

# FLEET MANAGEMENT SYSTEM

by

ABHISHEK CHALLA

B. Tech., Jawaharlal Nehru Technological University, 2013

A REPORT

Submitted in partial fulfillment of the requirements for

MASTER OF SCIENCE

Department of Computer Science  
College of Engineering

KANSAS STATE UNIVERSITY  
Manhattan, Kansas

2016

Approved by:

Major Professor  
Dr. Daniel Andresen

# **Copyright**

ABHISHEK CHALLA

2016

## **Abstract**

Web services have become quintessential in web application development. RESTful web services are one way of providing interoperability between computer systems on the internet. REST-compliant web services allow requesting systems to access and manipulate textual representations of web resources using a uniform and predefined set of stateless operations. These services, which are online APIs, can be accessed from various applications and the results can be used to offer specific functionality to users.

This project consists of an Android app, a Server application and a Client application. The Server application exposes a REST API (Web Services developed using REpresentational State Transfer (REST) protocol) using, which the consuming client applications can make use of various functionalities as services across the network.

The Android app would be installed in the smart phone present in each vehicle of the fleet, this app would send live location data to the database using the REST API. The manager uses the client application to track the vehicles in real time, the manager can also choose to track a particular vehicle. The API could also be used to integrate the services with other systems.

This project serves to a wide variety of users, from small local businesses owning tens or hundreds of vehicle to parents who would like to track the location of their children in real time. This project aims to help the managers/owners better control and track the vehicles. Also, the exposed API could be used by other developers to customize or extend this application. This project is easy to install, use and hence friendly for users with even minimal computer skills.

# Table of Contents

FLEET MANAGEMENT SYSTEM.....	i
List of Figures.....	v
Acknowledgements.....	vi
Chapter 1 - Project Description.....	1
1.1 Introduction.....	1
1.2 Motivation.....	2
Chapter 2 - Background.....	3
Chapter 3 - Setup and Software Requirements.....	8
Chapter 4 - Requirements Analysis.....	9
4.1 Functional Requirements.....	9
Chapter 5 - System Design.....	10
5.1 Use Case Diagram.....	10
5.2 Class Diagram.....	12
5.3 Data Flow Diagram.....	14
Chapter 6 - Implementation.....	15
6.1 Output Screens.....	15
Chapter 7 - Testing.....	23
7.1 Unit Testing.....	23
7.2 Integration Testing.....	23
7.3 Performance Testing.....	25
Chapter 8 - Conclusion.....	30
Chapter 9 - Future Work.....	31
References.....	32

## List of Figures

Figure 1: Spring Architecture [4].....	3
Figure 2: Request processing workflow in SpringMVC [5].....	4
Figure 3: Use case diagram.....	11
Figure 4.1 Android Class Diagram .....	13
Figure 4.2 Class Diagram .....	13
Figure 4.3 Data Flow Diagram .....	14
Figure 5.1 Welcome Page .....	15
Figure 5.2 Registration Page.....	16
Figure 5.3 Registration successful .....	17
Figure 5.4 Login successful .....	17
Figure 5.5 Log file showing periodic updates .....	18
Figure 5.6 Admin Home .....	19
Figure 5.7 Admin Login .....	19
Figure 5.8 Web Vehicle Registration.....	20
Figure 5.9 Map Displaying the Vehicles. ....	21
Figure 5.10 Updating the vehicle using an API-key.....	21
Figure 5.11 Table depicting the API.....	22
Figure 6 Integration Testing.....	24
Figure 7.1 table depicting the throughput for different configurations. ....	26
Figure 7.2 Response Time Graph for Test Plan 1.....	26
Figure 7.3 Throughput graph for Test Plan 1 .....	27
Figure 7.4 Response Time Graph for Test Plan 2.....	27
Figure 7.5 Throughput Graph for Test Plan 2.....	28
Figure 7.6 Response Time Graph for Test Plan 3.....	28
Figure 7.7 Throughput Graph for Test Plan 3.....	29
Figure 8 Table Lines of Code .....	30

## **Acknowledgements**

I would like to thank my Major professor, Dr. Daniel Andresen for the consistent support, guidance and constructive feedback. It has been a pleasure to have been associated with him. I also thank Dr. Mitchell L Neilsen and Dr. Torben Amtoft, for taking time off their busy schedules to serve on my committee. It would have been really difficult without their support.

I would like to thank my parents Mr. Ashok Challa and Mrs. Satyavathi Challa for having the faith in me and for their love and blessings, without which I could not have achieved any of this. Special thanks to my sister Ms. Varsha Challa for being my support system. I would also like to thank all my friends who have been a part of my ups and downs.

# Chapter 1 - Project Description

## 1.1 Introduction

This project consists of a server application, a client application, and an Android app. The server application exposes a REST API which provides all the functionalities to manage a Fleet of vehicles. This API is developed using the Spring MVC architecture and uses the Hibernate framework to perform CRUD (Create, Read, Update and Delete) operations on the underlying MySQL database. This API responds to the different types of HTTP requests and communicates using the JSON format. Basic authentication is provided using the Spring Security module for the admin. The REST requests are secured using an API-key.

The Android app is installed in the smart device available in each of the vehicles belonging to the fleet. This app is responsible to register the vehicles using the IMEI of the smart device or a user defined Vehicle Id. Once the Vehicle is registered this app is responsible to send the current location data (co-ordinates) to the database using the API in a timely manner.

The Client application retrieves the current / last known location data of all the vehicles registered with the system using the API. Then the data is rendered on a Map using the Google Maps API. The manager can track the real – time location data of all the vehicles registered in the fleet.

## **1.2 Motivation**

The motivation to develop this project “Fleet Management System” is driven by two reasons.

The first and the foremost reason is my strong interest to learn the popular frameworks used in the industry to develop web applications, such as Spring MVC, Spring Security, Spring REST, and Hibernate. I also wanted to hone my mobile development skills by building an Android application. The reason to stick to Java and Android is their wide usage and acceptance across the globe.

The second reason is to create an application/ API which can be used in multiple ways. It is a common scenario where a business manager would like to track the location of all the vehicles owned by the firm. This application could be used by many businesses like delivery trucks, taxi services and public transport. Also this API can be consumed by developers who want to develop their own applications, such as an application to track all the vehicles belonging to a group of friends who are on a trip.



## Chapter 2 - Background

In order to better understand the working and implementation of this project we need to understand the frameworks and technologies used for developing this project. In this chapter I would like to describe in brief about the choices I have made and the advantages they have.

### 2.1 Spring Framework

Spring is a popular framework for developing enterprise java applications. It is one of the most widely used frameworks in the industry to develop web and enterprise applications. It is an open source platform and was initially written by Rod Johnson. The below figure depicts the architecture of the Spring framework. The core container can be considered as the heart of this framework.

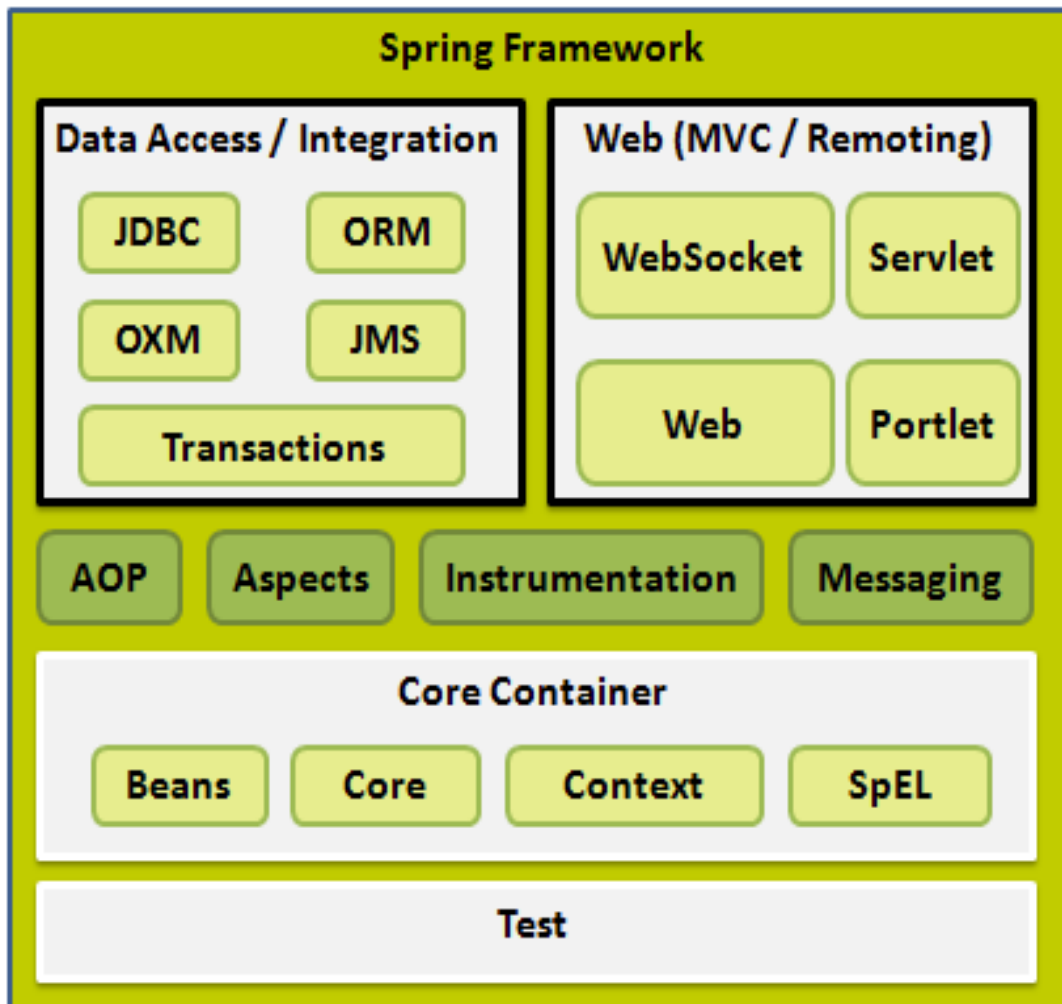


Figure 1: Spring Architecture [4]

### 2.1.1 Spring MVC

The MVC or Model View Controller is a popular design pattern used for developing web application. By using this design pattern to develop a web application we can separate the application logic from the presentation logic, thus making it easier and flexible to change and manipulate individual components with finer granularity. Spring supports this architecture with the help of a module called the Spring MVC module. In brief Model is responsible for the structure and maintenance of data, usually represented using a Plain Old Java Object (POJO), View is responsible for displaying the model data to the user with all the required formatting, the Controller is responsible for manipulating and processing the data, it could also be used to interact between the view and the model.

