Weather Data Collection and Analysis System

System Requirements and Design Decisions Overview Report

Project Overview

This project is a weather data collection and analysis system designed to periodically collect weather data, analyze it, and provide data visualization through a web application. The system automates the process of fetching weather data from public sources, storing it in a database, analyzing the collected data, and displaying it through a user-friendly web interface.

Problems Addressed

The system aims to solve the problem of automated weather data collection and analysis. By periodically fetching weather data from public sources, storing and analyzing the data, users can easily view historical and current weather conditions.

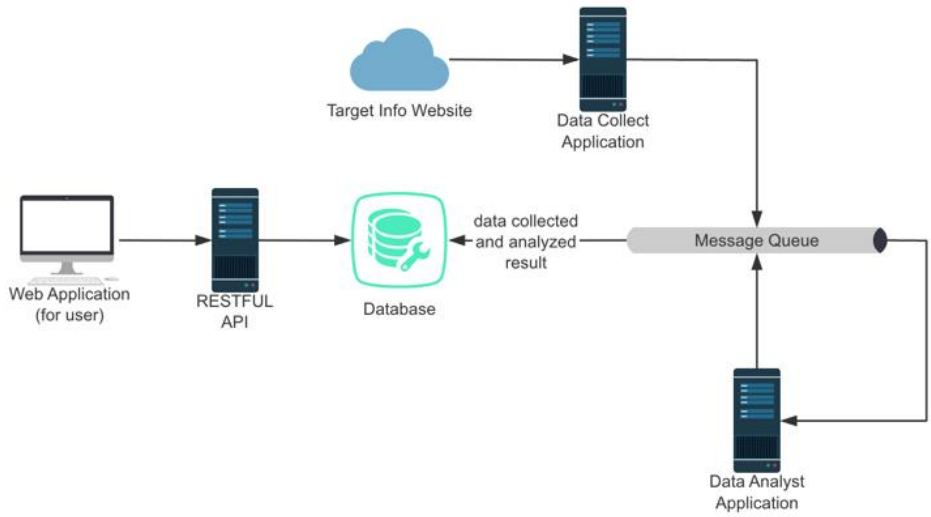
Target Audience

- Data scientists and analysts
- Weather enthusiasts
- Industry professionals needing weather data for decision-making, such as agriculture and logistics

Unique Features

- Automated data collection and analysis
- Uses lightweight components (e.g., SQLite and RabbitMQ)
- Provides a RESTful API for easy data access and integration
- Static web application for simple and intuitive data visualization

Architecture Diagram



Main Components and Their Interactions

Data Collection Application

Periodically fetches data from a public weather API and stores it in the SQLite database.
Uses APScheduler to schedule data collection tasks.
This service ensures that the latest weather data is available in the database for analysis.

RESTful API

Provides endpoints for the frontend to query weather data.
Acts as an interface between the frontend and the backend database.
Allows easy access to weather data for both the web application and potential third-party integrations.

Data Analysis Application

Analyzes the data stored in SQLite.
Uses RabbitMQ to receive messages that trigger analysis tasks.
Processes the collected data to generate insights or prepare it for visualization.

Message Queue

Uses RabbitMQ to coordinate data collection and analysis tasks.
Decouples data collection and analysis processes to ensure scalability and flexibility.
Facilitates efficient communication between different parts of the system.

Web Application

A static frontend that displays data fetched from the RESTful API.

Provides users with a simple and intuitive interface to view weather data.
Displays both current and historical weather conditions.

Design Decisions and Rationales

Database Choice

SQLite:

- **Rationale**: SQLite is chosen for its simplicity, lightweight nature, and ease of setup. It is suitable for this project as the amount of data and the number of concurrent users is expected to be moderate. SQLite does not require a separate server process, making deployment straightforward.
- **Alternative Consideration**: A more complex database like PostgreSQL or MySQL could be used if the project scales significantly, requiring more advanced features like concurrent access management and complex querying capabilities.

Message Queue

RabbitMQ:

- **Rationale**: RabbitMQ is a robust and lightweight message broker that supports various messaging protocols. It decouples the data collection and analysis processes, allowing them to run independently and scale as needed. RabbitMQ is easy to set up and integrate with the existing components.
- **Alternative Consideration**: Apache Kafka could be considered for higher throughput and more complex message processing requirements. However, RabbitMQ is more than sufficient for the current needs of this project.

Frontend Technology

Static Web Application:

- **Rationale**: A static web application is chosen for its simplicity and performance. By using static files (HTML, CSS, JavaScript), the frontend can quickly fetch and display data without the overhead of dynamic page generation. This approach is also cost-effective and easy to deploy.
- **Alternative Consideration**: A dynamic single-page application (SPA) using frameworks like React or Vue.js could offer more interactive features but would introduce additional complexity and overhead.

Data Collection and Scheduling

APScheduler:

- **Rationale**: APScheduler is used for scheduling the data collection tasks due to its flexibility and ease of use. It allows for various scheduling options (e.g., interval, cron) and integrates well with Python applications.
- **Alternative Consideration**: Cron jobs could be used for scheduling but would require additional setup and management, particularly in containerized environments.