# Weather Forecast Service – Design Specification

**Framework**

.NET 7

**Architecture**

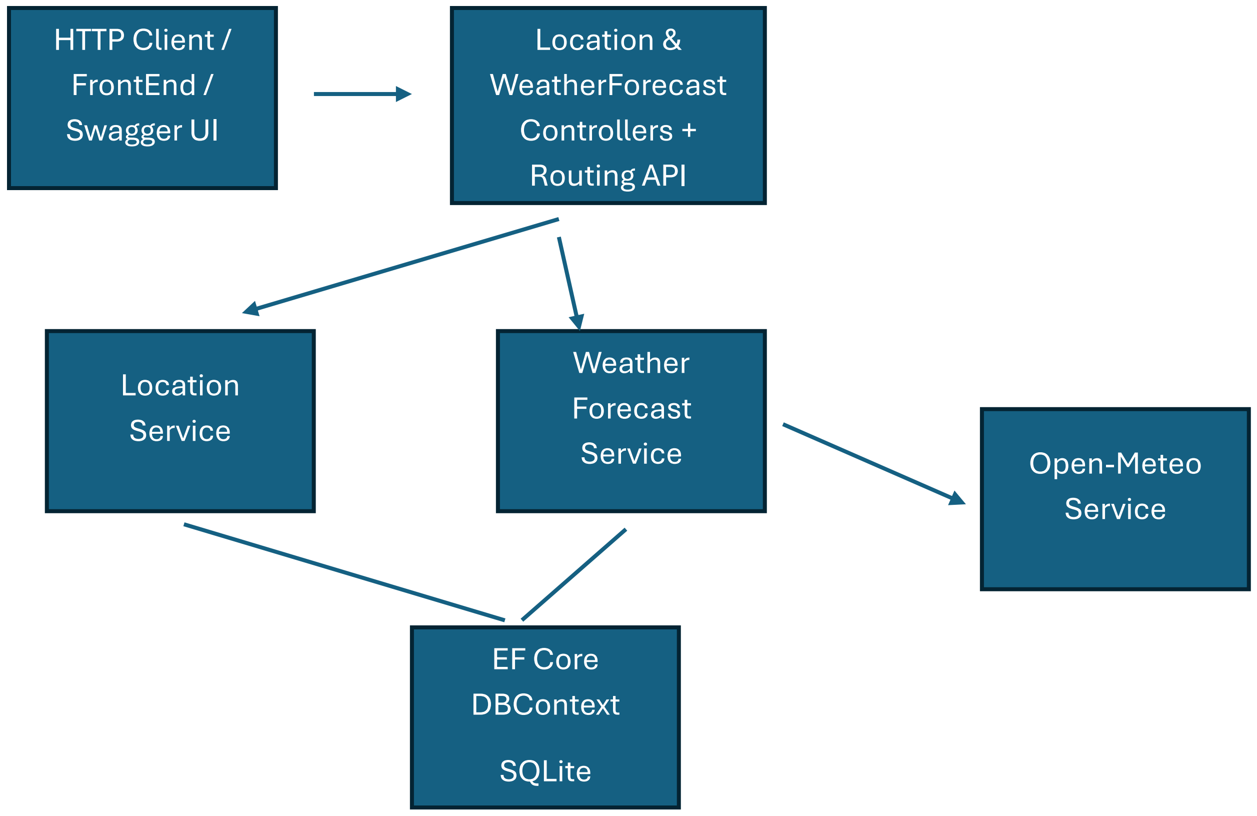
Clean Layered API with EF Core + SQLite + External REST Integration

## 1. Purpose

This project provides a **RESTful API** that allows clients to:

1. **Manage Locations** — add, list, and delete latitude/longitude coordinates, persisted in a SQLite database.
2. **Retrieve Weather Forecasts** — fetch live weather data for any coordinates (or stored location) using the Open-Meteo public API.

## 2. System Overview


HTTP Client / FrontEnd / Swagger UI
Location & WeatherForecast Controllers + Routing API
Location Service
Weather Forecast Service
Open-Meteo Service
EF Core DBContext
SQLite

## 3. Component Design

### 3.1 Controllers

**LocationController**

- **Route:** /api/locations
- **Responsibilities:**
    - POST — Add a new location or return existing one.
    - GET — Retrieve all stored locations.
    - DELETE — Remove a location by ID.

**Endpoints:**

| **Method** | **Route** | **Description** | **Returns** |
|---|---|---|---|
| POST | /api/locations | Adds new latitude/longitude | 200 OK with Location |
| GET | /api/locations | Lists all locations | 200 OK / 404 Not Found |
| DELETE | /api/locations/{id} | Deletes a location by ID | 204 No Content / 404 |

**WeatherForecastController**

- **Route:** /api/weatherforecast
- **Responsibilities:**
    - Fetch current weather for any given coordinates.
    - Fetch current weather for a stored location (by ID).

