**WEB APPLICATION FOR EFFICIENT TRAVEL DECISION MAKING USING MVC ARCHITECTURE**

A PROJECT REPORT

Presented to the Department of 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.

College Designee:

Antonella. Sciortino, Ph.D.

By Keval Soni

B.E, 2011, University of Mumbai, India

May 2017

ABSTRACT

**WEB APPLICATION FOR EFFICIENT TRAVEL DECISION MAKING USING MVC ARCHITECTURE**

By

Keval Soni

May 2017

Software Engineering is an extremely dynamic field which can be used to solve many real-world problems. Software engineering principles and design patterns provide abstract ways and approaches to be implemented while developing new software or resolving core engineering problem. Travel Helper is a web application built on Model-View-Controller (MVC) architecture.

In the presented document, Model-View-Controller (MVC) architecture and Spring Framework are explained and their use in developing Travel Helper web application. Travel Helper application assists in deciding best option to travel from source to destination. Another feature of the application allows user to save the future travel and notify user best time to start the travel using Push Notification. Documents explains different components of Spring Framework, Hibernate and REST API used in application development. Document also presents object oriented analysis and design methodology used while application development.

**TABLE OF CONTENTS**

ABSTRACT….………………………………………………………………………………. #

LIST OF TABLES…………………………………………………………………………… #

LIST OF FIGURES…………………………………….…………………………………..... #

LIST OF ABBREVIATIONS……………………………………………………………….. #

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

2. SYSTEM ANALYSIS……………………………………………………………. #

3. CONCEPT BREAKDOWN……………………………………………………… #

4. METHODOLOGY……………………………………………………………….. #

5. RESULTANT SYSTEM…………………………………………………………. #

6. CONCLUSION…………………………………………………………………… #

APPENDIX: USE CASES

REFERENCES

**LIST OF TABLES**

1. New Customer Registration………………………………………………………… #
2. Customer Login……………………………………………………………………... #
3. Search economic travel option……………………………………………………… #
4. Search faster travel option…………………………………………………………... #
5. Customer saves travel decision……………………………………………………… #
6. Schedule future travel for notification………………………………………………. #
7. System prompts user about travel start time………………………………………… #
8. Customer views travel dashboard…………………………………………………… #
9. Filter report data on date range……………………………………………………… #
10. Download travel history report……………………………………………………… #

**LIST OF FIGURES**

Figure 1: MVC Architecture………………………………………………………………… #

Figure 2: MVC Flow Diagram……………………………………………………………… #

Figure 3: Spring MVC Architecture…………………………………………………………#

Figure 4: Hibernate Architecture…………………………………………………………… #

Figure 5: Push notification request response messages……………………………………. #

Figure 6: Firebase Cloud Messaging Overview…………………………………………… #

Figure 7. Landing page……………………………………………………………………. #

Figure 8: Login Page……………………………………………………………………… #

Figure 9: Authentication error page………………………………………………………. #

Figure 10: Signup Form…………………………………………………………………. #

Figure 11: Home Page of the application………………………………………………… #

Figure 12: Travel Mode Search Page……………………………………………………. #

Figure 13: Result of travel search showing available travel information…………………#

Figure 14: Schedule Travel Page with Enable Notification screen……………………… #

Figure 15: Form to enter details for scheduling future travel

Figure 16: Confirmation message on saving travel information for notification

Figure 17: Push notification from application indicating start of scheduled travel

**CHAPTER 1**

**INTRODUCTION**

The evolution of the computers in the last decade has been absolutely amazing. The pace at which the power and size of the computers have changed is immense. The concept of communicating information between computers using computer networks and protocols led to rise of World Wide Web famously known as Internet. Internet has grown at an enormous speed in a short span of time. Today, majority of the world population is connected by the Internet. Growth of internet has been popular because it is full of services: search engines,

online stores, weblogs, wikis, calculators, and games. Rather than installing all this data

and all these programs on your own computer, you install one program—a web browser

—and access the data and services through it.

Web based application are Internet based software which can be accessed using web browser. Web applications are developed based on client-server model where the application is deployed on server and it can be accessed using client like web browser. Web applications became immensely popular and have taken over the traditional desktop software applications. The reason is the simplicity and ease in application development, accessible anywhere, platform independence and ease in support and maintenance.

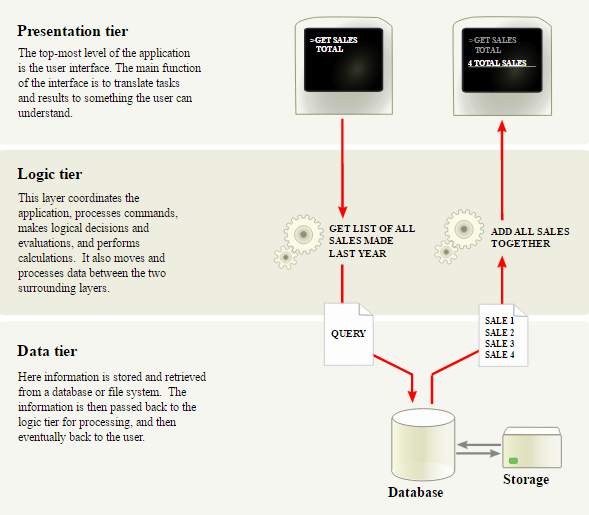
Today, every business and non-profit organization owns a web site used for business or informational purpose. It is easy to conduct business online in term of online shopping or trading etc. Web applications can be customized and designed with respect to business requirements. Therefore, web applications are services available to users over the Internet.

Web Applications can be of 2-tier or 3-tier architecture. 2-tier architecture consists of presentation and logic tier. 3-tier application consists of presentation, logic and data tier.

Presentation tier – This is the client facing layer consisting of user interface. User Interface displays information to users for reading and perform user actions on interface for future action.

Logic tier – Application logic is deployed on this tier. It acts as a coordinator between presentation and data tier. Requests from presentation layer are processed, perform computations based on user requests and interacts with database to store or retrieve data. Response is forwarded back to user interface.

Data tier – This tier consists of a data store containing database management system for storing data. The information is stored and retrieved from database. The processed information is passed back to logic tier and result is displayed to user interface.



**Figure 1: MVC Architecture**

Model View Controller or MVC as it is known popularly, is a software design pattern for developing web applications. A Model View Controller pattern is made up of the following three parts:

Model - The lowest level of the pattern which is responsible for maintaining data.

View – This is the client facing user interface responsible for displaying information to user.

Controller – It acts as an interface between model and view.

Spring is a Java based framework containing Spring MVC module which provides model-view-controller architecture that can be used for development of web applications.

Travel Helper is a web application developed using Spring MVC framework. Various modules of travel helper are user profiling, efficient travel decision making and schedule future travel. User can do analysis of past travel history using dashboard. Travel Helper is a web application that will demonstrate the benefits of developing web application using Spring MVC architecture. Travel Helper also showcase the ease of implementation and the power of REST API. This document also reflects upon the development of software using Object Oriented Analysis and Design methodology.

**CHAPTER 2**

**SYSTEM ANALYSIS**

**2.1 Goal**

The application should enable the user to select best and preferred travel option available. This will relieve users from doing multiple search across multiple applications to find cheapest or fastest travel option available. Features of pre-scheduling travel and real-time traffic congestion information will help users in saving time and money.

**2.2 Object Oriented Analysis and Design**

Object–Oriented Analysis and Design (OOAD) is the procedure of identifying software engineering requirements and developing software specifications in terms of a software system’s object model, which comprises of interacting objects. It is a popular technical approach for analysing, designing an application, system, or business by applying the object-oriented paradigm and visual modelling throughout the development life cycles to foster better stakeholder communication and product quality. The development of Travel Helper was done following the life cycle proposed by this model.

First step in any software development lifecycle is to gather requirement from stakeholders or describe problem statement of software to be built. The output of requirement phase of software engineering is Software Requirement Specification (SRS) also known as Requirement Analysis Document (RAD). This document lays a foundation for software engineering activities. It has complete details of functional and non-functional requirements of the software. It also includes Unified Modeling Language (UML) diagrams like use cases, sequence and class diagram to describe interaction between the user and software system.

UML is a graphical language for OOAD that gives a standard way to write a software system’s blueprint. It helps to visualize, specify, construct, and document the artifacts of an object-oriented system. It is used to depict the structures and the relationships in a complex system.

Below sections describes the how requirements are gathered and analysed in RAD document. The explanation is supported with requirement analysis for Travel Helper application.

2.2.1 **Functional and Non Functional Requirement**

In software engineering, Functional requirements specify the functionality of various modules of an application. Functional requirements capture the working requirements of the software and plan the way software needs to function for user. Functional requirements specify what the system must do. Non-functional requirements describe expected behaviour of the functionality specified for the systems. It specifies design constraints and characteristics that need to be taken care of to satisfy non-functional requirements. System design specifies plan to implement functional requirements whereas System architecture specifies plan to implement non-functional requirement. Functional requirements can be explained using use cases and non-functional are listed in a table. Functional Requirements for Travel Helper are explained using Use Cases. Non-Functional Requirements are listed in Appendix A.

2.2.2 **Use Case**

Use case is a behavioural diagram. It depicts the sequence of actions a user of a system can perform to achieve functionality of system. 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 functional requirement of the system.

Below are the various section of use case:

1. Use Case Name: Describes the name of the action.
2. Actors: User 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.
4. Flow of Control: It lists the actions that needs to be performed for successful execution of use case.
5. Post Conditions: Conditions that should be executed successfully after the actor has completed actions 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 execution of the use case.

