**Weather Data Aggregator - Project Documentation**

**Overview**

The Weather Data Aggregator is a Java-based application designed to fetch and display real-time weather data for multiple cities using a graphical user interface (GUI). It uses a multi-threaded architecture to ensure responsiveness and efficiency, allowing weather data to be fetched concurrently for multiple cities.

**Key Features**

1. **Real-Time Weather Data:** Fetches current temperature, wind speed, and wind direction for selected cities.
2. **Graphical User Interface (GUI):** Built using Swing for user-friendly interaction.
3. **Multi-Threading:** Utilizes threads to handle concurrent API requests efficiently.
4. **Dynamic Updates:** Automatically updates the weather data every 5 seconds.
5. **City Selection:** Allows users to select from a predefined list of cities.
6. **Custom Thread Count:** Enables users to specify the number of threads for concurrent operations.
7. **Separate Windows:** Displays individual weather data in dedicated windows for each city.
8. **Table Display:** Summarizes weather data in a JTable for easy reference.

**Technologies and Libraries Used**

**1. Programming Language:**

* Java 17

**2. Libraries and APIs:**

* **Swing**: For building the graphical user interface.
* **Google GSON:** For parsing JSON responses from the weather API.
* **Java Concurrency Utilities:** To manage threads, including ExecutorService, Callable, Future, and ScheduledExecutorService.
* **HTTPURLConnection:** For making HTTP requests to the weather API.

**3. API Used:**

* **Open-Meteo API:** Provides real-time weather data based on latitude and longitude.

**Components and Their Roles**

**1. Main Application:**

* **WeatherProject**
  + Sets the default locale to English.
  + Launches the GUI by initializing WeatherAppGUI.

**2. Graphical User Interface (GUI):**

* **WeatherAppGUI**
  + Provides an interface for users to select cities, specify the number of threads, and view weather data.
  + Includes input fields, buttons, and a JTable to display weather data.

**3. Weather Fetching:**

* **WeatherFetcher**
  + Implements Callable<WeatherData>.
  + Fetches weather data using the Open-Meteo API based on city coordinates.
  + Parses JSON responses to extract temperature, wind speed, and wind direction.

**4. Weather Data Model:**

* **WeatherData**
  + A simple data class to store weather attributes (city, temperature, wind speed, wind direction).

**5. Thread Management:**

* **BuildThread**
  + Manages thread pools using ExecutorService.
  + Spawns threads to fetch weather data concurrently for multiple cities.
  + Updates the JTable and creates individual ThreadWindow instances for each city.

**6. Individual Weather Display:**

* **ThreadWindow**
  + Displays weather data for a single city in a dedicated window.
  + Updates dynamically as new data is fetched.

**Multi-Threading Implementation**

1. **ExecutorService:**
   * Fixed thread pool is used to manage threads efficiently based on the user-specified thread count.
2. **Concurrency Utilities:**
   * **Callable**: Fetches weather data asynchronously.
   * **Future**: Retrieves results of the Callable tasks.
   * **ScheduledExecutorService**: Schedules periodic weather data updates.
3. **Thread Safety:**
   * **ConcurrentHashMap:** Ensures safe access and updates to shared city window objects.
   * Swing updates are executed on the Event Dispatch Thread using SwingUtilities.invokeLater to prevent race conditions.
4. **Deadlock Prevention:**
   * No synchronized blocks or locks are used that could cause circular dependencies.

**Challenges and Solutions**

1. **Concurrency Issues:**
   * Ensured thread-safe operations using concurrent data structures and proper thread management.
2. **API Errors:**
   * Handled HTTP errors gracefully by throwing exceptions and logging errors.
3. **GUI Responsiveness:**
   * Performed all long-running tasks (e.g., API calls) on separate threads to keep the GUI responsive.

