**EFFICIENT TRAVEL DECISION MAKING USING WEB APPLICATION BASED ON MVC ARCHITECTURE**

A PROJECT REPORT

Presented to the Department of Computer Engineering and Computer Science

California State University, Long Beach

In Partial Fulfilment

of the Requirements for the Degree

Master of Science in Computer Science

Committee Members:

Frank Murgolo, Ph.D. (Chair)

Michael Hoffman, Ph.D.

Birgit Penzenstadler, Ph.D.

College Designee:

Antonella Sciortino, Ph.D.

By Keval Soni

B.E, 2011, University of Mumbai, India

May 2017

ABSTRACT

**EFFICIENT TRAVEL DECISION MAKING USING WEB APPLICATION BASED ON MVC ARCHITECTURE**

By

Keval Soni

May 2017

Software Engineering is a dynamic field where web applications provide services to users over the Internet. Web applications are constantly updated to incorporate new functionality and improve software quality. Web applications are developed based on Model-View-Controller (MVC) architecture. MVC architecture divides the application into different modules: user interface, intermediate controller, and data persistence. This modular approach helps achieve loose coupling between user interface and business logic, and it facilitates ease in maintenance of application and independence in upgrading the application.

The project showcases the benefits of using MVC architecture, Spring MVC Framework, and Restful service integration. It reflects upon the system analysis and design models built by using Object Oriented Analysis and Design methodology and Unified Modeling Language (UML) diagrams**.**

The application presents travel data from different travel modes in a unified view for efficient decision making. Users canschedule future travel plans and get notified about those plans.

**TABLE OF CONTENTS**

ABSTRACT………………………………………………………..…………………….……… ii

LIST OF TABLES……………………………………………………………………..………... iv

LIST OF FIGURES……………………………………………….………..……………………. v

LIST OF ABBREVIATIONS…………………………………….……………………………. vii

1. INTRODUCTION…………………………………………….…………...……….….…….. 1

2. SYSTEM ANALYSIS AND DESIGN……………………….…………………...………… 4

3. WEB TECHNOLOGIES……………………………………..……..………..…………..… 10

4. IMPLEMENTATION DETAILS…………………………………..………………………. 22

5. RESULTANT SYSTEM…………………………………………..….................................. 29

6. CONCLUSION…………………………………………………….…………….………… 41

APPENDICES………………………………………………………….………………………. 43

A. NON-FUNCTIONAL REQUIREMENTS.………………………..………………………. 44

B. USE CASES MODELS………………………………………..…………………………... 47

C. CLASS DIAGRAM………………………………………………..………………………. 57

D. SEQUENCE DIAGRAMS………………………………………..……………………….. 59

E. DATABASE TABLES………………………………………….….…………………….... 63

REFERENCES………………………………………………………….…………………….... 66

**LIST OF TABLES**

1. New User Registration Use Case…………………………………………..…………… 48
2. User Login Use Case…….……...………………………………………….…………... 49
3. Search Economic Travel Option Use Case……...…………………………..…..……… 50
4. Search Faster Travel Option Use Case…...………………………………...…………... 51
5. Save Travel Decision Use Case………...…………………………………….………… 52
6. Schedule Future Travel for Notification Use Case………………………….….………. 53
7. Send Push Notification Use Case………………………....………………….………… 54
8. View Travel Search History Use Case………………….…..…….…………….……… 55
9. View Travel Schedule History Use Case……………………..…….………….…….… 55
10. Filter Report Data on Date Range Use Case……………………...………….………… 56
11. Download Travel History Report Use Case………………………...……….….……….57

**LIST OF FIGURES**

1. 3-tier web application architecture...............................………………..……………....………. 1

2. MVC architecture…………………………………………………………………..………… 11

3. Flow of request in Spring MVC ……………………………………………………..……..…12

4. Hibernate architecture………………………………………………………………..….…… 16

5. Cloud messaging components for push notification messages……………………..………... 20

6. Landing page…………………………….…………………………………………..……..… 30

7. Login page………………………………………………………………………….…..……..31

8. Login failed message……………………………………………………………….………....32

9. New account signup form …………………………………………………………..……...... 32

10. Signup form validation errors ……………………………………………………...………. 33

11. Travel Helper homepage ………………………………………………………….……….. 34

12. Travel Search page ………………………………………………………………….……... 34

13. Travel Search result………………………………………………………………….……... 35

14. Enable Push Notification page …………………..………………………………….……... 36

15. Schedule Travel page………………………………………………...…………….………. 36

16. Push Notification message………………………………………………….……….……... 37

17. Dashboard home page…………………………………………………….……….……..… 38

18. Travel Search Dashboard page…..……………………………………….……….………... 38

19. Schedule Travel Dashboard page……………………………………….…………….…….. 39

20. Schedule Travel Detailed history……………………………………….……………….….. 39

21. Schedule Travel History downloaded report …………………………………………….… 40

22. Filter results option for Schedule Travel history …………………….…………………….. 40

23. Travel Helper class diagram……………………………..……………………………….… 58

24. Sequence diagram for new account creation…………………………………….……….… 60

25. Sequence diagram for User Login……………………………….……………..…………... 60

26. Sequence diagram for efficient travel search…………………….…………….………..….. 61

27. Sequence diagram for scheduling future travel plan…………….…………….…………..... 62

28. Sequence diagram for display travel history……………………….…………..………….... 62

**LIST OF ABBREVIATIONS**

ADA Americans with Disabilities Act

AJAX Asynchronous Javascript Request

AOP Aspect Oriented Programming

API Application Programming Interface

CSS Cascading Style Sheets

DI Dependency Injection

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

HQL Hibernate Query Language

IoC Inversion of Control

J2EE Java 2 Enterprise Edition

JDBC Java Database Connectivity

JSP Java Server Pages

MVC Model-View-Controller

OOAD Object Oriented Analysis and Design

ORM Object Relational Mapping

POJO Plain Old Java Object

RAD Requirement Analysis Document

REST Representational State Transfer

SDLC Software Development Life Cycle

SOAP Simple Object Access Protocol

SRS Software Requirement Specification

UML Unified Modeling Language

URI Uniform Resource Identifier

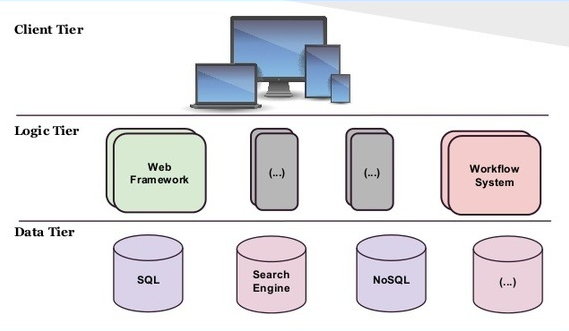
WWW World Wide Web

**CHAPTER 1**

**INTRODUCTION**

Software engineering is a young and dynamic discipline used to solve real world problems. The demand of software has always been on the rise, and the technology is continually evolving with new and better features. The huge demand of software gave rise to the need for solutions that can help in faster and simpler methods for development of the software. With the exponential rise of information technology, software is becoming more complex and the traditional web development methods do not provide solutions to the current requirements of web applications. The software design patterns are proven solutions to recurring design problems [1]. Software engineering architecture and design patterns provide abstract ways and approaches to be implemented while developing new software or resolving core engineering problems.

Web applications have become immensely popular and have taken over the traditional desktop software applications. A web application can be of 2-tier or 3-tier architecture. A 2-tier architecture consists of a client tier and a logic tier, while a 3-tier application consists of a client tier, a logic tier, and a data tier [2]. Figure 1 shows the three tiers of a web application and function of each tier is as follows:



**FIGURE 1. 3-tier web application architecture.**

*Client tier*: Client tier is the user facing layer consisting of the user interface. The user interface renders a web page to display information. Users browse the web page, send web request, and perform various actions on the user interface to fetch or save data.

*Logic tier*: Application logic is deployed on logic tier. It acts as a coordinator between the client and the data tier. Requests from the client layer are processed and business logic is executed to store or retrieve information. Results are sent back to the user interface.

*Data tier*: Data tierconsists of a database with a database management system for handling data. Logic tier connects to the data tier and executes database queries for storing or retrieving data.

Web applications using legacy systems are tightly coupled. The fast growth of web applications resulted in the need for error free application development and maintenance. The frequent problems with the development and maintenance of legacy web applications gave rise to the need for a development environment, which will provide less coupling, high cohesion, independent module design, and better scalability [3].

Model-View-Controller (MVC) is a software architecture for developing loosely coupled web applications. MVC architecture promotes modular application development and maintenance. Use of MVC architecture has been on the rise due to separation of the user interface and the business logic. MVC architecture is made up of the three components: the model, the view, and the controller. The model contains the core functionality of the application. The view provides a client facing user interface responsible for displaying the state of the model. The controller handles mapping of the user request and connects the view and the model [4].

Spring is a Java based framework containing Spring MVC module, which provides model-view-controller architecture that can be used to develop web applications. The goal is to develop web application with the objectives of simplicity, portability, and maintainability. Spring framework satisfies the objectives by providing MVC module, Inversion of Control (IoC), and stable integration with Hibernate Object Relational Mapping (ORM) [5].

Travel Helper is a web application developed using the Spring MVC framework. The various modules of Travel Helper application are User Profiling, Travel Search, Schedule Travel, and Dashboard.

*User Profiling*: This module handled the user profile and authentication.

*Travel Search*: This module is used to make efficient travel decisions. The travel source and destination are used to gather travel data. Travel Search provides a unified view of many travel modes such as walking, bicycling, driving, public transit, Uber, and Lyft. It provides comprehensive detail of distance, duration, and cost of each travel mode for decision making.

*Schedule Search*: This module is used to store future travel plans and preferred travel mode. The application alerts the user to begin traveling by sending a push notification to the user’s device.

*Dashboard*: This module is used to analyse past travel preferences and download travel history.

Travel Helper web application performs comprehensive data collection of travel information using Javascript promise and various RESTful services. The project showcases the importance and the benefits of using MVC architecture, Spring MVC Framework with Hibernate, and Restful web service integration. This project also reflects upon the system analysis and design models built by using Object Oriented Analysis and Design methodology and Unified Modeling Language (UML) diagrams.

**CHAPTER 2**

**SYSTEM ANALYSIS AND DESIGN**

Software applications undergo the Software Development Life Cycle (SDLC) that involve requirement gathering, system design, application development, quality analysis, and maintenance. Different strategies such as extreme programming, agile, and scrum can be employed to achieve efficient software development and maintenance plans. System analysis and design models built using object oriented methodology and UML help in understanding the purpose of software application and act as a blueprint to implement the software.

**Object Oriented Analysis and Design**

In software engineering, Object Oriented Analysis and Design (OOAD) approach is important for successful planning and execution of web applications, which can function efficiently over a long period of time. OOAD approach requires clear understanding of requirements to develop design models and implement the application based on design models [6].

The first step of software development lifecycle is to gather requirements from stakeholders or describe a problem statement of software to be built. The output of the requirement phase of software engineering is Software Requirement Specification (SRS), also known as Requirement Analysis Document (RAD). RAD document lays a foundation for software design and implementation, and it describes the functional and non-functional requirements of the software. It includes UML diagrams such as use cases models, sequence diagrams, and class diagram to describe the relationship between different modules of the application and user actions with application interface. UML diagrams represent the interaction between the user and the software system through tables and graphical representations [7]. These diagrams help to understand and visualize the user interface and actions that can be performed to achieve expected results. The following sections describe the requirement analysis and design using UML models for Travel Helper application.

