NAVIWAY

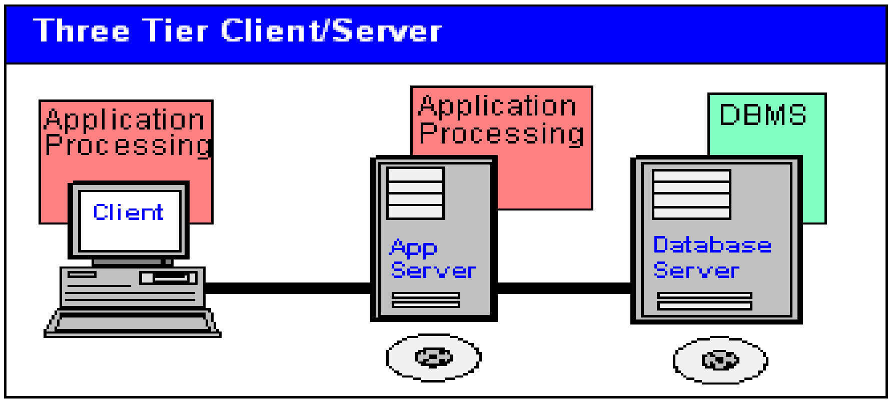
Sravanthi Adibhatla

PERSON ID: 50288587 UBIT NAME: sadibhat

INTRODUCTION

Client–server model

The **client–server model** is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients. Often clients and servers communicate over a computer network on separate hardware, but both client and server may reside in the same system. A server host runs one or more server programs which share their resources with clients. A client does not share any of its resources, but requests a server's content or service function. Clients therefore initiate communication sessions with servers which await incoming requests. Examples of computer applications that use the client–server model are Email, network printing, and the World Wide Web.



