SMART PARKING SYSTEM

By

SIRI CHANDANA YADAVALLI

B-Tech, Jawaharlal Nehru Technological University, India, 2014

A REPORT

Submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

Department of Computing and Information Sciences

College of Engineering

KANSAS STATE UNIVERSITY

Manhattan, Kansas

2016

Approved by:

Major Professor
Dr. Daniel Andresen

# ABSTRACT

Locating a parking spot during peak hours in most populated areas like shopping malls, universities, exhibitions or convention centers is difficult for the drivers. The difficulty rises from not knowing where the available spots may be at that required time. Smart parking is a solution to metropolitan cities to reduce congestion, cut vehicle emission totals and save persons' time by helping them in finding a spot to park.

Smart Parking is a parking system, usually a new one that is equipped with special structured devices (things) to detect the available parking slots at any parking area. This is an application based on Internet of Things (IoT) that in Real-Time environment have sensors and devices embedded into parking spaces, transmitting data on the occupancy status; and the vehicle drivers can search for parking availability using their mobile phones or any infotainment system that is attached to the vehicle. Hence the driver would know where there is an available spot to park his vehicle in less time, reducing the energy consumption and air pollution. The Client or the sensor posts the parking slot occupancy status to a web service URL. The Java based web service is built using Spring and Hibernate to connect to the backend system. The web service (.war) file is deployed on Apache Tomcat Server and the backend used is MySQL database.

# Table of Contents

# Table of Figures

## List of Tables

# Acknowledgements

I am deeply grateful to my major professor Dr. Daniel Andresen for his guidance, support, pointing me in the right direction. I appreciate the trust he placed in me. I would like to thank Dr. Mitchell L. Neilsen and Dr. Torben Amtoft for making their courses valuable and informative. It was a privilege to take courses under them, as they provided me with more practical guidance and experience. Thank you for serving on my committee. My sincere thanks to Joe Horan for supporting me through my study at K-State.

I would like to thank my parents and grandmother for their unconditional love and support. Special thanks to Ashish Pinninti, for the support he extended in difficult times and for being there for me when I needed him the most. And my heartfelt thanks to my younger sister and friends of mine for their best wishes.

# CHAPTER 1 – INTRODUCTION

## 1.1 Project Description

Seeking a vacant parking space during peak hours in areas like Hospitals, Hotels & Shopping Centers, Airports, Universities, and Exhibitions & Convention Center has always been frustrating for many drivers. Surveys says that traffic generated by cars searching for vacancies in Parking Spaces is up to 40% of the total traffic [1]. Now that is a serious issue to look after, and Smart Parking System is one of the best available solutions to at least reduce the traffic congestion caused due to the above problem. This application gives information about the occupancy status of the spaces in the parking lot equipped with sensors that detect the presence of vehicles.

Smart Parking is an Internet of Things (IoT) based application, used to detect the available parking slots. This app uses ultrasonic sensor to detect the presence of a vehicle (whether the parking slot is occupied or not). Based on the parking slot occupancy, the status (occupied/unoccupied) is displayed on the web application dashboard. In real time, the environment have sensors and devices embedded into parking spaces, transmitting data on the occupancy status; and the vehicle drivers can search for parking availability using their mobile phones or any infotainment system that is attached to the vehicle. Hence the driver would know where there is an available spot to park his vehicle in less time, reducing the energy consumption and air pollution.

The second part in this application is doing analysis on parking trends in a parking lot. The analysis gives information about which parking space is most occupied and least occupied and at what times of the day. This information is helpful in choosing one parking space when there are multiple available, keeping in mind the history of that space. For example, when there are more than one vacant slots the driver will want to choose the one that has less occupancy rate because the high occupancy rated slot might be wanted by many other drivers and you don't want to waste your time reaching that slot.

## 1.2 Motivation

A Smart Parking System like this helps drivers make smart decisions which will reduce congestion and make the most of available spaces. Finding a parking space has become a daily concern these days, and that is where the motivation for this project came up from. With the evolution of technology, we have smartphones, sensors that detect the presence of any object and my idea is having a system where parking spaces are equipped with these ultrasonic sensors that tells about the occupancy status of the parking spaces and a central management system that posts this occupancy status to a web application to guide the drivers in finding a vacant slot.

# CHAPTER 2 – REQUIREMENT ANALYSIS

## 2.1 Requirement Gathering

As soon as the project idea is confirmed, I have started working on the requirements for the implementation of the project. The idea is to develop a web service that can receive information about the occupancy status of the parking space from Client (here the database) and post that information to the web application. Also passing information to database through web service URL and updating the changes in the system. I did some research on current technologies that are used in industry and decided on understanding how Spring Framework works, how to connect to database with Hibernate instead of JDBC, also seen some best practices in writing JavaScript.

## 2.2 Requirement Specification

These are the technical requirements to develop Smart Parking System web application.

### 2.2.1 Software Requirement

Operating System: Windows XP, Windows 7 or Windows 8

IDE: Eclipse with STS Plugin

Application Server: Apache Tomcat 7.0.67

Front End: HTML5, JavaScript, jQuery, AJAX and CSS

Frameworks and APIs: Spring and Hibernate

Web Service: RESTful web services

Database: MySQL

Browser: Chrome or Firefox or Internet Explorer

### 2.2.2 Hardware Requirement

Processor: Intel Core 2 Duo or Higher

RAM: 2GB

Users who want to use the application can directly deploy the war file in Tomcat server and run the app using localhost with the port number that is configured during installation of the web server. The path to the web application is passed along with localhost to the browser.

# CHAPTER 3 – DEVELOPMENT BACKGROUND AND APPROACH

In order to understand our choice of frameworks we have to describe Smart Parking System's particular application requirements. This project has two primary components; a web service that receives the occupancy status from the client or the sensor and a web application that displays the parking slot occupancy status and also the parking history trends. The Java based web service is built with Spring and Hibernate to connect to the backend system which is MySQL database. The web service is deployed in Apache Tomcat Server. And the web application has front end designed using HTML5, CSS, JavaScript and jQuery. And has Ajax call to the web service URL.

## 3.1 Spring Framework [10]

The Spring Framework [2] is an application framework and inversion of control container for the Java platform. It has become popular as a replacement for, alternative to, or even addition to the EJB model. The Spring Framework provides about 20 modules which can be used based on an application requirement. The below figure shows typical Spring Framework which can be categorized into 4 groups: Core Container, Data Access/Integration, Web and Miscellaneous.
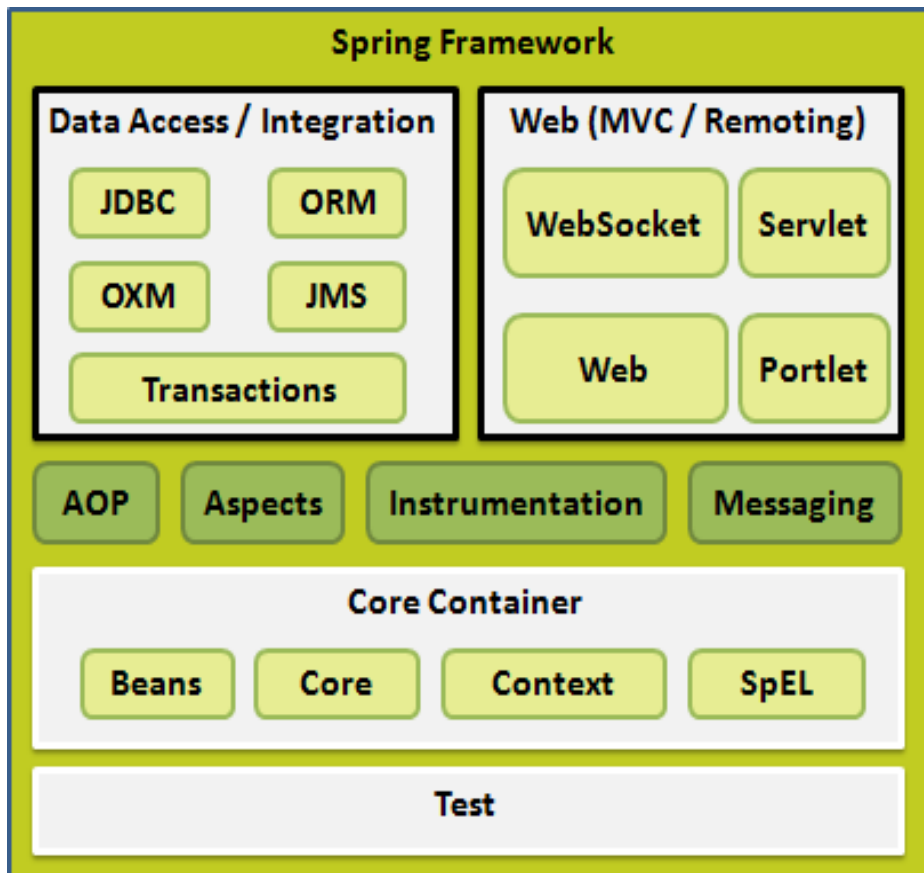


*Figure 3.1 - Spring Framework [3]*

In brief about each,

### 3.1.1 Core Container

The Core Container consists of Beans that provide BeanFactory, Core that provides features like Dependency Injection and Inversion of Control, Context that provides ApplicationContext which is the focal point of this module, SpEL provides a powerful expression language for manipulating and querying an object graph at runtime.

