Student name : Pauline ZHOU

Student ID : 202400396

Repport : Final Project - Weather APP

The Weather Application is an advanced web-based platform designed to deliver real-time weather updates, air quality information, and extended forecasts to users worldwide. It leverages the robust capabilities of the OpenWeather API to provide accurate and comprehensive data through an intuitive and visually engaging interface. Built with a focus on responsive design, the application ensures seamless usability across a variety of devices.

I. Tools Used

Frontend

1. **HTML5**

- Structured using semantic elements to enhance accessibility, SEO optimization, and maintainable code.

2. **CSS3**

- Theming: Utilizes CSS variables for consistent and efficient styling across the application.
- Responsive Layouts: Implements CSS Grid and Flexbox for adaptive designs that adjust to different screen sizes using media queries.
- Animations: Smooth transitions and hover effects improve the overall user experience.

3. **JavaScript**

- Asynchronous Programming: Uses async/await to manage API calls effectively.
- Dynamic Content Updates: Handles real-time updates based on user interactions through DOM manipulation.

Libraries and Frameworks

- **Chart.js**: A powerful JavaScript library for creating interactive visualizations, such as temperature, humidity, and air quality index charts, to provide users with engaging and informative experience.

APIs

- OpenWeather API

- Current Weather Data: Offers real-time information, including temperature, humidity, wind speed, and sunrise/sunset times.
- Air Quality Index (AQI): Provides pollutant levels (e.g., PM2.5, NO2, CO) with easy-to-understand visual indicators.
- Forecasting: Supplies detailed five-day weather predictions.

Development Tools

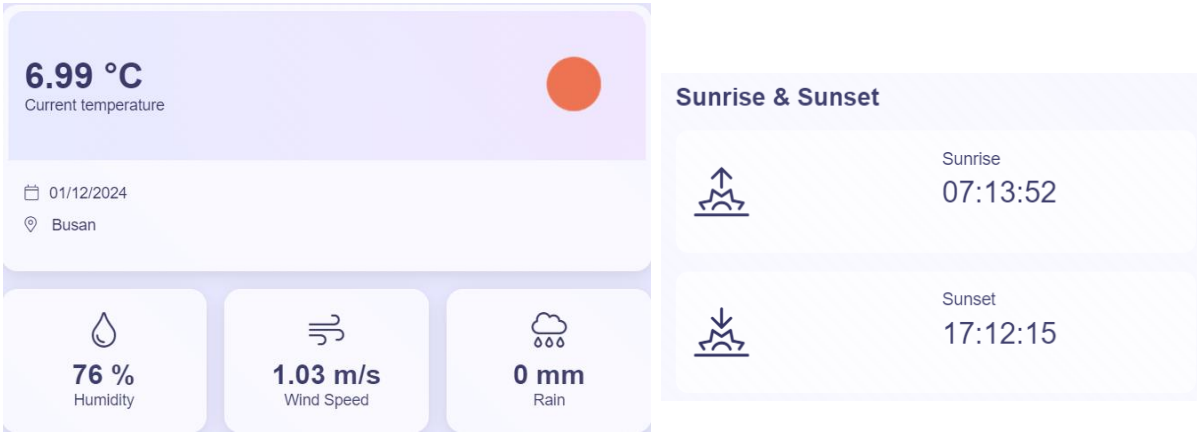
- **Visual Studio Code**: Used for code development, offering robust features such as syntax highlighting, debugging, and version control integration.

- **Git & GitHub**: Enable version control, collaborative development, and centralized repository management.

