**Final Deliverable**

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**

April 27th 2015

Copyright © Florida International University 2015. 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.

**Executive Summary**

The Virtual Queue System will be designed for theme parks and other businesses that have multiple rides or events for which patrons typically wait in line. The idea is that both the theme park and the patron would benefit by the patrons walking around the park (and maybe spending money) rather than standing in line.

This document gives an introduction to the Virtual Queue System. Chapter 1 gives a basic introduction about the Virtual Queue System, including the problem definition, scope of the system, overall development methodology, definitions, acronyms, and abbreviations, and overview of document. Chapter 2 introduces from the feasibility study the description of the current system identifying limitations and constrains, the description of the alternative solutions considered and explanations of why the solution was selected. Chapter 3 includes the project organization, project personnel and hardware and software resources, the identification of tasks, milestones and deliverables and costs of project. Chapter 4 presents the system requirements, including functional and no functional requirements and requirement analysis. Chapter 5 introduces the system design, subsystem decomposition, hardware and software mapping, persistent data management, and security and privacy of the VQ system. Chapter 6 is the detailed design chapter including static model, dynamic model and code specification. Chapter 7 specifies the subsystem test, system tests, and evaluation of tests. Chapter 8 defines the domain terms used in the document and Chapter 9 contains the Appendixes from A to H in the following order: Project Schedule, All use cases with non functional requirements, User Interface Designs, Analysis Models, Design Models, Documented Class Interfaces and constrains, Documented code for test drivers and stubs, and diary of meetings. Finally chapter 10 shows all references that were used.

Table of Content

1. Introduction

Introduce the introduction (one or two paragraphs)

* 1. Problem definition.
  2. Scope of system.
  3. Over all development methodology.
  4. Definitions, acronyms, and abbreviations (at most one page).
  5. Overview of document

1. Feasibility Study
   1. Description of current system. Identify limitations and constraints
   2. Description of alternative solutions considered.
   3. Recommendation with explanation of why the solution was selected.
2. Project Plan
   1. Project Organization
      1. Project Personnel
      2. Hardware and Software Resources
   2. Identification of Tasks, Milestones and Deliverables
   3. Cost of the Project
3. System Requirements

Introduce the proposed system (one or two paragraphs).

* 1. Functional and Nonfunctional Requirements – similar to RD
  2. Requirements Analysis - similar to RD

1. System Design (i.e., overall system design)

Introduce the system design chapter (one or two paragraphs). State the uses cases you are implementing.

* 1. Overview – overview of system decomposition. Identify the architectural patterns used and state why they were selected.
  2. Subsystem Decomposition - describe each of the major subsystems. Identify the use cases (or parts of use cases) associated with each subsystem. Refer to use cases in appendix B.
  3. Hardware and Software Mapping – map subsystems to h/w and s/w.
  4. Persistent Data Management – identify data that needs to be stored e.g., attributes of objects, and primary attributes (may use a table format).
  5. Security/Privacy – describe user authentication processes, encryption of data, and use of firewalls or security servers.

1. Detailed Design

Introduce the detailed design chapter (one or two paragraphs).

* 1. Overview – briefly describe the behavior and structure of each subsystem. Describe the design patterns used and why they were selected.
  2. Static model – detailed description of the structure for each subsystem. May include detailed class diagrams. Appendix C.
  3. Dynamic model – state machine diagram for the main control object in each subsystem. Include the design of the algorithms used in the problem solution. Refinement of the sequence diagram from the analysis model. Appendix D.
  4. Code Specification - describe the class interfaces (attributes and method signatures) and constraint (invariants, pre-condition and post-conditions). Code should be in Appendix E.

1. System Validation

Introduce the system validation chapter (one or two paragraphs).

* 1. Subsystem Tests – test each of the subsystems. This will involve the creation of a test drivers and stubs. Include the code for the test drivers and stubs in Appendix G.
  2. System Tests - For each use case create at least 3 test cases, 2 sunny day and one rainy day, should include security test cases. Each test case should include: test case id, purpose, test setup environment, test inputs, and expected outputs.
  3. Evaluation of Tests – evaluate how successful the tests were. Use a tabular form.

1. Glossary - define terms used in document, especially domain specific terms.
2. Appendix
   1. Appendix A - Project schedule (Gantt chart or PERT chart).
   2. Appendix B – All use cases with nonfunctional requirements.
   3. Appendix C – User Interface designs.
   4. Appendix D – Analysis models (static and dynamic)
   5. Appendix E – Design models (static and dynamic)
   6. Appendix F – Documented Class interfaces (code) and constraints.
   7. Appendix G – Documented code for test drivers and stubs.
   8. Appendix H – Diary of meeting and tasks for the **entire semester**.
3. References
4. 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. Problem definition

Theme parks today as they grow in popularity have one fundamental flaw, the constant waiting in line for each ride. With each of the top tier rides constantly growing in wait time, this only makes matters worse for the average visitor. Standing in line begins to spend the majority of their time rather than sightseeing, exploring, or enjoying some tasty snacks at any store within the park and buying souvenirs. To make matters worse this is not taking into consideration weather conditions the standard waiting line may be under. As for the business aspect, it would be to the theme parks benefit to have more guests spending their time browsing the souvenir shops or food stands rather than spending their time idle while waiting for their turn to get on an attraction. Every minute longer the guests spend active can potentially increase park revenue.

Thus was the creation of the Virtual Queue System, it is the proposed solution to the problem. The system provides the visitors the freedom and flexibility to spend their time doing exactly as they wish while simultaneously waiting in line to ride the attraction they want. As for the Theme Park, it is increasing the traffic flow in between souvenir shops and food courts. Potentially increasing sales and making the theme park overall a more enjoyable place to vacation with. In addition to all of this, the system will also keep track of user information as well as aid in the operation of each ride.

* 1. Scope of system

This project (Virtual Queue) will consist of an online alternative to the standard waiting lines that currently exist within theme parks today. By allowing visitors the ability to queue up for attractions online rather than standing idle waiting their turn. In addition, this system allows administrators the ability to keep track of visitor informations and aid in the automatic run time per attraction.

* 1. Over all development methodology

The design methodology used for the VQ system is done using the Agile Software Development Model. With this approach, the whole application grows incrementally and in stages in order to ensure client satisfaction. It also allows for the requirements and architecture to change as a better understanding of what the project would become is achieved. The VQ system is built on the client server architecture with the foresight of handling high volume traffic. The use of class diagrams, sequence diagrams, use case diagrams, etc aid in the representation for the design of the VQ system, which will be shown later on the document.

* 1. 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. Overview of document

The Final Deliverable covers all aspects of the Virtual Queue project. Chapter 1 covers general information such as problem statement, background information and definitions for this project. Chapter 2 encloses the feasibility study with an overview of the system that will be implemented, and the description and limitations of the current system. Chapter 3 specifies the project and hardware and software requirements. Chapter 4 contains the Appendix, in which miscellaneous information, such as charts and tables are shown. Finally, Chapter 5 contains works used as references.

In chapter 1, the main problem is introduced, along with the design methodology used for the project, definitions, acronyms and abbreviations. Chapter 2 encloses the feasibility study with an overview of the system that will be implemented, and the description and limitations of the current system.

Following, chapter 3 will introduce the project plan of the system, scheduling of the system and the hardware and software requirements. Next, Chapter 4 will contain information on the System Requirements with its functional and functional non requirements. Chapter 5 will contain the detailed System design of the Virtual Queue, containing subsystem decomposition and persistent data management. Chapter 6 will contain system validation. Chapter 8 contains the glossary of the document. Finally, Chapter 9 contains appendix of the document, that contains diagrams for most of the chapters.

1. Feasibility Study
   1. Description of current system (Limitations and Constraints)

The current system requires the customers to buy their tickets and wait in line at the venue until it’s their turn for their event or ride. At some theme parks, there is not even the possibility of buying the ticket for the desired ride online; it has to be purchased by the customers at the theme park. There is no mechanism to allow the user to register and sign in to see the available rides and select the rides they want; or to logout when they decide to do so; or to allow registered users to log in and logout. It lacks of a system that allows the user the ability to be added to their selected rides. It does not have a system with the ability to allow the user to see all the queues he/she registered. There is no mechanism for adding/removing registered users from the queue according to the queuing algorithm, or to allow users to remove themselves from a registered ride. There is no system with the capability of notifying the users before their selected rides/events starts.

* 1. Description of alternative solutions considered

Considering that time spent at a theme park or other venue is sometimes limited due to the fact that customers have to wait in line for rides; the implementation of the VQ system is a good idea.

This new system will provide the customers the ability to see which rides are available and sign in to different ones, as long as there is no time conflict. It will also give the customer the possibility to see all the rides he signed on for and received a notification when their turn is coming on, so they can go to their selected ride. Therefore, providing this system, customers will have a more enjoyable experience at the venue by not having to wait in line, and the venue will also benefit by customers shopping or enjoying other amenities rather than waiting in line.

Overall, the main purpose of the new system is to add functionalities and new features to the current system, which will greatly benefit both the customers and the park/event place.

* 1. Recommendation with explanation of why the solution was selected.

1. Project Plan

This chapter introduces the VQ system a project management perspective. First of all, the project organization that is going to be used will be described, with all roles listed. Following, hardware and software required in order to develop the project will be listed. Finally, tasks, milestones, and deliverables will also be listed.

* 1. Project Organization

For this project, I will be in charge of all functionality and roles of the system.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Roles | Tasks | Periods required | Key Phases |
| Kenneth Kon | • Developer • Scrum master (ALT.) • Test Engineer • Document Editor | • Project Status  • Schedule/Task • Implementation • Testing • Project Documentation | 1/12/15 to 05/01/15 | All |
| Michael Lazo | • Developer • Scrum master (ALT.) • Test Engineer • Document Editor | • Project Status  • Schedule/Task • Implementation • Testing • Project Documentation | 1/12/15 to 05/01/15 | All |

* + 1. Project Personnel

All sections of this project will be assigned evenly among the two members of the group, which will be in charge to develop all the components needed to support the different functionalities.

* + 1. Hardware and Software Resources

**Hardware**

The following are the specification of the computer that will be used for the development of this project:

**Laptop with:**

**- Processing Power:** 1.8 GHz Intel Core i5

**- RAM Memory:** 4 GB 1600 MHz DDR3

**- Available space on hard drive:** 128 GB flash storage

**Other Devices**

- **Input devices**

a) Keyboard- Standard keyboard

b) Mouse- Trackpad mouse

- **Output devices**

a) Monitor display- 13-inch (1440 x 990) Intel HD Graphics 4000 graphics

**Software**

1. **Safari 7.0.6:** last, most updated version of the Safari browser that will be used to test Virtual Queue.
2. **Google Chrome 37.0:** last, most updated version of the Google Chrome browser that will be used to test Virtual Queue.
3. **Spring Framework 4.0:** last, most updated version of Spring Framework, an MVC-based, Java framework used for development. It will be used to develop the front-end and back-end of Virtual Queue.
4. **Quartz Scheduler**: open source job scheduling library that can be integrated within virtually any Java application. It will be used to create, start, stop jobs, and send notifications and delete users from rides.
5. **Bootstrap 3**: most popular HTML, CSS, and JS framework for developing responsive, mobile first projects on the web. It will be used for the front-end design development.
6. **Spring JDBC template**: make database call access to MySQL database.
7. **JQuery**: Fast, small, and feature-rich JavaScript library. It will be used to connect the front-end development with the back-end development.
8. **MySQL**: database that will be used to store data for the Virtual Queue system.
9. **StarUML:** UML software platform that will be used to create diagrams for the document.
10. **VMWare:** virtual machine software that will be used for the deployment of the software.
11. **Eclipse Luna:** IDETool software used to write the java classes for the Virtual Queue system.
12. **Apache Tomcat 8.0**: Java J2EE Application Server used to deploy and run the Virtual Queue system.
    1. Identification of Tasks, Milestones and Deliverables

Below is a list of all different tasks, milestones, and deliverables for the project, which can be changed according to the development of the project:

|  |  |
| --- | --- |
| Milestones | Task & Deliverables |
| Documentation | * Feasibility Study * Requirement Document * Design Document * Deliverable 4 |
| Environment Setup | * IDE Installation (Eclipse) * JDK 8 Install * StartUML Install * Source Tree for Github * Jboss Installation and Configuration * Maven Project Setup * Spring framework setup * Project dependencies and third parties libraries * Dependency injections setup * MySQL install |
| * Development Tasks: | |
| UI Design | * Create main page template view * Create login page template view * Create My Account page template view * Create Add ride to user page template view * Create Select User template view * Create reset password page template view * Create Add User page template view |
| Database Design and Implementation | * Create Tables * Create tables relationships and constrains * Create EER diagram |
| Main Page Functionality | * Add dynamic content to main page. (JQuery, Ajax calls) * Add server side implementation to main page. * Create main controller with URL definitions and contracts * Create main service interface/implementation * Create main DAO for data access. |
| Login and Logout functionalities | * Add dynamic content to login page (JQuery, Ajax calls) * Server Side Implementation. * Create login controller with URL definitions and UI contracts * Login controller unit test. * Create user Bean. * Unit test to user bean. * Add user information to session when user log * Create login service interface/Implementation * Login service unit test * Create login DAO to facilitate data access * DAO unit test. * Add logout controller with URL contract * Logout controller unit test. * Remove user information from session data. * Unit test. |
| Add User Functionality | * Add dynamic content to user page (JQuery, Ajax calls) * Server Side Implementation. * Create User controller with URL definitions and UI contracts * User controller unit test. * Add User information to session when user log * Create User service interface/Implementation * User service unit test * Create User DAO to facilitate data access * DAO unit test. |
| Reset Password Functionality | * Add dynamic content to user page to reset password (JQuery, Ajax calls) * Server Side Implementation. * Add method User controller with URL definitions and UI contracts for password reset. * User controller unit test. * Add functionality to User service interface/Implementation * User service unit test * Add functionality to User DAO to facilitate data access * DAO unit test. |
| User Ride Functionality | * Add dynamic content to user Ride page (JQuery, Ajax calls). * Add UI data validations. * Server Side Implementation. * Create login controller with URL definitions and UI contracts * Add controller unit test * Create Ride Bean. * Unit test to ride bean. * Add user information to session when user log * Create user ride interface/Implementation * Unit test for service interfaces * Create User Ride DAO to facilitate data access * DAO unit test. * Add User Validator interface when registering for a ride. * Validator unit test. * Add User validator implementation with rule definitions. * Validator implementation unit test for each method. * =================================== * Add Ride Registration Business Rules Interface * Add Ride Registration Business Rule implementation. * Add Unit test for Ride Registration * Add Rules Interfaces * Add Rules Implementations. * Add unit test to Rules implementations. * Add Rule Builder Class. * Unit test to Rule Builder. |
| Admin functionalities | * Create UI design for Login and Search modal for admin user. * Server Side Implementation. * Add method Admin controller with URL definitions and UI contracts for login. * Add functionality to Admin service interface/Implementation * Add functionality to Admin DAO to facilitate data access. |

* 1. Cost of the Project

There is no cost to work on this project. All the tools used are free.

1. System Requirements

The proposed system (Virtual Queue v2.0) requires the visitors log in the VQ in order to select the desired attractions. In turn it also allows the visitors the privilege to remove themselves from any given attraction if they no longer desire to be. The VQ grants administrative rights to admin users; Allowing them to automatically run attractions that remove visitor records from any given attraction while simultaneously sending notifications to each visitor whose turn is approaching via email. Administrators can also manipulate any ride instance in order to account for changes to any particular ride if need be, Administrators can also add visitor records to any particular ride. This chapter will include the functional and non-functional requirements of the system and the requirements analysis phase of the system.

* 1. Functional and Nonfunctional Requirements – similar to RD

Allow unregistered users to register.

* **Usability**: No previous training time. System is simple and easy to follow.
* **Reliability**: The system should perform correctly 99% of the time.
* **Performance**: The system should be sent and saved within 3 seconds.
* **Supportability**: The system should be easy to maintain, make appropriate changes, and be correctly handled by IE, Mozilla, Chrome and Safari.

Allow users to validate their account.

* **Usability**: No previous training time. System is simple and easy to follow.
* **Reliability**: The system should perform correctly 99% of the time.
* **Performance**: The system should be sent and saved within 3 seconds.
* **Supportability**: The system should be easy to maintain, make appropriate changes, and be correctly handled by IE, Mozilla, Chrome and Safari.

Allow registered users to delete their accounts.

* **Usability**: No previous training time. System is simple and easy to follow.
* **Reliability**: The system should perform correctly 99% of the time.
* **Performance**: The system should be sent and saved within 3 seconds.
* **Supportability**: The system should be easy to maintain, make appropriate changes, and be correctly handled by IE, Mozilla, Chrome and Safari.

Allow registered users to sign in to their accounts.

* **Usability**: No previous training time. System is simple and easy to follow.
* **Reliability**: The system should perform correctly 99% of the time.
* **Performance**: The system should be sent and saved within 3 seconds.
* **Supportability**: The system should be easy to maintain, make appropriate changes, and be correctly handled by IE, Mozilla, Chrome and Safari.

Allow registered users to logout of their account.

* **Usability**: No previous training time. System is simple and easy to follow.
* **Reliability**: The system should perform correctly 99% of the time.
* **Performance**: The system should be sent and saved within 3 seconds.
* **Supportability**: The system should be easy to maintain, make appropriate changes, and be correctly handled by IE, Mozilla, Chrome and Safari.

Allow users to reset their password.

* **Usability**: No previous training time. System is simple and easy to follow.
* **Reliability**: The system should perform correctly 99% of the time.
* **Performance**: The system should be sent and saved within 3 seconds.
* **Supportability**: The system should be easy to maintain, make appropriate changes, and be correctly handled by IE, Mozilla, Chrome and Safari.

Allow registered users to have access to the available rides/events.

* **Usability**: No previous training time. System is simple and easy to follow.
* **Reliability**: The system should perform correctly 99% of the time.
* **Performance**: The system should be sent and saved within less than a second.
* **Supportability**: The system should be easy to maintain, make appropriate changes, and be correctly handled by IE, Mozilla, Chrome and Safari.

Allow registered users to be added to their selected rides/events.

* **Usability**: No previous training time. System is simple and easy to follow.
* **Reliability**: The system should perform correctly 99% of the time.
* **Performance**: The system should be sent and saved within less than a second.
* **Supportability**: The system should be easy to maintain, make appropriate changes, and be correctly handled by IE, Mozilla, Chrome and Safari.

Allow registered users to see all the rides/events they signed on for.

* **Usability**: No previous training time. System is simple and easy to follow.
* **Reliability**: The system should perform correctly 99% of the time.
* **Performance**: The system should be sent and saved within less than a second.
* **Supportability**: The system should be easy to maintain, make appropriate changes, and be correctly handled by IE, Mozilla, Chrome and Safari.

Allow registered users to delete themselves from registered rides/events.

* **Usability**: No previous training time. System is simple and easy to follow.
* **Reliability**: The system should perform correctly 99% of the time.
* **Performance**: The system should be sent and saved within 3 seconds.
* **Supportability**: The system should be easy to maintain, make appropriate changes, and be correctly handled by IE, Mozilla, Chrome and Safari.

Allow the user to be notified before their rides/events time occurs.

* **Usability**: No previous training time. System is simple and easy to follow.
* **Reliability**: The system should perform correctly 99% of the time.
* **Performance**: The system should be sent and saved within 3 seconds.
* **Supportability**: The system should be easy to maintain, make appropriate changes, and be correctly handled by IE, Mozilla, Chrome and Safari.

Allow the system to add and/or remove registered users from their selected rides.

* **Usability**: No previous training time. System is simple and easy to follow.
* **Reliability**: The system should perform correctly 99% of the time.
* **Performance**: The system should be sent and saved within 3 seconds.
* **Supportability**: The system should be easy to maintain, make appropriate changes, and be correctly handled by IE, Mozilla, Chrome and Safari.

Allow the system to send notification to registered user before ride time added approaches.

* **Usability**: No previous training time. System is simple and easy to follow.
* **Reliability**: The system should perform correctly 99% of the time.
* **Performance**: The system should be sent and saved within 3 seconds.
* **Supportability**: The system should be easy to maintain, make appropriate changes, and be correctly handled by IE, Mozilla, Chrome and Safari.

Allow the system to store and retrieve information regarding to the rides/events.

* **Usability**: No previous training time. System is simple and easy to follow.
* **Reliability**: The system should perform correctly 99% of the time.
* **Performance**: The system should be sent and saved within 3 seconds.
* **Supportability**: The system should be easy to maintain, make appropriate changes, and be correctly handled by IE, Mozilla, Chrome and Safari.

Allow the system to check for duplicates registration or multiple registrations for same user.

* **Usability**: No previous training time. System is simple and easy to follow.
* **Reliability**: The system should perform correctly 99% of the time.
* **Performance**: The system should be sent and saved within 3 seconds.
* **Supportability**: The system should be easy to maintain, make appropriate changes, and be correctly handled by IE, Mozilla, Chrome and Safari.

Allow the system to handle network connectivity issues.

* **Usability**: No previous training time. System is simple and easy to follow.
* **Reliability**: The system should perform correctly 99% of the time.
* **Performance**: The system should be sent and saved within 3 seconds.
* **Supportability**: The system should be easy to maintain, make appropriate changes, and be correctly handled by IE, Mozilla, Chrome and Safari.

Allow the user to be notified as soon as they register for their rides/events time occurs.

* **Usability**: No previous training time. System is simple and easy to follow.
* **Reliability**: The system should perform correctly 99% of the time.
* **Performance**: The system should be sent and saved within 3 seconds.
* **Supportability**: The system should be easy to maintain, make appropriate changes, and be correctly handled by IE, Mozilla, Chrome and Safari.

Allow the admin to remove visitors from the queue as soon as they finish their events time occurs.

* **Usability**: No previous training time. System is simple and easy to follow.
* **Reliability**: The system should perform correctly 99% of the time.
* **Performance**: The system should be sent and saved within 3 seconds.
* **Supportability**: The system should be easy to maintain, make appropriate changes, and be correctly handled by IE, Mozilla, Chrome and Safari.

Allow the admin to add visitors to the queue for a specific event.

* **Usability**: No previous training time. System is simple and easy to follow.
* **Reliability**: The system should perform correctly 99% of the time.
* **Performance**: The system should be sent and saved within 3 seconds.
* **Supportability**: The system should be easy to maintain, make appropriate changes, and be correctly handled by IE, Mozilla, Chrome and Safari.

Allow the admin to add visitors to the queue for a specific event.

* **Usability**: No previous training time. System is simple and easy to follow.
* **Reliability**: The system should perform correctly 99% of the time.
* **Performance**: The system should be sent and saved within 3 seconds.
* **Supportability**: The system should be easy to maintain, make appropriate changes, and be correctly handled by IE, Mozilla, Chrome and Safari.

Allow the admin to edit ride instances to manipulate time calculations for a specific event.

