DormShare
Analysis and Design Document
Student: Eric Dumea
Group: 30433

Revision History

Date	Version	Description	Author
13/apr/18	1.0	Initial Fill	Eric Dumea
22/apr/18	1.1	Iteration 1.2	Eric Dumea

Table of Contents

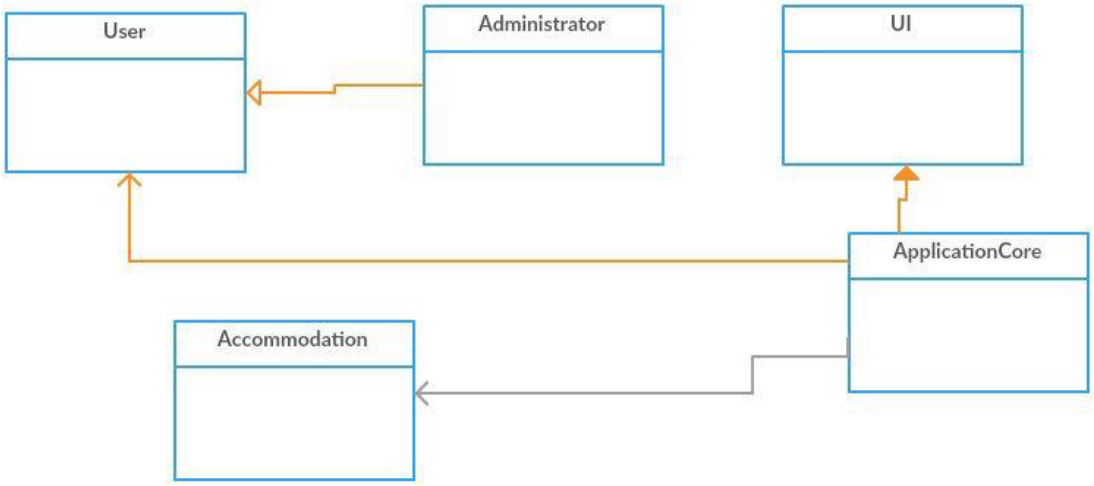
I. Project Specification

DormShare is a software solution that tries to deal with the problem of students not affording accommodation in big cities that usually have touristic attractions by sharing dorm rooms. The application will provide a easy to use, minimalistic interface for the students.

II. Elaboration – Iteration 1.1

1. Domain Model

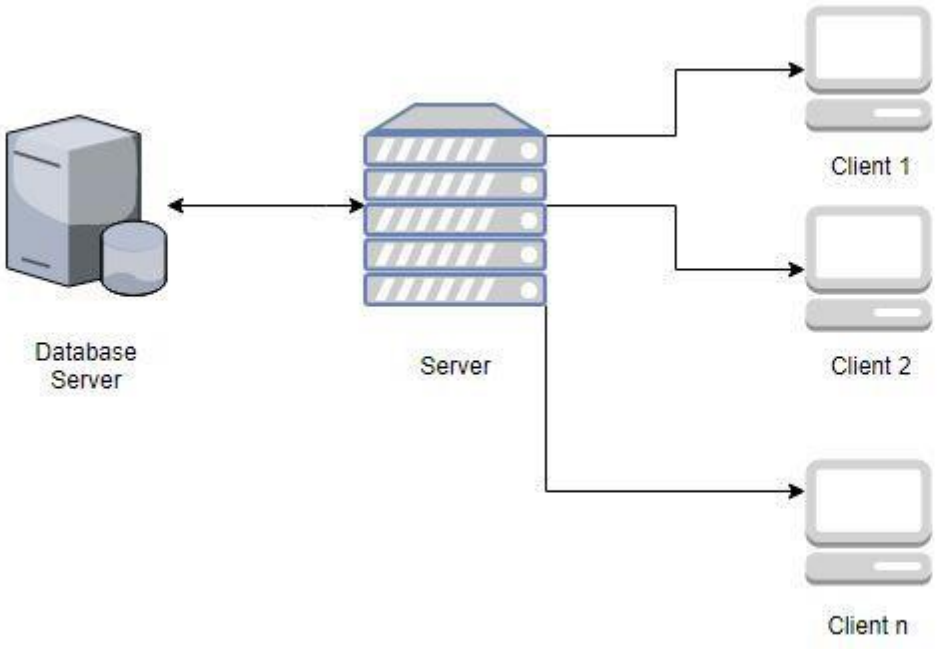
The DormShare application will be developed obeying the OOP principles, but also the SOLID principles. Thus, multiple classes will be needed, most probably one for each table in the Database design, at least one for the GUI and some for the interaction between the basic models and the finite product. A preliminary class design diagram is:



2. Architectural Design

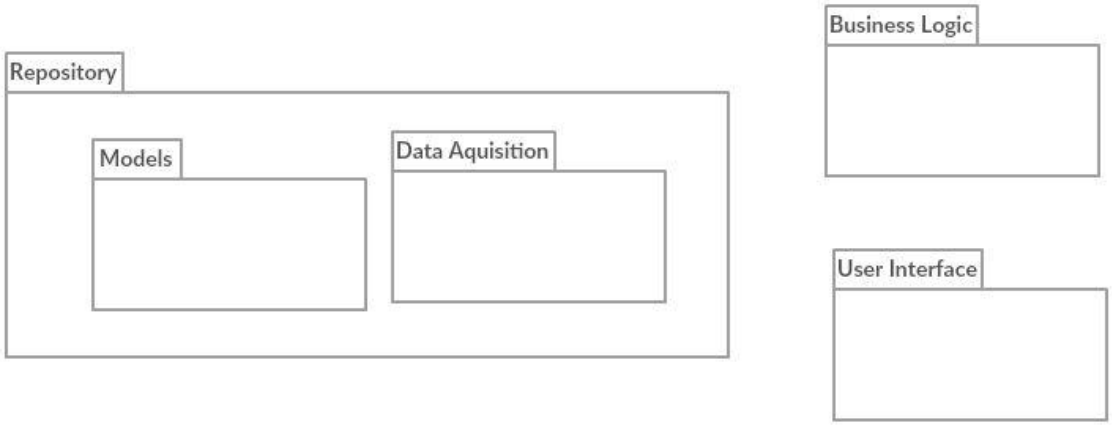
2.1 Conceptual Architecture

The project will be implemented using the Client-Server architectural pattern. This choice was made because the application manages the interaction between multiple users, and it needs a Server application to do the connection to the database and also to manage the users. The Server application will be the backend of the application. Provided is a system architecture diagram:

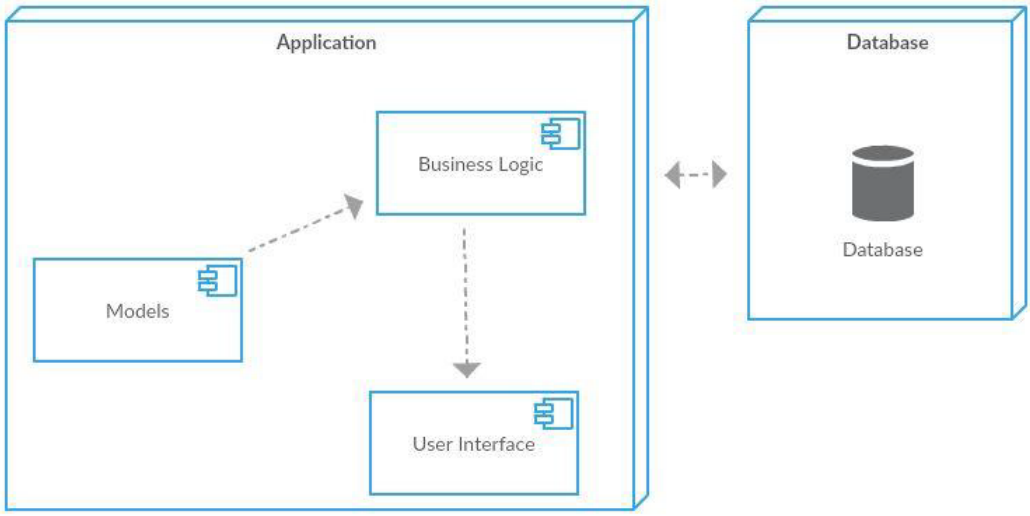


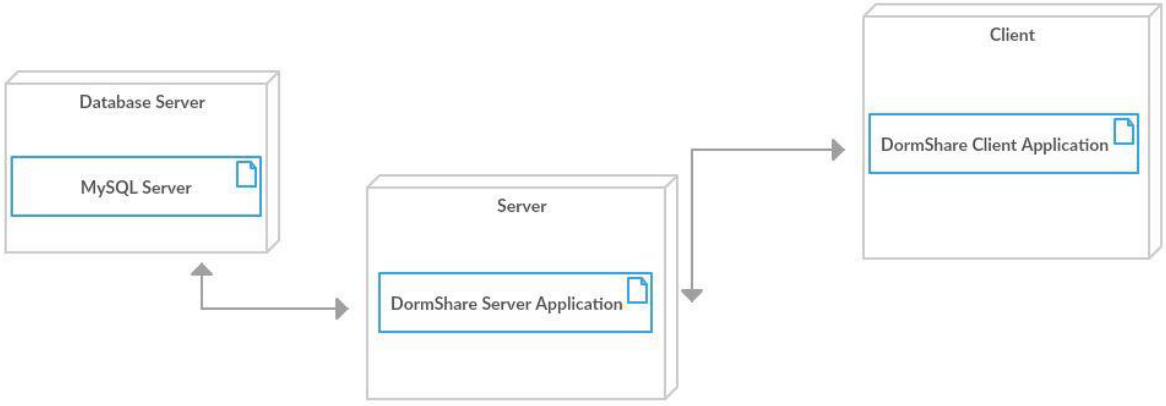
2.2 Package Design

Packages are used to separate components that are not related or loosely related.



