Black Dog – Assignment 2

Student: Victor Pădurean

**Group: 30434**

Table of Contents

1. Requirements Analysis 3

1.1 Assignment Specification 3

1.2 Functional Requirements 3

1.3 Non-functional Requirements 3

2. Use-Case Model 3

3. System Architectural Design 3

4. UML Sequence Diagrams 3

5. Class Design 3

6. Data Model 3

7. System Testing 3

8. Bibliography 3

1. Requirements Analysis

# Assignment Specification

Design and implement a desktop application for a Veterinary Lab. The application should have two types of users (a regular user - the doctor/employee - and an administrator user) which must provide a username and a password in order to use the application.

# Functional Requirements

The regular user can perform the following operations:

* CRUD on consultations
* CRUD on animals involved in consultations
* Search the list of consultations/animals

The administrator can perform the following operations:

* CRUD on animals
* CRUD on regular users' information.
* Generate two types of report files, one in pdf format and one in txt or html format, with the consultation’s details. The reports \*\*need\*\* to be saved in a \*\*user-selected location\*\* (not predefined by the application), similar on how one would save a file from Notepad.

# Non-functional Requirements

* Use Java or C#.
* The data used in the application will be stored in a database, using an ORM (Object Relational Mapping) framework.
* Use the Model-View-Controller (or Model-View-ViewModel if using WPF) in designing the application. It's \*\*mandatory\*\* that the Controllers/ViewModels don't have direct reference to the View class (e.g., the one that inherits from JFrame or JPanel), but through an interface if really needed.
* Use the Factory Method design pattern for generating the reports (i.e., selecting between the type of report to generate).
* All the inputs of the application will be validated against invalid data before submitting the data and saving it.
* Unit test the \*\*users-related\*\* controller(s)/view model(s) using a testing framework (e.g., JUnit, NUnit). Use a Mocking framework (e.g., Mockito, Moq) to mock dependencies on repositories/views. The tests \*\*should not\*\* need nor have direct access to the database.

2. Use-Case Model

A close up of text on a white background

Description automatically generated

Use case: Create consultation

Level: User-goal level

Primary actor: Regular User

Main success scenario:

* The user selects the corresponding doctor from a list of existing ones
* The user selects the corresponding patient from a list of existing ones
* Additional details are provided by the user
* The create button is hit
* As a result, a new consultation with the corresponding details is created and added to the database, the list of consultations being updated

Extensions: if the fields are supplied with corrupted data, the user will be notified through a popup

3. System Architectural Design

**3.1 Architectural Pattern Description**

The used architectural pattern will be the Model-View-Controller pattern. This is done using the help of JavaFX. Yet, the project is run using Spring, so FX Weaver was used to integrate Java FX with Spring.

The Model-View-Controller pattern has three main components, as its name describes: the view, which handles the visual appearance of the GUI; the controller, which handles the user’s commands and application logic; and the model, which stores the states and data. For this project, the controllers will delegate the application logic to service methods, so that they will not become bloated.

**3.2 Diagrams**

*A screenshot of a cell phone

Description automatically generated*

A screenshot of a cell phone

Description automatically generatedA close up of a map

Description automatically generated

4. UML Sequence Diagrams

*A screenshot of a cell phone

Description automatically generated*

5. Class Design

**5.1 Design Patterns Description**

The only used design pattern (besides those that Spring and Lombok bring) is the Factory Design Pattern, used for managing report creation. It can clearly be seen in the UML class diagram that the ConsultationService is not responsible with the details of report creation, handling only an interface and the factory class. This class is invoked for creating the necessary report type accordingly.

**5.2 UML Class Diagram**

*A screenshot of a cell phone

Description automatically generated*

6. Data Model

*A screenshot of a cell phone

Description automatically generated*

7. System Testing

*[Present the used testing strategies (unit testing, integration testing, validation testing) and testing methods (data-flow, partitioning, boundary analysis, etc.).]*

8. Bibliography