POLITECNICO
MILANO 1863

Design Document

Lorenzo Carlassara
Angelica Iseni
Emma Lodetti
Virginia Valeri

Software Engineering for Geoinformatics

Academic year 2022/23

Title:	Design Document
Authors:	Carlassara L. (10601118), Iseni A. (10668862), Lodetti E. (10619244), Valeri V. (10639607)
Date:	May 24, 2023
Professor:	Giovanni E. Quattrocchi

Contents

1 Introduction

1.1 Purpose

Before diving into coding and implementing a software project, it is crucial to have a clear understanding of the design goals that our web application seeks to achieve. This is where the "Design Document" comes in.
The purpose of this document is to outline the architecture and functionality of an interactive client-server application for air quality and weather monitoring that enables users to access, query and visualize air quality data retrieved from existing public digital archives. The system consists of three main components: a database to ingest and store the selected data, a web server (backend) to expose a REST API for querying the database, and a dashboard to provide means for requesting, processing (descriptive statistics and forecasting), and visualizing data (e.g. maps, dynamic graphs, etc.).
The Design Document is intricately connected to the Requirements Analysis Specification Document (RASD), authored Carlassara L., Iseni A., Lodetti E., Valeri V. This document serves as the foundation for the Design Document and provides essential insights into the project requirements.
You can access the RASD on GitHub at the following link:
https://github.com/carls31/SE4GEO-Lab/blob/main/GRUPPO_GEO_RASD.pdf

1.2 Context and motivations

The European Commission's agreement on electronic fuels will enable the sale of heat-powered vehicles beyond 2035, when the ban on gasoline and diesel cars comes into effect. This is provided that these vehicles run on synthetic, climate-neutral fuels. Additionally, European energy ministers have approved the regulation to phase out gasoline and diesel engines by 2035 through a majority vote.

Public authorities collect air quality and weather observations in near-real time from ground sensor stations and store them in digital archives. Ground sensors data are composed of long time series of observations and sensors metadata, including coordinates, type of measured variable, etc. Often, observations from different sensors and/or providers require different patterns for data accessing, harmonization, and processing. Interactive applications and dashboards, capable of facilitating such tasks, are key for supporting both public authorities as well as ordinary citizens in data processing and visualization [1].

This tool aims to provide people with a clear and scientifically based view of the current state of knowledge about air pollutants and their potential environmental impacts on human health.

1.3 Definitions, acronyms, abbreviations

- **RASD:**Requirements Analysis and Specification Document.
- **REST:**REpresentational State Transfer.It is and architectural style for distributed hypermedia systems.
- **API:** Application Programming Interface. It is a set of rules that allow programs to talk to each other. The developer creates the API on the server and allows the client to talk to it.
- **OSM:**Open Street Map.
- **JSON:**JavaScript Object Notation.
- **DBMSs:**DataBase Management Systems.
- **SQL:**Strcutured Query Language.
- **HTML:** HyperText Markup Language.

- **CRUD:**Create, Read, Update, and Delete operations.
- **WSGI:**Web Server Gateway Interface.
- **EEA:** European Environment Agency.

1.4 Scope and limitations

The primary goal of developing this product is to raise awareness and engage users in understanding the air pollution situation in the most polluted cities in Europe. Our objective is to create an Open-Source web application that enables users to access, visualize, analyze, and save data related to air pollution. Through this application, users will have the opportunity to gain insights into the chosen data. The cities available for the analysis belongs to the following countries in Europe [2]:

Andorra	Austria	Belgium
Bosnia	Bulgaria	Croatia
Czech Republic	Denmark	Estonia
Finland	France	Germany
Gibraltar	Greece	Hungary
Ireland	Italy	Lithuania
Luxembourg	Malta	Netherlands
Norway	Poland	Portugal
Romania	Serbia	Slovakia
Slovenia	Spain	Sweden
Switzerland	United Kingdom	

Table 1: European Countries with dataset of interest

This cities will be evaluated in order to discern the differences in the main pollution indices.These pollutants can have negative impacts on human health, as they can be inhaled deep into the lungs and bloodstream and cause respiratory problems as asthma, bronchitis and cardiovascular problems. Long-term exposure can increase the risk of chronic diseases such as lung cancer and heart diseases. Furthermore,they can also affect visilibility, air quality, and the health of surronding environment.
The design document is primarily a high-level description of the application and its functionality.
It focuses on providing a conceptual understanding of the system rather than exploring specific implementation details. Therefore, finer technical aspects and code-level details are not covered in this document. It also does not address potential issues or challenges that may arise during the development process. For a deeper understanding of the software's scope and limitations, it is recommended to refer to the complementary Requirements Analysis Specification Document (RASD). The RASD provides more detailed information about the software requirements, including its capabilities and constraints.

