Analysis and Design Document

Student:Iancu Daiana-Sorana

**Group:30431**

**Animal Shelter**

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| <dd/mmm/yy> | <x.x> | <details> | <name> |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

I. Project Specification 4

II. Elaboration – Iteration 1.1 4

1. Domain Model 4

2. Architectural Design 4

2.1 Conceptual Architecture 4

2.2 Package Design 4

2.3 Component and Deployment Diagrams 4

III. Elaboration – Iteration 1.2 4

1. Design Model 4

1.1 Dynamic Behavior 4

1.2 Class Design 4

2. Data Model 4

3. Unit Testing 4

IV. Elaboration – Iteration 2 4

1. Architectural Design Refinement 4

2. Design Model Refinement 4

V. Construction and Transition 5

1. System Testing 5

2. Future improvements 5

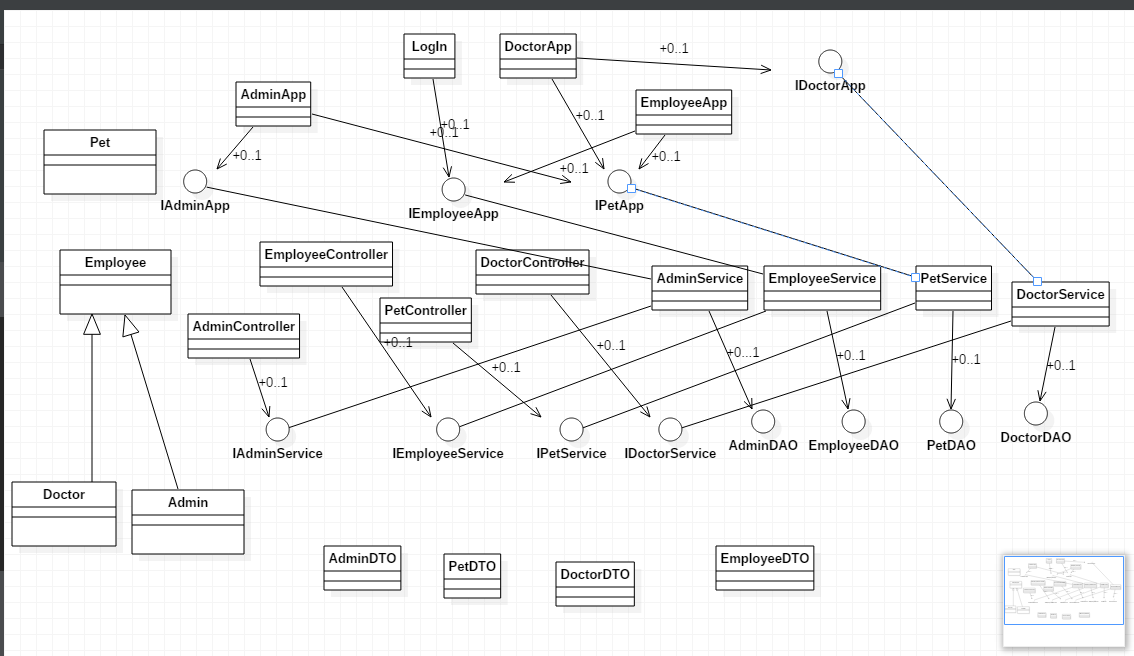
VI. Bibliography 5

# Project Specification

The Animal Shelter application is an application which has relatively low system requirements (a connection to the internet, database and a server) and can be used by anyone who needs to administer this type of organization. The application is designed to help the owner of an animal shelter improve service quality by making it easier for the employees to keep track of the animals and data about them and by making it easier for the clients to adopt.

# Elaboration – Iteration 1.1

# Domain Model

**

# Architectural Design

## Conceptual Architecture

For this project, a Model-View-Controller (MVC) pattern will be used. This is used in order to divide the application into three parts through developing the necessary interfaces.

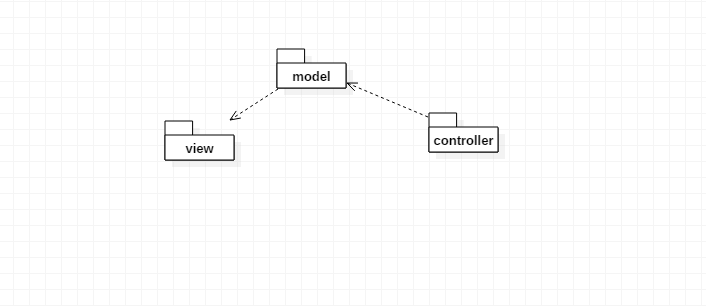
The model is represented by the classes corresponding to the database tables and the ones representing the service of them.