### 3.1.2 Data Access/Integration

This module consists of JDBC that provides an abstraction layer, ORM that provides integration layers for Java Persistence API, Hibernate etc., OXM provides an abstraction layer that support Object/XML mapping implementations, Java Messaging Service which contains features for producing and consuming messages, and Transactions to support programmatic and declarative transaction management for classes that implement POJOs.

### 3.1.3 Web

This module consists of Web-Socket that provide support for two way communication between client and server, Web-MVC that contains MVC implementation, Web that provides basic web-oriented integration features and Web-Portlet that provides MVC implementation for Portlet environment.

### 3.1.4 Miscellaneous

Some other components on Spring Architecture are AOP that provides aspect-oriented programming implementation, Aspects that provide integration with AspectJ which is an AOP framework, Instrumentation that provides class instrumentation support and class loader implementations, Messaging that provides support to STOMP, Test to support the testing of Spring Framework components with JUnit or TestNG frameworks.

## 3.2 Hibernate ORM

Hibernate ORM (Object Relational Model) is a high performance object-relational persistence and query service. This eliminates the time spent on writing the native SQL queries and lets users to develop persistent classes using object-oriented principles. Using the metadata describing the mapping between the objects and the database, ORM persists the objects in a Java application into the tables of a relational database.
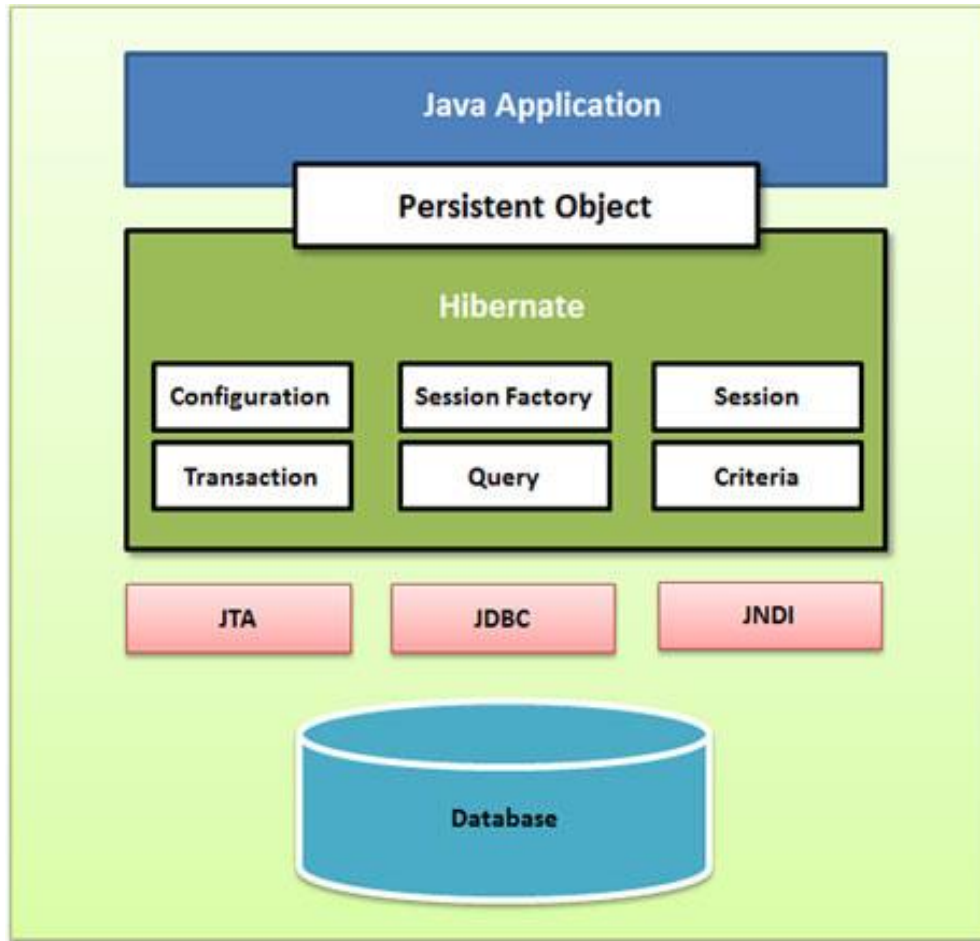
*Figure 3.2 - Hibernate Architecture [4]*

The Hibernate architecture is as shown above in the figure 2.2. Let us discuss a little bit of it here.

Hibernate [5] uses Java APIs like

JTA (Java Transaction API) – To integrate with J2EE application servers

JDBC (Java Database Connectivity) – Provides abstraction to allow any database with JDBC drivers to be supported by Hibernate.

JNDI (Java Naming and Directory Interface) - To integrate with J2EE application servers

Brief about core interfaces of Hibernate Framework:

- Configuration: used to configure and bootstrap hibernate. The instance of this interface is used by the application to specify the location of hibernate specific mapping documents.
- Session Factory: This delivers the session objects to hibernate application. There will be a single Session Factory for the whole application and it will be shared among all the threads.
- Session: The instances of this interface are lightweight, they are not thread safe. Session interface is the primary interface used by hibernate applications.

- Transaction: The above three interfaces are mandatory, whereas this is an optional interface that abstracts the code from any kind of transaction implementations.
- Query and Criteria: This allows user to perform queries and also control the flow of the query execution.

### 3.3 JavaScript

The front end of the application is developed using JavaScript and jQuery [7] library of JavaScript [6]. It is the programming language of HTML and the Web. Smart Parking System application uses:

HTML to define the content of web page

CSS to specify the layout of web page

JavaScript to program the behavior of the web page

And AJAX (Asynchronous JavaScript and XML) calls are made to the web service every 2 sec to display the latest occupancy statuses retrieving data from the database.

### 3.4 Tomcat 7.0.67 Web Server

Tomcat [8] is an open-source web server developed by the Apache Software Foundation. Catalina is Tomcat's servlet container, implements Sun Microsystems' specifications for Servlet and Java Server pages. The web application that is developed in Eclipse IDE is exported to a war file which is then deployed on Apache Tomcat Server. Using any web browser, we can run the web application that is deployed on Tomcat Server.

# CHAPTER 4 – SYSTEM DESIGN

## 4.1 System Design

UML diagrams are used to explain the design of the system. Once the requirement gathering is completed, system design is done using Unified Modeling Language (UML). UML plays an important role in designing object oriented software by using graphical notations to depict the design of the system.

### 4.1.1 Use Case Diagram

In its simplest form, a use case can be described as a specific way of using the system from a user's (actor's) perspective. A more detailed description might characterize a use case as:

- A pattern of behavior the system exhibits
- A sequence of related transactions performed by an actor and the system
- Delivering something of value to the actor

**Use Cases provide a means to:**

- Capture system requirements
- Communicate with the end users and domain experts
- Test the system

The User of the system is a vehicle driver who would be searching for a vacant parking space, and the use cases are the sequence of actions that provide something of measurable value to the user like checking the parking lot for vacant spaces, checking the parking history to find the most occupied and less occupied parking slots. And finally after finding a vacant space, he can park his vehicle.

The below figure depicts the Use Case Diagram for the Vehicle Drivers:
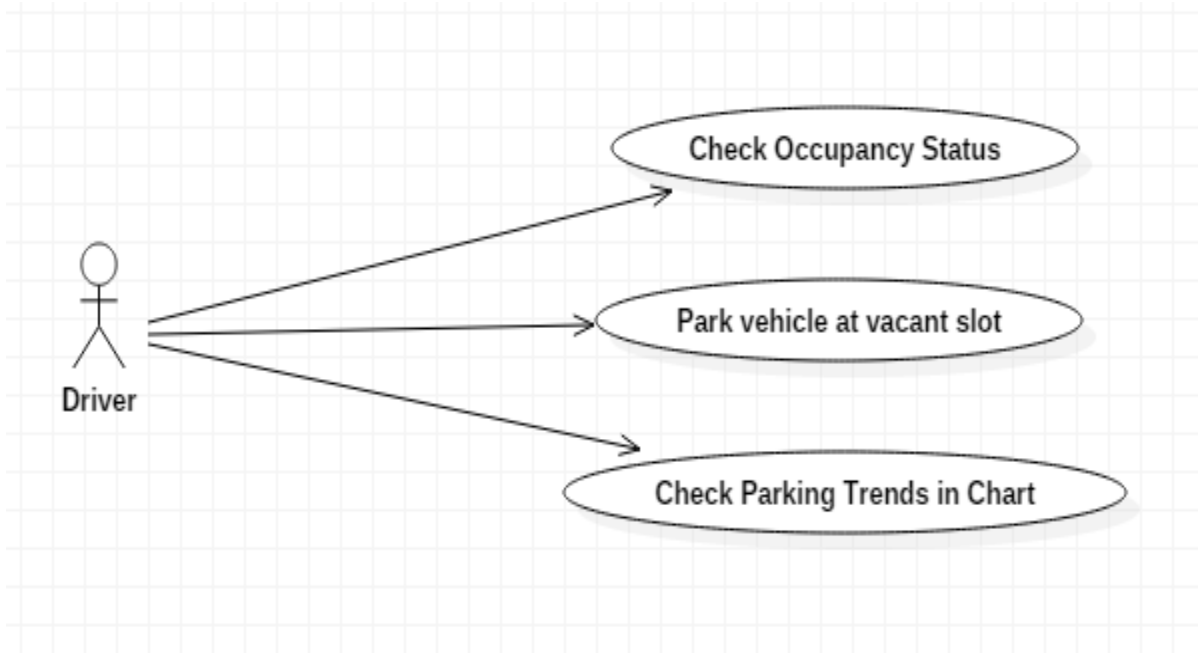


