Project Deliverable 1 - Documentation

Healthcare Management System

**Software Design**

Faculty of Automation and Computer Science

2022-2023



Submitted by:

**Noémi Veres**

group – 30433

Contents

[Introduction 3](#_Toc130805983)

[Technologies 4](#_Toc130805984)

[Software Architecture 5](#_Toc130805985)

[Functional requirements 6](#_Toc130805986)

[Non-functional requirements 7](#_Toc130805987)

[Diagrams 8](#_Toc130805988)

[Class Diagram 8](#_Toc130805989)

[Package Diagram 10](#_Toc130805990)

[Database Diagram 11](#_Toc130805991)

[Use Case Diagrams 12](#_Toc130805992)

[Sequence Diagrams 14](#_Toc130805993)

Table of Figures

[Figure 1: Use Case Diagram 6](#_Toc130148390)

[Figure 2: Package Diagram 10](#_Toc130148391)

[Figure 3: Class Diagram 11](#_Toc130148392)

[Figure 4: Database Diagram 13](#_Toc130148393)

Introduction

A healthcare management system is a type of software that enables healthcare providers to manage and coordinate various aspects of patient care. From scheduling appointments and managing patient records to processing billing and insurance claims, healthcare management systems help healthcare organizations operate more efficiently and effectively.

With the increasing demand for quality healthcare services and the growing complexity of healthcare operations, healthcare management systems have become essential tools for healthcare providers of all sizes. The aim of this project is to design a flexible, scalable, and user-friendly system, which would be ideal for hospitals, clinics, and other healthcare organizations looking to improve their operations and provide better patient care.

The sub-objectives are the following:

* Analyze the problem and identify requirements
* Design the system for asking and answering questions
* Implement the system for asking and answering questions
* Test the system for asking and answering questions

This documentation will guide you through the features and functionalities of my healthcare management system.

Our healthcare management system serves a variety of users, including healthcare providers, administrators, and patients. Healthcare providers, such as doctors, nurses, and other medical professionals, are the primary users of the system. They use the system to manage patient records, schedule appointments, order tests and medications and check blood test results. Administrators, such as office managers, also use the healthcare management system to manage administrative tasks, such as managing appointments, and tracking patient information. Patients can also benefit from using the healthcare management system, as it allows them to schedule appointments, view their medical records, and communicate with their healthcare providers.

Some of the key features of my healthcare management system include:

* Patient records management: the system maintains electronic records of each patient’s medical history, diagnoses, medications, and other important information.
* Appointment scheduling: the system enables users to schedule patient appointments and track patient attendance.
* Billing: the system manages billing, and tracks payments and balances.
* Communication tools: the system provides secure communication channels for healthcare providers to communicate with patients.
* Reporting: the system generates reports on patient care
* Inventory management: the system helps healthcare providers manage medical supplies and equipment.

Technologies

The core programming language used to write this application is Java, used together with the Spring Framework, providing features such as inversion of control and support for web application development. The IDE I am using for this assignment is IntelliJ IDEA, which allows me to write code, debug and test the application. A database management system was also needed, in order to store application data. My choice was MySQL. As a build tool for this Java Spring application, I want to use Maven. The web server used to deploy and run the application is Apache Tomcat in this case. For testing my code, I can use the JUnit testing framework, which enables me to write and execute unit tests.

REST (Representational State Transfer) is a software architectural style used to create web services that are lightweight, scalable, and easy to maintain. RESTful web services are in their turn a type of web service that adheres to the REST architectural principles. They use HTTP protocols to communicate between the client and server, and the server exposes a set of resources or endpoints that the client can interact with using standard HTTP methods such as GET, POST, PUT, and DELETE, corresponding to the CRUD operations: Create, Read, Update, Delete.

Overall, the combination of these technologies and tools is aimed to allow the development of robust and scalable Java Spring applications that are both efficient and easy to maintain.

As a front-end web application framework, I chose Angular, which is a popular and powerful JavaScript framework that is well-suited for developing robust and scalable web applications. There are several reasons why I chose Angular for this project. Angular's two-way data binding allows for seamless synchronization between the model and the view, making it easier to develop complex user interfaces and provide real-time updates. Its dependency injection system makes it easier to manage application components and promotes modular code design, allowing for greater flexibility and reusability. The component-based architecture encourages the development of modular and reusable code components, which makes it easier to maintain and update the application over time.

Software Architecture

**Layered Architecture** is a popular architectural pattern that is often used in web application development. It separates the application into different layers based on the responsibilities. The main different layers for this application are: Controller, Service, Model, DAO and Repository.

**Controller Layer**: This layer is responsible for receiving user input from the UI, processing it, and sending it to the appropriate Service Layer. It also handles HTTP requests and responses, routing, and authentication. The Controller Layer is the entry point for the application and is responsible for controlling the flow of data between the UI and the Service Layer.