**Functional and Non-Functional Requirement**

Requirement analysis is performed to document the functional and non-functional requirements of the application. The system design outlines a plan to implement functional requirements; whereas, system architecture outlines a plan to implement non-functional requirement. Functional requirements capture the working requirements of the software and illustrate the expected functionality of the system. Functional requirements are described using use case models (see Appendix B). Non-functional requirements describe expected behaviour of the functionality specified for the system and specify the design constraints and characteristics that need to be taken care of for satisfying the user requirements. They specify the behaviour of the system with respect to various quantifiable attributes such as performance, availability,and scalability [8]. Non-functional requirements applicable to Travel Helper application are listed in Appendix A.

**Use Case**

Use case is a behavioural diagram. It depicts the sequence of actions a user of a system can perform to achieve the functionality of the system. The user has a role in the system which needs to be satisfied. Use case describes complete details of the role, condition, and action corresponding to the functional requirement of the system. There are different formats to represent the use cases. The seven sections of the use case model used for this application are explained as follows:

1. Use Case Name: Describes the name of the user action.
2. Actors: External agents responsible for performing specified action.
3. Pre Conditions: Conditions that need to be satisfied before the action specified in the use case can be performed successfully.
4. Flow of Control: Lists the steps in order of actions that need to be performed to execute the use case.
5. Post Conditions: Conditions that will be true after the actor has completed actions based on the steps listed in the Flow of Control required for the use case.
6. Error Conditions: Unsuccessful execution of the action specified by the use case due to possible errors or issues that can occur during the execution of the use case.
7. Non-Functional Requirements: Lists the quantifiable non-functional requirements that need to be satisfied for actions performed in the use case.

The following list contains the use cases designed to describe various functionalities for the Travel Helper application, which are illustrated in the tables of Appendix B.

1. New user registration: This use case depicts the actions that need to be performed by the user to create a new profile in the application. Detailed steps of this use case are present in Use Case Table 1.
2. User login: The user logs into an application by following the steps presented in Use Case Table 2.
3. Search economical travel option: The user finds the economical travel option by following the steps listed in Use Case Table 3.
4. Search faster travel option: The user finds the faster travel option to reach a destination in less travel time by following steps listed in Use Case Table 4.
5. Save travel decision: The user saves the preferred travel option in the application by

following steps mentioned in Use Case Table 5.

1. Schedule future travel for notification: The user follows steps mentioned in Use Case Table 6 to schedule future travel plan to receive travel start time notification.
2. Send push notification: System notifies the user to start travel by sending push notification based on Use Case Table 7.
3. View travel search history: The user can view travel search preference and detailed history by following steps in Use Case Table 8.
4. View schedule travel history: The user can view travel schedule preference and detailed history by following steps in Use Case Table 9.
5. Filter report data on date range: The user can filter report data using date range for historical travel analysis following steps in Use Case Table 10.
6. Download travel history report: The user can download report by following steps listed in Use Case Table 11.

**Class Diagram**

Class diagram represents high level structure of the software system. It is a diagram that shows the classes of the system and the relationship that exists between the classes. Classes are entities in the application, and each class has attributes and methods. The class diagram shows relationship between classes and cardinality of the relationship. Visibility modes such as public, protected, default, and private can be used to show the scope of the attributes and classes. Attributes should be private or protected. Multiplicity is used to show cardinality in relationship between classes. Dependency is shown between classes using inheritance, association, dependency, aggregation, and composition relation. Application domain experts use class diagram to model the application domain. Developers use the class diagram during the application system design and implementation. The class diagram for Travel Helper application is shown in Appendix C.

**Sequence Diagram**

Sequence Diagram is a graphical description of the objects participating in a use case or scenario using a directed acyclic graph notation. Each use case can be modelled as a sequence diagram showing the actions of the user and request flow in the system in response to the action. Three important parts of sequence diagram are boundary, control, and entity. Boundary objects interface with system actors and interact with the actor and control objects. Control objects also known as controller, handle the flow of request in the application and mediate between boundaries and entities. The entity object represents the system data and only interacts with the controller.Sequence diagrams for Travel Helper application are present is Appendix D.

**Mockups**

The mockups are the prototype of the user interface that assist developers to visualize how the user interface will look. It is important that mockups are well-designed to provide great usability experience to users. The wireframes are sketches of the user interface, which provide a skeleton and visual guide of the web pages. The mockups are created based on the user interface designed in wireframes. Hypertext Markup Language (HTML) pages, Javascript, and Cascading Style Sheets (CSS) are the building blocks of mockups. The mockups for Travel Helper application are represented in Figures 7 through 17 in Chapter 5.

**Database Schema**

The database schema represents the logical view of the database. It is the blueprint of how the data will be stored. The database schema consists of tables, attributes, and relationships between the other tables of the application. It also specifies constraints on data that can be inserted into tables. Thus, the database schema acts as a complete design document to visualize the complete flow of data in the application. Database tables for Travel Helper application are included in Appendix E.

**CHAPTER 3**

**WEB TECHNOLOGIES**

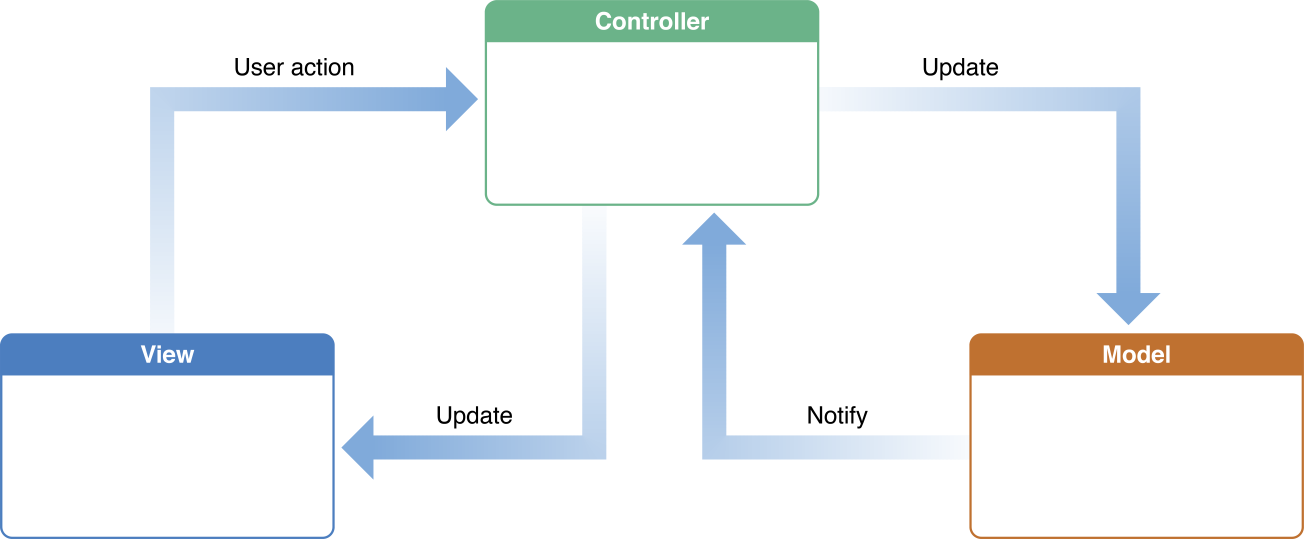
**Client Server Architecture**

The traditional web applications were developed using client server architecture. The client and server machine can reside on the same computer system or may reside in different infrastructure that is connected using the computer network and the Internet. The server can host a single or multiple instances of the application and share resources with the client. The user sends the request to the application server using a client browser. A common protocol used by the client and server to exchange messages is Hypertext Transfer Protocol (HTTP). The server receives the request from the client by various HTTP methods. The server hosts the application business logic and is connected to the database server. Thus, the server processes the client request, performs computation, and sends response back to client to display the result in the browser. World Wide Web (WWW), Email, and Network printers are developed using client server architecture.

However, traditional web applications using client server architecture are tightly coupled and difficult to maintain. The frequent problems encountered in traditional web application development gave rise to the need for a loosely coupled architecture to ease web application development and maintenance.

**Model-View-Controller Architecture**

Model-View-Controller (MVC) architecture is used in the majority of web applications.MVC architecture separates the user interface, control, and data persistence [3]. The modularity helps in the ease of the web application development as it separates coupling between system interface and core functionality. Three main components of MVC architecture are the model, the view and the controller. Figure 2 shows different components of the MVC architecture and flow of control between the components.



**FIGURE 2. MVC architecture [9].**

*The model*:The model incorporates the core functionality of the application. It performs the computation, establishes connectivity with database to execute queries, and handles the transaction management. The controller sends the request to the model, and results are returned back to the controller after processing the request. The model is independent of how the data will be rendered in the view.

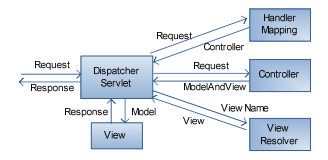
*The view*: The view is the user interface of the application. The data is rendered in the view, and it represents the state of the model. The view handles the rendering and styling of information on user interface. The users interact with the view and perform actions to execute the functionality of the application.

*The controller*: The controller handles the flow of control and data in the application. The user requests are initially processed by the controller. Based on the request mapping, the controller creates the objects and invokes the model to execute the functionality. The model returns the data to the controller, and the controller forwards the data to the corresponding view for displaying the results.

The advantages of MVC architecture are low coupling and high cohesion between application modules, flexibility in designing multiple views for different actions, clear application design, better scalability, and ease in maintenance of the application [3].

**Spring MVC Framework**

The enterprise web applications are generally developed using framework that supports MVC architecture. Spring is a famous Java 2 Enterprise Edition (J2EE) based framework. Spring supports Inversion of Control (IoC) and Aspect-Oriented Programming (AOP) [10]. Spring framework has various modules to support structured application development such as core module, aspect oriented programming module, data access module, object relational mapping module and web module. The web module of Spring framework is known as Spring MVC. Spring MVC module is used for web application development. Spring MVC leverages the benefits of MVC architecture and benefits developers in application development and maintenance by promoting code reusability. Spring MVC is request-driven and uses a central dispatcher servlet to control flow of request within the application [11]. Figure 3 shows the flow of request in Spring MVC.



**FIGURE 3. Flow of request in Spring MVC [10].**

As shown in Figure 3, the user request is sent to a centralised servlet known as dispatcher servlet. The dispatcher servlet is the central controller, which contains configuration for the handler mapping, the controller, and the view resolver. The dispatcher servlet forwards the request to the handler mapping. The handler mapping returns the base class path of the controller to the dispatcher servlet. Further, dispatcher servlet forwards the request to the controller for mapping the user request to execute the function. The controller invokes the service class of the model. The model executes the business logic and forwards the result back to the controller. The controller creates a ModelAndView object and stores the result in the object. The ModelAndView object is sent to the dispatcher servlet to invoke the appropriate view. Then, the dispatcher servlet sends the view name to the view resolver to render the view and display information on the web page [10].

Spring supports Inversion of Control (IoC) which is also known as Dependency Injection (DI). DI plays an important role in achieving loose coupling between dependent modules. Spring MVC supports annotations to simplify the code, eliminate configuration file, and increase code reusability. Therefore, Spring MVC improves code structure and reduces complexity. It provides customization bindings, better security, and performance improvement features. The following sections explain some of the spring framework components used in the implementation of Travel Helper application.