Below list contains brief description of the use case designed to describe various functionalities for the Travel Helper application. All use cases have been included in Appendix A.

1. New User registration: This use case depicts the actions need performed by user to register new profile in application.
2. User Login: User can login to application by following steps presented in Use Case Table 2.
3. Search economical travel option: Users can find travel option to reach to destination economically by following steps listed in Use Case Table 3
4. Search faster travel option: Users can find travel option to reach to destination economically by following steps listed in Use Case Table 4
5. Customer saves travel decision: User can save the travel option being preferred in system by following steps mentioned in Use Case Table 5.
6. Schedule future travel for notification: User follows steps mentioned in Use Case Table 6 to save future travel for notification.
7. System prompts user about travel start time: System notifies user to start travel based on Use Case Table 7.
8. Customer views travel dashboard: User can view travel history following steps in Use Case Table 8.
9. Filter report data on date range: User can filter report data using date for analysis following steps in Use Case Table 9.
10. Download travel history report: User can download report by following steps listed in Use Case Table 10.

2.2.3 **Mockups**

Mockups are the design of user interface created using wireframes. Wireframes are design of user interface of application which provide visual guide and act as skeleton of view of web pages. Mockups are created using wireframes. Hypertext Markup Language (HTML) pages, Javascript (JS) and Cascading Style Sheets (CSS) are building blocks of mockups. Therefore mockups lay a great foundation by designing the user interface which can help in development phase.

2.2.4 **Database Schema**

Database Schema represent the logical view of the database. It is the blueprint of how the data will be stored in database. Database Schema consists of tables, attributes in tables and relationship amongst tables. Database schema also present constraints on data that can be inserted in tables. Thus database schema acts as a complete design document to visualize complete flow of data in application.

Database tables for Travel Helper application are included in Appendix A

**CHAPTER 3**

**CONCEPT BREAKDOWN**

**3.1 Web Application**

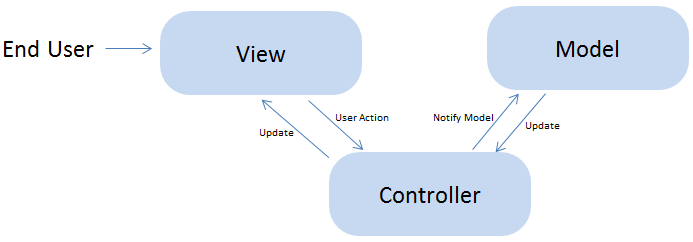
Web applications are developed using client server architecture. The client–server model is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients. Client and Server can reside on same system or may reside in different data-centre are connected using computer network and Internet. Server can host single or multiple instances of the application and share resources with client. Client makes connection with application to fetch content and display result to the user. World wide web, email, network printers are some of the examples of client server model.

Client and server exchange messages using HTTP protocol request response messages. Client send a request to server. Server can be a web server or application server. Role of Web servers is to manage request load by distributing request to application server. Application server hosts the application business logic and is connected to database server. Thus, application server processes client request, performs necessary computation and performs database query operation if required. Response is then sent back to client to display result in browser.

Therefore, client server architecture acts as a base for developing applications with less coupling and more cohesion. Further, application framework try to achieve same within application server level.

**3.2 MVC**

Model View Controller or MVC is a famous software design pattern for developing web application. MVC is popular as it isolates the application logic from the user interface layer and supports separation of concerns. Here the Controller receives all requests for the application and then works with the Model to prepare any data needed by the View. The View then uses the data prepared by the Controller to generate a final presentable response.



**Figure 2: MVC Flow Diagram**

**The model**

The model is responsible for managing the data of the application. It responds to the request from the view and it also responds to instructions from the controller to update itself.

**The view**

A presentation of data in a particular format, triggered by a controller's decision to present the data. They are script based templating systems like JSP, ASP, PHP and very easy to integrate with AJAX technology.

**The controller**

The controller is responsible for responding to user input and perform interactions on the data model objects. The controller receives the input, it validates the input and then performs the business operation that modifies the state of the data model.

**3.3 Spring MVC**

Enterprise application are developed using framework which provide support for MVC. Developing application with MVC architecture helps in loose coupling and high cohesion. The Spring Framework is an application framework and inversion of control container for the Java platform. The framework's core features can be used by any Java application, but there are extensions for building web applications on top of the Java EE platform. Spring provides a POJO (Plain Old Java Object) based configuration environment, a container to manage the instantiation and lifecycle of your POJO components, and a framework to help you put into place some established best practices for your applications. The idea behind Spring is that your code should be well-factored, and components, kept pristine. Your components should run with or without a container, and be testable with minimal to no intrusion from outside classes.

Spring provides its own MVC web application framework known as Spring MVC. The Spring Web model-view-controller (MVC) framework is designed around a DispatcherServlet that dispatches requests to handlers, with configurable handler mappings, view resolution, locale, time zone and theme resolution as well as support for uploading files. The default handler is based on the @Controller and @RequestMapping annotations, offering a wide range of flexible handling methods.

**3.3.1 Dependency Injection**

In software engineering, dependency injection is a design principle in which code creating a new object supplies the other objects that the new object depends on for operation. A dependency is an object that can be used (a service). An injection is the passing of a dependency to a dependent object (a client) that would use it. The service is made part of the client's state. [1] Passing the service to the client, rather than allowing a client to build or find the service, is the fundamental requirement of the pattern.

The intent behind dependency injection is to decouple objects to the extent that no client code has to be changed simply because an object it depends on needs to be changed to a different one. Dependency injection is one of many forms of inversion of control. Rather than low level code calling up to high level code, high level code can receive lower level code that it can call down to. This inverts the typical control pattern seen in procedural programming.

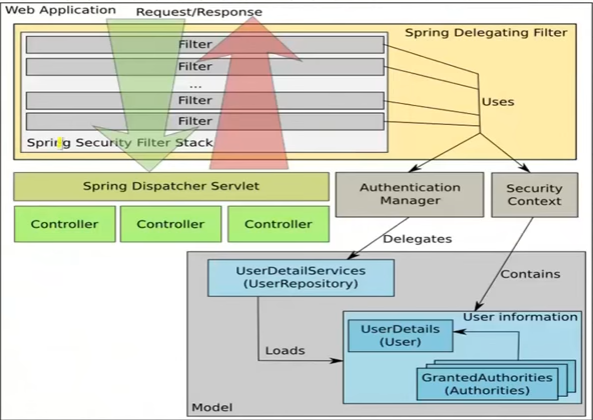
There are three common means for a client to accept a dependency injection: setter-, interface- and constructor-based injection.

**3.3.2 Transaction Management**

A database transaction is a sequence of actions that are treated as a single unit of work. These actions should either complete entirely or take no effect at all. Transaction management is an important part of and RDBMS oriented enterprise applications to ensure data integrity and consistency. Spring framework provides an abstract layer on top of different underlying transaction management APIs.

**3.3.3 Spring Security**

Spring Security provides comprehensive security services for J2EE-based enterprise software applications. Two major areas of application security are “authentication” and “authorization” (or “access-control”). These are the two main areas that Spring Security targets. “Authentication” is the process of establishing a principal is who they claim to be (a “principal” generally means a user, device or some other system which can perform an action in your application). “Authorization” refers to the process of deciding whether a principal is allowed to perform an action within your application. To arrive at the point where an authorization decision is needed, the identity of the principal has already been established by the authentication process.



**Figure 3: Spring MVC Architecture**

In spring security, all the user requests are first captured by Spring Delegating Filter before the request reaches the controller. Authentication and authorization is done before the user is granted access to resources. Irrespective of how authentication was undertaken, Spring Security provides a deep set of authorization capabilities. There are three main areas of interest in respect of authorization, these being authorizing web requests, authorizing whether methods can be invoked, and authorizing access to individual domain object instances.

**3.3.4 Spring Scheduler**

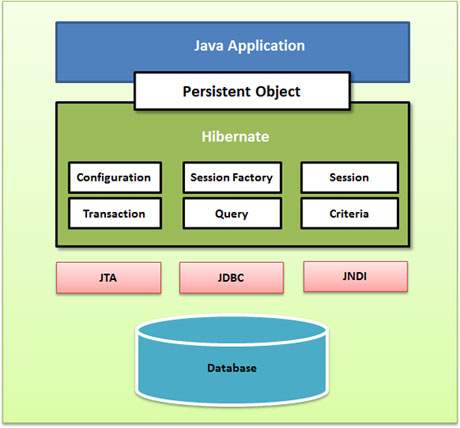
For almost all application, there is need of some background job to be performed automatically over the time consistently. This is done with the help of schedulers. Schedulers can be configured to run at specified time interval. Every time the scheduler runs, it runs the job and performed the action required. The Spring Framework provides abstractions for asynchronous execution and scheduling of tasks with the TaskExecutor and TaskScheduler interfaces, respectively.

**3.4 Hibernate**

Working with both Object-Oriented software and Relational Databases can be cumbersome and time consuming. Development costs are significantly higher due to a paradigm mismatch between how data is represented in objects versus relational databases. Hibernate is an Object/Relational Mapping (ORM) solution for Java environments. The term Object/Relational Mapping refers to the technique of mapping data between an object model representation to a relational data model representation.

Hibernate uses xml file to store configurations. Database connection details are stored in configuration file. Mapping of tables and java objects is referred using configuration file. Java class contains mapping of tables and columns with class name and member variables. Setter and Getter functions are used to perform database operations by using Java objects.

