Vetclinic Project

MAS subject
Yeva Vogau
s22354

Table of contents

User requirements 2
The use case diagram 3
The class diagram - analytical 4
The class diagram - design 5
The scenario of the use case 6
The activity diagram for picked use case 7
The state diagram for selected class 8
The interaction (sequence) diagram for selected use case 9
The GUI design 10
The discussion of design decisions and the effect of dynamic analysis 12
- Overlapping. Dynamic. 12
- Complex attribute. 13
- Association with an attribute. 14
- Multi-inheritance. 14
- Multi-aspect. 15
- State 16
- Composition. 17
- XOR. 18

User requirements

The system manages all the information and database of the many Vet clinics. It helps employees to make changes or view the information, also the owners of the pets can have an account and see all information about their pets.

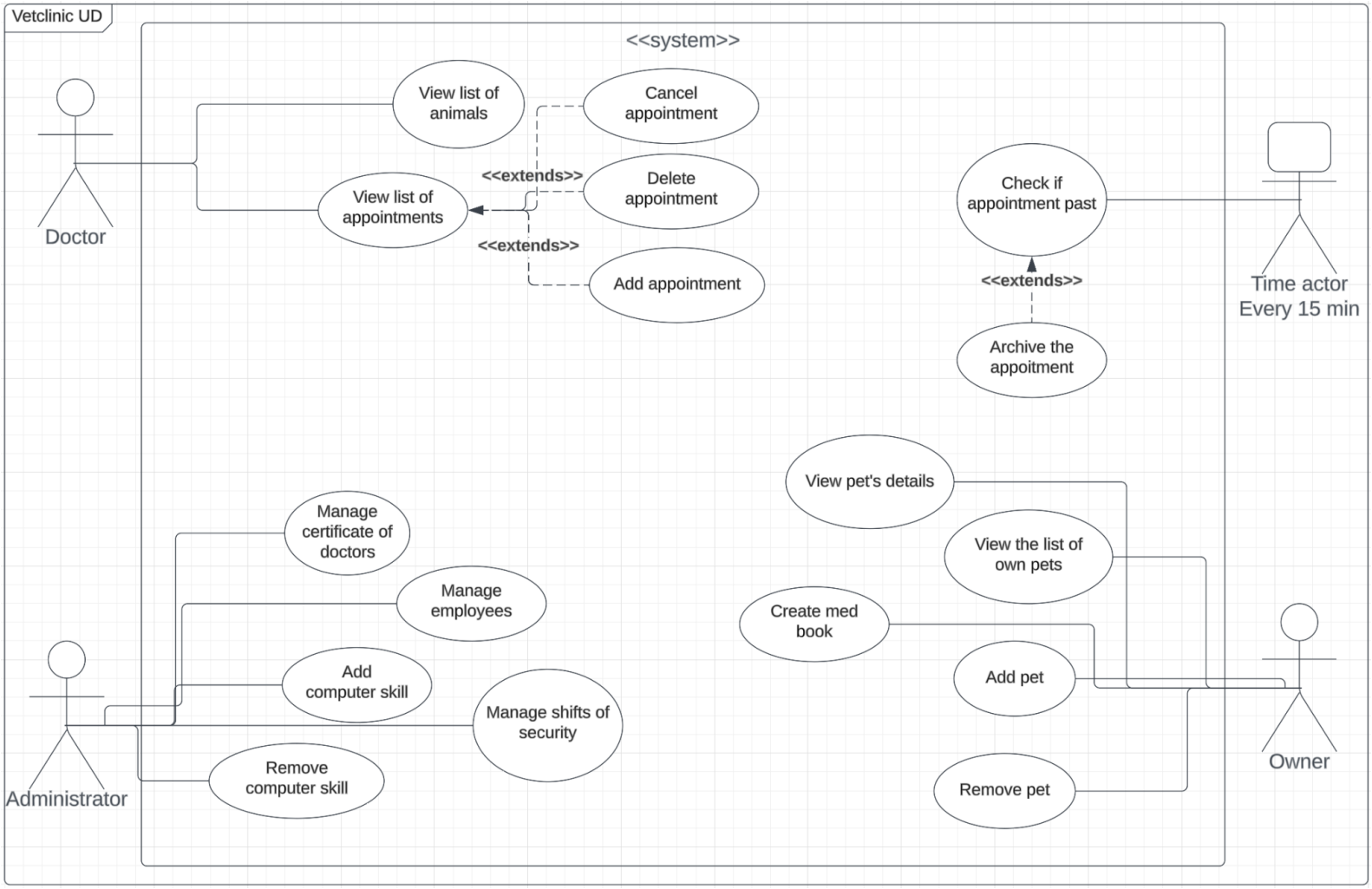
The Vet clinic object consists of basic information (name, address, sos number) and additional optional rating. The address is a complex attribute , with street and number.

In this clinic work the employees, who have basic attributes such as name, surname and in Employee additionally add employment date, monthly salary. Employees have different types: Doctor, Administrator, Security. The types can be changed by time and be two roles at the same time. All of them have specific fields for their roles. Security has shift, the Administrator has a set of computer skills and Doctors, of course, should have a set of certificates. An employee can be a pet owner and has the discount card in addition to the usual client.

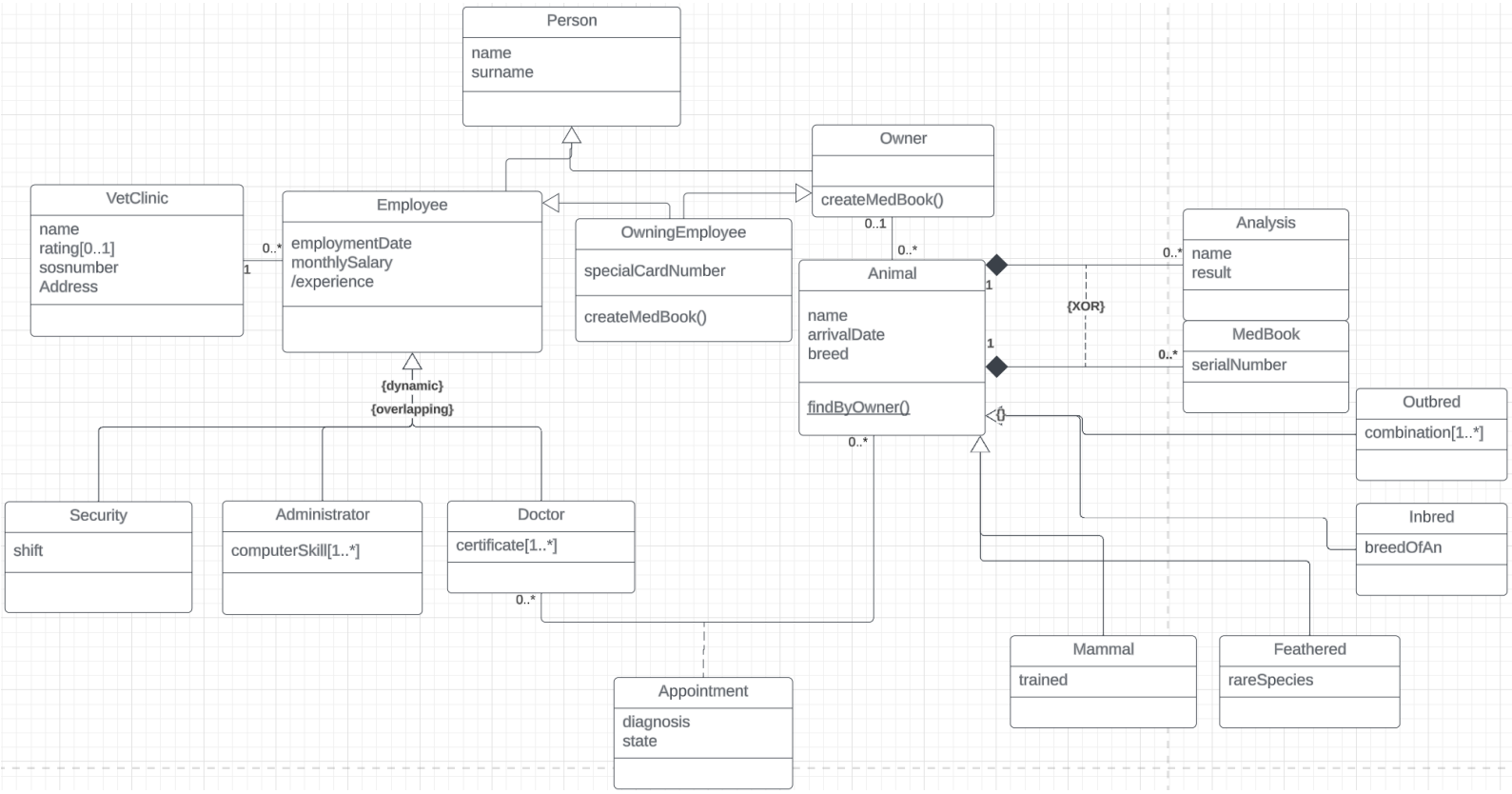
The clients of this clinic are owners of the animals, and if the animal is abandoned, its data is collected in the doctor's database and this doctor is responsible for it, without the owners. The class Owner can have a connection with an Animal class as “pets”. Animals are distinguished by unique ids. Animal is a class with a name, arrival date, breed (Outbred or Inbred). Outbred animals have the list of combinations and Inbred has the concrete type of the animal. The user can filter Animals by the owner. Two classes are inherited from abstract class Animal, these are Mammals and Feathered. Mammals should have a boolean field trained and Feathered should have a boolean field rear species.
Depending on the presence of the medical book, the animal can have analises. But if the pet already has a MedBook, then he cannot have Analises, because the results are already written in the medical book. And otherwise, medical books cannot be created, while analyses are in the system. Analisis has name and result, MedBook has serial number.