II. Features Done

1. Real-Time Weather Data

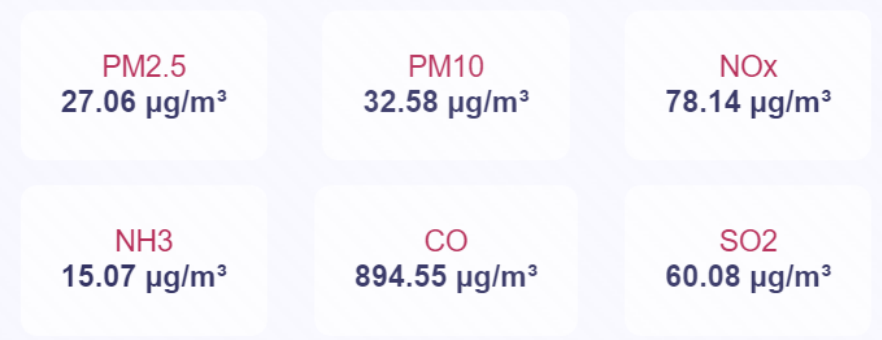
- Displays current conditions, including temperature, humidity, wind speed, and additional details such as sunrise and sunset times.



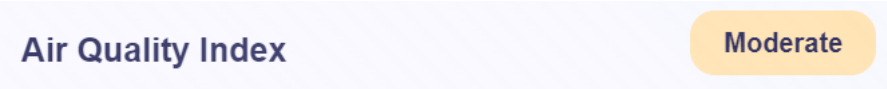
- User can enter the city's name that he wants to check the weather (/!\ only city)

2. **Air Quality Monitoring**

Real-time AQI data with pollutant breakdowns (PM2.5, PM10, NO2, etc.).



- Color-coded AQI indicators for easy interpretation.

