НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий інститут атомної та теплової енергетики

Кафедра цифрових технологій в енергетиці

«До захисту допущено»
Завідувач кафедри
_____ Наталія АУШЕВА
“___”_____2025 р.

Дипломна робота
на здобуття ступеня бакалавр

За освітньою програмою “Цифрові технології в енергетиці”

Спеціальності 122 “Комп’ютерні науки”

на тему: “Дистанційне керування пристроями IoT на базі wifi технології”

Виконав: студент 4 курсу, групи ТР-12
Руденко Владислав Ігорович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник професор каф. ЦТЕ, д.т.н., професор Сергій ОТРОХ _____
(посада, науковий ступінь, вчене звання, ім’я, ПРІЗВИЩЕ) (підпис)

Рецензент професор каф. ІПЗЕ, д.т.н., професор Олег БАРАБАШ _____
(посада, науковий ступінь, вчене звання, ім’я, ПРІЗВИЩЕ) (підпис)

Н.контроль доцент каф. ЦТЕ Артем ГУРІН _____
(посада, ім’я, ПРІЗВИЩЕ) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2025

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ

Кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

Рівень вищої освіти – перший (бакалаврський)
спеціальність 122 “Комп’ютерні науки”
Освітньо-професійна програма “Цифрові технології в енергетиці”

ЗАТВЕРДЖУЮ
Завідувач кафедри ЦТЕ
_____ Наталія АУШЕВА
(підпис)
“___” _____2025 р.

ЗАВДАННЯ
на дипломну роботу студенту

Руденку Владиславу Ігоровичу
(прізвище, ім’я, по батькові)

1. Тема роботи “Дистанційне керування пристроями IoT на базі wifi технології”
Науковий керівник Отрох Сергій Іванович, д.т.н.,професор
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від “ 02 ” Червня 2025 року № 1875-С

2. Термін подання студентом роботи 09.06.2025

3. Вихідні дані до роботи середовище розробки Visual Studio, фреймворк WPF, мова програмування C#, мова програмування C++, мікроконтролер ESP8266/ESP32, протокол MQTT, система контролю версій Git.

4. Перелік питань, які потрібно розробити

1) дослідити можливості використання апаратної платформи Arduino як основи для IoT-пристроїв з бездротовим з’єднанням;

2) проаналізувати сучасні дослідження у сфері IoT, бездротових мереж, протоколу MQTT та архітектурних моделей;

3) сформувати підхід до побудови архітектури програмної системи;

4) розробити модель даних і програмну архітектуру систем;

5) описати роботу користувача з системою, надати системні вимоги.

5. Орієнтований перелік ілюстративного матеріалу діаграму зв'язків бази даних, діаграму класів програмного забезпечення для Arduino, діаграму класів клієнтського застосунку, діаграму прецедентів, взаємодії користувача з системою, модель зібраної системи.

6. Дата видачі завдання 13.09.2024

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Вибір теми роботи	01.03.2025-10.04.2025	
2.	Аналіз методів та засобів розв'язання задачі	14.04.2025-20.04.2025	
3.	Розробка архітектури та загальної структури системи	21.04.2025-24.04.2025	
4.	Розробка окремих підсистем	27.04.2025-01.05.2025	
5.	Програмна реалізація системи	03.05.2025-10.05.2025	
6.	Оформлення пояснювальної записки	13.05.2025-14.05.2025	
7.	Захист програмного забезпечення	15.05.2025	
8.	Передзахист	28.05.2025	
9.	Захист	16.06.2025-21.06.2025	

Студент _____ Руденко В.І.
(підпис) (прізвище та ініціали)

Керівник _____ Отрох С.І.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дипломна робота виконана на 64 сторінках, містить 26 ілюстрацій, 3 таблиці, 2 додатки, 23 джерела в переліку посилань.

Мета роботи – розробити систему дистанційного керування IoT-пристроями на базі Wi-Fi технології з використанням протоколу MQTT, що забезпечує надійну двосторонню передачу даних і простий інтерфейс для користувача.

Методи та засоби: Visual Studio, WPF, C#, C++, ESP8266/ESP32, PlatformIO, MQTT, MQTTX, JSON, Microsoft SQL Server, Git.

Результат – розроблено клієнтський застосунок та програмне забезпечення для пристрою IoT, що забезпечують стабільну взаємодію через MQTT-протокол, відображення стану пристроїв у реальному часі, а також можливість їхнього налаштування, моніторингу й управління за допомогою зручного графічного інтерфейсу.

Ключові слова: WPF, Arduino, C#, C++, MQTT, IoT, MVVM, сенсорні пристрої.

ANNOTATION

The thesis is made on 64 pages, contains 26 illustrations, 3 tables, 2 appendices, 23 sources in the list of references.

Objective – develop a system for remote control of IoT devices based on Wi-Fi technology using the MQTT protocol, which provides reliable two-way data transmission and a simple user interface.

Methods and tools: Visual Studio, WPF, C#, C++, ESP8266/ESP32, PlatformIO, MQTT, MQTTX, JSON, Microsoft SQL Server, Git.

Result – a client application and software for the IoT device have been developed, providing stable interaction through the MQTT protocol, displaying the state of devices in real time, as well as the ability to configure, monitor and control them using a convenient graphical interface.

Keywords: WPF, Arduino, C #, C++, MQTT, IoT, MVVM, touch devices.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

IoT – Інтернет Речей (перекл. Internet of Things).

JSON – спеціалізований формат запису даних (перекл. JavaScript Object Notation).

MQTT – мережевий протокол (перекл. Message Queue Telemetry Transport).

QoS – рівень гарантій (перекл. Quality of Service)

Ардуіно – перекл. Arduino.

Wi-Fi – загальновживана назва передавання цифрових потоків даних.

IEEE – перекл. Institute of Electrical and Electronics Engineers

SSID – найменування бездротових мереж (перекл. Service Set Identifier).

DHCP – протокол прикладного рівня (перекл. Dynamic Host Configuration Protocol)

WPF – механізм візуалізації (перекл. Windows Presentation Foundation)

XAML – мова розмітки (перекл. Extensible Application Markup Language).

UI – інтерфейс користувача (перекл. User Interface)

MVVM – архітектурна модель (перекл. Model-ViewModel-Model).

ВСТУП

В умовах сучасного темпу розвитку цифрових технологій однією з тенденцій є інтеграція фізичних пристроїв в одну мережу, що реалізується за допомогою концепції IoT. Зростаюча популярність їх використання та значна потреба у дистанційному налагоджуванні таких пристроїв, моніторингу їхнього стану та обміну даними створює необхідність в розробці ефективної та гнучкої системи керування, що спроможна керувати ними віддалено [1].

Важливим аспектом такої системи є вибір протоколу спілкування IoT пристрою та клієнта, що спроможний забезпечити швидку та стабільну передачу даних. Ефективним вирішенням даної проблеми є використання протоколу MQTT, що спроможний мінімізувати трафік і реалізувати гнучкий обмін даними між компонентами системи.

Об’єктом дослідження в даній роботі є система дистанційного керування пристроями IoT на основі Wi-Fi технології. Предметом дослідження виступає архітектура комунікаційної взаємодії в межах мікроконтролера, MQTT-брокера та клієнтського застосунку.

Мета роботи полягає в розробці системи керування пристроями IoT та MQTT протоколом, яка забезпечує ефективний обмін даними, енергоефективність і надійність роботи. Досягнення мети роботи супроводить такі завдання: розробка IoT-пристрою, реалізація обміну даними через MQTT протокол, створення клієнтського застосунку та реалізація моніторингу та керування пристроєм в реальному часі.

У процесі дослідження застосовано методи моделювання програмно-апаратної взаємодії, проєктування архітектури застосунку, тестування та налагодження в реальному часі, а також методи об’єктивно-орієнтованого проєктування програмного забезпечення.

Практичною цінністю роботи є створення діючого прототипу системи дистанційного керування, що адаптований для вирішення прикладних завдань у сфері автоматизації.

Апробацію роботи було здійснено під час XXII Міжнародної науково-

практичної конференції молодих вчених і студентів “Сучасні проблеми наукового забезпечення енергетики” [8].

Дипломна робота містить 26 рисунків, 3 таблиці, 2 додатоки, перелік використаних джерел включає 23 найменування. Загальний обсяг роботи становить 64 сторінки. У першому розділі розглянуто загальні принципи побудови систем дистанційного керування пристроями IoT на базі Wi-Fi технології, сформульовано мету та завдання проєкту. У другому розділі проведено аналіз існуючих досліджень та прикладів реалізації подібних систем, визначено їхні переваги, недоліки та обґрунтовано доцільність створення власного рішення. Третій розділ присвячено обґрунтуванню вибору апаратних і програмних засобів розробки, включаючи клієнтське середовище, бібліотеки та протоколи обміну. У четвертому розділі детально описано програмну реалізацію системи: архітектуру застосунку, структуру моделей, передачу повідомлень через MQTT та взаємодію з апаратною частиною. У п’ятому розділі подано опис інтерфейсу клієнтського застосунку, сценаріїв використання з боку користувача, наведено технічні вимоги до середовища запуску та послідовність налаштування системи.

1 ДИСТАНЦІЙНЕ КЕРУВАННЯ ПРИСТРОЯМИ ІОТ НА БАЗІ WIFI ТЕХНОЛОГІЇ

З розвитком концепції Інтернету речей (IoT) зростає потреба у розробці ефективних систем, що здатні забезпечити налагодження та керування віддаленими пристроями в режимі реального часу. Необхідної актуальності набувають рішення, які передбачають обмін даними через мережу з мінімальними затримками, низьким енергоспоживанням та високою здібністю до масштабування [1]. Подібний підхід дозволяє підвищити ефективність управління технічними засобами та розширити функціональні можливості вже існуючих систем.

Реалізація має бути розрахована на підтримку сучасного протоколу передачі даних, що дозволить організувати надійну комунікацію і передавати команди керування та зворотній зв’язок. Система має передбачити можливість масштабування та інтеграції з іншими сервісами та доцільними апаратними рішеннями.

1.1 Платформа Arduino як пристрій Інтернету речей

Апаратно обчислювальна платформа Arduino (Ардуіно), зокрема плата ESP8266 [2], широко використовується для створення низькозатратних IoT-рішень завдяки поєднанню широкого функціонального спектру можливостей, доступності та простоти використання. На цій платі передбачено Wi-Fi модуль, що дозволяє створити інтернет з’єднання, підключаючи пристрій до мережі інтернет без потреби в додаткових апаратних засобах, дозволяючи забезпечити базову обчислювальну потужність, що задовольняє потреби користувача.

Однією з переваг платформи є енергоефективність, що дозволяє використовувати пристрої в автономному режимі або в умовах обмеженого енергоспоживання [3]. Це робить його зручним для застосування у мобільних,

побутових або енергетично обмежених системах, де важлива тривала робота без частого обслуговування.

Суттєвою перевагою є велика екосистема підтримки: платформа Arduino має відкриту архітектуру, широку документацію та активну спільноту розробників. Для мікроконтролера доступні сотні бібліотек, включно з тими, що забезпечують роботу з популярними сенсорами (температури, вологості, газу, відстані), модулями зв’язку, дисплеями. Наявність готових рішень значно спрощує інтеграцію додаткових компонентів до системи та зменшує час на розробку, що є особливо важливим у рамках обмежених проєктних термінів.

Окремо варто відзначити повну підтримку мережевих протоколів, зокрема TCP/IP і MQTT. Саме підтримка MQTT [4] є ключовою в контексті дистанційного керування IoT-пристроями, оскільки дозволяє реалізувати ефективний обмін повідомленнями між мікроконтролером і клієнтським застосунком. Arduino дозволяє не лише швидко створити робочий прототип, а й застосовувати його в реальних умовах для вирішення завдань моніторингу, керування та передачі даних у режимі реального часу.

1.2 Формулювання задачі розробки

У процесі використання пристроїв Інтернету речей (IoT) зростає необхідність забезпечення зручного, стабільного та надійного керування пристроями у реальному часі. Поширені рішення на ринку часто є складними в налаштуванні, або потребують глибокої технічної підготовки. За відсутності централізованого інтерфейсу керування користувач змушений вручну виконувати базові дії з налаштування мережі, тем підписки, параметрів пристроїв тощо, що ускладнює впровадження IoT-технологій у малих системах, навчальному процесі або побутовій автоматизації.

Актуальним є створення універсального засобу дистанційного керування IoT-пристроями, який забезпечує обмін повідомленнями за допомогою протоколу MQTT, має простий інтерфейс і підтримує базові функції взаємодії з

мікроконтролером. У контексті такого рішення важливо забезпечити відображення поточного стану пристрою, керування параметрами сенсорів, зміну MQTT-тем, додавання та видалення пристроїв із профілю користувача. При цьому не передбачається реалізація складної логіки автоматизації або сценаріїв – система надає користувачу ручні інструменти керування з мінімальними вимогами до налаштування та супроводу.

Розробка передбачає створення клієнтського застосунку для платформи Windows на основі WPF [5], реалізацію програмного забезпечення для IoT-пристрою, а також використання публічного MQTT-брокера для передачі даних. У результаті має бути отримано практичне рішення, яке можна використовувати у навчальних цілях, лабораторних умовах або для побутового застосування. Такий підхід сприятиме спрощенню взаємодії з IoT-пристроями, забезпечить стабільність з’єднання, централізований облік пристроїв і можливість їх ручного конфігурування без складної логіки.

2 АНАЛІЗ ІСНУЮЧИХ ДОСЛІДЖЕНЬ

У сучасній практиці розробці систем дистанційного керування пристроями IoT на базі Wi-Fi технології активно використовуються різні підходи, що основані на відкритих програмно-апаратних платформах та енергоефективних протоколах передачі даних. Популярними технічними рішеннями виступають використання мікроконтролерів з вбудованими Wi-Fi модулями, здатними забезпечити зв’язок без використання зовнішніх апаратних частин (ESP32, ESP8266 та ін.) [3]. Для забезпечення обміну даними використовують брокер-сервери (Mosquitto, HiveMQ, MQTTX) та клієнтських застосунків на базі веб- та десктопних застосунків.

Низка промислових компаній і установ зосереджують увагу на підвищені надійності, безпеки та мобільності таких систем. Попри значний прогрес, більшість існуючих систем залишаються фрагментованими, що ускладнює їхнє масштабування та інтеграцію в єдине середовище.

2.1 Бездротові мережі пристроїв IoT

У сучасних бездротових системах передавання даних важливою складовою є здатність пристроїв забезпечувати стабільне підключення до мережі для обміну інформацією з сервером або іншими вузлами. У випадку з мікроконтролерами, зокрема платформою Arduino, підключення до бездротових мереж найчастіше реалізується за допомогою вбудованих Wi-Fi-модулів. Ці модулі підтримують стек протоколів IEEE, що дає змогу інтегрувати пристрій у локальне мережеве середовище або інтернет. Надійність та швидкодія підключення напряму впливають на загальну функціональність системи, особливо в умовах динамічної мережевої інфраструктури [2].

