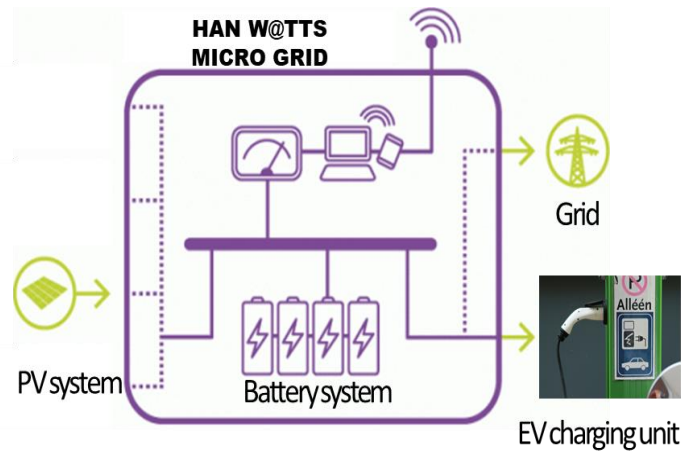


# HANW@TTS Interface

## Project report



### Internship company, Department:

Sustainable Electrical Energy Centre of Expertise (SEECE),  
Measurement and Control Engineering Department

### Faculty, year:

HAN University of Applied Sciences,  
Electronics Engineering Department,  
Semester 5

Student names	Student number
Cristian Batog	612780

Clients and people involved	Contact information
Trung Nguyen (instructor, contact person)	+31 655240818, <a href="mailto:NguyenXuan.Trung@han.nl">NguyenXuan.Trung@han.nl</a>
Ballard Asare Bediako (project client)	+31 655240904, <a href="mailto:Ballard.AsareBediako@han.nl">Ballard.AsareBediako@han.nl</a>
Johan Korten (supervising teacher)	+31 6 14239260, <a href="mailto:johan.korten@han.nl">johan.korten@han.nl</a>

**Place and Date Published:**

Arnhem, January 21<sup>st</sup>, 2021

**Version History**

Version: 1.1

Number	Date	Main reason of change of (sub-)Version number
1.1	January 25 <sup>th</sup> , 2021	Updated after GUI timezone fix
1.0	January 21 <sup>st</sup> , 2021	Draft

## Summary

The 2021 “HANW@TTS interface” internship project was executed by a third-year student from the Electronics Engineering Department of HAN University of Applied Sciences. One company supervisor and one supervising teacher guide the student during the internship.

The objective of this project can be specified in one sentence - “Improve the functionality and reliability of the SOPRA microgrid prototype and establish a connection and interface to the simulation table”. These were the goals set for the project. The main focus was to improve the connection between the subsystems of the SOPRA microgrid and the user interface for interacting with it, leaving out the establishment of the new common interface between the microgrid and the “Smart Grid” simulation table for the end. This last task was not accomplished because of difficulties reportedly encountered by the other team involved while overhauling the simulation table and quarantine measures that impeded physical testing. Still, microgrid-related software was heavily renovated, and many changes were made to make it more reliable and self-sustaining, and to give users access to more data and interactive features through 2 new websites.

During the development process, a lot of time was spent on deconstructing the old software that lacked documentation, updating, simplifying and improving it. An iterative approach based on Rapid prototyping and Rapid Application Development were used to incrementally change the structure, content of the messages and the parsing process used by the 2 controllers to communicate wirelessly. Preliminary versions and testing were done on test boards on a local network before rolling out changes to the real network.

Even though it was challenging to learn to program GUIs in JavaScript and to understand all the details of how the subsystems work in the beginning, the result is quite robust in terms of error handling and added features in comparison with the start prototype. Much more functionality was implemented to the SOPRA microgrid network in contrast to the first anticipated design in the plan of action, partly because of the delayed development of the simulation table.

Of course, as always, there is room for improvement, which has been summarized in the Issues and Unfinished Tasks part of this report. The team has provided a more reliable and user-friendly product that can only be improved from here on out. In short, besides the connection to the simulation table, the suggested improvements are to: block EV chargers from starting a charge without a valid RFID (- the Olimex controllers do not listen to these signals from the Photons), add a car registering page for users on the website, and move the MQTT clients to the prepared local secured broker.

## Table of Contents

Summary .....	3
Table of Contents .....	4
Introduction .....	5
Background .....	5
Analysis .....	7
Problem statement .....	7
Assignment .....	7
Preconditions .....	8
Functional requirements .....	8
Technical Requirements .....	8
Comments regarding the requirements .....	9
Working method: .....	9
Design .....	11
Functional design .....	11
Technical design .....	15
Realisation .....	25
Conclusion and recommendations .....	31
Appendix 1 .....	32

## Introduction

This report describes the system description, subsystems' architecture, realisation stage, tests and conclusions for the internship on the project "HANw@TTS interface" and gives details for why the results turned out as can be seen. The appendices include tutorials to help new users and developers become familiar with the system faster.

The Measurement and Control Engineering Department contributes to a good balance between energy demand and energy supply. The research group focuses on Smart Grids and Energy Systems in the Built Environment.

The research group responds to current developments and international agreements, such as the Paris Agreement. In 2016, government leaders agreed that global warming should be limited to keep the increase in global average temperature to well below 2 °C above pre-industrial levels; and to limit the increase to 1.5 °C. It is no longer self-evident that energy will always be available and at all desired locations. That is why the research group formulated the following goal: "Maintaining the balance between energy supply and demand, both in place and in time."

## Background

The project is being carried out for the Technology and Society Knowledge Centre of HAN.

The organization comprises 10 research groups and 2 associate research groups which conduct applied research and thereby develop knowledge and instruments that contribute to technological developments in professional practice and education. It develops and shares knowledge about renewable energy, in particular electrical energy. The common thread in the research is decentralized generation, storage and electrical energy networks.

HANW@TTS, established in 2019, is run as part of the works of the Measurement and Control Technology Department of the Centre. The contact person for this project is Xuan Trung Nguyen, a researcher at this department.

The aim of the whole project is to create a unified testing and research environment with a user-friendly interface for the use of students and researchers to develop and validate their own power grid simulations on a real setup. Hence, to link the existing system for real-time low voltage interactive simulations, called the "Smart Grid table", to the practical SOPRA microgrid composed of solar panels connected to a battery system, the distribution grid, and to electric car chargers. The data collected from all the elements of the microgrid will be gathered and stored for analysis and used in the Smart Grid table for further investigations.

In 2019, a student group started to implement the communication in the Modbus and the MQTT networks. Data from energy meters in the Modbus network could be read and sent to other nodes in a MQTT network. The SOPRA microgrid software was written in Python on a Raspberry Pi on the solar panels' side and in C++ on Particle Photon microcontrollers on the EV chargers' side. The Pi contains 2 databases that store the energy-related measurements and statistics on the usage of the chargers.

The project was carried out by Cristian Batog and Thijs Meidam separately. Cristian Batog worked on the microgrid side of the project. They spent 115 days = 920 hours each on the assignment during the internship period.

The document is divided into four chapters. The use of the iterative model for the working method ended up with 3 phases for each subsystem in the project – Design, Realisation, Conclusion. The phases for each subsystem are grouped in chapters for readability after Chapter 1: Analysis, where the results of the initial carried-out research are recorded. Chapter 2: Design covers the functional and technical specifications of the chosen systems and the reasoning behind them. Chapter 3: Realisation describes the features of the finished product and test results. The last is Chapter 4 - the conclusion describing the final result; it gives the reader possible suggestions and recommendations for improving the product and states unfinished tasks or known errors. The appendices contain tutorials for future teams working on this project.

## Analysis

### Problem statement.

Initially, there was an EV charger control and user management system working with databases updated with user information, energy consumption measurements from the chargers and energy production data from the PV system assembled by the previous group, but there were errors in the software that need to be addressed, the user authentication system was not functioning, the calculation method used for the renewable energy charging mode was rudimentary and faulty, and the databases were not used to their full potential. Additionally, new software would need to be written at least for the Raspberry Pi if the microgrid would be connected to a bigger wireless network in the future.

Visual representation of the product:

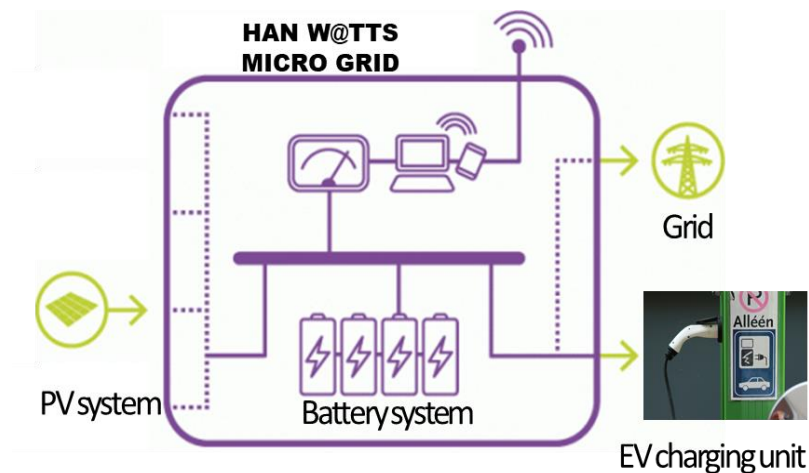


Figure 1 Conceptual model

The working goal was to solve all the issues and errors in the microgrid to prepare it for a connection to a future shared interface and network with the Smart Grid table. The settings of the user interface of the SOPRA system had to be adjusted, the SQLite databases, especially the user list - remade, the message structure and MQTT library used by the EV charger controllers - changed to allow for larger messages and easier parsing, the memory storage solution for scripts and databases of the Raspberry Pi connected to the PV system was investigated, the resistance of the Raspberry Pi to power outages or internet disconnections was to be tested and guaranteed for future use.

### Assignment

The mission of this new assignment was to polish off the SOPRA microgrid's features, fix several issues present in the beginning both with the communication between the controllers and the databases themselves, test and ensure the system's robustness, and to route the collected information from the database into a new network to be shared with the Smart Grid Table.

## Preconditions

The following preconditions were considered when carrying out this project:

- Solve the existing issues with the local network between the Raspberry Pi of the PV system and the Particle Photon microcontrollers of the EV chargers
- Work with the current controllers and software.
- Ensure the robustness of the microgrid in relation to power or internet outages.
- Improve the communication structure and software of the EV charger controllers and implement a new calculation interface for the sustainable charging mode.
- Design an input and output interface between the different systems within HANW@TTS network.
- Setup a local and global network for HANW@TTS (between the Smart Grid table and SOPRA).
- Evaluate and test the interface by establishing a connection between the Smart Grid table simulation and the microgrid.

Nr.	Functional requirements	
F1	The PV system will reliably communicate with the EV charger system.	
	F1.1	The Raspberry Pi's storage containing the databases will be backed up without the removal of its memory card.
	F1.2	The Pi will automatically run the scripts and connect to the broker and the EV charger system upon start-up or restart.
	F1.3	The Pi's SQLite scripts will automatically reconnect to the broker if internet is disconnected.
	F1.4 *	The EV charger measurements and user list databases will be linked, and functions that make use of this will be written.
F2	The EV charger system will be able to send and receive full user and consumption statistics to the database, preferably an entire user list/database, so that usage of the chargers is regulated.	
F3	Other types of measurement variables will be taken from the PV meters and logged in the database.	
F4	A new calculation method and control system for the sustainable charging mode of the EV chargers will be implemented.	
F5	The website will have a page that shows measurements and messages from EV chargers.	
	The website will have a new page where users can register into the user list (but must be verified before being allowed to charge).	
	Create a protected website for the administrator	
	F5.1	It will contain a switch for renewable mode for each charger.
	F5.2	It will have functions to set manual setpoints for every socket.
	F5.3	It will have functions to reset Photons and Olimex controllers.
	F5.4	It will have access to and will be able to edit the user list saved on the Pi (to validate new users and check their data).
F6	The SOPRA microgrid will be connected to a bigger interface together with the Smart Grid table.	
Nr.	Technical Requirements	
T1	MQTT libraries will be used to communicate between the subsystems.	
T2	Python 3.7 and C++ will be used as the programming languages.	
T3	The used libraries will be updated if needed.	



T4	The EV charger system's Particle Photon controllers will send messages bigger than the 512b limit imposed by the current MQTT library.
T5	The Photon will send messages in JSON format, as opposed to the current custom format.
T6 *	The occasional negative power value shown by the PV system on the GUI will be investigated and solved.
T7	The port used for the site of the GUI will be changed for easier user access.

