НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри
Комп'ютерних систем, мереж та кібербезпеки
_____ Касаткін Д.Ю., к.пед.н., доц.
(підпис) (ПІБ, вчене звання і ступінь)

«___»_____2025 р.

КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА

На тему: «Проектування та розробка UI-системи IoT для керування компонентами розумного дому»

Спеціальність F7 «Комп’ютерна інженерія»

Гарант освітньої програми
к.фіз.-мат.н., доц. _____ / Нікітенко Є.В. /
(підпис) (ПІБ)

Керівник дипломного проекту: _____ / Назаренко В.А. /
(підпис) (ПІБ)

Виконав: _____ / Ануа П.В./
(підпис) (ПІБ)

КИЇВ-2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

«ЗАТВЕРДЖУЮ»
завідувач кафедри
комп'ютерних систем, мереж та кібербезпеки
_____/ Касаткін Д.Ю., к.пед.н., доц. /
(підпис) (ПІБ, вчене звання і ступінь)
«__» _____ 20__ р.

З А В Д А Н Н Я
ДО ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ БАКАЛАВРСЬКОЇ СТУДЕНТУ

_____Ануа Поліни Вікторівни_____
(прізвище, ім'я, по батькові)

Спеціальність (напрям підготовки): комп'ютерна інженерія

Тема кваліфікаційної бакалаврської роботи: «Проектування та розробка UI-системи IoT для керування компонентами розумного дому»
затверджена наказом ректора НУБіП України від "16" 12 2024р. №2250 «С»
Термін подання завершеної роботи на кафедру _____

Вихідні дані до кваліфікаційної бакалаврської роботи проектування та розробка UI-системи IoT для керування компонентами розумного дому
Перелік питань, що підлягають розробці:

1. Аналіз існуючих систем контролю мікроклімату в приміщенні
2. Вибір мікроконтролерів та датчиків для розробки системи контролю температури
3. Розробка програмного забезпечення

Перелік графічного матеріалу (за потреби)

Дата видачі завдання "_____"_____2024 р.

Керівник кваліфікаційної роботи _____ Назаренко В.А., доктор філософії.
(підпис) (прізвище та ініціали)

Завдання прийняв до виконання _____ Ануа П.В./
(підпис) (прізвище та ініціали студента)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Аналіз предметної області		Виконано
2	Проектування системи		Виконано
3	Реалізація системи		Виконано
4	Тестування системи		Виконано
5	Оформлення пояснювальної записки		Виконано
6	Оформлення графічного матеріалу		Виконано

Студент _____ **П.В. Ануа**

(підпис) (ініціали та прізвище)

Керівник роботи ____ **В.А.Назаренко**

РЕФЕРАТ

Пояснювальна записка: 66 сторінок, 40 рисунків, 3 таблиці, 11 лістингів, 23 найменування у списку використаних джерел, 2 додатки.

ІНТЕРНЕТ РЕЧЕЙ, КОРИСТУВАЦЬКИЙ ІНТЕРФЕЙС, КЕРУВАННЯ ПРИСТРОЯМИ, MQTT, РОЗУМНИЙ ДІМ, IOT-СИСТЕМА, ВІДДАЛЕНИЙ ДОСТУП.

Об’єктом дослідження є процес дистанційного керування пристроями розумного дому. Предметом дослідження є IoT-система керування та її інтерфейсна частина, яка реалізує взаємодію між користувачем та апаратними компонентами через мережевий протокол.

Робота складається із чотирьох розділів. У першому розділі здійснено детальний аналіз предметної області, зокрема огляд сучасних IoT-платформ і принципів їх роботи, досліджено архітектуру розумного дому та сформульовано технічне завдання із обґрунтуванням вибору апаратної бази. Другий розділ присвячений проєктуванню IoT-системи: побудовано загальну архітектуру, виконано функціональне моделювання, проєктування користувацького інтерфейсу та логічної моделі взаємодії компонентів. У третьому розділі описано реалізацію програмної частини, вибрано інструменти розробки, наведено приклади клієнтського коду, а також представлено створення сенсорних та виконавчих вузлів для IoT-системи. Четвертий розділ містить методику та результати тестування розробленої системи, а також рекомендації щодо її впровадження в реальні умови.

Кваліфікаційна робота на тему «Проєктування та розробка UI-системи IoT для керування компонентами розумного дому» присвячена створенню користувацького інтерфейсу для дистанційного керування елементами системи розумного дому. Метою роботи є проєктування та реалізація користувацького інтерфейсу для дистанційного керування елементами системи розумного дому з використанням архітектури IoT.

У роботі наведено опис апаратної та програмної складових системи, архітектуру взаємодії між UI та пристроями через MQTT-брокер, а також реалізацію прототипу інтерфейсу.

ABSTRACT

The paper contains 66 pages, 40 figures, 3 tables, 11 listings, 23 references, and 2 appendices.

INTERNET OF THINGS, USER INTERFACE, DEVICE MANAGEMENT, MQTT, SMART HOME, IOT SYSTEM, REMOTE ACCESS.

The object of research is the process of remote control of smart home devices. The subject of the study is the IoT control system and its interface part, which implements the interaction between the user and hardware components via a network protocol.

The work consists of four sections. In the first section, a detailed analysis of the subject area is carried out, in particular, an overview of modern IoT platforms and the principles of their operation, the architecture of a smart home is investigated and a technical task is formulated with a justification for the choice of hardware base. The second section is devoted to the design of the IoT system: the general architecture is built, functional modeling, the design of the user interface and the logical model of component interaction are performed. The third section describes the implementation of the software part, selects development tools, provides examples of client code, and presents the creation of touch and execution nodes for an IoT system. The fourth section contains the methodology and results of testing the developed system, as well as recommendations for its implementation in real conditions.

The qualification work on the topic “Design and development of an IoT UI system for controlling smart home components” is devoted to the creation of a user interface for remote control of smart home system elements. The aim of the work is to design and implement a user interface for remote control of smart home system elements using IoT architecture.

The paper describes the hardware and software components of the system, the architecture of interaction between the UI and devices via an MQTT broker, and the implementation of a prototype interface.

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

IoT	Internet of Things
MQTT	Message Queuing Telemetry Transport
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
UI	User Interface
OEM	Original Equipment Manufacturer
HTTP	HyperText Transfer Protocol
BLE	Bluetooth Low Energy
ITU	International Telecommunication Union
USB	Universal Serial Bus
I2C	Inter-Integrated Circuit
PWM	Pulse Width Modulation

ВСТУП

Інтернет речей (IoT) є підходом, який дозволяє об’єднати в єдину мережу фізичні пристрої для збирання, обміну та обробки даних без безпосередньої участі людини. У побутовій сфері концепція IoT реалізується у вигляді систем розумного дому, які забезпечують можливість автоматизації освітлення, опалення, охоронних систем, побутової техніки та інших елементів інфраструктури житлового приміщення. Зростання кількості доступних апаратних засобів, зокрема мікроконтролерів з підтримкою бездротового зв’язку, а також наявність відкритих протоколів передавання даних створюють передумови для масштабного впровадження подібних рішень у житлових і комерційних об'єктах.

У межах даної роботи розглядається проєктування та реалізація системи користувацького інтерфейсу для керування компонентами розумного дому з використанням IoT-технологій. Проєкт охоплює архітектурне планування системи, реалізацію інтерфейсу доступу до функцій керування, організацію обміну даними між клієнтською частиною, серверним програмним забезпеченням та апаратними модулями. Для обміну повідомленнями між компонентами використовується MQTT-протокол, який орієнтовано на передачу коротких повідомлень за схемою «видавець-підписник» і застосовується в умовах обмеженої пропускної здатності або ресурсів.

Метою роботи є проєктування та реалізація користувацького інтерфейсу для дистанційного керування елементами системи розумного дому.

Для досягнення поставленої мети у роботі вирішуються такі завдання:

— проведення аналізу наявних IoT-платформ та технічних засобів;
— побудова функціональної моделі взаємодії системних компонентів;
— реалізація UI-інтерфейсу з можливістю керування пристроями;
— проведення тестування програмного забезпечення;
— забезпечення сумісності з мобільними та десктопними пристроями.

Об'єктом дослідження є процес дистанційного керування пристроями розумного дому.

Предметом дослідження є IoT-система керування та її інтерфейсна частина.

Під час реалізації проєкту використовуються наступні інструменти: мікроконтролери ESP8266/ESP32, протокол обміну повідомленнями MQTT, серверна платформа Express.js, а також інтерфейс, реалізований із використанням HTML, CSS і JavaScript.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ТЕХНІЧНОГО ЗАВДАННЯ

1.1 Аналіз сучасних IoT-платформ

1.1.1 Огляд поширених IoT-рішень

Серед поширених рішень для побудови IoT-систем у побутовій сфері можна виокремити такі платформи, як Tuya, Home Assistant, SmartThings і Tasmota. Кожна з них орієнтована на виконання типових завдань розумного дому, включаючи керування освітленням, кліматичними системами, охоронною інфраструктурою та побутовими електроприладами. Залежно від архітектури та цільової аудиторії реалізовано різні підходи до організації інтерфейсу користувача, автоматизації дій та адміністрування пристроїв. Деякі платформи базуються на хмарних сервісах із централізованим зберіганням даних і керуванням, інші – на локальних рішеннях з високим рівнем кастомізації. Такі особливості визначають напрямок застосування кожної системи, від масового користувача до спеціалізованого розробника або ентузіаста [4, с. 7-18].

Функціональне призначення платформи Tuya полягає у наданні кінцевому користувачу інструментів для централізованого керування великою кількістю сумісних пристроїв. Через мобільний застосунок (рисунок 1.1) здійснюється контроль освітлювальних елементів, розеток, систем безпеки, побутової техніки та інших компонентів. Платформа підтримує створення сценаріїв автоматизації, організацію пристроїв у віртуальні кімнати, планування подій за розкладом, а також інтеграцію з голосовими помічниками. Вся взаємодія з обладнанням здійснюється через хмарну інфраструктуру, що дозволяє забезпечити доступ до керування з будь-якого місця з підключенням до мережі. Відмінною особливістю є велика екосистема OEM-пристроїв, які можна додавати без складної конфігурації.

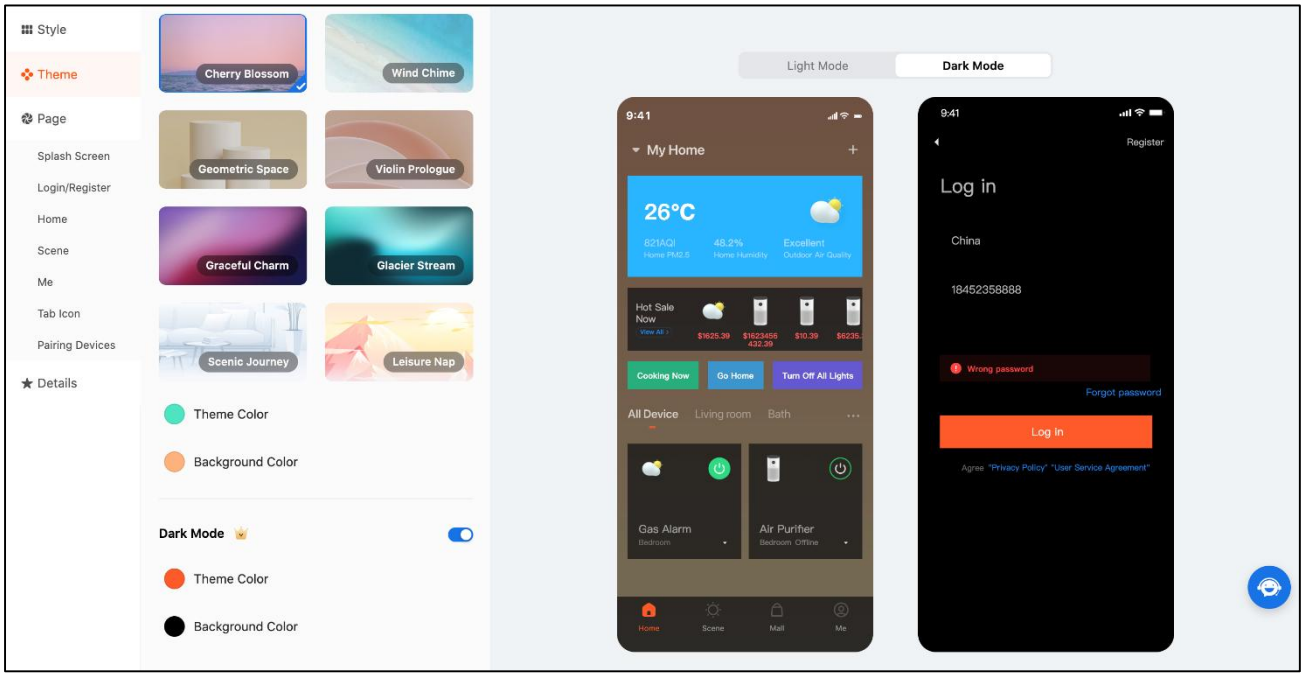


Рисунок 1.1 – Вигляд інтерфейсу користувача Tuya [13]

Система Home Assistant орієнтована на користувачів, які мають потребу в гнучкому налаштуванні взаємодії між пристроями, не обмежуючись лише фірмовими компонентами. Платформа дозволяє реалізовувати складні сценарії автоматизації з умовами, затримками, подіями та діями. Основна увага приділяється можливості інтеграції великої кількості пристроїв і сервісів у межах локальної мережі без обов'язкового використання хмари.

Через веб-інтерфейс (рисунок 1.2) забезпечено візуалізацію даних із сенсорів, запуск дій за подіями, формування панелей керування з динамічними елементами. Можливість створення користувацьких інтерфейсів, переглядів і правил надає широкі можливості для індивідуального налаштування під конкретні потреби.

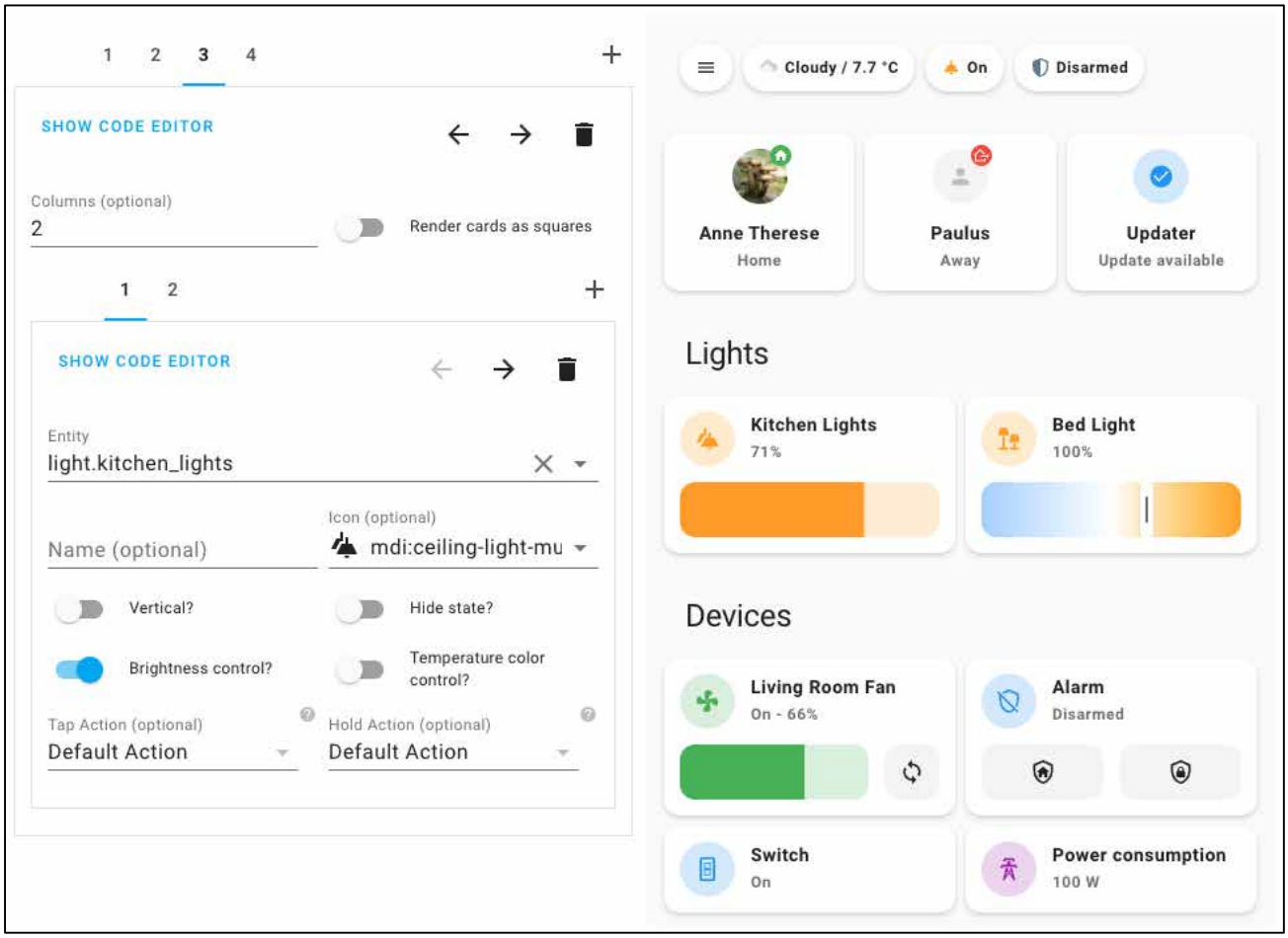


Рисунок 1.2 – Головна сторінка веб-інтерфейсу Home Assistant [11]

Платформа SmartThings забезпечує централізоване керування пристроями в межах фірмової екосистеми, з акцентом на взаємодію зі сторонніми сервісами через хмарну інфраструктуру. Через мобільний застосунок (рисунок 1.3) або вебінтерфейс користувач може здійснювати моніторинг стану, створювати сценарії, організовувати автоматизовані дії за тригерами на основі подій, таких як зміна геолокації, час доби чи зміна стану пристрою.

Особливістю системи є тісна інтеграція з продукцією Samsung та підтримка SmartThings-compatible пристроїв, що дозволяє створювати наскрізні рішення з автоматичним виявленням та налаштуванням компонентів. Крім того, платформа має розвинені механізми для побудови сценаріїв із врахуванням контексту користувача, таких як геолокація або час, що сприяє адаптивному керуванню. Управління здійснюється переважно через хмару, що дозволяє синхронізувати

стан пристроїв між різними пристроями користувача та підтримувати віддалений доступ.

Рисунок 1.3 – Інтерфейс мобільного застосунку SmartThings [5]

Tasmota представляє інший підхід до організації керування IoT-пристроями, орієнтований на автономну експлуатацію без зовнішніх серверів. Прошивка встановлюється безпосередньо на пристрої на базі ESP8266 або ESP32, після чого через вбудований вебінтерфейс (рисунок 1.4) користувач отримує доступ до локального керування, конфігурування та створення автоматизованих дій. Платформа підтримує MQTT, HTTP-запити, правила логіки на рівні пристрою. У порівнянні з іншими платформами, користувач має повний контроль над усіма параметрами, що дає змогу адаптувати прошивку під конкретні задачі, мінімізуючи зовнішні залежності. Tasmota часто використовується у невеликих проєктах або для локального прототипування без залучення хмарних сервісів.

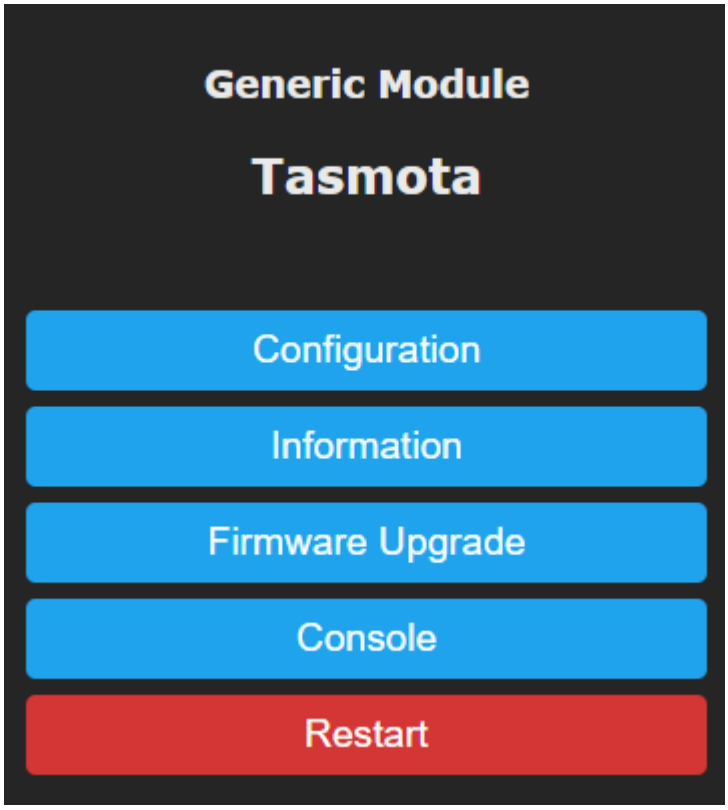


Рисунок 1.4 – Конфігураційна сторінка веб-інтерфейсу Tasmota [22]

1.1.2 Принципи роботи IoT-систем

Для побудови систем IoT використовуються три базові типи компонентів: сенсори, виконавчі пристрої та контролери. Взаємодія цих елементів утворює основу функціонування розумного будинку або будь-якої автоматизованої системи. Кожен із компонентів виконує окрему функцію в загальній структурі, забезпечуючи процес збору, аналізу й реалізації дій на основі даних.

