POLITECNICO
MILANO 1863

Politecnico di Milano

Software Engineering for Geoinformatics

REPORT

# Software Release Document for AQI-GDP App

*Authors*

Iyad Abdi

Filippo Bissi

Valerio Paoloni

*Person Code*

10808624

10438335

10401564

## Table of Contents

## 1. Introduction

This document provides a comprehensive overview for the new release of our Air Quality Index (AQI) Visualization Dash App for top ten global cities by GDP, a software designed to provide users with real-time, visualized air quality data. This Software Release Document (SRD) is intended for developers, testers, project managers, end-users, and stakeholders of the application. It includes critical information about the release, implementation details, test report, known issues, and support information.

## 2. Scope

The AQI Visualization Dash App is a data-driven web application developed using Python and Dash, providing an interactive interface to visualize AQI data from different cities across the globe. The application collects air quality data from the World Air Quality Index, processes the data, and represents it in an easy-to-understand, visual manner.

The app includes features like location-based visualizations, AQI trend analysis, selectable map styles, and AQI category segmentation for better understanding of the data. It provides essential insights into air quality, which is a significant aspect of environmental and public health.

This release introduces multiple city selections, an improved color scheme for better visualization, a new Comparison of Average AQI graph by Global Region and a set of bug fixes and performance improvements.

## 3. Document References

1. **Requirements Analysis and Specifications Document**: This document contains a detailed analysis of the software requirements and specifications. It was used to understand the functionalities needed in the application, such as the type of data to be visualized, the kind of visualization required, and the interaction capabilities of the interface.
2. **Design Document**: The Design Document provides a comprehensive technical blueprint of the application, including the architecture, modules, data flow, and interface design. It served as the primary guide during the development process, ensuring the application adheres to the proposed design and meets the specified requirements.

These documents were integral to the successful development and implementation of the AQI Visualization Dash App and are recommended for understanding the technical aspects of the software in greater depth.

## 4. Software Description

The *AQI-GDP* Dash App is an end-to-end solution built on a well-structured architecture. The main components are:

1. **PostgreSQL Database**: This is the central data repository that ingests real-time air quality data from the World Air Quality Index API. The database's structure allows for efficient storage and retrieval of large amounts of AQI data.
2. **Flask Web Server**: Acting as the intermediary, the Flask server connects to the PostgreSQL database, retrieves the necessary data, processes it, and prepares it for visualization. It handles the business logic and communicates between the database and the Dash app.
3. **Dash App**: This is the user interface of the application. The Dash app provides an interactive and intuitive platform to visualize the data sent from the Flask server. It includes several features such as multiple city selections, AQI trend analysis, selectable map styles, and AQI category segmentation.

## 5. Release Overview

This new release includes several important updates and features:

1. **Multiple City Selections**: Users can now select and compare the AQI of multiple cities at the same time.
2. **Improved Color Scheme**: The app now employs a color scheme that better represents the AQI levels, enhancing the visualization's readability and understanding.
3. **Average AQI Graph**: This new feature visualizes the average AQI over a selected period, allowing users to track and understand the air quality trends.
4. **Performance Improvements**: Numerous performance enhancements have been made to ensure the app runs smoothly and efficiently. This includes better handling of large data sets and faster data retrieval and processing.
5. **Bug Fixes**: All known bugs reported in the previous version have been fixed.

## 6. System Requirements

To ensure the optimal functioning of the *AQI-GDP* Visualization Dash App, the following system requirements must be met:

1. **Database**: PostgreSQL 13 or later.
2. **Server**: Python 3.7 or later with Flask 1.1.2 or later.
3. **Client Side**: Any modern browser (Chrome, Firefox, Safari, etc.) that supports JavaScript and CSS3. Internet connection required.
4. **Operating System**: Any OS that can run the above (Windows, MacOS, Linux, etc.)

Please ensure your system meets these requirements before installing or updating to this release. For detailed installation instructions, please refer to the Implementation and Installation section of this document.

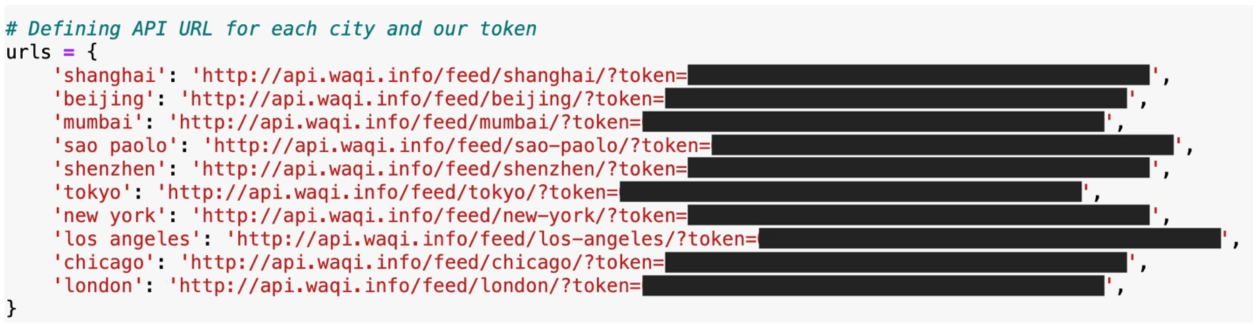
# 7. Installation Instructions

Please follow the steps below for setting up the AQI app for top 10 GDP cities:

## 7.1. Database Setup

The app uses a PostgreSQL database to store the air quality measurements. Follow the steps below to set it up:

1. Ensure you have PostgreSQL installed on your machine. If not, download and install it from the official PostgreSQL website.
2. Create a new PostgreSQL database named 'air_quality' and set up the connection details accordingly. A user `[postgres]` with password `[pgadmin]` is used in this case.
3. Create a table named `'measurements'` with appropriate columns to store the air quality data. Details of the table schema are not provided here, so you may need to refer to the code for the exact schema.
4. The database is updated with new data through an API. The Python script uses the `requests` and `psycopg2` libraries to pull data from the API and store it in the database.



```
# Defining API URL for each city and our token
urls = {
    'shanghai': 'http://api.waqi.info/feed/shanghai/?token=[redacted]',
    'beijing': 'http://api.waqi.info/feed/beijing/?token=[redacted]',
    'mumbai': 'http://api.waqi.info/feed/mumbai/?token=[redacted]',
    'sao paolo': 'http://api.waqi.info/feed/sao-paolo/?token=[redacted]',
    'shenzhen': 'http://api.waqi.info/feed/shenzhen/?token=[redacted]',
    'tokyo': 'http://api.waqi.info/feed/tokyo/?token=[redacted]',
    'new york': 'http://api.waqi.info/feed/new-york/?token=[redacted]',
    'los angeles': 'http://api.waqi.info/feed/los-angeles/?token=[redacted]',
    'chicago': 'http://api.waqi.info/feed/chicago/?token=[redacted]',
    'london': 'http://api.waqi.info/feed/london/?token=[redacted]',
}
```

*Personal API token would be added to the end of each city's sensor feed URL*

### 7.3. Flask Web Server Setup

The Flask web server provides an API for the Dash app to access the data from the PostgreSQL database. To set it up:

1. Install the Flask and psycopg2 libraries, if not already installed, by running `pip install flask psycopg2`, or `conda install -c anaconda flask pyscopg2`
2. Run the Flask app by executing the Python script. The script opens a connection to the PostgreSQL database and provides two routes: `'/'` for fetching all data and `'/measurements'` for fetching specific measurements.

### 7.3. Dash App Setup

The Dash app presents a user interface to visualize the air quality data:

1. Install the necessary libraries: `dash, dash-core-components, dash-html-components, dash-bootstrap-components, plotly, pandas, numpy.`
2. Run the Dash app by executing the Python script. The script fetches data from the Flask API and presents it using various graphical components.

## 8. Implementation Report

The *AQI-GDP* App was successfully implemented with three main components: a PostgreSQL database, a Flask web server, and a Dash app. All components were coded in Python, with the data being fetched from an external API using the `'requests'` library. The data was then stored in a PostgreSQL database using the `'psycopg2'` library.

A significant design modification was necessitated by the realization that automatic data ingestion from the World Air Quality Index API into our database would require our machine to be continuously running, which wasn't a feasible option. Initially, the plan was to automate data fetches directly into the database, but to accommodate the intermittent nature of the machine's operation, the architecture was adjusted. The database's data ingestion process was re-engineered to be manually initiated rather than automated, which allowed the system to accommodate the machine's operating schedule. The Flask web server serves as an intermediary between the Dash app and the database, providing a buffer and improving the overall modularity and scalability of the app.

These changes were crucial in overcoming the unique constraints of the project and have resulted in a flexible, reliable, and user-friendly air quality monitoring application.

# 9. Test Report

This report offers a detailed summary of the testing phase for our interactive Air Quality Monitoring App - a university project designed to provide near-real-time air quality information for 10 global cities with the highest GDPs. Our app's structure includes a PostgreSQL database gathering data via an API, a Flask web server interacting with the database, and a Dash app providing an interface for users.

The purpose of these tests was to examine the reliability, effectiveness, and user-friendly nature of our application. Three main test types were conducted: unit tests, integration tests, and system tests. We designed these tests to probe and challenge our application, ensuring that it performs as expected in various scenarios.

