

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

Student Name: Alexandra Bain

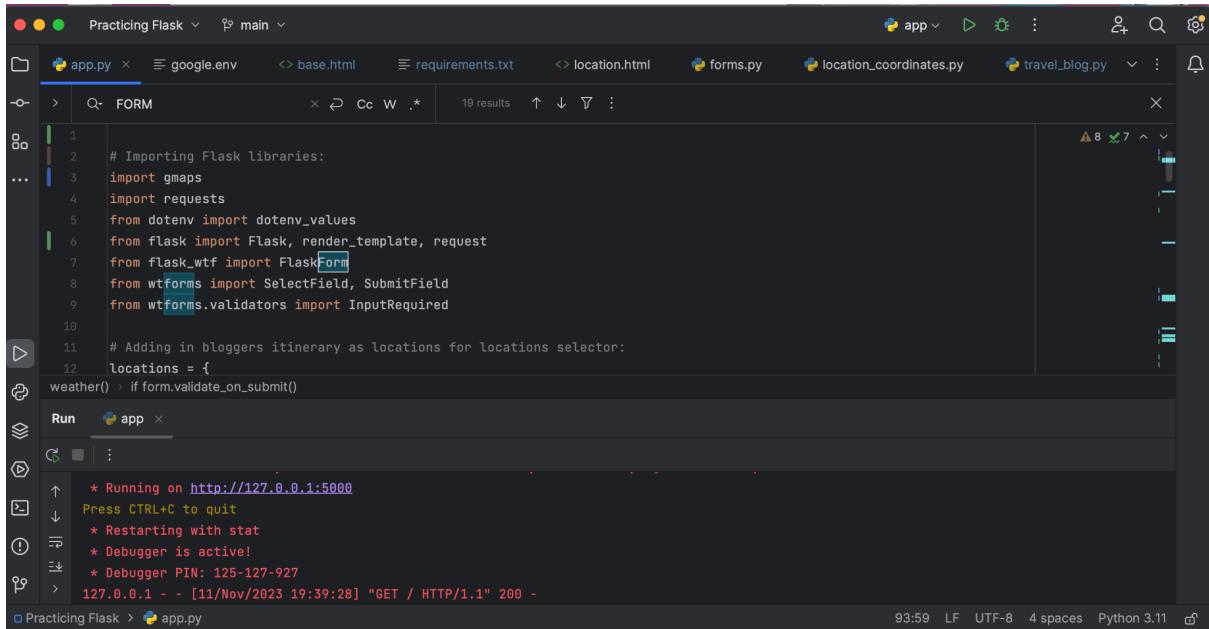
Student Number: 103083826

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 toolbar, 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 screenshot shows a code editor window titled "Practicing Flask". The current file is "app.py". The code in the editor is as follows:

```
 1 # Importing Flask libraries:
 2 import gmaps
 3 import requests
 4 from dotenv import dotenv_values
 5 from flask import Flask, render_template, request
 6 from flask_wtf import FlaskForm
 7 from wtforms import SelectField, SubmitField
 8 from wtforms.validators import InputRequired
 9
10
11 # Adding in bloggers itinerary as locations for locations selector:
12 locations = {
weather() > if form.validate_on_submit()
}
Run app
```

The terminal at the bottom shows the application running:

```
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 125-127-927
127.0.0.1 - - [11/Nov/2023 19:39:28] "GET / HTTP/1.1" 200 -
```

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:

The screenshot shows the PyCharm IDE interface with the following details:

- Project Structure:** Shows files like pp.py, base.html, requirements.txt, location.html, forms.py, location_coordinates.py, travel_blog.py, and weather.html.
- Code Editor:** The base.html file is open, displaying a navigation bar template with links to Home, About, Locations, Contact, Search, and 3-Day Forecast.
- Run Tool Window:** The app configuration is selected, showing the application is running on `http://127.0.0.1:5000`. Other logs include restarting with stat, debugger status, and a successful HTTP request from port 127.0.0.1.

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

The screenshot shows a code editor window for a Flask application named "Practicing Flask". The current file is "base.html", which contains the following code:

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <!-- Bootstrap CSS -->
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css" integrity="sha384-gg0yR0iXcbMqV3Xi...>
    <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}>
<title>{% block title %}AB Locations Website{% endblock %}</title>
</head>
<body>
```

The terminal pane shows the command: /Users/ruthfisher-bain/.zshrc:12: command not found: echo export weather_3="21ee57cc48c3e65295c11d3a13a0de8e

The screenshot shows the same code editor window after modifications. The "base.html" file now includes additional JavaScript imports and a container div:

```
...>
<!-- jQuery first, then Popper.js, then Bootstrap JS -->
<script src="https://maps.googleapis.com/maps/api/js?key={{ api_key }}"></script>
<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-q8i/X+965Dz00rT7abK4JStQIAqVgRVzbzo5sm...>
<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js" integrity="sha384-U2eT0CpHqdSJQ6...>
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js" integrity="sha384-JJSmVgyd0p3pXB1rRi...>
<div class="container">
    {% block content %}
    {% endblock %}
</div>
```

The terminal pane shows the same command error: /Users/ruthfisher-bain/.zshrc:12: command not found: echo export weather_3="21ee57cc48c3e65295c11d3a13a0de8e

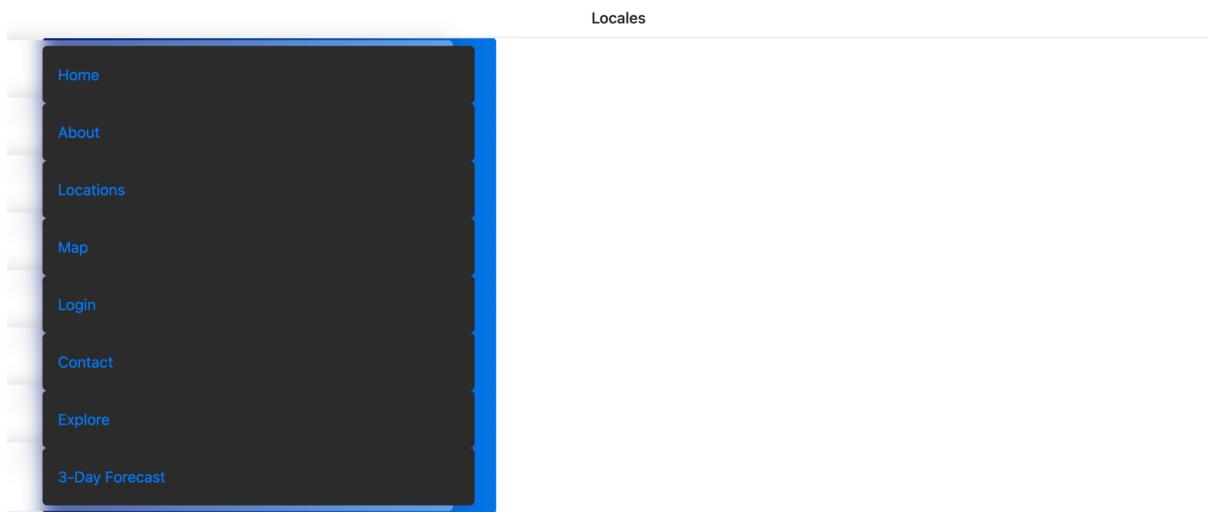
The base.html file with css styling (file style.css lives in the “static” folder), has made this list a navigational tool. Every other html file in the templates folder is an extension of this base.html file:

