

A project report on

# Smart City Traveler

Master of Technology  
in  
Computer Science & Engineering

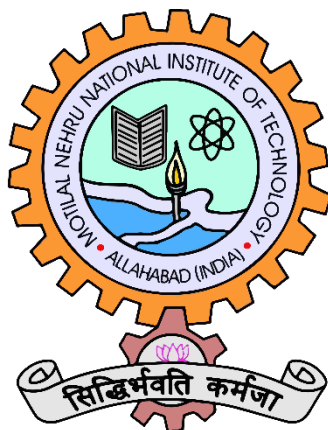
Submitted By

**Saurav Chandra      2018SW13**

**Aman Srivastava      2018IS14**

**Pankaj Joshi          2018IS10**

Under the subject  
**Programming Lab 2   CS22201**



**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING  
MOTILAL NEHRU NATIONAL INSTITUTE OF TECHNOLOGY  
ALLAHABAD 211004**

# CONTENTS

## 1. INTRODUCTION

## 2. WORKING

## 3. FEATURES

## 4. APIs USED

### i. FOURSQUARE API

#### a. PLACES API

#### b. GET VENUE RECOMMENDATIONS

### ii. GOOGLE MAPS DIRECTIONS API

### iii. GOOGLE MAPS API

### iv. GEOMETRY LIBRARY

### v. PHP MAILER API

## 5. FLOWCHART

## 6. MODULES

### i. MODULE I – SIGN IN / SIGN UP

### ii. MODULE II – DASHBOARD

### iii. MODULE III – RESULT

### iv. MODULE IV – MAP VIEW SHOWING PATH

## 7. SOFTWARE REQUIREMENTS

## 8. HARDWARE REQUIREMENTS

## 9. ADVANTAGES

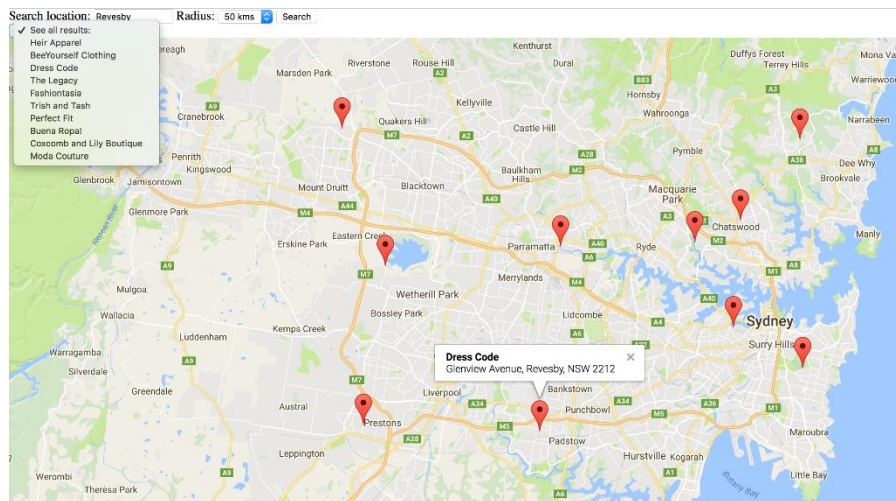
## 10. DISADVANTAGES

## 11. FUTURE SCOPE

## 12. REFERENCES

# 1. INTRODUCTION

Smart City Traveler is a web application to create a schedule for a traveler travelling to a city to explore it by specifying interests and types of places want to visit. Our system then smartly analyzes the questionnaire and creates a schedule for traveler based on provided time and gives a shortest route to reach all places from one to another.



By the name it itself indicates the way in analyzing user's likes and dislikes and is basically used to help a traveler new to the city or anyone who wants to explore a city in the given time period and their interests, the system first fetches user's current location using GPS and then makes use of the **Foursquare API** to get all the locations and places with all their information to sort and place it before the user to make his choice. The places are sorted and selected based on the top rankings by the foursquare and a shortest route is displayed to the user to save their time.

## 2. WORKING

After the Registration, the user is asked some questions helping the system to filter out in searching the places, the places are displayed on a map giving a clear idea of the location and giving the paths from one place to another from the start location to the end location. The system also asks the user whether he/she wants to visit an adventure or water park or a temple or want to have coffee and will show the options based on the rankings and reviews about the place. Since the Traveler may be new to the city not knowing any place, in the map view if the user clicks on the marker, he/she can see the ratings and reviews which are recorded from the Foursquare itself. The best thing is that the system will also forward a mail to the user containing information about his/her plan. The System requires a working internet connection all the time for the application to work.

### 3. FEATURES

- **Registration:** The user has to register into the system with their basic details.
- **Login:** The user has to Login into the System to make use of it.
- **Dashboard:** The user is allowed to see his/her travel plan on dashboard, i.e. the places to visit.
- **Planning Your Day:** The user has to just put their interests and enable their GPS and the rest work is done by system.
- **Map View:** The user will get an interactive map with all the places to visit as marked(pinned) on the map making it easier to see all the location.
- **Shortest Route:** The user can see the shortest route that connects all the places marked saving their money wasted in taking longer route.
- **E-mail:** The system will be sent an e-mail to the user's mailing address with all the details.

### 4. APIs used

- Foursquare API:** The Foursquare Places API provides location-based experiences with diverse information about venues, users, photos, and check-ins. The API supports real time access to places, Snap-to-Place that assigns users to specific locations, and Geo-tag. Additionally, Foursquare allows developers to build audience segments for analysis and measurement. JSON is the preferred response format.

#### **Local search and recommendations**

Foursquare lets users search for restaurants, nightlife spots, shops and other places of interest in their surrounding area. It is also possible to search other areas by entering the name of a remote location. The app displays personalized recommendations based on the time of day, displaying breakfast places in the morning, dinner places in the evening etc. Recommendations are personalized based on factors that include a user's check-in history, their "Tastes" and their venue ratings.

