Міністерство освіти і науки України
Криворізький національний університет
Факультет інформаційних технологій
Кафедра автоматизації, комп'ютерних наук і технологій

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти - магістр
за освітньо-професійною програмою
«Кіберфізичні системи в промисловості, бізнесі та транспорті»

зі спеціальності
174 – Автоматизація, комп'ютерно – інтегровані технології та робототехніка

тема роботи:
«*Розробка web-інтерфейсу керуванням розумними будинком на базі технології Node-Red*»

Виконав студент гр. АКІТР-24-2м _____ Філоненко В. М.

Керівник _____ Маринич І. А.

Нормоконтроль _____ Маринич І. А.

Завідувача кафедри _____ Рубан С. А.

Кривий Ріг – 2025

КРИВОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет: *інформаційних технологій*

Кафедра: *автоматизації, комп'ютерних наук і технологій*

Ступінь вищої освіти: *Магістр*

Спеціальність: *174 – Автоматизація, комп'ютерно-інтегровані технології та робототехніка*

ЗАТВЕРДЖУЮ

Зав. кафедри: *к.т.н. Рубан С.А.*

_____

« 15 » липня 2025 р.

ЗАВДАННЯ

на кваліфікаційну роботу магістра

студентові групи АКІТР-24-2м. *Філоненко Владиславу Миколайовичу*

1. Тема кваліфікаційної роботи: *«Розробка web-інтерфейсу керуванням розумними будинком на базі технології Node-Red»»*

затверджено наказом по університету № *595с* від *28.06.2024 р.*

2. Термін здачі кваліфікаційної роботи: *01.12.2025 р.*

3. Склад кваліфікаційної роботи: *Пояснювальна записка обсягом 111с., додатки, презентація у Microsoft PowerPoint (13 слайдів) в електронному та друкованому вигляді*

4. Консультанти кваліфікаційної роботи:

Розділ 1-3 доц. *Маринич І. А.*

Нормоконтроль доц. Маринич І. А.

5. Календарний план:

№	Етапи роботи	Терміни виконання
1	*Вступ*	*10.02.25*
2	*Розділ 1*	*15.04.25*
3	*Розділ 2*	*18.07.25*
4	*Розділ 3*	*19.11.25*
5	*Висновки*	*20.11.25*
6	*Оформлення кваліфікаційної роботи*	*25.11.25*
7	*Підготовка презентації та графічного матеріалу*	*28.11.25*
8	*Підготовка доповіді до захисту*	*01.12.25*

6. Дата видачі завдання: *24.12.2024р.*

Керівник_____/ Маринич І. А/

7. Запевнення: *Я, Філоненко Владислав Миколайович, запевняю, що ця кваліфікаційна робота виконана самостійно, не містить академічного плагіату, фабрикації, фальсифікації. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.*

Із чинним Положенням про академічну доброчесність Криворізького національного університету ознайомлений.

Чітко усвідомлюю, що в разі виявлення у кваліфікаційній роботі умисних порушень робота не допускається до захисту або оцінюється незадовільно.

Здобувач _____/ Філоненко В. М./

АНОТАЦІЯ

Філоненко В. М. Розробка web-інтерфейсу керуванням розумними будинком на базі технології Node-Red : кваліфікаційна робота магістра : 174 – Автоматизація, комп’ютерно – інтегровані технології та робототехніка. Кривий Ріг. Криворізький національний університет, 2025.

Мета роботи – розробка функціональної системи керування розумним будинком на базі Raspberry Pi з веб-інтерфейсом для моніторингу та контролю пристроїв у реальному часі через протокол MQTT.

Об'єкт проектування – система керування розумним будинком на базі Raspberry Pi з веб-інтерфейсом для автоматизації освітлення, вентиляції, моторизованих вікон та системи безпеки з використанням датчиків руху, освітленості та RFID-авторизації.

У першому розділі проведено аналіз технологій «розумного будинку», протоколів передачі даних та порівняння існуючих систем автоматизації.

У другому розділі описано побудову макетної схеми системи з підключенням датчиків до GPIO-пінів Raspberry Pi та реалізацію програмного забезпечення на Python з багатопоточною архітектурою.

У третьому розділі описано розробку веб-інтерфейсу для керування системою. Реалізовано MQTT-клієнт з автоматичним відновленням підписок, розроблено компоненти перемикачів стану пристроїв та модуль конфігурації автоматичних сценаріїв на основі часових розкладів та показань датчиків.

Результатом роботи є функціональна система розумного будинку з віддаленим керуванням через веб-браузер, автоматизацією пристроїв та системою безпеки з RFID-авторизацією.

Ключові слова: NODE-RED, РОЗУМНИЙ БУДИНОК, WEB-ІНТЕРФЕЙС, MQTT, RASPBERRY PI 3, NODE.JS, АВТОМАТИЗАЦІЯ, ДАТЧИКИ, МОНІТОРИНГ, СИСТЕМА БЕЗПЕКИ.

ANNOTATION

Filonenko V. M. Automation of the Development of a web interface for smart home control based on Node-Red technology : 174 – Automation, computer-integrated technologies, and robotics. Kryvyi Rih. Kryvyi Rih National University, 2025.

The aim of the work is to develop a functional smart home control system based on Raspberry Pi with a web interface for real-time monitoring and control of devices via MQTT protocol.

The object of design is a smart home control system based on Raspberry Pi with a web interface for automation of lighting, ventilation, motorized windows and security system using motion sensors, light sensors and RFID authorization.

The first section analyzes smart home technologies, data transmission protocols and compares existing automation systems. The classification of existing smart home systems is carried out.

The second section describes the construction of the system layout scheme with sensors connection to Raspberry Pi GPIO pins and implementation of Python software with multithreaded architecture. An object-oriented approach through the SmartHomeRPI class is implemented for hardware control and MQTT communication.

The third section describes the development of a web interface for system control. An MQTT client with automatic subscription recovery is implemented using the MQTT.js library for WebSocket connections. Device state switch components and automatic scenarios configuration module based on time schedules and sensor readings are developed.

The result of the work is a functional smart home system with remote control via web browser, device automation and security system with RFID authorization.

Keywords: NODE-RED, SMART HOME, WEB INTERFACE, MQTT, RASPBERRY PI 3, NODE.JS, AUTOMATION, SENSORS, MONITORING, SECURITY SYSTEM.

ЗМІСТ

ВСТУП

Сьогодні автоматизація відіграє важливу роль не лише у промислових галузях, таких як машинобудування, металообробка та харчова промисловість, але й у повсякденному житті, зокрема в облаштуванні побуту. Розвиток технологій дозволяє автоматизувати рутинні дії в будинках і квартирах, забезпечуючи дистанційне керування освітленням, температурою, системою безпеки та іншими аспектами через інтернет. Ці можливості реалізуються завдяки впровадженню систем «Розумний будинок».

Система «Розумний будинок» — це інтегрована система високотехнологічних пристроїв, яка забезпечує автоматизацію побутових процесів, підвищуючи комфорт, безпеку та енергоефективність. Вона здатна не лише реагувати на різноманітні події, як-от зміна температури або рух у приміщенні, а й дозволяє налаштовувати сценарії дій для автоматичного управління будинком. Наприклад, система може автоматично налаштовувати освітлення і кліматичні умови відповідно до заданих параметрів.

Ключовою особливістю сучасних систем «Розумний будинок» є можливість дистанційного керування через веб-інтерфейс або мобільні додатки. Це дозволяє власнику будинку або квартири віддалено контролювати роботу всіх підключених пристроїв, перевіряти стан приміщень та втручатися у роботу системи за необхідності. Однак впровадження подібних систем часто пов’язане з високими витратами на готові рішення, а також зі складністю їх інтеграції та налаштування.

Метою цієї роботи є розробка універсального та гнучкого рішення для керування системою «Розумний будинок» за допомогою технології Node-Red. Node-Red — це потужний інструмент для візуального програмування потоків даних, який дозволяє легко інтегрувати різні пристрої та сервіси в єдину систему. Використовуючи Node-Red, можна створити веб-інтерфейс для керування

будинком, що дозволяє не тільки моніторити стан сенсорів у реальному часі, але й задавати сценарії автоматизації.

У рамках цієї роботи було розроблено web-інтерфейс для керування «Розумним будинком» на базі Node-Red з використанням платформи Raspberry Pi 3. Raspberry Pi 3 виступає як центральний контролер, який обробляє сигнали від різноманітних сенсорів (температури, освітлення, руху, безпеки) і передає їх на сервер для подальшої обробки. Через веб-інтерфейс користувач може дистанційно керувати освітленням, опаленням, сигналізацією та іншими пристроями. Також було передбачено інтеграцію з хмарними сервісами для збору та аналізу даних, що дозволяє покращити ефективність управління будинком.
Node-Red дозволяє реалізувати підтримку різних протоколів, таких як MQTT та HTTP, що робить систему гнучкою та розширюваною. Усі дані, які надходять від сенсорів, обробляються на сервері, а потім відображаються у зручному web-інтерфейсі. Завдяки можливості налаштування сценаріїв автоматизації, користувач може створювати правила для керування пристроями, наприклад, вмикати освітлення при вході в кімнату або регулювати температуру заздалегідь заданим графіком.

Таким чином, розроблений веб-інтерфейс на базі Node-Red забезпечує зручне та ефективне керування системою «Розумний будинок», надаючи можливість інтегрувати різноманітні пристрої, збирати дані з сенсорів та автоматизувати рутинні дії. Цей підхід забезпечує економічність та гнучкість системи, знижуючи витрати на впровадження та надаючи користувачам можливість налаштовувати систему відповідно до їхніх потреб.

РОЗДІЛ 1 АНАЛІЗ ОСНОВНИХ ПОНЯТЬ ТЕХНОЛОГІЙ «РОЗУМНОГО БУДИНКУ», ТА СУЧАСНІ ПЛАТФОРМИ ДЛЯ АВТОМАТИЗАЦІЇ

1.1 Характеристика поняття “Розумний будинок”

Сучасні технічні новації мають на меті спростити життя та створити комфортні умови для проживання. Інноваційні технології сприяють формуванню теплих, світлих і енергоефективних житлових просторів. Проте збільшення кількості автоматизованих систем ускладнює їх управління, навіть для досвідчених користувачів. У цьому контексті система «Розумний будинок» стає важливим рішенням, яке інтегрує всі технічні підсистеми для зручного їх контролю.

Рисунок 1.1- Концептуальний малюнок системи “Розумний будинок”

Система «Розумний будинок» охоплює кілька ключових аспектів, зокрема управління електроприводами, кліматичний контроль та безпеку[1]. Вона має можливість виявляти та реагувати на різні ситуації в будинку. Ключова особливість цієї системи полягає в об'єднанні окремих підсистем в єдиний керований комплекс, що дозволяє одному елементу впливати на інші відповідно

до заданих алгоритмів. Це новаторська концепція, яка забезпечує користувачеві зручний доступ до свого житлового простору.

Завдяки системі «Розумний будинок» користувач може задавати необхідні параметри без використання численних перемикачів. Автоматика, враховуючи зовнішні та внутрішні умови, а також переваги власника, контролює роботу всіх систем і пристроїв. У результаті немає потреби самостійно налаштовувати окремі вимикачі для освітлення, управління вентиляцією, опаленням чи безпекою.

Система дозволяє користуватися єдиним веб-додатком або спеціальною програмою на комп'ютері чи смартфоні для контролю всіх пристроїв. Будинок реагує на команди користувача, враховуючи час доби, наявність людей у кімнаті та освітленість, щоб забезпечити оптимальні умови проживання. Таким чином, «Розумний будинок» не лише спрощує управління системами, а й робить повсякденне життя більш зручним і комфортним.

1.2 Характеристика протоколів передачі даних

У сучасних системах "Розумний будинок" одним із ключових аспектів, що впливають на їх ефективність і надійність, є протоколи передачі даних. Вибір протоколу визначає, як пристрої взаємодіють один з одним, як здійснюється передача інформації між компонентами системи та наскільки стабільно і швидко відбувається це спілкування. Різноманіття протоколів, таких як 1-Wire, ZigBee, X10, Z-Wave, Wi-Fi, Bluetooth, Thread, LoRa, MQTT та інші, дозволяє розробникам адаптувати свої рішення під специфічні вимоги користувачів.

Ці протоколи забезпечують різні рівні інтеграції та функціональності для систем "Розумний будинок", дозволяючи користувачам вибирати рішення, які найкраще відповідають їхнім потребам. Розуміння можливостей кожного з протоколів допоможе спростити налаштування та управління системою, покращуючи комфорт і безпеку в житловому просторі.

Протокол 1-Wire використовує двонаправлену інформаційну шину для передачі даних між компонентами системи. Управління відбувається через комп'ютер або мікроконтролер, з підключенням до комп'ютера за допомогою спеціального адаптера.

Переваги: низька вартість, що робить систему доступною для багатьох користувачів.

Недоліки: обмежена відмовостійкість через низьку якість дроту, що може викликати проблеми в роботі системи.

Таким чином, 1-Wire підходить для простих автоматизаційних рішень, але його вразливість до проблем з якістю компонентів може обмежити використання в складніших системах.

Протокол X10, розроблений у 1975 році, залишається популярним у сфері автоматизації "розумних будинків".[2] У цій системі електропроводка використовується як магістраль для передачі даних, що дозволяє з'єднувати пристрої через існуючу електричну мережу. Управління бездротовими пристроями відбувається через спеціальні перетворювачі, які передають сигнали через електромережу.

X10 підтримує різноманітні модулі, що дозволяє автоматизувати освітлення, системи безпеки та полив рослин. Однак для належної роботи потрібні спеціалізовані мікроконтролери.

Переваги: низька вартість і універсальність, завдяки можливості інтеграції з різними пристроями. Недоліки: низька швидкість передачі даних (понад 1 секунду) і складність монтажу, що може вимагати додаткових знань.

Таким чином, X10 є ефективним для базової автоматизації, але обмеження у швидкості та монтажі можуть бути проблемою для складніших систем.

Wi-Fi (Wireless Fidelity) — це популярний метод бездротової передачі даних, широко використовуваний у системах "розумного будинку". Він дозволяє

створювати бездротові мережі для автоматизації та контролю пристроїв у житлових приміщеннях.

Основні переваги Wi-Fi включають високу швидкість передачі даних і надійні методи шифрування, такі як WPA2 і WPA3. Однак його ефективність може бути обмежена якістю сигналу та високою щільністю пристроїв у мережі, що може призвести до зниження швидкості.

Отже, Wi-Fi є потужним протоколом для "розумного будинку", але його продуктивність залежить від умов роботи мережі.

Bluetooth — це стандарт бездротових персональних мереж, що дозволяє пристроям обмінюватися даними на частоті близько 2,4 ГГц. Він використовує алгоритм FHSS для зменшення ймовірності перешкод. Bluetooth активно застосовується в системах "розумного будинку" для з'єднання різних компонентів, а також серед розробників самостійних рішень.

Серед переваг Bluetooth — низька вартість, простота налаштування і енергозбереження, що важливо для акумуляторних пристроїв. Проте, його недоліками є обмежений радіус дії (до 10-30 метрів), низька швидкість передачі даних у порівнянні з Wi-Fi, а також вразливість до перешкод від інших бездротових пристроїв.

Отже, протокол Bluetooth пропонує зручний та економічний спосіб з'єднання компонентів систем "розумного будинку", але має свої обмеження щодо радіусу дії та швидкості передачі даних. Ці фактори слід враховувати при проектуванні автоматизованих систем для дому.

MQTT (Message Queuing Telemetry Transport) — легкий протокол обміну повідомленнями, розроблений для передачі даних між пристроями з обмеженими ресурсами, такими як датчики та IoT-пристрої. Він забезпечує комунікацію в режимі «публікації-підписки», що дозволяє пристроям обмінюватися інформацією без прямого з'єднання, працюючи через TCP/IP.

Цей протокол широко використовується в системах "розумного будинку", де різні компоненти можуть взаємодіяти через центральний брокер, що керує доставкою повідомлень. MQTT спроектований для роботи в умовах обмежених ресурсів, що робить його ідеальним для енергоощадних пристроїв. Він забезпечує швидку і надійну доставку повідомлень та дозволяє легко масштабувати системи за рахунок гнучкої моделі підписки. Протокол також підтримує різні рівні якості служби (QoS), що дозволяє обирати баланс між швидкістю і надійністю передачі даних.

Проте MQTT має деякі недоліки. Всі повідомлення проходять через центральний брокер, що може стати вузьким місцем при високому навантаженні. Для роботи протоколу необхідне стабільне з'єднання з Інтернетом, що може бути проблематичним у випадках обмеженого зв'язку. Крім того, хоча протокол має механізми безпеки, такі як SSL/TLS, він потребує додаткових зусиль для налаштування безпечного з'єднання.

У загальному підсумку, MQTT є потужним протоколом для створення систем "розумного будинку", забезпечуючи ефективну і надійну комунікацію між пристроями, хоча важливо враховувати його залежність від брокера і необхідність налаштування безпеки.

Протокол ZigBee — це технологія бездротової комунікації, призначена для пристроїв з низьким енергоспоживанням, що ідеально підходить для систем "розумного будинку". Його децентралізована архітектура забезпечує високу відмовостійкість і можливість інтеграції великої кількості пристроїв у мережу.

Серед переваг ZigBee — низьке енергоспоживання, що дозволяє пристроям працювати на батарейках тривалий час. Технологія також забезпечує швидку передачу даних, що важливо для автоматизації в реальному часі. ZigBee може підтримувати до 65 000 пристроїв у мережі, що робить його ідеальним для великих будівель. Крім того, протокол використовує гнучкі топології (зірка, дерево, сітка) для проектування мережі.

Проте ZigBee має деякі недоліки. По-перше, існує проблема несумісності між пристроями від різних виробників, оскільки багато з них використовують власні реалізації протоколу. По-друге, пропускна здатність ZigBee обмежена в порівнянні з Wi-Fi, що може вплинути на швидкість передачі даних. Також його діапазон дії зазвичай обмежений до 100 метрів у прямій видимості, що може стати проблемою в великих або складних приміщеннях.[3]

У загальному підсумку, ZigBee є ефективним протоколом для "розумних" систем завдяки енергоефективності та можливості інтеграції багатьох пристроїв, але користувачам слід враховувати проблеми з несумісністю та обмеженнями діапазону.

Z-Wave — це бездротовий протокол зв'язку, створений для автоматизації будинків, що забезпечує функціонування пристроїв з низьким енергоспоживанням. Він акцентує увагу на стандартизації та сумісності пристроїв, що спрощує інтеграцію в системи "розумного будинку"[4].

Серед основних переваг Z-Wave — висока сумісність: усі пристрої, що підтримують цей протокол, можуть взаємодіяти незалежно від виробника, що спрощує додавання нових елементів. Він також простий в установці та налаштуванні, що робить його доступним для користувачів без спеціалізованих знань. Z-Wave оптимізований для роботи на батарейках, що забезпечує тривалий термін служби пристроїв. Протокол працює на частотах 868 МГц (в Європі) або 908 МГц (в США), що зменшує перешкоди від інших бездротових пристроїв.
Проте Z-Wave має деякі недоліки. По-перше, вартість пристроїв може бути вищою, ніж у аналогів, таких як ZigBee або Wi-Fi, що може обмежити доступність. По-друге, хоча Z-Wave підтримує до 232 пристроїв у мережі, це менше, ніж можливості ZigBee, що може стати обмеженням для великих систем. Загалом, Z-Wave є надійним вибором для тих, хто шукає стандартизований протокол для "розумного будинку" з високою сумісністю та простотою

установки. Однак потенційним користувачам варто зважити на вартість пристроїв при плануванні своїх рішень.

Thread — це бездротовий протокол, розроблений спеціально для IoT (Internet of Things) та систем "розумного будинку"[5]. Він забезпечує надійний та енергоефективний зв'язок між пристроями, спрощуючи їх інтеграцію у великі мережі. Thread оптимізований для роботи на батарейках, що робить його ідеальним для бездротових датчиків та пристроїв з низьким енергоспоживанням.

Протокол створює децентралізовану мережу, де всі пристрої можуть взаємодіяти один з одним без необхідності в центральному хабі, що підвищує відмовостійкість системи. Він також підтримує велику кількість пристроїв, що робить його ідеальним для великих установок "розумного будинку". Thread працює на базі стандарту IPv6, що дозволяє легко інтегрувати пристрої з Інтернетом та іншими мережами, що підтримують IP. Крім того, протокол забезпечує вбудовані механізми шифрування для захисту даних.

Однак, незважаючи на свої переваги, Thread має й недоліки. Кількість доступних пристроїв, що підтримують цей протокол, поки що обмежена в порівнянні з Wi-Fi або Z-Wave. Крім того, налаштування Thread-мережі може бути дещо складним для користувачів, які не мають технічного досвіду. Загалом, Thread є перспективним протоколом для створення сучасних систем "розумного будинку", але користувачам слід врахувати доступність пристроїв та можливі труднощі в налаштуванні.

