****

**Visvesvaraya Technological University,**

**Jnana Sangama, Belgaum - 590014**

A SRS project report on

**Personalized Tourism Itinerary Recommender System**

Submitted in partial evaluation of the 8th semester project progress review-1

Submitted by:

1PI13CS047 Chiranth Gopal K

1PI13CS062 Gourish Hebbar

Under the guidance of

**Dr. B Narsing Rao**

Professor, PESIT

**Jan – May 2016**

|  |  |
| --- | --- |
| PESIT logo.jpg | **Department of Computer Science & Engineering**  **PES Institute of Technology,**  **(An Autonomous Institute under VTU, Belgaum and UGC, New Delhi)**  **100FT RING ROAD, BSK 3RD STAGE, BENGALURU - 560085** |

**Abstract:**

When we want to go visiting places we are new to, we would not know what places to go and which are good ones that we would like. So, we are building a web app, in which the user will first login and enter his preferences. Each user, once he logs in, will have option to tell what his duration of trip will be, i.e., the dates, and for each day, the time 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. Then using this information, if the user has already visited certain places and has given ratings to that place, then using that information, using the rating information for places, average time spent at place, location information stored in the database, we will generate an itinerary for the user. It will contain a map, with the recommended places to visit, in the shortest time possible between places, with the start point the user selects as the destination point as well. So, with this, the user can get a good idea as to how he could 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.

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **Chapter No** | **Chapter Name** | **Page no** |
| 1 | Introduction | 4 |
|  | 1.1 Problem definition | 4 |
|  | 1.2 Generic proposed solution | 5 |
|  | 1.3 Acknowledgement |  |
| 2 | Literature Survey | 7 |
|  | 2.1 References | 8 |
| 3 | Software Requirement Specification | 9 |
|  | 3.1 High level architecture | 10 |
|  | 3.2 Environment used |  |
|  | 3.2.1 Hardware Requirements | 11 |
|  | 3.2.2 Software Requirements | 11 |
|  | 3.2.3 Functional Requirements and Test cases | 12 |
|  | 3.2.4 Non functional Requirements | 13 |
|  | 3.2.5 Constraints | 14 |
|  | 3.2.6 Requirement Traceability Matrix | 14 |
|  | 3.2.7 Use case scenarios | 15 |
|  |  | 17 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

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 itinerary for attractions based on user location and preferences.

**1.2 Generic proposed solution:**

These are the detailed steps of our solution:

1. Database: We are building a database, which is populated by scraping the data from multiple sites like TripAdvisor, Google, which we collect and create a database using MySQL. Since each site will have it's 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: We are using the rating data of the place from different sites, 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**

This section is for the acknowledgement of the guide towards problem definition and the proposed approach for solving the problem.

**Signature of the guide**

1. **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

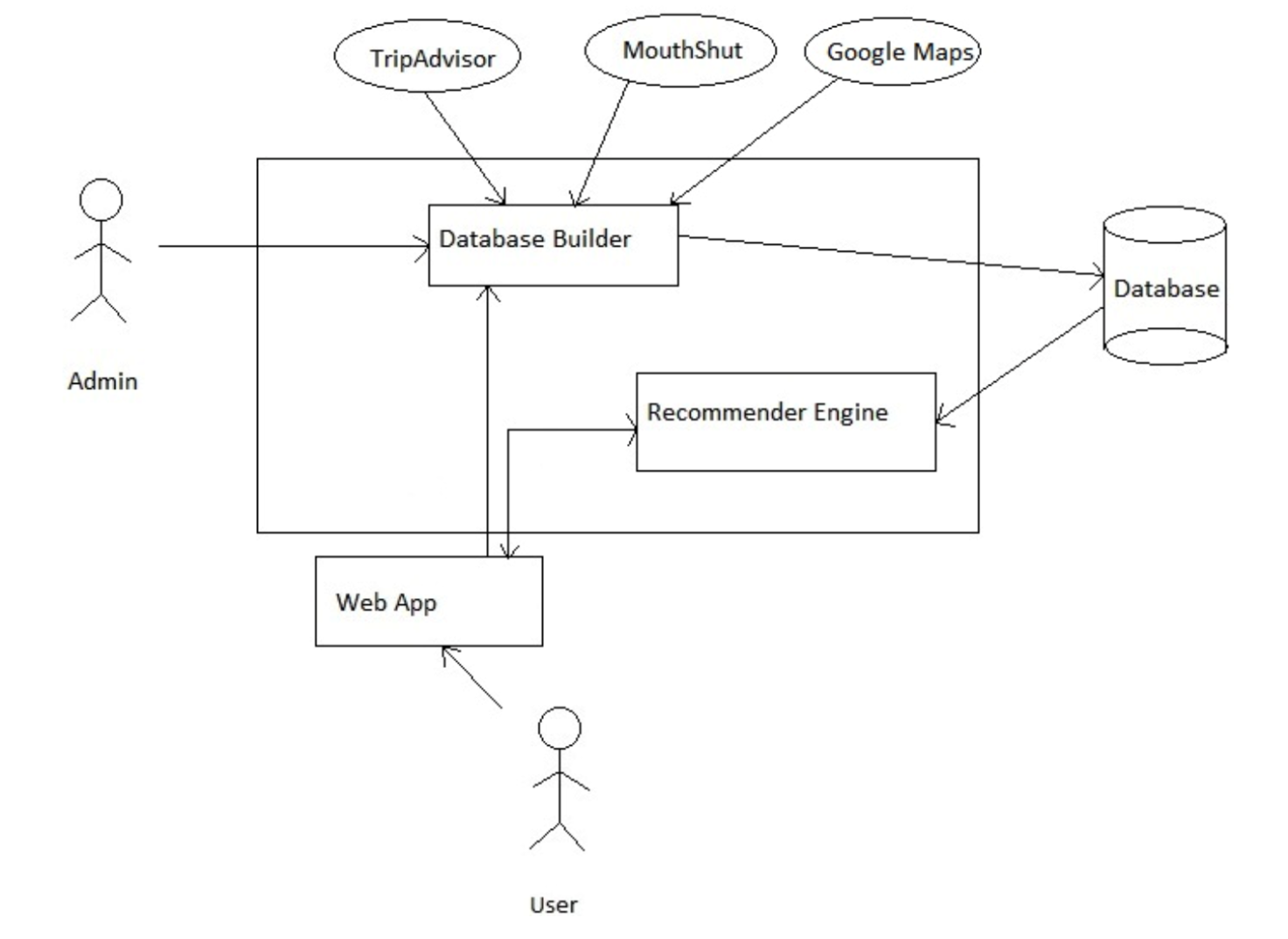
This paper provides a survey on location-aware recommender systems in scenarios involving mobile computing. It describes the fundamentals of recommendation systems, shows the most relevant existing approaches for location aware systems. Then it talks about the current applications of location aware recommender 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.

1. 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.

1. 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.

1. **System Requirement Specification:**
   1. **High level architecture diagram**
   2. **Hardware Requirements**

* Server - laptop
* Client device - laptop, phone

Since we are not yet planning to host our web app online after completion, our hardware requirements, i.e., server and client, are both our own personal devices.

**3.3** **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
  1. **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 choses 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
  1. **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

**3.6 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

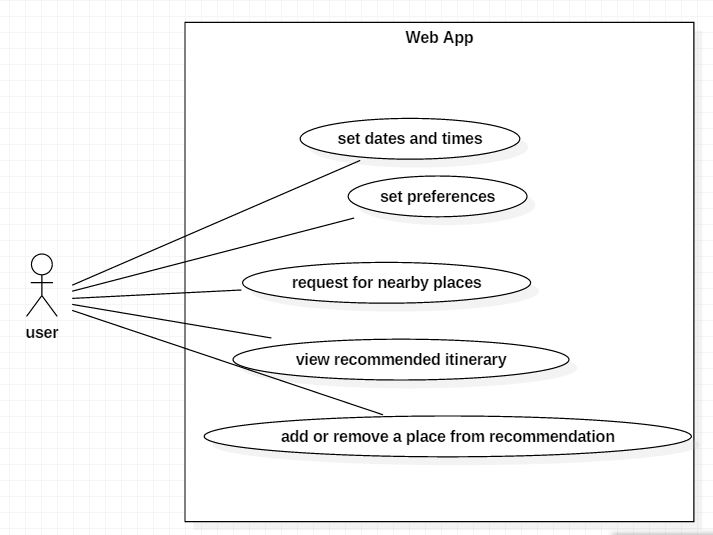
**3.7 Requirement Traceability Matrix (RTM)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **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 |  |  |  |

