Software Engineering for
Geoinformatics 2025

PolluTrack

Design Document (DD)

Air Pollution Data Display and Custom Graph Generation

By

Viplove Gadkari, Tarig Mohammed Nouraldaim Ahmed

Deliverable: DD- ‘PolluTrack’

Title: Design Document

Authors: Viplove Gadkari, Tarig Mohammed Nouraldaim Ahmed

Version: 2.0

Date: July 3,2025

Link: https://github.com/vidyl911/soft-geo-project

Table of Contents

1. Introduction

Air pollution continues to pose major risks to public health and urban sustainability. PolluTrack is a web-based application designed to increase the accessibility of real-time and historical air quality data to diverse stakeholders. This document builds on the previously established requirements analysis and focuses on the architectural and design elements of the system, with particular attention to usability and maintainability.

1.1 Context and Motivations

Monitoring urban air quality is crucial for ensuring a safe and sustainable living environment. However, raw sensor data is often complex and inaccessible to non-expert users. PolluTrack aims to bridge this gap by offering intuitive tools for exploring pollution trends. Through interactive dashboards and geospatial analytics, the system empowers government officials, researchers, and citizens to understand and act on pollution data in their area.

1.2 Definitions, Acronyms, Abbreviations

- PM2.5, PM10 – Particulate matter (2.5µm and 10µm diameter)
- NO_2, NOx – Nitrogen Dioxide, Nitrogen oxides
- CO – Carbon monoxide
- O_3 – Ozone
- API – Application Programming Interface
- Admin Dashboard Page- Government Dashboard, Government Administrator
- RASD – Requirements Analysis and Specification Document

1.3 Purpose

The purpose of this document is to detail the software architecture and design of the PolluTrack system. The application delivers location-based air quality insights via role-based dashboards, enabling informed decision-making and raising public awareness. It supports data ingestion, visualization, role-specific access, and user interaction through a unified platform.

1.4 Scope and Limitations

This design phase includes the development of PolluTrack's data ingestion pipelines, database schema, backend logic, and frontend interfaces. While the platform offers high usability and real-time capabilities, it is not intended to replace official regulatory monitoring systems. Its accuracy and completeness depend on the underlying sensor network and consistent internet connectivity, which may impose limitations in certain deployments.

2. Architecture Overview

PolluTrack is designed as a modular, end-to-end platform for real-time air quality monitoring and stakeholder-specific visualization. It integrates a Python-powered backend, PostgreSQL database, Folium-based mapping, Plotly visualization and responsive HTML interfaces to support citizens, government officials, and private industry.

2.1 Frontend Layer

This layer contains the user interface components rendered by Flask using **HTML** templates styled with Tailwind CSS. It includes:

- **website_front_page.html**: Public landing page introducing PolluTrack.
- **user_selection.html**: Allows the user to choose a role: **Government**, **Private Sector**, or **Citizen**.
- **login.html / register.html**: Authentication system for privileged users.
- **government_dashboard.html**: Tabular and chart-based pollutant insights for authenticated roles.

- **dashboard_export.html**: Provides downloadable reports and summaries.
- ✅ **sensor_map.html**: Renders an **interactive pollution map using Folium**. The map is **generated server-side in Python**, embedded into the template as HTML/JS, and presented via a container.

This interface is **accessible without login**, enabling citizen engagement and open data transparency.

2.2 Backend Layer (Flask Framework)

The Flask app (main_flask_app.ipynb) acts as the central orchestrator:

- Handles URL routing and user authentication.
- Renders HTML templates using Jinja2.
- Serves pre-rendered Folium maps to sensor_map.html.
- Provides RESTful endpoints for time series data, station metadata, and exports.
- Manages user sessions and secure login handling.

2.3 Data Processing Layer

Implemented via the Jupyter notebook Database_final [data processing].ipynb, this module:

- Loads raw sensor and metadata CSVs.
- Filters inactive sensors based on DataStop.
- Joins sensor measurements with station metadata.
- Cleans pollutant readings and formats timestamps.
- Outputs two key datasets:
    - sensors_data_lombardia_final.csv: Filtered, cleaned air quality data for active sensors.
    - distinct_sensor_info_lombardia_final.csv: Unique metadata (station names, coordinates, IDs) for valid sensors.

These files are used for both database population and map rendering.

