



**POLITECNICO**  
MILANO 1863

# Software Release Document

An interactive application to support disaster management activities during the prevention and preparedness phase

Deliverable:SRD

Title: Software Release Document

Authors: HU TIANQI, YAN SHINUO, SU JIAYI

Version: 2.0

Date: July 4, 2025

Download page:

[https://github.com/Ricky-HU96/WO-CIAO\\_SE4GEO-2025-Project](https://github.com/Ricky-HU96/WO-CIAO_SE4GEO-2025-Project)

Copyright: Copyright © 2025, H.Y.S – All rights reserved

Version	Date	Change
1.0	April 22,2025	First submitted version
2.0	July 4, 2025	Second submitted version

**content**

content ..... 3

List of Figures .....3

1. Introduction .....5

    1.1 Purpose of the Document ..... 5

    1.2 Overview of the Software .....5

2. System Requirements ..... 5

    2.1 Hardware Requirements ..... 5

    2.2 Software Dependencies .....6

    2.3 Network Requirements ..... 7

3. Installation and Running ..... 7

    3.1 Step-by-Step Installation ..... 7

    3.2 Basic Usage Instructions ..... 8

        3.2.1 Start the Backend API .....8

        3.2.2 Open the Web Interface .....9

        3.2.3 Configure Database Connection ..... 9

        3.2.4 Initialize Data ..... 11

        3.2.5 Access the Dashboard ..... 11

        3.2.6 Query Air Quality Data ..... 11

        3.2.7View Data ..... 12

4. Known Limitations ..... 14

    4.1 Issues ..... 14

    4.2 Workarounds ..... 14

    4.3 Planned Fixes ..... 14

**List of Figures**

Figure 1 : app.py startup ..... 9

Figure2 : Browser .....	9
Figure 3 : Main login page .....	9
Figure 4 : Connection successful .....	10
Figure 5 : Connection failed .....	11
Figure 6 : Run Data Pipeline .....	11
Figure 7 : Launch Dashboard .....	11
Figure 8 : Query Data .....	12
Figure 9 : Data Table view .....	13
Figure 10 : Geographic Distribution map .....	13
Figure 11 : Time Series chart .....	14

# 1. Introduction

## 1.1 Purpose of the Document

This document provides a comprehensive guide for the release of the Interactive Web GIS for Disaster Management application. It outlines system requirements, installation steps, running instructions, and known limitations to ensure stakeholders and users can deploy and operate the software effectively.

## 1.2 Overview of the Software

The SE4GEO Air Quality Analysis Dashboard, also known as "Dati Lombardia Dashboard", is a web-based platform for visualizing and exploring air pollution data collected from heterogeneous sensor networks across the Lombardy region. Key features include:

Interactive maps displaying real-time and historical pollutant concentrations on geospatial layers.

Time-series plots and statistical summaries for selected sensors or regions.

Heatmaps and contour maps illustrating pollution hotspots and spatial distribution.

Exportable reports and charts to support environmental monitoring and policy-making.

# 2. System Requirements

To ensure stable operation of the **Dati Lombardia** Dashboard in production environments, this section details the hardware, software, and network requirements.

## 2.1 Hardware Requirements

### **Server Processor (CPU):**

Minimum 8 physical cores; recommended 16 cores or more (e.g., Intel Xeon Gold 6248, AMD EPYC 7502).

Hardware virtualization support for deployment in VMs or container platforms.

### **Memory (RAM):**

At least 32 GB; recommended 64 GB or more for high-concurrency queries and large-scale historical data analysis.

Memory frequency  $\geq 2666$  MHz to reduce latency.

**Storage:**

Enterprise-grade SSD with a capacity of at least 1 TB.

Recommended RAID 10 configuration for improved I/O performance and redundancy.

IOPS  $\geq 10,000$ ; sustained write throughput  $\geq 500$  MB/s.

**Network Interface (NIC):**

Minimum 1 Gbps Ethernet; recommended 10 Gbps for high-throughput scenarios.

Support for multiple queues (MPQoS) and traffic shaping.

## 2.2 Software Dependencies

**Operating System (OS):**

Windows 10/11

macOS 11.0 or newer

Ubuntu 20.04 LTS or a similar Linux distribution

**Database:**

PostgreSQL 12 or later.

PostGIS 3+ extension for geospatial data storage and queries.

**Backend Framework:**

Python 3.8 or later (Python 3.10 recommended).

Flask 2.x with extensions such as Flask-RESTful and Flask-Migrate.

