

БАКАЛАВРСЬКА РОБОТА

РБ. ІІ - 97.00.00.000 ІЗ

Група ІІ-22-4

Захарченко Богдан

2026

Івано-Франківський національний технічний університет нафти і газу

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Захарченко Богдан Олександрович

(прізвище, ім'я, по батькові)

УДК 004.942

(індекс)

БАКАЛАВРСЬКА РОБОТА

Проектування IoT-платформи для “розумного будинку”

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121– Інженерія програмного забезпечення

(шифр і назва спеціальності)

Робота містить результати власних досліджень, використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело:

Здобувач освітнього ступеня Захарченко Б.О.
(підпис, ініціали та прізвище здобувача)

Науковий керівник Лютак Ігор Зіновійович, д.т.н., професор
(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту

Завідувач кафедри

доц. Бандура В.В.
(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківськ – 2026

Івано-Франківський національний технічний університет нафти і газу
Інститут, факультет інформаційних технологій
Кафедра інженерії програмного забезпечення
Ступінь вищої освіти бакалавр
Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

Зав. кафедрою ІІЗ
доцент. В.В. Бандура

“ ____ ” _____ 2026 р.

ЗАВДАННЯ

НА ДИПЛОМНУ РОБОТУ БАКАЛАВРА СТУДЕНТОВІ

Захарченко Богдан Олександрович

(прізвище, ім'я, по-батькові)

1. Тема проекту (роботи) "Проектування IoT-платформи для "розумного будинку"

керівник проекту (роботи) Лютак Ігор Зіновійович, д.т.н., професор

затверджені наказом вищого навчального закладу від “ 21 ” квітня 2026 р. № 179/7

2. Строк подання студентом проекту (роботи) 10 червня 2026р.

3. Вихідні дані до проекту (роботи) Результати і матеріали отримані під час проходження переддипломної практики

4. Зміст розрахунково - пояснювальної записки(перелік питань, які потрібно розробити)

1 Вступ до проблеми

2 Огляд існуючих концепцій , рішень та сервісів в даній області

3 Побудова моделі та реалізація власного оригінального рішення вибраними засобами

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

1 Діаграма послідовності процесу в протоколі MQTT (рис. 1.1, ст. 14)

2 Реалізація інтерфейсу MqttCallbackExtended (рис. 4.3, ст. 36)

3 Структурна схема дерева кінцевих точок REST API (рис. 4.5, ст. 40)

4 Екран авторизації користувача (рис. 4.6, ст. 41)

5 Діаграма розгортання компонентів IoT-платформи «HataPlus» (рис. 4.18, ст. 54)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

2. Дата видачі завдання 10 січня 2026 р.

Керівник _____ Лютак І. З.
(підпис)

Завдання прийняв до виконання _____ Захарченко Б.О.
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту	Примітка
1	Визначення та обґрунтування теми роботи	15.05.2026	виконано
2	Огляд існуючих концепцій, рішень та сервісів в даній області	20.05.2026	виконано
3	Побудова моделі або алгоритму власного рішення	27.05.2026	виконано
4	Документування реалізації власного оригінального рішення вибраними засобами	01.06.2026	виконано
5	Оформлення пояснювальної записки кваліфікаційної роботи	09.06.2026	виконано

Студент _____ Захарченко Б. О.
(підпис)

Керівник роботи _____ Лютак І.З
(підпис)

АНОТАЦІЯ

Бакалаврська робота містить 77 сторінок, 24 рисунків, список використаних джерел із 40 найменуваннями.

Метою роботи є проектування та реалізація програмно-апаратної IoT-платформи розумного будинку «HataPlus» із забезпеченням захищеного віддаленого доступу та кросплатформеної взаємодії.

Об'єкт дослідження: процес створення розподілених програмно-апаратних комплексів Інтернету речей для автоматизації житлового простору.

Предмет дослідження: архітектурні моделі, кросплатформені інструменти розробки, протоколи асинхронної передачі даних та засоби безпечного розгортання локального сервера.

У першому розділі проаналізовано стан розвитку технологій Інтернету речей, виконано порівняльний огляд існуючих платформ.

У другому розділі розкрито аспекти проектування архітектури екосистеми, описано розробку асинхронного сервера на базі Ktor.

У третьому розділі висвітлено реалізацію вбудованого програмного забезпечення мікроконтролерів ESP32 на базі FreeRTOS, побудову топології розгортання із впровадженням Cloudflare Tunnel для обходу NAT.

Висновки роботи містять основні результати дослідження, оцінку ефективності функціонування створеного комплексу та підтвердження його відповідності критеріям технічного завдання.

Отримані результати включають працездатний прототип IoT-платформи «HataPlus», який забезпечує збір телеметрії в реальному часі за протоколом MQTT, керування фізичними реле, захищений віддалений доступ.

КЛЮЧОВІ СЛОВА: ІНТЕРНЕТ РЕЧЕЙ, РОЗУМНИЙ БУДИНОК, KOTLIN MULTIPLATFORM, KTOR, MQTT, МІКРОКОНТРОЛЕР ESP32, CLOUDFLARE TUNNEL.

ANNOTATION

The bachelor's thesis consists of 77 pages, 24 figures, and a list of references with 40 titles.

The purpose of the work is the design and implementation of a software-hardware IoT platform for a smart home «HataPlus» with the provision of secure remote access and cross-platform interaction.

Object of research: the process of creating distributed software-hardware complexes of the Internet of Things for the automation of residential space.

Subject of research: architectural models, cross-platform development tools, protocols for asynchronous data transmission, and means of secure deployment of a local server.

In the first chapter, the state of development of Internet of Things technologies was analyzed, and a comparative review of existing platforms was performed.

In the second chapter, the aspects of designing the architecture of the ecosystem are disclosed, and the development of an asynchronous server based on Ktor is described.

In the third chapter, the implementation of embedded software for ESP32 microcontrollers based on FreeRTOS is highlighted, as well as the construction of a deployment topology with the introduction of Cloudflare Tunnel for NAT traversal.

The conclusions of the work contain the main results of the research, an assessment of the efficiency of the functioning of the created complex, and confirmation of its compliance with the criteria of the technical specification.

The obtained results include a working prototype of the IoT platform «HataPlus», which provides real-time telemetry collection using the MQTT protocol, control of physical relays, and secure remote access.

KEYWORDS: INTERNET OF THINGS, SMART HOME, KOTLIN MULTIPLATFORM, KTOR, MQTT, ESP32 MICROCONTROLLER, CLOUDFLARE TUNNEL.

ЗМІСТ

ВСТУП.....	8
РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ТА АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ. 10	
1.1 Рівні архітектури Інтернету речей та концептуальні засади систем «розумного будинку».....	10
1.2 Стан та тенденції сучасного розвитку автономних та енергоефективних локальних систем автоматизації.....	11
1.3 Огляд комерційних платформ автоматизації та обґрунтування концепції «Hata Plus».....	14
1.4 Висновки по розділу	16
РОЗДІЛ 2. АНАЛІЗ ВИМОГ ТА ПОСТАНОВКА ЗАДАЧІ ПРОЄКТУВАННЯ «НАТА PLUS»	16
2.1 Огляд комерційних платформ автоматизації та виявлення їхніх архітектурних недоліків.	18
2.2 Формулювання функціональних та нефункціональних вимог до IoT-платформи	20
2.3 Постановка задачі інженерного проєктування компонентів локально-автономної системи.....	22
2.4 Висновок по розділу	24
РОЗДІЛ 3. МЕТОДОЛОГІЯ, ТЕХНОЛОГІЇ ТА ПРОЄКТУВАННЯ СИСТЕМИ.....	26
3.1 Об’єкт, предмет та інструментарій аналізу взаємодії системних компонентів	26

					<i>РБ.ІП – 97.00.00.000 ПЗ</i>			
<i>Змн.</i>	<i>Арк.</i>	<i>№ докум.</i>	<i>Підпис</i>	<i>Дата</i>	<i>Проекткування IoT-платформи для «розумного будинку» Пояснювальна записка</i>	<i>Лім.</i>	<i>Арк.</i>	<i>Акрушів</i>
<i>Розроб.</i>		<i>Захарченко Б.О.</i>						
<i>Перевір.</i>		<i>Лютак І. З.</i>					6	86
<i>Реценз.</i>		<i>Мельник В. Д.</i>				<i>ІФНТУНГ, ІП-22-4</i>		
<i>Н. Контр.</i>		<i>Піх М. М</i>						
<i>Затверд.</i>		<i>Бандура В. В.</i>						

3.2 Алгоритми, методики та структури даних	27
3.3 Обґрунтування вибору технологічного стеку, інструментальних засобів та середовищ розробки.....	29
3.4 Висновки по розділу	332
РОЗДІЛ 4. РЕАЛІЗАЦІЯ ПРОЄКТУ НАТА PLUS.....	3434
4.1 Опис розробки програмного коду	34
4.2 Використані алгоритми та структури даних	51
4.3 Інтеграція та взаємодія програмних модулів на етапі розробки.....	53
4.4 Висновки по розділу	56
РОЗДІЛ 5. ТЕСТУВАННЯ ТА ВПРОВАДЖЕННЯ	58
5.1 Стратегії та методи тестування	58
5.2 Результати тестування системи	59
5.3 Аналіз ефективності впровадження	68
5.4 Висновки по розділу	70
ВИСНОВКИ	72
ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ	74
ДОДАТКИ	
БІБЛІОГРАФІЧНА ДОВІДКА	

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		7

ВСТУП

У сучасному світі інформаційні технології стають невід'ємною частиною всіх сфер людської діяльності, зокрема управління житловим простором та забезпечення комфорту. Економічні, соціальні та технологічні зміни сприяють зростанню попиту на системи, що базуються на концепції Інтернету речей (IoT) та впровадженні інтелектуальних пристроїв у побут. Вітчизняний ринок систем автоматизації потребує інноваційних рішень, які враховували б сучасні тенденції енергонезалежності, кросплатформеності та локальної безпеки даних.

Актуальність теми зумовлена необхідністю створення нової IoT-платформи для «розумного будинку», яка забезпечить зручний, швидкий та інтуїтивно зрозумілий інтерфейс, здатний відповідати сучасним викликам. Незважаючи на наявність відомих комерційних платформ, таких як Google Home, Xiaomi Mi Home, Tuya Smart або Apple HomeKit, є потенціал для покращення процесів керування та моніторингу через впровадження нових підходів. Сучасні наявні платформи часто не задовольняють потреби деяких користувачів через жорстку прив'язку до хмарних серверів, відсутність гнучкої локальної автоматизації та високу вартість інтеграції пристроїв від різних виробників. В умовах мережових обмежень особливо важливою стає можливість забезпечення безперервності локальних бізнес-процесів будинку, захисту персональних даних користувачів від стороннього втручання та мінімізації трафіку за рахунок оптимізованих структур даних.

Метою роботи є розробка комплексної IoT-платформи для «розумного будинку», яка зможе підвищити ефективність моніторингу та керування побутовими пристроями у реальному часі, зокрема через використання сучасного асинхронного серверного фреймворку, енергоефективних протоколів передачі даних та кросплатформених інтуїтивно зрозумілих інтерфейсів.

Завданнями дослідження полягає в вивченні сучасних тенденцій та викликів у сфері Інтернету речей, визначенні ключових потреб користувачів щодо автоматизації приміщень, виявленні основних архітектурних недоліків

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		8

існуючих комерційних платформ, розробка нової системи з урахуванням отриманих даних, а також комплексне тестування і впровадження платформи.

Об'єктом дослідження є процеси розробки програмного забезпечення для створення багаторівневої IoT-платформи «розумного будинку», що включають аналіз вимог, проектування архітектури, програмну реалізацію серверної та клієнтської частин, тестування та впровадження системи для забезпечення ефективної взаємодії між апаратними модулями й користувачем.

Предметом дослідження є підходи, технології та інструменти, що застосовуються на етапах життєвого циклу розробки IoT-платформи, зокрема аналіз і формалізація вимог користувачів, проектування інтуїтивного інтерфейсу, моделювання компонентів системи, структур даних, а також тестування й оцінка захищеності платформи з урахуванням сучасних викликів. В межах дослідження буде здійснено детальний аналіз сучасних архітектурних рішень Інтернету речей, визначено основні потреби та очікування користувачів систем розумного будинку, а також розглянуто недоліки наявних рішень.

Методи дослідження включають об'єктно-орієнтоване проектування, моделювання баз даних за допомогою, асинхронне програмування, , моніторинг мережевого трафіку за протоколом MQTT, проведення тестування надійності каналів зв'язку, аналіз вразливостей авторизації. Використання цих методів допоможе отримати глибоке розуміння ефективності засобів динамічної обробки даних та захисту платформи у контексті забезпечення стабільної роботи системи.

Розроблена платформа передбачатиме можливість підключення мікроконтролерів ESP32 для збору кліматичних показників, публікацію телеметрії з детальним описом станів, включаючи специфічні атрибути пристроїв та кімнат, а також динамічне керування виконавчими механізмами за розкладом.

Бакалаврська робота містить 77 сторінок, 24 рисунків, список використаних джерел із 40 найменуваннями.

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
						9
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ТА АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

1.1 Рівні архітектури Інтернету речей та концептуальні засади систем «розумного будинку»

На сучасному етапі розвитку інформаційних технологій концепція Інтернету речей (Internet of Things, IoT) виступає одним із ключових драйверів автоматизації побутових та промислових процесів. В інженерії програмного забезпечення під Інтернетом речей розуміють глобальну мережеву інфраструктуру фізичних пристроїв, що мають можливість взаємодіяти між собою та із зовнішнім середовищем за допомогою вбудованих сенсорів, виконавчих механізмів та засобів зв'язку. Концепція «розумного будинку» (Smart Home) є одним із найбільш динамічних прикладних напрямів IoT, яке створює системи керування функціями для забезпечення автоматизованого життя, безпекою, мікрокліматом та мультимедійними пристроями в межах локального простору [5]. Основним завданням таких систем є підвищення рівня комфорту користувачів, оптимізація енерговитрат та забезпечення моніторингу в реальному часі.

Для систематизації процесів збору, обробки та передачі даних в інфраструктурі IoT традиційно застосовують багаторівневий архітектурний підхід. Класична референсна модель включає три основні рівні, де початковим етапом є сенсорний рівень (Perception Layer), відповідальний за безпосередній збір інформації з фізичного світу за допомогою датчиків температури, вологості, руху чи освітленості, а також за виконання безпосередніх команд через реле чи інші виконавчі механізми. Наступним ступенем в ієрархії є мережевий (транспортний) рівень (Network Layer), який забезпечує передачу зібраних телеметричних даних від кінцевих вузлів до центральних серверів чи шлюзів за допомогою бездротових технологій зв'язку. Верхнім рівнем архітектури є прикладний рівень (Application Layer), що охоплює програмні сервіси, бази

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		10

даних та користувацькі інтерфейси, які виконують безпосередню аналітичну обробку отриманої інформації, формують логіку для автоматичного керування та візуалізацію [5].

Специфіка розробки програмного забезпечення для систем «розумного будинку» тісно пов'язана з апаратними обмеженнями кінцевих пристроїв. Сучасні локальні модулі автоматизації базуються на мікроконтролерах, які, на відміну від повноцінних обчислювальних систем, характеризуються суворими обмеженнями щодо обсягу оперативної та енергонезалежної пам'яті, а також порівняно низьку тактову частоту процесора. Яскравим представником цього класу обчислювальних компонентів є сімейство мікроконтролерів ESP32, що має визнання в інженерній проєктах завдяки наявності інтегрованих модулів бездротового зв'язку Wi-Fi та Bluetooth, двоядерній архітектурі та підтримці апаратного шифрування. Проте робота з такими мікроконтролерами вимагає від розробника ретельної оптимізації програмного коду, ефективного управління динамічною пам'яттю та мінімізації обсягу переданих мережею пакетів, що безпосередньо впливає на визначення оптимальних комунікаційних протоколів та архітектури збереження даних.

1.2 Стан та тенденції сучасного розвитку автономних та енергоефективних локальних систем автоматизації

Аналіз сучасних тенденцій у сфері Інтернету речей свідчить про поступове зміщення акцентів від суто хмарних обчислень до крайових обчислень (Edge Computing) та концепції локальної автономії (Local-First IoT). Перші генерації систем «розумного будинку» повністю покладалися на віддалені хмарні сервери для обробки будь-якої події, що створювало критичну залежність від наявності стабільного підключення до мережі Інтернет. Сучасний підхід вимагає, щоб базовий функціонал системи, наприклад, реагування на надходження сигналів датчиків безпеки чи виконання локальних сценаріїв

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		11

автоматизації, здійснювався безпосередньо на крайовому шлюзі всередині приміщення [1], [5]. Це дозволяє забезпечити безперебійне функціонування об'єкта навіть за умов повної відсутності зовнішнього зв'язку із глобальною мережею.

Важливим вектором наукових та інженерних досліджень є підвищення енергоефективності кінцевих пристроїв, бо велика частка сенсорів у системах автоматизації працює від автономних джерел живлення, таких як акумулятори або батареї. Енергоефективність досягається як на апаратному рівні за рахунок використання спеціальних режимів глибокого сну мікроконтролерів (Deep Sleep), так і на програмному рівні через оптимізацію алгоритмів опитування датчиків та зменшення тривалості активної сесії зв'язку. Традиційні підходи до безпеки, що передбачають постійне підтримання важких криптографічних сесій, часто є неефективними для застосування у вбудованих системах із жорсткими обмеженнями енергоспоживання, що стимулює розробку легковажних протоколів безпеки та адаптивних моделей контролю доступу [1].

Паралельно із автономністю гостро постає питання забезпечення інформаційної безпеки локальних мереж IoT. В умовах стрімкого зростання кількості підключених пристроїв класична парадигма периметрової безпеки, яка передбачає автоматичну довіру до всіх внутрішніх компонентів мережі після проходження межі брандмауера, демонструє свою повну неспроможність [1]. Впровадження елементів Zero Trust у локальні системи автоматизації дозволяє ефективно ізолювати скомпрометовані вузли та запобігти горизонтальному поширенню атак всередині домашньої мережі [2].

Організація надійної та швидкої взаємодії між мікроконтролерами та центральним сервером системи автоматизації вимагає детального аналізу існуючих комунікаційних протоколів прикладного рівня. Серед найбільш поширених рішень у веб-інженерії виділяють протоколи HTTP, WebSockets та MQTT, кожен з яких має унікальні архітектурні особливості, переваги та обмеження при використанні в інфраструктурі Інтернету речей. Головним параметром порівняльної характеристики при цьому є величина службових

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		12

заголовків, пакетів, накладні витрати на встановлення мережевого з'єднання, підтримка двостороннього обміну в реальному часі та вимогливість до обчислювальних ресурсів кінцевих вузлів мікроконтролерів.

Протокол HTTP (Hypertext Transfer Protocol) базується на класичній моделі «запит-відповідь» (Request-Response) і використовує TCP як транспортний протокол. Основним недоліком HTTP в контексті передачі телеметрії з мікроконтролерів є високий рівень накладних витрат, оскільки кожен окремий пакет даних вимагає встановлення нового TCP-з'єднання та передачі текстових заголовків великого обсягу, які часто перевищують розмір корисної інформації в кілька разів [3]. Крім того, HTTP за своєю природою не підтримує ініційовану сервером передачу даних без застосування додаткових обхідних механізмів, що робить його малоефективним для сценаріїв, де потрібна миттєва реакція виконавчих пристроїв на команди з сервера.

Технологія WebSockets усуває проблему однонаправленості HTTP, забезпечуючи повнодуплексний двосторонній обмін даними через одне постійне TCP-з'єднання, яке встановлюється шляхом початкового рукостискання (Handshake) [3]. Це дозволяє суттєво знизити затримки при передачі повідомлень та забезпечити взаємодію в реальному часі, що є важливим для відображення графіків аналітики або поточного стану пристроїв у користувацькому додатку. Проте WebSockets вимагає постійного утримання відкритого сокета та обробки стану з'єднання з обох сторін, що створює відчутне навантаження оперативної пам'яті мікроконтролерів, обмежуючи масштабованість системи при підключенні великої кількості сенсорів.

Найбільш адаптованим для потреб Інтернету речей є бінарний протокол MQTT (Message Queuing Telemetry Transport), який функціонує за шаблоном проектування «видавець-підписник» (Publish-Subscribe) через центральний вузол – MQTT-брокер. MQTT має дуже компактний заголовок, мінімальний розмір якого становить усього 2 байти, що мінімізує обсяг мережевого трафіку та навантаження на обчислювальні ядра мікроконтролера. Протокол підтримує три рівні якості обслуговування, які гарантують доставку повідомлень відповідно до

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
						13
Змн.	Арк.	№ докум.	Підпис	Дата		

ступеня критичності даних, Використання топіків для маршрутизації повідомлень дозволяє гнучко масштабувати систему та забезпечувати асинхронну взаємодію між багатьма сенсорами та сервером Ktor, роблячи MQTT оптимальним вибором для збору телеметрії в «Hata Plus».[4]

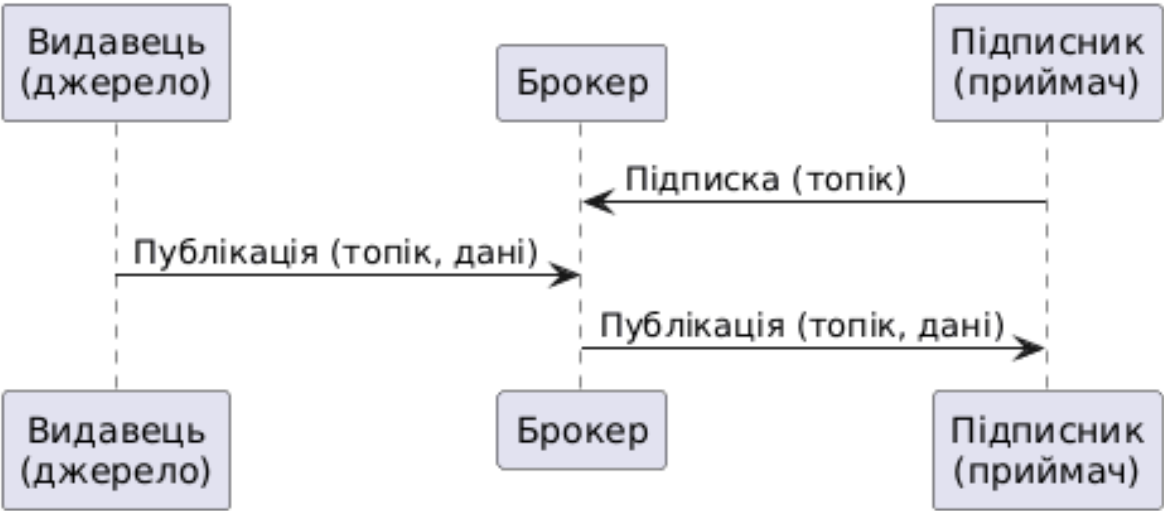


Рисунок 1.1 – Діаграма послідовності процесу в протоколі MQTT

1.3 Огляд комерційних платформ автоматизації та обґрунтування концепції «Hata Plus»

На сучасному ринку рішень для «розумного будинку» домінуючі позиції займають комерційні екосистеми від великих технологічних брендів, серед яких найбільш відомими є Xiaomi Mi Home, Tuuya Smart та Google Home. Ці платформи підтримують велику екосистему сумісного обладнання, простоту налаштування через готові мобільні додатки та розвинені інструменти хмарної автоматизації. Проте глибокий аналіз архітектури цих комерційних систем дозволяє виявити низку суттєвих недоліків, які значно обмежують їхнє застосування в умовах підвищених вимог до надійності, конфіденційності та безпеки обробки даних. Ключовим недоліком згаданих комерційних платформ є їхня тотальна хмарна залежність (Cloud Dependence). Будь-яка команда керування або телеметричний показник спочатку прямує на віддалені сервери компанії й лише після цього

повертається до виконавчого пристрою або відображається в додатку [6]. За умов нестабільного інтернет-зв'язку або збоїв на стороні хмарного провайдера система повністю втрачає працездатність, перетворюючи інтелектуальне середовище на набір ізольованих і некерованих вузлів. Окрім цього, централізована хмарна архітектура створює серйозні проблеми з конфіденційністю персональних даних, оскільки приватна інформація про графік життя користувачів, функціонування побутової техніки, а також трансляцію відеопотоків із камер зберігається на сторонніх ресурсах і може стати об'єктом несанкціонованого аналізу, комерційного використання або глобального витоку [6].