**Service Layer**: This layer contains the business logic of the application. It is responsible for processing and manipulating data. The Service Layer is where the main application logic resides, and it acts as a mediator between the Controller Layer and the Model Layer. It receives requests from the Controller layer, processes them, and interacts with the DAO and Repository layers to perform database operations.

**Model Layer**: This layer represents the application's data and is responsible for the creation and manipulation of business objects. The Model Layer contains classes and data structures that represent entities in the application, such as users, questions, and answers in our case.

**DAO Layer**: It is the short for Data Access Object and is responsible for managing the interactions between the application's business logic layer and the underlying database. The DAO layer acts as an intermediary between the application's code and the database, it provides an abstraction over the Repository layer and interacts with it to retrieve and manipulate data. It receives requests from the Service layer and performs the necessary database operations.

**Repository Layer**: This layer is responsible for persisting data in the database. It provides an abstraction layer between the Model Layer and the database, and it is responsible for querying, inserting, updating, and deleting data. It communicates with the database to perform CRUD operations (Create, Read, Update, Delete). It encapsulates the low-level details of data access and provides a simplified interface for the DAO layer to interact with.

In a typical Layered Architecture, each layer is independent and can be developed and tested separately. It also enables developers to change one layer without affecting the other layers.

Therefore, Layered Architecture is a widely used pattern that is particularly useful for building large and complex web applications. It provides a clear separation of concerns, which makes the application more modular, flexible, and easier to maintain.

Functional requirements

Functional requirements describe what a software system should do to meet the needs of its users. For my healthcare management system, the functional requirements include:

* Patient management: the system should allow healthcare providers to manage patient records, including demographic information, medical history, medications, allergies, and other relevant data.
* Appointment scheduling: the system should allow users to schedule appointments, including sending reminders and notifications to patients.
* Billing and payment processing: the system should provide functionality to process patient billing and payments.
* Electronic medical records (EMR): the system should provide a centralized repository for storing and managing patient medical records, including test results, diagnoses, treatment plans, and other relevant data.
* Telemedicine: The system should provide functionality for remote consultations via chat.
* Health monitoring: The system should allow patients to track and monitor their health, including vital signs, symptoms, and other relevant data, and share this information with healthcare providers as needed.

Non-functional requirements

* Users must be logged in to perform any actions.
* User passwords must be stored in the database encrypted.
* The system should be fast and responsive.
* The system should be reliable and available at all times.
* The system should be secure, with appropriate measures in place to prevent unauthorized access and other security issues.
* The system should be easy to use and intuitive for both regular and administrator users.
* The system should be scalable, with the ability to handle a large number of users and data growth.

Diagrams

Diagrams help to visually represent complex information in a clear and concise manner, making it easier for people to understand our system. They can also help to identify patterns, relationships, and trends. This is why they are used for software applications, to communicate ideas, processes, and concepts more effectively.

Class Diagram

A class diagram is a type of UML diagram, used to visualize the structure of a system. The package diagram, as its name says, focuses on the relationship between packages, but the class diagram provides an overview of the classes and interfaces and their relationships.

In a class diagram, each class is represented by a rectangular box, to which we can add its attributes and methods. Attributes describe the properties of an object, while methods describe the behaviours or actions that can be performed by the object.

Classes are connected to each other by lines and arrows that represent relationships between them. Different kinds of relationships exist, like inheritance, association, aggregation and composition.

The main entities of my system are:

* **Patient**: represents an individual who is receiving medical care. It includes personal information such as name, age, gender, contact details and medical history.
* **HealthcareProvider**: this entity represents a person who is responsible for diagnosing and treating patients. It includes information such as name, contact details, medical qualifications, and specializations.
* **Administrator**: represents a healthcare manager who can make appointments for the patients. It includes information such as name, contact details.

For the entities above, we can consider a generalized **User** entity, which would include the name, contact details of a person, as well as a password for accessing the online services.

* **MedicalFacility**: represents a hospital, clinic, or other medical facility where patients receive medical care. It may include information such as name, location, contact details, and services provided.
* **MedicalEquipment**: represents the medical devices and equipment used to diagnose and treat patients. It may include information such as the name of the equipment, manufacturer, and maintenance schedule.
* **MedicalRecords**: represents the medical history of a patient, including their diagnoses, treatments, and test results. It may include information such as the date of the visit, the name of the healthcare provider, and the type of service provided.
* **Appointment**: represents an appointment for a patient and includes information such as creation date, the patient for whom the appointment was made, the medical staff, specialization, scheduled time and duration.

All of the entities before should contain constructors, a no-argument version and a parameterized one. They should also contain getters and setter for their private attributes. These entities are represesented in Figure 1.