Сенсори забезпечують зчитування параметрів навколишнього середовища або стану об’єктів. Вони фіксують фізичні, хімічні або електричні характеристики й формують цифрові або аналогові сигнали, які надсилаються далі по системі. Наприклад, сенсори температури, вологості, освітленості або руху передають поточні показники, що дозволяє системі відслідковувати зміни в режимі реального часу. Дані від сенсорів можуть оброблятись безпосередньо або накопичуватись для подальшого аналізу.

Для обробки інформації від сенсорів використовують контролери, які являють собою централізовані вузли обробки. Вони отримують сигнали від сенсорів, виконують необхідну логіку керування та формують керуючі команди.

У залежності від складності системи, контролери можуть мати різний рівень обчислювальних можливостей – від базових мікроконтролерів до повноцінних вбудованих систем із багатозадачним програмним середовищем. Програмне забезпечення, що виконується на контролері, зазвичай визначає правила реакції на вхідні дані, враховуючи задані пороги, часові умови чи сценарії.

Контролери також виконують функції узгодження протоколів зв’язку між пристроями, зберігання даних, синхронізації процесів та оновлення стану системи. У багатьох реалізаціях вони є єдиним точкою доступу до зовнішніх мереж або до локального інтерфейсу користувача. У разі виявлення певних умов, контролер приймає рішення про активацію або деактивацію виконавчих механізмів [3, с. 9-11].

Після обробки даних контролер може передати команду до виконавчого пристрою, який виконує фізичну дію. Такі пристрої реалізують зміни у фізичному середовищі – наприклад, вмикають освітлення, регулюють положення заслінки або вмикають електропривід. Вони є кінцевою ланкою в логіці IoT-системи та виконують дії, які безпосередньо впливають на об'єкти керування.

Виконавчі пристрої можуть бути представлені електромеханічними, пневматичними або електронними модулями, які мають інтерфейси керування, сумісні з контролерами. Вони можуть реагувати миттєво або з затримкою, залежно від заданих умов, а також надавати зворотній зв’язок про свій стан. Вибір типу виконавчого елемента визначається специфікою завдання, параметрами навантаження й умовами експлуатації.

1.2 Архітектура розумного дому

Побудова інфраструктури розумного дому передбачає використання розподіленої системи пристроїв, що об’єднані між собою в єдину мережу. Фізичні компоненти, такі як сенсори, актори, контролери й маршрутизатори, взаємодіють через стандартизовані протоколи зв’язку, забезпечуючи передачу команд та даних у реальному часі. Характеристики мережевої взаємодії залежать

від обраних технологій бездротового або дротового з’єднання, які визначають енергоспоживання, дальність передавання сигналу, пропускну здатність та топологію системи.

Для побудови бездротової інфраструктури найчастіше використовуються технології Wi-Fi, ZigBee, Bluetooth та LoRa. Кожен із цих варіантів орієнтовано на певні класи задач. Наприклад, Wi-Fi забезпечує високошвидкісну передачу даних, проте має обмежену масштабованість при великій кількості вузлів. ZigBee передбачає використання мережі з низьким енергоспоживанням та підтримкою топології типу mesh, що дозволяє масштабувати систему без потреби в централізованому маршрутизаторі. Bluetooth, зокрема у варіантах BLE, переважно застосовується для короткодистанційної передачі між персональними пристроями та IoT-вузлами. LoRa забезпечує далекодіапазонну передачу даних із дуже низькою швидкістю, що доцільно для моніторингових систем, розгорнутих на великій території [7].

Типова архітектура розумного дому, представлена на рисунку 1.5, ілюструє класифікацію пристроїв у IoT-системах відповідно до рекомендації ITU-T Y.2060 [23].

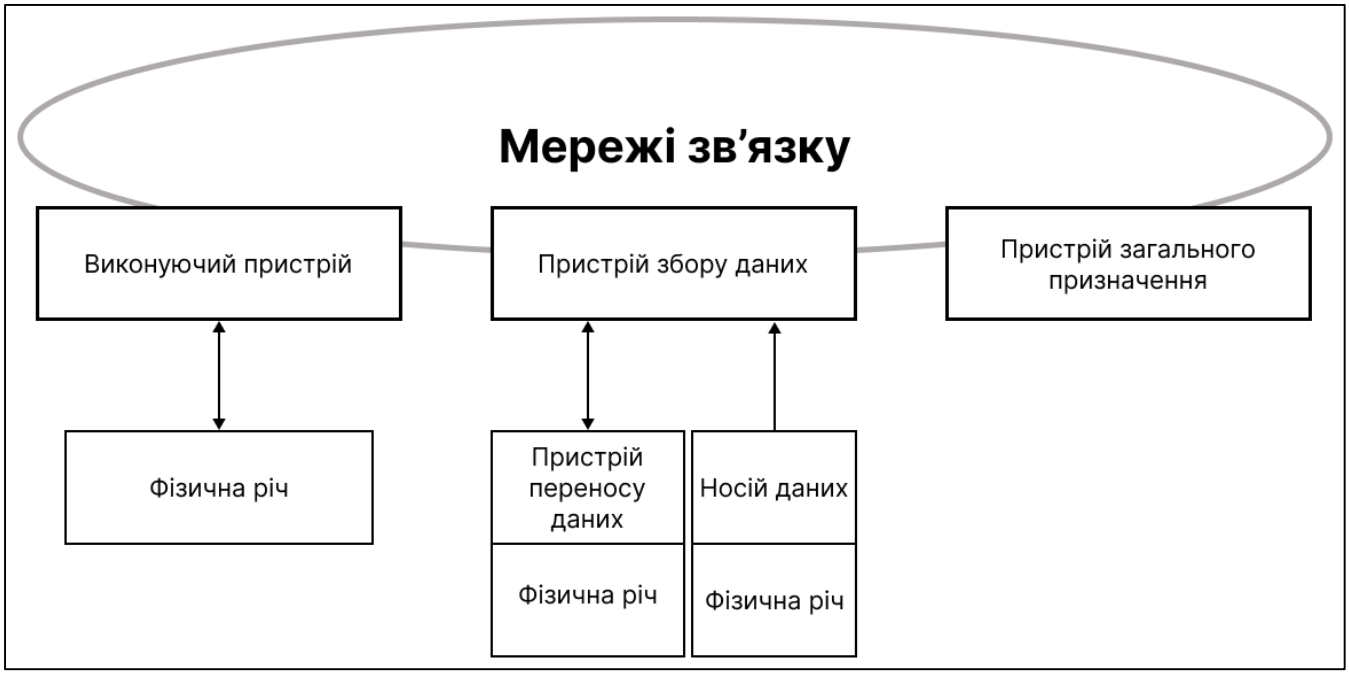


Рисунок 1.5 –Типова архітектурі розумного дому

Пристрої збору даних взаємодіють із фізичними об’єктами для зчитування інформації, що надалі передається до мережі. Пристрої переносу даних забезпечують зв’язок між фізичними об’єктами та комунікаційною інфраструктурою, виступаючи посередниками для передачі даних у мережу. Сенсорні пристрої здійснюють вимірювання параметрів середовища, а виконавчі пристрої перетворюють отримані цифрові команди в дії над фізичним світом. Пристрої загального призначення містять вбудовані можливості обробки і зв’язку, що дозволяє їм безпосередньо взаємодіяти з мережею, прикладами яких є контролери або мобільні пристрої.

Зв’язок між цими категоріями організовується за допомогою різних технологій передачі даних, включно з радіочастотними, інфрачервоними, оптичними або гальванічними методами. Взаємодія може бути прямою або через проміжні пристрої, що забезпечує гнучкість і масштабованість інфраструктури.

Таким чином, архітектура розумного дому відповідає базовим принципам IoT, де система складається з різних класів пристроїв, що виконують спеціалізовані функції для збору, передачі, обробки даних та управління фізичними об’єктами. Така організація забезпечує розподіленість, модульність і адаптивність системи.

1.3 Постановка завдання

Об’єктом розробки є система управління параметрами розумного дому, яка забезпечує дистанційне та локальне керування пристроями, збирання та відображення інформації від сенсорів і виконавчих модулів. Система має реалізувати інтерфейс користувача, який адаптується до різних типів пристроїв.

До функціональних вимог належать:

— забезпечення дистанційного керування пристроями;

— забезпечення локального керування пристроями;

— перегляд стану та параметрів пристроїв у реальному часі;

— підтримка адаптивного інтерфейсу користувача.

Технічні вимоги включають:

— підтримку необхідного рівня безпеки при передачі та збереженні даних;

— забезпечення мінімального часу відгуку системи на події та команди;

— сумісність з основними типами мобільних, десктопних та апаратних платформ.

Для реалізації поставленої мети визначено наступні завдання.

1. Аналіз існуючих IoT-платформ та технічних засобів для вибору архітектурного рішення;

2. Розробка функціональної моделі взаємодії системних компонентів з урахуванням вимог до безпеки та продуктивності;

3. Реалізація користувацького інтерфейсу з підтримкою адаптивності;

4. Проведення тестування програмного забезпечення для оцінки функціональності та сумісності;

5. Забезпечення сумісності системи з мобільними та десктопними тплатформами.

1.4 Вибір апаратної бази

Для IoT-застосунків використовують різноманітні мікроконтролери та одноплатні комп’ютери, які відрізняються за технічними характеристиками, рівнем інтеграції периферії, підтримкою комунікаційних протоколів та енергоспоживанням. Серед найбільш поширених платформ у цій сфері можна виділити Arduino, Raspberry Pi та ESP32. Кожна з них має специфічні особливості, які впливають на вибір апаратної бази залежно від вимог проєкту.

Arduino є платформою, що базується на різних мікроконтролерах, переважно сімейства AVR, з тактовою частотою близько 16 МГц. Ця платформа відома простотою апаратної реалізації та широкою підтримкою периферійних пристроїв. Стандартні моделі Arduino мають обмежену кількість вбудованих інтерфейсів і не включають бездротові засоби зв’язку [12]. Для реалізації

мережевих функцій зазвичай необхідне підключення додаткових модулів, наприклад, Wi-Fi або Bluetooth. Arduino застосовують у проєктах із базовими вимогами до обчислювальної потужності, де пріоритетом є пряме керування апаратурою з мінімальними ресурсами.

Хоча Arduino володіє обмеженою продуктивністю, її архітектура робить цю платформу придатною для навчання та прототипування простих пристроїв. Велика кількість бібліотек та спільнот сприяє швидкому запуску проєктів. Проте через обмеження за пам’яттю і процесорною потужністю Arduino не підходить для задач, що передбачають складну обробку даних або реалізацію бездротових мереж без додаткового обладнання.

Raspberry Pi являє собою одноплатний комп’ютер із процесором ARM, який забезпечує значно вищу обчислювальну потужність у порівнянні з Arduino. Він здатен працювати під керуванням повноцінної операційної системи Linux, що розширює функціональні можливості платформи. Raspberry Pi підтримує різноманітні інтерфейси, включаючи USB, HDMI, Ethernet, а також у деяких моделях – вбудований Wi-Fi і Bluetooth [20].

Завдяки наявності операційної системи Raspberry Pi може виконувати складні обчислювальні завдання, управляти кількома процесами одночасно та інтегруватися з різноманітними мережевими сервісами. Водночас, платформа має більші габарити, споживання енергії і вартість, що може бути недоцільним для компактних або бюджетних IoT-рішень, які вимагають автономності і простоти.

ESP32 – це мікроконтролер із двоядерним процесором, що працює на частоті до 240 МГц. Він має вбудовані модулі Wi-Fi 802.11 і Bluetooth Low Energy, що дозволяє реалізовувати бездротові з’єднання без додаткового обладнання. ESP32 оснащений різноманітними цифровими і аналоговими входами/виходами, а також інтерфейсами SPI, I2C, UART, PWM [15]. Завдяки цьому він підходить для інтеграції з датчиками та виконавчими пристроями у системах розумного дому та IoT.

Нижче наведено порівняльну таблицю 1.1, що містить основні технічні характеристики та функціональні можливості Arduino, Raspberry Pi і ESP32.

Таблиця 1.1 – Порівняння розглянутих платформ

Характеристика	**Arduino Uno**	**Raspberry Pi 4 Model B**	**ESP32**
Процесор	AVR, 16 МГц	ARM Cortex-A72, 1.5 ГГц	Dual-core Tensilica, 240 МГц
ОЗУ	2 КБ SRAM	2-8 ГБ LPDDR4	520 КБ SRAM
Вбудована флеш-пам’ять	32 КБ	microSD (залежить від карти)	4 МБ SPI flash
Підтримка Wi-Fi/Bluetooth	Немає (потрібні модулі)	Wi-Fi 802.11ac, Bluetooth 5.0	Wi-Fi 802.11 b/g/n, BLE
Інтерфейси	Digital I/O, ADC, PWM, UART, I2C, SPI	USB, HDMI, Ethernet, GPIO, I2C, SPI, UART	GPIO, ADC, DAC, SPI, I2C, UART, PWM
Живлення	5 В	5 В USB-C	3.3 В
Енергоспоживання	Низьке	Високе	Помірне
Орієнтовна вартість	$20	$35-$75	$6-$10

Як видно з таблиці вище, ESP32 має компактні розміри та широкий набір вбудованих інтерфейсів, серед яких цифрові та аналогові входи, а також підтримка стандартних протоколів передачі даних. За рівнем енергоспоживання ESP32 характеризується нижчим показником порівняно з Raspberry Pi, що потребує значно більшого живлення.

Водночас, на відміну від Arduino, ESP32 забезпечує більшу обчислювальну потужність та обсяг пам’яті, що дає змогу реалізувати складніші задачі, зокрема обробку мережевих протоколів та підтримку багатозадачності. На рисунку 1.6 наведено вигляд плати ESP32 із позначенням основних елементів [14].

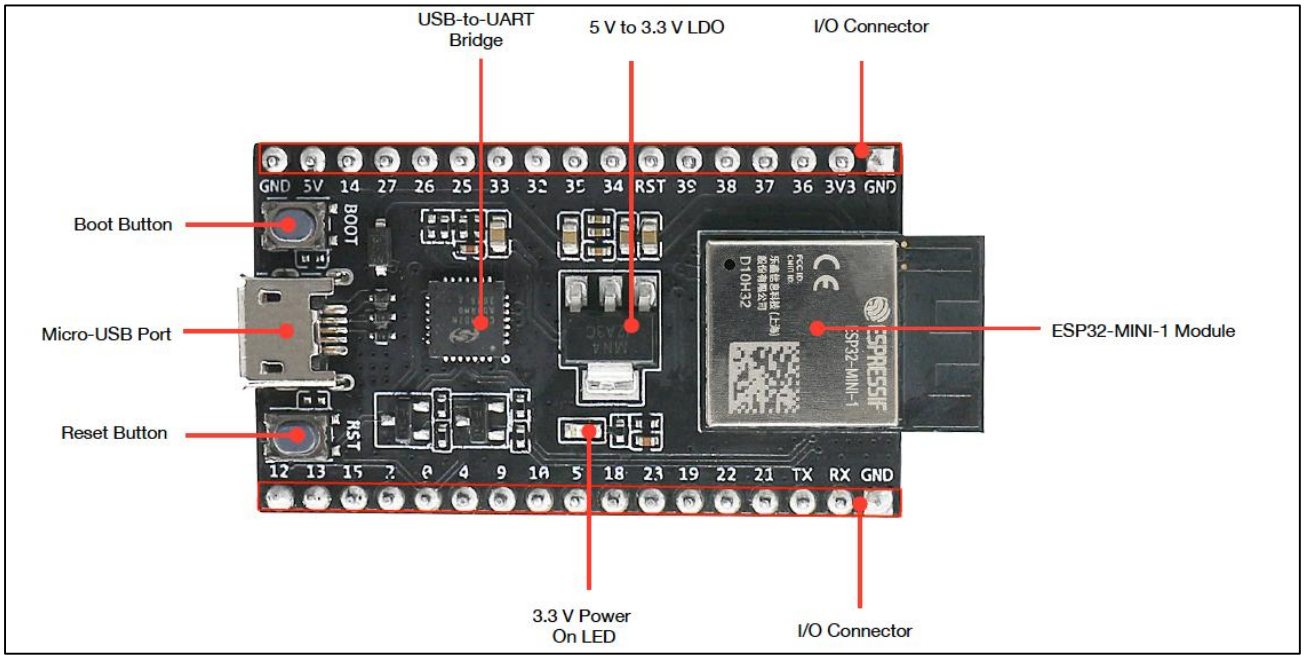


Рисунок 1.6 – Вигляд плати ESP32

Отже, враховуючи співвідношення технічних характеристик, наявність вбудованих бездротових інтерфейсів, компактність та відносно невисоку вартість, було прийнято рішення використати ESP32 як апаратну базу для реалізації системи керування пристроями розумного дому.

РОЗДІЛ 2. ПРОЄКТУВАННЯ IoT-СИСТЕМИ

2.1 Загальна архітектура системи

У запропоній системі керування пристроями Інтернету речей беруть участь кілька взаємодіючих компонентів, кожен з яких виконує визначені функції в процесі збору, передачі та обробки даних. До складу системи входять користувач, інтерфейс взаємодії, сервер обробки даних, брокер MQTT, а також пристрої сенсорного та виконавчого типу. На апаратному рівні сенсорні й виконавчі модулі реалізовані за допомогою плат ESP32, які інтегрують функції збору даних і керування виконавчими механізмами, одночасно забезпечуючи бездротовий зв’язок.

Протокол MQTT (Message Queuing Telemetry Transport) застосовується для організації обміну повідомленнями в межах системи. Він реалізує архітектуру «публікація-підписка», де всі повідомлення проходять через центральний компонент – брокер MQTT, який виступає посередником між відправниками й отримувачами даних. У цьому проєкті як брокер використовується Mosquitto – поширена реалізація MQTT з відкритим кодом, що підтримує необхідний функціонал для обробки повідомлень у мережі IoT [17].

Загальна архітектура системи наведена на рисунку 2.1.

Відповідно до схеми, користувач взаємодіє із системою через графічний інтерфейс, що реалізує прийом команд та відображення отриманої інформації. Інтерфейс передає команди серверу обробки даних, який виконує функції централізованого контролю, зберігання та обробки інформації. Сервер виступає як логічний компонент, що координує роботу системи та здійснює комунікацію з брокером MQTT. Варто зазначити, що брокер MQTT і сервер є різними сутностями, хоча вони можуть бути розміщені на одному фізичному пристрої. Брокер виконує роль маршрутизатора повідомлень, приймаючи публікації від клієнтів та доставляючи їх підписникам згідно з визначеними темами.

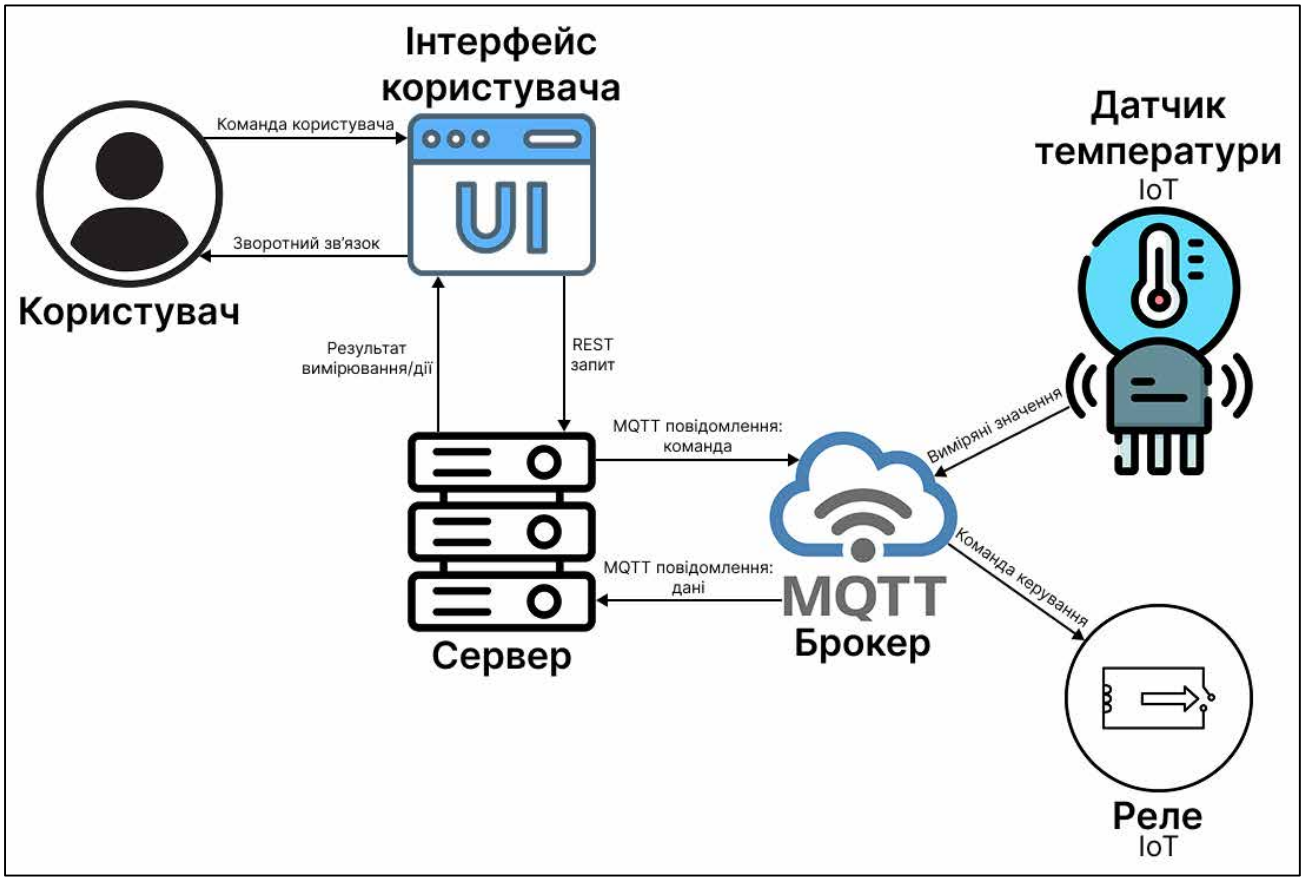


Рисунок 2.1 – Загальна архітектура IoT-системи

Пристрої сенсорного типу, які в даній системі реалізовані на базі ESP32 з підключеними датчиками температури та вологості, передають вимірювані дані у вигляді повідомлень на брокер MQTT. У цій ролі сенсори виступають як «публікатори», що формують та надсилають інформацію про стан навколишнього середовища без очікування зворотного зв'язку. Водночас виконавчі пристрої, також побудовані на ESP32, підписуються на відповідні теми MQTT та приймають команди для керування периферійними виконавчими модулями. У більшості випадків це модулі з релейним інтерфейсом, які застосовуються для ввімкнення або вимкнення живлення навантаження. Такий підхід обумовлений уніфікацією апаратної бази, де незалежно від типу керованого пристрою (освітлення, вентилятори, побутові прилади) виконується однакова логіка комутації. Окремі виконавчі вузли можуть додатково реалізовувати аналогове керування за допомогою широтно-імпульсної модуляції, зокрема у випадку використання світлодіодних стрічок або інших пристроїв, що

потребують регулювання потужності. У цьому випадку релейне керування не забезпечує необхідної функціональності, тому застосовується комбінація цифрових та аналогових методів впливу.

