Advanced Software Engineering CEN 5011U01

Project Team Members:

Steve Foo

Juan Sotomayor

Maral Kargarmoakhar

Freny Patel

Professor:

Dr. Peter J. Clarke

04/29/2016

ABSTRACT

QicFix is an application to help people to find a tower to tow their car for a very low cost and in a very short period. Currently some insurance companies are offering a similar service such as AAA. AAA is a membership only car/travel company. All members have the benefit of getting their car towed anywhere within 25 miles for free. Clients who are not part of AAA have to rely on calling towing companies they find over the internet and negotiating a tow from them. The clients can be charge an unreasonable rate because they are at the mercy of towing company. There is a lack of an application that fills this gap. What our system does is combine both the convenience and variety of tow truck drivers available into one application.

This document is a written document for the software application called "QicFix". This document briefly describes about the application with a Software Engineering perspective. The documents hold the blueprint for the application. The document contains all the necessary items required for the software to run as well as it also has the diagrammatic representation of each feature in the application. The document contains all the necessary information required to understand the system as well as the technical jargons associated with the application or the system. It also briefly summarizes the estimation of cost and effort required for the software to develop. The document also includes a brief summary of the developer's meetings and all the discussions that went into making this application. It begins by describing the problem, the scope of this project, and a guide for the rest of the document. USDP (Unified Software Development Process) was used for processing this project.

With this application the users will be able to get fast a low cost towing service that will be rated by the same users making it more reliable to the new users. The application can be downloaded for free and our benefits from a small percentage per transaction.

Contents

1. Introduction: 5
   1.1. Purpose of system: 5
   1.2. Scope of the System: 5
   1.3. Development Methodology: 6
   1.4. Definitions, Acronyms, and Abbreviations: 7
   1.5. Overview of document 8
2. Current System 9
3. Project Plan: 10
   3.1. Project Organization: 10
   3.2. Hardware & Software Requirements: 10
   3.3. Work Break Down Structure: 12
4. Requirements of System: 13
   4.1. Functional and Nonfunctional Requirements : 13
   4.2. Use case diagram: 16
   4.3. Requirements Analysis: 20
5. Proposed Software Architecture 21
   5.1. Overview: 21
   5.2. Subsystem Decomposition: 23
   5.3. Hardware and Software Mapping: 27
   5.4. Persistent Data Management: 29
   5.5. Security Management: 30
6. Detailed Design: 31
   6.1. Overview 31
   6.2. State machine: 45
   6.3. Object Interaction: 49
   6.4. Detailed Class Design: 73
7. Glossary: 80
8. Signatures: 82
9. References: 83
10. Appendix: 84

10.1. Appendix A – Project Schedule: 84
10.2. Appendix B – Use Cases: 85
10.3. Appendix C – User Interface Designs: 116
10.4. 11.4. Appendix D – Detailed Class Diagrams: 132
10.5. Appendix E – Class interfaces 141
10.6. Appendix G – Diary of meeting and tasks for the entire semester 141

1. Introduction:

In this chapter, we introduce the main convincing arguments and informative pieces about our mobile system, QicFix. It is important to know the current problem definition of the old system because otherwise one would not understand why we are making this new system. No implementable system can be infinite, therefore in this chapter we also define the scope of our system and what can be accomplished without compromising too many of the original goals. This chapter will also serve as an invaluable guide to the rest of the document and terms used throughout.

1.1. Purpose of system:

The current system of getting a car repaired or towed revolves around getting referrals from others or going into the yellow pages. This system has been somewhat effective over the years, but still can be improved upon. One of the flaws of the current system is that many people are going to these servicemen based off a referral from their friends or family. The experience a friend or family member receives from servicemen can vary greatly and what seemed satisfactory to them may be unsatisfactory to you. Another flaw of current system is the speed at which you can contact a serviceman. Just to get in contact with a serviceman and arrange an appointment could involve waiting a couple hours before they arrive.

QicFix is an “Uber” like application but for people looking for tow truck drivers. Every Client needs to have credit card in order to register. The system allows for a Client put their location and destination they want to get towed. Local Towers sees this service request and can choose to accept it or not. Once a Tower accepts the service request they are given the Client’s information to contact them. Once the Client is brought to their destination their card is charged. We believe that our application cuts down on the time it takes for a client to get their car towed.

1.2. Scope of the System:

The scope of the system is to provide a quicker and more convenient way for users to interact with servicemen. The system will allow for users to pick the closest serviceman around them based off their GPS location. It will also allow them to pick the best serviceman based off reviews other clients have left for that serviceman. Lastly, it will allow them to pick the cheapest serviceman based off other servicemen’s prices. Phone notifications will be used to keep a constant stream of

contact between the client and serviceman. Serviceman can be a person to tow or a mechanic who can help to repair the car.

1.3. Development Methodology:

In software engineering, a software development methodology (also known as a system development methodology, software development process, software process) is a splitting of software development work into distinct phases containing activities with the intent of better planning and management.

For our project, we are using USDP (Unified Software Development Process), which has five steps. In the first step we had Analysis model which was introduced completely in first deliverable, in this step we have Design model and deployment model and in the next steps we are going to have Implementation model and Test model.

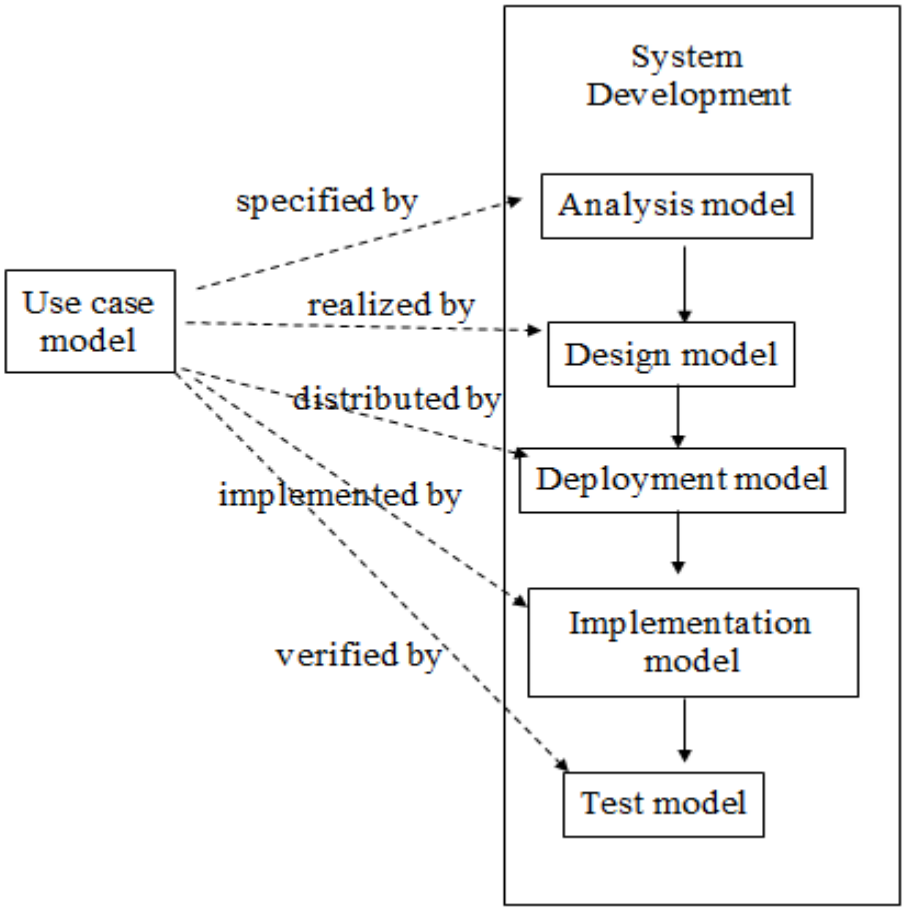


Figure 1.3.USDP Model

1.4. Definitions, Acronyms, and Abbreviations:

- **Tower** - a person who drives a tow truck
- **GPS** - Global Positioning system
- **Android** - a mobile operating system based on the Linux Kernel and currently developed by Google.
- **API** - Application Program Interface
- **Request** - An instruction to a computer to provide information or perform another function.
- **Server** - a computer program or a device that provides functionality for other programs or devices, called "clients".

- **Timeout -** a cancellation or cessation that automatically occurs when a predefined interval of time has passed without a certain event occurring.
- **USDP -** Unified Software Development Process
- **Cocomo** - Constructive Cost Model is a algorithmic software cost estimation model

1.5. Overview of document

The remainder of this document describes the requirements of the QicFix mobile system. Chapter 2 discusses Software Architectural Patterns, System Decomposition, Hardware and Software Mapping and also we discuss about Security Management and describe the security protocols we use in the design. Chapter 3 we have State Machine for the overall system, discuss about object interactions and also we have Detailed Class Design. Chapter 4 contains a glossary of terms used throughout the paper. Finally, Chapter 6 is an Appendix that contains all use cases, diagrams, and images mentioned throughout this paper.

2. Current System

AAA is a company which provides similar service but required a yearly membership and upfront fee. QicFix does not require yearly membership fee and it just need registration. Clients who are not member of AAA have to rely on calling towing companies they find over the internet and negotiating a tow from them. The clients can be charged an unreasonable rate because they are at the mercy of towing company. There is a lack of an application that will fill this gap. What our system will do is combine both the convenience and variety of tow truck drivers available into one application which is cheaper and more accessible.

3. Project Plan:

Project Plan is defined as the application of skills, knowledge, tools and methodology for the project activities to meet the requirements. This is accomplished through the application and integration of the project management process they are planning, execution, monitoring, controlling. Software Engineering is also a managed process. In this chapter we will define how the team is organized and the Hardware and Software required for the implementing the system. Also this chapter will serve as the detailed information about the duration of the project based on its work breakdown structure. Also it will guide to the cost estimation for the Project using the COCOMO II model.

3.1. Project Organization:

This are the Roles which we assigned during the Software Development Life Cycle for the QicFix System.

	Team Leader	Tester	Minute Taker	Time Keeper	Developer	Requirement Engineer
Deliverable 1	**Steve**		**Juan**	**Maral**		**Freny**
Deliverable 2	**Freny**	**Freny, Juan**	**Maral**	**Juan**		**Steve,Juan**
Deliverable 3	**Juan**	**Freny, Maral**	**Freny**	**Steve**	**Steve,Juan**	**Freny, Steve**

3.2. Hardware & Software Requirements:

The team has identified the next development environment requirements.

3.2.1 Hardware Requirements:

- **Laptop PC ASUS 17”**
  - CPU: Intel Core i7 - 2630QM CPU 2 Ghz
  - RAM: 8GB
  - Processor type: 64 bit
- **Mid 2013 Apple Macbook Air 13”**
  - Operating System: Mac OS X-Yosemite

- Processor: 2.5 GHz Intel Core i5
- Memory: 4 GB 1600 MHz DDR3
- Graphics: Intel HD Graphics 4000 1024 MB

- **HP Notebook 15”**
  - Operating System: Windows 10
  - Processor: Intel(R) Core(™ i3-5010U CPU)2.10GHz
  - Memory: 6GB • Lenovo Yoga 13.3”
  - Operating System: Windows 10 Home
  - Processor: Intel Core i7-6500U
  - Memory: 16 GB DDR3L
  - Graphics: Integrated Intel® HD Graphics
- **Virtual Server Configuration**
  - PC Type: Virtual Machine
  - vCPU: 1, 2.5 GHz o RAM: 1GB
  - HDD: 8GB
- **Mobile Device Hardware**
  - Brand: Huawei o Model: Nexus 6P
  - Screen size: 5.7 inch screen
  - Feature: 4G LTE, Unlocked
  - Storage capacity: 32 GB
  - USB cable type: C

3.2.2 Software Requirements:

- **Development PC Software**
  - Host PC OS: Windows 10 Home 64 bit
  - Virtual Machine Software: Oracle VM VirtualBox
  - Interface Development Env: Eclipse 4.5 (Mars) with Papyrus plugin
  - Mobile IDE: Android Studio 1.5.1 o Java: JDK/JRE 8
  - Web Server / Web services: Apache 2 Tomcat 8
  - Web Site: Java/ HTML /Javascript /CSS
  - Firewall configurations: deactivated
- **Evolus Pencil**
- **Virtual Server Software**
  - Virtual Machine OS: Centos 7 64 bit
  - Java: JDK/JRE 8
  - Web Server / Web services: Apache 2 Tomcat 8
  - Web Site: Java/ HTML /Javascript /CSS
  - Firewall configurations: deactivated
- **Android Studio**
- **Google Drive**

- **Google Hangouts**
- **Mobile Device Software**
    - Operating system: Android 4.0 or later
    - Developer Options: activated o USB debugging: activated

3.3. Work Break Down Structure:

The total duration for this project is 65 days until implementation and 71 days after final tests are completed. See Appendix 10.1

4. Requirements of System:

Purpose of this application is to find a local towing company which can help you in an emergency situation if you car has an issue or has mechanical problem and you get stuck in the middle of your trip. Sometimes you travel to some places that you are not familiar with the area and if something happen with your car or truck, you need a tow person to help you and take your vehicle to a mechanic shop. Imagine in a snowy day in New Jersey you had trouble hailing a mechanic or a tow person. So you came up with a simple idea tap a button, get a help and tow your car to a close and open mechanic, QicFix can save your money, time and sometimes your life if you get stuck in a bad weather situation. QicFix use a technology to give people or travelers what they need in a car accident, mechanical car issue and also when they want these services.

For the women and men who has towing car which is called tow person, our application represents a flexible new way to earn money and help them to find their clients faster. For cities and especially urban areas QicFix improve access to local towing companies and make also make safety for travelers. When you make easy and reliable access to what you need as running water, everyone benefits. Especially when it's snowing outside or someone needs help in a bad weather situation.

QicFix is an on-demand service which can bring a revolution in our daily life and can be used all across the world. The business model of QicFix has made it possible for people to simple tap their smartphone and have a towing vehicle at their location in the minimum possible time to help them with their car issues.

Understanding of how the system should work: Users who want to use QicFix needs to register and make an account, importing their information and payment information in order to use the application. On the other side, the towing companies or personal tow persons need to register and make an account. QicFix is equipped with GPS to find the closest tow person to the client. Once the user login to QicFix with detecting his/her location, closest tow person can be seen on the application.

4.1. Functional and Nonfunctional Requirements :

Use Case ID	Description	Non Functional Req.	Appendix B	Appendix C
QicFix-C-104	The system shall allow the client to see previous ratings and comments of Tower	• **Usability** - less than 1 minute to perform • **Reliability** - 1 failure out of 300 executions	**11.2.12**	
QicFix-C-101	The system shall allow the client to pick up to 3 different towers	• **Usability** - less than 1 minute to perform • **Reliability** - 1 failure out of 300 executions	**11.2.10**	**11.3.5** **11.3.17**

		• **Performance -** takes less than 1 second to complete		
QicFix-C-103	The system shall allow the client to rate and comment the Tower	• **Usability -** less than 2 minute to perform • **Reliability -** 1 failure out of 300 executions • **Performance -** takes less than 2 second to complete	**11.2.13**	
QicFix-C-102	The system shall allow the client to pay a Towing fee for a Tower	• **Usability -** less than 1 minute to perform • **Reliability -** 1 failure out of 300 executions • **Performance -** takes less than 2 second to complete	**11.2.11**	
QicFix-T-201	The system shall allow the Tower to update his/her profile.	• **Usability -** less than 1 minute to perform • **Performance -** takes less than 1 second to complete	**11.2.1**	**11.3.3** **11.3.4** **11.3.11** **11.13.12**
QicFix-T-202	The system shall allow the Tower to see their own rating and client's comments	• **Usability -** less than 1 minute to perform • **Performance -** takes less than 2 second to complete	**11.2.2**	
QicFix-T-203	The system shall allow the Tower to accept a job.	• **Usability -** less than 1 minute to perform • **Reliability -** 1 failure every 24 hours • **Performance -** takes less than 5 second to complete	**11.2.3**	**11.3.6** **11.3.16**
QicFix-T-205	The system shall allow the Tower to decline a job.	• **Usability -** less than 1 minute to perform • **Reliability -** 1 failure every 72 hours • **Performance -** takes less than 5 second to complete	**11.2.5**	**11.3.6** **11.3.16**
QicFix-T-204	The system shall allow a Tower to charge a client	• **Usability -** less than 1 minute to perform • **Performance -** takes less than 2 second to complete	**11.2.4**	

QicFix-CT-105	The system shall allow the User to have up to 5 tries to login	• **Usability -** less than 1 minute to perform • **Reliability -** 1 failure out of 300 executions • **Performance -** takes less than 1 second to complete	**11.2.14**	**11.3.2** **11.3.10**
QicFix-T-207	The system shall timeout the User from the system if they are inactive too long.	• **Usability -** less than 1 minute to perform	**11.2.6**	
QicFix-C-301	The system shall allow the client to modify their current location for a Tow	• **Usability -** less than 1 minute to perform • **Reliability -** 1 failure every 72 hours • **Performance -** takes less than 1 second to complete	**11.2.7**	**11.3.17**
QicFix-C-305	The system shall allow the client to modify their destination for a Tow	• **Usability -** less than 1 minute to perform • **Reliability -** 1 failure every 72 hours • **Performance -** takes less than 1 second to complete	**11.2.8**	**11.3.17**
QicFix-C-310	The system shall allow the client to see an ordered list of Best Rated Towers	• **Usability -** less than 1 minute to perform • **Reliability -** 1 failure every 72 hours • **Performance -** takes less than 2 second to complete	**11.2.9**	
QicFix-CT-401	The system shall allow the client/tower to login	• **Usability -** less than 1 minute to perform • **Reliability -** 1 failure every 24 hours • **Performance -** takes less than 5 second to complete	**11.2.17**	**11.3.2** **11.3.10**
QicFix-C-402	The system shall allow the client to register	• **Usability -** less than 1 minute to perform • **Reliability -** 1 failure every 24 hours • **Performance -** takes less than 5 second to complete	**11.2.15**	**11.3.1** **11.3.9**

QicFix-C-403	The system shall allow the tower to register and make account	• **Usability** - less than 1 minute to perform • **Reliability** - 1 failure every 24 hours • **Performance** - takes less than 5 second to complete	**11.2.16**	**11.3.1** **11.3.8**
QicFix-C-404	The system shall allow the client/tower to logout	• **Usability** - less than 1 minute to perform • **Reliability** - 1 failure every 24 hours • **Performance** - takes less than 5 second to complete	**11.2.18**	**11.3.7**

4.2. Use case diagram:

This use case diagram shows an overview of our system's use cases and actors. The external actors of our system are the client and tower which inherit from the user. The clients are regular users who will be requesting service from the tower actors. The user's use cases are held within the common pkg, the client's use cases are held within the client pkg and the tower's use cases are held within the tower pkg. The google maps actor interacts with the client and tower pkg by providing responses to requests made with the following use cases: QicFix-C-101, QicFix-C-301, QicFix-C-305, QicFix-C-310, QicFix-T-203, and QicFix-T-205

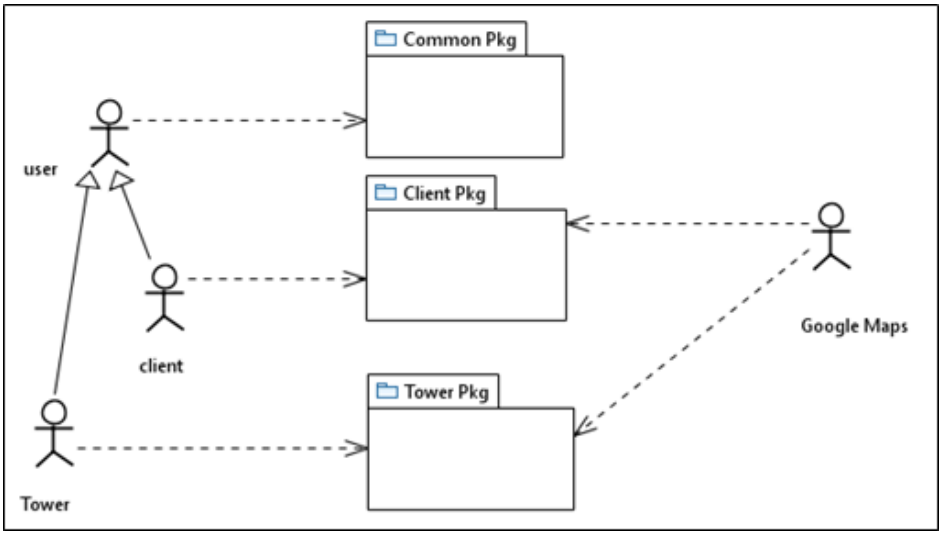


Figure 4.2.1 Reduced All Use Case Diagram

In the common package use case diagram the user actor has the ability to perform all the use cases listed here in the QicFix system. As stated previously these use cases are inherited by the client and tower actors.

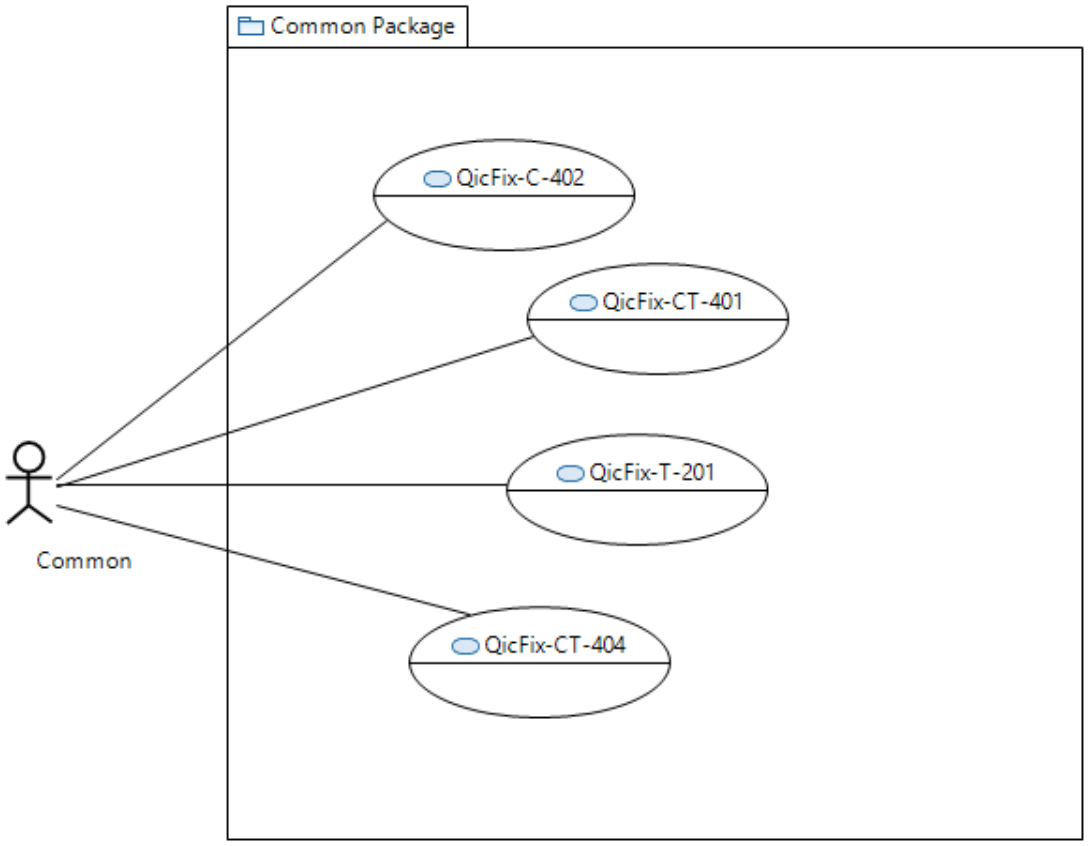


Figure 4.2.2 Common User Package Use Case Diagram

In the tower package the tower actor has the ability to perform all the use cases listed here in the QicFix system.

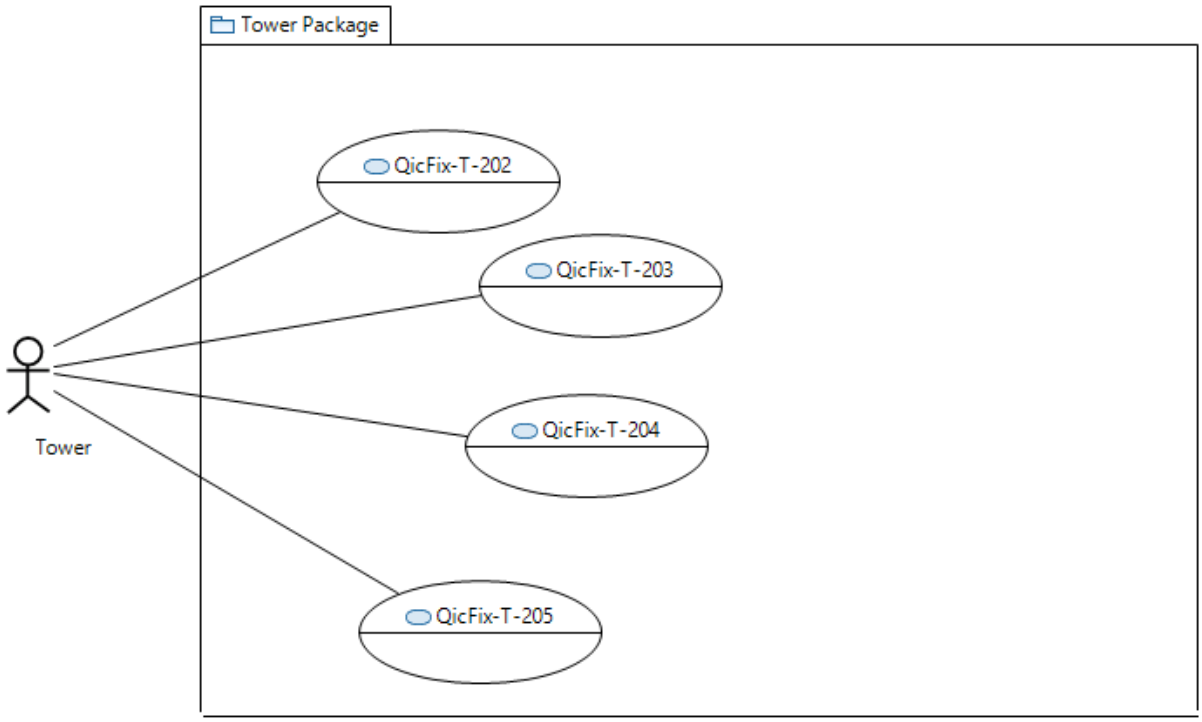


Figure 4.2.3.Tower Package Use Case Diagram

In the client package the client actor has the ability to perform all the use cases listed here in the QicFix system except for QicFix-C-315.

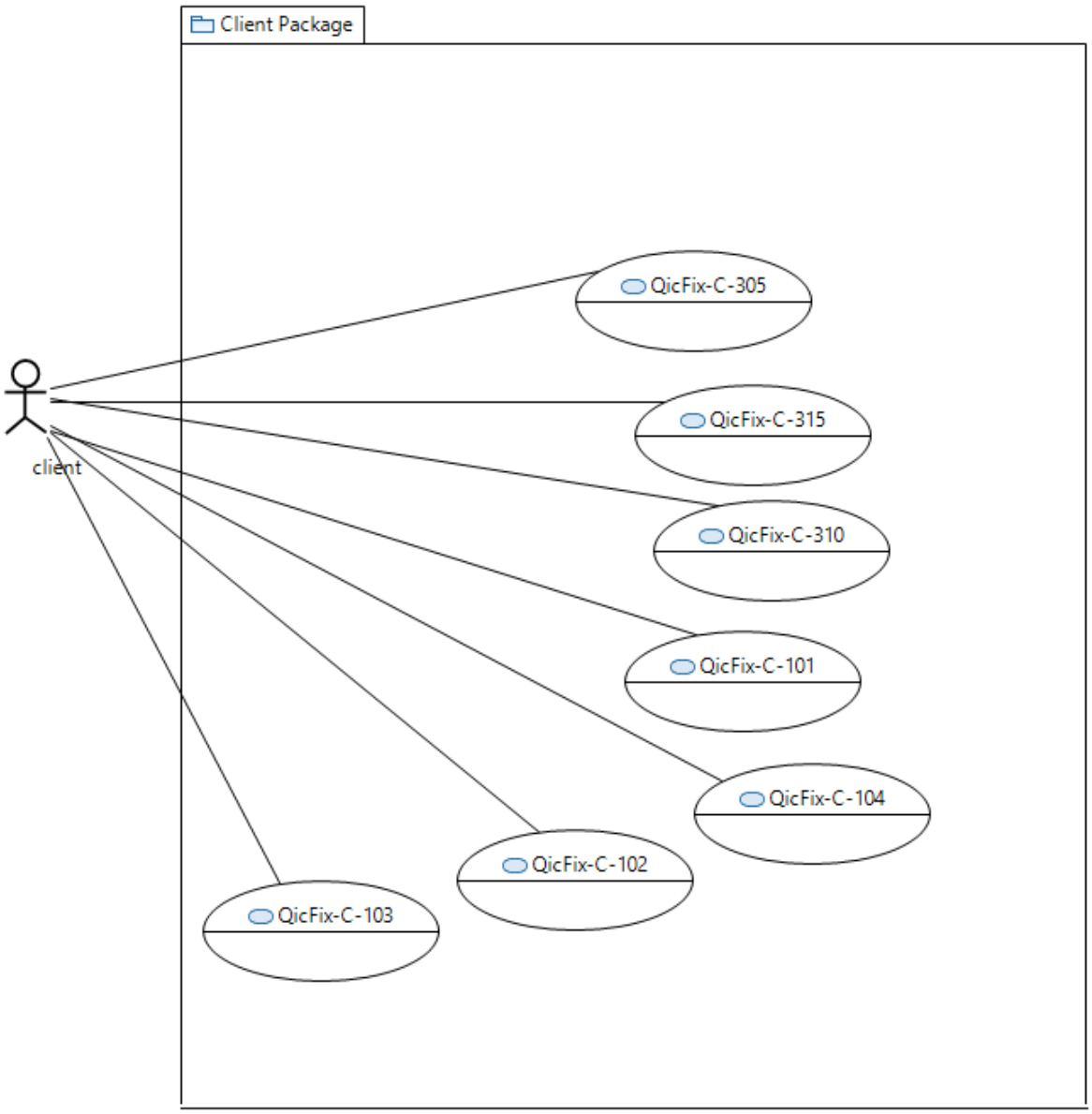


Figure 4.2.4 Client Package Use Case Diagram

4.3. Requirements Analysis:

After gathering the functional and non-functional requirements, we performed requirements analysis in the form of creating scenarios. Each scenario represented an instance for our use cases. From the scenarios, we were able to derive object diagrams. Iteratively, we were then able to create a class diagrams. Variables created in a class diagram corresponded to the variables of their object diagram counterparts. As part of creating the class diagrams, we had to identify boundary, control, and entity objects. The object diagrams and the class diagrams formed our Object Diagram Model.

Once we had our object diagram, we were able to further analyze our requirements in the form of a dynamic model. The dynamic model was comprised of sequence diagrams which modeled the interactions between actors and the objects in the system. The life lines created in the sequence diagrams came from the class diagrams. The methods were added to the class diagrams when links between lifelines in the the sequence diagrams were being established.

5. Proposed Software Architecture

This chapter describes the proposed overall system design for the QicFix system. An architectural overview of the system is provided, displaying the architectural patterns that were chosen along with all of the major subsystems. Choosing the proper architectural patterns is essential to the success of the system as we must ensure adherence to the non-functional requirements.

Once the appropriate architectural patterns are proposed and reasoned upon, the chapter presents a subsystem decomposition for all major subsystems. Subsystems are defined and further decomposed into several packages in order to lay the foundation for promoting low coupling and high cohesion throughout the system design and subsequently, the detailed object design. In addition to system architectural design, a deployment model is presented detailing the mapping between the software and all necessary hardware for the execution of the QicFix system.

5.1. Overview:

In order to comply with supportability requirements, the QicFix system needs to be able to service multiple device types, including mobile and web devices. While presenting to multiple device types, we must also adhere to usability constraints. As a system expands to multiple platforms, it is highly prone to usability issues and inconsistencies in behavior. In addition, the performance of the system must yield very low response times. On average, most of the implemented use cases (see Appendix A) require response times of less than 5 seconds. In order to meet such performance thresholds, it is important to physically separate some of the system components. Due to these supportability, performance, and usability constraints, the chosen primary architecture is a Four-Tier Architecture with the following subsystems (see Figure 5.1.1):

- **Client –** Represents all of the subsystems necessary to provide unique views to different devices. This subsystem will be servicing both mobile and web devices.
- **Presentation –** Responsible for formatting view models for each specific client.
- **Application Logic –** Responsible for handling all application logic. Provides an interface for all necessary create/read/update/delete operations across the application. Handles enforcement of role-based security and other logic such as required field validations.
- **Data –** Responsible for providing a set of models and a means of communicating with a third party database management system (MySQL).

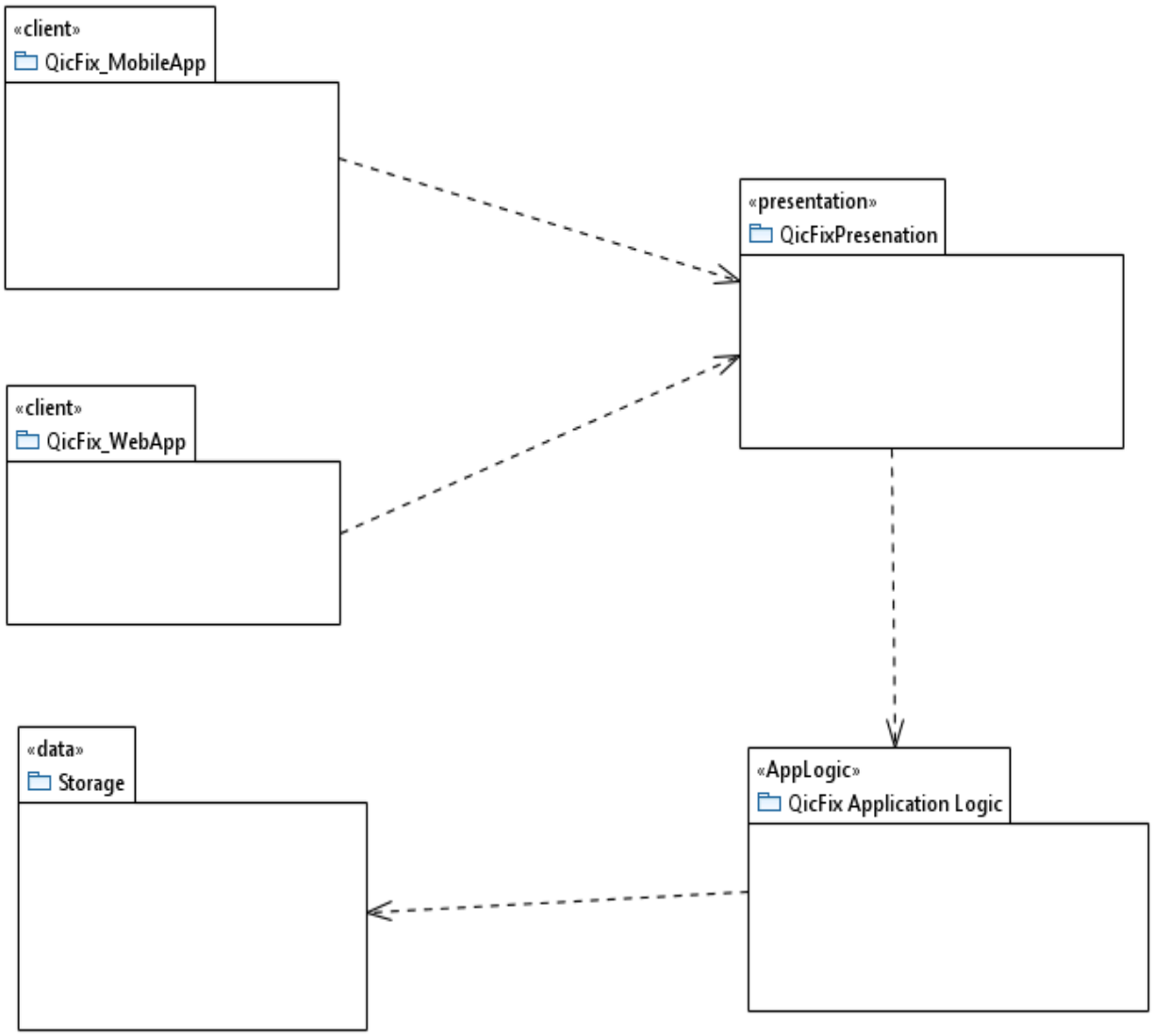


Figure-5.1.1: 4-Tier Architecture (Primary Architecture)

The Four-Tier Architecture will allow us to address the outlined non-functional requirements. Having separate Client and Presentation tiers allows us to decouple the specialized views and logic for each device from the overall presentation strategies. In addition, having the ability to present the system in multiple different ways depending on device allows us to tailor a highly usable system regardless of the type of platform, which further addresses the usability constraints we must adhere to.

Apart from the non-functional requirements addressed by the primary architecture, the requirements call for high reliability and robustness of the system. The second architecture that we have chosen is the MVC(Model-View-Controller) architecture. The MVC architecture is used within the Client subsystem (see Figure 5.1.2). The MVC architecture allows for a separation of responsibility between each system. This enables our client system to have low coupling and high cohesion within its subsystems. All packages have facades being used to access each other's

methods. Inside the Model there is all the persistent data in the client. In the Controller all the logic inside the client is done. Inside the view all possible UI views is held.

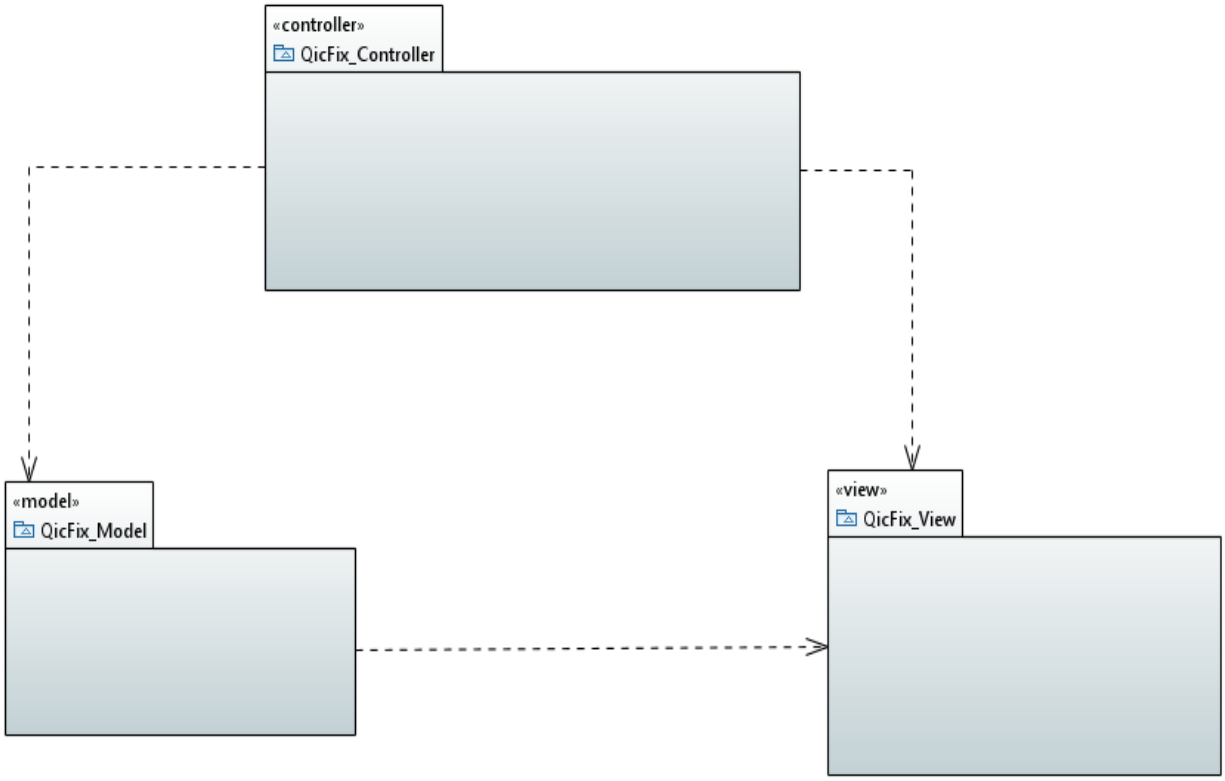


Figure-5.1.2: MVC Architecture (Secondary Architecture)

Future considerations as the system continues to scale include introducing more tiers strictly for load balancing. This would eventually include load balancing to multiple presentation servers and multiple data servers via data replication and redundancy strategies.

5.2. Subsystem Decomposition:

In this section, each major subsystem is described in detail and mapped to all associated use cases. Refer to Appendix A for the use case diagram of all implemented use cases.

- **Client –** Decomposed into MVC subsystems and it has two types of Client in it-Mobile and Web Client.

  Web – Includes all specialized views and view logic specific to web devices.
  Mobile Client – Includes all specialized views and view logic specific to mobile devices.
  ○ **Model**: It stores temporary data while performing operations or transactions. Represent the low-level entities as they will be persisted, along with their relationships.

