# WEATHER AND POLLUTION DATA VISUALIZER FOR DUTCH CITIES

## API Integration, SQL Translation, and GUI Implementation

**CHRISTIAN GABRIEL CENTENO**

*LINKEDIN*
*GITHUB*

# INDEX

# 1. INTRODUCTION

The objective of the project is to be able to extract the **API** information in **JSON** format using API calls using **Python pipelines,** store the information in an **SQLite** database and manipulate the information through a **GUI application**.

I used a free Openweathermap account to get all the weather and pollution information.

OpenWeather

The project is divided into four main areas:

- *API Gateway*: Scripts were created using Python that are responsible for making **requests,** transforming data into a unified format and storing them in a specific data structure for future storage.
- *SQL Translator*: Created an adapter and translator between Python and **SQLite** to abstract the storage and extraction of useful information.
- *GUI & Auxiliary Modules*: Multiple modules have also been created for the general operation of the application, such as **visual object builders, map creators** and handlers of **GeoJSON** and **demographic files** from the Dutch Government.

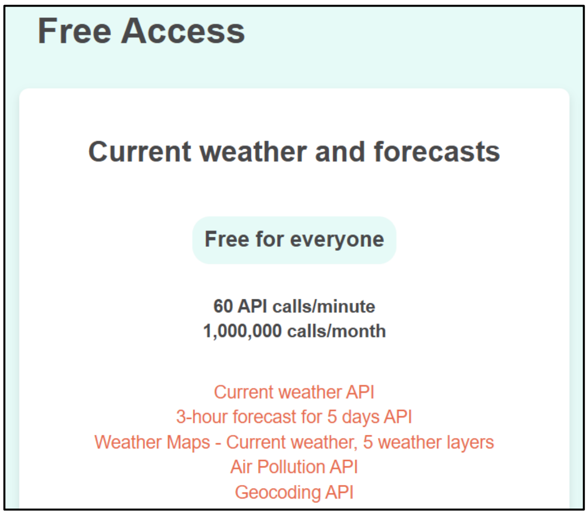
So, let's see!

# 2. API GATEWAY

## 2.1 Getting to Know the API platform

First, we determined which application we wanted to use as the basis for the project. In this case, we chose ***OpenWeatherMap***.

This decision was made because, with a free account, you can make quite a few interesting calls.



Free Access

Current weather and forecasts

Free for everyone

60 API calls/minute
1,000,000 calls/month

Current weather API
3-hour forecast for 5 days API
Weather Maps - Current weather, 5 weather layers
Air Pollution API
Geocoding API

Since this is a small project for training purposes, the free account is the best option. This limits the project to a limit of 60 calls per minute, which means the final application is subject to rate limits, requiring it to wait minutes per call stack when trying to retrieve the weather forecast for Dutch cities (making the loading time longer).

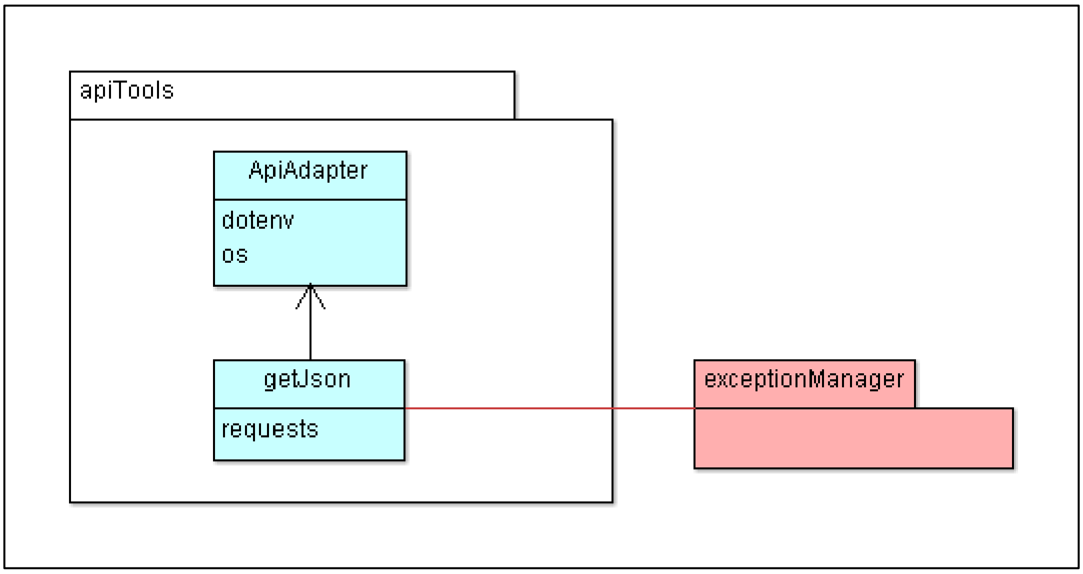
But it's still worth it.

## 2.2 The architecture of the Python API package

For this section, the activity has been concentrated in a package named "**apiTools**" with 2 unitary scripts:

- *getJson*: Which executes the **requests** and returns formatted data.
- *ApiAdapter*: Which orchestrates **getJson,** builds the URLs, and **allow the API to be abstracted from the rest of the packages.**

There is also a relationship with the ***exceptionManager*** package where exception types are defined for managing in the main module (***UI.py***).