* **Usability**: No previous training time. System is simple and easy to follow.
* **Reliability**: The system should perform correctly 99% of the time.
* **Performance**: The system should be sent and saved within 3 seconds.
* **Supportability**: The system should be easy to maintain, make appropriate changes, and be correctly handled by IE, Mozilla, Chrome and Safari.
  1. Requirements Analysis - similar to RD

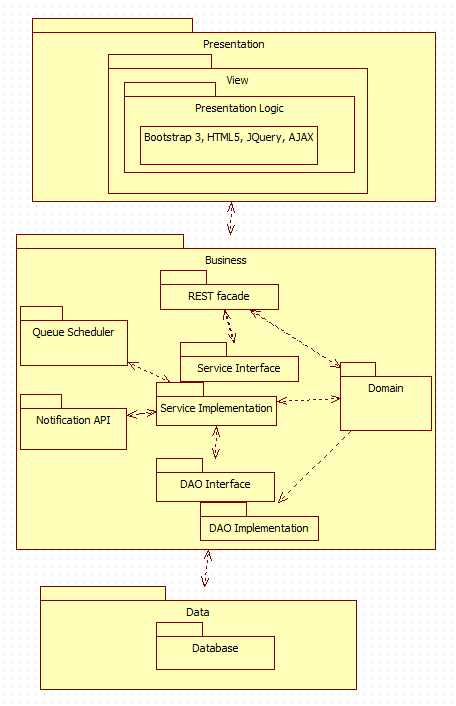
The VQ allows for any individual person to vacation over to a Theme Park and take his/her entire family. While maximizing the time they can spend together by not having to spend all those long hours waiting to get on an attraction. A family of four can simultaneously register together for up to three attractions while relaxing in the food court over lunch. The VQ also allows the Theme Park to handle larger traffic flow of visitors maximize profits and create a better park that everyone can enjoy.

1. 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.

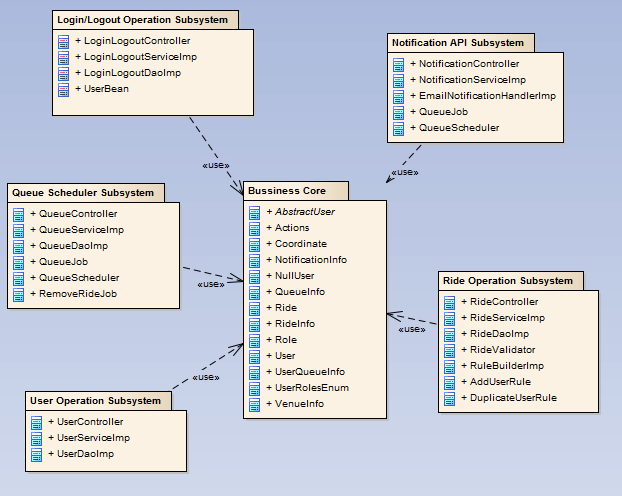
* 1. Overview – overview of system decomposition. Identify the architectural patterns used and state why they were selected.

For the VQ system, a three tier and a multilayer architecture was 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.



* 1. 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.

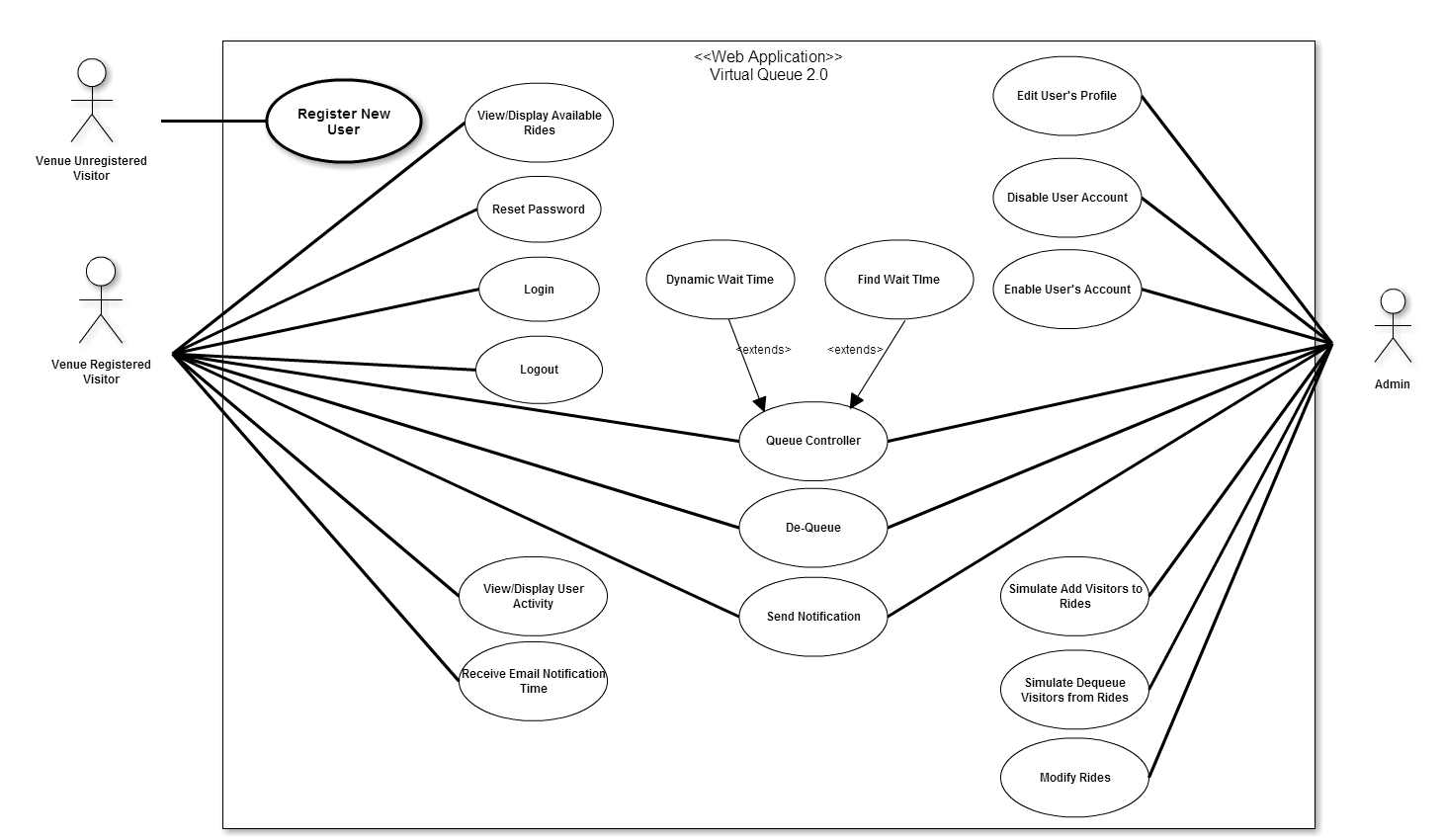
Login/Logout Operation: The login operation subsystem will provide system permissions to allow user or admin of the VQ system access to specific resources based on their permissions and roles they are assign to.

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.

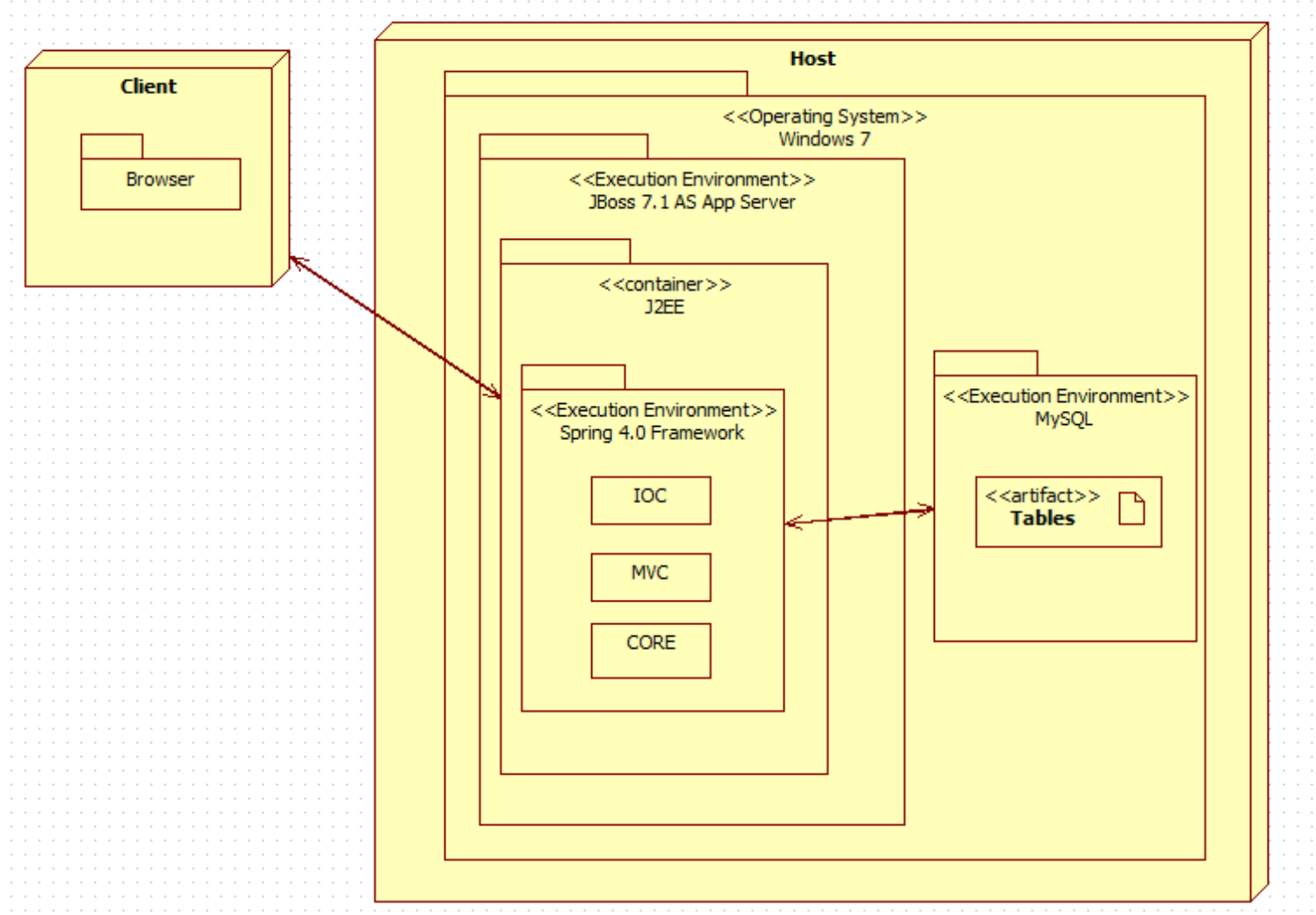
Notification API: The notification API subsystem will notify the user before their time for a ride they queued for approaches based on a particular implementation, such as: email notification, SMS notification, in this way, the notification API only exposes their interfaces so that it is decoupled from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

The use case diagram describes the list of steps that defines the interaction between the three types of users displayed in the diagram: venue guest user, venue registered user, and single venue admin and the system. They all work together to accomplish the goal of the proposed system. Below is the Use Cases Diagrams using UML for specific details with the specific functionalities that were implemented.



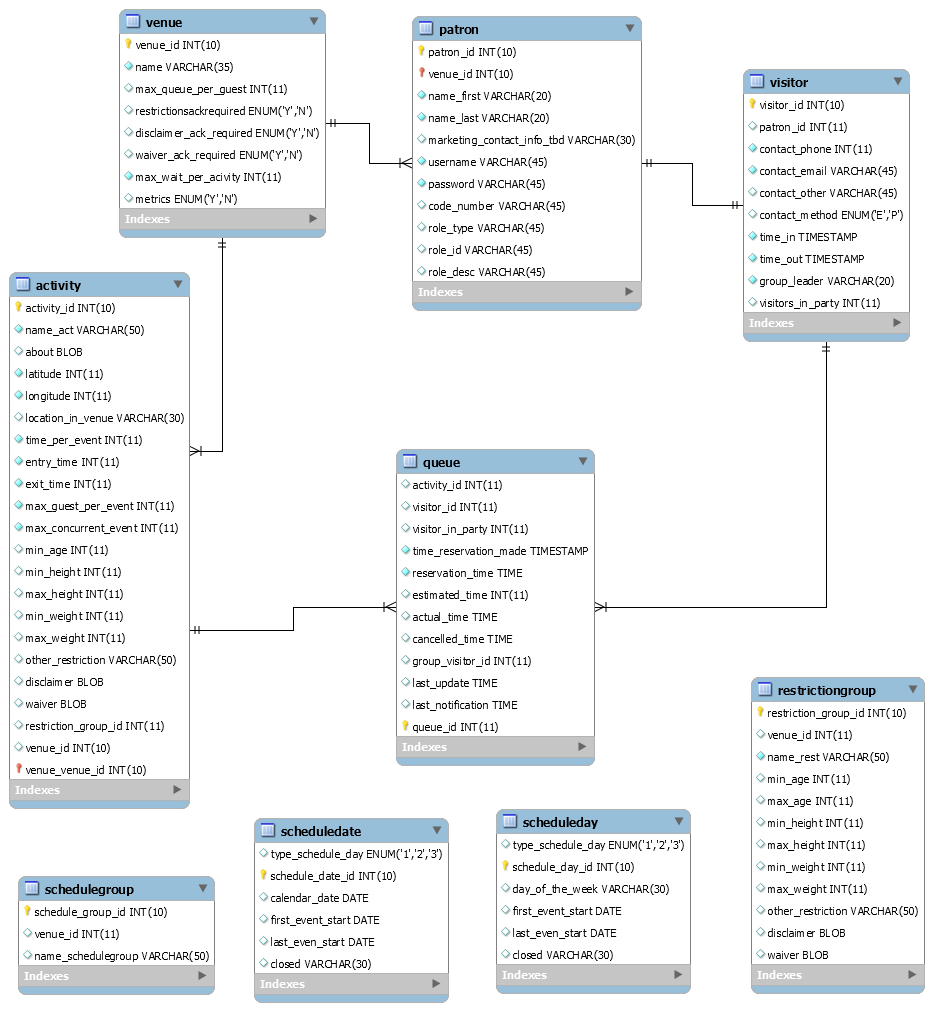
* 1. Hardware and Software Mapping

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 Apache Tomcat v8.0 application server and Spring 4.0 framework and MySQL database on the FIU SCIS network. Below is the representation of it.



* 1. 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. In each table the fields with yellow exponent mark are the Unique Key.

****

* 1. 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.

1. 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.

* 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.

* 1. Static model – detailed description of the structure for each subsystem. May include detailed class diagrams. Appendix C.

To represent the Virtual Queue system static model, each subsystem explained in the section 5.2 (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. The figures are located in Appendix C.

* 1. Dynamic model

Sequence diagrams are located in Appendix D

* 1. Code Specification - describe the class interfaces (attributes and method signatures) and constraint (invariants, pre-condition and post-conditions). Code should be in Appendix E.

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. Class Interfaces is located in Appendix E.

1. System Validation

This chapter covers the system validation of the VQ, which will have in detail the testing process of the system. The chapter is composed of two section 7.1 that has Subsystem Tests, covering testing done for each subsystem individual prior to integration and section 7.2 System Tests, covering all the system tests conducted.

* 1. Subsystem Tests – test each of the subsystems. This will involve the creation of a test drivers and stubs. Include the code for the test drivers and stubs in Appendix G.

|  |  |
| --- | --- |
| **Login/Logout Subsystem** | **Test Description** |
| Login User | Tests user is valid and store user information on session. |
| Logout User | Tests that user is in session and remove it from it. |

|  |  |
| --- | --- |
| **User Operation Subsystem** | **Test Description** |
| Register User | Tests if the system successfully adds a user to the system |
| Retrieve user info by ID | Tests that the id passed exist on the database and retrieve all user data for that specific id. |
| Update user | Tests if user is successfully updated on the database |
| Get All Users | Tests if method returns all users from database |
| Reset Password | Tests if password is successfully changed for a specific user. |

|  |  |
| --- | --- |
| **Ride Operation Subsystem** | **Test Description** |
| Get All Rides | Tests if method returns all rides from the park |
| Get Ride by User id | Tests if method returns all rides related to that user id. |
| Add User to Ride | Tests if user is successfully added to a specific ride. |
| Remove Ride from user | Tests that a user is remove from a ride. |

* 1. System Tests - For each use case create at least 3 test cases, 2 sunny day and one rainy day, should include security test cases. Each test case should include: test case id, purpose, test setup environment, test inputs, and expected outputs.

|  |  |
| --- | --- |
| Test Case ID | VQ01\_Login\_SunnyTest01 |
| Purpose | Test for successful connection to the VQ system with valid username and password. |
| Test Setup | MySQl database must have been setup correctly and program must be running. |
| Input | username: “kcid001@fiu.edu” password: “kely” |
| Expected Results | The user should be granted access to the system and be presented with account page. |

|  |  |
| --- | --- |
| Test Case ID | VQ01\_Login\_SunnyTest02 |
| Purpose | Test for successful connection to the VQ system with valid username and password. |
| Test Setup | MySQl database must have been setup correctly and program must be running. |
| Input | username: “felineyes151@gmail.com” password: “kely1” |
| Expected Results | The user should be granted access to the system and be presented with account page. |

|  |  |
| --- | --- |
| Test Case ID | VQ01\_Login\_RainyTest03 |
| Purpose | Test for connection to the VQ system with valid username and incorrect password. |
| Test Setup | MySQl database must have been setup correctly and program must be running. |
| Input | username: “kcid001@fiu.edu” password: “kely1” |
| Expected Results | The system does not allow access to user. |

|  |  |
| --- | --- |
| Test Case ID | VQ02\_Logout\_SunnyTest04 |
| Purpose | Test for successful logout from the VQ system by logged user. |
| Test Setup | MySQl database must have been setup correctly, program must be running, and user must be logged in. |
| Input | Click the logout button |
| Expected Results | The user is logged out of the VQ system and presented with the Home page. |

|  |  |
| --- | --- |
| Test Case ID | VQ02\_Logout\_SunnyTest05 |
| Purpose | Test for successful logout from the VQ system by logged user. |
| Test Setup | MySQl database must have been setup correctly, program must be running, and user must be logged in. |
| Input | Click the “X” in the top right corner |
| Expected Results | The user is logged out of the VQ system. |

|  |  |
| --- | --- |
| Test Case ID | VQ02\_Logout\_RainyTest06 |
| Purpose | Test for successful logout from the VQ system by logged user. |
| Test Setup | MySQl database must have been setup correctly, program must be running, and user must be logged in. System was unexpectedly disconnected from the database |
| Input | User tried to logout with a disconnected database. |
| Expected Results | Unable to logout dialog. |

|  |  |
| --- | --- |
| Test Case ID | VQ03\_ResetPass\_SunnyTest07 |
| Purpose | Test for successfully resetting password in the VQ system by registered user. |
| Test Setup | MySQl database must have been setup correctly, program must be running, user must be registered and must be on Login menu. |
| Input | User clicks the Reset Password button, fills out the form presented, clicks submit, and try to log back in to the system. |
| Expected Results | User is granted access to the system with the new password. |

|  |  |
| --- | --- |
| Test Case ID | VQ03\_ResetPass\_SunnyTest08 |
| Purpose | Test for successfully resetting password in the VQ system by registered user. |
| Test Setup | MySQl database must have been setup correctly, program must be running, user must be registered and must be on MyAccount menu. |
| Input | User clicks the Reset Password button, fills out the form presented, clicks submit, and try to log back in to the system. |
| Expected Results | User is granted access to the system with the new password. |

|  |  |
| --- | --- |
| Test Case ID | VQ03\_ResetPass\_RainyTest09 |
| Purpose | Test for resetting password in the VQ system by registered user with incorrect security answer. |
| Test Setup | MySQl database must have been setup correctly, program must be running, user must be registered and must be on Reset password form. |
| Input | For Username: [test@test.com](mailto:test@test.com), Security answer: two  User fills out the rest of the reset password form presented correctly, and clicks submit. |
| Expected Results | User is requested to enter the correct security answer on the form. |

|  |  |
| --- | --- |
| Test Case ID | VQ04\_ConfirmIdent\_SunnyTest10 |
| Purpose | Test if correct user by resetting password with valid username and security question. |
| Test Setup | MySQl database must have been setup correctly, program must be running, user must be registered and forgot password. |
| Input | User clicks on Reset Password from Login Page. User enters Username: [test@test.com](mailto:test@test.com), Security answer: one, and fills out the rest of the reset password form presented correctly, and clicks submit. |
| Expected Results | User is validated by the system, and password is correctly changed on the VQ database system. |

|  |  |
| --- | --- |
| Test Case ID | VQ04\_ConfirmIdent\_SunnyTest11 |
| Purpose | Test if correct user by resetting password with valid username and security question. |
| Test Setup | MySQl database must have been setup correctly, program must be running, user must be registered and must want to reset password. |
| Input | User clicks on Reset Password from Account Page. User enters Username: [test@test.com](mailto:test@test.com), Security answer: one, and fills out the rest of the reset password form presented correctly, and clicks submit. |
| Expected Results | User is validated by the system, and password is correctly changed on the VQ database system. |

|  |  |
| --- | --- |
| Test Case ID | VQ04\_ConfirmIdent\_SunnyTest12 |
| Purpose | Test if correct user by resetting password with wrong username and security question. |
| Test Setup | MySQl database must have been setup correctly, program must be running, user must be registered and must want to reset password. |
| Input | User clicks on Reset Pasword from Login Page. User enters Username: [test11@test.com](mailto:test11@test.com), Security answer: wrong, and fills out the rest of the reset password form presented correctly, and clicks submit. |
| Expected Results | User is validated and asked to correct the reset password form. Password is not changed on the VQ database system. |

|  |  |
| --- | --- |
| Test Case ID | VQ05\_RegisterNewUser\_SunnyTest13 |
| Purpose | Test to see if VQ system correctly adds a user to the VQ database. |
| Test Setup | MySQl database must have been setup correctly, program must be running, user requires an account. |
| Input | User clicks on Register button from the Home page, user fills the register form presented correctly, and clicks submit. |
| Expected Results | User is able to log in to the VQ system with the credential chosen. |

|  |  |
| --- | --- |
| Test Case ID | VQ05\_RegisterNewUser\_SunnyTest14 |
| Purpose | Test to see if VQ system correctly adds a user to the VQ database. |
| Test Setup | MySQl database must have been setup correctly, program must be running, user tries to login to the system without an account. |
| Input | User clicks on Register button from the login page, user fills the register form presented correctly, and clicks submit. |
| Expected Results | User is able to log in to the VQ system with the credential chosen. |

|  |  |
| --- | --- |
| Test Case ID | VQ05\_RegisterNewUser\_RainyTest15 |
| Purpose | Test to see if VQ system adds a user to the VQ database that already exists. |
| Test Setup | MySQl database must have been setup correctly, program must be running, user chooses to enter an already registered account. |
| Input | User clicks on Register button from Home page, fills the register form presented with username: [test@test.com](mailto:test@test.com) already on database, and clicks submit. |
| Expected Results | User is requested to choose another username because that one is already been used. |

|  |  |
| --- | --- |
| Test Case ID | VQ06\_DisplayAvaiRides&WTime\_SunnyTest16 |
| Purpose | Test if user can display the list of rides that are available. |
| Test Setup | MySQl database must have been setup correctly, program must be running, user is logged in. Rides are made available. |
| Input | User clicks on Rides button from the Home page |
| Expected Results | VQ system should allow the user to see all the available rides at the park with their respective and updated waiting time. |