  Associated use cases:
  - **QicFix-C-310_ListTowers**

- **QicFix-C-305_ChangeDestination**
- **QicFix-C-301_ChangePickup**
- **QicFix-T-204_ChargeClient**
- **QicFix-T-203_Accept/DeclineRequest**
- **QixFixT-205_Sec_SessionTimeout**
- **QicFix-T-202_ViewRatings**
- **QicFix-T-201_EditProfile**
- **QicFix-C-102_PayTower**
- **QicFix-C-103_ReviewTower**
- **QicFix-C-101_QueueTowers**

○ **View:** Display all the view related to the system.
Associated use cases are:

- **QicFix-T-403_Tower Registration**
- **QicFix-C-402_Client Registration**
- **QicFix-CT-401_Client Tower Login**
- **QicFix-CT-404_Client Tower Logout**
- **QicFix-C-310_ListTowers**
- **QicFix-C-305_ChangeDestination**
- **QicFix-C-301_ChangePickup**
- **QicFix-T-204_ChargeClient**
- **QicFix-T-203_Accept/DeclineRequest**
- **QixFixT-205_Sec_SessionTimeout**
- **QicFix-T-202_ViewRatings**
- **QicFix-T-201_EditProfile**
- **QicFix-C-102_PayTower**
- **QicFix-C-103_ReviewTower**
- **QicFix-C-101_QueueTowers**
- **QicFix-C-105_LoginAttempts**

o **Controller:** In the Controller all the logic inside the client is done.
Associated Use Cases are:

- **QicFix-T-403_Tower Registration**
- **QicFix-C-402_Client Registration**
- **QicFix-CT-401_Client Tower Login**
- **QicFix-CT-404_Client Tower Logout**
- **QicFix-C-310_ListTowers**
- **QicFix-C-305_ChangeDestination**
- **QicFix-C-301_ChangePickup**
- **QicFix-T-204_ChargeClient**
- **QicFix-T-203_Accept/DeclineRequest**
- **QixFixT-205_Sec_SessionTimeout**

- **QicFix-T-202_ViewRatings**
- **QicFix-T-201_EditProfile**
- **QicFix-C-102_PayTower**
- **QicFix-C-103_ReviewTower**
- **QicFix-C-101_QueueTowers**
- **QicFix-C-105_LoginAttempts**

- **Presentation**: Includes objects that will be serialized/de-serialized back and forth between the clients and the presentation logic. In order to display fine-grained data that is exactly relevant to specific views, these view models represent subsets of full data models.
  - **QicFix-T-403_Tower Registration**
  - **QicFix-C-402_Client Registration**
  - **QicFix-CT-401_Client Tower Login**
  - **QicFix-CT-404_Client Tower Logout**
  - **QicFix-C-310_ListTowers**
  - **QicFix-C-305_ChangeDestination**
  - **QicFix-C-301_ChangePickup**
  - **QicFix-T-204_ChargeClient**
  - **QicFix-T-203_Accept/DeclineRequest**
  - **QixFixT-205_Sec_SessionTimeout**
  - **QicFix-T-202_ViewRatings**
  - **QicFix-T-201_EditProfile**
  - **QicFix-C-102_PayTower**
  - **QicFix-C-103_ReviewTower**
  - **QicFix-C-101_QueueTowers**
  - **QicFix-C-105_LoginAttempts**

- **Application Logic:** Includes objects that encapsulate the appropriate business logic per each major type of entity.
  - **QicFix-T-403_Tower Registration**
  - **QicFix-C-402_Client Registration**
  - **QicFix-CT-401_Client Tower Login**
  - **QicFix-CT-404_Client Tower Logout**
  - **QicFix-C-310_ListTowers**
  - **QicFix-C-305_ChangeDestination**
  - **QicFix-C-301_ChangePickup**
  - **QicFix-T-204_ChargeClient**
  - **QicFix-T-203_Accept/DeclineRequest**
  - **QixFixT-205_Sec_SessionTimeout**
  - **QicFix-T-202_ViewRatings**
  - **QicFix-T-201_EditProfile**

- **QicFix-C-102_PayTower**
- **QicFix-C-103_ReviewTower**
- **QicFix-C-101_QueueTowers**
- **QicFix-C-105_LoginAttempts**

- **Storage**: Stores the persistent data of the QicFix System.
  Associated use cases are:
  - **QicFix-T-403_Tower Registration**
  - **QicFix-C-402_Client Registration**
  - **QicFix-CT-401_Client Tower Login**
  - **QicFix-CT-404_Client Tower Logout**
  - **QicFix-C-310_ListTowers**
  - **QicFix-C-305_ChangeDestination**
  - **QicFix-C-301_ChangePickup**
  - **QicFix-T-204_ChargeClient**
  - **QicFix-T-203_Accept/DeclineRequest**
  - **QixFixT-205_Sec_SessionTimeout**
  - **QicFix-T-202_ViewRatings**
  - **QicFix-T-201_EditProfile**
  - **QicFix-C-102_PayTower**
  - **QicFix-C-103_ReviewTower**
  - **QicFix-C-101_QueueTowers**
  - **QicFix-C-105_LoginAttempts**

5.3. Hardware and Software Mapping:

A deployment diagram in the Unified Modeling Language models the physical deployment of artifacts on nodes. The deployment diagram (Figure 2.3) shows 4 device nodes (Mobile, PC, Web server, DB server), 5 execution environment nodes, the artifacts and the communication links between different artifacts.

Client

The two clients are mobile device and PC. The Android mobile device e.g. Samsung Note 3 has a Quad core processor with a processing speed of 1.5 GHz, 2GB Ram and 32 GB internal memory and has a QicFix.apk file deployed on it. The mobile device communicates with Web Server (QicFix Server) at a transfer rate of 5 Mbit/sec.
The other client device Windows PC/Desktop has Intel Core Processor i3 5010U -CPU@ 2.10GHz, 8 GB RAM, 4 MB cache and 1 TB Internal memory and communicates with the Web Server through the Google Chrome browser artifact at a transfer rate of 30 Mbit/sec.

Web Server

The Web Server (QicFix Server) has an Intel Xeon Processor X5570 with a processing speed of 2.5 GHz, 1GB RAM and 8 TB HDD shared with SSD. It runs on Amazon Linux execution environment and uses Apache 2 Tomcat 8.0 to handle requests from the client. It communicates with clients through HTML pages and further uses artifacts like QicFix.war and googlemap.lib to process those requests. The webserver forwards requests to the Database Server at a transfer rate of 10 Gbit/sec.

The Web Server encompasses both the Presentation and the Business Logic tiers of our system. The team made this design choice in order to reduce latency of data transmission between the tiers. The system is designed in such a way that in the future if load balancing becomes a problem, the two tiers may be physically separated.

Database Server

The Database Server (QicFix Server) has an Intel Xeon Processor X5570 with a processing speed of 2.5 GHz, 1GB RAM and 8 TB HDD shared with SSD and runs on Amazon Linux execution environment. It uses MySQL 5.6 to store various database tables like Client Table, Tower Table and Payment Table.

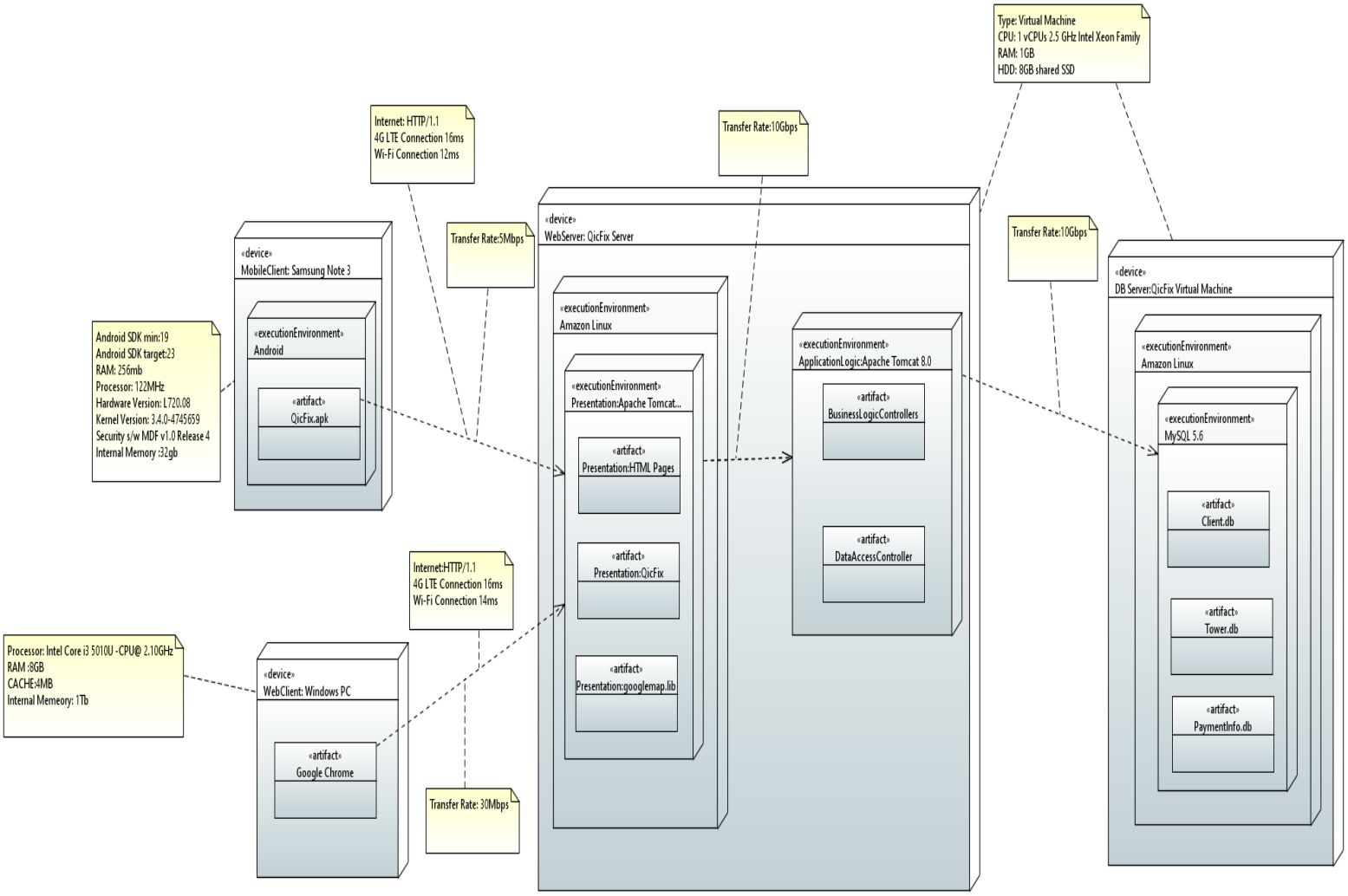


Figure-5.3.1: Deployment Diagram

5.4. Persistent Data Management:

The QicFix System keeps information in a persistent data storage. This data storage is modeled under the Relational Model and using MySQL DBMS version 5.5.46 Community Server (GPL). This Subsystem feeds with the requested information by the Web Services which sends the information to the client in Json format. The translation between Java Objects and Json Objects is made with Gson Library 2.6.2.

The Data Storage Subsystem is available permanently through the Web Services and it is backed up daily.
The main entities represented in this ER Diagram are:

- User,
- Client,
- Tower,
- Service,
- User_type,
- Payment, and
- Application

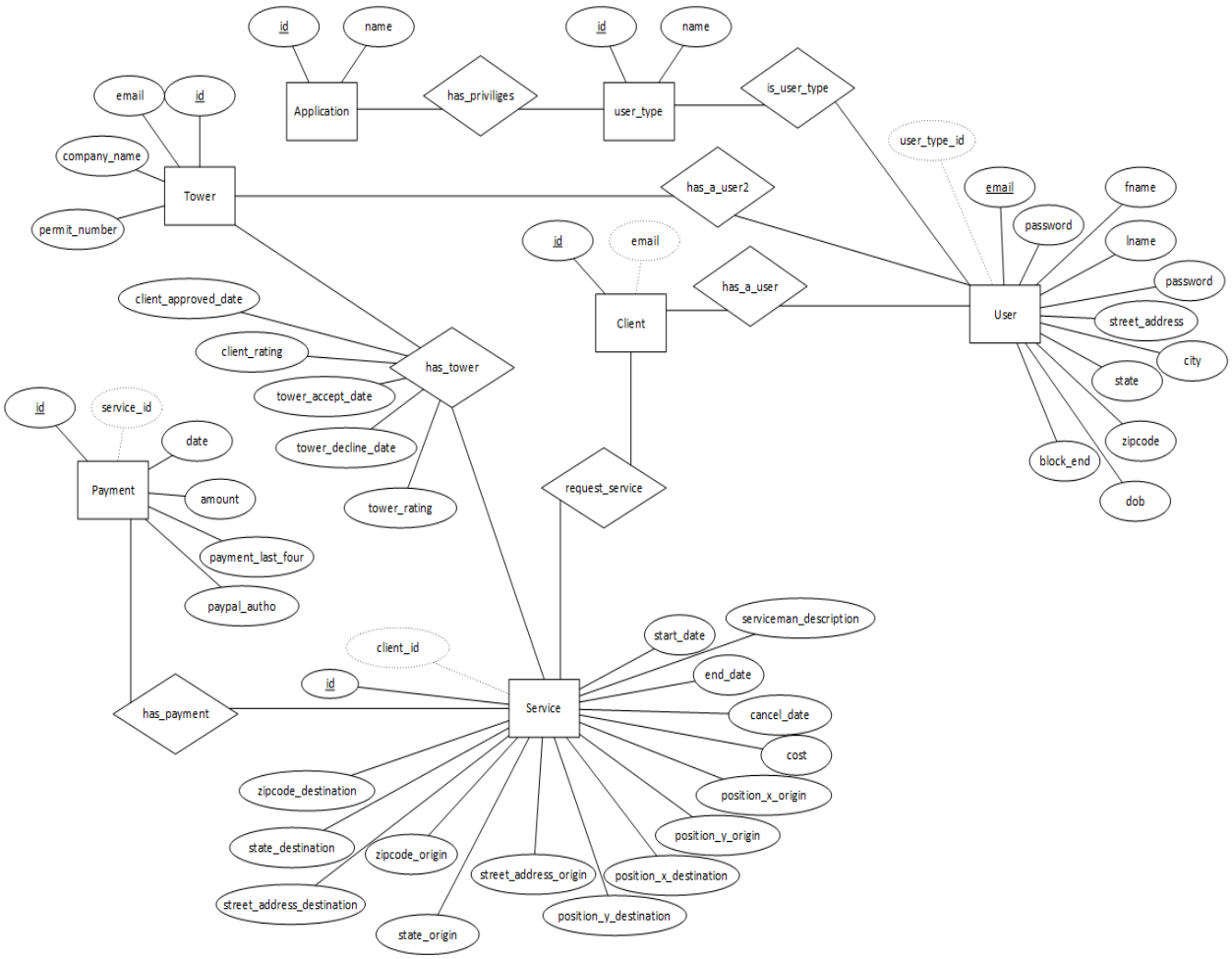


Figure-5.4.1: Access Control Matrix

5.5. Security Management:

QicFix system makes use of a global access table for the purposes of securing specific system functionality. For each role, specific functionalities are either allowed or restricted. The security to be implemented in QicFix consists in a Control Access Matrix (Table), which displays every option allowed to a certain role: client or serviceman, that they have access to.
Another security mechanism implemented:

- **Token Based Authentication:**
  QicFix creates a valid token per user and it validates this token per each request made to the server.

- **Session Timeout:**
  This feature will automatically logout idle users after 30 minutes of inactivity.

- **Maximum Login Attempts:**
  QicFix allows a limited number of login attempts to any account. Once the limit is reached, the account is locked and the owner is notified email.

ACCESS CONTROL MATRIX			
Activity	**Use Case Id**	**Client**	**Serviceman**
User Login	QicFix-CT-401_Client Tower Login	X	X
Client Registration	QicFix-C-402_Client Registration	X	
Tower Registration	QicFix-T-403_Tower Registration		X
User Logout	QicFix-CT-401_Client Tower Logout	X	X
Change Pickup Location	QicFix-C-301_ChangePickup	X	
Change Destination Location	QicFix-C-305_ChangeDestination	X	
List Towers	QicFix-C-310_BestRatedTowers QicFix-C-315_NearestTowers	X	
Tower Rating	QicFix-C-104_TowerReviews QicFix-T-202_ViewRatings	X	
Queue Tower	QicFix-C-101_QueueTowers	X	
Review Tower	QicFix-C-103_ReviewTower	X	
Payment	QicFix-C-102_PayTower	X	
Update User Account	QicFix-T-201_EditProfile	X	X
Create Request	QicFix-C-320_CreateRequest	X	
Update Request	QicFix-T-203_AcceptRequest QicFix-T-205_DeclineRequest	X	X
Charge Client	QicFix-T-204_ChargeClient		X

Figure-5.5.1: Access Control Matrix

6. Detailed Design:

This chapter describes the detailed object design for the QicFix Application. First, an overview of the classes in all subsystems that will be implemented are presented in the form of a minimal class diagram. Once all classes across all subsystems are defined, several state machines are shown describing both the overall system as well as the main control object for each major subsystem. The state machines are presented in a two-level fashion. Next, a set of sequence diagrams are presented in order to showcase interactions amongst objects within the system. Furthermore, the detailed class designs are shown in a hierarchical decomposition fashion. All necessary design patterns that are to be implemented are also described in detail. Finally, Object Constraint Language is used to describe constrains for the main control object in each major subsystem.

6.1. Overview

This section describes a minimal class diagram used to display all classes for the subsystems to be implemented. The minimal class diagram does not show attributes nor methods. The entire minimal class diagram is shown first. Subsequently, for each major grouping of classes on the minimal class diagram, a separate diagram is shown, and each class is briefly described.

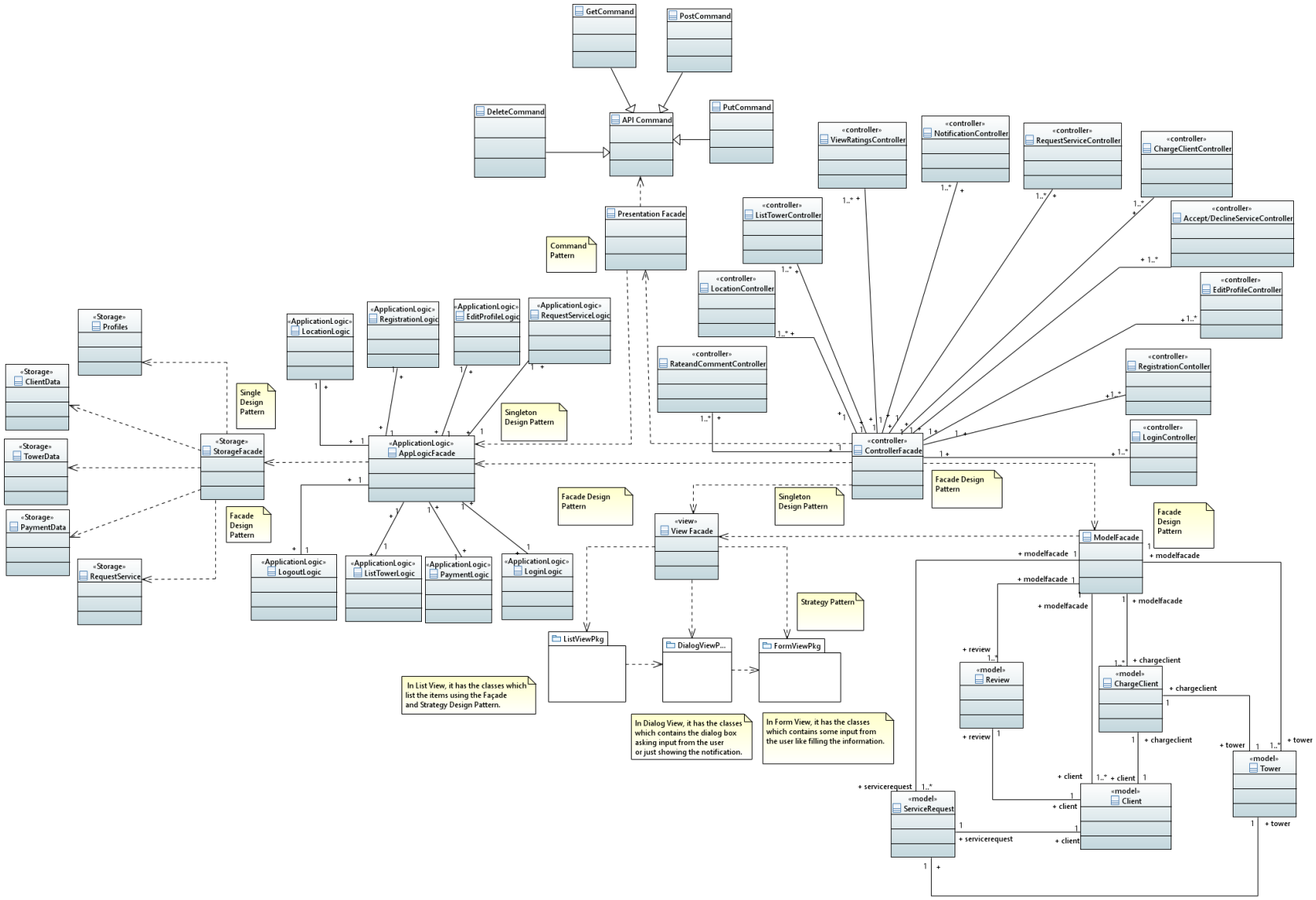


Figure-6.1: Minimal ClassDiagram

The first subsystem for the Client is described is the Model subsystem. The classes for this subsystem are described below:

- **Model Façade:** Responsible for decoupling the Model subsystem from the Controller and View subsystem. Provides a single interface for accessing various domain services. Follows the Façade design pattern in order to provide a unified interface to a set of interfaces in the Controller and View subsystem, in turn simplifying the interaction between the Controller Model and View subsystems.

- **Client**: Represents a domain model for a Client.

- **Tower:** Represents a domain model for a Tower.

- **Service**: Responsible for providing the ability to request service and cancel service from the Tower for the job needed to be done.

- **ChargeClient:** Responsible for providing the ability to Charge the Client after the Job has been done by the Tower.

- **Review:** Responsible for providing the ability to the Client about the Tower Ratings and Comments given by other Clients upon the completion of the job.

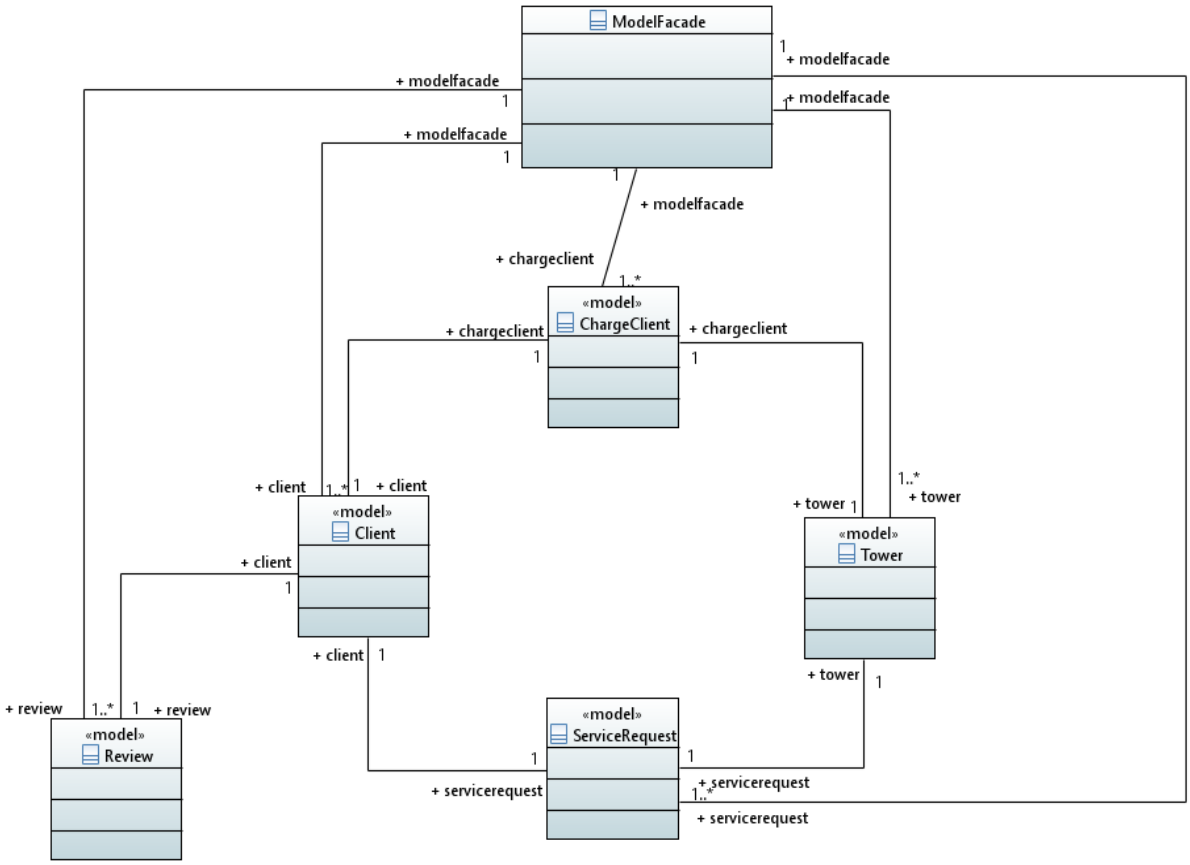


Figure-6.2: Minimal_ClassDiagram (Model Subsystem)

The second subsystem described is the Controller subsystem. The classes for this subsystem are described below:

- **Controller Facade:** The main control object of the presentation subsystem. Responsible for accepting HTTP requests and coordinating communication to the Business Logic subsystem and creation and serialization of view model for each specific device. Provides endpoints for authentication, getting data, posting data, session management, data serialization and deserialization, and for changing views for a particular client.

- **LoginController:** The control object of the LoginView.

- **RegistrationController:** The control object of the Registration View in the Form View Subsystem.

- **RequestServiceController:** The control object of the Requesting Service for the Service Model in the Model Subsystem.

- **EditProfileController:** The control object of the EditProfileView in the Form View Subsystem.

- **Accept/DeclineServiceController:** The control object of the Accept/Decline notification View in the Dialog View Subsystem.

- **NotificationController:** The control object of the Notification View in the Dialog View Subsystem.

- **ChargeClientController:** The control object of the Charge Client View in the Form View Subsystem.

- **ListTowerController:** The control object of the ListTower View in the List View Subsystem for listing the Tower for requesting the Job.

- **LocationController:** The control object of the InputAddress View where we need to delegate the Googlemap.lib for retrieving the location.

- **ViewRatingController:** The control object of the List the Rating of the Tower given by the Client in the List View Subsystem.

- **RateandCommentController:** The control object of the Rate and Comment View which provides details to the class Rate and Comment in Dialog View Subsystem.

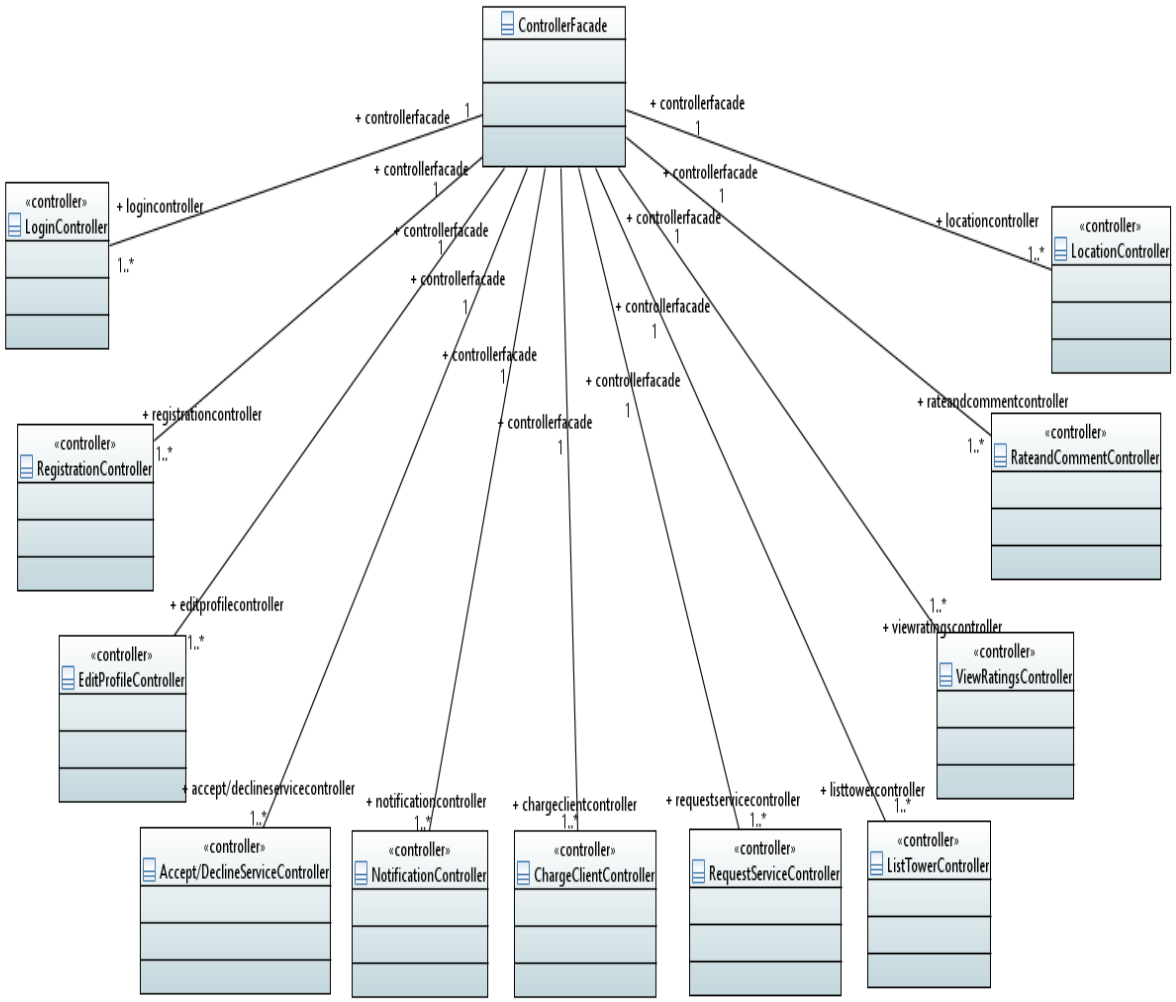


Figure-6.3: Minimal_ClassDiagram (Controller Subsystem)

The third subsystem described is the View subsystem. The View system is further divided into the 3 Views as Form View, List View, Dialog View. The overall View Subsystem is shown below in Figure-3.4). And further, there is decomposition of the View Subsystem with its classes are shown below:

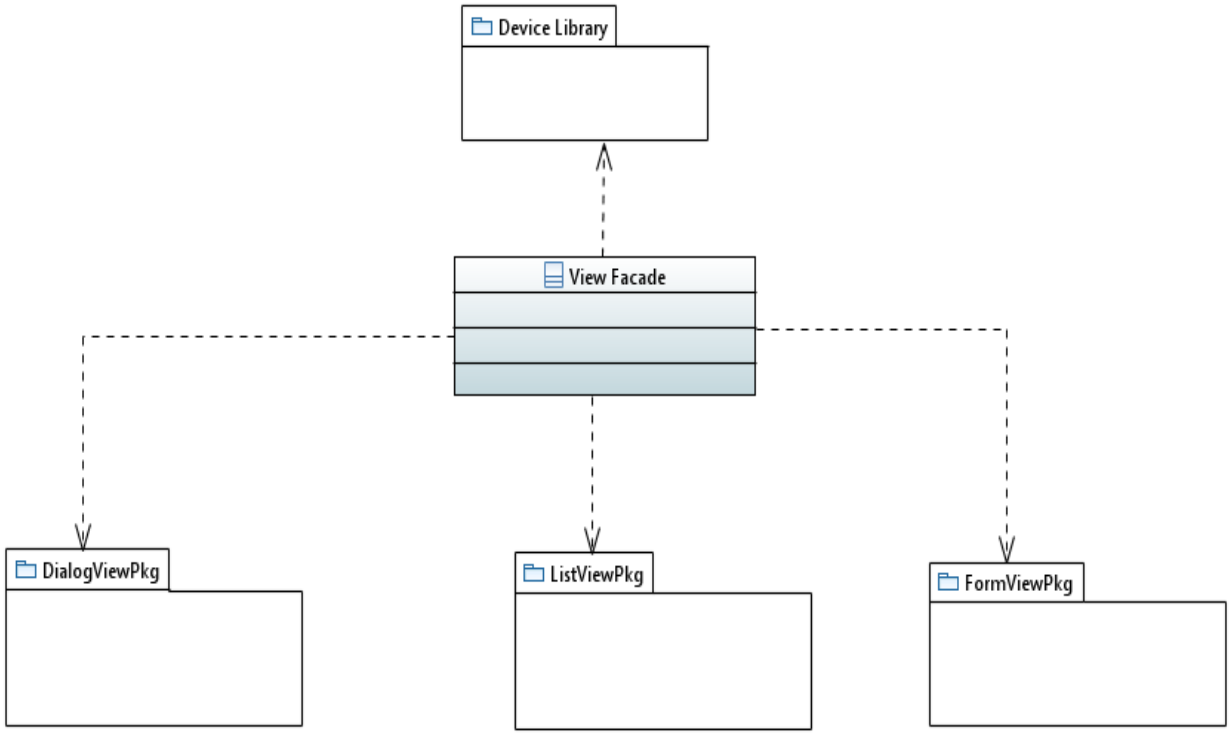


Figure-6.4: Minimal _Class Diagram(View Subsystem)

Further, the View Subsystem is decomposed into List View Package, Form View Package and Dialog View Package. The Classes for all of the is described and shown below:

List View: In List View, it has the classes which list the items using the Façade and Strategy Design Pattern. The classes are as follow:

- **ListFacade:** Responsible for decoupling the List View subsystem from the Form View and Dialog View subsystem. Provides a single interface for accessing various domain services. Follows the Façade design pattern and Strategy Design Pattern.

- **ListTowerView:** An abstract class which follows the strategy design pattern. Responsible for defining a contract for any view strategy class. A view strategy defines a means of mapping from a low-level system entity to an appropriate view model. Responsible for creating the List of Towers by applying the algorithm of sorting.

- **BestTowerView:** Responsible for creating a view model containing Best Tower List with its information. Follows the strategy design pattern.

- **CheapestTowerView:** Responsible for creating a view model containing Cheapest Tower List with its information. Follows the strategy design pattern.

- **ClosestTowerView:** Responsible for creating a view model containing Closest Tower List with its information. Follows the strategy design pattern.

- **ListRatingView:** Represents compact and filtered data regarding the rating and comments for the Tower given by the Client.

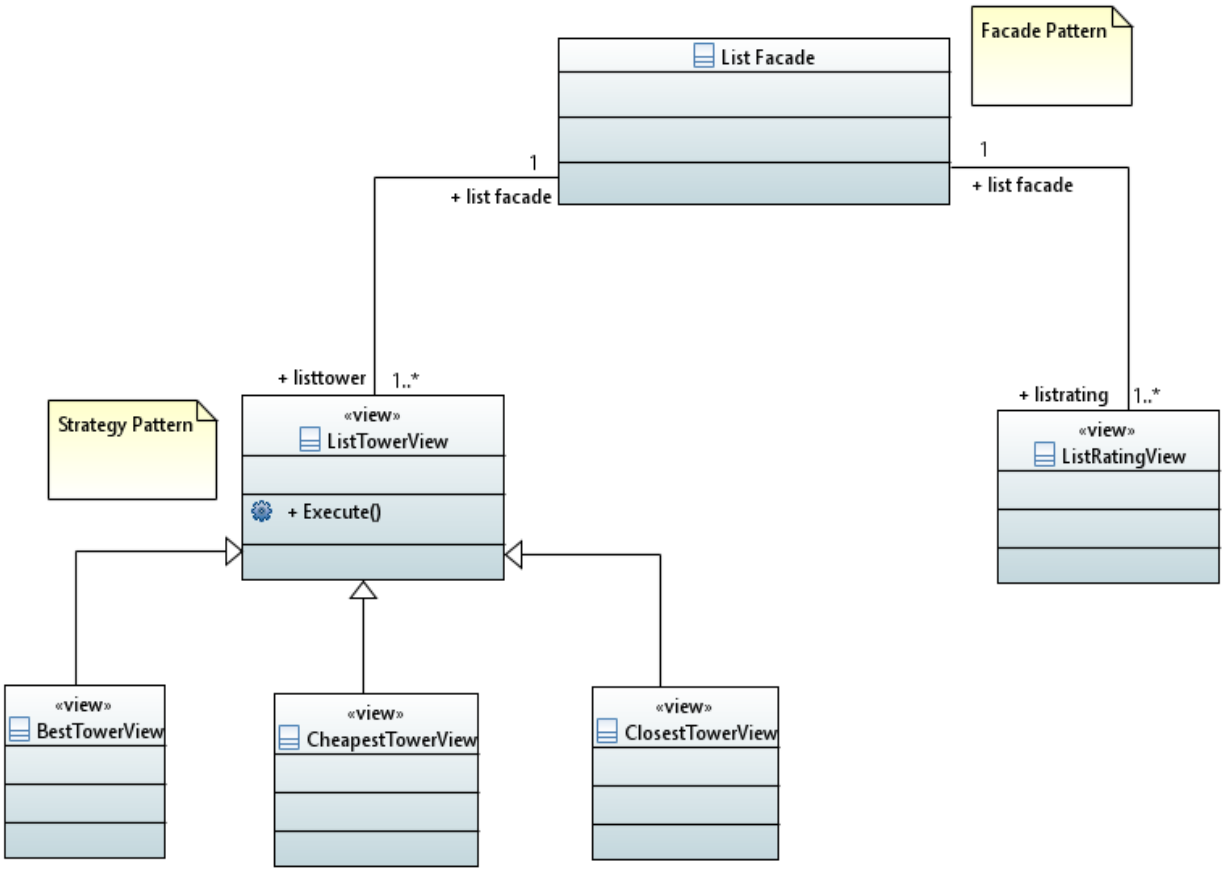


Figure-6.5: Minimal Class Diagram (ListView Subsystem)

Form View: In Form View, it has the classes which contains some input from the user like filling the information. The Classes for the Form View Subsystem is as follow:

- **FormFacade:** Responsible for decoupling the Form View subsystem from the List View and Dialog View subsystem. Provides a single interface for accessing various domain services. Follows the Façade Design pattern, Strategy Design Pattern and Singleton Design Pattern.

- **LoginView:** Represents the Login View to the Client/Tower. Follows the Singleton Design Pattern.

- **RegistrationView:** An abstract class which follows the strategy design pattern. Responsible for defining a contract for any view strategy class. A view strategy defines a means of mapping from a low-level system entity to an appropriate view model. Responsible for Notifying the client about different things.

- **ClientRegistrationView:** Responsible for creating a view model containing the Registration Page for the Client. Follows the strategy design pattern.

- **TowerRegistrationView:** Responsible for creating a view model containing the Registration Page for the Tower. Follows the strategy design pattern.

- **ChargeClientView:** Responsible for creating a view model containing the Charge and Client name to be charged for the job completed by the Tower for the Client.

- **InputAddressView:** Responsible for the creating the View model for giving the location for the job to be done by the Tower.

- **EditProfileView:** Represents the view in a compact and filtered way for editing the profile for Client/Tower.

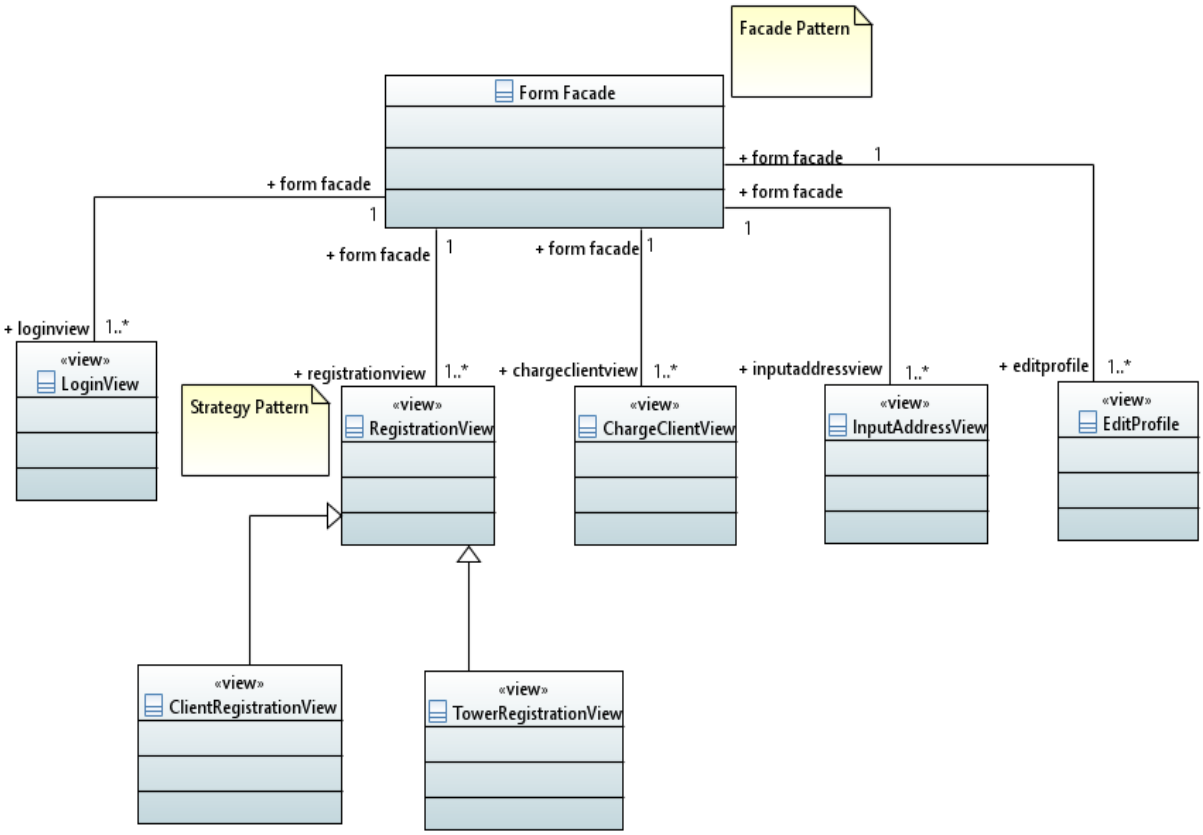


Figure-6.6: Minimal_Class Diagram (Form View Subsystem)

Dialog View: In Dialog View, it has the classes which contains the dialog box asking input from the user or just showing the notification. The classes for the Dialog Subsystem is as follow:

- **Dialog Façade:** Responsible for decoupling the Dialog View subsystem from the Form View and List View subsystem. Provides a single interface for accessing various domain services. Follows the Façade design pattern and Strategy Design Pattern.

- **RatingandCommentsView:** Represents the view of rating and comments for the Tower by the Client after job has been done.
- **JobRequestView:** Represents the view of Requesting Job to the Tower selected by the Client.
- **NotificationView:** An abstract class which follows the strategy design pattern. Responsible for defining a contract for any view strategy class. A view strategy defines a means of mapping from a low-level system entity to an appropriate view model. Responsible for Notifying the client about different things.
- **Re-RequestTowerView:** Represents compact and filtered notification for the re-requesting the tower as it will be shown to a Client. Follow the Strategy Design Pattern.

- **DeclineRequestNotifyView:** Represents compact and filtered notification for the Decline request by the tower as it will be shown to a Client. Follow the Strategy Design Pattern.
- **AcceptRequestNotifyView:** Represents compact and filtered notification for the Accept request by the tower as it will be shown to a Client. Follow the Strategy Design Pattern.
- **ChargeNotificationView:** Represents compact and filtered notification for the Charge needed to be paid for the job done by the tower as it will be shown to a Client.

