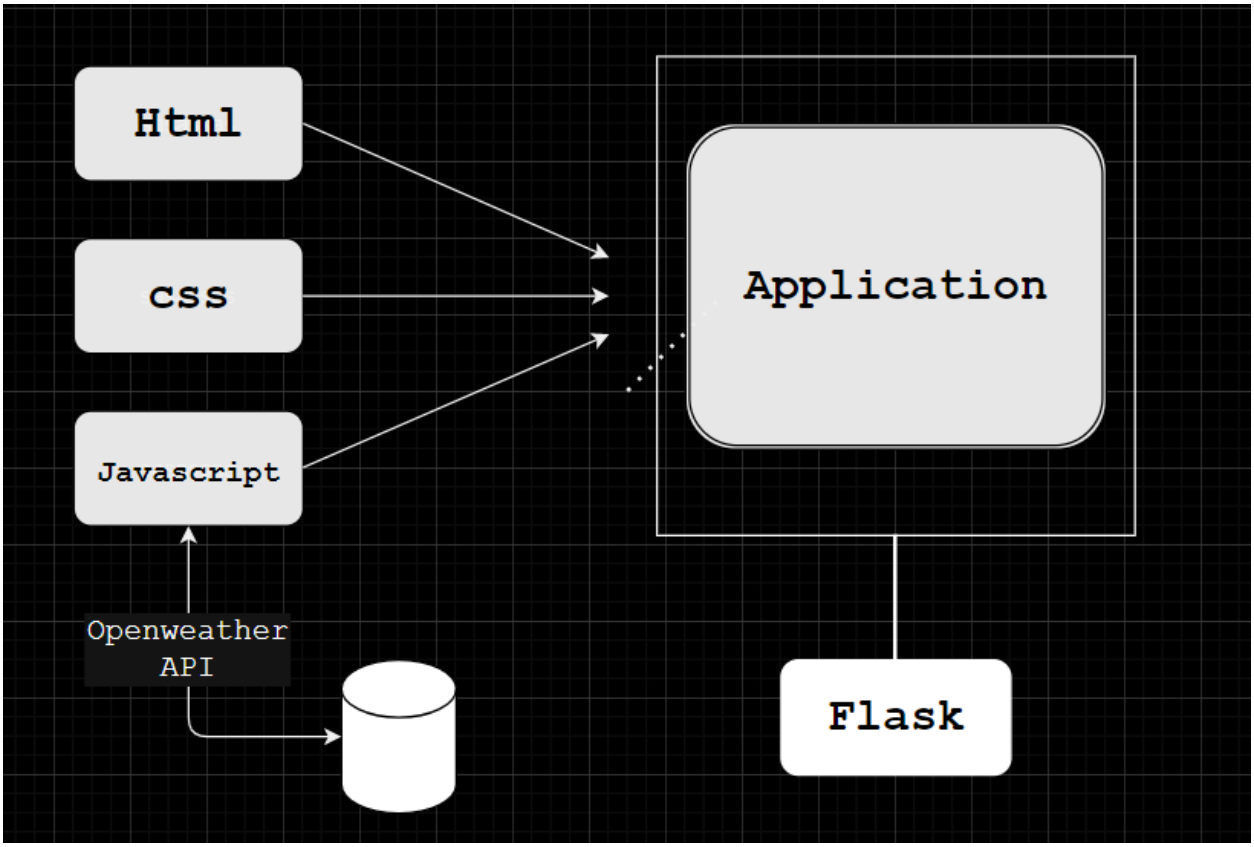
WEATHER.IO:Weather App

Project Description:

The Weather.io is a web application that provides real-time weather information for a specified location. It utilizes the OpenWeatherMap API to fetch weather data and displays it in a user-friendly interface. Users can search for a location by city name and receive detailed weather information, including temperature, humidity, wind speed, and weather conditions.

Technical Architecture:



Pre-Requisites:

To complete this project, you will need:

- A code editor (such as Visual Studio Code, Sublime Text, or Atom)
- A web browser
- An internet connection
- HTML, CSS, and JavaScript knowledge
- Flask
- OpenWeatherMap API key (sign up at https://openweathermap.org/ to obtain one)

Project Objectives:

By the end of this project, you will:

Create a user interface using HTML and CSS to display weather information.

Utilize JavaScript to interact with the OpenWeatherMap API and fetch weather data.

Dynamically update the UI with the fetched weather data.

Allow users to search for weather information by city name or zip code.

