Medical Appointment Scheduling Application

Project documentation

Student: Keresztes Beáta

**Group: 30432**

**Contents**

[I Project specification](#_heading=h.gjdgxs) **3**

[1.1 Domain Model Diagram](#_heading=h.30j0zll) [3](#_heading=h.30j0zll)

[II Use-Case model](#_heading=h.1fob9te) **3**

[2.1 Users and stakeholders](#_heading=h.f2irg1azp7rm) [4](#_heading=h.f2irg1azp7rm)

[2.2 Use-Case identification](#_heading=h.3znysh7) [4](#_heading=h.3znysh7)

[2.3 UML Use-Case diagram](#_heading=h.2et92p0) [4](#_heading=h.2et92p0)

[III Architectural design](#_heading=h.tyjcwt) **5**

[3.1 Conceptual architecture](#_heading=h.3dy6vkm) [5](#_heading=h.3dy6vkm)

[3.2 Package diagram](#_heading=h.1t3h5sf) [5](#_heading=h.1t3h5sf)

[3.3 Class diagram](#_heading=h.4d34og8) [5](#_heading=h.4d34og8)

[3.4 Database (E-R/Data model) diagram](#_heading=h.2s8eyo1) [6](#_heading=h.2s8eyo1)

[3.5 Sequence diagram](#_heading=h.17dp8vu) [6](#_heading=h.17dp8vu)

[3.6 Activity diagram](#_heading=h.3rdcrjn) [6](#_heading=h.3rdcrjn)

[IV Supplementary specifications](#_heading=h.26in1rg) **6**

[4.1 Non-functional requirements](#_heading=h.lnxbz9) [6](#_heading=h.lnxbz9)

[4.2 Design constraints](#_heading=h.35nkun2) [6](#_heading=h.35nkun2)

[V Testing](#_heading=h.1ksv4uv) **7**

[5.1 Testing methods/frameworks](#_heading=h.44sinio) [7](#_heading=h.44sinio)

[5.2 Future improvements](#_heading=h.2jxsxqh) [7](#_heading=h.2jxsxqh)

[VI Bibliography](#_heading=h.z337ya) **7**

# I Project specification

This web application aims to provide an easy way to schedule and manage medical appointments in an online environment.

One of its main purposes is to keep evidence of and efficiently manage the scheduled appointments at a given clinic. It stores information about each patient and their appointments in an organized way, so that anyone can quickly look up the details of an appointment or patient profile. It also allows the medical service providers or doctors to view and edit their work schedule, add breaks or view the appointments assigned to them, as well as the patient’s profile. Nevertheless, this app also provides flexibility for the users, as they can schedule appointments at any time of the day, even outside the working hours of the clinic.

It can be used by the patients and medical staff of a clinic, in order to efficiently manage their patients’ appointments, save time and maintain an organized schedule.

There are in total 3 types of users with different roles:

* **Patients** can request an appointment, confirm the appointment a day before or cancel the appointment at any time.
* The **receptionist** creates the appointments upon the request of a patient with respect to the requested service and the availability of the doctors.
* The **doctors** working at the clinic can view their work schedule and the details of each planned appointment. They are also able to update their own schedule, by adding breaks or modifying their working intervals, in order to indicate when they are available.

## 1.1 Domain Model Diagram

Diagram, schematic

Description automatically generated

# II Use-Case model

The main functionalities provided by the application include scheduling new appointments for patients, keep track of the planned appointments in an organized manner to enhance the efficiency of the services offered by the clinic.

## 2.1 Users and stakeholders

The application supports 3 types of users: Patient, Doctor and Receptionist, each with their own set of functionalities.

All users can:

* Register
* Log in
* Log out

Patient can:

* Request an appointment
* Confirm appointment
* Cancel appointment
* View all their appointments
* View prescriptions

Receptionist:

* Create appointment
* Delete appointment
* Update appointment status
* Check-in a patient
* View appointments of a patient
* View schedule of a doctor

Doctor:

* Add prescription for patient
* View their work schedule
* View appointment details and patient profile

The Manager of the Clinic, who supervises the employees, both the receptionists and the doctors, represents a major stakeholder, as well as the developers of the application, who implemented and designed the given software.

The medical clinic which provides their patients the opportunity of scheduling an appointment online, could benefit from this technological advancement, as it could attract more patients and it also increases the productivity of the medical staff.