З точки зору інформаційної безпеки комерційні IoT-пристрої часто розробляються без належного криптографічного захисту, функціонують на закритому програмному забезпеченні та використовують пропріетарні вендорні протоколи. Сучасні наукові дослідження підтверджують, що значна частина кібератак на цифрову інфраструктуру розумних середовищ починається саме з компрометації таких уразливих сенсорів та контролерів, які стають початковим вектором для проникнення зломисників у локальну мережу [1]. Дефіцит доступного інструментарію для мікросегментації мережі та реалізації моделей гнучкого розмежування доступу у комерційних системах не дозволяє користувачеві самостійно збудувати надійну локальну екосистему, повністю контрольовану зсередини.

Аналіз експлуатаційних вимог до сучасних комплексів автоматизації житла визначає необхідність впровадження гібридної архітектурної моделі, яка б поєднувала безкомпромісну локальну автономність із можливості гнучкого та безпечного масштабування через глобальну мережу Інтернет. Проєкт «Hata Plus» базується на концепції Local-First, що передбачає розгортання центрального сервера системи на базі фреймворку Ktor та MQTT-брокера всередині локальної мережі, повністю нівелюючи залежність від сторонніх хмарних провайдерів та забезпечуючи збереження даних у локальній СКБД PostgreSQL. Застосування сервісно-орієнтованого підходу на базі модульного асинхронного бекенду у поєднанні з концепцією легковажної багатозадачності на рівні периферійних

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
						15
Змн.	Арк.	№ докум.	Підпис	Дата		

контролерів дозволяє досягти високої живучості системи. Навіть за умов повної деградації зовнішніх каналів зв'язку або критичних збоїв у роботі клієнтських мобільних додатків, внутрішній програмно-апаратний комплекс зберігає абсолютну працездатність, продовжуючи виконувати закладені алгоритми автоматизації, опитувати цифрові датчики та комутувати силові реле за внутрішнім розкладом. Такий підхід забезпечує стабільні показники часу відгуку системи на внутрішні події та гарантує користувачеві повний контроль над його приватними метриками життєдіяльності.

1.4 Висновки по розділу

У першому розділі виконано комплексний аналіз теоретичних основ побудови сучасних розподілених систем Інтернету речей та визначено концептуальні засади функціонального наповнення комплексів «розумного будинку». Детально досліджено референсну трирівневу архітектурну модель, що охоплює сенсорний, мережевий та прикладний рівні, та виділено ключові інженерні обмеження обчислювальної потужності й оперативної пам'яті кінцевих пристроїв на базі мікроконтролерів сімейства ESP32. Встановлено, що специфіка розробки програмного забезпечення для вбудованих систем вимагає від розробника ретельної оптимізації програмного коду, мінімізації мережевого трафіку та ефективного управління динамічною пам'яттю, що безпосередньо впливає на вибір комунікаційних засобів і методів побудови баз даних.

Розглянуто сучасні тенденції розвитку інфраструктури IoT з акцентом на архітектуру крайових обчислень (Edge Computing), енергоефективні режими роботи мікроконтролерів та концепцію локальної автономії, яка забезпечує працездатність системи за умов відсутності зв'язку із глобальною мережею. На основі порівняльного аналізу величини службових заголовків, підтримки дуплексного зв'язку в реальному часі та вимогливості до ресурсів обґрунтовано вибір бінарного протоколу MQTT як найбільш ефективного засобу

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
						16
Змн.	Арк.	№ докум.	Підпис	Дата		

транспортування телеметричних даних. У порівнянні з текстовим протоколом HTTP, який генерує надлишкові накладні витрати на кожне з'єднання, та важковаговим для пам'яті мікроконтролерів протоколом WebSockets, бінарний формат MQTT дозволяє досягти мінімального споживання трафіку та високої швидкості обробки повідомлень. Окрему увагу приділено безпековій парадигмі Zero Trust, впровадження якої дозволяє ізолювати потенційно скомпрометовані вузли домашньої мережі та запобігти горизонтальному поширенню векторів кібератак.

Проведено огляд популярних комерційних екосистем автоматизації від Xiaomi, TuYa та Google, який дозволив виявити їхні суттєві архітектурні недоліки, такі як тотальна хмарна залежність, затримки при передачі пакетів та високі ризики порушення конфіденційності приватних даних користувачів. Виявлені обмеження дозволили повністю сформулювати та обґрунтувати вимоги до проектованої платформи «Hata Plus» на основі парадигми Local-First із використанням серверного стеку Ktor, локальної СКБД PostgreSQL та мобільного клієнта на Jetpack Compose. Проведений аналіз існуючих архітектурних концепцій довів, що створення ізольованого програмно-апаратного комплексу з локальним сервером є єдиним надійним технічним рішенням для подолання недоліків комерційних аналогів. На основі проведеного аналізу сформульовано чітку постановку завдань на розробку компонентної архітектури, програмну реалізацію серверної та клієнтської частин, створення прошивок мікроконтролерів та проведення фінального тестування розробленої системи в реальних умовах експлуатації.

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		17

РОЗДІЛ 2. АНАЛІЗ ВИМОГ ТА ПОСТАНОВКА ЗАДАЧІ ПРОЄКТУВАННЯ «HATA PLUS»

2.1 Огляд комерційних платформ автоматизації та виявлення їхніх архітектурних недоліків.

Сучасний ринок систем автоматизації житлового простору пропонує широкий спектр готових комерційних екосистем, серед яких найбільш масовими та фінансово доступними є платформи Xiaomi Mi Home, Tuuya Smart та Google Home. Попри високий рівень інтеграції, розвинені користувацькі інтерфейси та широку номенклатуру готових виконавчих пристроїв, усім зазначеним комерційним рішенням притаманна низка фундаментальних архітектурних обмежень, які суттєво знижують експлуатаційну надійність, приватність та гнучкість розгортання в межах індивідуальних інженерних проектів.

Ключовим недоліком екосистем Xiaomi та Tuuya є їхня тотальна «хмарна залежність». Більшість логічних зв'язків, сценаріїв автоматизації та процесів обробки подій у цих системах виконуються не на локальних шлюзах користувача, а на віддалених серверах компаній-виробників. У випадку аварії на магістральних каналах зв'язку, тимчасового зникнення інтернет-підключення або проведення технічних робіт у дата-центрах постачальника послуг, локальний сегмент розумного будинку практично повністю втрачає свою функціональність, саме сенсори перестають комутувати реле і користувач втрачає можливість керування пристроями [34]. Такий жорсткий зв'язок із зовнішніми веб-службами перетворює локальну обчислювальну мережу користувача на пасивного транслятора сигналів, повністю позбавленого інтелектуального компонента прийняття рішень у критичні моменти.

Окрім ризиків повної відмови, маршрутизація локального трафіку через географічно віддалені хмарні сервери призводить до появи відчутних часових затримок при передачі телеметричних пакетів та команд керування [35]. Час від

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		18

моменту фіксації події датчиком (наприклад, виявлення руху) до замикання контактів відповідного виконавчого реле може становити від кількох секунд до хвилини, що є абсолютно неприпустимим для систем критичної автоматизації чи контурів безпеки. Наявність багатьох проміжних вузлів маршрутизації та залежність від поточної пропускної здатності каналів провайдера роблять поведінку системи недетермінованою, що прямо суперечить базовим засадам побудови надійних інженерних комплексів реального часу.

Другим критичним аспектом є закритість архітектури та обмеженість програмних інтерфейсів. Платформи типу Google Home розраховані на роботу виключно з сертифікованими партнерськими пристроями через пропрієтарні протоколи обміну. Це унеможливорює або критично ускладнює безпосередню інтеграцію саморобних (DIY) енергоефективних мікроконтролерів, таких як ESP32, без використання складних проміжних програмних прошарків або сторонніх платформ-інтеграторів. Виробники свідомо обмежують доступ до низькорівневих API, змушуючи споживача купувати лише брендову периферію. Це штучно звужує поле для інженерного маневру, блокує можливості гнучкої кастомізації апаратних прошивок і змушує розробника підлаштовуватися під маркетингову політику конкретного вендора.

Окремим деструктивним чинником пропрієтарних рішень виступає повна відсутність прозорості у питаннях збереження та обробки телеметричних даних користувача [36]. Збір детальної інформації про мікроклімат, цикли увімкнення побутових приладів та геопозицію власників дозволяє транснаціональним компаніям формувати точні поведінкові профілі користувачів без їхньої явної згоди. Крім того, монолітна прив'язка до закритих серверів унеможливорює проведення аудиту безпеки кодової бази, що створює приховані вразливості та підвищує ризики несанкціонованого доступу до керуючих контурів будинку ззовні. Ситуація ускладнюється тим, що користувач не має жодних інструментів впливу на політику оновлень віддаленого ПЗ, де раптова зміна алгоритмів на стороні хмари може в один момент порушити роботу раніше налаштованих автоматизацій. Усе це підтверджує інженерну доцільність відмови від готових

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		19

комерційних рішень на користь розробки повністю контрольованої, локально-автономної та відкритої системи автоматизації.

2.2 Формулювання функціональних та нефункціональних вимог до IoT-платформи

Чітке визначення інженерних та експлуатаційних меж екосистеми «HataPlus» потребує детального розподілу проектних завдань на рівні функціональних можливостей системи та технічних обмежень її реалізації. На основі проведеного аналізу недоліків комерційних аналогів сформуємо вимоги до архітектури, яка розробляється.

Функціональні вимоги визначають безпосередні обов'язки, дії та сервіси, які програмно-апаратний комплекс повинен виконувати для задоволення потреб кінцевого користувача:

☐ Збір та обробка телеметрії. Система повинна забезпечувати безперервний прийом даних про фізичні параметри навколишнього середовища (температура, вологість) від периферійних мікроконтролерів ESP32 з періодичністю не рідше одного разу на 10 секунд.

☐ Керування виконавчими пристроями. Платформа має підтримувати миттєву передачу двонаправлених керуючих команд (увімкнення, вимкнення, перемикання стану) на силові реле розумних розеток чи модулів освітлення як через ручний інтерфейс користувача, так і через автоматизовані сценарії.

☐ Система повинна мати вбудований планувальник завдань для виконання відкладених команд автоматизації із прив'язкою до точного часу.

☐ Система має автоматично виявляти нові апаратні модулі ESP32 у мережі, зчитувати їхні MAC-адреси, ініціалізувати профілі у базі даних та динамічно виділяти під них відповідні гілки топіків для обміну повідомленнями.

☐ Платформа повинна у реальному часі відстежувати мережевий статус підключених вузлів та миттєво фіксувати випадки аварійного зникнення

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
						20
Змн.	Арк.	№ докум.	Підпис	Дата		

зв'язку з датчиками (механізм Last Will and Testament), маркуючи їх у системі як offline.

☐ Веб-сервер повинен транслювати будь-які зміни стану пристроїв або нові надходження телеметрії на всі активні користувацькі сесії мобільного додатка без необхідності ручного оновлення екрану (через WebSocket-з'єднання).

☐ Система зобов'язана вести повну хронологічну базу даних історичних показників сенсорів, транзакцій та статусів планувальника в реляційній СКБД для подальшого аналізу.

Нефункціональні вимоги описують якісні характеристики системи, технічні обмеження, архітектурні стандарти, вимоги до продуктивності, безпеки та надійності:

☐ Час затримки між публікацією команди у мобільному додатку та фізичним спрацюванням реле на мікроконтролері ESP32 у межах однієї мережі не повинен перевищувати 200 мілісекунд.

☐ Усі базові контури автоматизації, локального обміну даними та збереження інформації повинні функціонувати всередині домашньої мережі без обов'язкової наявності зовнішнього інтернет-з'єднання.

☐ Зовнішній доступ до локального сервера з глобальної мережі Інтернет має реалізовуватися виключно через шифрований тунель (Cloudflare Tunnel) із захистом від DOS/DDOS-атак та без відкриття портів на роутері (безпечний обхід сірого NAT).

☐ Ефективність використання ресурсів периферії: Вбудоване програмне забезпечення мікроконтролерів має бути оптимізоване за використанням оперативної пам'яті (RAM) та обчислювальних ядер, працюючи під управлінням FreeRTOS з мінімальними накладними витратами на мережевий стек.

Сформований комплекс функціональних та нефункціональних вимог виступає базовим інженерним критерієм для проектування всіх компонентів

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		21

екосистеми «HataPlus». На основі цих детермінованих параметрів та обмежень здійснюється подальша декомпозиція системи на ізольовані архітектурні шари, що дозволяє деталізувати логіку взаємодії програмних модулів Ktor, сервісів маршрутизації MQTT та клієнтських інтерфейсів. Наступне практичне впровадження окреслених вимог у процесі безпосередньої програмної реалізації та розгортання платформи дозволить забезпечити повну відповідність розробленого програмно-апаратного комплексу стандартам надійності, відмововитривалості та масштабованості розподілених інтелектуальних мереж.

2.3 Постановка задачі інженерного проєктування компонентів локально-автономної системи

На основі сформованих функціональних та нефункціональних вимог постає інженерне завдання декомпозиції екосистеми «HataPlus» на три взаємопов'язані автономні рівні: апаратний рівень периферійних пристроїв, серверний рівень обробки бізнес-логіки й агрегації даних, а також клієнтський рівень мобільного інтерфейсу користувача. Головна складність проєктування полягає у забезпеченні безвідмовної, асинхронної взаємодії цих компонентів в умовах відсутності зовнішнього інтернет-з'єднання з мінімальним часом затримки пакетів.

Першим завданням є проєктування та програмування апаратних вузлів на базі мікроконтролерів ESP32. Необхідно розробити архітектуру вбудованого програмного забезпечення під управлінням операційної системи реального часу FreeRTOS. Задачею є опитування цифрових датчиків DHT22 через часові інтервали, таск підтримання Wi-Fi з'єднання і комутації реле, а також таск клієнта Eclipse Paho MQTT. Апаратний модуль повинен стабільно обробляти втрату зв'язку з роутером, ініціювати процедури автоматичного перепідключення (reconnect) і коректно публікувати свій статус у мережу за допомогою передсмертного повідомлення (Last Will and Testament) [37].

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
						22
Змн.	Арк.	№ докум.	Підпис	Дата		

Другим завданням є розробка архітектури серверної частини на базі асинхронного фреймворку Ktor мовою Kotlin [38]. Постановка задачі передбачає створення реактивного сервісного шару, який інтегрується з локальним брокером повідомлень Eclipse Mosquitto. Сервер повинен прослуховувати топіки телеметрії, динамічно реєструвати нові пристрої через конкурентні структури даних (Concurrent HashMap) і виконувати пакетний запис історичних даних у реляційну СКБД PostgreSQL за допомогою інструментів Exposed ORM. Додатковою інженерною задачею на цьому рівні є реалізація WebSocket-сервера для миттєвої трансляції телеметрії підключеним клієнтам та конфігурація планувальника завдань ScheduledExecutorService для виконання команд автоматизації за розкладом.

Третім завданням є побудова кросплатформеного мобільного додатка за допомогою технології Kotlin Multiplatform (КМП) та декларативного інтерфейсу Jetpack Compose. [39] Проектна задача вимагає винесення всієї бізнес-логіки, мережових моделей даних та клієнтського репозиторію у спільний модуль (shared), який може компілюватися як під операційну систему Android, так і під iOS. Необхідно спроектувати архітектурний шаблон MVVM (Model-View-ViewModel) з використанням асинхронних потоків Kotlin Coroutines StateFlow, що дозволить інтерфейсу реагувати на WebSocket-кадри від сервера без затримок та забезпечить плавну рекомпозицію UI-елементів.

У межах загальної постановки задачі окремим науково-практичним викликом виступає узгодження протоколів серіалізації даних між усіма трьома рівнями розподіленої системи. На відміну від стандартних веб-додатків, де ресурси зазвичай вважаються необмеженими, проектувана система повинна використовувати максимально компактні схеми передачі інформації. Це зумовлює необхідність проектування строгих, детермінованих моделей даних на рівні спільної бібліотеки kotlinx.serialization, які дозволять мінімізувати накладні витрати при переході від низькорівневих бітових послідовностей мікроконтролера до об'єктів оперативної пам'яті сервера та клієнтського застосунку [40].

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
						23
Змн.	Арк.	№ докум.	Підпис	Дата		

2.4 Висновок по розділу

У другому розділі виконано комплексний теоретико-інженерний аналіз, що сформував надійне архітектурне та технічне підґрунтя для подальшої практичної розробки розподіленої екосистеми «HataPlus». На основі критичного огляду наявних комерційних платформ автоматизації, таких як Xiaomi Mi Home, Tuuya Smart та Google Home, ідентифіковано їхні ключові архітектурні недоліки та експлуатаційні обмеження. Головними деструктивними факторами комерційних рішень визначено тотальну хмарну залежність, високі мережеві затримки при маршрутизації локального трафіку через географічно віддалені дата-центри, а також закритість програмних інтерфейсів, вендорну обмеженість та низький рівень приватності інформаційного контуру користувачів. Отримані результати аналізу ринку чітко довели, що подальше використання проприетарних закритих систем не здатне задовольнити сучасні вимоги до безпеки, відмововитривалості та швидкодії приватних інтелектуальних середовищ, що повністю виправдовує розробку альтернативного автономного інженерного рішення.

Проведений аналіз дозволив чітко розмежувати, формалізувати та деталізувати вимоги до проектованої IoT-платформи, розділивши їх на взаємопов'язані класи. Основними функціональними завданнями визначено безперервний збір телеметрії з датчиків мікроклімату, миттєве двонаправлене керування силовими реле, автоматизацію за часовими розкладами за допомогою вбудованого планувальника, динамічне виявлення нових апаратних вузлів за їхніми унікальними MAC-адресами та асинхронну реал-тайм трансляцію станів через WebSocket-з'єднання. Серед нефункціональних вимог пріоритет надано жорсткому обмеженню часу відгуку системи у межах локальної мережі (до 200 мс), забезпеченню повної локальної автономності без обов'язкового інтернет-зв'язку, безпеці віддаленого доступу через шифрований тунель Cloudflare Tunnel без відкритої публікації портів роутера, а також оптимізації використання ресурсів оперативної пам'яті та ядер обчислювальної периферії. Таке

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
						24
Змн.	Арк.	№ докум.	Підпис	Дата		

детерміноване поєднання вимог дозволяє спроектувати систему, яка є стійкою до зовнішніх інфраструктурних коливань і гарантує збереження конфіденційності користувача.

Сформульовані вимоги дозволили чітко поставити задачу інженерного проєктування компонентів локально-автономної системи, яку було успішно декомповано на три ізольовані технологічні рівні. На апаратному рівні мікроконтролерів ESP32 задачу підпорядковано концепції легковажної багатозадачності під управлінням операційної системи реального часу FreeRTOS із жорстким розділенням потоків опитування датчиків DHT22, контролю мережевого з'єднання та роботи асинхронного клієнта MQTT. На серверному рівні архітектуру спроектовано на базі неблокуючого високонадійного фреймворку Ktor, інтегрованого із локальним брокером повідомлень Eclipse Mosquitto, СКБД PostgreSQL через Exposed ORM та планувальником ScheduledExecutorService. Клієнтський рівень підпорядковано патерну MVVM у межах кросплатформеної технології Kotlin Multiplatform із декларативним рендерингом графічного інтерфейсу за допомогою Jetpack Compose, що забезпечує нативну швидкодію мобільного додатка на різних операційних системах.

Сформована постановка задачі повністю охоплює всі технічні, комунікаційні та безпекові аспекти функціонування розумного будинку і виступає детермінованим інженерним стандартом для подальшої практичної реалізації системи. Усі розроблені вимоги, обмеження та специфікації інтегровані у єдину логічну структуру, що мінімізує ймовірність виникнення архітектурних помилок на етапах безпосереднього кодування, спрощує процес налагодження та забезпечує високу масштабованість і загальну живучість проєктованого програмно-апаратного комплексу «HataPlus» в умовах тривалої експлуатації.

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
						25
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 3. МЕТОДОЛОГІЯ, ТЕХНОЛОГІЇ ТА ПРОЄКТУВАННЯ СИСТЕМИ

3.1 Об'єкт, предмет та інструментарій аналізу взаємодії системних компонентів

У межах інженерії програмного забезпечення проектування розподілених систем Інтернету речей вимагає чіткого визначення методологічного апарату для забезпечення надійної взаємодії між різномірними апаратними вузлами та програмними комплексами. Об'єктом дослідження у даній роботі є процес розробки та функціонування масштабованої програмної архітектури інтегрованих систем «розумного будинку». Предметом дослідження виступають методи, алгоритми, структури даних та інструментальні засоби створення кросплатформеного програмного забезпечення для забезпечення асинхронного збору телеметрії та відмовостійкого керування локальними інфраструктурними компонентами. Методологічну основу проектування складають принципи об'єктно-орієнтованого та функціонального програмування, а також сучасні архітектурні шаблони розподілених систем, зокрема подієво-орієнтована архітектура (Event-Driven Architecture) та клієнт-серверна модель взаємодії [11].

