

**APTECH GARDEN CENTER**

**E-project**

**EarthScape**

|  |  |
| --- | --- |
|  |  |

APWA Complex, 1st Floor, Agha Khan 3 Rd, Garden East Saddar Town, Karachi Pakistan.

Developer Guide

**How to Run the Project on Your Local Machine**

This guide provides detailed steps to set up and run the project on your local machine.

**Step 1: Activate the Virtual Environment (Django Folder)**

Before running the project, you need to activate a virtual environment to isolate dependencies.

1. **Open Terminal or Command Prompt**:
   * Navigate to the project directory where your code is located.
2. **Activate the Virtual Environment**:
   * For **Windows**:
   * myenv\Scripts\activate

After running this command, you should see (myenv) at the beginning of your terminal prompt.

* + For **macOS/Linux**:
  + source myenv/bin/activate

Similarly, (myenv) will appear in your terminal prompt to indicate activation.

1. **Verify the Virtual Environment**:
   * Run the command:
   * python --version

Ensure it shows the correct Python version installed in your virtual environment.

**Step 2: Start the Backend (Django)**

The backend handles business logic, APIs, and database interactions.

1. **Navigate to the Backend Directory**:
   * If your backend is in a subdirectory, move to it:
   * cd EarthScape
2. **Apply Database Migrations**:
   * Migrations are necessary to create database tables as defined in your models.py files.
   * python manage.py migrate

This will set up the database schema.

1. **Start the Django Development Server**:
   * Launch the backend server:
   * python manage.py runserver
   * The terminal will display a URL like http://127.0.0.1:8000/.
2. **Test the Backend**:
   * Open your browser and navigate to http://127.0.0.1:8000/ to confirm the backend server is running.
   * If you have APIs, test them using tools like Postman or a browser (if they use GET requests).

**Step 3: Start the Frontend (Python)**

The frontend is responsible for displaying data and handling user interaction.

1. **Navigate to the Frontend Directory**:
   * Move to the directory where the Streamlit application is located:
   * cd EarthScape
2. **Run the Streamlit Application**:
   * Launch the frontend:
   * streamlit run app.py
   * After running this command, Streamlit will provide a URL like http://localhost:8501.
3. **Open the Streamlit App**:
   * Copy the URL provided in the terminal and paste it into your browser to access the frontend interface.

**Step 4: Verify Both Backend and Frontend**

1. **Check the Backend**:
   * Ensure the Django server is running by visiting http://127.0.0.1:8000/.
   * Test any APIs to confirm the backend is working as expected.
2. **Check the Frontend**:
   * Open the Streamlit app at the URL http://localhost:8501.
   * Interact with the frontend to verify it displays data correctly.
3. **Test Backend-Frontend Communication**:
   * Perform actions in the frontend (e.g., submitting forms or fetching data) to ensure it interacts with the backend.

**Step 5: Common Troubleshooting**

If you encounter issues while running the project, check the following:

1. **Virtual Environment Activation**:
   * Ensure (myenv) is displayed in your terminal prompt. If not, activate the virtual environment again.
2. **Missing Dependencies**:
   * Install missing dependencies using:
   * pip install -r requirements.txt
3. **Port Conflict**:
   * If the backend (Django) or frontend (Streamlit) fails to start due to port conflict, specify a different port:
     + For Django:
     + python manage.py runserver 8080
     + For Streamlit:
     + streamlit run app.py --server.port 8502
4. **Database Issues**:
   * If you face migration errors, delete the db.sqlite3 file (if using SQLite) and run migrations again:
   * python manage.py migrate
5. **API Errors**:
   * Use tools like Postman to debug API responses and verify data flow.

**How to Run Python Scripts in the data-science-EarthScape (clean & sort) Folder**

This guide explains how to execute the Python scripts (sort.py and index.py) in the data-science-EarthScape (clean & sort) folder. These scripts are typically used for tasks like data cleaning, sorting, and indexing. Follow the steps below for detailed instructions.

**1. Prerequisites**

Before running the scripts, ensure the following:

1. **Python Installation**:
   * Python (version 3.8 or higher) is installed on your system.
   * Verify the installation by running:
   * python --version
2. **Required Libraries**:
   * Ensure all necessary Python libraries/modules are installed. These dependencies are typically listed in a requirements.txt file.
   * To install them, run:
   * pip install -r requirements.txt
3. **Folder Structure**:
   * Ensure you are in the correct directory where the sort.py and index.py scripts are located:
   * Data-Science-EarthScape (clean & sort)

**2. Running the sort.py Script**

The sort.py script is designed to clean and sort data. It prepares datasets for further processing.

