**Architecture and Detailed Design Specification**

**GET-A-WAY**

# **Revision History**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Date** | **Description** | **Author** | **Approver** | **Comments** |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

# **Document Approval**

The following Software Requirements Specification has been accepted and approved by the following:

|  |  |  |  |
| --- | --- | --- | --- |
| **Signature** | **Printed Name** | **Title** | **Date** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**CONTENTS**

|  |  |  |
| --- | --- | --- |
| **Sl.** **No.** | **TITLE** | **PAGE NO.** |
| 1. | Architectural and component ­level design |  |
| 2. | System Structure |  |
|  | 2.1 Architectural Diagram |  |
|  | 2.2 Description of Components |  |
|  | 2.3 Interaction Diagram |  |
|  | 2.4 Usage Scenario |  |
|  | 2.5 Architectural Styles and Patterns |  |
| 3. | User Interface Design |  |
| 4. | Detailed Design Approach |  |
|  | 4.1 Design Patterns considered |  |

1.0 Architectural and component-level design

Get-A-Way is a **client-server** based system.The product consists of a webpage and an android application. The webpage and the application both contain a client side and a server side, as well as a front end and a back end. Both the webpage and the android page can be used to sign up, log in, view the spots, select the locations the user would like to visit and find the optimal path. However, only the app can be used once the user starts his trip (to change the route, emergencies etc).

**The server side**

* The server side has a database which stores the usernames and passwords of the users on sign up, which will be required to verify a user's identification on log in.
* It also maintains a database which contains a list of all the places within Karnataka which can be considered tourist spots.
* For each of these places, it contains the corresponding details such as location, timings, best time to visit etc.

**The client side**

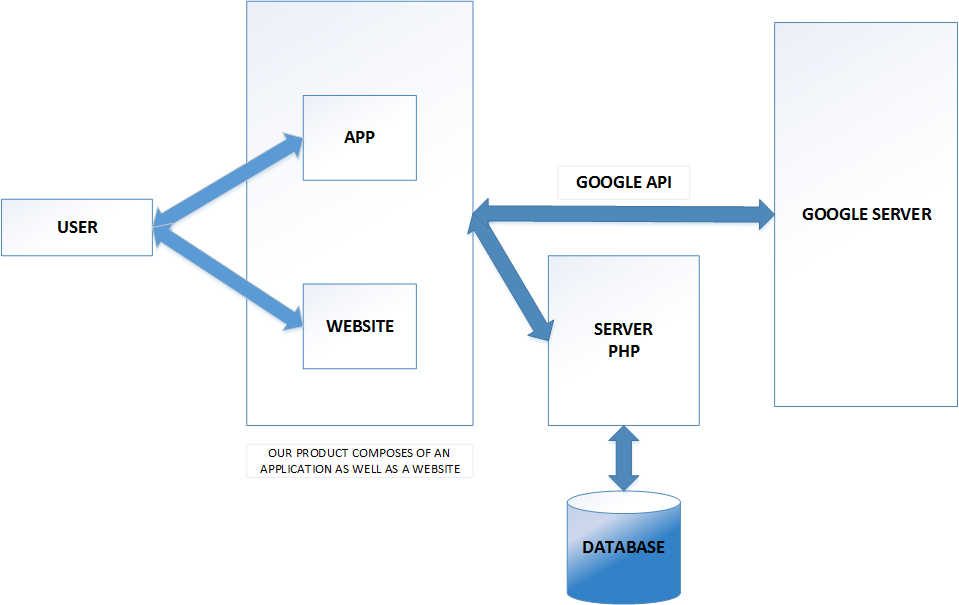
This is the side which would be made use of by someone who is traveling with the help of our product. The client side is private to a particular user. The user needs to log in according to the username and password that he/she has registered with when signing up. (This is stored and verified in the database on the server side).

Once the identification of a user is verified, the user may proceed to plan his trip. The client side allows the user to select the places that he would like to visit. Once the user has identified the locations, the server receives this information and proceeds to calculate the optimal path that the user can take. This may be done on the app or the webpage.

While on the trip, the front end of the app uses maps to show the user's exact location, the path suggested to him, nearby places in case of emergencies or a sudden change of plans. In case the user wishes to change the initial path selected, the client side deals with such issues. According to the locations selected by the user, an optimal path which covers all the places is selected. A ChatBot present on the app makes it more user friendly, thus providing information in a more interesting manner.