**3.8 Use case scenarios**

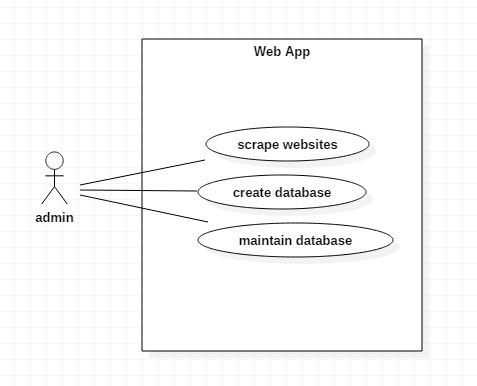
* User

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

****

* Admin

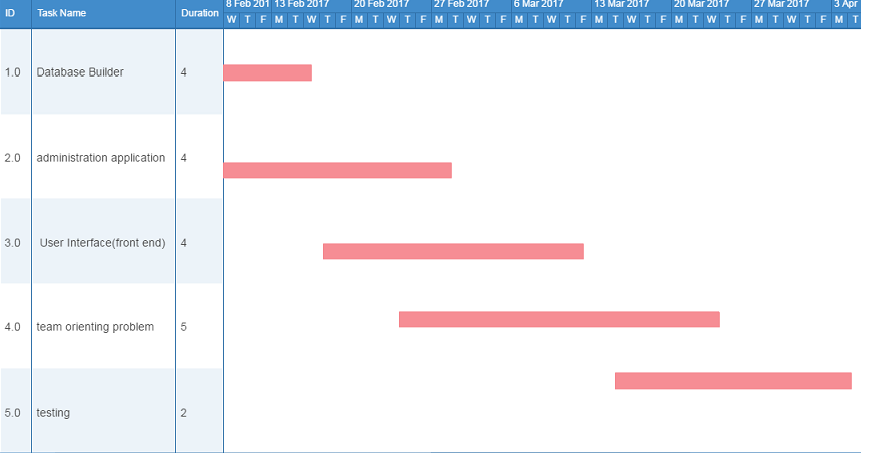
The following diagram denotes the high level use cases of the admin’s interaction with the web app’s database.



1. **Schedule**

**Gantt Chart**

Attached below is the Gantt Chart for our project.

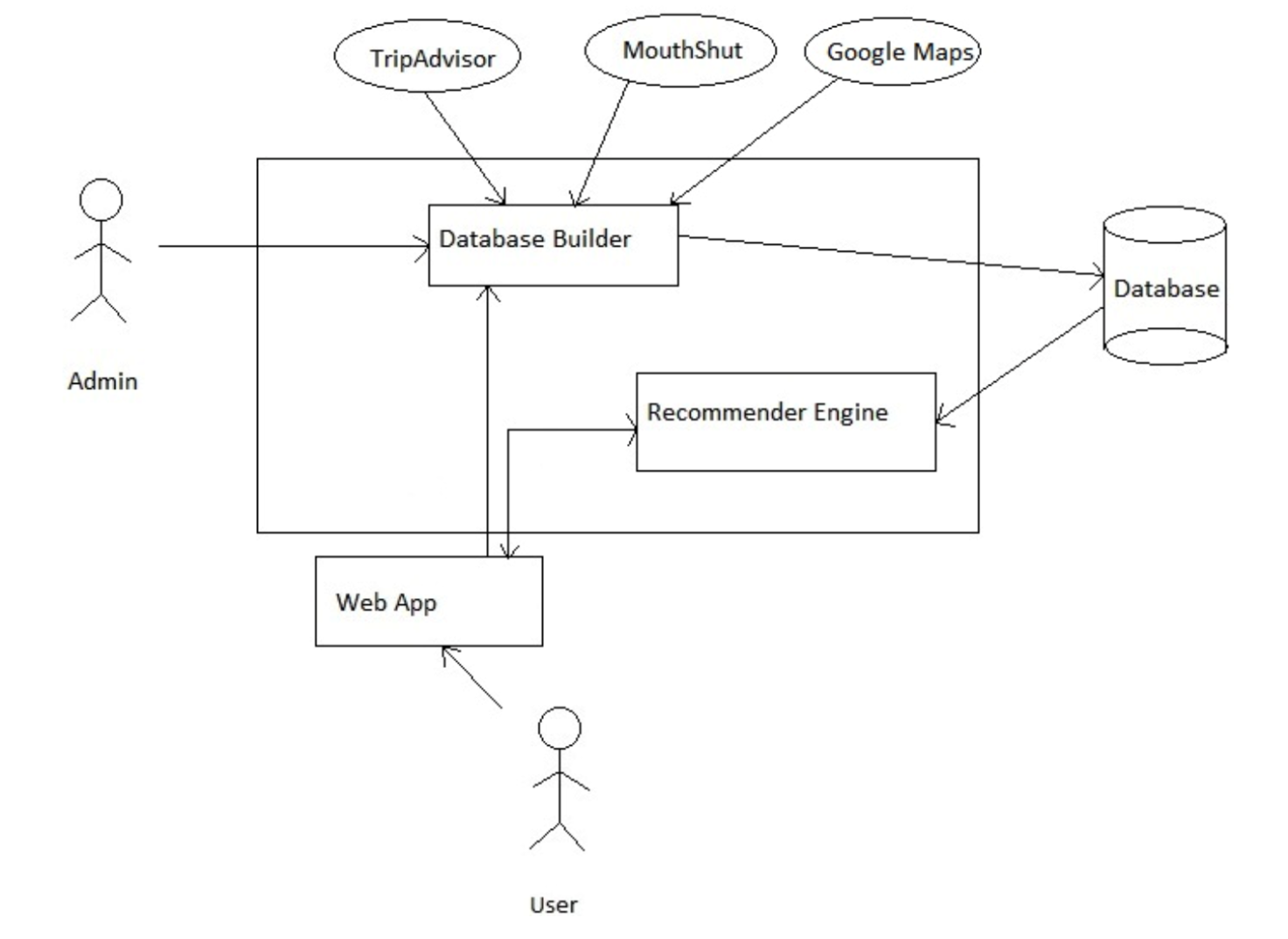


**Task breakdown table**

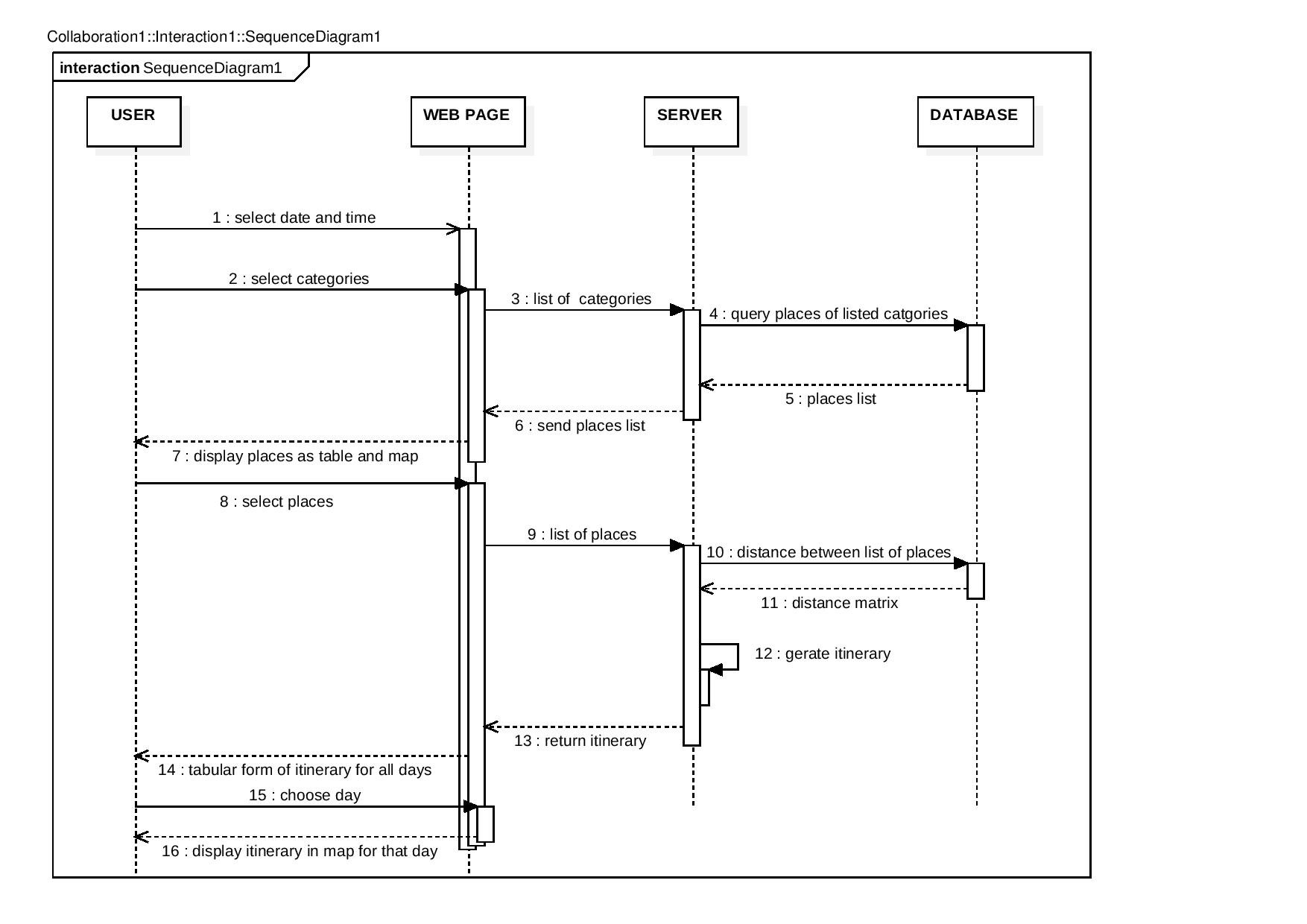
The tasks mentioned in Gantt chart are further broken down in terms of tasks and the dates on which they were expected to be completed on.

|  |  |  |
| --- | --- | --- |
| **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 |

1. **System Design**
   1. **Architectural Diagram**

****

* 1. **Sequence Diagram**

****

* 1. **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 difference 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.