### Comments regarding the requirements

'\*' – signals optional requirements.

F1.1 – At the moment, the SD card of the Pi maxes its memory usage and another USB drive or card cannot be inserted to back its data up in case of a power outage or crash of the card, so the Pi needs to be turned off and its SD must be taken out to be backed up.

F3 – Other types of measurements like active and reactive power, power factor, etc. will also be requested and logged through new functions.

T2 – The old Python software is written in Python 2, so it will have to be changed to work with Python 3.

T4 – The current MQTT library limits the sent message size to 512b, so only the user's ID card number is sent to the Pi. It is desired to have it send information regarding the user, the charged car, voltage, current, power usage, and whether the card is already in use at another charger to block multiple connection attempts.

T6 – The GUI shows that the power value of a certain PV energy meter flips to negative (with the correct absolute value calculated from voltage and current) for some minutes at times. The reason is unclear.

### Working method:

The project was executed using the iterative model based on Rapid prototyping and Rapid Application Development to develop the parts of the system incrementally with breaks in-between them to test and evaluate them with the supervisor and customer before moving on to the following part. The modularity of the system and the lack of connection between the main tasks points to the iterative and incremental development model. There were uncertainties in some areas, so the schedule was subjected to change if one of the tasks ended up as unfulfillable.

The V-model is suitable for the development and integration of these smaller blocks into the complete product. So, the V-model was followed in each individual block because it provides a direct guideline to ensure that no step is missed before an attempt to add it to the prototype is made. It is also important that new errors do not pop up later and that the project does not leave problems or modules that would need to be rebuilt in the future, which is a key moment addressed through unit testing and integration testing as dictated by the V-model.

The issues related to the communication scheme, reliability, and robustness of the Raspberry Pi as an autonomous controller were addressed first.

The Particle Photon microcontroller's communication issues were worked on afterwards, so that it can properly monitor users and act as a working product.

Then, the new calculation method for the sustainable charging mode of the EV chargers was implemented.

Minor issues and the GUI features had to be looked at later.

The last task was to connect the subparts of the microgrid and test that they work reliably, after which another script for the Pi would be written to send the measurements to the Smart Grid table as part of the HANw@TTS interface. This remains an open task.

The programming languages that were used are Python, SQL, C++, and JavaScript. The wireless communication uses the MQTT protocol, JSON messages and dedicated libraries for them.

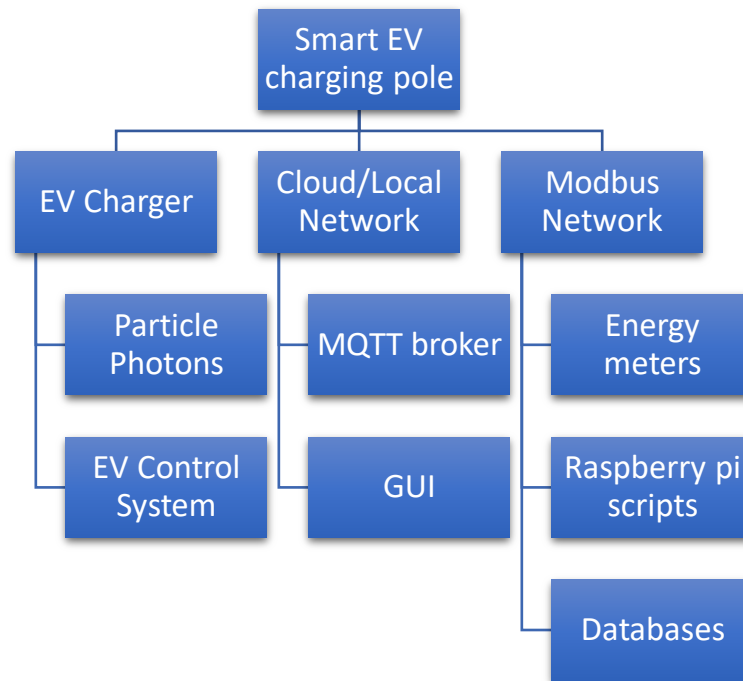
Besides that, the student used the SOPRA microgrid's subsystems physically for testing and validation purposes.

Details regarding the basic working principles of the used communication protocols (Modbus, MQTT, SPI, webhooks), software used, and the electrical connections between the controllers inside the system are not included in the report because they are already given and explained in the last groups' project reports. They should be consulted first if any questions regarding the points arise.

## Design

### Functional design

This chapter discusses the functional design and its related diagrams. The subsystems block diagram is updated but the input-process-output diagrams of the subsystems have not changed since the past year.



*Figure 2 Subsystems block diagram*

This project assignment is not concerned with the established wired connections between the Photons and the controllers of the EV chargers, or between the DEIF MIC mk1 energy meters and the Raspberry Pi, as these were tested in past revisions of the system. Thus, the system to be worked on is divided into 3 main subsystems: the MQTT network, the Modbus network of the Raspberry Pi, and the GUI that connects to both networks.

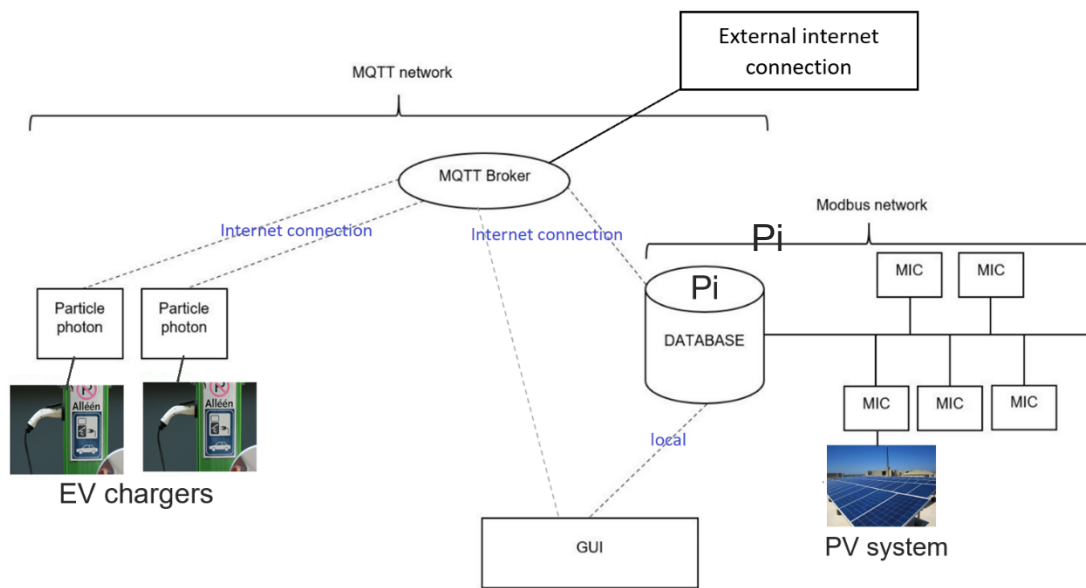


Figure 3 Conceptual design

The **Modbus network** contains the Raspberry Pi communicating through a MAX 485 to 5 DEIF MIC mk1 energy meters that continuously take electrical measurements from the: electrical energy grid, energy

reserve, Battery system at the HAN, solar panels, and a Demonstration setup. The SPI messages from the Pi are converted to Modbus and vice versa.

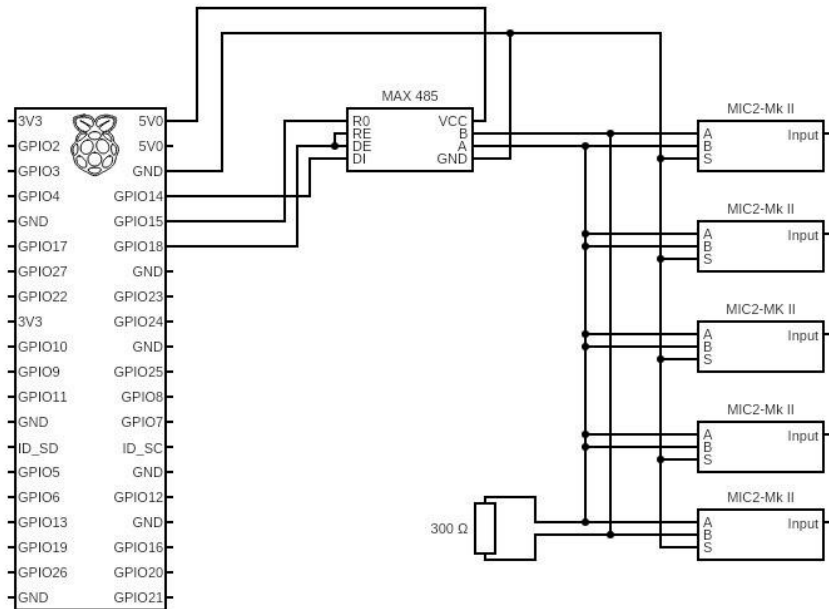
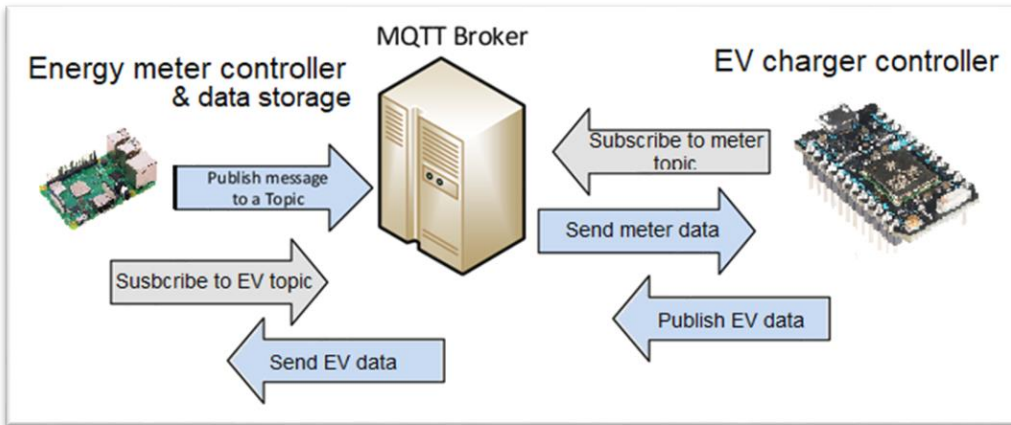


Figure 4 Modbus network electrical schematic

\*The electrical schematic is taken from the previous team's report and is used for reference only.

The **MQTT network** is set around the MQTT broker, used to establish the connection between the scripts on the Raspberry Pi, the databases, the Photons and the GUI, so the MQTT network is used to send messages:

- from Raspberry Pi to Particle Photon: renewable mode setpoint for Current output, answer to deny access to a user or start charging a car
- from Particle Photon to Pi: measurements from the EV chargers, RFID card swipes from users
- from Particle Photon to GUI: RFID card swipes, answers/feedback to users after a card swipe
- from GUI to Photon: admin functions (setpoints, reset, switch charging mode)
- from GUI to Pi: new user register data to userlist, admin edits to userlist in user database



*Figure 5 Basic MQTT flows*

The **Graphical User Interface** consists of 2 separate websites - one for all users, where energy meter readings and EV charger measurements are shown, feedback from the Photon is displayed, and new users can register into the database. The other website is secured by password because it is intended for administrator use. The Photons can be reset or the charging mode can be changed. Access and editing rights for the user database are available on this page.

The GUI is ran on the Raspberry Pi because it needs direct access to the databases. It also receives MQTT messages published by the Photons and can send commands to them from the administrator page.

## Technical design

This chapter discusses the technical design of each subsystem and its related diagrams.

The **Raspberry Pi** is part of both the MQTT network and the Modbus network. Thus, the process box on the left of the diagram contains the script that manages all the necessary software functions of the Modbus network and writes measurement data to the Modbus database. It also needs the number of active chargers, information which is logged in the users & photon database. The publish message uses the JSON structure, as opposed to the old custom ‘%’-separated values.

The process box on the right side explains the Pi’s functions in the MQTT network. This script interacts with the Particle Photons and the users & photon database. A MQTT loop is forked at the start of the program, then the main loop checks if the MQTT client is disconnected and restarts it. The `send_email()` function is also executed by the main loop every 5 minutes.