# 3. SQL TRANSLATOR

## 3.1 What Database will Be Used?

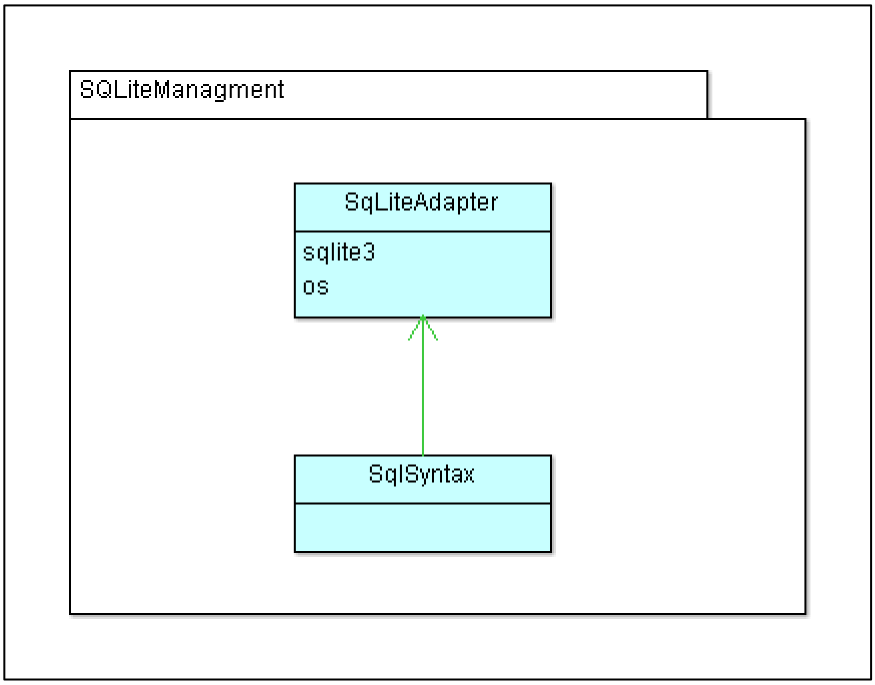
Once we knew the density of the information I would be using, the scope of the project established, and the minimum technical requirements were met. I decided to use **SQLite** because it's the smallest and easiest database to manage. It's also very portable, and no prior installation is required.

## 3.2 The architecture of the SQL gateway package

The activity has been concentrated in a package named "**SQLiteManagment**"

This package consists of 2 scripts:

- *SqlSyntax*: By using different Python and SQL syntax, this script **has focused on performing the most basic SQL constructs as a translator.**
- *SqLiteAdapter*: It's responsible for working as a gateway to the database, **managing connections and using the translator**. This way, other modules only have to focus on their intended purpose.



# 4. GUI & AUXILIARY MODULES

Finally, there's the GUI section, which centralizes the rest of the packages and provides visual value.

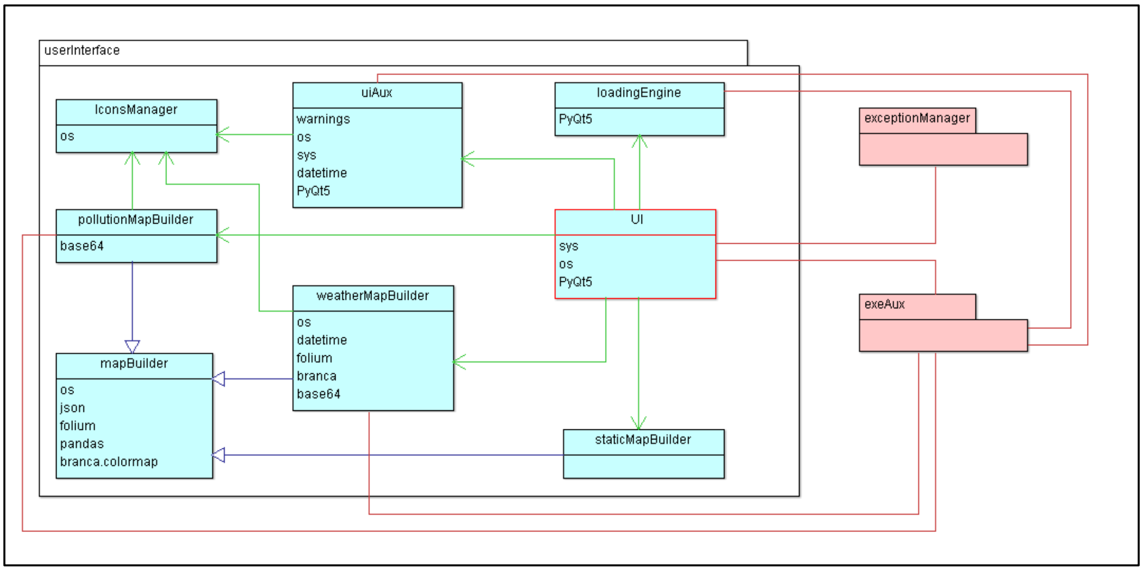
## 4.1 GUI

### 4.1.1 GUI Package Architecture

The "**userInterface**" package is the most complex module, **having the most connections between components**. This is mainly due to the fact that the visual objects have been distributed among different auxiliary objects created by constructors. It also represents the core module of the program, so it also has connections to other modules for operability.

It's composed of:

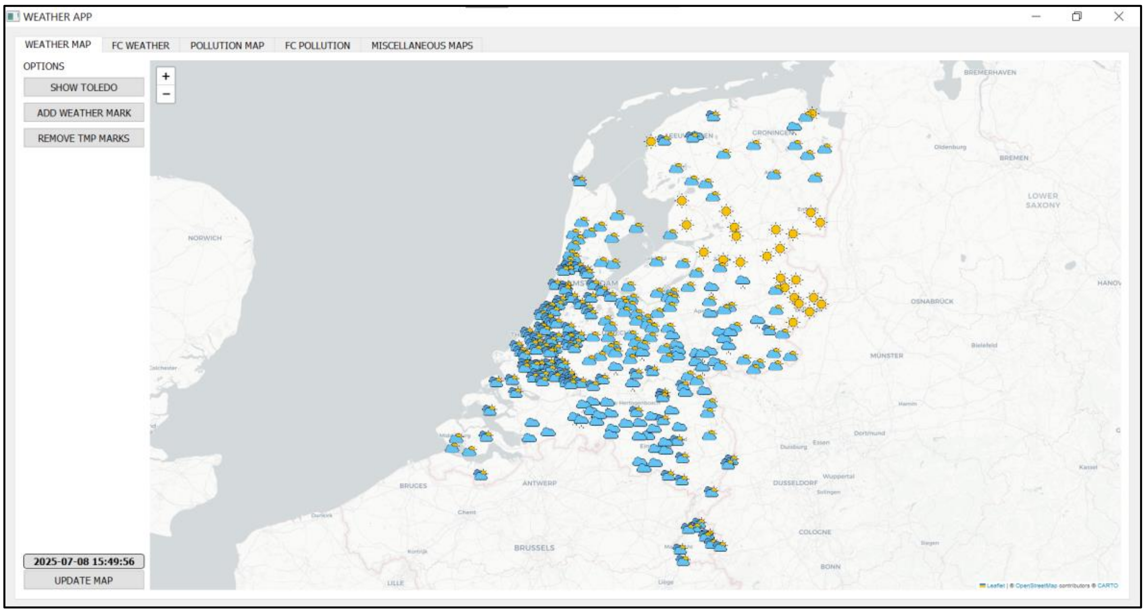
- *UI*: Is the main module, where the rest of the principal modules are called to orchestrate the application and with which the user interacts. It also uses the **exceptionManager** package, as this is where potential errors in the core scripts are debugged.
- *uiAux*: It's responsible for creating all visual objects.
- *mapBuilder*: It's the main class of map builder, from which different types of maps have been generated that will then be visually hosted in the application.
- *weatherMapBuilder*: Child map builder, focused on the weather map.
- *pollutionMapBuilder*: Child map builder, focused on the pollution map.
- *staticMapBuilder*: Child map builder, focused on the "default" map (population and dem).
- *IconsManager*: Manages the necessary icons.
- *loadingEngine*: responsible for bulk loading information about the most relevant locations in the Netherlands.



