



POLITECNICO
MILANO 1863

Software Engineering
for Geoinformatics 2022

Design Document

*Bigai S., Calcaterra M., Leonardi J., Rzewuski M.,
Zallem N.*



AIR Cloud

Deliverable:	DD
Title:	Design Document
Authors:	Bigai S., Calcaterra M., Leonardi J., Rzewuski M., Zalleme N.
Version:	2.0
Date:	June 7, 2022
Download page:	https://github.com/Rkomi98/S4GEO
Copyright:	Copyright © 2017, Bigai S., Calcaterra M., Leonardi J., Rzewuski M., Zalleme N. – All rights reserved

Contents

List of Tables	4
1 Introduction	5
1.1 Design Document	5
1.2 Product Description	6
2 Project database	7
2.1 AQICN Database	7
2.2 PostgreSQL Database	8
3 System overview	10
4 Software structure	11
4.1 Presentation Tier	11
4.2 Application Tier	12
4.2.1 Template Engine	12
4.2.2 Software functionalities	14
4.3 Data Tier	20
5 User interface	21
5.1 Navigation Bar	21
5.2 HomePage	21
5.3 More Info	22
6 User cases application	24
6.1 Register	24
6.2 Log In	25
6.3 Log Out	25
6.4 Query Data	26
6.5 Geographic visualization	27
6.6 Data manipulation	28
6.7 Analyze data and Export Analysis	29
7 Organization	31

List of Figures

1	Structure of the API response	7
2	Database Structure	8
3	Software structure	11
4	Template engine scheme	12
5	Navigation Bar	21
6	Home page	21
7	More Info page. As you can see it is slit in "General Idea" and "Data" section	22
8	Web page structure	23
9	Register	24
10	Login	25
11	New Project	26
12	Choosing the City	27
13	Choosing the Data Type	27
14	The result of a sample query	27
15	UML of all User cases structure	30

List of Tables

1	User table attributes definition	9
2	Project table attributes definition	9
3	Home Page	21
4	More Info Section	22
5	Register Page	24
6	Log In Page	25
7	Log Out Button	25
8	Query Data	26
9	Geographic visualization	27
10	Data manipulation	28
11	Analyze data and Export Analysis	29
12	Effort spent with details	31

1 Introduction

1.1 Design Document

Before starting coding and implementing a software, an important step is to have a clear idea of the design goals that our web-application is going to accomplish. All these goals should be established in this specific Design Document.

The key reason of writing this document is defining both structure and organization, that will be followed through the period of creating until finalizing the software project. Doing this, confusions and failing could be avoided. With less words, it is an important guideline document for the engineering team, in which they can rely in any further future steps.

The Software Design Document is a technical description, that serves both on developer and client side. The goal of the developer is to satisfy the client objective but at the same time to help himself with providing an efficient workflow. Each person involved in a project must have the knowledge upon its contents.

It is important to emphasize the fact that the Design Document is built upon the (RASD) Requirements Analysis Specification Document, written by Bigai S., Calcaterra M., Leonardi J., Rzewuski M., Zallemini N. You can find this document in the following [GitHub link](#).

The connection between the above-mentioned documents helps us to be careful with not avoiding any important functionality.

We should keep in mind the fact that the Design Document should be presented as a requirements and function document and not an implementation specification. The characteristics specified would not be changed or described in another way but will remain an implicit knowledge for the development team.

At the very least, it should be a description of the application, criteria for completion and milestones. More specifically it will include the following characteristics:

1. **Project Database:**

This is the most important part of the project we are developing. Database design can be defined as a collection of tasks or processes that enhance the designing, development, and maintenance of data management system. With other words, it is the most complex part where the designer determines what data must be stored and how the data elements are connected and interacting with each other.

2. **System Overview:**

This section will provide a general description of the structure and functionality of the software system.

3. **Software Structure:**

Our software, as a traditional client-server application, will be established as a three-tier architecture. This structure will make possible the interaction between the client and the server, considering both static and dynamic units. According to this architecture the application is organized into three logical and physical layers or tiers, which are:

- Presentation tier
- Application tier
- Data tier

Further details will be explained in the corresponding section in the body of the document.

4. **User cases application:**

This section will include the use cases and requirements map for each component of the software. Further details are prementioned in the RASD Document.

5. Organization:

This is a key ingredient that makes the project a successful one and as mentioned above helps the design team to easily overcome confusions and fails. The development team is composed of 5 students, who equally participate in each step of the project. However, an important assignment, in order to prevent miscommunication in the workflow, is to divide the work, activities and responsibilities to each engineer student of the group.

The description should also provide an answer to some key functionality questions listed below:

- What does the application do?
- What are the operations we can do and the executions after opening the web-application?
- Is the user allowed to create entries and if yes what are the limitations?

1.2 Product Description

The purpose of developing this product is to inform and involve the users about the air pollution situation in 5 most polluted cities¹ in Europe by means of an Open-Source web application. This web-application will allow users to query, visualize, analyze data, and save their analysis of the chosen data. On the website the user will be able to find general information regarding this environmental issue and also write their own comments on the data analysis.

The information provided by the web application will be both static and dynamic. In the static group are included all the information that will not change with the user interaction. This information will be provided by HTML code.

On the other hand, dynamic refers to all information that will change with the user. With adding dynamic elements, we create a truly interactive website experience for the users. This part of the software will be implemented using JavaScript and Python language.

¹This cities have been chosen by us considering the most populated and polluted cities in Europe, according to <https://www.iqair.com/> and <https://air8.tech/most-polluted-cities-in-europe-in-2021/>

2 Project database

The data for this project, including information about point positioning and pollution level, will be retrieved from the API of the cities coming from the following website: <https://aqicn.org/>. They are preprocessed² by us and only then managed by the user.

The web app will then interact with DBMS and perform operations on the PostgreSQL database to store all the requests of the user to increase the quality of our application. The advantages for storing the data in a PostgreSQL database as opposed to fetch them directly from AQCIN include:

1. Reduce the probability of data loss.
2. Store the same project on multiple devices. For this we need an external database subscription and we haven't it yet.

2.1 AQICN Database

At the following [link](#) it is possible to see an example of data provided by the api in form of JSON file for one of the city we used for our analysis.

We focused on the real time data (which can be found in "iaqi" section of the API) on the section geo, where there are all the information about the geo-reference of the points and the data.

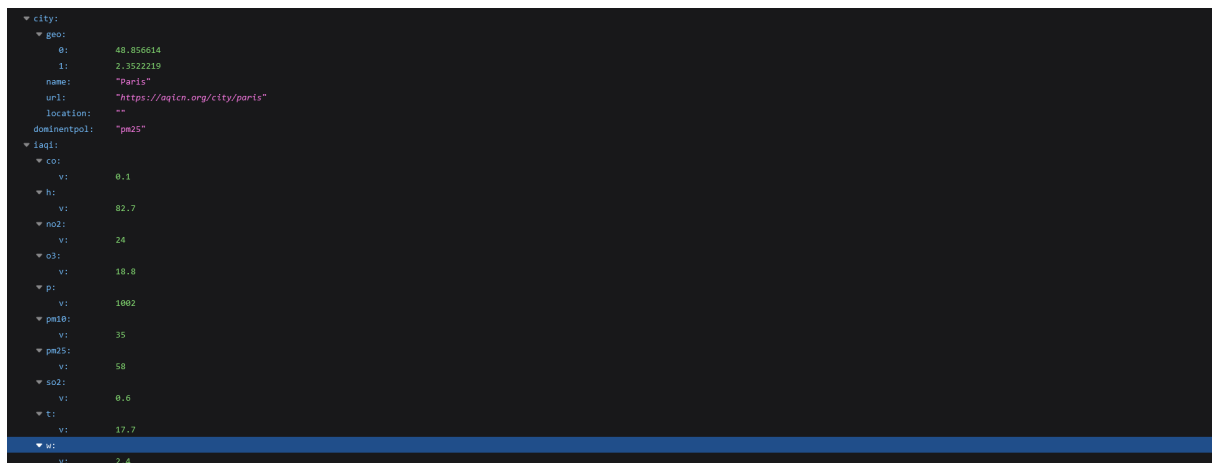


Figure 1: Structure of the API response

A data entry contains the following attributes:

