CPSC 332 Web Development

Gonzaga University

Daniel Olivares

Homework 8 – Node.js – Weather Dashboard using open APIs

Individual, non-collaborative assignment

Learner Objectives

By completing this assignment, learners will:

1. Integrate third-party APIs for geolocation and weather data retrieval.
2. Configure and manage environment variables securely using dotenv.
3. Build a Node.js and Express server with route handling.
4. Develop a dynamic frontend that changes themes and displays custom icons based on weather conditions.
5. Handle and display error messages.

Prerequisites

Before starting this assignment, learners should:

1. Be familiar with JavaScript, HTML, and CSS basics.
2. Understand Node.js and Express framework fundamentals.
3. Know how to install and use npm packages.
4. Be able to set up and retrieve environment variables using dotenv.

Overview and Requirements

In this assignment, you'll create a weather dashboard where users enter a city and US state to retrieve current weather information. The project will involve setting up an Express server, using the OpenWeatherMap API to fetch geolocation and weather data, dynamically changing the frontend theme, and displaying custom SVG icons based on the weather condition. All sensitive information, including the API key, must be stored securely in a `.env` file using dotenv.

To complete this assignment, you must:

1. Use OpenWeatherMap's geocoding API to get latitude and longitude for a specified city and state.
2. Use OpenWeatherMap's weather API to fetch current weather data based on the retrieved coordinates.
3. Configure the project to use dotenv for storing and retrieving the API key.
4. Implement a frontend that:

- Accepts city and state inputs.
- Displays the temperature, weather description, and wind speed.
- Dynamically changes the theme based on the weather condition.
- Displays a custom SVG icon below all text elements.

5. Handle errors gracefully and provide user-friendly messages.

Task Instructions

1. **Setup and Initialization**
   - Initialize a new Node.js project and install the required packages: `express` and `dotenv`.
   - *(possibly not required)* Install `node-fetch`:
     - If you are using Node.js version **>=18**, `fetch` is natively available. However, if using an older version of Node.js, install `node-fetch` by running:

```
npm install node-fetch
```

   - Create a `.env` file in the root directory to store your API key and port configuration. Use the following format:

```
OPENWEATHER_API_KEY=your_openweathermap_api_key
PORT=3001
```

   - Refer to dotenv documentation for setting up and managing environment variables in Node.js.
   - Set up a `.gitignore` using the Node.gitignore template to prevent the *node_modules* directory from being added to your repository and sensitive data from being released with your source code.
     - *What line in this file prevents your API key from being included in your repository?*

2. **Obtain an OpenWeatherMap API Key**
   - Go to OpenWeatherMap and create a free account.
   - Once registered, navigate to the **API keys** section to obtain your API key.
   - Paste the API key into your `.env` file as shown above.

3. **Geocoding and Weather Data Integration (*server.js*)**
   - ***Hint:*** *A template string can be used for the API calls to make URL modification easier.*

- **Geocoding API**: Use OpenWeatherMap's geocoding API to retrieve the latitude and longitude based on the city and state provided by the user.
  - **Example API Call**:

```
https://api.openweathermap.org/geo/1.0/direct?q=San+Francisco,CA,US&limit=1&appid=YOUR_API_KEY
```

- **Weather Data API**: Use the latitude and longitude coordinates from the geocoding API to fetch the current weather data.
  - **Example API Call**:

```
https://api.openweathermap.org/data/2.5/weather?lat=37.7749&lon=-122.4194&units=imperial&appid=YOUR_API_KEY
```

- Ensure the weather data is in Fahrenheit by adding `&units=imperial` to the request.

4. **Backend - Express API Route (*server.js*)**

- Create a route in Express that accepts GET requests to fetch weather data. The route should take `city` and `state` as query parameters (e.g., `/weather?city=San+Francisco&state=CA`).
- Use the geocoding API to retrieve latitude and longitude for the specified city and state, then use these coordinates to get weather data.
- Return data in JSON format that includes temperature, weather description, wind speed, and condition.
- Refer to zyBook sections **11.3 Express, 11.4 Express request data,** and **11.12 RESTful web APIs** to structure and handle incoming requests.

5. **Frontend - Dynamic Theme and Icon Display (*index.html, script.js*)**

- Build a basic HTML form where users can enter a city and state.
- Refer to zyBook **3.2 Forms** to set up a form layout.
- On form submission, make a call to your **local** Express API (i.e., a get request!) to retrieve weather data.
- Refer to zyBook **8.15 Fetch API** and **8.14 async and await** for guidance on using Fetch and asynchronous code.
- Display the following on the page:
  - Temperature (in Fahrenheit)
  - Weather description (e.g., sunny, cloudy)
  - Wind speed
- **Dynamic Theme and Icon**: Based on the weather condition, apply a background color and display a unique custom SVG icon. Position the SVG icon below all text elements in the weather display.
  - Icons should be stored in `public/icons`, with file names matching conditions (e.g., `clear.svg`, `cloudy.svg`, etc.).

