Summarized Weather Collection Design Documentation

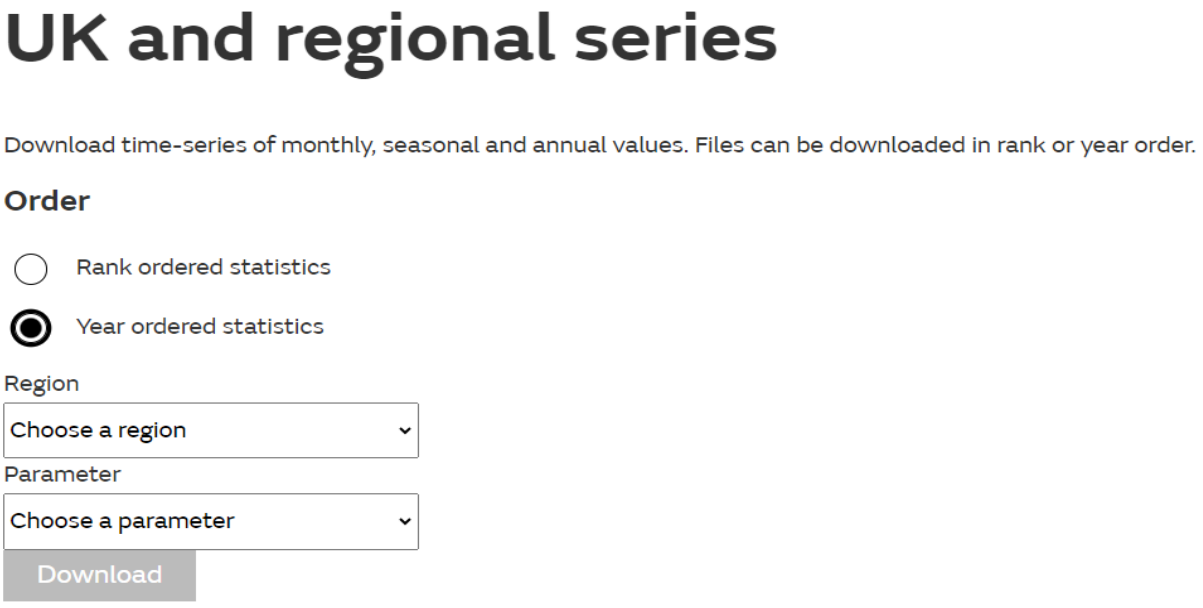
Table of Contents

1. Introduction

This design documentation outlines a project's architecture and components that involve web scraping data from UK MetOffice website and serving it through an API. The project aims to retrieve data from web pages, store it in a database, and expose it through a RESTful API for further consumption. It outlines the key components, database design, scraping method, testing strategy, and potential future improvements.

2. Project Overview

The API aims to serve Summarized Weather data for the past few years. It provides functionalities for getting and deleting weather data for Multiple regions and parameters.



img 1: Screenshot of URL to be scraped

This involves the following key functionalities:

- **Web scraping**: Retrieve data weather data from the UK MetOffice website " https://www.metoffice.gov.uk/research/climate/maps-and-data/uk-and-regional-series#yearOrdered" by parsing HTML content.
  Note: For this application only Year ordered statistics are scrapped.

- **Data storage**: Store the scraped data in a database for efficient retrieval and management.
- **API design**: Expose the scraped weather data through a RESTful API to provide programmatic access to the information.

3. Architecture and Components

The project follows a client-server architecture with the following components:

- **Client**: Initiates requests to scrape data from websites and interacts with the API to consume the retrieved information.
- **Server**: Handles client requests, performs web scraping, stores the data, and serves it through the API.

The server component can be further divided into the following subcomponents:

Web scraping module: Responsible for retrieving data from websites by parsing HTML content. This utilizes the requests library to get static content from the URL.
Data storage module: Handles the storage and retrieval of scraped data. For demonstration purposes Django's default database which is Sqlite3 is used. Although, other databases like PostgreSQL can also be utilized.
API module: Implements the RESTful API endpoints to expose the parsed data. It handles client requests, retrieves data from the storage module, and returns the data in JSON format.

4. Web Scraping

The Problem statement requires scraping data from
"https://www.metoffice.gov.uk/research/climate/maps-and-data/uk-and-regional-series#yearOrdered"
But, the final URL where we get the required data is:
https://www.metoffice.gov.uk/pub/data/weather/uk/climate/datasets/{parameter}/date/{region}.txt
where,
parameter = weather parameter on the basis of which historic data is parsed.
Region = region whose historic data is required
For example:
For, region = UK and parameter = Max Temp
URL looks like ,
https://www.metoffice.gov.uk/pub/data/weather/uk/climate/datasets/Tmax/date/UK.txt

Scraping involves the following steps:

1. Identifying the target website(s) on the basis of selected parameter and region. Scraping specific information.
2. Utilizing requests library in python to parse the HTML content and extract the records.
3. Implementing data cleaning and preprocessing techniques to ensure the scraped data is accurate and consistent.
4. Implementing error handling and retries to handle potential connection issues or server-side errors.
5. Data Storage in SQlite3 database.

5. Data storage

The data storage module is responsible for storing the scraped weather data in a database. For demonstration purposes Django's default database which is Sqlite3 is used.
It involves the following steps:

1. Defining an appropriate Django Model "WeatherData" with fields like:
   region, parameter, year, jan, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec, winter Summer, autmn, spring, annual.

2. Ensuring data integrity and handling potential errors during data storage operations.
   Example: region, parameter and year should be unique in database. To avoid duplicate records

6. API Design

The API module exposes the scraped weather data through a RESTful API, providing programmatic access to the information. It involves the following steps:

6.1. Designing the API endpoints based on the project's requirements:

   a. GET request Historic Region and Parameter specific data
   b. GET request to fetch weather data for specific year, region and weather parameter.
   c. DELETE request to delete weather data for specific year, region and weather parameter.

6.2. Implementing Pagination for endpoint no a. to reduce load on API.

7. Testing Strategy

To ensure the reliability and correctness of the project, following testing strategies are used:

Functional Testing: Test the API endpoints and data retrieval to validate the expected behavior and data accuracy.

8. Future Improvements

While the initial project covers the basic functionalities, several potential improvements can be considered for future iterations, including:

d. Implementing data caching mechanisms to improve performance. Which employs caching data for URL's with mostly used region and parameter thereby reducing the load on the target website.
e. Adding rate limits to API so that number of requests do not exceed than a limit
f. Adding support for handling dynamic content or JavaScript-rendered pages during web scraping.
g. Implementing data validation and integrity checks during the web scraping process to ensure the accuracy of the retrieved information.
h. Integrating with additional external APIs or data sources to enrich the scraped data.

9. Docker Use in Vscode:

1. Clone the repo into your local directory.
2. Install VSCode. We will be using VSCode Dev Containers to set up our environment.
3. Install VSCode Remote Container extension within VSCode.
4. Create a .env file in the location "web_scraping\backend\". The devcontainers will be looking for this file to properly set up your vscode environment.
5. backend folder have its own development container set up. To open your workspace under different this container, open up a New Window for each service you want to work on. Use the command palette (CTRL+SHIFT+P or CMD+SHIFT+P) and use Remote-Containers: Open Folder in Container to open the project directory at "web_scraping\backend\". VSCode should open a new window and build the respective .devcontainer.json for that project
6. Once the container is built, VSCode should be operating within the container with local debug features.