

Weather Dashboard with Chatbot Integration using OpenWeather API

Objective:

Create a fully responsive weather dashboard using **HTML**, **CSS**, and **Javascript**.

1. For weather information, you will use the **OpenWeather API** (free tier)
2. For the chatbot API, you will use the **Gemini API**
3. For data visualization you will use **Chart.js**

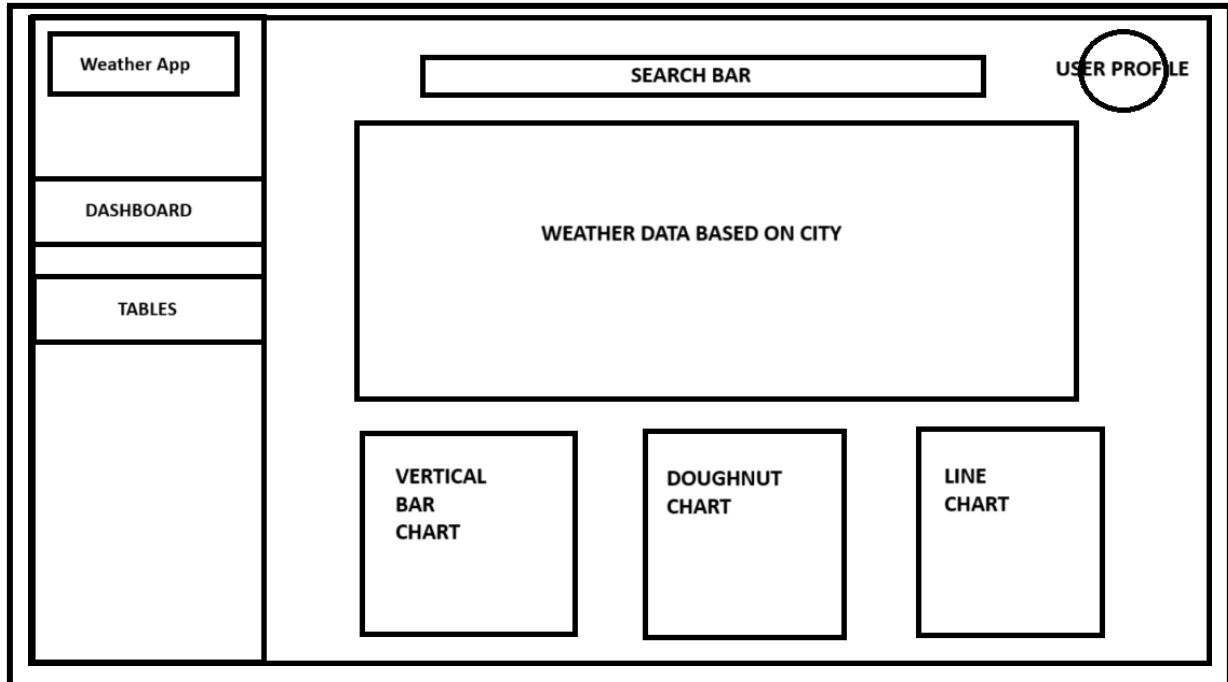
Your dashboard will consist of:

- A side menu (which has the logo of the weather app at the top, an option to go to the dashboard page and an option to go to the tables page)
- A widget for weather details based on the city selected by the user.
 - The background of the widget should change based on the weather conditions of the city chosen (e.g; if the user searched for "London" and the weather is "overcast clouds" then the background of the widget should be cloudy.) You can consult this [link](#) for weather conditions shown by the OpenWeather API.
 - There should be **three** types of charts shown under the weather widget using Chart.js:
 - 1. **Vertical Bar Chart** showing temperatures for the next 5 days in the city.
 - 2. **Doughnut Chart** showing the percentage of different weather conditions over the 5 day period (e.g; if there are 4 sunny days and 1 cloudy day, then the doughnut chart will depict almost the whole pie as sunny and a small slice as cloudy).
 - 3. **Line Chart** showing temperature changes for the next 5 days.
- To your charts you must add the following **animations (using Chart.js)**:
 - **Delay** (for vertical bar chart and doughnut chart)
 - **Drop** (for line chart)
- A **table** with temperature forecast for the next 5 days.
 - The first **10 entries** should be shown, after that you should implement **pagination**.