- "status": if ok, it means that at least one sensor is working
- "data": is split in:
 - "aqi": pollution score that a city can have less is, better is
 - "idx": index, which is different for all the city.
 - "attributions": all the information, so logo and website about the servers which are providing their API.
 - "city": geographical coordinates and name of the place where there is the sensor where the data comes from.
 - "iaqi": contains all the index we need
 - * "h": value of the relative humidity

²We selected only data we are interested in

- * "no2": Concentration of NO_2
 - * "o3": Concentration of O_3
 - * "p": Pressure measured
 - * "pm10": Concentration of PM10
 - * "pm25": Concentration of PM2.5
 - * "t": Temperature
- "time": information about date and time and UTC
 - "forecast": all the daily value of O_3 , PM10, PM2.5, UV index forecast for 4 days after the day when the user gets the data
 - "debug": which gives you the information about the last attempt to connect to the API

2.2 PostgreSQL Database

We decided to store the users information (only name and password) and data retrieved on a local server of PostgreSQL. We chose it, because it is an Open Source relational database management system and because it supports the operation with georeferenced data. It is possible, thanks to PostGis, to work with geometry data and perform operation Polygons, LineString, Points, . . . , or spatial operations with Raster and vectors, (e.g. buffer, union, clipping, . . .), write predicates to measure intersection between geometries and spatial operations to measure area, length and so on.

Furthermore it is possible to work and query data with Python thanks to the library psycopg2.

The structure of the database should be as follows:

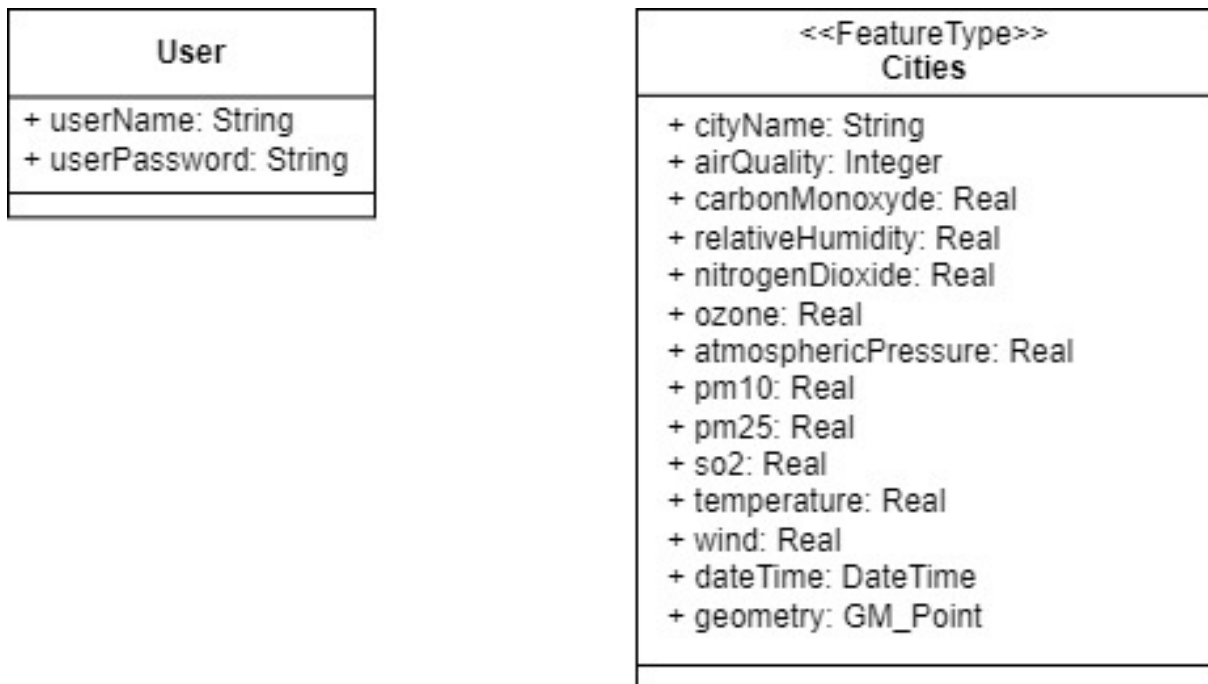


Figure 2: Database Structure

The PostgreSQL-PostGIS database will be composed by tables with the following attributes:

User Table	
user_id	Unique identifier of the user from the Database point of view
user_name	Name of the user
user_password	Password of the user

Table 1: User table attributes definition

Cities Table	
air_quality	Score of the air city quality
city	Name of the city
carbon_monoxide	Concentration of CO in the air
relative_humidity	Measurement of the humidity in the air
ozone	Concentration of O_3 in the air
atmospheric_pressure	Atmospheric pressure measured in the city
PM10	Concentration of PM_{10} in the air
PM25	Concentration of $PM_{2.5}$ in the air
sulphur_dioxide	Concentration of SO_2 in the air
temperature	Temperature in the city
wind	Velocity of the wind in the city
date	Date and time of the measurement plus the time zone
x	Latitude of the station in the city
y	Longitude of the station in the city
geometry	Kind of geometry
ID	ID of the user who performed the request
nitrogen_dioxide	Concentration of NO_2 in the air

Table 2: Project table attributes definition

3 System overview

The system is split on multiple pages which are highly interactive, based on a client-server architecture. On this system:

- The resources are dynamically loaded from the web server in response to users' actions. The code on the server includes the services that are going to be provided to the client. As a response to clients requests the web browser will display the information taken by the server.
- When the browser requests the URL, from the server, the server responds with the page containing HTML, CSS, JS and Python code.
- The client then requests the data from the server, by invoking the corresponding API by a query, which retrieves the data from the data base and returns the information in tabular format to the client. This enhances the user experience.
- The information will be requested by the server in real time and the user will be able to interact with it and visualize it in different ways (tabular/map).
- The web application uses Python (in particular the libraries Pandas, Geopandas and Scikit learn) to implement all the statistical analysis.