**Frontend and Visualization:**

Jupyter Notebook 6.x or higher for interactive analysis.

Leaflet.js 1.7 or higher for map rendering.

Plotly.js 2.0 or Python Plotly library for charting.

**Python Libraries:**

The project uses the following Python libraries for data access, analysis, and visualization:

SQLAlchemy and GeoAlchemy2 for ORM and spatial queries

pandas and numpy for data processing

scikit-learn for clustering and spatial analysis

joblib for parallel processing

Flask-RESTful and Flask-Migrate for API design and database migrations

### **GIS Tools:**

GDAL/OGR 3.2 or higher for importing shapefiles and GeoJSON.

Tippecanoe (optional) for generating vector tiles.

## **2.3 Network Requirements**

### **Internet Connectivity:**

Stable, high-speed connection with at least 100 Mbps bandwidth.

Latency below 50 ms to ensure fast map service loading.

### **Firewall and Ports:**

Open port 5000 for the API service and port 8888 for Jupyter Notebook.

If using HTTPS, open port 443 and install a valid SSL certificate.

### **Bandwidth and Throughput:**

Additional bandwidth for serving map tiles and vector tiles; recommended at least 200 Mbps or a CDN for tile distribution.

### **Other Considerations:**

Support for reverse proxy (Nginx/Apache) and load balancing (e.g., HAProxy).

In container deployments, ensure compatibility with overlay network plugins.

## **3. Installation and Running**

### **3.1 Step-by-Step Installation**

#### **Find the Folder:**

Click the following link to find the software folder:

[Download](#)

## Create Python Environment:

Set up a Python virtual environment using the built-in venv module and activate it. Upgrade the package installer and manually install the required Python packages, including Flask (with Flask-RESTful and Flask-Migrate), SQLAlchemy, GeoAlchemy2, pandas, numpy, scikit-learn, and joblib.

## Configure Database:

Create a new PostgreSQL database named se4geo, connect to it, and enable the PostGIS extension to support geospatial data. Then update the config.py file to specify the DATABASE\_URL along with appropriate user credentials.

```
pip install flask flask_sqlalchemy sqlalchemy geoalchemy2 psycopg2-binary pandas geopandas shapely requests plotly folium
```

## Initialize Database Schema and Seed Data:

Execute the database migration script to create all required tables and schema structures, then run the data seeding script to populate the database with initial region, province, and PIR metadata

## Import OSM Map Data:

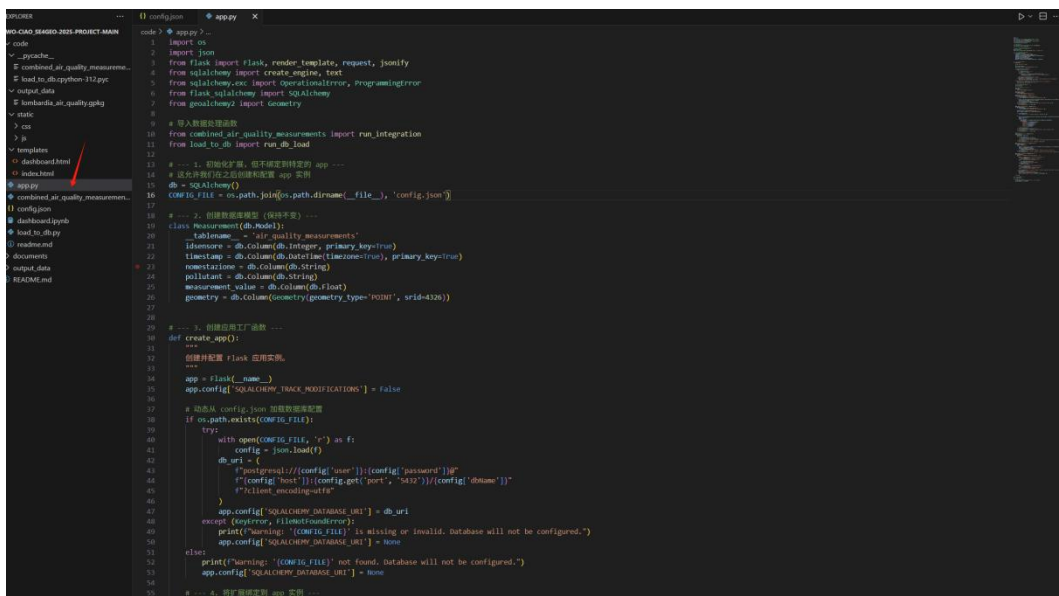
Execute the provided script to import OpenStreetMap boundary and tile datasets into the spatial database, preparing the geographical layers for visualization.