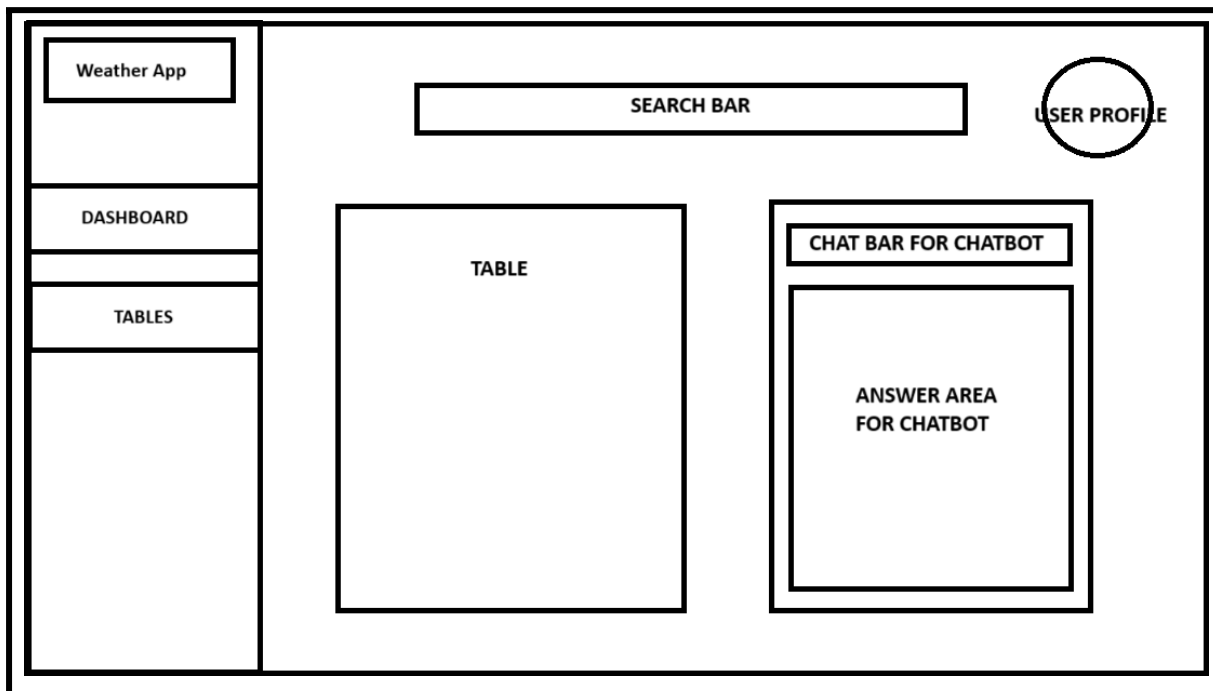
Integrate the Gemini Chatbot API:

1. **Register** and Obtain Gemini API Key from here (<https://ai.google.dev/aistudio>)
2. **Prepare and integrate** the weather API into your chatbot flow; Identify when the user is asking for weather (e.g., by detecting the word "weather" in the query). Fetch weather data using the weather API and store the result.
3. **Use Gemini API** to handle non-weather-related queries.
4. **Test the chatbot** by asking both general questions and weather-related questions.

A sample layout for the dashboard page has been shown below:



A sample tables page has been shown below:



Tools Required:

- HTML, CSS

- JavaScript (vanilla or jQuery)
 - OpenWeather API (Free plan)
 - Chart.js
-

Key Learning Outcomes:

1. Practice making API requests using JQuery/Ajax/Fetch API.
 2. Learn to manipulate API responses and display data dynamically on a webpage.
 3. Enhance user experience through responsive design and user-friendly interfaces.
 4. Handle errors such as invalid city input, API errors or where required.
-

Assignment Instructions:

Step 1: Setting Up OpenWeather API

1. Create an API Key:

- Sign up to OpenWeather API by going to [this link](#).
- Confirm your email address.
- Go to your dashboard, under API Keys you will find your “Default” API key. You can use this or generate a new one by giving the key a name and clicking on “Generate”.
- **Note:** The free plan has certain [restrictions](#), amongst this is the fact that you can only do **60 calls/minute**. Please keep this in mind when working with this API.

2. API Documentation Overview:

- The API calls you will need for this assignment are:
 - [Current weather](#)
 1. The format for this API call is:
`https://api.openweathermap.org/data/2.5/weather?lat={lat}&lon={lon}&appid={API key}`
 2. You can also use this to specify the city name:
`https://api.openweathermap.org/data/2.5/weather?q={city name&appid={API key}`
 - [5-day weather forecast](#)
 1. The format for this API call is
`api.openweathermap.org/data/2.5/forecast?lat={lat}&lon={lon}&appid={API key}`
 - In the APIs you must replace {API key} with your own API that you generated in step 1 (or use the default one).
 - Some parameters you will need are:
 1. lat - which is for latitude of the city you want to get information from

