Analysis and Design Document

Cornescu Andreeea Mara

**30431**

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 24/04/2018 | 1.0 | Project specification and Elaboration | Cornescu Andreea Mara |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

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

Design and implement an application which will help people make online bookings for busses anywhere in the country. There will be two main types of users: the travel agent (representing the bus agency) and the customer (travelers). Travel agencies will present their available busses, the destination, and the seat number and will constantly update the information if any changes occur. Besides the regular users, there is also the administrator which will make sure everything goes as planned.

The administrator can perform the following operations:

* CRUD on bus agencies
* CRUD on users

All users can:

* Create an account
* Log in
* Log out

A travel agent can perform the following operations:

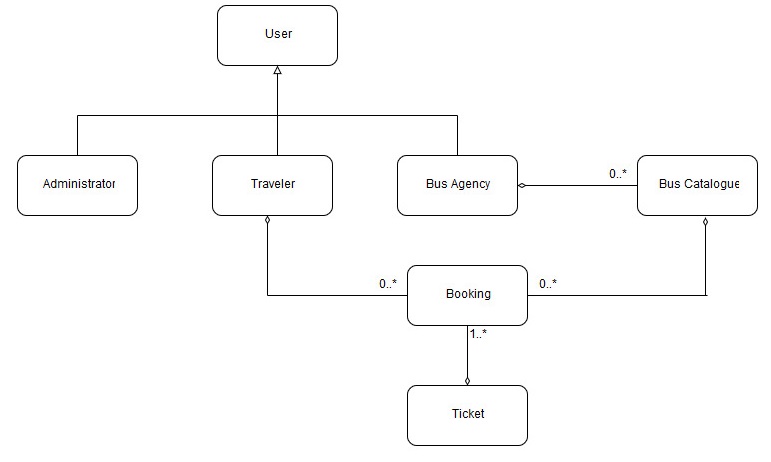
* View all bookings
* Add booking
* Sell tickets

The customer can perform the following operations:

* View bus program and routes
* Search for a bus
* Book a ticket
* Choose payment method
* Cancel ticket

# Elaboration – Iteration 1.1

# Domain Model

**

# Architectural Design

## Conceptual Architecture

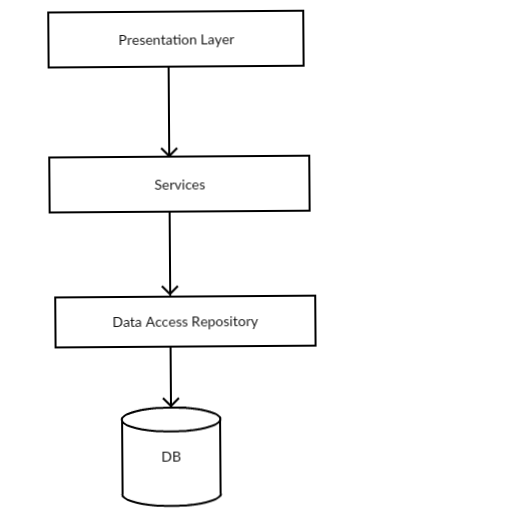
The application is designed following the **Layered Architecture pattern** and **MVC Architectural Pattern**.

**Layered Architecture pattern**

Components within the layered architecture are organized into horizontal layers, each layer having a well defined role within the application. Each layer provides a set of services which can be used by the layer above.

The architecture pattern is composed of the following layers:

* Presentation Layer – responsible with the UI;
* Services Layer – business logic behind the application;
* Data Access Repository – provides access to the database and performs specific operations;

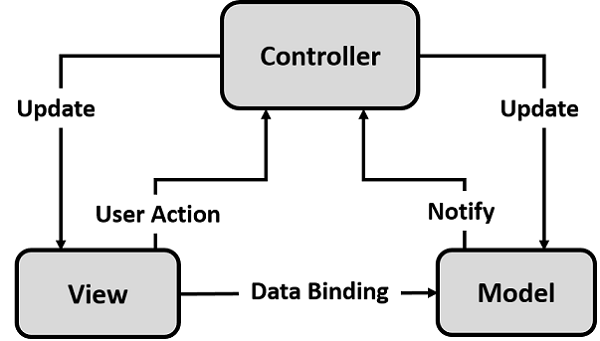
****

**Motivation:**

1. The concept of layered architecture is easy to learn and implement.
2. It provides an ease of maintenance of the code base, managing presentation code and business logic separately, so that a change to business logic, for example, does not impact the presentation layer.
3. It adds reliability and more independence of the underlying servers or services.

**MVC Architectural Pattern** stands for Model-View-Controller Pattern. This pattern is used to separate the data access layer, business logic code and the graphical user interface that has to be defined and designed to let the user interact with the application.

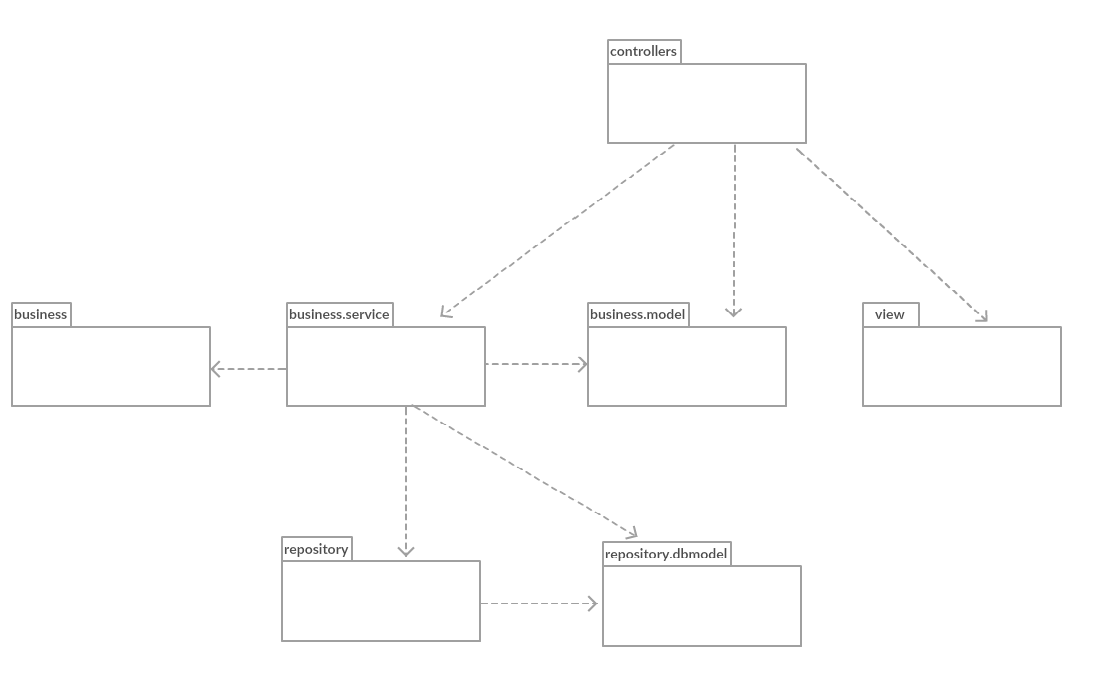
* **Model** - Model represents an object or JAVA POJO carrying data. It can also have logic to update controller if its data changes.
* **View** - View represents the visualization of the data that model contains.
* **Controller** - Controller acts on both model and view. It controls the data flow into model object and updates the view whenever data changes. It keeps view and model separate.