#### **Tips and expertise**

Foursquare eschews the traditional concept of letting users leave long-form reviews, and instead encourages the writing of "Tips" - short messages about a location that let other users know what is good (or bad) there. Tips are limited to 200 characters in length, but can include a URL to link to an external site with more information, and can include a photo.

## **Tastes**

"Tastes" let a user personalise their search experience Foursquare has a defined list of "tastes" in particular food items, styles of cuisine or environmental aspects, which users may add to their profiles to let the app know what they like. The app uses natural language processing to match a user's tastes with the tips at nearby venues that mention them.

## **Location detection**

Foursquare on phone watch Foursquare uses its own proprietary technology, Pilgrim, to detect a user's location. When users opt in to always-on location sharing, Pilgrim is able to understand a user's current location by comparing historical check-in data with the user's current GPS signal, cell tower triangulation, cellular signal strength and surrounding Wi-Fi signals. The app uses the location service to track a user's location in the background, enabling push notifications of things the user might find interesting in their vicinity. It uses this ability to learn about the kinds of places a user likes, based on when and how often they visit different venues. It then uses this data to improve a user's recommendations and gauge the popularity of a venue.

## **Ratings**

In addition to leaving Tips, Foursquare lets users rate venues by answering questions. The questions help Foursquare understand how people feel about a place, including whether or not a user likes the place, how trendy it is, its cleanliness, and its noise level. It also uses these questions to fill in missing venue information such as asking whether the venue takes credit cards, or whether it has outdoor seating.

Foursquare gives each venue a numeric score between 0.1 and 10 to indicate its general popularity when compared to other venues. Scores are calculated automatically factoring in check-in data, explicit user ratings, tip sentiment, foot traffic behaviour and other signals.

## **Lists**

Users can add venues to a personal "to do" list and curated lists to track neighbourhood hot-spots or things to do while traveling.

## **Places API**

It is one of the APIs Foursquare launched for Start-Ups which provides access to Foursquare location intelligence capabilities at a price point between the free tier account and the Enterprise tier account. The Places API allows companies to integrate apps with Foursquare platform data and capabilities. Applications like UBER, Apple Maps, Spotify, Subway, TouchTunes, Snapchat, Twitter etc. are built on Foursquare API.

## Get Venue Recommendations

It returns a list of recommended venues near the current location. For more robust information about the venues themselves (photos/tips/etc.), please see our venue details endpoint. If authenticated, the method will personalize the ranking based on you and your friends.

To request venue recommendations, we need to write the following line:

“GET <https://api.foursquare.com/v2/venues/explore>”

We have used two parameters here: near, section

**near:** A string naming a place in the world. If the near string is not geocodable, returns a failed geocode error. Otherwise, searches within the bounds of the geocode and adds a geocode object to the response. e.g. Chicago, IL

**section:** One of food, drinks, coffee, shops, arts, outdoors, sights, trending, nextVenues (venues frequently visited after a given venue), or topPicks (a mix of recommendations generated without a query from the user). Choosing one of these limits results to venues with the specified category or property.

A sample of the Response is given below:

```
{ "meta": { "code": 200, "requestId": "5ac51ef86a607143de8eg5cb" },
  "response": { "warning": { "text": "There aren't a lot of results near you. Try something more general, reset your filters, or expand the search area." }, "suggestedRadius": 600, "headerLocation": "Lower East Side", "headerFullLocation": "Lower East Side, New York", "headerLocationGranularity": "neighborhood", "totalResults": 230, "suggestedBounds": { "ne": { "lat": 40.724216906965616, "lng": -73.9896507407283 }, "sw": { "lat": 40.72151724718017, "lng": -73.98693222860872 } }, "groups": [ { "type": "Recommended Places", "name": "recommended", "items": [ { "reasons": { "count": 0, "items": [ { "summary": "This spot is popular", "type": "general", "reasonName": "globalInteractionReason" } ] }, "venue": { "id": "49b6e8d2f964a52016531fe3", "name": "Russ & Daughters", "location": { "address": "179 E Houston St", "crossStreet": "btwn Allen & Orchard St", "lat": 40.72286707707289, "lng": -73.98829148466851, "labeledLatLngs": [ { "label": "display", "lat": 40.72286707707289, "lng": -73.98829148466851 } ] }, "distance": 130, "postalCode": "10002", "cc": "US", "city": "New York", "state": "NY", "country": "United States", "formattedAddress": [ "179 E Houston St (btwn Allen & Orchard St)", "New York, NY 10002", "United States" ] }, "categories": [ { "id": "4bf58dd8d48988d1f5941735", "name": "Gourmet Shop", "pluralName": "Gourmet Shops", "shortName": "Gourmet", "icon": { "prefix": "https://ss3.4sqi.net/img/categories_v2/shops/food_gourmet_", "suffix": ".png" }, "primary": true } ] }, "venuePage": { "id": "77298563" } } ] } ] }
```

## ii. Google Maps Directions API

The Directions API is a service that calculates directions between locations using an HTTP request.

With the Directions API, you can:

- Search for directions for several modes of transportation, including transit, driving, walking or cycling.
- Return multi-part directions using a series of waypoints.
- Specify origins, destinations, and waypoints as text strings (e.g. "Chicago, IL" or "Darwin, NT, Australia"), or as latitude/longitude coordinates, or as place IDs.

The API returns the most efficient routes when calculating directions. Travel time is the primary factor optimized, but the API may also take into account other factors such as distance, number of turns and many more when deciding which route is the most efficient.

Sample request and response:

You access the Directions API through an HTTP interface, with requests constructed as a URL string, using text strings or latitude/longitude coordinates to identify the locations, along with your API key.

The following example requests the driving directions from Disneyland to Universal Studios Hollywood, in JSON format:

[https://maps.googleapis.com/maps/api/directions/json?origin=Disneyland&destination=Universal+Studios+Hollywood&key=YOUR\\_API\\_KEY](https://maps.googleapis.com/maps/api/directions/json?origin=Disneyland&destination=Universal+Studios+Hollywood&key=YOUR_API_KEY)

You can test this request by entering the URL into your web browser (be sure to replace YOUR\_API\_KEY with your actual API key. The response returns the driving directions.