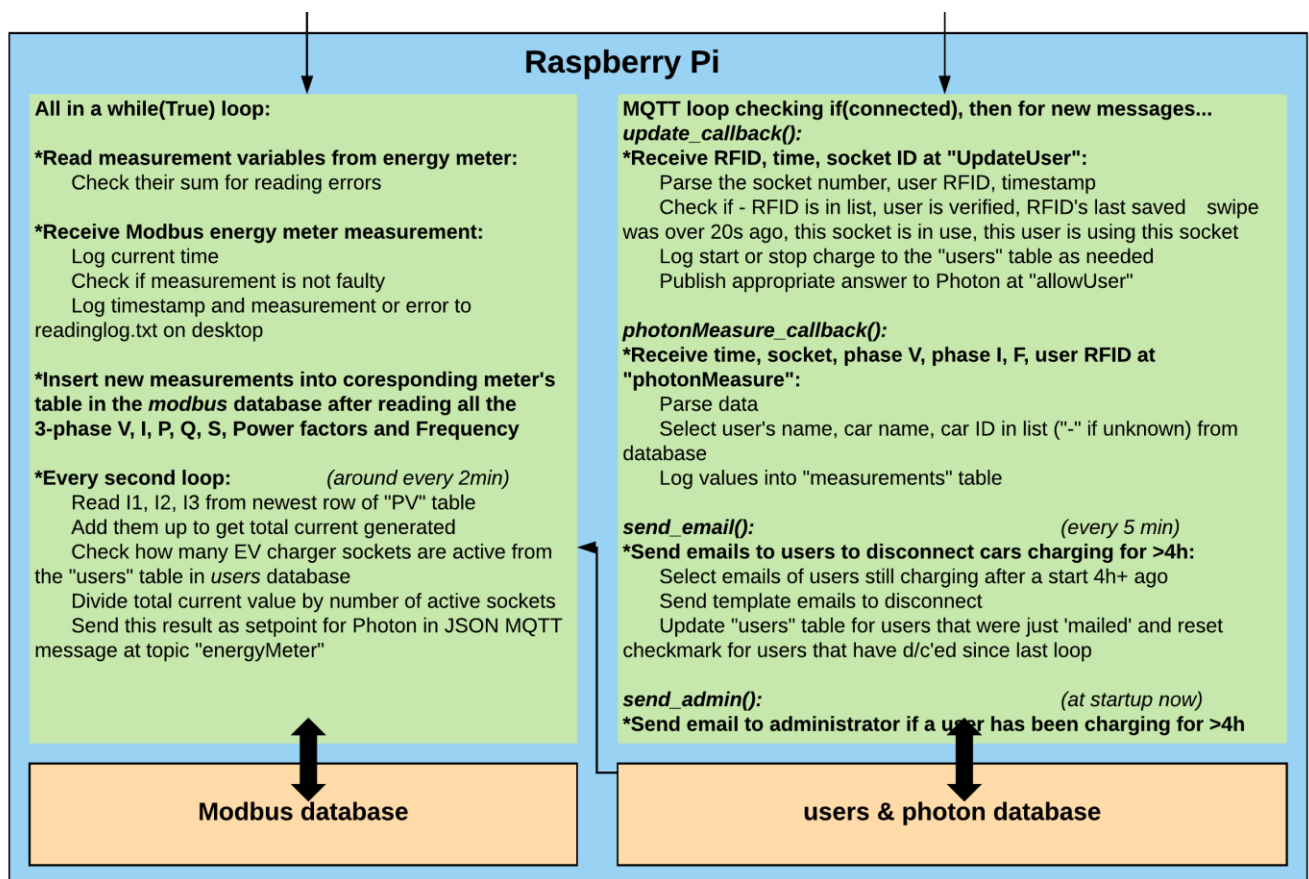
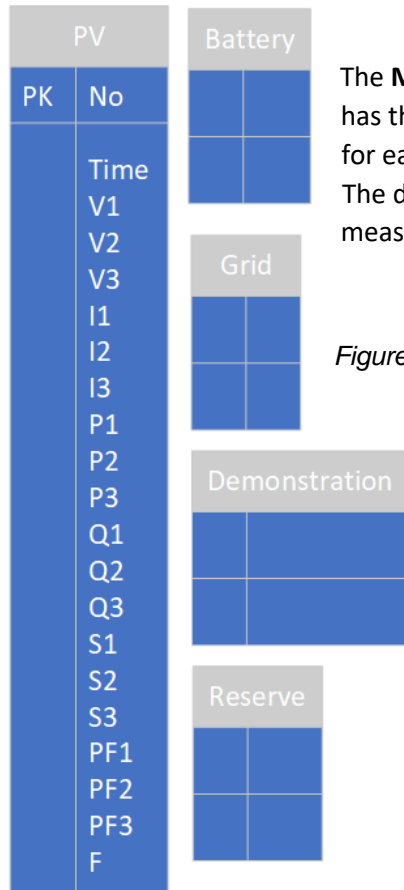


Figure 6 Raspberry Pi pseudocode process diagram

The **databases** used have multiple tables each. The system used for them is SQLite because it is light, local, does not need Internet or server access, which simplifies the task and is more efficient considering the small processing power of the Raspberry Pi. The old databases have been changed and rebuilt at this stage of the project. They are also saved both on the SD card of the Pi and in HAN's WebDAV server.



The **Modbus database** has 5 tables – 1 for each energy meter. Each table has the same columns for energy-related measurements. The primary key for each of them is the measurement 'No'. There are no foreign keys here. The database is backed up and cleaned up monthly because a lot of measurements are stored in it.

*Figure 7 Modbus database entity relationship diagram*

The **users & photon database** has 8 tables. Each table has the same columns for energy-related measurements. The primary keys differ here; the *car\_of\_user* and *error\_codes* tables have composite primary keys. The other tables' primary keys are 'id', the default PK in SQLite. The structure of this database is taken from an old MySQL database used before the HANW@TTS project was started.



The *data* and *error\_codes* tables are not used actively in the current state of the project. They were added for future versions of the product. The *charging\_stations* table contains the unique 'DeviceId' of the Particle Photons available and the *charging\_station\_sockets* table contains the unique charging sockets derived from the number of Photons in *charging\_stations*. Their *chargingStationId* is a foreign key for the *users* and *measurements* tables to include the correct real Id of each socket in the tables. The unique 'uidTag' column from *users* is a foreign key for 'userId' in *measurements*. 'carId' is a foreign key from the *cars* table. Each user – car combination is saved in the *car\_of\_user* table, where each 'carId' and 'userId' are a (unique) composite key, made to link existing 'id's from *users* and *cars*.

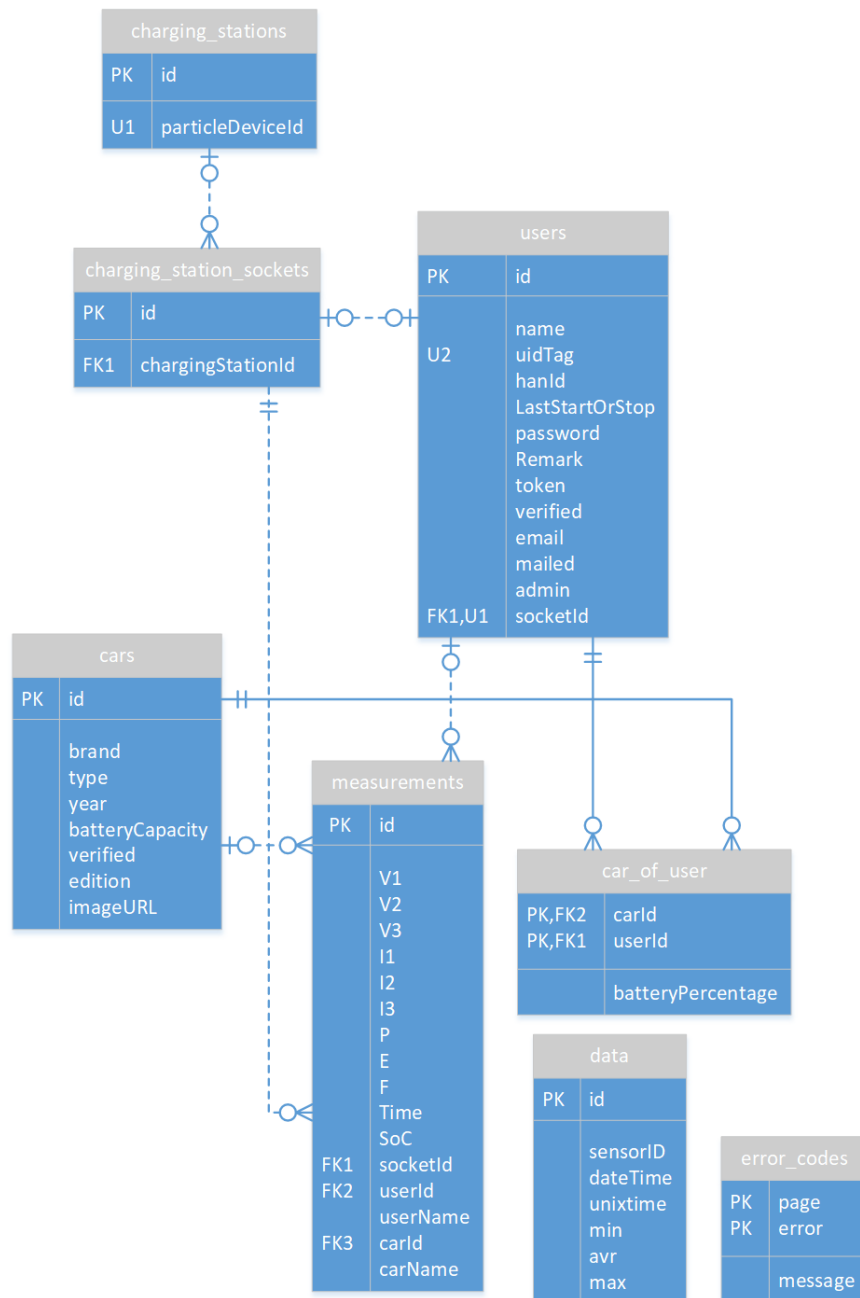


Figure 8 Users database

The **Graphical User Interface** is built in Node-RED on the Pi and divided into 2 separate parts: one for normal users, and one for the administrator. The last team already made a GUI that showed data from the Modbus database consisting of 5 pages with energy graphs for the meters and one to show the latest important measurements. The idea here was to build on what was already done and add functionality. The meter graphs also started in the wrong time zone and that had to be fixed.

