**Weather Dashboard – Full Stack Project Overview**

This document summarizes the design, architecture, and improvements implemented in the Weather Dashboard project. It highlights both the technical structure and the challenges faced during development, along with the solutions applied.

**Tech Stack**

• Frontend: React (Vite, Context API, Custom Hooks, CSS styling)

• Backend: ASP.NET Core 8 Web API (C#)

• Testing: NUnit (API), React Testing Library (UI)

• Resiliency: Polly (Retry, Timeout Policies)

• Caching: In-Memory Cache (IMemoryCache)

• Dependency Injection: Built-in .NET DI container

• Mapping: AutoMapper for DTO ↔ Model conversions

• Dev Tools: Swagger for API docs, LoggerFactory for structured logging

**Architecture Overview**

The Weather Dashboard application is built with a clean separation between a React front-end and an ASP.NET Core back-end.  
The code follows best practices including async/await, dependency injection, structured logging, and component-based design.  
The application implements caching to optimize API usage, robust error handling for invalid inputs, and a responsive UI.  
Both layers include automated unit tests for core features such as weather retrieval, caching, and default city management..

Key components include:  
- DTO Layer: Defines data contracts between the API and client.  
- Service Layer: Handles HTTP calls, caching, retry policies, and error handling.  
- Controller: Orchestrates API responses and validation.  
- React Frontend: Implements Context for global weather state and a Custom Hook for business logic.

**Challenges and Resolutions**

1. **Slow API responses and retry loops:**

Problem: API calls for invalid cities were taking up to 14 seconds due to nested retry policies.  
Resolution: Removed duplicate retry logic from WeatherService and centralized retry + timeout in Program.cs using Polly. Added explicit 5-second timeout and graceful handling for TaskCanceledException.

1. **DTO Mapping and Null Safety Warnings**

Problem: Non-nullable warnings were appearing due to DTO properties without initial values.  
Resolution: Made DTO properties nullable or initialized defaults, and introduced AutoMapper for clean conversions.

1. **React-side Validation & State Handling**

Problem: Validation messages were not displaying properly when users set a default city with no input.  
Resolution: Introduced a centralized custom hook (useWeatherDashboard) that manages all UI messages and error states with temporary timeouts.

1. **UI/UX and Responsiveness**

Problem: Dashboard layout and messages were not mobile-friendly.  
Resolution: Refined internal CSS with flexible grid layouts, centered titles, and responsive message styling without relying on Tailwind.

1. **Testing Stability**

Problem: React tests failed intermittently due to asynchronous message rendering.  
Resolution: Used waitFor and findByText to handle delayed DOM updates correctly.

**Outcome and Learnings**

The final solution is resilient, cleanly architected, and production-ready. It handles transient API failures gracefully, provides fast cached responses, and presents a polished UI. The project demonstrates end-to-end engineering skills in .NET, React, and DevOps practices.