Laboratory 12 - Thread, Task, Async, Await, Parallel

Introduction

Your task for this laboratory is to write a program that will, in an asynchronous way, download and process weather data for specified cities. You are provided with the `HttpSimulator` library class, which mock-ups a weather service, and a `Main` method that demonstrates the basic usage of this class and `HttpClient` for retrieving temperature data.

Example API Request:

For a request:

`https://127.0.0.1:2137/api/v13/forecast?city=New%20York&daily=temperature`

The HTTP response contains data as JSON in the following structure:

```
{
    "Daily": {
        "Temperature": [12.3, 7.8, 14.5, -2.1, 6.9, 10.4, 8.7]
    }
}
```

The `Temperature` array represents the daily temperature (in Celsius) for the next 7 days for the requested city.

Requirements:

1. Use asynchronous programming techniques (`async/await`, `Task`) to handle requests and data processing efficiently.
2. Handle all potential errors properly, including HTTP request failures, deserialization errors, and invalid input.
3. Ensure the program is modular and easy to understand.

Tasks

Task 1: Fetch Weather Data Asynchronously (3 points)

1. Rewrite the provided starter code to fetch weather data asynchronously using `HttpClient`.
2. Deserialize the JSON response into the provided `Data.cs` classes.
3. Fix a bug in the `HttpSimulator` class inside the `ForecastEndpoint` method (details are in the starter code).
4. Ensure proper error handling for network issues and deserialization errors.

Task 2: Process Data - Calculate Average Temperature (3 points)

1. Implement a method to calculate the **average temperature** for each city.
2. Use asynchronous techniques for processing the data.
3. Ensure the method can handle edge cases, such as empty or missing temperature data.

Task 3: Additional Data Processing (2 points)

1. Identify **extreme temperature days** (days with temperatures below 0°C or above 30°C) for each city.
2. Find the city with the **highest average temperature** across all cities.
3. Perform all checks asynchronously, ensuring non-blocking execution.

Additional Notes:

- **Error Handling:** Ensure robust error handling, including invalid city names, API failures, and malformed JSON responses.
- **Performance:** Optimize the solution using asynchronous programming to fetch and process data for multiple cities in parallel.