2 Architectural design

2.1 Overview

The architectural design section of the Design Document provides an overview of the overall structure and organization of the software system being developed. It outlines the major components within the system and their interactions. This section focuses on the high-level design choices and principles that will form the architecture of the system.
It describes the major components of the system,including the client-server model,database, and external services and APIs. It also explains how these components work together to achieve the desired functionality of the application. In addition,it may highlight the technology stack used,including programming languages, frameworks, and libraries. It may also describe the design patterns and architectural styles employed to ensure system's scalability, maintainability, and performance.

2.2 Component diagrams

Component diagrams provide a visual representation of the different components of a system and how they interact with each other. They help to understand the structure and organization of the system's architecture.
The web application consists of two types of content: static and dynamic. The static content refers to information that remains unchanged and is presented to users as HTML code. This includes general information and elements that cannot be changed through user interaction. On the other hand, the dynamic content refers to information that can be modified based on user interaction. It is developed using Python code and allows for a more interactive and personalized experience.

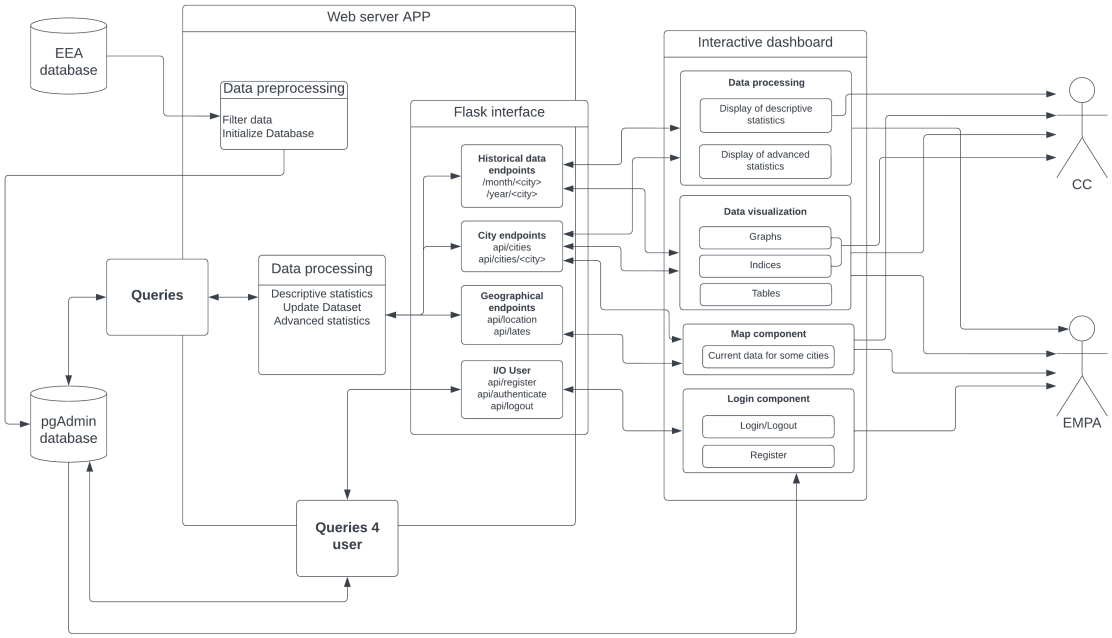

Figure 1: Architectural components diagram

The software's architecture consists of a three-layer system composed of a Web Server (HTTP Server), a Logical Server (Application Server) and a Database Server. The software is also divided into three main programs:

Database Configuration Script: this script is responsible for creating the database structure. Setting up the database configuration is essential to establish a server that can store, retrieve, and manipulate the necessary data according to user requirements.In our configuration script, we develop the code that enables us to fetch and manipulate the data that needs to be stored in the database. After importing the dataset, we obtained the data by making a request to the REST API using the "request" library. Subsequently, we transformed the data into a JSON file using the "json" library. Once this step was completed, we converted the data into a Pandas dataframe using the "pandas" library. Finally, we enriched the dataframe by incorporating geometry attributes and converting it into a geodataframe using the "geopandas" library. We are utilizing the EEA database for data preprocessing tasks. The EEA database contains a wealth of environmental data and information that is relevant to the project. It is initialized to have three tables, city, users and contacts.Additionally, we are using the pgAdmin database, an adiministrator and management tool for PostgreSQL, to execute queries and perform operations. During data processing, we are employing various techniques such as descriptive statistics, dataset updates, and advanced statistical analyses to gain valuable insights from the data.

