

**VISVESVARAYA TECHNOLOGICAL UNIVERSITY
JNANA SANGAMA, BELGAUM - 590014**



A Project Report on

“Personalized Tourism Itinerary Recommender System”

Submitted in partial fulfillment of the requirements for the award of degree of
Computer Science & Engineering

Submitted by:

**CHIRANTH GOPAL K
GOURISH HEBBAR**

**1PI13CS047
1PI13CS062**

Under the guidance of

Guide

**Dr. B NARSING RAO
Professor**

Jan – May 2017



**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
PES INSTITUTE OF TECHNOLOGY
100FT RING ROAD, BSK 3RD STAGE,
BENGALURU – 560085**



PES INSTITUTE OF TECHNOLOGY

(An Autonomous Institute under VTU, Belgaum)
100 Feet Ring Road, BSK- III Stage, Bangalore – 560 085

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CERTIFICATE

Certified that the eighth semester project work titled “*Personalized Tourism Itinerary Recommender System*” is a bonafide work carried out by

Chiranth Gopal K **1PI13CS047**

Gourish Hebbar **1PI13CS062**

in partial fulfillment for the award of Bachelor of Engineering in Computer Science and Engineering of Visvesvaraya Technological University, Belgaum during the academic semester January 2017 – May 2017. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the report deposited in the departmental library. The project report has been approved as it satisfies the academic requirements in respect of project work prescribed for the said Bachelor of Engineering.

Signature of the Guide
Dr. B Narsing Rao

Signature of the HOD
Prof. Nitin V. Pujari

Signature of the Principal
Dr. K S Sridhar

External Viva:

Name of Examiners

Signature with Date

ACKNOWLEDGEMENT

The satisfaction that accompanies with the successful completion of any task would be incomplete without the mention of the people who made it possible, and whose constant guidance and encouragement helped us in completing the project successfully. We consider it a privilege to express gratitude and respect to all those who guided throughout the course of the completion of the project.

It is our pleasure to express our deep sense of thanks and gratitude to our mentor and guide Dr. B Narsing Rao, Professor. Above all his overwhelming attitude, his dedication and keen interest to help us have been solely and mainly responsible for completion of our project. His timely, scholarly advice and scientific approach have helped us a lot to accomplish this task.

We would like to thank our project coordinator Prof. V R Badri Prasad, Associate Professor, for providing us suggestions and technical guidance whenever required which was of great help in the period of work on this project. We sincerely thank him for the encouragement in carrying out this project.

We owe a deep sense of gratitude to Prof. Nitin V Pujari, Professor and Head of Department (Computer Science and Engineering) for his guidance, support and enthusiasm which helped us complete this project.

We would also like to thank our principal, Dr. K S Sridhar for giving us this golden opportunity to embark on this project. It is to him we owe our deepest gratitude.

We would like to thank our peer Mr. Darshan S J for his support and help in guiding us with respect to the various obstacles we faced during the project.

We thank profusely all the staff, parents and friends for their continued support and cooperation throughout our project.

Abstract

When tourists visit any destination they would naturally like to make the most of their visit within the limited time they will typically have. The purpose of this project is to develop an application that will help tourists plan a multi-day itinerary for any destination. Inputs to the application will include the starting (and ending) point within the city, duration of trip (starting and ending dates), for each day, the times between which he wishes to go visiting places, the start/end point of the trip and also the type of places he wishes to see that day. Using this information as well as its database, the application will generate a multi-day itinerary for the tourist that will maximize his expected benefit from visiting that particular destination. The application will generate a map, with the recommended places to visit on each day. Using this, the user can get a good idea as to how he can most effectively use his time to visit the places he likes without having to scout through a bunch of different websites and looking up their locations. The application will be demonstrated using a database for Mysore city.

Index

List of figures.....	7
List of tables.....	8
1 Introduction	9
1.1 Problem definition.....	9
1.2 Generic proposed solution.....	10
1.3 Acknowledgement.....	12
2 Literature Survey.....	13
2.1 References.....	14
3 Software Requirement Specification.....	16
3.1 High level architecture.....	16
3.2 Environment used.....	17
3.2.1 Hardware Requirements.....	17
3.2.2 Software Requirements.....	17
3.2.3 Functional Requirements and Test cases.....	18
3.2.4 Non functional Requirements.....	19
3.2.5 Constraints.....	20
3.2.6 Requirement Traceability Matrix.....	21
3.2.7 Use case scenarios.....	22
4 Schedule.....	24
5 System Design or High Level Design.....	26
5.1 Architectural Diagram.....	26
5.2 Sequence Diagram.....	27
5.3 UI Design.....	28
5.4 Updated RTM.....	35
6 Design.....	36
6.1 Data Flow Diagram.....	36

6.2	Updated RTM.....	39
7	Implementation.....	40
7.1	Pseudo code.....	40
7.2	Codebase Structure.....	43
7.3	Coding Guidelines.....	46
7.4	Sample Code.....	48
7.5	Unit Test Cases.....	52
7.6	Metrics for Unit Test Cases.....	54
7.7	Updated RTM.....	57
8	Testing.....	58
8.1	System Function Spec for the Project.....	58
8.2	Test Environment Used.....	60
8.3	Test Procedure.....	60
8.4	Example Test Result.....	61
8.5	Test Metrics.....	63
8.6	Updated RTM.....	64
9	Results and Discussions.....	65
10	Retrospective.....	69
11	References and Bibliography.....	71
12	User Manual.....	72

List of Figures

Figure No.	Title	Page No.
3.1	Architecture Diagram	16
3.2	User Use Case	22
3.3	Admin Use Case	23
4.1	Gantt Chart	24
5.1	Architecture Diagram	26
5.2	Sequence Diagram	27
5.3	Screen Flow 1	29
5.4	Screen Flow 2	30
5.5	Screen Flow 3	30
5.6	Screen Flow 4	31
5.7	Screen Flow 5	31
5.8	Screen Flow 6	32
5.9	Screen Flow 7	32
5.10	Screen Flow 8	33
5.11	Screen Flow 9	33
5.12	Screen Flow 10	34
5.13	Screen Flow 11	34
6.1	Initial Set Up	36
6.2	Work Flow A	37
6.3	Work Flow B	38
7.1	Defect Find Rate	55
7.2	Defect Fix Rate	56
9.1	Snapshot 1	66
9.2	Snapshot 2	66

List of Tables

Table No.	Title	Page No.
3.1	Requirement Traceability Matrix	21
4.1	Task Breakdown	25
5.1	Updated RTM	35
6.1	Updated RTM	39
7.1	Overall Test	52
7.2	Database Test	53
7.3	Itinerary Test	53
7.4	UI Test	53
7.5	Defects Found	54
7.6	Defects Fixed	55
7.7	Updated RTM	57
8.1	Test Results	61
8.2	Validation Results	61
8.3	Implementation Results	62
8.4	Unexpected Results	62
8.5	Updated RTM	64
9.1	Verification of RTM	68

1. Introduction

Current landscape of tourism travel in India is that if a person wants to visit a place, he can find a few websites which will allow him to search for places around or give recommendations on the places based on his preferences. But say if we have some time constraints, we do not have something that would give us an entire itinerary which would consider our preferences, time constraints, the time to spend at the place etc. and give a map with our itinerary. The user may not know much about the locations of the places, so he can be rest assured that the path we suggest is the most optimal one, thus saving him precious time as well as money. Since the user may also not know the time slot that the place is open to visiting, it is difficult to manually search details about each place and then plan accordingly, or else he may go and visit during wrong hours. So we see that these problems are actually pretty common, but they have not yet been properly addressed. So we set out to solve it, by building a web app.

1.1 Problem definition:

To provide personalized multi-day itinerary for attractions based on user location and preferences, which will include the types of attractions the user is interested in.

1.2 Generic proposed solution:

These are the detailed steps of our solution:

1. Database: A tourism database is built by scraping the data from multiple sites like TripAdvisor, Google, which we collect and create a database using MySQL. Since each site will have its own format of storing data, we are trying to create modules, which are designed specifically for the particular site, but all of the modules give out standard format of data to the database. We are also increasing the level of abstraction, by scraping using code, and making a generic data parser, rather than using tools and manually scrape the data.
2. Recommender engine: rating data is obtained from different sites and includes number of reviews, data from the user's previous places and his ratings, his current preference, his location and all of this will be assigned weights. Then, we take the weighted average of these scores, and calculate the scores for each place. Then based on the proximity of places, the assigned scores and the time slot that the user is free, we will use a variant of Travelling Salesman algorithm where we try to maximize the weights and create an itinerary, which can be edited by the user.
3. Front end: The UI will consist of a web app, with login to authenticate the user. Then, we take the following inputs from the user:
 - Dates of travel
 - Time of availability for the day
 - Start/end location of travel
 - Category of places

Then, we will take inputs and generate a few recommendations, and using those we generate an itinerary, which will be shown on the map, with the list of places on the side. The places not included in the itinerary are also shown, so the user can add/remove places from the itinerary and new itineraries will be generated on the fly.

We can say the project would be complete, when our web app is ready, which is by April 14 with above functionalities and features included.

1.3 Acknowledgement



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

PES INSTITUTE OF TECHNOLOGY

(An Autonomous Institute under VTU, Belgaum)

100 Feet Ring Road, BSK- III Stage, Bangalore – 560 085

Project ID: **PW056**

Project Title: **‘Personalized Tourism Itinerary Recommender System’**

Project Team:

1PI13CS047

Chiranth Gopal K

1PI13CS062

Gourish Hebbar

This project report was submitted for review on 23-04-2017. I acknowledge that the project team has implemented all recommended changes in the project report.

Guide signature with date:

Guide Name: Dr. B Narsing Rao

2. Literature survey

Most mainstream tourism web sites that exist today, have certain main functionalities. Below, we mention those main things that they are capable of doing:

- **MakeMyTrip, TripAdvisor:**

These sites offer a lot of functionalities, like being able to book hotels, air tickets and restaurants. The differences are that, MakeMyTrip is more of travel oriented site, with more options to book travel options like bus and trains, whereas TripAdvisor contains exhaustive knowledge related to places. TripAdvisor can be used to search for places nearby, based on preferences of user, but it is not possible to create an itinerary of places to visit.

- **Google Now:**

Google generates recommendations automatically on places nearby once we are in a given location. We can also search based on the type of places we want to see, and it will show places sorted in increasing order of distances from the our current location. There is no option to create an itinerary of places that we would want to see

- **Inspirock:**

Inspirock takes a place as input from the user, and generates an itinerary of places, with option to add or remove places. This is the most similar existing solution, but there are however certain differences, them being:

There is no option to set the start and end time of the trip for the day

User cannot extend the time which is fixed at 6:00 PM in the evening, so rules out places to visit after sunset

The start and end points are different, and cannot be changed by the user

Once an itinerary is created, there is no option to get recommendations to restaurants closer to the place which the user is currently in if the users want to have a break for eating

2.1 References:

1. Location-Aware Recommendation Systems by María del Carmen Rodríguez-Hernández, Sergio Ilarri, Raquel Trillo-Lado, Ramón Hermoso - Article 2015

This paper provides a survey on location-aware recommender systems in scenarios involving mobile computing. It describes the fundamentals of recommendation systems, shows the most relevant existing approaches for location aware systems. Then it talks about the current applications of location aware recommender systems in different domains. From this paper, we got an idea of how recommender systems are used in various cases, and how it applies in our case of tourism. This prompted us to think in terms of what we could do and how we could leverage the abundance of location data to build a good recommender system.

2. The Use of Machine Learning Algorithms in Recommender Systems: A Systematic Review by Ivens Portugal, Paulo Alencar, Donald Cowan - arXiv 2015

This paper presents a review of the literature that analyzes the use of machine learning algorithms in recommender systems and identifies the research opportunities in the field of software engineering research. Along with comparing the use of algorithms, it also compares the

usage of recommender systems in different domains. From this, we learnt that among the papers that the authors have reviewed on recommender systems, the two most used algorithms were Bayesian and Decision trees. The authors think that this is because of the simplicity of the algorithms, i.e, since they are less computationally expensive. Another important thing that we found was that among the papers, tourism was among the last in the list of domains ordered by the number of papers on them. So it was an added incentive for us that we were choosing a field where there is not much research done before.

3. Web application for recommending personalised mobile tourist routes by D. Gavalas, M. Kenteris, C. Konstantopoulos, G. Pantizou - IET Softw., 2012

This paper deals with the problem of creating personalized recommendations for daily sightseeing itineraries for tourists visiting places. Their approach considers a few selected places of interest that a traveler would potentially want to visit and derives a near optimal itinerary solution for each day of visit and the places of potential interest are selected based on stated or implied preferences. Their method enables planning of customized, each day based, personalized tourist itineraries considering user preferences, time available for visiting sights on a daily basis, opening days of sights and average visiting times, using heuristics. This was the most relevant paper for our reference, as the concept of creating itinerary is same, and also many of the features. But they have not given their algorithm, so we will have to develop our own.