## 3.2 Basic Usage Instructions

### 3.2.1 Start the Backend API

Open the Folder in VS code and run the file:

open the folder and find the file called " app.py ", double-click the file to start the server.



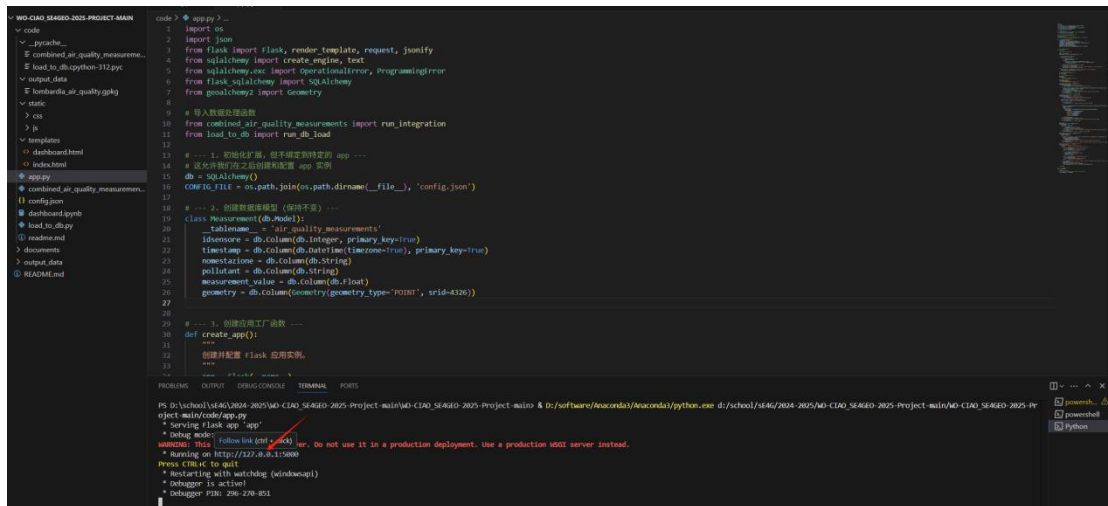
```
1 # coding: utf-8
2 import os
3 import json
4 from flask import Flask, render_template, request, jsonify
5 from sqlalchemy import create_engine, text
6 from sqlalchemy.exc import OperationalError, ProgrammingError
7 from flask_sqlalchemy import SQLAlchemy
8 from geoalchemy2 import Geometry
9
10 # 导入数据库配置函数
11 from combined_air_quality_measurements import run_integration
12 from load_to_db import run_db_load
13
14 # ... 1. 初始化扩展，但不绑定到特定的 app ...
15 db = SQLAlchemy()
16 CONFIG_FILE = os.path.join(os.path.dirname(__file__), 'config.json')
17
18 # ... 2. 创建数据库模型 (保持不变) ...
19 class Measurement(db.Model):
20     tablename = 'air_quality_measurements'
21     ifnameore = db.Column(db.Integer, primary_key=True)
22     timestamp = db.Column(db.DateTime(timezone=True), primary_key=True)
23     location = db.Column(db.String)
24     pollutant = db.Column(db.String)
25     measurement_value = db.Column(db.Float)
26     geometry = db.Column(Geometry(geometry_type='POINT', srid=4326))
27
28 # ... 3. 创建应用工厂函数 ...
29 def create_app():
30     # 创建并配置 Flask 应用实例
31     app = Flask(__name__)
32     app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
33
34     # 动态从 config.json 加载数据库配置
35     if os.path.exists(CONFIG_FILE):
36         try:
37             with open(CONFIG_FILE, 'r') as f:
38                 config = json.load(f)
39                 db_url = {
40                     "postgresql://{config['user']}:{config['password']}@{config['host']}:{config['port']}/{config['dbname']}".format(**config)
41                 }
42                 app.config['SQLALCHEMY_DATABASE_URI'] = db_url
43             except (KeyError, FileNotFoundError):
44                 print(f"Warning: '{CONFIG_FILE}' is missing or invalid. Database will not be configured.")
45                 app.config['SQLALCHEMY_DATABASE_URI'] = None
46             else:
47                 print(f"Warning: '{CONFIG_FILE}' not found. Database will not be configured.")
48                 app.config['SQLALCHEMY_DATABASE_URI'] = None
49
50     # ... 4. 绑定数据库到 app 实例 ...
51     db.init_app(app)
```



