# SENIOR FRONT-END ENGINEER PRACTICAL TEST
# Weather App

## Objective:

Develop a responsive, single-page application (SPA) using HTML, CSS, JavaScript (TypeScript), and Vue.js that interacts with a public API, demonstrating proficiency in the required technologies.

## Instructions:

You are to create a mini weather web application that fulfills the following requirements based on this figma design. This test is designed to assess your skills in front-end development, code organization, problem-solving, logical thinking, and adherence to best practices. This test consists of a series of tasks which may take between 2 - 4 hours to complete, depending on your experience level. Please navigate through the interactive prototype to better understand the scope and requirements.

## Task Overview:

**1. Project Setup:**

- Initialize a new Vue.js project using TypeScript.
- You may use Vue CLI or Vite for scaffolding.
- Set up a clean and logical project structure.
- Configure your development environment to support modern JavaScript features and linting tools.

**2. API Integration:**

- Use  the OpenWeatherMap API https://openweathermap.org/api
- Fetch data from the API and handle asynchronous operations gracefully.
- Implement proper error handling for API requests (loading states, error messages).

**3. Component Architecture:**

- **Atomic Design Principles:**

    - Structure your components following the Atomic Design methodology. This includes organizing components into the following categories:
        - **Atoms:** Basic building blocks (e.g., buttons, input fields, icons).
        - **Molecules:** Groups of atoms functioning together (e.g., a search bar combining an input field and a button).
        - **Organisms:** Complex, reusable sections of the UI (e.g., a navigation bar or product card list).
        - **Templates:** Page-level structures defining the layout.
        - **Pages:** Fully assembled views composed of templates and data.
    - Clearly organize your project folder structure to reflect this hierarchy.
- **Components to build:**
    - **List Component:** Displays a list of items retrieved from the API.
    - **Detail Component:** Shows detailed information about a selected item.
    - **Form Component:** Allows users to input data (e.g., search functionality or data submission).
        - Include robust form validation to ensure data integrity:
        - Validate required fields (e.g., prevent empty submissions).
        - Use appropriate validation patterns (e.g., email format, character limits).
        - Provide user-friendly error messages and visual cues for invalid inputs.
- Use Vue Router to navigate between different views/components.
- Apply the Composition API for state and logic management within components.

**4. State Management:**

- Implement global state management using Vuex or the Vue Composition API with reactive state.
- Ensure state consistency across components and views.
- Manage complex state (nested objects, arrays) effectively.

**5. TypeScript Proficiency:**

- Utilize TypeScript features such as interfaces, enums, generics, and type annotations.
- Ensure type safety throughout the application.
- Configure strict type-checking options in your tsconfig.json.
- Use async/await for handling asynchronous code.

**6. Styling and Responsive Design:**

- Style your application using CSS, SCSS, or a CSS-in-JS solution.
- Implement a responsive layout using Flexbox or CSS Grid.
- Ensure the application is visually appealing and consistent across devices.
- Follow a consistent naming convention (e.g., BEM) for CSS classes.

**7. Accessibility and Semantic HTML:**

- Use semantic HTML5 elements appropriately.
- Ensure your application is accessible (ARIA roles, keyboard navigation, contrast ratios).

**8. Performance Optimization:**

- Optimize the application for performance.
- Lazy load components where appropriate.
- Minimize unnecessary re-renders.
- Optimize images and assets.

**9. Web API Integration:**

- Select and integrate at least one advanced Web API into your application. Full list of Web API available here https://developer.mozilla.org/en-US/docs/Web/API
- The chosen Web API should provide a clear enhancement to the functionality, performance, or user experience of the application.

**10. Documentation and Git Practices:**

- Write a clear and concise README.md explaining:
    - Project setup and installation instructions.
    - Your approach and any architectural decisions.
    - How to run tests.
    - Use meaningful commit messages following a convention (e.g., Angular's commit message guidelines).
- Document your code with comments where necessary.

# Submission Guidelines:

- Upload your code to a public Git repository (GitHub, GitLab, etc.).
- Ensure all dependencies are listed in your package.json.
- Include instructions on how to run the application in README.md
- Do not include node_modules in your repository.

# Evaluation Criteria:

**1. Code Quality and Organization:**

- Clean, readable, and maintainable code.
- Proper use of modern JavaScript/TypeScript features.
- Logical project and file structure.

**2. Functionality and Correctness:**

- The application meets all the specified requirements.
- Works without errors or bugs.
- Handles edge cases and error states gracefully.

**3. Technical Proficiency:**

- Demonstrated expertise in HTML, CSS, JavaScript, TypeScript, and Vue.js.
- Effective use of Vue.js features (components, directives, lifecycle hooks).

**4. Best Practices and Patterns:**
- Proper use of design patterns and architectural principles.
- Adherence to DRY (Don't Repeat Yourself) ,KISS (Keep It Simple, Stupid) principles and SOLID principles.
- Efficient state management and data flow.

**5. Performance and Optimization:**
- Efficient loading and rendering.
- Optimization techniques applied where appropriate.

**6. Accessibility and Semantics:**
- Use of semantic HTML elements.
- Accessibility considerations implemented.

**7. Styling and UX:**
- Visual appeal and user experience.
- Consistency in styling and responsive design.

**8. Documentation and Communication:**
- Clarity and thoroughness of the README.md.
- Quality of code comments and documentation.
- Clarity and consistency in commit messages.

## Additional Notes:

- **Demonstrate your capabilities:** Please ensure that you demonstrate your capabilities as a senior front-end engineer throughout the project.
- **Time Management:** While there is no strict time limit, we recommend spending no more than 8-12 hours on this task to simulate real-world constraints.
- **Questions:** If you have any questions or need clarifications, feel free to reach out.

We look forward to reviewing your submission and discussing it with you!