**Design Document**

CIS 4911 – Senior Project

Virtual Queue

**Member:**

Kenneth Kon

Michael Lazo

**Instructor**

Masoud Sadjadi

**Mentor**

[Bernard Parenteau](http://spws-dev.cis.fiu.edu/senior-project-website-v4/user/231)

**Date**

March 1st 2015

Copyright © Florida International University 2014. All Rights Reserved. No part of the Virtual Queue Project or documentation may be reproduced or modified without the express consent of Florida International University, School of Computing & Information Sciences. Redistribution or commercial use is strictly prohibited. The Software is provided “AS IS”, without warranty of any kind, express or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose, if no event shall the authors or copyright holders be liable for any claim, damage or other liability.

**ABSTRACT**

The Design Document gives a better understanding of the Virtual Queue system structure. The reader will be able to capture the idea of how the Virtual Queue system was implemented because the design methodology will be explained, as well as the system architecture and subsystem decomposition, the security and privacy of the system, software and hardware mapping, and persistent data management. Chapter 1 gives basic information about the Virtual Queue (VQ) system, including introduction, problem definition, design methodology used, definitions, acronyms, and overview of the document. Chapter 2 will describe the system decomposition of the VQ by giving an overview of the system, provide a detailed description of the subsystem decomposition, map the hardware and software, identify the persistent data management, and describe security and privacy.

Chapter 3 introduces the detailed design chapter starting with an overview of the behavior and structure of each subsystem, the static and dynamic diagram model, and a description of the code specification. Chapter 4 will have the glossary of terms used in the document, specially the domain specific terms. Chapter 5 contains the appendix of the project with the use case diagrams for the implemented use cases, document class interfaces and diary of meeting and tasks. Finally, Chapter 6 includes any other documents that have been used for reference.

Contents

1. Introduction 5

1.1. Problem definition 5

1.2.   Design methodology used 5

1.3 Terminology - Definitions, acronyms, and abbreviations. 6

1.4.   Overview of document 6

2. System Design (i.e., overall system design) 8

2.1 Overview 8

2.2 Subsystem Decomposition 9

2.3 Hardware-Software Mapping 11

2.4 Persistent Data Management 12

2.5 Security/Privacy 13

3. Detailed Design 15

3.1 Overview 15

3.2 Static models 16

3.3 Dynamic Model 21

3.4 Code Specification 28

4. Glossary 29

5. Appendix 30

5.1 Appendix A - Use case diagram for use cases being implemented 30

5.2 Appendix B - Use cases being implemented (from the RD). 31

5.3 Appendix C – Documented Class Interfaces (code) for the subsystem(s) you will implement and the constraints. 44

5.4 Appendix D – Diary of Meetings 49

6. References 57

# 1. Introduction

The introductory chapter gives some background information about the Virtual Queue system. In the following sections, the problem definition, and scope of the system will be described. Following, the design methodology used is identified. In addition, definitions, acronyms, and abbreviations of terms used in this deliverable will be provided and explained on this chapter. Finally, the chapter will conclude with a brief explanation of what to expect from the following chapters of the document.

## 1.1. Problem definition

When going to a park, or to any other venue that provides multiple recurring rides or events, customers typically wait in line until is time for them to go into the ride or event. This is definitely time consuming for that venue user, since one could be doing something else like be walking around, buying souvenirs, or food, or going perhaps to another ride or event. In addition, nobody likes to wait in line, even if venue users could just sit on a bench to relax, imagine the satisfaction. On the business side, the venue is making money for that specific ride/event, but is losing potential additional sales by having that customer in line rather than walking around the venue visiting other areas of the venue like restaurants or shops.

The creation of the Virtual Queue system is the proposed solution to the problem explained above. The system will provide customers the benefit of enjoying other amenities offered in the venue (including, but not limited to rides, food) instead of waiting in line, event if they can sit and relax, imagine the satisfaction. In addition, the system will keep information about all available rides at the park, and allow the customers to sign in to different events or rides, and the respective waiting time per each one. They will also be able to add rides to their account and delete any ride if they decide to do so; if not, they will be notified as their time for that specific ride approaches. Also, once the time for the ride comes up, venue users will be automatically dequeue from that ride. It will also provide more functionalities, like reset password and create account for a regular user, and the admin will be able to edit, enable and disable venue users’ account.

## 1.2.   Design methodology used

The design methodology used for the VQ system is the Agile Software Development Model. Using this approach, the whole application grows together. It also allows for the requirements and architecture to change as a better understanding of what the project would become is achieved. On the other hand, the VQ system also uses the client server architecture. The used of class diagrams, sequence diagrams, use case diagrams, etc are used to show the design of the VQ system, which are shown later on the document.

## 1.3 Terminology - Definitions, acronyms, and abbreviations.

**Definitions**

* **Guest Users**: Anyone who wants to browse through the site and view offered deals before placing an order.
* **Registered Users**: Users that have already created an online account and can place orders and view previous orders they have requested.
* **System:** The system itself.
* **Theme park/Event**: Amusement park with a unifying setting or idea.

**Acronyms**

* **VQ**: Virtual Queue
* **FIU:** Florida International University
* **SCIS:** School of Computing & Information Sciences

**Abbreviations**

As of right now, there are no abbreviations for this project.

## 1.4.   Overview of document

In chapter 1, the main problem is introduced, along with the design methodology used for the project, definitions, acronyms and abbreviations. In chapter 2 the system decomposition of the VQ system will be described, and a detailed description of the subsystem decomposition, map of the hardware and software, identification of the persistent data management, and a description of security and privacy will be provided.

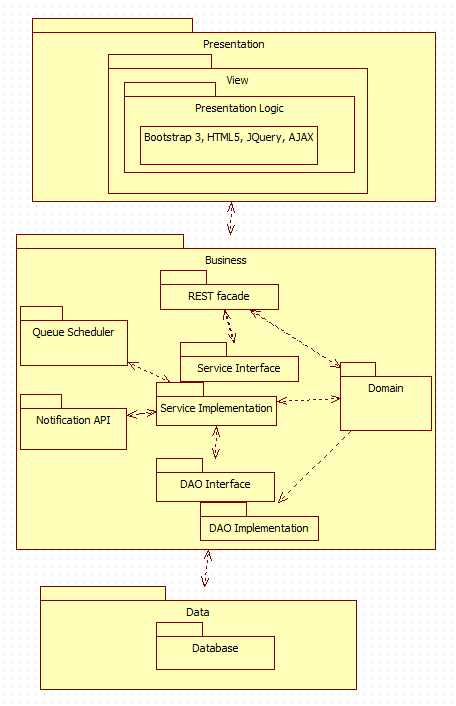
Following, chapter 3 will introduce the detailed design chapter giving an overview of the behavior and structure of each subsystem, the static and dynamic diagram model, and a description of the code specification. Consequently, chapter 4 will have the glossary of terms used in the document, specially the domain specific terms. Chapter 5 will contain the appendix of the project with the use case diagrams for the implemented use cases, document class interfaces and diary of meeting and tasks. Finally, Chapter 6 will include any other documents that have been used for reference.

# 2. System Design (i.e., overall system design)

This chapter will describe the system and subsystem design. It will explain the decomposition of the VQ by giving an overview of the system design architecture. It will provide a detailed description of the subsystem decomposition for each major subsystem. It will cover how the hardware and software are mapped. It will identify the persistent data management that needs to be stored and the structure of the data. Lastly, it will describe security and privacy user authentication processes, encryption of data and all other security parameters being implemented.

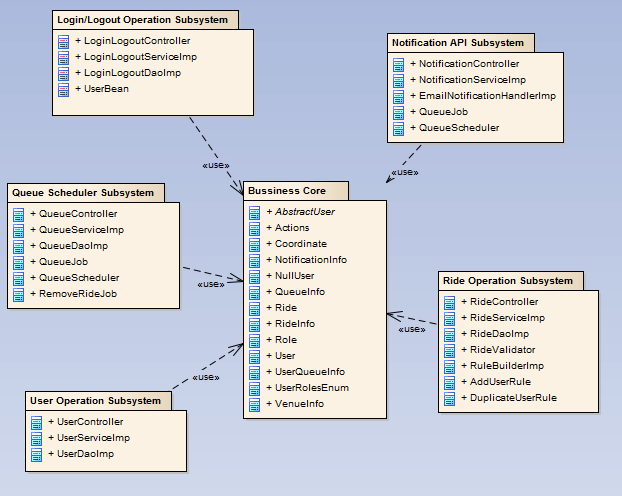
## 2.1 Overview

For the VQ system, a three tier and a multilayer architecture were used. The design approach chosen for the VQ system will allow for horizontal scalability, which will let an increment of number of instances on the server side without affecting the UI tier, and multi-cluster environment, which easily handles high volume traffic. Also, it will allow for a thin client principle that will have a faster UI response since all the presentation logic happens on the browser. In addition, the design architecture was built around a resource presentation logic using REST principles for all the UI and server side integration. The VQ system is composed by different logical components, which are: UI subsystem, User Operations subsystem, Ride, Operations subsystem, Queue Scheduler subsystem, Authorization and Authentication subsystem, and Notification API subsystem that share the domain. Below is a basic illustration showing the different tiers and layers inside the VQ system.



