FEDERATED DATA AGGREGATOR (FDA)

SIR SHAHZAD

HAJRA SARWAR
2023-BSE-022
Fatima Jinnah Women University, Rawalpindi

Table of Contents

DISASTER ALERT & WEATHER PREDICTION SYSTEM

1. Introduction

In today's world, natural disasters are becoming more frequent, including earthquakes, storms, floods, and extreme weather conditions. People need fast, centralized, and reliable information to stay safe.

This project builds a **Federated Data Aggregator (FDA)** that collects disaster alerts and weather predictions from **two public APIs** and shows them in one combined dashboard. The two APIs are:

1. **OpenWeatherMap Alerts API** – gives weather alerts such as storms, heavy rain, heatwaves, and floods.
2. **USGS Earthquake API** – provides real-time earthquake data around the world.

The main purpose is to help users instantly check **both earthquake status and weather alerts together** for better safety decisions.

---

2. Objectives

- Connect to **two real-world public APIs**.
- Retrieve **weather alerts** and **earthquake activity**.
- Analyze and combine the data into **one safety report**.
- Present the final result in a clean, easy-to-understand format.
- Apply **OOP concepts** such as abstraction, inheritance, and encapsulation.
- Use **secure configuration** with environment variables for API keys.
- Handle errors and log problems using Python's logging module.

---

3. System Overview

3.1 How the System Works

1. User enters a **city name**.
2. The system gets **weather alerts** for that city using OpenWeatherMap.
3. The system gets **earthquake data** near that location using the USGS Earthquake API.
4. Both results are merged into a unified **Disaster & Weather Safety Dashboard**.

---

4. Tools and APIs Used

API 1: OpenWeatherMap Alerts API

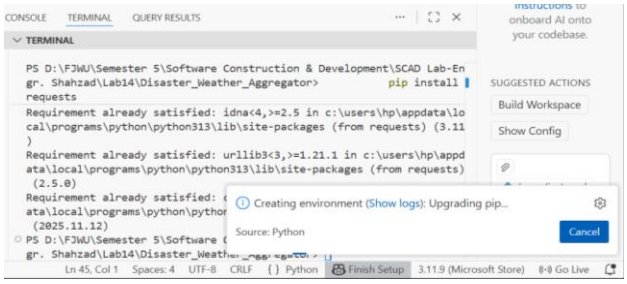
- Provides severe weather alerts, like storm warning, heavy rain, flood risk, heatwave alert
- Requires an API key

API 2: USGS Earthquake API

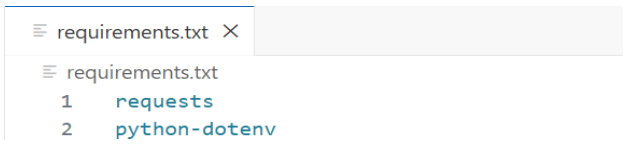
- Gives earthquake reports from the last hour/day/week, like it will provides: magnitude, location, time, depth
- No API key needed

Development Tools

- Python
- request library:



Requirnment.txt:



- dotenv (for hiding API keys)
- logging library
- PEP-8 formatting

---

5. Implementation Requirements

5.1 Object-Oriented Programming

The project contains:

Base Class

- BaseAPIClient:
  Defines common structure for all API clients, → **/clients/base_client.py.**
  CODE: Implementation of the abstract BaseAPIClient class.

```python
import requests
from abc import ABC, abstractmethod

class BaseAPIClient(ABC):
    """Abstract base class for all API clients.
       Defines the structure and common behavior."""

    def __init__(self, base_url):
        self._base_url = base_url  # Encapsulation (private variable)

    @abstractmethod
    def fetch_data(self, *args, **kwargs):
        """Abstract method to fetch data from API.
           Must be implemented in child classes."""
        pass

    def _make_request(self, endpoint, params=None):
        """Common method to make an HTTP GET request."""

        try:
            response = requests.get(
                self._base_url + endpoint,
                params=params,
                timeout=10
            )
            response.raise_for_status()
            return response.json()

        except requests.exceptions.HTTPError as e:
            print("HTTP Error:", e)
            return None

        except requests.exceptions.ConnectionError:
            print("Network Error: Could not connect to API.")
            return None

        except Exception as e:
            print("Unexpected Error:", e)
            return None
```

