POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Design Document - GeoCICI – Group

Dispatchment of air quality data through interactive maps and time series

Authors
Milad Ramezani Ziarani
10930504

Deliverable:	DD
Title:	Design Document
Authors:	Milad Ramezani Ziarani
Version:	2.0
Date:	18th of June 2023
Copyright:	

Revision history

Version	Date	Change
version 1.0	25th of May, 2023	First submitted version
version 2.0	18 th of June, 2023	Second submitted version

Table of Contents

1 Introduction

1.1 Design Document

This Design Document serves as a guide for our web application project titled "Dispatchment of Air Quality Data through Interactive Maps and Time Series." Its purpose is to establish clear design goals and provide comprehensive guidelines to ensure the success of our project.

The Software Design Document is relevant to both the development team and the client, serving as a technical description of the application. It helps the developers fulfill client objectives and provides an efficient workflow. Every project member should be familiar with its contents.

This Design Document is built upon the Requirements Analysis Specification Document (RASD) created by our group, ensuring no important functionality is overlooked.

It is important to note that this document focuses on application requirements and functions, rather than implementation details. The specified characteristics remain implicit knowledge for the development team.

The document includes a description of the application, completion criteria, milestones, and the following key characteristics:

1. Database: PostgreSQL and PostGIS
2. Web server: Flask
3. Data processing and utilities: Pandas, GeoPandas, Dask, Xarray, Rasterio
4. Dashboard: Jupyter Notebook, Jupyter Widgets, ITables, Mercury, Matplotlib, Plotly, Bokeh, IpyLeaflet, Folium, geemap

Our project is led by the Geo-CICI group, committed to delivering a high-quality and interactive air quality data dispatchment system through this web application. While a GitHub link is currently unavailable, we are dedicated to the success of our project.

- **Project Database**

The project database plays a crucial role in our web application, involving the design, development, and maintenance of a data management system. We have chosen PostgreSQL and PostGIS as our software tools.

PostgreSQL is an open-source relational database management system known for its power and extensibility. It offers features for data storage, retrieval, and management, making it suitable for complex datasets. PostGIS, an extension to PostgreSQL, adds support for geographic objects and spatial analysis, enabling us to incorporate spatial data into our database.

Using PostgreSQL and PostGIS, we can create an efficient database structure for storing and retrieving spatial and non-spatial data, supporting interactive maps and time series analysis in our web application.

- **System Overview**

Our project focuses on developing an intuitive and engaging web application called "Dispatchment of Air Quality Data through Interactive Maps and Time Series." The system aims to provide users with a platform to explore and analyze air quality data effectively.

The software structure follows a dashboard web application architecture, ensuring seamless interaction between the client and server components. This architecture forms the foundation for organizing and structuring our application.

- **Software Structure**

Further details regarding the software structure will be elaborated in the subsequent sections of this document. It will provide a comprehensive explanation of how different components of the application are organized and collaborate to deliver the desired functionality.

- **User Cases and Requirements**

This section will outline the use cases and requirements map for each component of the software. It references the detailed analysis conducted in the Requirements Analysis Specification Document (RASD). The use cases demonstrate the functionalities and interactions within the application, ensuring all requirements are covered.

Referencing the RASD, we will provide an overview of the use cases and their corresponding requirements. This comprehensive approach ensures that all functionalities and user interactions are considered, facilitating the achievement of desired outcomes.

1.2 Product Description

Our web-based application aims to educate users about air pollution in Europe by providing real-time air quality data and interactive features.

The software retrieves up-to-date air quality data from the website https://discomap.eea.europa.eu/Index/ using a REST API. This data serves as the foundation for users to learn about the types of pollutants present in the air and gain insights into the air quality trends over the past seven days.

The application offers users the ability to visualize the countries on a map of Europe. They can explore detailed time series of pollutant concentrations and compare the air quality among different countries. Additionally, a ranking system will highlight the countries with the best and worst air quality.

The web application provides a combination of static and dynamic information. Static content, presented through HTML, offers general information about air pollution and its environmental impact. Dynamic elements, implemented using Python, facilitate interactive features such as data querying, visualization, analysis, and user comments.

