



universität
wien

BACHELORARBEIT

DATA PRE-PROCESSING OF EVENT-LOGS AND ENVIRONMENTAL WEATHER DATA

Verfasserin ODER Verfasser

Antonio Gašparević

angestrebter akademischer Grad

Bachelor of Science (BSc)

Wien, 2023

Studienkennzahl lt. Studienblatt: UA 033 526

Fachrichtung: Informatik - Wirtschaftsinformatik

Betreuerin / Betreuer: Dipl.-Ing. Marian Lux

Contents

1	Introduction.....	5
1.1	Motivation	5
1.2	Problem	5
2	Related Work	6
3	Technologies	7
3.1	PostgreSQL.....	7
3.2	API (Application Programming Interfaces)	7
3.3	Docker	8
3.4	Git	9
3.5	PL/Python.....	9
3.6	pgAdmin4	9
4	Implementation	10
4.1	Architecture	10
4.2	Pre-processing of Data	11
4.3	Gap.....	11
4.4	Handling of different Data sources	11
4.5	Redundancy	12
4.6	Business Logic.....	12
4.7	Insertion of History Data.....	12
4.8	How to start the Project	13
4.9	Scripts	14
4.10	Environment Variables	15
5	Evaluation and Discussion	16
5.1	Diagram	16
5.2	Unit Tests/Results	18
6	Conclusion.....	19
6.1	Summary	19
6.2	Lessons Learnt	19
6.3	Future Work	19
6.3.1	GitHub.....	19
7	Acknowledgements	20

8	List of Tables	21
9	List of Figures	21
10	Bibliography	21

Abstract

This thesis outlines the most common challenges when combining multiple data sources into one database on weather data, shows a custom solution that not only resolves those issues, but it also creates a common ground for future machine learning projects in this field which involve the weather data generated from this automated PostgreSQL database. It is programmed to be used with any Linux or Windows operating system via Docker with expanding possibilities in the future. The use of Cronjobs with automated Shell scripts make it effortlessly usable for any user with little to no experience. It addresses the pitfalls of databases and working with business logic inside of a database instead of only using it in the application layer and shows how to integrate that business logic inside of it. Views are used to display the acquired data for future machine learning projects associated with weather data to train on.

1 Introduction

1.1 Motivation

Databases serve as the backbone of every business today. This bachelor thesis was commissioned by the Workflow Systems and Technology Research Group at the University of Vienna and has given me the opportunity to learn more about databases. Specifically, PostgreSQL databases were the focus of this thesis, working with bulk of data which need to be pre-processed to be usable for different purposes after storing, Python as programming language inside of the Postgres database and learning about the pitfalls when combining multiple sources of data into one.

The sources are MetGIS¹ and OpenWeather², which are the foundation for the whole structure of the database. All data was provided by the APIs of the two sources, mostly a combination of historical weather data and current/forecast data.

This project is here to shed light on how automated databases work in its core and how this could revolutionize the effectiveness of data storing and collecting. It can serve as foundation for future automation projects in this field and the database can be used for machine learning algorithms to train on bulk collected data.

1.2 Problem

There were many difficulties when it comes to an automated Database structure. The task was to combine weather Data from multiple Sources inside of one united PostgreSQL Database, which should acquire Data automatically from different sources, be validated for their values by using researched approaches and displayed for machine learning algorithms afterwards, so the acquired data can be used.

One of the problems was to handle so called Gaps in datasets, which can be a problem when combining multiple sources. This thesis shows a solution to the gap problem in datasets and acknowledges other possibilities on how to provide a solution to this problem. These solutions however cannot be applicated on every kind of gap problem.

¹ <https://www.metgis.com/de/> Last Visited: 17.09.2023

² <https://openweathermap.org/> Last Visited: 17.09.2023

2 Related Work

Historical Data is one of the main components for this project since it is mandatory for calculating accurate values for the SQL views. [1] shows how important automated weather data collection is for making decisions in agriculture and forestry. Also, they address the problem with missing values and analyse methods on how to handle missing values.