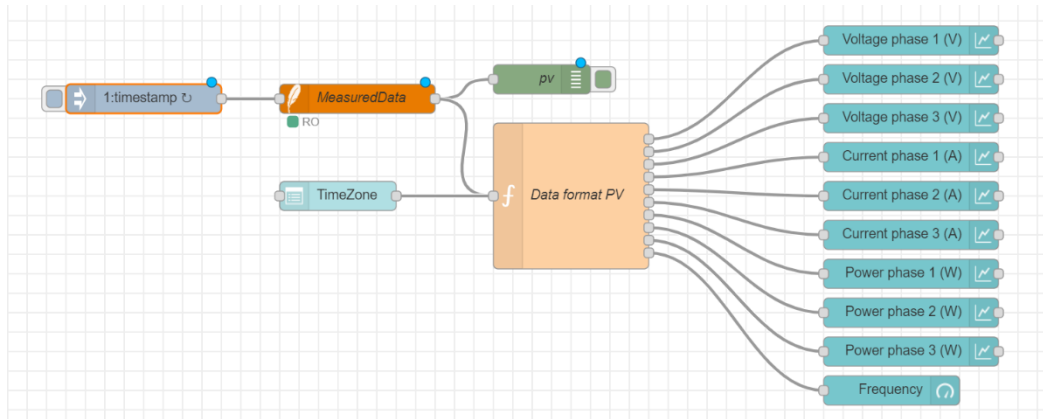


Figure 9 Meter data flow

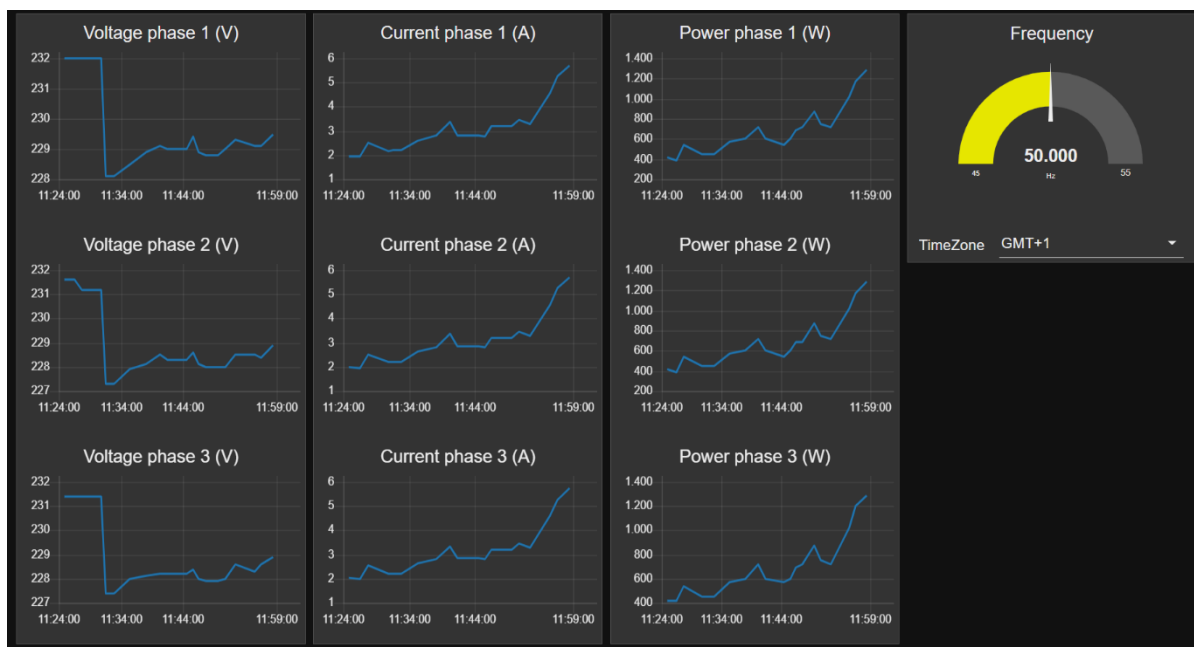


Figure 10 Meter GUI page

Thus, a new page to show the data received from the EV chargers and a page for registering new users had to be made to make the GUI more useful for the customers. It is inspired by the original meter pages, but does not show the power values and shows all 3 voltage phases in one graph (, same for current,) to make the page more compact and easy-to-read. Every charger socket has the same data flow that is triggered by a timestamp every 10s. The latest Frequency value is shared between the two sockets of each charger. The RFID swipe responses from the Photons are picked by the MQTT input node and shown for each socket ('C3:' here).

The socket watchdog resets its textbox if no new messages were received in the last 5 minutes. The 'Change\_Check' node only passes data if it has changed and the 'Watchdog chart' node sends a NULL value to cut the charging line in the graph if no new measurements were received in the last 4 minutes.

These watchdogs make it easy to see that a socket is not used at the moment because its boxes are empty and its graph lines are severed (= not connected between 2 car charges that are hours apart).

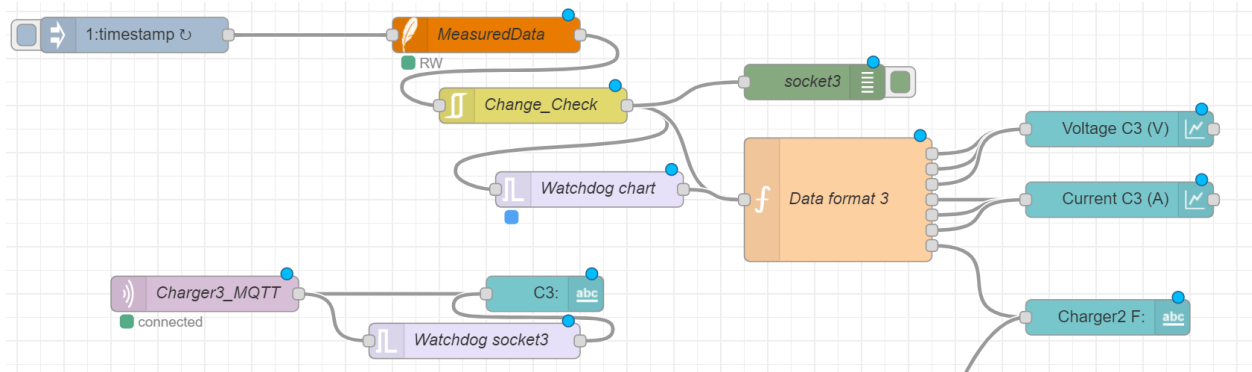


Figure 11 Charger data flow

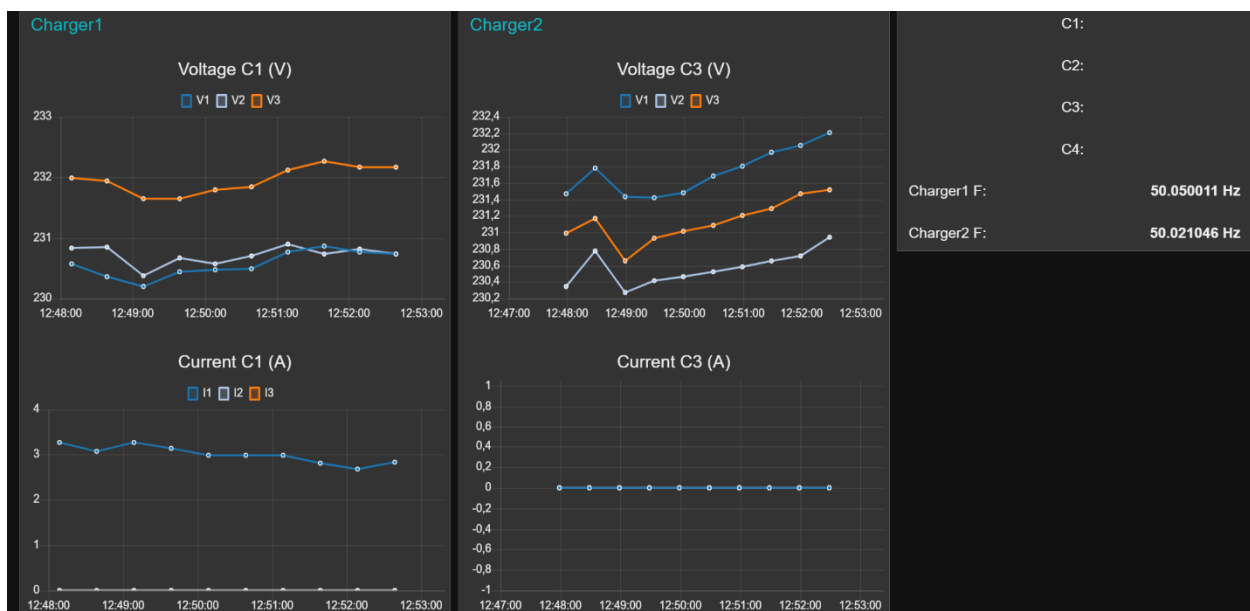


Figure 12 Chargers GUI page

The user registering page is quite simple from a technical and functionality aspect. A user writes his name, RFID, email, optionally HAN ID and a password, then submits these in the 'user\_form'. The 'rbe' node only passes a different new message (in case the user pressed submit multiple times), then the data is parsed and sent to the users database. No answer from the SQL node triggers a successful register text to appear in the feedback textbox. Otherwise, the 'error\_catcher' shows what went wrong.

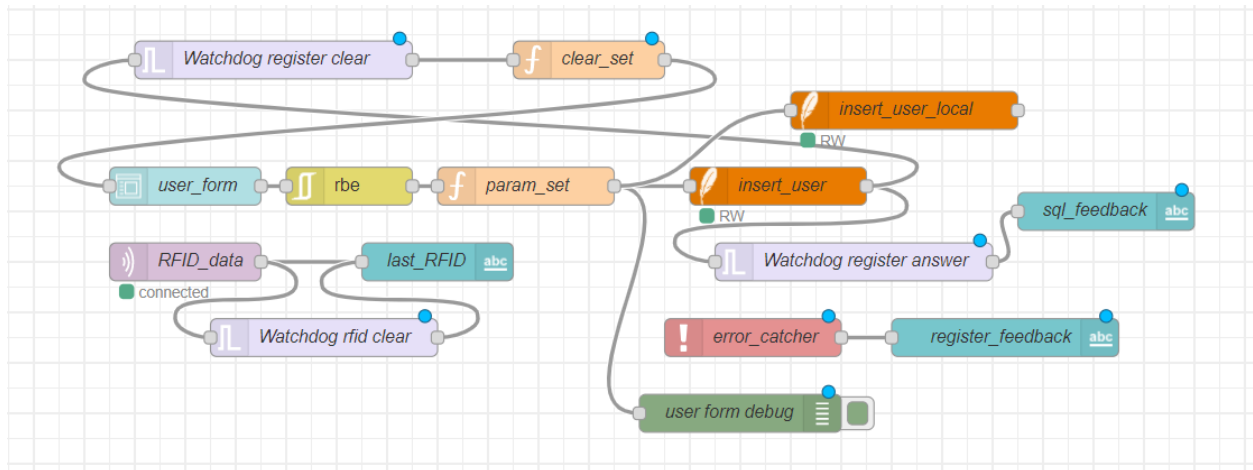


Figure 13 Register page data flow

Another small feature is a textbox that shows the last swiped RFID and received by MQTT. This should help users find out the RFID tag of their card if they want to register it. The 'Watchdog register clear' and node clears the 'user\_form' after a successful sign up and the 'Watchdog rfid clear' – the RFID textbox if it was not updated in the last minute. This is needed because Node-RED is a one-user interface, so multiple computers see exactly the same page and a user's data should not remain visible on the page after he signs up.

Figure 14 GUI registering page

A new administrator GUI for interacting with the Photons and with the *users* database needed to be created as well. It is built from another Node-RED repository and has another web address and port number because it has to be user-password protected and not accessible to normal users.

The Photon functions are triggered by button presses or input in textboxes that send messages through MQTT to their own topics.

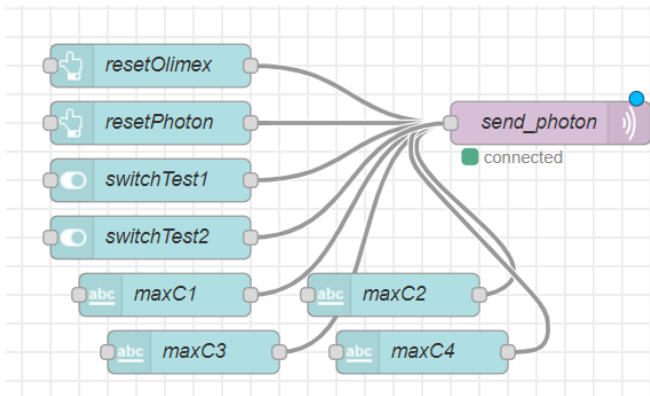


Figure 15 Photon functions data flow

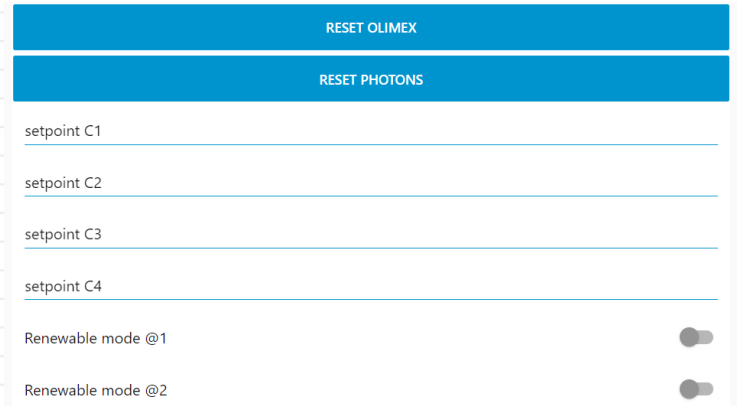


Figure 16 Photon functions GUI

The access and editing to the *users* table of the *users* database needed a special node type not available in the default Node-RED palette. The “node-red-node-ui-table” v0.3.10 was chosen because it lets the programmer choose what to display from the given input and what columns to make editable while also auto-formatting the table to fit on the screen of the used device. This is convenient for our use case scenario. The final table is somewhat similar to an Excel sheet.

