УНИВЕРЗИТЕТ “Св. КИРИЛ И МЕТОДИЈ” - СКОПЈЕ

ФАКУЛТЕТ ЗА ЕЛЕКТРОТЕХНИКА И ИНФОРМАЦИСКИ ТЕХНОЛОГИИ

- **дипломска работа** -

од област

МЕРЕЊА ВО ЕЛЕКТРОТЕХНИКА

Тема

РЕАЛИЗАЦИЈА И МЕТРОЛОШКА ВЕРИФИКАЦИЈА НА СИСТЕМ ЗА МЕНАЏИРАЊЕ НА ЕЛЕКТРИЧНА ЕНЕРГИЈА

Ментор:
проф. д-р Живко Коколански

Изработил:
Теодора Шарламанова, индекс бр. 145/2019
e-mail: teasharlamanova@icloud.com

Скопје, октомври 2023

Содржина

Листа на слики

Листа на табели

Листа на акроними

RPi	Raspberry Pi
USB	Universal Serial Bus
HDMI	High-Definition Multimedia Interface
SD Card	Secure Digital Card
MQTT	Message Queue Telemetry Transport
IoT	Internet of Things
AMI	Advanced Metering Infrastructure
HTTP	Hypertext Transfer Protocol
M2M	Machine-to-Machine
SMTP	Simple Mail Transfer Protocol
UID	Unique Identifier
ToU	Time of Use

Апстракт

Со постојаното зголемување на побарувачката за енергија и сè поголемата загриженост за одржливоста на животната средина, огромна е потребата за ефикасни решенија за управување со енергијата. Во овој дипломски труд се истражува примената на така наречени паметни мерила на енергија базирани на IoT(интернет на нештата) системи како сигурен пристап кон справувањето со овие предизвици. IoT технололгијата овозможува следење и контрола на потрошувачката на енергија во реално време, обезбедувајќи вредности сознанија за оптимизирање на користењето на енергијата, намалување на потрошувачката и подобрување на севкупната енергетска ефикасност. Главната цел на овој систем кој го дизајниравме и имплементираме е автоматското отчитување на податоци од енергетско мерило и нивно прикажување на IoT платформа, која овозможува понатамошна анализа и контрола над системот.

Клучни зборови: IoT, Raspberry Pi, OpenRemote, Linux, Python, мерни инструменти, мерило, интелегентно енергетско мерило, напредна мерна инфраструктура

Вовед

Брзиот раст на побарувачката за енергија и потребата за одржливо управување со енергијата доведоа до развој на иновативни технологии како што се паметни мерила на енергија базирани на *IoT*. Овој дипломски труд го претставува дизајнот, програмирањето и интеграцијата на паметен систем за мерење на енергија, користејќи принципи на *IoT* и *Raspberry Pi* како централна платформа.

Примарната цел на ова истражување е да се создаде функционален паметен систем за мерило на енергија кој нуди можности за следење во реално време, собирање податоци и далечинско управување за да се подобри енергетската ефикасност и да се овозможи информирано донесување одлуки за потрошувачите. Архитектурата на системот вклучува мерило на електрична енергија, *Raspberry Pi* како основа и веб-интерфејс лесен за корисникот за визуелизација и контрола на податоците.

Процесот на развој вклучува избор на хардвер, кој опфаќа соодветно мерило способно за прецизно мерење на потрошувачката на електрична енергија. *Raspberry Pi* служи како централен центар, одговорен за агрегација на податоци и пренос до облакот. Системот е програмиран со помош на *Python,* овозможувајќи беспрекорна комуникација помеѓу мерилото и *Raspberry Pi.*

Интеграцијата на протоколот MQTT (*Message Queuing Telemetry Transport*) обезбедува безбеден и ефикасен пренос на податоци на платформата во облак, овозможувајќи им на корисниците пристап до податоците за потрошувачката на енергија во реално време преку веб-интерфејсот.

Резултатите ја демонстрираат успешната имплементација на системот за мерење на енергија базиран на *IoT* со помош на *Raspberry Pi,* покажувајќи го неговиот потенцијал да ги револуционизира практиките за управување со енергија. Системот ги овластува потрошувачите да ја следат нивната употреба на енергија во реално време, овозможувајќи им да усвојат навики за заштеда на енергија и да придонесат за одржливост на животната средина.

1 Паметни мерила за енергија

Паметните мерила за енергија, познати и како напредна мерна инфраструктура(AMI) или интелегентни енергетски мерила, се модерни електронски уреди дизајнирани да ја мерат и прикачуваат потрошувачката на електрична енергија во реално време или во чести интервали. За разлика од традиционалните аналогни мерила, паметните мерила за енергија се опремени со напредни комуникациски способности, овозможувајќи двонасочна комуникација помеѓу мерилото и централниот систем на корисникот.

Неколку од клучните карактеристики и придобивки од паметните мерила на енергија се:

1. **Следење во реално време.** Паметните мерачи на енергија обезбедуваат податоци во реално време за потрошувачката на електрична енергија. Корисниците можат да пристапат до овие информации преку различни средства, како што се веб-портали, мобилни апликации или прикази во домот. Ова им овозможува на потрошувачите да ја следат нивната употреба на енергија и да донесуваат информирани одлуки за поефикасно управување со потрошувачката.
2. **Двонасочна комуникација.** Паметните мерила на енергија можат да испраќаат и примаат податоци, овозможувајќи двонасочна комуникација. Ова го олеснува читањето на броилата од далечина, автоматското наплатување и можноста за спроведување стратегии за одговор на побарувачката за време на врвната потрошувачка.
3. **Тарифи за време на употреба (*ToU*).** Со паметни броила, добавувачите на енергија можат да имплементираат тарифи за време на употреба. Потрошувачите можат да ја прилагодат нивната потрошувачка на енергија за да ги искористат предностите на пониските стапки за време на часовите надвор од врвната потрошувачка, што може да доведе до заштеда на трошоците.
4. **Подобрена точност.** Паметните мерила на енергија се поточни од традиционалните броила. Тие можат поточно да ја мерат потрошувачката на енергија, намалувајќи ги грешките во фактурирањето и споровите меѓу потрошувачите и комуналните претпријатија.
5. **Далечинско исклучување и повторно поврзување.** Добавувачите на енергија можат далечински да го поврзат или исклучат снабдувањето со електрична енергија преку паметни броила за енергија. Оваа функција го рационализира процесот на справување со барањата за услуги и ја елиминира потребата за физички посети на просториите на потрошувачите.
6. **Анализа на податоци.** Податоците собрани од паметните мерила на енергија може да се анализираат за да се идентификуваат моделите на потрошувачка, да се детектираат аномалии и да се оптимизира дистрибуцијата и производството на енергија. Овие информации се вредни за комуналните претпријатија за поефикасно планирање и управување со нивните мрежи.
7. **Откривање и обновување на дефекти.** Паметните мерила на енергија можат да детектираат прекини на струја и да обезбедат податоци во реално време до

добавувачите на енергија, забрзувајќи ги процесите за откривање и обновување на дефекти.

Севкупно, паметните мерила на енергија играат клучна улога во модернизацијата на енергетската инфраструктура, оптимизирањето на користењето на енергијата и промовирањето на одржливи енергетски практики. Со овозможување поефикасно управување со енергијата и поттикнување на поблиски односи меѓу потрошувачите и добавувачите на енергија, паметните мерила на енергија придонесуваат за попаметен, поеколошки и поотпорен енергетски екосистем.

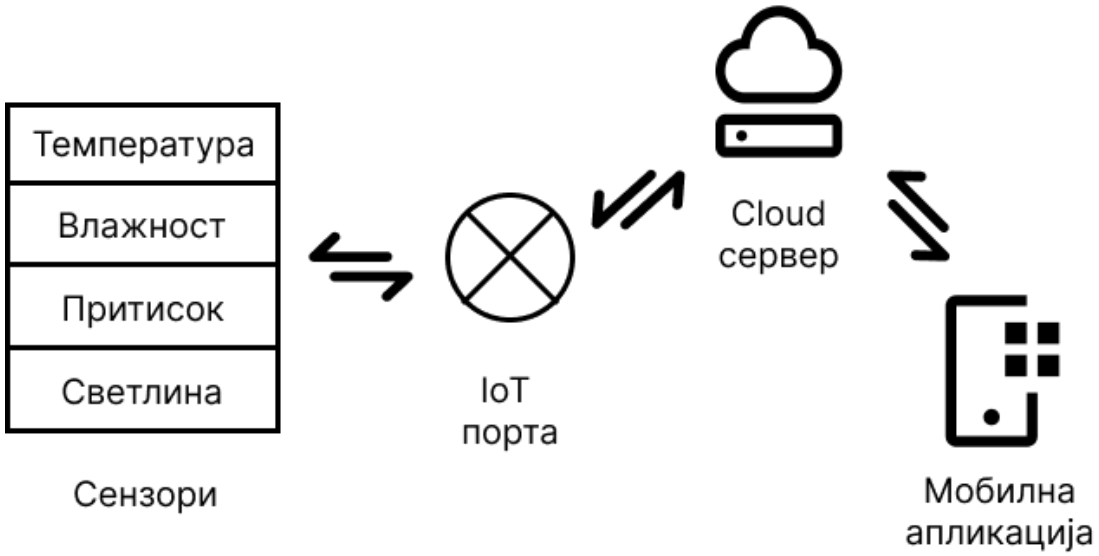
2 IoT – Интернет на нештата

Во денешно време, интернет-базираната информациската архитектура овозможува размена на услуги и добра помеѓу сите елементи, опрема и објекти поврзани на мрежата. IoT се однесува на мрежно поврзување на оние секојдневни објекти, кои често се опремени со некој вид на интелигенција. Во овој контекст, интернетот може да биде и платформа за уредите да комуницираат електронски и да споделуваат информации и конкретни податоци со светот околу нив. Дел од карактеристиките на оваа архитектура може да се видат на слика 1. Во најголем дел, интернетот се користеше за апликативни протоколи ориентирани кон поврзување како што се HTTP (Протокол за пренос на хипертекст) и SMTP (протокол за пренос на едноставна пошта). Меѓутоа, во денешно време голем број паметни уреди комуницираат меѓу себе и со други контролни системи. Овој концепт е познат како M2M *(Machine-to-Machine communications).*

Слика 1. Карактеристики на IoT мрежата

Интернетот на нештата, или IoT, е систем на меѓусебно поврзани компјутерски уреди, механички и дигитални машини, предмети, животни или луѓе кои се обезбедени со единствени идентификатори (UID) и можност за пренос на податоци преку мрежа.

Најдобри примери на IoT се поврзани паметен град, паметна индустрија, паметен транспорт, паметни згради, паметна енергија, паметно производство, паметно следење на околината, паметно живеење, паметно здравје, паметно следење на храна и вода. Слика 2 ја прикажува архитектурата на мрежата IoT. Оваа архитектура има многу IoT сензори за сензорни цели како температура, влажност, притисок итн. По сензорирањето, овие податоците се пренесуваат на облак сервер преку *IoT gateway*(порта). Понатаму, корисниците можат да пристапат до овие податоци преку мобилни апликации и така натаму [1].



Слика 2. Архитектура на IoT мрежата

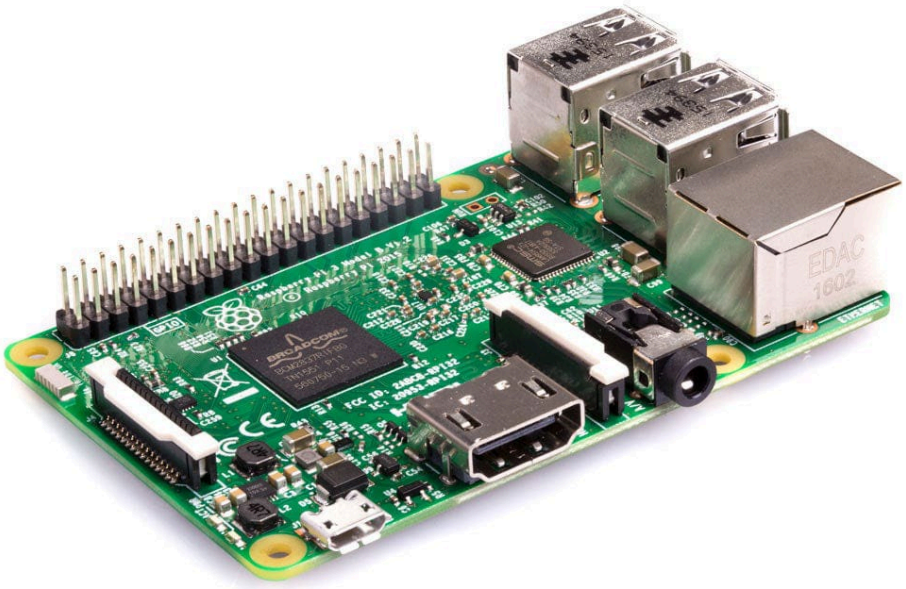
3 Хардверска инфраструктура

3.1 Процесен компјутер (Raspberry Pi)

Raspberry Pi е еден вид на компјутери со една плоча (SBC – single-board computers), што значи дека целиот негов хардвер е поставен на една електронска плочка.

