CSU44000 Internet Applications
Assignment 1: Reactive Internet Application

Eliel Oshiokameh, 21367772

October 20, 2024

Introduction

This assignment tasked us with the development of a reactive internet application that leverages both client-side and server-side technologies to provide a weather forecasting service. Utilising Vue.js on the client side and Node.js with Express on the server side, the application interacts with the OpenWeatherMap API to fetch weather and pollution data for a specified city. The core features include a 5-day weather forecast, pollution levels, and packing advice based on expected weather conditions. Additionally, an innovative feature was incorporated: an hourly forecast that provides a more granular view of the weather, supported by dynamic charts.

This report elaborates on the design process, the technologies employed, and the overall implementation of the application. It will also outline the challenges encountered and how they were overcome, offering insight into both the technical and creative aspects of the project.

Design Process

The design process began with a thorough breakdown of the assignment requirements into discrete, manageable tasks. The application was divided into two core components: the **client-side** interface and the **server-side** API handling. A number of key design decisions were made to ensure the application was reactive, intuitive, and fulfilled the given requirements.

Client-Side Design

Vue.js was chosen as the client-side framework due to its simplicity, flexibility, and ability to facilitate reactive behaviour. The following design features were implemented on the client side:

- **Reactive Interface**: The application leverages Vue's reactivity system to dynamically update the interface without needing a page refresh. Users can input a city name and see immediate feedback in the form of weather data and pollution information. The use of Vue's `v-model` directive enables two-way data binding between the input field and the application state, providing a seamless user experience.
- **City Suggestions**: To enhance usability, an auto-suggestion feature was implemented using the OpenWeatherMap geocoding API. As the user types in a city name, suggestions are displayed in a dropdown, allowing users to quickly select their desired location. This was achieved through the use of Vue's `v-for` directive to dynamically generate suggestion elements.

- **Weather and Pollution Display**: Once the user selects a city, the application displays the weather forecast for the next five days, including temperature, wind speed, and rainfall levels. A summary packing recommendation is also provided, which is calculated based on the forecasted weather. In the case of elevated pollution levels, the application displays a warning along with a chart that visualises pollutant levels.
- **Hourly Forecast**: The innovative feature of the application is the inclusion of an hourly forecast. Users can click on any day in the 5-day forecast to view a detailed breakdown of hourly weather data, including temperature, rainfall, and wind speed, all presented in both tabular and graphical formats.

Server-Side Design

The server-side component was developed using **Node.js** with **Express**. Its main function is to serve as a middleman between the client and the OpenWeatherMap API, ensuring that the client only needs to make requests to a single origin (the server), thus bypassing **CORS** (Cross-Origin Resource Sharing) issues that arise when making direct API requests from the client.

Key design features of the server include:

- **API Fetching**: The server fetches weather data from the OpenWeatherMap API using Axios, which allows for smooth, asynchronous requests. The server processes the weather data and structures it in a way that is easy for the client to consume, minimising the amount of data processing that needs to occur on the client side.
- **Pollution Data Retrieval**: After obtaining the weather forecast, the server uses the city's latitude and longitude to fetch air pollution data from the OpenWeatherMap's Air Pollution API. The server processes this data and identifies any pollutants that exceed recommended levels, passing this information back to the client for display.
- **Data Structuring**: To reduce the complexity on the client side, the server processes the raw data returned by the OpenWeatherMap API into more user-friendly formats. For example, the server aggregates weather data into daily summaries that include the minimum and maximum temperatures, total rainfall, and wind speeds for each day.
- **Hourly Forecast Endpoint**: A second API endpoint was added to provide hourly weather data for a selected day. This was crucial for the innovative feature, allowing the client to fetch detailed hourly forecasts on demand.

Innovative Feature

The application's innovative feature is its detailed **hourly forecast**. Clicking on a day in the 5-day forecast opens a modal that shows the weather data for each hour of the selected day. The modal contains both a table of data and a dynamically generated chart that visualises temperature, wind speed, and rainfall over the course of the day. This feature enhances the application's utility by providing users with more detailed and actionable information.