|  |  |
| --- | --- |
| Test Case ID | VQ07\_AddRide\_SunnyTest17 |
| Purpose | Test if user can add the chosen ride. |
| Test Setup | MySQl database must have been setup correctly, program must be running, user is logged in. User clicks on Rides, and table is displayed. |
| Input | User clicks on “+” symbol to add Harry Potter ride. |
| Expected Results | User should be notified that the ride was successfully added, and ride should be displayed into his/her account with their respective and updated waiting time. |

|  |  |
| --- | --- |
| Test Case ID | VQ07\_AddRide\_SunnyTest18 |
| Purpose | Test if user can add the chosen ride. |
| Test Setup | MySQl database must have been setup correctly, program must be running, user is logged in. User clicks on Rides, and table is displayed. There will not be time conflict between rides. |
| Input | User clicks on “+” symbol to add Popeye ride. |
| Expected Results | User should be notified that the ride was successfully added, and ride should be displayed into his/her account with their respective and updated waiting time. |

|  |  |
| --- | --- |
| Test Case ID | VQ07\_AddRide\_RainyTest19 |
| Purpose | Test if user can add the chosen ride. |
| Test Setup | MySQl database must have been setup correctly, program must be running, user is logged in. User clicks on Rides, and table is displayed. There will be time conflict between rides. Total waiting time from other added rides is greater than all waiting times for rides available at the moment. |
| Input | User clicks on “+” symbol to add Jurasic Park ride. |
| Expected Results | User should be notified that the ride could not be added. |

|  |  |
| --- | --- |
| Test Case ID | VQ08\_EditUProf\_SunnyTest20 |
| Purpose | Test if admin can add edit a user. |
| Test Setup | MySQl database must have been setup correctly, program must be running, admin is logged in. Admin Search form is already displayed. |
| Input | Admin searches for “user@test.com and clicks on edit symbol for that user. Edit Form for with already loaded info for that user is presented and admin changes age from 28 to 29, and clicks Submit |
| Expected Results | Admin is acknowledged that the form was successfully submitted and the correct changes were made on the VenueRegisteredUser database table. |

|  |  |
| --- | --- |
| Test Case ID | VQ08\_EditUProf\_SunnyTest21 |
| Purpose | Test if admin can add edit a user. |
| Test Setup | MySQl database must have been setup correctly, program must be running, admin is logged in. Admin Search form is already displayed. |
| Input | Admin searches for “enrique@gmail.com” and clicks on edit symbol for that user. Edit Form for with already loaded info for that user is presented and admin changes email from [enrique@gmail.com](mailto:enrique@gmail.com)” to enrique@yahoo.com” , and clicks Submit |
| Expected Results | Admin is acknowledged that the form was successfully submitted and the correct changes were made on the VenueRegisteredUser database table. |

|  |  |
| --- | --- |
| Test Case ID | VQ08\_EditUProf\_RainyTest22 |
| Purpose | Test if admin can add edit a user. |
| Test Setup | MySQl database must have been setup correctly, program must be running, admin is logged in. Admin Search form is already displayed. |
| Input | Admin doesn not have any user requests and clicks on cancel option on the bottom of the page. |
| Expected Results | Search form is successfully closed. |

|  |  |
| --- | --- |
| Test Case ID | VQ09\_DisableUser\_SunnyTest23 |
| Purpose | Test if admin can disable a user. |
| Test Setup | MySQl database must have been setup correctly, program must be running, admin is logged in. Admin Search form is already displayed. |
| Input | Admin searches for “enrique@gmail.com” and clicks on disable option for that user. Admin is asked if he wants to proceed with the requests, and clicks OK |
| Expected Results | User is successfully disabled. |

|  |  |
| --- | --- |
| Test Case ID | VQ09\_DisableUser\_SunnyTest24 |
| Purpose | Test if admin can disable a user. |
| Test Setup | MySQl database must have been setup correctly, program must be running, admin is logged in. Admin Search form is already displayed. |
| Input | Admin searches for “user@test.com” and clicks on disable option for that user. Admin is asked if he wants to proceed with the requests, and clicks OK |
| Expected Results | User is successfully disabled. |

|  |  |
| --- | --- |
| Test Case ID | VQ09\_DisableUser\_RainyTest25 |
| Purpose | Test if admin can disable a user. |
| Test Setup | MySQl database must have been setup correctly, program must be running, admin is logged in. Admin Search form is already displayed. |
| Input | Admin searches for “user@test.com” and clicks on disable option for that user. Admin is asked if he wants to proceed with the requests, and clicks Cancel |
| Expected Results | User was not disabled. |

|  |  |
| --- | --- |
| Test Case ID | VQ10\_EnableUser\_SunnyTest26 |
| Purpose | Test if admin can enable a user. |
| Test Setup | MySQl database must have been setup correctly, program must be running, admin is logged in. Admin Search form is already displayed. |
| Input | Admin searches for “enrique@gmail.com” and clicks on enable option for that user. Admin is asked if he wants to proceed with the requests, and clicks OK |
| Expected Results | User is successfully enabled. |

|  |  |
| --- | --- |
| Test Case ID | VQ10\_EnableUser\_SunnyTest27 |
| Purpose | Test if admin can enable a user. |
| Test Setup | MySQl database must have been setup correctly, program must be running, admin is logged in. Admin Search form is already displayed. |
| Input | Admin searches for “user@test.com” and clicks on enable option for that user. Admin is asked if he wants to proceed with the requests, and clicks OK |
| Expected Results | User is successfully enabled. |

|  |  |
| --- | --- |
| Test Case ID | VQ10\_EnableUser\_RainyTest28 |
| Purpose | Test if admin can enable a user. |
| Test Setup | MySQl database must have been setup correctly, program must be running, admin is logged in. Admin Search form is already displayed. |
| Input | Admin searches for “user@test.com” and clicks on disable option for that user. Admin is asked if he wants to proceed with the requests, and clicks Cancel |
| Expected Results | User was not enabled. |

|  |  |
| --- | --- |
| Test Case ID | VQ11\_DeletRide\_SunnyTest29 |
| Purpose | Test if user can delete the chosen ride. |
| Test Setup | MySQl database must have been setup correctly, program must be running, user is logged in. User clicks on MyAccount from home page, and all his rides are displayed. |
| Input | User clicks on “X” option to delete Harry Potter ride. |
| Expected Results | User should be asked if he wants to proceed with the request, user clicks OK, and ride successfully delete from his account page, and time is updated on the Venue Available Ride table. |

|  |  |
| --- | --- |
| Test Case ID | VQ11\_DeletRide\_SunnyTest30 |
| Purpose | Test if user can delete the chosen ride. |
| Test Setup | MySQl database must have been setup correctly, program must be running, user is logged in. User clicks on MyAccount from home page, and all his rides are displayed. |
| Input | User clicks on “X” option to delete Popeye ride. |
| Expected Results | User should be asked if he wants to proceed with the request, user clicks OK, and ride successfully delete from his account page, and time is updated on the Venue Available Ride table. |

|  |  |
| --- | --- |
| Test Case ID | VQ11\_DeletRide\_RainyTest31 |
| Purpose | Test if user can delete the chosen ride. |
| Test Setup | MySQl database must have been setup correctly, program must be running, user is logged in. User clicks on MyAccount from home page, and all his rides are displayed. |
| Input | User clicks on “X” option to delete Jurassic Park ride. |
| Expected Results | User should be asked if he wants to proceed with the request, user clicks Cancel, and user is not dequeue from that ride, and still showing on his account. |

|  |  |
| --- | --- |
| Test Case ID | VQ12\_ViewUserAct\_SunnyTest32 |
| Purpose | Test if users can see their account activities. |
| Test Setup | MySQl database must have been setup correctly, program must be running, user is logged in. User has some rides he already queued for. |
| Input | User clicks on MyAccount page button from the Home page |
| Expected Results | User can see his account displaying all the rides he is queue for at that moment. |

|  |  |
| --- | --- |
| Test Case ID | VQ12\_ViewUserAct\_SunnyTest33 |
| Purpose | Test if users can see their account activity. |
| Test Setup | MySQl database must have been setup correctly, program must be running, user is logged in and on his/her account page. User has canceled all rides. |
| Input | No input necessary. |
| Expected Results | User can see his/her account page with no rides. |

|  |  |
| --- | --- |
| Test Case ID | VQ12\_ViewUserAct\_RainyTest34 |
| Purpose | Test if users can see their account activity. |
| Test Setup | MySQl database must have been setup correctly, program must be running, user is logged in and on his/her account page. |
| Input | User clicks on MyAccount page button from the Home page, and right after he clicks cancel on the bottom. |
| Expected Results | User account page is close and no activity will be shown. |

|  |  |
| --- | --- |
| Test Case ID | VQ13\_RecEmailNotif\_SunnyTest35 |
| Purpose | Test if user can receive a notification for a specific ride he added. |
| Test Setup | MySQl database must have been setup correctly, program must be running, user is logged in. User is queue for a Harry Potter ride that is due in 15 min. |
| Input | No Input neccesary. |
| Expected Results | User should get an email notification with the name of the Ride and the time he has to get to the place. |

|  |  |
| --- | --- |
| Test Case ID | VQ13\_RecEmailNotif\_SunnyTest36 |
| Purpose | Test if user can receive a notification for a specific ride he added. |
| Test Setup | MySQl database must have been setup correctly, program must be running, user is logged in. User is queue for a Popeye ride that is due in 10 min. |
| Input | No Input neccesary. |
| Expected Results | User should get an email notification with the name of the Ride and the time he has to get to the place. |

|  |  |
| --- | --- |
| Test Case ID | VQ13\_RecEmailNotif\_SunnyTest42 |
| Purpose | Test if user can receive a notification for a specific ride he added. |
| Test Setup | MySQl database must have been setup correctly, program must be running, user is logged in. User is queue for a Jurassic Park ride that is due in 20 min. |
| Input | User deletes the Jurassic Park ride before 25 minutes before is due. |
| Expected Results | User does not receive email notification for that ride because he is not on db table anymore. |

* 1. Evaluation of Tests – evaluate how successful the tests were. Use a tabular form.

On the table below are each of the system test cases id for the VQ system that were conducted and the actual results.

|  |  |
| --- | --- |
| Test Case ID | Actual Test Results |
| VQ01\_Login\_SunnyTest01 | PASS |
| VQ01\_Login\_SunnyTest02 | PASS |
| VQ01\_Login\_RainyTest03 | PASS |
| VQ02\_Logout\_SunnyTest04 | PASS |
| VQ02\_Logout\_SunnyTest05 | PASS |
| VQ02\_Logout\_RainyTest06 | PASS |
| VQ03\_ResetPass\_SunnyTest07 | PASS |
| VQ03\_ResetPass\_SunnyTest08 | PASS |
| VQ03\_ResetPass\_RainyTest09 | PASS |
| VQ04\_ConfirmIdent\_SunnyTest10 | PASS |
| VQ04\_ConfirmIdent\_SunnyTest11 | PASS |
| VQ04\_ConfirmIdent\_RainyTest12 | PASS |
| VQ05\_RegisterNewUser\_SunnyTest13 | PASS |
| VQ05\_RegisterNewUser\_SunnyTest14 | PASS |
| VQ05\_RegisterNewUser\_RainyTest15 | PASS |
| VQ06\_DisplayAvaiRides&WTime\_SunnyTest16 | PASS |
| VQ07\_AddRide\_SunnyTest17 | PASS |
| VQ07\_AddRide\_SunnyTest18 | PASS |
| VQ07\_AddRide\_RainyTest19 | PASS |
| VQ08\_EditUProf\_SunnyTest20 | PASS |
| VQ08\_EditUProf\_SunnyTest21 | PASS |
| VQ08\_EditUProf\_RainyTest22 | PASS |
| VQ09\_DisableUser\_SunnyTest23 | PASS |
| VQ09\_DisableUser\_SunnyTest24 | PASS |
| VQ09\_DisableUser\_RainyTest25 | PASS |
| VQ10\_EnableUser\_SunnyTest26 | PASS |
| VQ10\_EnableUser\_SunnyTest27 | PASS |
| VQ10\_EnableUser\_RainyTest28 | PASS |
| VQ11\_DeletRide\_SunnyTest29 | PASS |
| VQ11\_DeletRide\_SunnyTest30 | PASS |
| VQ11\_DeletRide\_RainyTest31 | PASS |
| VQ12\_ViewUserAct\_SunnyTest32 | PASS |
| VQ12\_ViewUserAct\_SunnyTest33 | PASS |
| VQ12\_ViewUserAct\_RainyTest34 | PASS |
| VQ13\_RecEmailNotif\_SunnyTest35 | PASS |
| VQ13\_RecEmailNotif\_SunnyTest36 | PASS |
| VQ13\_RecEmailNotif\_RainyTest36 | PASS |

1. Glossary - define terms used in document, especially domain specific terms.

**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.

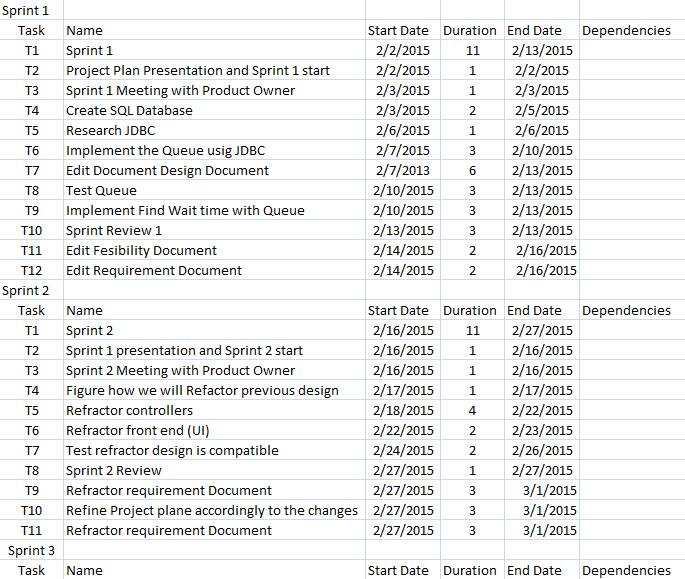
**Class Diagram:** A pictorial representation of all the classes in the system

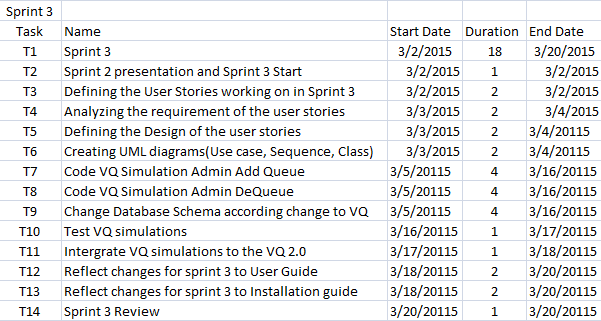
**Object Diagram:** A pictorial representation of an instance of a class with example of how the data of the class will be populated.

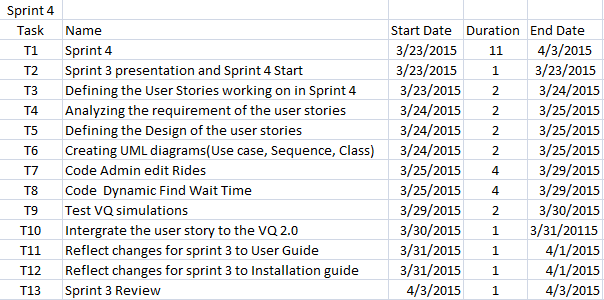
**Sequence Diagram:** A pictorial representation of how processes operate with one another and the user during the course of a specific piece of functionality.

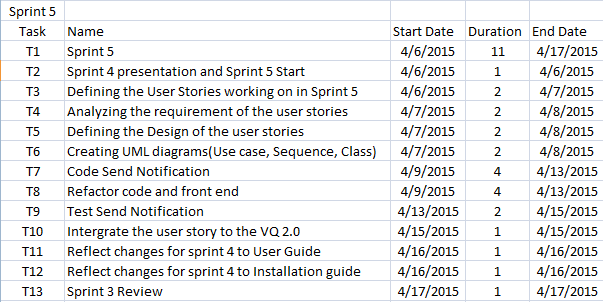
**Task:** A piece of job that serves as a unit of work.

1. Appendix
   1. Appendix A - Project schedule



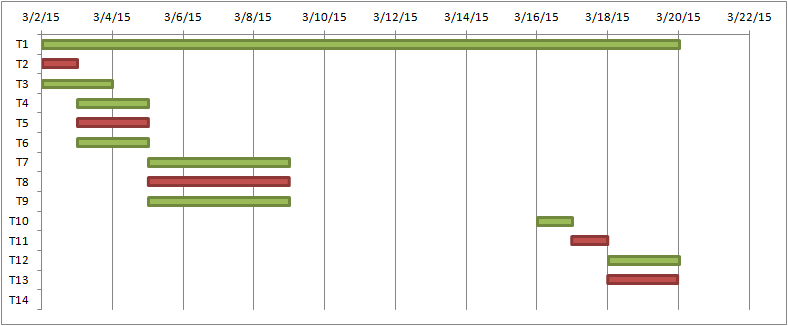
****





**Sprint 1 Gantt Chart**

**Sprint 2 Gantt Chart**

****

**Sprint 3 Gantt Chart**

**Sprint 4 Gantt Chart**

**Sprint 5 Gantt Chart**

* 1. Appendix B – All use cases with nonfunctional requirements.

*Use Case ID:* **VQ01 – User Login**

*Actors:* VenueRegistered Visitor, Single Venue Admin.

*Pre-conditions:*

1. Web page has been activated
2. Participants must have a web username and password.

*Description:*

1. Use case begins when the user accesses the login option.
2. The user will be prompted with a data entry template for username, password, and code if he/she is a Venue Registered Visitor.
3. The user will be prompted with a data entry template for username, and password if he/she is a Single Venue Admin.
4. The user provides a previously registered user name, password, and code if he/she is a Venue Registered Visitor.
5. The user provides a previously registered user name, and password if he/she is a Single Venue Admin.
6. User shall continue request by accessing the Login.
7. If the credentials are valid, the system will log the user into the corresponding page.
8. Use case ends when access is granted to respective user.

*Relevant requirements:* A user will only be allowed into the system if he/she has a valid username and password.

*Post-conditions:*

1. Access is granted.

*Alternative Courses of Action for Venue Registered Visitor:*

1. In step 2 of Description section there is an option that allows the Venue Registered Visitor to reset the password if he/she forgot. (See use Case **VQ03 – Reset Password**)
2. In step 6 of Description section users have the option to click cancel and close the data entry form.

*Exceptions:*

1. The login option on the Website is not active.
2. The cancel option on the Website is not active.
3. The option to reset password is not active.
4. The database is not active.

*Related Use Cases: Related Use Cases:* **VQ04 – Confirm Identity.**

**Special Requirements:**

* **Usability**: No previous training time. System is simple and easy to follow.
* **Reliability**: The system should perform correctly 99% of the time.
* **Performance**: The system should be sent and saved within 3 seconds.
* **Supportability**: The system should be easy to maintain, make appropriate changes, and be correctly handled by IE, Mozilla, Chrome and Safari.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

*Use Case ID:* **VQ02 – User Logout**

*Actors:* VenueRegistered Visitor, Single Venue Admin.

*Pre-conditions:*

1. Web page has been activated.
2. User must have been previously logged in.

*Description:*

1. Use case begins when the user accesses the logout option.
2. The system logs the user out.
3. Use case ends when the system logs user out and displays the home screen.

*Relevant requirements:* A user will only be logged out if he/she has previously logged in.

*Post-conditions:*

1. User gets successfully logged out of the system.

*Alternative Courses of Action:*

1. The user closes the webpage.

*Exceptions:*

1. The logout option on the webpage is not active.

*Related Use Cases:* None.

**Special Requirements:**

* **Usability**: No previous training time. System is simple and easy to follow.
* **Reliability**: The system should perform correctly 99% of the time.
* **Performance**: The system should be sent and saved within 3 seconds.
* **Supportability**: The system should be easy to maintain, make appropriate changes, and be correctly handled by IE, Mozilla, Chrome and Safari.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

*Use Case ID:* **VQ03 – Reset Password**

*Actors:* VenueRegistered Visitor.

*Pre-conditions:*

1. User must have previously registered into the system.
2. User must be logged out of the system.

*Description:*

1. Use case begins when the user accesses the option to reset password on the login screen.
2. The system will prompt the user with a data entry template.
3. The user should enter username, which will be their registered email, choose their security question, and write their security answer, new password and password confirmation.
4. The user shall complete the form and submit the changes.
5. If the username is on record, and security question, and security answer are correct, the system will update the password and notify the user if the request was submitted successfully. (See Use case **VQ04 – Confirm Identity**).
6. Use case ends when the request is stored in the system.

*Relevant requirements:* A user will only be allowed to reset password if he/she has previously registered.

*Post-conditions:*

1. The password gets successfully changed in the database.

*Alternative Courses of Action:*

1. In step 3 of Description section the username could not be found.
2. In step 3 of Description section the security question chosen does not match the records.
3. In step 3 of Description section the answer to the security answer does not match the records.
4. In step 4 of Description section the user has the option to cancel the request.

*Extensions:*

1**. VQ04 – Confirm Identity**

*Exceptions****:***

1. The option to reset the password is inactive.
2. The user table in the database is inactive.

*Related Use Cases:* **VQ04 – Confirm Identity.**

**Special Requirements:**

* **Usability**: No previous training time. System is simple and easy to follow.
* **Reliability**: The system should perform correctly 99% of the time.
* **Performance**: The system should be sent and saved within 3 seconds.
* **Supportability**: The system should be easy to maintain, make appropriate changes, and be correctly handled by IE, Mozilla, Chrome and Safari.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

*Use Case ID:* **VQ04 – Confirm Identity**

*Actors:* VenueRegistered Visitor, Single Venue Admin.

*Pre-conditions:*

1. Web Page has been activated.
2. User already has an active access account.
3. User accesses the login option.
4. User accesses the forget password option.

*Description:*

1. Use case begins when the user is prompted with a template for data entry either for login or reset password.
2. The user shall enter his username which is the email, password correctly, and code if she/he is a Venue Registered User trying to login.
3. The user shall enter his username which is the email, password correctly if she/he is a Single Venue Admin trying to login.
4. The user shall enter his username, which is the email, and finish the rest of the reset password form correctly if she/he is a Venue Registered User trying to login.
5. Use case ends when user clicks on submit button and is presented with the login form.

*Relevant requirements:* none.

*Post-conditions:* The Venue Registered User is taken back to Login if she/he is trying to reset the password, or presented the login form again if it is a Venue Registered User or a Single Venue Admin trying to login.

*Alternative Courses of Action:*

1. In step 1 of the Description section the user can cancel the request.
2. In step 2 of the Description section the username, password, or code are not found.
3. In step 3 of the Description section the username or password are not found.

*Exceptions****:***

1. The system is unavailable.

*Related Use Cases:* VQ01 Login.

*Related Use Cases:* VQ03 Reset Password.

**Special Requirements:**

* **Usability**: No previous training time. System is simple and easy to follow.
* **Reliability**: The system should perform correctly 99% of the time.
* **Performance**: The system should be sent and saved within 3 seconds.
* **Supportability**: The system should be easy to maintain, make appropriate changes, and be correctly handled by IE, Mozilla, Chrome and Safari.

