Хмельницький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерної інженерії та інформаційних систем

КВАЛІФІКАЦІЙНА РОБОТА

бакалавр
Освітній рівень

IoT-базована система охорони місцевості із передачею даних на сервер
Назва теми

КвРКІ 022022.22.01.14 ПЗ
Шифр

Галузь знань *12 «Інформаційні технології»*
Шифр, назва

Спеціальність *123 «Комп'ютерна інженерія»*
Шифр, назва

Освітня програма *«Комп'ютерна інженерія та програмування»*
Назва

Виконав: студент III курсу, група КІ2с-22-1 _____ Денис МАГОЛА
Підпис, Ініціали, прізвище

Керівник _____ Валерій МАРТИНЮК
Підпис, дата, Ініціали, прізвище

Нормоконтролер _____ Тетяна КИСІЛЬ
Підпис, дата, Ініціали, прізвище

До захисту допускаю:
Зав. кафедри комп'ютерної інженерії та інформаційних систем _____ Ольга ПАВЛОВА
Підпис, Ініціали, прізвище

« 9 » червня 2025 р.

Хмельницький 2025

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ

Освітній рівень БАКАЛАВР

Галузь знань 12 ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ

Спеціальність 123 КОМП'ЮТЕРНА ІНЖЕНЕРІЯ

Освітня програма «КОМП'ЮТЕРНА ІНЖЕНЕРІЯ ТА ПРОГРАМУВАННЯ»

ЗАТВЕРДЖУЮ

Зав. кафедри Ольга ПАВЛОВА

" 10 " 01 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА

Денису МАГОЛІ

Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) IoT-базована система охорони місцевості із передачею даних на сервер

Керівник проекту (роботи) Валерій МАРТИНЮК, д.т.н., проф.

Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 07.02.2025 р. № 23

2. Строк подання студентом проекту (роботи) на кафедру 01.06.2025 р.

3. Вихідні дані до проекту (роботи) Завдання на кваліфікаційну роботу

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

Аналіз програмно-технічної IoT-базованої системи охорони місцевості із передачею даних на сервер

Вибір компонентів розробки і проєктування IoT-базованої системи охорони місцевості із передачею даних на сервер

Програмно-технічна реалізація IoT-базованої системи охорони місцевості із передачею даних на сервер

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

Архітектура ПЗ проєкту

ER-діаграма системи

Схема функціонування системи

6. Консультанти розділів дипломного проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийв
Нормоконтроль	Тетяна КИСІЛЬ, доцент кафедри КПІС		
Антиплагіат	Андрій НІЧЕПОРУК, доцент кафедри КПІС		

7. Дата видачі завдання «_10_»_____01_____2025 р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів (розділів) дипломного проекту (роботи)	Термін виконання етапів проекту (роботи)	Приміт
1	Вибір напряму дослідження та узгодження тематики кваліфікаційної роботи з керівником	10.01.2025	виконан
2	Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження	01.02.2025	виконан
3	Робота над розділом 1 – дослідження предметної області та постановка задачі	01.03.2025	виконан
4	Робота над розділом 2 – вибір компонентів для проєктування IoT-базованої системи охорони місцевості із передачею даних на сервер	01.04.2025	виконан
5	Робота над розділом 3 – проєктування IoT-базованої системи охорони місцевості із передачею даних на сервер	29.04.2025	виконан
6	Оформлення пояснювальної записки згідно вимог	25.05.2025	виконан
7	Попередній захист ВКР	26.05.2025	виконан
8	Захист ВКР на засіданні ЕК	10.06.2025	

Студент _____ Денис МАГОЛА
Підпис Ініціали, прізвище

Керівник роботи _____ Валерій МАРТИНЮК
Підпис Ініціали, прізвище

№ рядка	формат	Позначення	Найменування	Кіл. листів	№ екз	Примітка
			Текстові документи			
1		КвРКІ 022022.22.01.14 ПЗ	Пояснювальна записка	59		
			Графічні матеріали			
2		КвРКІ 022022.22.01.14 Е8	Архітектура ПЗ проєкту	1		
3		КвРКІ 022022.22.01.14 Е8	ER-діаграма системи	1		
4		КвРКІ 022022.22.01.14 Е8	Схема функціонування системи	1		

Зм	Арк	№ докум	Підпис	Дата	КвРКІ 022022.22.01.14 ВП		
Розробив		Магола			Відомість проекту	Літера	Аркуш 1 / Аркушів 1
Перевір.		Мартинюк				У	
Н. контр.		Кисіль		09.06	ХНУ, КІ2с-22-1		
Затв.		Павлова		9.06.25			

АНОТАЦІЯ

Тема кваліфікаційної роботи: «IoT-базована система охорони місцевості із передачею даних на сервер».

Автор роботи: Денис МАГОЛА.

Керівник роботи: Валерій МАРТИНЮК.

Пояснювальна записка: 59 с., 9 рис., 4 дод., 50 джерел.

Графічна частина: 3 креслення.

ІОТ, АРХІТЕКТУРА, БАЗА ДАНИХ, СЕРВЕР, ТЕХНОЛОГІЯ.

Метою дипломної роботи є визначення умов та особливостей застосування IoT-базованої системи охорони місцевості із передачею даних на сервер. Робота акцентує увагу на ефективній архітектурі програмного забезпечення, протоколах обміну повідомленнями, моделюванні подій та взаємодії з користувачем через зручний і зрозумілий інтерфейс.

Об'єктом дослідження є процес охорони території із використанням IoT технологій.

Предметом дослідження є функціонування IoT-базованих систем, принципи і особливості реалізації їхньої архітектури і обробки даних.

У процесі дослідження було проведено систематичний аналіз доступних текстових джерел з метою вивчення актуальних підходів, рішень та особливостей у межах обраної предметної області.

Підпис студента

30.05.2025

Дата

ЗМІСТ

ВСТУП 4

1 АНАЛІЗ ПРОГРАМНО-ТЕХНІЧНОЇ ІОТ-БАЗОВАНОЇ СИСТЕМИ ОХОРОНИ МІСЦЕВОСТІ ІЗ ПЕРЕДАЧЕЮ ДАНИХ НА СЕРВЕР 5

1.1 Аналіз предметної області і виявлення наявних проблем і завдань 5

1.2 Порівняльний аналіз переваг та недоліків існуючих рішень 9

1.3 Підходи до вирішення задачі за темою дослідження та постановка задачі 12

1.4 Висновки 16

2 ВИБІР КОМПОНЕНТІВ РОЗРОБКИ І ПРОЄКТУВАННЯ ІОТ-БАЗОВАНОЇ СИСТЕМИ ОХОРОНИ МІСЦЕВОСТІ ІЗ ПЕРЕДАЧЕЮ ДАНИХ НА СЕРВЕР 18

2.1 Визначення апаратних компонентів програмно-технічного засобу 18

2.2 Проєктування архітектури основних модулів системи 18

2.3 Організація взаємодії між компонентами системи 25

2.4 Висновки 37

3 ПРОГРАМНО-АПАРАТНА РЕАЛІЗАЦІЯ ІОТ-БАЗОВАНОЇ СИСТЕМИ ОХОРОНИ МІСЦЕВОСТІ ІЗ ПЕРЕДАЧЕЮ ДАНИХ НА СЕРВЕР 38

3.1 Опис функціонування апаратної частини програмно-технічного засобу 38

3.2 Програмна реалізація серверної логіки та інтерфейсу керування 44

3.3 Опис процесу створення баз даних 53

3.4. Висновки 58

ВИСНОВКИ 60

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ 62

ДОДАТОК А Копія креслення «архітектура ПЗ проєкту» 67
ДОДАТОК Б Копія креслення «ER-діаграма системи» 68
ДОДАТОК В Копія креслення «схема функціонування системи» 69
ДОДАТОК Г Код програми 70

ВСТУП

Розвиток цифрових технологій, зокрема “Інтернету речей” (IoT), відкриває нові можливості для підвищення рівня безпеки об’єктів та територій. Тенденція до автоматизації процесів моніторингу, дистанційного контролю та обробки інформації призводить до активного впровадження IoT-рішень у сферу охоронних систем. Хоча на ринку представлено чимало варіантів, багато з них є дорогими, складними в налаштуванні або орієнтованими на використання спеціалізованого обладнання.

Актуальність теми даної кваліфікаційної роботи визначається потребою у створенні сучасної системи охорони місцевості, що ґрунтується на принципах IoT і здатна забезпечити ефективний контроль за станом певного об'єкта місцевості. Подібні системи можуть бути корисними як у приватному, так і в комерційному або промисловому секторі, і їх розробка дозволяє зменшити витрати на впровадження і обслуговування, а також відкриває можливості для подальшої модернізації та інтеграції з іншими сервісами.

Метою дипломної роботи є визначення умов та особливостей застосування IoT-базованої системи охорони місцевості із передачею даних на сервер. Робота акцентує увагу на ефективній архітектурі програмного забезпечення, протоколах обміну повідомленнями, моделюванні подій та взаємодії з користувачем через зручний і зрозумілий інтерфейс.

Об’єктом дослідження є процес охорони території із використанням IoT технологій.

Предметом дослідження є функціонування IoT-базованих систем, принципи і особливості реалізації їхньої архітектури і обробки даних.

У процесі дослідження було проведено систематичний аналіз доступних текстових джерел з метою вивчення актуальних підходів, рішень та особливостей у межах обраної предметної області.

1 АНАЛІЗ ПРОГРАМНО-ТЕХНІЧНОЇ ІОТ-БАЗОВАНОЇ СИСТЕМИ ОХОРОНИ МІСЦЕВОСТІ ІЗ ПЕРЕДАЧЕЮ ДАНИХ НА СЕРВЕР

1.1 Аналіз предметної області і виявлення наявних проблем і завдань

Інтернет речей (Internet of Things, IoT) – це концепція, яка полягає в об'єднанні фізичних пристроїв у мережу через Інтернет з можливістю обміну даними між ними. Сучасні IoT-рішення здатні не лише збирати інформацію, а й реагувати на неї в реальному часі, приймаючи певні рішення без втручання людини. IoT – це одна з тих технологій, які спочатку здавалися чимось фантастичним, а тепер стали частиною повсякденного життя. Суть у тому, що звичайні побутові пристрої підключаються до Інтернету і можуть передавати або отримувати інформацію. Завдяки цьому техніка “стає розумною”: вона може повідомити про несправність, автоматично виконати завдання, або навіть взаємодіяти з іншими пристроями без участі людини. Все це раніше здавалося складним і дорогим, але за останні кілька років усе змінилося. З'явилися доступні мікроконтролери, модулі бездротового зв’язку, відкриті платформи, і тепер, навіть студенти можуть реалізовувати цілком серйозні проєкти на базі IoT [1].

Вже зараз інтернету речей приділяється увага на найвищому рівні. Починаючи з 2009 року, у Брюсселі за сприяння Європейської комісії регулярно організовуються щорічні конференції під назвою “Annual Internet of Things”. У рамках цих заходів свої погляди на розвиток Інтернету речей представляють різні наукові дослідники та керівники провідних IT-компаній. За оцінками аналітичних центрів, найближчим часом очікується стрімке зростання цієї галузі, зокрема, аналітики компанії Gartner прогнозували, що до 2020 року кількість пристроїв, підключених до Інтернету, сягне понад 26 мільярдів, а сукупний дохід від продажу обладнання, програмного забезпечення та супутніх послуг перевищить 1,9 трильйона доларів США. Інші джерела навіть надавали ще більш оптимістичні прогнози щодо масштабів зростання. Світові технологічні гіганти вже активно конкурують за лідерство в цьому секторі, наприклад, у 2014 році компанія Intel,

після випуску системи на кристалі Edison, оголосила конкурс “Make it Wearable” із загальним призовим фондом у 1,3 мільйона доларів на кращі рішення в межах концепції IoT. Крім того, вона створила спеціалізований підрозділ “Internet of Things Solutions Group”, який займається розвитком цієї технологічної сфери.

Якщо говорити простими словами, то IoT – це не лише про комфорт, як у випадку з “розумним будинком”, а ще й про ефективність, безпеку, автоматизацію і зменшення людського фактору там, де це важливо. І якраз безпека одна з тих сфер, де такі технології найбільше потрібні [2].

Тематика охорони місцевості, особливо у контексті приватної власності, критичної інфраструктури або об'єктів особливого значення, є дуже актуальною сьогодні. На жаль, випадки крадіжок, вандалізму, незаконного проникнення трапляються постійно і в зв'язку із теперішньою ситуацією в Україні такі ситуації виникають все частіше. В містах, селах, на дачах, на фермах, у складських приміщеннях – скрізь, де територія лишається без нагляду хоча б на кілька годин чи днів, завжди виникає потенційна небезпека для безпеки майна або обладнання. Навіть звичайний приватний двір або гараж – це вже потенційний об'єкт для зловмисників. Всі хочуть відчувати себе в безпеці, але поставити охоронця біля кожного об'єкта не дуже доцільно, так як це дорого і ненадійно. Тому люди все частіше шукають способи зробити охоронну систему такою, яка працює сама, повідомляє власника про підозрілу активність, веде відеоспостереження, зберігає докази, а можливо, навіть вміє розпізнавати обличчя або об'єкти. Саме тому інтерес до автоматизованих систем охорони на базі IoT зростає все більше [3].

При створенні такої системи виникає перелік завдань, які потребують уваги, особливо якщо йдеться не просто про експериментальний прототип, а про практичне рішення, яке буде застосовуватись у реальних умовах. Одне з головних завдань – це забезпечення надійного виявлення руху або присутності сторонніх осіб на території. Тут важливо враховувати не лише чутливість самих сенсорів, а й правильну логіку обробки отриманих даних. У той самий час система має швидко реагувати на справжні загрози і не пропускати підозрілі події.

Наступним важливим аспектом є передача даних, при якій може виникати досить багато складнощів. Проблема в тому, що далеко не всі місця, які потребують охорони, мають стабільний доступ до Інтернету. Часто такі локації розташовані у віддалених або важкодоступних районах, де інфраструктура обмежена або відсутня взагалі. Це можуть бути ділянки за містом, склади на околицях, дачі, об'єкти без постійного електропостачання. У таких умовах потрібно враховувати можливість роботи в нестабільній мережі або взагалі в автономному режимі – з подальшою синхронізацією, коли з’явиться зв’язок.

Приклад рішення проблеми: LoRaWAN (Long Range Wide Area Network) – це протокол бездротового зв’язку з великою дальністю дії та низьким енергоспоживанням, який ідеально підходить для IoT-систем у віддалених або важкодоступних місцях. Завдяки здатності передавати дані на великі відстані (до 10 км у відкритих просторах) і працювати з невеликою кількістю енергії, LoRaWAN дозволяє організувати стабільний зв’язок навіть там, де немає покриття мобільного Інтернету чи WiFi. LoRaWAN підтримує асинхронний режим роботи, що означає, що пристрої можуть надсилати невеликі пакети даних періодично або за подіями, з мінімальним споживанням батареї. Це робить її ідеальною для об'єктів, які працюють від батарей або сонячних панелей, і там, де відсутнє постійне електропостачання. Також ця технологія працює в спеціальному діапазоні, що дозволяє їй проникати через перешкоди, такі як будівлі чи дерева. Завдяки мережевій архітектурі “зірка” (star topology), дані з багатьох датчиків передаються на центральний шлюз (gateway), який потім передає їх на сервер для обробки та візуалізації. Такий підхід дозволяє охопити великі території з мінімальними інфраструктурними витратами.

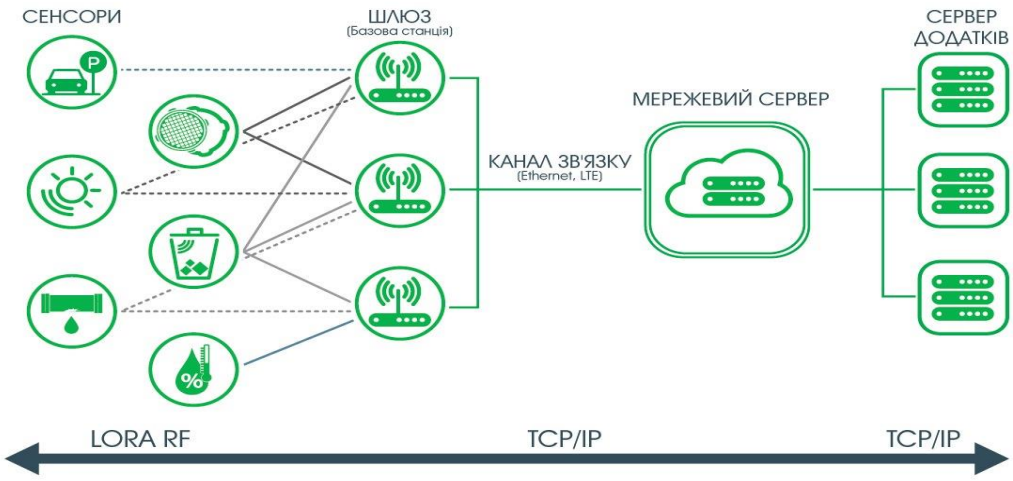


Рисунок 1.1 – Схема типової мережі LoRaWAN [4]

Окремо слід зосередити увагу на безпеці переданих даних, так як це стосується чутливої інформації, наприклад, відео з камер або повідомлення про тривогу. Якщо така інформація передається відкритими каналами або без шифрування, її можуть перехопити сторонні особи, змінити або заблокувати. Тому система має використовувати сучасні протоколи безпечного зв'язку, наприклад, HTTPS або MQTT з TLS. Важливо, щоб не лише канал зв'язку був зашифрованим, а й сама структура взаємодії між пристроями та сервером передбачала автентифікацію, перевірку достовірності джерела даних та захист від стороннього втручання. Це означає, що потрібно впроваджувати на рівні програмного забезпечення механізми контролю доступу, шифрування повідомлень, цифрові підписи або токени автентифікації.

Ще один нюанс – це безперервність роботи. Навіть, якщо зв'язок є, він може бути нестабільним, тому система має не просто відправляти дані в реальному часі, а й уміти їх буферизувати, повторно передавати у випадку збоїв, і мати механізм підтвердження доставки.

Отже, проблематика полягає не лише у зборі даних, а у забезпеченні їхньої якісної обробки, надійної передачі, захисту та гнучкого реагування на змінні умови. Саме тому ця тема і цікава, і складна одночасно, в ній потрібно поєднати багато

різних знань – не лише технічних, а й практичних, щоб отримати систему, яка буде справді працювати і приносити користь.

Таким чином, тема створення IoT-системи охорони місцевості є не лише актуальною, але й технічно цікавою, бо охоплює багато важливих напрямів, від роботи із сенсорами, до захисту передачі даних і налаштування ефективної та надійної обробки повідомлень.

1.2 Порівняльний аналіз переваг та недоліків існуючих рішень

На сучасному ринку представлено досить багато рішень для охорони територій з використанням технологій Інтернету речей (IoT). Зазвичай це комерційні комплекси, що включають у себе різноманітні датчики, камери спостереження, хмарні сервіси для збереження й аналізу даних, а також програмне забезпечення для управління системою з мобільного пристрою. У цьому розділі розглянемо кілька таких рішень, проаналізуємо їх переваги й недоліки, а також порівняємо їх із підходом до створення власної IoT-базованої системи охорони місцевості.