```

1  {% extends "base.html" %}
2
3
4  {% block title %}Home Page{% endblock %}
5  {% block content %}
6    <h1>{{content}}</h1>
7
8
9
10 {% endblock %}

```

Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat



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

```

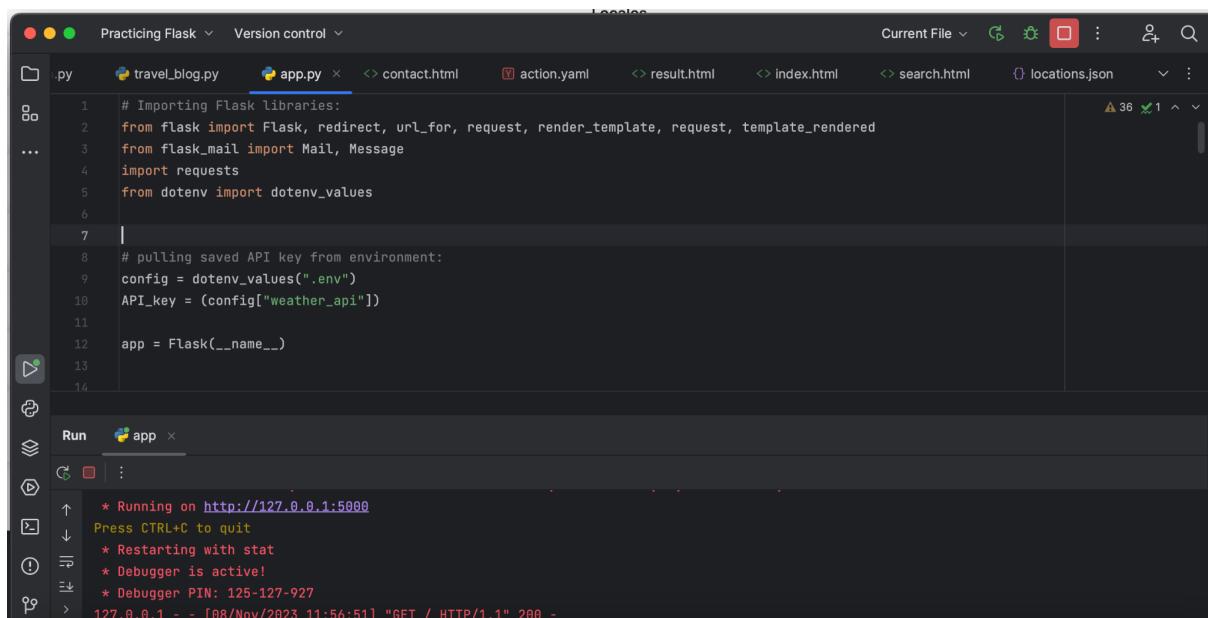
30
31 ul li {
32   width: 450px;
33   height: 60px;
34   display: flex;
35   align-items: center;
36   margin-right: 120px;
37   margin: 1.5em;
38   cursor: pointer;
39   padding: 1em;
40   background: rgb(43, 43, 43);
41   position: relative;
42   color: white;
43   border-radius: 5px;
44 }

```

Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 125-127-927
127.0.0.1 - - [12/Nov/2023 23:56:21] "GET / HTTP/1.1" 200 -

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:



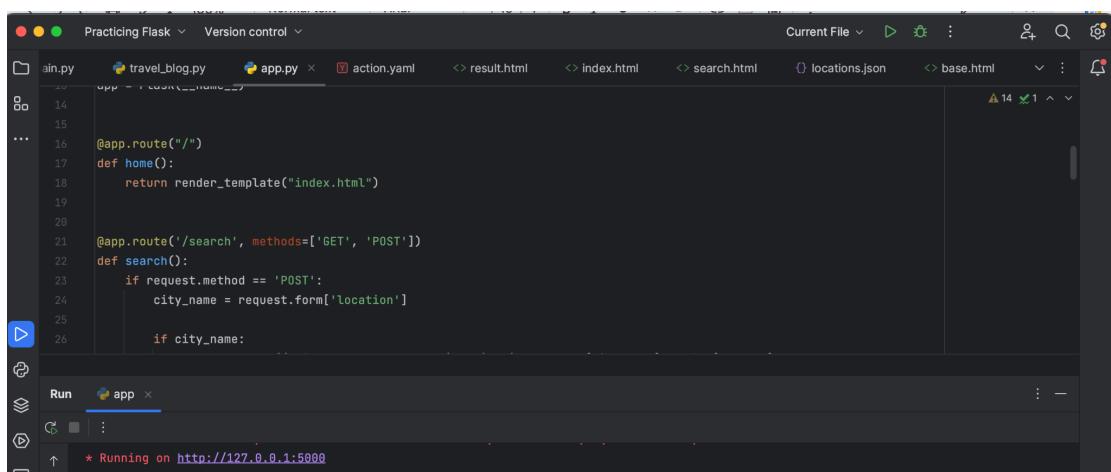
The screenshot shows a code editor interface with the following details:

- File Explorer:** Shows files: .py, travel_blog.py, app.py (selected), contact.html, action.yaml, result.html, index.html, search.html, locations.json.
- Code Editor:** Content of app.py:

```
1 # Importing Flask libraries:
2 from flask import Flask, redirect, url_for, request, render_template, response, template_rendered
3 from flask_mail import Mail, Message
4 import requests
5 from dotenv import dotenv_values
6
7
8 # pulling saved API key from environment:
9 config = dotenv_values(".env")
10 API_key = (config["weather_api"])
11
12 app = Flask(__name__)
13
14
```
- Run Tab:** Set to "app".
Output:

```
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
  * Debugger is active!
  * Debugger PIN: 125-127-927
> 127.0.0.1 - - [08/Nov/2023 11:56:51] "GET / HTTP/1.1" 200 -
```

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



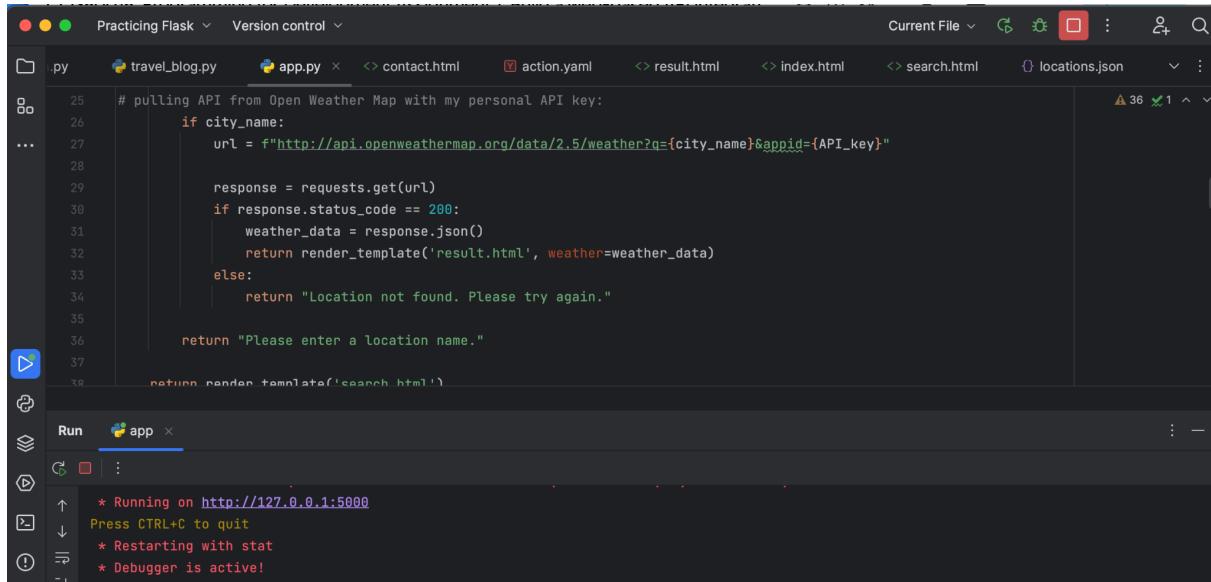
The screenshot shows a code editor interface with the following details:

- File Explorer:** Shows files: .py, travel_blog.py, app.py (selected), action.yaml, result.html, index.html, search.html, locations.json, base.html.
- Code Editor:** Content of app.py:

```
14
15
16 @app.route("/")
17 def home():
18     return render_template("index.html")
19
20
21 @app.route('/search', methods=['GET', 'POST'])
22 def search():
23     if request.method == 'POST':
24         city_name = request.form['location']
25
26         if city_name:
```
- Run Tab:** Set to "app".
Output:

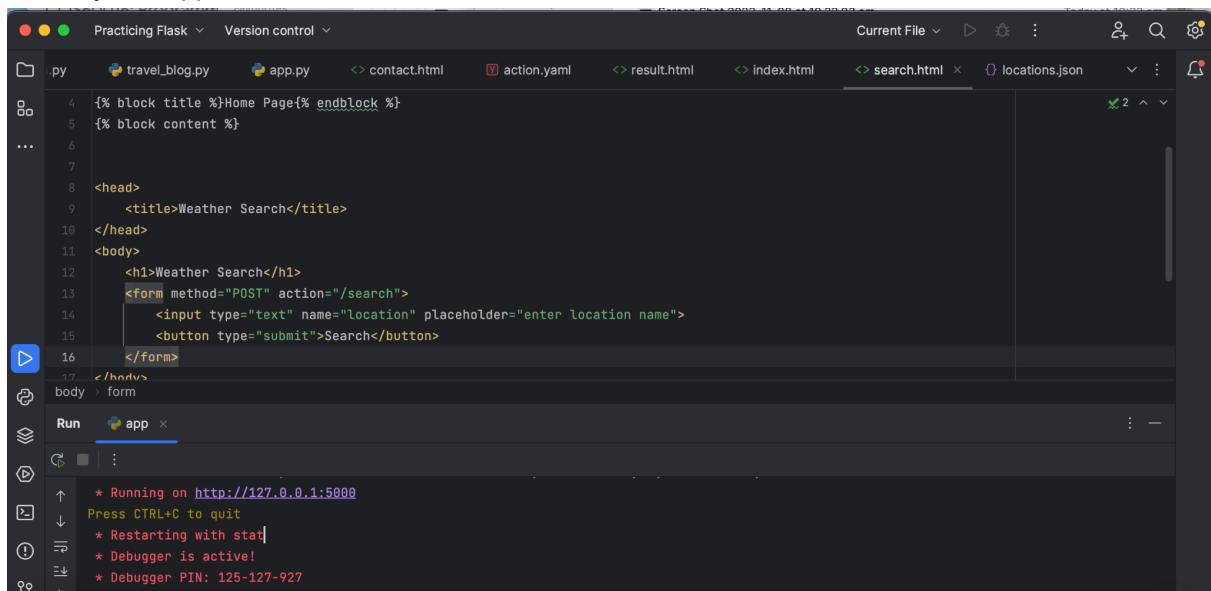
```
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

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



```
25     # pulling API from Open Weather Map with my personal API key:
26     if city_name:
27         url = f"http://api.openweathermap.org/data/2.5/weather?q={city_name}&appid={API_key}"
28
29         response = requests.get(url)
30         if response.status_code == 200:
31             weather_data = response.json()
32             return render_template('result.html', weather=weather_data)
33         else:
34             return "Location not found. Please try again."
35
36     return "Please enter a location name."
37
38     return render_template('search.html')
```

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



```
{% block title %}Home Page{% endblock %}
{% block content %}

<head>
    <title>Weather Search</title>
</head>
<body>
    <h1>Weather Search</h1>
    <form method="POST" action="/search">
        <input type="text" name="location" placeholder="enter location name">
        <button type="submit">Search</button>
    </form>
</body>
</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 screenshot shows a code editor interface with the following details:

- Title Bar:** Practicing Flask, Version control
- File Explorer:** Shows travel_blog.py, app.py, base.html, weather_3.html, contact.html, result.html (current file), index.html, and search.html.
- Code Editor:** Result.html file content:

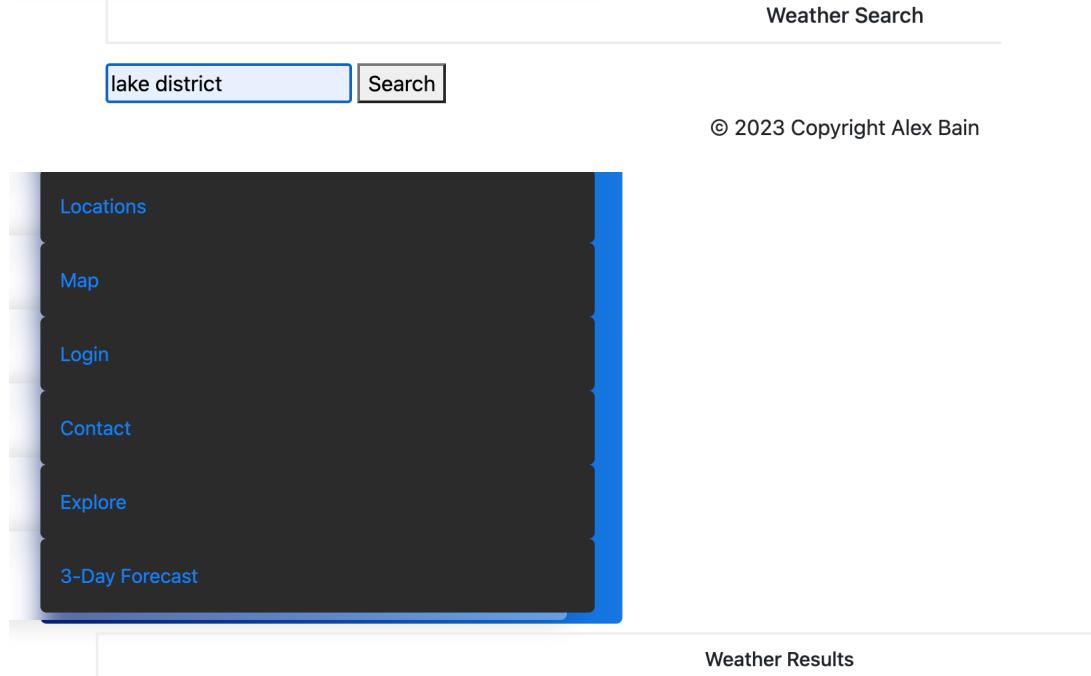
```
1  {% extends "base.html" %}\n2\n3  {% block content %}\n4      <title>Weather Results</title>\n5      <head>\n6          <body>\n7              <h1>Weather Results</h1>\n8              <p>Location: {{ weather.name }}</p>\n9              <p>Current Temperature: {{ temperature }}°C</p>\n10\n11         </body>\n12\n13     {% endblock %}\nbody
```
- Run Control:** Run configuration for app.py.
- Terminal:** Output of the run command:

```
* Running on http://127.0.0.1:5000\nPress CTRL+C to quit\n* Restarting with stat\n  * Debugger is active!
```

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:

← → ⌛ ① 127.0.0.1:5000/search

Search

3-Day Forecast

Weather Results

Location: Birmingham

Current Temperature: 11.94°C

Blogger UK Itinerary A Bain (Bain)

NORTHERN IRELAND, ISLE OF MAN, GREAT BRITAIN, ENGLAND, WALES, IRELAND, DUBLIN, LIMERICK, CORK, DUNDALK, DROGHEDA, MANCHESTER, LIVERPOOL, SHEFFIELD, BIRMINGHAM, NEWCASTLE, CARDIFF, EXETER, OXFORD, CAMBRIDGE, DURHAM, AMSTERDAM, ROTTERDAM.

Web Page, Complete

User Sel | group-2 | Checking | COS600 | Assignm | Checkou | Home Pe | Weather | Home Pe | +

← → ⌛ ① 127.0.0.1:5000/search

About

Locations

Contact

Search

3-Day Forecast

Weather Search

Birmingham

Blogger UK Itinerary A Bain (Bain)

NORTHERN IRELAND, ISLE OF MAN, GREAT BRITAIN, MANCHESTER, DUBLIN, LIMERICK, CORK, DUNDALK, DROGHEDA, LIVERPOOL, SHEFFIELD, BIRMINGHAM, NEWCASTLE, CARDIFF, EXETER, OXFORD, CAMBRIDGE, DURHAM, AMSTERDAM, ROTTERDAM.

Web Page, Complete

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':

The screenshot shows the PyCharm IDE interface. The top bar displays the project name "Practicing Flask" and the file "main". The code editor window contains the following Python code:

```
class LocationForm(FlaskForm):
    location = SelectField('Select a location', choices=[(location, location) for location in locations.keys()], validators=[InputRequired()])
    submit = SubmitField('Submit')

# pulling saved API key from environment:
config = dotenv_values('.env')
API_key = (config["weather_api"])

g_config = dotenv_values("google.env")
google_API = (g_config["google_API"])
```

The terminal window below shows the application running on port 5000:

```
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
  * Debugger is active!
  * Debugger PIN: 125-127-927
127.0.0.1 - - [11/Nov/2023 19:39:28] "GET / HTTP/1.1" 200 -
```

At the bottom, the status bar indicates the time is 33:31, and the Python version is 3.11.

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

The screenshot shows the PyCharm IDE interface. The top bar displays the project name "Practicing Flask" and the file "main". The code editor window contains the following Python code:

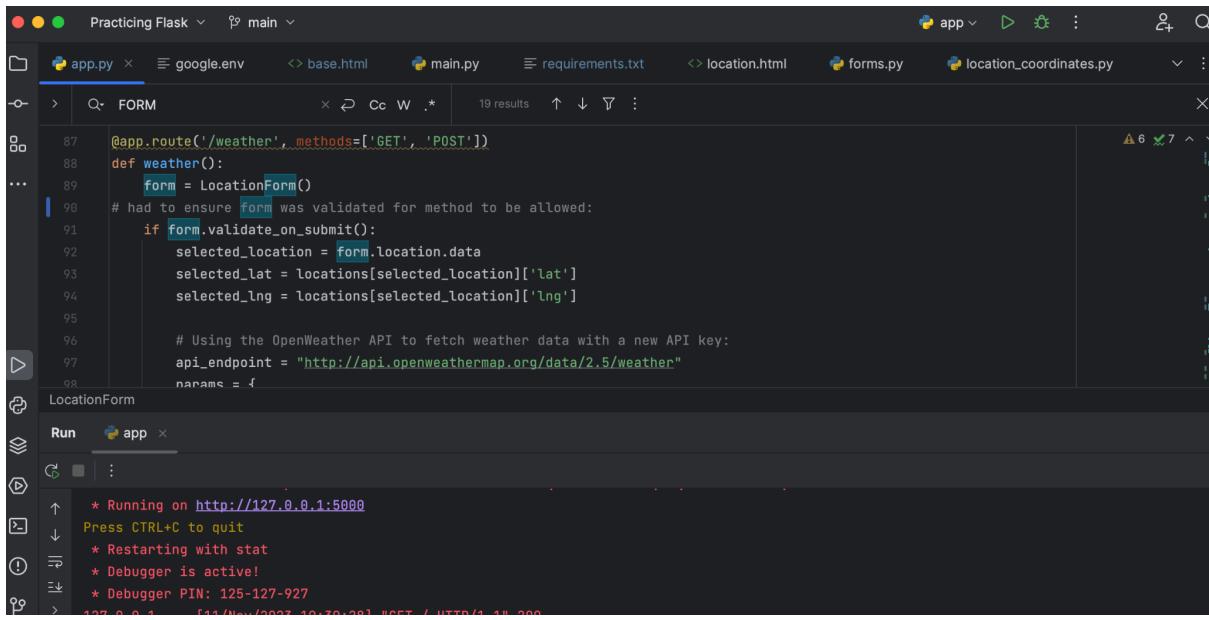
```
from dotenv import dotenv_values
from flask_wtf import FlaskForm
from wtforms import SelectField, SubmitField
from wtforms.validators import InputRequired

locations = {
    "Lake District National Park": {"lat": 54.4609, "lng": -3.0886},
    "Confe Castle": {"lat": 50.6419, "lng": -2.0554},
    "The Cotswolds": {"lat": 51.9294, "lng": -1.7203},
    "Cambridge": {"lat": 52.2053, "lng": 0.1218},
    "Bristol": {"lat": 51.4545, "lng": -2.5879},
    "Oxford": {"lat": 51.752, "lng": -1.2577},
    "Norwich": {"lat": 52.3700, "lng": -1.2076}
```

The terminal window below shows the application running on port 5000:

```
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
  * Debugger is active!
  * Debugger PIN: 125-127-927
127.0.0.1 - - [09/Nov/2023 13:55:19] "GET / HTTP/1.1" 200 -
```

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