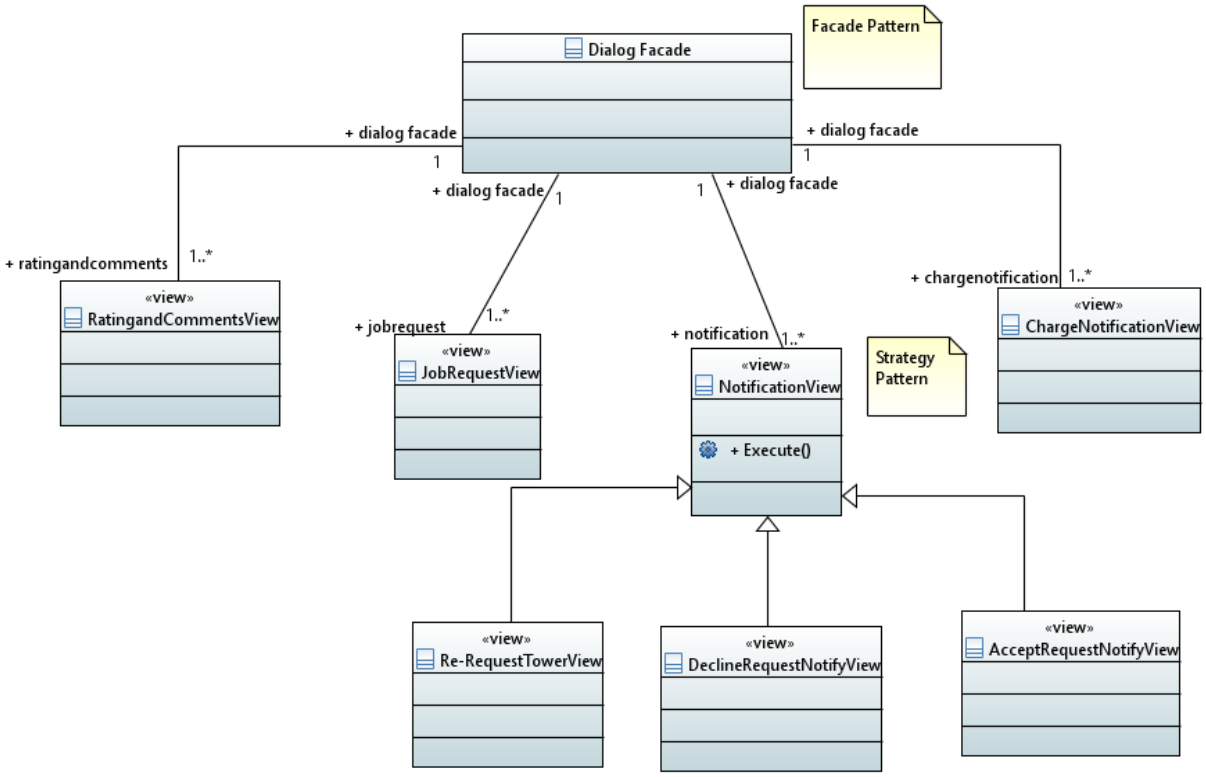


Figure-6.7: Minimal Class Diagram (Dialog View Subsystem)

The Second subsystem described is the **Presentation subsystem**. The classes for this subsystem are described below:

- **Presentation Façade:** Responsible for decoupling the Presentation subsystem from the Application Logic and the Client subsystem. Provides a single interface for accessing various domain services. Follows the Façade design pattern and Command Design Pattern.
- **API Command:** The main control object of the presentation subsystem. Responsible for accepting HTTP requests and coordinating communication to the Application Logic subsystem and creation and serialization of view model for each specific device.
- **Get Command:** Requests data from a specified resource such as Requesting TowerList for Requesting Service.
- **Put Command:** Submits data to be processed to a specified resource.E.g.It uploads information passed from Client from QicFix system to a server.
- **Post Command:** Submits data to be processed to a specified resource.

- **Delete Command:** Deletes the specified resource.

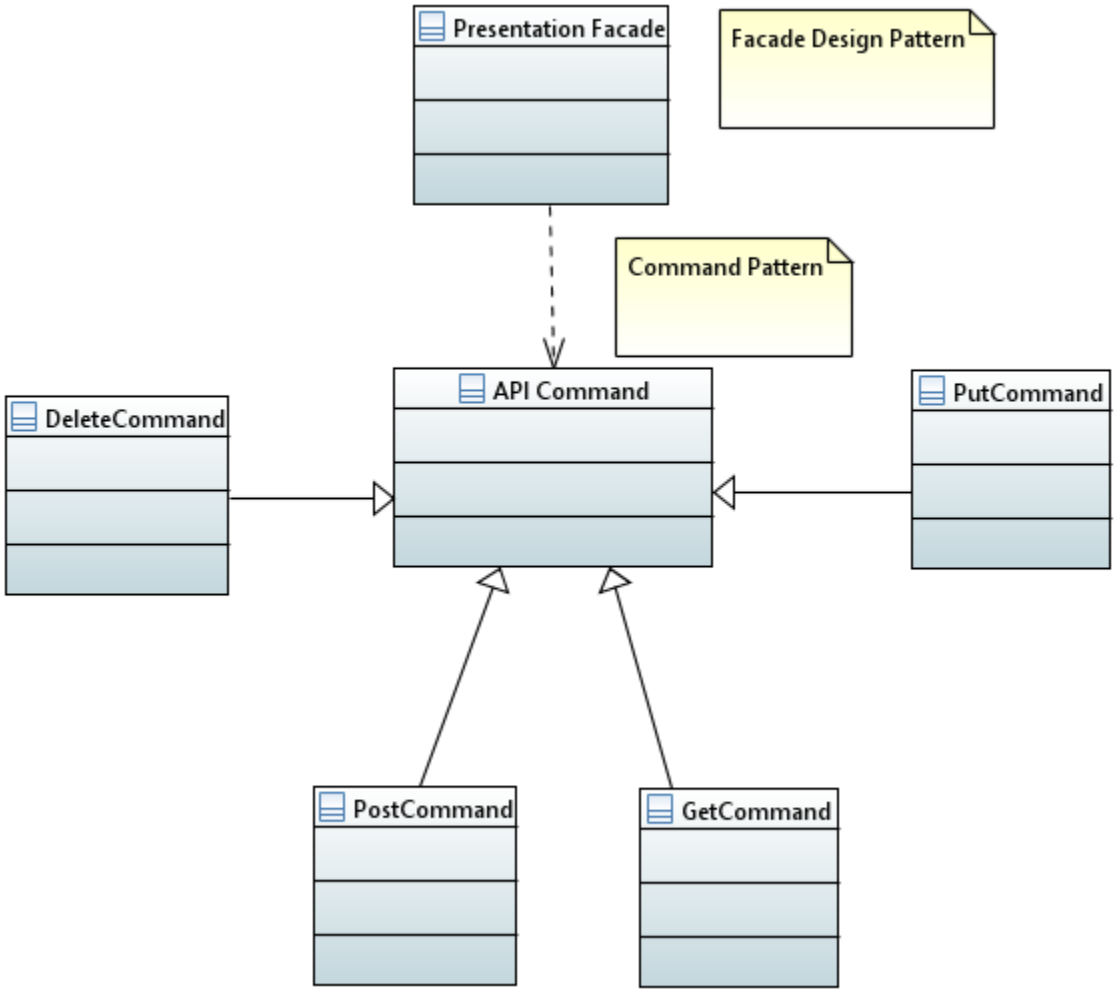


Figure-6.7: Minimal Class Diagram (Presentation Subsystem)

The Third subsystem described is the Application Logic subsystem. The classes for this subsystem are described below:

- **App Logic Façade**: Responsible for decoupling the Application Logic subsystem from the Presentation subsystem. Provides a single interface for accessing various domain services. Follows the Façade design pattern.

- **Registration Logic:** Responsible for Registration Logic which receives JSON Object and stores the details of the Tower passed from the Presentation Facade.

- **Request Service Logic:** Responsible for creating a request from Client Side and storing it into the StorageFacade.

- **Edit Profile Logic:** Responsible for Updating the Profile of the Tower or Client and passing it to the Storage Facade in form of Content return from the JSON.

- **List Tower Logic:** Responsible for Listing Tower Logic from the Presentation and passing it to the Storage facade and giving back the List of tower retrieved from Storage facade to Presentation Facade.

- **Login Logic:** Responsible for the Login Logic of the QicFix System for the Tower and Client.

- **Logout Logic:** Responsible for the Logout Logic of QicFix System for Tower and Client.

- **Payment Logic:** Responsible for the Payment and Charging Client Logic of the QicFix System.

- **Location Logic:** Responsible for interacting with the Google API for retrieving Client Pickup Location giving Destination for the service requested and for retrieving the Tower Location and tracking the route of QicFix System.

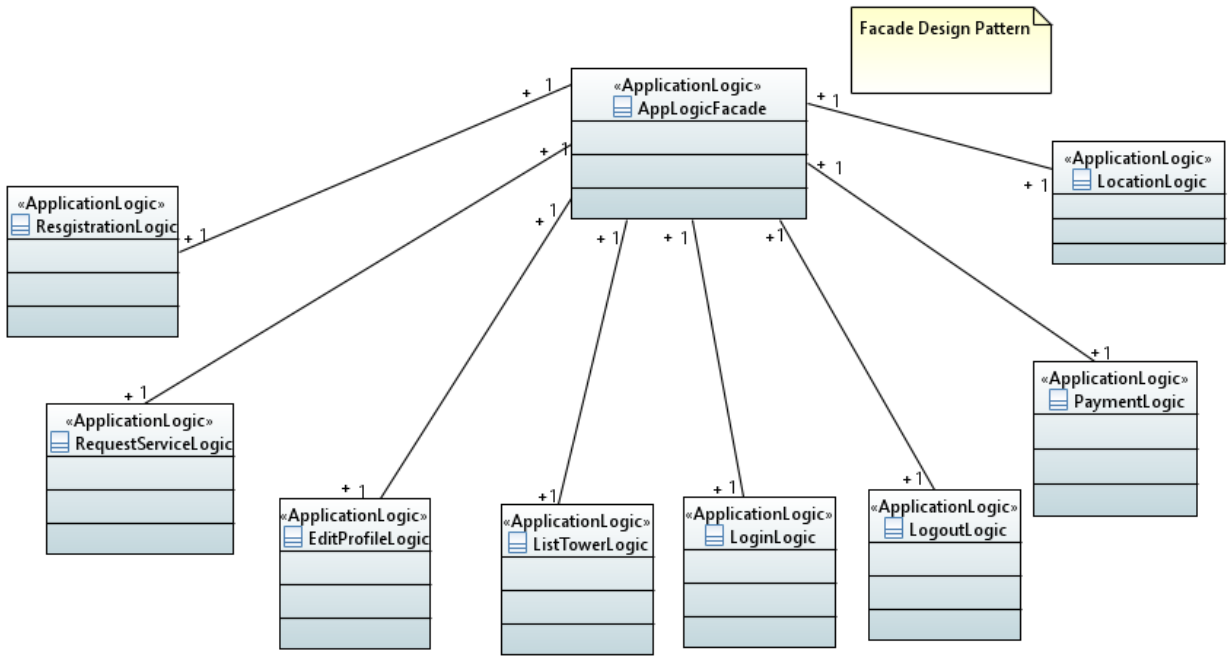


Figure-6.9: Minimal Class Diagram (Application Logic Subsystem)

The fourth and final subsystem described is the Data subsystem. The classes for this subsystem are described below:

- **Storage Façade:** Responsible for decoupling the Data subsystem from the Application Logic subsystem. Provides a single interface for accessing various data-related operations. Follows the Façade and Singleton design patterns.

- **Client Data**: Represents a domain model for a specialized Client user data.

- **Tower Data:** Represents a domain model for a specialized Tower user data.

- **Profiles:** Provides data access to profiles of the Client and Tower.

- **Request Service:** Represents a domain model for user-specific request made for the service and for the provided service information.

- **Payment Data:** Stores the Data related to the Charges made for the service provided by the Tower to the Client.

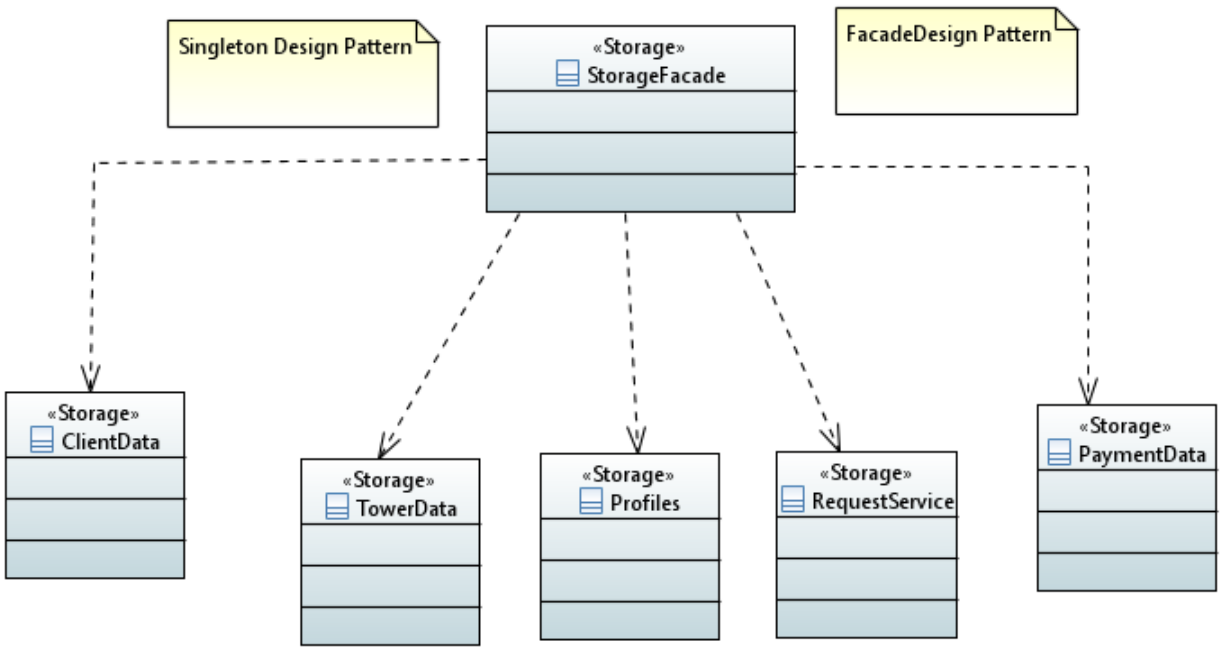


Figure-6.10: Minimal Class Diagram (Storage Subsystem)

6.2. State machine:

State machine diagrams are one of the five UML diagrams used to model the dynamic nature of a system. A state machine diagram is a directed graph of states connected by transitions, which act as the formal specification of the behavior of a class. State machine diagrams define different states of an object during its lifetime, along with events that change these states. State machine diagrams are useful for modeling how the system responds to external or internal events. They allow us to describe the flow of control from one state to another state. States are defined as a condition in which an object exists and it changes when some event is triggered. The most important purpose of state machine diagrams is to model the lifetime of an object from creation to termination.

In our state machine diagram, we have shown the overall system flow by the overall state machine diagram. We have two major subsystems: client and tower. We have shown the subsystem state machines in the second level by the respective sub machines; therefore, our state machine diagram has in total two levels of hierarchy.

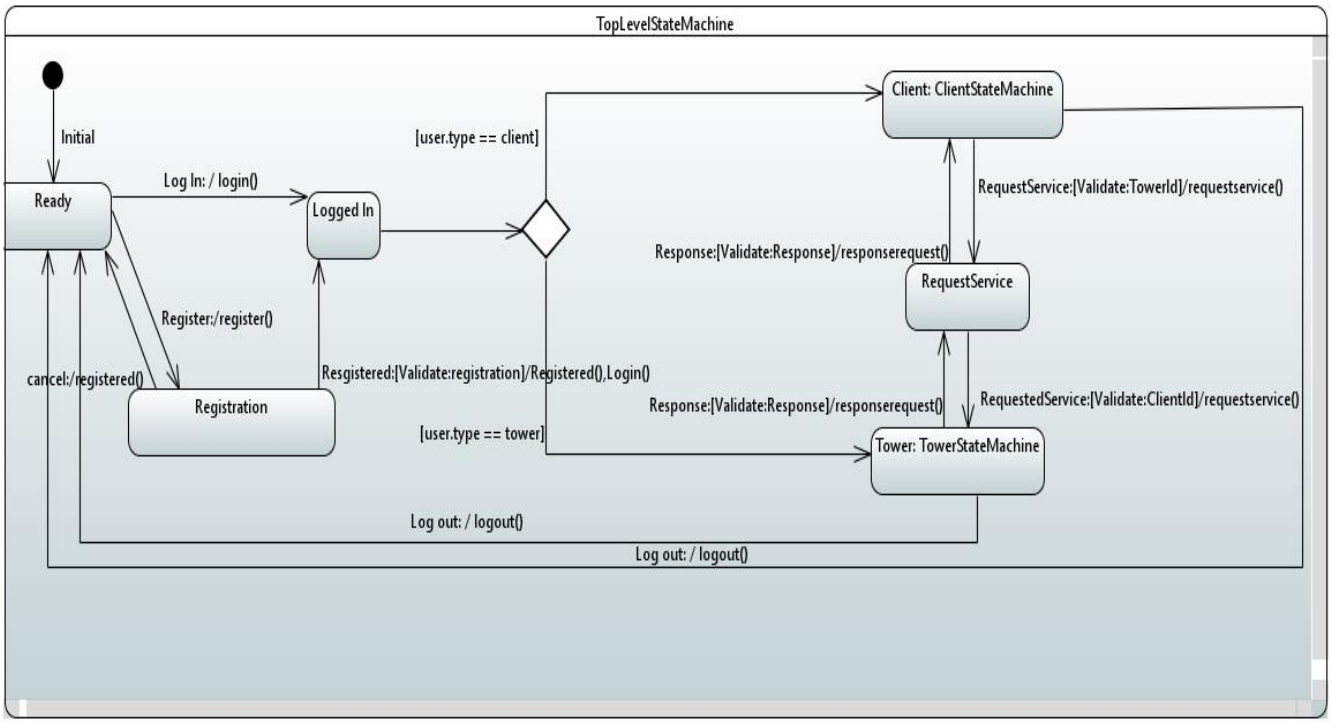


Figure-6.2.1: Top –Level State Machine

The states are:

1. **Registration –** represents the state of the system where user has to register for accessing the services of the QicFix System

2. **Logged in -** represents the state of the system after the user logs in and the system is determining the role of the user.

3. **Client: ClientStateMachine** - represents the second level state machine for the Client.

4. **Request Service –** represents the state the where request is generated for the particular tower and reponse given is send back to the client.

5. **Tower: TowerStateMachine -** represents the second level state machine for the Tower

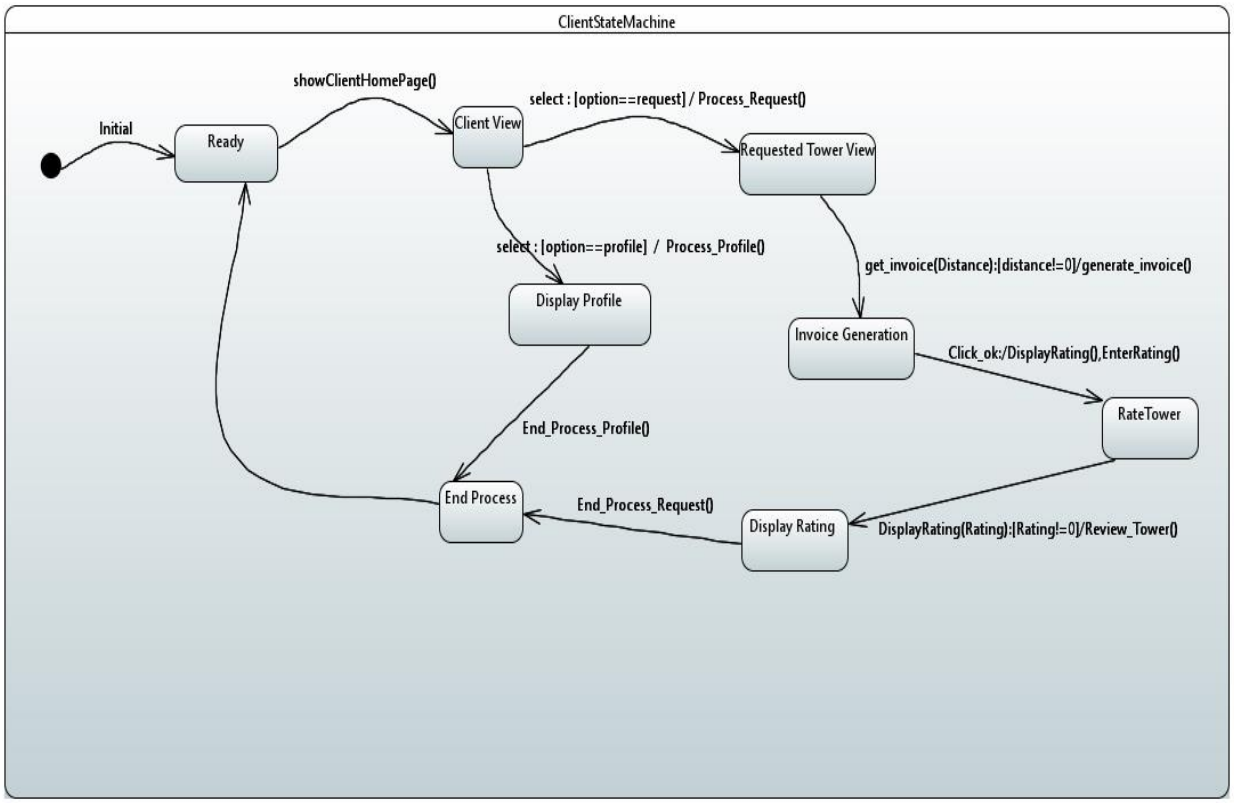


Figure-6.2.2: Client State Machine(Second Level)

The states are:

1. **Ready -** Represents that the Client View is ready to launch.

2. **Client View -** Represents the homepage state for the Client. This is where each Client begin when they are logged into the system.

3. **Display Profile-** Represents profile state that allows the client to change information about themselves in the system.

4. **Request Tower View -** Represents the request state that allows the client to contact local Towers around them.

5. **Invoice Generation -** Represents the invoice state where a request is processed and the client was charged a fee. The system display the invoice object to client in a form of a receipt.

6. **Rate Tower -** Represents the Rating object available in the system. It allows for clients to rate and comment on their Tower. The Rating Object is stored within the Tower.

7. **Display Ratings -** Represents the Rating of the tower given by the Client

8. **End Process -** Represents the end of flow for state and client back to the ready state.

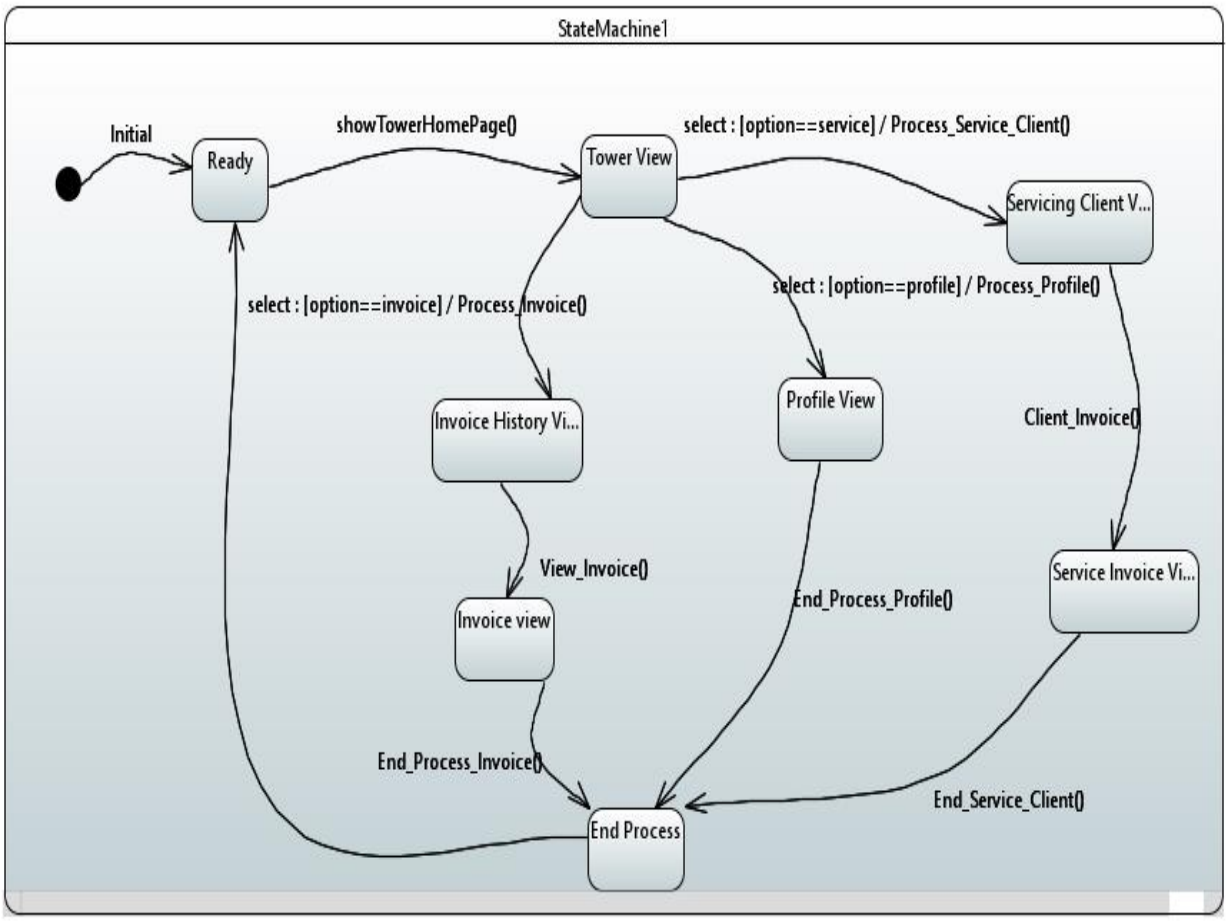


Figure-6.2.3: Tower State Machine(Second Level)

The states are:

1. **Ready -** Represents that the Client View is ready to launch

2. **Tower View -** Represents the homepage state for the Tower. This is where each Tower begin when they are logged into the system.

3. **Profile View -** Represents profile state that allows the tower to change information about themselves in the system.

4. **Service Client View -** Represents the request state that allows the Tower to contact Client they selected.

5. **Invoice History View -** Represents the list of all invoices the Tower has issued to Client in the application.

6. **Invoice View -** Represents the individual invoice the Tower can view from the list of invoices.

7. **Service Invoice View -** Represents the state where a service is processed and the client was charged a fee. The system displays the invoice object to tower in a form of a receipt.

8. **End Process -** Represents the end of flow for state and tower back to the ready state.

6.3. Object Interaction:

In this section, we present diagrams in order to describe interactions among the different elements in our design model. The purpose of interaction diagrams is to visualize the interactive behavior of the system. This interaction is a part of dynamic behavior of the system. The most common interaction behavior in UML is represented by a Sequence diagram. A sequence diagram emphasizes the sequence of messages between objects in the system. In our system we are implementing 12 use cases and thus presents 12 sequence diagrams.

1. **QicFix-T-201-EditProfile :**

   This use case allow a User(Client/Tower) to Edit its Profile into the system so that the system can have the Updated information for the User. This use case is initiated by a Tower clicking on the EditProfile button on its ProfileScreen.A Tower starts the usecase by calling the EditProfile() function which displays a EditProfileView with the help of Form Facade. The Tower update the filed which he/she wants to update in his/her profile "Submit". EditProfile View asks for confirmation via a popup. On successful confirmation, the request is posted as a string to the ControllerFacade via View Facade which extracts the fields user is updating and stores it into the database of the user.AppLogic updates the data using the ProfileId of the User and passes a response object to the Controller Facade via AppLogicFacade and PresentationFacade. Then the notification is shown to the user of successfully updating the profile.

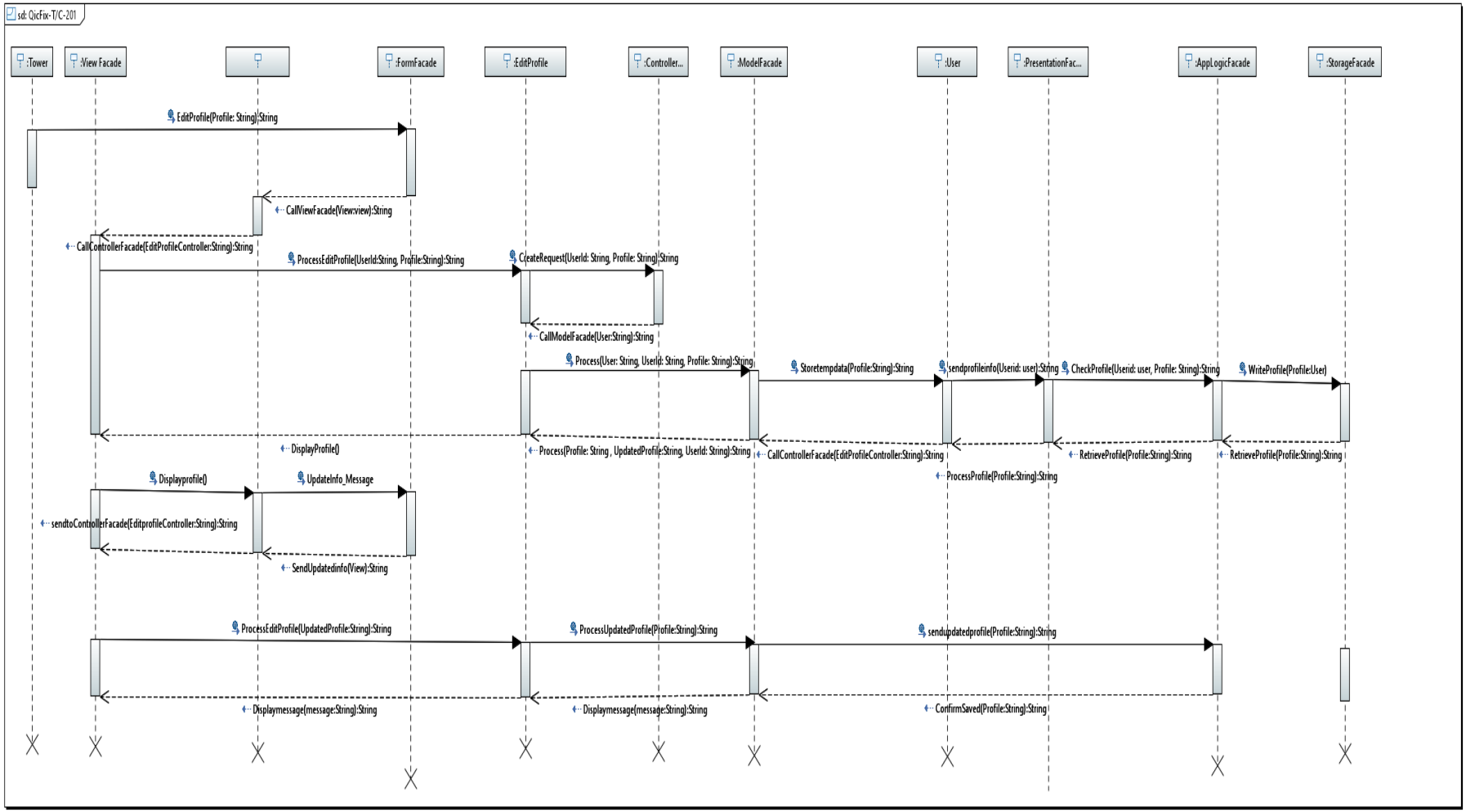


Figure 6.3.1: Sequence Diagram for QicFix-T-201-EditProfile

2. QicFix-T-202-ViewRatingandComments:

This use case allows a Client to view all reviews about the Tower given by the Client after the Service is completed by the particular Tower. The use case is initiated by a Client clicking on the "View RatingandComment" view button on the home screen. The client request is passed to the ListView Facade of the View Facade.Then it calls the Controller Facade to interact with the controller to perform the particular task of retrieving all the reviews of the Tower for the selection of the tower to request the job. The Client request is passed to through ViewRatingController and then to Model Facade for the Tower Profile. Then request is forwarded to the PresentationFacade and AppLogicFacade to retrieve the Tower Profile regarding the Reviews. Using the ControllerFacade, the PresentationFacade request for all reviews given about the Tower from the TowerDatabase. After successfully getting all the review object, it'll pass them to ViewRatingsController through the AppLogicFacade which again make a request to ViewFacade. This ViewFacade is passed to HomeScreen through JSON.

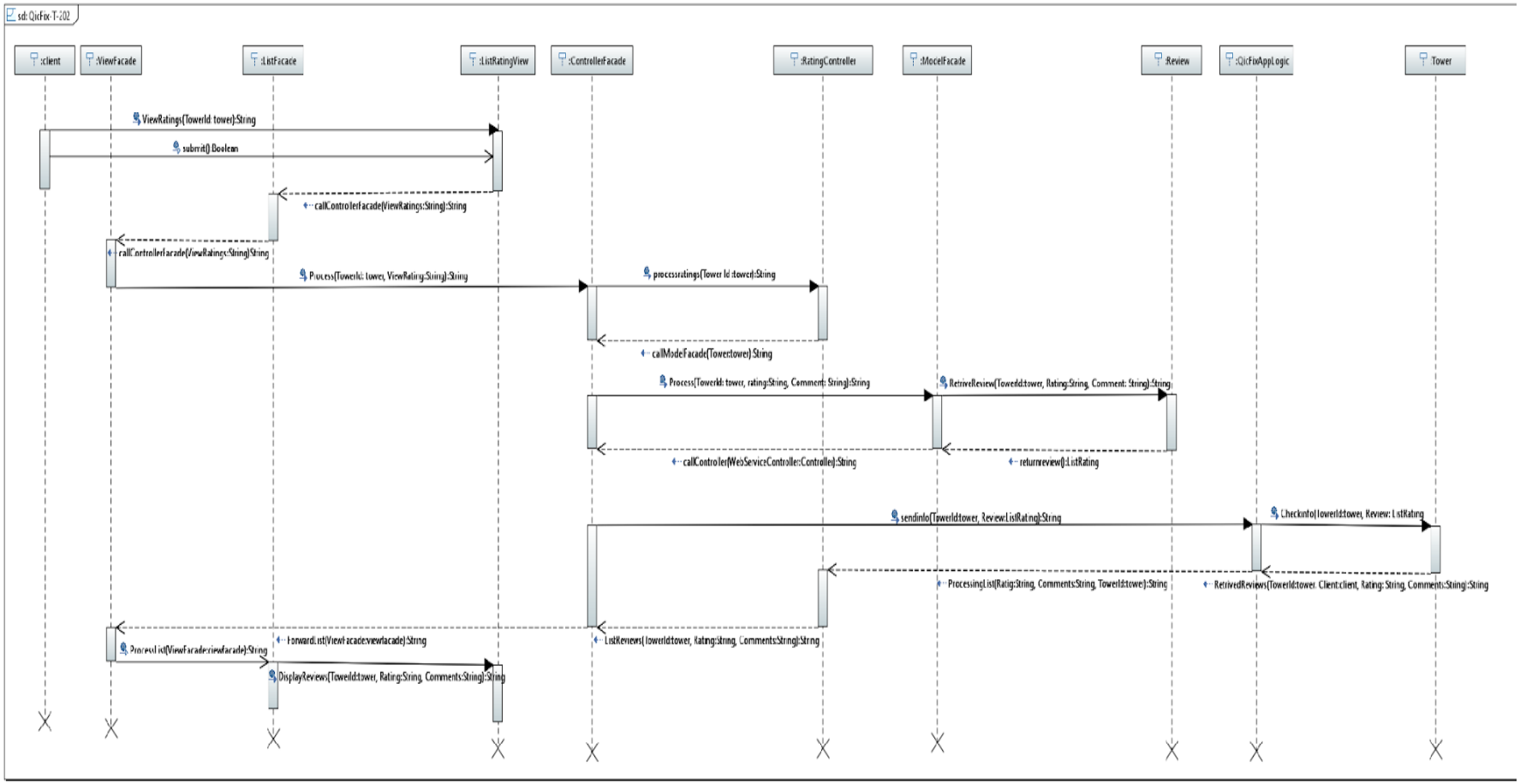


Figure 6.3.3: Sequence Diagram for QicFix-T-202-ViewRatingandComments

3. QicFix-T-203-Accept/DeclineRequest :

This Sequence Diagram Starts with the Tower getting Request from Client about the Service on the Tower's Accept/DeclineScreen.The View Facade coordiantes the flow of the action by passing the request to the Controller Facade. Then, controller Facade pass the request to the Accept/Decline Controller.Now, here is the alternate condition. Either tower can accept the request or decline the request. Based on the response of the Tower, web service interacts with the AppLogicFacade and the Request data are store and the else part of the condition terminates. And the flow is in the Active part of the request and then AppLogicFacade and PresentationFacade passes the information to the Controller facade and Controller Facade notifies the client about the Tower's Response.

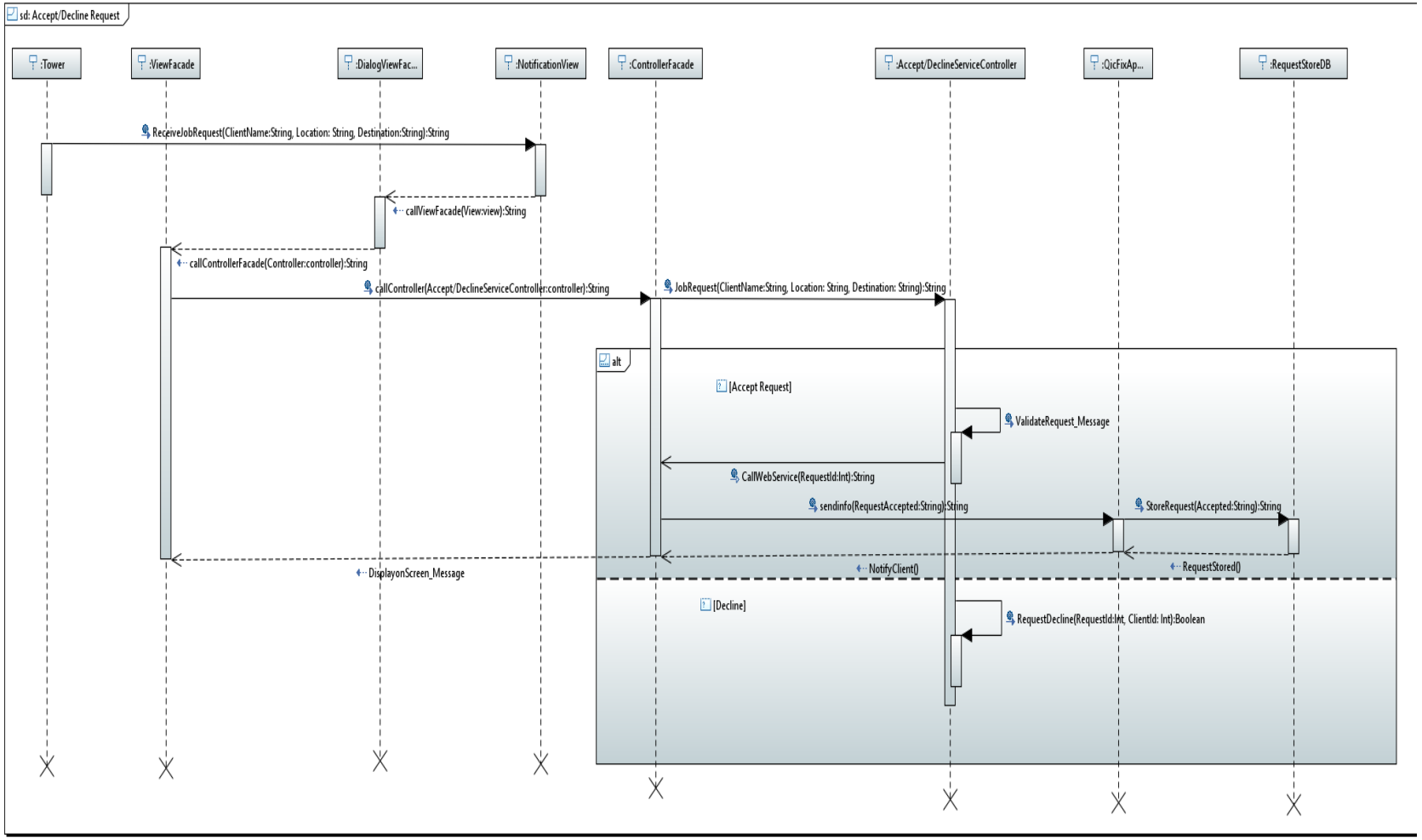


Figure 6.3.3: Sequence Diagram for QicFix-T-203-Accept/DeclineRequest

4. QicFix-T-204-ChargeClient :

This use case allows a Tower to Charge a client for the job provided. The use case is initiated by a Tower clicking on the “ChargeClient” button on the screen showed up after the the Tower has completed the Job requested by the Client. The Tower input the “Towed Miles”, “Amount”, and “ClientName” which is getting converted into CleintId. And Hits the submit button. The Tower request is passed to the ChargeClient of the Form Facade. The Form Facade pass the request to the ViewFacade which inturn, calls the Controller Facade to interact with the Paymentcontroller, ModelFacade and PresentationFacade to perform the particular task of Charging a Client by providing the information Job Done.Then the PresentationFacade calls the AppLogicFacade to store the data provided by the Tower. After successfully storing data in ChargeRecordsDB. it’ll pass them to PaymentController through the AppLogicFacade and PresentationFacade which again makes a request to ViewFacade. This ViewFacade is passed to HomeScreen through JSON.Sending Notification to Client about the Amount Charged and Job Details by the Tower. And Tower gets confirmation that Client has been Successfully Charged.

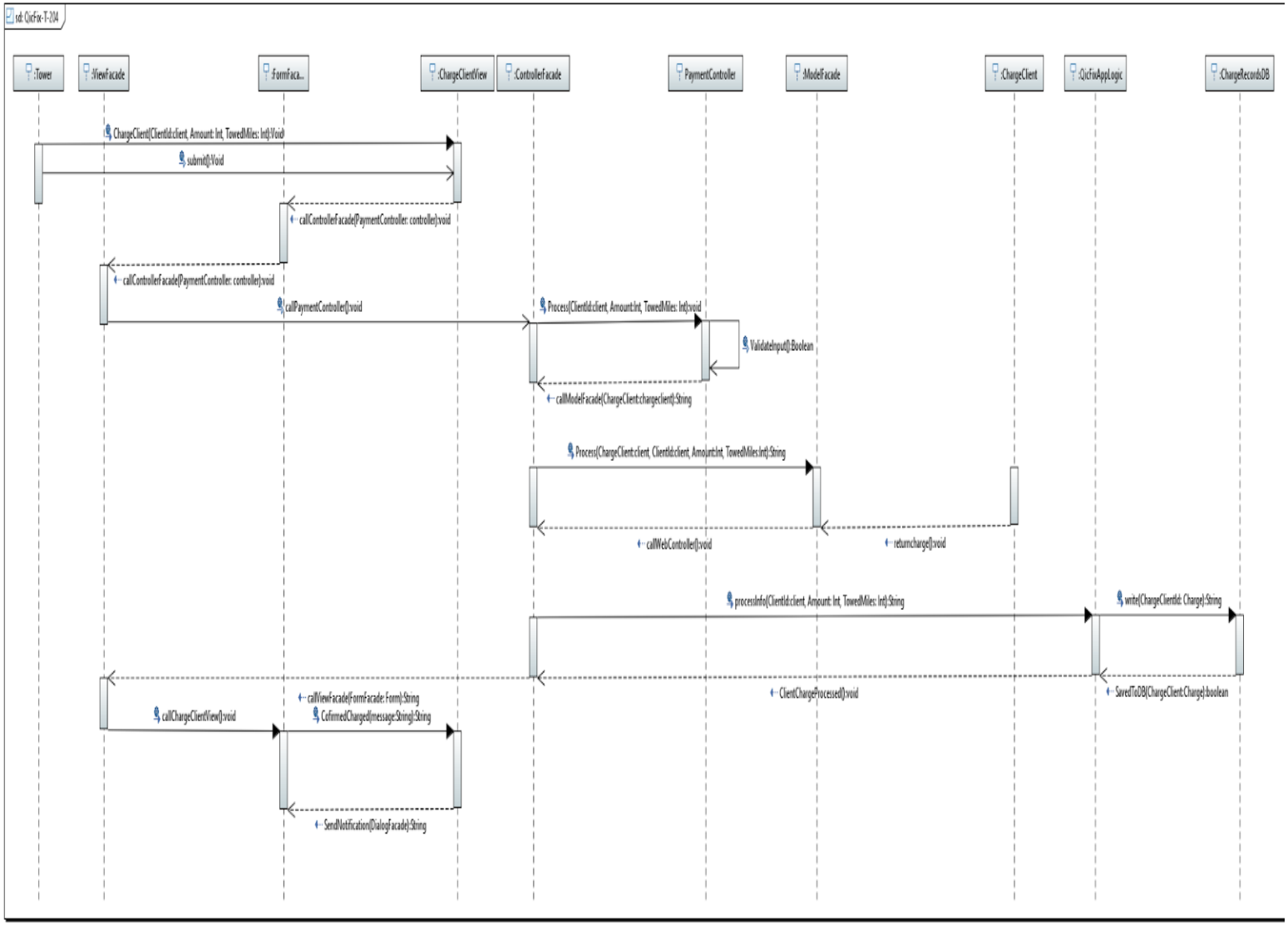


Figure 6.3.4: Sequence Diagram for QicFix-T-204-ChargeClient

5. QicFix-Sec-205-Session-Timeout:

This security use case allows auto logout if the user(Client/Tower) is inactive for more than 30 minutes. The use case is applicable for all users of the system. Any time the Controller gets a request from any user, it'll verify the user timer. If the gap between the current time and last activity time is greater than 20 minutes, the Controller logouts a user. Otherwise it'll update the activity timer with the current time.