Raspberry Pi плочата содржи ARM процесор базиран на *Broadcom*, графички чип, RAM, општо наменски влезно/излезен конектор и други конектори за надворешни уреди, како неопходни се тастатура и глушец преку *USB* приклучок, монитор преку HDMI приклучок, напојување micro-USB или USB-C тип. Процедурата на работење на *RPi* е многу слична во споредба со стандардниот персонален компјутер. Оперативниот систем се инсталира на *SD*-картичка и се заснова на *Linux*, кој е систем со отворен код.

Сè на сè, тоа е способен мал компјутер кој може да се користи за истите задачи што може да се направат на стандардниот персонален компјутер [3](*Raspberry Pi Foundation* [Реф. 27.7.2023]).



Слика 3. Raspberry Pi 3

RPi е релативно евтин, со почетна цена од 35$. Постојат неколку генерации на *RPi*: од Pi 1 до 4, па дури и Pi 400. Генерално од повеќето генерации постојат модел А и модел Б. Моделот А е поевтина варијанта, но има тенденција да има помалку RAM меморија и помал број на порти како USB и Етернет.

Досегашните модели на Raspberry Pi се [4]:

- *Pi 1 Model B (2012)*
- *Pi 1 Model A (2013)*
- *Pi 1 Model B+ (2014)*
- *Pi 1 Model A+ (2014)*
- *Pi 2 Model B (2015)*
- *Pi Zero (2015)*
- *Pi 3 Model B (2016)*
- *Pi Zero W (2017)*
- *Pi 3 Model B+ (2018)*
- *Pi 3 Model A+ (2019)*
- *Pi 4 Model A (2019)*
- *Pi 4 Model B (2020)*
- *Pi 400 (2021)*

3.1.1 Поставување / Setup

Како што е споменато погоре, за работа со *Raspberry Pi* е потребно да се приклучат неопходните уреди како тастатура, глушец, монитор, напојување, SD-картичка. SD-картичката го содржи оперативниот систем кој што овозможува на компјутерот да работи. Поставувањето на оперативниот систем е доста поразличен од повеќето компјутери.

Оперативниот систем може да се превземе од официјалната страница на *RPi* или со користење на *Raspberry Pi Imager* кој покрај симнување на оперативниот систем, врши и инсталација на истиот на SD-картичката.

Raspberry Pi Imager

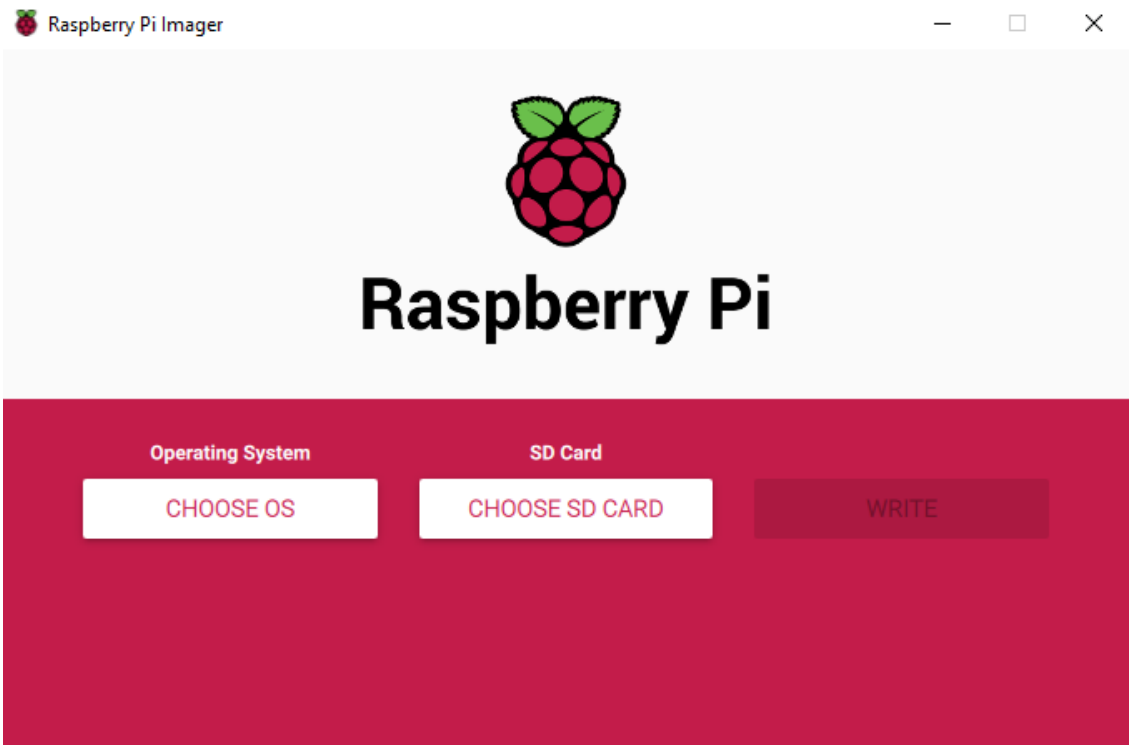
Raspberry Pi Imager е официјална софтверска алатка обезбедена од *Raspberry Pi Foundation*, слика 4, која е дизајнирана да го поедностави процесот на инсталирање оперативни системи на микрокомпјутерите *RPi. Raspberry Pi Imager* е достапен за *Windows, macOS* и *Linux* оперативни системи.

Raspberry Pi Imager поддржува широк опсег на оперативни системи, вклучувајќи го и *Raspbian*(официјален *Raspberry Pi* ОС), *Ubuntu, Debian* и други дистрибуции поддржани од заедницата.

Листа од неколку популарни и поддржани оперативни системи:

- *Raspbian*
- *Windows IoT Core*
- *Ubuntu Desktop*
- *Ubuntu Core*
- *Linutop*
- *Ubuntu Mate*
- *Arch Linux ARM*
- *Kali Linux*
- *Fedora*
- *Tiny Core*

Подетално за начинот и чекорите на поставување на оперативен систем на *RPi* може да најдете на www.raspberrypi.org/downloads.



Слика 4. Raspberry Pi Imager

3.2 Sommy GQ-12

Трифазното мерило на енергија - *Sommy GQ-12*, прикажан на слика 5, овозможува да се измерат и прикажат трифазни електрични параметри како трифазен напон, струја, активна моќност, реактивна моќност, фактор на моќност, фреквенција, привидна моќност, потрошувачка на енергија во KWh и друго. Една од поважните карактеристики за овој мерач е можноста за комуникација преку RS485 сериски стандард кој се заснова врз MODBUS протокол. Техничките спецификации на мерниот инструмент кој се користи во овој проект се дадени во табела 1.

Слика 5. Трифазно мерило на енергија – Sommy GQ-12

Табела 1. Технички спецификации за мерен инструмент Sommy GQ-12

Ставка	Спецификација
Точност	Активна енергија: 0.5S, Реактивна енергија: 1.0S
Номинален напон	3x120V/208V, 3x240V/415V
Номинална струја	1.5(6)A
Мерна мрежа	3 фази 4 жици
Номинална фреквенција	50/60Hz
Работен напон	Нормален работен напон: 0.9Un-1.1Un, граница на работен напон: 0.7Un-1.2Un
Потрошувачка на енергија	Напонско коло< 5VA/phase; Струјно коло< 4VA/phase
Комуникација	RS485, MODBUS-RTU или DL/T645-2007
Температурен ранг	Нормална работа температура: -10~+45°C Граница на работна температура: -20~+55°C Температура на складирање: -40~+70°C
Релативна влажност	≤95%RH
Димензии(mm)	126 length x 88 width x 65 height
Извор: Sommy GQ12 Series Rail Mounting 3Phase Energy Meter Manual - 2014	

3.3 Поврзување

Како прв чекор во поврзувањето е потребно на мерачот GQ12 да се провери местото за поврзување на RS485. Некои мерачи од оваа серија со еден или со два конектори. Тоа е сет од 3 терминали означени како „A+“, „B-“ И „GND“, од типот на завртки за брзо и лесно прикачување на комуникациските жици.

Потоа е потребно да се снабдиме со RS485 конвертер на USB што би бил компатибилен со оперативниот систем на кој работиме. Постојат доста голем број на варијанти на ваквиот конвертер, и треба внимателно да се избере во согласност со нашите потреби.

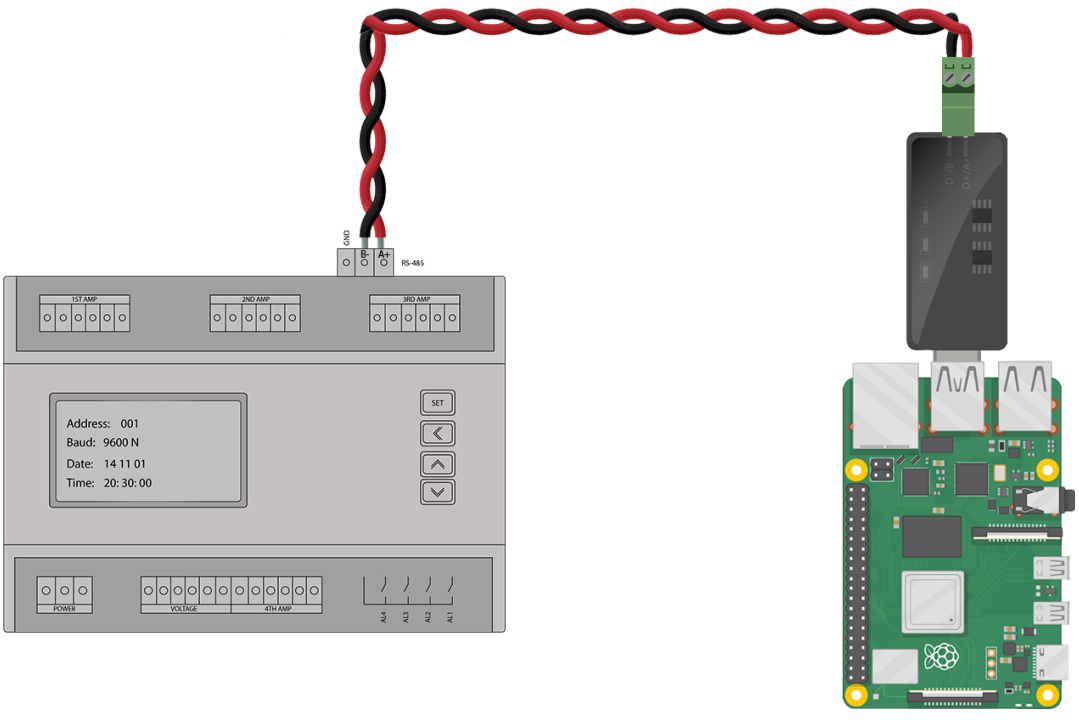
Доколку е потребно, најчесто на *Windows* оперативен систем, да се инсталираат потребните драјвери.

Потоа се врши поврзување на крајот од конвертотор со терминалите од мерачот, со двожична или повеќежична врска.

Во нашиот случај, како што е прикажано на слика 6, вршиме двожично поврзување на конвертотот со мерачот, така што:

- Го поврзуваме „A+“ терминалот од мерачот со една жица од кабелот
- Со другиот крај од истата жица го поврзуваме „A+ / DATA+“ излезот од конверторот
- Со едниот крај од другата жица го поврзуваме терминалот „B-“ од мерачот
- Па со другиот крај од таа жица го поврзуваме „B- / DATA-“ излезот од конверторот

Како е извршено поврзувањето може да се види и на слика 6 подолу.



Слика 6. Поврзување на елементите

За RS485 комуникација, се препорчува да се користи кабел со усукана парица. Тој е специјално дизајниран, составен од две изолирани проводници извртени една околу друга, а таквото извртување помага да се отстранат електромагнетните пречки, што е од суштинско значење за сигурна комуникација на подолги растојанија и во средини со електричен шум.

По завршеното поврзување, може USB завршетоткот да го приклучиме во компјутерот. Потребно е да ја провериме портата COM(на *Windows*) или tty уредот (на *Linux*) на која е доделен конверторот. Овие информации се од големо значење за понатамошно сетирање/поставување во *Python* кодот.