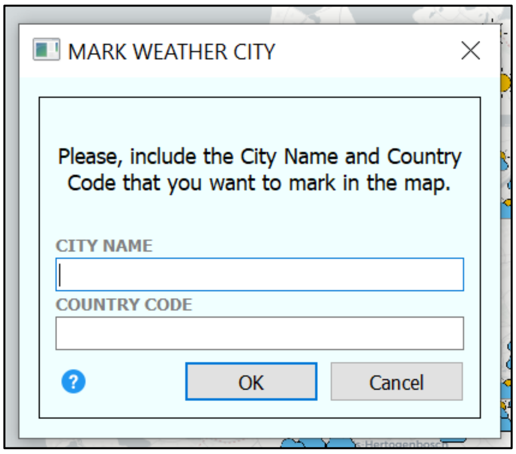
### 4.1.2 Sections

Finally, let's see all the tabs that the GUI App presents.

The first tab is **WEATHER MAP**. Where you can see the current weather in the Netherlands on a map.



You can also include or exclude markers manually.

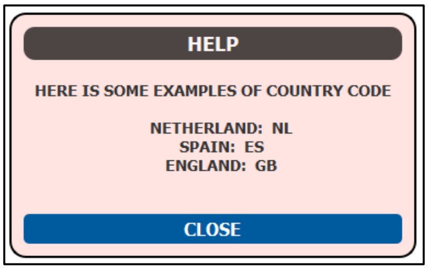


MARK WEATHER CITY

Please, include the City Name and Country Code that you want to mark in the map.

CITY NAME

COUNTRY CODE

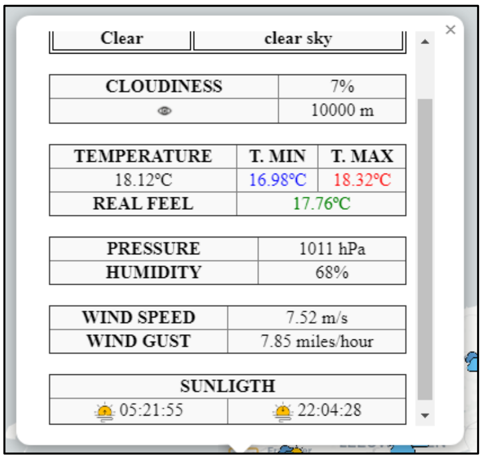


HELP

HERE IS SOME EXAMPLES OF COUNTRY CODE

NETHERLAND: NL
SPAIN: ES
ENGLAND: GB

When you click on one of the points, you can see more information about it.



| Clear | clear sky |
|---|---|

| CLOUDINESS | 7% |
|---|---|
| 👁 | 10000 m |

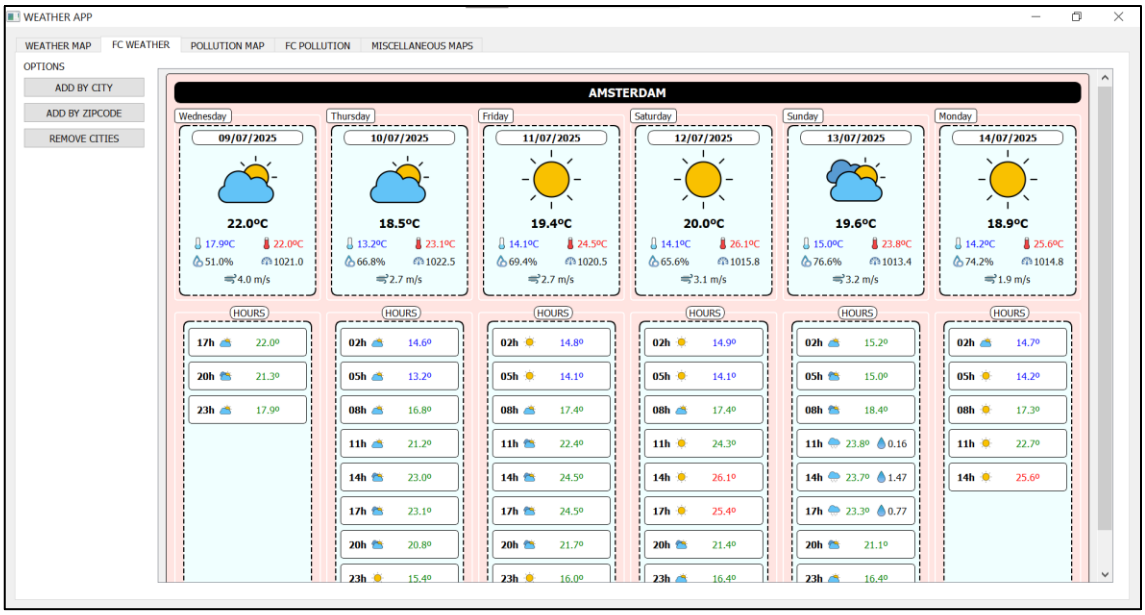
| TEMPERATURE | T. MIN | T. MAX |
|---|---|---|
| 18.12°C | 16.98°C | 18.32°C |
| REAL FEEL | 17.76°C | |

