Technical Test Overview

Objective:
Develop a microservice in ASP.NET Core(.NET 8) that integrates with an external weather data API and expose RESTful endpoints to retrieve current weather, forecast data, and optionally store historical data. Additionally, build a front-end application that consumes the backend services to display the weekly weather forecast for San Salvador. The entire solution should implement caching, resilience strategies (such as a circuit breaker), robust error handling, and include comprehensive tests.

Scope:

- **API Integration:** Fetch current weather and forecast data from an external API (e.g., OpenWeatherMap, Weatherbit, etc.).
- **Backend Microservice:**
    - Expose RESTful endpoints.
    - Persist historical data with a relational database using Entity Framework.
    - Implement caching and a circuit breaker to enhance resilience.
- **Front-End Application:**
    - Consume the backend endpoints to display the weather forecast for the current week specifically for San Salvador.
    - Provide a responsive and user-friendly view of the forecast.
- **Testing:** Include unit tests and integration tests.

Requirements

1. Backend Microservice Development

Develop an ASP.NET Core microservice with the following endpoints:

A. Service Endpoints

1. **GET /weather/current**
    - Retrieves current weather data from the external API.
    - Implements in-memory caching to prevent redundant API calls.

- Provides robust error handling if the external API is unavailable.

2. **GET /weather/forecast**
    - Retrieves forecast data (covering at least the next 7 days).
    - Should allow filtering by city or geographic region, with special focus on San Salvador when called by the front-end.

B. Persistence with Entity Framework

- **Data Models and Migrations:**
  Set up the required models to store weather data. Include migrations for database initialization.

C. Resilience and Optimization

- **Caching:**
  Use in-memory caching (or a similar approach) to store API responses for a set duration (e.g., 10 minutes).
- **Circuit Breaker:**
  Implement or utilize a circuit breaker pattern to handle external API failures gracefully.

D. Testing

- **Unit Tests:**
  Provide unit tests for core business logic—especially for handling API responses, caching, and exception scenarios.
- **Integration Tests:**
  Validate end-to-end behavior, including communication with the external API (or a mocked service) and database operations.

E. Logging & Error Handling

- Integrate a logging framework (like Serilog or the built-in ASP.NET Core logger) to track operations, API calls, and errors.
- Ensure structured exception handling with informative error messages.

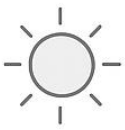
---

3. Front-End Application Development

Home About Contact

Weather Forecast for the Current Week in San Salvador

Monday



min° / max°
Sunny

Tuesday

min° / max°
Partly Cloudy

Wednesday

min° / max°
Sunny

Thursday

min° / max°
Sunny at

Friday

min° / max°
Rum max

Saturday

min° / max°
Sunnat

Saturday

min° / max°
Partly Cloudy

Sunday

min° / max°
Sunny

Contact information Copyright

Create a simple yet responsive front-end application whose purpose is to display the weather forecast for the current week in San Salvador. This part is meant to assess your ability to integrate the backend with a presentation layer.

Requirements:

- **Technology Stack:**
  Use a web framework or library of your choice (e.g., React, Angular, Vue, or even a simple HTML/JavaScript app). If you prefer a single-page application (SPA) framework, ensure minimal setup complexity.
- **Functionality:**
  - **Fetch Forecast Data:**
    The front-end should send a GET request to /weather/forecast with appropriate filtering or parameters to fetch data specifically for San Salvador.
  - **Display Data:**
    Render the weekly forecast in a clear, user-friendly format. Consider using cards, lists, or a table for each day's forecast, including key details such as:
    - Date
    - High/Low temperatures
    - Weather condition (e.g., cloudy, sunny)
    - Icon or graphical representation (if provided by the API)
  - **Responsive Design:**
    Ensure the interface is mobile-friendly and adapts to various device widths.
  - **User Instructions/Interaction:**
    Optionally add refresh or search functionality so the user could re-query data if necessary.
- **Integration Details:**
  Provide a README section or inline comments detailing:
  - How to run and build the front-end.
  - How the front-end communicates with the backend service.
  - Any configuration necessary (for instance, if the backend's base URL needs to be set).

Environment & Deliverables

Controlled Environment:

- The test will be conducted in a provided or controlled development environment to limit external tool usage. This ensures that the work is genuinely your own.

Submission Requirements:

- Source code repository (e.g., Git) with commit history for both backend and front-end parts.
- Test projects with unit and integration tests.
- Clear README instructions for running both the backend and front-end applications.

Evaluation Criteria

- **Code Quality:**
  Clear, maintainable code that follows best practices and proper design patterns.
- **Technical Depth:**
  Effective implementation of caching, circuit breaker patterns, error handling, and integration with the external API.
- **Testing Rigor:**
  Strong coverage through unit and integration tests, ensuring robustness and correctness.
- **Integration & Functionality:**
  A cohesive system where the front-end seamlessly consumes and displays the weather data provided by the backend.