The clinic works by the system of Appointments between the Animal and Doctor. In the class Appointment the field diagnosis should be written.

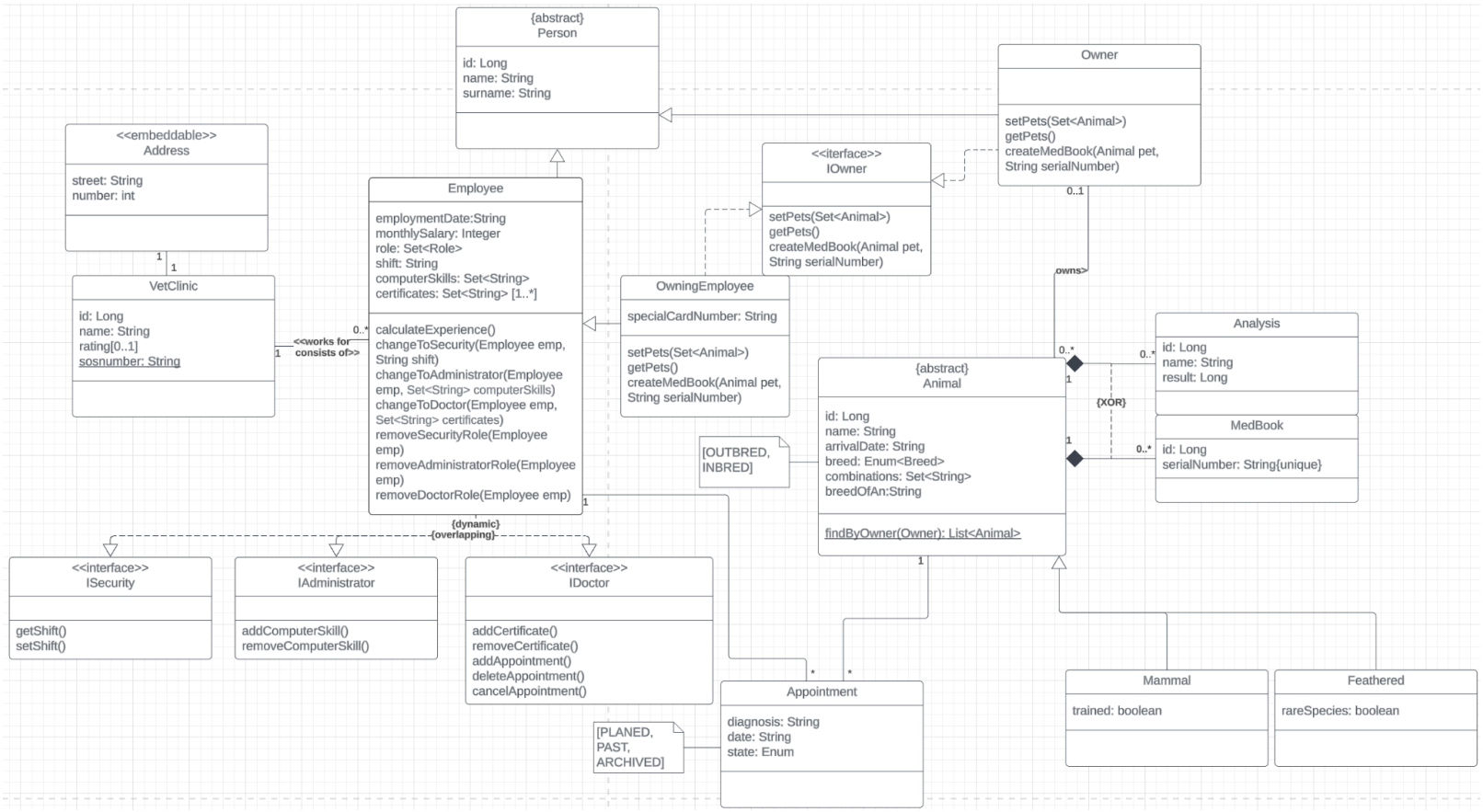
The use case diagram



The class diagram - analytical



The class diagram - design



The scenario of the use case

Use case: Delete an appointment
Actor: Doctor
Pre-condition: Doctor should log in into the system and doctor has animals and appointments
Basic flow of the event:

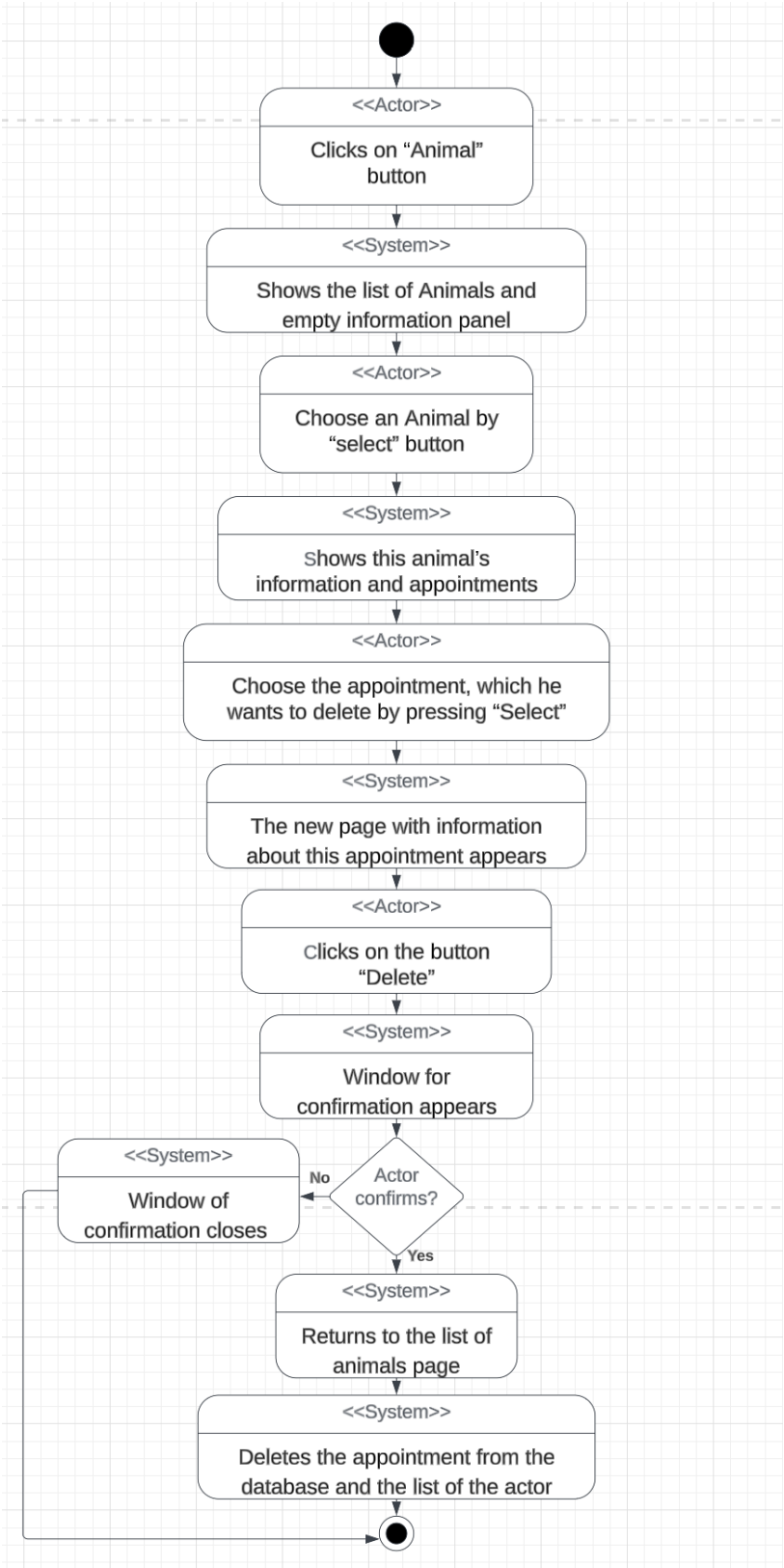
1. The actor clicks on “Animals” button
2. The system shows the list of Animals and empty information panel
3. The actor choose an Animal by “select” button
4. The system shows this animal’s information and appointments
5. The actor choose the appointment, which he wants to delete by pressing “Select”
6. The new page with information about this appointment appears
7. The actor clicks on the button “Delete”
8. The window for confirmation appears
9. The actor clicks on the button “Confirm”
10. The system returns to the list of animals page