Hibernate has helped to add more functionality to object oriented programming. Hibernate takes care of mapping Java classes to database tables using XML files and without writing any line of code. Provides simple APIs for storing and retrieving Java objects directly to and from the database. If there is change in Database or in any table then the only need to change XML file properties. The entire concept of Hibernate is to take the values from Java class attributes and persist them to a database table. A mapping document helps Hibernate in determining how to pull the values from the classes and map them with table and associated fields. Java classes whose objects or instances will be stored in database tables are called persistent classes in Hibernate. Hibernate works best if these classes follow some simple rules, also known as the Plain Old Java Object (POJO) programming model.



**Figure 4: Hibernate Architecture**

**3.5 Web Services**

Web services are open standard (XML, SOAP, HTTP etc.) based Web applications that interact with other web applications for the purpose of exchanging data.

The world of web-services has been on fast track growth and use by almost all of the web applications running on the Internet. The goal of webservices is to enable two or more applications to application communication possible. This led to the growth of web services architectures. Web services helped in distributed computing.

There are three major roles within the web service architecture:

Service Provider: This is the provider of the web service. The service provider implements the service and makes it available on the Internet.

Service Requestor: This is any consumer of the web service. The requestor utilizes an existing web service by opening a network connection and sending an XML request.

Service Registry: This is a logically centralized directory of services. The registry provides a central place where developers can publish new services or find existing ones. It therefore serves as a centralized clearing house for companies and their services.

**REST API**

REST stands for REpresentational State Transfer. REST is web standards based architecture and uses HTTP Protocol for data communication. It revolves around resource where every component is a resource and a resource is accessed by a common interface using HTTP standard methods.

In REST architecture, a REST Server simply provides access to resources and REST client accesses and presents the resources. Here each resource is identified by URIs/ global IDs. REST uses various representations to represent a resource like text, JSON and XML.JSON is the most popular format being used in web services.

The above mentioned technologies have been used to develop Travel Helper application. Spring MVC for core application functionality, Spring Security for user profiling which involves user authentication, Database operation using Hibernate ORM and Rest API for connecting to third party applications like Google, Uber and Lyft.

**CHAPTER 4**

**METHODOLOGY**

Technologies discussed in previous chapters have been implemented in Travel Helper web application. MVC architecture is the base of the application. Spring MVC Framework is developed based on MVC architecture. Travel Helper is developed using Spring MVC Framework. According to the architecture, application is divided into Model, View and Controller.

**View**

This is the user interface or front-end view of the application. View are developed using Java Server Pages(JSP). Bootstrap framework is used for responsive design of web page. It helps in dynamic resize of pages based on screen size for desktop, tablet and mobile devices. Javascript, Jquery, and CSS are used for scripting and styling pages.

Travel Helper has following views:

Index – This is the landing page of the application.

Login – View for login to access application and Signup purpose.

Welcome – Landing page on successful user authentication. User can navigate to different links.

TravelSearch – This allows user to view all the options for travelling from source to destination. Results will show details of each travel option like distance, duration, cost and time estimate. User will save the option he or she is going to use to travel for historical purpose.

ScheduleTravel – User can store future travel into application and provide details about time to reach the destination. System will send notification to user when it is time to start travel.

**Model**

Model contains business logic and connects to the database.

Spring exposes methods using interfaces and classes implement interface to provide implementation of methods containing business logic.

Travel Helper Model is divided into 2 modules: UserService and TravelService

Implementation classes use Spring annotations for implementing transaction management and dependency injection.

@Autowired - used to autowire bean on the setter method.

@EnableTransactionManagement – Spring functionality for ensuring transaction locks during database operations.

Hibernate is used for performing database operation. The Hibernate Session interface provides createCriteria() method which can be used to create a Criteria object that returns instances of the persistence object's class when executing a criteria query.

SELECT operation is done using criteria query where results can be obtained by applying condition using Hibernate restriction.

INSERT is performed by creating new object. New object is saved using hibernate by using save() method.

UPDATE is performed by retrieving the object of the record by SELECT query and then updating changes using setter method of the object. Object is then passed to update() method.

DELETE is performed by retrieving the object of the record by SELECT query and then passing the object to delete() method.

There, use of hibernate 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 Hibernate Query Language.

Model classes are POJO for creating mapping between database tables and columns to corresponding Java classes and variables. The getter and setter methods are used for manipulating data. @Entity annotation is used to allow Spring to understand that the class is used for Object Relational Mapping (ORM)

**Controller**

Controller is the first component which handles client request and response. It controls the flow of request with model and view. Every request sent to controller is analysed and controller forwards the same to appropriate model class for executing business logic. After execution of business logic, response is sent to the controller. Controller calls the appropriate view to display the result and forwards the information with respect. Controller communicates directly with model and view.

@Controller - allowing for implementation classes to be autodetect the controller class through classpath scanning.

@RequestMapping (value="/welcome") - Annotation for mapping web requests onto specific handler classes and/or handler methods. Value attribute is used to map url to map to particular controller and execute the corresponding function.

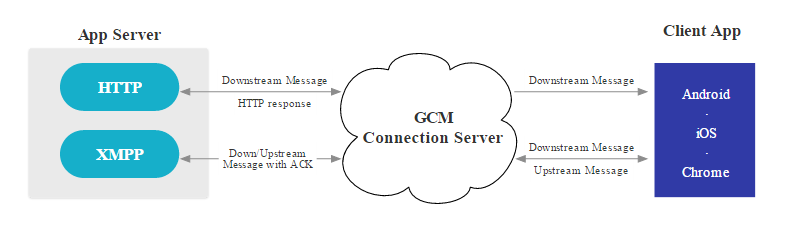
**Dispatcher Servlet**

Spring's Web MVC framework is designed around a DispatcherServlet that dispatches requests to handlers, with configurable handler mappings, view resolution, locale and theme resolution as well as support for upload files.

Dispatcher Servlet contains configuration for bean initialization, database configurations, datasource connection, base controller classpath.

**Push Notification**

Push notification is used by majority of the applications for altering user or sending update of new or important information. Push notification is sent across browser or as an alert in native android or iphone devices. Travel Helper send push notifications to user when the it is time for user to start the travel to reach the destination on time. Below is the architecture of push notification implementation



**Figure 5: Push notification request response messages**



**Figure 6: Firebase Cloud Messaging Overview**

Google FCM Connection Servers accept downstream messages from application server and send them to a client application. The XMPP or HTTP connection server can also accept messages sent upstream from the client app and forward them to your app server.

To receive and send FCM messages, application must register with FCM and get a unique identifier called a registration token. Each device is assigned a unique id which is used to for identification purpose and identify device to whom notification needs to be sent.

For every new user scheduling future travel, Travel Helper send request to Firebase for obtaining unique token for the device to send push notification on the device. Travel Helper registers each device id for the logged in user and maintains the same for sending notification. This device ID is stored in the database of Travel Helper.

Spring scheduler runs continuously and performs computations and determine the device ID to which notification need to be sent for starting travel. For each device which needs to be notified, Travel Helper application server sends a request to FCM server along with device ID.

The body of the notification message is stored in service worker configuration file. A service worker is a script that browser runs in the background, separate from a web page, opening the door to features that don't need a web page or user interaction. Therefore, user get notified via a popup even if application is not running on browser of the user. Service worker senses the push event sent by the FCM server and displays the push notification in user device.

**Rest API**

Travel Helper needs integration with Google, Uber and Lyft for data querying and integration. Integration with other application is done using REST API.

Integration with Google:

1. Google Maps API:

Autocomplete is a feature of the Places library in the Google Maps JavaScript API. When a user starts typing an address, autocomplete will fill in the rest. This will application the type-ahead-search behavior of the Google Maps search field. (https://maps.googleapis.com/maps/api/js?key=KEY&libraries=places&callback=initAutocomplete)

1. Google Distance Matrix API:

The Google Maps Distance Matrix API is a service that provides travel distance and time for a matrix of origins and destinations.

(https://maps.googleapis.com/maps/api/distancematrix/json?units=imperial&origins=Washington,DC&destinations=New+York+City,NY&key=YOUR\_API\_KEY)

Distance and duration calculation can be used for walking, cycling, driving, bicycling modes.

1. Google GeoCode API:

Geocoding is the process of converting addresses (like a street address) into geographic coordinates (like latitude and longitude) (https://maps.googleapis.com/maps/api/geocode/json?address="+addr+"&key="+API\_KEY)

1. Firebase Push API:

This API is called to send push message to device

Integration with Uber:

1. Products - The Products endpoint returns information about the Uber products offered at a given location. The response includes the display name and other details about each product, and lists the products in the proper display order.

(https://api.uber.com/v1.2/products)

1. Price Estimate - The Price Estimates endpoint returns an estimated price range for each product offered at a given location.

(https://api.uber.com/v1.2/estimates/price)

1. Time Estimate - The Time Estimates endpoint returns ETAs for all products currently available at a given location, with the ETA for each product expressed as integers in seconds.

(https://api.uber.com/v1.2/estimates/time)

Integration with Lyft:

1. Ride Types - The ridetypes endpoint returns the ride types available at the specified location, indicated by the specified latitude and longitude

(https://api.lyft.com/v1/ridetypes)

1. Cost - The cost endpoint returns the estimated cost, distance, and duration of a ride between a start location and end location. A success response will be broken down by ride types available at the specified location.