2.4 Database Layer (PostgreSQL + pgAdmin)

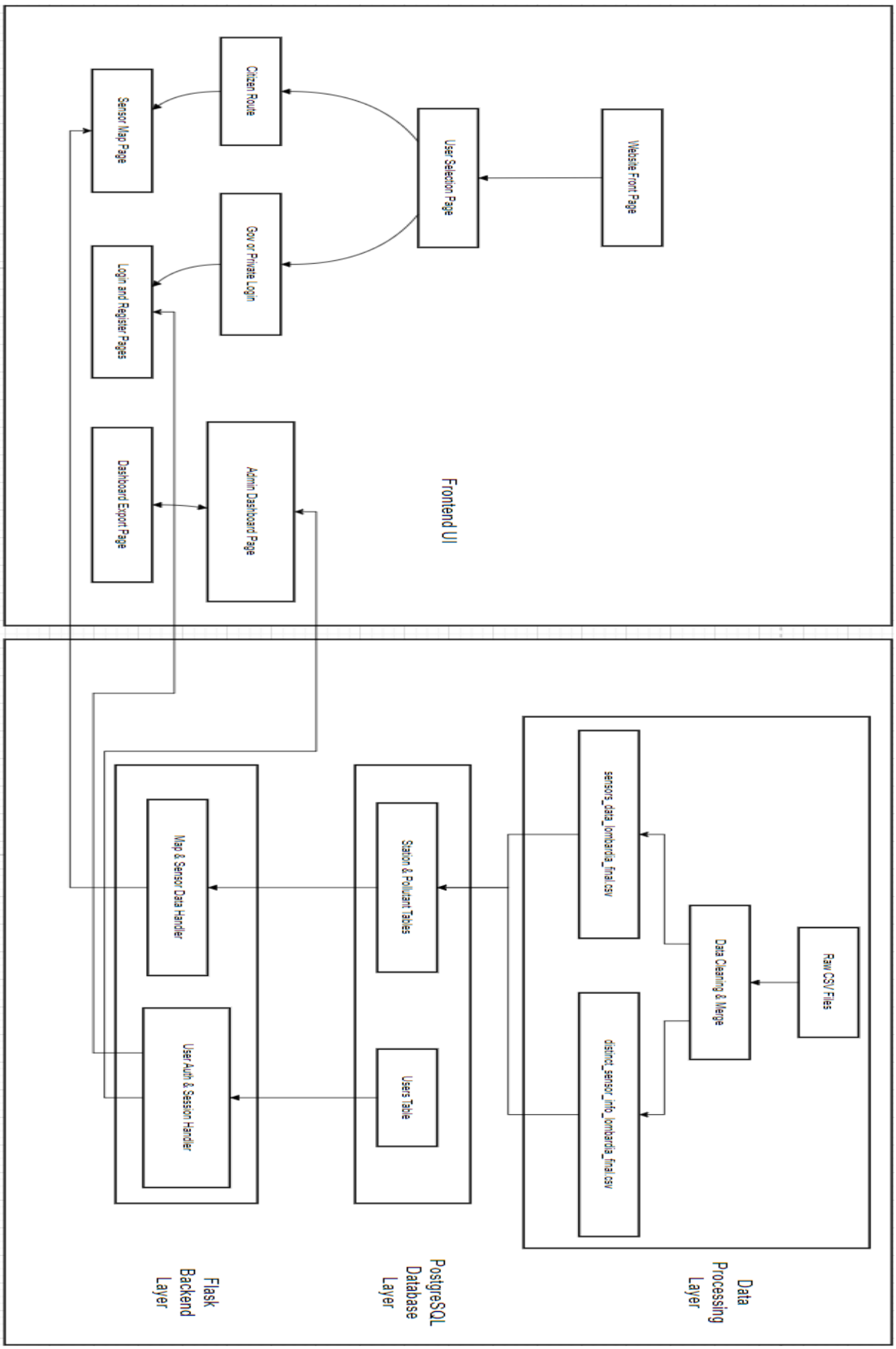
Database logic is implemented in DB_Connection [pgadmin connection].ipynb. The PostgreSQL schema includes:

- users: User credentials and roles.
- stations: Station metadata (ID, coordinates, region).
- pollutants: Time-indexed sensor readings.

Connections are handled using **psycopg2** and **SQLAlchemy**, enabling smooth integration with both Jupyter Notebooks and Flask.

2.5 System Workflow

1. User visits **website_front_page.html** → selects role via user_selection.html.
2. **Citizens** directly access the **pollution map** (sensor_map.html), rendered via Folium.
3. **Privileged users** log in → access dashboards and export tools.
4. Flask serves all pages and injects real-time data or visualizations.
5. The backend connects to PostgreSQL to retrieve sensor readings and metadata.
6. Map and dashboard views reflect updated data on each page load or API call.


Website Front Page
User Selection Page
Citizen Route
Gov or Private Login
Sensor Map Page
Login and Register Pages
Dashboard Export Page
Admin Dashboard Page
Frontend UI
Raw CSV Files
Data Cleaning & Merge
sensors_data_lombardia_final.csv
distinct_sensor_info_lombardia_final.csv
Data Processing Layer
Station & Pollutant Tables
Users Table
PostgreSQL Database Layer
Map & Sensor Data Handler
User Auth & Session Handler
Flask Backend Layer

3. Data Processing

This module is responsible for preparing clean, validated datasets to support both database ingestion and backend analytical operations. The processing logic is implemented using Python (via the Jupyter notebook Database_final [data processing].ipynb) and the pandas library for structured data manipulation.

3.1 Input Datasets

The following raw CSV files are used:

- air_quality_sensors_data.csv: Time series pollutant readings for each sensor.
- air_quality_stations.csv: Metadata including sensor type, station name, coordinates, operational period, and status.

3.2 Sensor Filtering Based on Status

The air_quality_stations.csv file includes a DataStop field. Sensors with DataStop = 'Present' are considered currently active. The dataset is filtered to retain only these operational sensors, and this list is cross-referenced with the sensor readings file to discard values from inactive or outdated sensors.

3.3 Time Window Filtering

To ensure data relevance, only measurements collected from Jan 1, 2022 to Jan 1, 2025 are retained. Datetime fields are parsed using pandas.to_datetime() to standardize timestamps across datasets.

3.4 Data Join and Cleaning

- The filtered sensors and stations are joined using sensor_id.
- Redundant columns (Quota, Comune, Storico, etc.) and unneeded fields (Stato, idOperatore) are dropped.
- Key columns are renamed for consistency, e.g.,:
    - idSensore → sensor_id
    - NomeTipoSensore → sensor_type
    - Valore → value
- Timestamps are converted to datetime objects for sorting and indexing.

3.5 Value and Type Cleaning

- Specific pollutant labels are standardized:
    - "PM10 (SM2005)" → "PM10"
    - "Biossido di Azoto" → "Nitrogen Dioxide (NO_2)"
    - "Monossido di Carbonio" → "Carbon Monoxide (CO)", etc.
- Records with invalid values (-9999) or out of scope pollutant types (e.g., "Benzene") are removed.