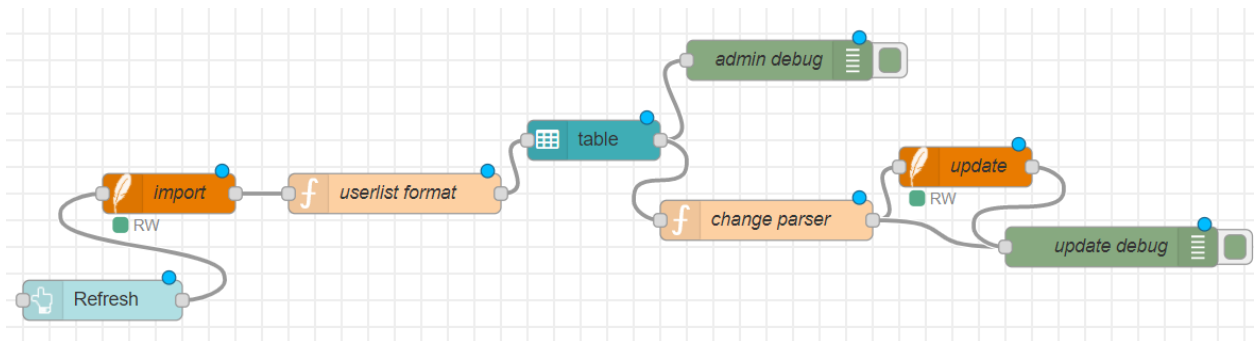


Figure 17 Usertable data flow

The needed data is imported from the SQLite database after a button to refresh it is clicked. Then, the data is parsed to show only the important columns and set which should be editable. Any changes are parsed back into commands and written to the database automatically, so a field should be edited carefully.

REFRESH										
id ▲	name ▲	RFID ▲	HAN ID ▲	last sw... ▲	remark ▲	verif... ▲	email ▲	notif... ▲	charger ▲	
1	Yuri van Geffen	97 2D 39 5D	567745	1552383791	567745	✓	example1	✗		
2	Trung Nguyen	04 4E 79 EA ...	567744	1541761670	1234567890	✓		✗		
3	Test 1	BB 77 E3 59	0	1552305521	1234567890	✓		✗		
4	Test 2	8E FD B3 89	1	1552375671		✓		✗		
5	Trung Test	66 FB 67 D9	2	1610537932	1234567890	✓		✗		
6	Menno Merts	60 8A 8A 7C	3	1539787113	1234567890	✓		✗		
7	Dave Mateman	B6 06 DB DB	4	1552983009	1234567890	✓		✗		
8	Boes-Voet Maria	60 AD 8F 7C	5	1550765473	1234567890	✓		✗		
9	Dave Mateman	04 81 3C AA ...	6	0	1234567890	✓		✗		
10	Johan Brussen	80 41 84 7C	7	0		✓		✗		
11	Grijff Katja	B0 49 8A 7C	8	0		✓		✗		
12	Johan Brussen	04 1F 51 12 ...	9	0		✓		✗		
13	Cornelissen Peter	90 03 8F 7C	10	1552978846		✓		✗		
14	New	67 1C B6 89	11	1548666182		✓		✗		
15	admin istrator	AA BB 11 22	111111	0	Dit is de ad...	✓	oose.canterbury@g...	✗		
17	Joris Huinink	04 5F 26 F2 ...	597240	0		✓	jorishuinink@hotm...	✗		
18	Wouter Noordhof	04 2C 22 52 ...	605251	0		✓	wh.noordhof@stud...	✗		
19	Dennis Gommer	04 7F 79 5A ...	598695	0		✓	Dennis.Gommer@...	✗		
20	Jasmijn Bartelds	45 HJ 67 JH	602898	0		✓	jasmijn.bartelds@h...	✗		
25	Joris Huinink	BB EE RR GG...	597241	0		✓	jowjoris@gmail.com	✗		
26	Arne Brethouwer	EV DA TA BA...	234567	0		✓	arne@ev-database....	✗		

Figure 18 Usertable GUI

The **MQTT network** contains not just the Pi and the Photons, but two separate GUIs as well. The user GUI receives and shows the swipe responses from the Photons at 'photonConverted/1' through 'photonConverted/4' and the swiped RFIDs because the EV chargers do not have physical displays so there is no other way of letting users know if they are allowed to charge or not (and the reason why). The shown RFID, taken from 'updateUser', is used to help them register in the database if needed.

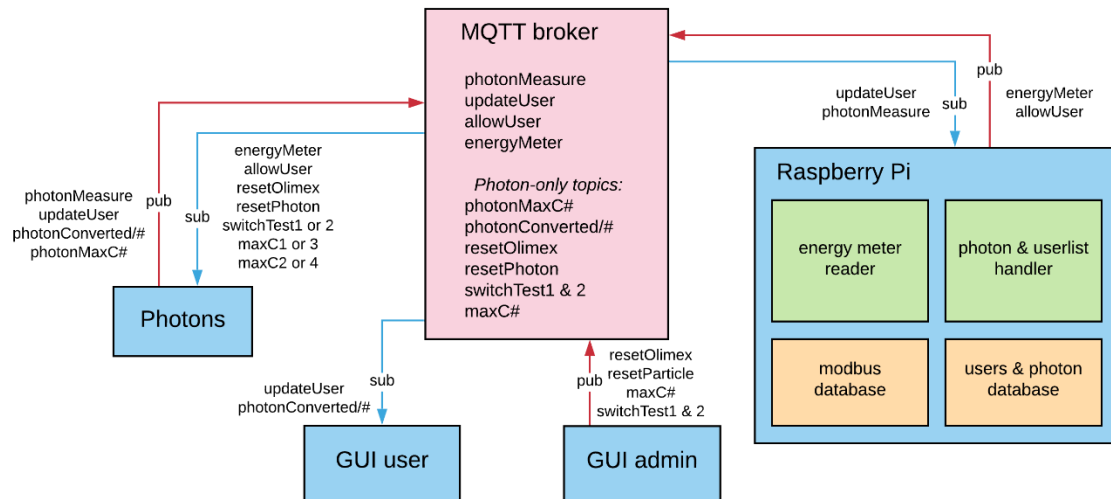


Figure 19 MQTT network diagram

The admin GUI only sends commands for the Photon, similarly to proprietary Particle functions. The ‘switchTest’ function switches between renewable and regular charging mode for each Photon separately (- ‘switchTest1’ and ‘switchTest2’, respectively). ‘maxC1’ through ‘maxC4’ are used to set the regular maximum charging current setpoints for every socket.

The Raspberry Pi sends ‘energyMeter’ setpoints for the renewable mode of the Photons and ‘allowUser’ responses to allow or deny charging with an explanation code after receiving new RFID swipes from Photons on topic ‘updateUser’. ‘photonMeasure’ messages are the energy measurements from the EV chargers. These are saved in the *users & photon* database in the Pi.

The **Particle Photon** can only run one program at a time. Thus, there is a `setup()` function that runs first at start-up or reset and a `loop()` functions continues to run indefinitely afterwards. Data received from the EV charger controllers (Olimex) is saved and parsed and MQTT messages are handled in the `loop()`. Incoming messages trigger callback functions.

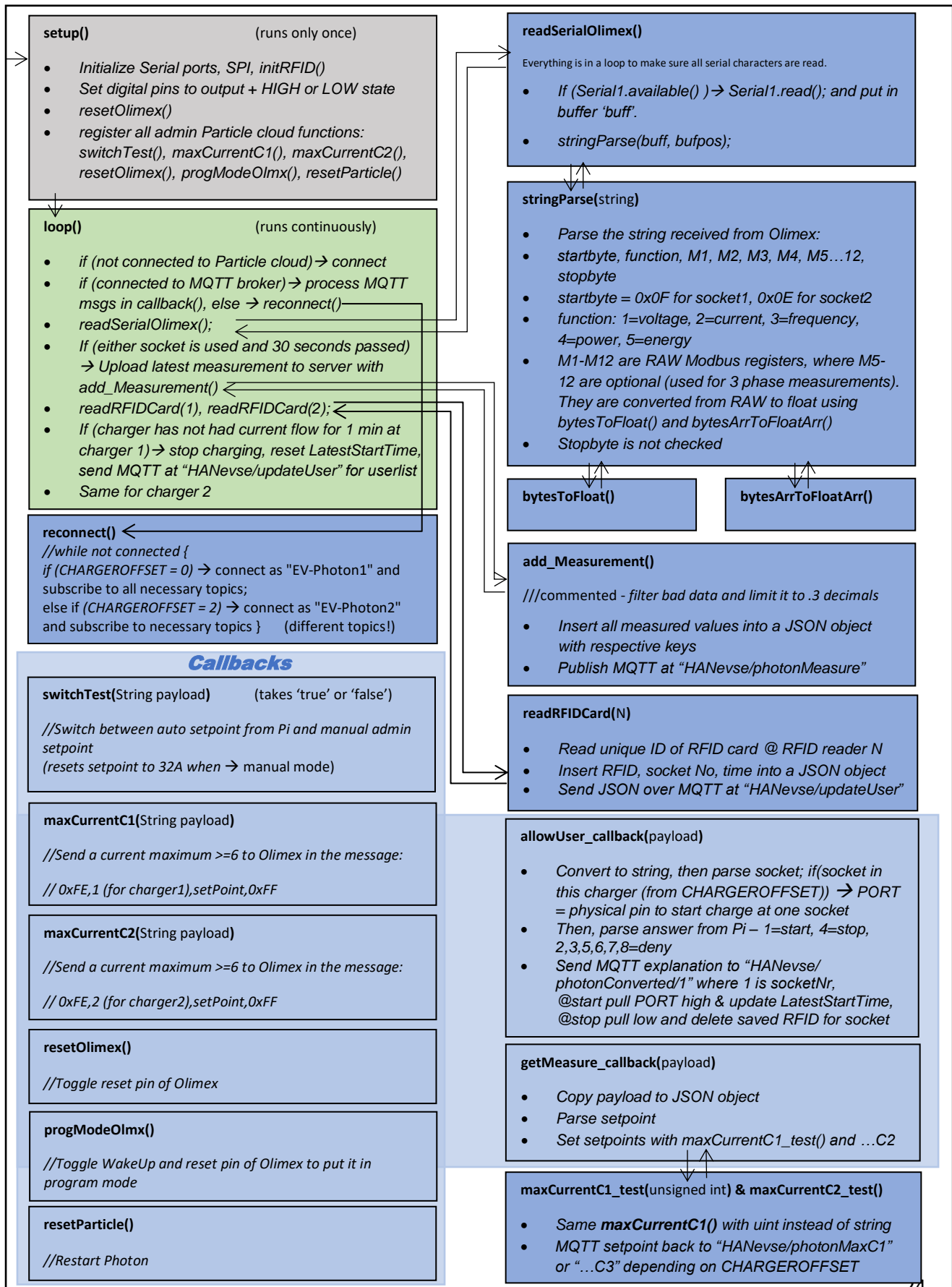


Figure 20 Process flow diagram for Photon in pseudocode



## Realisation

This chapter includes more details about the features of the finished subsystems and links to the software documentation for them. Tutorials on how access the Pi, Photon and the GUIs are given in the appendices section.

Every subsystem was developed separately first, then the networks were connected and the whole system was tested with an actual EV.

The **Raspberry Pi** runs its two main programs at boot and routes their outputs to the “userlog.txt” and “readinglog.txt” files for debugging. This command is written in “rc.local” in the /etc folder of the Pi. The main programs are “/home/pi/Documents/SQLfunction.py” and “home/pi/Documents/Modbus/main3.py”. They use many try-except blocks to catch connection errors to the DBs and avoid crashing. This allows them to run reliably for weeks, as tested.

The Pi is also configured to run a local Mosquitto MQTT broker (secured with named users and passwords for the MQTT clients of the project) with the “runmosquitto.sh” script, “mosquito.conf” and “passwords” files in its default user Home folder. This feature is not activated in “rc.local” at the moment because of time constraints and to ease access for debugging communication issues from outside the network. The public internet MQTT broker “broker.hivemq.com” is used at the moment.

The Pi is set up to log all SQL communication to the SQLite **databases** “usertable.sqlite3” and “modbusData.db” on a (16GB) SD card and to the HAN WebDAV server at the same time. This is implemented because there have been incidents with SD card corruption in the past. Still, the server itself is not always available or responding, so the SD card databases were kept in case of an internet disconnection. These two storage devices are mounted at boot as well in “rc.local”. the “/mnt/dav/Data” folder contains the server databases.

They receive data constantly from the chargers and energy meters, so the size of the databases increases rapidly. To help keep them light and accessible for the admin a script called “runDBreset.sh” was written to copy and save all 4 databases with the date of copy, then empty all the tables to clean them up. This script is run by the root user’s cron service every first day of the month. The file that holds this command can be opened and edited with “sudo crontab -e” in the terminal. Its copy in the report folder is named “root\_crontab”.

The user **GUI** has a systemd service that starts it at boot. All of its files are stored in the “.node-red” folder in the default user Home folder. The admin GUI has its own separate folder called “node-red-1881”. These two are separated because the admin GUI has to be password-protected and hidden from the normal users and Node-RED is a single-user tool that only accepts password entry for a whole GUI, not just one of its pages. The admin GUI is started separately at boot with the “runNodeRed.sh” script in “rc.local”, placed last, as it needs access to the internet and the databases to run correctly.