4 Software structure

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

- Presentation Tier also called Web Server
- Application Tier also called Logical Server
- Data Tier also called Database Server

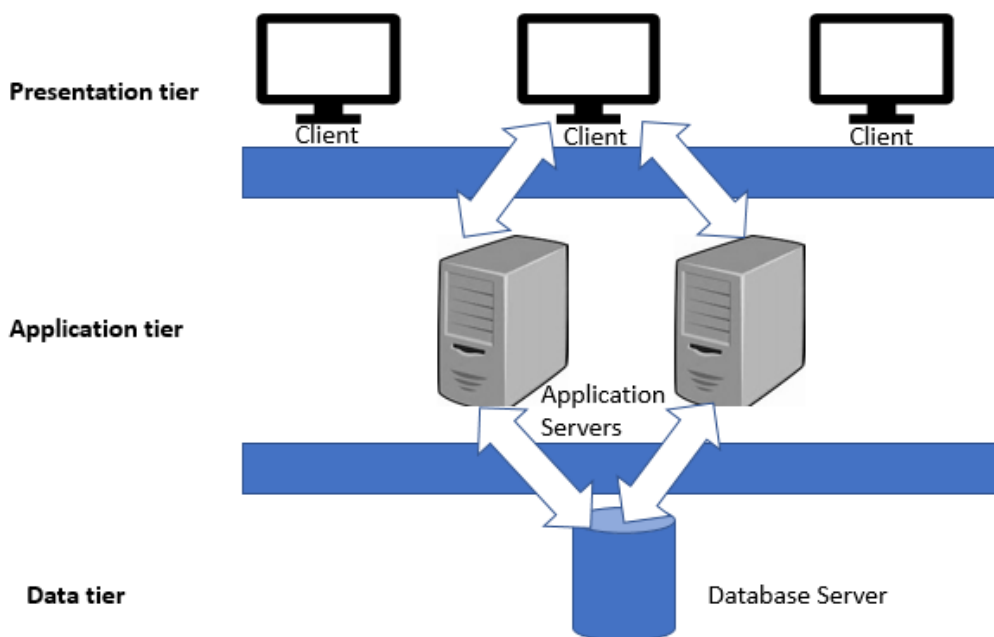


Figure 3: Software structure

4.1 Presentation Tier

This top-level tier is the communication layer of the application, where the users interact with the application. Its purpose is to display information to the users and, at the same time, to collect information from them.

The presentation tier will run on a web browser. The client will be able to communicate with HTTP Server through client requests such as: URL, clicking, filling forms etc. The HTTP server has as main purposes the processes of storing, processing, and delivering web pages to clients.

As an answer to users' requests, the HTTP server will provide a response to the user browser. The response will be developed by using HTML, CSS, JavaScript and Python.

- **HTML code:** this part of the software will contain the skeleton of the webpage, with all the connection between pages. In addition to text content, they will include images, style sheets and scripts.
- **CSS:** thanks to CSS, the website will be graphically clear, attractive and a comfortable environment for users.
- **JavaScript:** Will help to add interactivity to the web application.

4.2 Application Tier

This tier is known as the logic tier because it will provide all the logical operations that will be needed in order to fulfill the software requirements. All the information collected in the presentation tier are generated by requesting to the application server the information needed for producing the required web page. The application tier can also add, delete, or modify data in the data tier.

To develop the Application Server, as a coding language will be used Python together with SQL.

All the function that will be used in the software will be provided by Python code. It will be used to interact with the DBMS, performing operations between pages on the website, managing the interactive maps that are the main source of display for web applications together with the analysis that will be performed by Machine Learning algorithm.

The following libraries will be mandatory (they are divided in 3 different topic):

- **DBMS:** Sqlite, geoalchemy, postgresql, psycopg2, sqlalchemy.
- **Website developing:** Flask
- **Data analysis:** “Pandas, GeoPandas”, for data and geodata manipulation.

4.2.1 Template Engine

A template engine helps the developers to create a format based upon the desired output method. It allows us to use some data and programming constructs such as conditional and for loops to manipulate the output.

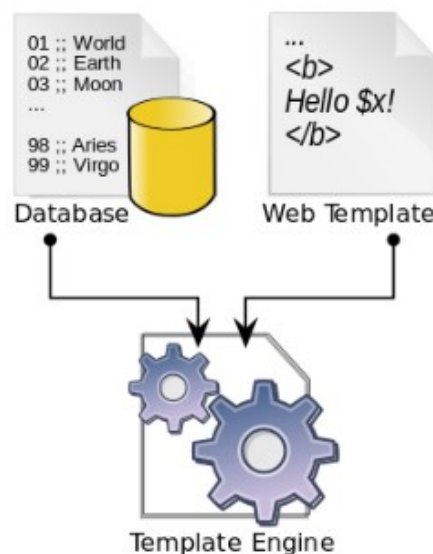


Figure 4: Template engine scheme

- **Flask Application**

Flask is an API of Python that allows us to build up our web application, in the most easy and less complex way. More specifically, it is a web framework, so a collection of modules and libraries, that contains all the necessary low-level codes such as protocols, thread managements. It uses templates to expand all web application functionalities.

Flask is based on Werkzeug WSGI toolkit and Jinja2 template engine.

- **Werkzeug toolkit**

Werkzeug is a collection of libraries used to build compatible web applications. This toolkit

is used to implement requests, response objects, and utility functions. It is used as a basis of Flask framework.

– Jinja2

Jinja2 is a Python template engine written as a self-contained open source project. This property makes it useful because it can be used as a dependency by other code libraries.

- **Jinja Template Engine**

Python templates are enabled using the Jinja2 template engine. We create a template engine, where we define static parts and dynamic parts.

Jinja Templates are just .html files, which are found in the */templates* directory in our Flask Project. Is the Jinja engine that dynamically builds these .html pages using familiar Python concepts, such as variables, loops, lists, functions, based on logic and context.

Jinja empowers us to do snippets between these pages. The rendering function later combines the templates with data. Templates will not execute without a Flask Application to serve them.

- **HTML**

In our WSGI application, the template engine generates the HTML output response when an HTTP request comes in with the particular URL.

Hypertext Markup Language is the standard markup language, that is downloaded from a web server to a web browser and is used to display the web application content. For this purpose we integrated codes in HTML files.

- **CSS**

Cascading Style Sheet is a style sheet language and not a programming or markup language, like HTML is.

CSS makes the designs easier, for the developers, to implement in the web application. CSS is what we use to style the HTML elements of our code.

Here we are listing the .html files for generating the web request and displaying retrieved data from database. To make this possible, HTML code is combined with python. Firstly, we create templates folder in which all the html files are connected with python code. There are:

- index.html: in this file is written the code for the Home page of the application. It is connected to the login page (login.html), to the more info page (generic.html) and to the project page (createProject.html)
- generic.html: in this file is written the general information about why we chose to build this application
- elements.html: in this file is written all the information about all the data elements, placed in the "Data" section, part of "More Info" section.
- createProject.html: this file provides the code that makes the user able to retrieve, sort and filter, analyze, visualize and export data collected from the sensors placed in the 5 cities we mentioned at the beginning of this document.

In the auth folder you can find:

- login.html: Contains the code of the Login In page. According to it, a user who is already in the database of the application can sign in the application. As an input serves the user's name and the password.
- register.html: In case the user is not registered before, this html page will appear, requiring for a name and password.

Furthermore we should mention the two most important python files.

- **main.py file:** It is the file where we define all the functions that the application needs to work. Only if we run this file the web server will run.
There are some general functions (that you can see from 2 to 12) and other functions provided by Flask, which are used to render the template in each HTML page, as you can see in 1.

```
1     return render_template('index.html', posts=posts)
2
```

Listing 1: render template

the html page will change depending on the route where the user is.

The first time the application is used on a device, it is mandatory running the createSchema.py

- **createSchema.py file:** It is used to build the database schema on the database S4G (more details in the next subsection 4.3).

At the end you can find a txt file, named dbConfig.txt, with the credential of the local database we used.

4.2.2 Software functionalities

In this subsection we will enumerate all the main functions of our application, showing you the code used.

