Міністерство освіти і науки України
Харківський національний університет імені В. Н. Каразіна
Навчально-науковий інститут комп’ютерних наук та штучного інтелекту
Кафедра комп'ютерних систем та робототехніки

«Затверджую»
В.о. завідувача кафедри комп’ютерних
систем та робототехніки
_____к. ф.-м. н., доц. ХРУСЛОВ М. М.
«__» червня 2025року

Пояснювальна записка
до кваліфікаційної роботи
бакалавра

на тему: **Створення універсальної платформи безпечного простору**

Спеціальність:123 – Комп’ютерна інженерія.
Галузь знань: 12 – Інформаційні технології
Освітня програма «Комп’ютерна інженерія»

Захищено на засіданні
Екзаменаційної комісії № 44
Протокол № _ від __.06.2025 р.
Оцінка _____ / _____
Голова Атестаційної комісії
_____Чугай А. М.

Виконав:
студент групи КІ–41
МАХНИБОРОДА Денис Володимирович

Керівник:
Старший викладач кафедри комп'ютерних систем та робототехніки
РАЛО Олександр Миколайович

Рецензент:
Кандидат фізико-математичних наук
Доцент кафедри математичного моделювання та аналізу даних
ПОКЛОНСЬКИЙ Євген Васильович

Харків – 2025

АНОТАЦІЯ

Пояснювальна записка до кваліфікаційної роботи бакалавра складається зі вступу, трьох розділів, висновків, списку використаних джерел і двох додатків. Загальний обсяг роботи складає 72 сторінки, із яких 37 сторінок основної частини з 22 рисунками, 11 найменуваннями списку використаних джерел та п’ятьма додатками.

Метою роботи є розробка універсальної платформи для безпечних просторів, яка дозволяє інтегрувати різні типи датчиків для моніторингу та у режимі реального часу.

Об'єкт дослідження – процес створення та функціонування універсальної платформи безпечних просторів, включаючи процес інтеграції різнорідних пристроїв безпеки, обробки даних від датчиків у реальному часі, виявлення та класифікації інцидентів.

Предмет дослідження – методи, моделі, технології відстеження та обробки інцидентів у реальному часі, інтеграція системи спостереження з веб-платформою, та архітектура програмного забезпечення для безпечних просторів.

Проблема, яку вирішує кваліфікаційна робота: Існуючі рішення часто обмежені специфічними типами просторів або мають складну інтеграцію з різними типами датчиків. Робота спрямована на створення гнучкої системи, яка дозволяє користувачам налаштовувати моніторинг безпеки для різноманітних просторів та ефективно отримувати сповіщення про інциденти.

Область застосування – Системи безпеки для житлових, комерційних та промислових приміщень. Розроблена платформа може бути використана для моніторингу будинків, квартир, офісів, складів, виробничих приміщень та інших об'єктів, де потрібен постійний контроль безпеки.

***Ключові слова*:** Безпечний простір, моніторинг безпеки, виявлення інцидентів, система відеоспостереження, датчики безпеки, Zigbee, IoT, веб-платформа.

ABSTRACT

The explanatory note to the bachelor's qualification work consists of an introduction, three chapters, conclusions, a list of sources used and two appendices. The total volume of the work is 72 pages, of which 37 pages are the main part with 22 figures, 11 names of the list of sources used and five appendices.

The purpose of the work is to develop a universal platform for safe spaces, which allows integrating various types of sensors for monitoring and in real time.

The object of the study is the process of creating and operating a universal platform for safe spaces, including the process of integrating heterogeneous security devices, processing data from sensors in real time, detecting and classifying incidents.

The subject of the study is methods, models, technologies for tracking and processing incidents in real time, integrating the surveillance system with a web platform, and software architecture for safe spaces.

Problem solved by the qualification work: Existing solutions are often limited to specific types of spaces or have complex integration with different types of sensors. The work aims to create a flexible system that allows users to configure security monitoring for a variety of spaces and effectively receive notifications about incidents.

Scope – Security systems for residential, commercial and industrial premises. The developed platform can be used to monitor houses, apartments, offices, warehouses, production facilities and other facilities where constant security control is required.

Keywords: Safe space, security monitoring, incident detection, video surveillance system, security sensors, Zigbee, IoT, web platform.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ І УМОВНИХ ПОЗНАЧЕНЬ

Python – високорівнева мова програмування загального призначення, що використовується для розробки програмного забезпечення, аналізу даних, автоматизації задач та створення веб-додатків.

API – (Application Programming Interface) інтерфейс програмування додатків. Набір функцій та протоколів для створення програмного забезпечення, що дозволяє різним програмам взаємодіяти між собою.

HTTP – (Hypertext Transfer Protocol) протокол передачі гіпертексту. Використовується для передачі даних у всесвітній павутині (World Wide Web) та є основою для обміну інформацією між веб-серверами та клієнтами.

React – JavaScript-бібліотека для побудови користувацьких інтерфейсів.

Redux – бібліотека для управління станом додатку в JavaScript-застосунках.

REST – (Representational State Transfer) архітектурний стиль для розподілених гіпермедіа систем.

RS-485 – стандарт послідовного інтерфейсу для передачі даних.

SPA – (Single Page Application) односторінковий веб-додаток.

Zigbee – відкритий стандарт бездротового зв'язку для створення персональних мереж з низьким енергоспоживанням.

JWT – (JSON Web Token) відкритий стандарт для безпечної передачі інформації між сторонами у вигляді JSON-об'єкта.

MQTT – (Message Queuing Telemetry Transport) легкий протокол обміну повідомленнями, оптимізований для IoT-пристроїв.