To ensure maximum robustness for the Pi, its entire used storage is automatically cloned to another spare SD card once a week with the “sudo rpi-clone sdb” command in the root crontab. This uses the rpi-clone repository from github.com to copy everything without unmounting the working storage device.

The *documentation* for the software of the Raspberry Pi is available either in .html form or in LaTeX format in the “HANwatts\_Pi-main/documentation” folder. The more user-friendly and the easier to access on any platform is the html documentation that can be opened by any web-browser. To open the main page for the html documentation open “/html/index.html” from the documentation folder. (it was not possible to make a hyperlink or a shortcut work here because this report and documentation are stored in Onedrive and Windows uses absolute paths).

The **Particle Photons** run the same software project that was built in C++ with Wiring libraries in MS Visual Studio Code with the Particle Workbench package and dependencies. They are flashed wirelessly with access to the administrator account. The project can also be accessed at “build.particle.io/” with access to the account. The project is called “HAN-evse-json1” on the Particle internet service. One paramount detail regarding programming them is that the constant “CHARGEROFFSET” in the main \*.ino file needs to be set to 0 for the Photon EV1 and to 2 for EV2 to avoid crashing the two Photons because of identical MQTT client IDs.

The new Photon software also uses the JsonParserGeneratorRK library from the Particle services to create and parse JSON MQTT messages. This simplifies the callback functions and makes it easier to debug and add more variables in them in the future.

The *documentation* for the software of the Particle Photons is available either in .html form or in LaTeX format in the “HANwatts\_Photon-main/documentation” folder. To open the main page for the html documentation open “/html/index.html” from the documentation folder.

The “Particle Photon code” part of it discusses more details about the main files for the Photon.

The **GUIs** are built in Javascript and Node.js. To make them accessible without the IP address of the Pi and to make communication more secure for the Pi, domain names were procured. The user GUI is connected to the URL <https://sustainablecharging.nl/>. It is also connected to <https://hanev.duckdns.org/> just in case something happens to the official site. The admin GUI is connected to <https://hanevadmin.duckdns.org/> because it does not need an official site and using a free Dynamic DNS provider was the easiest way to give it a URL. The port number does not need to be specified after the URLs anymore after a Nginx reverse proxy server was configured to route the user GUI URLs to port 1880 on the Pi and the admin GUI URL to port 1881. If the URLs are changed, the Pi will not respond anymore until another entry is added for the reverse proxy. To add more security and protect the Pi from internet attacks, a firewall was set up and auto-updating TLS certificates were made for these URLs. They can be accessed only with HTTPS now.

The programming side of the GUIs is accessed by adding “/admin” after the main URLs. To see and edit the flows used by Node-RED one needs to enter the programmer’s username and password (different for the two GUIs).

The inbound data for graphs and the userlist table is taken from the SQLite databases in the Web

server. If access to the server is cut, they do not update anymore. Ideally, the server is not available only when there is no internet access and the Node-RED programs do not work either. In reality, HAN's server is reset almost weekly on Sundays (presently), so an automatic reboot action on Monday at 4.30am was added in the root crontab of the Pi. The databases in the HAN server and the local SD card databases do not synchronize.

The basics of Node-RED programming are explained in the last team's report. The Modbus meter data flows were not modified since then. Only the graphs were cleaned up. The new Javascript functions for parsing and formatting the SQLite data from the databases is straightforward – a variable is given to each measurement value and they are sent as an array to the graphs. The graphs with multiple lines in them also use topics for their variables.

```
if (msg.payload === false)
{
  var msg_V1 = {topic:"V1", payload: null};
  var msg_V2 = {topic:"V2", payload: null};
  var msg_V3 = {topic:"V3", payload: null};
  var msg_I1 = {topic:"I1", payload: null};
  var msg_I2 = {topic:"I2", payload: null};
  var msg_I3 = {topic:"I3", payload: null};
  var msg_F = {payload: null};
}
else
{
  var msg_V1 = {topic:"V1", payload: msg.payload[msg.payload.length - 1].V1};
  var msg_V2 = {topic:"V2", payload: msg.payload[msg.payload.length - 1].V2};
  var msg_V3 = {topic:"V3", payload: msg.payload[msg.payload.length - 1].V3};
  var msg_I1 = {topic:"I1", payload: msg.payload[msg.payload.length - 1].I1};
  var msg_I2 = {topic:"I2", payload: msg.payload[msg.payload.length - 1].I2};
  var msg_I3 = {topic:"I3", payload: msg.payload[msg.payload.length - 1].I3};
  var msg_F = {payload: msg.payload[msg.payload.length - 1].F};
}
return [msg_V1, msg_V2, msg_V3, msg_I1, msg_I2, msg_I3, msg_F];
```

Figure 21  
EVcharger data  
format node

The user registering page sets message parameters from the user input instead of using variables because this allows for the use of prepared SQL statements to automatically insert the data into the correct columns and eliminates the risk of SQL injection.

```
msg.params = {
  $name:msg.payload.name,
  $uidTag:msg.payload.UserId.toUpperCase(),
  $hanId:msg.payload.hanId,
  $password:msg.payload.password,
  $email: msg.payload.email
}
return msg;
```

Figure 22 Registering

param\_set node

</> SQL Query    Prepared Statement    ▼

</> SQL Statement

```
1 INSERT INTO users (name,uidTag,hanId,password,email) VALUES (($name,$uidTag,$hanId,$password,$email))
```

Figure 23 Registering insert\_user node

The user registering page is the only one that interacts both with the server database and the local database.

## Test plan and results

The subsystems were tested for functionality swiftly and fixed along the way before being connected together. The controllers and the websites are (almost) always running and new measurement data is added all the time and can be checked by accessing the URLs.

The situation with the quarantine made it hard to execute physical tests for system verification with actual cars. Still, one test with an EV (Nissan Leaf) and another customer car took place on 05.01.2021.

No	Test	Expected result	Pass/Fail
1	Swipe card at free charger	Message with RFID and socket nr. is sent over MQTT and shown on website	Pass
2	Swipe card with a connected car	Message with RFID and socket nr. is sent over MQTT and shown on website	Pass
3	Swipe card with charging car	Message with RFID and socket nr. is still sent over MQTT and shown on website	Pass
4	Swipe card at all sockets	All 4 sockets send messages with the RFID and their socket nr. and the messages appear on the website	Pass
5	Swipe one card many times in a row	User RFID is logged only once, then messages that “you already swiped in the last 20s” are sent	Pass* a user card can be logged out after a minimum of 20s after logging in
6	Swipe different card after log in with user card	Second card is denied log in and an appropriate message is shown on the website	Pass
7	Swipe unauthorized card	Card is declined with a message on the website but Measurements still contain its RFID	Pass
8	Swipe card to log out, then swipe another user’s card	After first card is successfully logged out another RFID card is logged in immediately (after less than 20s)	Pass
9	Connect and charge car at all sockets	All 4 sockets send Measurements over MQTT	Fail – only socket 1 and 2 charge and send Measurement messages
10	Software reset on charger with a charging car	Charger keeps charging and sending Measurements even after reset	Fail – charger does not recognize the car after reset, it needs to be plugged in again to restart charging
11	Connect car without swiping card	Car still charges and Measurements are sent	Pass – charger works and sends Measurements with “No ID” as user RFID
12	Connect car after swiping card	Car charges and Measurements with the swiped RFID are sent	Pass
13	Unplug car without swiping card	Car can be unplugged and Measurements stop	Pass

14	Unplug car without swiping after starting charge with a swipe	Car can be unplugged and charger automatically sends log out message with RFID for database	Pass – log out message is sent after 1 minute
15	Disconnect fully charged car to plug another instead	When car is fully charged automatic log out message is sent and socket is unlocked	Pass
16	Try to disconnect another user's charging car	Another user's cable cannot be unplugged from socket if his car is still charging.	Pass* It can be unplugged after charger software reset.
17	Swipe another (unauthorised) card while a car is charging	The new card is denied and Measurements will still contain the initial swiped RFID at log in	Fail – the second card is always denied if someone else is logged in at socket but Measurements will contain the newly swiped RFID
18	Set Current setpoint from admin website	New setpoint is set and Measurements show capped charging current	Pass* After reset the has to be enabled in the control room first
19	Switch to sustainable mode from website	Charger reads the automatic setpoint messages sent by the Pi, answers with actual setpoint set (min. 6A), Measurements show the limited current	Pass
20	Switch back from sustainable mode to administrator setpoint mode	Measurements still work but the current is not limited by the old setpoint.	Fail – charger switches back successfully but setpoint has to be manually changed afterwards

There were 4 failure points found in this test. Afterwards, they were checked and point 9 and 20 were solved. All 4 charging sockets send measurements now because they were set to send measurements at all times, not only when the chargers are putting out current. The Photon function to switch back to the manual admin setpoint mode now automatically executes two `maxCurrentC#()` callbacks and sets the setpoints to 32A, which is the maximum output Current for the chargers.

An answer to point 9 was not found, so the Photons should not be reset while cars are using the chargers. Point 17 was fixed in software but it was not tested physically afterwards because of a lack of time.

## Conclusion and recommendations

In conclusion, the team managed to rebuild and expand the software of the Pi and of the Particle Photons, the databases, the GUI and its website with the functionality specified in the Realization part of the report. Looking back at the requirements given from the project description and analysis, most of them were met, apart from the issues labelled as future improvements below.

What was finished: The Pi's storage is backed up to another spare SD card automatically every week; the databases are backed up to HAN's server through WebDAV; the Pi runs all the necessary programs by itself (at boot); the scripts themselves do not crash if they cannot access the SQL DBs; the EV measurements add the charging user's name from the userlist and car information; the EV system sends consumption data, but not entire databases, as the Photons are too weak for that; Q, S, Pf are also requested from the Modbus meters and logged in the Modbus DB, a new calculation method for the setpoint was created and moved to the Pi, so it is still accurate if all sockets are used now; the GUI has a new page for the EV chargers and a new page for registering users who still need to be verified by the admin manually; a new GUI for the administrator to interact with the Photons and to verify new users was made, the energy meter graphs start in the correct time zone now.

On the technical side: MQTT is used by the controllers to communicate; Python 3.7, C++ 14 and a little JavaScript were used to program during the project; the used libraries were updated and JSON libraries were added to the Pi and Photons; all MQTT messages (besides the Pi answer to RFID swipes) use the JSON format; the wrong power values on the GUI were fixed; the GUI do not need port numbers to be accessed anymore.

Issues that could not be tackled: The SOPRA microgrid was not connected to a bigger interface together with the Smart Grid table because the simulation table project encountered issues and is not ready to communicate with the microgrid; The EV chargers do not send entire databases, just 1 measurement or 1 RFID swipe at a time because exceeding the 512b message size limit poses crashing risks for the Photons (very little RAM available).

Thus, Appendix 1 provides a Start-up and User Manual on how to access the Raspberry Pi over the internet, Appendix 2 discusses accessing and flashing new software to the Photons, and Appendix 3 shows how to enter the programming mode of the GUIs.

As a future improvement, it is suggested to: update the car registering feature of the user GUI because the table that links the userID and the carID is updated only in the server DB; test if the Photons hold a charging car's RFID and one cannot overwrite it with another swiped RFID while the car is still charging; make the user GUI take data from the local DBs if the server ones are unavailable and make admin GUI edit and write to the local user DB, too; synchronise the server DBs with the local ones or move most of the SQL data traffic to the local DBs and update the server DBs once a day or so.

Other observed issues are sometimes HAN's server refuses to respond and sends an empty database, so the Pi needs to reconnect to the server to start working again; both Photons randomly shut down one day at 10.20am and had to be physically restarted.