Flask Application Script: this script is built using the Flask framework.It contains the Python code that implements the necessary logical operations to meet the software requirements. It also provides functionalities to easily read and write JSON documents. We are integrating the Flask interface into our project, which provides endpoints for different functionalities. For instance, we have implemented the "misc" endpoint for general API-related interactions, such as contact information. Furthermore, we have developed historical data endpoint that allow users to access data for specific months,years and cities. Similarly, we have created city endpoints and geographical endpoints to provide data specific to cities and geographic locations. Moreover we have implemented an I/O user endpoint that enables users to register and access specific functionalities within the system.

Interactive Dashboard: finally, we have developed an interactive dashboard with Jupyter NoteBook that provides a user-friendly and visually appealing interface for data exploration and analysis. The interactive dashboard allows users to interact with the data, visualize insights, and customize their data exploration experience.
To improve the user experience, the interactive dashboard combines numerous data processing features. It offers both advanced statistics and descriptive statistics to deliver insightful analyses of the data. Through interactive visualizations such as graphs, indeces, and tables, users may explore and analyze the data. The dashboard also has a map feature that shows the most recent information for particular cities, enabling users to visually analyze regional trends and patterns. The dashboard includes a login component that supports login, logout, and registration features to guarantee secure access. In the image shown in the Figure, you can get a visual overview of everything that was just explained.

3 Software Design

3.1 Database design

The project will utilize data obtained from the API of the cities provided by the website: https://aqicn.org/. This data includes information about the geographical location of points and the level of pollution. This data will be preprocessed by our system before being managed by the user.
The software will use and interact with DBMS and perform CRUD operations on the PostgreSQL database. PostgreSQL is a powerful, open source object-relational database system that uses and extends the SQL language combined with many features that safely store and scale the most complicated data workloads. Databases will incorporate primary keys, which are unique identifiers assigned to each entity. This primary key will allow for clear identification of individual entities.
Setting up the database configuration is essential in order to have a server where the required data can be stored, retrieved, and from which it is possible to be manipulated at the user's need. To interact with PostgreSQL effectively, we need to perform a series of operations that involve establishing a connection, sending SQL commands or making changes, and finally, closing the connection to the server.
To facilitate this interaction between our software and PostgreSQL, we rely on specific libraries and functions that optimize the process:

Library:

- Psycopg2
- Sqlalchemy

In this way are created empty tables (table 4 describes these tables) and then they are filled with the appropriate data to start with.

- **STATION**: This table has the following columns: StationCode, StationName, StationAltitude, and Country. It stores information related to different monitoring stations.
- **POLLUTANT**: This table includes columns such as name, StationCode, DataTimeBeg, DataTimeEnd, DataTimeUpdate, and ValuePollutant. It stores data about various pollutants measured at different stations.
- **USEREQUEST**: This table consists of columns like UserID, DataSentRequest, StationCode, PollutantName, DataTimeBeg, and DataTimeEnd. It stores user requests for specific data, including the requested station, pollutant, and time period.

3.2 REST APIs

A REST API allows communication between different software applications using HTTP methods such as GET, POST, PUT, and DELETE.

- **GET:**The GET method is used to retrieve data from a server. It's a read-only operation.
- **POST:**The POST method is used to submit data to be processed by the server. It accepts the entity enclosed in the request as a new subordinate of the resource identified by the Request-URI.
- **DELETE:**The DELETE method is used to delete a specific resource from the server.
- **PUT;**The PUT method is used to update or replace an existing resource on the server. It requests that the enclosed entity be stored under the supplied Request-URI

Flask, a popular Python web framework, provides a straightforward way to build REST APIs.The Python code is responsible for handling the requests made by users through the web server. It interacts with the web server using the WSGI interface, which allows the code to receive and respond to HTTP requests. To design a REST API using Flask and Python, you can follow these steps:

Creating Dataframe: the code is responsible for downloading the dataframe file and loading it into a pandas dataframe for further processing.

```
import pandas as pd
import os
from se4g_utils import download_request
# website containing countries & pollutants list
'''https://discomap.eea.europa.eu/map/fme/latest/files.txt'''

# Set countries to be downloaded
COUNTRIES = ['AD','AT','BA','BE','BG','CH','CY','CZ','DE','DK','EE','ES','FI','SE']
# Set pollutant to be downloaded
POLLUTANTS = ['SO2','CO','PM10']
# Download and get the dataframe file name
file_name = download_request(COUNTRIES, POLLUTANTS)
print(file_name)
# Get the folder where the dataframe file is located
folder_out = 'data'
full_file = os.path.join(folder_out, file_name)
# Read the file
df = pd.read_csv(full_file)
```