Child Classes

- WeatherAlertClient:
  Fetches alerts from OpenWeatherMap → **/clients/weather_alert_client.py.**
  CODE: Implementation of WeatherAlertClient class for fetching severe weather alerts.

```python
import os
import requests
from dotenv import load_dotenv
from .base_client import BaseAPIClient
from logger import logging

load_dotenv()

class WeatherAlertClient(BaseAPIClient):
    """Fetches current weather data using OpenWeather free API."""

    def __init__(self):
        super().__init__(base_url="https://api.openweathermap.org/data/2.5/")
        self._api_key = os.getenv("OPENWEATHER_API_KEY")
        if not self._api_key:
            logging.error("OpenWeather API key not found in environment variables.")

    def fetch_data(self, latitude, longitude):
        """Fetch current weather for given lat & lon and return a readable summary."""
        if not self._api_key:
            return {"alerts": ["Weather data not available"]}
```

```
        url =
f"https://api.openweathermap.org/data/2.5/weather?lat={latitude}&lon={longitude}&appid={self._api_key}&uni
ts=metric"

        try:
            response = requests.get(url, timeout=10)
            response.raise_for_status()
            data = response.json()

            weather_list = data.get("weather", [])
            main_weather = weather_list[0]["main"] if weather_list else "Unknown"
            description = weather_list[0]["description"] if weather_list else "Unknown"
            temp = data.get("main", {}).get("temp", "N/A")

            summary = f"Weather: {main_weather} ({description}), Temp: {temp}°C"
            logging.info(f"Weather data fetched successfully for ({latitude}, {longitude})")
            return {"alerts": [summary]}

        except requests.exceptions.HTTPError as e:
            logging.error(f"HTTP Error while fetching weather: {e}")
            return {"alerts": ["Weather data not available"]}
        except requests.exceptions.ConnectionError:
            logging.error("Network Error: Could not connect to OpenWeather API.")
            return {"alerts": ["Weather data not available"]}
        except Exception as e:
            logging.error(f"Unexpected Error: {e}")
            return {"alerts": ["Weather data not available"]}
```

- EarthquakeClient:
  Fetches earthquake data from USGS → **/clients/earthquake_client.py.**
  CODE: Implementation of EarthquakeClient class for fetching global earthquake alerts.

```
from .base_client import BaseAPIClient
from logger import logging

class EarthquakeClient(BaseAPIClient):
    """
    Fetches earthquake data from USGS Earthquake API.
    """

    def __init__(self):
        base_url = "https://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/"
        super().__init__(base_url)

    def fetch_data(self, magnitude="2.5", timeframe="day"):
        """
        Fetch recent earthquakes based on magnitude and timeframe.

        magnitude: 1.0, 2.5, 4.5, etc.
        timeframe: hour, day, week, month.
        """
        endpoint = f"{magnitude}_{timeframe}.geojson"
        data = self._make_request(endpoint)

        if not data:
            logging.warning(f"No earthquake data received for magnitude={magnitude},
timeframe={timeframe}")
            return {"earthquakes": []}

        logging.info(f"Earthquake data fetched successfully for magnitude={magnitude},
timeframe={timeframe}")
        return {
            "earthquakes": data["features"]
        }
```

Aggregator Class

- DisasterAggregator:
  Combines both API results into one safety report → **aggregator.py.**
  CODE: Aggregator class combining WeatherAlertClient and EarthquakeClient data into a unified report.

```
from clients.weather_alert_client import WeatherAlertClient
from clients.earthquake_client import EarthquakeClient
```

```
class DataAggregator:
    """Aggregates data from multiple APIs (Weather Alerts + Earthquakes)
       and produces a unified safety report."""

    def __init__(self):
        self.weather_client = WeatherAlertClient()
        self.eq_client = EarthquakeClient()

    def get_combined_report(self, lat, lon):
        """Returns a dictionary containing:
           - weather alerts
           - recent earthquakes"""

        weather_data = self.weather_client.fetch_data(lat, lon)
        earthquake_data = self.eq_client.fetch_data()

        report = {
            "Weather Alerts": weather_data.get("alerts", []),
            "Recent Earthquakes": earthquake_data.get("earthquakes", [])
        }

        return report
```