Data preprocessing is a crucial step in acquiring usable data for the databases. In [2] the major problems of missing values and redundancies are getting addressed, which is valuable for the preparation of the datasets given by the sources MetGIS and OpenWeather. Also, the Data preparation, which is part of the Data preprocessing, is a major part of this thesis, as data that is gathered through the mentioned sources needs to be validated and checked for inconsistencies, missing entries and out of range values. Those missing entries also pose another challenge named gaps.

Data gaps pose significant challenges in database applications, impacting the quality of decision-making processes. First it was necessary to assess what those data gaps are and how these happen to exist inside of data. [3] was used to understand the severeness of data gaps and how the lack of data can be a problem for making decisions on a larger scale. Many datasets were combined and not every source had the necessary data provided to make decisions based of it.

Since everything is handled inside of the PostgreSQL database, the question arises if it is beneficial to implement the business logic inside of the database or the application layer. The application layer in this context is the programming language PL/Python, which is used to enable API queries to gather data from the sources. The storing and queries of the data are handled by PostgreSQL. [4]

3 Technologies

3.1 PostgreSQL

PostgreSQL is an open-source database that utilizes the SQL language and enhances it with many features. Some of those features are Extensions, Procedural Languages, SQL to name a few, but there are many more. PostgreSQL has been around for more than 35 years and has earned a strong reputation amongst the other competitors. One of the important points why PostgreSQL is used in this project is, that it runs on all major operating systems, which helped developing an automated database for Linux distributions and Windows seamlessly. This project is running on postgres:15.3-bullseye which is one of the more current versions of PostgreSQL.³

3.2 API (Application Programming Interfaces)

An API is an Interface used to communicate between two software components. In this project RESTful API is used to acquire data from MetGIS and OpenWeather via GET-Request. The data is exchanged via HTTP between client and server.⁴

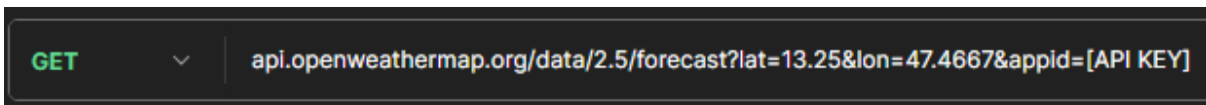


Figure 1: Example API Call

An example of an API Call is shown in Figure 1. The server is getting the request for the forecast data at the given latitude and longitude. For the [API KEY] a valid key needs to be inserted to work, the key can be created at the OpenWeather⁵ website. The tool for testing API calls in this project was Postman⁶. Postman is an API platform for building and using APIs. It is used to test and troubleshoot API Requests/Responses.

³ <https://www.postgresql.org/about/> Last Visited: 17.09.2023

⁴ <https://aws.amazon.com/de/what-is/api> Last Visited: 20.09.2023

⁵ <https://openweathermap.org/> Last Visited: 17.09.2023

⁶ <https://www.postman.com/product/what-is-postman/> Last Visited: 20.09.2023

```

{
  "cod": "200",
  "message": 0,
  "cnt": 40,
  "list": [
    {
      "dt": 1695286800,
      "main": {
        "temp": 303.38,
        "feels_like": 310.38,
        "temp_min": 303.38,
        "temp_max": 303.74,
        "pressure": 1008,
        "sea_level": 1008,
        "grnd_level": 1007,
        "humidity": 78,
        "temp_kf": -0.36
      },
      "weather": [
        {
          "id": 801,
          "main": "Clouds",
          "description": "few clouds",
          "icon": "02d"
        }
      ]
    }
  ]
}

```

Figure 2: Example Output for API Get Request

In Figure 2 an example of a JSON Response is shown. The provided data represents only a subset of the complete JSON dataset, as displaying the entirety would be overly extensive.