(https://api.lyft.com/v1/cost)

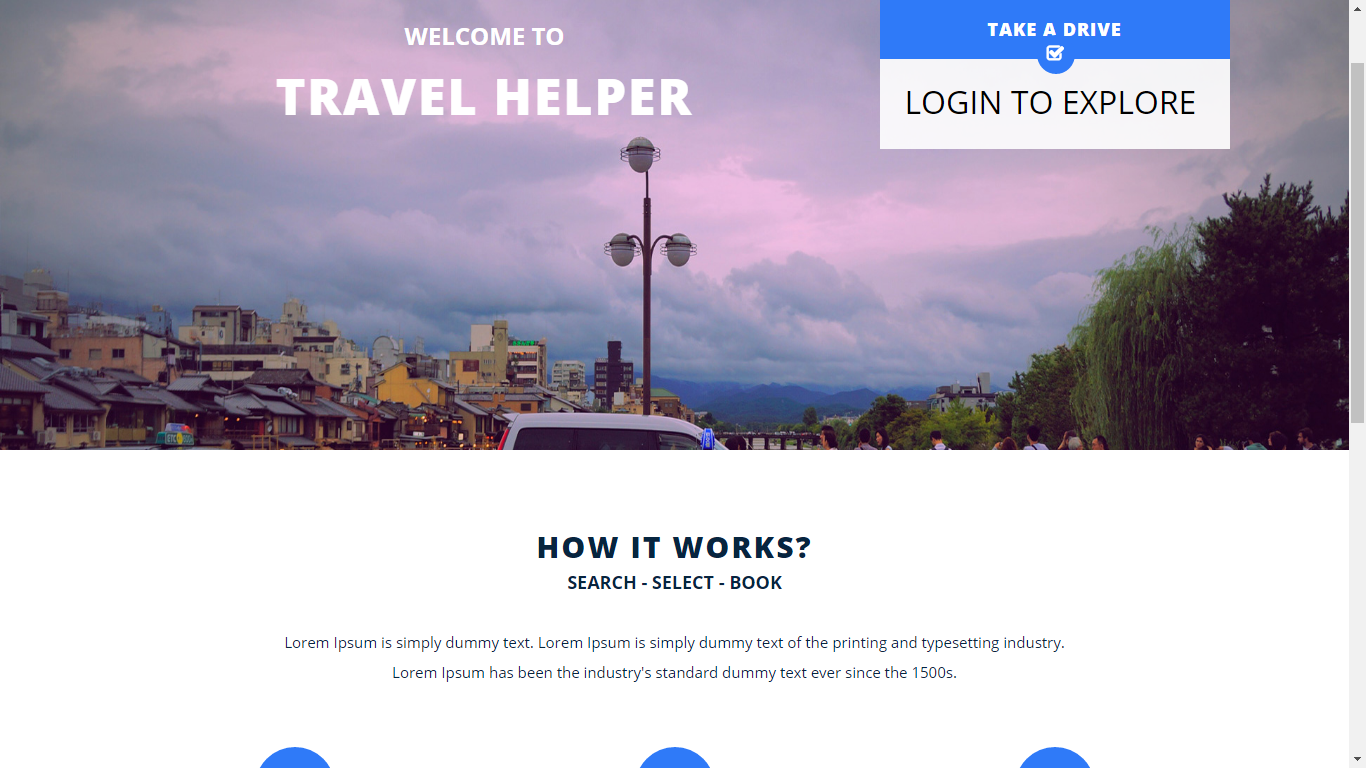
1. TimeEstimate - The eta endpoint returns the estimated time in seconds it will take for the nearest driver to reach the specified location. A success response will be broken down by ridetypes available at the specified location.

(https://api.lyft.com/v1/eta)

**CHAPTER 5**

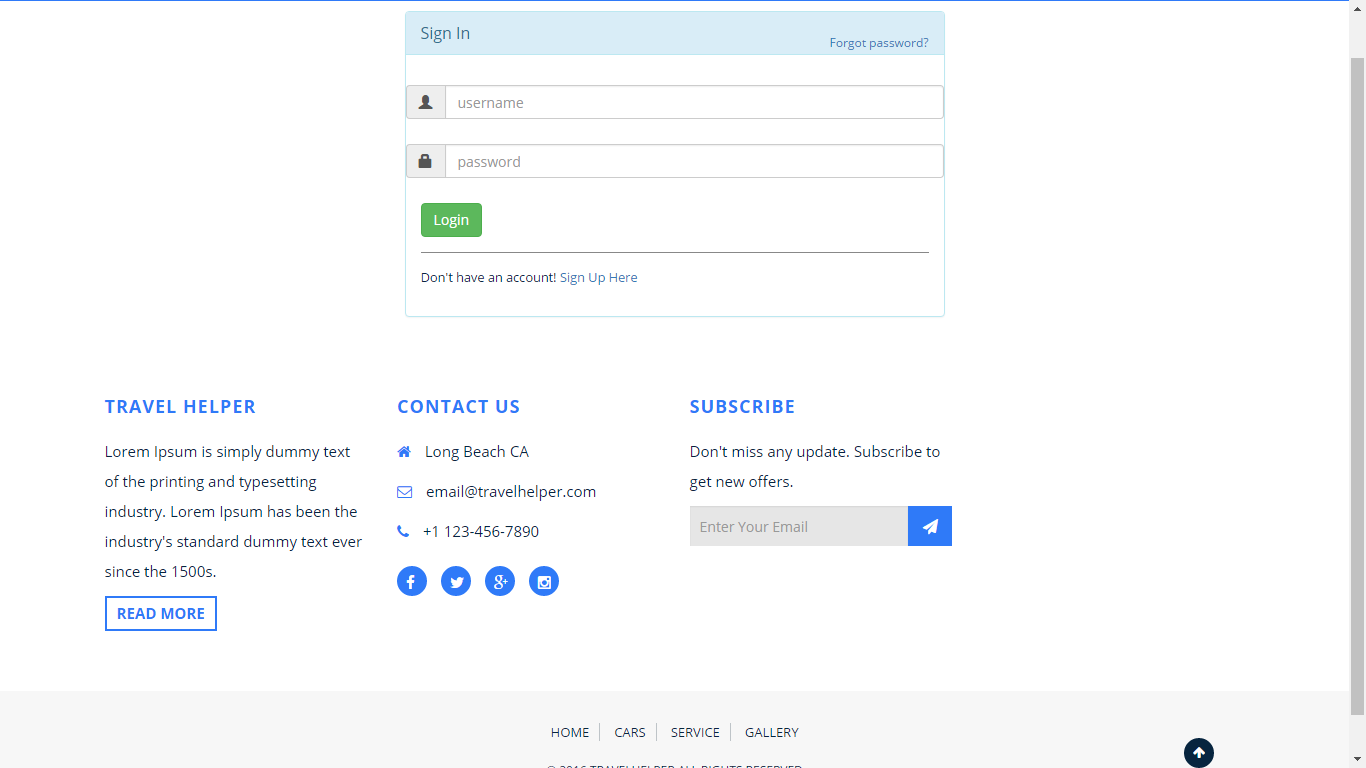
**RESULTANT SYSTEM**

In the Previous chapters, the technologies were explained in detail. MVC architecture and Spring MVC framework are the base of the application. The functionality of the application was developed with many modules like Spring security, scheduler, Google Firebase Push Notification and third-party API’s of Google, Uber and Lyft. This chapter will present mockup of the Travel Helper application.



