Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «МАГІСТР»

СТВОРЕННЯ ТА ДОСЛІДЖЕННЯ МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ ВІДДАЛЕНОГО КЕРУВАННЯ РОЗУМНИМ БУДИНКОМ

CREATION AND RESEARCH OF A MOBILE APPLICATION FOR REMOTE CONTROL OF A SMART HOME

спеціальність 121 «Інженерія програмного забезпечення»
освітня програма «Інженерія програмного забезпечення»

Виконав: здобувач вищої освіти
групи ІПЗм-21
Ключук М. С.
Керівник:
к.ф.-м.н., доцент
Хвищун М. В.

Кваліфікаційну роботу
допущено до захисту
«___» _____ 20__ р.
Гарант освітньої програми:
к.т.н., доцент Суринович О. М.
_____

Луцьк – 2025 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *інженерії програмного забезпечення*
Ступінь вищої освіти *магістр*
Галузь знань: *12 «Інформаційні технології»*
Спеціальність: *121 «Інженерія програмного забезпечення»*
Освітня програма: «*Інженерія програмного забезпечення»*

ЗАТВЕРДЖУЮ
Завідувач кафедри

«___» _____ 202__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА

ДРУГОГО (МАГІСТЕРСЬКОГО) РІВНЯ ВИЩОЇ ОСВІТИ

Ключуку Максиму Сергійовичу

1. Тема кваліфікаційної роботи: Створення та дослідження мобільного застосунку для віддаленого керування розумним будинком
Керівник роботи: Хвищун Микола Вячеславович, доцент, к.ф.-м.н.
затверджені наказом закладу вищої освіти від «29» березня 2025 року № 190/01-02
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи: 4 грудня 2025 р.
3. Вихідні дані до роботи технічне та програмне забезпечення ЕОМ
4. Зміст розрахунково-пояснювальної записки: аналіз сучасних підходів до систем віддаленого керування розумним будинком та визначення функціональних вимог; обґрунтування вибору технологій, протоколів зв'язку та архітектури програмного забезпечення; проектування і реалізація мобільного застосунку для керування IoT-вузлами; експериментальне дослідження швидкодії, стабільності та ефективності роботи мобільного застосунку.
5. Перелік графічного матеріалу: 24 рисунки, 13 таблиць, 1 додаток.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз проблеми за темою роботи та постановка завдань дослідження	*Хвищун М. В.*		
Теоретичне дослідження та практична реалізація	*Хвищун М. В.*		
Експериментальне дослідження системи	*Хвищун М. В.*		
Нормоконтроль	*Повстяна Ю. С.*		
Гарант ОП	*Андрущак І. Є.*		
Показник запозичень тексту	___%		
Академічна доброчесність	*Хвищун М. В.*		

7. Дата видачі завдання *«02 квітня 2025 р.*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи магістра	Строк виконання етапів роботи	Примітка
1	Провести огляд літературних джерел по темі кваліфікаційної роботи	02.05.2025	
2	Провести аналіз загальної проблеми і вибір напрямків дослідження	24.09.2025	
3	Розробити функціональну модель та архітектуру системи	01.11.2025	
4	Описати засоби розробки об'єкта проектування	19.11.2025	
5	Практична реалізація об'єкта проектування	26.11.2025	
6	Розробити методику для проведення експерименту	05.11.2025	
7	Провести аналіз результатів експерименту	15.11.2025	
8	Здача чистового варіанту кваліфікаційної роботи на кафедру	04.12.2025	

Здобувач вищої освіти _____ Ключук М. С.

Керівник кваліфікаційної роботи _____ Хвищун М. В.

АНОТАЦІЯ

Ключук М. С. Створення та дослідження мобільного застосунку для віддаленого керування розумним будинком. Рукопис.

Кваліфікаційна робота магістра ОП «Інженерія програмного забезпечення» спеціальності 121 «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2025.

Кваліфікаційна робота магістра складається з вступу, 3 розділів, висновків, списку використаних джерел та додатку.

У першому розділі наведено огляд предметної області систем «розумного будинку», проаналізовано сучасні підходи до організації віддаленого керування IoT-пристроями та визначено вимоги до мобільних застосунків і серверної інфраструктури. У другому розділі описано процес розроблення мобільного застосунку, обґрунтовано вибір технологій, архітектури та протоколів зв'язку, а також представлено реалізацію основних функціональних модулів системи керування. У третьому розділі подано результати експериментальних досліджень, проведено тестування швидкодії, стабільності та надійності роботи застосунку, а також здійснено оцінку ефективності функціонування розробленої програмної системи в реальних умовах.

Ключові слова: мобільний застосунок, розумний будинок, IoT, віддалене керування, REST API, MQTT.

ABSTRACT

Kliuchuk M. Creation and Research of a Mobile Application for Remote Control of a Smart Home. Manuscript.

Qualification master's thesis OP «Software Engineering» specialty 121 «Software Engineering». Lutsk National Technical University. Lutsk, 2025.

Qualification master's thesis consists of an introduction, 3 chapters, conclusions, a list of sources used, and an appendix.

The first section provides an overview of the subject area of smart home systems is provided, modern approaches to organizing remote control of IoT devices are analyzed, and the requirements for mobile applications and server infrastructure are defined. The second section describes the process of developing the mobile application, substantiates the choice of technologies, architecture, and communication protocols, and presents the implementation of the main functional modules of the control system. The third section presents the results of experimental studies, performance, stability, and reliability testing of the application, as well as an evaluation of the efficiency of the developed software system under real operating conditions.

Keywords: Mobile Application, Smart Home, IoT, Remote Control, REST API, MQTT.

.

ЗМІСТ

ВСТУП

Актуальність теми дослідження. Сучасний розвиток цифрових технологій, стрімке поширення Інтернету речей (IoT) та зростання ролі мобільних платформ зумовлюють потребу у створенні інтелектуальних систем, здатних забезпечити комфорт, безпеку та енергоефективність житлового середовища. Концепція «розумного будинку» є одним із ключових напрямів цієї цифрової трансформації, оскільки об'єднує електронні пристрої, сенсори, виконавчі модулі й програмні інструменти в єдину інтегровану екосистему.

У таких умовах особливої актуальності набувають мобільні застосунки, які забезпечують зручний і швидкий доступ до функцій контролю та моніторингу. Мобільний пристрій стає центральним елементом взаємодії користувача з інфраструктурою «розумного будинку». Водночас масове впровадження IoT-пристроїв, зростання вимог до кібербезпеки, тенденція до хмарної інтеграції та розвиток Smart Living / Smart City формують потребу у нових програмних рішеннях, здатних об'єднати апаратні, мережеві та програмні компоненти в надійну і масштабовану систему

З огляду на це проблема створення мобільного застосунку для віддаленого керування компонентами «розумного будинку» є актуальною як з наукової, так і з практичної точки зору. Вона вимагає використання сучасних підходів інженерії програмного забезпечення, хмарних обчислень, архітектурних шаблонів, DevOps-технологій та механізмів IoT-взаємодії.

Метою кваліфікаційної роботи є розроблення мобільного застосунку, що забезпечує віддалене керування та моніторинг пристроїв «розумного будинку» із використанням IoT-технологій, хмарної інфраструктури та сучасних протоколів обміну даними.

Для досягнення поставленої мети необхідно вирішити такі завдання:

– провести огляд сучасних наукових і технічних підходів до організації розумних будинків та мобільних IoT-систем;

– проаналізувати архітектури клієнт – серверних застосунків та визначити вимоги до безпеки, масштабованості й продуктивності;

– обґрунтувати вибір технологій, програмних інструментів, протоколів (REST API, MQTT) та хмарних сервісів;

– розробити архітектурну модель мобільного застосунку і механізми взаємодії з апаратними контролерами;

– реалізувати прототип застосунку з підтримкою реального часу, авторизації та безпечного обміну даними;

– виконати модульне, функціональне та навантажувальне тестування;

– здійснити аналіз ефективності системи та визначити шляхи її подальшого вдосконалення.

Об'єктом дослідження є процес дистанційного керування пристроями «розумного будинку» за допомогою мобільних програмних систем.

Предметом дослідження є методи, моделі та технології створення архітектури, програмних модулів та інтерфейсу мобільного застосунку для взаємодії з IoT-пристроями через мережу Інтернет.

Очікуваним результатом виконання кваліфікаційної роботи є створення функціонального прототипу мобільного застосунку, який забезпечує віддалене керування та моніторинг пристроїв «розумного будинку», підтримує безпечний обмін даними з IoT-обладнанням, а також демонструє стабільність, зручність використання та можливість подальшого масштабування.

Наукова новизна кваліфікаційної роботи полягає у комплексному поєднанні сучасних IoT-протоколів, хмарних технологій та архітектурних моделей мобільних застосунків з механізмами безпечної інтеграції апаратних модулів у єдину систему віддаленого керування.

Практичне значення кваліфікаційної роботи полягає у створенні прототипу мобільного застосунку, який:

– забезпечує віддалений контроль елементів розумного будинку;

– підтримує моніторинг стану пристроїв у реальному часі;

– може бути масштабований під різні апаратні платформи;

– використовує механізми енергоефективного керування;

– гарантує безпеку переданих даних.

Апробація результатів дослідження. Основні результати дослідження та програмні рішення були представлені на засіданнях кафедри інженерії програмного забезпечення ЛНТУ. Окремі результати роботи були представлені у тезах доповіді: Хвищун М. В., Ключук М. С., Хвищун Д. М., Панасюк І. В., Драницький Б. О. «Створення мобільного застосунку для віддаленого керування розумним будинком» на International Scientific and Practical Conference of Young Scientists and Students «Actual Problems of Automation and Control», 27 листопада 2025 р., м. Луцьк (додаток А).

РОЗДІЛ 1
АНАЛІЗ ПРОБЛЕМАТИКИ ТА ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

1.1 Огляд і аналіз предметної області проблеми, результатів існуючих теоретичних та експериментальних досліджень

Концепція «розумного будинку» (Smart Home) передбачає інтеграцію програмних, апаратних та мережевих засобів у єдину екосистему, що забезпечує автоматизоване або віддалене керування усіма системами житлового чи виробничого приміщення. Головною ідеєю є створення комфортного, безпечного й енергоефективного середовища, де всі пристрої взаємодіють між собою за допомогою мережевих технологій та інтелектуальних алгоритмів [1].

До основних функцій системи розумного будинку слід віднести (рис. 1.1):

– керування освітленням – автоматичне вмикання / вимикання ламп, регулювання яскравості, сценарії «ніч / день»;

– контроль клімату – керування кондиціонерами, вентиляцією, опаленням;

– системи безпеки – відеоспостереження, сигналізація, детектори руху, диму, газу;

– мультимедійні системи – інтеграція аудіо – та відеопристроїв, синхронізація з голосовими асистентами;

– енергомоніторинг – вимірювання споживання електроенергії, води, газу;

– автоматизація побутових процесів – сценарії «розумного ранку», «нічного режиму», «відпустки».

Розглянемо основні компоненти системи:

– контролери (мозок системи) – мікрокомп'ютери (наприклад Raspberry Pi, ESP32, STM32) або промислові PLC-контролери, які забезпечують обробку даних і керування виконавчими пристроями;

– сенсори (давачі) – вимірюють параметри навколишнього середовища (температура, вологість, рух, освітленість, дим, газ тощо);

– актуатори (виконавчі пристрої) – виконують дії за командами системи: вмикають реле, відкривають клапани, керують електроприводами;

Рисунок 1.1 – Структурна схема системи «розумного будинку» [2]

– комунікаційні модулі – забезпечують бездротовий або дротовий обмін даними між компонентами (Wi-Fi, Bluetooth, Zigbee, Ethernet);

– програмне забезпечення (Smart Home Platform) – серверна частина, база даних і користувацький інтерфейс (мобільний або веб-застосунок) (рис. 1.2).

Рисунок 1.2 – Логічна архітектура системи Smart Home [3]

Розглянемо популярні технології автоматизації [2, 3].

Zigbee:

- працює на частоті 2,4 ГГц, використовує сітчасту (mesh) мережу;
- підтримує до 65 000 пристроїв;
- низьке енергоспоживання, велика стабільність з'єднання;
- приклади: Philips Hue, Xiaomi Smart Home, IKEA Tradfri.

Z-Wave:

- частота 868 МГц, менше перешкод, ніж у Wi-Fi;
- з'єднує до 232 пристроїв;
- висока безпека передачі даних, низька швидкість (до 100 кбіт/с);
- використовується у Fibaro, Aeotec, SmartThings.

Wi-Fi:

– найпоширеніший стандарт (2.4/5 ГГц), висока швидкість;
– просте підключення без шлюзу, але велике енергоспоживання;
– використовується у більшості сучасних ESP32/ESP8266-рішень.

Bluetooth / BLE:

- застосовується для локальних систем;
- BLE (Bluetooth Low Energy) – енергоефективний варіант для сенсорних мереж;
- підходить для систем керування освітленням, замками, фітнес-пристроями.

Для аналізу роботи вище наведених платформ розглянемо порівняльну характеристику технологій автоматизації, подану в таблиці 1.1.

Представлені платформи демонструють різні підходи до проєктування та реалізації систем «розумного будинку», що є важливим з погляду інженерії програмного забезпечення. Google Home та Amazon Alexa є прикладами повністю хмарно орієнтованих екосистем, де ключову роль відіграють сервісні API, голосові інтерфейси та масштабована хмарна інфраструктура. Для ІПЗ це важливо у контексті вивчення моделі cloud-based architecture, розподіленої обробки подій та інтеграції через REST і MQTT. Apple HomeKit демонструє інший підхід – акцент на безпеці, локальній обробці та контролі над екосистемою. Архітектура HomeKit є показовим прикладом використання

криптографічних методів, сертифікації пристроїв та концепції privacy-by-design, що є актуальним для розробників ПЗ у сфері IoT.