The sequence of events which happen when a request is sent to the Spring MVC application are as follows, the request is first received by the Dispatcher Servlet, this Dispatcher Servlet now consults the Handler Mapping and invokes the corresponding method in the associated Controller pertaining to the request. The Controller handles this request by performing the defined functionality by using the service methods and manipulating the data, once the result is achieved, the method returns a Model and View Object which contains the model data and the view name to be rendered back to the Dispatcher Servlet. Now the Dispatcher Servlet delegates the responsibility to render the view by passing the view name to the View Resolver, it also passes the model object with all the data required by the View. Finally, the View renders the resultant data with all the required formatting.

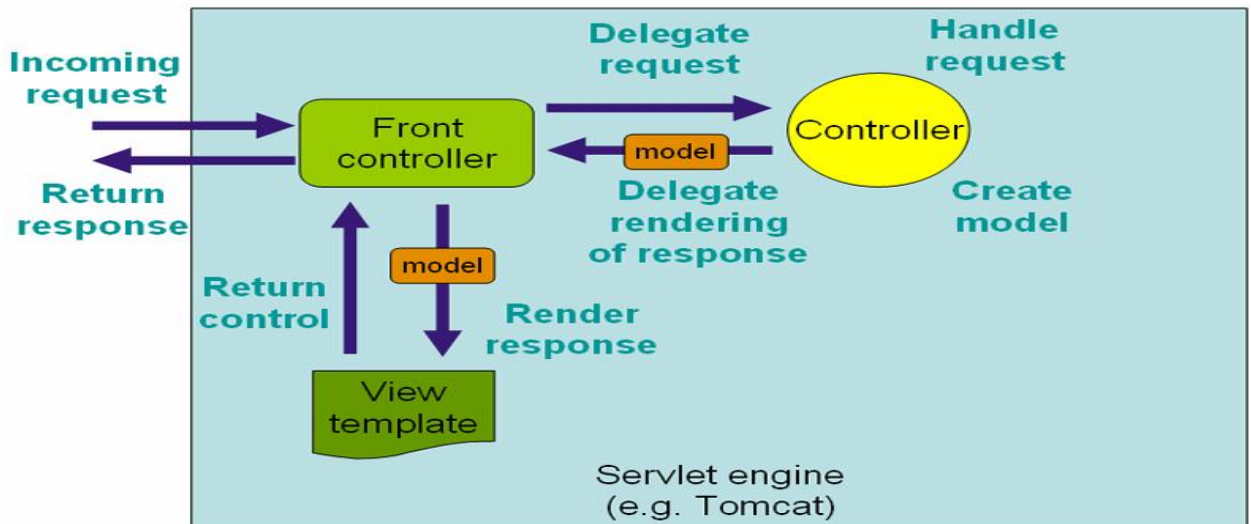


Figure 2: Request processing workflow in SpringMVC [5]

### **2.1.2 Spring Security**

The Spring Security module is responsible for providing the set of Security related functionalities and standards for the enterprise applications developed using this framework. This module addresses the two key areas of concern for the security of web applications, i.e. Authentication and Authorization. Authentication is the process of identifying and validating the true users using the system. Authorization is the process which controls what users can access what functionalities within the application.

### **2.1.3 Spring REST**

The “REST (REpresentational State Transfer) is an architectural style, and an approach to communications that is often used in the development of Web services” [1]. “A web service is a software system designed to support interoperable machine-to-machine interaction over a network” [2]. The REST architectural style describes six constraints, they are: Uniform Interface, Stateless, Cacheable, Client-Server, Layered System and Code on Demand. Spring started providing support for RESTful web services from Spring 3.0, RESTful functionality can be added to any Spring application using the Controller from Spring MVC.

The main advantages brought about by the Spring REST are

- RESTful web services are light weight and bring better performance.
- Responses from the RESTful web services could be presented in standard formats such as JSON and XML and hence can be consumed easily by any application or website.
- RESTful web services support a uniform interface since resources are accessed using the URL.

### **2.1.4 Dependency Injection**

The main and the most popular feature of Spring is Dependency Injection. We inject a dependency from outside the class, thus making that class free from any dependency. It is the Spring’s flavor of handling the general concept of Inversion of Control(IoC). While writing complex enterprise applications it is ideal to have minimal dependencies in the code so that the project would be flexible for changes and developer will not have to change the same code at multiple places for a tiny feature or requirement modification.

By minimizing the dependencies between classes and making them independent we can promote and foster code reuse of these classes and also these classes can be tested independently while performing unit testing. This Dependency Injection (DI) enables us to get rid of tight coupling in the code.

A minimal example to explain DI, consider a class A having a reference of class B within it. Now the class A is dependent on B, if the class A want to have B` instead, then we have to change the code in A or check all the places in the project where a B object is used and if there are no conflicts then change the functionality of B. Instead of going through all this hassle, we can just have a super type reference in class A and leave the instantiation to the Spring IoC container.

## **2.2 Hibernate**

Hibernate is an Object Relational Mapping framework. It is an implementation of the Java Persistence API. It is used as an alternative to the JDBC approach for storing data in the database. By using this framework, we can save the state of the object to a database and when needed create the object back from the database. By using this framework, we need not write explicit prepared statements to store the data of an object, instead we can just use the save method on the entity type objects to save the data to the database. This makes the development process very easy and saves a lot of time for the developer. Another advantage of using this framework is the seamless mapping between the classes and the database tables making the design more consistent with the class names and their underlying structure. This framework provides support to most of the features of the RDBMS such as joins, and association mappings. Spring also has inbuilt support for the Hibernate framework.

Apart from making the development process easier and faster Hibernate also offers additional advantages such as, the business logic access and deals with the objects rather than using prepared statements and dealing with the database tables, makes transaction management easier.

## **2.3 Android**

Android is the most used mobile OS in the world. It's an open source and Linux based operating system. Its syntax is very similar to Java and hence a lot of developers find it easy to start with. Android uses an underlying Linux kernel which acts as an interface between the hardware components of the device and the upper levels of the Android operating system. This

kernel basically consists of the hardware drivers for the various components such as camera, display, touchpad. The Hardware Abstraction Layer provides standard interfaces that expose device hardware capabilities to the higher-level Java API framework. Many core Android system components and services, such as ART and HAL, are built from native code that require native libraries written in C and C++ [6]. Android Runtime(ART) is written to run multiple virtual machines on low-memory devices by executing DEX files, a bytecode format designed especially for Android. The highest level is the system applications/ apps which are run on this OS and provide the basic functionalities required by the user.

## **2.4 JavaScript**

JavaScript is a high-level, interpreted programming language. It can be used for sending HTTP requests from a web page and then gather back the results. I have used JavaScript in my project for this reason. Apart from this I have also used it for form validations. The main purpose was however to provide location data for simulating live data.

## **2.5 JSP**

Java Server Pages (JSP) is the technology which is used for developing web pages that have both static and dynamic content. These pages can have embedded java code in them and thus makes them a better and flexible choice compared to servlets. The major benefit of using JSP is that multiple authors with various skill set can work on it together and achieve the required result. A UI developer need not have the knowledge of the java code and at the same time the java developer controlling the logic in the JSP need not understand the presentation code.

### **2.5.1 Advantages of JSP:**

- JSP pages easily combine static templates, including HTML or XML fragments, with code that generates dynamic content.
- “JSP pages are compiled dynamically into servlets when requested, so page authors can easily make updates to presentation code”. [7]
- Java Server Pages are built on top of the Java Servlets API, so like Servlets, JSP also has access to all the powerful Enterprise Java APIs, including JDBC, JNDI, EJB, JAXP etc.

## **Chapter 3 - Setup and Software Requirements**

The following are the tools, frameworks and software's I have used for the development of this project:

- Eclipse Mars IDE with support for Java EE.
- Android Studio.
- MySQL database.
- Apache Tomcat Server.
- REST client: Postman.
- Spring core.
- Spring Security.
- Spring MVC.
- Spring REST.
- Hibernate ORM framework.

## Chapter 4 - Requirements Analysis

Requirement Analysis is the first step in the process of developing any application. This step is necessary to have a contract between the user and the developer to stay even on the expectations. An exhaustive list is always good to start with, however, it is not really feasible in the real world. Hence the Requirements document gets updated based on the user needs, time and budget. Following are the list of functional requirements I have decided to implement for this project and hence this serve as a boundary to the scope of this project. I have come up with these requirements based on my vision for the application. I wanted an app that would allow me to track the vehicles on a map in real time and also an API. Hence these are the complete set of requirements fulfilling the description.

### 4.1 Functional Requirements

For the web application:

- The admin should be able to login.
- The admin should be able to perform Vehicle registration.
- The admin should be able to view all the vehicles on a map.
- The application should expose a service for vehicle registration.
- The application should have a service to get the details of a vehicle.
- The application should have a service to get the details of all the vehicles.
- The application should have a service to update the location of a vehicle.
- When a vehicle loses connectivity, the last known location would be displayed on the map.

For Android Application:

- The user should be able to register to the system.
- The user should be able to login.
- The user should be able to view his information on the screen after login.
- The user should be able to send live location data at a fixed interval to the server.
- The user should be able to logout.
- The app should be able to run on the background.