The controller is represented by a set of classes which send “signals” to the service classes, telling them what operations to perform. It is connected to the model through the interfaces implemented by the service of the model.

The view is represented by a set of Graphical User Interfaces. They are connected to the model(the service from the model) through interfaces implemented by the service.

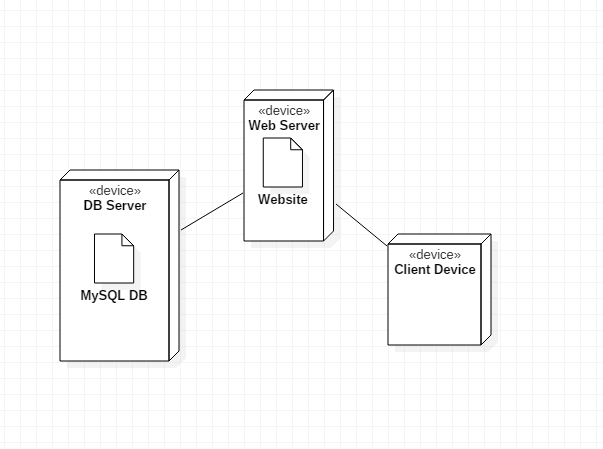
## Package Design

The package design illustrates the MVC architecture, having three main packages: model, view and controller.

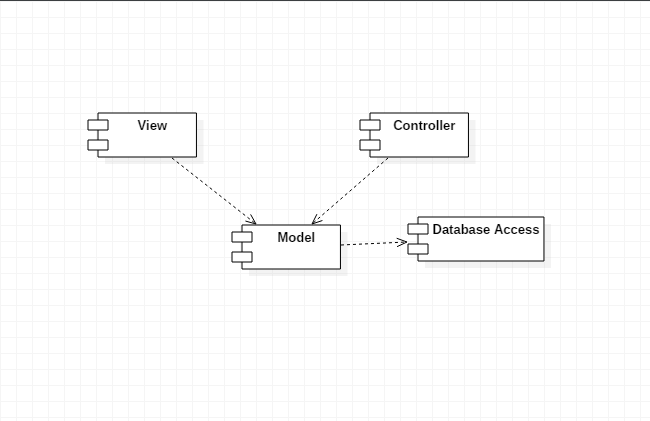
**

## Component and Deployment Diagrams

Deployment Diagram

**

Component Diagram

**

# Elaboration – Iteration 1.2

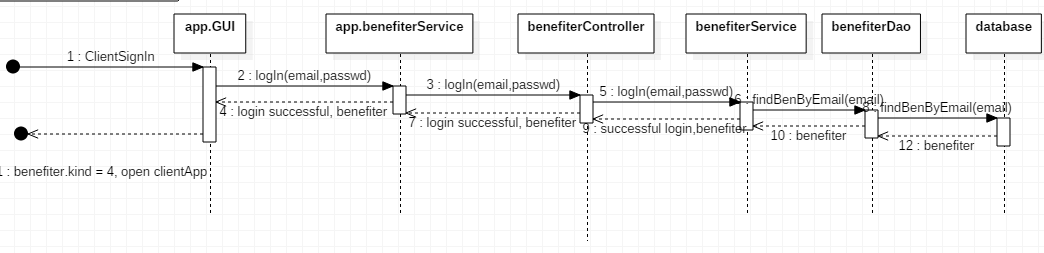
# Design Model

## Dynamic Behavior

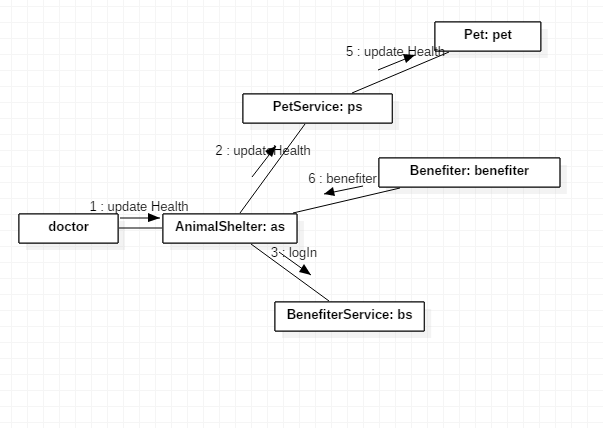
Two important scenarios are the following:

Client Sign In and Doctor Update Health.