## 2.2 Use-Case identification

**Use case name**: Request an appointment

**Level**: User-goal

**Main actor**: Patient

**Main success scenario**:

* Patient is logged in into the application.
* Selects “Request a new appointment” from the menu.
* Selects the type of medical service.
* Patient sends the request.
* A message appears notifying the patient that the appointment request was forwarded.
* The appointment will appear in the list of appointments of the patient if it could be successfully created by the receptionist of the clinic.

**Extension**: -

**Use case name**: Create a new appointment

**Level**: User-goal

**Main actor**: Receptionist

**Main success scenario**:

* Receptionist is logged into the application.
* Views the list of appointment requests and selects one of them.
* Based on the type of the requested medical service, he/she finds a doctor and searches for an empty time slot for a new appointment.
* If an empty time slot was found, he/she marks the appointment date.
* After assigning a doctor and a specific time interval, he/she saves the appointment.
* After the appointment was created, it will be removed from the list of unhandled requests.

**Extension**: -

**Use case name:** Cancel appointment

**Level**: User-goal

**Main actor**: Patient

**Main success scenario:**

* Patient is logged in into the application.
* Selects “View all my appointments” from the menu.
* Selects an item from the list and chooses the option “Cancel appointment”
* A message appears notifying the user that the appointment has been canceled.

**Extension**:

* If the patient has no scheduled appointments, the “Cancel appointment” button will be disabled for past appointments.

**Use case name:** Register

**Level**: User-goal

**Main actor**: Patient

**Main success scenario:**

* Select the “Register” option from the menu.
* Enter a username and a password.
* Select the type of account to be created.
* Click on “Next” to proceed.
* Enter the full name, birth date and contact information, such as email or telephone number. Specify any allergies (optional).
* Click on “Create account” to save the account.
* A notification message appears indicating that the account has been created successfully.
* The user is redirected to the login page.

**Extension**:

Duplicate username:

* The chosen username is already taken, in which case the registration fails and an error message appears notifying the user that the account could not be created, because the username is already taken.
* After the user closes the pop-up, he/she is redirected to the register page, with all the previous data saved except the field which contained invalid information.

Invalid contact information:

* The user attempts to input an email or phone number having an invalid format.
* A message appears notifying the user that the account could not be created because some of the contact information is invalid.
* After the user closes the pop-up, he/she is redirected to the register page, with all the previous data saved except the field which contained invalid information.

**Use case name:** Check in a patient

**Level**: User-goal

**Main actor**: Receptionist

**Main success scenario:**

* Receptionist is logged in into the application.
* Searches for the appointment of the patient by the patient’s name and the current date and sets the status of the appointment to “Completed”.

**Extension**:

* If the patient previously confirmed the appointment but didn’t show up the next day, the status of the appointment will be set to “Missed”.

**Use case name:** View work schedule

**Level**: User-goal

**Main actor**: Doctor

**Main success scenario:**

* Doctor is logged in into the application.
* Selects “View my work schedule” from the menu, which opens a calendar view.
* He/She can view the appointments for each day.
* The doctor clicks on any square from the calendar, representing a given day and the hourly schedule appears, indicating where an appointment has been added.
* By clicking on any of the appointments, he/she can view the details of the appointment as well as the profile of the patient.

**Extension**: -

## 2.3 UML Use-Case diagram

Diagram

Description automatically generated

# III Architectural design

The architecture defines the blue-print or skeleton of the project, and it helps to identify the high-level components of the system.

## 3.1 Conceptual architecture

The Medical Appointment Scheduling system is a Web Application, developed using the Spring Framework.

### Layered

The application is built on top of a layered architecture, separating the components into 3 layers:

* Presentation layer: represents the user interface of the application and presents its main features to the user.
* Business logic layer: contains the business logic that determines the application’s core functionalities.
* Data access layer: it is responsible for interacting with the database to save & update the application data.

The main purpose of the layered architecture is to achieve low-coupling and to reduce the dependency between components.

The application fits nicely the layered model, as it contains the web-pages displayed for the end users in the view layer, the business logic and the core functionalities are provided by the controller and service classes in the business logic layer, while the access to the database through repository manager classes is achieved in the data access layer. Each layer representing a different package or component, containing the relevant classes.

The application’s relevant data is stored in a MySql database, and the communication between the database server and the application is achieved through the Spring Data JPA, which provides Repository classes for accessing and managing data in the Data access layer.

### Model-View-Controller

Another type of architecture used in designing the project and organizing the logic in the user interface, is the Model-View-Controller architecture, based on the Spring MVC Framework, which ensures low-coupling between the components and managing the web application page flow.