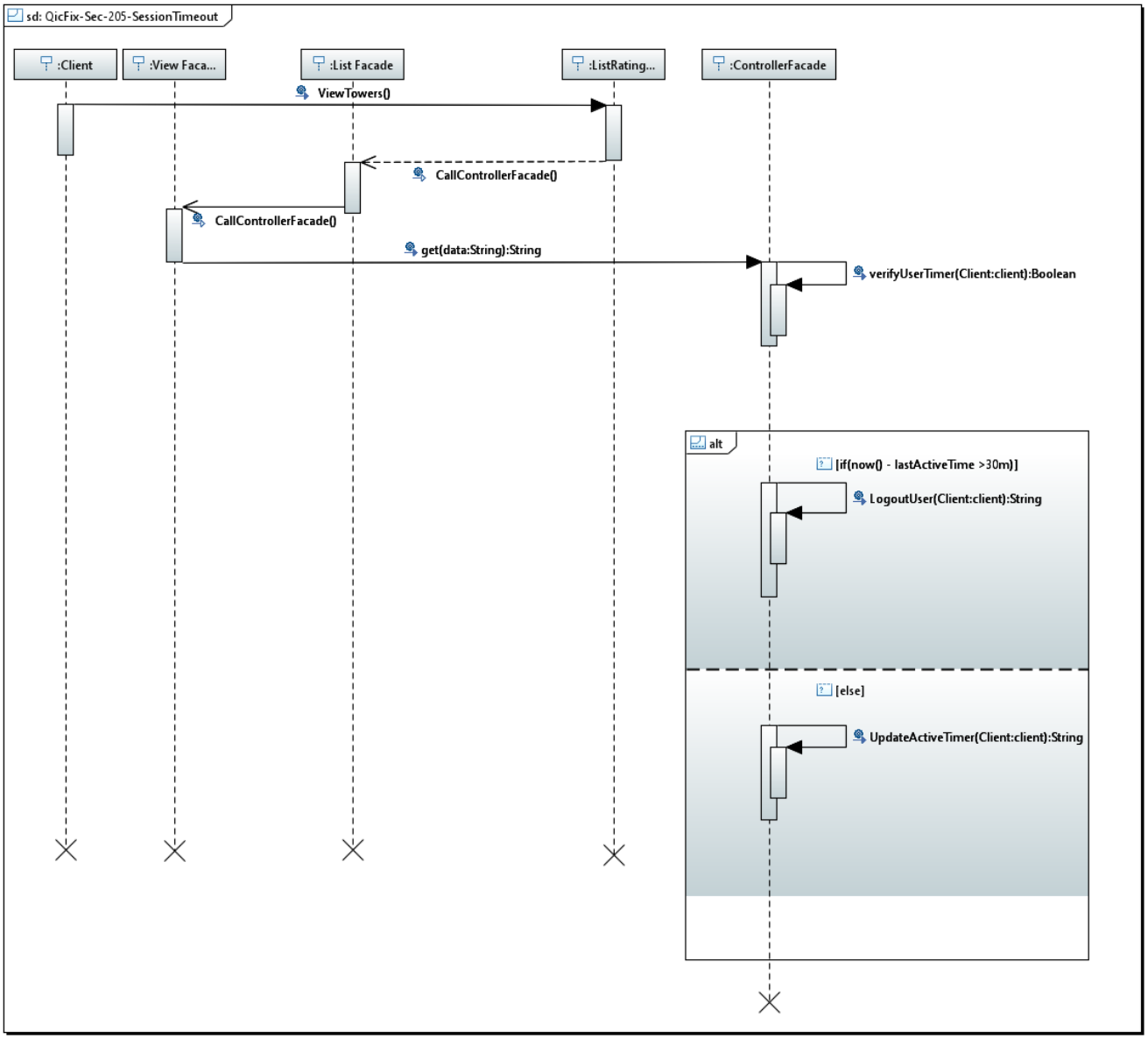


Figure 6.3.5: Sequence Diagram for QicFix-Sec-205-SessionTimeOut

6. QicFix-C-301-ChangePickup:

In the next sequence diagram (Figure 6.3.6) the client changes his or her pickup GPS location by keeping pressed the location icon by two seconds then the ControlFacade gives the user permission to move the icon around the map. After completing this action the user needs to press the button Confirm to confirm the new pickup location and the ControlFacade contacts the PresentationFacade and AppLogicFacade to send the new information to the StorageFaacade.

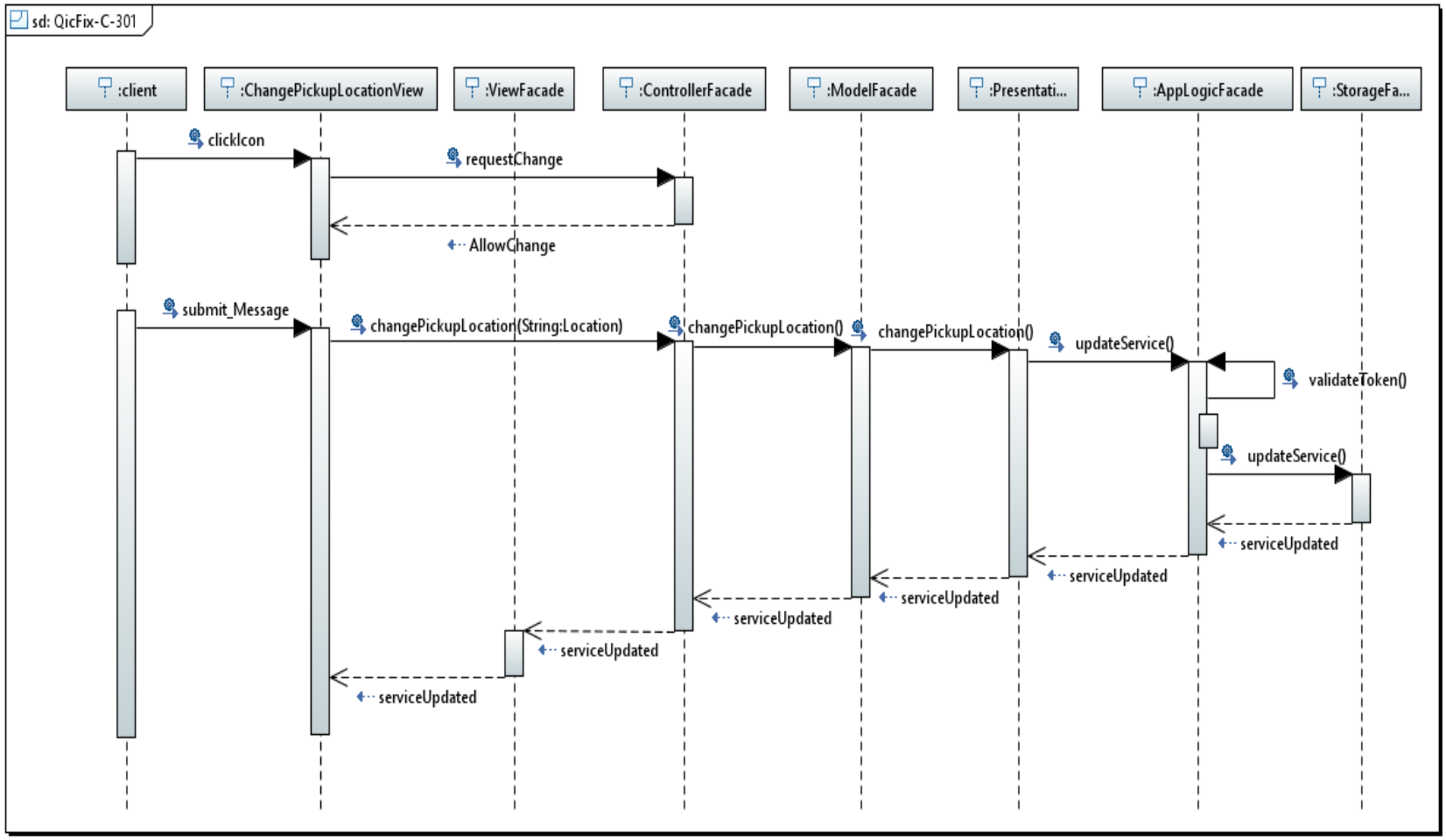


Figure 6.3.6: Sequence Diagram for Use Case QicFix-C-301-ChangePickup

7. QicFix-C-305-ChangeDestination :

In the next sequence diagram (Figure 6.3.7) the client changes his or her destination GPS location by keeping pressed the location icon by two seconds then the ControlFacade gives the user permission to move the icon around the map. After completing this action the user needs to press the button Confirm to confirm the new destination location and the ControlFacade contacts the PresentationFacade and AppLogicFacade to send the new information to the StorageFacade.

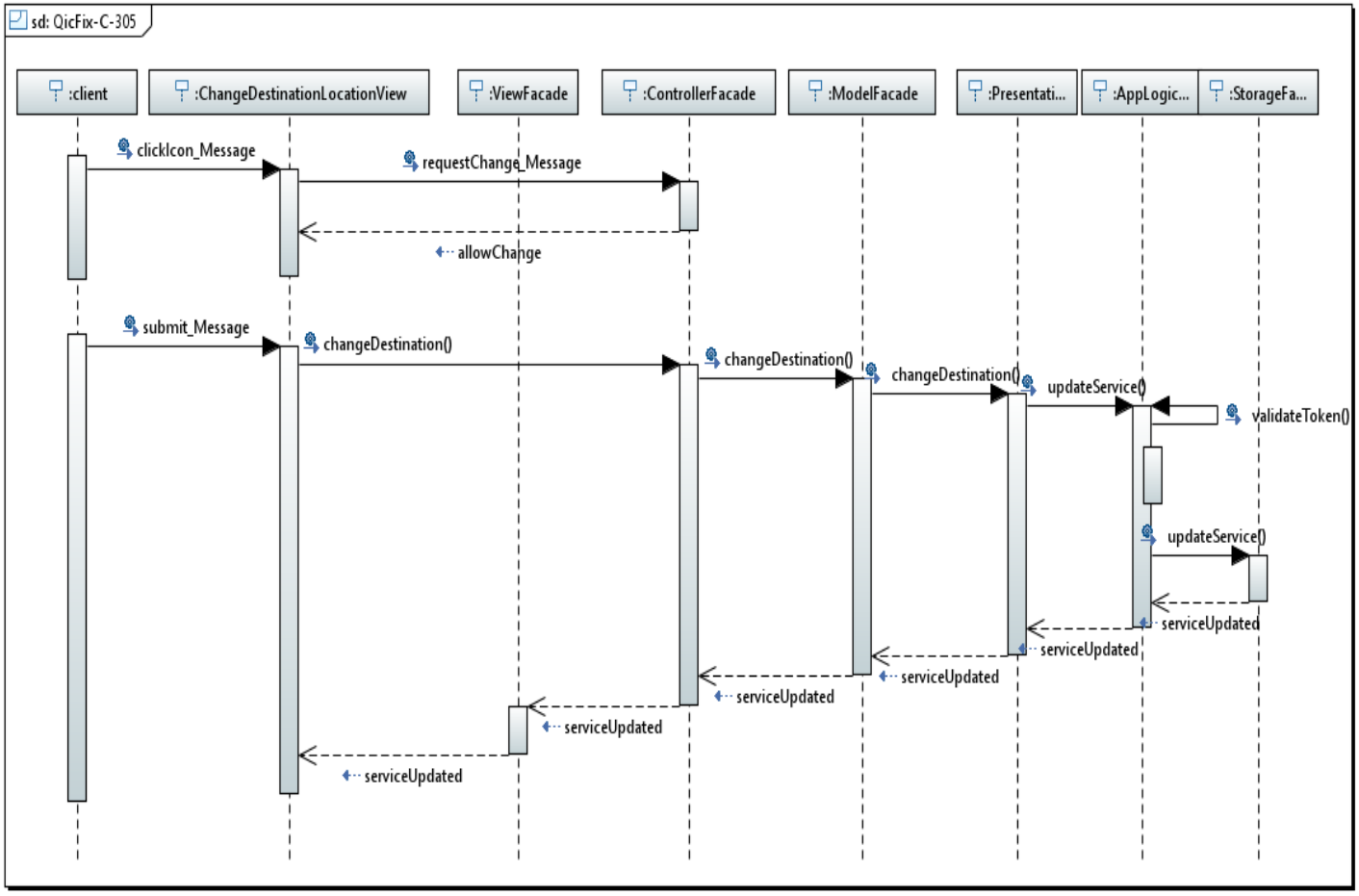


Figure 6.3.7: Sequence Diagram for Use Case QicFix-C-305-ChangeDestination

8. QicFix-C-301-TowerList:

In the next sequence diagram (Figure 3.3.13) the client opens the View ListTower to see a sorted list of the towers. The list can be sorted by:
1. Nearest tower regarding to the client's location, and
2. Best rated tower, rated by previous clients which have received service from each tower.

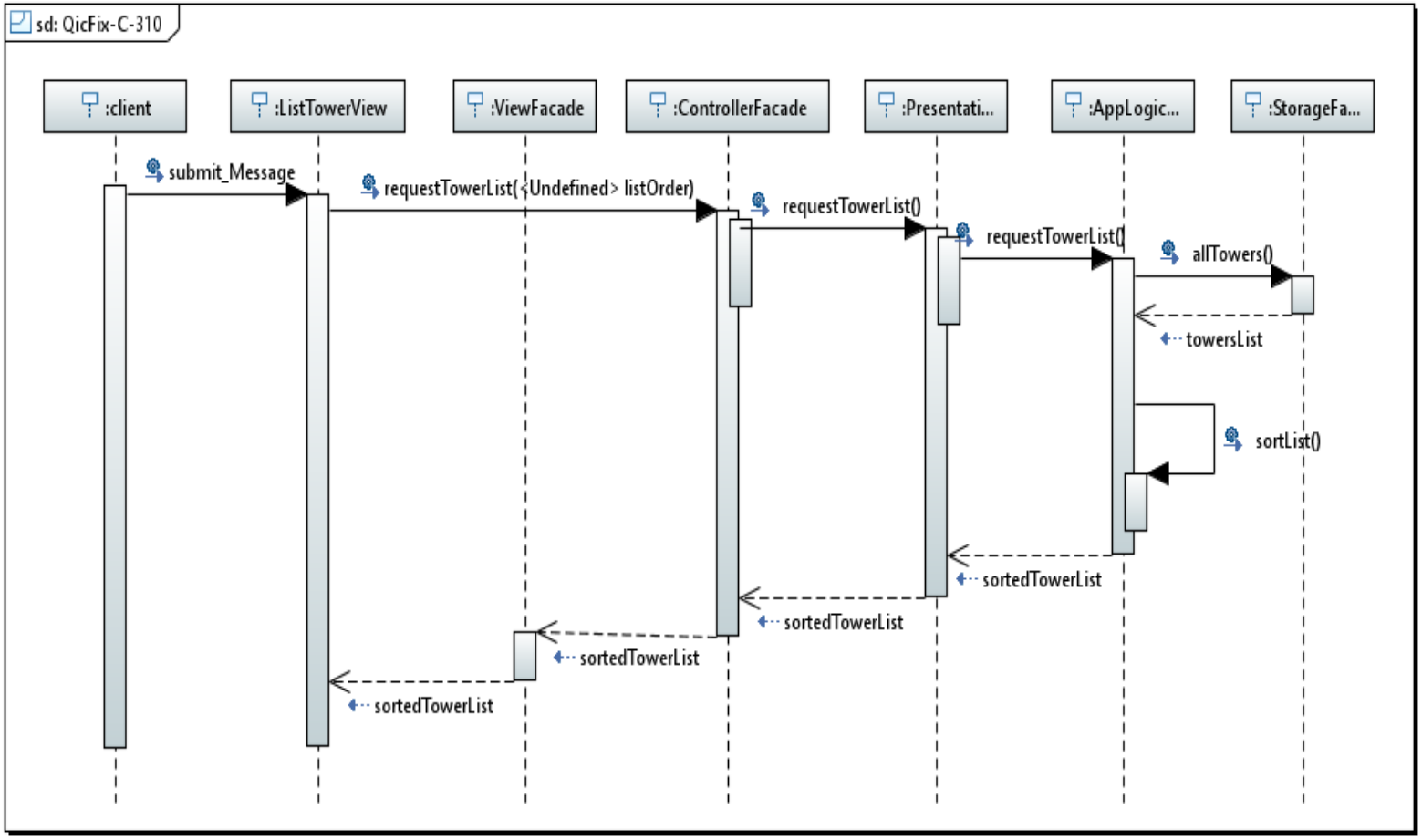


Figure 6.3.8: Sequence Diagram for the Use Case QicFix-C-310-TowerList

9. QicFix-C-101 RequestTower:

In the QicFix-C-101 sequence diagram the actor interacts with the view facade by entering their destination and location into the view. Then they hit the submit button and the view facade sends the information to the controller facade where a createRequest starts to make a request object. The request information is then sent to the model facade where a request object is made and then it's sent back to the controller facade. The controller facade sends the request to the PresentationFacade. The PresentationFacade contacts the AppLogicFacade with the request and the controller stores the service request in it's queue. A confirmation is sent back from the AppLogicFacade and PresentationFacade to the controller facade to confirm the request was received.

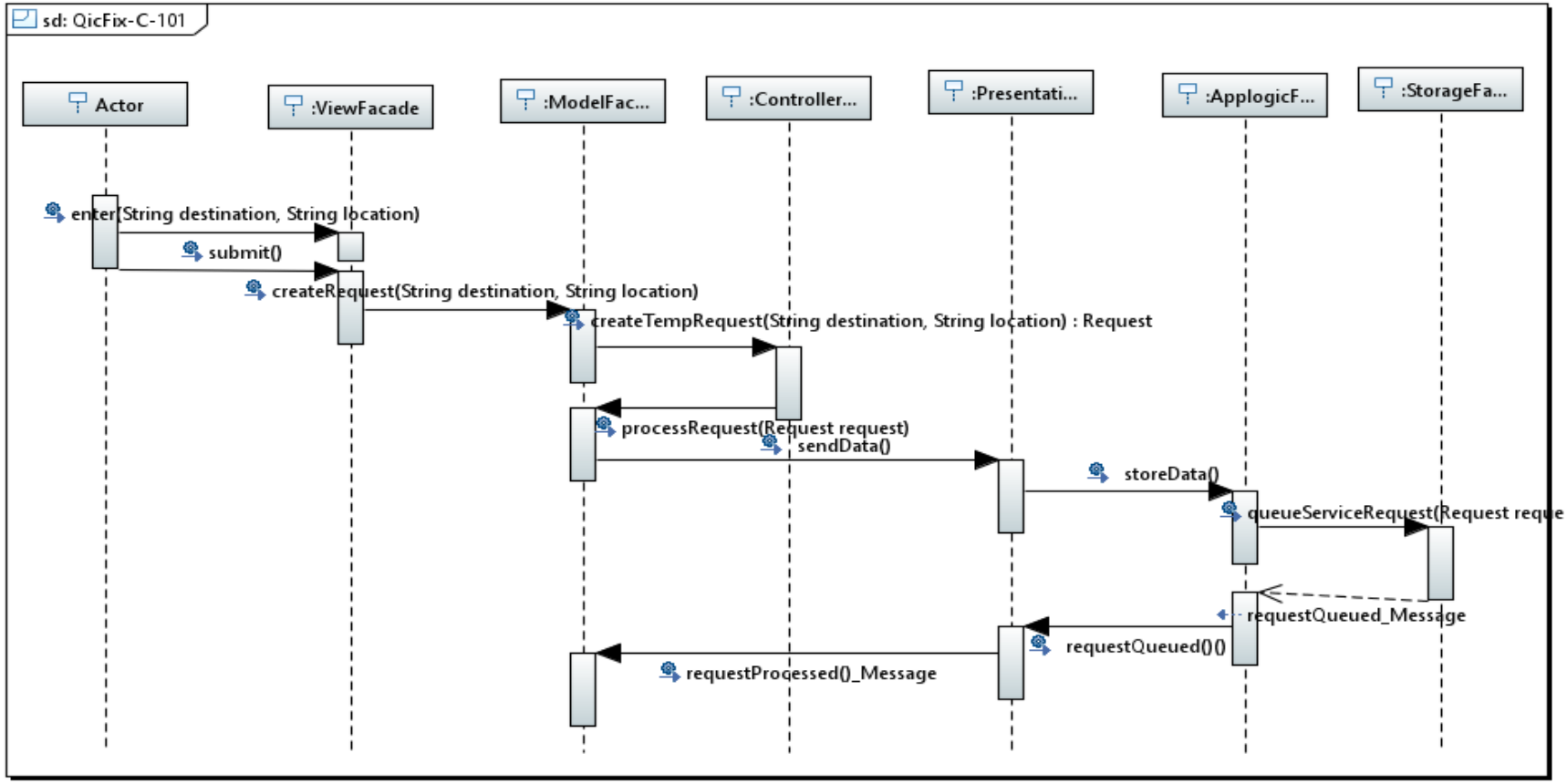


Figure 6.3.9: QicFix-C-101 Sequence Diagram

10. QicFix-C-102 PayTower:

In the QicFix-C-102 sequence diagram the actor must initiate paying the Tower by hitting the confirm button. Once they confirm the view facade contacts the controller facade about charging the client. The controller facade contacts the web service by sending it the client's information and towerId. Once inside the AppLogicFacade the client is charged and transaction is stored in the chargeClient. A receipt of the transaction is sent back from the PresentationFacade to the controller facade. The view facade displays the receipt of the transaction.

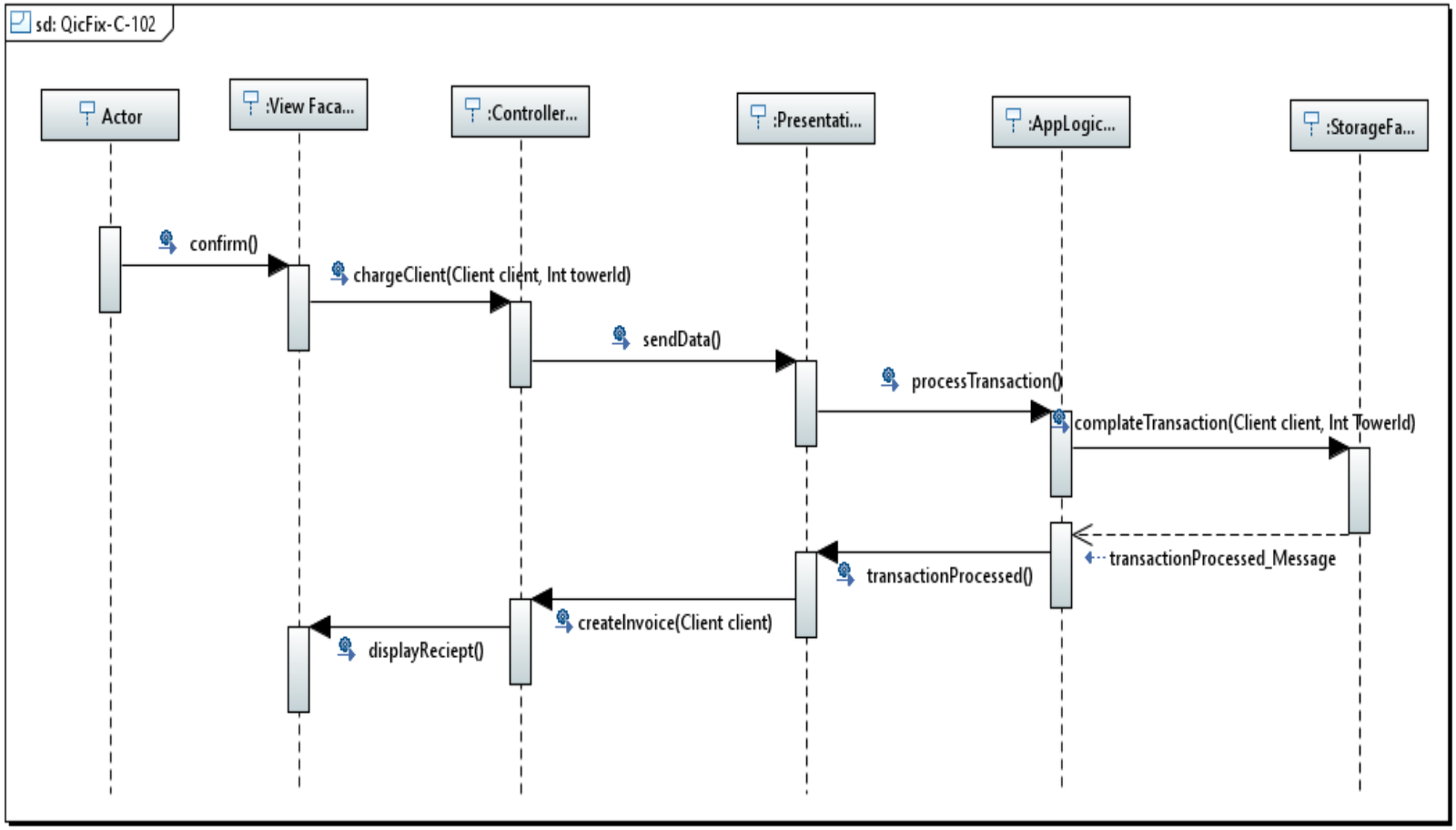


Figure 6.3.10: QicFix-C-102 Sequence Diagram

11. QicFix-C-103 ReviewTower :

In the QicFix-C-103 sequence diagram the Actor interacts with the view facade by sending it a rating and comment. Once the information is submitted the view facade contacts the controller facade to validate the information sent to it. The controller facade then goes through the model facade to create a review object. Once the review object is made the controller facade sends it to the PresentationFacade. The Presentation Façade stores the review with the service request and send confirmation back to the controller facade. The controller facade displays this confirmation to the Client on the view facade.

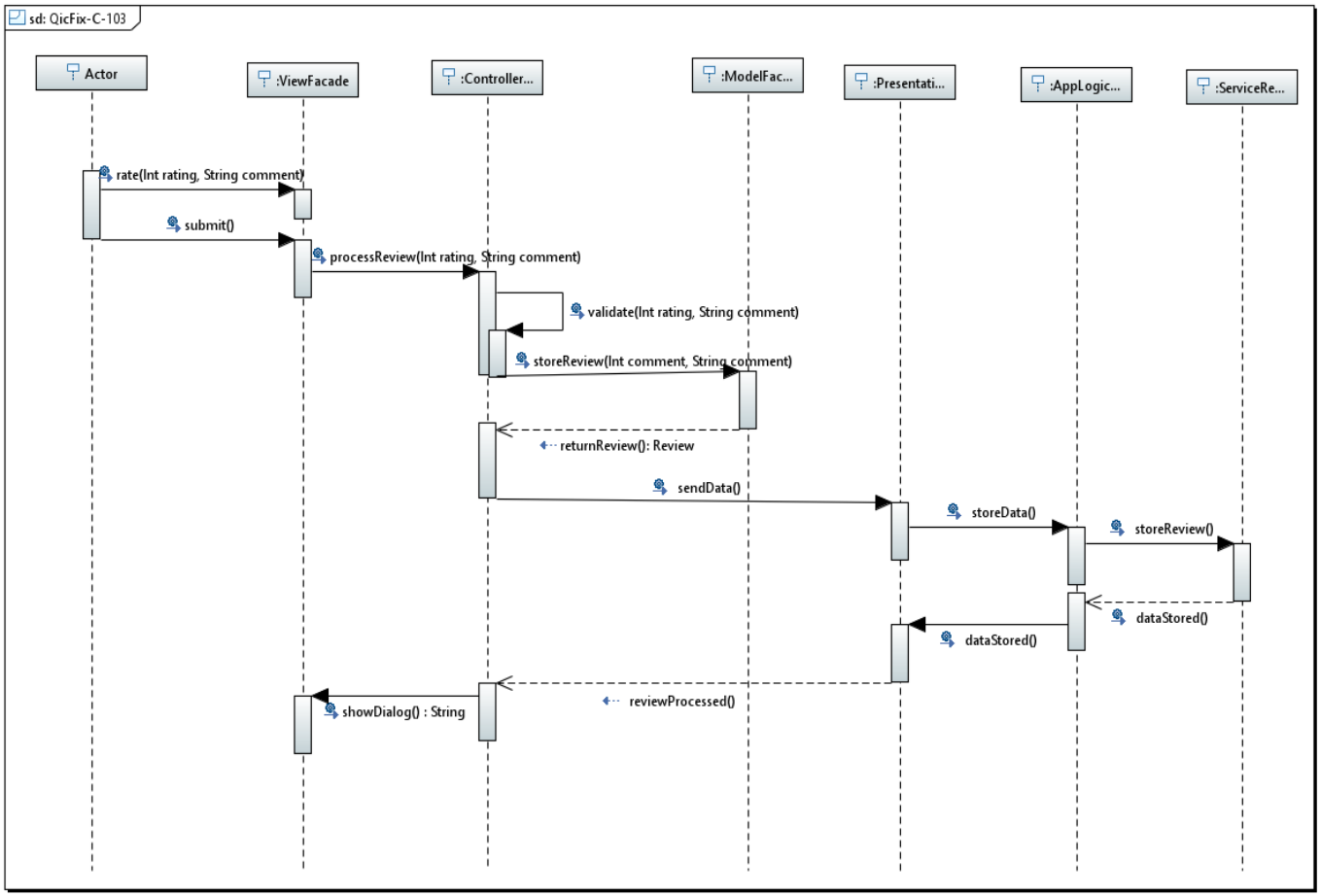


Figure 6.3.11: QicFix-C-103 Sequence Diagram

12. QicFix-C-105 LoginTimout:

In the QicFix-C-105 sequence diagram the Actor inputs an incorrect username and password into the view facade. They submit the information it is sent to the controller facade. The controller facade sends the information to the Presentation Facade. The AppLogicFacade verifies that the information is incorrect and sends a response back to the controller facade. If the Actor repeats this sequence five times then the controller facade sends an invalidateUser method to the PresentationFacade and a password reset email is sent to the actor.

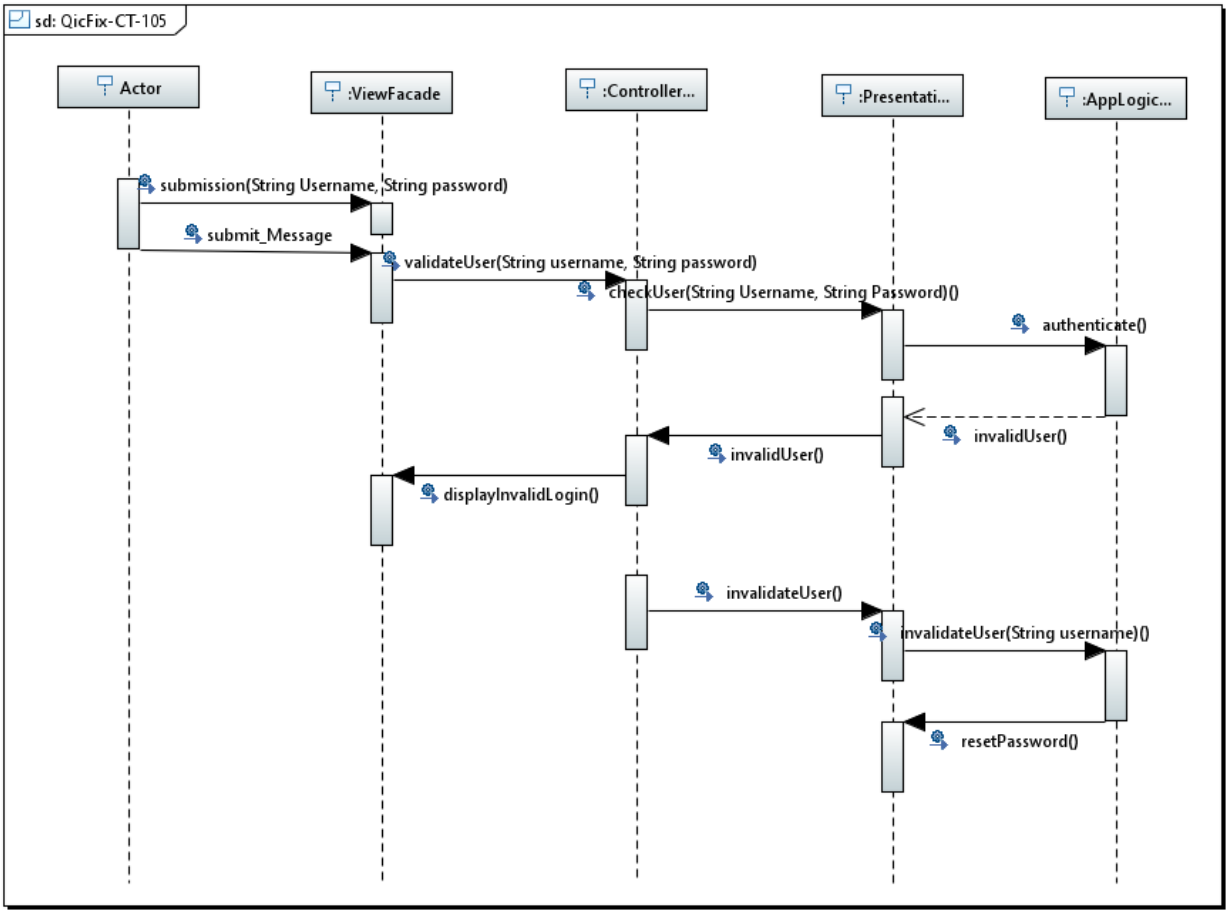


Figure 6.3.12: QicFix-CT-105 Sequence Diagram

13.QicFix-CT-401-UserLogin

In the QicFix-C-401 sequence diagram the actor interacts with the view facade by entering their email address and password into the view. Then they hit the submit button and the view facade sends the information to the controller facade where a createRequest starts to make a request object. The request information is then sent to the model facade where a request object is made and then it's sent back to the controller facade. The controller facade sends the request to the presentation layer. The presentation layer contacts the application logic layer with the request and the application logic validates the User information request in it's queue. A confirmation is sent back from the presentation layer to the application logic to confirm the request was received and then goes back to view and User interacts with the home view.

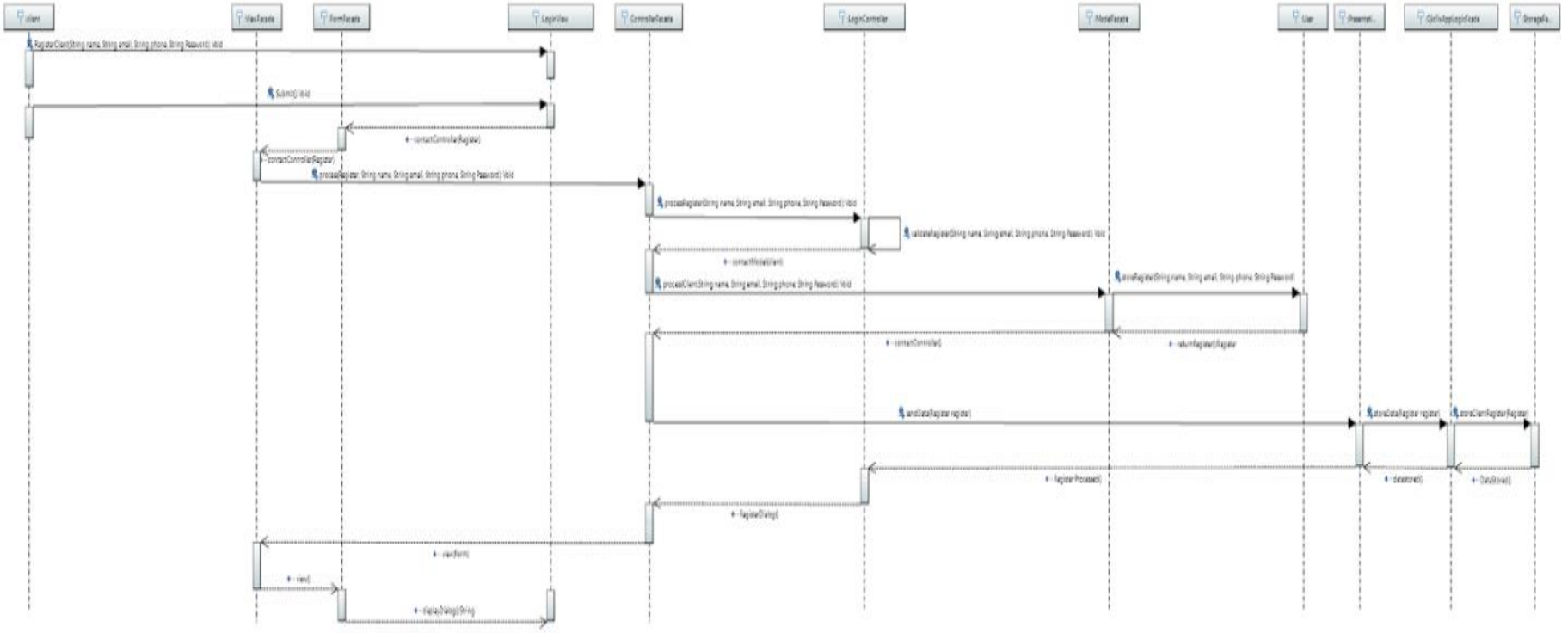


Figure 6.3.13: QicFix-CT-401 Sequence Diagram

14.QicFix-C-402-ClientRegistration :

In the QicFix-C-402 sequence diagram the actor interacts with the view facade by entering their name, phone number, email address and password into the view. Then they hit the submit button and the view facade sends the information to the controller facade where a createRequest starts to make a request object. The request information is then sent to the model facade where a request object is made and then it's sent back to the controller facade. The controller facade sends the request to the presentation layer. The presentation layer contacts the application logic layer with the request and the storage facade stores the Client information request in it's queue. A confirmation is sent back from the presentation layer to the controller facade to confirm the request was received and then goes back to view and client receive a message that Registration has been completed.

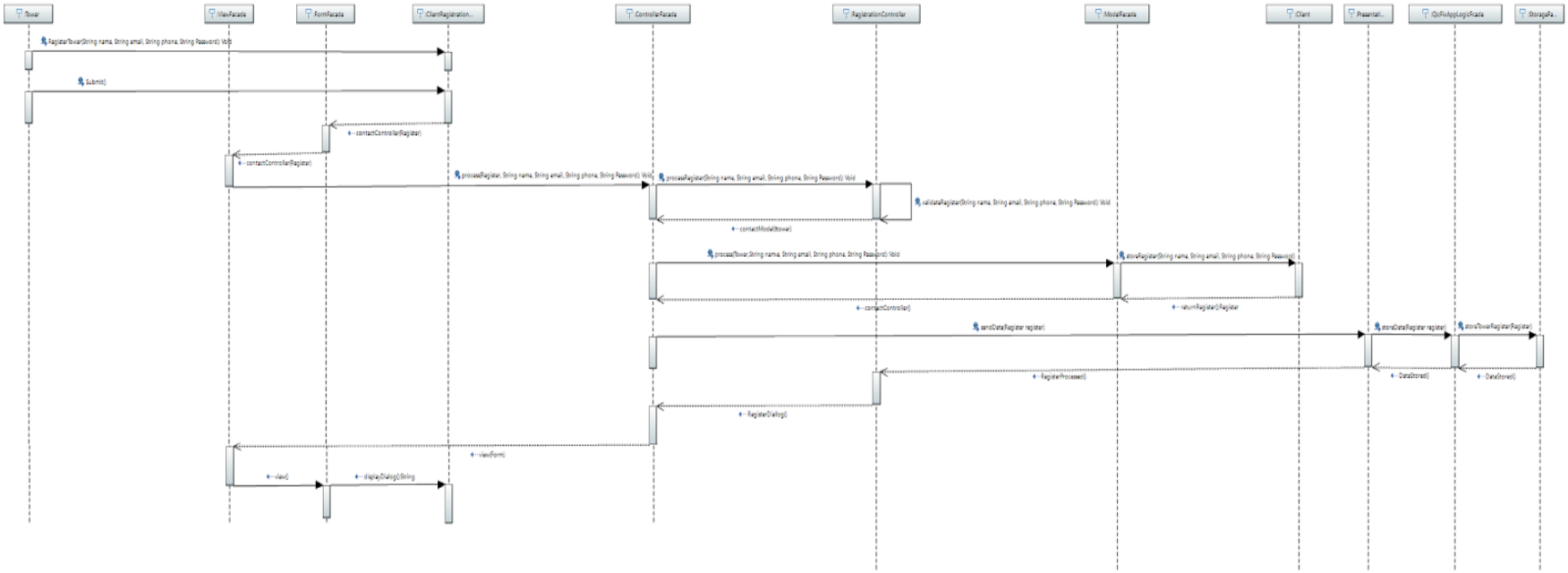


Figure 6.3.14: QicFix-C-402 Sequence Diagram

15.QicFix-T-403-TowerRegistration :

In the QicFix-C-403 sequence diagram the actor interacts with the view facade by entering their name, phone number, email address,password and TRAA certificate number into the view. Then they hit the submit button and the view facade sends the information to the controller facade where a createRequest starts to make a request object. The request information is then sent to the model facade where a request object is made and then it's sent back to the controller facade. The controller facade sends the request to the presentation layer. The presentation layer contacts the presentation layer controller with the request and the controller stores the Tower information request in it's queue. A confirmation is sent back from the presentation layer to the controller facade to confirm the request was received and then goes back to view and Tower receives a message that Registration has been completed.