- You may design your own icons or use a free resource, such as SVG Repo or other open-license repositories, to source weather-related icons.
- Refer to zyBook **7.8 JavaScript Object Notation (JSON)** to understand how to work with JSON-formatted data from API responses.

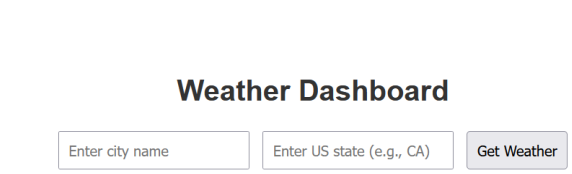
6. **Error Handling (*server.js, script.js*)**

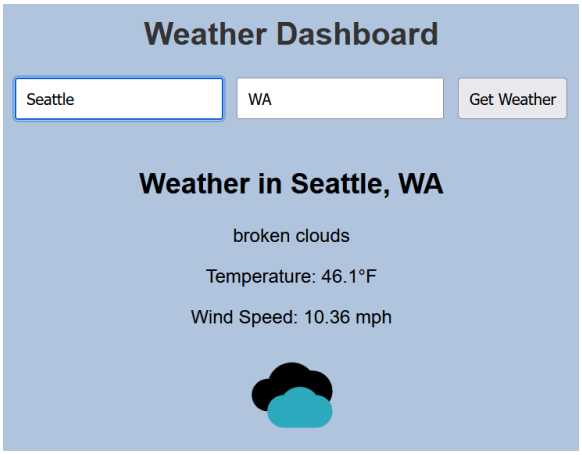
   - Implement error handling for:
     - Missing city or state inputs.
     - Unrecognized locations or failed API requests.
   - Display friendly error messages on the front end for a smooth user experience.
   - Refer to zyBook **7.7 Form validation** as a starting point to validate inputs.

7. **Styling (*styles.css*)**

   - Apply basic CSS styling to make the frontend visually appealing and responsive.
   - Ensure the SVG icon is displayed below all text.

Examples





Submitting Assignment

Submitting your assignment requires submitting in two steps:

1. Submit your assignment to the appropriate GitHub Classroom repository (commit to the repository!). You'll find the invitation URL as part of this assignment on the Canvas assignment page.
2. Submit one URL
    i. Submit your GitHub Classroom repository URL to Canvas (Line 1).

Grading Guidelines:

Setup and Environment Configuration (10 points)

- 5 points for **Environment Variables**: `.env` file is created with `OPENWEATHER_API_KEY` and `PORT` values.
- 5 points for **Secure Configuration**: `.env` variables are used in the code and `.env` is listed in `.gitignore`.

Backend Functionality (30 points)

- 10 points for **Express Server**: Server is set up and runs without errors.
- 5 points for **Geocoding API Integration**: The `/weather` route correctly fetches latitude and longitude based on city and state inputs.
- 5 points for **Weather Data API Integration**: The server retrieves and returns current weather data using latitude and longitude.
- 5 points for **JSON Response Structure**: The API response includes `temperature`, `wind_speed`, `description`, and `condition` properties.
- 5 points for **Error Handling**: Clear error responses are provided for missing parameters, invalid locations, or failed API requests.

Frontend Form and API Request (20 points)

- 5 points for **Form Layout**: User can enter city and state information into form fields.
- 5 points for **Form Validation**: Form prevents submission with empty or invalid fields.
- 5 points for **Fetch API Request**: Data is correctly fetched from the Express API upon form submission.
- 5 points for **Async/Await Usage**: Uses `async/await` in the JavaScript fetch request and handles API errors gracefully.

Dynamic Theme and SVG Icon Display (15 points)

- 5 points for **Dynamic Theme**: Background color changes based on the weather condition.
- 5 points for **SVG Icons**: Correct SVG icon is displayed based on the weather condition, with icon files correctly organized.
- 5 points for **CSS Styling**: CSS is applied to make the interface visually appealing.

User Experience and Error Handling (5 points)

- 5 points for **User-Friendly Messages**: Displays meaningful error messages for invalid input or data-fetching issues.

Documentation and Code Quality (10 points)

- 5 points for **README**: README file includes setup instructions, API references, and usage information as specified.
- 5 points for **Code Comments and Organization**: Code is well-organized with comments explaining functions and logic.