TravelSafe Website
Distributed Systems

Yash Narendra Saraf
UB Person No. 50290453
MS Computer Science and Engineering
School of Engineering and Applied Science
ysaraf@buffalo.edu

Abstract

The era of conventional traveling is a thing of the past; Engineering has upgraded most of the aspects of ones life. Digital Maps is one of those aspects. Maps help us to plan the travel beforehand so that no hassles are faced during travel. Still one of the issue which exits is encountering bad weather. The project combines two important aspects i.e. Maps API along with the Weather API to ideally guide you to the destination so no problems are faced. The website displays weather at the waypoints so that user can be prepared. The use of popular python based Flask library along with MongoDB has been done for the development of website.

1 Client Server Model

The client-server model describes how a server provides resources and services to one or more clients. Examples of servers include web servers, mail servers, and file servers. Each of these servers provide resources to client devices, such as desktop computers, laptops, tablets, and smartphones. Most servers have a one-to-many relationship with clients, meaning a single server can provide resources to multiple clients at one time.

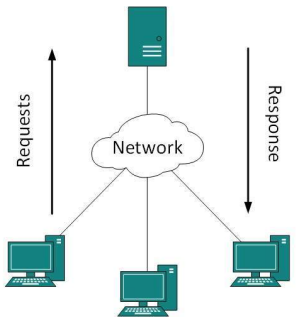

Figure 1: Client Server Model Representation

2 Project Description

The task is to implement a server side system which combines both the responses of Google maps and openweathermap APIs and plots a map in the frontend which gives a representation of the weather data from source to destination. The project has been implemented in 2 phases where in the first phase the APIs are being hit for all the requests and in the second phase using MongoDB the support for caching the data has been added.

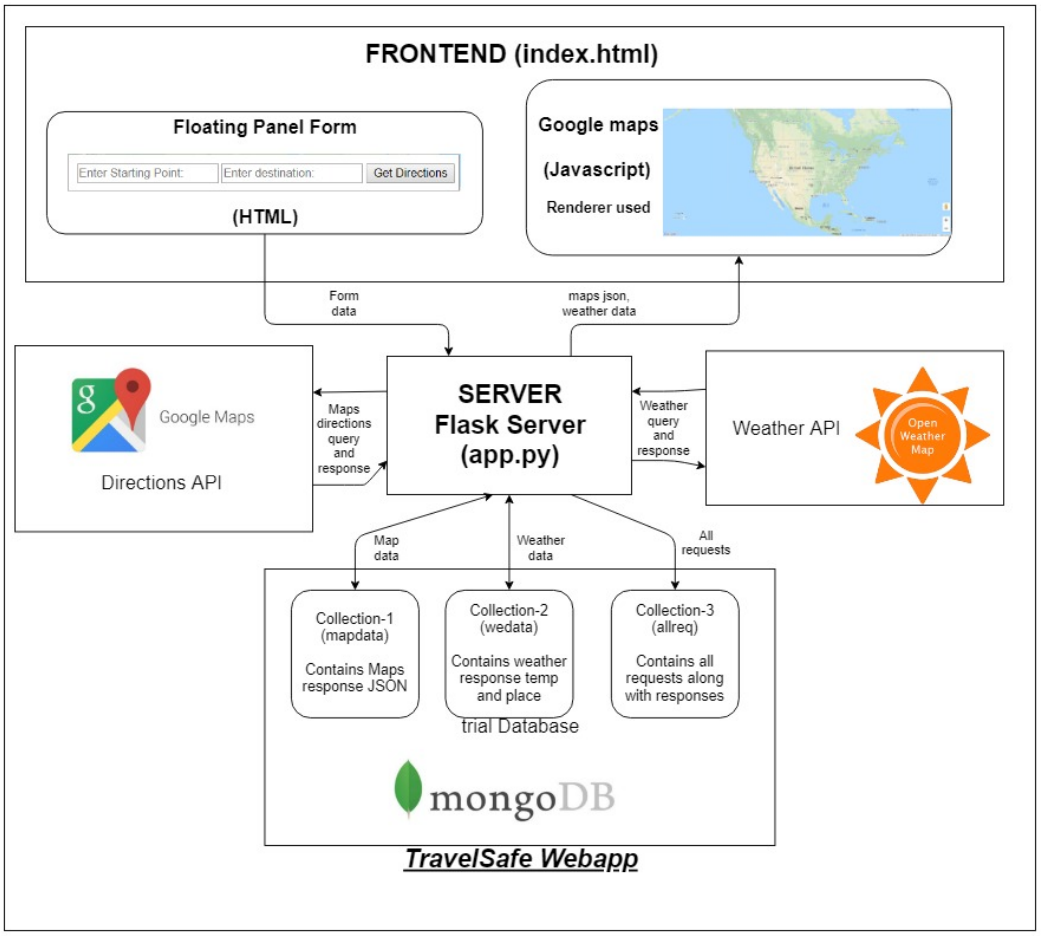

Figure 2: Architectural Model

2.1 Phase-1

In phase 1 a simple client server model has been implemented where for each request API calls are being made. Here there is no cache memory

2.1.1 Working

The webpage is a simple Map display with a floating window which takes user input for the origin and destination.