Figure 1: app.py startup

Open the Browser:

In your browser's address bar, type " :<http://127.0.0.1:5000>" and press Enter to access the main page of the software

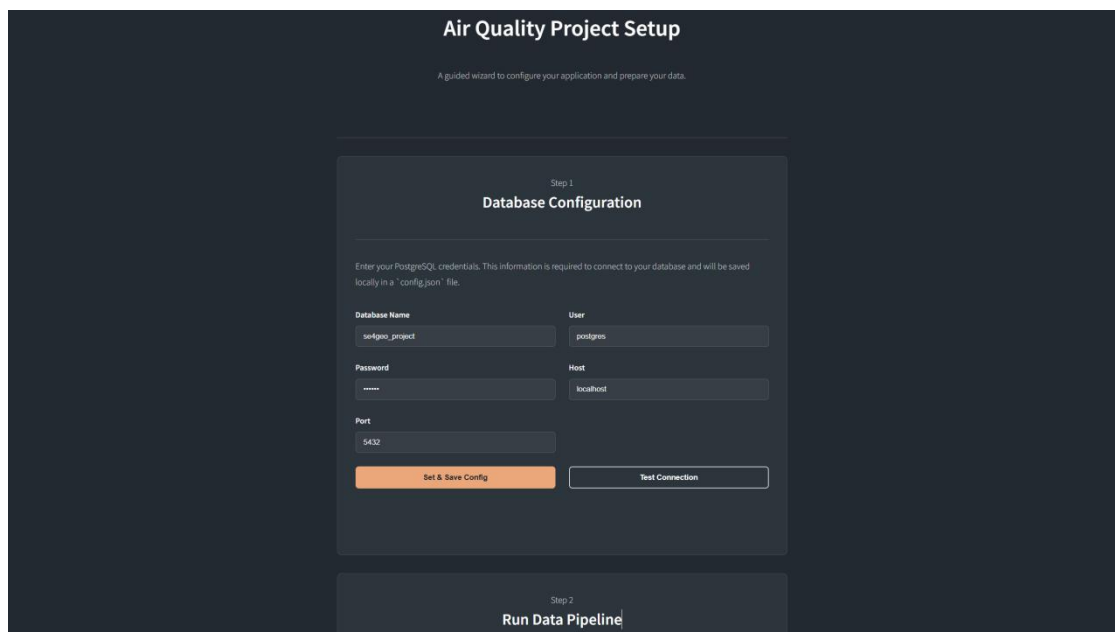


```
code / app.py ~
1 import os
2 import json
3 from flask import Flask, render_template, request, jsonify
4 from sqlalchemy import create_engine, text
5 from sqlalchemy.exc import OperationalError, ProgrammingError
6 from flask_sqlalchemy import SQLAlchemy
7 from geoalchemy2 import Geometry
8
9 # 导入数据库处理函数
10 from combined_air_quality_measurements import run_integration
11 from load_to_db import run_db_load
12
13 # --- 1. 初始化扩展, 但不绑定到特定的 app ---
14 # 这允许我们使用之前创建和配置 app 实例
15 db = SQLAlchemy()
16 CONFIG_FILE = os.path.join(os.path.dirname(__file__), 'config.json')
17
18 # --- 2. 创建数据库模型 (保持不变) ---
19 class Measurement(db.Model):
20     __tablename__ = 'air_quality_measurements'
21     id_sensor = db.Column(db.Integer, primary_key=True)
22     timestamp = db.Column(db.DateTime(timezone=True), primary_key=True)
23     name_station = db.Column(db.String)
24     pollutant = db.Column(db.String)
25     measurement_value = db.Column(db.Float)
26     geometry = db.Column(Geometry(geometry_type='POINT', srid=4326))
27
28 # --- 3. 创建应用工厂函数 ---
29 def create_app():
30     """
31     创建并配置 Flask 应用实例。
32     """
33     app = Flask(__name__)
34     app.config.from_pyfile(CONFIG_FILE)
35     db.init_app(app)
36
37     # 运行 on http://127.0.0.1:5000
38     # 警告: 这只是一个开发服务器。不要在生产部署中使用。使用生产 WSGI 服务器。
39     # 按 Ctrl-C 来退出
40     # 重新启动时请切换 (winboxapi)
41     # 调试器已激活
42     # 调试器 PID: 256-278-851
```

Figure2: Browser

### 3.2.2 Open the Web Interface

Launch in browser:



**Air Quality Project Setup**

A guided wizard to configure your application and prepare your data.