3.6 Output Datasets

Two processed and cleaned CSVs are produced:

- sensors_data_lombardia_final.csv: The master dataset containing joined, cleaned, and filtered readings for valid sensors in the Lombardy region.
- distinct_sensor_info_lombardia_final.csv: A deduplicated dataset capturing the last known value per sensor, enriched with air quality classifications (Good / Moderate / Unhealthy).

4. Database Connection

The PolluTrack application uses a PostgreSQL relational database to store processed air quality data and support analytical queries. The connection logic and table creation workflows are implemented in the companion notebook titled DB_Connection [pgadmin connection].ipynb, leveraging psycopg2 and SQLAlchemy for Python integration.

4.1 Database Setup

The PostgreSQL database is initialized and managed through pgAdmin, with the actual Python connection established using SQLAlchemy's create_engine() wrapper. Tables are created to organize the application's core datasets:

- users: Authentication and role management
- sensors_data_lombardia_final: Time series pollution data per sensor
- distinct_sensor_info_lombardia_final: Aggregated and labeled metadata for visualizations

The database is hosted locally on localhost:5432, using the schema name sensor_data_lombardia. Connection credentials (e.g., username/password) are securely passed via SQLAlchemy engine string and can be modified per developer environment.

4.2 Connection Implementation

The Python-PostgreSQL link is established using:

```
engine =
sa.create_engine('postgresql://postgres:<password>@localhost:5432/sensor_data_lom
bardia')

con = engine.connect()
```

The psycopg2 driver is used under the hood by SQLAlchemy for low-level communication. Once connected, pandas DataFrames are directly uploaded using to_sql():

python

CopyEdit

sensor_data.to_sql('sensors_data_lombardia_final', engine, if_exists='replace', index=False)

This enables seamless ingestion of large datasets processed in Jupyter into the PostgreSQL backend.

4.3 Insertion Workflow and Table Handling

The insertion process includes:

- Validation: Only cleaned, deduplicated, and type-checked DataFrames are inserted.
- Overwrite Protection: Existing tables are replaced (if_exists='replace') for development. In production, this would be changed to append or conditional insertions.
- Duplicate Checks: While not fully enforced at the database schema level, the notebook logic ensures each sensor_id/timestamp pair is unique before upload.

Each table is used as a backend source for frontend visualization and API delivery via the Flask application.

4.4 Role in Application Architecture

This database layer provides structured, query-efficient access to environmental data. It acts as the central data source for:

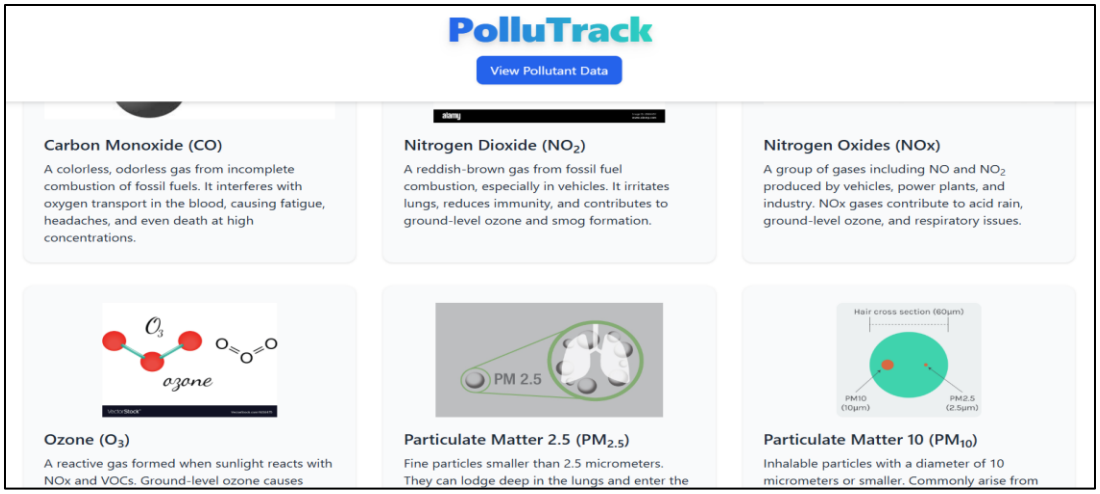
- Historical trend charts (from sensors_data_lombardia_final)
- Aggregated summaries per station (via distinct_sensor_info_lombardia_final)

The modular architecture also allows future integration with cloud-hosted DBMS (e.g., AWS RDS or Azure PostgreSQL) by simply updating the connection string.