| PRESSURE | 1011 hPa |
|---|---|
| HUMIDITY | 68% |

| WIND SPEED | 7.52 m/s |
|---|---|
| WIND GUST | 7.85 miles/hour |

| SUNLIGTH | |
|---|---|
| 05:21:55 | 22:04:28 |

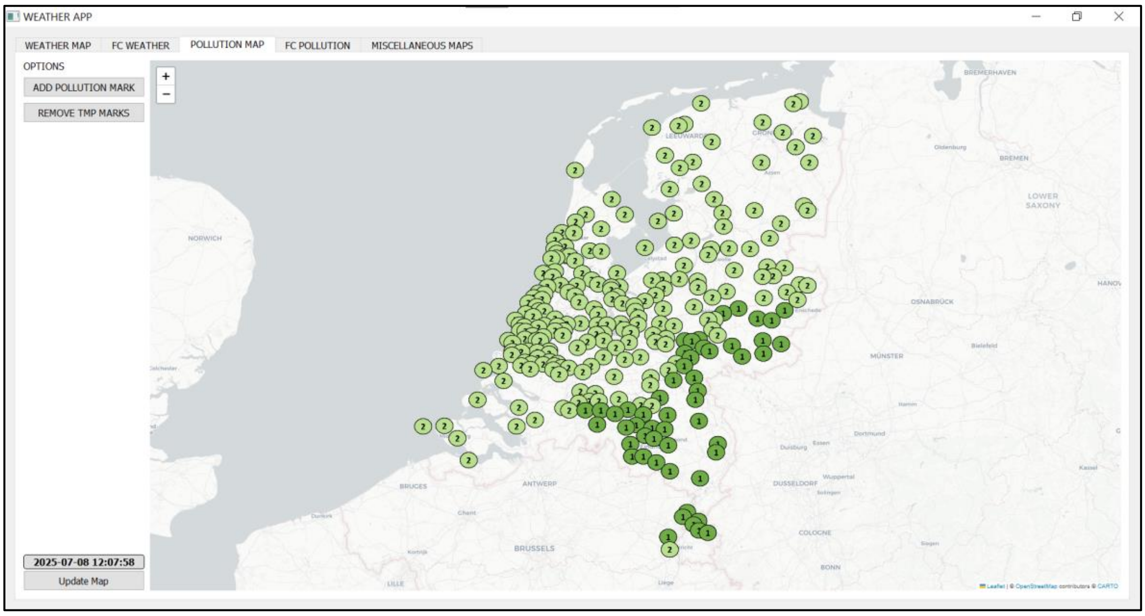
The second tab is **FC WEATHER** where you can see the weather forecast for any location in the world, you can add a new location by name or zip code.

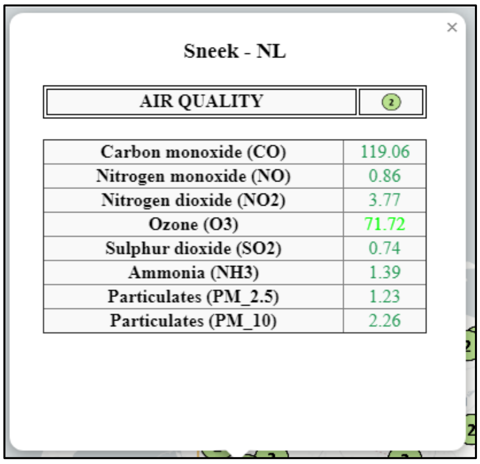


AMSTERDAM

| Wednesday | Thursday | Friday | Saturday | Sunday | Monday |
|---|---|---|---|---|---|
| 09/07/2025 | 10/07/2025 | 11/07/2025 | 12/07/2025 | 13/07/2025 | 14/07/2025 |
| 22.0ºC | 18.5ºC | 19.4ºC | 20.0ºC | 19.6ºC | 18.9ºC |
| 17.9ºC | 13.2ºC | 14.1ºC | 14.1ºC | 15.0ºC | 14.2ºC |
| 22.0ºC | 23.1ºC | 24.5ºC | 26.1ºC | 23.8ºC | 25.6ºC |
| 51.0% | 66.8% | 69.4% | 65.6% | 76.6% | 74.2% |
| 1021.0 | 1022.5 | 1020.5 | 1015.8 | 1013.4 | 1014.8 |
| 4.0 m/s | 2.7 m/s | 2.7 m/s | 3.1 m/s | 3.2 m/s | 1.9 m/s |
| HOURS | HOURS | HOURS | HOURS | HOURS | HOURS |
| 17h 22.0º<br>20h 21.3º<br>23h 17.9º | 02h 14.6º<br>05h 13.2º<br>08h 16.8º<br>11h 21.2º<br>14h 23.0º<br>17h 23.1º<br>20h 20.8º<br>23h 15.4º | 02h 14.8º<br>05h 14.1º<br>08h 17.4º<br>11h 22.4º<br>14h 24.5º<br>17h 24.5º<br>20h 21.7º<br>23h 16.0º | 02h 14.9º<br>05h 14.1º<br>08h 17.4º<br>11h 24.3º<br>14h 26.1º<br>17h 25.4º<br>20h 21.4º<br>23h 16.4º | 02h 15.2º<br>05h 15.0º<br>08h 18.4º<br>11h 23.8º 0.16<br>14h 23.7º 1.47<br>17h 23.3º 0.77<br>20h 21.1º<br>23h 16.4º | 02h 14.7º<br>05h 14.2º<br>08h 17.3º<br>11h 22.7º<br>14h 25.6º |

The next tab is **POLLUTION MAP** that is exactly like the first but pollution information based.