**Dependency Injection**

In software engineering, DI plays an important role to achieve loose coupling between application modules. DI helps to achieve modularity, reduce complexity, and improve maintainability of the code. As stated by Martin [12], the concept of the dependency injection is that “High-level modules should not depend upon low-level modules. Both should depend upon abstractions.” It is fundamentally the reverse process where objects define the dependencies and control initialization of dependent classes using Spring IoC container. The objects that control the DI process are called beans. Spring IoC container handles the instantiation of bean objects and inject dependency among them.

The configuration for dependency is defined using the Spring XML configuration file or annotation. The container injects the dependency at runtime by resolving the dependent class using the configuration. The Spring framework injects dependency using constructor or setter method. Therefore, the dependency injection is an important principle to achieve loose coupling and simplification.

**Transaction Management**

Transaction management is a critical aspect of the web application functionality. The Spring framework provides programmatic transaction management which is more efficient compared to the traditional global and local transaction management methods [13]. Spring transaction management is easy to integrate and provides a consistent model which is independent of the infrastructure. The Spring framework provides annotations to implement the transaction management which requires minimum code to be written and improves the code reusability.

**Spring Scheduler**

Scheduling is an important requirement of every web application. Every application has some background job to be executed periodically over the time it is running. The periodical execution of the job is done using schedulers. Schedulers are configured to run at a specified time interval and perform file or database operations. The Spring framework provides classes for scheduling a block of code to be executed as a job and configuration is done using annotations.

**Spring Security**

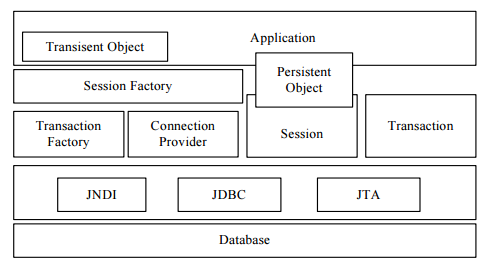
Every web application is required to implement security features to prevent malicious

activity. The web applications are exposed to many vulnerabilities, which are exploited by malicious users to gain confidential information of authenticated users or cause damage to the web application. Spring security module provides the enhanced security implementation features for web applications. Two major operations provided by Spring security are authentication and authorization [14]. Authentication is implemented to ensure that the valid user is able to access personal account in the application. Authentication ensures that the users with valid account get access to the application. Authorization is implemented to ensure that the information accessible to the user and scope of action is limited to the role defined for the user in the application. Authorization ensures that the access-control is implemented and that users are able to access the application based on the role assigned to them.

Spring security provides features of servlet filters to perform security checks and audit each request, before the request reaches the controller. Each user requests are initially captured by the Spring Delegating Filter. Filters are implemented to perform authentication and authentication checks on the request before the request reaches the controller known as a Spring Dispatcher Servlet. Once the request goes through each filter check, the dispatcher servlet performs the request mapping function and executes the business logic. Spring security provides interceptor filters which are used to customize access control to web pages based on the user authentication and user role. The user session information can be accessed on each web page using Spring security tag library. Therefore, Spring security provides high customization capabilities to handle each request based on its context. Spring security has deep defence capabilities and can prevent common web application vulnerabilities.

**Hibernate**

The traditional J2EE web applications used Java Database Connectivity (JDBC) connection to establish the database connectivity. JDBC connectivity caused connection problems during peak load because it was difficult to configure and manage the huge number of database connections. Another problem of JDBC connectivity was the need to write complex database SQL queries. The current scale of web application usage requires high number of reliable database connections. Hibernate is an Object Relational Mapping (ORM) framework for Java environments. Hibernate is integrated with Spring framework to support object oriented query framework. It allows mapping of columns in the database to attributes of classes. Therefore, Hibernate facilitates object relational mapping between Java objects and relational database. Hibernate architecture is shown in Figure 4.



**FIGURE 4. Hibernate architecture [15].**

As shown in Figure 4, Session is established for communication between the application and the database. Transaction Factory represents the pool of connection created by the Session Factory. The Persistent Object represents the serialized object used for executing database query and manipulating object values [15]. Hibernate framework handles the execution of query to perform database operation and return results in the form of Java objects. Hibernate also provides an alternative to execute complex SQL queries using Hibernate Query Language (HQL).

Hibernate uses a configuration file to retrieve database connectivity configurations. Mapping of the tables and the Java objects are stored using the configuration file. The Java class contains a mapping of tables and columns with class name and member variables. Hibernate uses Plain Old Java Object (POJO) methods to perform database operations. The new object is initialized to create a new record in the database, and setter function is used to store attribute values. Select, Update and Delete operations are performed by retrieving the record from the database in the form of an object, and various manipulations are performed on the object using Hibernate functions. Therefore, Hibernate is useful for web application, with heavy request load since it enhances scalability, reliability, extensibility, and provides high performance.

**Web Services**

Web services are primarily designed to facilitate communication between two software applications over the Internet. Web services use open standard protocols such as Simple Object Access Protocol (SOAP) and Hypertext Transfer Protocol (HTTP) for communication. They are useful for communication between two heterogeneous applications. Web services are accessible through the registry or exposed through an Application Programming Interface (API) over the Internet. Representational State Transfer (REST) web service also known as RESTful web service or REST API run on HTTP protocol. Web services are platform independent and provide greater interoperability.

The three major components of a web service architecture are:

*Producer:* The producer is an application that wants to share data with other applications.

A producer provides access to the data through an API which is hosted on the application server and accessible over the Internet.

*Consumer*: The consumer is an application that wants to fetch data from another application. The consumer has information about different URI or methods exposed by the producer for various operations. The consumer establishes the connection with the web service and sends the request to the producer.

*Registry*: The registry is a centralized database for all web services exposed over the Internet. The registry enables a seamless integration of different and heterogeneous applications over SOAP or HTTP protocol.

**Representational State Transfer (REST) API**

REST API or RESTful web service are used to establish communication and share data between two software applications. REST API are preferred by web applications since they are lightweight and suitable for distributed applications.

The use of REST API has increased in current web applications. With the huge amount of data being generated and processed by different applications, REST API are implemented by every web application to share information with other applications or to consume information. REST APIs enable web applications to communicate and integrate easily using common data formats such as XML and JSON.

The producers host the REST API on an application server, which is accessible on the Internet over the HTTP protocol and exposed as an API for other applications to integrate**.** REST API exposes every operation with unique Uniform Resource Identifier (URI) and other applications use HTTP methods such as GET or POST to send request to the RESTful service. The advantages of RESTful web services are clear separation of client and server, platform independence, reliability and scalability.

**Javascript Promise**

Javascript Promise are proxy objects that provide synchronous behaviour to asynchronous requests. It provides listeners to asynchronous request success or failure event. A promise object has one of these states: pending, fulfilled, and rejected. Promise object can handle the time at which the result of asynchronous is returned back to the calling function. The Promise object can be fulfilled or rejected. When the Promise object is resolved by one of the states, the associated then method of Promise is called to execute queued asynchronous request. REST APIs are invoked using Asynchronous Javascript Request (AJAX). The asynchronous calls to the REST API are stateless, and the application cannot handle the response from the API. Therefore, it is difficult to implement synchronous AJAX requests for multiple interdependent API calls [16].

Javascript Promise can be used to resolve the problem of interdependent API calls and achieve synchronous AJAX requests behaviour. The Promise object has various methods such as Promise.all() and Promise.resolve() which can be used to synchronize dependent REST API requests. The dependent asynchronous requests queued up in then() method execute only after Javascript Promise object changes to fulfilled or rejected state. Therefore, Javascript Promise is used in Travel Helper application to gather data from various REST APIs and build a single data structure containing travel information from all the integrated APIs.

**Cloud Messaging and Push Notification**

The push notification is used by the majority of the web applications to send updates and alerts to subscribed users. The push notification does not require users to be actively using the application. Notification is sent to the user device by the cloud messaging server. Travel Helper uses push notification to alert the users to start the travel based on saved preference to reach destination on time. Figure 5 shows the overview of cloud messaging components for push notification.



**FIGURE 5. Cloud messaging components for push notification messages [17].**

As shown in Figure 5, web application is hosted on an application server and uses HTTP or XMPP protocol to communicate with the Firebase Cloud Messaging (FCM) server. FCM server is used to send push messages to the devices. Notifications GUI console stores the information related to the cloud messaging.

The web application establishes connectivity with the FCM server to handle upstream and downstream messages. It registers a unique identifier called a registration token with the FCM server. The registration token is used to obtain a unique device identifier for each device, which can be a desktop, tablet, or mobile phone. Unique identifier helps to send push notification to correct device from FCM server. Application invokes FCM API to send push message to the user’s device using unique device identifier. Service workers are implemented as background service to listen to cloud messages and display push message in user’s device.

**CHAPTER 4**

**IMPLEMENTATION DETAILS**

The technologies discussed in previous chapters have been implemented to develop the Travel Helper web application. The main modules of the Travel Helper application are User Profiling, Travel Search, Schedule Search and Dashboard. The following sections discuss the implementation details.

**Model-View-Controller Architecture**

Travel Helper is developed using J2EE based Spring Framework. Spring MVC module provides support for MVC architecture. The following sections illustrates the breakdown of application modules in the model, the view, and the controller.

**The Model**

The model contains the core business logic of the application and handles the data persistence. Spring exposes classes of the model as a service to the controller. The classes implement the interface and contain the business logic. The model classes use ORM technology Hibernate and POJO for data persistence. The two main model classes of Travel Helper application are UserService and TravelService. UserService class exposes the functionality related to the user data and implements user profiling. TravelService class exposes the functionality related to Travel Search, Schedule Search, and Dashboard.

Dependency injection (DI) is used to initialize objects of implementing classes. Spring MVC uses configuration XML file (travelhelper-servlet.xml) and @Autowired spring annotation to inject dependency. The following code snippet shows the configuration used to inject dependency of class UserProfileDAOImpl in class reference userProfileDao. The travelhelper-servlet.xml contains the Spring configuration.

<bean id="userProfileDao" class="com.travelhelper.dao.UserProfileDAOImpl">

</bean>

The following code snippet is used to invoke the method of implementation class.

@Autowired

private UserProfileDAO userProfileDao;

userProfileDao reference calls the implementation method.

userProfileDao.fetchUserIdfromUsername(username);

Transaction Management is implemented using @EnableTransactionManagement Spring annotation to achieve data integrity during database operation. This annotation handles the transaction lock for the code that needs to be executed in a single transaction. It manages the database session and handles repository bean mapping to execute database operations.

Hibernate is used for handling database queries. The Hibernate framework provides various methods for Select, Insert, Update, and Delete database operations. Therefore, the use of Hibernate framework methods greatly reduces the problem of writing complex queries and provides a simple interface for performing database operations. It also has the option for writing SQL queries by using HQL. The model classes containing POJO are created for data persistence, and Hibernate ORM mapping between database tables and Java class.

**The View**

The view contains the user interface or front-end web pages of the application. The view renders the data on webpage and handles formatting using styles. Travel Helper uses Java Server Pages (JSP) to display data on the user interface. The content styling and scripting is handled by Bootstrap framework and front-end technologies such as CSS and Javascript. The controller creates the ModelAndView object for the view and returns the name of the view to ViewResolver. The following code shows the ViewResolver configuration in the Spring configuration file.

<bean id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver">

<property name="prefix" value="/WEB-INF/jsp/" />

<property name="suffix" value=".jsp" />

</bean>

The prefix and suffix property are used to configure the actual path of the web page. ViewResolver uses these properties to display the web page in browser from the configured path.