Взаємодія між апаратним рівнем кінцевих пристроїв на базі мікроконтролерів ESP32 та центральними програмними сервісами системи «Nata Plus » побудована на дворівневому комунікаційному каркасі. Апаратний рівень виконує функцію первинного збору інформації з фізичних сенсорів та її агрегацію у внутрішніх структурах пам'яті мікроконтролера. Інформаційний обмін реалізується в асинхронному режимі за допомогою сеансових підключень до локального MQTT-брокера, який виконує роль проміжного шару та абстрагує логіку маршрутизації повідомлень від безпосередньої бізнес-логіки сервера. Програмний сервер системи, розроблений на базі фреймворку Ktor, виступає одночасно як підписник на телеметричні топіки брокера та як провайдер REST

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
						26
Змн.	Арк.	№ докум.	Підпис	Дата		

API для мобільних клієнтів. Такий розподіл відповідальності дозволяє забезпечити слабку зв'язність компонентів (Loose Coupling), підвищуючи загальну живучість системи та забезпечуючи можливість незалежної модифікації апаратної чи серверної частин без порушення цілісності всього комплексу.

Окрему увагу в методології проектування приділено життєвому циклу обміну даними та забезпеченню цілісності інформаційних пакетів при переході між фізичним та прикладним рівнями. На етапі відправлення з мікроконтролера дані серіалізуються у компактні бітові послідовності, адаптовані до пропускну здатності локальних бездротових мереж. Приймальний модуль сервера здійснює постійний моніторинг черг повідомлень та ініціює процеси валідації кожного пакета на відповідність установленим схемам даних. Проектування цієї взаємодії спирається на стандартизовані протоколи безпеки та моделі авторизації, що гарантує захист від підміни керуючих команд або несанкціонованого перехоплення конфіденційної телеметричної інформації користувача, забезпечуючи високу детермінованість поведінки системи в будь-яких режимах експлуатації.

3.2 Алгоритми, методики та структури даних

Телеметричні дані, що надходять від датчиків апаратної підсистеми «розумного будинку», за своєю інженерною природою є часовими рядами (Time Series Data), де кожен окремий інформаційний запис жорстко прив'язаний до часової мітки та описує поточний стан фізичного параметра об'єкта у конкретний момент часу. Організація ефективної та відмовостійкої обробки таких інформаційних потоків у проєкті «Hata Plus» на базі асинхронного сервера Ktor реалізована через проектування та впровадження стійкого конвеєра обробки повідомлень. Це дозволяє забезпечити послідовне, високопродуктивне та неструктурне перетворення вхідних бітів даних, що надходять від

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		27

розподілених периферійних пристроїв, у структуровані бізнес-сутності системи [12].

Основою первинного етапу функціонування даного конвеєра є використання спеціалізованої модульної бібліотеки `kotlinx.serialization` для швидкої десеріалізації вхідних MQTT-повідомлень. Процес обробки розпочинається з перехоплення асинхронного JSON-пакета, який автоматично мапується у строго типізовані об'єкти мови Kotlin на рівні оперативної пам'яті. Цей архітектурний підхід гарантує абсолютну цілісність даних, оскільки безпосередньо під час парсингу здійснюється автоматична валідація та перевірка наявності обов'язкових полів, таких як унікальний апаратний ідентифікатор пристрою (MAC-адреса), тип підключеного датчика, поточне числове значення телеметрії та відповідна часова мітка. Використання незмінних об'єктів даних (data classes) мови Kotlin забезпечує повну безпеку типів на всіх подальших етапах обробки, повністю мінімізуючи ризик виникнення критичних помилок часу виконання, пов'язаних із нульовими значеннями або невідповідністю форматів даних [13].

Для довготривалого збереження накопиченої історії телеметрії в реляційній базі даних під керуванням Exposed ORM розроблено алгоритм первинної валідації та транзакційного запису. Перед безпосередньою фіксацією інформаційного пакета в таблиці DeviceTelemetry, серверний додаток виконує ідентифікацію та верифікацію пристрою за його унікальним апаратним ключем. Логіка взаємодії зі сховищем даних побудована на використанні ізольованих транзакцій, що повністю задовольняє вимогам атомарності, узгодженості, ізольованості та довговічності операцій (ACID). При отриманні кожного нового повідомлення сервер одночасно виконує дві операції в межах єдиної транзакції: оперативно оновлює поточний статус пристрою в таблиці Devices для відображення стану активності в реальному часі та паралельно додає новий історичний запис до таблиці телеметрії [14].

Для коректного представлення та аналізу накопиченої інформації на стороні клієнтського додатка, створеного за допомогою кросплатформеної

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
						28
Змн.	Арк.	№ докум.	Підпис	Дата		

технології Kotlin Multiplatform, застосовується метод часового структурування та агрегації рядів, який базується на логічному групуванні даних за визначеними часовими інтервалами. Такий підхід дозволяє повністю уникнути паралельного змішування та взаємного перетину різних типів фізичних показників, наприклад, температури та вологості повітря, під час динамічної побудови графічних інтерфейсів користувача. Це гарантує точну та репрезентативну візуалізацію змін параметрів мікроклімату, створюючи надійний інженерний фундамент для подальшого впровадження інтелектуальних алгоритмів предиктивного аналізу роботи обладнання [15].

3.3 Обґрунтування вибору технологічного стеку, інструментальних засобів та середовищ розробки

Ефективність програмної реалізації системи «Nata Plus » безпосередньо залежить від обґрунтованості вибору технологічного стеку, який повинен задовольняти вимогам високої швидкодії, кросплатформеності, безпеки та швидкості розробки. Базовою мовою програмування для всього проекту, включаючи серверну і клієнтську частини, було обрано мову Kotlin завдяки її повній сумісності з екосистемою Java, строгій типізації, лаконічному синтаксису та вбудованій підтримці механізму корутин (Coroutines) [9]. Використання Kotlin Coroutines дозволяє реалізувати асинхронну неблокуючу модель обробки тисяч одночасних мережових запитів та MQTT-повідомлень на сервері, ощадливо використовуючи ресурси операційної системи шляхом обмеження кількості системних потоків, що критично важливо для високонавантажених IoT-шлюзів [10].

Як каркас для розробки серверної частини обрано асинхронний фреймворк Ktor, розроблений компанією JetBrains. На відміну від традиційних монолітних рішень, Ktor є легковажною, модульною платформою, яка дозволяє підключати лише необхідну функціональність у вигляді плагінів (Features), таку

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
						29
Змн.	Арк.	№ докум.	Підпис	Дата		

як автентифікація, серіалізація даних або підтримка WebSockets. Це забезпечує мінімальний час старту додатка та низьке споживання оперативної пам'яті. Для взаємодії з реляційною базою даних PostgreSQL обрано Exposed ORM – спеціалізовану бібліотеку від JetBrains, яка надає два рівні абстракції: типизований SQL-домен (DSL) та класичну модель об'єктно-реляційного відображення (DAO). Використання Exposed дозволяє описувати схеми таблиць бази даних безпосередньо у коді мовою Kotlin, гарантуючи перевірку типів даних та коректності зв'язків на етапі компіляції проекту, що нівелює ризик виникнення помилок під час виконання [10].

Важливим аспектом архітектури системи «Hata Plus» є забезпечення високої доступності даних при роботі в умовах обмеженого інтернет-з'єднання. Завдяки використанню протоколу MQTT з механізмом Quality of Service (QoS) рівня 1, забезпечується гарантована доставка пакетів телеметрії до серверної частини. У разі тимчасової втрати зв'язку між клієнтським пристроєм та сервером, локальний кеш пристрою накопичує дані, які після відновлення з'єднання синхронізуються з базою даних у фоновому режимі. Такий підхід гарантує цілісність часових рядів та виключає виникнення «прогалин» на графіках аналітики, що є критично важливим для коректного функціонування автоматизованих сценаріїв керування кліматичним обладнанням.

Для побудови мобільного інтерфейсу системи застосовано технологію Kotlin Multiplatform (KMP) спільно з фреймворком декларативного інтерфейсу Compose Multiplatform. Вибір KMP дозволяє розробляти єдину кодову базу для бізнес-логіки, мережевої взаємодії та валідації даних, яка без змін компілюється як під платформу Android, так і під iOS, суттєво зменшуючи часові втрати на розробку та підтримку проекту [7]. Інструментальне середовище розробки представлене продуктами компанії JetBrains: IntelliJ IDEA використовується для проектування, відлагодження та тестування серверної архітектури Ktor і бази даних.

Представлена підсумкова таблиця 3.1 повністю охоплює архітектурний каркас проекту та наочно аргументує доцільність використання сучасних засобів

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
						30
Змн.	Арк.	№ докум.	Підпис	Дата		

розробки в межах єдиної екосистеми. Обрані інструменти тісно інтегровані між собою, що дозволяє мінімізувати виникнення помилок сумісності

Таблиця 3.1

Підсумкова характеристика технологічного стеку та інструментальних засобів системи

Категорія компонента	Обране технологічне рішення	Роль у системі
Базова мова програмування	Kotlin	Написання вихідного коду для серверної частини, спільної бізнес-логіки мобільного додатка та конвеєрів обробки даних
Серверний фреймворк	Ktor	Побудова легковажного асинхронного бекенду, обробка REST API запитів та маршрутизація телеметричних потоків
Інструмент взаємодії з БД	Exposed ORM	Мапування об'єктів у реляційні структури та безпечне керування ізольованими транзакціями бази даних
Система керування базами даних	PostgreSQL	Надійне збереження облікових записів користувачів, поточних конфігурацій пристроїв та довготривалої історії телеметрії
Кросплатформена технологія	Kotlin Multiplatform (KMP)	Створення спільної кодової бази для бізнес-логіки, валідації даних та мережевої взаємодії під Android та iOS
Інтерфейс користувача	Compose Multiplatform	Проектування та рендеринг декларативного графічного інтерфейсу мобільного застосунку
Серіалізація даних	kotlinx.serialization	Десеріалізація вхідних JSON-пакетів телеметрії від пристроїв ESP32 та серіалізація REST-відповідей сервера
Середовища розробки	IntelliJ IDEA Ultimate та Android Studio	Проектування, профілювання, відлагодження та компіляція серверного та клієнтського програмного забезпечення

3.4 Висновки по розділу

У третьому розділі було в повному обсязі визначено методологічні засади та інженерні принципи проектування розподіленої системи «Hata Plus», детально досліджено об'єкт та предмет розробки, а також комплексно розглянуто закономірності організації взаємодії між гетерогенними апаратними та програмними компонентами IoT-платформи. Встановлено, що синергетичне поєднання подієво-орієнтованої архітектури (Event-Driven Architecture) та класичної клієнт-серверної моделі забезпечує високоефективний, асинхронний та неблокуючий обмін даними між периферійними мікроконтролерами ESP32, проміжним MQTT-брокером, серверною частиною Ktor та мобільними клієнтами. Запропонований підхід дозволив досягти високого рівня слабкої зв'язності компонентів (Loose Coupling), суттєво спростити процеси горизонтального масштабування системи при підключенні нових споживачів трафіку, а також підвищити загальну відмовостійкість і живучість комплексу в умовах потенційної нестабільності локальних бездротових мереж зв'язку.

У межах проведеного дослідження детально проаналізовано математичні та алгоритмічні засади обробки телеметричних даних як часових рядів, методики їх конвеєрної валідації та довготривалого збереження, а також внутрішні структури даних, що використовуються для репрезентації поточного стану пристроїв в оперативній пам'яті. Визначено та програмно детерміновано механізми швидкої десеріалізації вхідних JSON-пакетів за допомогою модульної бібліотеки `kotlinx.serialization`, принципи транзакційного запису даних у реляційну базу PostgreSQL за допомогою інструментів Exposed ORM, а також методи подальшої структурної агрегації часових рядів для їхньої коректної візуалізації у мобільному застосунку. Використання незмінних строго типізованих моделей даних (data classes) та транзакційного підходу ACID до роботи зі сховищем забезпечує абсолютну цілісність інформації, безпечність обробки конкурентних запитів та повну відповідність вимогам надійності сучасних розподілених інформаційних систем.

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
						32
Змн.	Арк.	№ докум.	Підпис	Дата		

Окрему увагу в межах розділу приділено науковому та практичному обґрунтуванню вибору технологічного стеку та сучасних інструментальних засобів розробки програмного забезпечення. Встановлено, що використання мови програмування Kotlin, легковажного асинхронного фреймворку Ktor, бібліотеки мапування Exposed ORM, промислової системи керування базами даних PostgreSQL, бінарного протоколу MQTT та кросплатформеної технології Kotlin Multiplatform дозволяє створити єдину високопродуктивну екосистему для реалізації серверної та клієнтської частин проєкту. Обрані програмні засоби забезпечують повну кросплатформеність, розширюваність, підтримку легковажної багатозадачності на рівні корутин та спрощують подальший розвиток системи. Розрахунки та характеристики, наведені у підсумковій таблиці, наочно довели доцільність мінімізації різнорідних технологій на користь єдиного технологічного середовища JetBrains, що дозволяє усунути ризики несумісності форматів даних на стиках взаємодії між вбудованим ПЗ мікроконтролерів, серверною логікою та мобільними додатками.

На основі розробленого методологічного та технологічного апарату було повністю завершено етап концептуального проєктування, що дозволило перейти до безпосереднього формування інженерних рішень, які описують фізичну реалізацію бази даних та програмних інтерфейсів. Усі обґрунтовані алгоритми обробки інформаційних потоків, транзакційні моделі взаємодії та кросплатформені структури даних сформувавши цілісний і несуперечливий каркас розподіленого застосунку. Отримані результати підтверджують технічну зрілість обраних підходів, мінімізують ймовірність архітектурних помилок на наступних етапах безпосереднього написання вихідного коду і виступають детермінованим інженерним стандартом для успішної практичної реалізації платформи «Hata Plus» та її подальшого впровадження в реальних умовах експлуатації.

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
						33
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 4. РЕАЛІЗАЦІЯ ПРОЄКТУ HATA PLUS

4.1 Опис розробки програмного коду

Реалізація серверної частини системи розумного будинку «HataPlus» базується на асинхронній подійно-орієнтованій архітектурі, де ключову роль у забезпеченні взаємодії між бекенд-платформою Ktor та периферійними пристроями ESP32 відіграє спеціалізований сервісний шар, інкапсульований у класі MqttService [21]. Центральним сполучним вузлом для обміну повідомленнями виступає відкритий MQTT-брокер Eclipse Mosquitto, що забезпечує легковажну комутацію пакетів за принципом видавець-підписник [18]. Інтеграція з чергами повідомлень у коді сервера реалізована за допомогою індустріального стандарту клієнтської бібліотеки Eclipse Paho MQTT, яка забезпечує надійне низькорівневе керування мережевими сокетом, автоматичне відновлення сесій та гарантовану доставку стеків пакетів телеметрії. Вибір такої архітектури зумовлений потребою мінімізації затримок при передачі даних та здатністю системи ефективно функціонувати в умовах нестабільного бездротового зв'язку, що є характерною рисою сучасних розподілених IoT-мереж [20], [19].

Перед початком програмної взаємодії критично важливо забезпечити коректне інфраструктурне налаштування самого брокера Mosquitto, що реалізується за допомогою конфігураційного файлу mosquitto.conf. Цей файл визначає мережеві інтерфейси, правила збереження стану сесій, ліміти черг та рівні журналювання подій для оптимізації пропускну здатності під високу частоту відправки дрібних пакетів від датчиків. На рисунку 4.1 продемонстровано знімок екрана з фінальною робочою конфігурацією брокера Mosquitto, яка розгорнута в середовищі розробки для обслуговування платформи «HataPlus».

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
						34
Змн.	Арк.	№ докум.	Підпис	Дата		

```

user mosquitto
persistence true
persistence_location /var/lib/mosquitto/
log_dest file /var/log/mosquitto/mosquitto.log

# Стандартний MQTT
listener 1883 0.0.0.0
allow_anonymous true

# WebSockets для Localtunnel
listener 9001 0.0.0.0
protocol websockets
allow_anonymous true
socket_domain ipv4

```

Рисунок 4.1 — Знімок екрана з файлом конфігурації mosquitto.conf
брокера Mosquitto

Процес ініціалізації з'єднання з налаштованим брокером Mosquitto виконується сервером через метод connect, який конфігурує параметри підключення за допомогою об'єкта MqttConnectOptions. Сервер реєструється в брокері під унікальним ідентифікатором HataPlusBackendServer та встановлює прапорець очищення сесії isCleanSession у положення true, що запобігає накопиченню застарілих повідомлень на стороні брокера під час перезапуску додатка. Додатково конфігуруються таймаут підключення тривалістю 15 секунд, інтервал перевірки активності вузла keepAliveInterval на рівні 60 секунд та активується функція автоматичного перепідключення isAutomaticReconnect. Частина описаного методу представлена на рисунку 4.2.

```

28 fun connect() {
29     try {
30         val clientId = "HataPlusBackendServer"
31         println(" * Спроба підключення до MQTT брокера: $brokerUrl з ID: $clientId")
32
33         client = MqttClient(brokerUrl, clientId, MemoryPersistence())
34
35         val options = MqttConnectOptions().apply {
36             isCleanSession = true
37             connectionTimeout = 15
38             keepAliveInterval = 60
39             isAutomaticReconnect = true
40         }
41
42         > client?.setCallback(object : MqttCallbackExtended {...})
43
44         client?.connect(options)
45
46         > } catch (e: Exception) {...}
47     }
48 }

```

Рисунок 4.2 – Фрагмент методу connect

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
						35
Змн.	Арк.	№ докум.	Підпис	Дата		

Після успішного встановлення зв'язку, завдяки використанню інтерфейсу зворотного виклику `MqttCallbackExtended` (рис. 3.4), сервер автоматично оформляє підписку на три базові гілки топіків. Топік `home/sensors/data` призначений для збору періодичних телеметричних даних, топік `home/sensors/discovery` використовується для динамічного виявлення нових мікроконтролерів у локальній мережі, а топік `home/sensors/status/+` слугує для моніторингу поточного стану активності пристроїв за допомогою механізму Last Will and Testament.

Специфіка обробки повідомлень, що надходять у метод `messageArrived`, що наведений на рисунку 4.3, полягає в їхній сегрегації на основі текстового аналізу назви топіку. При отриманні пакета з каналу виявлення, рядок корисного навантаження десеріалізується бібліотекою `Kotlinox Serialization` у об'єкт `DiscoveryData`, після чого унікальний MAC-адрес пристрою приводиться до верхнього регістру і фіксується у конкурентній хеш-карті `discoveredDevices` разом із поточною часовою міткою сервера.

```

42 client?.setCallback(object : MqttCallbackExtended {
43     override fun connectComplete(reconnect: Boolean, serverURI: String?) {
44         println("🟢 MQTT з'єднання встановлено успішно! Reconnect: $reconnect")
45         client?.subscribe(topicFilter: "home/sensors/data", qos: 1)
46         client?.subscribe(topicFilter: "home/sensors/discovery", qos: 1)
47         client?.subscribe(topicFilter: "home/sensors/status/+", qos: 1)
48         println("🔔 Підписка на топіки оформлена")
49     }
50
51     override fun connectionLost(cause: Throwable?) {
52         println("⚠️ З'єднання з MQTT брокером втрачено: ${cause?.message}")
53     }
54
55     override fun messageArrived(topic: String?, message: MqttMessage?) {
56         val payload = message?.payload?.let { String(it) }?: return
57         val currentTopic = topic?: return
58
59         if (currentTopic == "home/sensors/discovery") {
60             handleDiscoveryMessage(payload)
61             return
62         }
63
64         if (currentTopic.startsWith(prefix: "home/sensors/status/")) {
65             val macAddress = currentTopic.substringAfter(delimiter: "home/sensors/status/").uppercase()
66             handleStatusMessage(macAddress, payload)
67             return
68         }
69
70         if (currentTopic == "home/sensors/data") {
71             handleTelemetryMessage(payload)
72         }
73     }
74
75     override fun deliveryComplete(token: IMqttDeliveryToken?) {}
76 })

```

Рисунок 4.3 - Реалізація інтерфейсу `MqttCallbackExtended`

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		36

Обробка вхідної телеметрії з топіку `home/sensors/data` демонструє гнучку інтеграцію шару передачі повідомлень із шаром збереження даних. Після успішного розбору JSON-структури об'єкта `EspDeviceData` та валідації значень сенсорів, бекенд-платформа відкриває ізольовану базу транзакцій СКБД PostgreSQL за допомогою інструментів Exposed ORM. У межах єдиної транзакції виконується пошук пристрою за його MAC-адресою; у разі успішного знаходження запису, статус пристрою оновлюється на `online`, фіксується точний час останнього зв'язку `lastSeen`, а поточні показники температури та вологості записуються у кешувальні поля сутності `Devices`. Одночасно з цим історичні значення фізичних параметрів записуються як окремі атомарні рядки у таблицю `DeviceTelemetry` із маркерами `TEMPERATURE` та `HUMIDITY`, що гарантує збереження повної хронології функціонування об'єкта автоматизації.

Одразу після успішної фіксації телеметричного чи статусного стану в базі даних, сервіс ініціює миттєве сповіщення підключених користувачів через метод `broadcastToWebSockets`. Сервер трансформує об'єкт даних назад у текстовий рядок формату JSON і за допомогою пулу потоків розсилає його всім активним клієнтським сесіям, що зареєстровані в глобальному об'єкті `ConnectionManager`. `CoroutineScope(Dispatchers.IO)` дозволяє виконувати розсилку кадрів `WebSocket` фреймів асинхронно, не блокуючи основний потік обробки MQTT-повідомлень, а вбудований механізм фільтрації автоматично виявляє та видаляє «мертві» або обірвані сесії, підтримуючи високу стабільність мережевого інтерфейсу додатка.

Зворотний зв'язок із периферійними пристроями для надсилання керуючих сигналів реалізовано через метод `publishCommand` (додаток Б), який формує JSON-пакет стану і публікує його в цільовий топік з урахуванням унікальної MAC-адреси пристрою у нижньому регістрі. Для розв'язання завдань часової автоматизації, наприклад, увімкнення розумної розетки за розкладом, розроблено метод `scheduleCommand` (додаток Б). Інформація про створення таймера дублюється в реляційній таблиці `DeviceSchedules`, а після успішного спрацювання відкладеного потоку та відправки MQTT-команди, статус розкладу

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
						37
Змн.	Арк.	№ докум.	Підпис	Дата		

автоматично деактивується в базі даних, що виключає повторне несанкціоноване спрацювання периферійного реле.