Процес підключення до бездротової мережі розпочинається з ініціалізації апаратного інтерфейсу Wi-Fi-модуля. Мікроконтролер активує модуль для виконання сканування радіоефіру з метою виявлення доступних точок доступу. На апаратному рівні це означає перемикання радіочастотного блоку в режим

передачі, активацію апаратних механізмів модуляції сигналу і відправлення пакетів автентифікації [6]. Вбудований криптомодуль (як частина ESP32) або програмна реалізація в ESP8266 забезпечує обробку шифрування без участі головної системи Arduino.

Під час сканування аналізується спектр сигналів, що відповідають стандартам IEEE. У результаті формується перелік SSID (ідентифікаторів мереж) разом із супутньою інформацією, яка включає рівень сигналу, тип автентифікації, шифрування, а також параметри каналу [7].

Після ідентифікації цільової мережі, процес підключення продовжується з обміну службовими пакетами між Wi-Fi-модулем і точкою доступу. Сам мікроконтролер лише передає необхідні параметри, очікуючи відповідей по послідовному інтерфейсі. Після успішного отримання IP-адреси мікроконтролер набуває повноцінної мережевої ідентичності, що дозволяє йому ініціювати з’єднання з віддаленими серверами, обмінюватися даними через протокол MQTT, а також реагувати на зовнішні запити в реальному часі.

2.3 Мережевий протоко MQTT

MQTT (Message Queuing Telemetry Transport) є легковаговим мережевим протоколом, спеціально розробленим для середовищ з обмеженими ресурсами, таких як системи Інтернету речей (IoT), де важливими є мінімізація обсягу переданих даних і надійність зв’язку. Протокол працює поверх TCP/IP та реалізує модель взаємодії “видавець-підписник”, що дозволяє ефективно організувати обмін повідомленнями між пристроями без необхідності постійного з’єднання між ними.

Ключовими компонентами архітектури MQTT є видавець, підписник та брокер (рисунок 2.1). Видавець надсилає повідомлення певної тематики (topic) до брокера, який виступає центральною точкою обміну. Користувачі підписуються на конкретні теми, й брокер надсилає їм відповідні повідомлення. Таким чином, обидві сторони не мають прямого з’єднання одна з одною, що спрощує

масштабування системи і забезпечує високий рівень ізоляції компонентів [8].

Обмін повідомленнями в MQTT здійснюється через фіксовану структуру пакета, яка включає заголовок і навантаження. Заголовок визначає тип повідомлення (CONNECT, PUBLISH, SUBSCRIBE, UNSUBSCRIBE, PINGREQ тощо) і параметри доставки, а навантаження містить безпосередньо дані користувача. Структура пакета оптимізована для передачі через нестабільні або вузькосмугові канали, що робить MQTT придатним для мобільних мереж і бездротових IoT-застосувань.

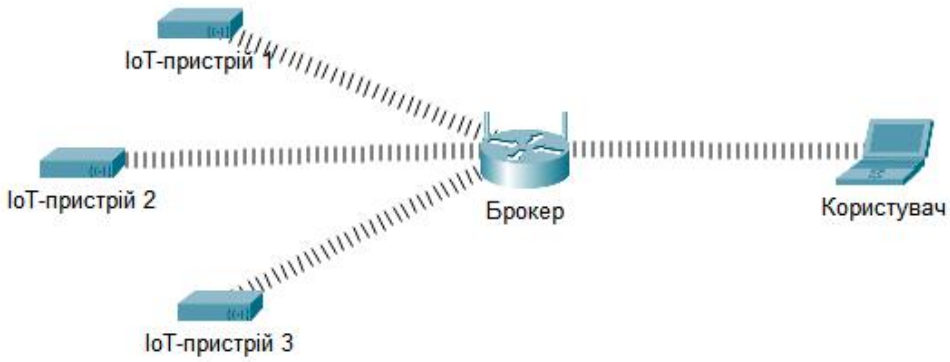

Рисунок 2.1 – Модель взаємодії компонентів MQTT

Протокол MQTT також підтримує механізм збережених повідомлень (retained messages) та повідомлень про відключення (Last Will and Testament – LWT). Збережені повідомлення дозволяють новим підписникам отримати останнє значення одразу після підписки, навіть якщо воно було опубліковане раніше. Повідомлення LWT надсилається автоматично брокером у разі неочікуваного розриву з’єднання , що дозволяє сигналізувати про несправність пристрою [9].

З точки зору безпеки, MQTT не включає в себе вбудованого шифрування, однак підтримує використання TLS/SSL поверх TCP-з’єднання. Це забезпечує захист переданих даних від перехоплення, автентифікацію учасників обміну та цілісність повідомлень. Додатково, сучасні MQTT-брокери реалізують механізми контролю доступу на основі імен користувачів, паролів, токенів чи сертифікатів, що є критично важливим для промислових або корпоративних IoT-рішень.

2.4 Гарантії повідомлень QoS

QoS (Quality of Service) в MQTT визначає рівень гарантій доставки повідомлень між клієнтом і брокером [9]. Протокол підтримує три рівні QoS: 0 (Максимум один), 1 (Мінімум один) і 2 (Рівно один). Кожен рівень відповідає певному компромісу між надійністю, пропускною здатністю і обчислювальними витратами. Рівень QoS встановлюється окремо для кожного повідомлення публікації або підписки.

QoS 0 (Максимум один) – це найпростіший рівень, при якому повідомлення надсилається лише один раз без підтвердження від отримувача. Клієнт передає брокеру пакет PUBLISH із прапором QoS=0, брокер, у свою чергу, не повертає жодного підтвердження (рисунок 2.2). Якщо з'єднання втрачається в момент передачі, повідомлення не буде доставлено. Цей режим не гарантує доставку й не допускає повторної передачі, тому підходить лише для сценаріїв, де втрата даних є припустимою – наприклад, для часто оновлюваних показників температури чи геопозиції [10].

Рисунок 2.2 – Рівень QoS 0

QoS 1 (Мінімум один) – Рівень QoS 1 гарантує, що повідомлення буде доставлено принаймні один раз, але можливе дублювання. Після відправки PUBLISH брокер має надіслати у відповідь PUBACK (рисунок 2.3). Якщо клієнт не отримує підтвердження у визначений таймаут, він повторно надсилає

повідомлення з тим самим Packet Identifier [10]. У разі повтору брокер зобов’язаний ідентифікувати дубль за ідентифікатором пакета й не генерувати нову подію, якщо вже обробив цей пакет.

Такий рівень забезпечує прийнятний компроміс між продуктивністю й надійністю у більшості застосувань, зокрема при надсиланні керуючих команд або змін параметрів пристроїв.

Рисунок 2.3 – Рівень QoS 1

QoS 2 (Рівно один) – гарантує унікальну доставку повідомлення без дублювання навіть у разі збоїв. Це найскладніший рівень, який передбачає обмін: після в чотири етапи отримання PUBLISH брокер відповідає PUBREC, клієнт надсилає PUBREL, а брокер завершує цикл PUBCOMP (рисунок 2.4). Протокол вимагає зберігати стан передачі до завершення транзакції, що вимагає додаткових ресурсів пам’яті й обробки [10].

Рівень QoS 2 використовується лише в критично важливих випадках, де неприпустиме дублювання або втрата повідомлень, наприклад – у фінансових або медичних IoT-системах.

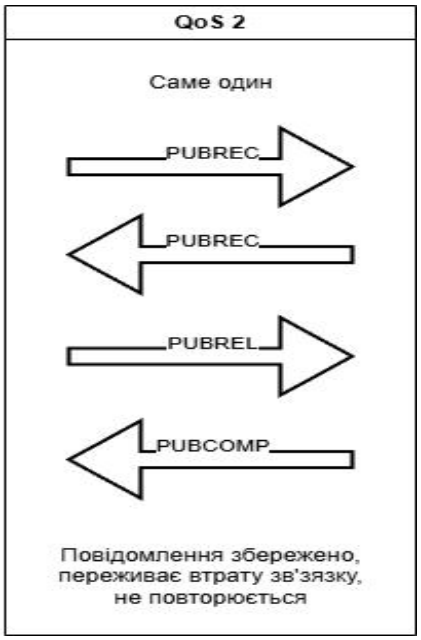

Рисунок 2.4 – Рівень QoS 2

При маршрутизації повідомлення брокер враховує QoS, вказаний як у публікації, так і в підписці. Рівень доставки визначається як мінімальний між QoS, з яким повідомлення було опубліковано, і QoS, з яким підписник оформив підписку.

2.5 Архітектурна модель MVVM

У розробці сучасних застосунків важливу роль відіграє архітектурна організація коду, що забезпечує розділення логіки представлення, бізнес-логіки та моделі даних. Однією з найбільш ефективних архітектурних моделей є шаблон MVVM (Model-View-ViewModel) [11]. Він був розроблений спеціально для WPF, щоб максимально використати можливості прив’язки даних, шаблонів та команд, закладені у .NET. MVVM сприяє зниженню зв’язності компонентів, покращує підтримку програмного забезпечення

Архітектурна модель MVVM поділяє застосунок на три основні компоненти: Model, View та ViewModel. Model відповідає за доступ до даних, їхню обробку та взаємодію з джерелами даних (рисунок 2.5). Вона реалізує бізнес-логіку та не

залежить від інтерфейсу користувача. View відповідає за графічне представлення, реалізоване у вигляді XAML-розмітки, та містить мінімальну логіку – переважно лише відображення інформації [12].

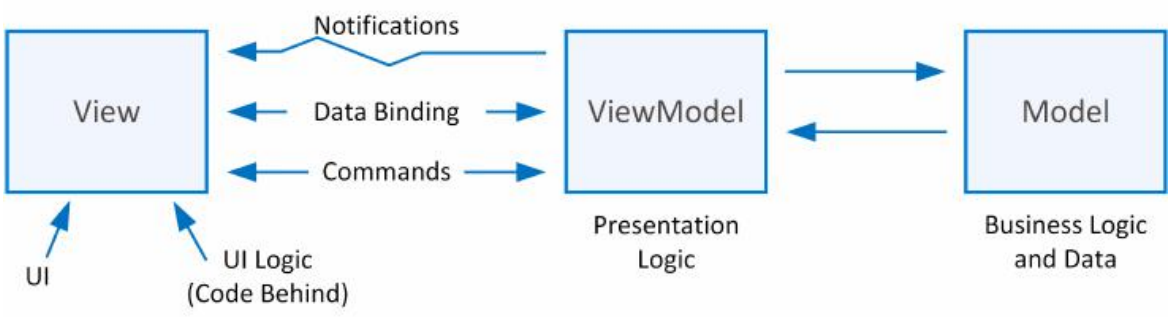

Рисунок 2.5 – Архітектурна модель MVVM

Компонент ViewModel є логічним представленням інтерфейсу користувача, але без прямої залежності від елементів графіки. Його основна функція – забезпечити взаємодію між даними та інтерфейсом. ViewModel використовує механізм прив’язки даних до властивостей та подій, що дозволяє автоматично оновлювати інтерфейс користувача при зміні стану даних. Обмін даними між View та ViewModel відбувається за допомогою двостороннього або одностороннього зв’язку. Це означає, що зміни в моделі автоматично відображаються в інтерфейсі, і навпаки – дії користувача можуть безпосередньо впливати на модель через ViewModel. Для обробки взаємодії модель MVVM застосовує спеціальний командний шаблон. Це дозволяє централізовано обробляти логіку без потреби у подієвих обробниках у самому інтерфейсі.

Основною перевагою MVVM є чітке розмежування обов’язків між компонентами застосунку. Це значно спрощує модульне тестування, повторне використання коду та масштабування проєктів. Відсутність прямої залежності між View та Model дозволяє розробникам та дизайнерам працювати незалежно один від одного. Крім того, у поєднанні з WPF ця модель дозволяє створювати гнучкі та адаптивні інтерфейси, що автоматично реагують на зміну стану програми.

2.6 Огляд існуючих рішень

Сфера дистанційного керування IoT-пристроями на базі Wi-Fi технології демонструє активний розвиток як комерційних, так і відкритих рішень, спрямованих на автоматизацію, налагоджування та моніторинг. Такі рішення, як правило, передбачають використання MQTT-протоколу, або інших легковагих протоколів (CoAP, AMQP) для обміну даними та мають високий рівень безпеки та взаємодію з зовнішніми API. У цьому контексті актуальним є аналіз існуючих систем, які можуть слугувати основною інженерних рішень для подальшої розробки.

Першим прикладом платформи є Node-Red [13], яка розробляється для побутового та напівкомерційного використання, зосереджуючи увагу на інтуїтивному керуванні пристроями. Система використовує власний брокер MQTT для обміну повідомленнями. Переважаючою особливістю Node-Red є наявність візуального середовища для гнучкого корегування пристроїв та створення продвинутих сценаріїв взаємодії без потреби в навичках в програмуванні (рисунок 2.6).

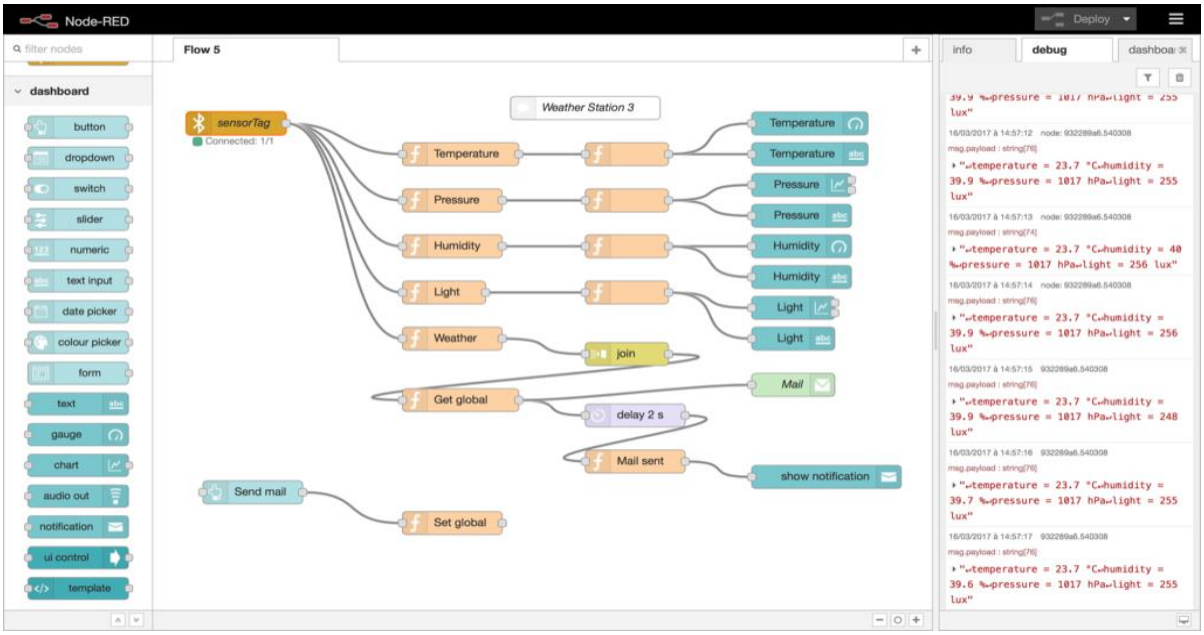

Рисунок 2.6 – Вікно середовища Node-Red

Другим прикладом системи є Google Cloud IoT Core, який забезпечує безпечне з'єднання IoT пристроїв через хмарну інфраструктуру Google [14]. Платформа підтримує мережеві протоколи MQTT та HTTP. Система дозволяє здійснювати обробку подій, зберігання історичних даних та інтеграцію з інструментами аналітики Google Cloud (рисунок 2.7). Особливу увагу приділено використанню TLS перевірки сертифікатів та IAM-політик доступу.