## Chapter 5 - System Design

Once the requirements are finalized, the next phase in the development is the System Design. Good System design forms a good foundation for a sound and stable project. The design of a system is usually depicted in the form of various UML diagrams. These are standardized diagrams and follow rules such that all the developers can stay on the same level when discussing about abstract concepts such as design.

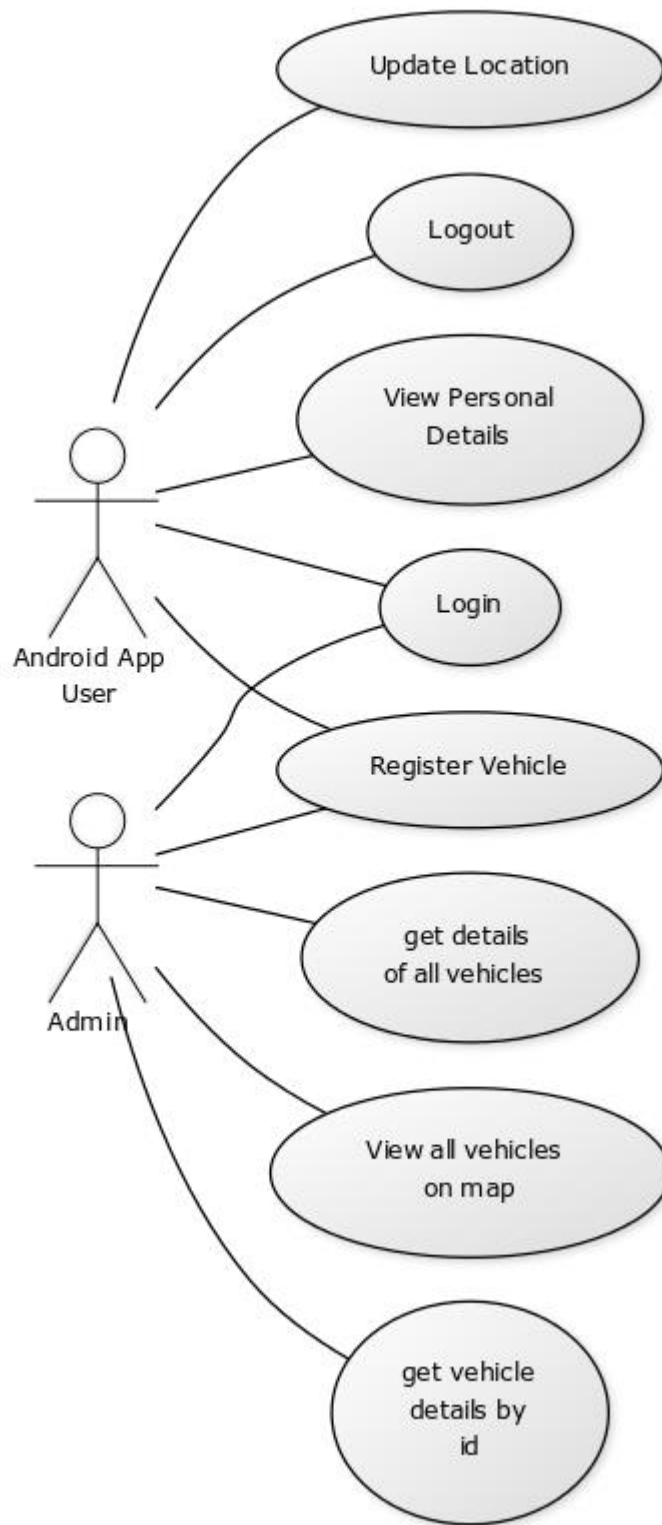
### 5.1 Use Case Diagram

Use case diagrams represent the way various actors interact with the system. Use cases provide a means to capture and depict the requirements. Use cases are simple and descriptive diagrams and hence easily understood by both the end users and the domain experts. These help for the smooth communication between the end user and the developer teams. The below use case diagram depicts the two roles Android App User and the WebApp User / Admin. The Android App user can interact with the system and achieve the following functions, they are, Register the Vehicle by entering all the required information, Login to the application, send and update the location data on a periodic basis, View the personal information.

The Web app user can use the Admin portal to login as an admin, the admin can view the location of all the registered vehicles on the map. The admin can explicitly register a vehicle. The admin can access all the services such as, retrieving the details of all the vehicles in the required format, retrieving the details of a single vehicle by id.

The below figure – 3 depicts the use case diagram for the project. It shows the two types of actors and all the different use cases using which the actors interact with the system.





*Figure 3: Use case diagram*

## 5.2 Class Diagram

A class diagram is a static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships between the classes.

The class diagram for this project contains these classes.

- **Vehicle class-** This is a model class, it is of the type Plain Old Java Object. This class has the attributes pertaining to a Vehicle in the real world and all the accessor methods used for using them.
- **KeyGen class-** The KeyGen class has the method to generate an API-key for each registered vehicle, the android app needs to save this and include it in the header every time it wants to send an update request. By doing this, we are providing an additional layer of security, and blocking all the false request hitting the server.
- **RegistrationDao class-** This class is responsible for having all the logic required to retrieve and store data back to the database. Hence, this type of classes are called as the Data Access Objects. I am using the Hibernate framework to deal with the database and hence this class has the corresponding annotations to let the framework identify the elements. For each method in the class a session is created, a transaction is performed and the session is closed.
- **VehicleController class-** This is the heart of the application. This class is responsible for responding to all the requests and hence is called the Controller class. Each method within this class is designed in such a way that it responds to a Http request. Each method is annotated with the type of Http request it needs to handle, the type of data it would return, the url it should respond to and then return the manipulated data.