ВСТУП

Актуальність роботи. У сучасному світі питання безпеки приватних, комерційних та промислових просторів набуває все більшого значення. Зростання кількості загроз різного характеру – від пожеж, затоплень та інших аварійних ситуацій до несанкціонованого доступу та крадіжок – вимагає розробки ефективних та універсальних рішень для їх моніторингу та попередження.

Розвиток технологій відеоспостереження, сенсорів та датчиків створює можливості для впровадження комплексних систем моніторингу, здатних забезпечити безпеку різноманітних просторів. Однак існує потреба в універсальній платформі, яка дозволить користувачам легко налаштовувати та адаптувати ці технології для своїх конкретних потреб, незалежно від типу приміщення чи характеру потенційних загроз.

Особливої актуальності набуває автоматизація процесів виявлення інцидентів та оперативного реагування на них. Своєчасне виявлення потенційно небезпечних ситуацій може запобігти серйозним наслідкам, включаючи матеріальні збитки та загрози життю і здоров'ю людей. При цьому важливим є не лише виявлення інцидентів, але й ефективна передача цієї інформації користувачам у реальному часі, що дозволить вжити необхідних заходів якомога швидше.

Інтеграція сучасних технологій веб-розробки, баз даних та систем моніторингу відкриває нові можливості для створення універсальної платформ безпеки, які можуть бути адаптовані для різних типів просторів та різноманітних потреб користувачів.

Метою роботи є розробка універсальної платформи для безпечних просторів, яка дозволяє інтегрувати різні типи датчиків для моніторингу та виявлення інцидентів у режимі реального часу. Робота спрямована на створення гнучкої системи, що забезпечує ефективне налаштування,

підтримку різних датчиків для виявлення небезпечних ситуацій в різноманітних просторах – від житлових приміщень до промислових об'єктів.

Об'єкт дослідження – це процес створення універсальної платформи безпечних просторів, процес інтеграції різнорідних пристроїв безпеки, обробки даних від датчиків у реальному часі, виявлення та класифікації інцидентів, а також організації взаємодії між компонентами системи.

Предмет дослідження – методи та технології відстеження та обробки інцидентів у реальному часі, інтеграція Zigbee з веб-платформою, та архітектура програмного забезпечення для безпечних просторів.

У процесі дослідження розв'язувалися наступні завдання:

1. Аналіз існуючих рішень для моніторингу безпеки просторів та виявлення їх обмежень.
2. Розробка архітектури універсальної платформи для безпечних просторів на основі сучасних технологій.
3. Створення методів та алгоритмів для інтеграції різних типів датчиків у єдину систему моніторингу.
4. Розробка алгоритмів виявлення інцидентів та аномалій на основі даних з датчиків.
5. Реалізація механізмів для оперативної передачі інформації про інциденти на веб-платформу.
6. Тестування та оцінка ефективності розробленої платформи в різних умовах та сценаріях використання.
7. Дослідження можливостей розширення функціональності платформи та інтеграції з іншими протоколами.

РОЗДІЛ 1.
ОГЛЯД І АНАЛІЗ ІСНУЮЧИХ СИСТЕМ ФІЗИЧНОЇ БЕЗПЕКИ ТА МОНІТОРИНГУ В ПРИМІЩЕННЯХ

1.1 Актуальність розробки універсальних систем безпеки

У сучасному світі питання безпеки приміщень та просторів різного призначення набуває особливої актуальності. Зростаюча кількість загроз – від побутових аварій, пожеж і затоплень до несанкціонованого доступу та крадіжок – вимагає комплексних та ефективних рішень. Традиційно системи безпеки розвивались у відокремлених напрямках, як-от відеоспостереження, охоронна сигналізація, контроль доступу, пожежна безпека. Однак такий підхід призводить до фрагментованості, складнощів з інтеграцією різних підсистем, підвищення витрат на обслуговування та експлуатацію.

Дослідження Ринку Глобальної Безпеки, проведене компанією IMARC, показує, що світовий ринок систем безпеки демонструє стабільне зростання і до 2033 року досягне 34,8 мільярдів доларів [1]. Зростання ринку обумовлене збільшенням використання ШІ в системах безпеки.

Розвиток технологій IoT, бездротових комунікацій та хмарних обчислень відкриває нові можливості для створення по-справжньому інтегрованих систем безпеки. Універсальна платформа безпечних просторів дозволяє об'єднати різні типи датчиків, пристроїв та систем в єдину екосистему, якою можна централізовано управляти з будь-якої точки світу через Інтернет. Це не лише підвищує зручність користування, але й значно розширює функціональність, покращує реагування на інциденти та зменшує загальну вартість володіння системою.

Створення універсальної платформи для безпечних просторів має ряд незаперечних переваг. По-перше, вона забезпечує інтеграцію різнорідних пристроїв у єдину систему, дозволяючи їм взаємодіяти та обмінюватись даними. По-друге, така платформа може адаптуватися до різних типів просторів – від приватних помешкань до комерційних об'єктів та виробничих

приміщень. По-третє, вона надає користувачу можливість контролювати систему безпеки віддалено, отримувати сповіщення про інциденти та реагувати на них у реальному часі.

1.2 Огляд існуючих систем фізичної безпеки

Сучасний ринок систем безпеки представлений численними рішеннями, які можна класифікувати як за призначенням, так і за технологічними підходами. Розглянемо основні категорії систем фізичної безпеки та проаналізуємо їх основні характеристики.

1.2.1 Системи охоронної сигналізації

Системи охоронної сигналізації є найбільш поширеним типом систем безпеки, призначеним для виявлення несанкціонованого доступу та захисту від вторгнень. Сучасні системи охоронної сигналізації складаються з центральної панелі управління, різноманітних датчиків (руху, відкриття, розбиття скла), виконавчих пристроїв (сирен) та засобів комунікації.