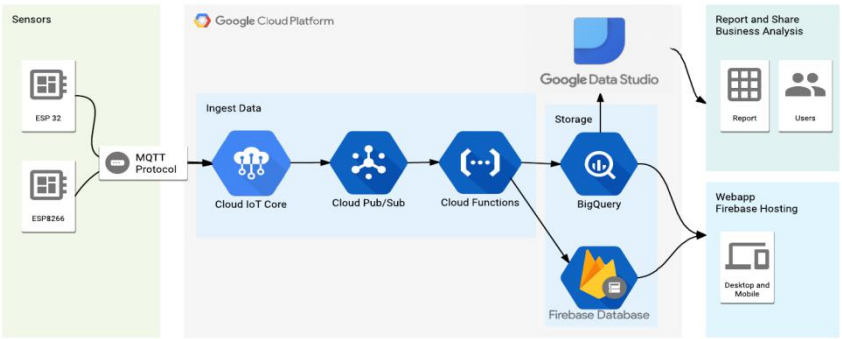

Рисунок 2.7 – Архітектура системи Google Cloud IoT Core

Третім рішенням є AWS IoT [15], що надає широкий спектр інструментів для підключення, керування та обробки даних від IoT пристроїв. Система підтримує більшість мережевих протоколів спілкування, має автентифікацію та шифрування даних і управління правами доступу. Однією з переваг є можливість інтеграції з сервісами Amazon, яка включає обробку потоків даних, аналіз та машинне навчання. Це робить платформу придатною до рішень з високими вимогами до надійності та масштабування.

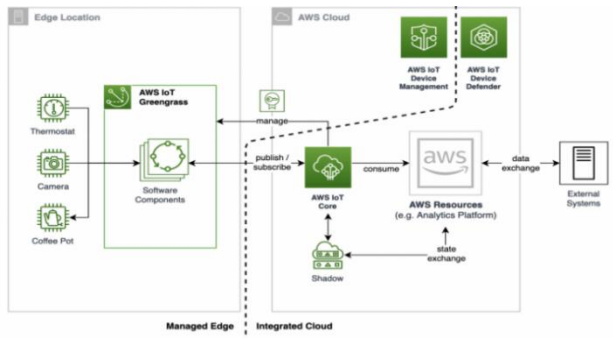

Рисунок 2.8 – Архітектура системи AWS IoT

Незважаючи на потужний функціонал, більшість рішень представлених платформ є надлишковими для задач середнього рівня користувачів, через складність використання і щільну інтеграцію з внутрішніми сервісами, високу вартість і потребу в стабільному інтернет з'єднанні. У цьому контексті перевагу надають зазвичай гібридним рішенням на основі відкритих брокерів, які дозволяють реалізувати критичний функціонал із мінімальними рішеннями.

Ефективне дистанційне керування IoT пристроями повинне передбачати баланс між простотою реалізації, функціональністю, безпекою та економічністю рішення. Вибір платформи залежить від контексту застосування, для побутових користувачів та малих бізнесів існує необхідність створення універсального рішення.

2.7 Формування підходу до реалізації

Розробка системи потребує формування підходу, який спрямований на специфіку інформаційної взаємодії, обмеження апаратних ресурсів та вимоги в зручному користувацькому інтерфейсі. Така система повинна підтримувати ефективний формат обміну даними, структуровану передачу повідомлень та гнучке масштабування. Апаратна частина основана на мікроконтролері сімейства Arduino, а саме плати що мають вбудовані Wi-Fi модулі (ESP8266, ESP32), що завдяки своїм характеристикам є оптимальним вибором для створення IoT середовища.

Для підтримки стандартів безпеки та сумісності зі зовнішніми сервісами передбачено інтеграцію з брокер-сервісами, що мають відповідати сучасним вимогам шифрування та автентифікації.

Функціональні та нефункціональні вимоги архітектурного рішення наведено в таблиці 2.1

Таблиця 2.1 Функціональні та нефункціональні вимоги

Функціональні вимоги	Нефункціональні вимоги
Підтримка з’єднання з MQTT-брокером	Можливість масштабування системи (додавання пристроїв, тем, користувачів)
Підтримка підписки на топіки та публікації повідомлень	Адаптивність інтерфейсу під різні розміри екранів
Реєстрація та збереження параметрів сенсорів	Уніфікований формат обміну даними, придатний для аналізу та обробки
Візуалізація інформації про стан пристроїв у реальному часі	Мінімальні вимоги до апаратного забезпечення клієнтської системи
Логування подій та змін стану пристроїв	
Додавання та видалення пристроїв з інтерфейсу	
Підтримка ручного налаштування параметрів підключення	

Основним транспортним рівнем обміну повідомленнями доцільно обрати протокол MQTT. Його особливістю є побудова взаємодії за моделлю “видавець-підписник”, що дає змогу реалізувати асинхронний обмін інформацією та мінімізувати кількість переданих даних. Усі повідомлення повинні передаватися через MQTT-брокер, який виконує роль посередника між пристроями та клієнтськими застосунками. Такий підхід дозволяє забезпечити розмежування між джерелами та споживачами даних, спростити логіку комунікації та підвищити загальну надійність системи.

Передача інформації між пристроями має здійснюватися у структурованому форматі, який легко інтерпретується як на стороні мікроконтролера, так і в середовищі високорівневого застосунку. Для цього доцільно застосувати формат JSON – текстовий, легко в оперуванні стандарт, що забезпечує уніфіковану структуру повідомлень. З його допомогою можливо передавати як окремі

параметри сенсорів, так і складні об’єкти з вкладеною структурою, що особливо зручно для подальшої обробки в клієнтському інтерфейсі.

Клієнтська частина реалізована у вигляді застосунку для систем сімейства Windows з використанням технології WPF. Це дозволяє створити адаптивний графічний інтерфейс із підтримкою сучасних патернів, зокрема моделі MVVM, що забезпечує розділення логіки інтерфейсу, обробки подій та моделі даних. Такий підхід є оптимальним для реалізації функцій моніторингу, управління, візуалізації та логування стану пристроїв у реальному часі.

Користувацький інтерфейс необхідно організувати інтуїтивним, що має дозволити користувачу без додаткової підготовки здійснити налаштування та необхідний моніторинг в мережі. Така структура дозволяє сформувати адаптивне середовище, яке може бути модифіковане в випадку необхідності. Запропонований підхід надає змогу забезпечити необхідний баланс між технічною реалізацією та потенціалом до подальшого розвитку.

Загальний підхід до проєктування системи має враховувати можливість її масштабування: розширення кількості пристроїв, підключення до хмарних сервісів або зміни конфігурації без потреби у зміні головної інфраструктури пристрою. На рівні програмної архітектури необхідно забезпечити незалежність між функціональними модулями, форматами обміну та мінімізацію впливу помилок на роботу всієї системи.

3 ВИБІР ЗАСОБІВ РЕАЛІЗАЦІЇ

Процес розробки системи дистанційного керування IoT-пристроями на базі Wi-Fi технології потребує чітко визначеного комплексу інструментальних засобів, які забезпечать якісну реалізацію поставленої задачі. Обрані засоби розробки мають відповідати сучасним вимогам до продуктивності, масштабованості, сумісності з існуючими протоколами обміну даними та забезпечувати достатню гнучкість для майбутньої адаптації чи розширення системи.

До основних критеріїв вибору належить наявність інструментів для моделювання, налагодження, а також ефективної взаємодії з апаратною складовою системи. Такі засоби мають забезпечувати не лише розробку логіки функціонування пристроїв, а й ефективну обробку та візуалізацію даних, інтеграцію з мережевими службами та підтримку необхідних стандартів.

Раціональний підхід до вибору засобів реалізації не лише дозволяє спростити процес розробки, а й підвищити якість фінального продукту розробки.

3.1 Онлайн-брокер повідомлень – MQTTX

Для реалізації поставленої задачі було обрано брокер MQTTX [16], що є відкритим, багатоплатформенним брокером, що підтримує мережевий протокол MQTT версії номер 5. MQTTX дозволяє реалізувати двосторонній обмін даними між клієнтським застосунком і IoT пристроєм у режимі реального часу. Сервіс підтримує базові налаштування безпеки та шифрування.

Брокер MQTTX забезпечує стабільну роботу з мінімальними затримками, що є критично важливим для IoT-систем, орієнтованих на керування у реальному часі. Підтримка тем мережі дозволяє логічно організовувати структуру повідомлень, а також здійснювати їх фільтрацію.

MQTTX сумісний з Arduino-платформами та клієнтами на C#, що забезпечує гнучкість при побудові багатокомпонентних систем. Брокер активно підтримується та оновлюється, збільшуючи його популярність у використанні.

3.2 Середовище розробки – Visual Studio

Середовище розробки Visual Studio було обрано як платформа для створення клієнтського застосунку через його широку підтримку .NET-платформи, велику кількість інструментів для відлагодження та зручну інтеграцію з системами керування версіями. Інтегроване середовище має потужний редактор інтерфейсів, що значно прискорює процес розробки застосунків для Windows [17].

Visual Studio підтримує інтеграцію з компонентами WPF, що дає змогу розробити різні графічні інтерфейси, використовуючи стилі та шаблони згідно до поставленої задачі. Наявність автоматичних підказок при розробці і розширень дозволяє швидко генерувати програмні модулі, перевіряти синтаксис та відстежувати логіку виконання.

3.3 Графічна підсистема – WPF

Для створення графічного інтерфейсу користувача було обрано технологію Windows Presentation Foundation (WPF), яка забезпечує потужний та гнучкий інструментарій для розробки додатків під операційну систему Windows. WPF базується на сучасних підходах до побудови інтерфейсів, що дозволяє розробникам формувати складні та інтерактивні візуальні компоненти із застосуванням апаратного прискорення графіки [11]. Завдяки цьому досягається висока продуктивність роботи застосунку.

Однією з ключових особливостей WPF є використання мови розмітки XAML, що дозволяє чітко розділяти представлення даних і їх обробку. Це сприяє підвищенню читабельності коду та полегшує його підтримку і розвиток. Використання XAML дає змогу швидко створювати та змінювати візуальні компоненти без потреби суттєвої модифікації логіки, що значно пришвидшує процес розробки та адаптації інтерфейсу [18].

Додаткову зручність при розробці забезпечує підтримка шаблонів оформлення, стилів і анімаційних ефектів, що покращує загальну взаємодію

користувача із застосунком.

Технологія WPF також надає широкі можливості для реалізації інтерактивних елементів управління, таких як кнопки, таблиці, діаграми, індикатори тощо. Це створює умови для формування повноцінного інтерфейсу керування, придатного як для навчального, так і для практичного застосування, з незначними вимогами до обчислювальних ресурсів комп'ютера.

3.4 Система керування версіями – GIT

У ході розробки програмного забезпечення було застосовано систему керування версіями Git, яка відзначається високою гнучкістю, надійністю та широкими функціональними можливостями [19]. Використання Git дало змогу впорядкувати процес розробки, забезпечити контроль за змінами у програмному коді та зберігати повну історію модифікацій. У межах створення клієнтського застосунку та програмного забезпечення для мікроконтролера Git використовувався як у локальному режимі, так і з підключенням до віддаленого репозиторію.

Завдяки підтримці створення ізольованих гілок, система дозволяє паралельно розробляти та тестувати окремі функціональні компоненти без ризику порушення основної версії проєкту. Це, у свою чергу, сприяє зменшенню ймовірності втрати даних і полегшує інтеграцію нових елементів у загальну структуру застосунку.

3.5 Система управління базами даних – Microsoft SQL Server

Для збереження даних, отриманих з IoT-пристрою, а також даних, що спрямовані на керування цими пристроями, було обрано систему управління базами даних Microsoft SQL Server (MSSQL). Цей вибір зумовлений її здатністю забезпечувати надійне зберігання, ефективну фільтрацію та обробку значних

обсягів інформації, що є критично важливим для систем, які працюють у режимі реального часу або близькому до нього. MSSQL зарекомендувала себе як стабільна й потужна платформа, яка підтримує складні SQL-запити, реалізацію транзакцій [20].

СКБД MSSQL дозволяє реалізувати гнучку структуру бази даних, що відповідає моделі збереження телеметричної інформації, часові мітки подій, станів пристроїв та команди управління. Таке структурування дозволяє оперативно реагувати на поточні події. Завдяки розширеним засобам безпеки – включаючи контроль доступу на рівні ролей, шифрування даних як у стані зберігання, так і при передачі, а також можливості автоматизованого резервного копіювання – MSSQL забезпечує високий рівень захисту критичної інформації.

3.6 Формат обміну даними – JSON

У сучасних інформаційних системах, особливо у контексті розподілених клієнт-серверних архітектур, важливе значення має стандартизований формат обміну даними між компонентами. Однією з найпоширеніших технологій, що забезпечує ефективну серіалізацію та десеріалізацію структурованих даних, є JSON [21]. Формат став ключовим інструментом для передачі даних у веб-застосунках, мобільних платформах, хмарних середовищах, а також у вбудованих системах, зокрема в мікроконтролерах та IoT-пристроях.

Структура даних у форматі JSON представлена у вигляді пар “ключ–значення”, де ключ є рядком, а значенням може бути рядок, число, логічне значення, масив або вкладений об’єкт [21].

У контексті клієнт-серверної взаємодії JSON широко використовується для обміну даними через протокол MQTT. На серверній стороні або у мікроконтролерному пристрої, сформований JSON-документ серіалізується у вигляді рядка, який передається на інший бік каналу зв’язку. Отримувач, у свою чергу, виконує десеріалізацію цього рядка назад у структури даних, придатні до обробки.

Головними перевагами JSON є його компактність, простота синтаксису, підтримка більшістю мов програмування та відповідність принципам REST-архітектури. Це робить його оптимальним вибором для передачі даних у середовищах з обмеженими ресурсами або ж при необхідності зменшити обсяг трафіку. У порівнянні з альтернативними форматами, такими як XML, JSON має меншу надлишковість і дозволяє досягти більш високої швидкості обробки.

3.7 Середовище розробки систем Ардуіно – PlatformIO

PlatformIO є потужним інструментом для розробки вбудованих систем, який широко використовується для створення, компіляції та налагодження систем на базі різних мікроконтролерів, включно з популярними сімействами Arduino та ESP32 [22]. Завдяки своїй універсальності та підтримці великої кількості апаратних платформ, PlatformIO забезпечує розробникам ефективний інструментарій для реалізації складних проєктів у сфері IoT та автоматизації.