**The Controller**

The controller is the first component that receives the user request and handles the flow of control between the model and the view. Spring detects the controller class using @Controller annotation. The base package of the controller classes is configured in the Spring configuration file. The following code snippet specifies the base package for the controller in the Spring configuration file.

*<context:component-scan base-package="com.travelhelper.controller" />*

The following code snippet of controller class shows the request mapping and the method calls to the model and the view.

@RequestMapping("/signupform")

public ModelAndView processSignupForm(ModelMap model,UserProfile profile) {

int id = userProfileService.createNewUserProfile(profile);

return new ModelAndView("login","successMessage", "Account Created");

}

@RequestMapping("/signupform") Spring annotation maps the request URI for the signup form. The model class is executed using userProfileService reference, and the object is initialized using DI. createNewUserProfile(profile) method executes the core logic for creating a new user profile and returns the new record identifier created for the new profile. The controller returns the ModelAndView object to ViewResolver, which renders the login page and displays the confirmation message.

**Javascript Promise Implementation**

The Javascript Promise is used to synchronize various asynchronous REST API calls. Travel Helper is using Javascript Promise to gather travel data from various REST APIs and build a single data structure containing travel information. The Travel Search module makes asynchronous calls to different REST APIs of Google, Uber, and Lyft. Some of the API calls are interdependent and need data from another REST API to fetch more information. Javascript Promise allows synchronization of dependent REST API calls by implementing Promise object. The dependent REST API calls are placed in “then” block, and it can only execute when Promise object is resolved. The following code snippet shows the implementation of Javascript Promise.

Promise.all([distanceCalculationTransit(travelSearchDetailsJson),

distanceCalculationDriving(travelSearchDetailsJson),

distanceCalculationWalking(travelSearchDetailsJson),

distanceCalculationBicycling(travelSearchDetailsJson),

]).then(function(){

fetchRideEstimate(travelSearchDetailsJson);

}).then(function(){

setTimeout(function(){

showTravelDetails(travelSearchDetailsJson);

},15000);

});

The various methods making REST API calls are placed inside the Promise.all() block. Each method declares a Promise object before making a REST API call. Once the response is received from REST API, it resolves the Promise object. The following code snippet shows the resolution of the promise object.

var distanceCalculationTransit = function(travelSearchDetailsJson){

return new Promise(function(resolve,reject){

// Call to REST API

resolve();

});

}

After the resolution of Promise object initialized by various methods in Promise.all()block, the Javascript Promise passes the control to then() block. The then()block contains the methods that make REST API calls using data returned by previous services. Therefore, Javascript Promise creates an environment for synchronous execution of dependent asynchronous REST API using Promise object.

**Push Notification Implementation**

Travel Helper provides feature to schedule future travel plans and to notify the user at the travel start time.The users can save the time at which they want to reach the destination and preferred mode of travel in the Schedule Search module. The application alerts the user to start travel based on their schedule by sending a push notification. To enable a push notification, Travel Helper sends a request to the FCM server for obtaining the unique identifier for each device. Travel Helper registers each device on FCM server and maintains the unique device identifier in the Travel Helper database to send a push notification.

Spring scheduler runs every minute to fetch details of the users that need to be notified about the start of travel. Travel Helper sends a request to the FCM server along with the device identifier to send a push notification message. The FCM server message is captured by the service worker running in the background of the browser. The service worker is customized to listen to the FCM server messages and send push notification on a device. A push notification contains details about the message that need to displayed to user.

**REST API Integration**

Travel Helper is integrated with REST API provided by Google, Uber, and Lyft to obtain travel related data. The following sections presents the details about the different API used by Travel Helper application.

**Integration with Google**

*Google Maps API*: Google Maps API provides autocomplete feature using Places library. It provides type-ahead-search feature for address entered by the user similar to Google Search. The suggestions provided by the API based on user input, assists the user to find correct location details [18].

URI: https://maps.googleapis.com/maps/api/js

*Google Distance Matrix API*: The Google Maps Distance Matrix API is a service that provides travel distance and duration for a matrix of origins and destinations. Distance and duration details are available for walking, driving, bicycling, and public transit modes [18].

URI: https://maps.googleapis.com/maps/api/distancematrix/json

*Google GeoCode API*: Geocoding API converts the addresses information into geographical coordinates such as latitude and longitude [18].

URI: https://maps.googleapis.com/maps/api/geocode/json

*Firebase Cloud Messaging Push Notification API*: This API is called to send push message to user device. Unique ID of user’s device is passed in the API and FCM server sends the message to user device, where service worker listens to the message and invokes push notification [17].

URI: https://fcm.googleapis.com/fcm/send

**Integration with Uber**

*Products API*: The Products API returns the information about various Uber products available at the specified location. The response from endpoint contains information with display name for each available product [19].

URI: https://api.uber.com/v1.2/products

*Price Estimate API*: The Price Estimates API provides price range for each Uber product. The price range is approximate and depends on the location at which product is requested [19].

URI: https://api.uber.com/v1.2/estimates/price

*Time Estimate API*:The Time Estimates API returns the estimated time for arrival of Uber product that is currently available at a given location. The time information provided by the API is real-time and expressed as integers in seconds [19].

URI: https://api.uber.com/v1.2/estimates/time

**Integration with Lyft**

*Ride Types API*: The ridetypes endpoint returns the lyft ride types available at the specified location [20].

URI: https://api.lyft.com/v1/ridetypes

*Cost API*: The cost endpoint returns the estimated cost, distance, and duration of a ride between a start location and end location [20].

URI: https://api.lyft.com/v1/cost

*Time Estimate API*: The ETA (estimated time arrival) endpoint returns the estimated time in seconds it will take for the nearest driver to reach the specified location [20].

URI: https://api.lyft.com/v1/eta

**CHAPTER 5**

**RESULTANT SYSTEM**

This chapter presents the features of the Travel Helper web application. The modules of the Travel Helper web application are User Profiling, Travel Search, Schedule Travel, and Dashboard.

*User Profiling*: This module handles the login page which contains the login form for existing users to login or provides link to sign up form for new account creation.

*Travel Search*: This module is used to make efficient travel decisions using Travel Search page. In Travel Search, the source and destination location are used to determine the route of travel and gather travel data. As compared to obtaining travel information from multiple sources, Travel Helper page provides details of different travel modes such as walking, bicycling, driving, public transit, Uber, and Lyft in a single view. The travel results can be used to make efficient travel decision.

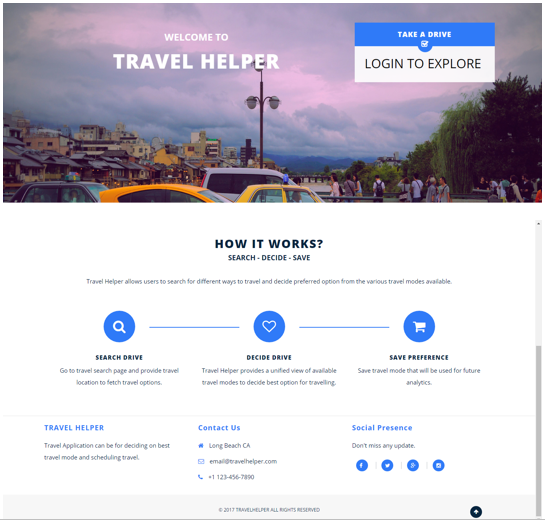
*Schedule Search*: This module allows users to enable push notification and store future travel plans along with preferred travel mode. The application alerts the user at the appropriate time to start the travel by sending a push notification to the user’s device.

*Dashboard*: This module is used to analyse past travel preferences, complete travel history, filter data using date range and download history.

The application is developed using J2EE based Spring MVC framework, which is based on MVC architecture. Loose coupling and clear separation of application modules was achieved using MVC architecture. The model, view and controller class files are modularised and XML file is used for Spring configuration. The Travel Helper application has responsive user interface; therefore, it can be rendered in desktop, tablet and mobile device. The following sections show the resultant Travel Helper web application with Mockups.

**Landing Page**

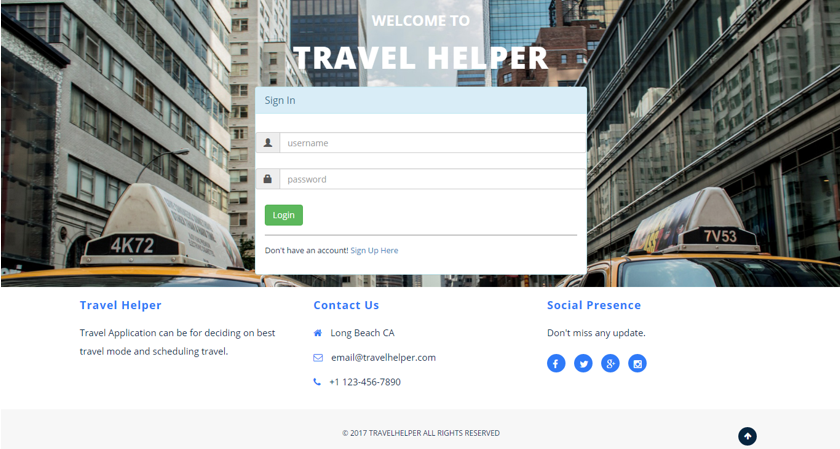
The landing page gives an overall look and feel of the Travel Helper application to the user. The user has to login into the application to use its features. The user clicks on “LOGIN TO EXPLORE” link to access login page. Figure 6 shows the landing page.



**FIGURE 6. Landing page.**

**User Profiling**

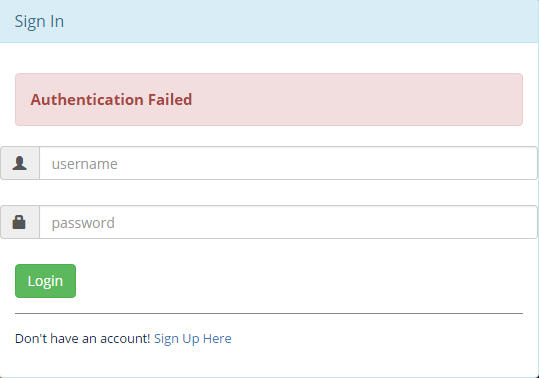
The user will be redirected to login page on click of “LOGIN TO EXPLORE” link on landing page. The username and password fields are used for authentication purpose. The users having an existing account can login through login page and access different features of the application. The user inputs are validated on client side and invalid input will result in authentication error message. Figure 7 shows the login page.



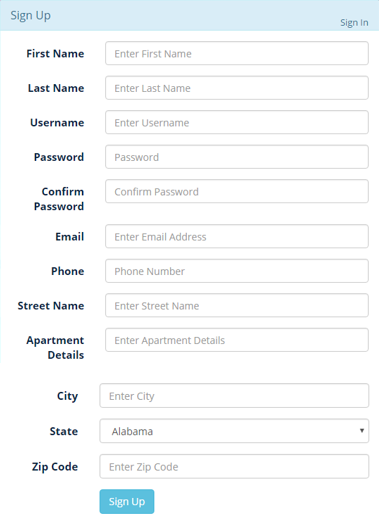
**FIGURE 7. Login page.**

As shown in Figure 7, login form has fields to enter username and password for user authentication purpose. The user input is mandatory for both the fields and should conform to allowed characters and size of input.

Figure 8 shows the authentication failed error message on login page, if user enters incorrect username and password for login.

