UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Veterinary Clinic

Name: Dunca Denisa

Group:30237

Table of Contents

Deliverable 1

Project Specification

The project “Veterinary Clinic” is a full stack application that represents a useful way for pet owners to program veterinary appointments and for doctors to stay organize with their work. The project is made in principle using Java and SpringBoot on backend, React Js on frontend and MySQL for the data base, also the testing part is made in Intellij with Mock annotation, and the data base connection test is made with Postman.

Functional Requirements

As every other application has a log in page, this one has one too, where users can log into their account or can create a new account.

This application has two kinds of users, the ones that are the veterinary doctors that can manage their appointments and see their everyday schedule for work, they also can add, delete or modify information about them or the other doctors, the ones that are the owners that can register their pets, find out information about the clinic and make appointments with the veterinary.

The owners, after they create an account, or register into one, have the functionality to add a pet into their list of pets, they must complete a short description about the pet that can be useful for their veterinary. Additionally, the owner can access information about the clinic, the prices, and their appointments. They can select a pet from their list and program an appointment for their pet problem.

The vets, after they log in, they can see their everyday appointments and a short description about their little animal patients and the problems that they have, so they can be prepared when the day of appointment arrives.

Use Case Model 1

Use Cases Identification

Use-Case: Owner log in, add pet and schedule appointment

Level: User Goal

Primary Actor: Owner

Main success scenario:

1. We assume that the owner has already created an account.
2. The first step he does is to log into his account by inserting his username and password.
3. After he grant access to the site, the owner selects from the header the button “My Pets”.
4. Here he can find a list of profiles for his pets if he already has created ones, or he can add a new one by selecting the button “Add new pet”.
5. After that, a pop up will appear where the owner must complete all the information needed for the pet.
6. When he completed all the information, he can finally press the button that says

"Finish" and the pop up will disappear, instead will appear a short notification that says, "Your pet has been added" and now the owner can see his pet on the list of pets.

7. After that he can schedule an appointment.

Extensions:

1. After logging in the owner does not see any pet already registered and he add one
2. The owner does not complete a field that is required and press the "Finish" button.
3. Short notification will appear near the uncompleted field that announce that the owner must complete it to press the "Finish button"
4. The owner completes the field, press "Finish", the pop up disappears, and the success notification appear
5. After that the owner logs out.

Use-Case: Owner schedule an appointment and log out

Level: User Goal

Primary Actor: Owner

Main success scenario:

1. After logging in, the owner selects from the list of pets, the one that he knows it needs an appointment with the vet.
2. After selecting the pet profile, the owner can press the button "Schedule appointment" and a new pop up will appear
3. Here he can select the doctor he wants and see the doctor's schedule so that the owner will know when the doctor is free for him.
4. The owner chooses a suitable date and can add a note about the problem that the pet has or about what is the purpose of the appointment and press "Finish".
5. The pop up will disappear and a new notification will appear that says: "Your appointment has been successfully added".
6. After that the owner can choose to see his appointments by pressing the button "My appointments"
7. Or can log out by selecting the button "Log out". Extensions:

1. After logging in the owner selects the pet for appointment.
2. Select "Schedule appointment"
3. The owner chooses a date that is already taken, or the doctor is not working
4. A new notification will appear that says "The date you chose is not satisfiable"
5. The owner chooses a good date and finish the appointment
6. He logs out

Use-Case: Vet sees his scheduled appointments

Level: User Goal Primary

Actor: Vet

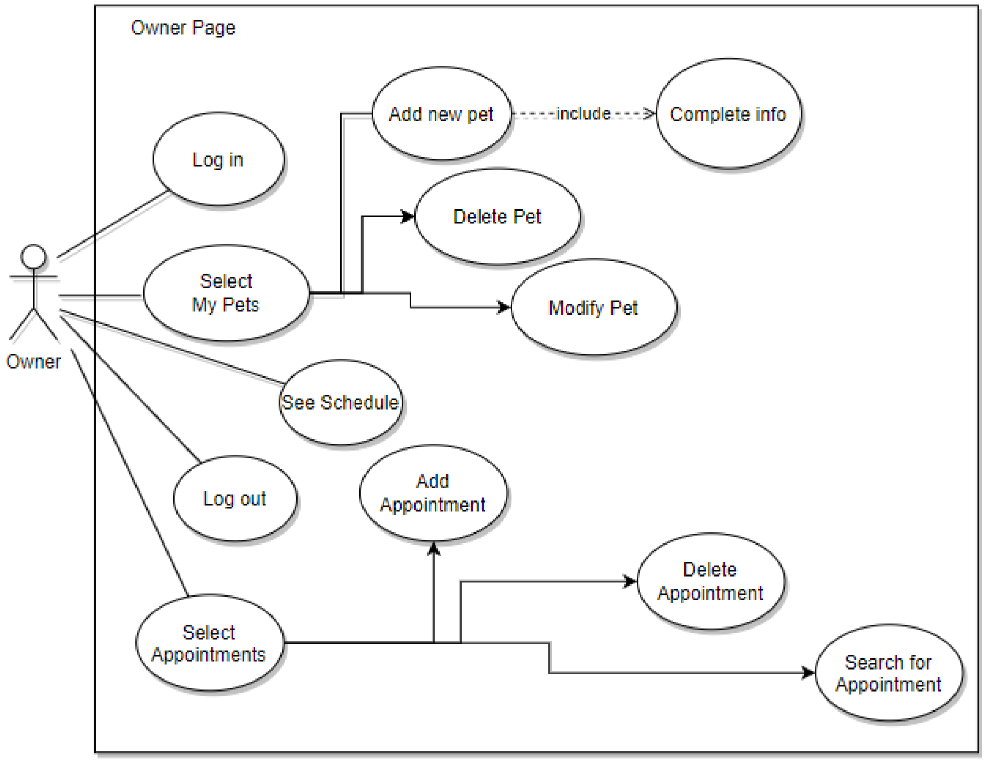
Main success scenario:

1. After the register phase, the vet can access his schedule, here he can see all the appointments for the day or the week, he can see the ones that have been completed or the ones that must be completed.
2. If the appointment is completed, the vet can press the button “Complete”, and the respective appointment will be moved to the completed appointments.
3. He can also choose to see the information about the pet and about the owner for every appointment.
4. After that he can log out.