Уште неколку информации потребни се комуникациските параметри како:

- Baud rate
- Data bits
- Stop bits
- Parity

Кои во нашиот случај се дифолтните вредности поставени од самото броило. (baudrate =9600, parity=None, databits=8, stopbits=1)

3.4 Комуникациски протокол

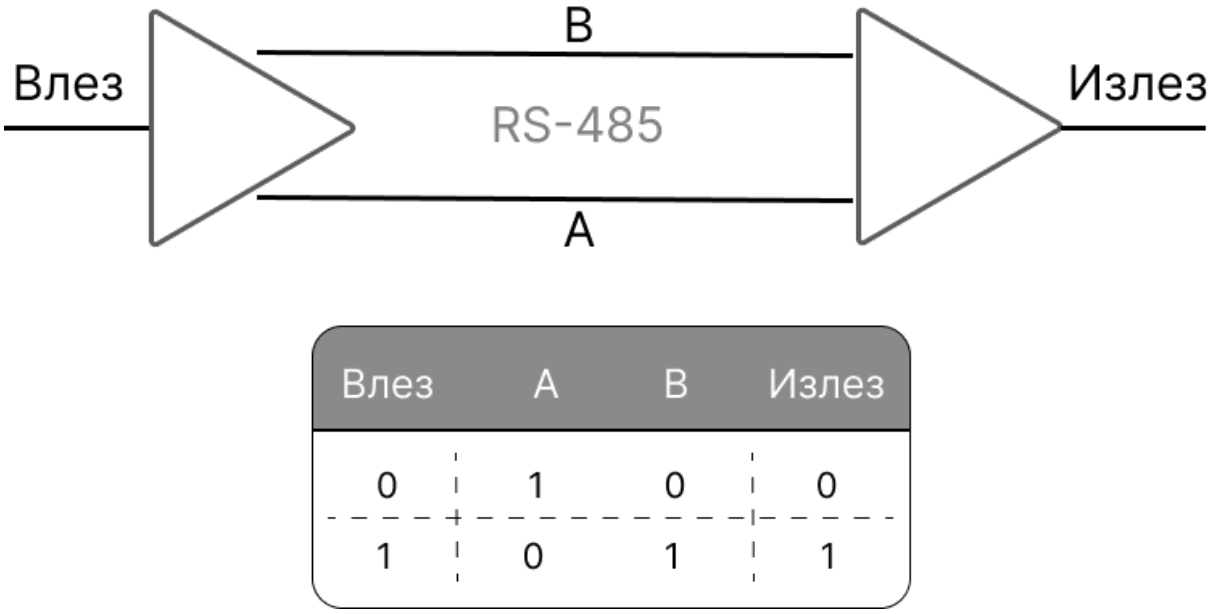
3.4.1 RS-485 комуникациски интерфејс

RS-485 е комуникациски стандард кој е доста користен во апликациите за прибирање(аквизација) и контрола на податоци. Една од поглавните карактеристики негови е тоа што овозможува да се постават неколку уреди кои треба да комуницираат на иста магистрала, а со тоа овозможува повеќе јазли да се поврзат едни со други.

Сериското RS-485 поврзување се врши со помош на кабел од две или три жици: жица за податоци, жица со превртени податоци, и некогаш жица за заземјување.

Основниот принцип на RS-485 интерфејсот е диференцијален (балансиран) пренос на податоци, преку примопредаватели поврзани со две жици (плетена парница). Тоа значи дека еден сигнал се транспортира преку две жици, едната од парот го пренесува оригиналниот сигнал, а другата ја пренесува својата копија, сликовито претставено на слика 7.

Како резултат на диференцијалниот пренос на сигнал секогаш постои потенцијална разлика помеѓу жиците. Тоа обезбедува висока отпорност на пречки и шумови. Плетената парница може да биде изолирана или неизолирана, а сето тоа овозможува пренос на податоци на долги растојанија со релативно големи брзини.



Слика 7. Сигнална линија на RS-485 комуникациски интерфејс

3.4.2 MODBUS

MODBUS е сериски комуникациски протокол кој е широко користен од индустриски електронски уреди

Протоколот MODBUS го дефинира форматот за барањето на господарот(хостот/master) и одговорот на роб(уредот/slave). Барањето ја содржи адресата на уредот, функционален код кој го дефинира бараното дејство, сите податоци што треба да се испратат и поле за проверка на грешки. Одговорот содржи полиња кои го потврдуваат преземеното дејство, сите податоци што треба да се вратат и поле за проверка на грешки. Ако се појави грешка при приемот на пораката или ако роб-уредот не може да го изврши бараното дејство, уредот конструира порака за грешка и ја испраќа како одговор.

GQ12 користи MODBUS RTU комуникациски протокол, RS485 полу-дуплекс комуникација, 16-цифрена CRC проверка.

Да ја разгледаме конкретно Host request рамката обезбедена од производителите на *Sommy GQ-12* прикажана на слика 8.

Host request (read multi-register)							
1	2	3	4	5	6	7	8
Meter address	Function code	Start address high bit	Start address low bit	data byte length high bit	data byte length low bit	CRC code low bit	CRC code high bit
0x01	0x03	0x01	0x6E	0x00	0x02	0xA4	0x2A

Слика 8. MODBUS рамка за барање (Sommy GQ-12 User Manual)

Рамката за барање се испраќа од главниот уред (како што е PLC или компјутер) за да побара податоци или да иницира дејство од slave уред (сензор, контролер или друг теренски уред).

Рамката се состои од:

Адреса. Во комуникацијата Modbus, секој уред на мрежата има единствена адреса. Ова поле одредува за кој уред е наменета пораката. Во случајот на Modbus RTU, тоа е обично 8-битна адреса, додека во Modbus ASCII, таа е претставена како два ASCII знаци.

***Функциски код*.** Ова поле го одредува типот на дејство што треба да го изврши уредот што прима. Тој дефинира дали пораката е барање за читање, барање за пишување или некој друг тип на команда.

***Почетна адреса*.** Адреса на регистерот кој го читаме, претставен преку две полиња, high bit и low bit.

***Податоци*.** Информација за должината и типот на податоци.

***Проверка на грешки*.** За да се обезбеди интегритет на пораката, рамките на Modbus често вклучуваат некоја форма на проверка на грешки, како што е циклична проверка на вишок (CRC) или контролна сума. Ова му овозможува на уредот што прима да открие дали се појавиле грешки при преносот.

Рамката за одговор се испраќа од slave-уредот како одговор на рамката за барање од хостот. Ги содржи бараните податоци или потврда за извршеното дејство.

На слика 9, може да се види и рамката за одговори, која се разликува од рамката за барања во тоа што не ги содржи почетните адреси на регистрите туку во себе ги содржи само податоците и нивната големина.

Slave normal answers (Read multi-register)								
1	2	3	4	5	6	7	10	11
Meter address	Function code	data byte number	Data 1 high bit	Data 1 low bit	Data 2 high bit	Data 2 low bit	CRC code low bit	CRC code high bit
0x01	0x03	0x04	0x00	0x21	0x91	0xC0	0xC7	0xF9

Слика 9. MODBUS рамка за одговор (Sommy GQ-12 User Manual)

4 Платформа за менаџирање со енергија

4.1 Софтверска платформа OpenRemote

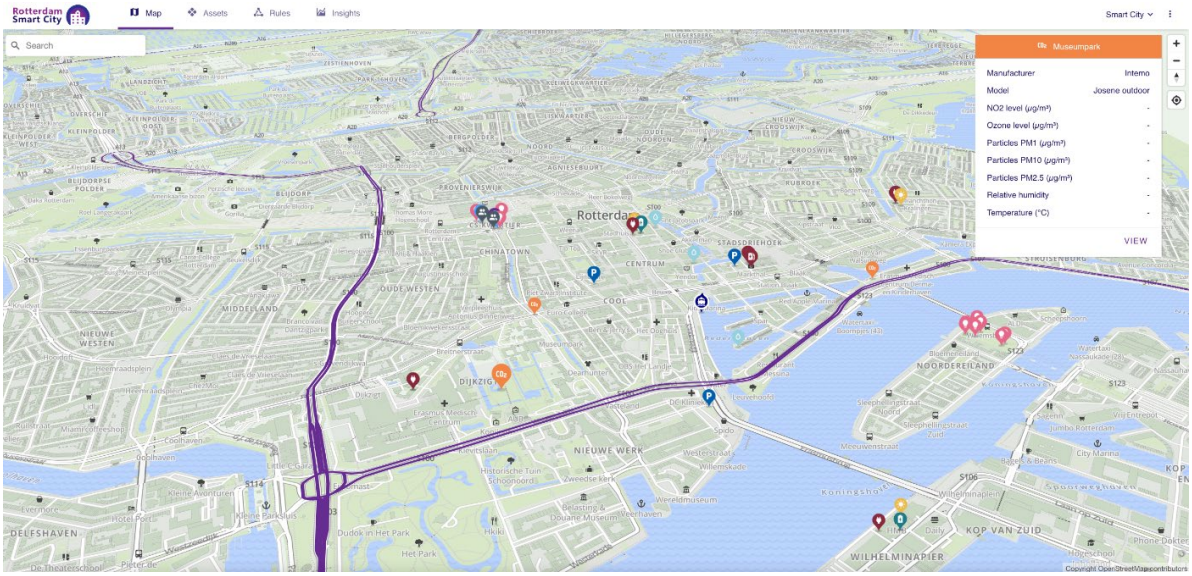
OpenRemote е интуитивна платформа за Интернет на нештата (IoT – Internet of Things) со отворен код (open source) што ги вклучува сите функции и карактеристики, од поврзување уреди до градење на паметни апликации како паметни домови и автоматизација на паметни градови [7].

Платформата интегрира многу различни протоколи и нуди визуелизација. Неколку од функциите кои ги нуди платформата се:

- Управување со средства(assets management) кои може целосно да се приспособат, да се визуелизираат на мапи и страници, како и да се складираат или користат за правила и контролни табли.
- Овозможува да се врши географски преглед на средствата и нивните моментални вредности. Овозможува да се измени картата се со цел да одговара на областа на интерес, а може да се искористи и геоградувањето за да се активираат аларми или да се испраќаат известувања.
- На страницата за увид (insights page), било кој извор на податоци или уред може да се споредат во однос на времето или едни со други.
- Овозможува да се дефинира совршен дигитален близнак на уредите или физичката средина. Се конфигурираат сопствени типови на средства, атрибути и однесување за да се направи апликацијата да одговара на реалноста.
- Голем број корисници може да добијат пристап до средствата и опкружувањата, со различни улоги и ограничувања.
- Постојат протоколни агенти кои поврзуваат надворешни уреди и услуги. Вклучени се генерички протоколи како HTTP, SNMP, MQTT, Bluetooth, Serial, TCP, UDP, Z-wave, како и специфични како KNX и Velbus.
- Со „кога-тогаш“ правилата може да се следи кој било атрибут или временска состојба и да се активира некое дејство или известување
- Со правилата за проток(flow rules) може да се обработуваат и манипулираат податоците за да се создадат нови податоци или атрибути. Односно може да се прават конверзии на дадени податоци, комбинирање на повеќе атрибути во еден нов.
- Groovy е јазик за скриптирање кој овозможува дефинирање на покомплексна логика на автоматизација.Тоа дава слодоба да може да се дефинира речиси се во системот што работи, на пр. Да се дефинира некое групно однесување на дадени уреди или некоја состојба на алармирање.
- Користејќи ги поставките за изглед, може лесно да се промени логото и стилот на бојата за секоја област. Да се постават ставките за навигацијата, промени областа на разгледување,односно да се промени картата, стандардните јазици и иконите кои се прикажуваат на картата.
- Овозможува инсталација, следење и одржување на компонентите, со пристап до истите преку веб-апликација или преку мобилна апликација.

Во изработка на овој проект е користен *OpenRemote v3* исто така познат и како *Manager 3.0*. Менаџерот 3.0 е создаден за да се справи со поголеми апликации со повеќе корисници. Вклучува генеричка структура на средства и атрибути, дозволувајќи им на корисниците да градат свои типови средства и да додаваат свои протоколи и да

користат генерички протоколи како HTTP и Bluetooth, или постари протоколи како KNX или BACnet.

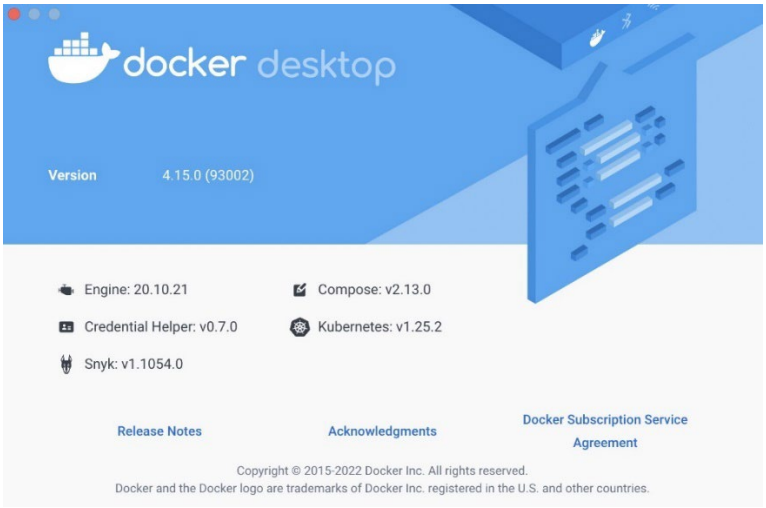


Слика 10. Програмски пакет OpenRemote v3 – демо верзија

Со помош на *OpenRemote v3* изградено е комплетно IoT решение за управување и собирање податоци од *Sommy GQ12* мерач за електрична енергија. Со помош на MQTT протоколниот агент кој е веќе вклучен во самата платформа, ги отчитавме податоците од мерачот. Таквите податоци собрани, може подоцна да се обработат, прикажат на карта, аналитички да се обработат.

4.1.1 Docker Desktop

Docker Desktop апликација која овозможува да се креираат и споделуваат апликации и микросервиси со контејнери. Овозможува директен GUI(Графички кориснички интерфејс) кој овозможува да се управува со контејнери, апликации и слики директно од локалната машина.