Connection to the Database All the credentials are contained in the *dbConfig.txt* file. Here the user can find the Database name, the password and the user name.

```
1 from flask import (
2     Flask, render_template, request, redirect, flash, url_for, session, g
3 )
4 from psycopg2 import (
5     connect
6 )
7
8 def get_dbConn():
9     if 'dbConn' not in g:
10         myFile = open('dbConfig.txt')
11         connStr = myFile.readline()
12         g.dbConn = connect(connStr)
13     print(g.dbConn)
14     return g.dbConn
15 def close_dbConn():
16     if 'dbConn' in g:
17         g.dbConn.close()
18         g.pop('dbConn')
```

Listing 2: Connection to the Database

Once the file is opened, the software connects to the database with the function **connect(connStr)**

API connection This function is used to connect the API to our website.

```
1 import requests
2 import json
3
4 def get_json_API(city):
5     link = "https://api.waqi.info/feed/" + city + \
6         "/?token=6b937a38a89b944787d29b8afca33fe1cf375bd1"
7     response = requests.get(link)
8
9     if str(response) != "<Response [200]>":
10         txt = "Invalid city name. No data found for: " + city
```

```

11         raise Exception(txt)
12
13     raw_data = response.text
14     data = json.loads(raw_data)
15     return data

```

Listing 3: API connection

Note that the token is constant, but the city can change, depending on the request of the user. In our application the cities are a close number (5), but the can be potentially many more.

The response is registered with the command `requests.get(link)`, then it is transformed in text thanks to `response.text` and at the end in json format with the function `json.loads(raw_data)`.

From Forecast json data to Dataframe This function is used to create a Dataframe, starting from the json file retrieved by the `get_json_API()` function, which receives as input the name of the city.

```

1 import pandas as pd
2
3 def get_forecast_data_to_DB(city):
4     data = get_json_API(city)
5
6     # from JSON to Pandas DataFrame: creating the forecast table
7
8     # extracting all the factors separately:
9     data_df_forecast_o3 = pd.json_normalize(
10         data['data']['forecast']['daily']['o3'])
11     data_df_forecast_pm10 = pd.json_normalize(
12         data['data']['forecast']['daily']['pm10'])
13     data_df_forecast_pm25 = pd.json_normalize(
14         data['data']['forecast']['daily']['pm25'])
15     data_df_forecast_uvi = pd.json_normalize(
16         data['data']['forecast']['daily']['uvi'])
17
18     # preparing each of them to be merged later:
19     data_df_forecast_o3 = data_df_forecast_o3.rename(
20         columns={'avg': 'avg_o3', 'max': 'max_o3', 'min': 'min_o3'})
21     data_df_forecast_o3.insert(0, 'day', data_df_forecast_o3.pop('day'))
22
23     data_df_forecast_pm10 = data_df_forecast_pm10.rename(
24         columns={'avg': 'avg_pm10', 'max': 'max_pm10', 'min': 'min_pm10'})
25     data_df_forecast_pm10.insert(0, 'day', data_df_forecast_pm10.pop('day'))
26
27     data_df_forecast_pm25 = data_df_forecast_pm25.rename(
28         columns={'avg': 'avg_pm25', 'max': 'max_pm25', 'min': 'min_pm25'})
29     data_df_forecast_pm25.insert(0, 'day', data_df_forecast_pm25.pop('day'))
30
31     data_df_forecast_uvi = data_df_forecast_uvi.rename(
32         columns={'avg': 'avg_uvi', 'max': 'max_uvi', 'min': 'min_uvi'})
33     data_df_forecast_uvi.insert(0, 'day', data_df_forecast_uvi.pop('day'))
34
35     # merging all the factors in one prediction table:
36     o3_pm10 = pd.merge(data_df_forecast_o3,
37         data_df_forecast_pm10, how="outer", on=["day"])
38     o3_pm10_pm25 = pd.merge(
39         o3_pm10, data_df_forecast_pm25, how="outer", on=["day"])
40     final_forecast_table = pd.merge(
41         o3_pm10_pm25, data_df_forecast_uvi, how="outer", on=["day"])
42     return final_forecast_table

```

Listing 4: From Forecast json data to Dataframe

We renamed the columns to make them clearer.

From Forecast Dataframe to HTML table We used `get_forecast_data_to_DB` function to get dataframe, starting from the city name.

```

1 def get_forecast_data(city):
2     data = get_json_API(city)
3     final_forecast_table = get_forecast_data_to_DB(city)
4
5     #extracting lon and lat:
6     data_df = pd.json_normalize(data['data'])
7
8     final_forecast_table['lat'] = data_df['city.geo'][0][0]
9     final_forecast_table['lon'] = data_df['city.geo'][0][1]
10
11     final_forecast_table_html = final_forecast_table.dropna(thresh=6).to_html(
12         index=False)
13     final_forecast_table_html = final_forecast_table_html.replace("class=\"
14         dataframe\"", "id=\"forecastTable\"")
15     return final_forecast_table_html

```

Listing 5: From Dataframe to HTML table

In this function we redefine some columns to make them clearer and then we drop the null rows. At this point we use the `.to_html` function to transform the dataframe in a table in HTML. We will need an other function to call the table in HTML.

Get Real time json data to dataframe This function prepares data to be sent to the Database

```

1 from pyproj import Proj, transform
2
3 def get_data_to_DataFrame(city, User):
4     data = get_json_API(city)
5
6     # from JSON to Pandas DataFrame: creating the real time data table
7     data_df_day = pd.json_normalize(data['data'])
8     data_df_day["date"] = data_df_day["time.s"] + data_df_day["time.tz"]
9
10    # dropping the unnecessary columns:
11    data_df_day = data_df_day.drop(columns=['idx', 'attributions', 'dominantpol',
12        'city.url', 'city.location', 'time.v', 'time.iso',
13        'forecast.daily.o3', 'forecast.daily
14        .pm10', 'forecast.daily.pm25', 'forecast.daily.uvi', 'debug.sync'])
15
16    # renaming the columns we will be using for clarity:
17    data_df_day = data_df_day.rename(columns={'city.name': 'city',
18        'aqi': 'air_quality',
19        'iaqi.co.v': 'carbon_monoxyde',
20        'iaqi.h.v': 'relative_humidity',
21        'iaqi.no2.v': 'nitrogen_dioxide',
22        'iaqi.o3.v': 'ozone',
23        'iaqi.p.v': 'atmospheric_pressure',
24        'iaqi.pm10.v': 'PM10',
25        'iaqi.pm25.v': 'PM25',
26        'iaqi.so2.v': 'sulphur_dioxide',
27        'iaqi.t.v': 'temperature',
28        'iaqi.w.v': 'wind',
29        'time.s': 'date_and_time',
30        'time.tz': 'time zone'
31    })
32
33    # creating two columns for geographical coordinates instead of one for
34    # easier access:

```



```

32 data_df_day['x'], data_df_day['y'] = transform(Proj(init='epsg:4326'), Proj(
init='epsg:3857'), data_df_day['city.geo'][0][1], data_df_day['city.geo'
][0][0])
33 data_df_day = data_df_day.drop(columns=['city.geo'])
34 data_df_day = data_df_day.drop(columns=['time zone'])
35 final_realtime_table = gpd.GeoDataFrame(
36 data_df_day, geometry=gpd.points_from_xy(data_df_day['x'], data_df_day['
y']))
37
38 final_realtime_table['ID']=User
39 return final_realtime_table

```

Listing 6: Get Real time json data to dataframe

As in the previous function, the function gets json data, receiving as input the name of the city. Then we renamed and drop some columns to make the visualization clearer. This function is used to prepare the data in a dataframe to be sent to the Database and to make the map. For this reason you can see that:

- It is slightly different from the function that creates the table (see the next paragraph).
- It contains date time and time zone (that will be dropped in the next function). In fact they are useful if you a collection of data (what happens in the database), but not if you show only real time data (so only a row).
- This function contains `transform (Proj (init = ' epsg :4326 ') , Proj (init = ' epsg :3857 ')`, which is useful for the map in bokeh

Note that at the end the User ID is registered. It can be used to perform other functionalities in next release of the application.

