POLITECNICO
MILANO 1863

DesignDocument

Deliverable: DD

Title: DesignDocument

Authors: HU TIANQI, YAN SHINUO, SU JIAYI

Version: 3.0

Date: May 25, 2025

Download page:
https://github.com/Ricky-HU96/WO-CIAO_SE4GEO-2025-Project

Version	Date	Change
1.0	April 22,2025	First submitted version
2.0	May 6,2025	Second submitted version
3.0	May 25, 2025	Third submitted version

Table of Contents

List of Tables

List of Figures

1. Introduction

1.1Design Document

Before starting to code and implement the software, an important step is to have a clear understanding of the design goals that our Web application will achieve. All of these goals should be established in this particular design document.

The key reason for writing this document is to define the structure and organization that will continue throughout the creation period until the software project is finalized. This is an important guide document for the engineering team, which they can rely on for any subsequent steps.

A software design document is a technical description that serves both the developer and outlines the plan to meet the project goals, while helping the development team maintain an efficient workflow.

It is important to emphasize the fact that the Design Document is built upon the Requirements Analysis Specification Document.

The connection between the above-mentioned documents helps us to be careful with not avoiding any important functionality. The characteristics specified would not be changed or described in another way but will remain an implicit knowledge for the development team.

More specifically it will include the following characteristics:

Project Database: This is a crucial part of the project we are developing. Database design defines the tasks and processes for structuring, developing, and maintaining the data management system for Dati Lombardia air quality data. It determines the necessary data elements and their relationships.

System Overview: This section will provide a general description of the structure and functionality of the software system, outlining the main components and their interactions.

Software Structure: Our software, as a client-server application, will be established as a three-tier architecture to manage interactions between the client and the server.

User Interface: This section will detail the design of the interactive Jupyter Notebook dashboard, aiming for an intuitive and user-friendly experience for exploring air quality data.

User Case Application: This section will include the use cases and requirements map for each component of the software, illustrating how users will interact with the system to achieve specific goals based on the RASD.

1.2Product Description

The purpose of developing this product is to provide an interactive application for visualizing and analyzing air quality data derived from the Dati Lombardia sensor network. This application aims to support users such as environmental agencies, public health organizations, researchers, and the general public in understanding air pollution levels, trends over time, and spatial distribution across the Lombardia region. This application will allow users to query specific data, visualize it on interactive maps and charts, and potentially perform basic statistical analysis using an accessible Jupyter Notebook interface.

2. Project Database

2.1 Dati Lombardia Data Source

The project database integrates data from publicly available datasets provided by Dati Lombardia, focusing on air quality monitoring within the region. The primary sources are:

https://www.dati.lombardia.it/Ambiente/Stazioni-qualit-dell-aria/ib47-atvt/about_data

These datasets form the foundation of the application, providing the raw measurements and the necessary context (sensor locations and characteristics) for meaningful analysis and visualization. Data integration involves retrieving, cleaning, transforming, and combining these sources into a structured format within our project database.

