Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних систем та мереж
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Хмарна система моніторингу споживання електроенергії на базі використання пристроїв IoT

Виконав: студент 4 курсу, групи СІс-41
спеціальності 123 «Комп'ютерна інженерія»
(шифр і назва спеціальності)

(підпис) Климчук Д.А. (прізвище та ініціали)

Керівник (підпис) Луцків А.М. (прізвище та ініціали)

Нормоконтроль (підпис) Луцик Н.С. (прізвище та ініціали)

Завідувач кафедри (підпис) Осухівська Г.М. (прізвище та ініціали)

Рецензент (підпис) Пастух О.А. (прізвище та ініціали)

Тернопіль
2025

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет *комп'ютерно-інформаційних систем і програмної інженерії*
(повна назва факультету)

Кафедра *комп'ютерних систем та мереж*
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ *Осухівська Г.М.*
(підпис) (прізвище та ініціали)
«27» січня 2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня *бакалавр*
(назва освітнього ступеня)

за спеціальністю *123 «Комп'ютерна інженерія»*
(шифр і назва спеціальності)

студента Климчук Дмитро Андрійович
(прізвище, ім'я, по батькові)

1. Тема роботи Хмарна система моніторингу споживання електроенергії на базі використання пристроїв IoT

Керівник роботи Луцків Андрій Мирославович, канд. техн. наук, доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від *« 27 » січня 2025 року № 4/7-53*
2. Термін подання студентом завершеної роботи *20 червня 2025 року*
3. Вихідні дані до роботи *Технічне завдання*

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. 1. Аналіз технічного завдання
2. Проєктна частина
3. Практична частина
4. Безпека життєдіяльності, основи охорони праці.
Висновки
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1.
2.
3.
4.

6. . Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	Пилипець М.І. д.т.н., проф. каф. МТ		

7. Дата видачі завдання *27.01.2025 р.*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	*Розробка технічного завдання*	*27.01 – 02.02*	*Викон.*
2.	*Аналіз технічного завдання та постановка задачі*	*03.02 – 15.02*	*Викон.*
3.	*Розробка архітектури системи*	*16.02 – 10.03*	*Викон.*
4.	*Реалізація збору даних з розеток (ESP32 → Home Assistant)*	*11.03 – 23.03*	*Викон.*
5.	*Розробка API для обміну даними між сайтом і Home Assistant*	*24.03 – 5.04*	*Викон.*
6.	*Створення вебінтерфейсу: авторизація, графіки, дашборд*	*6.04 – 17.04*	*Викон.*
7.	*Тестування взаємодії між ESP32, сайтом і розетками*	*18.04 – 22.04*	*Викон.*
8.	*Безпека життєдіяльності, основи охорони праці*	*23.04 – 30.04*	*Викон.*
9.	*Оформлення пояснювальної записки і графічного матеріалу*	*1.05 – 8.06*	*Викон.*
10.	*Перевірка на академічний плагіат, перевірка керівником та консультантами*	*9.06 – 15.06*	*Викон.*
11.	*Попередній захист кваліфікаційної роботи бакалавра*	*16.06 – 22.06*	*Викон.*
12.	*Захист кваліфікаційної роботи бакалавра*	*27.06*	*Викон.*

Студент _____ Климчук Д.А.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Луцків А.М.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Климчук Д.А. Хмарна система моніторингу споживання електроенергії на базі використання пристроїв IoT: робота на здобуття кваліфікаційного ступеня бакалавра: спец. 123 — комп'ютерна інженерія. Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя, 2025.

Ключові слова: хмарні сервіси, мікроконтролер, Інтернет речей, веб-сервіси, енергоспоживання, home assistance, моніторинг, дистанційний контроль.

Об'єкт проектування – Хмарна система моніторингу споживання електроенергії на базі використання пристроїв IoT.

Кваліфікаційна робота бакалавра складається з трьох розділів.

У першому розділі приводиться аналіз технічного завдання та області його застосування, вибір оптимального протоколу зв'язку та здійснюється опис загальної архітектури системи.

В другому розділі здійснюється побудова та опис структурної схеми, аналіз мікроконтролера ESP32, розгортання та налаштування вибраної платформи Home Assistant, додавання інтеграції Tuya Local, розробка програмного забезпечення.

В третьому розділі приводиться розгортання апаратних програмних компонент, налаштування Raspberry Pi 4, опис та налаштування розумних розеток, організацію доступу до хмарної системи моніторингу з інтернету та здійснення тестування системи.

ANNOTATION

Klymchuk D.A. Cloud-based system for monitoring electricity consumption using IoT devices: Bachelor’s Graduation Thesis: speciality 123 — computer engineering. Ternopil: Ternopil Ivan Puluj National Technical University, 2025.

Keywords: cloud services, microcontroller, Internet of Things, web services, energy consumption, home assistance, monitoring, remote control.

Design object - Development of a cloud-based system for monitoring electricity consumption based on IoT devices.

The bachelor's thesis consists of three sections.

The first section analyzes the terms of reference and its application, selects the optimal communication protocol, and describes the overall architecture of the system.

The second section describes the construction and description of the block diagram, analysis of the ESP32 microcontroller, deployment and configuration of the selected Home Assistant platform, adding Tuya Local integration, and software development.

The third section describes the deployment of hardware and software components, the configuration of the Raspberry Pi 4, the description and configuration of smart sockets, the organization of access to the cloud monitoring system from the Internet, and the testing of the system.

ЗМІСТ

ВСТУП

Інтегровані системи автоматизації активно впроваджуються в різні сфери життя, зокрема в інтернет речей (IoT, Internet of Things). Завдяки прогресу в галузі мікроконтролерів, бездротових технологій та хмарних сервісів, стало можливим створення систем для моніторингу та управління різними пристроями в реальному часі. Питання енергоефективності набувають особливої значущості, оскільки вони стосуються як домашніх споживачів, так і промислових компаній.

У побуті хмарні системи моніторингу споживання електроенергії забезпечують комфорт та енергоефективність, дозволяючи контролювати стан та споживання пристроїв в приміщенні. У комерційних та офісних приміщеннях такі системи сприяють оптимізації використання та моніторингу великої кількості пристроїв одночасно.

Хмарна система моніторингу споживання електроенергії може бути реалізована за допомогою різноманітних технологій, включаючи використання мікроконтролерів, сенсорів і виконавчих механізмів. Такі системи можуть інтегруватися у "розумний дім", забезпечуючи зручне управління через мобільні додатки, голосові команди чи за допомогою попередньо заданих сценаріїв.

Метою кваліфікаційній роботі є розробка хмарної системи для моніторингу споживання електроенергії, яка базується на пристроях IoT з використанням мікроконтролера та різних пристроїв або сенсорів. Дана система повинна надавати можливість відстежувати споживання електроенергії в режимі реального часу, відображати статистичні дані на сайті та на екрані, а також керувати станом пристроїв як з локального рівня (за допомогою кнопок), так і через хмарний інтерфейс (Home Assistant).

Розроблена система у кваліфікаційній роботі має бути простою у використанні, енергоефективною та інтегрованою у сучасні умови житлового чи комерційного простору. Вона повинна поєднувати функціональність, зручність та інноваційність, відповідаючи вимогам сучасного світу.

РОЗДІЛ 1 АНАЛІЗ ТЕХНІЧНОГО ЗАВДАННЯ

1.1. Аналіз технічого завдання

Розробка хмарної системи моніторингу споживання електроенергії на базі IoT-пристроїв потребує визначення ключових функцій, вимог до надійності та технічної реалізації, а також аналізу можливих варіантів побудови системи. Основною метою є створення платформи, яка дозволить у реальному часі відстежувати споживання електроенергії підключених пристроїв, аналізувати дані та здійснювати керування відповідним навантаженням [1].

Враховуючи сучасні технології в галузі Інтернету речей, було розглянуто кілька варіантів реалізації системи з точки зору вибору апаратної та програмної платформи.

1.1.1. Вибір мікроконтролера

Для зчитування даних із пристроїв та їх передачі до хмари необхідно використати мікроконтролер із підтримкою бездротового з’єднання. Серед розглянутих варіантів:

– Arduino Uno — доступна та проста в реалізації платформа, однак не має вбудованого Wi-Fi, що ускладнює передачу даних без додаткових модулів.

– Raspberry Pi — потужний міні-комп’ютер з широкими можливостями, проте перевищує потреби даного проєкту за вартістю та споживанням енергії.

– ESP8266 / ESP32 — компактні контролери з вбудованим Wi-Fi, широкою спільнотою та підтримкою необхідних бібліотек. ESP32 додатково має більшу продуктивність і більше периферії

Для даного проєкту оптимальним рішенням є ESP32 через його гнучкість, можливість паралельного зчитування даних, керування пристроями, енергоефективність та передачу інформації у хмару.

1.1.2. Вибір протоколу зв’язку

Одним із ключових аспектів розробки хмарної системи моніторингу споживання електроенергії на базі IoT-пристроїв є організація передачі даних між пристроями та центральною системою управління. Протокол зв’язку визначає, як інформація буде переміщуватися в мережі, забезпечуючи надійність, швидкість і безпеку обміну даними [2].

1.1.2.1.Взаємодія через протокол HTTP і REST API

HTTP — один із найбільш розповсюджених протоколів зв’язку, який використовується для обміну даними між клієнтом та сервером за допомогою стандартизованих запитів (GET, POST, PUT, DELETE). Використання RESTful API дозволяє створити простий інтерфейс для віддаленого керування пристроями та отримання даних про споживання електроенергії [2].

Серед переваг даного протоколу варто взяти до уваги, що він є простим у реалізації(REST API інтегрується практично з будь-якою платформою, і численні бібліотеки спрощують його використання), універсальним(взаємодія може здійснюватися як з мобільних пристроїв, так і через веб інтерфейси) та гнучким щодо розширення підтримки нових функцій за потреби.

Стосовно недоліків, то це збільшена частота запитів та обсяг трафіку. Для систем, де потрібна передача даних у режимі реального часу, REST API може бути менш ефективним через більші затримки порівняно з іншими протоколами.

1.1.2.2.Протокол MQTT

MQTT (Message Queuing Telemetry Transport) — легкий протокол публікації/підписки, призначений для роботи в умовах обмежених ресурсів та

ненадійних мереж. Він широко використовується у сфері IoT для обміну повідомленнями між пристроями через центрального брокера.

Перевагами даного протоколу є низькі накладні витрати, асинхронна комунікація та масштабованість. Мінімальний обсяг даних у повідомленнях робить протокол ідеальним для пристроїв з обмеженими ресурсами, забезпечує швидку доставку повідомлень у режимі реального часу та легко інтегрується у системи з великою кількістю пристроїв.

Також недоліками протоколу є потреба у брокері та безпека. Необхідність мати центральний сервер (брокер) для обробки повідомлень може ускладнювати архітектуру. Також треба ретельно налаштовувати рівні доступу та шифрування, щоб забезпечити захист даних.

1.1.2.3.Взаємодія через WebSocket

WebSocket — протокол, що забезпечує двосторонню комунікацію між клієнтом і сервером у режимі реального часу. На відміну від традиційного HTTP, WebSocket дозволяє встановити постійне з’єднання, через яке дані надсилаються миттєво за подіями.

Робота у реальному часі забезпечує швидкий обмін даними без необхідності повторних HTTP-запитів. Також постійне з’єднання зменшує накладні витрати на встановлення та розрив з’єднання.

Проте даний протокол вимагає більш складної реалізації, особливо коли йдеться про масштабованість та безпеку. Також не всі платформи та пристрої можуть підтримувати WebSocket за замовчуванням, що потрібно враховувати під час інтеграції.

1.1.2.4.Локальний протокол з використанням спеціалізованих API

Зважаючи на особливості деяких реалізацій, для систем моніторингу може використовуватися локальний протокол зв’язку із застосуванням специфічних API, що дозволяють взаємодіяти з пристроями без необхідності звертатися до хмарних серверів. Завдяки цьому всі запити й відповіді залишаються в межах

приватної домашньої мережі, а час затримки між подією (натисканням кнопки або зміною стану розетки) і реакцією системи становить лічені мілісекунди [4].

Крім того, локальна архітектура відкриває можливість використовувати розподілені інтелектуальні сценарії безпосередньо на ESP32 без додаткових затримок на мережеві транзити. Це дозволяє реалізовувати автоматизацію за складними умовами (таймінги, порогові значення сенсорів) навіть у разі відсутності інтернет-каналу.

Окремо варто зазначити, що для розширення функціоналу локального протоколу не потрібно оновлювати прошивку в усіх пристроях: достатньо підготувати новий скрипт або інтеграцію, і всі пов’язані з ним пристрої розпочнуть обмінюватися даними згідно з новими правилами. Це значно скорочує час обслуговування та знижує ризики помилок при масштабуванні системи [4].

Незважаючи на потенційні виклики при збільшенні числа підключених пристроїв, правильне планування ресурсів мережі та виділення ESP32 або додаткових вузлів (наприклад, декількох мікроконтролерів або MQTT-брокера в локальній мережі) допомагає зберегти високу продуктивність і стабільність роботи навіть у розширених конфігураціях. Таким чином, локальний протокол із спеціалізованими API забезпечує комбінацію безпеки, швидкодії та гнучкості, необхідних для ефективного моніторингу та керування енергоспоживанням в IoT-системах.

1.1.2.5.Вибір оптимального протоколу зв’язку

У процесі аналізу варіантів організації зв’язку між пристроями та центральною системою було розглянуто численні підходи: від простих RESTful HTTP-запитів до більш спеціалізованих рішень, таких як MQTT та WebSocket. Для даного проєкту в якому використовується ESP32 та має бути можливість інтегрування з платформою автоматизації пристроїв IoT, основною технологією стала організація локального зв’язку через HTTP, що реалізовано за допомогою вбудованого веб-сервера на ESP32.

1.1.3. Вибір елементів керування

Однією з важливих складових розробки хмарної системи моніторингу споживання електроенергії є забезпечення інтуїтивно зрозумілого інтерфейсу для взаємодії користувача з системою. У даному проєкті було обрано використання трьох фізичних кнопок, OLED-дисплею та веб-сайту, що в сукупності дозволяють здійснювати керування пристроями, отримувати статус енергоспоживання та переглядати статистичні дані.

Фізичні кнопки обрано як надійний механізм для управління окремими функціями системи. Перша кнопка відповідає за керування пристроєм типу «Comp Socket», дозволяючи при кожному натисканні перемикати його стан, а друга кнопка аналогічно керує пристроєм «Monitor Socket». Третя кнопка призначена для виклику відображення детальної статистики споживання електроенергії, що передбачає зчитування даних зі сенсорів для обох типів пристроїв. Такий розподіл завдань серед кнопок покращує зручність управління, оскільки кожна кнопка відповідає за чітко визначену функцію, що дозволяє користувачу швидко орієнтуватися в операціях системи [5].

OLED-дисплей застосовано для локального відображення інформації, що є надзвичайно важливим елементом інтерфейсу. Дисплей відображає як короткі повідомлення про стан пристроїв, так і детальну статистику споживання електроенергії. Вбудований дисплей дозволяє отримувати всю необхідну інформацію без необхідності підключення до зовнішніх ресурсів, що є значною перевагою для локального моніторингу.

Веб-сайт виступає як додатковий канал взаємодії з системою, забезпечуючи віддалений доступ до даних та можливість керування пристроями через інтернет. Такий підхід дозволяє зручно переглядати графічні інтерфейси, аналізувати історичні дані споживання та здійснювати оперативне керування пристроями навіть з мобільних пристроїв.

Порівняно з іншими альтернативами, такими як сенсорні екрани, які часто потребують значно більших фінансових витрат і ускладненої програмної підтримки, або спеціалізовані мобільні додатки, розглянутий підхід є більш

доступним і ефективним. Фізичні кнопки забезпечують надійний та простий спосіб управління, OLED-дисплей відображає важливу інформацію, а веб-сайт розширює можливості системи за рахунок віддаленого доступу. У підсумку, обрана конфігурація елементів керування забезпечує оптимальний баланс між простотою використання, надійністю та можливістю подальшого розширення функціоналу системи моніторингу споживання електроенергії на базі IoT-пристроїв [5].

1.2. Аналіз області застосування програмно-апаратної системи

Вбудовані системи відіграють важливу роль у сфері енергоменеджменту, дозволяючи реалізовувати гнучкі, масштабовані та енергоефективні рішення для побутових, комерційних і промислових приміщень. З розвитком Інтернету речей (IoT) з'явилися нові можливості для моніторингу, аналізу та керування енергоспоживанням в режимі реального часу. Хмарні системи на основі IoT, які поєднують мікроконтролери, датчики та хмарні сервіси, дозволяють значно підвищити прозорість у споживанні енергії, сприяючи зниженню витрат і підвищенню екологічної свідомості користувачів [6].

Хмарна система моніторингу електроспоживання, побудована на базі мікроконтролера та пристроїв або датчиків, здатна у реальному часі передавати дані до хмарного сервера (наприклад, через Home Assistant), забезпечуючи користувача аналітичними графіками, звітами та можливістю віддаленого керування пристроями. Область застосування такої системи охоплює:

Житлові приміщення. У приватних будинках і квартирах система дозволяє стежити за енергоспоживанням побутових пристроїв (пральних машин, бойлерів, обігрівачів тощо), визначати найбільш енерговитратні прилади та вживати заходів для економії. Також користувач може дистанційно вмикати/вимикати пристрої, налаштовувати автоматичні сценарії, отримувати сповіщення про перевищення заданих порогів споживання.