From Real Time Json data to HTML This function, as we have already said before, is slightly different from the last one, because we considered less columns.

```

1 def get_realtime_data(city):
2     data = get_json_API(city)
3
4     #from JSON to Pandas DataFrame: creating the real time data table
5     data_df_day = pd.json_normalize(data['data'])
6     data_df_day["date"] = data_df_day["time.s"] + data_df_day["time.tz"]
7     #dropping the unnecessary columns:
8     data_df_day = data_df_day.drop(columns=['idx','attributions', 'dominentpol',
9     'city.url', 'city.location', 'time.v', 'time.iso',
10     'forecast.daily.o3', 'forecast.daily.pm10', '
forecast.daily.pm25', 'forecast.daily.uvi', 'debug.sync',
11     'time.s', 'time.tz'])
12     if city == 'skopje' or city == 'krakow':
13         data_df_day = data_df_day.drop(columns=['iaqi.dew.v', 'iaqi.wg.v'])
14     if city == 'belgrad':
15         data_df_day = data_df_day.drop(columns=['iaqi.wg.v'])
16
17     #renaming the columns we will be using for clarity:
18     data_df_day = data_df_day.rename(columns={'aqi': 'air quality', 'city.name':
19     'city', 'iaqi.co.v': 'carbon monoxyde',
20     'iaqi.h.v': 'relative humidity', '
iaqi.no2.v': 'nitrogen dioxide',
21     'iaqi.o3.v': 'ozone', 'iaqi.p.v': '
atmospheric pressure', 'iaqi.pm10.v': 'PM10',
22     'iaqi.pm25.v': 'PM2.5', 'iaqi.so2.v': 'sulphur dioxide', 'iaqi.t.v': 'temperature',
23     'iaqi.w.v': 'wind'})

```

```

23 #creating two columns for geographical coordinates instead of one for easier
    access:
24 data_df_day['lat'] = data_df_day['city.geo'][0][0]
25 data_df_day['lon'] = data_df_day['city.geo'][0][1]
26 data_df_day = data_df_day.drop(columns=['city.geo'])
27
28 final_realtime_table = gpd.GeoDataFrame(data_df_day, geometry=gpd.
    points_from_xy(data_df_day['lon'], data_df_day['lat']))
29
30 final_realtime_table_html = final_realtime_table.to_html(index=False)
31
32 return final_realtime_table_html

```

Listing 7: From Real Time Json data to HTML

As we did in function 5, we use the function `.to_html` to transform a dataframe in a table for the HTML page.

Connection to the database First of all we wrote a function to easily define the engine, which changes depending on the setting of the database. All the setting are collected in the `dbconfig.txt` file. It explains the reason why we wrote 8

```

1 from sqlalchemy import create_engine, null
2 def connStr():
3     myFile = open('dbConfig.txt')
4     [dbname,user,password] = [x.split(sep=" ")[1] for x in myFile.readline().
        split()]
5     return "postgresql://" + user + ":" + password + "@localhost:5432/" + dbname

```

Listing 8: Settings of Pgadmin

At this point we need first to send data to the database. We can do it thanks to the `.to_postgis()` function.

```

1 import geopandas as gpd
2 def sendDFtoDB(db):
3     engine = create_engine(connStr())
4     db.to_postgis('cities', engine, if_exists = 'replace', index=False) #I can
        put some queries here

```

Listing 9: Send data to the database

Starting from a Geodataframe, it sends data to the database³ updating or creating the table called 'cities'. After the creation of the database, the application needs to download data, add the new row and then update the database. For this reason we wrote the following function:

```

1 def update_data_on_DB(db):
2     engine = create_engine(connStr())
3     Data = gpd.GeoDataFrame.from_postgis('cities', engine, geom_col='geometry')
4     DataNew = Data.append(db)
5     return DataNew

```

Listing 10: Download and update data

The function `.from_postgis()` is used to download the table on the database and to register it as a geodataframe as a local variable. After that the function adds the new row and returns the new geodataframe.

Transform a json request in dataframe The following function is used to retrieve station coordinates and names in the more info section

³name of the database is in `dbConfig.txt` file

```

1 def translate_data(response):
2     raw_data = response.text
3     data = json.loads(raw_data)
4     df_stations = pd.json_normalize(data['data'])
5     gdf = gpd.GeoDataFrame(df_stations["station.name"],
6                             geometry=gpd.points_from_xy(df_stations.lat,
7                                                         df_stations.lon))
8     G = gdf.set_crs('epsg:4326')
9     G.rename(columns={'station.name': 'Station_Name'}, inplace=True, errors=
10               'raise')
11     coordinate_list = [(x,y) for x,y in zip(G.geometry.x , G.geometry.y)]
12     return coordinate_list, G.Station_Name, df_stations.aqi

```

Listing 11: From json to Dataframe

This function is similar to the one seen in 3, but this time the function receives as input a link of an API. Furthermore, this time everything is recorded in a geodataframe. After that we set CRS and renamed columns. The function returns the list of coordinates, the name and the AQI index of the stations in the area considered.

Project HTML The following function is used to create html code to be inserted into project page template, depending on the data provided and which type of component for the project page is desired.

```

1 def project_html(data, html_type = null):
2     if html_type == "table1":
3         return "<div class=\"box\">\n
4             <header>\n
5                 <h2>Realtime data</h2>\n
6             </header>\n
7             <div class=\"table-wrapper\">" + data + "</div>\n
8         </div>"
9
10    if html_type == "table2":
11        return "<div class=\"box\">\n
12            <h2>Forecast data</h2>\n
13            <div class=\"table-wrapper\">" + data + "</div>\n
14            <p><em>To sort data click column header</em></p>\n
15            <form method=\"post\" action=\"#\">\n
16                <div class=\"col-12\">\n
17                    <p><em>To filter, first select column, then\n
18                    filtering type and lastly, type filter value in \"Filter forecast...\" input\n
19                    .</em></p>\n
20                    <div class=\"row gtr-uniform\">\n
21                        <div class=\"col-6 col-12-xsmall\">\n
22                            <select name=\"fcolumn\" id=\"fcolumn\" onchange\n
23                            =\"check()\" required>\n
24                                <option value=\"\" disabled selected>Column name</option>\n
25                                <option value=\"day\">Day</option>\n
26                                <option value=\"avg_o3\">Average O3</option>\n
27                                <option value=\"max_o3\">Maximum O3</option>\n
28                                <option value=\"min_o3\">Minimum O3</option>\n
29                                <option value=\"avg_pm10\">Average pm10</option>\n
30                                <option value=\"max_pm10\">Maximum pm10</option>\n
31                                <option value=\"min_pm10\">Minimum pm10</option>\n
32                                <option value=\"avg_pm25\">Average pm25</option>\n
33                                <option value=\"max_pm25\">Maximum pm25</option>\n
34                                <option value=\"min_pm25\">Minimum pm25</option>\n
35                                <option value=\"avg_uvi\">Average UVI</option>\n
36                                <option value=\"max_uvi\">Maximum UVI</option>\n
37                                <option value=\"min_uvi\">Minimum UVI</option>\n
38                            </select>\n
39                        </div>\n

