

Pet Clinic

Name: Chetan Anca

Group: 30236

Table of Contents

[Deliverable 1 4](#_Toc134692356)

[Project Specification 4](#_Toc134692357)

[Functional Requirements 4](#_Toc134692358)

[Use Case Model 1 4](#_Toc134692359)

[Use Cases Identification 4](#_Toc134692360)

[UML Use Case Diagrams 7](#_Toc134692361)

[Supplementary Specification 7](#_Toc134692362)

[Non-functional Requirements 7](#_Toc134692363)

[Design Constraints 8](#_Toc134692364)

[Glossary 9](#_Toc134692365)

[Deliverable 2 10](#_Toc134692366)

[Domain Model 10](#_Toc134692367)

[UML Class Diagram 11](#_Toc134692368)

[Architectural Design 12](#_Toc134692370)

[Conceptual Architecture 12](#_Toc134692371)

[Package Design 13](#_Toc134692372)

[Component and Deployment Diagram 14](#_Toc134692373)

[Deliverable 3 15](#_Toc134692374)

[Design Model 15](#_Toc134692375)

[Dynamic Behavior 15](#_Toc134692376)

[Class Diagram 16](#_Toc134692378)

[Data Model 16](#_Toc134692379)

[System Testing 17](#_Toc134692380)

[Future Improvements 17](#_Toc134692381)

[Conclusion 18](#_Toc134692382)

[Bibliography 18](#_Toc134692383)

# 

# Deliverable 1

## Project Specification

The pet clinic application I have chosen to develop is designed to address three types of users: administrator, animal owner, and doctor. However, the animal owner is the main user of this application. Its primary objective is to streamline the appointment booking process for the user by providing a list of available doctors, allowing them to choose the most suitable one for their pet and also rate their experience at the veterinary center. The administrator is responsible for managing the database of users, including adding, updating, and deleting them. Both the doctor and animal owner can access their appointment schedules at any time.

## Functional Requirements

One of the main functionalities of the application is the possibility of registration and login so every user who starts to perform any action within the application must register, with the exception of the administrator who already has an account, and the login step must be completed successfully so that the user can enter the next stage.

* A doctor can register, login, update doctor details, create appointments, find appointments, find animals by different criteria.
* An animal owner can register, login, update owner details, register an animal, update animal details, create appointment, update appointment details, find doctors, find animals, by different criteria, give rating to doctors.
* An administrator can perform modifications related to the database: add, update, delete.

Before any data is added to the database, such as during registration, appointment creation or modification by the administrator, validators are employed to verify that the data meets the required standards. The database cannot be updated until all data has been validated.

## Use Case Model 1

### Use Cases Identification

Use-Case: Login.

Level: Subfunction

Primary Actor: User

Main success scenario: Login successful.

Use-Case: Register.

Level: User Goal

Primary Actor: User

Main success scenario: Register successful.

Use-Case: Add user.

Level: Subfunction

Primary Actor: Admin

Main success scenario: User successfully added.

Use-Case: Update user.

Level: Subfunction

Primary Actor: Admin, Doctor, Animal Owner

Main success scenario: User successfully updated.

Use-Case: Delete user.

Level: Subfunction

Primary Actor: Admin

Main success scenario: User successfully deleted.

Use-Case: Create appointment.

Level: User Goal

Primary Actor: Owner

Main success scenario: Appointment successfully created.

Use-Case: Find doctors.

Level: Subfunction

Primary Actor: Animal Owner

Main success scenario: List of doctors successfully returned.

Use-Case: Find animals.

Level: Subfunction

Primary Actor: Doctor, Animal Owner

Main success scenario: List of animals successfully returned.

Use-Case: Find appointments.

Level: User Goal

Primary Actor: Doctor, Animal Owner

Main success scenario: List of appointments successfully returned.

Use-Case: Give rating.

Level: User Goal

Primary Actor: Animal Owner

Main success scenario: Rating registered.