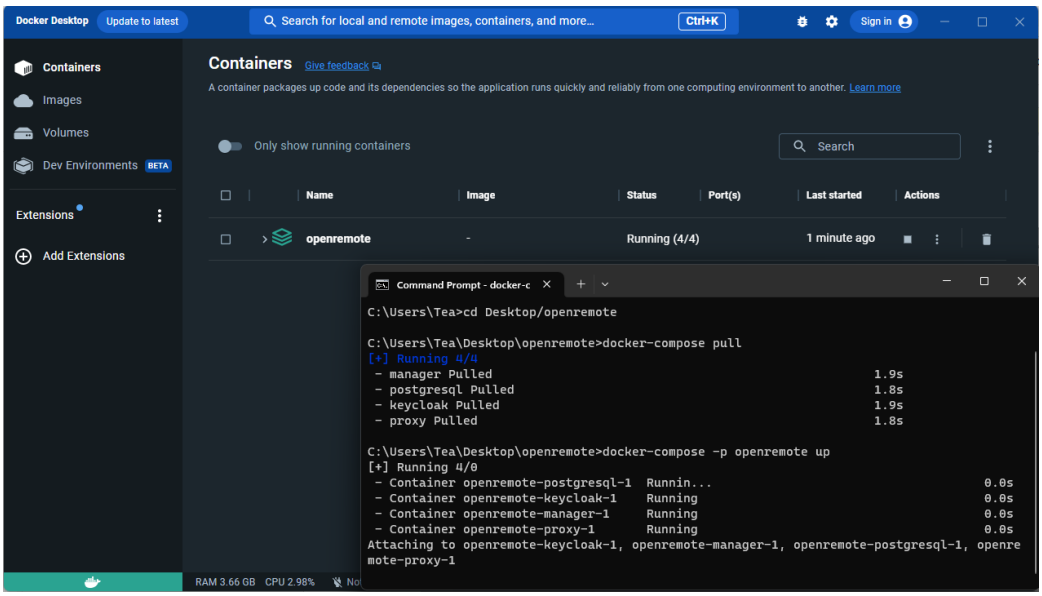


Слика 11. Docker Desktop

Docker Desktop се грижи за мапирањето на портите, проблемите на датотечниот систем и другите стандарди поставки и редовно се ажурира со поправки на грешки и безбедносни ажурирања.

4.1.2 OpenRemote и DockerDesktop

OpenRemote како платформа работи во *Docker* контејнери па затоа, до истата може да се пристапи од секој хардвер и оперативен систем, независно дали пристапуваме локално или на хостиран сервер. Како што е спомнато погоре во текстот, за целите на овој проект се користи *OpenRemote v3*, која функционира само користејќи го *Docker Desktop* како хост. За разлика, постарите верзии на *OpenRemote*, работеле само со *Docker Engine*.

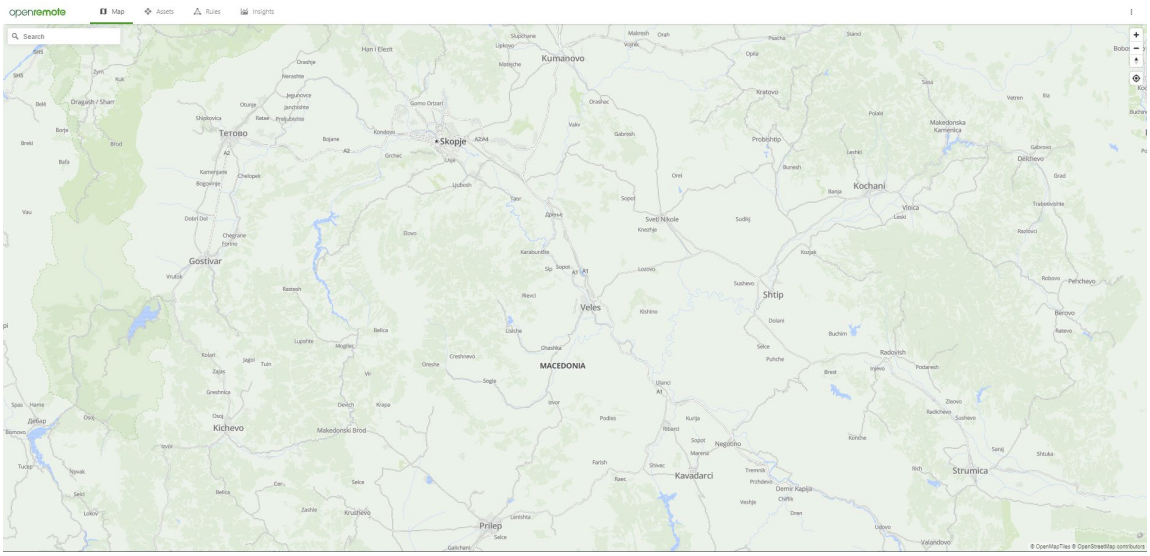


Слика 12. OpenRemote контејнери пристапени преку Docker Desktop

Docker Desktop може да се инсталира на *Linux* оперативен систем, но од серијата на *Ubuntu, Debian* или *Fedora*. За успешно инсталирање на *Docker Desktop*, оперативниот систем *Linux* и компјутерот, мора да се обезбедени 64-битен кернел, процесорска поддршка за визуелизација и најмалку 4GB *RAM*.

4.1.3 White Labeling

При инсталацијата на *OpenRemote*, предефинирано, добиваме целосен пристап до проекти за менаџирање на предели во Ротердам, Холандија, како што прикажавме погоре на слика 10. Доколку сакате да ја модифицираме и проширите областа на OpenRemote согласно на нашите потреби, односно да се постави сопствено лого, друг дел од мапа, област може да се изврши процес на т.н. white labeling каде може да се промени картата и координатите над кои треба да се поставуваат и управуваат новите проекти.



Слика 13. OpenRemote менаџерот по извршениот процес на white labeling

Накратко објаснување за тоа како се прави тој white labeling процес:

Потребно е во фолдерот креиран при самата инсталација на *OpenRemote*, каде е симнат docker-compose.yml датотеката, да се креира нов фолдер кој би го именувале deployment. Во него се креираат два нови фолдери именувани, едниот manager, а вториот map. Во manager фолдерот се креира нов фолдер именуван app. Доколку сакаме да се приспособи OpenRemote со сопствено име и лого од проектот, тогаш во фолдерот app се креира подфолдер именуван images каде подоцна би ги зачувале нашите логоа и слики. Меѓутоа оваа промена не ја земаме во предвид за овој проект.

Во овој проект вршиме промена на картата, картата од Ротердам, Холандија, ја заменуваме со картата од Скопје, Македонија.

Најпрво го следиме одделот за Downloading maps and extracting smaller tilesets објавен на:

https://github.com/openremote/openremote/wiki/Developer-Guide%3A-Working-on-maps

Од openmaptiles.org избираме регион чија векторска мапа сакаме да ја симнеме, го притискаме копчето Download на OpenStreetMap vector tiles. За да може да се изврши процесот на преземање потребно е да се изврши најава со маил. Може да се искористи google account за полесно.

Датотеката која се симна, ја префрламе во фолдерот кој го креиравме …/deployment/map и вршиме промена на името на датотеката во input.mbtiles. Потоа потребно е во terminal да го завршиме процесот кој е вклучен, доколку сеуште е вклучен OpenRemote, со помош на притискање Ctrl+C на тастатура. Следен чекор кој е потребно да го извршиме е во terminal да се позиционираме во фолдерот кој го креиравме map и да ја испишеме следната команда

```
$ docker run -it -rm -v PATH_TO_INPUT_MB_TILES_DIR:/mapdata nikolaik/python-nodejs:python3.8-nodejs12 bash
```

притоа PATH_TO_INPUT_MB_TILES_DIR го заменуваме со патеката до фолдерот map. (пр. /home/pi/Desktop/openremote/deployment/map)

Потоа ја впишуваме командата

```
$ npm install -g –undafe @mapbox/mbtiles @mapbox/tilelive.
```

Со помош на корисната алатка https://tools.geofabrik.de/calc/ избираме граници на областа на истражување и централна точка. Координатите ги копираме и ги впишуваме во командата

```
$ tilelive-copy –minzoom=0 –maxzoom=14 –bounds=”FILL_BOUNDS_HERE” /mapdata/input.mbtiles / mapdata/output.mbtiles.
```

FILL_BOUNDS_HERE го заменуваме со координатите кои сме ги добиле од Geofabrik калкулаторот. (пр. за Германија командата би го имала овој формат ilelive-copy –minzoom=0 –maxzoom=14 –bounds=”5.53, 47.23, 15.38, 54.96” /mapdata/input.mbtiles / mapdata/output.mbtiles).

Мора да се внимава координатите да се впишани во точен формат, [W-запад,S-југ,E-исток,N-север], меѓусебно одделени со запирка. Оваа команда овозможува да се креира output.mbtiles датотека кој всушност ги содржи информациите потребни за мапата. Ја преименуваме output.mbtiles во mapdata.mbtiles. (https://github.com/openremote/openremote/blob/master/manager/src/map/mapsettings.json) Го симнуваме Raw json кодот за поставките на мапата. Го зачувуваме исто така во фолдерот map. Таквиот документ го отвараме со Notepad, Notepad++ или друг соодветен програм и вршиме измена на координатите за централната точка и граници во трите мапи познати како default, smartcity и manufacturer.

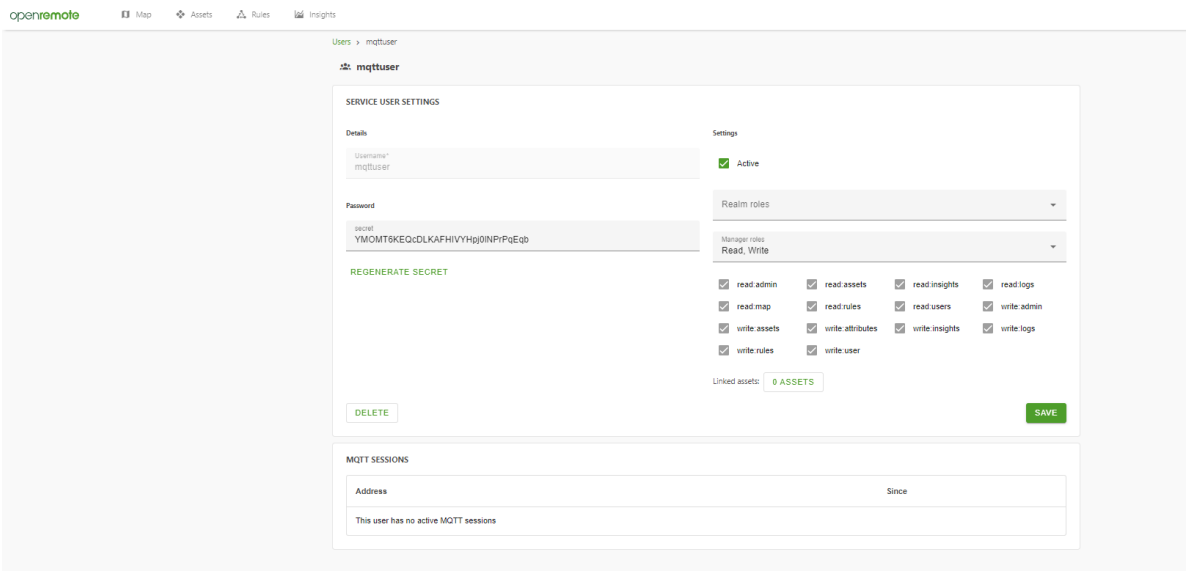
4.1.4 Корисници и средства

4.1.4.1 Креирање на корисник(user)

OpenRemote овозможува да се креираат нови корисници за избраното подрачје и да се постават неговите улоги. Може да се даде целосен пристам, или да се ограничи пристапот за неколку средства. Доколку е ограничен пристапот, дополнително треба да се постави ставката за конфигурација „ Access restricted user read/write “ на секој атрибут до кој сакаме да дадеме пристап на корисникот.

Пример за ограничен пристап е кога поврзуваме станбен комплекс и сакаме на секој од станарите да дадеме пристап само до средствата на неговиот или нејзиниот стан.