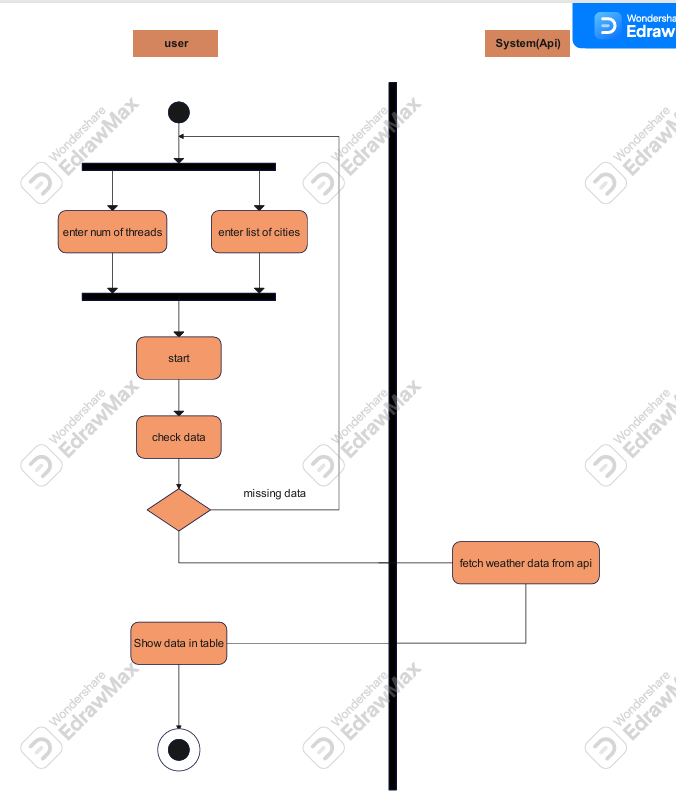
**The efficiency of your system with multithreading compared to a single-threaded version largely depends on the following factors:**

1. **Parallelism of Network Calls**:
   * **Multithreading**: Each weather data fetch operation runs independently in its own thread, utilizing network latency to process multiple cities simultaneously. While one thread waits for a network response, others can fetch data.
   * **Single-threading**: In a single-threaded version, the program would fetch data for one city at a time sequentially, wasting time during network waits.
2. **Number of Cities**:
   * If you are fetching data for many cities (e.g., 10+), multithreading will show a significant performance improvement because network calls are the primary bottleneck.
   * For a small number of cities, the difference might be negligible.
3. **Number of Threads**:
   * The thread pool size impacts performance. Too many threads can lead to overhead, while too few might underutilize available resources.
   * An optimal number of threads (e.g., equal to or slightly greater than the number of processor cores) can maximize efficiency.

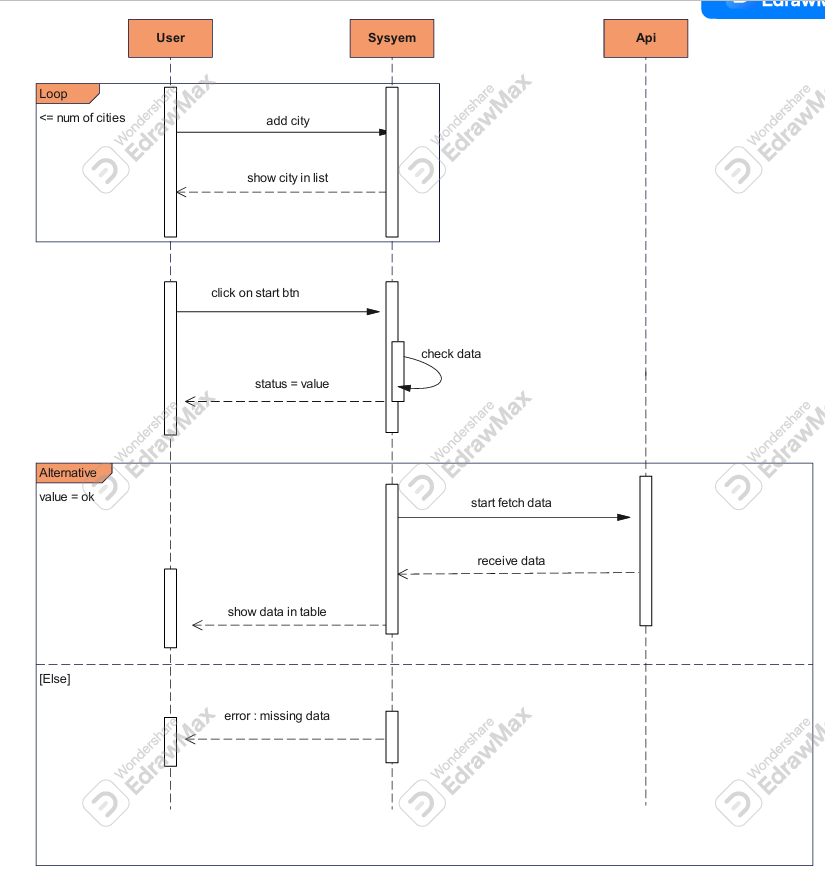
**References**

* [Open-Meteo API Documentation](https://open-meteo.com/)

Activity Diagram



Sequence Diagram



Visual representation