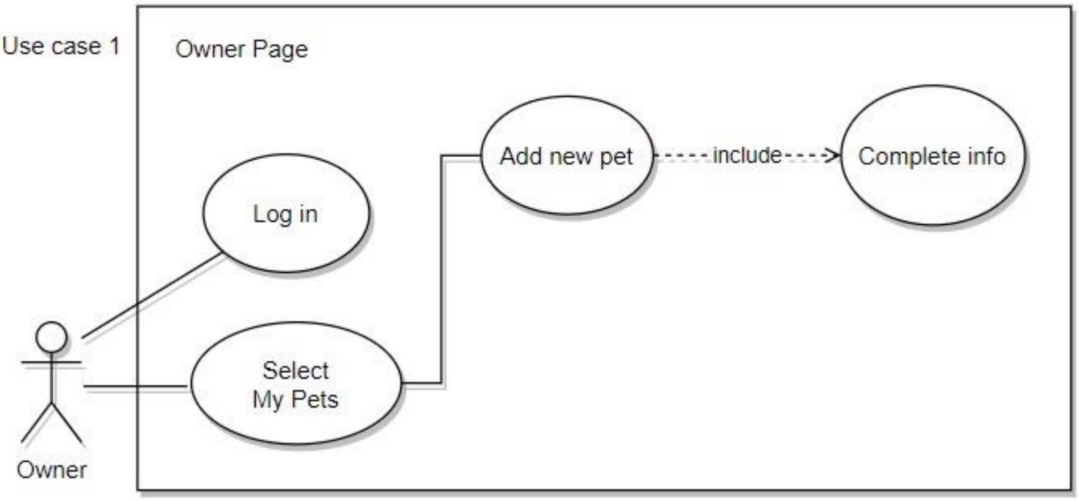
Extensions:

1. The vet accesses his appointments
2. Chooses one to read the information about it, by pressing the button “Info”
3. A pop up will appear with the pet information and the owner contact
4. After reading, the vet can press the button “Close”
5. And he can log out