### UML Use Case Diagrams

Diagram

Description automatically generated

## Supplementary Specification

### Non-functional Requirements

***Reliability*** is an important non-functional requirement that refers to a system’s ability to perform its intended functions consistently, without errors or failures. In my project, I took this into consideration by rigorously testing each component in multiple scenarios. This approach ensured that the system not only performed its intended functions successfully, but also gracefully handled cases where unexpected data or requests were encountered. To achieve this, I created specific exceptions within the application and implemented control flows to handle them effectively.

To enhance ***usability***, my application will feature an intuitive user interface that addresses users with varying levels of technical expertise. Users can interact with the application through intuitive buttons, structured forms, and informative content. The application provides users with suggestive buttons that are labeled clearly, which enables them to generate requests and view relevant information with a single click. Using this approach, the application guarantees an effortless user experience.

The application was designed in such a way that it can be organized in layers, and a higher layer depends on a lower layer through abstraction. This design aims to minimize dependencies between layers, ensuring that changes to one layer will not require significant modifications to the entire system. As a result, the application's ***maintainability*** should not be a concern, provided that the system was structured and modularized appropriately.

***Performance*** is an important non- functional requirement so the system should be designed to handle a large volume of concurrent requests without slowing down or crashing. This includes optimizing database queries, minimizing response times, and ensuring efficient use of system resources.

### Design Constraints

The system is built using the Spring framework, which is a widely used and popular framework for developing Java-based web applications. This framework offers a set of tools and libraries that help with the development of web applications, such as the ability to handle HTTP requests and responses, manage database connections, and more. On the data storage side I worked with MySQL.

My system follows the Model-View-Controller (MVC) architectural pattern, which separates the concerns of the application into three distinct components: the model (data and business logic), the view (presentation layer), and the controller (handles requests and manages communication between the model and view). This pattern ensures that the system is well-structured.

Also, within the application I used some external libraries in order to accelerate the development process and provide additional functionality. In order to map the database tables to Java and also to perform database operations in an efficient way, I used Hibernate and JPARepository. Lombok is another library which reduces the time and effort required for development by providing annotations to reduce boilerplate code in the source code.

## Glossary

* Administrator: A user with special privileges who is responsible for managing the database, including adding, updating, and deleting users.
* CRUD operations: Create, Read, Update, and Delete operations that can be performed on the database.
* Model-View-Controller (MVC): A software design pattern that separates an application into three distinct components: the model, the view, and the controller.
* Subfunction: A use case that is a part of a larger use case and helps to achieve the main goal of that use case.
* UML Use Case Diagram: A graphical representation of the use cases in a system and their relationships with other system components.
* User interface: The graphical user interface (GUI) through which the user interacts with the application.

# Deliverable 2

## Domain Model

The domain model helps to provide a clear understanding of the relationships and concepts that exist within the domain of a pet clinic, and how they can be translated into software applications. The domain model encompasses several key classes, including User, Doctor, Owner, Animal, Appointment, and MedicalFacility.

The User class represents a generalization of all types of users who interact with the application, having some attributes related to user identification: id, email, password and user type which can be ADMIN, DOCTOR and OWNER, provided by the enumerated type UserType.

The Doctor class represents the more detailed characterization of the DOCTOR userType, having the specific attributes of a doctor such as: doctorId, first name, last name, email, password, phone number, work schedule and a list of the appointments he has. To be consistent the doctorId is the same as the id of the User object containing the same email and password.

The Owner class represents the more detailed characterization of the OWNER user type, having the specific attributes of an animal owner such as: ownerId, first name, last name, email, password and a list of the animals they want to bring to the pet clinic. To be consistent the ownerId is the same as the id of the User object containing the same email and password.

The Animal class is a model for the animals that an OWNER user has, it is characterized by: animalId, name, owner, type, breed, age, weight and a list of appointments that it has in the pet clinic. The illustration of the animalType is insured by the enumerated type AnimalType which can be: CAT, DOG.

The Appointment class ensures the representation of an appointment, having the attributes: appointmentId, date and time of the appointment, the doctor, the animal and also a list of medical facilities that the owner of the animal has chosen.

The MedicalFacility class represents the medical services that this pet clinic can offer to patients, having as attributes: id, name and price.

