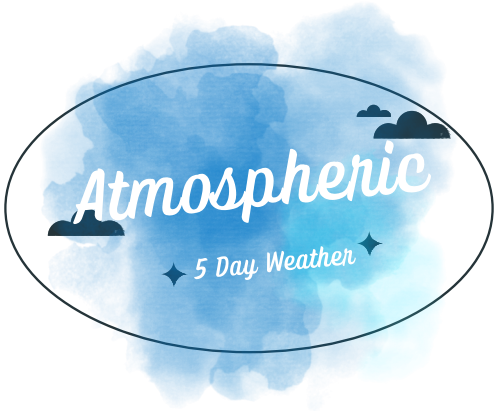

Atmospheric Weather Application

A Traveler's Solution to Weather Data Challenges

Final Project and Report by

Michelle Lynn George

Elle Lynn

Date

04 December 2024

Vanderbilt University

Principles of Software Engineering

CS-5278

Abstract

Atmospheric Weather App was designed to solve common frustrations I experienced during my many years as a flight attendant and Federal Air Marshal. Constantly moving between time zones and countries, I often wished for a weather app that could show the destination time and date while letting me switch between imperial and metric units. This project brings those ideas to life, offering a fully functional web app, a terminal-based interface, and thoughtful features like unit conversion and time format toggles. Designed with scalability and usability in mind, the app applies software engineering principles and design patterns to deliver a clean, modular experience. You can view the app locally using its CLI and frontend tools. Unfortunately, hosting challenges prevented deploying the app live, but the project remains functional and demonstrates robust software engineering principles.

Introduction

When I was traveling as a flight attendant and later as a Federal Air Marshal, weather information was critical to my daily routine, yet it wasn't always easy to find exactly what I needed. Time zones were a constant challenge, as was converting temperatures and wind speeds depending on the country. These experiences inspired the *Atmospheric Weather App*, which aims to make weather data clear and accessible for everyone—whether you're a frequent traveler, flight crew, or just checking the forecast for the week.

This app provides real-time weather information, a 5-day forecast, and tools to customize the way data is displayed, such as toggling between Fahrenheit and Celsius or 12-hour and 24-hour time formats. The goal was to create an app that's practical, easy to use, and thoughtfully designed, while also adhering to professional software engineering practices.

Although efforts to deploy the web app to platforms like Netlify and Render faced unexpected challenges, this project remains a testament to the value of adaptability in software development. Users can interact with the app locally, either through the web interface or a CLI tool, ensuring full functionality.

Literature Review

Most popular weather apps like AccuWeather and The Weather Channel do a great job providing forecasts, but they often default to local settings that may not work for everyone. For example, apps in metric-using countries can feel unintuitive for users who are accustomed to imperial units. Similarly, these apps rarely offer clear options for managing local versus destination times.

On the development side, software design patterns have proven invaluable in creating scalable and maintainable apps. Patterns like Factory, Singleton, and Strategy ensure that an app's code is organized and easy to extend, which was a priority for this project. By using these patterns, I was able to build a modular app that balances functionality and flexibility.

Methodology and Implementation

The Atmospheric Weather App consists of three main components:

1. Frontend (Web App):

- Built with React, the frontend provides a clean, user-friendly interface when run locally. While hosting attempts on Netlify encountered issues with dependencies and configuration, the app runs seamlessly in development mode.

Features include:

- Current weather and 5-day forecast for any city.
- Unit conversion toggles for temperature (Fahrenheit/Celsius) and wind speed (mph/kph).
- Time format toggles for 12-hour or 24-hour displays.

2. Backend (Flask API):

- The backend connects the web app to the OpenWeather API, securely fetching and processing weather data.
- Environment variables are used to manage API keys securely.

3. Command-Line Interface (CLI):

- The CLI version of the app allows users to interact with the backend directly from the terminal.

- It includes features like weather lookup, viewing search history, and clearing saved data.