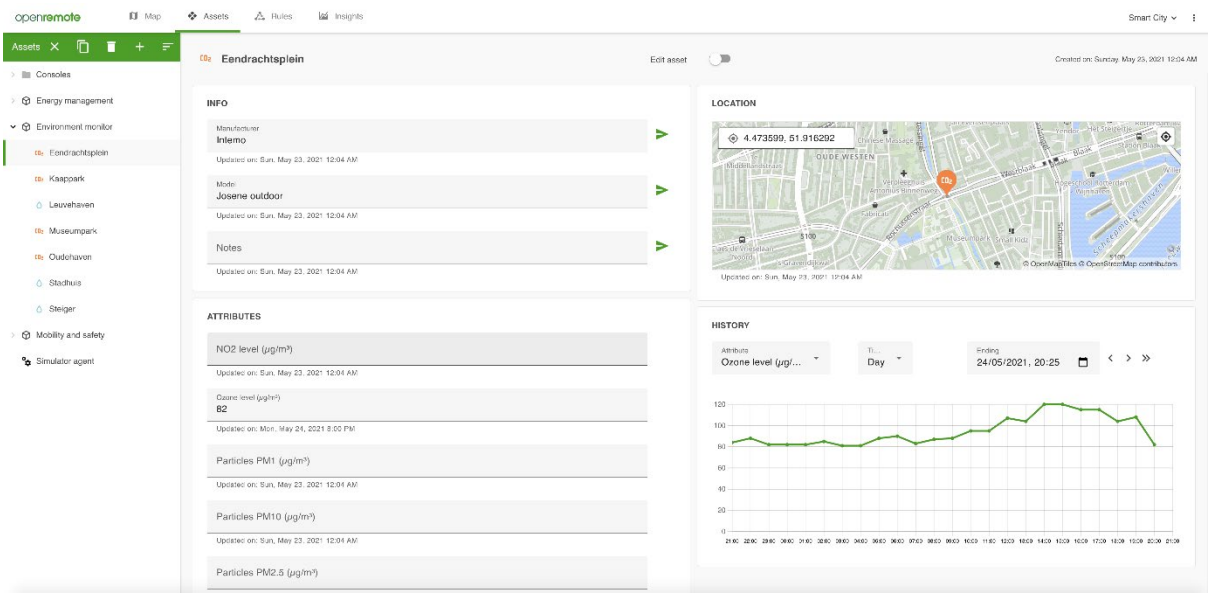
Во овој проект ние креираме корисник со целосен пристап. Го именуваме mqttuser.



Слика 14. Процес на креирање на корисник

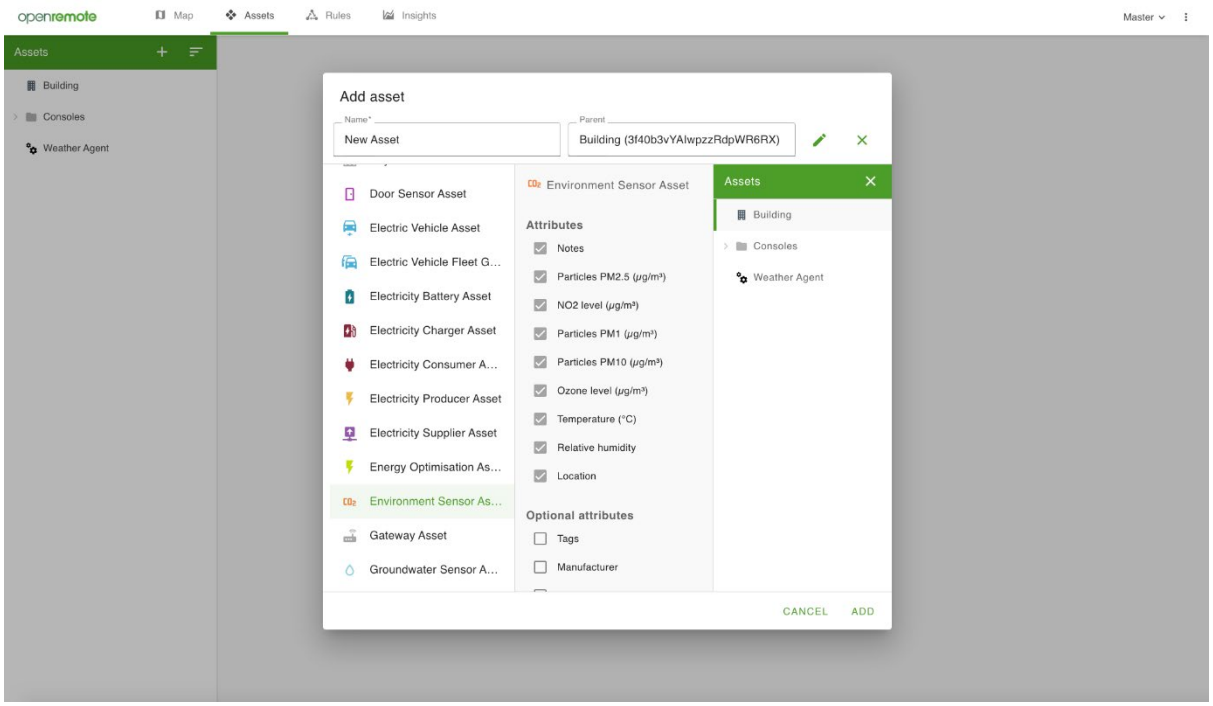
4.1.4.2 Креирање на средства(assets)

Страницата за средства/имоти овозможува преглед и модификација на средствата и нивните атрибути. На левата страна од екранот, слика 15, може да се забележи структурата на дрвото на средства, а деталите може да се видат на десната страна од екранот.



Слика 15. Пример за asset од типот 'environment monitor'

Страницата за средства содржи информациски панел, панел за атрибути, локациски панел и панел за историјата на средствата. Инфо и атрибут панелите даваат преглед на сите атрибути и нивните вредности. За некои атрубити, имаме потреба да ги запишуваме вредностите рачно, а тоа може да го направиме со впишување на вредност во полето и да се притисне стрелката која се наоѓа десно од полето за впишување. Други вредности на атрибутите може да бидат отчитувања во живо од сензори или автоматски ажурирања на вредностите преку некои од правилата.



Слика 16. Пример за креирање на нов asset.

Ново средство може да се креира со кликање на + знакчето во делот за дрво на средства. Тоа отвора прозорче, како на слика 16, кое ги прикажува достапните типови средства. Кога ќе се избере еден, ќе се видат атрибутите. Изборни атрибути може да се додадат со нивно избирање, тука во ова прозорче. Може да се постави или да се промени и родителот на средството.

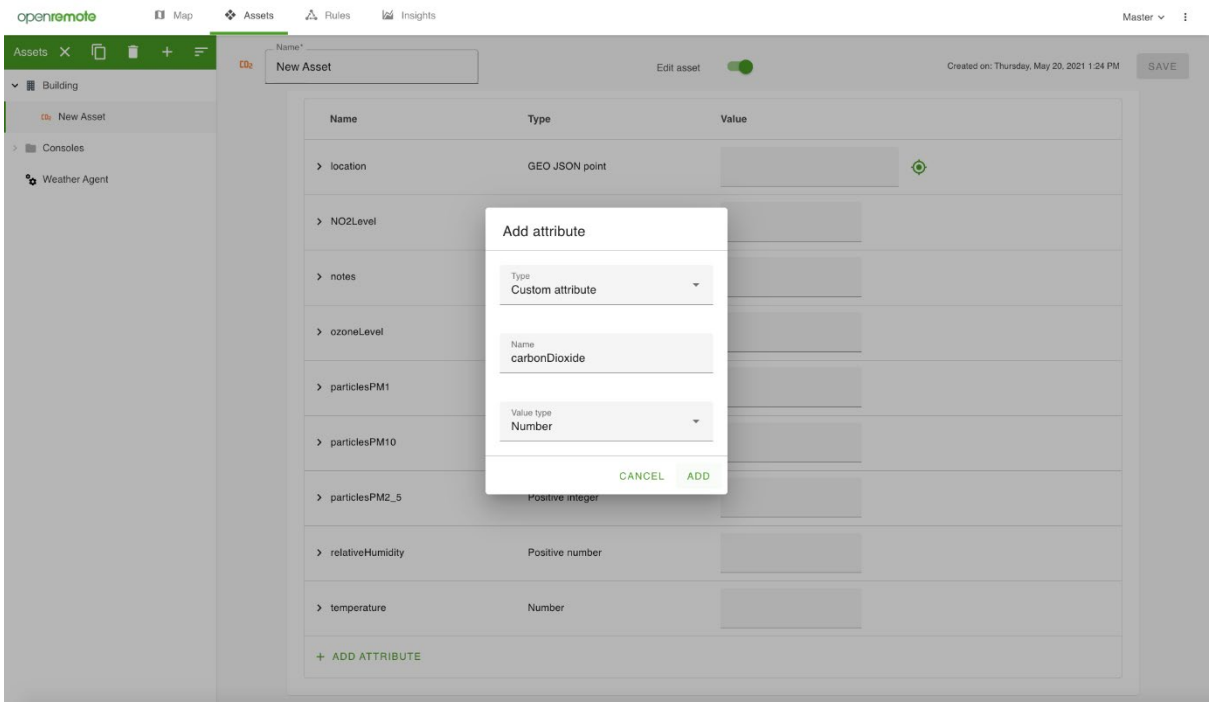
4.1.4.3 Додавање атрибути

Додека сме во Edit Asset модот, може секој од атрибутите да се модифицира, на тој начин што може тој атрибут да складира поранешни вредности и колку долго да складира, да се дадат дозволи за користење на тие податоци во Rules(when-then, flow, groovy, итн.), дали локацијата да се прикаже на картата. Неколку од модификациите кои е овозможено да се направат и нивното значење може да се погледнат во табела 2.

Табела 2. Најкористените ставки за конфигурација на атрибути во OpenRemote

Конфигурација	Опис
Access restricted read	Ограничените корисници можат да читаат ако имаат пристап до атрибутот за читање на средството
Access restricted write	Ограничените корисници можат да пишуваат доколку имаат пристап до атрибутот за пишување до средството
Data points max age days	Временски период за зачувување на податоците
Forecast	Се користи во комбинација со „Зачувај точки на податоци“ за прогнозирање на вредностите, видете во предвидувањето
Label	Додава пријателско име, заменувајќи го стандардното име
Read only	Податоците не може да се пополнат преку интерфејс, само преку агенти или правила
Rule state	Додава атрибут како опција за избор во правилата, од левата страна
Rule reset immediate	Дозволува правилото повторно да се активира веднаш. Може да биде корисен за податоци засновани на настани
Show on dashboard	Се користи во комбинација со „локација“ ќе го прикаже средството на приказот на картата
Store data points	Зачувува точки на податоци во базата на податоци, стандардно за еден месец
Units	Додава единица на вредноста на атрибутот, видете го составот и опциите
Извор: https://github.com/openremote/openremote/wiki/User-Guide:-Manager-UI	

Додавање на нов атрибут се врши во Modify модот, на крајот од листата на веќе постоечки атрибути, постои можност за избирање Add Attribute. Се појавува прозорче каде може да се постави име на атрибутот и тип на вредност која сакаме да се запише во тој атрибут, слика 17.



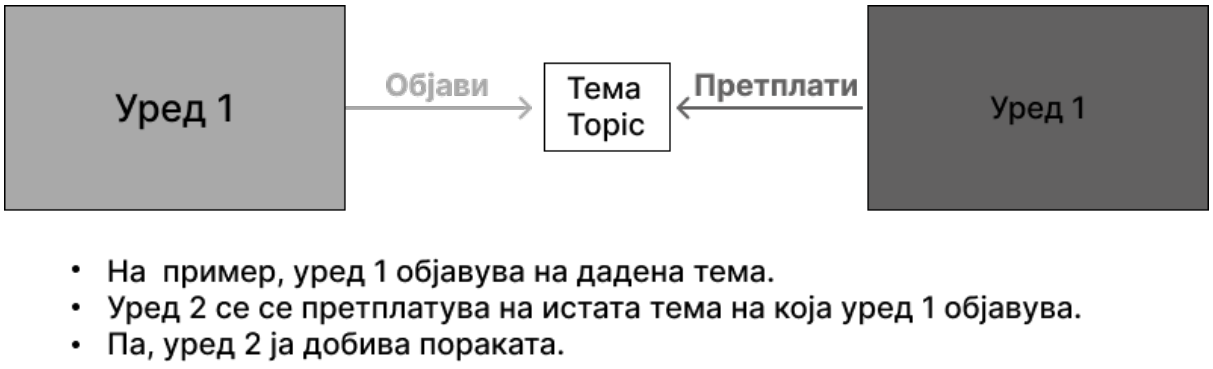
Слика 17. Пример за додавање на нов атрибут

4.2 Python библиотеки

4.2.1 Eclipse Paho (MQTT)

MQTT е кратенка за *Message Queuing Telemetry Transport*, и тоа претставува лесен, отворен протокол за испраќање и преземање пораки, кој е дизајниран за комуникација од машина до машина (M2M) и Интернет на нештата (IoT). Првично е развиен од *IBM* во 1999 година, а подоцна станува отворен стандард кој е одржуван од конзорциумот *OASIS* (Организација за унапредување на структурирани информациски стандарди).

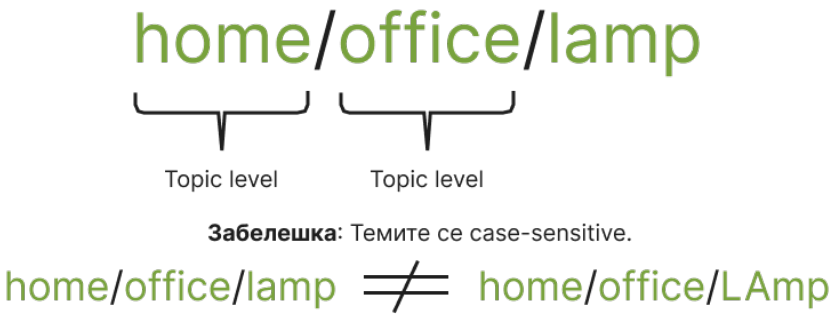
Протоколот MQTT е дизајниран да биде ефикасен и лесен за имплементација, што го прави идеален за употреба во ограничени средини со ограничен опсег, како што се IoT системите. Тој го следи моделот на објави-претплати (publish-subscribe), каде клиентите може да објават пораки до брокер или да се претплатат на одредени теми и да примаат пораки од брокерот.