**Steps to Execute:**

1. Open your terminal or command prompt.
2. Navigate to the folder where the script is stored:
3. cd path/to/Data-Science-EarthScape (clean & sort)
4. Run the script:
5. python sort.py

**What the Script Does:**

* **Input**: Reads raw data files (e.g., CSV, Excel) from a predefined directory.
* **Processing**:
  + Cleans the data by removing duplicates, handling missing values, and standardizing formats.
  + Sorts the data based on specified columns (e.g., by date, name, or value).
* **Output**: Saves the cleaned and sorted data to a new file in the same directory or an output folder.

**3. Running the index.py Script**

The index.py script is responsible for creating indexes on the processed data to improve search and retrieval efficiency.

**Steps to Execute:**

1. Ensure that the sort.py script has been run, as index.py may depend on its output.
2. Run the index.py script:
3. python index.py

**What the Script Does:**

* **Input**: Reads the cleaned and sorted data file (output from sort.py).
* **Processing**:
  + Generates indexes on specific fields or columns to optimize query performance.
  + Prepares data for quick lookups, such as plant ID, region, or species.
* **Output**: Saves the indexed data to a new file or updates an existing file.

**4. Verify Script Execution**

After running both scripts, verify their outputs:

1. **Check Output Files**:
   * Locate the output files (e.g., cleaned, sorted, or indexed data).
   * These files are usually saved in a predefined output folder or the working directory.
2. **Inspect Logs**:
   * If the scripts print logs or messages to the console, review them for errors or confirmation of successful execution.

**Example Workflow**

1. Navigate to the project folder:
2. cd path/to/Data-Science-EarthScape (clean & sort)
3. Run the sorting script:
4. python sort.py
5. After successful execution, run the indexing script:
6. python index.py

By following this guide, you can efficiently run and troubleshoot the sort.py and index.py scripts, ensuring that the data is cleaned, sorted, and indexed correctly for your project. Let me know if you need further assistance!

**Your Project Should Now Be Running Successfully**

By following these steps, both the backend and frontend should be operational. Ensure they are working seamlessly together by testing the features and functionality.

Here’s the updated guide without references to folders like model training.ipynb or Cleaned\_EarthScape\_Dataset\_All.xlsx. The dataset is assumed to be in the **same directory** as the model\_training.ipynb file.

**Steps to Load the Dataset**

**1. Place the Dataset in the Same Directory**

Ensure your dataset file Cleaned\_EarthScape\_Dataset\_All.xlsx is saved in the **same directory** as your model\_training.ipynb file. The structure will look like this:

model\_training.ipynb

Cleaned\_EarthScape\_Dataset\_All.xlsx

**2. Update the File Path in Your Code**

Since the dataset is in the same directory as the notebook, you can directly refer to it in your code:

import pandas as pd

# Load the dataset

df = pd.read\_excel('Cleaned\_EarthScape\_Dataset\_All.xlsx')

# Display the first few rows of the dataset

print(df.head())

**3. Verify Dataset Loading**

* Run the cell containing the pd.read\_excel command to ensure the dataset is loaded correctly.
* If successful, it will display the first few rows of the dataset using df.head().

**Common Issues and Solutions**

1. **FileNotFoundError**:
   * Error:
   * FileNotFoundError: [Errno 2] No such file or directory: 'Cleaned\_EarthScape\_Dataset\_All.xlsx'
   * **Solution**:
     + Ensure the file is in the same directory as the notebook.
     + Double-check the filename and its extension.
2. **Missing openpyxl Library**:
   * Error:
   * ImportError: Missing optional dependency 'openpyxl'.
   * **Solution**: Install the openpyxl library:
   * pip install openpyxl
3. **Permission Error**:
   * Error:
   * PermissionError: [Errno 13] Permission denied: 'Cleaned\_EarthScape\_Dataset\_All.xlsx'
   * **Solution**:
     + Ensure the file is not open in another program (like Excel) while running the code.