Комерційні об’єкти. У кафе, офісах, коворкінгах та готелях система сприяє оптимізації енергоспоживання через централізоване керування пристроями, наприклад, освітленням, вентиляцією або технікою. Це дозволяє знизити витрати на електроенергію та підвищити ефективність управління [6].

Освітні та медичні установи. У школах, університетах, лікарнях моніторинг споживання електроенергії дозволяє адміністраціям вчасно виявляти неефективну роботу обладнання або порушення режиму споживання, що може мати не лише економічні, а й безпекові наслідки.

Спеціалізовані об'єкти. Наприклад, у центрах обробки даних або науково-дослідних установах моніторинг енергоспоживання дозволяє забезпечити стабільну роботу серверів і лабораторного обладнання, запобігти перевантаженням і перегріву.

Для забезпечення взаємодії між пристроями та центральним сервером у системах моніторингу та керування енергоспоживанням можуть використовуватись різноманітні засоби комунікації — як локальні, так і хмарні. Такий підхід дозволяє інтегрувати широкий спектр пристроїв Інтернету речей (IoT), включаючи розумні розетки, реле, шлагбауми, освітлювальні прилади та інші елементи автоматизованої інфраструктури.

Система надає можливість моніторингу енергоспоживання в реальному часі, аналізу історичних даних і віддаленого керування пристроями через зручний веб інтерфейс. Це відкриває широкі перспективи її використання як у побутових умовах — для підвищення енергоефективності житлових приміщень, — так і в офісах, комерційних об’єктах, громадських установах або навіть на виробництві, де контроль за споживанням електроенергії є важливим чинником економії ресурсів та оптимізації витрат.

Гнучка архітектура дає змогу адаптувати систему до конкретних потреб користувача, забезпечуючи масштабованість, надійність і можливість інтеграції з популярними платформами автоматизації, зокрема системами розумного дому. Обраний спосіб підключення та комунікації може варіюватися залежно від

технічних умов і поставлених завдань, що робить систему універсальною основою для побудови сучасного рішення у сфері енергомоніторингу.

Таким чином, хмарна система моніторингу електроенергії на базі мікроконтролера є універсальним рішенням, яке може знайти застосування у широкому спектрі житлових, комерційних та спеціалізованих приміщень. Її перевагами є простота реалізації, низька вартість компонентів та можливість інтеграції з іншими системами "Розумного дому".

1.3. Загальна архітектура системи

Типова архітектура систем керування енергоспоживанням на базі IoT складається з кількох взаємопов’язаних рівнів. Першим є локальний рівень, що включає вимірювальні модулі — датчики струму, напруги та потужності, вбудовані в розумні розетки, реле чи спеціальні лічильники. Такі пристрої фіксують параметри мережі та передають їх далі для обробки.

Наступний рівень відповідає за комунікацію та агрегацію даних. Локальні сенсори можуть об’єднуватись у мережі на базі ZigBee або NB-IoT через хаби (шлюзи), які приймають показники від кількох точок і скеровують їх до централізованої системи. В інших реалізаціях датчики передають дані безпосередньо в хмарні сервіси або на локальні сервери за допомогою Wi-Fi, LoRaWAN, 4G/5G чи MQTT-з’єднань. Часто використовують готові модулі Tuya, які через власний шлюз або пряме MQTT-з’єднання з Tuya Cloud API забезпечують безпечну передачу телеметрії та отримання команд управління.

Дані з комунікаційного рівня потрапляють до сховищ — це можуть бути локальні бази даних (InfluxDB, SQL) на Raspberry Pi чи подібних платформах, а також хмарні сервіси (AWS IoT, Azure IoT Hub, Google Cloud IoT, Tuya Cloud (рис. 1.1). У сховищі відбувається накопичення телеметрії та виконання аналітичних алгоритмів: агрегація, виявлення аномалій і прогнозування споживання.

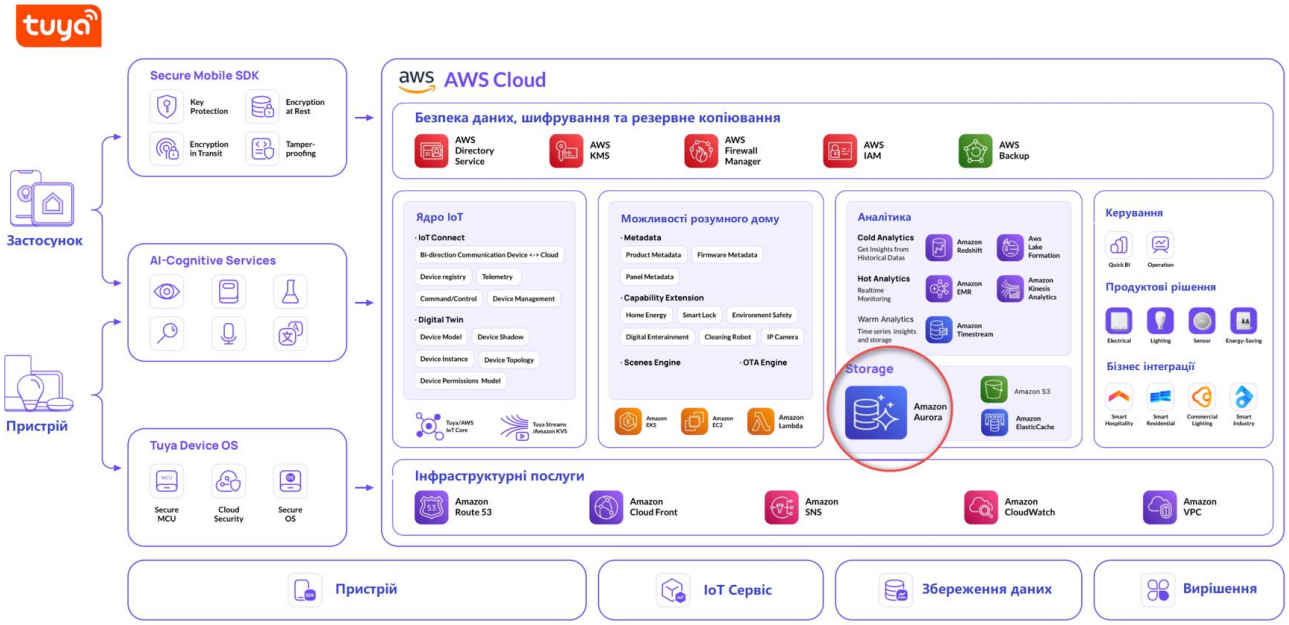

Рисунок 1.1 – Архітектура платформи розробника IoT Tuya, побудована на AWS

Нарешті, прикладний рівень забезпечує взаємодію користувача із системою. Через веб-дашборди, мобільні застосунки або спеціалізовані панелі з сенсорними дисплеями користувач отримує графіки споживання, повідомлення про перевищення лімітів і можливість дистанційно керувати пристроями (вмикати чи вимикати розетки). Інтерфейс взаємодіє зі сховищем через API або веб-сервіси, що гарантує оновлення даних у режимі реального часу та гнучкість налаштувань доступу [7].

Модульна організація архітектури дає змогу комбінувати різні протоколи зв’язку, апаратні засоби й програмні платформи для побудови як простих локальних систем, так і масштабованих хмарних рішень для ефективного моніторингу й оптимізації енергоспоживання.

1.3.1. Home Assistant як центральна платформа моніторингу

На сьогодні є багато різноманітних вирішень для забезпечення функціоналу платформи інтернету речей, більшість з них має наступні недоліки. 1 - низький рівень інтеграції, 2 – тісна пов’язаність з виробником апаратного або програмного забезпечення, 3 – закритість коду, 4 – низький рівень технічної

підтримки, 5 – вартість. Одним з таких прикладів є закрита комерційна платформа Apple Homekit, а відкритим аналогом є Home Bridge. Проте їм притаманні наведені недоліки. Таких недоліків не має платформа Home Assistant.

Home Assistant — це відкрита платформа для автоматизації «розумного дому», що володіє гнучким ядром і модульною архітектурою. Вона дозволяє об’єднати в єдину систему сотні різних IoT-пристроїв і сервісів, від сенсорів температури й розеток до складних аудіосистем. Завдяки використанню YAML-конфігурацій і графічного інтерфейсу Lovelace, Home Assistant надає користувачу можливість налаштовувати дашборди, автоматизації та сцени без залучення зовнішніх хмарних сервісів.

Серед ключових переваг Home Assistant варто відзначити повну локальну обробку даних (мінімальна залежність від інтернету), багатий набір вбудованих інтеграцій (понад 2000 на поточний момент) та можливість розширення через додатки (Add-ons), які запускаються під керуванням Supervisor. Платформа підтримує роботу на різному апаратному рівні — від невеликих SBC (Raspberry Pi, Odroid) до віртуальних машин і контейнерів Docker.

Особливістю Home Assistant є система подій і станів, у межах якої кожен пристрій представляється як «сутність» (entity) з власними атрибутами та історичними даними. Це дає змогу реалізувати складні сценарії автоматизації через простий YAML-скрипт чи графічний редактор, а також будувати детальні аналітичні графіки споживання електроенергії.

Використання Home Assistant в проєкті моніторингу електроспоживання на базі ESP32 дозволяє централізовано збирати й обробляти показники напруги, струму й потужності, керувати розетками, а також поєднувати локальні IoT-пристрої з віддаленими веб-інтерфейсами та мобільними додатками без значних додаткових витрат на розробку інфраструктури.

1.3.2. Вибір хмарної або локальної платформи

При розробці системи моніторингу споживання електроенергії одним із ключових аспектів є вибір платформи для збору, обробки та збереження даних.

Існують два основних підходи: використання хмарної платформи або розгортання системи на базі локальної мережі. Обидва варіанти мають свої переваги та недоліки, що безпосередньо впливають на функціональність, масштабованість і безпеку системи.

1.3.2.1.Хмарні сервіси

Хмарні платформи, зокрема ті, що забезпечуються Tuya Cloud, відкривають можливості для віддаленого моніторингу та керування пристроями у системах автоматизації, дозволяючи користувачу отримувати дані незалежно від його місцезнаходження. Завдяки централізованому зберіганню даних, користувач може контролювати пристрої через інтернет, що сприяє зручному управлінню розумними пристроями та розширенню функціональності системи (рис. 1.2).

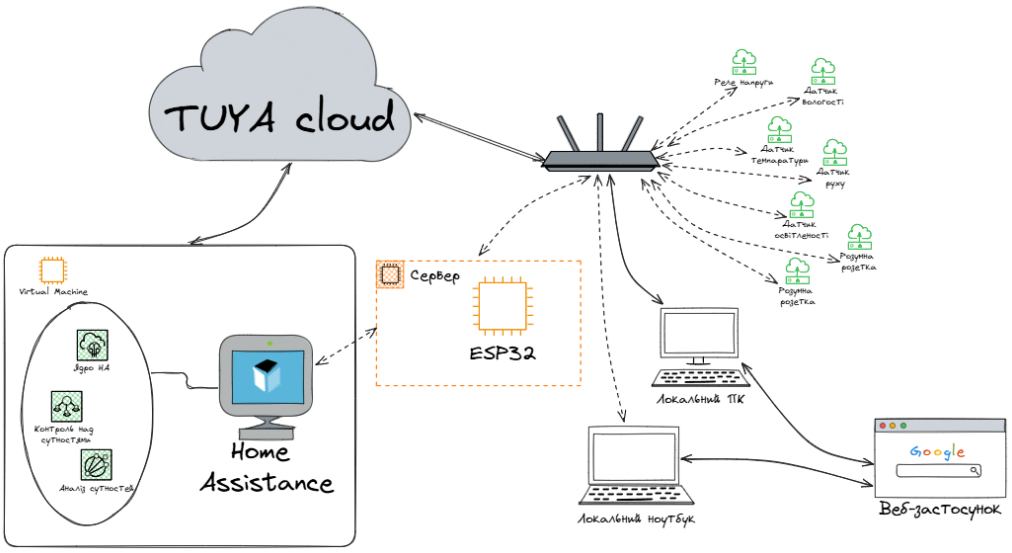

Рисунок 1.2 – Схема використання ESP32 разом з Home Assistant через Tuya Cloud

Проте, використання хмарних сервісів супроводжується значними ризиками безпеки. Дані, що передаються і зберігаються у Tuya Cloud, можуть стати ціллю несанкціонованого доступу, якщо механізми шифрування та

автентифікації не працюють належним чином. Це збільшує ймовірність отримання зловмисниками доступу до конфіденційної інформації або керування пристроями, що може призвести до втрати контролю над системою та завдати фінансових або репутаційних збитків.

Хмарні рішення забезпечують широкі можливості для масштабування — вони здатні обробляти великі обсяги даних і підтримувати інтеграцію з численними пристроями, що відкриває шлях до застосування передових алгоритмів аналізу даних та створення прогнозних моделей. Проте, масштабованість і зручність взаємодії повинні балансуватися з необхідністю забезпечення високого рівня безпеки. У зв'язку з цим важливо впроваджувати сучасні методи шифрування, а також механізми двофакторної автентифікації, щоб зменшити ризики несанкціонованого доступу до даних.

Зважаючи на ці аспекти, хоча хмарні платформи, такі як Tuya Cloud, пропонують зручність віддаленого моніторингу та розширену аналітику, критично важливе забезпечення додаткових заходів безпеки. Це дозволяє поєднувати функціональність хмарних рішень із перевагами локальних API для підвищення надійності системи моніторингу споживання електроенергії.

1.3.2.2.Локальні сервіси

Локальні платформи моніторингу, як-от реалізовані з використанням Local Tuya, дозволяють забезпечити повний контроль над IoT-пристроями в межах домашньої мережі без необхідності взаємодії з хмарними сервісами. У такому випадку дані з пристроїв не залишають локальну мережу, що значно знижує ризики витоку інформації та підвищує рівень безпеки системи загалом.

Цей підхід гарантує низьку затримку обміну даними, оскільки вся обробка відбувається локально. Система здатна миттєво реагувати на події, що критично важливо для сценаріїв, де потрібна висока швидкодія — наприклад, у випадках аварійного вимкнення навантаження або термінового сповіщення користувача. Крім того, локальне рішення працює незалежно від якості або наявності

інтернет-з’єднання, що забезпечує стабільну роботу навіть при зовнішніх мережевих перешкодах (рис. 1.3).

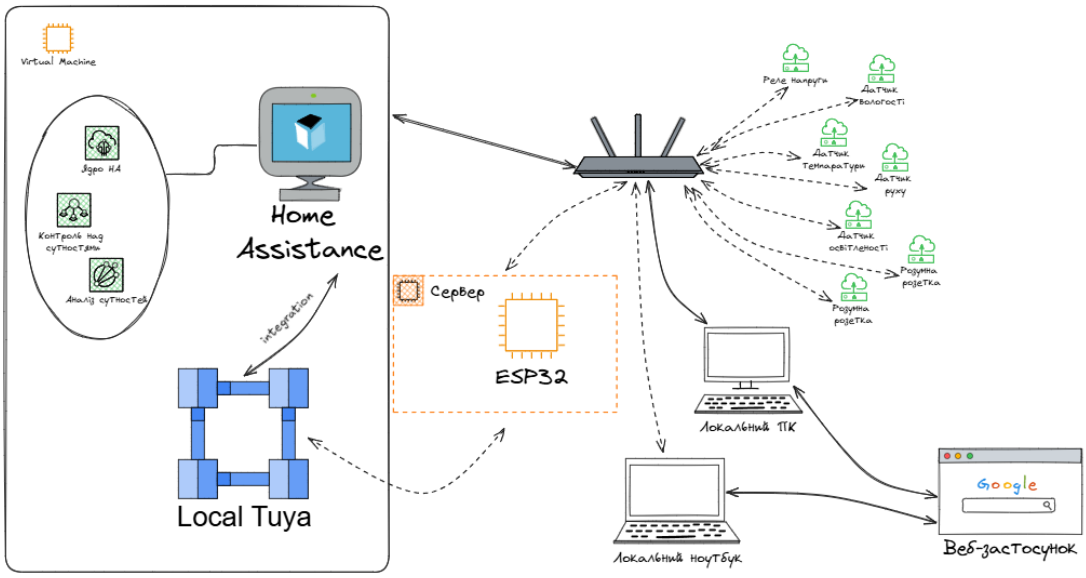

Рисунок 1.3 – Схема використання ESP32 разом з Home Assistant з інтеграцією Tuya Local

Реалізація через Local Tuya дозволяє зберігати всю логіку керування, аналізу та автоматизації в межах домашньої інфраструктури, що робить систему автономною і захищеною від впливу сторонніх сервісів. Такий підхід забезпечує високу гнучкість у налаштуванні та повну прозорість усіх процесів, що особливо важливо при розробці рішень для енергоефективного моніторингу споживання електроенергії.

Таким чином, локальні сервіси — це надійна альтернатива хмарним платформам, орієнтована на безпеку, швидкодію та автономність.

РОЗДІЛ 2 ПРОЄКТНА ЧАСТИНА

2.1. Побудова та опис структурної схеми

Структурна схема хмарної системи моніторингу електроенергії відображає основні функціональні блоки пристрою та взаємозв'язки між ними. Вона показує логічну організацію системи, що дозволяє зрозуміти принцип її роботи та взаємодію компонентів.

Структурна схема складається з таких основних блоків (рис. 2.1):

– Мікроконтролер ESP32 – центральний елемент системи, який забезпечує обробку даних, що надходять від сенсорів, та керує виконавчими механізмами. ESP32 приймає сигнали від кнопок, обробляє їх за допомогою програмного алгоритму та виводить результат на екран, а також відповідає за взаємодію з Home Assistant [4].