Graphical user interface, application, Teams

Description automatically generated

Figure 1 Class Diagram

* The different **Repository interfaces** are similar in their structure, they will extend the CrudRepository class. CrudRepository is an interface provided by the Spring Framework that allows the implementation of basic CRUD (Create, Read, Update, Delete) operations on entities. It provides several methods such save(), used to both create and update an object, findById(), findAll(), delete(), and deleteById(), which can be used to perform these common database operations.
* The **Service classes** use the @Autowired annotation for repository injection. In this way, the service class can call methods on the repository object, without having to create a new instance of that repository. Service classes contain methods that implement business logic for the application and interact with repositories or other to perform database operations.
* The **Controller classes** are responsible for handling incoming requests from clients. The controller typically acts as an intermediary between the user interface and the service layer, receiving input from the user and passing it to the appropriate service class to perform the necessary business logic. In our case, a corresponding service is injected into each controller class. Once the service has completed its processing, the controller then returns the appropriate response to the client.
* **DTO classes** should contain only the necessary data needed to transfer between layers, and it should be optimized for efficient transfer. To establish a relationship between entities and DTOs, I want to define a separate DTO class for each entity, including only the relevant data needed for a specific operation.

Package Diagram

Packages are used to organize the code and to provide a general view of the system's architecture. In the packages, I grouped together the classes which are correlated.

A package diagram is a UML diagram. It is representative for the system, as it shows the organization and the structure of the application. In such a diagram, the dependencies between packages should be indicated.



Figure 2 Package Diagram

In this diagram (see Figure 2), the **controller package** is responsible for handling user input and mapping it to DTOs. The **service package** contains the business logic of the application and interacts with the entity, DTO, and repository packages to perform database operations. The **entity package** defines the domain objects used in the application. The **DTO package** contains Data Transfer Objects that are used to transfer data between layers of the application. The **repository package** provides a higher-level interface for working with data in the database and encapsulates the low-level details of data access.

The arrows in the diagram indicate the direction of the method calls between the packages. The controller calls methods in the service, which in turn calls methods in the entity, DTO, and repository packages to perform database operations.

Database Diagram

A database diagram is a visual representation of the structure of a database. It shows the tables in the database, the relationships between them, and the columns and data types for each table, also indicating primary keys. Database diagrams can be used to help developers and database administrators understand the structure of a database and even identify potential problems. The database diagram of our system can be seen in Figure 3.

Diagram

Description automatically generated

Figure 3 Database Diagram

The "**user**" table contains personal information of users, including their name, email, phone number, and password. This table serves as the parent table for the "patient" and "healthcare\_provider" tables.

The "**patient**" table contains additional information specific to a patient, such as their age, gender, and medical history. It references the "user" table through a foreign key constraint.

The "**healthcare\_provider**" table contains information about healthcare providers, such as their medical qualifications. It also references the "user" table through a foreign key constraint.

The "**specialization**" table holds a list of medical specializations that healthcare providers can have.

The "**healthcare\_provider\_specialization**" table establishes a many-to-many relationship between healthcare providers and their specializations.

The "**medical\_facility**" table represents a medical facility where patients receive care. It includes information such as the name, location, and contact details of the facility.

The "**administrator**" table contains information about healthcare managers who can make appointments for patients. It references the "user" and "medical\_facility" tables.

The "**medical\_equipment**" table contains information about medical devices and equipment used to diagnose and treat patients. It references the "medical\_facility" table, as an equipment is only available at a certain place.

The "**appointment**" table represents an appointment for a patient and includes information such as the patient, healthcare provider, medical facility, and scheduled time. It references the "patient", "healthcare\_provider", "medical\_facility", and "medical\_equipment" tables.

The "**medical\_records**" table contains information about a patient's medical history, including their diagnoses, treatments, and test results. It references the "appointment" table.

Use Case Diagrams

A use-case diagram (see Figure 4) is a type of UML diagram used to visualize the functional requirements of a system or application. It provides a high-level view of the interactions between actors (users, systems, or other external entities) and the system, depicting the various use cases and the relationships between them. This diagram provides a structure for our application as it helps to identify components in the design phase. It also helps to capture the requirements, some of which are presented now in detail:

**Use Case (UC1)**: register

**Primary Actor**: user

**Main Success Scenario**:

1. The user clicks on the “Register now” button
2. The user enters the details and the chosen password 2 times
3. The user presses the “Register” button
4. The user will be registered

**Alternative Sequence**: The 2 passwords do not match

The client inserts 2 passwords which do not match

The application displays an error message and requests the user to insert the same password

The scenario returns to step 1

**Use Case (UC2)**: login

**Primary Actor**: user

**Main Success Scenario**:

1. The user selects that they want to do a login
2. The user enters their email and password
3. The user presses the “Log in” button
4. The user gets logged-in

**Alternative Sequence**: Incorrect password

The user inserts an incorrect password

The application displays an error message and requests the user to make sure that the password is correct