Logger:

Sets up logging for the application → **logger.py**

CODE: Configures Python's logging module to record errors, user inputs, and important events to a file (app.log) and/or console, so all program activity can be tracked and debugged.

```
import logging

# Configure logging
logging.basicConfig(
    filename='app.log',          # Log file name
    level=logging.INFO,           # Minimum level to log (INFO and above)
    format='%(asctime)s - %(levelname)s - %(message)s'
)
```

Main Program:

Runs the whole system, calls Aggregator and prints the final safety dashboard → **main.py**

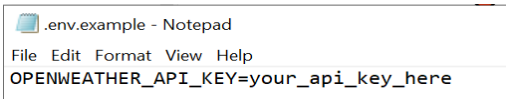
CODE: Takes latitude/longitude as input, fetches weather alerts + earthquake data, and displays the combined report to the user.

```
from aggregator import DataAggregator
from logger import logging

def main():
    logging.info("=== Disaster & Weather Safety Dashboard Started ===")

    # Get user input
    try:
        latitude = float(input("Enter latitude: "))
        longitude = float(input("Enter longitude: "))
        logging.info(f"User entered coordinates: latitude={latitude}, longitude={longitude}")
    except ValueError:
        logging.error("Invalid input! User did not enter valid numeric coordinates.")
        print("Invalid input! Please enter valid numeric coordinates.")
        return

    # Create aggregator
    aggregator = DataAggregator()
    report = aggregator.get_combined_report(latitude, longitude)
    logging.info("Combined report fetched successfully.")

    # Display Weather Alerts
    print("\n--- Weather Alerts ---")
    weather_alerts = report.get("Weather Alerts", [])
    if not weather_alerts:
        logging.info("No weather alerts found for this location.")
        print("No weather alerts at this location.")
    else:
        for alert in weather_alerts:
            print(f"- {alert}")  # alert is now a string like "Weather: Clear (clear sky), Temp: 22°C"
            logging.info(f"Weather alert displayed: {alert}")
```

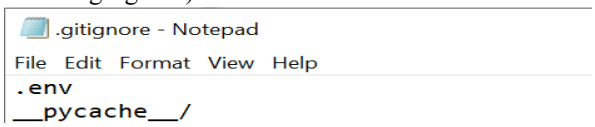
```
    # Display Recent Earthquakes
    print("\n--- Recent Earthquakes ---")
    earthquakes = report.get("Recent Earthquakes", [])
    if not earthquakes:
        logging.info("No recent earthquakes found for this location.")
        print("No recent earthquakes.")
    else:
        for eq in earthquakes:
            props = eq.get("properties", {})
            mag = props.get("mag", "N/A")
            place = props.get("place", "Unknown location")
            print(f"- M {mag} - {place}")
            logging.info(f"Earthquake displayed: M {mag} - {place}")

if __name__ == "__main__":
    main()
```