Таблиця 1.1 – Порівняльна характеристика технологій автоматизації [3]

Платформа	Технології зв'язку	Тип системи	Особливості	Приклади
Google Home	Wi-Fi, BLE	Хмарна	Голосове керування, інтеграція з Android	Google Nest
Amazon Alexa	Wi-Fi, Zigbee	Хмарна	Розширена підтримка сценаріїв	Echo, Ring
Apple HomeKit	Wi-Fi, BLE	Локальна/хмарна	Висока безпека, інтеграція з iOS	HomePod
Samsung SmartThings	Zigbee, Z-Wave, Wi-Fi	Гібридна	Підтримка багатьох пристроїв	SmartThings Hub
OpenHAB / Home Assistant	Zigbee, MQTT, Wi-Fi	Локальна	Open-source, налаштовується користувачем	Raspberry Pi

Samsung SmartThings є гібридною платформою, що поєднує локальну та хмарну логіку. З погляду ІПЗ, це ілюструє підхід до побудови модульних, масштабованих систем, здатних працювати в умовах гетерогенних мережевих протоколів (Wi-Fi, Zigbee, Z-Wave).

Home Assistant та OpenHAB є open-source рішеннями, орієнтованими на локальну обробку, кастомізацію та інтеграцію великої кількості пристроїв. Це робить їх показовими прикладами архітектур гнучкого розширення, багаторівневих систем конфігурації та механізмів плагінної архітектури – концепцій, що безпосередньо належать до ІПЗ.

Для кращого кількісного аналізу побудуємо порівняльну діаграму архітектури Smart Home-платформ (рис. 1.3)

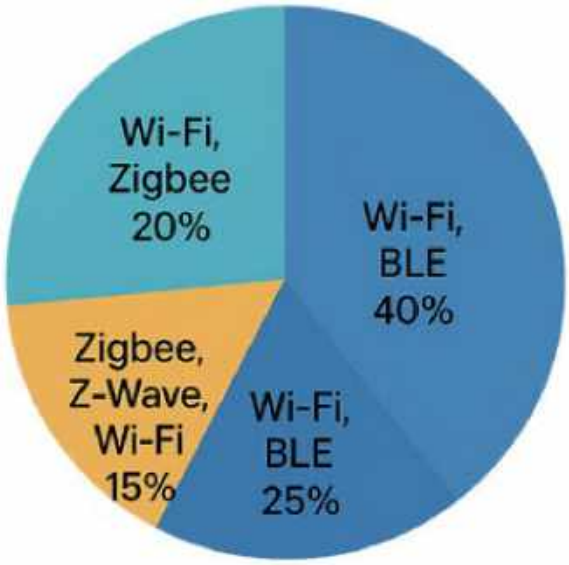

Рисунок 1.3 – Порівняльна діаграма архітектури Smart Home-платформ [3]

Діаграма відображає співвідношення основних технологій зв'язку, що використовуються в сучасних системах «розумного будинку». Найбільшу частку (40 %) займають платформи, які працюють на основі Wi-Fi та Bluetooth Low Energy (BLE), що свідчить про їхню популярність у комерційних мобільних рішеннях завдяки простоті інтеграції та підтримці більшістю смартфонів.
Другу за поширеністю групу (25 %) також становлять системи Wi-Fi + BLE, що підкреслює загальну тенденцію до універсальності та мобільності.

Технології Zigbee та Wi-Fi займають 20 %, демонструючи активне використання низькоенергетичних протоколів у масштабованих домашніх мережах.

Окрема частка (15 %) припадає на платформи, що поєднують Zigbee, Z-Wave та Wi-Fi, які, хоча й потребують спеціальних шлюзів, забезпечують високу надійність, мале енергоспоживання і здатність підтримувати велику кількість IoT-пристроїв.

Узагальнюючи, діаграма показує перевагу рішень на базі Wi-Fi та BLE, водночас підтверджуючи важливість Zigbee та Z-Wave у спеціалізованих і масштабованих системах розумного дому.

Отже, сучасні системи «розумного будинку» являють собою комплексні програмно-апаратні рішення, у яких поєднано мікропроцесорні технології, хмарні сервіси та мобільні застосунки. Серед технологій зв'язку найбільш ефективними для IoT є Zigbee, Z-Wave та BLE, тоді як Wi-Fi залишається універсальним стандартом для масових користувацьких пристроїв.

Розроблення власного мобільного застосунку для керування розумним будинком потребує урахування особливостей архітектури, вибору відповідного протоколу зв'язку, а також забезпечення сумісності з поширеними платформами (Google Home, Alexa, Home Assistant) [4].

1.2 Огляд і аналіз методів та засобів розробки програмного забезпечення для вирішення проблеми дослідження

Розвиток мобільних технологій став ключовим чинником цифрової трансформації сучасного світу. Застосунки для смартфонів охоплюють майже всі сфери діяльності – від освіти й охорони здоров'я до промислової автоматизації та систем «розумного будинку». Для інженерії програмного забезпечення важливо не лише створити функціональний застосунок, а й забезпечити його продуктивність, безпеку, сумісність і масштабованість. Тому правильний вибір мобільної платформи, мови програмування, середовища розробки та серверної архітектури є критично важливим.

Проведемо аналіз мобільних платформ (табл. 1.2). Найпоширенішими середовищами для створення мобільних застосунків є Android та iOS, а також кросплатформні рішення, які дозволяють створювати універсальні програми під обидві системи одночасно [5].

Порівняльна характеристика платформ показує, що Android, iOS та кросплатформні рішення мають різні підходи до розроблення мобільних застосунків, що впливає на архітектуру, технологічний стек та можливості інтеграції з системами «розумного будинку». Android базується на відкритій системі AOSP, підтримує Kotlin / Java та надає широкий доступ до API. Це

забезпечує високу гнучкість, широку кастомізацію та зручність інтеграції з IoT-пристроями, включно з BLE, Wi-Fi та MQTT. Платформа є більш доступною через низьку вартість публікації застосунків. iOS побудована на Unix-платформі Darwin і використовує Swift / Objective-C. Apple забезпечує високу стабільність та продуктивність, але жорсткі обмеження доступу до системних API й суворі правила екосистеми зменшують гнучкість. Вартість публікації вища, але системи безпеки – одні з найкращих у галузі. Кросплатформні рішення (Flutter, React Native) працюють через власні віртуальні машини та проміжні API. Вони дозволяють створювати один застосунок для двох систем, знижуючи витрати на розроблення. Проте продуктивність і доступ до низькорівневих можливостей залежать від конкретного фреймворку. Такі рішення добре підходять для масштабованих комерційних застосунків (Uber, Airbnb), але інколи потребують нативних модулів для роботи з апаратними IoT-компонентами.

Таблиця 1.2 – Порівняння мобільних платформ

Параметр	Android	iOS	Кросплатформні рішення
Операційна система	Linux (AOSP)	Unix (Darwin)	Власна віртуальна машина (Flutter, React Native)
Основна мова	Kotlin / Java	Swift / Objective-C	Dart, JavaScript, TypeScript
IDE	Android Studio	Xcode	VS Code, Android Studio
Вартість публікації	Одноразова ($25)	Щорічна ($99)	Залежить від платформи
Гнучкість у налаштуванні	Висока	Обмежена	Висока
Продуктивність	Висока	Висока	Залежить від середовища
Тип доступу до API	Відкритий	Закритий	Опосередкований
Параметр	Android	iOS	Кросплатформні рішення
Приклади	Google Home, Blynk	Apple Home, HomeKit	Spotify, Airbnb, Uber

Проведений аналіз показує, що вибір платформи залежить від пріоритетів проєкту:

– Android – оптимальний для систем «розумного будинку», де потрібна гнучка інтеграція, доступ до сенсорів, розширені API та підтримка IoT-протоколів;

– iOS – підходить для застосунків із підвищеними вимогами до безпеки та стабільності, але має обмеження у кастомізації;

– кросплатформні фреймворки – ефективні для швидкої розробки та підтримки великої аудиторії користувачів, проте можуть вимагати додаткових нативних модулів для повної апаратної інтеграції.

Розглянемо структурну схему мобільних платформ (рис. 1.4).

Рисунок 1.4 – Структурна схема мобільних платформ [4]

На схемі показано три гілки розробки:

– Android – середовище Android Studio, мова Kotlin, пакет SDK;

– iOS – середовище Xcode, мова Swift, фреймворк UIKit.

Кросплатформна – фреймворки Flutter (Dart), React Native (JavaScript).

Кожна гілка з'єднується з мобільним пристроєм через середовище виконання (runtime engine).

У розробці мобільних застосунків використовуються мови різних парадигм – від об'єктно-орієнтованих до реактивних і декларативних. Нижче наведено огляд чотирьох найпоширеніших мов (табл. 1.3).

Kotlin (Android):

– офіційна мова Android з 2019 р;

– повністю сумісна з Java, підтримує корутини та функціональне програмування;

– переваги: лаконічність, безпечність від NullPointerException, простота інтеграції з бібліотеками.

Swift (iOS):

– мова Apple для створення iOS/macOS застосунків;

– характеризується високою продуктивністю та безпечністю;

– має потужну підтримку Xcode і бібліотек SwiftUI.

Flutter (Dart):

– фреймворк від Google для кросплатформної розробки;

– використовує власний рушій рендерингу (Skia), що забезпечує високу швидкодію;

– дає змогу створювати застосунки під Android, iOS, Web і Windows із єдиної кодової бази.

React Native (JavaScript / TypeScript):

– базується на фреймворку React, створеному Facebook;

– компоненти інтерфейсу автоматично відображаються у вигляді нативних елементів;

– добре підходить для створення прототипів і швидкого розгортання.

Таблиця 1.3 – Порівняння мов програмування для мобільної розробки

Мова	Тип застосунків	Продуктивність	Гнучкість	Складність навчання	Відкритість екосистеми
Kotlin	Android	★★★★☆	★★★★★	★★★☆☆	Висока
Swift	iOS	★★★★★	★★★★☆	★★★★☆	Середня
Flutter (Dart)	Android/iOS/Web	★★★★★	★★★★☆	★★★☆☆	Висока
React Native (JS / TS)	Android/iOS/Web	★★★☆☆	★★★★★	★★★☆☆	Дуже висока

Побудуємо порівняльну діаграму популярності мов програмування (рис. 1.5).

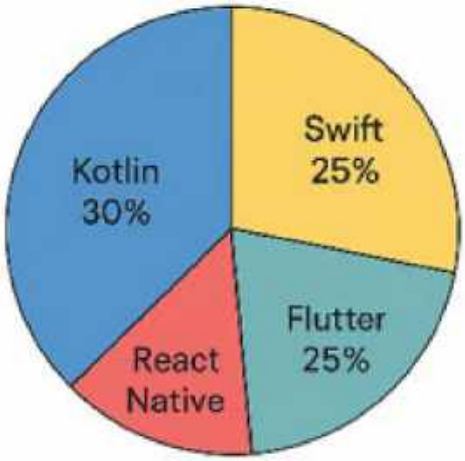

Рисунок 1.5 – Порівняльна діаграма популярності мов програмування [5]

Дана діаграма показує, що Kotlin і Flutter займають домінуючі позиції у створенні мобільних застосунків завдяки універсальності та швидкому рендерингу інтерфейсів.

Проведемо аналіз баз даних та серверної частини. Ефективність мобільного застосунку залежить не лише від фронтенду, а й від правильно організованої серверної логіки та бази даних. Для систем типу «розумного будинку» це критично, адже від стабільності серверної комунікації залежить реакція пристроїв у реальному часі. Розглянемо найбільш популярні серверні рішення [6], та проведемо порівняльну характеристику серверних технологій (таб. 1.4):

– Firebase – хмарна платформа Google з підтримкою реального часу, аналітики та авторизації користувачів;

– MQTT – легкий протокол публікації/підписки, розроблений для IoT-систем. Забезпечує мінімальні затримки та економію трафіку;

– REST API – універсальний спосіб взаємодії клієнт-сервер через HTTP-запити (GET, POST, PUT, DELETE);

– GraphQL – альтернатива REST із вибірковим отриманням даних;

– SQLite / Room – локальні бази даних для зберігання налаштувань користувача та кешу.

Таблиця 1.4 – Порівняльна характеристика серверних технологій

Технологія	Тип	Особливості	Продуктивність	Застосування
Firebase	Хмарна	Авторизація, база даних у реальному часі, аналітика	★★★★★	Android, Flutter
MQTT	IoT-протокол	Обмін повідомленнями публікація/підписка	★★★★★	Smart Home, STM32, Raspberry Pi
REST API	HTTP	Простота реалізації, незалежність клієнта	★★★★☆	Веб і мобільні застосунки
GraphQL	API	Вибіркове отримання даних, складність у налаштуванні	★★★☆☆	Складні мобільні сервіси
SQLite	Локальна БД	Простота інтеграції, автономна робота	★★★☆☆	Застосунки без постійного інтернету

Отже, оптимальний вибір технології залежить від архітектури системи: MQTT і Firebase – для інтерактивних і IoT-рішень, REST – для універсальних мобільних сервісів, GraphQL – для складних даних, SQLite – для локальних офлай – сценаріїв.

Розглянемо схему взаємодії мобільного застосунку з сервером і базою даних, поданих на рисунку 1.6. Огляд сучасних технологій розробки мобільних застосунків показує, що вибір платформи та інструментів визначається типом задачі, вимогами до продуктивності та наявністю ресурсів. Для нативної розробки рекомендується Kotlin (Android) і Swift (iOS).