*Use Case ID:* **VQ05 - Register New User**

*Actors:* Venue Unregistered Visitor

*Pre-conditions:*

1. Web page has been activated

*Description:*

1. Use case begins when guest user accesses the register option on the Website.

2. The system shall provide the guest user or new administrator with a template for data entry.

3. The guest user should enter the following data: first name, last name, password, security question, security answer, phone number, age, and code (for confirmation).

4. The guest user should send the request by accessing the submit option.

5. The system shall then notify the guest user if the request was submitted successfully.

6. When the request is received the system shall update the record for that new user.

7. Use case ends when the record is updated.

*Relevant requirements:*

1. A guest user will only be able to register if they do not have a profile created from before.
2. Venue Registered users will be rejected.

*Post-conditions:*

1. The record for that user has been created.

*Alternative Courses of Action:*

1. In step 4 o the f Description section the user has the option to cancel the request.
2. In step 5 o the f Description section, if any of the required fields are left blank the system shall notify the user of the missing fields, and how to fill the fields properly.

*Exceptions:*

1. The register option on the webpage is not active.
2. After the user enters the required information into the system the user gets rejected.

*Use Cases:* None.

**Special Requirements:**

* **Usability**: No previous training time. System is simple and easy to follow.
* **Reliability**: The system should perform correctly 99% of the time.
* **Performance**: The system should be sent and saved within 3 seconds.
* **Supportability**: The system should be easy to maintain, make appropriate changes, and be correctly handled by IE, Mozilla, Chrome and Safari.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

*Use Case ID:* **VQ06 – Display Available Rides and their Waiting Time.**

*Actors:* VenueRegistered Visitor.

*Pre-condition:*

1.      Web page has been activated.

*Description:*

1.      Use case begins when user accesses the option Rides from the home page or from his account page.

2.      The system shall get the list of the rides available at the venue along with each ride actual waiting time.

3.      Use case ends when the system displays the list of the rides available for the venue user along with each ride actual waiting time.

*Alternative Courses of Action:*

1. In Step 1 of Description section the Venue Registered User request could not be completed.
2. In step 3 of the Description section the user has the option to cancel the form.
3. In Step 3 of Description section the list of venue available rides and their respective waiting time could not be display.
4. In Step 3 of Description section the Venue Registered User can use the search option to search for a specific ride name or other criteria.
5. In Step 3 of Description section the Venue Registered User can switch the views of the table presented.
6. In Step 3 of Description section the Venue Registered User can update the complete table by clicking the update symbol.
7. In Step 3 of Description section the Venue Registered User can choose which columns they want to see from the table presented.

*Exceptions:*

1. The option to select ride/event is not active.
2. The system could not continue after submitting the selection.

*Related Uses case:* **VQ07 – Add (Queue) for a Ride**

**Special Requirements:**

* **Usability**: No previous training time. System is simple and easy to follow.
* **Reliability**: The system should perform correctly 99% of the time.
* **Performance**: The system should be sent and saved within 3 seconds.
* **Supportability**: The system should be easy to maintain, make appropriate changes, and be correctly handled by IE, Mozilla, Chrome and Safari.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

*Use Case ID:* **VQ07 – Add (Queue) User for a Ride**

*Actors:* VenueRegistered Visitor.

*Pre-condition:*

1. Web page has been activated.
2. Venue Registered User has an active account within the system.
3. Venue Registered User is already on his account page.

*Description:*

1. Use case begins when user clicks on Rides from his account page, and a list with available rides is presented.
2. User shall continue when she/he has already decided which ride she/he wants to add (queue).
3. User shall continue by clicking the “+” symbol to add their preferred ride.
4. The system shall notify the user if ride was successfully added.
5. Use case ends when customer accepts the notification displayed and she/he is taken to his/her account page where the ride added is displayed along with his/her waiting time.

*Post-conditions:*

1. The system shall update the changes made on the database.
2. The number of rides for the customer is increased by one in the database.
3. User is added to UserQueue table.
4. The waiting time for that specific ride is increased.

*Alternative Courses of Action:*

1. In step 3 of the Description section the user has the option to cancel the ride from his page.
2. In Step 4 of Description section the system notifies that the request could not be completed.

*Exceptions:*

1.      The option to add ride is not active.

2.      The option to accept the notification is not active.

3.      The system could not submit the request.

*Related Uses case:* **VQ06 – Display Available Rides and Their Waiting Time.**

**Special Requirements:**

* **Usability**: No previous training time. System is simple and easy to follow.
* **Reliability**: The system should perform correctly 99% of the time.
* **Performance**: The system should be sent and saved within 3 seconds.
* **Supportability**: The system should be easy to maintain, make appropriate changes, and be correctly handled by IE, Mozilla, Chrome and Safari.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

*Use Case ID:* **VQ08 - Edit User's Profile**

*Actors:* Single Venue Admin.

*Pre-conditions:*

1. User (administrator) has an existing profile.
2. Web page has been activated.
3. User (administrator) has successfully logged onto the system.

*Description:*

1. Use case begins when *Admin* selects the Search under Admin tab from the home page.
2. *Admin* search the user to be edited.
3. Admin finds the user and select Edit option for that specific user.
4. Admin is prompted with a data entry template with already loaded user info.
5. Admin can change any field they want (making sure all fields have an entry).
6. Admin shall accept “Submit”.
7. The system shall notify the admin if the request was submitted successfully.
8. Admin is taken back to the search page.
9. Use case ends when the system saves the updated information.

*Relevant requirements:* None.

*Post-conditions:*

1. User's profile is updated in the database.

*Alternative Courses of Action:*

1. In Step 4 of Description section the user has the option to cancel the request.
2. In Step 5 of Description section the system prompt the user to enter more data if a required field is left blank.
3. In Step 7 of Description section the system notifies that the request could not be completed.

*Exceptions:*

1. The edit option link is not active.
2. The cancel option link is not active.
3. The submit option is not active.

*Related Use Cases:* None

**Special Requirements:**

* **Usability**: No previous training time. System is simple and easy to follow.
* **Reliability**: The system should perform correctly 99% of the time.
* **Performance**: The system should be sent and saved within 3 seconds.
* **Supportability**: The system should be easy to maintain, make appropriate changes, and be correctly handled by IE, Mozilla, Chrome and Safari.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

*Use Case ID:* **VQ09 - Disable User's Account**

*Actors:* Single Venue Admin.

*Pre-conditions:*

1. User (administrator) has an existing profile.

2. Web page has been activated.

3. User has successfully logged onto the system.

*Description:*

1. Use case begins when *Admin* selects Search under Admin tab from the home page.
2. *Admin* search the user to be disabled.
3. Admin finds the user and selects disabled option for that specific user.
4. The system shall ask the *Admin* if they are sure they want to proceed.
5. The *Admin* shall confirm by accepting the “Ok” option.
6. The system shall notify the *Admin* if the request was submitted successfully.
7. *Admin* is taken back to the search page.
8. Use case ends when the system saves the updated information.

*Relevant requirements:* None.

*Post-conditions:*

1. Record for user is updated in the database.

*Alternative Courses of Action:*

1. In step 4 of Description section the *Admin* has the option to cancel the request.

*Exceptions:*

1. The system is unavailable.
2. The “Cancel” option is unavailable.
3. The confirmation dialog is not displayed.
4. The “Disable” option is unavailable.

*Related Use Cases:* None

**Special Requirements:**

* **Usability**: No previous training time. System is simple and easy to follow.
* **Reliability**: The system should perform correctly 99% of the time.
* **Performance**: The system should be sent and saved within 3 seconds.
* **Supportability**: The system should be easy to maintain, make appropriate changes, and be correctly handled by IE, Mozilla, Chrome and Safari.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

*Use Case ID:* **VQ10 - Enable User's Account**

*Actors:* Single Venue Admin.

*Pre-conditions:*

1. User (administrator) has an existing profile.

2. Web page has been activated.

3. User has successfully logged onto the system.

*Description:*

1. Use case begins when *Admin* selects Search under Admin tab from the home page.
2. *Admin* search the user to be enabled.
3. Admin finds the user and selects enabled option for that specific user.
4. The system shall ask the *Admin* if they are sure they want to proceed.
5. The *Admin* shall confirm by accepting the “Ok” option.
6. The system shall notify the *Admin* if the request was submitted successfully.
7. *Admin* is taken back to the search page.
8. Use case ends when the system saves the updated information.

*Relevant requirements:* None.

*Post-conditions:*

1. Record for user is updated in the database.

*Alternative Courses of Action:*

1. In step 4 of Description section the *Admin* has the option to cancel the request.

*Exceptions:*

1. The system is unavailable.
2. The “Cancel” option is unavailable.
3. The confirmation dialog is not displayed.
4. The “enable” option is unavailable.

*Related Use Cases:* None

**Special Requirements:**

* **Usability**: No previous training time. System is simple and easy to follow.
* **Reliability**: The system should perform correctly 99% of the time.
* **Performance**: The system should be sent and saved within 3 seconds.
* **Supportability**: The system should be easy to maintain, make appropriate changes, and be correctly handled by IE, Mozilla, Chrome and Safari.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

*Use Case ID:* **VQ11 – Delete (Dequeue) User from a Registered Ride**

*Actors:* VenueRegistered Visitor.

*Pre-condition:*

1. Web page has been activated.
2. Venue Registered User has successfully logged onto the system.
3. User is on his/her account page.
4. There is a user request to remove an already ride he/she queued for from his/her account.

*Description:*

1. Use case begins when user accesses MyAccount link from the home page, and the corresponding page is displayed.
2. User shall continue when she/he has already decided which ride she/he wants to delete (dequeue).
3. User shall continue by clicking the “X” symbol to dequeue from that specific ride.
4. The system shall ask the *user* if they are sure they want to proceed.
5. The *user* shall confirm by accepting the “OK” option.
6. Use case ends when customer accepts the notification displayed and she/he is taken to his/her account page where the ride delete is not displayed anymore on his account page.

*Post-conditions:*

1. The system shall update the changes made on the database.
2. The system shall dequeue user from ride when the time for that ride comes up, even if he/she have not select de “delete” option.
3. The number of rides for the customer is decreased by one in the database.
4. User is deleted to UserQueue table.
5. The waiting time for that specific ride decreases.

*Alternative Courses of Action:*

1. In step 3 of the Description section the user has the option to cancel the confirmation.
2. In Step 4 of Description section the system notifies that the request could not be completed.

*Exceptions:*

1.      The option to delete ride is not active.

2.      The option to accept the notification is not active.

3.      The system could not submit the request.

*Related Uses case:* **VQ07 – Add (Queue) for a Ride**

**Special Requirements:**

* **Usability**: No previous training time. System is simple and easy to follow.
* **Reliability**: The system should perform correctly 99% of the time.
* **Performance**: The system should be sent and saved within 3 seconds.
* **Supportability**: The system should be easy to maintain, make appropriate changes, and be correctly handled by IE, Mozilla, Chrome and Safari.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

*Use Case ID:* **VQ12 – View/Display User Activity**

*Actors:* VenueRegistered Visitor.

*Pre-condition:*

1.      Web page has been activated.

*Description:*

1.      Use case begins when user accesses the option MyAccount from the home page.

2.      The system shall get the list of the rides available for that specific user along with each ride actual waiting time.

3.      Use case ends when the system displays the list of the rides available for that user along with each ride actual waiting time.

*Alternative Courses of Action:*

1. In Step 1 of Description section the Venue Registered User request could not be completed.
2. In step 3 of the Description section the user has the option to close the modal.
3. In Step 3 of Description section the list of venue available rides and their respective waiting time could not be display.
4. In Step 3 of Description section the Venue Registered User can use the search option to search for a specific ride name or other criteria.
5. In Step 3 of Description section the Venue Registered User can switch the views of the table presented.
6. In Step 3 of Description section the Venue Registered User can update the complete table by clicking the update symbol.
7. In Step 3 of Description section the Venue Registered User can choose which columns they want to see from the table presented.

*Exceptions:*

1. The system could not display the form.

*Related Uses case:* None

**Special Requirements:**

* **Usability**: No previous training time. System is simple and easy to follow.
* **Reliability**: The system should perform correctly 99% of the time.
* **Performance**: The system should be sent and saved within 3 seconds.
* **Supportability**: The system should be easy to maintain, make appropriate changes, and be correctly handled by IE, Mozilla, Chrome and Safari.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

*Use Case ID:* **VQ13 – Send/Received Notification Time**

*Actors:* Venue Registered User.

*Pre-condition:*

1. Web page has been activated.
2. The user has successfully logged onto the system.
3. The user has successfully selected the ride preferred.

*Description:*

1. Use case begins when user has already added the ride requested to his/her account page.
2. User shall be kept in the ride queue until the time for his/her turn approaches.
3. System should create a job and send notification to each user that time for rides are coming up next on queue.
4. Use case ends when user receives the notification email with the name and time for the specific ride before the time approaches.

*Post-conditions:*

1. The system shall update the changes made on the database to update the jobs.

*Alternative Courses of Action:*

1. In step 2 of the Description section the user has the option to cancel their ride from their account page.
2. In step 3 of the Description section the could not create a job and send the notification.
3. In Step 4 of Description section the system notifies that the notification could not be sent.

*Exceptions*

1. The system could not create the job.
2. The system could not send the request.

*Related Uses case:* **VQ07 – Add (Queue) for a Ride**

**Special Requirements:**

* **Usability**: No previous training time. System is simple and easy to follow.
* **Reliability**: The system should perform correctly 99% of the time.
* **Performance**: The system should be sent and saved within 3 seconds.
* **Supportability**: The system should be easy to maintain, make appropriate changes, and be correctly handled by IE, Mozilla, Chrome and Safari.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

*Use Case ID:* **VQ14 – Start-Stop Jobs to Send Notifications and Dequeue users automatically from Rides**

*Actors:* Single Venue Admin.

*Pre-condition:*

1. Venue is going to be opened soon.

*Description:*

1. Use case begins when admin is decides is time to start the scheduler to send notifications and delete users from rides when is time to do so.
2. Admin shall type the appropriate URL path with the right command to start the scheduler jobs that will send the notification first and then dequeue user from ride on specific time.
3. Use case ends when system creates the appropriate jobs and execute in in the corresponding time.

*Post-conditions:*

1. The system shall update the users account once they are dequeue from ride.
2. The system shall update the changes made on the database to update the jobs.
3. The system should continue the jobs until the administrator decides to stop it.

*Alternative Courses of Action:*

1. In step 2 of the Description section the admin does not type the URL path correctly.
2. In step 3 of the Description section the system could not create a job and send the notification.
3. In step 3 of the Description section the system could not create a job to dequeue user from ride.
4. In Step 4 of Description section the system notifies that the notification could not be sent.

*Exceptions*

1. The system could not create the job.
2. The system could not send the request.

*Related Uses case:* **VQ13 – Send/Receive Email Notification**

**Special Requirements:**

* **Usability**: No previous training time. System is simple and easy to follow.
* **Reliability**: The system should perform correctly 99% of the time.
* **Performance**: The system should be sent and saved within 3 seconds.
* **Supportability**: The system should be easy to maintain, make appropriate changes, and be correctly handled by IE, Mozilla, Chrome and Safari.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

*Use Case ID:* **VQ15 – Find Wait Time**

*Descriptions:* Calculate how many riders per car to find appropriate wait times to populate lists

*User Story:* #109 Find Wait Time

*Actors:* Single Venue Admin and Venue Visitor/Non Visitor.

*Pre-condition:*

1. Admin or visitor logged in. Non Visitor does not need to be.

*Description:*

1. Use case begins when admin or visitor clicks on Ride Tab above the VQ home webpage.
2. Use case ends when the Ride's list populated with the ride names and their concurrent waiting times.

*Post-conditions:*

1. The system shall update the waiting time as users queue.
2. The system shall update the waiting time when users dequeue from the rides.

*Alternative Courses of Action:*

1. In Step 1 Users can click on My Account and click on Ride tab within to access the Ride Find Wait Time.

*Exceptions*

1. The system could not find wait time.
2. The system could not find activities.

*Related Uses case:* **N/A**

**Special Requirements:**

* **Usability**: No previous training time. System is simple and easy to follow.
* **Reliability**: The system should perform correctly 99% of the time.
* **Performance**: The system should be sent and saved within 2 seconds.
* **Supportability**: The system should be easy to maintain, make appropriate changes, and be correctly handled by IE, Mozilla, Chrome and Safari.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

*Use Case ID:* **VQ16 – AddQueue**

*Descriptions:* ‘Controller that allows finding wait time and queuing and dequeuing to the queue.

*User Story:* #108 Implementing the Queue

*Actors:* Single Venue Admin and Venue Visitor/Non Visitor.

*Pre-condition:*

1. Admin or visitor logged in. Non Visitor does not have access.

*Description:*

1. Use case begins when admin or visitor user clicks on Ride Tab on the VQ home webpage.
2. The User or Administrator will have a freshly updated list of Rides and their concurrent waiting times.
3. The admin or Visitor can click on the “Add Ride” button located to the right of each of the wait times.
4. Use case ends when the selected ride becomes added to the Queue and displayed within the user account under “My Account”.

*Post-conditions:*

1. The system shall update the waiting time as users queue.
2. The system shall update the admin or Visitor account with newly selected rides.

*Alternative Courses of Action:*

1. In Step 1 Users can click on My Account and click on Ride tab within to access the Ride List.

*Exceptions*

1. The system could not find wait time.
2. The system could not add into queue.
3. The system could not find activities.

*Related Uses case:* **VQ15 – Find Wait Time,****VQ17 – Visitor DeQueue**

**Special Requirements:**

* **Usability**: No previous training time. System is simple and easy to follow.
* **Reliability**: The system should perform correctly 99% of the time.
* **Performance**: The system should be sent and saved within 2 seconds.
* **Supportability**: The system should be easy to maintain, make appropriate changes, and be correctly handled by IE, Mozilla, Chrome and Safari.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

*Use Case ID:* **VQ17 – Visitor DeQueue**

*Descriptions:* If a user does not want to go on a particular ride they have queued up for. This allows them to remove themselves from the queue for that ride.

*User Story:* #121 Visitor Dequeue for rides

*Actors:* Single Venue Admin and Venue Visitor/Non Visitor.

*Pre-condition:*

1. Admin or visitor logged in. Non Visitor does not have access.

*Description:*

1. Use case begins when a visitor user clicks on MyAccount Tab on the VQ home webpage.

2. The User will have a freshly updated list of Rides and their concurrent waiting times.

3. The Visitor can click on the “Remove Ride” button located to the right of each of the wait times.

4. Use case ends when the selected ride becomes removed from the Queue as well as the users account and displayed within the user account under “My Account”.

*Post-conditions:*

1. The system shall update the waiting time as users queue.

2. The system shall update the admin or Visitor account with newly selected rides.

*Alternative Courses of Action:*

1. In Step 1 Users can click on My Account and click on Ride tab within to access the Ride List.

*Exceptions*

1. The system could not find wait time.

2. The system could not remove from queue.

3. The system could not find activities.

*Related Uses case:* **VQ15 – Find Wait Time,****VQ16 – Add Queue**

**Special Requirements:**

· **Usability**: No previous training time. System is simple and easy to follow.

· **Reliability**: The system should perform correctly 99% of the time.

· **Performance**: The system should be sent and saved within 2 seconds.

· **Supportability**: The system should be easy to maintain, make appropriate changes, and be correctly handled by IE, Mozilla, Chrome and Safari.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

*Use Case ID:* **VQ18 – Simulate Add Visitors to Rides**

* *Descriptions:* As an admin, I should have the controllers be able to add random visitor records to a particular ride, so to compare wait times for a ride and also to increase the wait time.

*User Story:* #111 **Simulate Add Visitors to Rides**

*Actors:*  Admin

*Pre-condition:*

1. Admin is logged in.

*Steps:*

1. Use case begins when admin clicks on Simulate Ride tab within the admin tab in the homepage.

2. system will display a pop up and the table within is already populated Then in the Simulate Ride Widow will pop up and the table within is already populated with all active rides and total # of records queued in that venue.

3. The actor clicks on add records button in simulate section of a table to the rides they want to add records to.

4. The system will display the Add Records to Ride window will open where the admin enters number of dummy records they want to add in the text box.

5. The admin clicks on Submit button in the Add Records to Ride window.

6. Use case ends the Admin is taken back to the Simulate Ride Window.

*Post-conditions:*

1. The system shall update the queue, by adding # of dummy records to the queue to the specific ride selected.

2. The system shall update the number of queue records on the Admin Ride Table for that specific ride selected.

*Alternative Courses of Action:*

1. In step 2 admin can enter value 0 (in Admin Queues text box) and press Admin Add Queue button for a ride, then no records will be added.

*Exceptions*

1. The system could not find list of rides.

2. Invalid value entered in the Admin Queues text box.

3. The system could not add # of records to that ride.

4. Admin cannot add more then 100 records at a time.

*Related Uses case:* **VQ19 – Simulate Dequeue Vistors from Rides**

· **Usability**: No previous training time. System is simple and easy to follow.

· **Reliability**: The system should perform correctly 99% of the time.

· **Performance**: The system should be sent and saved within 1 seconds.

· **Supportability**: The system should be easy to maintain, make appropriate changes, and be correctly handled by IE, Mozilla, Chrome and Safari.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

*Use Case ID:* **VQ19 – Simulate Dequeue Visitors from Rides***Descriptions:* Admin has ability to remove queues records for particular rides.

*User Story:* #90 Simulate Queue for admin: Add Multiple Visitors to Rides

*Actors:* Single Venue Admin

*Pre-condition:*

1. Admin is logged in.

*Steps:*

1. Use case begins when admin clicks on admin ride tab.

2. Then the Admin Ride Table will be populated with all active rides and total # of records queued in that venue.

3. Use case ends the Admin presses the Admin Remove Queue button for specific ride from the table.

*Post-conditions:*

1. The system shall update the queue, by removing # equal to the rides max capacity.

2. The system shall update the number of queue records on the Admin Ride Table for that specific ride.

*Alternative Courses of Action:*

1. In step 3, User does not press the Admin Remove Queue button.

*Exceptions*

1. The system could not find list of rides.

2. The system could not remove # equal to the rides max capacity.

3. The system does not remove the correct number of records from that queue.

4. The system removes some other rides that was not selected.

*Related Uses case:* **VQ18 – Simulate Add Visitors to Rides**

· **Usability**: No previous training time. System is simple and easy to follow.

· **Reliability**: The system should perform correctly 99% of the time.

· **Performance**: The system should be sent and deleted within 1 seconds.

· **Supportability**: The system should be easy to maintain, make appropriate changes, and be correctly handled by IE, Mozilla, Chrome and Safari.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

*Use Case ID:* **VQ20 – Dynamic Find Wait Time**

*Descriptions:* As an user and admin, I should be able to see that the wait time for the rides that I am queued up to change depending on how many visitors is queued up in front of me.

*User Story:* #97 Dynamic Find Wait Time

*Actors:* Admin and Visitor

*Pre-condition:*

1. Admin/Visitor is logged in. Admin/Visitor have already queued up to a ride.

*Description:*

1. Use case begins when admin/visitor clicks on My Account tab above the VQ home webpage.
2. The window will already populated with whatever ride admin/ride is queued up to.
3. Use case ends when admin/visitor clicks on refresh button and the ride wait time decreases only if the ride was dequeued and updated the table with new information.