Згідно з рекомендаціями ITU-T Y.2060, в архітектурі Інтернету речей виділяють кілька рівнів та ролей компонентів: Інтернет речі, пристрій переносу даних та носій даних. У розглянутій системі ESP32 з підключеним датчиком формує класичний приклад Інтернет речі, що інтегрує як функції збору інформації (носій даних), так і передачі повідомлень через бездротовий інтерфейс (пристрій переносу даних). Відповідно, внутрішній інтерфейс між сенсором і контролером виступає носієм даних, а мережевий інтерфейс Wi-Fi, що забезпечує з’єднання з брокером MQTT, виконує функцію пристрою переносу даних.

Мережева топологія побудована у вигляді зіркоподібної структури з використанням бездротового середовища передачі. Усі пристрої IoT з’єднані з точкою доступу через інтерфейс IEEE 802.11 (Wi-Fi) [21] у межах однієї локальної мережі. Для надання мережевих параметрів застосовується протокол DHCP, який автоматично призначає IP-адреси вузлам у підмережі. Це дозволяє зменшити навантаження на етапі конфігурації та підтримує динамічну зміну топології без порушення роботи брокера чи серверного компонента. Сервер і брокер MQTT розміщуються у тій же підмережі, що мінімізує затримки та не потребує маршрутизації через зовнішні шлюзи.

З урахуванням можливості розширення кількості сенсорних або виконавчих пристроїв, а також збереження логіки обміну повідомленнями, система підтримує горизонтальне масштабування. У разі збільшення кількості вузлів, вони можуть бути додані до мережі без зміни існуючої інфраструктури. MQTT-протокол забезпечує підтримку великої кількості підключень до брокера за рахунок своєї малоресурсної природи та відсутності необхідності підтримки постійного з’єднання з усіма клієнтами одночасно. На стороні серверного ПЗ масштабування може реалізовуватися шляхом розподілення навантаження між кількома інстанціями або контейнерами з підтримкою горизонтального масштабування.

Архітектура проєкту може бути формалізована за допомогою п’ятирівневої моделі IoT, описаної в роботі Rabah Kenaza [18] й наведена на рисунку 2.2.

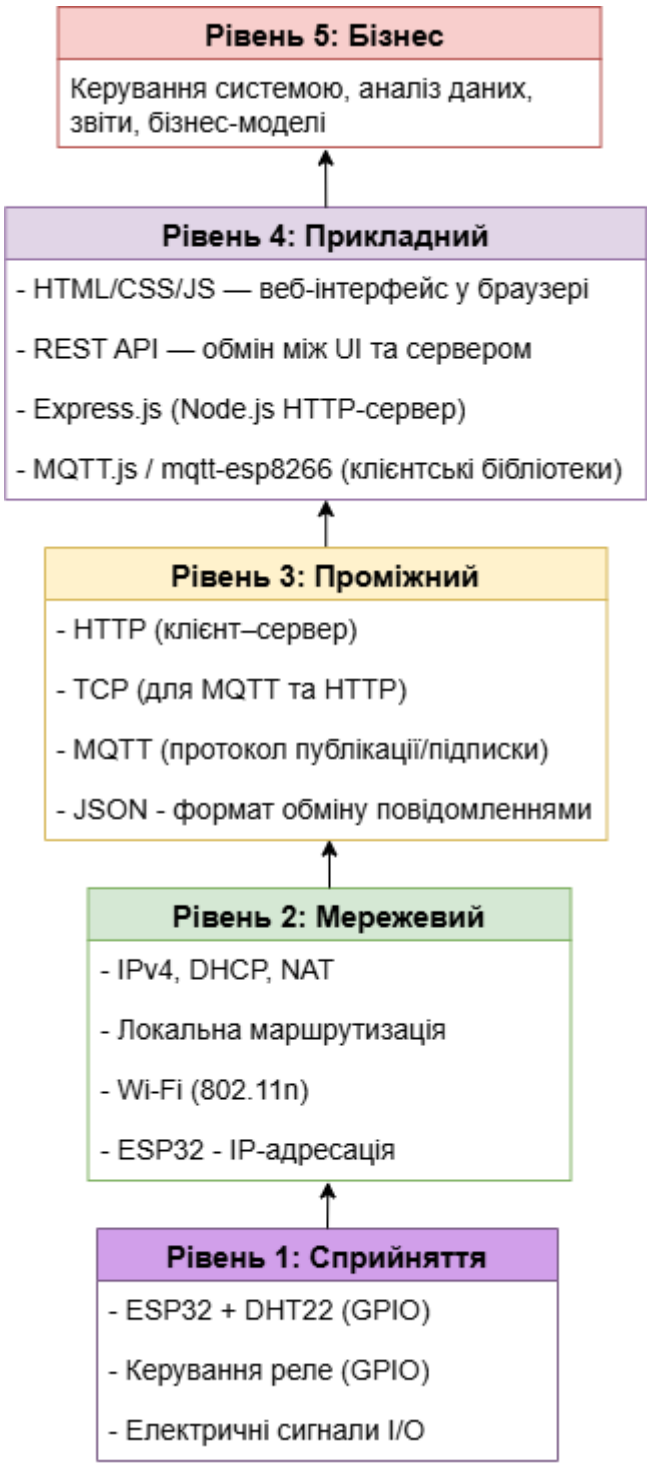


Рисунок 2.2 – Архітектура IoT-системи відповідно до п’ятирівневої OSI-моделі

Відповідно до цієї моделі, система реалізована за рівнями, які послідовно виконують функції від сприйняття фізичних сигналів до надання користувацьких сервісів.

Рівень 1 (Сприйняття) включає апаратну основу системи, яка забезпечує збір і первинну обробку інформації. До цього рівня відносяться датчики температури і вологості DHT22, а також мікроконтролер ESP32, що забезпечує управління периферійними пристроями через GPIO-інтерфейси. На цьому рівні реалізується фізичний зв'язок з навколишнім середовищем, перетворення фізичних сигналів у цифрові дані, а також керування виконавчими механізмами, зокрема реле. Передача електричних сигналів і використання стандартів на кшталт IEEE 802.11 забезпечують основу для подальшої мережевої взаємодії.

Рівень 2 (Мережевий) реалізує передачу даних між пристроями в межах локальної мережі або ширшої інфраструктури. Тут застосовуються протоколи IP для адресації, DHCP для динамічного призначення IP-адрес, а також механізми NAT для трансляції мережевих адрес. Каналом передачі слугує бездротова мережа Wi-Fi стандарту IEEE 802.11n, яка підтримується апаратною платформою ESP32. Цей рівень відповідає за маршрутизацію пакетів, встановлення з'єднань і підтримку стабільного обміну даними між вузлами системи.

Рівень 3 (Проміжний) забезпечує транспортні та комунікаційні протоколи, які керують форматом, надійністю і послідовністю передачі даних. На цьому рівні використовується протокол HTTP для клієнт-серверної взаємодії, а також TCP як транспортний протокол, який гарантує доставку пакетів у правильному порядку. Для обміну повідомленнями між пристроями IoT застосовується протокол MQTT, що працює за моделлю публікації/підписки і підтримує рівні якості обслуговування (QoS). Формат JSON служить стандартом для структурування переданих даних, що забезпечує сумісність між різними компонентами системи.

Рівень 4 (Прикладний) включає програмне забезпечення, що реалізує бізнес-логіку системи і надає сервіси для користувачів та інших рівнів. Серверна частина реалізована на основі Node.js із фреймворком Express.js [16], що

забезпечує обробку HTTP-запитів і управління API. Для підключення до MQTT-брокера використовуються клієнтські бібліотеки MQTT.js та mqtt-esp8266, які забезпечують зв’язок із пристроями на ESP32. Користувацький інтерфейс реалізований як веб-застосунок з використанням HTML, CSS та JavaScript, що дозволяє взаємодіяти із системою через браузер.

Рівень 5 (Бізнес) відповідає за загальне управління системою, обробку отриманих даних, аналіз і прийняття рішень на основі бізнес-логіки. На цьому рівні реалізуються функції контролю, моніторингу та конфігурації системи, а також формуються звіти і ведеться архівування інформації. Цей рівень може включати алгоритми аналізу даних, побудови моделей і прийняття управлінських рішень для оптимізації роботи всієї IoT-інфраструктури. У представленій архітектурі рівень бізнес-логіки інтегрований із прикладним рівнем, що дозволяє централізовано керувати всіма компонентами системи.

2.2 Функціональне моделювання

Проведено функціональне моделювання системи шляхом побудови UML-діаграм, що відображають взаємодію між основними учасниками. На даному етапі виконано формалізацію функцій інтерфейсу керування, користувача, сенсорів та виконавчих пристроїв у межах системи розумного дому.

Відповідно до поставленої задачі, основна увага зосереджена на інтерфейсній частині системи, яка виконує функції обробки команд користувача, передачі даних до пристроїв та отримання зворотного зв’язку. Для представлення загальної картини взаємодій побудовано діаграму варіантів використання, на якій визначено таких акторів: Користувач, Сенсор, Виконавчий пристрій та UI-система IoT. Користувач ініціює дії системи, пов’язані з керуванням пристроями, переглядом інформації та налаштуванням автоматизації. UI-система IoT виступає центральним об’єктом, що забезпечує обробку всіх запитів і взаємодію з апаратною частиною.

На рисунку 2.2 подано діаграму варіантів використання [6], яка ілюструє функціональні можливості системи та розподіл відповідальностей між акторами. UI-система IoT реалізує варіанти використання «Авторизація», «Перегляд стану пристроїв», «Перегляд даних пристроїв», «Увімкнення пристрою», «Вимкнення пристрою» та «Налаштування графіка». Окремі дії передбачають включення допоміжних процесів, таких як «Надання даних» від сенсора або «Повідомлення стану» від виконавчого пристрою, що позначено на діаграмі за допомогою зв’язків типу <<include>>.

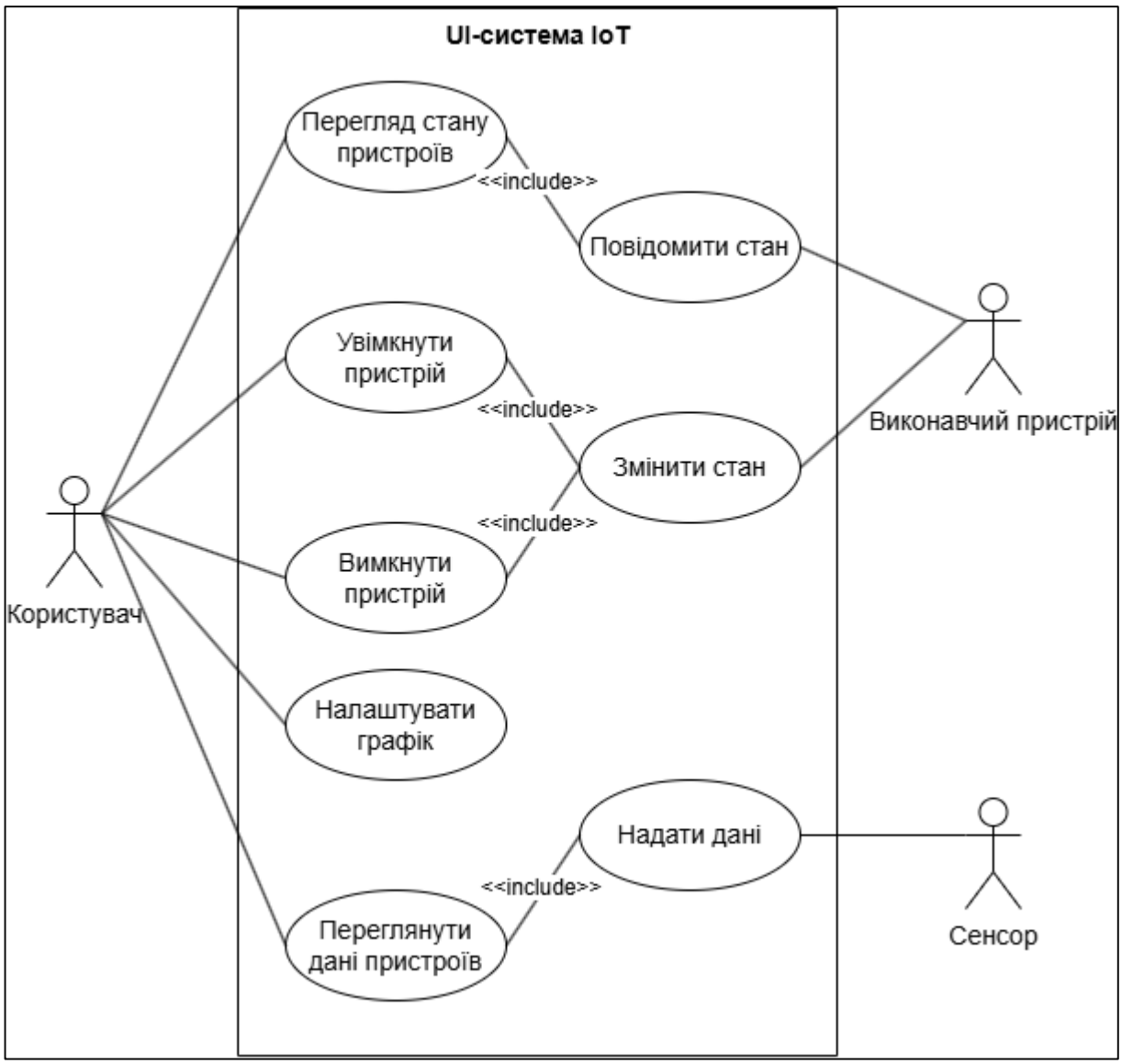


Рисунок 2.3 – Діаграма варіантів використання

Діаграма варіантів використання відображає лише зовнішню поведінку системи, не деталізуючи внутрішні механізми обробки повідомлень чи специфіку протоколів взаємодії. Для цього нижче наведено діаграми послідовностей [2], які демонструють динаміку обміну повідомленнями між об’єктами системи в різних сценаріях. У цих діаграмах UI-система IoT розглядається як єдиний логічний компонент, що поєднує функціональність користувацького інтерфейсу, сервера та MQTT-брокера.

Сценарій керування пристроєм наведено на рисунку 2.3. Користувач через інтерфейс відкриває сторінку пристроїв, після чого система відображає актуальні стани виконавчих пристроїв. Далі користувач може надіслати команду на увімкнення або вимкнення пристрою, що передається відповідному виконавчому пристрою через UI-систему. Отримавши підтвердження виконання команди, система оновлює інформацію про стан пристрою та відображає її користувачу.

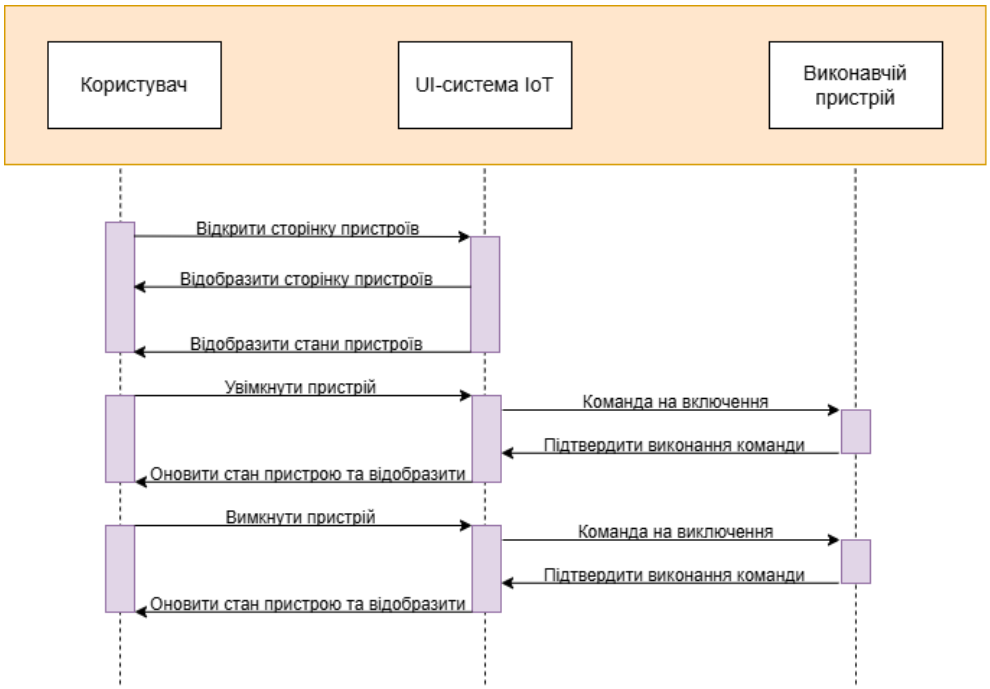


Рисунок 2.4 – Послідовність взаємодії для керування виконавчим пристроєм

На рисунку 2.4 наведено послідовність операцій для отримання даних із пристрою. Користувач ініціює запит на отримання інформації, який надсилається

до сенсора через UI-систему. Після отримання актуальних даних сенсора система обробляє інформацію та відображає її у користувацькому інтерфейсі.

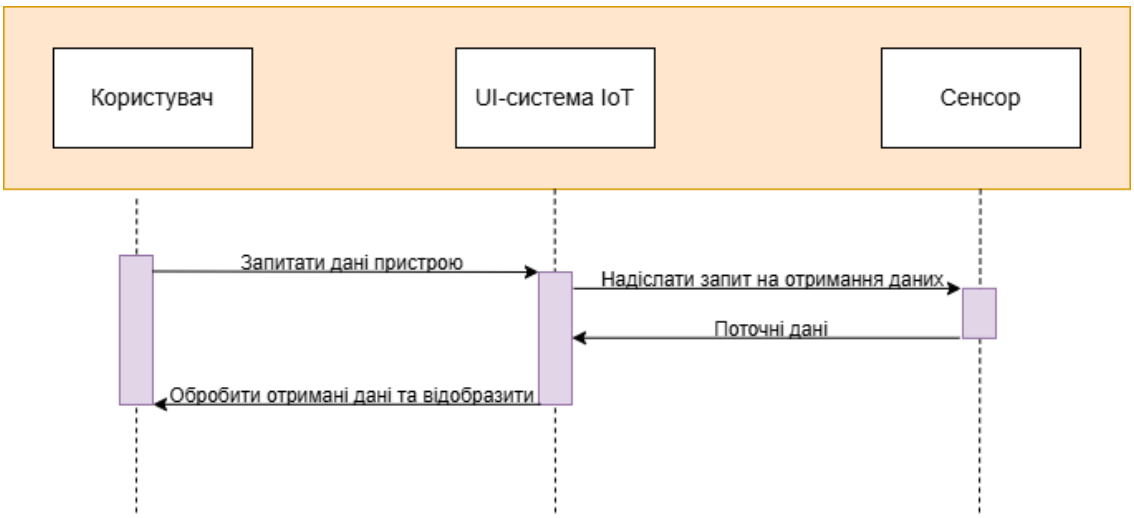


Рисунок 2.5 – Послідовність отримання даних із сенсора

Рисунок 2.5 демонструє процес налаштування автоматизації, що відбувається без участі безпосередніх IoT-пристроїв. Користувач відкриває інтерфейс налаштувань автоматизації, задає правила та зберігає їх у базі даних системи. Після успішного збереження система підтверджує виконання операції та відображає відповідний результат користувачу.