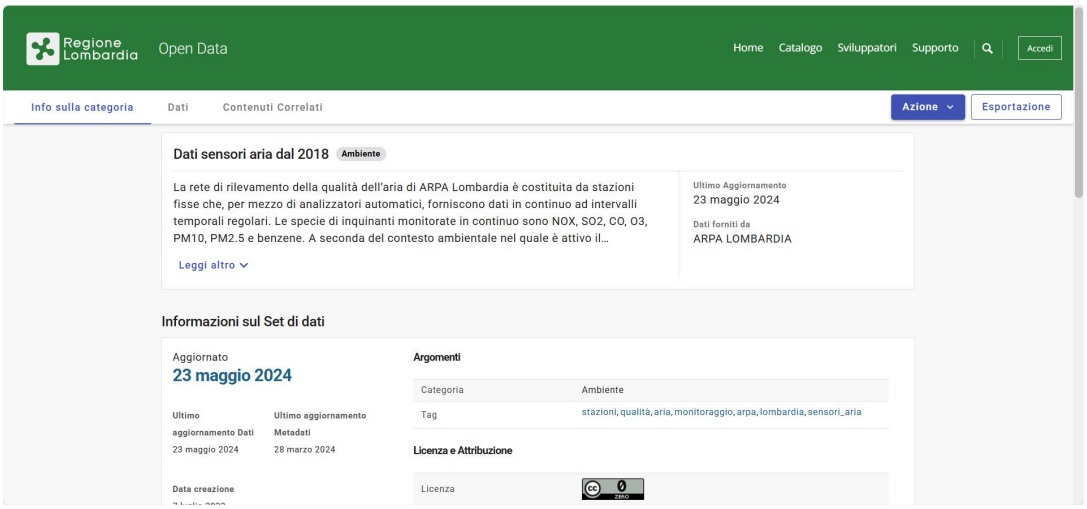

Figure 1：Dati Lombardia Air Quality Sensors Dataset

2.2 PostgreSQL Database

The data management for this application relies on PostgreSQL, a robust open-source relational database system, enhanced with the PostGIS extension for geospatial capabilities. This choice enables efficient storage, querying, and management of both the time series measurement data and the spatial locations of the air quality sensors.

PostgreSQL's relational structure is well-suited for storing the structured measurement data linked to sensor metadata. The PostGIS extension provides essential functions for storing sensor coordinates as geographic points and allows for potential future spatial analysis .

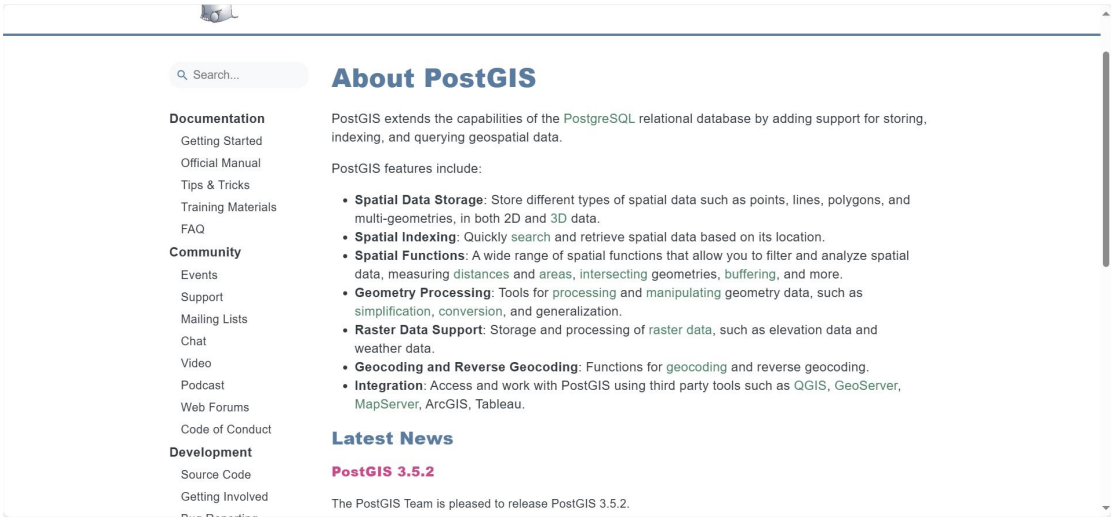

Figure 2：The PostGIS extension

3. System Overview

The system is designed to support air quality analysis using Dati Lombardia sensor data through an interactive, web-based application. It leverages geospatial data processing and visualization tools to provide decision-makers, researchers, and the public with critical insights into air pollution dynamics, including spatial distribution, temporal trends, and potential exposure risks within the Lombardia region. The application follows a client-server architecture, facilitating modularity and separation of concerns.

3.1 System Architecture

The system is designed with a robust architecture to efficiently manage, process, and visualize geospatial air quality data. It comprises three main components:

3.1.1 Database (PostgreSQL with PostGIS Extension):

Function: Serves as the persistent storage layer.

Content: Stores integrated and processed data from Dati Lombardia sources, including sensor metadata (locations, types) and time series air quality measurements.

Technology: Utilizes PostgreSQL for relational data management and PostGIS for handling geographic objects (sensor locations) and enabling spatial queries.