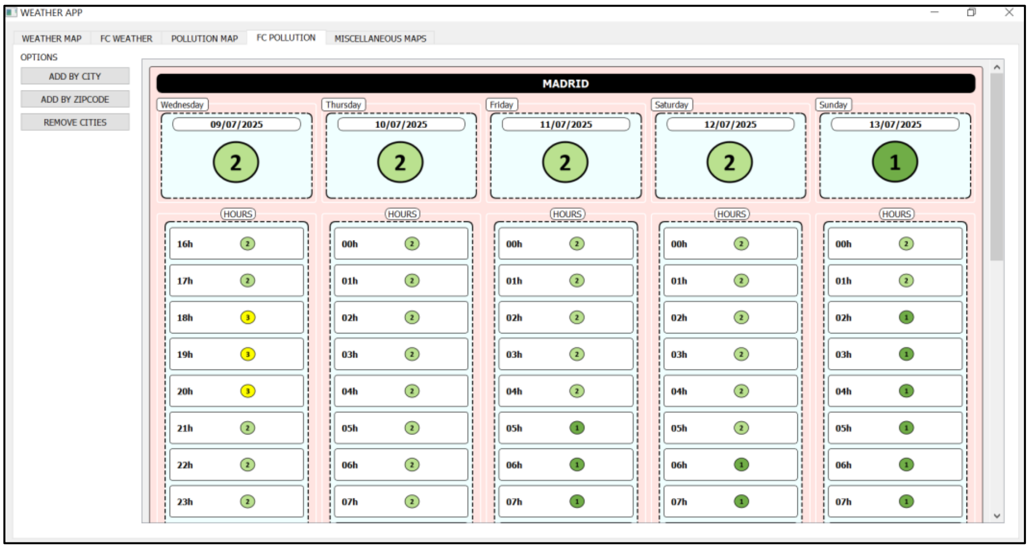


Sneek - NL

| AIR QUALITY | 2 |
|---|---|

| | |
|---|---|
| Carbon monoxide (CO) | 119.06 |
| Nitrogen monoxide (NO) | 0.86 |
| Nitrogen dioxide (NO2) | 3.77 |
| Ozone (O3) | 71.72 |
| Sulphur dioxide (SO2) | 0.74 |
| Ammonia (NH3) | 1.39 |
| Particulates (PM_2.5) | 1.23 |
| Particulates (PM_10) | 2.26 |

The same structure applies to the **FC POLLUTION** tab as in the previous ones.

