Distributed System Project 1 Report

Web application with Google map API and weather API

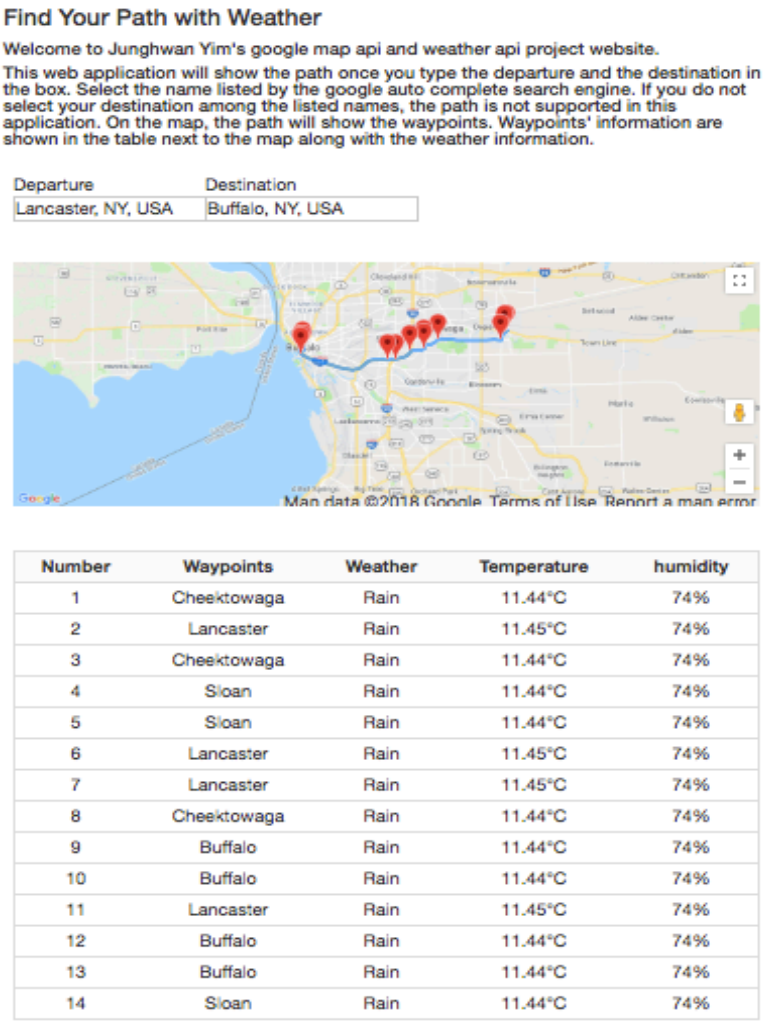
Junghawn Yim
50173046

1. Version 1

My version 1 is at **parse1.html** the zip file. First, input box for Departure and Destination was made to take input, and made the table for showing information of the waypoints. After making input boxes and table, add the map with following instruction in the Google Map API homepage sample, https://developers.google.com/maps/documentation/javascript/examples/places-autocomplete-directions. And Google Place API is used for finding the destination and departure before implementing finding route from departure to destination. With the google Place API, the auto-complete function have been implemented. It is automatically finding, and suggesting function the place where expected to you want through map current position and typed string. When choosing one among the suggested place, the place is set to departure or destination. By using set departure and destination, find the root through the Google Direction API. The API also supports the marker which marks the waypoints between the Departure point and Destination. Through the properties and functions which offered from the API, implement the function of showing path and waypoints with the markers.

After implementing the functions, use the weather API to retrieve the information of waypoints' weather, for example, temperature, humidity, place name and general weather. The API returns the JSON file which contains the weather information at waypoints through URI containing latitude and longitude which is the eigen numbers indicating the specific position on the Earth. From the Google API's marker instance extract the latitude and longitude values and get JSON from the weather API URL by GETJSON function of ajax. And extract the values of temperature, humidity, weather, and name by parsing JSON file. The information is indicated on the table.

For the clear and rounding design, use the bootstrap has been used, and other complicated pictures and image is not inserted to the website because this web application of main purpose is delivering knowledge or information clearly to clients. In the .zip file, css, fonts, and js directories are about the bootstrap files.