## UML Class Diagram

## A picture containing text, indoor Description automatically generated

# Architectural Design

## Conceptual Architecture

From the architectural point of view, the application is based on Layered architecture design pattern. The first architectural layer is the database where the data is stored, and it will be manipulated through the next layer. This layer serves as the actual repository for the software's information, comprised of both the programming that enables access to and management of the database powering the software product, as well as the fundamental database technology itself.

Thus, with the help of the persistence layer, data manipulation will be ensured through interaction with the database, and access to them will be clear. Actually, this access is possible through the JpaRepository API which offers the possibility to perform CRUD type operations.

Through the business layer, data received from the persistence layer will be manipulated and possible validations will be performed, and in case of non-conformities, this will be signaled by throwing suggestive exceptions. Through the intermediary of this layer, various services necessary within the application will be obtained, such as queries, delete, insert and update. Also, here you can find the models mentioned in the previous sub-chapter and also the DTOs which are basically a reflection of the models that only contain the data that you want to be transmitted to the next layer.

The last layer is the presentation layer, which deals with the definition of some controllers which facilitates the connection with the frontend through the HTTP communication protocol. It includes the graphical design of the software, as well as any code to manage user interaction. Only logic specific to the UI should be found within this layer.

## Package Design

Chart, box and whisker chart

Description automatically generated

## Component and Deployment Diagram

A picture containing text, indoor, white

Description automatically generatedDiagram

Description automatically generated

# Deliverable 3

## Design Model

### Dynamic Behavior

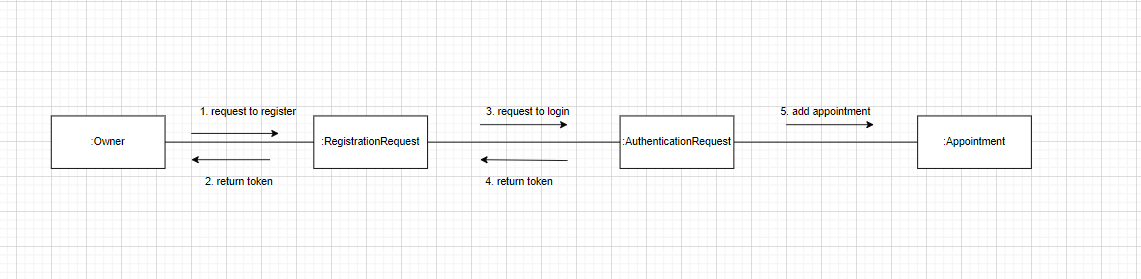
1. First scenario: make an appointment.
2. Second scenario: give rating.

A picture containing text, diagram, line, parallel

Description automatically generated

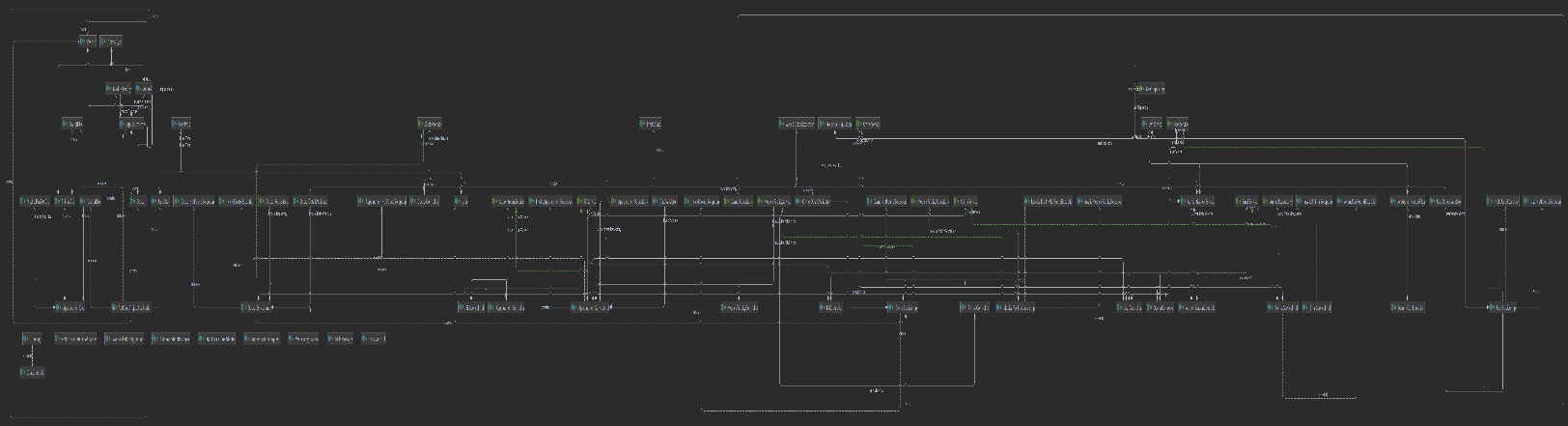
A picture containing text, diagram, parallel, line