The form is being submitted using the post method which gets recognized by the flask and the server renders a new page based on that.

So the input is first being processed by the server and the map direction API has been called. The API returns a JSON response which contains all the waypoints in the polyline format. This polyline is being decoded using the polyline decode function which returns latitude longitude pairs.

These latitude longitude pairs are reduced to a limited 10 using simple array iteration and then these pairs are used to call the weather API. The weather API returns JSON response from which place and temperature values are extracted.

The coordinates array along with places array, temperatures array and the maps directions API response has been returned to the frontend which is being plotted. In the frontend first the check variable is checked to see is the response is present or all variables are blank.

If the check result is present, then first all the markers are deleted from the map and the renderer function is called. The renderer functions uses the JSON response and edits it by using typecastRoutes() function. This is because the response of google maps directions API is different for both the HTML URL hit and the JavaScript directions service call. So using the JSON the map is first rendered. Then the coordinate array is passed to showSteps() function which plots all the markers and using attachInstructionText() the text is being added to all the markers.

2.2 Phase-2

Same client server model has been developed as was in phase 1 but here cache is added using the MongoDB database. The support is available for python by including the pymongo library.

2.2.1 Working

The web-page is a simple Map display with a floating window which takes user input for the origin and destination.

The form is being submitted using the post method which gats recognized by the flask and the server renders a new page based on that. The server first checks if the same request is present in the Database mapdata collection. If the record is present it retrieves the JSON file and compares the time-stamp for that to the current time. If this time is less than 24 hours, then this response is used.

If the time is greater than 24 hours, then the server will discard the last entry and call the Maps direction API and create a new entry in the database and also the same response is used.

If the request doesnt exist in the database, then the request is made to the Google maps direction API and the response is then stored in the mapdata collection and the same response is used.

Using the obtained JSON, first the polyline is extracted and decoded which returns a list of latitude and longitude pairs.

This list is then reduced to 10 equidistant points using simple array iterations. Then every pair of this list the place is checked in the wedata collection of the database and if the value is present then the time-stamp is compared to current time and if it is less than 6 hours then this weather data is used for that particular lat-long pair.

If the time difference is greater than 6 hours than the old values are deleted in database and new entry is made based on the new API call.

If the value is not present then the new API call is made and the response is stored in the database and also used.

The weather API returns JSON response from which place and temp values are extracted. The coordinates array along with places array, temperatures array and the maps directions API response has been returned to the front-end which is being plotted.

3 Technology Stack

The project has been implemented using Flask backend which is a python based micro-framework and used simple HTML and Javascript frontend. The details for all of them are as follows.

- Python 3.6, with following libraries installed.
  - pymongo - for mongodb support
  - gmaps - for google maps direction api call
  - googlemaps - for google maps direction api call
  - polyline - to decode the overview_polyline obtained in google maps response
  - Flask - Server setup using python
- MongoDB 4.0

Table 1: Cost function Comparisons:(All time in microseconds)

Request-Source	Request-Dest	Phase 1 time	Phase 2 time
Buffalo	New York	603977	15989
Buffalo	Boston	953427	16981
Buffalo	Chicago	883015	17921
Buffalo	Las Vegas	983671	20899
New York	Chicago	174582	17993
New York	Las Vegas	538564	20040
New York	Boston	621906	17937
New York	Dallas	665946	21212
Dallas	Las Vegas	561108	18920
Dallas	Boston	444375	53050
Average of all	————	643057	22094