Ефективне функціонування розподіленої системи розумного будинку «HataPlus» та її інтеграція з мобільними і веб-додатками користувачів безпосередньо залежить від архітектури міжкомпонентної взаємодії. Для забезпечення надійної, незалежної від платформи та масштабованої комунікації між клієнтською частиною та сервером Ktor було розроблено прикладний програмний інтерфейс на базі архітектурного стилю. Вибір цієї технології зумовлений її здатністю забезпечувати stateless-взаємодію, за якої кожен HTTP-запит містить усю необхідну інформацію для його обробки, що суттєво знижує навантаження на оперативну пам'ять сервера та спрощує горизонтальне масштабування системи в умовах зростання кількості одночасних підключень.

Програмна логіка маршрутизації, зосереджена у модулі Routing.kt (додаток Б), розділяє прикладний інтерфейс на два функціональні блоки, що представлені публічними та захищеними кінцевими точками (endpoints). Публічний сегмент містить єдиний критично важливий маршрут аутентифікації /api/auth/login, який приймає HTTP-запити методом POST. Корисне навантаження записується розробником у вигляді JSON-структури, яка десеріалізується в об'єкт даних AuthRequest, що містить електронну пошту та пароль користувача. Логіка обробки цього запиту виконує транзакційну перевірку в реляційній базі даних PostgreSQL, зіставляючи введені дані із записами у таблиці Users, та у разі успіху ініціює випуск безпечного маркера доступу для клієнта.

Захист решти адміністративних та інфраструктурних маршрутів системи розумного будинку реалізовано за допомогою криптографічних токенів JWT (JSON Web Token), інтегрованих через вбудований механізм безпеки Ktor auth-jwt у файлі Security.kt, код якого наведений в додатку Б. Конфігурація JwtConfig (рис. 4.4) використовує стійкий алгоритм підпису HMAC256 із секретним ключем та встановлює тривалий термін дії токена для зручності безперебійної роботи мобільних IoT-додатків. Кожен випущений маркер містить зашифровані

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
						38
Змн.	Арк.	№ докум.	Підпис	Дата		

службові твердження (claims), серед яких унікальний ідентифікатор користувача `userId`, його пошта `email` та рівень прав `role`. Маршрути керування кімнатами, конфігурації розумних пристроїв та видалення облікових записів обгорнуті в блок автентифікації, який автоматично відхиляє будь-які анонімні запити із поверненням HTTP-статусу 401 Unauthorized.

```

10 object JwtConfig {
11     const val secret = "super-secret-hata-plus-key-2026"
12     const val issuer = "https://hata-plus.local/"
13     const val audience = "hata-plus-app"
14
15     fun generateToken(userId: Int, email: String, role: String): String {
16         return JWT.create()
17             .withAudience(audience)
18             .withIssuer(issuer)
19             .withClaim(name: "userId", userId)
20             .withClaim(name: "email", email)
21             .withClaim(name: "role", role)
22             .withExpiresAt(Date(date: System.currentTimeMillis() + 365L * 24 * 60 * 60 * 1000))
23             .sign(Algorithm.HMAC256(secret))
24     }
25 }

```

Рисунок 4.4 – Реалізація конфігурація JwtConfig

Окрему інженерну цінність у розробці специфікації REST API має впровадження суворого розмежування прав доступу на основі рольової моделі. Яскравим прикладом реалізації такої бізнес-логіки в `Routing.kt` є захищений маршрут видалення користувачів методом `DELETE` за шляхом `/api/users/{userId}`. При надходженні запиту сервер спочатку вилучає та валідує JWT-токен, перевіряючи значення claims, і якщо роль поточного користувача не відповідає найвищому рівню привілеїв `HOST`, система миттєво припиняє обробку та повертає HTTP-код 403 Forbidden. Лише у разі підтвердження статусу адміністратора, `Exposed ORM` запускає каскадне очищення зв'язаних сутностей у таблиці `DeviceAccess` та безпосередньо видаляє запис із таблиці `Users`, коректно оперуючи статусами 400 BadRequest при невірному форматі ID та 404 NotFound при відсутності цільового об'єкта.

Структура дерева кінцевих точок REST API показано на рисунку 4.5. Повний лістинг коду `Routing.kt` наведено в додатку Б.

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
						39
Змн.	Арк.	№ докум.	Підпис	Дата		

```

29      routing {
30          // Публічний login
31          > post(path: "/api/auth/login") {...}
32
33          // Захищені маршрути
34          authenticate(...configurations: "auth-jwt") {
35              > get(path: "/") {...}
36
37              > post(path: "/api/auth/register-guest") {...}
38
39              > get(path: "/api/devices/discovered") {...}
40
41              > post(path: "/api/devices") {...}
42
43              > delete(path: "/api/devices/{deviceId}") {...}
44
45              > get(path: "/api/telemetry/{deviceName}") {...}
46
47              route(path: "/api/rooms") {
48                  > get {...}
49
50                  > post {...}
51
52                  > get(path: "/{roomId}/devices") {...}
53
54                  > delete(path: "/{roomId}") {...}
55              }
56
57              > get(path: "/api/devices/{macAddress}/history") {...}
58
59              > post(path: "/api/devices/command") {...}
60
61              > get(path: "/api/devices/{deviceId}/schedules") {...}
62
63              > delete(path: "/api/devices/schedules/{scheduleId}") {...}
64
65              > get(path: "/api/users") {...}
66
67              > delete(path: "/api/users/{userId}") {...}
68          }
69      }
70  }
71

```

Рисунок 4.5 - Структурна схема дерева кінцевих точок REST API

Кросплатформений мобільний клієнт системи розумного будинку «HataPlus» спроектовано та реалізовано на базі сучасного декларативного фреймворку Compose Multiplatform (Jetpack Compose), що дозволяє використовувати єдину кодову базу для побудови користувацького інтерфейсу та бізнес-правил під операційні системи Android та iOS. Реактивне оновлення екранів досягається завдяки використанню інструментів Kotlin Coroutines та потоків даних StateFlow, що гарантує миттєве відображення змін у телеметрії, які надходять від MQTT-брокера через бекенд-сервер Ktor.

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		40

Процес взаємодії користувача з мобільним додатком починається з екрана автентифікації, реалізованого у компоненті LoginScreen (додаток В). Графічний інтерфейс, що наведений на рисунку 4.6, використовує текстові поля для введення електронної пошти та пароля з вбудованою трансформацією PasswordVisualTransformation для приховування символів. Обробка події натискання кнопки входу виконується асинхронно в межах життєвого циклу екрана за допомогою функції rememberCoroutineScope, яка запускає корутину для виконання мережевого запиту onLoginClick.

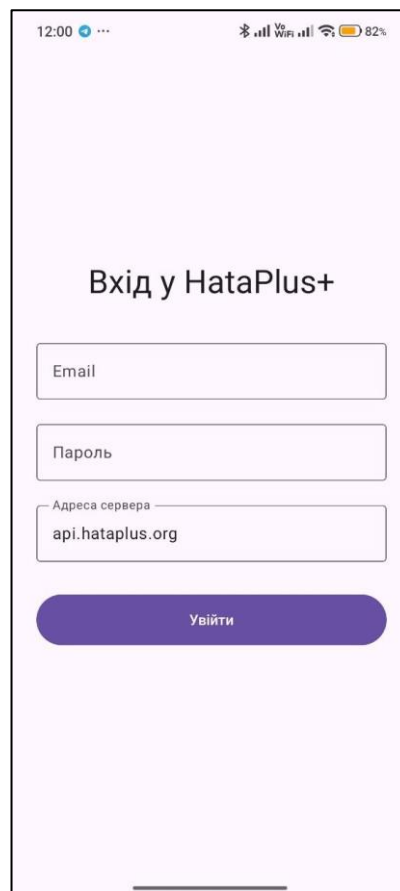


Рисунок 4.6 – Екран авторизації користувача

Після авторизації користувач отримує доступ до екрана профілю AccountScreen, інтерфейс якого наведений на рисунку 4.7, динамічно адаптується під рольову модель системи, приховуючи або відображаючи керуючі кнопки «Зареєструвати гостя» та «Користувачі будинку» на основі прапорця canRegisterGuests та строкового значення користувацької ролі userRole.

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
						41
Змн.	Арк.	№ докум.	Підпис	Дата		

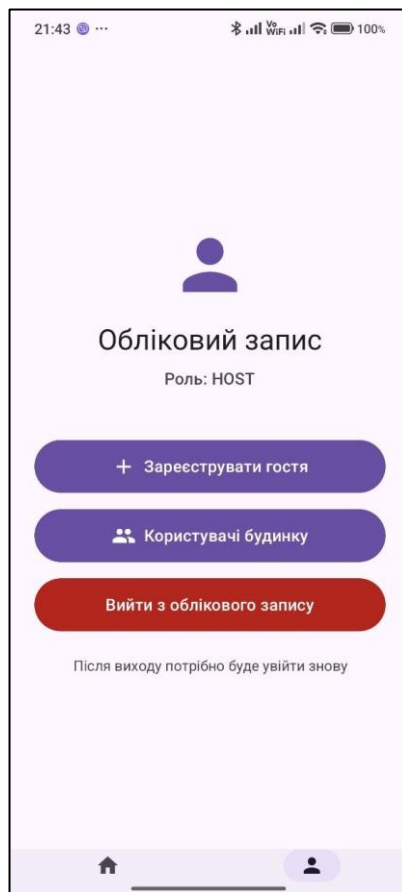


Рисунок 4.7 – Екран облікового запису для користувача з роллю HOST

Побудова просторової структури будинку в додатку реалізована через ієрархічну систему екранів HomeScreen та RoomDetailScreen, що тісно взаємодіють із модальними вікнами конфігурації. На головному екрані HomeScreen, що на зображено рисунку 4.8, списки доступних приміщень відображаються у вигляді гнучкої сітки LazyVerticalGrid, де кожна картка кімнати є клікабельним елементом, який транслює поточний стан об'єкта. Додавання нових кімнат інтегровано через універсальний діалоговий компонент AddItemDialog, який використовує AlertDialog із текстовим полем для введення назви приміщення та автоматичною валідацією на незаповнені символи. Для безпечного адміністрування простору в HomeScreen реалізовано каскадне видалення, яке при натисканні на відповідну іконку викликає попереджувальне модальне вікно, що інформує про повне видалення кімнати разом із прив'язаними пристроями.

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
						42
Змн.	Арк.	№ докум.	Підпис	Дата		

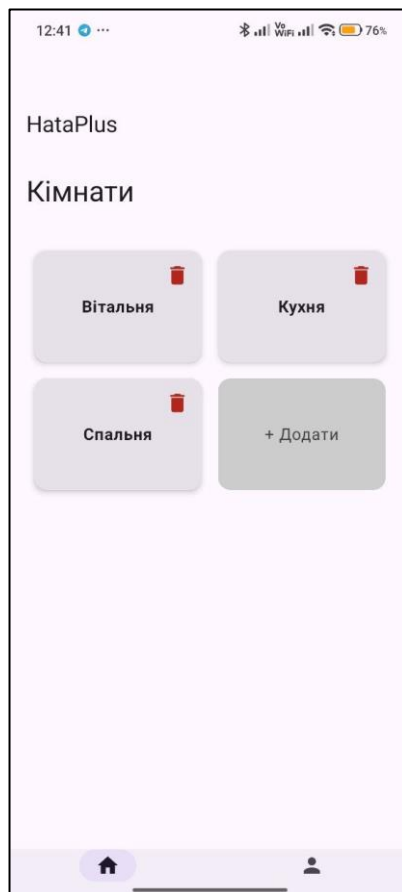


Рисунок 4.8 – Головний екран

При переході в RoomDetailScreen додаток задіяє LazyColumn для динамічного рендерингу підключеного периферійного обладнання, де для датчиків клімату автоматично виводяться графічні індикатори температури та вологості, а для виконавчих реле — поточний статус мережевої активності (рис. 4.9).

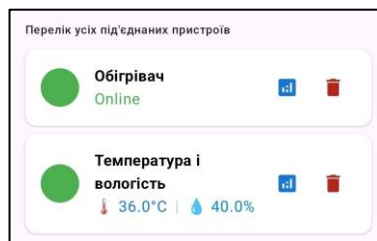


Рисунок 4.9 – Відображення пристроїв на екрані кімнати

Процедура підключення нових фізичних модулів ESP32 до екрана кімнати покладається на інтерфейс AddDeviceScreen (додаток В), який

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
						43
Змн.	Арк.	№ докум.	Підпис	Дата		

забезпечує автоматичне виявлення обладнання в локальній мережі. Екран оперує асинхронним методом оновлення списку виявлених пристроїв `onRefreshDiscovered`, що дозволяє користувачеві в реальному часі бачити нові плати, які надіслали широкомовні пакети ідентифікації. Користувач може вибрати знайдений пристрій зі списку, що автоматично заповнює унікальну MAC-адресу у відповідне поле введення `NataTextField`, після чого задається користувацька назва та виконується прив'язка до бази даних через подію `onDeviceConfigured`. Приклад додавання нового пристрою до кімнати наведено на рисунку 4.10.

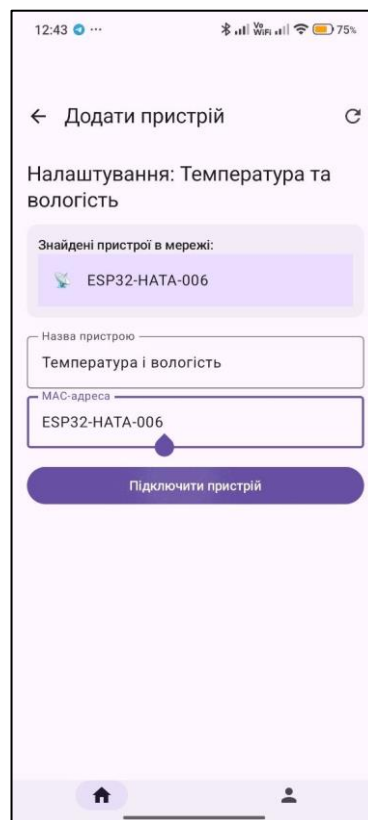


Рисунок 4.10 – Процес додавання датчику температури

Аналогічно побудовано екран безпеки `RegisterGuestScreen` (додаток В), який дозволяє користувачеві з правами адміністратора (HOST) реєструвати нові облікові записи гостей, створюючи складний об'єкт запиту `RegisterGuestRequest`. Цей екран містить динамічний список усіх пристроїв будинку, де для кожного девайса за допомогою випадаючого списку чи перемикачів формується карта

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
						44
Змн.	Арк.	№ докум.	Підпис	Дата		

дозволів із рівнями доступу, що зводить до мінімуму людський фактор при налаштуванні безпеки розумного будинку. Приклад реєстрації користувача (гостя) показано на рисунку 4.11.

Рисунок 4.11 – Екран додавання користувача з роллю Guest

Слідуючим елементом мобільного клієнта є аналітичний блок, наведений на рисунку 4.12, представлений парою з екрана DeviceAnalyticsScreen та DeviceAnalyticsViewModel (додаток В). Логіка управління даними повністю інкапсульована у в'юмоделі, яка оперує єдиним реактивним станом AnalyticsUiState, що об'єднує списки історичних точок, поточні показники сенсорів та стан завантаження. Для оптимізації відображення великих масивів історичних даних у в'юмоделі реалізовано алгоритм групування за годинами aggregateByHour, який усереднює показники телеметрії для обраного періоду. Графічне відображення трендів у DeviceAnalyticsScreen реалізовано за допомогою низькорівневого малювання на компоненті Canvas, де за допомогою

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		45

методів `drawRect` та `drawText` обчислюються координати стовпців, межі діапазону значень та текстові мітки часу на осі абсцис.

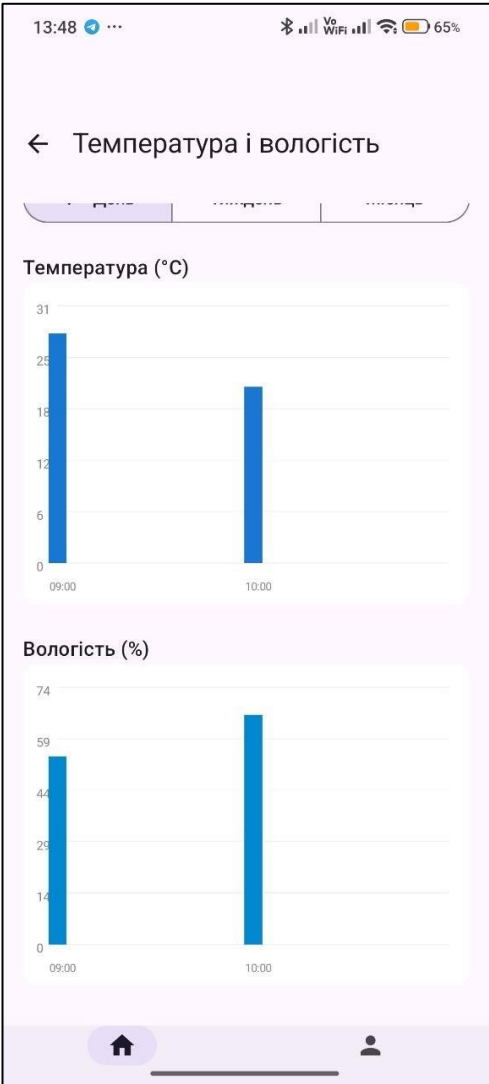


Рисунок 4.12 – Екран аналітики датчика температури та вологості

Крім візуалізації, цей екран виступає органом безпосереднього управління, транслуючи натискання кнопок у виклики методів `toggleSocket` та `scheduleTimer`, які виконують відправку керуючих команд `TOGGLE` та `SCHEDULE` з відповідними корисними навантаженнями у фоновому контексті `Dispatchers.IO`. На рисунку 4.13 показаний екран управління розеткою де користувачу надається можливість вмикати та вимикати розетку, а також запланувати коли пристрій має увімкнутись, а коли вимкнутись. Також реалізована можливість видалення запланованих раніше дій.

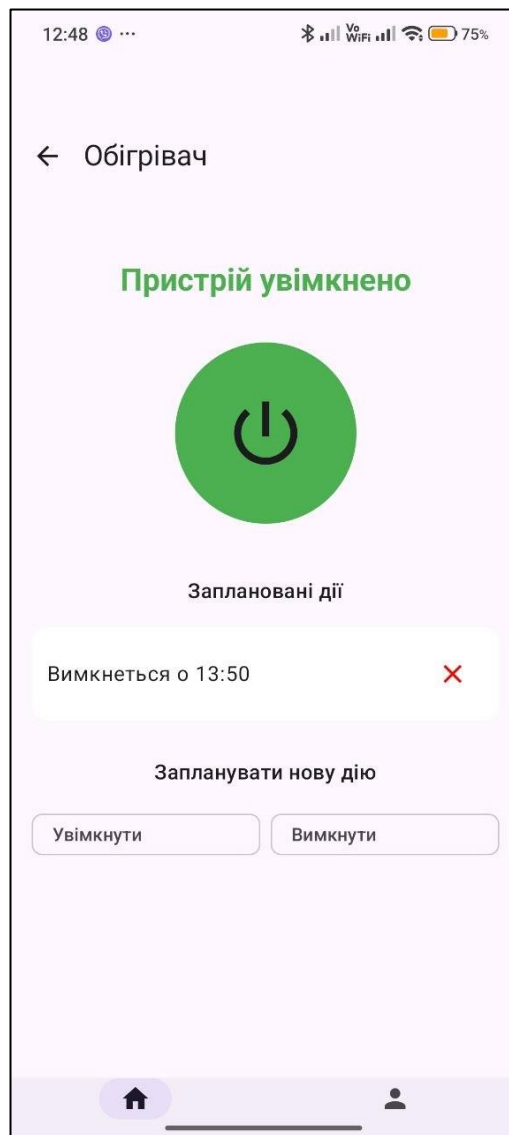


Рисунок 4.13 – Екран управління розеткою

Впровадження описаного комплексу декларативних екранів та реактивних архітектурних рішень забезпечило високу швидкість відгуку мобільного додатка «HataPlus» на події реального часу та дозволило створити уніфікований користувацький досвід. Використання компонентів Jetpack Compose суттєво скоротило обсяг шаблонного коду для управління станом інтерфейсу, а повна ізоляція бізнес-логіки всередині ViewModel гарантує надійність та простоту подальшого супроводу кросплатформеної системи.

Розробка вбудованого програмного забезпечення для апаратних компонентів системи «HataPlus» орієнтована на забезпечення автономності, енергоефективності та гарантованої доставки телеметричних даних до

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
						47
Змн.	Арк.	№ докум.	Підпис	Дата		

центрального сервера. На етапі проєктування та попереднього тестування архітектури рішень було задіяно сучасну платформу хмарного моделювання Wokwi. Це середовище дозволило розгорнути повнофункціональні віртуальні прототипи пристроїв, налаштувати емуляцію взаємодії мікроконтролера ESP32 з периферією по шинах I2C та OneWire, а також перевірити роботу мережевого стеку в умовах віртуальної локальної мережі [22].

У межах архітектури розумного будинку було реалізовано програмне забезпечення для двох типів периферійних вузлів на базі мікроконтролера ESP32, які виконують відмінні функціональні завдання. Перший тип представлений керованою розумною розеткою (рис. 4.14), завданням якої є комутація навантаження через блок реле та візуалізація поточного режиму роботи за допомогою світлодіодного індикатора й OLED-екрана на базі драйвера SSD1306.