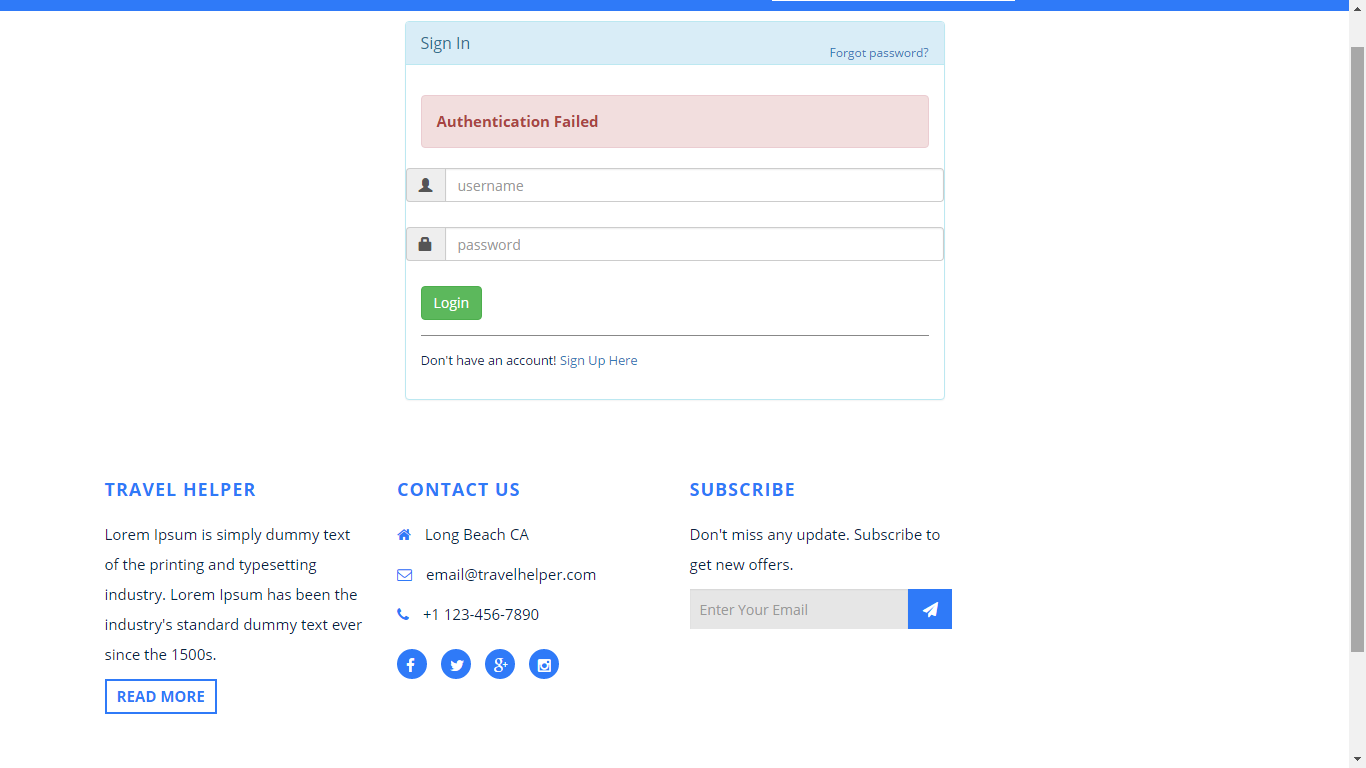
**Figure 7. Landing page**

This is the landing page of the application giving basic look and feel of the application and information of what it is. User has to login to the explore the application



**Figure 8: Login Page**

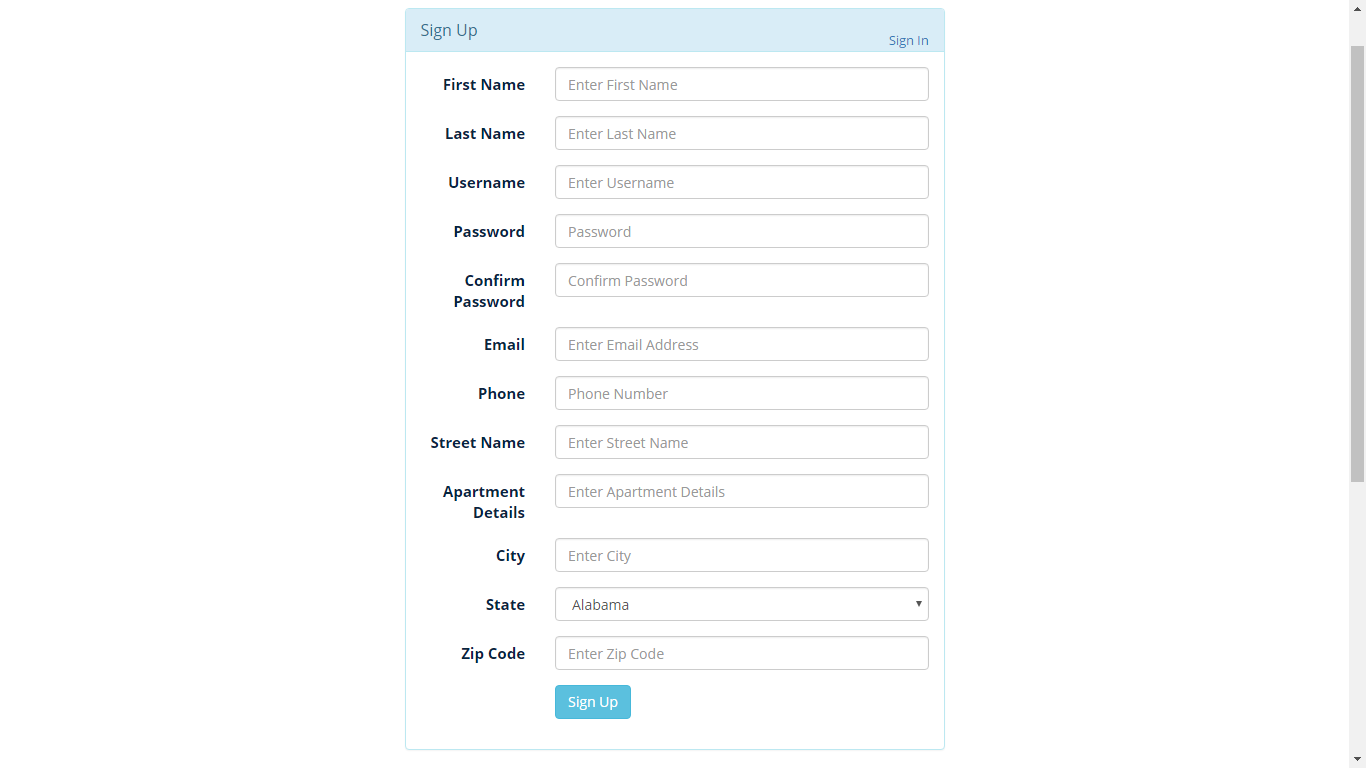
This is the login page. Registered user has to enter username and password to access account. If the entered credentials are correct, user will be redirected to welcome page. In case of incorrect credentials, application will deny access to account and will display error on login page as shown in figure xx.



**Figure 9: Authentication error page**

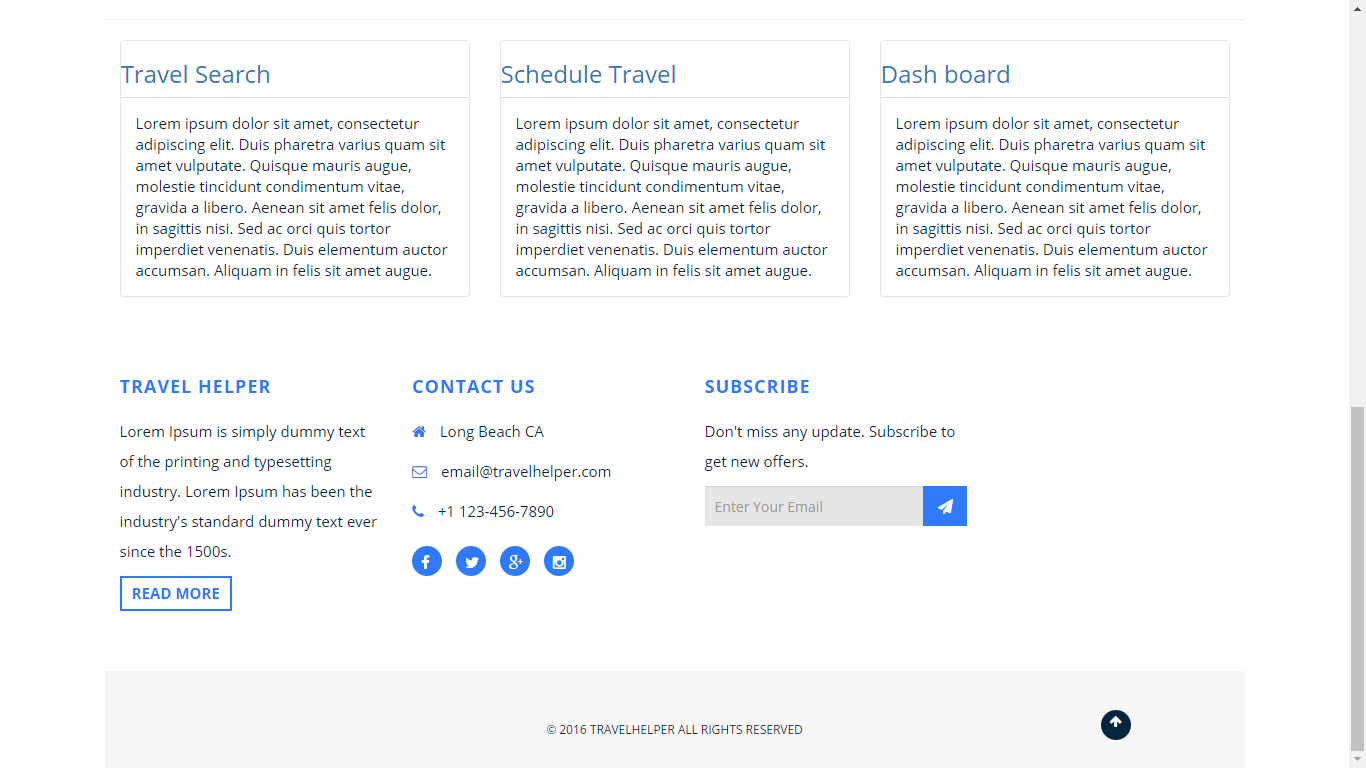
User authentication is implemented using Spring Security framework. Spring eases authentication and authorization functionality.

New user has to signup to create new account and access the application.



