SE4G 2024-2025

POLITECNICO
MILANO 1863

Design Document and Test Plan

Aether-Analytics

Bingwen Hao

Hangyun Wang

Yujue Wang

Xinchen Yu

1. Introduction

1.1 Purpose

This project aims to develop a client-server application that supports users in querying and visualizing air quality data from the "Dati Lombardia" open data platform. By integrating and processing real-time or historical air quality measurements, the system is designed to provide insights into pollution distribution, temporal trends, and exposure risks. The target users of the platform include researchers, environmental agencies, public health organizations, and the general public interested in monitoring air quality levels.

The system serves as an analytical and visualization tool that supports evidence-based decision-making and promotes environmental awareness. It facilitates users in exploring sensor-based air quality data at multiple geographic scales and over various time spans.

1.2 Scope

The system consists of three major components:

Database (PostgreSQL + PostGIS): Responsible for storing structured data retrieved from the "Dati Lombardia" datasets. This includes sensor metadata, pollutant concentration levels, timestamps, and geographic coordinates.

Web Server (Flask): Serves as the middleware that provides REST API endpoints for querying the database. It handles data preprocessing, filtering, and packaging results into JSON format for client consumption.

Interactive Dashboard (Jupyter Notebooks): A client-side component that allows users to explore the data through interactive maps and graphs. It supports user-generated visualizations and time-series analysis.

The project scope includes:

Air quality and sensor data retrieval and integration

REST API design and backend implementation

Database schema design and spatial indexing

Visualization and analysis dashboard in Jupyter

Out of scope features include advanced predictive modeling, personalized recommendations, and mobile application development.

1.3 Context and Motivation

Environmental health monitoring is a priority for public authorities and citizens alike. With increasing urbanization and industrial activities, real-time monitoring of air

pollutants is crucial. The "Dati Lombardia" portal offers extensive data on air quality from numerous sensors across the region, yet this data is not easily accessible for non-technical users.
This project addresses this gap by transforming complex raw datasets into comprehensible and actionable insights through an intuitive interface. The motivation behind the system lies in enhancing transparency, promoting data-driven policy decisions, and educating the public about environmental conditions in their surroundings.
Furthermore, the system will support spatial and temporal analysis, empowering stakeholders to compare pollution levels across different administrative areas and track changes over time. This contributes to more informed interventions and improved public communication strategies.

1.4 Definitions and Acronyms

Term / Acronym Definition
API Application Programming Interface. Enables communication between systems.
REST Representational State Transfer. A set of architectural principles for web services.
DB Database, typically referring to PostgreSQL in this context.
PostGIS An extension for PostgreSQL that supports geographic objects and spatial queries.
Jupyter Notebook An open-source web application for creating and sharing documents that contain live code, equations, visualizations, and narrative text.
Dati Lombardia The open data portal of the Lombardy region providing environmental datasets, including air quality sensors.

2. System Architecture

The system architecture of the project follows a multi-tiered design aimed at ensuring scalability, maintainability, and efficient communication among components. The architecture is divided into the following key layers:

2.1 Overview

The system adopts a classic three-tier architecture comprising:

- **Presentation Layer**
  This layer handles the user interface and interaction. It is developed as a web-based front end using modern JavaScript frameworks and provides access for different roles such as administrators, auditors, and general users.
- **Application Layer (Business Logic)**

The core logic resides in this layer. It processes user inputs, enforces rules, manages workflows, and interacts with data services. It is implemented using a modular service-based design (microservices or modular monolith), supporting scalability and fault isolation.

- **Data Layer**
  Responsible for persistent data storage and retrieval. It utilizes relational databases (e.g., MySQL or PostgreSQL) and possibly NoSQL databases for unstructured data. It provides APIs for querying, data analytics, and audit trails.