## 2.2 Subsystem Decomposition

The Virtual Queue (VQ) system shall be divided into five main subsystems where they all share the domain. Each subsystem follows a multilayer architecture that allows decoupling each layer using interfaces contracts only to communicate between the layers. All subsystem components are using spring framework, which use inversion of control and dependency injection framework.



User Interface: The user interface subsystem is composed of the web pages that the user will interact with in order to provide info about all rides offer in the venue. It will provide the user the ability to register into the VQ system, signing and see all available rides, select them, queue themselves to different rides and be notified as their time for the rides approaches, they can also de-queue themselves from the ride and logout among others functionalities.

User Operations: The user operation subsystem will group all the artifacts around the user business logic, including all layers architectures of the system such as: controllers, business services, Data access objects, and schedulers. This subsystem also provides user REST interface business services such as: register users on the system, and business validations for users.

Login/Logout Operation: The login operation subsystem will provide system permissions to allow user or admin of the VQ system access to specific resources based on their permissions and roles they are assign to.

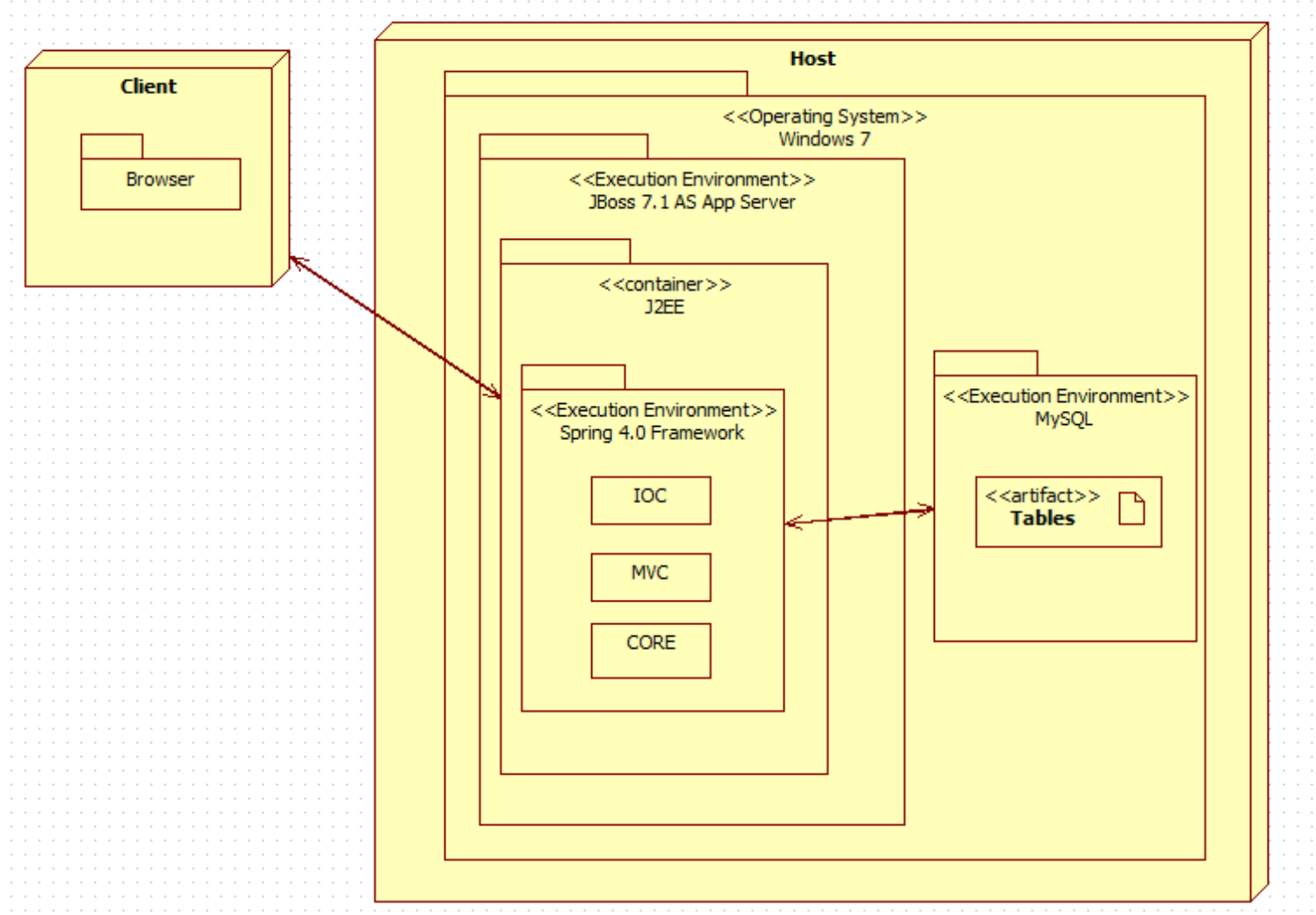
Ride Operations: The ride operation subsystem will group all artifacts around the ride business logic, including all layers architectures of the system such as controllers, business services, data access objects, validations, and domains. This subsystem also provides ride REST interface business services such as: add, delete, remove user from a specific queue, update and select new rides to the VQ system.

Queue Scheduler: The queue scheduler subsystem will schedule jobs based on business requirements. For example, the VQ system will schedule a job based on the ride properties to interact with the notification API to be able to notify users before their time approaches on a ride they queued for. This subsystem uses open source scheduler that provides underlying quartz functionalities.

Notification API: The notification API subsystem will notify the user before their time for a ride they queued for approaches based on a particular implementation, such as: email notification, SMS notification, in this way, the notification API only exposes their interfaces so that it is decoupled from the caller.

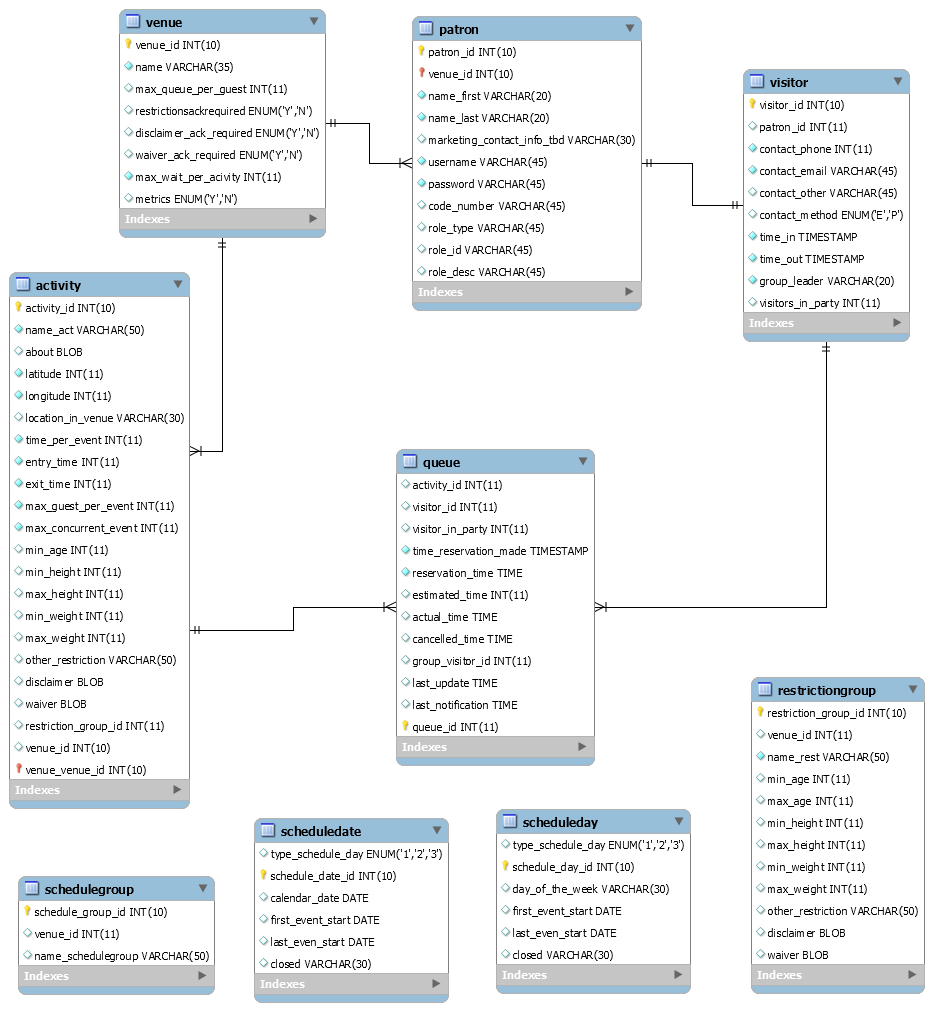
## 2.3 Hardware-Software Mapping

To map the hardware and software used for the Virtual Queue system, a deployment diagram was constructed. It is being hosted on a virtual machine with windows 7 running with JBoss 7.1 application server and Spring 4.0 framework and MySQL database on the FIU SCIS network. Below is the representation of it.



