Distributed Systems (CSE 586)

PROJECT REPORT

ON

WAYPOINTS WEATHERINFO

BY

SANDEEP KOTHAPALLY

INTRODUCTION:

The objective of the project is to design and implement a complete service-based distributed web application that provides users enhanced information about a travel route.

BACKGROUND:

Typical “map” applications provide the route, towns/cities along the route, alternate routes etc. In this application we want to show more than that. When a user inputs “From” and “To” locations on this application, we figure out the weather predicted for a given route and display it in a user-friendly form. We also save all the user requests (user input) for quick, efficient and cost-effective response to popular (and repeated) requests.

DESIGN & IMPLEMENTATION:

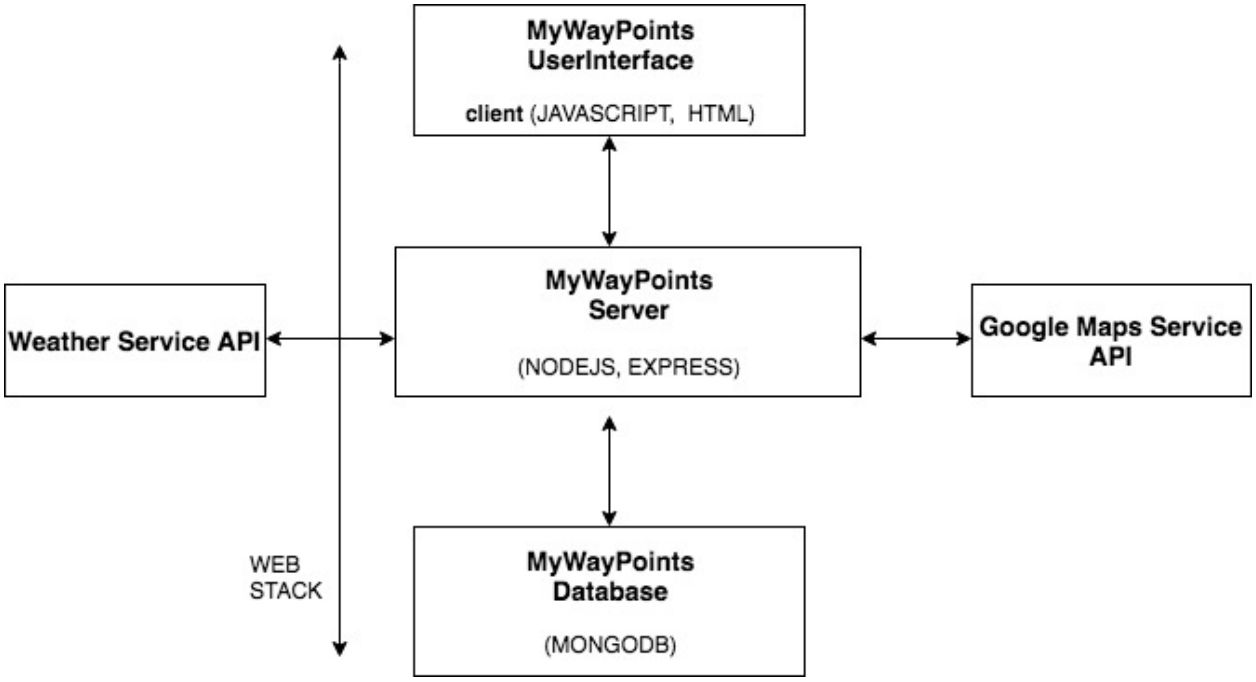


Figure 1: Architecture

TECHNOLOGIES USED:

This project is implemented using the MONGODB, NODEJS, EXPRESS software stack, which is free, open source Java script software stack useful to build dynamic websites and web applications. The details of choice of technologies are as follows.