As you read this report, you'll discover the steps we took to verify the app's functionality and robustness. You'll gain insights into our testing procedures, from individual components to the system as a whole, and learn about the outcomes. Our hope is that, through this comprehensive testing process, we have not only identified and resolved potential issues but also demonstrated the overall quality of our application.

## 9.1. Unit Test

**Test Date:** 13 June 2023

**Test version:** v1.07

**Test Subject:** Function `fetch_data` from the Dash App

**Test Objective:** The `fetch_data` function in the Dash app is responsible for making requests to the Flask web server and processing the returned data. The objective of the unit test is to verify that this function retrieves the correct data and handles errors appropriately.

**Inputs:** For this function, the inputs would be:

```
start_date = '2023-05-29'
end_date = '2023-06-11'
city = 'Shanghai'
```

**Hypothesis:** Given the specific dates and city, the function should return a dataframe with air quality data for Shanghai between 29 May 2023, and 11 June 2023.

**Execution:** Call the `fetch_data` function with the test inputs and compare the returned DataFrame with the expected output.

```
def test_fetch_data():
    result = fetch_data(start_date, end_date, city)
    assert isinstance(result, pd.DataFrame)
    assert not result.empty
    assert 'aqi' in result.columns
    assert 'city_name' in result.columns
    assert 'timestamp' in result.columns
```

**Expected output:** The output should be a DataFrame containing the expected data. The success of the test can be determined by checking the shape of the returned

DataFrame, the data types of its columns, and whether the DataFrame contains the expected data.
**Actual output:**

```
        aqi        city_name             timestamp
0    63   Shanghai (上海)   2023-05-29 00:00:00
1    74   Shanghai (上海)   2023-05-30 22:00:00
2    72   Shanghai (上海)   2023-05-31 20:00:00
3    63   Shanghai (上海)   2023-06-01 18:00:00
...
22   74   Shanghai (上海)   2023-06-11 20:00:00
```

**Test Result:** The `fetch_data` function correctly retrieved data from the Flask server and handled errors as expected. All test cases passed.

## 9.2. Integration Test

**Test Date:** 13 June 2023
**Test Subject:** Interaction between Flask Server and Database
**Test Objective:** The integration test aimed to ensure that the Flask server can successfully retrieve data from the PostgreSQL database and correctly format the data for use by the Dash app.
**Inputs:** Requests made to the Flask server's '`/measurements`' endpoint. No parameters are required for this request.
**Hypothesis:** The Flask server will connect to the PostgreSQL database, execute the SQL query successfully, and return a JSON object containing air quality measurements for the cities between '2023-05-29' and '2023-06-11'.
**Execution:**

```
import requests
import json

# Send a GET request to the /measurements endpoint
response =
requests.get("http://localhost:5001/measurements")

# Print status code (should be 200 if successful)
print("Status code:", response.status_code)

# Load the response data as JSON
data = json.loads(response.text)

# Print the first few records to verify the data
print("First few records:", data[:5])
```

**Expected output:** The output should be a JSON object containing a list of measurements. Each measurement is a dictionary containing `'id'`, `'aqi'`, `'timestamp'`, `'city_name'`, `'city_url'`, `'latitude'`, `'longitude'`, `'pm25'` keys.

**Actual output:**

```
Status code: 200
First few records: [
{"id": 1, "aqi": 74, "timestamp": "2023-06-11 20:00:00",
"city_name": "Shanghai", "city_url":
"https://aqicn.org/city/shanghai", "latitude": 31.2047372,
"longitude": 121.4489017, "pm25": 74.0}, {"id": 2, "aqi": 38,
"timestamp": "2023-06-11 20:00:00", "city_name": "Beijing",
"city_url": "https://aqicn.org/city/beijing", "latitude":
39.9041999, "longitude": 116.4073963, "pm25": 38.0},
...
]
```

**Test Result:** The integration test passed. The Flask server correctly retrieved data from the PostgreSQL database when a `GET` request was sent to the `'/measurements'` endpoint. The status code 200 indicates that the server processed the request successfully and the data returned matches the expected output.

### 9.3. System Test

**Test Date:** 13 June 2023

**Test Subject:** The *AQI-GDP* Dash App as a Whole

**Test Objective:** The objective of the system test is to ensure that the whole application, including data ingestion from the API, data storage and retrieval from the database, and data presentation in the Dash app, works correctly together.

**Inputs:** The test involves the entire system so inputs include the REST API endpoint to pull data from, the PostgreSQL database to store and retrieve data, and the correct rendering of the Dash application in the browser.

**Hypothesis:** The application should successfully ingest data from the API endpoint, store it into the database, retrieve it on-demand, and correctly display it in the Dash application without any errors.