2.2 Components

The system is composed of the following key components:

- **Authentication & Authorization Module**
  Manages user identity and access control based on roles and permissions. Integrates with LDAP or OAuth2 if required.
- **Workflow Engine**
  Orchestrates business processes and approval flows dynamically, allowing customization per organization needs.
- **Monitoring & Logging Module**
  Provides centralized logging, monitoring, and alerting using tools like Prometheus, ELK Stack, or Grafana.
- **API Gateway**
  Acts as a reverse proxy for routing requests to backend services securely and efficiently, handling concerns such as rate limiting, logging, and API versioning.
- **External Integration Module**
  Interfaces with third-party services such as government databases, email/SMS providers, or external data validation services.

2.3 Deployment

The system supports containerized deployment using Docker, and orchestration via Kubernetes or similar platforms, enabling flexible scaling and high availability. CI/CD pipelines are established to ensure rapid and reliable delivery.

2.4 Security Design

Security is embedded at each level:

- HTTPS encryption for all communication
- Role-based access control (RBAC)
- Input validation and sanitization

- Secure token-based session management
- Regular audits and vulnerability scans

3. Software Design

3.1Database Design

3.1.1 Overview

The system adopts **PostgreSQL** as the primary database management system, enhanced with **PostGIS** to support spatial data types and geospatial operations. This database is designed to manage and store essential data for the platform, including user accounts, air quality sensor metadata, and pollution measurement records.The main purpose of the database is to:

Store and organize real-time and historical air quality data from multiple sensors;

Enable spatial queries and map-based visualizations through geographic coordinates;

Support filtering and aggregation operations for pollutants, cities, and time ranges;

Manage user authentication and roles for accessing protected features.

The database plays a central role in enabling the system's core features, such as dashboard visualizations, statistical analysis, and interactive exploration of pollution data across the Lombardia region.

3.1.2 Table Structure

The database consists of six primary tables, each supporting a different functional aspect of the system. These tables are:

users: stores account information and user roles

sensors: stores metadata of air quality sensors

sensor_data: records observed pollutant measurements

reports: stores exported reports for tracking and download

regions: defines geographic groupings such as cities or districts

logins: tracks system access and activity history

All tables are designed to be normalized and consistent, with appropriate foreign key constraints to maintain referential integrity. The following subsections detail the structure of each table, including field names, purposes.

users table:

Field Name	Description
user_id	Primary key, auto-incremented
username	Unique user name
email	User's email address
password	Hashed password used for login authentication
user_type	Role of the user (general or expert)
created_at	Timestamp of account creation

Purpose:

This table stores basic account information for all users and serves as the foundation for authentication and role-based access control. The user_type field supports features differentiating expert users from general users, such as advanced data export capabilities. Passwords are securely stored in an encrypted format to protect user privacy.

sensors table

Field Name	Description
sensor_id	Primary key, uniquely identifies each sensor

name	Name or identifier of the sensor
location	Geographic coordinates of the sensor (used for map visualization)
installation_date	The date the sensor was installed
status	Operational status of the sensor (e.g:active, inactive, under maintenance)

Purpose:

This table stores metadata about all sensors in the system. Each record contains essential information such as location, name, and operational status. It supports map-based visualization, spatial queries, and helps associate data entries from the sensor_data table with the physical sensors.

sensor_data table

Field Name	**Description**
data_id	Primary key, auto-incremented
sensor_id	Foreign key, references sensors(sensor_id)
timestamp	Date and time of the measurement
pollutant	Type of pollutant measured (e.g:PM2.5, NO_2, O_3)
value	Measured concentration value
unit	Unit of measurement (e.g:µg/m^3)

Purpose:

This table records air pollution measurements collected by the sensors. Each row represents a single pollutant observation from a specific sensor at a specific time. The data supports temporal analysis and real-time/historical pollution visualization on the dashboard.

reports table

Field Name	Description
id	Unique identifier for each report
user_id	Foreign key referencing users table, indicating who exported the report
created_at	Timestamp when the report was generated
filters_applied	Summary of filters used when exporting the report (e.g: city, time range, pollutant)
file_path	File path or URL where the report file is stored
format	File format (e.g: CSV, PDF)

Purpose:

This table stores a history of data export actions initiated by users. It helps users review and manage previously generated reports, improves transparency, and supports potential audit or tracking features.