Однією з ключових переваг PlatformIO є інтеграція з багатьма бібліотеками та плагінами, що спрощує підключення різноманітних модулів і датчиків, а також надає готові рішення для взаємодії з мережевими протоколами, зокрема Wi-Fi, MQTT та іншими. Автоматизація процесів збірки, компіляції та прошивки знижує ризики помилок на ранніх етапах розробки, дозволяючи швидко тестувати та оптимізувати код перед його вивантаженням у цільовий пристрій.

Інструментарій PlatformIO також передбачає можливості налагодження та емулювання окремих частин системи, що дає змогу виявляти помилки та перевіряти працездатність алгоритмів без необхідності фізичного підключення апаратного забезпечення.

Завдяки PlatformIO можливо створювати універсальні компоненти, які згодом можуть бути перенесені на інші платформи або пристрої. Це забезпечує гнучкість, надійність та дозволяє перевірити коректність роботи логіки для впровадження її у фінальну систему.

3.8 Веб-платформа для симуляції – Wokwi

Wokwi – це веб-платформа для симуляції роботи мікроконтролерів, який може бути інтегрований у середовище розробки PlatformIO для розширення можливостей налагодження та тестування проєктів на базі Arduino та інших платформ. Він забезпечує емуляцію апаратної частини мікроконтролерів, дозволяючи виконувати код у віртуальному середовищі без фізичного підключення пристрою.

Wokwi підтримує розширені можливості для налагодження, такі як емуляція серійного порту, шин зв'язку та зовнішніх модулів, а також візуалізація роботи цифрових і аналогових компонентів [23]. Завдяки такій інтеграції забезпечується більш ефективний цикл розробки вбудованих систем із можливістю швидкого прототипування і тестування без необхідності залучення фізичного апаратного забезпечення.

3.9 Використані бібліотеки

У процесі розробки клієнтського застосунку та програмного забезпечення для IoT-пристрою було використано низку зовнішніх бібліотек, що забезпечили базову функціональність обміну даними, взаємодії з мережевими протоколами, обробки подій, графічного інтерфейсу та роботи з апаратними компонентами. Застосування перевірених бібліотек дозволило суттєво скоротити час розробки, уникнути типових помилок та забезпечити стабільність і масштабованість рішення. Усі бібліотеки було інтегровано відповідно до технічних вимог програмного середовища та протестовано в рамках цільових задач системи. Бібліотеки використані для програмного забезпечення клієнтського застосунку та пристрою IoT наведено в таблиці 3.1 та таблиці 3.2.

Таблиця 3.1 Використані бібліотеки для клієнтського додатку

Бібліотека	Призначення
LiveCharts	Відображення даних у вигляді графіку
EntityFrameworkCore	Взаємодія з базою даних
MQTTnet	Взаємодія через MQTT протокол
JSON	Серіалізація/десеріалізація JSON формату

Таблиця 3.2 Використані бібліотеки для пристрою IoT

Бібліотека	Призначення
ArduinoJSON	Серіалізація/десеріалізація JSON формату
DS3231	Набір інструкцій для апаратної частини таймеру
PubSubClient	Клієнт частина протоколу MQTT

Застосування готових бібліотек дозволило реалізувати надійну і модульну архітектуру програмної системи, яка підтримує основні функції IoT-взаємодії. Усі компоненти були обрані з урахуванням сумісності, підтримки спільноти та відповідності ліцензійних умов, що у сукупності підвищило ефективність розробки та забезпечило належну якість реалізованого рішення.

4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Програмна реалізація системи дистанційного керування IoT пристроєм охоплює розробку двох основних компонентів роботи системи: клієнтського застосунку на основі платформи WPF та системи для контролеру сімейства Arduino. Клієнтський застосунок реалізовано з використанням архітектурного шаблону Model-View-ViewModel (MVVM), який спрямований на розділення відповідальностей між елементами інтерфейсу [12].

Програмне забезпечення для платформи Arduino реалізовано з використанням парадигми Об'єктно-Орієнтованого програмування, що дало змогу створити ефективну робочу базу для подальшої інтеграції з зовнішніми апаратними модулями. Уся система протестована з використанням онлайн-брокера MQTTX. В ході розробки реалізовано зберігання інформації у базі даних MSSQL.

4.1 Основні функціональні можливості системи

У процесі розробки програмної системи для керування IoT-пристроями важливо забезпечити логічно структуровану модель взаємодії користувача з функціональними модулями. Для цього було створено діаграму прецедентів, що подана на рисунку 4.1.

Діаграма прецедентів демонструє ключові сценарії взаємодії користувача з системою. У центрі моделі знаходиться актор "Користувач", який через клієнтський інтерфейс виконує базові дії з профілем, керування мережею пристроїв (огляд доступних та пристроїв що відстежуються, додавання й видалення), а також адміністрування окремого пристрою (налаштування модулів, редагування параметрів, зміна теми підписки, моніторинг роботи).

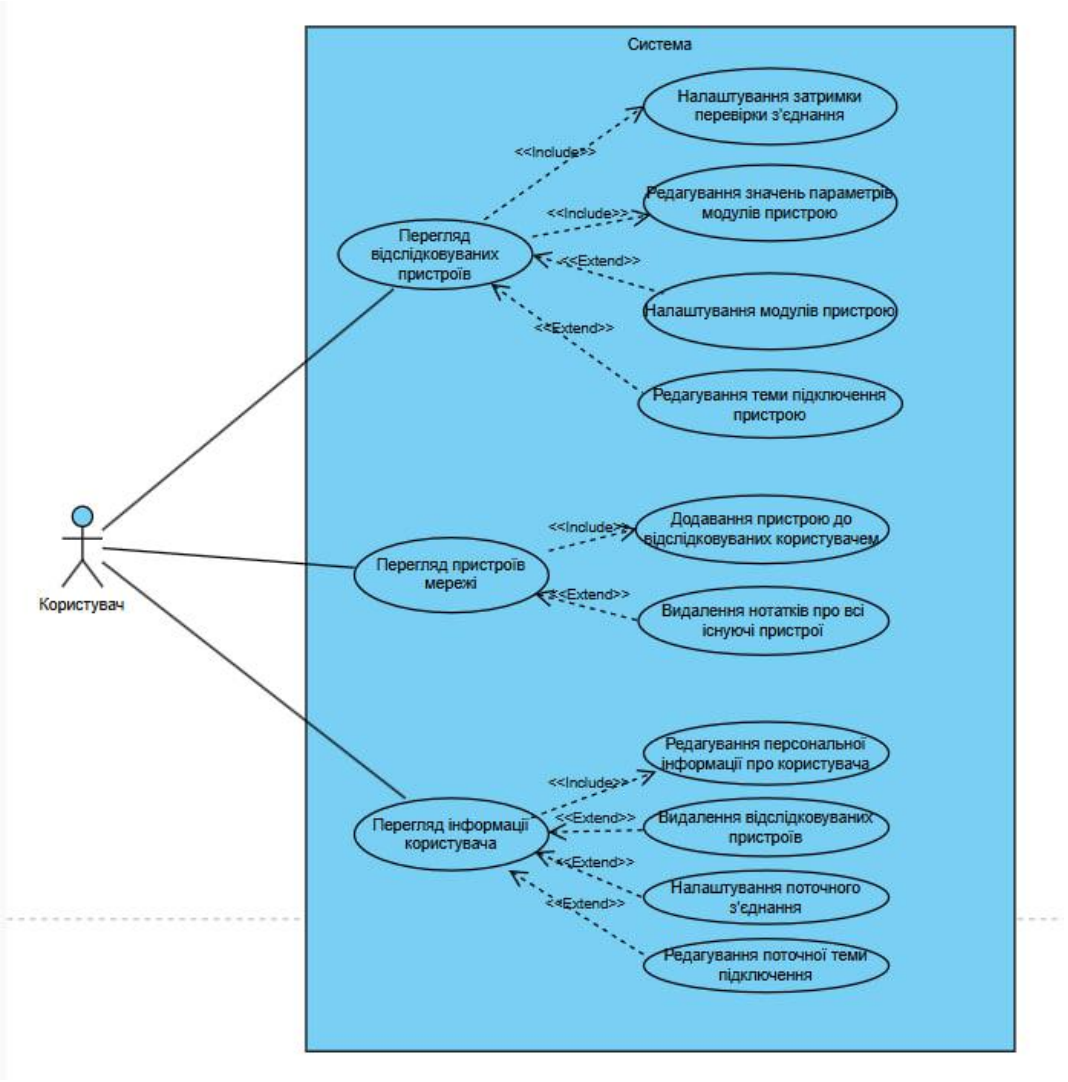

Рисунок 4.1 – Діаграма прецедентів

Наявність чітко визначених сценаріїв взаємодії дозволяє оптимізувати процес розробки, забезпечити відповідність реалізації вимогам до зручності, надійності та адаптивності системи дистанційного керування IoT-пристроями.

4.2 Організація моделей і зв’язків

Розробка програмної системи потребує чітко структурованих програмних компонентів, налагодження взаємодії між користувачем, серверною логікою та фізичними пристроями, а також побудови ефективної архітектури даних.

Підрозділ містить опис архітектури застосунку, структури даних, організації

класів програмного забезпечення для мікроконтролера та послідовності передачі повідомлень між основними компонентами системи.

Використання сучасних шаблонів проєктування, таких як MVVM, дозволило розділити відповідальність між окремими частинами коду та забезпечити його підтримуваність. Також реалізовано логічну модель бази даних, яка дозволяє ефективно зберігати та обробляти інформацію про користувачів і підключені IoT-пристрої.

4.2.1 Діаграма зв’язків сутностей

У рамках розробки системи важливою складовою є належна організація зберігання та обробки даних. Для цього була спроєктована реляційна база даних (рисунок 4.2), яка забезпечує збереження інформації про користувачів, пристрої, теми підключень, властивості пристроїв, а також параметри підключення.

Модель даних враховує логічні зв’язки між сутностями та дозволяє гнучко керувати взаємодією користувача з пристроями в контексті MQTT-подій.

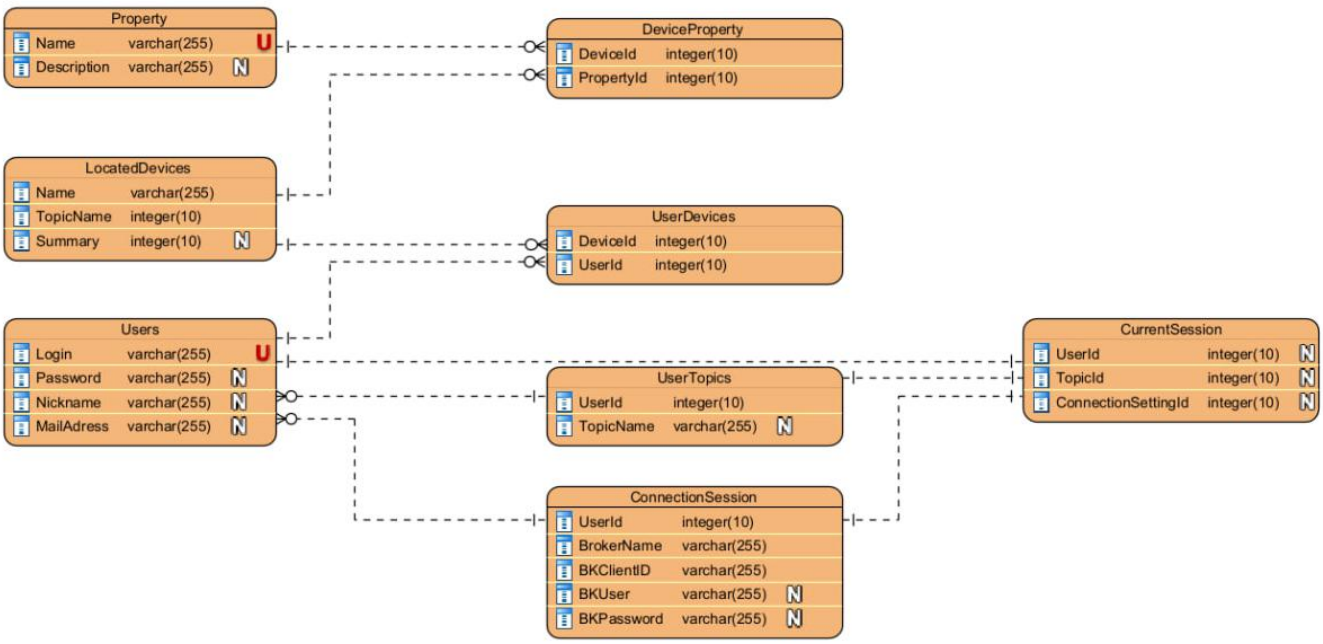

Рисунок 4.2 – ER-діаграма

База даних складається з основних таблиць: Users, LocatedDevices, UserDevices, UserTopics, DeviceProperty, Property, ConnectionSession, та CurrentSession. Центральною є таблиця Users, яка зберігає облікову інформацію про користувачів, включаючи логін, пароль, ім’я та електронну пошту. Кожен користувач може мати доступ до одного або декількох пристроїв, що реалізовано через проміжну таблицю UserDevices, яка пов’язує Users з LocatedDevices. Додатково, таблиця UserTopics визначає, до яких MQTT-тем підписаний користувач, що забезпечує адресність повідомлень у системі.

Загалом, така організація бази даних дозволяє реалізувати масштабовану, безпечну та структуровану платформу для інтерактивного керування IoT-пристроями.

Зв’язки між таблицями забезпечують логічну цілісність даних, тоді як нормалізована структура сприяє розширенню системи без втрати продуктивності.

4.2.2 Діаграма класів клієнтського застосунку

Діаграма класів клієнтської частини (рисунок 4.3) відображає архітектурну організацію програмного забезпечення, створеного із застосуванням платформи WPF та патерну MVVM (Model-View-ViewModel). Такий підхід дозволяє розділити відповідальність між різними шарами застосунку. Це підвищує зрозумілість структури програми, полегшує її супровід і масштабування, а також сприяє багаторазовому використанню коду та ефективному тестуванню окремих компонентів.

У центрі моделі перебувають ViewModel-класи, такі як MainViewModel, LoginViewModel, DeviceListPanelViewModel, EditDevicePageViewModel тощо. Вони реалізують логіку обробки взаємодій користувача з інтерфейсом. Завдяки цьому UI не містить бізнес-логіки, що забезпечує чисту архітектуру застосунку. Зв’язки між ViewModel і моделлю даних (DeviceManager, DataManager, MQTTManager) організовано через залежності, що дозволяє ViewModel легко обмінюватися даними та координувати взаємодії між різними частинами системи.