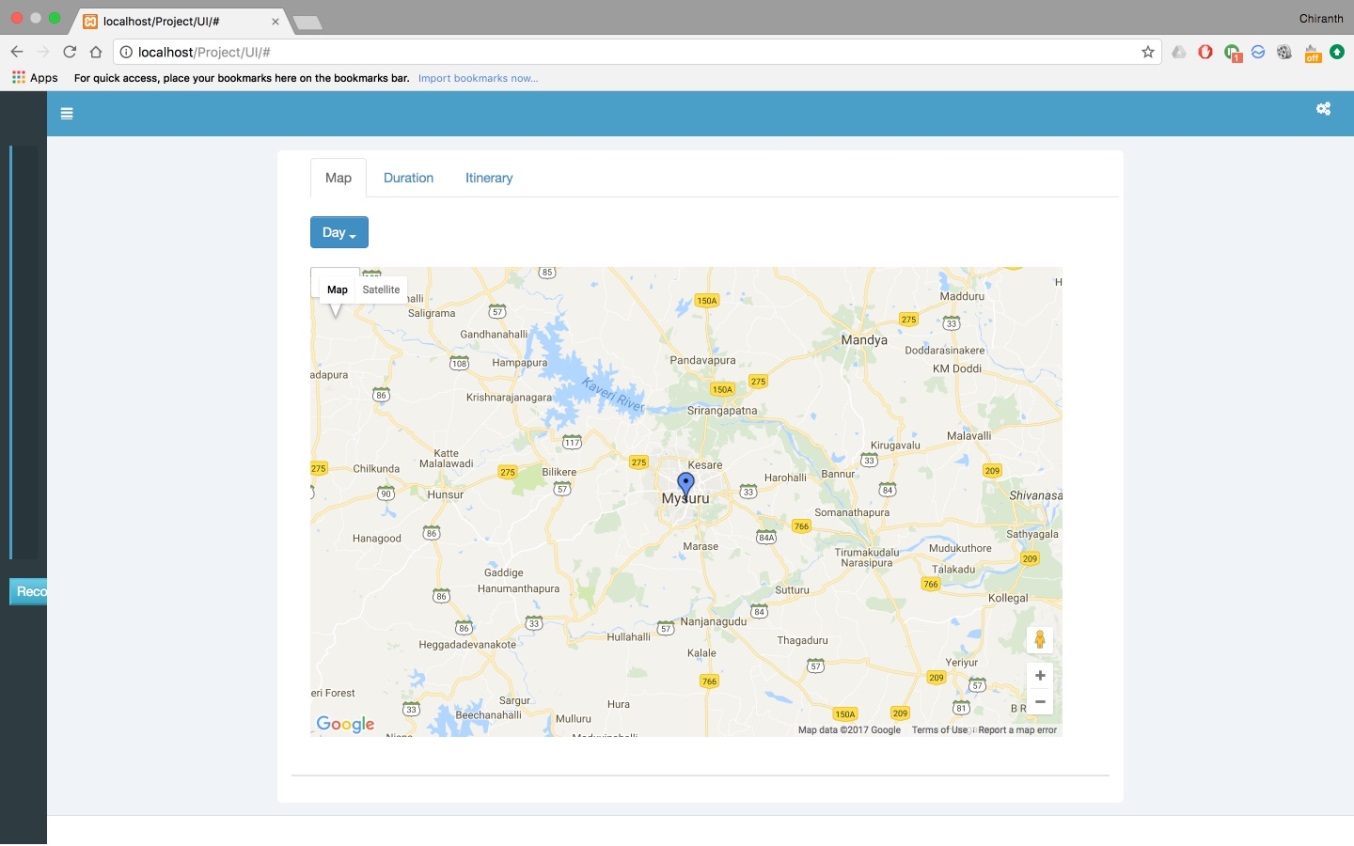


Fig 5A

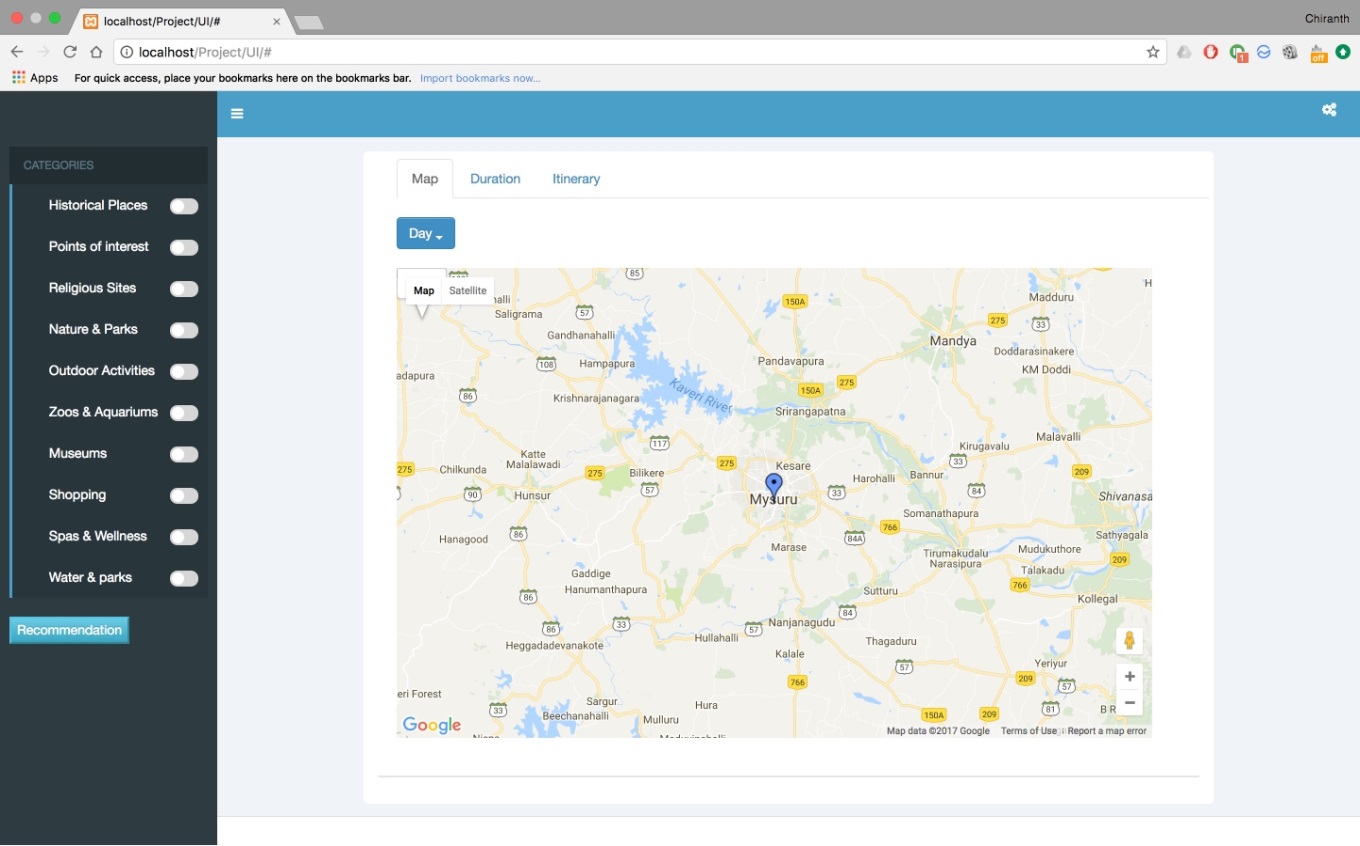


Fig 5B

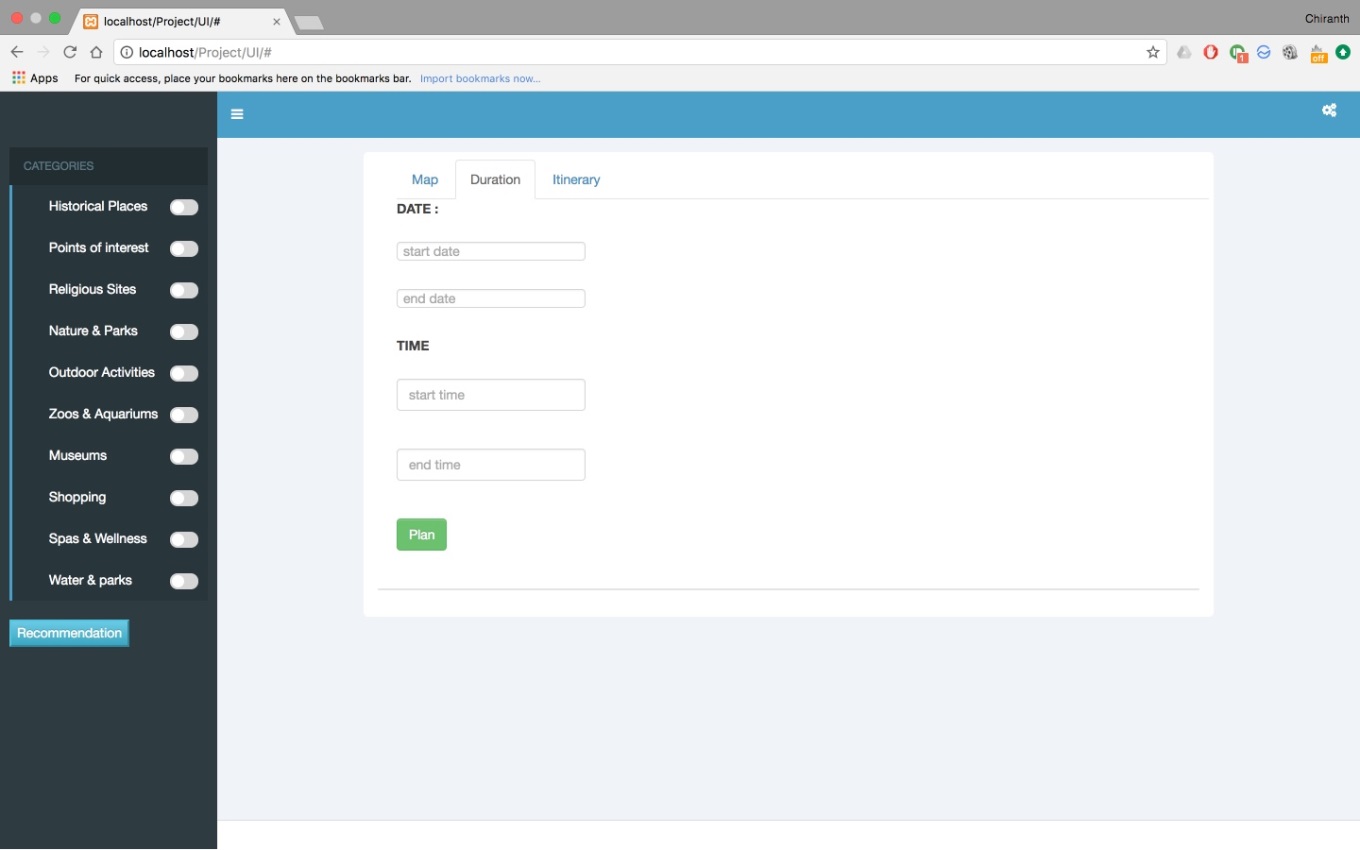


Fig 5C

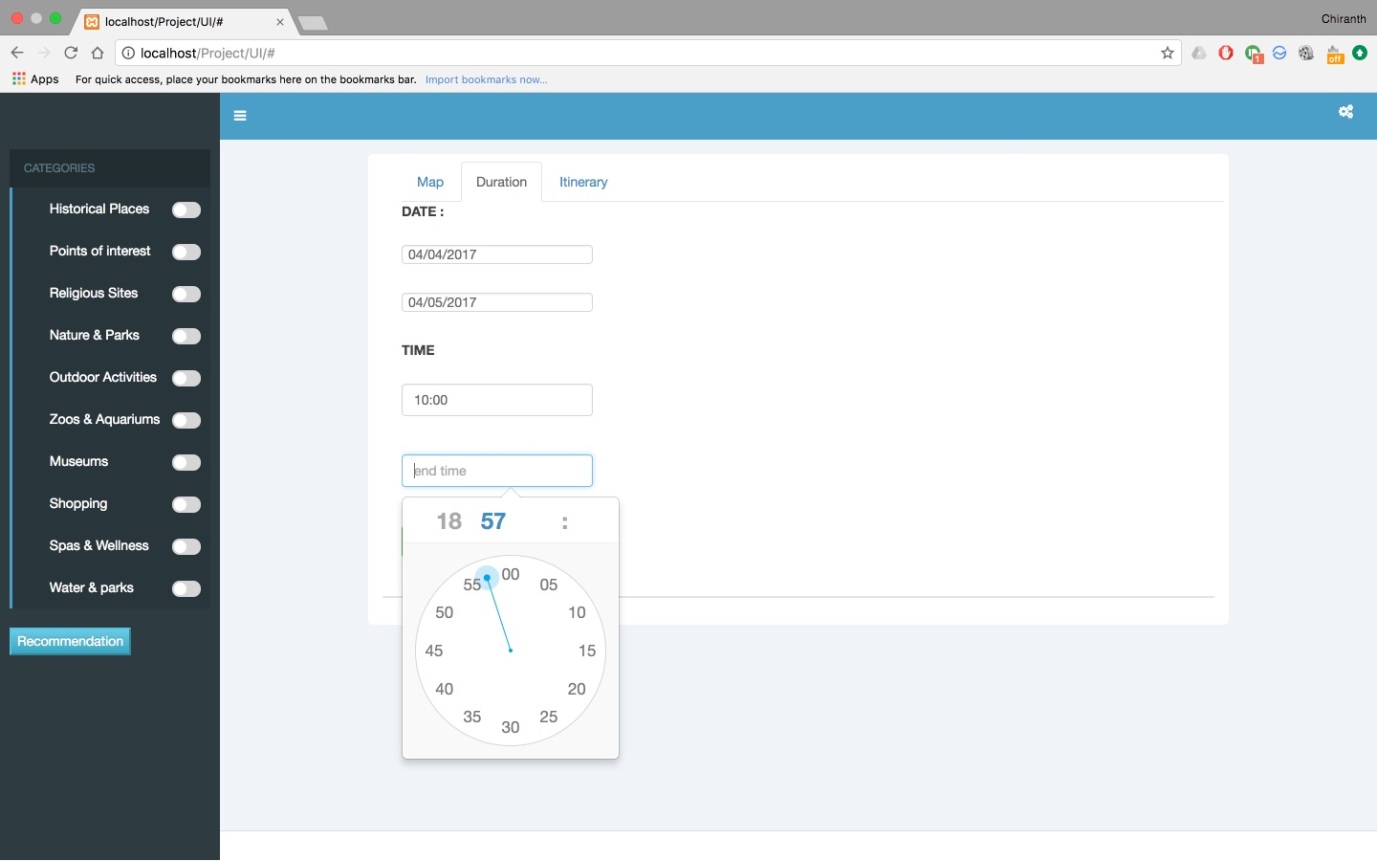


Fig 5D

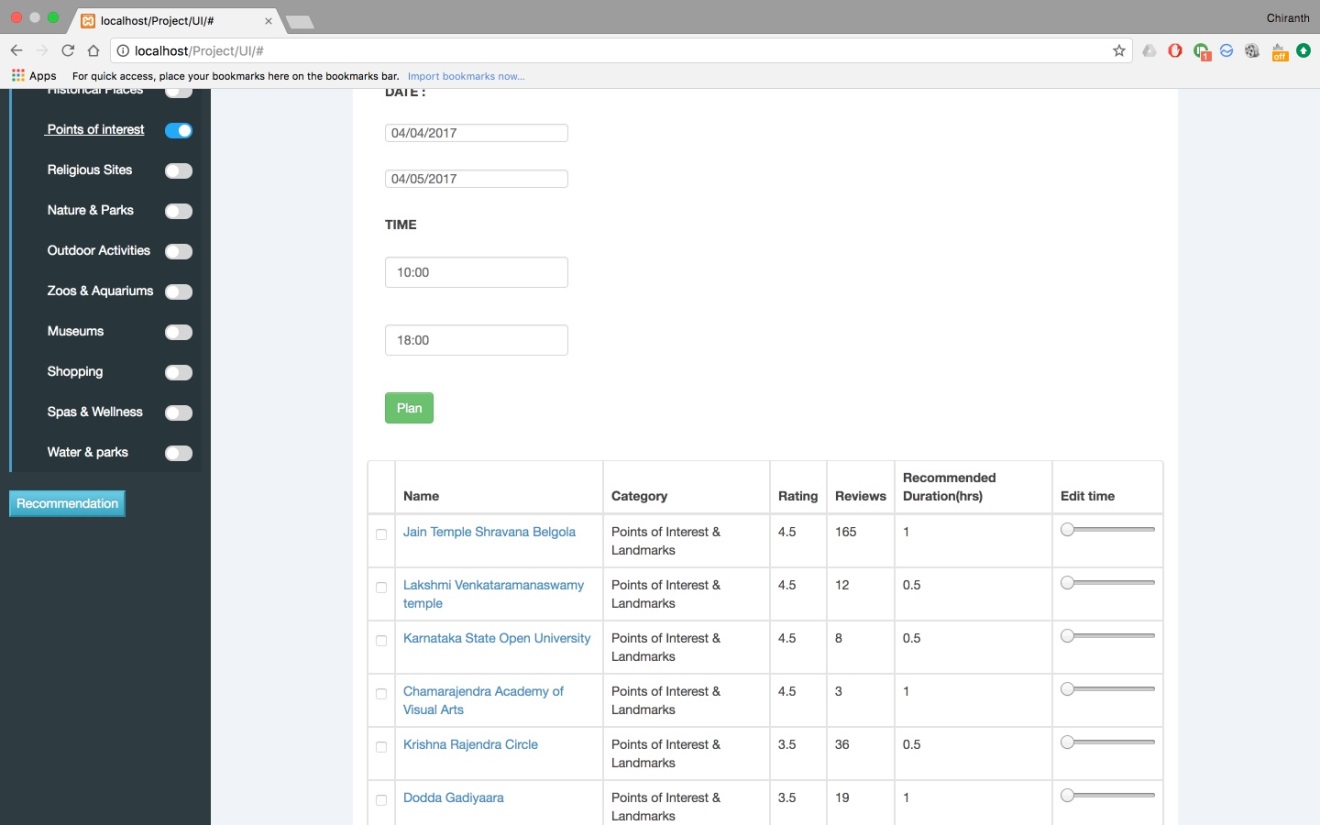


Fig 5E

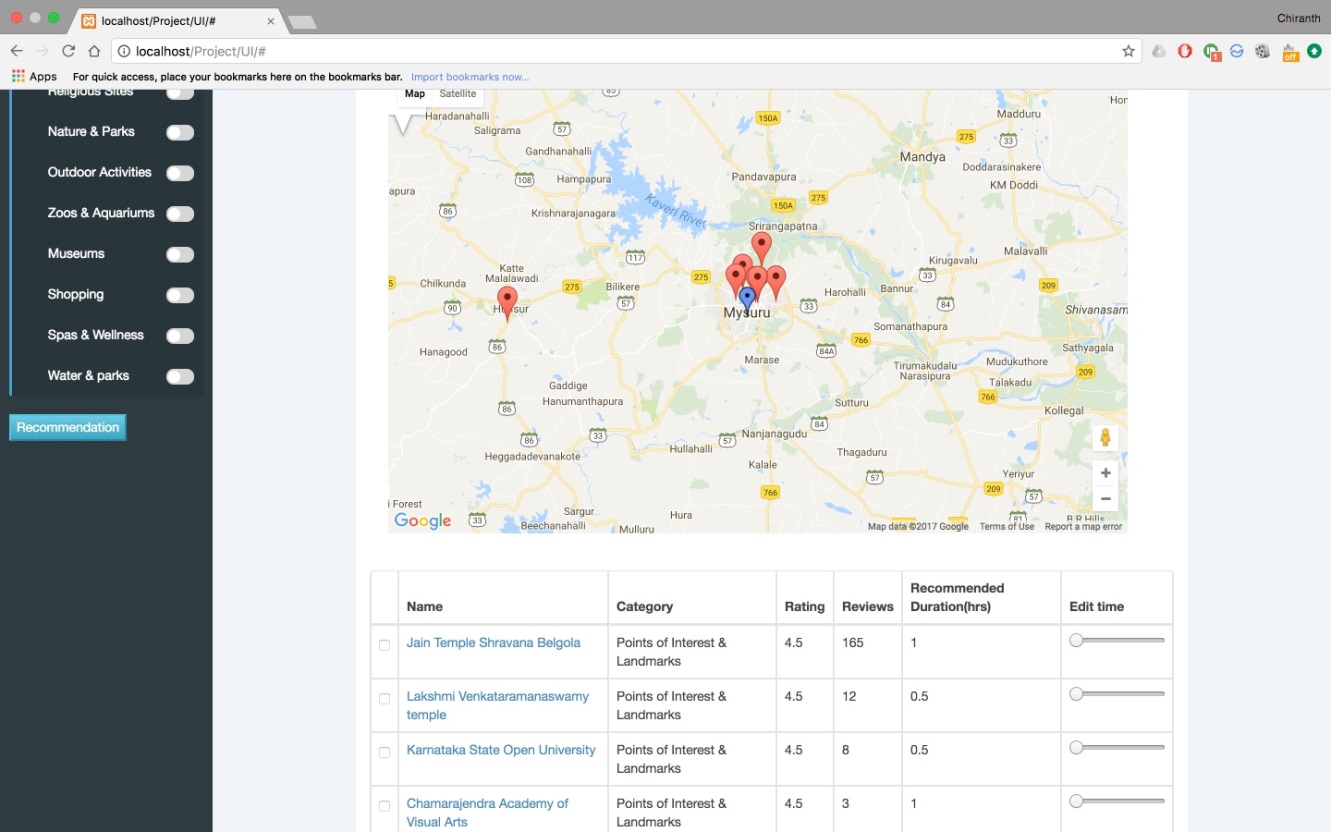