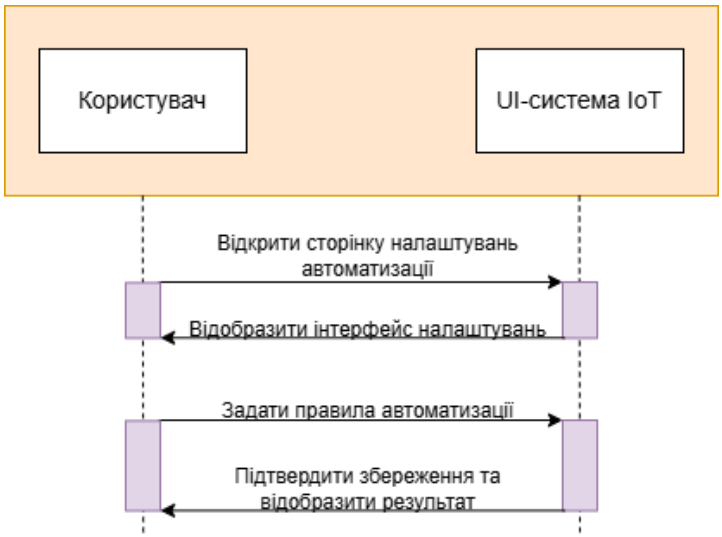


Рисунок 2.6 – Послідовність налаштування автоматизації в системі

Також на рисунку 2.7 наведено діаграму сутностей [1], що відображає основні компоненти системи та їх взаємозв’язки.

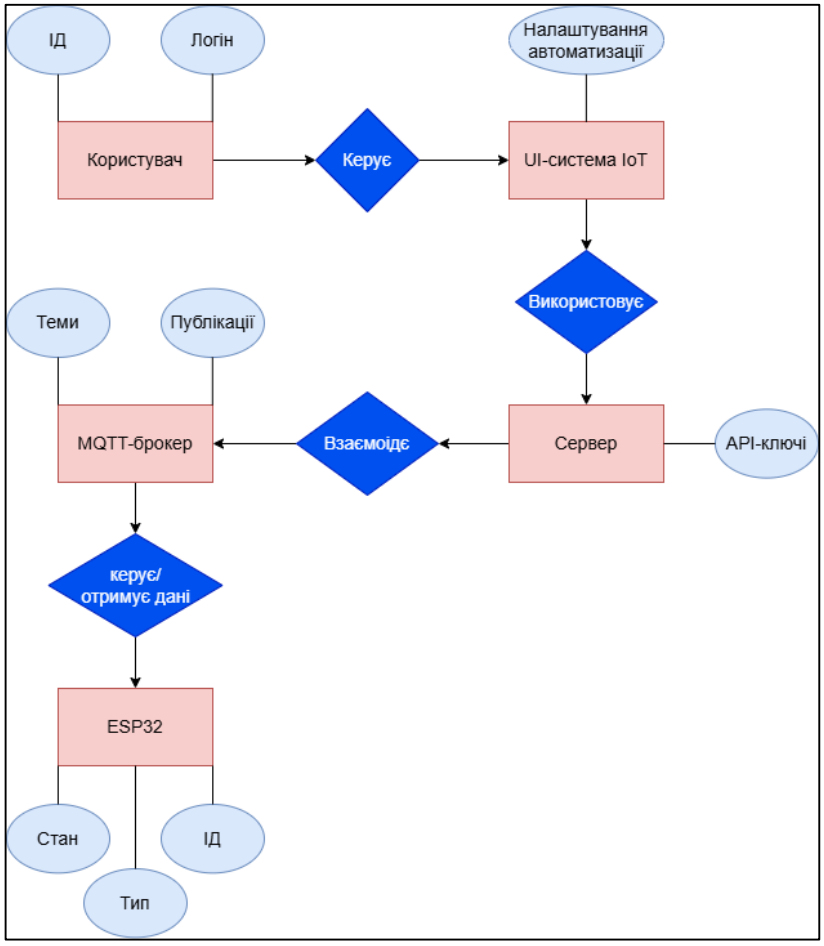


Рисунок 2.7 – Діаграма сутностей

Сутність Користувач, яка містить атрибути ідентифікатора, логіна та ролі, взаємодіє з UI-системою IoT, передаючи команди та отримуючи інформацію про стан системи. UI-система IoT, що включає атрибут конфігурацій автоматизацій, забезпечує управління логікою роботи пристроїв та зберігає відповідні налаштування.

Сервер, з атрибутами ідентифікатора та API-ключів, взаємодіє з UI-системою IoT для обробки запитів та забезпечення доступу до функцій системи. MQTT-брокер, що характеризується ідентифікатором та темами підписки і публікації, служить посередником для обміну повідомленнями між компонентами системи.

ESP32 як окрема сутність містить атрибути ідентифікатора, типу (сенсор або виконавчий пристрій), стану та параметрів. Вона підключається до UI-системи IoT, надаючи дані сенсорів або виконуючи команди виконавчих пристроїв.

2.3 Проєктування UI

Проєктування інтерфейсу користувача проводиться у векторному редакторі Figma. Розробка макету у Figma розпочинається зі створення нового проєкту, до якого додаються вісім екранів для кожного розділу панелі. Головна панель містить бічну панель із розділами та основний простір для відображення сповіщень, статусу температури і енергії, а також швидкого доступу до нещодавно використаних пристроїв. На екрані панелі додається бічна панель з іконкою будинку, назвою панелі та розділами, які представлені у текстовій формі. Такий макет зображено на рисунку 2.8.

Рисунок 2.8 – Загальний вигляд головної панелі з бічною навігацією

Елементи макету групуються для спрощення переміщення. Шаблон копіюється сім разів, що разом із головною панеллю утворює вісім розділів.

Кожен екран заповнюється необхідними компонентами: віконцем для сповіщень, картками статусу та картками пристроїв у швидкому доступі. Також додається кнопка «Очистити все». Відповідне оформлення представлено на рисунку 2.9.

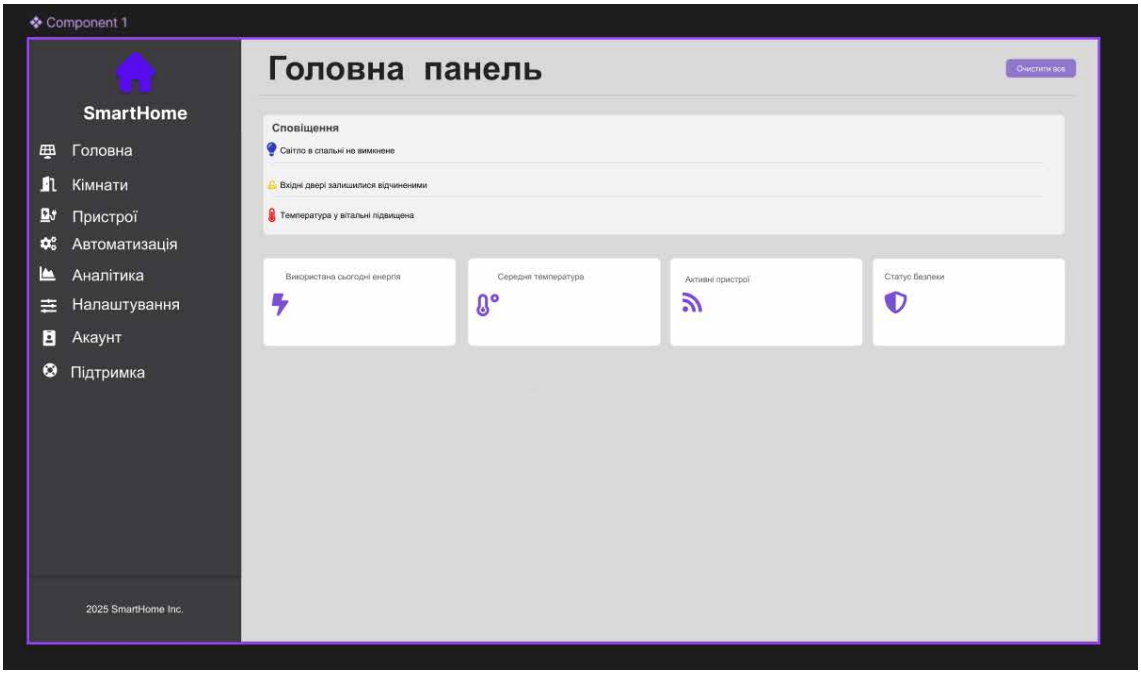


Рисунок 2.9 – Шаблон екрану панелі з компонентами сповіщень та статусів

Для іконок використовується плагін Figma «Font Awesome Icons». Приклад набору іконок наведено на рисунку 2.10.

Рисунок 2.10 – Приклад використання іконок із плагіна Font Awesome Icons

Розділ «Кімнати» призначений для додавання нових кімнат та керування пристроями у них. Він складається зі списку кімнат з перемиканням через

вкладки та 2D-планом кімнати з пристроями. Керування пристроями винесене окремо збоку екрану. На відміну від головного екрану, тут присутні кнопки «Додати кімнату» та «Додати пристрій», а також великий контейнер для відображення вмісту кімнати. Стрічка для перемикання між вкладками реалізує функцію вибору кімнати. Пристрої розміщуються внизу контейнера, а керування ними – збоку. Загальний вигляд розділу представлено на рисунку 2.11.

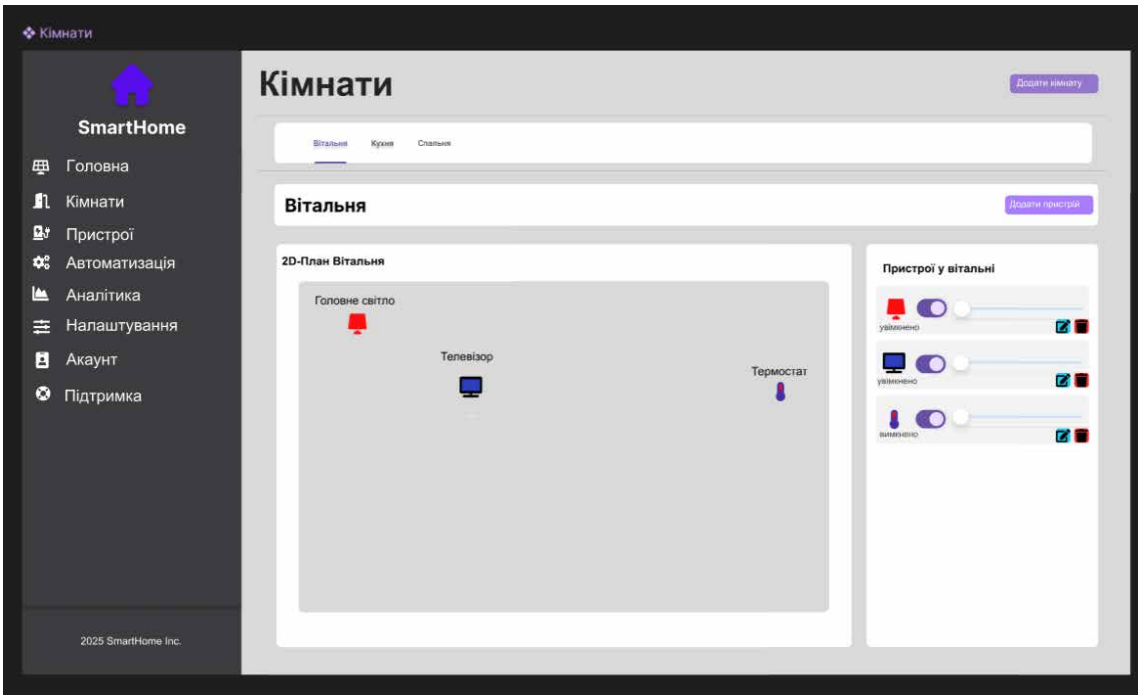


Рисунок 2.11 – Інтерфейс розділу «Кімнати» з 2D-планом і керуванням пристроями

Розділ «Пристрої» забезпечує додавання, видалення та керування пристроями. Екран містить пошуковий рядок із підказкою «Пошук пристроїв» та два випадаючі списки з відповідним контентом. Додана кнопка «Додати новий пристрій» і три картки пристроїв. Елементи згруповані для зручності. Ілюстрація розділу наведена на рисунку 2.12.

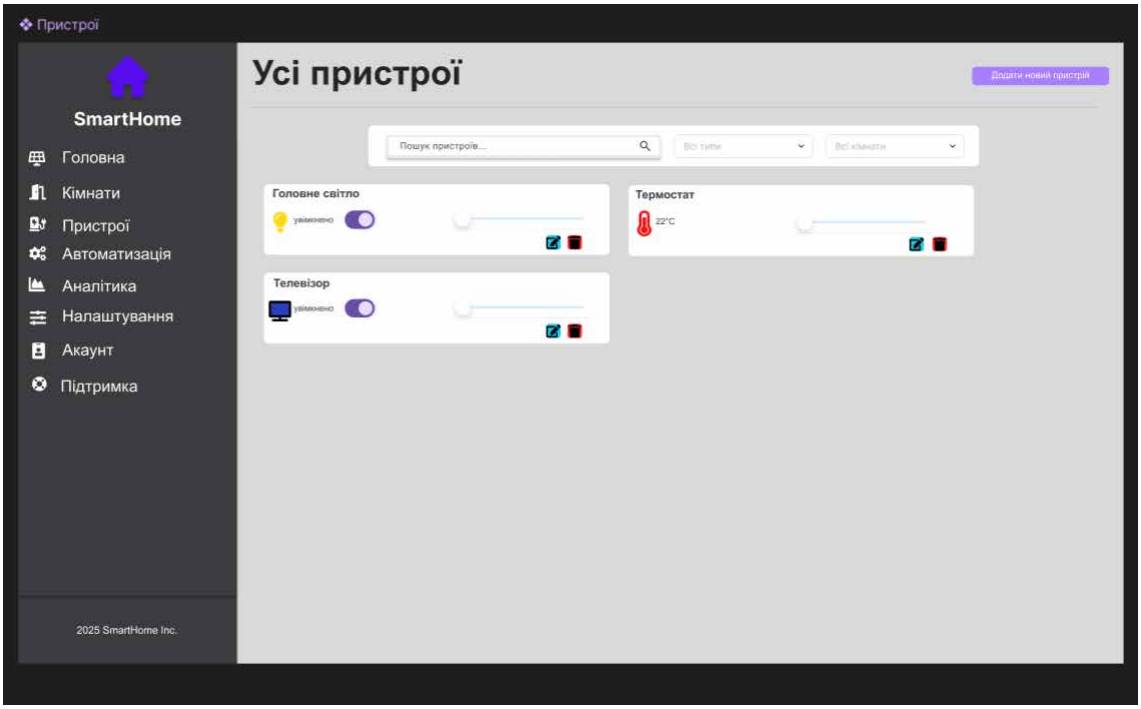


Рисунок 2.12 – Екран розділу «Пристрої» з функціями пошуку та додавання

Розділ «Автоматизація» служить для створення сценаріїв роботи пристроїв. Тут розміщується перелік сценаріїв, кнопка їх створення та назва розділу. Для кожного сценарію створюється окреме вікно з текстовим наповненням, перемикачем, кнопками активації і видалення, а також статусом сценарію. Вигляд цього розділу зображено на рисунку 2.13.

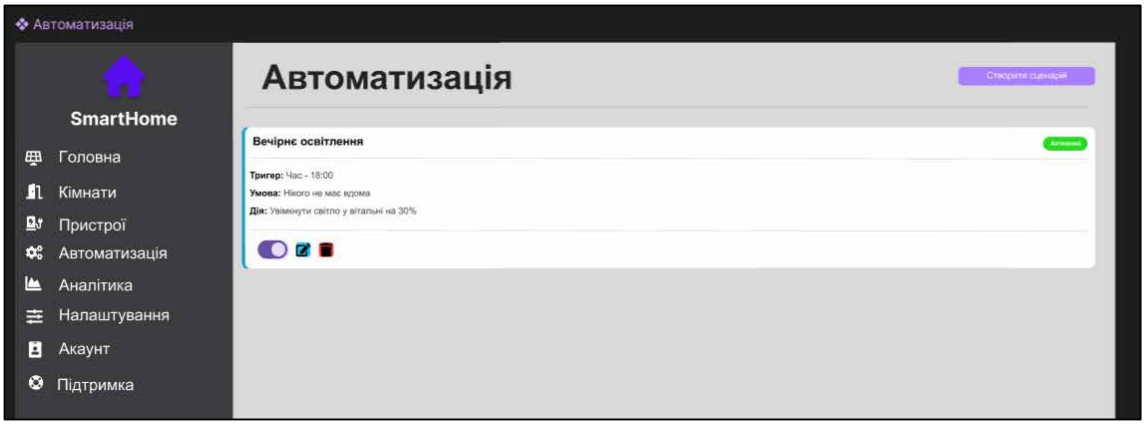


Рисунок 2.13 – Інтерфейс розділу «Автоматизація» зі списком сценаріїв

Розділ «Аналітика» містить статистичні дані у вигляді графіків, наприклад, активність пристроїв за певний період часу. На поточному етапі реалізовані лише

два випадаючі списки для вибору параметрів графіків. Візуалізація наведена на рисунку 2.14.

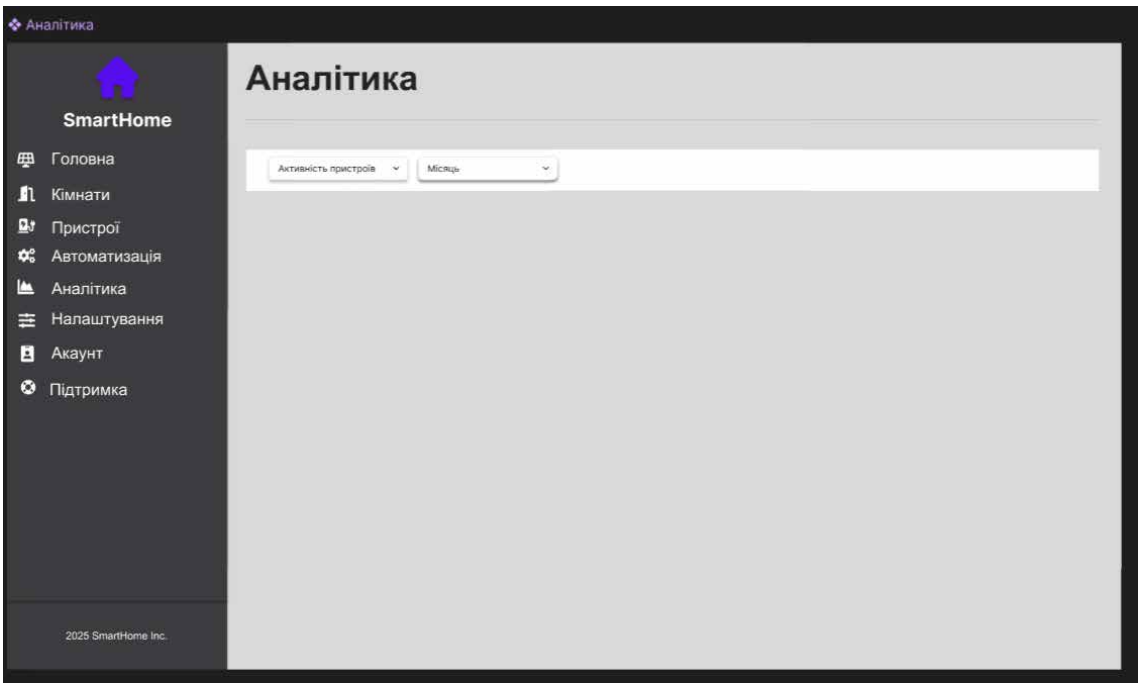


Рисунок 2.14 – Екран розділу «Аналітика» з параметрами графіків

У розділі «Налаштування» передбачено опції зміни теми оформлення, мови інтерфейсу та параметрів сповіщень. Створено три підкладки з назвами «Тема оформлення», «Мова інтерфейсу» та «Сповіщення». Для теми оформлення використовується випадаючий список із варіантами світлої та темної тем. Аналогічно налаштовано список мов – українська та англійська. Макет цього екрану відображено на рисунку 2.15.