Одним із найпопулярніших рішень в Україні та Європі є система безпеки Ajax [5]. Вона є повноцінною охоронною платформою, яка включає різні типи датчиків: руху, відкриття дверей, диму, затоплення тощо. Ajax пропонує централізований хаб, який об’єднує всі сенсори та передає дані через WiFi або GSM-зв’язок на сервер, звідти користувач отримує повідомлення у застосунку. Переваги Ajax: висока якість компонентів та стабільна робота; простота у встановленні й налаштуванні; дизайн, орієнтований на кінцевого користувача; можливість резервного живлення та автономної роботи при зникненні живлення; надійний захищений зв'язок між компонентами (власний протокол Jeweller з шифруванням). Недоліки: висока вартість, при якій комплект із базових елементів може коштувати понад 10 000 грн; закрита система, через яку неможливо додати нестандартні компоненти або змінити логіку роботи без прив'язки до їх екосистеми; налаштування обмежені лише тим, що дозволено в застосунку.

Ще один приклад – система Google Nest, яка позиціонується більше як частина екосистеми “розумного будинку”, але також має охоронні функції [6]. Наприклад, “Nest Cam” – це камера відеоспостереження з можливістю виявлення руху, зберігання відео у хмарі та надсилання сповіщень у разі виявлення активності. Інтеграція з Google Home дозволяє керувати камерою голосом або автоматизувати її через сценарії. Переваги Google Nest: якісне зображення та аналітика відео (розпізнавання людей, тварин, звуків); хмарне зберігання даних з можливістю перегляду архіву; можливість інтеграції з іншими пристроями (розумні дверні дзвінки, освітлення тощо); простота використання. Недоліки: для збереження відео потрібна підписка на платний тариф; дані зберігаються на сторонньому сервері, тому відсутній повний контроль над приватністю; висока вартість, через яку одна камера коштує близько 150–200 доларів; залежність від стабільного Інтернет-зв’язку.

Також існує популярне рішення Reolink – це компанія, що спеціалізується на камерах відеоспостереження [7]. Її моделі Reolink Go і Reolink Argus призначені для зовнішнього використання й працюють через мобільний зв’язок або WiFi. Це автономні пристрої, які мають вбудований акумулятор, сонячну панель, PIR-датчик руху, запис відео на SD-карту, а також надсилання сповіщень у застосунок. Переваги Reolink: працює без кабелів, повністю автономно; проста установка у важкодоступних місцях; наявність нічного бачення, запису по тригеру руху; можливість локального зберігання даних (SD-карта). Недоліки: не підтримує власну інтелектуальну обробку подій, лише фіксація руху; мінімальна кастомізація, знову ж таки залежність від мобільного Інтернету для сповіщень; обмежений обсяг пам’яті (SD-карта не завжди надійна у довготривалому зберіганні).

Узагальнюючи, більшість готових рішень мають одну спільну проблему – обмежене налаштування, та залежність від зовнішніх хмарних сервісів. У разі зміни політики компанії, відключення сервера або блокування доступу користувач може втратити контроль над системою. Крім того, у багатьох випадках дані користувача

зберігаються на серверах, які фізично розташовані за межами країни, що створює питання конфіденційності.

Інша проблема – це неможливість змінити логіку системи. Наприклад, якщо користувач хоче, щоб система надсилала зображення не лише в застосунок, а й на його Telegram, або щоб тривога активувалась лише при фіксації руху на кількох датчиках одночасно – реалізувати це в закритих системах практично неможливо.

Крім того, в умовах нестабільного або взагалі відсутнього Інтернет-зв'язку такі системи часто втрачають свою функціональність. І хоча деякі з них мають локальне збереження, без постійного зв'язку з сервером вони не здатні надати повноцінну аналітику або відправити сповіщення.

На фоні всіх перелічених прикладів постає необхідність в розробці власної системи. Власна IoT-базована система охорони дає повний контроль над її структурою, логікою, безпекою і способом взаємодії з користувачем. Наприклад, можна самостійно визначити, куди надсилаються повідомлення: на сервер, у Telegram, у локальну базу даних. Можна реалізувати буферизацію даних у разі втрати зв'язку або організувати зберігання подій на локальному сервері з подальшою синхронізацією. Також можна використати відкриті протоколи (MQTT, HTTPS з TLS), налаштувати багаторівневу автентифікацію, реалізувати кастомні сценарії обробки подій тощо.

Тому розробка власної системи допомагає глибше зрозуміти, як працюють сучасні охоронні комплекси, як обробляються дані, як реалізується захищений зв'язок, і як побудувати систему, яка справді відповідає конкретним потребам. У цій роботі хочеться не просто повторити те, що вже є, а створити свою зручну та економічно доцільну систему, яку можна було б реально застосувати у повсякденних умовах.

1.3 Підходи до вирішення задачі за темою дослідження та постановка задачі

Для побудови ефективної IoT-базованої системи охорони місцевості слід застосувати комплексний підхід, який охоплює кілька напрямів: вибір архітектури системи, методи збору й обробки даних, організація передачі інформації, забезпечення безпеки комунікацій та зручність взаємодії з користувачем.

Щоб Інтернет речей (IoT) працював, потрібні кілька ключових компонентів. Розглянемо основні з них:

1) датчики IoT;
2) з'єднання і мережа;
3) MQTT-брокер повідомлень;
4) система прийняття рішень (Hyper Decision Framework);
5) інтерфейс користувача.

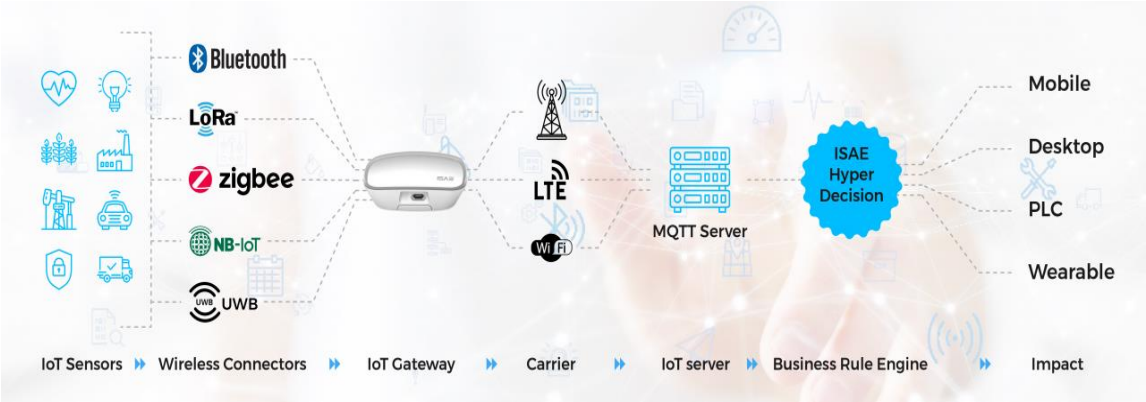


Рисунок 1.2 – Компоненти IoT-базованої системи і їх взаємозв'язок [8]

Саме завдяки датчикам "речі" в IoT набувають сенсу. Наприклад, якщо потрібно контролювати температуру обладнання, то звичайний датчик просто показував би значення на екрані або вмикав сигналізацію при перевищенні норми. Але в IoT-системі цей сенсор може сам передати дані далі, наприклад, у хмару або спеціальній системі, яка сама приймає рішення. Є багато типів таких датчиків: температури, вологості, руху, світла, тиску тощо.

Щоб ці пристрої могли передавати інформацію, їм потрібне конкретне підключення. Раніше були дроти, але зараз використовується бездротовий зв'язок. Існує багато протоколів для передачі даних: Bluetooth Low Energy (BLE), LoRa, ZigBee, SigFox, NB-IoT. Найпопулярніший з них – BLE. Він дозволяє передавати дані від датчиків до пристроїв або шлюзів, навіть, якщо вони не поруч, за допомогою Mesh-мережі.

Інформація зазвичай передається мережею на сервер, і для цього часто використовують протокол MQTT, який дозволяє ефективно передавати багато даних. На сервері у нас буде працювати MQTT-брокер повідомлень, який отримує інформацію та фільтрує її. Звичайні сервери можуть не витримати такого навантаження, тому потрібні спеціальні IoT-сервери.

Зібрані дані самі по собі нічого не дають, потрібно, щоб система могла реагувати на них. Наприклад, якщо працівник заходить у небезпечну зону, система одразу може сповістити відповідальних осіб. Це можливо завдяки спеціальному програмному модулю, який аналізує дані в реальному часі та діє за певними правилами.

Щоб люди могли керувати всією системою, потрібен зрозумілий інтерфейс – зазвичай це мобільний додаток або веб-панель. Він дозволяє переглядати дані з датчиків або керувати обладнанням. Наприклад, у випадку охоронної системи це може бути перегляд інформації про спрацювання приладів, аналіз отриманих подій або зміна налаштувань контролю для певної ділянки.

У цій роботі я планую обрати оптимальні підходи та інструменти, які дозволять реалізувати проєкт максимально ефективно і надійно. Методологічні підходи до вирішення задачі в моєму випадку включають як архітектурні рішення, так і вибір конкретних мов програмування, фреймворків, платформ та баз даних.

Почну з серверної частини, адже саме вона є центральним елементом, яка обробляє всі сигнали від IoT-пристроїв, взаємодіє з базою даних, формує відповіді, а також забезпечує логіку сповіщень, журналювання та аналітики. Для її розробки я б обрав мову Java разом із фреймворком Spring Boot [9]. Це перевірена часом

технологія, яка дозволяє створювати надійні, масштабовані вебсервіси. Крім того, Spring Boot має велику екосистему бібліотек та інтеграцій, що значно спрощує реалізацію авторизації, логування, роботи з базами даних, REST API тощо. Особливо зручно, що Spring Boot чудово інтегрується з Hibernate, який буде використаний як ORM (Object-Relational Mapping) для взаємодії з реляційною базою даних.

У якості бази даних я розглядаю два варіанти. Перший – PostgreSQL, реляційна СУБД, яка дозволяє ефективно зберігати структуровані дані: дані про користувачів, журнали доступу, історію сповіщень тощо. PostgreSQL є дуже потужною і стабільною системою, з підтримкою складних запитів, тригерів і транзакцій, що ідеально підходить для критичних систем безпеки. Проте для зберігання деяких неструктурованих або напівструктурованих даних, таких як логи, JSON-документи від сенсорів або медіафайли (наприклад, фото з камер), я також розглядаю MongoDB – документоорієнтовану NoSQL базу даних. Вона дозволяє гнучко працювати з динамічною структурою даних і масштабувати систему горизонтально, що є перевагою для великих систем моніторингу.

На рівні IoT-обладнання та симуляції сенсорів для тестування навантаження системи я планую використати або платформу Node.js, або мову Python. Python є дуже зручним для швидкого прототипування і часто використовується в IoT-середовищах завдяки спеціальним бібліотекам, що дозволяють просто взаємодіяти з сенсорами. Крім того, Python добре підходить для написання скриптів, які імітують роботу датчиків, наприклад, руху, температури чи шуму. Node.js є альтернативою, особливо коли мова йде про роботу з потоками даних або створення легких, асинхронних API. Наприклад, на Node.js зручно реалізовувати сервіси, які передають дані від сенсорів через WebSocket або MQTT-протокол.

Що стосується передачі даних між IoT-пристроями та сервером, основним протоколом я розглядаю MQTT, оскільки він спеціально створений для швидкої та енергоефективної комунікації в IoT-системах. Цей протокол дозволяє передавати

повідомлення з мінімальними затримками та ресурсами, що ідеально для пристроїв із обмеженими можливостями.

Інтерфейс керування системою (навіть у найпростішій формі) можна реалізувати як окремий вебдодаток або просто REST API, в якому можна перевірити статус сенсорів, переглянути журнал подій, змінити налаштування або вмикати/вимикати окремі модулі.

Архітектурно система буде побудована з урахуванням принципів мікросервісів, або принаймні модульної структури. Це дозволить окремо розвивати підсистеми: наприклад, окремий модуль відповідає за збереження даних, інший за аналітику, ще один за обробку повідомлень або керування пристроями. Такий підхід забезпечує гнучкість, масштабованість і легшу підтримку коду в майбутньому.

Загалом, обрані підходи й інструменти базуються на реальному досвіді, аналізі вимог до системи безпеки та з урахуванням сучасних тенденцій в IoT і розподілених системах. Я прагну не лише реалізувати технічне рішення, а й забезпечити його стабільну роботу, адаптивність до змін та можливість масштабування, якщо система буде використовуватись у ширшому контексті, наприклад, для охорони великих об'єктів.

У межах цієї роботи передбачається вирішити певні завдання, які стосуються проєктування та реалізації IoT-базованої системи охорони місцевості. Насамперед потрібно дослідити, як функціонують подібні системи, які підходи використовуються у сфері безпеки на базі Інтернету речей. Далі слід розглянути загальну структуру такої системи, визначити ключові її компоненти та способи взаємодії між ними.

В ході роботи планується створити власну модель системи, яка охоплює базові сценарії роботи: виявлення руху, передача даних на сервер, оповіщення користувача та зберігання інформації для подальшого аналізу. Також буде визначено цілі розробки та обґрунтовано доцільність створення саме такої системи. В результаті очікується реалізація базової версії працездатної IoT-системи, яка

зможе на практиці виконувати основні завдання охорони місцевості, а також буде зроблено висновки щодо ефективності обраного підходу.

Тому завданнями роботи є:

– дослідити процедури функціонування IoT-базованих систем охорони місцевості;

– ознайомитися з теоретичними основами та сучасним станом розвитку галузі;

– описати, з чого складається система охорони місцевості та як вона в загальному працює;

– розглянути існуючі приклади реалізації подібних систем, виявити типові проблеми та способи їх подолання;

– на основі зібраної інформації визначити, що саме має вміти система, а також сформулювати вимоги до її роботи й стабільності, створити уявлення про її функціональну модель;

– зробити висновки щодо доцільності та актуальності розробки такої системи;

– оцінити, наскільки вдалося виконати заплановані завдання в межах цієї роботи.

1.4 Висновки

В межах першого розділу було здійснено аналіз предметної області, пов’язаної з побудовою систем охорони територій на основі технологій Інтернету речей. У ході дослідження були виявлені ключові проблеми, що виникають при створенні таких систем: недостатня надійність, складність у забезпеченні безпечної та стабільної передачі даних, обмеження у масштабованості та сумісності компонентів, а також відсутність зручних засобів для оперативного сповіщення користувача про події.

Також було проведено порівняльний аналіз існуючих рішень, як комерційних, так і відкритих систем, з оцінкою їхніх сильних і слабких сторін. Зокрема, розглянуто готові системи охорони місцевості з базовим IoT-функціоналом, системи і платформи сигналізації. Аналіз показав, що багато з наявних рішень або надто дорогі, або обмежені у функціональності, не передбачають адаптації під конкретні умови, або мають застарілі підходи до обробки та захисту даних.

На основі цього аналізу була сформульована постановка задачі дипломної роботи. Основною метою є розробка IoT-базованої системи охорони місцевості, яка буде здатна ефективно фіксувати сторонню активність, безпечно передавати дані на сервер, підтримувати автоматизовану обробку інформації та забезпечувати зручне сповіщення користувача у реальному часі. Для досягнення цієї мети необхідно визначити структуру системи, спроєктувати оптимально всі програмні компоненти, реалізувати надійний механізм передачі даних та перевірити ефективність системи шляхом практичного тестування. Застосування такого підходу дозволить створити сучасну, та доступну систему охорони, яка буде адаптована до реальних потреб користувачів і здатна працювати в різних умовах експлуатації.

2 ВИБІР КОМПОНЕНТІВ РОЗРОБКИ ІОТ-БАЗОВАНОЇ СИСТЕМИ ОХОРОНИ МІСЦЕВОСТІ ІЗ ПЕРЕДАЧЕЮ ДАНИХ НА СЕРВЕР

2.1 Визначення апаратних компонентів програмно-технічного засобу

Після проведення детального аналізу предметної області та визначення вимог до програмно-технічного засобу потрібно розглянути можливі способи вирішення поставлених задач. У процесі розробки системи охорони важливо розуміти, з яких саме підсистем складається повноцінне рішення. У випадку з програмно-технічним засобом, який реалізує IoT-систему безпеки, до таких підсистем належать як апаратні компоненти, так і програмні модулі, які забезпечують їхню взаємодію, логіку реагування, передачу даних і візуалізацію.

В цьому розділі розглянуто компоненти, які можуть бути складовими програмно-технічного засобу IoT-системи охорони. Основна увага приділяється апаратним елементам, зокрема тим, які безпосередньо фіксують зміни в навколишньому середовищі та передають дані на сервер. В кінцевому варіанті проєкту апаратна частина також буде емульована програмними засобами, задля тестування і відтвореня критично важливих аспектів роботи системи, щоб можна було перевірити велике навантаження і відтворити реальну роботу. Звісно для повноти розуміння і перспектив подальшої реалізації важливо проаналізувати повноцінну систему з реальними пристроями.

Будь-яка IoT-система складається з декількох компонентів: сенсорів, що збирають інформацію, пристроїв збору та обробки даних, засобів зв'язку та центрального сервера або хмарної платформи, де ці дані зберігаються і обробляються. Центральним елементом на рівні збору даних виступає мікроконтролер, в основі якого знаходиться мікропроцесор, а за своєю будовою він схожий на ті, що використовуються в персональних комп’ютерах. Головна відмінність полягає в тому, що мікроконтролер містить у собі не лише процесор, а й вбудовану пам’ять для зберігання програм та даних, а також необхідні інтерфейси для підключення до зовнішніх пристроїв і сенсорів. Цей невеликий програмований

пристрій, може отримувати сигнали з датчиків, обробляти їх у певному обсязі, а також передавати ці сигнали далі через модулі бездротового зв'язку або інтерфейси передачі даних.

Мікроконтролери зазвичай оснащені різноманітною периферією, яка дозволяє їм виконувати широкий спектр завдань без потреби у додатковому обладнанні. Серед таких вбудованих компонентів можна виділити інтерфейси для обміну даними, зокрема UART, I2C, SPI, CAN і навіть USB; аналого-цифрові та цифро-аналогові перетворювачі (АЦП перетворює аналоговий сигнал, наприклад, з датчика температури чи світла у цифрове значення, яке мікроконтролер може обробити, а ЦАП робить зворотну операцію: перетворює цифрове значення у плавний аналоговий сигнал, наприклад, щоб керувати гучністю або яскравістю без різких стрибків); компаратори (прості електронні пристрої, які порівнюють два аналогових сигнали напруги між собою і видають цифровий результат: наприклад, “1”, якщо одна напруга більша за іншу, і “0”, якщо менша або рівна. Використовується для прийняття простих рішень на основі зміни сигналу, наприклад, коли треба виявити, чи перевищила температура певне значення); широтно-імпульсні модулятори (ШІМ – це спосіб управління середнім значенням напруги, при якому сигнал швидко вмикається і вимикається з різною тривалістю “вкл” і “викл”. Такий метод застосовується, наприклад, для регулювання яскравості світлодіодів або швидкості обертання моторів); таймери (дозволяють точно вимірювати час або керувати подіями).

Серед найпопулярніших мікроконтролерів для побудови IoT-систем можна виділити ESP32. Це мікроконтролер з вбудованими модулями WiFi та Bluetooth, що дозволяє йому виступати як повноцінний вузол збору та передачі даних. Даний пристрій має низьке енергоспоживання, достатню кількість GPIO-виходів і використовуються для підключення різноманітних сенсорів, а також підтримує сучасні протоколи зв'язку. На відміну від простіших варіантів, як ESP8266 чи Arduino UNO, ESP32 забезпечує кращу продуктивність і є більш зручним засобом для проєктування складніших систем.