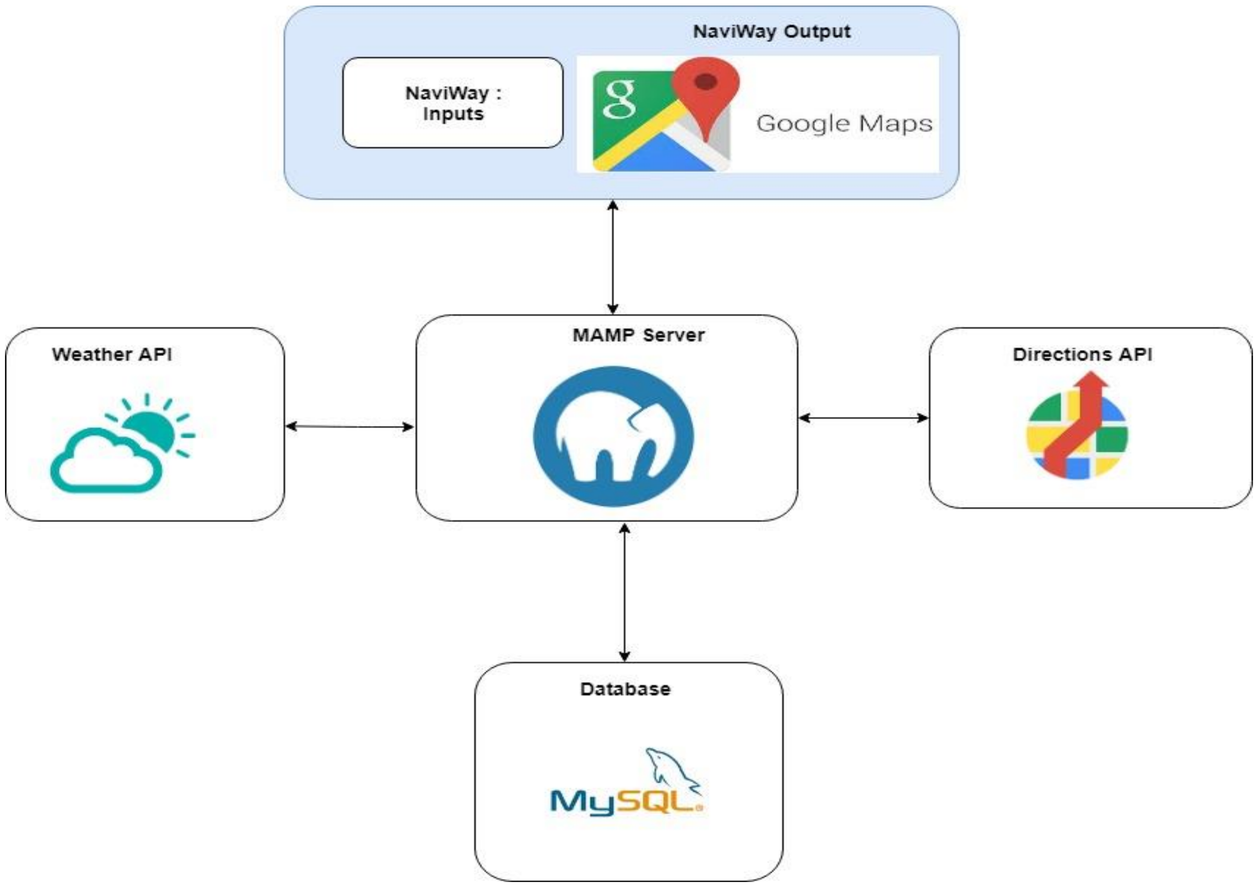
1(a) Client-Server Architecture

ABSTRACT

NaviWay is a navigation resolution for all who long drive lovers. NaviWay provides user with the best route. It shows not just the route but also the important cities that are on the way. It gives the temperature update of the way points on the way to the destination. Technologies used in developing this web application are:

- HTML/JS/CSS or angular for single page application UI
- MAMP server
- MySQL
- PhP to call the third party API

SYSTEM DESIGN OF NAVIWAY



1(b) System model of NaviWay

PHASE 1

In Phase 1, we basically take the start and end location from the user and send it to the server through AJAX call. Basically, client makes request to the server. NaviWay server make API calls to directions and weather services to get the latitude, longitude, temperature for the location as well as the way points between the start and end location. After getting all the details, those values are sent back to the UI and is rendered using Google Maps API.