Project Flow:

To accomplish the objectives, we will complete the following activities:

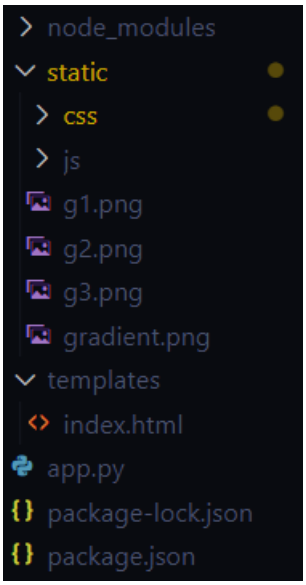
- Set up the project structure
- Design and implement the user interface
- Connect to the OpenWeatherMap API
- Fetch weather data based on user input
- Update the UI with the fetched weather data

Project Structure:

The project structure will include the following files:

(you can just pip install flask.

I have mistakenly installed it through npm, hence the files node_modules,package.json,package-lock.json)



index.html: The main HTML file that contains the structure of the web page.

style.css: The CSS file that defines the styles for the user interface.

script.js: The JavaScript file that handles API calls and updates the UI.

images/ (optional): A folder to store any necessary images for the UI.

Milestone 1: Set up the project structure

Create a new project folder for the Weather App.

Inside the project folder, create the following files/folders:

index.html

style.css

script.js

Next, put css and js files in **static** folder while index.html file in **templates** folder as you have to render it through flask.

Create app.py and write the python code for running your application

Milestone 2: Design and implement the user interface

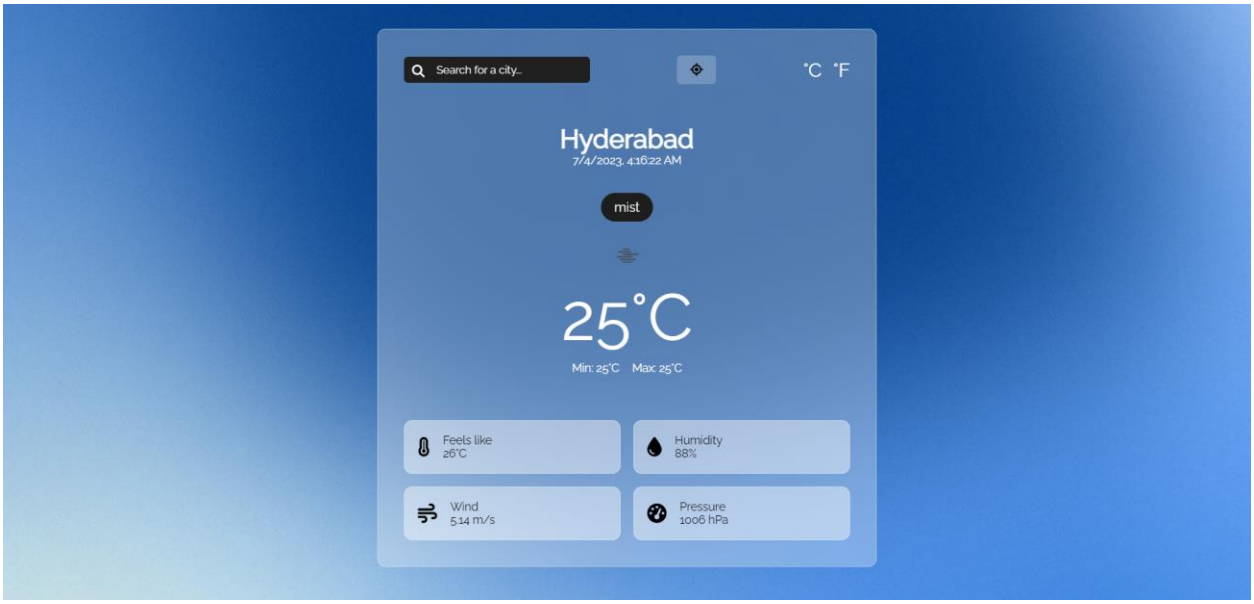
Open index.html in your code editor.

Set up the basic HTML structure.

Design the layout and structure of the user interface using HTML elements and CSS classes.

Apply styles to the UI elements using CSS in style.css.

Link style.css to index.html.



Milestone 3: Connect to the OpenWeatherMap API

In script.js, define a constant variable to store your OpenWeatherMap API key.

Create a function to handle API calls and fetch weather data from the OpenWeatherMap API.

Use the fetch() function or an AJAX library to make a GET request to the OpenWeatherMap API, passing the necessary parameters (e.g. city name).