Fig 5F

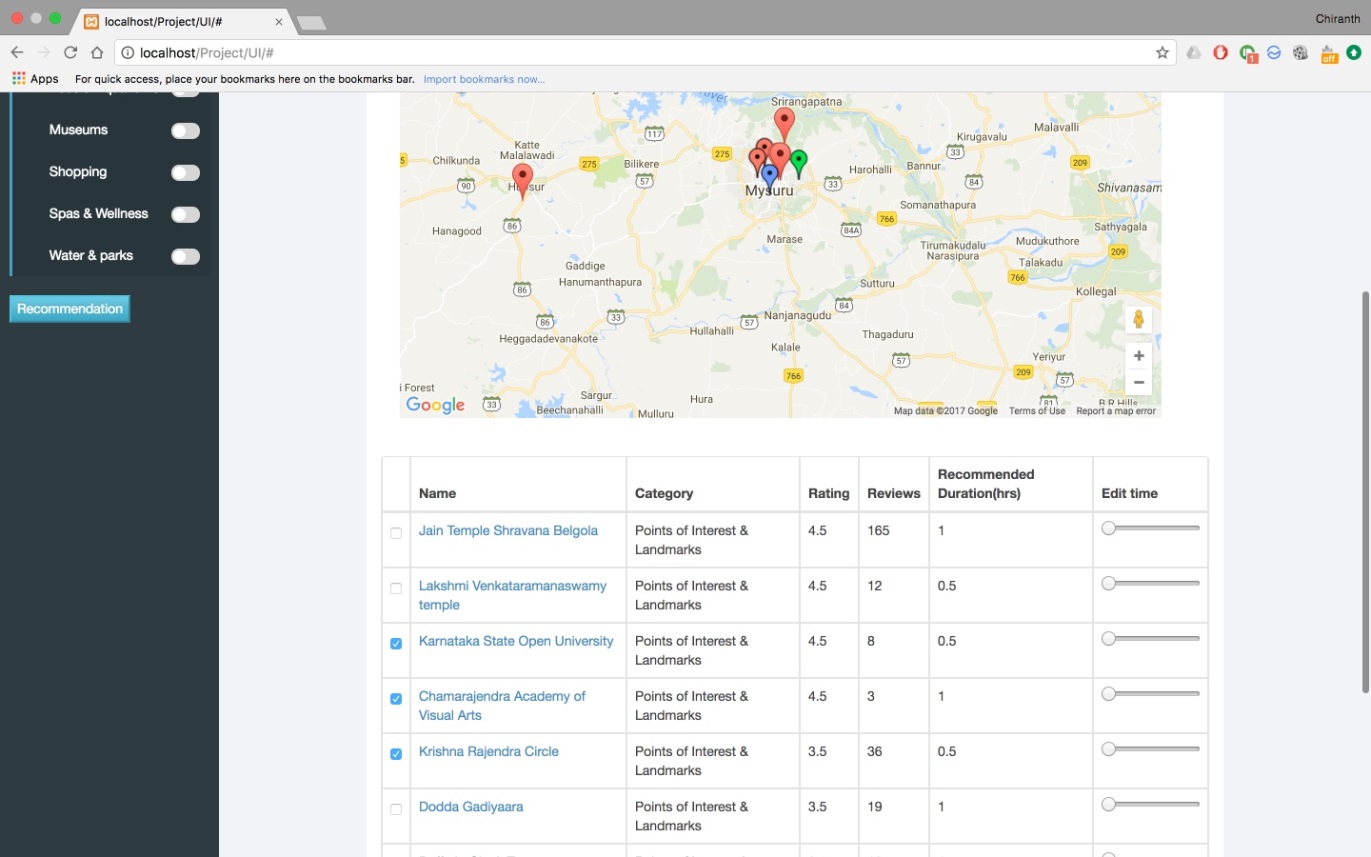


Fig 5G

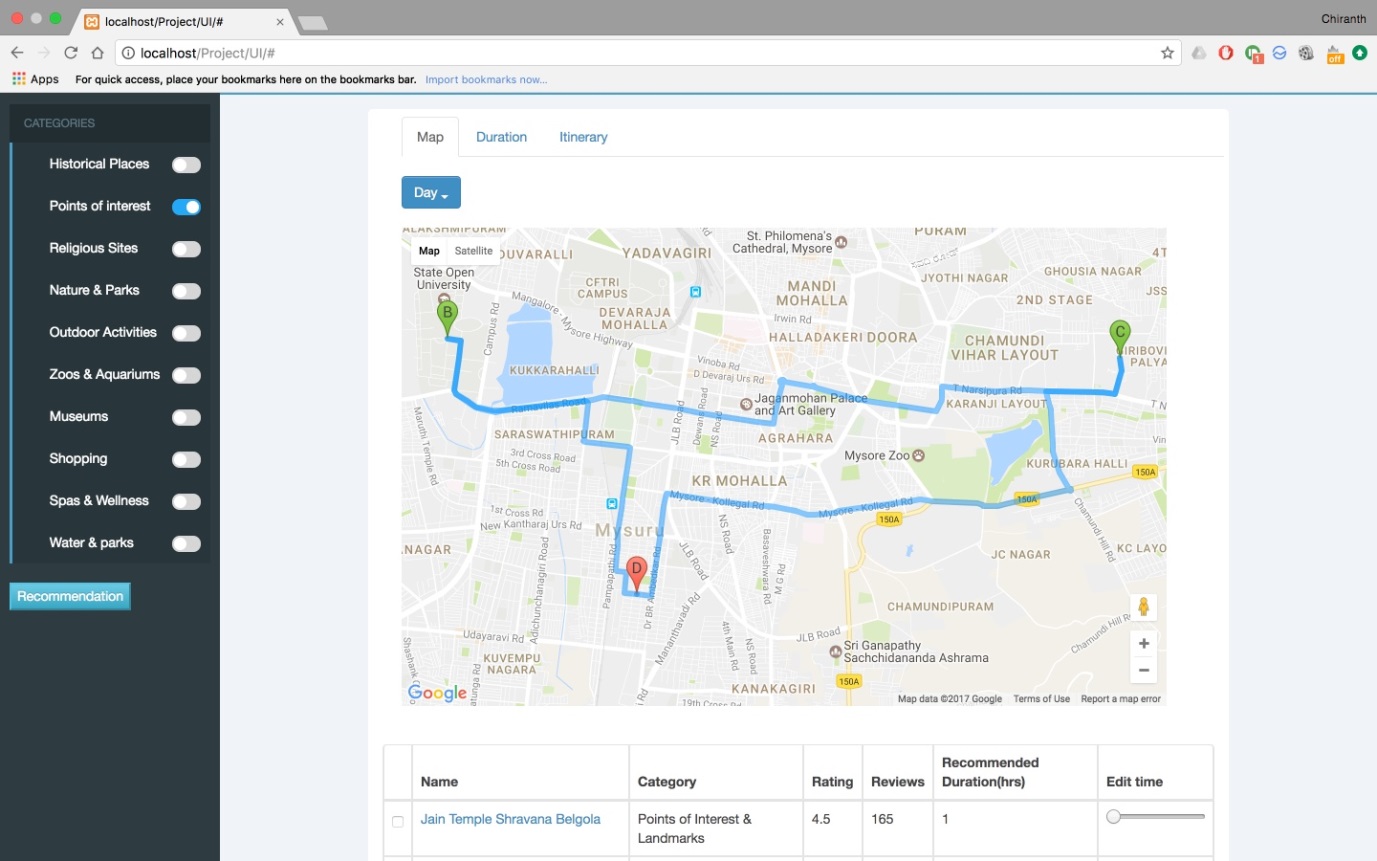


Fig 5H

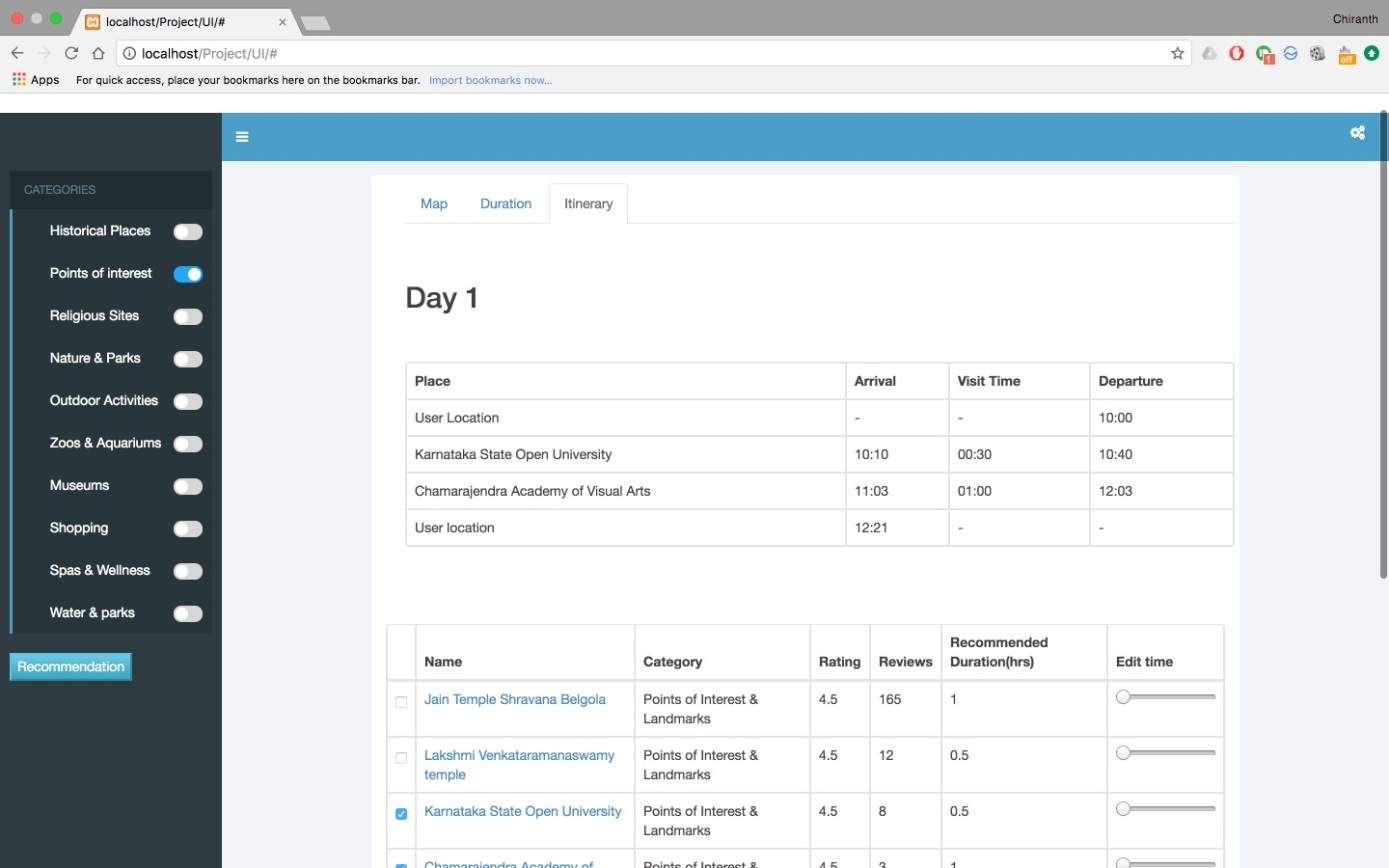


Fig 5I

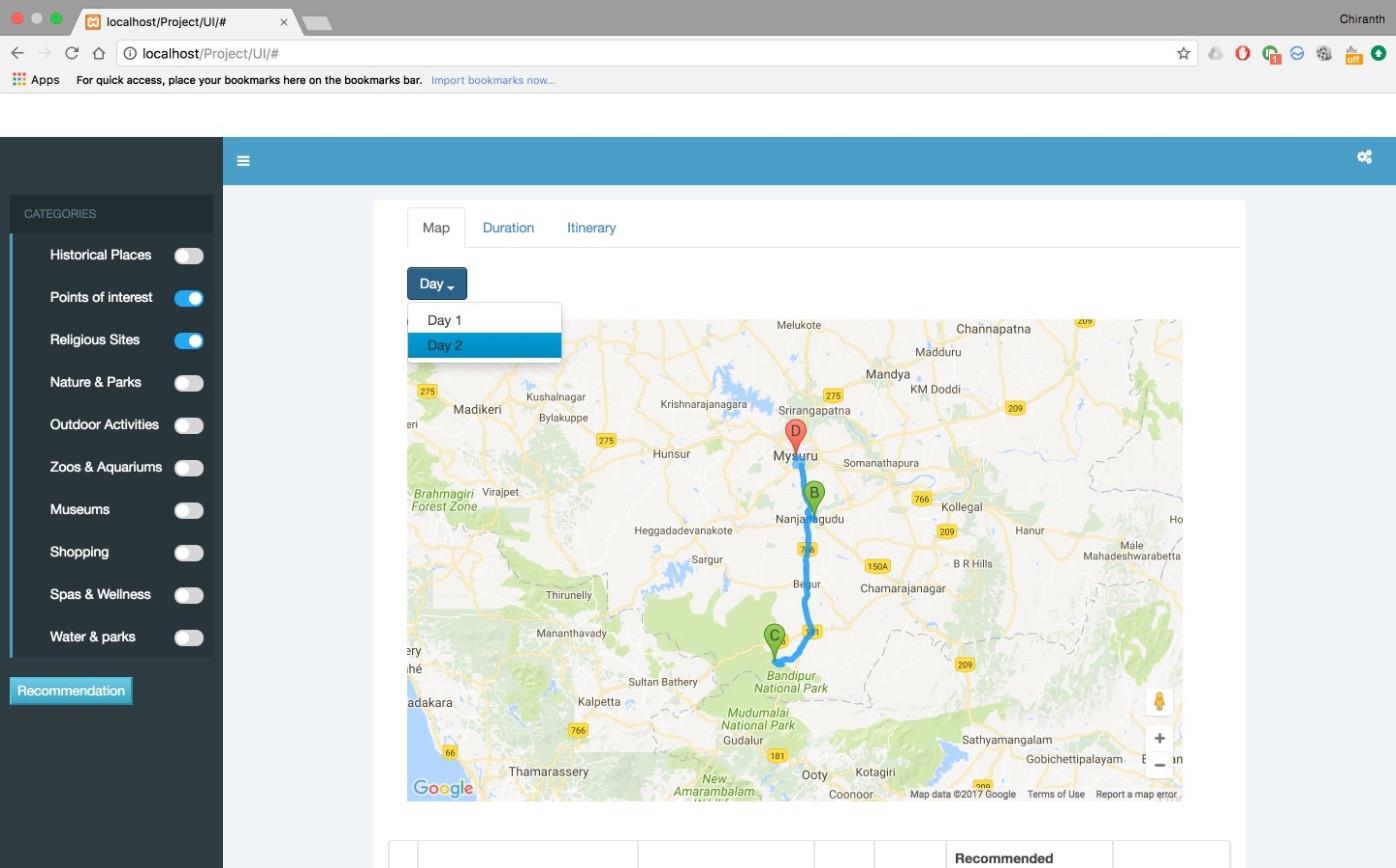


Fig 5J

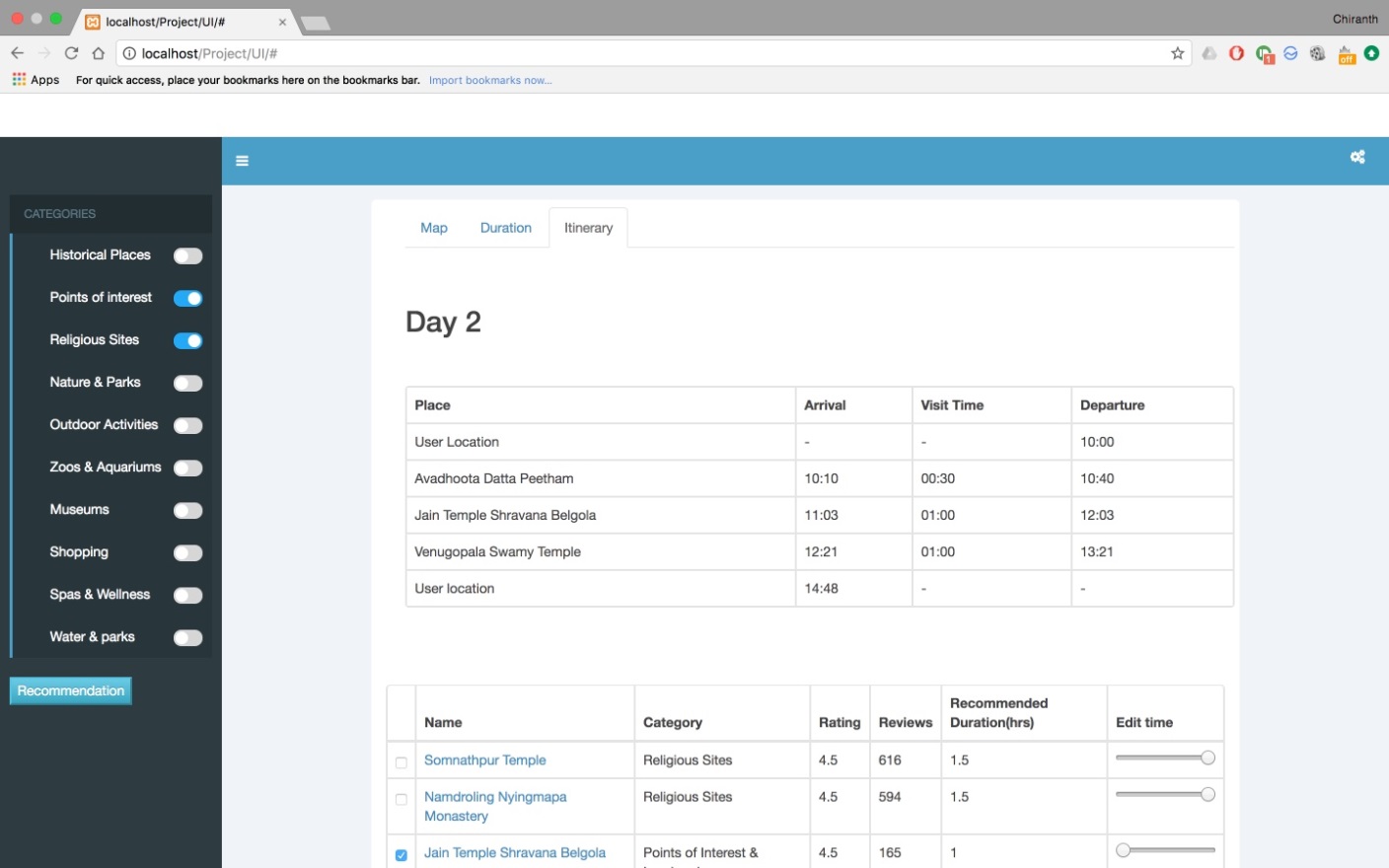


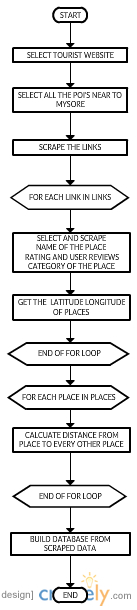