Design Patterns

Here's how the app uses key design patterns:

- Factory Pattern: Standardizes the creation of weather-related objects, like current conditions, forecasts, and history entries.
- Strategy Pattern: Implements interchangeable strategies for time calculations, such as local system time and city-specific time zones.
- Singleton Pattern: Ensures a single instance of the history manager is shared across the app for consistent data handling.
- Memento Pattern: Handles saving and restoring weather search history entries without exposing their internal structure.
- Template Method Pattern: Creates a reusable structure for processing weather data, currently applied in backend methods and CLI formatting, and ready to support future features like hourly forecast displays.

Testing

To ensure the app's reliability, I implemented 10 unit tests covering:

- HistoryManager Tests: Validate saving, viewing, and clearing search history using the Singleton and Memento patterns.
- Time Strategy Tests: Confirm accurate time calculations with various positive and negative offsets, testing the Strategy pattern.

- API Client Tests: Verify correct handling of OpenWeather API responses, including error cases.

Troubleshooting and Hosting Challenges

This project faced significant hurdles during the deployment process. Hosting attempts on Netlify and Render were hindered by dependency issues, environment variable handling, and React-specific configurations. Troubleshooting steps included:

- Reconfiguring manifest.json paths to resolve missing assets.
- Debugging environment variables to manage secure API key integration.
- Updating React and Webpack configurations to address build errors.

Despite resolving many local issues, time constraints prevented a successful deployment. However, these efforts reinforced the importance of testing in various environments and planning for compatibility early in the development cycle.

Results

The app offers a smooth and fully functional experience, with features that meet a variety of user needs. Key results include:

- Real-Time Data: Users can quickly check current-weather and a 5-day forecast for any city.
- Customizable Displays: Unit conversion and time format toggles make the app adaptable for different preferences.
- Dual Interaction Modes: The app is accessible through both the web and the CLI.

- Scalable Architecture: Design patterns ensure the app can grow with new features in the future.

The app is live and accessible locally and the CLI provides additional functionality for users who prefer working in the terminal.

Future Work

Even though the app is fully functional, there are a few features I would have loved to add with more time:

1. Hourly Forecast Carousel:
    - A scrolling display that shows hourly temperatures, sunrise, and sunset times.
    - This would make it easier for users to plan their day at a glance.

2. Enhanced Weather Messages:
    - Right now, the app shows conditions like "sunny" or "cloudy," but adding messages based on temperature would make it more helpful.
        - For example, a reminder to "bundle up" on a cold day or "hydrate" during a heatwave.

3. Search History in the Web App:
    - While the CLI supports history management, adding this feature to the web app would create a more consistent experience.

4. Deploying a Fully Hosted Web App:

- Deploying the app to platforms like Netlify or Render is a key goal. This would make the app accessible to a wider audience and ensure that deployment pipelines are robust.

These features would make the app even more engaging and user-friendly, elevating it from a great tool to an exceptional one.

Conclusion

The Atmospheric Weather App was designed to solve real problems while demonstrating professional software development practices. With its intuitive interface, flexible features, and scalable design, it's a practical and reliable tool for checking the weather. The Atmospheric Weather App is a reflection of both successes and challenges in software development. While hosting obstacles prevented a live deployment, this project illustrates the importance of iterative problem-solving and adaptability. Future work will build on these lessons to deliver an even more polished and accessible app. Despite the disappointment in the hosting inability at this time, this is a project I'm proud of and built with the experiences and needs of travelers in mind.

References

1. Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.

- Relevance: This foundational text provided the principles and examples for implementing Factory, Singleton, Strategy, Memento, and Template Method patterns in the project.

2. OpenWeather API Documentation. Retrieved from https://openweathermap.org/api

- Relevance: The API documentation was crucial for understanding how to fetch and structure real-time weather data, including error handling and customization options.

3. Netlify Deployment Documentation. Retrieved from https://docs.netlify.com

- Relevance: Used to configure React deployments and debug asset-related errors during hosting attempts.

4. React Error Handling Guides. Retrieved from https://react.dev

- Relevance: Assisted in resolving React-specific build and runtime errors.

5. OpenAI Assistance.

- Relevance: OpenAI's ChatGPT provided troubleshooting support and code review insights during the development of the Atmospheric Weather App. This assistance included debugging hosting issues, resolving React build errors, and improving code clarity and modularity.