Our objective is to create an open-source and user-friendly platform that empowers individuals to understand and engage with air pollution data. By raising awareness and encouraging active participation, we aim to foster a better understanding of air quality issues and inspire actions to improve the environment.

2 Architectural Design

2.1 Overview

The architectural design of our web application follows a client-server model, with a Flask app serving as the server-side component and a web interface as the client-side component. The Flask app collects data from various APIs, performs necessary calculations, formats the output data, and serves it to the web interface. The web interface, developed using HTML, CSS, and JavaScript, allows users to interact with the data, visualize air quality information on interactive maps, and analyze time series of pollutant concentrations.

The design emphasizes modularity and scalability, enabling the addition of new features without disrupting the existing structure. The use of RESTful APIs facilitates communication between the Flask app and the web interface, ensuring seamless data transfer and interaction. The architectural design also considers extensibility by allowing the integration of additional libraries or modules to enhance functionality.

2.2 High level architecture

The high-level architecture of the Dispatchment of air quality data through interactive maps and time series application can be divided into three main components: the client-side, the server-side, and the data storage.

- **Client-Side**

The client-side component consists of the web interface that users interact with. It is developed using HTML, CSS, and JavaScript to provide a visually appealing and user-friendly experience. The client-side handles user inputs, displays data visualizations, and communicates with the server-side through RESTful APIs. Users can register, log in, explore maps, play the game, view pollutant information, compare countries, and access various functionalities provided by the application.