Alternative path:

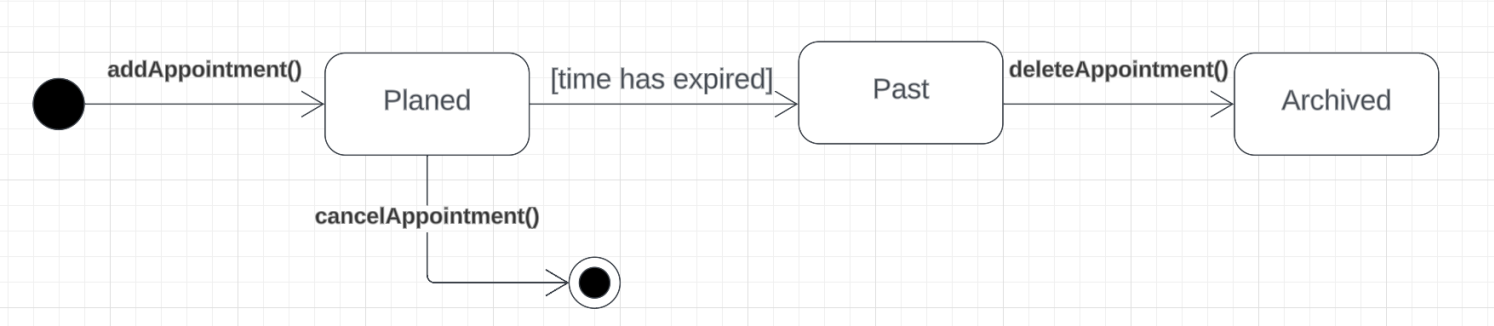
6. A. The actor clicks on the “Cancel” button
   B. The window of confirmation closes

Post-condition: The system deletes the appointment from the database and the list of the actor.