3. System Requirement Specification:

3.1 High level architecture diagram

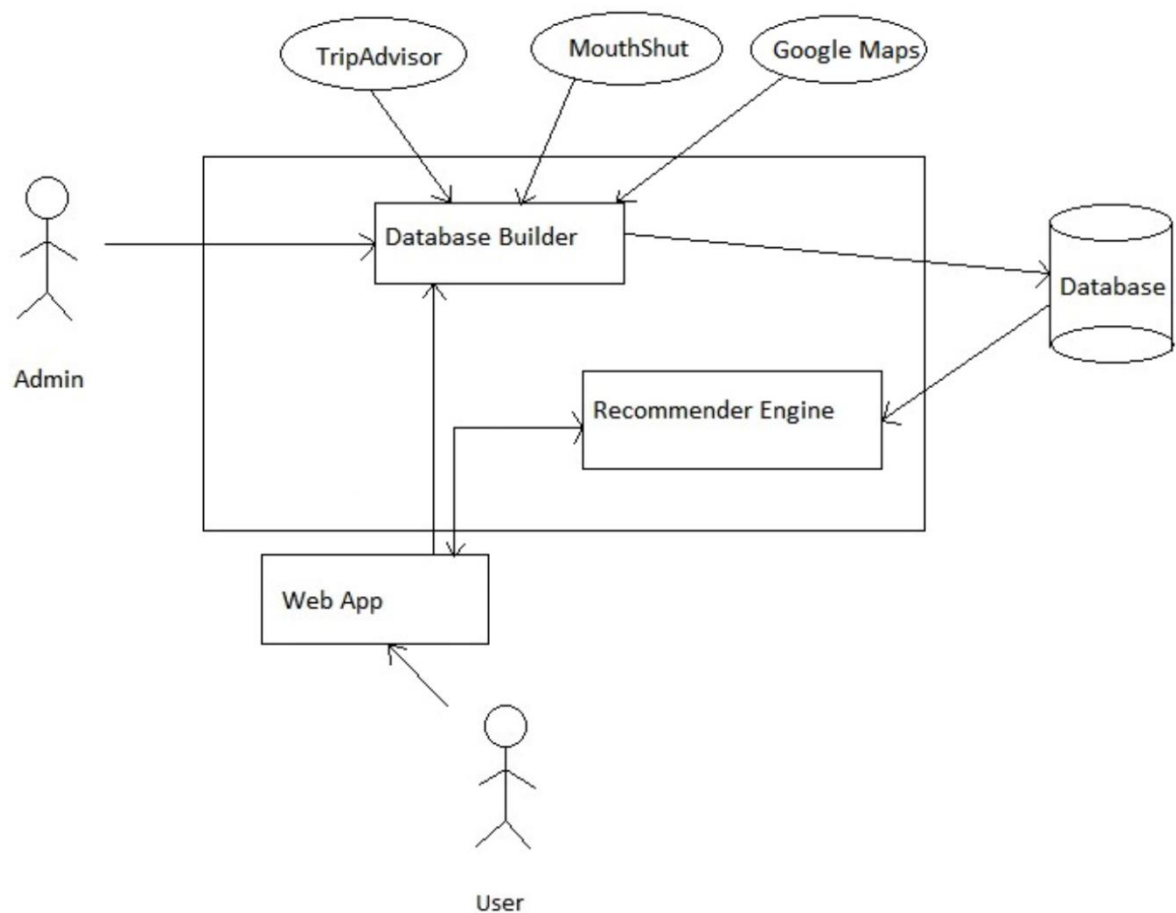


Fig 3.1 Architecture diagram

3.2.1 Hardware Requirements

- Server to host website and tourism database
- Client device - laptop, phone

3.2.2 Software Requirements

Programming Languages:

- Front end:

- Javascript, HTML, CSS, Bootstrap

Our familiarity with Javascript and HTML, and their robust usage around internet, led to us deciding on them. Bootstrap is chosen for a more responsive experience

- Back end:

- Server side: PHP

PHP was chosen as the backend language of choice

- Data extraction:

- Python

Python has vast amounts of support and libraries. Some of the libraries that we are using are mentioned below:

- beautifulsoup: Data Extraction
 - selenium: Parsing web sites
 - googleplaces: Getting latitude longitude data from place name
 - pickle: Storing and retrieving intermediate data
 - MySQLdb: To connect with MySQL database

- Database:

- MySQL

3.2.3 Functional Requirements and Test Cases

- **FR1 Set trip dates**
 - Description: The dates on which the user intends to go on the trip.
 - Evaluation-T1: Manual. We have to enter dates of the trip.

- **FR2 Set location**
 - Description: The place in which user chooses to go around sightseeing other places.
 - Evaluation-T2: Manual. Location is chosen by us, so have to manually choose in the map.

- **FR3 Set category preferences**
 - Description: Set the type of places the user wants to see.
 - Evaluation-T3: Manual. Category should be chosen manually.

- **FR4 Set schedule**
 - Description: Set the start and end time for the day.
 - Evaluation-T4: Manual. Time slots need to be chosen manually.

- **FR5 Generate itinerary**
 - Description: Create the itinerary on google maps and show user a list of the recommended places.
 - Evaluation-T5: Manual. See if the generated itinerary is properly displayed on the map

- **FR6 Add or remove places from recommended list**

- Description: Give user option to manually add or remove places from the itinerary generated.
 - Evaluation-T6: Manual. We have to test by manually adding and removing places.
-
- UI1 Data input page
 - UI2 Map generation/places recommender page
 - UI3 Itinerary page

3.2.4 Non Functional Requirements

- NF1 Quick data access
 - Description: When retrieving the data from the database, fast retrieval helps in generating recommendations quicker
- NF2 Reliability - Database has ACID properties (MySQL)
 - Description: MySQL comes with ACID properties of Atomicity, Concurrency, Isolation and Durability
- NF3 Optimal path generated
 - Description: Since the recommendations generated vary from user to user, the path in the itinerary generated will also vary. But we will be generating the optimal path among those recommendations

3.2.5 Constraints

- Data is not available. So we have to scrape the websites and extract data from them
- Some sites have dynamic content generation, i.e., the elements on the site are created by javascript functions on some action by the user. So we need to use a web automation tool to emulate the actions as it would be made by the user, and then grab the data
- As the recommendations that are generated will be different for different users based on their preferences and location, the generated itinerary will be different, thus making it harder for evaluation
- Generating shortest path is a NP-hard problem and hence it would take time to generate itineraries on the fly based on user input

3.2.6 Requirement Traceability Matrix (RTM)

Requirement ID	Requirement Type	Requirement Description	Test Scenario	Results	Comments
FR1	Functional	Setting dates for trip			
FR2	Functional	Setting location of trip			
FR3	Functional	Setting category preferences			
FR4	Functional	Setting time slots for the day			
FR5	Functional	Generating itinerary			
FR6	Functional	Add/remove places from list			
UI1	Functional	Data input page			
UI2	Functional	Map generation page			
UI3	Functional	Itinerary page			
NF1	Non Functional	Quick data access			
NF2	Non Functional	Reliability			
NF3	Non Functional	Optimal path generation			

Table 3.1 Requirement Traceability Matrix

3.2.7 Use case scenarios

- User

The following diagram denotes the high level use cases of the user's interaction with the web app.

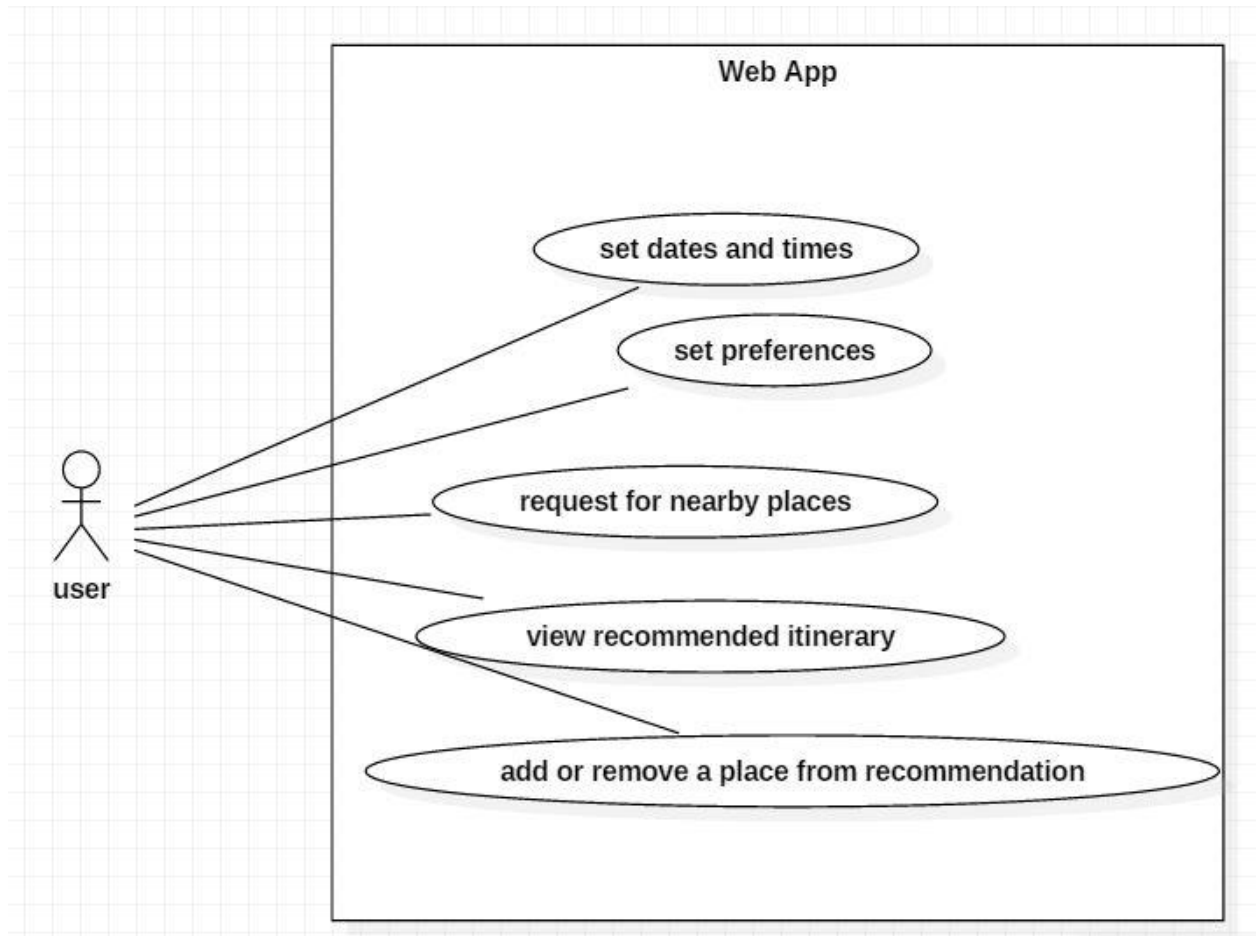


Fig 3.2 User Use Case

- Admin

The following diagram denotes the high level use cases of the admin's interaction with the web app's database.

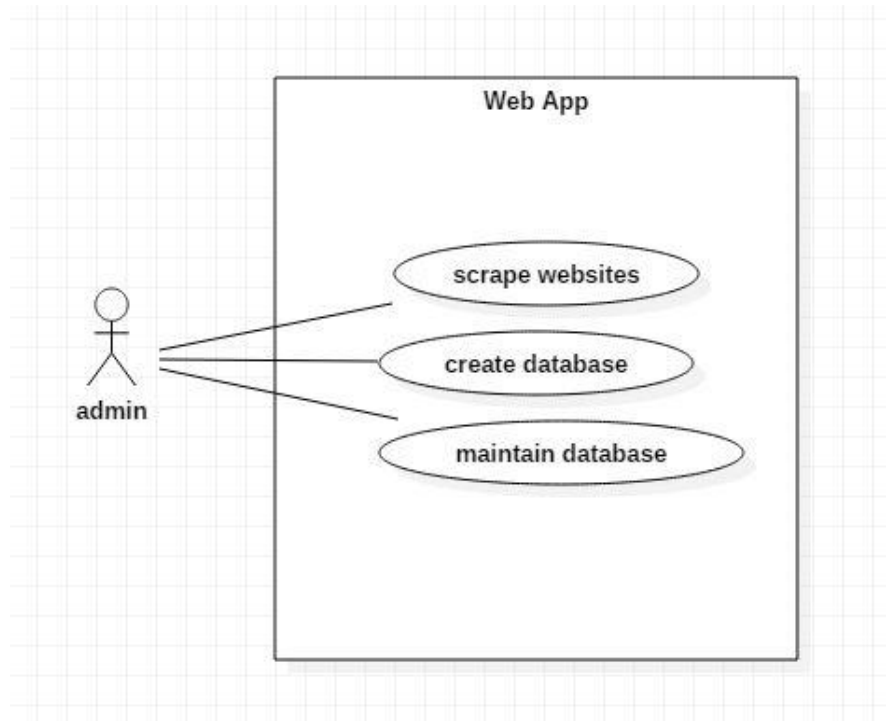


Fig 3.3 Admin Use Case

4. Schedule

Gantt Chart

Attached below is the Gantt Chart for our project.

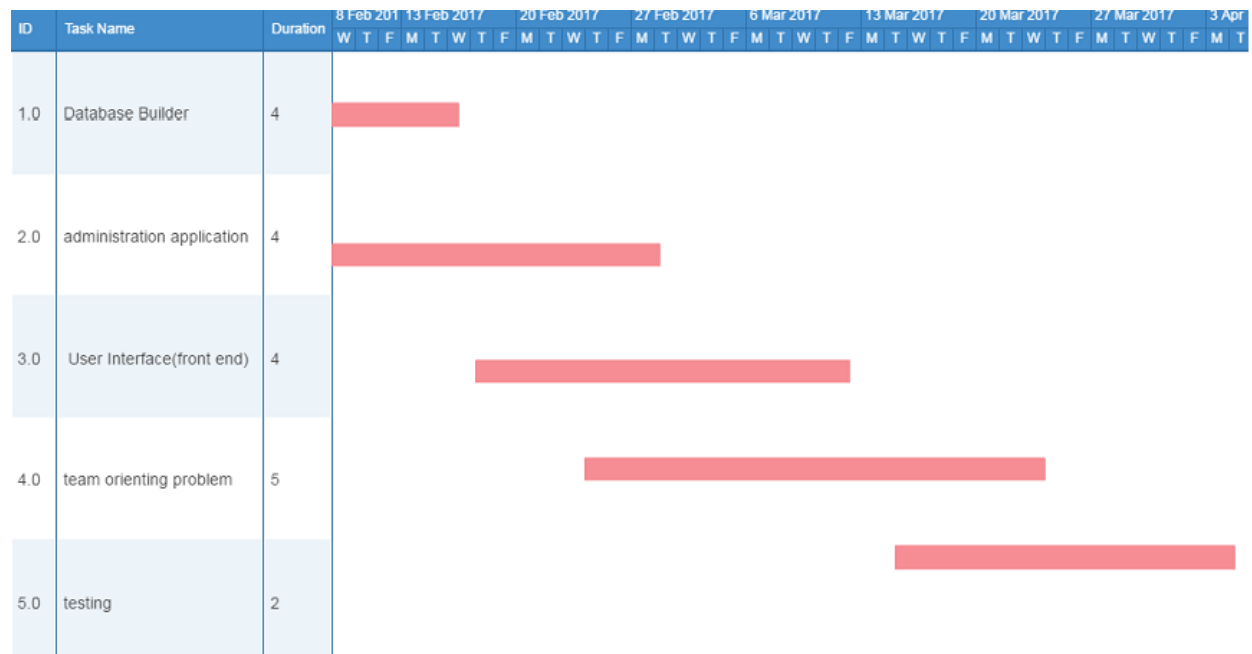


Fig 4.1 Gantt Chart

Task breakdown table

The tasks mentioned in Gantt chart are further broken down in terms of tasks and the dates on which they were expected to be completed on.