На ринку представлені як традиційні дротові системи, так і бездротові рішення. Дротові системи відрізняються надійністю та стабільністю роботи, проте вимагають прокладання кабелів та більш складного монтажу. Бездротові системи значно простіші в установці, але потребують регулярної заміни батарей у компонентах та можуть бути вразливими до радіоперешкод.

Як зазначають різні дослідники, бездротові системи мають специфічні вразливості, пов'язані з можливістю глушіння сигналу та вичерпанням заряду батареї [2].

Одним із лідерів серед бездротових систем охоронної сигналізації є Ajax Systems, що пропонує комплексне рішення на базі власного радіопротоколу Jeweller. Система відрізняється простотою налаштування, надійністю зв'язку (дальність до 2000 метрів на відкритій місцевості) та тривалим терміном автономної роботи компонентів (до 7 років). Особливістю Ajax є також інтелектуальні алгоритми захисту від хибних спрацювань та миттєве сповіщення про тривоги.

Серед інших відомих виробників систем охоронної сигналізації варто відзначити такі компанії як DSC, Paradox, Hikvision, Dahua та Partizan. Кожен виробник має власні технологічні особливості та орієнтований на певний сегмент ринку.

1.2.2 Системи відеоспостереження

Системи відеоспостереження призначені для візуального контролю простору, запису та аналізу відеоінформації. Сучасні системи відеоспостереження пройшли значну еволюцію – від аналогових CCTV-камер до високотехнологічних IP-систем з розрішенням 4K та вище.

Лідерами ринку відеоспостереження є китайські компанії Hikvision та Dahua, які пропонують широкий асортимент обладнання – від бюджетних рішень до професійних систем з розвиненою відеоаналітикою. Ці виробники розробляють власні процесори обробки зображення та алгоритми штучного інтелекту, що дозволяє реалізувати такі функції як розпізнавання облич, аналіз поведінки, виявлення аномалій тощо.

Важливою тенденцією розвитку систем відеоспостереження є інтеграція з хмарними сервісами, що дозволяє зберігати та аналізувати відеодані без потреби в локальних відеореєстраторах. Такий підхід спрощує масштабування системи та надає можливість віддаленого доступу з будь-якої точки світу.

1.2.3 Системи контролю доступу

Системи контролю доступу призначені для регулювання та обліку доступу в приміщення. Вони дозволяють надавати доступ лише авторизованим особам, вести журнал подій та гнучко налаштовувати права доступу.

Сучасні системи контролю доступу використовують різні методи ідентифікації [3] – від традиційних карт і брелків до біометричних даних (відбитки пальців, розпізнавання обличчя). Інтеграція з мобільними пристроями дозволяє використовувати смартфон як ідентифікатор, що значно підвищує зручність користування системою.

За даними дослідження GreenVision, впровадження біометричних систем контролю доступу зростає на 39% щорічно, причому найбільш швидко розвивається сегмент розпізнавання обличчя [4].

Провідні виробники систем безпеки, такі як Hikvision та Dahua, пропонують комплексні рішення контролю доступу, які легко інтегруються з іншими підсистемами безпеки. Це дозволяє, наприклад, автоматично відображати відео з камери при спробі доступу або блокувати доступ у разі виникнення надзвичайної ситуації.

1.2.4 Системи пожежної безпеки

Системи пожежної безпеки є критично важливими для захисту життя людей та майна. Вони включають підсистеми виявлення пожежі, оповіщення, евакуації та, за необхідності, автоматичного пожежогасіння.

Сучасні системи пожежної безпеки використовують різні технології виявлення – від традиційних димових та теплових датчиків до сенсорних детекторів та інтелектуальних алгоритмів аналізу. Це дозволяє значно зменшити кількість хибних спрацювань при збереженні високої чутливості до реальних пожежних ситуацій.

На ринку представлені як автономні пожежні сигналізації для невеликих об'єктів, так і комплексні адресні системи для великих будівель та комплексів. Останні дозволяють точно локалізувати місце виникнення пожежі та забезпечити ефективне реагування.

1.2.5 Системи екологічного моніторингу

До систем екологічного моніторингу відносяться рішення для контролю параметрів навколишнього середовища та виявлення потенційно небезпечних ситуацій, таких як витоки газу, протікання води, екстремальні температури тощо.

Важливим компонентом таких систем є датчики протікання води, які дозволяють своєчасно виявити аварійну ситуацію та запобігти затопленню приміщення. Розвинені системи не лише виявляють протікання, але й автоматично перекривають подачу води, мінімізуючи таким чином збитки.

Датчики газу (чадного, метану, пропану) є важливими для безпеки приміщень з газовим обладнанням. При виявленні небезпечної концентрації газу система може автоматично активувати вентиляцію, перекрити подачу газу та сповістити власника.

1.3 Аналіз протоколів зв'язку для систем безпеки

Ключовим аспектом універсальної платформи безпечних просторів є вибір ефективного протоколу зв'язку між компонентами системи. Сучасні системи безпеки використовують різні протоколи, кожен з яких має свої переваги та обмеження.

1.3.1 Дротові протоколи

Традиційно системи безпеки будувалися на дротових технологіях, які забезпечують надійний зв'язок та не залежать від батарейного живлення. Найбільш поширеними дротовими інтерфейсами є:

RS-485 – послідовний інтерфейс, що дозволяє підключати до однієї лінії зв'язку багато пристроїв. Він широко використовується в системах контролю доступу, охоронній та пожежній сигналізації завдяки можливості передачі даних на відстань до 1200 метрів.

Ethernet – основний стандарт для локальних мереж, який все частіше використовується в системах безпеки, особливо для відеоспостереження та інтегрованих систем. Він забезпечує високу швидкість передачі даних, можливість живлення пристроїв через PoE (Power over Ethernet) та просту інтеграцію з IT-інфраструктурою.