2. lon - which is for the longitude of the city you want to get information from
 3. units - which are in the formats standard, metric and imperial
- By default the API returns data in JSON format. It returns several fields that you can use. For example, when searching for London's current weather, the API returns this:

```
https://api.openweathermap.org/data/2.5/weather?
q=London&appid={API key}
```

```
{
  "coord": {
    "lon": -0.13,
    "lat": 51.51
  },
  "weather": [
    {
      "id": 300,
      "main": "Drizzle",
      "description": "light intensity drizzle",
      "icon": "09d"
    }
  ],
  "base": "stations",
  "main": {
    "temp": 280.32,
    "pressure": 1012,
    "humidity": 81,
    "temp_min": 279.15,
    "temp_max": 281.15
  },
  "visibility": 10000,
  "wind": {
    "speed": 4.1,
    "deg": 80
  },
  "clouds": {
    "all": 90
  },
  "dt": 1485789600,
  "sys": {
    "type": 1,
    "id": 5091,
    "message": 0.0103,
    "country": "GB",
    "sunrise": 1485762037,
    "sunset": 1485794875
  },
  "id": 2643743,
  "name": "London",
  "cod": 200
}
```

Step 2: Frontend Layout (HTML & CSS)

1. Create the HTML Structure:

- Design a basic HTML structure with the following elements:
 - An input box for users to enter a city name.
 - A button labeled “Get Weather” to fetch the data.
 - Sections to display:
 - Current weather information (city name, temperature, humidity, wind speed, weather description, and icon).
 - 5-day weather forecast (date, temperature, and weather conditions).

2. Apply CSS Styling:

- Use CSS to style the layout, ensuring the design is responsive (e.g., using flexbox or grid).
 - Add a background image or gradient related to the weather theme for a more polished look.
-

Step 3: Integrating the API

1. Fetch Current Weather Data (Main Task):

- Use any method (Ajax/JQuery/Fetch) to request the OpenWeather API when the user submits a city name.
- Extract data from the API response such as
 - City name
 - Temperature (allow the user to choose between Celsius and Fahrenheit)
 - Humidity
 - Wind speed
 - Weather description
 - Weather icon
- Display the weather data dynamically on the webpage.

2. 5-Day Weather Forecast (Secondary Task):

- Make an additional API request to the /forecast endpoint to retrieve a 5-day forecast for the city.
- Display the forecast data (e.g., daily temperature, weather condition) for each day in a grid format.

3. Error Handling:

- Implement error handling for invalid city names or API request failures. Display user-friendly error messages like “City not found” or “API limit reached”. Use error handling where necessary.

4. Optimize API Calls:

- Ensure that API calls are not repeated unnecessarily (e.g., avoid making multiple requests when input is unchanged).

Step 4: Filters:

1. **Show temperatures in ascending order:** Use the `sort()` method to arrange the temperatures from lowest to highest.
 2. **Filter out days without rain:** Use the `filter()` method to show only the entries where the weather condition includes rain.
 3. **Show the day with the highest temperature:** Use the `reduce()` method to find the entry with the highest temperature.
 4. **Show temperatures in descending order:** Use the `sort()` method to arrange the temperatures from highest to lowest.
 5. **Show temperatures in ascending order:** Use the `sort()` method to arrange the temperatures from lowest to highest.
-

Step 5: Additional Features (Optional)

- **Unit Conversion Toggle:**
 - Allow users to toggle between Celsius and Fahrenheit for temperature display.
 - **Geolocation Support:**
 - Use the browser's geolocation API to detect the user's location and show weather information for that location by default.
 - **Loading Spinner:**
 - Implement a loading spinner or progress bar while waiting for the API response.
 - **CSS Animations:**
 - Add simple animations to make the interface more interactive (e.g., weather icons fading in).
-

Submission Requirements:

1. A fully functioning weather dashboard with:
 - Current weather information
 - 5-day forecast
 - Proper error handling for invalid city names and API issues
 2. A zip file containing:
 - HTML, CSS, and JavaScript files
 - A README file explaining the project and instructions to run it locally
 3. Deploy the project using GitHub Pages (or any other preferred platform) and provide the live URL.
-

Evaluation Criteria:

- 1. API Integration (40%):**
 - Successful API requests and correct display of weather data.
- 2. Frontend Development (30%):**
 - Layout design, responsiveness, and usability.
- 3. JavaScript/jQuery Implementation (20%):**
 - Efficient use of AJAX, error handling, and dynamic DOM manipulation.
- 4. Code Quality & Documentation (10%):**
 - Clean, well-documented code with a clear README file.