The activity diagram for picked use case

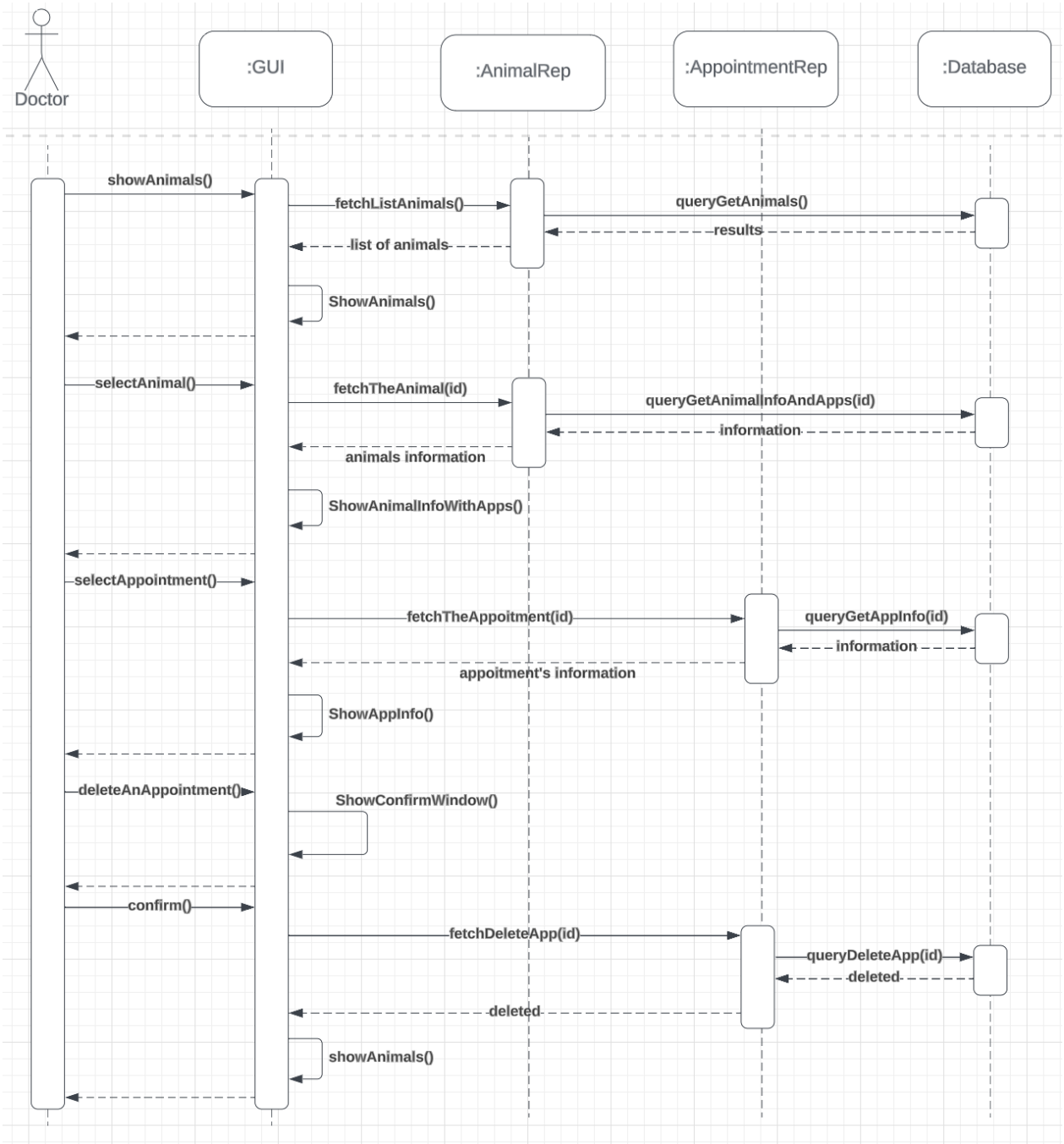


The state diagram for selected class

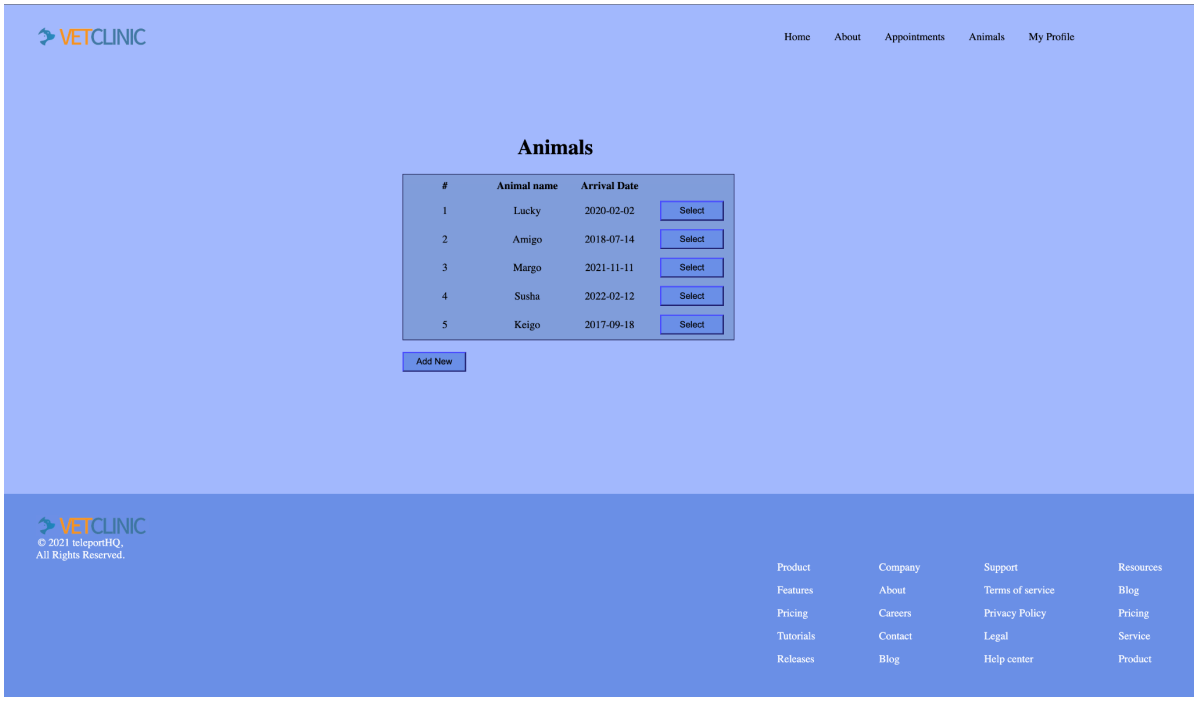


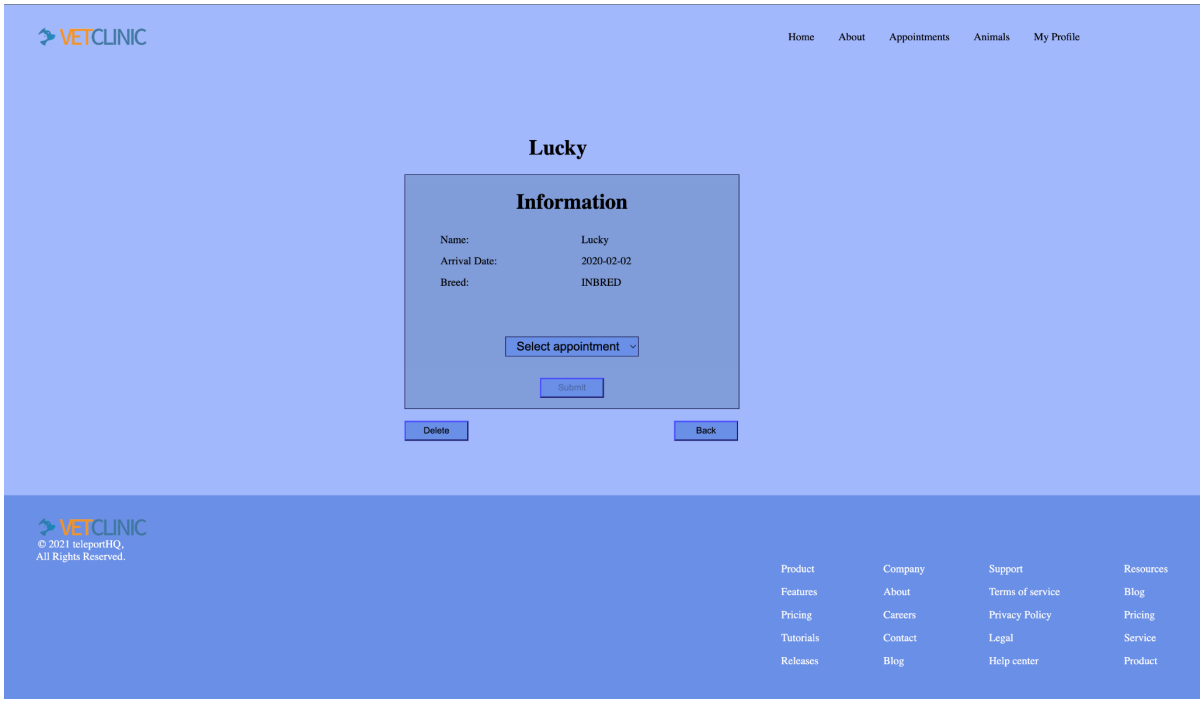
The interaction (sequence) diagram for selected use case

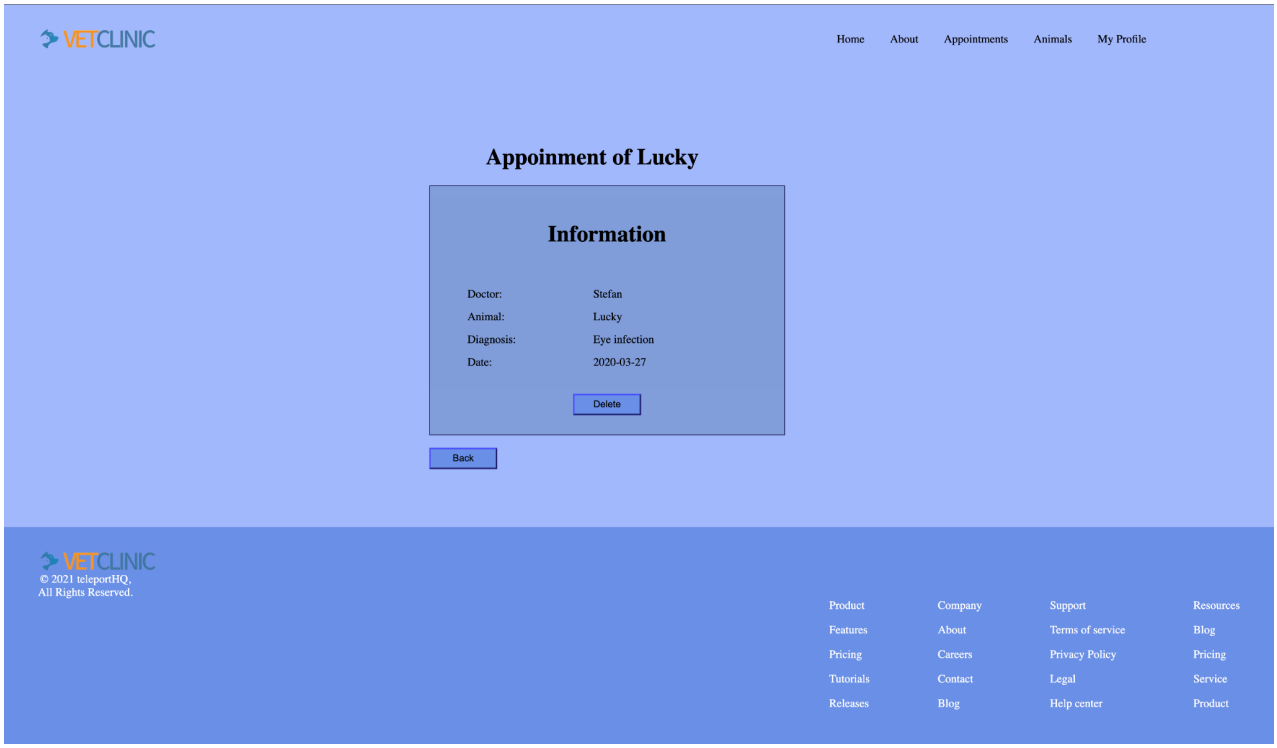
Use case: Remove an appointment
Actor: Doctor