## iii. Google Maps API

The Places API is a service that returns information about places using HTTP requests. Places are defined within this API as establishments, geographic locations, or prominent points of interest.

The following place requests are available:

- **Place Search** returns a list of places based on a user's location or search string.
- **Place Details** returns more detailed information about a specific place, including user reviews.
- **Place Photos** provides access to the millions of place-related photos stored in Google's Place database.

- **Place Autocomplete** automatically fills in the name and/or address of a place as users' type.
- **Query Autocomplete** provides a query prediction service for text-based geographic searches, returning suggested queries as users type.

Each of the services is accessed as an HTTP request, and returns either an JSON or XML response. All requests to a Places service must use the `https://` protocol, and include an API key.

The Places API uses a place ID to uniquely identify a place. For details about the format and usage of this identifier across the Places API and other APIs, see the Place IDs documentation.

#### iv. Geometry Library

The Maps JavaScript API geometry library provides utility functions for the computation of geometric data on the surface of the Earth. The library includes three namespaces:

- **spherical** contains spherical geometry utilities allowing you to compute angles, distances and areas from latitudes and longitudes.
- **encoding** contains utilities for encoding and decoding polyline paths according to the Encoded Polyline Algorithm.
- **poly** contains utility functions for computations involving polygons and polylines.

The **google.maps.geometry** library does not contain any classes; instead, the library contains static methods on the above namespaces.

#### Spherical Geometry Concepts

The images within the Maps JavaScript API are two-dimensional and "flat." The Earth, however, is three-dimensional, and is often approximated as either an oblate spheroid or more simply as a sphere. Within the Maps API we use a sphere, and to represent the Earth on a two-dimensional flat surface — such as your computer screen — the Maps API uses a projection.

Within 2D projections, appearances can sometimes be deceiving. Because the map projection necessarily requires some distortion, simple Euclidian geometry often is not applicable. For example, the shortest distance between two points on a sphere is not a straight line, but a great circle (a type of geodesic), and the angles that make up a triangle on the surface of a sphere add up to more than 180 degrees.

Because of these differences, geometric functions on a sphere (or on its projection) necessitate using Spherical Geometry to calculate such constructs as distance, heading, and area. Utilities to calculate these spherical geometric constructs are contained within the Maps API's **google.maps.geometry.spherical** namespace. This namespace provides static methods for computing scalar values from spherical coordinates (latitudes and longitudes).



## Distance and Area Functions

The distance between two points is the length of the shortest path between them. This shortest path is called a geodesic. On a sphere all geodesics are segments of a great circle. To compute this distance, call `computeDistanceBetween()`, passing it two `LatLng` objects.

You may instead use `computeLength()` to calculate the length of a given path if you have several locations. Distance results are expressed in meters. To compute the area (in square meters) of a polygonal area, call `computeArea()`, passing the array of `LatLng` objects defining a closed loop.

## Navigation Functions

When navigating on a sphere, a heading is the angle of a direction from a fixed reference point, usually true north. Within the Google Maps API, a heading is defined in degrees from true north, where headings are measured clockwise from true north (0 degrees). You may compute this heading between two locations with the `computeHeading()` method, passing it two from and to `LatLng` objects.

The following example creates two polylines when you click two points on the map — one geodesic and one "straight" line connecting the two locations — and computes the heading for travelling between the two points:

```
// This example requires the Geometry library. Include the
libraries=geometry
// parameter when you first load the API. For example:
// <script
src="https://maps.googleapis.com/maps/api/js?key=YOUR_API_KEY&libraries
=geometry">

var marker1, marker2;
var poly, geodesicPoly;

function initMap() {
  var map = new google.maps.Map(document.getElementById('map'), {
    zoom: 4,
    center: {lat: 34, lng: -40.605}
  });

  map.controls[google.maps.ControlPosition.TOP_CENTER].push(
    document.getElementById('info'));

  marker1 = new google.maps.Marker({
    map: map,
    draggable: true,
    position: {lat: 40.714, lng: -74.006}
  });
```

```

marker2 = new google.maps.Marker({
  map: map,
  draggable: true,
  position: {lat: 48.857, lng: 2.352}
});

var bounds = new google.maps.LatLngBounds(
  marker1.getPosition(), marker2.getPosition());
map.fitBounds(bounds);

google.maps.event.addListener(marker1, 'position_changed', update);
google.maps.event.addListener(marker2, 'position_changed', update);

poly = new google.maps.Polyline({
  strokeColor: '#FF0000',
  strokeOpacity: 1.0,
  strokeWeight: 3,
  map: map,
});

geodesicPoly = new google.maps.Polyline({
  strokeColor: '#CC0099',
  strokeOpacity: 1.0,
  strokeWeight: 3,
  geodesic: true,
  map: map
});

update();
}

function update() {
  var path = [marker1.getPosition(), marker2.getPosition()];
  poly.setPath(path);
  geodesicPoly.setPath(path);
  var heading = google.maps.geometry.spherical.computeHeading(path[0],
path[1]);
  document.getElementById('heading').value = heading;
  document.getElementById('origin').value = path[0].toString();
  document.getElementById('destination').value = path[1].toString();
}

```

## V. PHP Mailer API

PHPMailer is a code library to send emails safely and easily via PHP code from a web server. Sending emails directly by PHP code requires a high-level familiarity to SMTP standard protocol and related issues and vulnerabilities about Email injection for spamming.

### Class Features

- Probably the world's most popular code for sending email from PHP!
- Used by many open-source projects: WordPress, Drupal, 1CRM, SugarCRM, Yii, Joomla! and many more
- Integrated SMTP support - send without a local mail server
- Send emails with multiple To, CC, BCC and Reply-to addresses
- Multipart/alternative emails for mail clients that do not read HTML email
- Add attachments, including inline
- Support for UTF-8 content and 8bit, base64, binary, and quoted-printable encodings
- SMTP authentication with LOGIN, PLAIN, CRAM-MD5 and XOAUTH2 mechanisms over SSL and SMTP+STARTTLS transports
- Validates email addresses automatically
- Protect against header injection attacks
- Error messages in over 50 languages!
- DKIM and S/MIME signing support
- Compatible with PHP 5.5 and later
- Namespaced to prevent name clashes
- Much more!