– Кнопка button_Socket підключена до контакту D15. Служить для керування комп’ютерною розеткою (наприклад, увімкнення або вимкнення живлення ПК).

– Кнопка monitor_Socket підключена до контакту D21. Призначена для керування розеткою монітора або іншого периферійного пристрою.

– Кнопка show_stats підключена до контакту D22. Активує режим перегляду статистики енергоспоживання на OLED-дисплеї.

– Блок живлення забезпечує стабільне живлення всіх компонентів системи. Для живлення використовується адаптер на 3.3 В, який підключається до ESP32, а звідти напруга подається на інші компоненти.

Таким чином, структура схеми демонструє просту, але ефективну апаратну конфігурацію, яка дозволяє користувачу здійснювати локальне керування IoT-розетками та переглядати інформацію без потреби у зовнішніх пристроях.

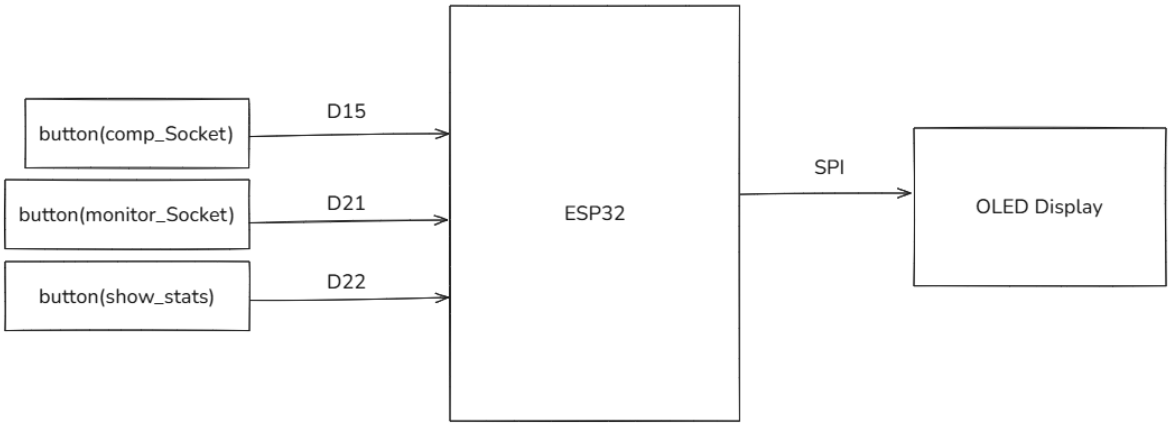

Рисунок 2.1 – Структурна схема

2.2. Аналіз обраного мікроконтролера ESP32

У цьому проєкті як основну апаратну платформу обрано мікроконтролер ESP32, який демонструє високу продуктивність та енергоефективність, необхідні для реалізації системи моніторингу споживання електроенергії. ESP32 заснований на двоядерному процесорі Tensilica Xtensa LX6 з тактовою частотою до 240 МГц, що забезпечує швидку обробку даних та можливість виконання одночасно кількох завдань, таких як зчитування сигналів з кнопок, відображення інформації на OLED-дисплеї та обмін даними з Home Assistant. Вбудований модуль Wi-Fi дозволяє ESP32 взаємодіяти з локальними системами керування, використовуючи HTTP-запити або спеціалізовані API, що є ключовим для інтеграції з платформою Home Assistant і забезпечення локальної взаємодії [7].

ESP32 має велику кількість універсальних виводів, що дозволяє підключити три фізичні кнопки для керування різними функціями системи, а також OLED-дисплей, який відображає поточний стан розеток, статистичні дані споживання електроенергії або повідомлення для користувача. Підтримка інтерфейсів SPI та I²C дозволяє легко інтегрувати дисплей і інші периферійні пристрої, що спрощує апаратну конфігурацію системи та зменшує складність з’єднань.

Окрім потужних апаратних характеристик, ESP32 відзначається енергоефективністю, що особливо важливо для систем, орієнтованих на моніторинг споживання електроенергії, де оптимізація ресурсів є критичним фактором. Завдяки можливостям переходу у режими зниженої активності мікроконтролер забезпечує продовжену роботу в автономному режимі, що суттєво підвищує ефективність системи.

Також важливою перевагою ESP32 є його широка підтримка у спільноті розробників та доступність численних бібліотек і прикладів для інтеграції з сучасними платформами розумного дому. Використання середовища розробки PlatformIO спрощує процес створення, налагодження та завантаження програмного забезпечення на мікроконтролер. Інтеграція з Home Assistant дозволяє забезпечити центральне управління системою, а також віддалений моніторинг, що відповідає сучасним вимогам до енергоефективних рішень [7].

Таким чином, ESP32 є оптимальним вибором для впровадження системи моніторингу споживання електроенергії завдяки своїм широким апаратним можливостям, підтримці бездротових технологій та високій енергоефективності, що робить його центральним елементом у створенні сучасної IoT-платформи для моніторингу споживання електроенергії.

2.3. Аналіз процесу розгортання та налаштування Home Assistant

Home Assistant виступає як центральна платформа для інтеграції та централізованого управління IoT-пристроями, що використовуються в системі моніторингу споживання електроенергії. Основна мета розгортання Home Assistant полягає у створенні зручного та гнучкого середовища, яке дозволяє легко додавати нові пристрої, аналізувати дані в реальному часі та надавати інтуїтивно зрозумілий веб інтерфейс для кінцевого користувача. Завдяки своїй модульній архітектурі та широкій підтримці різних протоколів зв'язку платформа забезпечує можливість керування як місцевими пристроями за допомогою локальних API, так і віддаленим доступом через Інтернет. Home

Assistant дозволяє інтегрувати дані з ESP32, що відповідають за моніторинг споживання енергії, і надавати це інформаційне поле в режимі реального часу для оперативного прийняття рішень щодо управління пристроями системи [8].

2.3.1. Обґрунтування вибору платформи для встановлення

Для розгортання Home Assistant існує декілька варіантів, кожен з яких має свої переваги та недоліки, зокрема Raspberry Pi, ПК та Docker. Один із найпопулярніших варіантів — встановлення платформи на Raspberry Pi. Цей підхід приваблює низькою ціною, компактністю та енергоефективністю, що ідеально підходить для постійного моніторингу в умовах "розумного дому". Проте при збільшенні навантаження або інтеграції великої кількості пристроїв може виникнути потреба у використанні більш потужного обладнання.

Іншим варіантом є використання ПК або серверного обладнання, що забезпечує вищу продуктивність та розширені можливості для зберігання та аналізу даних, але зазвичай пов'язане з більшими витратами і споживанням енергії. Ще одним популярним підходом є розгортання Home Assistant у вигляді контейнера Docker, який дозволяє легко масштабувати систему, ізолювати середовище розробки та легко інтегруватися з іншими сервісами.

У межах даного проєкту, як середовище розробки, було вирішено розгорнути Home Assistant на віртуальній машині (VM), що працює на локальному комп’ютері. Такий підхід поєднує в собі гнучкість налаштувань, достатню обчислювальну потужність та зручність у керуванні ресурсами. Віртуальна машина дозволяє швидко масштабувати систему в разі потреби, а також легко виконувати резервне копіювання та відновлення середовища. У порівнянні з Raspberry Pi, цей варіант забезпечує кращу продуктивність при взаємодії з великою кількістю інтеграцій, а на відміну від Docker-контейнера — спрощує початкове налаштування для повноцінного доступу до усіх функцій Home Assistant OS. Таким чином, під час створення прототипу системи, вибір на користь віртуальної машини виявився оптимальним з точки зору надійності, керованості та ефективності роботи всієї системи [8].

2.3.2. Встановлення Home Assistant

Процес встановлення Home Assistant на віртуальну машину включає кілька послідовних етапів, які забезпечують правильну підготовку середовища, розгортання системи та її запуск [9].

На першому етапі було створено нову віртуальну машину у середовищі віртуалізації VMware. Для цього було обрано образ операційної системи Home Assistant OS, який офіційно надається на сайті Home Assistant. Образ має формат VMDK, що дозволяє швидко імпортувати його до обраного гіпервізора. Після завантаження варто розпакувати архів.

Після створення нової VM із виділенням необхідних ресурсів (рекомендовано не менше 2 ГБ оперативної пам'яті та 32 ГБ дискового простору), було здійснене підключення до мережі з типом адаптера Bridged Adapter або NAT, що дало можливість Home Assistant мати доступ до локальної мережі та Інтернету для завантаження оновлень та інтеграцій. Також варто зайти у розділ Hard Disk(SCSI) та вибрати розташування образу Home Assistant (рис. 2.2).

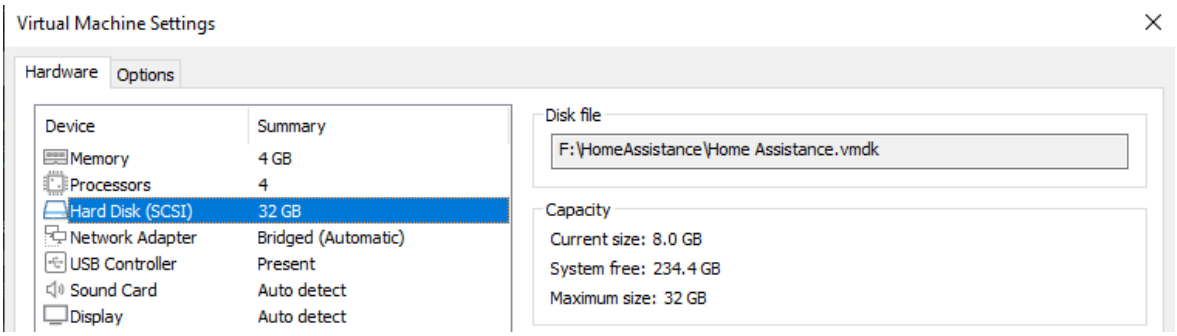

Рисунок 2.2 – Вікно налаштування віртуальної машини з образом Home Assistant

Після запуску віртуальної машини Home Assistant автоматично завантажується, виконує початкове налаштування і відкриває доступ до веб-інтерфейсу за локальною IP-адресою: 192.168.0.107:8123 (рис. 2.3). Перший

запуск може тривати до 20 хвилин, оскільки система встановлює основні компоненти.

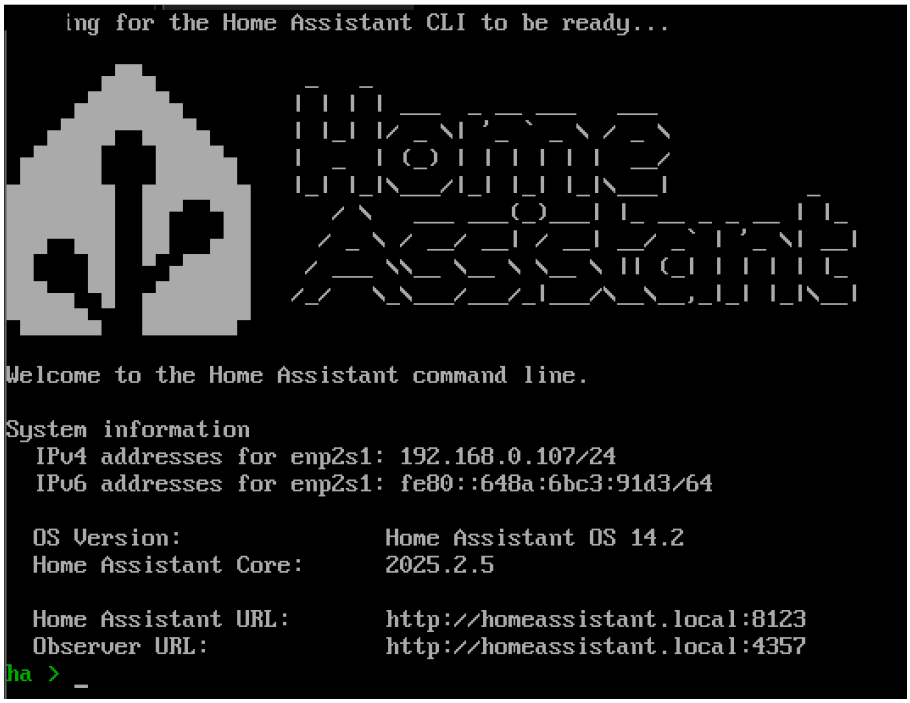

Рисунок 2.3 – Вікно терміналу Home Assistant

Далі у веб-браузері відкривається інтерфейс початкової конфігурації, де створюється обліковий запис користувача, налаштовується регіон, часова зона, мова інтерфейсу та базові параметри системи. Після цього Home Assistant готовий до подальшого налаштування інтеграцій, автоматизацій та пристроїв.

Перевагою даного методу є автономність Home Assistant OS, що включає в себе не тільки основну систему, але й Supervisor — інструмент для зручного керування оновленнями, резервним копіюванням, установленням додатків тощо. Завдяки цьому забезпечується стабільність роботи системи та її зручна підтримка без необхідності ручної конфігурації на рівні ОС [9].

У результаті розгортання Home Assistant на віртуальній машині дозволило швидко створити стабільне середовище для побудови системи моніторингу електроспоживання з можливістю гнучкого масштабування та налаштування.

2.3.3. Початкове налаштування інтерфейсу Home Assistant

Після завершення встановлення Home Assistant та створення облікового запису користувача виконується початкове налаштування інтерфейсу, яке є важливою частиною адаптації системи під конкретні потреби користувача. На цьому етапі конфігурується вигляд головної панелі, додаються перші картки з інформацією, налаштовуються області та кімнати, а також встановлюються необхідні інтеграції [9].

Першочергово Home Assistant автоматично виявляє деякі пристрої у локальній мережі — зокрема роутери, телевізори, медіаплеєри або розумні розетки, які підтримують протоколи автоматичного виявлення (наприклад, mDNS або SSDP). Якщо знайдені пристрої сумісні, вони пропонуються до встановлення як інтеграції.

Після цього виконується налаштування інформаційної панелі (Dashboard). Інтерфейс Home Assistant побудований на основі Lovelace — гнучкої системи створення карток, які відображають дані з пристроїв або дозволяють керувати ними.

Завдяки широкому набору вбудованих і сторонніх карток (графіки, гістограми, перемикачі, кнопки, сповіщення тощо), інтерфейс можна адаптувати до будь-якого рівня складності — від простого моніторингу одного пристрою до повноцінного центру управління розумним будинком [9].

Таким чином, початкове налаштування інтерфейсу Home Assistant дозволяє створити інтуїтивно зрозумілу панель управління для користувача, яка відображає найважливішу інформацію та забезпечує ефективну взаємодію з системою моніторингу електроспоживання.

2.3.4. Додавання інтеграції Tuya

Інтеграція Tuya в Home Assistant дозволяє з'єднати систему з IoT-пристроями, які використовують технології Tuya для бездротової передачі даних та керування. Цей крок є критично важливим для забезпечення взаємодії між пристроями, наприклад, розумними розетками, реле або іншими елементами

системи моніторингу споживання електроенергії, і центральною платформою керування [10].

Для додавання інтеграції необхідно скористатися вбудованим інтерфейсом Home Assistant. Перш за все, у меню «Налаштування» обирається пункт «Пристрої та служби», після чого клацають «Додати інтеграцію». У рядку пошуку вводиться «Tuya», що дозволяє Home Assistant знайти офіційну інтеграцію для пристроїв Tuya (рис. 2.4).

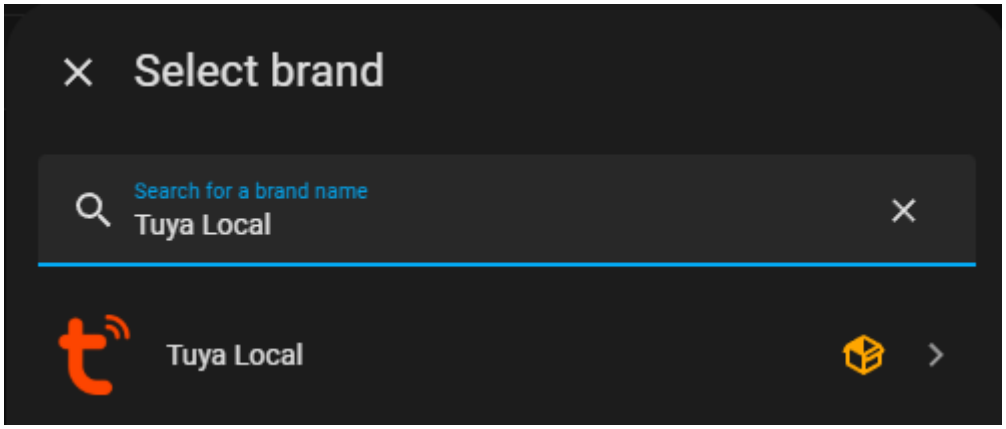

Рисунок 2.4 – Встановлення офіційної інтеграції Tuya Local

При використанні Local Tuya даний підхід дозволяє налаштувати обмін даними безпосередньо по локальній мережі, що сприяє зниженню затримок, підвищенню безпеки обміну даними та уникненню залежності від хмарних серверів. Після успішної аутентифікації Home Assistant починає спілкування з пристроями Tuya: дані про стан пристроїв автоматично завантажуються у систему, а також оновлюються у режимі реального часу, що забезпечує можливість їх оперативного керування.