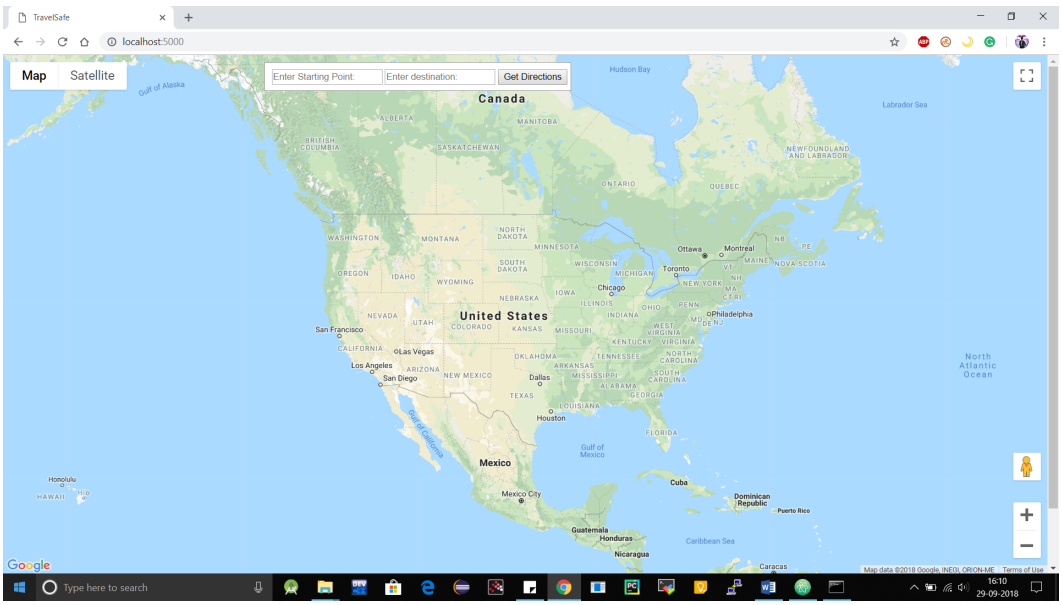

Figure 3: Front Page of the URL

4 Cost Function

For evaluating the cost function time has been used as the key. The time difference between the request made and the rendering has been logged and the plot has been obtained for a model which uses a database and which doesn't use a database. The results has been summarized in the following table. All the requests made when using the database are shown for those whose responses were present in the database.

The above table 1 shows that the time required for getting data from cache is way faster than obtaining the data after making the API call to the server. Approximately retrieving from the database is 30 times faster than retrieving using API.

5 Results

These are the final results based on the submitted code. The screen shot images for the start screen and response page has been added to show how the application works.

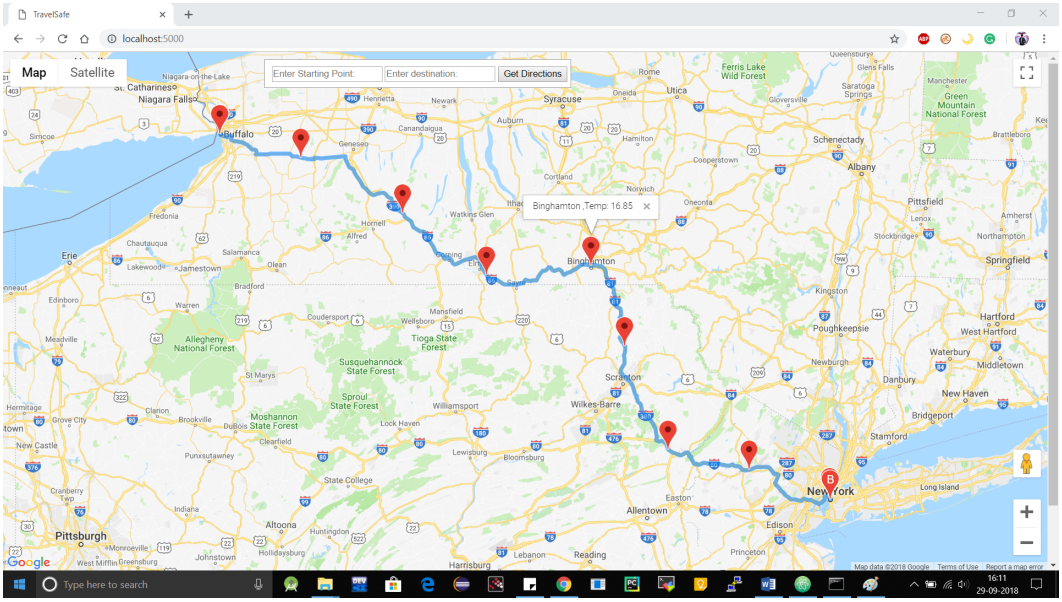

Figure 4: Response Page for query Buffalo to New York

The figure 3 represents start page where the page has only map div and a floating window which takes in user inputs. As soon as the submit button is clicked the second page loads up which displays the route and the markers which displays the temperature and place name.

6 Conclusion

The project was to understand the client server model and integrate multiple API. Also cost function was calculated for both the phase 1 and phase 2 approach and it is observed that retrieving from the database takes less than a second whereas getting data from the server may take up a lot more time depending upon the server response time which may vary.

References

[1] PyMongo Documentation, *https://API.mongodb.com/python/current/*. 2017

[2] Flask Documentation, *http://flask.pocoo.org/*. 2017

[3] Client Server model description, *https://techterms.com/definition/client-server_model*.

[4] Using JSON response of URL call with javascript renderer, *https://gis.stackexchange.com/questions/15197/google-maps-v3-in-javascript-API-render-route-obtained-with-web-API*.

[5] Google Maps Directions API Documentation, *https://developers.google.com/maps/documentation/directions/intro*.