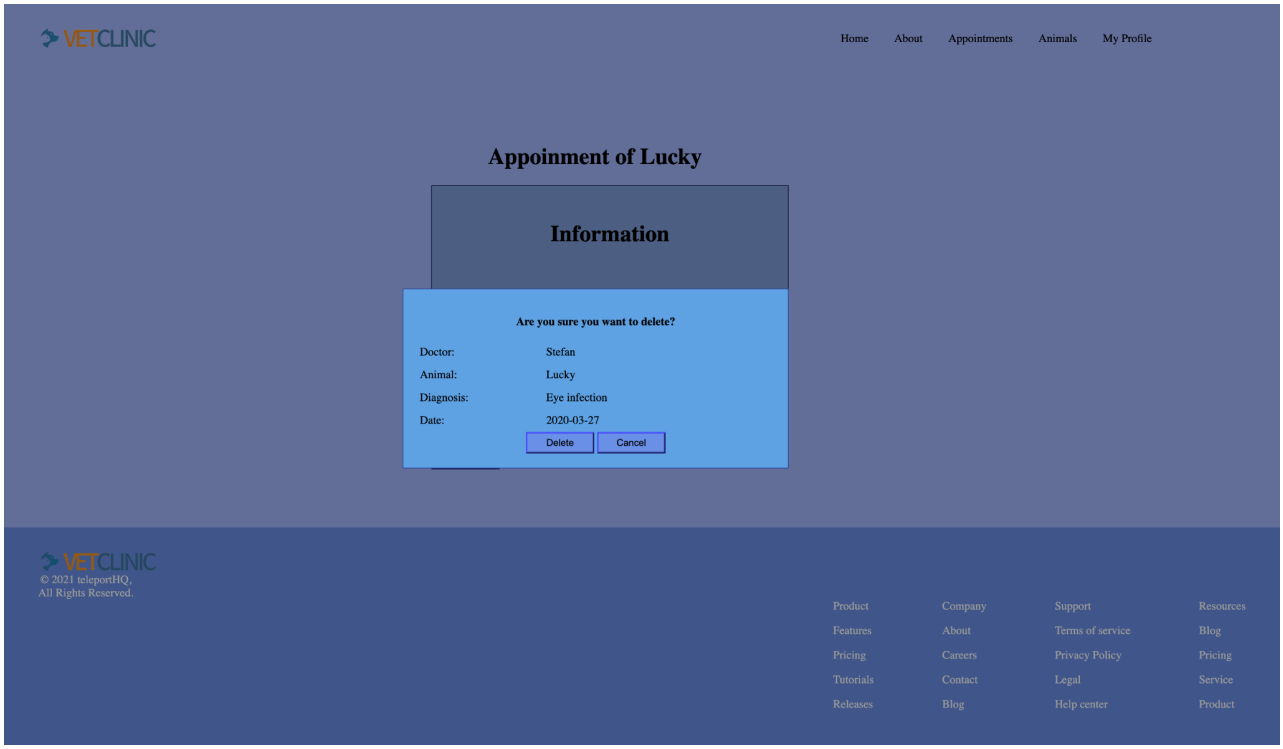


The GUI design






VETCLINIC
Home
About
Appointments
Animals
My Profile
Appoinment of Lucky
Information
Doctor:
Stefan
Animal:
Lucky
Diagnosis:
Eye infection
Date:
2020-03-27
Delete
Back
VETCLINIC
© 2021 teleportHQ,
All Rights Reserved.
Product
Features
Pricing
Tutorials
Releases
Company
About
Careers
Contact
Blog
Support
Terms of service
Privacy Policy
Legal
Help center
Resources
Blog
Pricing
Service
Product


VETCLINIC
Home
About
Appointments
Animals
My Profile
Appoinment of Lucky
Information
Are you sure you want to delete?
Doctor:
Stefan
Animal:
Lucky
Diagnosis:
Eye infection
Date:
2020-03-27
Delete
Cancel
VETCLINIC
© 2021 teleportHQ,
All Rights Reserved.
Product
Features
Pricing
Tutorials
Releases
Company
About
Careers
Contact
Blog
Support
Terms of service
Privacy Policy
Legal
Help center
Resources
Blog
Pricing
Service
Product

The discussion of design decisions and the effect of dynamic analysis

In this project I used the MVC (Model-View-Controller) architectural pattern.

I defined entities by annotating them with Hibernate annotations to specify mappings, relationships, and constraints.

I created repositories - interfaces or classes that define data access operations. Use Spring Data JPA, which integrates with Hibernate, to create repositories easily. These repositories will handle CRUD (Create, Read, Update, Delete) operations and other custom queries.

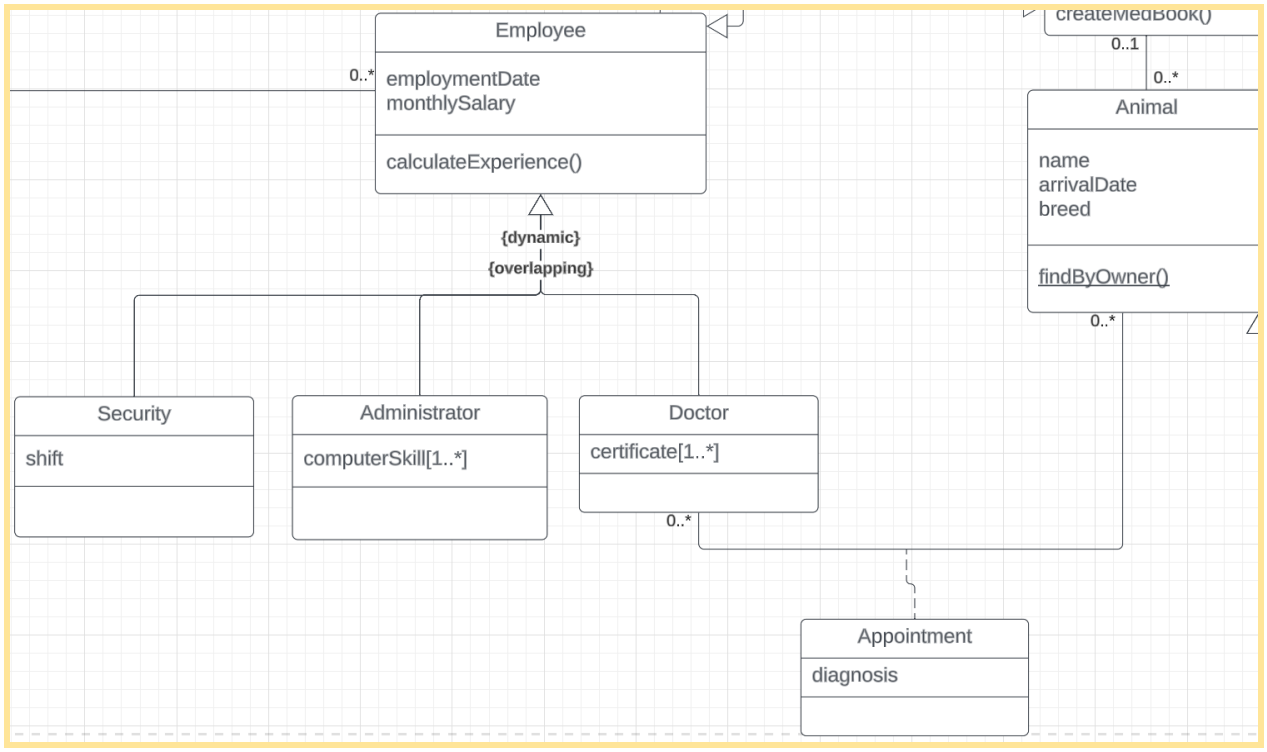
Services interact with the repositories to perform database operations and any additional processing required.

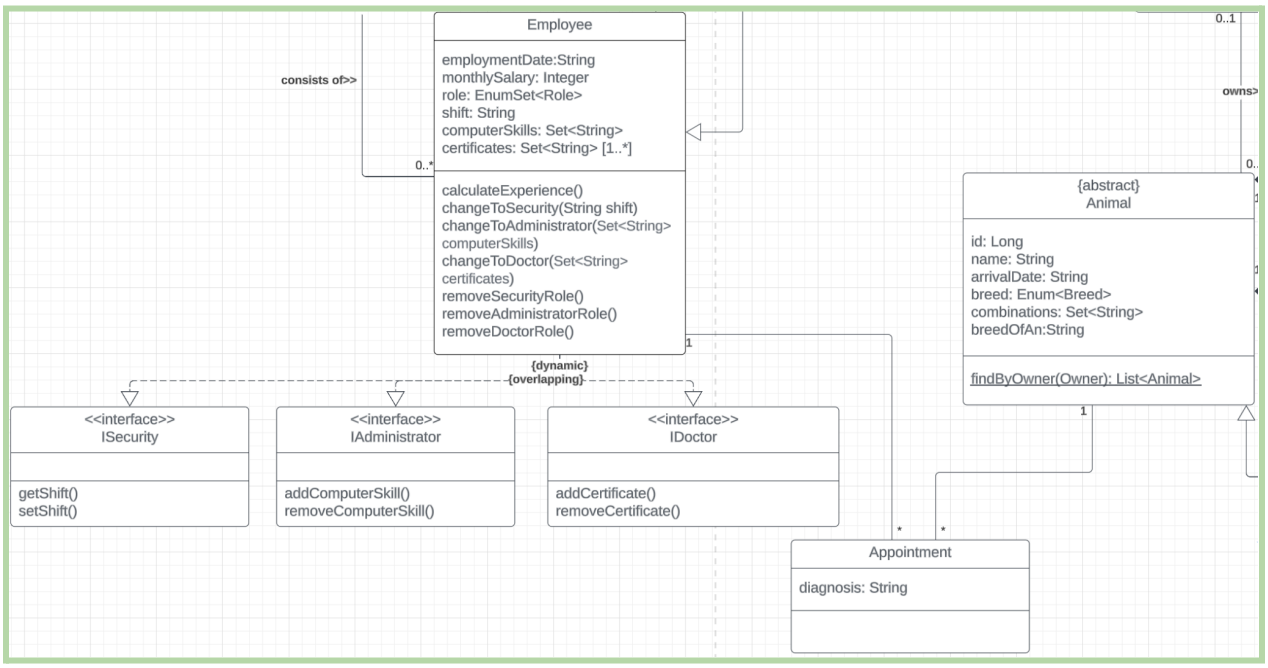
Controller classes to map HTTP requests to appropriate service methods. Use annotations like @Controller and @RequestMapping to define the routes and handle various HTTP methods.