****

**FIGURE 8. Login failed message.**

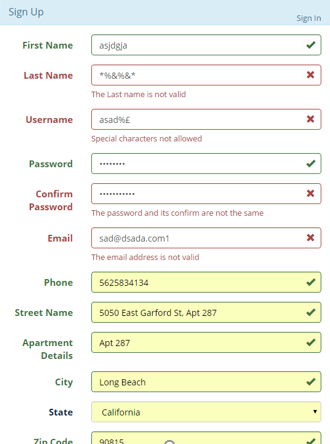


**FIGURE 9. New account signup form.**

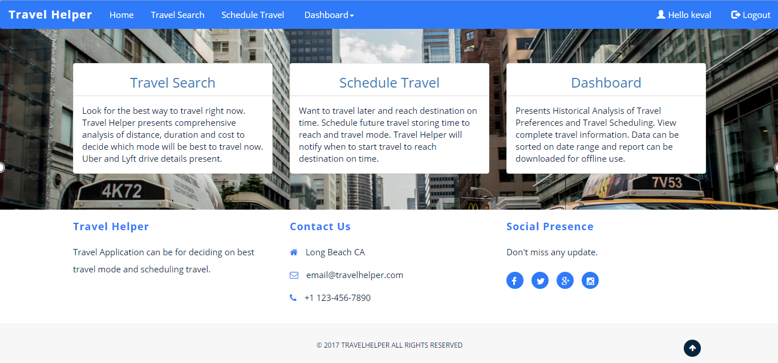
New users create account using “Sign Up Here” link. Figure 9 shows the Sign Up form. The Sign Up form requires personal details of user to successfully create a new account. The user has to enter valid input for successful processing of Sign Up form. Figure 10 shows the error messages displayed to user on entering invalid input. The username selected by the user should be unique. Hence, if the user enters a username which has already been registered in the application by another user, then new user will be shown a message to select another username. Upon successful creation of new account, the user will be redirected to login page. The user has to login with new account credentials and access the homepage.

**Homepage**

Homepage presents brief description of Travel Search, Schedule Search, and Dashboard modules with navigational links to the respective pages of the modules. The header contains navigational links to modules and account logout link. The footer contains contact information and links to social account of Travel Helper application. Figure 11 shows the homepage of the Travel Helper application.



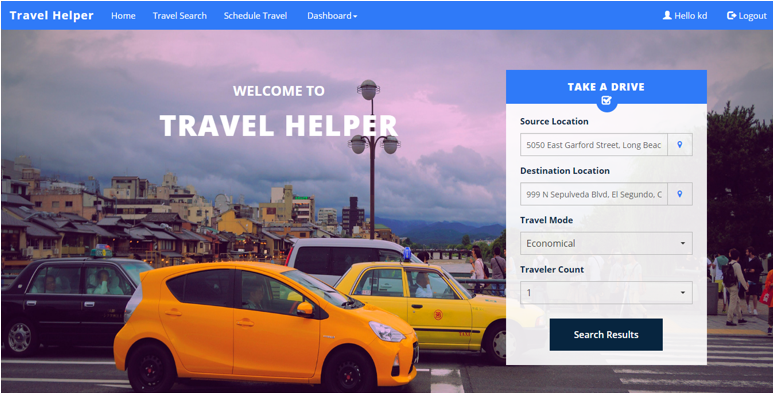
**FIGURE 10. Signup form validation errors.**



**FIGURE 11. Travel Helper homepage.**

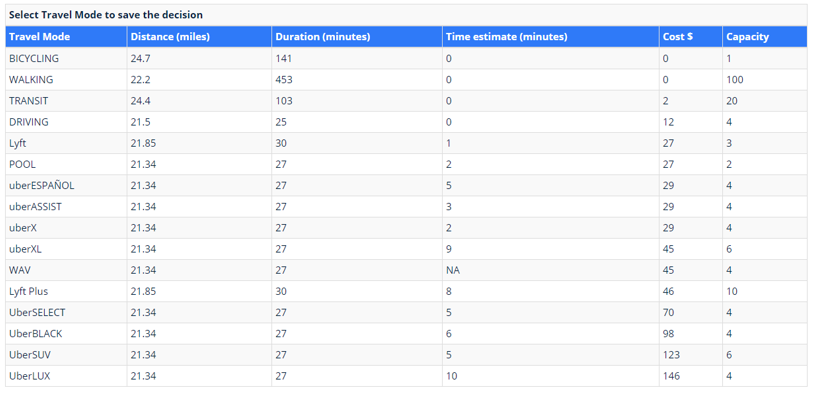
**Travel Search**

Travel Search provides travel information about various travel modes such as walking, bicycling, public transit, driving, Uber, and Lyft to make efficient travel decision. Figure 12 shows the Travel Search page.



**FIGURE 12. Travel Search page.**

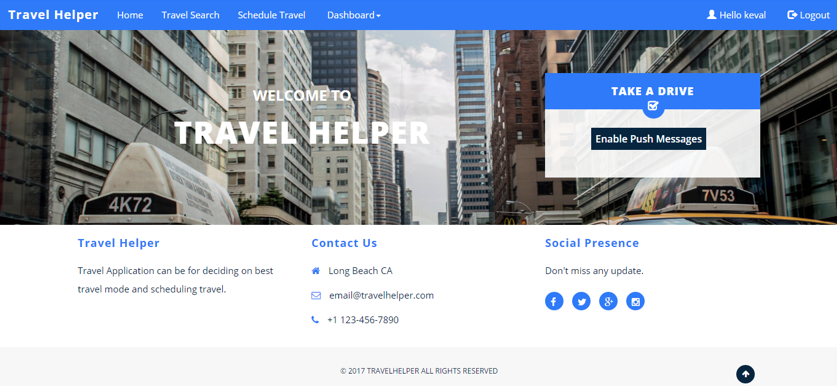
The user enters the source location and the destination location. The user can select economical or faster travel preference based on the need of the travel. The traveller count filters the results based on the capacity of the travel mode. Figure 13 shows the results returned by the search. The result provided by Travel Search presents a unified view of the various travel modes available to the user. The results are gathered by integration of Travel Helper application with REST APIs of Google, Uber, and Lyft. Each travel mode has details about the distance in miles, duration in minutes, estimated time required to start travel, cost in dollars and capacity of the travel mode. These comprehensive details of the various travel modes are used to make efficient travel decision. The decided mode is saved in the system by selecting the record of the travel mode.



**FIGURE 13. Travel Search result.**

**Schedule Travel**

Schedule Travel module is used to save future travel plans. The push notification is sent to user at the travel start time based on travel duration to reach destination on time. The user has to enable notification for getting notified by the push notification. Figure 14 shows the page where users can enable notifications using “Enable Push Messages” button. Once the push notification is enabled, users can schedule the future travel and store the same in the application for notification.



**FIGURE 14. Enable Push Notification page.**

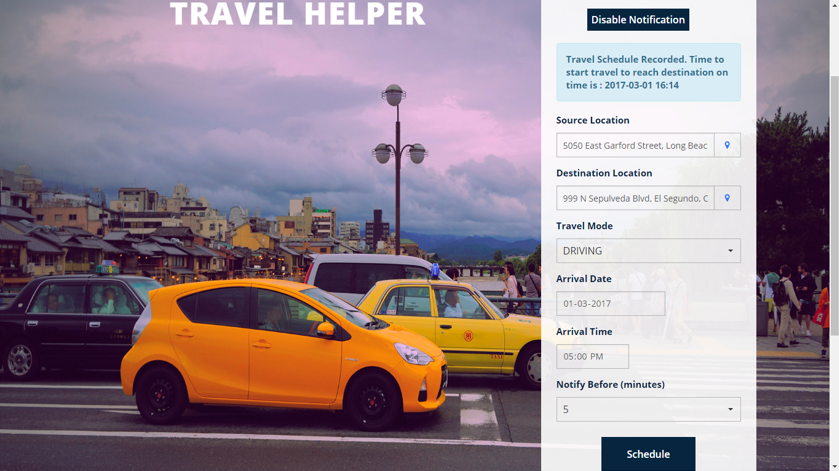
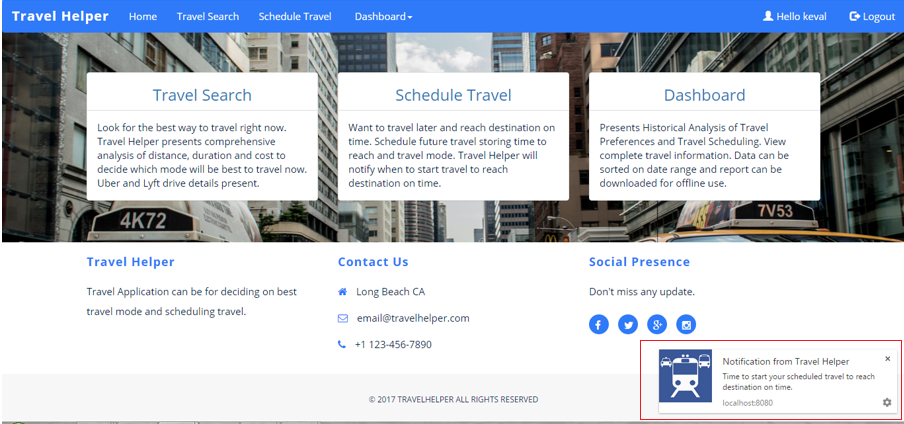
**FIGURE 15. Schedule Travel page.**

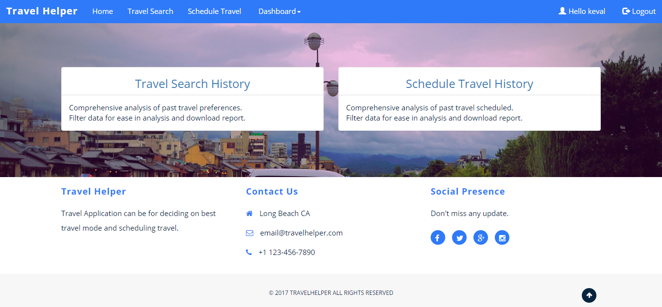
Figure 15 shows the Schedule Travel page. The user enters source location, destination location, preferred travel mode, and arrival date and time at which user has to reach at the destination. The Notify Before field is used to indicate the amount of time before the actual travel start time the user wants to get notified. Notifications can be disabled anytime by clicking on “Disable Notification” button. Once all the required details are entered a confirmation message will be displayed to user. Figure 16 shows the push notification message that is sent to the user.