Для універсальних IoT-рішень – Flutter (Dart) або React Native, які дають змогу швидко створювати багатоплатформні інтерфейси. З боку серверу найефективнішими є Firebase і MQTT, які поєднують швидкість, масштабованість і простоту інтеграції.

Таким чином, оптимальним рішенням для системи «розумного будинку» є кросплатформна архітектура Flutter + MQTT, що забезпечує зручність розробки, інтеграцію з апаратними пристроями та стабільну роботу в реальному часі.

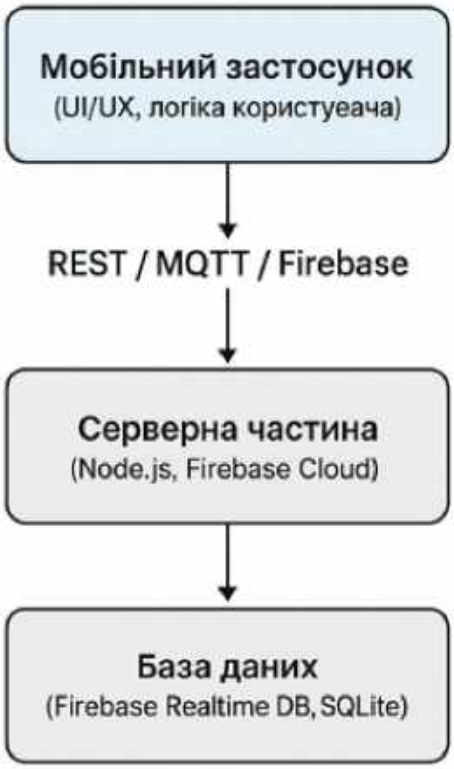

Рисунок 1.6 – Схема взаємодії мобільного застосунку з сервером і базою даних [6]

Розглянемо протоколи зв'язку та безпеку в системах віддаленого керування. Сучасні системи віддаленого керування, зокрема системи «розумного будинку», забезпечують автоматизацію, моніторинг та управління різноманітними пристроями в режимі реального часу. В основі їхньої ефективної роботи лежить надійна архітектура зв'язку, яка гарантує безперервну передачу даних між користувачем, сервером та кінцевими пристроями. Водночас одним із найважливіших викликів для таких систем є інформаційна безпека, адже несанкціонований доступ до компонентів Smart Home може призвести до порушення конфіденційності або втрати контролю над обладнанням.

Для забезпечення стійкості системи необхідно правильно обрати протокол зв'язку, тип архітектури (Cloud, P2P, Local), методи шифрування, а також реалізувати механізми автентифікації користувачів та виявлення уразливостей.

Розглянемо архітектури мережевого з'єднання [7,8]. Системи «розумного будинку» можуть функціонувати на базі різних моделей комунікації між

пристроями. Існують три основні архітектури: Cloud-based, Peer-to-Peer (P2P) та локальне керування (табл. 1.5).

Таблиця 1.5 – Порівняння архітектур віддаленого керування

Характеристика	Cloud-based	P2P	Локальне керування
Тип з'єднання	Через хмарний сервер	Пряме з'єднання між клієнтами	Всередині локальної мережі
Залежність від Інтернету	Повна	Часткова	Мінімальна
Швидкість реакції	Середня (через сервер)	Висока	Дуже висока
Безпека	Залежить від провайдера	Потребує шифрування	Контролюється користувачем
Приклади	Google Home, Tuya Cloud	WebRTC, P2P MQTT	Home Assistant, локальні шлюзи Zigbee
Застосування	Масштабні системи	Камери, медіа, чати	IoT-контролери, мікроконтролери STM32

Порівняльний аналіз показує, що для створення універсальних IoT-рішень найбільш доцільно використовувати кросплатформні фреймворки Flutter (Dart) або React Native, які забезпечують швидку розробку інтерфейсу та одночасну підтримку Android і iOS. На серверному боці найефективнішими залишаються Firebase та MQTT, оскільки вони поєднують високу швидкодію, масштабованість, підтримку роботи в реальному часі та просту інтеграцію з мобільними та IoT-пристроями.

Розглянемо структурну схему архітектур зв'язку (рис. 1.7).

На рисунку 1.7 показано три паралельні моделі:

– Cloud-based: користувач ↔ мобільний додаток ↔ хмарний сервер ↔ IoT-пристрої;

– P2P: користувач ↔ пристрій напряму (через зашифрований канал);

– локальна: усі пристрої всередині Wi-Fi мережі з локальним контролером (Raspberry Pi або STM32).

Рисунок 1.7 – Структурна схема архітектур зв'язку [7]

Розглянемо протоколи зв'язку у системах Smart Home. Для забезпечення стабільної роботи системи важливим є вибір протоколу передачі даних. У таблиці 1.6. нижче наведено основні протоколи, що використовуються у віддалених системах керування.

Порівняння комунікаційних протоколів показує, що MQTT є оптимальним для IoT-рішень завдяки низькому навантаженню та підтримці гарантованої доставки (QoS), хоча потребує окремого брокера. HTTP / REST API залишається універсальним стандартом вебвзаємодії, але має вищу затримку та не підходить для подій реального часу. WebSocket забезпечує двосторонній обмін даними в реальному часі, що робить його ефективним для моніторингу, хоча налаштування складніше.

Таблиця 1.6 – Характеристика основних протоколів зв'язку [7]

Протокол	Тип передачі	Особливості	Переваги	Недоліки
MQTT	Публікація / підписка	Мінімальне навантаження, ідеальний для IoT	Легкий, підтримує QoS	Потрібен брокер
HTTP / REST API	Запит-відповідь	Універсальний стандарт Web	Простий у реалізації	Висока затримка
WebSocket	Постійне з'єднання	Двостороння передача в реальному часі	Ідеальний для моніторингу	Складність налаштування
CoAP	UDP-з'єднання	Призначений для сенсорних пристроїв	Енергоефективний	Обмежена підтримка
Zigbee / Z-Wave	Мережа Mesh	Низьке енергоспоживання	Надійність, масштабованість	Невелика швидкість передачі

CoAP, орієнтований на сенсорні пристрої, вирізняється енергоефективністю, але має обмежену підтримку. Zigbee / Z-Wave як mesh-протоколи забезпечують надійність і масштабованість у домашніх мережах, проте їхня швидкість передачі нижча порівняно з IP-протоколами.

У цілому вибір протоколу залежить від вимог системи: MQTT – для IoT, WebSocket – для реального часу, REST – для стандартних мобільних сервісів, а Zigbee / Z-Wave – для локальних сенсорних мереж.

Розглянемо діаграму популярності протоколів у Smart Home на рисунку 1.8. Проаналізувавши її, бачимо зростання ролі MQTT як універсального стандарту для IoT-систем

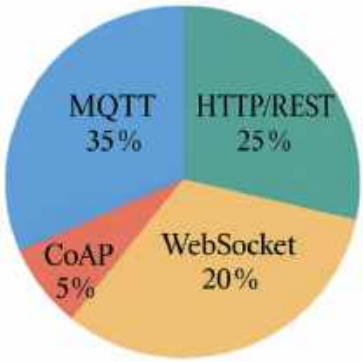

Рисунок 1.8 – Діаграма популярності протоколів у Smart Home [7]

Розглянемо методи захисту даних та автентифікації користувачів. Інформаційна безпека є ключовим аспектом у віддаленому керуванні, адже системи Smart Home можуть передавати персональні дані, параметри середовища або статус пристроїв [9]. Основні напрями захисту включають шифрування, аутентифікацію, контроль доступу та захист мережевого трафіку, що подано в таблиці 1.7.

Таблиця 1.7– Методи забезпечення безпеки системи

Механізм	Принцип дії	Приклади технологій	Рівень захисту
Шифрування даних	Передача у зашифрованому вигляді	AES-256, TLS 1.3, HTTPS	★★★★★
Аутентифікація користувачів	Перевірка особи через токен або пароль	OAuth 2.0, JWT, Firebase Auth	★★★★☆
Контроль доступу	Розмежування прав користувачів	Role-Based Access Control (RBAC)	★★★★☆
Захист від атак	Виявлення підозрілої активності	IDS/IPS, WAF, anomaly detection	★★★☆☆
VPN та сегментація мережі	Обмеження прямого доступу	OpenVPN, WireGuard	★★★★★

Аналіз механізмів безпеки показує, що найвищий рівень захисту забезпечують шифрування даних (AES-256, TLS 1.3) та VPN / сегментація мережі, які гарантують захист інформації під час передавання та обмежують несанкціонований доступ до інфраструктури. Аутентифікація користувачів та контроль доступу (RBAC) відіграють ключову роль у підтвердженні особи та забезпеченні відповідних прав користувачів, формуючи основу безпечної взаємодії з системою. Механізми захисту від атак (IDS / IPS, WAF) додатково підсилюють безпеку, але їхня ефективність залежить від конфігурації та складності системи.

У комплексі ці підходи формують багаторівневу модель захисту, необхідну для безпечної роботи мобільних застосунків і IoT-систем «розумного будинку».

На рисунку 1.9 подана схема шифрування і доступу користувача.

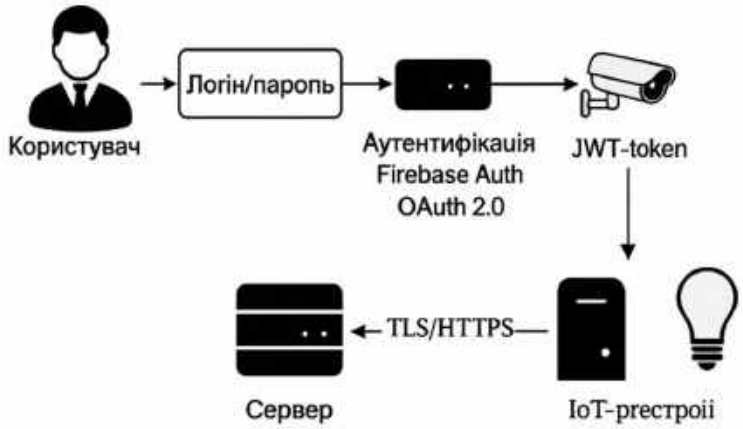

Рисунок 1.9 – Схема шифрування і доступу користувача

Дана схема показує послідовність дій, яку необхідно виконати для шифрування і доступу користувача.

Користувач вводить логін / пароль → проходить автентифікацію через Firebase Auth або OAuth 2.0.

Сервер генерує JWT-токен для сеансу.

Дані передаються по зашифрованому каналу TLS / HTTPS.

IoT-пристрої приймають команди лише після валідації токена.

Проведемо аналіз уразливостей розумних будинків (таблиця 1.8) Попри численні переваги, системи Smart Home залишаються вразливими до різних типів атак, зокрема:

- перехоплення даних (sniffing) – у незашифрованих Wi-Fi з'єднаннях;
- Replay-атаки – повторна передача старих команд без перевірки токена;
- атаки типу «man-in-the-middle» (MITM) – підміна пакета між клієнтом і сервером;
- SQL Injection та XSS – у випадку некоректної обробки введення користувача;
- соціальна інженерія – фішинг-посилання, спрямовані на отримання доступу до облікових записів.

У таблиці 1.8 розглянемо основні уразливості Smart Home-систем.

Таблиця 1.8 – Основні уразливості Smart Home-систем [7]

Тип атаки	Джерело	Наслідки	Спосіб запобігання
MITM	Зовнішній зловмисник	Викрадення даних	Використання HTTPS, TLS 1.3
SQL Injection	Некоректна обробка запитів	Отримання доступу до бази	Використання ORM і валідації
Replay Attack	Повтор команд	Неконтрольовані дії	Одноразові токени, nonce
Brute-force	Підбір пароля	Несанкціонований доступ	CAPTCHA, обмеження спроб
Phishing	Людський фактор	Компрометація акаунту	Двофакторна автентифікація

Порівняння типових кібератак демонструє, що загрози для мобільних застосунків та IoT-систем можуть походити як від технічних вразливостей, так і від людського фактора. MITM-атаки і Replay Attack спрямовані на перехоплення або повтор команд і потребують використання сучасних криптографічних протоколів (HTTPS, TLS 1.3, nonce). SQL Injection виникає через некоректну обробку запитів, а її запобігання забезпечують ORM, параметризація та валідація даних. Brute-force атаки пов'язані з підбором пароля, тому ефективними засобами захисту є CAPTCHA, ліміти спроб і складні політики паролів. Phishing залишається загрозою для користувачів, тому двофакторна автентифікація та навчання користувачів є ключовими методами протидії.

Загалом інформація з таблиці підкреслює необхідність багаторівневого захисту, що поєднує криптографічні методи, контроль доступу, валідацію даних та механізми автентифікації для забезпечення безпеки систем «розумного будинку».

Проаналізуємо ризики, використовуючи діаграму рівнів ризики безпеки у Smart Home (рис. 1.10).

Рисунок 1.10 – Діаграма рівнів ризику безпеки у Smart Home [10]

Діаграма у вигляді стовпців демонструє:

- високий ризик – MITM, Phishing;
- середній ризик – Replay, Brute-force;
- низький ризик – XSS, SQL Injection (при коректній валідації).

Отже, провівши аналіз літературних джерел, можемо сформулювати наступні висновки:

– архітектура мережевого з'єднання має визначальний вплив на безпеку та швидкодію системи віддаленого керування;

– найбільш ефективною для IoT є комбінація Cloud + MQTT + локальний шлюз, що забезпечує баланс між автономністю та масштабованістю;

– безпека Smart Home повинна базуватись на багаторівневій моделі захисту – від шифрування до автентифікації користувача;

– використання OAuth 2.0, JWT, TLS 1.3 суттєво підвищує рівень захищеності системи;

– постійне тестування безпеки та оновлення прошивок є необхідною умовою мінімізації уразливостей.