Regions table

Field Name	Description
id	Primary key, auto-increment ID
name	Name of the region (e.g: city or administrative area)
parent_id	ID of the parent region (for hierarchical structure)

level	Region level, such as city / province / region

Purpose:

This table stores spatial dimension information such as city, province, or regional names, enabling users to filter by location in the dashboard interface. It supports building a hierarchical structure (e.g:Province → City → District) for better organization and geographic queries.

logins table

Field Name	Description
id	Primary key, auto-increment
user_id	Foreign key referencing users(user_id)
login_time	Time when the user logged in
ip_address	IP address used during login
device_info	Optional field to store browser or device metadata

Purpose:

This table tracks each user login event, including the timestamp, originating IP, and optional device information. It supports security audits, user activity analysis, and can be extended for advanced features such as suspicious login alerts.

3.1.3 Table Relationships

This section outlines the logical relationships between tables in the system's database. These relationships support user account management, air quality data storage, and data export functionalities.

User-related relationships

Each user in the users table can have:

1. multiple login records in the logins table (user_id)
2. multiple exported reports in the reports table (user_id)

Sensor-related relationships

1. Each sensor in the sensors table is located in a specific region (region_id from regions table)
2. Each sensor can produce multiple air quality records in the sensor_data table (sensor_id)

The relationships are summarized as follows:

Table	Related Table	Foreign Key	Relationship Description
users	logins	user_id	One-to-Many: One user can have multiple login records
users	reports	user_id	One-to-Many: One user can generate multiple reports
regions	sensors	region_id	One-to-Many: One region contains multiple sensors
sensors	sensor_data	sensor_id	One-to-Many: One sensor produces multiple data entries

3.2 API Design

3.2.1 Overview

The system exposes a set of RESTful API endpoints that allow the frontend dashboard to interact with the backend server. These APIs enable users to fetch sensor metadata, retrieve pollution measurements, submit login or registration requests, and export filtered data as downloadable reports. All API responses are returned in JSON format, and requests are handled by a Flask-based server connected to the underlying PostgreSQL/PostGIS database.

The frontend, implemented using Jupyter Notebooks and JavaScript-based dashboards, communicates with the backend exclusively through these APIs. Parameters are

passed via URL queries (for GET requests) or JSON payloads (for POST requests). Each API is designed to be lightweight, stateless, and easy to extend, supporting future scalability.

3.2.2 API Endpoint List

The system exposes the following RESTful API endpoints to support user authentication, metadata retrieval, data query, and report export functionalities:

Endpoint	Method	Description
/api/register	POST	Registers a new user with username, email, and password
/api/login	POST	Authenticates user credentials and returns access token or session data
/api/user/profile	GET	Retrieves profile information of the currently logged-in user
/api/sensors	GET	Returns a list of all sensors and their metadata
/api/regions	GET	Returns a list of geographic regions (e.g:cities, districts)
/api/data	GET	Retrieves filtered air quality data based on pollutant type, time, region
/api/report	POST	Generates a downloadable report based on selected filters
/api/user/reports	GET	Returns a list of previously exported reports associated with the user

Each endpoint will be detailed in the next subsection, including request parameters, expected responses, and example usage.

3.2.3 Endpoint Details

This section provides detailed specifications for each API endpoint introduced in the previous section. For each endpoint, we outline the HTTP method, request parameters, expected responses, and example payloads. This is intended to support developers in building and testing the client-side application, and ensure consistency across backend interactions.

API: /api/register:

Item	**Description**
Method	POST
Purpose	Registers a new user with a username, email, and password.
Request Payload	**{** **"username": "alice88",** **"email": "alice@example.com",** **"password": "securePassword123"** **}**
Success Response	**{** **"message": "Registration successful",** **"user_id": 101** **}**
Error Response	**{** **"error": "Username or email already exists"**

	}

API: /api/login:

Item	**Description**
Method	POST
Purpose	Authenticates user credentials and initiates a session or returns a token.
Request Payload	**{** **"username": "alice88",** **"password": "securePassword123"** **}**
Success Response	**{** **"message": "Login successful",** **"user_id": 101,** **"access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."** **}**
Error Response	**{** **"error": "Invalid username or password"** **}**