## 2.4 Persistent Data Management

For the VQ system a database was implemented from scratch, taking into account all the information needed to be stored and support each functionality. Below is the EER diagram schema with all the tables and relations.

****

## 2.5 Security/Privacy

For the VQ system all data is password protected. Authentication of each registered user and administrators will be done by checking the database for a registered email, which will be the username as well; also, the table Role and User Role will be checked as well to determine user privileges. Registered users will only have access to their information, and their own data. On the other hand, administrators will have access to all registered users information and results.

# 3. Detailed Design

The detailed design of the VQ system is considered an abstract that translate to source code in a clear way. That is one of the reasons that it has to be detailed and clear. The detailed design chapter will present the system design in a variety of views where each uses a variety of modeling techniques. This chapter is composed for four sections. Section 3.1 is the overview of the chapter; it will give a brief description of the behavior and structure of each subsystem. Section 3.2 contains the static models of each subsystem, and Section 3.3 the dynamic model, which will provide the different diagrams for each subsystem. Section 3.4 delivers the class interfaces and constraints for the main control object in each system.

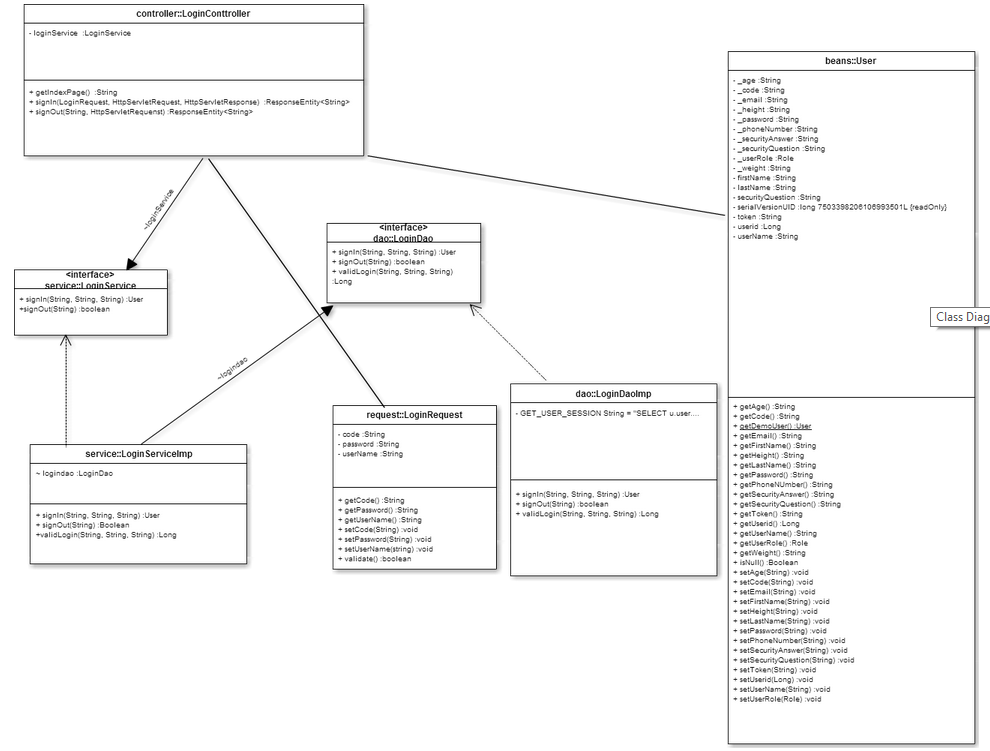
## 3.1 Overview

The VQ system that is to be created will be composed of various subsystems. Each subsystem will be group by a particular functionality. The user operation system components consists of venue unregistered user, venue registered user, and single venue admin who are required to provide an username, which will be their email, a password, and a code that will be provided on the bottom of the user ticket when they get into the venue. It will also provide user REST interface business services such as: register users on the system, sign in and out users, add users to a particular ride, remove user from a specific queue, and business validations for users. Another important system component of the VQ system is the ride operation component ride which have a REST interface business services such as: add, delete, update and select new rides to the VQ system.

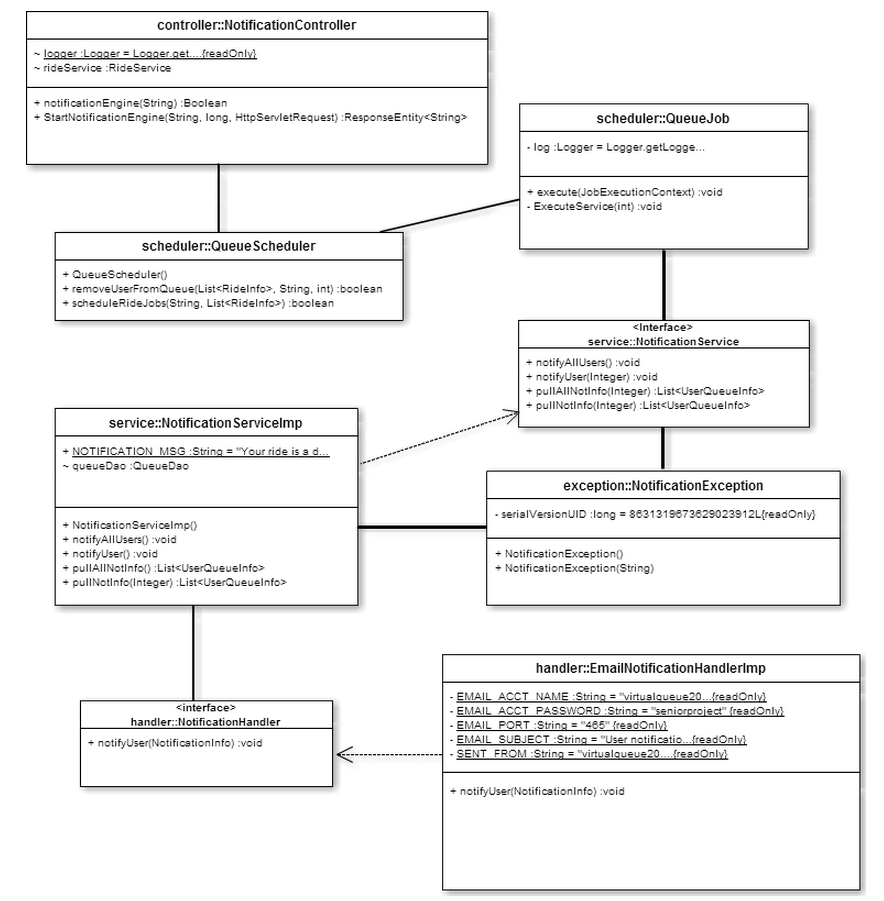
For the VQ system different design patterns were used. The MVC design pattern uses spring MVC framework to provide a web façade for the UI tier. It exposes application resources thru controllers and response request object serving the UI with data. Also, the factory method design pattern is use by the queue scheduler to create jobs, and triggers. In addition, it creates an object without exposing instantiation to the caller. Business Delegate Pattern is used to decouple the web facade layer and business layer. The controller delegates all the responsabilities of business logic to the service layers so that the controllers do not have any responsibility or knowledge on how the business logic is implemented. This pattern allows scalability and low coupling on the VQ system. The REST façade design pattern is also use to allow the VQ to expose all the system functionality thru resources and well define interfaces. Also, it does not stores any state between the communication from the user to the server, which let the system work and perform faster. It resolves issues with versioning and system deployments. The layered architecture design pattern is also used to define each layer based on the principle responsibility principle, which dictates that each layer only knows what its responsibility is and interact with the layer underneath thru the interface only. Lastly, Inversion of Control design pattern is use thru the spring IoC container. All the services and DAO’s are injected thru their interfaces at runtime so that the container is responsible for creating and destroying objects. This makes the VQ system application to have more memory efficiently with better performance.