2.3 Component and Deployment Diagrams

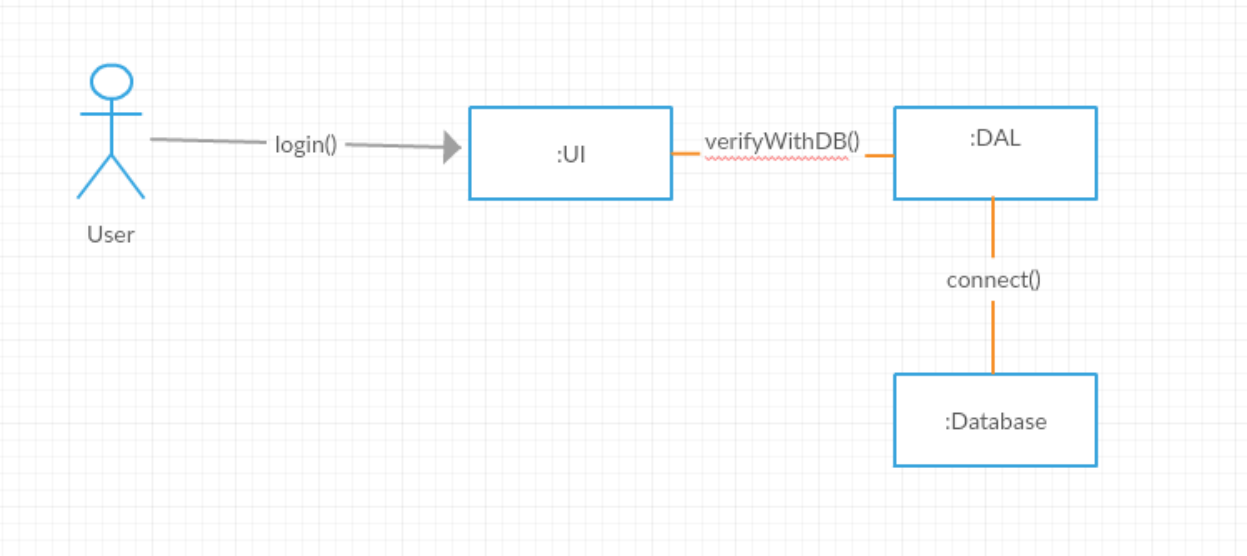


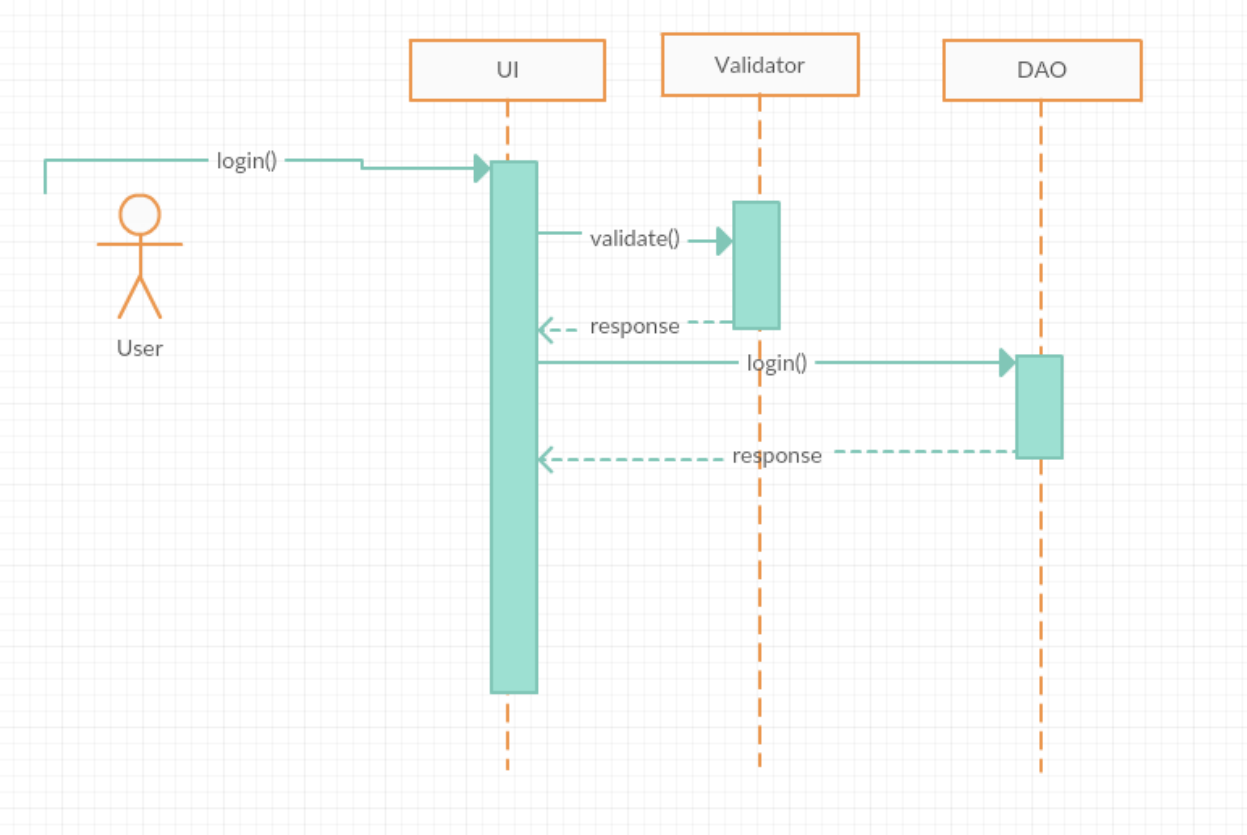


III. Elaboration – Iteration 1.2

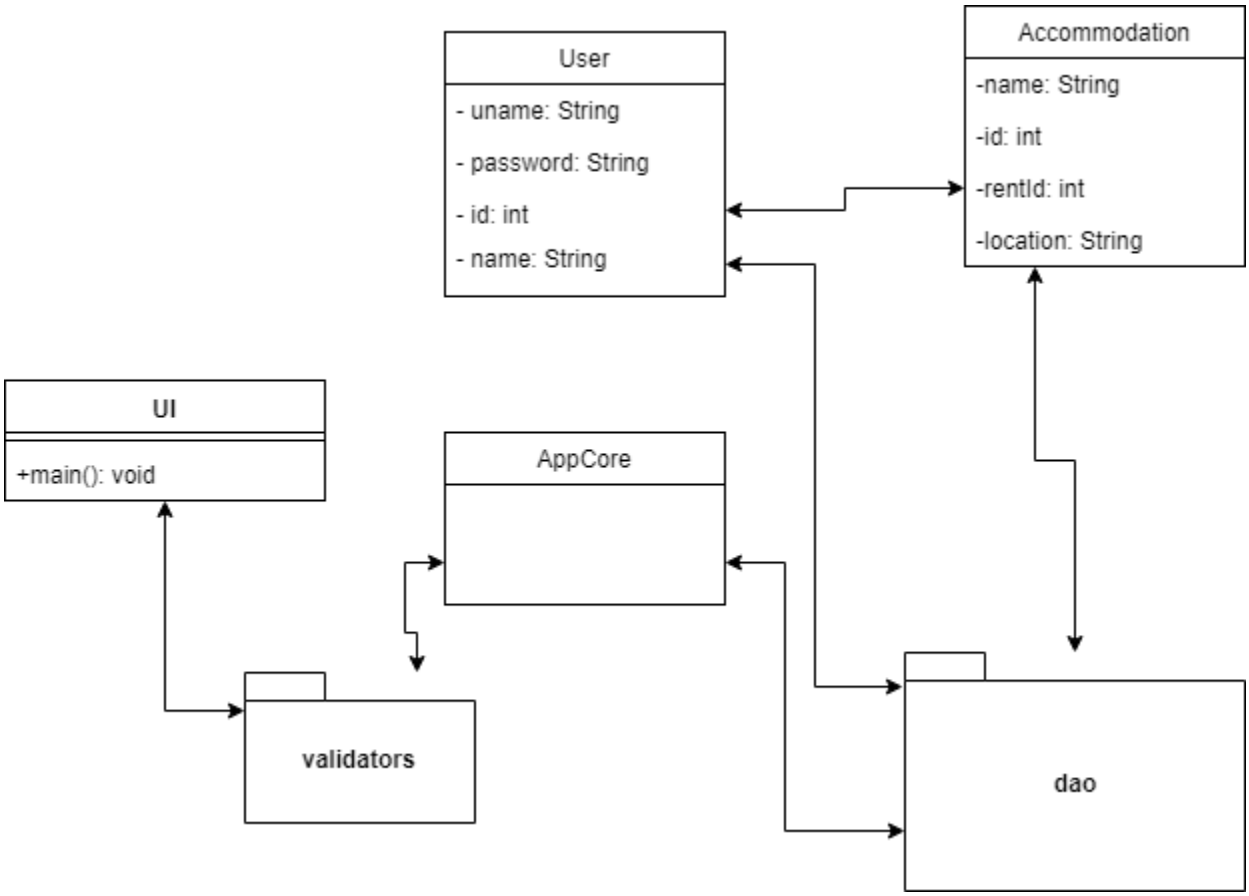
1. Design Model

1.1 Dynamic Behavior

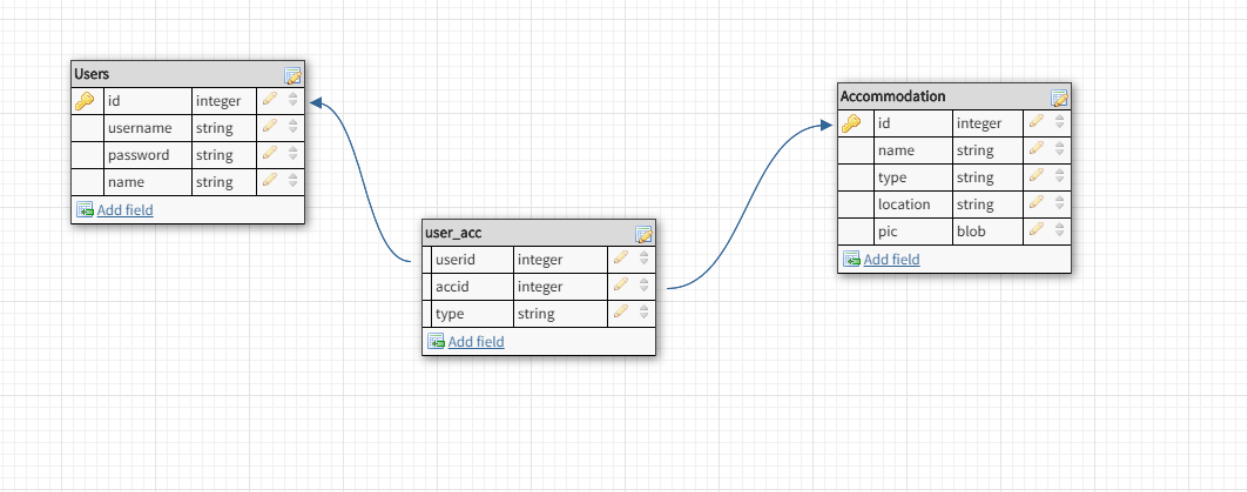




1.2 Class Design



2. Data Model



3. Unit Testing

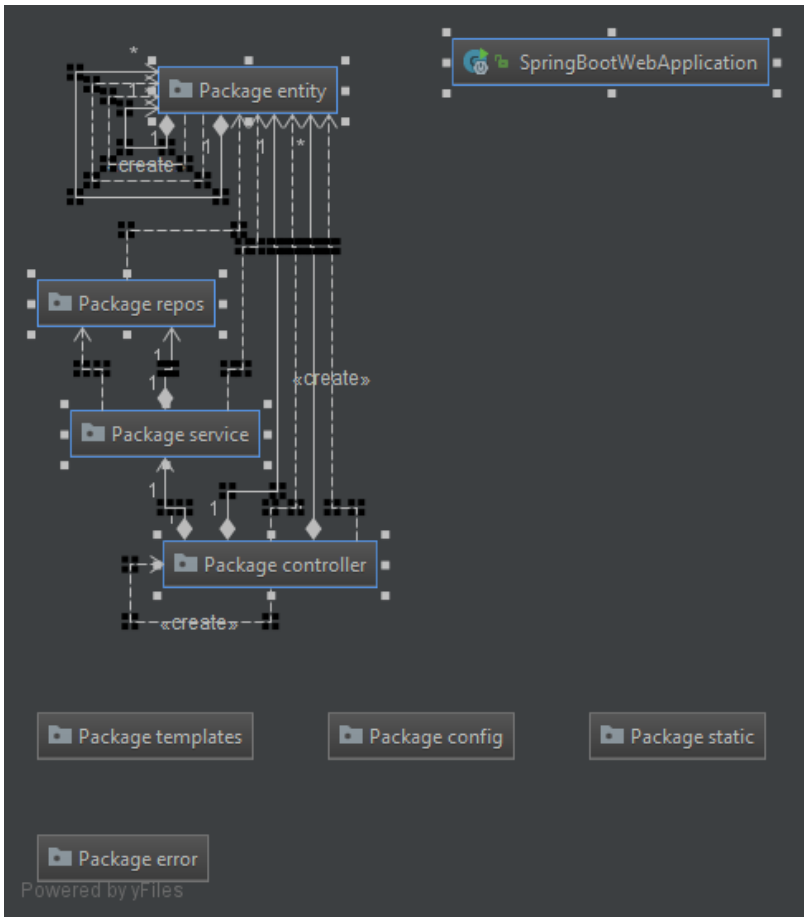
The project will be tested using Unit Testing provided by the JUnit framework. One specific scenario in which the system should be tested is when one user wants to rent a specific dorm room. This way, we'll test if the outcome of the system is the desired one, i.e. in this example, the one renting the accommodation should be notified. Also, valid unit tests could be conducted to test various methods that will be implemented, and can not be specified at the time of writing this document.