```

```

37         <div class=\"col-6 col-12-xsmall\">\
38         <p><b>Column</b></p>\
39         </div>\
40         </div>\
41         <div class=\"row gtr-uniform\">\
42         <div class=\"col-6 col-12-xsmall\">\
43         <select name=\"ftype\" id=\"ftype\" onchange=\"check()\"
required>\
44             <option value=\"\" disabled selected>Filter type</option>\
45             <option value=\"equal\">=</option>\
46             <option value=\"gt\">></option>\
47             <option value=\"lt\"><</option>\
48             <option value=\"egt\">>=</option>\
49             <option value=\"elt\"><=</option>\
50         </select>\
51         </div>\
52         <div class=\"col-6 col-12-xsmall\">\
53         <p><b>Type</b></p>\
54         </div>\
55         </div>\
56         <div class=\"row gtr-uniform\">\
57         <div class=\"col-6 col-12-xsmall\">\
58         <input type=\"text\" id=\"filterInput\" placeholder=\"Filter
forecast...\" disabled>\
59         </div>\
60         <div class=\"col-6 col-12-xsmall\">\
61         <p><b>Value</b></p>\
62         </div>\
63         </div>\
64     </div>\
65 </form>\
66 </div>\"
67 if html_type == \"tableStat\":
68     return \"    <div class=\"box\">\
69         <h2>Analysis</h2>\
70         <div class=\"table-wrapper\">\" + data + \"</div>\
71     </div>\"
72 if html_type == \"map\":
73     return \"    <div class=\"box\">\
74         <h2>Map</h2>\
75         <div>\" + data + \"</div>\
76     </div>\"

```

Listing 12: Project HTML

4.3 Data Tier

The data tier, also called database server or back-end is where the information processed by the application will be stored and managed. The web application will interact with PostgreSQL, which is a free and open-source database management system that will be used to perform query in Database and select Data we want to display.

PostgreSQL database (which we have called S4G) will be integrated with Python using psycopg2 module, which is a database adapter for the Python programming language.

It is important to underline that the usage of georeferenced data require to use the postgis extension in the database S4G.

5 User interface

In this section we analyze the user interface putting attention on common elements present in all the pages of the web application.

5.1 Navigation Bar

An important element of the user interface is the navigation bar, which is common for all the pages of the application. Here the user can click on each tab. For the "Project" tab to be opened the user has to be logged in first, and then the "Project" page can be opened.

The "More Info" tab is accessible from all the users. On this section every user can read information about the general idea of our project, data involved, the API used by our application.

The last button on the right is used to Log In in the application. If you are not registered, in the Log In page is be the possibility to register for every new user.



Figure 5: Navigation Bar

5.2 HomePage

Home page view	
Name identifier	DD 3
User:	All (registered and not)
Input	The following url http://127.0.0.1:5000/
Possible Actions	You can only scroll down to see a short overview of the project and, in the bottom, some ratings about the application, done by the users.

Table 3: Home Page

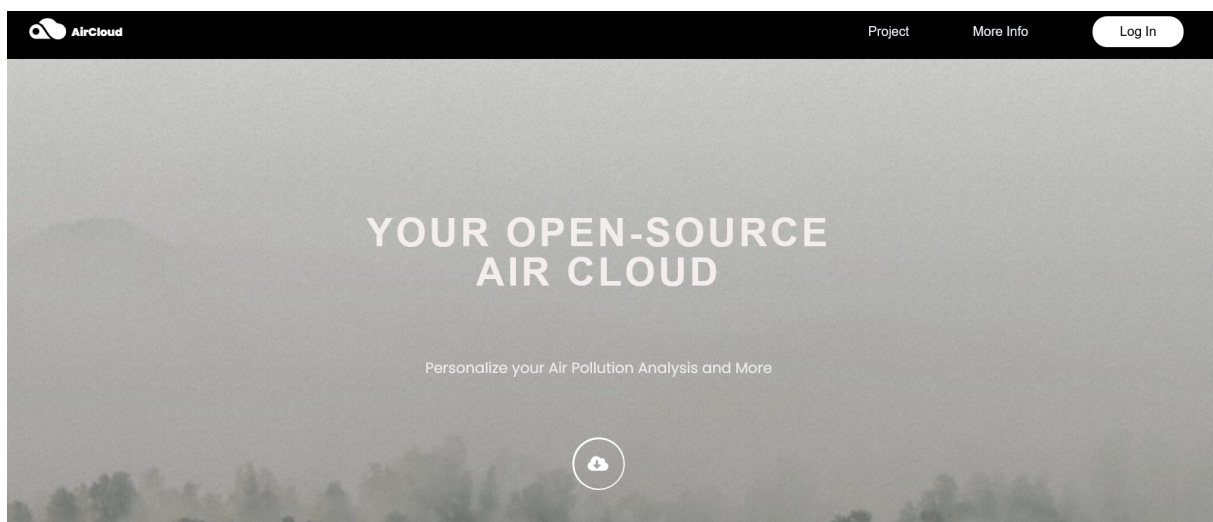


Figure 6: Home page

5.3 More Info

More Info	
Name identifier	DD 4
User:	Every user that wants to use the web application, even without logging in.
Input	To access the users, from the Home Page have to click on "More Info" button on the top bar.
Possible Actions	<p>When clicking the button the user views an interface similar to the one of Home Page but divided in two sections:</p> <ol style="list-style-type: none"> 1. General Idea: The user can read a brief article we wrote for explaining the reasons that led us to creating the "Air Cloud" web application. This section doesn't involve any dynamic feature, but it is rendered only as part of the static HTML part of the web. 2. Data: The user here can find all the information about the data treated in the projects and that can be found in details in the RASD document. In particular there is a clear explanation of all the chemical pollutants

Table 4: More Info Section

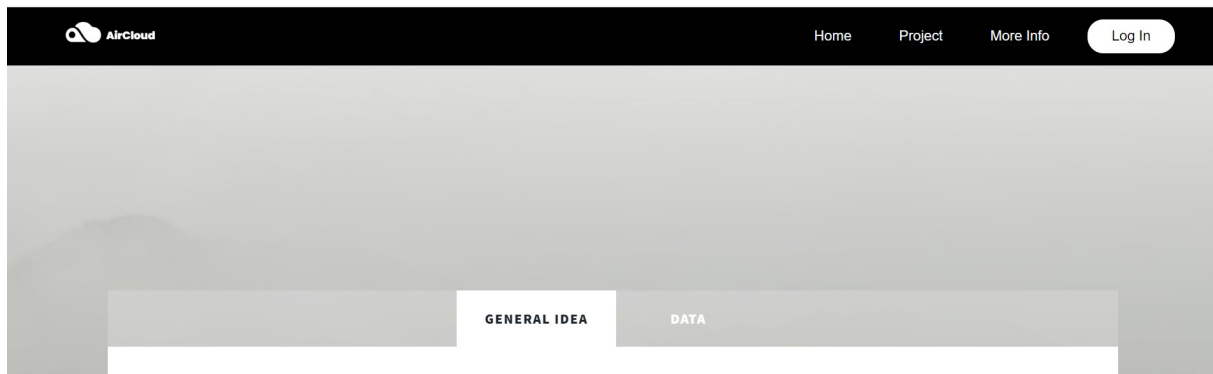


Figure 7: More Info page. As you can see it is slit in "General Idea" and "Data" section