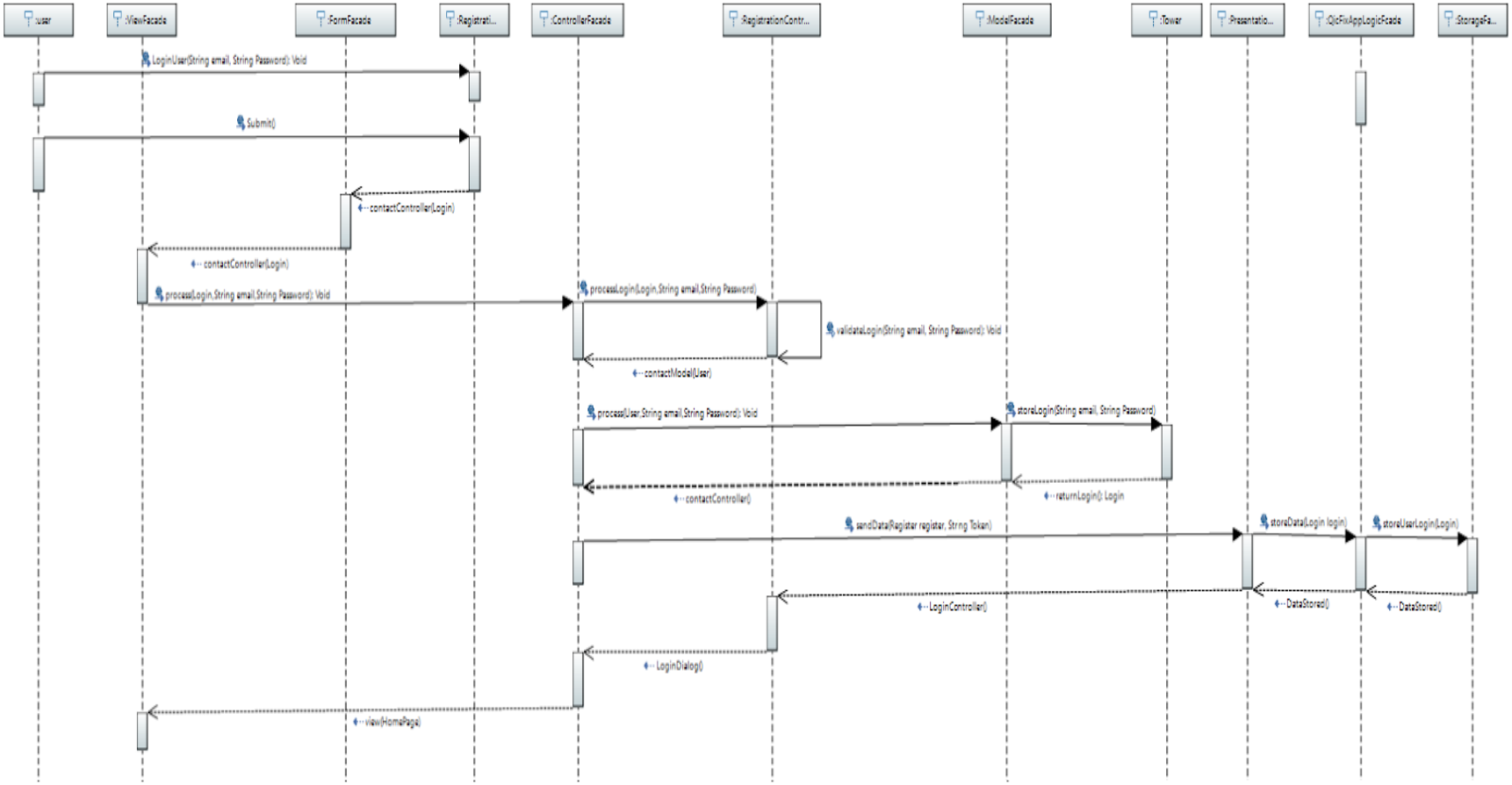


Figure 6.3.15: QicFix-T-403 Sequence Diagram

16.QicFix-CT-404-UserLogOut:

In the QicFix-C-404 sequence diagram the actor interacts with the view facade by hit the LogOut button and the view facade sends the information to the controller facade where a createRequest starts to make a request object. The request information is then sent to the model facade where a request object is made and then it's sent back to the controller facade. The controller facade sends the request to the presentation layer. The presentation layer contacts the application logic with the request and the controller deletes the User information from the queue. A confirmation is sent back from the presentation layer to the controller facade to confirm the request was received and then goes back to view and client has the view of LogOut page.

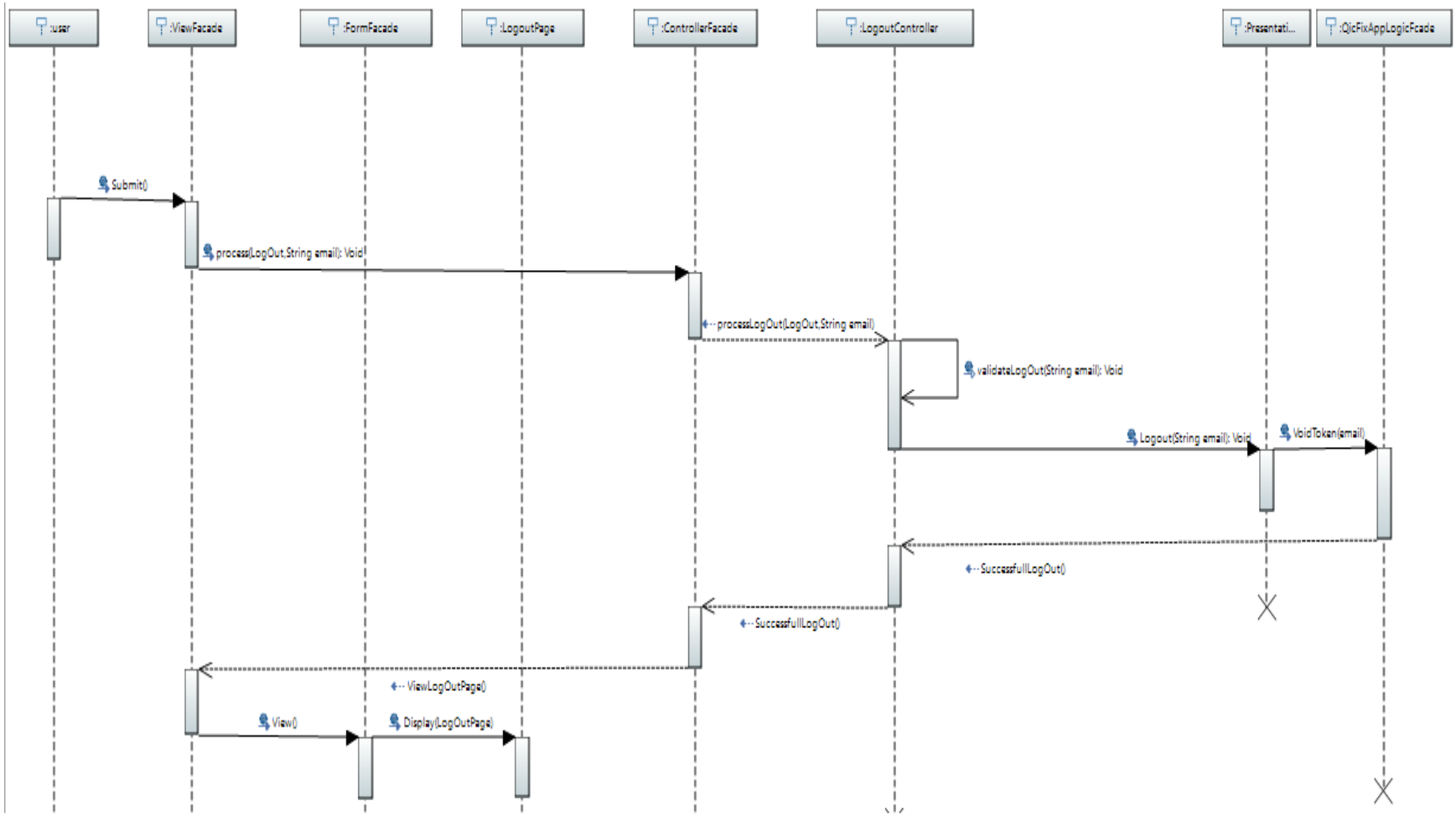


Figure 6.3.16: QicFix-CT-404 Sequence Diagram

6.4. Detailed Class Design:

In this section, each class along with its purpose is presented. For the control object in each major subsystem, constraints using Object Constraint Language are specified for the class invariants and for the pre-conditions and post-conditions for the methods in the class.

6.4.1. Class Overview:

The section is organized hierarchically by subsystems. Any design patterns that are used in the design are explicitly called out and reasoning is given to justify the use of each design pattern.

Client.Model Subsystem:

For detailed information regarding the following classes, refer to Appendix D Figure 11.4.1.

- **Model Façade:** Responsible for decoupling the Model subsystem from the Controller and View subsystem. Provides a single interface for accessing various domain services. Follows the Façade design pattern in order to provide a unified interface to a set of interfaces in the Controller and View subsystem, in turn simplifying the interaction between the Controller Model and View subsystems.

- **Client**: Represents a domain model for a Client Represents a domain model for a user. Contains properties for session key, email address, city and state, hashed password, maximum logon failures, logon failure counter, and first logon failure timestamp everything included in the list of array i.e. Profile.

- **Tower:** Represents a domain model for a Tower. Represents a domain model for a specialized employee user. Contains properties for role, first name, last name, phone number, email address, etc. in Profile Array.

- **Service**: Responsible for providing the ability to request service and cancel service from the Tower for the job needed to be done. It has properties like TowerId, Current Location and Destination.

- **ChargeClient:** Responsible for providing the ability to Charge the Client after the Job has been done by the Tower. It has the Properties such as Amount, ClientId which retrieves all the relevant information required and TowerId which retrieves all the information related to the tower.

- **Review:** Responsible for providing the ability to the Client about the Tower Ratings and Comments given by other Clients upon the completion of the job. Review class has the properties like Rate, Comment, ClientId and TowerId.

Client.Controller Subsystem:

For detailed information regarding the following classes, refer to Appendix D Figure 11.4.2.

- **Controller Facade:** The main control object of the presentation subsystem. Responsible for accepting HTTP requests and coordinating communication to the Business Logic subsystem and creation and serialization of view model for each specific device. Provides endpoints for authentication, getting data, posting data, session management, data serialization and deserialization, and for changing views for a particular client.

- **LoginController:** The control object of the LoginView through the controller façade in the Form View Subsystem.

- **RegistrationController:** The control object of the Registration View in the Form View Subsystem.

- **RequestServiceController:** The control object of the Requesting Service for the Service Model in the Model Subsystem.

- **EditProfileController:** The control object of the EditProfileView in the Form View Subsystem.

- **Accept/DeclineServiceController:** The control object of the Accept/Decline notification View in the Dialog View Subsystem.

- **NotificationController:** The control object of the Notification View in the Dialog View Subsystem.

- **ChargeClientController:** The control object of the Charge Client View in the Form View Subsystem.

- **ListTowerController:** The control object of the ListTower View in the List View Subsystem for listing the Tower for requesting the Job.

- **LocationController:** The control object of the InputAddress View where we need to delegate the Googlemap.lib for retrieving the location.

- **ViewRatingController:** The control object of the List the Rating of the Tower given by the Client in the List View Subsystem.

- **RatandCommentController:** The control object of the Rate and Comment View which provides details to the class Rate and Comment in Dialog View Subsystem.

Client.View.ListView Subsystem:

For detailed information regarding the following classes, refer to Appendix D Figure 11.4.3 and Figure 11.4.4.

- **ListFacade:** Responsible for decoupling the List View subsystem from the Form View and Dialog View subsystem. Provides a single interface for accessing various domain services. Follows the Façade Design pattern and Strategy Design Pattern.

- **ListTowerView:** An abstract class which follows the strategy design pattern. Responsible for defining a contract for any view strategy class. A view strategy defines a means of mapping from a low-level system entity to an appropriate view model. Responsible for creating the List of Towers by applying the algorithm of sorting.

- **BestTowerView:** Responsible for creating a view model containing Best Tower List with its information providing List of array. Follows the strategy design pattern.

- **CheapestTowerView:** Responsible for creating a view model containing Cheapest Tower List with its information providing List of array. Follows the strategy design pattern.

- **ClosestTowerView:** Responsible for creating a view model containing Closest Tower List with its information providing List of array. Follows the strategy design pattern.

- **ListRatingView:** Represents compact and filtered data regarding the rating and comments for the Tower given by the Client.

Client.View.FormView Subsystem:

For detailed information regarding the following classes, refer to Appendix D Figure 11.4.4.

- **FormFacade:** Responsible for decoupling the Form View subsystem from the List View and Dialog View subsystem. Provides a single interface for accessing various domain services. Follows the Façade Design pattern, Strategy Design Pattern and Singleton Design Pattern.

- **LoginView:** Represents the Login View to the Client/Tower. Input for the login has the properties like Username and Password. Follows the Singleton Design Pattern.

- **RegistrationView:** An abstract class which follows the strategy design pattern. Responsible for defining a contract for any view strategy class. A view strategy defines a means of mapping from a low-level system entity to an appropriate view model. Contains a single abstract method (createViewModel) responsible for enforcing a view model

creation algorithm to be implemented by all subclasses. Responsible for Notifying the client about different things.

- **ClientRegistrationView:** Responsible for creating a view model containing the Registration Page for the Client. Containing the properties like First Name, Last Name, Phone Number, Email id, and Card Details. Follows the strategy design pattern.

- **TowerRegistrationView:** Responsible for creating a view model containing the Registration Page for the Tower. Containing the properties like First Name, Last Name, Phone Number, Email id, Card Details, Towing Miles, Towing Rates and Tower Address. Follows the strategy design pattern.

- **ChargeClientView:** Responsible for creating a view model containing the amount Area towed and Client name to be charged for the job completed by the Tower for the Client.

- **InputAddressView:** Responsible for the creating the View model for giving the location for the job to be done by the Tower. It contains the property like Current Location and Destination which further coordinates with the Controller to retrieve the location using the googlemap.lib.

- **EditProfileView:** Represents the view in a compact and filtered way for editing the profile for Client/Tower. It contains all the information which is given at the time of creating the profile. It updates the information of the Client or Tower and saves the information to the database.

Client.View.DialogView Subsystem:

For detailed information regarding the following classes, refer to Appendix D Figure 11.4.5

- **Dialog Façade:** Responsible for decoupling the Dialog View subsystem from the Form View and List View subsystem. Provides a single interface for accessing various domain services. Follows the Façade design pattern and Strategy Design Pattern.

- **RatingandCommentsView:** Represents the view of rating and comments for the Tower by the Client after job has been done. It contains the Rate field and Comment Field with the TowerId associated.

- **JobRequestView:** Represents the view of Requesting Job to the Tower selected by the Client. It contains attributes like TowerId, Current Location and Destination for the job to be done.

- **NotificationView:** An abstract class which follows the strategy design pattern. Responsible for defining a contract for any view strategy class. A view strategy defines a means of mapping from a low-level system entity to an appropriate view model. Responsible for Notifying the client about different things. Responsible for providing the ability to retrieve, update, create information regarding a notification. Specific to our implemented use cases, provides functionality to retrieve notifications and mark notifications as seen.

- **Re-RequestTowerView:** Represents a domain model for user-specific notification information. Includes information such as re-requesting tower as the request was declined by the client for each notification belonging to a re-request event. Contains properties for sent time and user for which the notification belongs to. Follow the Strategy Design Pattern.

- **DeclineRequestNotifyView:** Represents compact and filtered notification for the Decline request by the tower as it will be shown to a Client. It contains property of TowerId. Follow the Strategy Design Pattern.

- **AcceptRequestNotifyView:** Represents compact and filtered notification for the Accept request by the tower as it will be shown to a Client. It notifies the Client about Tower name who is going to arrive and in how much time is going to arrive if requested is accepted. Follow the Strategy Design Pattern.

- **ChargeNotificationView:** Represents compact and filtered notification for the Charge needed to be paid for the job done by the tower as it will be shown to a Client with Amount and the TowerId.

Presentation Subsystem:

For detailed information regarding the following classes, refer to Appendix D Figure 11.4.7.

- **Presentation Façade:** Responsible for decoupling the Presentation subsystem from the Application Logic and the Client subsystem. Provides a single interface for accessing various domain services. Follows the Façade design pattern and Command Design Pattern.

- **API Command:** The main control object of the presentation subsystem. Responsible for accepting HTTP requests and coordinating communication to the Application Logic subsystem and creation and serialization of view model for each specific device. Provides endpoints for authentication, getting data, posting data, session management, data serialization and deserialization, and for changing views for a particular client.

- **Get Command:** Requests data from a specified resource such as Requesting Tower List for Requesting Service.

- **Put Command:** Submits data to be processed to a specified resource.E.g.It uploads information passed from Client from QicFix system to a server.

- **Post Command:** Submits data to be processed to a specified resource.

- **Delete Command:** Deletes the specified resource.

Application Logic Subsystem:

For detailed information regarding the following classes, refer to Appendix D Figure 11.4.8

- **App Logic Facade** – Responsible for decoupling the Application Logic subsystem from the Presentation subsystem. Provides a single interface for accessing various domain services. Follows the Façade design pattern in order to provide a unified interface to a set of interfaces in the Application Logic subsystem, in turn simplifying the interaction between the Presentation and Application Logic subsystems.

- **Registration Logic: Request Service Logic:** Responsible for creating a request from Client Side and storing it into the StorageFacade.

- **Edit Profile Logic:** Responsible for Updating the Profile of the Tower or Client and passing it to the Storage Facade in form of Content return from the JSON.

- **List Tower Logic:** Responsible for Listing Tower Logic from the Presentation and passing it to the Storage facade and giving back the List of tower retrieved from Storage facade to Presentation Facade.

- **Login Logic:** Responsible for the Login Logic of the QicFix System for the Tower and Client.

- **Logout Logic:** Responsible for the Logout Logic of QicFix System for Tower and Client.

- **Payment Logic:** Responsible for the Payment and Charging Client Logic of the QicFix System.

- **Location Logic:** Responsible for interacting with the Google API for retrieving Client Pickup Location giving Destination for the service requested and for retrieving the Tower Location and tracking the route of QicFix System.

Storage Subsystem:

For detailed information regarding the following classes, refer to Appendix D Figure 11.4.9 and Appendix D.

- **Storage Façade:** Responsible for decoupling the Storage subsystem from the Application Logic subsystem. Provides a single interface for accessing various data-related operations. Follows the Façade design pattern in order to provide a unified interface to a set of interfaces in the Storage subsystem, in turn simplifying the interaction between the Application Logic and Storage subsystems. Follows the singleton design pattern to ensure the Storage Façade class has only one instance and to provide a global point of access to it. The reason for using the singleton design pattern is due to this class having several complex dependencies that render the class very expensive to instantiate.

- **Client Data**: Represents a domain model for a user. Contains properties for role, first name, last name, and Address, Phone number, Date of Birth.

- **Tower Data:** Represents a domain model for a user. Contains properties for session key, email, address, city, state, Company name, Permit number, maximum logon failures, logon failure counter, and first logon failure timestamp.

- **Profiles:** Provides data access to profiles of the Client and Tower.

- **Request Service:** Represents a domain model for user-specific request made for the service and for the provided service information.

- **Payment Data:** Stores the Data related to the Charges made for the service provided by the Tower to the Client. Includes card details, Client id, Tower id, Rates, Distance Towed.

7. Glossary:

- **Tower** - a person who drives a tow truck.
- **GPS** - Global Positioning system.
- **Android** - a mobile operating system based on the Linux Kernel and currently developed by Google.
- **API** - Application Program Interface.
- **Request** - An instruction to a computer to provide information or perform another function.
- **Server** - a computer program or a device that provides functionality for other programs or devices, called "clients".
- **Timeout** - a cancellation or cessation that automatically occurs when a predefined interval of time has passed without a certain event occurring.
- **Class Diagram -** A model representing the different classes within an s/w system and how they interact with each other.
- **Component** - A physical and replaceable part of a system that conforms to and provides the realization of a set of interfaces.
- **Model** - An abstract representation of a system that enables us to answer questions about the system.
- **Post-condition** - A predicate that must be true after an operation is invoked.
- **Precondition** - A predicate that must be true before an operation is invoked.
- **Sequence Diagram** - A model representing the different objects and/or subsystems of a software project and how they relate to each other during different operations for a given use case.
- **State Machine Diagram** - A model representing the states and transitions of a particular system or control object

- **Deployment Diagram** - A model representing the hardware and software mapping of a particular software system.

- **Entity Relationship Diagram** - A model representing the persistent entities and relationships amongst entities to be used by a software system.

- **Unified Modeling Language (UML)** - A standard set of notations for representing models.

- **Use Case** - A general sequence of events that defines all possible actions between one or many actors and the system for a given piece of functionality.

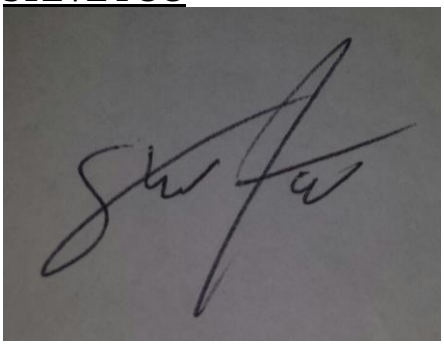
- **API -** Application Programming Interface. A set of routines, protocols, and tools for building software applications.

- **MySQL -** An implementation of a Structured Query Language Database Management System.

- **HTTP -** The Hypertext Transfer Protocol is an application protocol for distributed, collaborative, hypermedia information systems. HTTP is the foundation of data communication for the World Wide Web.

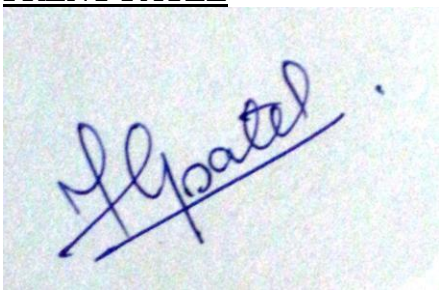
8. Signatures:

STEVE FOO

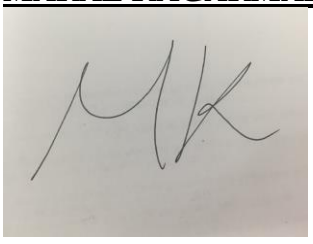


JUAN SOTOMAYOR

FRENY PATEL



MARAL KAGARMAKHAR



9. References:

1. Bernd Bruegge, Allen H Dutoit, “Object-Oriented Software Engineering: Using UML, Patterns, and Java”, 3rd Edition, Prentice Hall, ISBN 9780136061250
2. Martin, Robert C. (2000). "Design Principles and Design Patterns" (PDF).
3. Microsoft. “Architectural Patterns and Styles” http://msdn.microsoft.com/en-us/library/ee658117.aspx
4. "UML 2.0". Omg.org.

10. Appendix:

10.1. Appendix A – Project Schedule:

ID	Nombre de tarea	Durati	Start	Finish
1	**QicFix System**	**71 day**	**Tue 1/19/16**	**Tue 4/26/16**
2	**Analysis**	**26 day**	**Tue 1/19/16**	**Tue 2/23/16**
3	Use Cases	5 days	Tue 1/19/16	Mon 1/25/1
4	Class Diagrams	10 day	Tue 1/26/16	Mon 2/8/16
5	Sequence Diagrams	5 days	Tue 1/26/16	Mon 2/1/16
6	Create Scenarios	5 days	Tue 1/26/16	Mon 2/1/16
7	Object Diagrams	5 days	Tue 2/2/16	Mon 2/8/16
8	Cost Estimation (Coc	3 days	Tue 1/26/16	Thu 1/28/16
9	Security Use Case	5 days	Tue 1/19/16	Mon 1/25/1
10	Draft of SRD deliver	10 day	Tue 2/9/16	Mon 2/22/1
11	Presentation and turn in SRD deliverable	1 day	Tue 2/23/16	Tue 2/23/16
12	**Design**	**20 day**	**Wed 2/24/1**	**Tue 3/22/16**
13	Software Arquitecture Design	3 days	Wed 2/24/16	Fri 2/26/16
14	Class Design	2 days	Mon 2/29/1	Tue 3/1/16
15	Subsystem Decomposition	1 day	Mon 2/29/16	Mon 2/29/16
16	State Machine Diagr	2 days	Mon 2/29/1	Tue 3/1/16
17	Persistance Data Management	3 days	Wed 3/2/16	Fri 3/4/16
18	Hardware & Software Mapping	1 day	Mon 3/7/16	Mon 3/7/16
19	Draft of SDD deliver	10 day	Tue 3/8/16	Mon 3/21/1
20	Presentation and turn in SDD	1 day	Tue 3/22/16	Tue 3/22/16
21	**Deployment**	**12 day**	**Wed 3/23/1**	**Thu 4/7/16**
22	Deployment of Serv	10 day	Wed 3/23/1	Tue 4/5/16
23	Deployment of Webservices	10 days	Wed 3/23/16	Tue 4/5/16
24	Deploiyment of Wet	10 day	Wed 3/23/1	Tue 4/5/16
25	Deployment of Mobile Application	10 days	Wed 3/23/16	Tue 4/5/16
26	Unit testing	2 days	Wed 4/6/16	Thu 4/7/16
27	**Implementation**	**11 day**	**Fri 4/8/16**	**Fri 4/22/16**
28	Server initial configuration	3 days	Fri 4/8/16	Tue 4/12/16
29	Startup of services	3 days	Wed 4/13/1	Fri 4/15/16
30	User Documentation	5 days	Mon 4/18/1	Fri 4/22/16
31	System Documentati	5 days	Mon 4/18/1	Fri 4/22/16
32	Publication of Mobil	1 day	Mon 4/18/1	Mon 4/18/1
33	**Test**	**6 days**	**Tue 4/19/16**	**Tue 4/26/16**
34	System testing	1 day	Tue 4/19/16	Tue 4/19/16
35	Subsystem testing	2 days	Wed 4/20/1	Thu 4/21/16
36	Unit testing	2 days	Wed 4/20/1	Thu 4/21/16
37	Developing test suit	1 day	Fri 4/22/16	Fri 4/22/16
38	Draft of SDD deliver	1 day	Mon 4/25/1	Mon 4/25/1
39	Presentation and turn in SDD	1 day	Tue 4/26/16	Tue 4/26/16

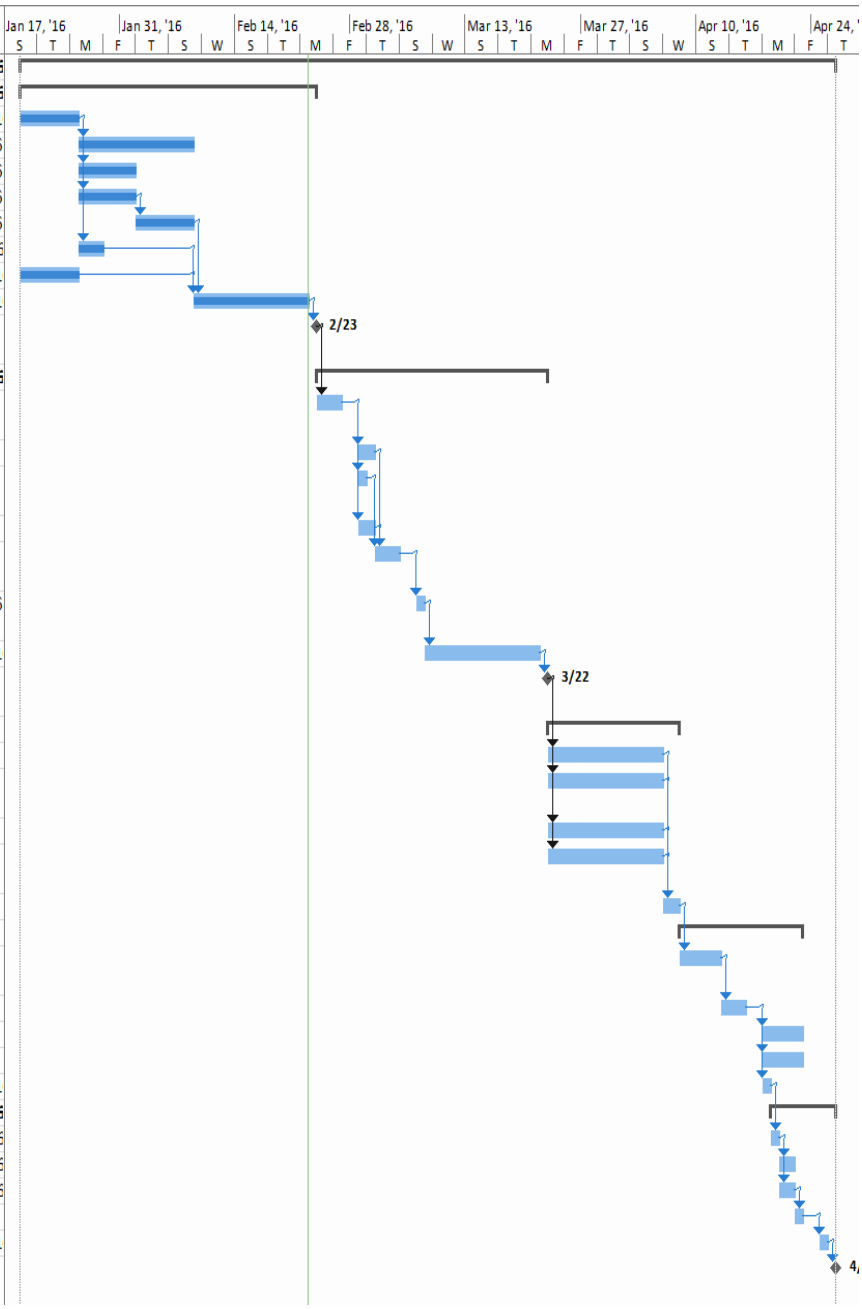


Figure 11.1.Gantt Chart

10.2. Appendix B – Use Cases:

This section presents all of the uses cases that are to be implemented in the QicFix App.

11.2.1 Edit Profile:

Use Case ID: QicFix-T-201.

Use Case Title: Edit Profile.

Use Case Level: System end-to-end level.

Details: The System shall allow Tower to edit its profile.

- **Actor**: Tower

- **Pre-conditions**: The System is functioning properly, User is already logged in to the system and the profile is already stored in the profile database.

- Description:

  Trigger: The user initiates an action by clicking the Edit Profile button on the Main Menu Screen.

  1. The system responds by opening the profile with current information stored in the database.
  2. The System displays the edit screen with the user's information.
  3. The User updates the data field accordingly, and submits the information to the System.
  4. The User clicks the 'Save' Button.
  5. The System reloads the page with the updated profile,

- **Relevant requirements:** None

- **Post-conditions**:

  1. **Successful Completion**: If the use-case ends in success,the user's profile data is updated.
  2. **Failure Condition**: If there is failure, User's data is not changed.

Alternative Courses of Action:

1. At Step-2, System displays error message if user submits a field with blank input.
2. At Step-2, System displays error message if confirm password does not match new password.
3. At Step-1,2 ,User clicks cancel button to cancel account changes.

Extensions: {Change Profile}, {Validate Information}

Exceptions: If Internet Connection gets lost, Request Time-out.

Concurrent Uses: None

Decision Support:
Frequency: Not more frequently. Can occur at most once per application run.

Criticality: Moderately Important. Not needed to run the program, but is necessary for the system to maintain the user records.

Risk: Low

Constraints:

- Usability: Must be intuitive and easy to use. Less than 20 seconds to perform task.
- Reliability: Must be very reliable for both the client and tower to update the profile. Mean time for failure: 1 out of every 300 executions fails.
- Performance: Speed is important for this operations and the system should complete the operation within less than 1 second.
- Supportability: Will be supported on web browsers and Android phones.
- Implementation: In java and in Android Studio.

Modification History
Owner: Freny Patel
Date last modified: 03-14-2016

11.2.2. View Rating and Comments:

Use Case ID: QicFix-T-202.

Use Case Title: View Rating and Comments.

Use Case Level: Moderate-Level.

Details: The System shall allow Tower to view its rating and comments from its profile.

- **Actor:** Tower

- **Pre-conditions**: The System is functioning properly, User is already logged in to the system.

- **Description**:

Trigger: The user initiates an action by clicking on view Profile button on the Main Menu Screen.

1. The system responds by opening the profile with current information stored in the database.
2. The System displays the profile screen with the user's information.
3. The User clicks on view ratings and comments link.
4. The System displays all the details of comments and rating given by the client.
5. The User clicks the 'OK' Button.

6. The System reloads the page with the profile.

- **Relevant requirements**: None.
- **Post-conditions**: The System displays the Ratings and Comments on User by Clients.

Alternative Courses of Action: None.

Extensions: None.

Exceptions: If Internet Connection gets lost, Request Time-out.

Concurrent Uses: None

Related Use Cases: Cancel, Log out.

Decision Support

Frequency: Frequently. Will occur at most once per application run.

Criticality: Medium. This is the only way a client knows how this Tower is.

Risk: Low Risk

Constraints:

- Usability - Must be intuitive and easy to use. Less than 5 seconds to identify what to do. Shouldn't take more than 10 seconds to view a review.
- Reliability - Must be reliable for the client to review the tower. Mean time for failure: 1 out of every 200 executions fails
- Performance - The system should complete and save the review in the system within less than 2 second.
- Supportability - Will be supported on web browsers and Android phones
- Implementation - In Java and Android Studio.

Modification History
Owner: Freny Patel
Date last modified: 03-13-2016

11.2.3. Accept Request:

Use Case ID: QicFix-T-203

Use Case Title: Accept Request

Use Case Level: High-level

Details: The system shall allow tower to accept request.

- **Actor**: Tower and Client

- **Pre-conditions**: The System is functioning properly and Actor already has a Profile stored in the Profile Database.

- **Description**:

  Trigger: The Client selects Tower from the Tower list.

  1. The system presents list of Towers with the current data and reviews of Towers.
  2. The client selects the Tower he/she wants from the list for the towing service.
  3. The System sends request to the Tower from the Client side.
  4. The System prompts the request to the Tower Screen for the service.
  5. The Tower accepts the Request for the specified service.
  6. Client gets confirmation for the service accepted for the job.

- **Relevant requirements:** None.

- **Postconditions**:

1. Successful Completion:
If the use case ends in success, the Client gets the confirmation message for the Request accepted for the job.

2. Failure Condition:
If there is a failure the user's Client needs to retry with the same process again and Tower doesn't get notify for the request.

Alternative Courses of Action:

1. At step 2, system displays error message if client sends request without selecting the Tower.
2. At step 5, system displays error message if Tower doesn't accept the request for the job.

Extensions: None.

Exceptions: If Internet connection gets lost, request time out.

Concurrent Uses: None

Related Use Cases: Decline Request.

Decision Support:
Frequency: Approximately one execution per application run.

Criticality: Moderately Important. Not needed to run the program, but is a necessary for the system

Risk: Low

Constraints:

- Usability: Must be intuitive to the user, no training required.
- Reliability: Must be very reliable, as it is not used often it must function when needed. 1 failure every 24 hours is acceptable.
- Performance: The system should complete the operation in less than 5 seconds
- Supportability: Should be available for all users of Android SDK 15+.
- Implementation: Should be done in Java for Android

Modification History
Owner:Freny Patel
Date last modified: 01-21-2016

11.2.4. ChargeClient:

Use Case ID: QicFix-T-204.

Use Case Title: Charge Client

Use Case Level: High-Level

Details:
The System shall allow Tower to Charge Client after the job has been done from its profile.

- **Actor**: Tower

- **Pre-conditions**: The System is functioning properly, User is already logged in to the system and has completed the job.

- **Description**:

  Trigger: The user initiates an action by clicking on Charge Client Button on its profile

  1. The system responds by opening the Charge Client Screen.
  2. The User input the values in the JobType and Rate Fields in the Page.
  3. The User clicks the 'OK' Button.
  4. The System re-directs the page to the profile.

- **Relevant requirements**: None.

- **Post-conditions**: The System charges client according to the User inputs.

Alternative Courses of Action: None.

Extensions: None.

Exceptions: If Internet Connection gets lost, Request Time-out.

Concurrent Uses: None

Related Use Cases: Cancel, Log out.

Decision Support:
● **Frequency**: Frequently. Will occur at most once per application run.
● **Criticality**: High. This is the only way a Tower can Charge a client for the service provided.

Constraints:

● **Usability** - Must be intuitive and easy to use. Less than 7 seconds to perform task. The client should be able to read the message and then see the amount they are charged by the Tower.
● **Reliability** - Must be very reliable for the Tower to charge the Client. Mean time for failure: 1 out of every 300 executions fails .
● **Performance** - Speed is important for this operations and the system should complete the operation within less than 2 second.
● **Supportability** - Will be supported on web browsers and Android phones.
● **Implementation** - In Java

11.2.5. Decline Request:

Use Case ID: QicFix-T-205

Use Case Title: Decline Request

Use Case Level: High-level

Details: The system shall allow tower to decline request.

● **Actor**: Tower and Client

● **Pre-conditions**: The System is functioning properly and Actor already has a Profile stored in the Profile Database.

● **Description:**

Trigger: The Client selects Tower from the Tower list.

1. The system presents list of Towers with the current data and reviews of Towers.
2. The client selects the Tower he/she wants from the list for the towing service.
3. The System sends request to the Tower from the Client side.
4. The System prompts the request to the Tower Screen for the service.
5. The Tower declines the Request for the specified job.

6. Client gets confirmation for that the request is declined for the job.

- **Relevant requirements:** None**.**

- **Post-conditions:**

1. Successful Completion:
If the use case ends in success, the Client gets the confirmation message for the Request declined for the job.
2. Failure Condition:
If there is a failure the Client needs to retry with the same process again and Tower doesn't get notify for the request.

Alternative Courses of Action:

1. At step 2, system displays error message if client sends request without selecting the Tower.
2. At step 5, system displays error message if Tower doesn't decline the request for the job.

Extensions: None**.**

Exceptions: If Internet connection gets lost, request time out.

Concurrent Uses: None

Related Use Cases: Accept Request.

Decision Support:

Frequency: Approximately one execution per application run.

Criticality: Moderately Important. Not needed to run the program, but is a necessary for the system.

Risk: Low

Constraints:

- Usability: Must be intuitive to the user, no training required.
- Reliability: Must be very reliable, as it is not used often it must function when needed. 1 failure every 24 hours is acceptable.
- Performance: The system should complete the operation in less than 5 seconds.
- Supportability: Should be available for all users of Android SDK 15+.
- Implementation: Should be done in Java for Android

Modification History
Owner: Freny Patel
Date last modified: 01-21-2016

11.2.6. Session Timeout (Security) :

Use Case ID: QicFix_T-207_Sec_SessionTimeout

Use Case Level: System-Level End-to-End

Details:

- **Actor:** User
- **Pre-conditions**:User is logged into the system.
- **Description**: QicFix ends a session after 10 minutes of inactivity
  Trigger: 10 minutes elapse without any user activity
  1. QicFix navigates to log in screen
- **Post-conditions:**
  1. The session belonging to the user is destroyed.
- **Alternative Courses of Action**: None.

Extensions: None.

Exceptions: Network connectivity issues could cause any login attempt to fail.

Concurrent Uses: None.

Related Use Cases:
QicFix_T-106_Misuse_UseOpenSessionOnPublicComputer

Decision Support

Frequency: Low - Performed every time a user wants to maliciously use another session (2 times per week)

Criticality: High - We must protect user accounts from being compromised.

Risk: Low

Constraints:

- **Usability**: No previous training is needed.
- **Reliability:** Mean Time to Failure - 1 failure for every 72 hours of operation is acceptable.
- **Performance**: The authentication process should take 2 seconds.
- **Supportability:** The phone application relies on the Android platform and must run on any Android device version 3.0 and above.
- **Implementation**: Client requests the implementation to be done in Java and the Eclipse framework.

Modification History:
Owner: Freny Patel
Initiation Date: 02/19/2016
Date Last Modified: 02/19/2016

11.2.7. Change Client Pickup Location

Use Case ID: QicFix-C-301
Use Case Title: Change client pick up location for a Tow Service
Use Case Level: High-level

Details: The user can change his or her pick up location in a GPS referenced map.

- **Actor:** Client, Tower and Google Maps API.
- **Pre-conditions:** Client must be logged in, internet service must be available.
- **Description:** The user changes his or her pick up location in a GPS referenced map.

Trigger: Client clicks to open Change Pick up Location option.

The system responds by…

1. Display Change Pick up Location screen.
2. The client must keep pressed 2 seconds over the symbol that indicates his or her current location.
3. The client moves the symbol where he or she believes indicates best his or her real pick up location in a GPS referenced map.
4. The system request confirmation of new pick up location.
5. The system updates this new pick up location in the server.

- **Relevant requirements:** None.
- **Post-conditions:** Active Tower' screen must be updated with new pick up location.

Alternative Courses of Action: Client doesn't change his or her current pick up location.

Extensions: Change service status.

Exceptions:

1. Internet connection gets lost.
2. Error of communication with server.
3. Error of communication from server to Tower.
4. Session timeout.

Concurrent Uses:

1. QicFix-S-301: Tower accepts client request.
2. QicFix-S-302: Tower decline from service previously accepted.

Related Use Cases:

1. QicFix-C-320: Client cancel service request.
2. QicFix-C-201: Client logout.

Decision Support
Frequency: Usually 1 time per application execution.

Criticality: Not Mandatory.

Risk: Low

Constraints:

- Usability: Must be easy to understand how to change user's location. It should take less than 1 minute to perform this task.
- Reliability: Must be accurate to where the user is selecting for the tower to arrive exactly at the location.
- Supportability: Must support changes in Android App and in webpage.

Modification History
Owner: Juan Pablo Sotomayor
Date last modified: 01-24-2016

11.2.8. Change Client Destination Location

Use Case ID: QicFix-C-305
Use Case Title: Change client destination location for a Tow Service
Use Case Level: High-level

Details: The user can change his or her destination location in a GPS referenced map.

- **Actor:** Client, Tower and Google Maps API.

- **Pre-conditions:** Client must be logged in, internet service must be available.

- **Description:** The user changes his or her destination location in a GPS referenced map.

Trigger: Client clicks to open Change Destination Location option.

The system responds by…

1. Display Change Destination Location form.
2. The client must keep pressed 2 seconds over the symbol that indicates his or her destination location.
3. The client moves the symbol where he or she believes indicates best his or her real destination location in a GPS referenced map.
4. The system request confirmation of destination location.
5. The system updates this new destination location in the server.

- **Relevant requirements:** None.

- **Post-conditions:** Active Tower' screen must be updated with new destination location.

Alternative Courses of Action: Client doesn't change his or her current destination location.

Extensions: Change service status.

Exceptions:
- Internet connection gets lost.
- Error of communication with server.
- Error of communication from server to Tower.
- Session timeout.

Concurrent Uses:
- QicFix-S-301: Tower accepts client request.
- QicFix-S-302: Tower decline from service previously accepted.

Related Use Cases:
- QicFix-C-320: Client cancel service request.
- QicFix-C-201: Client logout.

Decision Support
Frequency: Usually 1 time per application execution.

Criticality: Not Mandatory.

Risk: Low

Constraints:

- Usability Must be easy to understand how to change user's destination location. It should take less than 1 minute to perform this task.
- Reliability – Must be accurate to where the user is selecting for the tower to deliver exactly at the location. Mean time for failure: 1 out of every 300 executions fails
- Supportability: Must support changes in Android App and in webpage.

Modification History
Owner: Juan Pablo Sotomayor
Date last modified: 01-24-2016

11.2.9. Show List of Towers

Use Case ID: QicFix-C-310
Use Case Title: Display an ordered list of Towers
Use Case Level: High-level

Details: The user can request a sorted list of towers.

- **Actor:** Client and Google Maps
- **Pre-conditions:** Client must be logged in, internet service must be available
- **Description:** The user requests a list of best rated towers nearby his or her current location.

Trigger: Client clicks to open List of Towers option.

The system responds by…

1. Displaying a screen with the options for sorting the list of towers.
2. The client clicks on the preferred options.
3. The system retrieves a sorted list of towers from server with the option selected by the user.
4. The system displays in the client interface a List of Towers sorted by the option selected.
5. The client can browse through the list.

- **Relevant requirements:** None.

- **Post-conditions:** None.

Alternative Courses of Action: Cancel list request.

Extensions: None.

Exceptions:
- Internet connection gets lost.
- Error of communication with server.

Concurrent Uses: None.

Related Use Cases:
- QicFix-C-104: Client rates tower.
- QicFix-C-201: Client logout.

Decision Support
Frequency: Usually 1 time per application execution.

Criticality: Not Mandatory.