LoRa (Long Range) — це бездротова технологія зв’язку, спеціально розроблена для IoT, яка забезпечує довготривалі з’єднання на великих відстанях при низькому енергоспоживанні. Ця технологія особливо популярна в застосуваннях, де важливі дальність передачі та енергоефективність. LoRa здатна передавати дані на відстань до 15 км у відкритому просторі, що робить її ідеальною для сільського господарства, розумних міст та промислових застосувань.[6]

Пристрої, що використовують LoRa, можуть працювати на батареях протягом кількох років, знижуючи витрати на обслуговування та заміну елементів живлення. Ця технологія також підтримує підключення великої кількості пристроїв в одній мережі, що корисно для розподілених систем IoT. Завдяки своїй технології модуляції, LoRa демонструє високу стійкість до перешкод, що забезпечує надійний зв'язок у складних умовах.

Однак, незважаючи на численні переваги, LoRa має й обмеження. Наприклад, вона має нижчу швидкість передачі даних у порівнянні з іншими протоколами, такими як Wi-Fi або Bluetooth, що робить її менш придатною для застосувань, які вимагають високої швидкості. Для використання LoRa також потрібні базові станції або шлюзи для прийому сигналу, що може призвести до додаткових витрат на інфраструктуру. Крім того, LoRa не підходить для застосувань, де потрібна миттєва передача даних, таких як потокове відео.

Загалом, LoRa є потужним рішенням для специфічних застосувань в сфері IoT, де критично важливими є дальність зв’язку та енергоефективність. Вона ідеально підходить для проектів, що передбачають великий обсяг даних з віддалених датчиків, таких як сільське господарство, охорона навколишнього середовища та управління ресурсами.

1.3 Аналіз існуючих систем “Розумний будинок”

Системи "Розумний будинок" стають все більш популярними завдяки їх здатності забезпечувати комфорт, енергоефективність і безпеку у житлових приміщеннях. Основною метою таких систем є інтеграція різних технічних підсистем в єдину керовану платформу, що дозволяє автоматизувати виконання рутинних завдань і полегшити життя мешканців.

1.3.1 Класифікація систем “Розумний будинок”

Система "Розумний будинок" може виконувати широкий спектр функцій, які забезпечують комфорт, безпеку та зручність управління побутом. До основних функцій відносяться:

- Керування електрикою й освітленням: контроль електроживлення і освітлення в приміщенні, можливість автоматичного ввімкнення/вимкнення світла в залежності від присутності людей.
- Безпека: контроль систем відеоспостереження, датчики руху, пожежна сигналізація, контроль доступу.
- Клімат-контроль: автоматичне регулювання температури, вентиляції та вологості повітря, що забезпечує комфортний мікроклімат у приміщенні.
- Керування побутовими приладами: можливість дистанційного або автоматичного управління побутовими приладами, такими як пральна машина, кондиціонер, опалювальні системи.
- Мультимедіа: управління аудіо- та відеосистемами, інтеграція домашнього кінотеатру, музичних пристроїв.

Для роботи системи "Розумний будинок" необхідно забезпечити надійну передачу даних між елементами системи. Залежно від технології, використовуваної для передачі даних, такі системи поділяються на три основні типи:

- Дротові системи: передача даних здійснюється через дротову інформаційну шину, яка з'єднує всі елементи системи. Для цього можуть використовуватися спеціалізовані кабелі. Такі системи вирізняються високою стабільністю і надійністю, але їхня установка може бути досить трудомісткою і дорогою через необхідність прокладання кабелів.
- Бездротові системи: передача даних відбувається через радіоканал або мережу Інтернет. Для цього використовуються технології Wi-Fi, Bluetooth

або ZigBee. Основною перевагою є легкість установки та гнучкість у розміщенні елементів системи, але такі системи можуть бути вразливішими до перешкод і менш стабільними порівняно з дротовими аналогами.

- Змішані системи: поєднують у собі елементи як дротової, так і бездротової передачі даних. Це дозволяє успадкувати переваги обох типів систем та забезпечити більшу гнучкість і надійність.

Залежно від архітектури та способу управління, системи "Розумний будинок" поділяються на централізовані та децентралізовані:

- Централізовані системи: керуються єдиним центральним контролером, який відповідає за координацію всіх елементів системи. Всі датчики та виконавчі механізми підключаються до цього контролера. Такі системи надають можливість точного контролю за всіма процесами, але їхнім недоліком може бути висока залежність від єдиного контролера, в разі виходу якого з ладу вся система може припинити функціонування.
- Децентралізовані системи: кожен елемент системи має власний контролер, що дозволяє функціонувати незалежно від інших компонентів. Це підвищує надійність системи, оскільки вихід з ладу одного з елементів не вплине на інші частини системи. Децентралізовані системи забезпечують більшу гнучкість і модульність, але можуть бути складнішими у налаштуванні та інтеграції.

Сучасні системи "Розумний будинок" мають низку додаткових характеристик, які підвищують їхню ефективність та забезпечують користувачам більше можливостей:

- Інтуїтивний інтерфейс: більшість систем пропонують прості у використанні інтерфейси через мобільні додатки або веб-портали, що дозволяє легко контролювати систему навіть на відстані.

- Інтеграція з IoT: багато систем підтримують інтеграцію з Інтернетом речей (IoT), що дозволяє взаємодіяти з різноманітними пристроями та системами, такими як "розумні" лампи, термостати, побутові прилади.
- Збір та аналіз даних: деякі системи можуть збирати дані про умови в будинку та поведінку користувачів, що дозволяє налаштовувати роботу системи більш точно під їхні потреби.
- Системи енергозбереження: розумні будинки можуть контролювати енергоспоживання, автоматично вимикати непотрібні прилади та підтримувати оптимальний температурний режим для зниження витрат на енергію.
- Безпека та захист даних: сучасні системи оснащені високим рівнем захисту даних, шифруванням інформації та протоколами безпеки для запобігання несанкціонованому доступу.

Таким чином, система "Розумний будинок" забезпечує не тільки підвищений рівень комфорту та безпеки, але й дозволяє ефективніше використовувати енергетичні ресурси та інтегрувати новітні технології для створення максимально зручного середовища для проживання.

1.3.2 Порівняння існуючих систем “Розумний будинок”

Система “Розумний будинок” набуває все більшої популярності завдяки своїй здатності значно покращити якість життя користувачів, забезпечуючи комфорт, безпеку та енергоефективність. Все більше людей зацікавлені в інтеграції цієї технології у свої домівки, що зумовлює зростаючий попит на ринку. Як наслідок, існує безліч пропозицій від різних виробників, кожен з яких пропонує свої унікальні рішення та функціональні можливості.

Однак важливо розуміти, що не існує єдиного підходу до реалізації технології "Розумний будинок". Кожен розробник використовує свої ідеї та інноваційні рішення, що веде до наявності як переваг, так і недоліків у кожній

системі. Вибір оптимальної системи залежить від конкретних потреб та побажань користувачів, а також від технічних можливостей самих пристроїв.

Далі ми розглянемо кілька передових рішень, доступних на українському ринку. Ці системи пропонують різноманітні функції та способи їх реалізації, а також варіанти інтеграції в існуючу інфраструктуру. Порівнюючи їхні характеристики, ми зможемо виявити сильні та слабкі сторони кожного рішення, що допоможе потенційним користувачам зробити усвідомлений вибір.

Рисунок 1.2 – “Розумний будинок” від Xiaomi

Система "Розумний будинок" від Xiaomi є бюджетним рішенням для автоматизації дому, забезпечуючи зручне управління побутовими приладами. Вона має автономні пристрої, можливість масштабування та вбудовану камеру для відеоспостереження. Управляти системою можна через мобільний додаток, а також створювати автоматизовані сценарії. Ціна базового комплекту становить 90 доларів.

Проте система має обмеження: зона дії до 10 метрів, обмежений вибір сенсорів у базовому наборі та відсутність резервного живлення для хаба. Загалом, комплект є хорошою стартовою платформою для інтеграції додаткових пристроїв і забезпечення безпеки в домі.[7]

Рисунок 1.3 – “Розумний будинок” від BroadLink

Обладнання "Розумний будинок" від BroadLink складається з цифрових пристроїв для управління побутовою технікою, освітленням та охоронними системами. Кожен компонент може працювати самостійно або у взаємодії з іншими.

Система легко встановлюється, має різноманітні датчики для вимірювання вологості, температури та освітленості, а також бездротову комунікацію та можливість управління через Wi-Fi з будь-якої точки світу. Ціна стартує від 200 доларів.

Недоліки включають обмежений діапазон дії (до 50 м), відсутність резервного живлення для хаба та обмеження пульта дистанційного керування. BroadLink пропонує розширений функціонал, простоту використання та можливість налаштування роботи пристроїв через мобільний додаток.[8]

Рисунок 1.4 – “Розумний будинок” від Fabiro

Система "Розумний будинок" від Fibaro є професійним обладнанням для автоматизації та безпеки житла, що потребує досвідчених фахівців для встановлення та налаштування.[9] Вона пропонує широкий асортимент датчиків, вбудовану відеокамеру, можливість налаштування багатьох сценаріїв та надсилання повідомлень на декілька телефонів. Використання протоколу Z-Wave забезпечує сумісність з іншими пристроями, а датчик протікання оснащений сиреною для аварійних ситуацій.

Проте система має ряд недоліків, зокрема високу вартість (від 600 доларів), необхідність професійного монтажу та налаштування, а також залежність від центрального контролера Fibaro Home Center, який повинен бути підключений до інтернету через LAN-кабель. Крім того, система не працює без хаба, не має резервного живлення, має обмежену дальність сигналу (до 50 м) та можливі затримки в Push-повідомленнях. Fibaro відрізняється розширеним асортиментом датчиків для контролю стану приміщень, але налаштування комплексу потребує професійного втручання.

Рисунок 1.5 – “Розумний будинок” від Ajax

Система "Розумний будинок" від Ajax забезпечує управління життєзабезпеченням та безпеку приміщення, контролюючи зломи й потенційні загрози. Вона використовує зашифрований радіозв'язок Jeweller, має стильний дизайн і автономне живлення.

Основні переваги: легкий монтаж, бездротовий зв'язок, зона дії до 1200 м, захист від зняття датчиків, підтримка Wi-Fi і GSM, а також можливість підключення до 100 пристроїв. Ціна комплекту починається від 200 доларів.

Недоліки: система працює лише з контролером (Hub) і не має вбудованої камери, управління можливе тільки через мобільний додаток. Ajax відзначається функціональністю та доступною ціною.[10]

Рисунок 1.6 – “Розумний будинок” від Orvibo

Система "Розумний будинок" Orvibo є доступним комплектом обладнання для забезпечення безпеки будинку, з можливістю розширення до повноцінної системи. Вона дозволяє віддалений контроль через мобільний додаток, автоматично підключаючи сенсори до центрального хабу. Orvibo підтримує до 100 датчиків, включаючи пристрої від сторонніх виробників.

Переваги: легка установка, бездротовий протокол ZigBee, власна відеокамера, часткова автономність пристроїв і доступна ціна (від 150 доларів).

Недоліки: обмежена зона дії сигналу (до 30 м), обмежений набір пристроїв у базовій комплектації, відсутність резервного живлення для хаба та дротове підключення до Інтернету.[11]

Висновки до 1-го розділу

У першому розділі було проведено комплексний аналіз технологій, що лежать в основі концепції "Розумний будинок", та розглянуто сучасні платформи для автоматизації. Поняття "Розумний будинок" охоплює інтелектуальні системи автоматизації, які спрямовані на забезпечення комфорту, енергоефективності та безпеки житла шляхом інтеграції різних технологій та пристроїв в єдину систему управління.

В межах аналізу протоколів передачі даних було досліджено ключові стандарти, що використовуються для комунікації в "Розумних будинках", включаючи ZigBee, Z-Wave, Wi-Fi, Bluetooth, LoRa, KNX, X10 та інші. Кожен із протоколів має свої переваги та недоліки, що визначають сферу їхнього застосування. Наприклад, ZigBee та Z-Wave забезпечують низьке енергоспоживання та надійність у великих мережах, тоді як Wi-Fi та Bluetooth більше підходять для високошвидкісного передавання даних на невеликі відстані. Особлива увага приділяється протоколам LoRa та MQTT, що

використовуються для передачі даних на великих відстанях та в умовах низької енергоефективності.

Також було проаналізовано існуючі системи "Розумний будинок", включаючи популярні платформи, такі як Fibaro, Ajax, Orvibo, а також різні їх функціональні можливості, особливості монтажу, вартість і сумісність із протоколами передачі даних. Зокрема, системи, які використовують протоколи Z-Wave та ZigBee, відзначаються сумісністю з великою кількістю пристроїв, тоді як більш бюджетні варіанти можуть мати обмежену функціональність та складності в налаштуванні.

Таким чином, розглянуті технології та платформи показують великий потенціал для розвитку ринку "Розумних будинків", де вибір системи або протоколу залежить від потреб користувача, масштабів проекту та бюджету.

РОЗДІЛ 2 ПРОГРАМНА РЕАЛІЗАЦІЯ КЕРУВАННЯ СИСТЕМИ “РОЗУМНИЙ БУДИНОК”

У другому розділі буде розглянута технічна реалізація системи "Розумний будинок" на базі одноплатного комп'ютера Raspberry Pi 3 B+, який є доступним, потужним і гнучким рішенням для створення власної системи автоматизації, що дозволяє контролювати різні пристрої та системи в межах домашнього середовища.

2.1 Аналіз об'єкту та планування систем автоматизації

У цьому об'єкті автоматизації розглядається макет одноповерхового будинку (рис. 2.1), що складається з чотирьох кімнат. У межах будинку передбачається впровадження та налаштування таких інженерних систем:

- освітлення;
- безпеки;
- живлення від альтернативних джерел енергії;
- система кондиціонування повітря;
- система автоматизованого керування жалюзі.

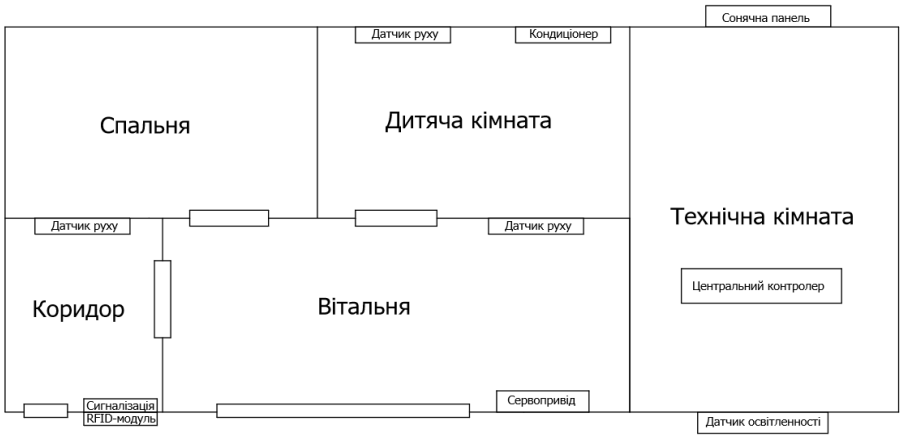

Рисунок 2.1 Схема розумного будинку

Система освітлення буде побудована на основі датчиків руху, які активуватимуть світлодіодні стрічки. Для системи безпеки планується використовувати датчики руху, звуковий модуль (динамік) та RFID-модуль для авторизації. Керування жалюзі забезпечуватиметься за допомогою серводвигуна та датчика освітленості, а система кондиціонування включатиме вентилятор, що працюватиме через реле.

Щоб підвищити ефективність та комфорт, передбачено два режими керування будинком – автоматичний та ручний, за допомогою web-додатку.

Цей підхід дозволяє провести комплексний аналіз об’єкта та ретельно спланувати інтеграцію систем автоматизації для забезпечення зручного управління інженерними мережами будинку.[12]

2.2 Побудова макетної схеми системи «Розумний будинок»

У цьому розділі ми розглянемо процес побудови макетної схеми, яка включає в себе інтеграцію та з’єднання різноманітних компонентів, що складають систему «Розумний будинок».Основною метою є створення функціональної моделі, яка демонструє, як усі елементи можуть взаємодіяти один з одним для досягнення цілей автоматизації та управління в умовах сучасного житла.

На початку необхідно визначити роль кожного компонента в загальному функціонуванні системи. Плата Raspberry Pi виступає в ролі центрального контролера, який забезпечує керування всіма підключеними елементами. Вона обробляє дані, отримані від датчиків, та надсилає команди на виконавчі елементи, такі як реле, мотор з редуктором, вентилятор, звуковий модуль, та світлодіодні стрічки.

Сонячна панель забезпечує систему альтернативною енергією, що робить її екологічно чистою. Акумуляторна батарея накопичує цю енергію, щоб живити

систему навіть під час відсутності сонячного світла. Таким чином, система може працювати безперебійно в будь-яких умовах.[13]

Мотор з редуктором додає механічну дію до системи, наприклад, для автоматичного відкриття або закриття вікон, а також може бути використаний у різноманітних автоматизованих системах, таких як підйомні механізми або дверні замки. RFID модуль забезпечує можливість ідентифікації користувачів, що дозволяє реалізувати системи контролю доступу, наприклад, для автоматичного відкриття дверей[14].

Звуковий модуль може бути використаний для виведення аудіосигналів, таких як сповіщення про події в системі. Кнопка-перемикач забезпечує зручний інтерфейс для ручного управління системою, на приклад, для увімкнення або вимкнення певних функцій.[15]

Датчики руху та освітленості виконують функцію моніторингу навколишнього середовища. Датчики руху активують певні функції, коли виявляють присутність людини, тоді як датчики освітленості можуть керувати інтенсивністю освітлення в приміщеннях в залежності від рівня природного світла.

Вентилятор, реле, драйвер мотору та світлодіодні стрічки забезпечують додаткові функції автоматизації, включаючи регулювання температури, освітлення та інші аспекти комфорту в домі. Діоди виконують функцію захисту та стабілізації напруги в електричних ланцюгах.

Усі ці компоненти з’єднуються за допомогою проводів та роз’ємів, що дозволяє створити компактну та ефективну систему. Застосування макетної плати полегшити процес прототипування, надаючи можливість легко підключати та перепідключати елементи, вивчати їх взаємодію, тестувати та налаштовувати.

Таким чином, побудова цієї макетної схеми не лише демонструє принципи роботи різних елементів, але й слугує основою для фізичного підключення елементів системи.

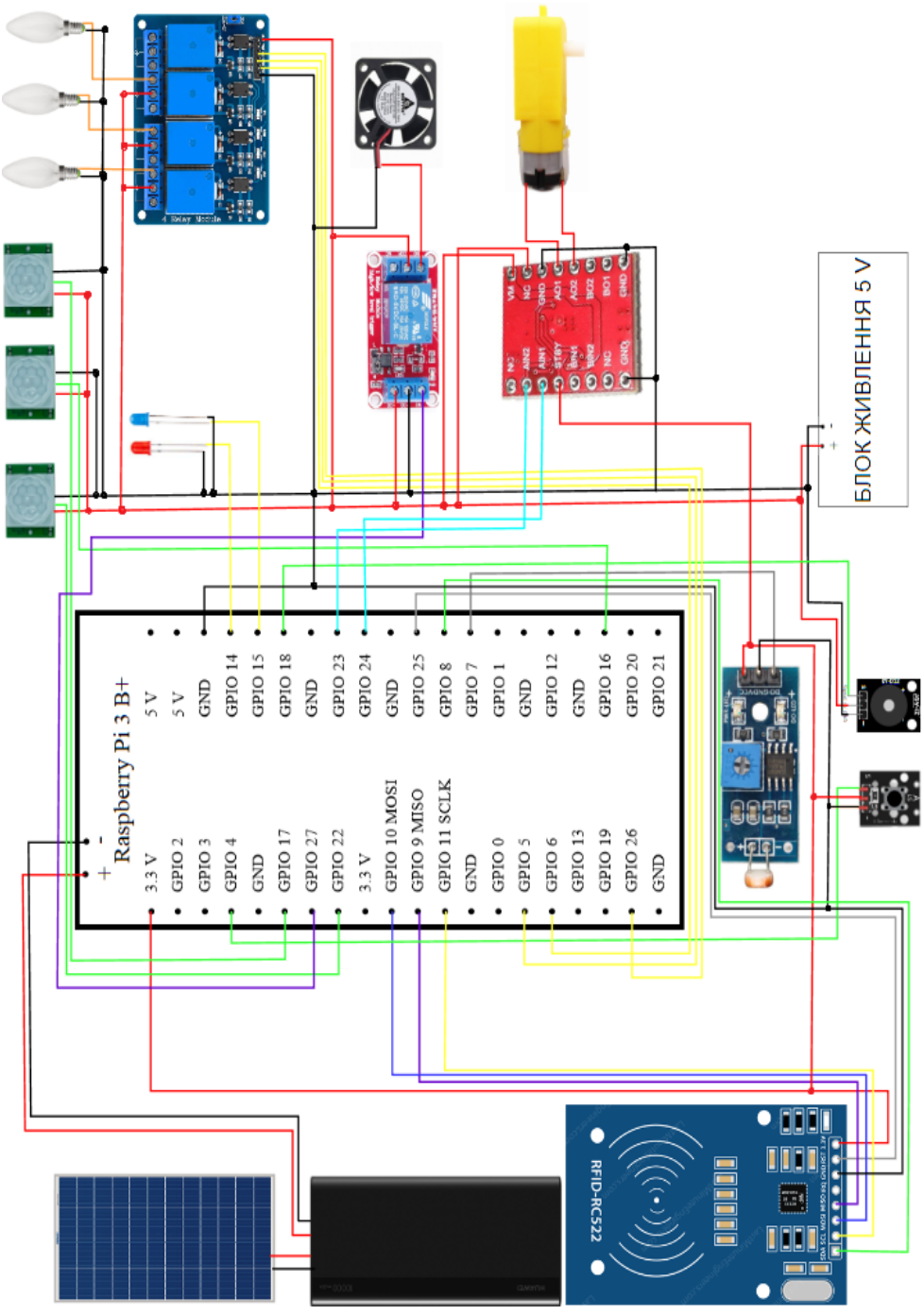

Рисунок 2.2 –Макетна схема підключення

2.3 Програмна реалізація системи управління

Для комфортного і енергоефективного користування системою розроблено три режими керування. Автоматичний режим працює на основі показників датчиків. Ручний режим дозволяє керувати пристроями через додаток або веб-інтерфейс. Режим за розкладом забезпечує автоматичне увімкнення пристроїв у заданий час через веб-сторінку, оптимізуючи енергоспоживання.