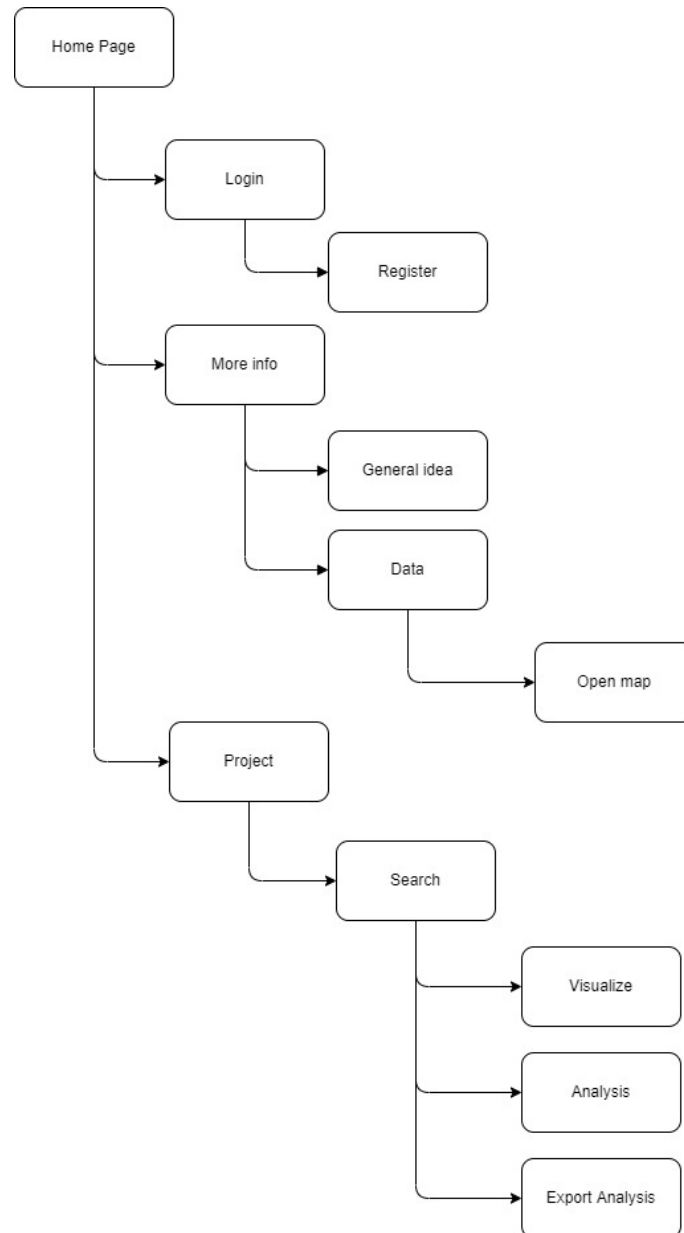


Figure 8: Web page structure

6 User cases application

In this section we explain the software functionalities, interactions between the components (all the possible exceptions will be considered in the testing document). The purpose is to list the actions taken by the software and users, useful for explaining each internal process of the application.

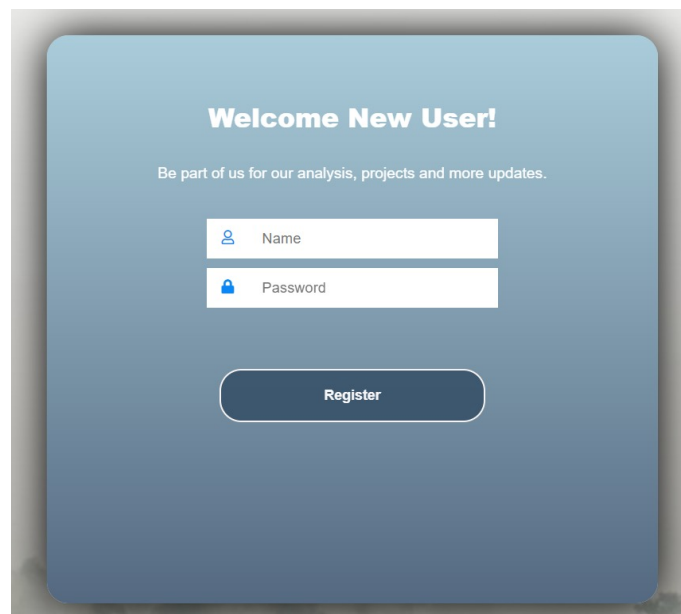
We describe what is going on from both server-side and client-side on each use case, by specifying the different actions or events that take place in these situations.

6.1 Register

The register environment can be found in figure 9

Register	
Name identifier	DD 5
User:	Only people that are not registered yet
Input	To access the users, from the Home page, has to go in the log in page and then click on "Register"
Possible Actions	In this page the user has to fill both the name and password. When the user clicks on Register, he/she is redirected automatically to the log in page .

Table 5: Register Page



The screenshot shows a registration form titled "Welcome New User!". Below the title is a subtitle: "Be part of us for our analysis, projects and more updates." The form contains two input fields: "Name" with a user icon and "Password" with a lock icon. Below these fields is a "Register" button.

Figure 9: Register

6.2 Log In

Log In	
Name identifier	DD 6
User:	Only people who are already registered
Input	To access the users, from the Home page, has click on "Log in".
Possible Actions	In this page the user has to fill both the user name and password. When the user clicks on Log In, he/she is redirected automatically to the Home page if the password is right, otherwise the user remains in the login page.

Table 6: Log In Page

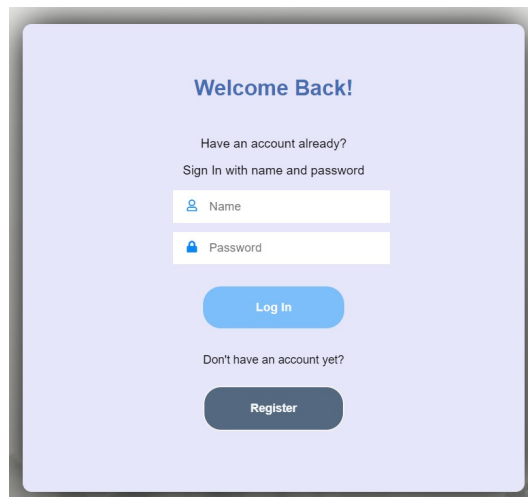


Figure 10: Login

6.3 Log Out

Log Out	
Name identifier	DD 7
User:	Only users that are registered
Input	In the top bar of the home page, users after having completed the Log In, will see the Log Out button instead of the Log In.
Possible Actions	When clicking the button the user logs out from his account and goes again in the home page but this time the button is transformed in Log In button.