Risk: Low

Constraints:
- Usability - Must be easy to understand the list order and rating points. Less than 5 seconds to identify what to do. Shouldn't take more than 30 seconds to understand the list ordering.
- Reliability - Must be accurate to what other users have rated. Mean time for failure: 1 out of every 200 executions fails
- Supportability: Must support changes in Android App and in webpage.

Modification History
Owner: Juan Pablo Sotomayor
Date last modified: 03-24-2016

11.2.10. Queue Towers

Use Case ID: QicFix-C-101
Use Case Title: Client's queue of towers
Use Case Level: system-level end-to-end

Details: This use case will allow the client to select up to 3 towers to put into their queue. A message will be sent to the tower prompting them of the job.

- **Actor:** Client

- **Pre-conditions:** Tower successfully registered with QicFix, and is on the application. Client successfully registered with QicFix, logged in. The Client enables the GPS location finder

- **Description:**

  Trigger: The Client selects either the closest, cheapest or best tower around.

  1) The system brings the client to a list of towers based off their preference.
  2) The client selects the tower.
  3) The system places the tower in one of 3 slots at the bottom of the app and grays them out from the list.

- **Post-conditions:** The tower will confirm the job and receive the contact information of the Client. The towers location will appear on a map for the Client.

Alternative Courses of Action:

1. After step 2, the Client can select 2 or 3 tower to put in their queue.
2. After step 2, the Client can deselect a tower
3. After step 1, the Client can view the information of the tower

Exceptions:

1. If a communication error occurs between the application and the database, then an error will be displayed.

Concurrent Uses: Another Tower receives a notification for a job.

Related Use Cases:

- Client can display:
  - Best tower around them

    - Cheapest Tower around them
    - Closest Tower around them
- Client can enable GPS location finding

Decision Support
Frequency: Frequently. Will occur at most 3 times per application run.

Criticality: High. This is the only way a client can notify a tower.

Risk: Low Risk

Constraints:

- Usability - Must be intuitive and easy to use. Less than 5 seconds to perform task. The client should know they can only pick up to 3 tower. Less than 10 seconds to fill queue
- Reliability - Must be very reliable for both the client and tower to get in contact with one another. Mean time for failure : 1 out of every 300 executions will fail
- Performance - Speed is important for this operations and the system should complete the operation within less than 1 second.
- Supportability - Will be supported on web browsers and Android phones
- Implementation - In Java

Modification History
Owner: Steve Foo

Date last modified: 1/30/16

11.2.11. Pay Towers

Use Case ID: QicFix-C-102
Use Case Title: Client paying the towing fee
Use Case Level: system-level end-to-end

Details: This use case will make the client pay for the tow truck drivers service when they arrive to their destination

- **Actor:** Client

- **Pre-conditions:** Tower successfully registered with QicFix, and is on the application. Client successfully registered with QicFix, logged in. The Client enables the GPS location finder. The

Client selects a Tower and the tower accepts the job. The Tower has taken the Client to their destination

- **Description:**

  Trigger: The Tower confirms that the client has arrived at their destination and charges the client.

  1) The client receives a message to confirm they arrived at their destination.
  2) Once they confirm, the client is prompted they will be paying $XX.XX.

- **Post-conditions:** The client will be able to rate the tower based off their service.

Alternative Courses of Action:

1. At step 1, the Client can not confirm they didn't arrive to their destination

Exceptions:

1. If a communication error occurs between the application and the database, then an error will be displayed.

Concurrent Uses:

Related Use Cases:

- Client can display:
  - Best tower around them
  - Cheapest Tower around them
  - Closest Tower around them
- Client can enable GPS location finding
- QicFix-C-102

Decision Support

Frequency: Frequently. Will occur at most once per application run.

Criticality: High. This is the only way a client can pay a tower for their service.

Risk: Low Risk

Constraints:

- Usability - Must be intuitive and easy to use. Less than 7 seconds to perform task. The client should be able to read the message and then see the amount they will be charged
- Reliability - Must be very reliable for the client to pay the tower. Mean time for failure : 1 out of every 300 executions will fail
- Performance - Speed is important for this operations and the system should complete the operation within less than 2 second.
- Supportability - Will be supported on web browsers and Android phones
- Implementation - In Java

Modification History
Owner: Steve Foo

Date last modified: 1/23/16

11.2.12. Review Towers

Use Case ID: QicFix-C-103
Use Case Title: Client rates the tower
Use Case Level: system-level end-to-end

Details: This use case will make the client rate the tower based off their service

- **Actor:** Client

- **Pre-conditions:** Tower successfully registered with QicFix, and is on the application. Client successfully registered with QicFix, logged in. The Client enables the GPS location finder. The Client selects a Tower and the tower accepts the job. The Tower has taken the Client to their destination and the Client has confirmed they've arrived at the destination

- **Description:**
  Trigger: The Client pays the Tower
  1) The Client sees dialog box with a 5 star rating system and a comment section
  2) The Client picks a rating and writes a comment. Then confirms it.

- **Post-conditions:** The client will be brought back to the home view of the applicaiton.

Alternative Courses of Action:
Exceptions:
1. If a communication error occurs between the application and the database, then an error will be displayed.

Concurrent Uses:
Related Use Cases:
- QicFix-C-101

Decision Support
Frequency: Frequently. Will occur at most once per application run.

Criticality: Medium. This is the only way a client can let others know how this Tower is.

Risk: Low Risk

Constraints:

- Usability - Must be intuitive and easy to use. Less than 5 seconds to identify what to do. Shouldn't take more than 30 seconds to submit a review.
- Reliability - Must be reliable for the client to review the tower. Mean time for failure : 1 out of every 200 executions will fail
- Performance - The system should complete and save the review in the system within less than 2 second.
- Supportability - Will be supported on web browsers and Android phones
- Implementation - In Java

Modification History
Owner: Steve Foo

Date last modified: 1/23/16

11.2.13. Client Views Tower's Ratings

Use Case ID: QicFix-C-104
Use Case Title: Client views the Tower rating and reviews
Use Case Level: system-level end-to-end

Details: This use case will allow the client to look at the rating and previous reviews of a Tow truck driver.

- **Actor:** Client

- **Pre-conditions:** Tower successfully registered with QicFix, and is on the application. Tower has been reviewed by a previous Client. Client successfully registered with QicFix, logged in. The Client enables the GPS location finder

- **Description:**
  Trigger: The Client selects either the closest, cheapest or best tower around.

1. The client selects the tower and views their information
2. an average rating is displayed and a list of reviews are shown.

- **Post-conditions:** The client will click off the review screen and look at the other Towers

Alternative Courses of Action:

Exceptions:

1. If a communication error occurs between the application and the database, then an error will be displayed.

Concurrent Uses:
Related Use Cases:

- Client can display:
    - Best tower around them
    - Cheapest Tower around them
    - Closest Tower around them
- Client can enable GPS location finding

Decision Support
Frequency: Frequently. Will occur at most 10 times per application run.

Criticality: Medium. The only way the user can see what a tower is rated.

Risk: Low Risk

Constraints:

- Usability - Must be intuitive and easy to use. Less than 5 seconds to perform task. The client should know they can look at rating of a tower.
- Reliability - Must be very reliable for client to see reviews of Tower. Mean time for failure : 1 out of every 300 executions will fail
- Performance - Speed is important for this operations and the system should complete the operation within less than 1 second.
- Supportability - Will be supported on web browsers and Android phones
- Implementation - In Java

Modification History
Owner: Steve Foo

Date last modified: 1/30/16

11.2.14. Login Attempts

Use Case ID: QicFix-CT-105
Use Case Title: Client fails five times to login
Use Case Level: system-level end-to-end

Details: This use case illustrate someone failing to login 5 times..

- **Actor:** Client

- **Pre-conditions:** Client successfully registered with QicFix.

- **Description:**

  Trigger: The Client opens the application

  1) The Client puts in an incorrect email and password.
  2) They submit the information but fail to login.
  3) The Client repeats step 1 five times.
- **Post-conditions:** The client will not be logged into the application

Alternative Courses of Action:

Exceptions:

1. If a communication error occurs between the application and the database, then an error will be displayed.

Concurrent Uses:
Related Use Cases:

Decision Support
Frequency: Will occur at most once per application run.

Criticality: High. The only way the user can access the application.

Risk: Low Risk

Constraints:

- Usability - Must be intuitive and easy to use. Less than 5 seconds to perform task.
- Reliability - Must be very reliable for both the client to login. Mean time for failure : 1 out of every 300 executions will fail
- Performance - Speed is important for this operations and the system should complete the operation within less than 1 second.
- Supportability - Will be supported on web browsers and Android phones
- Implementation - In Java

Modification History
Owner: Steve Foo

Date last modified: 1/30/16

11.2.15 Registration (of the Client):

Use Case ID: QicFix-C-402

Use Case Title: Registration (of the client)

Use Case Level: End to End

Details: Registration is required before client can use the application, client cannot enter to the application and start using before making the registration.

- **Actor:** Client
- **Pre-conditions:** The application must be installed before client can start using.
- **Description**: Registration is one of the main sections of the application since we can gather client's information before letting him/her use the application. Registration can help the application security and it makes it much easier for the owners to track how many people are using the application and track of the money that we make.

Username should be unique for each user, two users cannot use the same user name.

Password must be less than 8 characters and must be at least included a capital letter,

small letter and a number.

Trigger:

1. Client enters the full name.
2. Client enters the valid email address.
3. Client enters the valid phone number.
4. Client enters the password.
5. Client re-enters the password for verification.
6. Client enters the credit card number.
7. Client click submit.

- Relevant requirements

- **Post-conditions:** Registration is completed if the verification code matches our record

Alternative Courses of Action:

Extensions: If client successfully complete the registration process has already been logged in to the application and can start using it. Payment information can be placed after registration is complete in order to have client payment information.

Exceptions:

1. Internet disconnection can interrupt the registration

Related Use Cases:

QicFix-CT-401

QicFix-CT-404

Decision Support

Frequency:

Registration occurs only one time per user. Once the user has been registered the application, it can be used with saved login information and registration is not needed any more for each time use.

Criticality: This step is very critical an application cannot perform and cannot be used before registration is complete. Incomplete registration does not allow the user to enter to the application and use it's benefits.

Risk:

This step contains high risk, because usually people use the same password for everything, they are giving us their information including full name, email address, phone number and their password that usually the same for everything.

Constraints:

- **Usability:** In registration part, every step is going to be very simple including, entering full name, entering valid email address, entering valid phone number, entering valid password (8 alphanumeric characters), entering security code.
- **Reliability:** Registering is the most important part of the application; without correct registration the application is not usable.
- **Performance:** When they tapping the submit button they should receive a confirmation message less than 5 seconds, to be able to use the application.
- **Supportability:** We offer our application in English language and some parts have pictures that show what that part means (For example we have a tower vehicle picture for tower man and screw like picture for mechanic).

The Chrome is the web based browser and for smart phones we use Android.

- Implementation: It is the platforms that we discussed above.

Modification History

Owner: Maral Kargarmoakhar

Initiation date: 01/20/2016

Date last modified: 1/26/2016

11.2.16 Registration (of the Tow person):

Use Case ID: QicFix-T-403

Use Case Title: Registration (of the Tow person)

Use Case Level: End to End

Details: Registration is required before Tow person can get into the application, Tow person cannot enter to the application and start representing himself before making the registration.

- **Actor:** Tow person
- **Pre-conditions:** The application must be installed before Tow person can entering to the application.
- **Description:** Registration is one of the main sections of the application since we can gather Tow person's information before letting him/her use the application. Registration can help the application security and it makes it much easier for the owners to track how many tow persons are registered in the application.

Username should be unique for each tow person; two tow persons cannot use the same user name. Password must be less than 8 characters and must be at least included a capital letter, small letter and a number.

Trigger:

1. Tow person enters the full name.
2. Tow person enters the valid email address.
3. Tow person enters the valid phone number.
4. Tow person enters the password.
5. Tow person re-enters the password for verification.
6. Tow person enters his/ her bank account number.
7. Tow person enters he/she has TRAA certificate or not
8. Tow person enters he/she has a tow company or not.
9. Tow person click submit

- **Relevant requirements**
- **Post-conditions:** Registration is completed if the required information is valid when the tow person submits it.

Alternative Courses of Action:

Extensions: If tow person successfully complete the registration process has already been logged in to the application and can start using it.

Exceptions:

1. Internet disconnection can interrupt the registration

Concurrent Uses:

Related Use Cases:

QicFix-CT-401

Decision Support

Frequency:

Registration occurs only one time per user. Once the user has been registered the application, it can be used with saved login information and registration is not needed any more for each time use.

Criticality: This step is very critical an application cannot perform and cannot be used before registration is complete. Incomplete registration does not allow the user to enter to the application and use it's benefits.

Risk

This step contains high risk, because usually people use the same password for everything, they are giving us their information including full name, email address, phone number and their password that usually the same for everything.

Constraints:

- Usability: In registration part, every step is going to be very simple including, entering full name, entering valid email address, entering valid phone number, entering valid password (8 alphanumeric characters).

- Reliability: Registering is the most important part of the application; without correct registration the application is not usable.
- Performance: When they tapping the submit button they should receive a confirmation message less than 5 seconds, to be able to use the application.
- Supportability: We offer our application in English language and some parts we have pictures that show what that part means (For example we have a tower vehicle picture for tower man).

The Chrome is the web based browser and for smart phones we use Android.

- Implementation: It is the platforms that we discussed above.

Modification History

Owner: Maral Kargarmoakhar

Initiation date: 01/20/2016

Date last modified: 1/26/2016

11.2.17 User (Client/ Tow person) Login:

Use Case ID: QicFix-CT-401

Use Case Title: User (Client/ Tow person) Login

Use Case Level: End-to-End

Details: Login is required for the registered User before the application can be used, the registered client or tow person cannot start using the application without login. All users have a unique user name that application identifies you in login process.

- **Actor:** User
- **Pre-conditions:** The user must be registered before can login.

- **Description:** login is the procedure used to get access to an application, login requires that the user have a user name (in our application user name is email address of the user) and a password.

Trigger:

1. User opens the application.
2. User tap on login button.
3. User enters the email address.
4. User enters the password.
5. User taps on submit button.

- **Relevant requirements:**
- **Post-conditions:** User can login to the system and start using the application if the user name and password matches our record.

Alternative Courses of Action: User can try 3 times in order to confirm the email address and password otherwise the registration is blocked and the user receives a notification email, the alternative course of action is the user choose the Forget password and receive an email to reset the password.

Extensions: Login is in relation with its previous step which is registration. User cannot login before completing the registration and having valid username and password. Login is also in extensive relation with logout process, logout cannot be done before the login has been performed

Exceptions:

1. Internet disconnection can interrupt the registration

Concurrent Uses: {This use can occur concurrently with the uses listed in this section.}

Related Use Cases:

QicFix-C-402

QicFix-T-403

Decision Support

Frequency:

Login time interval is a time dependent it means that login time interval increase with the age of the application. When people knows more about our application, the number of people who use the application and login to the system changes. For the first year we expect that for each user login 5 times in a month in average (for each client it should not be more than 4 times a year and for each tow person it should not be more than once in a week).

Criticality:

It is very necessary to login to the application before start using that, Login process is a critical part of the application. In the other words, absence of login process causes the system to not work.

Risk:

Login process does not have any risk because the user name is unique and defined as the email address of the user. Therefore, two users cannot have the same username. No one can get into other person user account unless a user has shared his/her username and password with that person.

Constraints:

- Usability: In the login part, the user has to enter the email address as username and password which has been set in the registration part, and click the submit button.
- Reliability: Login is an important part in application and without entering valid email address and password, user cannot work with the application.
- Performance: When the user clicks the submit button it should not take more than 5 seconds to login.
- Supportability: Login part support the user personal information. No one can access to other users personal and payment information without having the username and password. And also login process can help the user to reset the password if it is forgotten.

- Implementation: Login is implemented in order to get access to the application and use its benefits. Without implementing the login section, application is not accessible.

Modification History

Owner: Maral Kargarmoakhar

Initiation date: 02/15/2016

Date last modified:02/20/2016

11.2.18 **User (Client/ Tow person) Logout:**

Use Case ID: QicFix-CT-404

Use Case Title: User (Client/ Tow person) Logout

Use Case Level: End-to-End

Details: Logout is required for the registered and logged in User in order to store information securely on your device.

- **Actor:** User
- **Pre-conditions:** The user must be logged in before can logout.
- **Description:** The application account should not be left logged in for the security purposes, all applications are inherently untrusted that's why is better to logout after using the application.

Trigger:

1. User is active in the application.
2. User tap on logout button.

- **Relevant requirements:**
- **Post-conditions:** User can logout the system if the user has been already logged in to the system.

Alternative Courses of Action: User can just close the application or logoff the device without logging out the application.

Extensions: Logout is in relation with its previous step which is login. User cannot logout before login to the system. There is no step after logout phase.

Exceptions:

1. Internet disconnection can leave the application unlogged out. Once the internet is connected user can open the application and complete the logout process.

Concurrent Uses:

Related Use Cases:

QicFix-CT-401

Decision Support

Frequency:

Number of times that user logout the application is usually less than the number of login times, because not all the users logout the application after using the application. For the first year we expect that number of logout is half of the number of users who have logged into the system. For each user number of logout is 3 times in a month in average (for each client it should not be more than 2 times a year and for each tow person it should not be more than once in a two weeks).

Criticality:

Logout process is a not a critical part of the application. In the other words, absence of logout process does not cause any issue.

Risk:

There is a risk involved in logout process. Not Logging out the application can cause losing the personal information of the user.

Constraints:

- Usability: In the logout part, user safely quit the application.

- Reliability: Logout is an important part in application which can secure the personal and payment information of the users.
- Performance: When the user clicks the logout button it should not take more than 5 seconds to logout.
- Supportability: Logout part support and secure the user personal information.
- Implementation: Logout is implemented in order to get out of the application safely.

Modification History

Owner: Maral Kargarmoakhar

Initiation date: 02/15/2016

Date last modified:02/20/2016

10.3. Appendix C – User Interface Designs:

For Android Platform:

1. **Registration Interface:**

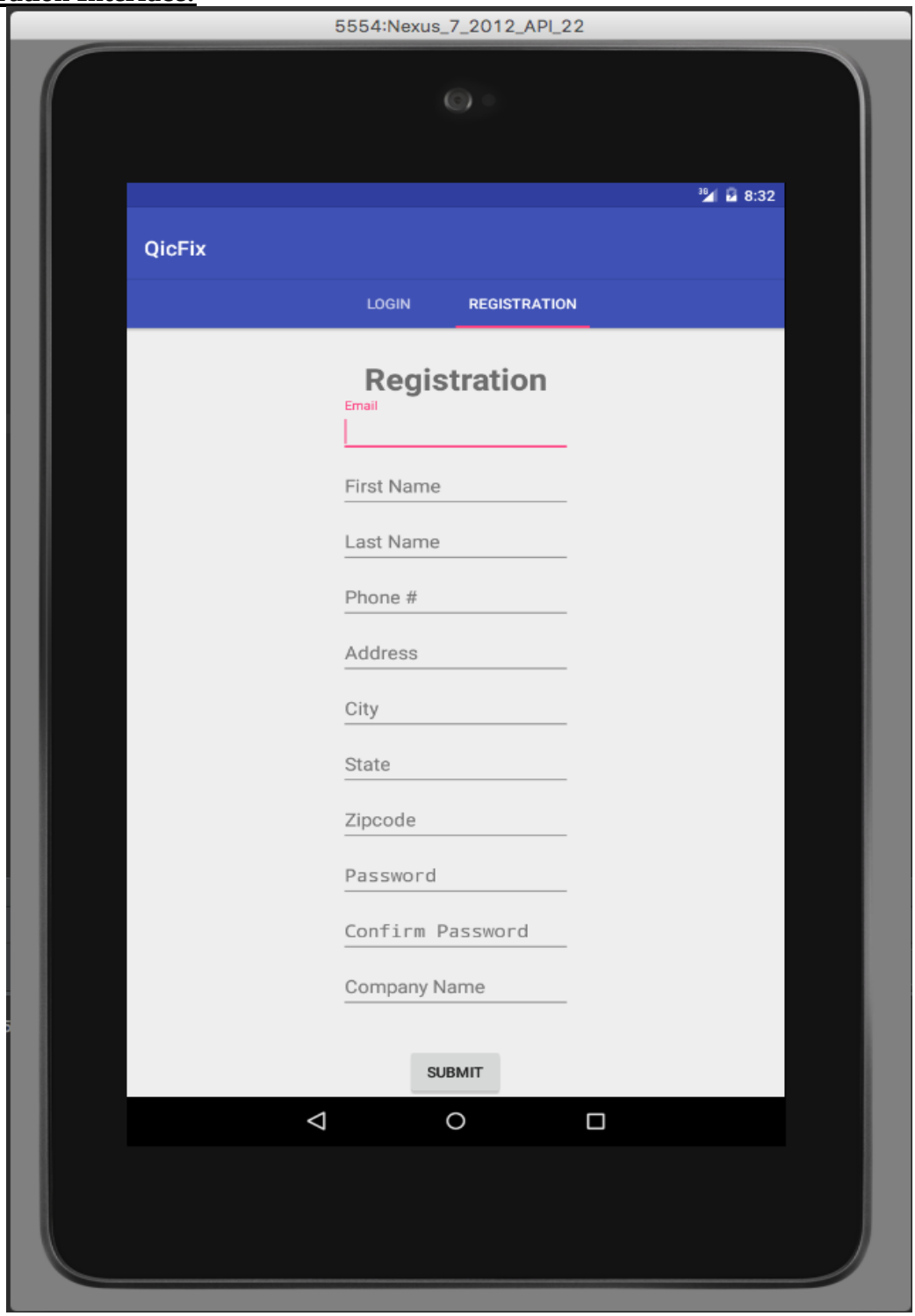


Figure:11.3.1. Registration Interface(Android App)

2. **Login Interface:**

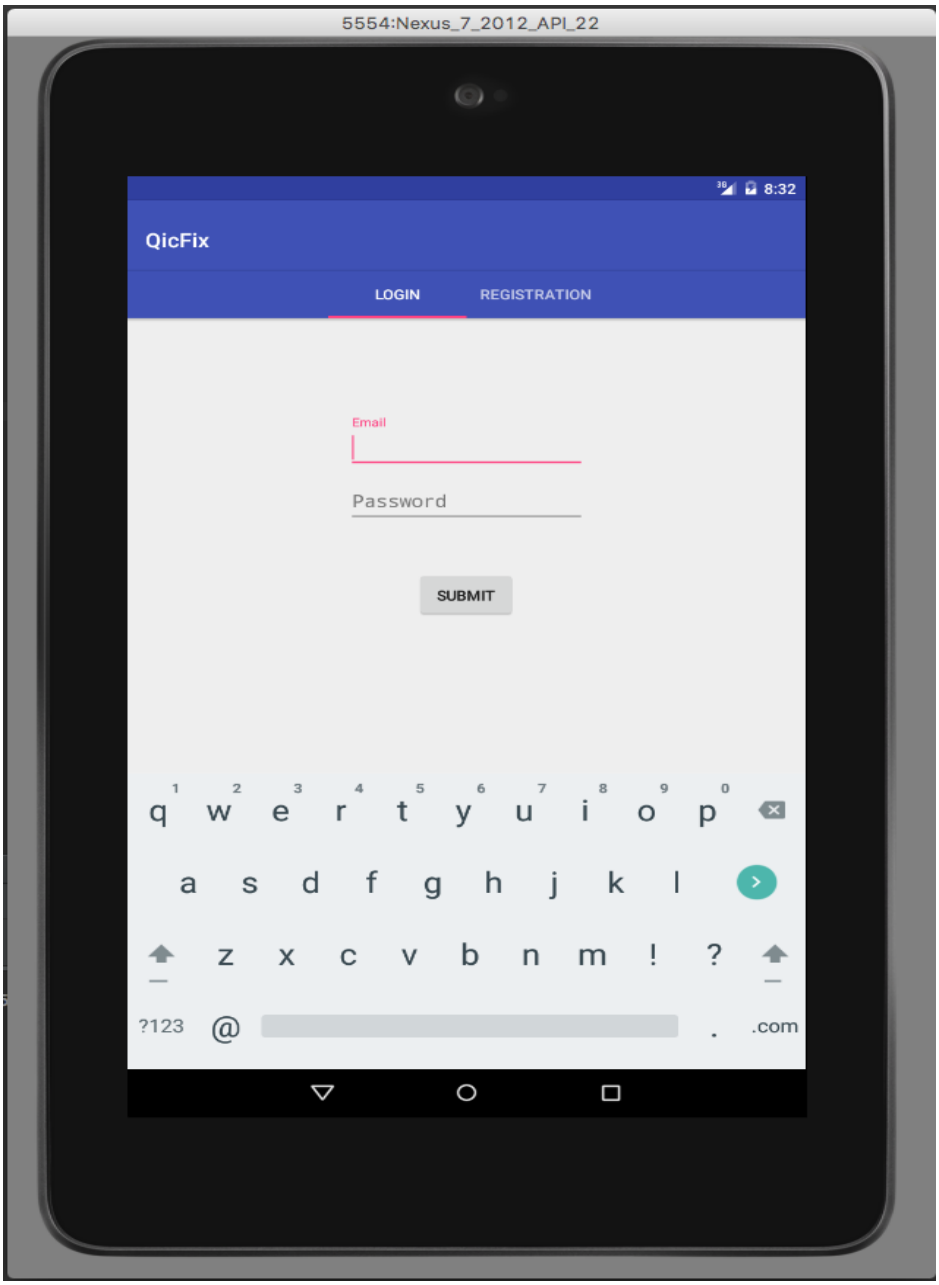


Figure:11.3.2. Login Interface(Android App)

3. **View Profile:**

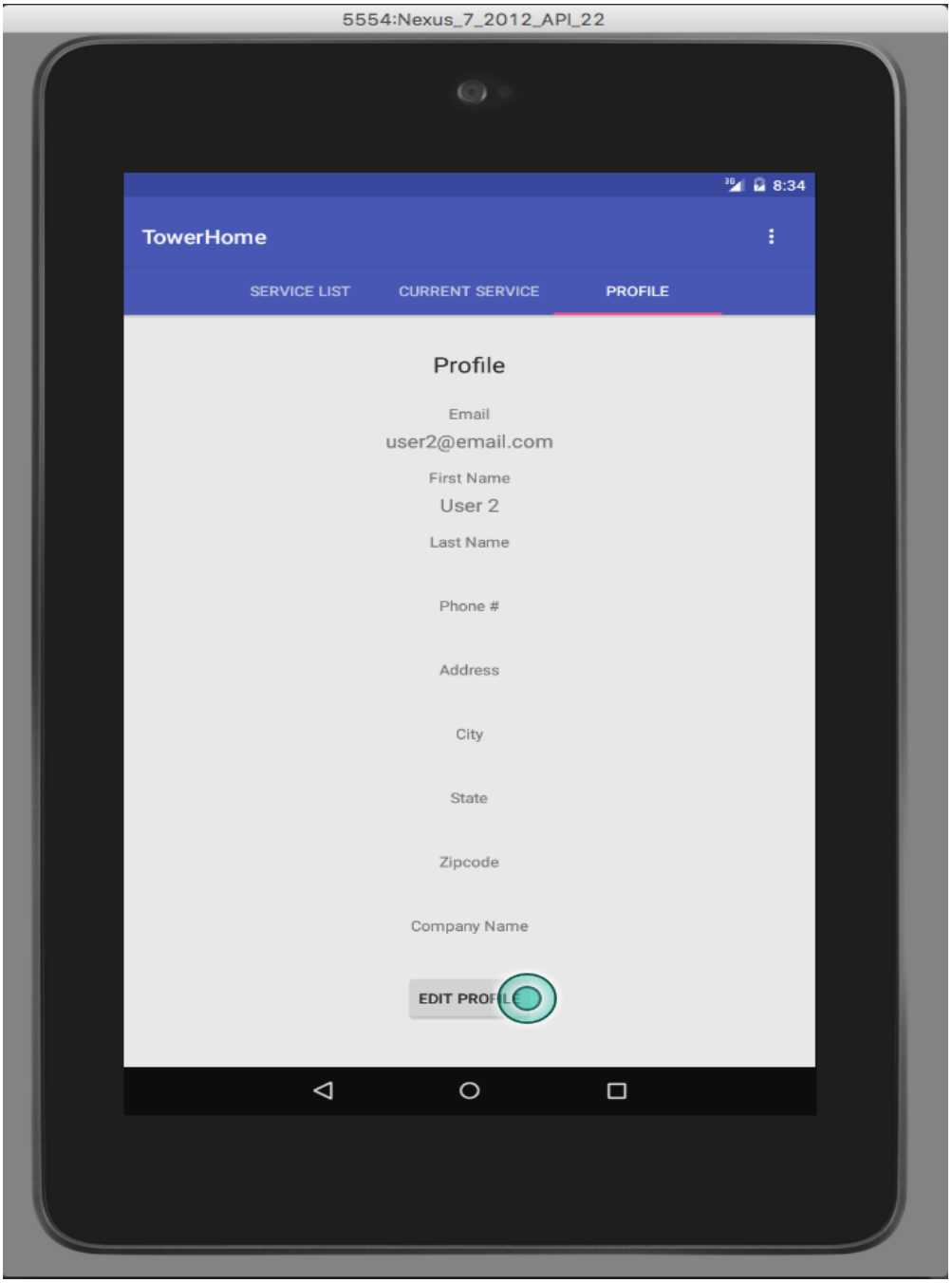


Figure:11.3.3. View Profile(Android App)

4. **Edit Profile:**

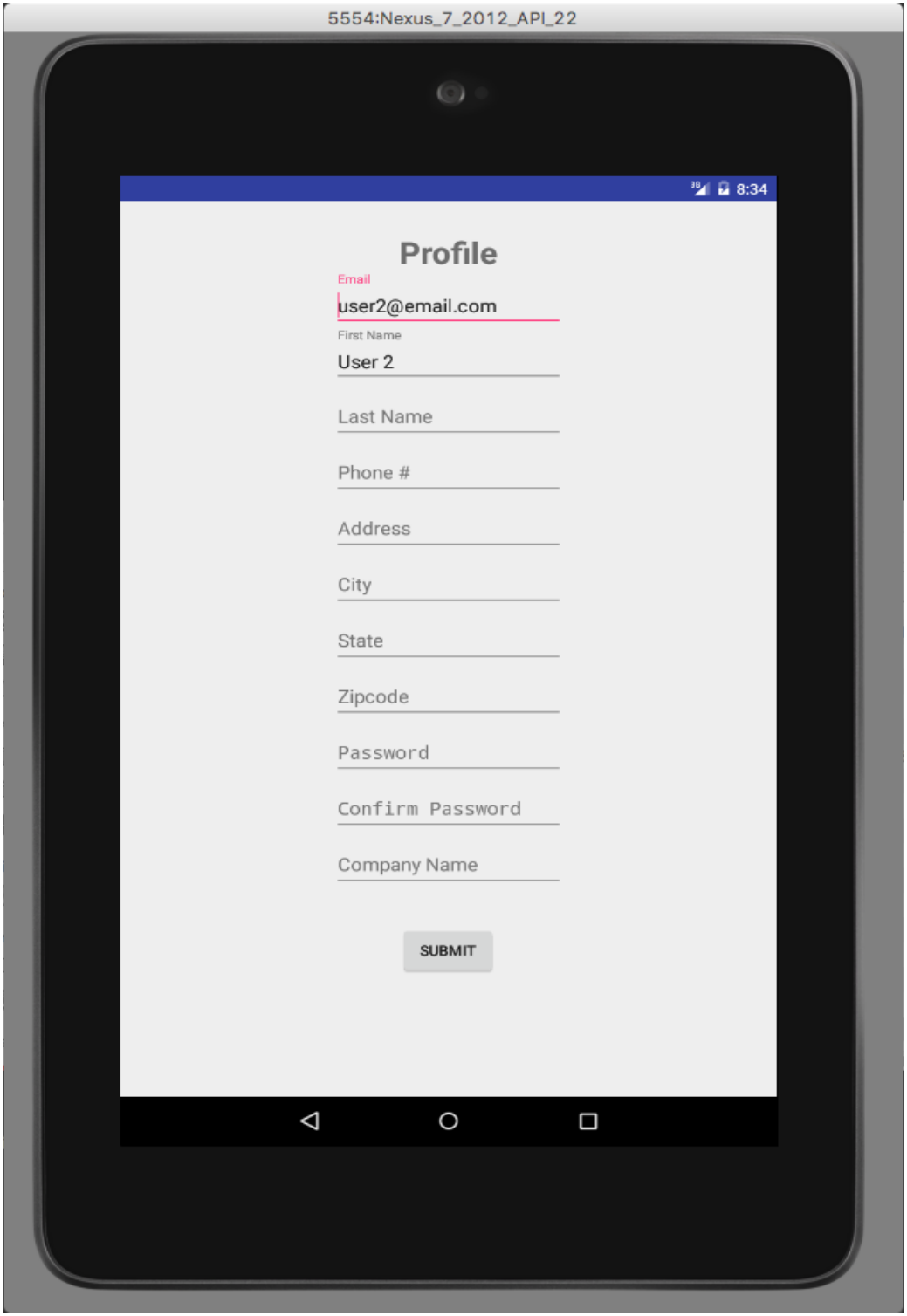


Figure:11.3.4. Edit Profile Interface(Android App)

5. **View All Services:**

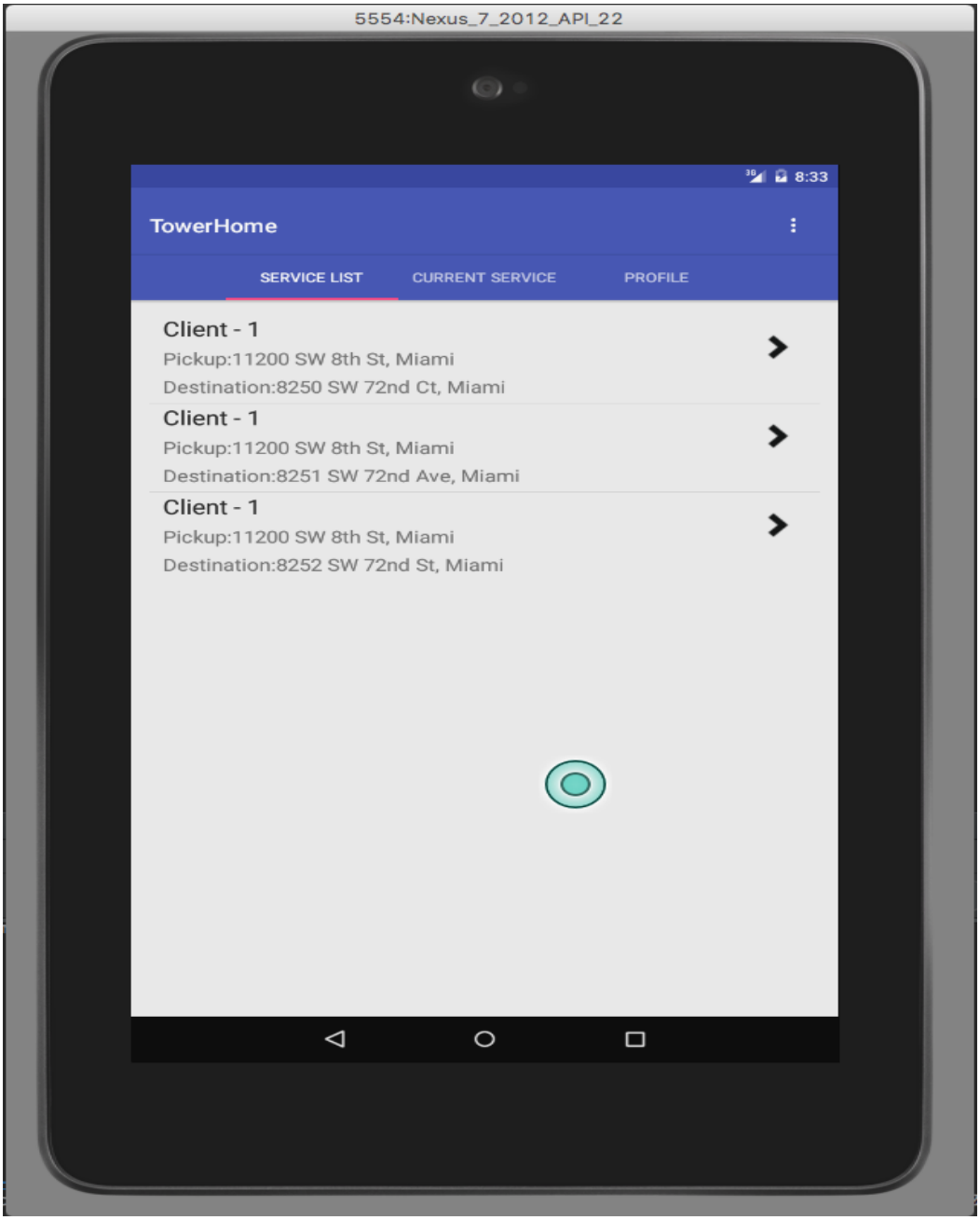


Figure:11.3.5. ViewAllServices Interface(Android App)

6. Accept/Decline Service:

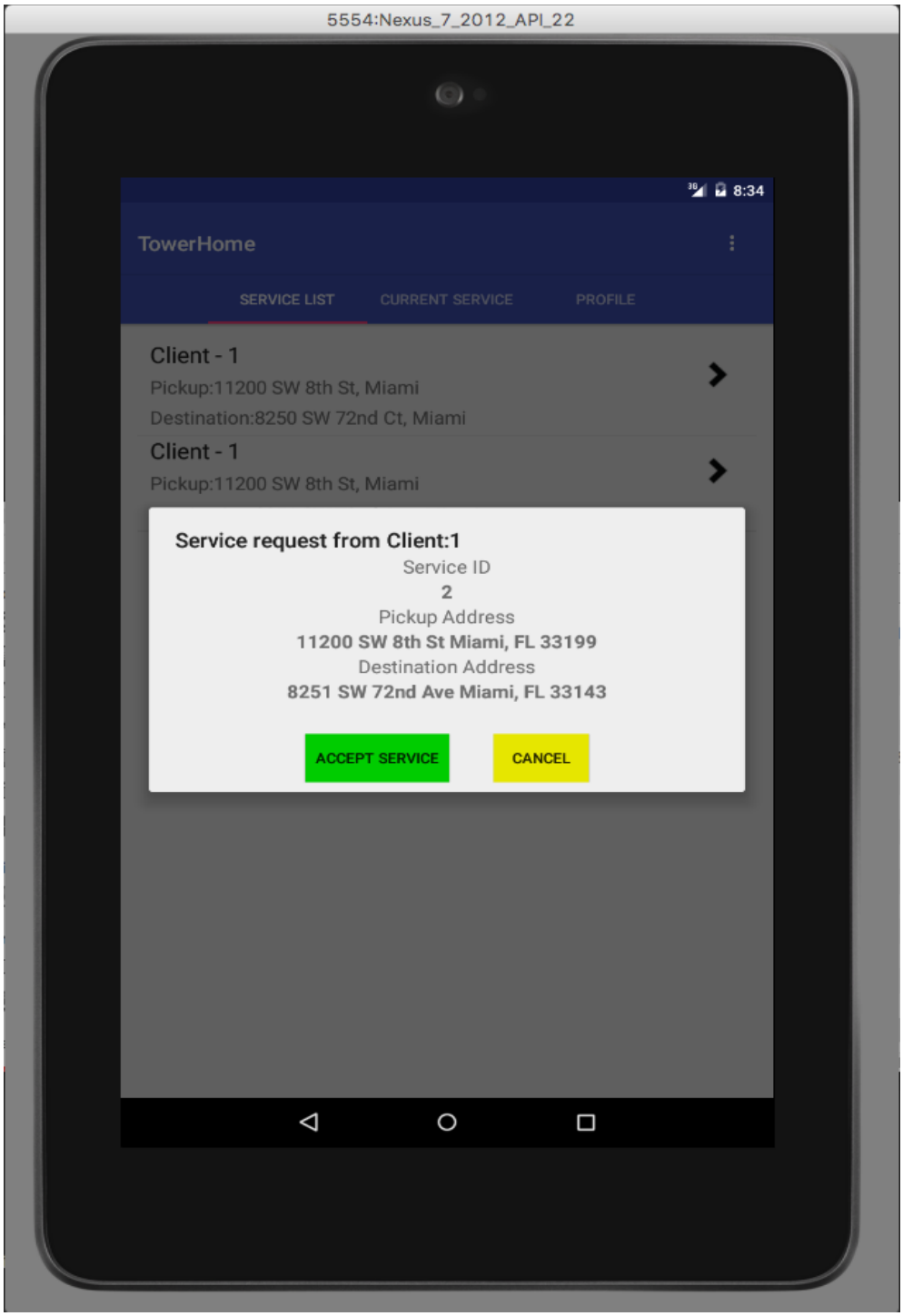


Figure:11.3.6. Accept/Decline ServicesInterface(Android App)

7. Logout:

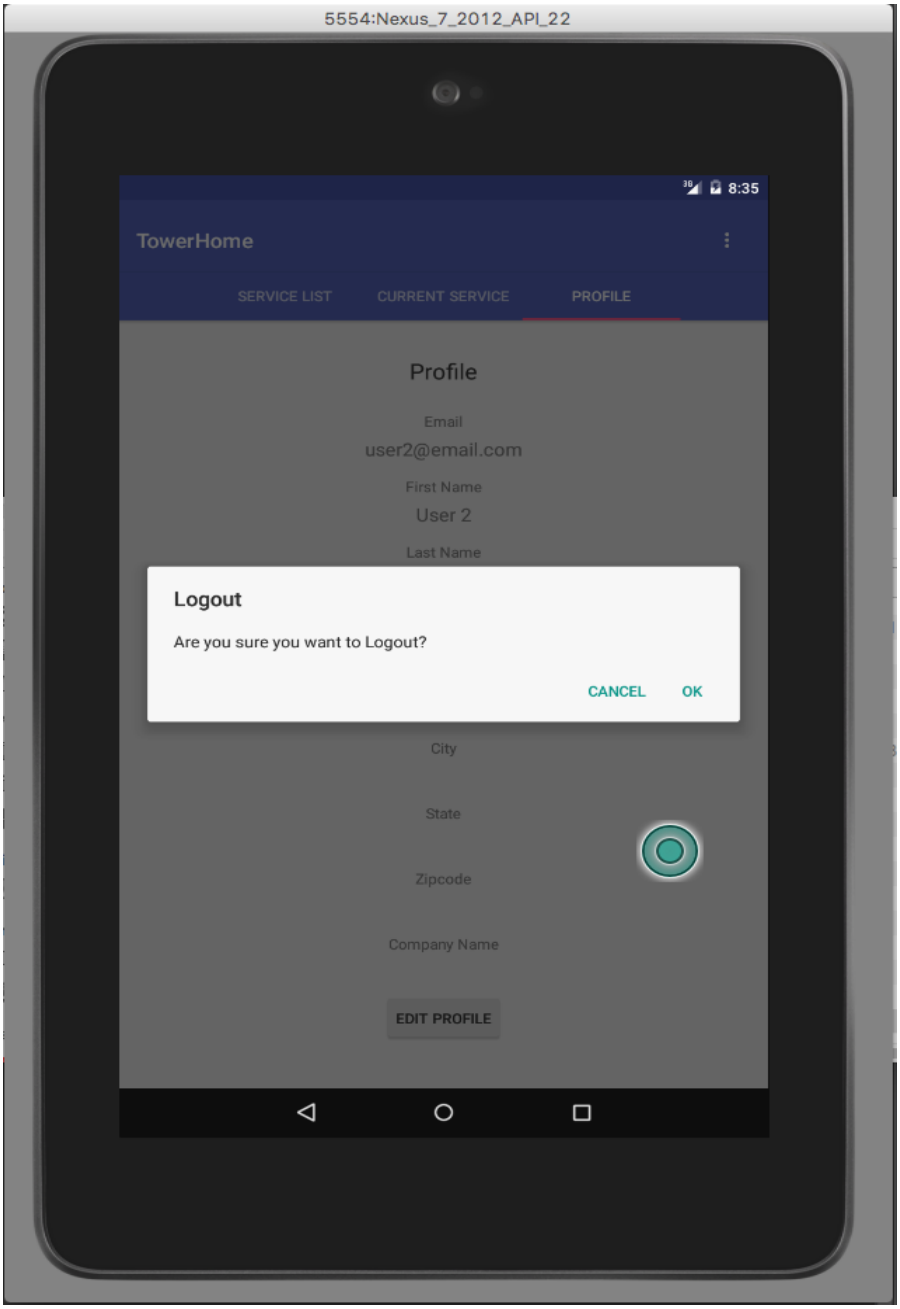


Figure:11.3.7.Logout Interface(Android App)