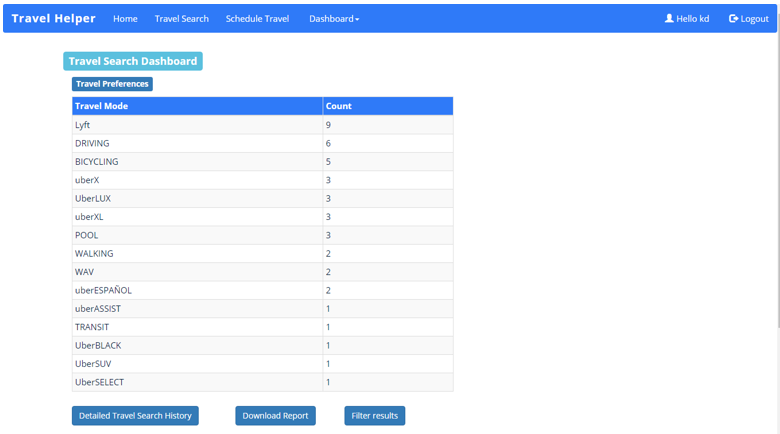
**FIGURE 16. Push Notification message.**

**Dashboard**

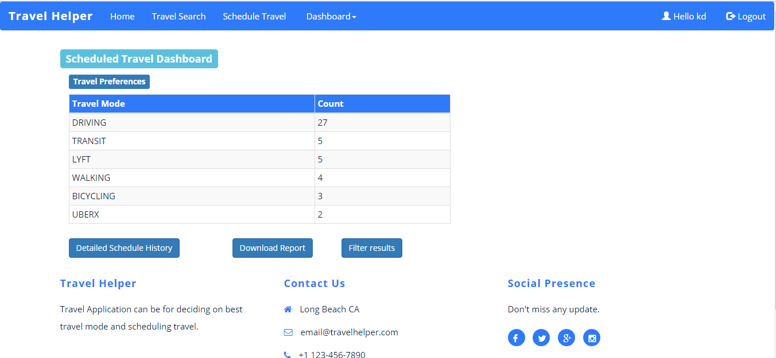
Dashboard provides historical information of Travel Search and Schedule Travel. Figure 17 shows the home of Dashboard page. The Travel Search Dashboard and Schedule Travel Dashboard page shows the user’s preferred travel mode based on the count of travel mode in past by the user. Both the pages contain option to view detailed historical information, filter data based on date range and download report for offline analysis. Figure 18 and Figure 19 shows the Travel Search Dashboard page and Schedule Travel Dashboard page respectively.



**FIGURE 17. Dashboard home page.**

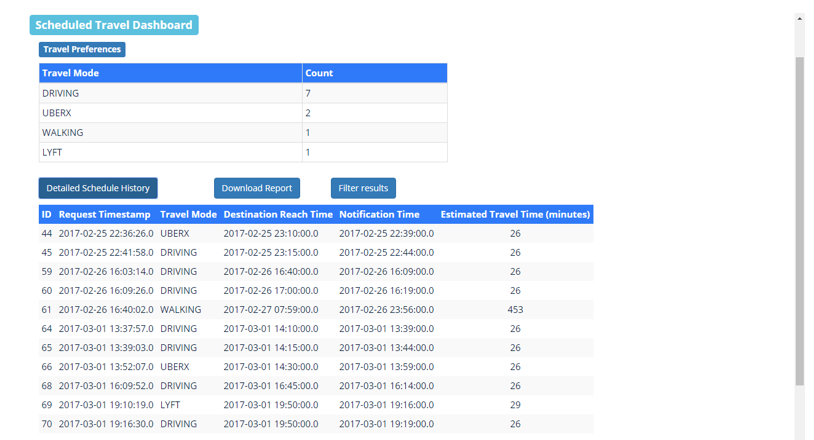


**FIGURE 18.** **Travel Search Dashboard page.**

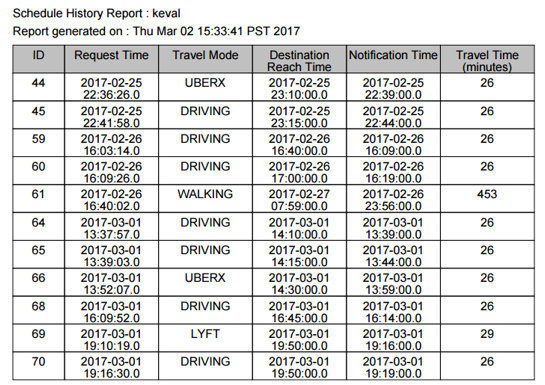


**FIGURE 19.** **Schedule Travel Dashboard page.**

Figure 20 shows the detailed historical records of Schedule Travel History obtained by clicking on “Detailed Schedule History” button. Similar historical records are also available for Travel Search History.

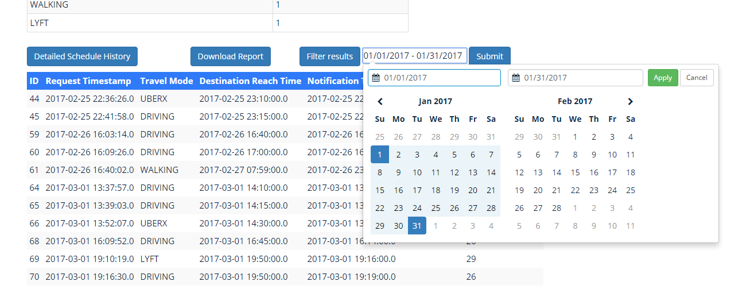


**FIGURE 20. Schedule Travel Detailed History.**



**FIGURE 21.** **Schedule Travel History downloaded report.**

Figure 21 shows the Schedule Travel History downloaded report in PDF format. The report contains past records of travel scheduled in the application for notification. Similar report is also available for Travel Search History. Figure 22 shows the filter results option for Schedule Travel History. Similar filter is also available for Travel Search History.



**FIGURE 22. Filter results option for Schedule Travel History.**

The user can logout of the account by clicking on Logout link in the header of web page. On click of logout user will be redirected to login page.

**CHAPTER 6**

**CONCLUSION**

Software engineering architecture and design patterns provide abstract ways and approaches to be used while developing software applications. Software architecture such as MVC incorporate object oriented concepts with proven solutions to common software development problems in design, development, and maintenance. MVC architecture is widely used in web applications because it facilitates reusable solutions and object-oriented models to assist in structured application development. MVC architecture resolves the problem of strong dependency between web application modules such as user interface, controller, business logic, and data persistence.

Travel Helper is a web application developed successfully using MVC architecture and Spring MVC framework. Travel Helper comprises of four major features: User Profiling, Travel Search, Schedule Travel and Dashboard. User profiling enables users to maintain an account with the application. Travel Search helps in deciding what will be the best way to reach the destination from available travel modes such as walking, bicycling, driving, public transit, Uber, and Lyft. The Application presents a comprehensive detail of distance, duration, and cost of each travel mode in a single view. Another feature is of scheduling future travel plans where user can save the destination and time to reach the place, and the application will notify the user at the time of travel start time to reach destination on time. The Dashboard allows the user to check a preferred travel mode that was used in the past and to search travel history.

The report started with introducing software development using object oriented analysis and design methodology. Requirement Analysis and Design are a critical part of the software development lifecycle because they laid a strong foundation of software development. UML diagrams help in drawing a blueprint of the software that can be used for development of the application. Further chapters explain the use of different technologies used to develop the Travel Helper application. Spring MVC Framework is used to develop application as it is based on MVC architecture. User profiling makes use of Spring security as it provides features to handle authentication and authorization along with security features. Database access is performed using Object-relational mapping (ORM) technology since promotes data persistence using objects. Hibernate is an ORM library used in the application for data query and persistence. Spring MVC with Hibernate eases the web application development with providing better security, maintainability, scalability, and improve development speed. Travel Helper is integrated with other applications such as Google, Uber, and Lyft using REST API for accessing travel related data and push based notifications. REST API offers loose coupling and platform independence that helps to achieve easy integration with MVC architecture. Therefore, Travel Helper allows users to make efficient decision for travelling based on personal preferences and time constraints.

**Future Work**

The future work on this project is to provide more personalization and travel recommendation using data mining and analytics of travel preference data stored by users over time that will assist users to make travel decisions with great ease.

**APPENDICES**

**APPENDIX A**

**NON-FUNCTIONAL REQUIREMENTS**

|  |
| --- |
| 1. Usability |
| * Application should be Americans with Disabilities Act (ADA) compliant. |
| * First time users should be able to easily navigate through application and use all functionalities. |
| * Error messages should be correct and assist user in correcting required inputs |
| * 30 users, 10 novice to computing (0-6 months experience), 10 intermediate (7 months - 2 years’ experience) and 10 experts (2+ years’ experience) are brought into the lab and all of them are introduced to the system. They are provided with certain tasks to perform with the system and their time to successfully perform the task is measured. It should be possible that each user requires less than 5 minutes to perform the task |

|  |
| --- |
| 1. Interoperability |
| * Capability to exchange data via common set of exchange format such as JSON, XML. |
| * Capability to send request and response using common protocols such as HTTP, SOAP and REST. |

|  |
| --- |
| 1. Maintainability |
| * Once the system is set to operate, it becomes necessary to ensure that all the major defects which can have impact on the operation should be identified and resolved within expected time  |  |  | | --- | --- | | Defect Severity | Maximum time for resolution | | High | 3 – 5 days | | Medium | 24 hours | | Low | 4 hours | |
| * Server start-up delay time should be less than or equal to 5 minutes. |
| * Application Code should be modularised with LOC less than 500 lines in a single file |

|  |
| --- |
| 1. Portability |
| * Application should run on all major Operating system such as Windows, Linux, UNIX and IOS. |
| * Application should function as expected in Desktop computer, Tablet and Mobile of all display size |

|  |
| --- |
| 1. Instalability |
| * New installation of complete application architecture should be done in 3 days which involves database setup, data migration, application/webserver server setup, code deployment. |

|  |
| --- |
| 1. Performance |
| * Load time of web pages should be less than or equal to 4 seconds. |
| * Application should be handle to manage load of 100 users at a particular moment. |
| * Response time of any function should not exceed 2 minutes |

|  |
| --- |
| 1. Security |
| * Application should maintain Confidentiality, Integrity and Authentication assurance to users. |

|  |
| --- |
| 1. Documentation |
| * User manual and system specifications documentations will be provided to application maintenance team. |
| * 20 hours of training will be provided to the team members. |

|  |
| --- |
| 1. Scalability |
| * The scalable system should be able to handle 30% more users with addition of each server into the existing architecture. |
| * Using parallel programming and multiple CPU's will decrease response time by 20%. |

**APPENDIX B**

**USE CASE MODELS**

**TABLE 1. New User Registration Use Case**

|  |  |
| --- | --- |
| Use Case Name: | New user registration |
| Actors: | User |
| Pre Conditions: | 1. The application is running.  2. Database connection is established. |
| Flow of Control: | 1. User clicks on “Login to explore” link on the application landing page.  2. System redirects to login screen.  3. User clicks on “Sign Up Here” link.  4. User enters appropriate details in the sign up form.  5. User clicks on “Sign Up” button to submits the form.  6. Appropriate message is shown to the user. |
| Post Conditions: | 1. New user account is created.  2. The user information is stored database. |
| Error Conditions: | 1. Invalid characters or more than maximum characters allowed – system prompts to correct error inputs.  2. Fields left incomplete at time user submits form - system prompts user to complete input fields. |
| Non-Functional Requirements: | 1. Availability: The web application must be available 95% of time in a span of 24 hours.  2. Performance: The page load time should be less than or equal to 4 sec. Application should be capable to handle minimum load of 100 users at a particular moment.  3. Security: Maximum failed login attempt allowed must be 5.  4. Compatibility: The user interface should be rendered as expected in browsers such as Internet Explorer (above version 8), Mozilla Firefox, Chrome and Safari |

**TABLE 2. User Login Use Case**

|  |  |
| --- | --- |
| Use Case Name: | User login |
| Actors: | User |
| Pre Conditions: | 1. The application is running.  2. Database connection is established.  3. User account already exists. |
| Flow of Control: | 1. User clicks on “Login to explore” link on the application landing page.  2. System redirects to login screen.  3. User enters username and password to login.  4. User clicks on “Login” button to submit the form. |
| Post Conditions: | 1. User is successfully logged in to the system and redirected to homepage.  2. User can access account details, search travel option and schedule travel. |
| Error Conditions: | 1. Incorrect login credentials.  2. Error in database connection. |
| Non-Functional Requirements: | 1. Availability: The web application must be available 95% of time in a span of 24 hours.  2. Security: Maximum failed login attempt allowed must be 5.  3. Compatibility: The user interface should be rendered as expected in browsers such as Internet Explorer (above version 8), Mozilla Firefox, Chrome and Safari |