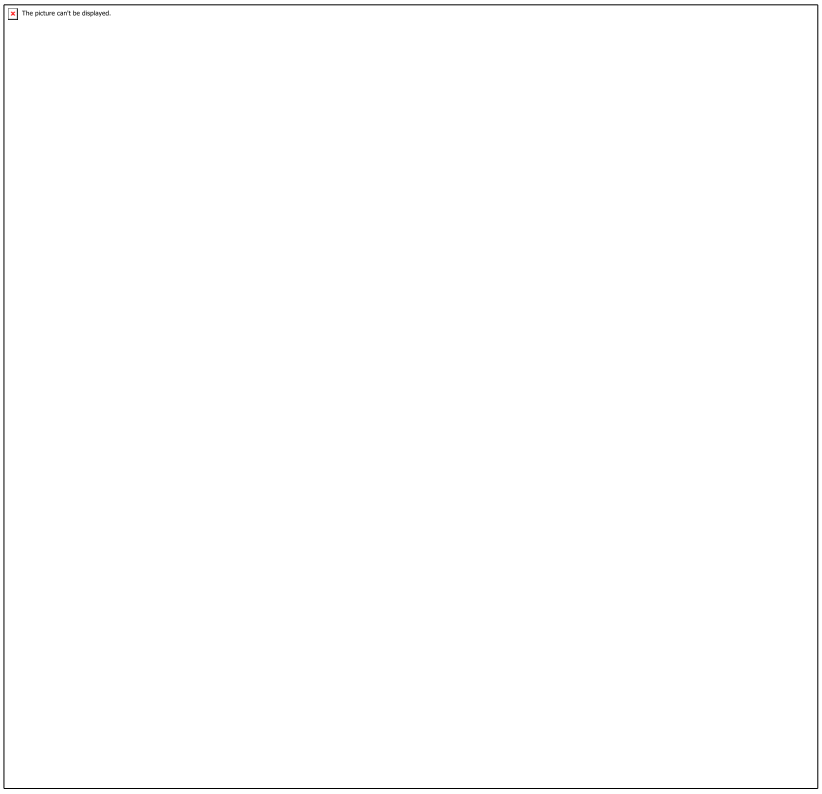


Рисунок 2.1 – ESP32 Basic Starter Kit і його вміст [10]

На зображенні представлений базовий набір для розробки IoT-проєктів на базі мікроконтролера ESP32 разом з основними аксесуарами. Цей набір включає в себе різноманітні компоненти для побудови, тестування та налагодження систем, що взаємодіють із фізичним середовищем. Такий базовий інструмент є дуже корисним для початку роботи з IoT-проєктами, він дозволяє моделювати та створювати універсальні системи. Завдяки широкому набору сенсорів і модулів, комплект дозволяє швидко протестувати ідеї, не вдаючись до пайки або складних підключень [11].

Розглянемо перелік основних елементів комплекту:

1. ESP32 Dev Board – головний мікроконтролер із підтримкою WiFi та Bluetooth. Забезпечує керування всіма підключеними компонентами.

2. Дисплей OLED – невеликий екран для відображення тексту, графіки або сенсорних даних.

3. PIR-сенсор руху – інфрачервоний датчик, що виявляє рух людини. В основному, використовується в системах безпеки.

4. Двоканальний реле-модуль – дозволяє вмикати або вимикати пристрої з високою напругою (наприклад, лампи або мотори).

5. Цифровий температурно-вологісний датчик DHT11 – вимірює температуру повітря та вологість.

6. IR-приймач – використовується для дистанційного керування за допомогою ІЧ-пульта.

7. Модуль живлення (AMS1117) – понижує напругу для живлення чутливих компонентів.

8. Фотоелектричний датчик (фотодіод/фоторезистор) – визначає рівень освітлення.

9. П’єзоелектричний дзвіночок (buzzer) – використовується для створення звукових сигналів.

10. Регулятор опору (потенціометр) – змінює напругу на вході для тестів аналогових входів.

11. Кнопки (push-buttons) – використовуються як прості елементи вводу.

12. Світлодіоди (LED) – для індикації подій чи стану системи.

13. Резистори різного номіналу – для обмеження струму у схемах.

14. З’єднувальні дроти (male-to-male, female-to-female, male-to-female) – використовуються для з’єднання компонентів на макетній платі.

15. USB-кабель – для підключення ESP32 до комп’ютера для прошивки та живлення.

Для ефективної роботи з мікроконтролером ESP32 також важливо розуміти призначення його основних портів та виводів. GPIO (General Purpose Input/Output) є універсальними портами і можуть бути запрограмовані як на вхід, так і на вихід, залежно від потреб проєкту. Вони використовуються, наприклад, для зчитування сигналів з кнопок або для керування світлодіодами. Окрім цифрових входів, ESP32 також має аналогові входи ADC (Analog-to-Digital Converter), які дозволяють

зчитувати аналогові сигнали з датчиків температури, вологості, освітленості та інших подібних елементів. Також є окремі виводи для живлення, які забезпечують стабільну напругу (зазвичай 3.3 В) для підключених компонентів. “Земля” або GND виступає спільним мінусовим контактом і необхідна для замикання електричного кола. Через USB-інтерфейс здійснюється передача даних і живлення мікроконтролера під час програмування або відладки. Деякі піни мають спеціальне системне призначення, наприклад, пін RUN використовується для перезапуску пристрою.

Кожен пін має унікальне позначення (наприклад, GP0, GP1, GP2 тощо), і може поєднувати кілька функцій. Наприклад, пін GP1 може працювати як стандартний цифровий вхід/вихід і водночас бути аналоговим входом ADC1_CH0. Такий функціонал дозволяє створювати універсальні системи на базі ESP32, де один мікроконтролер здатний керувати кількома пристроями одночасно, зчитуючи дані з сенсорів і передаючи команди на виконавчі елементи, наприклад, реле чи двигуни. Розуміти структур пінів і їх ключові моменти дуже важливо для правильної розробки апаратної частини IoT-проєкту [12].

У реальному втіленні системи охорони місцевості ключову роль відіграють різні сенсори, які можуть взаємодіяти з мікроконтролерами та передавати сигнали для подальшої обробки. В даному проєкті сигнали потрібно отримувати від певних пристроїв, тому слід розглянути, яке саме обладнання доцільно було б використати для фізичної реалізації системи.

Одним із ключових елементів у побудові систем охорони є сенсори, які дозволяють виявляти зміни в навколишньому середовищі. Зокрема, широкого застосування набули PIR-сенсори (Passive Infrared Sensor), що працюють на основі виявлення змін інфрачервоного випромінювання. Коли в зону дії такого сенсора потрапляє людина або інший теплокровний об’єкт, пристрій фіксує зміну температурного фону і генерує сигнал. Їх популярність полягає в простоті підключення, низькій вартості та високій сумісності з мікроконтролерами, такими як ESP32. Найпопулярніші моделі, як HC-SR501 або AM312, легко інтегруються у

проєкти, що не потребують складного налаштування. Попри переваги, PIR-сенсори мають і певні обмеження. Наприклад, вони можуть не реагувати на дуже повільні рухи або фіксувати лише об’єкти певного розміру, що може призвести до хибних спрацювань. Також вони не розпізнають тип об’єкта, тому можуть реагувати не лише на людину, а й на тварину. Проте в поєднанні з додатковими сенсорами або програмними фільтрами обробки сигналів (наприклад, аналіз частоти спрацювань чи комбінування з відеоаналітикою), ці датчики можуть значно підвищити ефективність системи безпеки. Найчастіше їх встановлюють по периметру охоронюваної території, де вони виступають першою лінією виявлення вторгнення і запускають ланцюг подій, наприклад, надсилання повідомлення на сервер або активацію сирени.

Окрім інфрачервоних сенсорів, у системах безпеки активно застосовуються герконові датчики, які реагують на зміну магнітного поля. Їх використовують для контролю стану дверей, вікон чи воріт. Принцип роботи простий: коли магніт роз'єднується з герконом, електричний контакт розривається або замикається, що генерує цифровий сигнал. Завдяки своїй надійності, довговічності та простоті геркони підходять для побутових і промислових систем, де необхідно фіксувати факт відкриття або закриття об'єкта. Їх можна без труднощів інтегрувати навіть у найпростіші охоронні пристрої.

Ще одним корисним компонентом сучасних охоронних систем є модулі камер. Вони дозволяють здійснювати відеофіксацію або фотозйомку подій, що відбуваються в зоні спостереження. Особливо зручною у використанні є плата ESP32-CAM, яка поєднує у собі мікроконтролер і камеру. Такий модуль може працювати як автономно, так і у зв'язці з іншими сенсорами. Наприклад, PIR-сенсор фіксує рух, після чого ESP32-CAM робить знімок і надсилає його через WiFi на сервер або користувачеві у месенджер, електронною поштою чи в мобільний застосунок. Це дозволяє оперативно реагувати на загрози в реальному часі.

Крім візуальних та магнітних сенсорів, доцільно використовувати і звукові модулі, які реагують на різкі шуми, удари або характерні звуки, як розбиття скла.

Наприклад, датчики типу KY-038 або MAX9814 здатні виявити зміну рівня шуму в середовищі та ініціювати відповідну реакцію системи. Це може бути запуск сигналізації, активація запису з камери або надсилання повідомлення на мобільний пристрій. Їх особливо доцільно використовувати у місцях, де можливий вандалізм або спроба силового проникнення. У деяких випадках такі датчики можуть працювати як додатковий рівень верифікації події, особливо коли рух або відкриття об'єкта відбувається разом зі звуковим супроводом.

У більш складних системах безпеки може застосовуватись комбінація декількох типів сенсорів, що дозволяє зменшити кількість хибних спрацювань і підвищити загальну точність виявлення. Наприклад, PIR-сенсор можна доповнити герконом і звуковим модулем, створивши багаторівневу систему аналізу подій. Таким чином, рішення про тривогу приймається лише у разі підтвердження з декількох джерел.

Тому в межах даної роботи планується використання PIR-сенсора моделі HC-SR501, оскільки він має просту конструкцію, стабільно працює з поширеними мікроконтролерами по типу ESP32 і Arduino, легко налаштовується за допомогою вбудованих потенціометрів (для регулювання чутливості та часу затримки), а також забезпечує надійне виявлення руху в радіусі до 5-7 метрів. Його енергоефективність, компактні розміри й доступна вартість роблять його оптимальним вибором для побудови базових охоронних систем.

Рисунок 2.2 – Піроелектричний інфрачервоний датчик руху HC-SR501 [12]

Окремо варто розглянути способи передавання зібраної інформації. Завдяки вбудованому WiFi-модулю ESP32 дозволяє надсилати повідомлення безпосередньо через MQTT-брокер або HTTP-запити. У нашій системі як основний протокол було обрано MQTT – легкий, швидкий та ефективний спосіб комунікації між пристроями та сервером. Це дає можливість оперативно передавати повідомлення про виявлений рух або зміну стану будь-якого сенсора.

Усі описані вище компоненти утворюють апаратний рівень програмно-технічного засобу, що є невід'ємною частиною повноцінного IoT-рішення. Таким чином, з технічної точки зору проєкт передбачає роботу з широким набором сенсорів, які можуть бути безпосередньо підключені до мікроконтролера, наприклад ESP32.

На етапі тестування та перевірки логіки роботи було прийнято рішення створити компонент емуляції роботи сенсорів і їх сигналів, з метою перевірки коректності логіки обробки даних, комунікації між пристроями та серверною частиною. Можна створити програмно емульовані MQTT-клієнти, які імітують поведінку реальних сенсорів, вони надсилають повідомлення на сервер з певною періодичністю або у відповідь на задані події, що дозволяє відлагодити всю архітектуру системи до впровадження реальної апартної частини. Це є достатньо ефективним для первинного тестування та демонстрації роботи IoT-системи охорони території в умовах обмежених ресурсів. Такий підхід дозволить протестувати систему без затрат на апаратне забезпечення, при цьому зберігаючи повну можливість фізичного розширення в майбутньому без змін у загальній архітектурі.

2.2 Проєктування архітектури основних модулів системи

Архітектура проєкту побудована з урахуванням масштабованості, модульності та можливості подальшого розвитку. Система реалізована у вигляді розподілених компонентів із чіткими зонами відповідальності, кожен з яких

виконує чітко визначену функцію. В основі системи закладено принцип поділу на логічні рівні:

– рівень збору даних (PIR-сенсори);
– рівень передачі даних (мережевий протокол MQTT);
– рівень обробки та зберігання інформації (backend-сервер);
– рівень представлення результатів кінцевому користувачу (веб-інтерфейс).

Ключовим архітектурним принципом проєкту є його можливість до подальших розширень. Система спроєктована таким чином, щоб у майбутньому було можливо легко додати нові типи сенсорів, збільшити кількість клієнтів, під’єднати додаткові зони моніторингу, чи інтегрувати систему з іншими зовнішніми сервісами. Це досягається шляхом модульної побудови кожного з компонентів, які можуть взаємодіяти між собою за допомогою стандартизованих протоколів, не потребуючи глибокої внутрішньої інтеграції.

Важливою особливістю є також розподілена обробка і поділ на зони функціоналу, замість централізованої логіки, де всі дії виконуються в одному модулі, система реалізує принцип розподіленої відповідальності. Наприклад, пристрої генерують первинні сигнали, MQTT-брокер здійснює маршрутизацію повідомлень, сервер відповідає за логіку обробки, зберігання та передачу даних. Такий підхід дозволяє досягти високої ефективності, зменшити навантаження на окремі вузли та спростити тестування кожного елементу окремо.

Також ще одним важливим архітектурним підходом є асинхронність обміну повідомленнями, реалізована за допомогою протоколу MQTT і його особливостей. Завдяки легкості реалізації та малому об'єму службових даних, цей протокол особливо добре підходить для пристроїв з обмеженими ресурсами. Це дозволяє системі залишатися ефективною, навіть за великої кількості одночасно активних пристроїв, забезпечуючи оперативність передачі даних і мінімальні затримки. Крім того, MQTT природно підтримує механізми масштабування, надійності передачі повідомлень, повторного підключення клієнтів і “QoS-рівні” (якість доставки повідомлень), що робить його ідеальним вибором для систем із IoT-технологіями.

Загальний опис структури системи:

1. MQTT-брокер.

Центральним елементом у системі обміну повідомленнями є MQTT-брокер – сервер, який забезпечує маршрутизацію, прийом і відправку повідомлень між усіма учасниками системи. Саме він виконує роль “посередника” у комунікації між сенсорами і програмною логікою, яка обробляє дані. Його використання дозволяє повністю роз'єднати джерело повідомлення (публікатора – publisher) і його одержувача (підписника – subscriber).

Саме поняття MQTT (Message Queuing Telemetry Transport) – це мережевий протокол публікації-підписки, який був спеціально розроблений для систем з обмеженими ресурсами, таких як вбудовані пристрої та сенсори. Його основні переваги – мінімальний розмір заголовків повідомлень (“headers” – як в HTTP), простота реалізації, підтримка різних рівнів якості доставки (QoS), збереження повідомлень у чергах та підтримка постійних з’єднань із повторними спробами при розриві. Усе це робить MQTT ідеальним рішенням для побудови надійних IoT-систем. У структурі системи MQTT- брокер є компонентом комунікації, саме сюди надсилають повідомлення всі пристрої-клієнти, які імітують поведінку сенсорів. Повідомлення публікуються у відповідні канали зв’язку (топіки – topics), які структуровано описують зміст і контекст повідомлення (наприклад, sensors/zone1/motion або sensors/zone2). Інші компоненти системи, які зацікавлені в цих даних, підписуються на ці топіки, отримуючи лише релевантну інформацію.

MQTT-брокер налаштований на прийом повідомлень зазвичай на стандартному порту “1883” для MQTT-з'єднань, а також у разі потреби може підтримувати WebSocket-з’єднання для інтеграції з браузером або іншим клієнтом. Крім того, сервер може виступати точкою інтеграції з іншими системами або сторонніми сервісами, завдяки доступності клієнтів і можливості створення каналів передачі. Це відкриває шлях до розширення системи, наприклад, через підключення системи телеграм-оповіщення, обчислювальних хмарних сервісів або баз даних реального часу.

Таким чином, MQTT-брокер забезпечує надійну, асинхронну та ефективну передачу даних між усіма компонентами. Його використання дозволяє суттєво спростити логіку комунікацій, зберігаючи при цьому високу продуктивність і гнучкість у розширенні. Без такого посередника реалізація повноцінної IoT-архітектури з великою кількістю джерел сигналів і підписників була б значно складнішою, а взаємодія між компонентами –менш стабільною і передбачуваною.

2. MQTT-клієнти.

В основі побудови системи охоронного моніторингу лежить архітектурна модель, що активно використовує концепцію MQTT-клієнтів, як пристроїв, здатних самостійно підключатися до MQTT-брокера, передавати дані та реагувати на команди. У запропонованій реалізації такі клієнти є фізичними пристроями, побудованими на базі мікроконтролерів, наприклад, ESP32, які взаємодіють із різними типами сенсорів. Програмування таких пристроїв здійснюється за допомогою платформи Arduino, яка забезпечує зручне середовище розробки, високу зручність у конфігуруванні логіки роботи та надійність у передачі даних.

В системі вони діють як автономні компоненти, які здатні самостійно ініціювати передачу повідомлень при виявленні певної події, або працювати в режимі постійного моніторингу з періодичними оновленнями. Кожен із таких клієнтів підключається до централізованого MQTT-брокера і передає дані у вигляді повідомлень, наприклад, у форматі JSON, що містять тип події, ідентифікатор пристрою, часову мітку, а також допоміжну інформацію про інтенсивність спрацювання чи зону дії. Така структура дозволяє централізовано обробляти всі сигнали, зберігати їх у базі даних, і виконувати відповідну реакцію, надсилаючи повідомлення адміністратору або вмикати тривогу.

У системі передбачається, що кожен сенсор, підключений до ESP32, виступає як окремий MQTT-клієнт. Завдяки використанню Arduino-фреймворку, ці пристрої мають можливість легко масштабуватися та конфігуруватися під специфічні потреби системи, включаючи налаштування частоти відправки повідомлень, логіки спрацювання, споживання енергії тощо. Такий підхід дозволяє створити модульну

охоронну інфраструктуру, де кожен клієнт виконує свою функцію в загальній екосистемі без потреби в постійному централізованому керуванні.

Окрім фізичних пристроїв, у проєкті передбачено також застосування програмно емульованих MQTT-клієнтів для навантажувального тестування системи. Такі клієнти можна реалізувати на Node.js платформі із використанням бібліотеки mqtt.js. Вони дозволяють моделювати сценарії високого навантаження, наприклад, одночасне надходження великої кількості повідомлень від різних зон або типів сенсорів. Це дає змогу перевірити надійність MQTT-брокера, обробку подій серверною частиною та загальну стабільність системи при роботі в умовах, наближених до реальних. Тому, MQTT-клієнти, як апаратні, так і програмні – це частина системи, яка виконує ефективну та надійну функцію передачі даних для їх подальшого опрацювання сервером в системі охоронного моніторингу.

3. Серверна логіка.