Кожна з імпортованих бібліотек та модулів виконує окрему функцію для забезпечення роботи системи "розумного будинку".

```
import RPi.GPIO as GPIO
import paho.mqtt.client as mqtt
from mfrc522 import SimpleMFRC522
import json
import time
import threading
import os
from datetime import datetime
```

Рисунок 2.3 – Підключені бібліотеки

Система керування використовує набір бібліотек Python для забезпечення функціонування всіх компонентів розумного будинку. Бібліотека RPi.GPIO керує GPIO пінами Raspberry Pi, дозволяючи читати дані з датчиків та керувати виконавчими пристроями через встановлення HIGH або LOW рівнів на відповідних пінах. Модуль paho.mqtt.client реалізує обмін даними через протокол MQTT, забезпечуючи публікацію стану пристроїв та отримання команд керування від веб-інтерфейсу в реальному часі. Бібліотека SimpleMFRC522 забезпечує роботу RFID-зчитувача RC522 через SPI-інтерфейс для зчитування ідентифікаторів карток та керування режимом охорони. Модуль json обробляє серіалізацію даних для обміну повідомленнями через MQTT та збереження конфігурацій.[16] Бібліотека time створює часові затримки між опитуваннями

датчиків, а модуль threading організовує паралельне виконання завдань через окремі потоки для одночасної обробки RFID-карток, моніторингу датчиків та MQTT-комунікації.[17] Модуль os перевіряє наявність файлів конфігурації перед їх завантаженням. Бібліотека datetime забезпечує реалізацію автоматизованих сценаріїв за розкладом через отримання поточного часу та порівняння з налаштованими часовими інтервалами для кожного пристрою.

```
class SmartHomeRPI:
    def __init__(self):
        self.lock = threading.Lock()
        self.running = True
        self.states = {
            'auto_mode': True, 'security': False, 'fan': False,
            'window': False, 'lamps': [False, False, False]
        }
        self.load_config()
        self.setup_gpio()
        self.mqtt_setup()
        self.rfid_reader = SimpleMFRC522()
```

Рисунок 2.4 Ініціалізація стану пристроїв

Клас SmartHomeRPI виконує ініціалізацію всіх компонентів системи керування розумним будинком. Створюється об'єкт блокування для забезпечення безпечного доступу до спільних змінних з різних потоків виконання. Словник states зберігає поточний стан системи, включаючи режим автоматизації, стан охорони, вентилятора, вікна та трьох ламп освітлення. Процес ініціалізації включає послідовне завантаження конфігурації розкладу з файлу, налаштування GPIO пінів Raspberry Pi, підключення до MQTT брокера для мережевого обміну даними та ініціалізацію RFID-зчитувача для системи контролю доступу. Така структура забезпечує організоване запуск всіх підсистем розумного будинку.

Ініціалізація системи розумного будинку виконується у чотири послідовних етапи. Перший етап – завантаження конфігурації розкладів з JSON-файлу, що містить налаштування автоматизації для всіх пристроїв. Другий етап – налаштування GPIO пінів Raspberry Pi, де система конфігурує режими роботи,

встановлює початкові стани та налаштовує підтягувальні резистори для датчиків. Третій етап – встановлення з'єднання з MQTT брокером для двостороннього зв'язку через підписку на топіки та налаштування обробників повідомлень. Четвертий етап – ініціалізація RFID-зчитувача SimpleMFRC522 для роботи з RC522-модулем та забезпечення контролю доступу через RFID-картки.

```
def load_config(self):
    default_config = {
        'fan': {'mode': 'disabled', 'days': [], 'start': '09:00', 'end': '18:00'},
        'window': {'mode': 'disabled', 'days': [], 'start': '07:00', 'end': '21:00'},
        'lamp1': {'mode': 'disabled', 'days': [], 'start': '18:00', 'end': '23:00'},
        'lamp2': {'mode': 'disabled', 'days': [], 'start': '18:00', 'end': '23:00'},
        'lamp3': {'mode': 'disabled', 'days': [], 'start': '18:00', 'end': '23:00'},
    }
    if os.path.exists(CONFIG_FILE):
        try:
            with open(CONFIG_FILE, 'r') as f:
                self.automation_config = json.load(f)
            print("Config loaded from file.")
            for key, val in default_config.items():
                if key not in self.automation_config: self.automation_config[key] = val
        except: self.automation_config = default_config
    else: self.automation_config = default_config

def save_config(self):
    try:
        with open(CONFIG_FILE, 'w') as f: json.dump(self.automation_config, f, indent=4)
    except Exception as e: print(f"Error saving config: {e}")
```

Рисунок 2.5 Завантаження та збереження конфігурації системи

Метод load_config реалізує механізм завантаження налаштувань автоматизації з файлу конфігурації у форматі JSON. Кожен пристрій системи має власний набір параметрів, що включає режим роботи (disabled, daily, schedule), перелік днів тижня для активації, час початку та закінчення роботи у форматі 24-годинного формату. Перед завантаженням система створює стандартну конфігурацію за замовчуванням, яка містить базові налаштування для всіх пристроїв, включаючи вентилятор та привід вікна з типовими часовими інтервалами роботи.

Процес завантаження включає перевірку наявності файлу конфігурації у файловій системі. Якщо файл існує, система намагається його прочитати та перетворити JSON-дані у словник Python. У разі успішного завантаження виконується автоматичне доповнення конфігурації – система перевіряє наявність усіх пристроїв у завантажених даних і додає стандартні налаштування для нових компонентів, які могли бути додані після створення файлу. Це забезпечує зворотну сумісність та гнучкість при розширенні системи. У випадку відсутності файлу або виникнення помилки при читанні використовується повна стандартна конфігурація.

Метод save_config забезпечує збереження поточного стану конфігурації у файл для постійного зберігання налаштувань між перезапусками системи. Важливою частиною є використання механізму блокування потоків (threading.Lock), що запобігає одночасному доступу до файлу з різних потоків виконання. Блокування гарантує виконання операції запису – під час збереження жоден інший потік не може змінити дані конфігурації, що виключає можливість пошкодження файлу або втрати даних. Конфігурація зберігається у читабельному форматі з відступами для зручності ручного редагування. Обробка винятків забезпечує стійкість системи до помилок файлової системи, виводячи інформативне повідомлення про проблему без аварійного завершення програми.

```
def setup_gpio(self):
    GPIO.setwarnings(False)
    GPIO.setmode(GPIO.BCM)
    for pin in PINS['lamps']: GPIO.setup(pin, GPIO.OUT, initial=GPIO.HIGH)
    for pin in [PINS['fan'], PINS['win_forward'], PINS['win_back'], PINS['led_red'], PINS['led_blue'], PINS['buzzer']]:
        GPIO.setup(pin, GPIO.OUT, initial=GPIO.LOW)
    for pin in PINS['motion'] + [PINS['light_sensor'], PINS['button']]:
        GPIO.setup(pin, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
    GPIO.output(PINS['led_blue'], GPIO.HIGH)
    GPIO.add_event_detect(PINS['button'], GPIO.FALLING, callback=self.handle_manual_button, bouncetime=300)
```

Рисунок 2.6 Налаштування GPIO пінів системи

Метод setup_gpio виконує повну конфігурацію GPIO пінів Raspberry Pi для роботи з апаратними компонентами. Система вимикає попередження про зайняті

піни, що дозволяє уникнути конфліктів при повторному запуску без перезавантаження. Встановлюється режим нумерації BCM (Broadcom SOC channel), який використовує номери GPIO каналів процесора та забезпечує сумісність з технічною документацією.

Конфігурація вихідних пінів враховує специфіку підключених пристроїв. Піни ламп налаштовуються як виходи з початковим станом HIGH через інвертовану логіку релейних модулів (Active LOW), де HIGH означає вимкнений стан. Це запобігає випадковому увімкненню при ініціалізації. Вентилятор, драйвери мотора, світлодіоди та п'єзодинамік налаштовуються з початковим станом LOW, використовуючи пряму логіку керування (Active HIGH).

Вхідні піни сенсорів конфігуруються з увімкненими підтягувальними резисторами до землі (pull-down) для стабільної роботи.[18] Резистори забезпечують визначений логічний рівень при відсутності сигналу, усуваючи електричні шуми та випадкові спрацювання. Після налаштування вмикається синій світлодіод як індикатор готовності. Для кнопки налаштовується обробник переривань зі спадаючим фронтом сигналу та часом 300 мілісекунд для запобігання множинним хибним спрацюванням.

```
def mqtt_setup(self):
    self.client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION2)
    self.client.username_pw_set(MQTT_USER, MQTT_PASSWORD)
    self.client.on_connect = self.on_connect
    self.client.on_message = self.on_message
    self.client.will_set(f"{TOPIC_PREFIX}/simulator/status", json.dumps({"state": "offline"}), retain=True, qos=1)
```

Рисунок 2.7 Налаштування MQTT клієнта

Метод mqtt_setup забезпечує ініціалізацію MQTT клієнта для організації мережевого зв'язку між контролером Raspberry Pi та інтерфейсом керування системою.[19] Спочатку створюється екземпляр клієнта на базі нової версії API протоколу, після чого система проходить автентифікацію на брокері, передаючи логін та пароль користувача. Визначаються дві функції зворотного виклику:

перша активується при встановленні з'єднання з брокером та виконує підписку на потрібні канали зв'язку, друга обробляє вхідні команди керування, що надходять від інтерфейсів користувача. Важливим елементом конфігурації є механізм Last Will and Testament, який працює як система контролю доступності – у випадку несподіваного відключення програми або розриву мережі, брокер самостійно розсилає повідомлення про недоступність системи у відповідний канал. Це повідомлення зберігається на брокері завдяки параметру retain та доставляється з гарантією через встановлений рівень QoS 1, який забезпечує принаймні одноразову доставку повідомлення до отримувача з підтвердженням отримання.

```
def on_connect(self, client, userdata, flags, rc, properties=None):
    if rc == 0:
        print("Connected to MQTT!")
        client.subscribe(f"{TOPIC_PREFIX}/+/set")
        client.subscribe(f"{TOPIC_PREFIX}/automation/+/set")
        client.subscribe(f"{TOPIC_PREFIX}/automation/get_config/set")
        client.subscribe(f"{TOPIC_PREFIX}/automation/save_all/set")
        client.subscribe(f"{TOPIC_PREFIX}/all/get_status/set")
        self.publish_state("simulator/status", "online", retain=True)
        self.sync_all_states()
        self.publish_config()
    else: print(f"MQTT failed rc={rc}")
```

Рисунок 2.8 Обробка підключення до MQTT брокера

Ініціалізація MQTT забезпечує мережеву взаємодію компонентів системи розумного будинку. У процесі налаштування створюється екземпляр клієнта на основі другої версії API протоколу MQTT. Виконується автентифікація на MQTT брокері шляхом передачі облікових даних користувача, що забезпечує захист від несанкціонованого доступу до системи керування. Здійснюється реєстрація callback-функцій для асинхронної обробки мережевих подій без блокування основного потоку виконання програми.[20] Функція обробки підключення

забезпечує автоматичну підписку на топіки після встановлення з'єднання, тоді як функція обробки повідомлень виконує декодування JSON-структур та передачу команд керування відповідним підсистемам.

При виявленні розриву з'єднання або аварійного завершення роботи програми. Параметр retain забезпечує збереження останнього відомого статусу на брокері для інформування клієнтів, що підключаються до системи пізніше. Передбачено обробку помилкових ситуацій при підключенні з виведенням діагностичної інформації, включаючи код помилки, що полегшує процес налагодження та усунення проблем мережевої взаємодії.

```
def on_message(self, client, userdata, msg):
    try:
        topic = msg.topic
        payload = json.loads(msg.payload.decode())

        if topic.endswith("all/get_status/set"):
            self.sync_all_states()
            return

        if topic.endswith("automation/get_config/set"):
            self.publish_config()
            return

        if topic.endswith("automation/save_all/set"):
            new_config = payload.get("state", payload)
            with self.lock: self.automation_config = new_config
            self.save_config()
            self.publish_config()
            return

        device = topic.split('/')[1]
        cmd_state = payload.get("state")

        with self.lock:
            if device == 'mode':
                self.states['auto_mode'] = cmd_state
                self.publish_state('mode', cmd_state, retain=True)
            elif device == 'security':
                self.set_security(cmd_state)
            elif not self.states['auto_mode']:
                if device == 'fan': self.set_fan(cmd_state)
                elif device == 'window': self.control_window(cmd_state)
                elif device.startswith('lamp'):
                    idx = int(device[4:]) - 1
                    if 0 <= idx < 3: self.set_lamp(idx, cmd_state)
    except Exception as e: print(f"Msg error: {e}")
```

Рисунок 2.9 Обробка вхідних MQTT повідомлень

Метод on_message є центральним обробником усіх вхідних команд, що надходять від веб-інтерфейсу через MQTT протокол.[21] Функція виконує декодування JSON-даних з отриманого повідомлення та визначає тип команди на основі топіку MQTT. Реалізовано чотири основні категорії обробки команд. Перша категорія – запит синхронізації всіх станів системи, який активується при перезавантаженні веб-інтерфейсу та забезпечує передачу поточного стану всіх пристроїв клієнту. Друга категорія – запит на отримання поточної конфігурації автоматизації з розкладами роботи пристроїв. Третя категорія – збереження нової конфігурації автоматизації, яка включає оновлення даних у пам'яті з використанням блокування потоків, запис на диск та підтвердження оновлення клієнту.

Обробка команд пристроїв здійснюється з урахуванням поточного режиму роботи системи. Системні команди, такі як перемикання загального режиму автоматизації та режиму охорони, виконуються незалежно від активного режиму. Команди керування окремими пристроями (вентилятор, вікно, лампи) обробляються лише при вимкненому автоматичному режимі, що запобігає конфліктам між ручним та автоматичним керуванням. При спробі ручного керування в активному автоматичному режимі команда ігнорується з виведенням відповідного повідомлення. Вся логіка обробки виконується в межах блоку try-except для забезпечення стійкості системи до помилок або виконання команд.

```
def publish_state(self, subtopic, state, retain=False):
    full_topic = f"{TOPIC_PREFIX}/{subtopic}" if subtopic.endswith(('state', 'status')) else f"{TOPIC_PREFIX}/{subtopic}/state"
    payload = json.dumps(state) if isinstance(state, dict) else json.dumps({"state": state})
    self.client.publish(full_topic, payload, retain=retain, qos=1)
```

Рисунок 2.10 Публікація стану пристроїв через MQTT

Метод publish_state забезпечує моніторинг та передачу актуальної інформації про стан пристроїв до веб-інтерфейсу через MQTT протокол. Функція викликається кожного разу при зміні стану будь-якого компонента системи, забезпечуючи синхронізацію інформації між контролером та користувацьким

інтерфейсом у реальному часі. Автоматично формується топік у форматі home/{device}/state для кожного пристрою, після чого дані серіалізуються у JSON-формат з урахуванням типу вхідної інформації. Параметр retain дозволяє зберігати останній відомий стан на брокері, що забезпечує миттєву синхронізацію при підключенні нових клієнтів без необхідності очікування наступної зміни стану. Така архітектура гарантує, що веб-інтерфейс завжди відображає поточний стан усіх компонентів системи розумного будинку. Безперервне оновлення даних відбувається автоматично при будь-яких змінах, включаючи спрацювання датчиків, виконання команд користувача або активацію автоматизованих сценаріїв. Це забезпечує користувачу актуальну інформацію про роботу системи без необхідності ручного оновлення сторінки або додаткових запитів на отримання поточного стану.

```
def set_fan(self, state):
    GPIO.output(PINS['fan'], GPIO.HIGH if state else GPIO.LOW)
    self.states['fan'] = state
    self.publish_state('fan', state, retain=True)
```

Рисунок 2.11 Метод керування вентилятором

Метод set_fan реалізує керування вентилятором системи з використанням прямої логіки GPIO. При отриманні команди на увімкнення встановлюється рівень HIGH на відповідному піні, що активує реле вентилятора, тоді як команда вимкнення встановлює рівень LOW. Після зміни фізичного стану пристрою оновлюється внутрішній стан у словнику states для збереження актуальної інформації про роботу компонента.

```
def set_lamp(self, index, state):
    if self.states['lamps'][index] != state:
        GPIO.output(PINS['lamps'][index], GPIO.LOW if state else GPIO.HIGH)
        self.states['lamps'][index] = state
        self.publish_state(f'lamp{index+1}', state, retain=True)
```

Рисунок 2.12 Метод керування освітленням

Метод set_lamp реалізує керування лампами освітлення з урахуванням інвертованої логіки релейних модулів. Перед виконанням команди здійснюється перевірка поточного стану лампи для уникнення зайвих операцій з GPIO та MQTT публікацій у випадку, якщо стан не змінився. Релейні модулі використовують інвертовану логіку керування, де рівень LOW на GPIO піні відповідає увімкненому стану лампи, а рівень HIGH – вимкненому. Після встановлення відповідного рівня на піні оновлюється внутрішній стан у масиві lamps за індексом конкретної лампи. Завершується процес публікацією нового стану через MQTT протокол з ідентифікатором лампи у форматі lamp1, lamp2 або lamp3 та параметром retain для забезпечення синхронізації з веб-інтерфейсом.

```
def set_security(self, state):
    self.states['security'] = state
    GPIO.output(PINS['led_red'], GPIO.HIGH if state else GPIO.LOW)
    self.publish_state('security', state, retain=True)
    if not state: self.beep(0.1)
```

Рисунок 2.13 Метод керування системою охорони

Метод set_security відповідає за керування режимом охорони в системі розумного будинку та синхронізацію його стану з індикаторами та MQTT-інтерфейсом. Після прийняття нового значення параметра state метод оновлює внутрішній словник states, фіксуючи активний або неактивний статус охорони.

Для користувача режим охорони дублюється візуально через червоний LED-індикатор. Управління світлодіодом здійснюється без інвертованої логіки: GPIO.HIGH відповідає увімкненню LED у разі активованої охорони, тоді як GPIO.LOW вимикає його при знятті охорони. Після зміни стану метод виконує MQTT-публікацію через функцію publish_state, передаючи ключ security та нове значення з параметром retain=True, що забезпечує коректну синхронізацію стану із зовнішніми клієнтами та веб-інтерфейсом.

Додатково, при вимкненні режиму охорони передбачено короткий акустичний сигнал підтвердження. Умова if not state: викликає метод beep з тривалістю 0.1 секунди, що реалізує звукове оповіщення користувача.

```
def control_window(self, open_cmd):
    if open_cmd == self.states['window']: return
    GPIO.output(PINS['win_forward'], GPIO.HIGH if open_cmd else GPIO.LOW)
    GPIO.output(PINS['win_back'], GPIO.LOW if open_cmd else GPIO.HIGH)
    time.sleep(1.0)
    GPIO.output(PINS['win_forward'], GPIO.LOW)
    GPIO.output(PINS['win_back'], GPIO.LOW)
    self.states['window'] = open_cmd
    self.publish_state('window', open_cmd, retain=True)
```

Рисунок 2.14 Метод керування жалюзями

Метод control_window реалізує керування напрямком обертання мотора, який використовується для відкриття або закриття жалюзів через H-bridge драйвер. На початку виконується порівняння командного параметра open_cmd із поточним станом, що зберігається у словнику states. Якщо стан вже відповідає запитаному положенню, метод завершує роботу без виконання GPIO-операцій, запобігаючи зайвому перемиканню драйвера та механічному зношуванню мотора.