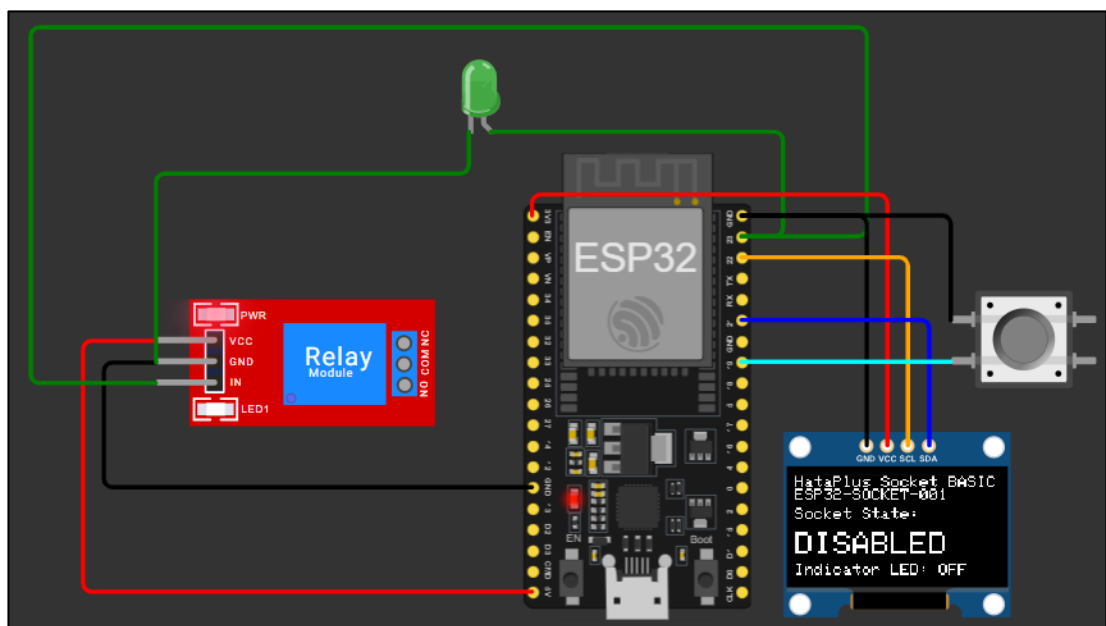


Рисунок 4.15 - Принципова апаратна схема модуля розумної розетки на базі мікроконтролера ESP32

Другий тип — це вимірювальний кліматичний сенсор (рис. 4.16), який здійснює циклічне опитування цифрового датчика DHT22 для фіксації

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		48

параметрів температури й вологості в приміщенні з подальшим виведенням інтегрованих показників на локальний дисплей.

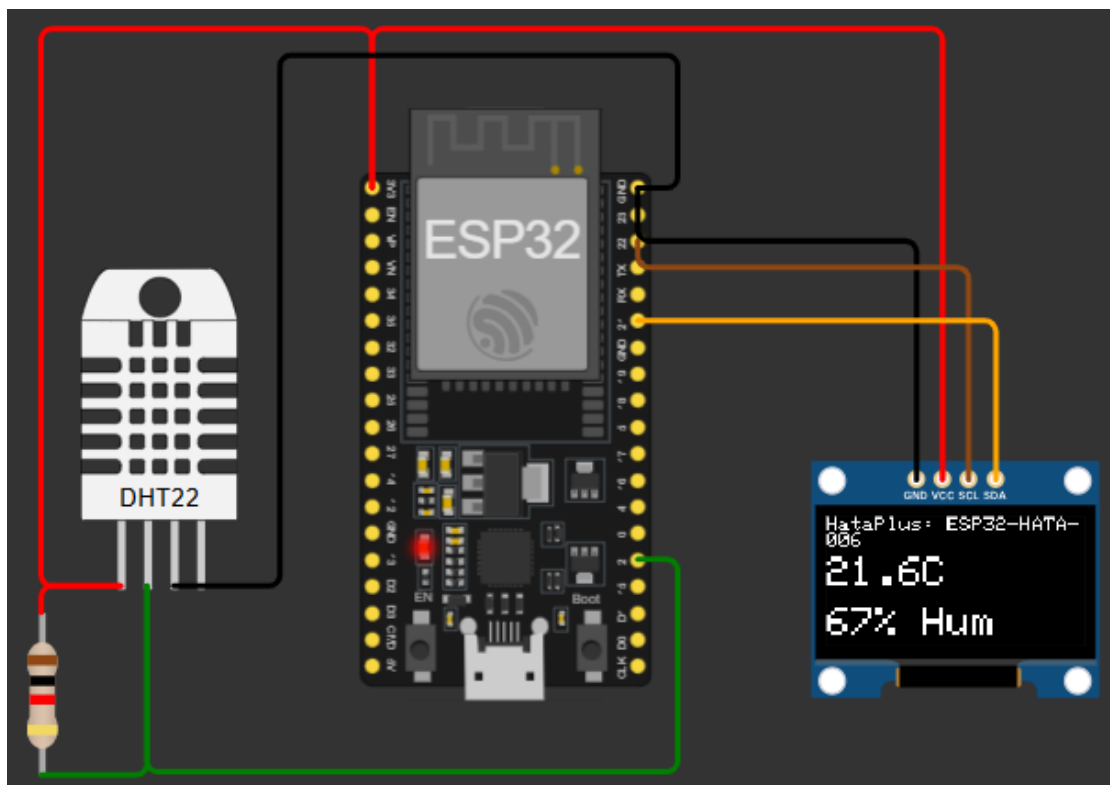


Рисунок 4.16 - Принципова апаратна схема кліматичного сенсорного модуля на базі датчика DHT22

Прошивка апаратних модулів розроблена з урахуванням багатозадачної специфіки операційної системи реального часу FreeRTOS, інтегрованої в базове обчислювальне ядро ESP32 [23]. Лістинг прошивок наведено в додатку Д. Програмна архітектура ефективно розподіляє навантаження між двома апаратними ядрами процесора Xtensa, забезпечуючи стабільність критично важливих обчислювальних потоків [23]. Системні завдання, пов'язані з обслуговуванням мережних протоколів Wi-Fi, утриманням сокет-з'єднань та обробкою трафіку TCP/IP, автоматично делегуються планувальником FreeRTOS на ядро Core 0. Усі прикладні операції, включаючи локальний рендеринг графічного інтерфейсу OLED, періодичне зчитування сенсорів та обробку користувацьких таймерів, виконуються в контексті основного завдання на ядрі Core 1 [23]. Такий розподіл гарантує, що короточасні затримки при роботі з

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		49

мережею або тривалі процедури очікування відгуку датчиків не блокують виконання критично важливих операцій, наприклад, миттєвого відключення реле за командою безпеки.

Стійкість функціонування розподіленої системи в умовах нестабільного бездротового покриття забезпечується вбудованою логікою автоматичного відновлення сесій зв'язку за протоколом MQTT [24]. Програма безперервно моніторить стан мережевого інтерфейсу та у разі втрати сигналу Wi-Fi або розриву з'єднання з брокером Mosquitto ініціює процедуру неблокуючого перепідключення. Важливою інженерною особливістю є інтеграція механізму Last Will and Testament (LWT) на рівні транспортного протоколу [6]. Під час авторизації на брокері кожен пристрій реєструє аварійне повідомлення «offline» у своєму статусному топіку. Якщо апаратний модуль раптово втрачає живлення, брокер самостійно розсилає це повідомлення іншим передплатникам системи, сигналізуючи про втрату зв'язку. При штатному старті чи успішному перепідключенні пристрій відразу публікує статус «online» та надсилає спеціалізований пакет автоматичного виявлення (discovery), що дозволяє серверній частині Ktor гнучко керувати реєстром активного обладнання без жорсткого кодування адрес.

Комунікаційний шар між мікроконтролерами та центральним сервером стандартизовано за допомогою структурованих пакетів у форматі JSON [25]. Для серіалізації поточних метрик датчиків та десеріалізації вхідних директив управління використовується оптимізована за пам'яттю бібліотека ArduinoJson, що оперує статичними буферами в оперативній пам'яті мікроконтролера. Обмін даними побудовано за асиметричним принципом: передача телеметрії (стан розетки, рівень температури й вологості) здійснюється строго за таймером із фіксованим інтервалом, що запобігає перевантаженню мережі, тоді як приймання команд відбувається подієво через механізм зворотних викликів (callbacks) [25]. Отримавши команду, пристрій розбирає JSON-структуру, перевіряє цілісність директиви та змінює логічний стан відповідного цифрового піна.

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
						50
Змн.	Арк.	№ докум.	Підпис	Дата		

4.2 Використані алгоритми та структури даних

Проектування та програмна реалізація архітектури збереження даних у системи розумного будинку «HataPlus» базується на інтеграції об'єктно-реляційного відображення Exposed ORM та реляційної системи керування базами даних PostgreSQL. Вибір СКБД PostgreSQL як центрального реляційного сховища обґрунтований її високою відмовостійкістю, повною відповідністю критеріям ACID, розширеною підтримкою конкурентних з'єднань та здатністю стабільно функціонувати в умовах безперервного транзакційного навантаження, притаманного комплексним IoT-платформам. Інтеграція цієї системи з легковажним архітектурним фреймворком Exposed від компанії JetBrains дозволяє побудувати оптимізований і безпечний шар доступу до даних, нівелюючи класичний розрив між реляційною моделлю сховища та об'єктно-орієнтованою парадигмою бекенд-платформи.

Завдяки використанню вбудованих архітектурних інструментів Exposed, уся структура бази даних повністю декларується у вигляді рідних синтаксичних конструкцій мови програмування у формі об'єктів-таблиць. Такий підхід надає розробнику можливість гнучкого та централізованого керування унікальними індексами, складними обмеженнями зовнішніх ключів та механізмами автоматизованого формування оптимізованих SQL-запитів без необхідності ручного написання та підтримки сирого реляційного коду. Опис фізичної схеми даних у вигляді програмних об'єктів суттєво спрощує технічний аудит архітектури, дозволяє ефективно відстежувати та рецензувати еволюцію сховища засобами стандартних систем контролю версій Git і повністю відповідає сучасним інженерним концепціям швидкої розробки, автоматизації та довготривалого супроводу складного програмного забезпечення [16]. Для наочного відображення логіко-фізичної структури сховища, взаємозв'язків між сутностями, обмежень цілісності та типів даних безпосередньо перед описом програмних класів була розроблена ER-діаграма системи. Схему, що демонструє архітектуру розподілу реляційних таблиць, первинні та зовнішні ключі, а також

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
						51
Змн.	Арк.	№ докум.	Підпис	Дата		

унікальні індекси, що виступають основою для генерації таблиць у середовищі PostgreSQL, наведено на рисунку 4.17.

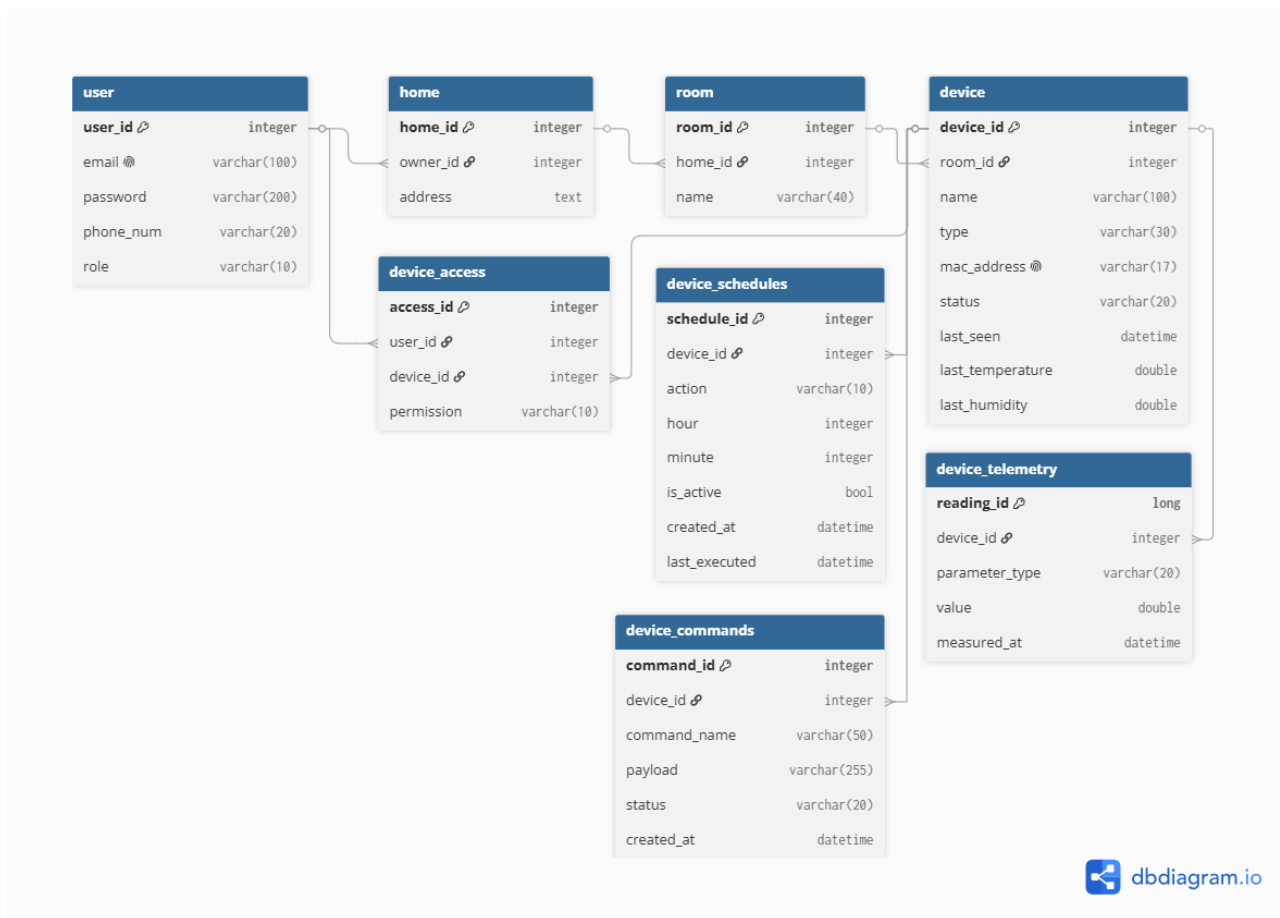


Рисунок 4.17 — Схема реляційної бази даних системи «HataPlus»

Базова реляційна структура системи відображає строгу просторову та ієрархічну топологію об'єктів автоматизації, описану в класичних працях з теорії моделювання баз даних. Сутність Users виступає первинною точкою входу, до якої через зовнішній ключ ownerId прив'язується таблиця Homes, яка, у свою чергу, є батьківською для таблиці кімнат Rooms. Кінцеві периферійні пристрої, зафіксовані в таблиці Devices, жорстко пов'язані з відповідними кімнатами через поле roomId. Усі зв'язки між сутностями підтримують цілісність посилок на рівні СКБД завдяки декларативним методам references, що дозволяє налаштовувати каскадне видалення або оновлення записів, запобігаючи появі неприв'язаних рядків у сховищі даних при видаленні користувача, будинку чи

кімнати. Повна програмна реалізація опису схеми сховища за допомогою Exposed DSL наведена у додатку А.

Окремим важливим завданням у межах платформи IoT є ефективна обробка та збереження інтенсивних потоків телеметричних часових рядів, які генеруються датчиками температури та вологості мікроконтролерів ESP32. Для розв'язання цієї задачі було розроблено спеціалізовану таблицю DeviceTelemetry, яка функціонує за принципом вузької моделі збереження, що є стандартом для реляційних баз даних, які адаптуються під збереження часових рядів. Кожен окремий вимір записується як атомарна подія, що містить унікальний ідентифікатор readingId типу Long з автоінкрементом для запобігання переповненню лічильника записів при тривалій експлуатації системи, посилання на пристрій deviceId, текстовий маркер типу фізичного параметра parameterType, числове значення value типу Double та часову мітку measuredAt. Така архітектура часових рядів дозволяє системі лінійно масштабуватися та накопичувати великі обсяги записів без деградації швидкодії, оскільки додавання нових типів сенсорів не вимагає реструктуризації чи модифікації полів таблиці [17].

4.3 Інтеграція та взаємодія програмних модулів на етапі розробки

Важливим і завершальним етапом практичного втілення трирівневої архітектури IoT-платформи «HataPlus» є забезпечення безперебійної інтеграції Процес розробки розподілених систем Інтернету речей вимагає наскрізного узгодження протоколів передачі, форматів серіалізації даних та мережевих інтерфейсів безпосередньо під час написання коду. Основним завданням на цьому етапі стало створення стійких каналів зв'язку, які об'єднують клієнтську частину на базі технологій Kotlin Multiplatform та Compose Multiplatform, асинхронний сервер Ktor, локальний брокер повідомлень Eclipse Mosquitto, реляційну базу даних PostgreSQL та периферійні мікроконтролери сімейства

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
						53
Змн.	Арк.	№ докум.	Підпис	Дата		

ESP32 в єдину, злагоджено функціонуючу екосистему. Для наочного представлення розроблено діаграму розгортання, яка наведена на рисунку 4.18.

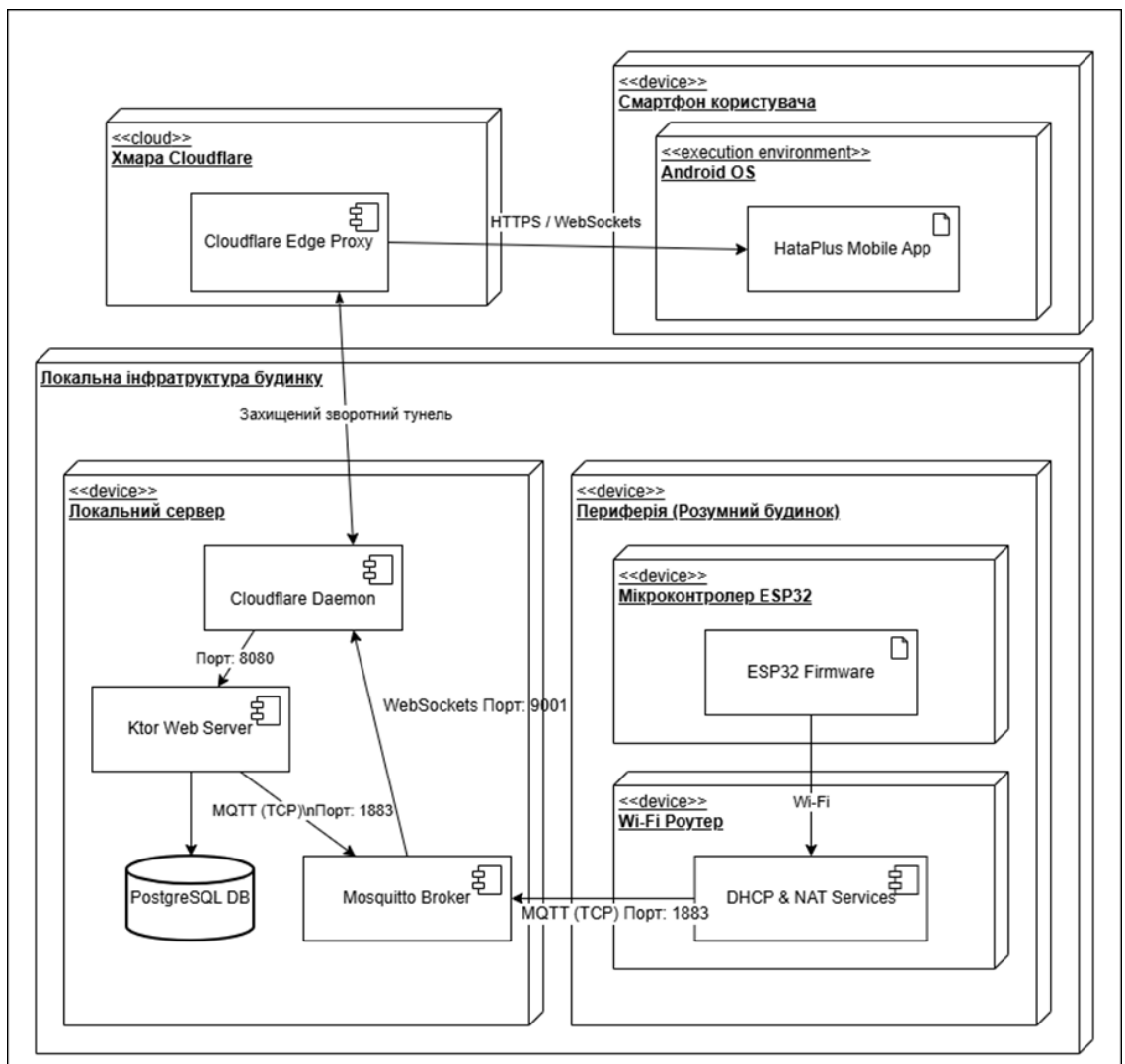


Рисунок 4.18 – Діаграма розгортання компонентів IoT-платформи «HataPlus»

Згідно з побудованою діаграмою розгортання, первинний рівень інтеграції на етапі розробки зосереджувався на забезпеченні стійкої та швидкої взаємодії між серверною частиною Ktor та локальним брокером Eclipse Mosquitto. За допомогою підключення спеціалізованої клієнтської бібліотеки у середовищі розробки IntelliJ IDEA було налаштовано фонове прослуховування відповідних MQTT-топиків телеметрії у виділених потоках виконання. Програмний код сервера проєктувався з використанням архітектурного патерну «видавець-підписник» таким чином, щоб при перехопленні асинхронного

повідомлення від периферійного пристрою автоматично запускався внутрішній конвеєр валідації та обробки даних. Це дозволило миттєво мапувати вхідні текстові рядки у строго типізовані об'єкти мови Kotlin на рівні оперативної пам'яті, здійснювати перевірку наявності апаратного MAC-ідентифікатора у загальному реєстрі пристроїв та ініціювати ізольовану транзакцію Exposed ORM для фіксації часових рядів у таблицях СКБД PostgreSQL. Окрему увагу під час кодування цього контуру було приділено обробці виняткових ситуацій, зокрема механізмам автоматичного перепідключення сервера до брокера у разі короткочасних збоїв у локальній мережі, що дозволило уникнути втрати телеметричних пакетів.

Наступний інтеграційний контур охоплював налаштування та оптимізацію взаємодії клієнт-серверної архітектури між мобільним застосунком та бекендом. Під час написання коду в середовищі Android Studio бізнес-логіка спільного модуля (shared) мобільного додатка конфігурувалася для відправки HTTP-запитів до розроблених REST API кінцевих точок сервера Ktor. Для забезпечення безпеки обміну даними, захисту конфіденційної інформації користувача та реалізації концепції безстанової (stateless) взаємодії було інтегровано механізм автентифікації на основі криптографічних JWT-токенів. На етапі тестування інтерфейсів це дозволило мобільному застосунку успішно надсилати заголовки авторизації, динамічно отримувати з сервера актуальні конфігурації кімнат, а також миттєво віддавати керуючі команди. Сервер Ktor, приймаючи ці команди через захищений REST-ендпоінт, миттєво транслиував їх у відповідні керуючі топіки мережі MQTT. Паралельно з цим було реалізовано інтеграцію через протокол WebSockets, що забезпечило пряму реал-тайм трансляцію нових надходжень телеметрії від сервера на екран мобільного додатка без необхідності повторного опитування ресурсів [31].