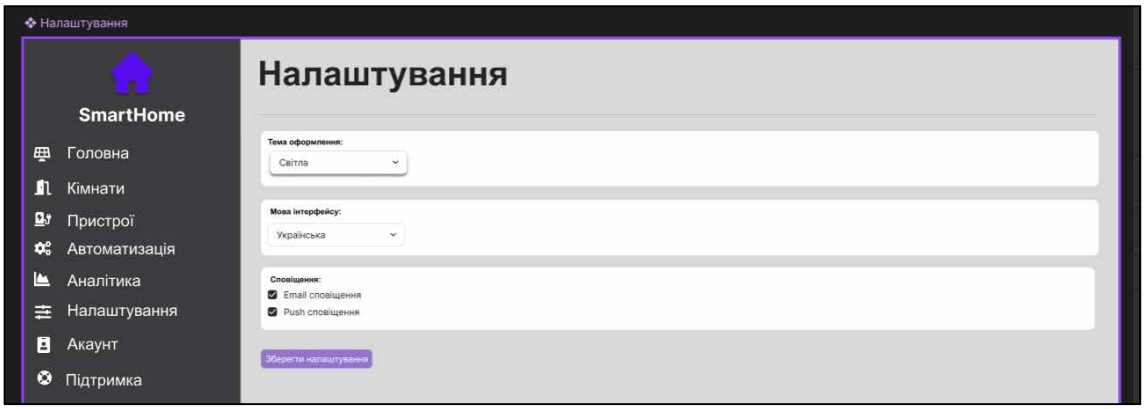


Рисунок 2.15 – Екран налаштувань з вибором теми та мови інтерфейсу

Налаштування акаунту передбачає конфігурацію таких параметрів, як ім’я користувача, електронна адреса та пароль. Екран містить заголовок та п’ять підкладок із відповідними підписами. Для поля «Ім’я користувача» додано текстове введення з текстом за замовчуванням. Поле «Email» містить підпис demo@example.com. Парольні поля розподілені на поточний, новий та підтвердження пароля з підказкою у вигляді placeholder для поточного пароля. Внизу розміщена кнопка «Оновити пароль». Розміщення елементів ілюструється на рисунку 2.16.

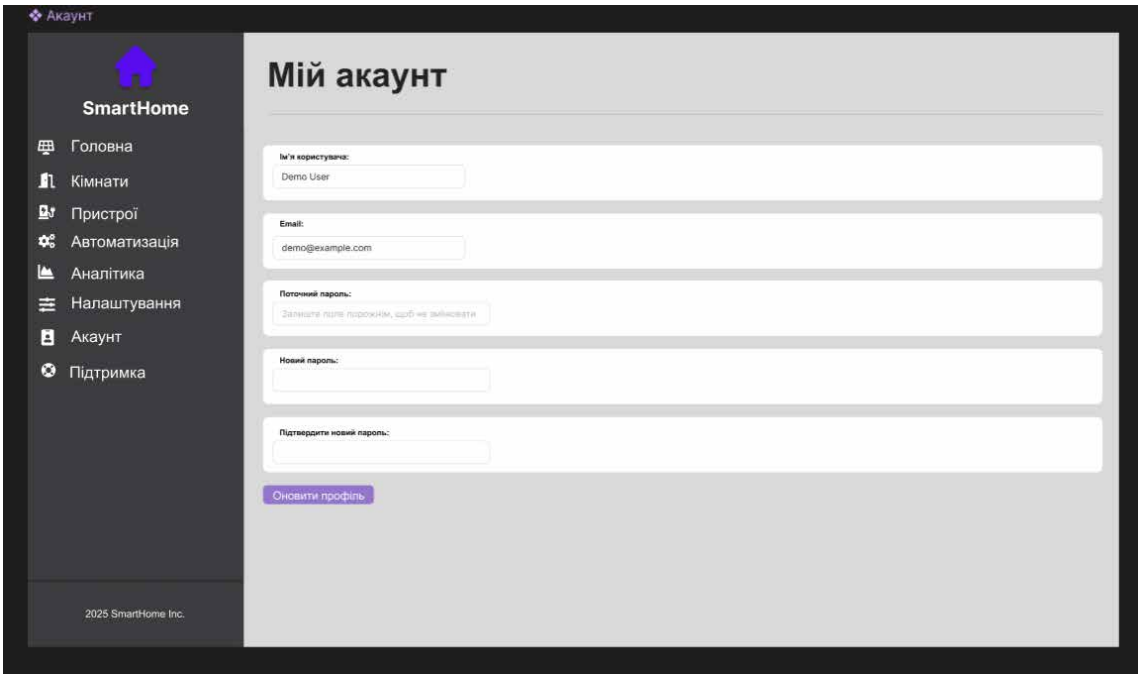


Рисунок 2.16 – Екран налаштувань акаунту з полями введення та кнопкою оновлення

Розділ «Підтримка» включає функції зв’язку з технічною підтримкою та часті запитання (FAQ). Екран поділено на дві частини з відповідними підкладками. Зліва знаходиться вікно FAQ із заголовками у вигляді тексту, справа – вікно «Зв’яжіться з нами» з чотирма полями для введення: «Ваше ім’я», «Ваш email», «Тема», а також великим текстовим полем для повідомлення і кнопкою «Надіслати». Конструкція цього екрану наведена на рисунку 2.17.

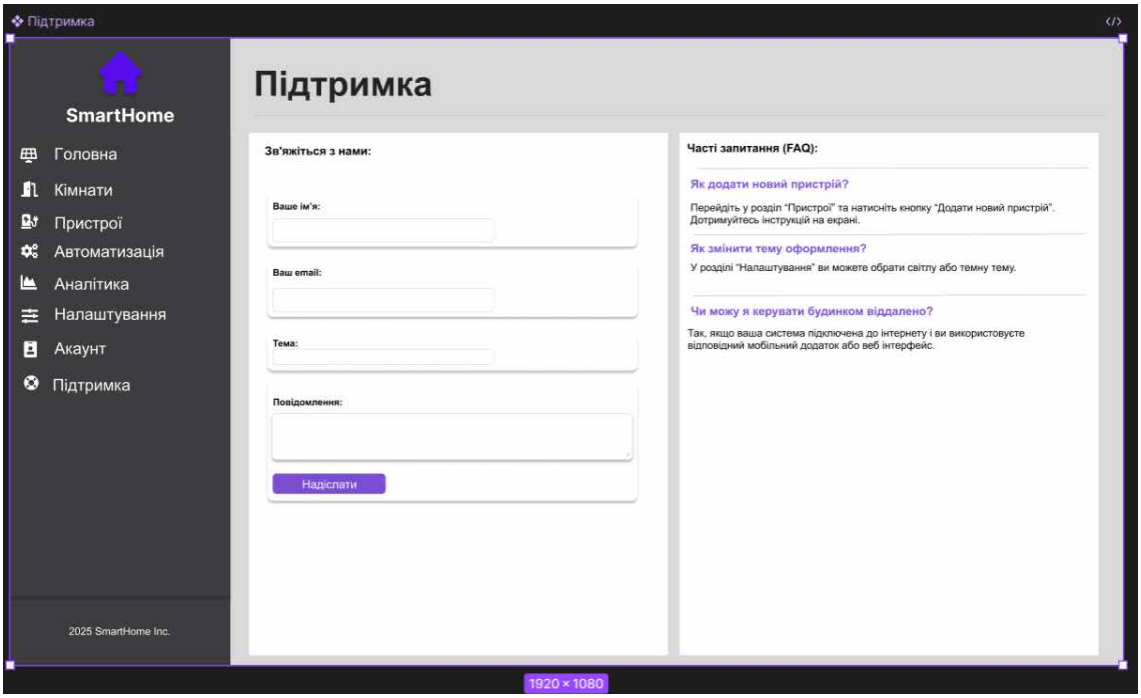


Рисунок 2.17 – Екран розділу «Підтримка» з формою зворотного зв’язку та FAQ

Всі екрани перетворюються на компоненти за допомогою функції «Create component» для подальшого прототипування. Налаштування переходів між розділами виконується в режимі прототипу, реалізовано прості переходи без спеціальних ефектів. Загальна схема переходів показана на рисунку 2.18.

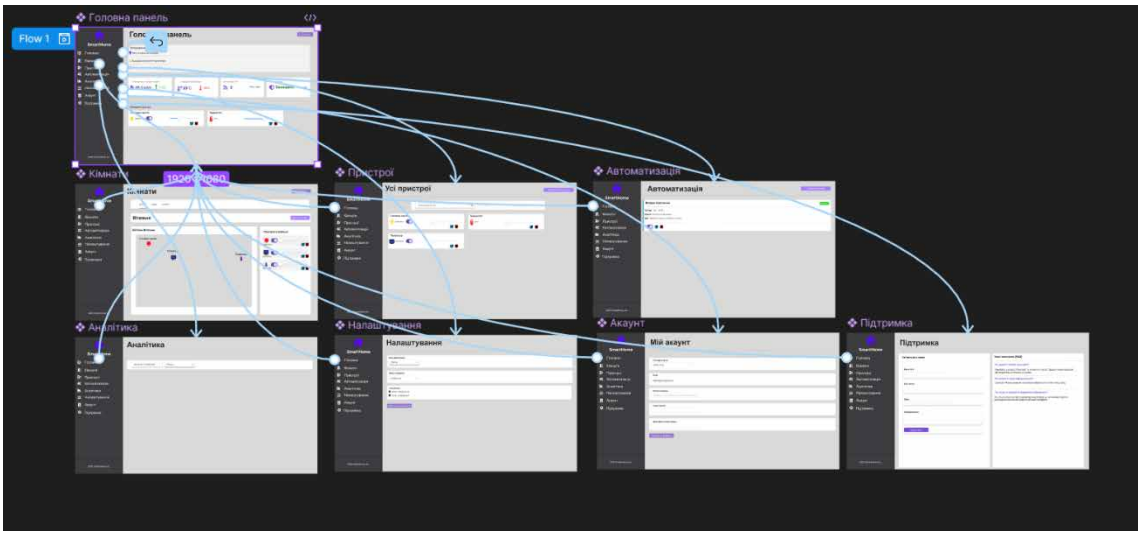


Рисунок 2.18 – Схема переходів між розділами інтерфейсу

2.4 Логічна модель взаємодії

Обмін інформацією у системі здійснюється через MQTT-топіки, поділені на дві основні категорії: sensors/ та actuator/. Топік sensors/ використовується для публікації даних сенсорів, наприклад температури і вологості від ESP32 з датчиком DHT. Повідомлення містять такі параметри: device_id, temperature, humidity, timestamp. Топік actuator/ призначений для команд виконавчим пристроям, таким як реле на базі ESP32. Формат повідомлень включає поля device_id, command, timestamp.

Обробка повідомлень відбувається сервером, який підписується на обидва топіки. Повідомлення з sensors/ оновлюють внутрішній стан пристроїв, а команди з actuator/ пересилаються виконавчим пристроям для реалізації керування. Така архітектура дозволяє розмежувати потоки даних і забезпечує асинхронний обмін інформацією.

Таблиця 2.1 – Опис MQTT-топіків та форматів повідомлень

Топік	**Джерело**	**Поля повідомлення**
sensors/	ESP32 з датчиком	- device_id - temperature - timestamp
actuator/	Сервер/Клієнт	- device_id - command - timestamp

Рисунок 2.19 ілюструє алгоритм роботи сенсорного пристрою на базі ESP32 з підключеним датчиком температури та вологості. Після запуску мікроконтролера виконується ініціалізація мережевого з'єднання та MQTT-клієнта. Далі пристрій переходить до циклічного зчитування даних з датчика. Отримані значення температури і вологості разом з ідентифікатором пристрою та часовою міткою формуються у повідомлення, яке публікується у відповідний MQTT-топік sensors/. Після цього виконується затримка до наступного циклу вимірювання

Рисунок 2.19 – Блок-схема алгоритму відправки даних сенсора

Рисунок 2.20 відображає алгоритм роботи виконавчого пристрою, що реагує на команди, отримані через MQTT-протокол. Після запуску пристрою виконується підключення до мережі та ініціалізація MQTT-клієнта з підпискою на топік actuator/. У циклі очікується надходження повідомлень. Кожне отримане повідомлення перевіряється на відповідність ідентифікатору пристрою. Якщо ID збігається, з повідомлення зчитується команда, яка виконується шляхом перемикання вихідного цифрового сигналу. Команди можуть відповідати вмиканню або вимиканню керованого елемента, наприклад реле. Після виконання команди пристрій повертається в режим очікування наступного повідомлення.

Рисунок 2.20 – Блок-схема алгоритму отримання команди з інтерфейсу

РОЗДІЛ 3. РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Вибір інструментів розробки

Для створення вебінтерфейсу системи застосовано комбінацію мов HTML, CSS та JavaScript. Такий підхід забезпечує платформонезалежний доступ до користувацького інтерфейсу через будь-який браузер без встановлення додаткового програмного забезпечення. Це дозволяє реалізувати централізоване керування з різних типів пристроїв, включно з мобільними.

Код інтерфейсу працює на клієнтській стороні та взаємодіє з MQTT-брокером за допомогою бібліотеки MQTT.js. Ця бібліотека реалізує повноцінний MQTT-клієнт для JavaScript та дозволяє реалізувати підписку на топіки, обробку отриманих повідомлень та публікацію команд. MQTT.js підтримує всі необхідні функції для інтеграції з брокером, зокрема налаштування QoS та обробку втрати з’єднання.

Для серверної частини було обрано Express.js – мікрофреймворк, що працює у середовищі виконання Node.js. Express.js надає мінімальний інтерфейс для створення маршрутизованих HTTP-запитів, що дозволяє організувати логіку обробки запитів клієнта та взаємодію з MQTT-клієнтом, який реалізований у середовищі Node.js.

Node.js забезпечує неблокуючу модель введення/виведення, що є доцільним у системах з подієво-орієнтованою архітектурою, як-от MQTT. Це дозволяє обробляти вхідні повідомлення від декількох пристроїв одночасно без потреби у створенні окремих потоків для кожного з’єднання, що знижує навантаження на сервер.

Програмування мікроконтролерів ESP32 виконано у середовищі Arduino IDE. Цей інструмент підтримує платформу ESP32 через відповідний набір бібліотек та дає змогу реалізовувати прошивки з підключенням до Wi-Fi, зчитуванням даних з датчиків, обробкою команд керування тощо.

Arduino IDE містить компілятор, інструмент для завантаження коду через USB та серійну консоль, що дозволяє виконувати налагодження під час розробки.

Бібліотеки для ESP32 реалізують API для роботи з мережею, підключення до MQTT-брокера, та роботи з периферійними модулями, зокрема цифровими та аналоговими входами/виходами.

Для забезпечення публікації даних сенсором і приймання команд виконавчим пристроєм було обрано бібліотеку PubSubClient [19], яка реалізує MQTT-клієнт для Arduino-сумісних платформ. Ця бібліотека дозволяє здійснювати публікацію повідомлень у заданий топік і підписку на вхідні повідомлення з подальшим викликом callback-функції.

Перед початком використання ESP32 у проєкті було встановлено додаткові пакети з підтримкою цієї платформи у середовищі Arduino IDE. Це забезпечило доступ до бібліотек керування Wi-Fi, а також дозволило використовувати внутрішні таймери, обробники переривань та інші апаратні ресурси мікроконтролера.

У проєкті використано стандартну реалізацію Wi-Fi підключення через бібліотеку WiFi.h, яка ініціалізує з’єднання з мережею з наданням статичної або динамічної IP-адреси. Після встановлення з’єднання, мікроконтролер виконує підключення до MQTT-брокера та переходить у режим обміну даними.

У якості програмного забезпечення для реалізації брокера MQTT було застосовано Mosquitto. Цей брокер є відкритим програмним продуктом, який підтримує останні версії протоколу MQTT та відповідає специфікаціям MQTT v3.1.1. Його використання забезпечує мінімальні затримки в обробці повідомлень та сумісність з широким спектром MQTT-клієнтів.

Mosquitto підтримує підписку на різні топіки, механізм збереження сесій, обробку автентифікації користувачів, а також журналювання подій, що дозволяє забезпечити контроль і діагностику роботи системи. У рамках даної реалізації брокер розгорнуто локально на тій же машині, що й серверна частина системи.

Вибір Mosquitto був зумовлений його стабільністю, підтримкою широкого функціоналу та відкритим кодом. Це забезпечує можливість розширення конфігурації у майбутньому без прив'язки до комерційних рішень. Крім того, його встановлення та налаштування не вимагають значних ресурсів системи.

У процесі конфігурації брокера було визначено базові параметри, такі як порт з’єднання, список дозволених топіків та механізм автентифікації через логін і пароль. Це дозволяє обмежити доступ до системи лише авторизованим клієнтам, що відповідає вимогам до захисту мережевої взаємодії.

Серверна частина програми, що виконується на Node.js, містить MQTT-клієнт, реалізований за допомогою пакета mqtt. Цей клієнт дозволяє серверу підписуватись на повідомлення, опубліковані сенсорами, а також публікувати команди до топіку actuator/ у відповідь на події або запити користувача.

Для побудови інтерфейсу застосовано адаптивну вёрстку з використанням CSS Flexbox, що забезпечує коректне відображення на екранах різного розміру. JavaScript забезпечує обробку подій, динамічне оновлення значень сенсорів та надсилання керуючих команд.

Таким чином, комбінація використаних інструментів дозволяє реалізувати систему, в якій пристрої з мікроконтролерами взаємодіють із сервером і користувачем через єдиний MQTT-брокер, використовуючи стандартні протоколи та бібліотеки.

3.2 Програмний код клієнтського інтерфейсу

Розробка інтерфейсу IoT-панелі розумного дому починається з визначення загальної структури, яка включає три основні елементи: основну панель контенту, бічну панель навігації та модальні вікна. Бічна панель реалізує функцію перемикання між розділами: Головна панель, Кімнати, Пристрої, Автоматизація, Аналітика, Налаштування, Акаунт користувача та Підтримка. Основна частина відображає відповідний контент залежно від обраного пункту навігації. Модальні вікна накладаються поверх основного інтерфейсу та використовуються для введення або редагування даних.

На лістингу 3.1 представлено шаблон HTML-документу з назвою «SmartHome Dashboard». Він містить базову структуру сторінки, де контейнер з класом app-container включає два основні блоки: бічну панель aside та головну частину main.

Лістинг 3.1 – Базова структура HTML-документу SmartHome Dashboard

```
<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>SmartHome Dashboard</title>
</head>
<body>
    <div class="app-container">
        <aside class="sidebar"></aside>
        <main class="main-content"></main>
    </div>
    <script src="..."></script>
</body>
</html>
```

У лістингу 3.2 реалізовано розмітку бічної панелі навігації, яка містить заголовок із логотипом та назвою, список пунктів меню з відповідними іконками з бібліотеки Font Awesome, а також футер із копірайтом.

Лістинг 3.2 – Структура бічної панелі навігації

```
<aside class="sidebar">
    <div class="sidebar-header">
        <i class="fas fa-home"></i>
        <h1>SmartHome</h1>
    </div>
    <nav class="sidebar-nav">
        <ul>
            <li class="active" data-target="dashboard"><i
class="fas fa-tachometer-alt"></i> <span>Головна</span></li>
            <li data-target="rooms"><i class="fas fa-door-
open"></i> <span>Кімнати</span></li>
            <!-- Інші пункти меню -->
        </ul>
    </nav>
    <div class="sidebar-footer">
        <p>&copy; 2025 SmartHome Inc.</p>
    </div>
</aside>
```

Наступним етапом є створення основної частини панелі. У лістингу 3.3 наведено розмітку розділу «Головна панель». Даний розділ включає заголовок та

кнопку очищення повідомлень, а також панелі з повідомленнями, статистикою та елементами швидкого доступу.

Лістинг 3.3 – Розмітка розділу "Головна панель"

```
<section id="dashboard" class="content-section active">
    <header class="section-header">
        <h2>Головна панель</h2>
        <button id="clear-notifications" class="btn btn-
secondary">Очистити все</button>
    </header>
    <div class="notifications-panel">...</div>
    <div class="stats-grid">...</div>
    <div class="quick-access">...</div>
</section>
```

У лістингу 3.4 представлено HTML-структуру розділу «Кімнати». Вона включає заголовок, кнопку додавання нової кімнати, вкладки та контейнери для відображення відповідного контенту.

Лістинг 3.4 – Розмітка розділу «Кімнати»

```
<section id="rooms" class="content-section">
    <header class="section-header">
        <h2>Кімнати</h2>
        <button id="add-room-btn" class="btn btn-primary"><i
class="fas fa-plus"></i> Додати кімнату</button>
    </header>
    <div class="room-tabs" id="room-tabs-container">...</div>
    <div id="rooms-content-container">...</div>
</section>
```

Розділ «Пристрої» описано у лістингу 3.5. Він включає заголовок, кнопку додавання пристрою, фільтри за типом та кімнатою, а також контейнер для виведення списку пристроїв.

Лістинг 3.5 – Розмітка розділу «Пристрої»

```
<section id="devices" class="content-section">
    <header class="section-header">
        <h2>Усі пристрої</h2>
        <button id="add-new-device-btn" class="btn btn-primary"><i
class="fas fa-plus"></i> Додати новий пристрій</button>
    </header>
```

Продовження лістингу 3.5

```
    <div class="filters">
        <input type="text" placeholder="Пошук пристрою..."
class="form-control" id="device-search-all">
        <select id="device-type-filter-all" class="form-control">
            <option value="">Всі типи</option>
        </select>
        <select id="device-room-filter-all" class="form-control">
            <option value="">Всі кімнати</option>
        </select>
    </div>
    <div id="all-devices-list-container" class="devices-grid">
        <p>Завантаження пристроїв...</p>
    </div>
</section>
```

У лістингу 3.6 наведено структуру розділу «Автоматизація». Він містить заголовок, кнопку створення нового сценарію, а також приклад картки з описом сценарію, умовами виконання та кнопками керування.