## 3.2 Static models

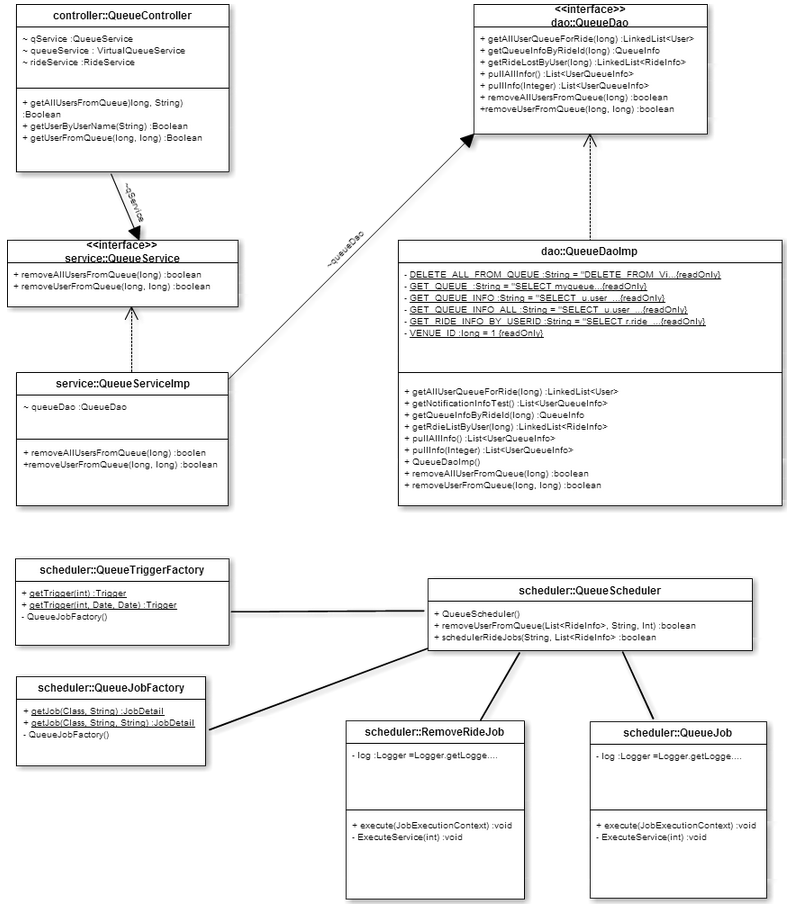
To represent the Virtual Queue system static model, each subsystem explained in the section 2.2 (Subsystem Decomposition) was individually represented into a detailed class diagram that describes the behavior and structure of the system so that it will be easier to understand. Below, the figures from 3.2.1 to 3.2.5 are presented.



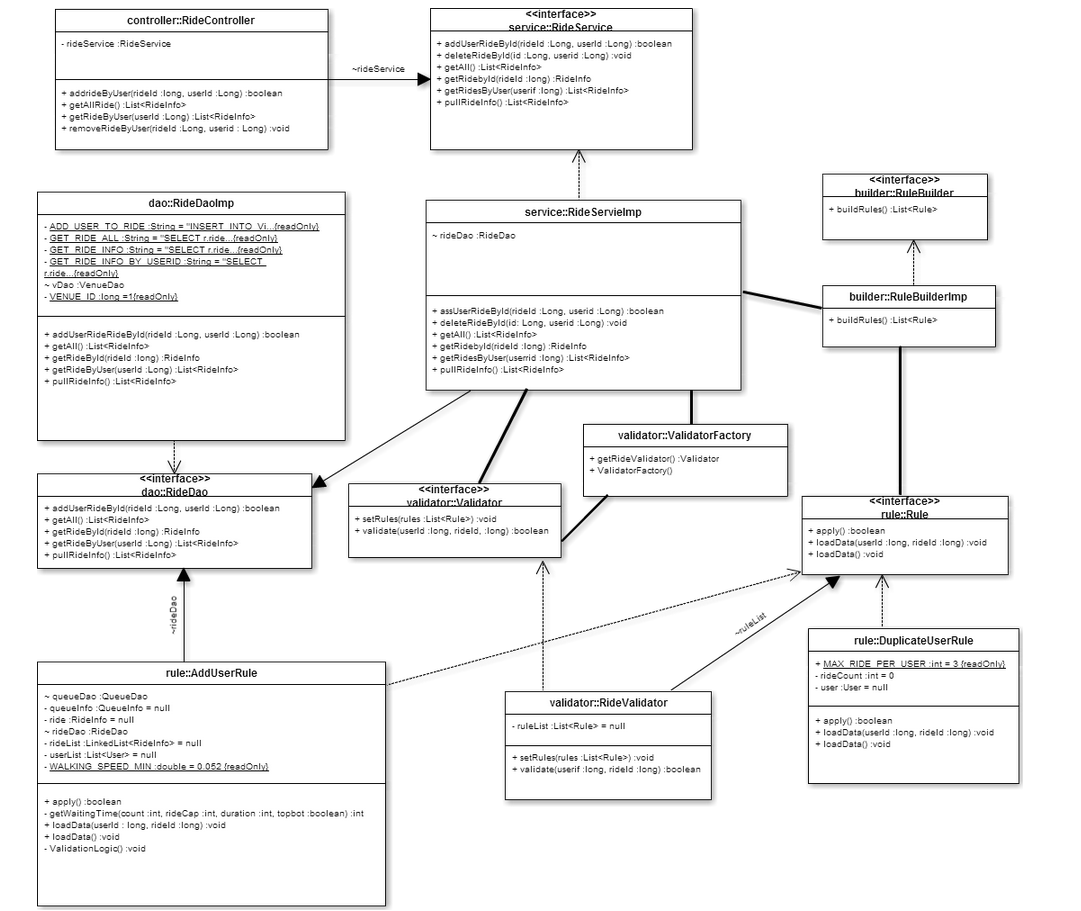
**Fig 3.2.1: Login Operation Subsystem class diagram (part of the whole VQ class Diagram):**



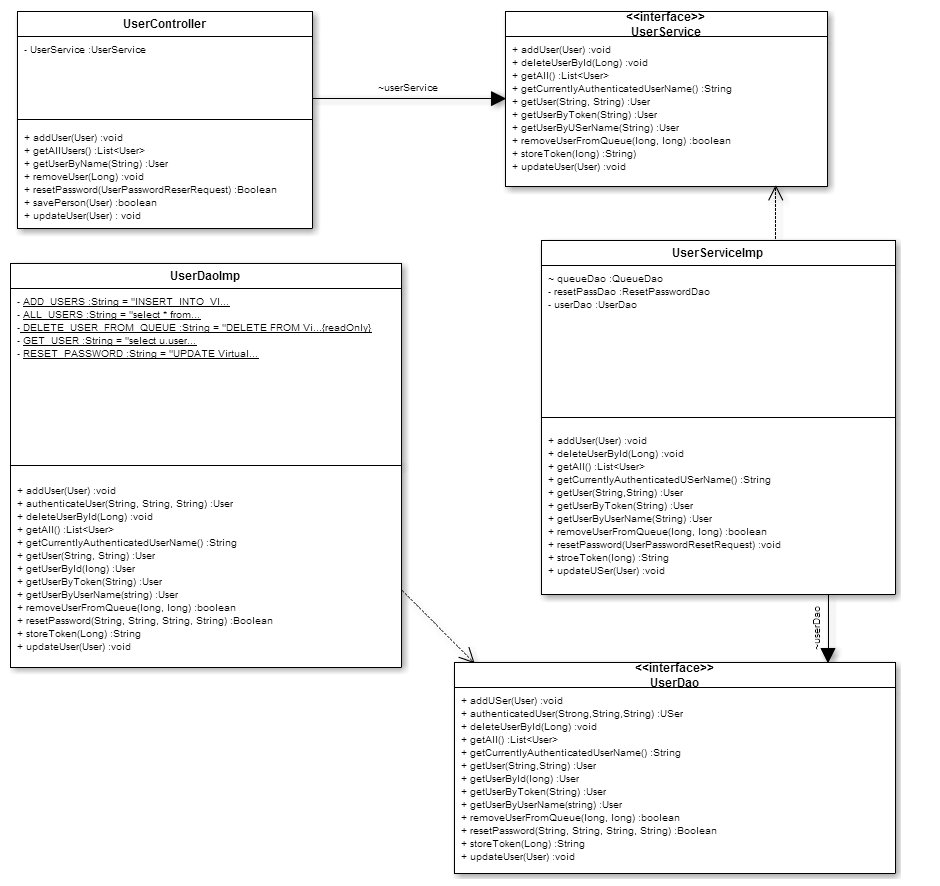
**Fig 3.2.2: Notification API Subsystem class diagram (part of the whole VQ class Diagram)**