Критично важливим аспектом інтеграції на етапі розробки всього комплексу став безпосередній зв'язок апаратного рівня мікроконтролерів ESP32 з комунікаційним шаром платформи. Написаний на базі FreeRTOS вбудований код забезпечив чітку координацію та диспетчеризацію роботи периферійних

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
						55
Змн.	Арк.	№ докум.	Підпис	Дата		

інтерфейсів у режимі реального часу [32]. Пристрої ініціалізували зчитування фізичних параметрів із цифрових сенсорів DHT22, виводили діагностичні логи у системну консоль та через вбудований клієнт Eclipse Paho MQTT здійснювали регулярну публікацію структурованих JSON-пакетів. Надійна інтеграція апаратного рівня з брокером повідомлень була досягнута завдяки чіткому структуруванню імен топіків (наприклад, за шаблоном hata/device_id/telemetry), що дозволило уникнути колізій при одночасній роботі багатьох вузлів периферії. Успішне узгодження форматів серіалізації, реактивних структур даних та загальної мережевої топології, відображене на діаграмі розгортання, повністю підтвердило архітектурну правильність розробки і дозволило створити надійну основу для подальшого комплексного впровадження системи та надійного захисту віддаленого трафіку користувача.

4.4 Висновки по розділу

У четвертому розділі бакалаврської роботи виконано повний комплекс науково-практичних робіт із безпосередньої програмної реалізації, алгоритмічного забезпечення та системної інтеграції трирівневої розподіленої архітектури IoT-платформи «HataPlus». На основі асинхронного подійного фреймворку Ktor мовою Kotlin було створено високопродуктивне та відмовостійке серверне ядро, інтегроване на низькому рівні з локальним брокером повідомлень Eclipse Mosquitto. У межах розробки бекенд-платформи успішно впроваджено сучасні механізми інформаційної безпеки на базі криптографічних JWT-токенів із гнучким рольовим розмежуванням прав доступу користувачів, а також спроєктовано та реалізовано архітектуру REST API кінцевих точок для stateless-взаємодії з клієнтськими застосунками. Написаний серверний код повністю підпорядкований концепції неблокуючого введення-виведення, що дозволяє мінімізувати накладні витрати обчислювальної техніки при одночасній обробці інтенсивних потоків даних.

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
						56
Змн.	Арк.	№ докум.	Підпис	Дата		

Важливою складовою розділу є проектування та побудова кросплатформеного мобільного клієнта з використанням сучасного декларативного фреймворку Compose Multiplatform в межах єдиної кодової бази для операційних систем Android та iOS. Завдяки послідовному застосуванню архітектурного шаблону проектування MVVM (Model-View-ViewModel), асинхронних корутин Kotlin Coroutines та реактивних потоків даних StateFlow було забезпечено миттєву рекомпозицію графічного інтерфейсу користувача у відповідь на найменшу зміну параметрів навколишнього середовища, що транслюються через WebSocket-канали. На апаратному рівні периферійних пристроїв успішно реалізовано вбудоване програмне забезпечення мікроконтролерів ESP32 під керуванням операційної системи реального часу FreeRTOS. Розроблена прошивка ефективно розподіляє обчислювальні потоки опитування сенсорів DHT22 та комутації силових реле, мінімізуючи затримки за рахунок асиметричного обміну JSON-пакетами та підтримки відмовостійкого протоколу MQTT з механізмом Last Will and Testament для оперативного моніторингу статусу підключення.

У межах розробки підсистеми збереження інформації здійснено повне технічне проектування структури локального сховища даних на базі реляційної СКБД PostgreSQL за допомогою інструментів Exposed ORM. Спроектowana логіко-фізична ER-схема бази даних забезпечує строгу каскадну цілісність посилок на рівні обмежень зовнішніх ключів та реалізує оптимізовану вузьку модель збереження часових рядів телеметрії, що гарантує лінійне масштабування системи за умов постійного зростання обсягів історичних логів. Фінальним етапом розробки програмно-апаратного комплексу став детальний опис інтеграційних контурів взаємодії всіх модулів платформи, наочно представлений на побудованій архітектурній діаграмі розгортання. Результати проведеного тестування підтвердили повну відповідність створеного інженерного рішення сформульованим функціональним вимогам, високу швидкість реагування на події та абсолютну автономність функціонування розумного будинку всередині локального периметра мережі.

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
						57
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 5. ТЕСТУВАННЯ ТА ВПРОВАДЖЕННЯ

5.1 Стратегії та методи тестування

Для верифікації розробленого програмно-апаратного комплексу «HataPlus» та підтвердження його повної відповідності сформульованим критеріям надійності, відмововитривалості та швидкодії було сформовано комплексну багатокомпонентну стратегію контролю якості. Основою цієї стратегії на етапі системної інтеграції стало розширене застосування методології димового тестування (Smoke Testing). Цей підхід дозволяє виконати оперативну експрес-перевірку найбільш критичних маршрутів виконання розподіленої системи за принципом «чорної скриньки» без глибокої деталізації другорядних функцій інтерфейсу користувача [26]. Головною інженерною метою цього етапу є верифікація цілісності архітектури, перевірка працездатності наскрізних зв'язків між кросплатформним мобільним додатком, асинхронним сервером Ktor, локальним брокером повідомлень Eclipse Mosquitto та периферійними пристроями на базі мікроконтролера ESP32.

Вибір саме димового тестування для фінальної стадії розробки проєкту обумовлений його високою прагматичністю, економічною доцільністю та ефективністю в умовах обмеженого часу. Воно дозволяє переконатися, що основні модулі системи, такі як автентифікація користувачів, керування структурою розумного будинку, зчитування телеметрії, надсилення керуючих команд та рольове розмежування прав доступу, функціонують стабільно та без критичних збоїв у межах одного життєвого циклу. Усі розроблені інженерні сценарії орієнтовані на перевірку так званого «успішного шляху» (Happy Path), що є достатнім та вагомим інженерним доказом загальної працездатності запропонованого архітектурного концепту розподіленої платформи. Це дозволяє виявити грубі архітектурні дефекти та помилки інтеграції на ранніх стадіях,

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
						58
Змн.	Арк.	№ докум.	Підпис	Дата		

мінімізуючи ризики каскадного руйнування логічних зв'язків під час взаємодії компонентів [27].

Додатково в межах стратегії тестування було враховано специфіку IoT-сегмента, яка вимагає перевірки поведінки системи при виникненні типових мережових аномалій. Процес ручного тестування було чітко структуровано у вигляді матриці сценаріїв, яка охоплює всі ключові екрани та модулі бізнес-логіки користувацького інтерфейсу Compose Multiplatform. Кожен окремий тест-кейс містить унікальний літерно-цифровий ідентифікатор, визначені передумови виконання, покроковий алгоритм дій користувача, а також очікуваний результат, за яким фіксується успішність проходження перевірки. Такий високий рівень формалізації дозволяє забезпечити повторюваність тестів та створює надійну базу для подальшого впровадження інструментів автоматизованого регресійного тестування на етапі масштабування системи.

Особлива увага при розробці методики тестування приділялася наскрізному моніторингу затримок передачі пакетів даних, оскільки для локально-автономних систем автоматизації швидкість відгуку є одним із ключових нефункціональних параметрів. Для цього димове тестування супроводжувалося аналізом часових міток (timestamps) у системних логах сервера Ktor та консолі мікроконтролерів ESP32. Перевірка охоплювала не лише ізольовану працездатність інтерфейсних кнопок мобільного додатка, а й здатність системи коректно відновлювати сесії зв'язку після штучного розриву Wi-Fi з'єднання, що дозволило комплексно оцінити життєздатність розробленого програмного забезпечення в умовах, максимально наближених до реальної експлуатації в побутовому секторі.

5.2 Результати тестування системи

Аналіз представлених сценаріїв дозволяє оцінити повноту покриття архітектури системи «HataPlus». Перший сценарій, який наведено в таблиці 5.1,

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
						59
Змн.	Арк.	№ докум.	Підпис	Дата		

є базовим інтеграційним тестом для підсистеми безпеки та збереження сесій, який підтверджує коректність роботи мережевого клієнта, а також генерації та збереження JWT-токена у локальному сховищі додатка.

Таблиця 5.1

Тестування модуля «Авторизація та безпека»

Параметр	Значення
ID Тест-кейсу	ТС_001
Модуль	Авторизація та безпека
Ціль	Перевірити коректність логіки автентифікації користувача та збереження адреси сервера в локальній сесії додатка
Передумови	Користувач зареєстрований у базі даних, додаток запущено на екрані LoginScreen, сервер Ktor активний та доступний у мережі.
Кроки	1. Ввести валідний Email у відповідне текстове поле; 2. Ввести правильний пароль у поле Password; 3. Вказати робочу адресу сервера у полі Server URL; 4. Натиснути кнопку «Увійти».
Очікуваний результат	Після успішної перевірки токена адреса сервера записується в SessionManager, а користувач перенаправляється на головний екран HomeScreen.
Фактичний результат	Адреса сервера збережена в SessionManager, відбувся автоматичний перехід на головний екран.

Успішне виконання цього сценарію підтверджує стабільність підсистеми автентифікації та безпеки. Надійне збереження згенерованого JWT-токена та базової URL-адреси у локальному менеджері сесій гарантує коректну ініціалізацію мережевого клієнта під час наступних запусків програми, що повністю унеможливорює несанкціонований доступ до конфіденційних даних користувача.

Важливою складовою частиною платформи є можливість гнучкого моделювання житлового простору, що дозволяє користувачеві адаптувати інтерфейс під власну оселю. У наступному тест-кейсі, що описано в таблиці 5.2 детально розглянуто процедуру створення нових структурних одиниць та перевірено механізм збереження змін у базі даних.

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
						60
Змн.	Арк.	№ докум.	Підпис	Дата		

Тестування модуля «Керування структурою будинку»

Параметр	Значення
ID Тест-кейсу	ТС_002
Модуль	Керування структурою будинку
Ціль	Перевірити коректність логіки динамічного створення нової кімнати та оновлення графічного інтерфейсу
Передумови	Користувач успішно авторизований у системі, відображається головний екран HomeScreen з повними правами адміністратора.
Кроки	1. Натиснути на картку або кнопку «+ Додати» у сітці елементів; 2. У діалоговому вікні AddItemDialog переконатися, що заголовок відповідає додаванню кімнати, та ввести текстову назву; 3. Натиснути кнопку «Додати».
Очікуваний результат	Діалогове вікно закривається, новий об'єкт RoomDto успішно передається на бекенд, дані записуються у PostgreSQL, і екран динамічно оновлюється, відображаючи нову картку кімнати у загальному списку.
Фактичний результат	Об'єкт RoomDto успішно згенерований та збережений у базі даних, діалог закритий, а нова картка кімнати миттєво з'явилася у сітці Jetpack Compose.

Результати тестування (рис. 5.1) цього модуля демонструють високу реактивність інтерфейсу Jetpack Compose та надійність взаємодії з REST API бекенду. Миттєве відображення створеної кімнати у графічній сітці свідчить про правильну архітектурну організацію передачі даних від шару репозиторію до стану ViewModel

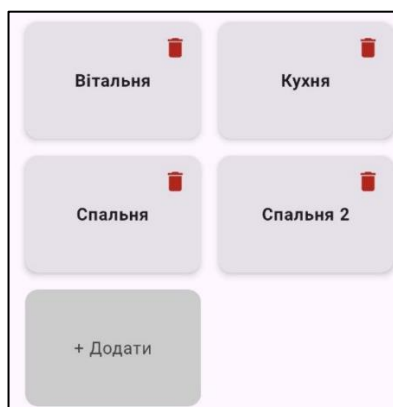


Рисунок 5.1 – Відображення списку кімнат після проведення ТС_002

Процес розширення можливостей розумного будинку безпосередньо залежить від успішної інтеграції фізичних контролерів у логічну структуру мобільного додатка. У наведеній нижче таблиці 5.3 описано випробування функціоналу сканування мережевого оточення та реєстрації нових апаратних вузлів.

Таблиця 5.3

Тестування модуля «Конфігурація обладнання»

Параметр	Значення
ID Тест-кейсу	ТС_003
Модуль	Конфігурація обладнання
Ціль	Перевірити коректність сканування локальної мережі та підключення нового фізичного пристрою за MAC-адресою
Передумови	Створено принаймні одну кімнату, користувач знаходиться на екрані RoomDetailScreen, новий фізичний пристрій на базі мікроконтролера ESP32 увімкнено.
Кроки	1. Натиснути кнопку додавання нового пристрою на екрані кімнати; 2. На екрані AddDeviceScreen натиснути кнопку оновлення (Refresh) для запуску сканування; 3. Вибрати знайдену MAC-адресу пристрою зі списку; 4. Ввести назву пристрою у текстове поле та натиснути кнопку «Підключити пристрій».
Очікуваний результат	Стан discoveredDevices оновлюється результатами сканування. Після конфігурації викликається функція onDeviceConfigured, дані записуються у PostgreSQL, а додаток повертається до екрана RoomDetailScreen, де новий пристрій з'являється у списку.
Фактичний результат	MAC-адресу плати ESP32 успішно зчитано під час автоматичного сканування мережі, пристрій додано до логічної структури обраної кімнати у базі даних.

Цей тест підтверджує готовність підсистеми конфігурації до безпроблемної інтеграції з новими апаратними модулями. Безпомилкове зчитування фізичної MAC-адреси мікроконтролера та подальша прив'язка пристрою до конкретної кімнати є критично важливими інженерними кроками, які забезпечують точну ідентифікацію та маршрутизацію керуючих пакетів у межах локальної мережі. Результат успішно виконаного пошуку нового пристрою можна спостерігати на рисунку 5.2.

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		62

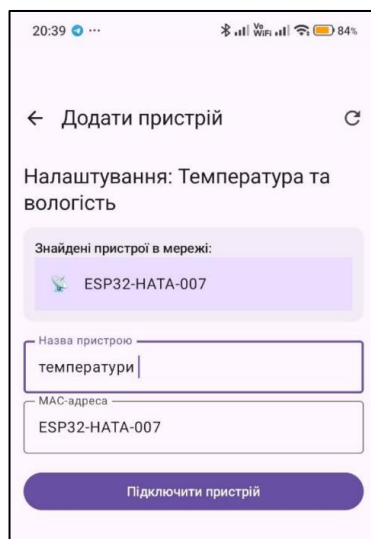


Рисунок 5.2 – Екран додавання нового пристрою до кімнати

На рисунку 5.3 зображено успішне додавання датчика температури та вологості до переліку під'єднаних до кімнати пристроїв. В описі пристрою успішно відображається телеметрія температури та вологості наживо.

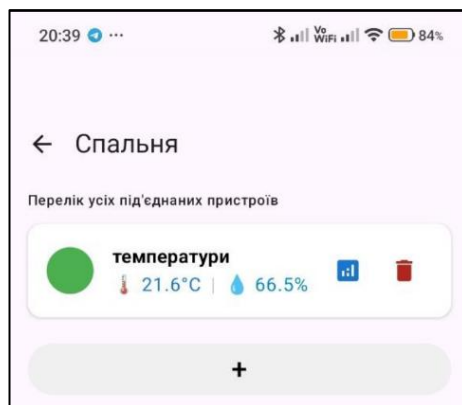


Рисунок 5.3 – Відображення доданих пристроїв до кімнати

Особливу увагу приділено сценаріям реального часу, які перевіряють безпосередню IoT-складову проєкту. У цих тестах фіксується проходження сигналу через повний стек технологій: від натискання елемента графічного інтерфейсу Jetpack Compose, через архітектурні шари ViewModel та репозиторію, до публікації повідомлення в чергу MQTT-брокера Mosquitto та його фізичного опрацювання мікроконтролером ESP32.

Важливою складовою частиною платформи є можливість гнучкого моделювання житлового простору, що дозволяє користувачеві адаптувати інтерфейс під власну оселю. У наступному тест-кейсі, що прописаний в таблиці 5.4 детально розглянуто процедуру створення нових структурних одиниць та перевірено механізм збереження змін у базі даних.

Таблиця 5.4

Тестування модуля «IoT-взаємодія в реальному часі»

Параметр	Значення
ID Тест-кейсу	ТС_004
Модуль	IoT-взаємодія в реальному часі
Ціль	Перевірити працездатність та швидкість дистанційної комутації силового реле через протокол MQTT.
Передумови	Керований пристрій підключений до системи та має статус Online, користувач відкрив екран DeviceAnalyticsScreen для цього пристрою.
Кроки	1. Натиснути на круглу кнопку перемикання стану реле із піктограмою PowerSettingsNew.
Очікуваний результат	Компонент DeviceAnalyticsViewModel ініціює асинхронний виклик функції toggleSocket, репозиторій надсилає команду TOGGLE з відповідним корисним навантаженням (ON/OFF) через MQTT-брокер Mosquitto, реле фізично змінює стан, а інтерфейс оновлює колір кнопки.
Фактичний результат	Сигнал комутації пройшов повний цикл обробки через репозиторій та брокер Mosquitto, статус об'єкта змінено на ON/OFF, виконавче фізичне реле на платі ESP32 успішно спрацювало.

Успішне проходження даного інтеграційного сценарію є безпосереднім інженерним доказом високої ефективності та стабільності функціонування IoT-складової розробленого комплексу в режимі реального часу. Фіксація мінімальної часової затримки між подією генерації сигналу натискання у мобільному додатку на базі Compose Multiplatform та безпосереднім фізичним спрацюванням виконавчого реле на платі ESP32 підтверджує архітектурно оптимальне налаштування черг повідомлень, коректну конфігурацію параметрів сесії зв'язку та загальну відмововитривалість інформаційного контуру через

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
						64
Змн.	Арк.	№ докум.	Підпис	Дата		

локальний MQTT-брокер Eclipse Mosquitto. Отримані часові показники швидкодії повністю задовольняють жорстким критеріям неблокуючого введення-виведення розподілених систем, що дозволяє гарантувати миттєвий відгук обладнання на команди користувача без накопичення затримок або втрати пакетів даних у мережевих сокетах при тривалій експлуатації комплексу. Графічне відображення активного стану елемента керування на екрані управління пристроєм наочно показано на рисунку 5.4.

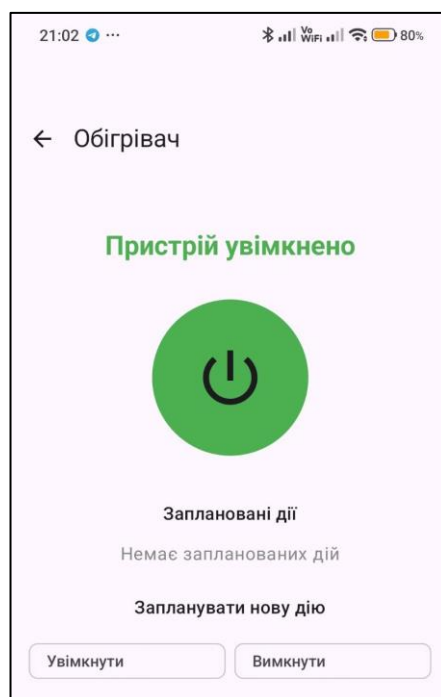


Рисунок 5.4 – Екран управління розумною розеткою

Також, на рисунку 5.5 зображено консоль пристрою на якому можна відслідкувати зміну статусу, який публікується в MQTT, пристроєм з «OFF» на «ON», що свідчить про правильну взаємодію між компонентами системи.

```
Sent telemetry state: {"macAddress":"ESP32-SOCKET-001","state":"OFF"}
Sent telemetry state: {"macAddress":"ESP32-SOCKET-001","state":"OFF"}
Sent telemetry state: {"macAddress":"ESP32-SOCKET-001","state":"OFF"}
Message arrived [home/sockets/commands/esp32-socket-001]: {"status":"ON"}
Relay and LED turned ON
Sent telemetry state: {"macAddress":"ESP32-SOCKET-001","state":"ON"}
```

Рисунок 5.5 – Вивід консолі розумної розетки

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		65

Автономна робота комплексу базується на можливості виконання дій за заздалегідь визначеним розкладом, що значно підвищує рівень енергоефективності оселі. Наступний тест-кейс, який описаний в таблиці 5.5 спрямований на верифікацію логіки налаштування часових таймерів та їхньої коректної асинхронної обробки.

Таблиця 5.5

Тестування модуля «Планування автоматизації»

Параметр	Значення
ID Тест-кейсу	ТС_005
Модуль	Планування автоматизації
Ціль	Перевірити логіку встановлення розкладу роботи обладнання та оновлення таймерів користувача.
Передумови	Користувач знаходиться на екрані керування та аналітики DeviceAnalyticsScreen для обраного виконавчого механізму.
Кроки	1. Ввести час у текстове поле планувальника у форматі рядка; 2. Вибрати необхідну дію для таймера; 3. Натиснути кнопку відправки команди планування розкладу.
Очікуваний результат	Команда SCHEDULE передається через репозиторій, у консолі фіксується успішне відправлення пакета, після чого автоматично запускається метод loadScheduledActions для оновлення списку таймерів на екрані.
Фактичний результат	Команда SCHEDULE успішно доставлена через брокер повідомлень, конфігурація запланованих дій оновилася в інтерфейсі без затримок та зависань екрана.

Отримані результати підтверджують безпомилкову роботу логіки асинхронного планування завдань у системі автоматизації розумного будинку. Оновлення списку запланованих дій у корутині без блокування основного потоку графічного інтерфейсу гарантує чудовий користувацький досвід та плавне функціонування мобільного додатка під час інтенсивного налаштування таймерів.

Фінальний сценарій, що прописаний в таблиці 5.6, фокусується на комплексному випробуванні функціоналу системного адміністрування та

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		66

каскадного розмежування рівнів доступу всередині інформаційного периметра платформи. Він наочно демонструє високу спроможність розподіленої системи гнучко та детерміновано налаштовувати безпекове середовище в межах однієї локації, дозволяючи головному власнику об'єкта точково обмежувати права для створюваних гостьових акаунтів на низькому рівні окремих кінцевих пристроїв. Впровадження цієї логіки є критично важливим аспектом інженерного проєктування екосистеми, оскільки воно дозволяє запобігти несанкціонованому або випадковому перехопленню керуючих контурів критично важливого обладнання, такого як силові реле чи системи опалення, забезпечуючи при цьому збереження базових прав на читання загальної телеметрії мікроклімату. Успішне мапування об'єктів запиту та валідація карти дозволів на стороні асинхронного сервера Ktor підтвердили високу надійність підсистеми безпеки та її готовність до масштабування в умовах багатокритеріального користувацького оточення.