У процесі інтеграції важливо переконатися, що всі пристрої знаходяться у межах однієї локальної мережі і правильно налаштовані для роботи з обраною інтеграцією Tuya. Home Assistant пропонує зручні інструменти для візуалізації підключених пристроїв, дозволяючи користувачеві у кілька натискань миші налаштовувати їхні параметри, створювати сценарії автоматизації та переглядати історію змін. Таким чином, інтеграція Tuya є невід'ємною

складовою загальної екосистеми системи моніторингу електроенергії, що підвищує її гнучкість і масштабованість, дозволяючи легко адаптуватися до різноманітних умов експлуатації [10].

2.3.5. Використання Local Tuya

У межах реалізації системи моніторингу електроспоживання було обрано інтеграцію Local Tuya як ефективну альтернативу хмарному рішенню Tuya Cloud. На відміну від стандартної взаємодії через інтернет, Local Tuya дозволяє з’єднуватися з розумними пристроями напряму через локальну мережу, минаючи зовнішні сервери. Такий підхід значно підвищує безпеку, швидкодію та стабільність системи.

Однією з основних переваг Local Tuya є збереження повного контролю над даними. Усі сигнали передаються локально, без участі зовнішніх хостів, що мінімізує ризик несанкціонованого доступу або втрати інформації. Це особливо важливо для систем, які оперують чутливими даними або працюють у критичних середовищах, де недопустимі затримки чи залежність від стабільності інтернет-з'єднання.

Інтеграція з Home Assistant здійснюється через встановлення відповідного плагіна, що дозволяє автоматично виявляти сумісні пристрої Tuya у мережі. Після отримання необхідних параметрів (ID пристрою, локального ключа та IP-адреси), Home Assistant отримує можливість керувати розетками, реле, датчиками температури, вологості, освітленості та іншими пристроями напряму, без залучення хмарної інфраструктури [10].

Для додавання пристроїв до локальної мережі Local Tuya варто відкрити інтеграцію Local Tuya. Далі натискаю кнопку Додати пристрій. Відкривається вікно, де є можливість вибрати спосіб підключення пристрою. Перший спосіб це здійснити авторизацію через Smart Life(додаток для керування розумними пристроями з смартфона), другий спосіб – ручне введення даних та налаштування пристроїв. Для простоти та безпеки було використано перший спосіб з використанням додатку Smart Life та натискаю Далі (рис. 2.5).

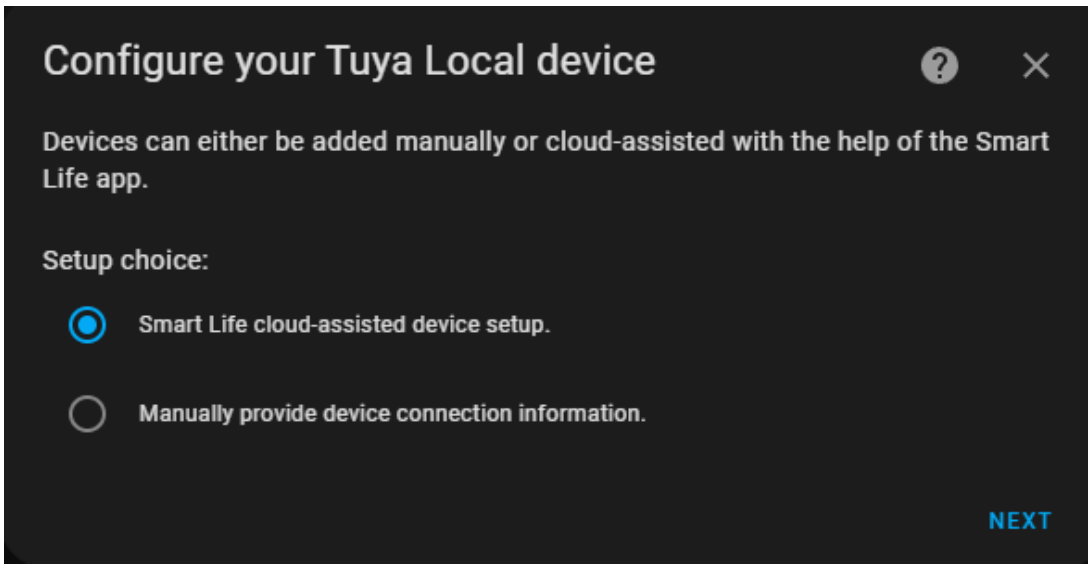

Рисунок 2.5 – Вікно додавання пристроїв у інтеграції Local Tuya

Далі відкривається вікно, де варто ввести код користувача, який можна знайти у додатку Smart Life зайшовши в Налаштування > Аккаунт та безпека. Далі варто відсканувати QR-код з додатку, щоб підтвердити особу. З'являється вікно з списком пристроїв, які зареєстровані у додатку. Варто вибрати один пристрій з списку та натиснути Додати. Після додавання пристрою, він буде відображений на головній сторінці інтеграції Local Tuya (рис. 2.6).

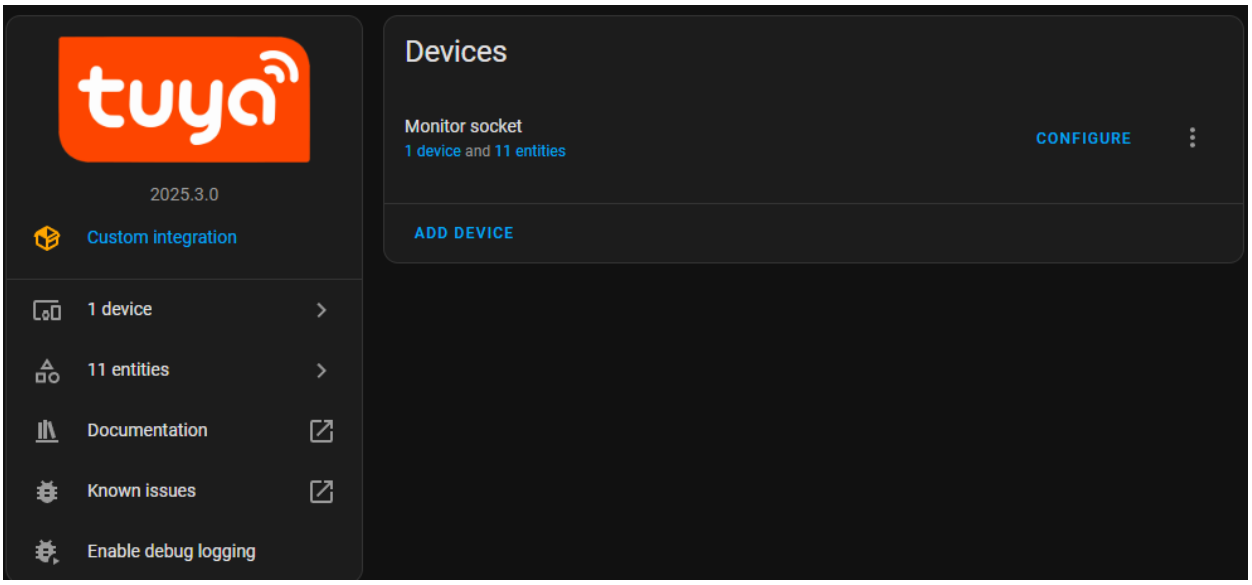

Рисунок 2.6 – Головна сторінка інтеграції Local Tuya разом з доданим пристроєм через мобільний додаток Smart Life

У проєкті використання Local Tuya дало змогу налагодити повністю автономну систему енергомоніторингу, де збір даних, аналіз і управління здійснюються всередині локального середовища. Такий підхід дозволяє швидко реагувати на зміни в енергоспоживанні, створювати сценарії автоматизації, візуалізувати показники в інтерфейсі Home Assistant і забезпечувати надійність функціонування системи навіть за умов повної відсутності доступу до інтернету.

2.4. Розробка програмного забезпечення для ESP32

2.4.1. Вибір інструменту розробки

У цьому проєкті для написання та налагодження прошивки ESP32 обрано Arduino IDE. Цей інтегрований середовище розробки відрізняється своєю простотою і водночас достатньою гнучкістю для реалізації IoT-рішень.

По-перше, Arduino IDE є сумісним із різними операційними системами: воно працює на Windows, macOS та Linux, що дозволяє команді розробників використовувати звичну їм операційну систему без додаткових обмежень.

По-друге, інтерфейс середовища максимально зрозумілий для новачків, але водночас містить усі необхідні інструменти для професійного кодування: підсвічування синтаксису, автодоповнення, менеджер бібліотек і вбудований Serial Monitor для відлагодження.

Третя важлива перевага — велика кількість готових бібліотек і прикладів. Відразу після встановлення IDE розробнику доступний широкий вибір модулів для роботи з Wi-Fi, HTTP-клієнтом, OLED-дисплеями та кнопками, що значно скорочує час розробки і тестування.

Нарешті, Arduino IDE є відкритим і безкоштовним програмним забезпеченням із розвиненою спільнотою: численні форуми, навчальні матеріали та документація дозволяють швидко вирішувати будь-які питання та освоювати нові можливості платформи.

З огляду на ці переваги, Arduino IDE забезпечує оптимальний баланс між простотою використання та потужністю функцій, що робить його ідеальним

вибором для реалізації прошивки ESP32 у рамках системи моніторингу споживання електроенергії.

2.4.2. Створення локального сервера на ESP32

Наявність локального HTTP-сервера на платі ESP32 дозволяє організувати зручний інтерфейс взаємодії як із веб-клієнтом (браузером), так і з внутрішніми апаратними елементами: кнопками, дисплеєм та мережею. У рамках цього проєкту реалізовано невеликий, але повнофункціональний HTTP-сервер, побудований на базі бібліотеки WebServer, що надає простий API для створення та обробки маршрутів (routes), формування відповідей і керування пристроями через REST-подібні запити.

2.4.2.1. Розгортання HTTP-сервера на ESP32

Сервер ініціалізується у функції setup() викликом конструктора: `WebServer server(80);` де 80 — стандартний HTTP-порт. Після встановлення Wi-Fi-з'єднання (`WiFi.begin(...)` і цикл очікування статусу WL_CONNECTED) виконується виклик `server.begin()`, що запускає прослуховування вхідних HTTP-запитів: `server.begin(); Serial.println("HTTP server started")`. Цей підхід забезпечує асинхронне оброблення запитів у фоні, тоді як в loop() постійно викликається `server.handleClient()`, яка перевіряє наявність нових HTTP-запитів і спрямовує їх на відповідні обробники.

2.4.2.2. Роутинги для керування засобами HTTP

Для кожного шляху (route) налаштовано свій обробник через метод `server.on()`:

– `"/"` — головна сторінка з HTML-інтерфейсом, реалізована функцією `handleRoot()`. Віддає готову HTML-сторінку з кнопками та графіком.

– "/data" — API-endpoint, що повертає JSON-об'єкт з останніми показниками потужності, напруги та струму, реалізований у handleData().

– "/switch/on" та "/switch/off" — шляхи для вмикання та вимикання розеток. Обидва використовують анонімні лямбда-функції, які викликають switchPlug(true/false) та повертають простий текстовий статус ON або OFF.

– "/setEntity" — шлях для зміни цільової сутності entityName (наприклад, monitor або comp), що дозволяє переключати контекст запитів та статистики. При наявності параметра name у GET-рядку налаштовує entityName і повертає JSON {success: true}.

Усі ці маршрути побудовані за моделлю GET (включно з викликами, які виконують POST-операції всередині), що спрощує взаємодію зі звичайним браузером або JavaScript-клієнтом (рис. 2.7).

```
server.on("/", handleRoot);
server.on("/data", handleData);
server.on("/switch/on", []() {
  switchPlug(true);
  server.send(200, "text/plain", "ON");
});
server.on("/switch/off", []() {
  switchPlug(false);
  server.send(200, "text/plain", "OFF");
});
server.on("/setEntity", []() {
  if (server.hasArg("name")) {···
  } else {
    server.send(400, "application/json", "{\"error\":\"Missing name parameter\"}");
  }
});
```

Рисунок 2.7 – Код задавання шляхів(роутинг) з обробниками

2.4.2.3. Формування JSON-відповідей

Для віддачі структурованої інформації використовується бібліотека ArduinoJson. У функції handleData() створюється

`StaticJsonDocument<200>` і заповнюється полями *`jsonDoc["power"], jsonDoc["voltage"] та jsonDoc["current"]`* (рис. 2.8).

```
StaticJsonDocument<200> jsonDoc;
jsonDoc["power"] = power;
jsonDoc["voltage"] = voltage;
jsonDoc["current"] = current;

String response;
serializeJson(jsonDoc, response);
server.send(200, "application/json", response);
```

Рисунок 2.8 – Створення StaticDocument з полями power, voltage та current

Аналогічно, у маршруті `"/setEntity"` формується JSON-відповідь про успіх операції (рис. 2.9).

```
StaticJsonDocument<200> jsonDoc;
jsonDoc["success"] = true;
String response;
serializeJson(jsonDoc, response);
server.send(200, "application/json", response);
```

Рисунок 2.9 – Створення StaticDocument з полем success

Такий підхід гарантує коректне кодування даних, легку інтеграцію з фронтендом (JS fetch/AJAX) та можливість подальшого розширення полів JSON-об’єктів без суттєвих змін у логіці сервера.

2.4.2.4. Зв’язок з внутрішніми компонентами ESP

Локальний сервер тісно взаємодіє з апаратною частиною:

– GPIO: зчитування трьох кнопок здійснюється викликами `digitalRead(buttonPinX)`, порівняннями з попереднім станом `lastButtonReadingX` і перемиканням логічних змінних `buttonStateX`. У залежності від натискання викликаються відповідні обробники, що оновлюють entityName, виконують `switchPlug()` або відображають статистику.

– WiFi: до запуску сервера ESP32 підключається до локальної мережі через `WiFi.begin(ssid, password)` і чекає успішного з'єднання, після чого виводить IP-адресу в консоль та приймає HTTP-запити на цю адресу.

– OLED-дисплей: перед обробкою маршруту виводиться головний екран із поточними станами пристроїв, а після натискання кнопок формується новий вміст дисплея (статус перемикання або статистика) за допомогою методів `u8g2.clearBuffer()`, `u8g2.setCursor()`, `u8g2.print()` та `u8g2.sendBuffer()`.

– HTTPClient: у функціях `switchPlug()` та `getSensorValue()` використовується клас HTTPClient для формування запитів до Home Assistant, додавання заголовків (Authorization: Bearer) та отримання/відправки JSON-даних.

Таким чином, розгортання локального HTTP-сервера на ESP32 поєднує маршрутизацію веб-запитів, формування JSON-відповідей та прямий контроль апаратної частини, забезпечуючи зручний і гнучкий інтерфейс для моніторингу та керування IoT-пристроями в реальному часі.

2.5. Прототипування компоненти локального керування

Для забезпечення швидкого й інтуїтивно зрозумілого керування системою без залучення зовнішніх пристроїв передбачено локальний інтерфейс на базі трьох фізичних кнопок. Ці кнопки дозволяють у реальному часі вмикати та вимикати дві розетки ("Comp Socket" та "Monitor Socket") і викликати на екран OLED-дисплея докладну статистику споживання енергії.

2.5.1. Підключення локального керування до ESP32

У проєкті локальне керування побудоване на трьох фізичних кнопках і OLED-дисплеї, які прямо з'єднані з GPIO-виводами ESP32 і живляться від його 3,3-вольтового виходу. Кожна кнопка під'єднана через один контакт до свого цифрового входу ESP32 (відповідно, перша — до GPIO15, друга — до GPIO21,

третя — до GPIO22), а другим контактом завжди замикається на загальну шину GND. Завдяки вбудованому підтягувальному резистору (режим INPUT_PULLUP) в ESP32, у стані спокою лінії утримуються в логічній високій позиції, а при натисканні кнопка замикає лінію на «землю» і змінює рівень на LOW, що дозволяє однозначно фіксувати подію натискання без використання зовнішніх резисторів (рис. 2.10).

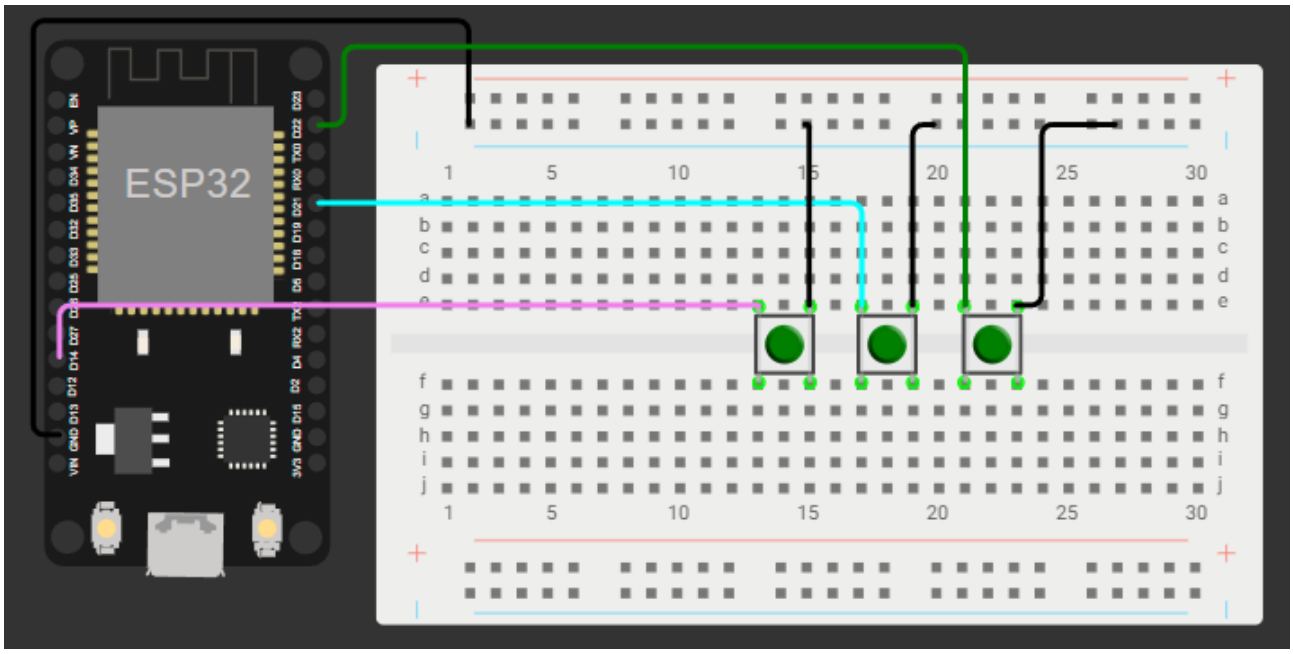

Рисунок 2.10 – Схема підключення трьох кнопок до ESP32

OLED-дисплей SH1107 з роздільною здатністю 128×128 пікселів підключений до ESP32 через апаратний інтерфейс SPI. Лінія вибору чіпа (CS) приєднана до GPIO5, сигнали DC і RST — до GPIO16 і GPIO17 відповідно, а для обміну даними за протоколом SPI використовуються виводи HSPI: SCK (GPIO18) та MOSI (GPIO23). Подача живлення здійснюється також від 3.3В виходу ESP32, а всі «земляні» контакти дисплея та кнопок об’єднані в одну шину GND, що гарантує стабільність і безпечність роботи без потреби в зовнішніх стабілізаторах.