Перевагою дротових протоколів включають високу надійність зв'язку, відсутність проблем з батарейним живленням та захищеність від радіоперешкод. Однак вони вимагають прокладання кабелів, що може бути проблематичним у деяких приміщеннях, особливо в уже завершених об'єктах.

1.3.2 Бездротові протоколи

Бездротові технології стають все більш популярними в системах безпеки завдяки простоті установки, гнучкості та можливості розгортання системи без

пошкодження інтер'єру. Розглянемо основні бездротові протоколи, які використовуються в системах безпеки:

Wi-Fi (IEEE 802.11) широко використовується для IP-камер та інших пристроїв, які потребують високої пропускної здатності. Однак він має високе енергоспоживання, що обмежує його використання в батарейних пристроях.

Bluetooth Low Energy (BLE) оптимізований для низького енергоспоживання, що робить його придатним для датчиків та інших пристроїв з батарейним живленням. Однак він має обмежену дальність дії (зазвичай до 10-50 метрів) та не підтримує повноцінну mesh-мережу, що обмежує його застосування в великих системах безпеки.

Z-Wave – бездротовий протокол, спеціально розроблений для домашньої автоматизації та систем безпеки. Він працює на нижчих частотах (868 МГц в Європі, 908 МГц в США), що забезпечує кращу проникну здатність через стіни. Z-Wave підтримує mesh-мережу, де кожен пристрій може ретранслювати сигнал, розширюючи загальне покриття. Недоліком є закритий характер протоколу та необхідність ліцензування.

Zigbee є відкритим протоколом, що працює в діапазоні 2.4 ГГц та спеціально розроблений для створення бездротових персональних мереж з низьким енергоспоживанням [5]. Подібно до Z-Wave, Zigbee підтримує mesh-мережу, де пристрої з постійним живленням можуть виступати як ретранслятори. Протокол забезпечує шифрування AES-128 для захисту даних та має механізми для запобігання перешкодам.

Висновки за розділом 1

Розглянувши та проаналізувавши існуючі системи фізичної безпеки і моніторингу в приміщеннях можна зробити висновок, що створення універсальної платформи для безпечних просторів є актуальним завданням, що відповідає сучасним тенденціям розвитку технологій у сфері безпеки. Інтеграція різних підсистем, таких як відеоспостереження, охоронна сигналізація, контроль доступу, пожежна безпека та екологічний моніторинг, в єдину екосистему дозволяє підвищити ефективність роботи системи в

цілому, спростити управління та знизити загальну вартість володіння системою безпеки.

Аналіз ринку систем безпеки показав, що він представлений широким спектром рішень різних виробників, які переважно спеціалізуються на окремих напрямках. Провідні компанії у сфері охоронних систем та відеоспостереження, такі як Hikvision, Dahua, Ajax Systems та Partizan, пропонують комплексні рішення, але вони часто мають обмежену сумісність з обладнанням інших виробників. Це створює суттєві перешкоди для користувачів, які прагнуть побудувати інтегровану систему безпеки з оптимальним співвідношенням ціни та якості.

Системи контролю доступу еволюціонують від простих карткових рішень до біометричних систем та інтеграції з мобільними пристроями. Це підвищує рівень безпеки та зручність використання, але також потребує уваги до питань захисту персональних даних та кібербезпеки.

Системи пожежної безпеки та екологічного моніторингу є критично важливими компонентами загальної безпеки приміщень, що дозволяють своєчасно виявляти та реагувати на загрози, такі як пожежа, витік газу або затоплення.

Детальний аналіз протоколів зв'язку для систем безпеки дозволив визначити, що для створення універсальної платформи оптимальним вибором є бездротові технології, зокрема протокол Zigbee. Цей протокол забезпечує низьке енергоспоживання, надійний зв'язок завдяки mesh-архітектурі та широку сумісність з пристроями різних виробників. На відміну від протоколів Wi-Fi та Bluetooth, Zigbee спеціально оптимізований для систем безпеки та домашньої автоматизації, що робить його ідеальним вибором для універсальної платформи безпечних просторів.

Сучасні тенденції розвитку систем безпеки свідчать про зростаюче застосування штучного інтелекту та алгоритмів машинного навчання для підвищення точності виявлення інцидентів та зниження кількості хибних спрацювань. Ці технології дозволяють системам "навчатися" на основі

накопичених даних та автоматично адаптуватися до змін середовища, що значно підвищує ефективність роботи та знижує вимоги до ручного налаштування.

Інтеграція різних типів пристроїв у єдину систему вимагає розробки гнучкої архітектури програмного забезпечення, яка дозволить об'єднати обладнання різних виробників та забезпечити єдиний інтерфейс управління. Важливими аспектами такої архітектури є масштабованість, модульність та відкритість для інтеграції нових типів пристроїв та технологій.

РОЗДІЛ 2.
ПРОЕКТУВАННЯ ПЛАТФОРМИ БЕЗПЕЧНОГО ПРОСТОРУ

2.1 Постановка завдання та визначення вимог до платформи

Розробка універсальної платформи безпечних просторів вимагає чіткого визначення функціональних та нефункціональних вимог, які забезпечать ефективне вирішення завдань безпеки у різноманітних середовищах – від вулиць до комерційних об'єктів.

2.1.1 Функціональні вимоги

Функціональні вимоги визначають основні можливості, які повинна забезпечувати платформа:

Моніторинг та виявлення подій безпеки. Платформа повинна забезпечувати безперервний збір та аналіз даних з різних типів датчиків (руху, відкриття дверей/вікон, диму, протікання води, газу тощо). Система має своєчасно виявляти потенційно небезпечні ситуації, визначати їх тип та рівень критичності.