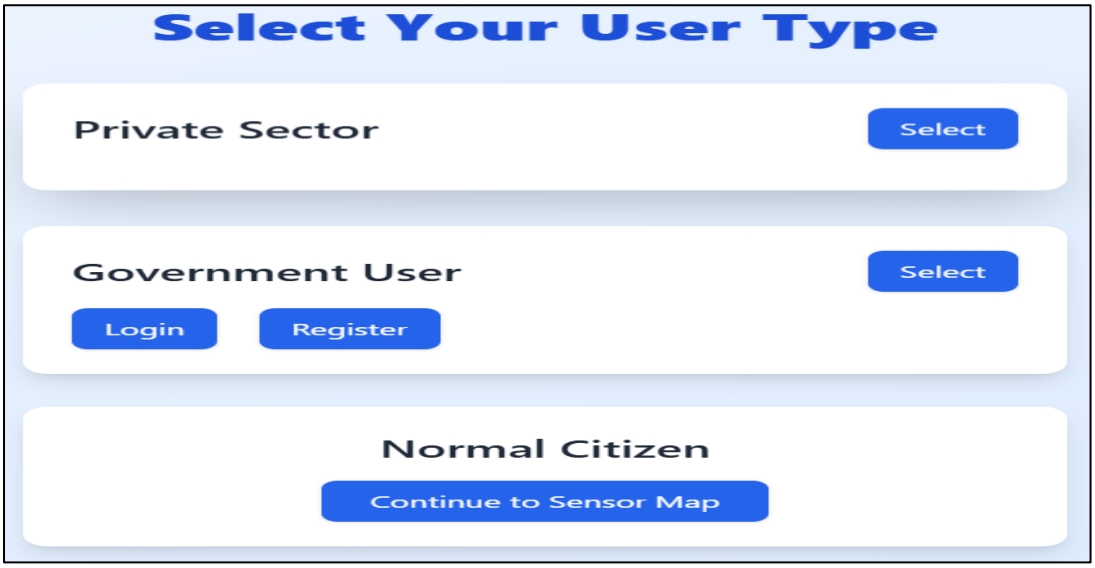
5. WebPages Design

PolluTrack's frontend interface supports three roles: Government, Private Entity, and Citizen. Each role experiences a customized flow through the application, served through a sequence of interactive pages. Below is a detailed breakdown:

- website_front_page.html
  This is the landing page of PolluTrack.
    - It introduces the application with a centered title ("PolluTrack") and a prominent "View Pollutant Data" button.
    - The button redirects users to the role-selection interface (/user-selection).
    - A short awareness section explains why tracking pollution is critical.
    - A second section visually presents key pollutants like CO, NO_2, NOx, O_3, PM2.5, and PM10 with their impacts and images.



Website Front Page

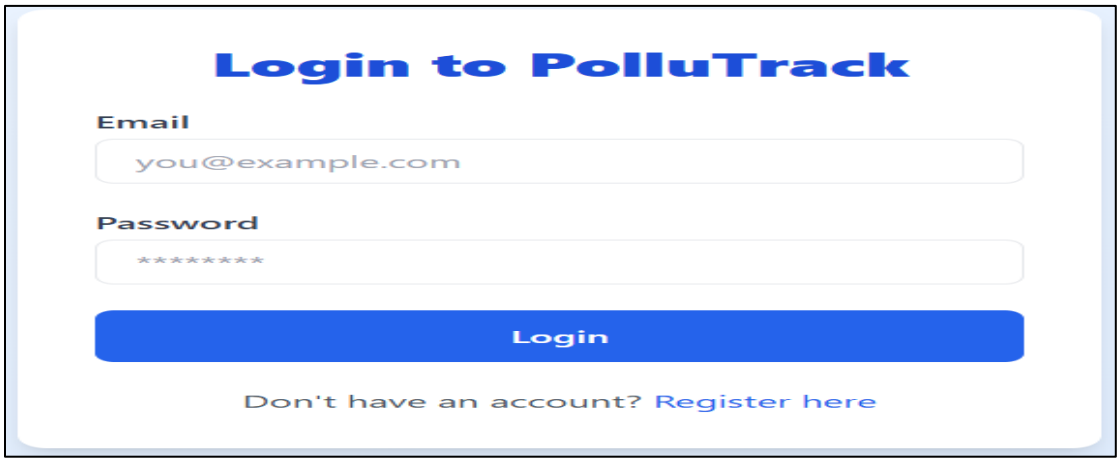


User Selection Page

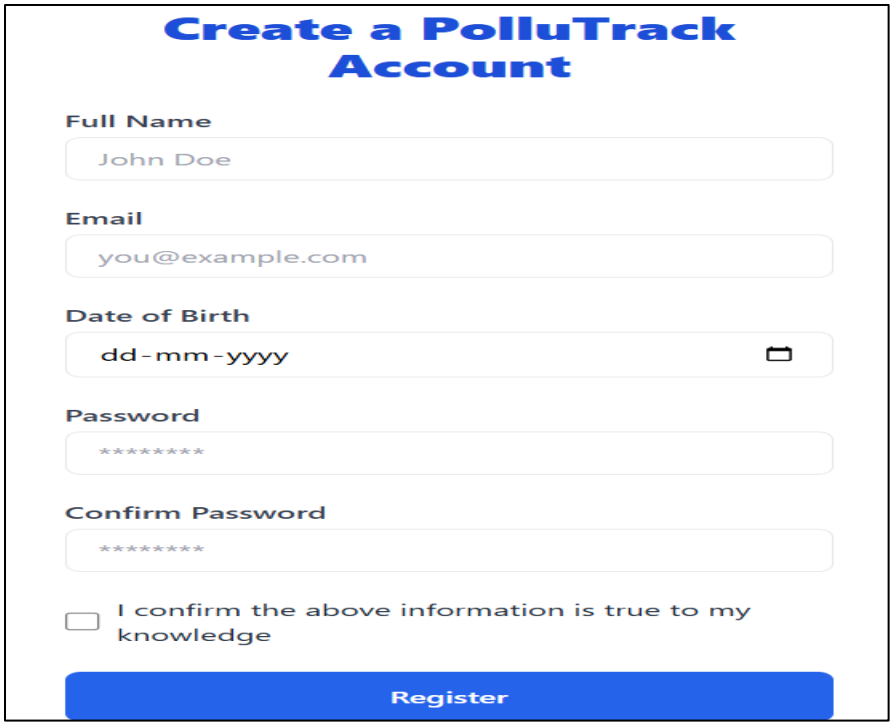
- user_selection.html
  This is the role selection interface offering three blocks:

    - Private Sector and Government Users: Both roles get “Login” and “Register” buttons upon selection, implemented with toggleable sections.

    - Normal Citizen: A direct “Continue to Sensor Map” button is shown, bypassing authentication and linking to sensor_map.



