Making Power Quality Monitoring more accessible

By emulating grid behavior and creating a tool to visualize power quality data.

Menno Ketelaar

***Abstract*—Power quality monitoring is becoming more and more relevant, due to the increasing market penetration of renewable energy sources. In addition to this power quality monitoring is currently very costly and only done by large power companies. This suffices if you are connected to a large stable grid, but what if you want to make use of your own generated power more effectively, or you have to manage a smaller microgrid, mostly reliant on renewable sources. For this reason, this paper first analyzes power quality and current power quality monitoring. After which the problems with current power quality monitoring are identified. Then a power-quality monitoring (PQM) device is introduced and analyzed, such that its behavior can be emulated. Then a visualization tool is used to make its data more accessible. This work is done to increase the effectiveness of the PQM-device and to show what can be done with power quality monitoring if done effectively.**

I. Introduction

Microgrids are a great way to create an electrical infrastructure in remote places, where there would normally not be any electricity at all. Microgrids in combination with renewable energy sources have large potential in developing countries, due to the social, economic and environmental benefits imposed by such a microgrid [1]. In addition to that microgrids are also seeing an increase in relevance in developed countries, where renewable energy sources are being utilized more frequently to reduce carbon emissions [2]. In developing countries, microgrids are likely to have outages due to the the inconsistent nature of sustainable energy and the unknown use cases of these microgrids. In developed countries the increasing penetration of renewable energy sources can cause the grid frequency stability to suffer, which in turn can cause grid instability and in worse cases grid failures [3] [4]. To reduce the risk of grid failures and therefore, increase the return of microgrids, data must be collected in order to gain insight into what is causing these malfuntions. Therefore new ways to monitor and maintain these microgrids should be implemented and designed. To introduce the topic of power quality a paper by Leferink [5] on increasing Electromagnetic interference (EMI) issues will be discussed. In addition to that 4 other scientific papers on power quality monitoring will be discussed. Firstly the paper that discusses the design of a relatively cheap and simple power quality monitor by Grootjans and Moonen [6]. Then Kilter *et al.* [7], discuss different purposes of power quality monitoring and present guidelines on what types of measurements should be performed to achieve this purpose are summarized. After which Kamyshev *et al.* [8], discuss methods to detect events, such as devices being connected to the grid, based on measurements performed on the grid. Even though the work in this project does not have a direct relation to the work presented in the paper, the methodologies presented can be useful in the future if the sampling rate of the PQM-device is changed to a higher sampling rate, as attempted by de Graaff [9]. The last paper has a similar goal to this project, as it discusses the OrigAMI system in Poland, which is a system of over 180,000 smart meters. Smolenski *et al.* [10], introduce a method to use the current smart meters in a way such that PQ issues can be located and resolved. These issues mostly occur in Poland, due to the increasing amount of prosumers, i.e. consumers who are also delivering energy to the grid, using sources like photovoltaic installations. All in all these papers are supposed to function as preparation and background knowledge to create a data classification algorithm and practical user-interface for the device designed in paper [6].

II. Problem statement

Currently data is often only available to grid operators and if there is data available for the consumer, it is often not presented in a way that is easily perceived. Not only is the data not available, it is also not recorded frequent enough that finding out what devices cause PQ-events is straightforward. Regular smart meters take measurements every 10 to 15 minutes, like in Poland, where the reactive and active power measurements are taken every 15 minutes and voltage and current measurements every 10 minutes [10]. Therefore, in order to make power quality monitoring accessible for consumers, or grid operators of small grids, a new power quality monitoring device has been designed, which takes measurements every 2 seconds and can be deployed easily.

The power quality monitor device designed in [6] saves data, such as the RMS-voltage, frequency and phase and sends it to a server for storage. The data is in tabular format as can be seen in Figure 1 and since no PQ events are flagged, the data from the device is difficult to interpret for the user. This is why a data qualification algorithm should be designed, coupled to a useful user-interface. By classifying and detecting events happening in the grid connected to the PQM, the cause of grid malfunctions can be more easily identified. In addition to that there should be an interface like in the OrigAMI system [10], of which a screenshot can be seen in Figure 2 with an overview of all deployed devices. This overview will make it easier to monitor many different points in the low voltage grid. In the case of the OrigAMI system the voltage is measured at transformer stations [10] and shows in red when the voltage is exceeding regulatory levels, or green when the voltage is within the limits. Using the PQM-device designed, the voltage can be measured at the consumer level of the grid, thus closer

to the possible sources of disturbances, such as private PV-installations or high impedance loads.

Line current RMS	Voltage RMS	Line mean active power	Line mean reactive power	Voltage frequency	Line power factor	Phase angle	Line mean apparant power	Metering status	System status	Temperature	Humidity	Timestamp
0	231713	0	-3	50	0	35	49	0	0	32	36	2024-06-23T19:55:31Z
0	231753	-2	-6	50	0	19	49	0	0	32	36	2024-06-23T19:55:33Z

Fig. 1: A screenshot of part of the current way data is displayed

Therefore, this project can allow end users to deploy affordable PQ-monitors in their house to monitor at what times power is consumed and produced. This data can be used to optimize their power usage, such that they consume more of their own power, instead of power from the power company, which results in costs saved on both consuming power, but also delivering power to the grid. As in the Netherlands, prosumers now have to pay a fee to the power company for delivering power to the grid [11].

In addition to that it can also be deployed in microgrids, to be used by the grid operator to gain insight in power production and consumption. The device can be installed in line with the power source, for example a PV-array. In this case it is possible to monitor power production, in combination with one or multiple devices, monitoring power consumption, the grid operator can adapt consumption to reduce the chance of grid failures. In addition to this, the PQM-devices can be used to find points where disturbances occur, which allows the operator to install a small energy storage device, to combat these disturbances [10].

Therefore the goal of this project is to find out, how can power quality monitor data be processed and displayed in a user-interface, such that causes of grid malfunctions can easily be identified and resolved.

III. Related work

The first paper by Leferink [5] introduces the topic of conducted interference. Conducted interference is interference,

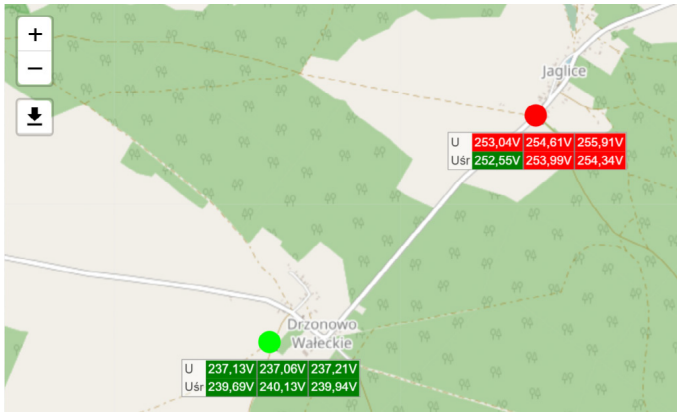

Fig. 2: Screenshot of the OrigAMI system interface [10]

which spreads from components which are physically connected to each other via the grid. In the past this used to consist mainly of the main hums and power supply harmonics and flicker, however over time more devices have started to create conducted interference. In 1987 a study was performed on the grid in 4 different environments. This study measured many voltage surges and transients, which were mostly caused by lightning or relay switching in devices, since these were the only known causes of conducted interference back then, these are the only causes the following electromagnetic compatibility (EMC) directive [12] attempted to address. After this EMC directive was released by the European Union, regulations were installed which then prompted industrialists to start protecting their devices against these causes. In recent years no surges and transients were measured, which is presumably caused by the protection circuits also working on devices connected to the same grid in parallel. Although this directive appears to have addressed the most common EMI issues, other EMI challenges caused by conducted interference persist. Leferink states that they are currently observing an increase conducted interference causing EMI issues. Currently society is moving more and more towards creating so called smart cities. From an EMI perspective, these are cities which have many energy efficient power electronic devices and measuring devices attached to the same grid. The energy efficient devices are often non-linear, which results in more reactive power, which is often overlooked when calculating power requirements. The measuring devices are there to gain insight into the performance of the network, but they also cause interference themselves. Using many of these devices will require additional protective devices, like varistors and filters to be used, which in turn will causes large current spikes between devices. All of these factors combined result in EMI issues in the 2-150 kHz range. As of right now not enough attention is paid to these EMI issues, which could cause problems in future cities, with even more non-linear and power quality measuring devices.

The second related work is a paper describing the design of a cost-effective and easy to implement power quality monitor (PQM), by Grootjans and Moonen [6]. This paper describes the design of an earlier iteration of the device for which a user interface and data processing algorithm will be created. This device is designed for use in multinode systems, where data will be sent to a central unit for processing. The device uses an ATM90E26 energy meter and a WSEN-HIDS humidity and temperature sensor, which connects to an ESP32, where the data is collected every 2 seconds and then grouped in chunks before it is uploaded to the server. Since this data is collected every 2 seconds and stored on an SD-card, detecting behaviour of devices over a longer time period is simple. However, if a certain device is plugged in and instantly overloads the grid, then the device will not be of much use. One can argue that this is irrelevant as when such a device is plugged in the user will instantly notice that the grid malfunctions, however it still should be taken into account that the sample rate may be too low for certain applications. According to the paper the analog signal is also connected to an ADC of the ESP32, which could allow the device to take more samples of the signal directly, however this may prove ineffective if the SD-card storage is

TABLE I: Measurements produced by the device

Measurement	Unit	Description
Line Current	A_{RMS}	Current
Voltage	V_{RMS}	Voltage
Line mean active power	W	Used power
Line mean reactive power	VAR	Power flowing back and forth due to capacitive and inductive loads
Frequency	Hz	Frequency
Line power factor	kVA	Ratio of active to apparent power
Phase angle	$^\circ$ (degrees)	Phase angle with respect to calibrated value
Line mean apparent power	VA	Sum of active and reactive power
Forward active energy	Wh	Consumed active energy
Reverse active energy	Wh	Exported active energy
Absolute active energy	Wh	Total active energy
Forward reactive energy	Wh	Consumed reactive energy
Reverse reactive energy	Wh	Exported reactive energy
Absolute reactive energy	Wh	Total reactive energy
Temperature	°C	
Humidity	g/kg	

filled quickly in combination with an unreliable network to send the data to a central server for processing. In order to find points of interest when an error occurs somewhere in the network, multipoint power quality monitoring is essential. Due to the affordability of this device, it would be easy to implement multiple of these in a microgrid. The device also has additional connections for extra sensors, for example for measuring electric fields, which could make it applicable to more complex grids with more devices connected as well. This device is able to produce the measurements on the variables in Table I with an error rate below 1%, which technically would classify it as a type B energy meter. This device will function as the backbone of the project.