****

**Motivation:**

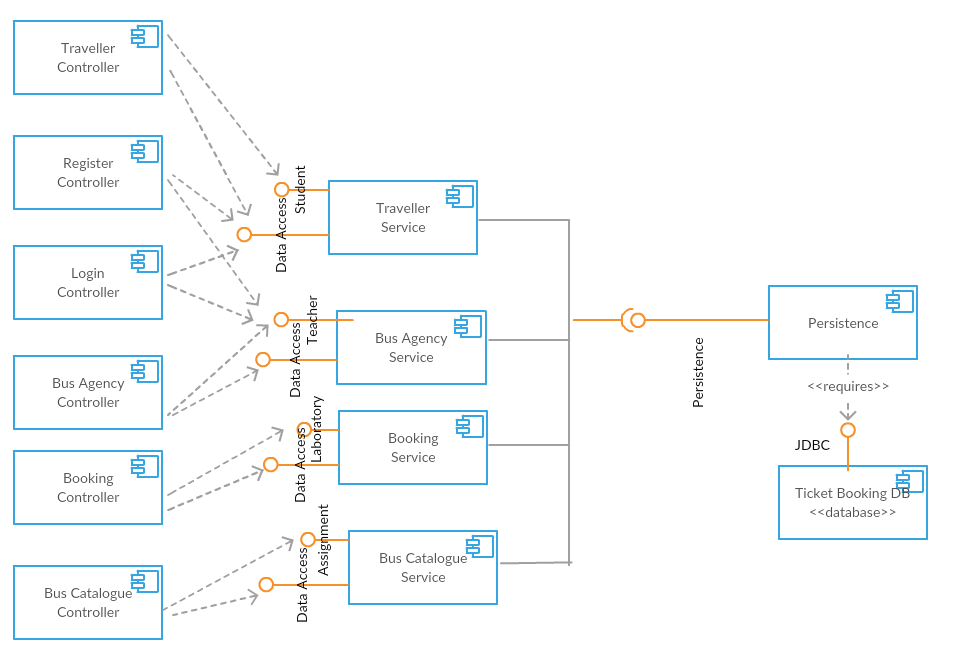
* Modification does not affect the entire model: The Model does not depend on the views part. Therefore, any changes in the Model will not affect the entire architecture.
* Fast development process: It is possible for a person to work on the view while another can work on the logic of the application