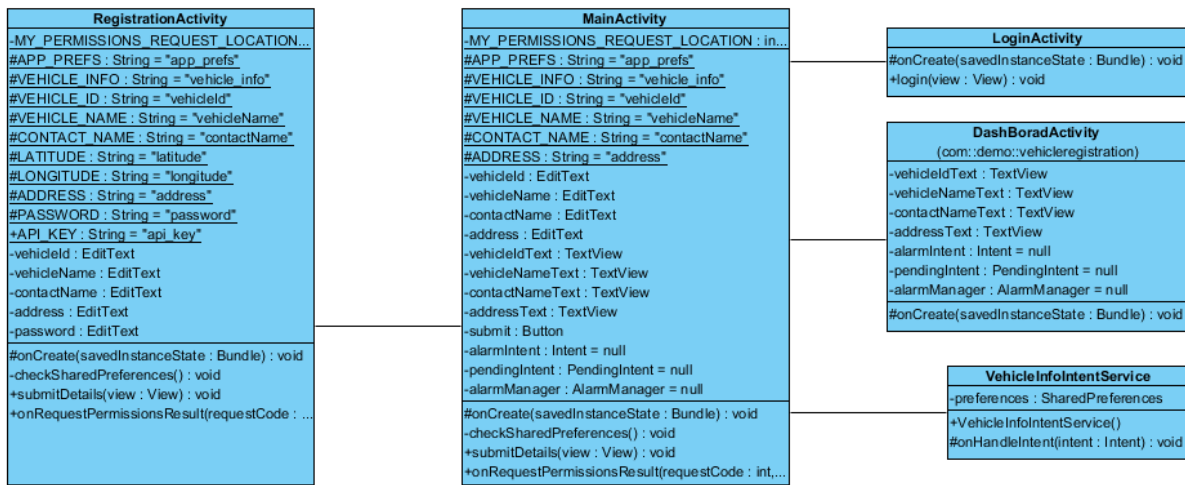


Figure 4.1 Android Class Diagram

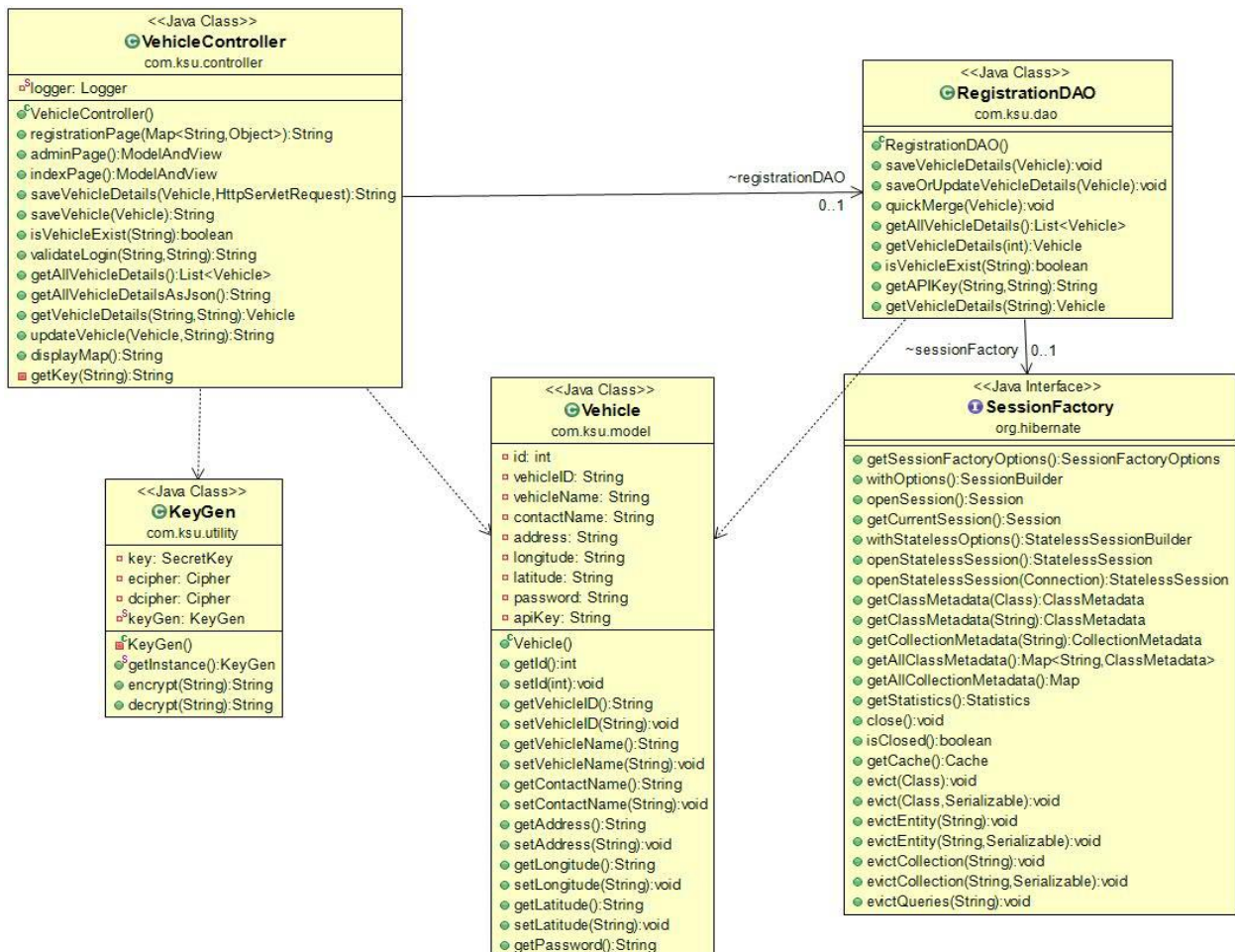
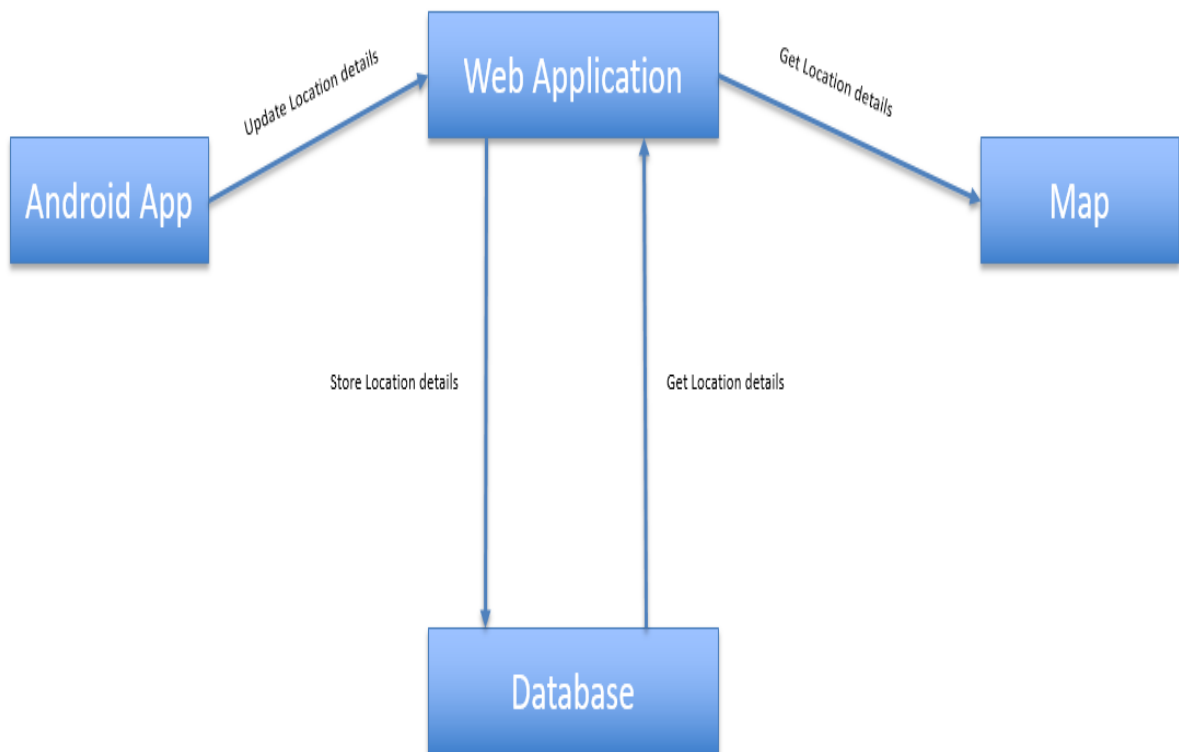


Figure 4.2 Class Diagram

### 5.3 Data Flow Diagram

A data flow diagram shows us how the data is passed between the different users/ modules of a system. I have come up with a higher level data flow diagram for the application.



*Figure 4.3 Data Flow Diagram*

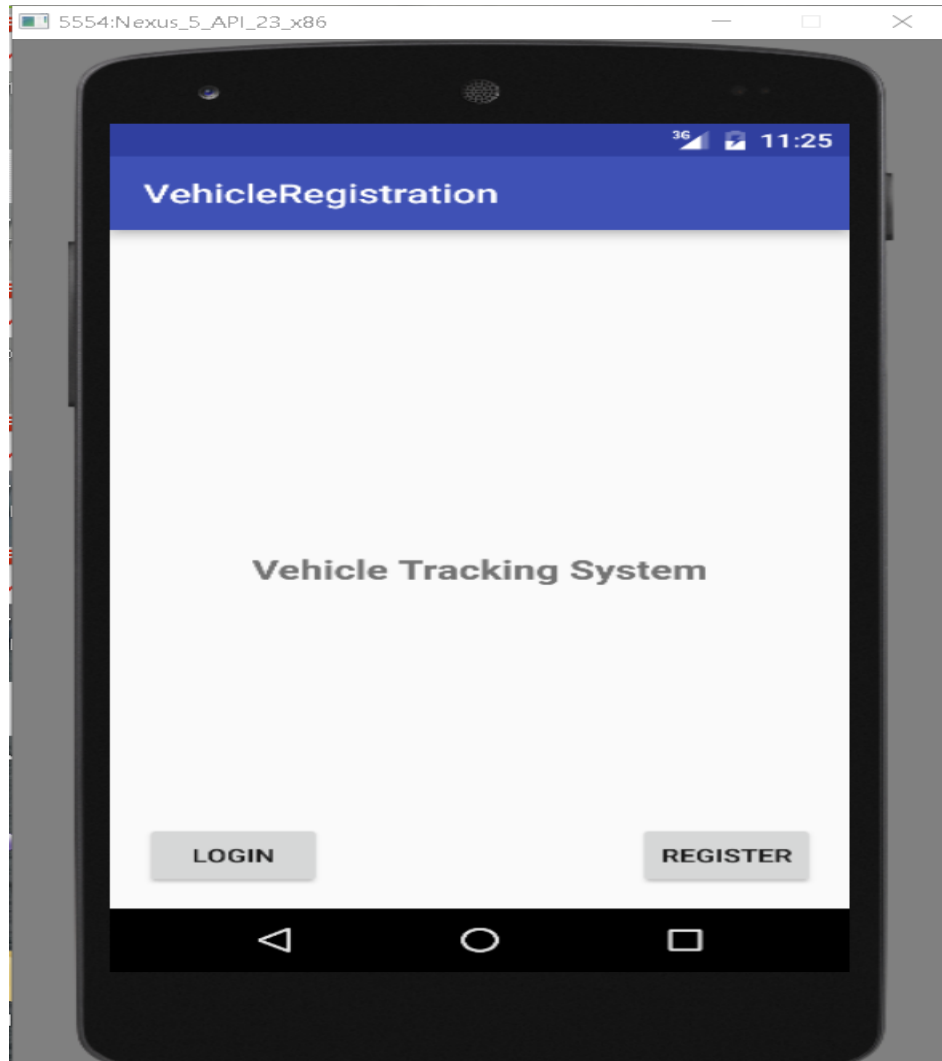
The Android App sends out an update location details request to the web application, the application handles this request and stores the details in the database. The Map requests the details of all the vehicles and the locations using the `getAllVehicleDetailsAsJson` request, the web app handles the request, by retrieving the details from the database and sending them over to the Map.

## Chapter 6 - Implementation

The fleet management system consists of a web application which exposes an API to register and track the vehicles. I have also implemented an Android app which uses the functionalities provided by this web application. I also have a web page which displays the location of the vehicles on the map and update the location of each vehicle on a periodic basis. This is the higher level implementation of the project. In this chapter I would like to describe my application with the help of some output screens.

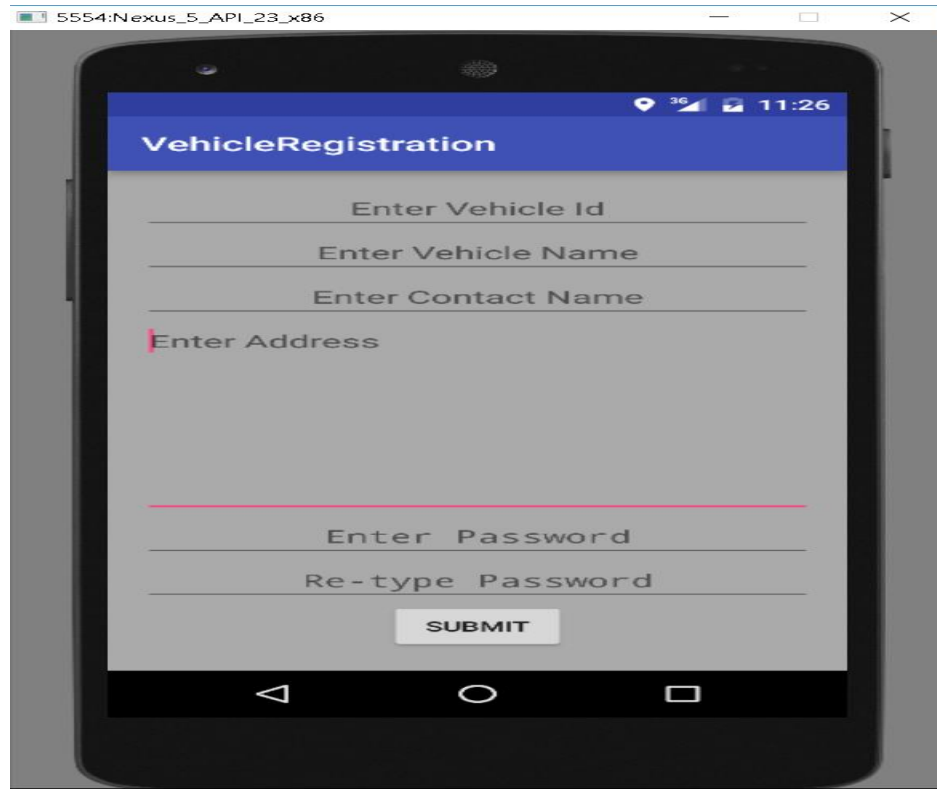
### 6.1 Output Screens

Below are the output screens for the Android application.