IV. Elaboration – Iteration 2

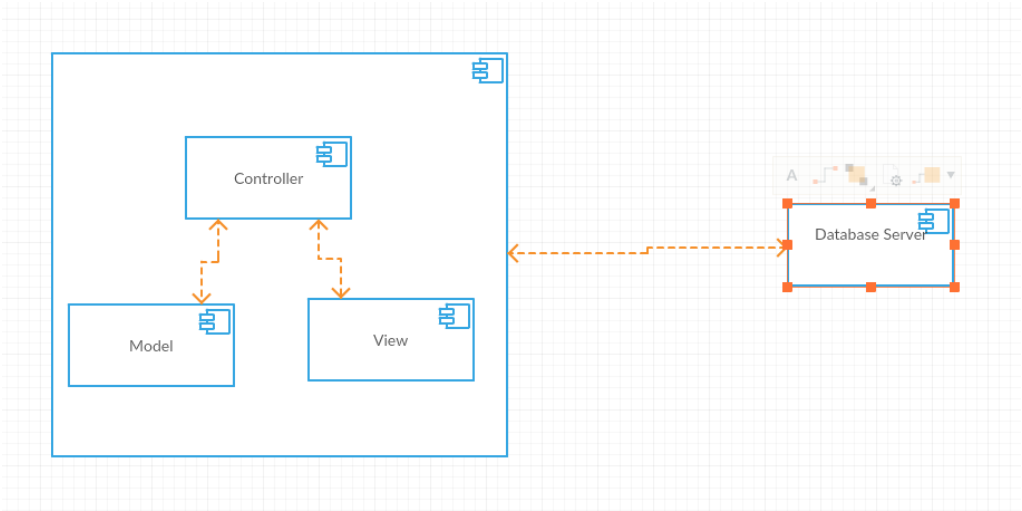
1. Architectural Design Refinement

Based on the fact that the application was developed in Spring Boot MVC framework, the conceptual architecture is now a client-server, with MVC functionality. This architectural design is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients. Often clients and servers communicate over a computer network on separate hardware, but both client and server may reside in the same system. A server host runs one or more server programs which share their resources with clients. A client does not share any of its resources, but requests a server's content or service function.

- Package Design

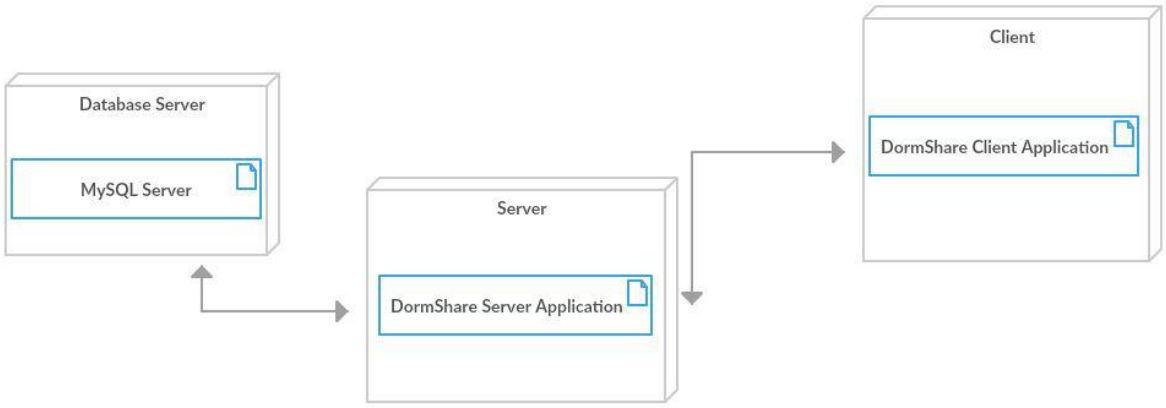


- Component diagram



- Deployment diagram

The deployment diagram has not changed from the latest version, this application still being a client-server one.