Сповіщення та реагування. При виявленні інцидентів система повинна негайно сповіщати користувачів через різні канали зв'язку. Крім того, платформа має підтримувати автоматичне виконання попередньо налаштованих сценаріїв реагування, як-от активація сирени, перекриття подачі води, блокування/розблокування дверей.

Управління просторами. Користувачі повинні мати можливість створювати та керувати різними типами просторів (будинок, квартира, офіс, склад, тощо), додавати та налаштовувати датчики.

Підтримка різних типів датчиків. Платформа має підтримувати широкий спектр датчиків та виконавчих пристроїв, включаючи датчики руху, відкриття, розбиття скла, диму, газу, протікання води, температури/вологості, а також сирени, розумні замки.

Історія подій та аналітика. Система повинна зберігати детальну історію всіх подій, дозволяти переглядати журнали, генерувати звіти та проводити аналіз трендів, що допоможе оптимізувати безпеку простору.

Інтеграція з іншими системами. Платформа має підтримувати інтеграцію з існуючими системами безпеки та автоматизації.

2.1.2 Нефункціональні вимоги

Нефункціональні вимоги визначають якісні характеристики платформи:

Надійність та відмовостійкість. Система повинна забезпечувати безперебійну роботу навіть у випадку відмови окремих компонентів. Зокрема, хаб-посередник має продовжувати виконувати базові функції моніторингу та реагування навіть при відсутності інтернет-з'єднання.

Безпека даних. Всі дані, що передаються між компонентами системи та зберігаються на сервері, повинні бути надійно захищені. Це включає шифрування даних, надійну автентифікацію користувачів, захист від несанкціонованого доступу та протидію спробам зламу системи.

Масштабованість. Платформа має підтримувати можливість розширення – від невеликої системи для квартири до комплексного рішення для великого об'єкту. Архітектура повинна дозволяти підключення сотень пристроїв без суттєвого зниження продуктивності.

Швидкодія. Затримка між виявленням події та відправкою сповіщення не повинна перевищувати 3 секунд за нормальних умов зв'язку. Час виконання команд управління пристроями не повинен перевищувати 5 секунд.

2.2 Архітектура платформи безпечного простору

Архітектура універсальної платформи безпечних просторів розроблена з урахуванням всіх визначених вимог та базується на сучасних технологіях IoT, хмарних обчислень та бездротового зв'язку. Система побудована за багаторівневим принципом, який забезпечує гнучкість, масштабованість та надійність роботи.

2.2.1 Загальна архітектура системи

Загальна архітектура системи складається з трьох основних рівнів:

Хмарний рівень представлений серверною інфраструктурою, яка забезпечує зберігання та обробку даних, управління користувачами, аналітику подій та надання веб-інтерфейсу. Хмарна платформа виконує функції централізованого управління всіма підключеними хабами та пов'язаними з ними просторами, а також забезпечує можливість віддаленого доступу до системи для користувачів.

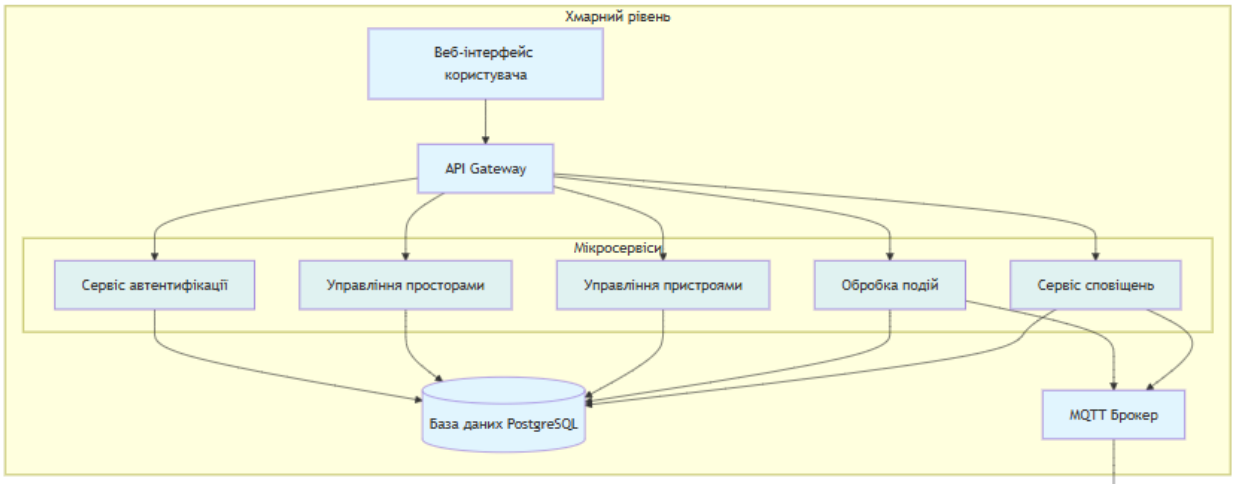

Рисунок 2.1 – Хмарний рівень.

Транспортний рівень відповідає за передачу даних між локальним та хмарним рівнями. Він забезпечує безпечний та надійний зв'язок хаба-посередника з хмарною платформою через інтернет.

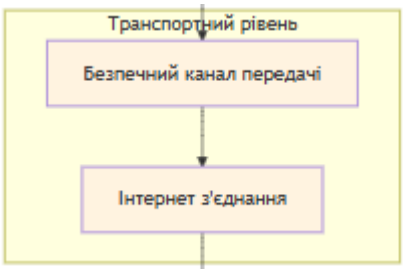

Рисунок 2.2 – Транспортний рівень.