У разі зміни стану формується напрям обертання двигуна шляхом подачі керуючих рівнів на два GPIO-виходи H-bridge модуля. Параметр open_cmd визначає режим роботи: при значенні True активується напрямок «forward», а лінія back» залишається у LOW; при значенні False логіка інвертується — мотор переходить у режим reverse.

Після встановлення логічних рівнів на пінів мотора виконується пауза тривалістю 1 секунду за допомогою time.sleep(1.0). Цей інтервал визначає час роботи двигуна достатній для виконання повного циклу відкриття або закриття.

```
def sync_all_states(self):
    with self.lock:
        self.publish_state('mode', self.states['auto_mode'], retain=True)
        self.publish_state('security', self.states['security'], retain=True)
        self.publish_state('fan', self.states['fan'], retain=True)
        self.publish_state('window', self.states['window'], retain=True)
        for i in range(3): self.publish_state(f'lamp{i+1}', self.states['lamps'][i], retain=True)
```

Рисунок 2.17 Синхронізація внутрішніх станів через MQTT

Метод sync_all_states відповідає за повну синхронізацію всіх внутрішніх станів пристрою з MQTT-брокером. Така операція необхідна після перезавантаження контролера, перепідключення до мережі або при увімкненні системи, щоб веб-інтерфейс і зовнішні клієнти одразу отримали актуальні значення всіх параметрів.

Для запобігання колізіям під час паралельного доступу використовується механізм блокування: виконання всього блоку публікацій обгорнуте у with self.lock. Це гарантує, що в момент синхронізації жоден інший потік не змінить станів і не викличе publish одночасно.

Далі метод послідовно публікує ключові стани системи через publish_state, передаючи їх значення зі словника states та прапор retain=True. Retain-флаг забезпечує збереження повідомлення на MQTT-брокері, щоб нові клієнти одразу після підключення отримали актуальні дані без додаткових запитів.

Для освітлення використовується цикл for i in range(3), який публікує стани усіх трьох ламп. Ідентифікатор формується у вигляді lamp1, lamp2, lamp3, а кожне значення береться зі списку states['lamps'] за відповідним індексом. Усі публікації відбуваються з retain=True для повної синхронізації із зовнішнім інтерфейсом.

```python
def check_schedule(self, config):
    if config['mode'] != 'schedule': return False
    now = datetime.now()
    current_js_day = (now.weekday() + 1) % 7
    if current_js_day not in config['days']: return False
    current_time = now.strftime("%H:%M")
    if config['start'] <= config['end']: return config['start'] <= current_time < config['end']
    else: return config['start'] <= current_time or current_time < config['end']
```

Рисунок 2.18 Перевірка режиму за розкладом

Метод check_schedule виконує перевірку, чи відповідає поточний момент часу умовам розкладу, визначеним у параметрі config. Логіка методу дозволяє враховувати день тижня, часовий інтервал та навіть ситуації, коли розклад перетинає північ.

На початку роботи перевіряється режим активації — якщо ключ mode у конфігурації не дорівнює schedule, метод одразу повертає False, оскільки розклад у такому випадку не є активним. Поточний час отримується через datetime.now(), після чого відбувається конвертація дня тижня у формат JavaScript: значення Python (0 - понеділок, 6 - неділя) зміщується на одну позицію та нормалізується модулем 7 для сумісності. Якщо розклад не містить поточного дня у списку days, метод також завершується поверненням False.

Далі локальний час переводиться у рядковий формат HH:MM, щоб коректно виконати порівняння з часовими межами у конфігурації. Перевірка інтервалу часу здійснюється у двох режимах. Якщо start менше або дорівнює end, використовується стандартна логіка перевірки проміжку. Якщо ж інтервал проходить через північ (наприклад 23:00–06:00), застосовується альтернативна умова: поточний час може бути як після початку проміжку, так і до його завершення. У результаті метод повертає True, якщо поточний час відповідає активному часовому діапазону, і False - у протилежному випадку.

```
    def start(self):
        try:
            self.client.connect(MQTT_BROKER, MQTT_PORT, 60)
            self.client.loop_start()
            threading.Thread(target=self.loop_sensors_and_auto, daemon=True).start()
            threading.Thread(target=self.loop_rfid, daemon=True).start()
            print("SYSTEM STARTED")
            while True: time.sleep(1)
        except KeyboardInterrupt: pass
        finally:
            self.running = False
            self.publish_state("simulator/status", "offline", retain=True)
            time.sleep(1)
            self.client.loop_stop()
            GPIO.cleanup()

if __name__ == '__main__':
    SmartHomeRPI().start()
```

Рисунок 2.18 Перевірка режиму за розкладом

Метод start є точкою входу програми та забезпечує ініціалізацію і координацію роботи всіх підсистем розумного будинку. Метод реалізує безпечний запуск із обробкою виключень та коректним завершенням роботи.

На початку методу встановлюється з'єднання з MQTT брокером через виклик self.client.connect() з параметрами адреси брокера, порту та таймауту підключення 60 секунд. Після успішного підключення запускається мережевий потік MQTT методом self.client.loop_start(), який забезпечує асинхронну обробку вхідних повідомлень та підтримку з'єднання у фоновому режимі.

Для паралельної роботи різних підсистем створюються два окремі потоки через модуль threading. Перший потік виконує функцію loop_sensors_and_auto, яка відповідає за циклічне опитування датчиків та виконання автоматизованої логіки керування. Другий потік виконує функцію loop_rfid для обробки подій від RFID зчитувача. Обидва потоки створюються з параметром daemon=True, що означає їх автоматичне завершення при закритті головної програми.

Після успішного запуску всіх підсистем виводиться повідомлення SYSTEM STARTED у консоль. Головний потік програми входить у нескінченний цикл

while True з паузою в одну секунду, що підтримує програму в активному стані та дозволяє фоновим потокам продовжувати виконання своїх завдань.

Блок except KeyboardInterrupt перехоплює переривання від користувача через комбінацію клавіш Ctrl+C, дозволяючи здійснити коректне завершення роботи програми. У блоці finally виконуються операції очищення ресурсів: встановлюється прапорець self.running у False для зупинки всіх циклів у фонових потоках, публікується статус offline через MQTT для інформування клієнтів про відключення системи, зупиняється мережевий потік MQTT через loop_stop() та звільняються GPIO піни методом GPIO.cleanup(), що запобігає конфліктам при наступному запуску програми.

Конструкція if name == 'main' забезпечує виконання коду лише при прямому запуску файлу, створюючи екземпляр класу SmartHomeRPI та викликаючи його метод start().

Висновки до 2-го розділу

У цьому розділі було здійснено комплексну реалізацію програмної частини системи розумного будинку, що включає проектування архітектури, побудову макетної схеми та розробку програмного забезпечення для керування всіма підсистемами. На основі аналізу об'єкта автоматизації було визначено склад обладнання та функціональні вимоги до системи, що охоплюють освітлення з трьома зонами керування, систему вентиляції, електропривод вікна, охоронну систему та контроль доступу через RFID технологію.

Реалізовано комплексну систему автоматизації з підтримкою роботи за розкладом та за показаннями датчиків. Програмне забезпечення дозволяє налаштовувати часові інтервали роботи для кожного пристрою з урахуванням днів тижня, автоматично керувати освітленням від датчиків руху та регулювати стан вікна залежно від рівня освітленості. Охоронна система з RFID контролем

доступу забезпечує захист приміщення та звукову індикацію при спрацюванні датчиків або використанні неавторизованих карток.

Впроваджено механізм збереження конфігурації у форматі JSON, що гарантує збереження налаштувань між перезавантаженнями системи. Використання потокобезпечних операцій забезпечує цілісність даних при одночасному доступі з різних потоків виконання. Система коректно завершує роботу зі звільненням всіх ресурсів, що запобігає конфліктам при повторному запуску та гарантує стабільність функціонування.

Розроблене програмне забезпечення забезпечує надійне керування всіма підсистемами розумного будинку та готове до інтеграції з веб-інтерфейсом для дистанційного моніторингу та управління. Модульна структура коду дозволяє легко розширювати функціональність системи додаванням нових пристроїв та алгоритмів автоматизації, забезпечуючи користувачам зручний і надійний доступ до керування системою.

РОЗДІЛ 3 РОЗРОБКА WEB ІНТЕРФЕСУ ДЛЯ КЕРУВАННЯ СИСТЕМОЮ

У цьому розділі буде розроблено веб-інтерфейс для взаємодії користувача з системою розумного будинку. Основною метою є створення зручної та функціональної панелі керування, яка дозволить користувачам здійснювати моніторинг стану всіх підсистем у реальному часі, перемикатися між автоматичним та ручним режимами роботи, налаштовувати розклади автоматизації для кожного пристрою окремо з можливістю вибору днів тижня та часових інтервалів.

Веб-інтерфейс буде реалізовано з використанням бібліотеки React для створення динамічних компонентів та Material-UI для забезпечення сучасного дизайну інтерфейсу. Для кожного пристрою буде передбачено три режими роботи: вимкнений, за розкладом та за сенсорами, що забезпечить гнучкість налаштування системи під різні сценарії використання. Інтеграція з MQTT брокером через бібліотеку mqtt.js дозволить отримувати оновлення станів пристроїв у реальному часі та відправляти команди керування без необхідності перезавантаження сторінки.

3.1 Реалізація MQTT клієнта веб-інтерфейсу

Для забезпечення двостороннього обміну даними між веб-інтерфейсом користувача та контролером Raspberry Pi необхідна реалізація надійного механізму комунікації, який дозволить передавати команди керування пристроями та отримувати оновлення їх станів у реальному часі. Оскільки серверна частина системи використовує MQTT протокол для обміну повідомленнями, веб-додаток повинен мати можливість підключатися до MQTT брокера та взаємодіяти з ним безпосередньо з браузера.[22]

```
import mqtt from 'mqtt';

export const MQTT_CONFIG = {
  broker: window.location.hostname === 'localhost' ? 'localhost' : window.location.hostname,
  port: 9001,
  username: 'mqtt_user',
  password: 'admin',
  topicPrefix: 'home/'
};

class MqttClient {
  constructor() {
    this.client = null;
    this.callbacks = new Map();
    this.connectionCallback = null;
    this._isConnected = false;
    this.connectUrl = `ws://${MQTT_CONFIG.broker}:${MQTT_CONFIG.port}`;
    this.options = {
      username: MQTT_CONFIG.username,
      password: MQTT_CONFIG.password,
      clean: true,
      connectTimeout: 4000,
      reconnectPeriod: 5000,
      protocolVersion: 4
    };

    this.handleConnect = this.handleConnect.bind(this);
    this.handleError = this.handleError.bind(this);
    this.handleClose = this.handleClose.bind(this);
  }
```

Рисунок 3.1 Конфігурація та конструктор MQTT клієнта

Об'єкт MQTT_CONFIG містить параметри підключення до MQTT брокера. Властивість broker визначається динамічно: при локальному доступі використовується localhost, при віддаленому – hostname з адресного рядка браузера, що дозволяє працювати як на комп'ютері розробника, так і з мобільних пристроїв у локальній мережі. Порт 9001 є стандартним для WebSocket з'єднань з MQTT брокерами.

Конструктор класу MqttClient ініціалізує властивості екземпляру. Властивість client зберігатиме об'єкт MQTT з'єднання, початкове значення null вказує на відсутність підключення. Властивість callbacks реалізована як Map для

зберігання пар топік-функція, де ключ – шлях MQTT топіку, значення – функція-обробник повідомлень.

Властивість connectUrl формується з протоколу WebSocket та параметрів конфігурації.[23]

Об'єкт options містить налаштування з'єднання: автентифікаційні дані, параметр clean для чистої сесії, connectTimeout у 4 секунди, reconnectPeriod у 5 секунд для автоматичного перепідключення та protocolVersion для стабільності WebSocket.

Завершується конструктор прив'язуванням контексту this до методів-обробників через bind, що необхідно для збереження правильного контексту при передачі цих методів як колбеків до бібліотеки mqtt.

```
connect() {
  if (this.client && (this.client.connected || this.client.connecting)) return;

  console.log(`Frontend: Connecting to ${this.connectUrl}`);
  this.client = mqtt.connect(this.connectUrl, this.options);

  this.client.on('connect', this.handleConnect);
  this.client.on('message', (topic, message) => {
    try {
      const payload = JSON.parse(message.toString());
      if (this.callbacks.has(topic)) {
        this.callbacks.get(topic)(payload);
      }
    } catch (e) {
      console.error("Frontend: MQTT message parse error:", e);
    }
  });
  this.client.on('error', this.handleError);
  this.client.on('close', this.handleClose);
}

handleConnect() {
  console.log('Frontend: Connected');
  this._isConnected = true;
  if (this.connectionCallback) this.connectionCallback(true);
  this.callbacks.forEach((_, topic) => {
    this.client.subscribe(topic, {
      qos: 0
    });
  });
}
```

Рисунок 3.2 Методи встановлення та обробки MQTT з'єднання

Метод connect реалізує встановлення з'єднання з MQTT брокером через WebSocket протокол.[24] На початку виконується перевірка існування активного з'єднання, і якщо воно вже встановлене, метод завершується. Це реалізує шаблон Singleton, гарантуючи єдине підключення протягом життєвого циклу додатку.

З'єднання створюється викликом mqtt.connect з передачею URL адреси та параметрів конфігурації. Далі встановлюються обробники подій: connect прив'язується до методу handleConnect, message отримує функцію для розбору вхідних повідомлень, яка конвертує буфер у рядок, парсить JSON через try-catch блок, перевіряє наявність колбеку для топіку в Map структурі та викликає відповідну функцію-обробник.

Метод handleConnect викликається при успішному підключенні до брокера. Метод встановлює прапорець _isConnected у true, викликає глобальний колбек connectionCallback для сповіщення інтерфейсу про зміну стану з'єднання.
Важливою особливістю методу є автоматичне відновлення підписок після переривання з'єднання. Метод виконує ітерацію по збережених колбеках через forEach та здійснює повторну підписку на кожен топік з параметром qos 0, що забезпечує безперервність функціонування додатку без необхідності ручної повторної ініціалізації підписок компонентами інтерфейсу.

```
publish(topic_suffix, state_or_payload) {
  if (!this.client || !this._isConnected) {
    console.warn("MQTT client not connected, publish canceled.");
    return;
  }

  let fullTopic;
  if (topic_suffix.startsWith(MQTT_CONFIG.topicPrefix)) {
      fullTopic = topic_suffix;
  } else {
      fullTopic = `${MQTT_CONFIG.topicPrefix}${topic_suffix}/set`;
  }
  const isSpecialPayload = topic_suffix.includes('automation/save_all') || topic_suffix.includes('all/get_status') || topic_suffix.includes('automation/get_config');

  const payload = (typeof state_or_payload === 'object' && isSpecialPayload)
    ? JSON.stringify(state_or_payload)
    : JSON.stringify({ state: state_or_payload });

  console.log(`Frontend: Publishing to ${fullTopic} with payload: ${payload}`);
  this.client.publish(fullTopic, payload, { qos: 0 });
}
```

Рисунок 3.3 Метод публікації команд до MQTT брокера

Метод publish реалізує відправку команд керування пристроями до MQTT брокера. Метод приймає два параметри: topic_suffix – ідентифікатор пристрою або частина топіку, та state_or_payload – стан пристрою або об'єкт з даними для відправки.

На початку виконується перевірка наявності активного з'єднання через умову this.client та this._isConnected, і якщо з'єднання відсутнє, метод завершується без виконання дій. Це запобігає помилкам при спробі відправки повідомлень до непідключеного брокера.

Формування повного шляху топіку виконується через умовний оператор. Якщо переданий суфікс вже містить префікс системи, він використовується без змін. В іншому випадку додається префікс з конфігурації та суфікс /set для позначення командного топіку, при цьому перевіряється наявність слеша у суфіксі для уникнення подвійного додавання /set до складних топіків.

Серіалізація даних виконується залежно від типу переданого параметра. Якщо state_or_payload є об'єктом, він безпосередньо конвертується у JSON рядок. Для примітивних значень створюється об'єкт з полем state, що забезпечує уніфікований формат повідомлень для серверної частини системи.

```
subscribe(deviceOrTopic, callback) {
  let fullTopic;
  if (deviceOrTopic && deviceOrTopic.startsWith(MQTT_CONFIG.topicPrefix)) {
    fullTopic = deviceOrTopic;
  } else if (deviceOrTopic) {
    fullTopic = `${MQTT_CONFIG.topicPrefix}${deviceOrTopic}/state`;
  } else return;

  this.callbacks.set(fullTopic, callback);
  if (this.client && this._isConnected) {
    this.client.subscribe(fullTopic, {
      qos: 0
    });
  }
}
```

Рисунок 3.4 Метод підписки на оновлення станів пристроїв

Метод subscribe реалізує підписку на MQTT топіки для отримання оновлень станів пристроїв. Метод приймає два параметри: deviceOrTopic – ідентифікатор пристрою або повний шлях топіку, та callback – функція-обробник, яка викликатиметься при отриманні повідомлень з цього топіку.

Формування повного шляху топіку виконується через умовну логіку. Якщо переданий параметр вже містить системний префікс, він використовується без змін як повний топік. Якщо передано лише ідентифікатор пристрою, формується повний топік додаванням префіксу з конфігурації та суфіксу /state для отримання оновлень стану. При відсутності параметра метод завершується без виконання дій.

Збереження колбеку виконується через метод set Map структури, де ключем є сформований повний топік, а значенням – передана функція-обробник. Це дозволяє швидко знаходити відповідний обробник при отриманні повідомлення з конкретного топіку.

```
const mqttClientInstance = new MqttClient();
export default mqttClientInstance;
```

Рисунок 3.5 Експорт єдиного екземпляру MQTT клієнта

Створення та експорт єдиного екземпляру класу MqttClient реалізує патерн Singleton на рівні модуля JavaScript. Константа mqttClientInstance ініціалізується викликом конструктора класу, створюючи об'єкт з налаштованими властивостями для роботи з MQTT брокером.

Експорт через export default забезпечує доступність екземпляру для імпорту в інших модулях додатку. Оскільки модулі JavaScript виконуються один раз при першому імпорті, всі компоненти отримують посилання на один і той самий об'єкт, гарантуючи єдине WebSocket з'єднання протягом життєвого циклу додатку.

Це запобігає створенню множинних підключень до брокера та забезпечує централізоване управління всіма підписками і станом з'єднання в одному місці.

3.2 Розробка компонента перемикача стану пристрою

```
import React, { useState, useEffect } from 'react';
import { Switch, FormControlLabel, Typography, Box } from '@mui/material';
import mqttClient from '../mqtt/MqttClient';

const DeviceSwitch = ({ device, label, disabled, icon }) => {
  const [isChecked, setIsChecked] = useState(false);

  useEffect(() => {
    const handleStateChange = (payload) => {
      if (payload && payload.state !== undefined) {
        setIsChecked(payload.state);
      }
    };
    mqttClient.subscribe(device, handleStateChange);
    return () => {
      mqttClient.unsubscribe(device);
    };
  }, [device]);
```

Рисунок 3.6 Компонент перемикача стану пристрою

Компонент DeviceSwitch реалізує багаторазово використовуваний перемикач для керування станом пристроїв системи. Компонент приймає три параметри: device – ідентифікатор для формування MQTT топіку, label – назва пристрою та disabled – прапорець блокування ручного керування.

Для збереження стану використовується React хук useState з ініціалізацією isChecked у false, що забезпечує автоматичне перерендерування при зміні значення.

Хук useEffect встановлює зв'язок з MQTT брокером при монтуванні компонента. Функція handleStateChange обробляє вхідні повідомлення, перевіряючи наявність поля state та оновлюючи локальний стан через setIsChecked. Це забезпечує синхронізацію з реальним станом обладнання при змінах від автоматики або інших джерел.