3.3 Docker

Docker⁷ is a containerization platform for developing, shipping, and running applications. Docker is used in this project as platform independent tool to use the automated PostgreSQL database on Linux or Windows. It is also used for testing the datasets with pgAdmin4. Docker also provides us with an isolated and scalable environment, which is very suitable for this project.






	database	-	Exited	
	automatedpostgres-1	database-automatedpostgres	Exited	5432:5432 
	pgadmin-1	dpape/pgadmin4	Exited	5050:80 

Figure 3: Docker Container Example

In Figure 3 an example is shown of the architecture in docker, that is defined in the docker-compose.yml for this project. A docker-compose⁸ file is used to run multi container applications like this project (PostgreSQL, pgadmin4). An example of the docker-compose.yml can be seen in Figure 4.

⁷ <https://docs.docker.com/get-started/overview/> Last visited: 20.9.2023

⁸ <https://docs.docker.com/compose/> Last visited: 20.9.2023


```

version: '3.8'

services:
  automatedpostgres:
    build:
      context: ..
      dockerfile: database/Dockerfile
    env_file:
      - .env
    volumes:
      - postgres-data:/var/lib/postgresql/data
      - ../sql:/sql
      - ../helperScripts:/helperScripts/
      - ../weather:/jsonfiles
    networks:
      - pgnetwork
    ports:
      - "${POSTGRES_PORT}:5432"

```

Figure 4: docker-compose.yml File Example

3.4 Git

Git is a free and open-source version control system which is used as distributor for the code. This code can be used by different developers to maintain the project by branching and merging.⁹ Git is easy to understand, learn, provides a good backup and portability of the code written.

3.5 PL/Python

PL/Python is implemented into the PostgreSQL database with an extension. It allows to use python code inside of the database to enable REST-APIs to be called and handle the data that is delivered for the project.¹⁰

```
CREATE EXTENSION plpython3u;
```

Figure 5: Example for extension file

3.6 pgAdmin4

pgAdmin4 is a feature rich open-source administration platform for PostgreSQL.¹¹ It is available for the most frequently used operating systems and therefore fitting for this project. pgAdmin4 is embedded into this project via a docker container.

⁹ <https://git-scm.com/about> Last Visited: 20.09.2023

¹⁰ <https://www.postgresql.org/docs/current/plpython.html> Last Visited: 20.09.2023

¹¹ <https://www.pgadmin.org/> Last Visited: 20.09.2023

4 Implementation

4.1 Architecture

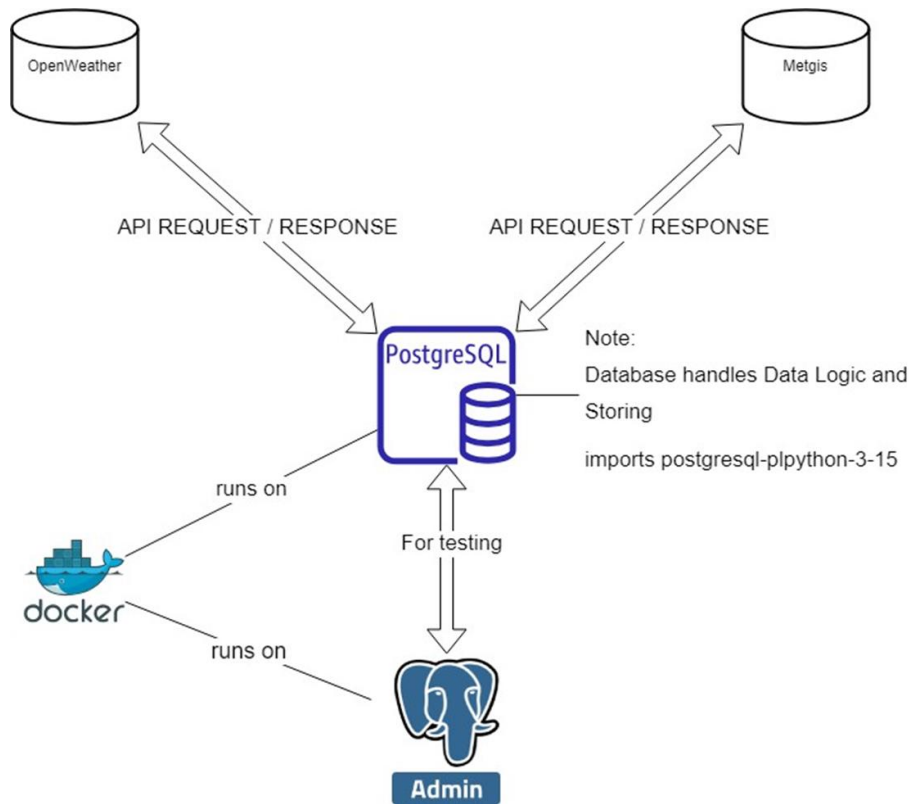
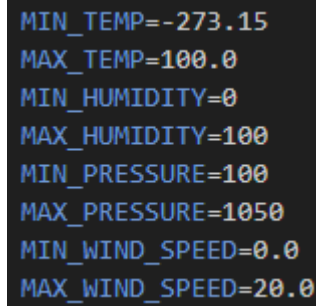