## Package Design

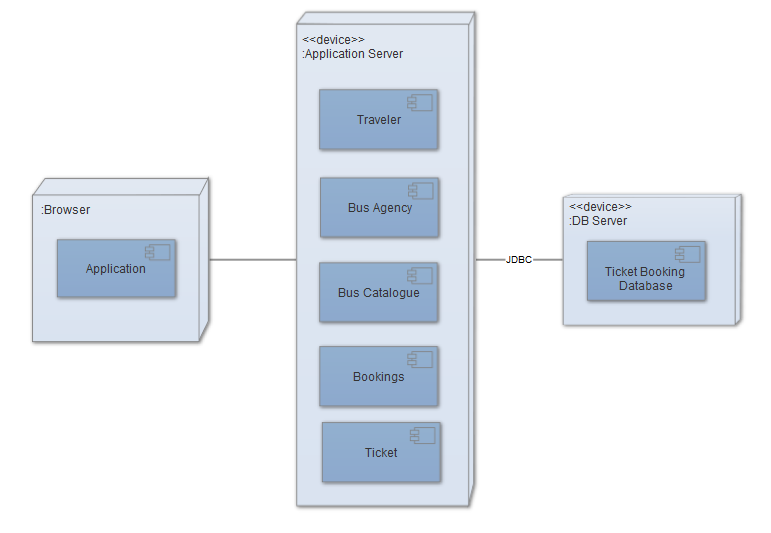
**

## Component and Deployment Diagrams

Component Diagram

**

Deployment Diagram



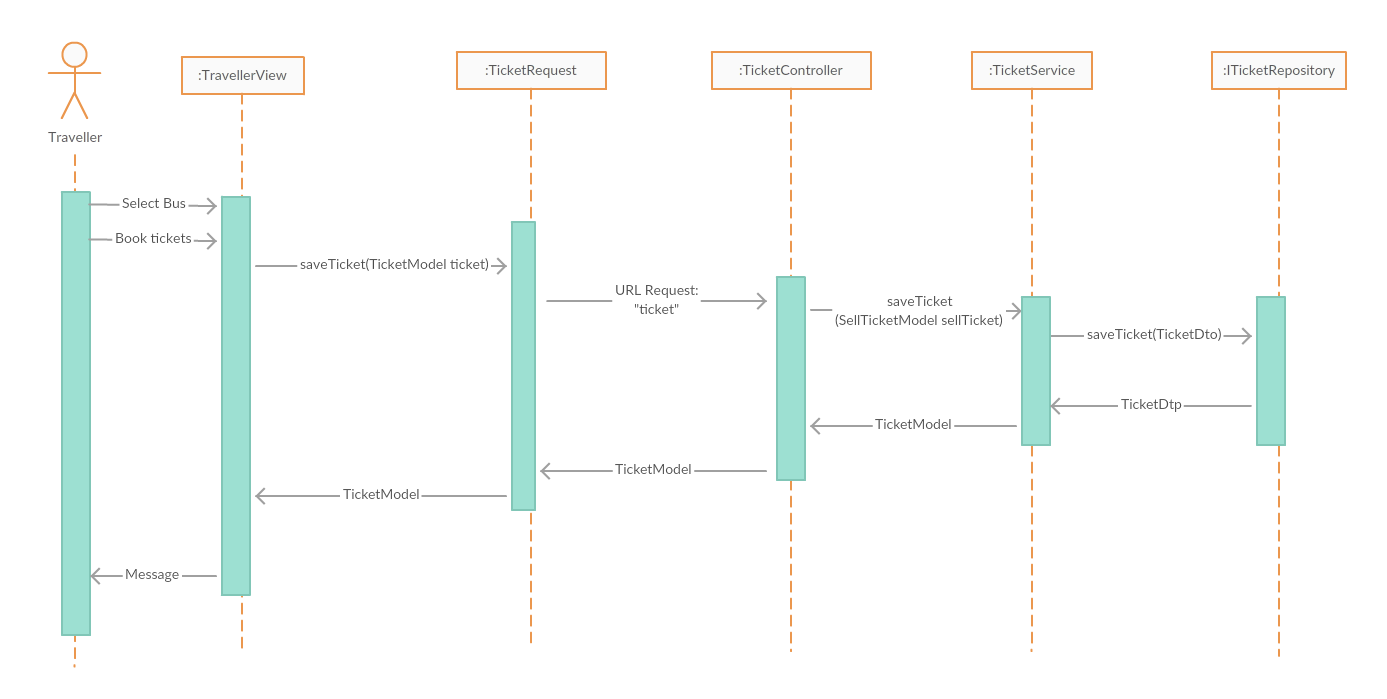
# Elaboration – Iteration 1.2

# Design Model

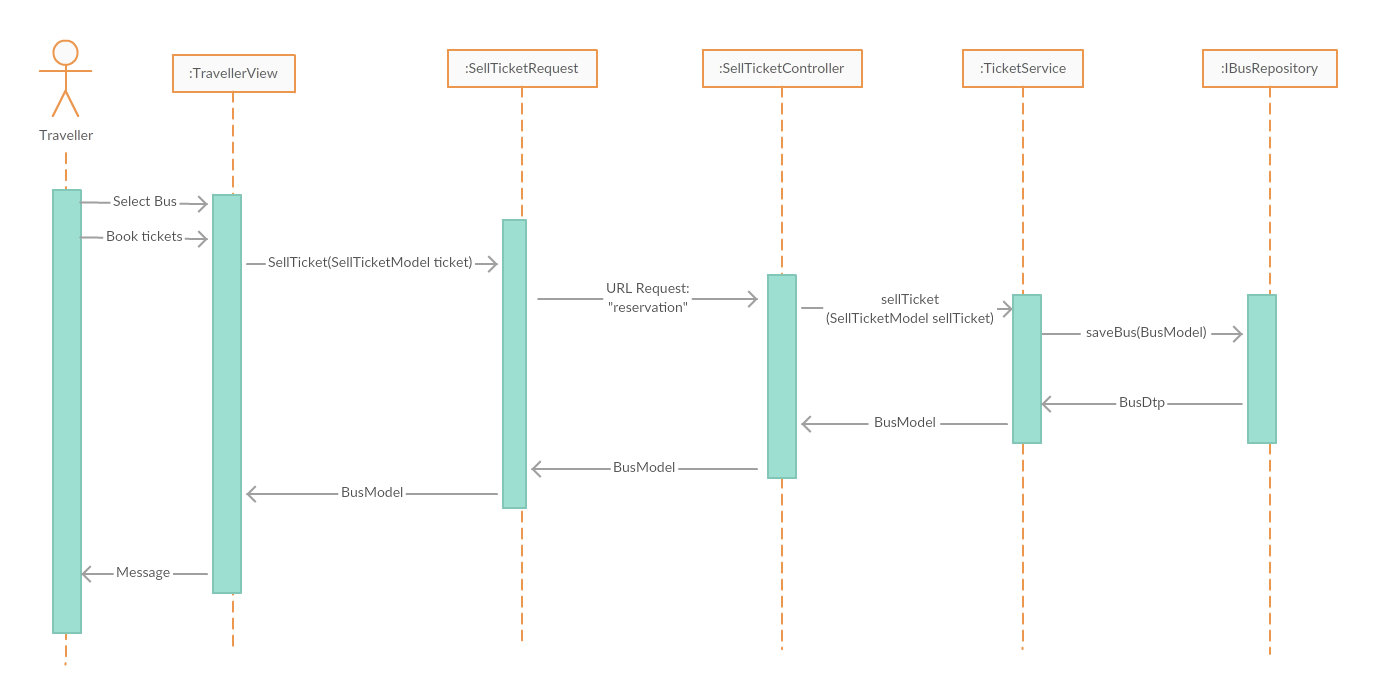
## Dynamic Behavior

Sequence diagram for booking a ticket. When a ticket is booked, the bus numbers of available seats decreases with the number of seats the user wants, which means that we make an update on the bus. Also, the ticket is saved in the database.

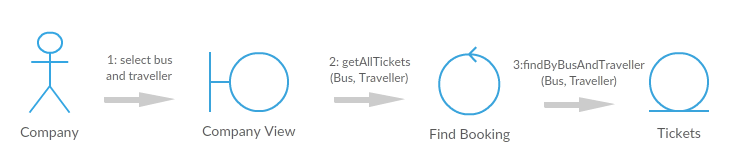
Save Ticket sequence diagram:



Update Bus sequence diagram:

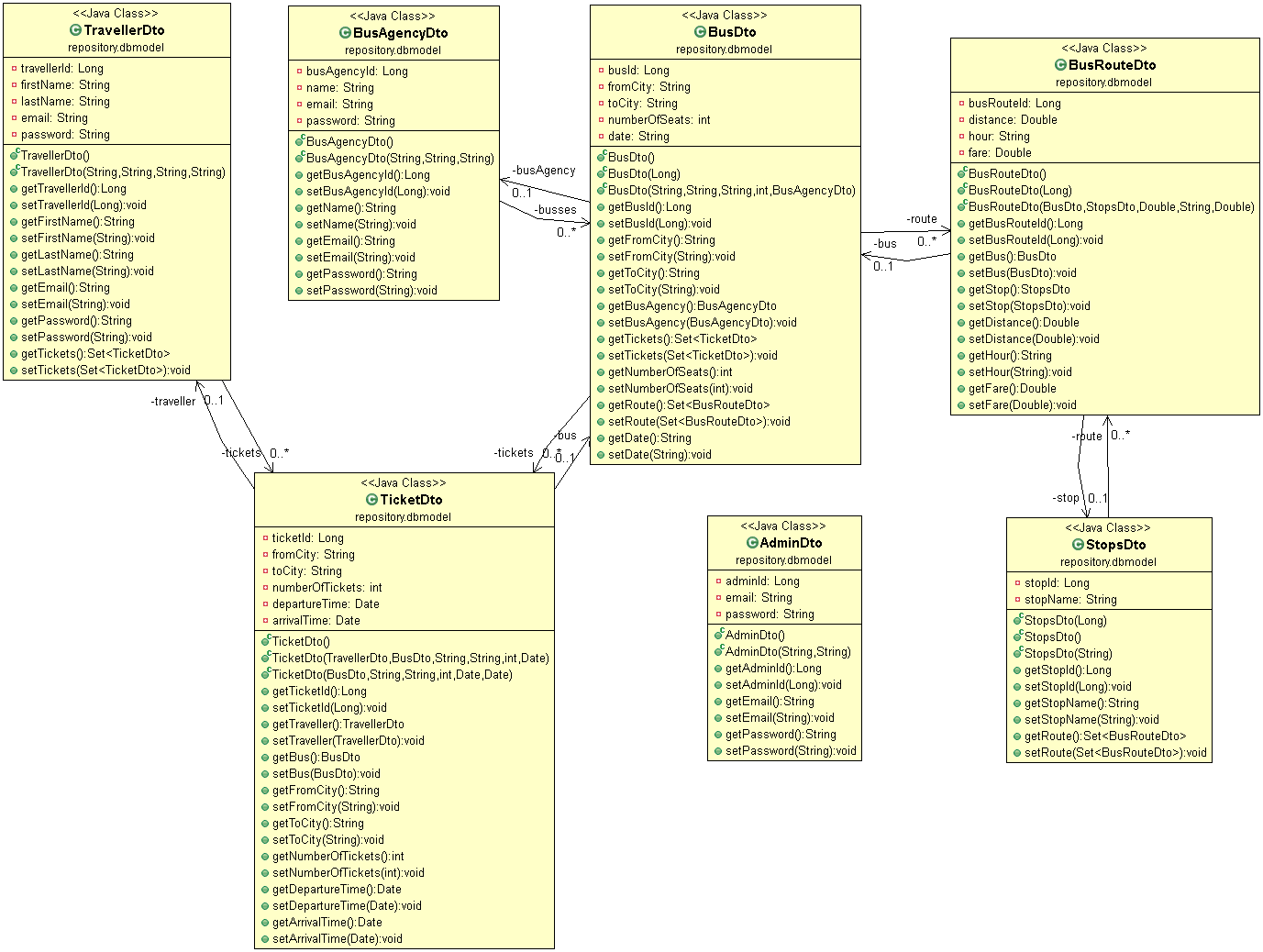


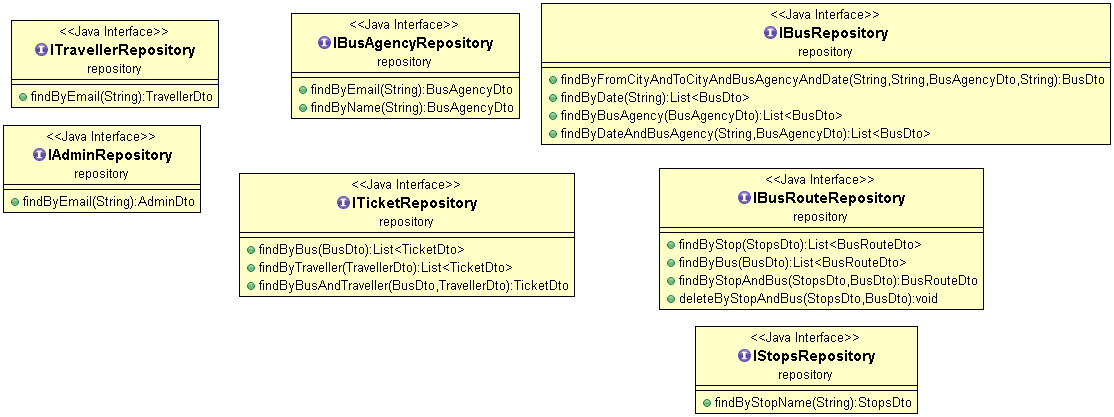
Communication Diagram:



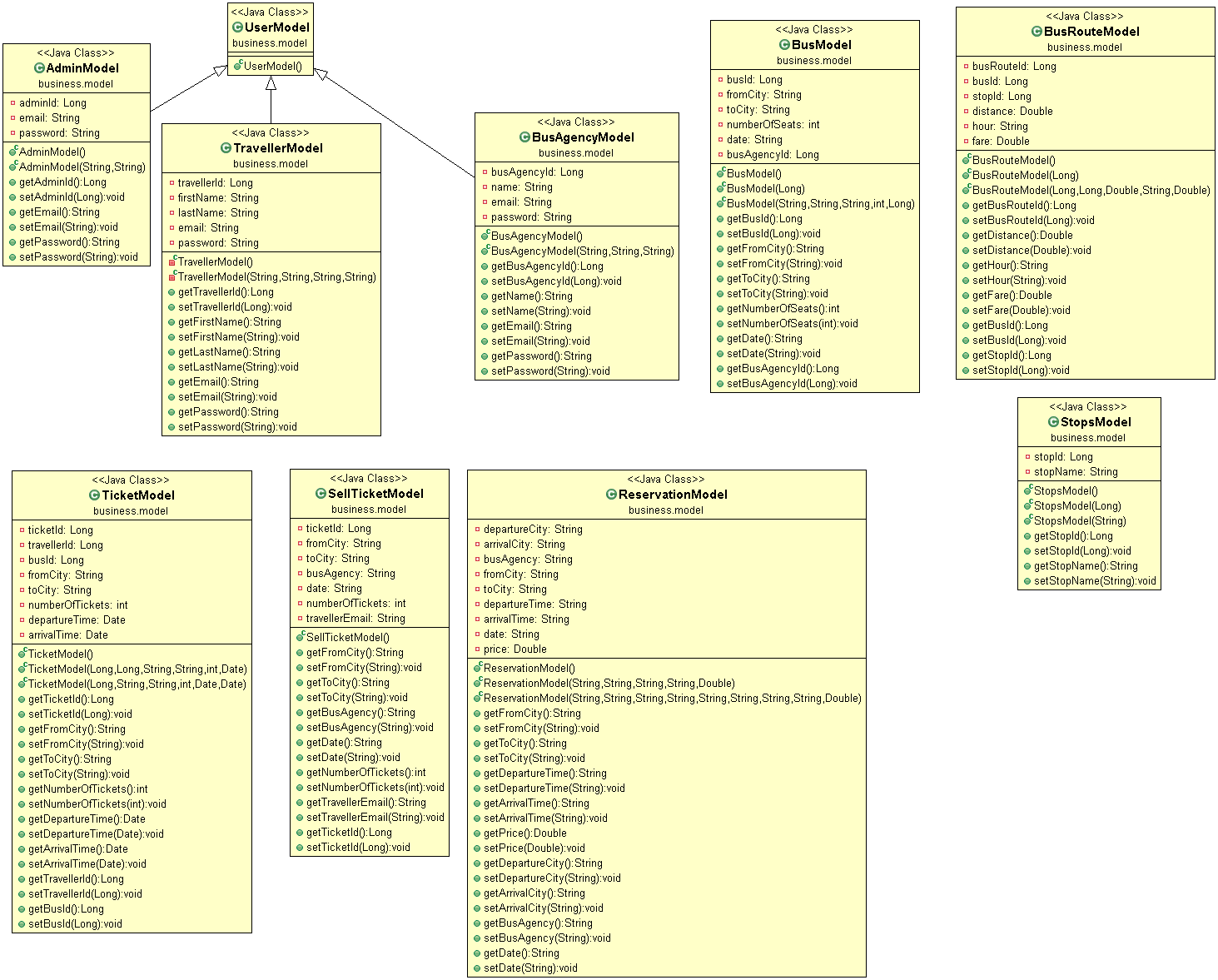
## Class Design

Repository – db Model

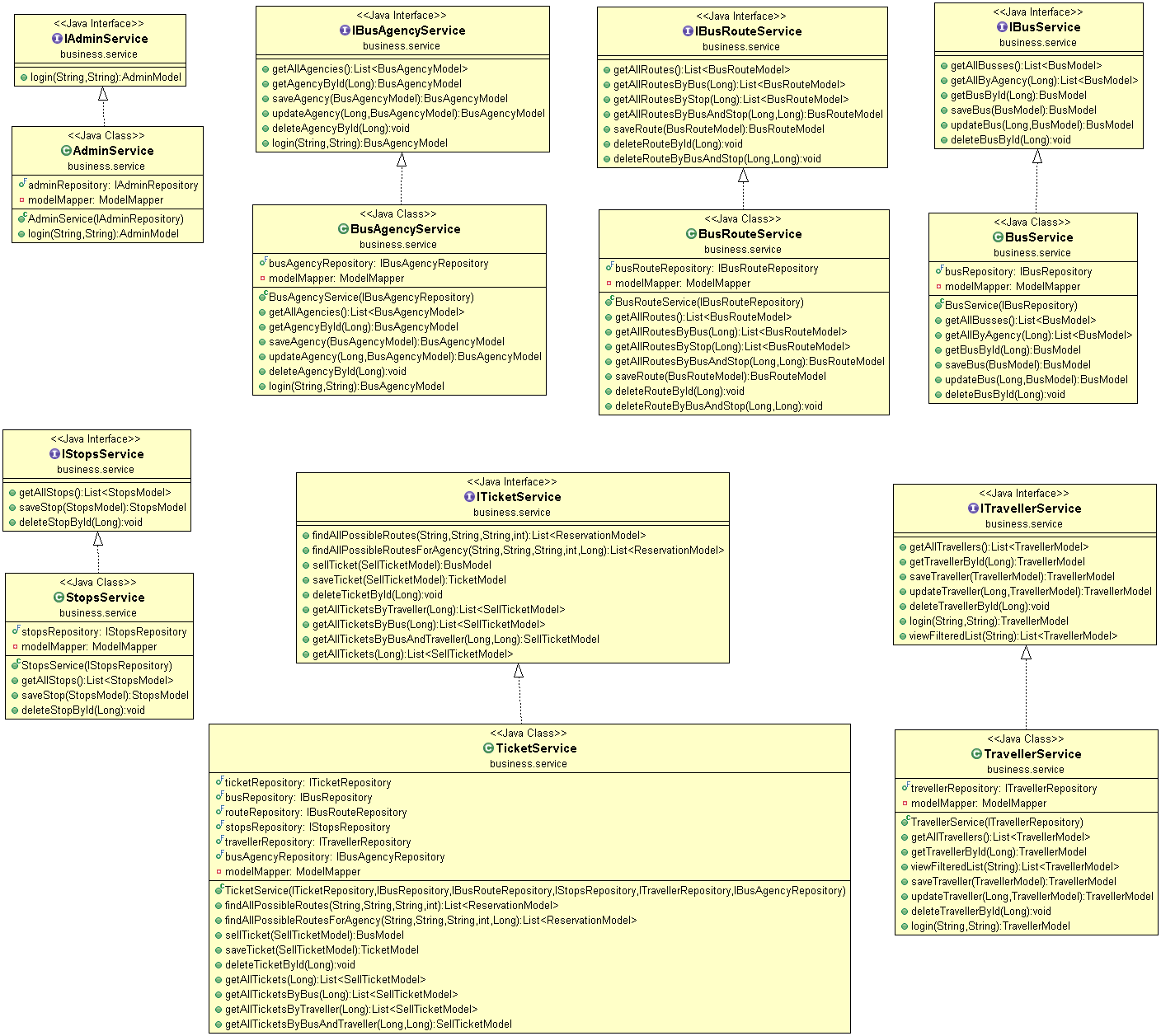


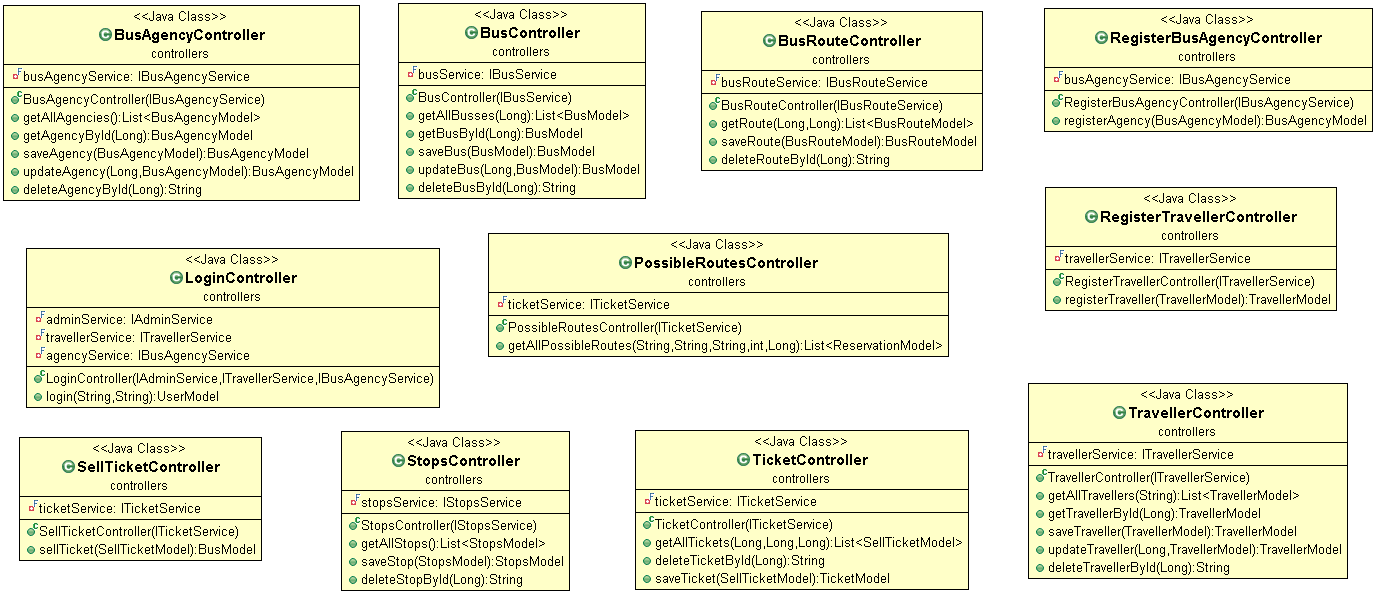
Repository

Model

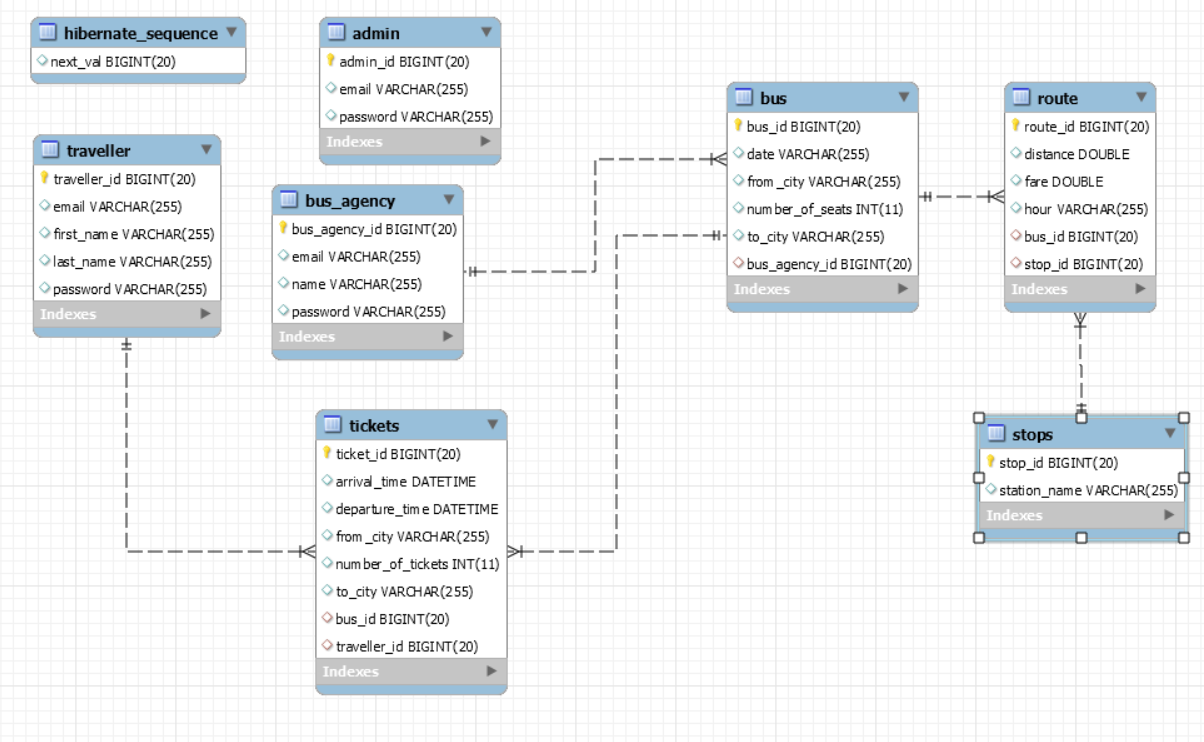


Services



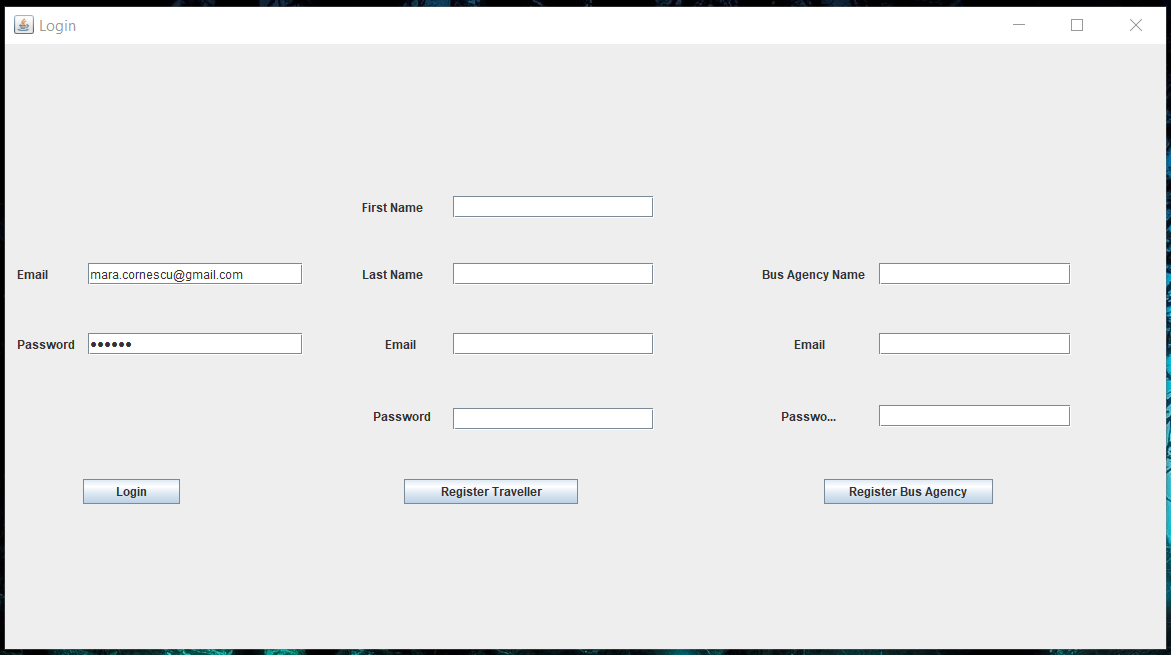