3.1.2Web Server

Flask Framework

Function: Acts as the backend application server (logic tier).

Responsibilities: Exposes a REST API that the frontend dashboard consumes. It handles client requests, queries the database, potentially performs necessary data preprocessing or aggregation, and returns data formatted as JSON.

Technology: Built using Flask, a lightweight Python web framework.

3.1.3Dashboard

Jupyter Notebooks:

Function: Serves as the frontend presentation and interaction layer.

Responsibilities: Provides an interactive user interface built within a Jupyter Notebook. Users interact with widgets to specify query parameters. The notebook then calls the backend API, receives data, performs analysis , and visualizes the results using maps and charts .

Technology: Jupyter Notebook, ipywidgets, requests, Pandas/GeoPandas, and various plotting/mapping libraries.

3.1.4HighlevelArchitechture

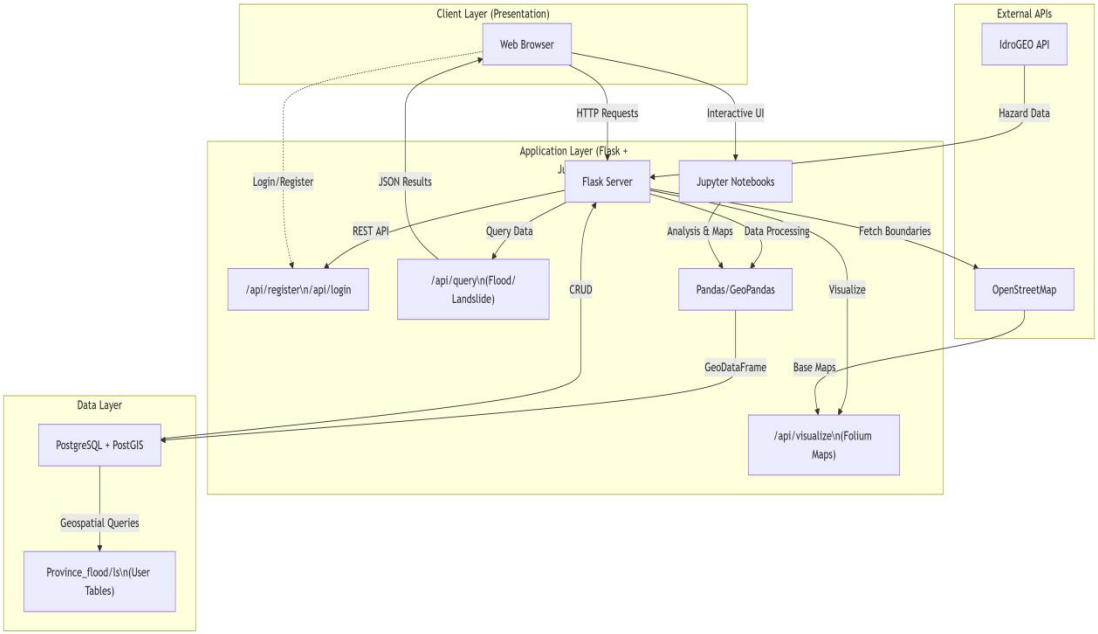

Figure 3：HighlevelArchitechture

The diagram shows how these components are interconnected. The Flask App interacts with the database (Postgres) and retrieves data originating from Dati Lombardia. The Jupyter Notebook acts as the web interface, allowing users to trigger API calls to Flask, receive data, and view interactive visualizations (maps, charts).

3.2 Key Features

Data Retrieval and Visualization: Users can query air quality data based on specific sensors, pollutants, and time periods using interactive controls. The system provides visualizations including:

Interactive maps displaying sensor locations, potentially styled based on data values.

Time series charts showing pollutant concentration trends.

Interactive Data Exploration: The Jupyter Notebook environment allows users to directly interact with the data and visualizations, facilitating exploration and analysis.

Combined Data View: Integrates sensor location and metadata with time series measurements for comprehensive analysis.