*Figure 4.1 - Use Case Diagram for Vehicle Driver*

### 4.1.2 Class Diagram

A class diagram is a picture for describing generic descriptions of possible systems. Class diagrams and collaboration diagrams are alternate representations of object models. This contain icons representing classes, interfaces, and their relationships. We can also create one or more class diagrams to depict classes contained by each package in our model; such class diagrams are themselves contained by the package enclosing the classes they depict; the icons representing logical packages and classes in class diagrams.

Below is the main class diagram for the entire application. The diagram clearly explains about the relationships between two or more classes and also between classes and interfaces.
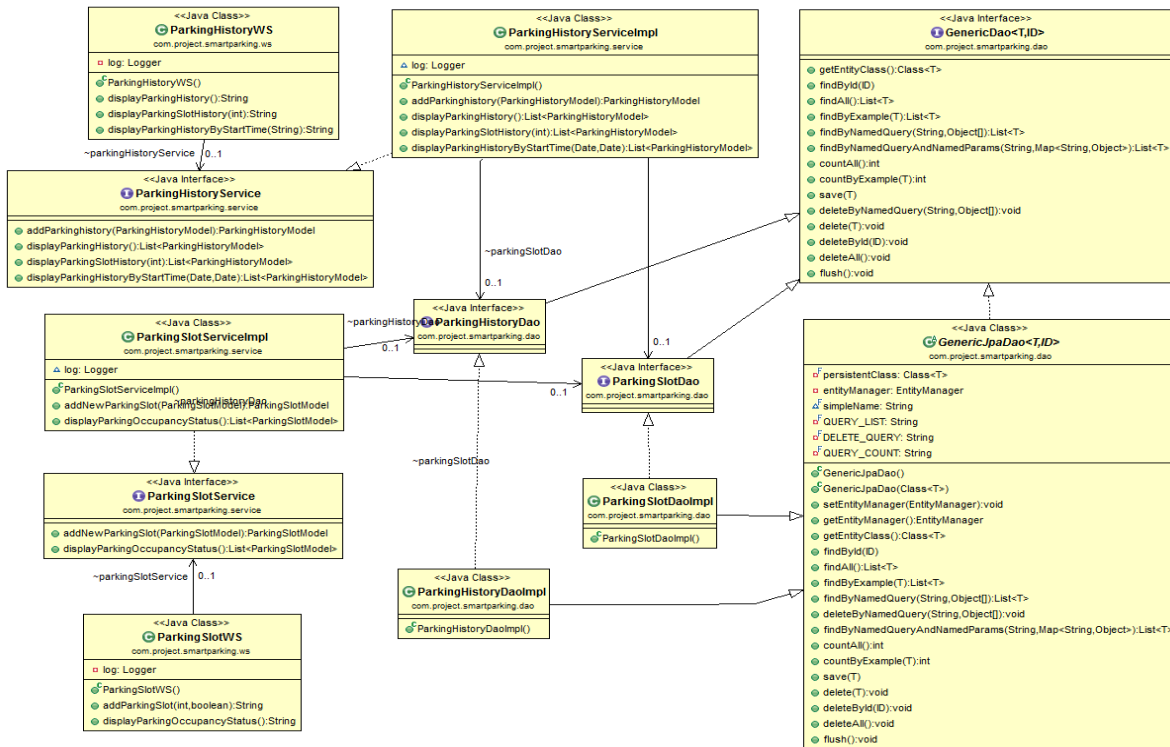


*Figure 4.2 - Class Diagram for Smart Parking System*

Here are the other classes and their relationships in the project:

- DAO Classes, their attributes clearly mentioned; top section is the class or the interface name, below are the attributes in those particular classes. Here we have the queries required to retrieve or update the database. GenericDao is an interface providing basic CRUD operations. And GenericJpaDao is the JPA implementation of the GenericDao interface. ParkingSlotDao is an interface extending GenericDao interface and ParkingSlotDaoImpl is the implementation class for the interface. Similar with ParkingHistoryDao interface and ParkingHistoryDaoImpl class.
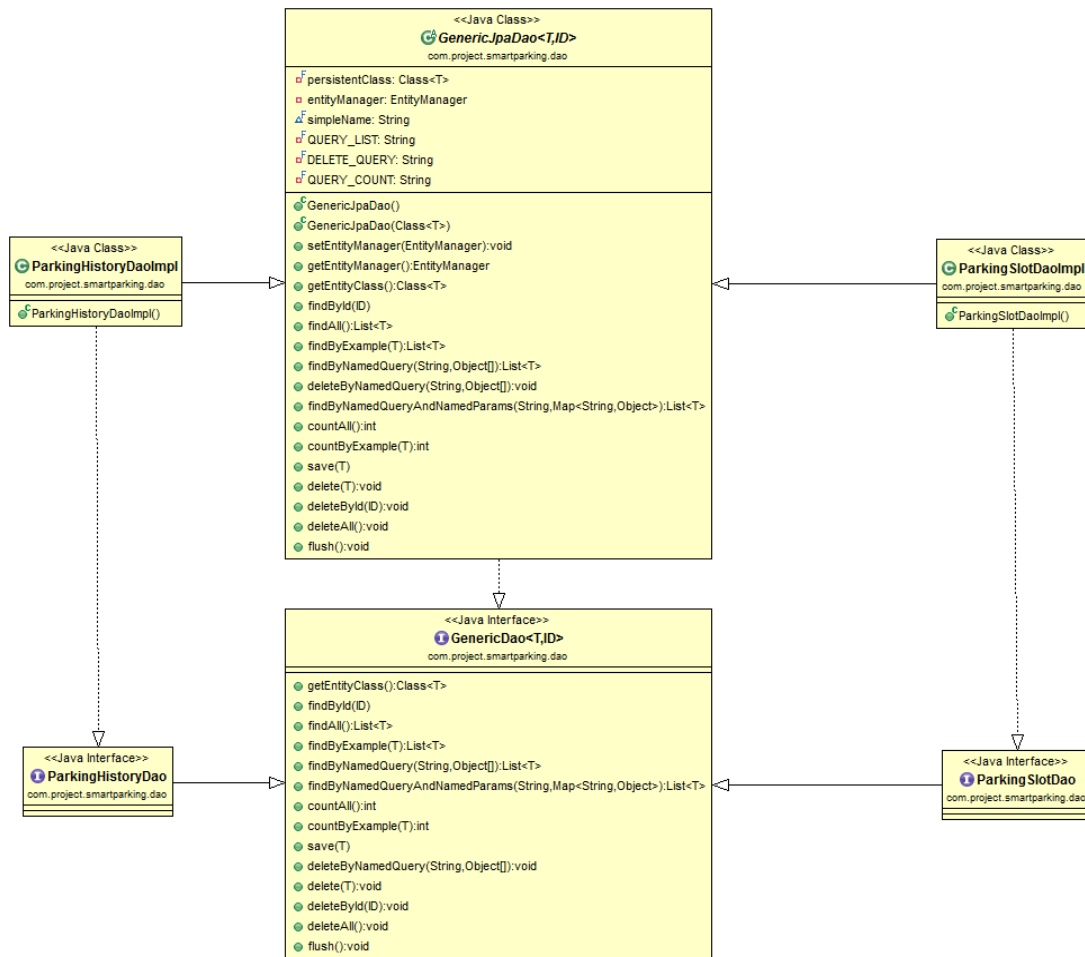