2. Design Model Refinement

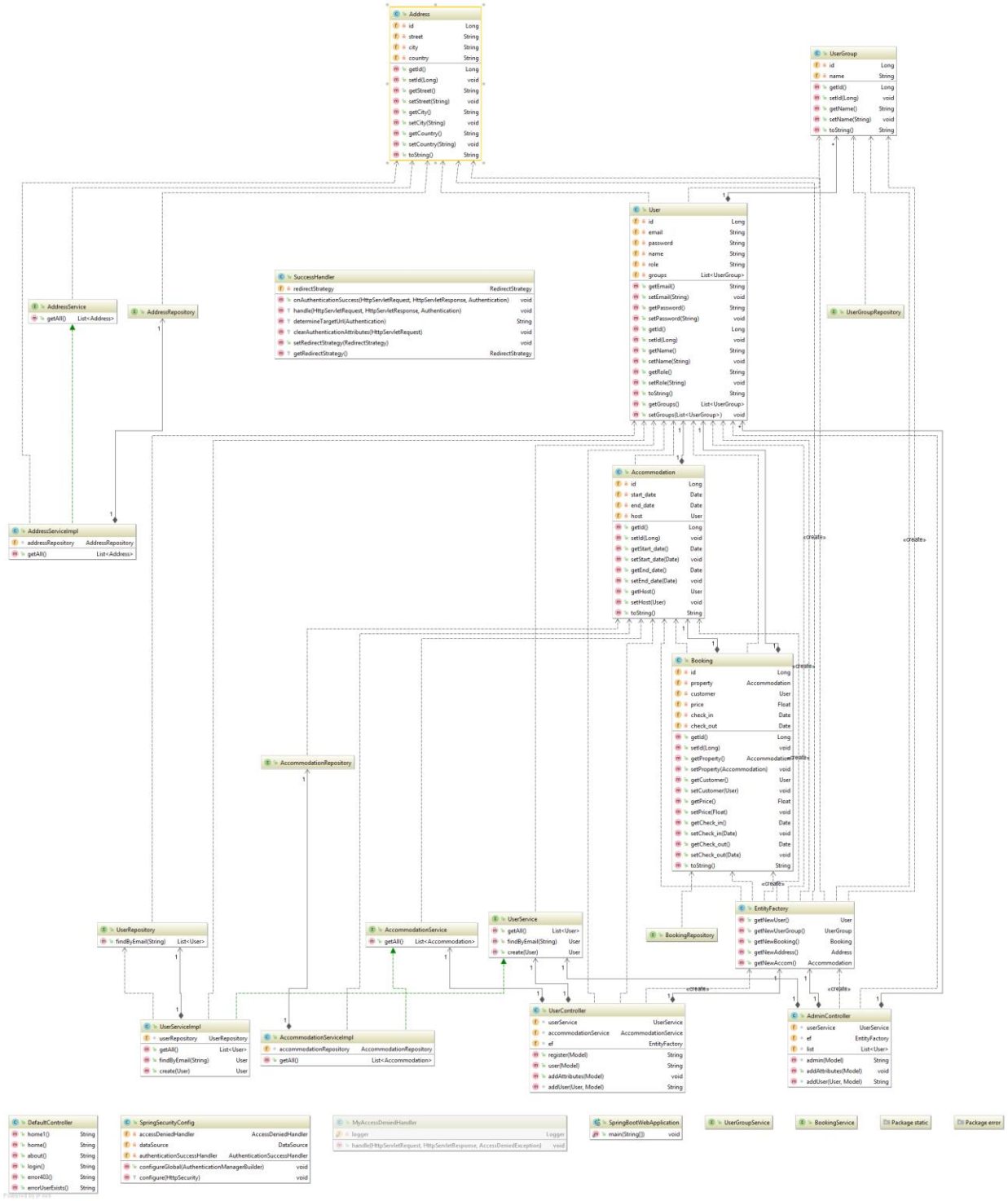
Major changes were made since the last version of this document, hence, attached to this section are the final class diagram and the data model (database) diagram.
Comparing to the previous version, this class diagram is more accurate and actually stands true to the implementation. It represents the Entities with their relationships along with the several Repository and Service objects created. The DAO part of the application was removed from the previous version, along with the MVC part, as the Spring application will be included in the Spring MVC.