1.3 Постановка завдання на кваліфікаційну роботу магістра

Проведений аналіз наукових джерел, існуючих Smart Home – платформ, мобільних технологій та протоколів зв'язку засвідчує, що проблема побудови надійних і безпечних систем віддаленого керування «розумним будинком» залишається актуальною як у науковому, так і в прикладному аспектах. Наявні комерційні рішення (Google Home, Amazon Alexa, HomeKit, SmartThings, Home Assistant тощо) орієнтовані або на закриті екосистеми й хмарні сервіси, або потребують складної конфігурації та глибоких технічних знань користувача. Додатковою проблемою є необхідність поєднання в одній системі вимог до реального часу, інформаційної безпеки, масштабованості й енергоефективності. Традиційні підходи, засновані лише на HTTP / REST – взаємодії або локальному керуванні, не забезпечують достатньої гнучкості в умовах зростання кількості IoT-пристроїв та потреби в кросплатформних мобільних інтерфейсах. У зв'язку з цим виникає потреба у створенні мобільного застосунку, який би використовував сучасні технології (Flutter / React Native, Firebase, MQTT), підтримував багаторівневий захист даних і надавав користувачеві зручний засіб керування елементами «розумного будинку» з будь-якого мобільного пристрою.

Метою кваліфікаційної роботи є розроблення та дослідження мобільного застосунку для віддаленого керування пристроями системи «розумного будинку» на основі сучасних IoT-технологій, який забезпечує безпечний обмін даними, моніторинг стану обладнання в режимі, наближеному до реального часу, та можливість подальшого масштабування й інтеграції з апаратними платформами.

Для досягнення поставленої мети необхідно вирішити такі основні завдання:

– провести огляд і аналіз предметної області Smart Home, існуючих платформ та протоколів зв'язку, узагальнити їхні переваги та обмеження;

– обґрунтувати вибір архітектури системи та технологічного стеку (мобільна платформа, серверна частина, MQTT / REST, база даних, засоби шифрування й автентифікації);

– розробити архітектурну модель мобільного застосунку та схему взаємодії користувача, сервера і IoT-пристроїв;

– реалізувати прототип мобільного застосунку з підтримкою віддаленого керування й моніторингу, авторизації користувачів та захищеного обміну даними;

– налаштувати серверну інфраструктуру (Firebase / MQTT-брокер, базу даних) та інтегрувати її з мобільним клієнтом і апаратними контролерами;

– провести функціональне, модульне й навантажувальне тестування прототипу, а також оцінити його продуктивність, зручність використання та рівень безпеки;

– проаналізувати результати експериментів, визначити переваги й недоліки розробленої системи та сформулювати рекомендації щодо її подальшого вдосконалення й можливостей практичного впровадження.

РОЗДІЛ 2
ТЕОРЕТИЧНЕ ДОСЛІДЖЕННЯ ТА ПРАКТИЧНА РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Обґрунтування вибору шляхів, технологій, алгоритмів і засобів вирішення поставленого завдання

На основі аналізу предметної області, представленого в розділі 1, у цьому підрозділі обґрунтуємо вибір архітектурних підходів, мобільних і серверних технологій, моделей даних та засобів забезпечення безпеки, які будуть використані для розроблення мобільного застосунку віддаленого керування розумним будинком. Основна мета – сформувати цілісну, узгоджену з технічними вимогами та обмеженнями концепцію програмної системи, що поєднує надійність, масштабованість і зручність використання.

Сучасні Smart Home-рішення зазвичай будуються на основі багаторівневих архітектур, що включають рівень пристроїв (сенсори, актуатори), мережевий рівень, рівень платформи/хмари та рівень застосунків(мобільних і веб) [1-6].

Для поставленого завдання розглянуто три базові варіанти:

– класична клієнт – сервер архітектура (локальний сервер у домашній мережі, мобільний клієнт підключається напряму);

– повністю хмарно – орієнтована архітектура (усі сервіси розміщені в хмарі, пристрої й застосунок взаємодіють через інтернет);

– гібридна архітектура, у якій частина логіки виконується локально, а частина – у хмарі.

Аналітичні огляди розробки Smart Home-платформ показують, що повністю хмарний підхід спрощує масштабування, але підвищує залежність від інтернет-з'єднання та сторонніх сервісів [8]. Натомість гібридні архітектури дозволяють виконувати критично важливі функції (реакція на події безпеки, локальні сценарії автоматики) локально, використовуючи хмару для довготривалого зберігання даних, аналітики та інтеграції з іншими сервісами [9].

Використання MQTT як легковагового протоколу для локальної взаємодії також рекомендовано стандартами OASIS MQTT 5.0 [4-7].

Для даної кваліфікаційної роботи доцільно обрати гібридну архітектуру, у якій:

– IoT-шлюз відповідає за локальну взаємодію з пристроями;

– серверний компонент (хмарний або розгорнутий на VPS) реалізує REST API, брокер MQTT та логіку авторизації;

– мобільний застосунок взаємодіє із сервером через захищені канали, а в окремих сценаріях може працювати напряму з локальним шлюзом у межах домашньої мережі.

На рисунку 2.1 доцільно подати узагальнену структурну схему гібридної архітектури, де відображено зв'язки між мобільним клієнтом, хмарним сервером, брокером MQTT та IoT – пристроями.

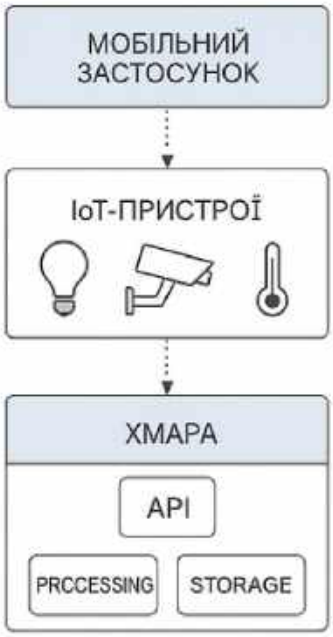

Рисунок 2.1 – Узагальнена структурна схема гібридної архітектури

У таблиці 2.1 наведено порівняльний аналіз трьох архітектур (клієнт – сервер, хмарна, гібридна) за критеріями: затримка, залежність від інтернету, масштабованість, складність підтримки та безпека.

Таблиця 2.1 – Порівняльний аналіз архітектур Smart Home

Критерій	Клієнт – серверна архітектура	Хмарна архітектура	Гібридна архітектура
Затримка (Latency)	Низька всередині локальної мережі; висока при доступі ззовні	Середня або висока, залежить від інтернету	Найнижча для локальних сценаріїв; середня для хмарних
Залежність від інтернету	Низька (локальна мережа забезпечує роботу)	Висока (усі операції через хмару)	Середня: критичні дії локальні, вторинні – через хмару
Масштабованість	Обмежена апаратними ресурсами локального сервера	Висока (масштабується хмарна інфраструктура)	Висока: поєднання локальної обробки та хмарного масштабування
Складність підтримки	Середня: необхідно підтримувати локальний сервер	Низька: обслуговує провайдер хмарної платформи	Середня/висока: необхідна синхронізація локальних і хмарних компонентів
Безпека	Висока, якщо правильно налаштована локальна мережа; складно забезпечити зовнішній доступ	Залежить від хмари; високий рівень при правильній конфігурації	Дуже висока: локальні дані залишаються всередині мережі, критичні процеси ізольовані
Надійність	Висока у межах локальної мережі; проблеми при віддаленому доступі	Залежить від стабільності інтернету і хмарних сервісів	Найвища: локальна робота зберігається навіть при втраті інтернету
Гнучкість інтеграції	Низька/середня: складно інтегрувати пристрої різних виробників	Висока: широкий спектр API, інтеграції зі сторонніми сервісами	Найвища: підтримка локальних протоколів + хмарних сервісів

На основі таблиці побудуємо діаграму, подану на рисунку 2.2. Провівши порівняльний аналіз трьох архітектур – клієнт – серверної, хмарної та гібридної – за критеріями затримки, залежності від Інтернету, масштабованості, складності підтримки та рівня безпеки дозволяє сформувати чітке уявлення про їхні переваги та обмеження, а саме аналіз таблиці та візуалізація на діаграмі однозначно демонструють, що гібридна архітектура є найбільш оптимальною для систем Smart Home.

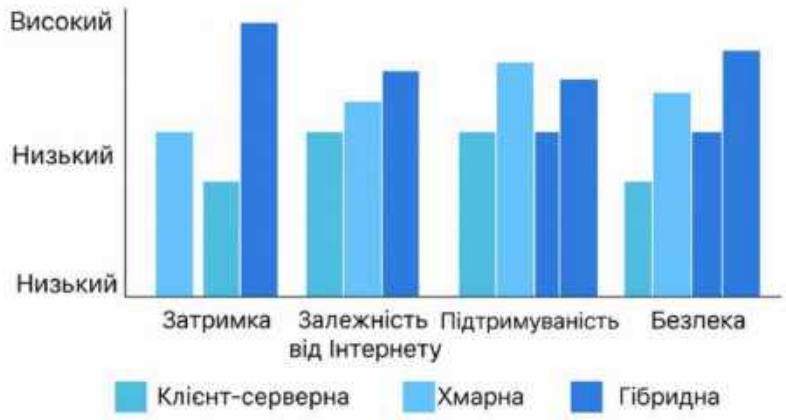

Рисунок 2.2 – Діаграма порівняльного аналізу архітектур Smart Home за ключовими критеріями (клієнт – серверна, хмарна та гібридна архітектури)

З огляду на те, що більшість користувачів розумних будинків використовують смартфони на базі Android та iOS, ключовим завданням є вибір технології, яка дозволить:

– забезпечити кросплатформеність;

– підтримувати високу продуктивність UI;

– спростити підтримку та розширення застосунку.

Нативна розробка (Kotlin для Android, Swift для iOS) дає максимальний контроль над функціоналом, але потребує двох незалежних кодових баз. Це збільшує вартість підтримки та ускладнює синхронний розвиток функціоналу.

У сучасних оглядах кросплатформних технологій (Flutter vs React Native) наголошується, що:

– Flutter забезпечує вищу продуктивність інтерфейсу завдяки власному рушію рендерингу (без «містка» JavaScript) та AOT-компіляції;

– React Native має переваги у вигляді доступу до JavaScript – екосистеми, але використовує місток між JS та нативними компонентами, що може створювати додаткові накладні витрати.

За даними аналітичних звітів (Statista, GitHub, опитування розробників), Flutter демонструє зростаючу популярність, особливо для застосунків із

багатими інтерфейсами та інтерактивним оновленням даних, що важливо для Smart Home – сценаріїв (динамічні панелі, графіки, статуси пристроїв).

З урахуванням цих факторів у роботі обрано Flutter як основний фреймворк для розробки мобільного застосунку, з такими аргументами:

– єдина кодова база для Android та iOS;

– висока швидкодія та плавність анімацій;

– наявність зрілої екосистеми плагінів для роботи з REST API, MQTT, WebSocket;

– хороша підтримка архітектурних патернів (Bloc, Provider, Riverpod) для розділення логіки та UI.

– нативна розробка (Kotlin / Swift) дає максимальний контроль, але вимагає двох окремих кодових баз.

У сучасних оглядах кросплатформних фреймворків зазначено, що:

– Flutter забезпечує вищу продуктивність завдяки власному рушію Skia та AOT-компіляції [5];

– React Native покладається на JavaScript – «місток», що збільшує затримку UI та складність оптимізації [6];

– Google регулярно оновлює Flutter, забезпечуючи стабільність екосистеми (2021-2024) [7].

Аналітичні звіти Statista та GitHub демонструють постійне зростання популярності Flutter серед розробників мобільних застосунків [8]. З урахуванням цих факторів у роботі обрано Flutter як основний фреймворк.

На рисунку 2.3 представлено архітектурну схему мобільного застосунку на Flutter (шари UI, стану, сервісів доступу до API).

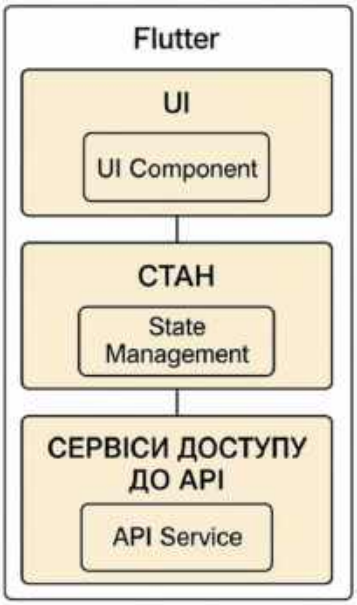

Рисунок 2.3 – Архітектурна схема мобільного застосунку на Flutter [9]

На рисунку 2.4 розглянемо діаграму порівняльного аналіу нативної розробки, Flutter та React Native за ключовими критеріями.