Description automatically generated



### A picture containing line, screenshot, diagram, plot Description automatically generated

### Class Diagram



Gof Patterns:

1. Composite design pattern: The composite design pattern was utilized in the development of the Appointment entity within the project. This entity encompasses a doctor and an associated animal, and both classes contain a list of appointments. Additionally, an animal is linked to an owner, and an owner maintains a list of animals. To be consistent, when a doctor or an animal is deleted, all the appointments related to them are deleted. This behavior is encountered withing the relations between Animal and Owner too, so when an Owner is deleted all the animals related to him/her are deleted. Therefore, the application of the composite design pattern becomes evident in these scenarios.
2. Builder design pattern: Builder is a creational design pattern that lets you construct complex objects step by step. I used this pattern within my project in the context of model classes to create objects of these types in a more flexible way and to prevent any problem that may be encountered. This pattern was especially helpful in terms of security at the registration and authentication part.

## Data Model

A picture containing text, diagram, line, plan

Description automatically generated

## System Testing

I have implemented unit tests for the service classes in order to validate the correctness of CRUD operations, including save, update, delete, and various queries. These test classes include UserServiceTest, OwnerServiceTest, DoctorServiceTest, AnimalServiceTest, and AppointmentServiceTest.

Within each of these test classes, I have thoroughly tested all the methods provided by the service classes. I have covered both scenarios that should pass successfully and scenarios where exceptions are expected to be thrown.

To facilitate the testing of these services, I have utilized the Mock framework. This allowed me to substitute and use different classes that are necessary for testing the operations, such as repository classes and validators. By using repositories, I was able to access and validate the CRUD operations, ensuring that the implemented service methods behave as expected. The validators were instrumental in validating the data before it is persisted in the database, ensuring data integrity.

By employing these unit tests with Mock, I can confidently assess the functionality of the service classes, ensuring that they perform the intended operations correctly and handle exceptions appropriately. These tests provide a solid foundation for maintaining code quality and verifying the expected behavior of the Pet Clinic system.

## Future Improvements

 Some of the improvements considered for the project:

1. Doctor Rating and Feedback

Extend functionality to allow animal owners to provide feedback and ratings for doctors after appointments. This will improve transparency and credibility while providing valuable insights for clinic management and prospective animal owners.

1. Integration with Payment Gateways

Integrate a payment gateway service to facilitate online payments for various clinic services, including appointment fees, medications, and other products. This will enhance convenience for animal owners and streamline the clinic's financial operations.

1. Automated Reminders and Notifications

Set up automated reminders and notifications to keep animal owners and doctors informed about upcoming appointments, medication schedules, and other important events. This will help reduce no-shows and improve communication between the clinic and its users.

When implementing these improvements, it is important to prioritize security, scalability, and performance to ensure the application can handle increased usage and protect sensitive user information.

## Conclusion

In conclusion, this application served as a valuable opportunity for me to delve into the development of a more intricate project and gain proficiency in programming with Java Spring Boot. Additionally, the frontend aspect of the application posed a challenge as it marked my initial foray into this domain. The veterinary clinic we designed proved to be a practical application that enhances the communication efficiency between animal owners and the clinic's doctors. Overall, this project has allowed me to broaden my skill set and deliver a useful solution to the veterinary domain.

## Bibliography

<https://spring.io/guides/tutorials/rest/>

<https://www.baeldung.com/spring-component-repository-service>

<https://spring.io/guides/gs/securing-web/>

<https://angular.io/guide/what-is-angular>

<https://material.angular.io/>