Task ID	Task Name	Due Date
1	Scraping	28-01-2017
2	Data extraction	28-01-2017
3	Google Maps API	28-01-2017
4	RDBMS Design	28-01-2017
5	RDBMS Queries	28-01-2017
6	Front end bootstrap	28-01-2017
7	Back end php	28-01-2017
8	SRS	04-02-2017
9	Map generation	04-02-2017
10	URL and rating data	04-02-2017
11	Choose location	11-02-2017
12	Recommended time spent	11-02-2017
13	Multi category selection UI	11-02-2017
14	Query and results backend	11-02-2017
15	Split sights into multi categories	18-02-2017
16	Map marker dynamic color	18-02-2017
17	Range slider	25-02-2017
18	User inputs	04-03-2017
19	Place selection	04-03-2017
20	Rating normalization	04-03-2017
21	Map itinerary	04-03-2017
22	Prim's algorithm	11-03-2017
23	Edge cost calculation	11-03-2017
24	Caching matrix data	08-04-2017
25	Time divided place itinerary	18-03-2017
26	UI Overhaul	08-04-2017
27	Multi day schedule	15-04-2017

Table 4.1 Task Breakdown

5. System Design

5.1 Architectural Diagram

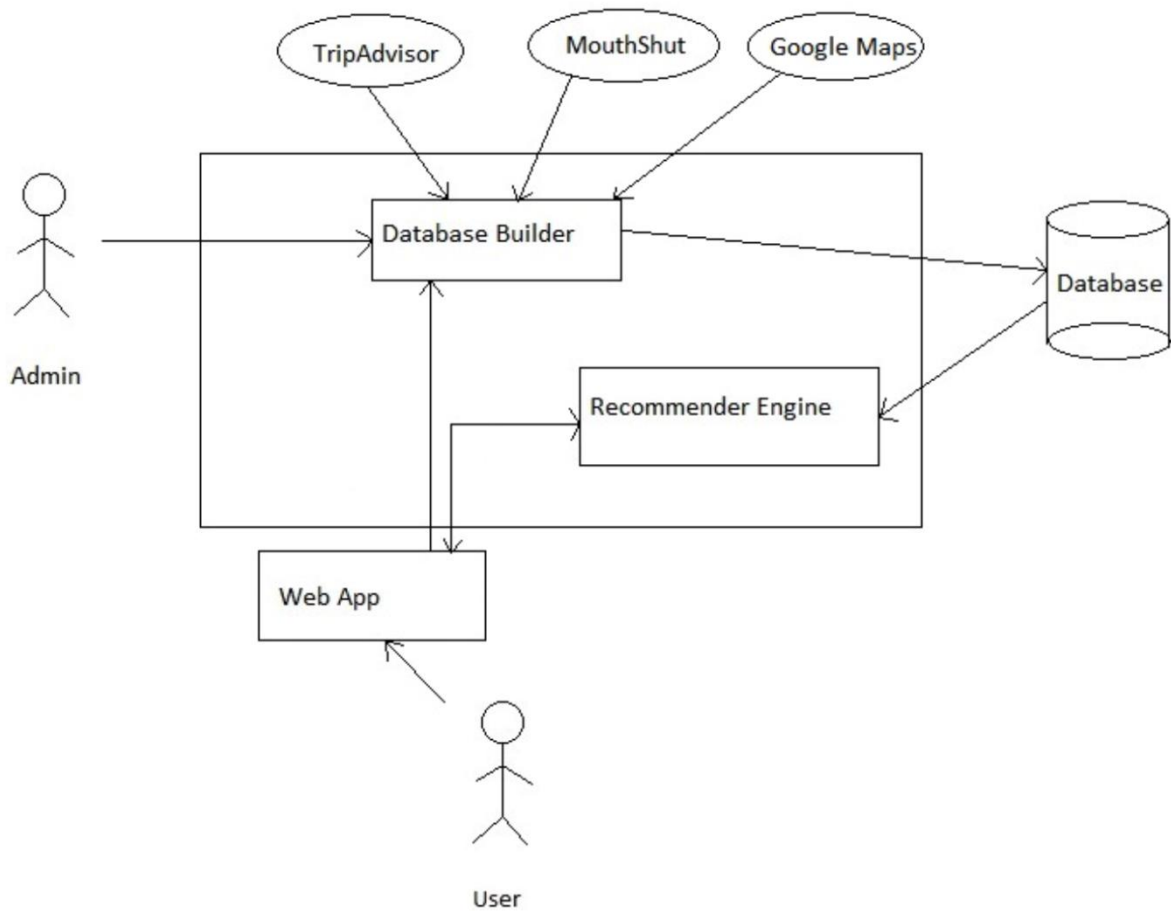


Fig 5.1 Architectural Diagram

5.2 Sequence Diagram

Collaboration1::Interaction1::SequenceDiagram1

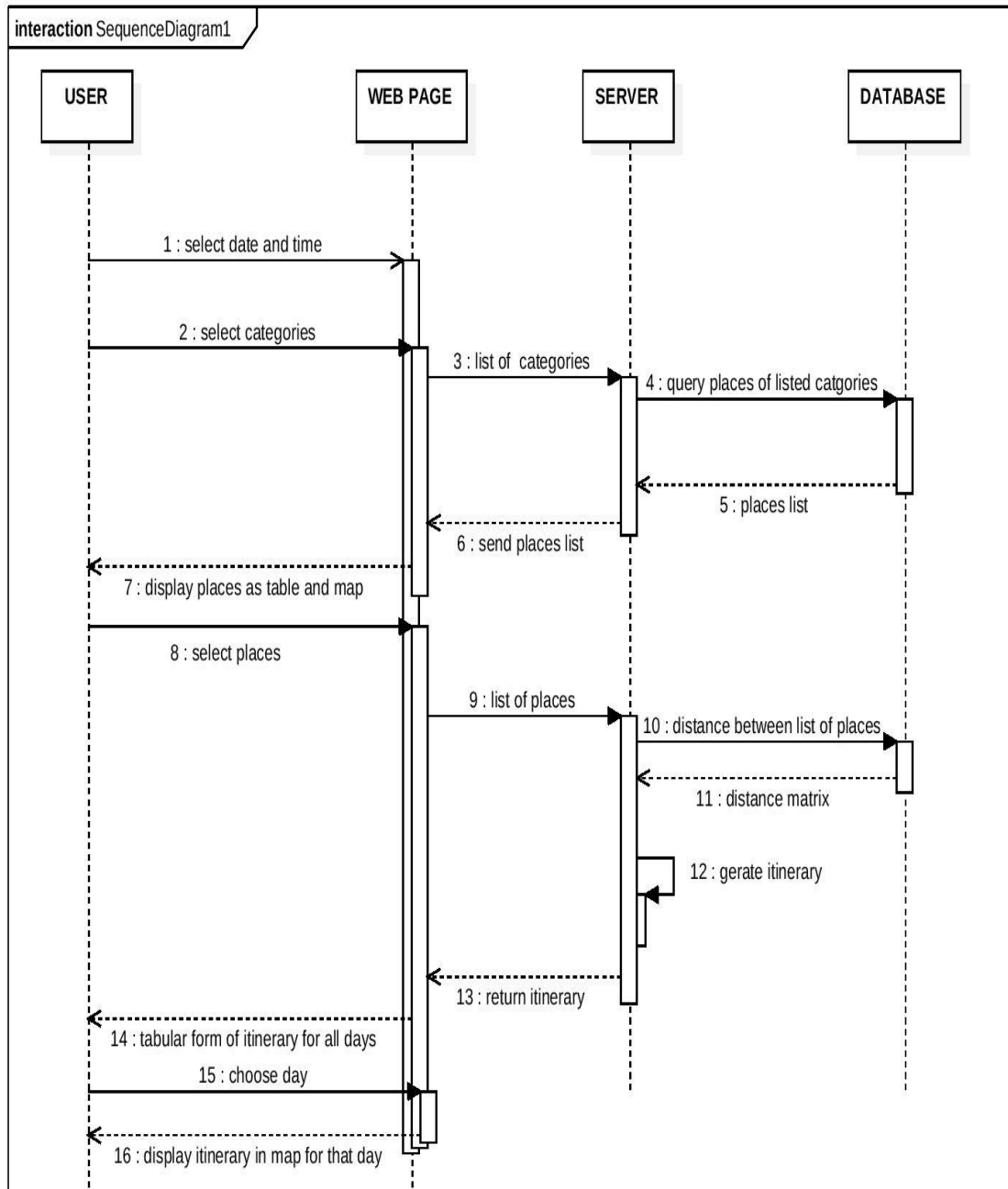


Fig 5.2 Sequence Diagram

5.3 UI Design

Description:

The features of the UI are described below:

- The main page contains a map from Google's Maps API
- The map is part of a tab, and there are two other tabs
- The second tab consists of
 - An input bar which opens a calendar to select the start date of the trip
 - An input bar which opens a calendar to select the end date of the trip
 - A time bar which opens a clock to select the start time of the trip
 - A time bar which opens a clock to select the end time of the trip
- The third tab consists of an empty page, which will get populated with the itinerary of places and times, for each day the user chooses.
- There is a sliding pane of categories of different places on the left side of the page, which can be hidden as and when the user chooses.
- This pane consists of each category, and a slider to its right to select/deselect the category
- The categories when selected give the results in the form of a table which is visible under all the tabs.
- Above the map, there is a dropdown to choose the day for which the itinerary is to be displayed.

UI screen flows:

- The user has the choice to either select the categories of places, or go to the second tab and set the requisite trip dates and times.
- The places displayed on in the table based on the categories are shown in the map.
- Selecting a particular place from the table will be shown on the map and in a highlighted marker.
- If user has not entered the dates and times, then he cannot generate an itinerary of places.
- If he has entered the times and dates, then he can proceed to generate itinerary.
- The generated itinerary will be shown on both the map in the first tab, as well as a table in the third tab.
- User can choose the particular day from the dropdown and he can see the itinerary change in both the map and table.