**2.0 System Structure**

**2.1 Architecture diagram**





**2.2**

**Description for Component**

**Front end UI(Website)**

The front end of the website takes user input such as places to visit and sends it to the server and receives the response from the server and parses it to display the result such as optimal path between places,nearby amenities to the user.User can save his trip by logging in.

**Front end UI(Android app)**

The front end of application has a Login feature where user inputs his login credentials that is sent to the server for authentication and receives the user’s saved trips as a response on user’s account. User can select one of his saved trips and a Google Map with optimal route and nearby amenities appear.

**Back end server**

The server is mainly used for user validation. This server can be accessed from the app as well as the website. It also returns the list of places that are considered as tourists attractions in a particular district of Karnataka on demand from the user.

**Google APIs**

Google APIs are a set of interfaces which allow communication between backend server and Google Services.

List of Google APIs used:

**=>** Google Places API Web Service

**=>** Google Maps Geocoding AP1

**=>** Google Maps Directions API

**=>** Google Maps Geolocation API

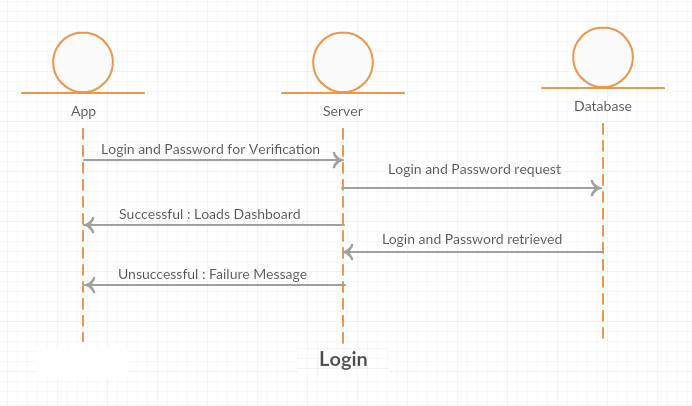
**=>** Google Maps Places API

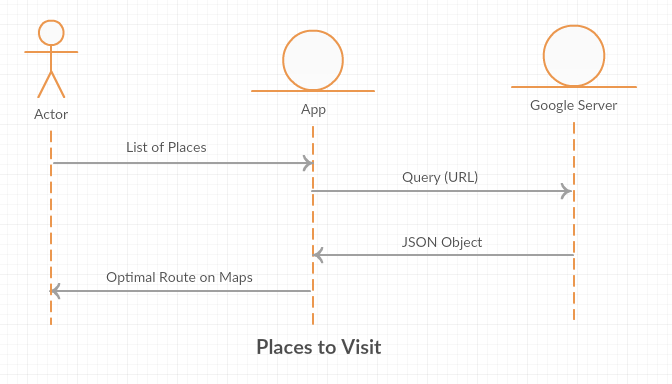
**Database**

The database maintains the information about a user and his saved trips, a list of all the places within Karnataka which can be considered tourist spots and the corresponding details such as location, timings, best time to visit etc.Requests for this information is sent from the front end to server, and in turn from server to database.The server receives the response,processes it and sends it to front end.