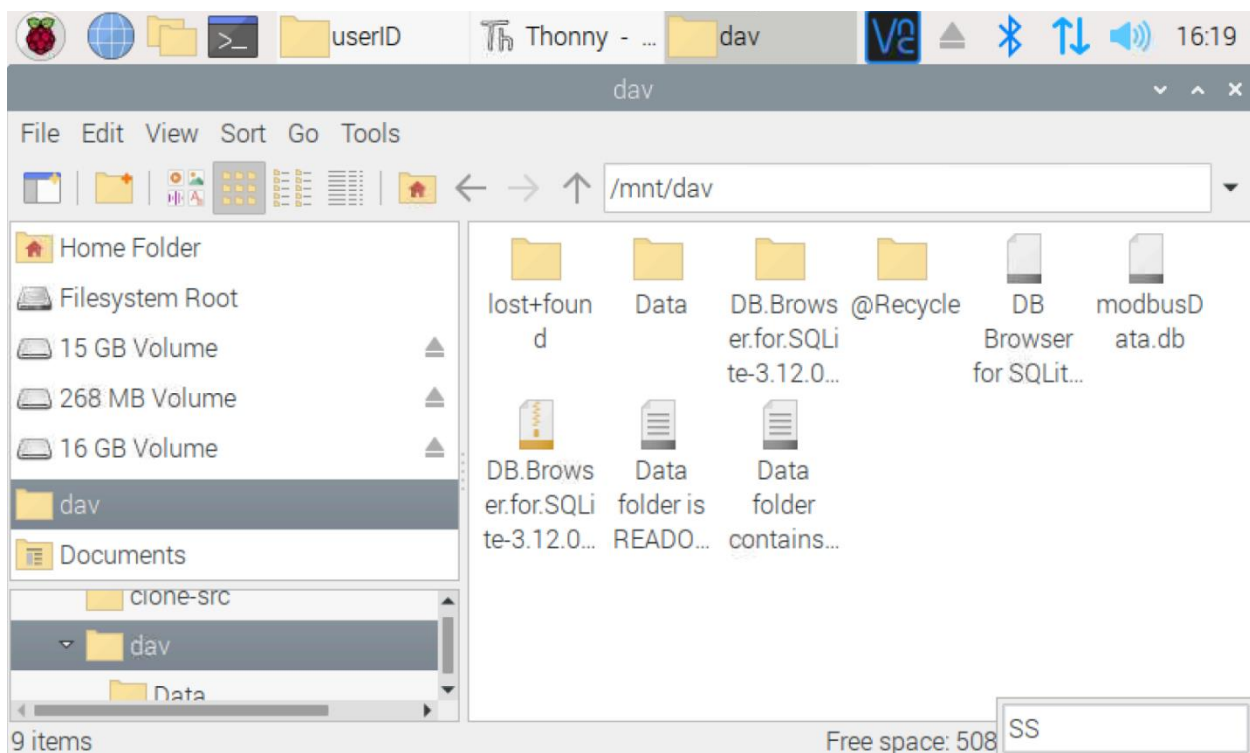
## Appendix 1 How to access the Raspberry Pi and SOPRA PC

To access the Raspberry Pi remotely, you have to use VNC Viewer. For Windows, download it from <https://www.realvnc.com/en/connect/download/viewer/windows/>.

Run VNC Viewer and enter the following VNC server address in the top search bar: 80.113.19.27

The username for it is: pi . The password for it is: controlsystem .

The log files for the scripts are on the desktop. The main scripts themselves are “/home/pi/Documents/Modbus/main3.py” and “/home/pi/Documents/userID/SQLfunction.py”. You can also find them through File Manager (2 folders icon at the top left of the screen). Other important files like the \*.sh scripts and the Node-RED folder are in “/home/pi/”. The HAN WebDAV server is “/mnt/dav/” and the databases are in its Data folder. The local databases are in “/media/DATABASE/”, also called 16 GB VOLUME on the left side of the File Manager.



To access the PC connected to the Pi and the EV chargers you need Teamviewer. The partner ID is 1487692927 . The password changes occasionally, so you need to ask your supervisor about it or find it in person in the SOPRA room.



The files used by the Pi are in the “HANwatts\_Pi-main” folder with the report. Their functions and locations in the Pi are given here as well for convenience. The filenames are the same in the Pi.

Filename	Location	Function
<b>/Modbus/main3.py</b>	/home/pi/Documents/Modbus/main3.py	Main Modbus program
	/home/pi/Desktop/readinglog.txt	Log file for main3.py
<b>/userID/SQLfunction.py</b>	/home/pi/Documents/userID/SQLfunction.py	Main EV charger program
	/home/pi/Desktop/Userlog.txt	Log file for SQLfunction.py
<b>sql_databases/modbusData.db</b>	/mnt/dav/Data/modbusData.db	Modbus databases
	Also in /media/DATABASE/	
<b>sql_databases/usertable.sqlite3</b>	/mnt/dav/Data/usertable.sqlite3	Users databases
	Also in /media/DATABASE/	
<b>runDBreset.sh</b>	/home/pi/runDBreset.sh	Script to copy and empty DBs (used in crontab)
<b>rc.local</b>	/etc/rc.local	File that runs scripts at boot
<b>root_crontab</b>	(type in terminal) <code>sudo crontab -e</code>	File that schedules periodic jobs
<b>runNodeRed.sh</b>	/home/pi/runNodeRed.sh	Script to run admin GUI (used in rc.local)
<b>/node-red-1881/settings.js</b>	/home/pi/node-red-1881/settings.js	Config file with user-passwords for admin GUI
<b>/node-red/settings.js</b>	/home/pi/.node-red/settings.js	Config file with user-passwords for normal user GUI
<b>runmosquitto.sh</b>	/home/pi/runmosquitto.sh	(unused) Script to run secured local MQTT broker
<b>mosquitto.conf</b>	/home/pi/mosquitto.conf	Configuration file for runmosquitto.sh
<b>passwords</b>	/home/pit/passwords	File with MQTT users and passwords for mosquitto.conf
<b>*Tutorial for Nginx, firewall</b>	<a href="https://discourse.nodered.org/t/node-red-server-with-nginx-reverse-proxy-howto-guide/27397">https://discourse.nodered.org/t/node-red-server-with-nginx-reverse-proxy-howto-guide/27397</a>	Tutorial needed to add new URLs to reverse proxy

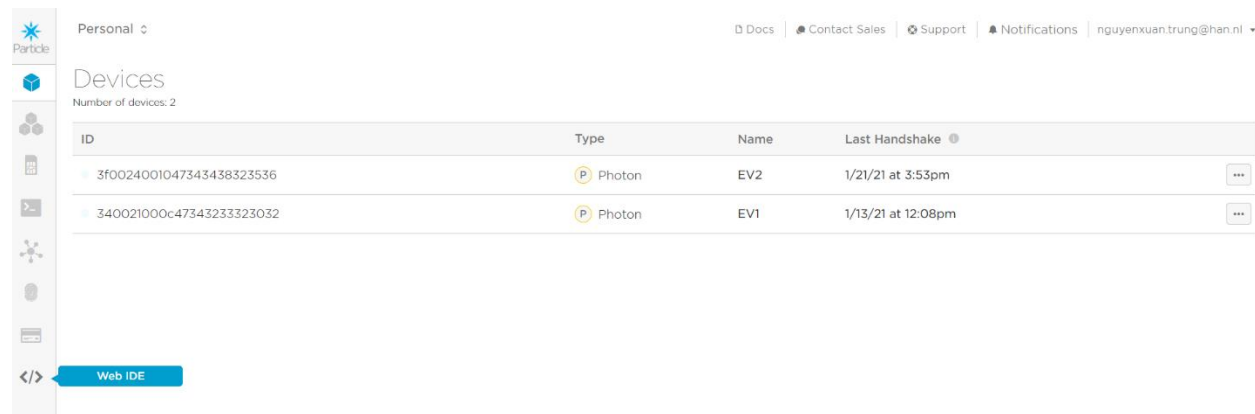
## Appendix 2 Particle Photon access

To access the Photons you need access to the Particle administrator account that is linked to them.

Go to <http://console.particle.io/> and log in. The user name is [nguyenxuan.trung@han.nl](mailto:nguyenxuan.trung@han.nl)

Ask Mr Nguyen for the password to the Particle account.

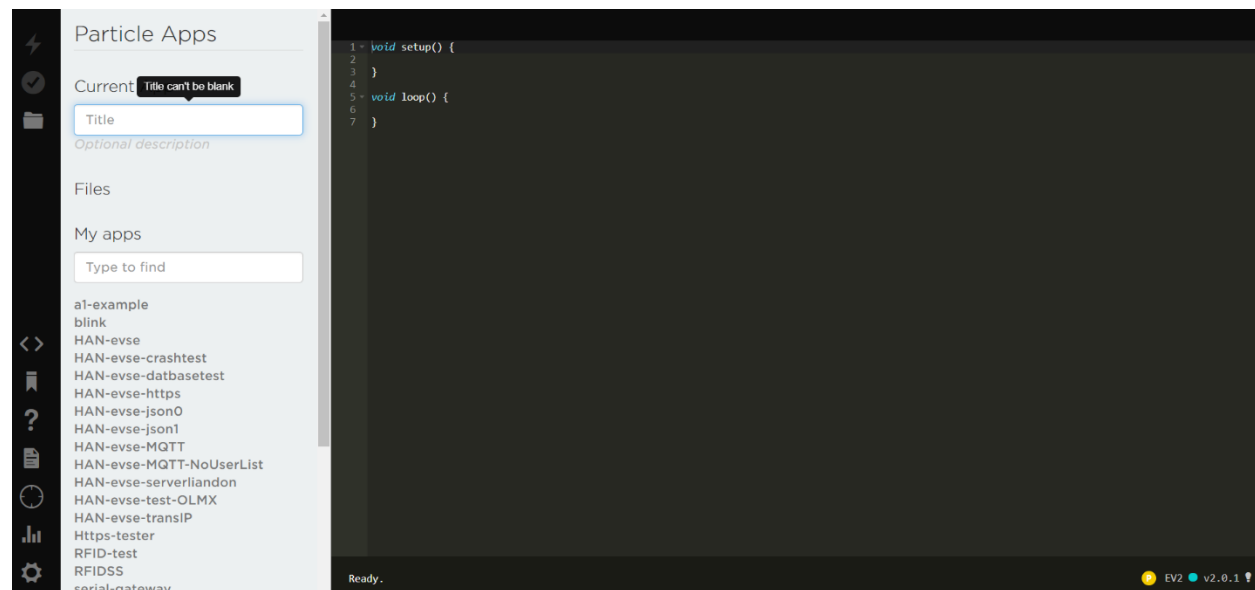
You should see the 2 EV Photons and their IDs after you log in.



A blue circle on the left of the IDs indicates that they are online.

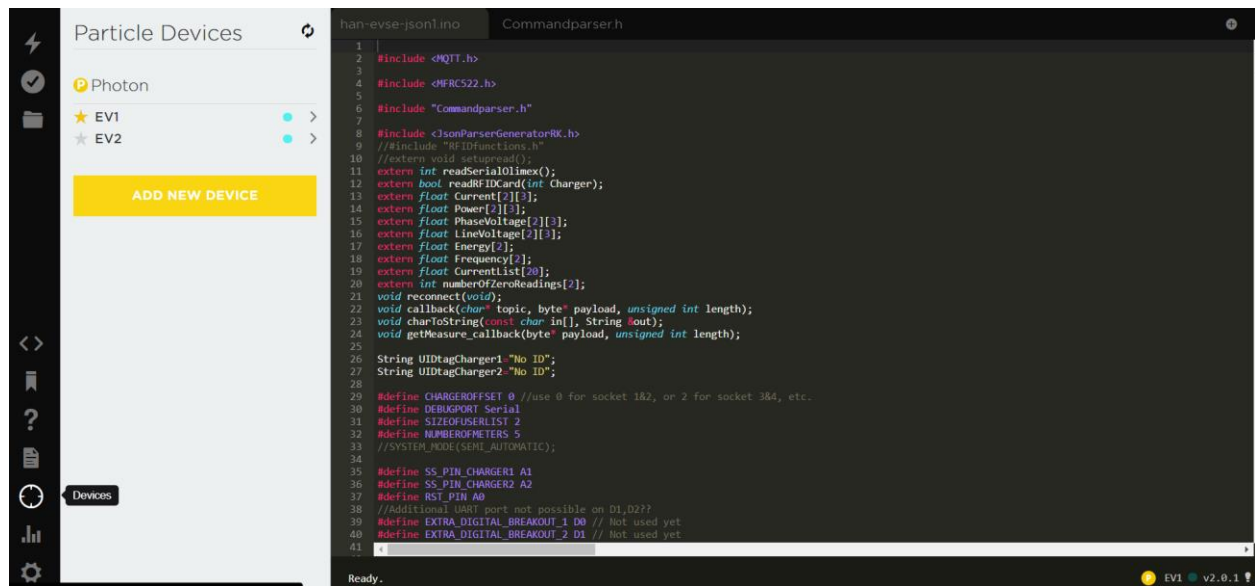
To access the IDE and the software of the Photons, click on the lowest icon on the left of the site (Web IDE).

To work on the current software running on the Photons choose “HAN-evse-json1” from the list of apps.



The current software is compiled for Photon v2.0.1.

Choose the device you want to flash from the target icon on the left of the screen, then press on the star to the left of the device's name. Don't forget to set the correct CHARGEROFFSET for your device.