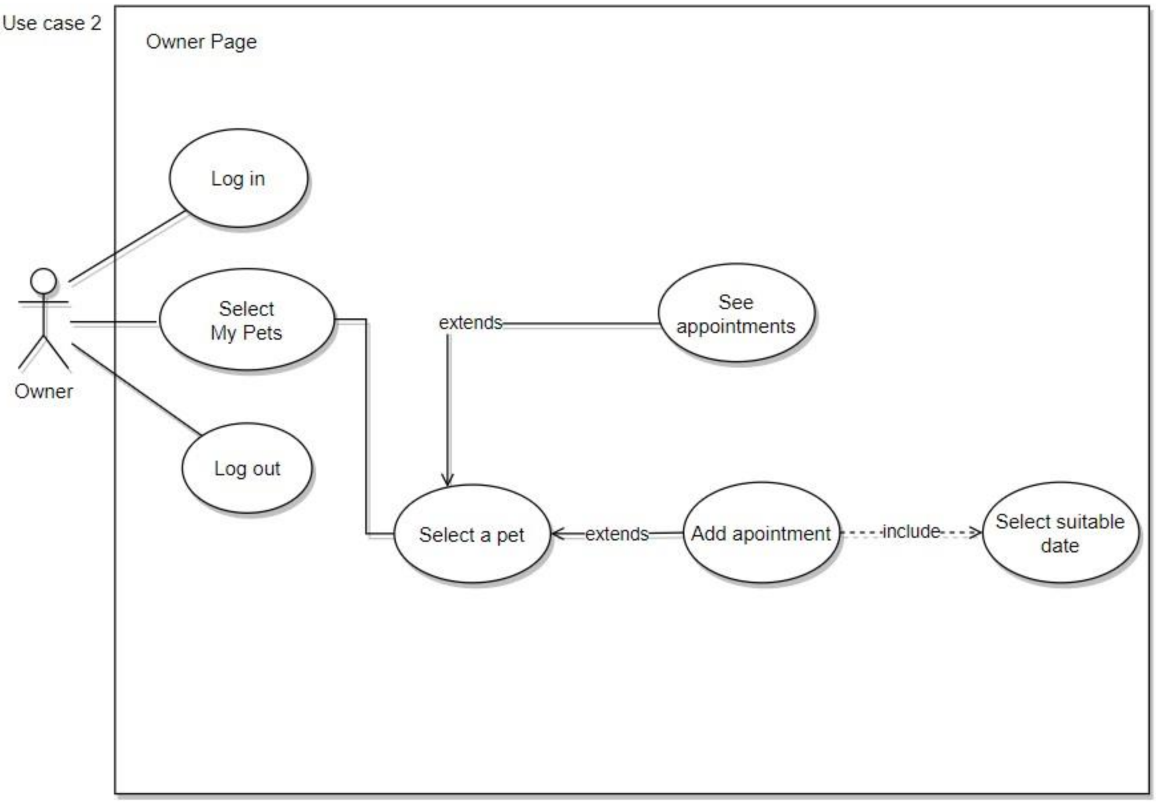
UML Use Case Diagrams

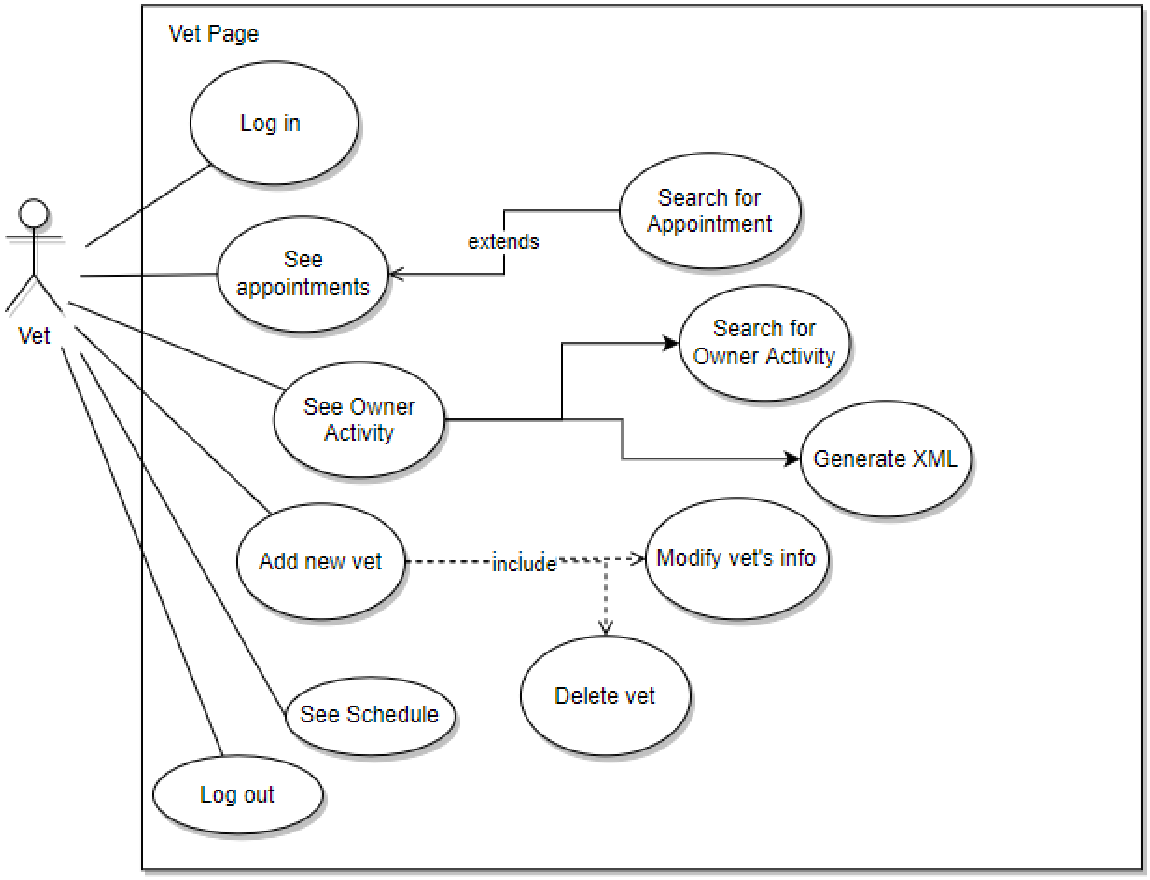


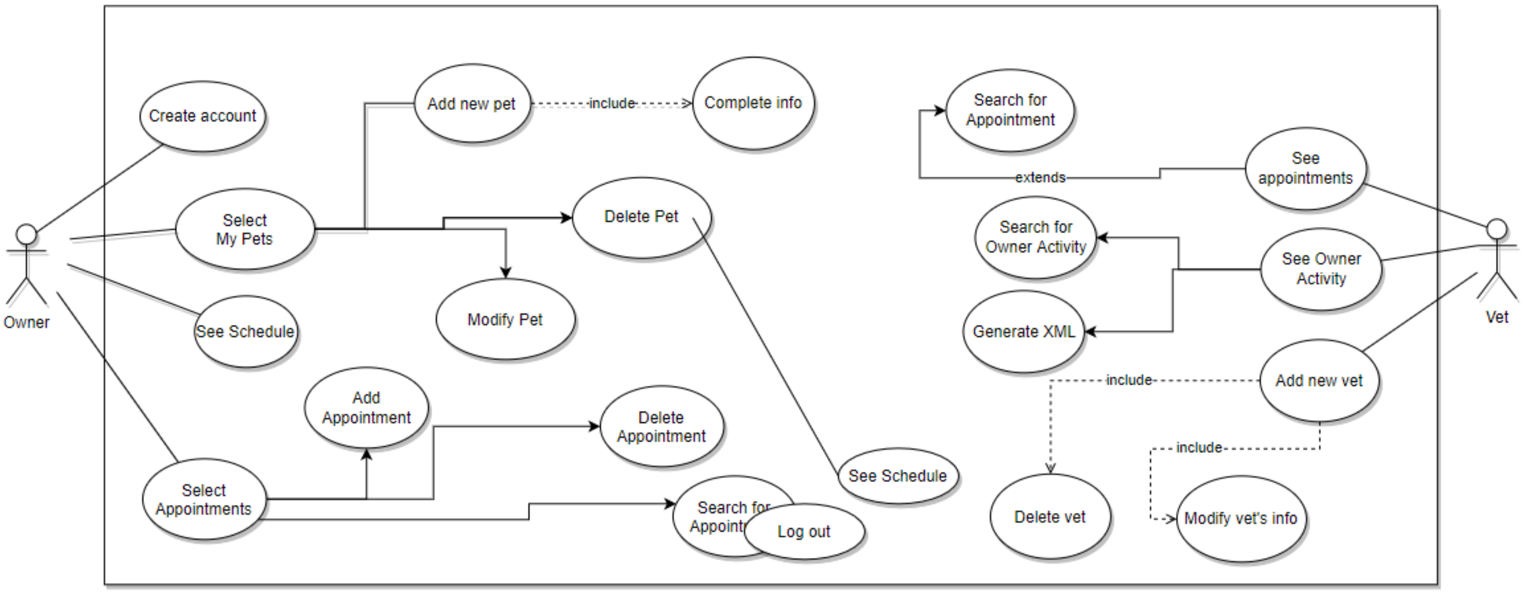
Use case 1



Use case 2




Vet Page
Vet
Log in
See appointments
extends
Search for Appointment
See Owner Activity
Search for Owner Activity
Generate XML
Add new vet
include
Modify vet's info
Delete vet
See Schedule
Log out


Owner
Create account
Select My Pets
See Schedule
Select Appointments
Add new pet
include
Complete info
Delete Pet
Modify Pet
Add Appointment
Delete Appointment
Search for Appointment
Log out
See Schedule
Search for Appointment
extends
See appointments
Search for Owner Activity
See Owner Activity
Generate XML
Add new vet
include
include
Delete vet
Modify vet's info
Vet

Supplementary Specification

Non-functional Requirements

This application has a series of nonfunctional requirements that add a plus of integrity and safety. The first one, that is also very important is the security functionality, this kind of application that requires implication of medical help and information about the patients has a high risk of data attacks that's why the application offers equally to the owners and vets a secure access by authenticating with a personalized username and password.

Another non-functional requirement is reliability, the site must function without any bugs for the whole time the owner or the vet uses it. This king of requirement is very important, so the user feel safe about the site and continue using it without asking if he did something wrong or if the site is a trustworthy one.