Controllers

# Data Model

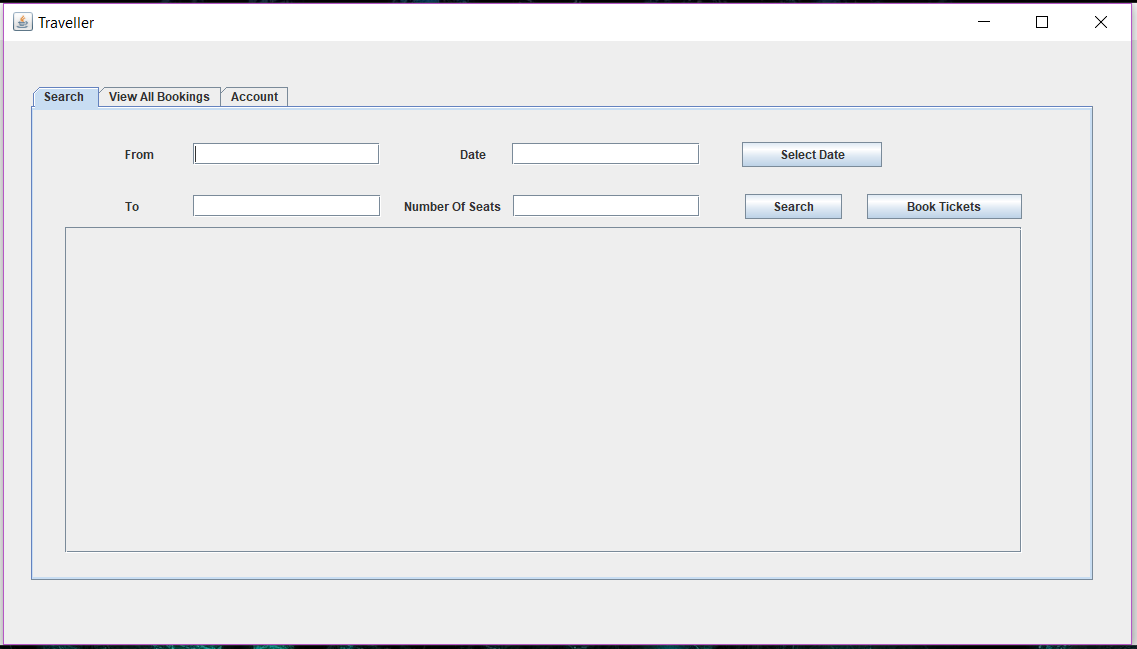


# Testing

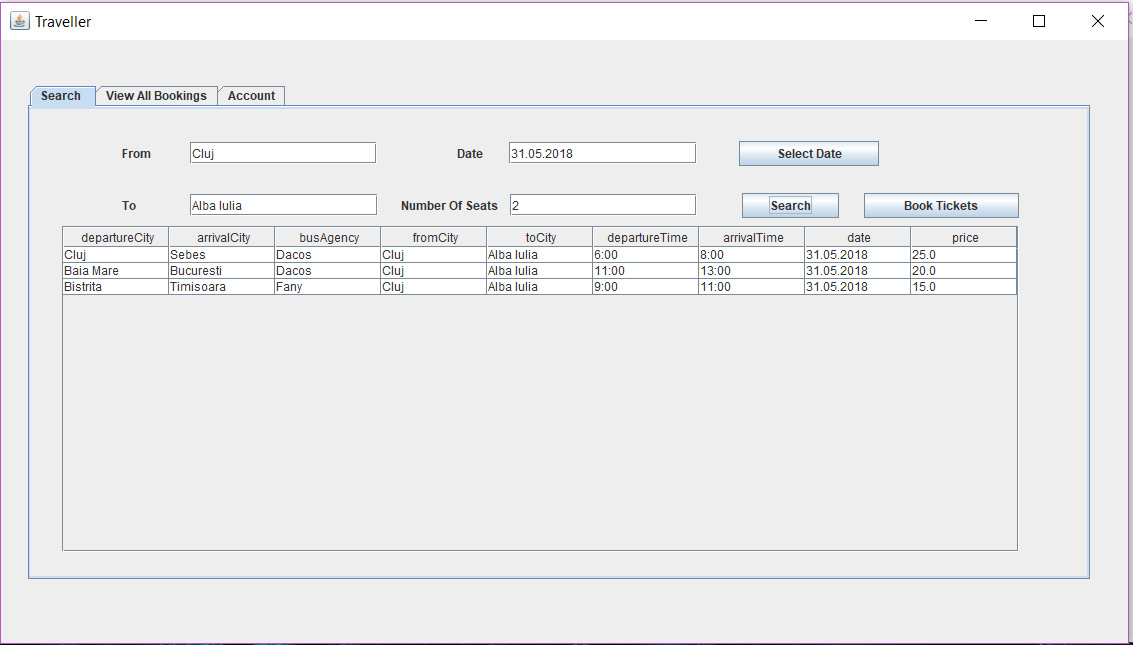
Login as traveler



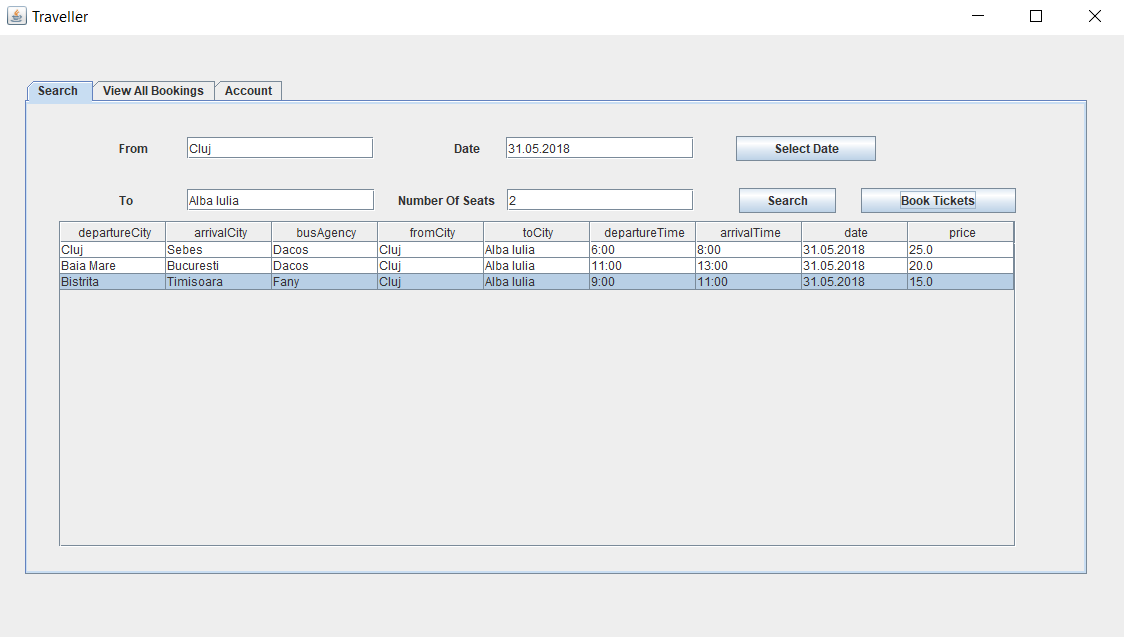
The traveler view opens



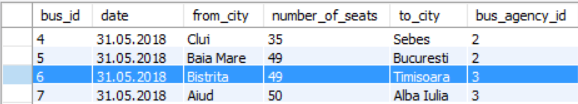
Search for a bus