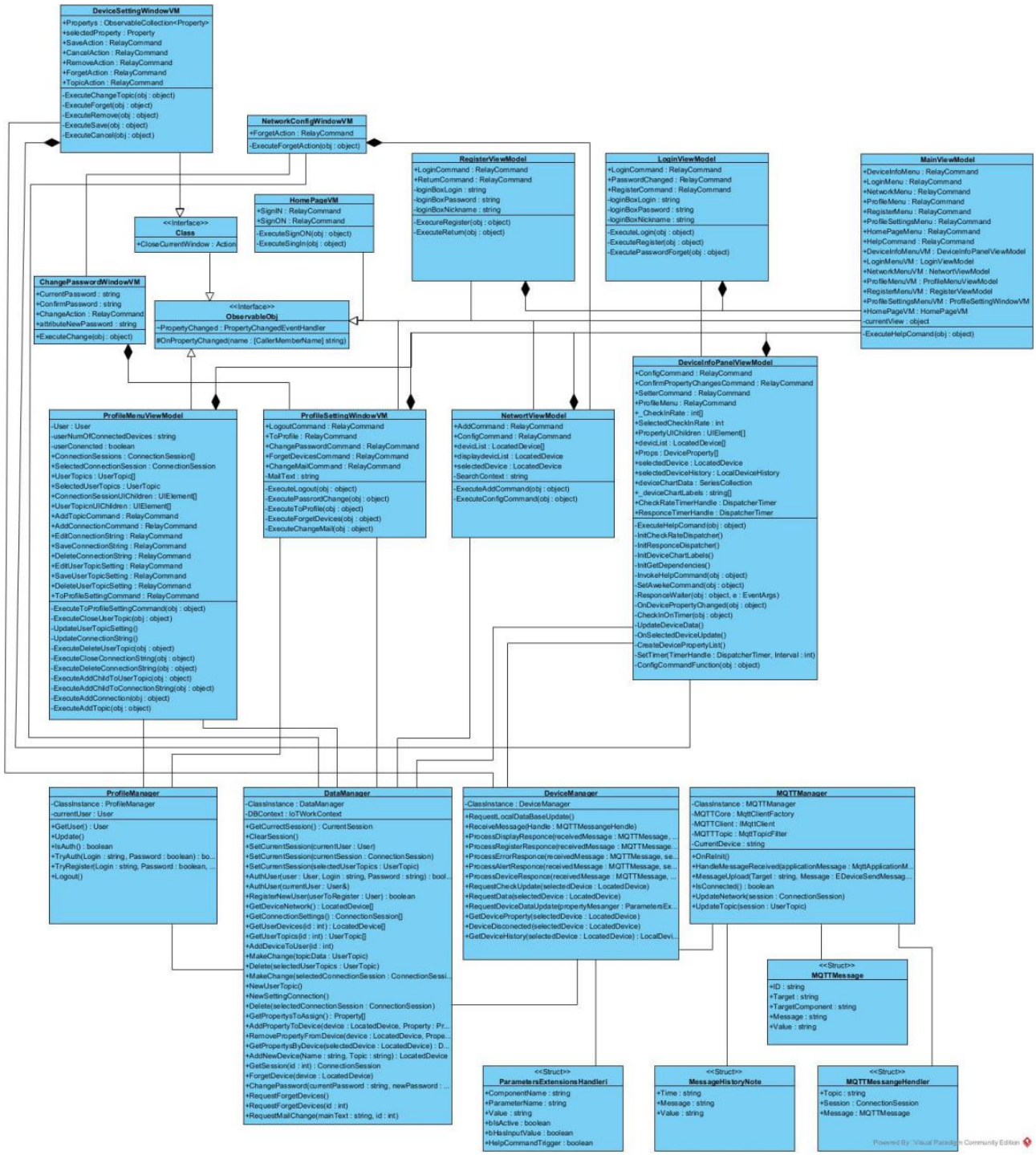

Рисунок 4.3 – Діаграма класів клієнтського застосунку

Модель включає також класи, що відповідають за обробку повідомлень (MQTTMessage, MQTTMessageHandler) та їхнє надсилання або приймання через MQTT-протокол. DeviceManager відповідає за управління пристроями, зокрема реєстрацію, зчитування властивостей, оновлення тощо. DataManager виконує функції збереження сесій, авторизації користувачів та обробки з’єднань. Загалом,

архітектура демонструє чітку організацію взаємодії компонентів клієнтської частини, орієнтовану на стабільність, масштабованість і підтримку в умовах динамічного розвитку IoT-рішень.

4.2.3 Діаграма класів програмного забезпечення для Arduino

Діаграма класів, представлена на рисунку 4.4, демонструє архітектуру програмного забезпечення мікроконтролерної частини IoT-системи, реалізованої на основі платформи Arduino. В основі побудови даної архітектури лежить принцип інтерфейсно-орієнтованого проєктування, що забезпечує високу гнучкість, масштабованість і повторне використання програмних компонентів. Такий підхід дозволяє кожному сенсору або модулю виступати як окрема сутність, яка реалізує стандартний набір інтерфейсів, спрощуючи інтеграцію нових пристроїв без необхідності модифікації базового коду.

Функціональні класи, зокрема MSmokeDetector, MEchoLocator, MTimer реалізують специфічну поведінку сенсорів та пристроїв керування Ці класи реалізують методи для ініціалізації, читання даних та виконання дій у відповідь на події. Завдяки цьому забезпечується чітке розділення обов’язків між логікою обробки сигналів та механізмами передачі даних.

Центральним елементом архітектури є клас Controller, який координує роботу усіх модулів системи. Він обробляє отримані MQTT-повідомлення за допомогою класу CMQTTMessageComponent і забезпечує виконання відповідних дій залежно від змісту повідомлень. Взаємодія із зовнішнім брокером MQTT здійснюється через клієнт PubSubClient, а структура SMQTTMessageContainer використовується для уніфікованого представлення контенту повідомлень. Така організація дозволяє зручно реалізовувати віддалене керування пристроями та зворотній зв’язок.

Додаткові структури, такі як EResponceType, EAwakeStatus, ELogPriority забезпечують стандартизацію форматів обміну даними та контроль логічних станів.

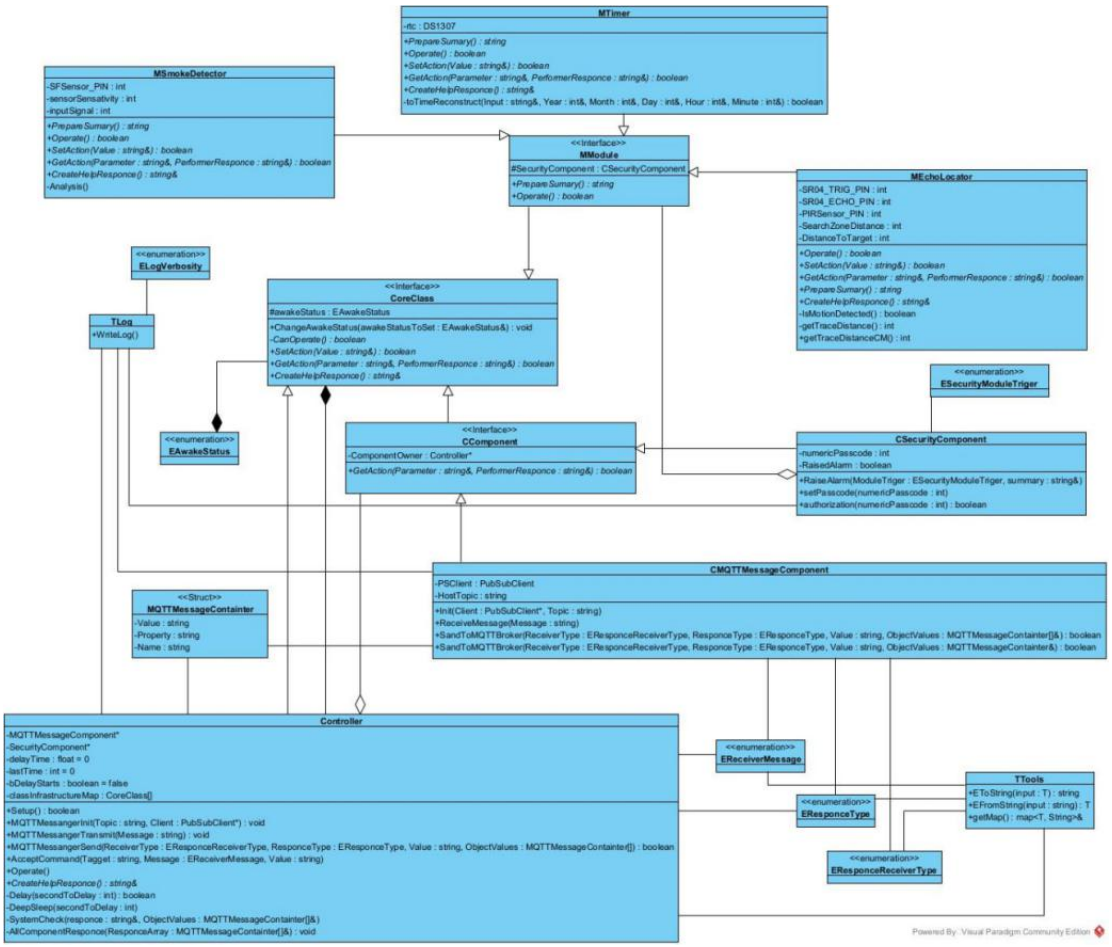

Рисунок 4.4 – Діаграми класів програмного забезпечення для Arduino

Завдяки запропонованій архітектурі забезпечується можливість динамічного масштабування системи, розширення функціоналу шляхом додавання нових модулів без втручання в основну логіку. Оптимізована взаємодія між модулями, чітко визначені інтерфейси та мінімізація взаємозалежностей дозволяють ефективно використовувати обчислювальні ресурси мікроконтролера

4.2.4 Діаграма послідовностей передачі даних через MQTTX

Діаграма послідовностей (рисунок 4.5) відображає порядок взаємодії між трьома основними компонентами системи: пристроєм (Device), брокером MQTT (Broker) та клієнтським застосунком (App).

Представлений сценарій демонструє обмін повідомленнями за протоколом MQTT у рамках архітектури типу “публікація-підписка”, що дозволяє реалізувати асинхронну, стійку до збоїв комунікацію. Така структура особливо придатна для побудови систем, які працюють у нестабільному мережевому середовищі або з великою кількістю пристроїв.

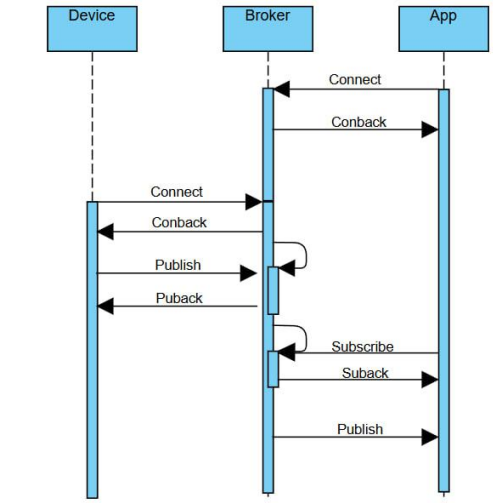

Рисунок 4.5 – Діаграма послідовностей

У процесі встановлення з’єднання як пристрій, так і клієнт ініціюють підключення до брокера за допомогою команди CONNECT. У відповідь брокер надсилає підтвердження (CONNACK), що означає успішне встановлення сесії. Після цього пристрій починає передавати телеметричні дані до брокера шляхом надсилання повідомлення типу PUBLISH. MQTT-брокер підтверджує отримання кожного повідомлення командою PUBACK, чим забезпечує гарантію доставки на транспортному рівні.

Паралельно клієнтський застосунок здійснює підписку на відповідні теми MQTT (топіки) через команду SUBSCRIBE, після чого отримує підтвердження SUBACK. Після налаштування підписки клієнт автоматично приймає всі повідомлення, які публікує пристрій у межах заданого топіка. Це дозволяє реалізувати повноцінну односторонню або двосторонню взаємодію між пристроєм і застосунком без необхідності прямого з’єднання між ними.

Після налаштування підписки застосунок автоматично отримує повідомлення, опубліковані пристроєм. Завдяки механізму публікації-підписки забезпечується слабкий зв’язок між відправником і отримувачем. Такий підхід дозволяє обслуговувати значну кількість пристроїв з мінімальними витратами на мережеву інфраструктуру, що особливо актуально для обмежених в ресурсах середовищ.

4.3 Тестування апаратної частини пристрою IoT

З метою комплексної перевірки працездатності IoT-системи було побудовано два експериментальні прототипи пристроїв IoT: один із застосуванням емулятора на базі мікроконтролера ESP32, інший – з використанням реальної апаратної частини на платформі ESP8266. Такий підхід дозволив протестувати як логіку сенсорної системи, так і стабільність функціонування фізичних компонентів у реальних умовах. В обох випадках взаємодія з клієнтською частиною забезпечувалась через MQTT-протокол за допомогою онлайн-брокера.

У симуляційній моделі було задіяно три умовні сенсорні компоненти: датчик диму, датчик руху та ультразвуковий датчик відстані (рисунок 4.6). Емуляція поведінки здійснювалась через ручну взаємодію з параметрами датчиків (рисунок 4.7). Такий спосіб реалізації дозволив створити віртуальне середовище, максимально наближене до реального, що сприяло аналізу продуктивності клієнтського застосунку та стабільності з’єднання з брокером MQTT.

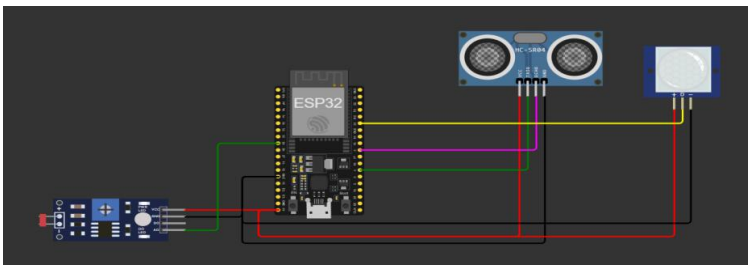

Рисунок 4.6 – Вигляд пристрою в симульованому середовищі

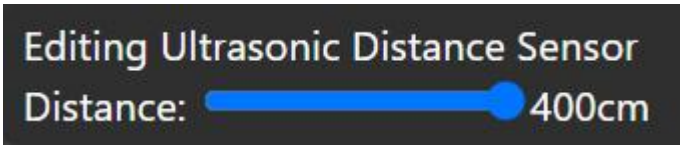

Рисунок 4.7 – Вікно взаємодії з параметром датчика в емульованому середовищі

У реальній апаратній реалізації було використано ESP8266 як основний контролер, до якого підключено датчик диму, ультразвуковий сенсор та програмований таймер (рисунок 4.8). Пристрій функціонував з живленням через USB-інтерфейс, і передавав дані безпосередньо до MQTT-брокера через Wi-Fi з’єднання. Всі компоненти фізично змонтовані на макетній платі з реалізованою логікою обробки сигналів через розроблену інфраструктуру.