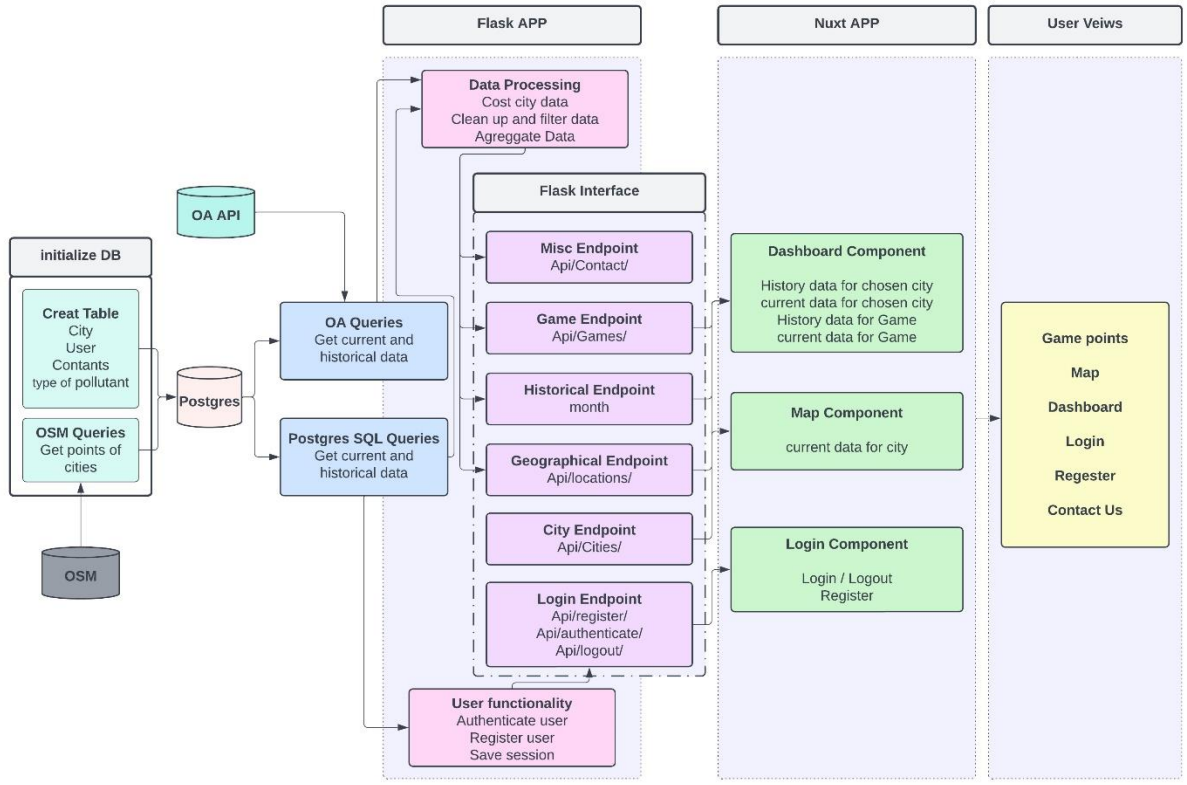
- **Server-Side**

The server-side component is built on a Flask app, which acts as the middleware between the client-side and the data storage. The Flask app collects data from the REST API of the European Air Quality database (https://discomap.eea.europa.eu/Index/) and performs necessary calculations and data processing. It also manages user authentication and session management. The Flask app exposes RESTful APIs that allow the client-side to retrieve data, send requests, and receive responses.

- **Data Storage**

The data storage component includes a PostgreSQL database, which stores user information (username, password) and the retrieved air quality data. The database is accessed and managed by the Flask app using the psycopg2 library. The data stored in the database includes pollutant concentrations for different countries, time series data, and game scores. The PostgreSQL database supports georeferenced data and spatial operations, enabling efficient handling of geometry data for maps and spatial analysis.

The high-level architecture ensures a clear separation of concerns, with the client-side responsible for the user interface, the server-side handling data processing and communication, and the data storage component managing persistent data storage. The use of RESTful APIs enables seamless communication between the client-side and server-side, ensuring efficient data transfer and interaction.



2.3 Interface

Endpoint type	Endpoint URL	Data supplied	input Accepted	JSON scheme
UserData	/api/register	User register, sends username and password,return success or not.(User already registered)	Payload JSON: {“username”:<string>, “password”:<string>}	{“user”: { “username”: <String>, “userID”: <Int>, “password” : <string>}, “register”: <Bool>, }
UserData	/api/login	User login, sends username and password,return success or not.	Payload JSON: {“username”:<string>, “password”:<string>}	{“user”: { “username”: <String>, “userID”: <Int>,“password” : <string> }, “login”: <Bool>, }
UserData	/api/logout	User logout, sends username and score,return success or not.	Payload JSON: {“username”:<string>, “score”:<string>}	{“user”: { “username”: <String>, “userID”: <Int>,“score” : <string> }, “login”: <Bool>, }
PollutantsData	/api/pollutants	User choose pollutants	None	{“pollutant”: [<String>, ...]}

PollutantsData	/api/pollutants /<pollutant	Values of all parameters for that pollutant	Pollutant information: string	{“pollutant”: {“pollutantName”: <String>, “lastUpdated”: <String>, “Measurements“: [{“parameter”: <String>, “unit”:<String>, “value”: <Int>},...]}}
CountryData	/api/countries	User choose country	None	{“countries”: [<String>, ...]}
CountryData	/api/countries/ < country>	Pollutant in the country	Information on pollutants in selected countries	{“country”: {“countryName”: <String>, “lastUpdated”: <String>, “pollutant“: [{“parameter”: <String>, “unit”:<String>, “value”: <Int>},...]}}
Geographical Data	/api/location s,/api/latest	/locations sends latest data for each station, /latest sends for each country.	None	{locations: [{“countryName”: <String>, “Coordinates”: [lng: <Float>, lat: <Float>], “particles”: [{“parameter”: <String>,”value”: <Float>,”unit”: <String>,
TimeData	/api/time	The timing of pollutants in selected countries	None	{“time”: [<String>, ...]}
ScoreData	/api/score	User score	Rankings: Int	{“score”: [<String>, ...]}
MapData	/api/map	Explore map	None	{locations: [{“countryName”: <String>, “Coordinates”: [lng: <Float>, time:: <Float>], “polluatnt”: [{“parameter”: <String>,”value”: <Float>,”unit”: <String>,“lastUpdate d”: <String>},...]}, ...]}
HazardData	/api/hazard	Show hazard	Hazard information: String	{“Hazard”: [<Text>, ...]}

A Flask interface is a web application or API built using the Flask micro web framework in the Python programming language. It allows developers to create web interfaces that can handle HTTP requests and return responses to clients, such as web browsers or other applications. Flask interfaces can be used to build a wide variety of web-based applications, from simple one-page websites to more complex applications that require database integration, user authentication, and other advanced features. Flask interfaces are known for their simplicountry and flexibility, making them a popular choice for developers who want to rapidly prototype and deploy web applications.