Data Download and Directory Management: The provided code defines a function *download_request* that is responsible for downloading CSV files from a specified URL and saving them locally in a specified folder.This function takes three arguments: COUNTRIES, POLLUTANTS, and folder_out.First,it checks to see if a directory named folder_out already exists. If it does not exist, the directory is created.Next,it creates a new subdirectory in folder_out using the current date and time in the format of daymonthyear_hour_minute_second. The name of the created subdirectory appears as output, indicating that the directory was successfully created. Next,it iterates through the country and contaminant combinations defined in the COUNTRIES and POLLUTANTS lists. For each combination,it creates a download URL using the ServiceUrl and the country and pollutant names. The file is then downloaded from the constructed URL using the *requests.get* method. The contents of the downloaded files are stored locally in the subdirectory created earlier with the country and pollutants names as filenames. After iterating through all the combinations and downloading the files, the code prints "Download finished" as the final output, indicating that the entire download process has been completed.

```
import os
import requests
import pandas as pd
from datetime import datetime
import ipywidgets as widgets
from IPython.display import display
from sqlalchemy import create_engine

```

```
# Download and get the dataframe file name
def download_request(COUNTRIES=
    ['AD','AT','BA','BE','BG','CH','CY','CZ','DE','DK','EE','ES','FI','SE'],
               POLLUTANTS= ['SO2','CO','PM10'],
               folder_out = 'data'):
   print ('-------------------------------------------------------------------------')
   # Set download url
   ServiceUrl = "http://discomap.eea.europa.eu/map/fme/latest"

   dir = datetime.now().strftime("%d-%m-%Y_%H_%M_%S")

   if not os.path.exists(os.path.join(folder_out, dir)):
      if not os.path.exists(folder_out):
         os.mkdir(folder_out)
      os.mkdir(os.path.join(folder_out, dir))
      print(dir,'directory created')

   for country in COUNTRIES:
      for pollutant in POLLUTANTS:
         fileName = "%s_%s.csv" % (country, pollutant)
         downloadFile = '%s/%s_%s.csv' % (ServiceUrl, country, pollutant)
         #Download and save to local path
         print('Downloading: %s' % downloadFile )

         file = requests.get(downloadFile).content
         full_file = os.path.join(folder_out, dir, fileName)

         output = open(full_file, 'wb')
         output.write(file)
         output.close()
         print ('Saved locally as: %s ' % fileName)
         print ('-----')
   print ('Download finished')
   return dir
```

Build the Dataframe: This provided code defines a function *build_dataframe* that is responsible for building a pandas DataFrame with a specific structure from the downloaded CSV files.The function starts by creating an empty list called dfs to store the DataFrames. It then iterates over each country in the COUNTRIES list and each pollutant in the POLLUTANTS list. For each combination, it creates the filenames based on the countries and pollutantants values.Next, we check if the first line starts with <!DOCTYPE html, indicating that the file is not an HTML document but a valid CSV file. If it doesn't start with <!DOCTYPE html, the file is considered valid, and the code proceeds to read the CSV content using the *pd.read_csv* function. The resulting DataFrame is then appended to a list called dfs. After the iteration through all the combinations and reading the valid CSV files, *pd.concat*is used to concatenate all DataFrames in the dfs list into a single DataFrame df_all.Finally the function returns DataFrame df_all, which can be used for further analysis.

```
# Build the dataframe with the required structure
def build_dataframe(dir,
              COUNTRIES =
                  ['AD','AT','BA','BE','BG','CH','CY','CZ','DE','DK','EE','ES','FI','SE'],
```

```
                POLLUTANTS = ['SO2','CO','PM10'],
                 folder_out = 'data',
                df_columns = ['station_code',
                              'station_name',
                             'station_altitude',
                             'network_countrycode',
                             'pollutant',
                             'value_datetime_begin',
                             'value_datetime_end',
                             'value_datetime_updated',
                             'value_numeric'] ):
    dfs = []
    for country in COUNTRIES:
       for pollutant in POLLUTANTS:

          fileName = "%s_%s.csv" % (country, pollutant)
          with open(os.path.join(folder_out, dir, fileName), 'r') as file:
             first_line = file.readline().strip()

          if not first_line.startswith('<!DOCTYPE html'):
              #first_line.startswith('network_countrycode'):
             #print(fileName,'exist')
             df_temp = pd.read_csv(os.path.join(folder_out, dir, fileName))
             dfs.append(df_temp[df_columns])

    df_all = pd.concat(dfs, ignore_index=True)
    print ('Database assembled')
    return df_all
```