Після апаратного з’єднання елементів необхідно виконати виклик u8g2.begin() для ініціалізації дисплея та вміщення його в програмний буфер.

Кожне оновлення інтерфейсу відбувається через формування вмісту за допомогою методів `u8g2.setCursor()` і `u8g2.print`. Така проста, але надійна апаратна схема дозволяє ESP32 миттєво реагувати на події кнопок і відобразити свіжі дані на екрані в реальному часі.

2.5.2. Програмування логіки локального керування

У головному циклі програми для кожної з трьох кнопок відбувається постійне зчитування рівнів із відповідних GPIO-входів і порівняння з попереднім станом, що дозволяє фіксувати момент реального натискання при переході від HIGH до LOW. Коли виявляється, що кнопка, призначена для перемикання розетки комп'ютера, опустилася в низький рівень, змінна, яка зберігає її поточний стан, інвертується і викликається функція `switchPlug()`, яка надсилає відповідний HTTP-запит до Home Assistant (рис. 2.11).

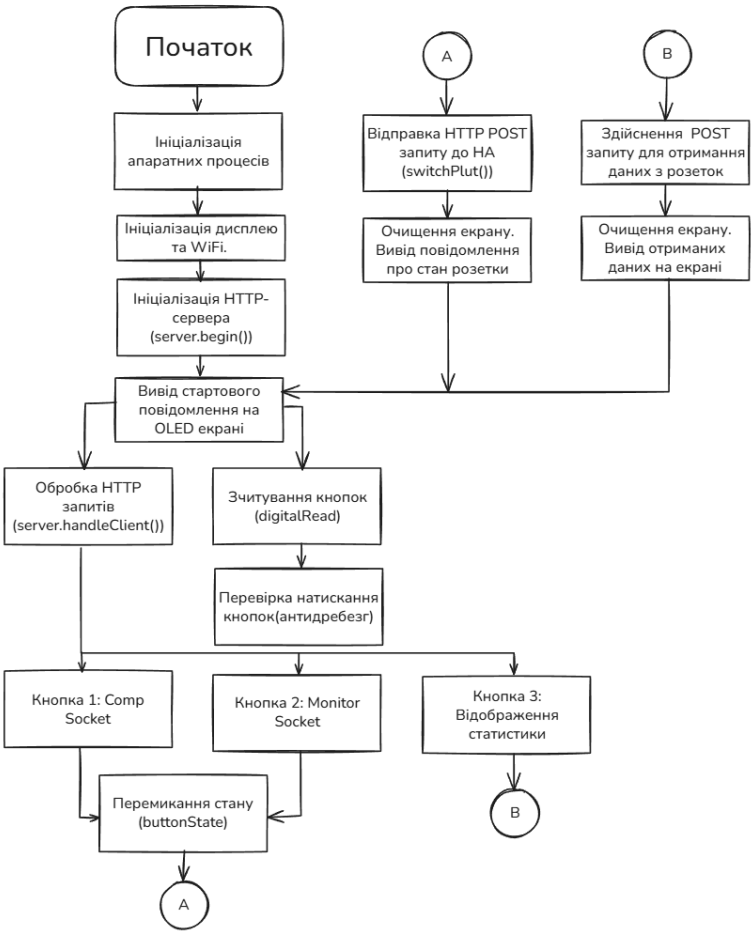

Рисунок 2.11 – Представлення алгоритму роботи локального керування

Після цього дисплей очищується та виводиться повідомлення про новий стан «Comp turned on» або «Comp turned off». Аналогічна логіка застосовується й до другої кнопки: при її натисканні інвертується стан розетки монітора, на дисплеї з'являється коротке повідомлення про «Monitor now on/off», а команда передається у Home Assistant за потреби.

Третя кнопка, призначена для відображення статистики, ініціює послідовний виклик функції `getSensorValue()` для сенсорів потужності, напруги і струму спочатку для монітора, а потім для комп'ютера. Після отримання значень на екрані формується докладна табличка з параметрами P, V, I для обох пристроїв. Наприкінці обробки кожного натискання оновлюється запис про останнє зчитування кнопки, що виключає багаторазові спрацьовування при тривалому утриманні. Завдяки такому підходу забезпечується чітка та надійна робота локального інтерфейсу керування без додаткових зовнішніх компонентів.

2.6. Робота з дисплеєм

Для інтеграції OLED-дисплею SH1107 із ESP32 обраний апаратний інтерфейс SPI, що гарантує високу швидкодію при оновленні графічного буфера. До виводу 3.3 В плати ESP32 підводиться живлення дисплея, а шина GND з'єднує «землі» обох пристроїв, забезпечуючи загальний опорний рівень.

Лінія вибору чіпа CS підключена до GPIO 5, що дозволяє мікроконтролеру активувати саме цей периферійний пристрій на шині SPI. Сигнал DC, який розрізняє команди від даних, приєднано до GPIO 16, а апаратне скидання RST — до GPIO 17, що дозволяє програмно ініціювати перезавантаження дисплея перед початком роботи. Основна передача даних здійснюється по HSPI: лінія SCK пов'язана з GPIO 18 (HSPI_CLK), а MOSI (дані) підключена до GPIO 23 (HSPI_MOSI). Після завершення апаратних з'єднань у функції setup() викликається метод `u8g2.begin()`, який налаштовує SPI-шину в потрібному режимі та ініціалізує контролер SH1107.

Подальші виклики `u8g2.clearBuffer()` і `u8g2.sendBuffer()` організовують оновлення зображення, а виклики `u8g2.setCursor()` разом із `u8g2.print()` дозволяють виводити текст і графіку. Така схема підключення забезпечує надійну роботу дисплея без додаткових перетворювачів рівня напруги та дає змогу ефективно використовувати весь простір екрана для відображення статусу розеток та статистики споживання.

2.6.1. Бібліотеки для роботи з дисплеєм

Для управління OLED-екраном SH1107 у проєкті використано бібліотеку U8g2, яка забезпечує універсальний інтерфейс для роботи з різними контролерами дисплеїв через SPI та I²C (рис. 2.12).

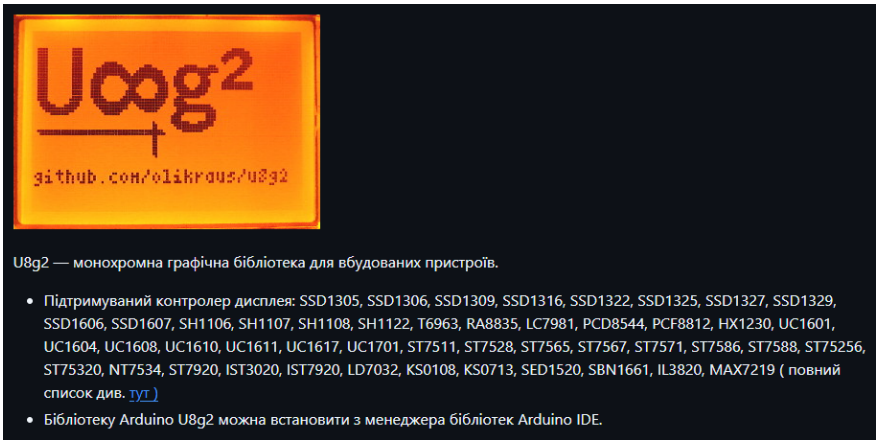

Рисунок 2.12 – Домашня сторінка бібліотеки U8g2

Вибір U8g2 обумовлений її високою оптимізацією пам’яті, широкою підтримкою різноманітних шрифтів і можливістю апаратної чи програмної ініціалізації інтерфейсу. У коді підключення відбувається через хедер `#include <U8g2lib.h>`, після чого створюється екземпляр драйвера з параметрами розміру екрана, конструктора для апаратного SPI та пінів управління (CS, DC, RST). Для відображення тексту й графіки використовуються методи `clearBuffer()`, `setFont()`, `drawGlyph()` і `print()`, а фінальна відправка буферу відбувається викликом `sendBuffer()` (рис. 2.13).

```
u8g2.clearBuffer();

u8g2.setFont(u8g2_font_unifont_t_symbols);
u8g2.drawGlyph(25, 25, 0x2603);

u8g2.setFont(u8g2_font_ncenB08_tr);
u8g2.setCursor(0, 50);
u8g2.print("Home page");

// Comp Socket
u8g2.setFont(u8g2_font_5x8_tr);
u8g2.setCursor(0, 70);
u8g2.print("Comp Socket: ");

u8g2.setCursor(80, 70);
u8g2.print(buttonState1 ? "On" : "Off");

// Monitor Socket
u8g2.setCursor(0, 80);
u8g2.print("Monitor Socket:  ");

u8g2.setCursor(80, 80);
u8g2.print(buttonState2 ? "On" : "Off");

u8g2.sendBuffer();
```

Рисунок 2.13 – Код використання бібліотеки у проєкті

2.6.2. Відображення статусу розеток

У головному циклі, перед перевіркою натискань кнопок, дисплей очищується викликом `u8g2.clearBuffer()`, після чого задається шрифт для заголовків і виводиться символ чи логотип (наприклад, `u8g2.drawGlyph(25, 25, 0x2603)`) разом із написом «Home page». Далі в менших шрифтах формується інформація про стан кожної з розеток: курсор встановлюється в координати Y=70 для першої розетки, виконується `u8g2.print("Comp Socket: ")` і праворуч від цього виводиться результат логічної змінної ("On" або "Off"). Аналогічна послідовність застосовується до другої розетки – напис «Monitor Socket: » формується на рядку Y=80, а потім відображається її поточний стан. Після завершення малювання обох рядків графічний буфер передається на екран викликом `u8g2.sendBuffer()`. Таке рішення забезпечує чіткий, легко читабельний вивід, що одразу інформує користувача про точний стан вихідних пристроїв без необхідності прокручування чи додаткових дій.

РОЗДІЛ 3 ПРАКТИЧНА ЧАСТИНА

3.1. Розгортання апаратних програмних компонент

Для створення повноцінного середовища моніторингу споживання електроенергії необхідно налаштувати Raspberry Pi як центральний вузол, на якому працюватиме Home Assistant, а також забезпечити підключення усіх апаратних компонентів (ESP32, OLED-дисплею, фізичних кнопок) і розумної розетки до цієї платформи.

3.1.1. Налаштування Raspberry Pi 4

Raspberry Pi у нашій системі відіграє роль центрального вузла, де розгорнуто Home Assistant та реалізовано логіку взаємодії між ESP32 і «розумною» розеткою. Спершу необхідно підготувати саме обладнання: стандартна модель Raspberry Pi 4 (рис. 3.1) з 2 ГБ оперативної пам’яті або вище встановлюється у захисний корпус, після чого вставляється карта пам’яті microSD [11].

Рисунок 3.1 – Зовнішній вигляд Raspberry Pi 4

Після того як microSD записано образом Home Assistant OS (з офіційного сайту hass.io) за допомогою утиліти balenaEtcher, карту вставляють у слот Raspberry Pi, підключають microHDMI кабель до монітора та USB-клавіатуру для початкової перевірки. Raspberry Pi живиться від адаптера 5 В на 3 А через порт USB-C. На екрані буде виведено повідомлення про завантаження Home Assistant και після приблизно 10–15 хвилин інсталяційних скриптів консолі виведе рядок «Home Assistant is running at http://homeassistant.local:8123». Потім у браузері варто ввести цю адресу з портом 8123, щоб потрапити на веб-інтерфейс [11].

3.1.2. Фізичне налаштування розумної розетки

Зовнішній вигляд цього пристрою виконано у поєднанні білого й чорного пластику: нижня частина корпусу — біла, в якій розташовані штифтові вилка й внутрішні електричні контакти, а верхня — чорною вставкою з кнопкою, яка прикрашена фронтальним круглим індикатором живлення (рис. 3.2). Цей індикатор, у вигляді підсвіченого кола, виконує роль фізичної кнопки керування, що дозволяє вручну вмикати або вимикати живлення під’єднаного приладу.

Рисунок 3.2 – Зовнішній вигляд розумної розетки

Приєднана до магістрального живлення розетка готова приймати команди від Home Assistant через локальний протокол Local Tuya, що забезпечує миттєве включення або відключення мереже-вого струму без участі хмарних серверів. У робочому режимі індикатор світиться червоним, підтверджуючи наявність живлення у внутрішньому реле, що дозволяє вимірювати споживану потужність та передавати відповідну інформацію до ESP32 і далі—у систему моніторингу (рис 3.3).

Рисунок 3.3 – Розумна розетка в робочому стані

При першому ввімкненні розумної розетки варто затиснути кнопку керування на 5 секунд та очікувати червоно-синє мерехтіння індикатора на розетці. Ввімкнення стану пошуку, варто відкрити застосунок для керування розумними розетками SmartLife. Після відкривання додатку, там необхідно зареєструватись. Після завершення реєстрації, користувача буде перенаправлено на стартову сторінку додатку.

Тепер треба додати розетку у додаток SmartLife. Для цього натискаю на стартовій сторінка піктограму “+” та вибираю Add Device(Додати пристрій). Далі варто включити технологію Bluetooth на телефоні та очікувати, поки розетка буде знайдена у пошуку (рис 3.4).

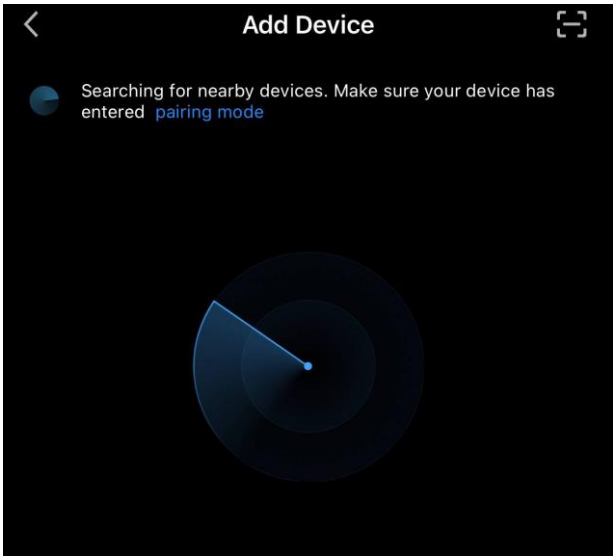

Рисунок 3.4 – Автоматний пошук розумних розеток з підтримкою застосунку SmartLife

Також є можливість додати розетку вручну, якщо вона не була знайдена автоматичним пошуком. Для цього необхідно натиснути Add Manually та вибрати тип необхідної розетки (рис. 3.5).

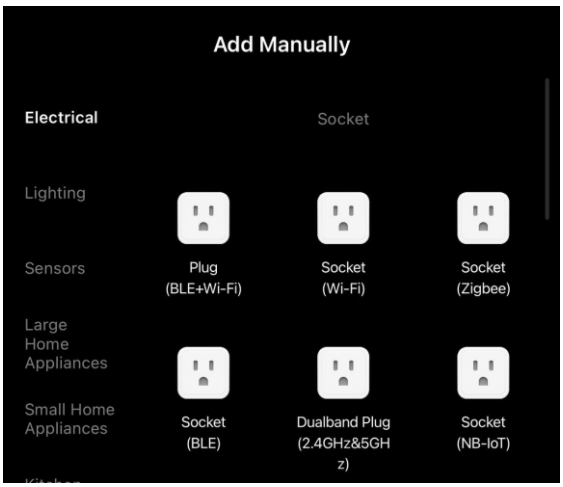

Рисунок 3.5 – Ручне додавання розумних розеток

Далі варто вибрати мережу, в якій буде здійснено пошук на частоті 2.4 ГГц та ввести її пароль (рис. 3.6). Після підтвердження, буде зображено інструкцію, як перезавантажити розетку та ввести її в стан пошуку.