Login Page



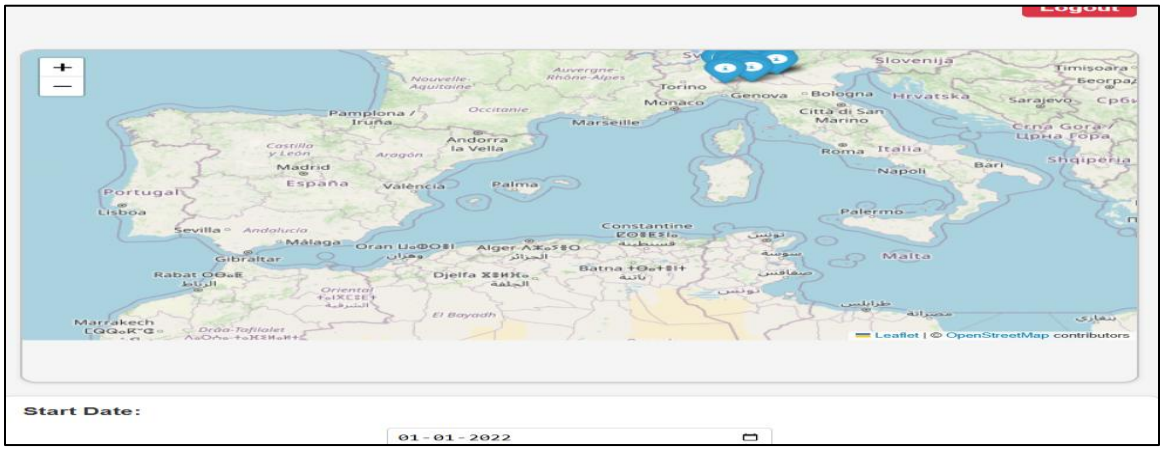
Register Page

- login.html and register.html
    - Both pages present clean TailwindCSS-based forms.
    - Used by Private and Government users for authentication
    - They POST credentials to backend Flask routes and include fields for username, email, and password with confirm password validation in registration.

government_dashboard.html

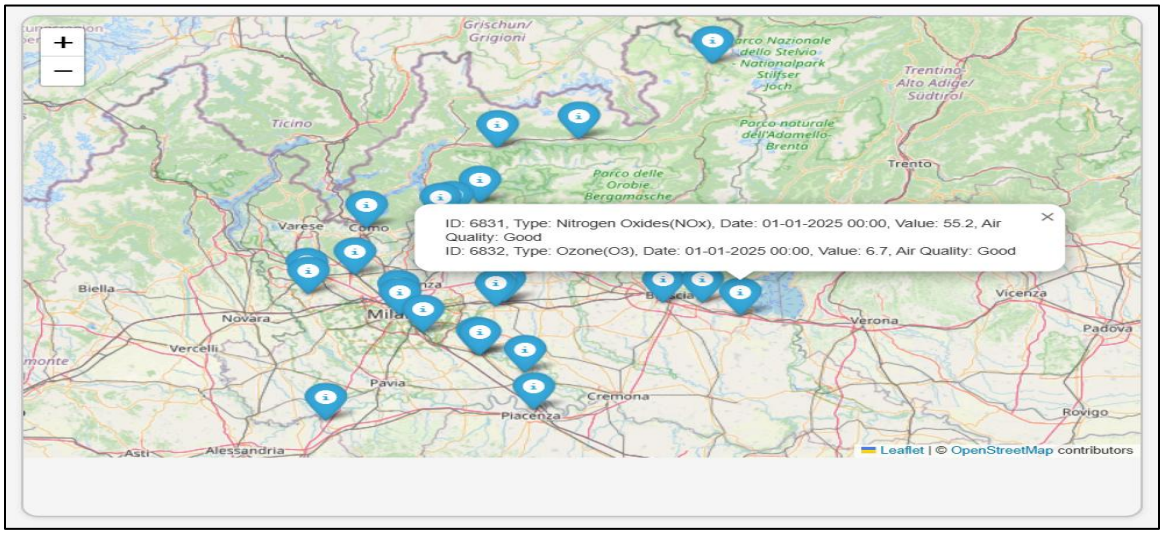
- Accessible only after login by Government users.
- Shows data visualizations using charting libraries (likely Chart.js or similar) embedded in <canvas> tags.
- Provides filter options (e.g., pollutant, date) and includes styling for clarity.
- Offers an “Export CSV” option that links to dashboard_export.

Government Dashboard



Dashboard Export

- dashboard_export.html
    - A minimal page that enables downloading sensor data as a CSV file.
    - Accessible only after login by Government or Private users.
    - This download is triggered automatically via a Flask backend route when accessed, providing filtered/exported data.
- sensor_map.html
    - Embedded map using Folium, injected via Jinja's {{ map_html|safe }}.
    - Citizens can visually explore pollutant levels through station markers.
    - Designed for public awareness and does not require login.



Sensor Map

6. Application Backend Architecture

PolluTrack's backend is implemented using Flask, offering a lightweight yet powerful web framework that supports routing, templating, session management, and database interaction. The backend serves as the core logic layer, coordinating between the UI, data analytics, and PostgreSQL database. The main backend logic resides in main_flask_app.ipynb.

6.1 Technology Stack

- Flask: Web routing, session handling, template rendering.
- psycopg2: Direct PostgreSQL database interaction (e.g., for custom SQL queries).
- SQLAlchemy: ORM layer for structured data handling.
- Pandas: Data wrangling and transformation.
- Folium: Geo-visualization used in sensor map rendering.
- Plotly: Interactive graph rendering for the government dashboard.