Метод mqttClient.subscribe створює підписку на топік пристрою з передачею обробника handleStateChange. Функція повернення useEffect виконує mqttClient.unsubscribe при демонтуванні для запобігання витоку пам'яті. Масив залежностей містить лише device, що гарантує повторну підписку при зміні ідентифікатора пристрою.

```
  const handleChange = (event) => {
    if (disabled) return;
    const newState = event.target.checked;
    setIsChecked(newState);
    mqttClient.publish(device, newState);
  };

  return (
    <Box sx={{ display: 'flex', justifyContent: 'space-between', alignItems: 'center', width: '100%' }}>
      <Box sx={{ display: 'flex', alignItems: 'center', gap: 1.5 }}>
        <span style={{ fontSize: '1.5rem' }}>{icon}</span>
        <Typography>{label}</Typography>
      </Box>
      <Switch
        checked={isChecked}
        onChange={handleChange}
        disabled={disabled}
        color="primary"
      />
    </Box>
  );
};

export default DeviceSwitch;
```

Рисунок 3.7 Обробник зміни стану та рендеринг перемикача

Функція handleChange реалізує обробку подій зміни стану перемикача при взаємодії користувача з інтерфейсом. На початку функції виконується перевірка параметра disabled, і якщо він має значення true, функція завершує виконання через оператор return без виконання жодних дій. Це реалізує захист від конфліктів між автоматичним та ручним режимами роботи, блокуючи можливість ручного керування пристроєм під час активної автоматизації.

Нове значення стану отримується з об'єкта події через властивість event.target.checked, яка містить булеве значення положення перемикача після дії

користувача. Отримане значення одразу застосовується до локального стану компонента через виклик setIsChecked, що реалізує патерн оптимістичного оновлення інтерфейсу. Такий підхід забезпечує миттєву візуальну реакцію на дію користувача без очікування підтвердження від сервера, що значно покращує сприйняття швидкодії системи та якість користувацького досвіду.

Після оновлення локального стану виконується публікація нового значення через виклик mqttClient.publish з передачею ідентифікатора пристрою та нового стану. Цей виклик формує та відправляє MQTT повідомлення до брокера, який передає команду контролеру для фізичної зміни стану обладнання.

Метод return компонента повертає JSX розмітку елемента FormControlLabel з бібліотеки Material-UI, який забезпечує стандартизоване відображення перемикача з текстовою міткою.[25] Властивість control містить компонент Switch з налаштованими параметрами: checked прив'язує поточний стан до візуального відображення перемикача, onChange підключає функцію-обробник зміни стану, а disabled керує можливістю взаємодії користувача з елементом, блокуючи його при активному автоматичному режимі. Властивість label відображає текстову мітку пристрою поруч з перемикачем для зручної ідентифікації користувачем.

Компонент експортується як модуль за замовчуванням через інструкцію export default, що дозволяє імпортувати та використовувати його у інших частинах веб-додатку для створення уніфікованих елементів керування різними пристроями системи розумного будинку без дублювання коду.

3.3 Реалізація головного компонента веб-додатку

Після створення базових елементів інтерфейсу у вигляді перемикачів пристроїв та модуля зв'язку з MQTT брокером постає необхідність об'єднання цих компонентів у єдину систему з централізованим управлінням станом,

маршрутизацією інтерфейсу та координацією взаємодії всіх частин додатку. Головний компонент має забезпечити синхронізацію станів пристроїв між веб-інтерфейсом та контролером, моніторинг доступності системи та організацію структури користувацького інтерфейсу.

```
import React, { useState, useEffect, useRef } from 'react';
import {
    Box, Card, CardContent, Typography, Switch, FormControlLabel,
    ThemeProvider, createTheme, Fade, Tab, Tabs, Button, CssBaseline, Paper
} from '@mui/material';
import HomeIcon from '@mui/icons-material/Home';
import SettingsRemoteIcon from '@mui/icons-material/SettingsRemote';
import SecurityIcon from '@mui/icons-material/Security';
import ScheduleIcon from '@mui/icons-material/Schedule';
import DashboardIcon from '@mui/icons-material/Dashboard';
import DeviceSwitch from './components/DeviceSwitch';
import AutomationTab from './components/AutomationTab';
import mqttClient, { MQTT_CONFIG } from './mqtt/MqttClient';
import './App.css';

const theme = createTheme({
    palette: {
        primary: { main: '#4e54c8' },
        secondary: { main: '#ff6b6b' },
        background: { default: '#f0f2f5' }
    },
    typography: {
        fontFamily: '"Roboto", "Helvetica", "Arial", sans-serif',
        h5: { fontWeight: 600 },
    },
    shape: { borderRadius: 16 },
});
```

Рисунок 3.8 Імпорти та налаштування теми Material-UI

Блок імпортів завантажує необхідні залежності для роботи головного компонента. З бібліотеки React імпортуються хуки useState для управління станом, useEffect для виконання побічних ефектів та useRef для збереження посилань на таймери. З бібліотеки Material-UI імпортуються компоненти інтерфейсу для побудови структури додатку та ThemeProvider для застосування теми. Імпортуються іконки для візуалізації елементів навігації та індикації різних розділів системи.

Локальні компоненти DeviceSwitch та AutomationTab імпортуються для використання в структурі додатку. Модуль mqttClient та об'єкт конфігурації MQTT_CONFIG імпортуються для забезпечення зв'язку з брокером.

Об'єкт theme створюється через функцію createTheme для визначення кольорової палітри інтерфейсу з фіолетово-синіми primary кольорами, червоними secondary акцентами та світло-сірим фоном. Параметр shape встановлює радіус заокруглення 16 пікселів для всіх компонентів Material-UI, забезпечуючи сучасний вигляд інтерфейсу.

```
const SIMULATOR_STATUS_TOPIC = `${MQTT_CONFIG.topicPrefix}simulator/status`;
const AUTOMATION_CONFIG_TOPIC = `${MQTT_CONFIG.topicPrefix}automation/config`;
const HEARTBEAT_TIMEOUT = 30000;
```

Рисунок 3.9 Оголошення констант для MQTT топіків та моніторингу

Константа SIMULATOR_STATUS_TOPIC формує повний шлях MQTT топіку для отримання статусу контролера Raspberry Pi шляхом конкатенації системного префіксу з конфігурації та суфіксу simulator/status. Цей топік використовується для моніторингу доступності контролера у реальному часі.

Константа AUTOMATION_CONFIG_TOPIC визначає топік для отримання та синхронізації налаштувань автоматизації між веб-інтерфейсом та контролером через шлях automation/config з системним префіксом.

Константа HEARTBEAT_TIMEOUT встановлює граничний інтервал очікування підтвердження життєздатності контролера у 30000 мілісекунд. Якщо протягом цього часу не надходить повідомлення від контролера через топік статусу, система автоматично визначає його як недоступний та змінює індикацію на інтерфейсі.

```
function App() {
    const [currentTab, setCurrentTab] = useState(0);
    const [isAuthenticated, setIsAuthenticated] = useState(false);
    const [passwordInput, setPasswordInput] = useState('');
    const [autoMode, setAutoMode] = useState(true);
    const [securitySystem, setSecuritySystem] = useState(false);
    const [isConnected, setIsConnected] = useState(false);
    const [simulatorStatus, setSimulatorStatus] = useState('Невідомо');
    const [automationConfig, setAutomationConfig] = useState({});
    const [isConfigLoaded, setIsConfigLoaded] = useState(false);
    const heartbeatTimeout = useRef(null);
```

Рисунок 3.9 Структура станів головного компонента

Стани компонента організовані у чотири логічні групи для управління різними аспектами додатку. Група станів UI включає currentTab для відстеження активної вкладки інтерфейсу з початковим значенням 0, isAuthenticated для контролю доступу користувача з початковим значенням false та passwordInput для збереження введеного пароля.

Група станів пристроїв містить autoMode для відображення поточного режиму роботи системи з початковим значенням true, що відповідає активному автоматичному режиму, та securitySystem для відстеження стану охоронної системи з початковим значенням false.

Група станів з'єднання включає isConnected для індикації активного WebSocket підключення до MQTT брокера та simulatorStatus для відображення доступності контролера Raspberry Pi з початковим значенням 'Невідомо'.

Група станів налаштувань містить automationConfig як порожній об'єкт для збереження конфігурації розкладів усіх пристроїв та isConfigLoaded як прапорець успішного завантаження конфігурації з контролера.

Хук useRef створює посилання heartbeatTimeout з початковим значенням null для збереження ідентифікатора таймера моніторингу доступності

контролера, що дозволяє скасовувати таймер при отриманні нових повідомлень або демонтуванні компонента.

```
useEffect(() => {
    const auth = localStorage.getItem('isAuthenticated');
    if (auth === 'true') setIsAuthenticated(true);

    mqttClient.onConnectionStateChange((connected) => {
        setIsConnected(connected);
        if (connected) {
            console.log("Connected! Requesting full sync (states + config)...");
            mqttClient.publish('all/get_status', {});
            mqttClient.publish('automation/get_config', {});
        }
    });

    mqttClient.connect();
```

Рисунок 3.10 Ініціалізація підключення та синхронізація станів

Хук useEffect виконує ініціалізацію додатку при монтуванні компонента. На початку виконується перевірка збереженої сесії авторизації через отримання значення з localStorage за ключем isAuthenticated. Якщо значення дорівнює рядку 'true', стан isAuthenticated встановлюється у true, що дозволяє користувачу залишатися авторизованим після перезавантаження сторінки.

Реєстрація обробника зміни стану з'єднання виконується через метод mqttClient.onConnectionStateChange з передачею колбек-функції, яка отримує булеве значення connected. Функція оновлює локальний стан isConnected для відображення індикатора підключення в інтерфейсі.

При успішному встановленні з'єднання виконується блок умови, який ініціює повну синхронізацію системи. Публікація повідомлення у топік all/get_status запитує у контролера поточні стани всіх пристроїв системи. Публікація у топік automation/get_config запитує актуальну конфігурацію розкладів автоматизації.

Це забезпечує відображення коректних даних в інтерфейсі одразу після підключення або після відновлення з'єднання.

Виклик mqttClient.connect() ініціює встановлення WebSocket з'єднання з MQTT брокером, запускаючи процес підключення та активації всіх зареєстрованих обробників подій.

```
mqttClient.subscribe(SIMULATOR_STATUS_TOPIC, (payload) => {
    const statusValue = payload && payload.state !== undefined ? payload.state : null;
    if (statusValue) {
        const newStatus = statusValue === 'online' ? 'Онлайн' : 'Офлайн';
        setSimulatorStatus(newStatus);
        clearTimeout(heartbeatTimeout.current);
        if (newStatus === 'Онлайн') {
            heartbeatTimeout.current = setTimeout(() => {
        if (newStatus === 'Онлайн') {
            heartbeatTimeout.current = setTimeout(() => {
                setSimulatorStatus('Офлайн');
            }, HEARTBEAT_TIMEOUT);
        }
    }
});
```

Рисунок 3.11 Ініціалізація підключення та синхронізація станів

Підписка на топік SIMULATOR_STATUS_TOPIC реалізує систему heartbeat моніторингу для відстеження доступності контролера Raspberry Pi. Колбек-функція отримує payload з повідомленням та витягує значення поля state, використовуючи 'unknown' як значення за замовчуванням. Статус перетворюється у локалізований текст через умовний оператор: 'online' перетворюється на 'Онлайн', інші значення на 'Офлайн'.

При отриманні кожного повідомлення виконується clearTimeout для скасування попереднього таймера через посилання heartbeatTimeout.current. Це запобігає хибному визначенню контролера як недоступного при регулярному надходженні повідомлень.

Якщо отриманий статус дорівнює 'online', запускається новий таймер через setTimeout з інтервалом HEARTBEAT_TIMEOUT у 30 секунд. Якщо протягом цього часу не надійде нове повідомлення від контролера, таймер спрацює та

автоматично встановить статус 'Офлайн', сигналізуючи користувачу про втрату зв'язку з обладнанням.

Додатково встановлюється початковий таймер для випадку, коли контролер не відправляє жодних повідомлень після запуску веб-додатку. Якщо через 30 секунд статус залишається 'Невідомо', він автоматично змінюється на 'Офлайн'.

```
mqttClient.subscribe('mode', (p) => { if(p && p.state !== undefined) setAutoMode(p.state) });
mqttClient.subscribe('security', (p) => { if(p && p.state !== undefined) setSecuritySystem(p.state) });

mqttClient.subscribe(AUTOMATION_CONFIG_TOPIC, (payload) => {
    console.log("Automation config received:", payload);
    const configData = payload.state !== undefined ? payload.state : payload;
    setAutomationConfig(configData);
    setIsConfigLoaded(true);
});
```

Рисунок 3.11 Підписки на оновлення станів пристроїв та конфігурації

Підписки на топіки mode та security реалізують отримання оновлень глобального режиму роботи та стану охоронної системи. Колбек-функції перевіряють наявність об'єкта payload та поля state перед оновленням відповідних локальних станів autoMode та securitySystem, забезпечуючи синхронізацію інтерфейсу при змінах від контролера.

Підписка на AUTOMATION_CONFIG_TOPIC обробляє отримання конфігурації розкладів автоматизації. Виконується перевірка структури payload: якщо конфігурація загорнута у поле state, використовується payload.state, інакше весь об'єкт трактується як конфігурація. Отримані дані зберігаються у automationConfig, а прапорець isConfigLoaded встановлюється у true для сигналізації готовності даних компоненту AutomationTab.

```
const handleTabChange = (event, newValue) => setCurrentTab(newValue);
const handleAutoModeChange = (event) => {
    setAutoMode(event.target.checked);
    mqttClient.publish('mode', event.target.checked);
};
```

Рисунок 3.12 Обробники подій користувацького інтерфейсу

Функція handleTabChange обробляє перемикання між вкладками інтерфейсу, отримуючи новий індекс вкладки з параметра newValue та оновлюючи стан currentTab. Це забезпечує відображення відповідного розділу інтерфейсу при взаємодії користувача з навігаційним меню.

Функція handleAutoModeChange обробляє зміну глобального режиму роботи системи між автоматичним та ручним керуванням. Спочатку оновлюється локальний стан autoMode значенням event.target.checked для миттєвого відображення зміни в інтерфейсі. Після цього нове значення публікується у MQTT топік mode через mqttClient.publish для передачі команди контролеру та синхронізації режиму роботи з серверною частиною системи.

```
const handleLogin = (e) => {
    e.preventDefault();
    if (passwordInput === 'admin') {
        setIsAuthenticated(true);
        localStorage.setItem('isAuthenticated', 'true');
    } else alert('Невірний пароль!');
};
```

Рисунок 3.13 Обробник авторизації користувача

Функція handleLogin реалізує процес авторизації користувача у веб-інтерфейсі системи. На початку виконується виклик e.preventDefault() для запобігання стандартній поведінці браузера при відправці форми, що дозволяє обробити авторизацію без перезавантаження сторінки.

Перевірка введеного пароля виконується через порівняння значення passwordInput з рядком 'admin'. При успішній авторизації локальний стан isAuthenticated встановлюється у true через setIsAuthenticated, що дозволяє користувачу отримати доступ до головного інтерфейсу системи. Одночасно значення 'true' зберігається у localStorage за ключем 'isAuthenticated', що забезпечує збереження сесії авторизації між перезавантаженнями сторінки.

При введенні некоректного пароля виконується alert з повідомленням 'Невірний пароль!', інформуючи користувача про помилку без надання доступу до системи керування.

```
const mqttStatusColor = isConnected ? '#4caf50' : '#f44336';
const simStatusColor = simulatorStatus === 'Онлайн' ? '#4caf50' : (simulatorStatus === 'Офлайн' ? '#f44336' : 'gray');

if (!isAuthenticated) {
    return (
        <ThemeProvider theme={theme}>
            <CssBaseline />
            <Box sx={{ display: 'flex', justifyContent: 'center', alignItems: 'center', height: '100vh', bgcolor: 'background.default' }}>
                <Card sx={{ minWidth: 300, p: 3, boxShadow: 3, borderRadius: 4 }}>
                    <Typography variant="h5" textAlign="center" mb={3} color="primary" fontWeight="bold">Вхід в Систему</Typography>
                    <form onSubmit={handleLogin}>
                        <input type="password" placeholder="Пароль" value={passwordInput} onChange={(e) => setPasswordInput(e.target.value)}
                            style={{ width: '100%', padding: '12px', marginBottom: '20px', borderRadius: '8px', border: '1px solid #ddd', fontSize: '16px' }} />
                        <Button type="submit" variant="contained" fullWidth size="large" sx={{ borderRadius: 2, textTransform: 'none', fontSize: '1rem' }}>Увійти</Button>
                    </form>
                </Card>
            </Box>
        </ThemeProvider>
    );
}
```

Рисунок 3.14 Система маршрутизації вкладок інтерфейсу

Константи mqttStatusColor та simStatusColor визначають кольорову індикацію станів підключення: зелений колір для активного з'єднання, червоний для відсутності зв'язку та сірий для невизначеного стану.

Компонент Tabs реалізує навігаційне меню з двома вкладками: "Головна" з іконкою HomeIcon та "Автоматизація" з іконкою ScheduleIcon. Властивість value прив'язана до стану currentTab, а onChange підключає обробник handleTabChange для перемикання між розділами.

Маршрутизація контенту реалізована через два компоненти Box з атрибутом role="tabpanel". Перша вкладка відображається при currentTab === 0 і

містить блоки керування пристроями через компоненти DeviceSwitch. Друга вкладка відображається при currentTab === 1 і містить компонент AutomationTab з передачею props: mqttClient для публікації налаштувань, initialConfig з поточною конфігурацією, isLoaded як прапорець готовності даних та onUpdateConfig для оновлення стану при змінах.

Властивості hidden та display забезпечують умовне відображення вкладок: активна вкладка має display: 'block', неактивна – hidden: true, що оптимізує рендеринг та приховує непотрібний контент від користувача.

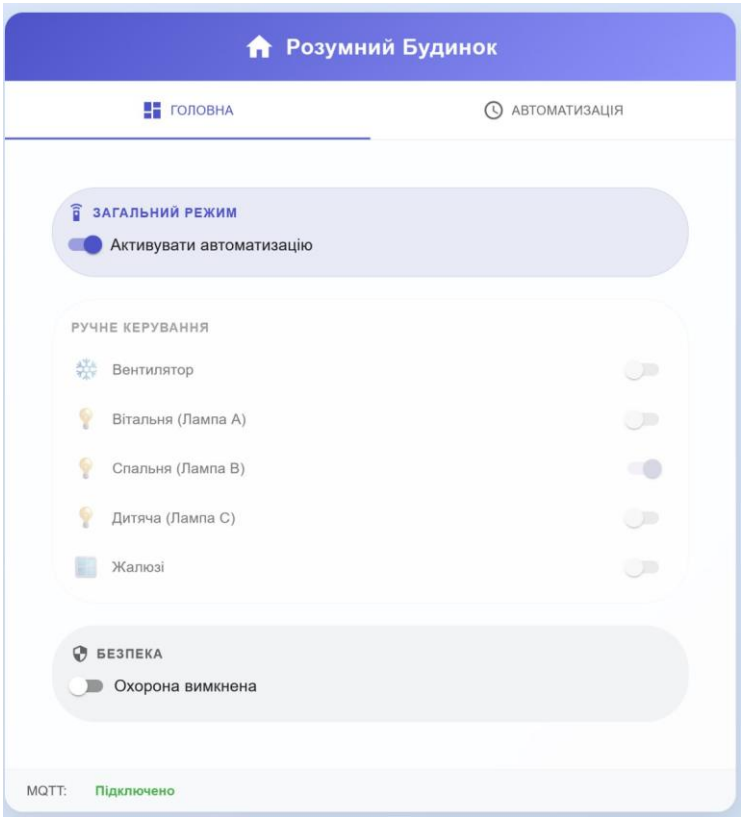

Рисунок 3.15 Інтерфейс головного компонента веб-додатку

Веб-інтерфейс розроблено з дотриманням принципів Material Design, що забезпечує сучасний та інтуїтивно зрозумілий вигляд системи. Кольорова палітра включає фіолетово-сині відтінки для основних елементів керування, червоні акценти для індикації попереджень та критичних станів, світло-сірий фон для

комфортного сприйняття інформації. Всі інтерактивні компоненти оформлено з заокругленими кутами радіусом 16 пікселів та м'якими тінями, що створює відчуття об'ємності та виділяє активні елементи. Градієнтні фони додають глибину інтерфейсу, а плавні анімації переходів забезпечують приємний відгук на дії користувача.

Структура інтерфейсу організована у три основні області: верхня частина містить навігаційне меню з вкладками для перемикання між розділами, центральна зона відображає блоки керування окремими пристроями з перемикачами станів, нижня панель представляє індикатори стану MQTT з'єднання та доступності контролера з кольоровим кодуванням. Адаптивна верстка на основі flexbox-сітки з медіа-запитами гарантує коректне відображення інтерфейсу на всіх типах пристроїв від мобільних телефонів до великих моніторів, автоматично підлаштовуючи розташування та розміри елементів під доступний простір екрану.