Many PHP developers utilize email in their code. The only PHP function that supports this is the mail() function. However, it does not provide any assistance for making use of popular features such as HTML-based emails and attachments.

Formatting email correctly is surprisingly difficult. There are myriad overlapping RFCs, requiring tight adherence to horribly complicated formatting and encoding rules - the vast majority of code that you'll find online that uses the mail() function directly is just plain wrong! Please don't be tempted to do it yourself - if you don't use PHPMailer, there are many other excellent libraries that you should look at before rolling your own - try SwiftMailer, Zend/Mail, eZcomponents etc.

The PHP mail() function usually sends via a local mail server, typically fronted by a sendmail binary on Linux, BSD and OS X platforms, however, Windows usually doesn't include a local mail server; PHPMailer's integrated SMTP implementation allows email sending on Windows platforms without a local mail server.

A simple example:

```
<?php
// Import PHPMailer classes into the global namespace
// These must be at the top of your script, not inside a function
use PHPMailer\PHPMailer\PHPMailer;
use PHPMailer\PHPMailer\Exception;

// Load Composer's autoloader
require 'vendor/autoload.php';

// Instantiation and passing `true` enables exceptions
$mail = new PHPMailer(true);

try {
    //Server settings
    $mail->SMTPDebug = 2; // Enable verbose debug output
    $mail->isSMTP(); // Set mailer to use SMTP
    $mail->Host = 'smtp1.example.com;smtp2.example.com'; // Specify main
and backup SMTP servers
    $mail->SMTPAuth = true; // Enable SMTP authentication
    $mail->Username = 'user@example.com'; // SMTP username
    $mail->Password = 'secret'; // SMTP password
    $mail->SMTPSecure = 'tls'; // Enable TLS encryption, `ssl` also accepted
    $mail->Port = 587; // TCP port to connect to

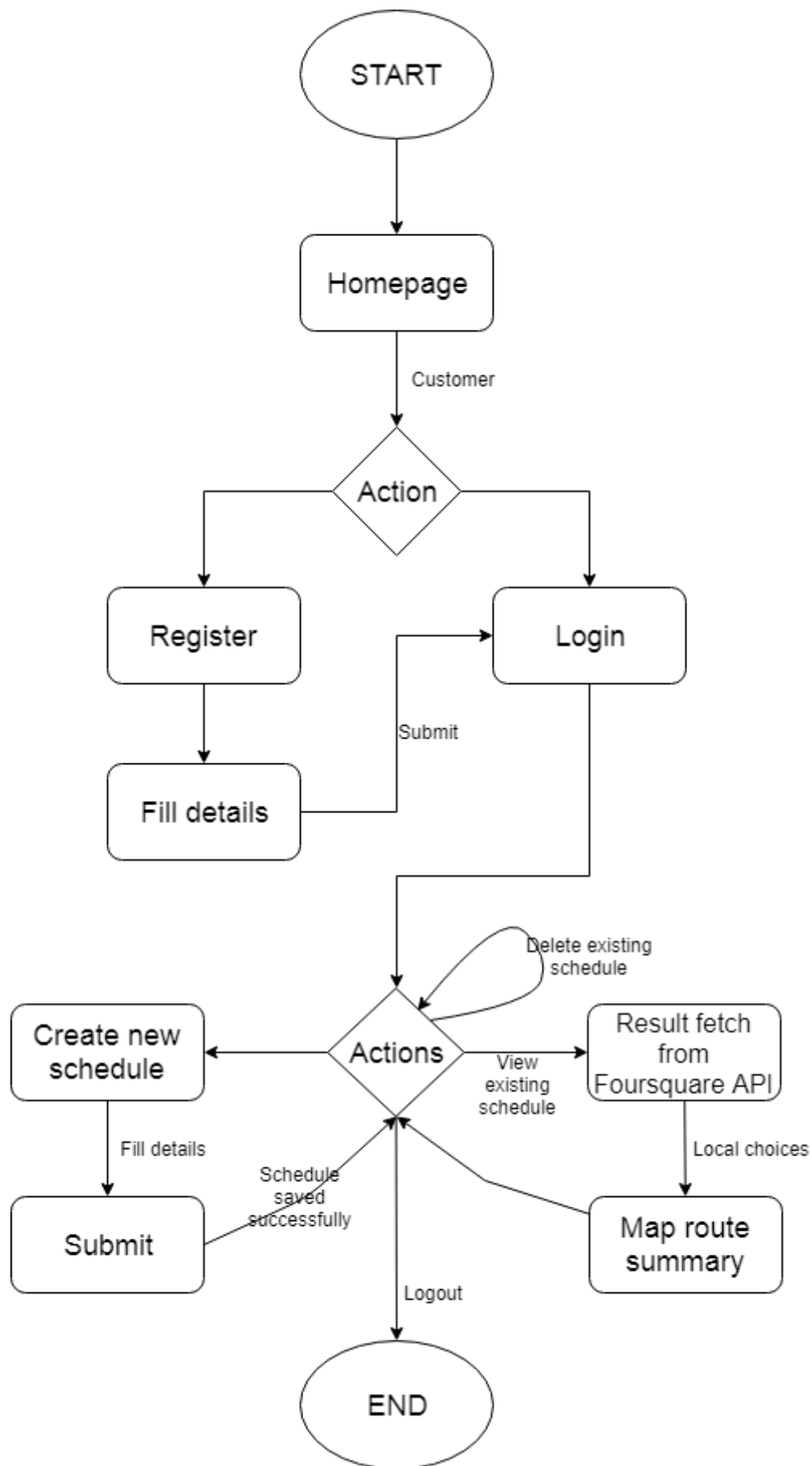
    //Recipients
    $mail->setFrom('from@example.com', 'Mailer');
    $mail->addAddress('joe@example.net', 'Joe User'); // Add a recipient
    $mail->addAddress('ellen@example.com'); // Name is optional
    $mail->addReplyTo('info@example.com', 'Information');
    $mail->addCC('cc@example.com');
    $mail->addBCC('bcc@example.com');

    // Attachments
    $mail->addAttachment('/var/tmp/file.tar.gz'); // Add attachments
    $mail->addAttachment('/tmp/image.jpg', 'new.jpg'); // Optional name

    // Content
    $mail->isHTML(true); // Set email format to HTML
    $mail->Subject = 'Here is the subject';
    $mail->Body = 'This is the HTML message body <b>in bold!</b>';
    $mail->AltBody = 'This is the body in plain text for non-HTML mail clients';

    $mail->send();
    echo 'Message has been sent';
} catch (Exception $e) {
    echo "Message could not be sent. Mailer Error: {$mail->ErrorInfo}";
}
```