3.3 Technologies Used

Backend: Flask (Python), PostgreSQL, PostGIS

Frontend: Jupyter Notebooks, ipywidgets, Folium / IpyLeaflet, Plotly / Bokeh / Matplotlib

Data Processing & Utilities: Pandas, GeoPandas, Requests, Psycopg2 / SQLAlchemy

Version Control: Git, GitHub

Development Environment: VSCode , Python 3.x

4. Software structure

The software's structure will be organized into three logical and physical computing tiers or system layers, which are:

Presentation Server (Jupyter Notebook via User's PCs/Browser)

Application Server (Flask)

Database Server (PostgreSQL)

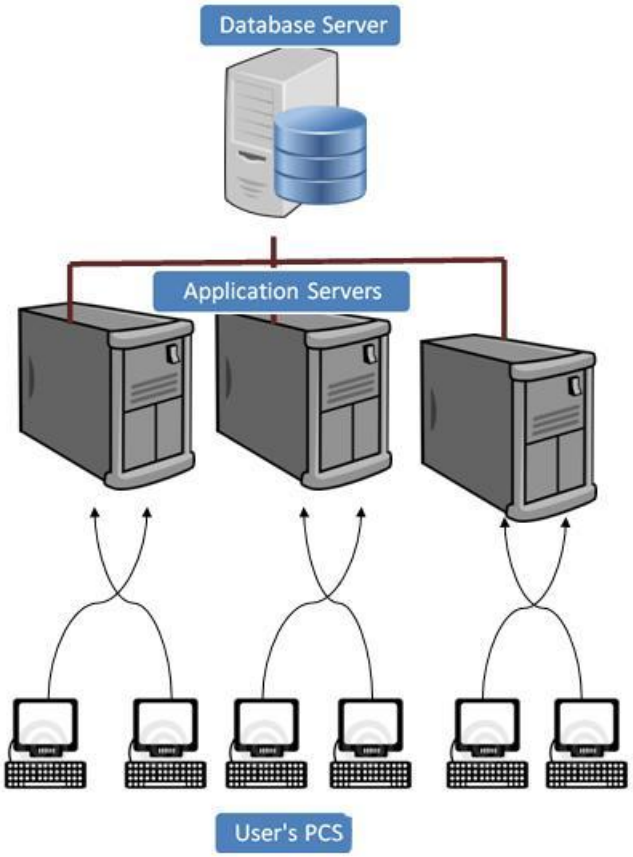

Figure 4: Software structure

4.1 Presentation Server

This top-level layer is the application's user interface, implemented using a Jupyter Notebook and accessed via a web browser. Users interact with the system through ipywidgets controls embedded within the notebook. Its role is to present information to the users and simultaneously gather input from them. The notebook environment itself handles rendering using HTML, CSS, and JavaScript, while Python code within the notebook orchestrates API calls and visualization updates using libraries like Folium, Plotly, etc.

4.2 Application Server

The application server, also known as the logic tier, is implemented using the Flask framework in Python. It handles all the necessary logical operations to

meet the software requirements based on requests from the presentation tier . It processes information gathered , interacts with the database tier to retrieve data, performs required data transformations or aggregations, and generates responses to be sent back to the Jupyter Notebook. Python, along with SQL for database interaction, is used to develop this server.

The essential libraries, categorized by function, are:

Web Framework: Flask (handling HTTP requests, routing, API endpoint definition)

Database Interaction: Psycopg2 or SQLAlchemy (connecting to PostgreSQL, executing queries)

Data Handling: Pandas

API Support: Flask-CORS

Key software functions within this tier:

API Endpoint Handlers: Functions mapped to specific URL routes that parse incoming request parameters.

Database Query Logic: Functions that construct and execute SQL queries based on the parsed parameters, interacting with the database server.

JSON Serialization: Functions to convert database results into the specified JSON format for the API response.

4.3 Database Server

The data tier consists of the PostgreSQL database server, augmented with the PostGIS extension. This is the back-end where the application's processed and integrated information is stored and managed persistently. The application server communicates with the database server, typically via a database adapter like Psycopg2 or SQLAlchemy, to execute SQL queries for retrieving the data needed to fulfill API requests. PostGIS provides the capability to store and query the geographic locations of the sensors efficiently.

