COS60016: Programming for Development || Assignment 1: Build a web-based API integration framework

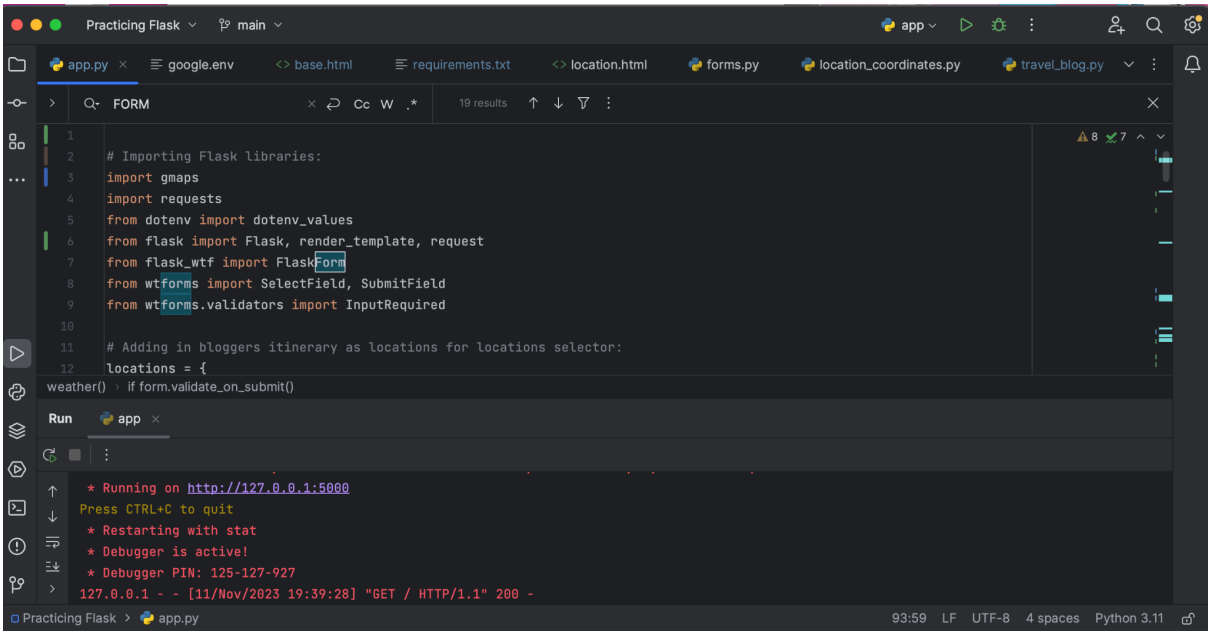
Student Name: Alexandra Bain
Student Number: 103083826

Scenario: Tasked with building an integrated web-based weather solution with requests from open APIs, the app is to return weather data from each specific location of the travel blogger's itinerary. Ten different locations have been provided.

I have decided to incorporate a search bar to return temperature in celsius, a drop-down list of the locations provided returning weather data, and OpenWeatherMap APIs to return weather using these functions. I also plan to utilise Google Maps to provide a location map.

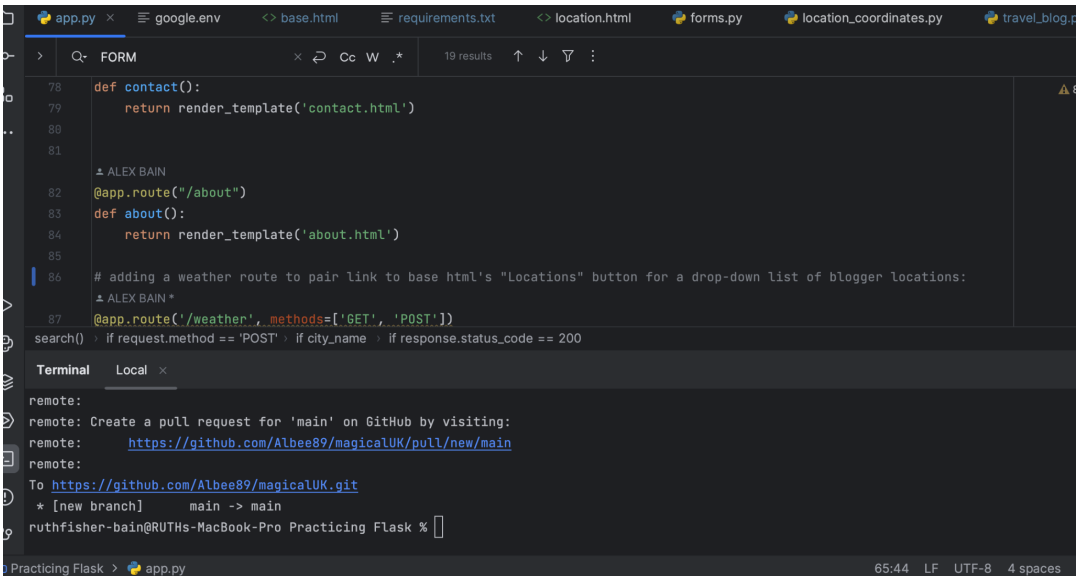
Wanting to build the app as a simple, easy-to-navigate taskbar, I downloaded the Flask microframework as a first step.

The requests library and Flask are imported into a python file, **app.py,** alongside various other module imports needed for the functions as I built:

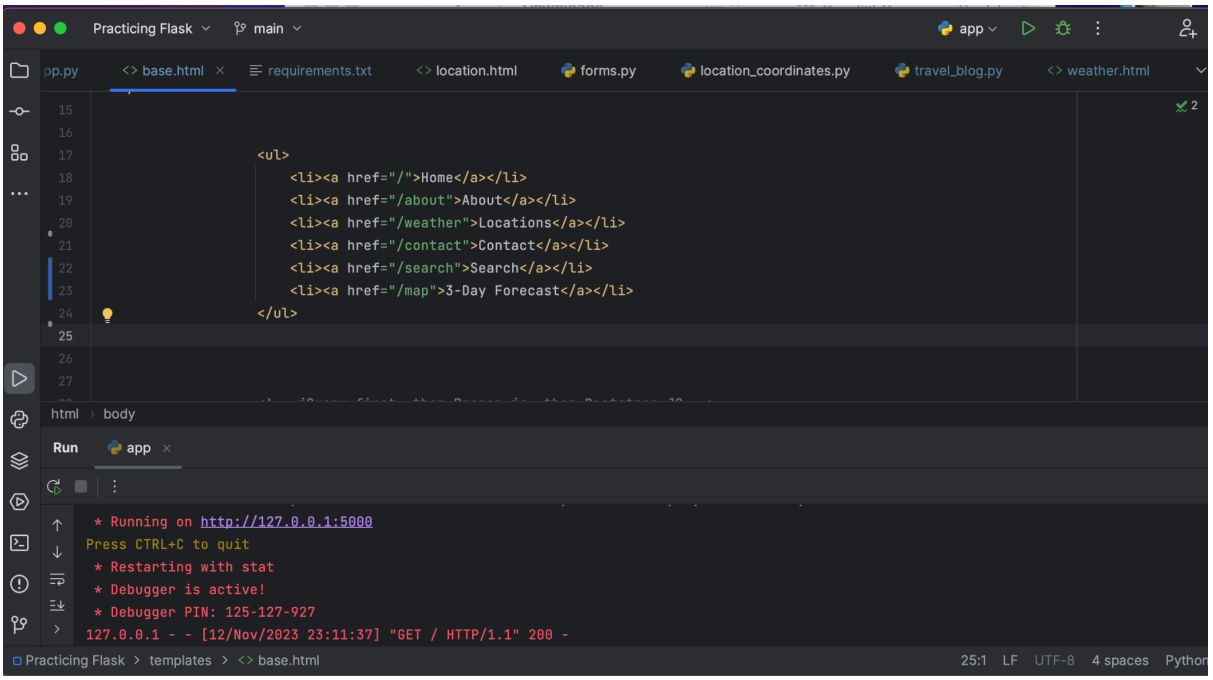


The app.py file serves as the central Python file for my project, as it features the app routes for the app and links - dynamic urls- to each html file; the routes consist of the HTTP method and a resource

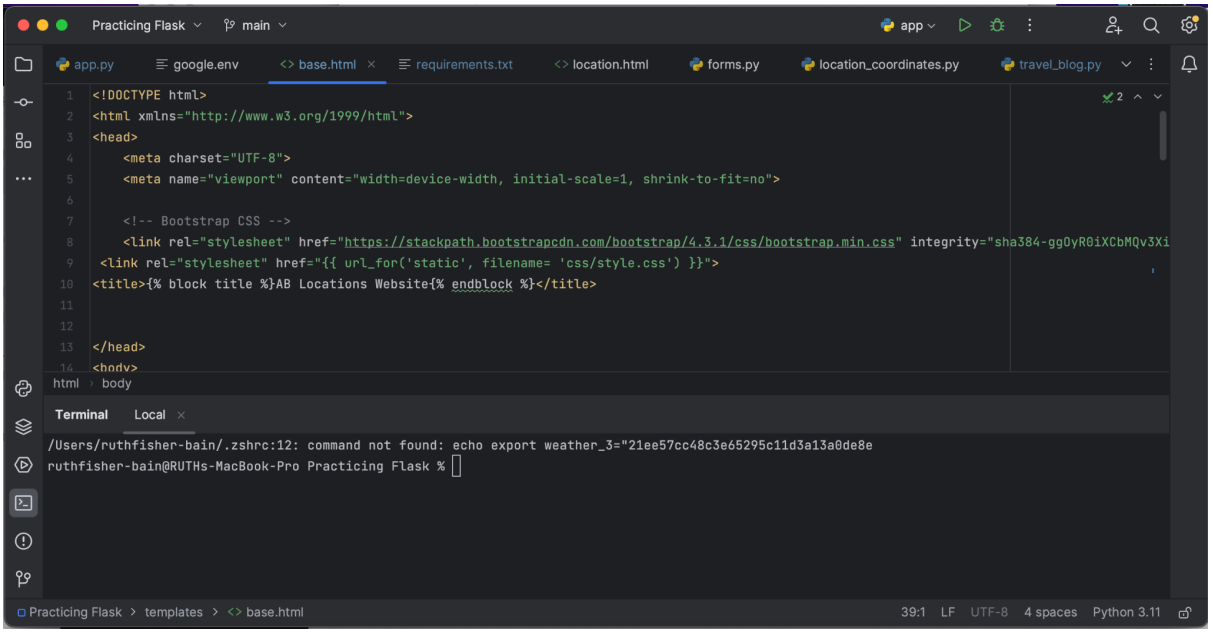
path. The app.routes will direct the incoming API requests to resources:

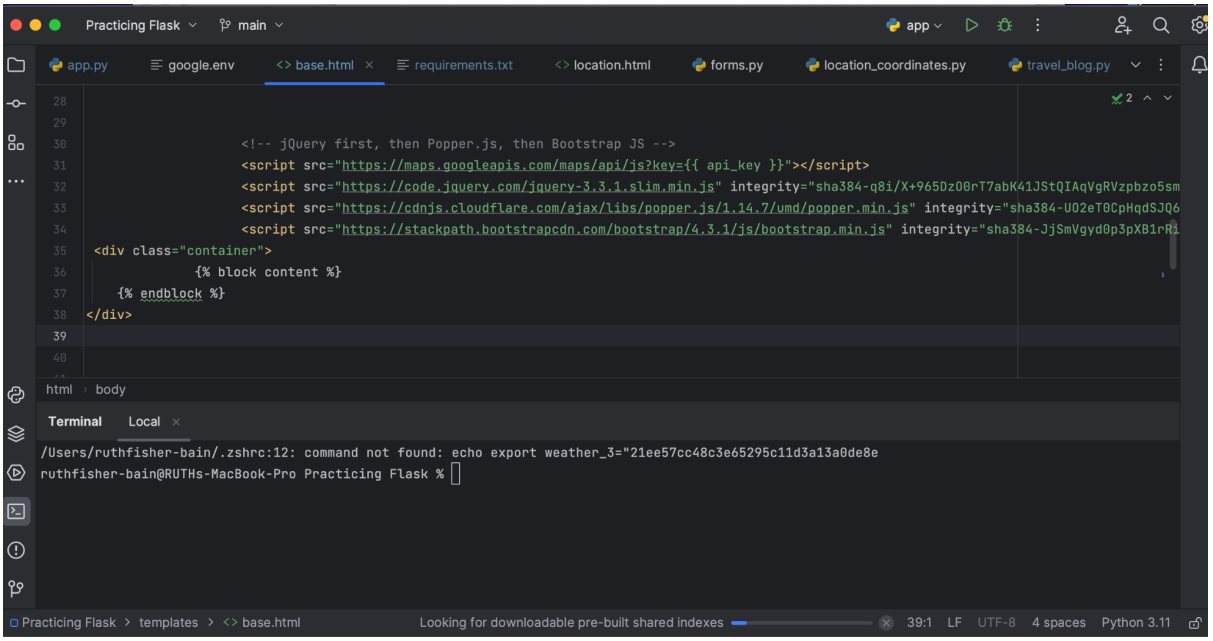


I created and wrote block templates for several html files, saved in a folder I named templates. These files use html blocks to integrate with Python, created under the Flask framework. They are accessible via a main html file, **base.html,** as seen in this navigation/unordered list:

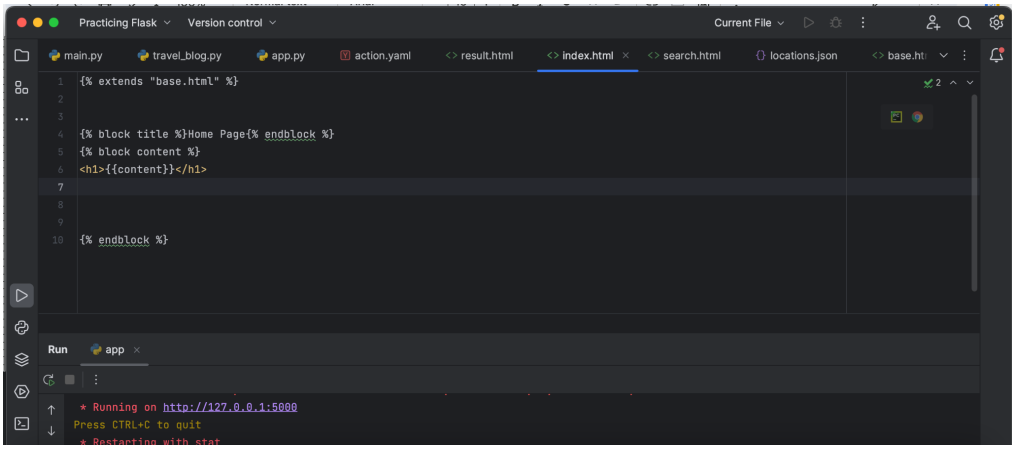


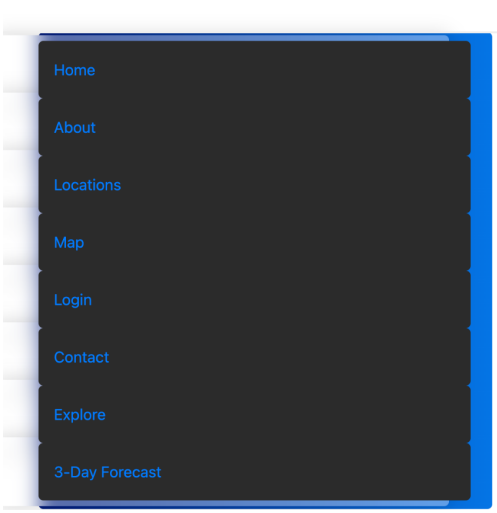
Original html format is used in this file, and all the bootstrap stylesheet and script sources are stored here:



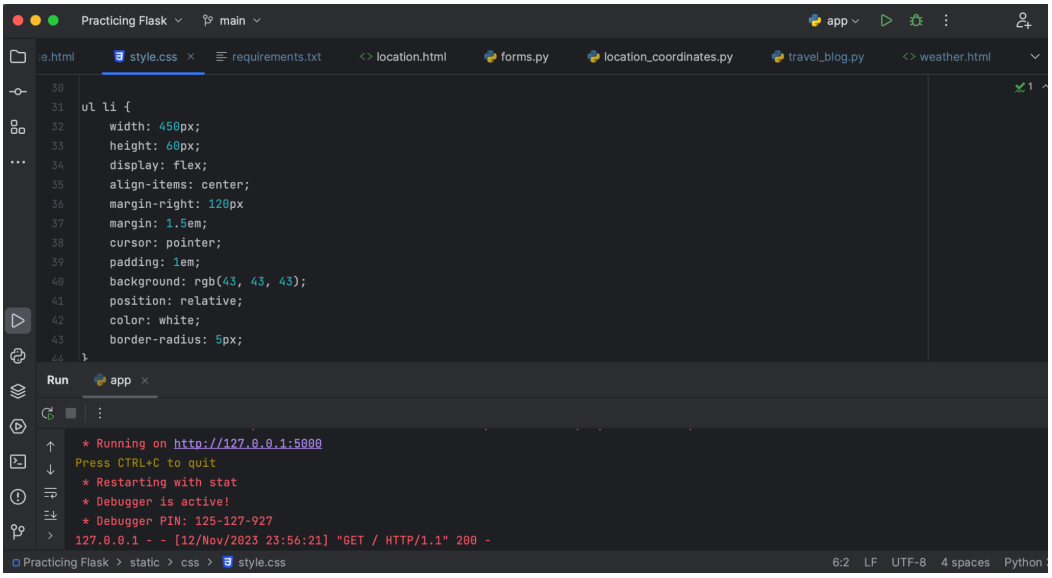


The base.html file with css styling (file style.css lives in the “static” folder), has made this list a navigational tool. Every other html file in the templates folder is an extension of this base.html file:



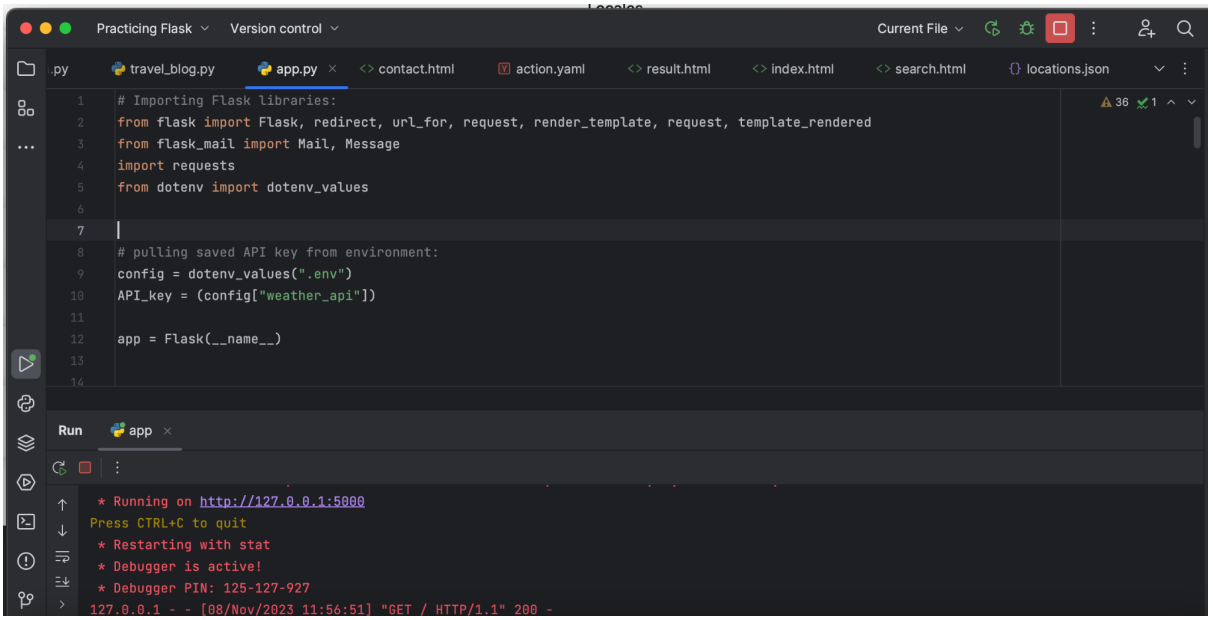


These template files give an html outline to each page on the python web app that will be functioning on the blogger's site.

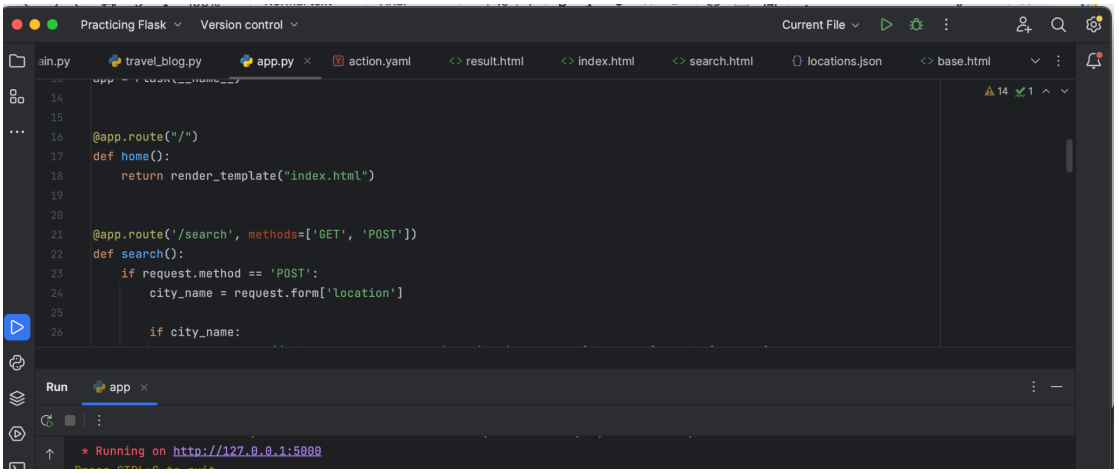


Search

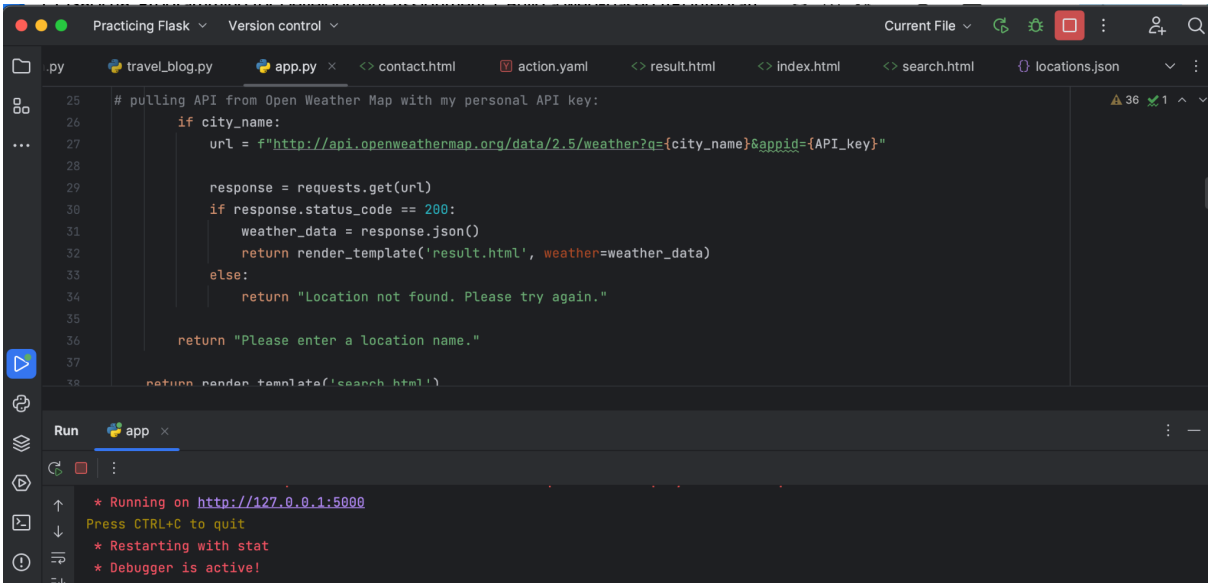
For the search app-route in **app.py,** I generate and store my first OWP API key using dotenv_values, having saved the key in my environment for security:



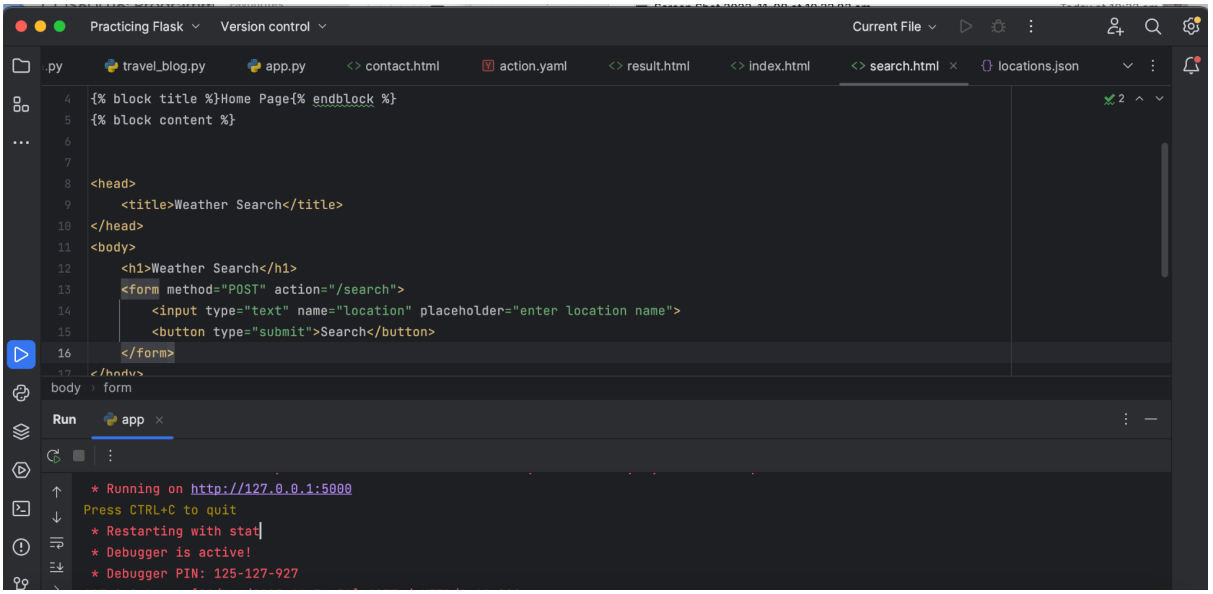
Pulling from the Open Weather Map API, I set the variable of the openweather API url. The 'search' app route is defined with GET and POST HTTP methods using if/else statements:



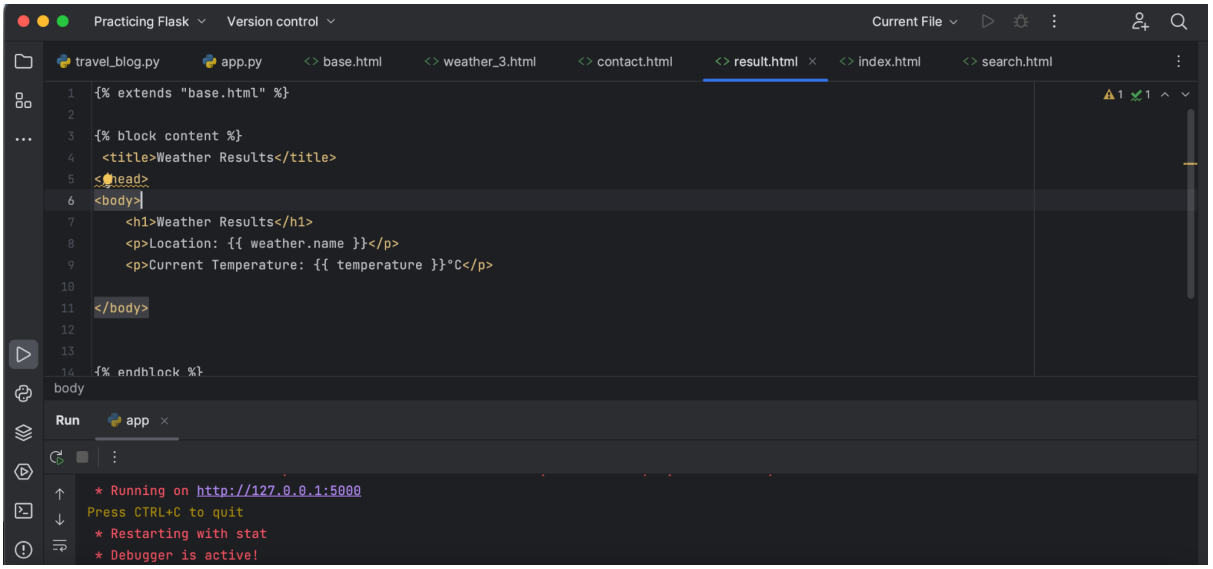
Wanting the user to be able to use the search bar for any location they want, I built a HTTP request method:



The search.html template is rendered regardless of if/else result, to provide the search bar rendered in the Python app via **search.html:**



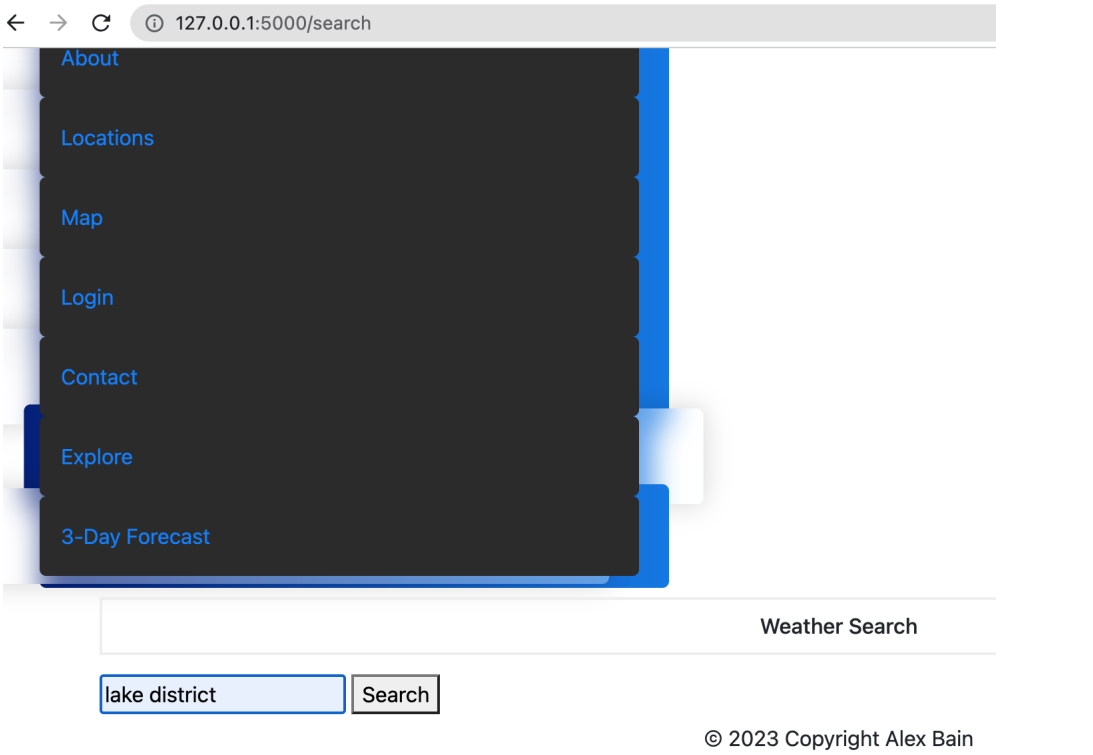
Once the location is added and the “Search” button pressed on the website, the app runs and if the status_code is 200:

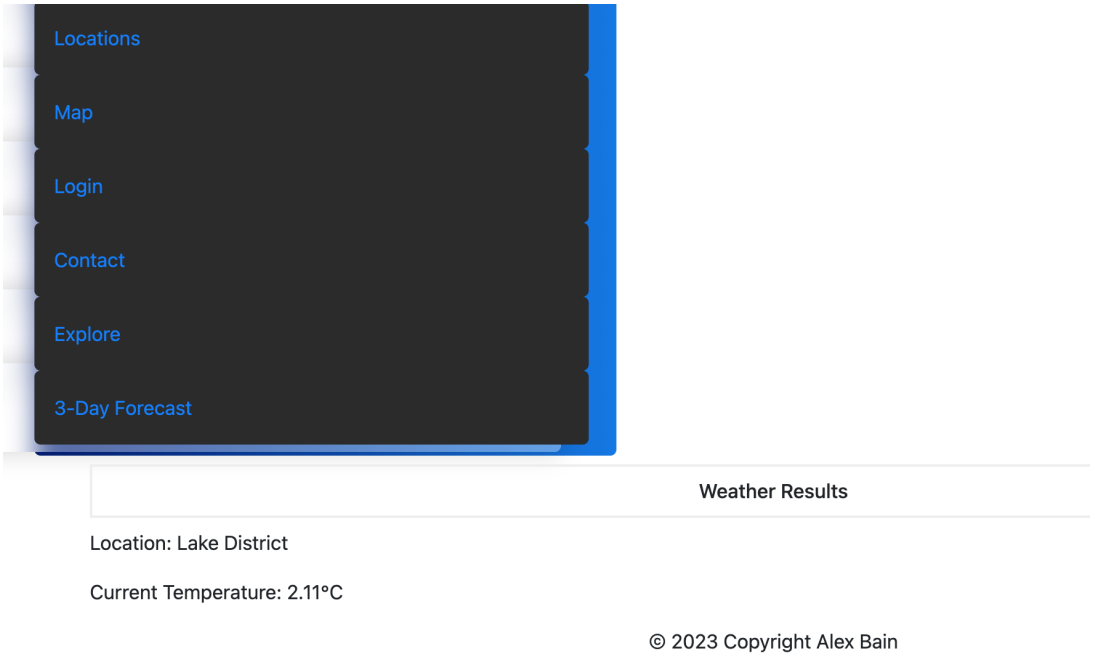


The temperature was additionally converted from kelvin to celsius temperature with round() method, rendered and displayed via the second template used in this section, **result.html**.

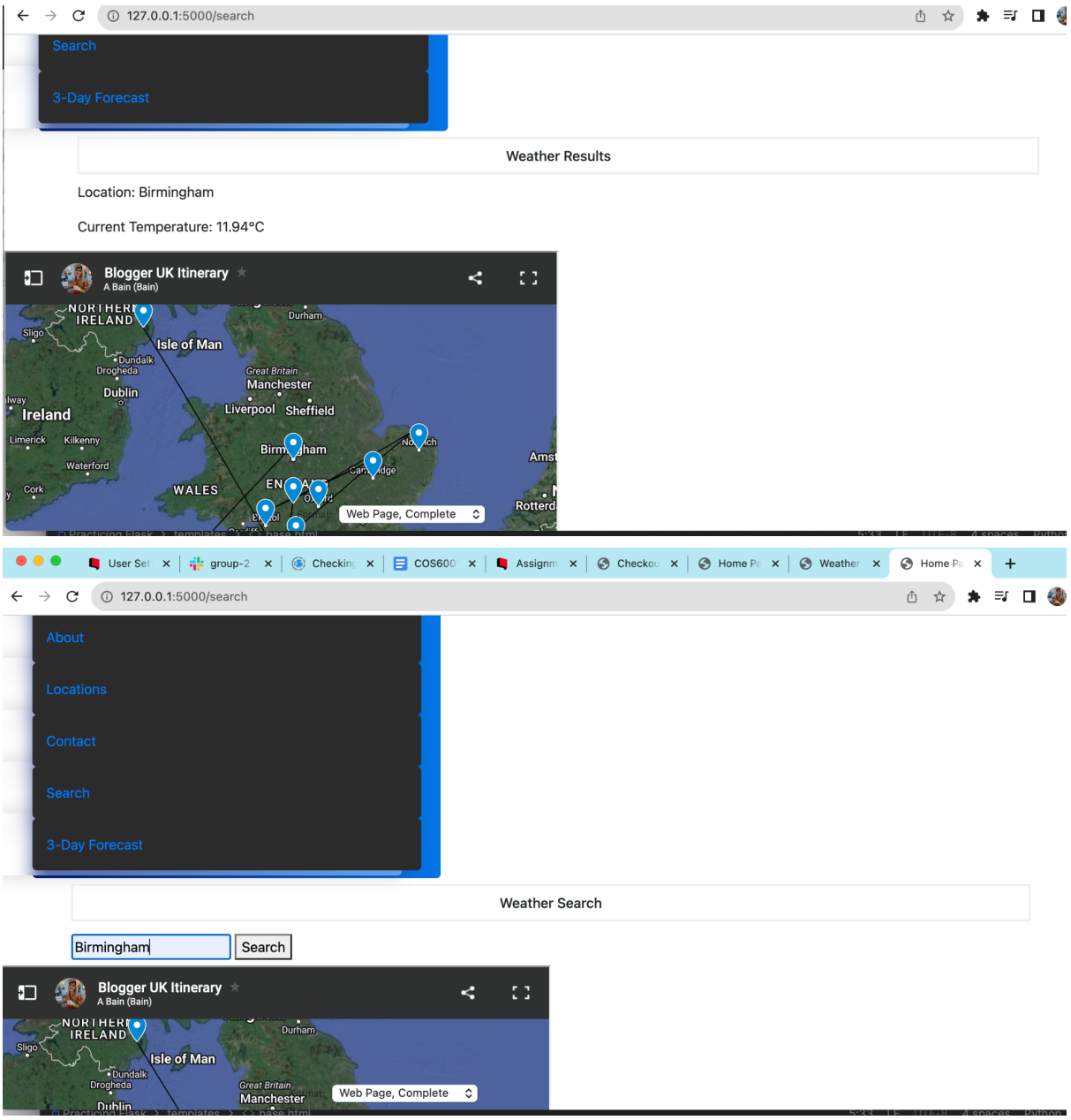
I had to ensure the celsius symbol was properly displayed in this html block: Building this method took some time to piece together, in particular with the API and keys. Remembering Python formatting, I was sure to take notes via comments in the code to explain components, remind myself of the function and note what I had learned.

