Exercise

Google Map API

Winsome Yuen

13th February 2018

Introduction

This short report describes the steps taken to create a application that uses the google map API. The following requirements have been identified for the exercise, including any optional features added in.

Requirements		**Priority**
1	Google Map Geocode API should be used	Core
2	Geocode response should be received in JSON	Core
3	Response should be transformed to display only the ID, Latitude and Longitude	Core
4	A form is used to enter location to be searched	Optional
5	Error message page displayed if no search result is found	Optional
6	Display results on a map	Optional

Design

Core Functionalities

Google Maps Geocode API: This is the specific API we need to use. Best thing to do is get an API key so there is higher limit on the number of searches that can be performed a day. Without the key it may be problematic during testing stages.

GET Request: The API is accessed through an HTTP interface. When researching the type of requests you can send with the google map API, I found that only GET requests can be sent

Python: As there is no set programming language to use, I picked python as they have frameworks which can easily implement an API and I have previously knowledge in using python with API's.

Also, there is extensive resources and google documentation on how to integrate the map API with python.

Optional Functionalities

Flask: decided to use Flask as it has a very fast performance which is suitable for this small exercise, and we don't require an extravagant design as it only has a single task to perform.

POST Request: For the API we use GET request, but for the web form we will need to use a POST request in order to retrieve the information about the location

Front-end Development: Typically using HTML & CSS, with some Javascript possibly just to make the web form look more user friendly

---

Basic Flow:

1. Location user wants to find the geographic coordinates for is stored in a variable
2. Variable is passed as a parameter in the geocode request
3. The output will be in JSON Format and will need to be converted to be readable
4. A dictionary object is returned and will have to pick out only the information required to be stored in variables
5. The contents of the variables with the result will be printed out in the specified format

Alternative flow: status returns with error on search, thus an error message will be printed instead

Tools

Used Google's Geocoding API and opted to use a key:

https://developers.google.com/maps/documentation/geocoding/start

Picked Flask framework to use to create a simple, fast web application:

http://flask.pocoo.org/

For the Map, instead of Google's API Map I opted for an open source map which did not need a key:

http://leafletjs.com/

Fonts used:

https://fonts.google.com/specimen/Comfortaa

Implementation

Explanation

def index() - tells the app to load up the form.html file as thehomepage

def api_message() - the request function is used to retrieve the user input for the location they want to search up. This is then put into a dictionary item stored as ‘address’, which is then passed as a parameter in the GET request to the Google Geocode API, using the requests.get() method.

The response from the API is stored in a variable and converted to a readable format using the json() function.

Then the status of the response is checked to see if it issuccessful. If successful then the JSON type object is passed down to extract the ID, Latitude and Longitude. This can then be printed out in the terminal. Although since I’m implementing the Flask framework, the variables are passed to be used in the result.html file.

geocode.py : This is the main python file that is the solution for the exercise given.

```
import logging

import requests
from flask import Flask, render_template, request

app = Flask(__name__)

# api-endpoint
url =
"https://maps.googleapis.com/maps/api/geocode/json?key=AIzaSyA_wOxjHNfPhmKu2zBo8N5HXsEpewgIQ
F0"


# [START form]
@app.route('/')
def index():
   return render_template('form.html')


# [END form]

# [START result]
@app.route('/result', methods=['POST','GET'])
def api_message():
   locate = request.form.get('location', None)
   payload = {'address': locate}

   response = requests.get(url=url, params=payload)
   data = response.json()

   if data['status'] != 'OK':
       print "Error"

       return render_template(
           'result.html',
           passed="no",
           message="Error"
       )

   else:
       id = data['results'][0]["place_id"]
       latitude = data['results'][0]['geometry']['location']['lat']
       longitude = data['results'][0]['geometry']['location']['lng']

       return render_template(
           'result.html',
           passed="yes",
           id=id,
           latitude=latitude,
           longitude=longitude
       )

# [END result]

if __name__ == '__main__':
   app.run(debug=True)


@app.errorhandler(500)
def server_error(e):
   # Log the error and stacktrace.
   logging.exception('An error occurred during a request.')
   return 'An internal error occurred.', 500
```

Problems encountered:

- When updating my css file it would not display the new changes. Upon research I found that it was due to the browser cache. So simply using CTRL+Shift+r fixed the problem.
- Results that were empty meant that the variables that were trying to store to geographic coordinates would terminate the problem and print error messages in the terminal. In order to fix the problem an IF-Else statement was used to deal with cases like those. Also,

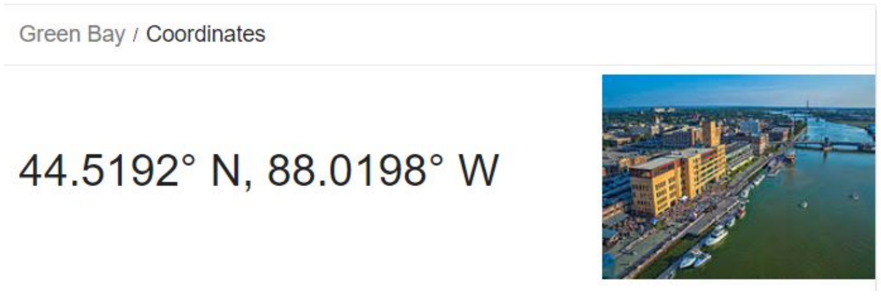
a variable would store the status of the API response, which could then be retrieved on the front-end side to display the correct output on screen. Again, another IF-Else statement was used.

Additional features

For the map that used it required some Javascript. To obtain the geographic coordinates the variables from the python file was passed to the javascript function and converted to JSON.

Conclusion

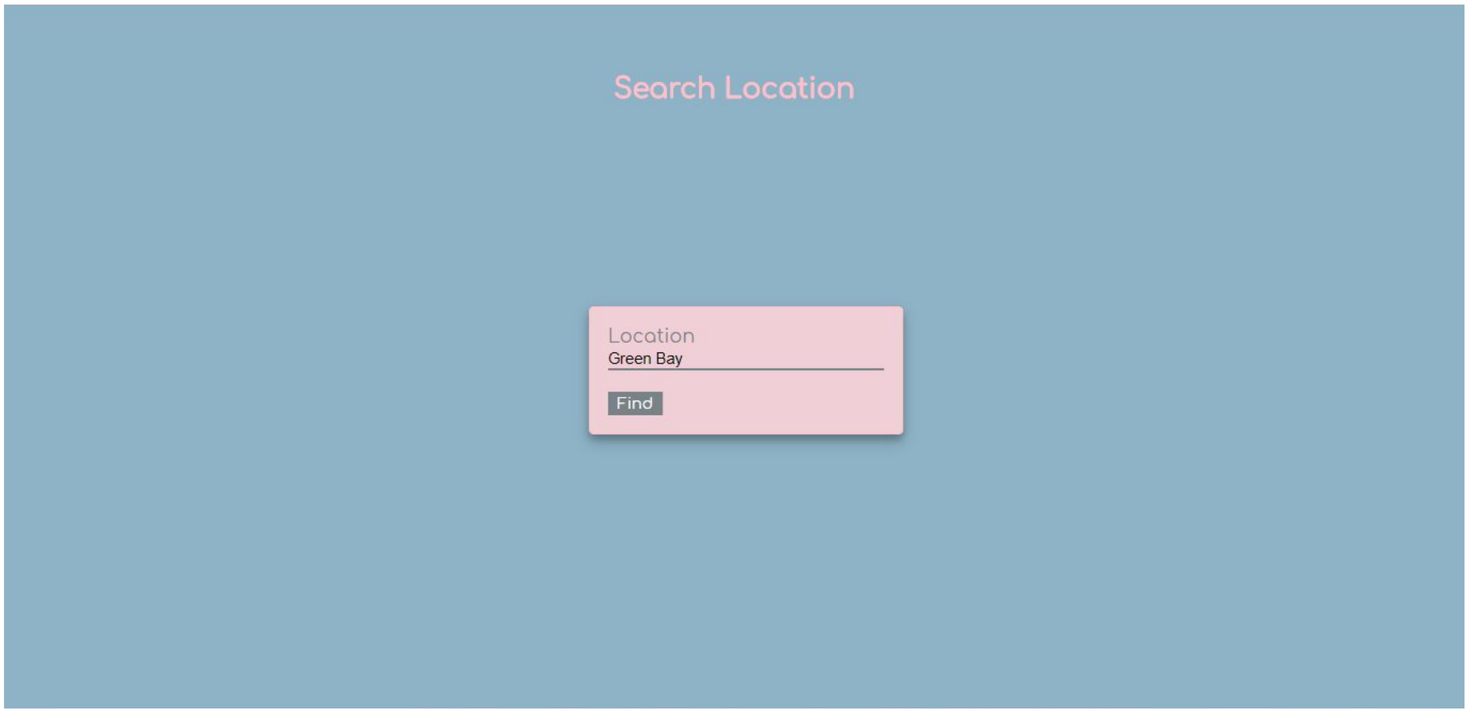
The final result when Green Bay is quered:



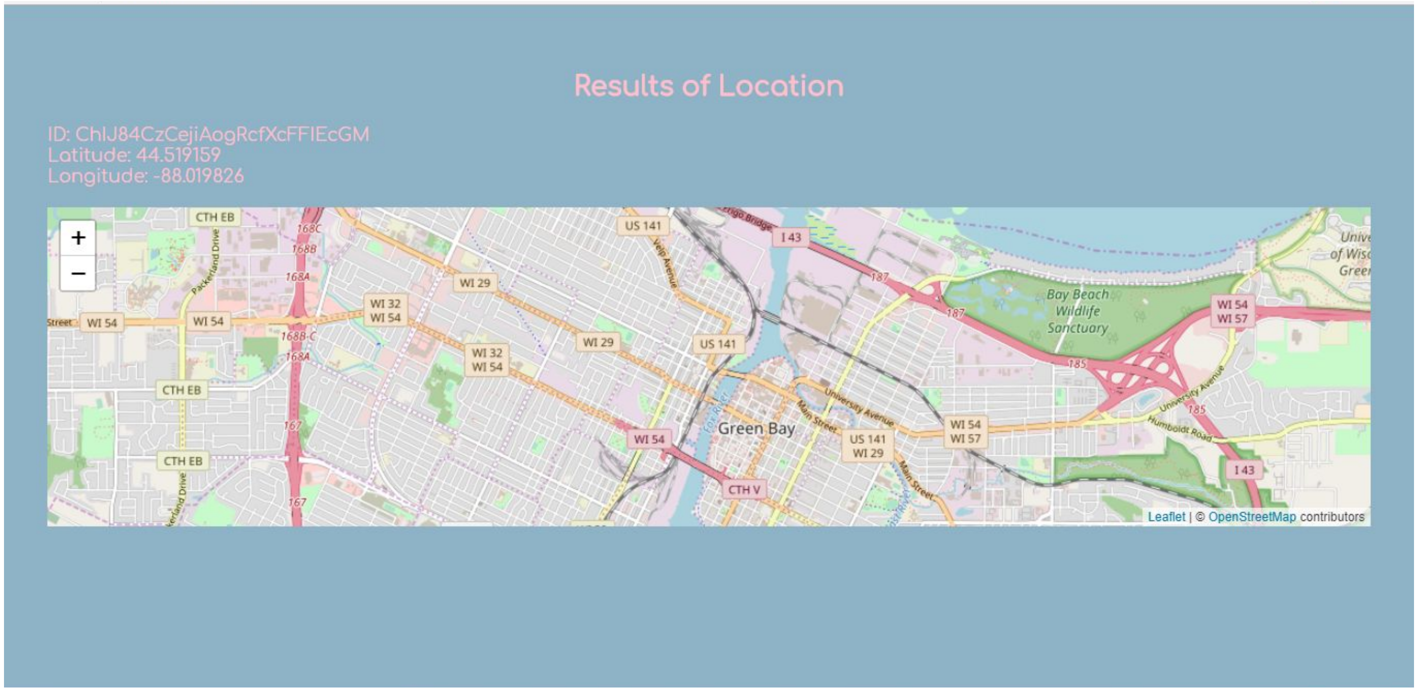
Green Bay / Coordinates

44.5192° N, 88.0198° W

templates/form.html



templates/result.html


Results of Location

ID: ChIJ84CzCejiAogRcfXcFFIEcGM
Latitude: 44.519159
Longitude: -88.019826

+
−
Green Bay
Bay Beach Wildlife Sanctuary
Leaflet | © OpenStreetMap contributors