Kilter *et al.* [7], recognizes the increase in power quality monitoring and attempts to create a guideline for how power quality should be measured for different applications. They discuss 6 main reasons for PQM, for each they have listed requirements in a table. It is to note that these guidelines are based on research performed with respect to transmission system operators and distribution system operators, which are in most cases large companies controlling major grid structures. These companies have large budgets for ensuring their power quality is in order and will therefore be able to afford expensive measuring equipment. In the case of the cost effective PQM-device [6], some guidelines can be adhered to somewhat loosely, as the device is mainly supposed to give insight into what causes microgrids to malfunction, which is on a much smaller scale. The paper recommends permanent monitoring at high and medium voltage points, which may not be relevant for microgrids; however, it would likely be beneficial at the energy source. Nonetheless, this is not the intended use of the device. The use case in this project is mainly troubleshooting and performance analysis, for which the requirements can be seen in a table from [7] in Figure 3.

For troubleshooting a sampling rate 500 kHz is recommended, which is a lot higher than the 0.5 Hz the device produces data at now, which may introduce difficulty when encountering issues like transients or large inrush currents.

A paper by Kamyshev *et al.* [8] describes two methods of energy disaggregation, which is a way of deconstructing signals into more single appliance signatures. Aplliance signatures are recognizable patterns within the grid current measurements, from which can be concluded what type of device is connected. Gaining insight into connected devices could help to find which devices often cause grids to malfunction. This paper discusses the drawbacks present in disaggregation, due to overfitting on limited data and proposes a physics based solution on both low and high sampling rate energy measurements. Previously generated data was used to model different appliances, however this is often limited to several appliances, while in real households dozens of devices are connected to the grid. Therefore, this paper focuses on methods to generate appliance signatures, based on prior knowledge on the characteristics of an electrical device. The methods this paper discusses can be divided into two categories, high and low sampling rates. The high sampling rate method [8], samples the current of the grid with a very small sampling period. This allows for a precise signature of a device to be recorded. The spectrum of the frequency harmonics of this signature are then described using a set of complex variables. For the low sampling rate method [8], primitive cycles are detected, primitive cycles here mean continuous non-zero power consumption. These primitive cycles can be deconstructed into five basis functions, which can be multiplied by each other to form most primitive cycles found in the REDD [13] and UK-DALE [14] datasets, that were used in the paper. Using the synthetic dataset can prove useful if a disaggregation algorithm is used to identify connected devices. Considering that the device discussed in [6] only samples once every 2 seconds, only the low sampling rate methods are interesting at this time, however the high sampling rate method could also be considered in the future if the device would be set up differently, as should be possible according to [6].

A different application of power quality monitoring, which can also be done with the device mentioned in [6], is described by Smolenski *et al.* [10]. This paper describes how monitoring the power quality in different parts of the grid can help power companies to gain insight into where they should invest in power storage. In the case of this paper it is not about microgrids, but more about managing an increasing amount of energy sources installed by consumers, such as PV-installations. These new power sources are all dependent on the weather, causing the production to be unstable. When there is a lot of power generated, but not used the power quality of the grid drops, due to an increasing RMS voltage. This happens locally, making it very difficult for grid operators to manage these PQ-issues. Therefore, there is a regulation which states that power generation with PV-panels should be stopped when the voltage measured is above 253 V [10]. This would however result in these PV-installation inverters to keep switching on and off in certain intervals, which is undesirable for both the prosumer and the distribution system operator. Therefore energy storage installations could prove to be a solution, such that the energy does not go to waste. Finding where energy storage devices should be installed could be an additional application of the PQM device and adding a location tag to each of the power quality monitors when installed should

Monitoring Objective	Variables	Sampling rate	Data averaging window	Data Format	Waveforms	Data storage	Instrumentation
Compliance Verification - Regulatory	Generally only voltage disturbances* including: - Steady state RMS voltage - Voltage unbalance - Voltage harmonics - Voltage flicker * Some regulatory standards do call for measurement of current disturbances	Between 3.5 and 6 kHz may suffice	As specified in the appropriate standard or regulation.	Proprietary/ Tabular/ Standardised	Not required	Centralised/ distributed/ hybrid.	IEC61000-4-30 Class A compliant
Compliance Verification – Connections Agreements/Premium Power Contracts	As for Regulatory Compliance Verification plus: - RMS current and harmonic currents - Voltage sags and swells	Between 3.5 and 6 kHz may suffice	As specified in the connection agreement	Proprietary/ tabular/ standardised	Not required	Centralised/ distributed/ hybrid.	IEC61000-4-30 Class A compliant
Performance analysis	Generally voltage disturbances limited to: - Steady state voltage - Voltage unbalance - Voltage harmonics - Voltage flicker - Voltage sags and swells - Highest and lowest rms voltage per 1 (or 10) min window	Between 3.5 and 6 kHz may suffice	Generally 10-minute averaging windows are sufficient 1-min for RMS voltage	Proprietary/ tabular/ standardised	Not recommended	Centralised	IEC61000-4-30 Class A, B or S compliant Other instrumentation such as smart meters or smart relays may also provide useful information
Site characterisation	As for Performance analysis plus transients Current measurements may also be useful for site characterization	It depends upon requested resolution of waveforms to be recorded. 20 kHz may suffice.	Generally 10-minute averaging windows are sufficient. Shorter time intervals may be used depending on the types of load connected	Proprietary/ standardised	May be useful.	Centralised/ distributed/ hybrid	IEC61000-4-30 Class A, B or S compliant
Troubleshooting	Disturbances to be measured will be dependent on the nature of the problem being investigated. Measurement of both voltage and current parameters is often necessary	It depends upon requested resolution of waveforms to be recorded. 500 kHz may suffice for most applications	Time intervals will depend on the nature of the problem being investigated.	Proprietary/ standardised	Recommended	Generally distributed	As appropriate (troubleshooting may involve the use of a range of instrumentation, including instruments other than power quality monitoring devices e.g. oscilloscope, handheld scopemeters)
Advanced applications and studies	Application dependant	Application dependant	Application dependant	Proprietary/ standardised	Application dependant	Centralised/ distributed/ hybrid	Application dependant
Active PQ-management	Few aggregated variables, application dependant	Between 3.5 and 6 kHz may suffice	Very short windows (almost real-time)	Proprietary/ standardised	May be useful	Centralised/ distributed/ hybrid	All available instrumentation that is useful for the task

Fig. 3: Guidelines on power quality monitoring in table form, from paper [7]

not be too difficult. However, if the plugs can be accessed by everyone then it might be necessary to implement some kind of geolocation method, to maintain a correct location in the database.

IV. Method

This project comprises three primary components: conducting research, processing and visualizing data, and, finally, evaluating the current system to identify potential applications and areas for improvement.

In the research phase, information was gathered on what data is needed to qualify PQ-events and how this data can be used to qualify PQ-events, this is already mostly done in section III. Then in the second stage, the current state of the system has to be assessed, emulated and then the processing system can be created. Currently the PQ-monitors send out data to a database, which is connected to a web interface of which a snippet can be seen in Figure 1. This database will first be copied and ran locally, such that the data processing can be done without tampering with the actual live database. This is done to make sure that none of the collected data is lost. An additional benefit to the separate database, is that it is now possible to experiment with simulated data and it is also possible to experiment with other structures of the database, such as extra tables. Then any added functionality to the database and flaws are discussed and suggestions to improve the system in the future are made.

A. The setup

The testing setup was made in a program called Docker [15]. This program allows the user to make a container with images of programs that you want to use. An image contains all necessary information to recreate a program, in this case locally. In the case of this project there are four images, PostgreSQL [16], pgAdmin, Grafana [17] and a Python application. Postgres runs the local database, pgAdmin is a graphical user interface for Postgres that allows accessing and modyfing PostgreSQL databases and Grafana is a tool to make dashboards with functional visualizations of the data in the database. Lastly Python was used to simulate data, since the dashboard is not connected to the actual database used by the device. In addition to that, actual measurements from the grid are less likely to trigger alerts, since the grid in the Netherlands is quite stable. Therefore, PQ-events were simulated, in order to test the triggering of alerts. An overview of the architecture of this setup can be seen in Figure 4, where the Python application acts as a replacement for the measurements coming from the PQM-device. At the time of writing this thesis three prototypes of the PQM-Device were taken to Turkey, in order to perform measurements in a hospital. This will likely result in data for future use.