**Sequence Diagram: Client Sign In**

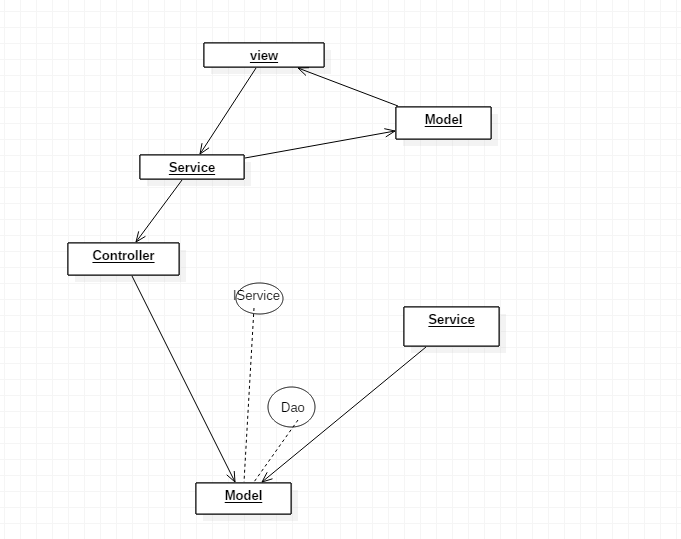


**Communication Diagram: Doctor Update Health**



## Class Design

For each of the objects in the model, the following can be applied:



# Data Model

The dada model of the system consists of the following classes:

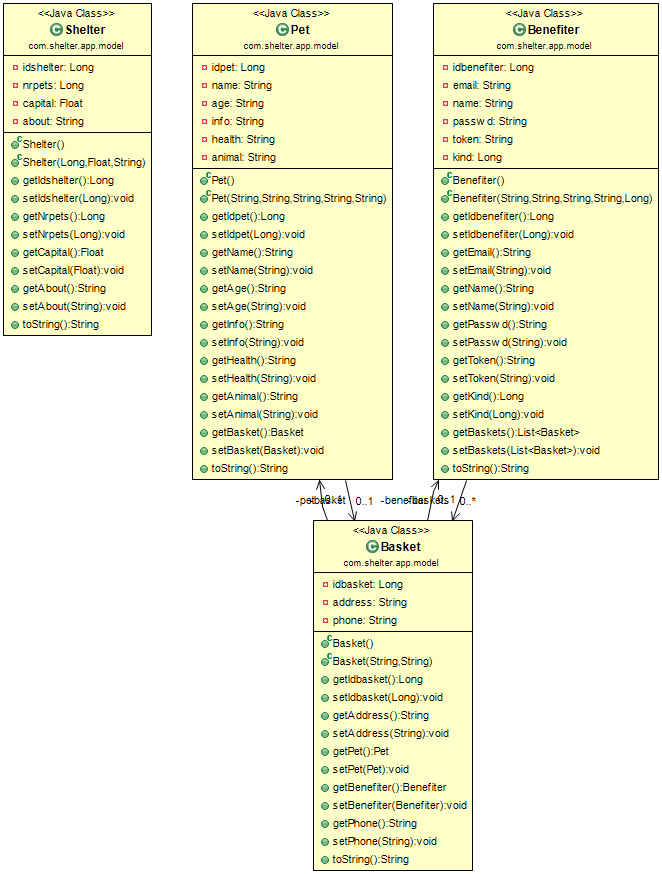
Benefiter

Pet

Shelter

Basket

The connections between the classes, as well as the class attributes and methods can be seen in the following image:



# Unit Testing

Both applications were tested separately and also together.

For example, for the first application, all the controllers were tested using Swagger, for each of the methods they provide.

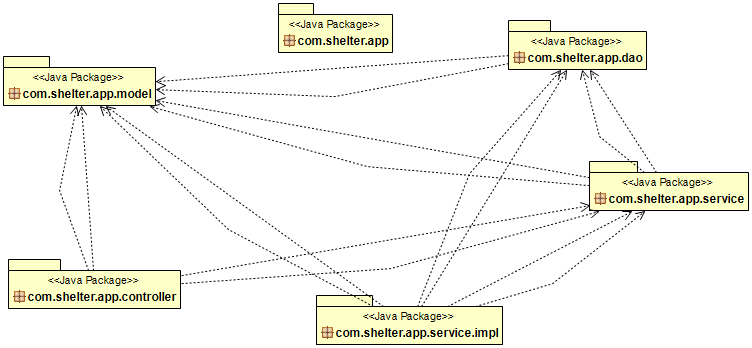
For the second application, a test class was firstly used, called TestClass, in order to test the methods from the service package as they were added. Afterwards, the tests were performed for the most important functionalities of the application, using the GUIs.

