  const handleSave = () => {
    props.updatePatient(patient, props.link);
    setCorona(undefined);
    window.location.reload();
  };

  return (
    <>
      {corona === undefined ? (
        <Checkbox
          name="coronaPatient"
          checked={patient.coronaPatient}
          onClick={coronaPatientConfrimation}
        />
      ) : (
        <>
          <Button onClick={handleSave}>Save</Button>
        </>
      )}
    </>
  );
};
export default CoronaPatient;
```

2)Stats.js: Component, ώστε να έχουμε την δυνατότητα στατιστικών στην εφαρμογή μας

```
import React, { useEffect, useState } from "react";
import Dialog from "@material-ui/core/Dialog";//H Modal φορμα του Material ui
import DialogActions from "@material-ui/core/DialogActions";
import DialogContent from "@material-ui/core/DialogContent";
import DialogTitle from "@material-ui/core/DialogTitle";
import Button from "@material-ui/core/Button";
import { Link } from "@material-ui/core";

const Stats = ({ patient }) => {
  const [open, setOpen] = useState(false);
  const [registrationNumber, setRegistrationNumber] = useState(0);
  const [coronaAffected, setCoronaAffected] = useState([]);

  useEffect(() => {
    if (patient.length > 0) {
      setRegistrationNumber(patient.length);
      let tempArr = [];
      patient.forEach((element) => {
        if (element.coronaPatient) {
          tempArr = [...tempArr, element];
        }
      });
      setCoronaAffected(tempArr);
    }
  }, [patient]);

  function percentage() {
    return coronaAffected.length / patient.length * 100;
  }

  // Open the modal form
  const handleClickOpen = () => {
    setOpen(true);
  };

  // Close the modal form
  const handleClose = () => {
    setOpen(false);
  };

  return (
    <div>
```

```
    <Link
      variant="outlined"
      color="primary"
      style={{ margin: 10 }}
      onClick={handleClickOpen}
      style={{ cursor: "pointer" }}
    >
      Stats Link
    </Link>
    <Dialog open={open} onClose={handleClose} fullWidth>
      <DialogTitle>Patients Stats</DialogTitle>
      <DialogContent>
        <div>
          <span style={{ display: "flex", alignItems: "center" }}>
            Number Of Registrations:   
            <b>{registrationNumber}</b>
          </span>
          <span style={{ display: "flex", alignItems: "center" }}>
            Basic Statistic of corona affected patients:   
            <b>{percentage()}%</b>
          </span>
        </div>
      </DialogContent>
      <DialogActions>
        <Button color="secondary" onClick={handleClose}>
          OK
        </Button>
      </DialogActions>
    </Dialog>
  </div>
 );
};

export default Stats;
```

5) Αλλαγές/προσθήκες στο patientList.js

1)

```
import Stats from "./Stats";
```

2)

```
import CoronaPatient from "./CoronaPatient";
```

3)

```
render() {
   const username = sessionStorage.getItem("username");
```

4)

```
{
      Header: "Telephone",
      accessor: "telephone",
    },
    {
      Header: "Corona",
      sortable: false,
      filterable: false,
      width: 100,
      accessor: "_links.self.href",
      Cell: ({ value, row }) => {
        return (
          <CoronaPatient
            patient={row}
            updatePatient={this.updatePatient}
            link={value}
          />
        );
      },
    },
```

5)

```
//columns για τον user
 const columnsUser = [
    {
      Header: "LastName",
      accessor: "lastName",
    },
    {
      Header: "FirstName",
      accessor: "firstName",
    },
    {
      Header: "BirthDate",
      accessor: "birthDate",
    },
    {
      Header: "City",
      accessor: "city",
    },
    {
      Header: "Address",
      accessor: "address",
    },
    {
      Header: "PatientAmka",
      accessor: "patientAmka",
    },
```

```
    {
      Header: "Postcode",
      accessor: "postcode",
    },
    {
      Header: "Telephone",
      accessor: "telephone",
    },
    {
      Header: "Edit",
      sortable: false,
      filterable: false,
      width: 100,
      accessor: "_links.self.href",
      Cell: ({ value, row }) => (
        <EditPatient
          patient={row}
          link={value}
          updatePatient={this.updatePatient}
          fetchPatients={this.fetchPatients}
        />
      ),
    },
    {
      Header: "Delete",
      sortable: false,
      filterable: false,
      width: 100,
      accessor: "_links.self.href",
      Cell: ({ value }) => (
        <Button
          size="small"
          color="secondary"
          onClick={() => {
            this.onDelClick(value);
          }}
        >
          Delete
        </Button>
      ),
    },
  ];
```

6)