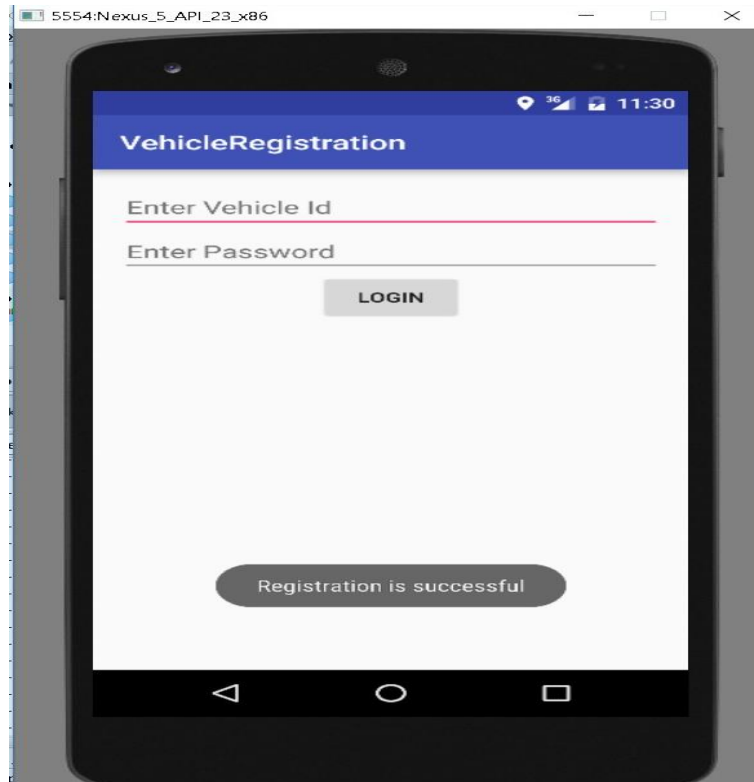
*Figure 5.1 Welcome Page*

The first output screen i.e. figure 5.1 shows us the welcome page of the Android app, it has two buttons to navigate to the Vehicle registration page and the Login page. This page is displayed to the user upon the launch of the app.

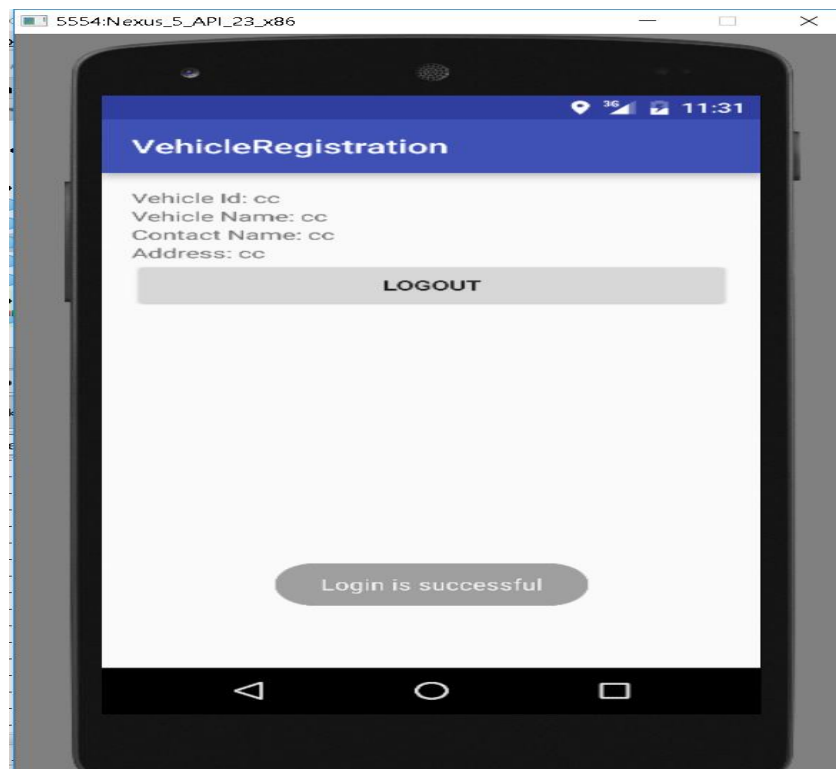


*Figure 5.2 Registration Page*

The second output screen - figure 5.2, shows the registration page, it has all the fields required for the user to register, once the user enters all the values and if they are valid, then the user will be shown a message that the registration is successful. Now the user is redirected to the home page. This is shown in figure 5.3.

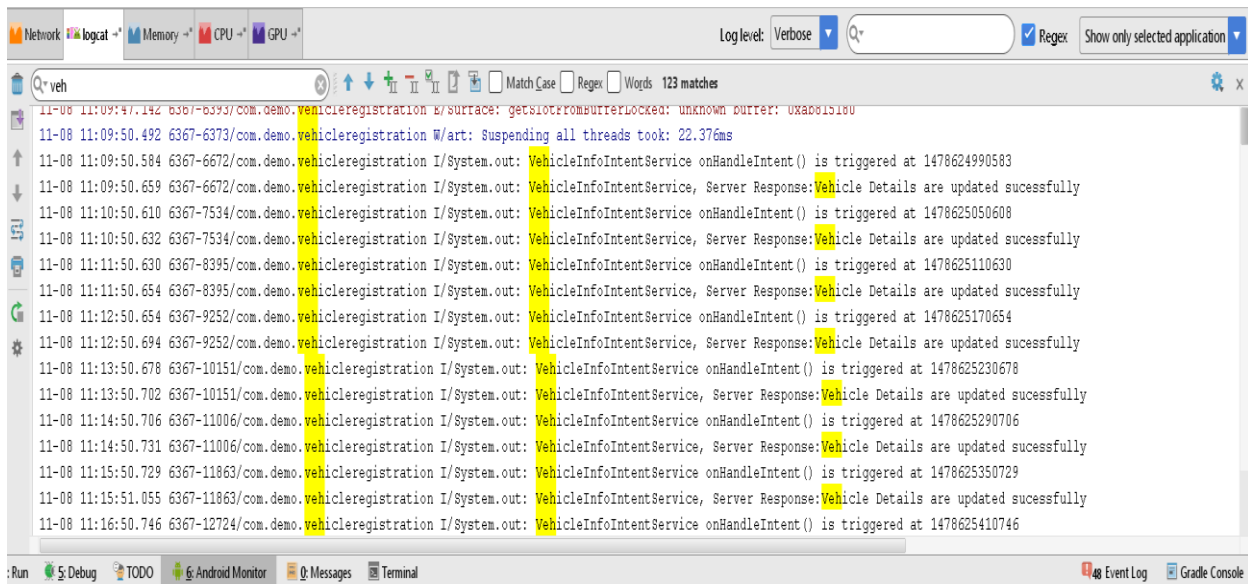


*Figure 5.3 Registration successful*



*Figure 5.4 Login successful*

The user can proceed to login with the credentials created, if the user attempts to login with valid credentials, then the user can successfully login to the system and the web application will send the user an API-key which has to be stored, and sent along as a header every time with the update location request. This will help us to avoid any malicious update requests. Once the user logs in, he will be redirected to a page with the vehicle info shown in figure 5.4, the user can now choose to stay on this page or move to a different app, no matter what our app would be run on the background and keeps updating the data base in a timely manner. Once the user has reached the destination he can choose to logout and that is when the updates would be stopped.



*Figure 5.5 Log file showing periodic updates*

The figure 5.5 shows us a console log, which shows that the alarm manager is invoking the vehicle location update service at regular intervals, which is in turn sending a put request to the server. Thus it could be observed that the Android app is sending periodic updates with the current location the web server. The Android app sends out requests when it is active and also when moved to the background, i.e. even when another app is currently operating. In the above figure, a request is being sent every one minute, as I have configured the Android app in such a way. This could be easily changed by re-configuring the Alarm Manager within the Android app.



