Veterinary Clinic: Veterinae

Project documentation

Student(s):

- Laura Andreea Nadu

Group: 30431

MINISTRY OF EDUCATION
TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA, ROMANIA

Contents

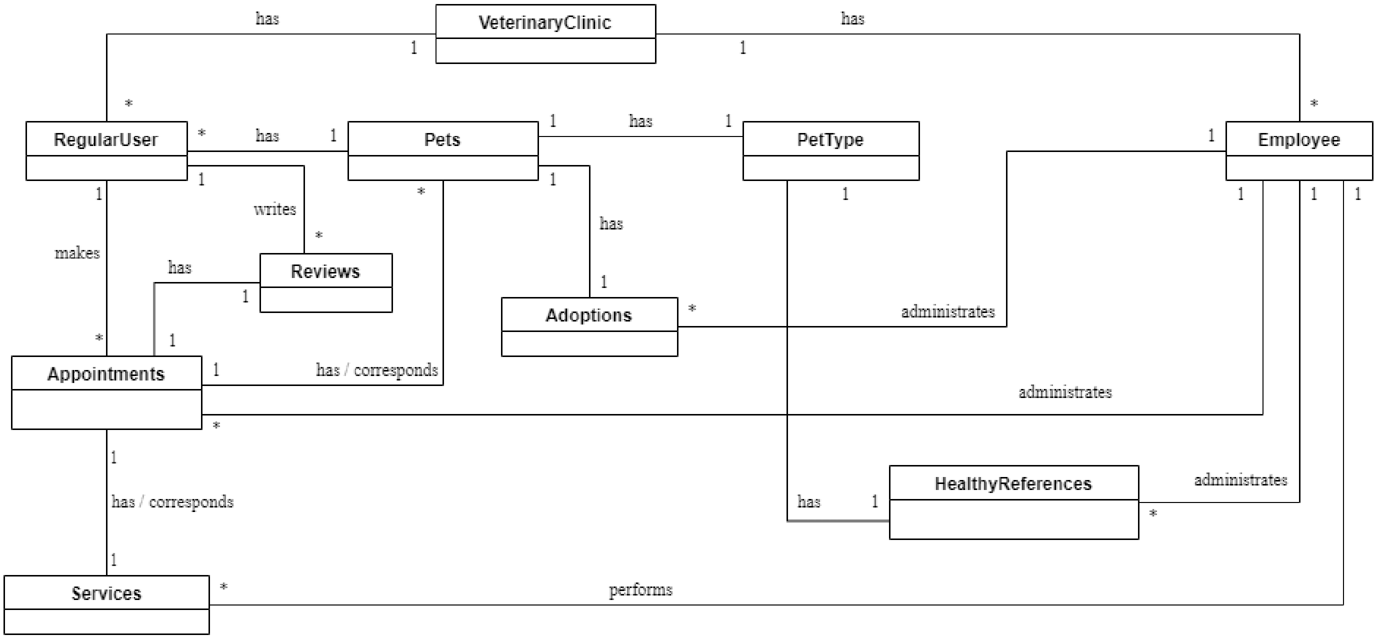
I Project specification

Veterinae is a Web Application for a veterinary clinic. This is an application that could be used by regular clients to inform themselves and make appointments to various services, while an employee to the clinic could log in to manage these appointments and to post available pets for adoption.

For this clinic a regular user could see information about the available services and make appointments, such as a medical control, analysis, surgeries, cosmetics, stomatology, vaccinations and other animal-related medical controls. Once finished, these appointments could receive a review from the user. They could also see a list with all animals that are given out for adoption, and they could create an appointment to a play-date with these animals. Besides these, the users could see an "Health Reference" page where they could see basic healthy diets and life-style guides for their animals.

The employee side of the application requires him/her to login in order to manage the appointments and to post the adoption list.

1.1 Domain Model Diagram



II Use-Case model

A use case represents a specific situation in which a product or service could potentially be used. It is a list of actions or event steps typically defining the interactions between an actor and a system in order to achieve a goal.

For the **Veterinae** Application, there are 2 actors: the regular user and the employee.