Ця частина виконує роль обчислень всієї охоронної IoT-системи. Вона відповідає за прийом, обробку, зберігання і подальшу взаємодію з даними, що надходять від MQTT-брокера. Її функціональність охоплює не лише отримання повідомлень, але й фільтрацію, перевірку, трансформацію, збереження у базі даних, а також передачу оновлених даних до клієнтського інтерфейсу для візуалізації. З архітектурної точки зору серверна логіка розділена на декілька модулів, кожен з яких має свою чітку відповідальність.

Основними компонентами серверної логіки є:

– підсистема прийому MQTT-повідомлень, яка підписується на визначені канали зв’язку в сервері повідомлень та постійно очікує вхідні дані;

– логіка обробки, що відповідає за перевірку коректності повідомлень, перетворення даних у внутрішній формат;

– сховище даних, куди записуються всі події, необхідні для подальшого аналізу або представлення для користувачів системи;

– API-інтерфейс, через який веб-клієнт отримує актуальну інформацію про стан системи;

Компонент MQTT-клієнта на сервері реалізовано з використанням відповідних бібліотек, які дозволяють зручно керувати підпискою і автоматично обробляти відновлення з’єднання. Після отримання повідомлення – його передає система в модуль бізнес-логіки. В рамках обробки даних сервер аналізує повідомлення на предмет їх валідності (наявність обов’язкових полів, допустимість значень), виконує попередню агрегацію або маркування, після чого зберігає їх у відповідній таблиці бази даних. Це дозволяє вести повну історію подій системи, виконувати статистичний аналіз або формувати звіти.

Також варто згадати про підготовку даних на сервері для її відображення у веб-інтерфейсі. У відповідь на запити клієнта сервер передає узагальнену або конкретну інформацію щодо стану об’єктів, зон, останніх спрацювань тощо. API-інтерфейс забезпечує взаємодію між логікою представлення і базою через HTTP-запити, з можливістю отримання оновлень у режимі реального часу або за потребою.

Таким чином, серверна логіка в загальній архітектурі забезпечує не лише зв'язок між усіма елементами, а й задає правила функіонування системи, візуалізації подій і тривалого збереження даних для відображення або подальшого аналізу.

4. База даних.

Забезпечує збереження та подальшу доступність усіх подій, які виникають у процесі роботи системи. До неї надходять повідомлення, отримані від сенсорів, які передаються через серверну логіку після попередньої обробки та фільтрації.

У системі база даних виконує декілька важливих функцій: зберігає історію подій, що дозволяє вести облік станів об’єктів охорони у часовому контексті; надає дані для відображення у візуальному інтерфейсі; також, слугує джерелом для подальшої аналітики, виявлення закономірностей або формування звітів. Усі дані зберігаються у структурованому вигляді, що дозволяє легко здійснювати вибірки за часом, типом подій чи джерелами. В результаті, ми маємо стабільний, та зручний

доступ до інформації, навіть у випадку розширення системи або підключення нових пристроїв.

5. Веб-інтерфейс.

Клієнтська частина системи потрібна для взаємодії користувача з охоронною платформою. Саме через цей компонент здійснюється доступ до всієї інформації, яка надходить від сенсорів або емуляторів пристроїв, і саме тут вона набуває візуально зрозумілого вигляду. Візуалізація даних, що генеруються системою, дозволяє оператору або адміністратору швидко оцінити ситуацію, виявити потенційні загрози та оперативно реагувати. Клієнтський інтерфейс забезпечує зручний перегляд усіх подій, що відбуваються в системі, у реальному часі. Інформація, яка зберігається в базі даних або надходить безпосередньо з MQTT-брокера, виводиться у вигляді оновлюваних таблиць, подій, графічних індикаторів або інших інтерактивних елементів. Клієнти можуть не лише спостерігати за змінами у стані об’єктів охорони, але й оцінювати динаміку, інтенсивність сигналів, і навіть локалізацію подій. При наявності великої кількості одночасних подій або підключених пристроїв, інтерфейс повинен залишатися зрозумілим, не перевантаженим і швидкодіючим.

Особливо важливою є підтримка режиму реального часу – інтерфейс має миттєво реагувати на нові сигнали та відображати їх користувачу без потреби в оновленні сторінки або перезавантаженні застосунку. Клієнтська частина також є засобом контролю та управління. У більш складних реалізаціях передбачається не лише пасивне спостереження за подіями, але й можливість керування певними параметрами системи, наприклад, активації чи деактивації охоронних зон, налаштування рівня чутливості датчиків або фільтрації повідомлень за важливістю.

У разі необхідності система може бути розширена на декілька рівнів: від додавання нових сторінок або функціональних блоків до повної інтеграції з іншими сервісами, включно з мобільними пристроями чи зовнішніми системами оповіщення. Таким чином, візуалізація не є лише додатковим елементом, а виступає повноцінною складовою системи, яка виконує як інформаційну, так і

управлінську функцію, і дозволяє перетворити сигнали сенсорів у зрозумілий, візуально структурований вигляд.

При розробці програмно-технічної охоронної системи особливу увагу було приділено не лише технічній реалізації окремих частин системи, а й загальним підходам до проєктування її архітектури. В основному має бути акцент на такі моменти, як відкритість до розширення, модульність, надійність та зрозумілість структури коду. Це дозволяє системі легко адаптуватися до зміни умов, додавання нових функцій або інтеграції з іншими сервісами.

Кожен компонент (MQTT-клієнти, MQTT-брокер, серверна логіка, база даних, клієнтська частина відображення даних) реалізовано як окрему логічну одиницю зі своєю відповідальністю. Це дозволяє розвивати чи замінювати компоненти незалежно один від одного, не порушуючи загальну структуру. Такий принцип особливо важливий для розподілених систем, де зміни в одній частині не повинні впливати на інші. В даній модульності частково реалізовано принципи Service-Oriented Architecture (SOA). Хоча система не є мікросервісною, між її модулями спостерігається чіткий поділ обов’язків і взаємодія за допомогою стандартизованих інтерфейсів, зокрема через мережевий протокол MQTT.

На рівні логіки взаємодії компонентів застосовувався принцип “оповіщення про події” (Event-Driven Architecture). Система побудована таким чином, що пристрої не просто передають дані, а генерують події, які обробляються у режимі реального часу. Це дозволяє оперативно реагувати на будь-які зміни стану сенсорів і передавати дані далі до обробника або інтерфейсу без затримки.

Крім того, у процесі реалізації враховувались і принципи SOLID, що є базовими у об’єктно-орієнтованому програмуванні. Наприклад, принцип єдиної відповідальності (Single Responsibility Principle) допоміг виокремити логіку MQTT-з’єднання в окремі модулі, тоді як принцип відкритості/закритості (Open/Closed Principle) дозволив передбачити можливість розширення функцій без переписування існуючого коду.

Використання цих методологій і принципів допомагає забезпечити не лише технічну якість рішення, але й його стійкість до змін у майбутньому. Навіть у разі значного зростання навантаження або додавання нових функціональних блоків, система зможе адаптуватися без необхідності повної зміни всього коду.

2.3 Організація взаємодії між компонентами системи

Під час проєктування системи також потрібно забезпечити надійний та ефективний спосіб зв’язку між усіма основними компонентами системи. Це зумовлено особливостями IoT-середовища, яке передбачає роботу з великою кількістю розподілених пристроїв, обмеженими обчислювальними ресурсами і необхідністю швидкої передачі даних у реальному часі. У зв’язку з цим у розглянутій системі охорони місцевості була обрана архітектура, що базується на проміжному компоненті-посереднику – MQTT-брокері, який організовує передачу повідомлень між IoT-клієнтами та серверною частиною системи.

Для взаємодії між датчиками використовується MQTT [13]. Це протокол, або набір правил, обміну повідомленнями, що використовується для взаємодії між пристроями. Інтелектуальні датчики, переносні прилади та інші пристрої Інтернету речей (IoT) зазвичай передають і отримують дані по мережах з обмеженими ресурсами, тому цей протокол є оптимальним для IoT-застосунків через свою низьку затримку передачі даних і можливість працювати в мережах з обмеженою пропускною здатністю. MQTT працює за принципом публікації та підписки (publish/subscribe), що суттєво спрощує масштабування системи та підвищує її гнучкість. У розглянутій системі MQTT-брокер, приймає повідомлення від клієнтів і розподіляє їх серед підписників, до яких належить серверна частина обробки всієї логіки додатку. Це дозволяє уникнути прямого зв’язку між окремими клієнтами і бекендом, що значно підвищує стійкість і надійність системи в цілому.

Протокол MQTT став стандартом для передавання даних IoT завдяки переліченим нижче перевагам:

1) Реалізація MQTT на пристрої IoT вимагає мінімальних ресурсів і тому може використовуватися на маленьких мікроконтролерах. Наприклад, мінімальне повідомлення MQTT може складатися всього лише з двох байтів даних. Заголовки повідомлень MQTT також малі, тому ми можемо оптимізувати пропускну здатність мережі.

2) Для налаштування MQTT протоколу на мікроконтролерах потрібна мінімальна кількість коду, для роботи якого потрібно зовсім небагато енергії. Також у протоколі вбудовані функції для забезпечення взаємодії з великою кількістю пристроїв IoT. Тому ми можемо реалізувати протокол MQTT для підключення мільйонів таких пристроїв.

3) MQTT спрощує для розробників завдання шифрування повідомлень і аутентифікації пристроїв і користувачів за допомогою сучасних протоколів аутентифікації, таких як OAuth, TLS1.3, керованих клієнтами сертифікатів та інших протоколів.

4) Хороша підтримка з боку різних платформ розробки і мов програмування також сильно допомагає в налагоджені і вдосконалені систем, що працюють із MQTT протоколом. Деякі мови, наприклад Python, NodeJS забезпечують хорошу підтримку протоколу, тому розробники можуть швидко реалізувати його з мінімальною кількістю коду в додатку будь-якого типу.

Принцип роботи протоколу MQTT описує модель – “publisher-subscriber”. Під час традиційної взаємодії мережею клієнти і сервери зв'язуються між собою напряму (типова “Клієнт-Серверна” архітектура). Клієнти запитують у сервера ресурси або дані, сервер обробляє запит і повертає відповідь. Але MQTT використовує інший шаблон реалізації комунікації, щоб відокремити відправника повідомлення (publisher) від одержувача (subscriber). Взаємодією між відправниками та одержувачами керує третій компонент – MQTT-брокер повідомлень, завдання якого полягає у фільтрації всіч вхідних повідомлень від відправників і відправлення їх відповідним одержувачам.

Тобто основна різниця полягає в тому, що MQTT-брокер відокремлює відправників від одержувачів, що в свою чергу надає багато переваг:

1) Поділ в просторі (відправник повідомлення (publisher) і його отримувач (subscriber) не знають про місцезнаходження один одного в мережі і не обмінюються такою інформацією, як IP-адреса і номер порту для комунікації).

2) Поділ в часі (сторони що взаємодіють між собою не обов'язково мають бути підключені до мережі одночасно для успішної передачі даних. Завдяки архітектурі "publish/subscribe" та ролі MQTT-брокера, повідомлення можуть зберігатися та передаватися асинхронно).

3) Роздільна синхронізація (відправник і отримувач можуть надсилати й отримувати повідомлення, не перериваючи роботу один одного. Наприклад, отримувачу (subscriber) не потрібно очікувати, коли відправник (publisher) надішле повідомлення).

Серверна частина системи підписується на відповідні топіки MQTT-брокера, отримує публікації від датчиків і здійснює комплексну обробку отриманих даних. На цьому етапі відбувається валідація інформації, яка полягає у перевірці коректності та цілісності повідомлень, що надходять, а також визначенні відповідності форматам і діапазонам значень. Далі дані зберігаються у базі даних, що забезпечує можливість подальшого історичного аналізу і звітування. Крім того, серверна логіка формує спеціалізовані повідомлення для відображення у веб-інтерфейсі користувача, які надсилаються у режимі реального часу.

Термін "топік" в даному контексті використовується для фільтрації повідомлень для клієнтів MQTT протоколу. Топіки представлені у вигляді ієрархії, як файли та директорії. Клієнти публікують повідомлення, що містять топік та дані в байтовому форматі, вони визначають формат файлових даних: текстові, бінарні, XML або JSON дані. Клієнт MQTT надсилає повідомлення "subscribe" MQTT-брокеру, щоб отримувати повідомлення із конкретного каналу зв'язку – топіку, який її цікавить (роблять підписку).

MQTT протокол не є RESTful, оскільки передача репрезентативного стану (REST) – це архітектурний підхід до мережевої взаємодії з використанням шаблону зв'язку “request-response” між відправниками та одержувачами. MQTT, навпаки, використовує модель взаємодії “publisher-subscriber” на рівні програми та вимагає стійкого підключення TCP для передачі push-повідомлень. Однак у MQTT версії 5 доданий новий метод, який діє подібно до REST і дозволяє відправнику прикріпити спеціальний топік відповіді (response topic), яку отримувач обробляє як певний запит до нього, а потім генерує назад відповідь.

Комунікація між серверною логікою та клієнтським відображенням в системі, здійснюється за допомогою комбінації HTTP REST API та WebSocket. REST API слугує для стандартних запитів до сервера, зокрема для отримання даних, конфігурації системи або ініціалізації сесій користувача. WebSocket, в свою чергу, забезпечує двонапрямлений зв’язок у реальному часі, завдяки чому клієнт (наприклад, веб-браузер) миттєво отримує оновлення про зміни у стані датчиків і може відобразити їх у вигляді динамічних графіків, сповіщень або інших візуальних елементів. Це дозволяє користувачам системи оперативно реагувати на потенційні загрози та контролювати ситуацію на території без затримок.

Таким чином, логіка зв’язку між компонентами системи є послідовною та структурованою. Спершу MQTT-клієнти генерують і передають інформацію через MQTT-брокер, який гарантує доставку повідомлень серверній частині. Сервер здійснює обробку та зберігання даних, а також транслює їх у вигляді оновлень через WebSocket на клієнт відображення, забезпечуючи актуальну і зручну візуалізацію для кінцевих користувачів. Використання MQTT як посередника дозволяє реалізувати слабке зв’язування між компонентами, що значно підвищує рівень архітектури і взагалі вважається дуже хорошою практикою в розробці програмного забезпечення. Тому, вибір протоколів та організація логіки зв’язку між компонентами системи є ключовими факторами для досягнення високої продуктивності, розширюваності і надійності IoT-системи охорони місцевості із передачею даних на сервер.

2.4 Висновки

В межах другого розділу було здійснено комплексний розгляд складових програмно-технічного засобу, включаючи як апаратну, так і програмну підсистеми. Розкрито логіку їх взаємодії, а також детально охарактеризовано функціональні ролі ключових модулів та інформаційних компонентів, що забезпечують узгоджену роботу системи.

Було детально проаналізовано архітектурні та функціональні особливості розроблюваної IoT-системи охорони місцевості з передачею даних на сервер. Здійснено обґрунтований вибір технологій, компонентів і протоколів комунікації, що забезпечують надійність і адаптивність системи в умовах змінного середовища. Зокрема, визначено доцільність використання MQTT-протоколу як основного засобу зв’язку між клієнтами та серверною частиною, його легка налаштованість, підтримка асинхронного обміну повідомленнями та можливість поділу в часі між відправником і отримувачем дозволяють ефективно реалізовувати обмін даними навіть за умов обмеженого або нестабільного інтернет-з'єднання.

Також проведено опис загальної логіки взаємодії між основними підсистемами: сенсорами, MQTT-брокером повідомлень, сервером і частиною відображення даних. Ретельно проаналізовано спосіб організації обробки та передачі інформації в реальному часі з урахуванням вимог до безпеки, швидкодії та зручності для кінцевого користувача. Таким чином, результати цього розділу стали базовим фундаментом для подальшої практичної реалізації проєкту із архітектурною моделлю, що відповідає сучасним вимогам до розумних охоронних систем з віддаленим доступом і керуванням.

3 ПРОГРАМНО-АПАРАТНА РЕАЛІЗАЦІЯ ІОТ-БАЗОВАНОЇ СИСТЕМИ ОХОРОНИ МІСЦЕВОСТІ ІЗ ПЕРЕДАЧЕЮ ДАНИХ НА СЕРВЕР

3.1 Опис функціонування апаратної частини програмно-технічного засобу

Апаратна частина системи на стадії розробки була концептуалізована як набір реальних IoT-пристроїв, зокрема мікроконтролерів, таких як ESP32, у поєднанні з сенсорами, що відстежують фізичні параметри навколишнього середовища (рух, освітлення, відкриття дверей тощо). Після того як загальна ідея системи охорони місцевості була визначена, стало зрозуміло, що апаратна частина повинна бути здатною працювати автономно, передавати дані без участі людини та при цьому залишатися простою у реалізації. Спочатку я розглядав різні варіанти, але зупинився на тому, щоб побудувати систему навколо реальних фізичних пристроїв – мікроконтролерів, які можуть взаємодіяти з навколишнім середовищем через датчики. Цей підхід дозволяє наблизити проєкт до умов реального використання, навіть, якщо зараз під час розробки використовується лише один тип сенсора. У перспективі такий принцип дозволяє підключити будь-які інші сенсори, які будуть працювати за тим самим підходом: фіксувати подію, формувати повідомлення і передавати його на сервер. Тому я визначився із апаратною частиною, яка справді використовується у сфері розумних пристроїв і систем автоматизації.

Для початку, я перейшов до вибору основного пристрою, який би міг виконувати роль центрального вузла кожного окремого сенсорного модуля. Тут я вирішив працювати з мікроконтролером ESP32. Основна причина цього вибору полягає в тому, що ця плата має вбудовану підтримку WiFi та Bluetooth, а отже, для передачі даних не потрібно додаткових модулів або плат розширення. Крім того, ESP32 підтримує роботу з багатьма датчиками, працює на популярному середовищі програмування Arduino і має достатньо ресурсів (як оперативної пам’яті, так і обчислювальної потужності) для реалізації не лише простих дій, а й складнішої логіки. Ще одним плюсом є доступність плати – вона відносно дешева, тому зручно

використовувати її навіть у кількох екземплярах одночасно для імітації кількох незалежних пристроїв. І що не менш важливо, в Інтернеті є велика кількість документації, прикладів та форумів, присвячених ESP32 [14]. Це значно полегшувало процес розробки і вирішення технічних проблем на ранніх етапах.