Лістинг 3.6 – Розмітка розділу «Автоматизація»

```
<section id="automation" class="content-section">
    <header class="section-header">
        <h2>Автоматизація</h2>
        <button class="btn btn-primary" id="create-automation-
btn"><i class="fas fa-plus"></i> Створити сценарій</button>
    </header>
    <div class="automation-list" id="automation-list-container">
        <div class="automation-card">
            <h4>Вечірнє освітлення <span class="automation-status
active">Активний</span></h4>
            <p class="automation-details"><span>Тригер:</span> Час
- 18:00</p>
            <p class="automation-details"><span>Умова:</span>
Нікого немає вдома</p>
            <p class="automation-details"><span>Дія:</span>
Увімкнути світло у вітальні на 30%</p>
            <div class="automation-controls">
                <label class="switch">
                    <input type="checkbox" checked>
                    <span class="slider round"></span>
                </label>
                <button class="btn btn-edit btn-sm"><i class="fas
fa-edit"></i></button>
                <button class="btn btn-delete btn-sm"><i class="fas
fa-trash"></i></button>
            </div>
        </div></div></section>
```

Решта розділів, таких як аналітика, налаштування, акаунт користувача та підтримка, реалізуються за аналогічною структурною моделлю. Кожен із розділів має окрему HTML-секцію з унікальним ідентифікатором, яка містить заголовок, керуючі елементи та блоки відображення контенту, що динамічно заповнюються за допомогою JavaScript. Результат розробки клієнтського інтерфейсу наведено на рисунку 3.1.

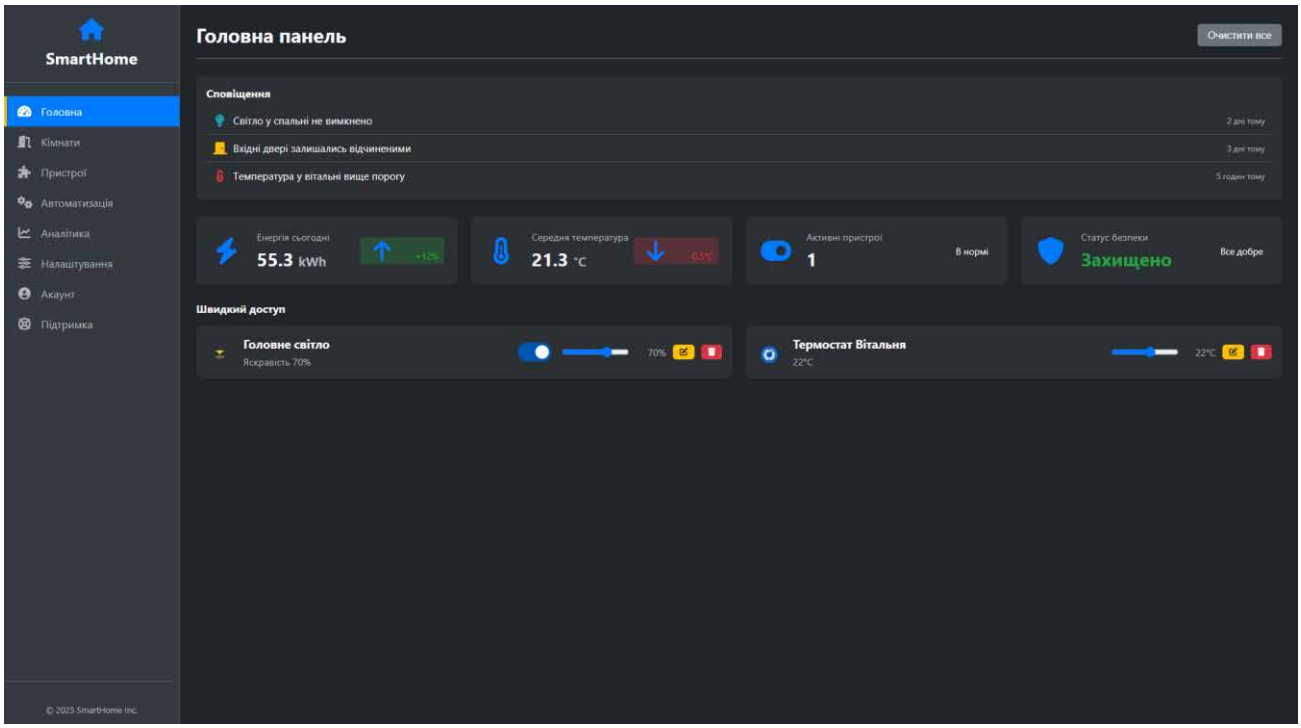


Рисунок 3.1 – Вигляд розробленого сторінки «Головна»

3.3 Реалізація тестових вузлів IoT для системи керування

3.3.1 Реалізація сенсорного вузла

Для побудови сенсорного вузла було використано плату ESP32 із підключеним датчиком температури та вологості DHT22. Для забезпечення мережевої взаємодії застосовуються бібліотеки WiFi, PubSubClient та ESPmDNS, які відповідають за підключення до Wi-Fi мережі, роботу з MQTT-протоколом і локальне іменування пристрою відповідно.

Параметри підключення до мережі Wi-Fi та MQTT-брокера задаються константами, приклад яких наведено у лістингу 3.7. З міркувань безпеки значення SSID та пароля замінено зірочками.

Лістинг 3.7 – Константи для підключення до Wi-Fi мережі та MQTT

```
const char* ssid = "*******";
const char* password = "***********";
const char* mqttServer = "DESKTOP-9PGCVTM";
const int mqttPort = 1883;
const char* mqttTopic = "sensors/dht22";
```

Для встановлення з'єднання з мережею та налаштування MQTT-клієнта в функції setup() використовується код, представлений у лістингу 3.8. Спершу здійснюється підключення до Wi-Fi, потім ініціалізується служба mDNS для розпізнавання брокера за іменем хоста, і нарешті встановлюється сервер MQTT. Для ідентифікації клієнта генерується унікальний ідентифікатор на основі MAC-адреси пристрою.

Лістинг 3.8 – Підключення до Wi-Fi та MQTT-брокера в функції setup()

```
WiFiClient espClient;
PubSubClient client(espClient);
void setup() {
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(1000);
    }
    if (!MDNS.begin("esp32")) {
        while (1) { delay(1000); }
    }
    IPAddress brokerIP = MDNS.queryHost(mqttServer);
    if (brokerIP == IPAddress(0, 0, 0, 0)) {
        while (1) { delay(1000); }
    }
    client.setServer(brokerIP, mqttPort);
    String clientId = "ESP32Sensor-" + String(WiFi.macAddress());
    client.connect(clientId.c_str());
}
```

Передача даних сенсора у систему здійснюється у циклі loop(). При цьому кожні 5 секунд виконується зчитування температури з датчика, формування JSON-повідомлення та публікація його в MQTT-топік. Формат повідомлення

містить ідентифікатор пристрою, поточну температуру та часову мітку у вигляді рядка. Фрагмент коду, що відповідає за формування повідомлення і публікацію, наведено в лістингу 3.9.

Лістинг 3.9 – Формування та публікація JSON-повідомлення в MQTT-топік

```
StaticJsonDocument<128> doc;
doc["device_id"] = WiFi.macAddress();
doc["temperature"] = temperature;
doc["timestamp"] = timeStr;
char payload[128];
serializeJson(doc, payload);
client.publish(mqttTopic, payload);
```

Для перевірки коректності публікації повідомлень у топік sensors/dht22 було використано програму MQTT Explorer. Цей інструмент дозволяє візуалізувати повідомлення, що надходять від сенсорного вузла, у зручному форматі JSON. Вигляд повідомлень у зазначеному топіку, зафіксований у MQTT Explorer, наведено на рисунку 3.2.

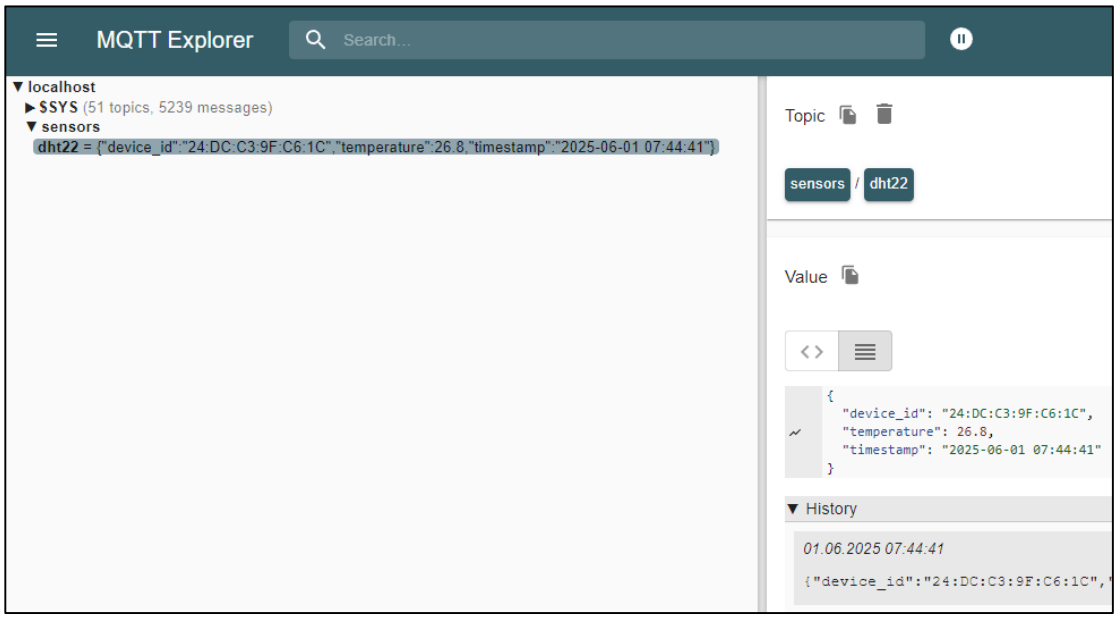


Рисунок 3.2 – Публікація повідомлень з даними температури

Для демонстрації фізичного вигляду сенсорного вузла та підтвердження його працездатності на макетній платі наведено фото пристрою, розміщене на

рисунку 3.3. Обидва елементи – перевірка повідомлень через MQTT Explorer та збірка пристрою на макетній платі – розроблені для подальшого тестування UI-застосунку, що реалізує керування компонентами розумного дому.

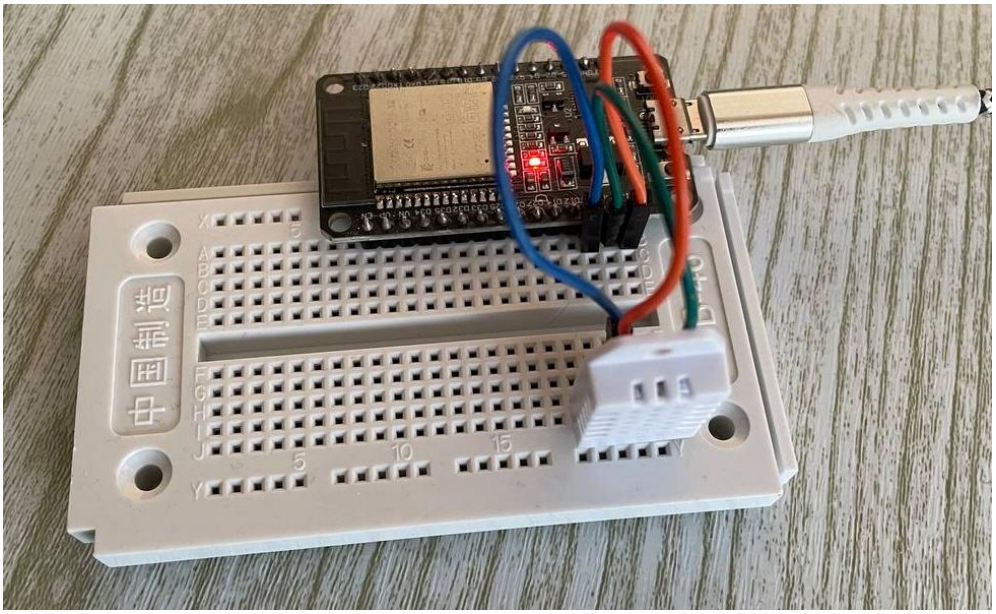


Рисунок 3.3 – Зовнішній вигляд тестової платформи сенсора

Повний код прошивки цього вузла наведено у лістингу Б.1.

3.3.2 Реалізація виконавчого вузла

Для реалізації виконавчого вузла використано іншу плату ESP32, на якій підключено реле, кероване через цифровий вивід 22. Для забезпечення мережевої взаємодії застосовуються ті ж бібліотеки, що й у сенсорному вузлі: WiFi, PubSubClient та ESPmDNS. Вони відповідають за підключення до Wi-Fi мережі, роботу з MQTT-протоколом і локальне іменування пристрою відповідно. Логіка підключення до мережі Wi-Fi та MQTT-брокера є подібною до реалізації сенсорного вузла, з відмінністю у використанні іншого MQTT-топіка для отримання команд. При цьому параметри підключення задаються константами, схожими на наведені у лістингу 3.7, але значення топіка відрізняється.

Після встановлення з'єднання у функції setup() виконується публікація реєстраційного повідомлення, яке містить MAC-адресу пристрою. Це

повідомлення надсилається у відповідний MQTT-топік для ідентифікації пристрою сервером та іншими вузлами мережі. Фрагмент коду публікації реєстраційного повідомлення наведено на лістингу 3.10.

Лістинг 3.10 – Реєстрація виконавчого пристрою в мережі

```
StaticJsonDocument<128> doc;
doc["device_id"] = WiFi.macAddress();
char payload[128];
serializeJson(doc, payload);
client.publish(mqttTopic, payload);
```

Для обробки вхідних MQTT-повідомлень використовується функція callback. Вона відповідає за розбір JSON-повідомлення, перевірку наявності ідентифікатора пристрою та поля command. Обробка виконується лише у разі співпадіння device_id з MAC-адресою пристрою. Залежно від значення поля command виконується керування реле: команда «on» вмикає реле, а «off» – вимикає. Визначення функції callback наведено на лістингу 3.11.

Лістинг 3.11 – Визначення функції callback

```
void callback(char* topic, byte* payload, unsigned int length) {
    StaticJsonDocument<256> doc;
    DeserializationError error = deserializeJson(doc, payload,
length);
    String ownMac = WiFi.macAddress();
    const char* deviceId = doc["device_id"];
    if (String(deviceId) != ownMac) {
        return;
    }
    const char* command = doc["command"];
    if (strcmp(command, "on") == 0) {
        digitalWrite(22, HIGH);
    } else if (strcmp(command, "off") == 0) {
        digitalWrite(22, LOW);
    } else {
        Serial.println("Невідома команда");
    }
}
```

Таким чином, виконавчий вузол приймає команди від сервера через MQTT, фільтрує їх за унікальним ідентифікатором пристрою і здійснює керування реле,

підключеним до виводу 22. Повний код прошивки цього вузла наведено у лістингу Б.2.

Для підтвердження працездатності виконавчого вузла та аналізу мережевої взаємодії використано MQTT Explorer. Візуалізація повідомлень у відповідному топіку представлена на рисунку 3.4.

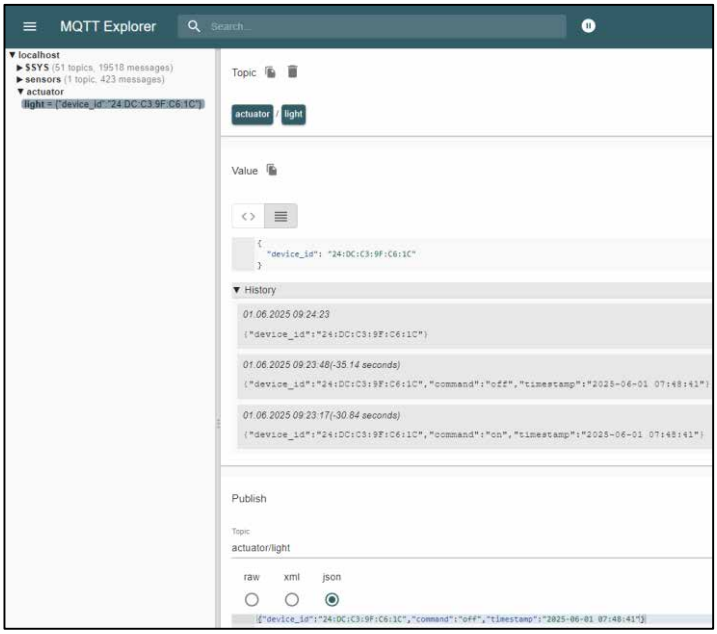


Рисунок 3.4 – Отримання команд у топіку виконавчого вузла

Фізичний вигляд тестового стенда виконавчого вузла з підключеним реле та платою ESP32 наведено на рисунку 3.5. Схеми підключень обох пристроїв наведено у додатку А.

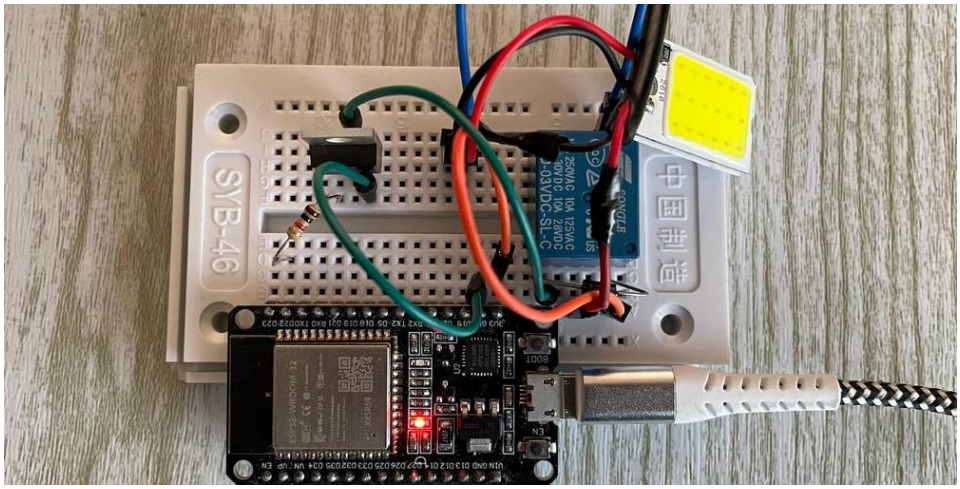


Рисунок 3.5 – Зовнішній вигляд тестової платформи виконавчого вузла

РОЗДІЛ 4. ТЕСТУВАННЯ ТА ВПРОВАДЖЕННЯ

4.1 Методика тестування

У рамках тестування програмного забезпечення застосовано підхід ad-hoc тестування [9], що передбачає перевірку функціонування системи без попередньо детально прописаних сценаріїв чи тест-кейсів. Тестування здійснювалося інтуїтивно, з фокусом на виявлення можливих візуальних і функціональних помилок у типових сценаріях взаємодії. Для фіксації результатів перевірок та подальшого звітування використовувалась система TestRail [10], у якій створено базові тест-кейси та зафіксовано статус їх виконання в межах відповідного тест-рану. Основна мета полягала у перевірці коректності взаємодії між інтерфейсом та апаратними компонентами через MQTT-протокол, а також у верифікації відображення елементів інтерфейсу.

Для тестування сформовано сім тест-кейсів, що подано у табличному вигляді (таблиця 4.1). Перші чотири кейси зосереджені на перевірці відображення різних сторінок інтерфейсу, п’ятий і шостий – на взаємодії користувача із виконавчим пристроєм через UI, а сьомий – на перевірці адаптивності інтерфейсу під різні розміри екранів.

Таблиця 4.1 – Перелік тест-кейсів

№	Назва тест-кейсу	Опис
1	Відображення головної сторінки	Перевірка коректного завантаження головної сторінки інтерфейсу
2	Відображення сторінки пристроїв	Перевірка коректного відображення списку підключених пристроїв
3	Відображення сторінки автоматизації	Перевірка роботи інтерфейсу для налаштування автоматичних дій
4	Відображення сторінки підтримки	Перевірка коректного завантаження сторінки підтримки користувача

Продовження таблиці 4.1

5-6	Включення/виключення виконавчого пристрою через UI	Перевірка подачі команди ввімкнення/вимкнення, зміни стану пристрою та кнопки
7	Перевірка адаптивності інтерфейсу	Оцінка коректності відображення UI на екранах різного розміру (ПК, планшет, смартфон)

Після визначення тест-кейсів у TestRail було створено тест-ран, в якому ці кейси зібрані для подальшого виконання. Результати тестування будуть фіксуватися у відповідних полях тест-рану. На момент підготовки документації тест-ран сформовано, але виконання тестів ще не завершено (рисунок 4.1).