B. Database

The database was a copy of of the actual postgreSQL database connected to the PQM-devices. This was done by creating a backup of the live database and then restoring it on the database ran in the Docker container. The current

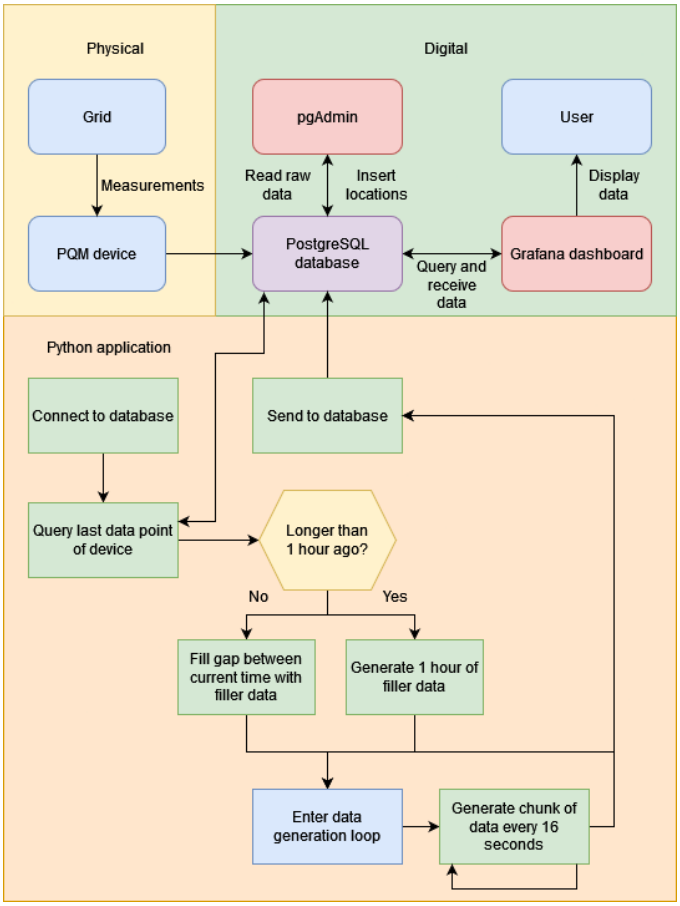

Fig. 4: Architecture of the test setup

database consists of two main tables in which the data is stored. In addition to that there are some tables for permission groups and some for the old interface, which can be seen in Figure 1. The two important tables are $cloud\_device$ and $cloud\_measurement$. The $cloud\_device$ table contains the device names and generates a unique incrementing device id for each new name added to the table. The device name has to be unique and is of the datatype "character varying". The device id is an integer of type bigint. The second table $cloud\_measurement$ also has an id of type bigint, which is a unique incrementing number assigned to each row added to the table. In addition to this id the measurements from the ATM90E26 energy metering integrated circuit and from the humidity sensor [6] accompanied by a timestamp and the corresponding device id are stored in this table.

A proposed change that was made during this project, is the addition of a new table. This table contains deployment data of each device. In this table the device name are stored together with the latitude, longitude and time and date of deployment. This allows the dashboard to select the latest location of each device and then show them on a map, such that the user will have an easy overview of all deployed devices.

C. The dashboard design

The user interface designed consists of one main page, which allows the user to gain a quick overview of all deployed devices, together with some important data and alerts. Another page was created which allows the user to see specific information of a device, such that once the location of the PQ issue is found, that it can be further analysed.

To do this it is important to keep in mind the deployment page, however the device specific page was first created, as this is where all data is gathered and processed.

1) Device page: The device page presents the following data:

- RMS Voltage over time
- RMS Current over time
- Power generated/consumed over time
- Grid frequency

From this data multiple PQ-events need to be derived and made aware to the user: firstly voltage swells and dips, secondly whether or not devices are connected to the PQ-monitor and what class of power consumption it belongs to. Lastly any abnormalities in the grid frequency should be noted.

To achieve this, a graph plotting the last hour of voltage and power data was plotted. In addition to this a histogram of the past frequency can be plotted, to show if there are any abnormalities occurring in the frequency. Lastly PQ-events are indicated through the use of alerts and annotations. Alerts only work on current data, while annotations mark abnormalities in the graphs, such that abnormalities which occurred in the past can also easily be identified. On this page a single device can be selected, such that the user can get a detailed overview of that sensor.

2) Overview page: In the main overview page, the first thing that is shown is a map, on which all deployed devices are be visible. Then this map marks any devices where PQ-events occur with a different colour, such that the user knows from which device the specific data needs to be analysed. In addition to this, the overview page has an alert list, which shows how many PQ-events are occurring and in which device they occur. This was decided, such that a user can quickly navigate to the device page of a PQM-device where a PQ-event is measured. The page should also show how many events occurred in the past couple of hours, ideally this window can be selected on the page, since the user will not monitor all devices at all times and might miss the moment when PQ-events occur.

On the overview page, there should also be some statistics from which the user can easily see if any PQ-events occurred in the past 24 hours. This is useful when the operator has not checked the dashboard recently, such that they are still being informed about the past events or lack thereof. In the absence of recorded events, the user is required to review only the dashboard overview page once a day, a task that takes less than one minute.

If PQ-events did occur, then it is possible for the user to easily find the time period where these occurred and their respective data, such as voltage and power consumption over time.

D. Simulation of data

Data simulation is done using a python script. In the end this script can run continuously, to emulate a connected PQ-device. At first some extreme data was be simulated, in order to test the data processing. This however gave an unrealistic view of what the interface will eventually look like with real data. Therefore, the data was made more realistic later. This is done by analyzing what PQ-events look like, in order to test the interface for a wide variety of events. An additional factor in this research is gathering actual data with the device, such that the emulated data is delivered to the database in the right format.

A couple of key events are emulated:

- Voltage dips and swells
- High, medium and low power draw or production.
- Reactive, apparent and real power
- Device failure or disconnection
- Optional: Deviation from grid target frequency (50Hz)

Frequency monitoring is optional and will only be done if there is time left. Since with the current system the frequency monitoring is not really useful, due to the frequency being stored as an integer.

1) General data emulation: The PQM-device takes a measurement every 2 seconds and these are sent in chunks every 15 seconds. To closely resemble this a data point is made for every 2 seconds. The data is created in chunks of 16 seconds in order to allow for more coherent data, as using purely random values would not resemble true grid data very well. Since the Grafana dashboard queries mostly in time ranges of 1 hour, starting the Python script at some point for 5 minutes, results in a possibly unusable selection of data, since the SQL query then selects 5 minutes of the current time and the 55 minutes of data before that. If this is a long time ago, the timescale in the graph will be extremely large, thus resulting in an unusable representation of data. This can probably be resolved by changing the SQL query, however this was not done at first since the PQ-devices in a real situation, are running continuously. Therefore a startup function was made within the Python application, which makes sure that there is at least 1 hour of data before the current time. The data generated in the startup function is mostly around the expected mean values of most parameters, as generating realistic signatures is easier to do within the smaller chunks, rather than one large chunk of data. Another important feature of the current database is that all values are stored as integers, therefore the columns for voltage, current and power are stored in millivolts, milliamps, milliwatts respectively. Therefore all data is also generated in this order of unit, such that the simulated data is in the same format as the real data.

2) Voltage dips and swells: In order to simulate voltage dips and swells the voltage was emulated first. This was done by generating small chunks of voltage varying slightly with a uniform distribution around a mean value chosen per chunk. Here a base value of 230 Volts is used, after which the base value will be set to the last value of the last chunk. Every time a new chunk is created the mean voltage level will be changed by a random value between -1.5 and +1.5 Volts. This should result in a voltage level fluctuating relatively slowly

and sometimes triggering a voltage alert, ideally approximately once every couple of hours.

3) High, Medium and low power draw or production: Power consumption is related to voltage and current, so it is simulated in relation with the voltage and current. Therefore the current draw was first simulated, there is a chance of it being zero, meaning that there is no device connected in line with the PQM-device. However there is also a chance that a device is connected. This is decided based on a random number generated, but also the previous state. As it is not logical for a device to be connected for only 16 seconds at a time. If the script is in the device connected state it will pick a random current between -10000mA and 10000mA, such that it can also emulate generated power, e.g. from PV-installations connected to the device. It is unlikely for this device to both be connected to a power generating and a power consuming device at the same time, however it would be possible and the grafana dashboard should be able to show either properly. Within grafana the power levels are classified based on their current draw or delivery.

4) Power: From the generated current values, the power draw or generation can be calculated. In order to emulate reactive and real power in addition to apparent power the phase angle was also generated. Apparent power will be calculated by multiplying the RMS Voltage and current, then the phase angle is determined per value of current. To first get an idea of how the device measures reactive, real and apparent power, measurements have been be performed with different loads. Based on this the phase angle was simulated with the python script. The phase angle will mostly correspond to a normal power factor of above 0.9, however there is a small chance that the phase angle is larger, such that the power factor is below 0.9, in that case the dashboard should show a notification.

5) Device failure or disconnection: When a device is disconnected the current will become zero, which sends an alert to the grafana dashboard.

6) Stability of grid frequency: In the current iteration of the PQM-device, the grid frequency is measured and stored as an integer, however in a real grid, the frequency can slightly differ based on the loads or power supplies connected. These small differences could have an effect on grid stability and the life cycle of devices connected to the grid. Therefore it would be useful for research if the frequency is stored as a float instead of an integer. For this more research would have to be done in order to figure out the relation with PQ issues and the stability of the frequency. It should be relatively easy to change the data type to a float, and therefore it will also be simulated as a float.

V. Implementation

A. The setup

At first the setup was made to run locally on a windows laptop, however it was soon found that running it through docker would make it easier to manage. Setting up the docker containers was quite simple and was done using a compose file, which can be found in subsection IX-B. In here the ports of the container images were defined, together with respective usernames and passwords such that the containers can interact with each other.

	id [PK] bigint	device character varying (17)	latitude double precision	longitude double precision	deploy_date timestamp without time zone
1	1	B8:D6:1A:02:BC:19	52.23919775578265	6.856133159931312	2024-09-25 14:10:30
2	2	B8:D6:1A:01:5A:39	52.23879558992885	6.858430425908494	2024-09-25 14:50:30
3	4	B8:D6:1A:01:5A:39	52.23879558992885	6.858430425908494	2024-09-25 15:50:30
4	5	B8:D6:1A:01:5A:39	52.23879558992885	6.858430425908494	2024-09-25 16:05:30

Fig. 5: Screenshot of the new table with deployment data

B. Database

As mentioned in subsection IV-B the database in the docker container is created based of of a backup of the live database. This results in a TAR file, which can then be imported through pgAdmin into the local database. Before it was possible to import this TAR file, a permission group had to be made which is equal to the permission group in the live database, such that the backup could be restored. After this was done a table called $cloud\_device\_location$ was added which contains the deployment data of the devices as discussed in subsection IV-B. The data structure of this table can be seen in Figure 5 and SQL code which generates this table can be found in subsection IX-C.