```
87     @app.route('/weather', methods=['GET', 'POST'])
88     def weather():
89         form = LocationForm()
90         # had to ensure form was validated for method to be allowed:
91         if form.validate_on_submit():
92             selected_location = form.location.data
93             selected_lat = locations[selected_location]['lat']
94             selected_lng = locations[selected_location]['lng']
95
96             # Using the OpenWeather API to fetch weather data with a new API key:
97             api_endpoint = "http://api.openweathermap.org/data/2.5/weather"
98             params = {
```

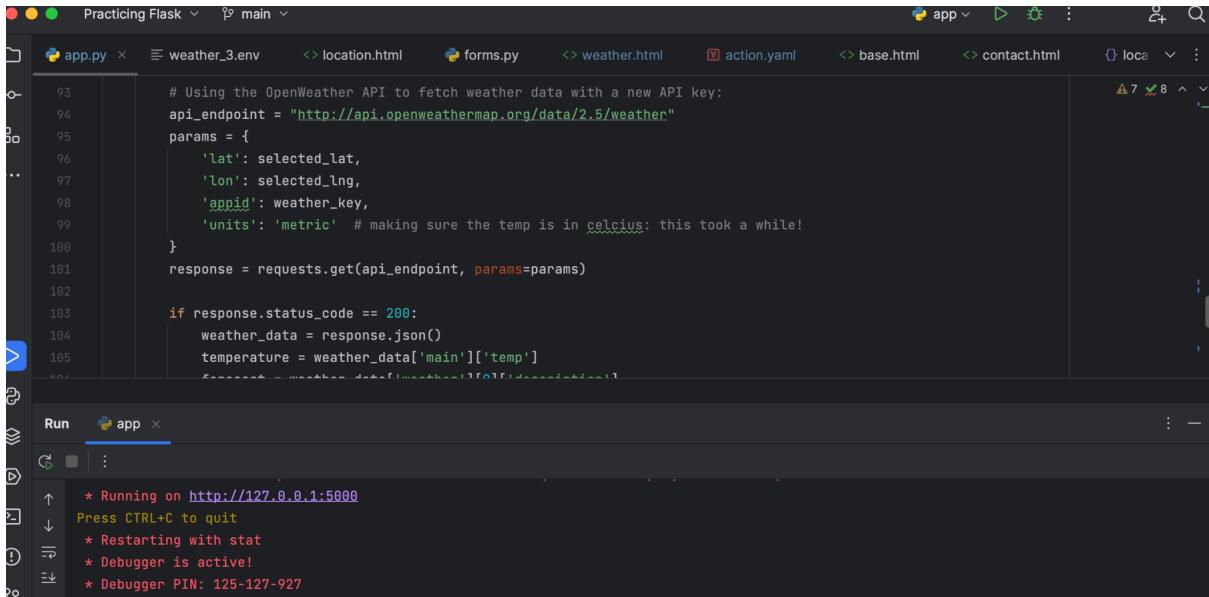
LocationForm

Run app

```
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 125-127-927
127.0.0.1 - - [14/Nov/2023 10:10:29] "GET / HTTP/1.1" 200
```

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.



```
93     # Using the OpenWeather API to fetch weather data with a new API key:
94     api_endpoint = "http://api.openweathermap.org/data/2.5/weather"
95     params = {
96         'lat': selected_lat,
97         'lon': selected_lng,
98         'appid': weather_key,
99         'units': 'metric' # making sure the temp is in celcius: this took a while!
100     }
101     response = requests.get(api_endpoint, params=params)
102
103     if response.status_code == 200:
104         weather_data = response.json()
105         temperature = weather_data['main']['temp']
```

Run app

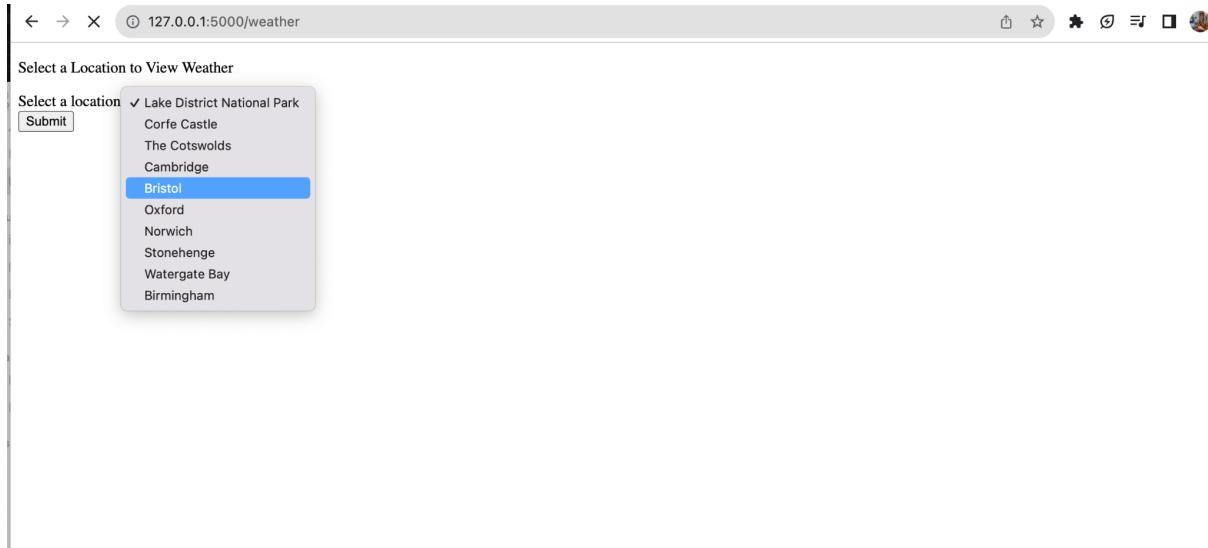
```
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 125-127-927
```

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 screenshot shows a code editor interface with several tabs open. The main tab is 'app.py' which contains Python code for a Flask application. The code includes logic for fetching weather data from an API and rendering it in a template. Below the code editor is a 'Run' section showing the application's output:

```
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
① * Debugger is active!
② * Debugger PIN: 125-127-927
> 127.0.0.1 - - [09/Nov/2023 13:55:19] "GET / HTTP/1.1" 200 -
```

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:



← → ⌛ ① 127.0.0.1:5000/weather

Select a Location to View Weather

Select a location: Bristol

Weather Information for Bristol

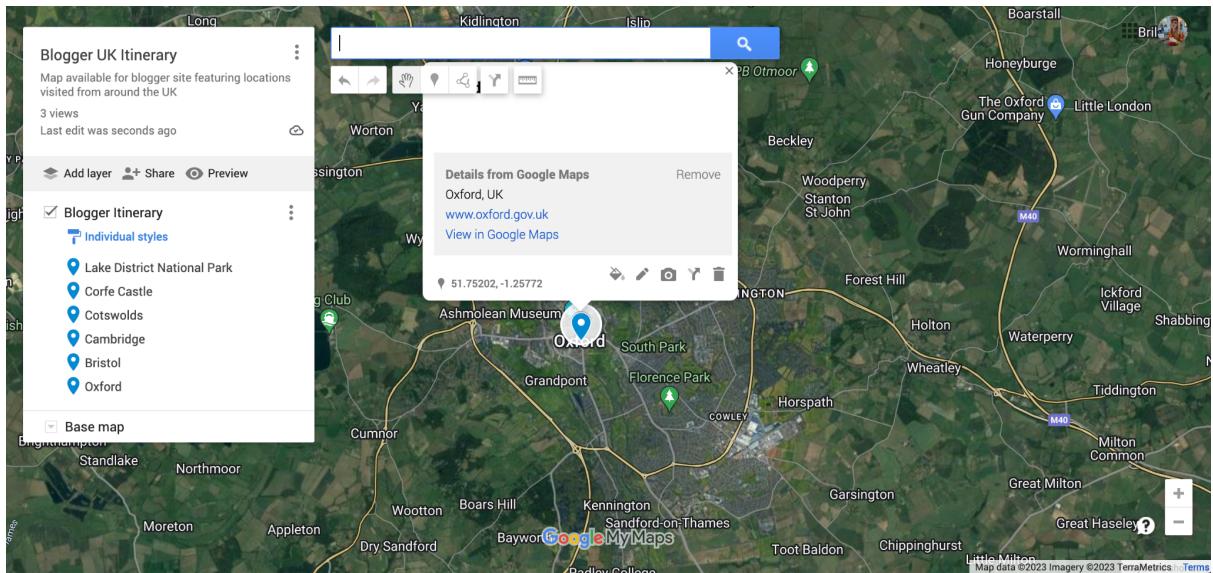
Latitude: 51.4545
Longitude: -2.5879
Temperature: 8.2°C
Forecast: broken clouds

The screenshot shows a code editor interface with multiple files open. The main file is `app.py`, which contains Python code for a Flask application. The code includes imports for `dotenv_values` and `gmaps`, environment variable configuration from `weather_3.env`, and the creation of a Flask application. The terminal below shows the command `python app.py` being run, and the output indicates that the application is running on port 5000.