- Node.js API IS used for server-side application development
- NPM package Express.js is used for server
- Mongo Db IS used for database application development
- JS & HTML is used for client-side implementation
- Mongoose is used for Object data modelling
- The application user interface homepage consists of a

  page to input source, destination.

- Java Script Maps API and Weather API will be used to fetch

  the routes and weather information on the fly

The relevant route and weather info of selected cities in the route will be displayed as output on request submit

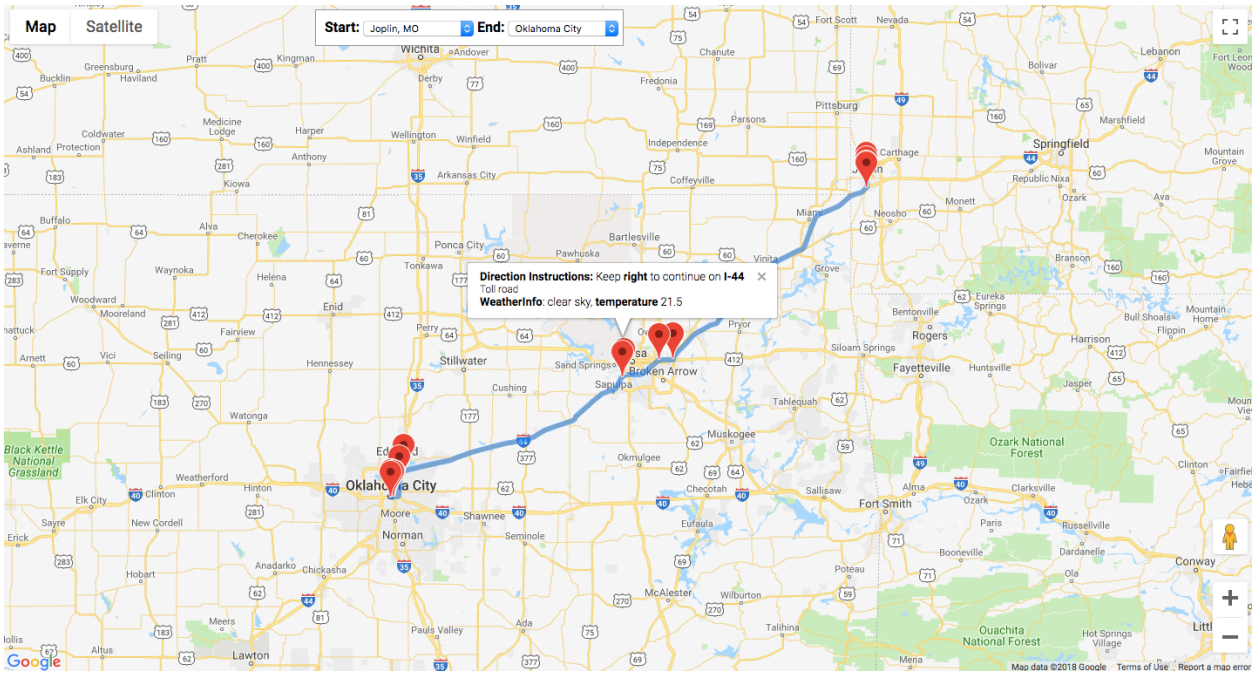


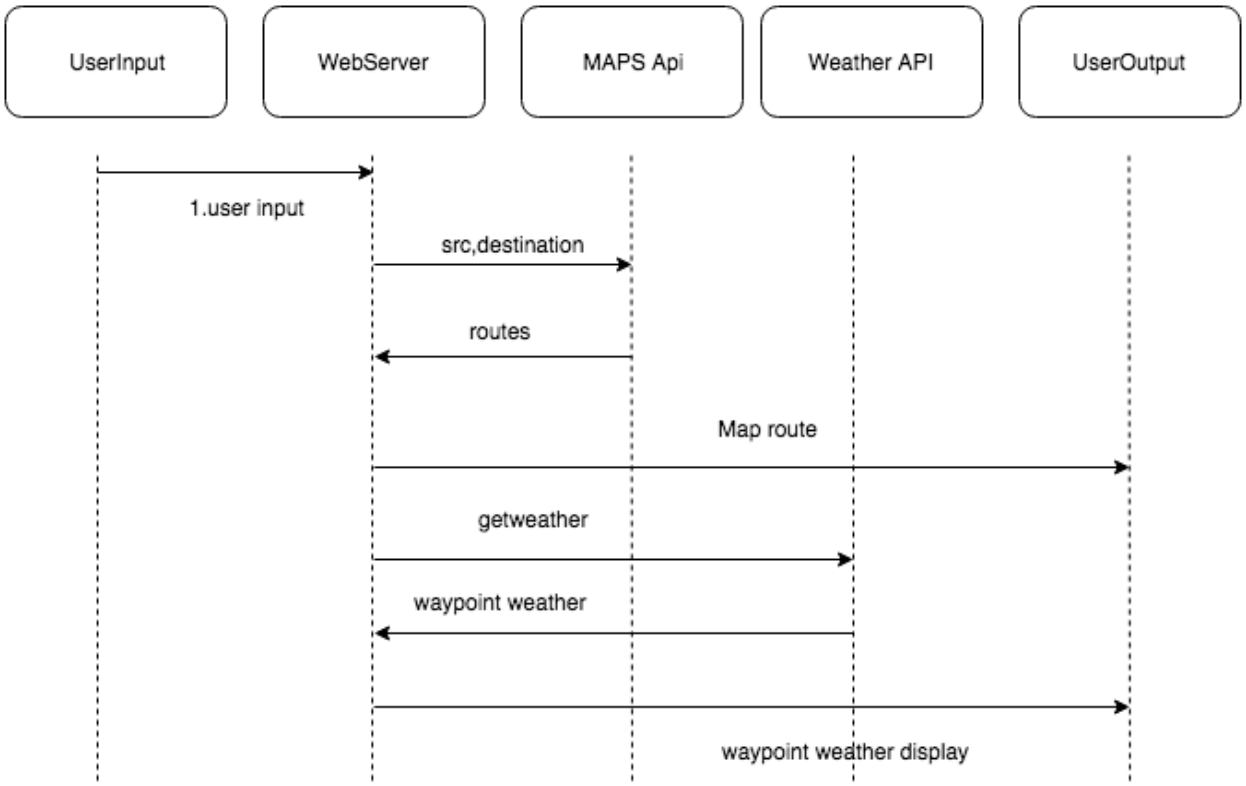
Figure 2 : user interface

As per the project requirements , the project is implemented in 3 phases :

Phase 1:

In Phase 1 , the route details and weather details are directly fetched from the Maps API and Weather API respectively at the Node JS server and send to client on request.