C. The design

In grafana two dashboards were created and connected to the database, the first dashboard shows data of a specific PQM-device, the other one shows an overview of all selected deployed devices.

D. Device page

The device page gives a detailed overview with graphs, such that relations between different variables can be analysed. An example of this is the current draw and grid voltage, if there is a large voltage drop in the grid, it is most likely due to a device being connected which draws a lot of current. Therefore, these graphs are below each other. In addition to this there is also a graph showing apparent, reactive and real power. These graphs were created with similar SQL queries all selecting data based on the device selected in the dropdown menu at the top of the page. This page also creates alerts based on the last data in the database. This is done to indicate voltage dips, swells, but also large power consumption or generation. The queries used on this page can be found under subsection IX-D.

1) Voltage monitoring: The first graph that is implemented is the voltage graph. By default this graph shows the last 1 hour of data of the device selected in the dropdown menu, however other time frames can also be selected with Grafana. In order to make it easier for the user to interpret the voltage a couple of voltage levels were defined. These are normal, low, high, too low or too high. Normal is defined as a voltage between 220 volts and 240 volts, this was chosen because the grid is made to have a voltage of 230 volts RMS and the voltage may fluctuate slightly, without any negative consequences. Low is between

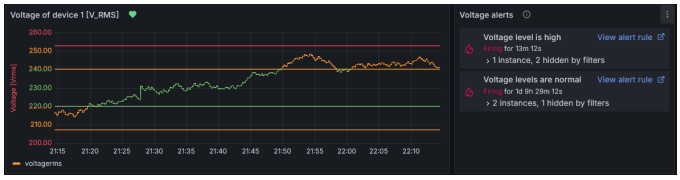

Fig. 6: High voltage alert in the Grafana [17] dashboard

Value [A]	Label
I <0	Power being delivered to grid
I = 0	No current draw
0 <I <1	Low power draw
1 ≤ I <5	Medium power draw
I ≥ 5	High power draw

TABLE II: Current value to annotation label mapping

207 and 220 volts and high is between 240 and 253 volts, these ranges are getting close to the definition of a voltage dip or swell, namely at a 10% offset. If the voltage is below 207 volts, then the voltage drop can be classified as a voltage dip, this is considered too low, since it dropped 10% below the level it is supposed to be. If the voltage is above 253 volts, then it can be classified as a voltage swell, since the voltage is 10% above the level that it is supposed to be. For these levels alerts are created, the *voltage is high* alert is shown in Figure 6. The alert below it, is for the other devices, for which the voltage level is currently normal. Since they are not receiving any data, they remain constant.

2) Current monitoring: The current graph does not have levels as the voltage graph, since there are no ”correct” or ”wrong” current levels defined. Therefore, the current graph shows annotations when a device is connected or disconnected. Annotations are dashed lines and when hovering above them a description of the annotation is shown. The annotations and their conditions can be seen in Table II

In addition to the annotations there are also alerts, showing the current state of the current level measured by the device. An example of this alert can be seen in Figure 7, where the alert which is happening since 30 seconds corresponds to the graph shown, the other alerts are from other devices, which are currently inactive.

3) Power monitoring: The power graph shows 5 different parameters, which are: the power factor, the phase angle and the real, reactive and apparent power. The power factor shows how efficient the apparent power is being utilized, this

Fig. 7: A screenshot of the current graph, its annotations and alerts in Grafana

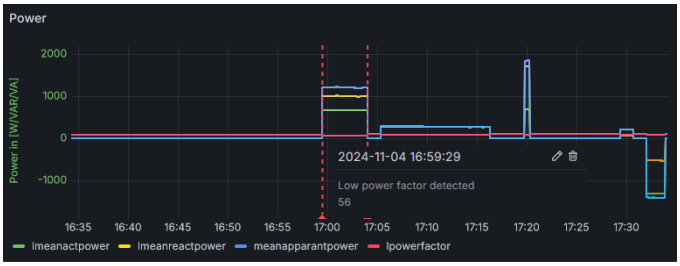

Fig. 8: A screenshot of the power graph in Grafana, with an annotation

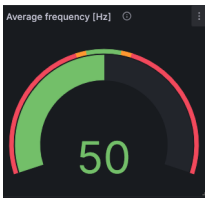

Fig. 9: Average frequency shown in Grafana gauge

parameter is also alerted on if it is below 0.8, since this results in larger energy losses [18]. It is difficult to determine how severe this low power factor is, since it depends on the draw of the power and how resilient the grid is to voltage drops and swells. However the EU has regulations which state that the power supplies of desktop computers should have a power factor of at least 0.9 at full load [19]. An example of how this power monitoring looks can be seen in Figure 8, where the annotation was moved, such that it can be seen in the same figure. The power factor is also given as a percentage, since the database can only store integers as of right now.

4) Frequency monitoring: The device page also shows a frequency panel, this states the average frequency of the past hour. This panel was supposed to be replaced by a histogram, however in the current state of the device, the frequency is stored as an integer, so only fluctuates between 49 and 50 Hertz. Further research should be done on a proper implementation of frequency monitoring. If the frequency would be stored as a float, displaying the frequency in a graph would most likely be the most insightful, since its data can then be correlated with the other data. As of now the frequency is not simulated to be any other value then 50Hz, therefore there is only an average frequency of the past hour shown with a gauge, which always displays 50Hz, this can be seen in Figure 9.

E. Overview page

In the top of the overview page a dropdown menu can be used to select which devices should be shown on the map. Next to the map there is a section where alerts are shown about all deployed devices. Below the map there is a table with all deployed devices, where a device can be clicked to view the details of that device. In section IV it was mentioned that a counter or a list of all alerts of the past 24 hours should be

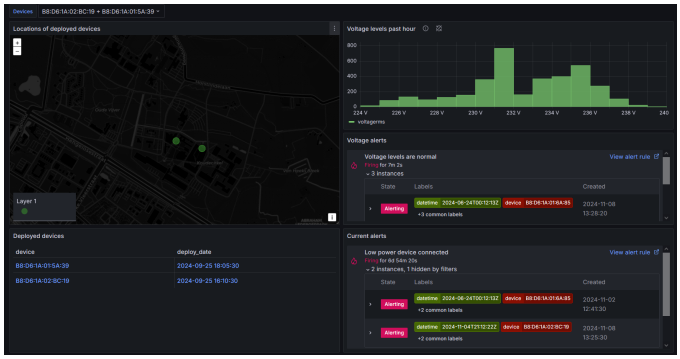

Fig. 10: A screenshot of the overview page in the Grafana dashboard, also larger in Figure 14

added. This has unfortunately not happened. There is no simple way to achieve this in Grafana and due to time constraints more effort was put into the device page. The alert list does show all current voltage and current alerts. The dropdown menu can be used to select which devices need to be included on the map and which devices their voltage needs to be put into the voltage histogram. The histogram takes the last hour of data from all of the selected devices and plots them, this way it is easy to see how stable or unstable the voltage has been in the past hour for all devices. A screenshot of this page can be seen in Figure 10. Currently the color of the blip does not change based on its current voltage, however this would be a nice addition inspired by Smolenski *et al.* [10]. The queries used in the overview page can be found in subsection IX-E.

F. Simulation of data

In order to simulate measurements taken by the PQM-device, the current state of the device was assessed. As mentioned in section II and subsection IV-D, the device takes a measurement every 2 seconds and are sent in chunks every 15 seconds. Since 15 seconds is not divisible by 2 and it is not efficient to create a queue of rows and then send as many as possible every 15 seconds in a system where the connection is always stable, the chunks are sent every 16 seconds instead. The connection in the simulation is always stable, because both the Python application and the database run on the same system. As mentioned above, the simulation of data is done in a Python script, of which the functionality is described in de following subsections. The data simulation script can be found in subsection IX-F.

1) Setup: In order to generate rows and send them to the database, a couple of libraries were used. The first library used is NumPy [20]. This library was used to create and alter arrays and to generate random and uniform distributed data. The second library used is psycopg2 [21]. This library allows a Python script to interact with a PostgreSQL database. Furthermore, three Python modules had to be imported, these are time [22], math [23] and datetime [24]. The time module is used for the sleep function, such that the time between data chunks does not load the system it is running on unneccesarily. The math module is used to perform certain calculations and to round values. The datetime module is used to get, convert and calculate timestamps.

After the important components required to make the code function are imported the functions are defined, which are discussed in their corresponding sections. And some variables can be set. The variables that need to be set are: the device id, the simulation time, the base voltage, the voltage deviation, the amount of rows in one data chunk and the information to connect to the database. In addition to this two global variables are declared, namely the last voltage and the last current. This is done such that the functions can use the data from the previous chunks to create more realistic looking data.

The device id simply has to be set to the unique incrementing integer corresponding to the device you want to generate data for. This id can be found in the $cloud\_device$ table for its corresponding device name.

The simulation time defines for how many minutes the Python application should generate data.

The base voltage defines the average voltage, for which you want to generate data, this can be useful if you want to simulate different kind of grids, in this case it was set to 230 Volts, since this is normal for most European grids, however in the US the grid voltage is around 120 Volts [25].