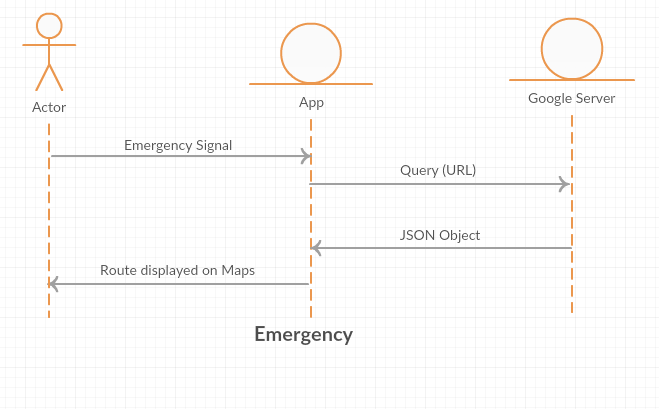
**2.3 Interaction Diagrams**

**Sequence diagram 1 : Logging into System**

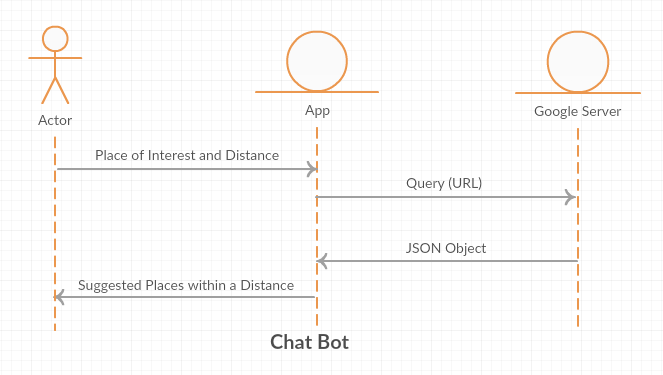


**Sequence diagram 2 : Planning a Trip**

**Sequence diagram 3 : Emergency**



**Sequence diagram 4 : Chatbot**



**2.4 Describe usage scenarios and how you would test that**

**Usage Scenarios**

1. The website can individually be used to lookup places for a good trip and also plan a trip accordingly. The perfect time of visit for a place and also nearby tourist places will be suggested to the user.

2. The mobile app too can be used to look up for places and plan the trip. The extra usage is services on the go of a trip such as a medical emergency,flat tire etc.. which suggests the best place nearby based on the emergency.

3. A general usage of the app and website can also be to just check out the best route from a source place to one or multiple destinations.

4. The chatbot feature in the app can be used to have an interactive trip.

**Testing**

1. Ensure that all the functionalities of website works across different browsers.

2. Test the website and app for optimal routes between different places.

3. Plan a completer trip in website or on the app and also use it on the go and also

check for an emergency situation.

4. Check the GPS and routes suggested by app on the go by visiting a few places.

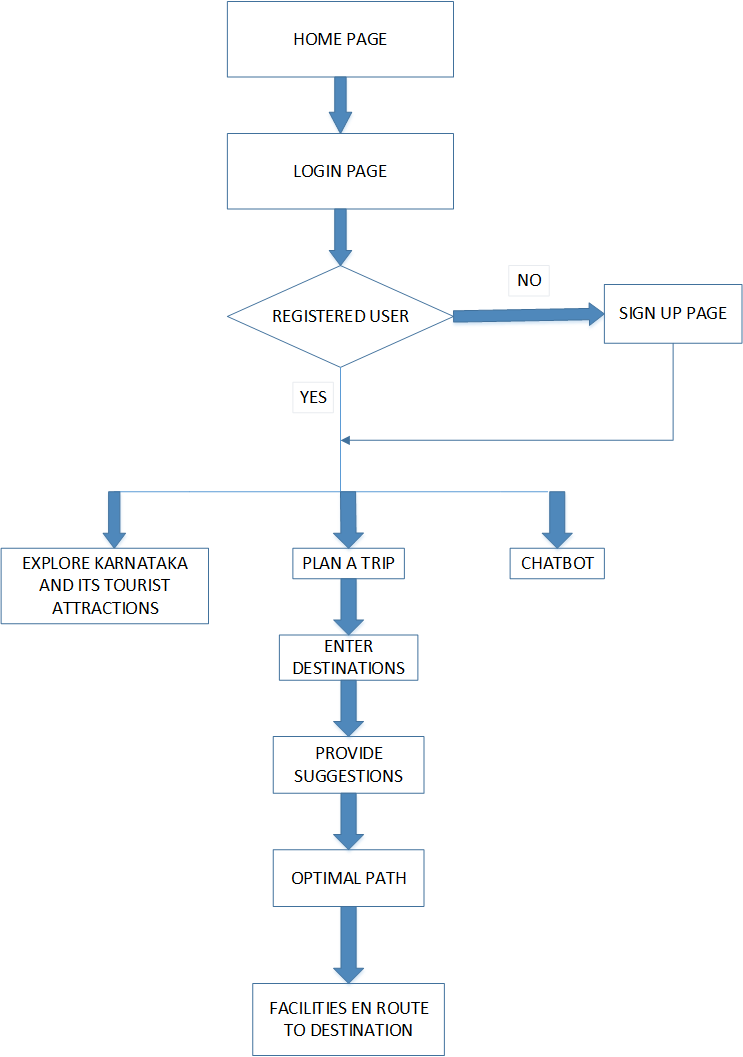
5. Change the route and check if services suggested by app is still valid and the next optimal route to complete the trip.