Develop GUI with Thymeleaf. I used Thymeleaf expressions to bind data from the backend to the templates.

Overlapping. Dynamic.

To implement the overlapping and dynamic structure, I transformed sub-classes (Security, Administrator, Doctor) to interfaces. Also implemented all classes' attributes, EnumSet into the class (Employee). For overlapping, EnumSet should contain roles of an employee and for dynamics, we use methods to change the role by deleting old ones and adding a new one with attributes respectively.. Because Doctor class had a connection with an Appointment, but Doctor now is an interface, I connected an Appointment with an Employee and will have a checking method for a Doctor role.

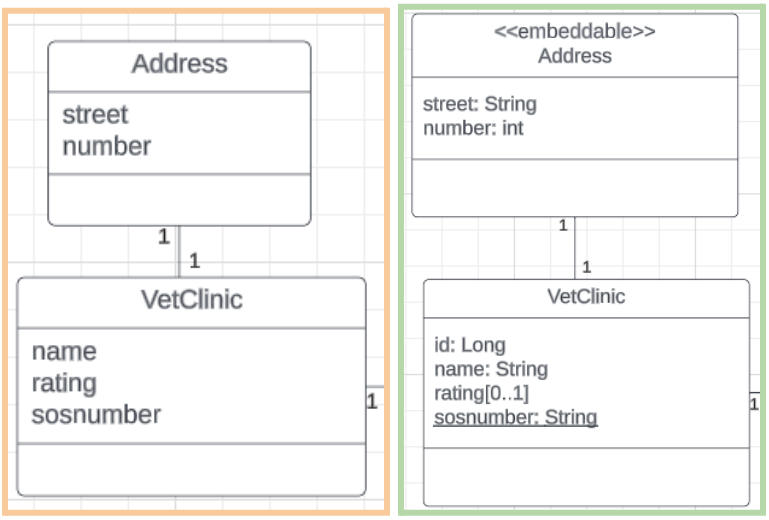




Complex attribute.

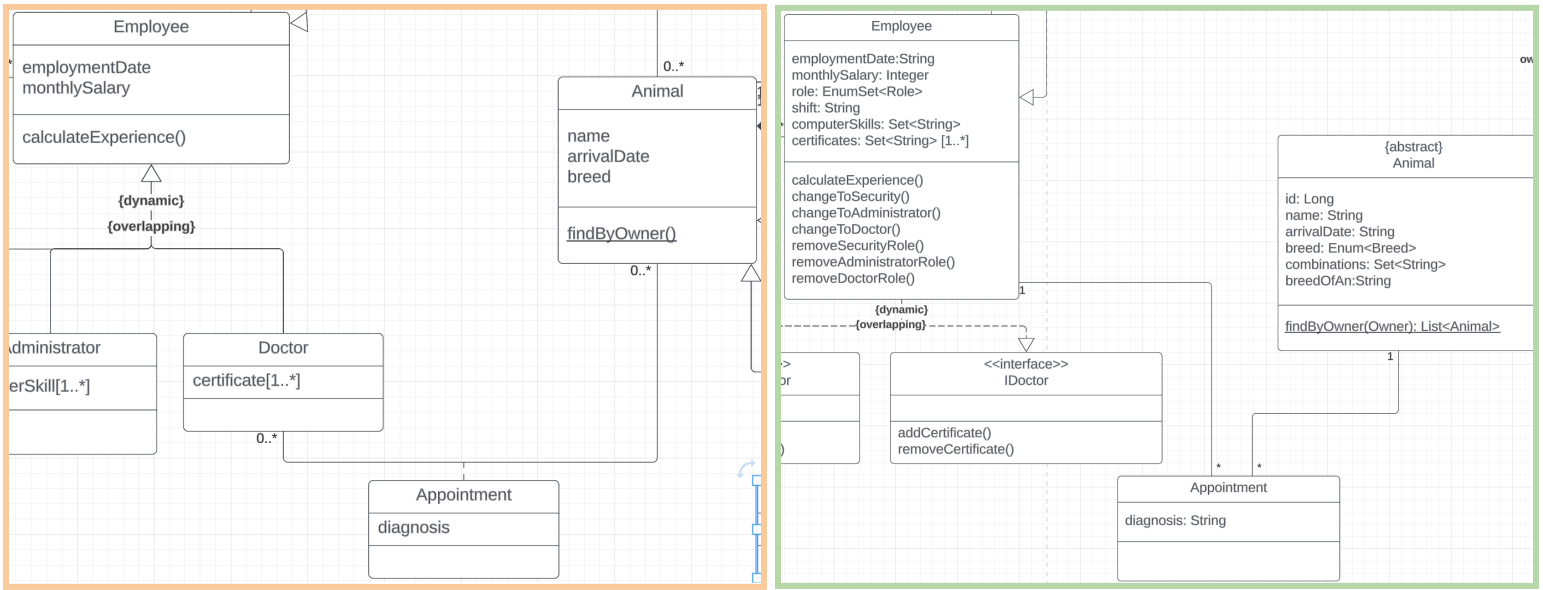
The attributes of VetClinic class, which describe the address of the building it is located in, I embed them into another entity (Address) by using the annotation

@Embedded. As a result, I have an entity VetClinic, embedding address details and mapping to a single database table.



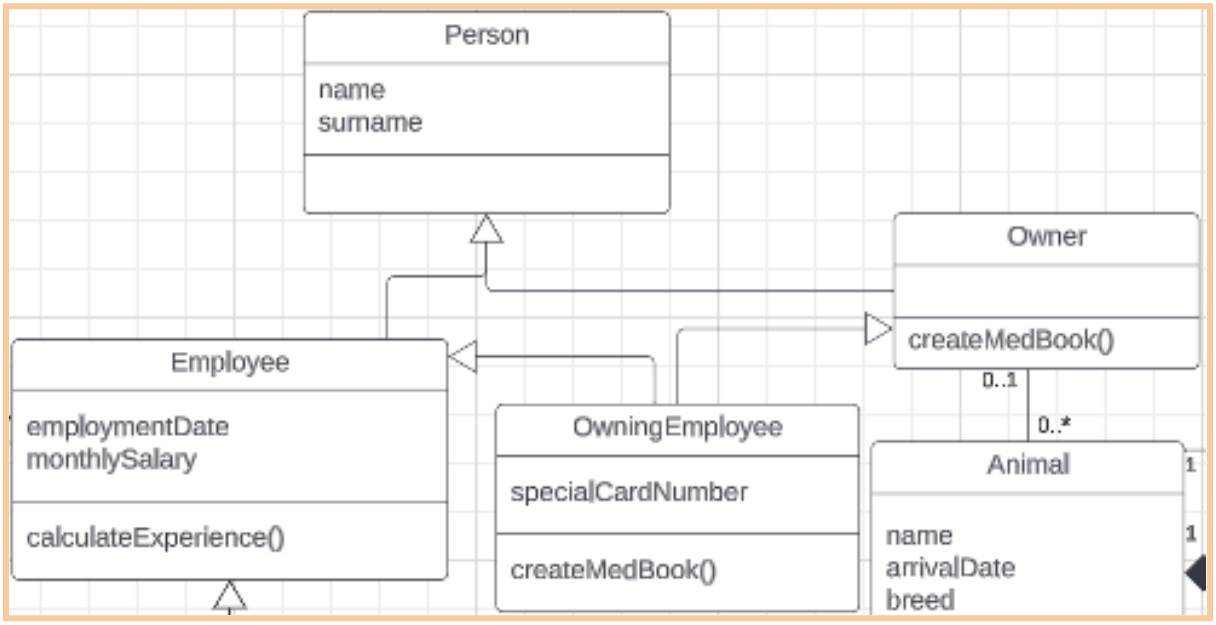
Association with an attribute.