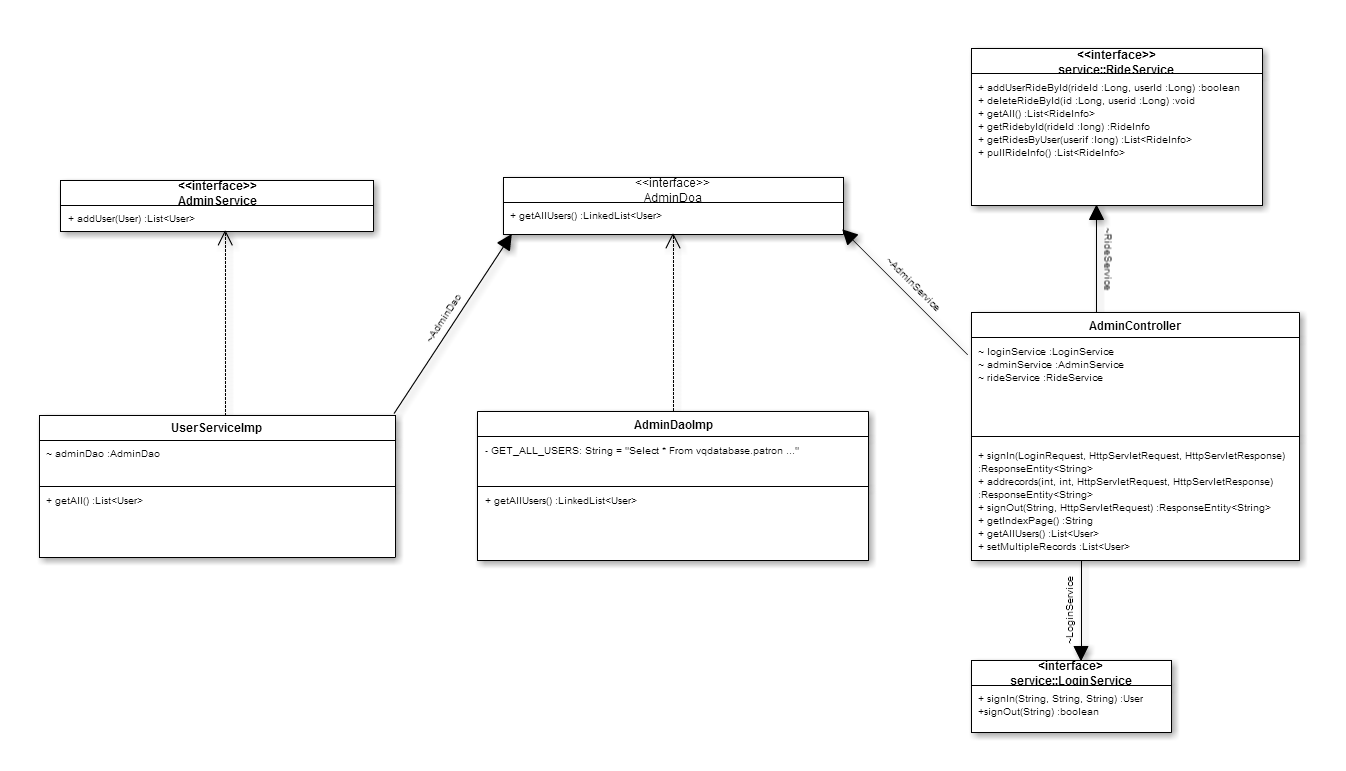
**Fig 3.2.3: Queue Scheduler Subsystem class diagram (part of the whole VQ class Diagram)**



**Fig 3.2.4: Ride Operation Subsystem class diagram (part of the whole VQ class Diagram)**



**Fig 3.2.5: User Operation Subsystem class diagram (part of the whole VQ class Diagram)**



**Fig 3.2.6: Admin Operation Subsystem class diagram (part of the whole VQ class Diagram)**

## 3.3 Dynamic Model

VQ01-Login:



VQ02-Logout:



VQ03-Reset Password:



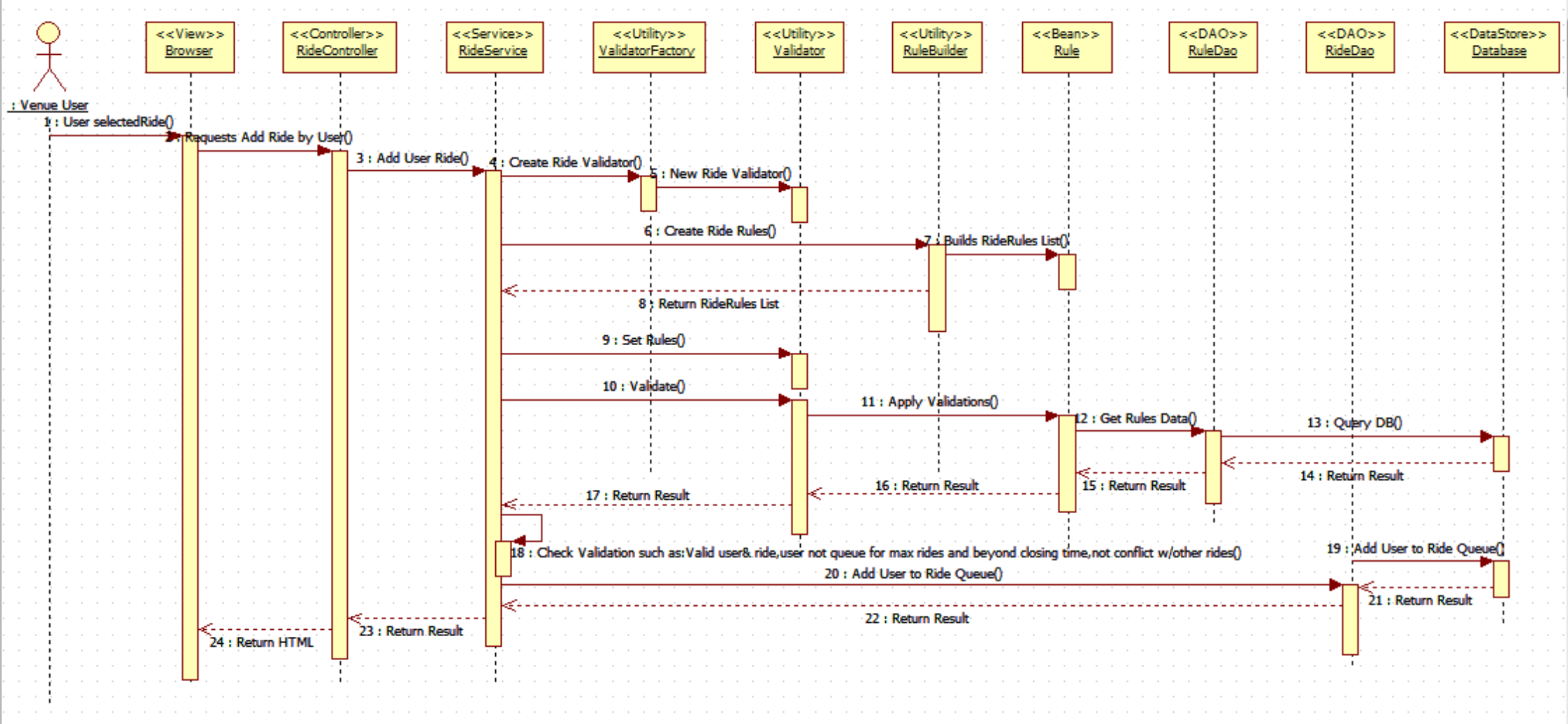
VQ05 - Register New User

****

VQ06 – Select Ride



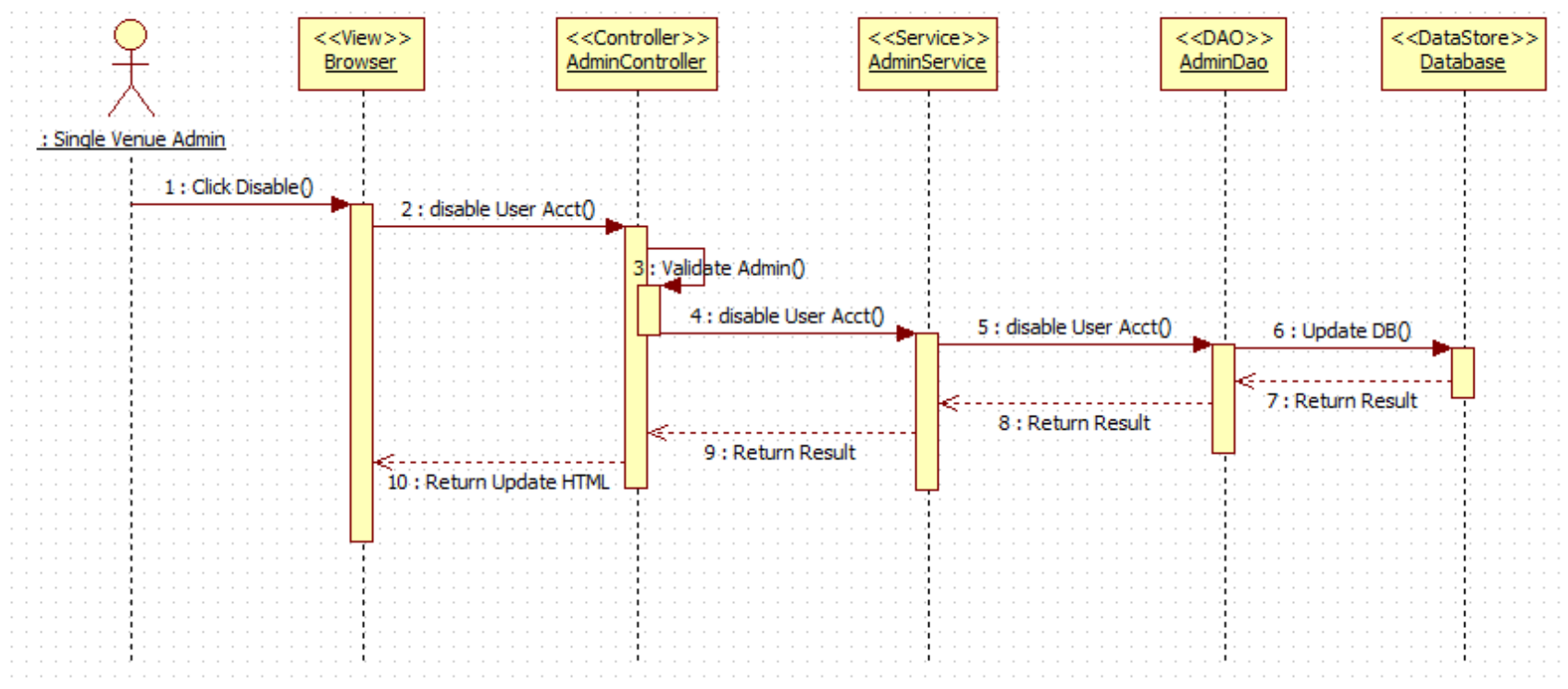
VQ07 – Add Ride (Queue User for Ride)



VQ08 - Edit User's Profile



VQ09 - Disable User's Account



VQ10 - Enable User's Account

****

VQ11 – Delete a Registered Ride (De-queue User from Ride)



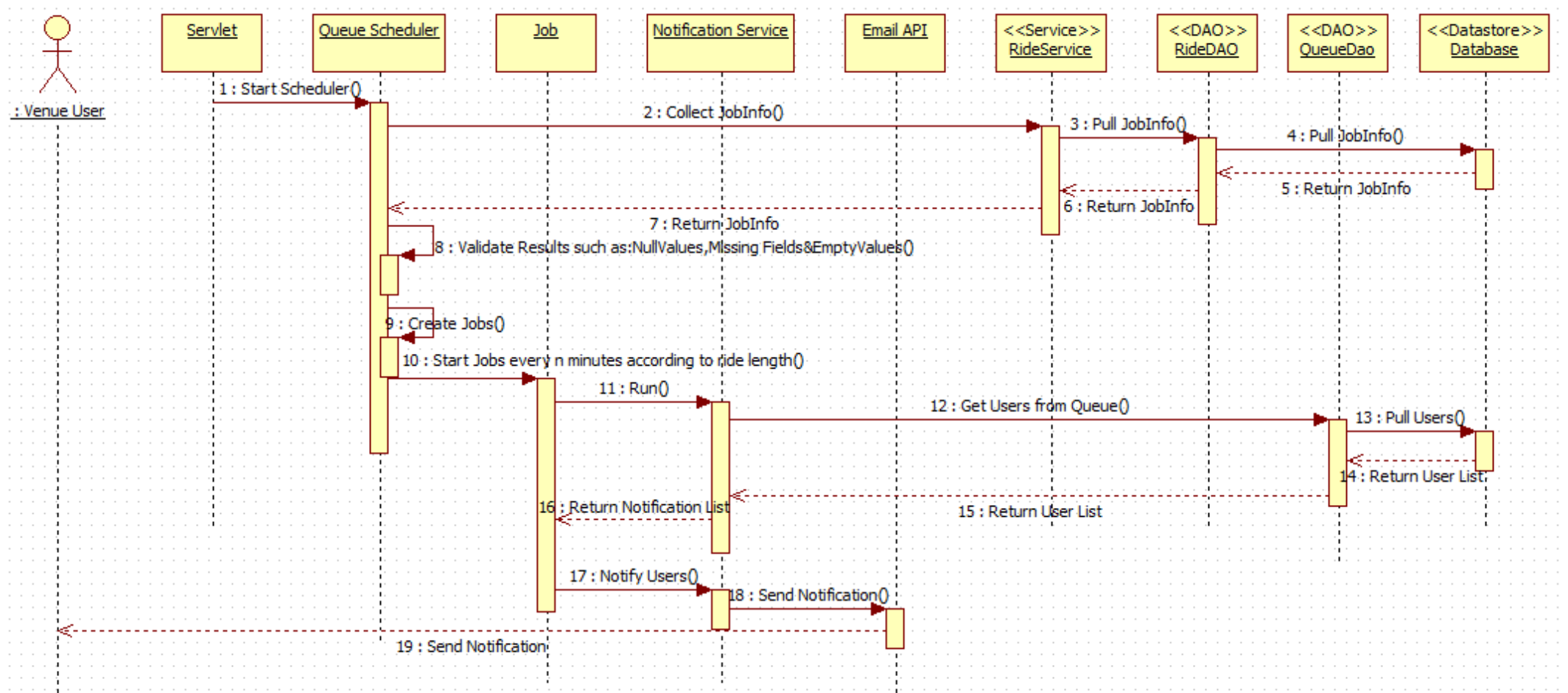
VQ12 – View Available Rides



VQ13 – View User Activity



VQ14 – Receive Notification (before ride time approaches)



VQ15 – Find Wait Times

## VQ16 – Add Queue

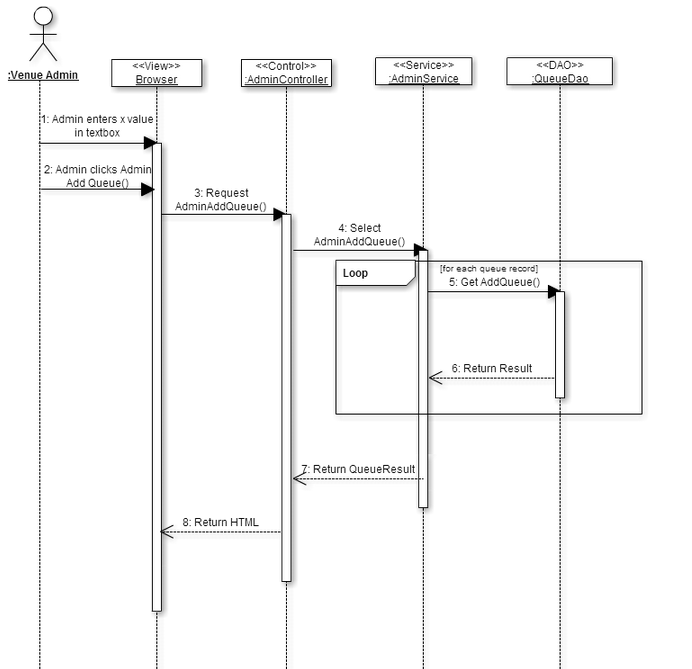
## 

## 

**VQ17 – Visitor DeQueue**



**VQ18 – Simulate Add Ride**



**VQ19 – Simulate Queue for Admin: Dequeue Rides**



## 3.4 Code Specification

The Virtual Queue has a low coupling system because of the extensive use of interfaces that it has. Each controller of the VQ calls the service interface specified on the path. Following, from each service implementation a call to the data access interface of the VQ is made. So, each layer is independent. Refer to appendix C for the list of all interfaces corresponding to each subsystem of the VQ.