The web application has spring security enabled for admin role, so the user has to provide the admin credentials to access any page within the system. Once the user provides the valid credentials, the user would be redirected to the admin home page. Figure 5.7 depicts the Admin Login page, the user accessing any path within the project would be first redirected to this page, upon successful Login the user would be redirected to the Admin Home displayed in figure 5.6. Here the user can choose to perform the web vehicle registration, View vehicles on map or logout

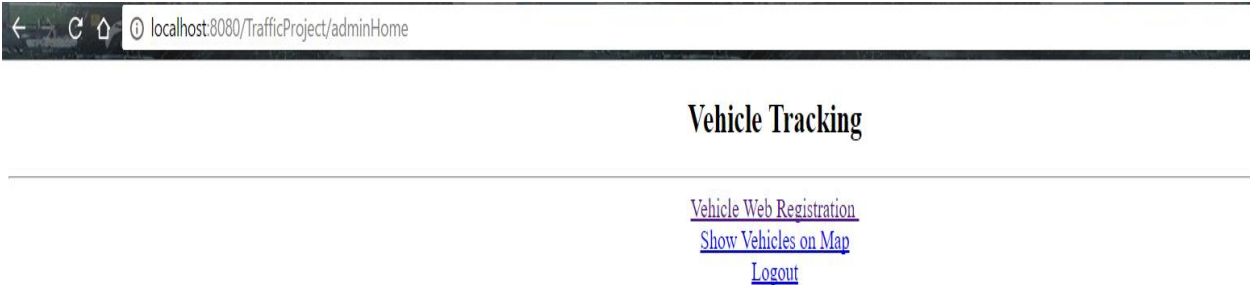


Figure 5.6 Admin Home

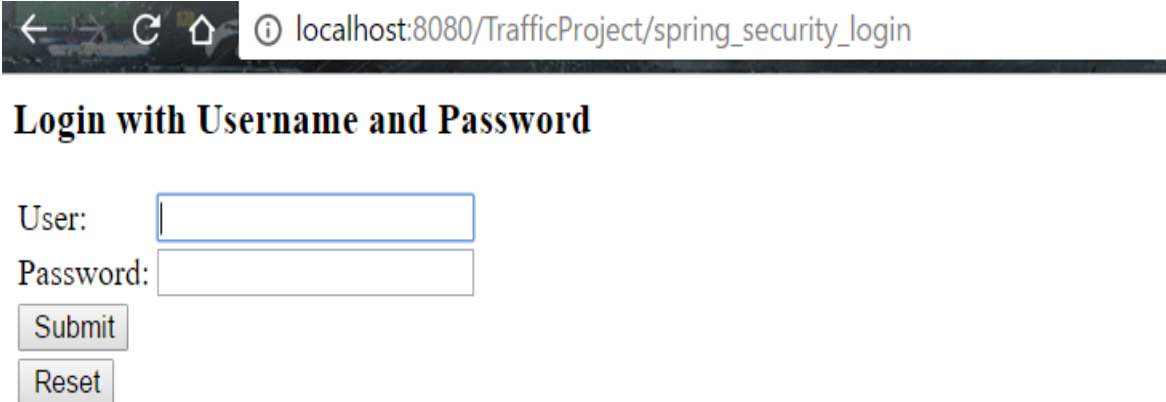
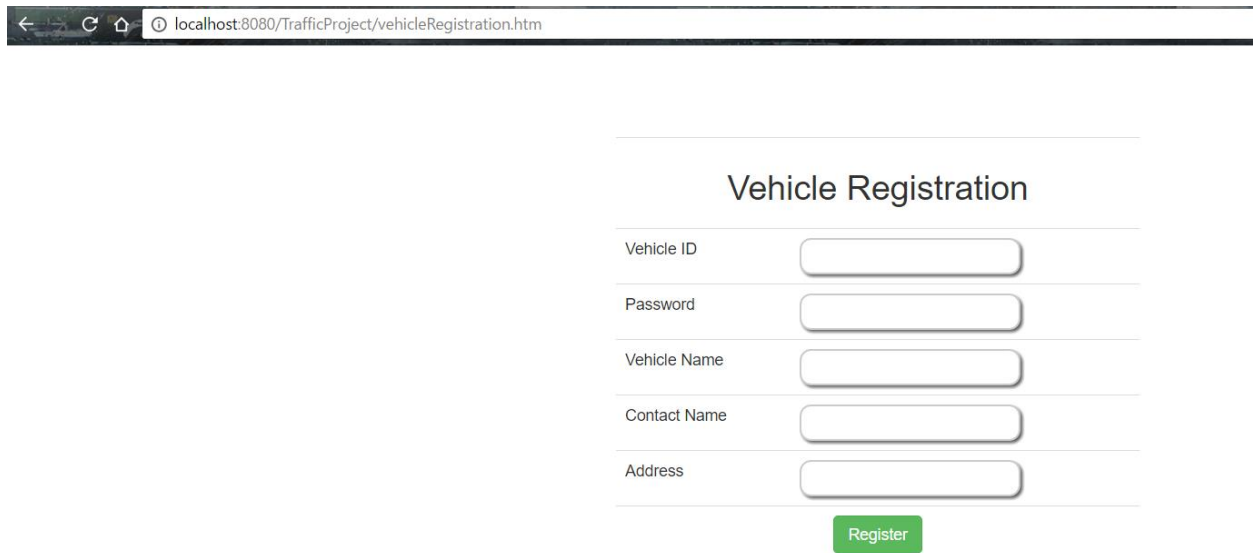


Figure 5.7 Admin Login

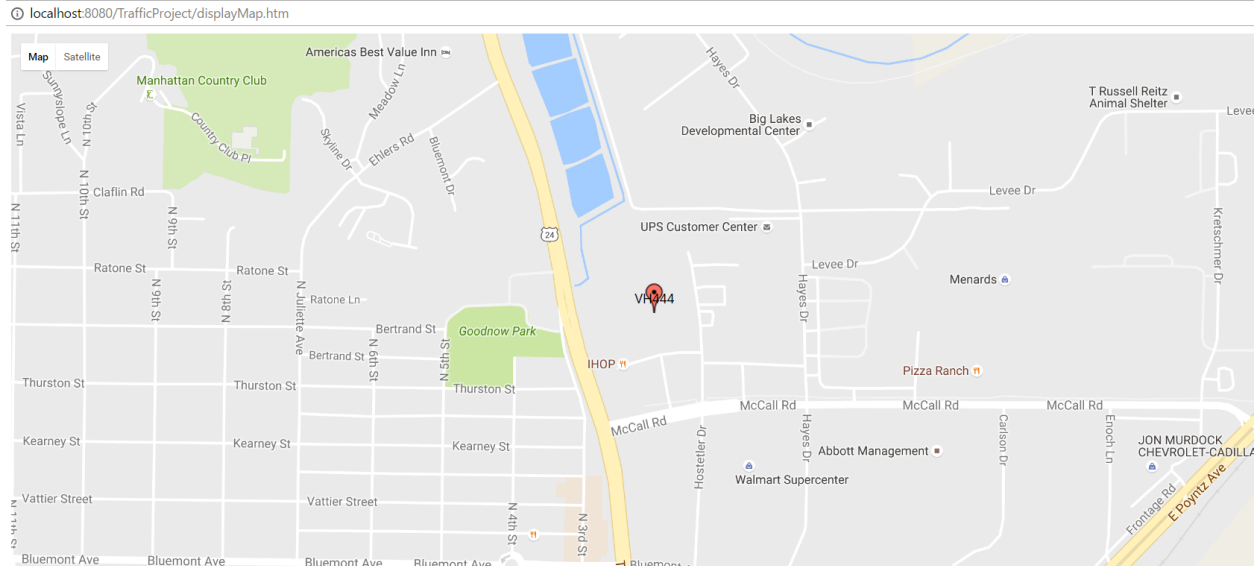
If the user chooses to perform web vehicle registration he would be redirected to a page which would look like, figure 5.8. This page has all the fields necessary for the registration of the vehicle. If an Admin would like to register a vehicle from the back end, he can do so by making use of this page.



The image shows a web browser window with the address bar displaying 'localhost:8080/TrafficProject/vehicleRegistration.htm'. The main content area features a form titled 'Vehicle Registration'. The form consists of five input fields, each with a label to its left: 'Vehicle ID', 'Password', 'Vehicle Name', 'Contact Name', and 'Address'. Below these fields is a green button labeled 'Register'.

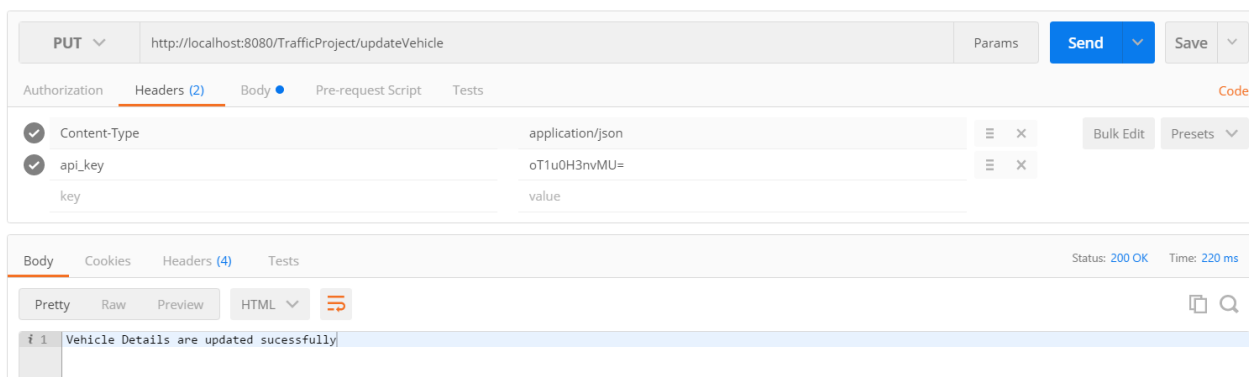
***Figure 5.8 Web Vehicle Registration***

If the user wants to view the vehicles on the map, he can do that by clicking on the displayMap and this page shows us all the vehicles currently registered in the system, on the map. The map is refreshed every 2 seconds to reflect any changes. I have made use of google maps API to display the vehicles using a marker and label on the map. This is achieved by using the getAllVehicles service provided by the server. This is displayed in figure 5.9.



**Figure 5.9 Map Displaying the Vehicles.**

Once the user is done monitoring he can choose to logout. The user can also access all the services provided by the server, such as `getAllVehicleDetailsAsJson` which would give the vehicle id and location of all the vehicles, in the JSON format. The user can also get the details of a particular vehicle by specifying the vehicle id. Figure 5.10 depicts an update request performed using the Postman REST client. The API-key is passed through the header. The response shows a success message.



**Figure 5.10 Updating the vehicle using an API-key**

I have tried to keep the user interface as simple and clean as possible. I believe that a simple and compact user interface is a key factor for the usability of the application. If an interface is simple to use it will be usable by more number of people and hence serves the purpose. I have installed the Android application in my device and checked the look and feel. I have also asked for feedback from my friends. I have taken into consideration the inputs from them and made the corresponding changes.