```
{username === "admin" ? (
          <Grid item style={{ padding: 15 }}>
            <Stats patient={this.state.patients} />
          </Grid>
        ) : null}
      </Grid>
      <ReactTable
        data={this.state.patients}
        columns={username === "admin" ? columns : columnsUser}
        filterable={true}
      />
```

4.9 GitHub Repositories /αποθετήρια

4.9.1 Σύνδεσμος του GitHub για το Backend

https://github.com/kveldes/coronadatabase.git

4.9.2 Σύνδεσμος του GitHub για το Frontend

https://github.com/kveldes/patientfront.git

5.Συμπεράσματα – Περίληψη

Στην εργασία αυτήν, στόχος ήταν να παρουσιαστεί μια συγκεκριμένη υπηρεσία αντιμετώπισης πανδημίας μέσω ενός ευχάριστου Web Site. Προτού επιδείξουμε την εφαρμογή που υλοποιεί αυτήν την υπηρεσία, έγινε αναφορά στα εργαλεία, στα framework και στα libraries, τα οποία αποτελούν τα δομικά του στοιχεία. Στη συνέχεια, παρουσιάστηκε η ίδια η εφαρμογή, και η βήμα προς βήμα δημιουργία της. Επίσης, προστέθηκαν και εικόνες για να μπορέσει ο αναγνώστης να κατανοήσει όσο καλύτερα γίνεται τις λειτουργίες που του προσφέρει. Τέλος, έγινε μια αναφορά στα εργαλεία και τον κώδικα που χρησιμοποιήθηκε, καθώς και τις γλώσσες προγραμματισμού και διαχείρισης βάσης δεδομένων που επιλέχθηκαν.

Έχοντας ολοκληρώσει την μελέτη του αντικειμένου και την κατασκευή της εφαρμογής μας, είμαστε σε θέση να αξιολογήσουμε σε έναν επαρκή βαθμό την υπηρεσία μας. Μπορούμε να πούμε λοιπόν πως η καταγραφή κρουσμάτων της πανδημίας, αν δομηθεί σωστά μπορεί να αποτελέσει ένα πολύ σημαντικό εργαλείο στα χέρια των βαθμίδων της υγείας και γενικότερα της χώρας. Φυσικά, όπως κάθε εφαρμογή, επιδέχεται ακόμη πολλές βελτιώσεις και προτάσεις βελτιστοποίησής της.

6.Βιβλιογραφία

20. CORS [WWW Document], n.d. URL https://docs.spring.io/spring-security/site/docs/5.0.x/reference/html/cors.html (accessed 8.26.22).
Amazon Corretto Production-ready distribution of OpenJDK [WWW Document], n.d. URL https://aws.amazon.com/corretto/?filtered-posts.sort-by=item.additionalFields.createdDate&filtered-posts.sort-order=desc (accessed 8.25.22).
BCrypt (spring-security-docs 5.7.3 API) [WWW Document], n.d. URL https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/crypto/bcrypt/BCrypt.html (accessed 8.26.22).
Configuring Spring Boot for MariaDB - Spring Framework Guru [WWW Document], n.d. URL https://springframework.guru/configuring-spring-boot-for-mariadb/ (accessed 8.25.22).
Download IntelliJ IDEA: The Capable & Ergonomic Java IDE by JetBrains [WWW Document], n.d. URL https://www.jetbrains.com/idea/download/#section=windows (accessed 8.25.22).
GitHub - jwtk/jjwt: Java JWT: JSON Web Token for Java and Android [WWW Document], n.d. URL https://github.com/jwtk/jjwt (accessed 8.26.22).
H2 Database Engine [WWW Document], n.d. URL https://www.h2database.com/html/main.html (accessed 8.25.22).
Hands-On Full Stack Development with Spring Boot 2 and React: Build modern ... - Juha Hinkula - Google Books [WWW Document], n.d. URL https://books.google.gr/books?hl=en&lr=&id=aQmaDwAAQBAJ&oi=fnd&pg=PP1&dq=Hands-On+Full+Stack+Development+with+Spring+Boot+2+and+React+-+Juha+Hinkula&ots=39Blw5vcF9&sig=mEjx91VOIThVoQaflDUKPwoibd4&redir_esc=y#v=onepage&q=Hands-On Full Stack Development wit (accessed 8.27.22).