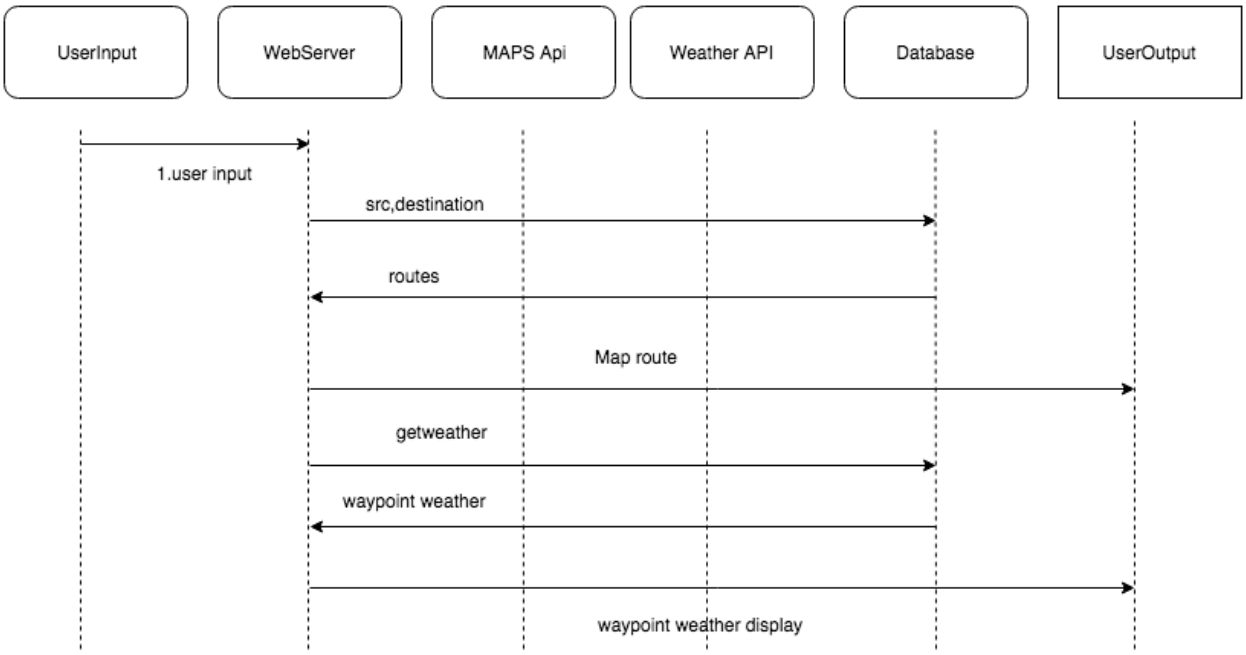
Figure 2 : Flow diagram for phase 1:



Phase 2:

In Phase 2 , a persistence layer in the form of MONGODB is added. It stores all the data related to user requests and results from the API access to the maps API as well as weather API. Hence for any user input first database is checked for any records ,if present sent directly to client else fetched from Maps/Weather Api , save results in database and are sent to client.

Figure 4: Flow diagram for phase 2:



Phase 3 :

In phase 3 , the results obtained from implementations in phase 1 and phase 2 are shown as below :

Figure 5: shows the total time required to fetch route details from Maps API for Vancouver to Calgary as 770 ms --(C1)