## 5. FLOWCHART



## 6. MODULES

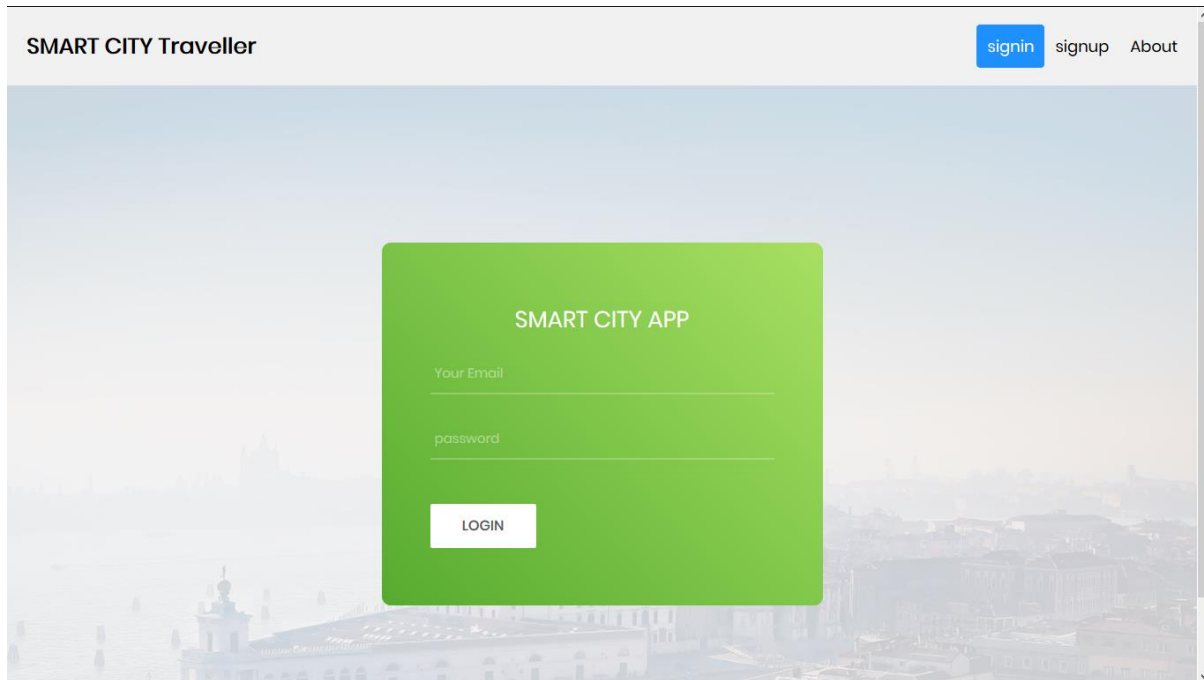
This project is divided into 5 modules:

### a. Module I – Sign in / Sign up

This module contains the pages for sign in and sign up.

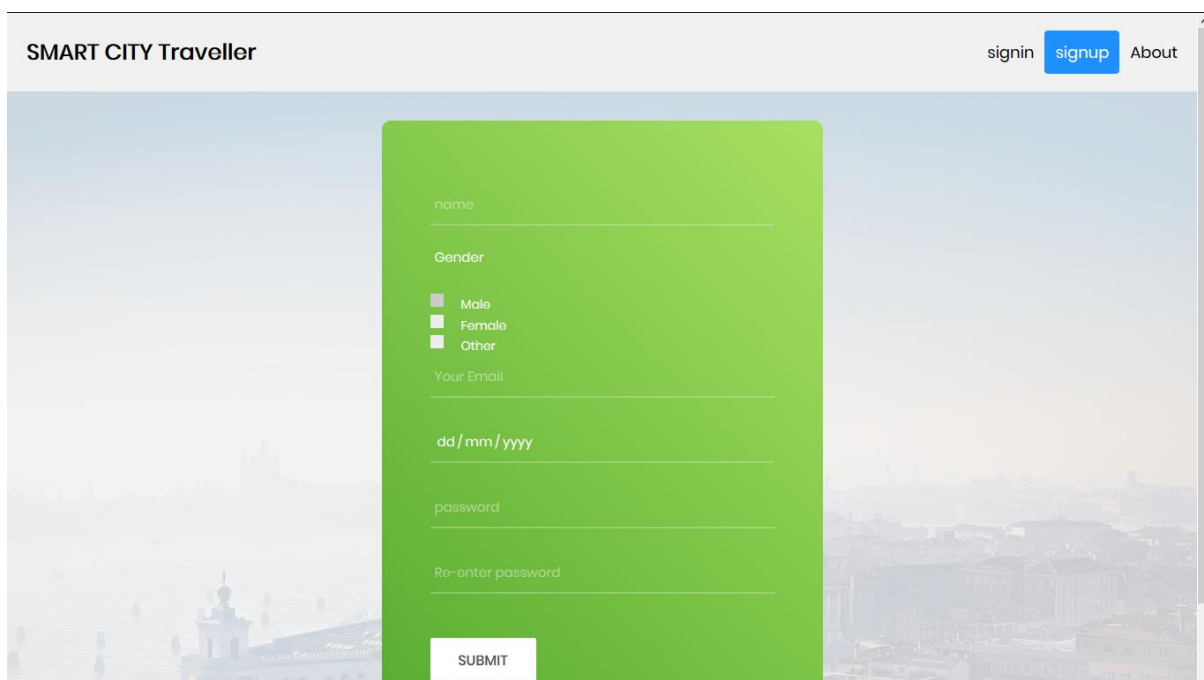
Here user is required to enter basic information like name, gender, dob, etc.