API: /api/user/profile:

Item	Description
Method	GET
Purpose	Retrieves profile information of the currently logged-in user.
Request Header	Authorization: Bearer <access_token>
Success Response	**{** **"user_id": 101,** **"username": "alice88",** **"email": "alice@example.com",** **"user_type": "general",** **"created_at": "2025-03-01T10:30:00Z"** **}**
Error Response	**{** **"error": "Unauthorized or invalid token"** **}**

API: /api/user/reports:

Item	Description
Method	GET
Purpose	Returns a list of reports previously exported by the logged-in user.
Request Header	Authorization: Bearer <access_token>

Success Response	**[** **{** **"report_id": 501,** **"created_at": "2025-04-20T14:22:00Z",** **"format": "PDF",** **"filters_applied": {** **"pollutant": "PM2.5",** **"region": "Lombardia",** **"start_date": "2025-04-01",** **"end_date": "2025-04-10"** **},** **"download_url": "https://example.com/reports/501.pdf"** **},** **{** **"report_id": 502,** **"created_at": "2025-04-21T09:10:00Z",** **"format": "CSV",** **"filters_applied": {** **"pollutant": "NO2"** **},** **"download_url": "https://example.com/reports/502.csv"** **}**

	]
Error Response	**{** **"error": "Unauthorized or token expired"** **}**

API:/api/sensors:

Item	**Description**
Method	GET
Purpose	Returns a list of all available air quality sensors and their metadata.
Request Parameters	(None)
Success Response	**[** **{** **"sensor_id": 201,** **"name": "Sensor A",** **"location": [45.464, 9.19],** **"status": "active",** **"region": "Milano"** **},** **{** **"sensor_id": 202,**

	"name": "Sensor B", **"location": [45.48, 9.21],** **"status": "inactive",** **"region": "Bergamo"** **}** **]**
Error Response	**{** **"error": "Failed to fetch sensor data"** **}**

API: /api/regions:

Item	**Description**
Method	GET
Purpose	Returns a list of all supported geographic regions (e.g., cities, provinces).
Request Parameters	(None)
Success Response	**[** **{** **"region_id": 1,** **"name": "Milano",** **"level": "city"**

	}, **{** **"region_id": 2,** **"name": "Lombardia",** **"level": "province"** **}** **]**
Error Response	**{** **"error": "Unable to retrieve region list"** **}**

API: /api/data：

Item	**Description**
Method	GET
Purpose	Returns air quality measurements based on filters like pollutant type, time range, region, or sensor ID.
Request Parameters	1. **pollutant (required):** Type of pollutant to query, such as "PM2.5", "NO_2", or "O_3". 2. **start_date and end_date (required):** Define the time range for data retrieval. Dates must be formatted as YYYY-MM-DD. 3. **region (optional):** A region name or identifier to filter results geographically. **4. sensor_id (optional):** The unique ID of a specific sensor to retrieve data from a particular location.

Success Response	**[** **{** **"timestamp": "2025-04-01T09:00:00Z",** **"pollutant": "PM2.5",** **"value": 42.7,** **"unit": "µg/m³",** **"sensor_id": 201,** **"region": "Milano"** **}** **]**
Error Response	**{** **"error": "Missing required parameters"** **}**

API: /api/report:

Item	**Description**
Method	POST
Purpose	Generates a downloadable report based on selected filters and returns a file link.
Request Payload	**{** **"pollutant": "PM2.5",**

	"start_date": "2025-04-01", **"end_date": "2025-04-07",** **"region": "Milano",** **"format": "PDF"** **}**
Success Response	**{** **"message": "Report generated successfully",** **"report_id": 501,** **"download_url": "https://example.com/reports/501.pdf"** **}**
Error Response	**{** **"error": "Invalid filter parameters"** **}**

3.3 Dashboard Design (Enhanced)

The dashboard is the central user interface designed to allow stakeholders to explore, analyze, and extract insights from the air quality dataset collected by the Dati Lombardia monitoring network. It is implemented using Jupyter Notebooks and leverages Python's data science ecosystem to create an interactive, user-friendly, and visually rich experience.