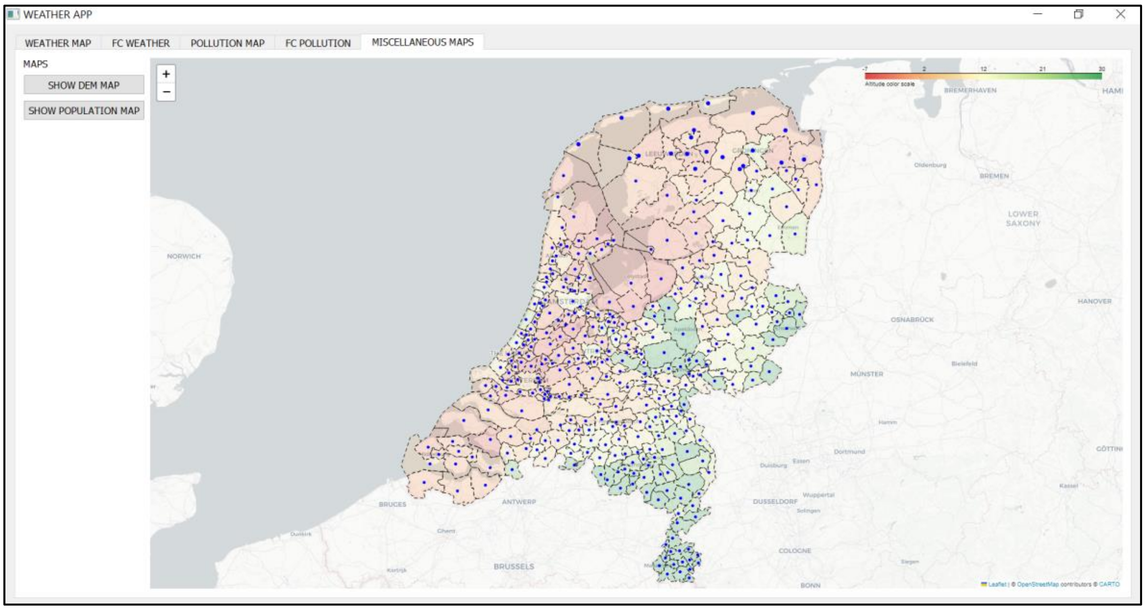


MADRID

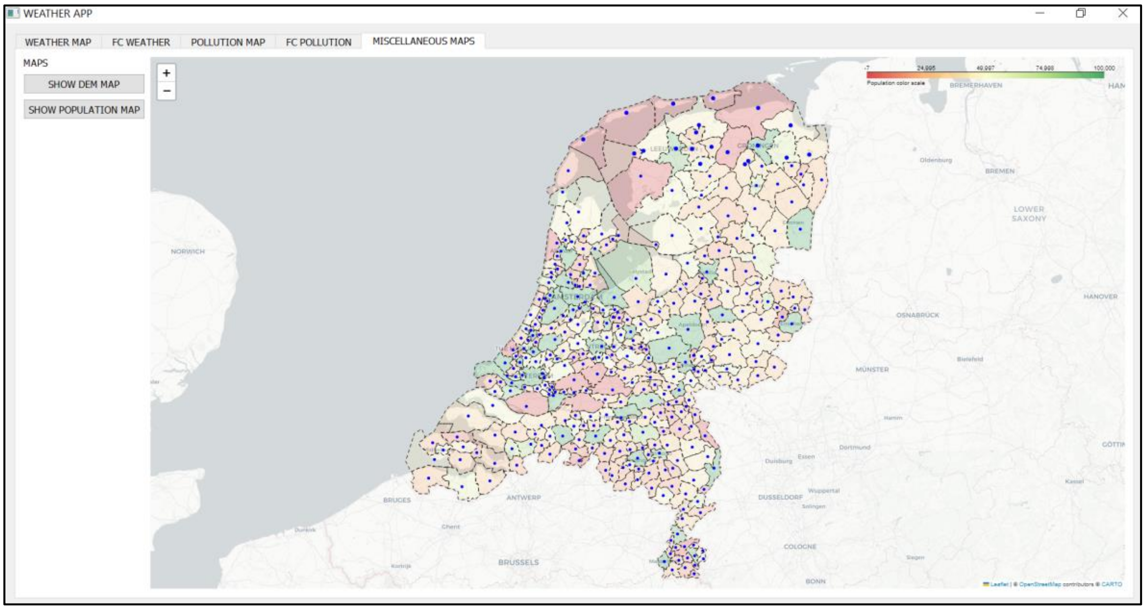
| Wednesday | Thursday | Friday | Saturday | Sunday |
|---|---|---|---|---|
| 09/07/2025 | 10/07/2025 | 11/07/2025 | 12/07/2025 | 13/07/2025 |
| 2 | 2 | 2 | 2 | 1 |
| HOURS | HOURS | HOURS | HOURS | HOURS |
| 16h 2 | 00h 2 | 00h 2 | 00h 2 | 00h 2 |
| 17h 2 | 01h 2 | 01h 2 | 01h 2 | 01h 2 |
| 18h 3 | 02h 2 | 02h 2 | 02h 2 | 02h 1 |
| 19h 3 | 03h 2 | 03h 2 | 03h 2 | 03h 1 |
| 20h 3 | 04h 2 | 04h 2 | 04h 2 | 04h 1 |
| 21h 2 | 05h 2 | 05h 1 | 05h 2 | 05h 1 |
| 22h 2 | 06h 2 | 06h 1 | 06h 1 | 06h 1 |
| 23h 2 | 07h 2 | 07h 1 | 07h 1 | 07h 1 |

Finally, there is the **MISCELLANEOUS MAPS** tab, which is where different thematic maps are displayed (more may be added in the future).

One of them classifies the Gemeente of the Netherlands according to altitude relative to sea level (**DEM**).



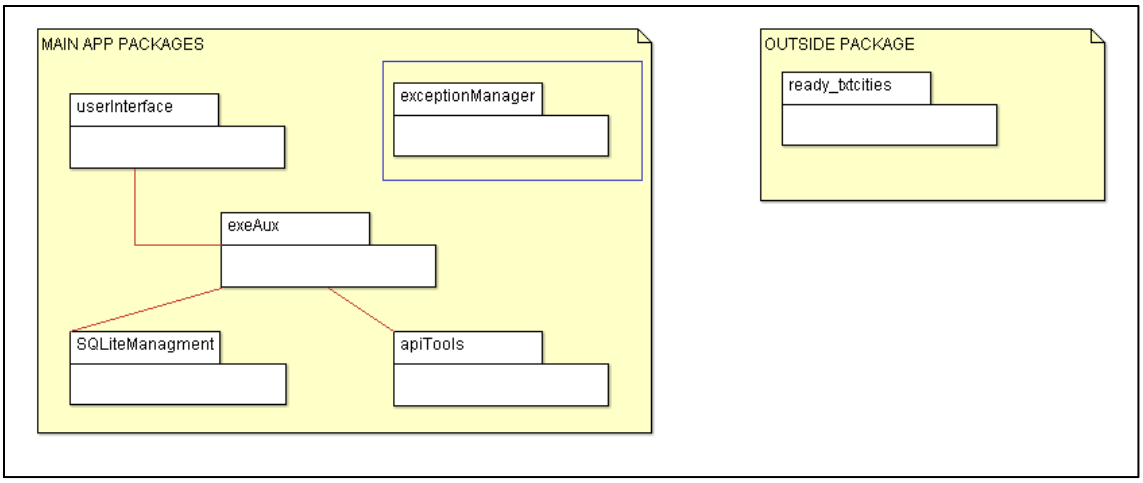
And another classifies the Gemeente of the Netherlands according to their **population**.



## 4.2 Auxiliary Modules

In order to carry out the project, it was necessary to create various auxiliary resources.

As for the packages used, this is the following diagram:

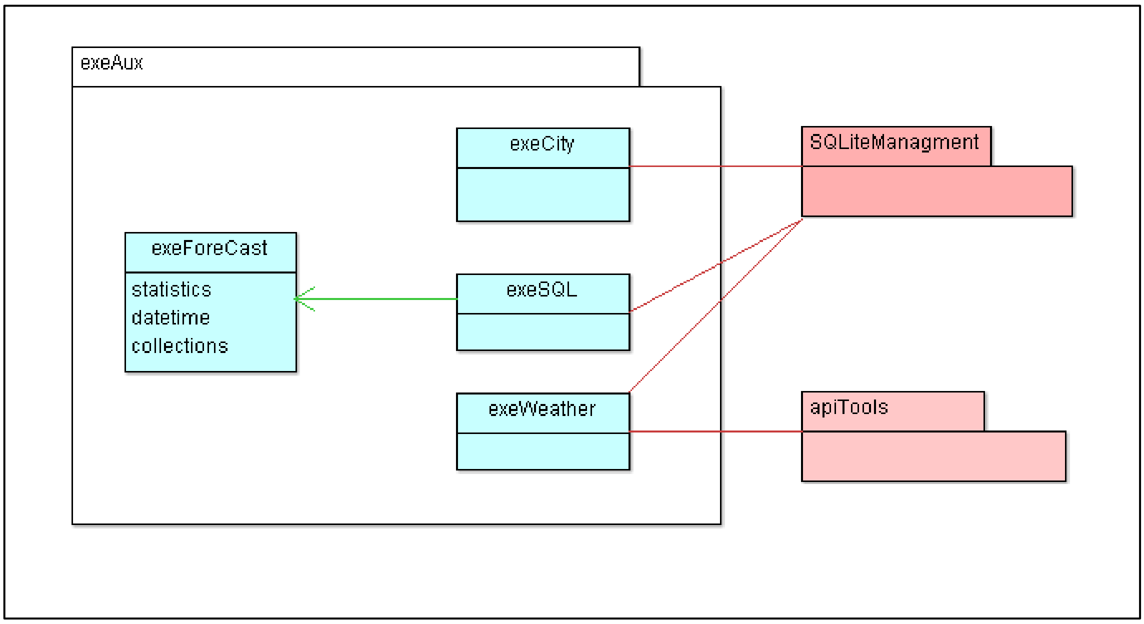


### 4.2.1 exeAux Package

This is the last package within the main program. It contains most of the logic engines used by the UI, designed to lower the level of abstraction to the gateways.