On the website, this displays as a searchable location, and it's temperature:





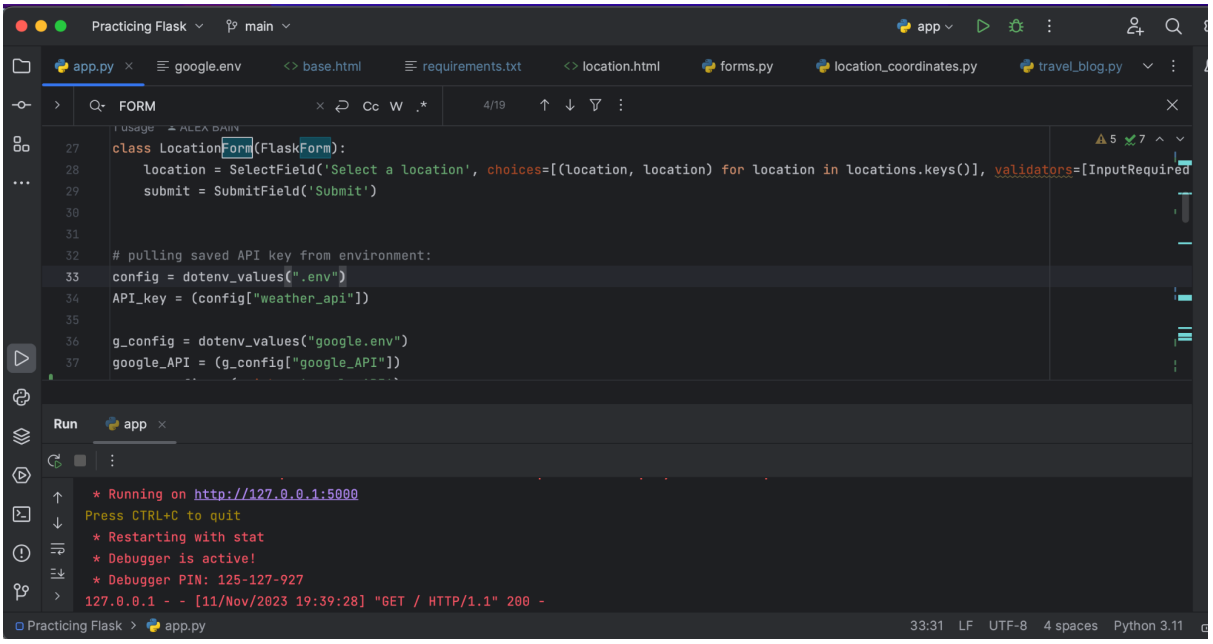
After updating **base.html** with Google Maps iframe, below:



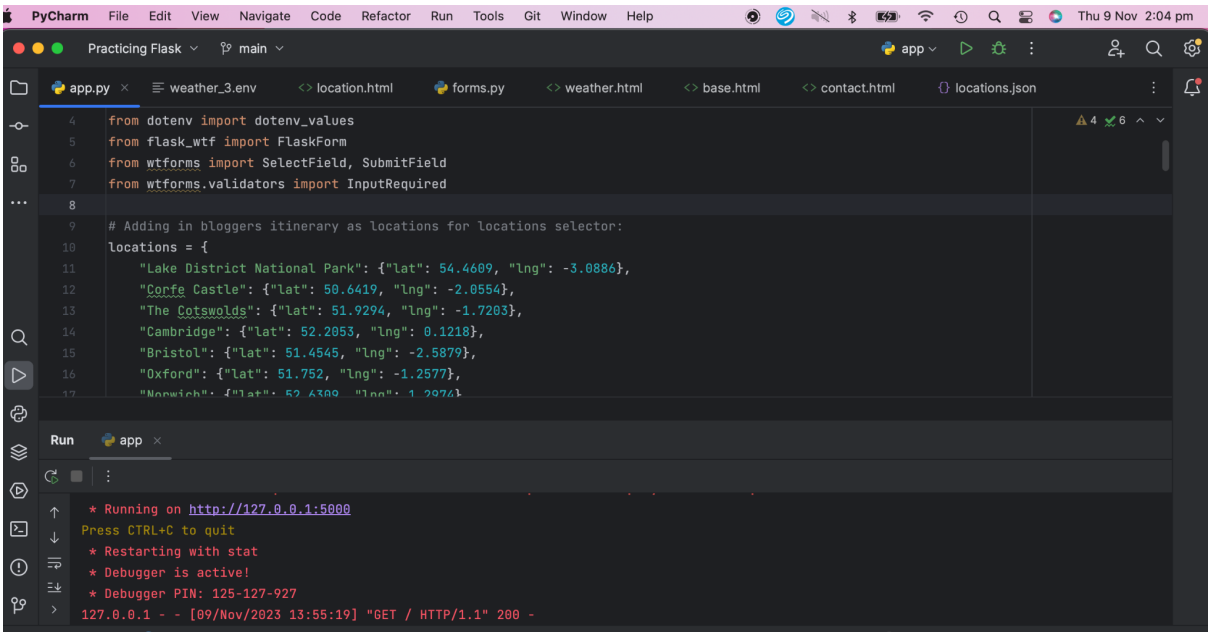
I used this latest development as an opportunity to commit and push my project to a git repository: https://github.com/Albee89/magicalUK
I am also continuously testing as part of the iterative experience of building each section of the app and managing time. The print() function is often used to check API Keys and python code.

Locations

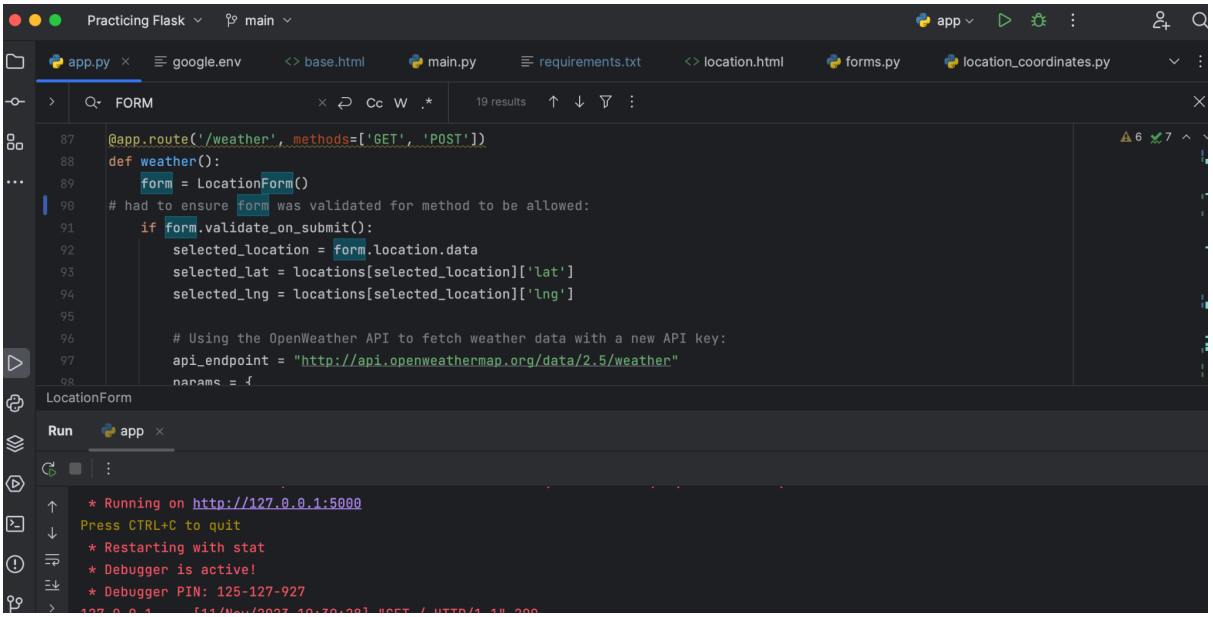
I looked up some examples of drop-down boxes and integrations with weather APIs and looked into a class method with a form, downloading packages such as flask_wtf.I practised some class methods, learning about a ‘FlaskForm’ and using it in the constructor of the class ‘Location Form’:



As I've added forms into my app, I have installed modules via pip and imported them into app.py:

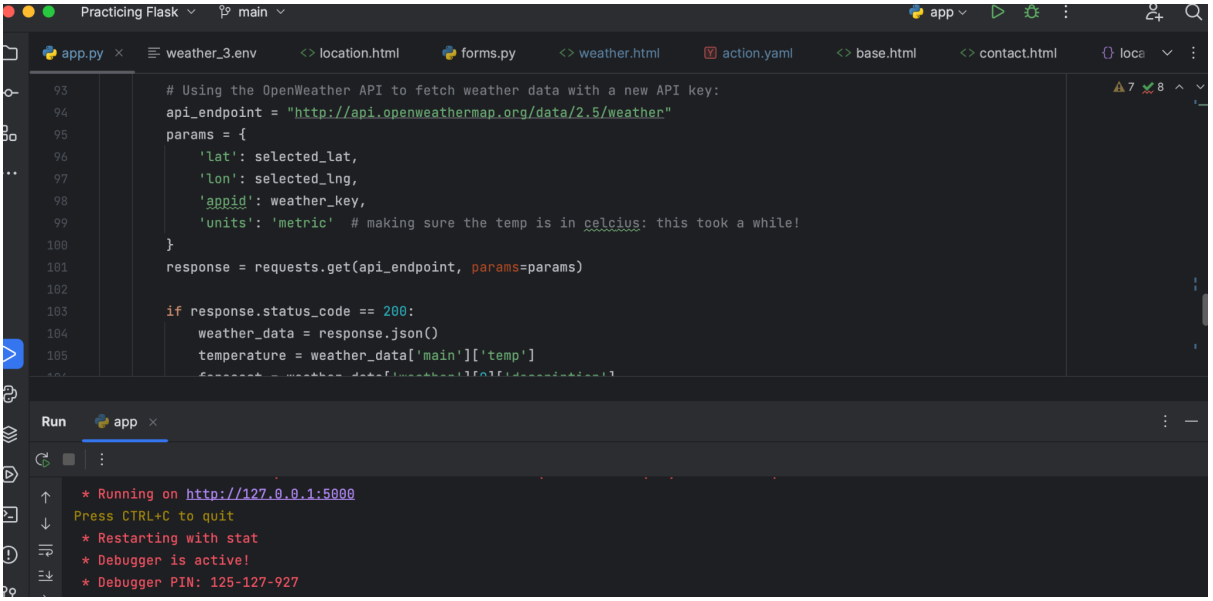


Into app.py, I built the 'locations' function, using weather.html as a template:

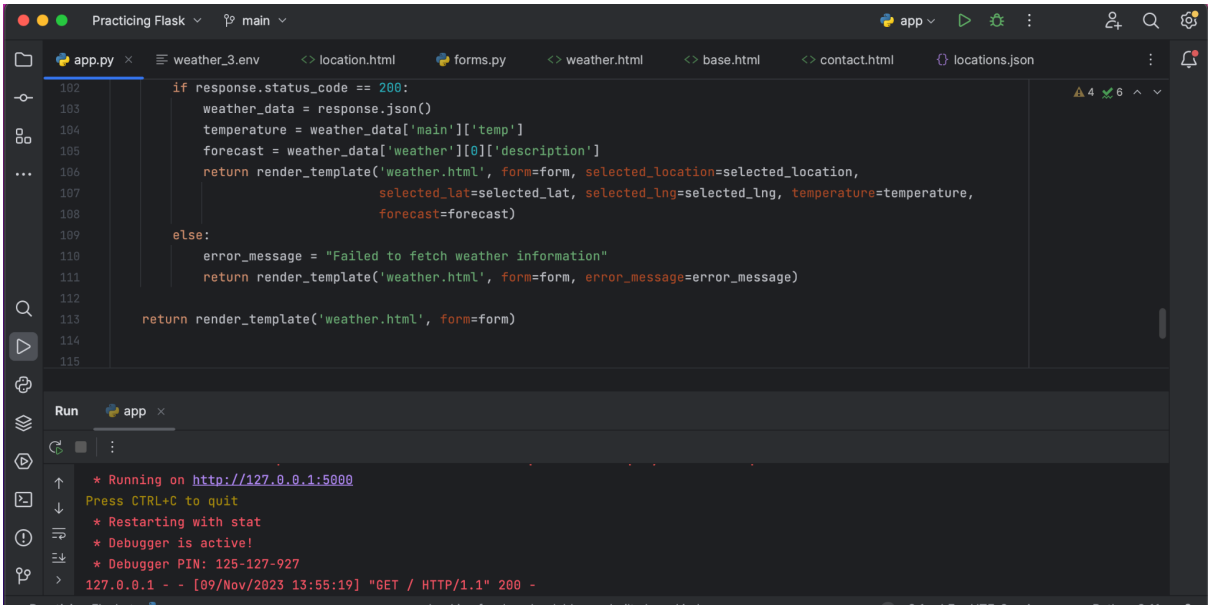


In Python, I created objects from the LocationForm() class. This delivers names and coordinates to fetch Latitude, Longitude, and Temperature Forecast from the API endpoint.