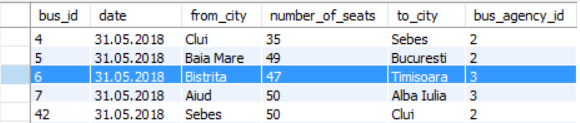
Select the desired bus and book the tickets



Database – before the booking



Database – after the booking

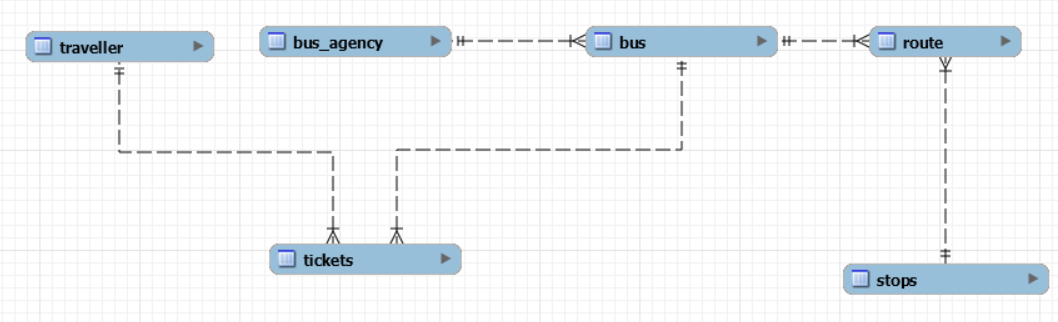


We can see that the number of available places has decreased with the number of tickets we have booked.