*Post-conditions:*

1. The system shall update the wait time for admin/visitor when the ride is dequeued.

*Alternative Courses of Action:*

1. In Step 3 the admin/visitor does not click the refresh button and wait time does not get updated.

*Exceptions*

1. The system could not find the dynamic wait time.
2. The system could not find visitor queued to any records.

*Related Uses case:* **VQ15 – Find Wait Time**

**Special Requirements:**

* **Usability**: No previous training time. System is simple and easy to follow.
* **Reliability**: The system should perform correctly 99% of the time.
* **Performance**: The system should be sent and saved within 2 seconds.
* **Supportability**: The system should be easy to maintain, make appropriate changes, and be correctly handled by IE, Mozilla, Chrome and Safari.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

*Use Case ID:* **VQ22 – Send Notifications for adding ride**

*Descriptions:* As a Visitor or an Admin, I should be notified when I queued myself to a ride and when its almost time to get onto the ride.

*User Story:* #112 Send Notifications

*Actors:* Admin & Visitor

*Pre-condition:*

1. Admin & Visitor is logged in.

*Description:*

1. Use case begins when admin/visitor clicks on My Account tab above the VQ home webpage.
2. Then the admin/visitor clicks on the Ride Tab within the My account tab, which Ride window will open.
3. Admin/visitor then queue themselves on the rides they want to get onto, by pressing Add Ride button.
4. Admin/visitor should exit out of the ride window by pressing the cancel button or x button.
5. Use case ends when the admin/visitor receives the email that they are queued up to a Ride. The email provides what the wait time and what position the Visitor is in the line.

*Post-conditions:*

1. System shall send another email to Visitors when they are almost ready to get onto a ride.

*Alternative Courses of Action:*

1. In Step 3, Visitor does not queue up to a ride.
2. The Admin/Visitor does not follow 2-5 steps. Instead they are already queued to ride they want to. When the ride dequeues as times goes on, and that user finally reaches 10 minute mark in that queue of the ride. The admin/visitor will receive email notifying that they have less than 10 minutes to go on the ride.

*Exceptions*

1. The system could not add admin/visitor to the Ride.
2. The system could not send email to the admin/visitor.

*Related Uses case:* VQ23 – Send Notifications for time warning

**Special Requirements:**

* **Usability**: No previous training time. System is simple and easy to follow.
* **Reliability**: The system should perform correctly 99% of the time.
* **Performance**: The system should be sent and saved within 2 seconds.
* **Supportability**: The system should be easy to maintain, make appropriate changes, and be correctly handled by IE, Mozilla, Chrome and Safari.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

*Use Case ID:* **VQ23 – Send Notifications for time warning**

*Descriptions:* As a Visitor or an Admin, I should be notified when I queued myself to a ride and when it's almost time to get onto the ride.

*User Story:* #112 Send Notifications

*Actors:* Admin & Visitor

*Pre-condition:*

1. Admin & Visitor is logged in and already queued up to a ride.

*Description:*

1. Use case begins when admin/visitor clicks on My Account tab above the VQ home webpage.
2. Use case ends when one of the rides that the admin/visitor is waiting for is dequeued and that users wait time becomes 10 minutes or less. At that point the system will send email notifications to each admin/visitor who has 10 minutes or less wait time for that ride.

*Post-conditions:*

1. System shall send another email to Visitors when they are almost ready to get onto a ride.

*Alternative Courses of Action:*

1. In Step 1 does not click on My Account rab to check his/her wait time.

*Exceptions*

1. Admin/Visitor is not queued up to the ride.
2. The system could not send email to the admin/visitor.

*Related Uses case:* VQ22 – Send Notifications for adding ride

**Special Requirements:**

* **Usability**: No previous training time. System is simple and easy to follow.
* **Reliability**: The system should perform correctly 99% of the time.
* **Performance**: The system should be sent and saved within 2 seconds.
* **Supportability**: The system should be easy to maintain, make appropriate changes, and be correctly handled by IE, Mozilla, Chrome and Safari.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

* *Use Case ID:* **VQ24 – Admin Modify Rides**
* *Descriptions:* As an Admin, I should be able to modify the ride information to accompany the many changes that may occur through the life of the venue

*User Story:* #119 Simulate Queue for admin: Add Multiple Visitors to Rides

*Actors:* Single Venue Admin

*Pre-condition:*

1. Admin is logged in.

*Steps:*

1. Use case begins when admin clicks on admin tab.

2. Then the Admin Ride Table will be populated with all active rides and total # of records queued in that venue.

3 The Admin presses the Edit ride button for specific ride from the table.

4. The admin user must than fill out the pop up window with the Interval Time, Entry Time, Exit Time, Max Capacity, and Concurrency ride.

5. The use case ends when the admin hits the submit button from this window and accepts the changes made.

*Post-conditions:*

1. The system shall update the ride table, updating the corresponding elements in the table according to what was added.

*Alternative Courses of Action:*

1. N/A

*Exceptions*

1. The system could not find list of rides.

2. The system could not update the Interval Time .

3. The system could not update the Entry Time. 4. The system could not update the Exit Time.

5. The system could not update the Max Capacity.

6. The system could not update the Concurrency ride.

*Related Uses case:* **N/A**

· **Usability**: No previous training time. System is simple and easy to follow.

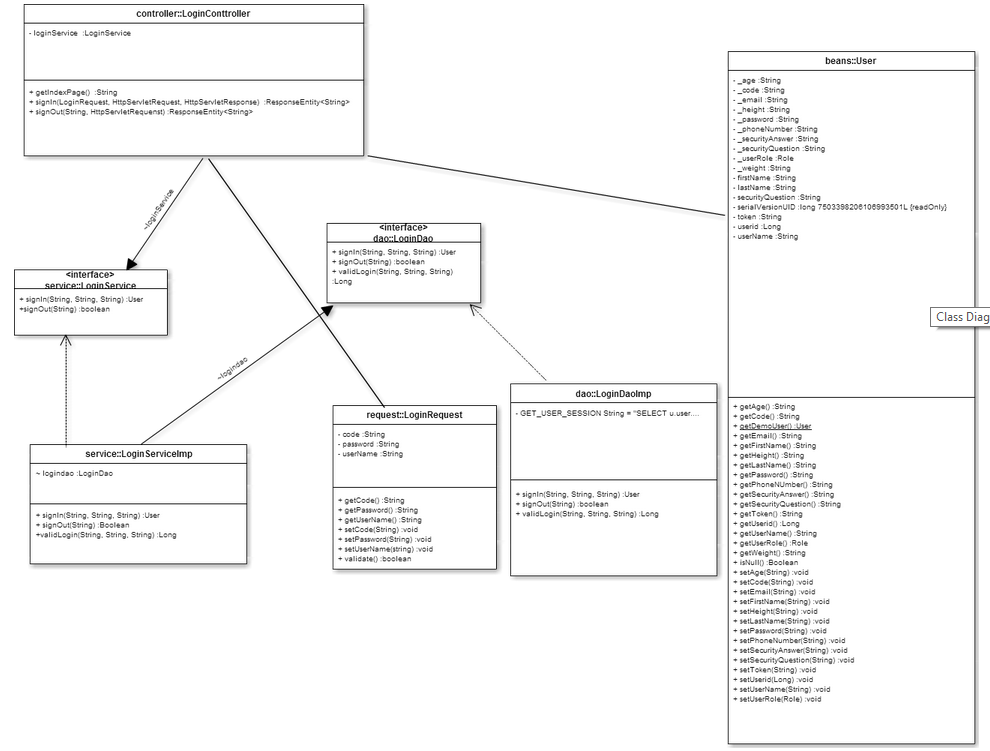
· **Reliability**: The system should perform correctly 99% of the time.

· **Performance**: The system should be sent and deleted within 1 seconds.

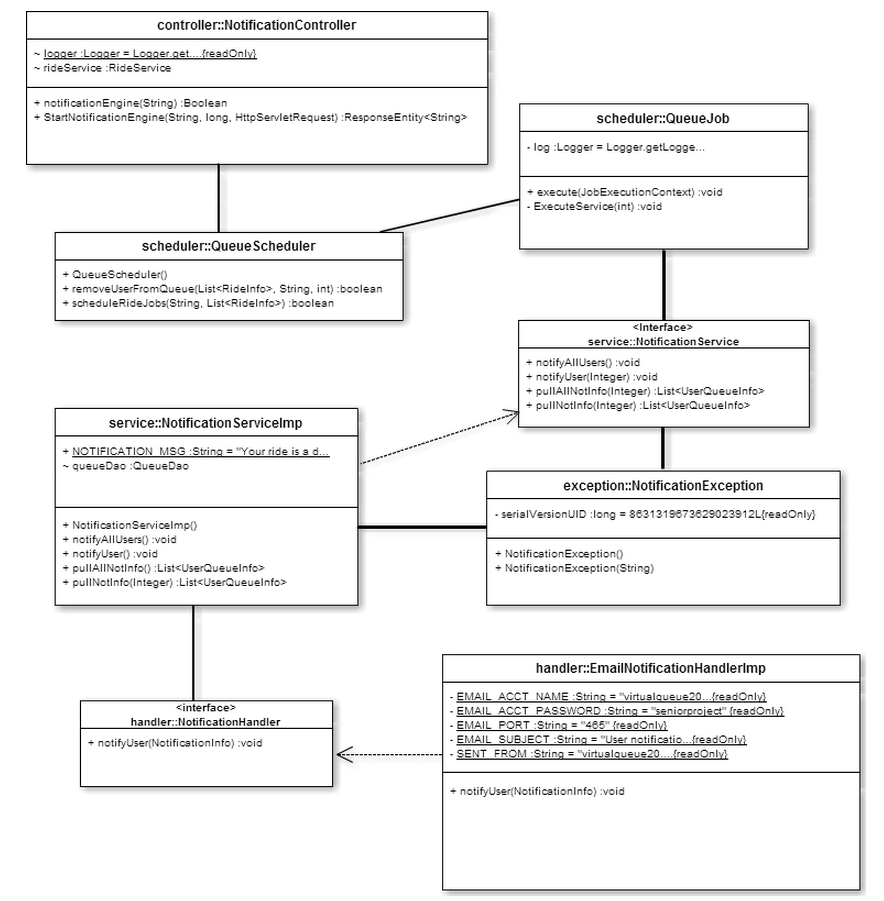
· **Supportability**: The system should be easy to maintain, make appropriate changes, and be correctly handled by IE, Mozilla, Chrome and Safari.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

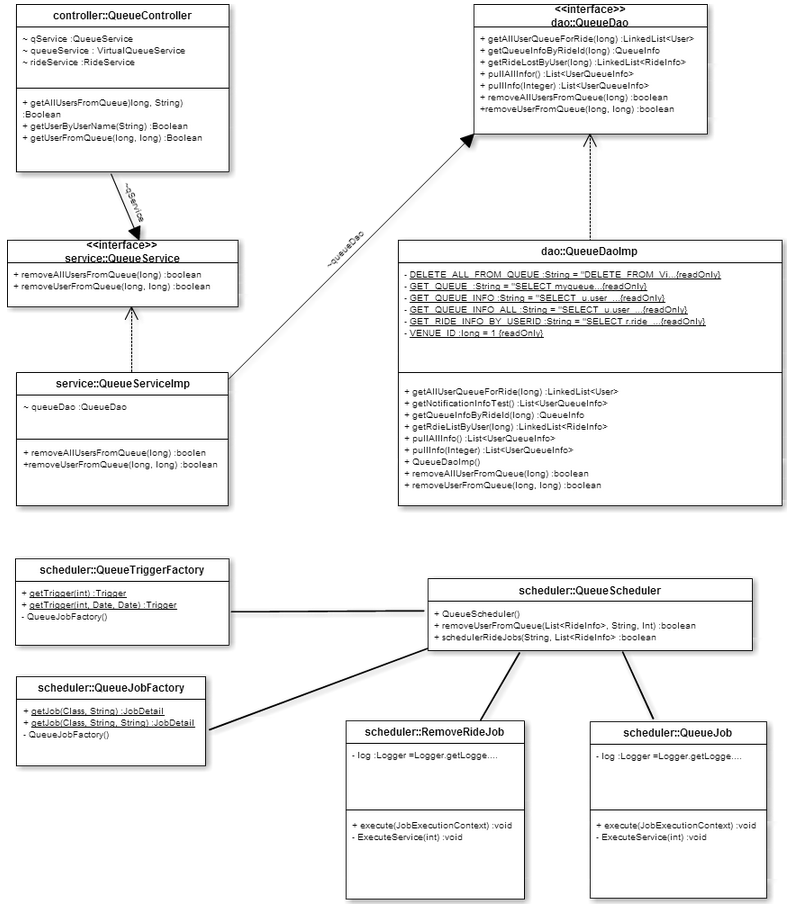
* 1. Appendix C – User Interface designs.



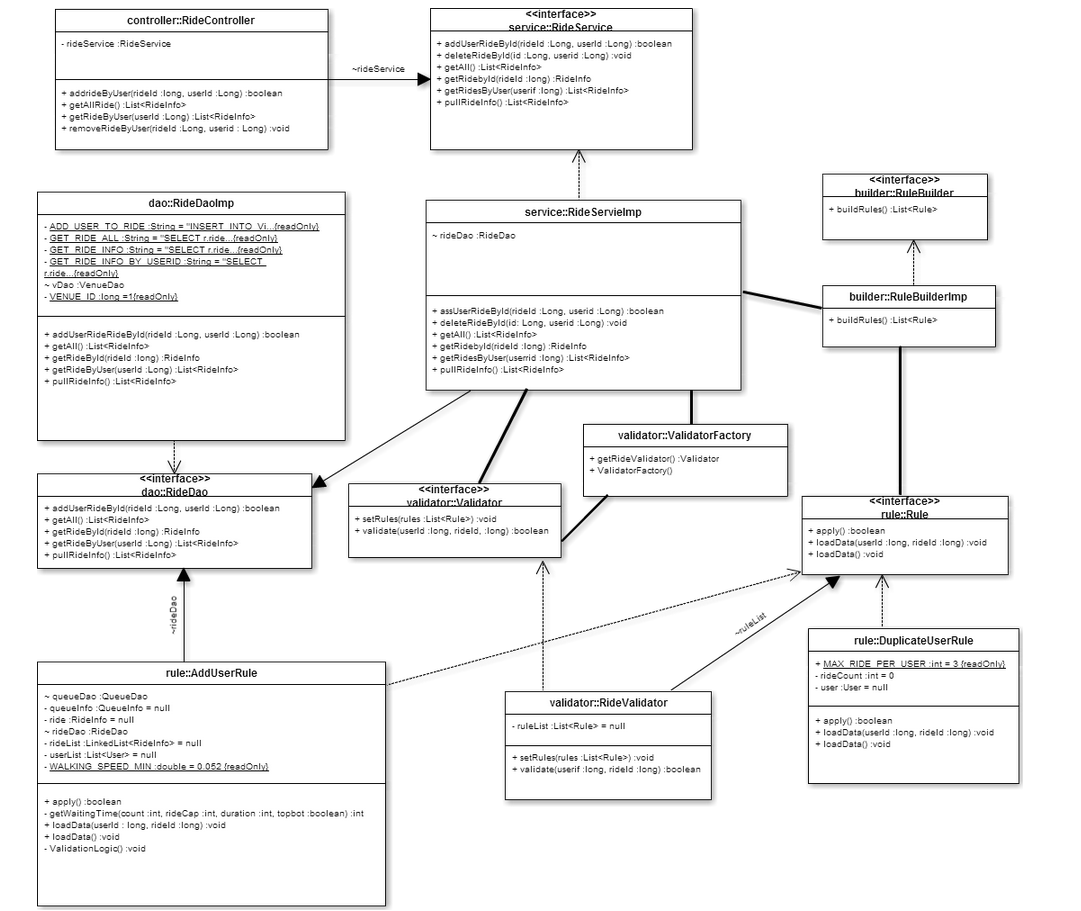
**Fig 3.2.1: Login Operation Subsystem class diagram (part of the whole VQ class Diagram):**



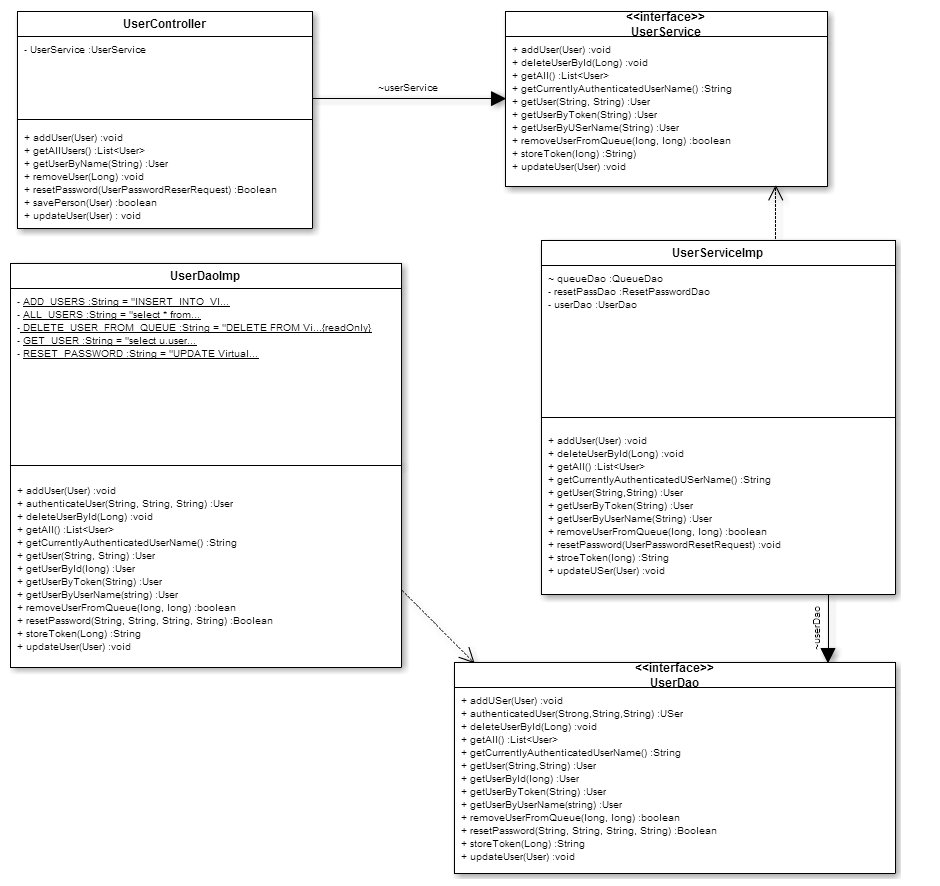
**Fig 3.2.2: Notification API Subsystem class diagram (part of the whole VQ class Diagram)**



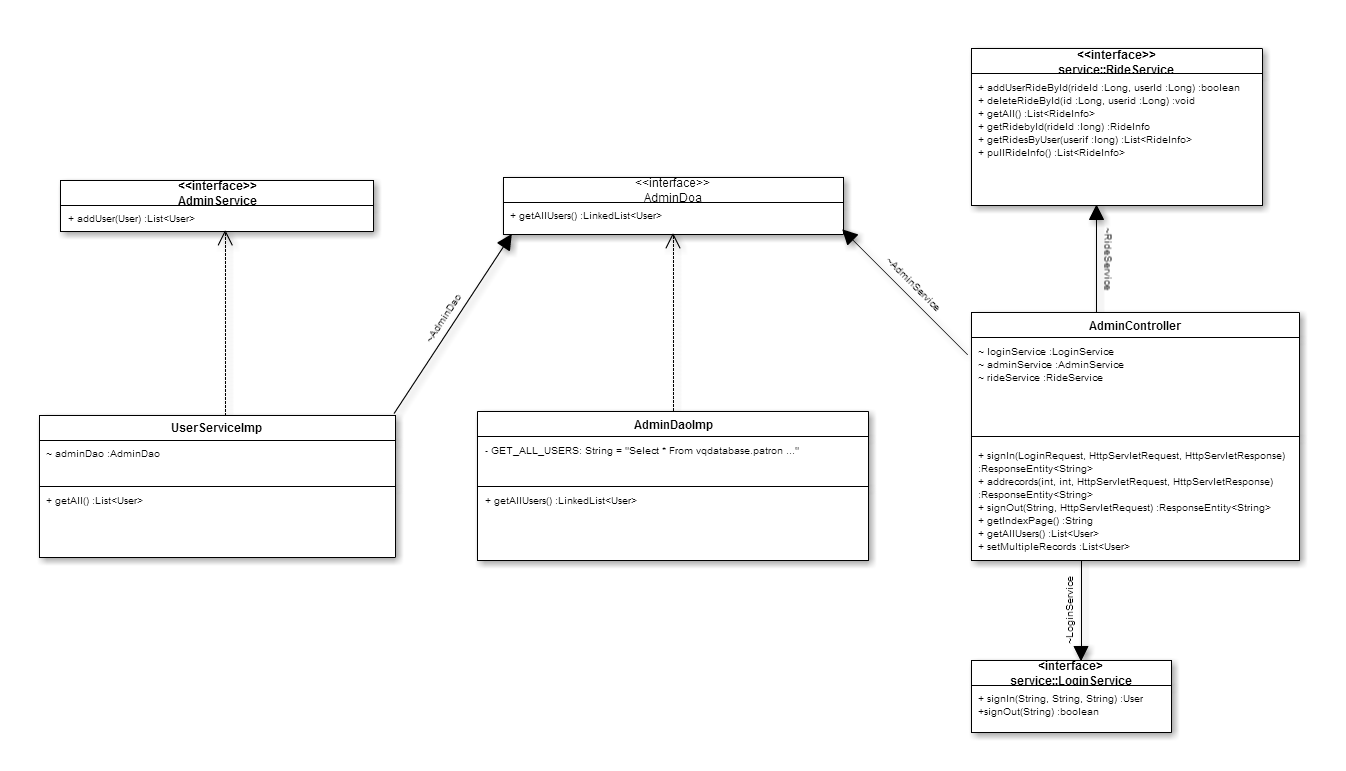
**Fig 3.2.3: Queue Scheduler Subsystem class diagram (part of the whole VQ class Diagram)**