The scalability requirement helps the site to grow and to add more features, the number of users that access the site or that makes an account can increase and the owners can add as many pets as possible. Furthermore, the frontend and backend are implemented so that can be open for extension.

The most logical nonfunctional requirement is the usability one, without it the site can't be used by anyone. Furthermore, with usability the user gets to access every information on the site that he needs and can manage to schedule appointments for the pets, which is a very needed feature and also the vet can access his schedule in a very easy mode.

Design Constraints

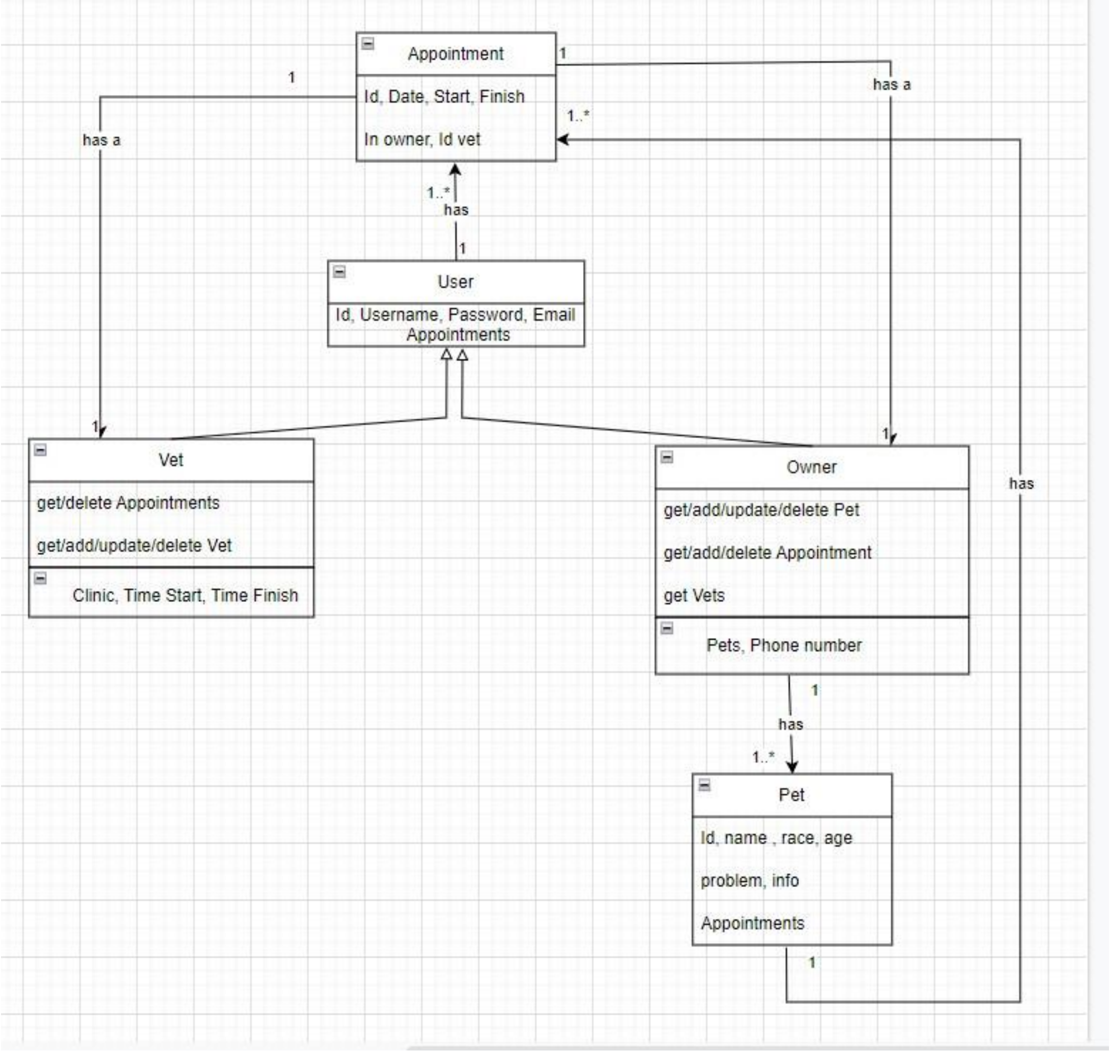
To create this project, different kind of libraries have been used. On backend, as I used Spring Boot to create the data base, to simplify the process I used ItelliJ with Maven. On frontend, as I used ReactJs I had to use some of the typical libraries for a site, such as React and React Dom. Besides, I used the layered architecture on backend and on frontend I used components made of functions instead of classes.

Glossary

JDBC- Java Database Connectivity
JPA – java Persistence API
API – Application programming interface
Js – Javascript

Deliverable 2

Domain Model



In the Domain Model Diagram, we can find the principle classes that are the core of the project:

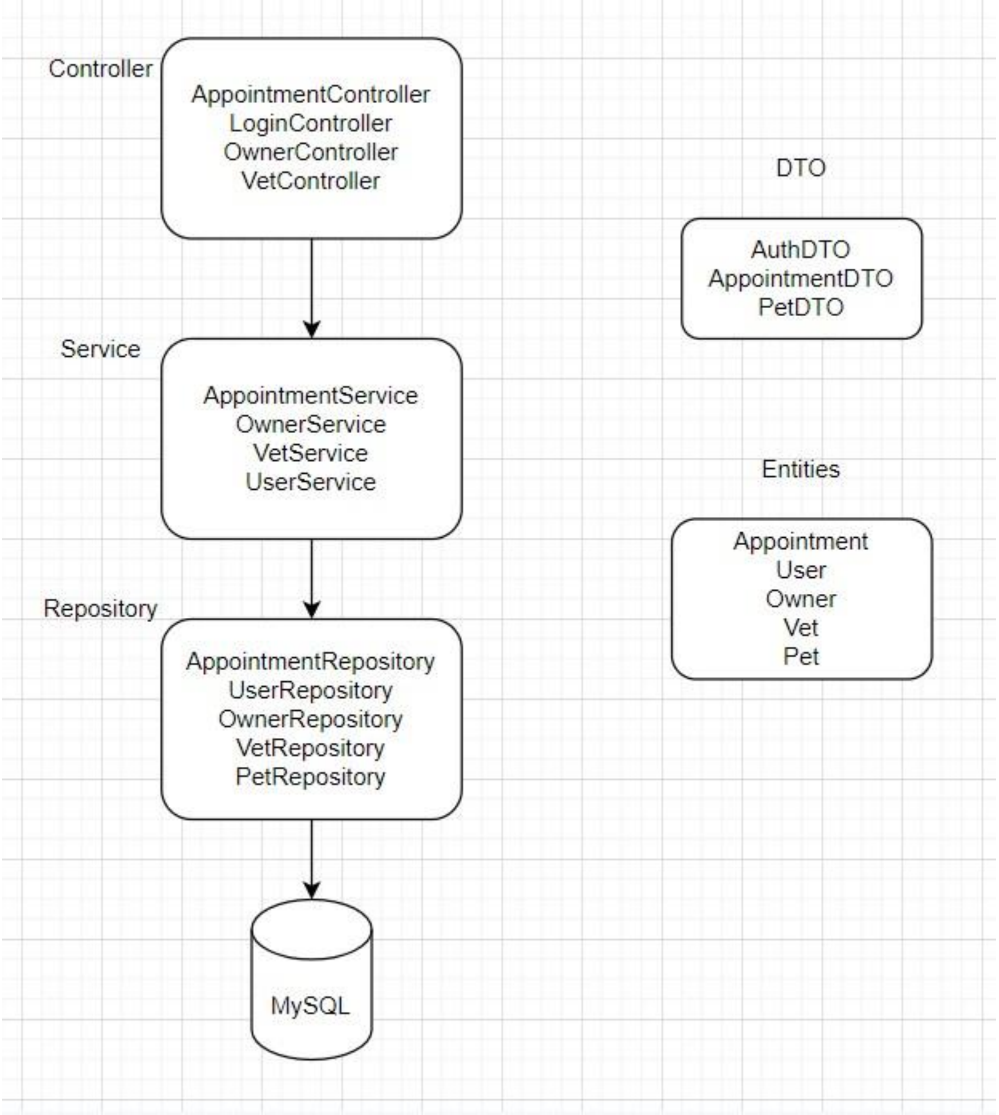
- The user, the class that has an appointment list, id, username, password and email
- The owner class and the admin class are the ones that extends the user class
- The owner also has a list of pets
- The pet class has information about the pet and a list of appointments
- The appointment class detain the information about the appointment and a vet id and a owner id

Architectural Design

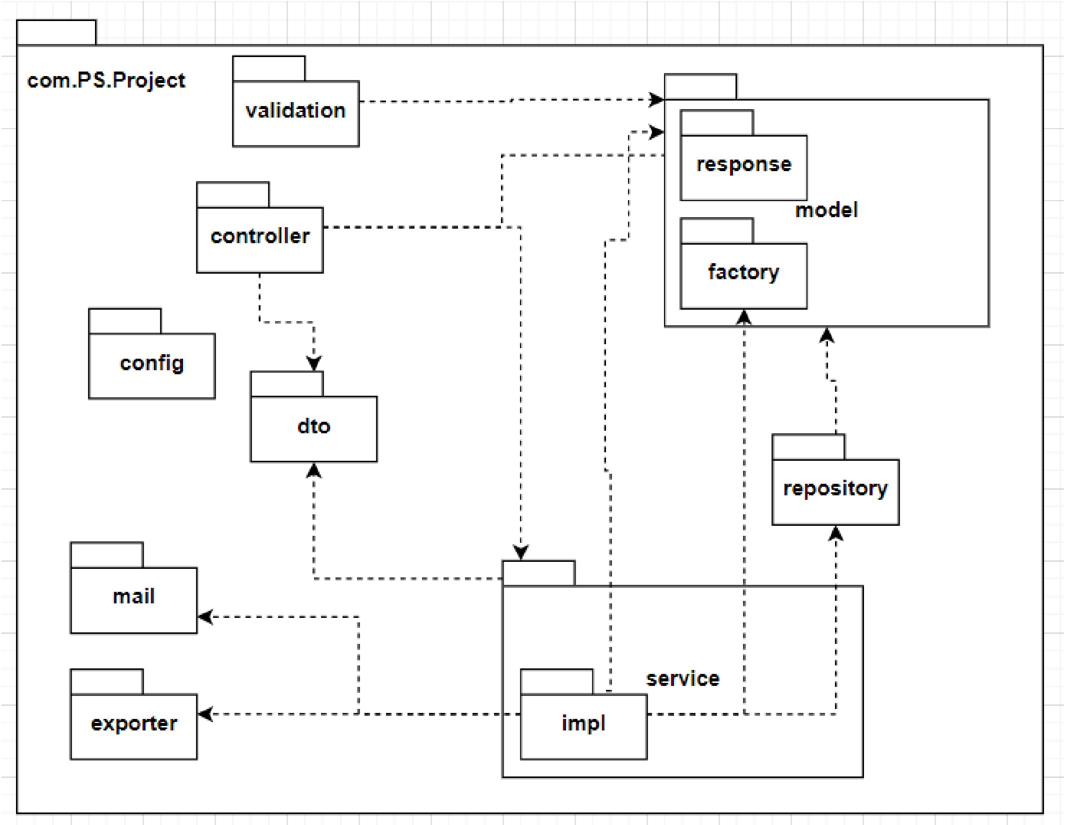
Conceptual Architecture

In order to create this project, I used the Layerd Architecture. A layerd Architecture is the way the project is organized, and it is made out of three big categories of layers: presentation layer, application layer, domain layer and infrastructure layer. I will explain shortly what contains each of these layers:

- The presentation layer contains all the classes responsible for presenting the user interface and sends the resonse back to the client. (Controller layer)
- The application layer contains all the logic in the application, all the funtionalities. (Service layer)
- The domain layer represents the entities and business rules. (Repository layer)
- The infrastructure layer contains the classes responsible for the technical stuff, in this project we have DAO classes, and entities form Model.