Use cases are a technique for capturing, modelling and specifying the requirements of a system. Thus, for the system to be implemented, from the use cases we should be able to extract the system requirements in order to achieve the desired goal.

2.1 Users and stakeholders

Users:

- Website visitor
- Regular user
- Employee

Stakeholders:

- Other Animal Hospitals and Clinics, like Karavet
- Development Team
- Testing team
- Project owner
- Project manager
- Partners and Sponsors
- Doctors
- Users / Clients

2.2 Use-Case identification

Use Case 1:

- **Use case name:** Regular user makes an appointment
- **Level:** User goal
- **Main actor:** Regular user
- **Main success scenario:**
    - Precondition: the user must be logged in
    - Behavior: The user selects a service and creates an appointment for one of his/her pets
    - Postcondition: The new appointment is saved in the database and could be seen by the employees.
- **Extension:** The user tries to make an appointment for a redundant operation such as an appointment to castrate a pet, which could be done only once in its life.

Use Case 2:

- **Use case name:** Regular user reviews an appointment
- **Level:** User goal
- **Main actor:** Regular user
- **Main success scenario:**
    - Precondition: the user must be logged in, the appointment has to be completed
    - Behavior: The user selects an appointment for one of his/her pets and leaves a review for the received service.
    - Postcondition: The review is saved in the database and will be taken into account in computing the overall score of the clinic's service.
- **Extension:** The user refuses to add a review, which means this appointment won't be taken into account for the overall score.

Use Case 3:

- **Use case name:** Employee completes an appointment
- **Level:** User goal
- **Main actor:** Employee
- **Main success scenario:**

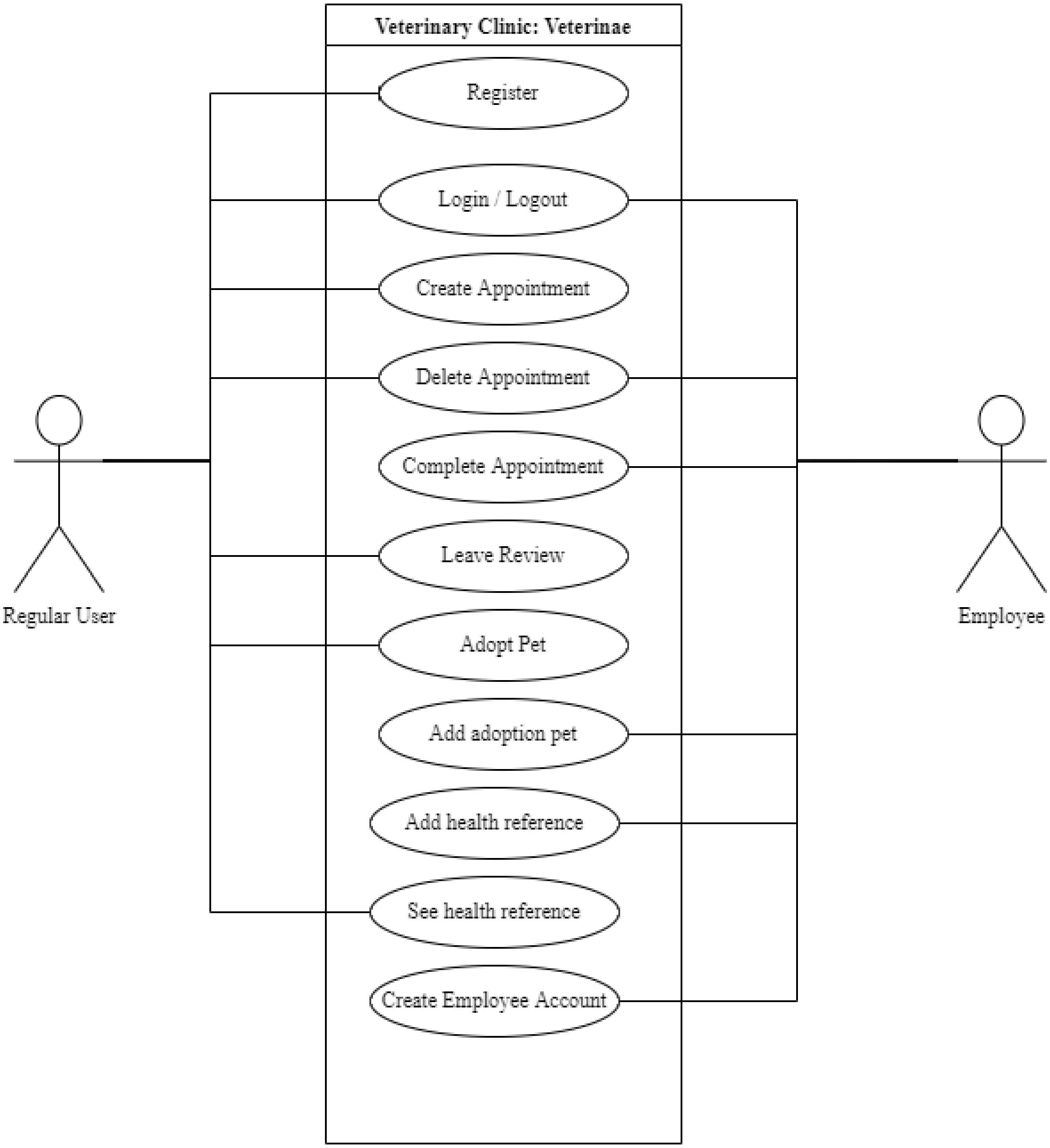
  - Precondition: the employee must be logged in, a user must have created an appointment to one of this employee’s services.
  - Behavior: The employee selects an appointment for one of his/her services and marks it as completes.
  - Postcondition: The appointment is saved in the database and the regular user that made the appointment should be able to leave a review.

