HEALTHQR

Object Orientated Programming

2nd year 1st semester

Benő Ákos

Contents

[1. Introduction 3](#_Toc156658559)

[2. Setup & Configuration 3](#_Toc156658560)

[3. Architecture 3](#_Toc156658561)

[4. Technologies used 4](#_Toc156658562)

[5. Database 4](#_Toc156658563)

[6. Presenting HealthQR 5](#_Toc156658564)

[5.1. Roles 5](#_Toc156658565)

[5.2. Authentication 5](#_Toc156658566)

[5.3. Navigation bar 6](#_Toc156658567)

[5.4. Registration 6](#_Toc156658568)

[5.5. Patients 8](#_Toc156658569)

[5.6. Treatments and Treatment Schedules 8](#_Toc156658570)

[6. Future improvements 9](#_Toc156658571)

[7. Project requirements 9](#_Toc156658572)

[8. Conclusion 10](#_Toc156658573)

# Introduction

HealthQR aims to provide a comprehensive hospital management system, with a clean and user-friendly interface. Its target is to make the recovering of a patient transparent and traceable. This project can manage complex process, like treatments, and one day maybe save lives.

The project is a Java-based web application which uses PostgreSQL database and the Spring Framework. Some of the applications features are: 5 types of roles named Admin, Doctor, Nurse, Patient or Guest; Treatments with their own treatment logs named treatment schedules or authentication and authorization processes. The project follows an enhanced MVC design pattern.

# Setup & Configuration

To set up the project, firstly Java 17 is needed. The project was developed using Amazon Corretto version 17.0.9 and IntelliJ IDE. The dependencies can be found in the ‘pom.xml’ file of the project, you can configure any additional dependency or in that file or using Maven.

The main application configuration is located in the ‘application.propeties’. Database connection settings can be customized there, as well as logging configurations.

# Architecture

HealthQR follows a layered architecture with 5 main layers. These layers are the: **Controllers, Services, Repositories, Models** and **Views.** The project also has DTOs and their mappers, which weren’t essential for every model, but it was a good practice for future projects.

The **model** are the most crucial elements of the project, and they are just like the tables in the database. The relationship between them is also specified in their model class, which can be many to many, one to many, many to one or one to one. All of these relationships are present in the project as well.

We also have for most of the models their **DTOs** which make easier the usage of the entities in the program. Here are necessary the **mapper** classes, they make possible the transition from the model to the model’s DTO and vice versa.

For most of the models there are **services**, each one with its own interface. They have all the CRUD operations for the entities, perform data validation and interact with the repositories.

In the **repositories** we cannot find too much code. They interact with the database using JPA (Java Persistence API), which is a very powerful tool. Only a few complicated queries are written like searching based on id or first name or last name of a patient. The common CRUD operations are inherited.

**Controllers** are the most central units of the project, they handle the communication between the frontend and the application’s backend. There we can find GET and POST methods, most of them satisfying the CRUD operations for the entities.

Last but not least the **views** form the interface visible for the user. We can find a layout which is the base of every other view (here are exceptions as well). The language is html, CSS, in the register.html there is also a little JavaScript, and all of the views uses Thymeleaf template engine.

# Technologies used

HealthQR uses Java 17 for its backend development. Spring 6 provides a powerful framework, and Thymeleaf is used for server-side templating. JPA helps the interaction with database, providing a wide range of simple queries.

The project was generated with the help of the page <https://start.spring.io/> as a Spring Boot application, and Maven is used for managing dependencies.

# Database

For the database side the project uses PostgreSQL database. While development DBeaver database manager and IntelliJ’s internal database manager was used simultaneously. Because of the @Data annotations in the model classes all of the tables are autogenerated if the database configuration is correct, but the default roles in the role table are hard coded, so upon regeneration the roles should be introduced again. The database architecture is quite simple, made of 9 tables, from which the table ‘users’ is the central, for obvious reasons. Inheritance and different kind of relations are present.

A screenshot of a computer

Description automatically generated

# Presenting HealthQR

## Roles

In the HeathQR there are 5 type of predefined users: ‘ADMIN’ with id 0, ‘DOCTOR’ with id 1, ‘NURSE’ with id 2, ‘PATIENT’ with id 3 and ‘GUEST’ with id 4. This 5 types can be divided into 3 categories: for roles ‘GUEST’ until now no functionality is implemented, so it is added only for future developments. Roles ‘DOCTOR’ and ‘NURSE’ have the same authority level through the application, because there are not yet implemented the functions, which would make the difference significant between these two types of users, so in the future they will be referred only as “staff”. In addition to these we have the roles ‘ADMIN’ and ‘PATIENT’. The admins have the highest level of authority in the application, they can access any page, and make any operation. However patients have the lowest level of authority, and they can access only the information about their treatment.