Локальний рівень включає всі фізичні пристрої, розташовані в просторі користувача: різноманітні датчики (руху, відкриття, диму, протікання тощо) та хаб-посередник, який виступає координатором локальної

мережі пристроїв. Пристрої об'єднані в бездротову мережу на базі протоколу Zigbee, яка забезпечує надійний зв'язок та низьке енергоспоживання.

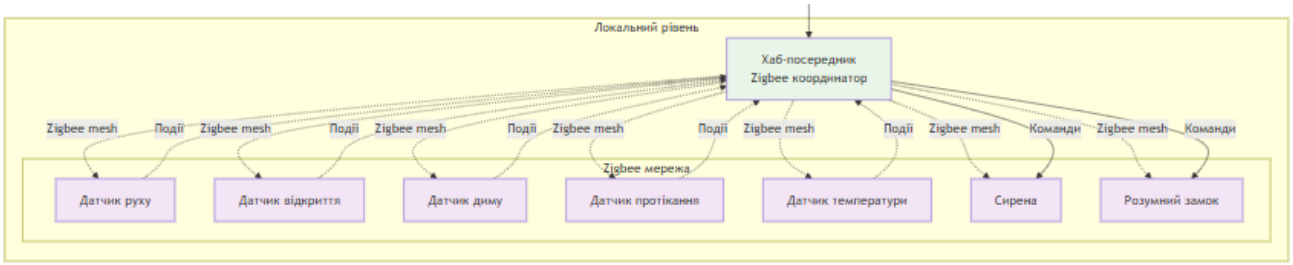

Рисунок 2.3 – Локальний рівень.

Для забезпечення модульності та гнучкості розробки, хмарна платформа побудована за мікросервісною архітектурою, де кожен сервіс відповідає за конкретний функціональний аспект системи (управління пристроями, обробка подій, сповіщення, автентифікація тощо).

2.2.2 Архітектура локального рівня

Локальний рівень є фундаментальним компонентом системи, який безпосередньо взаємодіє з фізичним простором та забезпечує первинний збір та обробку даних. Його ключовим елементом є хаб-посередник, який виконує роль центрального вузла для всієї локальної інфраструктури.

Хаб-посередник виконує кілька ключових функцій:

Локальна обробка даних: хаб обробляє дані, отримані від датчиків, виявляє події безпеки та виконує базові функції реагування навіть при відсутності зв'язку з хмарною платформою.

Зв'язок з хмарною платформою: хаб забезпечує безпечний зв'язок з хмарною платформою, передаючи дані про події та стан пристроїв, а також отримуючи команди управління.

Мережа пристроїв Zigbee організована за mesh-топологією, де кожен пристрій з постійним живленням (роутер) може ретранслювати повідомлення для інших пристроїв. Це забезпечує надійне покриття навіть у великих приміщеннях та обхід перешкод. Пристрої поділяються на три типи згідно зі специфікацією Zigbee [6]:

Координатор (Coordinator) – роль, яку виконує хаб-посередник. Існує лише один координатор у мережі, він ініціює створення мережі та керує нею.

Рисунок 2.4 – Координатор/Хаб SONOFF.

Роутери (Routers) – пристрої з постійним живленням, які можуть ретранслювати повідомлення та розширювати покриття мережі. До роутерів відносяться такі пристрої як розумні розетки, сирени, шлюзи управління освітленням тощо.

Кінцеві пристрої (End Devices) – пристрої з обмеженою функціональністю, які не ретранслюють повідомлення та можуть переходити в режим сну для економії енергії. Більшість датчиків (руху, відкриття, диму тощо) відносяться до цієї категорії.

2.2.3 Архітектура хмарного рівня

Хмарний рівень забезпечує централізоване управління, зберігання даних та надання інтерфейсу користувача. Він побудований за мікросервісною архітектурою, що забезпечує гнучкість, масштабованість та надійність.

Основні компоненти хмарного рівня:

API – єдина точка входу для всіх зовнішніх запитів до системи. Він виконує автентифікацію, авторизацію, маршрутизацію запитів до відповідних мікросервісів та базове кешування.

Сервіс автентифікації та управління користувачами відповідає за реєстрацію користувачів, автентифікацію.

Сервіс управління просторами та пристроями забезпечує створення та налаштування просторів, додавання та конфігурацію пристроїв, управління групами та режимами роботи.

Сервіс обробки подій відповідає за отримання, обробку та класифікацію подій від пристроїв, виявлення інцидентів на основі визначених правил та активацію відповідних сценаріїв реагування.

База даних зберігає інформацію про користувачів, простори, пристрої, налаштування та історію подій. Використовується реляційна база даних.

2.3 Проектування бази даних

Проектування бази даних є важливим етапом розробки універсальної платформи безпечних просторів, оскільки вона повинна ефективно зберігати та забезпечувати доступ до різнорідної інформації – від даних про користувачів та пристрої до історії подій та налаштувань.

2.3.1 Концептуальна модель даних

Концептуальна модель даних відображає основні сутності системи та зв'язки між ними:

Користувачі (Users) – люди, зареєстровані в системі, які мають доступ до одного або кількох просторів.

Простори (Spaces) – фізичні локації (будинки, квартири, офіси тощо), які контролюються системою.

Хаби (Hubs) – пристрої, які координують роботу мережі датчиків у конкретному просторі. Хаб пов'язаний з одним простором та має унікальний ідентифікатор та API-ключ для безпечного зв'язку з нашою універсальною платформою.

Пристрої (Devices) – датчики та виконавчі пристрої, підключені до хаба. Кожен пристрій має тип, унікальний ідентифікатор, стан та налаштування.

Події (Events) – повідомлення від пристроїв про зміну стану або виявлення певних умов (рух, відкриття дверей, задимлення тощо).