*Figure 4.3 - Class Diagram for DAO Classes*

- Entity Classes, and their attributes; these are the classes that does Object Relational Mapping to the database. These classes are the mapping of the database tables using ORM. ParkingHistory and ParkingSlot classes are mapped to Parking History and Parking Slot tables in the databases. And the column values of the tables are mapped with the attributes of the classes. We also define the relationships between the tables in this classes.
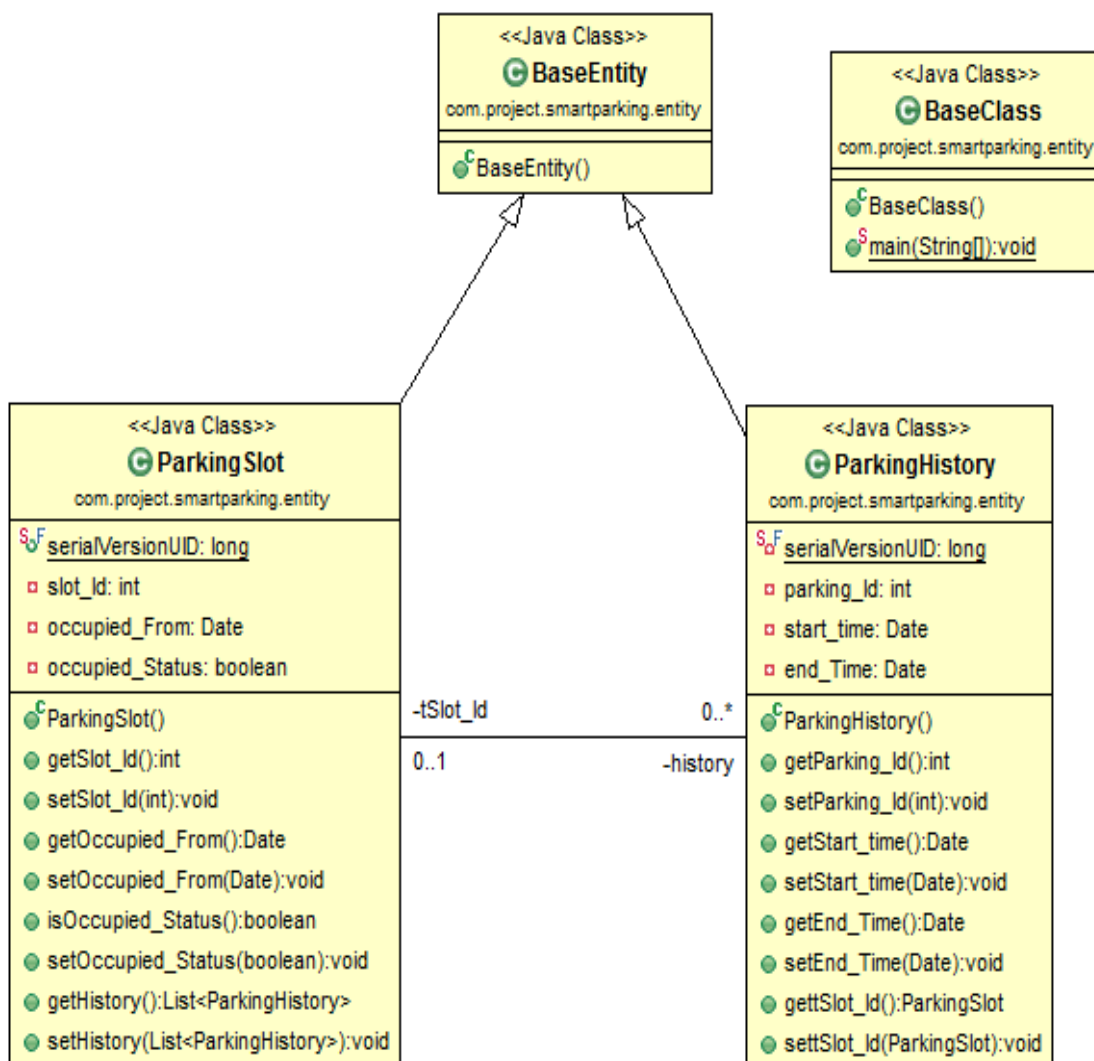


*Figure 4.4 - Class Diagram for Entity Classes*

11

- Model Classes, and their attributes; Relationship between Parking Slot table and Parking History table. These classes have methods that convert the Entity to Models with giving only the required information.
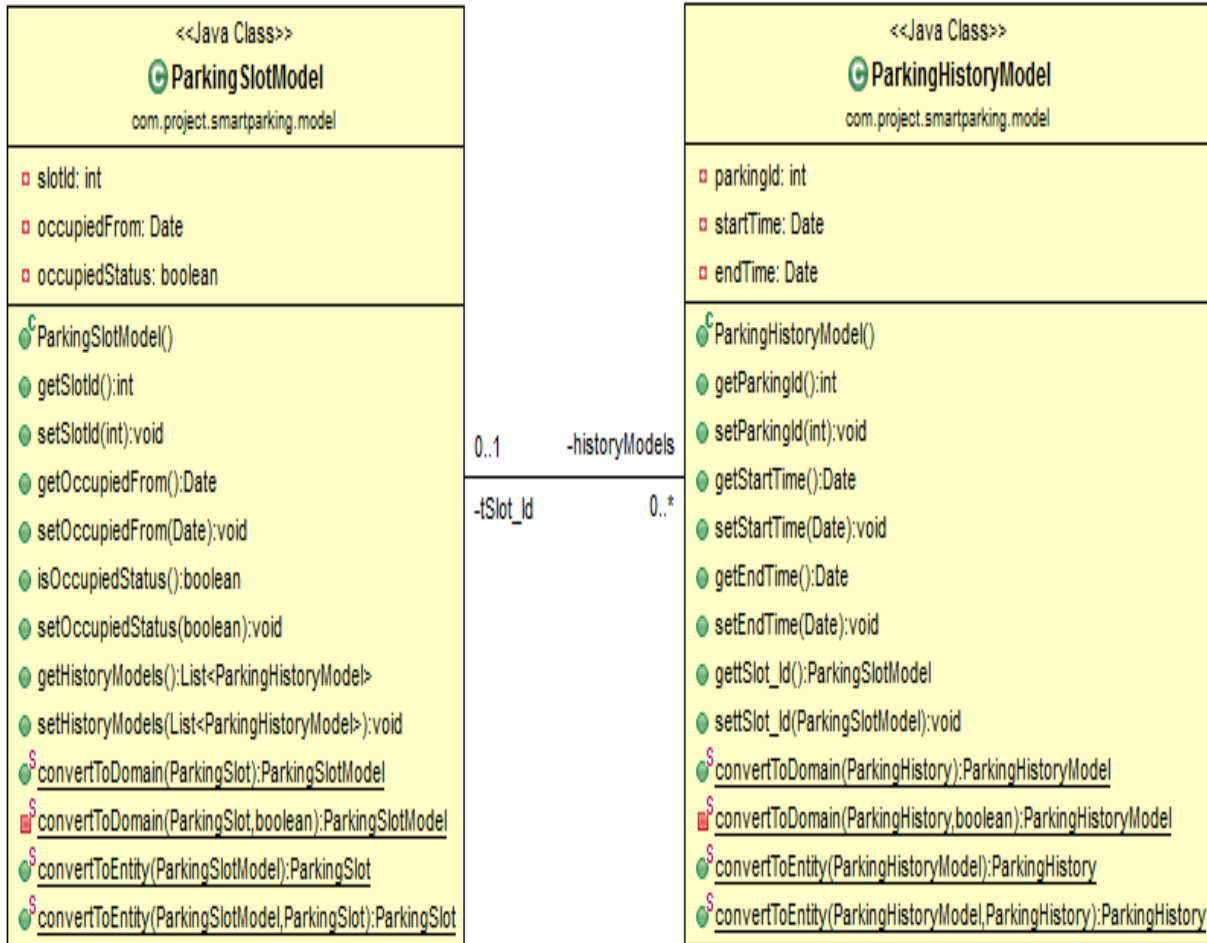


| <<Java Class>> ParkingSlotModel | <<Java Class>> ParkingHistoryModel |
|---|---|
| com.project.smartparking.model | com.project.smartparking.model |
| □ slotId: int | □ parkingId: int |
| □ occupiedFrom: Date | □ startTime: Date |
| □ occupiedStatus: boolean | □ endTime: Date |
| ParkingSlotModel() | ParkingHistoryModel() |
| getSlotId():int | getParkingId():int |
| setSlotId(int):void | setParkingId(int):void |
| getOccupiedFrom():Date | getStartTime():Date |
| setOccupiedFrom(Date):void | setStartTime(Date):void |
| isOccupiedStatus():boolean | getEndTime():Date |
| setOccupiedStatus(boolean):void | setEndTime(Date):void |
| getHistoryModels():List<ParkingHistoryModel> | gettSlot_Id():ParkingSlotModel |
| setHistoryModels(List<ParkingHistoryModel>):void | settSlot_Id(ParkingSlotModel):void |
| convertToDomain(ParkingSlot):ParkingSlotModel | convertToDomain(ParkingHistory):ParkingHistoryModel |
| convertToDomain(ParkingSlot,boolean):ParkingSlotModel | convertToDomain(ParkingHistory,boolean):ParkingHistoryModel |
| convertToEntity(ParkingSlotModel):ParkingSlot | convertToEntity(ParkingHistoryModel):ParkingHistory |
| convertToEntity(ParkingSlotModel,ParkingSlot):ParkingSlot | convertToEntity(ParkingHistoryModel,ParkingHistory):ParkingHistory |