3.4 Програмна реалізація модуля конфігурації автоматичних сценаріїв

Модуль налаштування автоматизації є центральною частиною веб-інтерфейсу для конфігурації сценаріїв роботи пристроїв згідно із заданим розкладом. Функціональність охоплює визначення режимів функціонування, встановлення активних днів тижня та конфігурацію часових інтервалів для кожного пристрою.

Архітектура побудована на бібліотеці React з використанням Material-UI фреймворку для створення адаптивного інтерфейсу. Комунікація з контролером Raspberry Pi реалізована через протокол MQTT, що забезпечує передачу налаштувань у реальному часі та їх збереження у файловій системі у форматі JSON. Далі розглядається структура, механізми обробки даних, процеси

синхронізації та принципи побудови користувацького інтерфейсу для керування параметрами автоматизації.

```
const DAYS_OF_WEEK = [
  { val: 1, label: 'Пн' }, { val: 2, label: 'Вт' }, { val: 3, label: 'Ср' },
  { val: 4, label: 'Чт' }, { val: 5, label: 'Пт' }, { val: 6, label: 'Сб' }, { val: 0, label: 'Нд' }
];
```

Рисунок 3.16 Структура даних для представлення днів тижня

Константа DAYS_OF_WEEK визначає масив об'єктів для представлення днів тижня, де кожен об'єкт містить числове значення дня та його скорочену назву українською мовою. Числові значення відповідають стандарту JavaScript Date, де неділя позначається як 0, а понеділок як 1, що забезпечує коректну синхронізацію з логікою перевірки розкладів на стороні контролера Raspberry Pi. Ця константа використовується для генерації інтерактивних кнопок вибору днів тижня у інтерфейсі налаштування розкладу.

```
const AutomationTab = ({ mqttClient, initialConfig, isLoaded, onUpdateConfig }) => {
    const [localConfig, setLocalConfig] = useState(initialConfig || {});
    const [hasChanges, setHasChanges] = useState(false);
    const [showSuccess, setShowSuccess] = useState(false);
```

Рисунок 3.17 Структура станів компонента

Компонент використовує три основні стани для управління процесом конфігурації автоматизації. Стан localConfig зберігає поточну конфігурацію всіх пристроїв у локальній пам'яті, включаючи налаштування режимів роботи, активних днів тижня та часових інтервалів. Стан hasChanges є булевим прапорцем для відстеження незбережених змін, що дозволяє активувати кнопку збереження лише за наявності модифікацій. Стан showSuccess керує відображенням повідомлення про успішне збереження конфігурації, забезпечуючи візуальний зворотний зв'язок з користувачем. Така структура

станів забезпечує ефективне управління життєвим циклом даних конфігурації та створює інтуїтивний користувацький досвід при роботі з налаштуваннями автоматизації.

```
useEffect(() => {
    if (isLoaded && initialConfig) {
        setLocalConfig(initialConfig);
    }
}, [initialConfig, isLoaded]);
```

Рисунок 3.18 Синхронізація конфігурації через useEffect

Хук useEffect забезпечує автоматичну синхронізацію локального стану компонента з конфігурацією, отриманою від контролера Raspberry Pi через MQTT протокол. Функція виконується при зміні залежностей initialConfig або isLoaded, перевіряючи чи завантажені дані та чи передана початкова конфігурація. Якщо обидві умови виконуються, локальний стан localConfig оновлюється актуальними даними з контролера. Цей механізм є критично важливим для коректної роботи інтерфейсу, оскільки гарантує відображення поточних налаштувань автоматизації при завантаженні сторінки, перезавантаженні компонента або отриманні оновлених даних від системи. Використання масиву залежностей запобігає зайвим перерендерам компонента та забезпечує реактивність інтерфейсу лише при реальних змінах конфігураційних даних.

```
const handleSaveAll = () => {
    mqttClient.publish('automation/save_all', localConfig);
    setHasChanges(false);
    setShowSuccess(true);
};
```

Рисунок 3.19 Обробник зміни конфігурації handleConfigChange

Функція handleConfigChange виконує оновлення параметрів конфігурації для конкретного пристрою при взаємодії користувача з елементами інтерфейсу. Оновлення стану виконується через функціональну форму setState з використанням глибокого копіювання об'єкта через оператор розпакування, що зберігає незмінність інших налаштувань. При кожній зміні автоматично встановлюється прапорець hasChanges у значення true, що активує кнопку збереження та інформує користувача про наявність незбережених модифікацій.

```
const handleSaveAll = () => {
    mqttClient.publish('automation/save_all', localConfig);
    setHasChanges(false);
    setShowSuccess(true);
};
```

Рисунок 3.20 Обробник збереження конфігурації handleSaveAll

Функція handleSaveAll виконує збереження всіх налаштувань автоматизації на контролері та синхронізацію з батьківським компонентом. Спочатку конфігурація публікується через MQTT протокол у топік automation/save_all для збереження у файловій системі контролера. Далі викликається callback-функція onUpdateConfig для оновлення глобального стану, реалізуючи оновлення без очікування підтвердження. Після відправки скидається прапорець hasChanges та активується повідомлення про успішне виконання операції через showSuccess, надаючи користувачу візуальний зворотний зв'язок.

```
if (!isLoaded) {
    return (
        <Box sx={{ display: 'flex', flexDirection: 'column', alignItems: 'center', justifyContent: 'center', minHeight: 300, gap: 2, color: 'text.secondary' }}>
            <CircularProgress size={40} thickness={4} />
            <Typography>Завантаження налаштувань...</Typography>
        </Box>
    );
}
```

Рисунок 3.21 Логіка рендеренгу

Умовний блок перевіряє стан завантаження компонента й забезпечує коректний рендеринг залежно від значення прапорця isLoaded. Якщо дані ще не отримано, інтерфейс повертає спеціальне проміжне представлення, яке інформує користувача про триваючий процес ініціалізації. На екран виводиться вертикально центрований контейнер із компонентом CircularProgress, що слугує візуальним індикатором активного завантаження, а також текстовим повідомленням про отримання налаштувань. Такий підхід дозволяє уникнути рендерингу основного контенту до моменту, поки конфігурація застосунку не буде повністю підготовлена, забезпечуючи користувачу зрозумілий і передбачуваний досвід взаємодії.

```
return (
    <Box sx={{ pb: 10 }}>
        {DEVICES.map(device => {
            const config = localConfig[device.id] || { mode: 'disabled', days: [], start: '09:00', end: '18:00' };
            const isSchedule = config.mode === 'schedule';
            const supportsSensorMode = !['fan', 'window'].includes(device.id);

            return (
                <Card key={device.id} sx={{ mb: 2.5, borderRadius: 3, boxShadow: '0 2px 8px □rgba(0,0,0,0.05)', border: '1px solid □#eee', overflow: 'visible' }}>
                    <CardContent sx={{ p: 2.5, '&:last-child': { pb: 2.5 } }}>
                        <Box sx={{ display: 'flex', alignItems: 'center', mb: 2 }}>
                            <Typography variant="h6" sx={{ fontWeight: 'bold', flexGrow: 1, display: 'flex', alignItems: 'center', gap: 1.5 }}>
                                <span style={{ fontSize: '1.5rem' }}>{device.icon}</span> {device.name}
                            </Typography>
                            <Chip
                                label={isSchedule ? "Розклад" : (config.mode === 'manual' ? "Сенсори" : "Вимкнено")}
                                color={isSchedule ? "primary" : (config.mode === 'manual' ? "success" : "default")}
                                size="small" variant="outlined"
                            />
                        </Box>

                        <FormControl fullWidth sx={{ mb: isSchedule ? 2 : 0 }} size="small" variant="outlined">
                            <InputLabel>Режим роботи</InputLabel>
                            <Select
                                value={config.mode}
                                label="Режим роботи"
                                onChange={(e) => handleConfigChange(device.id, 'mode', e.target.value)}
                                sx={{ borderRadius: 2 }}
                            >
                                <MenuItem value="disabled">🚫 Вимкнено (не реагує в авто-режимі)</MenuItem>
                                {supportsSensorMode && (
                                    <MenuItem value="manual">📡 За даними сенсорів</MenuItem>
                                )}
                                <MenuItem value="schedule">🗓️ За розкладом</MenuItem>
                            </Select>
                        </FormControl>
```

Рисунок 3.22 Генерація карток пристроїв із режимами роботи

Генерація карток для пристроїв реалізується через ітерацію масиву DEVICES методом map, що дозволяє динамічно створювати інтерфейсні блоки для кожного пристрою. Для кожного елемента визначається локальна конфігурація з localConfig; якщо така конфігурація відсутня, застосовується стандартний набір параметрів за замовчуванням з вимкненим режимом, порожнім списком днів та часом роботи з 09:00 до 18:00.

Додатково обчислюються допоміжні змінні isSchedule для визначення активності режиму розкладу та supportsSensorMode для перевірки підтримки сенсорного режиму, який недоступний для вентилятора та жалюзі. Кожна картка створюється через компонент Card з власним стилем (округлені кути, тінь, рамка), а внутрішній вміст організований за допомогою CardContent.

Усередині картки реалізовано вибір режиму роботи через FormControl та Select. Користувач може обрати режим вимкнено, сенсорний режим за даними датчиків або роботу за тижневим розкладом. Підхід забезпечує єдиний інтерфейс для різних типів пристроїв та гнучке управління їх режимами роботи. При обраному режимі schedule додаткові елементи для налаштування днів та часу активності відображаються через компонент Fade, що гарантує плавну анімацію та умовне рендерування контенту.

```
<Fade in={isSchedule} unmountOnExit>
    <Box sx={{ mt: 2, pl: 2, borderLeft: '4px solid #4e54c8', bgcolor: '#f8f9fa', p: 2, borderRadius: 2 }}>
        <Box sx={{ mb: 2 }}>
            <Typography variant="caption" sx={{ display: 'flex', alignItems: 'center', gap: 0.5, color: '#666', mb: 1, fontWeight: 'bold' }}>
                <DayIcon fontSize="inherit" /> АКТИВНІ ДНІ
            </Typography>
            <ToggleButtonGroup
                value={config.days}
                onChange={(e, newDays) => handleConfigChange(device.id, 'days', newDays)}
                size="small"
                sx={{ display: 'flex', justifyContent: 'space-between', width: '100%', bgcolor: 'white', borderRadius: 50 }}
                color="primary"
            >
                {DAYS_OF_WEEK.map(day => (
                    <ToggleButton
                        key={day.val}
                        value={day.val}
                        sx={{
                            flexGrow: 1, borderRadius: '50% !important', border: 'none', mx: 0.5, width: 36, height: 36, p: 0,
                            '&.Mui-selected': { color: 'white', bgcolor: 'primary.main', '&:hover': { bgcolor: 'primary.dark' } }
                        }}
                    >
                        {day.label}
                    </ToggleButton>
                ))}
            </ToggleButtonGroup>
        </Box>
```

Рисунок 3.23 Реалізація відображення налаштувань розкладу

Відображення налаштувань розкладу реалізовано умовно за допомогою компонента Fade, який активується лише тоді, коли вибрано режим schedule. Усередині блоку розташовано заголовок з роздільником та чіпом для наочності, що позначає секцію налаштування розкладу.

Дні тижня представлені через компонент ToggleButtonGroup, який дозволяє вибрати декілька активних днів. Кожен день відповідає кнопці ToggleButton, позначеної короткою назвою дня, що підвищує зручність взаємодії користувача. Час початку та завершення роботи пристрою задається через два поля введення TextField з типом time. Вони синхронізовані з локальною конфігурацією через обробник handleConfigChange, що забезпечує актуалізацію даних та можливість відстеження змін.

Такий підхід дозволяє гнучко налаштовувати активність пристроїв за днями та часом, забезпечує умовне відображення елементів тільки при вибраному режимі розкладу і створює інтуїтивно зрозумілий інтерфейс для користувача.

```
<Fade in={hasChanges}>
    <Paper elevation={6} sx={{ position: 'fixed', bottom: 24, left: '50%', transform: 'translateX(-50%)', zIndex: 1000, width: 'auto', borderRadius: 50, p: 0.5, bgcolor: 'white' }}>
        <Button
            variant="contained" color="primary" size="large" startIcon={<SaveIcon />} onClick={handleSaveAll}
            sx={{ borderRadius: 50, px: 4, py: 1.5, fontSize: '1rem', fontWeight: 'bold', textTransform: 'none', boxShadow: 'none' }}
        >
            Зберегти зміни
        </Button>
    </Paper>
</Fade>
```

Рисунок 3.24 Реалізація кнопки збереження налаштувань

Плаваюча кнопка збереження реалізована через компонент Paper, обгорнутий у Fade, що умовно відображається лише тоді, коли є незбережені зміни (hasChanges). Розташування кнопки фіксоване в нижній частині екрану по центру, що забезпечує постійну доступність для користувача під час прокручування сторінки.

Кнопка представлена компонентом Button з іконкою SaveIcon, великим шрифтом та округленими краями, що робить її помітною та інтуїтивно зрозумілою для взаємодії. Натискання кнопки викликає функцію handleSaveAll, яка відправляє локальні зміни на сервер через MQTT, оновлює стан конфігурації в додатку та відображає повідомлення про успішне збереження.

```
    <Snackbar open={showSuccess} autoHideDuration={3000} onClose={() => setShowSuccess(false)} anchorOrigin={{ vertical: 'bottom', horizontal: 'center' }}>
        <Alert severity="success" variant="filled" sx={{ width: '100%', boxShadow: 4, borderRadius: 2 }}>Налаштування успішно збережено!</Alert>
    </Snackbar>
</Box>
```

Рисунок 3.25 Інформування користувача про успішне збереження

Повідомлення про успішне збереження налаштувань реалізовано за допомогою компонента Snackbar, всередині якого розташовано Alert з типом success. Компонент відкривається при зміні стану showSuccess на true і автоматично приховується через 3000 мс або при натисканні на кнопку закриття. Повідомлення розташоване в нижній центральній частині екрану (anchorOrigin) і має ширину 100%, що забезпечує хорошу видимість без перекриття основного контенту. Використання компонента Alert з варіантом filled і зеленим кольором підкреслює успішний результат дії користувача.

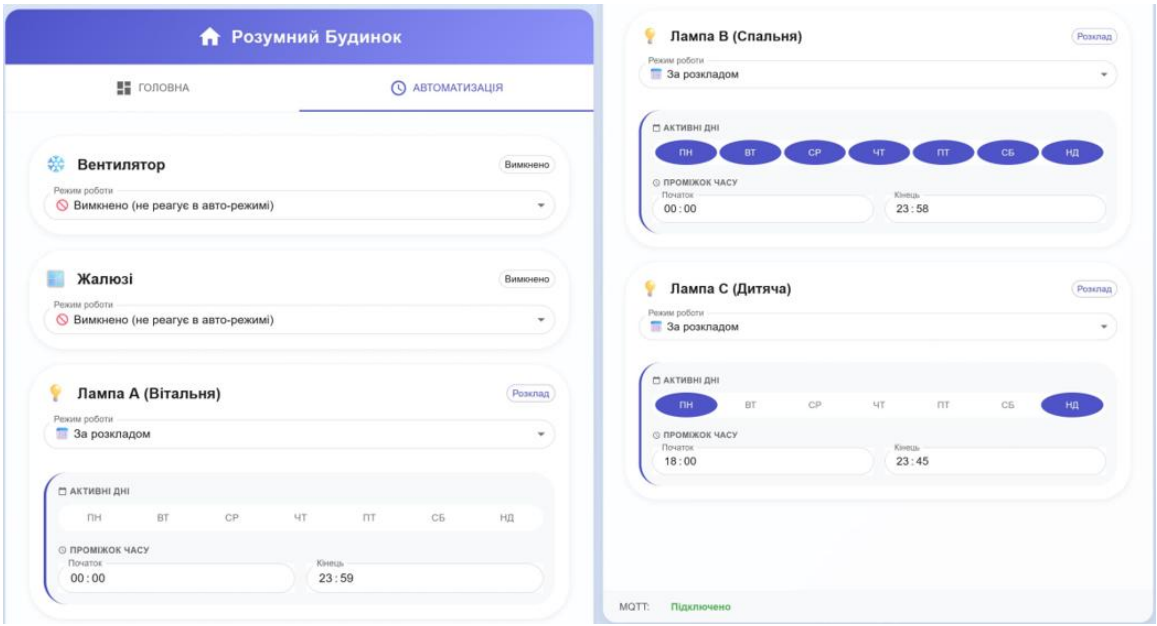

Рисунок 3.26 Інформування користувача про успішне збереження

Інтерфейс вкладки автоматизації дозволяє користувачу ефективно керувати режимами роботи всіх пристроїв у системі. Після завантаження даних відображається список карток для кожного пристрою, де користувач бачить назву, іконку та актуальні налаштування. Для кожного пристрою можна обрати режим роботи в автоматичному режимі: вимкнено, сенсорний режим або за тижневим розкладом.

При виборі режиму schedule на картці з’являються додаткові елементи для налаштування розкладу, включно з вибором днів тижня та часу початку і завершення роботи. Дні активності представлені у вигляді кнопок-перемикачів, що дозволяє швидко обирати необхідні дні, а час роботи задається через поля введення типу time. Усі ці елементи відображаються умовно та анімовано, що підвищує зручність користування та чистоту інтерфейсу.

Інтерфейс реалізовано у сучасному стилі з акцентом на інтуїтивність і зручність використання. Картки пристроїв мають округлі краї, та акуратні роздільники, що робить їх візуально приємними. Використання умовного рендерингу та анімацій дозволяє відображати тільки актуальні елементи інтерфейсу, не перевантажуючи користувача зайвою інформацією. В результаті користувач отримує зрозумілу і логічну панель управління автоматизацією, де легко контролювати роботу всіх пристроїв і швидко вносити необхідні зміни.

ВИСНОВКИ

Тема розробки систем «Розумного будинку» є надзвичайно актуальною у сучасних умовах, оскільки автоматизація житлових приміщень дозволяє підвищити рівень комфорту, енергоефективності та безпеки, а також створює можливість інтегрувати різні технології та пристрої в єдину керовану систему. З розвитком інтернету речей (IoT) та доступності недорогого обладнання, такого як мікроконтролери та сенсорні пристрої, з’являється необхідність у гнучких та зручних інструментах для централізованого управління будинком.

У ході виконання роботи проведено комплексний аналіз технологій і платформ «Розумного будинку», оцінено їх переваги та недоліки. Виявлено основні проблеми сучасних систем автоматизації, серед яких складність інтеграції різнорідних пристроїв, обмежені можливості для налаштування індивідуальних сценаріїв та недостатній рівень візуалізації стану системи для користувача. На основі цього проведено проектування системи керування, що забезпечує централізований контроль освітлення, клімату, безпеки та інших житлових параметрів у реальному часі.

Розроблена система передбачає обмін даними через протокол MQTT, що дозволяє ефективно синхронізувати стан всіх підключених пристроїв та забезпечує швидкий і надійний моніторинг. Програмна реалізація передбачає модульну структуру управління, гнучку конфігурацію автоматичних сценаріїв та умовне відображення стану окремих пристроїв. Це дозволяє користувачу оперативно реагувати на події, змінювати налаштування та оптимізувати енергоспоживання у приміщенні.

Розроблений веб-інтерфейс забезпечує інтуїтивну взаємодію користувача з системою, включаючи відображення поточного стану пристроїв, керування режимами роботи, а також швидке підтвердження змін через візуальні сповіщення. Такий підхід забезпечує прозорість роботи системи та підвищує

рівень зручності й безпеки для користувача. Впровадження автоматичних сценаріїв та можливостей налаштування розкладів дозволяє адаптувати роботу системи під індивідуальні потреби мешканців і підвищує ефективність використання енергоресурсів.

Загалом, виконана робота підтверджує, що сучасні системи «Розумного будинку» можуть бути доступними, гнучкими та масштабованими. Реалізована система демонструє практичну ефективність автоматизації житлового простору, підвищує комфорт та безпеку, а також забезпечує можливість інтеграції різноманітних пристроїв у єдину керовану платформу.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Рябчун Ю. В., Середа Д. Е., Кохан В. Р., Доля О. В. Можливості та переваги українського ринку технологій «Розумний будинок» : / Ю. В. Рябчун, Д. Е. Середа, В. Р. Кохан, О. В. Доля. – 2023. – 7 с.
2. Науково-технічна конференція «Сучасні інфокомунікаційні технології» : зб. тез / К.ДУТ. – 2019. – 254 с.
3. Микитин Г. В., Дудикевич В. Б., Ребець А. І., Мельник М. В. Безпровідні сенсорні мережі ZigBee, Wi-Fi та Bluetooth в кіберфізичних системах: концепція «об'єкт – загроза – захист» на основі моделі OSI // Системи обробки інформації. – 2019. – С. 114–120.
4. Riddeway J. Smart Home Automation with Z-Wave : / J. Riddeway. – Packt Publishing, 2022. – 193 с.
5. Жураковський Б. Ю., Зенів І. О. Технології Інтернету речей : навч. посіб. – Київ, 2021. – 56 с.
6. Руденко В. В. Огляд протоколу LoRa та перспективи використання // Control, Navigation and Communication Systems. – 2025. – № 2. – С. 244–248
7. oXorona.com. Xiaomi SmartHome : веб-сторінка. URL: https://oxorona.com/xiaomi-smarthome/ (дата звернення: 11.04.2025).
8. Z-Wave Україна. Broadlink RM mini 3 : веб-сторінка. URL: https://z-wave.com.ua/ua/p1094491869-universalnyj-pult-broadlink.html (дата звернення: 11.04.2025)
9. Fibaro: система автоматизацій будівель fibaro. URL: http://www.fibaro.com/ (дата звернення 11.04.2025).
10. Ajax Systems. StarterKit Cam Plus : веб-сторінка. URL: https://ajax.systems/ua/products/starterkit-cam-plus/ (дата звернення: 11.04.2025).