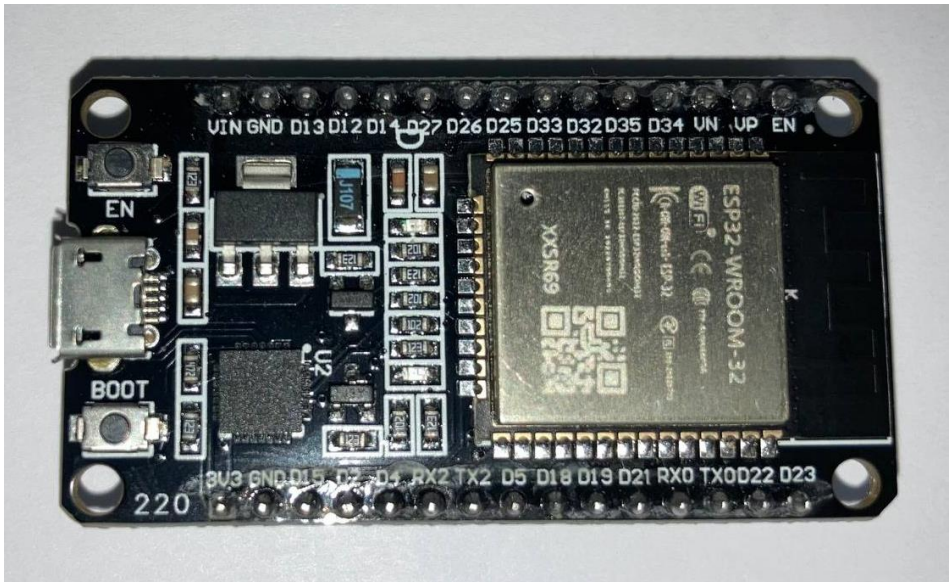


Рисунок 3.1 – Плата ESP32-WROOM-32

Опис основних контактів плати [15]:

1. VIN (Voltage In) – контакт для подачі зовнішнього живлення (зазвичай 5–12 В);
2. 3V3 (3.3 Volts) – стабілізоване живлення 3.3 В, яке використовується для живлення сенсорів і плати;
3. GND (Ground) – загальний провід, або “нуль”, для замикаючого кола;
4. GPIO0–GPIO39 (General Purpose Input/Output) – універсальні цифрові порти, які можуть працювати як вхід або вихід. Деякі з них мають додаткові функції:
   a. GPIO1 (TX0), GPIO3 (RX0) – UART (Universal Asynchronous Receiver-Transmitter) – послідовний інтерфейс для обміну даними між пристроями. TX, RX (Transmit, Receive) – передача і прийом даних по UART;

b. GPIO21 (SDA), GPIO22 (SCL) – I²C. (Inter-Integrated Circuit) – двопровідний інтерфейс для зв'язку з датчиками: SCL – годинниковий сигнал, SDA – дані;

c. GPIO5, GPIO18, GPIO19, GPIO23 – SPI (Serial Peripheral Interface) – швидкий послідовний інтерфейс для обміну даними з модулями/датчиками;

d. GPIO36–GPIO39 – тільки аналоговий вхід (ADC1), читання напруги. ADC (Analog-to-Digital Converter) – перетворює аналоговий сигнал (наприклад, напругу з датчика) у цифрове значення;

5. EN (Enable) – вхід для ввімкнення/перезавантаження мікроконтролера. Підключення до "землі" призводить до перезапуску;

6. BOOT – кнопка для переведення плати у режим прошивки (завантаження коду через USB);

Перед тим як перейти до програмування ESP32, потрібно було підготувати робоче середовище. Першим кроком стало встановлення відповідного драйвера для USB-з'єднання. Це обов'язковий етап, оскільки операційна система мого ноутбука (в даному випадку – Windows 11) спочатку не розпізнавала плату автоматично. Без встановлення драйвера пристрій просто не з'являвся в списку доступних COM-портів, а отже, прошити його було неможливо. На офіційному сайті виробника USB-чипа, який найчастіше використовується в платах ESP32 – це CP2102 або CH340, залежно від модифікації плати можна знайти і встановити потрібний драйвер, і після перезавантаження системи ESP32 з'явиться в диспетчері пристроїв як новий COM-порт.

Далі було встановлено і налаштовано необхідні інструменти написання коду для мікроконтролера. Я вирішив використовувати редактор коду Visual Studio Code у поєднанні з його розширенням PlatformIO. Цей вибір був цілком логічним для мене, бо я вже мав деякий досвід роботи з цим середовищем, а також воно надає багато зручностей при роботі з апаратною частиною. Visual Studio Code сам по собі є легким, проте дуже потужним інструментом для розробника, з широкими можливостями розширення функціоналу за рахунок численних плагінів.

Розширення PlatformIO інтегрується у VS Code як окрема панель, що дозволяє швидко створювати нові проєкти, конфігурувати середовище, компілювати код та завантажувати його на мікроконтролер без потреби виходити з редактора. Важливою перевагою такого підходу є можливість працювати з різними типами плат (ESP32, Arduino, STM32 та інші) у межах одного інтерфейсу.

Платформа PlatformIO побудована таким чином, що розробник отримує повний контроль над проєктною структурою. Наприклад, усі конфігурації середовища зберігаються у спеціальному файлі platformio.ini, де можна вказати модель плати (у моєму випадку - esp32dev), параметри завантаження прошивки, порти, бібліотеки та інші налаштування. Це дає змогу легко керувати великими проєктами або ж створювати кілька варіантів для різних плат, просто перемикаючись між конфігураціями. Однією з найзручніших особливостей є підтримка автоматичного завантаження бібліотек: якщо під час компіляції не вистачає певної бібліотеки, PlatformIO може сам запропонувати її встановлення. Такий механізм суттєво економить час і зменшує ймовірність помилок при ручному додаванні залежностей.

У моєму випадку я вирішив використовувати фреймворк Arduino, який підтримується у PlatformIO. Хоча Arduino IDE також є доволі популярним, вона має низку обмежень, зокрема у зручності керування залежностями та відсутності просунутого інструментарію. На відміну від класичної Arduino IDE, PlatformIO у VS Code дозволяє використовувати усі переваги сучасного середовища розробки: автоматичне завершення коду, підсвітку синтаксису, інтелектуальний аналіз помилок, роботу з Git, інтегровану систему терміналів і можливість налагодження коду. Крім того, VS Code підтримує розширення для роботи з Markdown, JSON, конфігураційними файлами та іншими форматами, що часто використовуються при роботі з IoT-пристроями.

PlatformIO також має зручний вбудований серійний монітор, за допомогою якого можна переглядати вивід мікроконтролера в реальному часі. Це дуже

важливо при налагодженні – можна одразу бачити, що відбувається в системі, чи спрацьовують певні умови, і як саме пристрій реагує на зовнішні подразники.

Такий підхід до розробки з використанням сучасного середовища VS Code, потужної платформи PlatformIO і зручного Arduino Framework значно підвищує продуктивність і дозволяє зосередитися безпосередньо на основній роботі проєкту, а не на технічних дрібницях. Це також спрощує масштабування проєкту в майбутньому, якщо буде потреба додати нові сенсори, підтримку інших мікроконтролерів або інтеграцію з додатковими сервісами. Коли робоче середовище було налаштоване, я створив перший тестовий проєкт для ESP32, щоб перевірити чи все працює коректно. У цьому тесті мікроконтролер просто виводив у серійний монітор повідомлення кожні кілька секунд, таким чином я переконався, що компіляція, завантаження і серійна передача даних працюють належним чином, що дало мені змогу перейти до наступного етапу – налаштування плати і підключення сенсора.

Для реалізації виявлення події я обрав інфрачервоний датчик руху типу PIR HC-SR501. Його перевага полягає в простоті використання, так як він має всього три виводи: живлення (VCC), земля (GND) та цифровий вихід (OUT), який змінює свій стан, коли в полі зору датчика з’являється рухомий об’єкт. Підключення до ESP32 не викликало жодних труднощів: я подав 3.3 В живлення з мікроконтролера на VCC, під’єднав GND до землі плати, а вихід підключив до одного з цифрових входів ESP32 (наприклад, до GPIO14). В Arduino-коді я налаштував відповідний пін на вхід і реалізував логіку, при якій, якщо зчитується високий рівень сигналу, то значить, що виявлено рух, і ця подія передається для подальшої обробки і реєстрації в системі. Окрім цього, модуль HC-SR501 дозволяє регулювати чутливість та час утримання сигналу за допомогою вбудованих потенціометрів, що додає певні опції в налаштуванні, можна адаптувати датчик до різних умов середовища, наприклад, зменшити кількість помилкових фіксацій. Також на модулі передбачений перемикач, що дозволяє вибирати між одноразовим і повторюваним режимами спрацювання. Це корисно, коли потрібно або чекати завершення події,

або безперервно реагувати на рух. Завдяки своїй стабільності, низькому енергоспоживанню та доступності, датчик HC-SR501 є популярним вибором у багатьох проєктах систем безпеки та автоматизації.

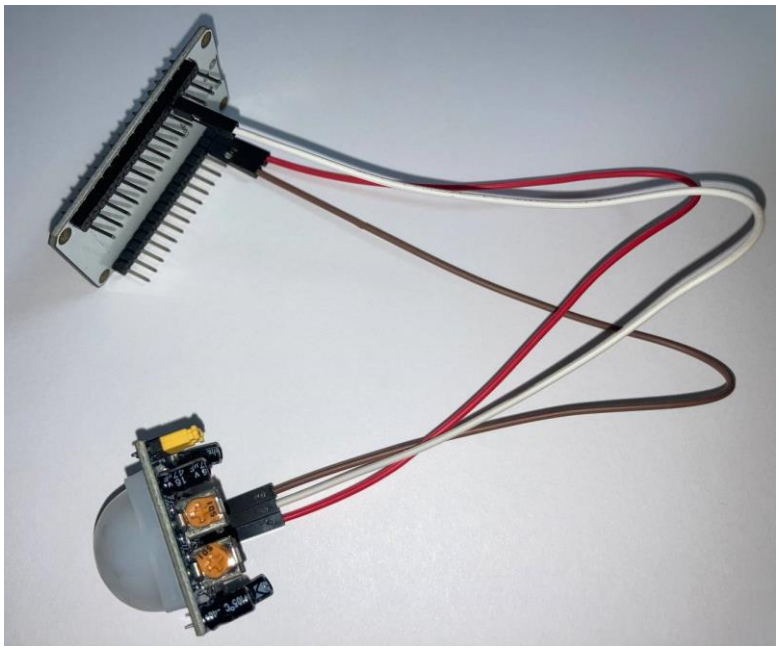


Рисунок 3.2 – Підключення плати ESP32 із PIR датчиком руху

Після того як сенсор було успішно підключено і перевірено, я перейшов до реалізації передачі даних з мікроконтролера до серверної частини. Для цього перш за все потрібно було налаштувати підключення плати ESP32 до локальної WiFi-мережі, щоб мікроконтролер при кожному запуску автоматично намагався під'єднатися до неї. Це дозволяє забезпечити доступ до Інтернету або до локальної мережі, в якій вже запущено MQTT-брокер. Далі я встановив необхідні бібліотеки для роботи з MQTT-протоколом, зокрема, використав бібліотеку PubSubClient, яка є однією з найпоширеніших і добре працює з ESP-платами. Також знадобились бібліотеки для підключення до WiFi та базова підтримка функцій ESP32. Наступним кроком я прописав у коді адресу сервера, порт, на який відправляються повідомлення, та назву каналу зв'язку, саме через який моя плата надсилатиме повідомлення про виявлення руху. На цьому етапі мені вдалося досягти того, що

при кожному спрацюванні датчика ESP32 формувала простий JSON-пакет з інформацією про подію та надсилала його в MQTT-брокер Mosquitto, звідки ці дані вже перехоплювались іншими компонентами системи.

3.2 Програмна реалізація серверної логіки та інтерфейсу керування

Програмна частина системи складається з кількох модулів, кожен з яких виконує лише свою роботу і розділяє відповідальність за працездатність продукту. Центральним елементом є серверна логіка, яка відповідає за прийом повідомлень від MQTT-клієнтів, обробку отриманих даних, та передачу інформації на клієнтську частину. Для забезпечення комунікації між пристроями та сервером застосовується MQTT-брокер повідомлень, побудований за спеціальним протоколом, що забезпечує легку та швидку передачу даних у реальному часі. На рівні MQTT-клієнтів реалізовано генерацію повідомлень, що представляють собою реальні спрацювання охоронних сенсорів. Повідомлення надсилаються на сервер із заданою періодичністю або ж в конкретний момент часу, залежно від сценарію. Такий підхід дозволив перевірити стійкість і працездатність усієї системи при високих навантаженнях і різних режимах роботи.

Для початку слід налаштувати сам MQTT-брокер [16]. Спершу розберемося, які типи взагалі існують і в чому полягають їхні ключові відмінності. MQTT-брокери бувають різних типів залежно від способу розгортання, рівня доступу до налаштувань, потреб у масштабованості та вимог до безпеки.

Розглянемо основні категорії MQTT-брокерів, які застосовуються в сучасних IoT-системах:

1. MQTT-брокери із відкритим вихідним кодом (open-source).

Це сервери, які доступні безкоштовно або на умовах вільної ліцензії. Їх основна перевага полягає у можливості модифікації програмного забезпечення відповідно до конкретних потреб розробника або організації. Вони активно підтримуються спільнотами розробників і часто застосовуються для створення

прототипів, дослідницьких або персональних проєктів, а також у малих та середніх за масштабом системах. Прикладом такого серверу є Eclipse Mosquitto, який є одним із найпопулярніших open-source рішень у сфері MQTT.

2. Хмарні (cloud-based) MQTT-брокери.

Даний тип розгортається не на локальному обладнанні, а на віддалених серверах, які обслуговуються провайдерами хмарних сервісів. Основною перевагою таких MQTT-брокерів є легка масштабованість, відсутність потреби в обслуговуванні фізичної інфраструктури та висока доступність сервісу. Хмарний тип ідеально підходить для розгортання великих IoT-систем, у яких важливо забезпечити безперервну обробку великого обсягу даних. Прикладами таких рішень є AWS IoT Core від Amazon або Azure IoT Hub від Microsoft.

3. Локальні (on-premises) MQTT-брокери.

Такі сервери встановлюється на власних серверах організації або пристроях кінцевого користувача. Їх головна перевага – повний контроль над середовищем роботи, доступом, безпекою та конфігурацією. Такі MQTT-брокери часто застосовуються в організаціях, які мають суворі вимоги до захисту даних або працюють в умовах обмеженого або відсутнього доступу до Інтернету. Цей підхід дозволяє повністю автономно керувати передачею MQTT-повідомлень, що може бути критично важливо для промислових або державних систем. Як приклад, можна згадати розширені (Pro) версії MQTT-брокерів, зокрема Mosquitto Pro Edition.

4. Корпоративні (enterprise) MQTT-брокери.

Це комерційні рішення, які призначені для використання у великих підприємствах або проєктах з підвищеними вимогами до надійності, безпеки, продуктивності та технічної підтримки. Такі сервери часто включають додаткові функції, як підтримка кластеризації, балансування навантаження, шифрування трафіку, розширений моніторинг, аналітика та інтеграція з іншими корпоративними системами. Вони підходять для критично важливих додатків, де недопустима втрата даних чи простої системи. До таких рішень також може

належати Mosquitto Pro Edition, але в контексті розгорнутого корпоративного використання з відповідними ліцензіями й підтримкою.

Тому на основі приведеної інформації було прийняте ріщення використати таку реалізацію MQTT-брокера, як Mosquitto. Це легкий та ефективний сервер для обробки повідомлень, який реалізує протокол MQTT у версіях 5.0, 3.1.1 та 3.1. Він поширюється з відкритим вихідним кодом відповідно до ліцензій EPL та EDL, що дозволяє як вільне використання, так і модифікацію під конкретні потреби розробника. Завдяки своїй малій ресурсомісткості Mosquitto придатний для розгортання як на повноцінних серверних платформах, так і на малопотужних пристроях, таких як одноплатні комп'ютери (наприклад, Raspberry Pi) або вбудовані мікроконтролерні системи. Його використання є особливо доцільним у сфері інтернету речей (IoT), де потрібна надійна комунікація міжпристроями з обмеженою обчислювальною потужністю та енергоспоживанням. Крім основного функціоналу, проєкт Mosquitto також включає в себе бібліотеку на мові C, що дозволяє розробляти власні MQTT-клієнти, а також надає набір інструментів командного рядка. Ці утиліти широко використовуються для відлагодження, тестування або побудови сторін комунікації у системах на основі протоколу MQTT. Така широка підтримка робить Mosquitto одним із найбільш популярних рішень для реалізації серверної частини у розподілених IoT-системах.

В процесі розгортання компонентів IoT-системи все більшої популярності набуває використання контейнеризації – технології, яка дозволяє ізолювати програмне середовище та забезпечити його портативність і стабільність. Одним з найпоширеніших інструментів у цій галузі є Docker – платформа, яка дає змогу запускати програмні компоненти у вигляді легких, ізольованих контейнерів [17]. Docker-контейнер містить у собі все необхідне для роботи конкретного застосунку: саму програму, її залежності, конфігураційні файли, бібліотеки та середовище виконання. Це дозволяє досягнути максимальної стабільності при розгортанні, оскільки відпадає потреба у встановленні кожного компонента окремо або турботах про сумісність середовищ на різних машинах.

Mosquitto доступний як офіційний Docker-образ, що дозволяє розгорнути готовий до використання MQTT-брокер за лічені хвилини за допомогою простих конфігурацій. Сам сервер працює у власному контейнері, ізольованому від решти системи і це дозволяє уникнути конфліктів між залежностями та підвищує безпеку. Його налаштування можуть бути винесені у зовнішні файли і підключені як певні томи (Docker Volumes), які синхронізують файлову систему на локальній машині і в контейнері, що дозволяє легко змінювати конфігурацію без перекомпіляції або втручання в сам контейнер [18].

Таким чином, використання Docker для розгортання MQTT-брокера спрощує процес налаштування та тестування системи, і забезпечує її стабільну роботу в різноманітних середовищах, сприяє кращій підтримці, супроводу та масштабуванню IoT-рішення загалом. Тому в рамках всього проєкту технологія Docker-контейнеризації активно використовується для розгортання і налаштування кожної підсистеми.

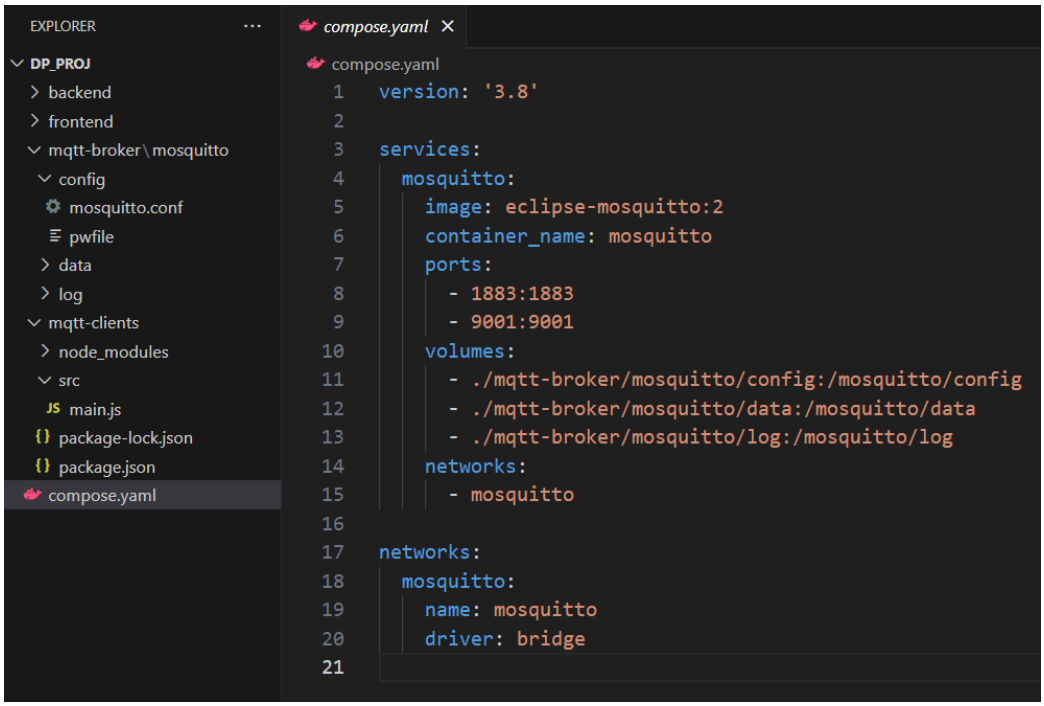


Рисунок 3.3 – Налаштування Docker-контейнера для Mosquitto MQTT-брокера

Після запуску Mosquitto в Docker-контейнері, я додатково налаштував його внутрішню конфігурацію, яка зберігається у файлі спеціальному конфігураційному файлі. Основні параметри, що підлягали редагуванню, стосувалися портів, безпеки та обмежень для підключення клієнтів. Наприклад, я визначив стандартний порт для з'єднань, а також прописав шлях до текстових файлів, які зберігають хронологічно події та інформацію про роботу системи, щоб відстежувати MQTT-події у процесі налагодження. Крім того, я встановив обмеження на розмір повідомлень і вказав шлях до файлу паролів (для автентифікації клієнтів у майбутньому), а також ввімкнув зберігання сеансів та ретрансляцію останніх повідомлень, що особливо корисно при розробці IoT-систем, де клієнти можуть періодично відключатися.