Relationship labels: 0..1   -historyModels   -tSlot_Id   0..*

*Figure 4.5 - Class Diagram for Model Classes*

- Service Classes, and their attributes; the main class diagram above depicts the relationships between these service classes and web services classes. These classes contains methods to retrieve the requests from the View and handle those requests by having methods to display current parking occupancy status of all slot ids and also parking history to generate chart. The response is sent back to the View (designed using JavaScript).
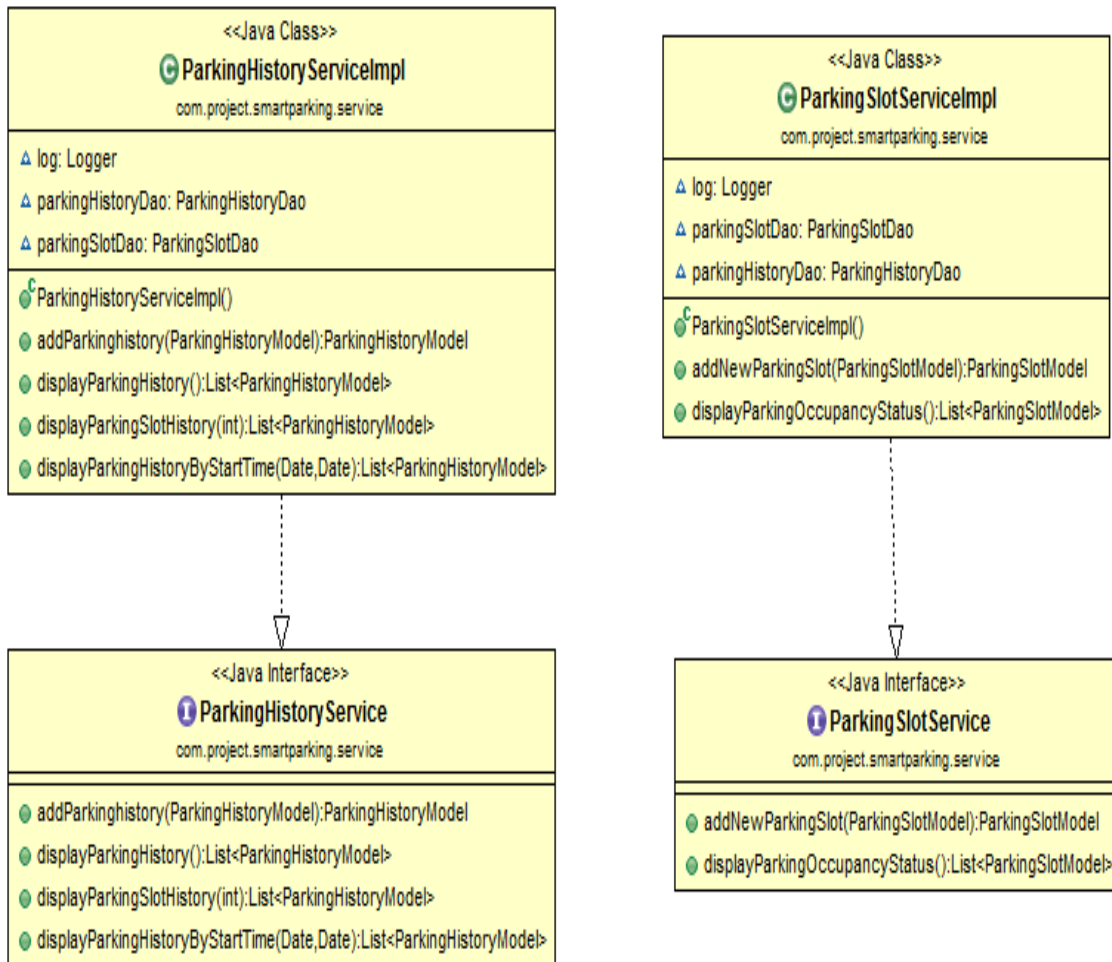


*Figure 4.6 - Class Diagram for Service Classes*

# CHAPTER 5 – IMPLEMENTATION

Smart Parking System is a web application that is developed to help drivers in finding a vacant parking space during any time of day. The main objective of the application is to provide the occupancy status (vacant/occupied) of the parking spaces and also to get information about the parking history data on any particular date. The user is allowed to choose any date and can get the information related to parking spaces occupancy throughout that day.

The main features of the application are:

Parking Slot Occupancy: This is the first part of the application where there is a dashboard that has occupancy statuses of the parking slots. The dashboard is refreshed every 2 sec and the latest information is displayed for the user.

Parking History Analysis: This is an extension to the application where User is able to see the occupancy rate of each parking lot on any particular day in the form of charts. Using this data, driver can choose the less occupied slot when there are more than one slot available. Also the parking lot owners can rent the spaces by charging based on their occupancy rates at any particular time of the day.

The Smart Parking Application is developed on Eclipse IDE with STS Plugin using Java 1.7. The user interface is designed using JavaScript and the business logic I coded in Java. The application also makes use of annotations in Java, especially Spring Framework and Hibernate annotations. The backend database is MySQL and I have used MySQL Query Browser in order to manage the database. The web application developed is archived to a war file and then deployed in Tomcat Server. The total number of lines of code breakdown with respect to language is listed below:

| Language | Number of LOC |
|---|---|
| Java | 1677 |
| XML | 132 |
| HTML5, CSS | 169 |
| JavaScript, jQuery | 164 |
| Total | 2142 |

*Table 5.1 Lines of Code*

## 5.1 Output Screens

The Graphical User Interface for this application is designed by keeping in mind that the application's user interface is everything that the user can see and interact with. Technologies used to develop this user interface are mainly JavaScript, jQuery, HTML5 and CSS. To interact with backend web service written in Java, jQuery methods get() and post() are mainly used. Ajax calls are made every 2 seconds by passing URL to be hit, and the required data from the database is

retrieved and passed back to the caller. Accordingly the Occupancy Status Message and Occupancy Indicator in the front end are updated every 2 seconds. Similar thing is with the chart showing the parking history. JavaScript FusionCharts are used to show the occupancy of parking spaces starting from zeroth hour to 24$^{th}$ hour. Here we can see the parking history of any day, by just setting the required date by using the calendar provided.

### 5.1.1 Smart Parking System

The figure 5.1 is seen when the Tomcat Server is started and then we run the application by calling the localhost in any browser. This screen contains a table that shows the occupancy status of the parking slots and a chart that depicts the parking history.

In the table, the first column is Parking Slot Number or Id, second column is the Occupancy Indicator, and third column is the Occupancy Message. Occupancy Indicator have two buttons or LEDs red and green. Green indicates that the parking slot is vacant and red indicates that it is already occupied by other vehicles. The last column is to display the Occupancy Message, i.e. 'Vacant' or 'Occupied'.

In the chart below the table, we can see that there are Slot no's and Slot Occupancy Status at particular 4 hour slot of the day. Upon hovering on the small blue bars, we can see the exact time when there was a vehicle in that slot.