Extension: The appointment is no longer valid, so the employee must delete it.

Use Case 4:

- **Use case name:** Employee adds a new pet to the adoption list
- **Level:** User goal
- **Main actor:** Employee
- **Main success scenario:**
  - Precondition: the employee must be logged in
  - Behavior: The employee adds a new pet for adoption.
  - Postcondition: The adoption is saved in the database and the regular users should be able to view it.
- **Extension:** The pet details are invalid, so the employee cannot create the adoption option.

2.3 UML Use-Case diagram



III Architectural design

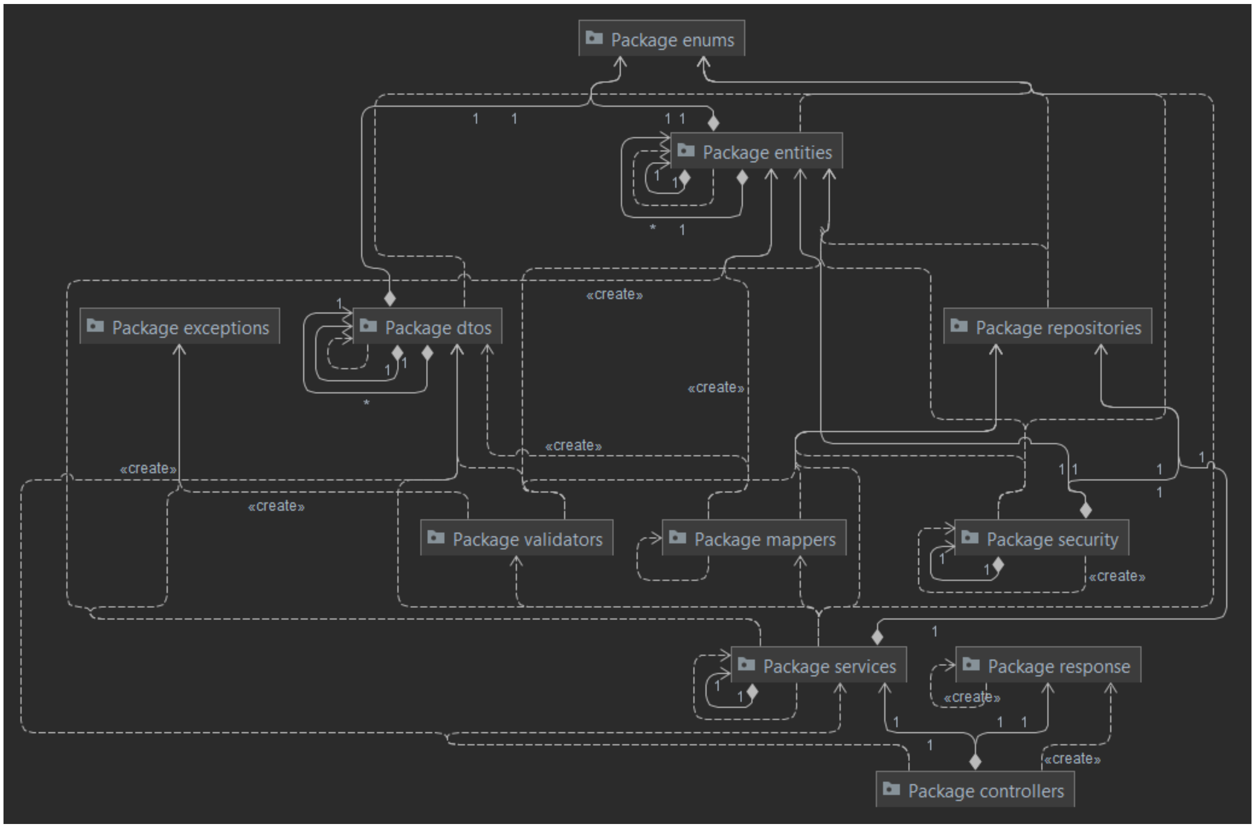
IEEE (Institute of Electrical and Electronics Engineers) defines architectural design as "the process of defining a collection of hardware and software components and their interfaces to establish the framework for the development of a computer system."