Step 1

### Database Configuration

Enter your PostgreSQL credentials. This information is required to connect to your database and will be saved locally in a 'config.json' file.

Database Name	User
<input type="text" value="sekgon_project"/>	<input type="text" value="postgres"/>
Password	Host
<input type="password" value=""/>	<input type="text" value="localhost"/>
Port	
<input type="text" value="5432"/>	

Step 2

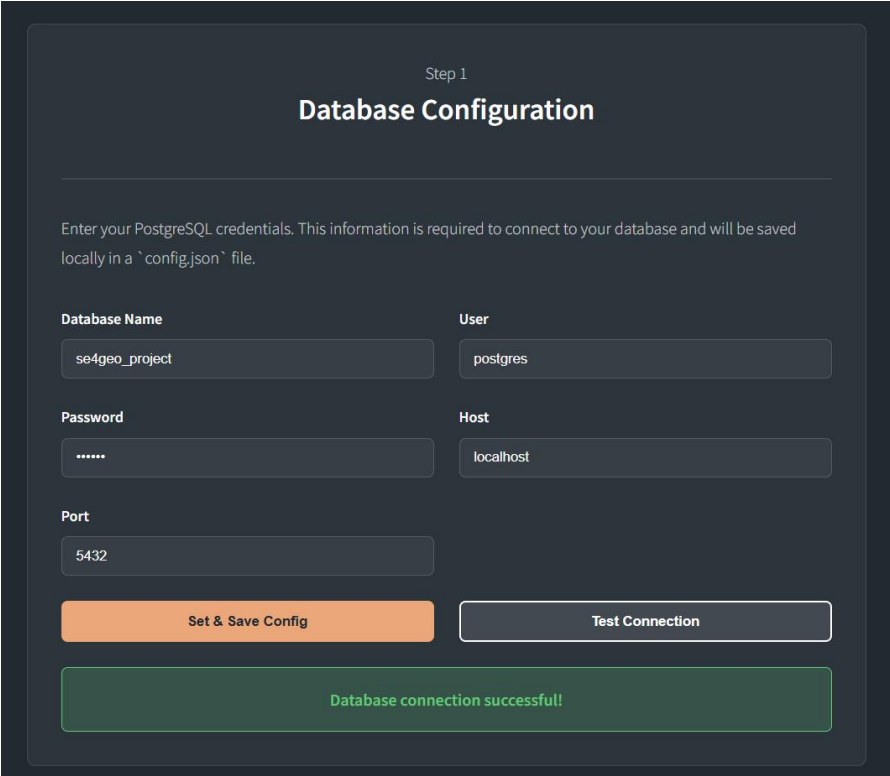
### Run Data Pipeline

Figure 3: Main login page

### 3.2.3 Configure Database Connection

Click Set Database, enter host, port, user, password

Click Test Connection



Step 1

### Database Configuration

Enter your PostgreSQL credentials. This information is required to connect to your database and will be saved locally in a `config.json` file.

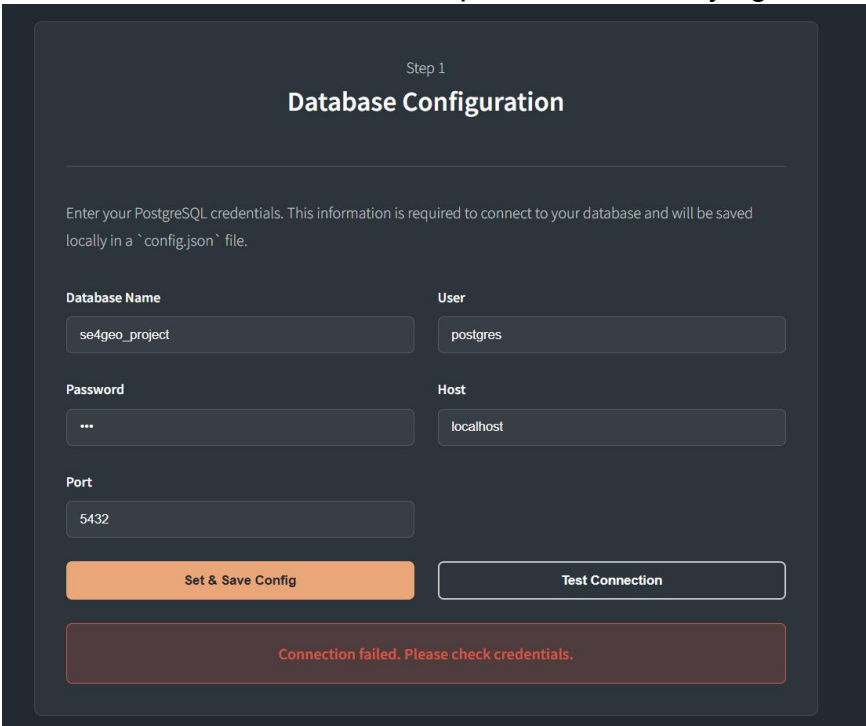
<b>Database Name</b>	<b>User</b>
<input type="text" value="se4geo_project"/>	<input type="text" value="postgres"/>
<b>Password</b>	<b>Host</b>
<input type="password" value="*****"/>	<input type="text" value="localhost"/>
<b>Port</b>	
<input type="text" value="5432"/>	