**2.5 Architectural Styles and Patterns considered and for what reason**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No** | **Architectural Styles/ Pattern** | **Intent of this pattern** | **Rationale for choosing** |
| 1 | Client-Server Architecture | Data in a shared database has to be accessed from a range of locations. Because servers can be replicated, may also be used when the load on a system is a variable. | As more users access the system a three-tier solution is more scalable than the other solutions because you can add as many middle tiers as needed to ensure good performance.  Security is also the best in the three-tier architecture because the middle layer protects the database tier. |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No** | **Architectural Styles/ Pattern** | **Intent of this pattern** | **Rationale for not choosing** |
| 1. | N­Tier | Segregates functionality into separate segments with each segment being a tier located on a physically separate computer. | We rejected this pattern as we did not have sufficient hardware to allocate the required number of servers to each tier. |
|  |  |  |  |

**3. User Interface Design**



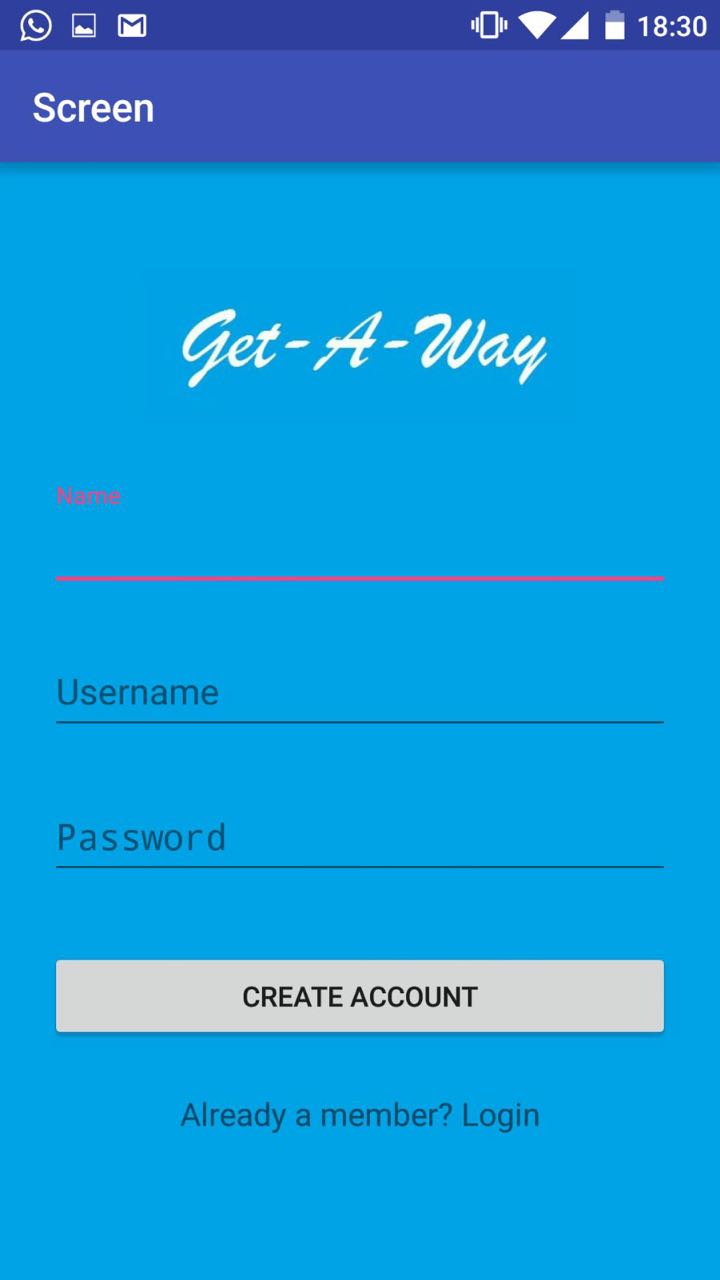
1. **Android Division**

**Splash screen**

■ Displayed until the application loads

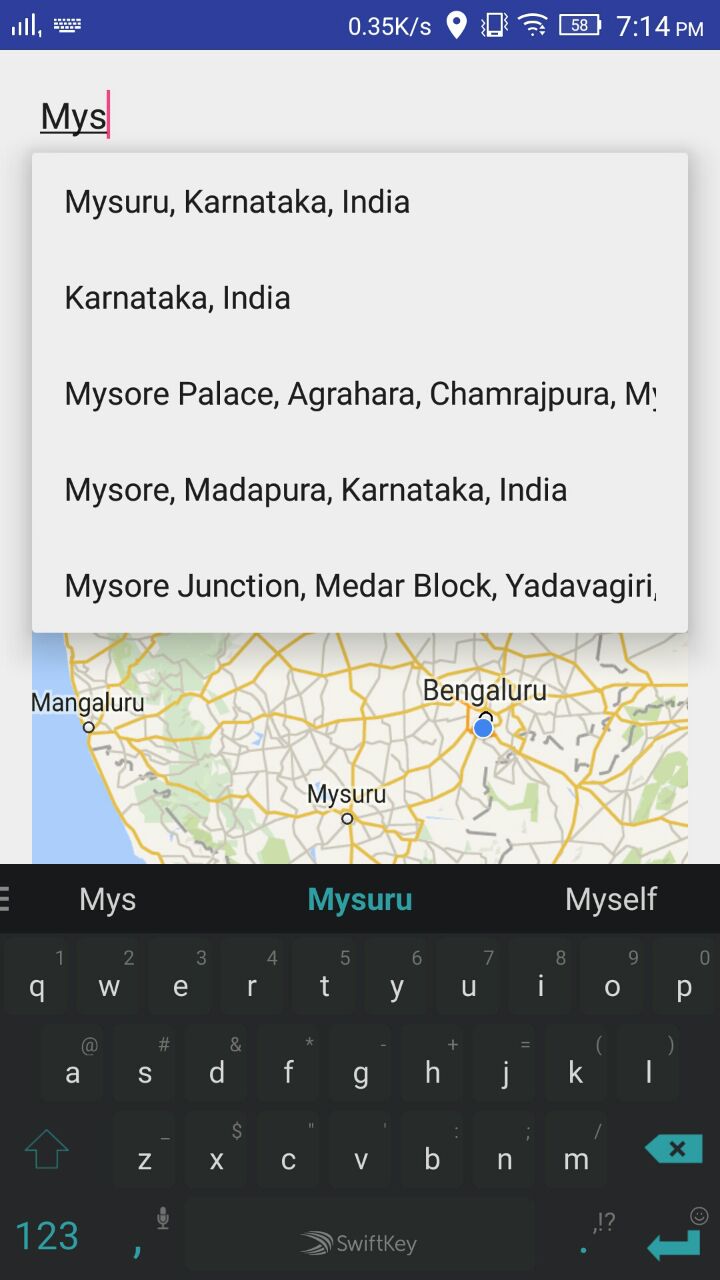


**Sign-in Page**



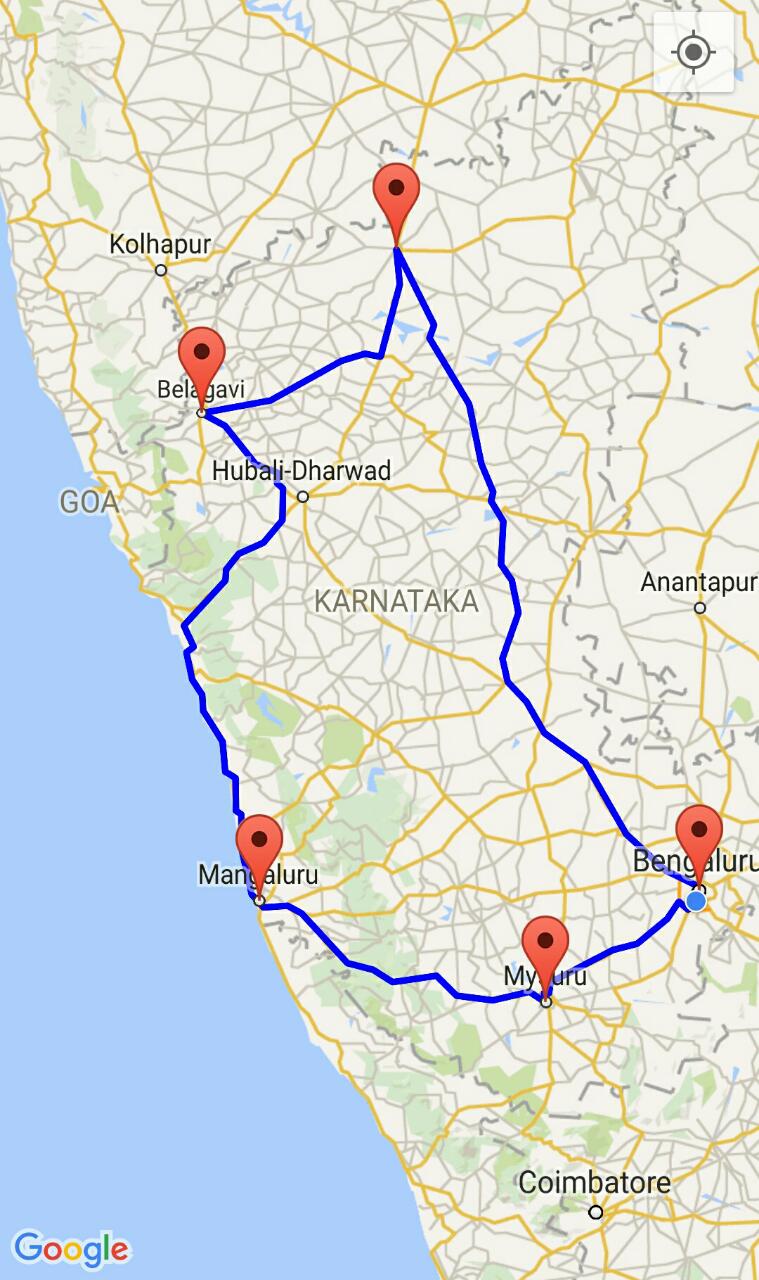
**Plan a trip**

This page allows the user to enter his/her destination. Multiple destinations can be accepted. An autocomplete feature is provided to ease the task of entering the destinations.



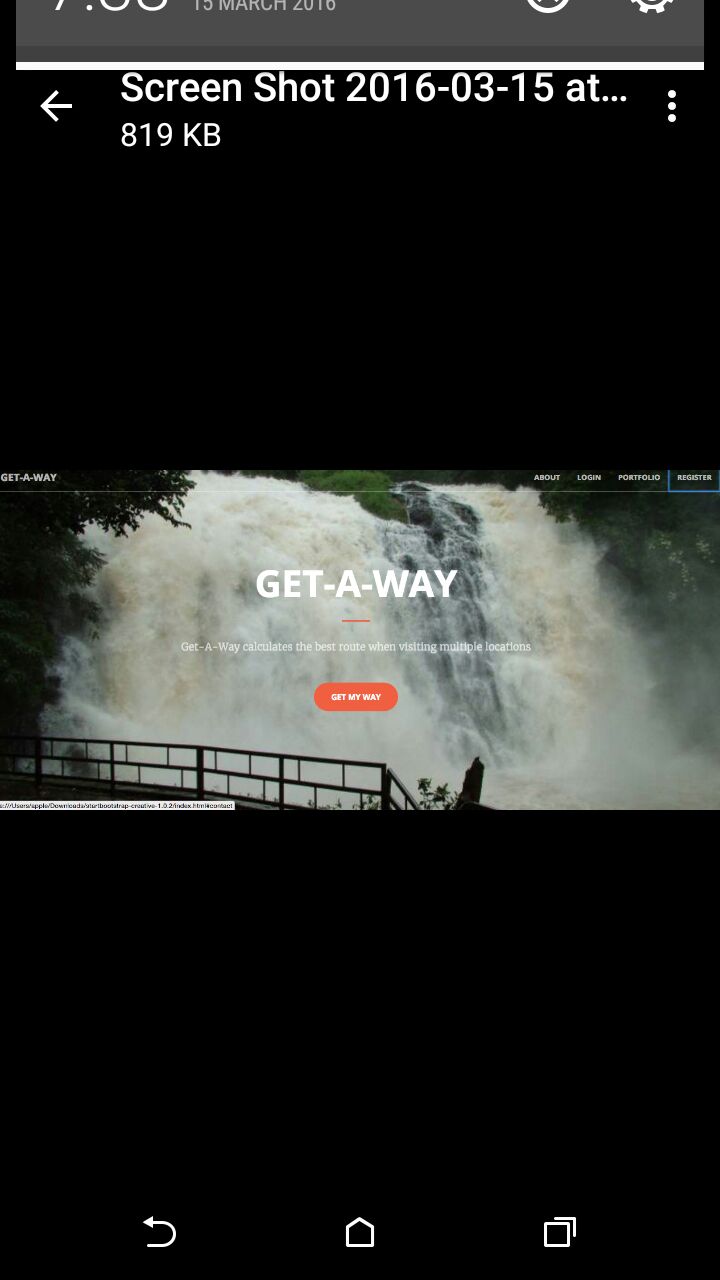
**Optimal Path**

After entering the destinations, an optimal route covering all destinations is generated.