Figure 6: Architecture

The project consists of three main components, which include a PostgreSQL database, pgadmin4 and the external services from Metgis and Openweather. The PostgreSQL database and pgadmin4 run inside of docker containers. These containers can be used in Linux or Windows operating systems. The logic of the data is handled inside of the database to reduce overhead with an extra server. The database also handles all API requests to the external services to acquire data for our calculations. After the data was acquired and validated for correctness, the views can be used to extract the information stored inside of the database. These calculations can be used to train machine learning algorithms in the future.

4.2 Pre-processing of Data

The data that is getting saved inside of the database for each service is checked for any “null” values and predefined range checks. These range checks are being set over the .env File also called environment File. This file sets all necessary environment variables for the operating system. An example of the environment file can be seen in Figure 7.



```
MIN_TEMP=-273.15
MAX_TEMP=100.0
MIN_HUMIDITY=0
MAX_HUMIDITY=100
MIN_PRESSURE=100
MAX_PRESSURE=1050
MIN_WIND_SPEED=0.0
MAX_WIND_SPEED=20.0
```

Figure 7: Part of Environment File

4.3 Gap

The Gap mentioned in the related work, is solved via inserting predefined values into empty data sets. This ensures that we don't have to handle null errors from inserting new values over the APIs. It is also checked that the values are in the given data range as mentioned above.

4.4 Handling of different Data sources

First, we have eight different tables for saving the gathered data inside of the database. Two are for the data of the Metgis API, two are for the data of the OpenWeather API. The other four tables are used for historical data. An analysis of the responses from the APIs was necessary to evaluate how the tables should be built and which data it needs to include, so it can be stored properly. The tool that was used to analyse the data is called Postman, as already mentioned in 3.2. With this knowledge it was possible to create the tables properly suited. The whole JSON file that was transmitted via HTTP Response is stored in the table and can be accessed with SQL views or simple SELECT statements. For example, Select * from Table1, which can be executed via pgAdmin4.

4.5 Redundancy

For redundancy a good solution is to use normalization in the structure of the database. Normalization is used to reduce errors and redundancy inside the database. There are multiple forms of Normalization which can be applied to a database. In this project it was not possible to achieve the third normalization form because it was necessary to store all the data in separate tables from the different sources. For example, there are multiple internal values and ids given by the sources which are not necessary for the calculations done in the SQL views, so they are stored inside of the table for that purpose but is not considered for the views and is simply ignored.

4.6 Business Logic

In this project the business logic was implemented inside of the database since it's an automated PostgreSQL database standalone. Some of the logic is implemented inside of the database with PL/Python for inserting data inside of the history tables, for example. Usually when business logic is applied in the application layer of the project it is easier to maintain and to change for different purposes. When the business logic is applied inside the database, it is more efficient to the application layer since it is already at the root of the data, the maintenance is like the application layer. Another advantage is that it can be visualized with SQL views which makes it very time efficient.