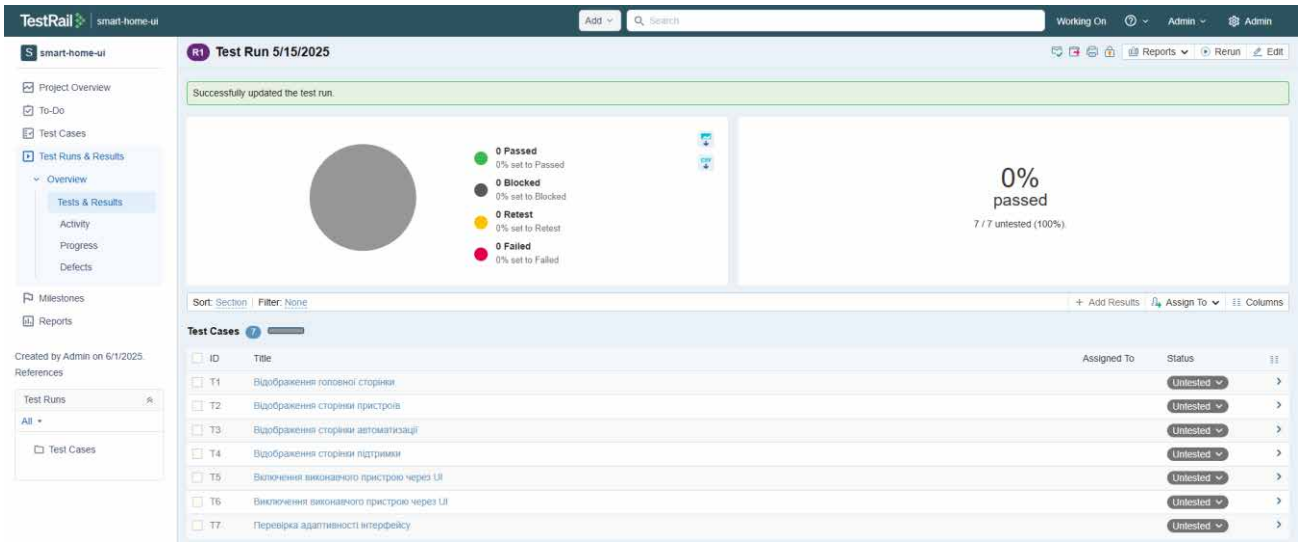


Рисунок 4.1 – Інтерфейс TestRail із створеним тест-раном

4.2 Проведення тестування та аналіз результатів

Проведення тестування виконувалося шляхом послідовного проходження тест-рану, створеного у системі TestRail. Кожен тест-кейс був виконаний у відповідності до описаних сценаріїв, результати фіксувалися у вигляді скріншотів інтерфейсу та спостережень за роботою виконавчих пристроїв. Усі тест-кейси пройдені успішно, що підтверджує відповідність розробленої системи функціональним вимогам.

Перший тест-кейс передбачав перевірку коректного відображення головної сторінки. На рисунку 4.2 наведено інтерфейс із елементами «Швидкий доступ», головною панеллю та відображенням середньої температури. Відображення відповідає технічним вимогам.

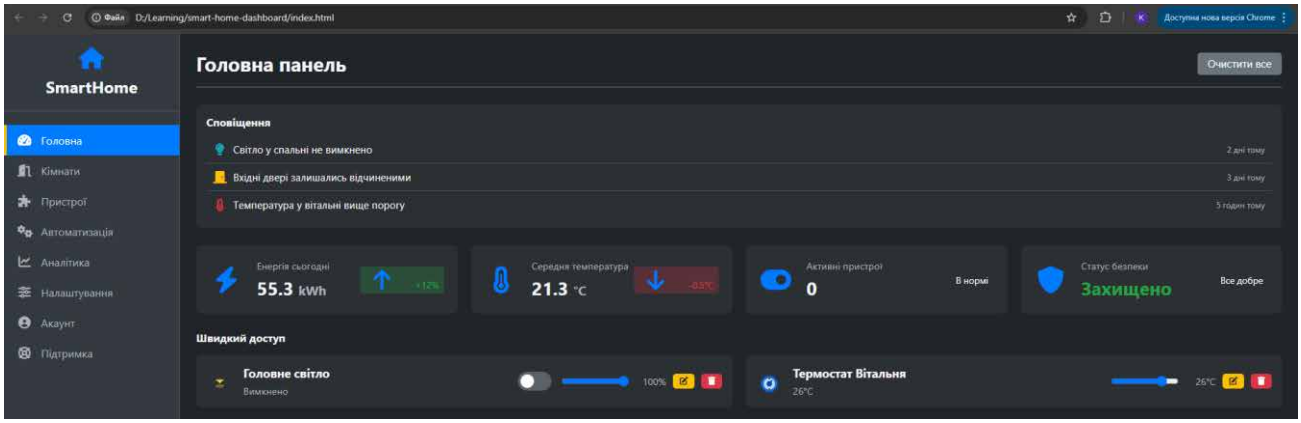


Рисунок 4.2 – Відображення головної сторінки застосунку

Другий тест-кейс стосувався сторінки пристроїв. Як показано на рисунку 4.3, у списку відображено термостат, що емулюється датчиком температури, два освітлювальних пристрої та розумний телевізор. Стан усіх пристроїв – виключений.

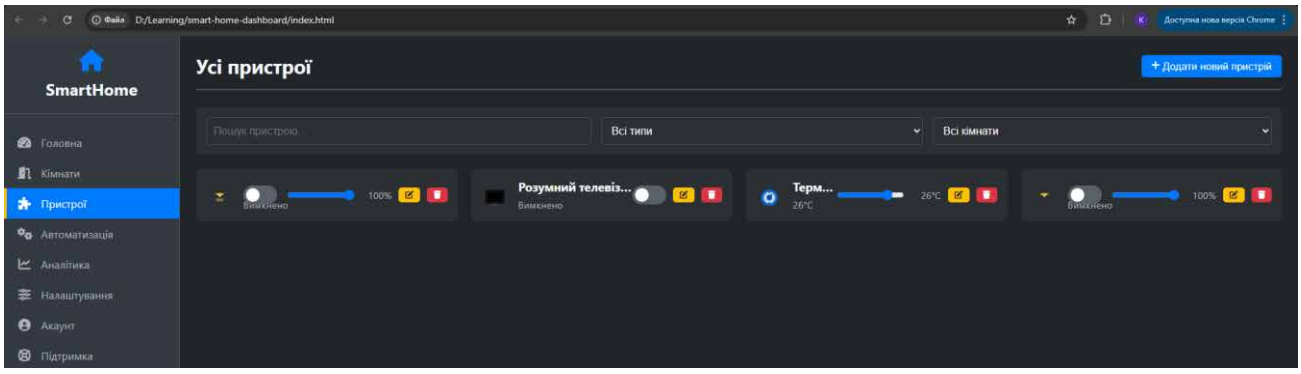


Рисунок 4.3 – Відображення сторінки пристроїв зі списком доступних компонентів

Третій тест-кейс охоплював сторінку автоматизації. На рисунку 4.4 наведено приклад налаштованої автоматизації «Вечірнє освітлення» з тригером за часом (18:00) та дією увімкнення світла у вітальні.

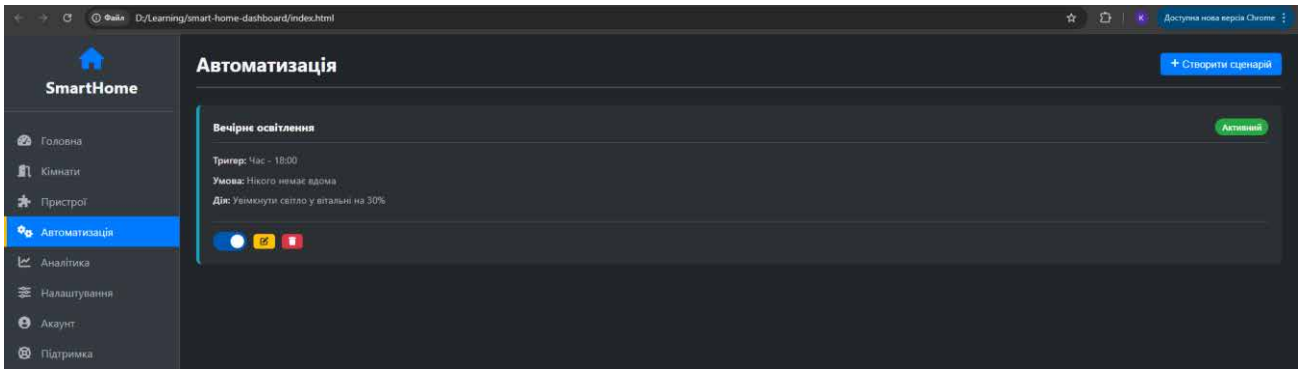


Рисунок 4.4 – Відображення сторінки автоматизації з налаштуванням сценарію

Четвертий тест-кейс перевіряв сторінку підтримки користувача, що містить форму зворотного зв’язку, як це показано на рисунку 4.5.

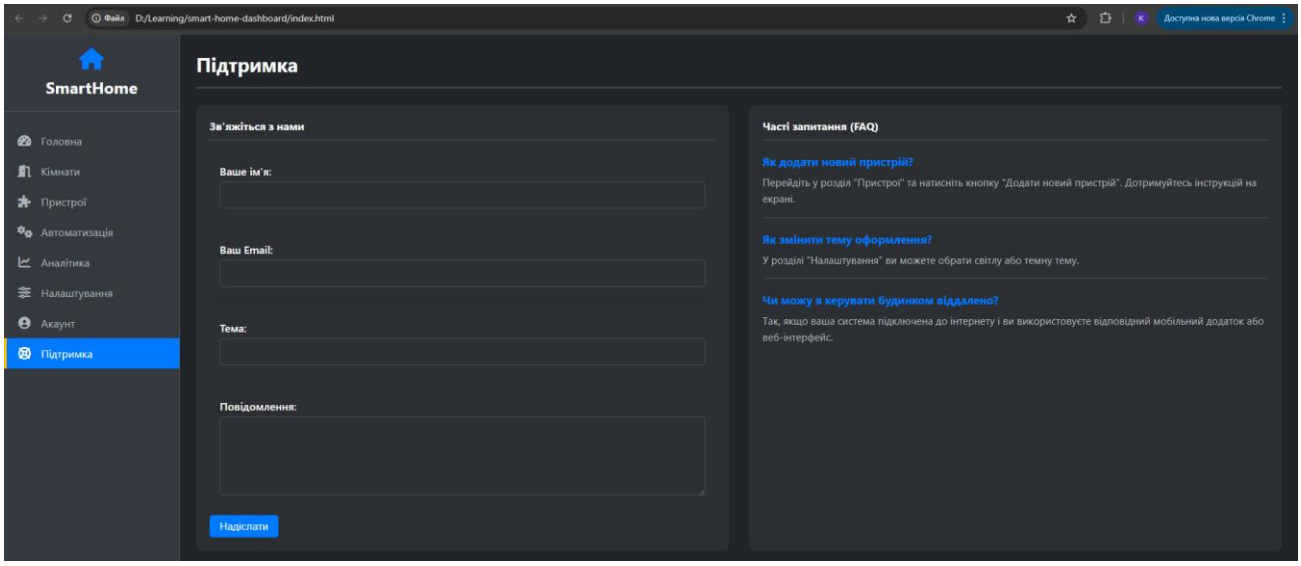


Рисунок 4.5 – Відображення сторінки підтримки користувача з формою зворотного зв’язку

П’ятий тест-кейс передбачав ввімкнення виконавчого пристрою через UI. На рисунку 4.6 зафіксовано зміну стану кнопки управління. Фактичне включення пристрою підтверджено на рисунку 4.7, де відображено стан освітлення на тестовій платі. А на рисунку 4.8 – відповідне повідомлення в MQTT-брокері.

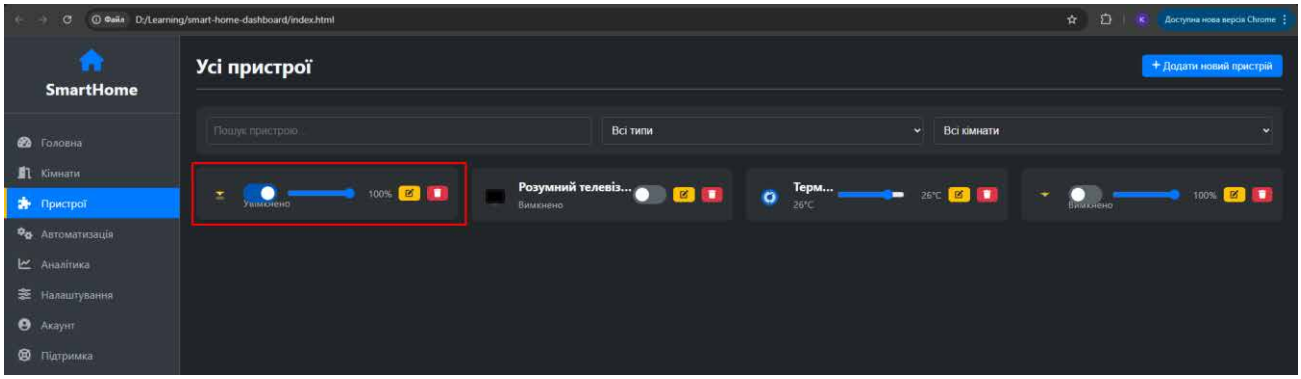


Рисунок 4.6 – Інтерфейс після увімкнення виконавчого пристрою через UI

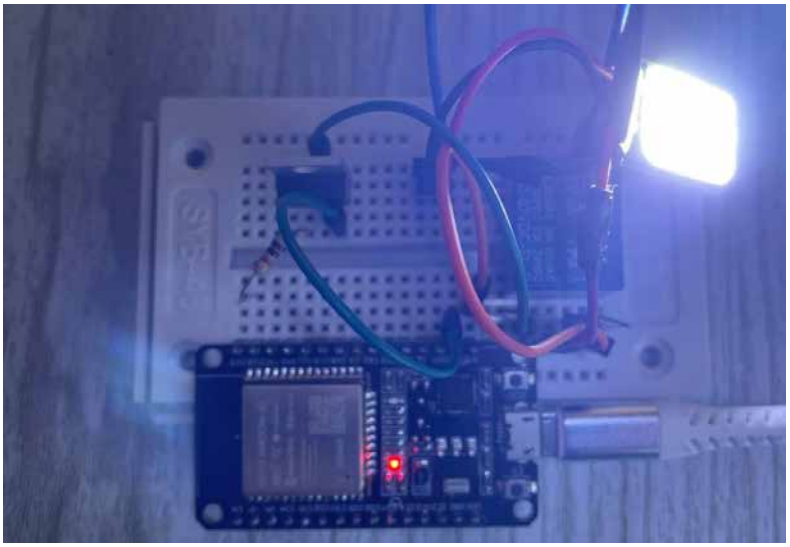


Рисунок 4.7 – Фізичний стан виконавчого пристрою після увімкнення

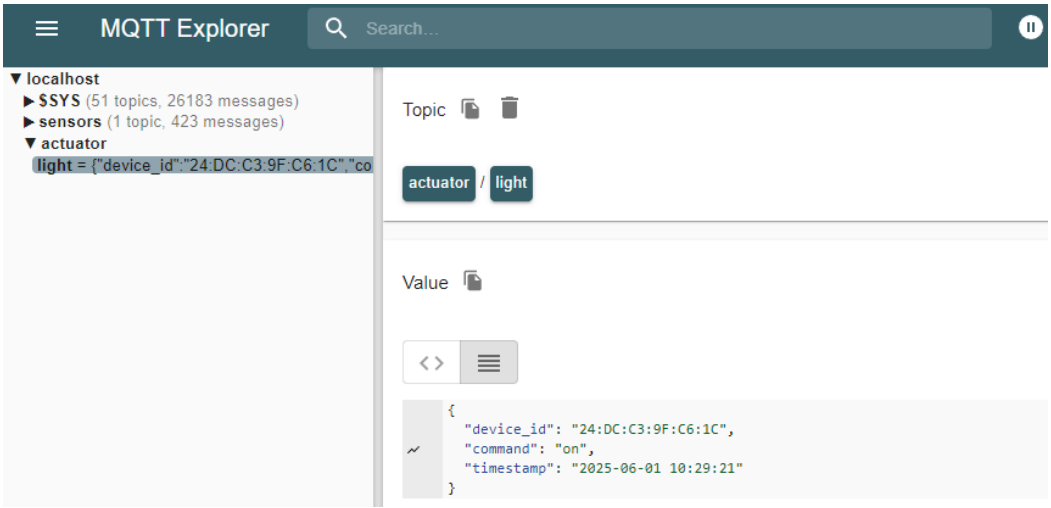


Рисунок 4.8 – Повідомлення на MQTT-брокері для включення пристрою

Шостий тест-кейс стосувався виключення пристрою через UI. Як видно на стан кнопки було змінено відповідно до команди виключення. Візуальне

відображення виключеного пристрою у інтерфейсі повторює стан із рисунку 4.3 і тому додатково не наводиться.

Сьомий тест-кейс стосується адаптивності інтерфейсу під мобільні екрани. За допомогою інструментів розробника браузера [8] проведено емуляцію різних розмірів екранів. Результат наведено на рисунку 4.9, де показано зміну інтерфейсу меню до більш компактного виду, що відповідає вимогам адаптивності.

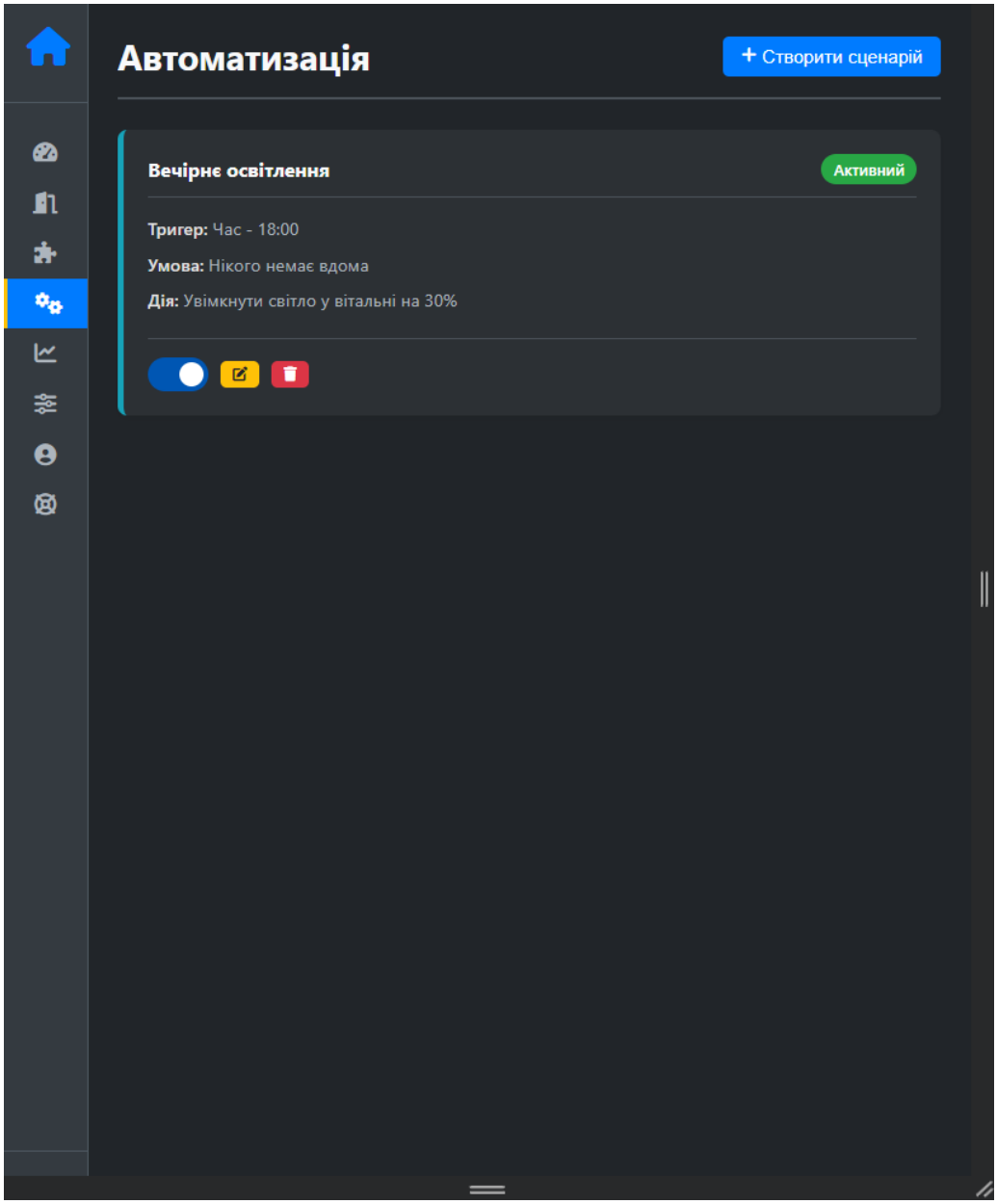


Рисунок 4.9 – Адаптивний вигляд застосунку на мобільному пристрої

На рисунку 4.10 наведено знімок екрану тест-рану після завершення проходження всіх тест-кейсів. Як видно, усі випадки виконання були успішними, що підтверджує відповідність функціоналу заявленим вимогам та відсутність помилок у перевірених сценаріях.