* Controller: the mediator between the Model and the View, who has the double role of accessing data in the model and displaying it on the view as well as handling the UI events and updating the model.
* View: present the user interface based on Thymeleaf templates
* Model: the entities which hold the relevant data for the application

The layered architectural model includes the MVC architectural pattern in the following way:

Diagram

Description automatically generated

The MVC pattern fits the application because it reduces the overload of managing the user interface directly from the business logic layer, and it provides a mechanism for updating the user interface for any change in the model. For example, upon creating/editing an appointment either by the Receptionist or the Patients it is important that the view for both users is updated in real-time, so that data across users will remain consistent.

## 3.2 Package diagram

Diagram

Description automatically generated

## 3.3 Class diagram

Model package contains the entities which define the domain model, their class diagrams is the following:

Diagram

Description automatically generated

Diagram

Description automatically generated

Package users which contains the 3 types of users and the related data grouped into different entities (profiles), having the common features/attributes included in a common superclass, UserProfile. The credentials and login related details are stored in the Account entity.

The methods are not shown in the class diagrams because they are only getter/setter methods. The Domain Model entities, which represent real entities in the database, should not contain any business logic, such as functionalities or activities. The allowed functionalities are declared and implemented separately, so that the functionality will not be strongly tied to the underlying representation of the data.

Diagram

Description automatically generated

## 3.4 Database (E-R/Data model) diagram

Graphical user interface

Description automatically generated

## 3.5 Sequence diagram

* Register new user:

Table

Description automatically generated

* Login user:

Diagram

Description automatically generated

* Patient requests an appointment:

## Diagram Description automatically generated

* Receptionist accepts appointment request and updates the appointment:

Diagram

Description automatically generated with low confidence

* Doctor creates a prescription for a patient:

Application

Description automatically generated with low confidence

## 3.6 Activity diagram

* Activity diagram for a patient:

# Diagram Description automatically generated

* Activity diagram for a doctor:

Diagram

Description automatically generated

# IV Supplementary specifications

*< Se va scrie o mica introducere./>*

## 4.1 Non-functional requirements

* **Availability**: the users should be able to request an appointment any time, even outside the working hours of the clinic. The appointment request will be handled the next day by a receptionist.
* **Security**: the passwords associated to each account which allow the users to log in to the application are encoded before storing them in the database. For example, by using the PasswordEncoder of Spring Security.
* **Scalability:** the application can be scaled by filling the database with more data, registering a large number of patients, storing a large number of appointments. The system should be able to perform as expected when multiple users are connected to it at the same time. The application can be used not only by clinics, but any organization or institute which needs to manage their clients’ appointments in an organized and efficient way.
* **Performance:** the response time to the user actions depends on the number of simultaneously connected users. Because access to the database is slow and two users cannot modify the stored data at the same time to avoid inconsistencies, the application can only support a limited number of connected users. However, the wait time for a certain action should not exceed 2s, otherwise it would negatively affect the user’s experience.
* **Usability:** the application has a simple, easy to understand interface, so that even the non-experienced users can use it without difficulties.

## 4.2 Design constraints

The application is built on top of the **Spring Boot Framework,** more specifically the Spring MVC and Spring Data JPA, which provide a mechanism for managing efficiently the user interface (view) as well as providing an interface, or API for accessing the application’s data stored in the database in an easy way.

For storing the application’s data the **MySql database server** is used. It allows to create relational databases, entities and specify the relationship between them, the type of associations. The communication with the database from within the application’s logic is achieved through the **JPA** based repositories, in the data access layer.

The front end is developed using **React** and HTML5.

Spring Boot automatically starts up an embedded **Tomcat** server when the application starts, representing a container in which the application can be deployed and run, and which contains the binaries for the server. Once the server is up and running, it should be able to serve client requests.

Security constraints such as protecting the web content and users’ data is achieved through the help of the **Spring Security** framework, which provides authentication, authorization, and protection against attacks.

Constraints related to the application’s **scope**, which represent the core functionalities the system is expected to perform, should not be changed during the implementation.

The **Functional requirements** also have a high impact on the application as they define the main functionalities and services that the system must provide. Adding more requirements leads to a more complex design, as they could introduce major changes in the current design.

**Commercial constraints** which in this case refer to the limited time allocated for designing and implementing the application. Due to the short time, only the core functionalities will be included in the application, without adding additional features.

# V Testing

*< Se va discuta la laborator./>*

## 5.1 Testing methods/frameworks

## 5.2 Future improvements

# VI Bibliography