This is the architecture of the package:

- *exeCity*: It's the engine that is responsible for manipulating all the information stored in the database in relation to locations that have been preloaded.
- *exeSQL*: It's the engine that is responsible for storing information in the database.
- *exeWeather*: It's the engine that is responsible for extracting all the pollution and weather information from the database.
- *exeForeCast*: It's the engine that is responsible for extracting, processing, and calculating all the information used to display the forecasts.



### 4.2.2 Preloading Cities Information for the Netherlands

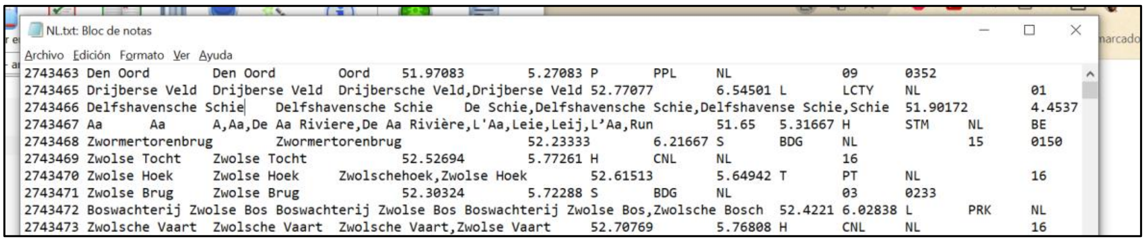
To do this, a script called **ready_txtcities** was created external to the main program.

Exploring the website https://www.geonames.org/

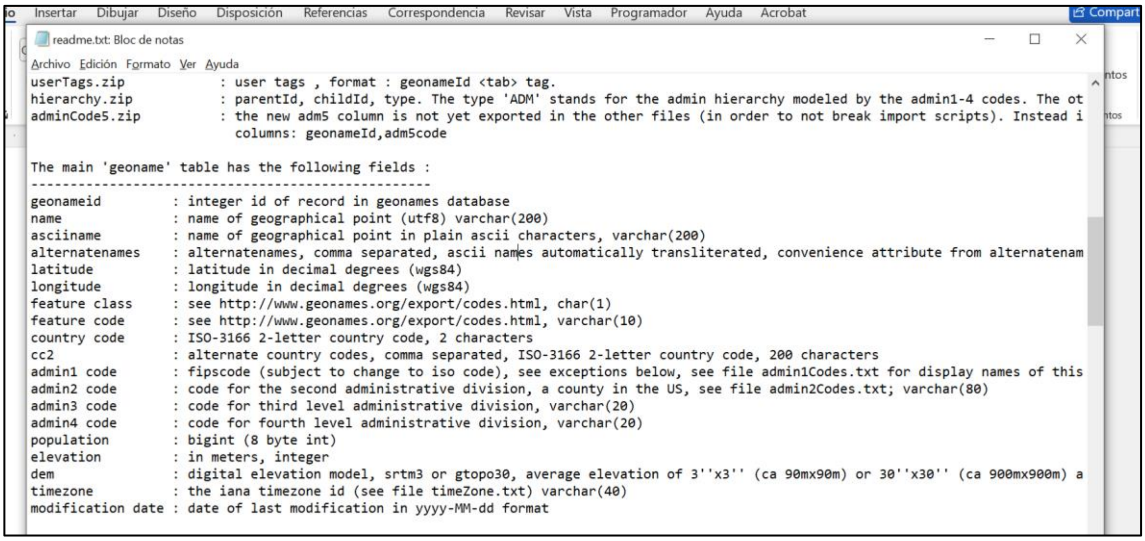
The GeoNames geographical database covers all countries and contains over eleven million placenames that are available for download free of charge.

enter a location, ex: "Paris", "Mount Everest","New York","47 9" (lat lng)

I downloaded all the information on the infrastructure data for the Netherlands in plain text.



```
2743463 Den Oord        Den Oord        Oord    51.97083        5.27083 P       PPL     NL              09      0352
2743465 Drijberse Veld  Drijberse Veld  Drijbersche Veld,Drijberse Veld 52.77077        6.54501 L       LCTY    NL              01
2743466 Delfshavensche Schie    Delfshavensche Schie    De Schie,Delfshavensche Schie,Delfshavense Schie,Schie  51.90172        4.4537
2743467 Aa      Aa      A,Aa,De Aa Riviere,De Aa Rivière,L'Aa,Leie,Leij,L’Aa,Run        51.65   5.31667 H       STM     NL      BE
2743468 Zwormertorenbrug        Zwormertorenbrug        52.23333        6.21667 S       BDG     NL              15      0150
2743469 Zwolse Tocht    Zwolse Tocht            52.52694        5.77261 H       CNL     NL              16
2743470 Zwolse Hoek     Zwolse Hoek     Zwolschehoek,Zwolse Hoek        52.61513        5.64942 T       PT      NL              16
2743471 Zwolse Brug     Zwolse Brug             52.30324        5.72288 S       BDG     NL              03      0233
2743472 Boswachterij Zwolse Bos Boswachterij Zwolse Bos Boswachterij Zwolse Bos,Zwolsche Bosch  52.4221 6.02838 L       PRK     NL
2743473 Zwolsche Vaart  Zwolsche Vaart  Zwolsche Vaart,Zwolse Vaart     52.70769        5.76808 H       CNL     NL              16
```


Insertar Dibujar Diseño Disposición Referencias Correspondencia Revisar Vista Programador Ayuda Acrobat
Compart
readme.txt: Bloc de notas
Archivo Edición Formato Ver Ayuda

