Task Description: Weather Data Analysis and Storage

Background:

You are tasked with creating a Python script that retrieves weather data from a public API, performs some analysis, visualizes the results, and stores the data into a database.

Requirements:

API Integration:

- Use the OpenWeatherMap API (https://openweathermap.org/api) to fetch current weather data for a given city.
- You need to sign up for a free API key.

Data Analysis:

- Extract relevant information such as temperature, humidity, and wind speed from the API response.
- Calculate the average temperature and humidity for the given data.

Data Visualization:

- Create a simple bar chart using a Python plotting library (e.g., Matplotlib or Plotly) to visualize the average temperature and humidity.
- Include labels and a title for the chart.

Database Storage:

- Choose a database (SQLite, MySQL, or any other of your choice).
- Store the weather data (city name, temperature, humidity, wind speed) along with the date and time into the database.

Asynchronous API Requests (10%):

- Modify your script to make asynchronous API requests for fetching weather data from multiple cities concurrently.
- Implement a concurrency library like `asyncio` or `httpx` for asynchronous requests.
- Compare the performance of asynchronous requests versus synchronous requests.

Exception Handling and Logging (10%):

- Implement comprehensive exception handling for various scenarios, logging relevant information using the `logging` module.
- Log errors, warnings, and informational messages with appropriate log levels.

Configurability (10%):

- Extend the configuration file to include parameters such as the list of cities to fetch weather data for and the units (e.g., Celsius or Fahrenheit).
- Ensure that the script is easily configurable without modifying the code.

Historical Weather Data (15%):

- Integrate historical weather data retrieval for a specified date range.
- Allow the user to input a start and end date, and fetch historical weather data for that period.
- Perform additional analysis on historical data, such as trends or anomalies.

Unit Testing Expansion (10%):

- Expand your unit tests to cover a broader range of scenarios, including edge cases and potential failures.
- Use a testing library such as `unittest` or `pytest`.

Additional Notes:

- Provide a configuration file for API key and database connection details. Do not hardcode sensitive information.
- Include error handling for potential issues such as API request failures or database connection errors.
- Write unit tests for at least one critical function in your code.

Submission:

A Python script or multiple scripts that fulfill the requirements.
Configuration file(s) for API key and database connection.
A brief document explaining your approach, design decisions, and any challenges faced.

Evaluation Criteria:

Functionality (50%):

- Successful API integration.
- Accurate data extraction and analysis.
- Correct implementation of data visualization.
- Proper storage of data in the database.

Code Quality and Structure (25%):

- Clean and modular code structure.
- PEP 8 compliance.
- Proper use of functions and comments.

Error Handling (15%):

- Adequate handling of potential errors during API requests, data processing, and database interactions.

Testing (10%):

- Presence of meaningful unit tests.
- Evidence of thorough testing.