The scenario returns to step 2

**Alternative Sequence**: Email address not found

The user inserts an email address which is not registered yet

The application displays an error message and requests the user to make sure that they are registered or that their email is correct

The scenario returns to step 2

**Use Case (UC3)**: schedule an appointment

**Primary Actor**: patient (a healthcare provider should also be able to do that)

**Main Success Scenario**:

1. The patient logs in to the application
2. The patient navigates to the “Schedule an Appointment” section
3. The patient selects a healthcare provider from a list of available providers
4. The patient selects a preferred date and time for the appointment
5. The application checks for availability and confirms the appointment
6. The patient receives a confirmation message with details of the appointment

**Use Case (UC4)**: create a medical record

**Primary Actor**: Healthcare Provider

**Main Success Scenario**:

1. The Healthcare Provider opens the application and logs in.
2. The Healthcare Provider selects the patient they want to create a medical record for.
3. The application displays the patient's information.
4. The Healthcare Provider selects the "Create Medical Record" option.
5. The application displays a form to enter the patient's medical information, including current symptoms, and treatment plans.
6. The Healthcare Provider fills out the form with the patient's medical information.
7. The Healthcare Provider saves the medical record.
8. The application confirms that the medical record has been successfully created.

**Alternative Scenarios**: incomplete or incorrect information

If the Healthcare Provider enters incomplete or incorrect information, the application displays an error message and prompts the Healthcare Provider to correct the information.

**Use Case (UC5)**: view medical records

**Primary Actor**: healthcare provider (or user for their own)

**Main Success Scenario**:

1. The healthcare provider logs into the application
2. The healthcare provider navigates to the “Medical Records” section for the corresponding patient
3. The application displays a list of the patient’s medical records
4. The healthcare provider selects a specific record to view details, such as diagnoses, treatments, and test results

**Use Case (UC6)**: manage appointments

**Primary Actor**: administrator

**Main Success Scenario**:

1. The administrator logs in to the application
2. The administrator navigates to the “Manage Appointments” section
3. The administrator views a list of appointments for a specific healthcare provider or for all providers
4. The administrator can cancel or reschedule appointments based on the availability of healthcare providers and medical facilities.

**Use Case (UC7)**: Search for Healthcare Providers by Specialization

**Primary Actor**: patient (or administrator as well)

**Main Success Scenario**:

1. The patient opens the application and logs in.
2. The patient selects the "Find a specialist" option from the main menu.
3. The application displays a list of specializations and prompts the patient to select a specialization.
4. The patient selects a specialization from the list.
5. The application displays a list of Healthcare Providers who have that specialization
6. The patient selects a Healthcare Provider from the list.
7. The application displays the Healthcare Provider's medical qualifications.
8. The patient makes an appointment.
9. The application confirms the appointment and sends a confirmation to the patient and the Healthcare Provider.

**Alternative Scenario**: healthcare provider not available

If the selected Healthcare Provider is not available at the selected appointment slot, the application displays a message informing the Patient that the Healthcare Provider is not available and prompts the Patient to select another appointment slot or another Healthcare Provider with the same specialization.

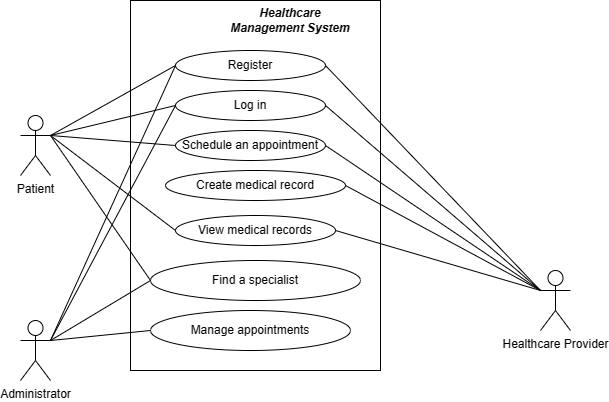


Figure 4 Use Case Diagram

Sequence Diagrams

A sequence diagram is a type of UML diagram which depicts the sequence of actions or messages exchanged between objects in the system, highlighting the order in which they occur. Sequence diagrams are used to model the behavior of a system, and they are often used in software development to design, document, and communicate the interactions between various components of a system. They are usually related to a use case.

The following 4 sequence diagrams correspond to the first 4 use cases mentioned before:

* Register (see Figure 5)
* Log in (see Figure 6)
* Schedule an appointment (see Figure 7)
* Create a medical record (see Figure 8)

A picture containing diagram

Description automatically generated

Figure 5 Sequence Diagram for UC1

Diagram

Description automatically generated

Figure 6 Sequence Diagram for UC2

Diagram

Description automatically generated

Figure 7 Sequence Diagram for UC3

Diagram

Description automatically generated

Figure 8 Sequence Diagram for UC4