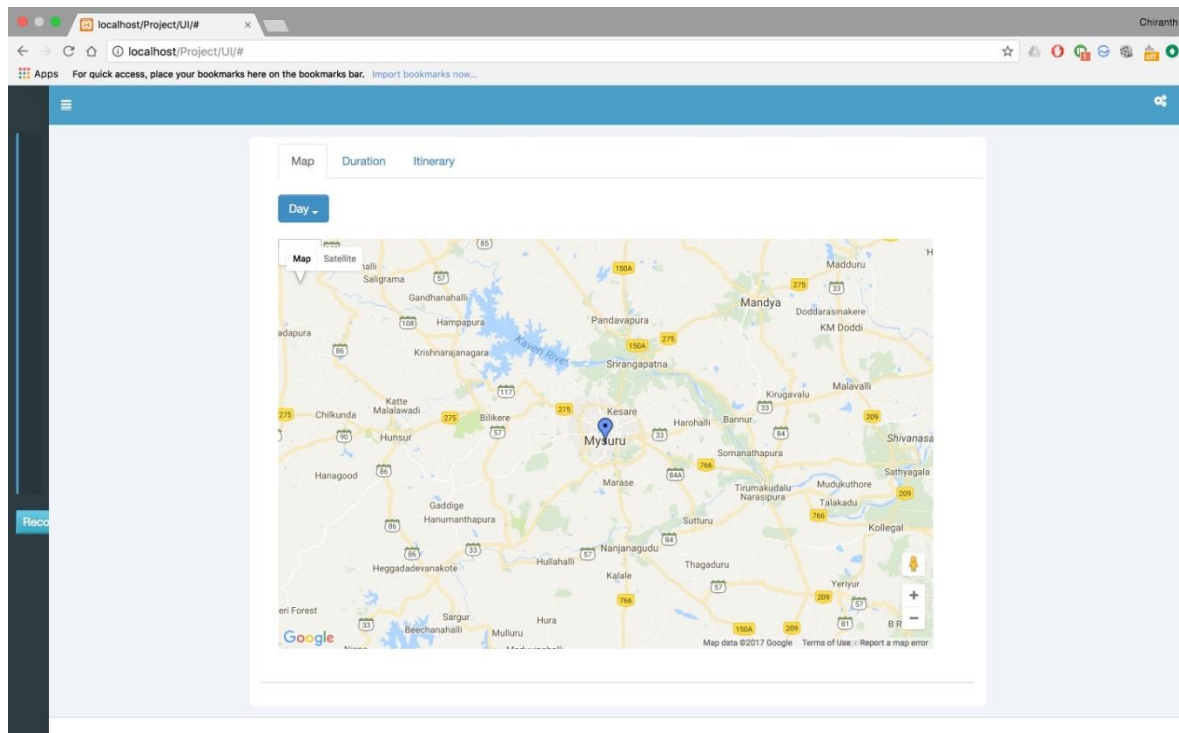


Fig 5.3 Screen Flow 1

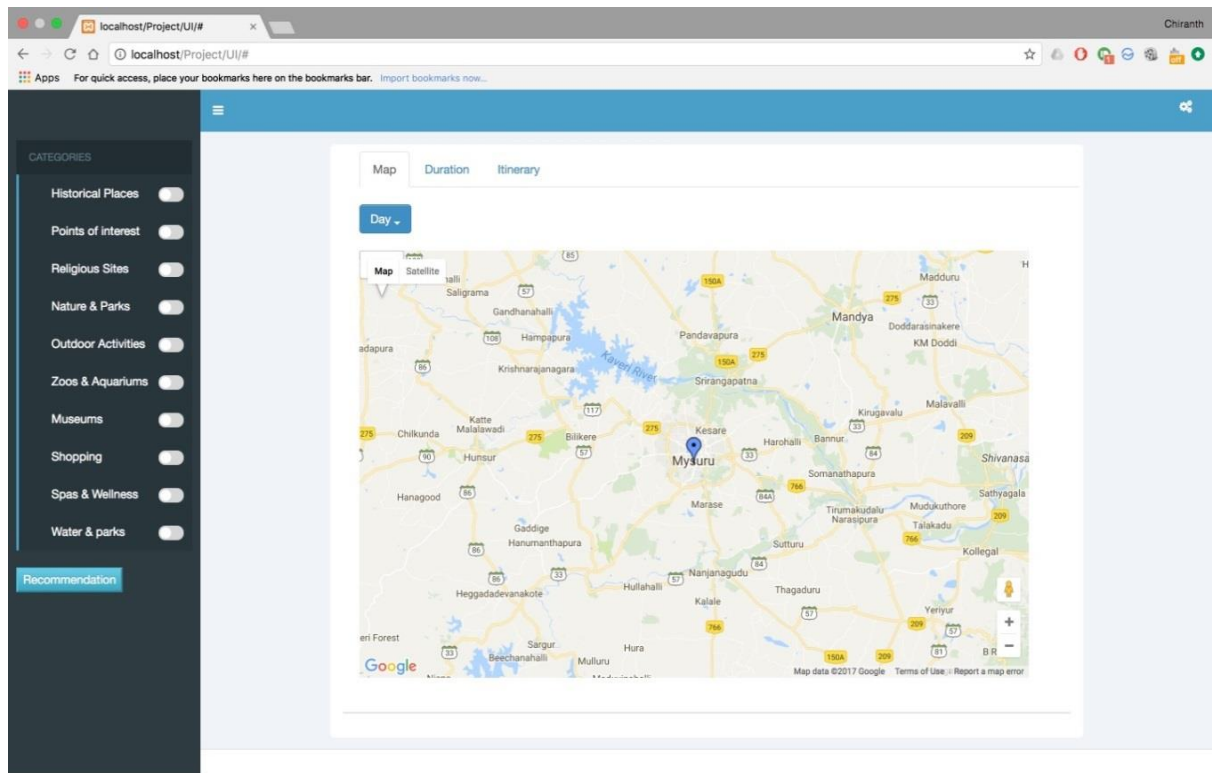


Fig 5.4 Screen Flow 2

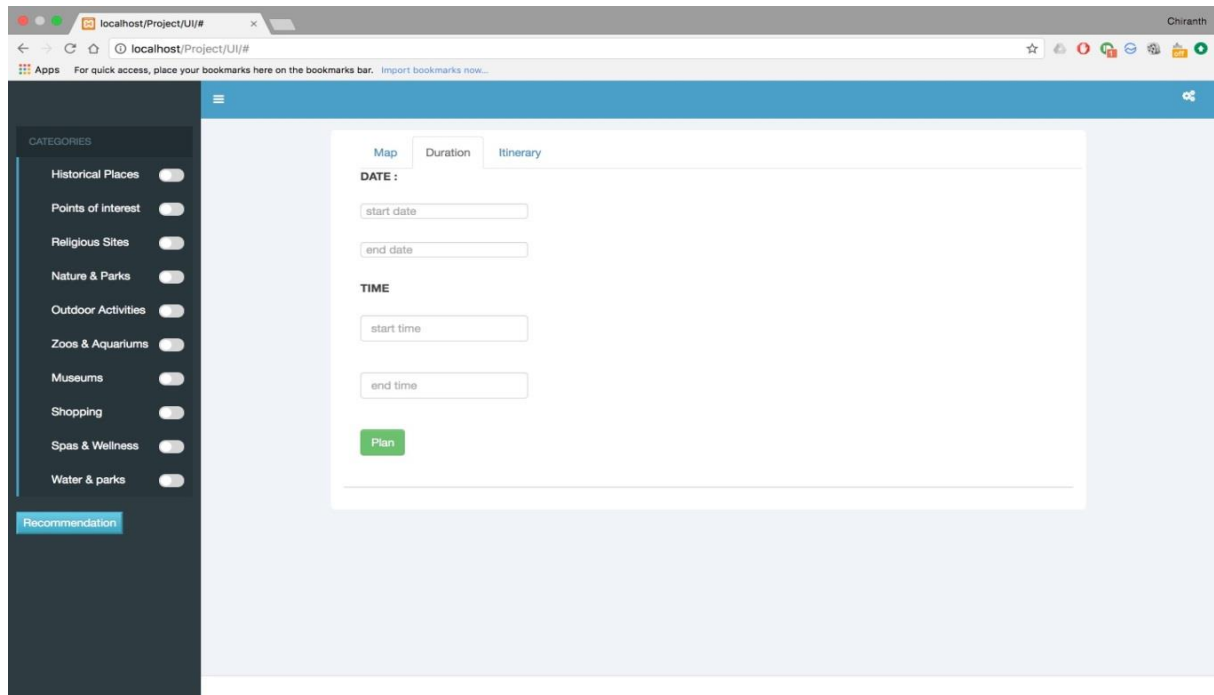


Fig 5.5 Screen Flow 3

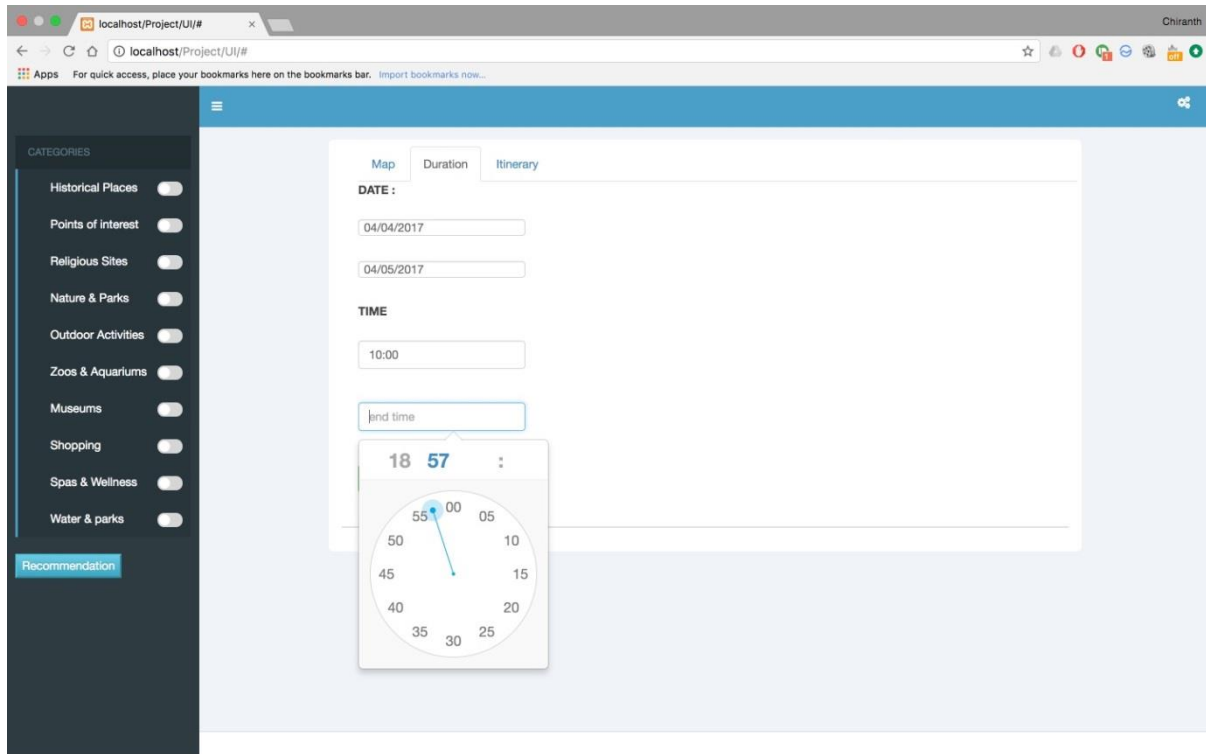
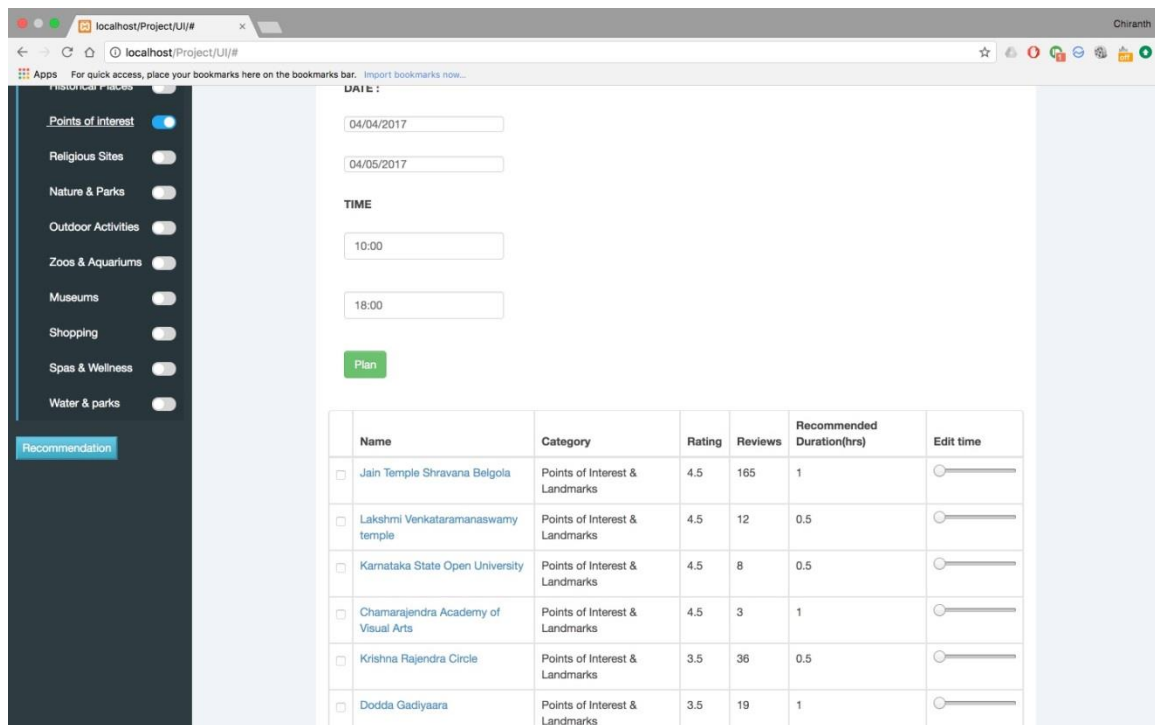


Fig 5.6 Screen Flow 4



	Name	Category	Rating	Reviews	Recommended Duration(hrs)	Edit time
<input type="checkbox"/>	Jain Temple Shravana Belgola	Points of Interest & Landmarks	4.5	165	1	<input type="range"/>
<input type="checkbox"/>	Lakshmi Venkataramanaswamy temple	Points of Interest & Landmarks	4.5	12	0.5	<input type="range"/>
<input type="checkbox"/>	Karnataka State Open University	Points of Interest & Landmarks	4.5	8	0.5	<input type="range"/>
<input type="checkbox"/>	Chamarajendra Academy of Visual Arts	Points of Interest & Landmarks	4.5	3	1	<input type="range"/>
<input type="checkbox"/>	Krishna Rajendra Circle	Points of Interest & Landmarks	3.5	36	0.5	<input type="range"/>
<input type="checkbox"/>	Dodda Gadiyaara	Points of Interest & Landmarks	3.5	19	1	<input type="range"/>