2.3.2 Логічна модель даних

На основі концептуальної моделі розроблена логічна модель даних, яка визначає структуру таблиць бази даних та зв'язки між ними (представлено у Додатку Г)

2.3.3 Вибір системи управління базами даних

PostgreSQL використовується для зберігання структурованих даних, таких як інформація про користувачів, простори, хаби, пристрої та сценарії. PostgreSQL забезпечує надійну транзакційність, підтримку складних запитів та JSON-даних, що дозволяє гнучко зберігати конфігурації пристроїв та сценаріїв.

2.4 Проектування серверної частини

Серверна частина платформи побудована на основі мікросервісної архітектури з використанням сучасних технологій розробки. Такий підхід забезпечує гнучкість, масштабованість та стійкість системи до відмов.

2.4.1 Вибір технологічного стеку

Для розробки серверної частини обрано наступний технологічний стек:

Мова програмування Python обрана як основна мова розробки серверної частини завдяки її простоті, виразності та багатій екосистемі бібліотек для роботи з різними технологіями. Python особливо ефективний для розробки веб-додатків, обробки даних.

FastAPI – сучасний веб-фреймворк для Python, який забезпечує високу продуктивність, автоматичну валідацію даних, автогенерацію документації та підтримку асинхронного програмування.

SQLAlchemy - бібліотека для Python, яка забезпечує гнучку роботу з реляційними базами даних. SQLAlchemy використовується для взаємодії з PostgreSQL та дозволяє працювати з базою даних на об'єктно-орієнтованому рівні.

MQTT – легкий протокол обміну повідомленнями, оптимізований для IoT-пристроїв. Використовується для комунікації між хабами та хмарною

платформою, забезпечуючи надійну та ефективну передачу даних про події та команди управління.

2.4.2 Реалізація мікросервісної архітектури

Мікросервісна архітектура передбачає розбиття серверної частини на невеликі, автономні сервіси, кожен з яких відповідає за конкретну функціональність. Це дозволяє розробляти, розгортати та масштабувати компоненти системи незалежно один від одного.

Сервіс автентифікації реалізований з використанням FastAPI та забезпечує реєстрацію користувачів, автентифікацію, управління токенами доступу та оновлення, а також базове управління користувачами.

Сервіс управління просторами та пристроями відповідає за створення та налаштування просторів, додавання та конфігурацію пристроїв, управління групами пристроїв та режимами роботи.

Сервіс обробки подій реалізує логіку обробки подій від пристроїв, виявлення інцидентів на основі визначених правил та активації відповідних сценаріїв реагування.

Рисунок 2.5 – Архітектура універсальної платформи.

2.5 Проектування інтерфейсу користувача

Інтерфейс користувача є ключовим компонентом платформи, який забезпечує взаємодію користувачів з системою. Він повинен бути інтуїтивно зрозумілим, зручним та ефективним, надаючи доступ до всіх функцій системи без перевантаження користувача зайвою інформацією.

2.5.1 Веб-інтерфейс

Веб-інтерфейс є основним засобом взаємодії користувачів з платформою та забезпечує доступ до всіх функцій системи з будь-якого пристрою з веб-браузером. Розроблений з використанням сучасних технологій фронтенд-розробки:

React – бібліотека JavaScript для побудови інтерактивних користувацьких інтерфейсів, яка забезпечує ефективне оновлення та рендеринг компонентів інтерфейсу.

Redux – бібліотека для управління станом додатку, яка забезпечує передбачуваний та централізований стан, що спрощує розробку складних інтерфейсів.

Material-UI – набір компонентів React, реалізованих відповідно до принципів Material Design, що забезпечує консистентний та сучасний вигляд інтерфейсу.

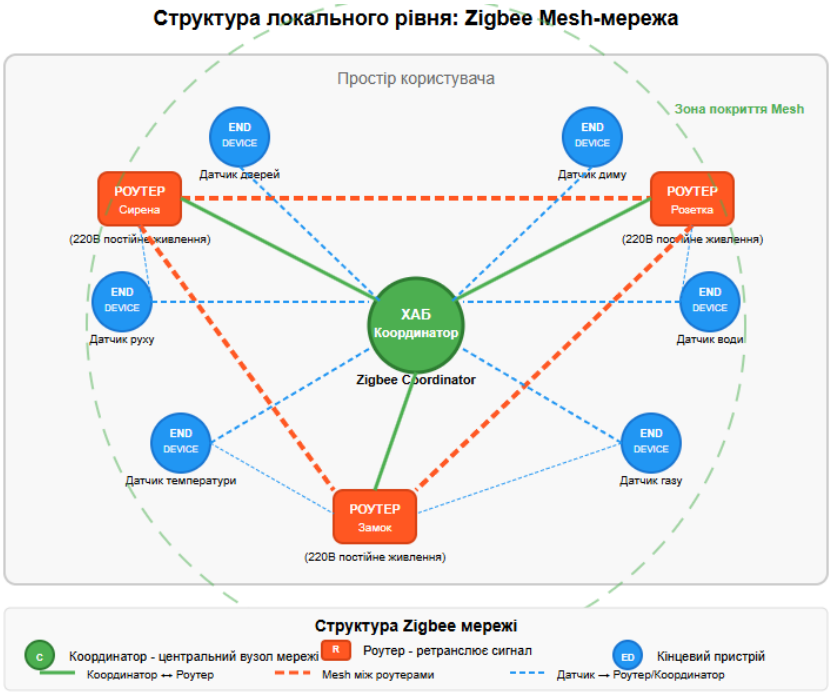

Рисунок 2.6 – Архітектура локального рівня.