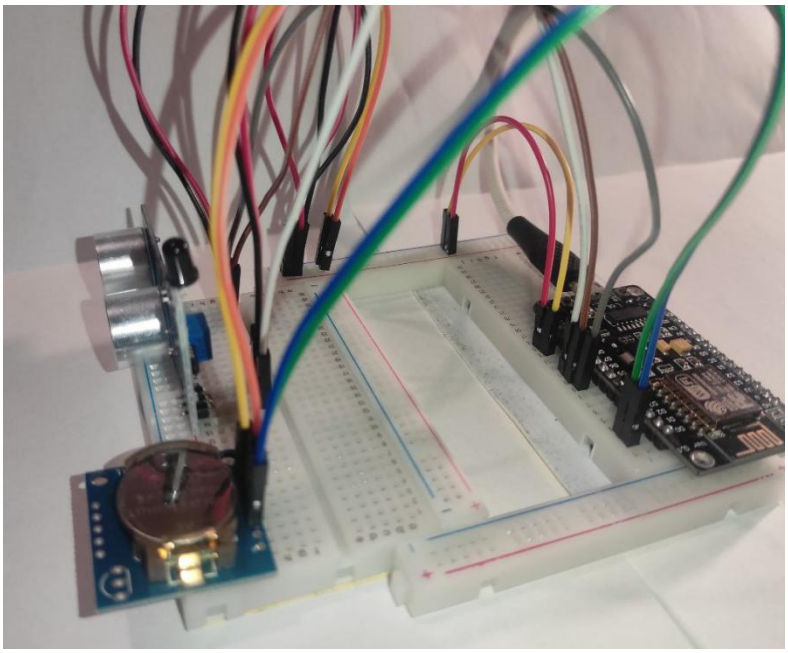

Рисунок 4.7 – Вигляд змодельованого пристрою

Основним завданням тестування було забезпечення надійності з’єднання між мікроконтролером та брокером MQTT при тривалій роботі в умовах змінної інтенсивності трафіку. Система успішно підтримувала підключення впродовж безперервної сесії довжиною 2 години без розривів або помилок повторного підключення, що свідчить про високу стійкість до тимчасових затримок або короткочасних перебоїв у мережі.

Другим етапом перевірки була стійкість до технічного навантаження, яке моделювалось шляхом передачі великої кількості MQTT-повідомлень з брокера на пристрій. Упродовж тестової сесії було згенеровано понад 100 повідомлень з інтервалами в межах 1 секунди, що дозволило оцінити швидкодію ESP8266 при високій частоті приймання команд. Усі повідомлення були оброблені без втрат, із середньою затримкою нижче 1 секунди.

Третім критичним параметром виступала узгодженість обробки даних між клієнтським застосунком, брокером та пристроєм. Тестування включало ситуації, у яких дані з сенсорів передавались із високою частотою, а клієнт у реальному часі виконував аналіз, відображення та логування параметрів. Було підтверджено, що навіть при одночасному надходженні кількох типів повідомлень система не втрачала цілісність даних та підтримувала правильну хронологію.

Крім того, було перевірено адаптивність пристрою до збоїв у передачі. У симуляційній моделі періодично вводилися помилкові MQTT-пакети та недоступність брокера. У кожному випадку пристрій коректно реагував: при втраті зв’язку переходив у режим очікування, автоматично перепідключався до брокера та відновлював сесію після зміни параметрів.

Результати тестування підтверджують, що реалізована система є стабільною, технічно надійною та здатною до роботи в умовах підвищеного навантаження. Обидва прототипи – як симульований, так і реальний – виконали поставлені задачі: швидкодії, передавання даних та збереження функціональної цілісності. Отримані результати створюють основу для масштабування проєкту та впровадження додаткових компонентів у майбутньому.

5 РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

Для забезпечення коректної роботи розробленої програмної системи користувач має виконати базове налаштування, яке включає підключення до MQTT-брокера та, за потреби, конфігурацію параметрів мережі пристрою. Усі дії виконуються через клієнтський застосунок, який надає графічний інтерфейс з доступом до основного функціоналу керування. Керування здійснюється за допомогою кнопок, випадаючих списків або текстових полів, залежно від реалізованих функцій пристрою. Усі запити передаються через MQTT-протокол, а відповіді автоматично відображаються в інтерфейсі. Застосунок також реалізує логування подій, що дозволяє відстежувати історію змін і діагностувати можливі збої у роботі.

5.1 Системні вимоги

Для забезпечення безперебійної та ефективної роботи клієнтського застосунку, розробленого на базі платформи Windows із використанням технології WPF, було визначено комплекс програмно-апаратних вимог. Ці вимоги відповідають мінімальним параметрам, необхідним для коректної та безперебійної роботи програми:

– операційна система: Windows 7;

– процесор: двох’ядерний 1 GHz;

– оперативна пам’ять: 2 Гігабайти;

– вільне місце на жорсткому диску: 200 Мегабайт;

– дисплей: 1366x768 пікселів;

– платформа .NET Framework: версія 4.7.2;

– наявність стабільного інтернет-з’єднання.

5.2 Огляд програмної системи

Після запуску застосунку користувач потрапляє на головне (вітальне) вікно, яке є початковим інтерфейсом взаємодії (рисунок 5.1). Це вікно містить коротке привітання, що підкреслює призначення системи, а також два основні елементи навігації: кнопки “Увійти” (Sign In) та “Зареєструватись” (Sign On). Вони забезпечують перехід до відповідних форм для автентифікації або створення нового облікового запису користувача.

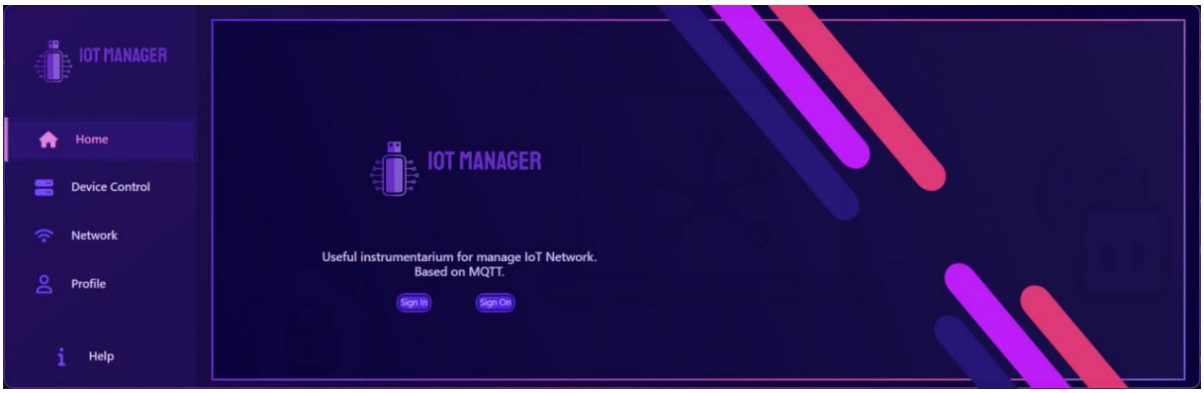

Рисунок 5.1 – Головне вікно програми

Після вибору однієї з опцій користувач взаємодіє з формами введення даних (рисунок 5.2): при вході необхідно ввести логін та пароль, а під час реєстрації – вказати Логін, за потреби, вказати додаткову інформацію.

Рисунок 5.2 – Вікно авторизації

У разі успішної авторизації користувач переходить до меню профілю, де відображається базова інформація про його обліковий запис (рисунок 5.3). Інтерфейс структуровано у вигляді окремих блоків, кожне з яких відображає певну корисну інформацію у системі. Одним із ключових є блок підключених пристроїв, яке демонструє кількість IoT-пристроїв, що наразі відслідковуються системою. Це дозволяє користувачу оперативно оцінити масштаб активного контролю.

Важливим компонентом є блок підключення до брокера, де в режимі реального часу відображається статус з'єднання із сервером MQTT. У випадку успішного з'єднання з'являється відповідний індикатор та позначка "Online", що свідчить про готовність системи до обміну даними.

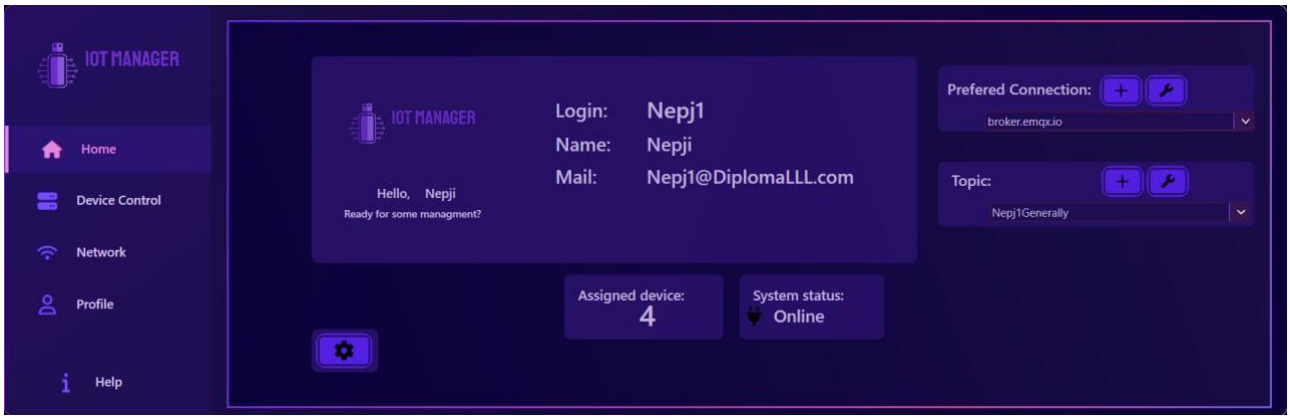

Рисунок 5.3 – Вікно профілю користувача

У межах вікна профілю особливе місце займає блок налаштування підключення, що зображено на рисунку 5.4, який виконує функцію конфігурації параметрів з'єднання з MQTT-брокером. Цей блок є важливим елементом забезпечення стабільного і безпечного з'єднання між клієнтським застосунком і сервером обміну повідомленнями. Його призначення полягає у збиранні та збереженні ключових даних, необхідних для встановлення мережевого зв'язку, що дає змогу користувачу самостійно вносити зміни у випадку зміни сервера або облікових даних.

Інтерфейс блоку передбачає чотири основні поля: BrokerURL – адреса

сервера MQTT, з яким встановлюється з’єднання; ClientID – унікальний ідентифікатор клієнта, що дозволяє брокеру розпізнати пристрій у мережі; User – логін користувача, під іменем якого виконується підключення; та Password – пароль для автентифікації. Всі поля є доступними для редагування, завдяки цьому блокові користувач має змогу вручну налаштувати з’єднання відповідно до заданих параметрів.

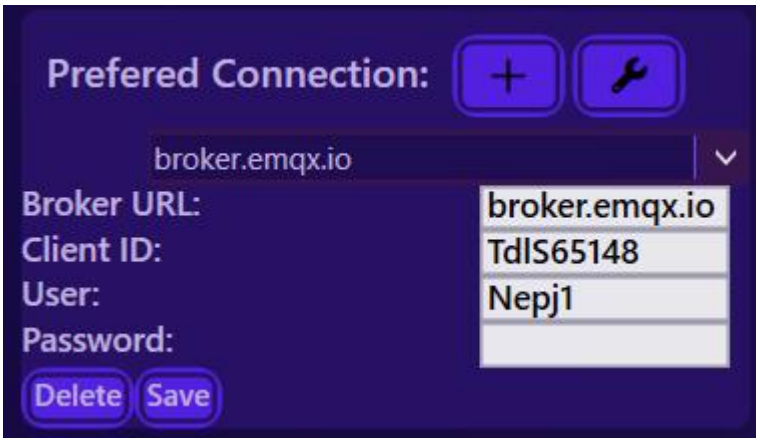

Рисунок 5.4 – Блок налаштування з’єднання

Крім параметрів з’єднання, у вікні профілю присутній блок налаштування теми підключення (рисунок 5.5), який дозволяє змінювати назву топіку (теми MQTT). Це поле слугує для визначення каналу, через який здійснюється обмін повідомленнями між клієнтом і IoT-пристроєм. Зміна назви топіку надає користувачу можливість гнучко організовувати структуру передавання даних у межах мережі відповідно до власних потреб або конфігурації пристроїв.

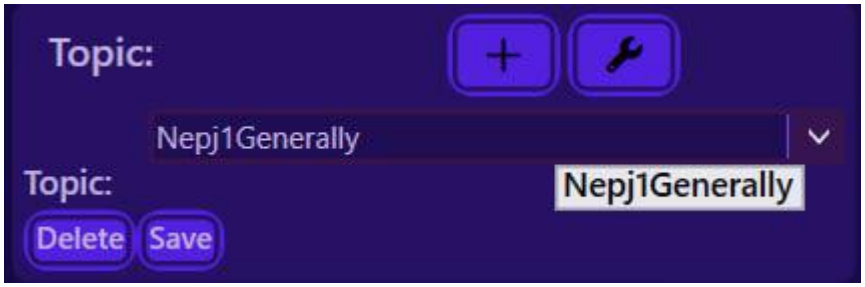

Рисунок 5.5 – Блок налаштування теми брокеру

Наступним ключовим елементом системи є вікно мережі зображене на рисунку 5.6, яке надає користувачеві зручний інтерфейс для перегляду доступних IoT-пристроїв. У центрі вікна розміщена таблиця, де перелічені виявлені пристрої – тобто ті, що надіслали хоча б одне повідомлення у відповідному топіку. Для кожного пристрою відображається назва, тема підключення (топік) та опис, що дозволяє користувачу швидко ідентифікувати пристрій і зрозуміти його функціональне призначення. Такий підхід забезпечує динамічне оновлення списку та дозволяє бачити лише активні елементи мережі.

У верхній частині вікна розміщено інструменти для зручної навігації: пошуковий рядок зліва дає змогу здійснювати фільтрацію пристроїв за ключовими словами, а справа знаходяться дві керуючі кнопки. Перша кнопка дозволяє додати пристрій до списку призначених пристроїв, тим самим виводячи його до основного інтерфейсу керування. Друга – відкриває налаштування самого вікна мережі. Додатково, у нижньому правому куті вікна постійно відображається активний топік, через який у даний момент здійснюється обмін повідомленнями, що дозволяє користувачеві контролювати стан підключення в режимі реального часу.