The User Interface of Smart Parking web application meets all the initial goals, being very precise and making every detail clear on screen. The UI is designed in a way where the occupancy statuses of parking slots are very clearly visible by just looking into the colors of the indicators. And further information can be seen in the Occupancy message column. And this UI can be extended to mobile platforms with the same design, as it is very much compatible with the mobile screen size as well. One can view the occupancy status in one panel and can see the parking history by scrolling down in the mobile screen.
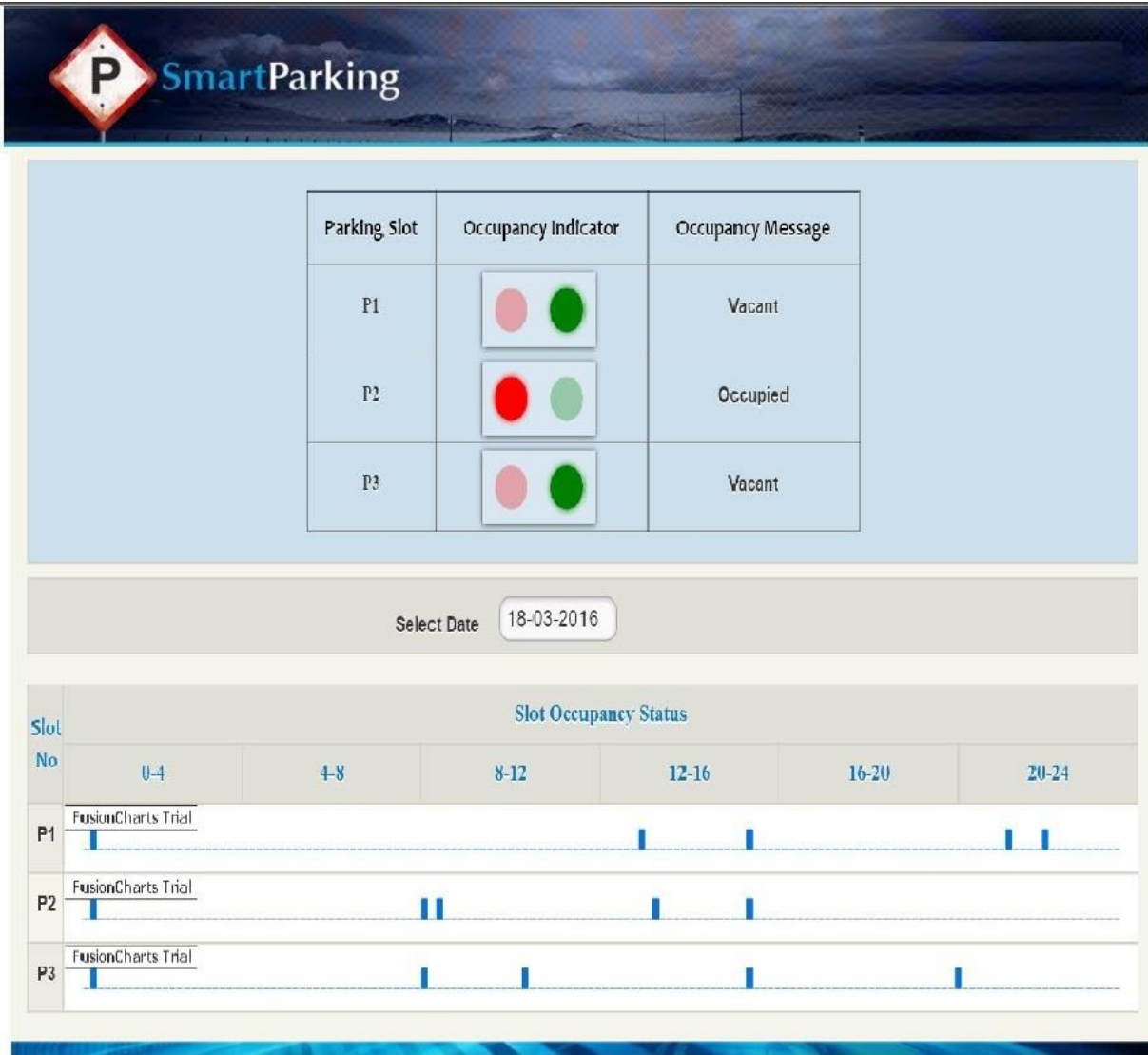
*Figure 5.1 - Smart Parking System*

The figure 5.2 shows to set any date required to see the parking slot occupancy statuses on that day. This is an easy way to select a date by either writing the numbers manually or by using the + and – signs. When a date is selected, just click on Set to apply the changes. And we would be given the information of parking spaces on that date.



*Figure 5.2 - Smart Parking System - Set Date*

## 5.1.2 Tomcat Server

The screenshot below (figure 5.3) shows the web service deployed on Apache Tomcat server.
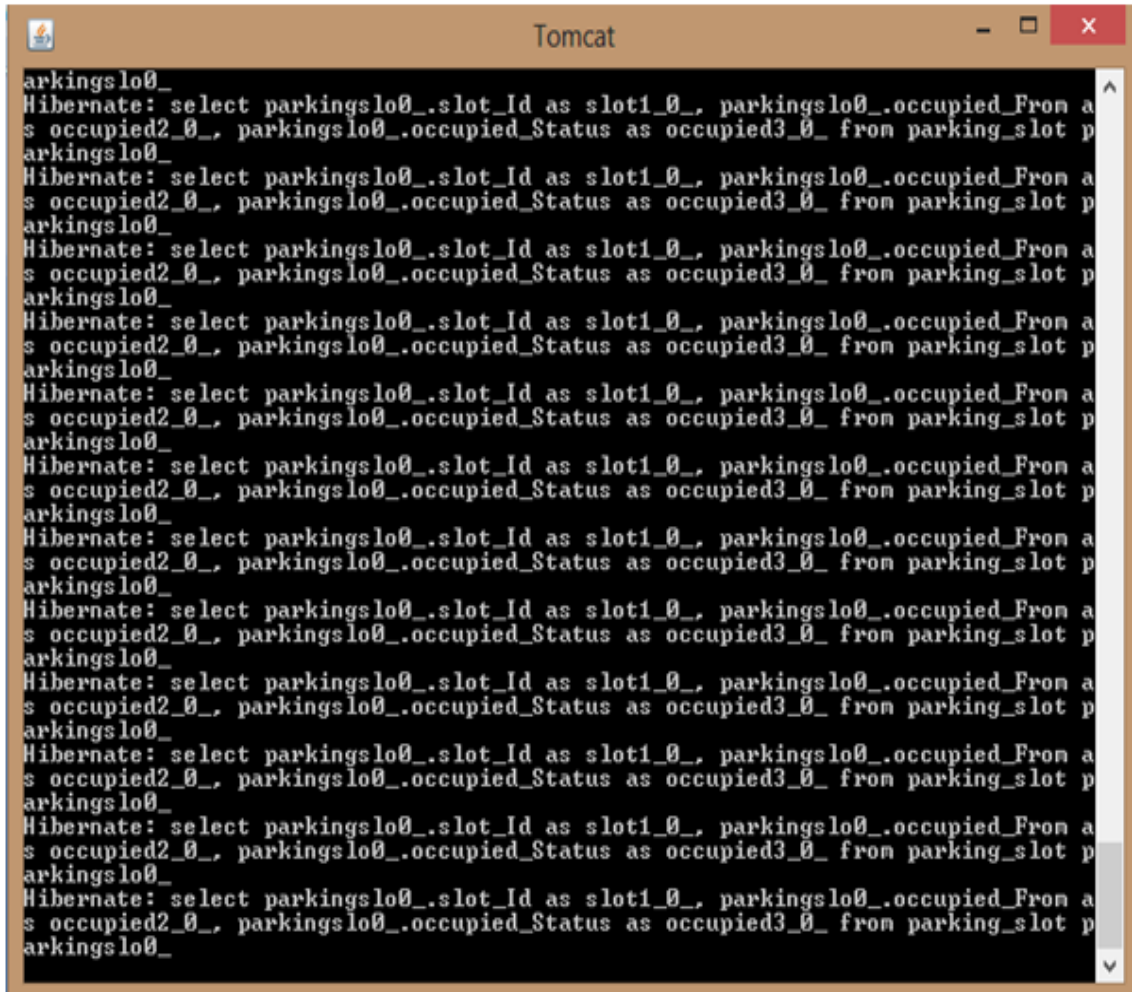


*Figure 5.3 - Web Service on Tomcat Server*

Now when the sensor detects the presence of a vehicle, it posts data to MySQL via web service URL:

http://localhost:8080/SmartParkingWS/smartParking/ParkingSlotWS/addParkingSlot/1/true

This web service call can be executed using any client like Android phone, web browser, or any IoT device.