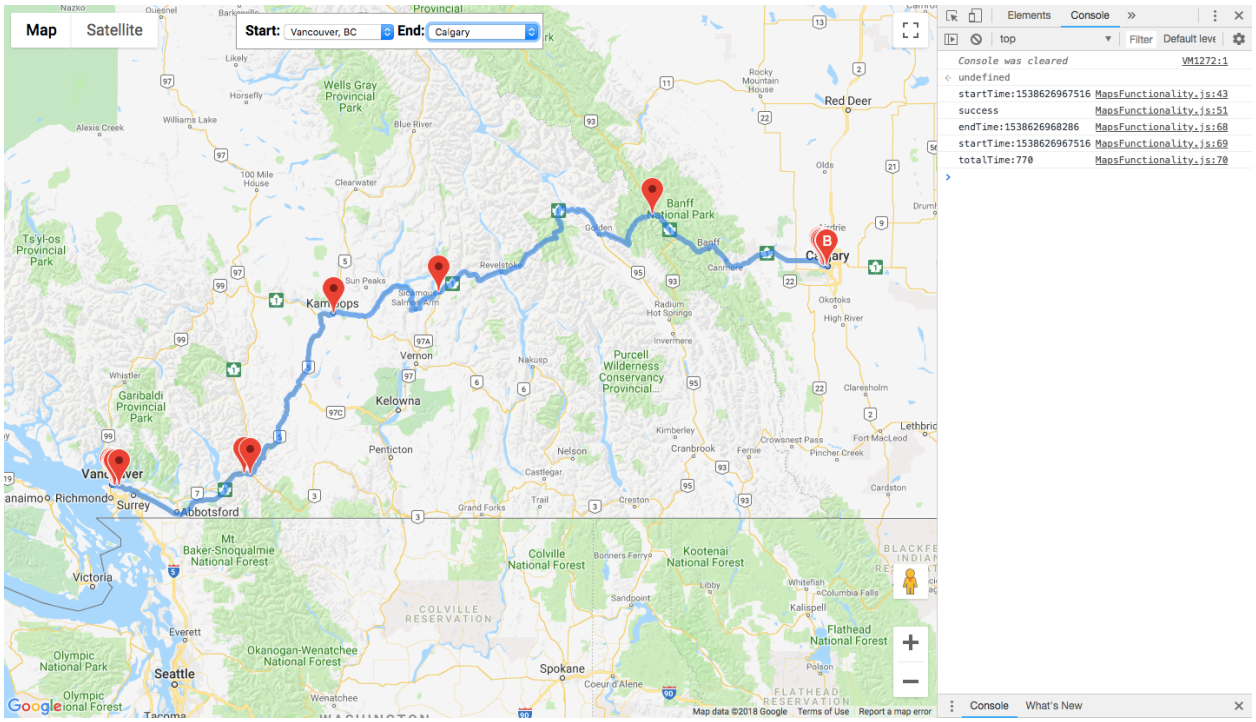


Figure 6: shows the total time required to fetch route details from database Vancouver to Calgary as 63 ms -- (C2)