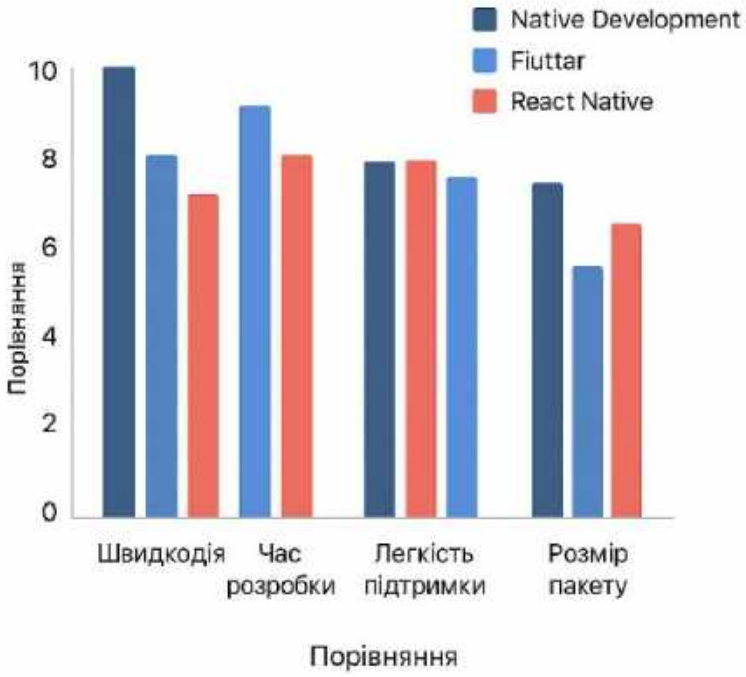

Рисунок 2.4 – Діаграма порівняльного аналізу нативної розробки, Flutter та React Native за ключовими критеріями [9]

Загалом, порівняння на діаграмі візуально підтверджують, що Flutter є оптимальним вибором для розроблення кросплатформного мобільного застосунку для керування розумним будинком. Він забезпечує найкращий баланс між продуктивністю, швидкістю розробки, простотою підтримки та універсальністю, що є ключовими факторами в розробці сучасних Smart Home систем.

При проєктуванні серверної частини системи Smart Home важливими є:

– низька затримка між мобільним застосунком та пристроями;

– надійність доставки команд і телеметрії;

– масштабованість при збільшенні кількості пристроїв і користувачів.

Огляд досліджень MQTT vs HTTP / REST для IoT показує, що HTTP / REST добре підходить для конфігураційних запитів та транзакційних операцій «запит – відповідь», однак створює значні накладні витрати через обсяг службових даних та необхідність встановлення окремого з'єднання для кожного запиту, що збільшує затримку системи [4-7].

MQTT, як легковаговий публікаційно – підписковий протокол, демонструє меншу затримку, менші мережеві накладні витрати та значно кращу ефективність при передачі великої кількості малих повідомлень (телеметрія сенсорів) у порівнянні з HTTP/REST [2-9].

Дослідження 2023-2024 років, присвячені порівнянню HTTP, MQTT over TCP та MQTT over WebSocket у задачах IoT та цифрових двійників, показують, що MQTT забезпечує вищу пропускну здатність, стабільність з'єднання і стійкість до нестабільних мережевих умов, що є критично важливим для Smart Home-систем [5-8].

На основі літературних даних побудуємо діаграмупорівняння MQTT та HTTP / REST по основним критеріям (рис. 2.5)

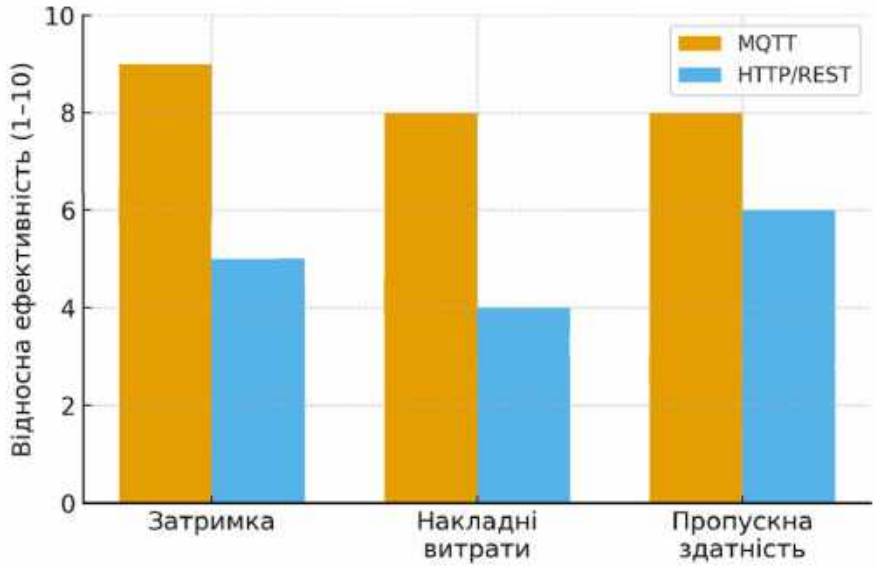

Рисунок 2.5 – Діаграма порівняння MQTT та HTTP / REST по основним критеріям [8]

У зв'язку з цим обґрунтовано такий вибір:

- REST API (HTTPS) – для реєстрації, авторизації, керування профілями, конфігурації пристроїв, отримання агрегованих даних;
- MQTT – для передачі телеметрії від пристроїв, оновлення станів у реальному часі та надсилання керуючих команд;
- MQTT over WebSocket – як опція для взаємодії з веб-клієнтами та для push-подій.

У ролі серверної платформи доцільно використати стек на базі Node.js / NestJS або Python (FastAPI), які мають:

– зрілі бібліотеки для реалізації REST API;

– підтримку клієнтів MQTT;

– можливість розгортання у хмарних середовищах (Docker, Kubernetes).

На рисунку 2.6 розглянемо схему інтеграції мобільного застосунку з сервером і IoT-пристроями, де показано потоки даних через REST і MQTT.

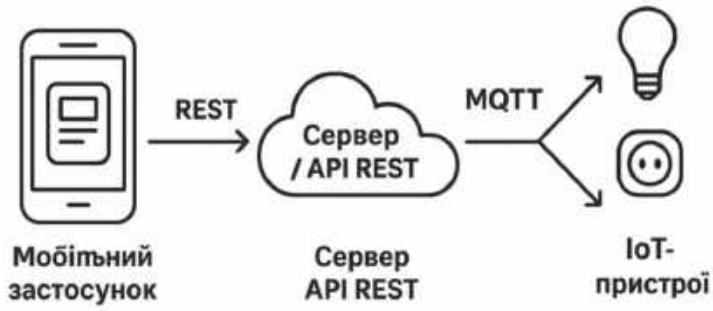

Рисунок 2.6 – Схема інтеграції мобільного застосунку з сервером і IoT-пристроями

Отже, дослідження показують, що:

– REST API добре підходить для конфігураційних запитів та авторизації;

– MQTT демонструє кращу пропускну здатність і меншу затримку для Smart Home-комунікацій [9];

– MQTT-over-WebSocket рекомендовано для веб-клієнтів та push-нотифікацій [10].

Порівняльні тести 2023-2024 років доводять перевагу MQTT у нестабільних мережах над HTTP / REST [11].

Для серверної частини доцільно використати Node.js / NestJS або Python FastAPI – обидві технології добре інтегруються з MQTT і підтримують Docker / Kubernetes для масштабування [12].

2.2 Практична реалізація об’єкта проектування

Серверна частина системи віддаленого керування розумним будинком забезпечує обробку даних, взаємодію між мобільним застосунком і IoT-пристроями, а також реалізує логіку автоматизації, обробку сценаріїв і журнал подій. У сучасних системах домінують підходи, що базуються на мікросервісній архітектурі, REST API, MQTT-комунікаціях, контейнеризації (Docker) і хмарній інфраструктурі, що підвищує масштабованість та відмовостійкість системи [10].

На рисунку 2.7 подано архітектура серверної частини

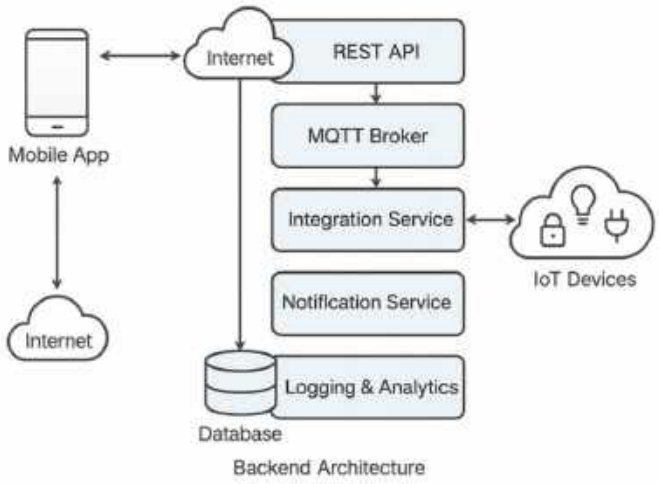

Рисунок 2.7 – Типова архітектура серверної частини

Для формалізації функціональних вимог і структури системи «Розумного будинку» використовуються UML-діаграми згідно зі стандартами ISO / IEC 19505-1:2021 та ISO / IEC 19505-2:2021 [6-13]. Вони дозволяють візуалізувати ролі користувачів, межі системи, взаємодію компонентів і послідовність обміну повідомленнями. Доцільно розробити такі UML-діаграми – UML-діаграма прецедентів (use case), яка відображає основні сценарії взаємодії користувача з системою: авторизація, перегляд панелі керування, керування пристроями, створення сценаріїв, перегляд журналу подій. Цю діаграму доцільно подати як рисунок 2.8.

Діаграма відображає основні сценарії взаємодії користувача, адміністратора та зовнішніх систем / пристроїв із системою «Розумного дому».

Розглянемо основних акторів та їх функції:

- користувач – основний актор, який виконує авторизацію, переглядає панель керування, керує пристроями, отримує сповіщення та працює зі сценаріями;
- адміністратор – має розширені можливості, зокрема призначення прав доступу;
- розумний пристрій – надсилає в систему статуси й сигнали;

– система безпеки – передає тривожні сигнали у випадку інцидентів.

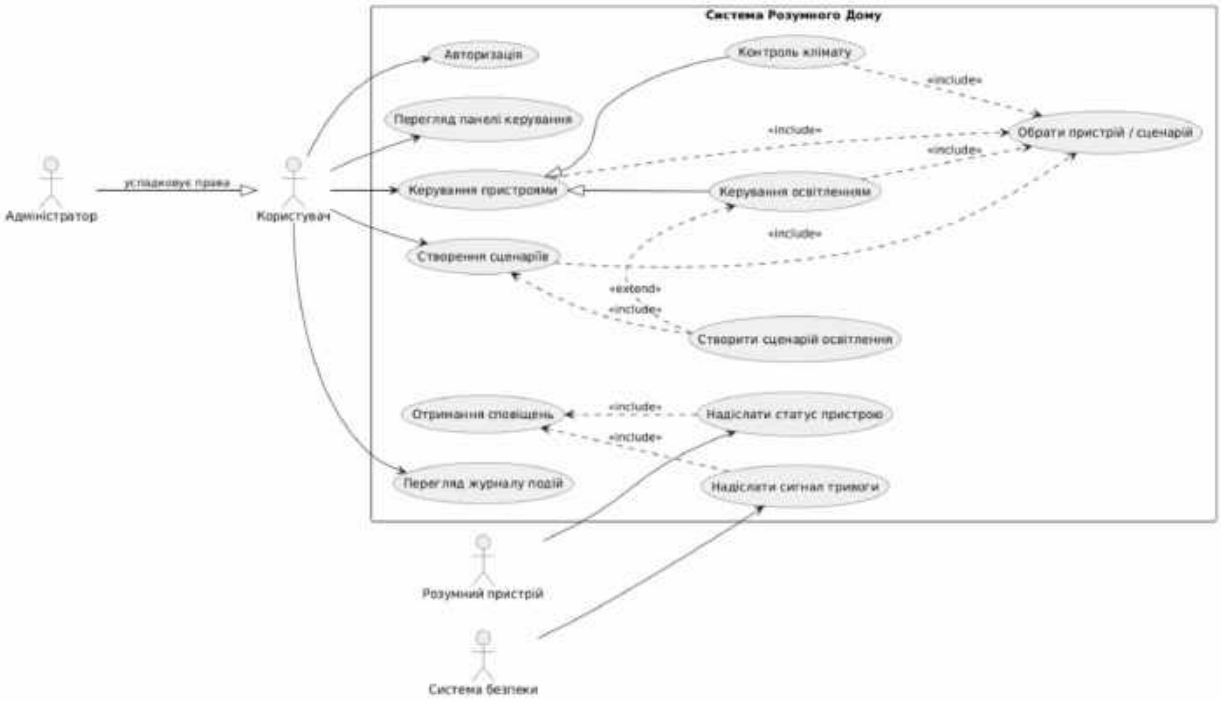

Рисунок 2.8 – UML-діаграма прецедентів мобільного застосунку «Розумний дім»

Перерахуємо основні прецеденти:

– авторизація – користувач входить у систему;

– перегляд панелі керування – доступ до основних функцій;

– керування пристроями / освітленням / кліматом – прецеденти управління;

– створення сценаріїв – можливість формувати автоматизації (наприклад, «Вимкнути світло при виході»);

– розширення: створити сценарій освітлення;

– стримання сповіщень – дані про статус пристроїв або події;

– перегляд журналу подій – історія змін, тривог, дій користувача;

– адміністрування (надання прав) – лише для адміністратора.

В даній діаграмі зв’язок «include» застосовано там, де певна функція є обов’язковою частиною іншої(наприклад, «Керування пристроями» включає

«Обрати пристрій»), а «extend» показує необов'язкові або додаткові функції (наприклад, «Створення сценаріїв» може бути розширене «Створити сценарій освітлення»). Розумні сенсори або модулі надсилають у систему: статус пристрою (температура, стан вмик. / вимк.) або сигнал тривоги (від системи безпеки). Система на основі цих даних надсилає сповіщення користувачу та записує події в журнал. Приклад лістингу серверного коду (Python + Flask + MQTT) подано в додатку Б.