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

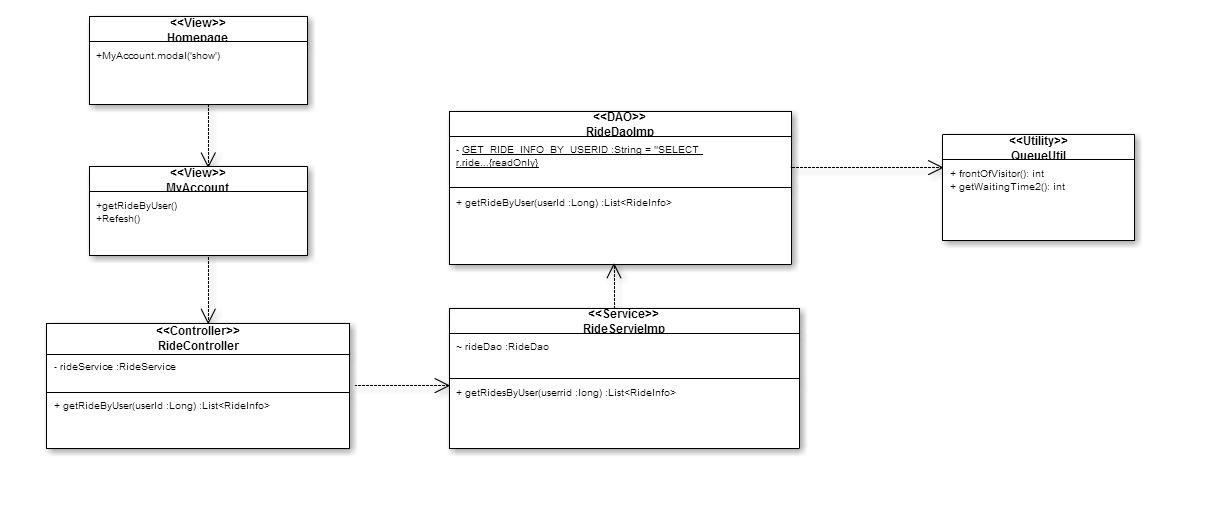


**Fig 3.2.5: User Operation Subsystem class diagram (part of the whole VQ class Diagram)**

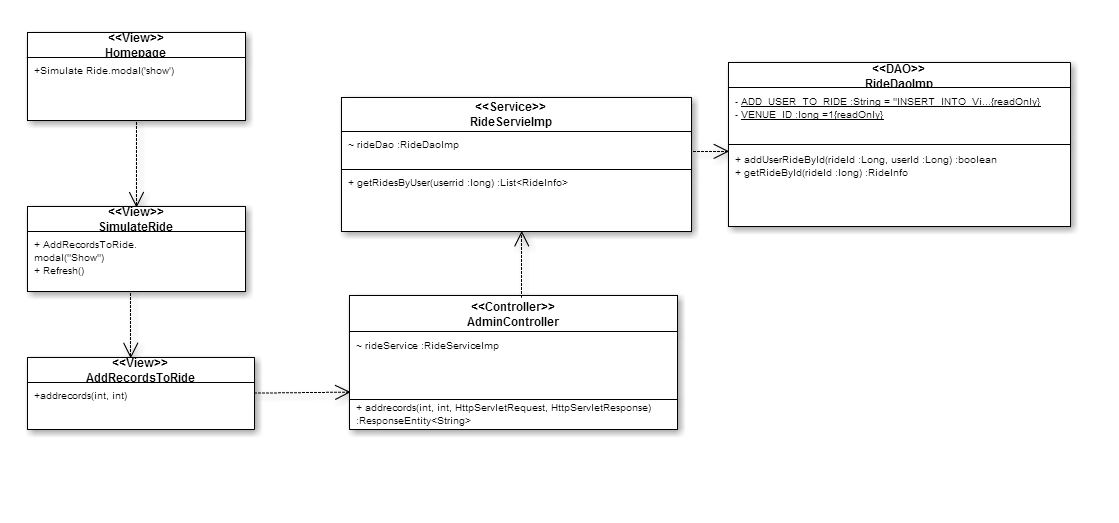


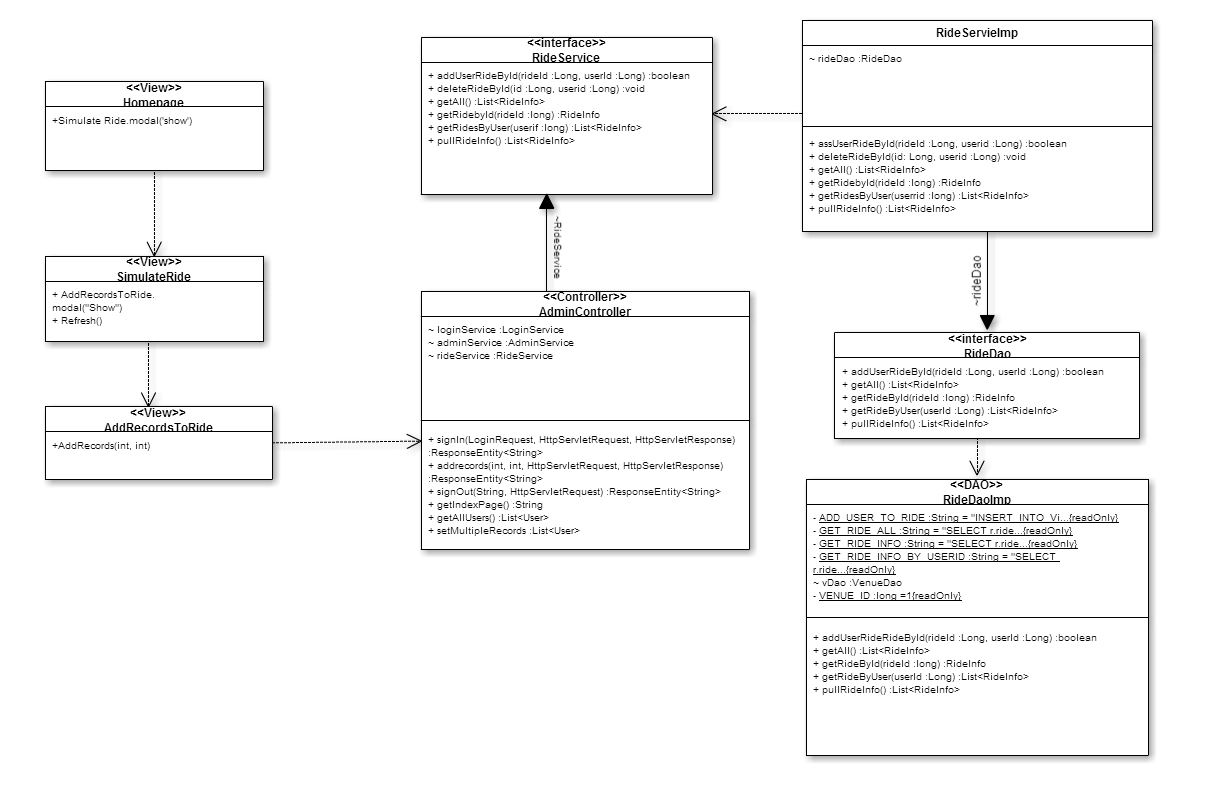
**Fig 3.2.6: Admin Operation Subsystem class diagram (part of the whole VQ class Diagram)**

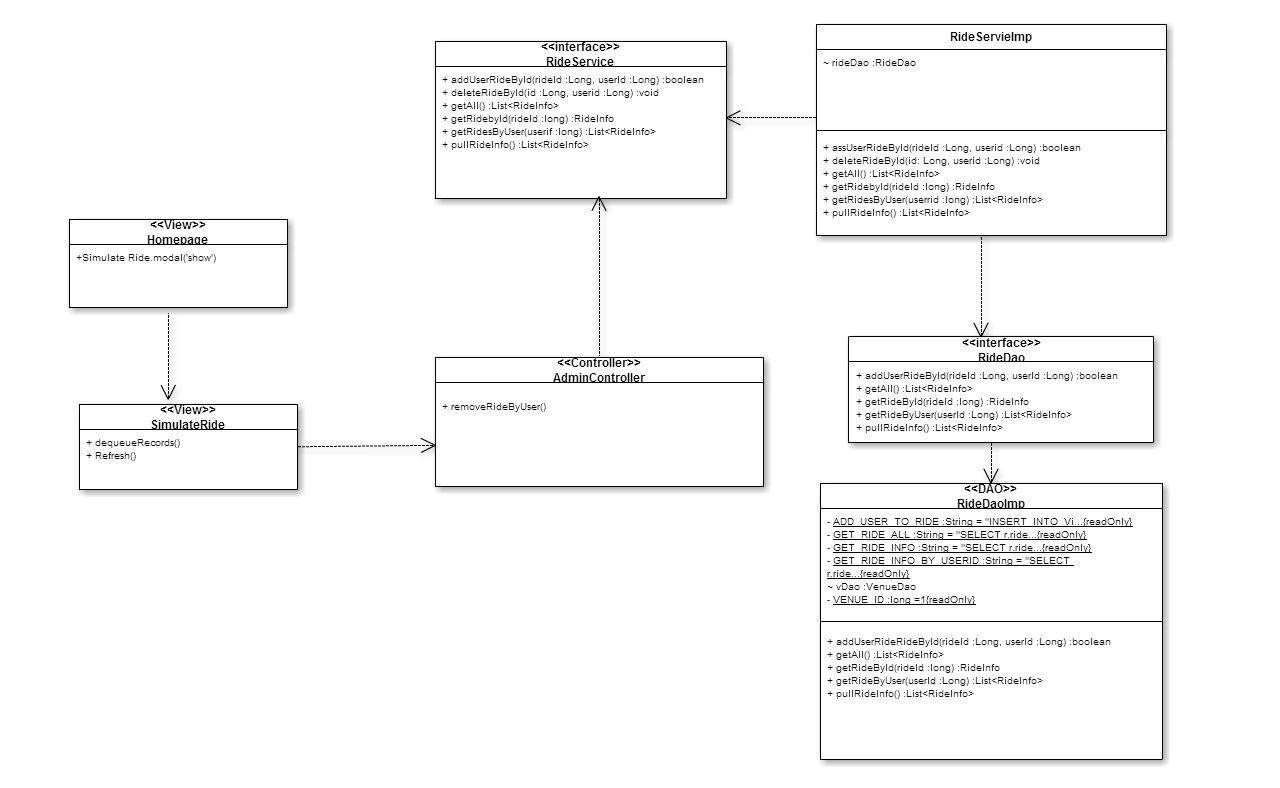
**VQ15 – Find Wait Times**

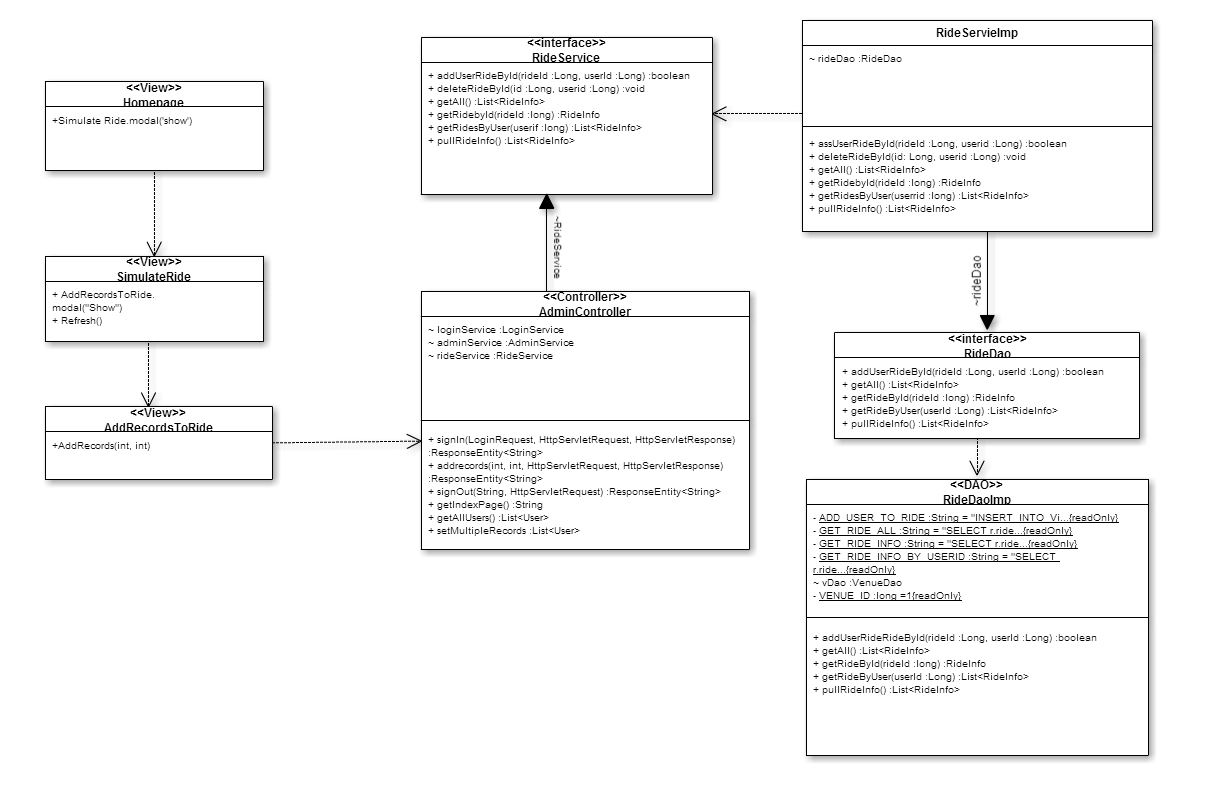
****

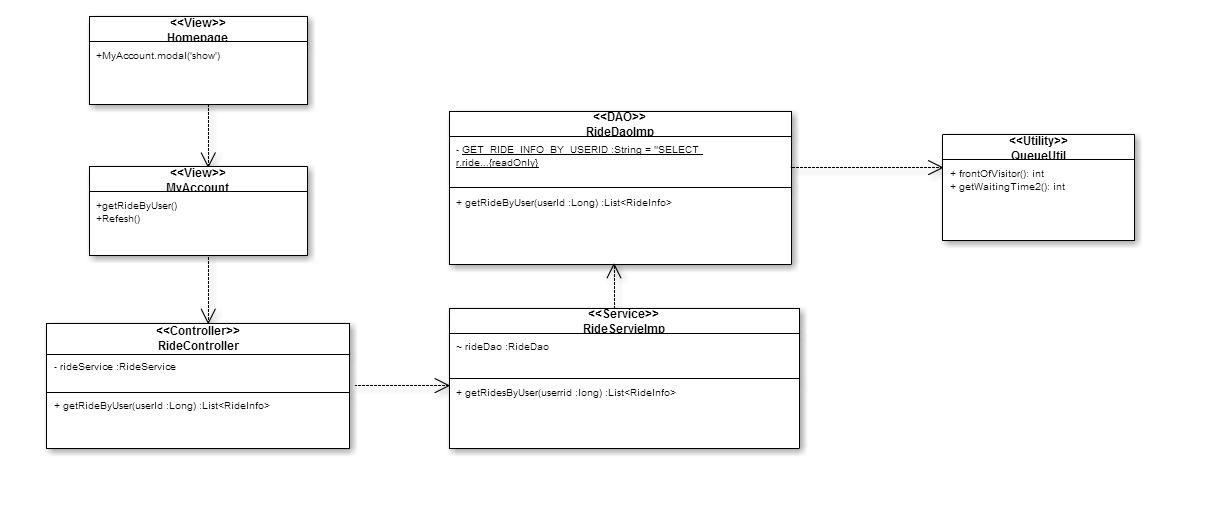
**VQ16 – Add Queue**



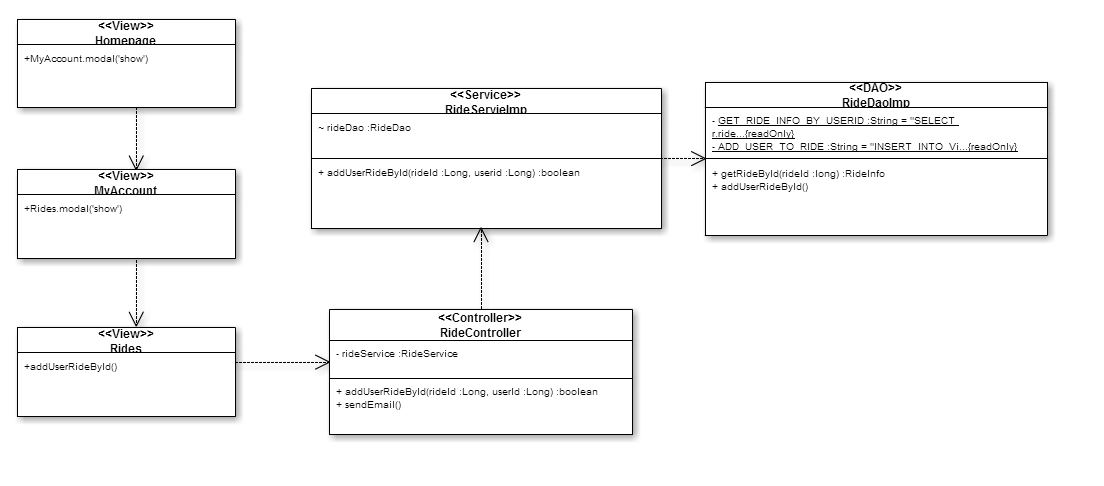
**VQ17 – Visitor DeQueue** ****

**VQ18 – Simulate Add Ride**

**VQ19 – Admin Simulate Dequeue for Rides**

**VQ20 – Dynamic Find Wait Time**

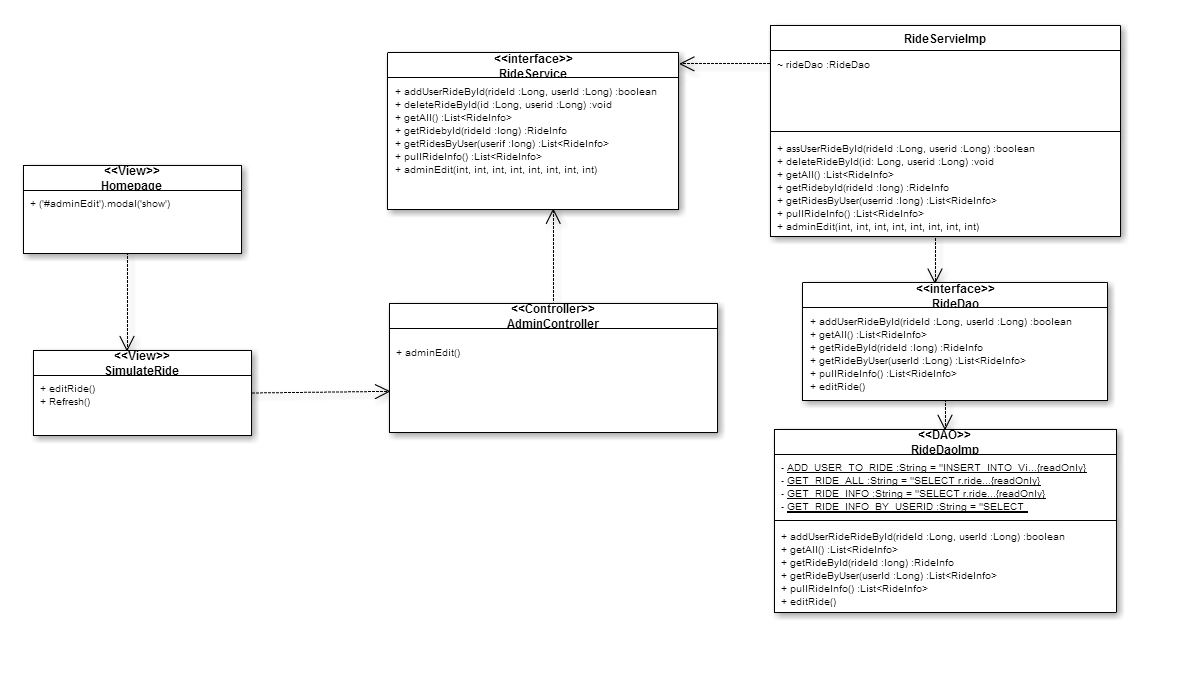
**VQ22 – Send Notification for adding ride**

****

**VQ23 – Send Notification for time warning**

****

**VQ24 – Admin Modify Rides**

****

* 1. Appendix D – Analysis models (static and dynamic)

VQ01-Login:



VQ02-Logout:



VQ03-Reset Password:



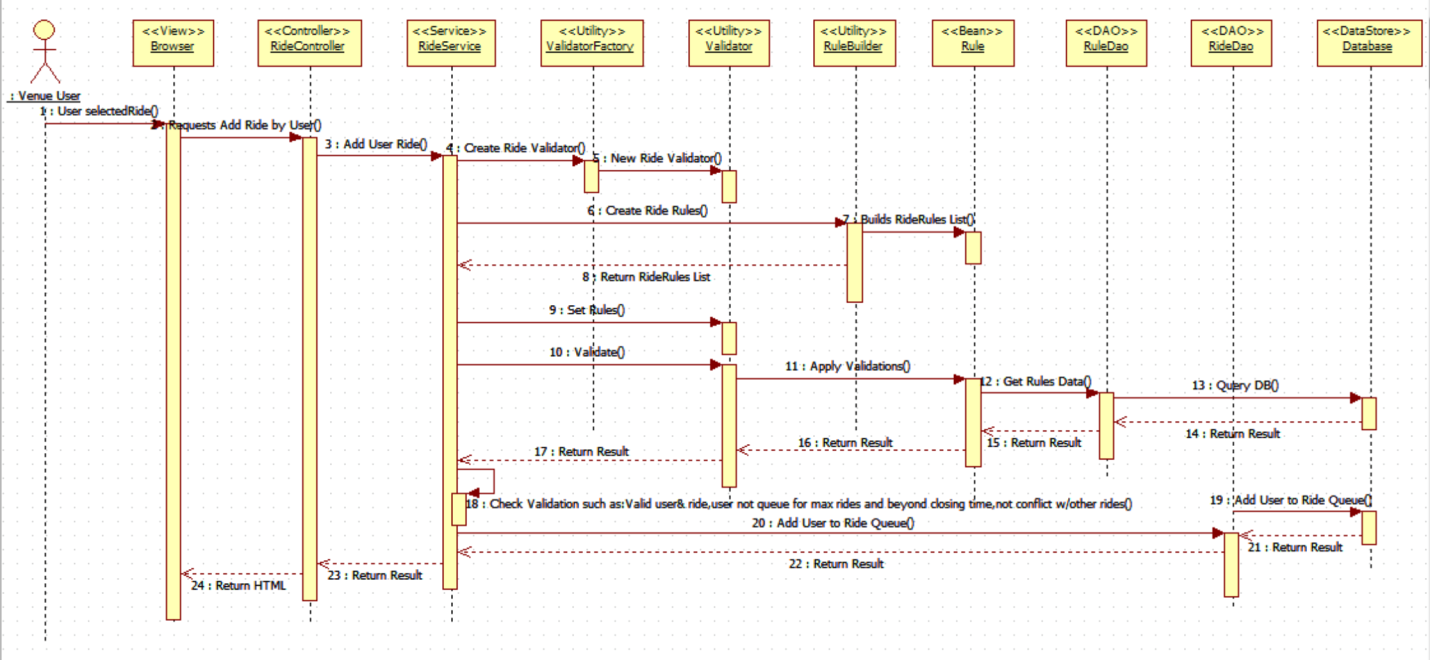
VQ05 - Register New User

****

VQ06 – Select Ride



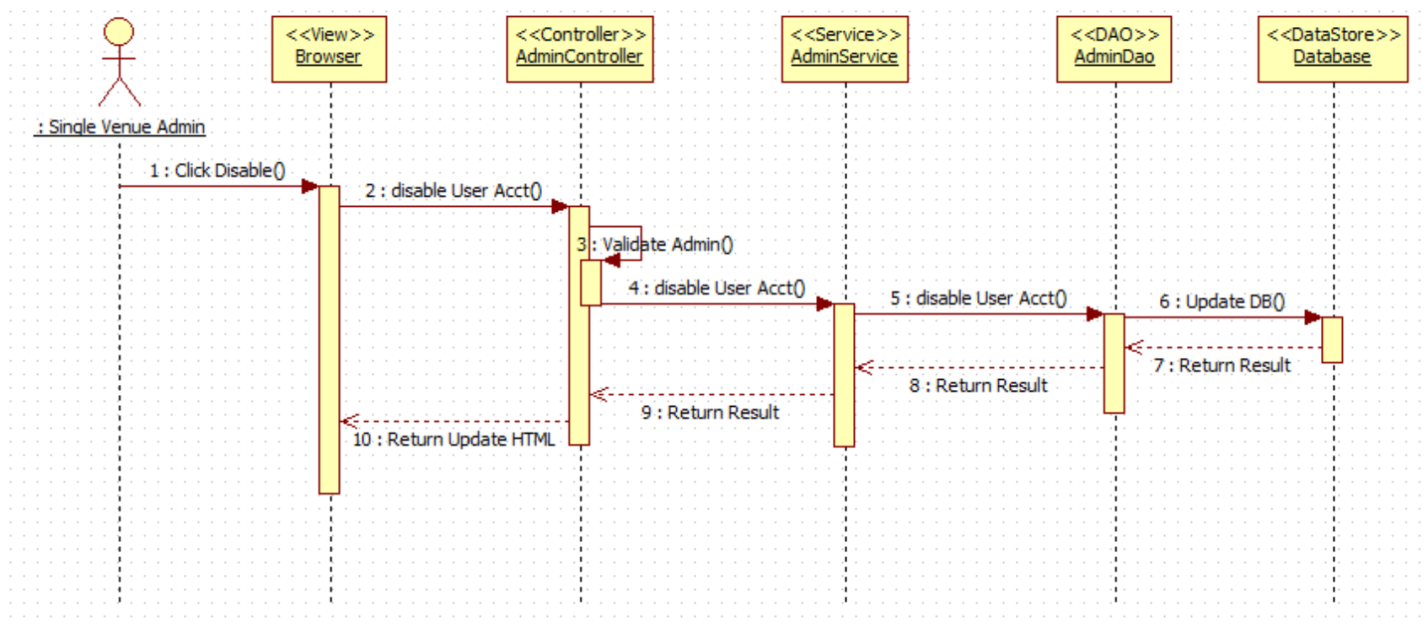
VQ07 – Add Ride (Queue User for Ride)