Once user creates an account, they will be redirected to login page and enter the email and password to login.



The screenshot shows the 'SMART CITY Traveller' website. The header includes the site name and navigation links for 'signin', 'signup', and 'About'. The main content area features a green login form titled 'SMART CITY APP'. The form contains two input fields: 'Your Email' and 'password', followed by a 'LOGIN' button. The background is a faded image of a cityscape.

Figure 1: Sign in page



The screenshot shows the 'SMART CITY Traveller' website. The header includes the site name and navigation links for 'signin', 'signup', and 'About'. The main content area features a green sign-up form. The form contains several input fields: 'name', 'Gender' (with radio buttons for Male, Female, and Other), 'Your Email', 'dd/mm/yyyy' (for date of birth), 'password', and 'Re-enter password'. A 'SUBMIT' button is at the bottom. The background is a faded image of a cityscape.

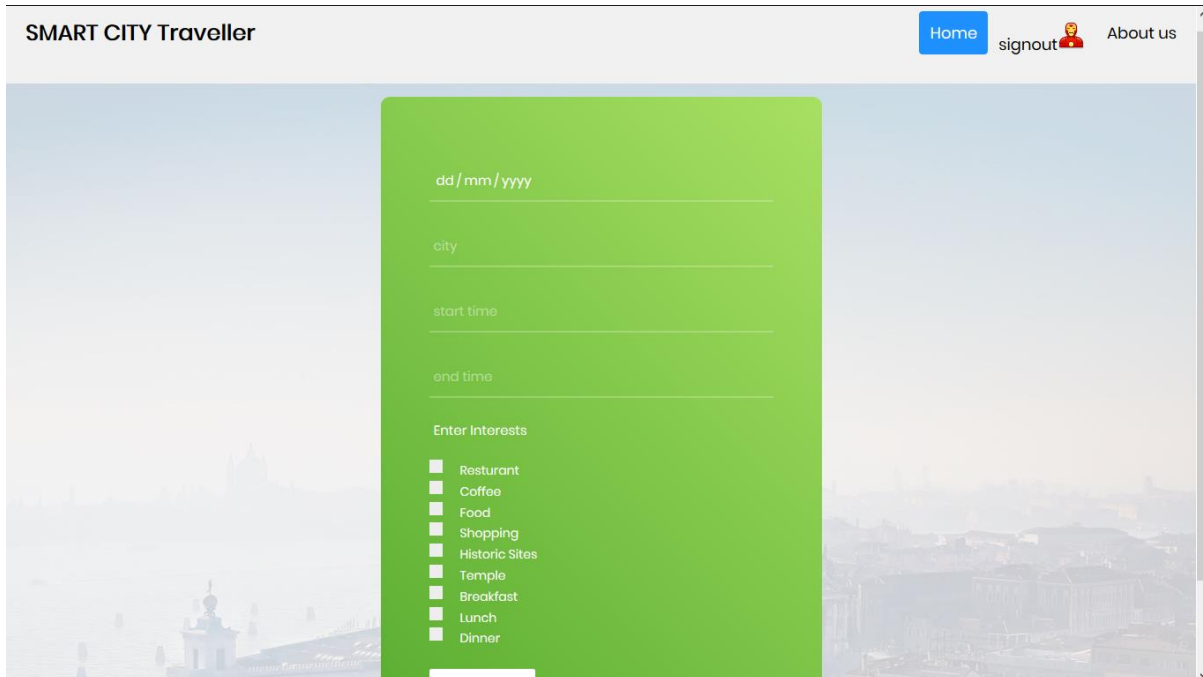
Figure 2: Sign up page

## b. Module II - Dashboard

This module contains Dashboard where you can create, view or erase schedules.

To create a schedule, user will be asked some questions to know user's interest for further processing.

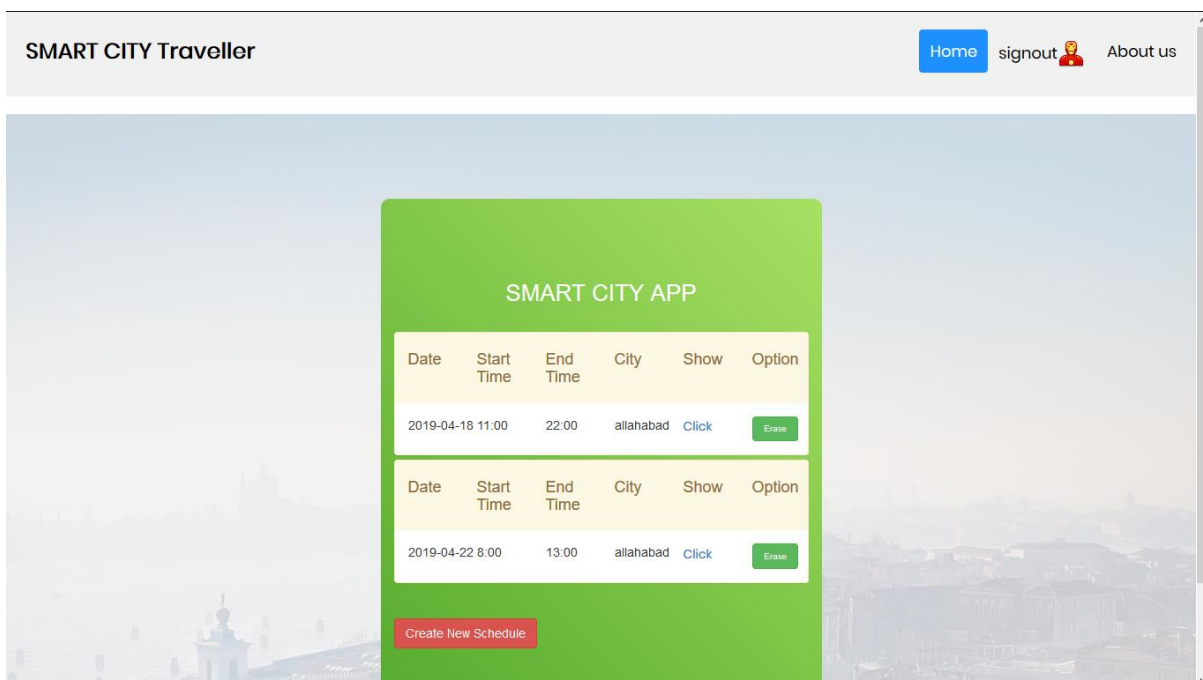
The keywords available in the form are standard keywords accepted by foursquare api, e.g. Restaurant, Temple, Shopping etc.