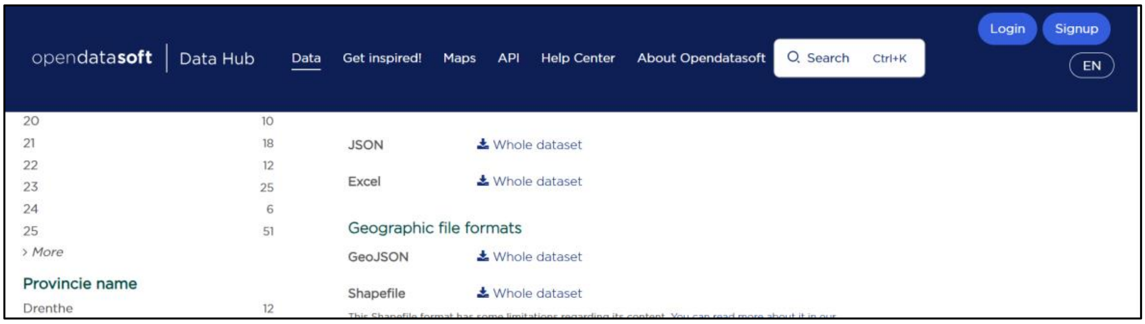
```
userTags.zip            : user tags , format : geonameId <tab> tag.
hierarchy.zip           : parentId, childId, type. The type 'ADM' stands for the admin hierarchy modeled by the admin1-4 codes. The ot
adminCode5.zip          : the new adm5 column is not yet exported in the other files (in order to not break import scripts). Instead i
                          columns: geonameId,adm5code

The main 'geoname' table has the following fields :
---------------------------------------------------
geonameid         : integer id of record in geonames database
name              : name of geographical point (utf8) varchar(200)
asciiname         : name of geographical point in plain ascii characters, varchar(200)
alternatenames    : alternatenames, comma separated, ascii names automatically transliterated, convenience attribute from alternatenam
latitude          : latitude in decimal degrees (wgs84)
longitude         : longitude in decimal degrees (wgs84)
feature class     : see http://www.geonames.org/export/codes.html, char(1)
feature code      : see http://www.geonames.org/export/codes.html, varchar(10)
country code      : ISO-3166 2-letter country code, 2 characters
cc2               : alternate country codes, comma separated, ISO-3166 2-letter country code, 200 characters
admin1 code       : fipscode (subject to change to iso code), see exceptions below, see file admin1Codes.txt for display names of this
admin2 code       : code for the second administrative division, a county in the US, see file admin2Codes.txt; varchar(80)
admin3 code       : code for third level administrative division, varchar(20)
admin4 code       : code for fourth level administrative division, varchar(20)
population        : bigint (8 byte int)
elevation         : in meters, integer
dem               : digital elevation model, srtm3 or gtopo30, average elevation of 3''x3'' (ca 90mx90m) or 30''x30'' (ca 900mx900m) a
timezone          : the iana timezone id (see file timeZone.txt) varchar(40)
modification date : date of last modification in yyyy-MM-dd format
```

This script cleaned the data and stored it in two different CSV files. One filtered the most populated locations for the weather/pollution Netherlands' map, and the other stored all the locations (which will be discussed in the next section).

### 4.2.3 Creating GeoJSON

For render maps by Gemeente, you must first obtain a *GeoJSON* document for the Netherlands. This document is a type of JSON file that contains geographic information that can be used to render maps. To do this, visit the page https://data.opendatasoft.com/explore/dataset/georef-netherlands-gemeente%40public/export/?disjunctive.prov_code&disjunctive.prov_name&disjunctive.gem_code&disjunctive.gem_name



| | |
|---|---|
| 20 | 10 |
| 21 | 18 |
| 22 | 12 |
| 23 | 25 |
| 24 | 6 |
| 25 | 51 |

› More

Provincie name

| | |
|---|---|
| Drenthe | 12 |

JSON ⬇ Whole dataset

Excel ⬇ Whole dataset

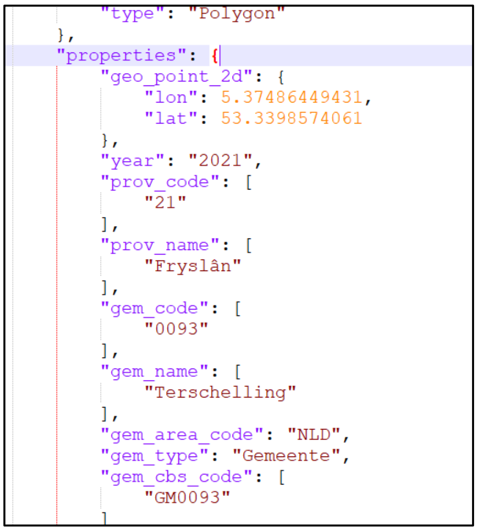
Geographic file formats

GeoJSON ⬇ Whole dataset

Shapefile ⬇ Whole dataset

This Shapefile format has some limitations regarding its content. You can read more about it in our

Where the document was downloaded.



```
            "type": "Polygon"
        },
        "properties": {
            "geo_point_2d": {
                "lon": 5.37486449431,
                "lat": 53.3398574061
            },
            "year": "2021",
            "prov_code": [
                "21"
            ],
            "prov_name": [
                "Fryslân"
            ],
            "gem_code": [
                "0093"
            ],
            "gem_name": [
                "Terschelling"
            ],
            "gem_area_code": "NLD",
            "gem_type": "Gemeente",
            "gem_cbs_code": [
                "GM0093"
            ]
```

Finally, by merging this information with the second CSV explained in the previous section, it has been possible to create regional maps according to their altitude or population (corresponding to the *miscellaneous tab*).

# 5. FINAL CONCLUSIONS

This project allowed us to review and practice creating pipelines that interact with databases and API calls. I also had the opportunity to learn more about the visual elements that Python offers with some of its most popular libraries. Finally, I would like to point out that one of the challenges was establishing the scope of the project, as it was very easy to want to cover more than the project's main purpose.

I hope this project was of interest to the reader.