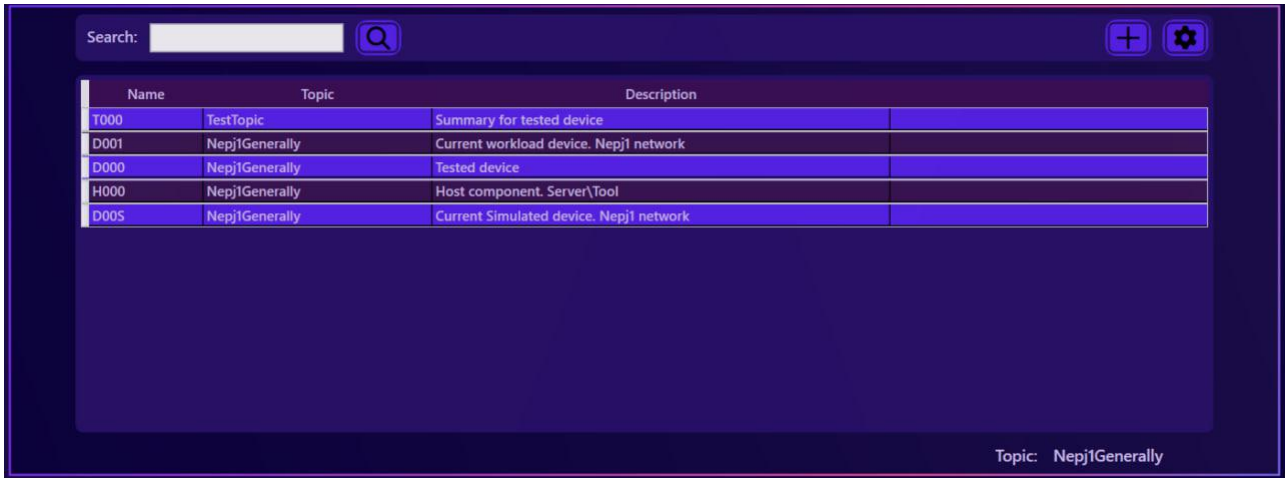

Рисунок 5.6 – Вікно мережі IoT пристроїв

Після перегляду вікна мережі користувач може перейти до вікна управління пристроями, що знаходиться на рисунку 5.7, яке забезпечує повноцінний моніторинг і контроль над призначеними IoT-пристроями. Інтерфейс цього вікна

структуровано за функціональними блоками для забезпечення максимальної зручності та логіки взаємодії. Першим елементом є налаштування частоти перевірки підключення, що дозволяє користувачу задати інтервал, з яким система здійснює опитування обраного пристрою щодо його актуального стану.

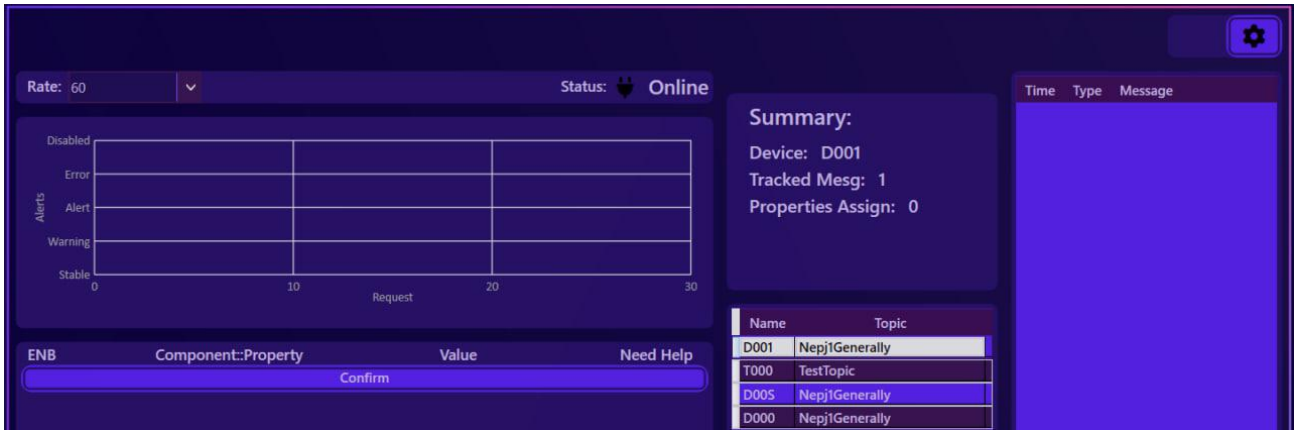

Рисунок 5.7 – Вікно управління пристроями

Центральне місце в інтерфейсі займає графік поточного пристрою, який візуально відображає історію надходження повідомлень від обраного пристрою (рисунок 5.8). На графіку представлені п’ять основних поділок: Stable (стабільне з’єднання), Warning (увага), Alert (тривога), Error (помилка) та Disabled (вимкнено). Це дозволяє користувачу швидко оцінити загальний стан пристрою та його активність.

Для забезпечення зручного й оптимального моніторингу графік обмежено до 30 останніх повідомлень, що дозволяє бачити поточну динаміку.

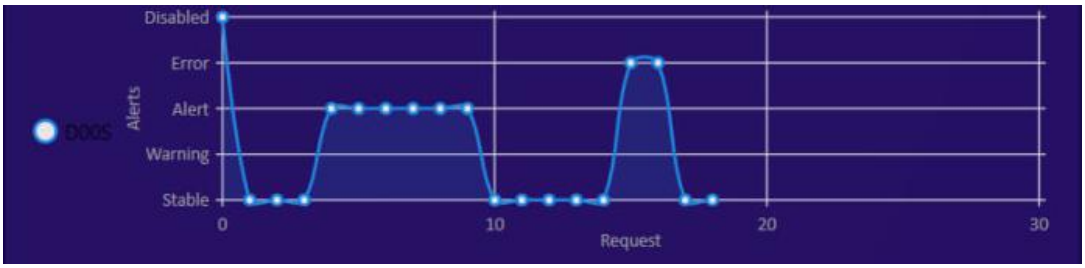

Рисунок 5.8 – Графік стану пристрою

Окрему функціональність представляє блок керування параметрами пристроїв, де відображаються всі призначені параметри обраного пристрою. З блоком можна ознайомитись на рисунку 5.9. Кожен параметр може бути індивідуально вимкнений, змінений вручну або налаштований через спеціальну функцію – запит на допомогу, що надсилає команду до довідкового модуля пристрою для отримання пояснення або рекомендацій щодо його призначення та впливу.

Такий механізм підвищує зручність у користуванні, дозволяючи уникнути помилок під час конфігурації, особливо для новачків або в складних сценаріях застосування. Після внесення змін інтерфейс вимагає підтвердження – натискання кнопки “Підтвердити” (Confirm), що забезпечує цілеспрямоване збереження налаштувань та запобігає випадковому редагуванню. Поруч розміщено блок керування обраним пристроєм, де перелічено всі призначені пристрої. Користувач може швидко перемикатися між ними, обираючи один для активного моніторингу.

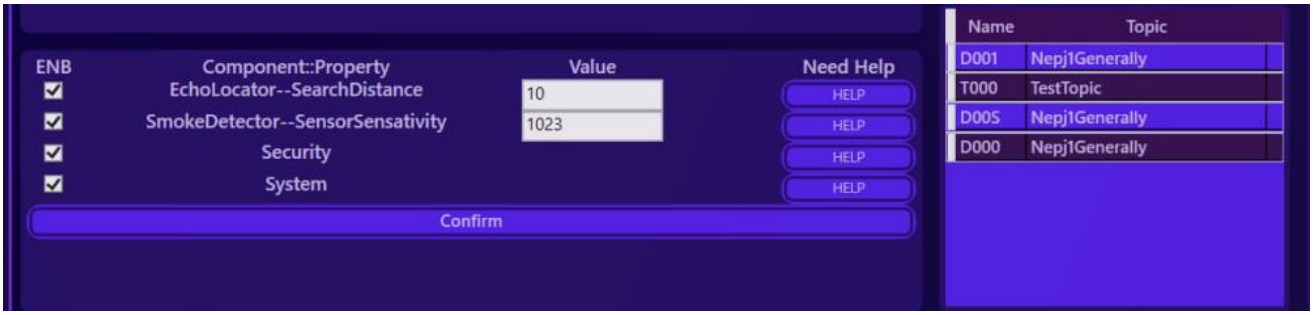

Рисунок 5.9 – Блоки параметрів та списку призначених пристроїв

Доповнює інтерфейс блок інформації про обраний пристрій (рисунок 5.10), який надає базові відомості – такі як назва, тип пристрою, ідентифікатор та активний топік.

Останній компонент – блок історії повідомлень, що зберігає весь журнал вхідних повідомлень від моменту авторизації користувача в системі. Для кожного запису зазначається час отримання, тип повідомлення (наприклад, Response, Warning, Error, Display) та вміст повідомлення, що дає змогу здійснювати повний аудит та аналіз роботи пристрою протягом поточної сесії.

Розроблений додаток для дистанційного керування IoT-пристроями на базі Wi-Fi технології забезпечує зручну, функціональну та зручну взаємодію з мережею розумних пристроїв.

Система поєднує інтуїтивну навігацію, адаптивність до різних середовищ та можливість розширення функціоналу відповідно до потреб користувача.

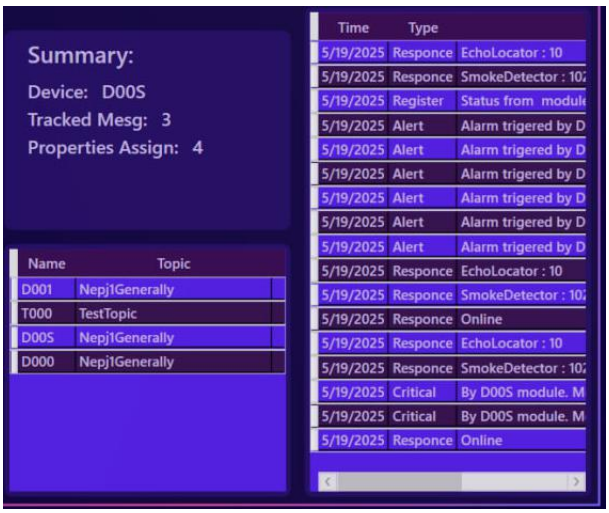

Рисунок 5.10 – Блоки інформації та історії пристрою

Додаток включає в себе всі необхідні елементи для повноцінної роботи з IoT-пристроями: авторизацію користувача, підключення до брокера, налаштування параметрів, моніторинг стану пристроїв та управління повідомленнями. Завдяки наочному відображенню інформації, гнучкому налаштуванню з’єднань і зручному управлінню параметрами пристроїв, система дозволяє забезпечити надійний контроль за функціонуванням IoT-інфраструктури навіть для користувачів без глибокої технічної підготовки.

ВИСНОВКИ

У результаті виконання дипломної роботи було досягнуто поставлену мету – реалізовано систему дистанційного керування IoT-пристроєм з використанням технології Wi-Fi та протоколу MQTT. Розроблена система дозволяє здійснювати ефективний обмін повідомленнями між клієнтським застосунком і IoT-пристроєм на базі мікроконтролера сімейства Arduino у режимі реального часу. Застосовано сучасні інструменти та середовища розробки, що забезпечило швидку реалізацію, модульність системи та її придатність до масштабування.

Аналіз обраних технологій показав переваги використання MQTT як легковагого та надійного протоколу для обміну даними в системах з обмеженими ресурсами. Застосування мікроконтролера Arduino дало змогу реалізувати енергоефективний IoT-пристрій із підтримкою бездротового з’єднання, а інтеграція із клієнтським застосунком забезпечила користувачу зручний інтерфейс для керування системою.

Результатом роботи стала система, створена для універсального підходу до організації комунікації між IoT-пристроєм і кінцевим користувачем. Розроблений застосунок може бути використаний у різних умовах. Структура системи дозволяє масштабувати її шляхом додавання нових пристроїв, розширення логіки керування або інтеграції з аналітичними платформами.

Технічна реалізація системи включає окремі компоненти, що виконують конкретні функції: обробка MQTT-повідомлень, управління пристроєм, візуалізація даних, зберігання інформації у базі даних. Завдяки розділеній архітектурі, забезпечено легкість супроводу та можливість подальшого вдосконалення. Налагодження системи відбувається у стандартному середовищі Windows.

Подальший розвиток системи може передбачати розширення підтримки апаратних платформ, реалізацію мобільного або веб-інтерфейсу, а також впровадження механізмів автоматичного реагування на події.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. State of IoT 2024: Number of connected IoT devices growing. IoT Analytics. URL: https://iot-analytics.com/number-connected-iot-devices/ (date of access: 17.05.2025).

2. Kale J. ESP8266 nodemcu using arduino IDE: getting start with ESP8266. Independently Published, 2018.

3. Thorpe E. Arduino: simple and effective strategies to arduino programming. Independently Published, 2019.

4. Hillar G. C. MQTT essentials - A lightweight iot protocol. Packt Publishing, 2017. 280 p.

5. Litvinavicius T. Exploring windows presentation foundation. Berkeley, CA : Apress, 2021. URL: https://doi.org/10.1007/978-1-4842-6637-3 (date of access: 27.05.2025).

6. Espressif Systems. ESP32 technical reference manual. Espressif Systems. Version 4.1. 2023. URL: https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf (date of access: 19.05.2025).

7. Information technology - telecommunications and information exchange between systems - local and metropolitan area networks. / ed. by IEEE Computer Society. LAN/MAN Standards Committee. et al. New York : Institute of Electrical and Electronics Engineers, 2001. 31 p.

8. Руденко В. Дистанційне керування пристроями iot на базі wifi технології. Сучасні проблеми наукового забезпечення енергетики : Матеріали XXII Міжнар. науково-практ. конф. молодих вчен. і студентів, м. Київ, 22 квіт. 2025 р. Київ, 2025. С. 256–257.

9. Hillar G. C. MQTT essentials - A lightweight iot protocol. Packt Publishing, 2017. 280 p.

10. Internet qos. Elsevier, 2001. URL: https://doi.org/10.1016/b978-1-55860-608-1.x5000-5 (date of access: 22.05.2025).

11. Learn WPF MVVM - XAML, C# and the MVVM pattern. LULU, 2017. 174 p.

12. Model-View-ViewModel - .NET documentation. Microsoft Learn. URL: https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm (date of access: 23.05.2025).

13. Kanagachidambaresan G. R. Home Electrification and Node-RED. Internet of things using single board computers. Berkeley, CA, 2022. P. 201–208. URL: https://doi.org/10.1007/978-1-4842-8108-6_6 (date of access: 27.05.2025).

14. IoT platform product architecture on google cloud | cloud architecture center. Google Cloud. URL: https://cloud.google.com/architecture/connected-devices/iot-platform-product-architecture (date of access: 26.05.2025).

15. AWS iot core documentation. amazon. URL: https://docs.aws.amazon.com/iot/ (date of access: 26.05.2025).

16. Mqttx documentation. mqttx.app. URL: https://mqttx.app/docs/advanced (date of access: 26.05.2025).

17. Visual Studio documentation. Microsoft Learn. URL: https://learn.microsoft.com/en-us/visualstudio/windows/?view=vs-2022 (date of access: 26.05.2025).

18. Brown P. Windows Store App Development : C# and XAML: C# and XAML. Manning Publications Co. LLC, 2013.

19. Chacon S. Pro git. Berkeley, CA : Apress, 2009. URL: https://doi.org/10.1007/978-1-4302-1834-0 (date of access: 27.05.2025).

20. Microsoft Corporation. Microsoft SQL server introduction: SQL server, including OLAP services. Redmond, Wash : Microsoft Press, 1998. 565 p.

21. Bassett L. Introduction to javascript object notation: a to-the-point guide to json. O'Reilly Media, Incorporated, 2015.

22. PlatformIO documentation. PlatformIO. URL: https://docs.platformio.org/en/stable/home/index.html (date of access: 25.05.2025).