Handle the API response and extract the relevant weather data.

```
async function fetchWeatherData(city) {
  try {
    const response = await fetch(
      `${baseUrl}?q=${city}&appid=${apiKey}&units=${units}`
    );
    if (!response.ok) {
      throw new Error("Weather data not available.");
    }
    const data = await response.json();
    updateWeatherInfo(data);
  } catch (error) {
    console.log(error);
  }
}
```

```
function updateWeatherInfo(data) {
  cityElement.textContent = data.name;
  datetimeElement.textContent = getCurrentTime();
  forecastElement.textContent = data.weather[0].description;
  iconElement.innerHTML = `<img src="http://openweathermap.org/img/wn/${data.weather[0].icon}.png" alt="Weather Icon">`;
  temperatureElement.innerHTML = `${Math.round(data.main.temp)}&#176;${
    units === "metric" ? "C" : "F"
  }`;
  minMaxElement.innerHTML = `<p>Min: ${Math.round(data.main.temp_min)}&#176;${
    units === "metric" ? "C" : "F"
  }</p><p>Max: ${Math.round(data.main.temp_max)}&#176;${
    units === "metric" ? "C" : "F"
  }</p>`;
  realFeelElement.innerHTML = `<p>${Math.round(data.main.feels_like)}&#176;${
    units === "metric" ? "C" : "F"
  }</p>`;
  humidityElement.textContent = `${data.main.humidity}%`;
  windElement.textContent = `${data.wind.speed} ${
    units === "imperial" ? "mph" : "m/s"
  }`;
  pressureElement.textContent = `${data.main.pressure} hPa`;
}
```

Milestone 4: Fetch weather data based on user input

Add an input field and a button to the UI to allow users to enter a city name or zip code.

Add an event listener to the button to trigger the weather data fetch function when clicked.

Retrieve the user input from the input field.

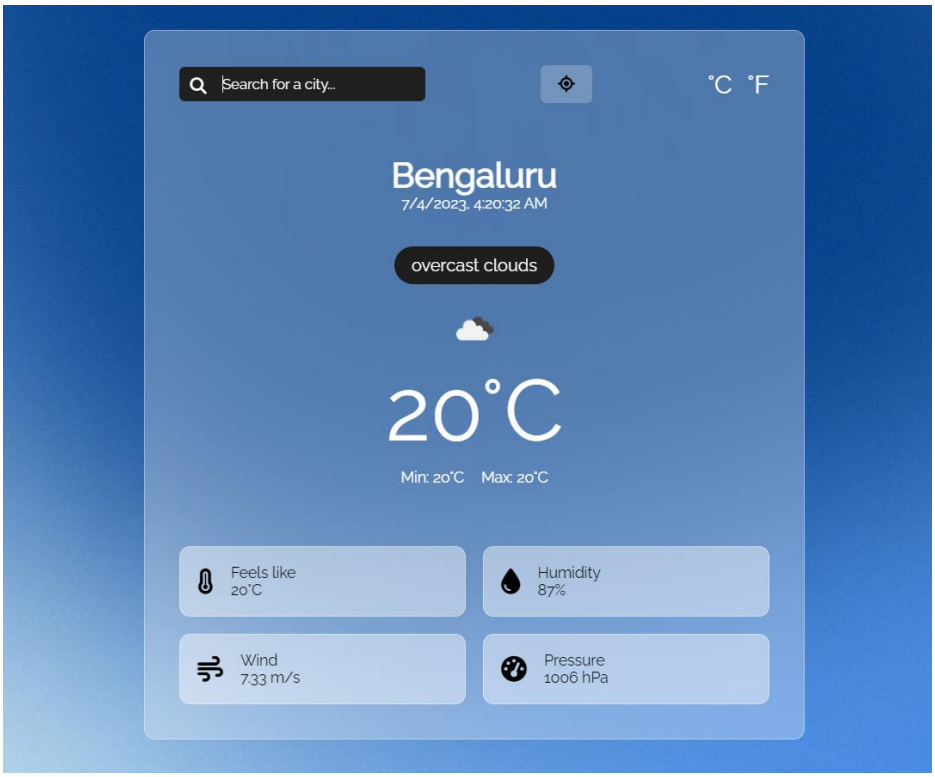
Call the API function, passing the user input as a parameter.

```
searchForm.addEventListener("submit", (e) => {
  e.preventDefault();
  const city = searchInput.value.trim();
  if (city !== "") {
    fetchWeatherData(city);
  }
  searchInput.value = "";
});
```

Milestone 5: Update the UI with the fetched weather data

Create functions to update the UI with the fetched weather data.

Select the necessary UI elements using JavaScript DOM manipulation methods.Modify the UI elements' content or styles to display the weather information dynamically.



As you can see there is button In between search and degrees,which is a button when clicked tells you the weather of your current location this uses geolocation property in js.

Milestone 6: Run it using Flask

Using the following code you can run your application using flask

```python
from flask import Flask, render_template

app = Flask(__name__)

@app.route("/")
def weather():
    return render_template("index.html")

if __name__ == "__main__":
    app.run()
```

Conclusion:

The Weather App is a web application that provides real-time weather information to users. By integrating the OpenWeatherMap API and implementing an intuitive user interface, users can easily retrieve weather data for a specific location. The project's modular structure allows for easy maintenance and further enhancements, such as adding additional features or optimizing the UI.