Слика 18. MQTT комуникациски протокол

Порака во MQTT протоколот е секоја информација која сакаме да ја разменime помеѓу уредите. Може да биде во вид на команда или податоци и вредности добиени од отчитување на некои сензори на пример.

Друг поважен концепт за MQTT протоколот се темите(topics). Темите се начинот на кој може да се регистиратме за дојдовни пораки или да одредиме каде сакаме да ја објавиме пораката.



Слика 19. MQTT Topics

Во овој проект ја користиме “paho-mqtt” библиотеката за *Python*, која всушност претставува официјална клиентска MQTT имплементација, развиена и одржувана од проектот Eclipse Paho. Таа овозможува на апликациите на Python да комуницираат со брокери на MQTT и да учествуваат во објавувањето и превземањето на пораки.

Потребна е инсталација на библиотеката, а тоа може да се направи со помош на pip менаџерот на *Python* пакети. Најпрво треба да бидеме сигурни дека компјутерот располага со веќе инсталиран *Python* и *Python pip*.

Тоа може да го провериме со испишување на командите во терминал

```
$ python3 --version
$ pip3 –version
```

Доколку добиете одговор како

```
$ pip 20.0.2 from /usr/lib/python3/dist-packages/pip (python 3.8)
```

значи дека веќе имате инсталирана верзија на pip.

Доколку немате инсталирано pip, потребно е тоа да го извршите со команда:

```
$ sudo apt-get install python3-pip
```

Потоа продолжуваме со инсталација на paho-mqtt

```
$ pip install paho-mqtt
```

За Ubuntu оперативен систем може да се искористи командата

```
$ sudo apt-get install python3-pip
$ sudo pip3 install paho-mqtt
```

Важно за paho-mqtt е класата client која содржи функции за:

1. поврзување со брокер – .connect()
2. Објавување на пораки на одредена тема - .publish()
3. Претплатување на одредена тема - .subscribe()
4. Неколку функции за повратни пораки од дадени настани - .on_connect(), .on_disconnect(), .on_publish(), .on_subscribe(), .on_unsubscribe()
5. За одржување на комуникацијата со брокерот - .loop_start()
6. Исклучување на клиентот од брокерот - .disconnect()

Подолу во овој труд, во делот Код во точка 4.3, подетално се објаснати секоја од овие функции.

4.2.2 TLS/SSL сертификати

Transport Layer Security(TLS) и неговиот претходник *Secure Sockets Layer*(SSL) се најширок користени безбедносни протоколи денес и првенствено се користат за опслужување две специфични функции[11]:

1. Автентикација и верификација: Сертификатот TLS/SSL има информации за автентичноста на одредени детали
2. Енкрипција на податоците: Чувствителните информации разменети преку веб-локацијата не можат да бидат пресретнати и прочитани од никој друг освен наменетиот примач.

Python обезбедува SSL модул во својата стандардна библиотека, која овозможува работа со SSL/TLS за да се обезедат мрежните врски како правење HTTPS барања, креирање SSL/TLS сервери или при обезбедување на други мрежни протоколи. Сето тоа да се овозможи потребно е да се импортира SSL модулот со впишување на „import ssl“ во Python кодот.

4.2.3 Minimalmodbus

Minimalmodbus е *Python* модул лесен за користење за разговор помеѓу уреди(робови) и компјтер(господар) со помош на MODBUS протоколот и е наменет да работи на главниот компјутер(мастерот). Единствена негова зависност е модулот pySerial, кој се користи во позадина за воспоставување сериска врска меѓу робот и мастерот.

Овој софтвер ги поддржува серриските комуникациски верзии на протоколот „Modbus RTU“, кој ни е потребен нам заради мерачот *Sommy GQ12*, и „Modbus ASCII“. Minimalmodbus е наменет за употреба на *Linux, OS X* и *Windows*, тој е со отворен код и има *Apache* лиценца. Може да се користи со *Python 3.8* и понови верзии.

Се инсталира со впишување во терминал, командата

```
$ pip3 install -U minimalmodbus
$ sudo pip3 install -U minimalmodbus
```

Како што спомнав погоре, зависно е од pySerial модулот, кој е потребно да е верзија 3.0 или понова.

За таа цел, впишуваме

```
$ pip install pyserial
```

Инсталацијата може да се направи и со симнување на minimalmodbus.py датотеката од *GitHub* и поставување на истата во директориумот каде се наоѓа и другиот *Python* код.

Стандардот Modbus го дефинира складирањето во:

- Битови. Секој бит има своја адреса.
- Регистри (16-битни). Секој регистар има своја адреса. Може да држи цели броеви во опсег од 0 до 65535 (дец), што е од 0 до ffff (хексадецимален приказ).

Modbus ги дефинира имињата на „табелите“ во зависност од тоа дали складирањето е во еден бит или во 16-битен регистар и дали е можно да се запише во складиштето, таквите дефинирања се дадени во табела 3.

Табела 3. Modbus “табели”

Складирање во	Пристап	Modbus „табела“	Пример за употреба
Бити	RO (само читање)	Дискретни влезови	Дигитален влез
Бити	RW (читање и запишување)	Намотки	Дигитален излез
16-битен регистар	RO	Влезни регистри	Неколку дигитални влезови, аналоген влез
16-битен регистар	RW	Задржувачки регистри	Неколку дигитални излези, параметар за поставување
Извор: https://minimalmodbus.readthedocs.io/en/stable/			

Функциите што треба да се користитат за читање и пишување регистри и битови од/на Modbus инструментот може да се видат во табела 4.

Табела 4. Имплементирани функции за читање и запишување од/на инструментот

Тип на податоци	Читање	F code (функциски код)	Запишување	F code (функциски код)
Бит	read_bit()	2	write_bit()	5
Бити Симултан пристап	read_bits()	2	write_bits()	15
Регистер integer	read_register()	3	write_register()	16
Long integer (32 или 64 бити = 2 или 4 регистри)	read_long()	3	write_long()	16
Float (32 или 64 бити = 2 или 4 регистри)	read_float()	3	write_float()	16
String 2 карактери по регистар	read_string()	3	write_string()	16
Регистри Integers	read_registers()	3	write_registers()	16
Извор: https://minimalmodbus.readthedocs.io/en/stable/				

Во Modbus RTU, пораката за барање се испраќа од мастерот во овој формат:

- Адреса на уредот [1 бајт]
- Код на функција [1 бајт]. Дозволениот опсег е од 1 до 127 (во децимална).
- Податоци за носивост [0 до 252 бајти]
- CRC [2 бајти]. Тоа е циклична шифра за проверка на вишок, за проверка на грешка на пораката

Одговорот од клиентот е сличен, но со други податоци за носивоста.

4.3 Програмски код за исчитување на паметното броило и ажурирање на OpenRemote платформата

```
import paho.mqtt.client as mqtt
import time
import ssl
import json
import minimalmodbus
```

Import делот од овој *Python* код е местото каде што ги импортираме потребните библиотеки и модули. Овие библиотеки обезбедуваат однапред дефинирани функции и класи кои кодот ќе ги користи за извршување на потребните задачи.

Import paho.mqtt.client as mqtt: со ова го импортираме mqtt модулот од paho бибиотеката која ја објаснивме погоре во текстот. MQTT е лесен протокол за објавување-претплаќање на пораки што често се користи во IoT(Интернет на нештата) апликации

Import time: Временски модул кој обезбедува различни функции за работа со времето. Во овој код ја користиме функцијата time.sleep() за да воведеме временски одложување меѓу одредени операции.

Import ssl: Со овој модул воведуваме шифрирање помеѓу клиентот и MQTT брокерот. Со тоа се обезбедува сигурна комуникација во мрежата.

Import json: Овој модул дозволува кодот да работи со JSON(*JavaScript Object Notation*) податоци. JSON е популарен формат за размена на податоци и во овој код се користи за кодирање и декодирање на JSON податоци при испраќање и примање пораки од/до MQTT брокерот.

Import minimalmodbus: Оваа библиотека се користи за комуникација со уреди преку MODBUS протоколот. Modbus е широко користен комуникацски протокол во индустриските системи за автоматизација и контрола, за читање и запишување податоци од/до уреди како сезори, мерила и контролери.

```
MODBUS_ADDRESS = 1
SERIAL_PORT = '/dev/ttyUSB0'

meter = minimalmodbus.Instrument(SERIAL_PORT, MODBUS_ADDRESS)
meter.serial.baudrate = 9600
meter.serial.bytesize = 8
meter.serial.parity = minimalmodbus.serial.PARITY_NONE
meter.serial.stopbits = 1
meter.serial.timeout = 0.1
```

Во овој дел од кодот вршиме конфигурација на комуникациските пармаметри за мерниот уред *Sommy GQ12.*

Со MODBUS_ADDRESS=1 доделуваме вредност 1 на променливата MODBUS_ADDRESS. Адресата Modbus е целобројна вредност што ја претставува единствената адреса на уредот-роб со кој сакаме да комницираме, во нашиот случај *Sommy GQ12* мерен уред. Адресата на уредот може да се промени според инструкциите дадени во упатството од производителот.

SERIAL_PORT='/dev/ttyUSB0' ја доделува низата '/dev/ttyUSB0' на променливата SERIAL_PORT. SERIAL_PORT го претставува комуникацискиот интерфејс, односно сериската порта од компјутерот преку која се одвива комуникацијата со мерачот. Работиме на Linux оперативен систем и затоа ја има вредноста во форма на ttyUSB*, а комуникацијата се врши преку USB конверторот приклучен на /dev/ttyUSB0 сериската порта. Доколку работата се врши на Windows тогаш вредноста на SERIAL_PORT би била од типот на COM*.

meter = minimalmodbus.Instrument(SERIAL_PORT, MODBUS_ADDRESS) создава примерок од класата instrument од веќе вклучената библиотека minimalmodbus и ја доделува на променлива meter. Со ова креираме инструмент со име meter и му дозволуваме на кодот да комуницира со Modbus уредот на адреса и сериска порта кои ги дефиниравме претходно.

Потоа вршиме конфигурација на параметрите за комуникација, кои параметри одредуваат како ќе се спроведува комуникацијата со мерниот уред.

Baudrate:9600, Bitesize:8, Parity: None, stopbits:1, timeout:0.1 се така поставени параметри да одговараат на поставките од мерниот уред. Истите вакви вредности се сметаат за дифолт вредности за Modbus конуникација преку сериски интерфејс.

Со извршување на овој дел од кодот, подготвени сме за комуникација со мерниот уред, а конфигурацијата на комуникациските параметри ни гарантира дека комуникацијата е правилно воспоставена.

```
def on_connect(client, userdata, flags, rc):
        if rc == 0:
            print("Connected to broker")
            global Connected
            Connected = True
        else:
            print("Connection failed")
```

```
def on_publish(client, userdata, result):
    print(f'Data published. Data: {attribute_value}')
pass

def on_message(client, userdata, message):
    msg = json.loads(message.payload.decode('utf-8'))
    id = msg['attributeState']['ref']['id']
    att_name = msg["attributeState"]['ref']['name']
    att_value = msg["attributeState"]["value"]
          print(f'Message received. Asset ID: {id}. Attribute name:
{att_name}. Attribute value: {att_value}')
```

Овие претставуваат функции за повратен повик(callback функции) кои се наменети за справување со конкретни настани во процесот на MQTT комуникацијата.

On_connect е повратен повик што се извршува кога клиентот успешно се поврзал со брокерот. Доколку е успешно поврзувањето со брокерот, глобалната променлива connected ја поставува на вредност True сигнализирајќи подолу во кодот дека врската е успешна.

On_publish е повратен повик кој се извршува кога клиентот успешно ќе објави порака до брокерот

Како што е објаснато погоре, *Sommy GQ12* ги зачувува своите вредности во 16-битни регистри. Поголем дел од вредностите се запишуваат како long вредности, па затоа е потребно да се искористат 2 регистри за запишување на една вредност.

Во следниот дел од кодот имаме функции за читање податоци од регистрите на Modbus уредот користејќи ја библиотеката minimalmodbus. Обезбедуваат поедноставен начин за читање податоци од регистрите со истовремено справување со грешките.

```
def read_2registers(starting_register_address):
    try:
        value_high= meter.read_register(starting_register_address,
functioncode=3)
        value_low =
meter.read_register(starting_register_address+1,functioncode=3)
        value_32=(value_high<<16)+ value_low
        return value_32
    except minimalmodbus.ModbusException as e:
        print("Modbus communication error:", e)
        return None

def read_1register(register_address):
    try:
        value = meter.read_register(register_address, functioncode=3)
        return value
    except minimalmodbus.ModbusException as e:
        print("Modbus communication error:", e)
        return None
```

`read_2registers(starting_register_address)` е функција се користи за читање на два последователни 16-битни регистри од Modbus уредот и нивно комбинирање во 32-битна цел број.

Starting_register_address е почетната адреса, односно првиот регистар за читање. Вториот регистар кој го читаме се наоѓа на следната адреса односно starting_register_address+1.