4.7 Insertion of History Data

The history data is inserted through already present JSON files which must be stored inside of the weather folder to fill the four history tables. The JSON files are provided by the MetGIS API. After the project is started and initialized properly it will automatically read all JSON files inserted into the weather folder. These files are read with PL/Python inside of an SQL Function defined in the database.

4.8 How to start the Project

All these steps are provided inside of the README.md as well. The project is started with an entry script. This entry script is found inside of the GitHub Repository, which can be found at the end of the paper. Make sure to have Docker installed on the OS. For Windows operating systems the run.bat is simply started and executed. If a Linux distribution is used as operating system, docker needs to be installed either with the dockerscript.sh that's provided, or with the commands that are provided inside the README.md file.

The project starts all the necessary containers automatically and there is no need to start anything else except the first script.

To examine the database simply open a browser of your choice and enter localhost:5050. If some other port is specified inside of the .env file change 5050 to that and enter pgAdmin4. The Login credentials for pgAdmin4 are also found inside of the .env file which can be change. The database will itself fill the history tables if the files are provided inside of the weather folder, the current and forecast data is getting accumulated through an automated script which get executed in a specified timeframe.

4.9 Scripts

The scripts provided in the repository should only be changed if the requirements change. For example, if multiple locations are used for the project, it is necessary to import multiple history JSON files in the folders to make it work. The scripts that need to be changed are `getDataFromFiles.sh` and `cronStart.sh`.

The `getDataFromFiles.sh` only needs to be adjusted, that the new JSON files are parsed properly into the respective history table.

```
#!/bin/bash

sleep 5

psql -U postgres -d AutomatedPostgres -c
"select import_percipitation('/jsonfiles/Werfenweng_MetGIS_Precipitation_hourly_20200901-20201101.json')"
>> status.txt
```

Figure 8: Part of `selectDataFromFiles.sh`

The `cronStart.sh` only needs to be changed if the time intervals need to be set differently. A tutorial on how to change the Cronjob is provided in the script itself.

```
#!/bin/bash

#set cronjobs which should be executed by the crontab
(crontab -l ; echo "*/5 * * * * /helperScripts/getDataCron.sh") | crontab -
(crontab -l ; echo "0 8 * * * /helperScripts/getDataForecastCron.sh") | crontab -
./helperScripts/getDataFromFiles.sh

# * = any value
# minute hour day(month) month day(week) is the syntax for the cronjob
# */5 means every 5 minutes
# for further reference use this website to make it easier to set cronjobs correctly (https://crontab.guru/#\*\_\*\_\*\_\*)
```

Figure 9: `cronStart.sh` Script

4.10 Environment Variables

The Environment Variables are listed in this table and can be changed to suit the needs of the user.

VARIABLE NAME	VALUE
POSTGRES_API_KEY_OPEN	API KEY
POSTGRES_API_KEY_METGIS	API KEY
POSTGRES_USER	User for postgres Database
POSTGRES_PASSWORD	Password for postgres Database
POSTGRES_DB	Name of the postgres Database
POSTGRES_PORT	Port of the postgres Database
POSTGRES_CITY	City which weather data should be collected from
POSTGRES_LAT	Latitude of the city
POSTGRES_LON	Longitude of the city
POSTGRES_ZIPCODE	Zip code of the city
POSTGRES_UNITS_OPEN	Metrics for the Openweather API (use 'metric' as default)
POSTGRES_OPENWEATHERLINK	Link for Openweather (please take it from the template)
POSTGRES_METGISLINK_CURRENT	Link for Metgis (please take it from the template)
POSTGRES_OPENWEATHERLINK_FORECAST	Link for Openweather (please take it from the template)
POSTGRES_METGISLINK_FORECAST	Link for Metgis (please take it from the template)
PGADMIN_DEFAULT_EMAIL	Email for pgadmin Login
PGADMIN_DEFAULT_PASSWORD	Password for pgadmin Login
MIN_TEMP	Minimum Temperature
MAX_TEMP	Maximum Temperature
MIN_HUMIDITY	Minimum Humidity
MAX_HUMIDITY	Maximum Humidity
MIN_PRESSURE	Minimum Pressure
MAX_PRESSURE	Maximum Pressure
MIN_WIND_SPEED	Minimum Windspeed
MAX_WIND_SPEED	Maximum Windspeed