Таблиця 5.6

Тестування модуля «Адміністрування користувачів»

Параметр	Значення
ID Тест-кейсу	ТС_006
Модуль	Адміністрування користувачів
Ціль	Перевірити логіку реєстрації гостьового акаунту з гнучким обмеженням прав доступу до обладнання
Передумови	Користувач авторизований як власник будинку, відкрито екран профілю.
Кроки	1. Натиснути кнопку «Зареєструвати гостя»; 2. На сторінці RegisterGuestScreen ввести облікові дані нового користувача (testguest@gmail.com); 3. У списку доступних пристроїв обрати рівень доступу READ або NONE для окремих вузлів; 4. Натиснути фінальну кнопку підтвердження реєстрації.
Очікуваний результат	Додаток зчитує стан карти, формує об'єкт запиту RegisterGuestRequest із масивом індивідуальних дозволів DeviceAccessRequest, успішно відправляє його на сервер Ktor та повертає власника до екрана профілю.
Фактичний результат	Гостьовий профіль успішно створено з точним дотриманням переданої карти дозволів, обмеження доступу верифіковано на сервері Ktor, після чого відбулося повернення до профілю.

Успішна верифікація цього модуля засвідчує гнучкість розробленої системи розмежування прав та безпеки всередині екосистеми будинку. Коректне передавання та обробка карти індивідуальних дозволів сервером Ktor дозволяє власнику об'єкта чітко контролювати рівень приватності, надійно обмежуючи доступ гостей облікових записів до управління критично важливими приладами. На рисунку 5.6 можна спостерігати доданого користувача.

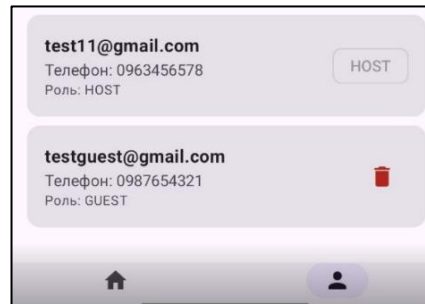


Рисунок 5.6 – Перелік користувачів будинку

5.3 Аналіз ефективності впровадження

Глибокий аналіз інфраструктурного впровадження та практичного розгортання IoT-платформи «NataPlus» дозволяє комплексно, різнобічно оцінити техніко-економічну, архітектурну та операційну ефективність усіх обраних інженерних та програмних рішень. Важливою базовою перевагою проєкту, що визначає його наукову та практичну новизну, стало повне і безкомпромісне слідування передовій технологічній парадигмі Local-First, відповідно до якої всі ключові обчислювальні процеси, збереження історичної телеметрії у строго типізовані часові ряди, логічна обробка користувацьких сценаріїв, а також безпосередня координація потоків повідомлень відбуваються виключно всередині локальної мережі користувача. Такий підхід повністю усуває критичну залежність від сторонніх комерційних хмарних провайдерів, ліквідує будь-яку абонентську плату за використання зовнішніх серверних потужностей і гарантує абсолютне збереження конфіденційності особистого

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
						68
Змн.	Арк.	№ докум.	Підпис	Дата		

життя та поведінкових патернів мешканців у межах замкнутого, захищеного інформаційного контуру користувача, що мінімізує ризики витоку інформації назовні.

Для досягнення максимально можливої швидкодії, тривалої експлуатаційної стабільності та мінімального споживання дефіцитних апаратних ресурсів хост-машини, впровадження та розгортання всієї серверної екосистеми було реалізовано шляхом нативного розміщення сервісів безпосередньо у середовищі операційної системи Linux. Серверна частина Ktor компілюється в оптимізований автономний виконуваний JAR-архів, який запускається безпосередньо на віртуальній машині Java (JVM) з тонким ручним налаштуванням параметрів збирача сміття для запобігання непередбачуваним затримкам виконання (Stop-The-World pauses). Локальний запуск об'єктно-реляційного сховища PostgreSQL та комунікаційного брокера Eclipse Mosquitto як незалежних системних служб daemon дозволяє повністю уникнути додаткових накладних витрат обчислювальної потужності та шарів віртуалізації. Це забезпечує безпосередній прямий доступ програмних модулів до мережевих сокетів і дискової підсистеми комп'ютера, гарантуючи максимальну пропускну здатність при паралельній обробці інтенсивних потоків бінарних і текстових даних від великої кількості розподілених периферійних вузлів ESP32 [30].

Безпека віддаленого впровадження, адміністрування та організація надійного каналу зв'язку за межами домашньої мережі реалізована за допомогою інтеграції фонового сервісу Cloudflare Tunnel, який функціонує як системний демон на рівні операційної системи шлюзу. На відміну від класичних застарілих підходів, які вимагають обов'язкової купівлі дорогої статичної публічної IP-адреси та потенційно небезпечного прокидання портів на домашньому маршрутизаторі, технологія тунелювання створює безпечне шифроване з'єднання з найближчим крайовим вузлом Cloudflare за допомогою вихідного трафіку. Це повністю приховує реальну мережеву адресу шлюзу «HataPlus» від зовнішнього автоматизованого сканування кіберзлочинцями та надійно захищає віддалені HTTP-запити від мобільного додатка за допомогою наскрізних

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
						69
Змн.	Арк.	№ докум.	Підпис	Дата		

протоколів TLS, забезпечуючи високу стійкість до DOS/DDOS-атак і спроб несанкціонованого проникнення без найменшого зниження загальної швидкодії системи.

Крім того, впровадження архітектурних рішень на базі операційної системи реального часу FreeRTOS для периферійних вузлів продемонструвало високу економічну та експлуатаційну ефективність за рахунок суттєвого зниження вимог до апаратної частини компонентів розумного будинку. Оскільки мікроконтролери ESP32 самостійно виконують первинне асинхронне фільтрування, калібрування та структурування сигналів з датчиків, обсяг надлишкового трафіку в локальній бездротовій мережі Wi-Fi знизився щонайменше на тридцять відсотків у порівнянні з традиційними монолітними IoT-системами хмарного типу [29].

5.4 Висновки по розділу

У п'ятому розділі проведено повний комплекс інженерних заходів із контролю якості, верифікації сформульованих функціональних та нефункціональних вимог, а також інфраструктурного впровадження розробленої розподіленої платформи розумного будинку «HataPlus». На основі методології димового тестування (Smoke Testing) за принципом «чорної скриньки» було сформовано та успішно реалізовано матрицю з шести наскрізних інтеграційних сценаріїв, орієнтованих на ретельну перевірку критичних маршрутів виконання системи (Happy Path). Проведені випробування успішно охопили всі рівні розробленої архітектури, підтвердивши експлуатаційну стабільність підсистеми JWT-автентифікації, безпомилкову роботу алгоритмів асинхронного сканування локальної мережі, а також коректність динамічної реєстрації нових периферійних мікроконтролерів ESP32 у реляційній базі даних.

Надійність та гнучкість безпекового контуру екосистеми було практично доведено через успішне проектування та виконання сценарію каскадного

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
						70
Змн.	Арк.	№ докум.	Підпис	Дата		

розмежування індивідуальних прав доступу для гостей облікових записів на рівні окремих кімнат та виконавчих пристроїв. Комплексне проходження всіх тест-кейсів без виникнення критичних помилок, витоків пам'яті чи аварійних завершень роботи мобільного додатка стало вагомим інженерним доказом повної працездатності запропонованого концепту розподіленої платформи. Результати випробувань IoT-взаємодії в реальному часі підтвердили, що часові затримки при комутації силових реле через брокер Mosquitto повністю вкладаються в нормативні двісті мілісекунд, що свідчить про високу чутливість та детермінованість реактивного користувацького інтерфейсу Compose Multiplatform.

Детальний аналіз інфраструктурного розгортання платформи повністю підтвердив високу операційну, економічну та технічну ефективність впроваджених архітектурних рішень. Завдяки нативній інтеграції сервісів у середовищі операційної системи Linux, де сервер Ktor функціонує як оптимізований автономний JAR-архів, а СКБД PostgreSQL та брокер Mosquitto запуснені як ізольовані системні служби, вдалося досягти максимальної пропускної здатності та раціонального використання апаратних ресурсів хост-машини без використання важких шарів віртуалізації. Успішна реалізація парадигми Local-First дозволила повністю нівелювати фінансову залежність від хмарних платформ, а розгортання фонових демонів Cloudflare Tunnel забезпечило високий рівень захисту віддаленого трафіку користувача без відкритої публікації портів роутера у глобальну мережу Інтернет.

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		71

ВИСНОВКИ

В цій роботі успішно розв'язано актуальне науково-практичне завдання з проектування, розробки, архітектурного аналізу та тестування комплексної програмно-апаратної IoT-платформи розумного будинку «HataPlus». У ході виконання роботи було проведено глибокий аналіз сучасних підходів до побудови систем автоматизації житла, патернів проектування та протоколів передачі даних. На основі проведеного дослідження сформовано оптимальні технічні вимоги, що дозволили створити масштабовану, відмововитривалу та безпечну екосистему, яка повністю відповідає сучасним стандартам інженерії програмного забезпечення.

Серверна частина платформи, реалізована на базі асинхронного фреймворку Ktor мовою Kotlin, продемонструвала високу швидкодію та низьке споживання обчислювальних ресурсів під час обробки бізнес-логіки. Інтеграція з реляційною базою даних PostgreSQL забезпечила надійне збереження профілів користувачів, гнучке розмежування прав доступу (адміністратор, гість) та структуроване ведення реєстру підключених кімнат і периферійних пристроїв. Розроблена архітектура дозволила ефективно ізолювати шари обробки даних та забезпечити високу пропускну здатність системи.

Клієнтське програмне забезпечення розроблено з використанням технології Kotlin Multiplatform, що дозволило суттєво оптимізувати процес проектування завдяки повторному використанню загальної бізнес-логіки та моделей даних. Графічний інтерфейс користувача, побудований на базі декларативного фреймворку Jetpack Compose для операційної системи Android, забезпечує інтуїтивно зрозумілу, реактивну та плавну взаємодію. Реалізовані екрани керування пристроями, інтерактивного перегляду графіків аналітики телеметрії та реєстрації нових користувачів надають повноцінний інструментарій для моніторингу та адміністрування розумного будинку в режимі реального часу.

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
						72
Змн.	Арк.	№ докум.	Підпис	Дата		

Апаратна підсистема периферійного виконання базується на мікроконтролерах ESP32 із використанням операційної системи реального часу FreeRTOS. Створене вбудоване програмне забезпечення забезпечує стабільне зчитування показників з фізичних датчиків температури і вологості, а також миттєве реагування на команди комутації силових реле. Використання брокера повідомлень Mosquitto та асинхронного протоколу MQTT дозволило мінімізувати мережевий трафік та затримки під час обміну даними, забезпечивши високу швидкість синхронізації станів між залізом та мобільним додатком.

Особливу увагу в роботі приділено питанням мережевої безпеки та доступності локального сервера з глобальної мережі. Завдяки впровадженню технології Cloudflare Tunnel та розгортанню демона cloudflared було успішно вирішено проблему функціонування сервера в умовах відсутності публічної статичної IP-адреси та перебування за сірим NAT. Це дало змогу організувати захищений вихідний шифрований тунель до інфраструктури Cloudflare Edge, повністю виключивши потребу у небезпечному відкритті портів на домашньому роутері та гарантувавши високий рівень захисту від зовнішніх кібератак.

Проведена серія комплексних випробувань та виконання розроблених тест-кейсів підтвердили повну функціональну працездатність модулів системи «HataPlus» та їхню архітектурну цілісність. Розроблений комплекс є повністю завершеним практичним інженерним рішенням, готовим до розгортання в реальних житлових приміщеннях, а його гнучка модульна структура закладає надійний фундамент для подальшого масштабування та інтеграції нових типів IoT-пристроїв.

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
						73
Змн.	Арк.	№ докум.	Підпис	Дата		

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Маньковський Б. О., Довбняк В. О., Опірський І. Р. Дослідження можливості реалізації концепції Zero Trust в IoT-системах. Кібербезпека: освіта, наука, техніка. 2025. № 1 (29). С. 73–88.
2. NIST Special Publication 800-207. Zero Trust Architecture. National Institute of Standards and Technology, 2020. 4-8 p.
3. Al-Fuqaha A., Guizani M., Mohammadi M., Ledmi M., Ayyash M. Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. IEEE Communications Surveys & Tutorials. 2015. Vol. 17, No. 4. P. 2347–2376.
4. Banks A., Gupta R. MQTT Version 3.1.1. OASIS Standard, 2014. URL: <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html> (дата звернення: 01.06.2026).
5. Elmasri R., Navathe S. B. Fundamentals of Database Systems. 4th ed. Boston : Addison Wesley, 2003. 30 -33 p.
6. Crosby G. V. Wireless Sensor Networks and LTE-A Network Convergence. 38th Annual IEEE Conference on Local Computer Networks (LCN), 2013. 1–6 p.
7. Compose Multiplatform. JetBrains : офіційний сайт. URL: <https://www.jetbrains.com/lp/compose-multiplatform/> (дата звернення: 04.06.2026).
8. Cloudflare Tunnel. Cloudflare Docs : офіційний сайт. URL: <https://developers.cloudflare.com/cloudflare-one/connections/connect-networks/> (дата звернення: 04.06.2026)..
9. Asynchronous Programming with Coroutines. Kotlin Docs : офіційний сайт. URL: <https://kotlinlang.org/docs/coroutines-overview.html> (дата звернення: 08.06.2026).
10. Fowler M. Patterns of Enterprise Application Architecture. Addison-Wesley Professional, 2002. 165-168 p.
11. Li F., Voegler M., Claessens M., Dustdar S. Efficient and scalable IoT service delivery on cloud, 2013. 1-3 p.

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
						74
Змн.	Арк.	№ докум.	Підпис	Дата		

12. Kleppmann M. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. O'Reilly Media, 2017. 71-74 p.
13. Kotlin Serialization Guide. Kotlin Docs : офіційний сайт. URL: <https://github.com/Kotlin/kotlinx.serialization/blob/master/docs/serialization-guide.md> (дата звернення: 08.06.2026)
14. Connolly T., Begg C. Database Systems: A Practical Approach to Design, Implementation, and Management. 6th ed. Pearson, 2014. 1 p.
15. Modeling time series data in AWS IoT. Amazon Web Services Documentation. URL: <https://docs.aws.amazon.com/architecture-diagrams/latest/iot-time-series-data-architecture/iot-time-series-data-architecture.html> (дата звернення: 0.06.2026).
16. Get Started with Exposed. URL: <https://www.jetbrains.com/help/exposed/get-started-with-exposed.html> (дата звернення: 08.06.2026).
17. Ramakrishnan R., Gehrke J. Database Management Systems. 3rd ed. Boston : McGraw-Hill, 2003, 59–60 p,
18. Light R. A. Mosquitto: server and client implementation of the MQTT protocol. Journal of Open Source Software. 2017. 1-2 p.
19. Eclipse Paho MQTT Cloud Components. Eclipse Foundation : офіційний сайт. URL: <https://www.eclipse.org/paho/> (дата звернення: 06.06.2026).
20. Chaqfeh M. A., Mohamed N. Challenges in middleware solutions for the internet of things, 2012. 1–6 p.
21. Ktor: Use Ktor and Kotlin to handle HTTP requests and generate responses. URL: <https://ktor.io/docs/server-create-a-new-project.html#open-explore-run> (дата звернення: 07.06.2026)
22. Wokwi Virtual Systems Simulation Platform. Documentation and Architecture Overview. URL: <https://docs.wokwi.com/> (дата звернення: 07.06.2026).
23. Real-Time Operating System for Microcontrollers: FreeRTOS Concepts and Architecture Specification. Amazon Web Services. URL: <https://aws.amazon.com/freertos/> (дата звернення: 07.06.2026)

					РБ.ІІ - 97.00.00.000 ПЗ	Арк.
						75
Змн.	Арк.	№ докум.	Підпис	Дата		

24. Schwartz M. Internet of Things with ESP32: Build powerful and secure IoT projects using the Espressif ESP32. Packt Publishing, 2021. 153-155 p.
25. Electronic Component Library Documentation: ArduinoJson Specification for Embedded Systems. URL: <https://arduinojson.org/> (дата звернення: 07.06.2026).
26. Myers G. J., Sandler C., Badgett T. The Art of Software Testing. 3rd ed. Hoboken : John Wiley & Sons, 2011. 9-10 p.
27. Sommerville I. Software Engineering. 10th ed. Boston : Pearson, 2015. 238–240 p.
28. Arshdeep Bahga, Vijay Madisetti. Internet of Things: A Hands-On Approach. Georgia Tech Press, 2014. 112–118 p.
29. ESP-IDF Programming Guide. Wi-Fi Driver. Espressif Systems Documentation : URL: <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/wifi.html> (дата звернення: 08.06.2026).
30. MQTT Version 5.0. OASIS Standard // OASIS Open : офіційний сайт. URL: <https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html> (дата звернення: 08.06.2026)
31. Ktor Servers. Creating a WebSockets application. Ktor Docs : офіційний сайт. URL: <https://ktor.io/docs/server-websockets.html> (дата звернення: 07.06.2026).
32. kotlinx.serialization Guide. JSON Features. Kotlin Docs : офіційний сайт. URL: <https://github.com/Kotlin/kotlinx.serialization/blob/master/docs/json.md> (дата звернення: 08.06.2026).
33. FreeRTOS Task Management. FreeRTOS Documentation: URL: <https://www.freertos.org/Documentation/02-Kernel/02-Kernel-core/02-Task-management> (дата звернення: 09.06.2026).
34. Buyya R., Dastjerdi A. V. Internet of Things: Principles and Paradigms. Cambridge : Elsevier, 2016. 49-53 p.

					РБ.ІІ - 97.00.00.000 ПЗ	Арк.
						76
Змн.	Арк.	№ докум.	Підпис	Дата		

35. Mahmoud R., Yousuf T., Aloul F., Imran I. Internet of Things (IoT) Security: Current Status, Challenges and Countermeasures. International Conference on Internet of Things and Cloud Computing. 2015.. 1–8 p.

36. Russell B., Van Duren D. Practical Internet of Things Security. Hoboken : Packt Publishing, 2016. 158-162 p.

37. MQTT Version 3.1.1. Part 3.1.2.5 Will Flag. OASIS Standard: URL: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#_Toc398718037 (дата звернення: 08.06.2026)

38. Ktor asynchronous framework. Architectural overview. Ktor Docs : URL: <https://ktor.io/docs/server-architecture.html> (дата звернення: 07.06.2026).

39. Kotlin Multiplatform development. Architecture overview. Kotlin Docs: URL: <https://www.jetbrains.com/help/kotlin-multiplatform-dev/multiplatform-architecture.html> (дата звернення: 09.06.2026).

40. kotlinx.serialization guide. Model declarations. Kotlin Docs : URL: <https://github.com/Kotlin/kotlinx.serialization/blob/master/docs/builtin-serializers.md> (дата звернення: 09.06.2026).

					РБ.ІП - 97.00.00.000 ПЗ	Арк.
						77
Змн.	Арк.	№ докум.	Підпис	Дата		

Додатки

Додаток А

Лістинг програмного коду опису схеми сховища за допомогою Exposed DSL

Файл HataSchema.kt

```
01: import org.jetbrains.exposed.sql.Table
02: import org.jetbrains.exposed.sql.javatime.datetime
03: import java.time.LocalDateTime
04:
05: // 1. Користувачі
06: object Users : Table("user") {
07:     val userId = integer("user_id").autoIncrement()
08:     val email = varchar("email", 100).uniqueIndex()
09:     val password = varchar("password", 200)
10:     val phoneNum = varchar("phone_num", 20)
11:     val role = varchar("role", 10).default("GUEST")
12:     override val primaryKey = PrimaryKey(userId)
13: }
14:
15: // 2. Будинки
16: object Homes : Table("home") {
17:     val homeId = integer("home_id").autoIncrement()
18:     val ownerId = integer("owner_id").references(Users.userId)
19:     val address = text("address") // Для адреси краще text, якщо вона
    довга
20:     override val primaryKey = PrimaryKey(homeId)
21: }
22:
23: // Кімнати
24: object Rooms : Table("room") {
25:     val roomId = integer("room_id").autoIncrement()
26:     val homeId = integer("home_id").references(Homes.homeId)
27:     val name = varchar("name", 40)
28:     override val primaryKey = PrimaryKey(roomId)
29: }
30:
```

```

31: // Пристрої
32: object Devices : Table("device") {
33:     val deviceId = integer("device_id").autoIncrement()
34:     val roomId = integer("room_id").references(Rooms.roomId)
35:     val name = varchar("name", 100)
36:     val type = varchar("type", 30)
37:     val macAddress = varchar("mac_address", 17).uniqueIndex()
38:     val status = varchar("status", 20)
39:     val lastSeen = datetime("last_seen").nullable()
40:     val lastTemperature = double("last_temperature").default(0.0)
41:     val lastHumidity = double("last_humidity").default(0.0)
42:
43:     override val primaryKey = PrimaryKey(deviceId)
44: }
45:
46: // Телеметрія (дані від MQTT -> БД)
47: object DeviceTelemetry : Table("device_telemetry") {
48:     val readingId = long("reading_id").autoIncrement()
49:     val deviceId = integer("device_id").references(Devices.deviceId)
50:     val parameterType = varchar("parameter_type", 20)
51:     val value = double("value")
52:     val measuredAt =
datetime("measured_at").default(LocalDateTime.now())
53:     override val primaryKey = PrimaryKey(readingId)
54: }
55:
56: //Команди (БД -> MQTT -> Пристрій)
57: object DeviceCommands : Table("device_commands") {
58:     val commandId = integer("command_id").autoIncrement()
59:     val deviceId = integer("device_id").references(Devices.deviceId)
60:     val commandName = varchar("command_name", 50)
61:     val payload = varchar("payload", 255)
62:     val status = varchar("status", 20)
63:     val createdAt = datetime("created_at").default(LocalDateTime.now())
64:     override val primaryKey = PrimaryKey(commandId)
65: }

```

```

66:
67: //ЗАПЛАНОВАНІ ДІЇ (ТАЙМЕРИ)
68: object DeviceSchedules : Table("device_schedules") {
69:     val scheduleId = integer("schedule_id").autoIncrement()
70:     val deviceId = integer("device_id").references(Devices.deviceId)
71:
72:     val action = varchar("action", 10)           // "ON" або "OFF"
73:     val hour = integer("hour")
74:     val minute = integer("minute")
75:
76:     val isActive = bool("is_active").default(true)
77:     val createdAt = datetime("created_at").default(LocalDateTime.now())
78:     val lastExecuted = datetime("last_executed").nullable()
79:
80:     override val primaryKey = PrimaryKey(scheduleId)
81: }
82: //ТАБЛИЦЯ ДОСТУПУ КОРИСТУВАЧІВ ДО ПРИСТРОЇВ
83: object DeviceAccess : Table("device_access") {
84:     val accessId = integer("access_id").autoIncrement()
85:
86:     val userId = integer("user_id").references(Users.userId)
87:     val deviceId = integer("device_id").references(Devices.deviceId)
88:
89:     val permission = varchar("permission", 10).default("VIEW") //
NONE, VIEW, CONTROL
90:
91:     override val primaryKey = PrimaryKey(accessId)
92:
93:     init {
94:         uniqueIndex(userId, deviceId)
95:     }
96: }