1 Project Database

The data required for this project, which includes information about point positioning and pollution levels, will be obtained from the API of various countries available on the website https://discomap.eea.europa.eu/Index/. We will preprocess this data to ensure its compatibility and usability within our web application.

The web application will interact with a Database Management System (DBMS) and utilize PostgreSQL as the chosen database. All user requests and interactions will be stored and managed within the PostgreSQL database, enhancing the overall functionality and quality of our application.

By leveraging the power and capabilities of PostgreSQL, we can efficiently handle and manipulate the data, allowing for seamless retrieval, storage, and retrieval of user-generated content. The database serves as a crucial component in ensuring the reliability and performance of our web application.

1.1 PostgreSQL Database

For our project "Dispatchment of Air Quality Data through Interactive Maps and Time Series," we have chosen to utilize a local server of PostgreSQL to store relevant data. This decision was driven by PostgreSQL's open-source nature and its ability to handle georeferenced data effectively.

PostgreSQL, in combination with the PostGIS extension, supports spatial operations and allows us to work with various geometries such as polygons, line strings, and points. This functionality enables us to perform spatial operations on both raster and vector data, including buffering, union, clipping, and measuring intersections, areas, and lengths.

Moreover, the integration of Python with PostgreSQL is made possible through the psycopg2 library. This integration will facilitate seamless data manipulation and querying, enhancing the functionality and interactivity of our web application.

In the context of "Dispatchment of Air Quality Data through Interactive Maps and Time Series," the structure of the PostgreSQL database will be designed to accommodate the specific requirements of storing air quality data, including pollutant concentrations, geographic information, and time series data.

Database

1.1 Data Sources and tool

The data we use will be retrieved from the website https://discomap.eea.europa.eu/Index/ by a REST API. We will use PostgreSQL to store and manage our data. PostgreSQL is free, stable and open source. Besides, it strongly supports geospatial data and can handle many types of geospatial data.

1.2 Data composition

After users registering in our web app, their data will be stored in our database and can be retrieved in the future. We mainly focus on the data of pollutants of European capital countries. Therefore, data stored in our database will mainly consist of two parts, user data and pollutants data. Bellowing are the tables with the attributes.

Table 1: User table attributes definition

Pollutants table	
country	Name of country
time	time and date of the pollutant measurement
CO	Concentration of CO in the air
PM2.5	Concentration of PM2.5 in the air
PM10	Concentration of PM10 in the air
NO	Concentration of NO in the air
NO2	Concentration of NO2 in the air
SO2	Concentration of SO2 in the air
O3	Concentration of O3 in the air
geometry	Kind of geometry
id	id of user who perform the request

Table 2: Pollutants table attributes definition

2 System Overview

The "Dispatchment of Air Quality Data through Interactive Maps and Time Series" system is designed with a client-server architecture, consisting of multiple highly interactive pages.

The system operates through a web-based interface, where the client interacts with the server to access various features and functionalities. The client-side of the system is responsible for rendering and displaying the web pages, while the server-side handles data processing, storage, and retrieval.

The system is organized into different pages, each serving a specific purpose and offering unique functionalities to the user. These pages are designed to provide an engaging and intuitive user experience, allowing users to explore air quality data, interact with maps, and analyze time series information.

The client-server architecture ensures seamless communication between the user's web browser and the server, enabling efficient data exchange and real-time updates. The server-side of the system manages the retrieval of air quality data from the designated API, processes and preprocesses the data, and stores it in the PostgreSQL database.

The interactive nature of the system allows users to interact with the data by selecting countries, viewing pollutant concentrations, and visualizing trends over time. The system also incorporates features such as game elements, rankings, and user comments to enhance user engagement and provide an immersive experience.

By leveraging this client-server architecture and interactive web pages, the system enables users to gain insights into air quality data, explore geographical information, and make informed decisions regarding air pollution.