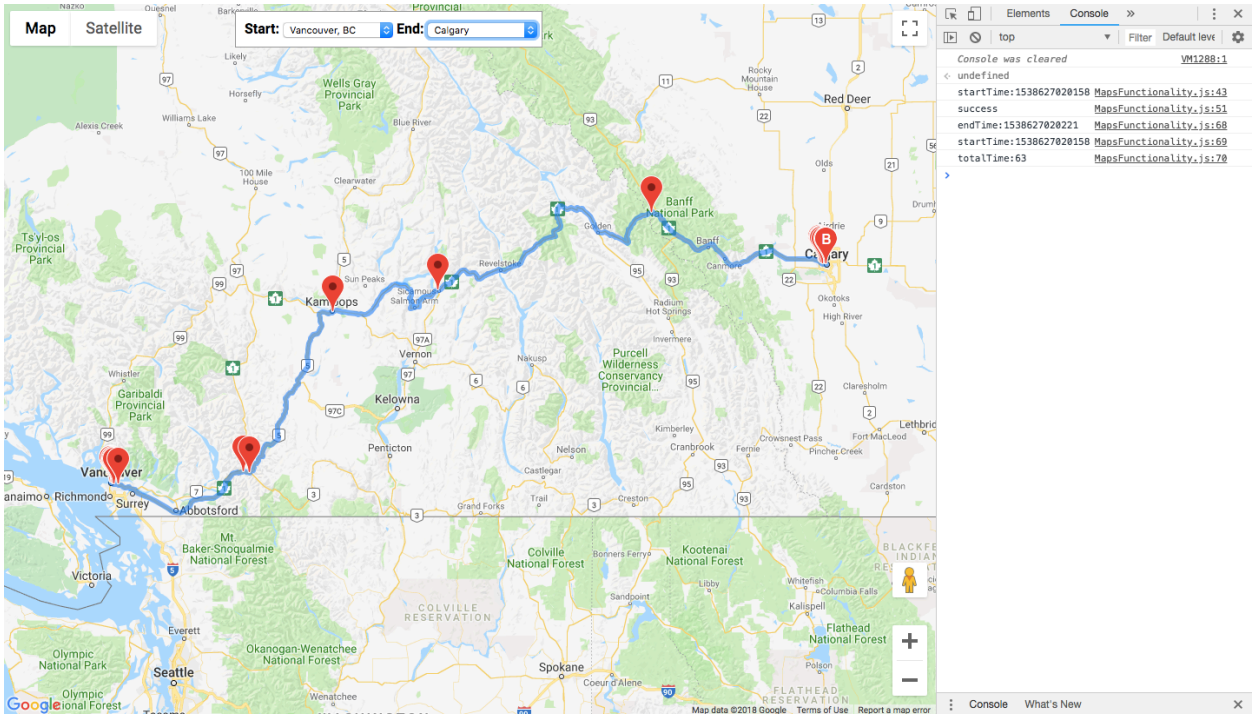


Figure 7: shows the total time required to fetch weather details of a way point from Weather Api for Oklahoma city to Vancouver as 172 ms (C3)

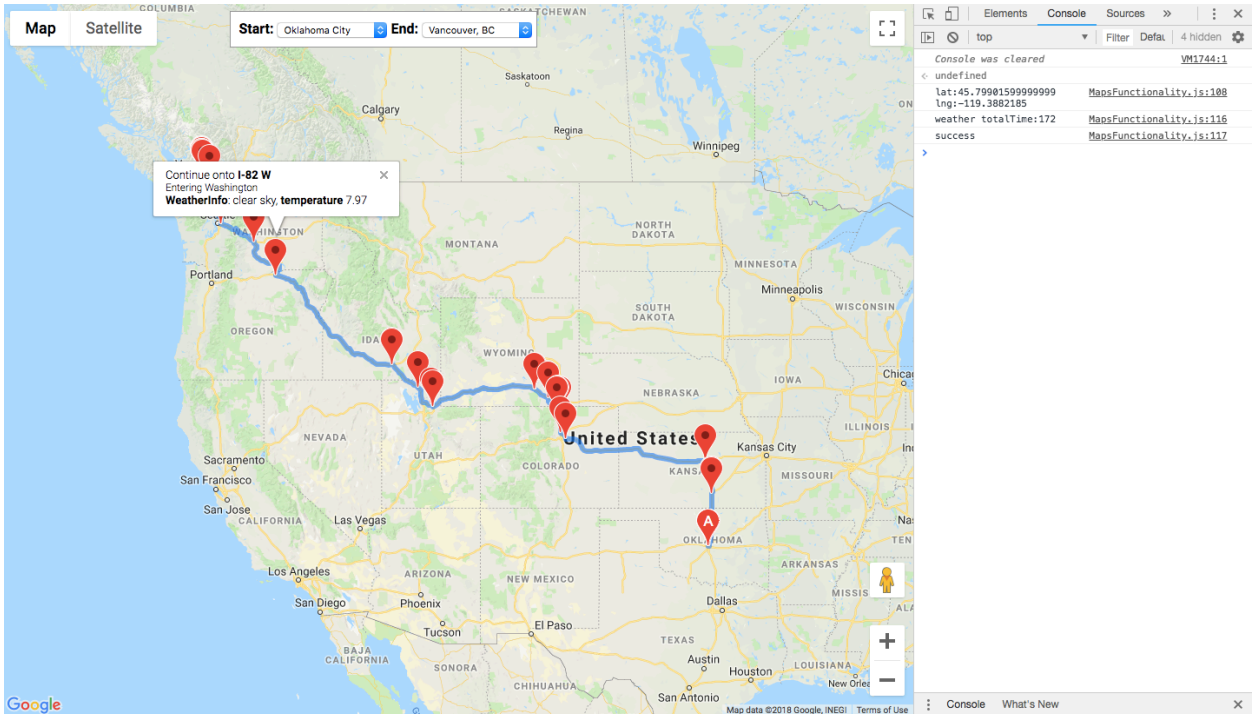
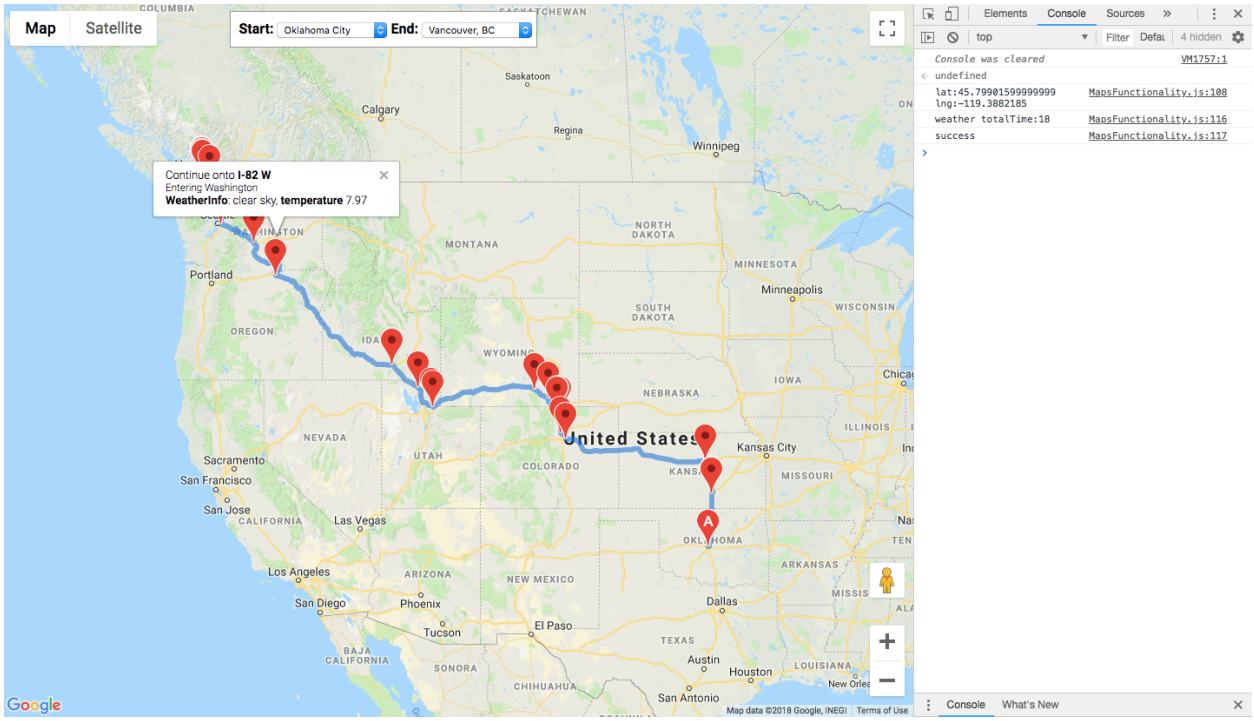


Figure 8: shows the total time required to fetch weather details of the same way point from database for Oklahoma city to Vancouver as 18 ms. --(c4)



COST ANALYSIS :

Figure 5: shows the total time required to fetch route details from Maps API for Vancouver to Calgary as 770 ms --(C1)

Figure 6: shows the total time required to fetch route details from database Vancouver to Calgary as 63 ms -- (C2)

Figure 7: shows the total time required to fetch weather details of a way point from Weather Api for Oklahoma city to Vancouver as 172 ms (C3)

Figure 8: shows the total time required to fetch weather details of the same way point from database for Oklahoma city to Vancouver as 18 ms. --(c4)

From the results , we see that fetching routes data from database instead of Maps Api saves about (C1 – c2) ~ 707ms

Similarly fetching weather data for a way point from database instead of weather api saves about (C3 – C4) ~ 154 ms

CONCLUSION:

From the results obtained we can conclude that fetching repeated requests from database may help in quickly answering user requests thus improving efficiency and scalability. Also this may address connectivity issues related to the remote APIs.

REFERENCES:

1. Maps API: https://developers.google.com/maps/web-services/client-library

2. Weather API: https://openweathermap.org/api