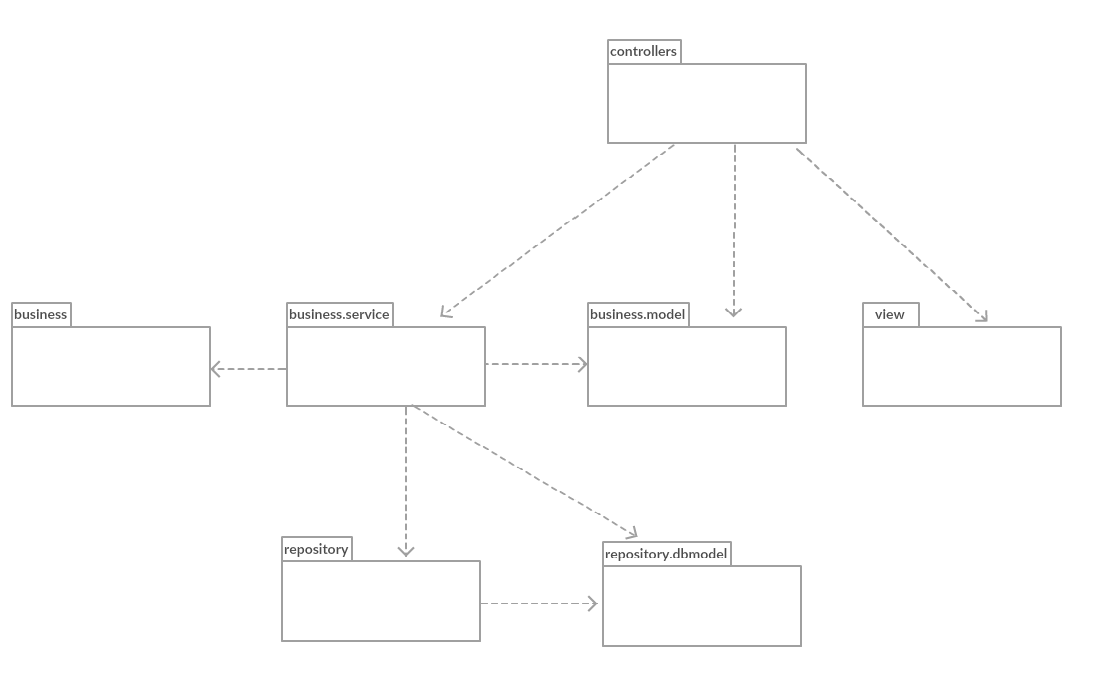
# Elaboration – Iteration 2

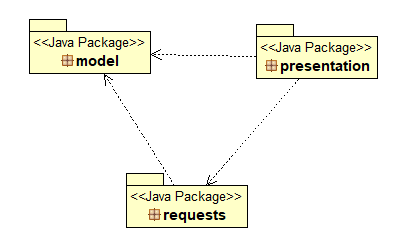
# Architectural Design Refinement

Conceptual architecture

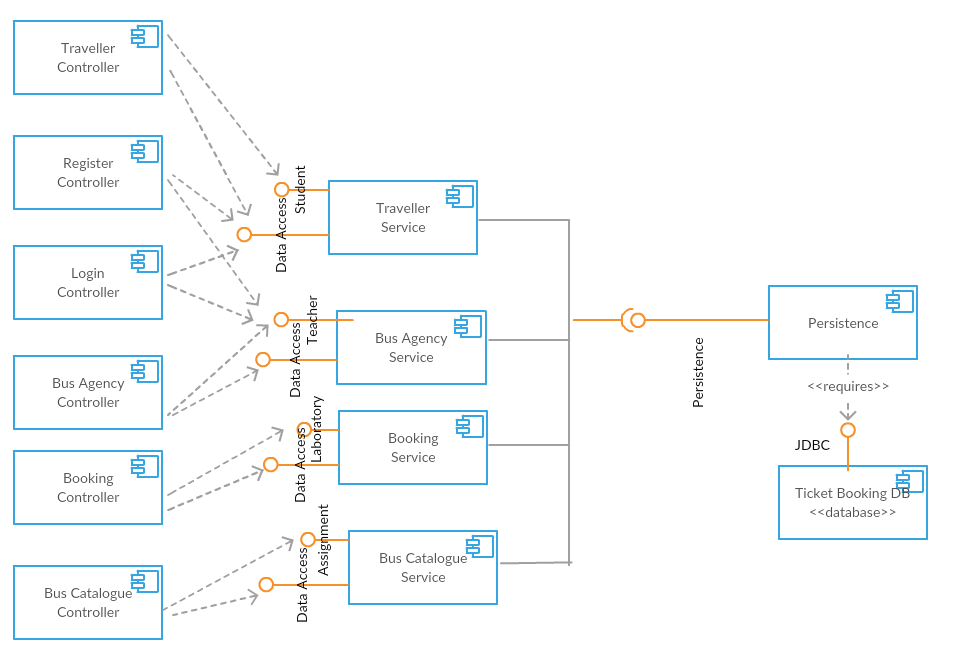


Package diagram

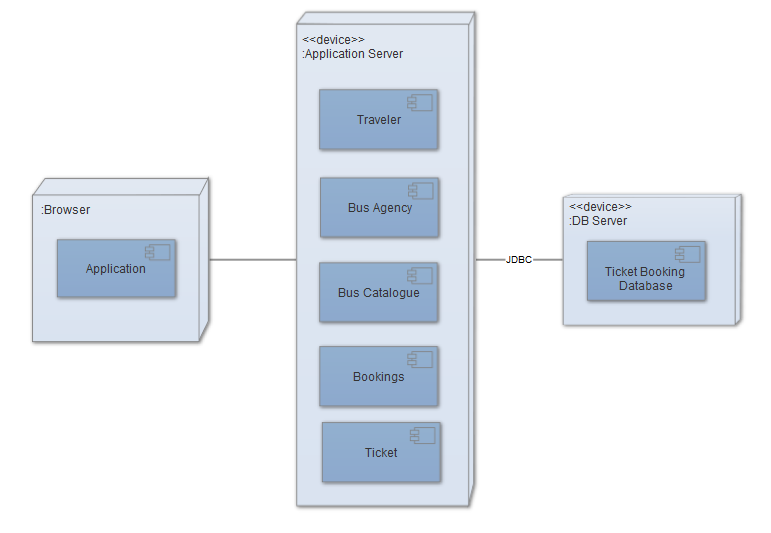




Component diagram



Deployment diagram



# Construction and Transition

# System Testing

# Future improvements

* Translate to web;
* Add the option to buy tickets;
* Tell the client where the departure and arrival is located;

# Bibliography

* MVC: <http://www.tutorialsteacher.com/mvc/mvc-architecture>
* Diagrams: <https://creately.com>
* Spring Boot: <https://docs.spring.io/spring-boot/docs/current/reference/html/getting-started-first-application.html>
* Testing: <https://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-testing.html>