6.2 Functional Modules in Backend

- Routing Handlers: Define navigation paths for login, register, dashboard, export, map, and role-selection pages.
- Session Management: Uses Flask's built-in session dictionary to track logged-in users and their roles.
- Database Queries: Performs CRUD operations on users, sensors, and pollutant-related tables via psycopg2/SQLAlchemy.

- Dynamic Page Rendering: Injects context-aware information into Jinja2 templates for customized user views.

6.3 Notebook-Driven Architecture

The main_flask_app.ipynb file drives the Flask app using code blocks that simulate modular app.py behavior. It simplifies rapid prototyping and visual testing of request flows while maintaining clarity for non-developer stakeholders.

6.4 REST API Details

Endpoint	HTTP Method(s)	Description
/	GET	Landing page (redirects to front page)
/front-page	GET	Displays the main website front page
/user-selection	GET, POST	Handles role-based user redirection
/sensor-map	GET	Displays real-time sensor map for citizen users
/government_dashboard	GET	Government dashboard with pollutant graphs and filters
/kpi_analysis	GET, POST	Generates KPI visualizations using Plotly
/export-data	POST	Allows government/private users to export sensor data
/login	GET, POST	Authenticates user credentials
/register	GET, POST	Registers a new user
/logout	GET	Logs out the current user and ends session

7. User Authentication & Role-Based Flow

PolluTrack supports a role-based authentication system with three types of users: Citizen, Private Entity, and Government Administrator. While private and government users must log in to access dashboards and protected views, citizens can access sensor information without login, offering an open and user-friendly experience for the general public.

The core logic is implemented in main_flask_app.ipynb, leveraging Flask, sessions, and PostgreSQL for secure access control.

7.1 Registration Logic

- Endpoint: /register
- Supports registration of only Private and Government users.
- On form submission, it checks and creates the users table (if not existing) with:
    - username, password, and role.
- Passwords are stored in plaintext for simplicity (⚠️ not secure for production systems).

🟢 Citizens do not register. They are redirected directly to the sensor map interface without authentication when they click on ‘Continue to Sensor Map’ button.

7.2 Login Workflow

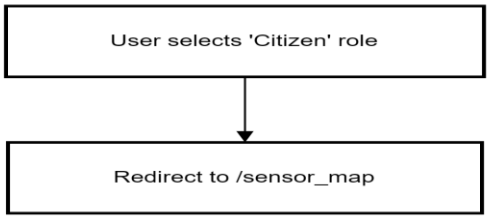
- Endpoint: /login
- Only Private and Government users log in via this form.
- After validating credentials against the users table:
    - A session is created with the user's role.
    - Role determines redirection as follows:
        - Citizen → Bypasses login, directly routed to /sensor_map.
        - Private → Redirected to /dashboard_export.
        - Government → Redirected to /government_dashboard.

7.3 Session & Security Handling

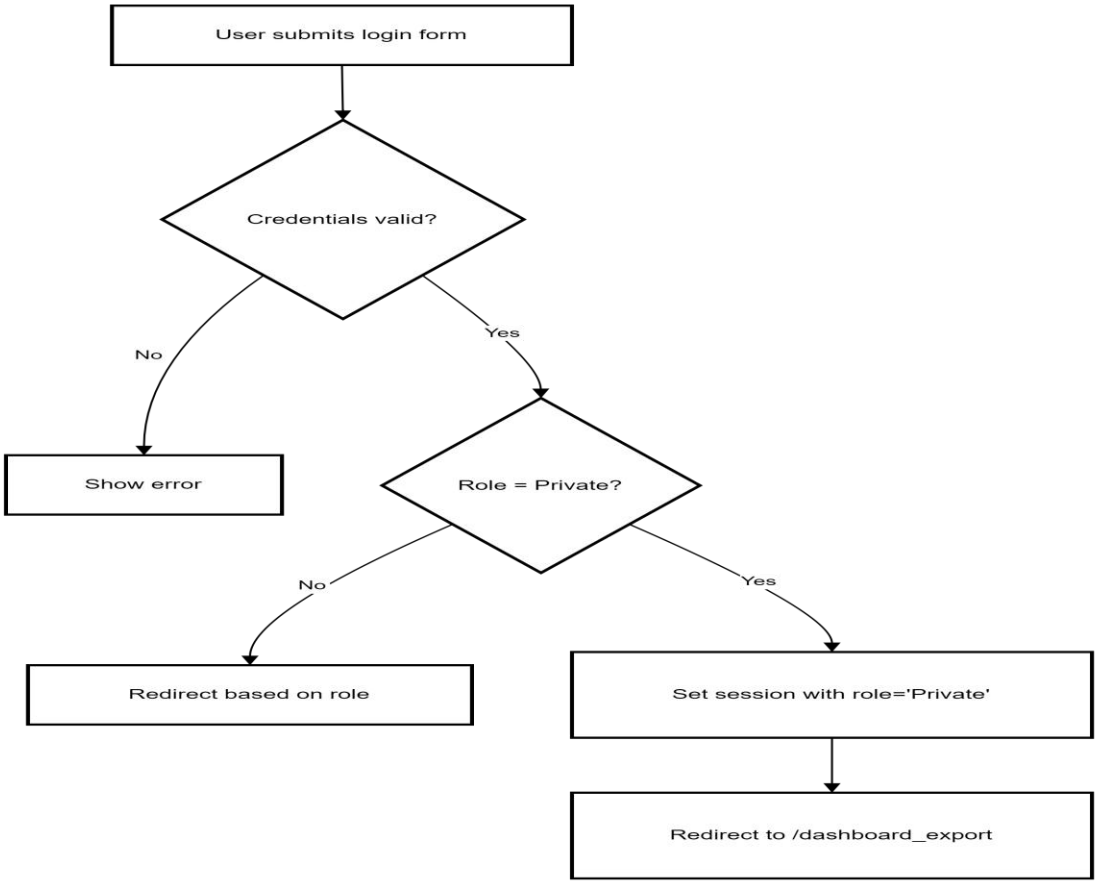
- Flask session stores username and role.
- Pages check session values to allow or restrict access.
- Any unauthorized or mismatched access results in redirection to /login.