**TABLE 3. Search Economic Travel Option Use Case**

|  |  |
| --- | --- |
| Use Case Name: | Search economic travel option |
| Actors: | User |
| Pre Conditions: | 1. The application is running.  2. Database connection is established.  3. User account already exists.  4. REST API are accessible over Internet. |
| Flow of Control: | 1. User visits homepage and clicks on “Travel Search” link.  2.User enters source, destination, and travel preferences as “Economical”.  3. User clicks on “Search Results” button to submit form.  4. System calls REST API to fetch travel details from Google, Uber, and Lyft.  5. System shows various details such as distance, duration, and cost of all available travel mode and sort results based on cost.  6. User selects the travel option and saves travel preference. |
| Post Conditions: | User travel preference is saved into the database. |
| Error Conditions: | 1. Invalid characters or more than max characters allowed – system prompts to correct error inputs.  2. Fields left incomplete at time user submits form - system prompts user to complete input fields. |
| Non-Functional Requirements: | 1. Interoperability: The web application should be able to connect and request/respond data to Travel API using REST API  2. Availability: Connectivity with Uber, Lyft, LB Transit and Metro should be available for 90%. |

**TABLE 4. Search Faster Travel Option Use Case**

|  |  |
| --- | --- |
| Use Case Name: | Search faster travel option |
| Actors: | User |
| Pre Conditions: | 1. The application is running.  2. Database connection is established.  3. User account already exists.  4. REST API are accessible over Internet. |
| Flow of Control: | 1. User visits homepage and clicks on “Travel Search” link.  2.User enters source, destination, and travel preferences as “Fastest”.  3. User clicks on “Search Results” button to submit form.  4. System calls REST API to fetch travel details from Google, Uber, and Lyft.  5. System shows various details such as distance, duration, and cost of all available travel mode and sort results based on duration.  6. User selects the travel option and saves travel preference. |
| Post Conditions: | New customer travel entry is saved into database. |
| Error Conditions: | 1. Invalid characters or more than max characters allowed – system prompts to correct error inputs.  2. Fields left incomplete at time user submits form - system prompts user to complete input fields. |
| Non-Functional Requirements: | 1. Interoperability: The web application should be able to connect and request/respond data to Travel API using REST API  2. Availability: Connectivity with Uber, Lyft, LB Transit and Metro should be available for 90%. |

**TABLE 5. Save travel decision use case**

|  |  |
| --- | --- |
| Use Case Name: | Save travel decision |
| Actors: | User |
| Pre Conditions: | 1. The application is running.  2. Database connection is established.  3. User account already exists. |
| Flow of Control: | 1. User clicks on travel option on the table row to save the preference selected.  2. Application saves the travel preference. |
| Post Conditions: | Details regarding travel preference are stored in database. |
| Error Conditions: | Error in database connection. |
| Non-Functional Requirements: | Usability: See Non Functional Requirement Section 1 |

**TABLE 6. Schedule Future Travel for Notification Use Case**

|  |  |
| --- | --- |
| Use Case Name: | Schedule future travel for notification |
| Actors: | User |
| Pre Conditions: | 1. The application is running.  2. Database connection is established.  3. User account already exists.  4. REST API are accessible over Internet. |
| Flow of Control: | 1. User clicks on ‘Schedule travel’ link on the homepage.  2. User click on “Enable notification” link.  3. System acknowledges and presents travel schedule form to the user.  4. User enters future travel schedule details.  5. User click on “Schedule” button.  6. System saves the details in database. |
| Post Conditions: | 1. User device identifier is registered in the database.  2. Travel details are stored. |
| Error Conditions: | 1. Invalid characters or more than max characters allowed – system prompts to correct error inputs.  2. Fields left incomplete at time user submits form - system prompts user to complete input fields. |
| Non-Functional Requirements: | Performance: Response time to complete the save should be less than 1 minute. |

**TABLE 7. Send Push Notification Use Case**

|  |  |
| --- | --- |
| Use Case Name: | System sends push notification about travel start time |
| Actors: | System |
| Pre Conditions: | 1. The application is running.  2. Database connection established.  3. REST API is accessible.  4. Google cloud connectivity is established.  5. User machine is connected to Internet.  6. Travel Helper scheduler is running. |
| Flow of Control: | 1. Travel Helper sends push notification to user regarding start of travel to reach destination in time.  2. User checks push notification message.  2.1 Based on customer preferences, go to use case 2 or 3. |
| Post Conditions: | See Use case 2 or 3 |
| Error Conditions: | NA |
| Non-Functional Requirements: | 1. Interoperability: Data exchange with google cloud messaging API should be done using REST.  2. Availability: Connectivity with google cloud should be established 98% of time. |

**TABLE 8. View Travel Search History Use Case**

|  |  |
| --- | --- |
| Use Case Name: | View travel search history |
| Actors: | User |
| Pre Conditions: | 1. The application is running.  2. Database connection is established.  3. User account already exists. |
| Flow of Control: | 1. User clicks on ‘Dashboard’ link on the homepage.  2. User clicks on ‘Travel Search History’ link.  3. System queries database to fetch details.  4. Travel summary is displayed to user with travel preferences.  5. User clicks on “Click for Details Travel History” link to view complete history. |
| Post Conditions: | NA |
| Error Conditions: | NA |
| Non-Functional Requirements: | Usability: See Non Functional Requirement Section 1 |

**TABLE 9. View Travel Schedule History Use Case**

|  |  |
| --- | --- |
| Use Case Name: | View travel schedule history |
| Actors: | User |
| Pre Conditions: | 1. The application is running.  2. Database connection is established.  3. User account already exists. |
| Flow of Control: | 1. User clicks on ‘Dashboard’ link on the homepage.  2. User clicks on ‘Travel Schedule History’ link.  3. System queries database for fetching details.  4. Travel summary is displayed to customer with travel preferences.  5. User clicks on “Click for Details Travel History” link to view complete history. |
| Post Conditions: | NA |
| Error Conditions: | NA |
| Non-Functional Requirements: | Usability: See Non Functional Requirement Section 1 |

**TABLE 10. Filter Report Data on Date Range Use Case**

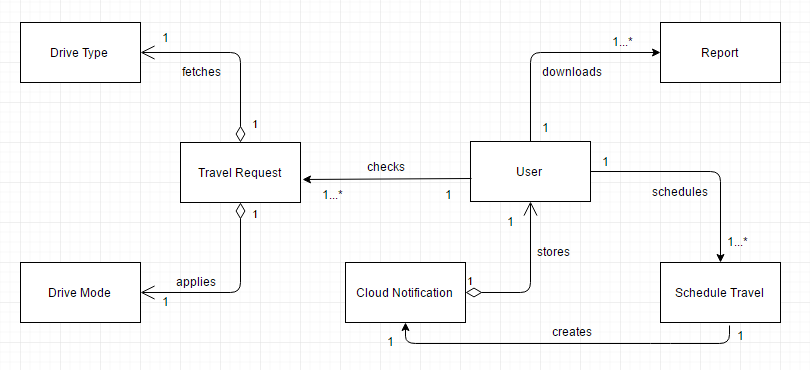
|  |  |
| --- | --- |
| Use Case Name: | Filter report data on date range |
| Actors: | User |
| Pre Conditions: | 1. The application is running.  2. Database connection is established.  3. User account already exists. |
| Flow of Control: | 1. User clicks on ‘Filter Results’ button on the dashboard page.  2. System queries the database to fetch filtered records.  3. Travel details are displayed to customer. |
| Post Conditions: | NA |
| Error Conditions: | NA |
| Non-Functional Requirements: | Usability: See Non Functional Requirement Section 1 |

**TABLE 11. Download Travel History Report Use Case**

|  |  |
| --- | --- |
| Use Case Name: | Download travel history report |
| Actors: | User |
| Pre Conditions: | 1. The application is running.  2. Database connection is established.  3. User account already exists. |
| Flow of Control: | 1. User clicks on ‘Download Report’ button on the dashboard page.  2. System generates pdf format report.  3. User saves the report. |
| Post Conditions: | NA |
| Error Conditions: | NA |
| Non-Functional Requirements: | 1. Accessibility: Document should be accessible within 5 minutes.  2. Usability: See Non Functional Requirement Section 1 |

**APPENDIX C**

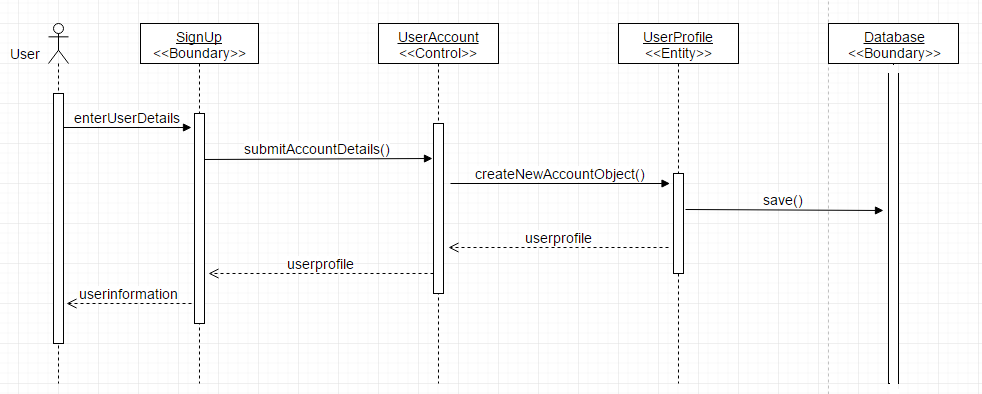
**CLASS DIAGRAM**

****

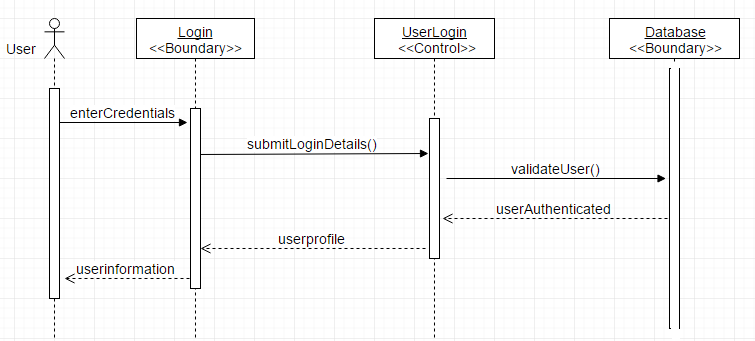
**FIGURE 23. Travel Helper class diagram.**