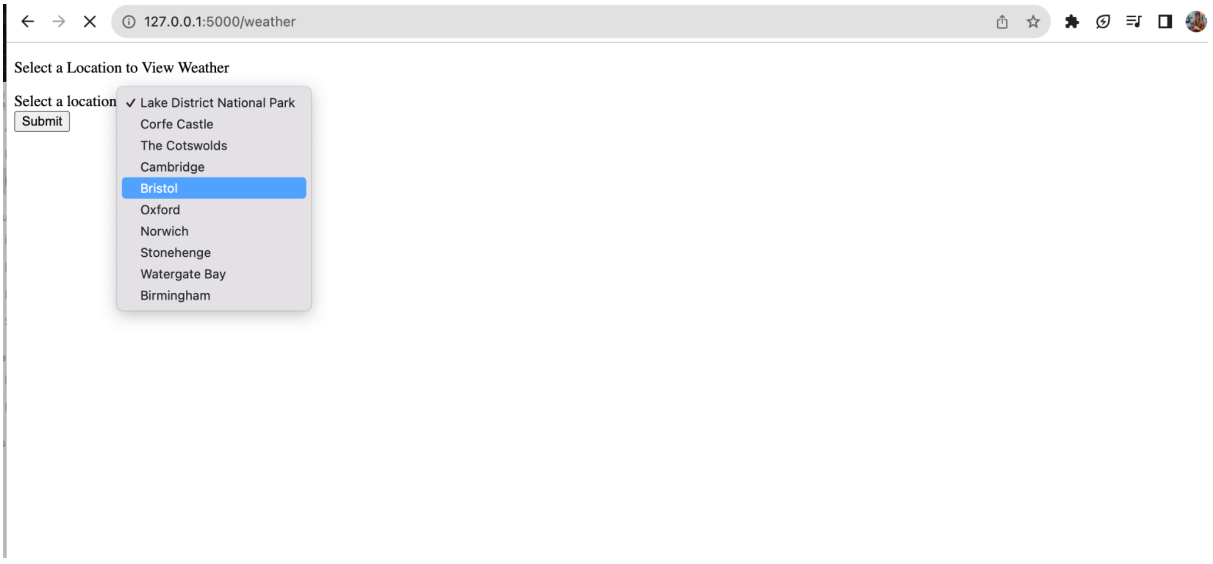
For this, I needed a new API key, weather_key, already saved in my environment.

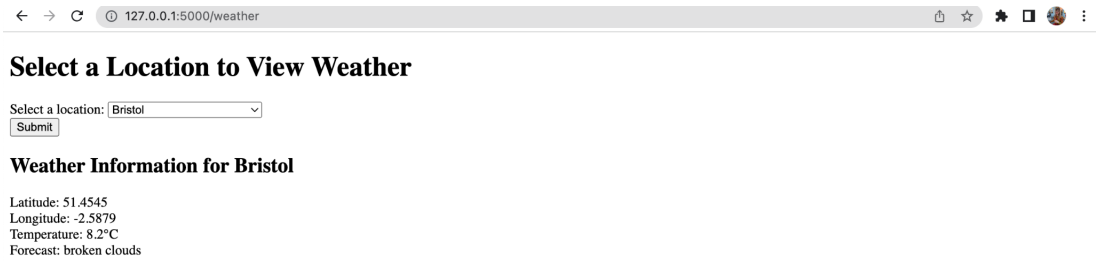


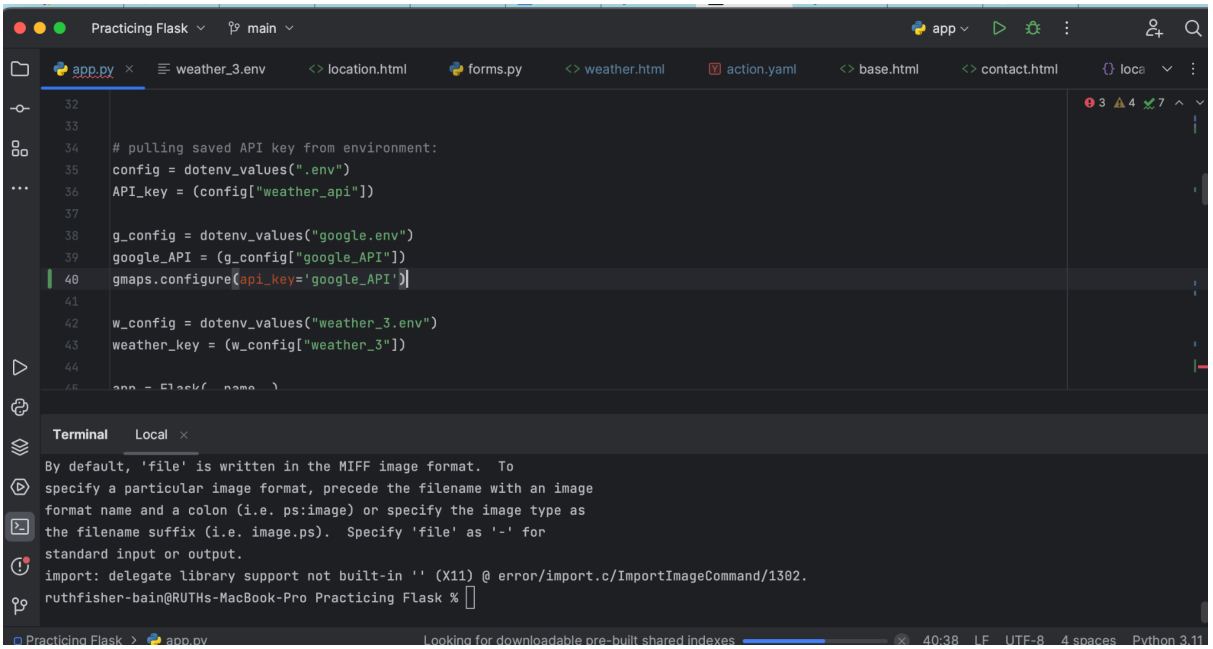
Listing coordination params, the endpoint and params are called in the response variable, and if the response status is 200, the weather data is generated in the below access attributes:



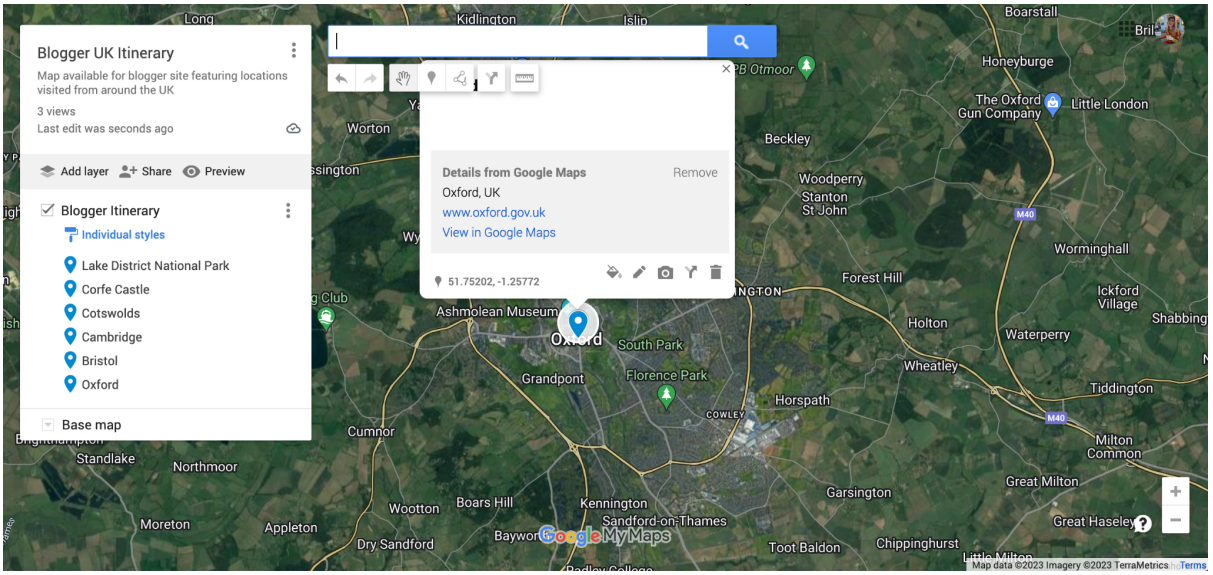
The locations function was very challenging, yet something I considered vital, making it easy for a visitor to the blogger's website to follow along with the blogger's journey with a set list of locations. Utilising forms and form validation, it was very satisfying to see it come together via the weather.html template:

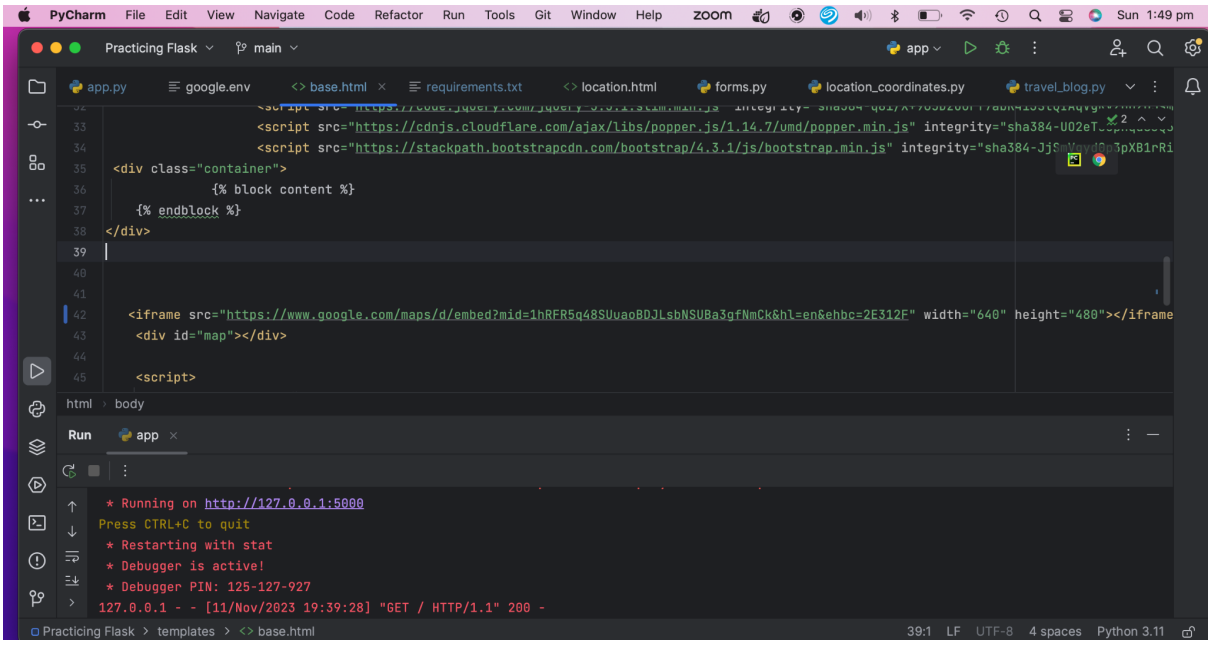




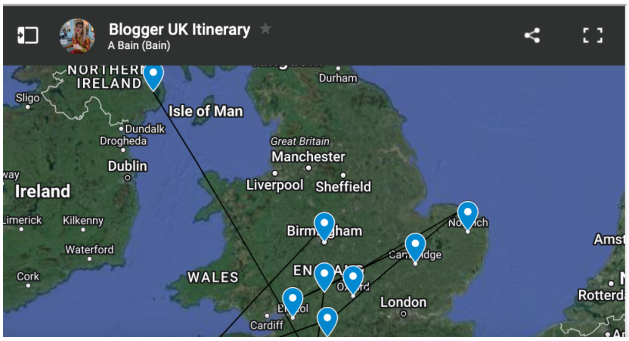


For the map, I used iframe with a map I created in the Google Cloud Console. This was a specific map with lines of the blogger's itinerary, and I added the link provided to my base.html file:

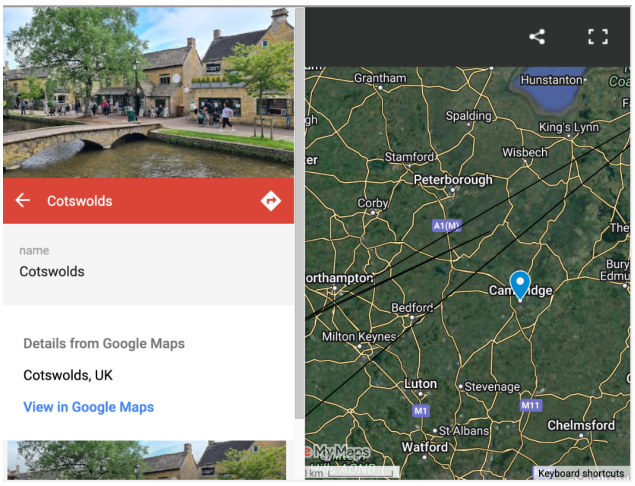


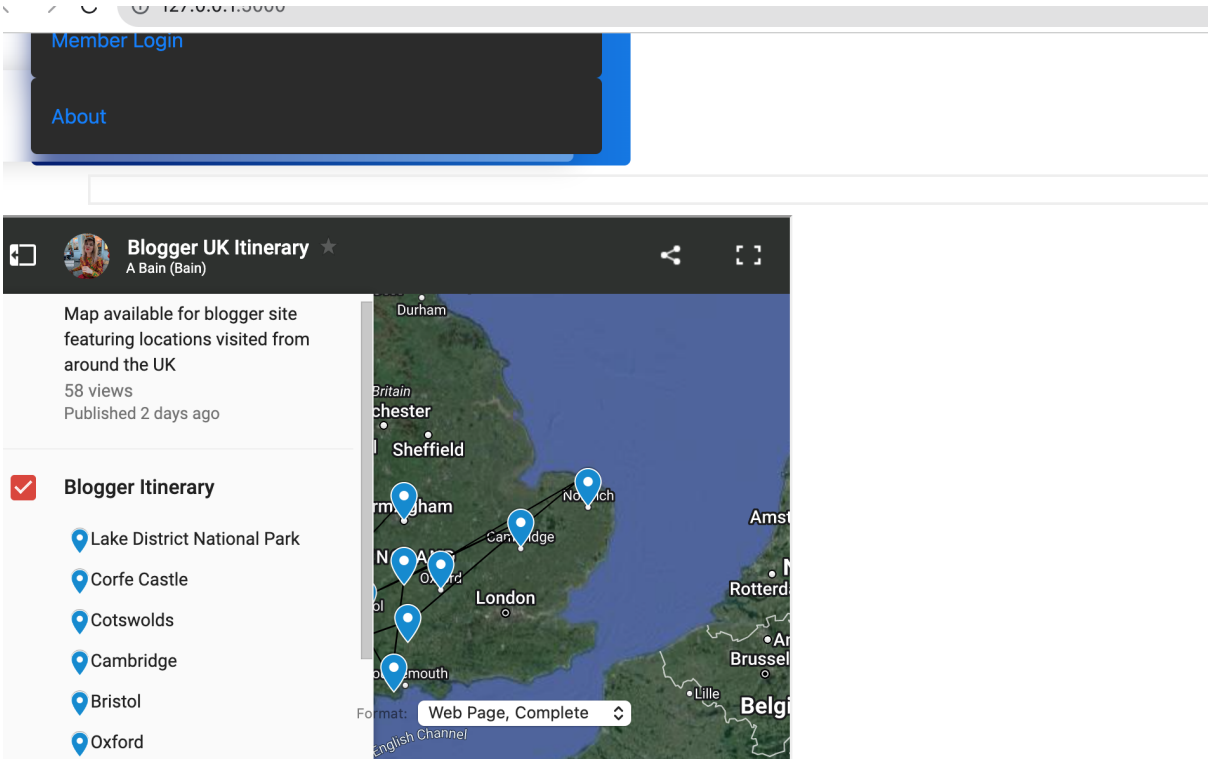


I liked the format and look of, set underneath the main/navigational list, and giving the user distances and a visual representation of the blogger's itinerary:

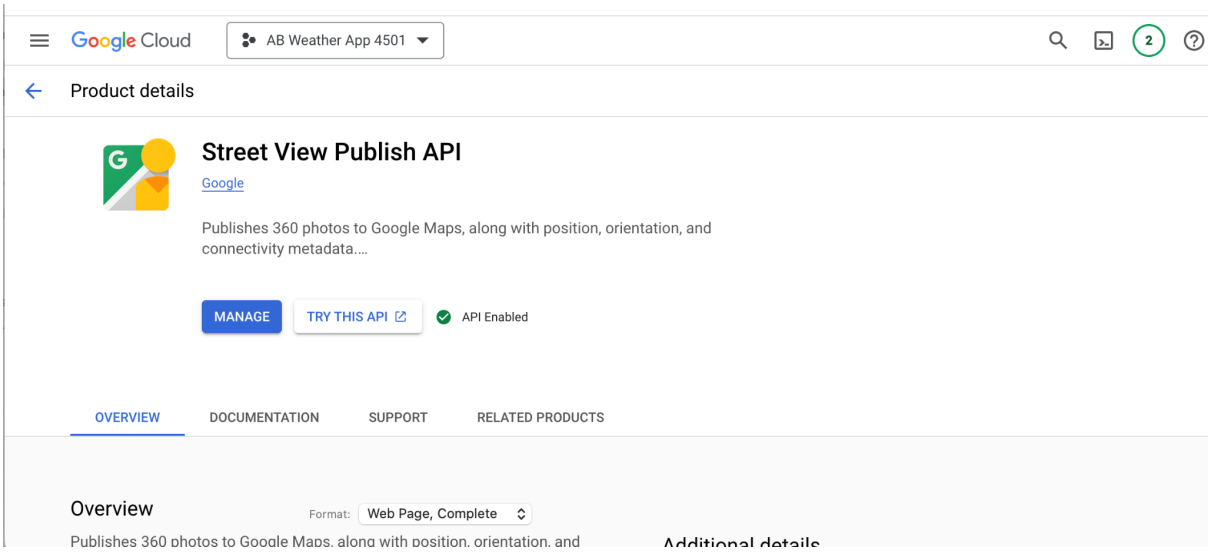


Current Temperature: 11.49°C

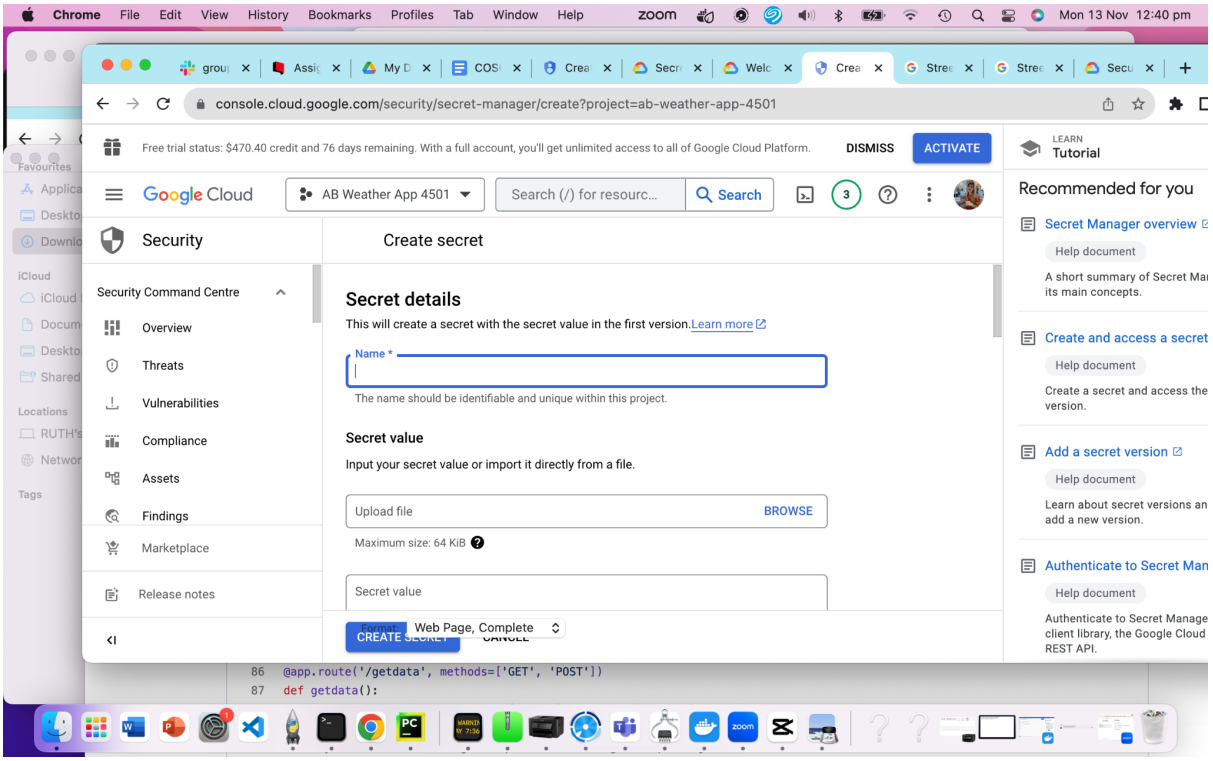




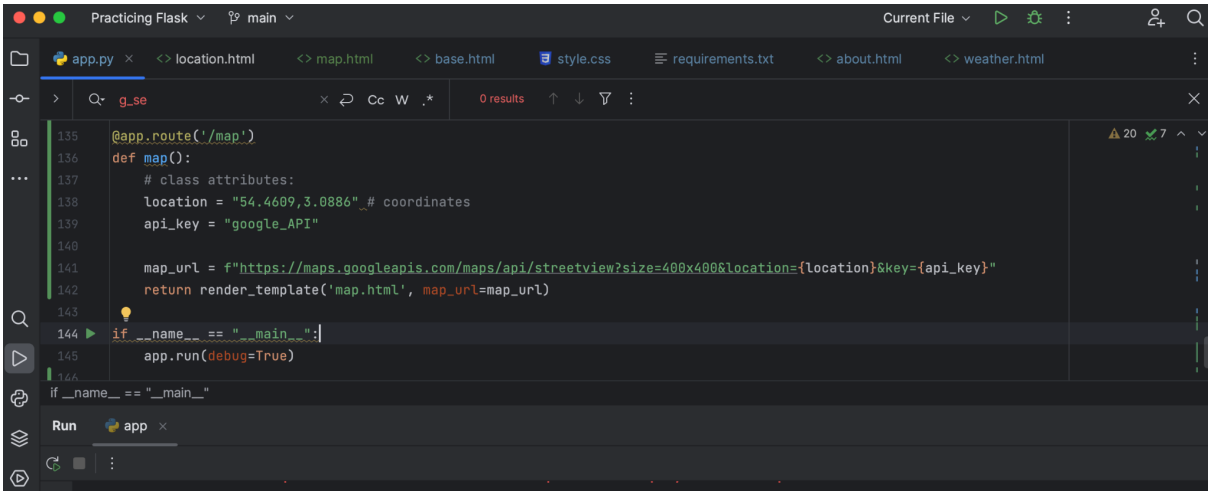
Wanting to better understand the google maps API process, I enabled the StreetView Maps API in the developer section of Google Cloud:

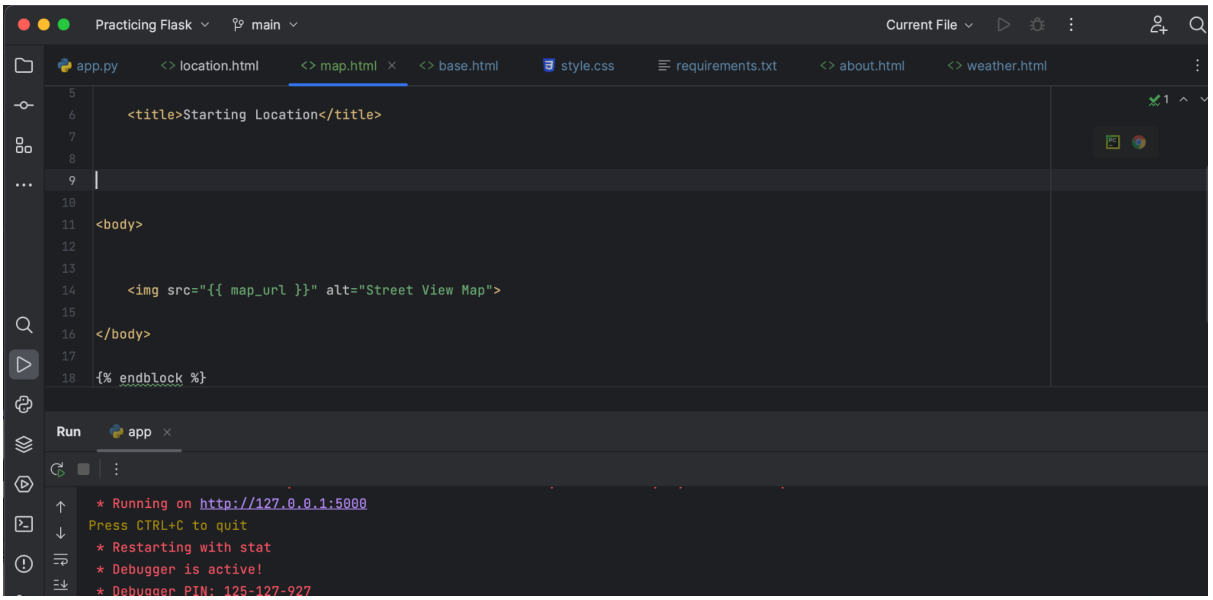


Once enabled, I created a secret key:



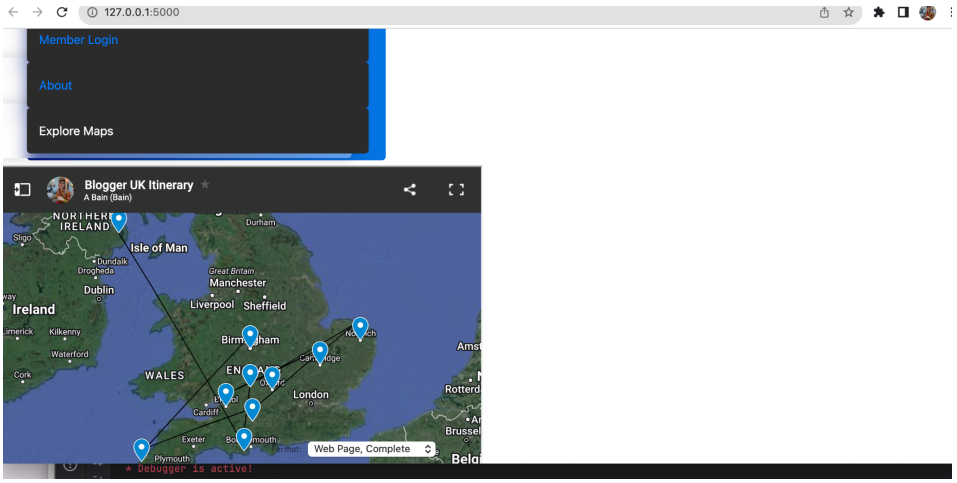
And used the Street View Map API documentation to pull data from the Google maps API:

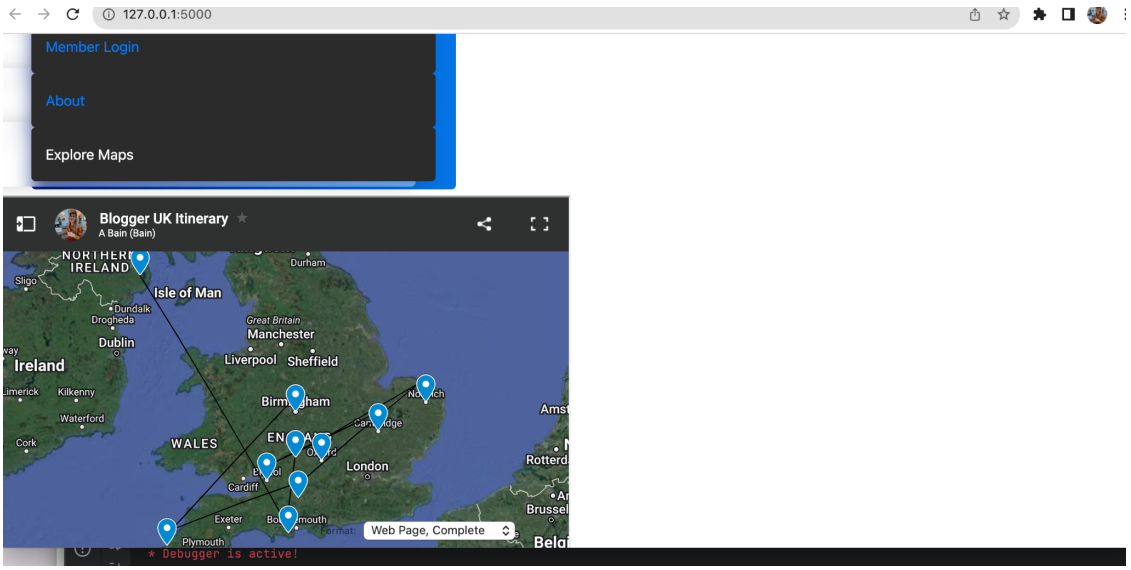




I was hoping to generate a whole new StreetView map page with Lakes District National Park as the starting location, but will develop in later iterations when I have worked more closely with Google Maps APIs.

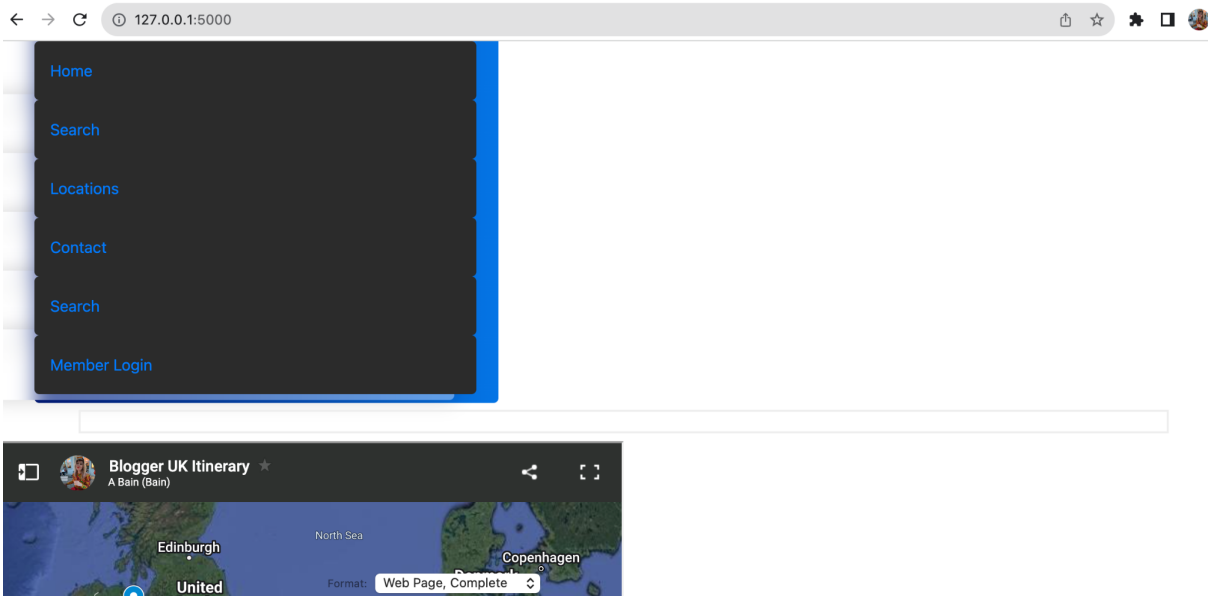
“Explore Places” currently sits above the map and acts as the title for this but only refreshes the page/ map when submitted:





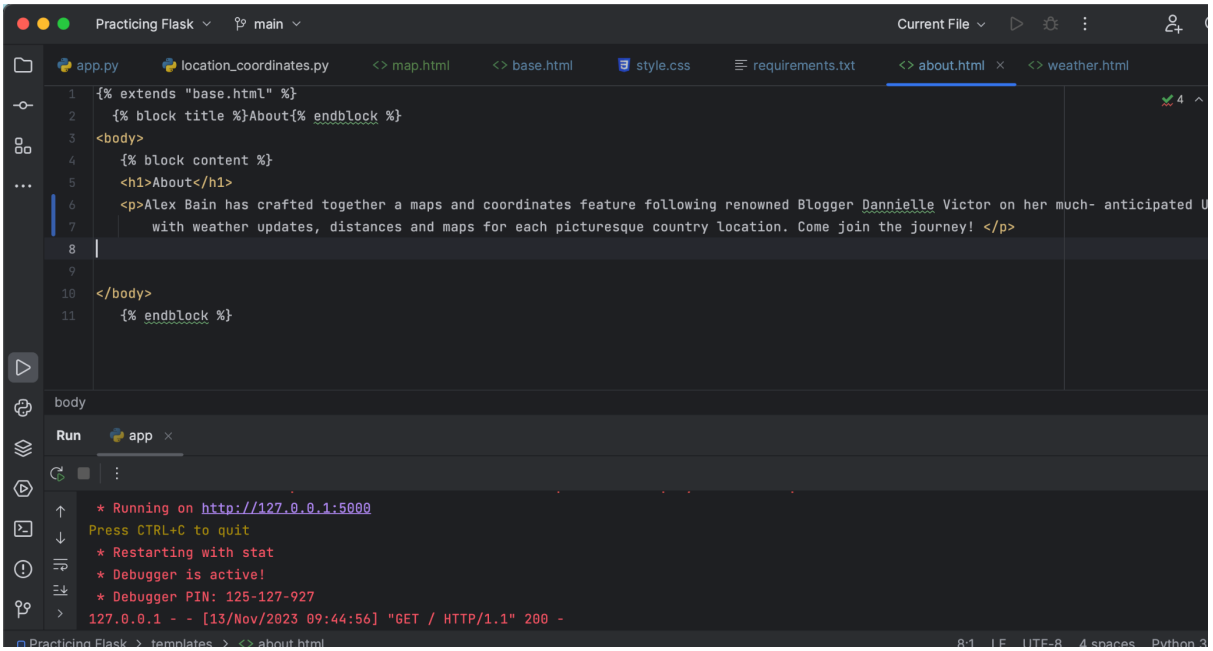
Other features of the site are:

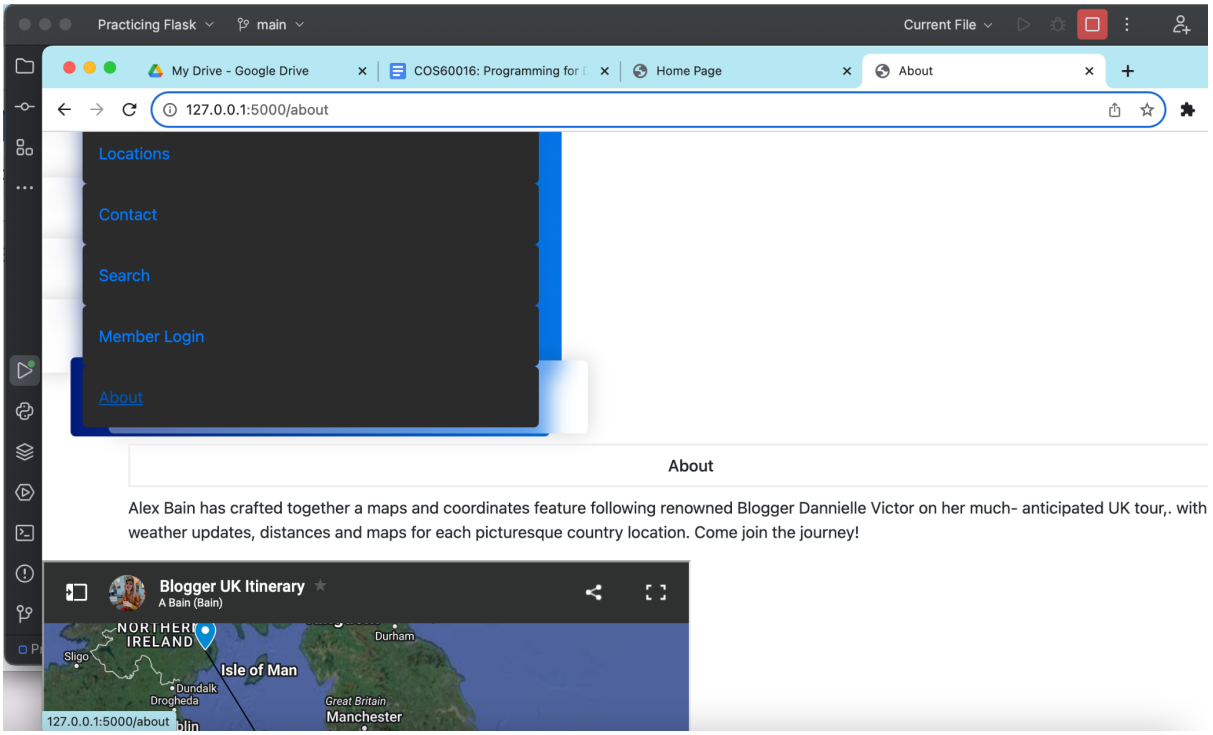
- The weather home page. This is the app.py landing page itself, and where the user is when they are looking at the navigation):