Objectives

The primary goals of the dashboard are:

To visualize air quality data in both geographic (map-based) and statistical (chart-based) formats.

To allow users to filter, compare, and interpret trends in pollutant levels across different locations and timeframes.

To provide interactive feedback (warnings, alerts) when data is missing, incomplete, or inconsistent.

To enable custom view generation (e.g., exporting plots, saving selections).

Component Architecture

The dashboard is composed of five main interactive components:

Component	Description
Map Viewer	Displays sensor station locations using Folium or IpyLeaflet. Clicking on a marker shows pollutant readings and metadata. Optionally styled with color-coded pollution levels.
Time-Series Plot	Plots pollutant concentration over time using Plotly, Bokeh, or Matplotlib. Updates dynamically based on user input.
Filter Panel	Built with ipywidgets, includes dropdowns, sliders, date-pickers, and parameter toggles for user-defined queries.
Output Panel	Provides inline text feedback including data status, validation messages, and alerts (e.g., “no data available”).
Export/Save Options	(Optional) Save current chart as PNG or export filtered dataset to CSV for external use.

4. Implementation & Testing

This section outlines the planned development process, testing strategy, and quality assurance approach.

4.1 Implementation Plan

The project follows an Agile-inspired workflow organized in weekly sprints. Tasks are tracked via GitHub Projects or Trello.

Sprint	Main Activities
Sprint 1	Setup GitHub repo, Flask project, and PostgreSQL DB schema
Sprint 2	Develop initial data ingestion and preprocessing scripts
Sprint 3	Implement backend APIs and test sample queries
Sprint 4	Create initial dashboard prototype (Jupyter Notebook)
Sprint 5	Integrate maps, widgets, and plots into the dashboard
Sprint 6	Final testing, error handling, export features
Sprint 7	Write SRD + Polish all components for final delivery

4.2 Testing Strategy

We apply unit tests, integration tests, and manual system tests to validate the backend and the dashboard.

Unit Testing (for Flask Backend)

Tool: pytest

Scope: Each API endpoint is tested for:

Correct input validation

Response structure (status 200/400/500)

Data format (valid JSON with expected keys)

Endpoint	Method	Description	Test Case
/api/sensors	GET	Returns list of sensors	Test 200 + array of sensor IDs
/api/data?city=X&param=PM10	GET	Get time-series data	Valid params: return 200; Invalid: return 400
/api/summary	GET	Returns average values	Check keys, values not null

Integration Testing

What is tested: Dashboard–Backend communication

Method: Run Jupyter cell → call Flask endpoint → assert result shape & values

Manual System Testing (End-to-End)

Scenario	Action	Expected Result
Select sensor on map	Click marker	Popup shows sensor details
Filter by date & pollutant	Set filters and submit	Chart updates accordingly
City with no data	Select invalid city	Warning: “No data found”
Backend offline	Reload dashboard	Error shown: “Server unavailable”
Export view	Click download	PNG or CSV saved successfully

5. Bibliography

SE4GEO Project Assignment 2025 Document
Dati Lombardia Air Quality Sensor Datasets:
https://www.dati.lombardia.it/Ambiente/Dati-sensori-aria-dal-2018/g2hp-ar79/about_data
https://www.dati.lombardia.it/Ambiente/Stazioni-qualit-dell-aria/ib47-atvt/about_data
Flask Documentation: https://flask.palletsprojects.com/
PostgreSQL Documentation: https://www.postgresql.org/docs/
PostGIS Documentation: https://postgis.net/
Jupyter Project: https://jupyter.org/
Python Libraries:
Pandas: https://pandas.pydata.org/
GeoPandas: https://geopandas.org/
Plotly: https://plotly.com/
Folium: https://python-visualization.github.io/folium/
IpyLeaflet: https://github.com/jupyter-widgets/ipyleaflet
Matplotlib: https://matplotlib.org/
Bokeh: https://docs.bokeh.org/