The output screenshot is shown in figure 5.4, where we can see that the occupancy status of the parking slot P1 is changed from Vacant to Occupied.
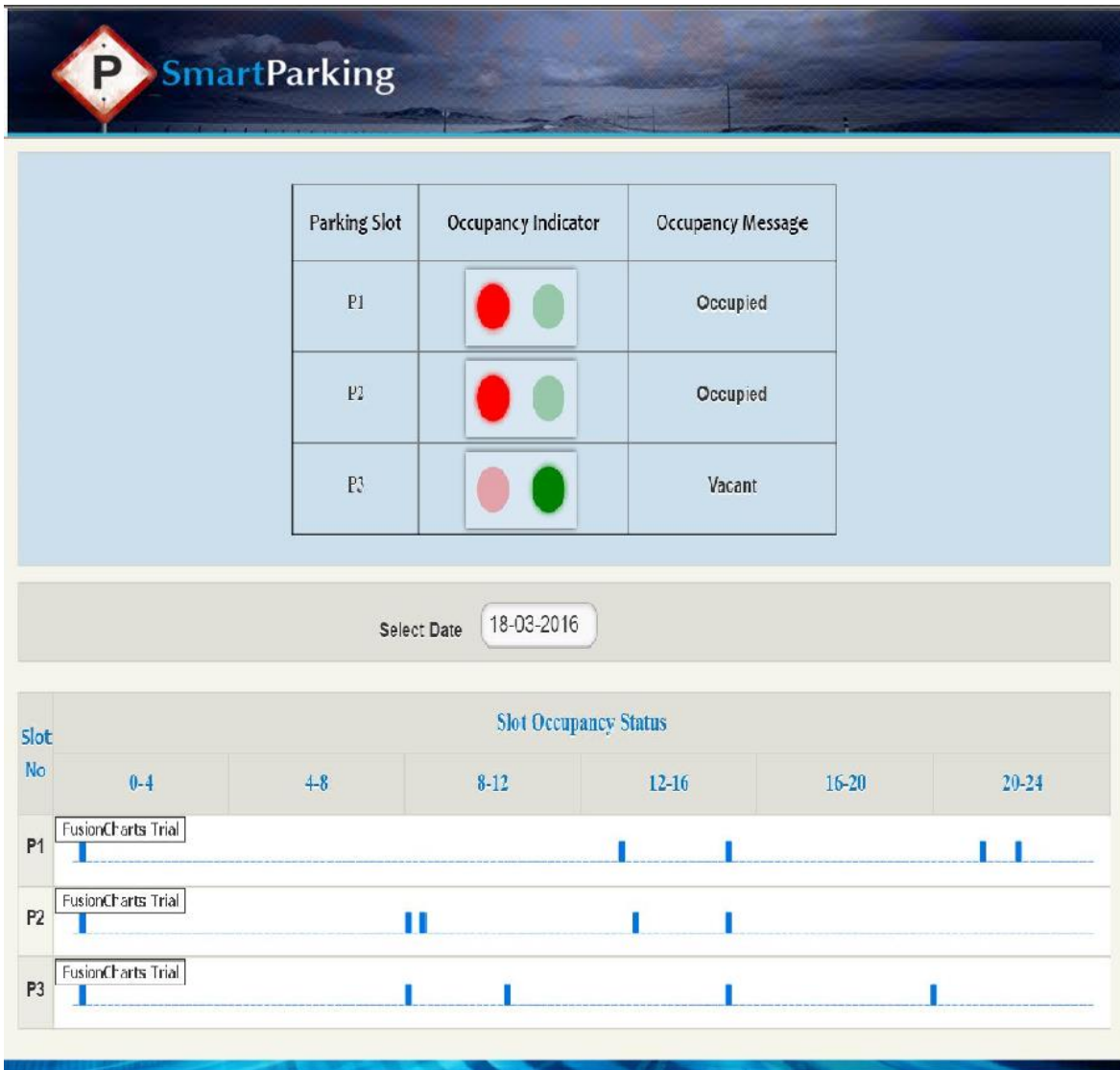


*Figure 5.4 – SPS: Change in Occupancy Status*

All the above screenshots shows a good view of the user interface of Smart Parking System. The goal of the UI is to clearly show the status of the parking slots. The Green and Red indicators gives the status in just first look. And the Occupancy Message is to precisely say the availability of the parking slot.

# CHAPTER 6 – TESTING

A process of executing a program with the explicit intention of finding errors, that is making the program fail. It is the process of detecting errors and performs a very critical - role for quality assurance, also for ensuring the reliability of software. The results of testing are used later on during maintenance also.

## 7.1 Unit Testing

It concentrates on each unit of the software as implemented in source code and is a white box oriented. Using the component level design description as a guide, important control paths are tested to uncover errors within the boundary of the module. In the unit testing, the steps can be conducted in parallel for multiple components in my project I tested all the modules individually related to main function codes and attacks also.

Below are the unit test cases that are run manually:

| Test Case | Expected Result | Result |
|---|---|---|
| After the start of Tomcat Server, on the load of html file | Display correct occupancy status messages for all the parking slots | Pass |
| After the start of Tomcat Server, on the load of html file | Display correct occupancy status indicators for all the parking slots | Pass |
| After the start of Tomcat Server, on the load of html file | Display Red indicator for 'Occupied' Status Message | Pass |
| After the start of Tomcat Server, on the load of html file | Display Green indicator for 'Vacant' Status Message | Pass |
| Upon receiving occupancy status change from client for parking slot 1 | Change in the parking slot 1 only, and should not affect other parking slot status messages. | Pass |
| Upon receiving occupancy status change from client for parking slot 1 | Change in the parking slot 1 only, and should not affect other parking slot status indicators. | Pass |
| Upon receiving occupancy status change from client for parking slot 2 | Change in the parking slot 2 only, and should not affect other parking slot status messages. | Pass |
| Upon receiving occupancy status change from client for parking slot 2 | Change in the parking slot 2 only, and should not affect other parking slot status indicators. | Pass |
| Upon receiving occupancy status change from client for parking slot 3 | Change in the parking slot 3 only, and should not affect other parking slot status messages. | Pass |

| Upon receiving occupancy status change from client for parking slot 3 | Change in the parking slot 3 only, and should not affect other parking slot status indicators. | Pass |
|---|---|---|
| Upon receiving Occupancy Status as 'Occupied' | Change Green indicator to Red indicator | Pass |
| Upon receiving Occupancy Status as 'Occupied' | Change Occupancy Message from 'Vacant' to 'Occupied' | Pass |
| Upon receiving Occupancy Status as 'Vacant' | Change Red indicator to Green indicator | Pass |
| Upon receiving Occupancy Status as 'Vacant' | Change Occupancy Message from 'Occupied' to 'Vacant' | Pass |
| Upon loading html page | Display occupancy trends of all parking slots in the chart at all the recorded timings | Pass |
| Upon a vehicle occupying a vacant parking slot | Display the presence of vehicle at that particular time in the chart | Pass |

*Table 7.1 Unit Test Cases*

## 7.2 Integration Testing

Here focus is on design and construction of the software architecture. Integration Testing is a systematic technique for constructing the program structure while at the same time conducting tests to uncover errors associated with interfacing. The objective is to take unit tested components and build a program structure that has been dictated by design. The goal here is to see if modules can be integrated properly, the emphasis being on testing interfaces between modules.

This testing activity can be considered as testing the design and hence the emphasis on testing module interactions. In this project the main system is formed by integrating all the modules. When integrating all the modules I have checked whether the integration effects working of any of the services by giving different combinations of inputs with which the two services run perfectly before integration.

Below are the few Integration Test Cases that are done manually:

| Test Case | Expected Result | Result |
|---|---|---|
| After the start of Tomcat Server, on the load of html file | Display correct occupancy status messages for all the parking slots and also display occupancy trends of all parking | Pass |

| | slots in the chart at all the recorded timings | |
|---|---|---|
| Upon receiving Occupancy Status as 'Occupied' | Change Green indicator to Red indicator and change in Occupancy Message from 'Vacant' to 'Occupied' | Pass |
| Upon receiving Occupancy Status as 'Vacant' | Change Red indicator to Green indicator and change in Occupancy Message from 'Occupied' to 'Vacant' | Pass |
| Upon a vehicle occupying a vacant parking slot | Display the presence of vehicle at that particular time in the chart and also change in the occupancy status and message | Pass |
| Upon refreshing the html page | Sync data with the database and update the page | Pass |

*Table 7.2 Integration Test Cases*

## 7.3 Validation Testing

In this, requirements established as part of software requirement analysis are validated against the software that has been constructed i.e., validation succeeds when software functions in a manner that can reasonably expected by the customer.

I made sure I have covered all the requirements that were discussed earlier at the start of the project. And also confirmed that the application works just the way it has to.

## 7.4 User Testing

User testing is a technique used to evaluate a product by testing it on different users. This can be seen as inimitable practice, as it gives direct input on how real users use the system. I have asked couple of my friends to try installing and running this application on their machines and got expected results. I have also presented this application in front of my Major Professor where I only completed the first phase of the application that the application would display the occupancy status of the parking space. But later we discussed on extending the application to display parking trends on a chart for any given date. I successfully completed implementing that part as well and tested to result as expected. The application was tested in all possible aspects and got just expected results.

## 7.5 Performance Testing

Performance Testing is performed to determine how fast a system performs under a particular load. It is also used to validate and verify other attributes of the system such as scalability, reliability and resource usage. Load testing is primarily concerned with testing that can continue to operate under specific load be it large amount of data or be it numerous users.

I have used Apache JMeter [9] to perform load testing which can be used to test performance both on static as well as dynamic resources (files, Servlets, Perl Scripts, Java Objects, Databases and Queries, FTP Servers and more). It can be used to simulate a heavy load on a server, network or object to test its strength or to analyze overall performance under different load types. Using JMeter I have taken increased number of users while using the page with the same Ramp-Up period and Loop Count.

Table below are the results of different test cases:

| No. Of Users | Ramp Up Period | Loop Count | Throughput |
|---|---|---|---|
| 50 | 5 | 100 | 11,645.9/min |
| 100 | 5 | 100 | 14,889.4/min |
| 200 | 5 | 100 | 27,434.8/min |

*Table 7.3 Performance Testing Analysis*

Using JMeter I have taken increased number of users while using the page with the same Ramp-Up period and Loop Count. We can see that with increase in the number of users, the throughput is increased.

