Technical Documentation – Aeris Project Backend

1. Overview

The **Aeris – Breathing the Future** project aims to transform scientific air quality data into accessible, interpretable, and meaningful information for society.
The application integrates data from sources such as **NASA TEMPO** and **OpenAQ**, performing processing, cleaning, and distribution through a REST API developed in **Python** using the **FastAPI** framework.
The backend is responsible for the entire data ingestion, processing, and delivery flow, ensuring integrity, security, and scalability for future integration with **machine learning modules** and **frontend applications**.

---

2. System Architecture

The backend architecture was designed to ensure modularity and clarity. The main workflow includes the following stages:

- **Data Collection:** Direct communication with external APIs (OpenAQ and NASA) to obtain measurement data.
- **Processing and Cleaning:** Utilization of libraries such as **Pandas** and **NumPy** for data treatment, standardization, and organization.
- **Temporary Persistence:** Storage of data in local structures (CSV and JSON) to serve the frontend and enable further analysis.
- **Data Exposure:** Use of **FastAPI** to serve REST endpoints with the processed information.

The system operates in the background through independent threads, ensuring that the API remains responsive during data processing.

---

3. Configuration and Initialization

This section describes imports, environment variable loading, and global constant definitions.

3.1 Imports

All required libraries are imported, including `requests` for HTTP requests, `pandas` for data manipulation, `os` for system operations, `json` for file handling, and **FastAPI** for the server.

3.2 Environment Variables

Environment variables are loaded from a `.env` file using **dotenv**. This approach ensures that sensitive credentials, such as API keys, are not hardcoded into the source code.

3.3 Security Constants

The request headers are configured with the API key. The system defines an `X-API-Key` header using the environment variable to authenticate every external data request.

3.4 Global Parameters

Defines the scope and limits of data processing, such as pollutants, record limits, and historical range for time-series analysis.

3.5 Directory Structure

Specifies file paths for saving results, including data directories, time-series outputs, and JSON or CSV files.

---

4. Data Collection Functions

The backend uses two main functions to interact with the air quality API.

fetch_paginated_data

Handles data collection from endpoints that return multiple pages of results. Implements pagination logic and handles common HTTP and connection errors.

generate_empty_output

Creates empty output files in case of data collection failure. This safety function ensures that the server remains stable and prevents unexpected errors in responses.

---

5. Core Processing Logic

5.1 Preparation and Data Collection

Data Verification: Prevents unnecessary recollection by loading existing data from CSV files. If the file does not exist, the collection process starts automatically.

Monitoring Station Retrieval: Identifies active air quality monitoring stations based on predefined parameters.

Measurement Collection: Gathers pollutant data for each location using the `fetch_paginated_data` function.

5.2 Data Transformation and Cleaning

Concatenation and Cleaning: Merges collected data into a single DataFrame, removes irrelevant columns, converts timestamps, and filters invalid records.

Coordinate Enrichment: Adds latitude and longitude data to each measurement to prepare for visualization and analysis.

Raw Data Persistence: Saves the processed DataFrame as a CSV file, serving as a local cache for future executions.

5.3 Time-Series Analysis and Alert Generation

Time-Series Processing: Generates JSON files with daily pollutant averages for each monitored location and parameter.

PM2.5 Alerts: Identifies areas exceeding the WHO-recommended limit and compiles an alert list.

Alert Persistence: Stores alert data in JSON format for API endpoints.

---

6. API Server (FastAPI)

FastAPI is used to asynchronously and securely expose processed data.

- **App Initialization:** Creates the API instance and defines metadata such as title and description for automatic documentation.
- **CORS Middleware:** Enables access from any origin, ensuring integration with frontend applications.
- **Data Loading Function:** Handles file reading and data validation before sending responses.
- **Endpoints:**

    - `/api/status`: Returns the current processing status.
    - `/api/locations`: Returns JSON with alerts and monitoring locations.
    - `/api/timeseries/{city}/{parameter}`: Returns daily historical data for a given pollutant in a specific city.
    - `/map`: Returns the generated interactive map HTML file.

- **Startup Event:** During initialization, the backend runs `run_analysis_and_save_data` in a new thread to automatically start data collection and processing without blocking API requests.

7. Execution

To run the backend:

Create a `.env` file in the project root and add your API key:

```
OPENAQ_API_KEY=YOUR_API_KEY_HERE
```

1.

Run the main script:

```
python your_script.py
```
or, alternatively, with **Uvicorn**:

```
uvicorn your_script:app --host 0.0.0.0 --port 8000
```

2.

The server will start on port **8000** and will be ready to receive requests.

---

8. Security and Best Practices

- API credentials are secured using the `.env` file.
- Exception handling is implemented in all HTTP requests.
- The structure follows **PEP 8** and modularization best practices.

- Execution logs are maintained for audit and failure analysis.

---

9. Conclusion

The **Aeris backend** was developed to ensure performance, stability, and reliability in the collection and distribution of environmental data.
Its modular architecture allows for future expansion, integration with machine learning models, and additional analytical layers.
The solution provides a solid foundation for real-time air quality monitoring and forecasting, aligned with the project's scientific and social goals.