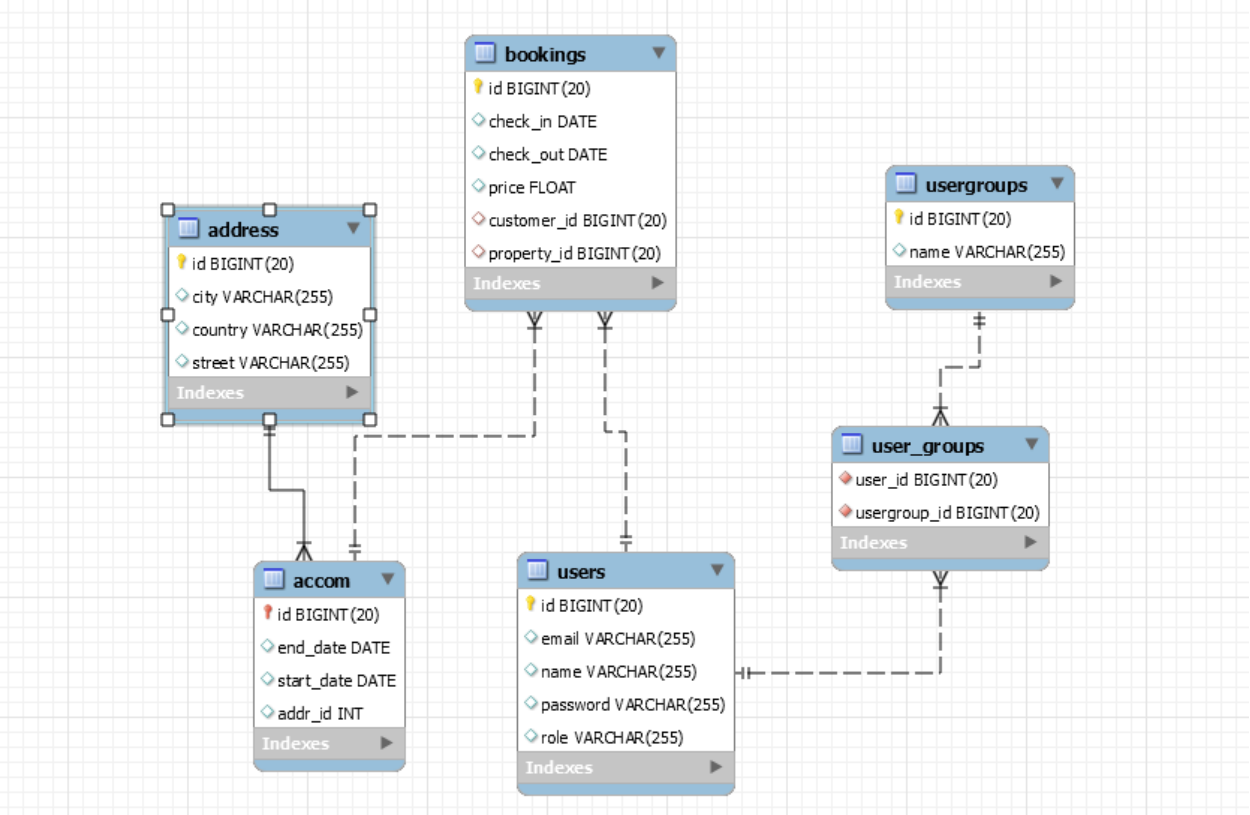
As you can see, two more design patterns were implemented in the application. These two are: Iterator and Factory Design Pattern. The first one of them was implemented through the MyIterator inner class, while the second one was implemented through the RepositoryFactory class.

In class-based programming, the factory method pattern is a creational pattern that uses factory methods to deal with the problem of creating objects without having to specify the exact class of the object that will be created. This is done by creating objects by calling a factory method—either specified in an interface and implemented by child classes, or implemented in a base class and optionally overridden by derived classes—rather

than by calling a constructor.

- Class diagram:





V. Construction and Transition

1. System Testing

System testing was made during development using lots of System.out.println() methods and lots of tears.

Also, testing was done running the application countless times.

2. Future improvements

As a future improvement, the application should have more features, the styling should be better and universities should be contacted for further reasoning.

VI. Bibliography

[1] - https://github.com/buzea/SoftwareDesign2018/