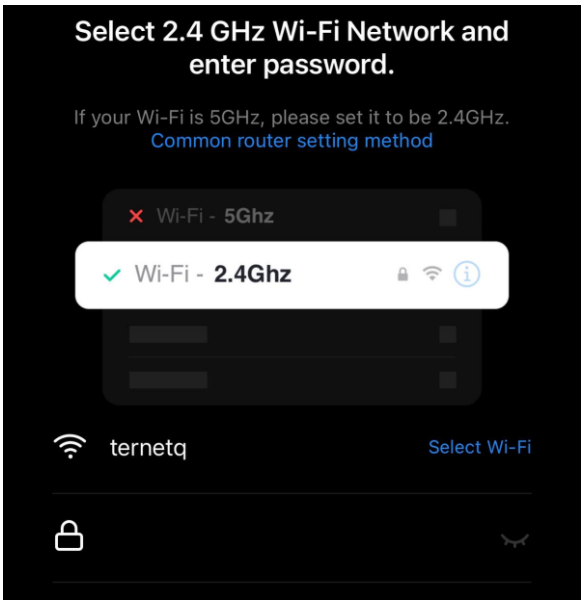

Рисунок 3.6 – Вибір мережі, в якій буде здійнено пошук розумних розеток

Після додавання розумної розетки на домашній сторінці буде відображено всі розумні розетки, до яких є доступ керування через додаток SmartLife (рис. 3.7).

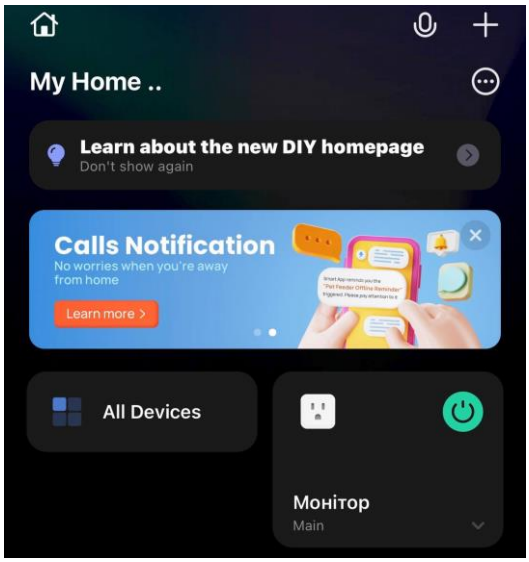

Рисунок 3.7 – Стартова сторінка додатку SmartLife

3.2. Організація доступу з мережі інтернет

Для керування системою моніторингу за межами локальної мережі варто використовувати надійний та шифрований VPN-тунель Cloudflare. Cloudflare Tunnel працює як постійно відкритий зашифрований канал між локальним сервером (наприклад, Raspberry Pi) і глобальною мережею Cloudflare, що дозволяє публікувати внутрішні веб-сервіси у Інтернеті, минаючи необхідність відкривати порти на домашньому роутері. Спочатку на локальному пристрої встановлюється програма-клієнт cloudflared, який після аутентифікації в обліковому записі Cloudflare отримує сертифікат для підтвердження того, що цей пристрій є довіреним. Як тільки сертифікат отримано, cloudflared встановлює зашифроване HTTPS-з'єднання з одним із серверів Cloudflare. Цей канал тримається відкритим у постійному режимі, передаючи періодичні сигнали keep-alive, щоб сервер Cloudflare не переривав з'єднання (рис. 3.8) [12].

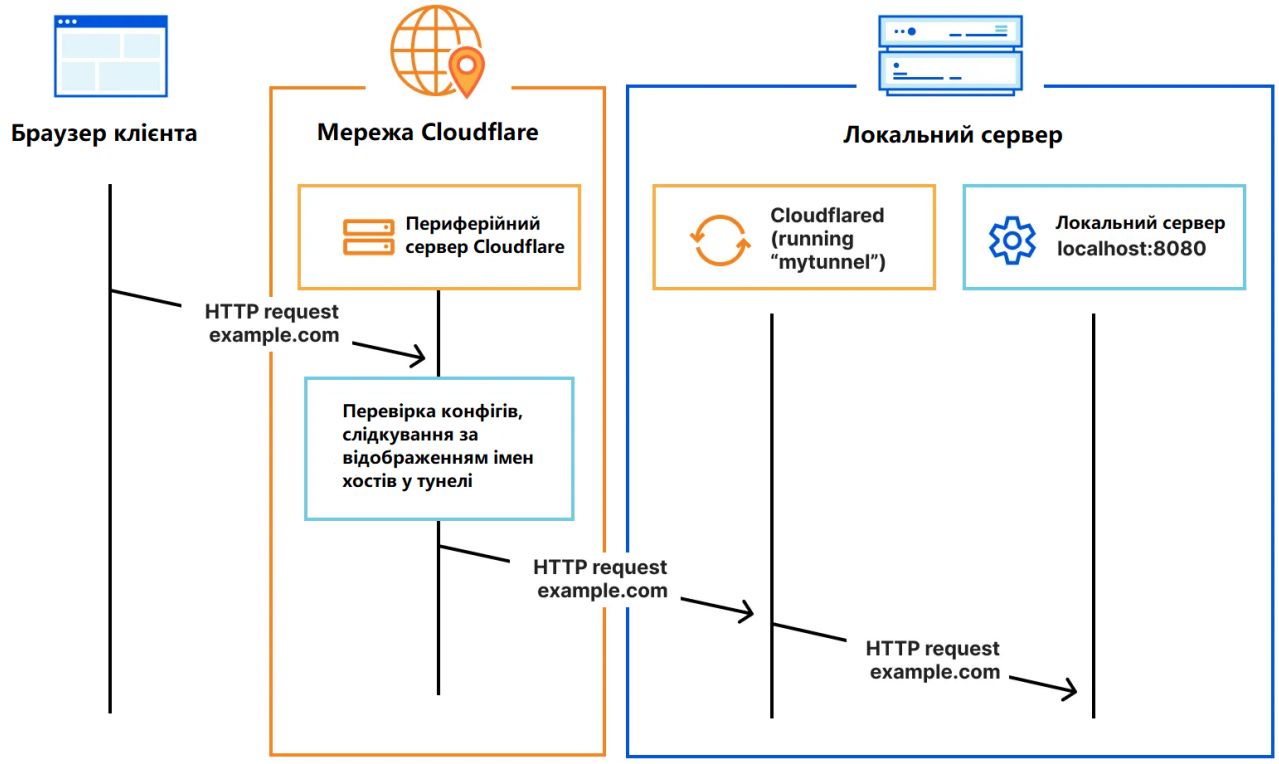

Рисунок 3.8 – Схема роботи VPN-тунелювання Cloudflare

Паралельно в панелі Cloudflare створюється необхідний домен (наприклад, home.example.com), який адресує всі DNS-запити не на вашу домашню мережу, а на внутрішні сервери Cloudflare. Коли хтось вводить адресу домену у браузері, запит потрапляє спочатку на мережу Cloudflare, далі у хмарі відбувається співставлення цього домену з уже встановленим тунелем, і Cloudflare пересилає запит внутрішнім каналом до cloudflared на Raspberry Pi. Оскільки з'єднання між cloudflared і Cloudflare зашифроване, всі дані передаються безпечно.

Після отримання запиту локальний веб-сервер (наприклад, Home Assistant на Raspberry Pi або веб-інтерфейс ESP32) формує відповідь, яку передає назад тому ж шляхом: сервіс Cloudflare приймає відповідь, розшифровує її та надсилає користувачу у браузер. Користувач не бачить жодного відкритого порту на своєму маршрутизаторі — усе спілкування відбувається через вихідний канал до Cloudflare. При цьому Cloudflare автоматично піклується про випуск SSL-сертифіката для вашого домену, тому звернення користувача відбувається через протокол HTTPS без додаткових кроків з вашого боку [12].

Завдяки такому підходу, навіть якщо домашній інтернет-канал змінює IP-адресу, тунель автоматично переналаштовується, а зовнішній доступ до Home Assistant чи веб-інтерфейсу ESP32 залишається стабільним. Крім того, Cloudflare надає вбудовані механізми захисту — Web Application Firewall, фільтрацію підозрілих запитів і можливість двофакторної автентифікації перед доступом, що гарантує надійний захист від несанкціонованого проникнення. Таким чином, Cloudflare Tunnel виконує роль VPN-подібного підключення, забезпечуючи конфіденційність і цілісність даних, а також дозволяє зручно й безпечно керувати розетками та переглядати їхній стан з будь-якої точки світу.

3.3. Тестування системи

У ході тестування була зібрана прототипна макетка системи, побудована навколо мікроконтролера ESP32, OLED-дисплея та трьох кнопок, що забезпечують локальне керування. Макетка змонтована на стандартній макетній

платі: до виводів ESP32 підключено OLED-екран 128×64 пікселів за інтерфейсом SPI (лінії CS, DC, RST, SCL і SDA отримували живлення безпосередньо від 3.3 В і GND мікроконтролера), а із цифрових входів контролера знімалися сигнали з трьох кнопок, під'єднаних із використанням внутрішніх підтягувальних резисторів. Живлення всіх компонентів подавалося від єдиного джерела 3.3 В, що гарантувало відсутність перепадів і додаткових шумів [13].

На рисунку 3.9 представлено схему підключення: OLED-дисплей 128x64 пікселів з'єднаний з ESP32 за протоколом SPI, де лінії SDA та SCL підключено до відповідних виводів контролера. Живлення дисплея забезпечується від 3.3 В та GND.

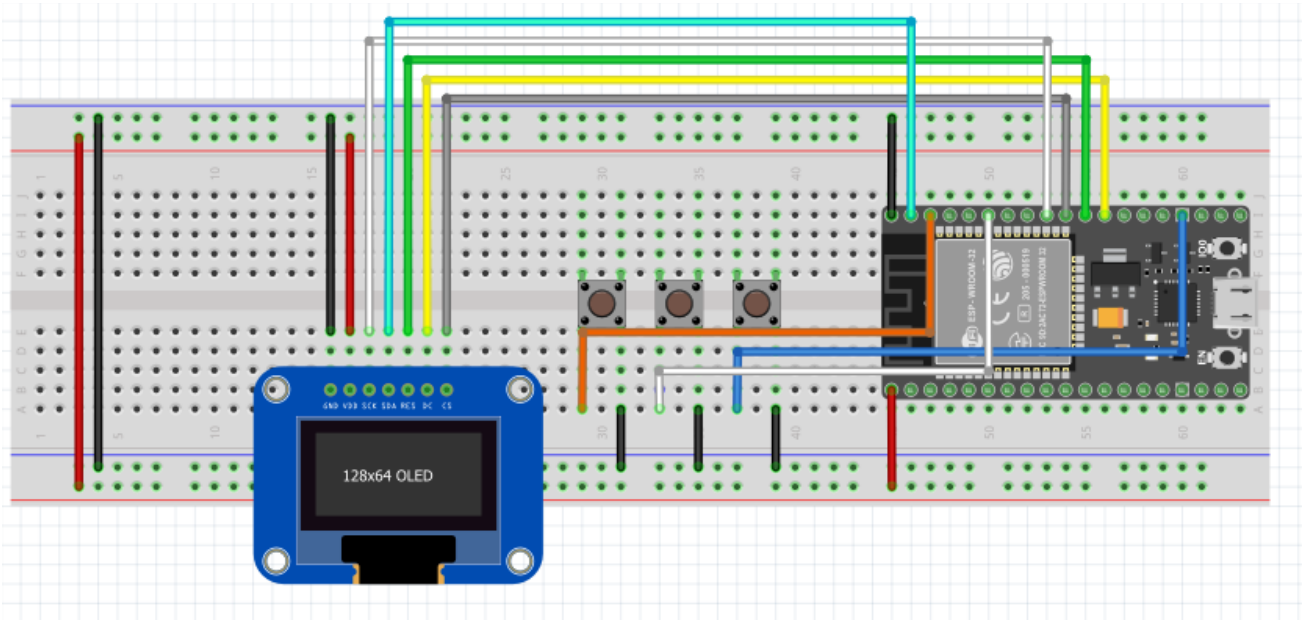

Рисунок 3.9 – Схема підключення ESP32 до всіх апаратних компонентів

Тестування розпочалося з перевірки належної роботи кнопок: кожна кнопка мала забезпечувати перемикання стану віртуальної «розетки» (перша кнопка), зміни режиму відображення (друга кнопка) або виклик виводу додаткової інформації (третя кнопка). На етапі налагодження було виявлено та усунуто дрібні проблеми з дебаунсом – додавання програмної фільтрації дозволило уникнути багатократних спрацювань при одиничному натисканні.

Паралельно було перевірено стабільність роботи OLED-дисплея. При тривалих тестах вдалося підтвердити, що після виклику u8g2.clearBuffer() та запису символів шрифтів відповідних розмірів дисплей оновлював інформацію без мерехтінь і збоїв. Час повного оновлення екрана вимірювався ліченими десятками мілісекунд, що забезпечило відсутність відчутної затримки при взаємодії користувача. Поза тим, система демонструвала коректне відображення рядків «Comp Socket: On/Off» і «Monitor Socket: On/Off» за першого завантаження, а при перемиканні станів кнопками змінювала ці написи миттєво.

Наступним кроком стало тестування взаємодії ESP32 із Home Assistant через REST-інтерфейс. Для симуляції роботи розумної розетки ESP32 відправляв HTTP-GET-запити на шляхи /switch/on та /switch/off, після чого Home Assistant одразу змінював стан відповідної сутності в базі даних. Після отримання підтвердження у вигляді простого тексту (ON або OFF) ESP32 повторно оновлював екран із повідомленням про новий стан. Середній час відправки запиту й отримання відповіді становив близько 100–150 мс у межах локальної мережі. Додаткові вимірювання показали, що при навантаженні з одночасним надсиланням кількох запитів система залишалася стабільною й не втрачала стан жодної з команд [13].

Також було перевірено правильність формування JSON-відповіді за маршрутом /data. ESP32 здійснював періодичний виклик цього шляху, отримував у відповідь JSON-структуру зі значеннями полів power, voltage і current і парсив ці дані через ArduinoJson. Відображені на екрані цифри повністю співпадали із значеннями, які паралельно виводив Home Assistant у веб інтерфейсі. Це підтвердило правильність роботи синхронізації даних між пристроями.

Особливу увагу було приділено тесту на стійкість мережевого з’єднання. Під час примусового відключення Wi-Fi у макетній платі (імітація втрати зв’язку із маршрутизатором) ESP32 повністю втратив доступ до REST-інтерфейсу Home Assistant, але миттєве повторне підключення (через кілька секунд після увімкнення Wi-Fi) відновлювало роботу, а екран відображав попереднє

повідомлення про відсутність зв’язку. У випадку якщо Home Assistant недоступний (наприклад, його зупиняли на Raspberry Pi), ESP32 вчасно виявляв помилку HTTP 404 і виводив на дисплей рядок «HA Offline», що зупиняло подальші запити до REST-API і уникало затримок.

Для оцінки навантаження проведено серію тестів із швидкими і повторними натисканнями кнопок і запитами /data кожні 200 мс. Протягом десятихвилинного тесту система залишалася стабільною: жодна запит-відповідь не була втрачена, дисплей не «зависав», а мікроконтролер не вимикався через перегрів чи надмірне використання пам’яті. Це підтвердило, що використання бібліотеки U8g2 разом із WebServer в Arduino не викликає ресурсних колапсів при інтенсивній роботі.

Завершальним кроком стало перевірення кінцевого сценарію: одночасне керування розеткою і запит статистики. Натисканням першої кнопки розетка (через Home Assistant) має змінювати стан, після чого на екрані з’являється повідомлення «Comp turned on» або «Comp turned off» з подальшим поверненням до головного екрану через секунду. Потім натискання другої кнопки миттєво показувало відповідний рядок «Monitor now on/off». Якщо ж натиснути третю кнопку, починався цикл опитування двох сенсорів (монітор і комп’ютер), отримані значення відображалися у вигляді двох блоків «Monitor Stats» і «Comp Stats» із полями P, V, I.

Таким чином, тестування макетки підтвердило, що система відповідає технічному завданню: кнопки реагують швидко, дисплей оновлюється чітко, REST-інтерфейс передає й приймає дані стабільно, а стійкість до мережевих перебоїв забезпечує безперервну роботу базової логіки незалежно від доступності Home Assistant. Все це свідчить про готовність прототипу до інтеграції з реальними пристроями й подальшого масштабування в межі локальної мережі чи за допомогою Cloudflare Tunnel для віддаленого доступу.

РОЗДІЛ 4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Долікарська допомога при ураженні електричним струмом

При ураженні електричним струмом необхідно негайно від’єднати потерпілого від джерела струму, відключивши обладнання від мережі або, якщо це неможливо зробити безпечно, використавши діелектричні матеріали, як-от сухий одяг, дошку чи ізольовані рукавиці.

Якщо використання таких інструментів неможливе, рятувальник може відтягнути потерпілого за сухий одяг, але лише перебуваючи ізольованим – відокремившись від металевих поверхонь і стоячи на гумовому покритті або сухій підлозі. Це дозволяє забезпечити безпеку під час найризикованішої миті допомоги – фізичного контакту з постраждалим.. У режимі підвищеної напруги процедура усунення контакту з джерелом струму потребує максимальної обережності, тому важливо діяти тільки маючи правильно підібрані засоби ізоляції [14].

Той, хто рятує, повинен ізолювати себе, надягнувши рукавички або обмотавши руки шарфом, і стати на гумовий килимок або іншу суху підкладку, уникаючи дотику до металевих предметів. Коли струм відключено, слід акуратно відтягнути потерпілого за сухий одяг, уникаючи контакту з його відкритими ділянками тіла чи металевими поверхнями. Переносити людину в інше місце варто лише тоді, коли існує небезпека й надалі.