Fig 5.7 Screen Flow 5

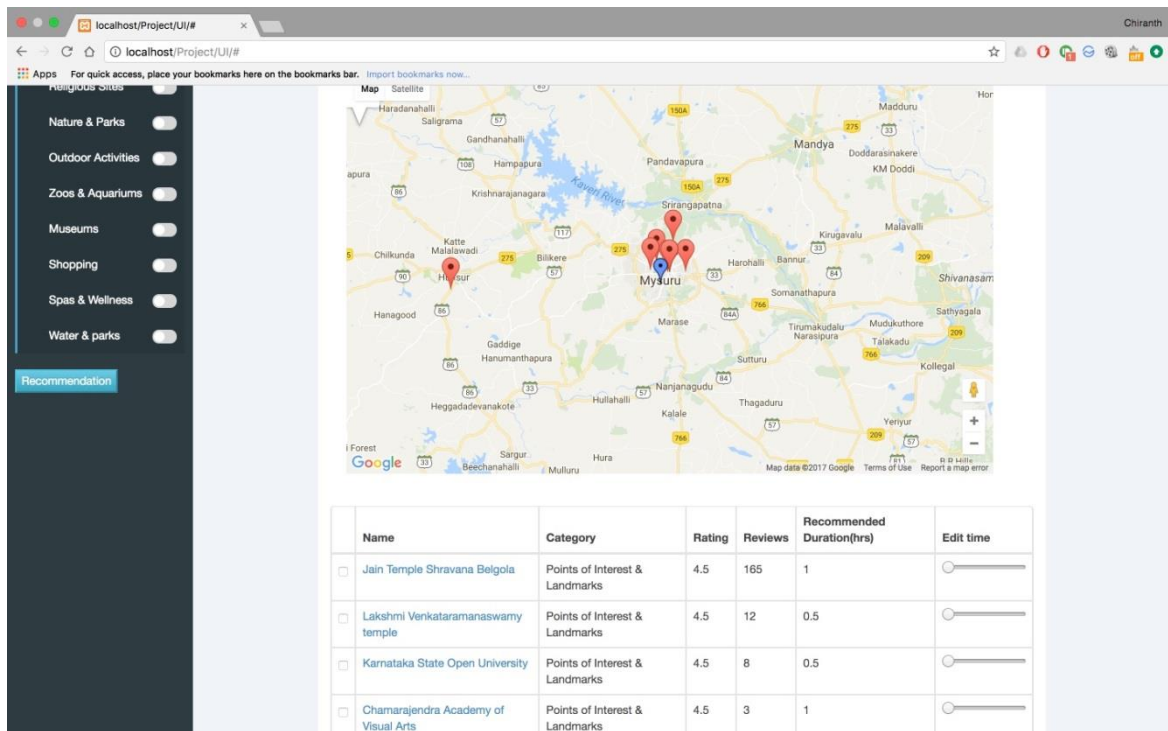


Fig 5.8 Screen Flow 6

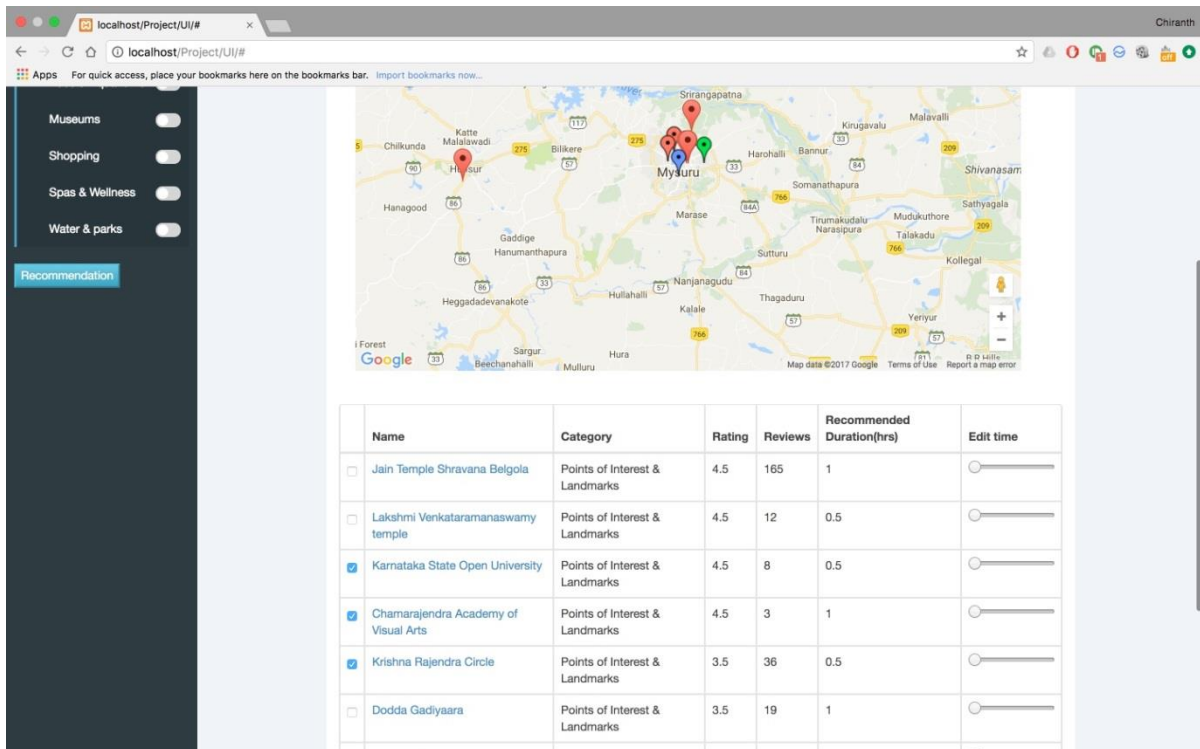


Fig 5.9 Screen Flow 7

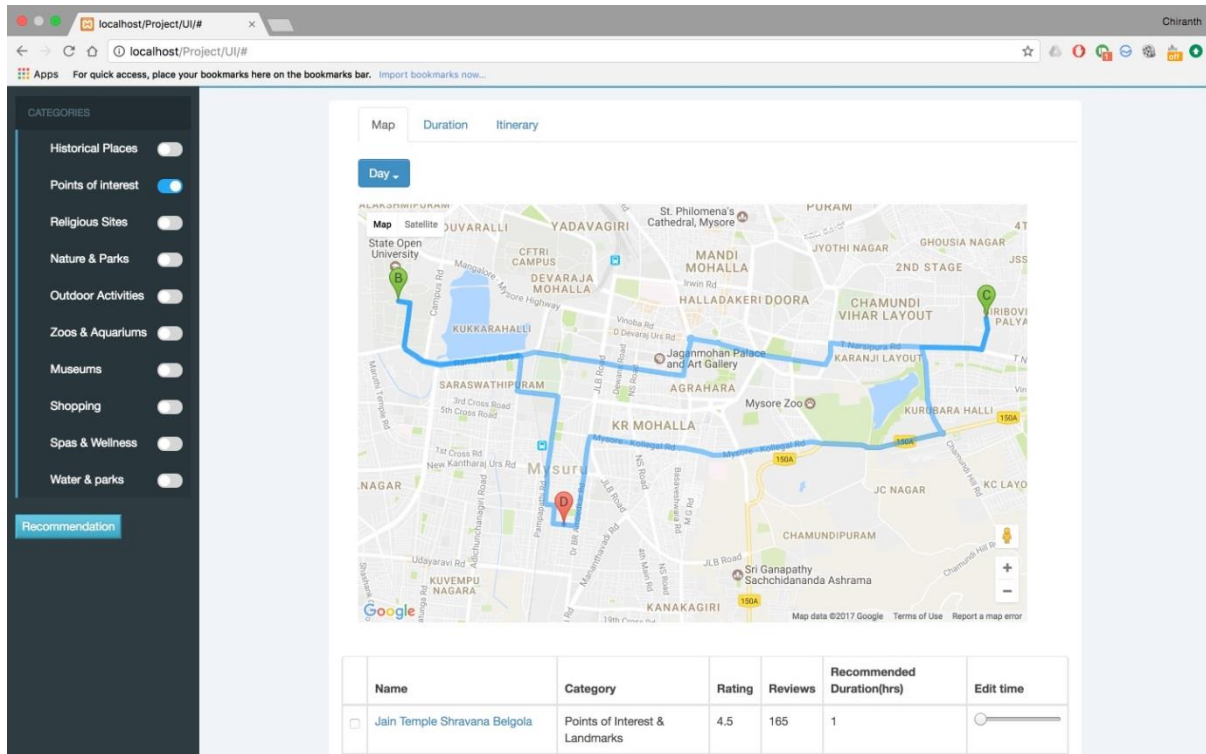


Fig 5.10 Screen Flow 8

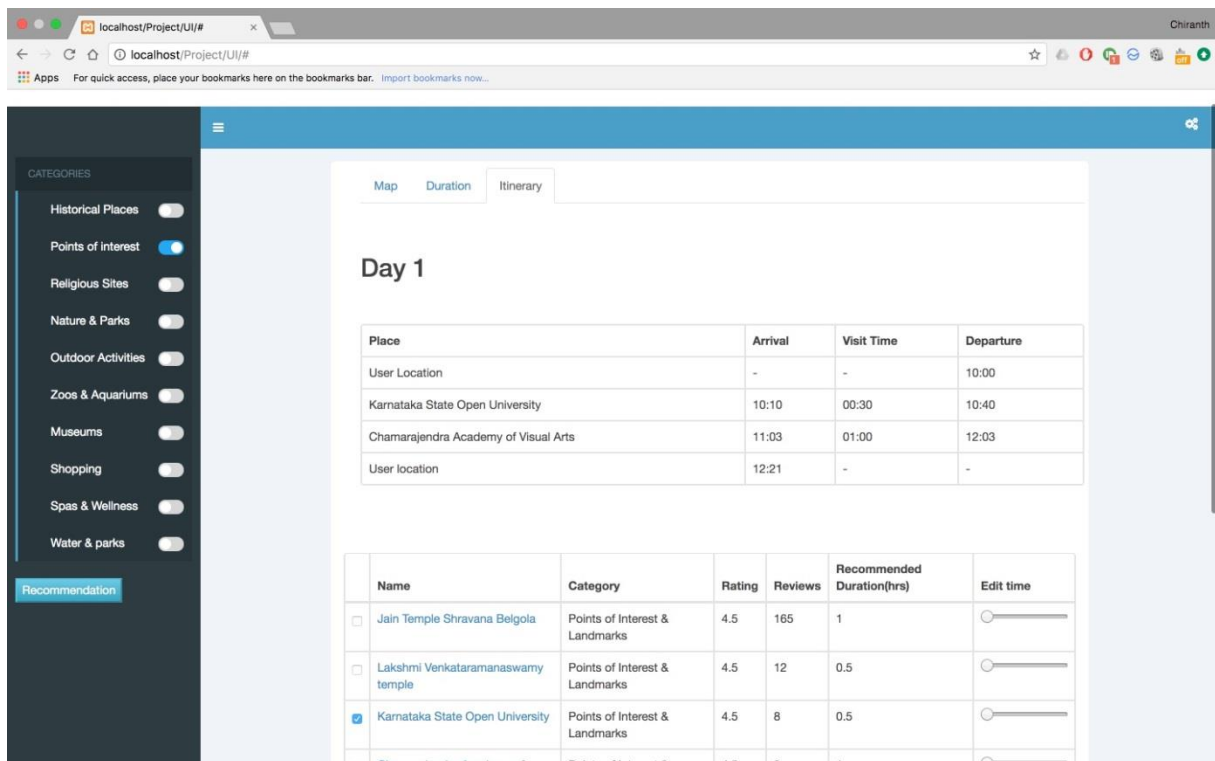


Fig 5.11 Screen Flow 9

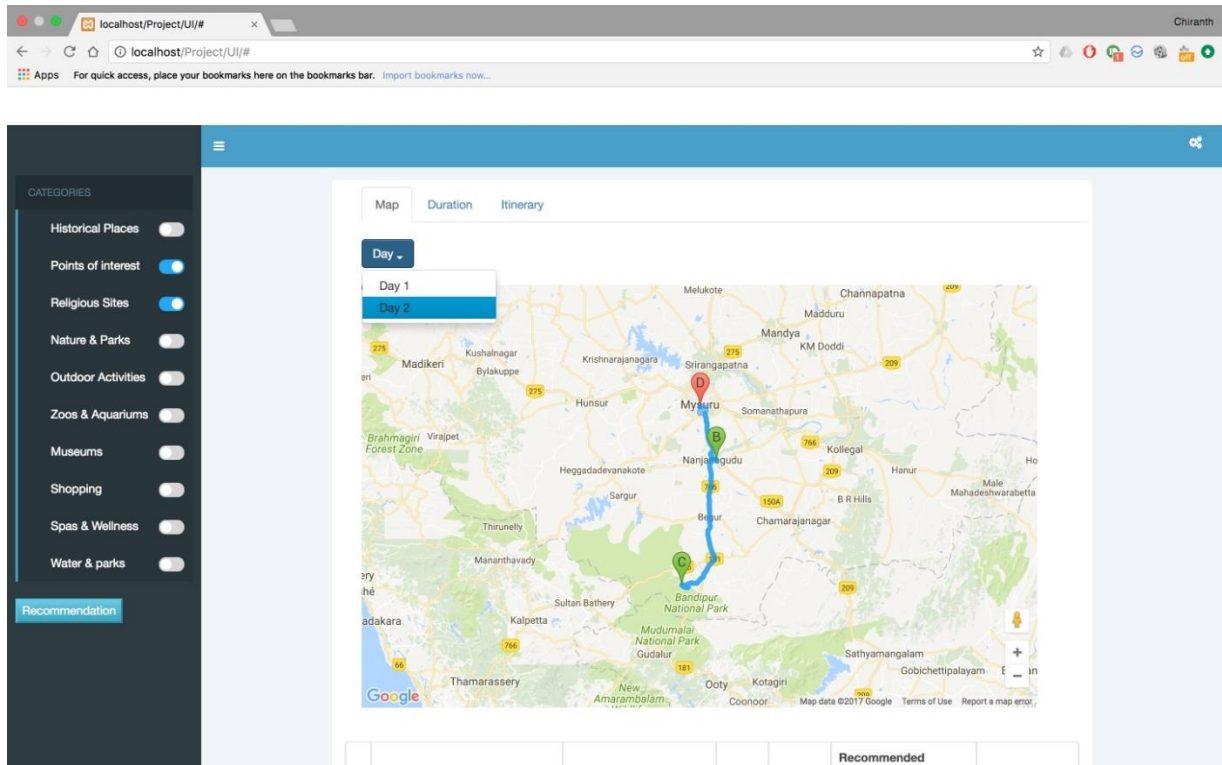


Fig 5.12 Screen Flow 10

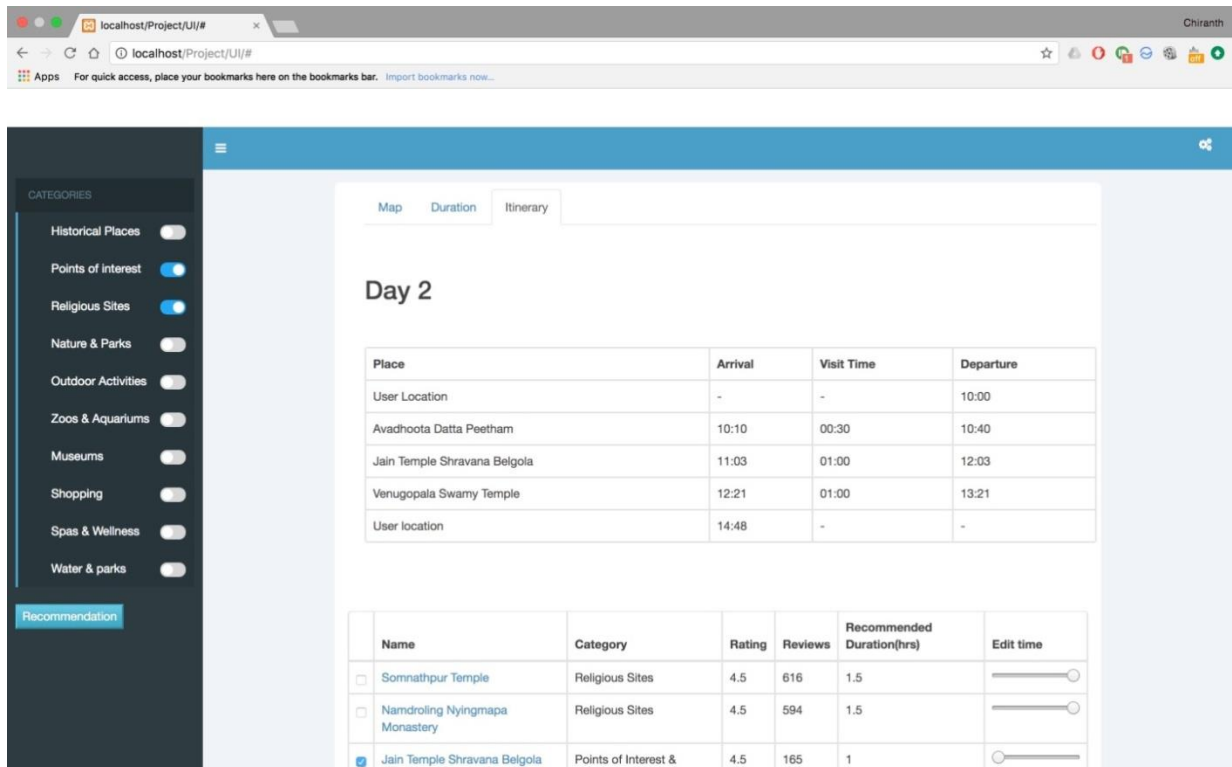


Fig 5.13 Screen Flow 11

5.4 Updated RTM

Requirement ID	Requirement Type	Requirement Description	Test Scenario	Results	Comments
FR1	Functional	Setting dates for trip			
FR2	Functional	Setting location of trip			
FR3	Functional	Setting category preferences			
FR4	Functional	Setting time slots for the day			
FR5	Functional	Generating itinerary			
FR6	Functional	Add/remove places from list			
UI1	Functional	Data input page	Manual	Pass	None
UI2	Functional	Map generation page	Manual	Pass	None
UI3	Functional	Itinerary page	Manual	Pass	None
NF1	Non Functional	Quick data access			
NF2	Non Functional	Reliability			
NF3	Non Functional	Optimal path generation			