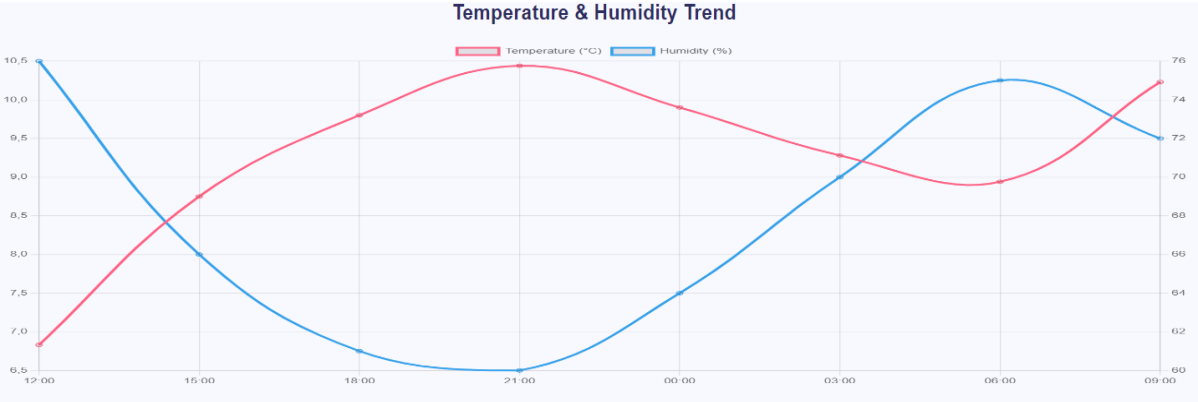


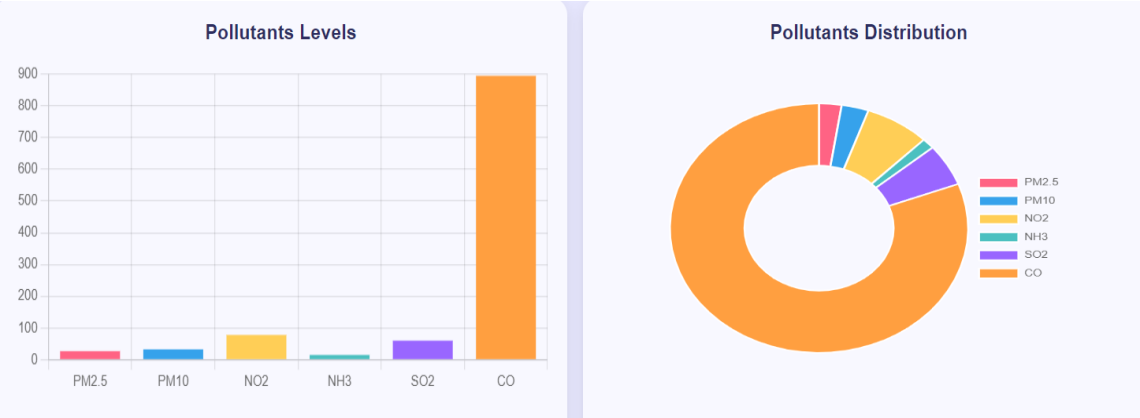
3. **Five-Day Forecast**

Detailed weather forecasts for the upcoming five days, with daily temperature highs/lows and descriptive weather icons.

4. **Data Visualization**

Interactive charts created with Chart.js, a line chart showing the temperature and humidity trend, a bar chart showing pollutants levels and a doughnut chart showing pollutants distribution



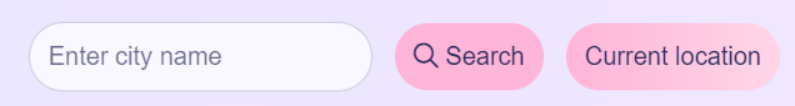


5. **Displaying the Evolution of Weather Data for the Next 5 Days**

I created graphs illustrating the evolution of temperature, humidity, wind speed, and air quality over the upcoming five days. These visualizations provide users with an intuitive understanding of how these metrics are expected to change during this period.

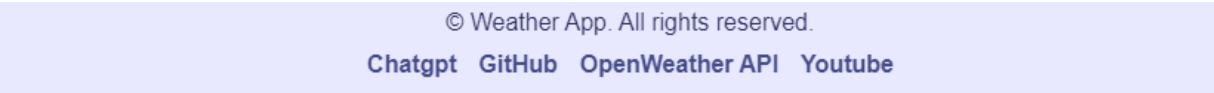
7. **User Geolocation**

Fetches weather data based on the user's geographical location using the Geolocation API and updates all displayed data automatically.



8. **Footer**

I added a footer which includes hyperlinks to the various resources I utilized for the project. These links provide users with direct access to documentation.



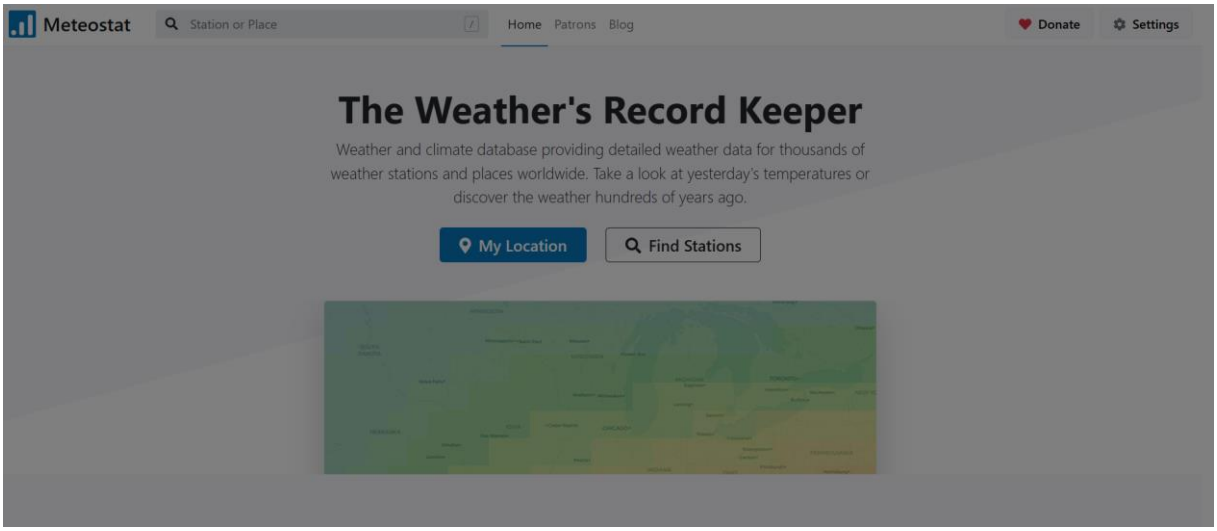
III. Features fails

Planned Long-Term Data Visualizations

Originally, the project aimed to include graphs representing data trends over various time periods: one week, one month, six months, and one year.