Fig 5K

* 1. **Updated RTM**

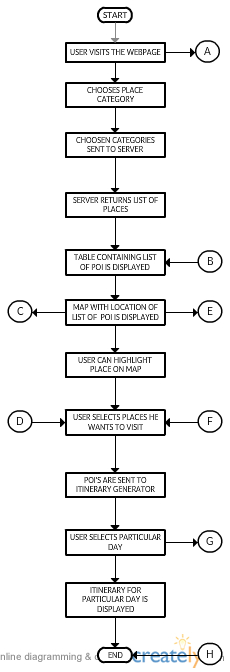
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Requirement ID** | **Requirement Type** | **Requirement Description** | **Test Scenario** | **Results** | **Comments** |
| **FR1** | Functional | Setting dates for trip |  |  |  |
| **FR2** | Functional | Setting location of trip |  |  |  |
| **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 |  |  |  |

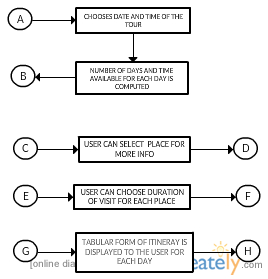
1. **Design**
   1. **Data Flow Diagram**

Initial set up

****

Work flow





* 1. **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 |

1. **Implementation**
   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

Score = x \* Travel time – y \* Rating + z \* 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
  1. **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.