Table 5.1 Updated RTM

6 Design

6.1 Data Flow Diagram

Initial set up

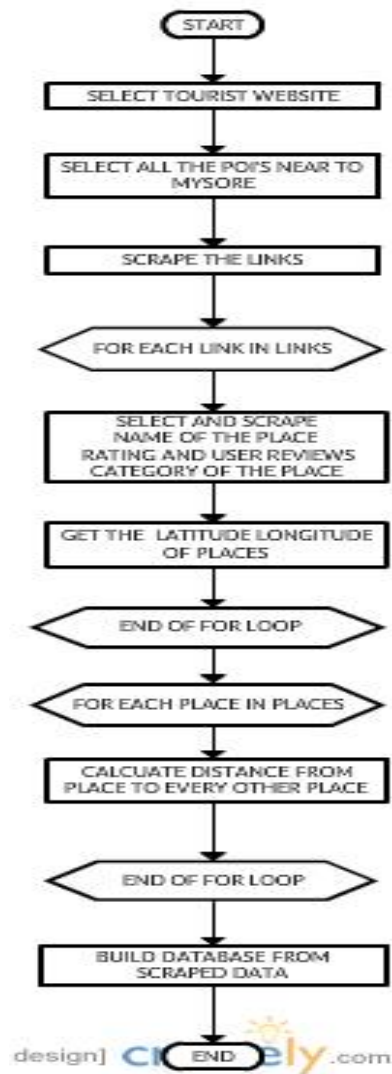


Fig 6.1 Initial Set up

Work flow

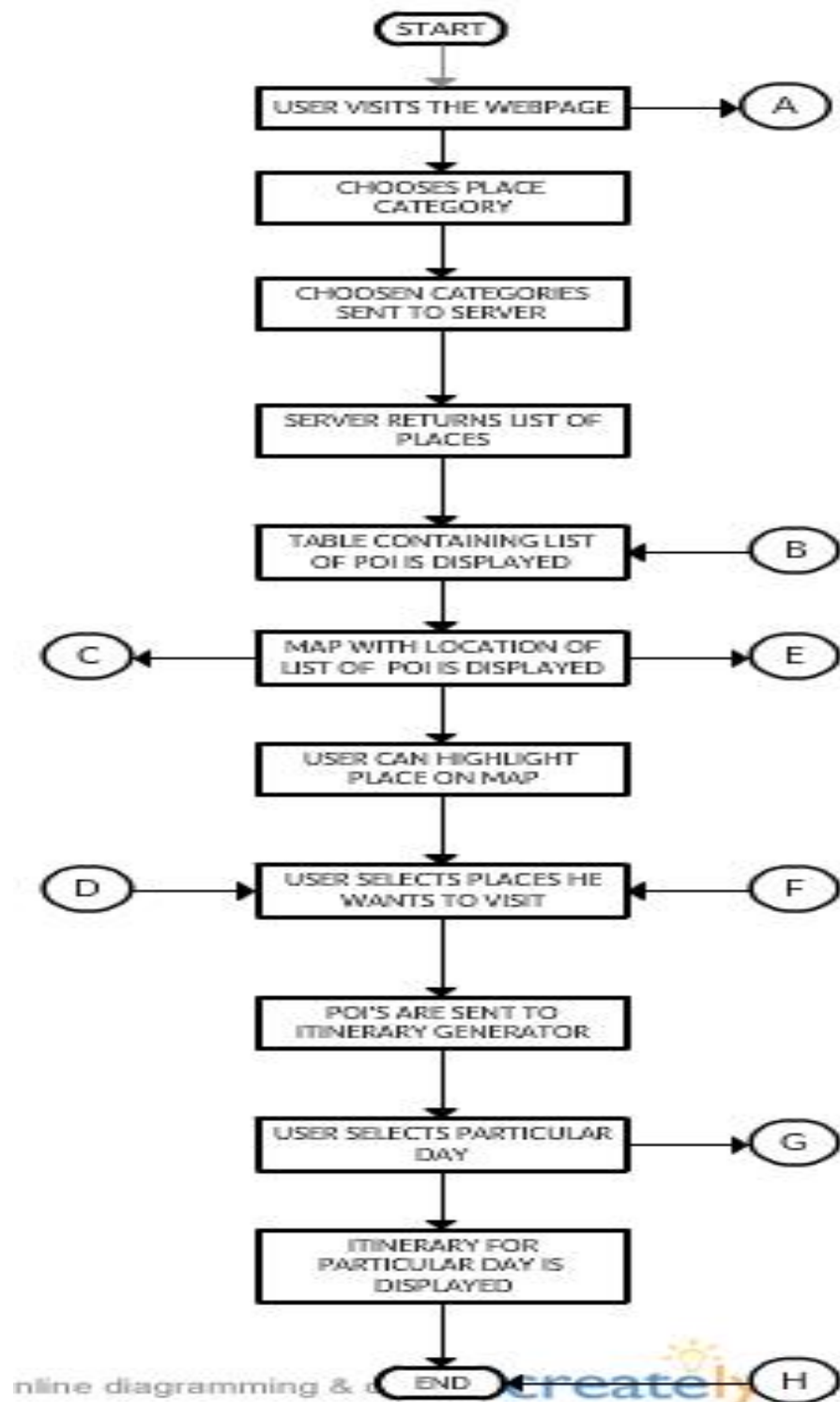


Fig 6.2 Work Flow A

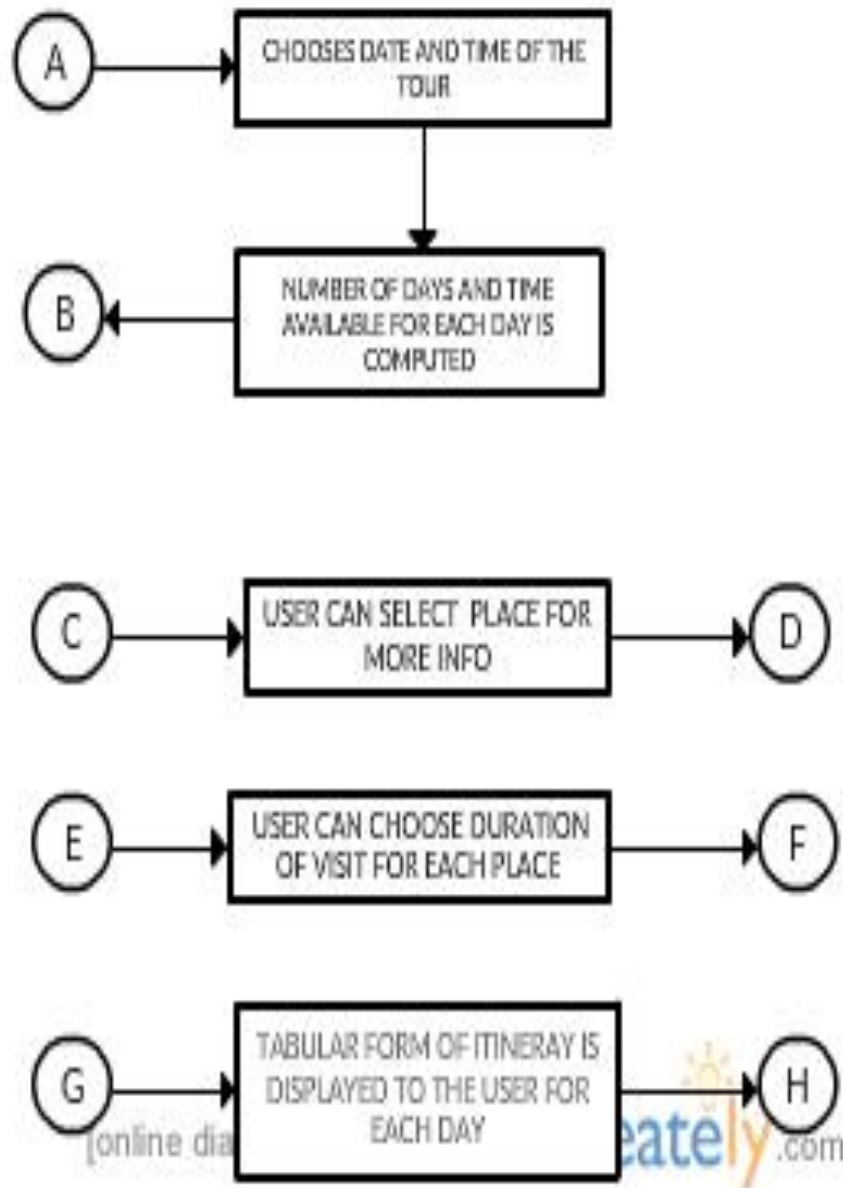


Fig 6.3 Work Flow B

6.2 Updated RTM

Requirement ID	Requirement Type	Requirement Description	Test Scenario	Results	Comments
FR1	Functional	Setting dates for trip			
FR2	Functional	Setting location of trip			
FR3	Functional	Setting category preferences			
FR4	Functional	Setting time slots for the day			
FR5	Functional	Generating itinerary			
FR6	Functional	Add/remove places from list			
UI1	Functional	Data input page	Manual	Pass	None
UI2	Functional	Map generation page	Manual	Pass	None
UI3	Functional	Itinerary page	Manual	Pass	None
NF1	Non Functional	Quick data access	Implicit	Pass	None
NF2	Non Functional	Reliability	Implicit	Pass	None
NF3	Non Functional	Optimal path generation	Implicit	Pass	None

Table 6.1 Updated RTM

7 Implementation

7.1 Pseudocode

The overall pseudocode for the project is as follows:

Data Scraping using Python:

- Set the main link to TripAdvisor site's places around Mysore page
- Use Selenium to identify each sub link to the categories based on category name
- Visit each sub link and get the URL of the page
- Store the URL in a list
- For each URL, visit the pages of the places in each sub category
- Get the URL of each place in the category
- Store the URL in a list
- Store the list as a dictionary
- For each URL, parse the page using BeautifulSoup and scrape the data
- Scrape place name, latitude, longitude, ratings, reviews
- Add average time spent by assigning a random value from a range for each place
- Store each in a list
- Append the list to a main list
- For each place, find out the time to travel and distance to every other place using Google distance matrix API
- Store as a list of lists

Database creation:

- Using MySQL for python for database
- Create table for places with values place_id, place_name, latitude, longitude, rating, reviews, URL and average recommended time
- Create table for categories with values category_id and category_name
- Create a table for the connection between places and categories, with values place_id and category_id
- Create a table for storing the distance and time between places, with values source_id, destination_id, time, distance
- All the tables have appropriate primary keys among them and foreign keys between them

Front end UI

- Get times and dates from user
- Side pane for selecting categories
- Multi tab front page with:
 - One tab for map using Google Maps API
 - One tab for getting input
 - One tab to display the itinerary

Frontend JavaScript

- Get the date and time from user
- Calculate the time available each day in hours
- Create a map using Google Maps API
- Take and store the user selected location
- On every category selected, check through all of them to identify which one is selected
- Create a string with selected categories separated by ‘;’ and send it to backend PHP file
- Use XHR call to send and data is received when status is OK and ready state is 4

- Take the places data and assign normalized scores based on rating and review
- Normalization is done using beta function
- Create a table and list the places arranged in descending order of their normalized scores
- Display the places on the map
- Add a slider for the user to be able to set the time they want to spend at the place
- For the selected places, XHR request is made to obtain the cached data
- The distance and time details between all the combinations of the places are obtained
- Use the latitude and longitude values of the places selected and apply K Means clustering algorithm to determine clusters of places, where K is determined by the minimum of the number of days chosen for the trip and number of places selected by the user
- Take the places in one cluster and use those as the list of places for that particular day
- Assign scores to each place calculated as

$$\text{Score} = x * \text{Travel time} - y * \text{Rating} + z * \text{Time spent at place}$$

Where x, y and z are normalizing co-efficients

- Apply Prim's algorithm to minimize the cost and cover maximum number of places
- Fit in as many places as possible in the given time constraint per day
- Return the data, and show it on the map with the source and destination being the user chosen location and the other places as waypoints in the order assigned by the algorithm
- Create an itinerary table with place, arrival time, departure time and time spent as the columns
- Fill the table with the results obtained
- Repeat this process for the number of days chosen by the user
- Dropdown element is created having option to select particular day
- For the selected day, the itinerary for that day is displayed

Backend PHP

Retrieving place data

- Receive the string of categories
- Separate them on semicolon
- Connect to MySQL database
- Run the query to get place_id, place_name, URL, ratings, reviews, recommended time
- Add each row as a list to another list
- Convert to JSON object and send

Retrieving cached distance/time data

- Receive the string of place_id's
- Separate them on semicolon
- Connect to MySQL database
- Run the query to get source_id, destination_id, time, distance
- Add each row as a list to another list
- Convert to JSON object and send

7.2 Codebase Structure

Project/UI/index.html

This is the front end HTML file with bootstrap. It contains the template of how our front end looks without any JavaScript. All the elements are static HTML elements, and only sources of scripts are included for all the interactions required.

Project/UI/js/

The main JavaScript files required for the project are located here. They include:

- Project/UI/js/frontend.js – The main backbone of our project, this JavaScript file has almost 1000 lines of code. All the interactions that index.html is bound happens in this file. This is also the file from where database access is made.
- Project/UI/js/cluster.js – This file has a JavaScript implementation of K-Means Clustering algorithm, but with a modification. The modification is that in normal K-Means uses Euclidean or similar measure of distance but since our project involved finding distances between latitude and longitude values, we had to replace the distance formula to geo-distance formula used for finding distances between latitude and longitude values.

Project/retrieve.php

This file contains the access and query to the database, i.e., when a request comes from frontend.js file for a set of categories that the user selects, the query is run to get the details of places belonging to the specified categories and return back to the js file.

Project/cache.php

This file is similar to the previous retrieve.php file, in that it connects to database and runs a query and returns the data to frontend.js file. The main difference is that the query runs on the distance and time between places data.

Project/UI/bootstrap/

All the files pertaining to bootstrap for index.html are present here. They include:

- Project/UI/bootstrap/css/ - All CSS files required for bootstrap
- Project/UI/bootstrap/fonts - All font files (ttf, woff etc.) required for bootstrap
- Project/UI/bootstrap/js - All offline JavaScript files required for bootstrap

Project/GetLinks.py

This file is used for getting the links of all the pages of different categories of TripAdvisor's Places around Mysore page using Selenium. All the returned data is stored in a list.

Places/DataExtract.py

This file contains the code to visit the link of every page of places listed under each of the categories and scrape the data from the page

Places/DatabaseCreation.py

It contains the code to create the tables in the database and set the appropriate primary and foreign key constraints.