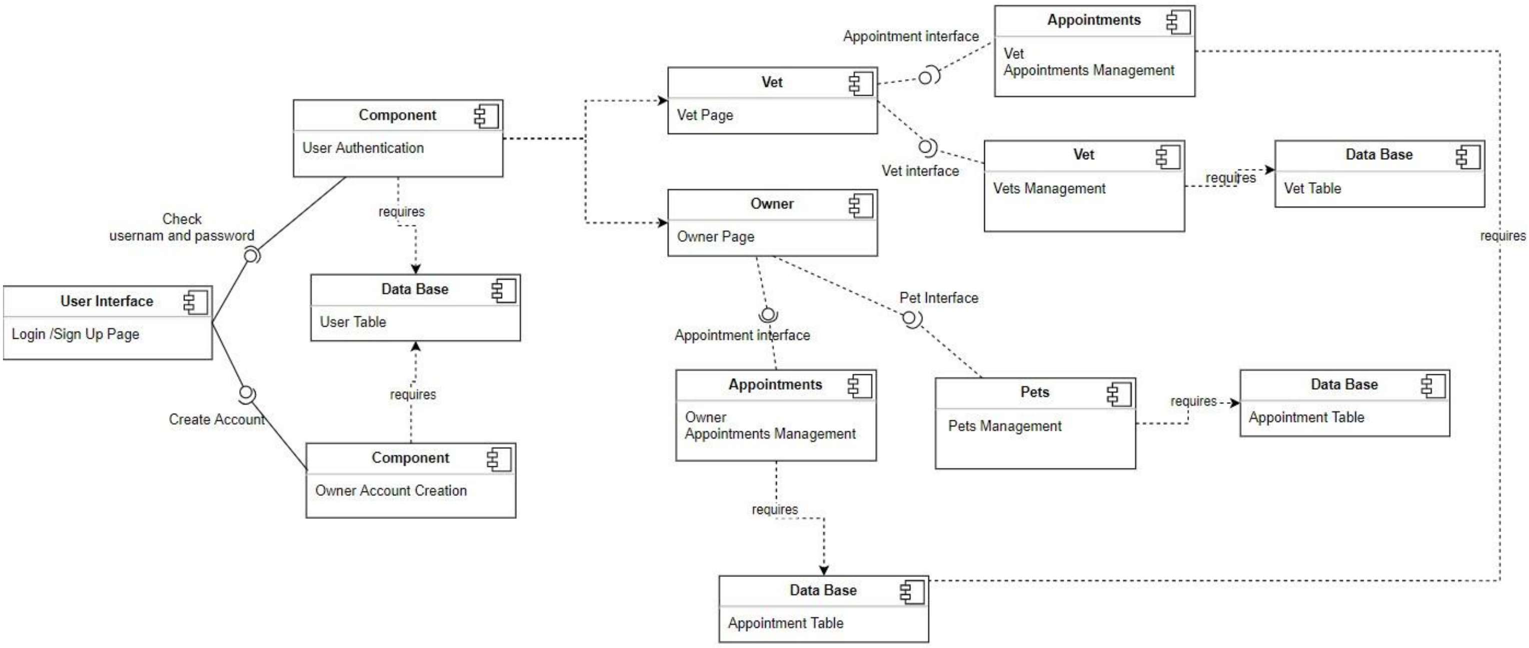


Package Design

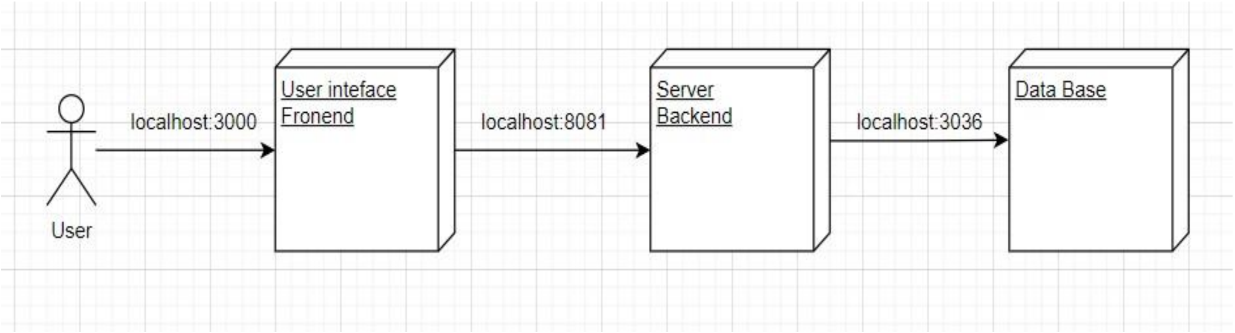


Component and Deployment Diagram

Component Diagram:



Deployment Diagram:



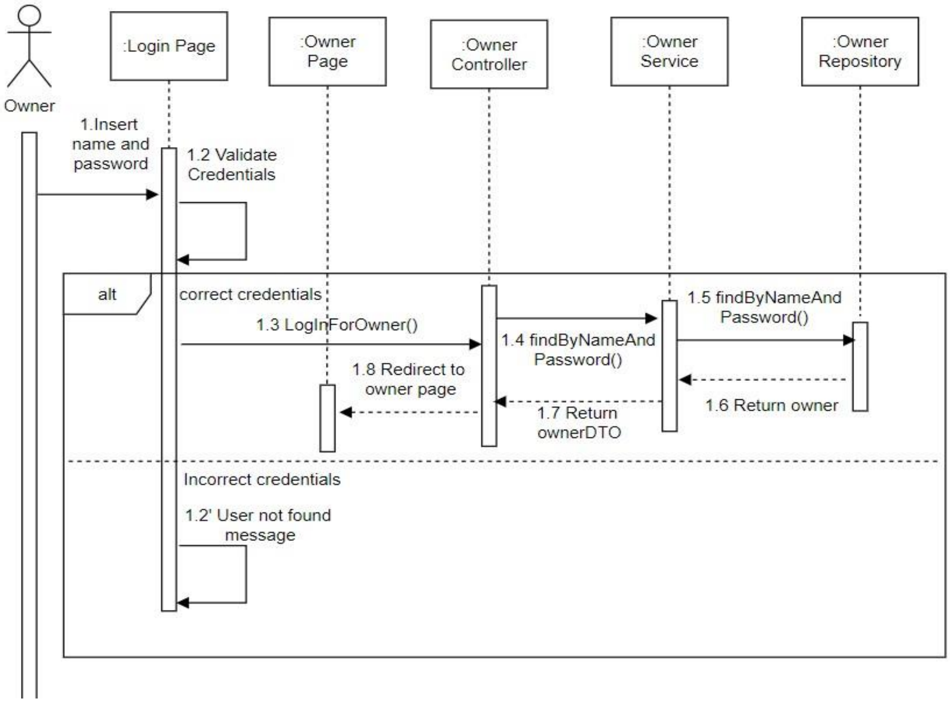
Deliverable 3

Design Model

Dynamic Behavior

Sequence Diagram for the log in for owner. The chronological events start with the insertion of the credentials, the name, and the username, and it goes on two different cases depending on the correctness of the input data. If the credentials are correct it redirects to the owner page, otherwise it shows an error message.