3.1 Conceptual architecture

The application is meant to be a Web Application, with a client-server interaction. The "client" will work with the front-end application developed in React, while the "sever" will reside in the back-end application developed in Java, using SpringBoot.

For such a client-server project, the best suited architecture, in my opinion, is the Layered Architecture, making it easy to define a flow from the client, through the server and up to the MySQL Database.

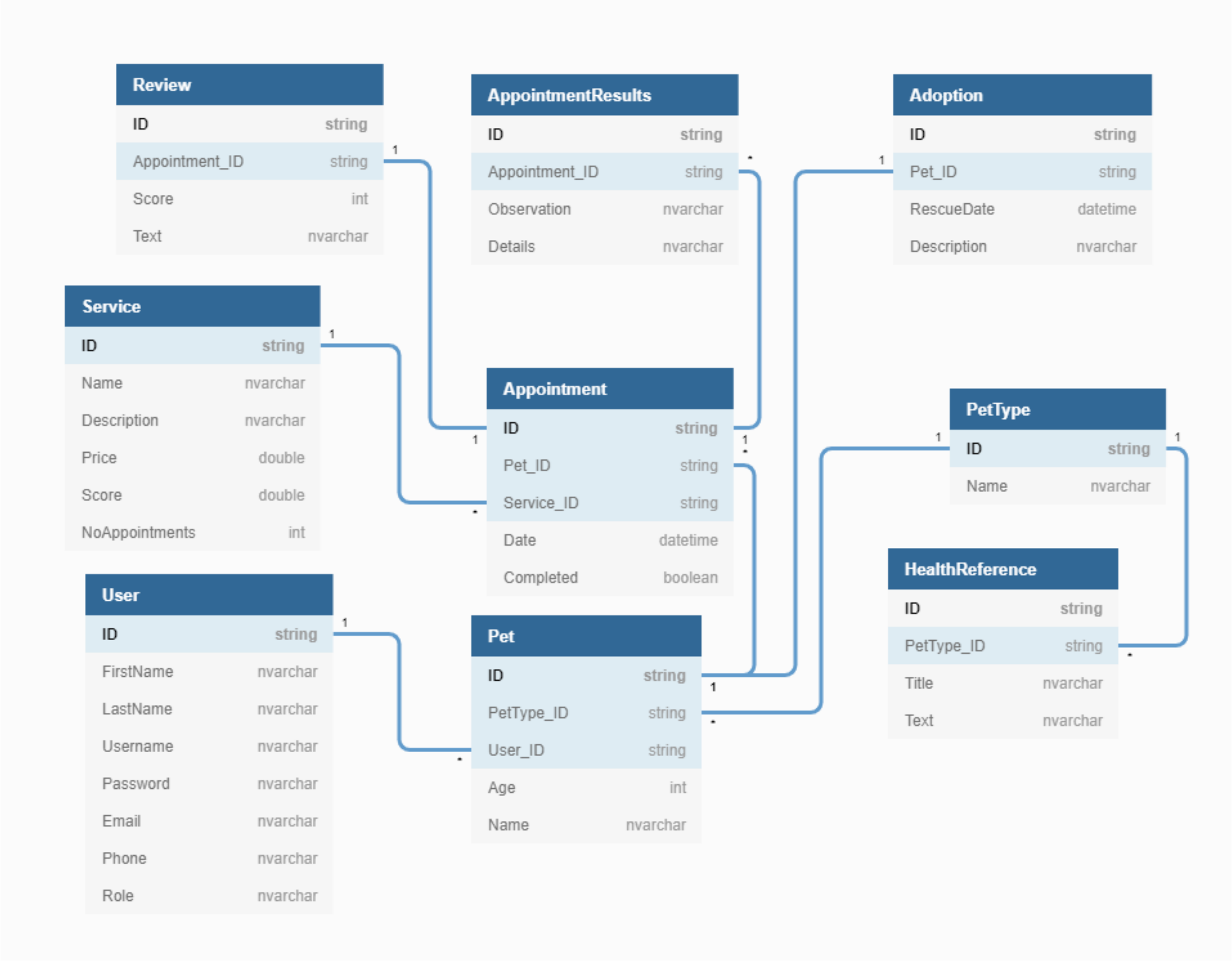
3.2 Package diagram



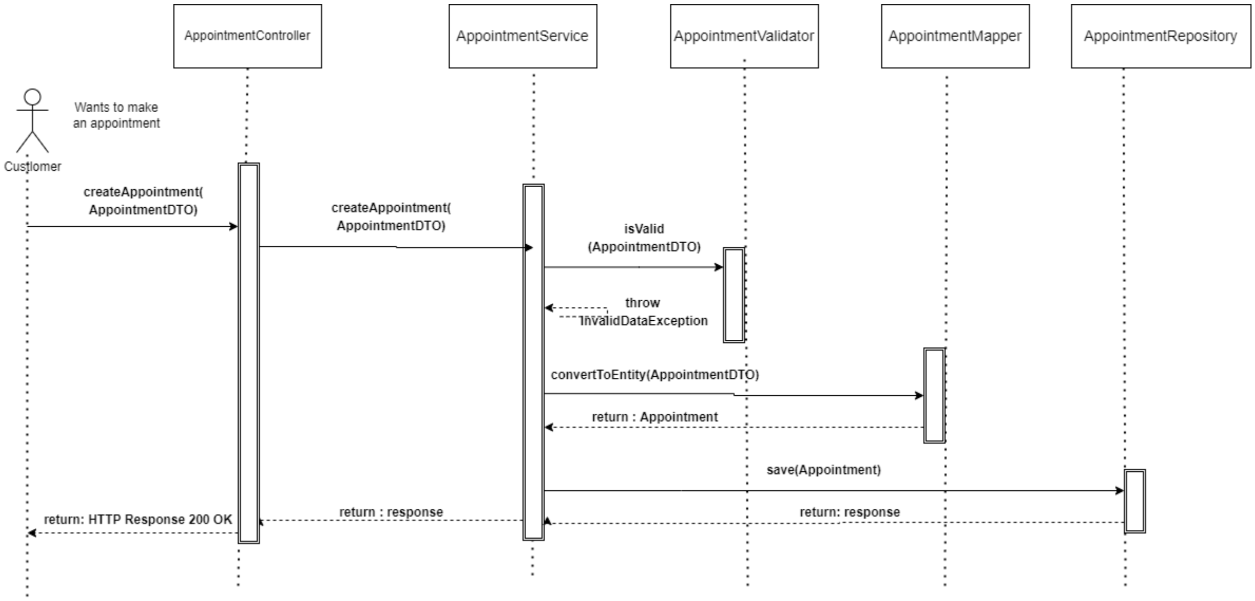