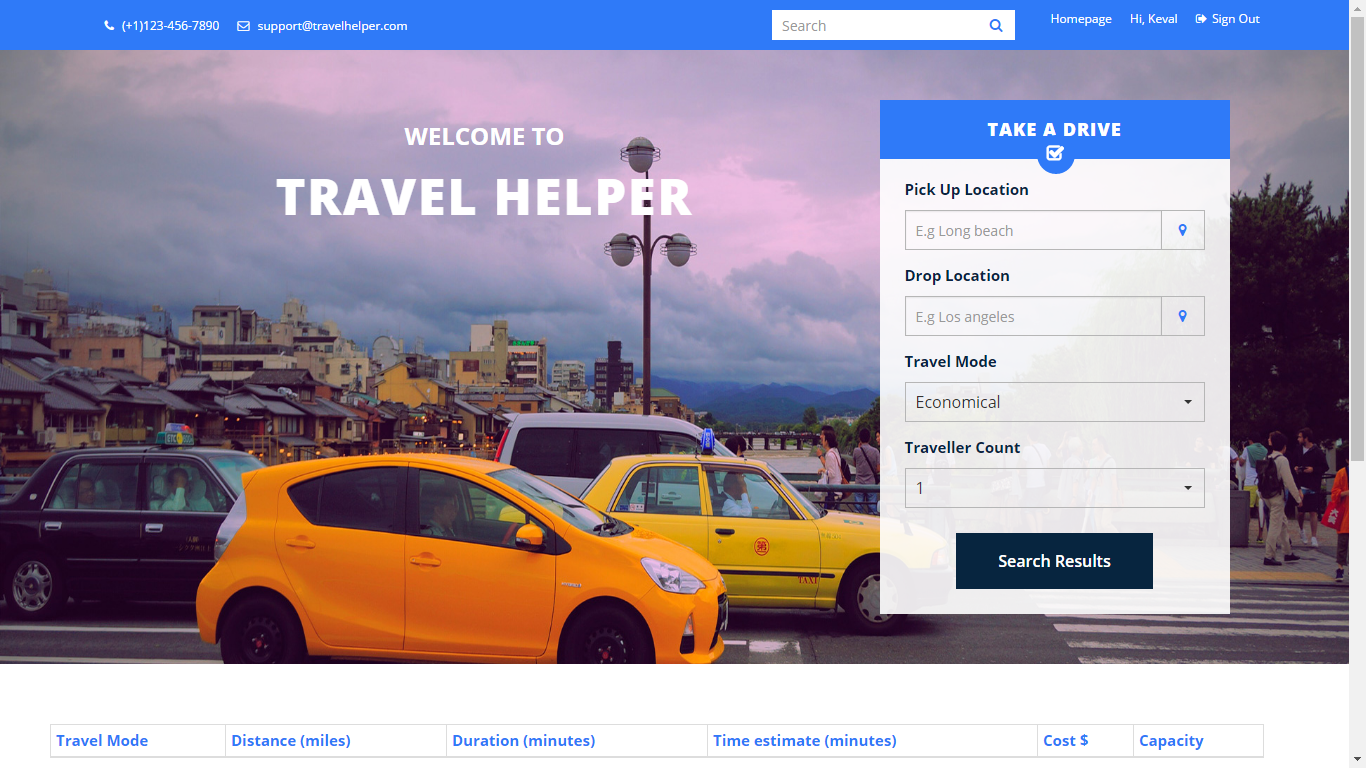
**Figure 10: Signup Form**

Once the user successfully logs into the system, user will be redirected to welcome page.



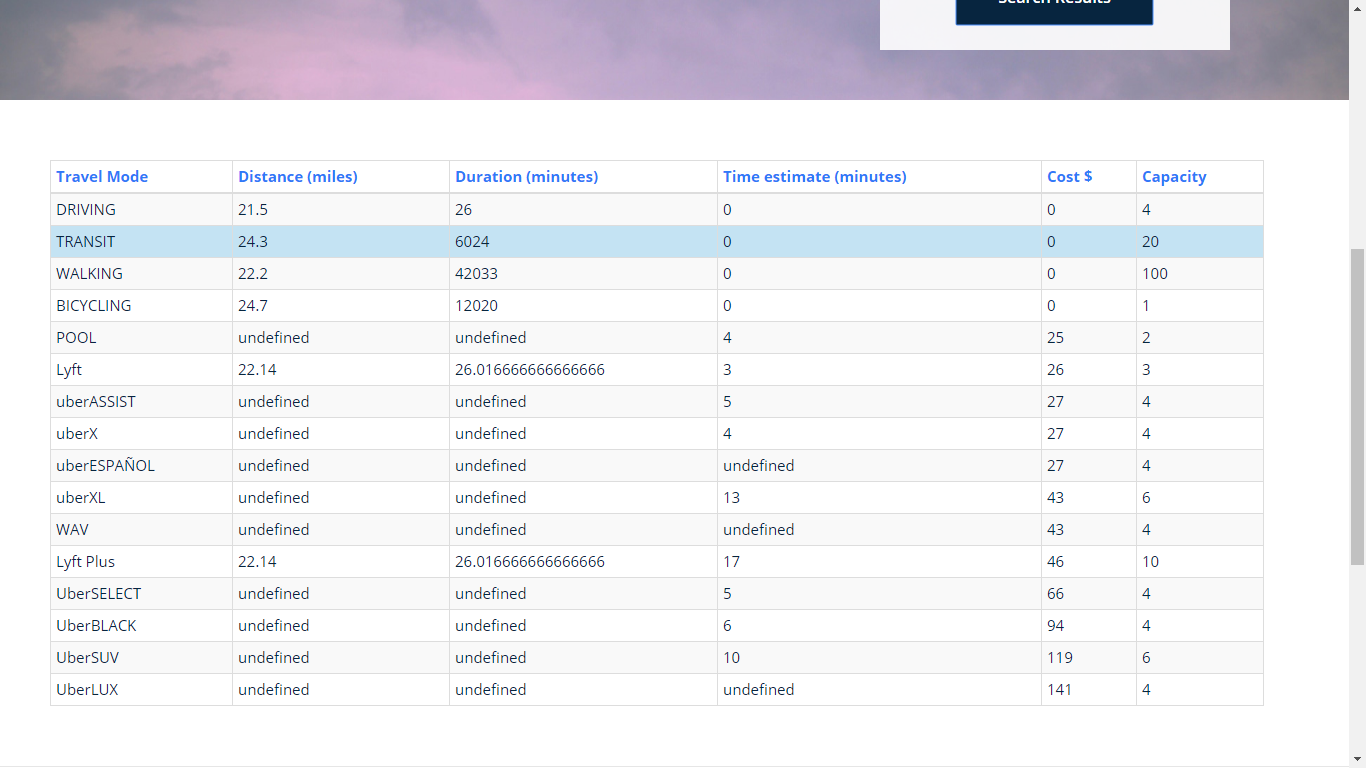
**Figure 11: Home Page of the application**

When user click on Travel Search page, page displayed in figure XX opens up.



**Figure 12: Travel Mode Search Page**

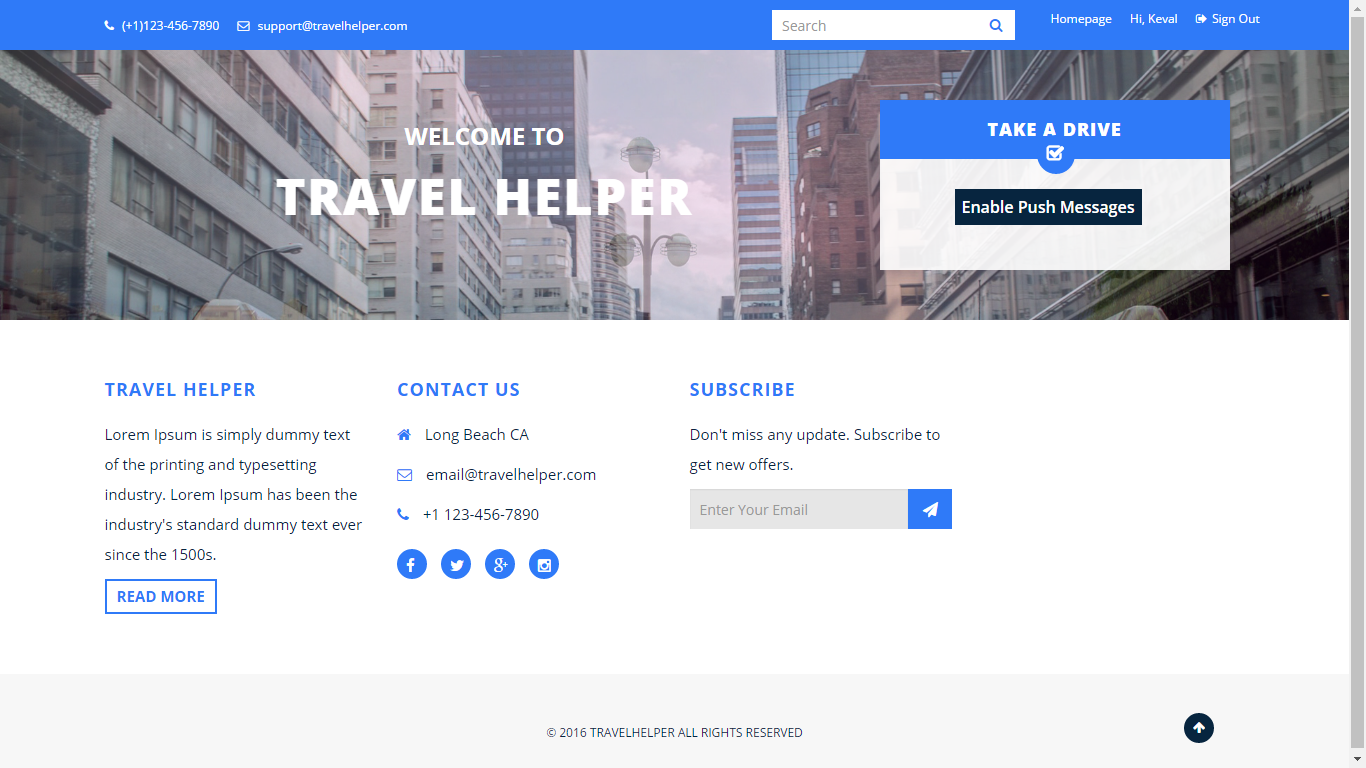
User selects source and destination along with travel mode and number of travellers. On click of search results, application connects to Google, Uber and Lyft API’s. Data for distance, duration and cost is obtained to make best travel decision.