## Authentication

HealthQR is intended to be a hospital management system. Therefore, registration is not available, only a user with an ‘ADMIN’ role can register new users, and the admin can choose any role for them. Through the Spring framework security configuration, it is set that only the ‘login’, ‘js’ and ‘css’ files are reachable without authentication. The logic behind this, is that only the manager of a hospital can gives newly registered accounts for the newly hired staff. An ‘ADMIN’ user can register a new user, with role ‘DOCTOR’, ‘NURSE’ or ‘PATIENT’ but a doctor or a nurse can register only a new user with role ‘PATIENT’. The Spring security by default uses username and password as credentials but in the application, these are changed to email and password, for logical reasons. For a user with role ‘PATIENT’ most of the navbar is not visible, but there are other security protocols. For example if a patient tries to reach an option for which he/her does not have the required role, like registering a new account the following page is loaded.

A screen shot of a computer

Description automatically generated

## Navigation bar

The content of the navigation bar depends on the role of the user. After authentication in the dropdown menu named ‘Welcome’ we can find a Logout button for every user. If the user is admin or staff member, we also have a button which navigates to the registering page, in the ‘Treatment’ dropdown there is a button which navigates to the treatments listing page, and another which navigates to the treatment schedules listing page. We also can access from here the patients listing page. In the ‘Welcome’ dropdown there is also a ‘Create Treatment’ button.

On the right hand side the Hospital management text can be seen, and in the middle the authenticated user’s email address and role is displayed.



## Registration

An admin can register new staff members accessing the register dropdown list. On the registration page after specifying all the personal information of the new staff member, its role can be selected from a dropdown list. Some of the data field has costume validation, like the first name should not be empty, or the email address must be a valid one. If the registration button is pressed and the data is not valid an error message is displayed under the data field. This mechanism is present also during update.A screenshot of a computer

Description automatically generated

Date of birth can be selected from a calendar. For a staff member, form the ‘Roles’ dropdown list only the Patient option is available, so they can register only a new patient.

A screenshot of a computer

Description automatically generated

If the option is selected using javascipt new value field appear without reloading the page, and these fields are specific for a patient.

A screenshot of a computer

Description automatically generated

After the registration proccess is done, the authenticated user is not logged out, so the given admin or staff member can continue his/her work in the application.

## Patients

The most used site is the patients-list.html under the Http address “/patients”. Here an admin or a staff member can see all the patients, and their administrative information. They can view there details, accessing the details page, where is a delete button also, or they can update their information. From here new treatment can be added also.

A screenshot of a computer

Description automatically generated

## Treatments and Treatment Schedules

The “/treatments” page is very similar to the “/patients” page but a staff member there can see only the treatments added by him/her, or an admin can see all the treatments. On a treatment card only the most essential information is displayed, like its id, creation date, status and patient id. A screenshot of a computer screen

Description automatically generated

A treatment’s details page is accessible pressing the view button. In the details page have options for deleting the treatment or adding a new treatment schedule to it.

Treatment Schedules can only be added at a treatment’s details page, because they have to be linked together, and every treatment schedule must have treatment id. Treatment Schedules can also be listed, edited or deleted.

A screenshot of a computer

Description automatically generated

# Future improvements

There are a lot of bugs in the application. Some of them is easy to correct, but others are difficult. Currently the project is functional, but there is a lot to be added. The difference between the roles nurse and doctor is to most important one, this should be implemented first. The most convenient solution would be if only a doctor could make new treatment schedule, and a nurse could only modify its status, with updating it, but only those which has as nurse id, the users person id.

Also, a lot of data filtering could be added for the transparency of the page, different types of search bars, and more complex models.

Currently there is no solution for deleting or editing a user, this should be added also.

The things were not added only because of the lack of time.

# Project requirements

I tried to fulfil all the project requirements. The project compiles and follows an enhanced MVC design pattern. Inheritance is present at multiple places, but the most remarkable is between the class person and class patient. Enumerations are used when selecting the blood types, and they are displayed in the browser with the help of the enumeration property. The project is a web application and has 14 views and one layout, all of them are functional. There are 5 types of roles, but 3 of them has remarkable differences: ADMIN, DOCTOR/NURSE and PATIENT. Exception handling is used in the security package’s SecurityConfing class, which is the configuration class for the Spring Security framework. The project is connecting to the database through the application.properties file, and CRUD operations are present on entities treatment, treatment schedule, and patient.

# Conclusion

The project had a lot of “firsts”. It way my first Java standalone web application, it was my first-time using Spring, JPA and Thymeleaf. When I started, I was way more ambitious, and I thought I will be able to implement much more functionalities. Now that I met the deadline, I think I was able to make a functional project, which has its mistakes, but it was an excellent challenge, from which I learned a lot.