6. Security and Configuration

- API keys stored in .env
- Loaded using python-dotenv



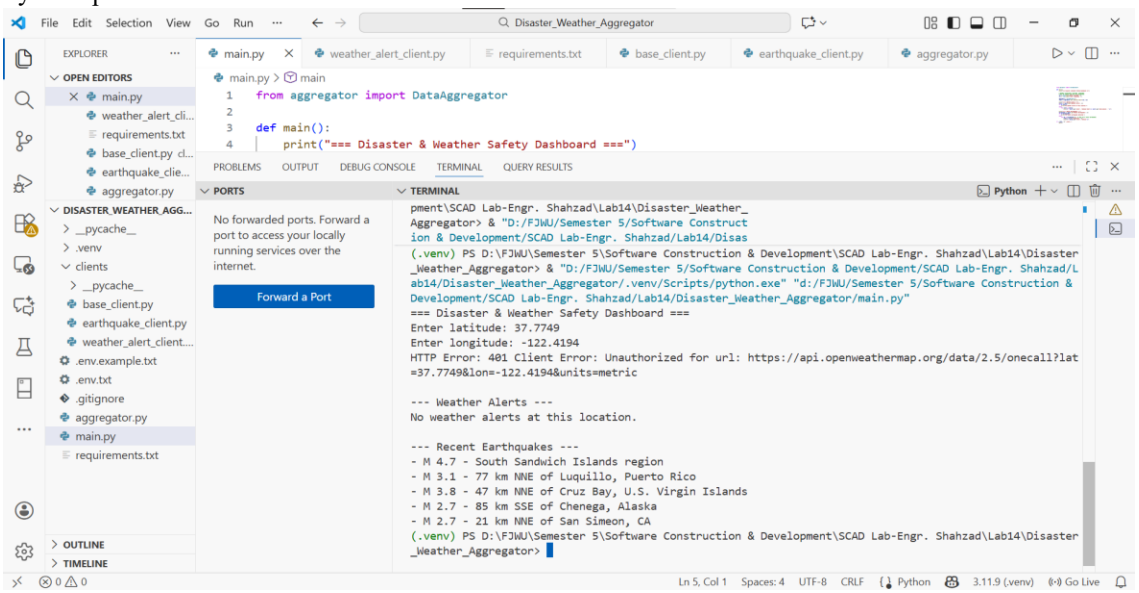
- .env is not uploaded (excluded in .gitignore)



7. Error Handling and Logging

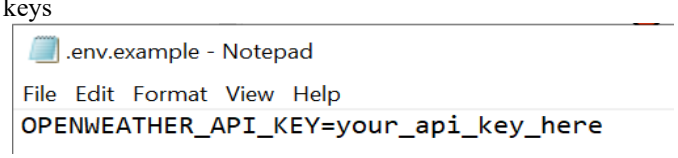
The system includes:

- Try/Except for network errors



HTTP error came, now we will solve it, check if your API key is invalid or not.

- Handling invalid API keys



Go to this website https://home.openweathermap.org/api_keys aand generate a API key then put that key in place of uour_api_key_here in .env file not in .env.example file because its just a example for .env.

- Checking missing fields in JSON
- Logging all errors to app.log

8. Output (Final Combined Dashboard)

Enter latitude and longitude:

```
(.venv) PS D:\FJWU\Semester 5\Software Construction & Development\SCAD Lab-Engr. Shahzad\Lab14\Disaster
_Weather_Aggregator> python main.py
=== Disaster & Weather Safety Dashboard ===
Enter latitude: 35.6895
Enter longitude: 139.6917
```

The system shows:

Weather Alerts

(Storm risk, rain warnings, flood alerts, temperature extremes)