The screenshot shows the 'Create a schedule' form in the SMART CITY Traveller application. The form is a green box with the following fields and options:

- Date: dd/mm/yyyy
- City
- Start time
- End time
- Enter Interests
  - ☐ Restaurant
  - ☐ Coffee
  - ☐ Food
  - ☐ Shopping
  - ☐ Historic Sites
  - ☐ Temple
  - ☐ Breakfast
  - ☐ Lunch
  - ☐ Dinner

Figure 3: Create a schedule



The screenshot shows the 'Dashboard' in the SMART CITY Traveller application. It displays a table of schedules and a button to create a new schedule.

Date	Start Time	End Time	City	Show	Option
2019-04-18	11:00	22:00	allahabad	<a href="#">Click</a>	<a href="#">Erase</a>

Date	Start Time	End Time	City	Show	Option
2019-04-22	8:00	13:00	allahabad	<a href="#">Click</a>	<a href="#">Erase</a>

[Create New Schedule](#)

Figure 4: Dashboard

Here, User's current location is being fetched in real time to know in which city user is. And this is done by the following code:

```
function get_response(callback)
{
const Http = new XMLHttpRequest();
const url =
"https://api.foursquare.com/v2/venues/search?near="+city+"&query="+QUERY+"&client_id="+CLIENT_ID+"&client_secret="+CLIENT_SECRET+"&v="+YYYYMMDD;
Http.open("GET", url);
Http.send();
Http.onreadystatechange=(e)=>
{
    outp1 = Http.responseText;
    callback();
}
}
```

### c. Module III - Result

In this module, all results are displayed after fetching all locations from user's interests and suggest users to check out the places based on the ratings.

One more attracting feature here is that system will send an email to the user's registered email id with all the details and schedule, using PHP-Mailer API.

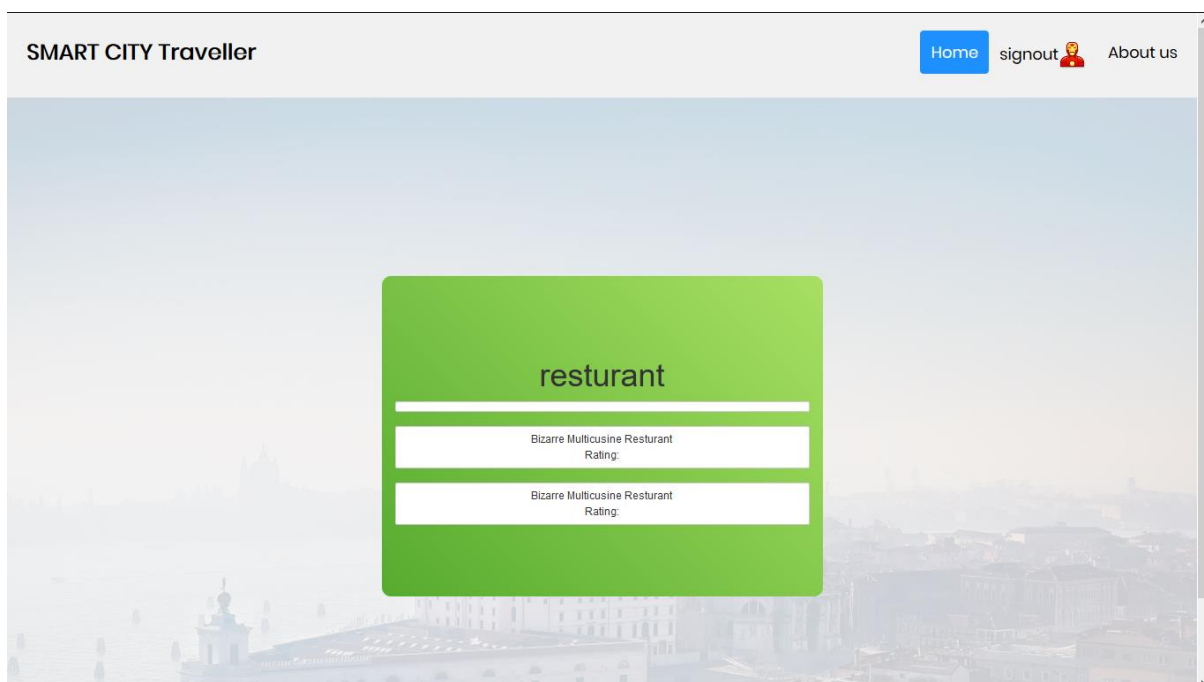


Figure 5: Results



## d. Module IV – Map view showing path

This last module contains the final map view showing all the recommended places with a connected path between them showing the smallest route saving user's cost and time.