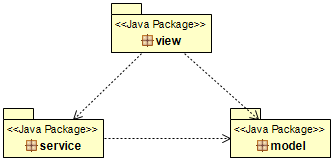
# Elaboration – Iteration 2

# Architectural Design Refinement

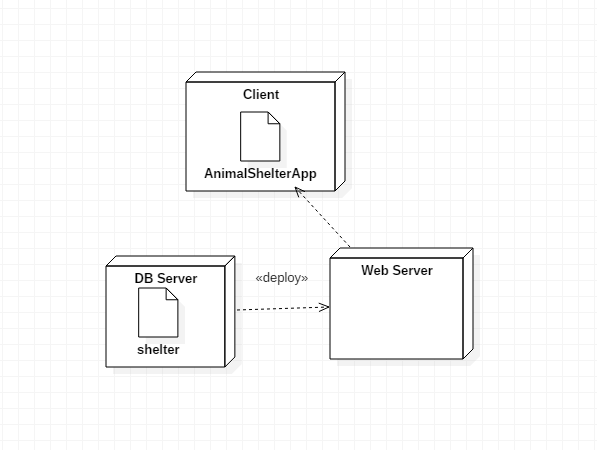
From the first iteration, the package design was changed. The application is composed of two applications, one created using SpringBoot and the other one being a maven, desktop application.

The following represent the **package diagrams** of each of these applications:





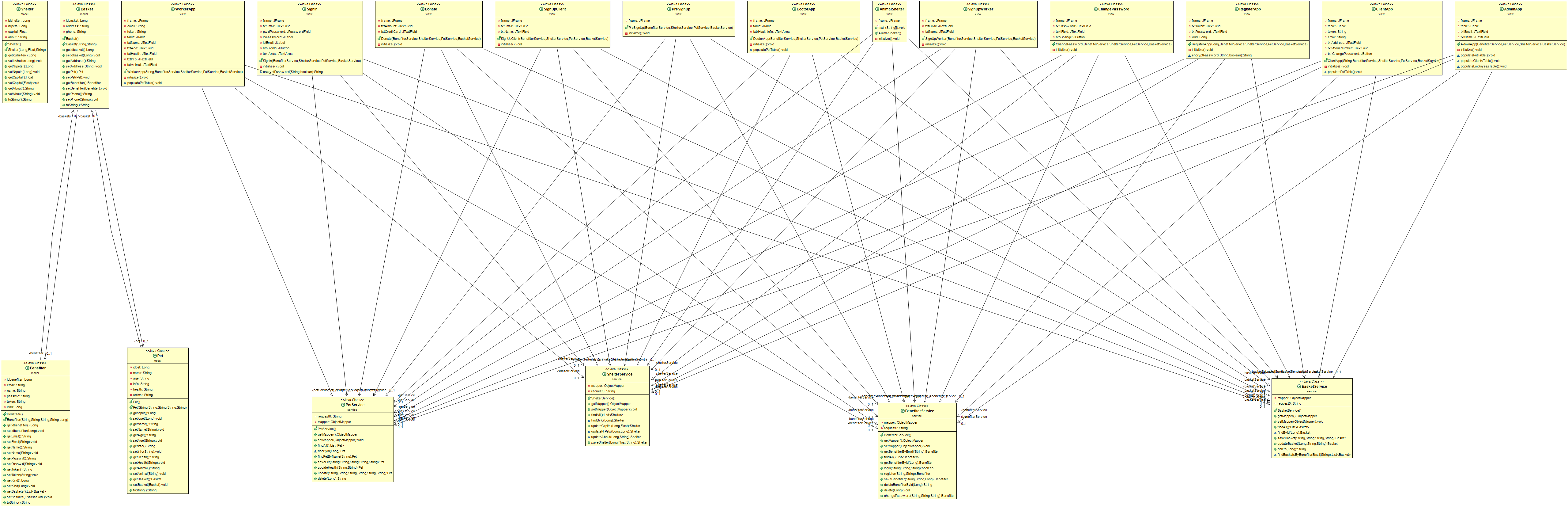
Next, the **deployment diagram**, which has changed, is presented:



# Design Model Refinement

The first application:

## 

****

The second application:

# Construction and Transition

# System Testing

As stated above, the testing was done using a test class which provided the necessary objects for each of the methods used. For example, for logIn, a BenefiterService object was created which called the logIn API.

# Future improvements

There are quite a few improvements which could be made to the project, among which the most important are:

- developing a web application instead of the desktop application which consumes the APIs:

This improvement can be done by using Thymeleaf

- adding more functionality to the application:

For example, introducing the option for the client to give up on some of the orders.

- using more methods to provide security:

Especially if the application would be turned into a web one, Spring Security could be used.

# Bibliography