Со meter.read_register(…) која е веќе дефинирана функција во minimalmodbus библиотеката ги отчитуваме вредностите од двата последователни регистри и ги зачувуваме во value_high од првиот регистар, и во value_low во вториот регистар. Користиме функциски код 3 за отчитување на holding регистри (може подетално да се разгледаат функциските кодови погоре во табела 4).

Двете вредности ги комбинираме во 32-битна int вредност со поместување на value_high во лево за 16 бита.

На истиот принцип работи и функцијата `read_1register(register_address)`. Разликата е во тоа што отчитуваме само еден регистар и не вршиме никакво поместување во лево.

Табела 5. Адреси на регистрите за секоја од променливите

Ознака	Адреса	Име на променлива	Тип на податок	Read/Write	Константа
U1	0x016E	A phase voltage high bit	long	R	0.0001V
	0x016F	A phase voltage low bit			
U2	0x0170	B phase voltage high bit	long	R	
	0x0171	B phase voltage low bit			
U3	0x0172	C phase voltage high bit	long	R	
	0x0173	C phase voltage low bit			
I1	0x0174	A phase current high bit	long	R	0.0001A
	0x0175	A phase current low bit			
I2	0x0176	B phase current high bit	long	R	
	0x0177	B phase current low bit			
I3	0x0178	C phase current high bit	long	R	
	0x0179	C phase current low bit			
P	0x017A	Total active power high bit	long	R	0.0001Kw
	0x017B	Total active power low bit			
P1	0x017C	A phase active power high bit	long	R	
	0x017D	A phase active power low bit			
P2	0x017E	B phase active power high bit	long	R	
	0x017F	B phase active power low bit			
P3	0x0180	C phase active power high bit	long	R	
	0x0181	C phase active power low bit			
Q	0x0182	Total reactive power high bit	long	R	0.0001Kvar
	0x0183	Total reactive power low bit			
Q1	0x0184	A phase reactive power high bit	long	R	
	0x0185	A phase reactive power low bit			
Q2	0x0186	B phase reactive power high bit	long	R	
	0x0187	B phase reactive power low bit			

Q3	0x0188	C phase reactive power high bit	long	R	
	0x0189	C phase reactive power low bit			
S	0x018A	Total apparent power high bit	long	R	0.0001kVA
	0x018B	Total apparent power low bit			
S1	0x018C	A phase apparent power high bit	long	R	
	0x018D	A phase apparent power low bit			
S2	0x018E	B phase apparent power high bit	long	R	
	0x018F	B phase apparent power low bit			
S3	0x0190	C phase apparent power high bit	long	R	
	0x0191	C phase apparent power low bit			
ER	0x011E	Total reactive energy high bit	long	R	0.01Kvarh
	0x011F	Total reactive energy low bit			
EA	0x0100	Total active energy high bit	long	R	0.01kWh
	0x0101	Total active energy low bit			
PF	0x0192	Total power factor bit	int	R	0.01
PF1	0x0193	A phase power factor bit	int	R	
PF2	0x0194	B phase power factor bit	int	R	
PF3	0x0195	C phase power factor bit	int	R	
F	0x0199	Frequency	int	R	0.01Hz
Извор: Sommy GQ12 Series Rail Mounting 3Phase Energy Meter Manual - 2014					

Функциите read_2registers и read_1register подоцна ги корстиме во main делот од кодот каде вршиме отчитување на вредностите запишани во регистрите, прикажани во табела 5, и нивно инстантно впишување во JSON формат, за објавување до MQTT брокерот.

```
Connected = False
username = 'master:mqttuser'
secret = '0ZyoQ2tnbgItsknQVx0vSNeW0rrGQM17'
host = '192.168.1.8'
port = 8883
clientID = 'mqtt'


assetIDwr = '6uVLJB0ZaHQb24Tgy7G6UF'
attributeWr = 'data'
```

Вредностите кои ги запомнавме при креирањето на user и asset во делот за openremote, ги користиме овде во кодот.

Со линија од кодот прикажан долу, сигурни сме дека откако ќе се воспостави конекција го вршиме објавувањето на податоците, во нашиот случај JSON објект, директно на интегрираниот MQTT брокер на *OpenRemote*.

```
clientMQTT.publish(f"master/{clientID}/writeattributevalue/{attributeWr
}/{assetIDwr}", attribute_value)
```

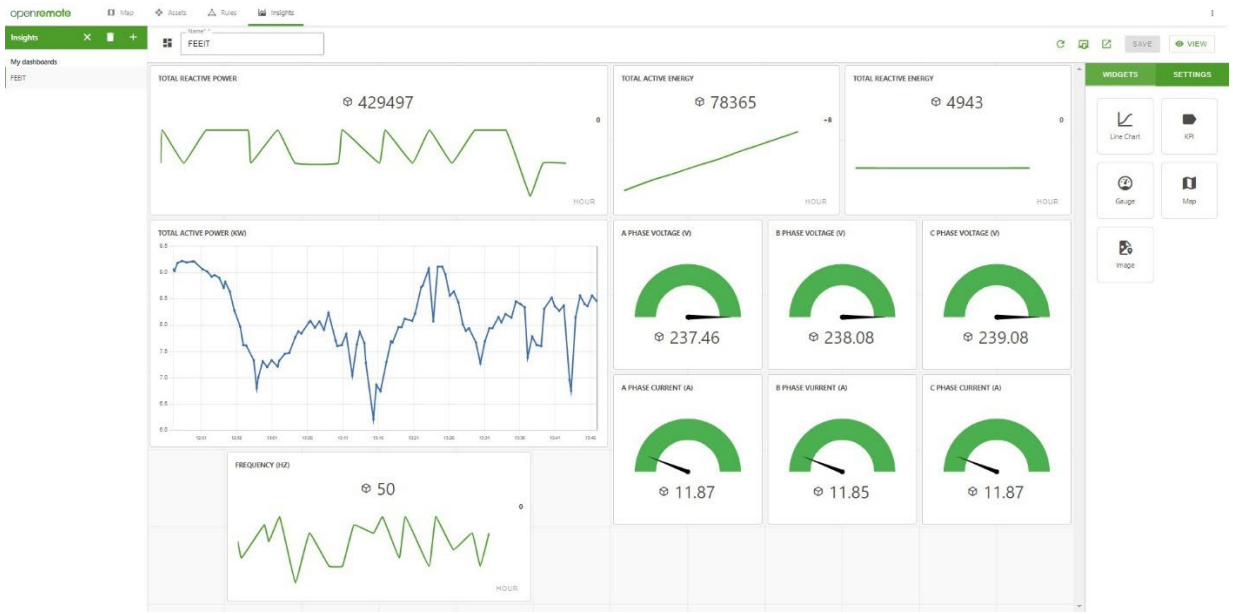
Во последните делови од кодот, овозможуваме да се изврши одјавување од брокерот и стопирање на MQTT јамката со претискање на Ctrl+C.

```
except KeyboardInterrupt:
    print("Disconnecting...")
    clientMQTT.disconnect()
    clientMQTT.loop_stop()
```

5 Експериментални резултати

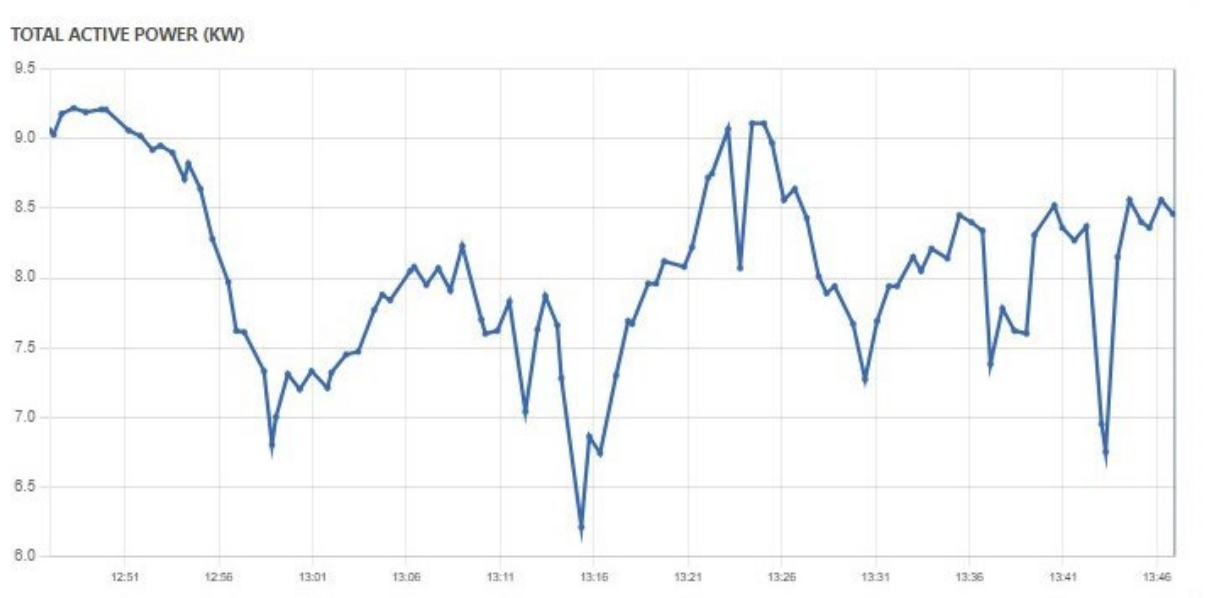
По изградбата на системот, истиот беше тестиран на фотоволтаичната централа при Факултетот за електротехника и информациски технологии во Скопје. Централата е со вкупна моќност од 30 kW. Тестот беше спроведен на 05.10.2023 година, во временски интервал од 12 часот до 14 часот. Времето и метеоролошките параметри за Скопје, во тој период, беа сончево со умерена облачност, температура од 26 °C и притисок од 986 hPA. Фотоволтаичната централа е поврзана на паметно мерило *Sommy GQ-12* кое го споменавме во делот за хардверска инфраструктура на страна 11.

На слика 20, може да се видат дел од резултатите отчитани од мерењето, претставени во Insights прозорецот од *OpenRemote* платформата. Insights овозможува резултатите кои се зачувани со помош на Store Data Points конфигуртацијата на атрибутите да се претстават сликовито со помош на линиски график, KPI (key performance indicator) или gauge [покажувач што се движи по калибрирана скала].



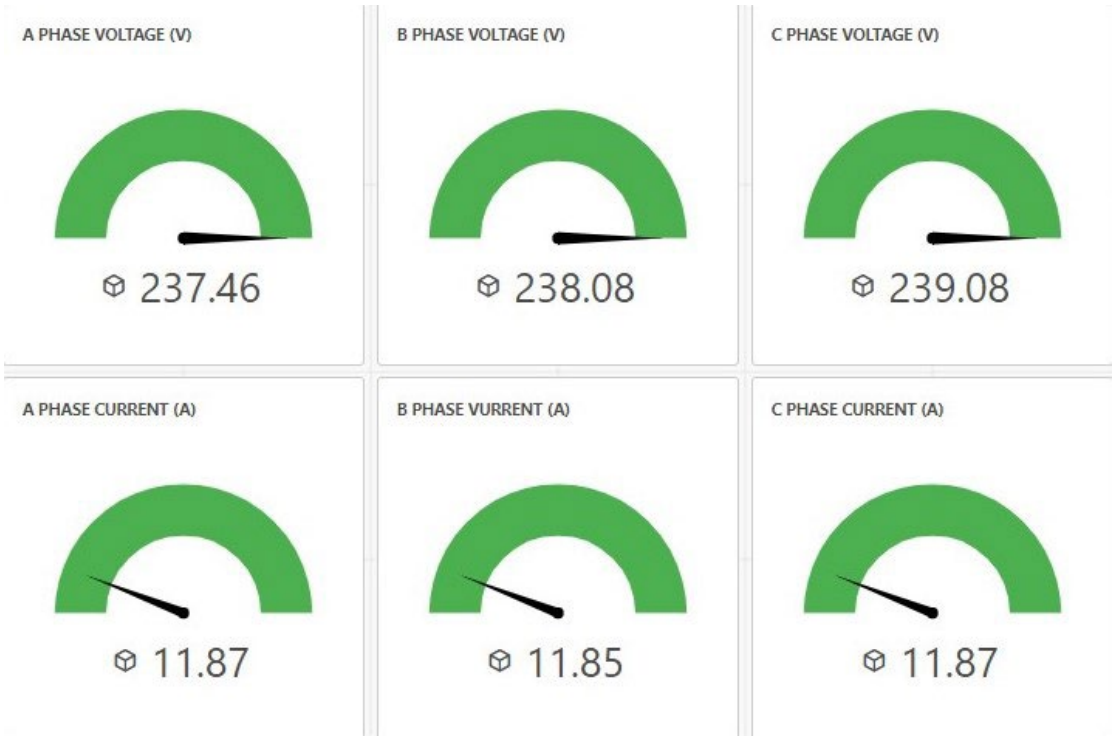
Слика 20. Сликовита претстава на резултатите отчитани од мерењето

Доколку ги разгледаме вредностите кои ги добивме за моќноста во тој интервал, на слика 21, при споменатите временски услови, може да се забележи дека таа достигнала некој максимум од 9.22 kW, и минимум вредност од 6.21 kW.