WEB INTERFACES:

8. Tower Registration Interface:

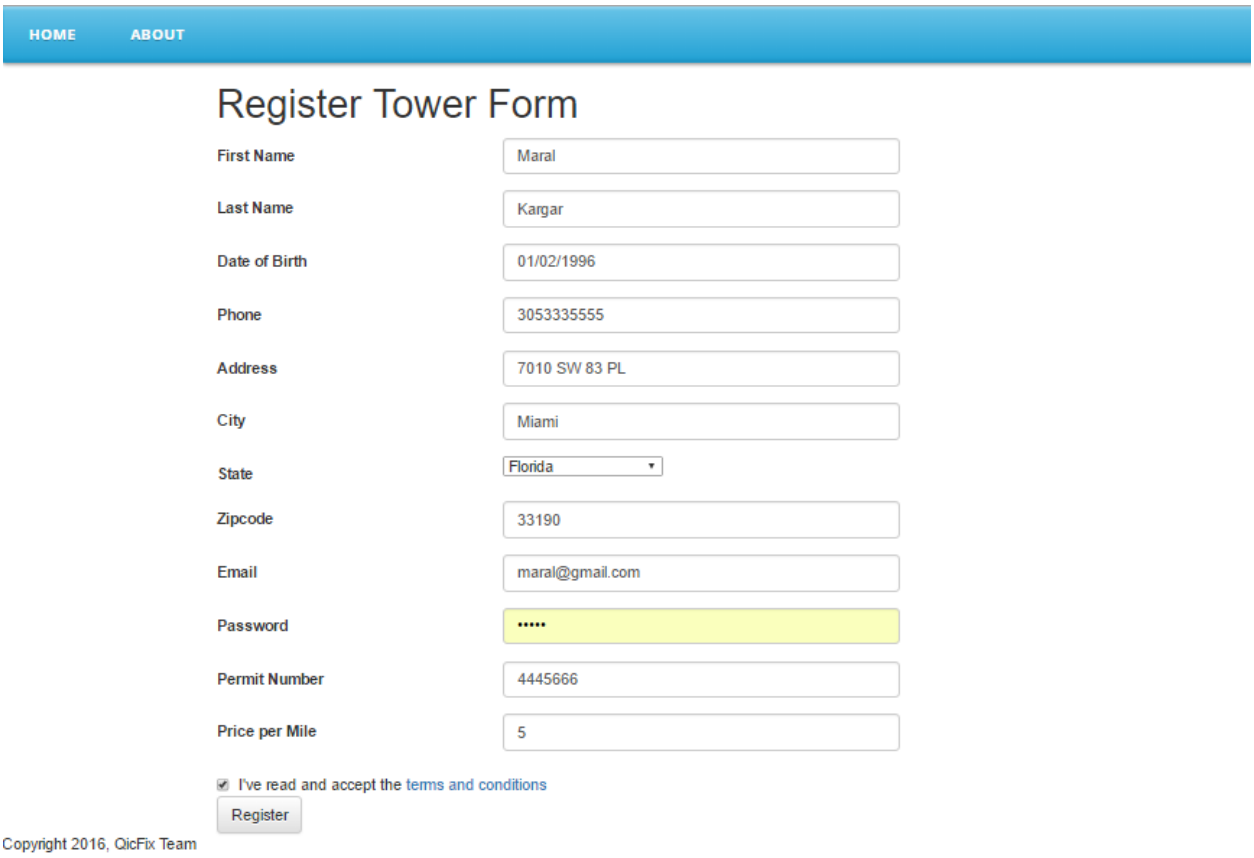


Figure:11.3.8.Tower Registration Interface(Web Platform)

9. Client Registration Interface:

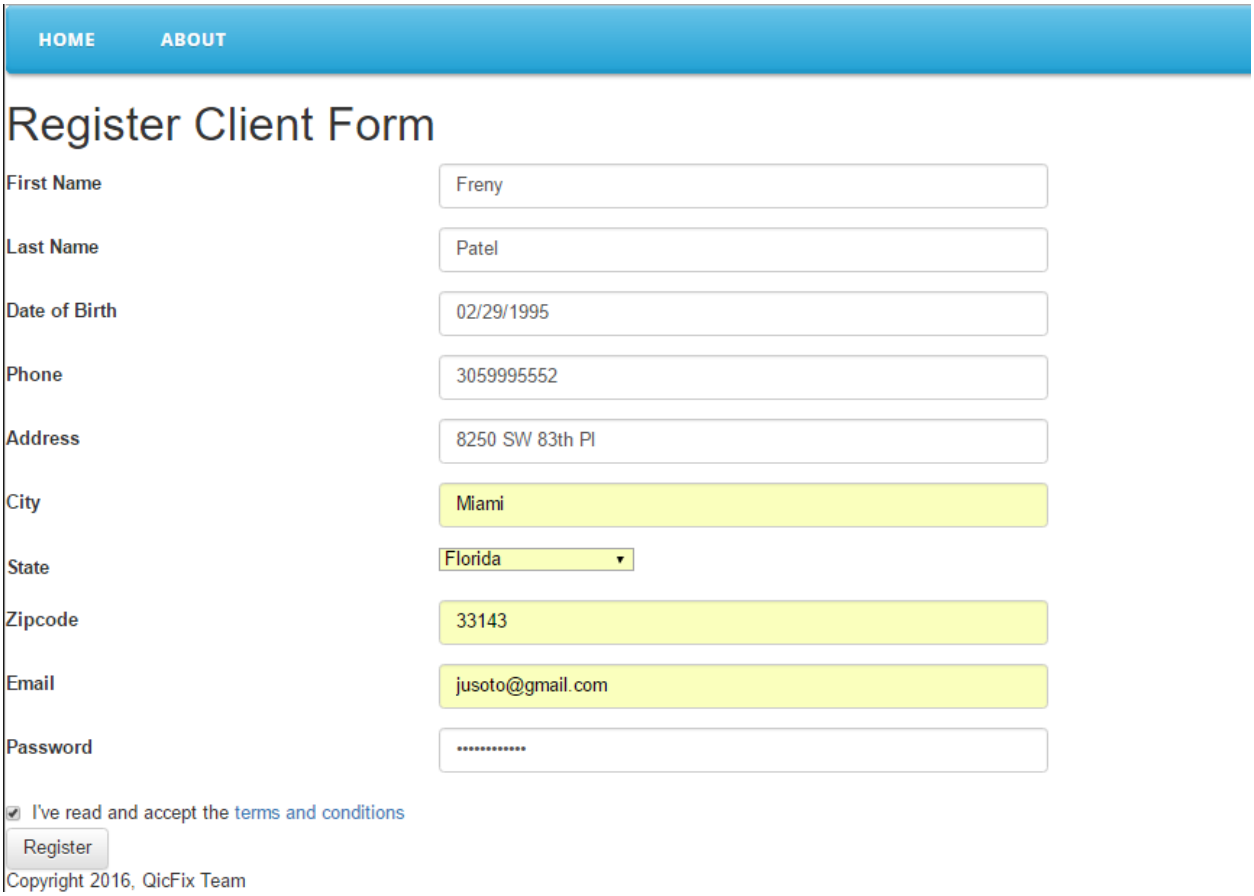


Figure:11.3.9.Client Registration Interface(Web Paltform)

10. Login Interface:

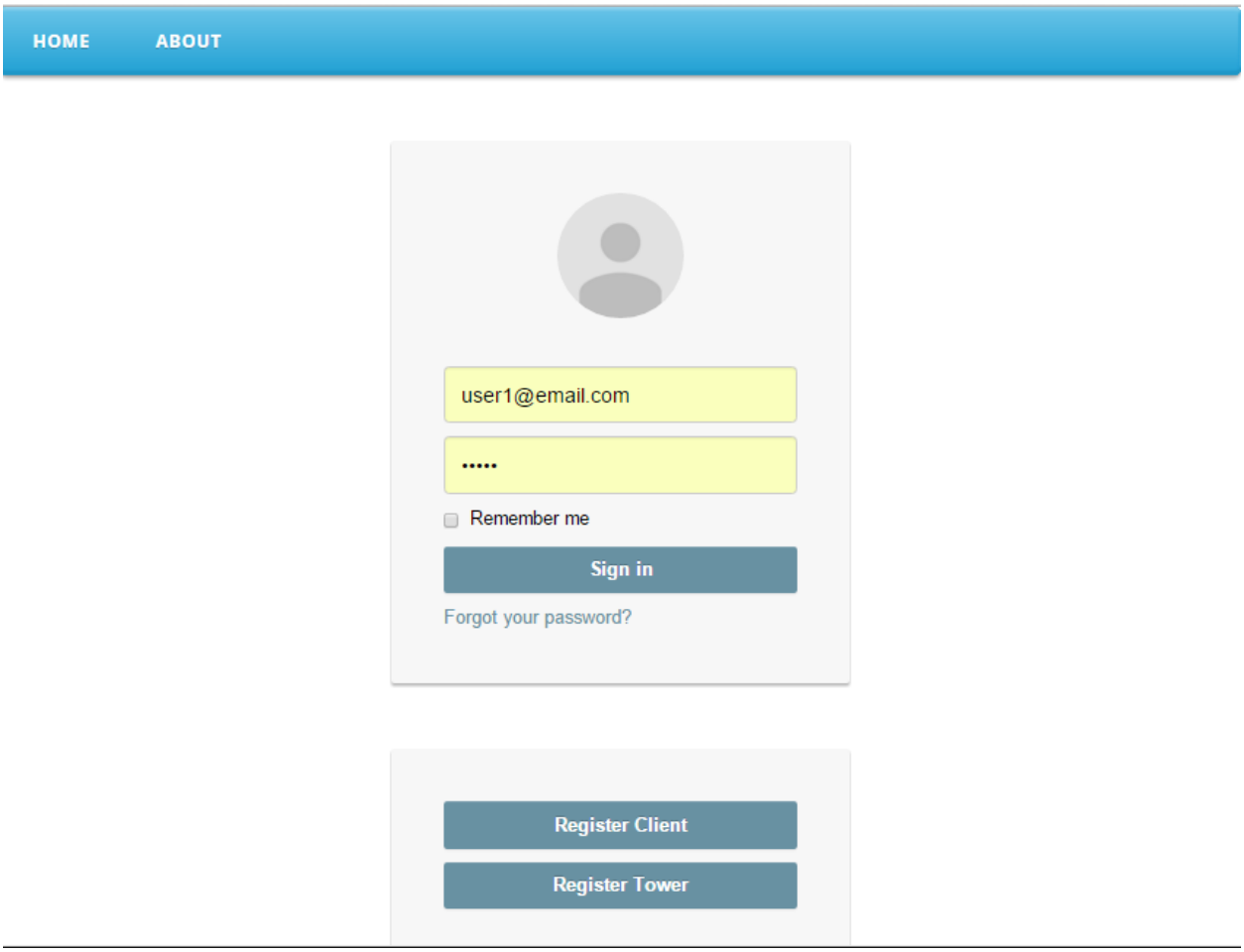


Figure:11.3.10.Login Interface(Web Platform)

11. Tower Profile Interface:

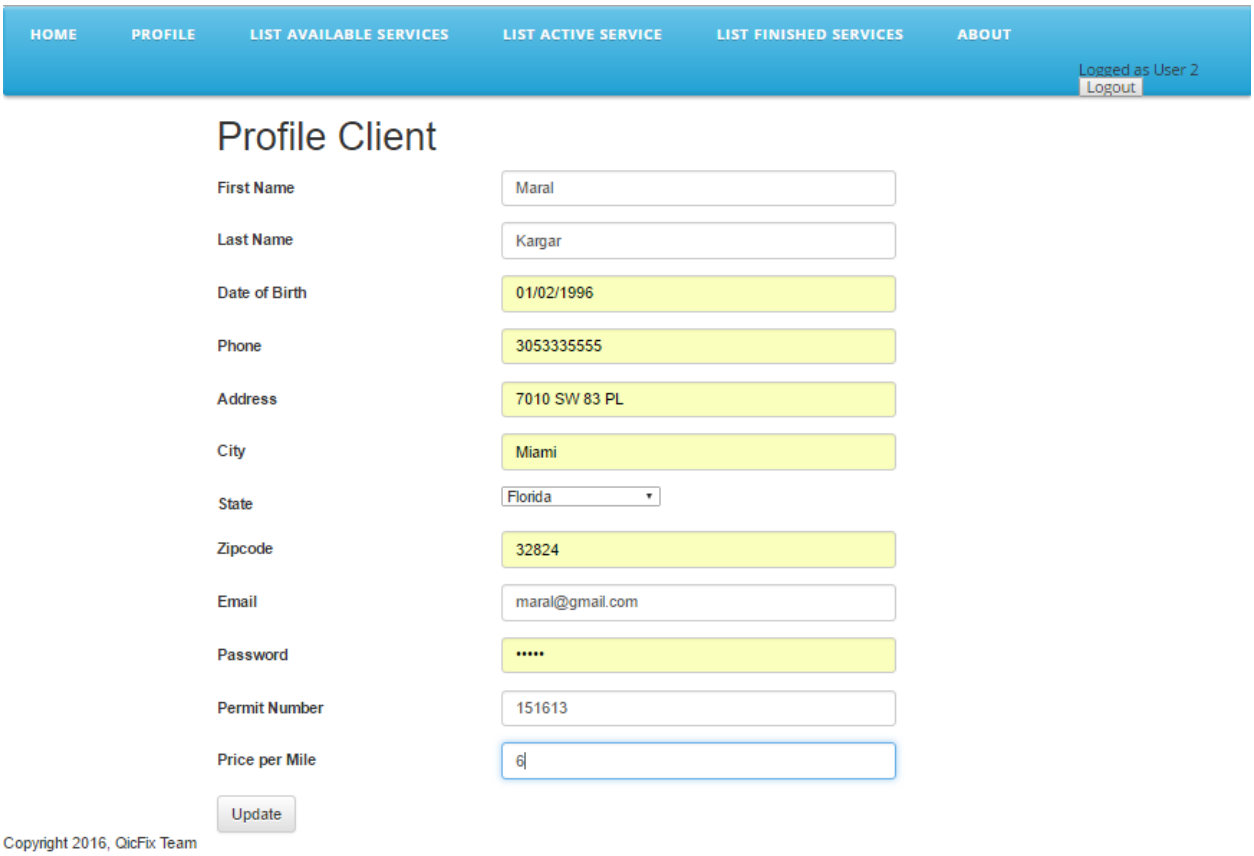


Figure:11.3.11.Tower Profile Interface(Web Platform)

12. Client Profile Interface:

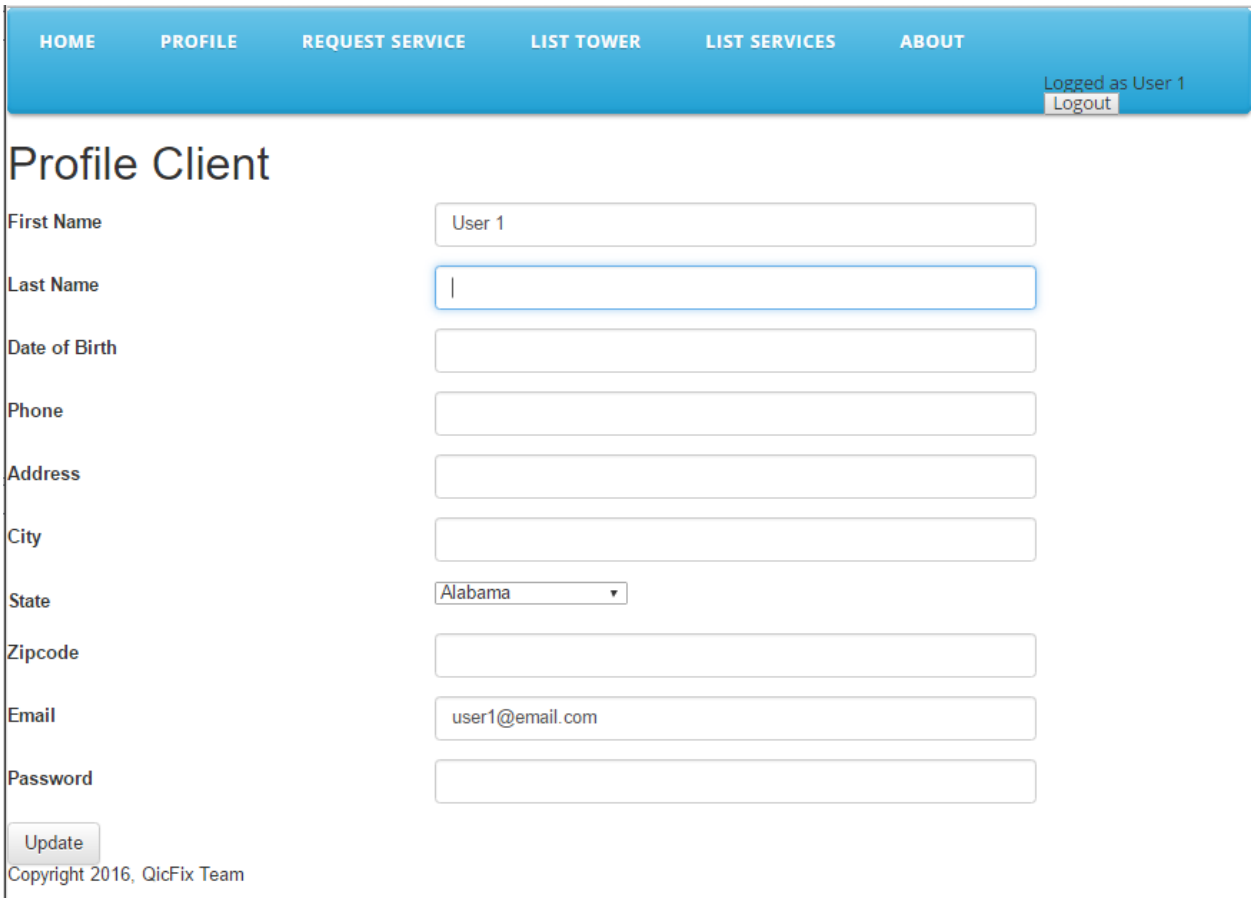


Figure:11.3.12.Client Profile Interface(Web Platform)

13. Tower Home Interface:

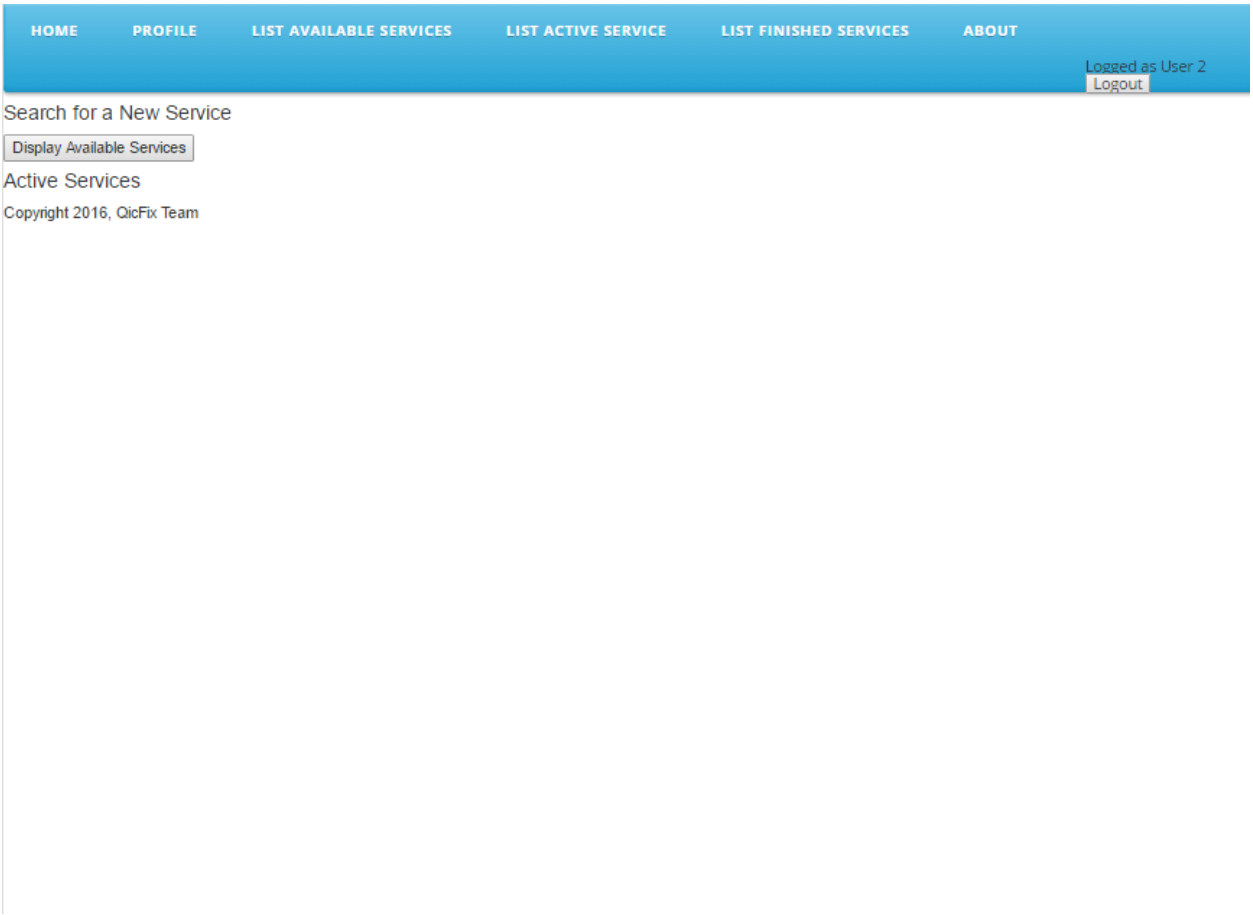


Figure:11.3.13.Tower Home Interface(Web Platform)

14. Client Home Interface:

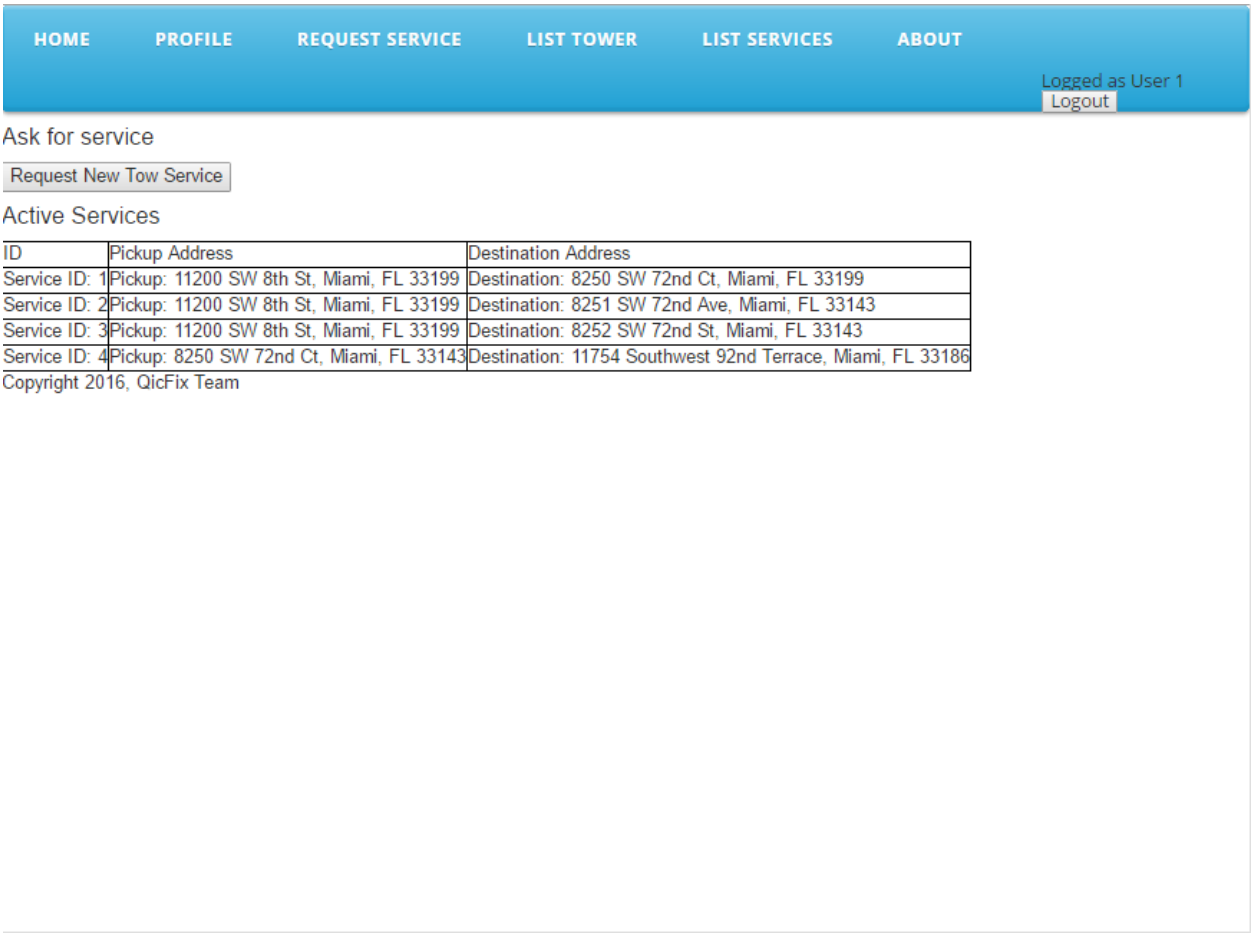


Figure:11.3.14. Client Home Interface(Web Platform)

15. About us Interface:

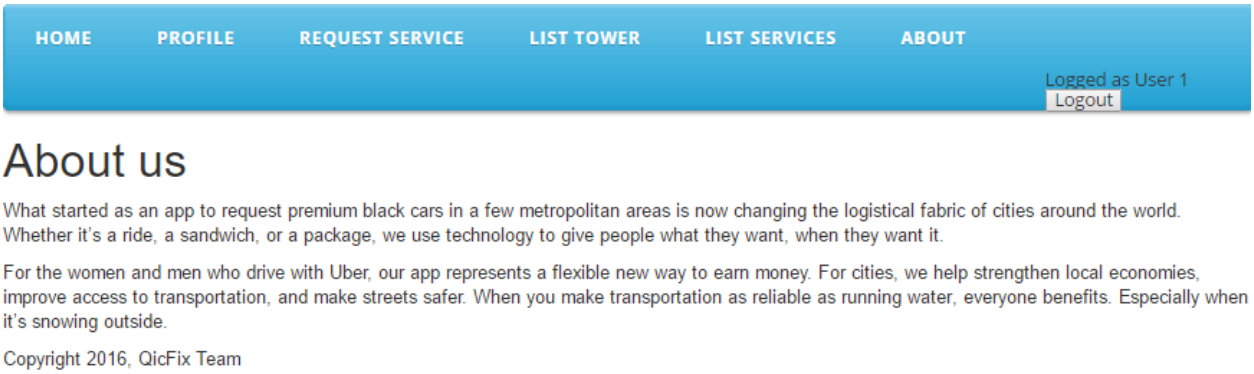


Figure:11.3.15.About Us Interface(Web Platform)

16. Request Service Interace:

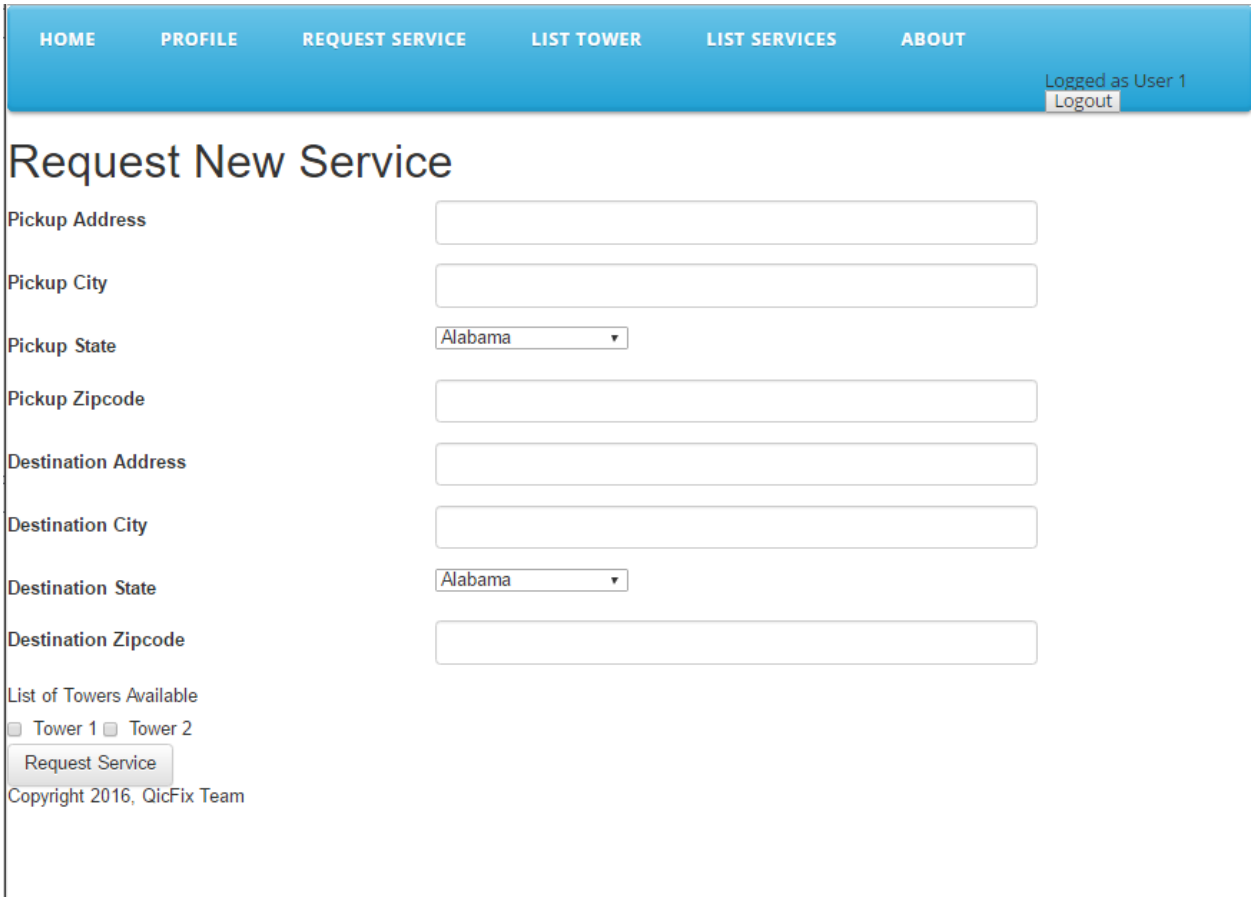


Figure:11.3.16.Request Service Interface(Web Platform)

17. List All Services Interface:

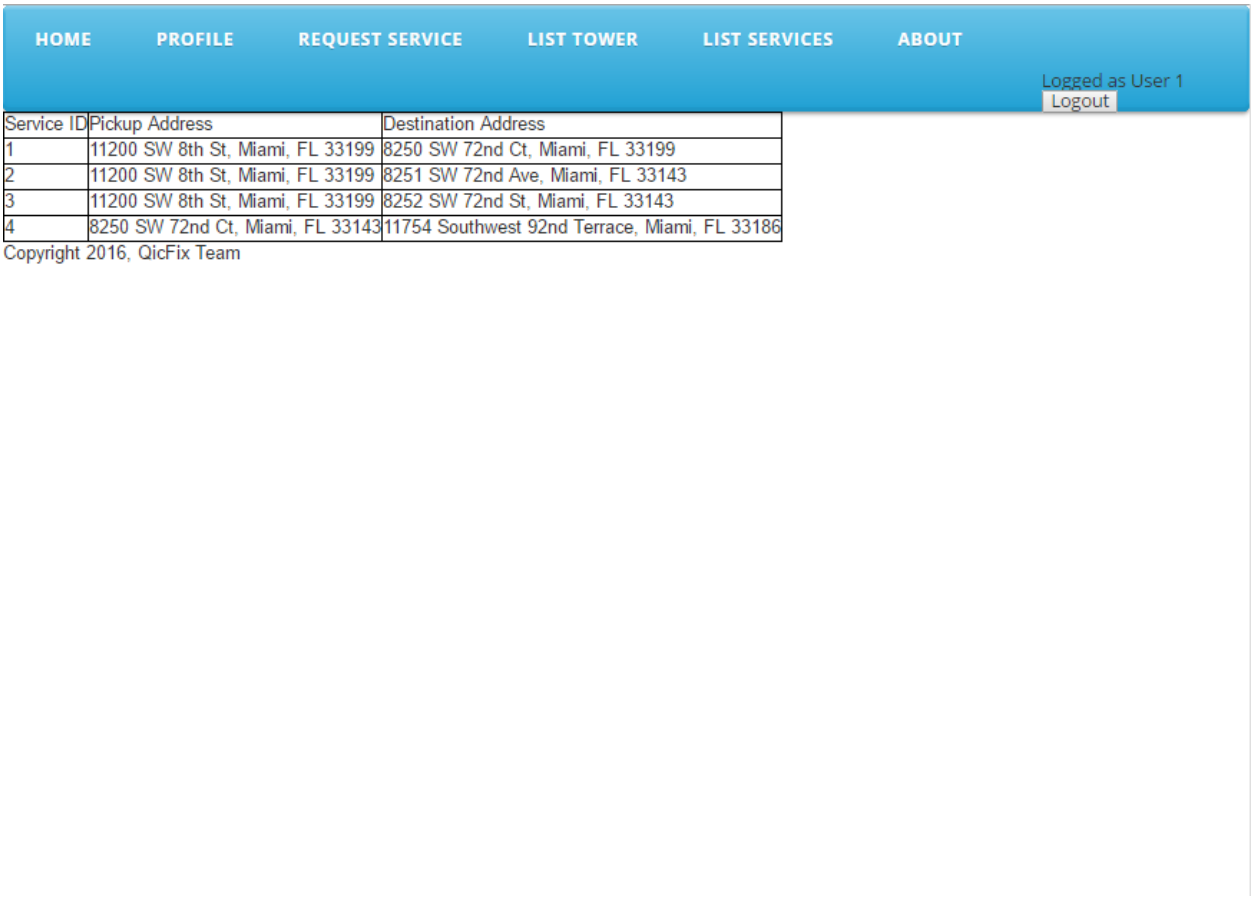


Figure:11.3.17. List All Services Interface(Web Platform)

10.4. 11.4. Appendix D – Detailed Class Diagrams:

The following class diagrams show attributes and methods for each class. The class diagrams follow a hierarchical decomposition approach starting from each major subsystem package. A note is shown on the top left of every class diagram denoting the package path for all of the classes on the diagram.

Client. Model Subsystem:

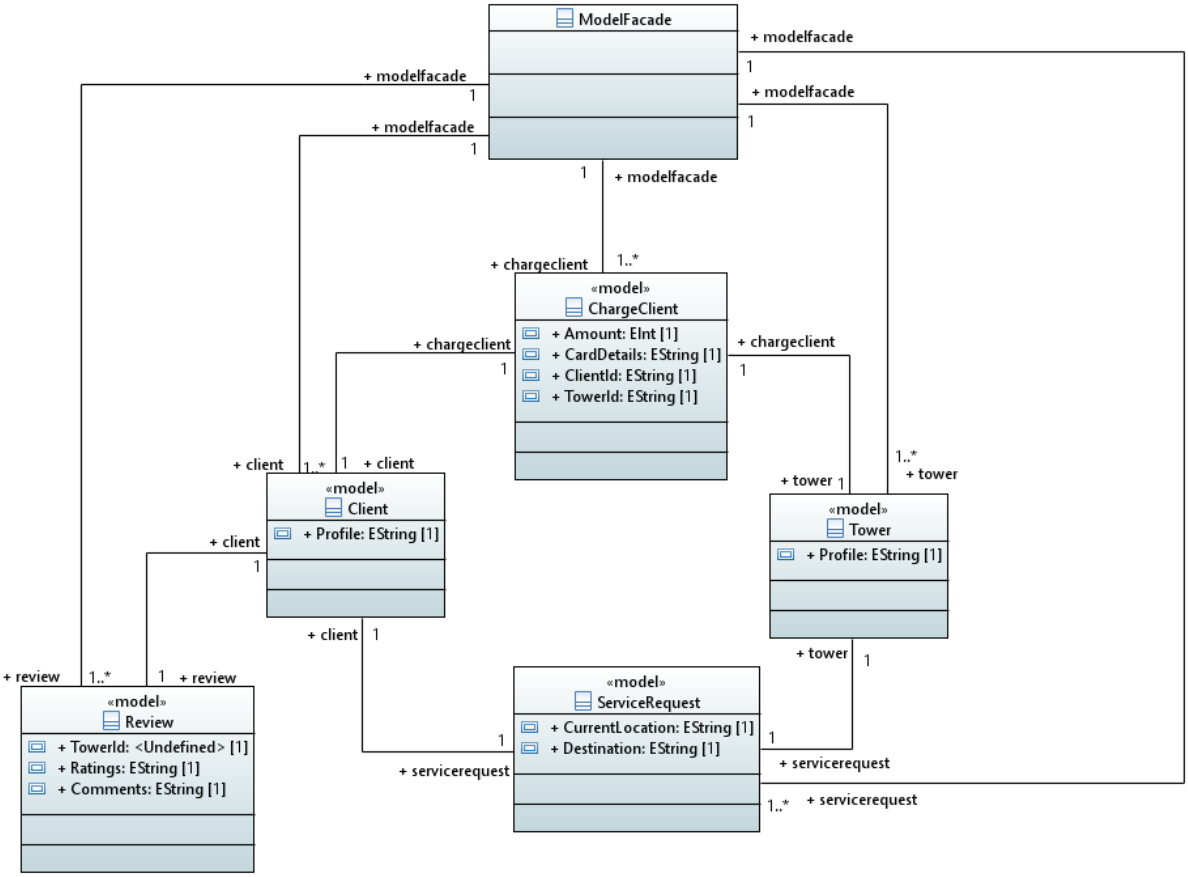


Figure:11.4.1:Model Subsystem Detailed ClassDiagram

Client.Controller Subsystem:

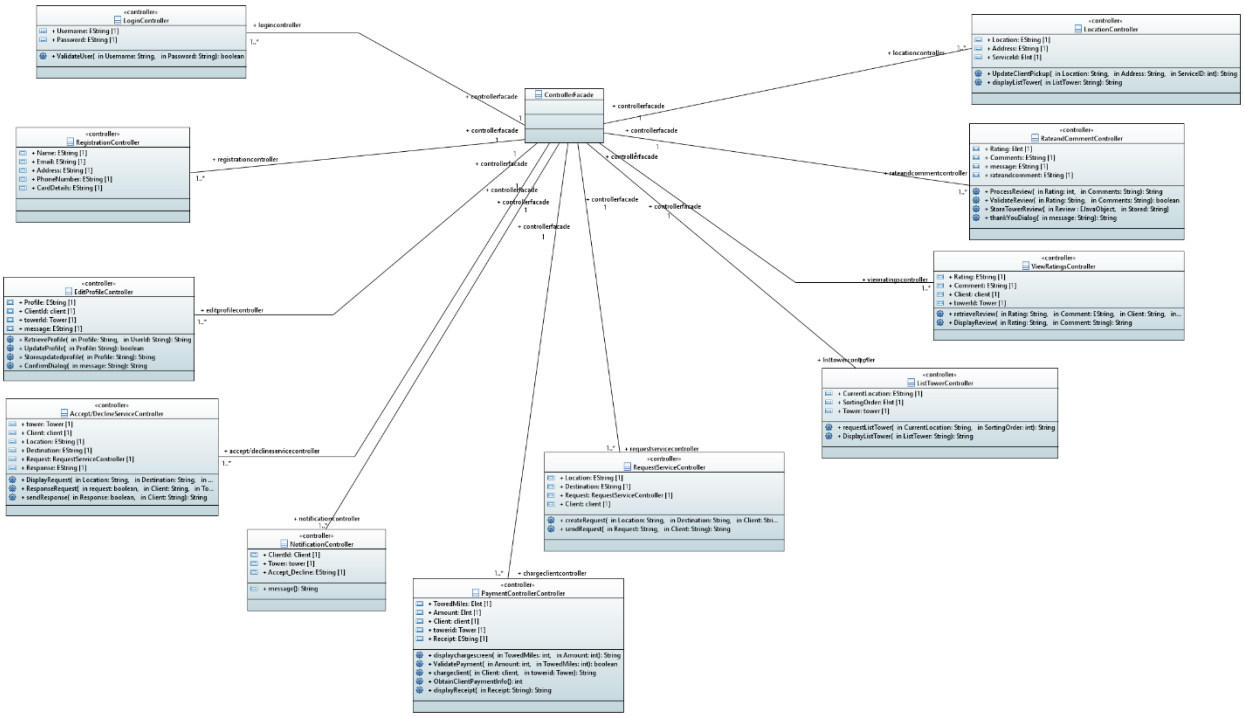


Figure 11.4.2: Controller Subsystem Detailed Class Diagram

Client.View Subsystem:

This is the top layer View subsystem with different facade interacting with each other.