Також я створив окрему директорію для конфігурацій і підключив її до контейнера через "Docker Volume". Це дозволило зберігати й редагувати конфігурацію сервера поза межами самого контейнера і в результаті сильно спростило процес оновлення і налагодження програми, без необхідності щоразу перебудовувати її.

Коли MQTT-брокер був налаштований і стабільно працював, я перейшов до реалізації серверної частини. В межах цієї частини системи було реалізовано декілька ключових компонентів: модуль для обробки вхідних MQTT-повідомлень, модуль для збереження отриманих даних у базу даних (з використанням Spring Data JPA), а також REST API для взаємодії з клієнтською частиною. Повідомлення, що надходять від ESP32, обробляються спеціальним сервісом, який фільтрує й трансформує їх у події системи безпеки. Всі події мають відповідну структуру та зберігаються в базі для подальшого перегляду та аналізу.

Для написання основного серверу системи я обрав мову Java та фреймворк Spring Boot. Такий вибір зумовлений тим, що цей інструмент дозволяє швидко розгортати готові вебсервіси з чіткою структурою, великою кількістю готових бібліотек, зручною роботою з базами даних, а також високою надійністю. Зважаючи на те, що сервер повинен обробляти повідомлення з MQTT-брокера,

зберігати події в базі даних і забезпечувати взаємодію з клієнтським інтерфейсом, даний фреймворк став цілком відповідним рішенням для цих задач.

Spring Boot є сучасним фреймворком для створення автономних Java-додатків, який значно спрощує розробку вебсервісів і API, дозволяючи зосередитись безпосередньо на основному функціоналі застосунку. Однією з головних його переваг є принцип “convention over configuration” – розробнику не потрібно вручну налаштовувати десятки конфігураційних файлів, усе налаштовано і працює відразу. Це дало змогу дуже швидко підняти робочий сервер, підключити всі необхідні залежності та зосередитись на вирішенні основних задач. Даний інстумент базується на основі популярного фреймворку Spring, але значно спрощує його використання, усуваючи складність початкового налаштування. Він автоматично підключає потрібні залежності, конфігурує середовище за типовими шаблонами і дозволяє зосередитися на головних моментах роботи системи, а не на технічних деталях. Однією з важливих особливостей є вбудований вебсервер (наприклад, Tomcat або Jetty), завдяки чому застосунки не потребують зовнішнього сервера і їх можна запускати як звичайні програми, що значно полегшує розгортання.

Spring Boot також підтримує просту інтеграцію з базами даних, системами безпеки, чергами повідомлень, хмарними сервісами та іншими технологіями. Багато поширених функцій доступні через стартери (попередньо упаковані набори залежностей), які автоматично додають відповідні бібліотеки та базові конфігурації. Крім того, фреймворк надає зручний механізм тестування, моніторингу стану застосунку, а також потужну систему логування. Завдяки активній спільноті та великій кількості документації, Spring Boot став одним із найпопулярніших інструментів у Java-екосистемі. Його використовують як у невеликих проєктах, так і у великих корпоративних рішеннях. Він добре масштабується, має зрозумілу архітектуру і підтримує мікросервісний підхід, що робить його універсальним вибором для сучасної серверної розробки.

У рамках мого проєкту Spring Boot виконує кілька важливих функцій. По-перше, він виступає як компонент, що приймає всі MQTT-повідомлення, що надходять з пристроїв, а точніше, сервер не працює напряму із сенсорами, а взаємодіє із ними через проміжний компонент – MQTT-брокер. Завдяки своїй модульній структурі, я зміг легко підключити зовнішню бібліотеку для роботи з MQTT-протоколом (наприклад, Eclipse Paho), і налаштувати підписку на певні канали зв'язку, які відповідають окремим пристроям. Повідомлення одразу передаються у відповідний сервіс, де відбувається їх обробка: виділення ключових полів (тип події, час, ідентифікатор пристрою), логування, а також збереження в базу даних. По-друге, я реалізував рівень збереження даних на основі Spring Data JPA. Це ще один модуль Spring, який дозволяє швидко створювати повноцінну структуру для взаємодії з базами даних. Замість того, щоб писати SQL-запити вручну, я описав сутності у вигляді Java-класів, а репозиторії, як інтерфейси. Завдяки цьому я можу працювати з базою дуже просто, наприклад, спеціальним методом я можу автоматично виконати пошук по відповідному полю, без потреби написання додаткового коду. Уся ця зручність дозволяє ефективно масштабувати систему в майбутньому і додати нові таблиці для користувачів, пристроїв і різних подій. По-третє, сервер на Spring Boot надає REST API для взаємодії з клієнтським інтерфейсом. Я створив окремий контролер, який на запит від клієнта (наприклад, React-додатку) повертає список пристроїв або історію подій. Spring MVC, який входить до складу Spring Boot, має дуже зручну модель для створення REST-контролерів: потрібно лише оголосити клас із відповідними анотаціями (спеціальні маркери, що надають певні метадані), та методи з прив'язкою до конкретного URL, за якими можна отримати доступ до функцій або даних, які пропонує API. Ця простота дуже доречна в умовах проєкту, де важлива швидкість реалізації.

Окремо слід відзначити, що Spring Boot має чудову підтримку для налаштування конфігурацій через свої спеціальні конфігураційні файли. Я використовував цей механізм для збереження параметрів з'єднання з базою даних, MQTT-параметрів, портів, CORS-налаштувань і логування. Це дозволяє зручно

змінювати поведінку програми без потреби перекомпіляції, що особливо зручно під час тестування і розгортання.

У підсумку, Spring Boot дав мені можливість реалізувати потужний і розширюваний сервер, який без проблем можна доповнити новими функціями, наприклад, додати автентифікацію користувачів, розмежування прав доступу, інтерфейс моніторингу чи навіть інтеграцію зі сторонніми сервісами. Його стабільність, велика спільнота і активна підтримка роблять його ідеальним вибором для будь-яких серйозних серверних застосунків, у тому числі для IoT-систем.

application.yaml

```yaml
server:
  port: 8080

spring:
  datasource:
    driver-class-name: org.postgresql.Driver
    username: guarduser
    password: 123qwe
    url: jdbc:postgresql://localhost:5400/securitydb
  jpa:
    show-sql: true
    hibernate:
      ddl-auto: validate
  liquibase:
    change-log: classpath:db-migrations/changelog/changelog-master.yaml

mqtt:
  broker-url: tcp://localhost:1883
  client-id: BackendClient
  username: user
  password: qweqwe
  topic: devices/+/events
```

Рисунок 3.4 – Налаштування основного серверного застосунку системи на базі Spring Boot фреймворку

Крім обробки MQTT і збереження даних, сервер також відповідає за надання інтерфейсу керування у вигляді вебсторінки. Для реалізації візуальної частини проєкту я використав JavaScript з бібліотекою React, яка дозволяє створювати динамічні й зручні інтерфейси, а також легко оновлювати окремі частини сторінки без перезавантаження. На даному етапі був реалізований базовий візуал у вигляді

адміністративної панелі, яка відображає список підключених пристроїв, отримані події, а також дозволяє переглядати останні сигнали, що надходили від сенсорів. Весь інтерфейс розміщено на окремому сервері, який взаємодіє з іншими компонентами системи через REST API.

Вибір бібліотеки React зумовлений її популярністю, та величезною кількістю готових компонентів, які значно спрощують розробку. Даний інструмент підтримує розділення інтерфейсу на компоненти (невеликі частини, кожна з яких відповідає за окрему частину функціональності). Це дозволяє створювати масштабовані інтерфейси, які легко підтримувати та змінювати. Наприклад, таблиця з пристроями чи список подій реалізовані як окремі компоненти, що отримують дані через запити до API і автоматично оновлюються при зміні стану.

Для стилізації інтерфейсу я використав базові CSS-інструменти, а також бібліотеки інтерфейсних компонентів, які дозволяють швидко зібрати привабливий і сучасний дизайн. Завдяки цим інструментам вдалося реалізувати адаптивну верстку, яка коректно відображається як на великих моніторах, так і на портативних пристроях. Панель адміністратора виконана в мінімалістичному стилі, де на головному екрані відображається список усіх пристроїв з їхнім статусом, часом останнього сигналу, і є можливість перегляду детальнішої інформації.

Бібліотека React також зручно поєднується з інструментами розробника, такими як Webpack і Babel, що дозволяють збирати проєкт, автоматично оновлювати сторінку при зміні коду та слідкувати за помилками під час розробки. Крім того, використання такого інстументу дозволяє у майбутньому легко реалізувати складніший функціонал, наприклад, інтерактивну карту розташування пристроїв, налаштування сповіщень, вхід користувача, зміну прав доступу тощо. Таким чином, використання React для реалізації візуальної частини проєкту дало змогу створити легкий у використанні, зрозумілий та сучасний інтерфейс, який можна легко масштабувати й доповнювати новими можливостями. Поки реалізовано лише базову версію панелі, але вона вже виконує свої основні функції і слугує основою для подальшого розвитку програмно-технічного засобу.

Кінцевий результат програмної реалізації серверної логіки та інтерфейсу керування – це функціональна система, де кожен компонент виконує свою роль у загальному процесі збору, обробки та представлення даних, водночас залишаючись незалежною частиною і здатною до окремого налаштування чи вдосконалення. Завдяки такому підходу систему можна легко масштабувати й доповнювати новими можливостями.

3.3 Опис процесу створення баз даних

Для зберігання усієї необхідної інформації в межах проєкту було реалізовано реляційну базу даних на основі PostgreSQL, яку я розгортаю локально в Docker-контейнері, щоб спростити її запуск і забезпечити однакові умови для різних середовищ виконання коду. Для управління структурою бази даних та організації міграцій інформації було обрано інструмент Liquibase, який дозволяє безпечно оновлювати схему на всіх етапах розробки [19]. Структура бази даних описана у вигляді SQL-скриптів, згрупованих у спеціальні файли (changeset files), що зберігаються окремо від коду застосунку, але підключаються під час запуску Spring Boot сервера.

Насамперед я створив контейнер з базою даних за допомогою Docker інструменту і використавши систему управління PostgreSQL [20]. Для цього мені потрібно було написати відповідну конфігурацію у файлі docker-compose.yml, де вказуються необхідні параметри – образ бази даних, обсяг томів для збереження даних, ім’я контейнера, порт, на якому працюватиме програма, логін і пароль для підключення. Такий підхід дозволяє швидко створити середовище із потрібними сервісами без потреби встановлювати PostgreSQL локально, а також можна легко переносити чи виконувати його на інших машинах. Окрім цього, ізоляція контейнера гарантує, що база працюватиме стабільно й не конфліктуватиме з іншими сервісами на комп’ютері.

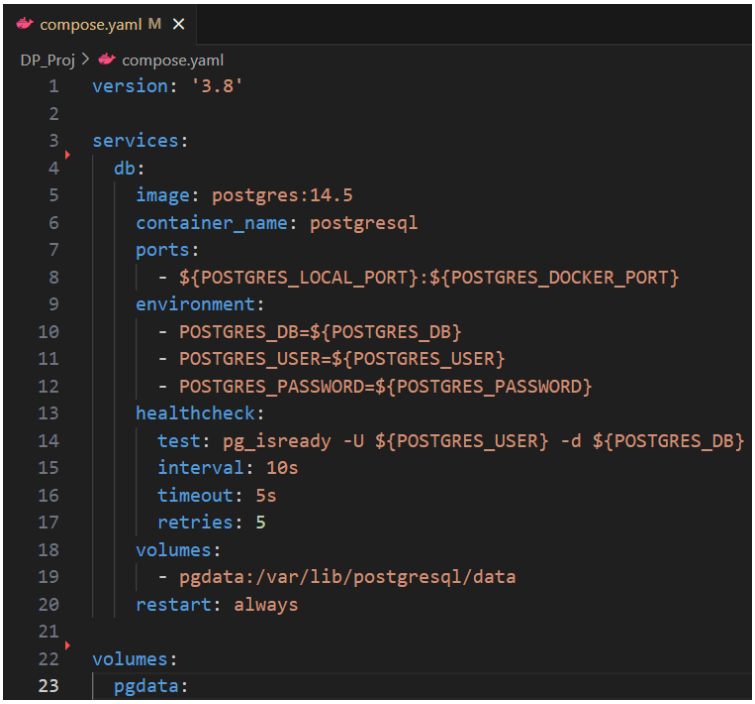


Рисунок 3.5 – Налаштування PostgreSQL в Docker-контейнері

Після того як контейнер із базою даних був налаштований і успішно запущений, я перейшов до проєктування основних сутностей, що відображають логічну структуру системи. На цьому етапі важливо було правильно визначити об’єкти, які будуть зберігатися в базі даних, їхні атрибути та взаємозв’язки між ними. Я орієнтувався на реальні компоненти IoT-системи, тому створив моделі для користувачів, їх ролей, пристроїв, подій, зон місцевості, та локацій спостереження.

В даному проєкті використано певні сутності (entity), кожна з яких представляє собою важливий і ключовий компонент серверної логіки. Всі вони були описані в власному SQL-файлі міграцій Liquibase. Нижче наведу докладний опис кожної з цих сутностей:

1. app_users

Сутність “app_users” відповідає за зберігання інформації про користувачів системи, включаючи ім’я, пароль та токен для ідентифікації в системі. Також містить зв’язки з ролями, які визначають права доступу.

SQL-скрипт для створення таблиці app_users:

```
CREATE TABLE IF NOT EXISTS app_users (
```

```
    id BIGINT,
    username VARCHAR(255) NOT NULL,
    password VARCHAR(255) NOT NULL,
    token VARCHAR(255) NOT NULL,
    CONSTRAINT pk_app_users PRIMARY KEY (id),
    CONSTRAINT unique_users_token UNIQUE (token)
);
```

2. roles

Сутність “roles” містить ролі користувачів (наприклад, ADMIN або USER), які дозволяють розмежувати доступ до функцій системи залежно від прав.

SQL-скрипт для створення таблиці roles:

```
CREATE TABLE IF NOT EXISTS roles (
    id BIGINT GENERATED BY DEFAULT AS IDENTITY,
    name VARCHAR(255) NOT NULL,
    CONSTRAINT pk_roles PRIMARY KEY (id)
);
```

3. locations

Сутність “locations” представляє певне місце, яке створюється і контролюється користувачем. Вона має зв’язок із конкретним користувачем і може містити кілька своїх зон спостереження.

SQL-скрипт для створення таблиці locations:

```
CREATE TABLE IF NOT EXISTS locations (
    id BIGINT,
    name VARCHAR(255) NOT NULL,
    is_active BOOLEAN NOT NULL DEFAULT 'true',
    user_id BIGINT,
    CONSTRAINT pk_locations PRIMARY KEY (id),
    CONSTRAINT fk_locations_users FOREIGN KEY (user_id)
REFERENCES app_users (id)
);
```

4. areas

Сутність “areas” описує окремі зони (наприклад, кімнати або ділянки території) в межах заданої локації. Вона є дочірньою сутністю до “locations” та містить пристрої, що являють собою сенсори руху.

SQL-скрипт для створення таблиці areas:

```
CREATE TABLE IF NOT EXISTS areas (
    id BIGINT,
    name VARCHAR(255) NOT NULL,
    location_id BIGINT,
    CONSTRAINT pk_areas PRIMARY KEY (id),
    CONSTRAINT fk_areas_locations FOREIGN KEY (location_id) REFERENCES locations (id)
);
```

5. devices

Сутність “devices” зберігає інформацію про підключені пристрої, містить свій тип (наприклад, датчик руху), назву яку встановлює користувач, стан активності та унікальний токен для ідентифікації в системі серед інших пристроїв.

SQL-скрипт для створення таблиці devices:

```
CREATE TABLE IF NOT EXISTS devices (
    id BIGINT,
    name VARCHAR(255) NOT NULL,
    type VARCHAR(255) NOT NULL,
    is_enabled BOOLEAN NOT NULL DEFAULT 'true',
    token VARCHAR(255) NOT NULL,
    area_id BIGINT,
    CONSTRAINT pk_devices PRIMARY KEY (id),
    CONSTRAINT fk_devices_areas FOREIGN KEY (area_id) REFERENCES areas (id),
    CONSTRAINT unique_devices_token UNIQUE (token)
);
```

6. events

Сутність “events” використовується для реєстрації подій, які надсилають пристрої, вона включає тип події, час, текстовий опис (payload) та зв’язок із конкретним пристроєм.

SQL-скрипт для створення таблиці events:

```
CREATE TABLE IF NOT EXISTS events (
    id BIGINT GENERATED ALWAYS AS IDENTITY,
    type VARCHAR(255) NOT NULL,
    timestamp TIMESTAMP,
    payload VARCHAR(255),
    device_id BIGINT,
    CONSTRAINT pk_events PRIMARY KEY (id),
    CONSTRAINT fk_events_devices FOREIGN KEY (device_id) REFERENCES devices (id)
);
```

7. users_roles

Сутність “ users_roles” використовується для того, щоб зберігати інформацію про зв’язок конкретного користувача із конкретною ролю в системі саме у вигляді окремої таблиці

SQL-скрипт для створення таблиці users_roles:

```
CREATE TABLE IF NOT EXISTS users_roles (
    user_id BIGINT NOT NULL,
    role_id BIGINT NOT NULL,
    CONSTRAINT composite_pk_users_roles PRIMARY KEY (user_id, role_id),
    CONSTRAINT fk_user FOREIGN KEY (user_id) REFERENCES app_users (id),
    CONSTRAINT fk_role FOREIGN KEY (role_id) REFERENCES roles (id)
);
```

В проєкті для взаємодії з базою даних застосовується модуль Spring Data JPA, який значно полегшує роботу з об'єктно-реляційним відображенням. Завдяки такому інструменту можна швидко і зручно працювати з сутностями бази даних,

без написання складного SQL-коду. Java Persistence API (JPA) є стандартом Java Enterprise Edition (EE), що забезпечує відображення об’єктів Java у таблиці бази даних і навпаки. Spring Data JPA дозволяє створювати інтерфейси репозиторіїв, а реалізація більшості CRUD-операцій генерується автоматично, достатньо лише оголосити метод з правильно сформульованою назвою, а фреймворк сам згенерує необхідний запит до бази даних. Ще однією перевагою даного інструменту є висока абстракція, яка дозволяє фокусуватись на основних задачах, замість низькорівневої реалізації доступу до даних.