The main motivation was to have a user interface using which all the functionality can be accessed and also to expose a comprehensive API implementing the most required and useful functionality.

This will help other developer to build systems using this API.

<b>URL</b>	<b>Method Type</b>	<b>Description</b>
/VehicleRegistration	GET	Displays the form for vehicle registration
/admin	GET	Displays login page for admin
/saveVehicle	POST	The passed vehicle is saved
/isVehicleExists/{VehicleID}	GET	Returns true if vehicle exists
/validateLogin/{vehicleID}/{password}	GET	Returns the API-Key if the credentials match
/getAllVehicleDetails	GET	Returns the location details of all the Vehicles
/getVehicleDetails/{vehicleID}	GET	Returns the details of the required vehicle
/updateVehicle	PUT	Updates the location of the vehicle, provided the correct API-Key is passed

*Figure 5.11 Table depicting the API*

## Chapter 7 - Testing

Once the development phase is completed, the next important phase is the Testing phase. Any application should be thoroughly tested so that the user can be assured of a functioning and reliable product. Hence testing is a non-trivial activity and should be performed with enough time and effort. Testing cannot establish that a product functions properly under all conditions but can only establish that it does not function properly under specific conditions. Our goal is to attain the expected outcomes in the most scenarios. Various types of testing need to be performed such as, unit test cases to test each class, integration testing to check how all the modules are binding, performance testing to evaluate the performance of the system and finally user acceptance testing to see if the user has received what he has accepted.

### 7.1 Unit Testing

Testing the application at class level or method level helps us to understand if each of the building blocks are functioning as expected, so that there would not be a problem when each of these individual elements collaborate to perform the higher level functionalities of the system. Unit testing could be performed by sending mock requests to each of the methods in the controller class. A basic example test method would have the following code

```
mockMvc.perform(get("/")).andExpect(status().isOk());
```

So using the mock objects and passing test data we can test each of the methods in the controllers. All the exported services have been tested using the above said procedure. The same procedure has been followed to test the data access methods in the DAO classes. The keyGen class has been tested using a Junit test case.

### 7.2 Integration Testing

Once all the modules are put together we can test the higher level functionalities of the system, this phase is called the integration testing.

<b>Test Case</b>	<b>Expected results</b>	<b>Actual Result</b>
Admin Login with valid user id and password.	User redirected to admin page.	Pass
Admin login with invalid user id and password.	Error message is displayed.	Pass
Admin clicks on Vehicle Registration hyper link.	Redirected to Vehicle Registration page.	Pass
Admin enters all the required fields with valid data.	Vehicle is successfully registered.	Pass
Admin does not enter all the fields for vehicle registration.	Error message is displayed.	Pass
Admin clicks on View vehicles on Map hyperlink.	The map is displayed with markers for all the vehicles.	Pass
Admin should be able to update the vehicle location with valid API-key.	The co-ordinates are updated successfully.	Pass
Admin tries to update the location of the vehicle without a valid API-key.	The co-ordinates are not updated. Error code is returned.	Pass
Admin attempts to retrieve the details of all the vehicles.	Receives the location details of all the vehicle.	Pass
Admin attempts to retrieve the details of all the vehicles in json format.	Receives the location details of all the vehicle in json format.	Pass
Android user registers with valid data.	User registered Successfully.	Pass
Android user attempts login with valid data.	Successfully logs into the system.	Pass
Android user attempts login with invalid data.	Error message is displayed.	Pass

*Figure 6 Integration Testing*

### **7.3 Performance Testing**

It is important that we need to test the application for performance. Good performance ensures that the user has good availability of the services even under the load. I have used Apache JMeter tool for load and performance testing. This tool simulates a group of users sending requests to the application server and returns various statistics that show the performance of the server in the form of table data and corresponding graphs.

A Test Plan in JMeter has the configurations required for running a test, in test plan we specify the thread group or the number of users, the number of times to loop, the ramp up time, the url to hit, the path to the exact resource, the port number, the parameters to be passed to the request, headers if any, the type of request and the type of content. Once all the parameters are configured we can execute this test plan whenever we want.

I have performed performance testing to the most important service in my project, i.e., the service to retrieve the data of all the vehicles. I have chosen this service as it requires the most database resources and deals with relatively more amount of data. I have used a laptop with 8GB RAM and ran an instance of the project in the tomcat server. I have used one of the campus computers to perform load testing, by sending the request over the Wi-Fi. The data transferred per request is 11.14 Kb.

Test plan 1 simulates a thread group of 100 users, a ramp up period of 1 seconds and a loop count of 10. From the response time graph for test plan 1 we can observe that the response time increases, and through put remains about constant. This test plan simulates 1000 requests and tries to send out the 1000 requests by the end of 1 second.

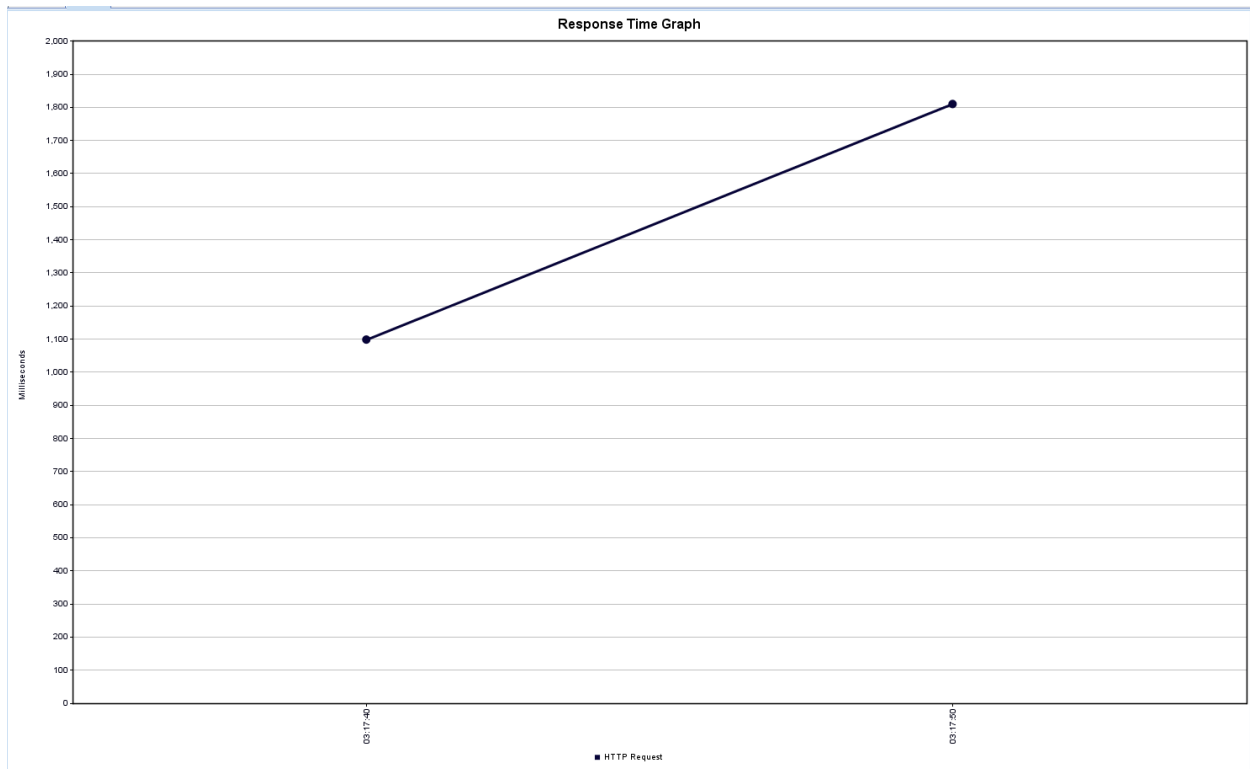
Test plan 2 has a thread group of 200 users, a ramp up period of 1 second and a loop count of 10. From the response time graph for test plan 2 we can see that the response time spiked at the middle. The deviation in the through put is less, so the performance is acceptable. This test plan simulates a bigger number than the first test plan, it simulates 2000 request and has a ramp-up period of 1 second.

Test plan 3 has a thread group of 500 users, a ramp up period of 1 seconds and a loop count of 10.

Response time graph has ups and downs, I think this is attributed to the concurrent operations on the database. The through put is 3200 pm. This test pan sends out even more requests compared to the previous test plan. This plan simulates a total of 5000 requests and has a ramp up period of 1 second.