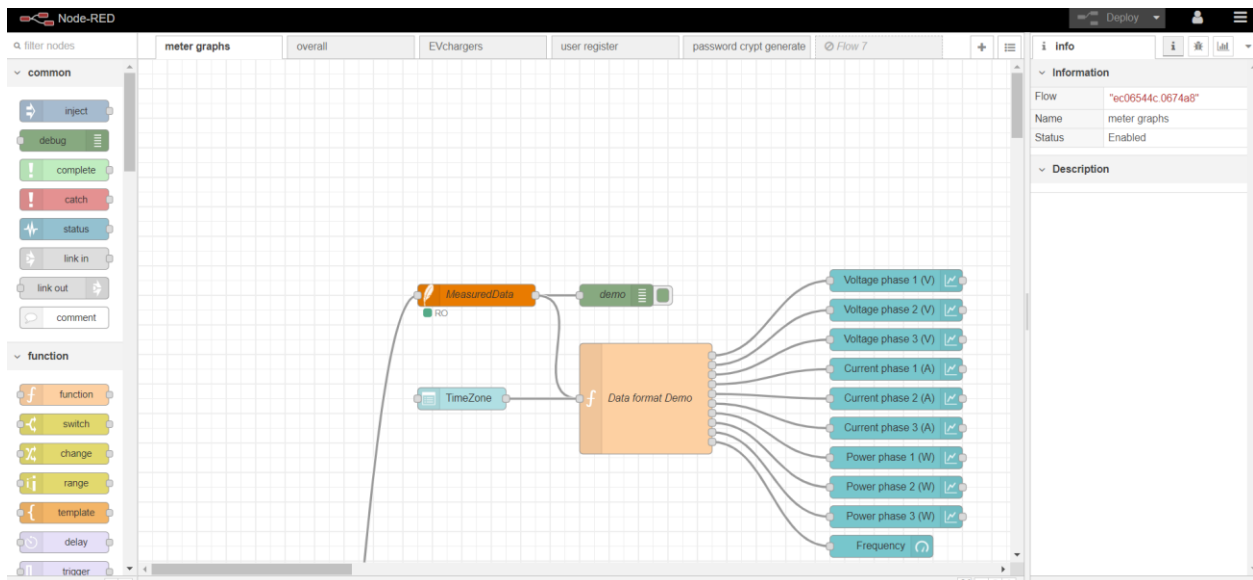


After modifying the program, verify it by clicking the second top icon on the left side (the checkmark). The IDE will say **"Code verified! Great work."** if you have no errors. You can flash it to your device by clicking the top icon on the left side (the lightning).

## Appendix 3 Node-RED GUI editing access

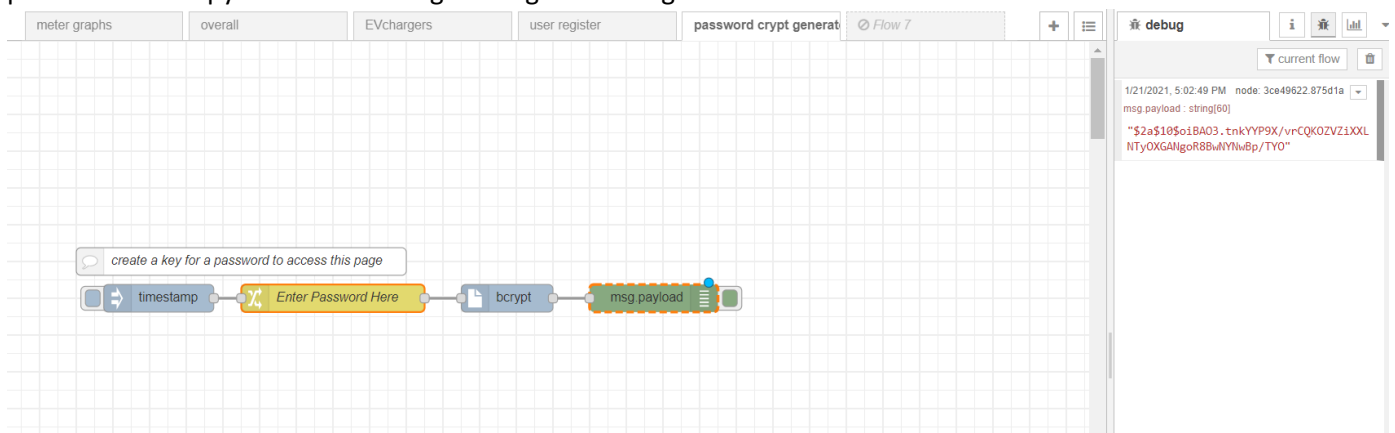
To edit the normal user GUI at <https://sustainablecharging.nl/> you have to go to <https://sustainablecharging.nl/admin/>. The username is: admin . The password is: controlsystem . The credentials can be changed after finding adminAuth in the "settings.js" file in the "/home/pi/.node-red/" folder. The password must be encrypted with bcrypt. You can go to the "password crypt generate" page in the editor to generate a new password fast.

You should see this page after you have logged in. The pages here do not represent the pages in the GUI.



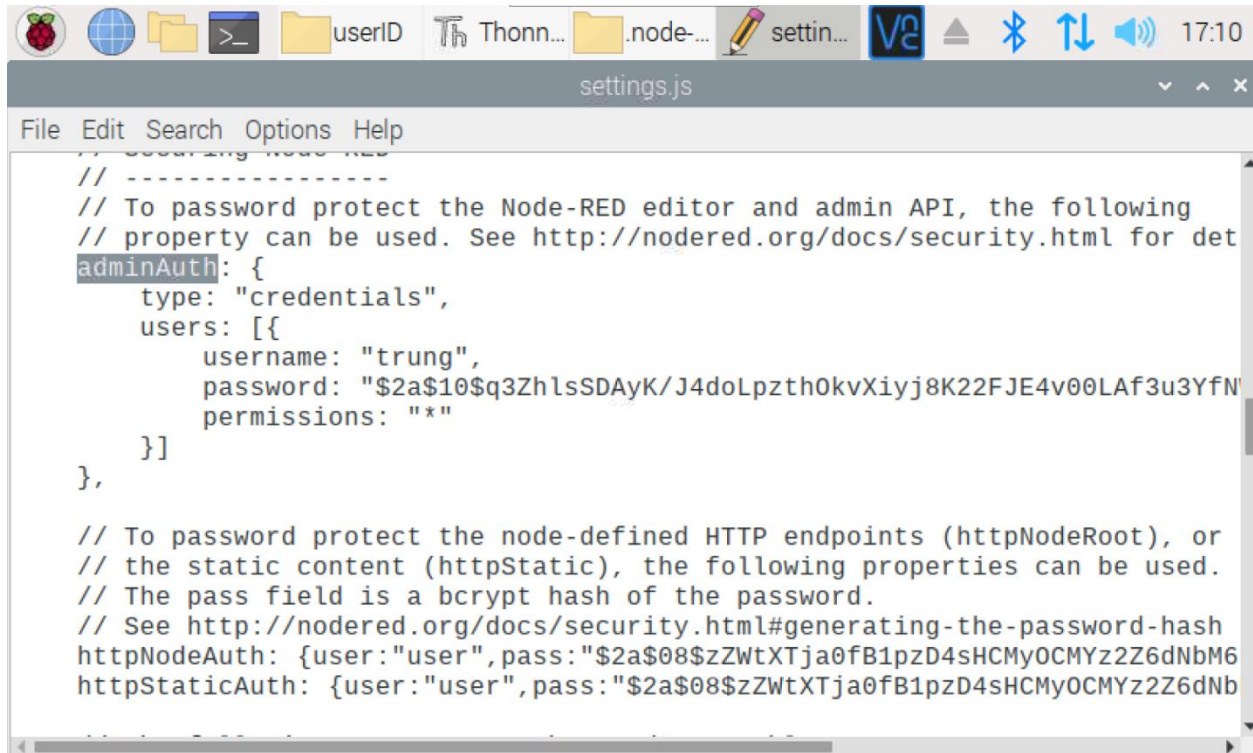
After you make any changes to the nodes you must click Deploy in the top right corner of the site. It will turn red if there are any changes to save.

To generate a new password, go this page and enter your password instead of controlsystem in the yellow-ish node. Click on the bug icon on the right side of the screen and select "current flow" from the 3 filters under it. Click on the blue square on the left of the timestamp node to generate the encrypted password and copy it from the string message on the right side of the screen.



To access the admin GUI, go to <https://hanevadmin.duckdns.org/> and input username: user. The password is: password. These credentials can be changed after finding httpNodeAuth in the "settings.js" file in the "/home/pi/node-red-1881/" folder.

To access the editor for this GUI go to <https://hanevadmin.duckdns.org/admin/> and enter username: trung and password: controlsystem . These credentials can be changed in the same way as the ones for user GUI, but in the "/home/pi/node-red-1881/" folder.



```
// -----  
// To password protect the Node-RED editor and admin API, the following  
// property can be used. See http://nodered.org/docs/security.html for det  
adminAuth: {  
  type: "credentials",  
  users: [{  
    username: "trung",  
    password: "$2a$10$Q3ZhlsSDAyK/J4doLpzth0kvXiyj8K22FJE4v00Laf3u3YfN",  
    permissions: "*"   
  }]  
},  
  
// To password protect the node-defined HTTP endpoints (httpNodeRoot), or  
// the static content (httpStatic), the following properties can be used.  
// The pass field is a bcrypt hash of the password.  
// See http://nodered.org/docs/security.html#generating-the-password-hash  
httpNodeAuth: {user:"user",pass:"$2a$08$ZzWtXTja0fB1pzD4sHcMyOCMyz2Z6dNbM6"},  
httpStaticAuth: {user:"user",pass:"$2a$08$ZzWtXTja0fB1pzD4sHcMyOCMyz2Z6dNbM6"}
```

## Appendix 4 Webdav

Open Terminal.

Type `cd /etc/davfs2`

`sudo nano secrets`

change username and password

`/mnt/dav CristianBatog "W@ttsN3xt"`

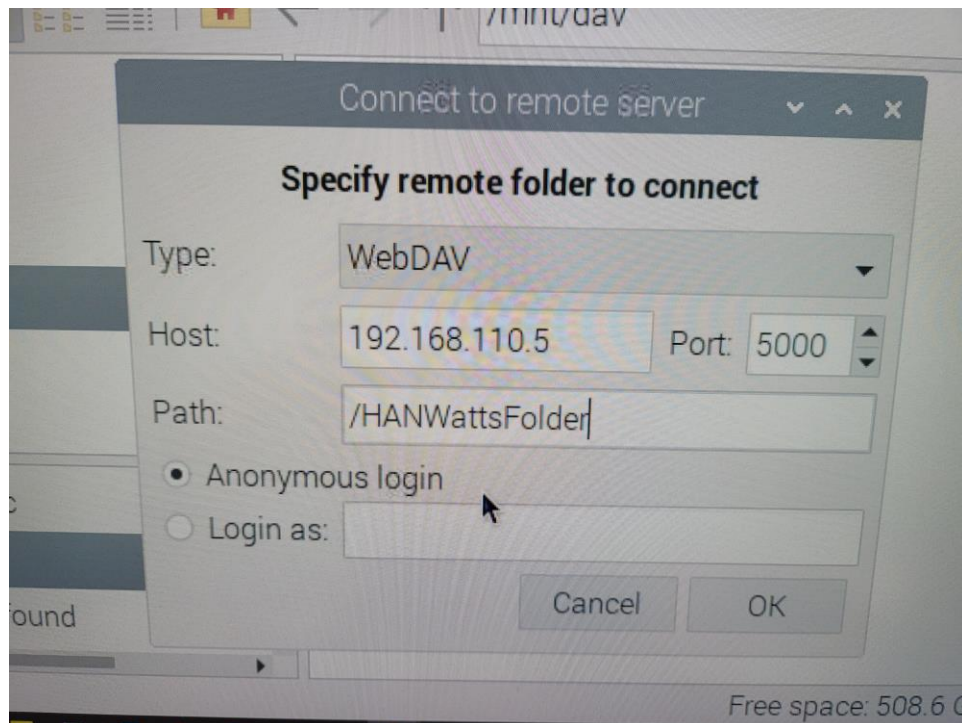
-----

Crontab for Automate the job is in etc.

`Sudo nano rc.local`

### Last issues:

Port 80 has been changed to port 5000 -> Fixed by `sudo nano /etc/fstab` change port to 5000



192.168.110.5 port 5000

/HANWattsFolder

CristianBatog "W@ttsN3xt"