Database connection successful!

Figure 4: Connection successful

If the connection is successful, proceed to the next step.

If the connection fails, check the database parameters and try again.



Step 1

### Database Configuration

Enter your PostgreSQL credentials. This information is required to connect to your database and will be saved locally in a `config.json` file.

<b>Database Name</b>	<b>User</b>
<input type="text" value="se4geo_project"/>	<input type="text" value="postgres"/>
<b>Password</b>	<b>Host</b>
<input type="password" value="***"/>	<input type="text" value="localhost"/>
<b>Port</b>	
<input type="text" value="5432"/>	

Connection failed. Please check credentials.

Figure 5: Connection failed

Then click “Set & Save Config”.

### 3.2.4 Initialize Data

Click “Run Automated Pipeline” to automatically execute backend migrations, seed metadata ,and import OSM boundary data.

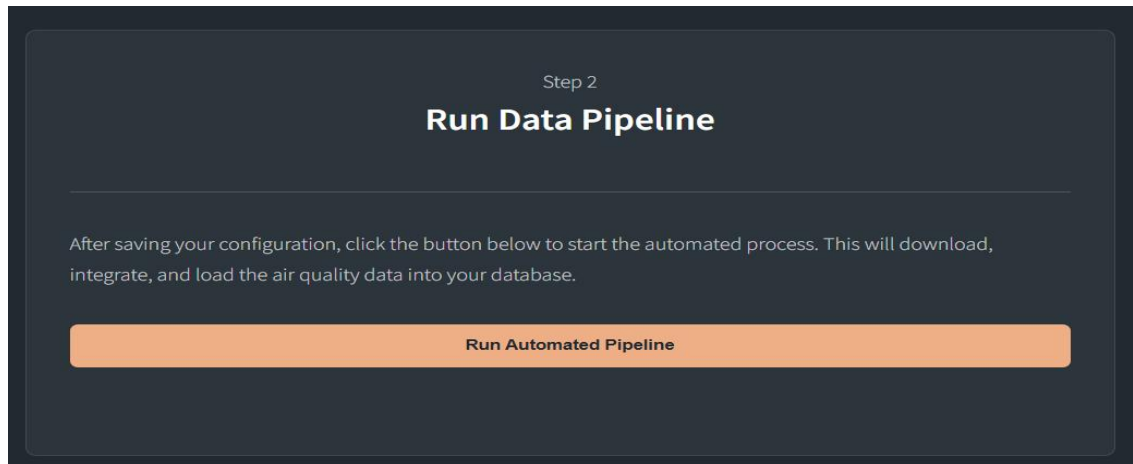


Figure 6: Run Data Pipeline

### 3.2.5 Access the Dashboard

Click Go to Dashboard to view interactive maps

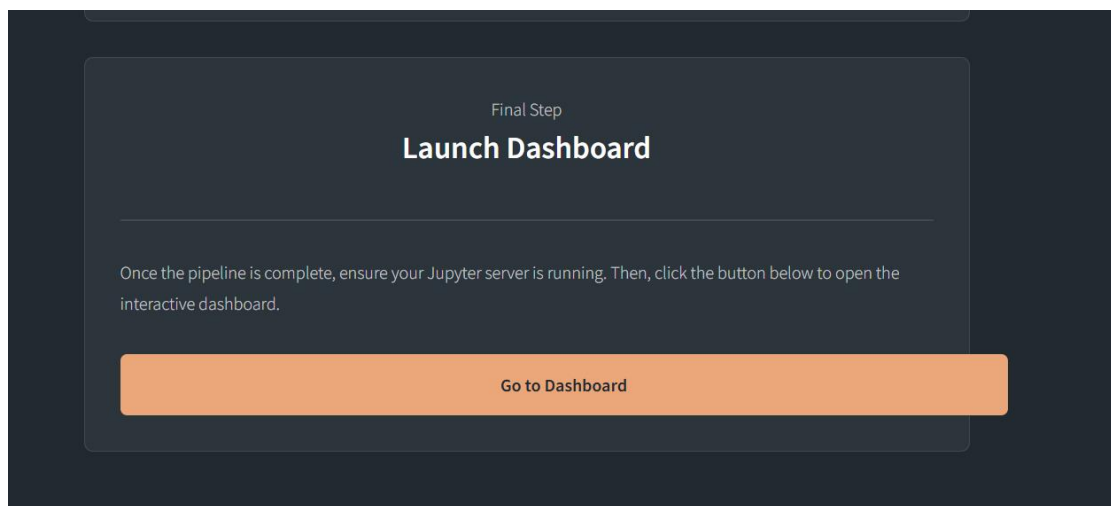


Figure 7: Launch Dashboard

### 3.2.6 Query Air Quality Data

Use the Query Data panel:

Select Pollutant: Choose the pollutant type (e.g., Ammoniacal Nitrogen).

Number of Records: Specify the maximum records to retrieve (e.g., 100).