3.3 Class diagram

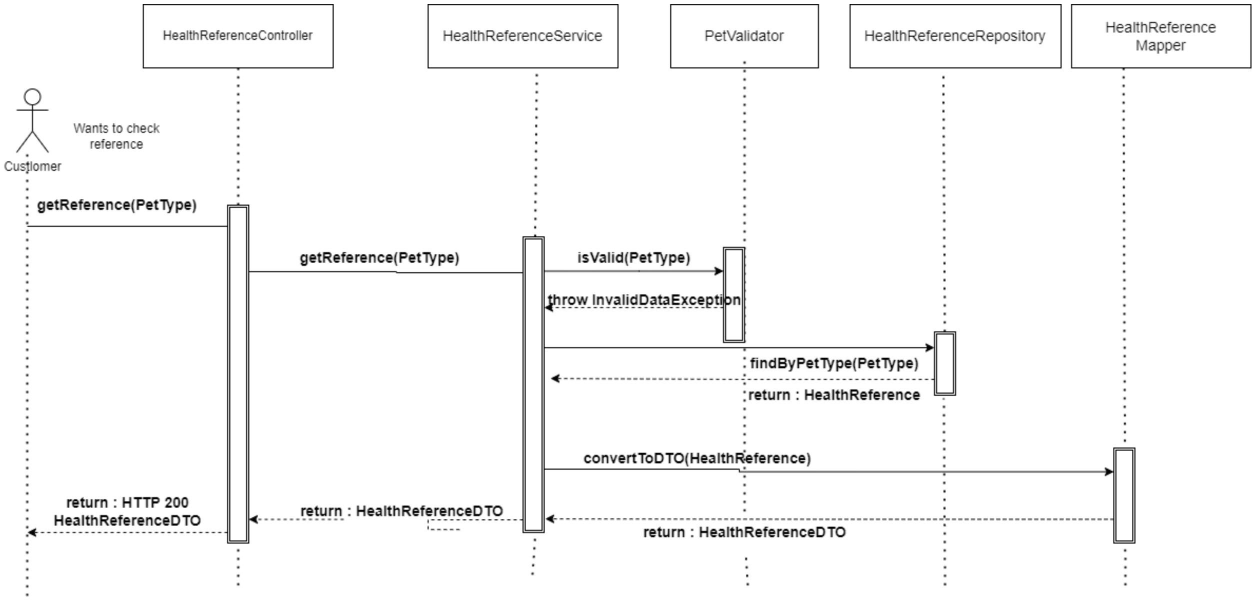
< (Class Diagram)/>

3.4 Database (E-R/Data model) diagram



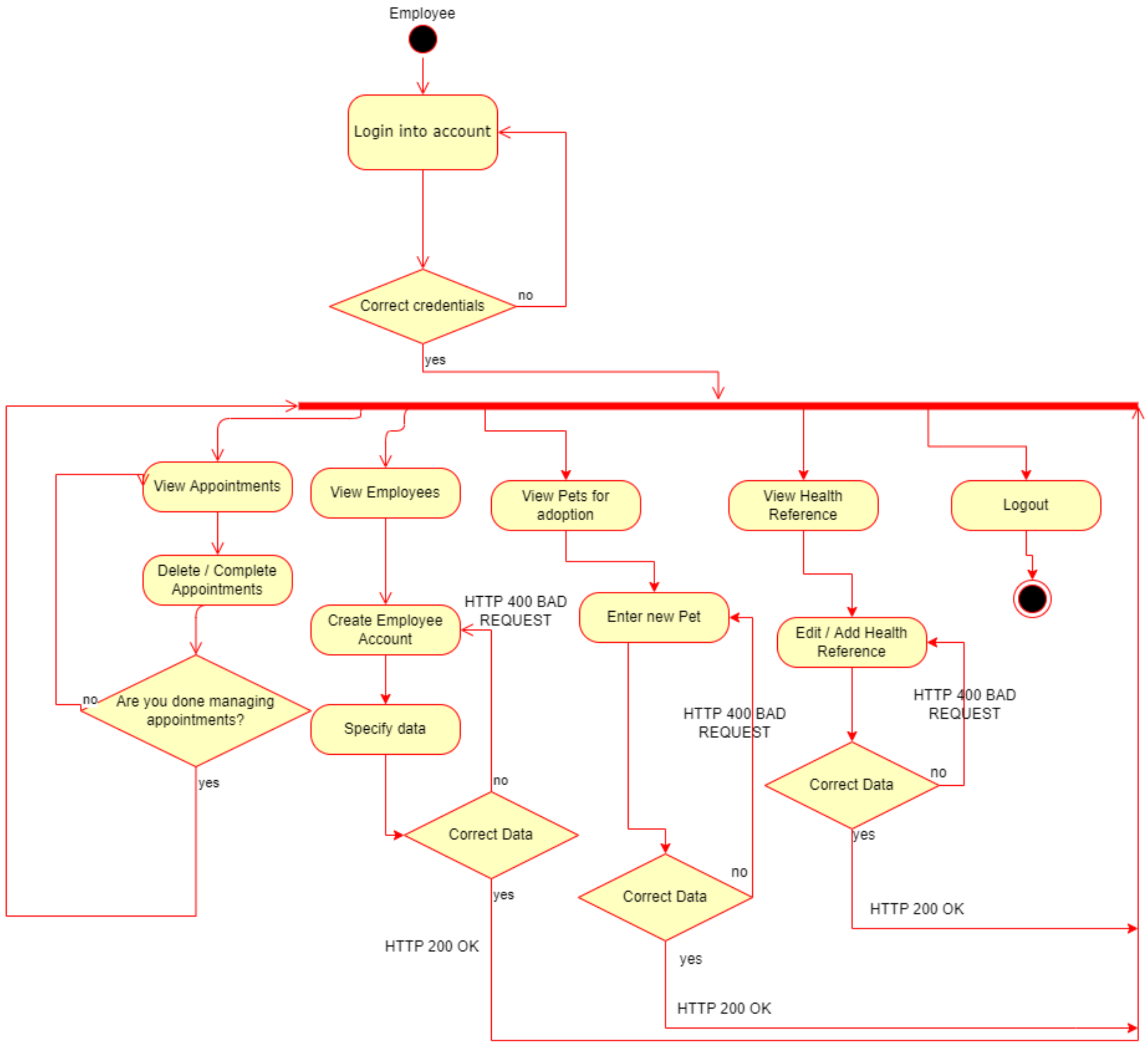
3.5 Sequence diagram





HealthReferenceController
HealthReferenceService
PetValidator
HealthReferenceRepository
HealthReference Mapper
Customer
Wants to check reference
getReference(PetType)
getReference(PetType)
isValid(PetType)
throw InvalidDataException
findByPetType(PetType)
return : HealthReference
convertToDTO(HealthReference)
return : HTTP 200 HealthReferenceDTO
return : HealthReferenceDTO
return : HealthReferenceDTO

3.6 Activity diagram



IV Supplementary specifications

The supplementary specifications will describe some of the details that we have to take into consideration while developing the application.

4.1 Non-functional requirements

- **Availability:**
  The application should be available to every user, no matter their role.
- **Security:**
  The user will be identified based on a JWT (JSON Web Token) and its password will be encrypted before it is stored in the database.
- **Scalability:**
  The application could easily be moved to the Cloud, for faster computation time, as well as it could integrate more endpoints / APIs if future features require it.
- **Accessibility Across Devices:**
  The application could be accessed from any browser, so it can be accessed from any device that can open a Web Browser.

4.2 Design constraints

- Languages required: Java, JavaScript, HTML, SCSS
- Frameworks required: SpringBoot, Hibernate, React, JPA
- Database requirements: MySQL Database, Relational Database Knowledge

V Testing

< Se va discuta la laborator./>

5.1 Testing methods/frameworks

5.2 Future improvements

VI Bibliography