However, due to limitations in the OpenWeather API, which primarily provides real-time and short-term forecast data, implementing this feature was not feasible.

To overcome this limitation, I turned to MeteoStat (https://meteostat.net/en/) where I successfully obtained a new API key. Unfortunately, after some testing, I realized that their service was unreliable at the time, as the website often malfunctioned and failed to deliver the required data consistently.



As a last resort, I tried using WeatherAPI, but despite multiple attempts, I couldn't retrieve the necessary data. The API consistently returned error messages, making it impossible to integrate the historical data feature into the project.

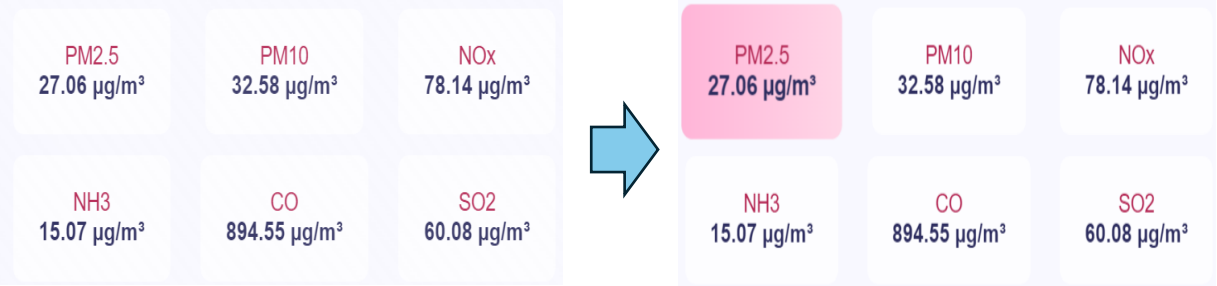
IV. Implementation Details

CSS Styling

- **Variables**: Centralized in the :root selector for simplified management of themes and styles.

- **Responsive Layouts**: Media queries adjust component layouts, ensuring usability across all screen sizes.

- **Transitions and Animations**: Subtle animations enhance interaction feedback, such as hover effects for buttons and cards.



JavaScript Functionality

1. **Data Fetching**

- Utilizes asynchronous functions to interact with the OpenWeather API.

- Robust error handling ensures graceful degradation and user feedback during API failures or invalid input.

2. **Dynamic Updates**

- Real-time updates reflect user input or geolocation changes.

- Efficient handling of API responses ensures a seamless experience.

3. **Chart Integration**

- Chart.js is used to create dynamic visualizations.

- Charts are updated or recreated dynamically to reflect the latest data.

API Integration

API endpoints are configured to fetch:

- Current weather conditions by city name.

- AQI data using geographic coordinates.

- Five-day weather forecasts for the specified location.

V. Challenges and Solutions

1. **API Rate Limits**

- Addressed by minimizing redundant API calls and implementing caching mechanisms.

2**. Graph Updating Issues**

Initially, the graphs did not update properly when displaying new data. Indeed, the issue was traced to the reuse of the same <canvas> elements without proper cleanup.

After that, the solution involved destroying the existing **Chart.js** instances before rendering new charts. This approach ensures that the <canvas> elements are reset, allowing the new graphs to display correctly.

3. **User Location Retrieval**

First, I had no idea how to retrieve the user's current location to provide localized weather data. After thorough research, I discovered Geolocation **API**, which provides the user's geographical coordinates with their permission.

Integrating this API into the application required handling user consent, error scenarios (e.g., when location access is denied), and fallback mechanisms for manual location input.

Successfully implementing this feature added significant value by enabling personalized weather data based on the user's real-time location.

Demo link : https://zpau7.github.io/WebSite_Project/