# 4. Glossary

**UML**: Stands for Unified Modeling Language. It is a standardized language that is used to model various things within the field of software engineering.

**Theme park:** An amusement park with a unifying setting or idea.

**User Interface (UI):** The way through which a user interacts with the computer system.

**Class Diagram:** A pictorial representation of all the classes in the system

**Object Diagram:** A pictorial representation of an instance of a class with example of how the data of the class will be populated.

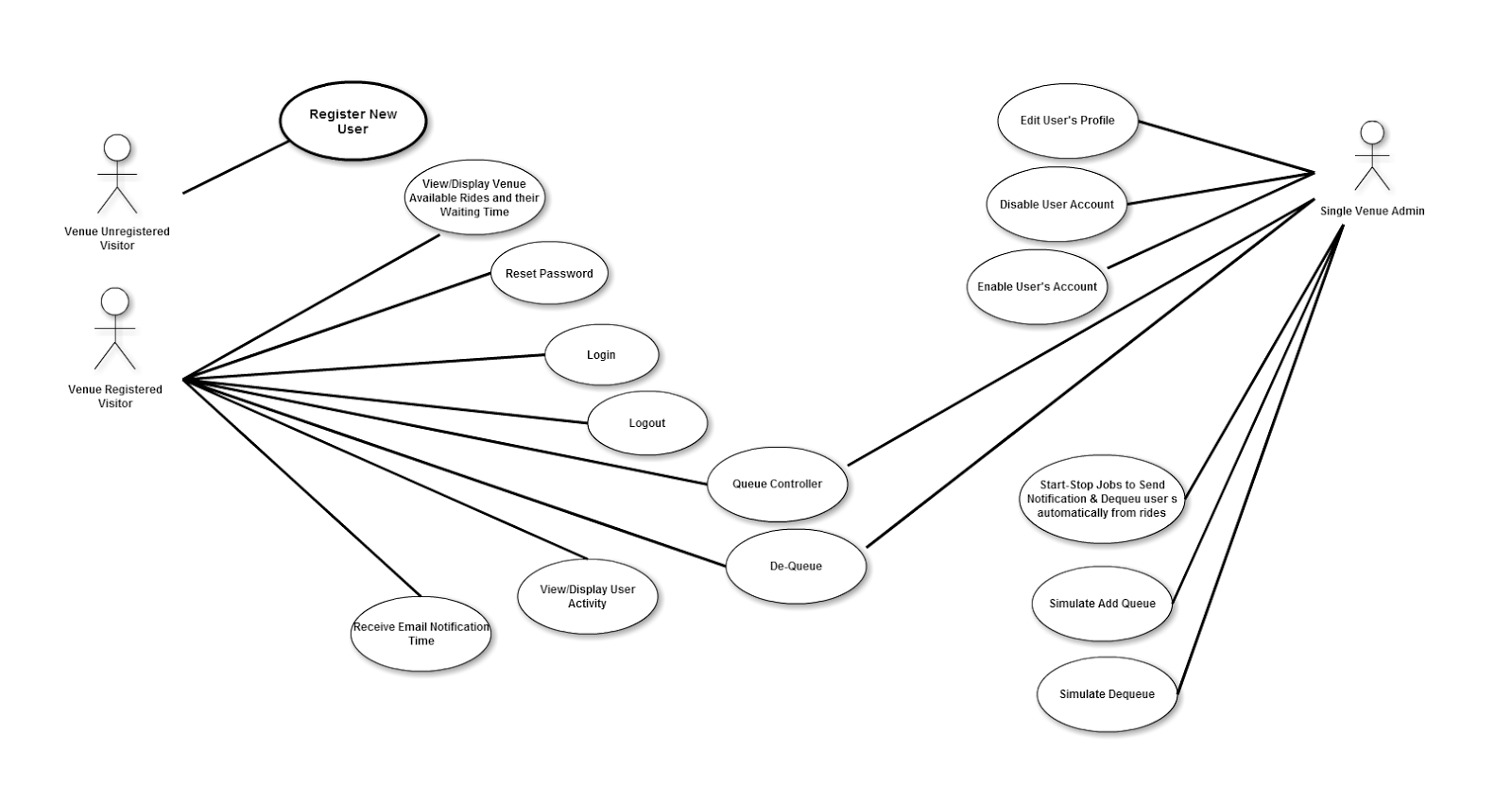
**Sequence Diagram:** A pictorial representation of how processes operate with one another and the user during the course of a specific piece of functionality.

**Task:** A piece of job that serves as a unit of work.

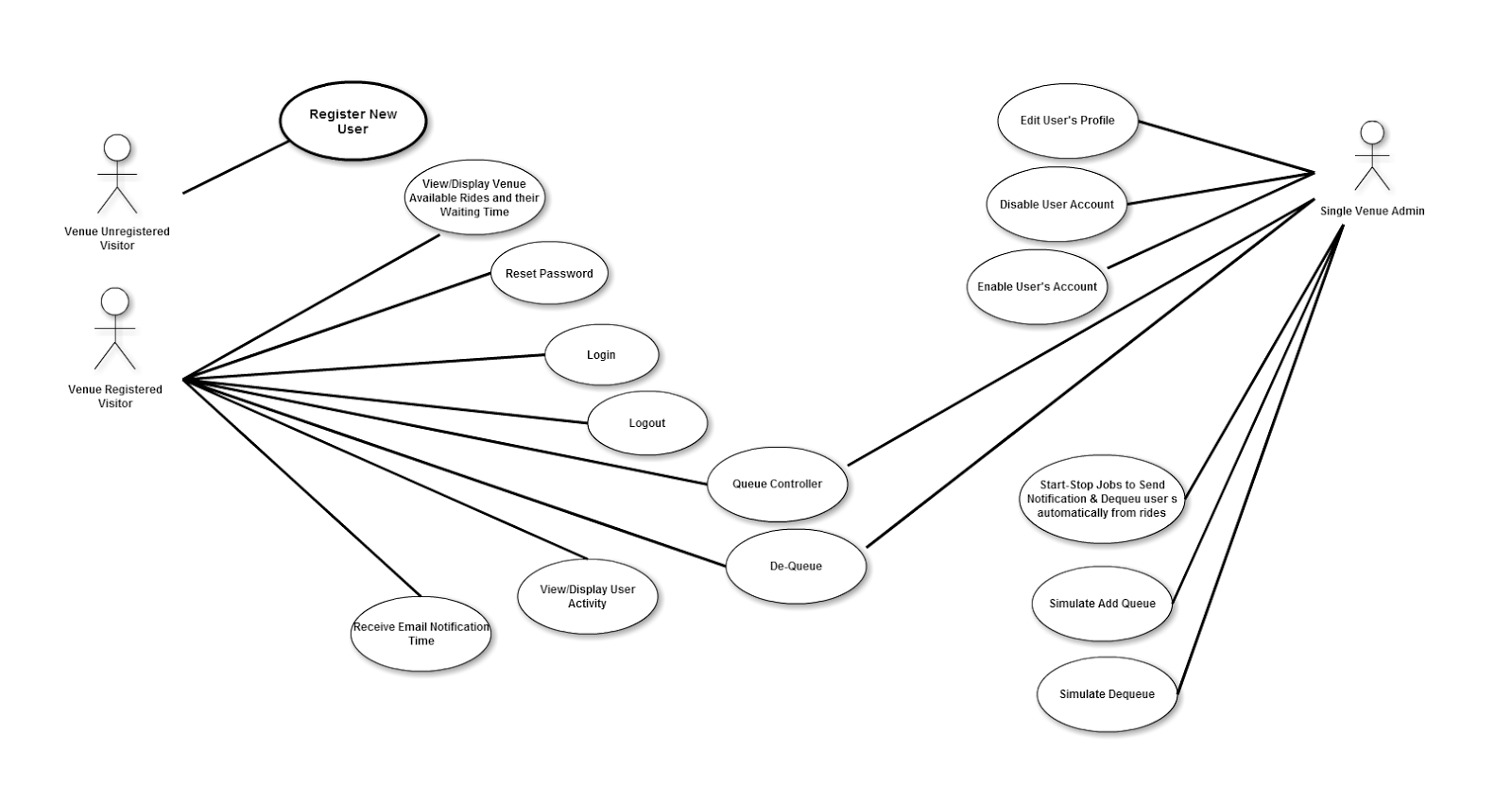
# 5. Appendix

This chapter will contain the use case diagram for the use cases that were implemented based on the requirement document as well as the use case description, the documented classes interfaces and meetings.