**Figure 13: Result of travel search showing available travel information**

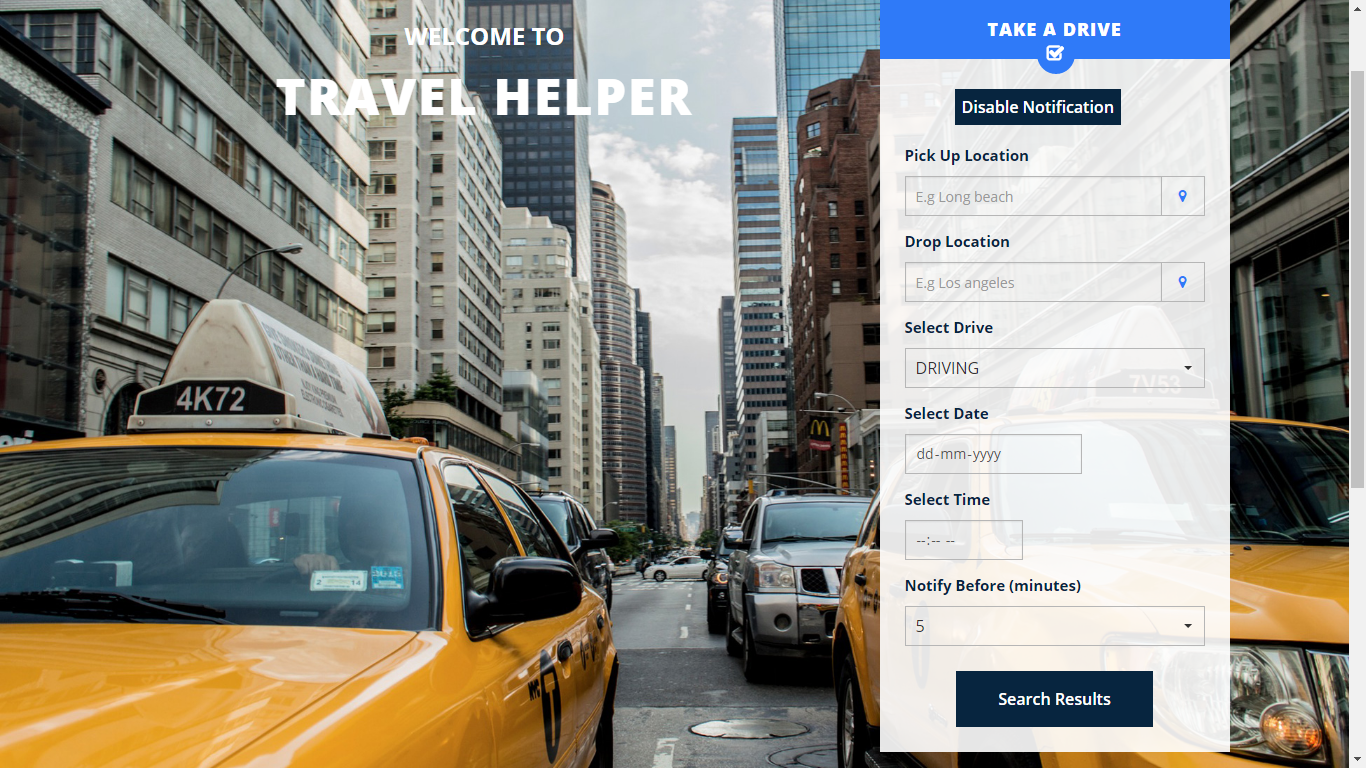
Once user decides on the travel mode which will be best to travel, preference can be stored in the application as it will be stored for historical analysis. This can be done by clicking on the row of the travel mode.

For, scheduling future travel user has to navigate to schedule travel page. Since the user will be notified during start of travel, notifications need to be enabled by click on “Enable Push Messages” as show in figure XX.



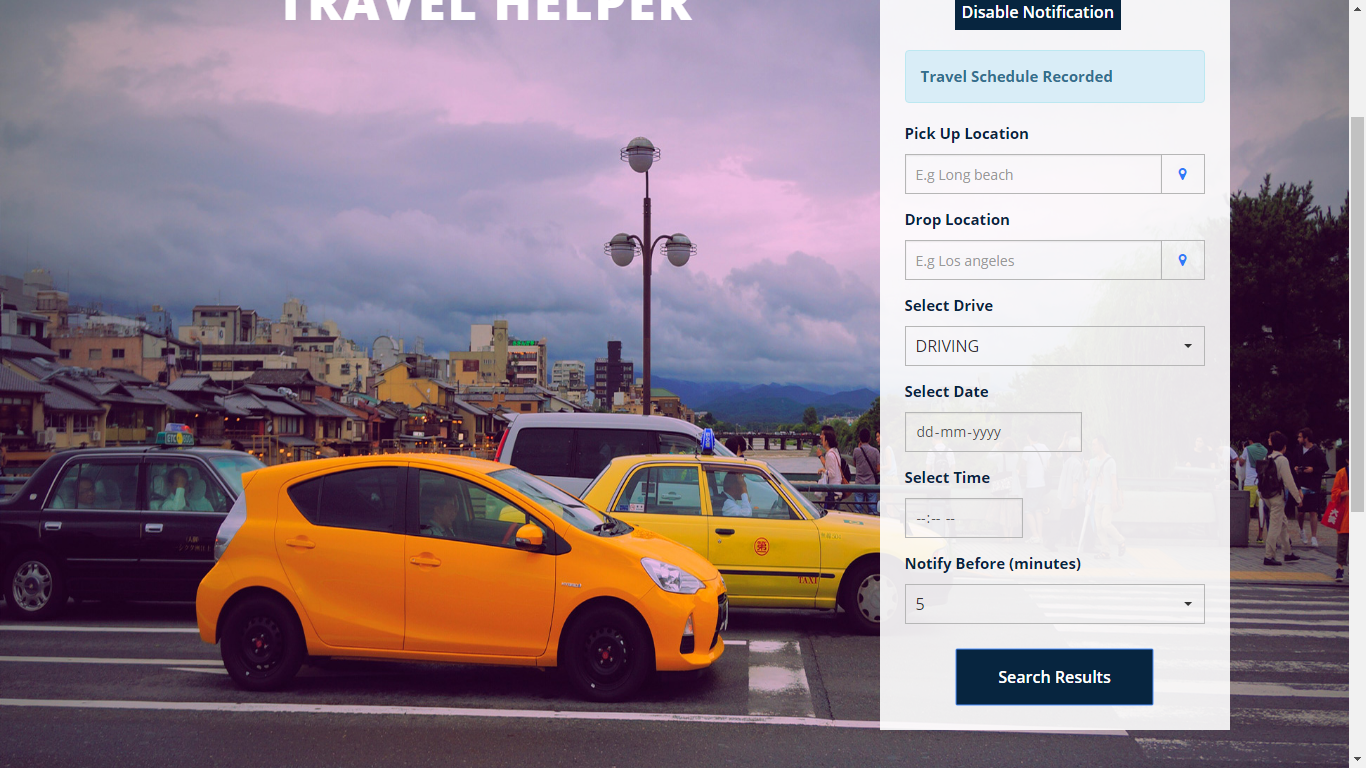
**Figure 14: Schedule Travel Page with Enable Notification screen**

Once the push notification is enabled user can schedule the future travel and store the same in the application for notification.



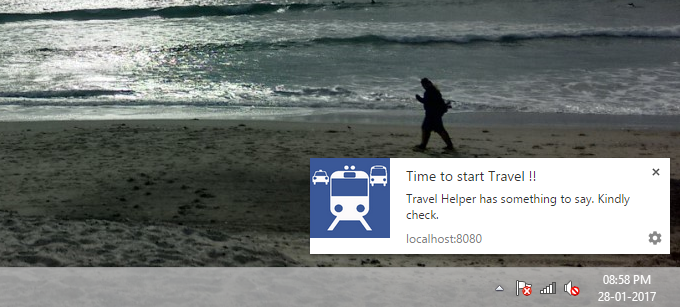
**Figure 15: Form to enter details for scheduling future travel**

For scheduling travel, user has to enter the details as specified in Figure XX. Notifications can be disabled anytime by clicking on “Disable Notification” button. Once the details are entered a confirmation message “Travel Schedule Recorded will be displayed”. Now a push notification will be sent by application when user needs to start travel.



**Figure 16: Confirmation message on saving travel information for notification**

Scheduler runs in the background as a service and checks every minute for notification eligible record. Therefore, after saving future travel, notification was sent to user by application to start travel to reach destination on time with travel mode specified earlier. It is not necessary for Travel Helper to be open in browser for receiving notification as seen in figure XX.



**Figure 17: Push notification from application indicating start of scheduled travel.**

User can view past history about travel using Dashboard.

<< Development work is being done on those screen >>

**CHAPTER 6**

**CONCLUSION**

The chapters in the project have been described to explain importance of web application development using Model-View-Controller framework. The report started with introducing software development using object oriented analysis and design. Requirement and Design diagrams are a critical part of software development lifecycle as they lay a strong foundation of software development.