The voltage deviation is the maximum positive or negative deviation that can happen within the voltage of a chunk, which is generated with a uniform distribution around a value generated per chunk as is discussed in subsubsection V-F3.

Lastly the information required to connect to the database is stored in a $psycopg2.connect$ object. This information includes: the database its hostname, its port, the username, the password and the database name.

2) Startup cycle: The startup cycle of the Python application is ran every time that the application is started. This function first checks what the last datapoint in the database is, for the device id for which data is being generated. If this datapoint is further than one hour in the past compared to the current time, the function creates data from 1 hour in the past to the current time and then start generating the data chunks as normal. If the last datapoint is less than one hour in the past, then the script simply creates datapoints between the timestamp of the last datapoint and the current time. To achieve this the startup function uses the same functions as the main loop, however it uploads the data in one large chunk, instead of uploading a small chunk every 16 seconds. As mentioned, this results in it just varying the voltage uniformly around 230 Volts with the specified voltage deviation.

3) Voltage simulation: Simulating the voltage was done as specified in subsection IV-D. Normally voltage levels in a grid depend on a lot of factors, such as loads connected, but also on the current power of the sun if PV-installations are connected, or wind strength if there are windmills connected. However in this case to keep the simulation relatively simple, no external factors were modeled. Since this simulation of data functions as a way to test a dashboard and its accompanying features, with which insight can be gained in the power quality of the network.

The voltage simulation function has a couple of steps, first

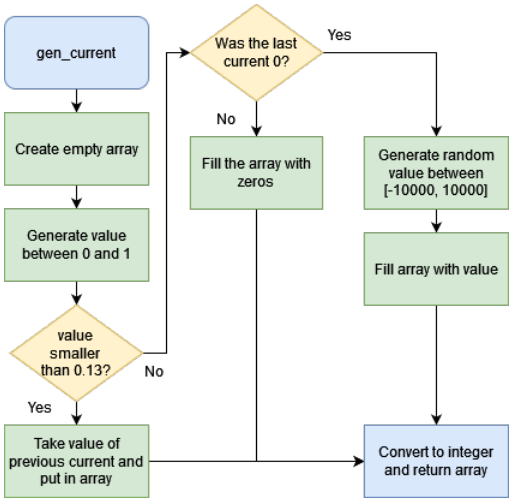

Fig. 11: Flowchart of the *current_gen* function

it checks the last voltage entered into the database. If this is within 205 and 255 volts, a random value between -1500mV and 1500mV is picked, which is then added to the last value entered, the sum of this then becomes the new base value. The voltage generation function then creates an array with the length of one chunk which contains a uniform distribution around the new base voltage with a deviation of plus or minus the voltage deviation specified earlier. If the voltage is outside of this range, so smaller than 205 volts or larger than 255 volts, then the random value added to create the new base value is biased to make sure the voltage value does not diverge too much from regular values. So in the case of the last voltage being smaller than 205 volts the random value is picked between -500mV and 1800mV and if the last voltage is larger than 255 volts the random value is picked between -1800mV and 500mV. These values were determined based on trial and error, using these values the voltage levels were simulated according to the desired specifications in subsection IV-D.

4) Current simulation: The current simulation is done in a similar data type as the voltage, namely an array with the length of a chunk size. What is different however is that the voltage level constantly should have a value, often around 230 volts, while the current level depends on whether a device is connected to the PQM-device. Therefore, the current generation function is also based on chance, however in this case there is a small chance per chunk that the value has to switch from 0 amperes to a positive or negative randomly generated value or the other way around. The larger chance is that the value of the current remains the same. A flowchart of this function can be seen in Figure 11, where the value generated is a random value between 0 and 1 and the small chance is 13%, therefore it is checked if the value is smaller than 0.13.

5) Power: Real measurements were taken of a vacuum cleaner being connected in line with the PQM-device to the grid, this data was then visualized in the Grafana dashboard, but also inspected in pgAdmin in order analyze what parameters are recorded and how they relate to each other. From the data that can be seen in Figure 13. From this the conclusion was drawn that the voltage and current multiplied approximately equal the mean apparent power. To verify this the average absolute error in this assumption was calculated using the 243 rows of data from this measurement which contained power values, this is equal to 7.8 minutes of data. This calculation was done using an excel sheet made by exporting the results of the SQL query showing the right time frame. This resulted in an average error of approximately 121 millivolts.

In order to generate data for the three different types of power that are measured: real, reactive and apparent power. First the apparent power is calculated based on the voltage and current, simply by multiplying the arrays of voltages and currents with each other and then the phase angle for the device is generated in order to calculate the real and reactive power. This is done such that all power data generated obeys the laws of physics and therefore should give an accurate representation of real data. For each load with a different constant current draw, there is a corresponding relatively constant phase angle. Therefore, a new phase angle is generated every time the current changes. The phase angle further depends on the impedance of the load, which depends on the type of device connected. This is however not simulated, so the phase angle only depends on the current, while keeping a relatively normal power factor. This was the method discussed in subsection IV-D, however for power quality monitoring it is more interesting to detect whether a device has a bad power factor as opposed to a certain phase angle. Therefore, the phase angle was determined from the power factor and not the other way around. In this case it is simulated that there is a 20% chance at a device with a bad power factor, between 0.4 and 0.8 and a 80% chance at a device with a power factor between 0.9 and 0.98. After which the phase angle can be calculated using Equation 1 from Cadence [26].

$$\begin{aligned} PF &= \cos\varphi = \frac{Re[S]}{|S|} \\ \varphi &= \arccos(PF) \end{aligned} \tag{1}$$

Then using the power triangle [27], which is a right angled triangle which shows the relation between apparent, real and reactive power, the equations for the corresponding real and reactive power can be derived. These can be found in Equation 2.

$$\begin{aligned} P_{real} &= P = S\cos(\varphi)\ [W] \\ P_{reactive} &= Q = S\sin(\varphi)\ [VAR] \end{aligned} \tag{2}$$

Now the reactive and real power can be calculated as well. After which the reactive, real and apparent power are returned to the main function together with the phase angle and the power factor. A flowchart of this function can be seen in Figure 12

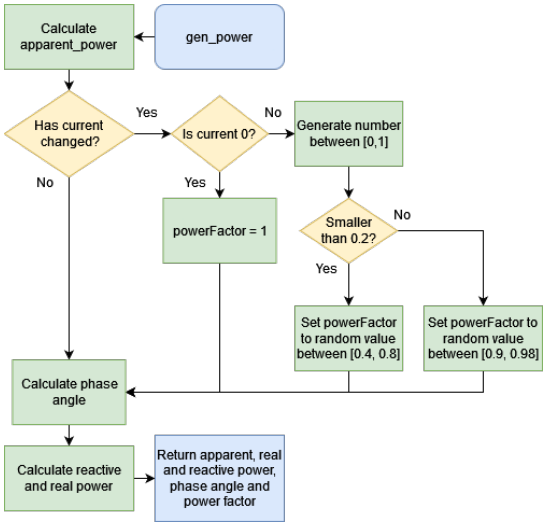

Fig. 12: Flowchart of the *gen_power* function

Fig. 13: Screenshot of the dashboard showing real data

VI. Usage

In order to evaluate how the implementation of the dashboard works, a segment of data measured by the PQM-device was analysed using the Grafana dashboard. The data recorded is of a vacuum cleaner being used, then turned to a different setting for a short period of time and then returned to the original setting and turned off.

A screenshot of the dashboard is shown in Figure 13. In this screenshot multiple grid parameters are shown in graph form, these are: RMS voltage, RMS current, mean apparent power, mean actual power, mean reactive power and the phase angle. As an operator of a small scale grid these measurements can be useful to monitor grid performance under heavy load. Note that the phase angle is plotted here opposed to the normally plotted power factor. This is done, because there was no power factor data for these measurements, or there was data, but every entry was zero. The cause of this and a possible solution are discussed in section VII.

Starting at the top graph we can see that the voltage always remains within the orange lines, which means that the voltage is within normal limits. If it were to reach beyond the orange lines the voltage level would be moving towards being classified as a dip or swell. However, outside of the red lines the voltage level would classify as a dip or swell. When the load is connected, in this case the vacuum cleaner being turned on, a drop in voltage can be seen, at the same time the current spikes shortly and then stabilizes at an almost constant level. The red dashed line in the current graph shows that a device is being connected, by hovering above this marker, one can also see at what power class it is classified. This shows that the vacuum cleaner falls into the high power device class at first, however after stabilizing, it is in the medium power class. With this classification devices using less than approximately 1150 Watts fall under the medium power devices. This means that most household devices, are within the low and medium power class, however kitchen appliances, such as electric kettles and ovens, will likely fall under the high power class. We can also look at the power graph, here the phase angle is very low and stable at first, which results in a high power factor, which means that almost all power is used effectively. Later however, the phase angle rises and the power draw decreases, this happened because the vacuum cleaner was turned to a lower setting. This most likely happened because the load on the motor was reduced, as said by Kamran:"When the load on the motor is small, its power factor reduces to 0.2–0.3 and it increases to 0.8–0.9 at full load." [28]. This of course depends on the type of motor, however the general message can be applied to any motor. From the graph it can be seen that the power factor drops significantly. While in this case the rise in reactive power was not significant enough to cause any issues, in smaller and less stable grids, a device with a low power factor could result in a large generation of reactive power, thus increasing the grid voltage. It is also visible that even though the current draw is only reduced by around 25%, the voltage of the grid is almost back at its normal level.