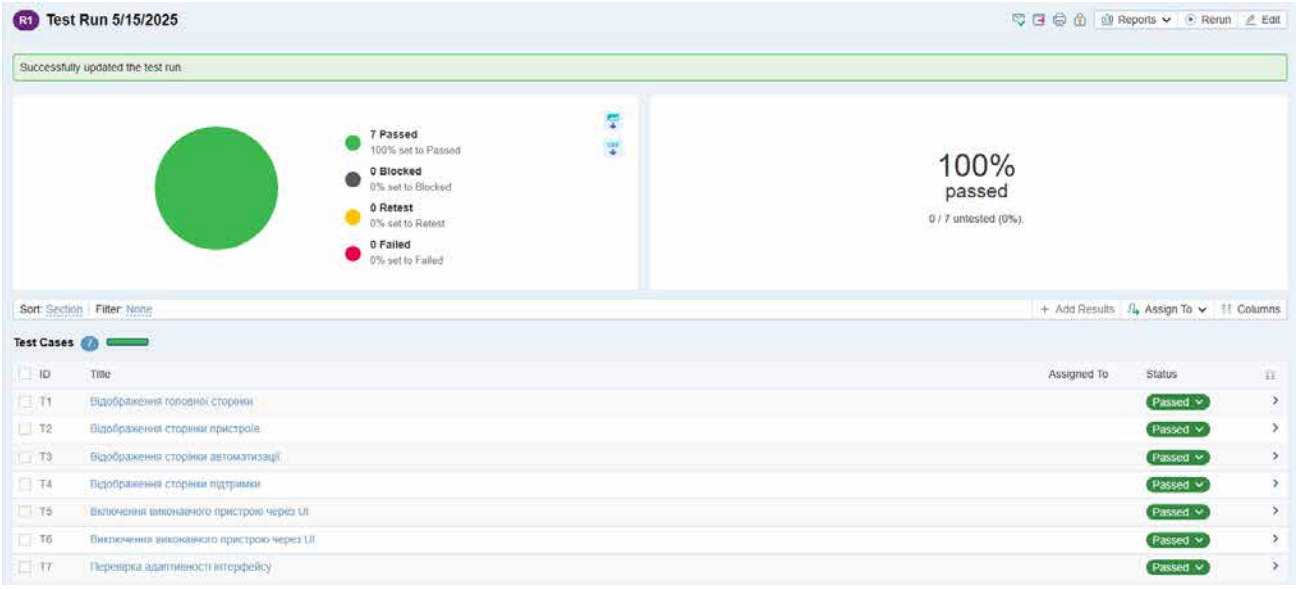


Рисунок 4.10 – Тест-ран у TestRail після проходження тест-кейсів

Проведений аналіз результатів тестування свідчить про відповідність функціональних можливостей системи визначеним технічним вимогам. Послідовне проходження тест-рану дозволило підтвердити коректність відображення основних інтерфейсних сторінок, що забезпечує користувачам доступ до всіх необхідних функцій. Результати тестування взаємодії з виконавчими пристроями продемонстрували стабільність роботи механізмів керування через UI та коректну реакцію апаратної частини, що підтверджено як візуально на платі, так і повідомленнями в MQTT-брокері.

Перевірка адаптивності інтерфейсу показала відповідність вимогам до роботи на мобільних пристроях, що забезпечує можливість експлуатації системи на різних платформах з різними розмірами екранів. Відсутність помилок під час проходження всіх тест-кейсів дає підстави вважати, що система готова до подальшого впровадження та використання у реальному середовищі.

4.3 Рекомендації до впровадження

Розглянемо мінімальні апаратні вимоги для впровадження розробленої системи. Вони включають основні параметри, необхідні для стабільної роботи програмного забезпечення та забезпечення комунікації з виконавчими пристроями.

- Процесор: мінімум двоядерний із тактовою частотою не менше 1,5 ГГц.
- Оперативна пам’ять (ОЗП): не менше 2 ГБ.
- Накопичувач: SSD або інший швидкодіючий накопичувач із вільним простором не менше 16 ГБ.
- Мережевий інтерфейс: підтримка Ethernet або Wi-Fi стандарту не нижче 802.11n.

Для варіантів масштабування системи на понад 10 пристроїв рекомендується використовувати контролери з оперативною пам’яттю не менше 4 ГБ, накопичувачі об’ємом від 32 ГБ та наявність розподіленої мережевої інфраструктури із резервуванням зв’язку.

Обґрунтування вибору компонентів базується на принципах модульності та відкритості. Використання стандартних протоколів, таких як MQTT, забезпечує сумісність із широким спектром пристроїв і полегшує інтеграцію нових модулів без необхідності значних змін у архітектурі. Модульний підхід у розробці програмного забезпечення дозволяє замінювати або оновлювати окремі компоненти системи без впливу на загальну функціональність. Відкриті апаратні платформи та бібліотеки надають можливість адаптувати систему під специфічні вимоги користувача.

Таким чином, запропоновані рекомендації забезпечують умови для надійного функціонування системи в межах мінімальних вимог, дозволяють масштабувати рішення відповідно до зростання кількості пристроїв і враховують принципи гнучкості та розширюваності при виборі компонентів.

ВИСНОВКИ

У результаті виконання дипломної роботи було розглянуто процес розробки системи керування IoT-пристроями з акцентом на функціональність, сумісність та адаптивність інтерфейсу для різних типів пристроїв. Проведено аналіз існуючих IoT-платформ і технічних засобів, що дозволило сформувати технічні вимоги до системи та визначити основні компоненти архітектури. Розроблено функціональну модель взаємодії між клієнтською частиною, сервером і апаратними модулями з використанням протоколу MQTT як засобу передачі повідомлень за схемою «видавець-підписник».

Користувацький інтерфейс реалізовано за допомогою HTML, CSS та JavaScript з підтримкою адаптивності для мобільних та десктопних пристроїв, що забезпечує уніфікований доступ до функцій керування. Серверна частина виконана на основі Express.js, а апаратна складова базується на ESP8266 та ESP32. Впровадження MQTT-брокера Mosquitto забезпечує надійну комунікацію між компонентами системи. Програмне забезпечення пройшло функціональне тестування із застосуванням TestRail, що підтвердило відповідність системи технічним вимогам і коректність роботи основних сценаріїв.

Архітектура системи має модульну структуру, що сприяє масштабуванню, зокрема підтримці одночасного керування понад десятьма пристроями без суттєвих змін у конфігурації чи кодовій базі. Використання відкритих стандартів і протоколів створює передумови для подальшої комерціалізації розробки як платформи для домашньої або офісної автоматизації.

Перспективи розвитку системи полягають у впровадженні додаткових функціональних можливостей, зокрема інтеграції голосового управління для підвищення зручності користування та організації відеоспостереження на базі WebRTC-технологій. Подальше вдосконалення може включати розширення набору підтримуваних пристроїв, підвищення безпеки комунікацій та поглиблення інтеграції з іншими IoT-платформами для забезпечення більш комплексної автоматизації.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Браун Ф. Модель діаграми зв’язків сутностей. Guru99. URL: https://www.guru99.com/uk/er-diagram-tutorial-dbms.html (дата звернення: 19.05.2025).

2. Дідковська М. Проектування програмного забезпечення засобами UML: лекція. Київ, 2020. 10 с. URL: http://mmsa.kpi.ua/sites/default/files/disciplines/Розробка%20і%20тестування%20програм/didkovska_m_v_testing_lecture_6.pdf (дата звернення: 17.05.2025).

3. Ємельянова В. Автоматизована система керування мікрокліматом теплиці: магістерська робота. Суми, 2024. 45 с. URL: https://essuir.sumdu.edu.ua/bitstream-download/123456789/97949/1/Yemelianova_mag_rob.pdf (дата звернення: 28.05.2025).

4. Заводовська Є. А. Система для збору та аналізу даних від розумних девайсів: магістерська робота. Київ, 2024. 109 с. URL: https://ela.kpi.ua/server/api/core/bitstreams/86929666-c5ab-4db9-b8ec-098e5f953087/content (дата звернення: 23.05.2025).

5. Застосунок Samsung SmartThings для зручного керування розумним домом | Samsung Україна. Samsung ua. URL: https://www.samsung.com/ua/smartthings/app/ (дата звернення: 26.05.2025).

6. Зосім М. Варіанти використання та сценарії. Maxym Zosym. URL: https://www.maxzosim.com/use-cases-and-scenarios/ (дата звернення: 31.05.2025).

7. Кучеренко Ю. Перспективи розвитку мережевих технологій в автоматизованих системах обліку енергоресурсів. Ефективність та автоматизація інженерних рішень у приладобудуванні : XVII Всеукр. науково-практ. конф. студентів, аспірантів та молодих вчен., м. Київ, 7–8 груд. 2021 р. 2021. С. 151–154. URL: https://ela.kpi.ua/server/api/core/bitstreams/0fc9d2ea-92e5-48aa-9989-528299896232/content (дата звернення: 20.05.2025).

8. Мисак Д. Chrome DevTools: налаштування, можливості та способи перевірки коду. DOU. URL: https://dou.ua/lenta/articles/chrome-dev-tools-guide/ (дата звернення: 30.05.2025).

9. Різниця між ad-hoc та дослідницьким тестуванням. QATestLab. URL: https://training.qatestlab.com/blog/technical-articles/what-is-ad-hoc-testing/ (дата звернення: 23.05.2025).

10. Робота з системою Testrail. QATestLab. URL: https://training.qatestlab.com/blog/technical-articles/work-with-testrail-system/ (дата звернення: 18.05.2025).

11. Свердлюк Б. Налаштовуємо мінімалістичний інтерфейс Home Assistant. DOU. URL: https://dou.ua/forums/topic/43592/ (дата звернення: 31.05.2025).

12. ATmega328P DATASHEET. Microchip. URL: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf (дата звернення: 25.05.2025).

13. Custom app tools. Tuya Developer. URL: https://developer.tuya.com/en/docs/iot/custom-build-tools?id=Kboyzonqdqecn (дата звернення: 29.05.2025).

14. ESP32-DevKitM. Technical Documents | Espressif Systems. URL: https://docs.espressif.com/projects/esp-idf/en/v4.3/esp32/hw-reference/esp32/user-guide-devkitm-1.html (дата звернення: 01.06.2025).

15. ESP-WROOM-32 datasheet. ALLDATASHEET.COM. URL: https://www.alldatasheet.com/datasheet-pdf/pdf/1179101/ESPRESSIF/ESP-WROOM-32.html (дата звернення: 30.05.2025).

16. Express. Node.js web application framework. URL: https://expressjs.com/ (дата звернення: 20.05.2025).

17. FAQ. MQTT – The Standard for IoT Messaging. URL: https://mqtt.org/faq/ (дата звернення: 27.05.2025).

18. Internet of things (iot): architecture, applications, and security challenges / R. Kenaza et al. 2022 4th international conference on pattern analysis and intelligent

systems (PAIS), Oum El Bouaghi, Algeria, 12–13 October 2022. 2022. URL: https://doi.org/10.1109/pais56586.2022.9946918 (дата звернення: 18.05.2025).

19. PubSubClient. Docs Arduino. URL: https://docs.arduino.cc/libraries/pubsubclient/ (дата звернення: 22.05.2025).

20. Raspberry Pi Ltd. Raspberry Pi 4 Model B. Release 1.1. Raspberry Pi Datasheets. URL: https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-datasheet.pdf (дата звернення: 24.05.2025).

21. The working group setting the standards for wireless LANs. IEEE802. URL: https://www.ieee802.org/11/ (дата звернення: 21.05.2025).

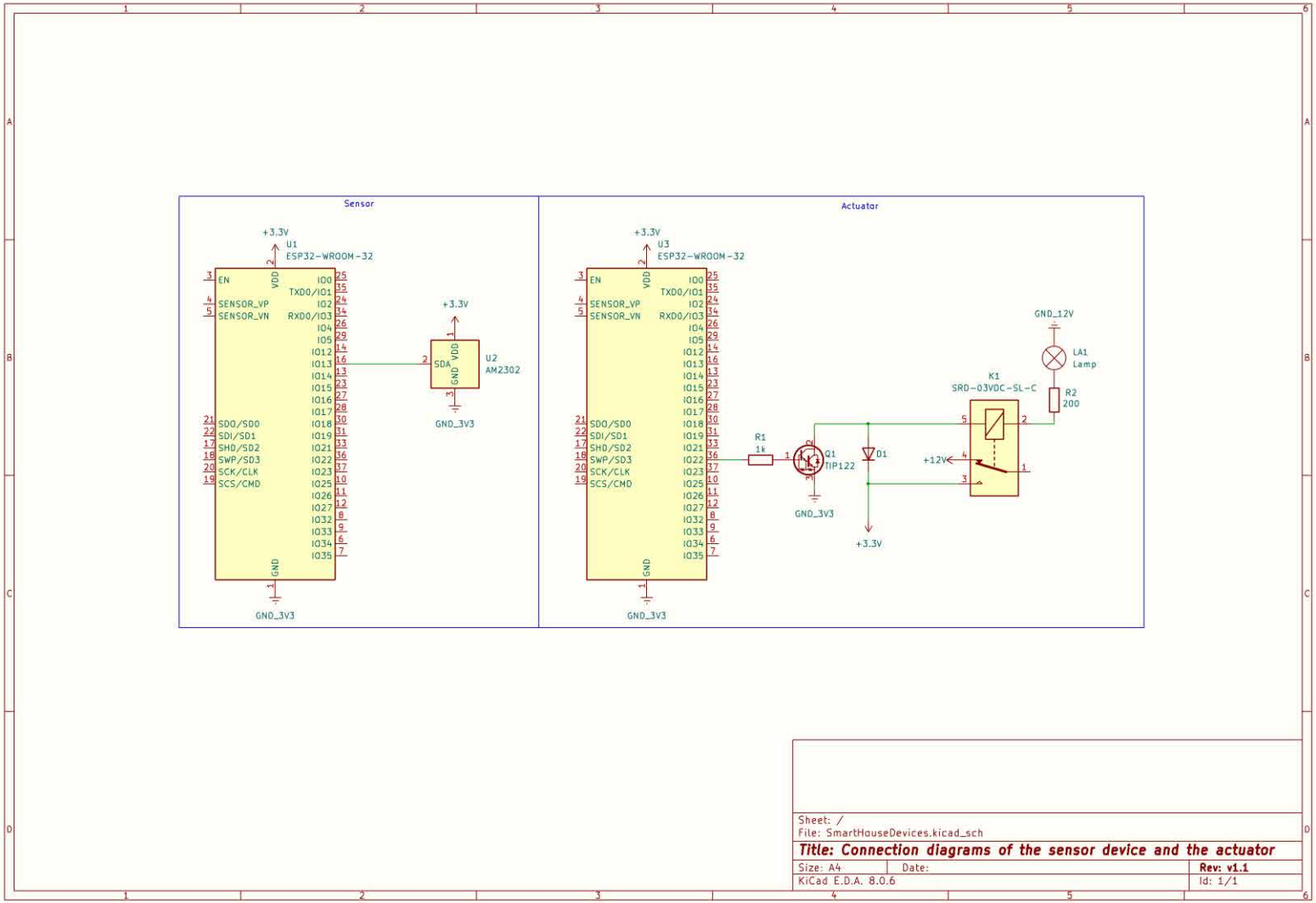
22. WebUI. Redirecting to Tasmota Documentation. URL: https://tasmota.github.io/docs/WebUI/ (дата звернення: 19.05.2025).

23. Y.2060: overview of the internet of things. ITU. URL: https://www.itu.int/rec/t-rec-y.2060-201206-i (дата звернення: 22.05.2025).

ДОДАТКИ

Додаток А

Схеми з’єднання апаратних модулів



Додаток Б

Лістинг конфігурації прошивки

Лістинг Б.1 – Повний код прошивки сенсора

```
#include <WiFi.h>
#include <PubSubClient.h>
#include <ESPmDNS.h>
#include <DHT.h>
#include <ArduinoJson.h>
#include "time.h"

#define DHTPIN 13
#define DHTTYPE DHT22

const char* ssid = "********";
const char* password = "********";

const char* mqttServer = "DESKTOP-9PGCVTM";
const int mqttPort = 1883;
const char* mqttTopic = "sensors/dht22";

const char* ntpServer = "pool.ntp.org";
const long gmtOffset_sec = 3 * 3600;
const int daylightOffset_sec = 0;

WiFiClient espClient;
PubSubClient client(espClient);
DHT dht(DHTPIN, DHTTYPE);

unsigned long lastPublishTime = 0;
const unsigned long publishInterval = 5000;

void reconnect() {
    while (!client.connected()) {
        Serial.print("Підключення до MQTT...");
        String clientId = "ESP32Sensor-" +
String(WiFi.macAddress());
        if (client.connect(clientId.c_str())) {
            Serial.println("Успішно!");
        } else {
            Serial.print("Помилка: ");
            Serial.println(client.state());
            delay(5000);
        }
    }
}

void setup() {
    Serial.begin(115200);
    dht.begin();
```

```
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(1000);
        Serial.print(".");
    }
    Serial.println("WiFi підключено!");
    Serial.print("IP-адреса ESP32: ");
    Serial.println(WiFi.localIP());

    if (!MDNS.begin("esp32")) {
        Serial.println("mDNS ініціалізація не вдалася!");
        while (1) { delay(1000); }
    }
    Serial.println("mDNS ініціалізовано!");

    IPAddress brokerIP = MDNS.queryHost(mqttServer);
    if (brokerIP == IPAddress(0, 0, 0, 0)) {
        Serial.println("Не вдалося знайти брокер за іменем
хоста!");
        while (1) { delay(1000); }
    }
    Serial.print("MQTT брокер IP: ");
    Serial.println(brokerIP);

    client.setServer(brokerIP, mqttPort);

    reconnect();

    configTime(gmtOffset_sec, daylightOffset_sec, ntpServer);
}

void loop() {
    if (!client.connected()) {
        reconnect();
    }
    client.loop();

    unsigned long currentMillis = millis();
    if (currentMillis - lastPublishTime >= publishInterval) {
        lastPublishTime = currentMillis;

        float temperature = dht.readTemperature();

        if (isnan(temperature)) {
            Serial.println("Помилка зчитування температури з
DHT22");
            return;
        }

        struct tm timeinfo;
        if (!getLocalTime(&timeinfo)) {
            Serial.println("Не вдалося отримати час");
            return;
        }
```

```
        char timeStr[25];
        strftime(timeStr, sizeof(timeStr), "%Y-%m-%d %H:%M:%S",
&timeinfo);

        StaticJsonDocument<128> doc;
        doc["device_id"] = WiFi.macAddress();
        doc["temperature"] = temperature;
        doc["timestamp"] = timeStr;

        char payload[128];
        serializeJson(doc, payload);

        client.publish(mqttTopic, payload);
        Serial.println("Опубліковано:");
        Serial.println(payload);
    }
}
```

Лістинг Б.2 – Повний код прошивки виконавчого вузла

```
#include <WiFi.h>
#include <PubSubClient.h>
#include <ESPmDNS.h>
#include <ArduinoJson.h>

#define RELAY_PIN 22

const char* ssid = "********";
const char* password = "********";

const char* mqttServer = "DESKTOP-9PGCVTM";
const int mqttPort = 1883;
const char* mqttTopic = "actuator/light";

WiFiClient espClient;
PubSubClient client(espClient);

void callback(char* topic, byte* payload, unsigned int length) {
    StaticJsonDocument<256> doc;
    DeserializationError error = deserializeJson(doc, payload,
length);
    if (error) {
        Serial.println("Помилка розбору JSON");
        return;
    }

    if (!doc.containsKey("device_id")) {
        Serial.println("Відсутнє поле device_id");
        return;
    }

    String ownMac = WiFi.macAddress();
```

```
    const char* deviceId = doc["device_id"];

    if (String(deviceId) != ownMac) {
        return;
    }

    if (!doc.containsKey("command")) {
        return;
    }

    const char* command = doc["command"];

    if (strcmp(command, "on") == 0) {
        digitalWrite(RELAY_PIN, HIGH);
        Serial.println("Реле увімкнено");
    } else if (strcmp(command, "off") == 0) {
        digitalWrite(RELAY_PIN, LOW);
        Serial.println("Реле вимкнено");
    } else {
        Serial.println("Невідома команда");
    }
}

void reconnect() {
    while (!client.connected()) {
        Serial.print("Підключення до MQTT...");
        String clientId = "ESP32Actuator-" +
String(WiFi.macAddress());
        if (client.connect(clientId.c_str())) {
            Serial.println("Успішно");
            client.subscribe(mqttTopic);
        } else {
            Serial.print("Помилка: ");
            Serial.println(client.state());
            delay(5000);
        }
    }
}

void setup() {
    Serial.begin(115200);
    pinMode(RELAY_PIN, OUTPUT);
    digitalWrite(RELAY_PIN, LOW);

    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(1000);
        Serial.print(".");
    }
    Serial.println("WiFi підключено");
    Serial.print("IP-адреса ESP32: ");
    Serial.println(WiFi.localIP());

    if (!MDNS.begin("esp32-actuator")) {
```

```
        Serial.println("mDNS ініціалізація не вдалася");
        while (1) { delay(1000); }
    }
    Serial.println("mDNS ініціалізовано");

    IPAddress brokerIP = MDNS.queryHost(mqttServer);
    if (brokerIP == IPAddress(0, 0, 0, 0)) {
        Serial.println("Не вдалося знайти брокер за іменем хоста");
        while (1) { delay(1000); }
    }
    Serial.print("MQTT брокер IP: ");
    Serial.println(brokerIP);

    client.setServer(brokerIP, mqttPort);
    client.setCallback(callback);

    reconnect();

    StaticJsonDocument<128> doc;
    doc["device_id"] = WiFi.macAddress();
    char payload[128];
    serializeJson(doc, payload);
    client.publish(mqttTopic, payload);
}

void loop() {
    if (!client.connected()) {
        reconnect();
    }
    client.loop();
}
```