Завдяки використанню модуля Spring Data JPA процес створення сутностей був доволі зручним, кожен клас у Java автоматично відображався у відповідну таблицю в базі даних, а зв’язки між об’єктами задавалися за допомогою спеціальних конструкцій в коді.

Для керування структурою бази даних у проєкті використовується ще Liquibase – інструмент, що дозволяє відстежувати та застосовувати зміни до схеми БД у вигляді послідовних міграцій. Основний його файл конфігурації містить посилання на окремі changelog-файли, кожен з яких відповідає за створення чи оновлення окремої сутності. Тому ми можемо точно контролювати всі зміни в структурі бази даних, і маємо стабільну роботу застосунку.

3.4. Висновки

У межах розділу 3 представлено реалізацію програмно-апаратного засобу IoT-базованої системи охорони місцевості із передачею даних на сервер. Було розглянуто опис як апаратної частини, що включає підключення датчика руху до плати ESP32 і програмування самого мікроконтролера із всіма його особливостями, так і серверну частину, яка забезпечує приймання даних, їх обробку та збереження для подальшого аналізу і відображення користувачам. Також проведено налаштування необхідних сервісів, таких як система управління базами даних PostgreSQL і MQTT-брокер, із застосуванням Docker-контейнеризації.

Під час реалізації системи було враховано як апаратні, так і програмні аспекти функціонування. Основною метою цього етапу стало створення стабільного і працездатного продукту, який вміє ефективно приймати, обробляти та візуалізувати дані від сенсорних пристроїв. Ключові можливості системи впроваджені згідно з визначеними вимогами і на даному етапі забезпечують її повну працездатність. Архітектура проєкту спроєктована з урахуванням можливості масштабування, тому в подальшому її легко розширити новими модулями, пристроями, чи логікою обробки подій. Виконання цього етапу підтверджує практичну доцільність обраного підходу до розробки.

ВИСНОВКИ

У роботі за результатами виконаних теоретичних та практичних досліджень було розроблено програмно-апаратний засіб IoT-базованої системи охорони місцевості із передачею даних на сервер.

У першому розділі було здійснено огляд існуючих технологій, архітектур та рішень, що застосовуються в галузі систем безпеки з використанням Інтернету речей. Проведено аналіз типових підходів до реалізації систем охорони, засобів передачі даних та способів інтеграції апаратної та програмної частин. На основі цих даних було сформульовано підходи до вирішення задачі за темою дослідження, та визначено найбільш доцільні методи для виконання роботи.

У другому розділі проведено проєктування основних модулів, визначено сруктуру системи і розглянуто різні способи взаємодії між її компонентами. Значну увагу приділено вибору відповідних засобів комунікації, які дозволяють відтворити стабільну роботу системи в умовах із можливими мережевими обмеженнями. У процесі визначення способів взаємодії між компонентами було аргументовано використання зручного та легкого для налаштування MQTT протоколу, здатного підтримувати обмін повідомленнями у режимі, що не потребує постійного з’єднання між сторонами. У цьому розділі також детально розглядаються різні компоненти, для створення апаратної частини IoT-базованої системи охорони. Особливу увагу приділено технічним засобам, які виконують роль сенсорів і безпосередньо виявляють зміни у навколишньому середовищі, збирають необхідні дані і передають їх на серверну частину системи для подальшої обробки та аналізу. Крім того, розглядаються принципи взаємодії датчиків з мікроконтролерами, їх різновиди та особливості. Також проєктувалась логічна модель взаємодії між сенсорними пристроями, MQTT-брокером, серверною частиною та інтерфейсом візуалізації, де чітко визначено шляхи передачі даних, ролі кожного компонента в загальній системі та принципи їх роботи.

У третьому розділі було детально описано процес реалізації і функціонування апаратної і програмної частини системи згідно з розробленою архітектурою та технічними вимогами. Наведені конкретні приклади і способи реалізації основного функціоналу, які включають в себе всі налаштування ключових модулів, їх особливості і методи застосування. Було детально описано процес створення бази даних і кожної її функціональної сутності, а також використано інструменти для забезпечення надійності даних і зручного налаштування системи управління базами даних. На кожному етапі розробки наведено опис і застосування використаних бібліотек, фреймворків і всіх засобів залучених для реалізації системи. Таким чином, третій розділ затвердив практичну доцільність обраних архітектурних і технічних рішень. Реалізована система повністю відповідає поставленим вимогам і демонструє потенціал до подальшого масштабування, впровадження нових функцій і використання в реальних умовах для підвищення ефективності охорони об’єктів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Як працює Internet of Thing. URL: https://weagro.com.ua/blog/internet-rechej-iot-shho-cze-ta-jogo-vykorystannya-v-silskomu-gospodarstvi/ (дата звернення: 07.02.2025).
2. Інтернет речей (IoT): де і як використовується? URL: https://lanmarket.ua/news/internet-rechey-iot-v-ukraini-de-i-yak-vikoristovu-tsya/ (дата звернення: 09.02.2025).
3. Принципи застосування Інтернет речей у сучасному світі техніки. URL: https://www.tech.vernadskyjournals.in.ua/journals/2020/6_2020/part_1/26.pdf (дата звернення: 07.02.2025).
4. Схема типової мережі LoRaWAN. URL: https://iotji.io/osoblyvosti-lorawan/ (дата звернення: 09.02.2025).
5. Ajax системи безпеки. URL: https://ajax.systems/ua/product-categories/intrusion-protection/ (дата звернення: 07.02.2025).
6. Google Next технології. URL: https://store.google.com/us/category/connected_home?hl=en-US (дата звернення: 09.02.2025).
7. Reolink пристрої. URL: https://reolink.in.ua/osnovni-osoblivosti-kamer-reolink-duo-2 (дата звернення: 09.02.2025).
8. Компоненти IoT-базованої системи і їх взаємозв'язок. URL: https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82_%D1%80%D0%B5%D1%87%D0%B5%D0%B9 (дата звернення: 12.02.2025).
9. Java and the Spring framework. URL: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-java-spring-boot (дата звернення: 12.02.2025).
10. ESP32 Basic Starter Kit і його вміст. URL: https://ua.sz-kuongshun.com/uno/aruino-kit/esp32-basic-starter-kit-wifi-iot-development.html (дата звернення: 12.02.2025).

11. Lafvin. ESP32 BasicStarterKit. Overview and detailed description of the work. URL: https://www.mikroshop.ch/pdf/ESP32_Start_Kit.pdf (дата звернення: 12.02.2025).

12. Піроелектричний інфрачервоний датчик руху HC-SR501. URL: https://rtp.com.ua/catalog/arduino/datchyky-ta-dodatkovi-moduli/hc-sr501-datchyk-rukhu-dlja-arduino (дата звернення: 23.02.2025).

13. Опис протоколу MQTT і принцип його роботи. URL: https://aws.amazon.com/what-is/mqtt/ (дата звернення: 05.03.2025).

14. Get started with ESP32-DevKitC: debugging and project analysis. URL: https://docs.platformio.org/en/latest/tutorials/espressif32/espidf_debugging_unit_testing_analysis.html (дата звернення: 05.03.2025).

15. Розташування та призначення контактів плати ESP32. URL: https://esp32.implrust.com/esp32-intro/pinout.html (дата звернення: 23.02.2025).

16. Документація Mosquitto MQTT-брокера. URL: https://mosquitto.org/ (дата звернення: 05.03.2025).

17. Docker Desktop. URL: https://docs.docker.com/desktop/use-desktop/ (дата звернення: 05.03.2025).

18. Налаштування Mosquitto MQTT-брокера в Docker контейнері. URL: https://cedalo.com/blog/mosquitto-docker-configuration-ultimate-guide/ (дата звернення: 05.03.2025).

19. Документація системи управління міграціями бази даних Liquibase. URL: https://docs.liquibase.com/home.html (дата звернення: 05.03.2025).

20. What is PostgreSQL? URL: https://www.ibm.com/think/topics/postgresql (дата звернення: 05.03.2025).

21. Bakhai J., Eapen A., Rao A., Mendon N., & Padmanabha V. IoT based home security. *Journal of Student Research*. 2020. P. 45-50.

22. Sruthy S., Yamuna S., & George S. N. An IoT based Active Building Surveillance System using Raspberry Pi and NodeMCU, 2020. 437 p.

23. Alsaffar H. G., & Erçelebi E. Design and Implementation of the IoT Surveillance System using Electronic Appliances with Raspberry Pi. *Al-Iraqia Journal for Scientific Engineering Research*. 2024. P. 71-79.

24. Sendhilkumar N. C., Malarvizhi C., Anand M., & Periyarselvam K. (2021). Internet of Things Based Indoor Smart Surveillance and Monitoring System using Arduino and Raspberry Pi. *Journal of Physics: Conference Series*. 1964(6). P. 21–40.

25. Jha J., Dubey P. R., Pradhan P., & Pai S. N. IoT-Based Home Security System with Wireless Communication. In A. Hassanien, R. Bhatnagar, & A. Darwish. *Advanced Machine Learning Technologies and Applications*. 2021. P. 489–498.

26. Sarimole F. M., & Septianto A. E. Implementation of IoT-Based Facial Recognition for Home Security System Using Raspberry Pi and Mobile Application. *International Journal Software Engineering and Computer Science*. 2024. P. 453–462.

27. Ray A. K., & Bagwari A. IoT based Smart home: Security Aspects and security architecture. *In 2020 IEEE 9th International Conference on Communication Systems and Network Technologies (CSNT)*. 2020. P. 218–222. IEEE.

28. Gou Q., Yan L., Liu Y., & Li Y. Construction and strategies in IoT security system. *In 2013 IEEE international conference on green computing and communications and IEEE internet of things and IEEE cyber, physical and social computing*. (2013, August). P. 1129-1132. IEEE.

29. Sisavath C., & Yu L. Design and implementation of security system for smart home based on IOT technology. *Procedia Computer Science*. 2021. P. 4-13.

30. Xu T., Wendt J. B., & Potkonjak M. Security of IoT systems: Design challenges and opportunities. *In 2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. (2014, November). P. 417-423. IEEE.

31. Perwej Y., Parwej F., Hassan M. M., & Akhtar N. The internet-of-things (IoT) security: A technological perspective and review. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology (IJSRCSEIT)*. 2019. P. 462-482.

32. Çavaş M., & Ahmad M. B. A review advancement of security alarm system using internet of things (IoT). *International Journal of New Computer Architectures and their Applications (IJNCAA)*. 2019. P. 38-49.

33. Mahendra S., Sathiyanarayanan M., & Vasu R. B. Smart security system for businesses using internet of things (iot). *In 2018 Second International Conference on Green Computing and Internet of Things (ICGCIoT)*. (2018, August). P. 38-49. IEEE.

34. Sinha S., Teli E. H., & Tasnin W. Remote Monitoring and Home Security System. *In 2021 Innovations in Power and Advanced Computing Technologies (i-PACT). (*2021, November). P. 1-8. IEEE.

35. Kodali R. K., Rajanarayanan S. C., Koganti A., & Boppana L. IoT based security system. *In TENCON 2019 IEEE Region 10 Conference (TENCON)*. (2019, October). P. 1253-1257. IEEE.

36. Ionescu O., Dumitru V., Pricop E., Buiu O., Cobianu C., Raneti M., & Marica, C. On the development of a robust cyber security system for Internet of Things devices. *In 2019 11th International Conference on Electronics, Computers and Artificial Intelligence (ECAI).* (2019, June). P. 1-5. IEEE.

37. Zhao K., & Ge L.. A survey on the internet of things security. *In 2013 Ninth international conference on computational intelligence and security.* (2013, December). P. 663-667. IEEE.

38. Gou Q., Yan L., Liu Y., & Li Y. Construction and strategies in IoT security system. *In 2013 IEEE international conference on green computing and communications and IEEE internet of things and IEEE cyber, physical and social computing. (*2013, August). P. 1129-1132. IEEE.

39. Dineva K., & Atanasova T. *Security in IoT systems. Int. Multidiscip. Sci. GeoConference Surv. Geol. Min. Ecol. Manag. SGEM*. 2019. P. 569-577.

40. Chan M., Campo E., Estève D., & Fourniols J. Y. *Smart homes current features and future perspectives*. 2009. P. 90-97.

41. Hwang Y. H. Iot security & privacy: threats and challenges. *In Proceedings of the 1st ACM workshop on IoT privacy, trust, and security*. (2015, April). P. 1-1.

42. Russell B., & Van Duren D. *Practical internet of things security*. *Packt Publishing Ltd.* 2016. P. 12-15.

43. Singh J., Pasquier T., Bacon J., Ko H., & Eyers D. *Twenty security considerations for cloud-supported Internet of Things. IEEE Internet of things Journal*. (2015). P. 269-284.

44. Kodali R. K., Rajanarayanan S. C., Koganti A., & Boppana L. *IoT based security system. In TENCON 2019-2019 IEEE Region 10 Conference (TENCON)*. (2019, October). P. 1253-1257. IEEE.

45. Hoque M. A., & Davidson C. Design and implementation of an IoT-based smart home security system. *International Journal of Networked and Distributed Computing*. 2019. P. 85-92.

46. Patil N., Ambatkar S., & Kakde S. IoT based smart surveillance security system using raspberry Pi. *In 2017 international conference on communication and signal processing (ICCS).* (2017, April). P. 344-348. IEEE.

47. Lulla G., Kumar A., Pole G., & Deshmukh G. IoT based smart security and surveillance system. *In 2021 international conference on emerging smart computing and informatics (ESCI).* (2021, March). P. 385-390. IEEE.

48. Devaiah P. A., Jeevan H. C., Rohith N. S., Harish C., & Tiwari S. *IoT based smart surveillance security system using Raspberry Pi. TEST Engineering & Management*. 2020. P. 12345-12350.

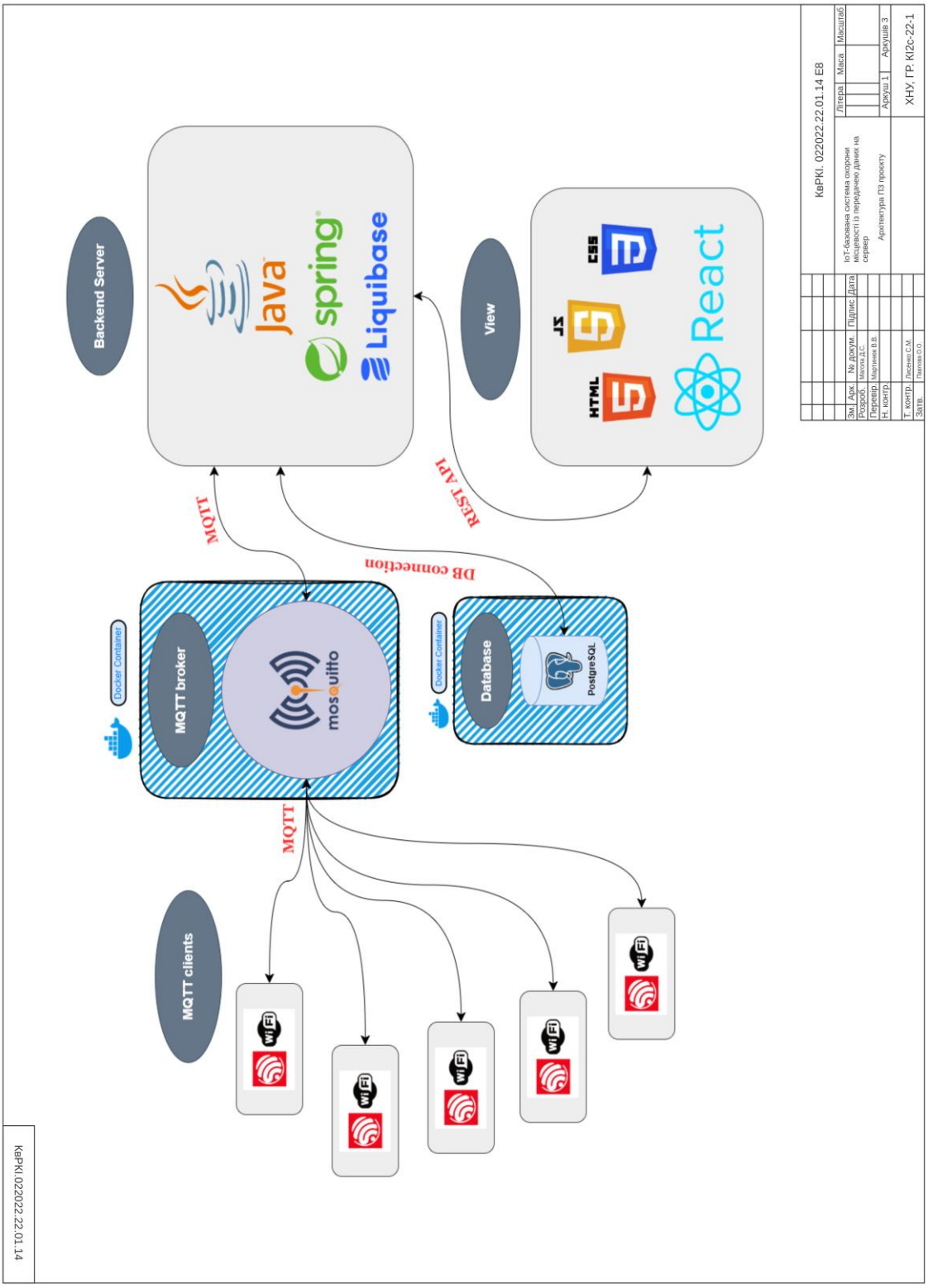
49. Patil R., Srinivas R., Yelubenchi R., & Vinay N. R. IoT enabled video surveillance system using Raspberry Pi. In *2017 2nd International Conference on Computational Systems and Information Technology for Sustainable Solution (CSITSS)*. (2017, December). P. 1-6. IEEE

50. Margapuri V., Penumajji N., & Neilsen M. PiBase: An IoT-based security system using Raspberry Pi and Google Firebase, 2021.