Після звільнення потерпілого від дії струму його укладають спиною на тверду поверхню і перевіряють наявність дихання та пульс на сонній артерії. Одночасно слід звернути увагу на розмір зіниць: розширена нерухома зіниця свідчить про погіршення мозкового кровопостачання.

Якщо потерпілий перебуває при свідомості, йому потрібно забезпечити

повний спокій і зручне положення, постійно спостерігаючи за його диханням і пульсом, але не дозволяючи рухатися або повертатися до роботи. У разі затримки викликом лікаря необхідно самостійно доставити потерпілого до медичного пункту [14].

Якщо ж потерпілий не дихає або дихає слабко й уривчасто, необхідно негайно приступати до серцево-легеневої реанімації: натискання на грудну клітку з глибиною 5–6 см з частотою 100–120 на хвилину, чергуючи 30 натискань із двома вдихами. Якщо бронхіальна вентиляція утруднена або неможлива, слід проводити лише масаж грудної клітини без перерв до прибуття медиків. Використання автоматичного зовнішнього дефібрилятора, якщо він доступний, має проводитися негайно за вказівками голосових інструкцій пристрою [12].

Коли потерпілий непритомний, але дихає, його розташовують горизонтально, розстебнувши комір і пасок, забезпечують приплив свіжого повітря і дають понюхати нашатирний спирт. Якщо ж потерпілий не дихає або дихає уривчасто, слід негайно розпочати серцево-легеневу реанімацію, виконуючи натискання на грудину з частотою 100–120 натисків на хвилину і глибиною 5–6 см, чергуючи 30 натискань із двома вдихами.

Якщо неможливо забезпечити штучну вентиляцію легенів, треба виконувати лише безперервний масаж грудної клітки. Як тільки з’явиться автоматичний зовнішній дефібрилятор, його вмикають, розміщують електроди на оголеній грудній клітці та дотримуються голосових або візуальних інструкцій апарата. Реанімацію слід проводити безперервно до стабілізації стану потерпілого або до приїзду професійних медиків [15].

Навіть якщо після надання першої допомоги людина знову приходить до тями, її не можна відпускати додому або залишати без нагляду. Ураження електричним струмом часто має відстрочені наслідки: можливі порушення серцевого ритму, гостра ниркова недостатність, нервові чи м’язові ушкодження, які можуть проявитися через кілька годин.

Тому обов’язково слід транспортувати потерпілого до лікарні для подальшого обстеження та спостереження. Щоб уникнути таких ситуацій,

необхідно завжди дотримуватися правил електробезпеки: не ремонтувати мережу самотужки, не користуватися пошкодженими приладами, особливо у вологих умовах, і дбайливо ставитися до ізоляції живих частин обладнання. Бережіть себе і інших — своєчасна та правильна допомога при ураженні струмом може врятувати життя.

4.2 Заходи щодо захисту від ураження електричним струмом.

Електробезпека охоплює сукупність організаційних та технічних заходів, спрямованих на захист людей від шкідливої та небезпечної дії електричного струму, електричної дуги, електромагнітного поля та статичної електрики. Першим і найважливішим із них є забезпечення ізоляції струмопровідних частин обладнання, причому передбачена як базова робоча ізоляція, так і випадки застосування подвійної чи посиленої ізоляції для підвищеної надійності. Матеріали та конструктивні рішення, що утворюють захисні бар’єри або огородження, а також розташування електропроводки на висоті й у захищених каналах, гарантують відсутність випадкового контакту з оголеними провідниками навіть за ослаблення первинної ізоляції [16].

Установка та експлуатація електроустаткування регламентуються державними стандартами і правилом «Правила улаштування електроустановок», які визначають норму максимально допустимої напруги на доступних частинах, а також категорії приміщень залежно від рівня небезпеки. В умовах підвищеної небезпеки та особливо небезпечних приміщень використовують малу напругу (до 42 В), яка стає найбільш ефективним засобом зниження ризику ураження. Для ручного інструменту і портативних світильників часто застосовують трансформатори, що знижують напругу до 12 чи 36 В, при цьому вторинні обмотки таких трансформаторів заземлюють або зрівнюють потенціали їх середніх точок, аби унеможливити перенесення високої напруги у разі пробою ізоляції.

Захисне заземлення і занулення є ключовими колективними заходами безпеки. Перший метод полягає в свідомому з’єднанні металевих частин, які не повинні перебувати під напругою, з землею чи її еквівалентом. Це створює шлях найменшого опору для струму витоку й оберігає людину від ураження, оскільки величина потенційної різниці на корпусі, навіть у випадку пробою, залишається безпечною [16].

Захисне вимкнення доповнює систему заземлення й занулення і реалізується через реле або пристрої захисного відключення, які виявляють зміну струму або напруги (замикання на землю, зниження опору ізоляції, торкання людини тощо) та розмикають живильний ланцюг за мілісекунди.

Для безпечного виконання робіт підвищеної небезпеки застосовують індивідуальні засоби захисту: діелектричні рукавички, калоші й килимки, інструменти з ізольованими ручками, ізоляційні штанги та кліщі. Всі ці засоби перед використанням проходять періодичні випробування змінним струмом частотою 50 Гц—від півріччя для рукавичок до трьох років для жиростійких діелектричних черевиків—із наступним маркуванням тих, що витримали тест [17].

Якість і надійність усіх видів захисту підтримується постійним технічним контролем: вимірюванням опору заземлювальних пристроїв, перевіркою цілісності та опору ізоляції силових і освітлювальних мереж за допомогою мегометрів, а також оглядами захисних огороджень і функціональних випробувань пристроїв захисного вимкнення. Недотримання цих правил може призвести до трагічних наслідків, як це сталося під час нещасного випадку з електромонтером, який виконував роботи без нагляду, без засобів індивідуального захисту та без увімкненого заземлення [18].

Комплексний підхід до електробезпеки—від ретельної ізоляції та застосування малої напруги до систем заземлення, занулення, захисного вимкнення та регулярного контролю технічного стану—гарантує мінімізацію ризиків ураження електричним струмом і забезпечує безпечну експлуатацію електроустановок.

ВИСНОВКИ

1) Проаналізовано і обґрунтовано контролери для реалізації хмарної системи для моніторингу споживання електроенергії, яка базується на пристроях IoT, а саме обрано контролер ESP32.

2) Проведено аналіз і обґрунтування платформи для реалізації хмарної системи для моніторингу споживання електроенергії, яка базується на пристроях IoT, зокрема, обрано програмну платформу Home Assistant, яка працює на Raspberry Pi 4.

3) Розроблено структуру апаратних компонент системи та реалізовано її прототип з використанням дисплею 1.5" 128x128 OLED Shield Screen Module SH1107.

4) Реалізовано програмне забезпечення системи на мові C++, з використанням протоколу взаємодії HTTP та бібліотек WiFi.h, HTTPClient.h, ArduinoJson.h, WebServer.h, ArduinoJson.h, U8g2lib.h та SPI.h.

5) Забезпечено захищений доступ до створеної системи з мережі Інтернет, використовуючи VPN Cloudflare

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Жаровський Р.О., Луцик Н.С., Осухівська Г.М., Паламар А.М., Тиш Є.В. Методичні вказівки до виконання кваліфікаційної роботи бакалавра для здобувачів першого (бакалаврського) рівня вищої освіти за спеціальністю 123 «Комп'ютерна інженерія» усіх форм навчання. Тернопіль: ТНТУ, 2024. 39 с.
2. Микитишин А.Г., Митник М.М., Стухляк П.Д., Пасічник В.В. Комп'ютерні мережі. Книга 1. Львів: «Магнолія 2006», 2024. 256 с.
3. Дячук О.А., Жаровський Р.О. Використання SDN для оптимізації передачі даних в комп'ютерних мережах. Матеріали XI науково-технічна конференція Тернопільського національного технічного університету імені Івана Пулюя «Інформаційні моделі системи та технології» (13-14 грудня 2023 року). Тернопіль: ТНТУ. 2023. С. 149-150.
4. Ларіоник Р.В., Луцик Н.С., Паламар А.М. Система для моніторингу якості атмосферного повітря на базі IoT. Матеріали IX науково-технічної конференції "Інформаційні моделі, системи та технології" Тернопільського національного технічного університету імені Івана Пулюя (Тернопіль, 8–9 грудня 2021 року), Тернопіль: ТНТУ. 2021. С. 116.
5. Оконський М.В., Лупенко С.А., Паламар А.М. Інформаційно-вимірювальна система для контролю метеорологічних параметрів на основі Інтернету речей. Матеріали IX науково-технічної конференції "Інформаційні моделі, системи та технології" Тернопільського національного технічного університету імені Івана Пулюя (Тернопіль, 8–9 грудня 2021 року), Тернопіль: ТНТУ, 2021. С. 118.
6. Романов Д.В., Осухівська Г.М., Паламар А.М. Система управління зовнішнім освітленням на основі Інтернету речей. Актуальні задачі сучасних технологій : збірник тез доповідей X міжнародної науково-практичної конференції молодих учених та студентів (Тернопіль, 24-25 листопада 2021 року), Тернопіль: ТНТУ, 2021. С. 120.

7. Monk S. Programming Arduino: Getting Started with Sketches. Нью-Йорк, 2022. 178 с.

8. Home Assistant Green. URL: https://support.nabucasa.com/hc/en-us/categories/24638797677853-Home-Assistant-Green (дата звернення: 18.03.25)

9. Home Assistant 101. Посібник для початківців. URL: https://dou.ua/forums/topic/38947/ (дата звернення: 25.03.25)

10. Tuya Smart Delivers IoT Best Practice Using Amazon Aurora. URL: https://www.tuya.com/news-details/tuya-smart-delivers-iot-best-practice-using-amazon-aurora-Kda5vyu6mnsxd (дата звернення: 29.03.25)

11. Getting started with your Raspberry Pi. URL: https://www.raspberrypi.com/documentation/computers/getting-started.html (дата звернення: 18.04.25)

12. Налаштовуємо віддалений доступ до Home Assistant через Cloudflare. URL: https://dou.ua/forums/topic/40931/ (дата звернення: 21.04.25)

13. Свергун С., Жаровський Р. Тестування програмного продукту, побудованого на мікросервісній архітектурі на основі BDD. Матеріали X науково-технічної конференції Тернопільського національного технічного університету імені Івана Пулюя «Інформаційні моделі системи та технології» (7-8 грудня 2022 року). Тернопіль: ТНТУ. 2022. С. 93.

14. Правила надання першої допомоги при ураженні електричним струмом. URL: https://lviv.dsp.gov.ua/pravyla-nadannia-pershoi-dopomohy-pry-u/10723/ (дата звернення: 23.04.25)

15. Перша допомога при ураженні електричним струмом. URL: https://bozhedarivska-selrada.gov.ua/news/1576497483/ (дата звернення 23.04.25)

16. Методи і засоби захисту від ураження електричним струмом. URL: https://buklib.net/books/35195/ (дата звернення: 23.04.25)

17. Заходи захисту від ураження електричним струмом. URL: https://studies.in.ua/bjd-zaporojec/1247-125-zahodi-zahistu-vd-urazhennya-elektrichnim-strumom.html (дата звернення: 24.04.25)

18. Відокремлений підрозділ «Науково-проектний центр розвитку Об’єднаної енергетичної системи України» державного підприємства «Національна енергетична компанія «Укренерго». Правила улаштування електроустановок. Київ, 2017. 617с.

ДОДАТОК А

Технічне завдання

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії

Кафедра комп'ютерних систем та мереж

"Затверджую"

Завідувач кафедри КС

_____ Осухівська Г.М.

"___" _____2025 р

Хмарна система моніторингу споживання електроенергії на базі використання пристроїв IoT

ТЕХНІЧНЕ ЗАВДАННЯ

на _9_ листках

Вид робіт: Кваліфікаційна робота

На здобуття освітнього ступеня «Бакалавр»

Спеціальність 123 «Комп'ютерна інженерія»

«УЗГОДЖЕНО»	«ВИКОНАВЕЦЬ»
Керівник кваліфікаційної роботи	Студент групи СІс-41
_____ к.т.н., доц. Луцків А.М.	_____ Климчук Д.А.
«____» _____ 2025 р.	«____» _____ 2025 р.

Тернопіль 2025

1 Загальні відомості

1.1 Повна назва та її умовне позначення

Повна назва теми кваліфікаційної роботи: «Хмарна система моніторингу споживання електроенергії на базі використання пристроїв IoT».

Умовне позначення кваліфікаційної роботи: КС КРБ 123.259.00.00

1.2 Виконавець

Студент групи СІс-41, факультету комп'ютерно-інформаційних систем і програмної інженерії, кафедри комп'ютерних систем та мереж, Тернопільського національного технічного університету імені Івана Пулюя, Климчук Д.А.

1.3 Підстава для виконання роботи

Підставою для виконання кваліфікаційної роботи є наказ по університету (№4/7-53 від 27.01.2025 р.)

1.4 Планові терміни початку та завершення роботи

Плановий термін початку виконання кваліфікаційної роботи – 27.01.2025 р.

Плановий термін завершення виконання кваліфікаційної роботи – 23.06.2025 р.

1.5 Порядок оформлення та пред'явлення результатів роботи

Порядок оформлення пояснювальної записки та графічного матеріалу здійснюється у відповідності до чинних норм та правил ISO, ЕСКД, ЕСПД та ДСТУ.

Пред’явлення проміжних результатів роботи з виконання кваліфікаційної роботи здійснюється у відповідності до графіку, затвердженого керівником роботи. Попередній захист кваліфікаційної роботи відбувається при готовності роботи – наявності пояснювальної записки та графічного матеріалу.

Пред’явлення результатів кваліфікаційної роботи відбувається шляхом захисту на відповідному засіданні ЕК, ілюстрацією основних досягнень за допомогою графічного матеріалу.

2 Призначення і цілі створення системи

2.1 Призначення системи

Хмарна система що розробляється призначена для:

– Керування розумними розетками.

– Моніторингу спожитої електроенергії.

2.2 Мета створення системи

Метою кваліфікаційної роботи є розробка та налаштування хмарної системи моніторингу споживання електроенергії на базі використання пристроїв IoT.

2.3 Характеристика об'єкту

Розробляювана хмарна система призначена для використання в домашніх мультимедійних системах.

3 Вимоги до системи

3.1 Вимоги до системи в цілому

3.1.1 Вимоги до структури та функціонування системи

Хмарна система повинна складатись з:

– Програмного додатку.

– Системи моніторингу.

3.1.2 Вимоги до способів та засобів зв’язку між компонентами системи

Основна вимога, яка ставиться до способів та засобів інформаційного обміну – це їх узгодженість.

3.1.3 Вимоги до режимів функціонування системи

Для системи визначено два режими функціонування:

– нормальний режим функціонування;

– аварійний режим функціонування.

Основним режимом функціонування єнормальний режим.

Для забезпечення нормального режиму функціонування системи необхідно виконувати вимоги і дотримуватись умов експлуатації програмного забезпечення і комплексу технічних засобів системи, вказані у відповідних технічних документах (технічна документація, інструкції з експлуатації і т. д.).

Аварійний режим функціонування системи характеризується відмовою одного або декількох компонент програмного і (або) технічного забезпечення. При цьому функції роботи системи продовжують підтримувати роботу системи моніторингу в межах базових налаштувань.

3.1.4 Вимоги по діагностуванню системи

Для діагностування системи використовуються інструменти діагностування основних процесів системи, які вмонтовані в операційну систему і програмне забезпечення, а також засоби для діагностики апаратного забезпечення.

Інструменти повинні забезпечувати зручний інтерфейс для можливості перегляду діагностичних подій, моніторингу процесу виконання програм.

3.1.5 Перспективи розвитку, проектування системи

Дана система може бути розширена завдяки використанню додаткових програмних компонентів, а також передбачено розширення пам’яті для збільшення ресурсів хмарної системи.

3.2 Показники призначення

Система повинна передбачати можливість масштабування. Можливості масштабування повинні забезпечуватися засобами використовуваного базового програмного і технічного забезпечення.

3.2.1 Вимоги до надійності

Система повинна забезпечувати працездатність та відновлення своїх функцій при виникненні наступних ситуацій:

– при збоях в системі електропостачання апаратної частини;

– при помилках в роботі апаратних засобів;

– при помилках, пов'язаних з програмним забезпеченням (ОС і драйвери пристроїв).

Для захисту апаратури від стрибків напруги і комутаційних завад повинні застосовуватися мережні фільтри.

3.3 Вимоги до безпеки

Зовнішні елементи технічних засобів системи, що перебувають під напругою, повинні мати захист від випадкового дотику, а самі технічні засоби мати занулення або захисне заземлення.

Система електроживлення повинна забезпечувати захисне вимикання при перевантаженнях і коротких замиканнях в колах навантаження, а також аварійне ручне вимикання.

Загальні вимоги пожежної безпеки повинні відповідати нормам на побутове електрообладнання. У разі пожежі не мають виділятися отруйні гази і дим.

3.3.1 Вимоги до експлуатації, технічного обслуговування, ремонту і зберігання компонентів системи

Мікроклімат в приміщеннях повинен відповідати нормам виробничого мікроклімату по ДСН 3.3.6.042-99:

– температуру повітря в межах від +10ºС до +35ºС;
– відносну вологість повітря при 25ºС в межах від 30% до 80%;
– атмосферний тиск 760±25 мм рт. ст.