```
32
33
34 # pulling saved API key from environment:
35 config = dotenv_values(".env")
36 API_key = (config["weather_api"])
37
38 g_config = dotenv_values("google.env")
39 google_API = (g_config["google_API"])
40 gmaps.configure(api_key='google_API')
41
42 w_config = dotenv_values("weather_3.env")
43 weather_key = (w_config["weather_3"])
44
app = Flask(__name__)

Terminal Local
By default, 'file' is written in the MIFF image format. To
specify a particular image format, precede the filename with an image
format name and a colon (i.e. ps:image) or specify the image type as
the filename suffix (i.e. image.ps). Specify 'file' as '-' for
standard input or output.
import: delegate library support not built-in '' (X11) @ error/import.c/ImportImageCommand/1302.
ruthfisher-bain@RUTHs-MacBook-Pro Practicing Flask %
```

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:



The screenshot shows the PyCharm IDE interface. The top navigation bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, Git, Window, Help, and zoom. The title bar says "Practicing Flask". The code editor displays a file named "base.html" with the following content:

```
<div class="container">
    {% block content %}
    {% endblock %}
</div>

<iframe src="https://www.google.com/maps/d/embed?mid=1hRFR5q48SUuaeBDJLsbNSUBa3gfNmCk&hl=en&ehbc=2E312E" width="640" height="480"></iframe>
```

The bottom terminal window shows the application running on port 5000:

```
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
→ * Debugger is active!
→ * Debugger PIN: 125-127-927
> 127.0.0.1 - - [11/Nov/2023 19:39:28] "GET / HTTP/1.1" 200 -
```

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:

← → ⌛ ⓘ 127.0.0.1:5000

Contact

Explore

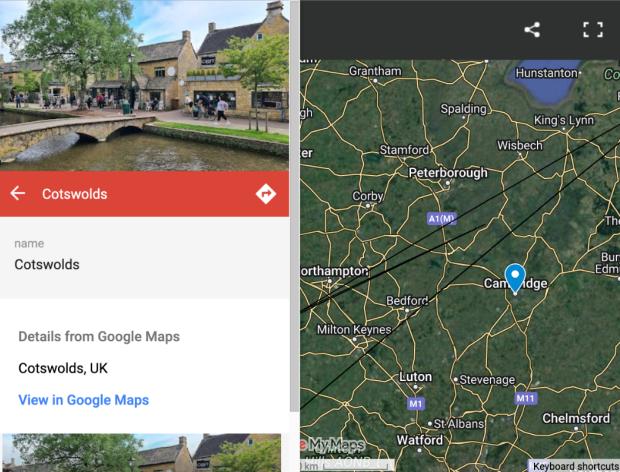
3-Day Forecast


Blogger UK Itinerary
A Bain (Bain)

NORTHERN IRELAND: Sligo, Dundalk, Drogheda, Belfast, Dublin, Limerick, Kilkenny, Waterford, Cork.

IRELAND: Isle of Man, Liverpool, Manchester, Sheffield, Birmingham, Bristol, Cardiff, London, Cambridge, Norwich, Ipswich, Newmarket, Bury St Edmunds, Chelmsford, Stevenage, Watford, St Albans, Luton, Bedford, Milton Keynes, Corby, Northampton, Peterborough, Grantham, Spalding, King's Lynn, Wisbech, Hunstanton, Fakenham, Cromer, Great Yarmouth, Norwich, Lowestoft, Ipswich, Felixstowe, Southwold, Bury St Edmunds, Chelmsford, Stevenage, Watford, St Albans, Luton, Bedford, Milton Keynes, Corby, Northampton, Peterborough, Grantham, Spalding, King's Lynn, Wisbech, Hunstanton, Fakenham, Cromer, Great Yarmouth, Norwich, Lowestoft, Ipswich, Felixstowe, Southwold.

Current Temperature: 11.49°C


Cotswolds

name
Cotswolds

Details from Google Maps
[Cotswolds, UK](#)
[View in Google Maps](#)

© 2023 Copyright Alex Bain

Blogger UK Itinerary ★
A Bain (Bain)

Map available for blogger site featuring locations visited from around the UK
58 views
Published 2 days ago

Blogger Itinerary

- 📍 Lake District National Park
- 📍 Corfe Castle
- 📍 Cotswolds
- 📍 Cambridge
- 📍 Bristol
- 📍 Oxford

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

Street View Publish API

Publishes 360 photos to Google Maps, along with position, orientation, and connectivity metadata....

MANAGE TRY THIS API API Enabled

OVERVIEW DOCUMENTATION SUPPORT RELATED PRODUCTS

Overview Format: Web Page, Complete
Publishes 360 photos to Google Maps, along with position, orientation, and Additional details

Once enabled, I created a secret key:

The screenshot shows a Google Chrome browser window on a Mac OS X desktop. The title bar indicates it's Monday, November 13, at 12:40 pm. The main content is the Google Cloud Security Center interface, specifically the 'Create secret' page. On the left, there's a sidebar with 'Security Command Centre' and various sub-options like Overview, Threats, Vulnerabilities, Compliance, Assets, Findings, Marketplace, and Release notes. The main area has a heading 'Secret details' with a note about creating a secret with a secret value in the first version. It includes fields for 'Name *' (with a placeholder 'My Secret'), 'Secret value' (with a 'Upload file' button and a 'Maximum size: 64 KiB' note), and a 'CREATE SECRET' button. To the right, there's a sidebar titled 'Recommended for you' with links to 'Secret Manager overview', 'Create and access a secret', 'Add a secret version', and 'Authenticate to Secret Manager'. The Mac OS X dock at the bottom shows various application icons.

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

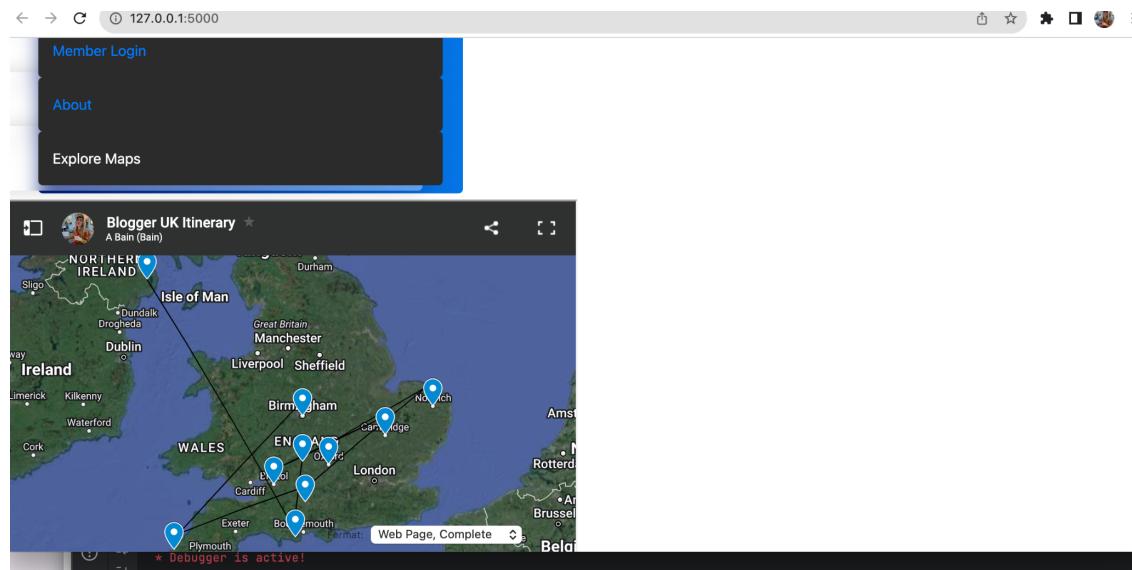
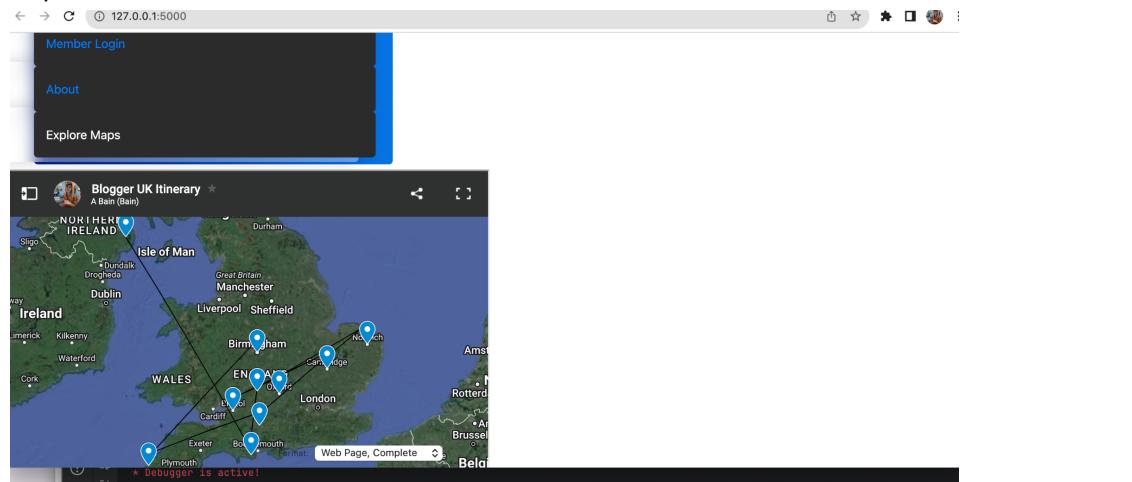
```
135     @app.route('/map')
136     def map():
137         # class attributes:
138         location = "54.4609,3.0886" # coordinates
139         api_key = "google_API"
140
141         map_url = f"https://maps.googleapis.com/maps/api/streetview?size=400x400&location={location}&key={api_key}"
142         return render_template('map.html', map_url=map_url)
143
144 if __name__ == "__main__":
145     app.run(debug=True)
146
if __name__ == "__main__":

```