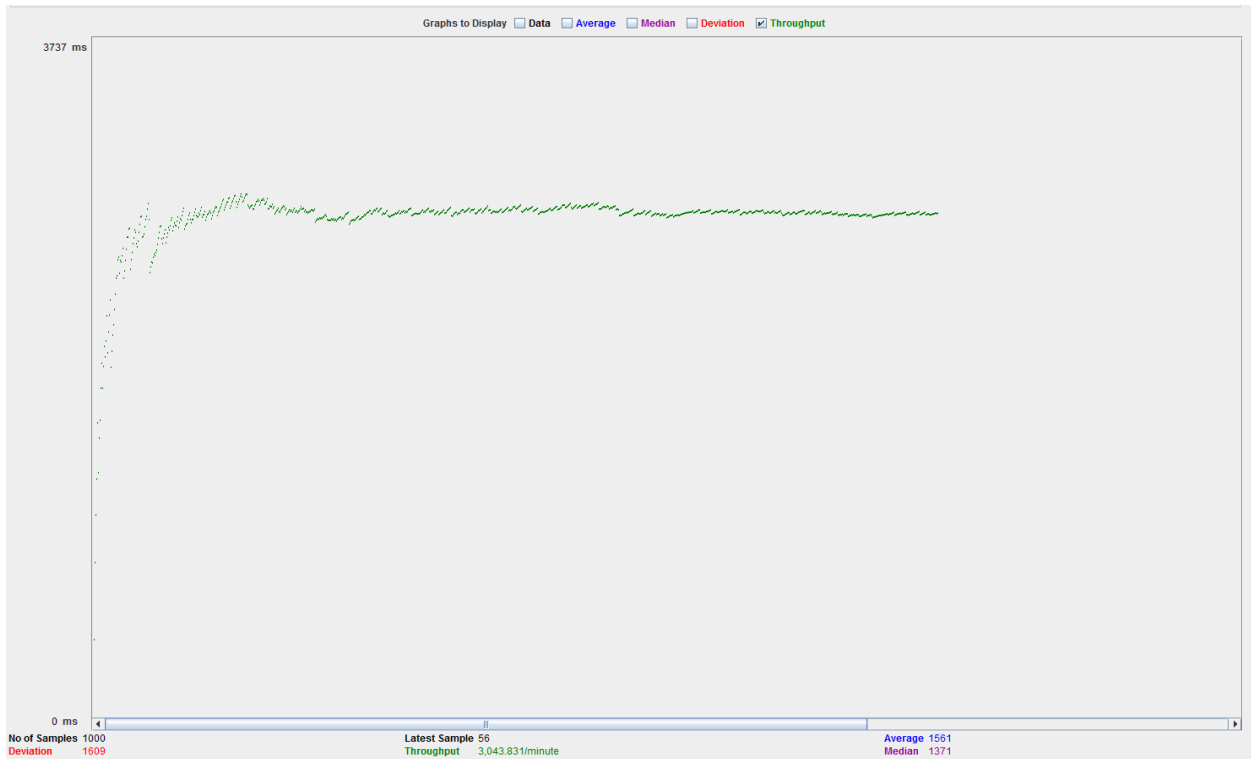
No. of Users	Ramp up period	Loop Count	Throughput
100	1	10	3043.831/minute
200	1	10	2906.132/minute
500	1	10	3211.991/minute

*Figure 7.1 table depicting the throughput for different configurations.*

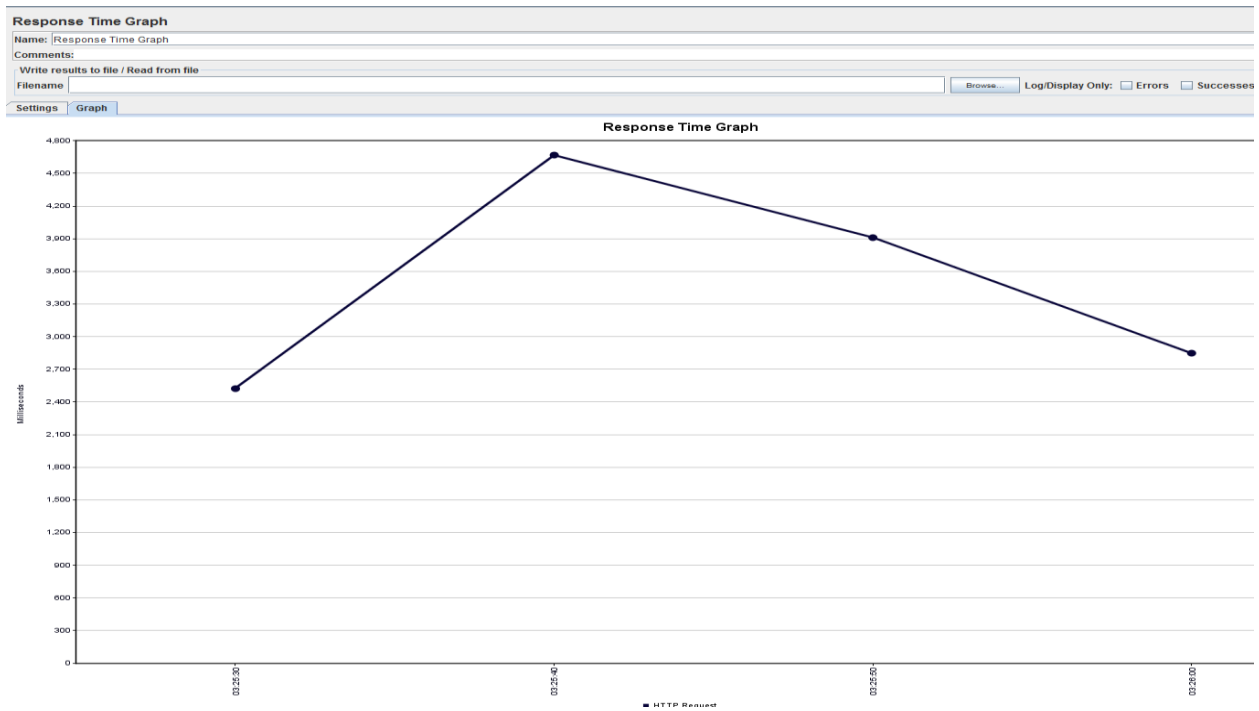


*Figure 7.2 Response Time Graph for Test Plan 1*





**Figure 7.3 Throughput graph for Test Plan 1**



**Figure 7.4 Response Time Graph for Test Plan 2**



Figure 7.5 Throughput Graph for Test Plan 2

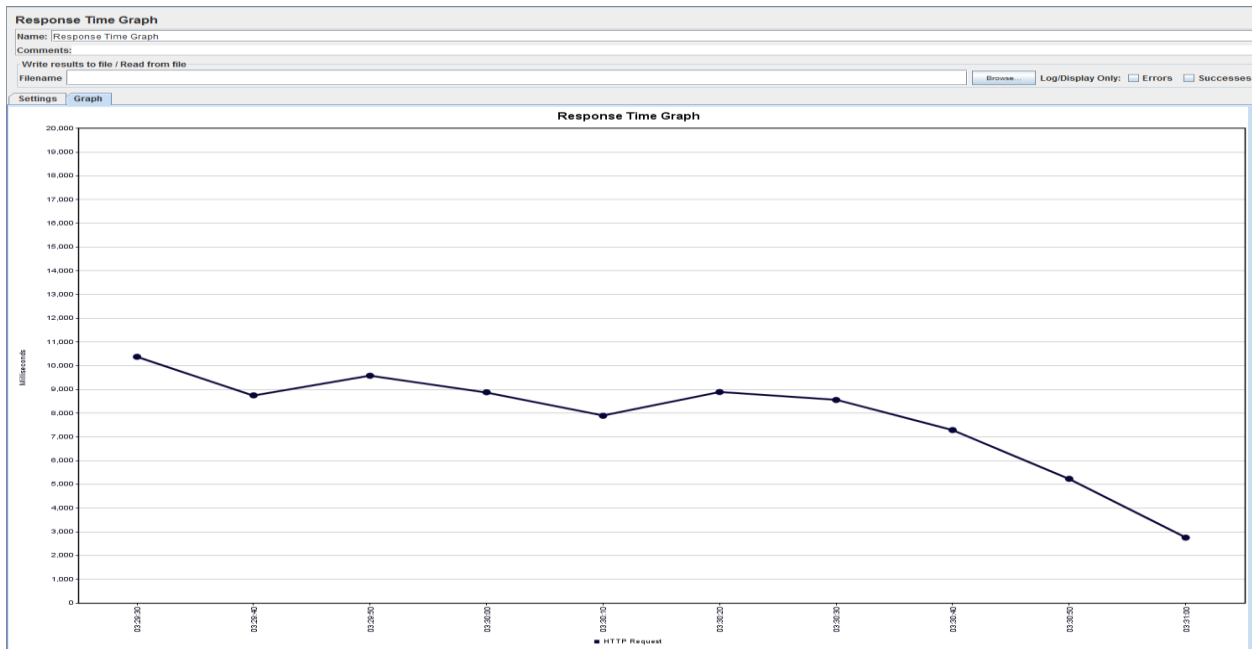


Figure 7.6 Response Time Graph for Test Plan 3



**Figure 7.7 Throughput Graph for Test Plan 3**

## Chapter 8 - Conclusion

The fleet management system could be used by anyone who owns a fleet of vehicles and would like to track each of the vehicles in real time. This way it is easy to keep track of the progress and have up to date information of each vehicle. This application also exposes an API that could be used by any developer to make use of the underlying features and develop his/her own application. For example, if a developer wants to develop an application, which tracks and shows each vehicle of a group of friends who are on a trip together going to the same destination, he can use the API and show all the vehicles of the group on a map. This project also avoids malicious updates by making use of a unique key for each of the user, which needs to be sent in the header. Thus, this project is a complete application, which could be used by a wide variety of users for various purposes. Developing this application has given me a good grip on web application development, as this project has end to end functionality and also has integration with an Android app. I have gained the required experience with the popular frameworks used in the industry for web application development such as Spring and Hibernate, thus making me confident to start my career as a Developer.

<b>Language</b>	<b>Lines of Code</b>
Java	1600
XML	250
JavaScript	300
<b>Total</b>	2150

*Figure 8 Table Lines of Code*

The above table depicts the breakdown of the lines of code in my project with respect to the language.

## **Chapter 9 - Future Work**

There is a lot of scope to extend this project, a dedicated desktop application could be built to track the vehicles and it can have filters to select different vehicles. The project could be hosted on an online server and made available to a wider user base. The UI could be improved to make the application look more elegant and rich. Add the functionality to store the trips made by each vehicle on a particular day. Perform analysis on the old data and generate reports, which could be useful for the business.

## References

[1] Representational state transfer (12/14)

<http://searchsoa.techtarget.com/definition/REST>

[2] Web services Glossary (02/02)

<https://www.w3.org/TR/ws-gloss/>

[3] Hibernate Architecture (2004)

<https://docs.jboss.org/hibernate/orm/3.5/reference/en-US/html/architecture.html>

[4] Spring Framework Architecture (2004)

[https://www.tutorialspoint.com/spring/spring\\_architecture.htm](https://www.tutorialspoint.com/spring/spring_architecture.htm)

[5] Spring Model View Controller

<http://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html>

[6] Android Platform Architecture (2010)

<https://developer.android.com/guide/platform/index.html>

[7] Advantages of using JSP

<http://www.xyzws.com/jspfaq/what-are-advantages-of-using-jsp/9>