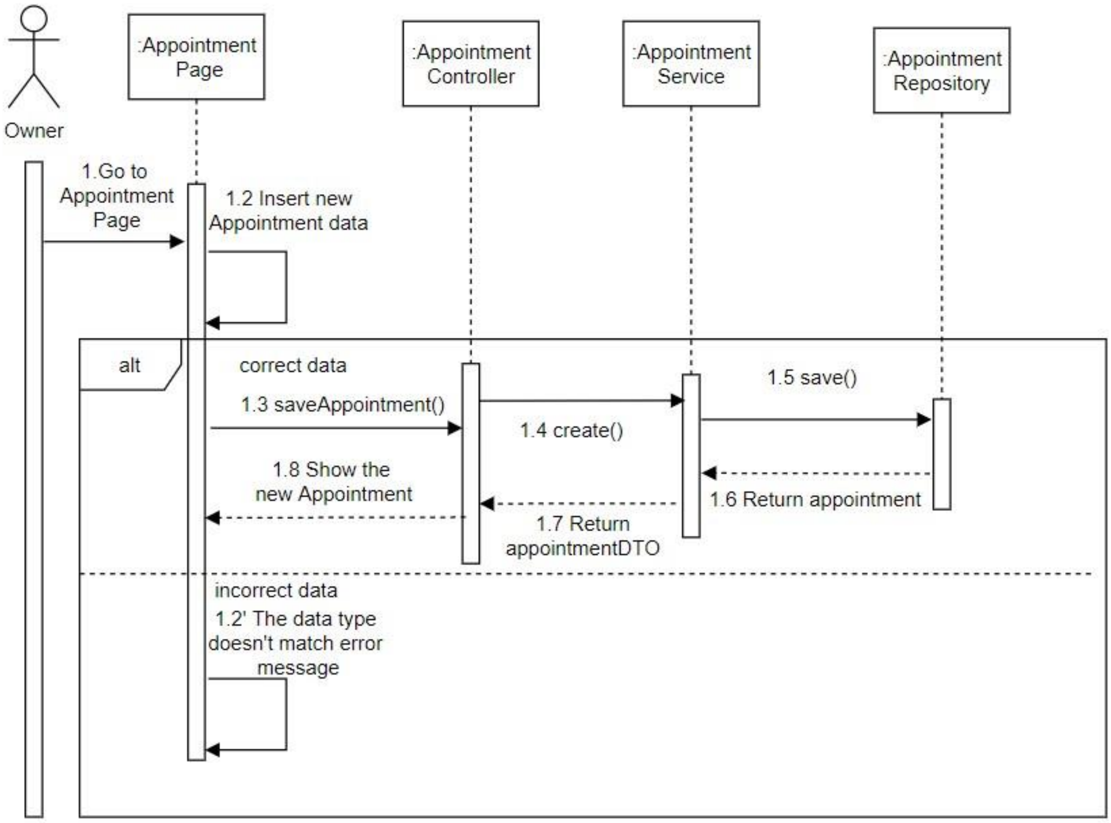
Before the redirecting part to work, the inserted data needs to be given as parameters to the LogInForOwner() method from Owner Controller, after that they are given to the findByNameAndPassword() method from Owner Service which calls the method with the same name from Owner Repository. Here we have access to the database, and we get returned the owner that has the name and password that matches the ones from database. After that the owner is converted to ownerDto and the owner is redirected to the Owner page (the log in succeed)



The second Sequence Diagram is for adding a new Appointment to the list of appointments. The chronological events start with the owner being on the Appointment Page, here the owner can see the list of the appointments, and the add new appointment button. After inserting the data about the new appointment and pressing the save button, if the inserted data is correct, the appointment is added, otherwise some error messages will be shown.

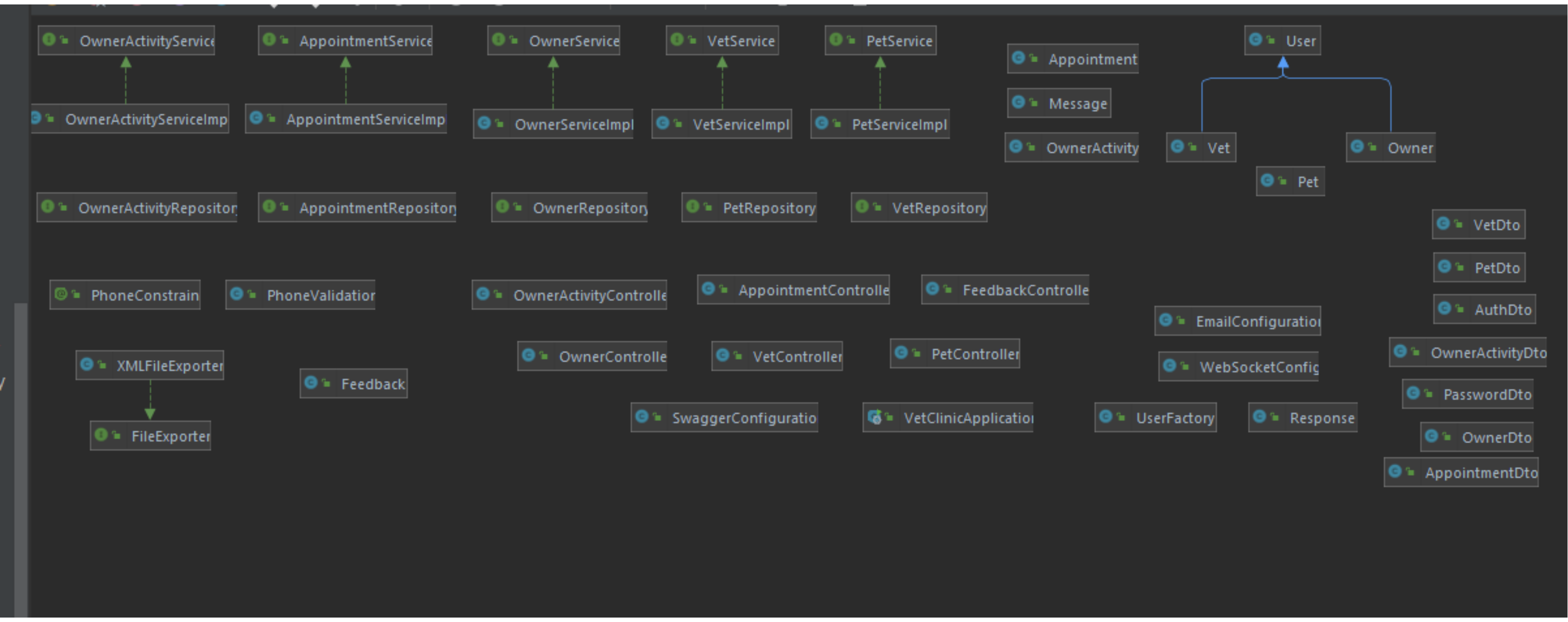
In order to add the appointment, the inserted data as an appointment is given as parameter to the saveAppointment() method from Appointment Controller, here is called the method create() from the Appointment Service. Finally, in the service is called the method save() from Appointment Repository which access the data from database and store the new appointment in the table.