Click Fetch & Visualize Data.

A success alert confirms the fetched record count.

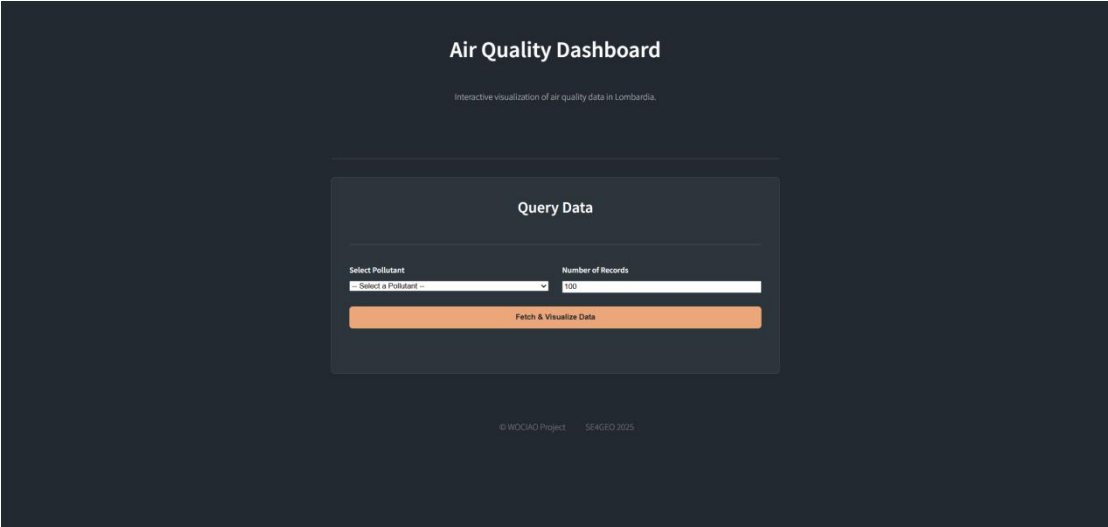


Figure 8: Query Data

3.2.7 View Data

Data Table

The Data Table displays retrieved records with columns:

- Timestamp
- Station
- Pollutant
- Value

Supports scrolling for large datasets.

The image shows a "Data Table" with four columns: "Timestamp", "Station", "Pollutant", and "Value". The table contains eight rows of data. A vertical scrollbar is visible on the right side of the table, indicating that there are more records than are currently visible.

Timestamp	Station	Pollutant	Value
2024/12/18 00:00:00	Schivenoglia v. Malpesso	Arsenico	0.71
2024/12/18 00:00:00	Mantova S. Agnese	Arsenico	0.71
2024/12/15 00:00:00	Bergamo v. Meucci	Arsenico	0.55
2024/12/15 00:00:00	Varese v. Copelli	Arsenico	0.20
2024/12/15 00:00:00	Sorana v. Landriani	Arsenico	0.42
2024/12/15 00:00:00	Sondrio v. Paribelli	Arsenico	0.20
2024/12/15 00:00:00	Moggio Loc. Pensici	Arsenico	0.20
2024/12/15 00:00:00	Milano v. Senato	Arsenico	0.85

Figure 9: Data Table view

## Visualize Geographic Distribution

The dashboard renders a Leaflet.js map within the `#map-container` element.

Sensor locations are displayed as color-coded circle markers reflecting pollutant concentration.