Періодичне технічне обслуговування використовуваних технічних засобів має проводитися відповідно до вимог технічної документації, але не рідше ніж один раз на рік.

Періодичне технічне обслуговування і тестування технічних засобів повинні включати обслуговування і тестування всіх використовуваних засобів, датчики, контроллери, системи передачі даних, пристрої безперебійного живлення.

На підставі результатів тестування технічних засобів повинні проводитися аналіз причин виникнення виявлених дефектів і прийматися заходи по їх ліквідації.

3.4 Вимоги до захисту інформації від несанкціонованого доступу

Система повинна забезпечувати захист від несанкціонованого доступу на рівні не нижче встановленого вимогами, що пред'являються до категорії 1Д

по класифікації документа, що діє, “Автоматизовані системи. Захист від несанкціонованого доступу до інформації. Класифікація автоматизованих систем”.

Компоненти підсистеми захисту від НСД повинні забезпечувати:

– ідентифікацію користувача;

– перевірку повноважень користувача при роботі з системою;

– розмежування доступу користувачів.

Рівень захищеності від несанкціонованого доступу засобів обчислювальної техніки, що здійснюють обробку конфіденційної інформації, повинен відповідати вимогам класу захищеності згідно вимогам документу “Засоби обчислювальної техніки. Захист від несанкціонованого доступу до інформації. Показники захищеності від несанкціонованого доступу до інформації”.

3.4.1 Вимоги по збереженню інформації при аваріях

Інформація, при виникненні аварійних ситуацій повинна бути збережена на резервних носіях.

3.4.2 Вимоги по стандартизації і уніфікації

Система повинна відповідати вимогам ергономіки і зручності користування за умови комплектування високоякісним обладнанням (ЕОМ, монітор і інше обладнання), що має необхідні сертифікати відповідності і безпеки.

3.4.3 Вимоги до функцій (завдань), що виконуються системою:

– забезпечення зручного інтерфейсу;

– забезпечення зберігання медіафайлів;

– пошук необхідної інформації;

– забезпечення високої швидкодії.

4 Вимоги до документації

Документація повинна відповідати вимогам ЄСКД та ДСТУ

Комплект документації повинен складатись з:

– пояснювальної записки;

– графічного матеріалу:

а) Структурна схема.

б) Блок схема роботи комп’ютеризованої системи.

в) Схема електрична принципова.

г) Схема електрична монтажна (з’єднань).

*Примітка: У комплект документації можуть вноситися зміни та доповнення в процесі розробки.

5 Стадії та етапи проектування

Таблиця 1 – Стадії та етапи виконання кваліфікаційної роботи бакалавра

№ етапу	Назва етапу виконання кваліфікаційної роботи	Термін виконання
1	Розробка технічного завдання	27.01.2025р. – 02.02.2025р.
2	Аналіз технічного завдання та постановка задачі	03.02.2025р. – 15.02.2025р.
3	Розробка архітектури системи	16.02.2025р. – 10.03.2025р.
4	Реалізація збору даних з розеток (ESP32 → Home Assistant)	11.03.2025р. – 23.03.2025р.
5	Розробка API для обміну даними між сайтом і Home Assistant	24.03.2025р. – 5.04.2025р.
6	Створення вебінтерфейсу: авторизація, графіки, дашборд	6.04.2025р. – 17.04.2025р.
7	Тестування взаємодії між ESP32, сайтом і розетками	18.04.2025р. – 22.04.2025р.
8	Безпека життєдіяльності, основи охорони праці	23.04.2025р. – 30.04.2025р.
9	Оформлення пояснювальної записки і графічного матеріалу	1.05.2025р. – 8.06.2025р.

10	Перевірка на академічний плагіат, перевірка керівником та консультантами	9.06.2025р. – 15.06.2025р
11	Попередній захист кваліфікаційної роботи бакалавра	16.06.2025р. – 22.06.2025р.
12	Захист кваліфікаційної роботи бакалавра	27.06.2025р.

6 Додаткові умови виконання кваліфікаційної роботи

Під час виконання кваліфікаційної роботи у дане технічне завдання можуть вноситися зміни та доповнення

ДОДАТОК Б

Лістинг коду програми

```
#include <WiFi.h>
#include <HTTPClient.h>
#include <ArduinoJson.h>
#include <WebServer.h>
#include <ArduinoJson.h>
#include <U8g2lib.h>
#include <SPI.h>
#define OLED_CS 5
#define OLED_DC 16
#define OLED_RST 17
U8G2_SH1107_128X128_F_4W_HW_SPI u8g2(U8G2_R2, OLED_CS, OLED_DC, OLED_RST);
WebServer server(80);
const int buttonPin1 = 15;
const int buttonPin2 = 21;
const int buttonPin3 = 22;
bool buttonState1 = false;
bool buttonState2 = false;
int lastButtonReading1 = HIGH;
int lastButtonReading2 = HIGH;
int lastButtonReading3 = HIGH;
const char* ssid = "wifi";
const char* password = "password";
const char* haServer = "http://192.168.0.107:8123";
// IP-адреса Home Assistant
const char* haToken = "token";  // HA токен
String entityName = "monitor";
void handleRoot();
void handleData();
void switchPlug(bool turnOn);
float getSensorValue(String endpoint);
void handleSetEntity();
void setup() {
  pinMode(buttonPin1, INPUT_PULLUP);
  pinMode(buttonPin2, INPUT_PULLUP);
  pinMode(buttonPin3, INPUT_PULLUP);
  Serial.begin(115200);
  u8g2.begin();
  u8g2.clearBuffer();
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.print(".");
  }
  Serial.println("\nWiFi Connected!");
  Serial.print("ESP32 IP: ");
  Serial.println(WiFi.localIP());
  server.on("/", handleRoot);
  server.on("/data", handleData);
  server.on("/switch/on", []() {
    switchPlug(true);
    server.send(200, "text/plain", "ON");
  });
  server.on("/switch/off", []() {
    switchPlug(false);
    server.send(200, "text/plain", "OFF");
  });
```

```
  server.on("/setEntity", []() {
    if (server.hasArg("name")) {
      entityName = server.arg("name");
      String entityId = "switch." + entityName + "_socket";
      String entityPower = "sensor." + entityName + "_socket_power";
      String entityVoltage = "sensor." + entityName + "_socket_voltage";
      String entityCurrent = "sensor." + entityName + "_socket_current";
      StaticJsonDocument<200> jsonDoc;
      jsonDoc["success"] = true;
      String response;
      serializeJson(jsonDoc, response);
      server.send(200, "application/json", response);
    } else {
      server.send(400, "application/json", "{\"error\":\"Missing name
parameter\"}");
    }
  });
  server.begin();
  Serial.println("HTTP server started");
}
void loop() {
  server.handleClient();
  int reading1 = digitalRead(buttonPin1);
  int reading2 = digitalRead(buttonPin2);
  int reading3 = digitalRead(buttonPin3);
  u8g2.clearBuffer();
  u8g2.setFont(u8g2_font_unifont_t_symbols);
  u8g2.drawGlyph(25, 25, 0x2603);
  u8g2.setFont(u8g2_font_ncenB08_tr);
  u8g2.setCursor(0, 50);
  u8g2.print("Home page");
  // Comp Socket
  u8g2.setFont(u8g2_font_5x8_tr);
  u8g2.setCursor(0, 70);
  u8g2.print("Comp Socket: ");
  u8g2.setCursor(80, 70);
  u8g2.print(buttonState1 ? "On" : "Off");
  // Monitor Socket
  u8g2.setCursor(0, 80);
  u8g2.print("Monitor Socket:  ");
  u8g2.setCursor(80, 80);
  u8g2.print(buttonState2 ? "On" : "Off");
  u8g2.sendBuffer();
  // КНОПКА 1
  if (reading1 == LOW && lastButtonReading1 == HIGH) {
    buttonState1 = !buttonState1;
    entityName = "comp";
    switchPlug(buttonState1);
    u8g2.clearBuffer();
    u8g2.setFont(u8g2_font_ncenB08_tr);
    u8g2.setCursor(0, 64);  // Зміщення вниз, бо 0,0 може бути десь у повітрі
    u8g2.print("Comp turned ");
    u8g2.setCursor(75, 64);
    u8g2.print(buttonState1 ? "on" : "off");
    u8g2.sendBuffer();
    delay(1000);
  }
  lastButtonReading1 = reading1;
  // КНОПКА 2
  if (reading2 == LOW && lastButtonReading2 == HIGH) {
    buttonState2 = !buttonState2;
    entityName = "monitor";
    // switchPlug(buttonState2);
```

```
    u8g2.clearBuffer();

    u8g2.setFont(u8g2_font_ncenB08_tr);
    u8g2.setCursor(0, 64);  // Зміщення вниз, бо 0,0 може бути десь у повітрі
    u8g2.print("Monitor now ");
    u8g2.setCursor(78, 64);
    u8g2.print(buttonState2 ? "on" : "off");
    u8g2.sendBuffer();
    delay(1000);
  }
  lastButtonReading2 = reading2;
  // КНОПКА 3
  if (reading3 == LOW && lastButtonReading3 == HIGH) {
    entityName = "monitor";
    // Отримання та відображення виміряних значень для поточного entity (comp)
    float power = getSensorValue("/api/states/sensor." + entityName +
"_socket_power");
    float voltage = getSensorValue("/api/states/sensor." + entityName +
"_socket_voltage");
    float current = getSensorValue("/api/states/sensor." + entityName +
"_socket_current");
    entityName = "comp";
    // Отримання та відображення виміряних значень для поточного entity (comp)
    float power_c = getSensorValue("/api/states/sensor." + entityName +
"_socket_power");
    float voltage_c = getSensorValue("/api/states/sensor." + entityName +
"_socket_voltage");
    float current_c = getSensorValue("/api/states/sensor." + entityName +
"_socket_current");
    u8g2.clearBuffer();
    u8g2.setFont(u8g2_font_ncenB08_tr);
    u8g2.setCursor(0, 10);
    u8g2.print("Monitor Stats:");
    u8g2.setFont(u8g2_font_5x8_tr);
    u8g2.setCursor(0, 25);
    u8g2.print("P: ");
    u8g2.print(power);
    u8g2.print(" W");
    u8g2.setCursor(0, 40);
    u8g2.print("V: ");
    u8g2.print(voltage);
    u8g2.print(" V");
    u8g2.setCursor(0, 55);
    u8g2.print("I: ");
    u8g2.print(current);
    u8g2.print(" A");
    u8g2.setFont(u8g2_font_ncenB08_tr);
    u8g2.setCursor(0, 75);
    u8g2.print("Comp Stats:");
    u8g2.setFont(u8g2_font_5x8_tr);
    u8g2.setCursor(0, 90);
    u8g2.print("P: ");
    u8g2.print(power_c);
    u8g2.print(" W");
    u8g2.setCursor(0, 105);
    u8g2.print("V: ");
    u8g2.print(voltage_c);
    u8g2.print(" V");
    u8g2.setCursor(0, 120);
    u8g2.print("I: ");
    u8g2.print(current_c);
    u8g2.print(" A");
    u8g2.sendBuffer();
```

```
    delay(3000);
  }
  lastButtonReading3 = reading3;
}
void handleRoot() {
  String html = R"rawliteral(
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>ESP32 Monitoring</title>
   <style>
    body {
      font-family: Arial, sans-serif;
      margin: 0;
      padding: 0;
      background: #f0f2f5;
      color: #333;
    }
    header {
      background: #4267b2; /* Приклад кольору шапки */
      color: #fff;
      padding: 1rem;
      text-align: center;
      margin-bottom: 1rem;
    }
    .container {
      max-width: 1200px;
      margin: 0 auto;
      padding: 0 1rem;
    }
    h1 {
      margin: 0;
    }
    .controls {
      display: flex;
      flex-wrap: wrap;
      gap: 1rem;
      align-items: center;
      margin-bottom: 1rem;
    }
    .controls select,
    .controls button {
      padding: 0.5rem 1rem;
      font-size: 1rem;
      border: 1px solid #ccc;
      border-radius: 4px;
      cursor: pointer;
      background: #fff;
    }
    .controls button:hover,
    .controls select:hover {
      background: #e9ecef;
    }
    .info {
      display: flex;
      flex-wrap: wrap;
      gap: 2rem;
      margin-bottom: 1rem;
    }
    .info p {
      background: #fff;
      padding: 1rem;
```

```
    border-radius: 5px;
    box-shadow: 0 0 5px rgba(0,0,0,0.1);
    min-width: 120px;
    text-align: center;
    margin: 0;
  }
  #chart-container {
    background: #fff;
    padding: 1rem;
    border-radius: 5px;
    box-shadow: 0 0 5px rgba(0,0,0,0.1);
    margin-bottom: 2rem;
  }
</style>
<script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
<script>
  let chart;
  let dataLabels = [];
  let powerData = [];
  let voltageData = [];
  let currentData = [];
  function createChart() {
    const ctx = document.getElementById('myChart').getContext('2d');
    chart = new Chart(ctx, {
      type: 'line',
      data: {
        labels: dataLabels,
        datasets: [
          {
            label: 'Voltage (V)',
            data: voltageData,
            borderColor: 'blue',
            fill: false,
            spanGaps: true
          },
          {
            label: 'Power (W)',
            data: powerData,
            borderColor: 'red',
            fill: false,
            spanGaps: true
          },
          {
            label: 'Current (A)',
            data: currentData,
            borderColor: 'green',
            fill: false,
            spanGaps: true
          }
        ]
      },
      options: {
        responsive: true,
        scales: {
          x: {
            title: { display: true, text: 'Time' }
          },
          y: {
            type: 'logarithmic',
            title: { display: true, text: 'Value' }
          }
        }
      }
```

```
      });
    }
    function updateData() {
      fetch('/data')
      .then(response => response.json())
      .then(data => {
        document.getElementById("power").innerText = data.power;
        document.getElementById("voltage").innerText = data.voltage;
        document.getElementById("current").innerText = data.current;
        // Додаємо нову мітку часу і дані для графіка
        let now = new Date().toLocaleTimeString();
        dataLabels.push(now);
        powerData.push(data.power);
        voltageData.push(data.voltage);
        currentData.push(data.current);
        // Зберігаємо максимум 20 точок для збереження продуктивності
        if (dataLabels.length > 20) {
          dataLabels.shift();
          powerData.shift();
          voltageData.shift();
          currentData.shift();
        }
        chart.update();
      })
      .catch(error => console.error('Error:', error));
    }
    window.onload = function() {
      createChart();
      setInterval(updateData, 2000);
    };
  </script>
  <script>
  function setEntity(name) {
    fetch('/setEntity?name=' + name)
    .then(response => response.json())
    .then(data => {
      if (data.success) {
        document.getElementById("selectedEntity").innerText = name;
      }
    })
    .catch(error => console.error('Error:', error));
  }
</script>
</head>
<body>
<header>
    <h1>ESP32 Monitoring</h1>
  </header>
  <div class="container">
    <div class="controls">
      <label for="entitySelector"><strong>Selected entity:</strong></label>
      <span id="selectedEntity">monitor</span>
      <select id="entitySelector" onchange="setEntity(this.value)">
        <option value="monitor">Monitor Socket</option>
        <option value="comp">Computer Socket</option>
      </select>
      <button onclick="fetch('/switch/on')">Turn ON</button>
      <button onclick="fetch('/switch/off')">Turn OFF</button>
    </div>
    <div class="info">
      <p><strong>Voltage:</strong><br><span id="voltage">...</span> V</p>
      <p><strong>Power:</strong><br><span id="power">...</span> W</p>
      <p><strong>Current:</strong><br><span id="current">...</span> A</p>
```

```
    </div>
    <div id="chart-container">
      <canvas id="myChart"></canvas>
    </div>
  </div>
</body>
</html>
)rawliteral";
  server.send(2000, "text/html", html);
}
void handleSetEntity() {
  if (server.hasArg("name")) {
    entityName = server.arg("name");
  }
  server.send(200, "text/plain", "Entity updated");
}
void handleData() {
  String entityPower = "sensor." + entityName + "_socket_power";
  String entityVoltage = "sensor." + entityName + "_socket_voltage";
  String entityCurrent = "sensor." + entityName + "_socket_current";
  float power = getSensorValue("/api/states/" + entityPower);
  float voltage = getSensorValue("/api/states/" + entityVoltage);
  float current = getSensorValue("/api/states/" + entityCurrent);
  StaticJsonDocument<200> jsonDoc;
  jsonDoc["power"] = power;
  jsonDoc["voltage"] = voltage;
  c = current;
  String response;
  serializeJson(jsonDoc, response);
  server.send(200, "application/json", response);
}
void switchPlug(bool turnOn) {
  HTTPClient http;
  String entityId = "switch." + entityName + "_socket";
  String url = String(haServer) + "/api/services/switch/" + (turnOn ? "turn_on"
: "turn_off");
  http.begin(url);
  http.addHeader("Authorization", "Bearer " + String(haToken));
  http.addHeader("Content-Type", "application/json");
  String jsonPayload = "{\"entity_id\": \"" + entityId + "\"}";
  http.POST(jsonPayload);
  http.end();
}
float getSensorValue(String endpoint) {
  HTTPClient http;
  String url = String(haServer) + endpoint;
  http.begin(url);
  http.addHeader("Authorization", String("Bearer ") + haToken);
  http.addHeader("Content-Type", "application/json");
  int httpCode = http.GET();
  if (httpCode == HTTP_CODE_OK) {
    String payload = http.getString();
    DynamicJsonDocument doc(1024);
    deserializeJson(doc, payload);
    http.end();
    return doc["state"].as<float>();
  }
  http.end();
  return 0;
}
```