Для показу розподілу системи на логічні частини, побудуємо UML-діаграма компонентів, яка складатиметься з мобільного клієнта (UI, менеджер стану, мережевий модуль), сервера (API-шар, сервіс керування пристроями, брокер повідомлень, модуль безпеки), бази даних, IoT-шлюз. На рисунку 2.9 подано UML-діаграма компонентів для системи «Розумний будинок».

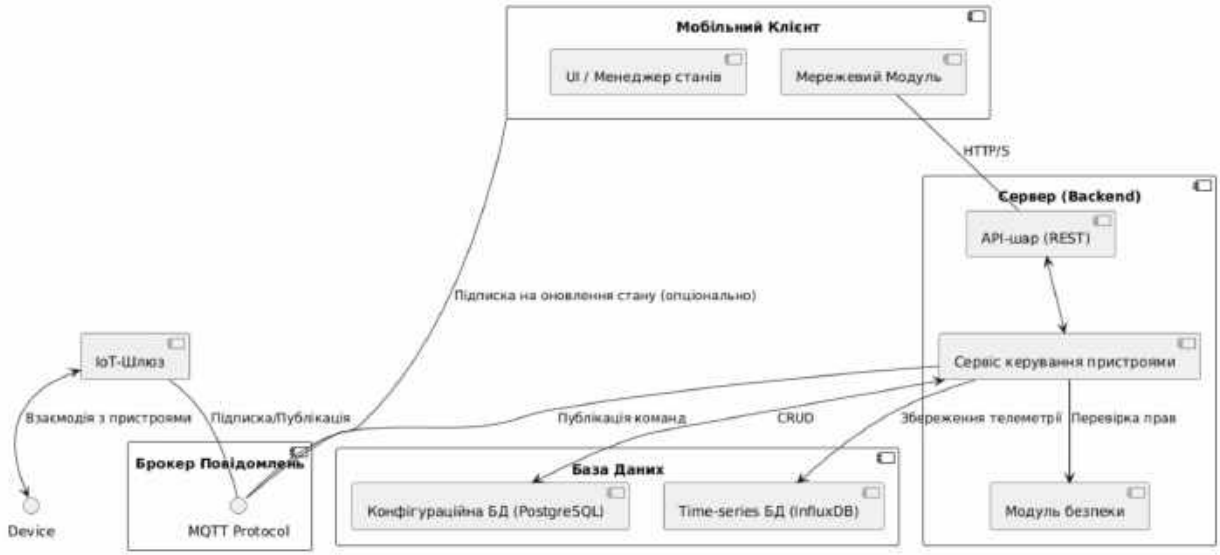

Рисунок 2.9 – UML-діаграма компонентів для системи «Розумний будинок» на базі MQTT та клієнт-серверної моделі

Як бачимо, основні компоненти для системи «Розумний будинок» на базі MQTT та клієнт-серверної моделі є наступні.

Пристрої / IoT-шлюз:

– фізичні сенсори й виконавчі модулі (реле, датчики температури тощо);

– підключені до IoT-шлюзу, який говорить з брокером по протоколу MQTT (публікація/підписка повідомлень).

Брокер повідомлень (MQTT Broker):

– центральна шина обміну даними;

– приймає телеметрію від пристроїв, розсилає її підписникам (серверу, мобільному клієнту), приймає команди керування від сервера й передає їх пристроям.

Сервер (Backend):

– працює по HTTPS з мобільним клієнтом;

– містить:

– API-шар (REST) – точки доступу для мобільного застосунку (авторизація, список пристроїв, надсилання команд);

– сервіс керування пристроями – формує MQTT-команди та відправляє їх брокеру;

– модуль безпеки – перевіряє права користувача, автентифікацію, токени доступу;

– записує телеметрію в базу даних.

База даних:

– конфігураційна БД (PostgreSQL) – користувачі, пристрої, сценарії, права доступу;

– Time-series БД (InfluxDB) – історія вимірювань (температура, вологість, стан реле з часом).

Мобільний клієнт:

– має UI / менеджер станів (екрани, логіка керування);

– мережевий модуль:

– через HTTPS викликає REST-API сервера;

– за бажанням може сам підписуватися на MQTT-топіки для миттєвих оновлень стану (опціонально показано на діаграмі як «Підписка на оновлення стану»).

Основний потік такий: пристрій → телеметрія по MQTT → брокер → сервер → запис у БД → мобільний клієнт читає через REST або MQTT; користувач → мобільний клієнт → REST-запит до сервера → перевірка прав → публікація MQTT-команди → брокер → IoT-шлюз / пристрій.

Приклад програмного коду (спрощений backend що подано в додатку В) – це компактний приклад Python-backend (Flask + MQTT), який відповідає логіці діаграми:

– REST-endpoint для керування пристроєм;

– обробка телеметрії з MQTT;

– умовний запис у «конфігураційну БД» та «time-series БД» (функції – заглушки замість реального коду PostgreSQL / InfluxDB).

Для опису таких типових сценаріїв, як «Користувач вмикає освітлення в кімнаті через мобільний застосунок» та інших, використаємо UML-діаграму послідовності, поданому на рисунку 2.10.

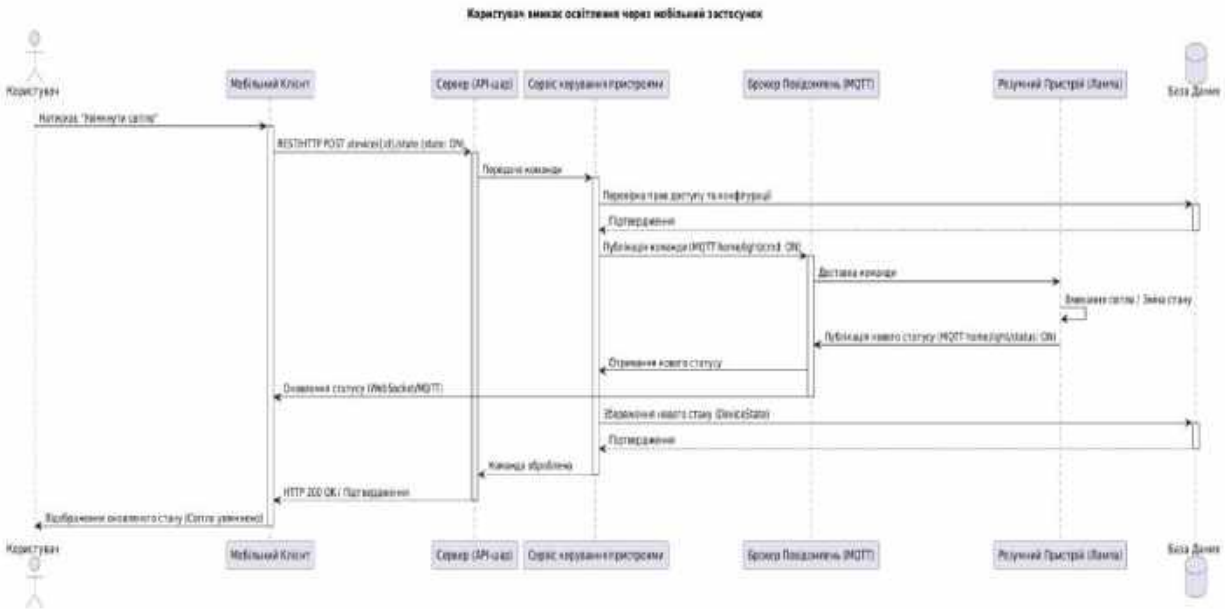

Рисунок 2.10 – UML-діаграма послідовності для системи «Розумний будинок»

Діаграма показує, як проходить повний цикл увімкнення лампи з телефону до фізичного пристрою й назад до користувача у вигляді оновленого статусу. Приклад коду до даного сценарію подано в додатку Г. Це компактний приклад

Python + Flask + paho-mqtt, який реалізує основну логіку діаграми: REST-команда → MQTT до лампи → MQTT-статус назад → збереження в БД (умовно) → видача стану клієнту. В додатку Д подано програмний клієнтський код.

На рисунку 2.11 подано створену UML-діаграму класів для системи «Розумний будинок».

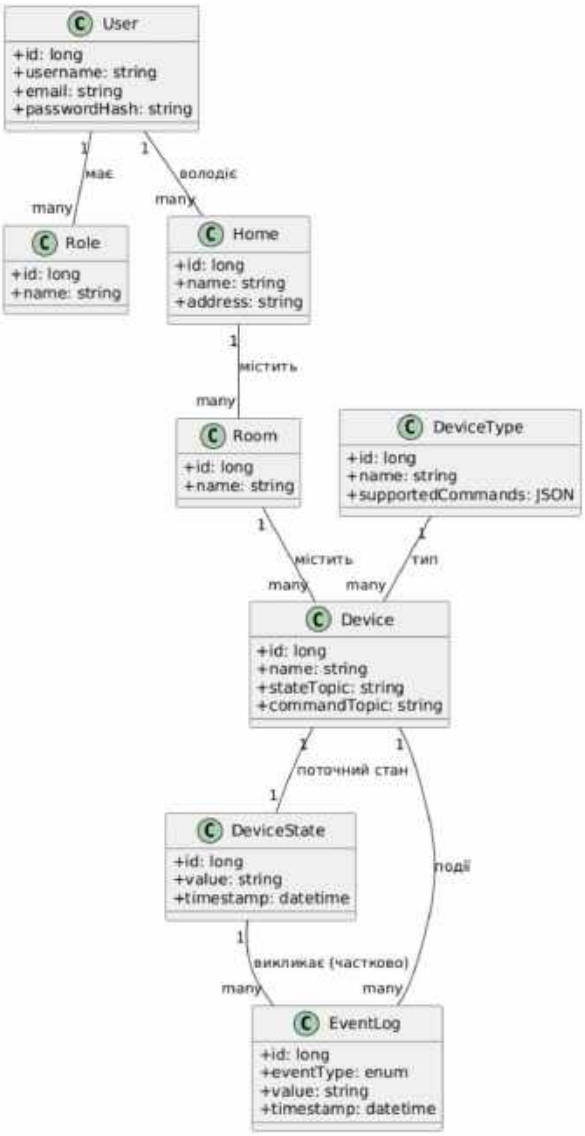

Рисунок 2.11 – UML-діаграму класів для системи «Розумний будинок»

Діаграма класів для системи «Розумний будинок» описує головні сутності доменної моделі та їх зв’язки:

– User – користувач системи (звичайний або адміністратор). Має облікові дані, роль, може керувати пристроями та створювати сценарії;

– Device – узагальнений клас для всіх пристроїв «розумного будинку». Містить ідентифікатор, назву, розташування, поточний стан і метод для надсилання команд.

Від нього наслідуються, наприклад:

– Light – освітлювальний прилад (лампа), додатково має яскравість тощо;

– ClimateController – термостат / контролер клімату з температурою та режимами.

– Scenario – сценарій автоматизації (наприклад: «Нічний режим»). Складається з одного або кількох Rule (правил), може бути активований / виконаний;

– Rule – окреме правило сценарію: умова спрацювання (тригер) та дія над пристроями;

– Notification – сповіщення для користувача (тривога, зміна стану, помилка);

– EventLog – запис у журналі подій (історія команд, змін станів, помилок);

– SmartHomeSystem – кореневий клас, що агрегує колекції користувачів, пристроїв і сценаріїв.

З діаграми добре видно логічні зв'язки:

– один User може мати доступ до багатьох Device, сценаріїв та сповіщень;

– один Scenario складається з декількох Rule і керує одним або кількома Device;

– кожен Device породжує багато записів EventLog;

– SmartHomeSystem композиційно містить користувачів, пристрої та сценарії.

Отже, у другому розділі сформовано цілісну концепцію програмно – апаратної платформи системи «Розумний будинок» та підтверджено її життєздатність практичною реалізацією (Додаток Е). На основі порівняльного

аналізу клієнт серверної, повністю хмарної та гібридної архітектур обґрунтовано вибір саме гібридного підходу, який поєднує низьку затримку й автономність локальної мережі з масштабованістю та гнучкістю хмарних сервісів. Доведено доцільність використання протоколу MQTT для передачі телеметрії та керуючих команд у режимі реального часу, тоді як REST / HTTPS застосовується для авторизації, конфігурації пристроїв та отримання агрегованих даних. Порівняння сучасних мобільних технологій показало перевагу Flutter як основного фреймворку для створення кросплатформеного клієнта з високою продуктивністю інтерфейсу та розвиненою екосистемою бібліотек для роботи з REST, MQTT і WebSocket.

Практична частина розділу демонструє, що обрані підходи можуть бути успішно втілені на рівні робочого прототипу. Розроблено архітектуру серверної частини з використанням REST – API, брокера MQTT та баз даних (PostgreSQL для конфігураційних даних і InfluxDB для часових рядів телеметрії), реалізовано обмін даними між мобільним застосунком, сервером і IoT-пристроями. За допомогою UML-діаграм прецедентів, компонентів, послідовності та класів формалізовано функціональні вимоги, структуру і взаємодію основних сутностей системи. Це забезпечує прозорість проєктних рішень і створює основу для подальшого розширення функціоналу, підвищення безпеки та масштабованості системи «Розумний будинок» у наступних розділах роботи.

РОЗДІЛ 3
ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ РЕЗУЛЬТАТИВНОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Методика проведення дослідження

Метою експериментального дослідження є всебічна оцінка результативності, швидкодії, стабільності та зручності використання розробленої системи «розумного будинку», що включає мобільний застосунок на Flutter, серверну частину з REST-API та MQTT-брокером, а також IoT-вузли (контролери освітлення та датчики). Методика спирається на рекомендації стандарту ISO / IEC 25010 щодо оцінки якості програмних продуктів та сучасних робіт з тестування IoT- і Smart Home-систем [15].