3 Software Structure

The software's structure for the "Dispatchment of Air Quality Data through Interactive Maps and Time Series" project will be organized into logical and physical computing tiers or system layers. As we have decided to use Python for our project, the software structure will be based on Python-based technologies and frameworks.

- **Presentation Layer**

This layer focuses on the user interface and interaction. It includes the web pages, forms, and components that users interact with. We will utilize web technologies such as HTML, CSS, and JavaScript to create an intuitive and visually appealing user interface.

- **Application Layer**

The application layer contains the business logic and functionality of the system. It will be implemented using Python and the Flask web framework. Flask provides a lightweight and flexible environment for building web applications. This layer will handle user requests, process data, and interact with the database.

- **Database Layer**

This layer involves the storage and retrieval of data. We have chosen PostgreSQL as our database management system due to its open-source nature, robustness, and support for georeferenced data. We will utilize the PostGIS extension to work with spatial data efficiently. The Python library psycopg2 will facilitate the interaction between the application layer and the database.

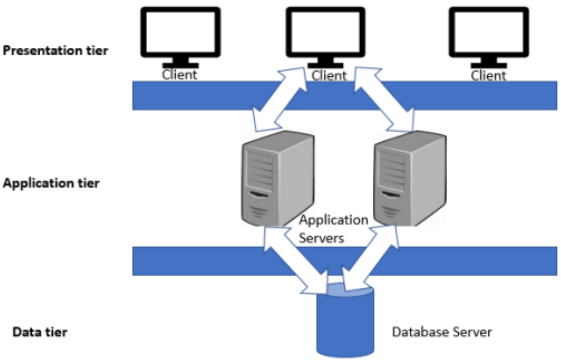


Figure 3: Software structure

By organizing the software into these logical and physical layers, we ensure separation of concerns, modular development, and scalability. Each layer has a specific responsibility and can be developed and maintained independently, allowing for easier testing, debugging, and future enhancements.

4 User Interface

The user interface of the "Dispatchment of Air Quality Data through Interactive Maps and Time Series" web application focuses on providing a consistent and intuitive experience across all pages. We have identified common elements that will be present throughout the application to enhance usability and user interaction.

- **Navigation Menu**

A navigation menu will be placed at the top or side of each page, allowing users to easily access different sections and functionalities of the application. It will provide links to the main pages, such as the home page, interactive maps, time series analysis, and user profile.

- **Interactive Maps**

The web application will feature interactive maps that display air quality information for various countries. Users can interact with the maps, zoom in/out, and click on specific locations to view detailed information about pollutant levels and other relevant data.

- **Time Series Analysis**

The application will provide time series analysis of pollutant concentrations for the selected countries. Users can choose a specific time range and view trends and patterns in air quality over that period. Interactive charts and graphs will be used to present the data in a visually appealing and understandable manner.

- **Responsive Design**

The user interface will be designed to be responsive, ensuring that the application is accessible and usable across different devices, including desktops, tablets, and mobile phones. This will provide a seamless user experience regardless of the device used to access the application.

By incorporating these common elements in the user interface, we aim to create a user-friendly and engaging experience for users to explore and analyze air quality data.

5 Use Cases

This section outlines the functionalities and interactions between the components of the "Dispatchment of Air Quality Data through Interactive Maps and Time Series" web application. It describes the actions taken by the software and users, providing insight into each internal process of the application. Both server-side and client-side actions are considered for each use case, highlighting the events and actions that occur in different situations.

1. Interactive Map Exploration:
   - Client-Side: The user navigates to the interactive map page and selects a country or location of interest.
   - Server-Side: The server retrieves the air quality data for the selected country from the PostgreSQL database and sends it to the client-side for visualization on the map.

2. Time Series Analysis:
   - Client-Side: The user selects a specific time range and pollutant type for time series analysis.
   - Server-Side: The server fetches the corresponding data from the database and performs calculations to generate the time series plot. The resulting graph is sent to the client-side for display.

3. User Comment Submission:
   - Client-Side: The user writes a comment regarding the analysis or air quality data and submits it.
   - Server-Side: The server receives the comment, validates and stores it in the database, associating it with the corresponding user and data analysis.