After that the appointment is also return so it can be shown on the list of appointments from Appointment Page.



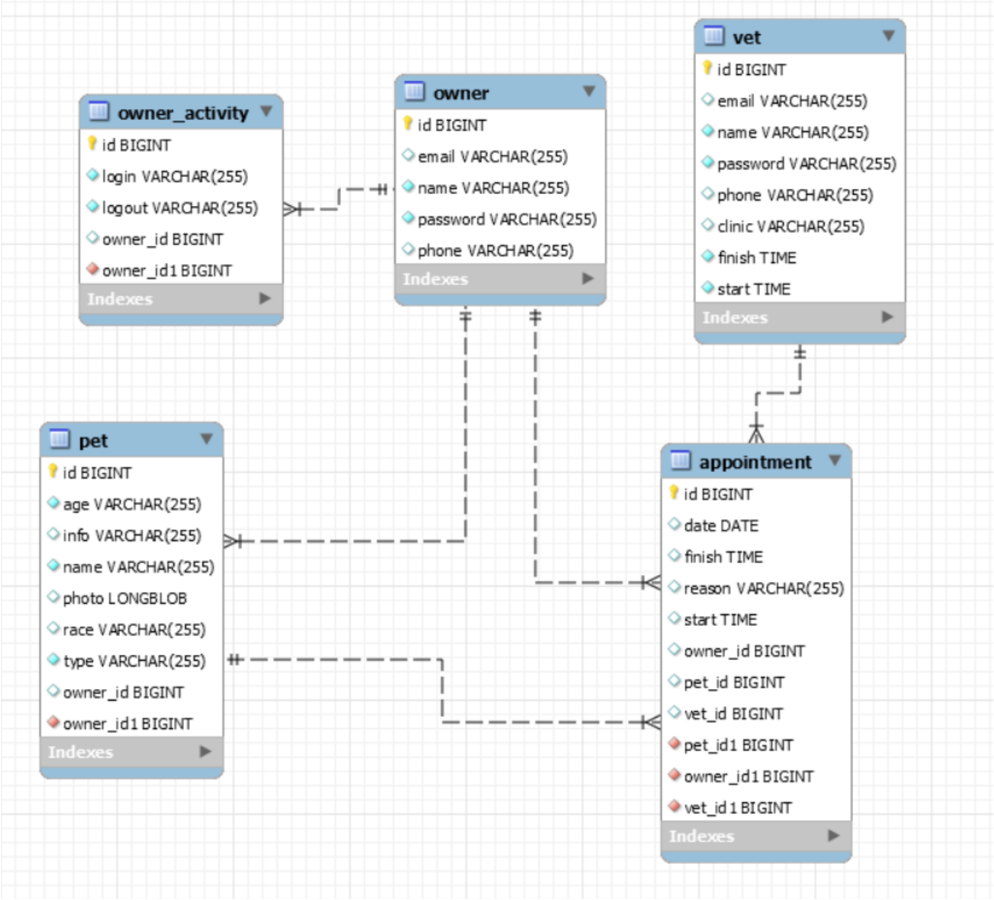
Class Diagram

Here is the class diagram generated from IntelliJ, there are all the classes used for the project and the links between them. The inheritance relation between User and Vet and Owner class, and the implement relation between service interfaces and their implementation.



Data Model

The Data Model diagram is the diagram generated by MySQL for the tables.



System Testing

To test that the CRUD operations are working correctly, I've created two test classes OwnerServiceImplTest and VetServiceImplTest. In each of them I've created test methods for testing if the find by name and password method from repository works as it should, and methods for testing if the save and delete actions from repository return the expected data.

I used the framework Mockito, which is used for mocking objects in software testing, it simplifies the development of tests for classes with external dependencies. Mocks can return different values depending on arguments passed into a method. The when().thenReturn() method chain is used to specify a return value for a method call with pre-defined parameters.

Future Improvements

Some future improvements to this project could be, a proper time picker for choosing the time for the appointments, more useful data about the clinics and information about the vets and the prices for every pent control. Also, the interface could be made much better with proper CSS and more style. However, I think that this project contains good functionalities for a starter project, and it is open for adding more improvements in the future.

Conclusion

As a conclusion, I could say that the project has respect the requirements and it has many interesting and useful functionalities, such as reCAPTCHA, forgot password, a scheduler API, upload photos, encrypting password, WebSocket notification and so on.

Bibliography

https://app.diagrams.net/?src=about
https://devexpress.github.io/devextreme-reactive/react/scheduler/
https://www.npmjs.com/package/react-google-recaptcha
https://mailtrap.io/inboxes
https://www.geeksforgeeks.org/file-uploading-in-react-js/
https://www.javatpoint.com/how-to-encrypt-password-in-java
https://betterprogramming.pub/solving-real-life-problems-in-javascript-sending-server-side-notifications-through-websockets-a3bdb2cc065