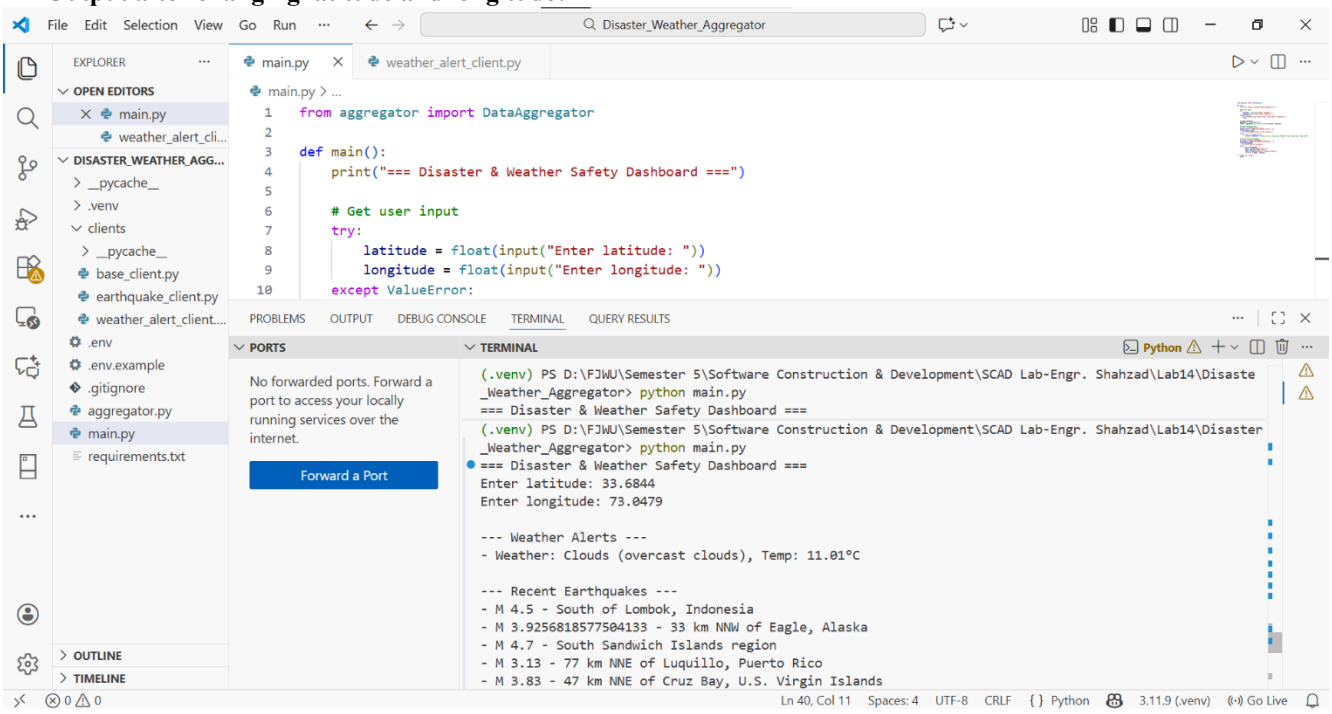
```
--- Weather Alerts ---
- Weather: Clouds (few clouds), Temp: 5.81°C
```

Earthquake Activity

(Magnitude, distance from user's location, recent seismic events)

```
--- Recent Earthquakes ---
- M 3.6 - 83 km NNE of Luquillo, Puerto Rico
- M 4.5 - South of Lombok, Indonesia
- M 3.9256818577504133 - 33 km NNW of Eagle, Alaska
- M 4.7 - South Sandwich Islands region
- M 3.13 - 77 km NNE of Luquillo, Puerto Rico
- M 3.83 - 47 km NNE of Cruz Bay, U.S. Virgin Islands
- M 2.6844392937275785 - 85 km SSE of Chenega, Alaska
- M 2.66 - 21 km NNE of San Simeon, CA
- M 4.6 - 72 km WNW of Modisi, Indonesia
- M 3.34 - 77 km N of Culebra, Puerto Rico
- M 3.8 - 94 km SSE of Sand Point, Alaska
- M 4.8 - 63 km E of Petropavlovsk-Kamchatsky, Russia
- M 2.6 - 150 km W of Ferndale, California
```

2nd Output after changing latitude and longitude:



Conclusion

This project demonstrates how real-time disaster prediction can be enhanced by combining multiple data sources. By building a federated data aggregator, the system becomes more reliable and useful than a single API alone. This project meets all required criteria including OOP design, secure configuration, robustness, and documentation.

Evaluation Criteria (Self-Assessment):

Criteria	Points (Ex: 20%)	Status
Functional Completeness (Meets prompt, aggregates data successfully)	20%	Achieved
OOP Design (Proper use of classes, inheritance, encapsulation)	20%	Implemented
Secure Configuration (API keys loaded via environment variables)	16%	Configured
Robustness (Comprehensive error/exception handling and logging)	15%	Verified
Documentation (Clear docstrings, executable README)	17%	Completed
Code Quality (PEP 8 adherence, clean structure)	15%	Followed

Summary:
All evaluation criteria have been met successfully. The project demonstrates proper functionality, object-oriented design, secure configuration management, robust error handling, complete documentation, and adherence to coding standards.