* 1. **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
  1. **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

* 1. **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 |

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 |

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 |

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 |

* 1. **Metrics for Unit Test Cases**

The metrics we used are number of test cases run, number of test cases passed and number of test cases failed.

* 1. **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 |

1. **Testing**

* 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.

* 1. **Test Environment used**

1.      Operating System – Linux and mac

2.      Browser –  Chrome

3.      Python version – 3.5 with the required dependencies

* 1. **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

* 1. **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 |  |

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 |  |

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 |

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 |

* 1. **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
  1. **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 |

1. **Results and Discussions**
   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.

* 1. **Snapshots**

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

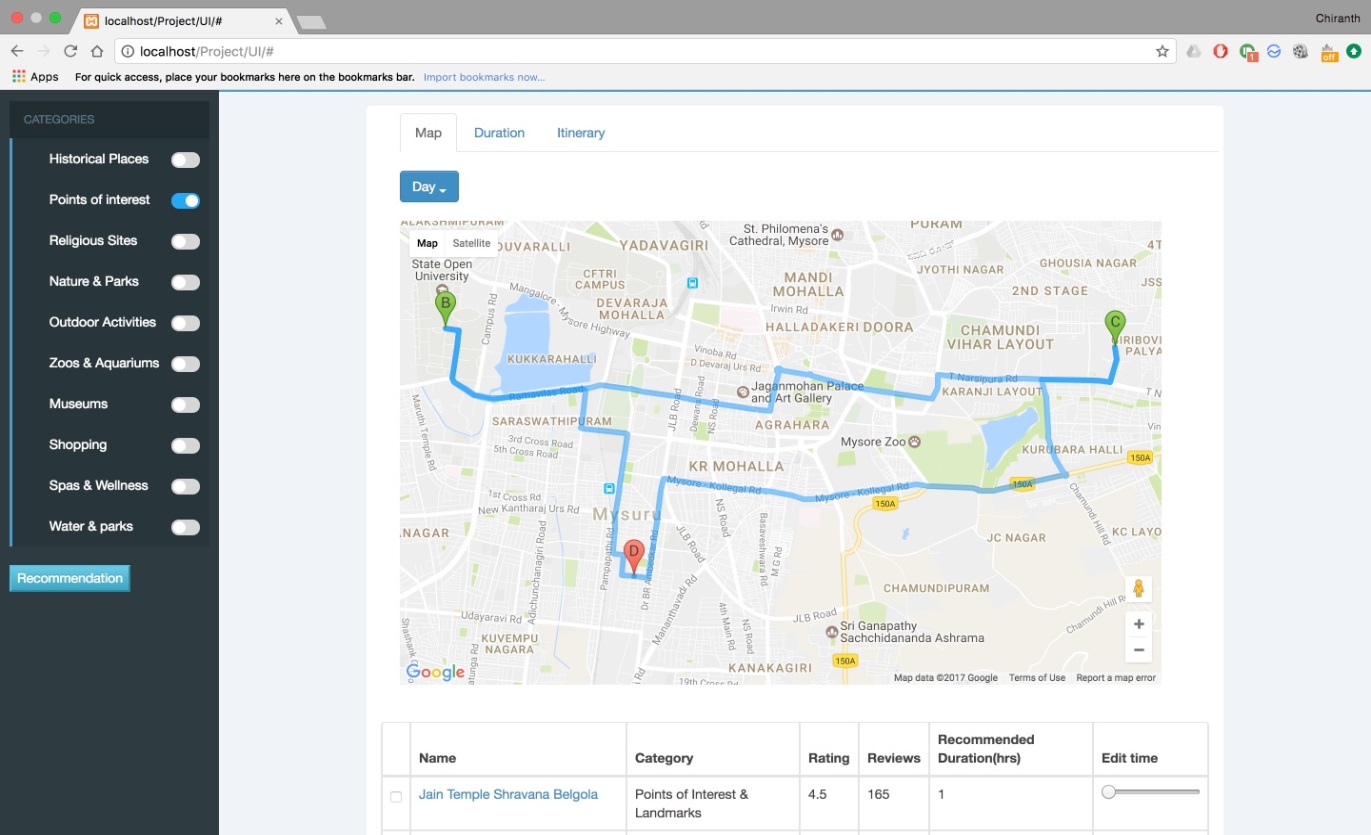


Fig 9A

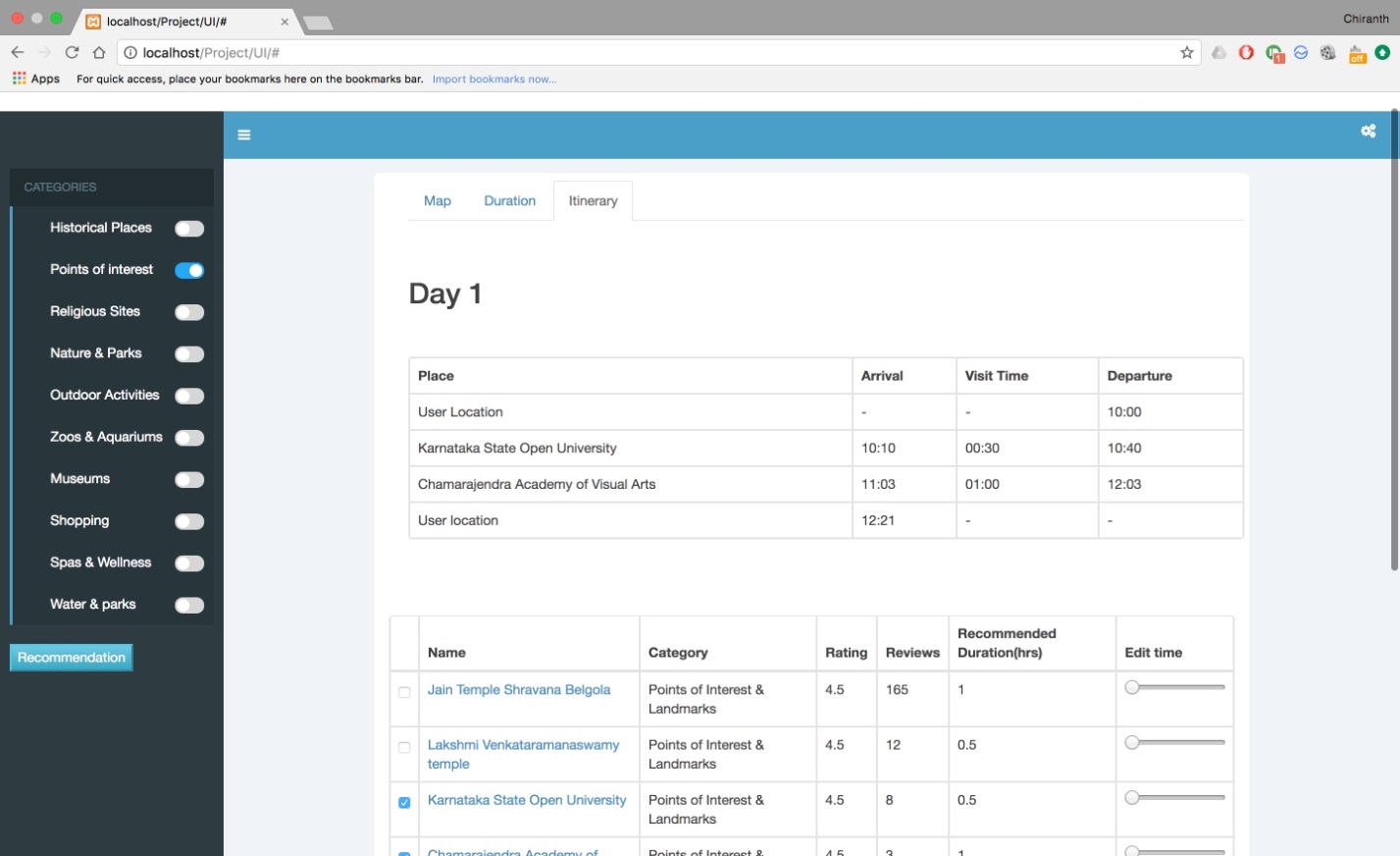


Fig 9B

* 1. **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.

* 1. **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.
  1. **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 |

1. **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.

1. **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](http://www.stackoverflow.com)
   5. [www.python.org](http://www.python.org)
   6. [www.w3schools.com](http://www.w3schools.com)
2. **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