Hibernate (framework) - Wikipedia [WWW Document], n.d. URL https://en.wikipedia.org/wiki/Hibernate_(framework) (accessed 8.25.22).
Introduction to Java Config for Spring Security | Baeldung [WWW Document], n.d. URL https://www.baeldung.com/java-config-spring-security (accessed 8.26.22).
Introduction to the Java Persistence API - The Java EE 6 Tutorial [WWW Document], n.d. URL https://docs.oracle.com/javaee/6/tutorial/doc/bnbpz.html (accessed 8.25.22).
io.jsonwebtoken.Jwts.parser java code examples | Tabnine [WWW Document], n.d. URL https://www.tabnine.com/code/java/methods/io.jsonwebtoken.Jwts/parser (accessed 8.26.22).
java - Redirect in result of AbstractAuthenticationProcessingFilter - Stack Overflow [WWW Document], n.d. URL https://stackoverflow.com/questions/43773746/redirect-in-result-of-abstractauthenticationprocessingfilter (accessed 8.26.22).
JSON Web Token Introduction - jwt.io [WWW Document], n.d. URL https://jwt.io/introduction (accessed 8.25.22).
JWT tokens and security - working principles and use cases [WWW Document], n.d. URL https://www.vaadata.com/blog/jwt-tokens-and-security-working-principles-and-use-cases/ (accessed 8.25.22).
MariaDB Foundation - MariaDB.org [WWW Document], n.d. URL https://mariadb.org/ (accessed 8.25.22).
Multitenancy With Spring Data JPA | Baeldung [WWW Document], n.d. URL https://www.baeldung.com/multitenancy-with-spring-data-jpa/ (accessed 8.26.22).
Object–relational mapping - Wikipedia [WWW Document], n.d. URL https://en.wikipedia.org/wiki/Object–relational_mapping (accessed 8.25.22).
Postman API Platform | Sign Up for Free [WWW Document], n.d. URL https://www.postman.com/ (accessed 8.26.22).
React – A JavaScript library for building user interfaces [WWW Document], n.d. URL https://reactjs.org/ (accessed 8.25.22).
Representational state transfer - Wikipedia [WWW Document], n.d. URL https://en.wikipedia.org/wiki/Representational_state_transfer (accessed 8.25.22).
Spring Boot [WWW Document], n.d. URL https://spring.io/projects/spring-boot/#overview

(accessed 8.25.22).
Spring Boot CommandLineRunner - running beans with CommandLineRunner [WWW Document], n.d. URL https://zetcode.com/springboot/commandlinerunner/ (accessed 8.26.22).
Spring Boot Security + JWT Hello World Example | JavaInUse [WWW Document], n.d. URL https://www.javainuse.com/spring/boot-jwt (accessed 8.26.22).
Spring Data - CrudRepository save() Method | Baeldung [WWW Document], n.d. URL https://www.baeldung.com/spring-data-crud-repository-save (accessed 8.26.22).
Spring Data REST [WWW Document], n.d. URL https://spring.io/projects/spring-data-rest (accessed 8.26.22).
Spring Initializr [WWW Document], n.d. URL https://start.spring.io/ (accessed 8.25.22).
Spring Security: Database-backed UserDetailsService | Baeldung [WWW Document], n.d. URL https://www.baeldung.com/spring-security-authentication-with-a-database (accessed 8.26.22).
Spring Security [WWW Document], n.d. URL https://spring.io/projects/spring-security (accessed 8.25.22).
Spring Security custom authentication filter using Java Config - Stack Overflow [WWW Document], n.d. URL https://stackoverflow.com/questions/27507862/spring-security-custom-authentication-filter-using-java-config (accessed 8.26.22).
Tutorial | Building REST services with Spring [WWW Document], n.d. URL https://spring.io/guides/tutorials/rest/ (accessed 8.26.22).