```
5
6     <title>Starting Location</title>
7
8
9
10
11    <body>
12
13
14        
15
16    </body>
17
18  {% endblock %}
```

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):

← → C ⓘ 127.0.0.1:5000

The screenshot shows a web application interface. At the top, there is a header bar with navigation icons and the URL '127.0.0.1:5000'. Below the header is a dark sidebar containing links: 'Home', 'Search', 'Locations', 'Contact', 'Search', and 'Member Login'. To the right of the sidebar is a light-colored main area. In the main area, there is a map showing the British Isles and parts of Scandinavia. A specific location is highlighted with a blue marker, and the text 'Blogger UK Itinerary' and 'A Bain (Bain)' is displayed above the map. Below the map, there is a dropdown menu labeled 'Format' with the option 'Web Page, Complete' selected.

- 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 *
```

Practicing Flask

```

app.py      location_coordinates.py    map.html    base.html    style.css    requirements.txt    about.html    weather.html
1  {% extends "base.html" %}          ...
2  2  {% block title %}About{% endblock %}
3  3  <body>
4  4  {% block content %}           ...
5  5  <h1>About</h1>
6  6  <p>Alex Bain has crafted together a maps and coordinates feature following renowned Blogger Dannielle Victor on her much- anticipated U
7  7  with weather updates, distances and maps for each picturesque country location. Come join the journey! </p>
8
9
10 </body>
11  {% endblock %}

```

body

Run app

* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 125-127-927
127.0.0.1 - - [13/Nov/2023 09:44:56] "GET / HTTP/1.1" 200 -

Practicing Flask > templates > about.html

Practicing Flask

My Drive – Google Drive | COS60016: Programming for ... | Home Page | About

127.0.0.1:5000/about

Locations

Contact

Search

Member Login

About

About

Alex Bain has crafted together a maps and coordinates feature following renowned Blogger Dannielle Victor on her much- anticipated UK tour, with weather updates, distances and maps for each picturesque country location. Come join the journey!

Blogger UK Itinerary ★
A Bain (Bain)

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

The screenshot shows a code editor interface with a dark theme. At the top, there's a navigation bar with tabs for 'Practicing Flask' and 'main'. Below the tabs, a file tree shows files like 'login.html', 'location_coordinates.py', 'map.html', 'base.html', 'style.css', 'requirements.txt', 'about.html', and 'weather.html'. The main area displays the content of 'login.html':

```

3  {% block title %}Members Login{% endblock %}
4
5  {% block content %}
6
7
8  <form action="#login" method="post">
9    <p>Name:</p>
10   <p><input type="text" name="nm" /></p>
11   <p><input type="submit" value="submit"/></p>
12 </form>
13
14
15  {% endblock %}

```

Below the code editor is a terminal window showing the application's startup logs:

```

* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
  * Debugger is active!
  * Debugger PIN: 125-127-927
127.0.0.1 - - [13/Nov/2023 00:30:21] "GET / HTTP/1.1" 200 -

```

The bottom status bar indicates the current file is 'Practicing Flask > templates > login.html'.

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;

The screenshot shows the same code editor interface with a dark theme. The file tree now includes 'app.py'. The main area displays the content of 'app.py':

```

115
116  @app.route("/login", methods=["POST", "GET"])
117  def login():
118      if request.method == "POST":
119          user = request.form["nm"]
120          return redirect(url_for("user", usr=user))
121      else:
122          return render_template("login.html")
123
124  @app.route("/<usr>")
125  def user(usr):
126      return f"<h1>{usr}</h1>"