23. Wokwi documentation. Wokwi Docs. URL: https://docs.wokwi.com (date of access: 27.05.2025).

Додаток А

Реалізація обміну інформації через MQTT-брокер

Програмні засоби:

– мова програмування – C#;

– база даних – MSSQL;

– взаємодія з протоколом MQTT – бібліотека MQTTNet;

Код реалізації десеріалізації вхідного повідомлення:

```
private void HandleMessageReceived(MqttApplicationMessage
applicationMessage)
{
   String topic = applicationMessage.Topic;
   String payload = Encoding.UTF8.GetString(applicationMessage.Payload);
   if (!payload.Contains(CurrentDevice) && !payload.Contains("AH"))
   {
      return;
   }

   Logger.WriteLog(ELogVerbosity.Log, $"Received. On topic: {topic}.
Message: {payload}");
   MQTTMessage? ReceivedMessageDeserialized =
JsonSerializer.Deserialize<MQTTMessage>(payload);
   if (ReceivedMessageDeserialized == null)
   {
      Logger.WriteLog(ELogVerbosity.Error, $"JSON Parse failed.");
      return;
   }
   MQTTMessangeHendle MessangeHendle = new MQTTMessangeHendle()
   {
      Topic = applicationMessage.Topic,
```

```
            Session =
DataManager.Get().GetSession(DataManager.Get().GetCurrectSession().ConnectionSet
tingId),
            Message = ReceivedMessageDeserialized
        };

        DeviceManager.Get().ReceiveMessage(MessangeHendle);
    }
```

Код реалізації десеріалізації вхідного повідомлення:

```
    public void MessageUpload(String Target, EDeviceSendMessage Message,
EDeviceComponentTarget Type, String Value = "")
    {

        MQTTMessage Uploader = new MQTTMessage
        {
            ID = CurrentDevice,
            Target = Target.Trim(),
            Message = Message.ToString().Trim(),
            TargetComponent = Type.ToString().Trim(),
            Value = Value.Trim(),
            Objects = null

        };
        String UploaderJson = JsonSerializer.Serialize(Uploader);

        MqttApplicationMessageBuilder UploadMessage = new
MqttApplicationMessageBuilder();
        UploadMessage.WithPayload(UploaderJson);
        UploadMessage.WithTopic(MQTTTopic.Topic);
```

```
            Logger.WriteLog(ELogVerbosity.Log, $"To send. On topic:
{MQTTTopic.Topic}. Message: {UploaderJson}");

            if (!IsConnected())
            {
                Logger.WriteLog(ELogVerbosity.Error, "Message upload was delayed...
MQTTManager::MessageUpload");
                return;
            }

            Task.Run(async () =>
            {
                OnReInit();
                await MQTTClient.PublishAsync(UploadMessage.Build());
            });
        }
```

Код обробки вхідного повідомлення:

```
        internal void ReceiveMessage(MQTTMessangeHendle? Handle)
        {
            if (Handle == null)
            {
                Logger.WriteLog(ELogVerbosity.Warning, $"MQTTManager: Received
empty handle");
                return;
            }
            MQTTMessage receivedMessage = Handle.Message;
            if (receivedMessage == null)
            {
                Logger.WriteLog(ELogVerbosity.Warning, $"MQTTManager: Received
empty message");
```

```
            return;
        }
        LocatedDevice? sender =
DataManager.GetLocatedDevice(receivedMessage.ID);
        if (sender == null)
        {
            if (receivedMessage.ID == null)
            {
                Logger.WriteLog(ELogVerbosity.Error, $"MQTTManager: Received
empty ID.");
                return;
            }
            sender = DataManager.AddNewDevice(receivedMessage.ID, Handle.Topic);
            if (sender == null)
            {
                Logger.WriteLog(ELogVerbosity.Error, $"MQTTManager: Located new
device, but failure to add");
                return;
            }
            RequestData(sender);
        }
        sender.Name = sender.Name.Trim();

        RequestLocalDataBaseUpdate();
        if (!LocalDeviceDataStorage.ContainsKey(sender.Name.Trim()))
        {
            Logger.WriteLog(ELogVerbosity.Error, $"MQTTManager: Device input
was ignored. Not Tracked.");
            return;
        }
```

```
DataTool.TryGetEnumFromString<EResponceReceived>(receivedMessage.Message,
out EResponceReceived DeviceResponce);


        if (DeviceResponce == EResponceReceived.Responce)
        {
            ProcessDeviceResponce(receivedMessage, sender);
        }
        else if (DeviceResponce == EResponceReceived.Alert)
        {
            ProcessAlertResponce(receivedMessage, sender);
        }
        else if (DeviceResponce == EResponceReceived.Critical)
        {
            ProcessErrorResponce(receivedMessage, sender);
        }
        else if (DeviceResponce == EResponceReceived.Register)
        {
            ProcessRegisterResponce(receivedMessage, sender);
        }
        else if (DeviceResponce == EResponceReceived.Display)
        {
            ProcessDisplayResponce(receivedMessage, sender);
        }
        else
        {
            NewDeviceDataStorageRecord(sender.Name.Trim(),
receivedMessage.Message.ToString().Trim(),
                $"Received unknown responce: {receivedMessage.Message}");
            Logger.WriteLog(ELogVerbosity.Warning, $"DeviceManager: Received
```

```
unknwon message.\nResponce: {DeviceResponce}\nMessage:
{receivedMessage.Message}\nValue: {receivedMessage.Value}");
            }

            Messenger.Default.Send(new UpdateDataRequest());
        }
```

Код підготовки повідомлення з запитом на зміну параметрів:

```
public void RequestDeviceDataUpdate(ParametersExtensions propertyMesanger,
LocatedDevice? Device)
        {
        if (Device == null)
        {
            Logger.WriteLog(ELogVerbosity.Warning, $"MQTTManager: Property
update to device with NULL.");
            return;
        }
        LocalDeviceHistory? target;
        try
        {
            target = LocalDeviceDataStorage [Device.Name.ToString()];
        }
        catch (Exception ex)
        {
            Logger.WriteLog(ELogVerbosity.Warning, $"MQTTManager: Ghost
Device.");
            Noti.Get().ShowError("Device was not logedIn yet.");
            return;
        }
        if (target == null || target.Parameters == null)
        {
```

```
            Logger.WriteLog(ELogVerbosity.Warning, $"MQTTManager: Device doesnt
exist in LocalContext.");
            return;
        }

        DataTool.TrimStringProperties(propertyMesanger);

        List<PropertyDictionaryDifference> Propdifference =
DataTool.GetDictionaryDifference(target.Parameters.ParamsDicitonary,
propertyMesanger.ParamsDicitonary);

        //Upload Local
        target.Parameters.ParamsDicitonary = new Dictionary<string,
ParametersExtensionsHandler>(propertyMesanger.ParamsDicitonary);

        foreach (PropertyDictionaryDifference CurrentProp in Propdifference)
        {
            if (CurrentProp.NewValue == null)
            {
                continue;
            }
            ParametersExtensionsHandler Handler = CurrentProp.NewValue;

DataTool.TryGetEnumFromString<EDeviceComponentTarget>(Handler.ComponentNa
me, out EDeviceComponentTarget componentTarget);
            if (CurrentProp.OldValue?.Value != CurrentProp.NewValue.Value)
            {
                MQTTManager.MessageUpload(Device.Name, EDeviceSendMessage.SET,
componentTarget, Handler.Value);
            }
            if (CurrentProp.OldValue?.bIsActive != CurrentProp.NewValue.bIsActive)
```

```
            {
                EDeviceSendMessage Addmessage = Handler.bIsActive ?
EDeviceSendMessage.TurnON : EDeviceSendMessage.TurnOFF;
                MQTTManager.MessageUpload(Device.Name, Addmessage,
componentTarget, "");
            }
            if (CurrentProp.OldValue?.HelpCommandTrigger !=
CurrentProp.NewValue.HelpCommandTrigger)
            {
                MQTTManager.MessageUpload(Device.Name, EDeviceSendMessage.Help,
componentTarget, "");
            }
        }
        }
```

Додаток Б

Апробація роботи

УДК 004.5

[1] Бакалаврант 4 курсу Руденко В.І.; [1] Бакалаврант 4 курсу Імаєва Т.А.
[1] Проф., д.т.н. Отрох С.І.
https://scholar.google.com.ua/citations?hl=uk&user=Aye6WfAAAAAJ

[1] КПІ ім. Ігоря Сікорського

ДИСТАНЦІЙНЕ КЕРУВАННЯ ПРИСТРОЯМИ IOT НА БАЗІ WIFI ТЕХНОЛОГІЇ

Постановка проблеми та її актуальність. В умовах розвитку технологій Інтернету речей (IoT) виникає проблема створення ефективних методів дистанційного керування пристроями, що забезпечують якісну комунікацію та безпечний обмін даними. Зазвичай, традиційні системи автоматизації технічно обмежені, через фізичну прив'язку контролера до інфраструктури, що ускладнює масштабування системи та підвищує витрати на обслуговування.

Технологія Wi-Fi спроможна забезпечити виконання поставлених задач з більшою ефективністю. Включає в собі якісні переваги, такі як: доступність, енергоефективність та надійність. Ідея використання даної технології дозволяє інтегрувати різноманітні IoT пристрої в одній мережі з використанням меншої кількості ресурсів, що також зменшує ціну обслуговування даної системи. Додатковою та головною перевагою є можливість дистанційного керування пристроями, що значно розширює доступний функціонал системи.

Однак подібні системи мають недоліки, що проявляються у вигляді необхідності розробки продуманої системи комунікацій між приладами, системи що включає в себе рішення ефективної передачі даних та необхідності оптимізації процесів обробки. Крім того, оптимізована обробка процесів є критичною необхідністю в даних типах системах, оскільки велика кількість пристроїв може генерувати значні обсяги трафіку, що потребують більш ефективного підходу оперування.

Аналіз останніх досліджень. Дослідження у сфері керування IoT-пристроями з використанням Wi-Fi технологій часто зосередженні на розробці трьох напрямів: протоколи зв'язку для взаємодії пристроїв в системі, засоби інформаційної безпеки та оптимізації мережевих архітектур.

На кожен напрямок існує прогресивний лідер, що сфокусований на власних універсальних рішеннях. Прикладом таких представників будуть: OASIS (розробка стандарту MQTT) [2], Cisco (виявлення аномалій в мережі для підвищення безпеки), [3] Huawei (дослідження автономності систем) [1]. Але через особливості рішень виникає необхідність створення архітектури, яка спроможна об'єднати можливості різних напрямів в одне ціле. Це особливо актуально для великих розподілених мереж IoT, де стабільність передачі даних та енергоефективність є ключовими факторами успішної роботи системи.

Формулювання мети. Метою дослідження є розробка системи керування IoT пристроями за допомогою технології Wi-Fi, що має забезпечити надійну комунікацію, обмін даними та обробкою інформації. Робота передбачає використання сучасних протоколів зв'язку, зокрема MQTT. Розроблена система має бути оптимізованою стосовно енергоспоживання, задовольняти потреби стабільної передачі даних та підготовленою до гнучкого використання, розширивши можливості взаємодії пристроїв.

Основна частина. Керування пристроями IoT на базі Wi-Fi потребує для своєї роботи ефективної комунікації між пристроями та безпечного обміну даними. Використання MQTT протоколу задовольняє дані потреби та дозволяє реалізувати взаємодію між пристроями, брокером та клієнтською частиною [2]. MQTT працює за принципом "видавець-підписник", що дозволяє забезпечити високу швидкість передачі даних та гнучке керування пристроями Основні процеси, які виконує система включають в

себе прийом та виконання пристроєм зазначених команд та передавання інформації, отриманої в ході роботи.

На Рисунку 1 зображено спроектовану архітектуру взаємодії між основними компонентами системи. Користувач має можливість керувати пристроєм через додаток за допомогою запитів, які надсилаються на брокер-сервер. Брокер натомість надсилає інформацію до підключених пристроїв, що виконують певні функції в системі та за необхідності передавати зворотну відповідь у вигляді інформації з периферії. Подібна система спроможна забезпечити базовий двосторонній зв'язок та передачу інформації між компонентами.

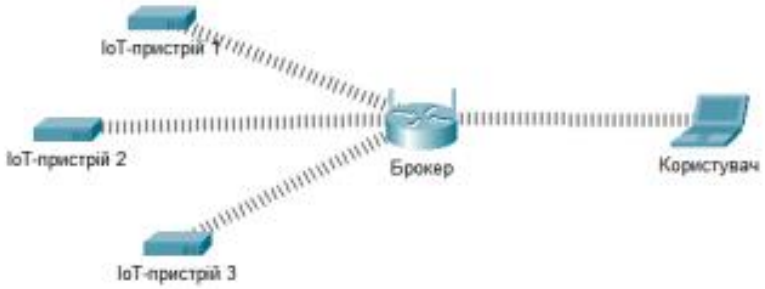

Рисунок 1 Модель взаємодії IoT-пристроїв з системою

Система пристрою розробляється на базі мікроконтролера, який попри недоліки потужності має переваги у вигляді простоти використання, енергоефективності та мобільності [4]. Завдяки своїм характеристикам такий пристрій підходить до реалізації проектів IoT, оскільки дозволяє забезпечити роботу з використанням мінімум енергетичних ресурсів.

Такий пристрій підключений до MQTT-брокера, що представляє розгорнутий публічний сервер, такий підхід дозволяє передавати дані з мінімальними витратами енергії [2]. Взаємодія між кінцевими приладами, що включає в себе IoT-пристрій та додаток користувача, відбувається за допомогою запитів в JSON форматі. Такий підхід дозволяє інтегрувати пристрої із зовнішніми сервісами, наприклад, хмарними платформами для збору та аналізу даних [4].

Автоматизація керування пристроями можлива завдяки заздалегідь сформованим сценаріям, які надаються користувачеві для вибору. Це дозволяє адаптувати систему під конкретні потреби замовника, забезпечуючи гнучке налаштування поведінки пристроїв залежно від заданих умов або тригерів.

Висновки. Розроблена система керування IoT-пристроями на базі Wi-Fi технології дозволяє здійснювати ефективний та безпечний обмін даними між мікроконтролерами, брокером та клієнтським додатком. Ключовими факторами є реалізація механізму надійної передачі даних та комунікацій. Впроваджені рішення допоможуть оптимізувати енергоспоживання в системі та підвищити рівень обслуговування.

Перелік посилань:

1. Dang W., Huang R. Autonomous Driving Network: Network Architecture in the ERA of Autonomy. CRC Press LLC, 2024.

2. Shah P. H. MQTT Systems: A Survey. International Journal for Research in Applied Science and Engineering Technology. 2024. Vol. 12, no. 3. P. 1063–1067. URL: https://doi.org/10.22214/ijraset.2024.59000 (date of access: 03.03.2025).

3. Network management fundamentals / ed. by C. S. Inc. Indianapolis, IN : Cisco Press, 2007. 510 p.

4. Herrero R. Fundamentals of IoT Communication Technologies. Cham : Springer International Publishing, 2022. URL: https://doi.org/10.1007/978-3-030-70080-5.