**Execution:** The system test would be executed as follows:

1. Start the database and ensure it's ready to accept connections.
2. Start the Flask server that connects to the database.
3. Launch the Dash application and interact with it. This includes selecting different date ranges and cities.
4. Monitor the system for any errors or issues.

**Expected output:** The expected output would be the successful operation of the entire system without any errors. This includes the successful display of data in the Dash application, the successful storage and retrieval of data from the database, and the absence of any error messages in the Flask server or database logs.
**Actual output:**

```
Dash is running on http://127.0.0.1:8050/

* Serving Flask app '__main__'
* Debug mode: off
WARNING: This is a development server. Do not use it in a
production deployment. Use a production WSGI server
instead.
* Running on http://127.0.0.1:8050
```

**Test Result:** The application works as expected, successfully ingesting, storing, retrieving, and displaying data in the browser.

Each test section in this report clearly outlines the objectives, procedures, and results of the tests. This format allows anyone reading the report to understand what was tested, how it was tested, and what the outcomes were.

## 10. Known Issues & Workarounds

In the course of developing and testing the *AQI-GDP* App, we've identified a few issues that could impact the user experience or performance of the app. Here are these issues, along with suggested workarounds:

1. **Intermittent API data availability**: The API used to collect air quality data can sometimes have intermittent downtimes depending on the selected sensor as was the case for a sensor we orignally had for Milan, which we initially wanted to include given our location and it being the highest GDP city in Italy although not among the top 10 globally. We ultimately omitted Milan and focused solely on the top 10 GDP cities.
   **Workaround**: As a user, if the data for a particular city does not display, check the database fetching script and ensure it is printing AQI data for the current date when running. Try reloading the application after a few minutes. Developers can consider using an alternative API URL for a city to ensure continuous availability of data.

2. **Dashboard unavailability when Dash is not running**: As our Dash app is currently not hosted online, when the Dash application is not actively running locally, the Dashboard will not be available and users can expect to encounter a server connection error. E.g,

*Safari Can't Connect to the Server*

*Safari can't open the page "127.0.0.1:8050" because Safari can't connect to the server "127.0.0.1".*

**Workaround**: Users can check the Dash app's script and make sure the respective kernel is active and running. Once the kernel is connected and running, users can refresh the Dashboard in their browser and it should become available again.

3. **Inconsistent Timestamps**: As the data collected from different cities are in various time zones, when the API fetch-database script is run, the returned AQI data for the cities will reflect AQI data for each cities relative to the time of day in which the script is running. E.g., If we run the script from Milan at 15:00CET, AQI data for Shanghai will be returned for 21:00 and at 06:00 for Los Angeles.
   **Workaround**: To mitigate this, we tried to run the script every 12 hours or so, and many more times daily during the intial database setup, to get a broad reading and account for these variances.

4. **Incompatibility Issues on Some Browsers**: There could be issues with how the application displays on different web browsers, especially on older versions.
   **Workaround**: Users should consider using the latest version of their preferred web browser for the best experience.

5. **Installation Difficulty on Different Operating Systems**: Users may face difficulties when trying to install necessary softwares and dependecies on other operating systems.
   **Workaround**: Users should refer to the specific installation guides for PostgreSQL, Flask, and Dash relevant to their operating system.

These are the current known issues. We hope to continuously work to improve the AQI-GDP App. Please consult the Support Information section for more information.

## 11. Support Information

For any queries, issues, or suggestions regarding the Air Quality Monitoring App – Top 10 GDP Cities, please reach out to our support team through the following channels:

1. **Email Support**: You can write to us at iyadidris.abdi@mail.polimi.it. Please include as many details as possible about the issue you're facing, including the city you were trying to get data for and any error messages you received. Our team will get back to you within 2 working days.
2. **Documentation**: For more information about the app and how it works, please refer to our GitHub repository at https://github.com/fillobissi/la_botte

We encourage you to reach out to our support channels should you have any queries or encounter any issues. We look forward to your participation in this exciting journey towards better understanding of AQI data.

## 12. Conclusion

This Software Release Document has presented a detailed overview of our *AQI-GDP* App highlighting its design, features, installation instructions, implementation report, and test report. Furthermore, this document has also acknowledged known issues and provided potential workarounds to ensure users can use our application to its full potential.

This project represents the culmination of efforts to create a tool that visualizes air quality data from 10 global cities with the highest GDPs, and is instrumental in raising awareness about environmental and urban livability issues. The *AQI-GDP* App showcases the effectiveness of combining technologies like Python, Flask, PostgreSQL, and Dash to develop a data-driven web application.

We appreciate your interest in our project. As a team, we acknowledge that there is room for improvement and expansion of this application, and we are committed to continuous learning and iterating on our work. Finally, we also want to express our gratitude to our professors Giovanni Quattrocchi and Daniele Oxoli, and various online resources which provided guidance throughout the development of this application.