## 5.1 Appendix A - Use case diagram for use cases being implemented

****

## 5.2 Appendix B - Use cases being implemented (from the RD).



## 5.3 Appendix C – Documented Class Interfaces (code) for the subsystem(s) you will implement and the constraints.

**Class Interfaces (code) for the subsystem Login Operation:**

**package** com.virtual.queue.dao;

**import** com.virtual.queue.beans.User;

**public** **interface** LoginDao {

**public** Long validLogin(String user,String password, String code);

**public** User signIn(String userName,String password, String code);

**public** **boolean** signOut(String userName);

}

**package** com.virtual.queue.service;

**import** com.virtual.queue.beans.User;

**public** **interface** LoginService {

**public** User signIn(String userName, String password, String code);

**public** **boolean** signOut(String userName);

}

**Class Interfaces (code) for the subsystem Notification API:**

package com.virtual.queue.service;

import java.util.List;

import com.virtual.queue.beans.UserQueueInfo;

public interface NotificationService {

public List<UserQueueInfo> pullNotInfo(Integer rideId);

public List<UserQueueInfo> pullAllNotInfo();

public void notifyUser(Integer rideId) throws Exception;

public void notifyAllUsers() throws Exception;

}

package com.virtual.queue.handler;

import com.virtual.queue.beans.NotificationInfo;

import com.virtual.queue.exception.NotificationException;

public interface NotificationHandler {

public void notifiyUser(NotificationInfo info) throws NotificationException;

}

**Class Interfaces (code) for the subsystem User Operation:**

package com.virtual.queue.dao;

import java.util.List;

import com.virtual.queue.beans.User;

public interface UserDao {

public User getUser(String username, String passwd);

public String getCurrentlyAuthenticatedUserName();

public User getUserByToken(String token);

public String storeToken(Long userId);

public List<User> getAll();

public void addUser(User user);

public void updateUser(User user);

public void deleteUserById(Long id);

public User getUserByUserName(String userName);

public Boolean resetPassword(String userName, String securityAnswer,

String securityQuestion, String newPassword) throws Exception;

public User authenticateUser(String userName,String securityQuestion,String securityAnwser);

public User getUserById(long userId);

public boolean removeUserFromQueue(long userId,long rideId);

}

package com.virtual.queue.service;

import java.util.List;

import com.virtual.queue.beans.User;

import com.virtual.queue.request.UserPasswordResetRequest;

public interface UserService {

public User getUser(String username, String passwd);

public String getCurrentlyAuthenticatedUserName();

public User getUserByToken(String token);

public String storeToken(long userId);

public List<User> getAll();

public void addUser(User user);

public void updateUser(User user);

public void deleteUserById(Long id);

public User getUserByUserName(String userName);

public void resetPassword(UserPasswordResetRequest passwordReset) throws Exception;

boolean removeUserFromQueue(long rideId, long userId);

}

**Class Interfaces (code) for the subsystem Queue Scheduler:**

**package** com.virtual.queue.service;

**public** **interface** QueueService {

**public** **boolean** removeUserFromQueue(**long** rideId,**long** userid);

**public** **boolean** removeAllUsersFromQueue(**long** rideId);

}

package com.virtual.queue.dao;

import java.util.LinkedList;

import java.util.List;

import com.virtual.queue.beans.QueueInfo;

import com.virtual.queue.beans.UserQueueInfo;

import com.virtual.queue.beans.RideInfo;

import com.virtual.queue.beans.User;

public interface QueueDao {

public List<UserQueueInfo> pullInfo(Integer rideId);

public List<UserQueueInfo> pullAllInfo();

public LinkedList<User> getAllUserQueueForRide(long rideId);

public QueueInfo getQueueInfoByRideId(long rideId);

public boolean removeUserFromQueue(long rideId, long userid);

public boolean removeAllUsersFromQueue(long rideId);

public LinkedList<RideInfo> getRideListByUser(long userId) throws Exception;

}

**Class Interfaces (code) for the subsystem Ride Operation:**

package com.virtual.queue.dao;

import java.util.List;

import com.virtual.queue.beans.RideInfo;

import com.virtual.queue.exception.NotificationException;

public interface RideDao {

public List<RideInfo> pullRideInfo() throws NotificationException ;

public List<RideInfo> getRideByUser(Long userId) throws NotificationException;

public RideInfo getRideById(long rideId) throws NotificationException;

public boolean addUserRideById(Long rideId, Long userId);

public List<RideInfo> getAll();

}

package com.virtual.queue.builder;

import java.util.List;

import com.virtual.queue.rule.Rule;

public interface RuleBuilder {

public List<Rule> buildRules();

}

package com.virtual.queue.validator;

import java.util.List;

import com.virtual.queue.rule.Rule;

public interface Validator {

public void setRules(final List<Rule> rules) throws Exception;

public boolean validate(long userId,long rideId);

}

package com.virtual.queue.service;

import java.util.List;

import org.springframework.stereotype.Service;

import com.virtual.queue.beans.Ride;

import com.virtual.queue.beans.RideInfo;

import com.virtual.queue.beans.User;

import com.virtual.queue.exception.NotificationException;

public interface RideService {

public List<RideInfo> getAll();

public void addRide(Ride ride);

public void updateRide(Ride ride);

public void deleteRideById(Long id, Long userid);

public boolean removeRidebyId(String id);

public boolean addUserRideById(Long rideId, Long userid) throws Exception;

public List<RideInfo> pullRideInfo();

public RideInfo getRidebyId(long rideId) throws NotificationException;

public List<RideInfo> getRidesByUser(long userId) throws NotificationException;

}

**package** com.virtual.queue.rule;

**public** **interface** Rule {

**public** **void** loadData(**long** userId,**long** rideId) **throws** Exception;

**public** **boolean** apply();

**void** loadData();

}

## 5.4 Appendix D – Diary of Meetings

**Meeting 1**

**Time:** 6:15PM-7:15PM

**Date:** 1/22/2015

**Members Participated:**

Michael Lazo - scrum master

Kenneth Kon

Bernard Parenteau

**Topic**:

Talking about the requirements with Bernard for Virtual Queue 2.0

**Meeting 2**

**Time:** 6:15PM-7:15PM

**Date:** 1/29/2015

**Members Participated:**

Michael Lazo

Kenneth Kon - scrum master

Bernard Parenteau

**Topic**:

Discussed about the Database layout and the fields of Virtual Queue 2.0

**Meeting 3**

**Time:** 6:15PM-7:15PM

**Date:** 2/03/2015

**Members Participated:**

Michael Lazo - scrum master

Kenneth Kon

Bernard Parenteau

Topic:

Discussed on what User Story we should work on with the Product Owner/Mentor, for Sprint 1.

**Meeting 4**

**Time:** 6:15PM-7:15PM

**Date:** 2/09/2015

**Members Participated:**

Michael Lazo

Kenneth Kon - scrum master

Bernard Parenteau

**Topic**:

Discussed our progress on the Sprint 1, discussed any impediments.

**Meeting 5**

**Time:** 6:15PM-7:15PM

**Date:** 2/13/2015

**Members Participated:**

Michael Lazo

Kenneth Kon - scrum master

Bernard Parenteau

**Topic**:

Discussed Sprint 1 Review, if User Story was satisfy the requirements.

Discussed also discussed the impediments and need to refactor the previous design.

**Meeting 6**

**Time:** 6:15PM-7:15PM

**Date:** 2/17/2015

**Members Participated:**

Michael Lazo

Kenneth Kon

Bernard Parenteau - scrum master

**Topic**:

Discussed Sprint 2 Planning, convince Product owner Team Story has higher priority.

**Meeting 7**

**Time:** 6:15PM-7:15PM

**Date:** 2/28/2015

**Members Participated:**

Michael Lazo

Kenneth Kon - scrum master

Bernard Parenteau

**Topic:**

Discussed Sprint 2 Review, display the new design of the Refactored product.

Discussed also discussed the impediments and need to Refactor the previous design.

**Meeting  8**

**Time:** 6:15PM-7:15PM

**Date:** 3/3/2015

**Members Participated:**

Michael Lazo - Scrum master

Kenneth Kon

Bernard Parenteau - Product Owner

**Topic:**

Get more information on the stories we will be working on for Sprint 3 from our product owner.

# 6. References

Images used:

1. <http://www.plan-family-reunions.com/themeParks.html>
2. http://ru.forwallpaper.com/wallpaper/abstract-circles-patterns-dots-light-colors-bokeh-abstraction-172728.html