In further chapters, MVC architecture and Spring MVC are discussed which form the base of the Travel Helper Application. Travel Helper is a web application which is developed using Spring MVC framework.

Further chapters included details understanding of concepts of the Spring architecture, spring components like spring security, spring scheduler, hibernate and Rest API which are used in development of Travel Helper application.

Travel Helper helps in determining the best travel mode to reach from source to destination. Future travel can also be saved by user, where the application will notify the user at the time of starting the travel to reach destination on time. Application also has dashboard features for past travel data analysis. User account are managed by the application for personalization purpose.

Therefore, MVC architecture allows ease in development of application by separating coupling between view and model where controllers control the flow of application. Use of spring MVC framework along with features of spring security, schedulers help developer in avoiding writing boiler plate code and enhances code reuse. Spring also helps in integrating easily with REST API’s of third party applications for integration.

Therefore, Travel Helper provides exciting features in making efficient travel decision making and scheduling travel for push notification. This application development using MVC and Spring Framework exhibits benefits of developing enterprise web application using these technologies. In 10 years the Web has changed the way we live, but it’s got more change left to give.

**APPENDIX**

**USE CASE**

**TABLE 1:**

|  |  |
| --- | --- |
| Use-case Name: | 1. New Customer registration |
| Actors: | Customer |
| Pre-Conditions: | 1.   The application is running.  2.   Database connection established. |
| Flow of Control: | 1.   Customer clicks on ‘Register’ button on the system.  2.   System acknowledges and presents registration form to the customer.  3. Customer inputs information into appropriate fields.  4. Customer submits the form.  5. System saves the form. |
| Post-Conditions: | 1.   New customer created.  2.   New customer information stored in 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. Availability – Web application must be available 95% of time in a span of 24 hours. 2. Performance - 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.   1. Security - Maximum failed login attempt allowed must be 5. 2. Compatibility – User Interface should be rendered as expected in browsers like Internet Explorer (above version 8), Mozilla Firefox, Chrome and Safari |

**TABLE 2:**

|  |  |
| --- | --- |
| Use-case Name: | 1. Customer Login |
| Actors: | Customer |
| Pre-Conditions: | 1. The application is running.  2. Database connection established.  3. Account already exists for the customer. |
| Flow of Control: | 1. Customer clicks on ‘Login’ button.  2. System acknowledges and presents login form to the customer.  3. Customer inputs information into appropriate fields.  4. Customer clicks on ‘Submit’ button. |
| Post-Conditions: | 1.   User successfully logs in to the system.  2.   User can access account details, search travel option and schedule travel. |
| Error-Conditions: | 1. Incorrect username and password. 2. Internet connection lost. 3. Database connectivity lost. |
| Non-Functional Requirements: | 1. Availability – 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 – User Interface should be rendered as expected in browsers like Internet Explorer (above version 8), Mozilla Firefox, Chrome and Safari |

**TABLE 3:**

|  |  |
| --- | --- |
| Use-case Name: | 1. Search economic travel option |
| Actors: | Customer |
| Pre-Conditions: | 1. The application is running.  2. Database connection established.  3. Customer is registered with application.  4. API is running. |
| Flow of Control: | 1.   Customer clicks on ‘Economical Travel’ button on the system.  2.   System prompts for source and destination location from customer.  3.   System calls API to find travel cost using Uber, Lyft, LB Transit or Metro at current time.  4.   System shows all possible options with less cost option at top.  5.   Customer selects travel option.  5.1 If customer selects booking option, see use case no. 4  6. System saves travel preference selected by customer. |
| Post-Conditions: | 1.   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 – 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:**

|  |  |
| --- | --- |
| Use-case Name: | 1. Search faster travel option |
| Actors: | Customer |
| Pre-Conditions: | 1. The application is running.  2. Database connection established.  3. Customer is registered with application.  4. API is running. |
| Flow of Control: | 1.   Customer clicks on ‘Faster Travel’ button on the system.  2.   System prompts for source and destination location from customer.  3.   System calls API to find travel cost using Uber, Lyft, LB Transit or Metro at current time.  4.   System shows all possible options with fastest travel option at the top.  5.   Customer selects travel option.  5.1 If customer selects booking option, see use case no. 4  6. System saves travel preference selected by customer. |
| Post-Conditions: | 1.   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 – 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:**

|  |  |
| --- | --- |
| Use-case Name: | 1. Customer saves travel decision |
| Actors: | Customer |
| Pre-Conditions: | 1.   The application is running.  2.   Database connection established.  3.   Customer is registered with application. |
| Flow of Control: | 1.   Customer clicks on Travel option on the table row to save the preference selected.  2.   Details are saved into system. |
| Post-Conditions: | 1.   Details regarding travel preference are stored in database. |
| Error-Conditions: | 1. Database connection lost |
| Non-Functional Requirements: | Usability: See Non Functional Requirement Section 1 |

**TABLE 6:**

|  |  |
| --- | --- |
| Use-case Name: | 1. Schedule future travel for notification |
| Actors: | Customer |
| Pre-Conditions: | 1.   The application is running.  2.   Database connection established. |
| Flow of Control: | 1. Customer clicks on ‘Schedule travel’ button on the system.  2. System acknowledges and presents travel schedule form to the customer.  3. Customer enters destination and expected time to reach destination.  4. Customer submits the form.  5. System saves the form. |
| Post-Conditions: | 1. 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:**

|  |  |
| --- | --- |
| Use-case Name: | 1. System prompts user about travel start time |
| Actors: | Travel Scheduler (TS) |
| Pre-Conditions: | 1. The application is running.  2. Database connection established.  3. API is running.  4. Google cloud connectivity is established.  5. Customer Internet is working. |
| Flow of Control: | 1. TS sends notification to customer regarding start of travel to reach destination in time. 2. Customer clicks on notification to view notification. 3. 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: | Interoperability: Data exchange with google cloud messaging API should be done using REST.  Availability: Connectivity with google cloud should be established 98% of time. |

**TABLE 8:**

|  |  |
| --- | --- |
| Use-case Name: | 1. Customer views travel dashboard |
| Actors: | Customer |
| Pre-Conditions: | 1.   The application is running.  2.   Database connection established.  3.   Customer is registered with application. |
| Flow of Control: | 1. Customer clicks on ‘Travel Dashboard’ button on the system.  2. System queries database for fetching details.  3. Travel summary is displayed to customer. |
| Post-Conditions: | NA |
| Error-Conditions: | NA |
| Non-Functional Requirements: | Usability: See Non Functional Requirement Section 1 |

**TABLE 9:**

|  |  |
| --- | --- |
| Use-case Name: | 1. Filter report data on date range |
| Actors: | Customer |
| Pre-Conditions: | 1.   The application is running.  2.   Database connection established.  3.   Customer is registered with application. |
| Flow of Control: | 1. Customer clicks on ‘Filter Results’ button on the system.  2. System queries database based on filter for fetching details.  3. Travel details are displayed to customer. |
| Post-Conditions: | NA |
| Error-Conditions: | NA |
| Non-Functional Requirements: | Usability: See Non Functional Requirement Section 1 |

**TABLE 10:**

|  |  |
| --- | --- |
| Use-case Name: | 1. Download travel history report |
| Actors: | Customer |
| Pre-Conditions: | 1.   The application is running.  2.   Database connection established.  3.   Customer is registered with application. |
| Flow of Control: | 1. Customer clicks on ‘Download report’ button on the system.  2. System generates pdf format report.  3. Customer saves report in his computer. |
| Post-Conditions: | NA |
| Error-Conditions: | NA |
| Non-Functional Requirements: | Accessibility: Document should be accessible within 5 minutes.  Usability: See Non Functional Requirement Section 1 |

**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 like JSON, XML. |
| * Capability to send request and response using common protocols like 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 like 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%. |

**Database Tables**

**USER\_PROFILE**

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Datatype** | **Remark** |
| USER\_ID | Integer (20) | Primary key, auto\_increment |
| 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, auto increment |
| USER\_ID | Integer (20) | Foreign Key |
| START\_POINT\_X\_COORDINATE | Varchar(20) | **Google Co-ordinates** |
| 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) | References “TRAVEL\_MODE\_ID” from Table “TRAVEL\_MODE” |
| TRAVEL\_DRIVE\_SELECTED | tinyint (3) | References “DRIVE\_ID” from Table “TRAVEL DRIVE” |
| 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) | References “DRIVE\_ID” from Table “TRAVEL\_DRIVE” | |
| NOTIFICATION\_TIME | datetime |  | |
| destination\_reach\_time | datetime |  | |
| traveltime\_expected | Int (20) | Time In minutes | |
| 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**

[1] I.T., Titanium. "James Shore: Dependency Injection Demystified". www.jamesshore.com. Retrieved 2015-07-18.

[2] Prasanna, Dhanji R. *Dependency injection*. Manning Publications Co., 2009. (Dependency injection)

[3] Yang, Hong Yul, Ewan Tempero, and Hayden Melton. "An empirical study into use of dependency injection in java." *Software Engineering, 2008. ASWEC 2008. 19th Australian Conference on*. IEEE, 2008. (Dependency injection)

[4] https://www.youtube.com/watch?v=Q3yStECBuAg - spring security arch

[5] http://docs.jboss.org/hibernate/orm/5.2/quickstart/html\_single/

[6] http://docs.spring.io/spring-framework/docs/2.0.x/reference/mvc.html

[7] https://developers.google.com/web/fundamentals/getting-started/codelabs/push-notifications/