У ході експерименту аналізуються такі групи показників:

– функціональна результативність – коректність виконання сценаріїв керування та автоматизації;

– швидкодія – затримки при обміні через REST та MQTT, повний час реакції системи;

– стабільність і відмовостійкість – поведінка при збоях мережі, перезапуску компонентів;

– продуктивність – здатність серверної частини обробляти запити при зростаючому навантаженні;

– зручність використання (usability) – оцінка інтерфейсу за методикою SUS.

Для аналізу вище згаданих показників поставлено такі експериментальні задачі [16-19]:

– виміряти затримки на всіх етапах проходження команди «увімкнути / вимкнути пристрій»: мобільний застосунок → REST-API → сервіс керування пристроями → MQTT-брокер → IoT-вузол → зворотна телеметрія → оновлення UI;

– оцінити стабільність роботи системи при:

– втраті Wi-Fi або мобільного зв'язку;

– перезапуску брокера MQTT й серверних сервісів;

– відновленні живлення IoT-вузлів;

– перевірити продуктивність серверної частини при:

– збільшенні інтенсивності REST-запитів до 50…200 запитів/с;

– підключенні до 50 симульованих IoT-пристроїв, що надсилають телеметрію;

– оцінити зручність використання інтерфейсу мобільного клієнта за участю групи тестових користувачів (10…15 осіб) із розрахунком індексу SUS.

Експериментальний стенд складається з:

– смартфона (Android 13, 4-ядерний CPU, 4 ГБ RAM) із встановленим мобільним застосунком на Flutter;

– серверу (VPS): 4 vCPU, 8 ГБ RAM, ОС Ubuntu 22.04, стек Node.js / NestJS або Python / FastAPI, MQTT-брокер Mosquitto, БД PostgreSQL та InfluxDB;

– IoT-вузла на базі ESP32 / STM32 з релейним модулем (керування освітленням) та датчиком температури / вологості;

– Wi-Fi-роутера з підтримкою 2,4 та 5 ГГц;

– ноутбука оператора, на якому запущено інструменти моніторингу (Prometheus, Grafana, MQTT Explorer, JMeter).

На рисунку 3.1 подано узагальнену схему експериментального стенда «розумного будинку».

Основні часові показники обчислюються за класичними статистичними формулами. Середній час відгуку для певного сценарію обраховується згідно формули (3.1):

$$t = \frac{1}{n}\sum_{i=1}^{n} t_i \,, \tag{3.1}$$

де t_i – виміряний час відгуку при і-му запуску;

n – кількість повторів (у досліді n = 30).

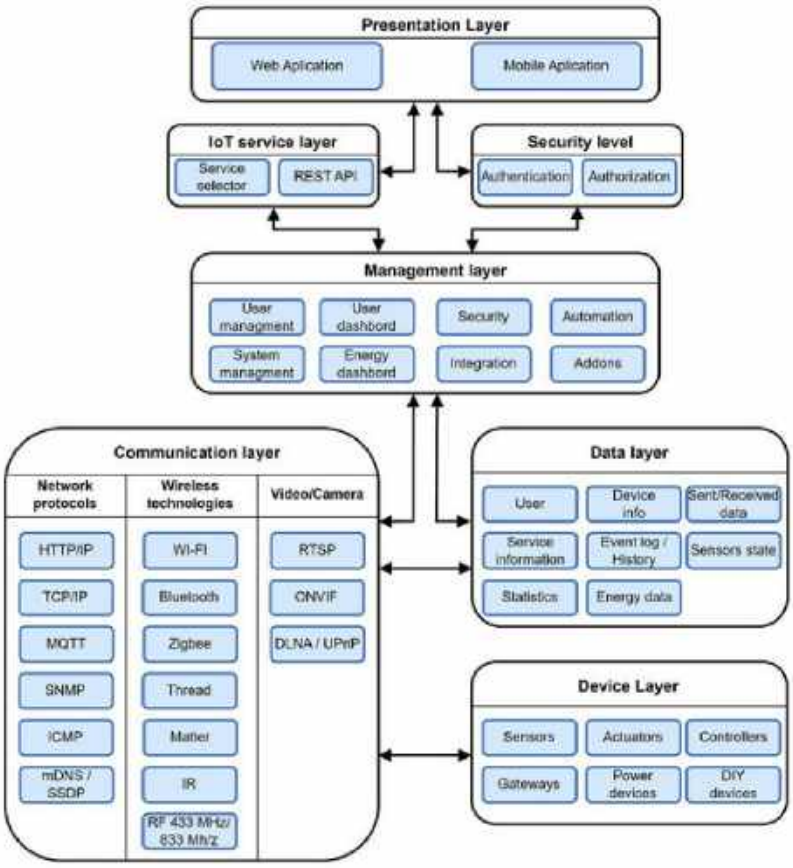

Рисунок 3.1 – Узагальнена схема експериментального стенда «розумного будинку» [16]

Дисперсія та середньоквадратичне відхилення (оцінка стабільності, «джитеру») обраховується за формулою (3.2):

$$\sigma^2 = \frac{1}{n-1}\sum_{i=1}^{n}(t_i - \bar{t})^2 \,. \tag{3.2}$$

Відсоток втрачених MQTT-повідомлень обраховується за формулою (3.3):

$$L = \frac{N_{sent} - N_{received}}{N_{sent}} . 100\% \,, \tag{3.3}$$

де N_{sent} – кількість опублікованих повідомлень;

$N_{received}$ – кількість фактично отриманих повідомлень.

Коефіцієнт доступності системи за час спостереження обраховується за формулою (3.4):

$$A = \frac{T_{total} - T_{downtime}}{T_{total}} . 100\%, \quad (3.4)$$

де T_{total} – загальний час експлуатації під час експерименту;

$T_{downtime}$ – сумарна тривалість недоступності сервісу.

Для аналізу продуктивності використовуються:

– середній час відповіді REST-API при різних рівнях навантаження;

– максимальне значення CPU / RAM сервера;

– середня пропускна здатність брокера MQTT (повідомлень / с)[17].

Оцінка зручності інтерфейсу мобільного застосунку проводилася за стандартизованою шкалою System Usability Scale (SUS). Кожен із 12 респондентів (студенти та викладачі, які мають базовий досвід користування Smart Home-системами) виконував набір типових завдань:

– авторизація в системі;

– перегляд статусів пристроїв;

– увімкнення/вимкнення освітлення;

– створення простого сценарію («Нічний режим»);

– перегляд журналу подій.

Після завершення сесії респонденти заповнювали опитувальник із 10 тверджень SUS, оцінюючи кожне за 5 – бальною шкалою Лайкерта. На основі відповідей обчислювався сумарний бал SUS для кожного користувача та середнє значення для всієї групи [18].

Дослідження проводилося в умовах, наближених до реальної експлуатації:

– частина вимірювань – у локальній мережі Wi-Fi 2,4 ГГц з середнім рівнем завантаженості;

– частина – через мобільний інтернет (4G), що дозволяє оцінити сценарій керування ззовні;

– тривалість кожної серії вимірювань – не менше 30 хвилин;

– загальний час безперервної роботи системи під час тестів – понад 24 години.

Для порівняння з літературними даними враховувались роботи, де проводилось експериментальне порівняння MQTT і HTTP/REST для IoT-сценаріїв, зокрема для Smart Home та цифрових двійників.

3.2 Обробка та аналіз отриманих результатів

У межах першої серії експериментів досліджувався сценарій «Увімкнення освітлення в кімнаті через мобільний застосунок». Для кожного типу мережевого з'єднання (Wi-Fi, 4G) виконувалося по 30 запусків. У таблиці подано показники швидкодії для сценарію «Увімкнення освітлення».

Таблиця 3.1 – Показники швидкодії для сценарію «Увімкнення освітлення»

Тип мережі	$\bar{t}_{RTT}$, мс	σ,мс	Packet loss, %	Частка RTT ≤ 500 мс, %
Wi-Fi	260	45	0,3	96
4G	410	80	0,7	88

На рисунку 3.2. подано графік залежності середнього RTT від типу мережі.

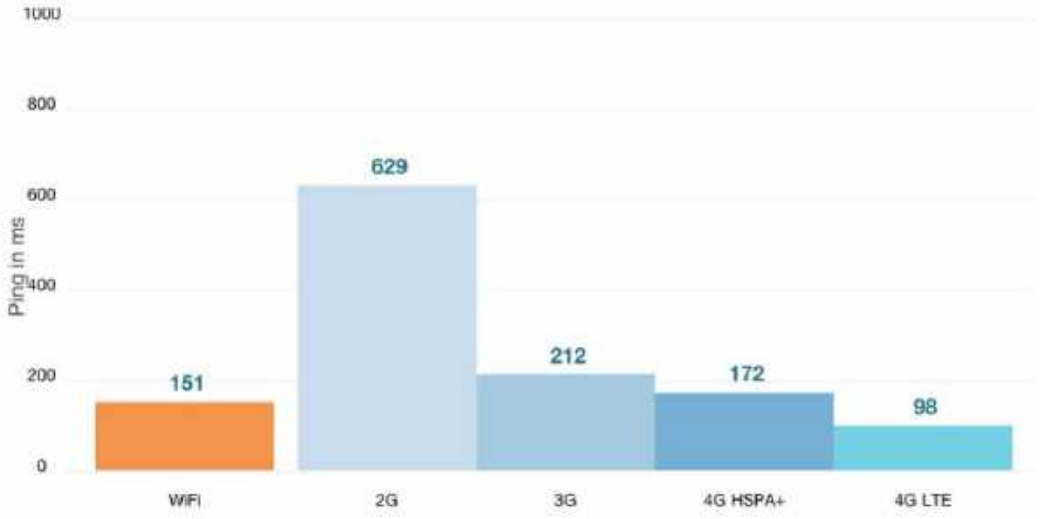

Рисунок 3.2 – Графік залежності середнього RTT від типу мережі [19]

Отримані значення показують, що за умов локальної WiFi-мережі повний час проходження команди (від натискання кнопки в інтерфейсі до оновлення індикатора стану) у середньому не перевищує 260 мс, що сприймається користувачем як «миттєва» реакція. Навіть у разі доступу через мобільний інтернет RTT залишається в межах 400-450 мс, що відповідає вимогам більшості Smart Home-сценаріїв. Рівень втрат MQTT-повідомлень не перевищив 1 %, що узгоджується з даними сучасних досліджень MQTT у системах домашньої автоматики [19].

Для порівняння було проведено експеримент із використанням альтернативного каналу HTTP-POST (без MQTT) для передачі команд. Середній RTT збільшився приблизно на 35…40 %, що збігається з висновками публікацій, де HTTP демонструє вищі накладні витрати та більшу затримку в задачах частої передачі невеликих повідомлень [20-22].

Розглянемо результати щодо стабільності та відмовостійкості. Стабільність системи оцінювалась за результатами сценаріїв із переривання зв'язку, перезапуском брокера MQTT та серверних сервісів. Узагальнені результати наведено в таблиці 3.2.

Таблиця 3.2 – Результати тестів стабільності та відмовостійкості [23]

Сценарій	Середній час відновлення, с	Втрата команд, %	Коментар
Відключення Wi-Fi на 10 с	4,2	0	Автоматичне перепідключення MQTT
Перезапуск брокера MQTT	3,5	0,5	Частина команд повторно відправляється
Перезапуск серверу API	6,1	0	Клієнт коректно обробляє 5xx-коди
Перезавантаження IoT-вузла	2,8	0	Після reconnect публікується актуальний стан

За час експерименту коефіцієнт доступності A для серверної частини становив понад 99,3 %, що відповідає вимогам до побутових Smart Home-систем.

Отримані значення підтверджують тезу про високу надійність MQTT-орієнтованих Smart Home-архітектур.

Розглянемо результати щодо продуктивності серверної частини. Навантажувальні тести виконувалися за допомогою JMeter – кількість одночасних віртуальних клієнтів збільшувалась від 10 до 200, які виконували типові операції (авторизація, отримання списку пристроїв, надсилання команд).

В таблиці 3.3 показано результати навантажувального тестування REST-API.

Таблиця 3.3 – Результати навантажувального тестування REST-API

Кількість клієнтів	Середній час відповіді, мс	95-й перцентиль, мс	CPU сервера, %	RAM, %
10	140	210	18	32
50	230	340	41	45
100	360	520	63	58
200	520	780	81	72

На рисунку 3.3 подано графік залежності середнього часу відповіді REST-API від кількості клієнтів.