🔒 Citizen-facing map is public. Government and Private dashboards are strictly session-gated.

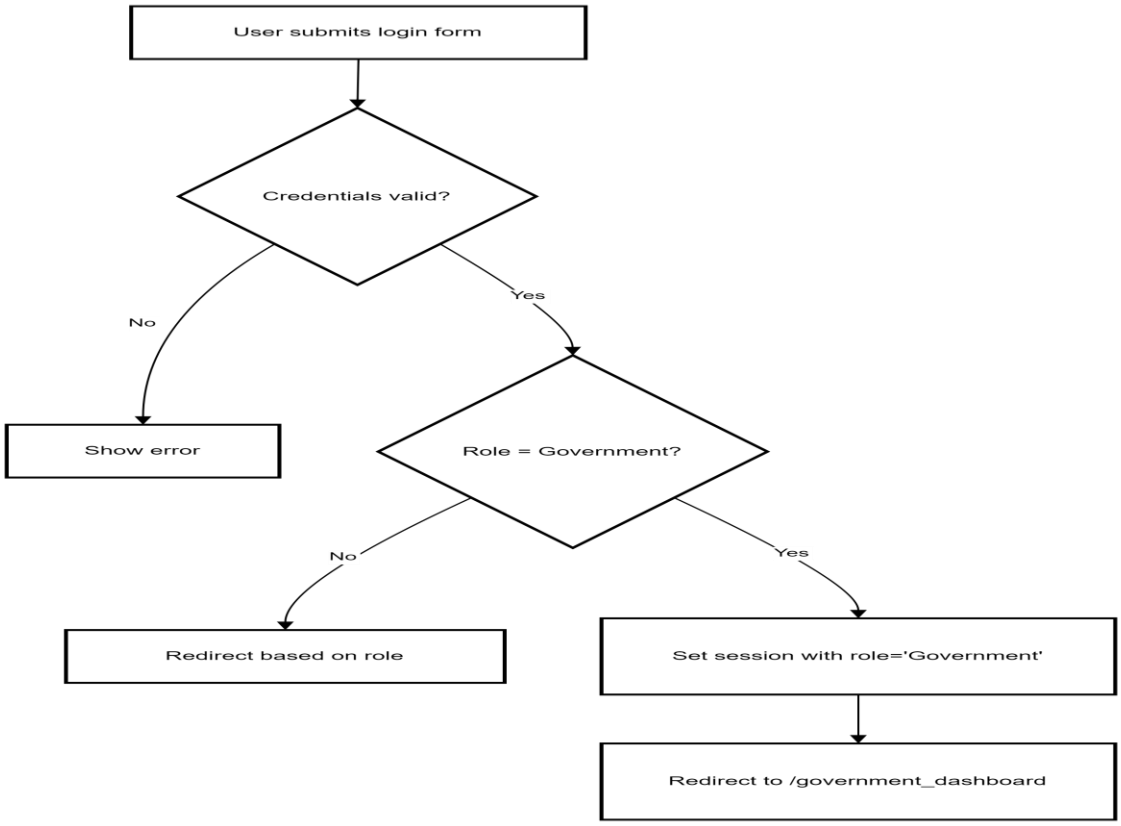
Citizen User-



Private Sector User-



Government User-



8. Frontend Experience by Role

PolluTrack offers a tailored frontend experience for each user role — Citizen, Private Entity, and Government Administrator — ensuring ease of use, security, and functionality according to access privileges.

All frontend interfaces are implemented using HTML templates rendered by Flask, styled with Bootstrap, and connected to backend logic via Flask routes.

8.1 Citizen Interface (No Login Required)

- Entry Point: website_front_page.html → user_selection.html → /sensor_map
- Citizens choose their role from a dropdown or button-based selector.

- On selecting *Citizen*, they are directly redirected to:
    - sensor_map.html: Renders a Folium map showing real-time pollution sensor locations and metadata.
    - The Folium map is embedded as an HTML iframe using Jinja templating.
    - Interactivity includes: sensor name, pollutant levels, and location info via popup tooltips.

🟢 No authentication or data entry required, making it accessible for public awareness.

8.2 Private Entity Interface

- Entry Point: login.html → /dashboard_export
- After login and role verification, the user is redirected to:
    - dashboard_export.html: Displays a simple dashboard interface that:
        - Loads pollutant data.
        - Allows users to download a clean CSV for further analysis.
        - Supports minimal interactivity for lightweight use.

🔐 Private entities only see data relevant to export and do not have graph-based insights.

8.3 Government Administrator Interface

- Entry Point: login.html → /government_dashboard
- After successful authentication, government users access:
    - government_dashboard.html: A full interactive dashboard featuring:
        - Dropdowns to filter pollutants and stations.
        - Real-time visualizations using Plotly.
        - Role-based access to advanced data options.
        - Embedded link to dashboard_export.html for CSV download.

📊 Government Dashboard Functionality

The government_dashboard.html page is the core interface for visualization and analysis. It is structured into two main areas:

A. Key Performance Indicators (KPIs)

- Auto-generated KPIs for major pollutants:
    - **NO_2, PM_{10}, $PM_{2.5}$**
- Each pollutant is compared to its **defined hazard threshold**.
    - Values exceeding thresholds are visually flagged, alerting the government user of environmental risks.