**Graphical Analysis for the same test cases:**

The graphs below shows Response Times over Time. The vertical axis is the Response Time in milliseconds and the horizontal axis is Elapsed Time i.e. granularity/500 milliseconds. In all the three test cases, we can see that the Response Times over Time is a constant graph.
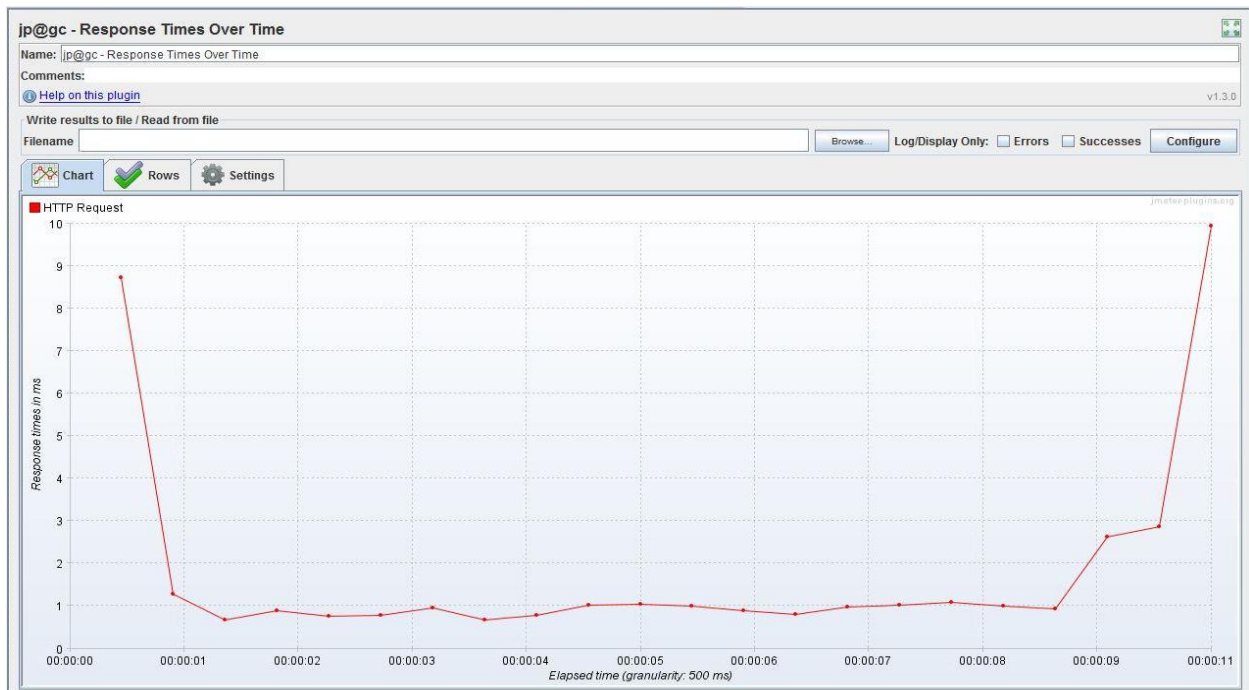
Test Case 1: No. Of Users = 50



*Figure 7.1 - Response Time Analysis for 50 Users*
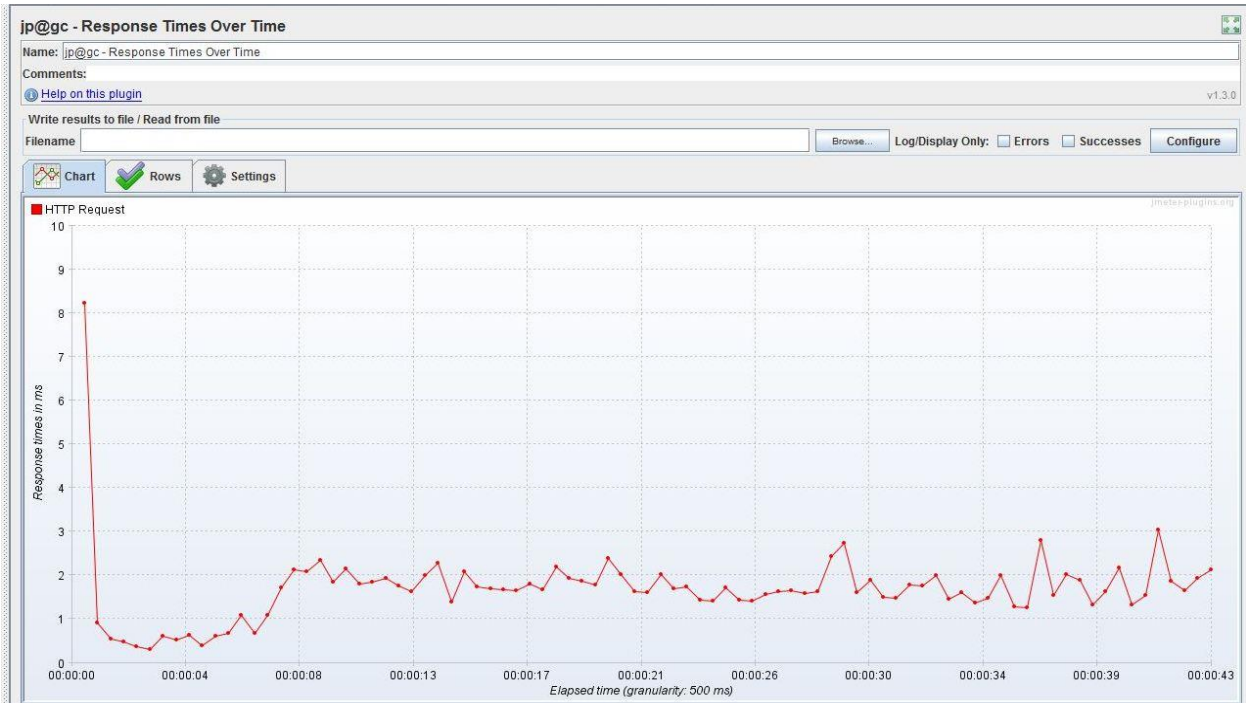
Test Case 2: No. Of Users = 100



*Figure 7.2 - Response Time Analysis for 100 Users*

Test Case 3: No. Of Users = 200



*Figure 7.3 - Response Time Analysis for 100 Users*

24

# CHAPTER 7 – CONCLUSION AND FUTURE WORK

## 7.1 Conclusion

Smart Parking System is a solution to the existing traffic congestion, to reduce drivers' frustration by providing information about the occupancy status of the parking spaces. The project development went smoothly while teaching me many best practices in programming using the current trending technologies like Spring Framework, Hibernate ORM and REST APIs. I could see that all the initial requirements of the project are achieved and also I tried doing minor data analysis on the parking spaces occupancy statuses.

The web application is user friendly; any user can easily find the status (vacant/occupied) of the parking space and also can set the required date to see the parking history. Starting from coming up with project idea, understanding the requirements and choosing the best technologies for the implementation, all this gave me very good experience and exposure in development of a full stack web application. Upon completing this project successfully, I got familiar with Eclipse IDE Shortcuts, Database tools and Tomcat server usage.

## 7.2 Future Work

This application is an initial step in reaching the effective solution for the daily concern. This project can be extended in multiple ways:

- To provide a central management system that make sure only authenticated information is sent to the Client, i.e. dealing with the security issues.
- More analysis can be done using the parking history data by which User can get recommendations or suggestions on parking spaces and their availability trends.
- And this analysis can be used while reserving a parking space by User or while renting a space, to decide the price of the parking space.
- We could also do a mobile application through which driver can get the occupancy statuses of the parking spaces.

# CHAPTER 8 – BIBLIOGRAPHY

[1] A Reservation – based Smart Parking System by Hongwei Wang and Wenbo He – April 01, 2016

http://cse.unl.edu/~byrav/INFOCOM2011/workshops/papers/p701-wang.pdf

[2] Spring Architecture – April 01, 2016

https://sites.google.com/site/sureshdevang/why-spring-framework

[3] Spring Framework – Architecture – April 01, 2016

http://www.tutorialspoint.com/spring/spring_architecture.htm

[4] Hibernate Architecture – April 02, 2016

http://www.tutorialspoint.com/hibernate/hibernate_architecture.htm

[5] Hibernate ORM – April 02, 2016

http://www.javabeat.net/hibernate-interview-questions/

[6] JavaScript Basics – April 04, 2016

http://www.w3schools.com/js/default.asp

[7] JQuery Methods – April 04, 2016

http://api.jquery.com/jquery.post/

[8] Apache Tomcat – April 04, 2016

https://en.wikipedia.org/wiki/Apache_Tomcat

[9] Apache JMeter – Performance Testing – April 9, 2016

http://jmeter.apache.org/

[10] Spring Framework Introduction – April 01, 2016

https://docs.spring.io/spring/docs/current/spring-framework-reference/html/overview.html