**APPENDIX D**

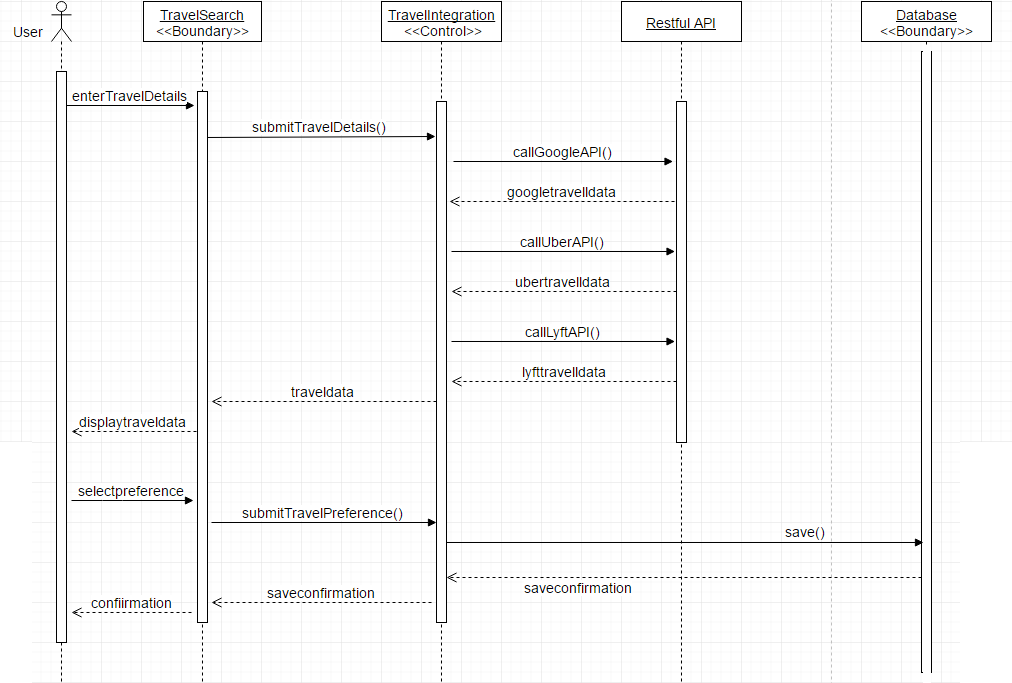
**SEQUENCE DIAGRAMS**

****

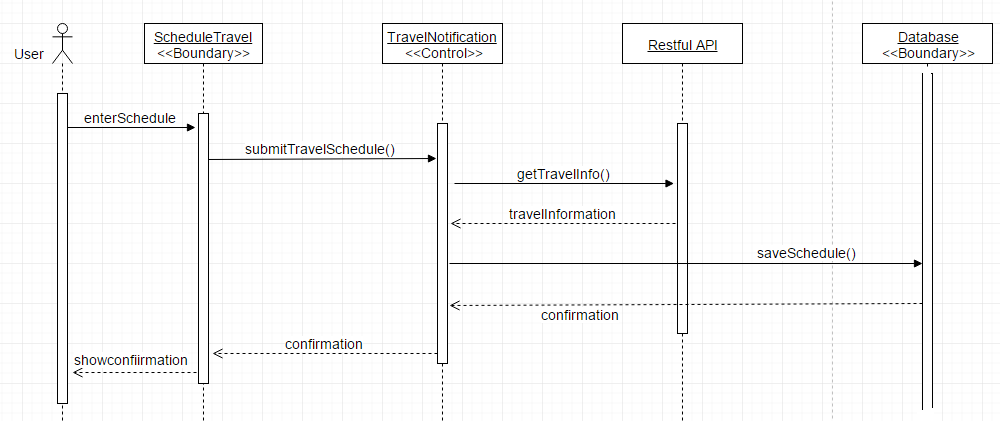
**FIGURE 24. Sequence diagram for new account creation.**

****

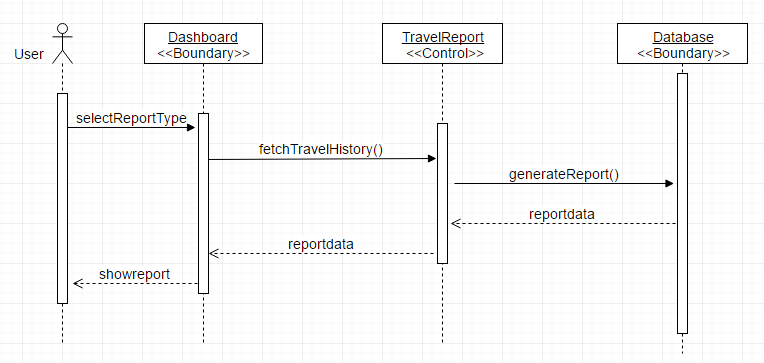
**FIGURE 25. Sequence diagram for User Login.**

****

**FIGURE 26. Sequence diagram for efficient travel search.**

****

**FIGURE 27. Sequence diagram for scheduling future travel plan.**

****

**FIGURE 28. Sequence diagram for display travel history.**

**APPENDIX E**

**DATABASE TABLES**

**USER\_PROFILE**

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Datatype** | **Remark** |
| USER\_ID | Integer (20) | Primary key |
| FIRST\_NAME | Varchar (20) |  |
| LAST\_NAME | Varchar (20) |  |
| ADDR\_STREETNAME | Varchar (20) |  |
| ADDR\_APTNO | Varchar (20) |  |
| ADDR\_CITY | Varchar (20) |  |
| ADDR\_STATE | Varchar (20) |  |
| ADDR\_ZIP | Varchar (20) |  |
| CONTACT\_NUMBER | Varchar (20) |  |
| EMAIL | Varchar (50) |  |
| Username | Varchar (50) |  |
| PASSWORD | Varchar (100) |  |
| Role\_id | Int(10) |  |
| Active | Int(5) |  |

**TRAVEL\_REQUEST**

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Datatype** | **Remark** |
| TRAVEL\_REQUEST\_ID | Integer (20) | Primary Key |
| USER\_ID | Integer (20) | Foreign Key |
| START\_POINT\_X\_COORDINATE | Varchar(20) |  |
| START\_POINT\_Y\_COORDINATE | Varchar(20) |  |
| END\_POINT\_X\_COORDINATE | Varchar(20) |  |
| END\_POINT\_Y\_COORDINATE | Varchar(20) |  |
| REQUEST\_TIMESTAMP | Datetime |  |
| TRAVEL\_DISTANCE\_IN\_MILES | Integer (10) |  |
| TRAVEL\_MODE\_SELECTED | tinyint (3) |  |
| TRAVEL\_DRIVE\_SELECTED | tinyint (3) |  |
| TRAVEL\_STATUS | Varchar (20) |  |
| TRAVEL\_DURATION | Integer(20) |  |
| user\_selected\_drive | Varchar(20) |  |
| Cost | Integer(10) |  |

**FUTURE\_SCHEDULED\_TRAVEL**

|  |  |  |  |
| --- | --- | --- | --- |
| **Column Name** | **Datatype** | | **Remark** |
| RECORD\_ID | Int(5) Autoincrement | | Primary Key |
| USER\_ID | Integer (20) | Foreign Key | |
| START\_POINT\_X\_COORDINATE | Varchar(20) |  | |
| START\_POINT\_Y\_COORDINATE | Varchar(20) |  | |
| END\_POINT\_X\_COORDINATE | Varchar(20) |  | |
| END\_POINT\_X\_COORDINATE | Varchar(20) |  | |
| PRE\_NOTIFICATION\_TIME\_IN\_MINUTES | Integer (20) |  | |
| DRIVE\_ID | tinyint (3) |  | |
| NOTIFICATION\_TIME | datetime |  | |
| destination\_reach\_time | datetime |  | |
| traveltime\_expected | Int (20) |  | |
| request\_savetime | datetime |  | |
| Drive\_selected | Varchar(20) |  | |

**USER\_CLOUD\_NOTIFICATION\_ID**

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Datatype** | **Remark** |
| USER\_ID | Integer (5) | Primary Key , Foreign Key |
| GCM\_REGISTERATION\_ID | Varchar (250) | Primary Key |
| ACTIVE | tinyint (3) |  |

**TRAVEL\_DRIVE**

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Datatype** | **Remark** |
| DRIVE\_ID | tinyint (3) | Primary Key |
| DRIVE\_NAME | Varchar (20) | Unique |

**TRAVEL\_MODE**

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Datatype** | **Remark** |
| TRAVEL\_MODE\_ID | tinyint (3) | Primary Key |
| TRAVEL\_MODE\_NAME | Varchar (20) | Unique |

**REFERENCES**

**REFERENCES**

[1] J.O. Coplien, "Software Design Patterns," *Encyclopedia of Computer Science*, Wiley, 2003, pp. 1604-1606.

[2] S. R. Kumar, R. Sharma, and K. Gupta, "Strategies for Web Application Development Methodologies," *Proc.* *Intl. Conf. on Computing, Communication and Automation* (ICCCA2016), 2016, pp. 160-165.

[3] D. M. Selfa, M. Carrillo, and M. Del Rocio Boone, "A Database and Web Application Based on MVC Architecture," *Proc.* *16th* *IEEE Intl. Conf. on Electronics, Communications and Computers*, 2006, pp. 48-48.

[4] Y. Ping, K. Kontogiannis, and T. C. Lau, "Transforming Legacy Web applications to the MVC Architecture," *Proc.* *11th Annual* *Intl. Workshop on Software Technology and Engineering Practice* (STEP’04), 2003, pp. 133-142.

[5] J. Arthur, and S. Azadegan, "Spring Framework for Rapid Open Source J2EE Web Application Development: A Case Study," *Proc.* *6th Intl. Conf. on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing and First ACIS International Workshop on Self-Assembling Wireless Networks* (SNPD/SAWN’05)*,* 2005, pp. 90-95.

[6] D. Kung and J. Lei. "An Object-Oriented Analysis and Design Environment," *Proc.* *29th Intl. Conf. on Software Engineering Education and Training*, 2016, pp. 91-100.

[7] L. Richardson and S. Ruby, “The Resource-Oriented Architecture,” *RESTful Web Services*, O'Reilly Media, Inc., 2008, pp. 79-105.

[8] X. Franch and P. Botella, "Putting Non-functional Requirements into Software Architecture," *Proc.* *9th Intl. Workshop on Software Specification and Design*, 1998, p. 60.

[9] Apple Inc, “Model-View-Controller,” Oct.2015; https://developer.apple.com/library

/content/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html.

[10] R. Johnson, J. Hoeller, K. Donald, C. Sampaleanu, R. Harrop, T. Risberg, and A. Arendsen et al, "The Spring Framework Reference Documentation," presented at Interface 21, 2004.

[11] Spring, “Web MVC Framework,” n.d; https://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html

[12] R. C. Martin, “The Dependency Inversion Principle,” *C++ Report*, vol.8, 1996, pp. 61–66.

[13] Spring, “Transaction Management,” n.d; http://docs.spring.io/spring-framework/docs/4.2.x/

spring-framework-reference/html/transaction.html

[14] Spring, “Spring Security,” n.d; http://docs.spring.io/spring-security/site/docs/3.0.x/ reference/introduction.html

[15] W. HaiTao and J. BaoXian, "Research Based on Web Development of Spring Integration Framework," *Proc.* *Intl. Forum on Information Technology and Applications*, 2010, pp. 325-328.

[16] Mozilla Developer Network, “Promise,” 2017; https://developer.mozilla.org/en-US/docs/ Web/JavaScript/Reference/Global\_Objects/Promise

[17] Firebase, “Firebase Cloud Messaging,” Feb. 2017; https://firebase.google.com/docs/cloud- messaging/

[18] Google, “Google Maps APIs,” Jan. 2017; https://developers.google.com/maps/

[19] Uber Developers, “Riders API Guide,” 2017; https://developer.uber.com/docs/riders/ references/api

[20] Lyft, “Ride Request API,” 2017; https://developer.lyft.com/v1/reference