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)



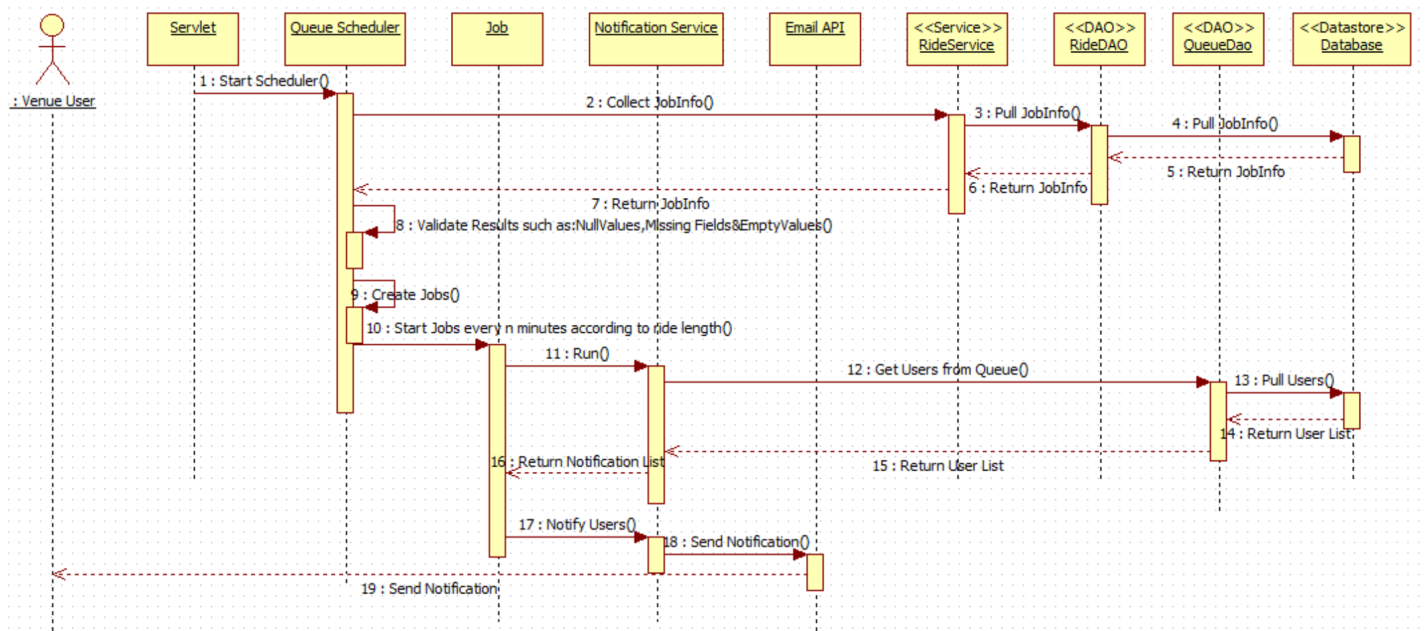
VQ12 – View Available Rides



VQ13 – View User Activity



VQ14 – Receive Notification (before ride time approaches)



VQ15 – Find Wait Times

## VQ16 – Add Queue

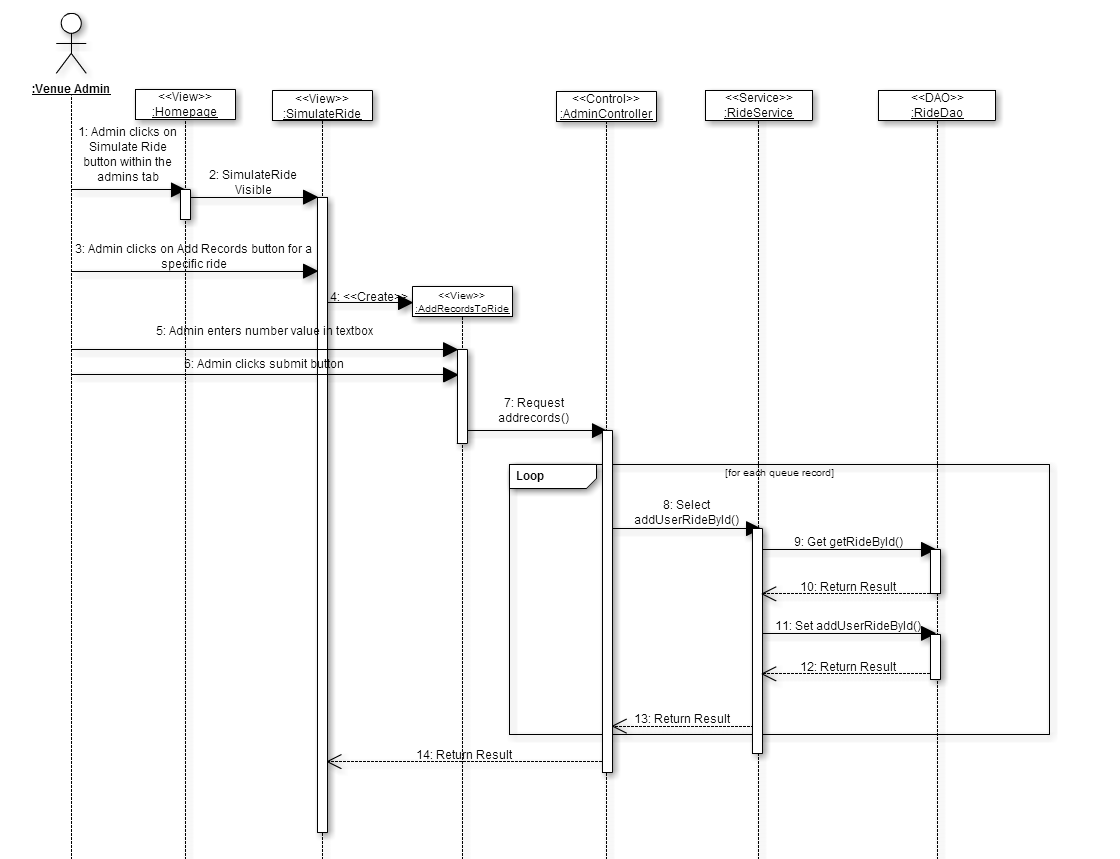
## 

## 

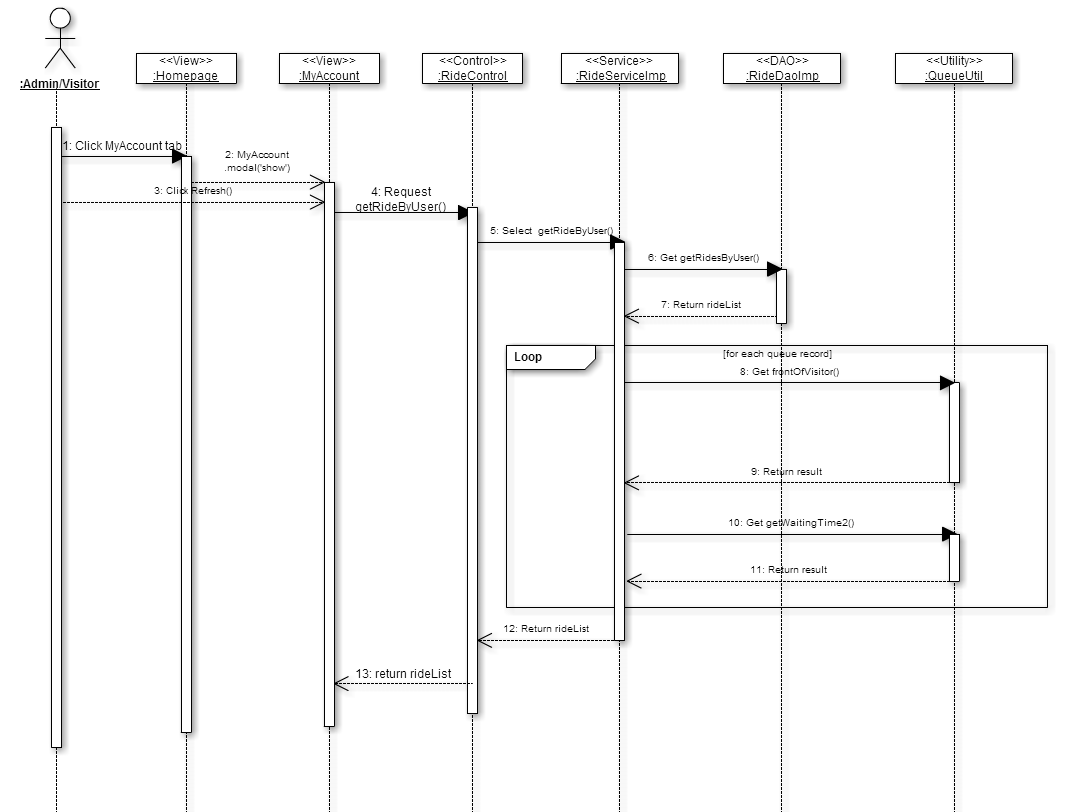
**VQ17 – Visitor DeQueue**

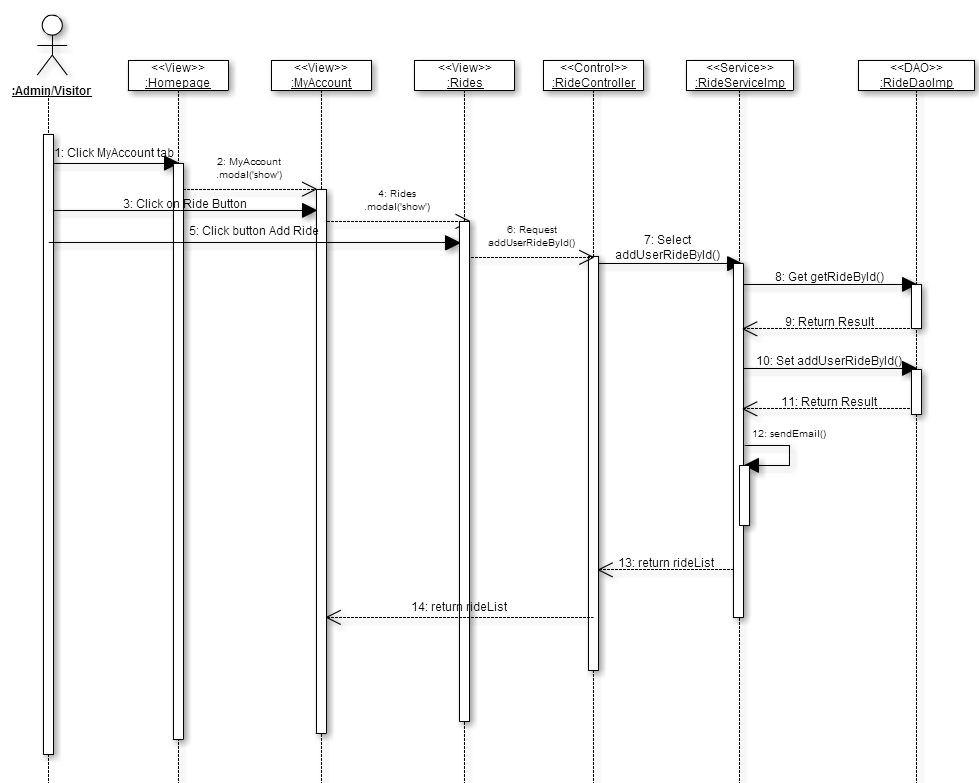


**VQ18 – Simulate Add Ride**

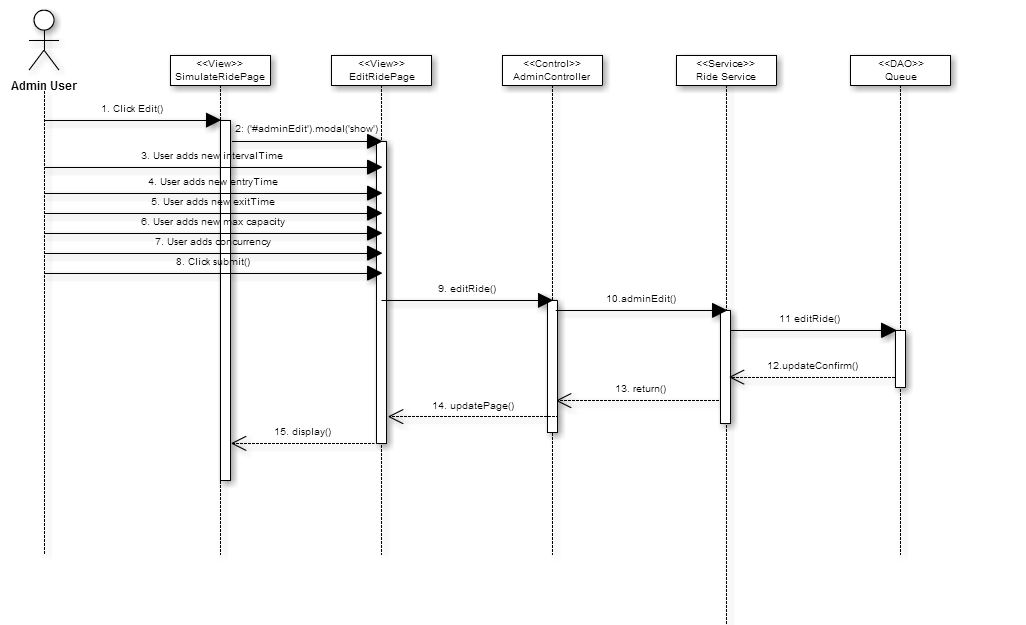


**VQ19 –Admin Simulate Dequeue Rides**

**VQ20 – Dynamic Find Wait Time**

**VQ22 – Send Notification for adding ride**

**VQ23 – Send Notification for time warning**

**VQ24 – Admin Modify Rides**

* 1. Appendix E – Design models (static and dynamic)

**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();

}

* 1. Appendix F – Documented Class interfaces (code) and constraints.
  2. Appendix G – Documented code for test drivers and stubs.

Test Drivers and Stubs for Email Notification

package com.test.spring;

import static org.junit.Assert.\*;

import org.junit.After;

import org.junit.AfterClass;

import org.junit.Before;

import org.junit.BeforeClass;

import org.junit.Test;

/\*\*

\* @author SoftwareDevelopment

\*

\*/

public class NotificationServiceImpTest {

/\*\*

\* @throws java.lang.Exception

\*/

@BeforeClass

public static void setUpBeforeClass() throws Exception {

}

/\*\*

\* @throws java.lang.Exception

\*/

@AfterClass

public static void tearDownAfterClass() throws Exception {

}

/\*\*

\* @throws java.lang.Exception

\*/

@Before

public void setUp() throws Exception {

}

/\*\*

\* @throws java.lang.Exception

\*/

@After

public void tearDown() throws Exception {

}

/\*\*

\* Test method for {@link com.virtual.queue.service.NotificationServiceImp#NotificationServiceImp()}.

\*/

@Test

public void testNotificationServiceImp() {

}

/\*\*

\* Test method for {@link com.virtual.queue.service.NotificationServiceImp#pullNotInfo()}.

\*/

@Test

public void testPullNotInfo() {

}

/\*\*

\* Test method for {@link com.virtual.queue.service.NotificationServiceImp#notifyUser()}.

\*/

@Test

public void testNotifyUser() {

}

}

Mockup for NotificationDao:

**package** com.test.spring.mockups;

**import** java.util.ArrayList;

**import** java.util.List;

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

**import** com.virtual.queue.service.NotificationService;

**public** **class** NotificationServiceImpMock **implements** NotificationService {

@Override

**public** List<UserQueueInfo> pullNotInfo(**long** rideId) {

**if** (rideId > 0) {

**return** **null**;

}

List<UserQueueInfo> list = **new** ArrayList<UserQueueInfo>();

UserQueueInfo info = **new** UserQueueInfo();

info.setEmail("test@test.com");

list.add(info);

**return** list;

}

@Override

**public** List<UserQueueInfo> pullAllNotInfo() {

List<UserQueueInfo> list = **new** ArrayList<UserQueueInfo>();

UserQueueInfo info = **new** UserQueueInfo();

info.setEmail("test@test.com");

list.add(info);

**return** list;

}

@Override

**public** **void** notifyUser(**long** rideId) **throws** Exception {

**if**(rideId==0)**throw** **new** Exception("invalid rideId");

}

@Override

**public** **void** notifyAllUsers() **throws** Exception {

System.***out***.println("notifyAllUser");

}

}

Test Drivers and Stubs for Login

**public** **class** LoginDaoImpTest {

LoginDao dao;

@BeforeClass

**public** **static** **void** setUpBeforeClass() **throws** Exception {

}

@AfterClass

**public** **static** **void** tearDownAfterClass() **throws** Exception {

}

@Before

**public** **void** setUp() **throws** Exception {

dao=**new** LoginDaoImp();

}

@After

**public** **void** tearDown() **throws** Exception {

}

@Test

**public** **void** testSignOut() {

}

@Test

**public** **void** testValidLogin() {

}

@Test

**public** **void** testSignIn() {

}

}

package com.test.spring;

import static org.junit.Assert.\*;

import org.junit.After;

import org.junit.AfterClass;

import org.junit.Before;

import org.junit.BeforeClass;

import org.junit.Test;

import com.test.spring.mockups.LoginDaoImpMock;

import com.virtual.queue.beans.User;

import com.virtual.queue.dao.LoginDao;

import com.virtual.queue.service.LoginService;

import com.virtual.queue.service.LoginServiceImp;

public class LoginServiceImpTest {

LoginDao dao;

LoginService service;

@BeforeClass

public static void setUpBeforeClass() throws Exception {

}

@AfterClass

public static void tearDownAfterClass() throws Exception {

}

@Before

public void setUp() throws Exception {

dao=new LoginDaoImpMock();

service= new LoginServiceImp(dao);

}

@After

public void tearDown() throws Exception {

}

@Test

public void testSignIn() {

User user= service.signIn("test", "passwrod", "code");

assertNotNull(user);

assertEquals(user.getUserName(), user.getDemoUser().getUserName());

}

@Test

public void testSignOut() {

//not implemented

}

}

LoginDao Mockup

package com.test.spring.mockups;

import com.virtual.queue.beans.User;

import com.virtual.queue.service.LoginService;

public class LoginServiceImpMock implements LoginService {

@Override

public User signIn(String userName, String password, String code) {

if (userName == null || password == null)

return null;

User user = new User();

user.setUserid(1l);

user.setFirstName("FirstName");

return user;

}

@Override

public boolean signOut(String userName) {

if (userName == null)

return false;

return true;

}

@Override

public User adminSignIn(String userName, String password) {

if (userName == null)

return null;

User user = new User();

user.setUserid(1l);

user.setFirstName("FirstName");

return user;

}

}

Ride Stubs and Drivers:

package com.test.spring;

import static org.junit.Assert.\*;

import org.junit.After;

import org.junit.AfterClass;

import org.junit.Before;

import org.junit.BeforeClass;

import org.junit.Test;

public class RideDaoImpTest {

@BeforeClass

public static void setUpBeforeClass() throws Exception {

}

@AfterClass

public static void tearDownAfterClass() throws Exception {

}

@Before

public void setUp() throws Exception {

}

@After

public void tearDown() throws Exception {

}

@Test

public void testPullRideInfo() {

}

@Test

public void testPullRideInfoLong() {

}

}

package com.test.spring;

import static org.junit.Assert.\*;

import java.util.List;

import org.junit.After;

import org.junit.AfterClass;

import org.junit.Before;

import org.junit.BeforeClass;

import org.junit.Test;

import com.test.spring.mockups.QueueDaoImpMock;

import com.test.spring.mockups.RideDaoImpMock;

import com.test.spring.mockups.UserDaoImpMock;

import com.virtual.queue.beans.Ride;

import com.virtual.queue.beans.RideInfo;

import com.virtual.queue.dao.QueueDao;

import com.virtual.queue.dao.RideDao;

import com.virtual.queue.dao.UserDao;

import com.virtual.queue.exception.NotificationException;

import com.virtual.queue.service.RideService;

import com.virtual.queue.service.RideServiceImp;

/\*\*

\* @author SoftwareDevelopment

\*

\*/

public class RideServiceImpTest {

RideService service;

RideDao rDao;

QueueDao qDao;

UserDao uDao;

Ride ride;

/\*\*

\* @throws java.lang.Exception

\*/

@BeforeClass

public static void setUpBeforeClass() throws Exception {

}

/\*\*

\* @throws java.lang.Exception

\*/

@AfterClass

public static void tearDownAfterClass() throws Exception {

}

/\*\*

\* @throws java.lang.Exception

\*/

@Before

public void setUp() throws Exception {

rDao = new RideDaoImpMock();

qDao = new QueueDaoImpMock();

uDao = new UserDaoImpMock();

ride = new Ride();

ride.setrName("RideName");

service=new RideServiceImp( rDao,qDao,uDao);

}

/\*\*

\* @throws java.lang.Exception

\*/

@After

public void tearDown() throws Exception {

}

/\*\*

\* Test method for {@link com.virtual.queue.service.RideServiceImp#getAll()}

\* .