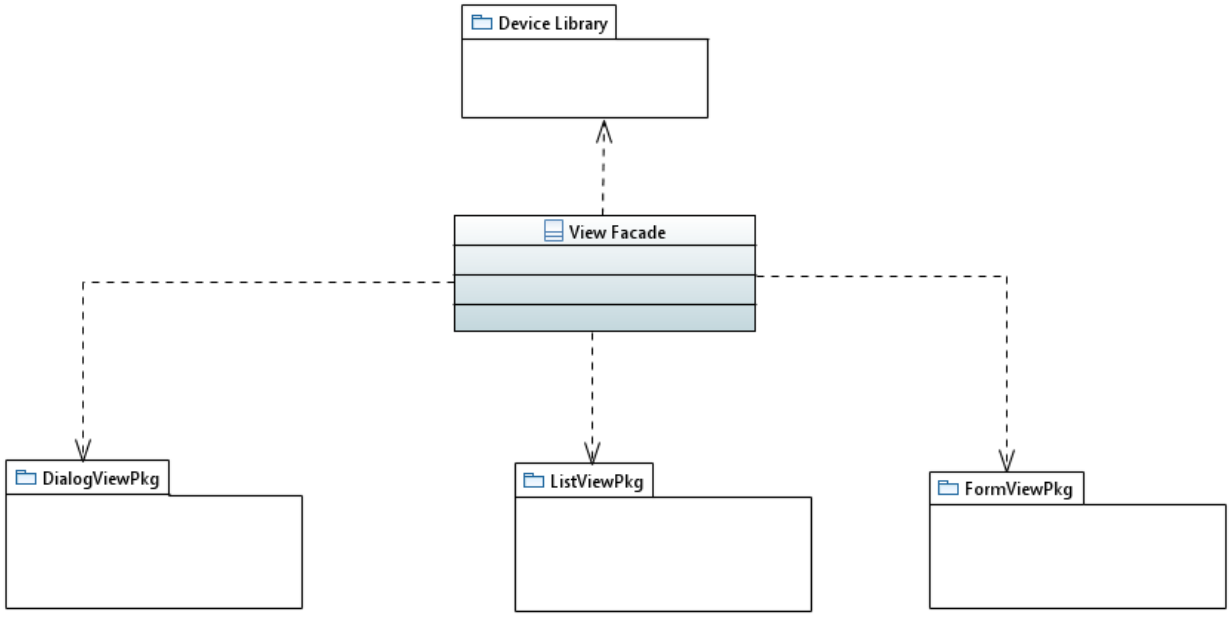


Figure:11.4.3: View Subsystem ClassDiagram

● Client.View.List View Diagram:

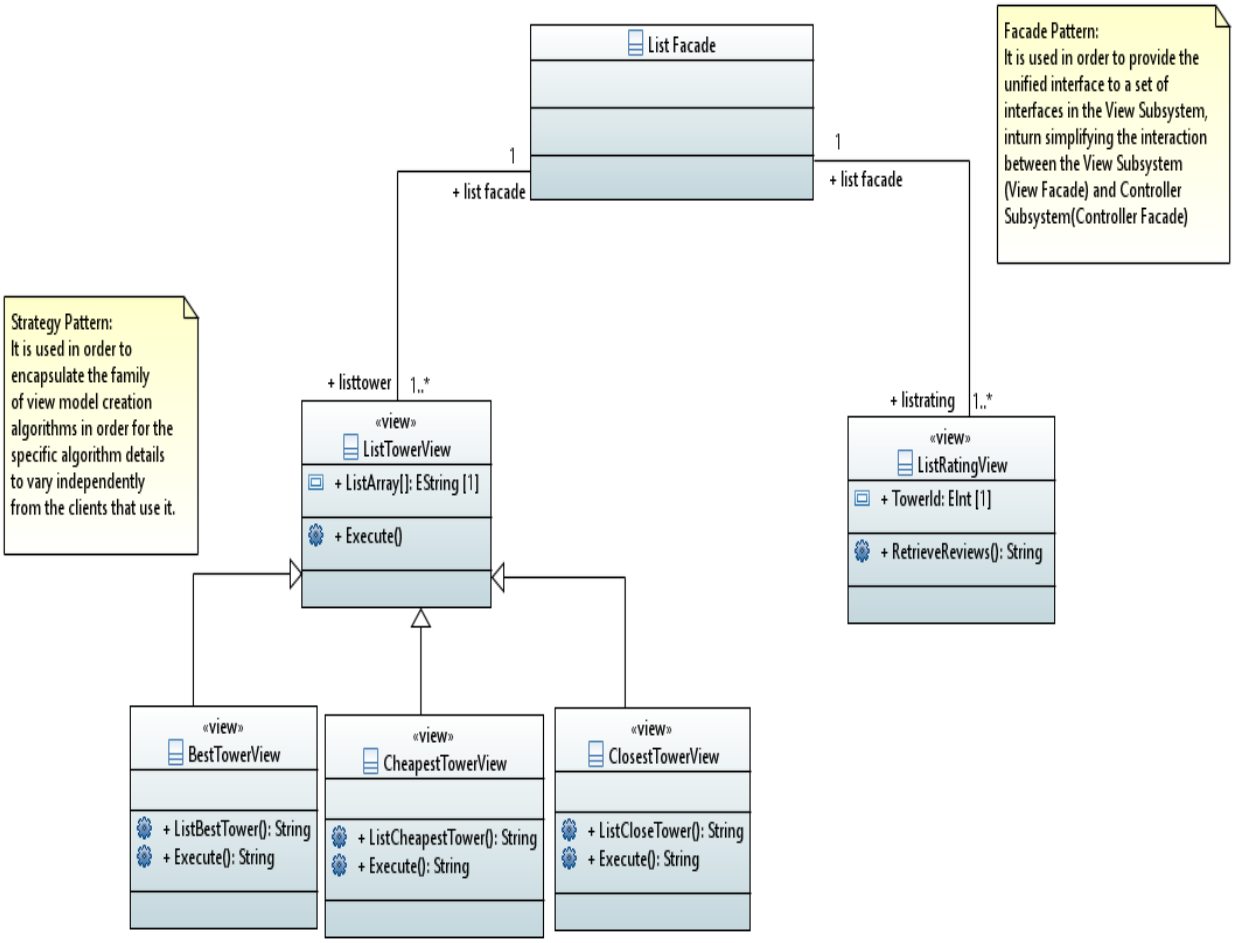


Figure:11.4.4: List View Detailed Class Diagram

● Client.View.Form View Diagram

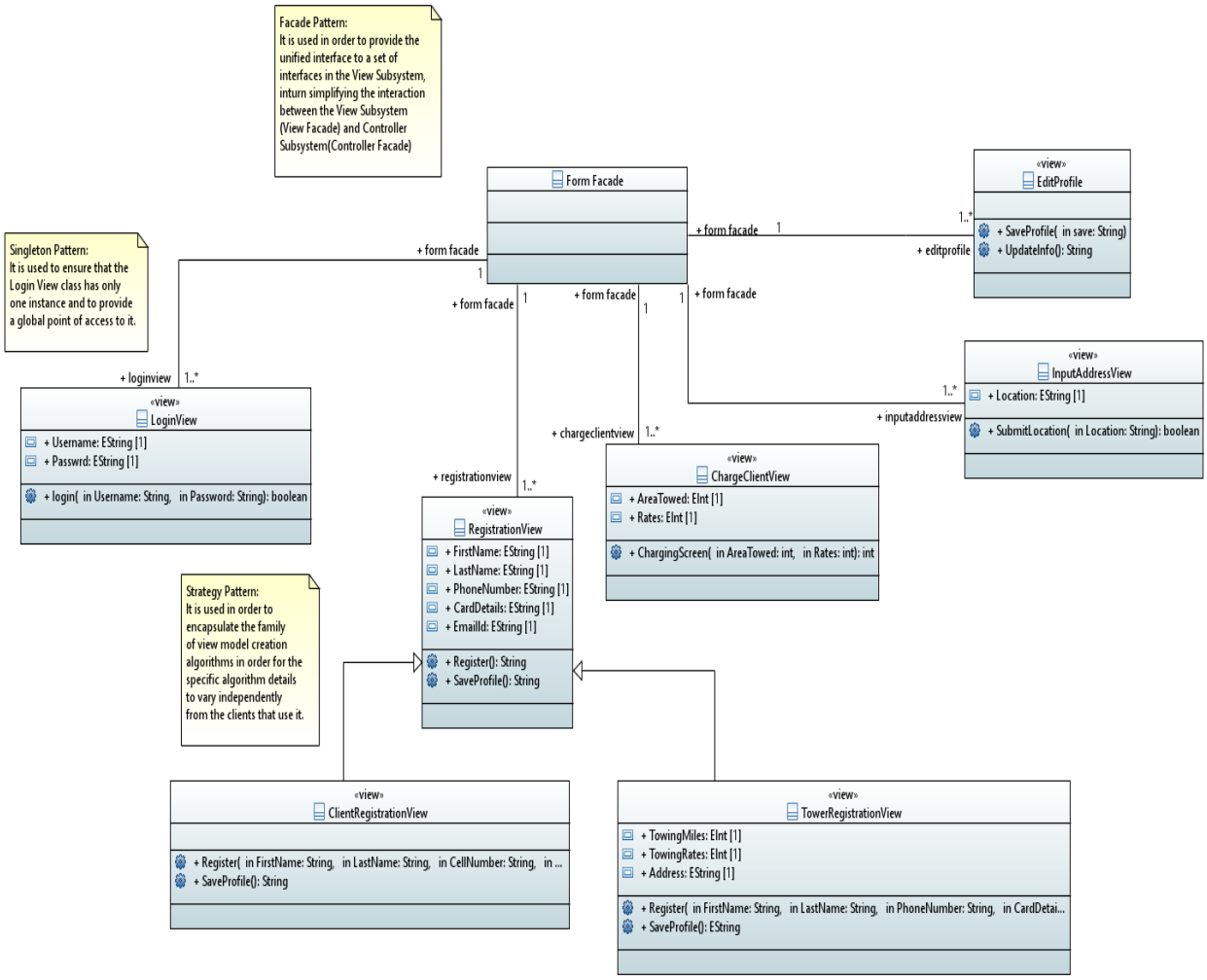


Figure:11.4.5: Form View Detailed Class Diagram

● Client.View.Dialog View Diagram:

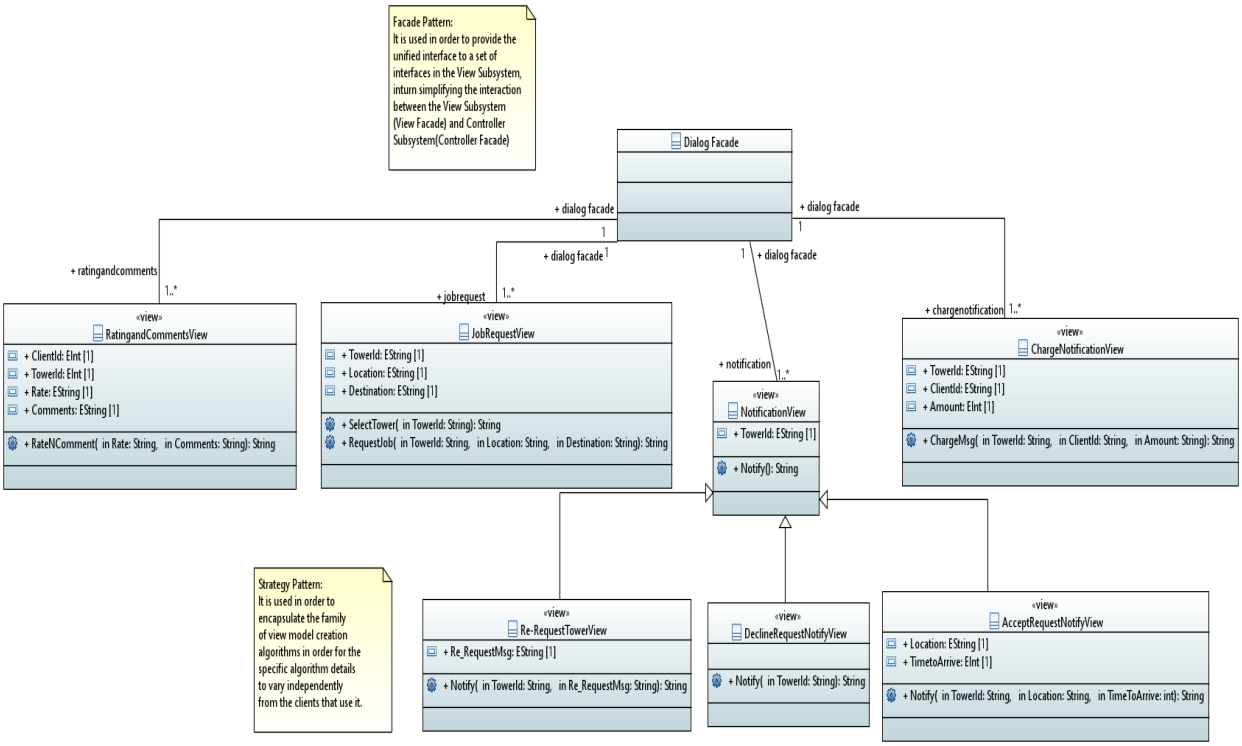


Figure:11.4.6:Dialog View Detailed Class Diagram

Presentation Subsystem Diagram:

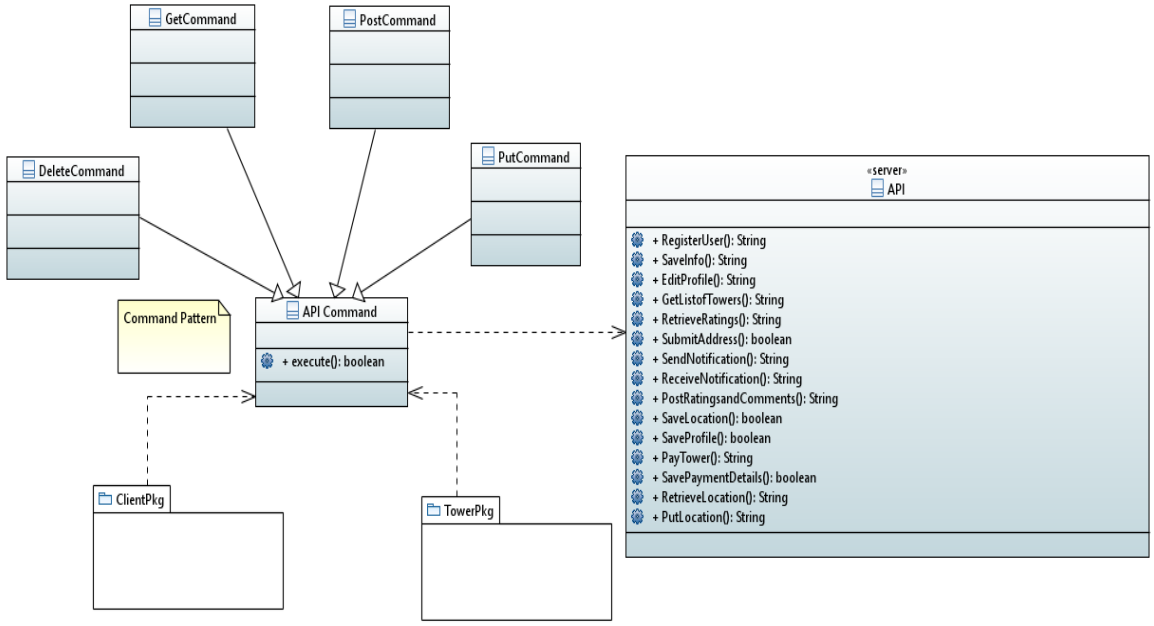


Figure:11.4.7:Presentation Detailed Class Diagram

Application Logic Subsystem Diagram:

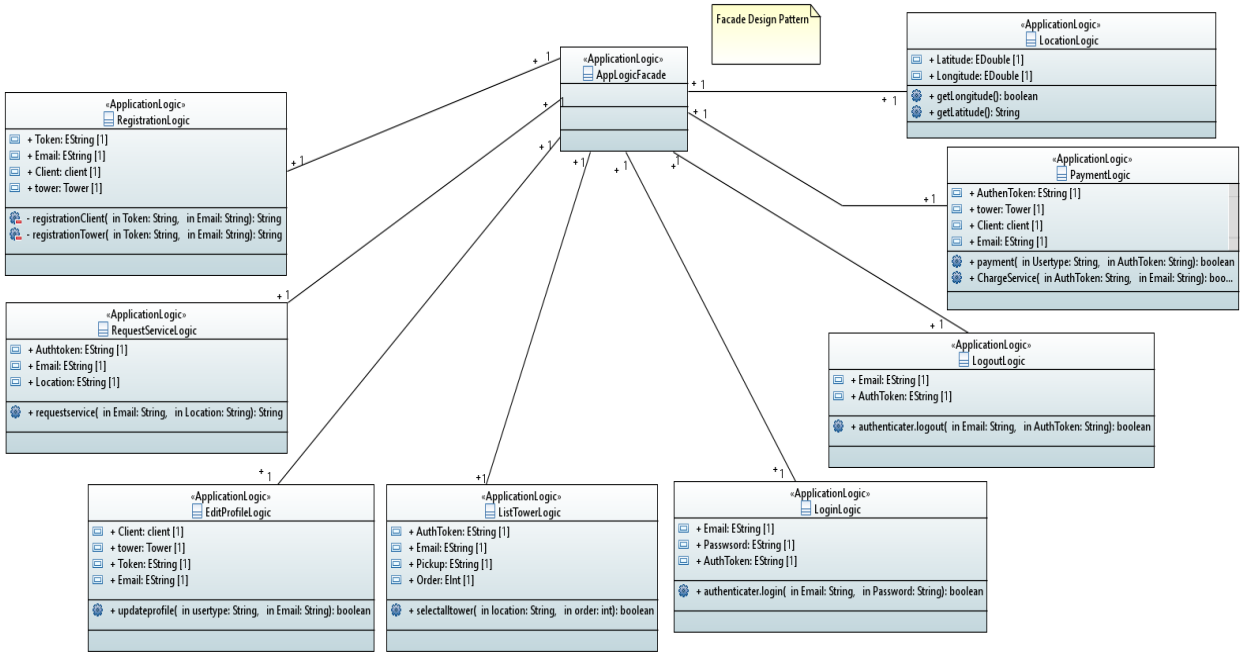


Figure:11.4.8:Application Logic Detailed Class Diagram

Storage Subsystem Diagram:

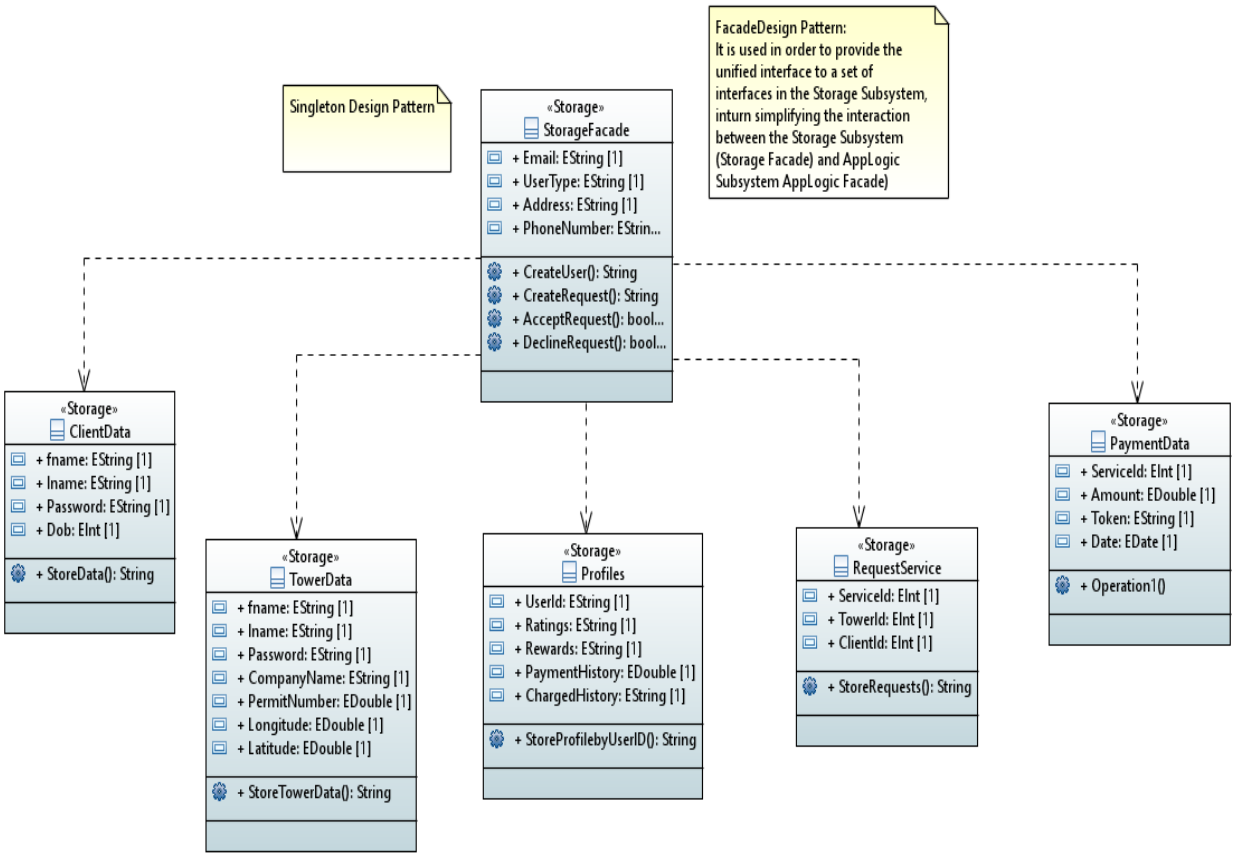


Figure:11.4.9:Storage Detailed Class Diagram

10.5. Appendix E – Class interfaces

QicFixSystem/javadoc

10.6. Appendix G – Diary of meeting and tasks for the entire semester

Session # 1
Date:
01/14/2016

Time in: 21:25
Time out:
22:40
Attendees: Steve Foo, Freny Patel, Maral Kargarmoakhar, Juan Sotomayor

Late Attendees: None

Agenda: Choose Project and declare use cases

Summary: We discussed many project ideas every team member came up with and chose the project "Mechanic/Tow GPS directory". We also defined these use cases:

User	Mechanic/Tower	Security
• Login/Logout • Register • Pick Tow/Repair • Modify Current Location/Destination • Tow List → Best Rated Tow → Closest Tow → Cheapest Tow • Repair List → Best Rated Repair → Closest Repair • Rating • Payment Referal charges Tow charges • Recommendation to friends • Choose Destination • GPS advertisement	• Login/Logout • Register → Tow → Mechanic • Ratings / Comments • Establish rates → Distance • User Profile • Specializations • Recommends	• Password encryption • SQL injection • Max password chances (5) → password reset • DDoS • Phishing attack (webpage) → HTTPS Authenticity Certificate • Payment encryption → Paypal

Table 4 Initial Use Cases Schema

Tasks: Discuss in chat group and distribute work under Peter indications: 4 use cases and 1 security use case per team member.

Session # 2

Date: 01/20/2016
Time in: 18:10

Time out: 19:10
Attendees: Steve Foo, Freny Patel, Maral Kargarmoakhar, Juan Sotomayor

Late Attendees: None

Agenda: Start documenting one use case per team member.

Summary: Steve explained with examples every field that must completed in Use Case Template shared by Peter. We also review all the homework requirements.

Tasks: Everyone must complete at least one use case, use case diagram, scenario, object diagram and sequence diagram.

Session # 3
Date: 01/24/2016

Time in: 21:30

Time out: 22:30

Attendees: Steve Foo, Freny Patel, Maral Kargarmoakhar, Juan Sotomayor
Late Attendees:
Agenda: Review first assignment of deliverable 1
Summary: We met in an online meeting where everyone showed their progress of the first assignment which includes:

- One use case (doc format)
- One use case diagram (papyrus)

- One scenario (doc format)
- One object diagram (papyrus)

- One sequence diagram (papyrus)

Everyone reported errors in the validation of sequence diagram.

Tasks: Try to solve all errors for next meeting.

Session # 4
Date: 01/26/2016
Time in: 18:30

Time out: 22:30

Attendees: Steve Foo, Freny Patel, Maral Kargarmoakhar, Juan Sotomayor

Late Attendees:

Agenda: Review Papyrus errors for class presentation tomorrow.

Summary: We helped each other to find solutions to multiple errors. We managed to solve most of the errors. Most of the errors came from wrong Class declaration and in Sequence Diagrams due to the use of "Action Execution Specification" we changed to "Behavior Execution Specification" and most of the problems were solved.

Tasks: Review each one job for consistency between:

- Use Case,

- Use Case Diagram

- Scenario,
- Object Diagram, and
- Sequence Diagram

Session # 5

Date: 01/28/2016
Time in: 18:30
Time out: 21:30

Attendees: Steve Foo, Freny Patel, Maral Kargarmoakhar, Juan Sotomayor

Late Attendees:

Agenda:
1. Review each other Papyrus errors.
2. Change to Luna Eclipse version.
3. Assign tasks in SRD.

4. Make a unified version of Papyrus Project.

Summary: By experience was better to work with Mars Eclipse version in order to work with Papyrus than Luna Eclipse version.

We made a unified version of Papyrus Project but after first validation showed up to 113 errors.

Inside a shared Google Document named "SRD-082015", containing the format of SRD, Steve established tasks for everyone.

Tasks:
- Steve, JP: create list of task for Gantt

- Freny: Gantt
- Freny, Maral: Cocomo

- Steve, JP: Hardware and Software required for Analysis Fase.
- JP, Maral: create roles for entire project.

Session # 6
Date: 02/01/2016
Time in: 16:00
Time out: 17:00

Attendees: Steve Foo, Freny Patel, Maral Kargarmoakhar, Juan Sotomayor

Late Attendees:

Agenda: Main screens for mobile application

Summary: We created the some screens for the mobile application, as shown in image above, which include:

- Login/Register,

- Change GPS location,
- List of best rated tower,
- Service Request, and

- Payment

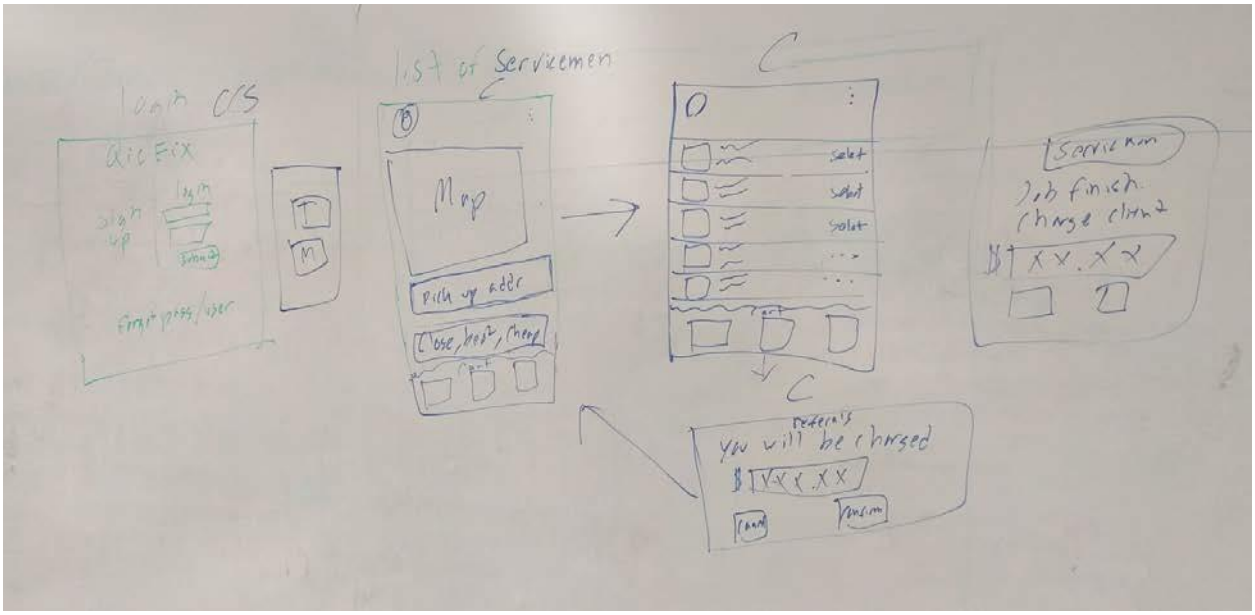


Figure 9-12 Drafts for mobile app screens

Tasks: Everyone, create draft of screens in Android Studio for meeting on 2/3/2016

Session # 7
Date: 02/03/2016
Time in: 18:30
Time out: 19:30

Attendees: Steve Foo, Freny Patel, Maral Kargarmoakhar, Juan Sotomayor

Late Attendees:

Agenda: Review draft of mobiles application screens

Summary: General review of everyone screens.

Steve presented screen made in Graphical Environment for App Screens creation. Everyone else presented Android Studio screens with default template for Nexus Mobile.

Tasks: Present these screen to Peter Clarke

Session # 8
Date: 02/11/2106

Time in: 17:20
Time out: 17:50

Attendees: Steve Foo, Freny Patel, Maral Kargarmoakhar, Juan Sotomayor

Late Attendees:

Agenda: Review SRD progress and Papyrus diagrams

Summary: Everyone reported progress in their distributed work. The average progress was 50%.

Tasks:

Maral: Introduction to chapters 3, 4, 5
Steve: Chapters 1 and 2.
Freny:
JP: Non-Functional Requirements

Session # 9

Date: 02/19/2106
Time in: 17:00
Time out: 18:30

Attendees: Steve Foo, Freny Patel, Maral Kargarmoakhar, Juan Sotomayor

Late Attendees:

Agenda: Review SRD remaining information

Summary: Steve organized and distributed the work like this:

JP & Steve: Finish Gantt/Work Breakdown in MS Project
Everyone, tasks already distributed en "SRD-08215" Google Document.

Tasks: Everyone must complete distributed work and assist to Google Hangouts meeting review next day.

Session # 10
Date: 02/20/2106
Time in: 18:00
Time out: 18:50
Attendees: Steve Foo, Maral Kargarmoakhar, Juan Sotomayor

Late Attendees:

Agenda: Merge distributed parts of SRD

Summary: Steve, Maral and Juan merged their content responsibilities in one single Google Document "Deliverable 1" in shared folder "QicFix".

Tasks: Merge remaining information:

- All Use Cases diagram
- Use Cases to be implemented diagram

- Everyone: use cases, class, object and sequence diagrams

Note: Freny Patel had a personal urgency and couldn't assist to meeting. She kept communication via Whatsapp.

Session # 11
Date: 02/21/2016

Time in: 18:00
Time out: 19:05

Attendees: Steve Foo, Freny Patel, Maral Kargarmoakhar, Juan Sotomayor

Late Attendees:

Agenda: Check SRD progress

Summary: We decided to change General Use Case and the Implemented one to meet advises from Steve.

Also we decided to keep Papyrus project as it is but for next deliverable we should merge all class diagrams into one per person.
PowerPoint presentation draft was created.

Maral, proposed two cover pages for deliverable and we chose option 2.

Tasks:
JP: Create new General Use Case with only Packages.

Maral : Prepare Disclosure agreement tomorrow.

Everyone: Finish Diagrams and merge them into "Deliverable 1" Google Document.

Session # 12
Date: 02/22/2016
Time in: 18:25
Time out: 23:15

Attendees: Steve Foo, Freny Patel, Maral Kargarmoakhar, Juan Sotomayor

Late Attendees:

Agenda: Final Review of SRD and PPT

Summary: We merged our remaining document elements

Tasks: JP Format SRD in MS Word with index.

Session # 13 Date: 03/04/2016
Time in: 15:30
Time out: 16:45
Attendees: Freny Patel, Maral Kargarmoakhar, Juan Sotomayor
Late Attendees: Steve Foo

Agenda: Assign tasks to teammates

Summary: Everyone talked about their skills and team leader assigned tasks to the group members.

Tasks: Everyone must complete assigned work and be prepared for the next meeting.

Session # 14
Date: 03/07/2106
Time in: 15:00
Time out: 16:00
Attendees: Steve Foo, Freny Patel, Maral Kargarmoakhar
Late Attendees:

Agenda: Organize the papyrus for next deliverable, Assign more tasks to groupmates.

Summary: The group members worked on making changes in their diagrams that had problem in the first deliverable. Also, we discussed about deployment diagram, and marked out questions to ask from professor Peter Clarke

Tasks: Everyone must complete assigned work and get prepared to meet and review the project for March 9th.

Session # 15
Date: 03/09/2016
Time in: 18:30
Time out: 19:00
Attendees: Steve Foo, Juan Sotomayor,Freny Patel
Late Attendees: Maral Kargarmoakhar

Agenda: Start working on Hardware and Software Mapping and also deciding on how to do the OCL.

Summary: Each team member has to do their own part for OCL. The group members started working on how to map the hardware and software parts in papyrus, and listed our questions to ask from professor Peter Clarke

Tasks: Everyone must complete the given work and attend to meet and review the project for March 10th.

Session # 16
Date: 03/10/2106
Time in: 15:30
Time out: 16:45
Attendees: Steve Foo, Juan Sotomayor, Maral Kargarmoakhar, Freny Patel
Late Attendees:

Agenda: Review the document with comments made by professor Peter Clarke.

Summary: The group discussed about how to improve the document for the next deliverable, all group members had to make changes of their own part and take care of the comments.

Tasks: Everyone must complete assigned work and help for the next meeting on March 11th.

Session # 17
Date: 03/11/2016
Time in: 15:30
Time out: 16:45
Attendees: Steve Foo, Juan Sotomayor, Maral Kargarmoakhar, Freny Patel
Late Attendees:

Agenda: Checking the new changes that has been done on the new document to verify that document doesn't have duplicate mistakes for the second deliverable.

Summary: The group discuss about how to improve the document for the next deliverable.

Tasks: Maral had to make all the changes from deliverable one and import them to deliverable two. Other team members should go through the other requirements for the second deliverable.

Session # 18
Date: 03/13/2106
Time in: 18:00
Time out: 19:00
Attendees: Steve Foo, Juan Sotomayor, Maral Kargarmoakhar, Freny Patel
Late Attendees:

Agenda: Discussing about minimal class diagram and how to prepare sequence diagrams which shows up the object interactions.

Summary: The groupmates worked on detail class design and object interactions, each team member had to do their own part and at the end add them together in one file.

Tasks: Freny had to do the minimal class diagrams and other group members had to make sure about the accuracy of their diagrams.

Session # 19
Date: 03/15/2016
Time in: 14:00
Time out: 15:30
Attendees: Steve Foo, Juan Sotomayor, Maral Kargarmoakhar, Freny Patel
Late Attendees:

Agenda: Making interfaces

Summary: Group discussed on how to prepare the interfaces for part 6.4.

Tasks: Juan made the interfaces and made a hyperlink in document as was mentioned in the document format deliverable 2. Other team members should go through the document.
Session #20
Date: 03/15/2016
Time in: 14:00
Time out: 15:30
Attendees: Steve Foo, Juan Sotomayor, Maral Kargarmoakhar, Freny Patel
Late Attendees:

Agenda: Discussing about OCL

Summary: How to prepare OCL for the controllers that we have in our mobile application.

Tasks: Everyone must complete assigned work and assist for the next meeting which is on Tuesday at 2 pm in school lab.

Session # 21
Date: 03/21/2016
Time in: 15:00
Time out: 16:00
Attendees: Steve Foo, Juan Sotomayor, Maral Kargarmoakhar, Freny Patel
Late Attendees:

Agenda: Making the State Machine.

Summary: How to making State machine diagrams in papyrus for the second deliverable.

Tasks: Everyone must go through the deliverable and Steve also had to do the State Machine Diagrams.

Session # 22
Date: 03/22/2016
Time in: 15:00
Time out: 16:00
Attendees: Steve Foo, Juan Sotomayor, Maral Kargarmoakhar, Freny Patel
Late Attendees:

Agenda: Preparing Sequence diagram for our mobile application.

Summary: All group members discuss about how to prepare their sequence diagram.

Tasks: Everyone must complete their sequence diagram with a brief description of their diagram and upload it on the google drive.

Session # 23
Date: 03/23/2016
Time in: 15:00
Time out: 16:00

Attendees: Steve Foo, Juan Sotomayor, Maral Kargarmoakhar, Freny Patel

Late Attendees:

Agenda: Checking the whole document to determine which parts are missing.

Summary: Each team member had to check the shared google drive and determine which pars are missing or need to be redone, and make notes.

Tasks: Everyone must complete assigned work and assist for the next meeting in school.

Session # 24
Date: 4/4/2016
Time in:15:00
Time out:17:00

Attendees: Steve Foo, Maral Kargarmoakhar, Juan Sotomayor, Freny Patel

Late Attendees:

Agenda: Discuss the break down of the final Deliverable. Assign each group member sections. Continue Development of project.

Summary: Maral was given sections 1 & 2 of the deliverable. Freny is going to redo the papyrus to rework the new architecture of our system. Changing from a client server primary architecture to a 4-tier architecture.

Tasks:
- Steve will begin the development of the Android application
- Juan will be testing out web server
- Freny will be working on the Papyrus
- Maral will be looking into software testing and starting sections 1 and 2 of the deliverable

Notes: May put Maral on System testing for the web app. Easiest to do.

Session # 25
Date: 4/6/2016
Time in:
Time out:

Attendees: Steve Foo, Maral Kargarmoakhar, Juan Sotomayor, Freny Patel

Late Attendees:

Agenda: Work on understanding new testing tools and ways. Steve will be explaining to everyone the differences between system, subsystem and unit testing.

Summary: We found a couple tutorials online for everyone to look at in regards to unit testing and subsystem testing. We have already made facades as one of our design patterns so subsystem testing should be easy. Juan is still working the web server. Steve is learning the new features of Marshmallow so he can develop on SDK 21 and up. Freny is trying to implement the new architecture to other UML diagram in the papyrus.

Tasks:
- Maral will be reviewing the tool Selenium for system testing
- Freny is working with the Payrus and integrating out new architecture with other diagrams.
- Steve continues to learn about android and brushes up on testing
- Juan continues to work on webserver and begin HTML coding of site.

Notes:
Derek Banas and newboston are a couple youtube channels that provide good tutorials in regards to testing**.**

Session # 26
Date: 4/8/2016
Time in:15:00
Time out:19:00
Attendees: Steve Foo, Maral Kargarmoakhar, Juan Sotomayor, Freny Patel

Late Attendees:

Agenda: Have everyone start working on their system test cases and translating them from their scenarios

Summary: Steve and Freny explained to Maral how to transform scenarios into Test cases. Juan figured out how to get proper calls from the webserver. Now web client and app can make proper calls to it and get GSON. Freny worked on papyrus minimal class diagram.

Tasks:
· Steve must write his system test cases and continue developing Android app
· Freny must write her system test cases and continue to work on Papyrus over the weekend
· Juan will work on his system test cases and will start on developing the web app
· Maral will work on her system test cases and continue to look into system testing.

Notes: Waiting on Clarke to return last deliverable so we can start making changes to final deliverable.

Session # 27
Date: 4/11/2016
Time in:15:00
Time out:18:00
Attendees: Steve Foo, Maral Kargarmoakhar, Juan Sotomayor, Freny Patel

Late Attendees:

Agenda: Review each others test cases to see if they need improving. Begin to look at unit testing, code coverage tools, Selenium and Robotium.

Summary: Some of Freny and Maral's system testing needed some improvement. Juan was explaining the web server code to us so we can begin unit testing it. Freny continues to work on the papyrus. She has almost finished with the class diagrams

Tasks:
· Steve is going to refresh on how Robotium testing is implemented. Began to make login and registration screen for app. Will try to turn them into fragments so he can tab between them
· Juan will be making Façade classes in the web server. He also coding up the front end of the web app.

· Maral needs to finish her sections 1 and 2 before the end of the week.
· Freny is taking on the papyrus in order to give Steve and Juan more time to develop.

Notes:
Juan needs to share the API commands Steve can make from the android application.

Session # 28
Date: 4/13/2016
Time in:15:00
Time out:19:00
Attendees: Steve Foo, Maral Kargarmoakhar, Juan Sotomayor, Freny Patel
Late Attendees:
Agenda: **S**tart working the final deliverable because Clarke will be sending an email with comments and critiques from the previous deliverable.

Summary: Worked on section 11 be adding use cases with non functional requirements to final deliverable. We also added the use case diagram, a few interfaces and our system test cases to the document.

Tasks:
· Steve is charge of section 4 of the document
· Maral is in charge of section 1 and 2 of the document
· Freny is working on sequence diagrams in papyrus so we can add them to the document. She is working on sections 5 and 6 of the document
· Juan is working finishing facades and developing a new project plan to add to the document.

Notes:
Presentations will be happening next week so we need to have as much of document finished so we can pull information from the document to the presentation.

Session # 29
Date: 4/15/2016
Time in:15:00
Time out:20:00
Attendees: Steve Foo, Maral Kargarmoakhar, Juan Sotomayor, Freny Patel
Late Attendees:
Agenda: Clarke send the previous deliverable. Have to make a lot of changes. Must show Clarke a demo by next class for presentation

Summary: Maral and Freny worked on the document. Freny finished the sequence diagrams and added them to the final deliverable. Maral is almost done with section 1. Steve developed a HTTP request and response class in android. Will be testing logging and logging out over the weekend. Juan continues to develop the front end so Maral and Freny can test it.

Tasks:

- Steve Develop Android so he can have at least 4 use cases done.
- Juan finish developing web app so he can have 4 use cases done.
- Freny work on the document and presentation.

Notes:

Juan and Steve have to finish development by Monday in order for there to be time to test the applications.

Session # 30
Date: 4/18/2016
Time in: 15:00
Time out:21:00
Attendees: Steve Foo, Maral Kargarmoakhar, Juan Sotomayor, Freny Patel

Late Attendees:

Agenda: Clarke issues an extension on all demo for the class. Working on finishing the presentation so we can have it ready for tomorrow.

Summary: Worked on added images from the deliverable to the slide. Had to decide what test cases to add. Chose register and edit profile because they would be the easiest to implement in the application.

Tasks:

- Everyone finished his or her parts for the presentation.
- Everyone must practice what they'll say for their parts.

Notes:

Maral will talk about use cases. Juan will introduce everyone. Steve will talk about architecture, design patterns and class diagram. Freny will talk about the test cases.

Session # 31
Date: 4/20/2016
Time in:15:00
Time out:16:00

Attendees: Steve Foo, Maral Kargarmoakhar, Juan Sotomayor, Freny Patel

Late Attendees:

Agenda: Steve and Juan have to finish working on their respective application. Everyone must learn software testing tools.

Summary: Freny and Maral are studying how to unit testing, Mockito and system testing. Steve showed off a previous project he did in Software testing to show how Mockito worked.

Tasks:
- Steve must finish the application before the end of the weekend
- Juan must finish the application before the end of the weekend
- Freny will be adding sections to the document and practice using testing tools.
- Freny will be unit testing the web server.

Notes:
Have to look into system testing recorder for the android. Also Steve needs to see if unit testing is possible in Android.

Session # 32
Date: 4/25/2016
Time in:15:00
Time out:1:00am
Attendees: Steve Foo, Maral Kargarmoakhar, Juan Sotomayor, Freny Patel
Late Attendees:

Agenda: Working on testing and trying to fix the bug in the Application

Summary: Freny worked on Testing to finish to it for the presentation. Juan and Steve working on System Testing.And fixing the bugs in the application.

Tasks:
Everyone finishes his or her part of work for the presentation and Demo.

Notes: Freeze the changes as it may affect the chanegs in System and testing process for tomorrow's demo.

Session # 33
Date: 4/27/2016
Time in:14:00
Time out:16:00
Attendees: Steve Foo, Maral Kargarmoakhar, Juan Sotomayor, Freny Patel
Late Attendees:

Agenda: Assign task and review the SRD

Summary: Guiding Maral for the test case and assigning her tasks and reviewing the doc and problems in testing.

Tasks:

Everyone tries to finish his/her part of work before next meeting.

Session # 34
Date: 4/28/2016
Time in: 13:00
Time out:20:00
Attendees: Steve Foo, Maral Kargarmoakhar, Juan Sotomayor, Freny Patel
Late Attendees:

Agenda: Assign the remaining part of the SRD and start drafting it.

Summary: Steve and Freny completed the task assigned to them and Juan fixing the Stubbing Error in Subsystem testing.
Tasks:

Everyone finishes his or her part by tonight so Freny and Juan can Document the Final SRD and submit it before 5.

Notes: Keep the consistency in Test case Format and naming conventions.