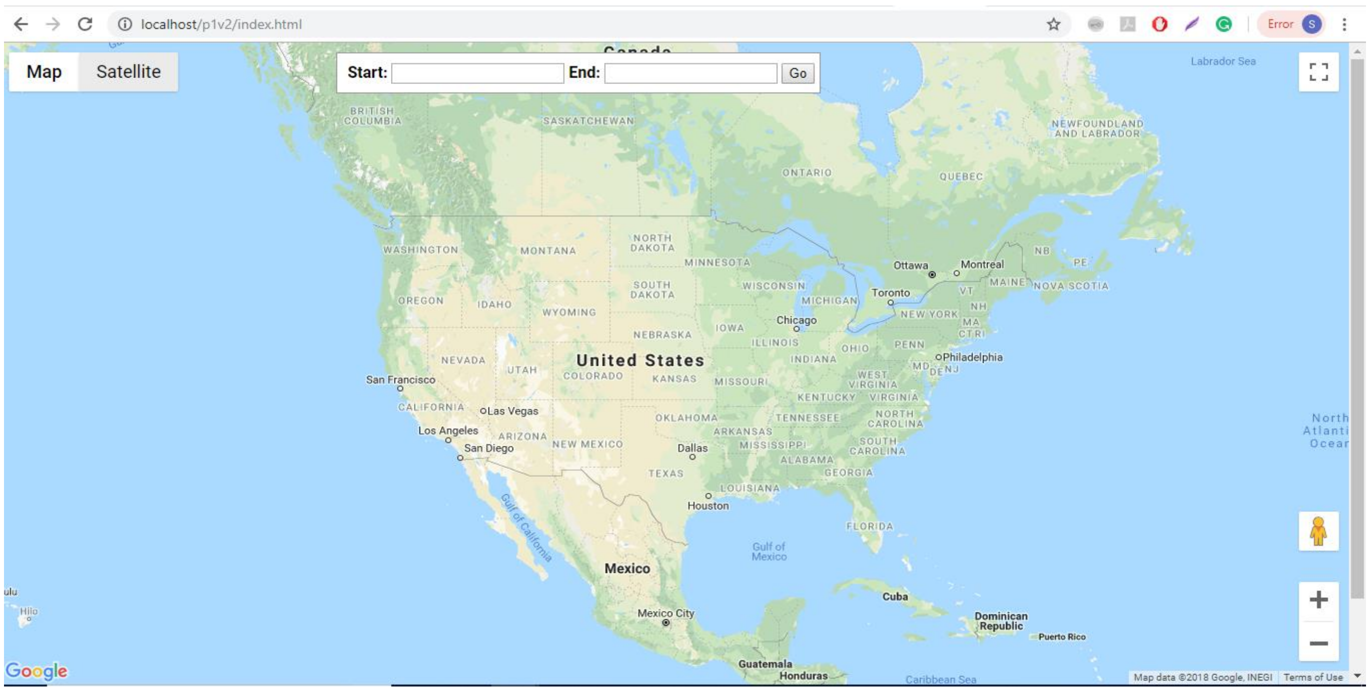
PHASE 2

In Phase 2, we add the persistence layer to the application. After the client makes request to server through AJAX call, and then hit the Directions API. Here, we add in the feature of having persistence layer. We first check if there is any data present with the respective start and end location in database. If yes, instead of hitting the weather API we take the data from the dataset and send it to the client.

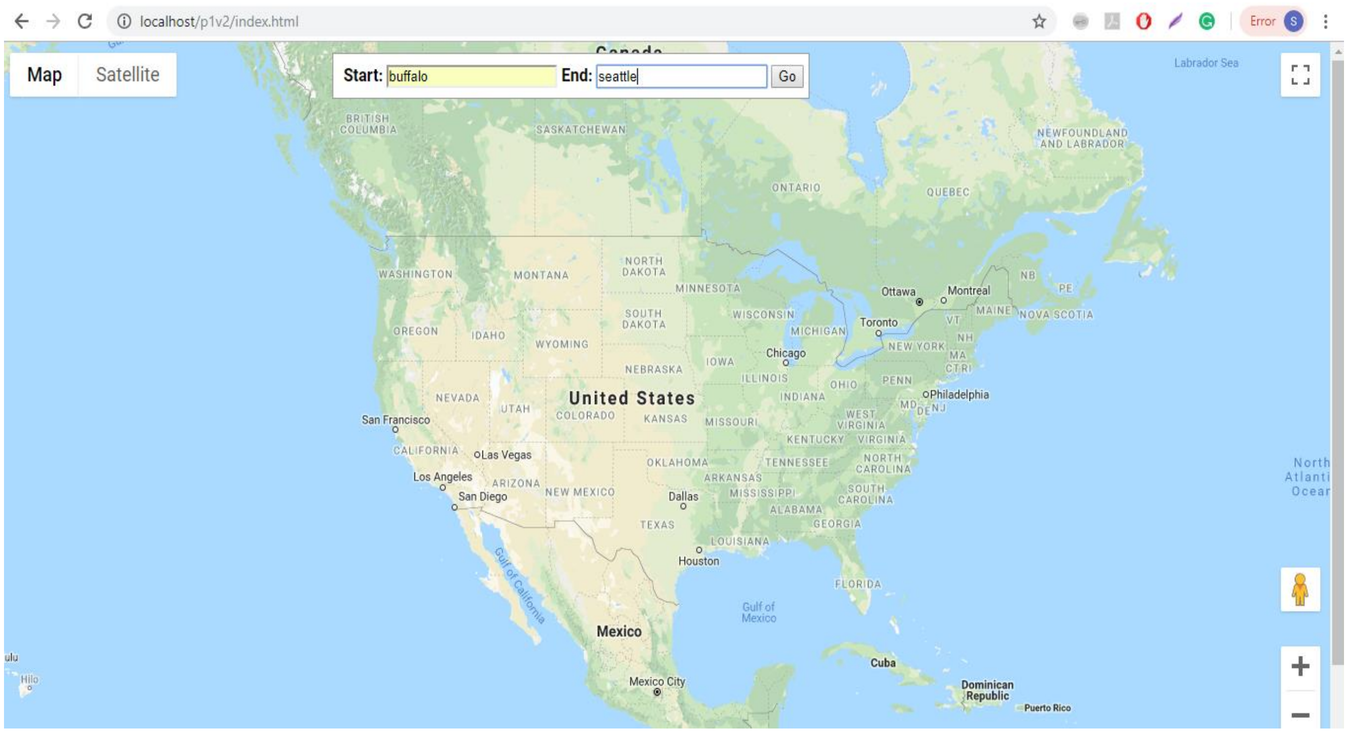
The advantage of having persistence layer is we reduce the number of API call to the weather API. We even reduce the cost function, because the speed of retrieving data from database is much faster than from the external API. This is the main purpose of using Database Layer.