\*/

@Test

public void testGetAll() {

try {

List<RideInfo> list = service.getAll();

assertNotNull(list);

assertNotSame(0, list.size());

assertEquals("RideName", list.get(0).getrName());

} catch (Exception e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

/\*\*

\* Test method for

\* {@link com.virtual.queue.service.RideServiceImp#addRide(com.virtual.queue.beans.Ride)}

\* .

\*/

@Test

public void testAddRide() {

Ride ride = new Ride();

ride.setrName("RideName");

service.addRide(ride);

}

/\*\*

\* Test method for

\* {@link com.virtual.queue.service.RideServiceImp#updateRide(com.virtual.queue.beans.Ride)}

\* .

\*/

@Test

public void testUpdateRide() {

service.updateRide(ride);

}

/\*\*

\* Test method for

\* {@link com.virtual.queue.service.RideServiceImp#deleteRideById(java.lang.Long, java.lang.Long)}

\* .

\*/

@Test

public void testDeleteRideById() {

service.updateRide(ride);

}

/\*\*

\* Test method for

\* {@link com.virtual.queue.service.RideServiceImp#addRideById(java.lang.Long, java.lang.Long)}

\* .

\*/

@Test

public void testAddRideById() {

try {

//TODO:pending dependencies

//assertEquals(true, service.addUserRideById(1L, 1L));

} catch (Exception e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

/\*\*

\* Test method for

\* {@link com.virtual.queue.service.RideServiceImp#removeRidebyId(java.lang.String)}

\* .

\*/

@Test

public void testRemoveRidebyId() {

assertEquals(false, service.removeRidebyId("1"));

}

/\*\*

\* Test method for

\* {@link com.virtual.queue.service.RideServiceImp#pullRideInfo()}.

\*/

@Test

public void testPullRideInfo() {

List<RideInfo> list;

try {

list = service.pullRideInfo();

assertNotNull(list);

assertNotSame(0, list.size());

assertEquals("RideName", list.get(0).getrName());

} catch (NotificationException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}

package com.test.spring;

import static org.junit.Assert.\*;

import org.junit.After;

import org.junit.AfterClass;

import org.junit.Before;

import org.junit.BeforeClass;

import org.junit.Test;

public class RuleBuilderImpTest {

@BeforeClass

public static void setUpBeforeClass() throws Exception {

}

@AfterClass

public static void tearDownAfterClass() throws Exception {

}

@Before

public void setUp() throws Exception {

}

@After

public void tearDown() throws Exception {

}

@Test

public void test() {

}

}

Mockup for Ride Dao

package com.test.spring.mockups;

import java.util.ArrayList;

import java.util.List;

import com.virtual.queue.beans.RideInfo;

import com.virtual.queue.dao.RideDao;

import com.virtual.queue.exception.NotificationException;

public class RideDaoImpMock implements RideDao {

@Override

public List<RideInfo> pullRideInfo() throws NotificationException {

List<RideInfo> list = new ArrayList<RideInfo>();

RideInfo info1 = new RideInfo();

info1.setrName("RideName");

info1.setRideId(1);

list.add(info1);

return list;

}

@Override

public List<RideInfo> getRideByUser(Long userId)

throws NotificationException {

List<RideInfo> list = new ArrayList<RideInfo>();

RideInfo info1 = new RideInfo();

info1.setrName("RideName");

info1.setRideId(1);

list.add(info1);

return list;

}

@Override

public RideInfo getRideById(long rideId) throws NotificationException {

RideInfo info1 = new RideInfo();

info1.setrName("RideName");

info1.setRideId(1);

return info1;

}

@Override

public boolean addUserRideById(Long rideId, Long userId) {

// TODO Auto-generated method stub

return true;

}

@Override

public List<RideInfo> getAll() {

List<RideInfo> list = new ArrayList<RideInfo>();

RideInfo info1 = new RideInfo();

info1.setrName("RideName");

info1.setRideId(1);

list.add(info1);

return list;

}

}

Queue Scheduler stubs and Drivers:

package com.test.spring;

import static org.junit.Assert.\*;

import org.junit.After;

import org.junit.AfterClass;

import org.junit.Before;

import org.junit.BeforeClass;

import org.junit.Test;

public class QueueSchedulerTest {

@BeforeClass

public static void setUpBeforeClass() throws Exception {

}

@AfterClass

public static void tearDownAfterClass() throws Exception {

}

@Before

public void setUp() throws Exception {

}

@After

public void tearDown() throws Exception {

}

@Test

public final void testQueueScheduler() {

}

@Test

public final void testScheduleRideJobs() {

}

@Test

public final void testRemoveUserFromQueue() {

}

}

package com.test.spring;

import static org.junit.Assert.\*;

import org.junit.After;

import org.junit.AfterClass;

import org.junit.Before;

import org.junit.BeforeClass;

import org.junit.Test;

public class QueueFactoryTest {

@BeforeClass

public static void setUpBeforeClass() throws Exception {

}

@AfterClass

public static void tearDownAfterClass() throws Exception {

}

@Before

public void setUp() throws Exception {

}

@After

public void tearDown() throws Exception {

}

@Test

public final void testGetJobClassString() {

}

@Test

public final void testGetJobClassStringString() {

}

}

Mockup for Queue Scheduler DAO:

ackage com.test.spring.mockups;

import java.util.ArrayList;

import java.util.LinkedList;

import java.util.List;

import com.virtual.queue.beans.QueueInfo;

import com.virtual.queue.beans.RideInfo;

import com.virtual.queue.beans.User;

import com.virtual.queue.beans.UserQueueInfo;

import com.virtual.queue.dao.QueueDao;

public class QueueDaoImpMock implements QueueDao {

@Override

public List<UserQueueInfo> pullInfo(long rideId) {

if (rideId == 0)

return null;

List<UserQueueInfo> list = new ArrayList<UserQueueInfo>();

UserQueueInfo info = new UserQueueInfo();

info.setRideName("RideName");

list.add(info);

return list;

}

@Override

public List<UserQueueInfo> pullAllInfo() {

List<UserQueueInfo> list = new ArrayList<UserQueueInfo>();

UserQueueInfo info = new UserQueueInfo();

info.setRideName("RideName");

list.add(info);

return list;

}

@Override

public LinkedList<User> getAllUserQueueForRide(long rideId) {

if (rideId == 0)

return null;

LinkedList<User> list = new LinkedList<User>();

list.add(User.getDemoUser());

return list;

}

@Override

public QueueInfo getQueueInfoByRideId(long rideId) {

// TODO Auto-generated method stub

if (rideId == 0)

return null;

QueueInfo info = new QueueInfo();

info.setQueueId(1);

return info;

}

@Override

public boolean removeUserFromQueue(long rideId, long userid) {

if (rideId == 0 || userid == 0)

return false;

return true;

}

@Override

public boolean removeAllUsersFromQueue(long rideId) {

if (rideId == 0)

return false;

return true;

}

@Override

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

LinkedList<RideInfo> list = new LinkedList<RideInfo>();

RideInfo info = new RideInfo();

info.setrName("RideName");

list.add(info);

RideInfo info1 = new RideInfo();

info1.setrName("RideName");

list.add(info1);

return list;

}

@Override

public List<Long> getUserToRemoveFromQueue(Long rideId) {

List<Long> list = new ArrayList<Long>();

list.add(1L);

return list;

}

@Override

public LinkedList<Long> getAllUserQueueForRide(long rideId, int interval) {

LinkedList<Long> list = new LinkedList<Long>();

list.add(1L);

list.add(2L);

return list;

}

}

User Stubs and Drivers:

package com.test.spring;

import static org.junit.Assert.\*;

import org.junit.After;

import org.junit.AfterClass;

import org.junit.Before;

import org.junit.BeforeClass;

import org.junit.Test;

public class UserDaoImpTest {

@BeforeClass

public static void setUpBeforeClass() throws Exception {

}

@AfterClass

public static void tearDownAfterClass() throws Exception {

}

@Before

public void setUp() throws Exception {

}

@After

public void tearDown() throws Exception {

}

@Test

public void testGetUser() {

}

@Test

public void testGetCurrentlyAuthenticatedUserName() {

}

@Test

public void testGetUserByToken() {

}

@Test

public void testStoreToken() {

}

@Test

public void testGetAll() {

}

@Test

public void testAddUser() {

}

@Test

public void testUpdateUser() {

}

@Test

public void testDeleteUserById() {

}

@Test

public void testGetUserByUserName() {

}

@Test

public void testResetPassword() {

}

@Test

public void testAuthenticateUser() {

}

}

**package** com.test.spring;

**import** **static** org.junit.Assert.\*;

**import** java.util.List;

**import** org.junit.After;

**import** org.junit.AfterClass;

**import** org.junit.Before;

**import** org.junit.BeforeClass;

**import** org.junit.Test;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** com.test.spring.mockups.QueueDaoImpMock;

**import** com.test.spring.mockups.ResetPasswordDaoImpMock;

**import** com.test.spring.mockups.UserDaoImpMock;

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

**import** com.virtual.queue.dao.QueueDao;

**import** com.virtual.queue.dao.ResetPasswordDao;

**import** com.virtual.queue.dao.UserDao;

**import** com.virtual.queue.request.UserPasswordResetRequest;

**import** com.virtual.queue.service.UserService;

**import** com.virtual.queue.service.UserServiceImp;

**public** **class** UserServiceImpTest {

**private** UserDao userDao;

**private** ResetPasswordDao resetPassDao;

**private** QueueDao queueDao;

UserService service;

UserPasswordResetRequest passReset;

@BeforeClass

**public** **static** **void** setUpBeforeClass() **throws** Exception {

}

@AfterClass

**public** **static** **void** tearDownAfterClass() **throws** Exception {

}

@Before

**public** **void** setUp() **throws** Exception {

userDao = **new** UserDaoImpMock();

queueDao = **new** QueueDaoImpMock();

resetPassDao = **new** ResetPasswordDaoImpMock();

service = **new** UserServiceImp(userDao, queueDao, resetPassDao);

passReset=**new** UserPasswordResetRequest();

passReset.setConfirmNewPassword("confirmation");

passReset.setNewPassword("newPassword");

passReset.setSecurityAnswer("securityQ");

passReset.setSecurityQuestion("securityQuestion");

passReset.setUserName("username");

}

@After

**public** **void** tearDown() **throws** Exception {

}

@Test

**public** **void** testGetAll() {

List<User> list=service.getAll();

assertNotNull(list);

assertNotSame(0, list.size());

}

@Test

**public** **void** testAddUser() {

service.addUser(User.getDemoUser());

}

@Test

**public** **void** testUpdateUser() {

service.updateUser(User.getDemoUser());

}

@Test

**public** **void** testDeleteUserById() {

service.deleteUserById(1L);

}

@Test

**public** **void** testResetPassword() {

**try** {

service.resetPassword(passReset);

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

Mockup for DAO:

package com.test.spring.mockups;

import java.sql.SQLException;

import java.util.ArrayList;

import java.util.List;

import com.virtual.queue.beans.User;

import com.virtual.queue.dao.UserDao;

public class UserDaoImpMock implements UserDao {

@Override

public User getUser(String username, String passwd) {

User testUser = new User();

testUser.setUserid(1L);

testUser.setFirstName("Ian");

testUser.setLastName("Smith");

return testUser;

}

@Override

public String getCurrentlyAuthenticatedUserName() {

return "username";

}

@Override

public User getUserByToken(String token) {

User testUser = new User();

testUser.setUserid(1L);

testUser.setFirstName("Ian");

testUser.setLastName("Smith");

testUser.setToken(token);

return testUser;

}

@Override

public String storeToken(Long userId) {

if(userId ==null ||userId ==0)return null;

return "token";

}

@Override

public List<User> getAll() {

List<User> list = new ArrayList<User>();

User testUser = new User();

testUser.setUserid(1L);

testUser.setFirstName("Ian");

testUser.setLastName("Smith");

list.add(testUser);

User testUser1 = new User();

testUser.setUserid(1L);

testUser.setFirstName("Juan");

testUser.setLastName("Caca");

list.add(testUser1);

return list;

}

@Override

public void addUser(User user) throws SQLException {

System.out.println("User added");

}

@Override

public void updateUser(User user) {

System.out.println("User Updated");

}

@Override

public void deleteUserById(Long id) {

System.out.println("User deleted");

}

@Override

public User getUserByUserName(String userName) {

if (userName == null)

return null;

User user = new User();

user.setFirstName("FirstName");

return user;

}

@Override

public Boolean resetPassword(String userName, String securityAnswer,

String securityQuestion, String newPassword) throws Exception {

if (userName == null || securityAnswer == null || newPassword == null)

return null;

// TODO Auto-generated method stub

return true;

}

@Override

public User authenticateUser(String userName, String securityQuestion,

String securityAnwser) {

if (userName == null || securityQuestion == null

|| securityAnwser == null)

return null;

User user = new User();

user.setFirstName("FirstName");

return user;

}

@Override

public User getUserById(long userId) {

if (userId > 0)

return null;

User user = new User();

user.setFirstName("FirstName");

return user;

}

@Override

public boolean removeUserFromQueue(long userId, long rideId) {

return true;

}

@Override

public User getUserToModify(long userId) {

if (userId > 0)

return null;

User user = new User();

user.setFirstName("FirstName");

return user;

}

@Override

public Boolean editUserById(String newFirstName, String newLastName,

String newEmail, String password, String newUserName,

String securityAnswer, String securityQuestion, String newCell,

String newAge, String newCode) throws Exception {

if (password == null)

return false;

return true;

}

@Override

public boolean AddRole(long userId, String roleType) throws SQLException {

if (userId == 0)

return false;

return true;

}

@Override

public boolean enabledDisabledUser(long userId, String flag) {

if (userId == 0 || flag == null)

return false;

return true;

}

}

import resources.DynamicFindWaitTimeAsAdminHelper;

import com.rational.test.ft.\*;

import com.rational.test.ft.object.interfaces.\*;

import com.rational.test.ft.object.interfaces.SAP.\*;

import com.rational.test.ft.object.interfaces.WPF.\*;

import com.rational.test.ft.object.interfaces.dojo.\*;

import com.rational.test.ft.object.interfaces.siebel.\*;

import com.rational.test.ft.object.interfaces.flex.\*;

import com.rational.test.ft.object.interfaces.generichtmlsubdomain.\*;

import com.rational.test.ft.script.\*;

import com.rational.test.ft.value.\*;

import com.rational.test.ft.vp.\*;

import com.ibm.rational.test.ft.object.interfaces.sapwebportal.\*;

/\*\*

\* Description : Functional Test Script

\* @author Kenneth

\*/

public class DynamicFindWaitTimeAsAdmin extends DynamicFindWaitTimeAsAdminHelper

{

/\*\*

\* Script Name : <b>DynamicFindWaitTimeAsAdmin</b>

\* Generated : <b>Apr 1, 2015 4:19:24 PM</b>

\* Description : Functional Test Script

\* Original Host : WinNT Version 6.3 Build 9600 ()

\*

\* @since 2015/04/01

\* @author Kenneth

\*/

public void testMain(Object[] args)

{

// HTML Browser

// Document: Virtual Queue: http://localhost:8080/Virtual\_Queue/

link\_admin().click();

link\_simulateRides().click();

button\_buttonrefresh().click();

button\_cancelbutton().click();

link\_myAccount().click();

button\_buttonrefresh2().click();

link\_accountRides().click();

link\_add().click();

//

button\_htmlDialogButtonOK().click();

// HTML Browser

// Document: Virtual Queue: http://localhost:8080/Virtual\_Queue/

button\_buttonrefresh2().click();

button\_cancelbutton2().click();

link\_admin().click();

link\_simulateRides().click();

button\_buttonrefresh().click();

form\_adminForm().drag(atPoint(271,617), atPoint(675,604));

link\_dequeueRecords().click();

//

button\_htmlDialogButtonOK().click();

// HTML Browser

// Document: Virtual Queue: http://localhost:8080/Virtual\_Queue/

form\_adminForm().drag(atPoint(692,612), atPoint(317,610));

button\_buttonrefresh().click();

button\_cancelbutton().click();

link\_myAccount().click();

button\_buttonrefresh2().click();

}

}

import resources.DynamicFindWaitTimeAsAdminVer2Helper;

import com.rational.test.ft.\*;

import com.rational.test.ft.object.interfaces.\*;

import com.rational.test.ft.object.interfaces.SAP.\*;

import com.rational.test.ft.object.interfaces.WPF.\*;

import com.rational.test.ft.object.interfaces.dojo.\*;

import com.rational.test.ft.object.interfaces.siebel.\*;

import com.rational.test.ft.object.interfaces.flex.\*;

import com.rational.test.ft.object.interfaces.generichtmlsubdomain.\*;

import com.rational.test.ft.script.\*;

import com.rational.test.ft.value.\*;

import com.rational.test.ft.vp.\*;

import com.ibm.rational.test.ft.object.interfaces.sapwebportal.\*;

/\*\*

\* Description : Functional Test Script

\* @author Kenneth

\*/

public class DynamicFindWaitTimeAsAdminVer2 extends DynamicFindWaitTimeAsAdminVer2Helper

{

/\*\*

\* Script Name : <b>DynamicFindWaitTimeAsAdminVer2</b>

\* Generated : <b>Apr 1, 2015 4:32:56 PM</b>

\* Description : Functional Test Script

\* Original Host : WinNT Version 6.3 Build 9600 ()

\*

\* @since 2015/04/01

\* @author Kenneth

\*/

public void testMain(Object[] args)

{

// HTML Browser

// Document: Virtual Queue: http://localhost:8080/Virtual\_Queue/

image\_universal().click();

link\_admin().click();

link\_login().click();

text\_userName().click(atPoint(122,23));

browser\_htmlBrowser(document\_virtualQueue(),DEFAULT\_FLAGS).inputChars("mlazo007@fiu.edu");

text\_password().click(atPoint(134,19));

browser\_htmlBrowser(document\_virtualQueue(),DEFAULT\_FLAGS).inputChars("321");

button\_loginsubmit().click();

html\_search().drag(atPoint(1907,211), atPoint(1919,484));

button\_cancelbutton().click();

link\_myAccount().click();

button\_buttonrefresh().click();

link\_accountRides().click();

link\_add().click();

//

button\_htmlDialogButtonOK().click();

// HTML Browser

// Document: Virtual Queue: http://localhost:8080/Virtual\_Queue/

button\_buttonrefresh().click();

button\_cancelbutton2().click();

link\_admin().click();

link\_simulateRides().click();

button\_buttonrefresh2().click();

form\_adminForm().drag(atPoint(406,618), atPoint(773,624));

link\_dequeueRecords().click();

//

button\_htmlDialogButtonOK().click();

// HTML Browser

// Document: Virtual Queue: http://localhost:8080/Virtual\_Queue/

form\_adminForm().drag(atPoint(601,613), atPoint(156,586));

button\_buttonrefresh2().click();

button\_cancelbutton3().click();

link\_myAccount().click();

button\_buttonrefresh().click();

}

}

import resources.SimulateAddRides01Helper;

import com.rational.test.ft.\*;

import com.rational.test.ft.object.interfaces.\*;

import com.rational.test.ft.object.interfaces.SAP.\*;

import com.rational.test.ft.object.interfaces.WPF.\*;

import com.rational.test.ft.object.interfaces.dojo.\*;

import com.rational.test.ft.object.interfaces.siebel.\*;

import com.rational.test.ft.object.interfaces.flex.\*;

import com.rational.test.ft.object.interfaces.generichtmlsubdomain.\*;

import com.rational.test.ft.script.\*;

import com.rational.test.ft.value.\*;

import com.rational.test.ft.vp.\*;

import com.ibm.rational.test.ft.object.interfaces.sapwebportal.\*;

/\*\*

\* Description : Functional Test Script

\* @author Kenneth

\*/

public class SimulateAddRides01 extends SimulateAddRides01Helper

{

/\*\*

\* Script Name : <b>SimulateAddRides01</b>

\* Generated : <b>Mar 21, 2015 4:32:47 PM</b>

\* Description : Functional Test Script

\* Original Host : WinNT Version 6.3 Build 9600 ()

\*

\* @since 2015/03/21

\* @author Kenneth

\*/

public void testMain(Object[] args)

{

// HTML Browser

// Document: Virtual Queue: http://localhost:8080/Virtual\_Queue/

link\_admin().click();

link\_simulateRides().click();

form\_adminForm().drag(atPoint(371,507), atPoint(663,528));

link\_dequeuerecords().click();

button\_buttonrefresh().click();

button\_buttonrefresh().click();

button\_cancelbutton().click();

}

}

/\*\*

\* Description : Functional Test Script

\* @author Michael

\*/

public class SimulateDequeueRides01 extends SimulateDequeueRides01Helper

{

/\*\*

\* Script Name : <b>SimulateAddRides01</b>

\* Generated : <b>Mar 21, 2015 4:32:47 PM</b>

\* Description : Functional Test Script

\* Original Host : WinNT Version 6.3 Build 9600 ()

\*

\* @since 2015/03/21

\* @author Lazo

\*/

public void testMain(Object[] args)

{

// HTML Browser

// Document: Virtual Queue: http://localhost:8080/Virtual\_Queue/

link\_admin().click();

link\_simulateRides().click();

form\_adminForm().drag(atPoint(371,507), atPoint(663,528));

link\_dequeuerecords().click();

button\_buttonrefresh().click();

button\_buttonrefresh().click();

button\_cancelbutton().click();

}

}

* 1. Appendix H – Diary of meeting and tasks for the **entire semester**.

**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.

**Meeting  9**

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

**Date:** 3/17/2015

**Members Participated:**

Michael Lazo

Kenneth Kon - Scrum master

Bernard Parenteau - Product Owner

**Topic:**

Talked about the progress of sprint 3. Clarified on Simulate Add Ride user story.

**Meeting  10**

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

**Date:** 3/22/2015

**Members Participated:**

Michael Lazo

Kenneth Kon - Scrum master

Bernard Parenteau - Product Owner

**Topic:**

Sprint 3 Review, showcased User Story Simulate Add Ride, Simulate Dequeue and Visitor Dequeue user story. Discussed what we would need to work on for Sprint 4.

**Meeting  11**

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

**Date:** 3/24/2015

**Members Participated:**

Michael Lazo - Scrum master

Kenneth Kon

Bernard Parenteau - Product Owner

**Topic:**

Sprint 4 requirement elicitation on the user stories assigned to us.

Mostly on Dynamic Find wait time and Edit Rides as Admin

**Meeting  12**

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

**Date:** 3/29/2015

**Members Participated:**

Michael Lazo

Kenneth Kon - Scrum master

Bernard Parenteau - Product Owner

**Topic:**

Sprint 4 review, both of our user stories were accepted. After that Product owner assigned us sprint 5 user stories.

**Meeting  13**

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

**Date:** 4/2/2015

**Members Participated:**

Michael Lazo - Scrum master

Kenneth Kon

Bernard Parenteau - Product Owner

**Topic:**

Sprint 5 requirement elicitation on the user stories assigned to us.

**Meeting  14**

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

**Date:** 4/10/2015

**Members Participated:**

Michael Lazo - Scrum master

Kenneth Kon

Bernard Parenteau - Product Owner

**Topic:**

Sprint 5 reviews, both of our user stories were accepted.

1. 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