Слика 21. Вредности од измерената моќност претставени со линиски график

Во врска со резултатите за напонот, слика 22, може да се каже дека за неговата вредност постојано беа отчитани 230V±10%, која всушност претставува и негова номинална вредност.

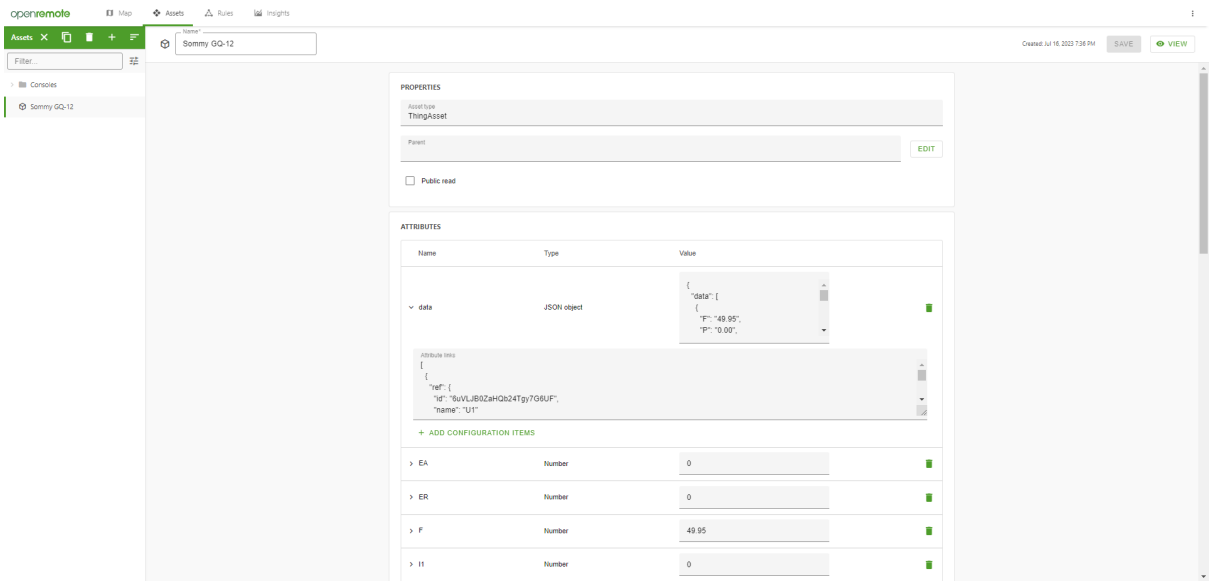


Слика 22. Вредности од измерениот напон и струја

5.1 Филтри искористени во Openremote

Како што напоменавме погоре, податоците отчитани од уредот, ги испраќаме на *OpenRemote* како JSON објект во еден атрибут од дадено средство. Доколку сакаме да отчитаме поединечни вредности од тој објект, и да ги поставиме како додатни атрибути од една вредност, како пример да го земеме вредноста на напонот U1, потребно е да се искористат т.н. филтри.

Во делот за Modify Asset, каде имаме пристап до листата на атрибути, на атрибутот на кој го испраќаме JSON објектот, во нашиот случај атрибутот со име data, го модифицираме на тој начин што со притискање на +Add configuration items копчето избираме да додадеме Attribute link. Со избирање на Attribute link се појавува прозорче каде треба да испишеме краток код. Процесот е прикажан на сликата подолу.



Слика 23. Процесот на додавање на филтри кои овозможуваат исчитување на една вредност од целиот JSON објект

Кодот кој би го испишале, за да ја добиеме вредноста на напонот U1 во истиот Asset но нов атрибут е:

```
[
 {
  "ref": {
   "id": "6uVLJB0ZaHQb24Tgy7G6UF",
   "name": "U1"
  },
  "filters": [
   {
    "type": "jsonPath",
    "path": "$.data[0].U1",
```

```
        "returnFirst": true,
        "returnLast": false
      }
    ]
  }
]
```

Во делот “ref” се внесуваат податоците за атрибутот кој сме го креирале се со цел во него да се запише исчитаната вредност. Id ја има информацијата за тоа кој asset го користиме, односно идентификацискиот број на средството каде што сакаме да се зачува таа вредност. Name ја има информацијата за како сме го крстиле атрибутот. Во нашиот случај за id го користиме истото како и assetIDwr од *Python* кодот.

Постои можност да се креира ново средство, а секое средство има различен id.

6 Заклучок

Целта на овој дипломски труд е да се развие интелегентен мерен инструмент за енергија базиран на IoT користејќи микрокомпјутер *Raspberry Pi* и *Sommy GQ-12* како мерен инструмент за енергија. Успешното завршување на проектот претставува значајно достигнување во областа на управувањето со енергија и автоматизацијата. Преку прецизно планирање и користење на најсовремена технологија, создадовме ефикасен систем за следење на енергијата кој има голем потенцијал и за станбени и за индустриски апликации.

Нашиот избор на *Raspberry Pi* како централна контролна единица и интеграцијата на мерниот уред *Sommy GQ-12* се покажаа како моќна комбинација, овозможувајќи ни прецизно да ја мериме и следиме потрошувачката на енергија во реално време. Ова не само што им овозможува на корисниците да стекнат вредни сознанија за нивните шеми за користење на енергијата, туку исто така придонесува за поширокатa цел за одржливо управување со енергијата.

Усвојувањето на *OpenRemote* како наша IoT платформа додаде дополнителен слој на функционалност и пристапност на нашиот проект. Го олесни далечинското следење и контрола, осигурувајќи дека корисниците можат удобно да управуваат со нивната потрошувачка на енергија од каде било, а со тоа промовирајќи ја енергетската ефикасност и заштедата на трошоците.

Овој проект го прикажува потенцијалот на IoT технологијата да го револуционизира начинот на кој ги следиме и зачувуваме енергетските ресурси. Со континуираниот напредок на IoT технологијата и посветеноста на иноватори како нас, можеме да се радуваме на поодржлива и енергетски ефикасна иднина.

И покрај тоа што фокусот на овој дипломски труд е да се изгради систем за менаџирање со електрична енергија, истиот се тестира и во реални околности. Тестирањето успешно се извршува на Факултетот за електротехника и информациски технологии. Резултатите од мерењето ги внесуваме и прикажуваме во *OpenRemote* платформата. Исто така ги разгледуваме и различните начини како резултатите може да бидат прикажани сликовито.

7 Appendix – Програмски код во Python

```
import paho.mqtt.client as mqtt
import time
import ssl
import json
import minimalmodbus

MODBUS_ADDRESS = 1
SERIAL_PORT = '/dev/ttyUSB0'

meter = minimalmodbus.Instrument(SERIAL_PORT, MODBUS_ADDRESS)
meter.serial.baudrate = 9600
meter.serial.bytesize = 8
meter.serial.parity = minimalmodbus.serial.PARITY_NONE
meter.serial.stopbits = 1
meter.serial.timeout = 0.1

def on_connect(client, userdata, flags, rc):
        if rc == 0:
            print("Connected to broker")
            global Connected
            Connected = True
        else:
            print("Connection failed")

def on_publish(client, userdata, result):
    print(f'Data published. Data: {attribute_value}')
pass

def on_message(client, userdata, message):
    msg = json.loads(message.payload.decode('utf-8'))
    id = msg['attributeState']['ref']['id']
    att_name = msg["attributeState"]['ref']['name']
    att_value = msg["attributeState"]["value"]
    print(f'Message received. Asset ID: {id}. Attribute name: {att_name}.
Attribute value: {att_value}')

def read_2registers(starting_register_address):
    try:
        value_high= meter.read_register(starting_register_address,
functioncode=3)
        value_low =
meter.read_register(starting_register_address+1,functioncode=3)
        value_32=(value_high<<16)+ value_low
        return value_32
    except minimalmodbus.ModbusException as e:
        print("Modbus communication error:", e)
```

```
        return None

def read_1register(register_address):
    try:
        value = meter.read_register(register_address, functioncode=3)
        return value
    except minimalmodbus.ModbusException as e:
        print("Modbus communication error:", e)
        return None

Connected = False
username = 'master:mqttuser'
secret = 'YMOMT6KEQcDLKAFHIVYHpj0lNPrPqEqb'
host = '192.168.1.8'
port = 8883
clientID = 'mqtt'


assetIDwr = '6uVLJB0ZaHQb24Tgy7G6UF'
attributeWr = 'data'

clientMQTT = mqtt.Client(clientID)
clientMQTT.username_pw_set(username, password = secret)

context = ssl.SSLContext(ssl.PROTOCOL_TLSv1_2)
context.check_hostname = False
clientMQTT.tls_set_context(context)

clientMQTT.on_connect = on_connect
clientMQTT.on_publish = on_publish
clientMQTT.on_message = on_message
clientMQTT.connect(host, port=port)
clientMQTT.loop_start()

while Connected != True:
    time.sleep(0.1)

try:
    while True:
        data = {"data":[{
           "U1":"".join(format(read_2registers(0x016E)*0.0001, ".2f")),
           "U2":"".join(format(read_2registers(0x0170)*0.0001, ".2f")),
           "U3":"".join(format(read_2registers(0x0172)*0.0001, ".2f")),
           "U12":"".join(format(read_2registers(0x016E)*0.0001*1.732, ".2f")),
           "U23":"".join(format(read_2registers(0x0170)*0.0001*1.732, ".2f")),
           "U31":"".join(format(read_2registers(0x0172)*0.0001*1.732, ".2f")),
           "I1":"".join(format(read_2registers(0x0174)*0.0001, ".2f")),
           "I2":"".join(format(read_2registers(0x0176)*0.0001, ".2f")),
           "I3":"".join(format(read_2registers(0x0178)*0.0001, ".2f")),
```

```
            "P1":"".join(format(read_2registers(0x017C)*0.0001, ".2f")),
            "P2":"".join(format(read_2registers(0x017E)*0.0001, ".2f")),
            "P3":"".join(format(read_2registers(0x0180)*0.0001, ".2f")),
            "P":"".join(format(read_2registers(0x017A)*0.0001, ".2f")),
            "Q1":"".join(format(read_2registers(0x0184)*0.0001, ".2f")),
            "Q2":"".join(format(read_2registers(0x0186)*0.0001, ".2f")),
            "Q3":"".join(format(read_2registers(0x0188)*0.0001, ".2f")),
            "Q":"".join(format(read_2registers(0x0182)*0.0001, ".2f")),
            "S1":"".join(format(read_2registers(0x018C)*0.0001, ".2f")),
            "S2":"".join(format(read_2registers(0x018E)*0.0001, ".2f")),
            "S3":"".join(format(read_2registers(0x0190)*0.0001, ".2f")),
            "S":"".join(format(read_2registers(0x018A)*0.0001, ".2f")),
            "PF1":"".join(format(read_1register(0x0193)*0.01, ".2f")),
            "PF2":"".join(format(read_1register(0x0194)*0.01, ".2f")),
            "PF3":"".join(format(read_1register(0x0195)*0.01, ".2f")),
            "PF":"".join(format(read_1register(0x0192)*0.01, ".2f")),
            "EA":"".join(format(read_2registers(0x0100)*0.01, ".2f")),
            "ER":"".join(format(read_2registers(0x011E)*0.01, ".2f")),
            "F":"".join(format(read_1register(0x0199)*0.01, ".2f"))
            }]}
        attribute_value=json.dumps(data)
        clientMQTT.publish(f"master/{clientID}/writeattributevalue/{attributeW
r}/{assetIDwr}", attribute_value)
        time.sleep(10)

except KeyboardInterrupt:
    print("Disconnecting...")
    clientMQTT.disconnect()
    clientMQTT.loop_stop()
```

8 Референци

[1] Seifedine Kadry, „Internet of Things (IoT)“, Noroff University College, ResearchGate, мај 2021, стр. 1-3

[2] Alexander S. Gillis, internet of things(IoT), https://www.techtarget.com/iotagenda/definition/Internet-of-Things-IoT, 31.07.2023

[3] Raspberry Pi Foundation, https://www.raspberrypi.org/help/what-%20is-a-raspberry-pi/, 27.07.2023

[4] OpenSource, https://opensource.com/resources/raspberry-pi, 27.07.2023

[5] Electronic Team Inc., https://www.eltima.com/article/rs485-communication-guide/, 28.07.2023

[6] Electronic Team Inc., https://www.eltima.com/article/modbus-vs-rs485/, 28.07.2023

[7] OpenRemote, https://www.openremote.io/product/, 29.07.2023

[8] https://github.com/openremote/openremote/wiki/User-Guide:-Manager-UI, 29.07.2023

[9] Random Nerd Tutorials, https://randomnerdtutorials.com/what-is-mqtt-and-how-it-works/, 29.07.2023

[10] Roger Light, PyPi, https://pypi.org/project/paho-mqtt/, 29.07.2023

[11] Digicert, „Begginer’s Guide to TLS/SSL Certificates“, 2019, стр. 1

[12] Jonas Berg, https://minimalmodbus.readthedocs.io/en/stable/, 29.07.2023

[13] GQ12 Series Rail Mounting 3 Phase Energy Meter Manual, https://www.sommy.com.cn/up_pic/file/20180413113859875987.pdf , 23.07.2023