Table 1 Environment Variables

The diagram illustrates the system architecture. At the top, two data sources, 'OpenWeather' and 'Metgis', are shown as cylinders. Arrows point from both to a central 'PostgreSQL' database, represented by a cylinder with a square top. A callout box points to the PostgreSQL database with the text: 'collects Data over Cronjobs which are defined in cronStart.sh and saves them in tables'. Below the data sources, a script icon (notepad) is labeled 'run.sh / run.bat'. An arrow points from this script to the 'docker' logo (a blue whale), with the text 'starts Dockerfile over Docker compose' above the arrow. Another arrow points from the 'docker' logo to the 'PostgreSQL' database, labeled 'creates'. At the bottom right, an 'Admin' user icon (a person) is shown. An arrow points from the 'PostgreSQL' database to the 'Admin' icon. A long arrow also points from the 'Admin' icon back to the 'PostgreSQL' database, indicating a feedback loop or data retrieval.


```
version: '3.8'

services:
  automatedpostgres:
    build:
      context: ..
      dockerfile: database/Dockerfile
    env_file:
      - .env
    volumes:
      - postgres-data:/var/lib/postgresql/data
      - ../sql:/sql
      - ../helperScripts:/helperScripts/
      - ../weather:/jsonfiles
    networks:
      - pgnetwork
    ports:
      - "${POSTGRES_PORT}:5432"

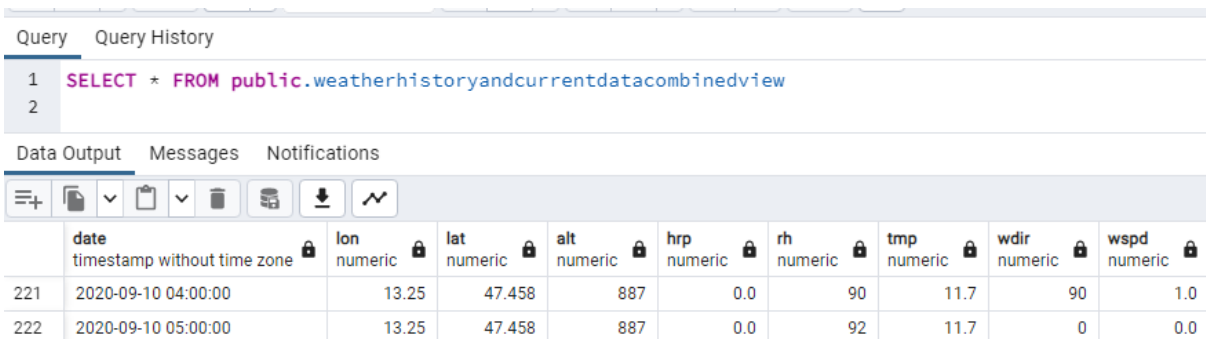
  pgadmin:
    image: dpage/pgadmin4
    volumes:
      - ./pgpass:/pgpass
      - ./servers.json:/pgadmin4/servers.json
      - pgadmin-data:/var/lib/pgadmin
    env_file:
      - .env
    networks:
      - pgnetwork
    ports:
      - "5050:80"

volumes:
  postgres-data:
  pgadmin-data:

networks:
  pgnetwork:
    external: true
```

Figure 11: docker-compose.yml with removable objects in red squares

5.2 Results



The screenshot shows the pgAdmin4 interface. At the top, there are tabs for 'Query' and 'Query History'. The 'Query' tab is active, showing a SQL query: `1 SELECT * FROM public.weatherhistoryandcurrentdatacombinedview` and a second line '2'. Below the query editor, there are tabs for 'Data Output', 'Messages', and 'Notifications'. The 'Data Output' tab is active, displaying a table of results. Above the table is a toolbar with icons for various actions like copy, paste, and refresh. The table has 11 columns: 'date' (timestamp without time zone), 'lon' (numeric), 'lat' (numeric), 'alt' (numeric), 'hrp' (numeric), 'rh' (numeric), 'tmp' (numeric), 'wdir' (numeric), and 'wspd' (numeric). The first two rows of data are shown, with row numbers 221 and 222 in the first column.