Places/DatabaseInsertion.py

This contains code to insert the data scraped and stored in lists to be entered in the appropriate tables.

7.3 Coding Guidelines Used

We have used multiple languages in our project, and hence we will be listing the coding guidelines for all of them.

Python:

- Indentation: 1 tab per indentation level, where 1 tab is 4 spaces
- Line breaks before binary operator
- Blank spaces in functions only when logical separation is required
- Source file encoding is of UTF-8 format
- Import statements on separate lines and not comma separated
- Spaces only after commas and not before
- No unnecessary spaces preceding and succeeding brackets
- Comments are on separate lines and not inline
- Naming styles are either camel case or underscore separated lowercase characters and no uppercase characters
- Function names begin with lowercase letters
- Constants are in all capital letters

JavaScript:

- All variable names start with letter
- Indentation: 1 tab per indentation level, where 1 tab is 4 spaces
- Simple statements are ended with semicolon
- Opening bracket and closing bracket will be in new line
- Key value pairs are separated by colon plus one space for objects
- Quotes around string values are used in objects
- Script are always written in separate files
- Comments are on separate lines and not inline

- `/* */` , `//` both type of comments are used
- Naming styles are either camelcase or underscore separated lowercase characters and no uppercase characters
- Function name begin with lowercase letters

HTML

- File should start with `<!DOCTYPE html>`
- All element tags are in lowercase letters
- Every HTML elements is closed
- All attribute names are in lowercase letters
- Attribute values are in both lowercase and uppercase letters
- Quotes are used around the attribute values
- Spaces are not used around equal sign while writing attribute name value pairs
- Blank lines are added in the middle to separate logical code blocks
- For indentation 1 tab is used
- `<html>` and `<body>` tags are not omitted

PHP

- Indentation 1 tab is used for each level
- Simple statements are ended with semicolon
- Opening bracket and closing bracket will be in new line
- Functions are called and defined with no spaces between the function name and opening parenthesis, and first parameter
- `/* */` , `//` both type of comments are used
- `<?php ?>` is used to delimit PHP code. Shorthand `<? ?>` is not used
- Variable names are started with `$`
- Both lowercase and uppercase letters are used in variables
- One statement per line
- Comments are on separate lines and not inline

7.4 Sample Code

JavaScript – frontend.js

```
MAX = 99999;
function calculate_cost(matrix, wish_list){
  res = [];
  for(i=0; i<matrix.length; i++){
    temp = [];
    for(j=0; j<matrix.length; j++){
      if(i == j || j == 0){
        temp.push(MAX);
      } else if(j != 0){
        ratingNorm = wish_list[j-1]['rating'] * 2; //Rating on scale of 10
        avgTimeNorm = wish_list[j-1]['visit_time'] * 3; //Avg Time on scale of 10
        travelTimeNorm = matrix[i][j] * 10; //Travel time on scale of 10

        score = travelTimeNorm - ratingNorm + avgTimeNorm;
        //console.log(travelTimeNorm);
        temp.push(score);
      }
    }
    res.push(temp);
  }
  return res;
}
```


HTML – index.html

```
<div class="tab-pane" id="templates">
<div>
<label>DATE :</label>
<br><br>
<input type="text" class="form-control" style="width:200px" id="start_date" placeholder="start
date" name="date">
<br><br>
<input type="text" class="form-control" style="width:200px" id="end_date" placeholder="end
date" name="date">
<br><br>
<label>TIME</label>
<br><br>
<input class="form-control" style="width:200px" id="start_time" placeholder="start time"
name="time">
<br><br>
<input class="form-control" style="width:200px" id="end_time" placeholder="end time"
name="time">
</div>
<br/><br/>
<button type="button" class="btn btn-success" onclick="getTime()">Plan</button>

</div>
```

PHP – cache.php

```
$query = "SELECT p.place_id,p.place_name,c.category_name, p.latitude, p.longitude,  
p.url,p.rating,p.review,p.avg_time FROM place AS p, category AS c, placeToCategory AS pc  
WHERE p.place_id = pc.place_id AND c.category_id = pc.category_id AND c.category_name  
IN ('$ids') GROUP BY p.place_name ORDER BY p.rating DESC, p.review DESC;";
```

```
#Return the results of the query
```

```
$results = mysql_query($query);
```

```
if(!$results)
```

```
{
```

```
die("Problem in querying.." . mysql_error());
```

```
}
```

```
#Create an array and fill it with the results
```

```
$toSend = array();
```

```
$found = false;
```

```
while($row = mysql_fetch_array($results,MYSQL_ASSOC))
```

```
{
```

```
    $found = true;
```

```
    $toSend[] = $row;
```

```
}
```

```
#Send the array as a JSON object
```

```
echo json_encode(array('place' => $toSend));
```

```
if(!$found)
```

```
{
```

```
echo '<h2>NO RESULTS..SORRY :( </h2>';
```

```
}
```

Python – DataExtraction.py

```
#Dictionary for each place type and list of links of all places in its page
subLinks = {}

driver =
webdriver.Chrome('/Users/Chiranth/anaconda/selenium/webdriver/chrome/chromedriver')
for placeType, link in links.items():
    print(placeType, link)
    driver.get(link)

    #List of elements for each place type
    elem = []
    #Keep index at 0
    i = 0
    while True:
        try:
            temp =
            driver.find_elements_by_xpath("//div[@class='property_title']")[i].find_element_by_tag_name("a").get_attribute("href")
            elem.append(temp)
            i += 1

            #30 elements displayed per page, so after 30, go to next page and start from 0th index
            if(i >= 30):
                i = 0
                driver.find_element_by_link_text("Next").click()
        except:
            break

    #Now add the place type and its links to dict
    subLinks[placeType] = elem
```

7.5 Unit Test Cases

Unit Testing

- System is divided into following components :

Data scraping and mining (database builder)

Database

Itinerary generator

User interface

Separate tests on each component to make sure that system works properly

Testing for Data scraping and mining:

Test cases	Expected output	Actual output	Result
Test whether duplicate data is present	Duplicate data do not exist	Duplicate data do not exist	pass
Test for gap in the mined data	No	No	pass

Table 7.1 Overall Test

Testing for database

Test cases	Expected output	Actual output	Result
Checking for functional dependency	No functional dependency	No functional dependency	pass
Check for valid foreign key reference	Valid	Valid	pass

Data type of primary key and foreign key	Should be same	Same	pass
Presence of duplicate row	Duplicate row should not be present	Duplicate row is not present	pass

Table 7.2 Database Test

Testing Itinerary generator:

Test case	Expected output	Actual output	Result
Checking for every place in database	Should work for every place	Worked for every place	Pass
Assigning coefficients	Should balance the user rating and distance	Balanced the user rating and distance to certain extent	Passed for most cases, but difficult to accurately balance distance and rating

Table 7.3 Itinerary Test

Testing UI:

Test case	Expected output	Actual output	Result
Handle inconsistent date, time values	Handle the conditions gracefully	Prints about inconsistent values.	Pass
Unexpected errors	Program should not crash	Program exited without any error	Pass

Table 7.4 UI Test

7.6 Metrics for Unit Test Cases

We have calculated progress metrics on the number of major defects found per week and number of defects solved per week. We have tabulated the results and also have the graphs plotted.

Defects Found

Week No.	No. of defects found
1	3
2	5
3	6
4	8
5	10
6	7
7	6
8	7
9	5
12	4
11	3
12	2

Table 7.5 Defects Found

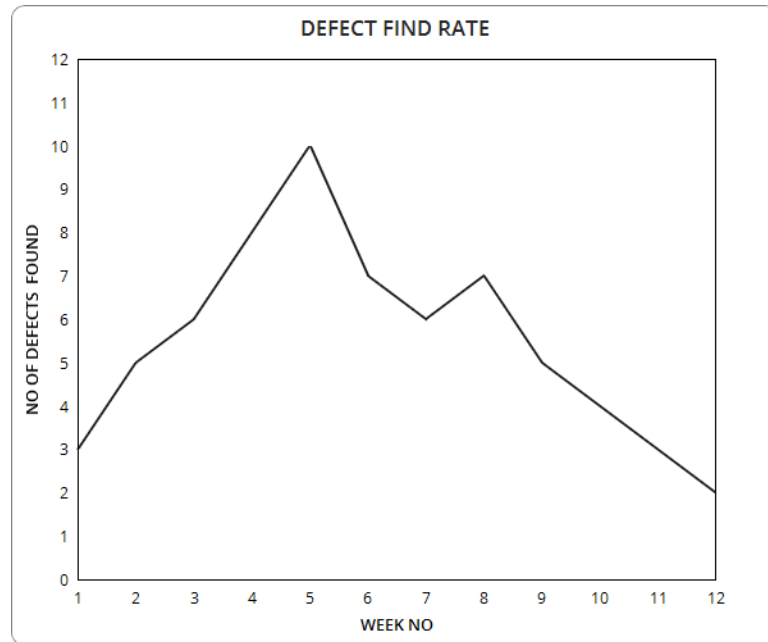


Fig 7.1 Defect Find Rate

Defects fixed

Week No.	No. of defects fixed
1	3
2	4
3	4
4	9
5	10
6	9
7	4
8	5
9	9
12	4
11	3
12	2

Table 7.6 Defects Fixed

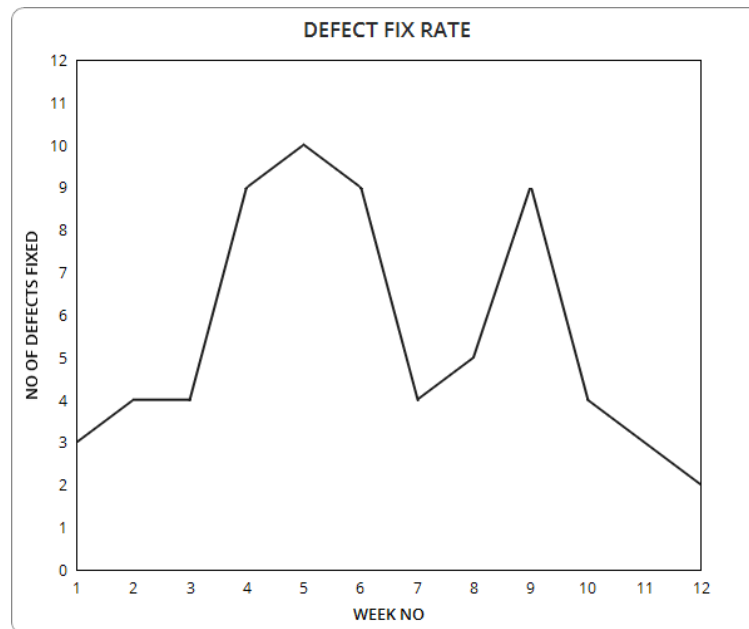


Fig 7.2 Defect Fix Rate

7.7 Updated RTM

Requirement ID	Requirement Type	Requirement Description	Test Scenario	Results	Comments
FR1	Functional	Setting dates for trip	T1	Pass	None
FR2	Functional	Setting location of trip	T2	Pass	None
FR3	Functional	Setting category preferences			
FR4	Functional	Setting time slots for the day	T4	Pass	None
FR5	Functional	Generating itinerary			
FR6	Functional	Add/remove places from list			
UI1	Functional	Data input page	Manual	Pass	None
UI2	Functional	Map generation page	Manual	Pass	None
UI3	Functional	Itinerary page	Manual	Pass	None
NF1	Non Functional	Quick data access	Implicit	Pass	None
NF2	Non Functional	Reliability	Implicit	Pass	None
NF3	Non Functional	Optimal path generation	Implicit	Pass	None

Table 7.7 Updated RTM

8 Testing

8.1 System Testing

Functional testing -

- FR1: It was made sure that dates were settable
- FR2: Location marker was tested to be movable to user's choice of preference on the map
- FR3: Category selection pane was made functional and tested by selecting multiple categories
- FR4: It was made sure that dates and times were settable
- FR5: Itinerary was generated for the places, according to the algorithm, tested to work with different places as inputs across categories and geo locations
- FR6: Places were made to be able to be chosen by the user, tested to be both selected and deselected
- UI1: Data input page was checked to be working by printing the values got from front end in JavaScript
- UI2: Map generated itinerary was made sure to be working fine by checking with the places that were supposed to be shown according to the algorithm and also by checking if the places locations matched that on Google Maps website
- UI3: When user selects the places in the table, itinerary for selected places for each day is displayed, tested with the places displayed on the map

Non-functional -

- NF1: All the requests for data from the JavaScript code to PHP code to retrieve data from database is done very quickly, all tested to be under 2 seconds
- NF2: Database uses MySQL with its advanced security and reliability features so implicitly taken care of
- NF3: Optimal path generation uses a variation of Prim's algorithm for Travelling Salesman Problem. Since the optimal path generation is NP hard problem, we have achieved relatively good results as tested by the places and how long it takes to visit from one place to another

Load testing: Itinerary requests with various different number of

- places
- categories
- days
- time

is used to test the system. Optimum load within the threshold is desirable.

Regression testing:

At each stage of the development phase like database builder, itinerary generator and integration of these components we made sure that new build did not create any new bug or brought back the old one.

Stress testing:

The performance of the system to different stress amount is related to how good the browser (web interface via which the system is running) can handle the stress.

8.2 Test Environment used

1. Operating System – Linux and mac
2. Browser – Chrome
3. Python version – 3.5 with the required dependencies

8.3 Test procedure

Testing for data scraping and database:

Many tourist websites with lot of information was taken into consideration .then websites with most trustworthy , most popular in terms of rating and user review is selected for scraping. Database was tested with wrong queries for detection of error in declaration of data types. Functional dependencies are also tested with proper test queries .

Testing for itinerary generator:

Different number places were given as input to the itinerary generator. Also

Places with

- Low rating + long distance
- Low rating + short distance
- High rating + long distance
- High rating + short distance

are fed as input to the itinerary generator to observe the behaviour of algorithm under different conditions.