4. Saved Queries and Analysis:
   - Client-Side: The user saves their queries and analysis for future reference.
   - Server-Side: The server stores the user's saved queries and analysis in the database, allowing them to retrieve and view their saved information in their user profile.

These use cases provide an overview of the software's functionalities and the interactions between the server-side and client-side components. By considering all possible exceptions and documenting them in the testing document, we ensure comprehensive testing and robustness of the application.

6 Implementation Plan

4.1 Implementation Approach

The implementation of the interactive air quality monitoring application will follow an iterative and incremental approach, allowing for continuous feedback and improvement throughout the development process. The team will utilize the Agile methodology, which promotes flexibility and collaboration, to ensure efficient and effective implementation.

The implementation approach includes the following steps:

1. Requirements Analysis: Review and analyze the requirements specified in the Requirements Analysis and Specification Document (RASD). Identify any potential gaps or ambiguities in the requirements and seek clarification if needed.

2. System Design: Based on the design specifications outlined in this Design Document (DD), create a detailed system design that defines the architecture, components, and modules of the application. Identify the technologies, frameworks, and tools to be used.

3. Development Iterations: Divide the development process into multiple iterations, each focused on implementing a set of features or functionalities. Prioritize the implementation tasks based on their importance and dependencies.

4. Coding and Unit Testing: Implement the functionality of each component/module according to the design specifications. Write clean and well-documented code following coding best practices. Perform unit testing to ensure the correctness of individual components.

5. Integration and System Testing: Integrate the developed components/modules into a cohesive system. Conduct comprehensive system testing to verify the correct integration of the components and validate the application against the functional and non-functional requirements.

6. User Interface Development: Develop the user interface components, including the dashboard, maps, charts, and search functionality. Ensure a visually appealing and user-friendly interface that meets the usability and accessibility standards.

7. Database Setup: Set up the PostgreSQL database and implement the necessary tables and relationships according to the database design specifications. Test the database functionality and data retrieval operations.

8. Deployment and Deployment Strategy: Prepare the application for deployment to a production environment. Define a deployment strategy that outlines the steps and considerations for deploying the application, including server setup, configuration, and deployment automation.

9. Documentation: Create comprehensive documentation for the application, including user guides, installation instructions, and API documentation. Ensure that the documentation is up to date and reflects the final implementation.

4.2 Development Tasks

The development tasks will be divided among the team members based on their expertise and skills. The following tasks are identified:

1. Backend Development: Implement the backend logic, including data retrieval from the database, API development, and integration with the frontend components.

2. Frontend Development: Develop the user interface components, including the dashboard, maps, charts, and search functionality. Ensure a responsive and intuitive user interface design.

3. Database Implementation: Set up the PostgreSQL database and implement the necessary tables, relationships, and data storage functionality.

4. Testing: Conduct unit testing for individual components/modules to ensure their correctness. Perform integration testing to validate the interaction between components and system testing to verify the overall application functionality.

5. Documentation: Prepare comprehensive documentation for the application, including user guides, installation instructions, and API documentation.

4.3 Resource Allocation
The resources required for the implementation of the application include:

1. Development Environment: Each team member will require a development environment with the necessary software tools, including IDEs, code editors, version control systems, and database management tools.

2. Server Infrastructure: Set up servers for development, testing, and production environments. Ensure that the servers meet the system requirements and have the necessary resources, such as CPU, memory, and storage.

3. Database Server: Set up a PostgreSQL database server to host the application's database. Allocate sufficient resources to handle the expected data volume and user load.

4. Collaboration Tools: Utilize collaboration tools such as project management software, communication platforms, and version control systems to facilitate efficient collaboration and coordination among team members.

5. Testing Environment: Set up a separate testing environment that closely resembles the production

7 Testing Plan

1. Introduction
The purpose of this testing plan is to outline the approach and strategies for testing the interactive air quality monitoring application. The testing process will focus on ensuring the functionality, performance, usability, and reliability of the application. This plan covers various types of testing, including unit testing, integration testing, system testing, and user acceptance testing.