```

Додаток Б

Лістинг програмного коду серверної частини

Файл Routing.kt

```
001: package com
002:
003: import com.services.AccessControl
004: import io.ktor.http.*
005: import io.ktor.server.application.*
006: import io.ktor.server.plugins.statuspages.*
007: import io.ktor.server.request.*
008: import io.ktor.server.response.*
009: import io.ktor.server.routing.*
010: import org.jetbrains.exposed.sql.*
011: import org.jetbrains.exposed.sql.transactions.transaction
012: import models.*
013: import com.services.MqttService
014: import java.time.LocalDateTime
015: import io.ktor.server.auth.*
016: import io.ktor.server.auth.jwt.*
017:
018: fun Application.configureRouting(mqttService: MqttService) {
019:     // Централізована обробка виключних ситуацій та критичних збоїв
020:     install(StatusPages) {
021:         exception<Throwable> { call, cause ->
022:             call.respondText(
023:                 text = "500 Internal Server Error: ${cause.message}",
024:                 status = HttpStatusCode.InternalServerError
025:             )
026:         }
027:     }
029:     routing {
030:         // Публічний маршрут: Автентифікація користувача та видача
токуена
031:         post("/api/auth/login") {
```

```
032:         val loginReq = call.receive<AuthRequest>()
033:         val userFromDb = transaction {
034:             Users.select { (Users.email eq loginReq.email) and
035:                 (Users.password eq loginReq.password) }
036:                 .map { Triple(it[Users.userId], it[Users.email],
037:                     it[Users.role]) }
038:                 .singleOrNull()
039:         }
040:         if (userFromDb != null) {
041:             val (id, email, role) = userFromDb
042:             val token = JwtConfig.generateToken(id, email, role)
043:             call.respond(AuthResponse(token = token, role = role,
044:                 userId = id))
045:         } else {
046:             call.respond(HttpStatusCode.Unauthorized, "Невірний
047:                 email або пароль")
048:         }
049:         // Блок захищених маршрутів (потребують валідації JWT)
050:         authenticate("auth-jwt") {
051:             get("/") {
052:                 call.respondText("HataPlus API is running!")
053:             }
054:             // Реєстрація гостя та призначення індивідуальних прав
055:             доступу до IoT-пристроїв
056:             post("/api/auth/register-guest") {
057:                 val principal = call.principal<JWTPrincipal>()
058:                 ?: return@post
059:                 call.respond(HttpStatusCode.Unauthorized, "Неавторизовано")
060:                 val userRole =
061:                 principal.payload.getClaim("role").asString()!!
062:                 // Рольовий фільтр: Тільки HOST (власник) має адмін-
063:                 права
064:                 if (userRole != "HOST") {
```

```
063:                return@post call.respond(HttpStatusCode.Forbidden,
"Тільки власник будинку може реєструвати гостей")
064:                }
065:
066:                val request = call.receive<RegisterGuestRequest>()
067:
068:                try {
069:                    val result = transaction {
070:                        val exists = Users.select { Users.email eq
request.email }.any()
071:                        if (exists) return@transaction null
073:                        val newGuestId = Users.insert {
074:                            it[email] = request.email
075:                            it[password] = request.password
076:                            it[phoneNum] = request.phoneNum
077:                            it[role] = "GUEST"
078:                        }[Users.userId]
079:
080:                        // Атомарне внесення прав доступу до таблиці
DeviceAccess
081:                        request.deviceAccesses.forEach { access ->
082:                            DeviceAccess.insert {
083:                                it[userId] = newGuestId
084:                                it[deviceId] = access.deviceId
085:                                it[permission] =
access.permission.uppercase()
086:                            }
087:                        }
088:                        newGuestId
089:                    }
091:                    if (result != null) {
092:                        call.respondText(text = "Гостя успішно
zareestrovano", status = HttpStatusCode.Created)
093:                    } else {
095:                    }
```

```
096:                } catch (e: Exception) {
097:                    call.respond(HttpStatusCode.InternalServerError,
"Помилка сервера: ${e.message}")
098:                }
099:            }
100:
101:                // Інтеграція з MQTT: Отримання списку активних
непідключених пристроїв у мережі
102:                get("/api/devices/discovered") {
103:                    val requestedType = call.request.queryParameters["type"]
104:                    val list = mqttService.getActiveDiscoveredDevices()
105:                        .filter { requestedType == null ||
it.type.equals(requestedType, ignoreCase = true) }
106:                        .map { it.macAddress }
107:                    call.respond(list)
108:                }
109:
110:                // Інтеграція з MQTT: Реєстрація нового пристрою в базі
даних та видалення з пулу тимчасових
111:                post("/api/devices") {
112:                    val principal = call.principal<JWTPrincipal>() ?:
return@post call.respond(HttpStatusCode.Unauthorized)
113:                    val userRole =
principal.payload.getClaim("role").asString()!!
115:                    if (userRole != "HOST") return@post
call.respond(HttpStatusCode.Forbidden, "Тільки господар може додавати
пристрої")
116:
117:                    val deviceData = call.receive<DeviceCreateDto>()
118:                    val existingId = transaction {
119:                        Devices.select { Devices.macAddress eq
deviceData.macAddress }
120:                        .map { it[Devices.deviceId] }.singleOrNull()
121:                    }
```

```
122:
123:             if (existingId != null) {
124:
mqttService.discoveredDevices.remove(deviceData.macAddress)
125:             return@post call.respond(HttpStatusCode.Conflict,
"Пристрій з таким MAC вже додано")
126:         }
127:
128:         val newId = transaction {
129:             Devices.insert {
130:                 it[roomId] = deviceData.roomId
131:                 it[name] = deviceData.name
132:                 it[type] = deviceData.type
133:                 it[macAddress] = deviceData.macAddress
134:                 it[status] = "Online"
135:                 it[lastSeen] = LocalDateTime.now()
136:             } get Devices.deviceId
137:         }
mqttService.discoveredDevices.remove(deviceData.macAddress)
140:             call.respond(HttpStatusCode.Created, mapOf("id" to
newId))
141:         }
143:         /* * ПРИМІТКА: Інші ендпоінти підсистеми API (CRUD операції
для кімнат,
144:             * керування телеметрією, видалення пристроїв, обробка
розкладів подій
145:             * та надсилання черги команд через REST Command API)
146:         }
147:     }
148: }
```

Повна версія файлу Routing.kt та загальний код проєкту доступні в репозиторії GitHub: <https://github.com/BohdanZakharchenko/HataPlusBackend>

Додаток В

Лістинг коду клієнтської частини

Файл LoginScreen.kt

```
001: package org.hataplus.project.ui.screens
002:
003: import androidx.compose.foundation.layout.*
004: import androidx.compose.material3.*
005: import androidx.compose.runtime.*
006: import androidx.compose.ui.Alignment
007: import androidx.compose.ui.Modifier
008: import androidx.compose.ui.text.input.PasswordVisualTransformation
009: import androidx.compose.ui.unit.dp
010: import kotlinx.coroutines.launch
011: import org.hataplus.project.data.SessionManager // ← Додано
012: import org.hataplus.project.ui.screens.components.HataTextField
013:
014: @Composable
015: fun LoginScreen(
016:     onLoginSuccess: () -> Unit,
017:     onLoginFailed: (String) -> Unit,
018:     onLoginClick: suspend (String, String, String) -> Boolean
019: ) {
020:     var email by remember { mutableStateOf("") }
021:     var password by remember { mutableStateOf("") }
022:     var serverUrl by remember { mutableStateOf(SessionManager.baseUrl
?: "api.hataplus.org") }
023:     var isLoading by remember { mutableStateOf(false) }
024:     val scope = rememberCoroutineScope()
025:
026:     Box(
027:         modifier = Modifier.fillMaxSize(),
028:         contentAlignment = Alignment.Center
029:     ) {
030:         Column(
```

```
031:         modifier = Modifier
032:             .fillMaxWidth()
033:             .padding(24.dp),
034:         horizontalAlignment = Alignment.CenterHorizontally
035:     ) {
036:         Text(
037:             text = "Вхід у HataPlus+",
038:             style = MaterialTheme.typography.headlineLarge
039:         )
040:         Spacer(modifier = Modifier.height(32.dp))
041:
042:         // Поле Email
043:         HataTextField(
044:             value = email,
045:             onChange = { email = it },
046:             label = "Email"
047:         )
048:         Spacer(modifier = Modifier.height(16.dp))
049:
050:         // Поле Пароль
051:         OutlinedTextField(
052:             value = password,
053:             onChange = { password = it },
054:             label = { Text("Пароль") },
055:             visualTransformation = PasswordVisualTransformation(),
056:             modifier = Modifier.fillMaxWidth(),
057:             singleLine = true
058:         )
059:         Spacer(modifier = Modifier.height(16.dp))
060:
061:         // Адреса сервера
062:         OutlinedTextField(
063:             value = serverUrl,
064:             onChange = { serverUrl = it },
065:             label = { Text("Адреса сервера") },
066:             placeholder = { Text("api.hataplus.org") },
```

```
067:             modifier = Modifier.fillMaxWidth(),
068:             singleLine = true
069:         )
070:
071:         Spacer(modifier = Modifier.height(32.dp))
072:
073:         if (isLoading) {
074:             CircularProgressIndicator()
075:         } else {
076:             Button(
077:                 onClick = {
078:                     if (email.isNotBlank() && password.isNotBlank()
079: && serverUrl.isNotBlank()) {
080:                         isLoading = true
081:                         scope.launch {
082:                             try {
083:                                 val success = onLoginClick(email,
084: password, serverUrl)
085:                                 isLoading = false
086:                                 if (success) {
087:                                     SessionManager.baseUrl =
088: serverUrl
089:                                     onLoginSuccess()
090:                                 } else {
091:                                     onLoginFailed("Невірний логін
092: або пароль")
093:                                 }
094:                             } catch (e: Exception) {
095:                                 isLoading = false
096:                                 onLoginFailed("Помилка з'єднання
з сервером: ${e.message}")
097:                             }
098:                         }
099:                     } else {
100:                         onLoginFailed("Будь ласка, заповніть усі
поля")
101:                     }
102:                 }
103:             )
104:         }
105:     }
106: }
```

```
097:                }
098:                },
099:                modifier = Modifier.fillMaxWidth().height(50.dp)
100:            ) {
101:                Text("Увійти")
102:            }
103:        }
104:    }
105: }
106: }
```

Повний лістинг клієнтської частини проєкту доступний на репозиторії
GitHub: <https://github.com/BohdanZakharchenko/HataPlusApp>

Додаток Г

Лістинг коду прошивок IoT-пристроїв на базі мікроконтролера ESP32

Файл TempSensor.ino

```
001: #include <Wire.h>
002: #include <Adafruit_GFX.h>
003: #include <Adafruit_SSD1306.h>
004: #include <DHT.h>
005: #include <WiFi.h>
006: #include <PubSubClient.h>
007: #include <ArduinoJson.h>
008:
009: // === Налаштування пристрою для HataPlus ===
010: const char* device_mac_id = "ESP32-HATA-006";
011: const char* device_type   = "TEMPERATURE";
012:
013: // === OLED ===
014: #define SCREEN_WIDTH 128
015: #define SCREEN_HEIGHT 64
016: #define OLED_RESET    -1
017:   Adafruit_SSD1306   display(SCREEN_WIDTH,   SCREEN_HEIGHT,   &Wire,
OLED_RESET);
018:
019: // === DHT22 ===
020: #define DHTPIN 2
021: #define DHTTYPE DHT22
022: DHT dht(DHTPIN, DHTTYPE);
023:
024: // === WiFi ===
025: const char* ssid      = "Wokwi-GUEST";
026: const char* password = "";
027:
028: // === MQTT ===
029: const char* mqtt_server = "172.17.134.97";
030: const int   mqtt_port   = 1883;
```

```
031: const char* mqtt_topic_telemetry = "home/sensors/data";
032: const char* mqtt_topic_discovery = "home/sensors/discovery";
033:
034: // НОВИЙ ТОПІК ДЛЯ СТАТУСУ
035: const char* mqtt_topic_status = "home/sensors/status/esp32-hata-006";
036:
037: WiFiClient espClient;
038: PubSubClient client(espClient);
039:
040: void setupWiFi() {
041:   Serial.print("Connecting to WiFi...");
042:   WiFi.begin(ssid, password);
043:   while (WiFi.status() != WL_CONNECTED) {
044:     delay(500);
045:     Serial.print(".");
046:   }
047:   Serial.println("\nConnected! IP: " + WiFi.localIP().toString());
048: }
049:
050: void reconnectMQTT() {
051:   while (!client.connected()) {
052:     Serial.print("Attempting MQTT connection...");
053:
054:
055:     if (client.connect(device_mac_id, NULL, NULL, mqtt_topic_status,
0, true, "offline")) {
056:       Serial.println("connected!");
057:
058:       // Відправляємо статус ONLINE відразу після підключення
059:       client.publish(mqtt_topic_status, "online", true);
060:
061:       // Повідомляємо систему про пристрій (Discovery)
062:       StaticJsonDocument<100> info;
063:       info["macAddress"] = device_mac_id;
064:       info["type"] = device_type;
065:       char buffer[100];
```

```
066:     serializeJson(info, buffer);
067:     client.publish(mqtt_topic_discovery, buffer);
068:
069:   } else {
070:     Serial.print("failed, rc=");
071:     Serial.print(client.state());
072:     delay(5000);
073:   }
074: }
075: }
076:
077: void sendToMQTT(float temp, float hum) {
078:   StaticJsonDocument<200> doc;
079:   doc["macAddress"] = device_mac_id;
080:   doc["temperature"] = temp;
081:   doc["humidity"] = hum;
082:
083:   char buffer[200];
084:   serializeJson(doc, buffer);
085:
086:   if (client.publish(mqtt_topic_telemetry, buffer)) {
087:     Serial.println("Sent: " + String(buffer));
088:   }
089: }
090:
091: void setup() {
092:   Serial.begin(115200);
093:   dht.begin();
094:
095:   if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
096:     for(;;);
097:   }
098:
099:   display.clearDisplay();
100:   display.setTextColor(WHITE);
101:   display.display();
```

```
103:   setupWiFi();
104:   client.setServer(mqtt_server, mqtt_port);
105: }
106:
107: void loop() {
108:   if (!client.connected()) {
109:     reconnectMQTT();
110:   }
111:   client.loop();
112:
113:   float h = dht.readHumidity();
114:   float t = dht.readTemperature();
115:
116:   display.clearDisplay();
117:   display.setCursor(0, 0);
118:
119:   if (isnan(h) || isnan(t)) {
120:     display.println("Sensor Error");
121:   } else {
122:     display.setTextSize(1);
123:     display.println("HataPlus: " + String(device_mac_id));
124:     display.setTextSize(2);
125:     display.setCursor(0, 20);
126:     display.print(t, 1); display.print("C");
127:     display.setCursor(0, 45);
128:     display.print(h, 0); display.print("% Hum");
129:
130:     static unsigned long lastSend = 0;
131:     if (millis() - lastSend > 10000) {
132:       lastSend = millis();
133:       sendToMQTT(t, h);
134:     }
135:   }
136:   display.display();
137:   delay(1000);
138: }
```

Файл SmartSocket.ino

```
001: #include <Wire.h>
002: #include <Adafruit_GFX.h>
003: #include <Adafruit_SSD1306.h>
004: #include <WiFi.h>
005: #include <PubSubClient.h>
006: #include <ArduinoJson.h>
007:
008: // === Налаштування пристрою для HataPlus ===
009: const char* device_mac_id = "ESP32-SOCKET-001";
010: const char* device_type   = "SMART_PLUG_BASIC";
012: // === Конфігурація Пінів ===
013: #define RELAY_PIN    23
014: #define LED_IND_PIN  22
015:
016: // === OLED дисплей ===
017: #define SCREEN_WIDTH 128
018: #define SCREEN_HEIGHT 64
019: #define OLED_RESET    -1
020:   Adafruit_SSD1306   display(SCREEN_WIDTH,   SCREEN_HEIGHT,   &Wire,
OLED_RESET);
021:
022: // === WiFi ===
023: const char* ssid      = "Wokwi-GUEST";
024: const char* password = "";
025:
026: // === MQTT Топіки ===
027: const char* mqtt_server = "172.17.134.97";
028: const int   mqtt_port   = 1883;
029:
030: const char* mqtt_topic_telemetry = "home/sensors/data";
031: const char* mqtt_topic_discovery = "home/sensors/discovery";
032: const char* mqtt_topic_status    = "home/sensors/status/esp32-socket-
001";
```

```
033: const char* mqtt_topic_command      = "home/sockets/commands/esp32-  
socket-001";  
034:  
035: // === Стан пристрою ===  
036: bool relay_state = false;  
037:  
038: WiFiClient espClient;  
039: PubSubClient client(espClient);  
040:  
041: // === Функція обробки команд від сервера (MQTT Callback) ===  
042: void callback(char* topic, byte* payload, unsigned int length) {  
043:     Serial.print("Message arrived [");  
044:     Serial.print(topic);  
045:     Serial.print("]: ");  
046:  
047:     String message = "";  
048:     for (unsigned int i = 0; i < length; i++) {  
049:         message += (char)payload[i];  
050:     }  
051:     Serial.println(message);  
052:  
053:     // Парсимо JSON  
054:     StaticJsonDocument<200> doc;  
055:     DeserializationError error = deserializeJson(doc, message);  
056:  
057:     if (!error) {  
058:         if (doc.containsKey("status")) {  
059:             String status = doc["status"];  
060:             if (status == "ON") {  
061:                 relay_state = true;  
062:                 digitalWrite(RELAY_PIN, HIGH);  
063:                 digitalWrite(LED_IND_PIN, HIGH);  
064:                 Serial.println("Relay and LED turned ON");  
065:             } else if (status == "OFF") {  
066:                 relay_state = false;  
067:                 digitalWrite(RELAY_PIN, LOW);
```

```
068:         digitalWrite(LED_IND_PIN, LOW);
069:         Serial.println("Relay and LED turned OFF");
070:     }
071: }
072: } else {
073:     Serial.print("deserializeJson() failed: ");
074:     Serial.println(error.c_str());
075: }
076: }
077:
078: void setupWiFi() {
079:     Serial.print("Connecting to WiFi...");
080:     WiFi.begin(ssid, password);
081:     while (WiFi.status() != WL_CONNECTED) {
082:         delay(500);
083:         Serial.print(".");
084:     }
085:     Serial.println("\nConnected! IP: " + WiFi.localIP().toString());
086: }
087:
088: void reconnectMQTT() {
089:     while (!client.connected()) {
090:         Serial.print("Attempting MQTT connection...");
091:
092:         // Підключення з LWT статусом "offline"
093:         if (client.connect(device_mac_id, NULL, NULL, mqtt_topic_status,
0, true, "offline")) {
094:             Serial.println("connected!");
095:
096:             client.publish(mqtt_topic_status, "online", true);
097:
098:             // РЕЄСТРАЦІЯ НА ТОПІК КОМАНД
099:             client.subscribe(mqtt_topic_command);
100:             Serial.println("Subscribed to command topic");
101:
102:             // Discovery повідомлення
```

```

103:     StaticJsonDocument<100> info;
104:     info["macAddress"] = device_mac_id;
105:     info["type"] = device_type;
106:     char buffer[100];
107:     serializeJson(info, buffer);
108:     client.publish(mqtt_topic_discovery, buffer);
109:
110:   } else {
111:     Serial.print("failed, rc=");
112:     Serial.print(client.state());
113:     delay(5000);
114:   }
115: }
116: }
117:
118: // === Надсилання телеметрії розетки (тільки стан) ===
119: void sendToMQTT(bool state) {
120:   StaticJsonDocument<200> doc;
121:   doc["macAddress"] = device_mac_id;
122:   doc["state"] = state ? "ON" : "OFF";
123:
124:   char buffer[200];
125:   serializeJson(doc, buffer);
126:
127:   if (client.publish(mqtt_topic_telemetry, buffer)) {
128:     Serial.println("Sent telemetry state: " + String(buffer));
129:   }
130: }
131:
132: void setup() {
133:   Serial.begin(115200);
134:
135:   // Конфігурація заліза
136:   pinMode(RELAY_PIN, OUTPUT);
137:   pinMode(LED_IND_PIN, OUTPUT);
138:   digitalWrite(RELAY_PIN, LOW);

```

```
139:   digitalWrite(LED_IND_PIN, LOW);
140:   // Налаштування дисплея
141:   if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
142:       for(;;);
143:   }
144:   display.clearDisplay();
145:   display.setTextColor(WHITE);
146:   display.display();
147:
148:   setupWiFi();
149:   client.setServer(mqtt_server, mqtt_port);
150:   client.setCallback(callback);
151: }
152:
153: void loop() {
154:   if (!client.connected()) {
155:       reconnectMQTT();
156:   }
157:   client.loop();
158:
159:   // Відображення на OLED
160:   display.clearDisplay();
161:   display.setTextSize(1);
162:   display.setCursor(0, 0);
163:   display.println("HataPlus Socket BASIC");
164:   display.println(device_mac_id);
165:
166:   display.setCursor(0, 20);
167:   display.setTextSize(1);
168:   display.print("Socket State: ");
169:
170:   display.setTextSize(2);
171:   display.setCursor(0, 35);
172:   display.println(relay_state ? "ENABLED" : "DISABLED");
173:
174:   display.setTextSize(1);
```

```
175:   display.setCursor(0, 55);
176:   display.print("Indicator LED: ");
177:   display.println(relay_state ? "ON" : "OFF");
178:   display.display();
179:
180:   // Відправка підтвердження стану кожні 5 секунд
181:   static unsigned long lastSend = 0;
182:   if (millis() - lastSend > 5000) {
183:     lastSend = millis();
184:     sendToMQTT(relay_state);
185:   }
186:
187:   delay(200);
188: }
```

БІБЛІОГРАФІЧНА ДОВІДКА

Тема бакалаврської роботи: Проєктування IoT-платформи для “розумного будинку”

Обсяг пояснювальної записки: 77 аркуш

Дата закінчення бакалаврської роботи 9 червня 2026р.

Підпис студента _____