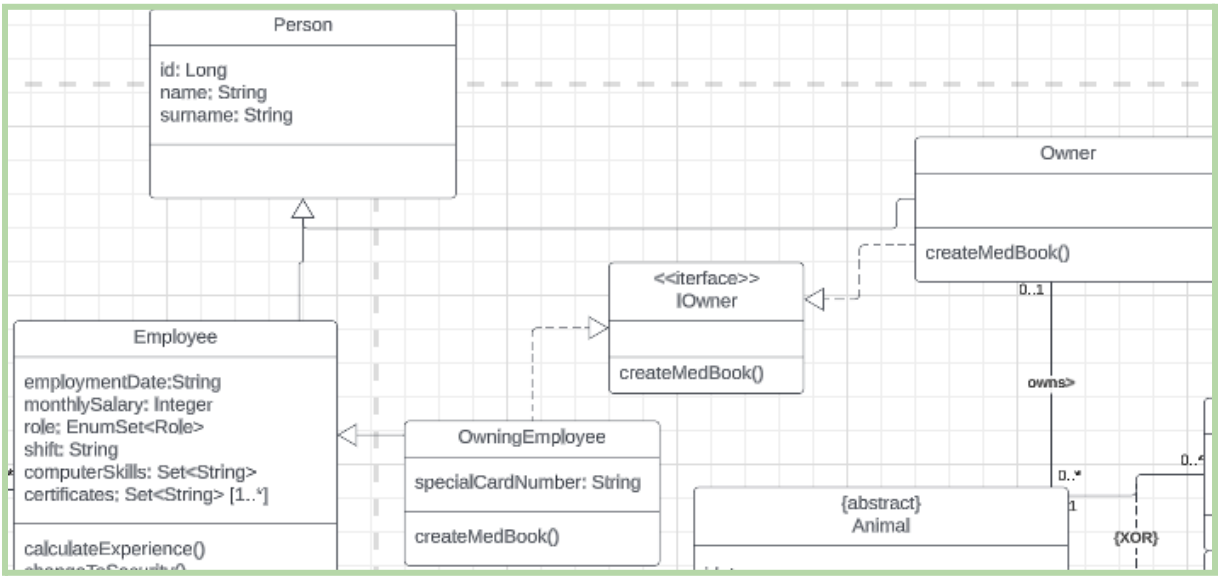
The Appointment entity represents the many-to-many relationship between the Doctor and Animal entities and has an attribute - diagnosis. Because in the Design diagram the Doctor class is an interface, I connect the Appointment class to an Employee class with checking the role Doctor. I connect Appointment with classes by @ManyToOne and @JoinColumn annotations and accordingly connect Appointment by @OneToMany annotation in the Employee and Animal classes.



Multi-inheritance.

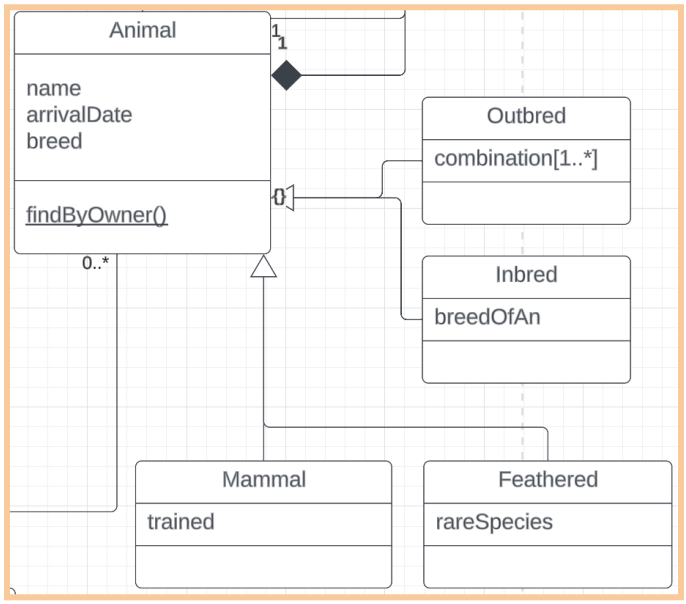
Class OwningEmployee is inherited from the Employee and Owner classes, which are inherited from class Person. Because the class cannot extend two classes in Java, we use implementation of an interface IOwner and extension of the Employee class. Class Owner also implements this interface. This approach covers all our needs: all methods and attributes are transferred to an OwningEmloyee.

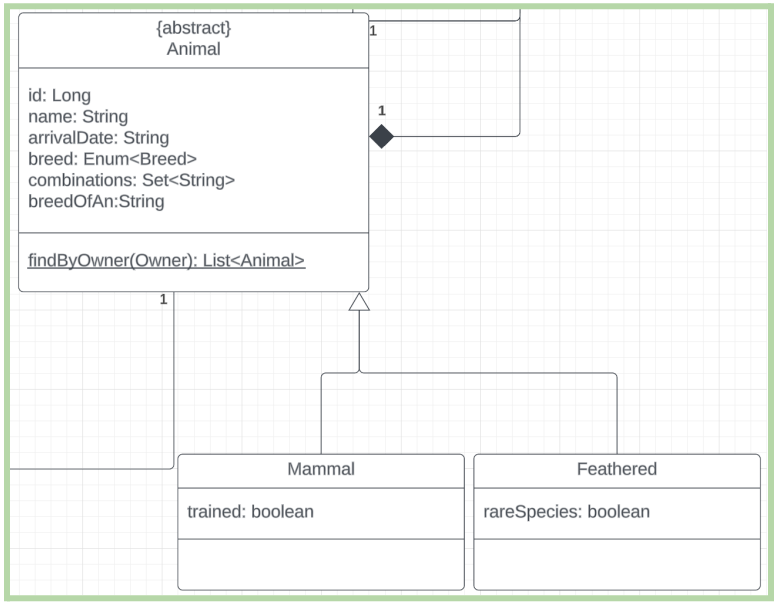




Multi-aspect.

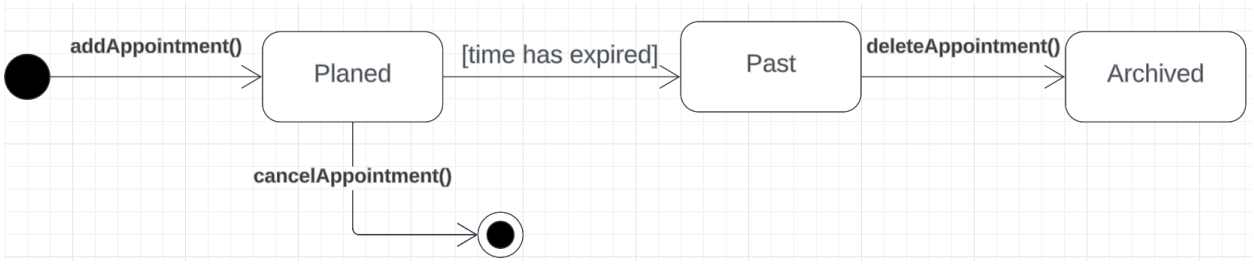
Classes Outbred and Inbred are one aspect of the abstract class Animal and classes Mammal and Feathered are inherited from the same class Animal. To implement the multi-aspect structure, I created an enum Breed as a field and put the attributes of the aspect classes in the class Animal. Depending on the enum type, I assign the right attributes.