2. Testing Objectives
The main objectives of the testing process are as follows:

- Verify that the application meets the specified functional and non-functional requirements.
- Identify and fix defects, bugs, and vulnerabilities.
- Ensure the application's compatibility with different platforms and devices.
- Validate the performance and responsiveness of the application under different load conditions.
- Assess the usability and user experience of the application.
- Validate the accuracy and reliability of the air quality data displayed by the application.

3. Test Levels
The testing process will be conducted at the following levels:

- Unit Testing: Testing individual components and modules in isolation to ensure their correctness and proper functioning.
- Integration Testing: Testing the interaction between different components and modules to ensure their seamless integration and interoperability.
- System Testing: Testing the entire system as a whole to validate the overall functionality and performance of the application.
- User Acceptance Testing: Involving end users to evaluate the application's usability, user interface, and overall satisfaction.

4. Test Types
The following test types will be conducted during the testing process:

- Functional Testing: Verifying that the application functions as intended and meets the specified requirements.
- Performance Testing: Assessing the application's performance under different load conditions to ensure optimal response times and scalability.
- Security Testing: Identifying and mitigating potential security vulnerabilities, including authentication, authorization, and data protection.
- Usability Testing: Evaluating the user interface, navigation, and overall user experience to ensure ease of use and satisfaction.
- Compatibility Testing: Testing the application's compatibility with different web browsers, operating systems, and devices.
- Regression Testing: Re-testing previously tested features to ensure that changes or enhancements have not introduced new defects.
- Data Accuracy Testing: Validating the accuracy and reliability of the air quality data displayed by the application.

5. Test Environment
The test environment will consist of the following components:

- Test Servers: Set up dedicated servers to host the application for testing purposes.
- Test Database: Create a separate database instance to store test data and isolate it from production data.
- Test Data: Generate realistic test data representative of different air quality scenarios.

- Test Automation Tools: Utilize testing frameworks and tools to automate test execution and generate test reports.
- Test Devices: Test the application on different devices, including desktops, laptops, tablets, and mobile phones, running various operating systems and browsers.

6. Test Deliverables
The following deliverables will be produced during the testing process:

- Test Plan: Documenting the overall testing approach, objectives, and strategies.
- Test Cases: Detailed test cases specifying the steps, inputs, and expected outputs for each test scenario.
- Test Data: Test data sets representative of different air quality conditions.
- Test Scripts: Automated test scripts to facilitate efficient and repeatable testing.
- Test Reports: Summarizing the test results, including identified issues, test coverage, and overall application quality.

7. Test Execution and Reporting
The testing process will follow the following steps:

- Test Preparation: Set up the test environment, including servers, databases, and test data.
- Test Execution: Execute the test cases, both manually and using automation tools, following the specified test scenarios.
- Defect Tracking: Document any defects, bugs, or issues encountered during testing, including detailed steps to reproduce them.
- Test Reporting: Generate comprehensive test reports summarizing the test results, including pass/fail status, coverage metrics, and identified issues.

- Issue Resolution: Collaborate with the development team to address and fix the identified issues.
- Retesting: Perform regression testing to ensure that fixed issues do not introduce new defects.

8. Test Schedule and Resources
The testing schedule and resources will be determined based on the project timeline, available team members, and the complexity of the application. It is important to allocate sufficient time for each testing phase and involve all necessary stakeholders in the testing process.

9. Risks and Mitigation
Identify potential risks and challenges that may impact the testing process, such as time constraints, resource limitations, and technical dependencies. Develop mitigation strategies to minimize these risks and ensure the smooth execution of the testing activities.

10. Sign-off
Obtain sign-off from relevant stakeholders once the testing process is completed, all identified issues are addressed, and the application meets the specified criteria for release.

8 Bibliography

- PostgreSQL Documentation. (2022). PostgreSQL: The World's Most Advanced Open-Source Relational Database. Retrieved from https://www.postgresql.org/docs/
- Flask Documentation. (2022). Flask: A Microframework for Python. Retrieved from https://flask.palletsprojects.com/
- https://cordis.europa.eu/docs/projects/cnect/3/609023/080/deliverables/001-MYWAYD21RequirementSpecificationAndAnalysiscnectddg1h520142537023.pdf
- Robert L. Lux and C. Arden. Pope. Air quality index. U.S. Environmental Protection Agency, 2009.