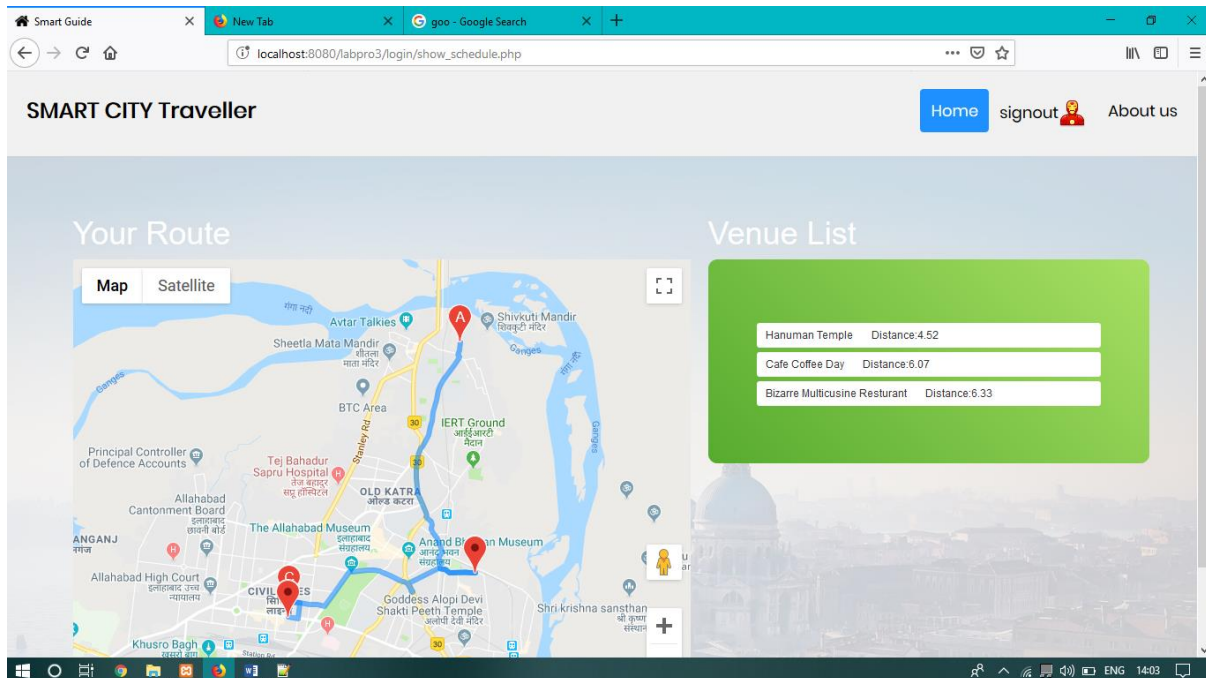


Figure 6: Map view showing connected path between all places

Here Google Maps API is used to display the map and all the pinned places. To display the directions and shortest route and distance between the places, Google Maps Directions API is used. This also fetches the longitude and latitude of all the places.

And to show distance of all the places from user's location, Geometry Library is used. This enables user to know which of recommended places is nearest among them.

To show the nearest places, all the places are sorted on the basis of distance using the following code (internally using insertion sort):

```
//calculates distance between two points in km's
var ct=1;
var mn=2000000;
for(var j=1;j<markers.length;j++)
{
    var p1 = new
google.maps.LatLng(markers[0].coords.lat,markers[0].coords.lng);
    var p2 = new
google.maps.LatLng(markers[j].coords.lat,markers[j].coords.lng);
    distance[ct]=calcDistance(p1, p2);
    if(mn > parseFloat(distance[ct])){
        mn = distance[ct];
    }
}
```

```

        ct++;
    }
    console.log("min distance="+mn);

    for(var i=1; i < ct; i++){
        var k = parseFloat(distance[i]);
        var lat1 = markers[i].coords.lat;
        var lng1 = markers[i].coords.lng;
        var name_loc = loc_name[i];

        var j = i - 1;

        while(j >= 1 && parseFloat(distance[j]) > k){
            markers[j+1].coords.lat = markers[j].coords.lat;
            markers[j+1].coords.lng = markers[j].coords.lng;
            distance[j+1] = distance[j];
            loc_name[j+1] = loc_name[j];
            j--;
        }

        markers[j+1].coords.lat = lat1;
        markers[j+1].coords.lng = lng1;
        distance[j+1] = k;
        loc_name[j+1] = name_loc;
    }
}

```

## 7. SOFTWARE REQUIREMENTS

- Operating System: Windows 7 or higher
- HTML
- CSS
- PHP
- JavaScript
- MySQL
- XAMPP/WampServer
- Google Chrome (or any latest version web browser)

## 8. HARDWARE REQUIREMENTS

- Intel i3 processor or higher (or equivalent AMD processor)
- Minimum Hard Disk - 5 GB
- Minimum Memory - 2GB RAM

## 9. ADVANTAGES

- The data is very accurate and authentic as we take all the data from Foursquare.
- It gives a better view to the users.
- Since the location can be viewed in map, the user can even zoom in and zoom out to get a better view.
- The usage of this application greatly reduces the time required to search for a place.
- The application also leads to quicker decision making with respect to places to visit.
- The user can also find the paths to follow to reach the final destination in map which gives a better view to the users.
- The application will also save user's time and money spent in travelling between the places as system will suggest shortest route.

## 10. DISADVANTAGES

- Since for the data the system is dependent so if anything goes wrong with the foursquare, the system is liable to give wrong data.
- The user will not be able to insert or view details if the server goes down. Thus, there is disadvantage of single point failure.

## 11. FUTURE SCOPE / APPLICATIONS

This system can be used by any company or institution related to tourism which may help their clients or agents to help the users in gaining an advantage over a Paper Based Map. This will also help a user to schedule their places to visit and will choose the best route so that user can visit all the places in minimum required time. This system will also help any user who is new to the city does not want a guide to give tour to the city.

## 12. REFERENCES

- I. <https://developer.foursquare.com/>
- II. <https://developer.foursquare.com/docs/api/venues/explore>
- III. <https://enterprise.foursquare.com/products/places>
- IV. <https://developers.google.com/places/web-service/intro>
- V. <https://developers.google.com/maps/documentation/directions/start>
- VI. <https://developers.google.com/maps/documentation/javascript/geometry>
- VII. <https://github.com/PHPMailer/PHPMailer>