**Final Setup**

Ensure that both the model\_training.ipynb file and the Cleaned\_EarthScape\_Dataset\_All.xlsx file are in the **same directory**. There’s no need for additional folders or subdirectories. With this setup, the dataset will be accessible directly.

If you encounter any issues, let me know!

**Developer Guide for Your Project**

This document provides an in-depth guide for developers to understand, contribute to, and maintain your project. It covers the project’s purpose, workflow, setup process, and best practices to ensure smooth development and scalability.

**Project Overview**

**Name: Real-Time Monitoring and Feedback Dashboard**

**Purpose:**

This project is designed to monitor real-time metrics (e.g., temperature, humidity) and enable users to set threshold-based alerts. It also features a feedback system for users to report issues or provide suggestions, creating a user-friendly and interactive experience.

**Key Features:**

1. Real-time data visualization and monitoring.
2. Configurable thresholds for generating alerts.
3. Feedback and support system for issue reporting.
4. Integration of backend (Django) and frontend (Streamlit).
5. Dark-themed, responsive, and intuitive user interface.

**Technical Stack**

**Backend:**

* **Framework**: Django (Handles APIs, database, and business logic).
* **Database**: SQLite (easily scalable to PostgreSQL or MySQL).
* **APIs**: RESTful APIs to connect with the frontend.

**Frontend:**

* **Framework**: Streamlit (Builds interactive data visualizations and forms).
* **Libraries**:
  + Plotly (for real-time charts).
  + Requests (for API communication).

**Version Control:**

* Git for managing code versions and collaboration.

**Setup Instructions**

**1. Prerequisites:**

* **Python**: Version 3.9 or higher.
* **Dependencies**: All required packages are listed in the requirements.txt file.
* **Virtual Environment**: Use tools like myenv to isolate dependencies.

**2. Setting Up the Backend (Django):**

* Navigate to the backend directory and perform the following:

1. **Install Dependencies**: Install required Python packages using pip.
2. **Run Migrations**: Apply migrations to initialize the database.
3. **Start Server**: Launch the Django server locally to test APIs.

**3. Setting Up the Frontend (Streamlit):**

* Navigate to the frontend directory and execute the following:

1. **Install Streamlit**: Ensure all frontend dependencies are installed.
2. **Run Streamlit App**: Start the Streamlit application to visualize data and interact with the system.

**Developer Workflow**

**1. Workflow Overview:**

* This project follows a **modular development approach**, separating backend (Django) and frontend (Streamlit) tasks.
* Each developer can work independently on either the backend (e.g., API creation) or the frontend (e.g., chart updates).

**2. Backend Development Workflow:**

**Adding New Features:**

1. Plan the database changes and implement them in models.py.
2. Add necessary endpoints in views.py to expose functionality via APIs.
3. Test APIs thoroughly before integrating with the frontend.

**3. Frontend Development Workflow:**

**Extending Functionality:**

1. Use Streamlit’s built-in components for forms, charts, or interactive elements.
2. Integrate APIs with the frontend for dynamic updates.
3. Test user interactions to ensure smooth functionality.

**Best Practices**

1. **Follow PEP 8**: Adhere to Python’s style guide to maintain readable and consistent code.
2. **Error Handling**: Implement robust error handling in both backend (APIs) and frontend.
3. **Testing**:
   * Unit test backend APIs and database interactions.
   * Test frontend elements for responsiveness and dynamic updates.
4. **Version Control**: Commit frequently with clear messages and use feature branches for new developments.
5. **Documentation**: Keep README.md and inline comments updated for new features.

**Key Components**

**1. Real-Time Monitoring:**

* Displays live metrics (e.g., temperature, humidity) using real-time data fetching.
* Allows users to start and stop monitoring dynamically.

**2. Threshold-Based Alerts:**

* Users set customizable thresholds for alerts (e.g., high temperature).
* System triggers notifications when values exceed thresholds.

**3. Feedback System:**

* A structured form collects user feedback or issues, ensuring efficient resolution by the support team.

**Deployment Instructions**

**1. Containerization:**

* Use Docker to containerize both the backend and frontend for consistent deployment environments.

**2. Database Management:**

* Migrate from SQLite for scalable production use.

**Conclusion**

This guide provides all the necessary instructions for developers to set up, contribute to, and maintain the project. By following the outlined steps and best practices, the development team can ensure a scalable, robust, and user-friendly system.