Number	Waypoints	Weather	Temperature	humidity
1	Cheektowaga	Rain	11.44°C	74%
2	Lancaster	Rain	11.45°C	74%
3	Cheektowaga	Rain	11.44°C	74%
4	Sloan	Rain	11.44°C	74%
5	Sloan	Rain	11.44°C	74%
6	Lancaster	Rain	11.45°C	74%
7	Lancaster	Rain	11.45°C	74%
8	Cheektowaga	Rain	11.44°C	74%
9	Buffalo	Rain	11.44°C	74%
10	Buffalo	Rain	11.44°C	74%
11	Lancaster	Rain	11.45°C	74%
12	Buffalo	Rain	11.44°C	74%
13	Buffalo	Rain	11.44°C	74%
14	Sloan	Rain	11.44°C	74%

2. Version 2

The PHP and MySQL has been used for server and database, and use the Bitnami MAMP web server to construct the server and it is already connected with the MySQL server from https://bitnami.com/stack/mamp. To store the information of waypoints of already searched path between departure and destination at least once, the database is established with MySQL. Connected to database with the command ./mysql -hlocalhost -uroot -p with password, 123456, set when establish the database. And create database which name is 'googleweather' with command 'CREATE DATABASE googleweather;' After selecting the database by the command 'use googleweather;', made two tables which are 'path', and 'waypoints'. Here, the path is containing the departure, destination, and waypoints array information. The waypoints tables have longitude, and latitude of marker, waypoints, as well as the information of waypoints' weather, temperature, name, and humidity which searched from weather API. The database tables' properties is like the next. Here the lng and lat is change their type to float. The data is used to webpage as well as the data stored in database, make the php files to connect html and database. There is four php files. **The parse2.php** is the main server file to manage all system therefore, most of parse.html codes is contained to parse2.php file. The **check.php** is the server file to access data base. It is determining whether the current accessing path with departure and destination, and if so, returns JSON file type of the data of the paths containing waypoints' data, and if not, search data from API like the version 1. **Upload.php** is uploading the new path and regard information with the path to database, and the **uploadpoint.php** is uploading the new waypoints' information to database. Due to these database structure and php server files, search the path between departure and destination where already searched at least one time without the downloading from Google Direction API, and Whether API.

```
mysql> DESC path
    -> ;
+-----------+---------------+------+-----+---------+-------+
| Field     | Type          | Null | Key | Default | Extra |
+-----------+---------------+------+-----+---------+-------+
| orig      | varchar(80)   | YES  |     | NULL    |       |
| dest      | varchar(80)   | YES  |     | NULL    |       |
| waypoints | varchar(2048) | YES  |     | NULL    |       |
+-----------+---------------+------+-----+---------+-------+
3 rows in set (0.00 sec)

mysql> DESC waypoints;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| name        | varchar(80) | YES  |     | NULL    |       |
| weather     | varchar(40) | YES  |     | NULL    |       |
| temperature | varchar(40) | YES  |     | NULL    |       |
| humidity    | int(11)     | YES  |     | NULL    |       |
| lng         | int(11)     | YES  |     | NULL    |       |
| lat         | int(11)     | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

How to Deploy

To deploy the Version 2 files, recommend to

a) install the bitnami MAMP web server from https://bitnami.com/stack/mamp. And install all component with database which password has to set to '123456'.

b) After then in the command line, move to the below directory

```
/Applications/mampstack-5.4.40-0/mysql/bin
```

c) And type

```
./mysql -hlocalhost -uroot -p 123456
```

d) And type MySQL instruction,

CREATE DATABASE googleweather;

```
CREATE TABLE waypoints(
    -> name VARCHAR(80),
    -> weather VARCHAR(40),
    -> temperature VARCHAR(40),
    -> humidity int(11),
    -> long float(7,4),
    -> lat float(7,4)
    -> );

CREATE TABLE path(
     -> orig VARCHAR(80),
     -> dest VARCHAR(80),
     -> waypoints VARCHAR(2048)
     -> );

After setting of Database,
```

e) place the source files from the zip file to `/Applications/mampstack-5.4.40-0/apache2/htdocs`

f) with changing Parse2.php file name to index.php.

After then,

g) enter to localhost:8080 in the address input box in internet browser.

Compare the two implementations of versions 1 and 2. Assign cost functions c1, c2, and c3 to the API accesses and the DB access. Discuss the saving achieved by version 2. This can be done in the project report.

3. Discussion

The version 1 and 2 are web application using Google Map API and Weather API offered from

1. Maps API: https://developers.google.com/maps/web-services/client-library
2. Weather API: https://openweathermap.org/api

But, they have difference in whether using Database or not in case that search again the same path have been searched before.

Let say the database access cost function is c1, and
the Google map API access cost function is c2, and
the Weather API access cost function is c3,

We can assume that c1 could be lesser than c2 and c3 because database's physical distance is closer than APIs' servers within high probability.

The cost function of version 1 and 2 is equivalent to c2+c3 in the case search path which have not searched before. In the case that search path which have been searched before, the cost function of version 1 is same as before, c2+c3, but, the cost function of version 1 is c1.

When we compare the two cases are distributed half and half,

Cost function of version 1: $0.5*(c2+c3) + 0.5*(c2+c3) = (c2+c3)$
Cost function of version 2: $0.5*(c2+c3) + 0.5*(c1) = 0.5*(c2+c3+c1)$

When $2*c1$ is lesser than $c2+c3$, the version 2 is efficient even though offering same service.

Deployed Site : http://18.217.117.213/dashboard/