Додаток А

(обов'язковий)

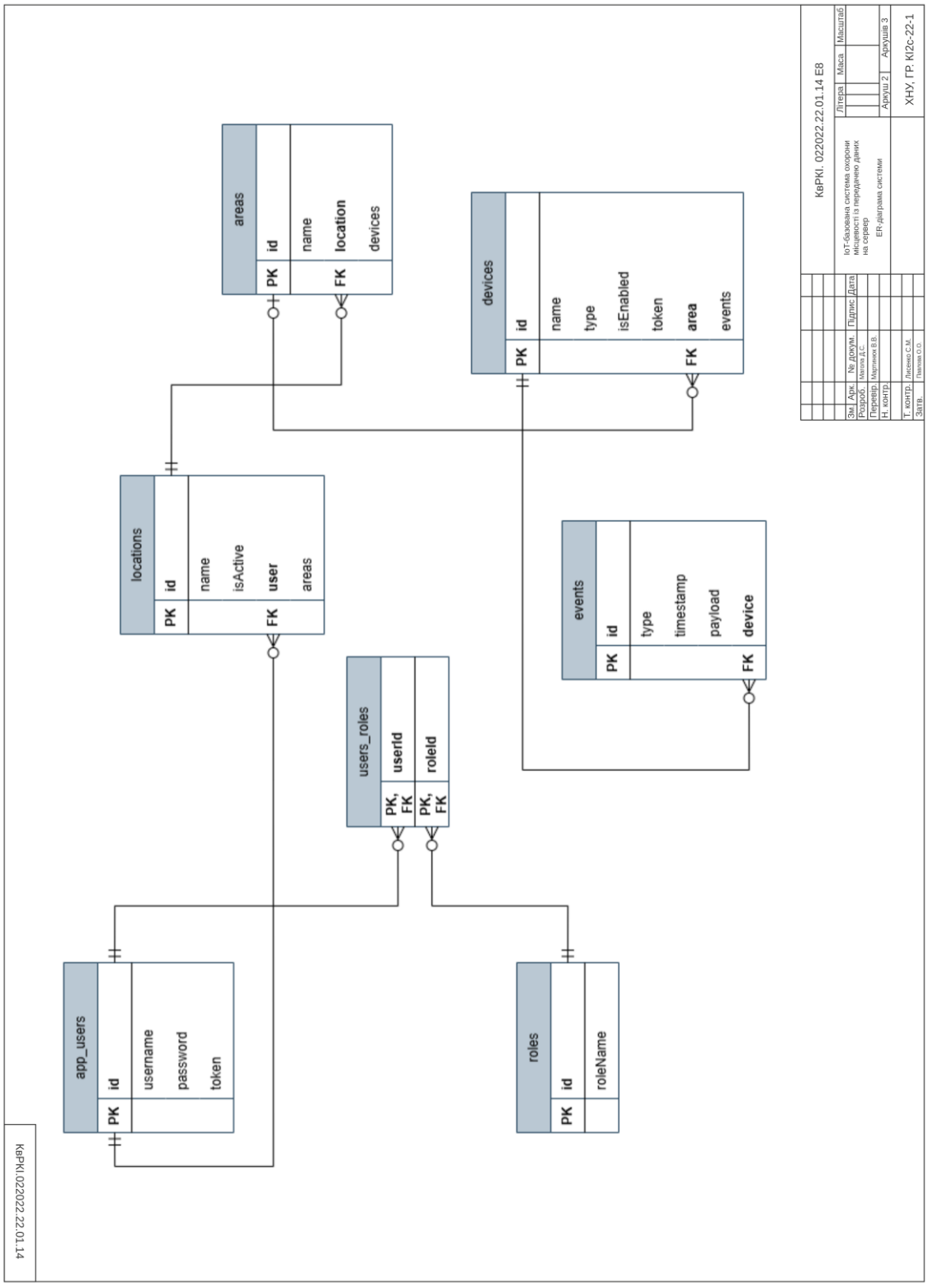
КОПІЯ КРЕСЛЕННЯ «АРХІТЕКТУРА ПЗ ПРОЄКТУ»



Додаток Б

(обов’язковий)

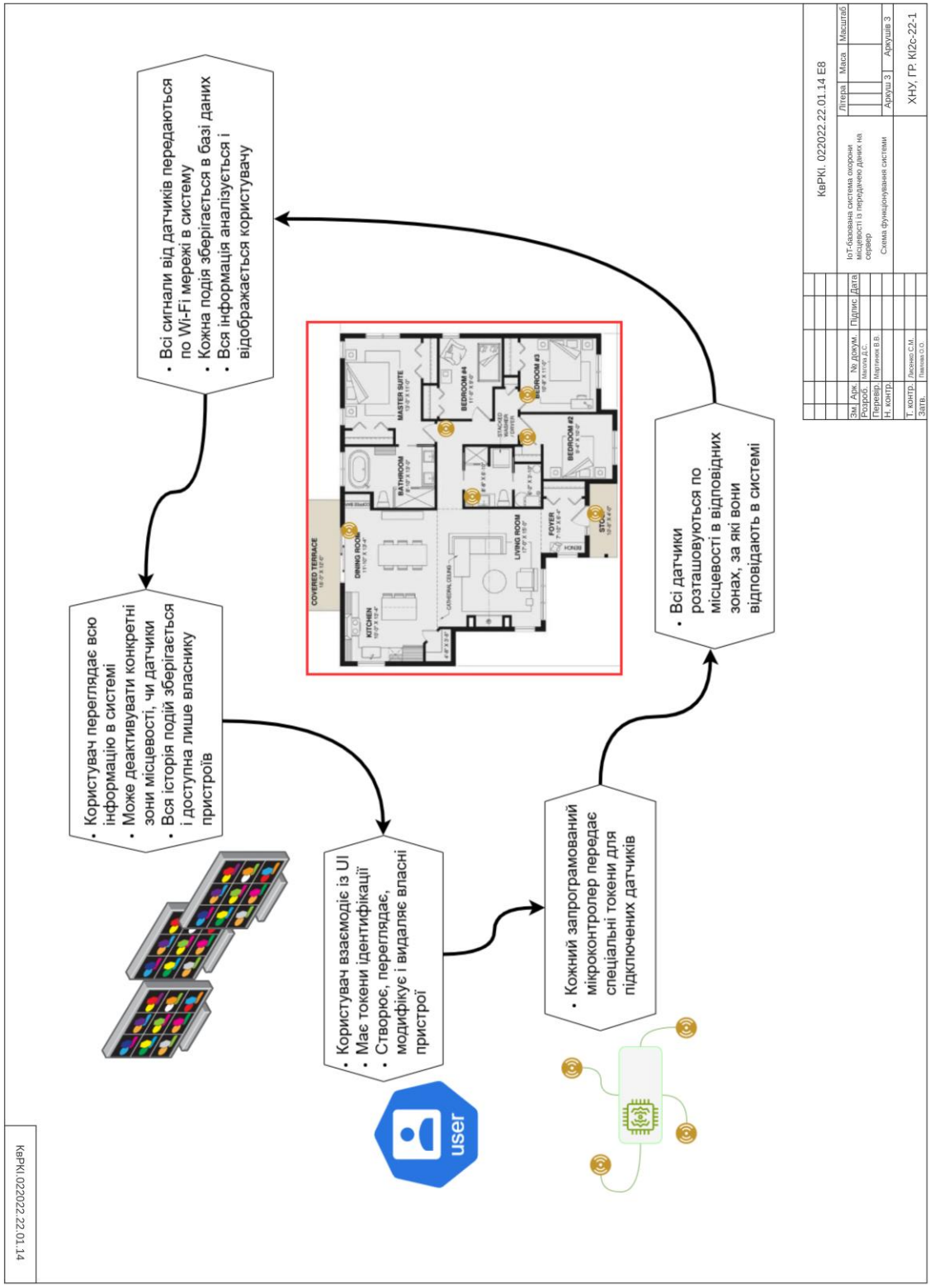
КОПІЯ КРЕСЛЕННЯ «ER-ДІАГРАМА СИСТЕМИ»



Додаток В

(обов'язковий)

КОПІЯ КРЕСЛЕННЯ «СХЕМА ФУНКЦІОНУВАННЯ СИСТЕМИ»



Додаток Г

Код програми

ПРОГРАМУВАННЯ МІКРОКОНТРОЛЕРА ESP32

```
#include <Arduino.h>
#include <PubSubClient.h>
#include <WiFi.h>

const char* network_ssid = "TP-Link_XXXX";
const char* network_password = "XXXXXXXXX";
const char* mqtt_server = "192.168.0.XXX";
const char* mqtt_username = "user";
const char* mqtt_password = "qweqwe";
const int mqtt_port = 1883;

const char* user_token = "user-123";
const char* device_token = "device123";
String mqtt_topic = "devices/" + String(user_token) + "/events";
String clientId = "Esp32Client-" + String(device_token);

WiFiClient espClient;
PubSubClient client(espClient);
const int pirPin = 12;
bool motionDetected = false;

void setup_wifi() {
  delay(100);
  WiFi.begin(network_ssid, network_password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("\nWiFi connected");
```

```
}

void reconnect() {
  while (!client.connected()) {
    Serial.println("Attempting connect to MQTT broker");

    if (client.connect(clientId.c_str(), mqtt_username,
mqtt_password)) {
      Serial.println("MQTT connected");
    } else {
      Serial.print("MQTT connection failed, reason=");
      Serial.print(client.state());
      delay(2000);
    }
  }
}

void setup() {
  pinMode(pirPin, INPUT);
  Serial.begin(115200);
  setup_wifi();
  client.setServer(mqtt_server, mqtt_port);
}

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

  int pirState = digitalRead(pirPin);
  if (pirState == HIGH && !motionDetected) {
    motionDetected = true;

    String payload = "{\"device_token\":\"";
```

```
    payload += device_token;
    payload += "\",\"event\":motion_detected}";

    client.publish(mqtt_topic.c_str(), payload.c_str());
    Serial.println("Motion detected and published to MQTT");

    delay(6000);
  } else if (pirState == LOW && motionDetected) {
    motionDetected = false;
    Serial.println("Motion stopped");
  }

  delay(100);
}
```

Налаштування MQTT протоколу на сервері

```
package com.example.backend.config;

import com.example.backend.model.Device;
import com.example.backend.model.DeviceEvent;
import com.example.backend.repository.DeviceEventRepository;
import com.example.backend.repository.DeviceRepository;
import java.time.Instant;
import java.time.LocalDateTime;
import java.time.ZoneId;
import lombok.RequiredArgsConstructor;
import org.eclipse.paho.client.mqttv3.MqttConnectOptions;
import org.json.JSONObject;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.integration.annotation.ServiceActivator;
import org.springframework.integration.channel.DirectChannel;
```

```
import
org.springframework.integration.mqtt.core.DefaultMqttPahoClientFacto
ry;
import
org.springframework.integration.mqtt.core.MqttPahoClientFactory;
import
org.springframework.integration.mqtt.inbound.MqttPahoMessageDrivenCh
annelAdapter;
import
org.springframework.integration.mqtt.outbound.MqttPahoMessageHandler
;
import org.springframework.messaging.MessageChannel;
import org.springframework.messaging.MessageHandler;
import org.springframework.messaging.MessageHeaders;

@Configuration
@RequiredArgsConstructor
public class MqttConfig {
@Value("${mqtt.broker-url}")
private String brokerUrl;

@Value("${mqtt.client-id}")
private String clientId;
@Value("${mqtt.username}")
private String mqttServerUsername;
@Value("${mqtt.password}")
private String mqttServerPassword;
@Value("${mqtt.topic}")
private String topic;

@Autowired
private DeviceRepository deviceRepository;

@Autowired
private DeviceEventRepository deviceEventRepository;
```

```
    @Bean
    public MqttConnectOptions mqttConnectOptions() {
        MqttConnectOptions options = new MqttConnectOptions();
        options.setServerURIs(new String[]{brokerUrl});
        options.setUserName(mqttServerUsername);
        options.setPassword(mqttServerPassword.toCharArray());
        options.setCleanSession(true);
        return options;
    }

    @Bean
    public MqttPahoClientFactory clientFactory() {
        DefaultMqttPahoClientFactory factory = new
DefaultMqttPahoClientFactory();
        factory.setConnectionOptions(mqttConnectOptions());
        return factory;
    }

    @Bean
    public MessageChannel mqttInputChannel() {
        return new DirectChannel();
    }

    @Bean
    public MqttPahoMessageDrivenChannelAdapter adapter() {
        MqttPahoMessageDrivenChannelAdapter adapter =
                new MqttPahoMessageDrivenChannelAdapter(clientId +
"_IN", clientFactory(), topic);
        adapter.setOutputChannel(mqttInputChannel());
        return adapter;
    }

    @Bean
    @ServiceActivator(inputChannel = "mqttInputChannel")
```

```
    public MessageHandler messageHandler() {
        return message -> {
            try {
                String rawPayload = message.getPayload().toString();
                MessageHeaders headers = message.getHeaders();
                System.out.println("Received MQTT message: " +
rawPayload + headers);

                JSONObject jsonPayload = new JSONObject(rawPayload);
                String deviceToken =
jsonPayload.getString("device_token");
                String eventType = jsonPayload.getString("event");

                Device device =
deviceRepository.findByToken(deviceToken)
                        .orElseThrow(() -> new
RuntimeException("Device not found for token: " + deviceToken));

                DeviceEvent.EventType type =
mapToEventType(eventType);
                LocalDateTime timestamp =
Instant.ofEpochMilli(headers.getTimestamp())
                        .atZone(ZoneId.systemDefault())
                        .toLocalDateTime();
                DeviceEvent deviceEvent = DeviceEvent.builder()
                        .type(type)
                        .timestamp(timestamp)
                        .payload(rawPayload)
                        .device(device)
                        .build();

                deviceEventRepository.save(deviceEvent);
            } catch (Exception e) {
                throw new RuntimeException("Something went wrong!",
e);
```

```
            }
        };
    }

    private DeviceEvent.EventType mapToEventType(String str) {
        return switch (str.toLowerCase()) {
            case "motion_detected" -> DeviceEvent.EventType.MOTION;
            default -> throw new IllegalArgumentException("Unknown
event type: " + str);
        };
    }

    @Bean
    public MessageChannel mqttOutboundChannel() {
        return new DirectChannel();
    }

    @Bean
    @ServiceActivator(inputChannel = "mqttOutboundChannel")
    public MessageHandler mqttOutbound() {
        MqttPahoMessageHandler messageHandler =
                new MqttPahoMessageHandler(clientId + "_OUT",
clientFactory());
        messageHandler.setAsync(true);
        messageHandler.setDefaultTopic(topic.replace("+", "user-
den"));
        return messageHandler;
    }
```

Anti-Plagiarism (UA) v-15.281 Educational

The maximum coincidence with one document 1.0%

Dictionaries check: en_US, ru_RU, ua_UA. **Errors in the documents: 10%**

	Document		Sum coincidence on the DB	
ID: 244091 Title: БКР IoT-базована система охорони місцевості із передачею даних на сервер Added in a DB: 2025-06-08 Authors: Денис МАГОЛІА Heads: Валерій МАРТИНЮК Consultants: Opponents:	Symbols	Lexemes	Symbols	Lexemes
	104631	756	1250 (1%)	14 (2%)

Plagiarism sources

ID	Description	Plagiarism presence in the document	
		Symbols	Lexemes

РІШЕННЯ ЕКСПЕРТНОЇ КОМІСІЇ

КАФЕДРИ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: IoT-базована система охорони місцевості із передачею даних на сервер

Автор: Денис МАГОЛА

Спеціальність: 123– Комп'ютерна інженерія

Освітня програма: освітньо-професійна

Науковий керівник: Валерій МАРТИНЮК, д.т.н, професор

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) в проєкті застосовуються загальноприйняті формати представлення даних, що використовуються у відкритих технічних стандартах;
2) усі запозичення фрагментарні, або мають належним чином оформленні посилання;
3) окремі виявлені збіги є загальновживаними фразами або виразами, про що свідчить посилання системи на збіг з 5-10 джерелами на один фрагмент речення;
4) в деяких частинах системи використовуються чотирьохрозрядні двійкові послідовності, які є типовими вхідними даними для багатьох задач і не підлягають захисту авторським правом, тому їх використання не може вважатися порушенням;
5) всі зафіксовані системою ознаки модифікації тексту відносяться до комбінування латинських символів зі україномовними скороченнями індексів в формулах, що не є модифікацією тексту.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості StrikePlagiarism, складає 5.89% і адресується до 36 першоджерела; та системою Anti-Plagiarism складає 1%, що, з урахуванням наведених обґрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

Керівник роботи _____ Валерій МАРТИНЮК

Гарант ОП _____ Андрій НІЧЕПОРУК

Завідувач кафедри КІІС _____ Ольга ПАВЛОВА

Завідувачу кафедри КІІС
д-р. філософії, доц. Ользі ПАВЛОВІЙ

Дениса МАГОЛИ
ПІБ здобувача вищої освіти

ФІТ, 3 курсу, групи КІ2с-22-1

ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності у Хмельницькому національному університеті» від 01.07.2022, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений (а). Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений(а) та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Strike-Plagiarism та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

9.06 2025 року

Протокол аналізу звіту подібності експертом

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

Автор: Денис МАГОЛА

Співавтор:

Назва: МАГОЛА_IoT-базована система охорони місцевості із передачею даних на сервер

Експерт:

Підрозділ: Кафедра комп'ютерної інженерії та інформаційних систем

Коефіцієнт подібності 1:5.9%

Коефіцієнт подібності 2:1.9%

Мікропробіли: 39

Заміна букв: 0

Інтервали: 0

Білі знаки: 1

Дата створення звіту: 2025-06-08 07:18:42.0

Після аналізу Звіту подібності констатую наступне:

☒ **Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається.**

☐ **Запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Таким чином робота повертається на доопрацювання.**

☐ **Виявлено запозичення і плагіат або навмисні текстові спотворення (маніпуляції), як передбачувані спроби укриття плагіату, які роблять роботу невідповідною вимогам законодавства (Ст. 32. ЗУ Про вищу освіту, пункт 3.1, Ст. 42. ЗУ Про освіту) та вимог НАЗЯВО (Критерій 5), а також кодексу етики і процедурам. Таким чином робота не приймається.**

Обгрунтування:

2025-06-08 Доцент Андрій Нічепорук

Дата експерт

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Дипломник: Магола Денис Сергійович

Тема: IoT-базована система охорони місцевості із передачею даних на сервер

Спеціальність: 123 «Комп'ютерна інженерія»

Обсяг кваліфікаційної роботи:

Кількість листів креслень ___3____ Кількість сторінок записки ___59___

1. Короткий зміст роботи та прийнятих рішень: Метою кваліфікаційної роботи є визначення умов та особливостей застосування програмно-апаратного комплексу для віддаленого моніторингу території з використанням IoT-технологій та передачею їх даних на сервер.

2. Висновок про відповідність роботи дипломному завданню: Робота повністю відповідає поставленому завданню.

3. Характеристика виконання кожного розділу, ступінь використання останніх досягнень науки, техніки і передових методів роботи: У першому розділі кваліфікаційної роботи подано огляд сучасних рішень у сфері IoT-базованих систем безпеки, проаналізовано технології, підходи до реалізації систем охорони, способи передачі даних та інтеграції апаратної і програмної частин. На основі аналізу сформовано напрям для побудови власної системи з урахуванням актуальних технічних рішень. Другий розділ присвячено проєктуванню структури системи та її основних модулів. Визначено способи взаємодії між компонентами, підібрано ефективні засоби зв'язку, що дозволяють забезпечити стабільну роботу в умовах можливих мережевих обмежень. MQTT обрано як простий і надійний протокол обміну повідомленнями. Детально розглянуто технічні компоненти системи, взаємодію сенсорів із мікроконтролерами, типи датчиків та принципи їх роботи. У третьому розділі описано процес реалізації системи згідно з розробленою архітектурою. Показано, як виконано налаштування компонентів, створено базу даних, реалізовано логіку обробки інформації, і забезпечено надійну взаємодію між елементами. В роботі було використано сучасні програмні інструменти та бібліотеки, і

вона демонструє повноцінну IoT-систему охорони з можливістю розширення функціоналу та застосування в реальних умовах.

4. Позитивні сторони роботи: Робота виконана із використанням сучасних технологій і демонструє практичну цінність IoT-рішень у сфері безпеки.

5. Негативні сторони роботи: Не розглянуто більш детально аспекти захисту даних і кібербезпеки системи.

6. Оцінка графічного оформлення та пояснювальної записки роботи: Пояснювальна записка оформлена коректно, згідно діючих стандартів оформлення документації.

7. Відгук про роботу в цілому: Робота виконана на належному науково-технічному рівні.

8. Інші зауваження: _____

9. Оцінка дипломної роботи: добре

Рецензент (прізвище, ім'я, по батькові, посада, місце роботи) _____

Корнієнко Юрій Вікторович

доцент кафедри ПЗ

"9" червня 2025 р. _____ (підпис)