```

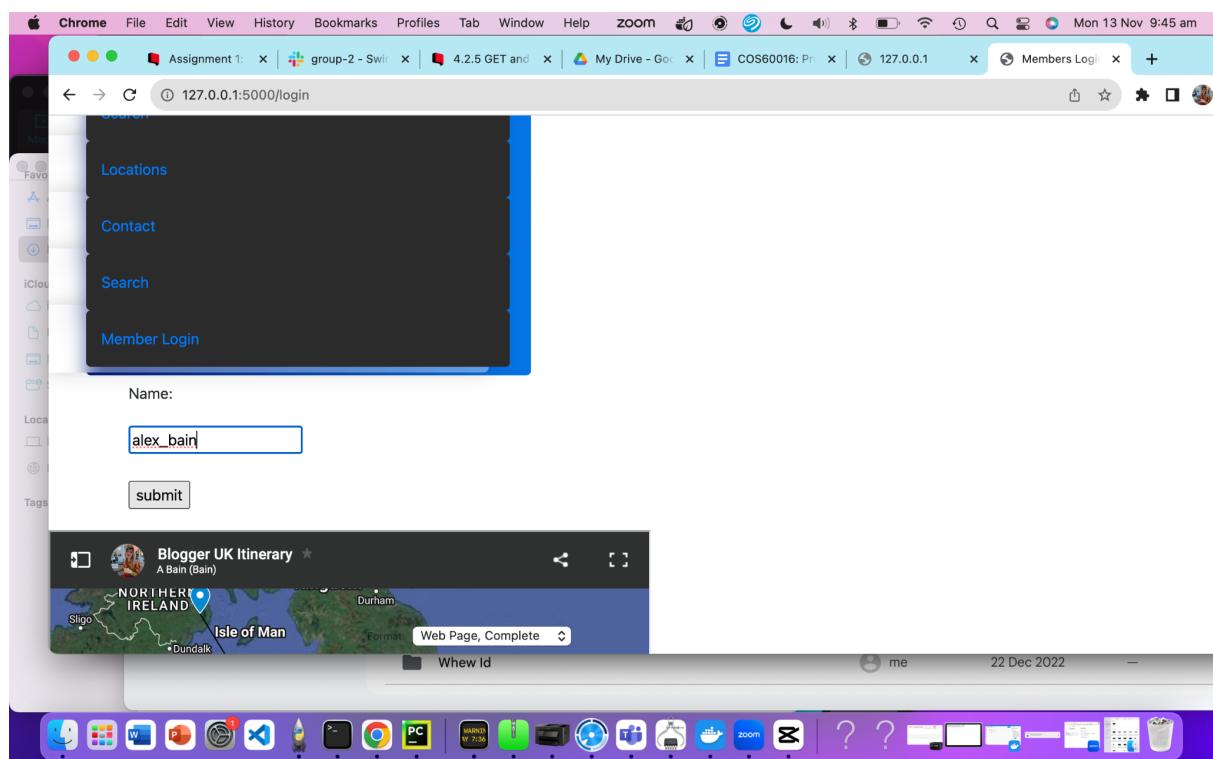
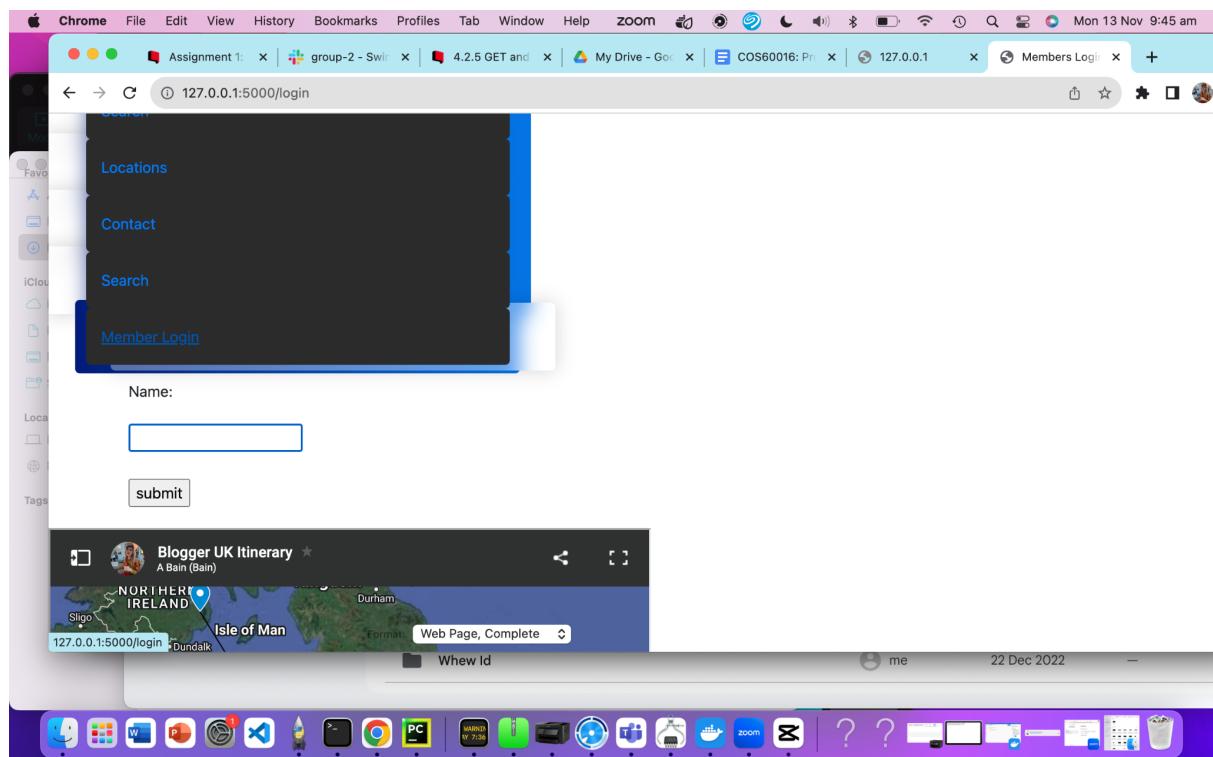
Below the code editor is a terminal window showing the application's startup logs:

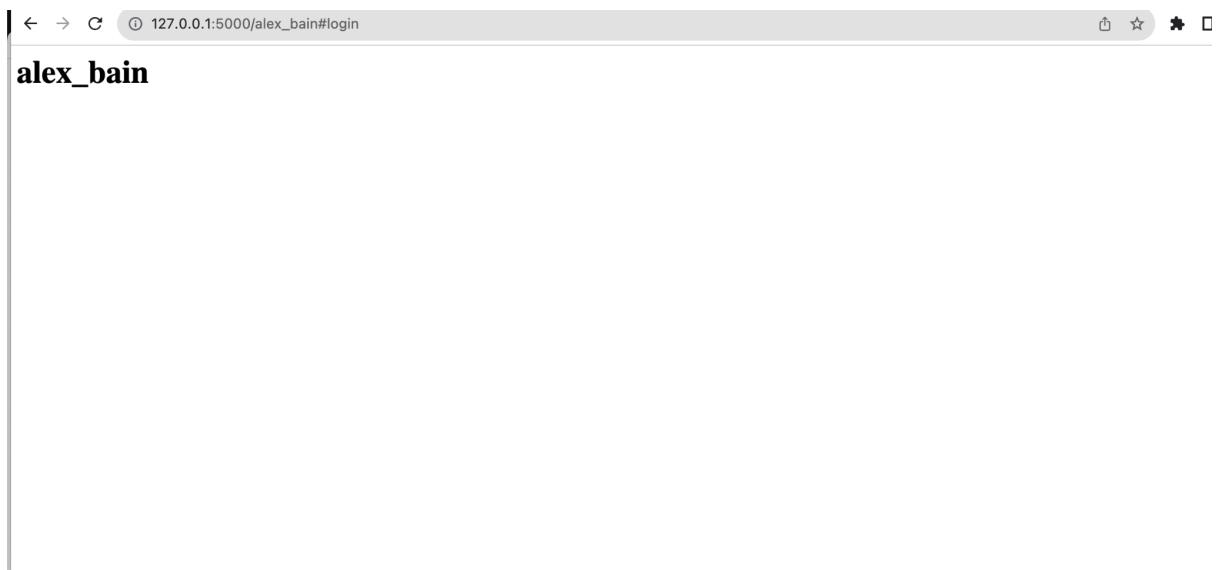
```

* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
  * Debugger is active!
  * Debugger PIN: 125-127-927
127.0.0.1 - - [13/Nov/2023 00:30:21] "GET / HTTP/1.1" 200 -

```

Testing the requests:





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.