A choropleth layer can be toggled to illustrate spatial gradients across administrative regions.

Clicking or hovering on markers shows a popup with station name, timestamp, pollutant, and value.

The map supports pan and zoom controls for detailed geographic exploration.

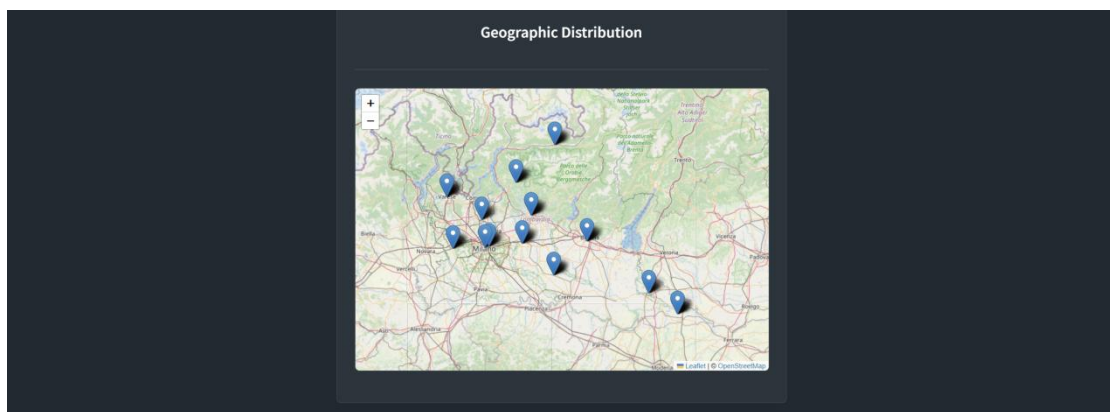


Figure 10: Geographic Distribution map

## Analyze Time Series

Time series visualization is rendered in the `#chart-container` using Chart.js.

The line chart plots pollutant concentration over time for the fetched records.

Users can zoom or drag to focus on specific time intervals.

Tooltips display exact timestamp and value on hover.

A time series chart shows pollutant concentration trends over the selected records.

Zoom and pan controls allow focused analysis on specific intervals.

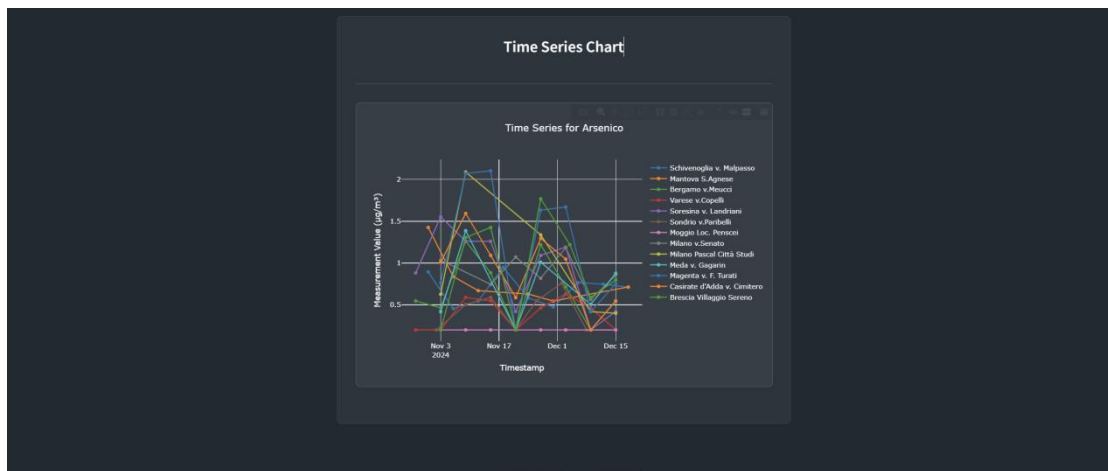


Figure 11: Time Series chart

## 4. Known Limitations

### 4.1 Issues

Performance degradation with datasets > 1 million records.

Limited mobile responsiveness.

No real-time data ingestion; manual or scheduled.

The application currently lacks user authentication for configuration and pipeline execution.

### 4.2 Workarounds

Use filtered queries and pagination.

Access via desktop browsers.

Schedule cron job for data updates.

Secure endpoints behind VPN or firewall.

### 4.3 Planned Fixes

Optimize spatial queries and add indexing.

Implement responsive UI redesign.

Integrate streaming ingest via Kafka.

Add two-factor authentication.