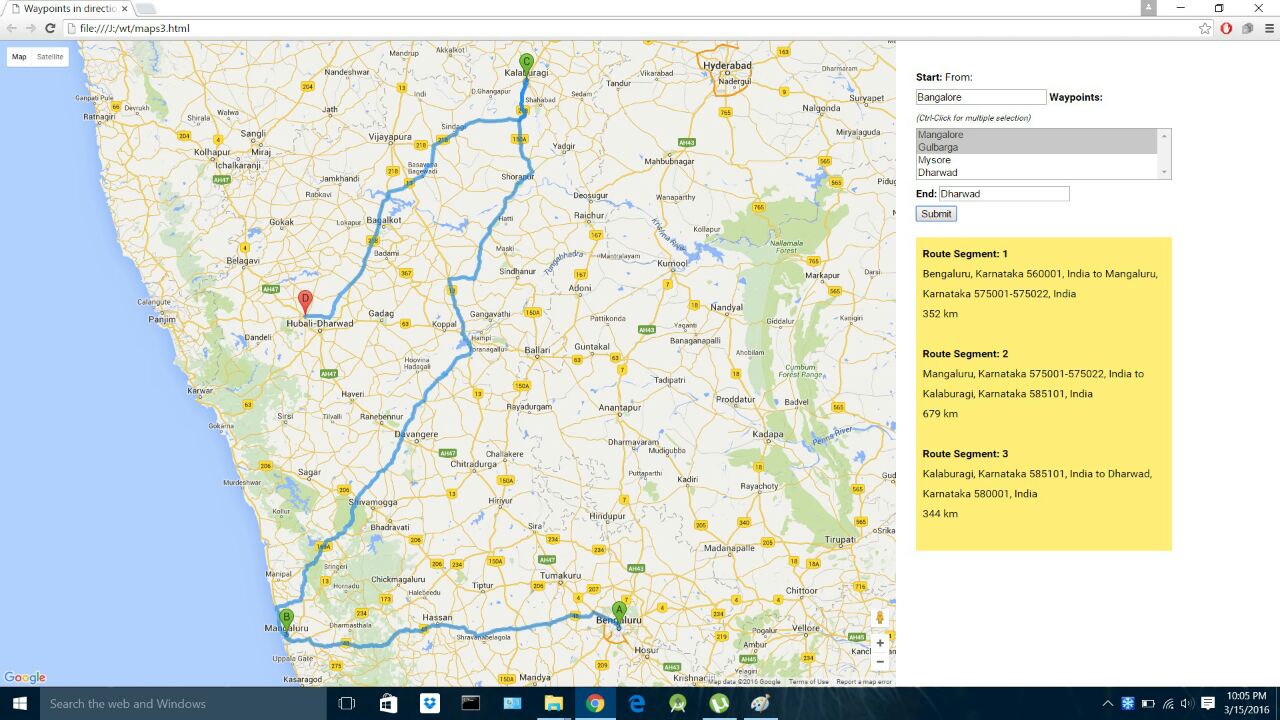


**2. Web Division**

Home Page



**Login Page :**

**Plan a trip**

# **4.0 Detailed Design Approach**

We are using the **Model-View-Presenter** pattern as a part of our project.

The Model-View-Presenter pattern is a perfect fit for our project as it involves the development of a mobile application at its crux.

The role of Views in this pattern are:

* serving as an entry point
* rendering components
* routing user events to the presenter

This allows us to implement your model in the following manner :

**View** - this contains your UI components, and handles events for them.

**Presenter** - this will handle communication between the model and the view, looking at it as a gateway to the model. Meaning, if there exists a complex domain model, and our view only needs a very small subset of this model, the presenters job is to query the model and then update the view. For example, the model contains a lot of information regarding the places to be visited by the user but only a part of that information is to be displayed. This is when the presenter will read the data needed from the model, and update the view accordingly.

**Model** - this is basically the full domain model of our implementation. It is built so as to make testing easier.

Described below are design patterns that have not been employed for this project:

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No** | **Design Pattern** | **Intent of this pattern** | **Rationale for not choosing** |
| 1. | Factory method | Define an interface for creating a single object, but let subclasses decide which class to instantiate. | The factory method is a creational pattern, and is not appropriate for this project because of the structure and client-server architecture that the project employs. |
|  |  |  |  |