Table 7: Log Out Button

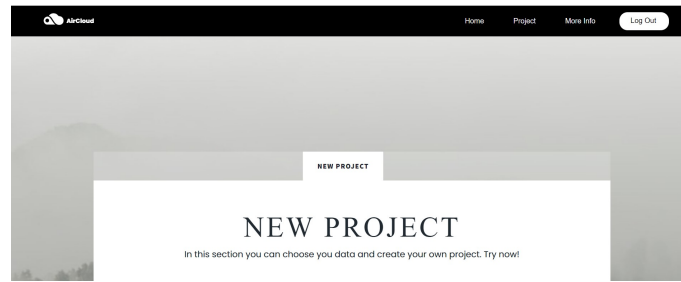


Figure 11: New Project

6.4 Query Data

Query Data	
Name identifier	DD 8
User:	A registered and logged in user after accessing the “project page”.
Input	<ol style="list-style-type: none"> 1. City: chooses one of the 5 cities available (Skopje, Belgrad, Kraków, Paris and London) from a drop down list. 2. Data Type: the user chooses what kind of data she/he wants to query: Real Time air pollution data, Forecast of air pollution for the next days or Both.
Possible Actions	<p>After inputting the wanted parameters the user clicks on the “search” button and retrieves the data, which are presented in tables. Additionally with the queried data tables 2 new sections on the web-page will appear:</p> <ol style="list-style-type: none"> 1. Map: An interactive Bokeh map of the city the user has currently queried. 2. Basic Statistics: A table with the basic statistics of the queried data. <p>The user can then click on any visible button to continue working with the retrieved data.</p>

Table 8: Query Data

Select city and type of data

City

City
Skopje
Kraków
Belgrad
Paris
London

Figure 12: Choosing the City

Select city and type of data

Skopje
Data Type
Data Type
Real Time
Forecast
Both

SEARCH

Figure 13: Choosing the Data Type

Select city and type of data

City
Data Type

SEARCH

REALTIME DATA

VISUALIZE

ANALYZE

AIR QUALITY	CITY	CARBON MONOXIDE	RELATIVE HUMIDITY	OZONE	ATMOSPHERIC PRESSURE	PM10	PM2.5	SULPHUR DIOXIDE	TEMPERAT
53	Centar, Skopje, Macedonia (Центар)	0.6	59	2.6	1020	53	29	0.6	26.5

Figure 14: The result of a sample query

6.5 Geographic visualization

Geographic visualization	
Name identifier	DD 9
User:	A registered and logged in user after querying the data.
Input	The user clicks on "visualize" and is redirected to the part of the web-page where the interactive map of the currently queried place is embedded.
Possible Actions	The user can hover on the red dot in the map to visualize a pop-up window with the basic air quality information about the place. The user can also proceed with analysis by choosing one of the buttons at the top of the data.

Table 9: Geographic visualization

6.6 Data manipulation

Data manipulation	
Name identifier	DD 10
User:	A registered and logged in user after querying the data.
Input	<p>The user has the options to choose to:</p> <ol style="list-style-type: none"> 1. Sort: by clicking on the column by which the user wants to sort, the data will reorganize accordingly to the arrow next to the column name (ascending or descending sort). (Only for Forecast data) 2. Filter: the user after clicking on the “filter” button can enter the phrases she/he looks for to extract that data from the table. 3. Analyze: the user clicks on the “analyze” button at the top of the data and she/he is redirected down to the simple statistic analysis table that is retrieved with the data, when queried.
Possible Actions	After simple data manipulation the user can export the analysis or query new data.

Table 10: Data manipulation

6.7 Analyze data and Export Analysis

Analyze data and Export Analysis	
Name identifier	DD 11
User:	A registered and logged in user with queried data.
Input	The user clicks on the "Export Analysis" button underneath the simple statistics table returned after querying the data.
Possible Actions	After clicking on the "Export Analysis" button the user is redirected to a new page with more advanced analysis performed on the data. Once the page is created the user can print the html page to export it as pdf

Table 11: Analyze data and Export Analysis

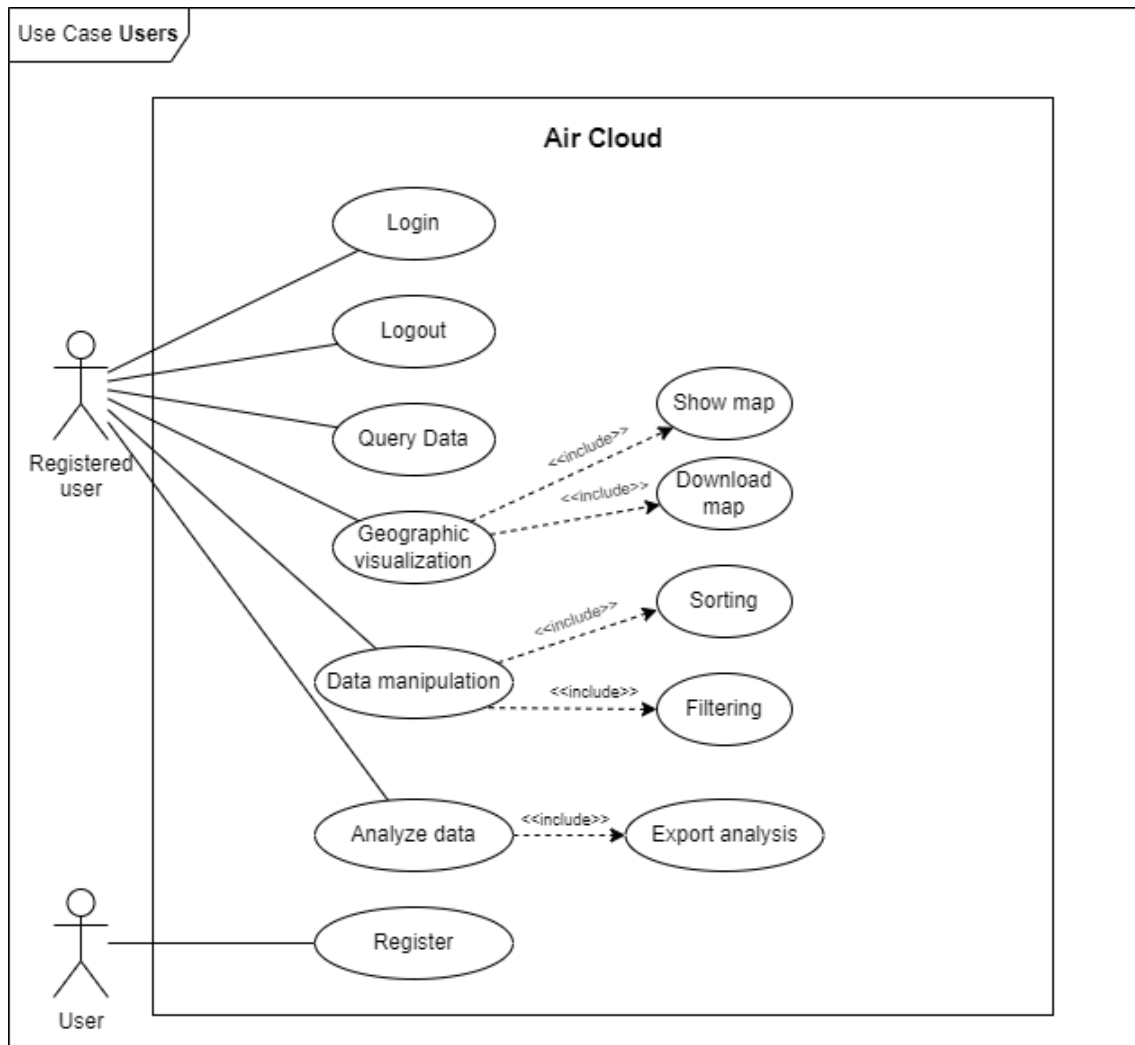


Figure 15: UML of all User cases structure

7 Organization

Name	Hours spent (h)	Details
Bigai	35	Data section.
Calcaterra	35	Project Database section and some use cases.
Leonardi	35	Some use cases.
Rzewuski	35	UML of the application.
Zallem	35	All the introduction, software structure and some use cases.

Table 12: Effort spent with details