**Endpoints:**

| **Method** | **Route** | **Query/Params** | **Description** | **Returns** |
|---|---|---|---|---|
| GET | /api/weatherforecast | latitude, longitude | Gets live forecast from Open-Meteo | 200/400/404 |
| GET | /api/weatherforecast/{id} | Location ID | Gets forecast for stored location | 200 OK / 404 |

Both endpoints validate coordinate ranges and wrap results into clean Data Transfer Object (WeatherForecastResult).

### 3.2 Services

**LocationService**

Encapsulates all database operations on the Location entity using EF Core.

**Methods**

| Method | Purpose |
|---|---|
| AddLocationAsync() | Adds location if not duplicate (unique latitude/longitude pair). Handles concurrent inserts gracefully. |
| GetAllLocationsAsync() | Returns all locations using AsNoTracking() for performance. |
| GetLocationByIdAsync() | Finds location by primary key. |
| DeleteLocationAsync() | Removes record and commits transaction. |

**Concurrency Handling:**

DbUpdateException with UNIQUE constraint is caught and the existing record is returned.

**WeatherForecastService**

Handles communication with the external **Open-Meteo API** using injected HttpClient.

**Configuration**

- Reads base URL from appsettings.json → "WeatherApi:BaseUrl".
- Defaults to https://api.open-meteo.com/v1/.

**Method**

```
Task<WeatherForecast?> GetCurrentWeatherAsync(
  double latitude, double longitude, CancellationToken cancelToken)
```

**Behavior**

1. Builds endpoint:
   forecast?latitude={lat}&longitude={lon}&current_weather=true
2. Sends HTTP GET request.
3. On success → deserializes JSON into WeatherForecast.
4. Catches and logs network, cancellation, or JSON errors gracefully.
5. Returns null if anything fails.

### 3.3 Data Layer

**AppDbContext**

- Defines DbSet<Location> Locations
- Uses **SQLite** via UseSqlite(connectionString).
- Enforces **unique composite index** on (Latitude, Longitude).

**AppDbContextFactory**

Used only by dotnet EF CLI for migrations.

Builds configuration and constructs AppDbContext with SQLite connection.

### 3.4 Models and Data Transfer Objects

| Type | Purpose |
|---|---|
| Location | Database entity for coordinates + creation timestamp |
| WeatherForecast & CurrentWeather | Map Open-Meteo JSON response (JsonPropertyName attributes preserve snake_case) |
| AddLocationRequest | Input DTO for creating locations |
| WeatherForecastResult | Output DTO for formatted API response |
| GetWeatherRequest | Alternate request object for latitude/longitude queries |

## 4. Persistence Design

**Database:** SQLite
**Entity:** Location
**Table Structure:**

| Column | Type | Constraint |
|---|---|---|
| Id | INTEGER | PK, Auto Increment |
| Latitude | REAL | Required, Range –90 to 90 |
| Longitude | REAL | Required, Range –180 to 180 |
| CreationTime | TEXT | Default = UTC Now |
| **Unique Index** | (Latitude, Longitude) | Prevent duplicates |

## 5. Configuration

**appsettings.json**

```
{
 "ConnectionStrings": {
  "DefaultConnection": "Data Source=weather.db"
 },
 "WeatherApi": {
  "BaseUrl": "https://api.open-meteo.com/v1/"
 }}
```

## 6. Error Handling & Validation

| Layer | Type | Behavior |
|---|---|---|
| Controller | Bad input | Returns 400 Bad Request |
| Controller | Missing data | Returns 404 Not Found |
| Service | DB conflict or network error | Logs to console, returns null or existing record |
| DataAnnotations | [Range], [Required] | Automatically validated by ASP.NET Model Binding |

**7. Dependency Injection Setup**

```
builder.Services.AddDbContext<AppDbContext>(options =>
   options.UseSqlite(builder.Configuration.GetConnectionString("DefaultConnection")));

builder.Services.AddScoped<LocationService>();
builder.Services.AddHttpClient<WeatherForecastService>();

builder.Services.AddControllers();
builder.Services.AddSwaggerGen();
```

**8. Testing Strategy**

| **Scope** | **Framework** | **Description** |
|---|---|---|
| Unit Tests | **xUnit** | Tests WeatherForecastService using fake HttpMessageHandler. |

**Example test coverage:**

- Success response → valid forecast.
- Network failure → returns null.
- Malformed JSON → handled gracefully.
- Config missing → defaults to base URL.

**9. Deployment & Execution**

**To run locally:**

```
dotnet restore
dotnet ef database update
dotnet run
```

Open: https://localhost:{port}/swagger

**Deliverables**

- Modular C# solution (.NET 7 API project + Test project)
- SQLite database with EF Core Migrations
- Swagger for API exploration
- Unit test coverage for Weather Service