VII. Discussion

A dashboard was designed for the PQM-device designed by Grootjans and Moonen [6]. In addition to this a full system was setup to mimic the current PQM-device and the corresponding database. This was done such that changes could be made to the database, without tampering with real data. This dashboard

visualizes the data from the simulated PQM-device in such a way, that it is possible for users with little to no expertise in the PQ field to monitor their own grid locally. In addition to this, it is also possible to monitor multiple devices at once using the overview page, such that this system is also interesting for operators of microgrids. Before the Grafana dashboard is deployed in the real world, some configurations still need to be changed, such as the logging of alerts, which should be quite easy with Grafana, since it can mail alerts to an email address. With the dashboard a user can derive if a large load should or should not be connected to the grid, by looking at the voltage graphs, or checking if there are any alerts indicating problems with the grid. In addition to that the queries in Grafana, sometimes take quite a long time to return all the data, this is partially caused due to limited resources available to the programs through docker. However the queries are also not optimized to the fullest extent yet, since some optimization ideas are difficult to achieve in combination with Grafana and PostgreSQL. The simulation of data was done with a Python script, in order to achieve this simulation the PQM-device its output was analysed first. After which the simulation of data was done with some aspects depending on chance, however most aspects are calculated using proven theories, therefore resulting in quite realistic data. However, the way that the voltage rises or falls is not quite realistic as it does not depend on the load connected, but rather just climbs and falls randomly. The other thing which is done randomly is the connection and disconnection of a device, this is hard to avoid, however could be improved by creating a couple of predefined signatures of devices being plugged in and disconnected. For example based on the work by Kamyshev *et al.* [8]. Linking the voltage level to the current and power data would be a great improvement and could turn this relatively simple Python script into an interesting model for testing the effect of certain devices on a specific grid. This could be done by implementing a weight factor based on the total size and stability of the grid. To implement this a grid would first have to be analyzed thoroughly, however this could be done in future work. The dashboard works quite well, however could be improved by adding some extra functionality, namely frequency monitoring and a more advanced deployment page. The deployment page could be improved by making alerts visible in the map, or getting specific PQ information by hovering over a device on the map. In addition to this if alerts could be customized in Grafana this would also allow for more readable and compact alerting, therefore increasing the amount of alerts that can be shown, such as recent alerts, instead of only current alerts and also just showing the necessary data. Frequency monitoring is becoming more relevant due to the increasing usage of renewable sources in the grid, which results in less frequency stability [4]. However, to implement this the frequency monitoring of the device needs to be updated, right now it is stored in the database as an int, however it should be stored as a float, such that deviations below 1Hz can also be observed. Another variable that needs to be changed is the power factor, right now it does not seem store anything in the database, which could be due to it being a float stored as an int and therefore always truncating the number to 0, since power factor is between 0 and 1. The last improvement that can be made to the database is adding a table which contains data on the location of the device. This was done in the test database and allows a user to get a quick overview of all deployed devices on a map. This is useful if there are multiple devices deployed throughout a small town and the location of a PQ problem needs to be found, such that the cause can be investigated. Currently it is not possible in the dashboard to select a larger amount of time than 1 hour at once. This is because the time window selected cannot be used in PostgreSQL queries. It may be possible with a workaround, however it is currently not supported by Grafana by default. A possible solution for this could be switching from PostgreSQL [16] to Prometheus [29], which has a database structure based on time series data, which is exactly what the PQM-device uses, it also allows for easy alerting and monitoring. Querying data should also be more efficient with Prometheus resulting in a more responsive dashboard. Another feature that could be implemented in the future is analyzing data on a daily basis to check if there are any recurring patterns, such as voltage level changes at a recurring time, which could be caused by high loads being disconnected at the end of a workday. This way certain PQ-issues could be prevented, by either connecting or disconnecting a device at or before that time. Lastly local energy storage devices could also be connected based on the data from the PQM-device, such as suggested by Smolenski *et al.* [10].

VIII. Conclusion

First of all the importance of power quality monitoring was analyzed and discussed, after which a PQM-device by Grootjans and Moonen [6] was introduced. This device was created to make in-depth and distributed power quality monitoring more accessible and affordable. The output of this device was then analyzed in order to get an idea of how to visualize this data in a more accessible way. It was then decided that this visualization should be done in a dashboard type of environment, ideally accessible from any browser on any device. In order to assist in developing this dashboard data was required, at first a copy was made of real data, however this data did not contain specific events, which were interesting to monitor or act upon with alerting features. Therefore, research was done into power quality in general and power quality monitoring. In order to both work on the simulation of data and creating the dashboard within a relatively small period of time, an off-the-shelf data monitoring platform, Grafana [17] was used. In the end a significant improvement to the currently existing PQM-device its ecosystem was made, by developing a tool in Grafana, which can graphically visualize the data. The data can easily be selected based on a time range and a drop-down menu to select a device. In addition to this a second page, where an overview of all deployed devices can be seen, was created. This page shows current alerts and in which device it occurs, such that the user can take a closer look at the data of that device. When taking a look at the data of a specific device in a specific time range, the user is assisted by some features, which create alerts of current occurrences

and annotations in the graphs containing the data to classify events. In addition to this a first step has been made towards creating a mathematically correct model for simulating PQ-events. Unfortunately linking the voltage to the current and power draw was not done, however this would greatly improve the simulation system. In addition to this realistic current signatures inspired by the data from Kamyshev *et al.* [8] could be made for devices and then inserted in this program to model certain grid responses. Furthermore, certain suggestions have been made to improve the functionality of the current system its database, such as changing the frequency and power factor data to floats and adding a table to contain information about the deployment of devices.

Ultimately this project first shows the importance of power quality monitoring, analyzes ways in which it is currently done and identifies problems in current power quality monitoring. Then introduces a device, which could make power quality monitoring more accessible and then discusses the design of a grid emulating Python application in order to assist in the creation of a graphical user interface for this device. The simulation and the dashboard are both not currently a finished product, however do show a potential final form of the PQM-device [6] and the systems surrounding it.

REFERENCES

[1] B. D. B. Uchenna Godswill Onu, Antonio Carlos Zambroni de Souza, "Drivers of microgrid projects in developed and developing economies," *Utilities Policy*, vol. 80, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957178722001515#bib58

[2] Ghazanfar Shahgholian, "A brief review on microgrids: Operation, applications, modeling, and control," *International Transactions on Electrical Energy Systems*, 2021. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1002/2050-7038.12885

[3] T. R. S. Saha, M.I. Saleem, "Impact of high penetration of renewable energy sources on grid frequency behaviour," *International Journal of Electrical Power Energy Systems*, vol. 145, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0142061522006974

[4] G. Y. Kamala Sarojini Ratnam, K. Palanisamy, "Future low-inertia power systems: Requirements, issues, and solutions - a review," *Renewable and Sustainable Energy Reviews*, vol. 124, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1364032120300691?via%3Dihub

[5] Frank Leferink, "Conducted interference, challenges and interference cases," *IEEE Electromagnetic Compatibility Magazine*, vol. 4, pp. 78–85, Quarter 1 2015.

[6] Roelof Groojans, Niek Moonen, "Design of cost-effective power quality and emi sensor for multinode network," *LETTERS ON ELECTROMAGNETIC COMPATIBILITY PRACTICE AND APPLICATIONS*, vol. 5, no. 4, pp. 131–136, December 2023.

[7] Jako Kilter, Jan Meyer, Sean Elphick, Jovica V. Milanović, "Guidelines for power quality monitoring – results from cigre/cired jwg c4.112," *International Conference on Harmonics and Quality of Power*, vol. 16, pp. 703–707, 2014.

[8] Ilia Kamyshev, Sahar Moghimian Hoosh, Henni Ouerdane, "Physics-informed appliance signatures generator for energy disaggregation," *IEEE the 7th Conference on Energy Internet and Energy System Integration*, pp. 3591–3596, 2023.

[9] T. de Graaf, "Power quality monitoring and data collection," 2024.

[10] Robert Smolenski, Pawel Szczesniak, Wojciech Drozdz, Lukasz Kasperski, "Advanced metering infrastructure and energy storage for location and mitigation of power quality disturbances in the utility grid with high penetration of renewables," *Renewable and sustainable energy reviews*, 2022.

[11] "Dutch prosumers must pay fees to feed surplus electricity to grid," *Balkan Green Energy News*, 2024, accessed: 2024-10-28. [Online]. Available: https://balkangreenenergynews.com/dutch-prosumers-must-pay-fees-to-feed-surplus-electricity-to-grid/

[12] The council of the European communities, "Council directive 89/336/eec of 3 may 1989 on the approximation of the laws of the member states relating to electromagnetic compatibility," *L 139*, pp. 19–26, 1989.

[13] J. Zico Kolter, Matthew J. Johnson, "Redd: A public data set for energy disaggregation research," 2011. [Online]. Available: https://people.csail.mit.edu/mattjj/papers/kddsust2011.pdf

[14] Jack Kelly, William Knottenbelt, "The uk-dale dataset, domestic appliance-level electricity demand and whole-house demand from five uk homes," 2015. [Online]. Available: https://arxiv.org/pdf/1404.0284v3

[15] Docker Inc., "Docker: Empowering app development for developers," https://www.docker.com, accessed: 2024-10-28.

[16] PostgreSQL Global Development Group, "The worlds most advanced open source database," https://www.postgresql.org/, accessed: 2024-11-04.

[17] Grafana Labs, "The open observability platform," https://grafana.com/, accessed: 2024-11-04.