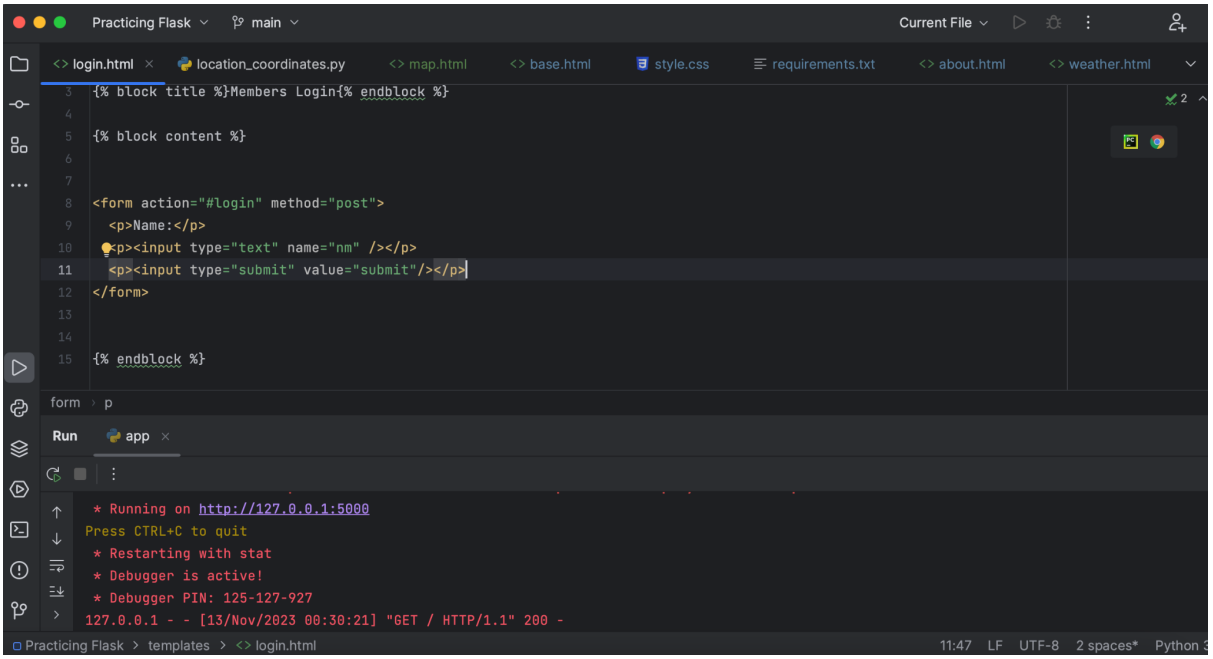
- A simple “about” section, with text:

```
75
76
77
    ALEX BAIN
78  @app.route("/about")
79  def about():
80      return render_template('about.html')
81
82  # adding a weather route to pair link to base html's "Locations" button for a drop-down list of blogger locations:
    ALEX BAIN *
```



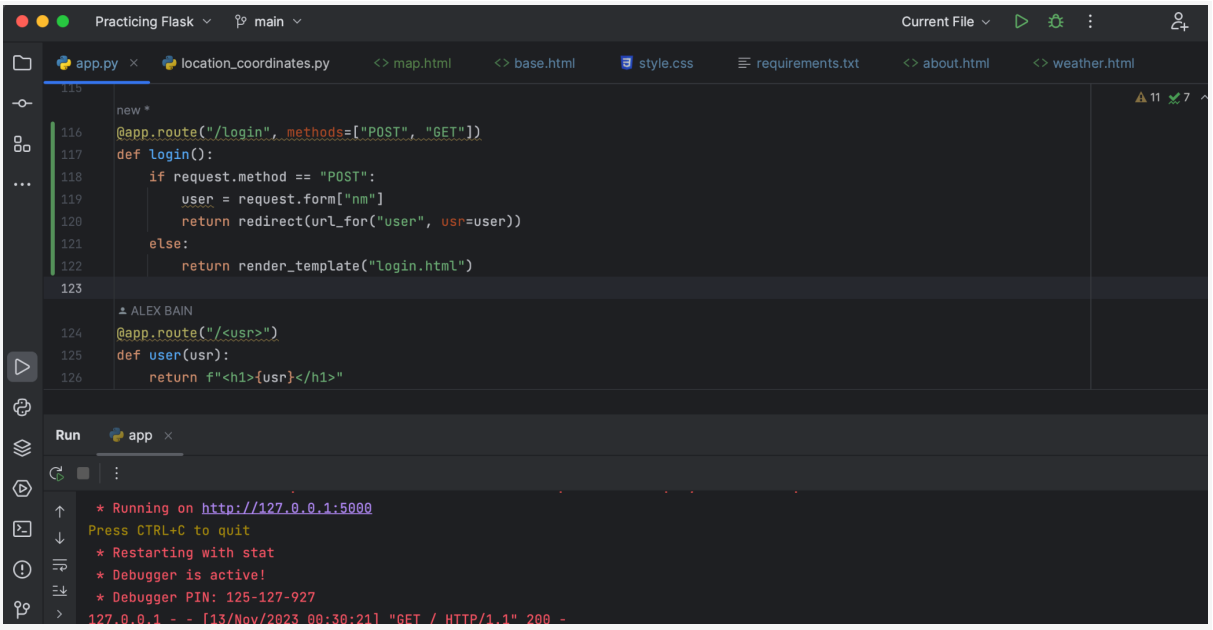


- A “login” form, which, like all other files extending base.html, is built in html blocks and saved in the templates folder:

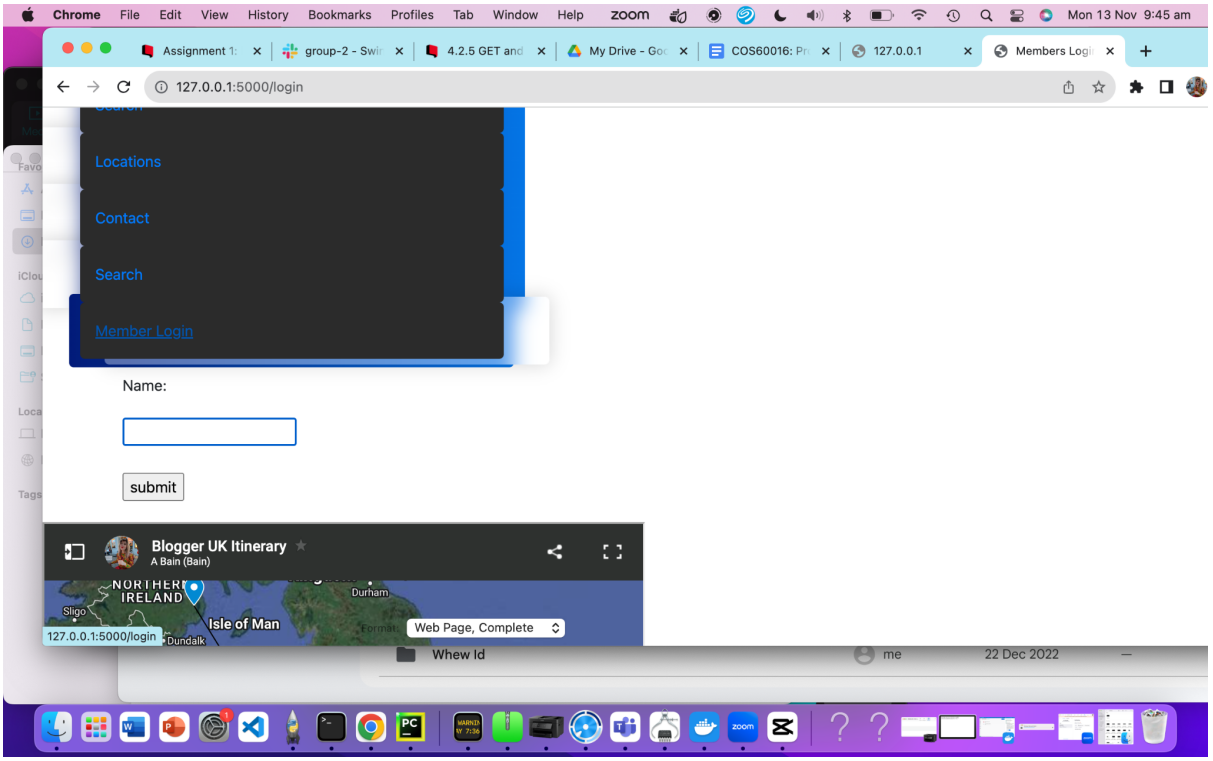


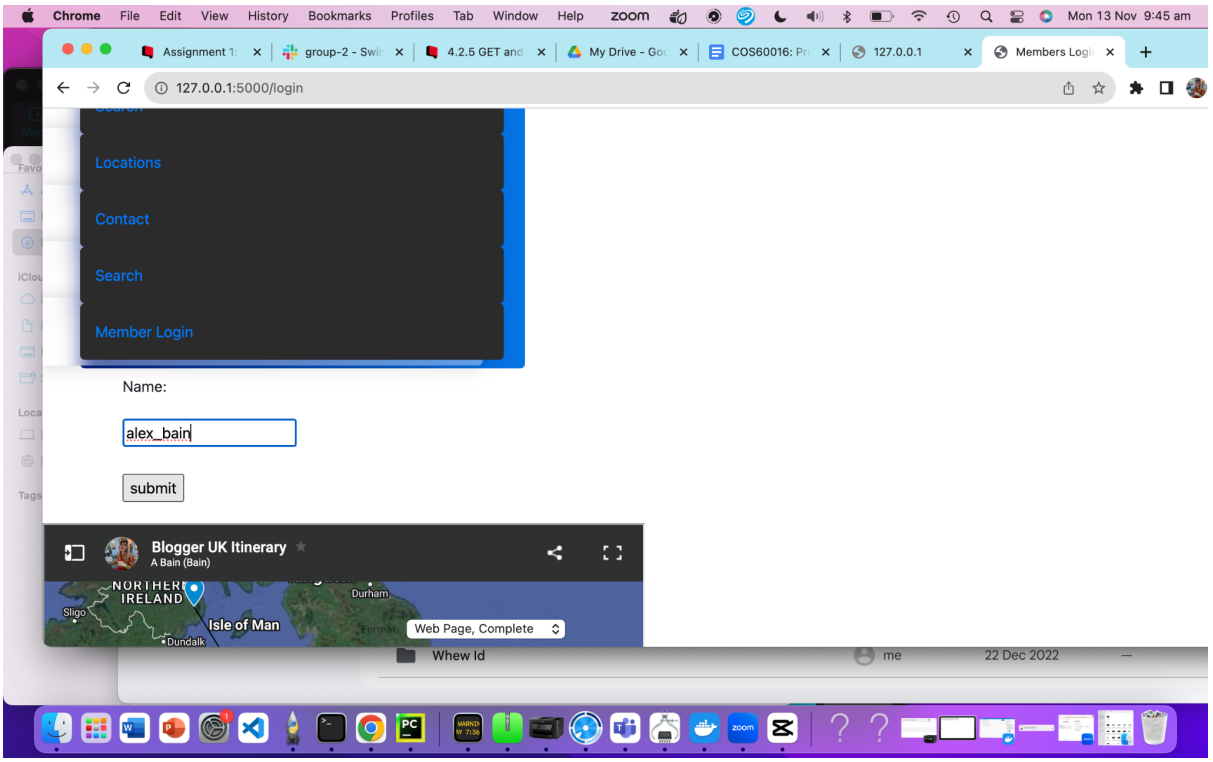
In further iterations of the app, I can see the login form accessing a member portal where the user can personalise their data.

I add the **login** app.route to app.py, writing a GET and POST request and creating two new routes to two new endpoints: where the login form is held, and the route the customer takes once they have logged in;



Testing the requests:


Chrome File Edit View History Bookmarks Profiles Tab Window Help zoom Mon 13 Nov 9:45 am
127.0.0.1:5000/login
Locations
Contact
Search
Member Login
Name:
submit
Blogger UK Itinerary
A Bain (Bain)
NORTHERN IRELAND
Sligo
Isle of Man
Dundalk
Durham
127.0.0.1:5000/login
Web Page, Complete
Whew Id
me
22 Dec 2022


Chrome File Edit View History Bookmarks Profiles Tab Window Help zoom Mon 13 Nov 9:45 am
127.0.0.1:5000/login
Locations
Contact
Search
Member Login
Name:
alex_bain
submit
Blogger UK Itinerary
A Bain (Bain)
NORTHERN IRELAND
Sligo
Isle of Man
Dundalk
Durham
Web Page, Complete
Whew Id
me
22 Dec 2022

I didn’t feel like this section needed a heading as an addition to a website, so I committed and pushed these updates to the github address.

I can already see the enormous benefits of learning API and Python methods. I have learned an invaluable lesson on the backend process through APIs, HTTP methods and Object Oriented Programming.