5. User Case Application

In this section, we will show the possible functions and use cases of the website, and consider the exceptions that may occur under different conditions.

Use case 1: Specified Condition Data Query	
Name	Specified Condition Data Query
User	Any user interacting with the dashboard.
Condition	The Jupyter Notebook dashboard is running and connected to the backend API and database.
Flow of Events	1. User inputs or selects query conditions on the interface. 2. After confirming the query conditions, the user clicks the "Query" or "Update" button. 3. The dashboard sends a request to the appropriate Flask API endpoint . 4. The dashboard receives and processes the JSON response. 5. Eligible data is displayed/updated in the relevant chart and statistics panels. 6. The user reviews the query results and may perform further analysis or refine the query.
Exit condition	User closes the dashboard/browser or navigates away. User modifies the query conditions and initiates a new query.
Exceptions	User enters query conditions in an incorrect format) -> Input validation in Jupyter or 400 error from API. There are system errors or data source connection issues (API unavailable, DB unavailable) -> Error message shown in dashboard. The query results are empty (no data for criteria) -> "No data available" message shown.

Table 1: Specified Condition Data Query

Use case 2: Data Geographic Visualization	
Name	Data Geographic Visualization
User	Any user interacting with the dashboard.
Condition	The dashboard is running, connected to the backend, and sensor location data is available.
Flow of Events	1. User selects criteria for visualization. 2. User triggers the visualization process . 3. The dashboard requests necessary data . 4. The system generates a geographic visualization in the Map panel. 5. Sensor locations are displayed as markers. Markers may be styled or have popups based on the selected/latest data values.
Exit condition	User interacts with other dashboard elements, changes visualization parameters, or closes the dashboard.
Exceptions	No data matches the selected criteria for styling markers -> Markers shown with default style or indication of no data. In case of data retrieval errors -> Map may show only base layer or an error message. Map rendering issues .

Table 2: Data Geographic Visualization

Use case 3: Base Map Switch	
Name	Base Map Switch
User	Any user interacting with the map view in the dashboard.
Condition	The interactive map is displayed in the dashboard. The mapping library is configured with multiple base map tile layer options.

Flow of Events	1. Users locate the base map switch functionality on the map interface. 2. Users click the layer control and select a different base map option . 3. The system loads the appropriate map tiles based on the user's selected base map type, replacing the current map background display while keeping data overlays visible.
Exit condition	User selects another base map, interacts with other map features, or navigates away from the map view.
Exceptions	User selects an invalid or unavailable base map type -> Selection might fail, or map shows errors. Users encounter issues while switching base maps -> Map background may appear blank or show loading errors.

Table 3:Base Map Switch

6 . User Interface

The system uses a user-friendly dashboard built with Jupyter Notebook, Flask (backend), and PostgreSQL (database). It allows users to register, log in, view air quality data, make queries, and see results through maps and charts.

6.1 Design Principles

Easy to Use: Clear layout and simple controls, even for non-technical users.

Fast Response: The interface updates quickly after user actions.

Modular: Each part of the interface (map, chart, analysis) works separately for easy updates.

Secure: All user data and communication are protected.

6.2 Main Interface Components

6.2.1 User Registration & Login

Registration Page: New users can sign up with a username, email, and password. The system checks for duplicate usernames and weak passwords.

Login Page: Registered users can log in. Errors are shown if login fails.

Database Link:

Registration info is sent to the backend via Flask API.

The backend saves the info into the users_user table in PostgreSQL.

Passwords are encrypted for safety.

6.2.2 Query Panel

Users can choose search filters:

Date range

Pollutant type

Then click "Search" to get results from the backend

6.2.3 Map View

Shows sensor locations on an interactive map using Folium or IpyLeaflet

Color-coded by pollution level

Base maps can be changed

Clickable markers show details

6.2.4 Charts

Interactive charts using Plotly, Matplotlib, or Bokeh

Show trends over time or comparisons between pollutants

Zoom, hover, and export options available

6.2.5 Analysis Panel

Shows average, max, and min values of selected data

Highlights possible pollution risks

Shows patterns and changes over time