[18] J. Ware, "Power factor correction," 2006. [Online]. Available: https://electrical.theiet.org/media/1687/power-factor-correction-pfc.pdf

[19] E. Union, "implementing directive 2009/125/ec of the european parliament and of the council with regard to ecodesign requirements for computers and computer servers," accessed: 2024-11-03. [Online]. Available: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=celex:32013R0617

[20] N. Team, "Numpy: The fundamental package for scientific computing with python," accessed: 2024-11-01. [Online]. Available: https://numpy.org/

[21] J. E. Daniele Varrazzo, "psycopg2 - python-postgresql database adapter," accessed: 2024-11-01. [Online]. Available: https://pypi.org/project/psycopg2/

[22] Python Software Foundation, "time — time access and conversions," accessed: 2024-11-01. [Online]. Available: https://docs.python.org/3/library/time.html

[23] ——, "math — mathematical functions," accessed: 2024-11-01. [Online]. Available: https://docs.python.org/3/library/math.html

[24] ——, "datetime — basic date and time types," accessed: 2024-11-01. [Online]. Available: https://docs.python.org/3/library/datetime.html

[25] O. of Electricity, "Electricity 101," accessed: 2024-11-01. [Online]. Available: https://www.energy.gov/oe/electricity-101

[26] C. P. Solutions, "Use the phase angle formula to understand power delivery," accessed: 2024-11-01. [Online]. Available: https://resources.pcb.cadence.com/blog/2020-use-the-phase-angle-formula-to-understand-power-delivery

[27] E. Tutorials, "Power triangle and power factor," accessed: 2024-11-03. [Online]. Available: https://www.electronics-tutorials.ws/accircuits/power-triangle.html

[28] M. Kamran, "Chapter 3 - power grids," in *Fundamentals of Smart Grid Systems*, M. Kamran, Ed. Academic Press, 2023, pp. 71–131. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780323995603000053

[29] Prometheus team, "Monitoring system and time-series database," https://prometheus.io/, accessed: 2024-11-04.

IX. Appendix

A. *Enlarged figures*

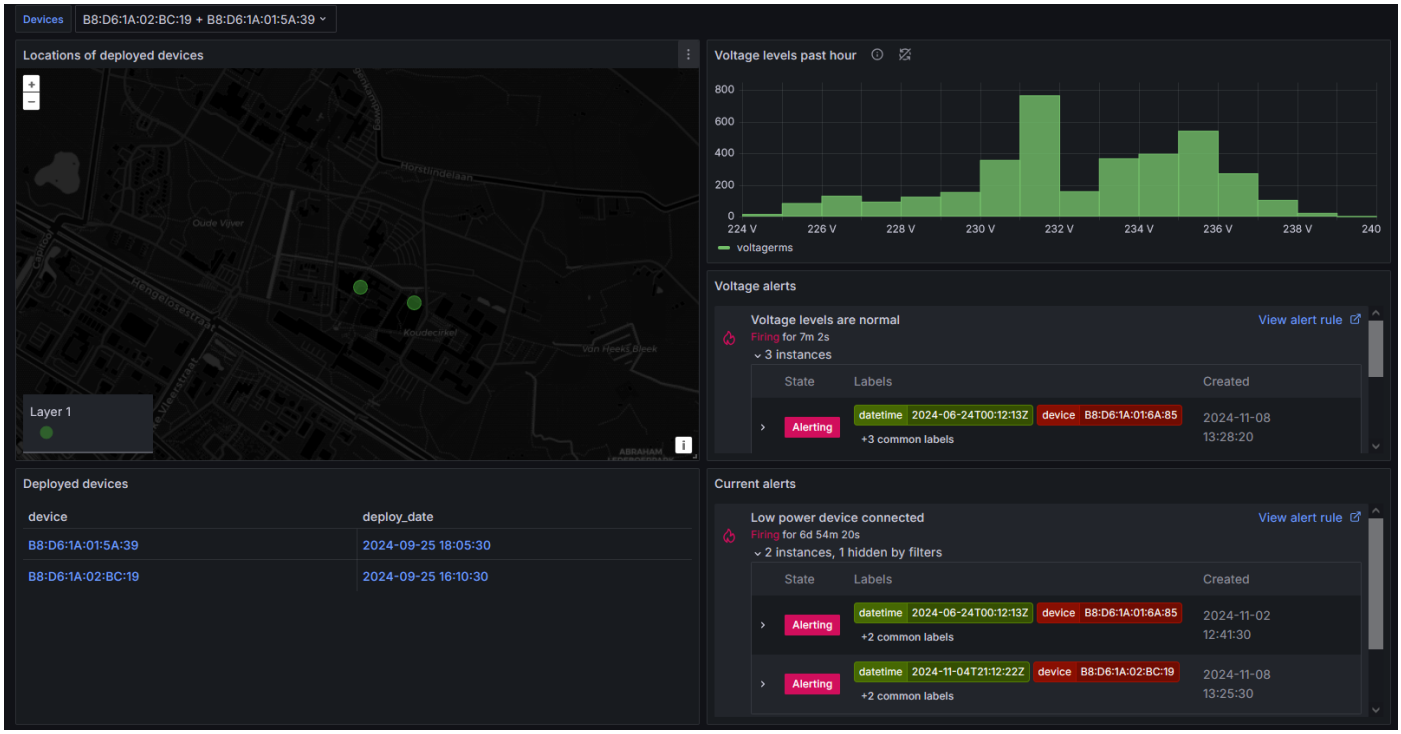

Fig. 14: Enlarged version of Figure 10

B. Docker compose file

```
services:
  db-new:
    image: postgres
    restart: always
    environment:
      POSTGRES_PASSWORD: ****
      POSTGRES_USER: postgres
      POSTGRES_DB: local_PQ
    volumes:
      - pgdata:/var/lib/postgresql/data
    ports:
      - 5432:5432
  pgadmin4:
    container_name: pgadmin4
    image: dpage/pgadmin4:latest
    restart: always
    ports:
     - 80:80
    environment:
         - PGADMIN_DEFAULT_EMAIL=default@default.com
         - PGADMIN_DEFAULT_PASSWORD=default
  grafana:
    container_name: grafana
    image: grafana/grafana-enterprise
    restart: unless-stopped
    user: '0'
    ports:
      - 3000:3000
    volumes:
      - grafana:/var/lib/grafana
  python_app:
    build:
      context: .
      dockerfile: fake_data/docker/dockerfile
    volumes:
      - ./fake_data/docker:/usr/app/src
    ports:
      - "2000:2000"
volumes:
  pgdata:
  grafana:
  python_app:
```

Listing 1: Docker compose file

C. SQL code of the new suggested *cloud_device_location* table

```
CREATE TABLE IF NOT EXISTS public.cloud_device_location
(
    id bigint NOT NULL GENERATED BY DEFAULT AS IDENTITY ( INCREMENT 1 START 1 MINVALUE 1 MAXVALUE
    9223372036854775807 CACHE 1 ),
    device character varying(17) COLLATE pg_catalog."default" NOT NULL,
    latitude double precision NOT NULL,
    longitude double precision NOT NULL,
    deploy_date timestamp without time zone NOT NULL,
    CONSTRAINT cloud_device_location_pkey PRIMARY KEY (id)
)
```

Listing 2: SQL code of the *cloud_device_location* table

D. Grafana device page postgreSQL queries

```
SELECT (voltagerms::float / 1000) AS voltagerms, cast(datetime as timestamp)-INTERVAL '1 hour'
FROM cloud_measurement
WHERE device_id= (SELECT id FROM cloud_device WHERE device = '$Device_name' )
AND datetime < '2050-06-23T20:55:33Z'
ORDER BY datetime DESC
LIMIT 1800
```

Listing 3: Voltage selection query for voltage graph

```
WITH last_hour AS
(
  SELECT DISTINCT ON (device_id) device_id, (voltagerms::float / 1000) AS voltagerms, datetime FROM
    cloud_measurement
  WHERE datetime < '2050-06-23T20:55:33Z'
  ORDER BY device_id DESC
  LIMIT 1800
)

SELECT cast(d.device AS varchar), voltagerms, datetime
FROM last_hour data
JOIN cloud_device d ON data.device_id = d.id
WHERE data.device_id = d.id;
```

Listing 4: Voltage alert query. Using threshold logic in Grafana, this query was used for all voltage related alerts.

```
SELECT (lcurrentrms::float/1000) AS currentrms, cast(datetime as timestamp)-INTERVAL '1 hour' FROM
    cloud_measurement
WHERE device_id = (SELECT id FROM cloud_device WHERE device = '$Device_name' ) AND datetime < '2050-06-23T20
    :55:33Z'
ORDER BY datetime DESC
LIMIT 1800
```

Listing 5: Current selection query

```
WITH last_hour AS
(
  SELECT DISTINCT ON (device_id) device_id, (lcurrentrms::float / 1000) AS lcurrentrms, datetime FROM
    cloud_measurement
  WHERE datetime < '2050-06-23T20:55:33Z'
  ORDER BY device_id DESC
  LIMIT 1800
)

SELECT cast(d.device AS varchar), lcurrentrms, datetime
FROM last_hour data
JOIN cloud_device d ON data.device_id = d.id
WHERE data.device_id = d.id;
```

Listing 6: Current alert query, Grafana thresholds were used to trigger the alert with the right name

```
WITH RankedEntries AS (
    SELECT lcurrentrms, cast(datetime as timestamp)-INTERVAL '1 hour' AS datetime,
           ROW_NUMBER() OVER (PARTITION BY device_id ORDER BY datetime DESC) AS row_num
    FROM cloud_measurement
    WHERE datetime < '2050-06-23T20:55:33Z' AND device_id= (SELECT id FROM cloud_device WHERE device = '
    $Device_name' )
),
last_hour AS
(
  SELECT lcurrentrms, cast(datetime as timestamp)
  FROM RankedEntries
  ORDER BY datetime DESC
  LIMIT 1800
),
state_text AS
(
SELECT lcurrentrms, datetime,
CASE
  WHEN lcurrentrms BETWEEN '1' AND '1000' THEN 'Low current device connected'
  WHEN lcurrentrms BETWEEN '1000' AND '5000' THEN 'Medium current device connected'
  WHEN lcurrentrms > '5000' THEN 'High power draw'
  WHEN lcurrentrms < '0' THEN 'Power being delivered to grid'
  ELSE 'No current draw'
  END AS state_text
FROM last_hour
),
difference AS
(
SELECT lcurrentrms, datetime, state_text, LAG(state_text, 1)
OVER (
    ORDER BY datetime ASC
) prev_state_text
FROM state_text
)
SELECT lcurrentrms, datetime, state_text
FROM difference
WHERE state_text != prev_state_text
```