RESULTS

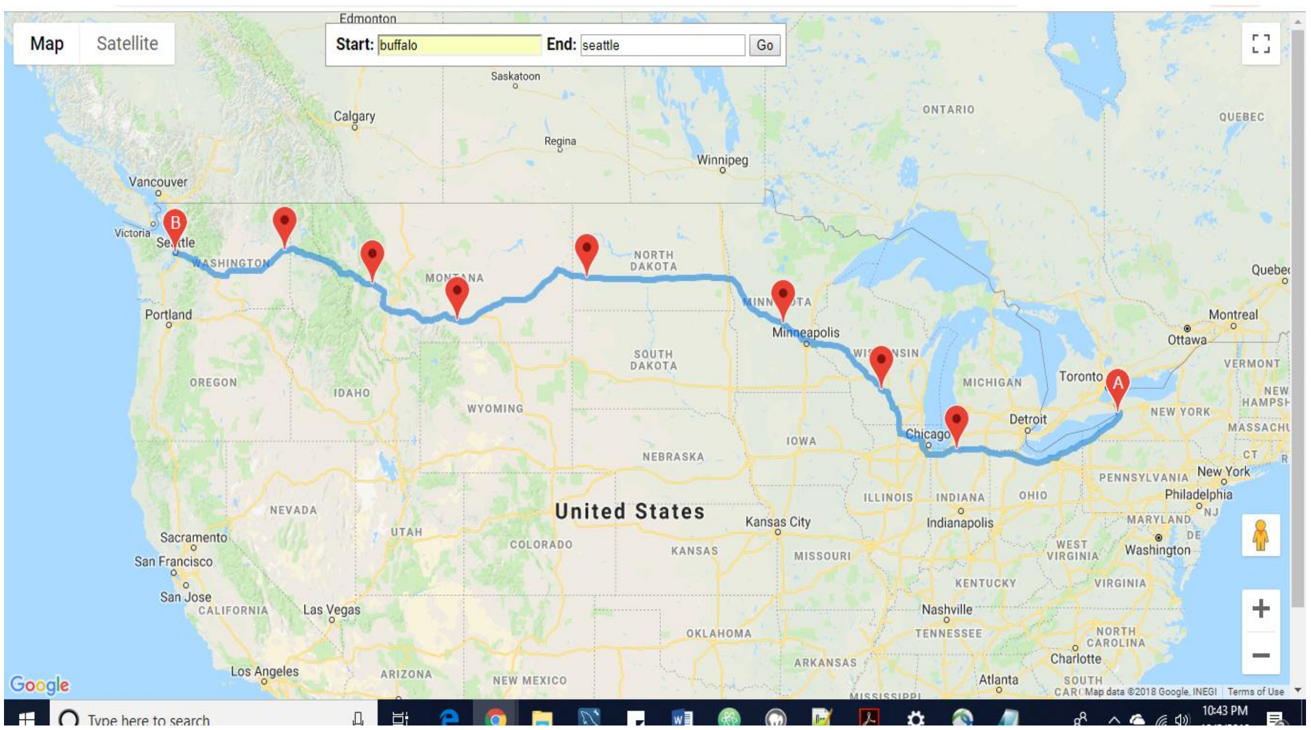
Home screen:



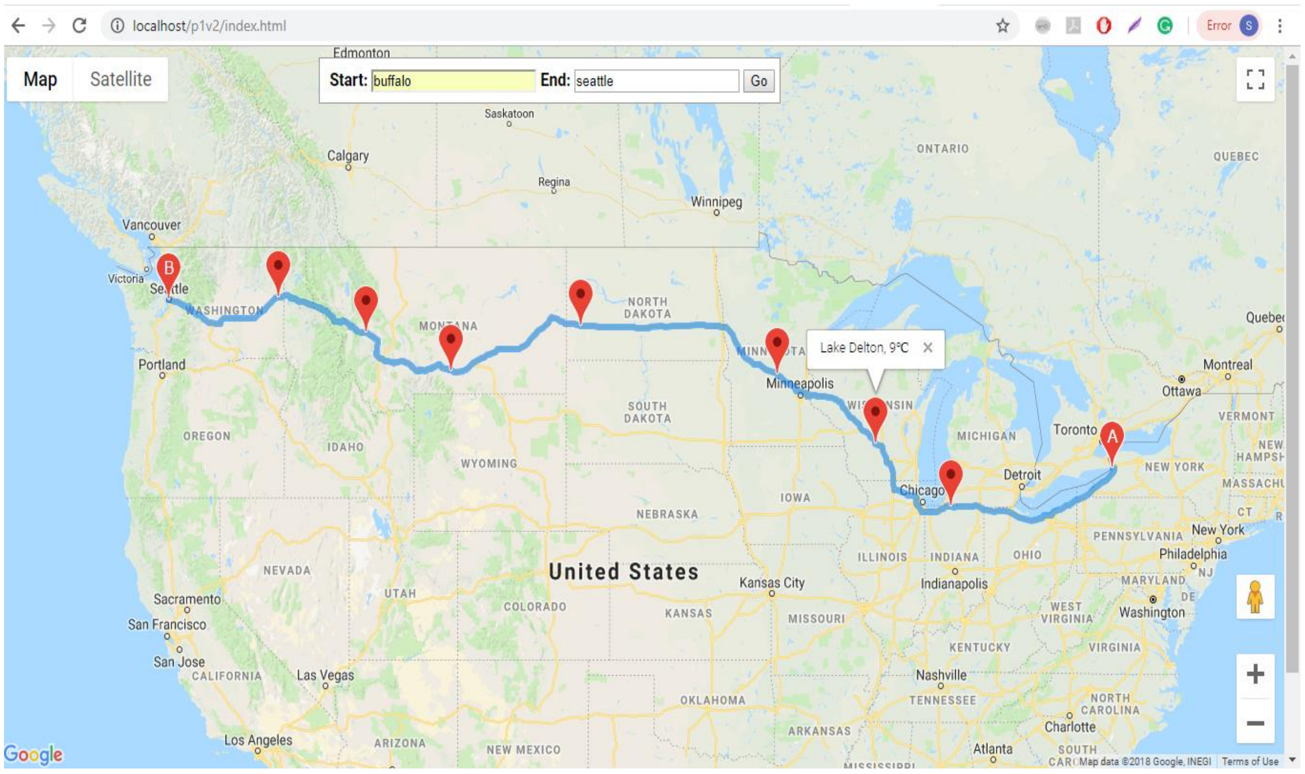
Input Screen:



Output with waypoints:



Waypoints with temperature:



CONCLUSION

So, the take away from this project is the difference of time in extracting data using a third party API and extracting data using own database. We make this conclusion using cost function where c1 is time taken when the request hits our server, c2 is the time taken when we hit the weather API and get data, and c3 is the time when we hit the database and get data. It is observed that c3 is less than c1 and c2, which ensures that the API calls are expensive compared to DB calls. The difference between the times is very little but on larger scale it matters a lot.