Testing for UI

The UI can be tested based on providing different kinds of inputs through the front-end and how system handles unexpected conditions and exceptions

8.4 Example Test results

The table below summarizes the test cases employed for various test categories and the test results obtained for each test case :

Test Case ID	Description	Result	Comments
Validation Testing	Testing the correctness provided by the software	Pass	The data provided for the software for different combination of the inputs were tested and verified.
Implementation Testing	Testing the implementation details of the project	Pass	

Table 8.1 Test Results

Test types summary table – The table below contains the list of validation test run

Test Case	Description	Expected	Output	Result	Comments
Validation testing for valid input for itinerary generator	Test sample is user selected places	Optimal itinerary	Optimal itinerary	Pass	
Validation testing for multiple days	Test sample is a user selected date and time	Itinerary for each day	optimal Itinerary for every day	pass	

Table 8.2 Validation Results

Validation Tests Table - The table below contains the list of Implementation Tests run:

Test Case	Description	Expected	Output	Result	Comments
Implementation testing for the overall correctness of the software.	Model that generates optimal itinerary for user	No errors	No errors	Pass	Although perfect itinerary is not possible to achieve yet , optimal solution was obtained
Implementation testing for the overall correctness of the software.	Check the user flow of the software	Error free and easy flow	Error free and simple user interaction	Pass	This should be as simple as possible to enhance usability

Table 8.3 Implementation Results

Test Incidents –

The following were the unexpected results and defects that occurred during testing:

Incident	Description	Resolve status
Limit on Distance matrix range	Google maps API restricted the distance matrix request for max of 25 elements	Resolved (by caching and pre-processing the distance between two places)
Tester UI issue	UI could be more enhanced and attractive and advanced.	Resolved

Table 8.4 Unexpected Results

8.5 Test Metrics

Number of test cases run = 37

Test cases passed = 29

Test cases failed = 8

Analysis:

- Improper handling of data returned from PHP
- Incorrect usage of Google Maps API
- Array of places overflow indexing errors
- DOM elements handling errors

8.6 Updated RTM

Requirement ID	Requirement Type	Requirement Description	Test Scenario	Results	Comments
FR1	Functional	Setting dates for trip	T1	Pass	None
FR2	Functional	Setting location of trip	T2	Pass	None
FR3	Functional	Setting category preferences	T3	Pass	None
FR4	Functional	Setting time slots for the day	T4	Pass	None
FR5	Functional	Generating itinerary	T5	Pass	None
FR6	Functional	Add/remove places from list	T6	Pass	None
UI1	Functional	Data input page	Manual	Pass	None
UI2	Functional	Map generation page	Manual	Pass	None
UI3	Functional	Itinerary page	Manual	Pass	None
NF1	Non Functional	Quick data access	Implicit	Pass	None
NF2	Non Functional	Reliability	Implicit	Pass	None
NF3	Non Functional	Optimal path generation	Implicit	Pass	None

Table 8.5 Updated RTM

9 Results and Discussions

9.1 Justifications for the results obtained

The results obtained in our project vary vastly based on the user's location, the category of places and the choice of places he/she wishes to see and the time he has available. Hence it cannot be directly validated by any known quantitative measure. However, it doesn't need to be validated as well, since the user has the option to chop and change the places even after the itinerary is displayed. This gives the user the flexibility to change the results to suit his/her taste and requirements.

Also, Prim's algorithm will try to find minimal cost path to cover as many places as possible, so user need not worry about the order of visit or choosing one place over the other to visit. For multi day plans, K-Means algorithm will be appropriate to find particular clusters and then apply Prim's algorithm so that a cluster of nearby places are chosen for the day and hence saves the user on travelling time.

9.2 Snapshots

Here are a few screen shots of the results we got.

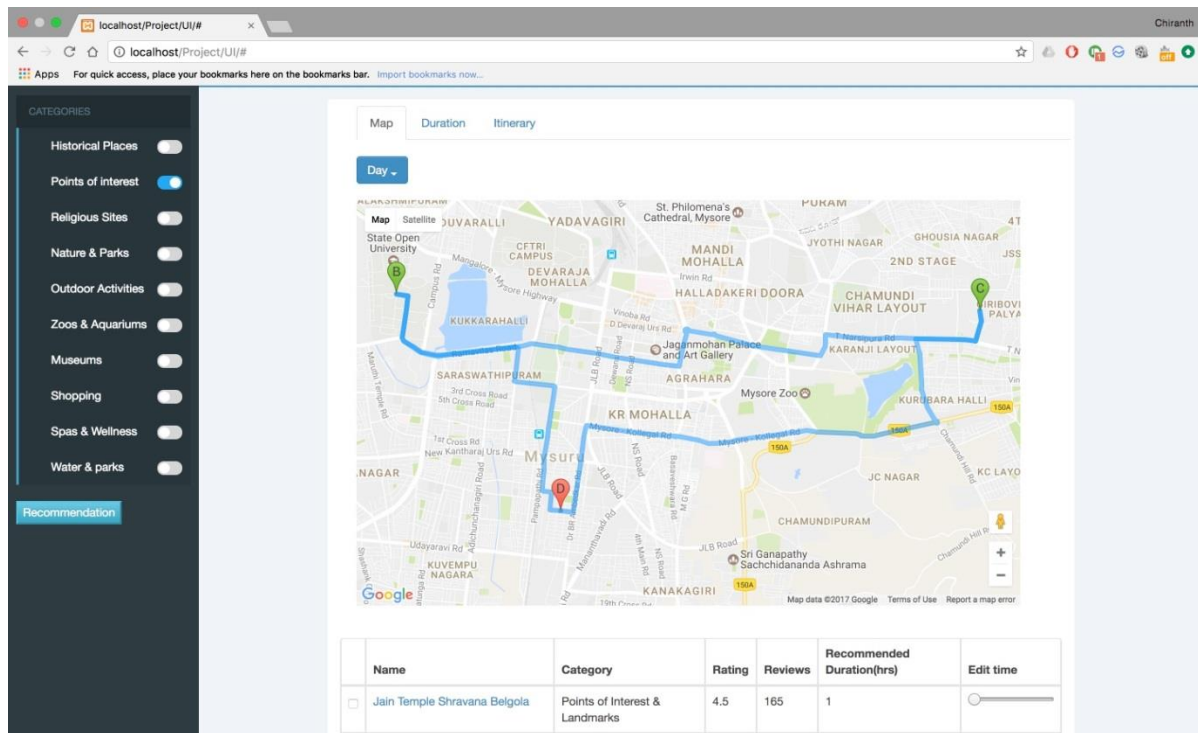


Fig 9.1 Snapshot 1

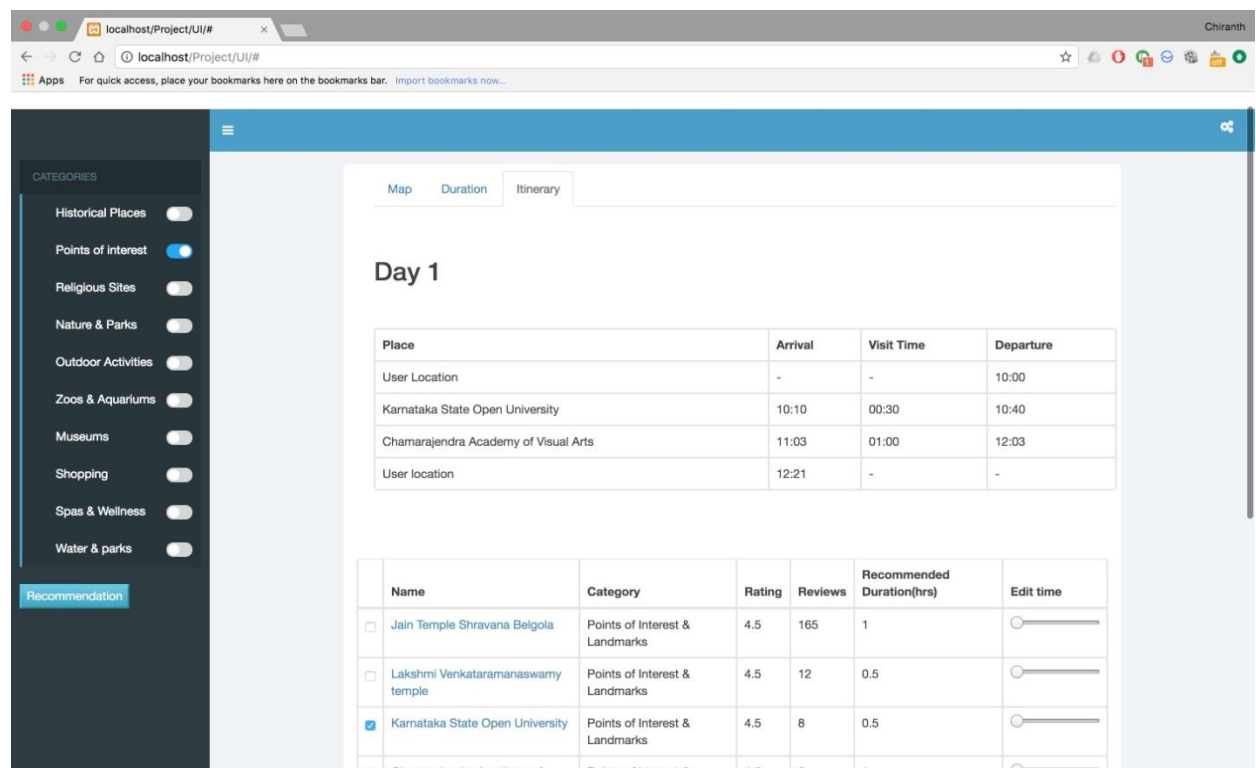


Fig 9.2 Snapshot 2

9.3 Conclusion

The project was successfully completed with all the required features added. The core functionality of the project is operational, which is to recommend a set of places and create an itinerary for the user, based on his requirements. He also has the ability to tweak it and play around with it and add/remove places.

9.4 Opportunities of future enhancements

We have put our best efforts to come up with a useful and meaningful project with as many features added as we could have, in the amount of time we had. But irrespective of that, there is definitely a scope for future enhancements. Below listed are the improvements that we think could be added later on.

- Currently, our project only has data of places around Mysore, but that can be increased to add attractions around other places as well.
- For our data, we have relied on data from one source, i.e., TripAdvisor. We can also scrape data from other places.
- We can add rating and review data from Google, which we had already planned to do. But Google does not have an API separate for this purpose, and we had to scrape data from actual Google page searches. This way of scraping was a little unreliable, and also combining it to the fact that many places listed in TripAdvisor were not found. This made us to skip adding Google data into our data
- Our algorithm for generating least cost path, depends on the cost that we assign to a place in the first place, and we calculate that cost using some factors to normalize the variables. But this may not be the best way to generate the cost and there may be a better normalization or a better function itself.

- User login can be added, so that each user has his own separate account, to keep track of the places he has visited and liked/disliked, which can then later be used for generating better recommendations.
- The UI that we have created for our project is quite a basic one, with less fancy features and more functional ones, which are just required to showcase the functionality of our project. We could make a better UI which concentrates more on user experience and enhance the feel, thereby making it more intuitive and interesting to use.

9.5 Verification of RTM

Requirement ID	Requirement Type	Requirement Description	Test Scenario	Results	Comments
FR1	Functional	Setting dates for trip	T1	Pass	None
FR2	Functional	Setting location of trip	T2	Pass	None
FR3	Functional	Setting category preferences	T3	Pass	None
FR4	Functional	Setting time slots for the day	T4	Pass	None
FR5	Functional	Generating itinerary	T5	Pass	None
FR6	Functional	Add/remove places from list	T6	Pass	None
UI1	Functional	Data input page	Manual	Pass	None
UI2	Functional	Map generation page	Manual	Pass	None
UI3	Functional	Itinerary page	Manual	Pass	None
NF1	Non Functional	Quick data access	Implicit	Pass	None
NF2	Non Functional	Reliability	Implicit	Pass	None
NF3	Non Functional	Optimal path generation	Implicit	Pass	None

Table 9.1 Verification of RTM

10 Retrospective

What went well

The project had turned out to be more or less what we would have wanted to do when we began working on it. We added the features which were most important, and core functionality was achieved. The problem statement was attacked with a proper solution.

What did not go well

Mid-way through the project, we were faced with a roadblock, which was that Google Maps API had a limit for generating the distance and time matrix, of 5 places. This meant that the user could only select 5 places at a time for generating itinerary and selecting any more places resulted in an error. The problem was, we couldn't do it 5 places at a time as well, since we needed to find out distance from once place to every other place. This was a major issue, since it basically meant a big part of our core functionality would end up being affected.

We informed this problem to our guide, who gave a suggestion to generate the matrix before-hand and store it in the database. So we decided to follow the suggestion and wrote a JavaScript code to generate distance matrix for one place to 25 places at a time. We went ahead and did that, but unfortunately we were dealt with the negative side of vagaries of how JavaScript asynchronous calls work. What was happening was that, we were trying to add the results obtained for each place, to a list. This list, when printed in console.log, was showing as if the list was filled as expected, but in reality, the list was not getting filled with anything. In the code, trying to access the elements of the list was throwing an error saying that the list was

empty. Later we came to know that the values in the list as shown in console.log was actually dynamically getting added as and when we checked in the console.log and not in the code.

Again, after spending lot of time to make it work, we went ahead to do the same thing in Python, which later worked but had another issue, which was that it was inexplicably stopping working intermittently. So we had to manually keep continuing its run whenever it stopped to run and then save the results. So all this process to avoid Google's limitation made us lose a lot of time and hence towards the end we had to lose sleep to reach the targets we wanted to reach.

Learnings beyond technologies

Most important learning for us has been time management, to manage time for project. Since both of us are interns, we don't really get much time for project, so within the few hours we had per day, we had to manage to work. So what ended happening was that we had to sacrifice sleep, hanging out with friends, going to movies, and in some cases had to sacrifice working late at office to working for our project. That taught us a lot about priority management along with time management and now we are satisfied with the work we have done for our project and how it has shaped up to be.

11 References and Bibliography

1. Location-Aware Recommendation Systems by María del Carmen Rodríguez-Hernández, Sergio Ilarri, Raquel Trillo-Lado, Ramón Hermoso - Article 2015
2. The Use of Machine Learning Algorithms in Recommender Systems: A Systematic Review by Ivens Portugal, Paulo Alencar, Donald Cowan - arXiv 2015
3. Web application for recommending personalised mobile tourist routes by D. Gavalas, M. Kenteris, C. Konstantopoulos, G. Pantizou - IET Softw., 2012
4. www.stackoverflow.com
5. www.python.org
6. www.w3schools.com

12 User Manual

Steps to be followed:

1. Select the location from where you would like to start the trip on the map
2. Move to the second tab
3. Enter the dates and times of the trip
4. Select plan button
5. On the left hand side, select the categories from the pane
6. You can either choose Auto recommend option or choose the places manually and select the Choose option
7. The places are listed in all the tabs for convenience purposes
8. Once selected, itinerary will be generated in on the map in the first tab in the order of visit marked by succeeding letters of the alphabet
9. The itinerary in the form of table will be generated in the third tab, with arrival, departure and visit duration details