Technologies Used

Vue.js

Vue.js was the natural choice for the client-side framework due to its simplicity and flexibility. The reactive nature of Vue.js made it easy to dynamically update the user interface in response to server-side data changes, improving user experience. Key features used in Vue.js include:

- **Directives**: Vue's directives such as `v-model`, `v-for`, and `v-if` allowed for dynamic rendering and easy two-way data binding between form inputs and application state.
- **Computed Properties and Methods**: Vue's computed properties were used to calculate packing recommendations based on the forecasted weather conditions, while methods were employed to handle API requests and user interactions.

Axios

Axios was used for handling asynchronous HTTP requests from both the client and server. Its simplicity and support for Promises made it ideal for interacting with the OpenWeatherMap API. Axios was utilised on the server to fetch weather and pollution data, and on the client to make requests to the server.

Node.js and Express

On the server side, **Node.js** was selected for its asynchronous, non-blocking nature, making it ideal for handling API requests. **Express** was used to create a lightweight and efficient server that exposed two key API endpoints: one for the weather forecast and pollution data, and another for the hourly forecast.

- **Environment Variables**: The API key for OpenWeatherMap was stored securely using environment variables with the help of the `dotenv` package. This ensures that sensitive information is not hard-coded into the source code.

OpenWeatherMap API

The **OpenWeatherMap API** was integral to the project. Two endpoints were used: one for retrieving the 5-day weather forecast and another for getting air pollution data. The API returned detailed information such as temperature, wind speed, rainfall levels, and pollutant concentrations.

- **GeoLocation API**: This API was also used to fetch city suggestions based on user input, enhancing the user experience by making it easier to select cities.

HTML and CSS

The user interface was built with **HTML** and styled with **CSS**. Special attention was given to the visual design of the application, ensuring that it was both functional and aesthetically pleasing.

- **Background Image**: The inclusion of the `sunset.jpg` background image provides a visually appealing backdrop that enhances the overall user experience.
- **Responsive Design**: The application was designed to be responsive, ensuring that it works well on devices with varying screen sizes. This was particularly important for the modal, which adjusts to fit different screen resolutions.

Implementation

Client-Side (Vue.js)

The client-side application is contained within a single Vue instance. It manages user input, fetches weather data from the server, and updates the interface dynamically. Here are the key implementation details:

- **City Input**: Users can input a city name, which triggers a request to the server to fetch the corresponding weather data. If an error occurs (e.g., the city is not found), an error message is displayed.
- **Weather Forecast**: Upon receiving a successful response, the client displays a 5-day weather forecast in a table format, alongside packing advice based on the forecasted conditions.
- **Pollution Chart**: If the server detects elevated pollution levels, a bar chart is rendered using **Chart.js**, showing the concentration of various pollutants.
- **Hourly Forecast Modal**: When a user clicks on a day in the forecast, a modal appears showing the hourly weather data for that day, both in table and chart format. This feature is powered by a separate API request to the server for hourly data.

Server-Side (Node.js and Express)

The server-side logic handles all the heavy lifting of interacting with the OpenWeatherMap API. It processes the raw data into a more user-friendly format and handles errors gracefully, returning appropriate messages to the client in case of issues with API requests.

- **Weather Data Handling**: The server fetches the 5-day weather forecast, processes it, and sends it back to the client. This involves aggregating weather data into daily summaries and generating packing advice based on forecasted conditions.
- **Air Pollution Data**: The server fetches pollution data using the city's geographic coordinates and processes this to determine if any pollutant exceeds safe levels. If so, a warning is generated and returned to the client.
- **Hourly Forecast**: The server exposes a separate API endpoint for fetching hourly forecast data, which the client uses to populate the modal for hourly forecasts.

Conclusion

In conclusion, this project successfully met the requirements outlined in the assignment brief. By using Vue.js on the client side and Node.js with Express on the server side, we created a fully reactive application that provides users with detailed weather and pollution data. The innovative hourly forecast feature enhances the application by providing more granular insights into the weather, offering users both tabular and graphical representations of the data. This project demonstrated a solid understanding of both client-side and server-side web development, as well as the ability to integrate third-party APIs into a seamless user experience.