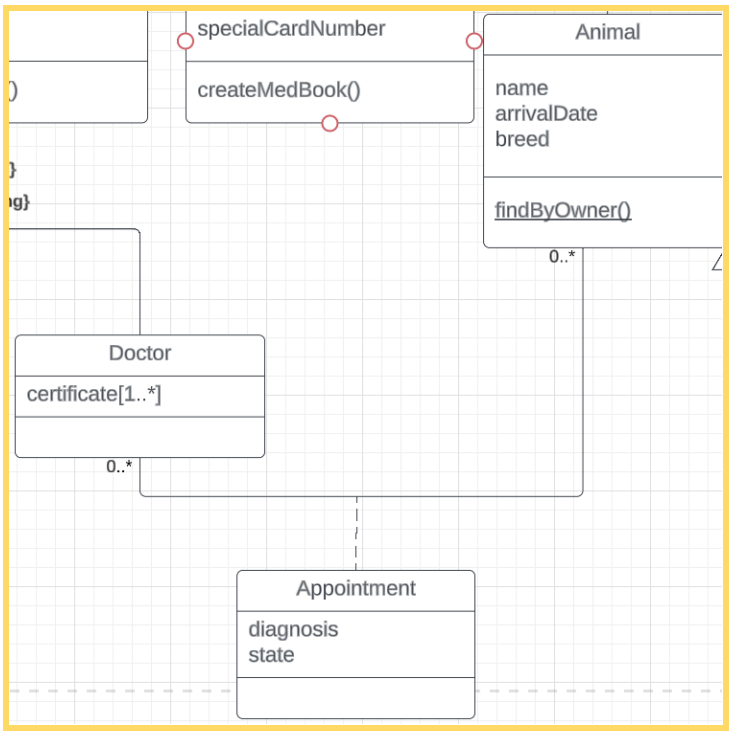


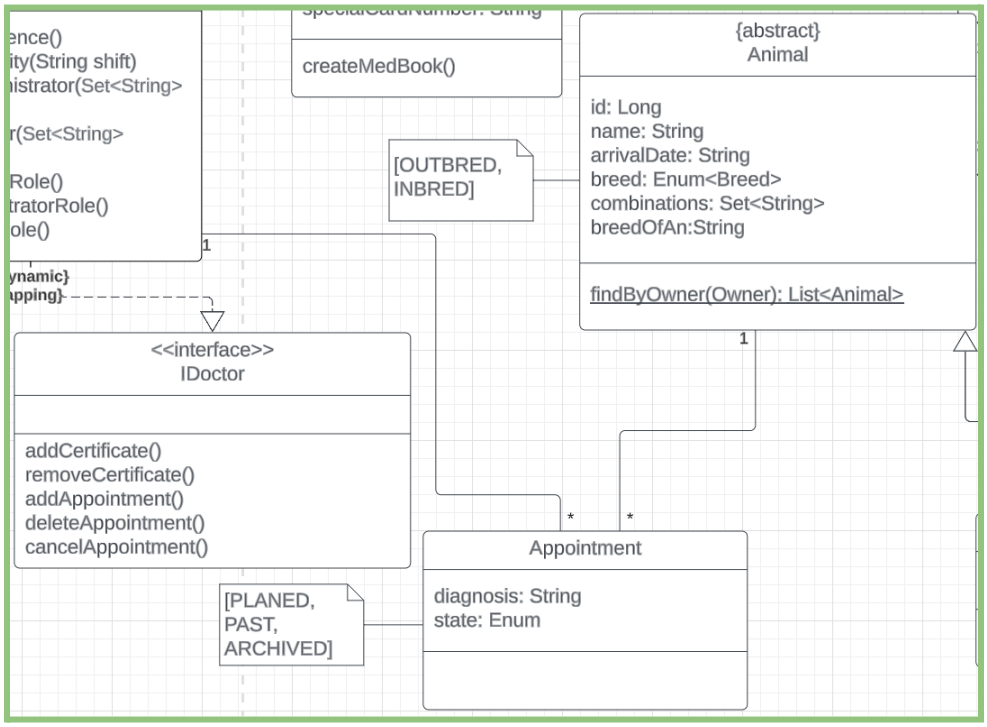


State

The Appointment class has a state attribute described as enum (Planed, Past, Archived), which corresponds to a state of an object depending on the methods called by the object of the class Doctor. Doctor has methods addAppointment(), deleteAppointment() and cancelAppointment(). I described the flow in the state diagram.



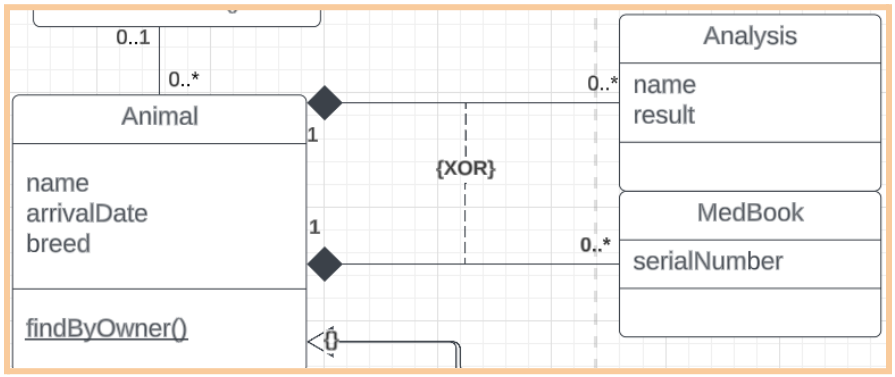


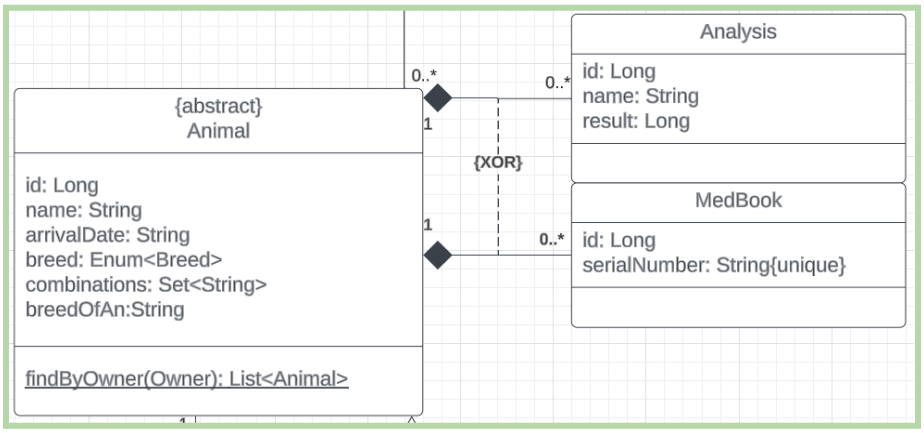


Composition.

The class MedBook connected to an Animal by the composition one-to-many association, that means the object of this class cannot exist without the Animal. I

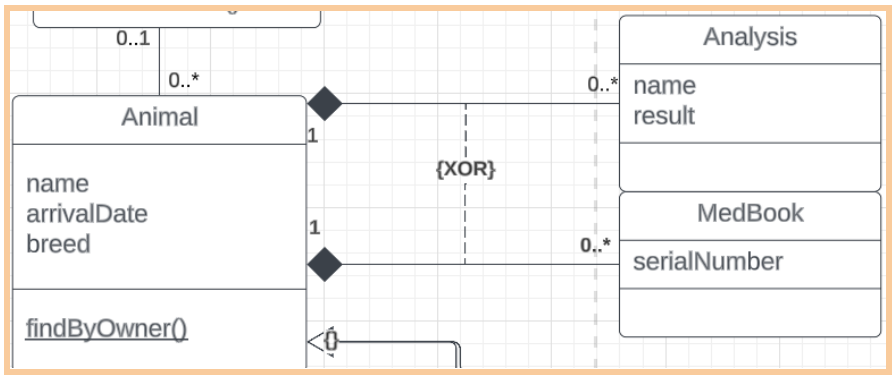
implemented this structure by adding the annotations: @OneToMany (Animal), @ManyToOne (MedBook). Important attribute to an annotation @OneToMany is cascade = CascadeType.REMOVE, that means that if object Animal were deleted, all MedBooks connected to it also will be deleted/removed. The same structure I have with connection Animal-Analysis.

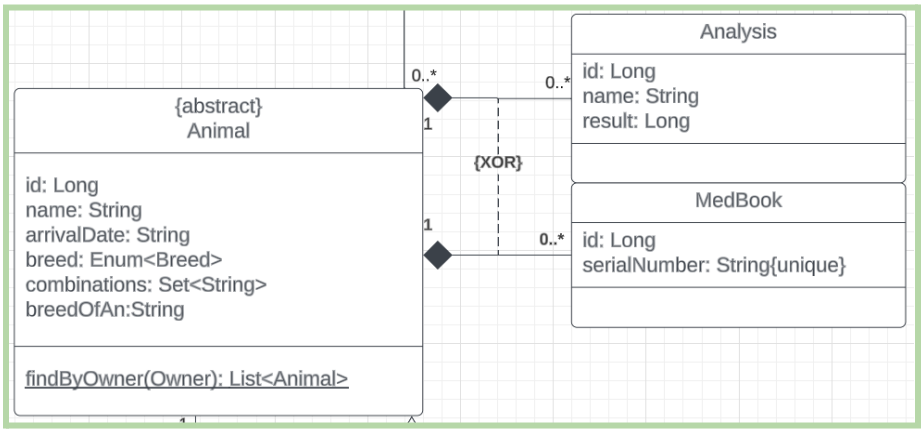




XOR.

The class Animal has two excluded associations with Analysis and MedBook. These two classes cannot exist at the same time and cannot be created when another one exists. I implement such an approach by the validation before creating an object.


0..1
0..*
Animal
name
arrivalDate
breed
findByOwner()
1
1
{XOR}
0..*
0..*
Analysis
name
result
MedBook
serialNumber


{abstract}
Animal
id: Long
name: String
arrivalDate: String
breed: Enum<Breed>
combinations: Set<String>
breedOfAn:String
findByOwner(Owner): List<Animal>
0..*
1
1
{XOR}
0..*
0..*
Analysis
id: Long
name: String
result: Long
MedBook
id: Long
serialNumber: String{unique}