	date timestamp without time zone	lon numeric	lat numeric	alt numeric	hrp numeric	rh numeric	tmp numeric	wdir numeric	wspd numeric
221	2020-09-10 04:00:00	13.25	47.458	887	0.0	90	11.7	90	1.0
222	2020-09-10 05:00:00	13.25	47.458	887	0.0	92	11.7	0	0.0

Figure 12: Example Output of History and Current Data View in PgAdmin4

The results were shown with pgAdmin4 as in Figure 12. The two views that are present inside the database can be shown via a simple request over pgAdmin4 as shown above. The data that is collected over the scripts with the functions implemented in PL/Python, can be accessed over these views. This data can then be used for further investigations and analysis.

6 Conclusion

6.1 Summary

Data pre-processing of Event-Logs and environmental weather data shows related works in the field of databases and weather data. It addresses one of the most common problems when combining multiple sources into one database which is gaps in data sets. Many technologies were used to ensure that the project can run on multiple operating systems. It shows the implementation of a custom-made automated PostgreSQL database which utilizes PL/Python for API calls to the sources MetGIS and OpenWeather as well as extracting data from the historical JSON files given inside of the weather folder.

6.2 Lessons Learnt

Interestingly I didn't think that the permissions in Linux could cause so many problems towards executing scripts inside of a docker container. After learning on how to solve this problem I could focus on the more important tasks.

This project helped me a lot to start developing platform independently. It also has shown me what challenges come with programming that way.

6.3 Future Work

This project can be used for future projects as fundamental starting point which work with training machine learning algorithms to train on weather data. For example, to learn when a big temperature spike could occur.

6.3.1 GitHub

The project can be found on <https://github.com/Angas96/automatedPostgres> and can be used freely. Feel free to insert any suggestions over an issue on GitHub.

7 Acknowledgements

I would like to thank Dipl.-Ing. Marian Lux for his involvement, contribution and help on this project.

8 List of Tables

Table 1 Environment Variables	15
-------------------------------------	----

9 List of Figures

Figure 1: Example API Call.....	7
Figure 2: Example Output for API Get Request	8
Figure 3: Docker Container Example.....	8
Figure 4: docker-compose.yml File Example.....	9
Figure 5: Example for extension file	9
Figure 6: Architecture.....	10
Figure 7: Part of Environment File.....	11
Figure 8: Part of selectDataFromFiles.sh	14
Figure 9: cronStart.sh Script.....	14
Figure 10: Diagram of the system functionality.....	16
Figure 11: docker-compose.yml with removable objects in red squares	17
Figure 12: Example Output of History and Current Data View in PgAdmin4	18

10 Bibliography

- [1] F. Rafii and K. Tahar, "Collection of Historical Weather Data: Issues with Missing Values," *SCA '19: Proceedings of the 4th International Conference on Smart City Applications*, vol. 8, no. 19, p. 1–8, 2019.
- [2] S. García, J. Luengo and F. Herrera, "Tutorial on practical tips of the most influential data preprocessing algorithms in data mining," *Knowledge-Based Systems*, pp. 1-29, 15 April 2016.
- [3] E. Osuteye, C. Johnson and D. Brown, "The data gap: An analysis of data availability on disaster losses in sub-Saharan African cities," *International Journal of Disaster Risk Reduction*, pp. 24-33, 14 September 2017.
- [4] F. Dimitri, *The Art of PostgreSQL*, London: Lulu.com, 2022.
- [5] M. Fathi, M. Haghi Kashani, S. M. Jameii and E. Mahdipour, "Big Data Analytics in Weather Forecasting: A Systematic Review," *Archives of Computational Methods in Engineering*, pp. 1886-1784, 28 Juni 2021.