Listing 7: Current annotation query

```
SELECT lmeanactpower::float/1000 AS lmeanactpower,
lmeanreactpower::float/1000 AS lmeanreactpower,
meanapparantpower::float/1000 AS meanapparantpower,
lpowerfactor,
cast(datetime as timestamp)-INTERVAL '1 hour'
FROM cloud_measurement
WHERE device_id= (SELECT id FROM cloud_device WHERE device = '$Device_name' ) AND datetime < '2050-06-23T20
    :55:33Z'
ORDER BY datetime DESC
LIMIT 1800
```

Listing 8: Power selection query

```
WITH RankedEntries AS (
    SELECT lpowerfactor, cast(datetime as timestamp)-INTERVAL '1 hour' AS datetime,
           ROW_NUMBER() OVER (PARTITION BY device_id ORDER BY datetime DESC) AS row_num
    FROM cloud_measurement
    WHERE datetime < '2050-06-23T20:55:33Z' AND device_id= (SELECT id FROM cloud_device WHERE device = '
    $Device_name' )
),
last_hour AS
(
  SELECT lpowerfactor, cast(datetime as timestamp)
  FROM RankedEntries
  ORDER BY datetime DESC
  LIMIT 1800
),
state_text AS
(
SELECT lpowerfactor, datetime,
CASE
  WHEN lpowerfactor < '80' THEN 'Low power factor detected'
  ELSE 'High power factor detected'
  END AS state_text
FROM last_hour
),
difference AS
(
SELECT lpowerfactor, datetime, state_text, LAG(state_text, 1)
OVER (
    ORDER BY datetime ASC
) prev_state_text
FROM state_text
)
SELECT lpowerfactor, datetime, state_text
FROM difference
WHERE state_text != prev_state_text
```

Listing 9: Power factor annotation

E. Grafana Overview page queries

```
SELECT device, latitude, longitude, MAX(deploy_date) as deploy_date FROM cloud_device_location
WHERE device IN ($Devices)  GROUP BY device, latitude, longitude
```

Listing 10: Location selection query

```
WITH NumberedRows AS (
    SELECT
        voltagerms::float / 1000 AS voltagerms,
        device_id,
        datetime,
        ROW_NUMBER() OVER (PARTITION BY device_id ORDER BY datetime DESC) AS row_num
    FROM
        cloud_measurement
    WHERE
        device_id IN (SELECT id FROM cloud_device WHERE device IN ( $Devices ))
        AND datetime < '2050-06-23T20:55:33Z'
)
SELECT
    datetime,
    voltagerms
FROM
    NumberedRows
WHERE
    row_num <= 1800
ORDER BY
    datetime DESC;
```

Listing 11: Voltage histogram selection query

F. Python script

```
import numpy as np
import psycopg2
import time
import math
from datetime import datetime, timedelta

#Function which generates the current values to be inserted
def gen_current(num_rows, last_current):
    fake_current = np.array([])
    if (np.random.rand(1,1) < 0.1):
        if last_current != 0:
            fake_current = np.zeros(num_rows, dtype=int)
        else:
            current_level = np.random.randint(low=-10000, high=10000)
            for j in range(num_rows):
                fake_current = np.append(fake_current, current_level)
    else:
        if last_current != 0:
            current_level = last_current
            for j in range(num_rows):
                fake_current = np.append(fake_current, current_level)
        else:
            fake_current = np.zeros(num_rows, dtype=int)
    fake_current = fake_current.astype(int)
    return fake_current

#Function which generates the voltage values to be inserted
def gen_voltage(num_rows, last_voltage, V_deviation):
    fake_voltage = []
    if last_voltage > 255000:
        new_voltage = last_voltage + np.random.randint(low=-1800, high=500)
    elif last_voltage < 205000:
        new_voltage = last_voltage + np.random.randint(low=-500, high=1800)
    else:
        new_voltage = last_voltage + np.random.randint(low=-1500, high=1500)
    fake_voltage = np.random.uniform(new_voltage-V_deviation, new_voltage+V_deviation, num_rows)
    fake_voltage = fake_voltage.astype(int)
    return fake_voltage

#Function which generates the timestamps to be inserted
def gen_fake_dates(num_rows, startup_offset):
    fake_dates = []
    current_time = datetime.now()
    timestamp = datetime.timestamp(current_time)
    timestamp = math.trunc(timestamp)
    print(timestamp)
    current_time = datetime.fromtimestamp(timestamp-startup_offset)
    for i in range(num_rows):
        time_offset = timedelta(seconds=(2*i))
        new_time = current_time+time_offset
        fake_dates.append(new_time.strftime("%Y-%m-%dT%H:%M:%SZ"))
    return fake_dates

#Function which generates the power and power related values to be inserted
def gen_power(fake_voltage, fake_current, powerFactor):
    p_apparent = fake_voltage*fake_current
    p_apparent = p_apparent/1000
    if (fake_current[-1] != last_current):
        if (fake_current[-1] != 0):
            if (np.random.rand(1,1) < 0.2):
                powerFactor = np.random.uniform(0.4, 0.8)
            else:
                powerFactor = np.random.uniform(0.9, 0.98)
        else:
            powerFactor = 1
    phaseAngle = round(math.acos(powerFactor)*(180/math.pi))
    p_real = p_apparent*powerFactor
    p_real = p_real.astype(int).tolist()
```

```
    p_reactive = p_apparent*math.sin(phaseAngle/(180/math.pi))
    p_reactive = p_reactive.astype(int).tolist()
    p_apparent = p_apparent.astype(int).tolist()
    return p_apparent, p_real, p_reactive, phaseAngle, powerFactor

#Function which inserts the generated values, by putting them in a query per row of data
def insert_rows(num_rows, fake_current, fake_voltage, fake_dates, p_apparent, p_real, p_reactive, phaseAngle,
     powerFactor):
    sql_insert = "INSERT INTO cloud_measurement(lcurrentrms, voltagerms, lmeanactpower,lmeanreactpower,
    frequency,lpowerfactor,phaseangle,meanapparantpower,forwardactenergy,reverseactenergy,absactenergy,
    forwardreactenergy,reversereactenergy,absreactenergy,meteringstatus,sysstatus,temperature,humidity,
    datetime, device_id) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s,
    %s);"
    for i in range(num_rows):
        data = (fake_current[i].item(), fake_voltage[i].item(), p_real[i], p_reactive[i], 50, powerFactor
    *100, phaseAngle, p_apparent[i],0,0,0,0,0,0,0,0,0,0, fake_dates[i], device_id)
        cur.execute(sql_insert, data)

#insert first hour of data instantly such that you don't have to run the program for 1 hour before data gets
    updated properly
def startup(powerFactor):
    cur.execute("SELECT datetime FROM cloud_measurement WHERE datetime < '2050-12-12T08:54:37Z' AND device_id
     = '%s' ORDER BY datetime DESC LIMIT 1;", [device_id])
    last_date = list(cur.fetchone())
    #print(type(last_date))
    last_timestamp = datetime.strptime(last_date[0], "%Y-%m-%dT%H:%M:%SZ")
    current_time = datetime.now()
    timestamp = datetime.timestamp(current_time)
    timestamp = math.trunc(timestamp)
    current_time = datetime.fromtimestamp(timestamp)
    time_delta = current_time - last_timestamp
    time_delta = timedelta.total_seconds(time_delta)
    if (time_delta > 3600):
        num_rows = 1800
        startup_offset = 3600
    else:
        num_rows = math.ceil(time_delta/2)
        startup_offset = time_delta
    fake_current = gen_current(num_rows, last_current)
    fake_voltage = gen_voltage(num_rows, last_voltage, V_deviation)
    fake_dates = gen_fake_dates(num_rows, startup_offset)
    p_apparent, p_real, p_reactive, phaseAngle, powerFactor = gen_power(fake_voltage, fake_current,
    powerFactor)
    insert_rows(num_rows, fake_current, fake_voltage, fake_dates, p_apparent, p_real, p_reactive, phaseAngle,
     powerFactor)
    #Commit data to the database
    conn.commit()
    return powerFactor

device_id = 1
sim_time = 120
#simulation time in minutes rounded up per 16 second chunk. e.g. 5 minutes becomes 304 seconds instead of 300
#voltages in millivolts
base_voltage = 230000
last_voltage = base_voltage
last_current = 0
V_deviation = 200
powerFactor = 0.92

#Define the connection object and cursor
conn = psycopg2.connect(host=dbhostname, port=dbport, user=dbusername, password=dbpassword, dbname=dbname)
cur = conn.cursor()

powerFactor = startup(powerFactor)

#set num rows to the amount of rows of data you want to send to database per chunk.
num_rows = 8 #number of samples per chunk sent to database
chunk_count = math.ceil((sim_time*60)/(num_rows*2))

#Main loop cycling through all functions per chunk of data
```

```
for k in range(chunk_count):
    fake_current = gen_current(num_rows, last_current)
    fake_voltage = gen_voltage(num_rows, last_voltage, V_deviation)
    last_voltage = fake_voltage[-1]
    fake_dates = gen_fake_dates(num_rows, 0)
    p_apparent, p_real, p_reactive, phaseAngle, powerFactor = gen_power(fake_voltage, fake_current,
    powerFactor)
    last_current = fake_current[num_rows-1]
    print(p_apparent)
    print(type(p_apparent[1]))
    #sleep time important, otherwise, arrays made with current time +, so if wrong sleep then wrong data.
    time.sleep(num_rows*2)
    insert_rows(num_rows, fake_current, fake_voltage, fake_dates, p_apparent, p_real, p_reactive, phaseAngle,
     powerFactor)
    #Commit data to the database
    conn.commit()

cur.close()
conn.close()
```