11. Worldvision.com.ua. Комплект для «умного дому» Orvibo Security Kit: веб-сторінка. URL: https://worldvision.com.ua/komplekt-dlja-umnogo-doma-orvibo-security-kit/ (дата звернення: 11.04.2025).

12. Жураковський Б. Ю., Зенів І. О. Технології Інтернету речей : навч. посіб. – Київ, 2021. – 56 с.

13. Альтернативні джерела енергії та технології їх використання: підруч. / В. В. Клименко, В. П. Солдатенко, С. П. Плєшков, О. В. Скрипник, А. І. Саченко; за ред. д-ра техн. наук, проф. В. В. Клименка. – Кропивницький : ПП Ексклюзив-Систем, 2023. – 268с.

14. RFID модуль RC522 datasheet. URL: https:/www.nxp.com/docs/en/data-sheet MFRC522.pdf (дата звернення:18.10.2025).

15. Сучасні інформаційні технології, засоби автоматизації та електропривод : матеріали VIII Всеукр. наук.-практ. конф., 18–20 квіт. 2024 р. / за заг. ред. О. Ф. Тарасова. – Краматорськ–Тернопіль : ДДМА, 2024. – 235 с.

16. Інтегровані інформаційні системи : конспект лекцій / укладач: О. В. Бойко. – Суми : Сумський державний університет, 2023. – 130 с.

17. Технології інтернету речей. Навчальний посібник: навч. посіб. для студ. спеціальності 126 «Інформаційні системи та технології», спеціалізація «Інформаційне забезпечення робототехнічних систем» / Б. Ю. Жураковський, І.О. Зенів; КПІ ім. Ігоря Сікорського. – Електронні текстові дані (1 файл: 12,5 Мбайт). – Київ: КПІ ім. Ігоря Сікорського, 2021. – 271 с.

18. WonderfulPCB. Pull-up and Pull-down Resistors: Function, Application, Selection : веб-сторінка. URL: https://www.wonderfulpcb.com/uk/blog/pull-up-and-pull-down-resistors-function-application-selection/ (дата звернення: 22.10.2025).

19. Кошмак Є. С., Поліщук І. А. Використання MQTT протоколу. Принцип роботи та налаштування // Молодий вчений. – 2018. – № 5 (57), травень. – С. 34–39. – Нац. техн. ун-т України «Київ. політехн. ін-т ім. Ігоря Сікорського».

20. IT-notes. Callbacks basics in JavaScript : веб-сторінка. URL: https://www.it-notes.wiki/javascript/callbacks-basics-in-javascript/ (дата звернення: 23.10.2025).

21. Технології сучасних кібер-фізичних систем: Навчальний посібник: навч. посіб. для студ. спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології», освітньо-професійна програма «Автоматизація та комп'ютерно-інтегровані технології кібер-енергетичних систем»; укладач: Ю.Є. Грудзинський. – Київ : КПІ ім. Ігоря Сікорського, 2020. – 327 с.

22. Технічні засоби Інтернету речей: навч. посіб. для студ. спеціальності 171 «Електроніка», спеціалізації «Електронні системи мультимедіа та засоби Інтернету речей» / КПІ ім. Ігоря Сікорського; уклад.: Ю.О.Оникієнко, О.О. Титаренко. – Електронні текстові данні – Київ : КПІ ім. Ігоря Сікорського, 2020. –124 с.

23. JavaScript.info. WebSocket : веб-сторінка. URL: https://uk.javascript.info/websocket (дата звернення: 25.10.2025).

24. Комп'ютерно-інтегровані технології: освіта, наука, виробництво : наук. журн. – Луцьк, 2021. – Вип. 42. – 6 с.

25. React. Describing the UI : веб-сторінка. URL: https://react.dev/learn/describing-the-ui (дата звернення: 15.11.2025)

26. Тронь В. В., Маринич І. А. Методичні вказівки до виконання магістерської кваліфікаційної роботи для студентів спеціальності 174 Автоматизація, комп'ютерно-інтегровані технології та робототехніка". Кривий Ріг: Видавничий центр КНУ, 2022. 50 с.

27. ДСТУ 3008:2015. Інформація та документація. Звіти у сфері науки і техніки. Структура і правила оформлення. Київ, ДП «УкрННЦ», 2015. 26с.(Інформація та документація).

28. ДСТУ 8302:2015. Бібліографічне посилання. Загальні вимоги та правила складання Київ, ДП «УкрННЦ», 2016. 16 с.(Інформація та документація).

29. ДСТУ 3582:2013. Бібліографічний опис. Скорочення слів і словосполучень в українській мові. Загальні вимоги та правила. Київ, ДП «УкрННЦ», 2013. 23 с.(Інформація та документація).

30. ДСТУ 3651.0-97 Метрологія. Одиниці фізичних величин. Основні одиниці фізичних величин Міжнародної системи одиниць. Основні положення, назви та позначення Київ, Держстандарт України, 1998. 27 с.(Інформація та документація).

ДОДАТОК А

Лістинг програми керування розумним будинком

```
import RPi.GPIO as GPIO
import paho.mqtt.client as mqtt
from mfrc522 import SimpleMFRC522
import json
import time
import threading
import os
from datetime import datetime

MQTT_BROKER = "localhost"
MQTT_PORT = 1883
MQTT_USER = "mqtt_user"
MQTT_PASSWORD = "admin"
TOPIC_PREFIX = "home"
CONFIG_FILE = "config.json"

PINS = {
    'button': 4,
    'fan': 27,
    'light_sensor': 7,
    'win_forward': 23,
    'win_back': 24,
    'led_red': 14,
    'led_blue': 13,
    'buzzer': 18,
    'lamps': [6, 5, 26],
```

```
    'motion': [17, 22, 16]
}

TAG_ID_AUTHORIZED = 631769472242

class SmartHomeRPI:
    def __init__(self):
        self.lock = threading.Lock()
        self.running = True

        self.states = {
            'auto_mode': True,
            'security': False,
            'fan': False,
            'window': False,
            'lamps': [False, False, False]
        }

        self.load_config()
        self.setup_gpio()
        self.mqtt_setup()
        self.rfid_reader = SimpleMFRC522()

    def load_config(self):
        default_config = {
            'fan': {'mode': 'disabled', 'days': [], 'start': '09:00', 'end': '18:00'},
            'window': {'mode': 'disabled', 'days': [], 'start': '07:00', 'end': '21:00'}
        }
        if os.path.exists(CONFIG_FILE):
            try:
```

```
            with open(CONFIG_FILE, 'r') as f:
                self.automation_config = json.load(f)
            for key, val in default_config.items():
                if key not in self.automation_config:
                    self.automation_config[key] = val
        except:
            self.automation_config = default_config
    else:
        self.automation_config = default_config

def save_config(self):
    try:
        with self.lock:
            with open(CONFIG_FILE, 'w') as f:
                json.dump(self.automation_config, f, indent=4)
    except Exception as e:
        print(f"Error saving config: {e}")

def setup_gpio(self):
    GPIO.setwarnings(False)
    GPIO.setmode(GPIO.BCM)

    for pin in PINS['lamps']:
        GPIO.setup(pin, GPIO.OUT, initial=GPIO.HIGH)

    for pin in [PINS['fan'], PINS['win_forward'], PINS['win_back'], PINS['led_red'], PINS['led_blue'], PINS['buzzer']]:
        GPIO.setup(pin, GPIO.OUT, initial=GPIO.LOW)

    for pin in PINS['motion'] + [PINS['light_sensor'], PINS['button']]:
```

```
            GPIO.setup(pin, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)

        GPIO.output(PINS['led_blue'], GPIO.HIGH)

        GPIO.add_event_detect(PINS['button'], GPIO.FALLING, callback=self.handle_manual_button, bouncetime=300)

    def mqtt_setup(self):
        self.client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION2)
        self.client.username_pw_set(MQTT_USER, MQTT_PASSWORD)
        self.client.on_connect = self.on_connect
        self.client.on_message = self.on_message

        self.client.will_set(f"{TOPIC_PREFIX}/simulator/status", json.dumps({"state": "offline"}), retain=True, qos=1)

    def on_connect(self, client, userdata, flags, rc, properties=None):
        if rc == 0:
            print("Connected to MQTT!")
            client.subscribe(f"{TOPIC_PREFIX}/+/set")
            client.subscribe(f"{TOPIC_PREFIX}/automation/+/set")
            client.subscribe(f"{TOPIC_PREFIX}/automation/get_config/set")
            client.subscribe(f"{TOPIC_PREFIX}/automation/save_all/set")
            client.subscribe(f"{TOPIC_PREFIX}/all/get_status/set")

            self.publish_state("simulator/status", "online", retain=True)
            self.sync_all_states()
            self.publish_config()

    def on_message(self, client, userdata, msg):
```

```
try:
    topic = msg.topic
    payload = json.loads(msg.payload.decode())

    if topic.endswith("all/get_status/set"):
        self.sync_all_states()
        return

    if topic.endswith("automation/get_config/set"):
        self.publish_config()
        return

    if topic.endswith("automation/save_all/set"):
        new_config = payload.get("state", payload)
        with self.lock:
            self.automation_config = new_config
        self.save_config()
        self.publish_config()
        return

    device = topic.split('/')[1]
    cmd_state = payload.get("state")

    with self.lock:
        if device == 'mode':
            self.states['auto_mode'] = cmd_state
            self.publish_state('mode', cmd_state, retain=True)

        elif device == 'security':
            self.set_security(cmd_state)
```

```
            elif not self.states['auto_mode']:
                if device == 'fan':
                    self.set_fan(cmd_state)
                elif device == 'window':
                    self.control_window(cmd_state)
                elif device.startswith('lamp'):
                    idx = int(device[4:]) - 1
                    if 0 <= idx < 3:
                        self.set_lamp(idx, cmd_state)
            else:
                print(f"Manual command for {device} ignored: Auto Mode is ON")

        except Exception as e:
            print(f"Msg error: {e}")

    def publish_state(self, subtopic, state, retain=False):
        full_topic = f"{TOPIC_PREFIX}/{subtopic}" if subtopic.endswith(('state', 'status')) else f"{TOPIC_PREFIX}/{subtopic}/state"
        payload = json.dumps(state) if isinstance(state, dict) else json.dumps({"state": state})
        self.client.publish(full_topic, payload, retain=retain, qos=1)

    def set_fan(self, state):
        GPIO.output(PINS['fan'], GPIO.HIGH if state else GPIO.LOW)
        self.states['fan'] = state
        self.publish_state('fan', state, retain=True)

    def set_lamp(self, index, state):
        if self.states['lamps'][index] != state:
            GPIO.output(PINS['lamps'][index], GPIO.LOW if state else GPIO.HIGH)
```

```
        self.states['lamps'][index] = state
        self.publish_state(f'lamp{index+1}', state, retain=True)

    def set_security(self, state):
        self.states['security'] = state
        GPIO.output(PINS['led_red'], GPIO.HIGH if state else GPIO.LOW)
        self.publish_state('security', state, retain=True)
        if not state:
            self.beep(0.1)

    def control_window(self, open_cmd):
        if open_cmd == self.states['window']:
            return

        GPIO.output(PINS['win_forward'], GPIO.HIGH if open_cmd else GPIO.LOW)
        GPIO.output(PINS['win_back'], GPIO.LOW if open_cmd else GPIO.HIGH)

        time.sleep(1.0)

        GPIO.output(PINS['win_forward'], GPIO.LOW)
        GPIO.output(PINS['win_back'], GPIO.LOW)

        self.states['window'] = open_cmd
        self.publish_state('window', open_cmd, retain=True)

    def beep(self, duration=1, count=1):
        for _ in range(count):
            GPIO.output(PINS['buzzer'], GPIO.HIGH)
            time.sleep(duration)
            GPIO.output(PINS['buzzer'], GPIO.LOW)
```

```
        time.sleep(0.1)

    def handle_manual_button(self, channel):
        with self.lock:
            self.set_fan(not self.states['fan'])

    def check_schedule(self, config):
        if config['mode'] != 'schedule':
            return False
        now = datetime.now()
        current_js_day = (now.weekday() + 1) % 7
        if current_js_day not in config['days']:
            return False

        current_time = now.strftime("%H:%M")
        if config['start'] <= config['end']:
            return config['start'] <= current_time < config['end']
        else:
            return config['start'] <= current_time or current_time < config['end']

    def loop_sensors_and_auto(self):
        while self.running:
            with self.lock:
                global_auto = self.states['auto_mode']
                security_on = self.states['security']
                configs = self.automation_config.copy()

            if security_on:
                if any(GPIO.input(pin) == GPIO.HIGH for pin in PINS['motion']):
                    print("ALARM!")
```

```
            self.beep(0.5)

    if global_auto:
        for i in range(3):
            lamp_id = f'lamp{i+1}'
            cfg = configs.get(lamp_id, {'mode': 'disabled'})
            should_be_on = self.states['lamps'][i]

            if cfg['mode'] == 'schedule':
                should_be_on = self.check_schedule(cfg)
            elif cfg['mode'] == 'manual':
                should_be_on = GPIO.input(PINS['motion'][i]) == GPIO.HIGH

            if cfg['mode'] != 'disabled' and self.states['lamps'][i] != should_be_on:
                self.set_lamp(i, should_be_on)

        fan_cfg = configs.get('fan', {'mode': 'disabled'})
        if fan_cfg['mode'] == 'schedule':
            if self.states['fan'] != self.check_schedule(fan_cfg):
                self.set_fan(self.check_schedule(fan_cfg))

        win_cfg = configs.get('window', {'mode': 'disabled'})
        if win_cfg['mode'] == 'schedule':
            should_open = self.check_schedule(win_cfg)
            if self.states['window'] != should_open:
                self.control_window(should_open)
        elif win_cfg['mode'] == 'manual':
            should_open = GPIO.input(PINS['light_sensor']) == 1
            if self.states['window'] != should_open:
                self.control_window(should_open)
```

```
            time.sleep(1.0)

    def loop_rfid(self):
        while self.running:
            try:
                id, text = self.rfid_reader.read_no_block()
                if id:
                    if str(id).strip() == str(TAG_ID_AUTHORIZED).strip():
                        with self.lock:
                            self.set_security(not self.states['security'])
                        time.sleep(2.0)
                    else:
                        self.beep(0.1, 3)
                        time.sleep(1.0)
            except:
                pass
            time.sleep(0.1)

    def sync_all_states(self):
        for key, value in self.states.items():
            if key == "lamps":
                for i, v in enumerate(value):
                    self.publish_state(f"lamp{i+1}", v, retain=True)
            else:
                self.publish_state(key, value, retain=True)

    def publish_config(self):
        self.client.publish(f"{TOPIC_PREFIX}/automation/config",
json.dumps(self.automation_config), qos=1, retain=True)
```

```
    def start(self):
        try:
            self.client.connect(MQTT_BROKER, MQTT_PORT, 60)
            self.client.loop_start()

            threading.Thread(target=self.loop_sensors_and_auto, daemon=True).start()
            threading.Thread(target=self.loop_rfid, daemon=True).start()

            print("SYSTEM STARTED")
            while True:
                time.sleep(1)
        except KeyboardInterrupt:
            pass
        finally:
            self.running = False
            self.publish_state("simulator/status", "offline", retain=True)
            time.sleep(1)
            self.client.loop_stop()
            GPIO.cleanup()

if __name__ == '__main__':
    SmartHomeRPI().start()
```

ДОДАТОК Б
Реалізація MQTT клієнта для веб-інтерфейсу

```
import mqtt from 'mqtt';

export const MQTT_CONFIG = {
  broker: window.location.hostname === 'localhost' ? 'localhost' : window.location.hostname,
  port: 9001,
  username: 'mqtt_user',
  password: 'admin',
  topicPrefix: 'home/'
};

class MqttClient {
  constructor() {
    this.client = null;
    this.callbacks = new Map();
    this.connectionCallback = null;
    this._isConnected = false;
    this.connectUrl = `ws://${MQTT_CONFIG.broker}:${MQTT_CONFIG.port}`;
    this.options = {
      username: MQTT_CONFIG.username,
      password: MQTT_CONFIG.password,
      clean: true,
      connectTimeout: 4000,
      reconnectPeriod: 5000,
      protocolVersion: 4
    };

    this.handleConnect = this.handleConnect.bind(this);
    this.handleError = this.handleError.bind(this);
```

```
  this.handleClose = this.handleClose.bind(this);
}

onConnectionStateChange(callback) {
  this.connectionCallback = callback;
}

connect() {
  if (this.client && (this.client.connected || this.client.connecting)) return;

  console.log(`Frontend: Connecting to ${this.connectUrl}`);
  this.client = mqtt.connect(this.connectUrl, this.options);

  this.client.on('connect', this.handleConnect);
  this.client.on('message', (topic, message) => {
    try {
      const payload = JSON.parse(message.toString());
      if (this.callbacks.has(topic)) {
        this.callbacks.get(topic)(payload);
      }
    } catch (e) {
      console.error("Frontend: MQTT message parse error:", e);
    }
  });
  this.client.on('error', this.handleError);
  this.client.on('close', this.handleClose);
}

handleConnect() {
  console.log('Frontend: Connected');
  this._isConnected = true;
```

```
  if (this.connectionCallback) this.connectionCallback(true);
  this.callbacks.forEach((_, topic) => {
    this.client.subscribe(topic, {
      qos: 0
    });
  });
}

handleError(err) {
  console.error('Frontend: MQTT Error:', err);
  this._isConnected = false;
  if (this.connectionCallback) this.connectionCallback(false);
}

handleClose() {
  console.log('Frontend: Connection closed');
  this._isConnected = false;
  if (this.connectionCallback) this.connectionCallback(false);
}

publish(topic_suffix, state_or_payload) {
  if (!this.client || !this._isConnected) {
    console.warn("MQTT client not connected, publish canceled.");
    return;
  }

  let fullTopic;
  if (topic_suffix.startsWith(MQTT_CONFIG.topicPrefix)) {
      fullTopic = topic_suffix;
  } else {
      fullTopic = `${MQTT_CONFIG.topicPrefix}${topic_suffix}/set`;
```

```
    }
    const isSpecialPayload = topic_suffix.includes('automation/save_all') ||
topic_suffix.includes('all/get_status') || topic_suffix.includes('automation/get_config');

    const payload = (typeof state_or_payload === 'object' && isSpecialPayload)
      ? JSON.stringify(state_or_payload)
      : JSON.stringify({ state: state_or_payload });

    console.log(`Frontend: Publishing to ${fullTopic} with payload: ${payload}`);
    this.client.publish(fullTopic, payload, { qos: 0 });
  }

  subscribe(deviceOrTopic, callback) {
    let fullTopic;
    if (deviceOrTopic && deviceOrTopic.startsWith(MQTT_CONFIG.topicPrefix)) {
      fullTopic = deviceOrTopic;
    } else if (deviceOrTopic) {
      fullTopic = `${MQTT_CONFIG.topicPrefix}${deviceOrTopic}/state`;
    } else return;

    this.callbacks.set(fullTopic, callback);
    if (this.client && this._isConnected) {
      this.client.subscribe(fullTopic, {
        qos: 0
      });
    }
  }

  unsubscribe(deviceOrTopic) {
    let fullTopic;
    if (deviceOrTopic && deviceOrTopic.startsWith(MQTT_CONFIG.topicPrefix)) {
      fullTopic = deviceOrTopic;
```

```
    } else if (deviceOrTopic) {
      fullTopic = `${MQTT_CONFIG.topicPrefix}${deviceOrTopic}/state`;
    } else return;

    this.callbacks.delete(fullTopic);
    if (this.client && this._isConnected) {
      this.client.unsubscribe(fullTopic);
    }
  }
}

const mqttClientInstance = new MqttClient();
export default mqttClientInstance;
```