B. Line Graph Visualizations

- **Plotly-powered interactive line graphs** allow monitoring of pollutant levels over time.
- Government User can dynamically select:
    - **Station**
    - **Pollutant**
    - **Time range**
- The graphs update without page reload using embedded JavaScript tied to backend Flask routes.
- Graph features include:
    - Zoom, pan, hover tooltips
    - **Hazard threshold overlay lines** (if configured)
    - **PNG export button** for direct image download.

C. Data Table & CSV Export

- Beside the sensor map, a structured data table displays raw readings.
- Users can **export the filtered dataset as a CSV** file via the "Download Data" button, which redirects to /dashboard_export.

- The export reflects all filters set in the dashboard.

This visualization and reporting flow enables government officials to:

- Detect pollution spikes early
- Correlate trends with specific regions or pollutants
- Share cleanly exported reports (graphs as PNGs, data as CSVs) for further policy and response planning

9. Implementation and Test Plan

9.1 Implementation

PolluTrack was developed through an iterative and collaborative process. Regular team meetings were held to monitor progress, resolve issues, and adjust priorities based on evolving needs. Development tasks were self-assigned and completed based on availability and expertise.

Instead of a strict sprint structure, work was divided flexibly and reviewed collaboratively to ensure quality. Each component was tested during integration to maintain stability and functionality across the system. This approach enabled timely progress, early bug detection, and consistent coordination across the team.

9.2 Testing and Deployment

Testing during development was focused on validating the core functionality and usability of the PolluTrack system as defined in the RASD. Rather than implementing formal unit or integration tests, the development team performed iterative manual testing to simulate user interactions across all roles—citizen, private entity, and government admin. Each feature, from registration and login to pollutant visualization and data export, was tested directly through the frontend interface. Special attention was given to data accuracy, API routing, session handling, and graph rendering. This hands-on testing approach ensured a stable and intuitive application experience while confirming alignment with all key functional requirements.

Functional Requirement Test Records

FR1	The system shall provide a landing page with app name and description.
Executor	Team Member 2
Date	01.07.2024
Time used	4-6 seconds
Evaluation	Success
Comment	Landing/Home page displayed correctly.

FR2	The system shall display a list of pollutants with descriptions.
Executor	Team Member 2
Date	01.07.2024
Time used	4-6 seconds
Evaluation	Success
Comment	Pollutant descriptions appeared as expected.

FR3	The system shall display 'View Pollution Data' button.
Executor	Team Member 2
Date	01.07.2024
Time used	4-6 seconds
Evaluation	Success
Comment	Button redirected to user selection page.

FR4	The system shall be accessible without login for Citizen.
Executor	Team Member 1
Date	01.07.2024
Time used	4-6 seconds
Evaluation	Success
Comment	Citizen accessed map directly without login.

FR5	The system shall be accessible with login (for new users- login after registration) for Government/Private Sector user.
Executor	Team Member 1
Date	01.07.2024
Time used	4-6 seconds
Evaluation	Success
Comment	Login form validated and routed user based on role.

FR6	The system shall allow navigation to the interactive dashboard.
Executor	Team Member 1
Date	01.07.2024
Time used	4-6 seconds
Evaluation	Success
Comment	Dashboard loaded from user selection.

FR7	The system shall render an interactive map of Lombardy.
Executor	Team Member 1
Date	01.07.2024
Time used	4-6 seconds
Evaluation	Success
Comment	Interactive map showed markers with correct data.

FR8	The system shall allow hover and click actions on markers in a region.
Executor	Team Member 1
Date	01.07.2024
Time used	4-6 seconds
Evaluation	Success
Comment	Hover/click events worked on markers.

FR9	The system shall display the latest pollutant data.
Executor	Team Member 1
Date	01.07.2024
Time used	4-6 seconds
Evaluation	Success
Comment	Displayed current pollutant values per station.

FR10	The system shall support CSV data export for selected date ranges for Government/Private Sector User.
Executor	Team Member 1
Date	01.07.2024
Time used	4-6 seconds
Evaluation	Success
Comment	CSV export worked for selected date range.

FR11	The system shall support graph display & export for selected date ranges for Government Users only.
Executor	Team Member 1
Date	01.07.2024
Time used	4-6 seconds
Evaluation	Success
Comment	Graph displayed and downloaded as PNG correctly.

FR12	The system shall support logout for government/private sector users.
Executor	Team Member 2
Date	01.07.2024
Time used	4-6 seconds
Evaluation	Success
Comment	Logout redirected to front page and cleared session.

10. Individual Contributions

Name	Approximate Individual Contribution
Viplove Gadkari	Data processing (2nd part), HTML page design (sensor map, dashboard export, government dashboard), HTML -> Flask integration, sensor map integration, dashboard logic, Flask app development, role-based logic, visualization backend, dashboard export logic
Tarig Mohammed	Data processing (1st part), HTML page design (Homepage, user selection, login, register), database connection

BIBLIOGRAPHY

Elmasri, Ramez and Shamkant B. Navathe (2015). *Fundamentals of Database Systems*. 7th Edition. Pearson.

PostgreSQL (2024). 'About'. In: *PostgreSQL.org*. URL: https://www.postgresql.org/about/.

Fielding, Roy T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. Doctoral Dissertation, University of California, Irvine. Available at: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.

Grinberg, Miguel (2018). *Flask Web Development: Developing Web Applications with Python*. 2nd Edition. O'Reilly Media.

PostGIS (2023). 'About PostGIS'. In: *PostGIS Documentation*. URL: https://postgis.net/.

Bass, Len, Paul Clements, and Rick Kazman (2012). *Software Architecture in Practice*. 3rd Edition. Addison-Wesley.