Update the final Dataset: This code defines a function called *update_dataset* that updates an existing dataset with new data. The code first checks if a file named "se4g_pollution_dataset.csv" exists in the specified folder_out directory(the output folder where the dataset is stored) using *os.path.isfile*. If the file exists, it proceeds with the update process and it enters the conditional block. Inside the block, it reads the existing dataset from the file using *pd.read_csv* and assigns it to the variable df. The 'value_datetime_begin' column is converted to datetime format for both the existing dataset(df) and the new dataset (new_df).Then the function filters the rows from the new dataset that have 'value_datetime_begin' values greater than the maximum value in the existing dataset df. This ensures that only the latest and relevant data is added to the dataset. If there are filtered rows, the function concatenates the existing dataset and the filtered rows into a new DataFrame called updated_df. The updated dataset is then saved to the file, using *updated_df.to_csv*. After the dataset has been updated, the function prints a success message indicating whether the dataset was updated successfully.

```
# Update the final dataset
def update_dataset(new_df, folder_out = 'data'):

    fileName = "se4g_pollution_dataset.csv"
    full_path = os.path.join(folder_out, fileName)

    if os.path.isfile(full_path):
       # Open the CSV dataset
       df = pd.read_csv(full_path)
```

```
        df['value_datetime_begin'] = pd.to_datetime(df['value_datetime_begin'])
        new_df['value_datetime_begin'] = pd.to_datetime(new_df['value_datetime_begin'])

        # Filter rows from new_df based on the datetime
        filtered_rows = new_df[new_df['value_datetime_begin'] >
            df['value_datetime_begin'].max()]
        if not filtered_rows.empty:
            # Update the dataset by adding the filtered rows
            updated_df = pd.concat([df, filtered_rows], ignore_index=True)

            # Save the updated dataset
            updated_df.to_csv(full_path, index=False)
            print("Dataset updated successfully")

    else:
        new_df.to_csv(full_path, index=False)
        print("Dataset created successfully")
    return fileName
```

User login: The code below defines a function called *login_required* that implements an user login functionality in Jupyter Notebook to create an interactive interface where users can enter their login credentials. By using widgets, the code generates input fields for the username and password, along with a login button. This displayed widgets enable users to provide their login information. Within the function *handle_login_button_click* the values entered in the username and password input fields are retrieved and stored in the variables username and password, respectively. If the entered username is "postgres" and the password is "carIs3198", the code establishes a connection to a PostgreSQL database. This is achieved by utilizing the *create_engine* function. Once the database connection is established, further operations can be performed using the obtained connection object.In case of a successful login, the code indicates a successful connection to the database by printing "connected with localhost" and returns the database connection object for further use.

```
# User login
def login_required():
    user = widgets.Text(
        placeholder='Type postgres',
        description='Username:',
        disabled=False
    )

    psw = widgets.Password(
        placeholder='Enter password',
        description='Password:',
        disabled=False
    )

    login_button = widgets.Button(description="Login")
    display(user, psw, login_button)

    def handle_login_button_click(button):
        username = user.value
        password = psw.value
```

```
21
22          # Check if username and password are valid
23          if username == "postgres" and password == "carIs3198":
24              # Connect to the database
25              engine =
                    create_engine('postgresql://'+username+':'+password+'@localhost:5432/se4g')
26              con = engine.connect()
27              # Perform any necessary database operations
28              # ...
29              # Return the database connection or perform any other actions
30              print('connected with localhost')
31              return con
32          else:
33              print('it does not work')
34
35      login_button.on_click(handle_login_button_click)
```

3.3 Dashboard design

When designing a dashboard for a project, it is important to consider both functional and aesthetic aspects. Here are some important elements to include in our dashboard:

- Dropdowns and Radio items
- Scatter Plot
- Time Series Graphs
- Slider
- Callback Functions

In general, the dashboard design provides an interactive data exploration interface where users can select X-axis and Y-axis variables,filter the data by year and view the corresponding scatter plot and time series graphs.

4 Implementation and test plan

References

[1] G. E. Quattrocchi, “Se4geo project,” 2023.

[2] “Download of utd air quality data,” 2023-04-05. [Online]. Available: https://discomap.eea.europa.eu/map/fme/AirQualityUTDExport.htm