ДОДАТОК В

Програмний код компонента DeviceSwitch для керування пристроями

```
import React, { useState, useEffect } from 'react';
import { Switch, FormControlLabel, Typography, Box } from '@mui/material';
import mqttClient from '../mqtt/MqttClient';

const DeviceSwitch = ({ device, label, disabled, icon }) => {
  const [isChecked, setIsChecked] = useState(false);

  useEffect(() => {
    const handleStateChange = (payload) => {
      if (payload && payload.state !== undefined) {
        setIsChecked(payload.state);
      }
    };
    mqttClient.subscribe(device, handleStateChange);
    return () => {
      mqttClient.unsubscribe(device);
    };
  }, [device]);

  const handleChange = (event) => {
    if (disabled) return;
    const newState = event.target.checked;
    setIsChecked(newState);
    mqttClient.publish(device, newState);
  };

  return (
```

```
    <Box sx={{ display: 'flex', justifyContent: 'space-between', alignItems: 'center', width: '100%' }}>
      <Box sx={{ display: 'flex', alignItems: 'center', gap: 1.5 }}>
        <span style={{ fontSize: '1.5rem' }}>{icon}</span>
        <Typography>{label}</Typography>
      </Box>
      <Switch
        checked={isChecked}
        onChange={handleChange}
        disabled={disabled}
        color="primary"
      />
    </Box>
  );
};

export default DeviceSwitch;
```

ДОДАТОК Г

Програмний код головного компонента веб-інтерфейсу

```
import React, { useState, useEffect, useRef } from 'react';
import {
  Box, Card, CardContent, Typography, Switch, FormControlLabel,
  ThemeProvider, createTheme, Fade, Tab, Tabs, Button, CssBaseline, Paper
} from '@mui/material';
import HomeIcon from '@mui/icons-material/Home';
import SettingsRemoteIcon from '@mui/icons-material/SettingsRemote';
import SecurityIcon from '@mui/icons-material/Security';
import ScheduleIcon from '@mui/icons-material/Schedule';
import DashboardIcon from '@mui/icons-material/Dashboard';
import DeviceSwitch from './components/DeviceSwitch';
import AutomationTab from './components/AutomationTab';
import mqttClient, { MQTT_CONFIG } from './mqtt/MqttClient';
import './App.css';

const theme = createTheme({
  palette: {
    primary: { main: '#4e54c8' },
    secondary: { main: '#ff6b6b' },
    background: { default: '#f0f2f5' }
  },
  typography: {
    fontFamily: '"Roboto", "Helvetica", "Arial", sans-serif',
    h5: { fontWeight: 600 },
  },
  shape: { borderRadius: 16 },
```

```
});

const SIMULATOR_STATUS_TOPIC = `${MQTT_CONFIG.topicPrefix}simulator/status`;
const AUTOMATION_CONFIG_TOPIC = `${MQTT_CONFIG.topicPrefix}automation/config`;
const HEARTBEAT_TIMEOUT = 30000;

function App() {
    const [currentTab, setCurrentTab] = useState(0);
    const [isAuthenticated, setIsAuthenticated] = useState(false);
    const [passwordInput, setPasswordInput] = useState('');
    const [autoMode, setAutoMode] = useState(true);
    const [securitySystem, setSecuritySystem] = useState(false);
    const [isConnected, setIsConnected] = useState(false);
    const [simulatorStatus, setSimulatorStatus] = useState('Невідомо');
    const [automationConfig, setAutomationConfig] = useState({});
    const [isConfigLoaded, setIsConfigLoaded] = useState(false);
    const heartbeatTimeout = useRef(null);

    useEffect(() => {
        const auth = localStorage.getItem('isAuthenticated');
        if (auth === 'true') setIsAuthenticated(true);

        mqttClient.onConnectionStateChange((connected) => {
            setIsConnected(connected);
            if (connected) {
                console.log("Connected! Requesting full sync (states + config)...");
                mqttClient.publish('all/get_status', {});
                mqttClient.publish('automation/get_config', {});
            }
        });
```

```
        mqttClient.connect();

        mqttClient.subscribe(SIMULATOR_STATUS_TOPIC, (payload) => {
          const statusValue = payload && payload.state !== undefined ? payload.state : null;
          if (statusValue) {
            const newStatus = statusValue === 'online' ? 'Онлайн' : 'Офлайн';
            setSimulatorStatus(newStatus);
            clearTimeout(heartbeatTimeout.current);
            if (newStatus === 'Онлайн') {
              heartbeatTimeout.current = setTimeout(() => {
                setSimulatorStatus('Офлайн');
              }, HEARTBEAT_TIMEOUT);
            }
          }
        });

        mqttClient.subscribe('mode', (p) => { if(p && p.state !== undefined) setAutoMode(p.state) });
        mqttClient.subscribe('security', (p) => { if(p && p.state !== undefined)
setSecuritySystem(p.state) });

        mqttClient.subscribe(AUTOMATION_CONFIG_TOPIC, (payload) => {
          console.log("Automation config received:", payload);
          const configData = payload.state !== undefined ? payload.state : payload;
          setAutomationConfig(configData);
          setIsConfigLoaded(true);
        });

        heartbeatTimeout.current = setTimeout(() => {
           if (simulatorStatus === 'Невідомо') setSimulatorStatus('Офлайн');
```

```
    }, HEARTBEAT_TIMEOUT);

    return () => clearTimeout(heartbeatTimeout.current);
  }, []);

  const handleTabChange = (event, newValue) => setCurrentTab(newValue);
  const handleAutoModeChange = (event) => {
    setAutoMode(event.target.checked);
    mqttClient.publish('mode', event.target.checked);
  };
  const handleSecuritySystemChange = (event) => {
    setSecuritySystem(event.target.checked);
    mqttClient.publish('security', event.target.checked);
  };
  const handleLogin = (e) => {
    e.preventDefault();
    if (passwordInput === 'admin') {
      setIsAuthenticated(true);
      localStorage.setItem('isAuthenticated', 'true');
    } else alert('Невірний пароль!');
  };

  const mqttStatusColor = isConnected ? '#4caf50' : '#f44336';
  const simStatusColor = simulatorStatus === 'Онлайн' ? '#4caf50' : (simulatorStatus === 'Офлайн'
? '#f44336' : 'gray');

  if (!isAuthenticated) {
    return (
      <ThemeProvider theme={theme}>
        <CssBaseline />
```

```
          <Box sx={{ display: 'flex', justifyContent: 'center', alignItems: 'center', height: '100vh', bgcolor: 'background.default' }}>
            <Card sx={{ minWidth: 300, p: 3, boxShadow: 3, borderRadius: 4 }}>
              <Typography variant="h5" textAlign="center" mb={3} color="primary" fontWeight="bold">Вхід в Систему</Typography>
              <form onSubmit={handleLogin}>
                <input type="password" placeholder="Пароль" value={passwordInput} onChange={(e) => setPasswordInput(e.target.value)}
                  style={{ width: '100%', padding: '12px', marginBottom: '20px', borderRadius: '8px', border: '1px solid #ddd', fontSize: '16px' }} />
                <Button type="submit" variant="contained" fullWidth size="large" sx={{ borderRadius: 2, textTransform: 'none', fontSize: '1rem' }}>Увійти</Button>
              </form>
            </Card>
          </Box>
        </ThemeProvider>
      );
    }

    return (
      <ThemeProvider theme={theme}>
        <CssBaseline />
        <div className="app-container">
          <Fade in={true} timeout={800}>
            <Card className="main-card" sx={{ maxWidth: 800, mx: 'auto', mt: { xs: 2, sm: 4 }, boxShadow: 6, borderRadius: 4, overflow: 'hidden' }}>
              <div className="card-header" style={{ background: 'linear-gradient(135deg, #4e54c8, #8f94fb)', color: 'white', padding: '20px', textAlign: 'center' }}>
                <Typography variant="h5" sx={{ display: 'flex', alignItems: 'center', justifyContent: 'center', gap: 1.5, fontWeight: 'bold', letterSpacing: 0.5 }}>
                  <HomeIcon fontSize="large" /> Розумний Будинок
                </Typography>
```

```
            </div>

            <Box sx={{ borderBottom: 1, borderColor: 'divider', bgcolor: 'background.paper' }}>

              <Tabs value={currentTab} onChange={handleTabChange} variant="fullWidth" textColor="primary" indicatorColor="primary" sx={{ '& .MuiTab-root': { py: 2 } }}>

                <Tab icon={<DashboardIcon />} label="Головна" iconPosition="start" sx={{ minHeight: 64 }} />

                <Tab icon={<ScheduleIcon />} label="Автоматизація" iconPosition="start" sx={{ minHeight: 64 }} />

              </Tabs>

            </Box>

            <CardContent sx={{ p: { xs: 2, sm: 4 } }}>

              <Box role="tabpanel" hidden={currentTab !== 0} sx={{ display: currentTab === 0 ? 'block' : 'none' }}>

                <Box className="controls-section">

                  <Paper elevation={0} sx={{ mb: 3, p: 2, bgcolor: '#e8eaf6', borderRadius: 3, border: '1px solid #c5cae9' }}>

                    <Typography variant="subtitle2" color="primary" gutterBottom sx={{ display: 'flex', alignItems: 'center', gap: 1, fontWeight: 'bold', letterSpacing: 1 }}>

                      <SettingsRemoteIcon fontSize="small" /> ЗАГАЛЬНИЙ РЕЖИМ

                    </Typography>

                    <FormControlLabel control={<Switch checked={autoMode} onChange={handleAutoModeChange} size="medium" />} label={<Typography fontWeight="500" fontSize="1.1rem">Активувати автоматизацію</Typography>} />

                  </Paper>


                  <Paper elevation={0} className={`control-group ${autoMode ? 'disabled' : ''}`} sx={{ mb: 3, p: 2.5, border: '1px solid #eee', borderRadius: 3, opacity: autoMode ? 0.5 : 1, pointerEvents: autoMode ? 'none' : 'auto', transition: 'all 0.3s ease' }}>

                    <Typography variant="subtitle2" color="textSecondary" gutterBottom sx={{ fontWeight: 'bold', letterSpacing: 1, mb: 2 }}>РУЧНЕ КЕРУВАННЯ</Typography>

                    <Box sx={{ display: 'flex', flexDirection: 'column', gap: 2 }}>

                      <DeviceSwitch device="fan" label="Вентилятор" icon="❄️" disabled={autoMode} />
```

<DeviceSwitch device="lamp1" label="Вітальня (Лампа A)" icon="💡" disabled={autoMode} />

<DeviceSwitch device="lamp2" label="Спальня (Лампа B)" icon="💡" disabled={autoMode} />

<DeviceSwitch device="lamp3" label="Дитяча (Лампа C)" icon="💡" disabled={autoMode} />

<DeviceSwitch device="window" label="Жалюзі" icon="🪟" disabled={autoMode} />

</Box>

</Paper>

<Paper elevation={0} sx={{ p: 2.5, borderRadius: 3, bgcolor: securitySystem ? '#ffebee' : '#f1f3f4', border: '1px solid', borderColor: securitySystem ? '#ffcdd2' : 'transparent', transition: 'all 0.3s ease' }}>

<Typography variant="subtitle2" color={securitySystem ? "error" : "textSecondary"} gutterBottom sx={{ display: 'flex', alignItems: 'center', gap: 1, fontWeight: 'bold', letterSpacing: 1 }}>

<SecurityIcon fontSize="small" /> БЕЗПЕКА

</Typography>

<FormControlLabel control={<Switch checked={securitySystem} onChange={handleSecuritySystemChange} color="error" size="medium" />} label={<Typography fontWeight="500" fontSize="1.1rem" color={securitySystem ? "error" : "inherit"}>{securitySystem ? "Система під охороною" : "Охорона вимкнена"}</Typography>} />

</Paper>

</Box>

</Box>

<Box role="tabpanel" hidden={currentTab !== 1} sx={{ display: currentTab === 1 ? 'block' : 'none' }}>

<AutomationTab mqttClient={mqttClient} initialConfig={automationConfig} isLoaded={isConfigLoaded} onUpdateConfig={setAutomationConfig} />

</Box>

</CardContent>

```
                <Box sx={{ px: 3, py: 1.5, borderTop: '1px solid #eee', display: 'flex', justifyContent: 'space-between', fontSize: '0.9rem', color: '#555', bgcolor: '#f8f9fa' }}>

                    <Box sx={{ display: 'flex', alignItems: 'center', gap: 1 }}>MQTT: <span style={{ display: 'inline-block', width: 10, height: 10, borderRadius: '50%', bgcolor: mqttStatusColor, marginRight: 5 }}></span><span style={{ color: mqttStatusColor, fontWeight: 'bold' }}>{isConnected ? 'Підключено' : 'Від\'єднано'}</span></Box>

                    <Box sx={{ display: 'flex', alignItems: 'center', gap: 1 }}></Box>

                </Box>

            </Card>

        </Fade>

    </div>

  </ThemeProvider>

 );

}


export default App;
```

ДОДАТОК Д

Програмний код компонента AutomationTab для керування розкладам

```
import React, { useState, useEffect } from 'react';
import { Card, CardContent, Typography, Box, FormControl, InputLabel, Paper, Select, MenuItem, TextField, ToggleButton, ToggleButtonGroup, Button, CircularProgress, Fade, Alert, Snackbar, Chip } from '@mui/material';
import { Save as SaveIcon, AccessTime as TimeIcon, CalendarToday as DayIcon } from '@mui/icons-material';


const DAYS_OF_WEEK = [
  { val: 1, label: 'Пн' }, { val: 2, label: 'Вт' }, { val: 3, label: 'Ср' },
  { val: 4, label: 'Чт' }, { val: 5, label: 'Пт' }, { val: 6, label: 'Сб' }, { val: 0, label: 'Нд' }
];


const DEVICES = [
    { id: 'fan', name: 'Вентилятор', icon: '❄️' },
    { id: 'window', name: 'Жалюзі', icon: '🪟' },
    { id: 'lamp1', name: 'Лампа A (Вітальня)', icon: '💡' },
    { id: 'lamp2', name: 'Лампа B (Спальня)', icon: '💡' },
    { id: 'lamp3', name: 'Лампа C (Дитяча)', icon: '💡' },
];


const AutomationTab = ({ mqttClient, initialConfig, isLoaded, onUpdateConfig }) => {
    const [localConfig, setLocalConfig] = useState(initialConfig || {});
    const [hasChanges, setHasChanges] = useState(false);
    const [showSuccess, setShowSuccess] = useState(false);


    useEffect(() => {
        if (isLoaded && initialConfig) {
```

```
            setLocalConfig(initialConfig);
        }
    }, [initialConfig, isLoaded]);

    const handleConfigChange = (deviceId, field, value) => {
        setLocalConfig(prev => ({ ...prev, [deviceId]: { ...prev[deviceId], [field]: value } }));
        setHasChanges(true);
    };

    const handleSaveAll = () => {
        mqttClient.publish('automation/save_all', localConfig);
        setHasChanges(false);
        setShowSuccess(true);
    };

    if (!isLoaded) {
        return (
            <Box sx={{ display: 'flex', flexDirection: 'column', alignItems: 'center', justifyContent: 'center', minHeight: 300, gap: 2, color: 'text.secondary' }}>
                <CircularProgress size={40} thickness={4} />
                <Typography>Завантаження налаштувань...</Typography>
            </Box>
        );
    }

    return (
        <Box sx={{ pb: 10 }}>
            {DEVICES.map(device => {
                const config = localConfig[device.id] || { mode: 'disabled', days: [], start: '09:00', end: '18:00' };
```

```
const isSchedule = config.mode === 'schedule';
const supportsSensorMode = !['fan', 'window'].includes(device.id);

return (
  <Card key={device.id} sx={{ mb: 2.5, borderRadius: 3, boxShadow: '0 2px 8px rgba(0,0,0,0.05)', border: '1px solid #eee', overflow: 'visible' }}>
    <CardContent sx={{ p: 2.5, '&:last-child': { pb: 2.5 } }}>
      <Box sx={{ display: 'flex', alignItems: 'center', mb: 2 }}>
        <Typography variant="h6" sx={{ fontWeight: 'bold', flexGrow: 1, display: 'flex', alignItems: 'center', gap: 1.5 }}>
          <span style={{ fontSize: '1.5rem' }}>{device.icon}</span> {device.name}
        </Typography>
        <Chip
          label={isSchedule ? "Розклад" : (config.mode === 'manual' ? "Сенсори" : "Вимкнено")}
          color={isSchedule ? "primary" : (config.mode === 'manual' ? "success" : "default")}
          size="small" variant="outlined"
        />
      </Box>

      <FormControl fullWidth sx={{ mb: isSchedule ? 2 : 0 }} size="small" variant="outlined">
        <InputLabel>Режим роботи</InputLabel>
        <Select
          value={config.mode}
          label="Режим роботи"
          onChange={(e) => handleConfigChange(device.id, 'mode', e.target.value)}
          sx={{ borderRadius: 2 }}
        >
          <MenuItem value="disabled">🚫 Вимкнено (не реагує в авто-режимі)</MenuItem>
```

```
                    {supportsSensorMode && (
                      <MenuItem value="manual">📡 За даними сенсорів</MenuItem>
                    )}
                    <MenuItem value="schedule">🗓️ За розкладом</MenuItem>
                  </Select>
                </FormControl>

                <Fade in={isSchedule} unmountOnExit>
                  <Box sx={{ mt: 2, pl: 2, borderLeft: '4px solid #4e54c8', bgcolor: '#f8f9fa', p: 2, borderRadius: 2 }}>
                    <Box sx={{ mb: 2 }}>
                      <Typography variant="caption" sx={{ display: 'flex', alignItems: 'center', gap: 0.5, color: '#666', mb: 1, fontWeight: 'bold' }}>
                        <DayIcon fontSize="inherit" /> АКТИВНІ ДНІ
                      </Typography>
                      <ToggleButtonGroup
                        value={config.days}
                        onChange={(e, newDays) => handleConfigChange(device.id, 'days', newDays)}
                        size="small"
                        sx={{ display: 'flex', justifyContent: 'space-between', width: '100%', bgcolor: 'white', borderRadius: 50 }}
                        color="primary"
                      >
                        {DAYS_OF_WEEK.map(day => (
                          <ToggleButton
                            key={day.val}
                            value={day.val}
                            sx={{
                              flexGrow: 1, borderRadius: '50% !important', border: 'none', mx: 0.5, width: 36, height: 36, p: 0,
```

```
                                    '&.Mui-selected': { color: 'white', bgcolor: 'primary.main', '&:hover': { bgcolor: 'primary.dark' } }
                                }}
                            >
                                {day.label}
                            </ToggleButton>
                        ))}
                    </ToggleButtonGroup>
                </Box>

                <Box>
                    <Typography variant="caption" sx={{ display: 'flex', alignItems: 'center', gap: 0.5, color: '#666', mb: 1, fontWeight: 'bold' }}>
                        <TimeIcon fontSize="inherit" /> ПРОМІЖОК ЧАСУ
                    </Typography>
                    <Box sx={{ display: 'flex', gap: 2 }}>
                        <TextField
                            label="Початок" type="time" size="small" fullWidth value={config.start}
                            onChange={(e) => handleConfigChange(device.id, 'start', e.target.value)}
                            InputLabelProps={{ shrink: true }} inputProps={{ step: 300 }}
                            sx={{ bgcolor: 'white', '& .MuiOutlinedInput-root': { borderRadius: 2 } }}
                        />
                        <TextField
                            label="Кінець" type="time" size="small" fullWidth value={config.end}
                            onChange={(e) => handleConfigChange(device.id, 'end', e.target.value)}
                            InputLabelProps={{ shrink: true }} inputProps={{ step: 300 }}
```

```
                                sx={{ bgcolor: 'white', '& .MuiOutlinedInput-root': { borderRadius: 2 } }}
                            />
                        </Box>
                    </Box>
                </Box>
            </Fade>
        </CardContent>
    </Card>
  );
})}

<Fade in={hasChanges}>
    <Paper elevation={6} sx={{ position: 'fixed', bottom: 24, left: '50%', transform: 'translateX(-50%)', zIndex: 1000, width: 'auto', borderRadius: 50, p: 0.5, bgcolor: 'white' }}>
        <Button
            variant="contained" color="primary" size="large" startIcon={<SaveIcon />} onClick={handleSaveAll}
            sx={{ borderRadius: 50, px: 4, py: 1.5, fontSize: '1rem', fontWeight: 'bold', textTransform: 'none', boxShadow: 'none' }}
        >
            Зберегти зміни
        </Button>
    </Paper>
</Fade>

<Snackbar open={showSuccess} autoHideDuration={3000} onClose={() => setShowSuccess(false)} anchorOrigin={{ vertical: 'bottom', horizontal: 'center' }}>
    <Alert severity="success" variant="filled" sx={{ width: '100%', boxShadow: 4, borderRadius: 2 }}>Налаштування успішно збережено!</Alert>
</Snackbar>
</Box>
```

```
    );
};

export default AutomationTab;
```