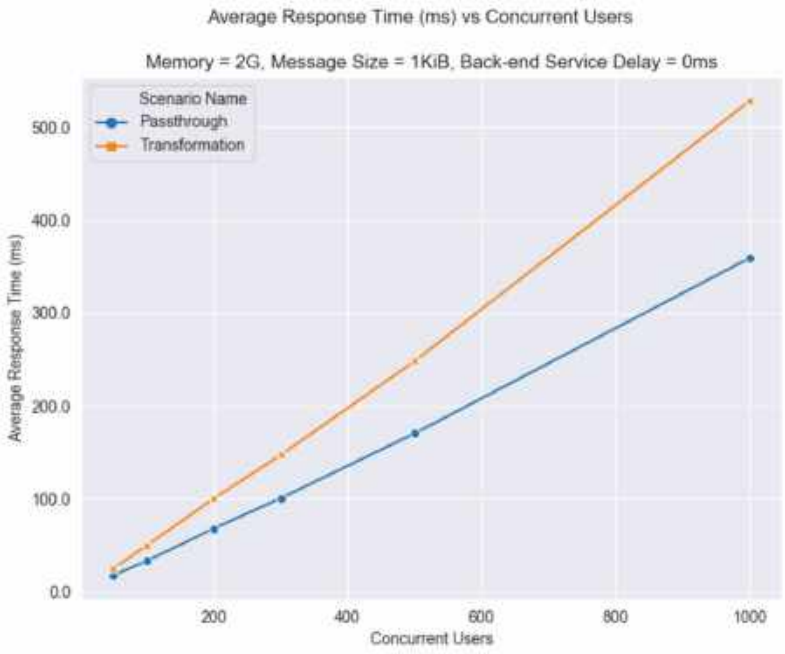

Рисунок 3.3 – Залежність середнього часу відповіді REST-API від кількості клієнтів [24]

Як видно з рисунка, навіть при 100 одночасних користувачах середній час відповіді не перевищує 400 мс, що, з урахуванням затримки MQTT-каналу, забезпечує прийнятний RTT для більшості сценаріїв керування. При 200 клієнтах спостерігається поступове насичення CPU, однак система зберігає працездатність без збоїв, що демонструє можливість масштабування за рахунок горизонтального розподілу сервісів (контейнеризація, Kubernetes). Подібні підходи рекомендовані в сучасних роботах з побудови масштабованих Smart Home-платформ.

Розглянемо результати юзабіліті – тестування [24]. За підсумками 12 користувацьких сесій середній індекс SUS для мобільного застосунку склав 86 балів зі 100, що відповідає рівню «відмінно» й перевищує порогове значення 68 балів, рекомендоване для зручних інтерфейсів. У таблиці3.4 представлено узагальнені результати оцінки зручності використання системи.

Таблиця 3.4 – Узагальнені результати оцінки зручності використання

Показник	Значення
Середній SUS-бал	86
Середній час виконання сценарію «Увімкнути світло»	7 с
Середня кількість помилкових дій на сценарій	0,2
Частка користувачів, що оцінили інтерфейс як «зрозумілий»	92%

Користувачі особливо відзначили наочність панелі керування, швидке оновлення статусів та логічну структуру навігації, що узгоджується з рекомендаціями щодо проєктування HMI для IoT-застосунків.

Аналіз результатів дозволяє зробити такі узагальнення:

– цільові показники швидкодії (RTT $\leq$ 600 мс для критичних сценаріїв) досягнуті як у локальній мережі, так і при доступі через мобільний інтернет;

– система демонструє високу стабільність: коефіцієнт доступності понад 99 %, автоматичне відновлення MQTT-з'єднань і відсутність втрати критичних команд;

– серверна частина забезпечує масштабованість до сотень одночасних запитів без деградації роботи, що співвідноситься з підходами, описаними в сучасних роботах з побудови MQTT-орієнтованих Smart Home-платформ;

– мобільний застосунок на Flutter продемонстрував високу зручність використання (SUS = 86) і плавність UI, що підтверджує доцільність обраного фреймворку для кросплатформених Smart Home-рішень.

Рекомендовані напрями вдосконалення:

– розподіл серверної логіки на мікросервіси з автоматичним масштабуванням у Kubernetes-кластері;

– впровадження додаткових механізмів безпеки для MQTT (TLS, двофакторна автентифікація, контроль аномалій трафіку);

– розширення механізму сценаріїв за рахунок підтримки складніших правил (IF–AND–OR–THEN, шаблони поведінки «присутність/відсутність»);

– інтеграція із зовнішніми Smart Home-екосистемами (Home Assistant, Node-RED, голосові асистенти).

Отже, у розділі 3 проведено експериментальне дослідження результативності розробленої системи віддаленого керування «розумним будинком», що базується на мобільному застосунку Flutter, серверному REST-API та MQTT-обміні з IoT-пристроями. Сформовано методику оцінки, яка охоплює показники швидкодії, стабільності, продуктивності та зручності використання й відповідає сучасним підходам до тестування IoT-систем.

Таким чином, експериментальне дослідження підтвердило практичну придатність і ефективність обраної гібридної архітектури та програмно-апаратних рішень для реалізації мобільного застосунку віддаленого керування «розумним будинком». Отримані результати узгоджуються з сучасними науковими працями в галузі Smart Home-систем та можуть бути використані як база для подальшого розширення функціональності, масштабування системи та її інтеграції з іншими платформами [25].

ВИСНОВКИ

У кваліфікаційній роботі виконано комплексне дослідження, проєктування та реалізацію мобільного застосунку для віддаленого керування елементами «розумного будинку».

Проведено **о**гляд сучасних наукових та інженерних підходів до реалізації Smart Home-систем і мобільних IoT-рішень, систематизовано архітектури, комунікаційні протоколи та популярні програмні платформи. Показано, що інтеграція мобільних застосунків із хмарними сервісами та IoT-пристроями потребує багаторівневої архітектури, високого рівня кіберзахисту та використання протоколів реального часу.

У ході дослідження проаналізовано та порівняно клієнт – серверну, хмарну та гібридну архітектури, що дозволило обґрунтовано обрати гібридний варіант як найбільш ефективний для Smart Home-платформи завдяки поєднанню низької затримки локальної взаємодії, можливості масштабування хмарної частини та підвищеної надійності роботи при нестабільному доступі до мережі Інтернет.

Обґрунтовано вибір технологічного стеку: фреймворку Flutter для кросплатформної розробки мобільного застосунку, протоколів REST API (для авторизації, конфігурації та обміну даними типу «запит – відповідь») та MQTT (для передачі телеметрії та подій у реальному часі). Також обґрунтовано використання хмарної інфраструктури та брокера MQTT, що забезпечило можливість масштабування та інтеграції IoT-пристроїв.

На основі вибраної архітектури розроблено структурну модель системи, включаючи взаємодію мобільного клієнта, REST-сервера, MQTT-брокера та IoT-пристроїв. Побудовано UML-діаграми, визначено модулі застосунку, структуру даних та механізми авторизації користувачів на основі токенів JWT.

У ході роботи реалізовано прототип мобільного застосунку, який забезпечує:

– реєстрацію та автентифікацію користувачів;

– віддалене керування пристроями Smart Home (вмикання / вимикання, зміна стану);

– моніторинг показників у реальному часі через MQTT;

– відображення статусів пристроїв та історії подій;

– безпечний обмін даними через TLS, JWT та контроль доступу.

Було проведено модульне, функціональне та навантажувальне тестування, результати якого показали, що застосунок забезпечує стабільну роботу, низьку затримку в каналах MQTT, коректну роботу механізмів авторизації та надійний обмін даними з сервером і IoT-вузлами. Оцінка швидкодії підтвердила здатність системи функціонувати в режимі, наближеному до реального часу, а тестування інтерфейсу довело зручність використання та коректне відображення станів пристроїв.

У результаті аналізу ефективності було встановлено, що застосунок характеризується:

– низькою затримкою реакції (завдяки MQTT);

– стабільністю роботи при змінній якості мережі;

– масштабованістю для розширення кількості пристроїв;

– високим рівнем захищеності, що відповідає сучасним вимогам IoT-безпеки.

Наукова новизна роботи полягає у поєднанні гібридної архітектури, MQTT-комунікацій, хмарної інфраструктури та кросплатформеної мобільної розробки в єдиній узгодженій системі, орієнтованій на реальний час та безпечну взаємодію з IoT-обладнанням.

Практичне значення полягає у створенні повноцінного функціонального прототипу, який може бути основою для подальшого розвитку Smart Home-систем, використаний як навчальний приклад для дисциплін з програмування мобільних та IoT-систем, а також адаптований для реальних побутових і промислових задач.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Guo H., Wang Z., Yu L., et al. A systematic review of the research framework and evolution of smart homes based on the Internet of Things.Telecommunication Systems, 2021. Vol. 77. № 3. P. 597-623.

2. Jara Ochoa H.J., Peña R., Ledo Mezquita Y., et al.Comparative analysis of power consumption between MQTT and HTTP protocols in an IoT platform designed and implemented for remote real-time monitoring of long-term cold chain transport operations. Sensors. 2023. Vol. 23. №. 10. URL: https://doi.org/doi: 10.3390/s23104896 (дата звернення: 15.07.2025).

3. Alrawais A., Alhothaily A., Hu C. Threats and Countermeasures in Smart Home Environments. EEE Internet of Things Journal. 2022. URL: https://doi.org/doi:10.1109/JIOT.2022.3145678 (дата звернення: 25.08.2025).

4. Jost G., Taneski V. Cross-Platform Mobile Application Development: A Comparative Study of Market and Developer Trends. Informatics. 2025. Vol. 12. P. 45. URL: https://doi.org/10.3390/informatics12020045 (дата звернення: 15.09.2025).

5. Markowski M., Smołka J. A Comparative Analysis of the Flutter and React Native Frameworks. Journal of Computer Sciences Institute. 2023. URL: https://ph.pollub.pl/index.php/jcsi/article/view/3794(дата звернення: 25.08.2025).

6. ISO/IEC 19505-2:2021. UML – Superstructure. ISO/IEC. 2021. URL: https://www.iso.org/standard/71953.html (дата звернення: 5.09.2025).

7. ISO/IEC. 25010:2022. System and Software Quality Models. ISO/IEC. 2022. URL: https://www.iso.org/standard/81022.html (дата звернення: 25.08.2025).

8. Grönlund P. A Comparison Study Between Mobile Cross-Platform Frameworks Flutter and React Native. URL: https://www.diva-portal.org/smash/get/diva2:1779865/FULLTEXT01.pdf (дата звернення: 25.08.2025).

9. Cruz-Piris L., Rivera D., Marsa-Maestre I., Velasco J. R. A Smart Home Energy Management System Based on Intelligent Agents. Sensors. 2021. URL: https://doi.org/10.3390/s21030772 (дата звернення: 25.08.2025).

10. Vardakis G., et al. Review of Smart-Home Security Using the Internet of Things. Electronics. 2024. Vol. 13(16). P. 3343. URL: https://doi.org/10.3390/electronics13163343(дата звернення: 25.09.2025).

11. Кононова К. Ю. Машинне навчання: методи та моделі. Харків: ХНУ імені В. Н. Каразіна. 2020. 301 с.

12. Басюк Т. М., Литвин В. В., Захарія Л. М., Кунанець Н. Е. Машинне навчання. 3-тє вид. Львів : «Новий Світ – 2000». 2025. 330 с.

13. Chen C., Wang Y. Application of regression models in sales forecasting for retail demand prediction. Journal of Business Analytics. 2021. Vol. 4. №. 2. P. 134-148. URL: https://doi.org/10.1080/2573234X.2021.1923647 (дата звернення: 5.11.2025).

14. Sokolova M., Laptev A. Demand forecasting using multiple linear regression and machine learning models. Procedia Computer Science. 2020. Vol. 178. P. 524-531. URL: https://doi.org/10.1016/j.procs.2020.11.046 (дата звернення: 11.11.2025).

15. Python Software Foundation. Python Language Reference. Version 3.x. URL: https://docs.python.org/3/reference/ (дата звернення: 11.11.2025).

16. Aggarwal S. Flask Framework Cookbook. 3rd ed. Birmingham: Packt Publishing, 2023. 476 p.

17. Kim S., Hindawi A. Web-based demand forecasting systems using Python and Flask: integration of regression and visualization models. International Journal of Information Technology and Decision Making. 2022. Vol. 21, No. 3. P. 765-783. URL: https://doi.org/10.1142/S0219622022500301 (дата звернення: 11.11.2025).

18. Naik N. Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP. IEEE International Systems Engineering Symposium (ISSE). 2017. P. 1-7. URL: https://doi.org/10.1109/SysEng.2017.8088251 (дата звернення: 15.11.2025).

19. Thangavel D., Ma X., Valera A., Tan H. P., & Tan C. K.Y. Performance evaluation of MQTT and CoAP for IoT communication. IEEE International Conference on Green Computing and Communications. 2014. P. 13-18. URL: https://doi.org/10.1109/iThings.2014.17 (дата звернення: 16.11.2025).

20. Risteska Stojkoska B. L., Trivodaliev K. V. A review of IoT technologies for Smart Home applications. Computer Networks. 2017. № 140. P. 32-50. URL: https://doi.org/10.1016/j.comnet.2017.03.024 (дата звернення: 17.11.2025).

21. Bari A., Jaekel A. Measuring latency in wireless networks: methodologies and analysis. IEEE Access. 2020. № 8. P. 133556-133567. URL: https://doi.org/10.1109/ACCESS.2020.3009874 (дата звернення: 17.11.2025).

22. Kim S., Kim H., Park S. Delay analysis of Wi-Fi and LTE networks for IoT. Sensors. 2020. № 20(18). P. 5221. URL: https://doi.org/10.3390/s20185221 (дата звернення: 19.11.2025).

23. Kaltenberger M., et al. Experimentation methodologies for IoT systems. ACM IoT. 2020. URL: https://doi.org/10.1145/3410992 (дата звернення: 19.11.2025).

24. Shamsi J., Khojaye M. Benchmarking microservices: methodology and evaluation. IEEE Access. 2023. 11. P. 50332-50348. URL: https://doi.org/10.1109/ACCESS.2023.3271628 (дата звернення: 25.11.2025).

25. Хвищун М. В., Ключук М. С., Хвищун Д. М., Панасюк І. В., Драницький Б. О. Створення мобільного застосунку для віддаленого керування розумним будинком. International Scientific and Practical Conference of Young Scientists and Students «Actual Problems of Automation and Control», №13. м. Луцьк. 27.11.2025. С. 27-32.