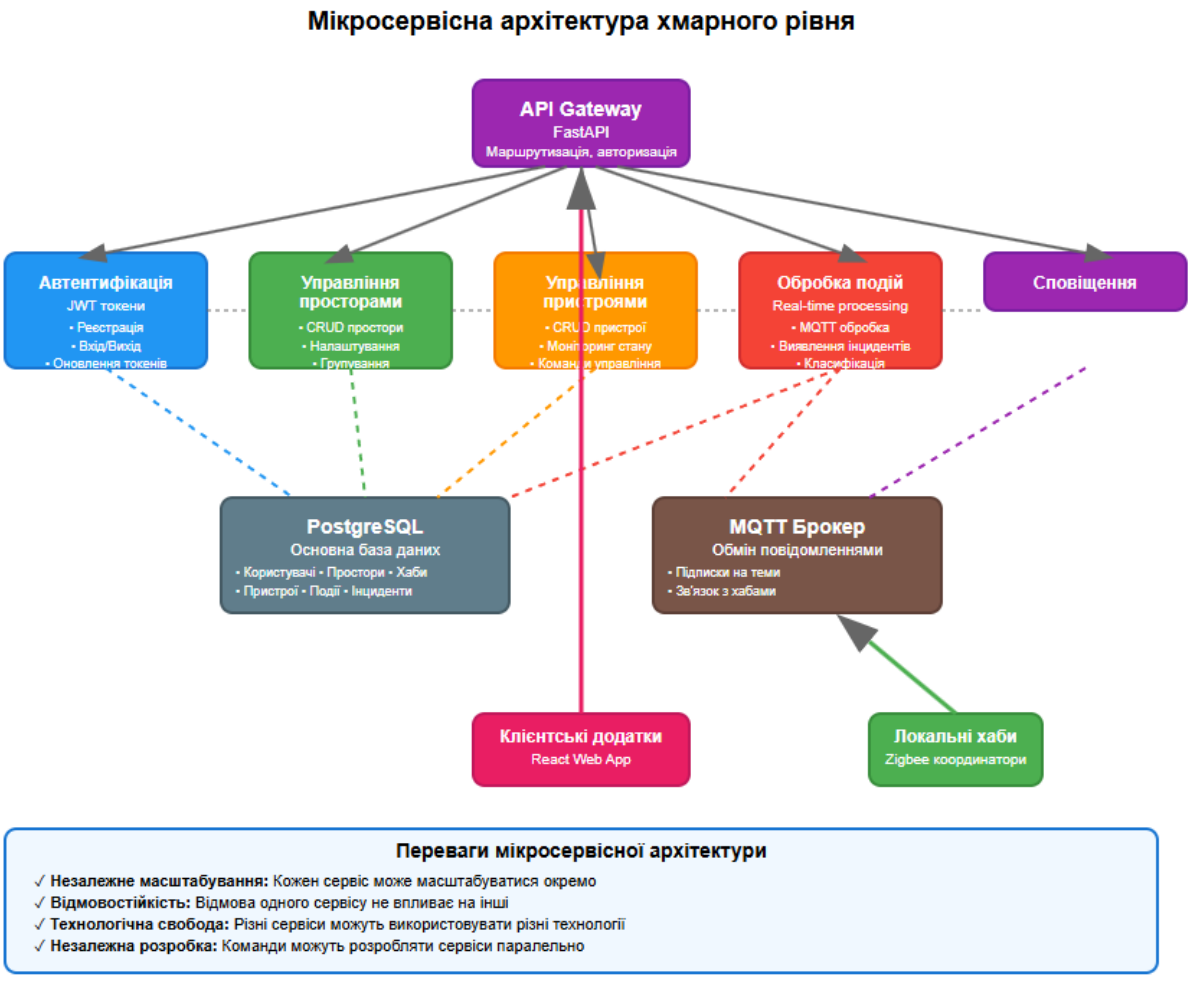

Рисунок 2.7 – Архітектура хмарного рівня.

Висновки за розділом 2

У другому розділі було розроблено архітектуру та спроектовано основні компоненти універсальної платформи безпечного простору. На основі аналізу вимог та існуючих рішень, проведеного в першому розділі, було визначено функціональні та нефункціональні вимоги до системи, що стали основою для подальшого проектування (див. рисунок 2.5).

Ключовим архітектурним рішенням стало використання багаторівневої структури системи, яка складається з локального, транспортного та хмарного рівнів. Такий підхід забезпечує гнучкість, масштабованість та надійність платформи, дозволяючи їй ефективно функціонувати в різноманітних умовах – від невеликих житлових приміщень до великих комерційних об'єктів.

Локальний рівень, побудований на базі технології Zigbee, забезпечує надійний зв'язок між пристроями та хабом-посередником, який виконує роль координатора мережі. Mesh-топологія мережі Zigbee дозволяє розширювати покриття системи та забезпечує стабільну роботу навіть при наявності фізичних перешкод. Розроблена архітектура локального рівня передбачає автономну роботу хаба-посередника при відсутності зв'язку з хмарною платформою, що підвищує надійність системи в цілому (див. рисунок 2.6).

Хмарний рівень реалізовано за принципами мікросервісної архітектури, що забезпечує незалежне масштабування окремих компонентів системи відповідно до навантаження. Визначено основні мікросервіси (автентифікація, управління просторами, обробка подій тощо) та їх взаємодія між собою, що дозволяє розподілити відповідальність та спростити подальшу розробку і підтримку системи (див. рисунок 2.7).

Спроектована база даних забезпечує ефективне зберігання та доступ до інформації про користувачів, простори, хаби, пристрої. Концептуальна та логічна моделі даних відображають ключові сутності системи та зв'язки між ними. Вибір PostgreSQL як системи управління базами даних обґрунтований її надійністю, підтримкою транзакцій, та популярністю.

Детально розглянуто питання безпеки платформи, включаючи механізми автентифікації користувачів, шифрування даних та безпечної комунікації між компонентами системи. Використання JWT для автентифікації забезпечує належний рівень захисту від несанкціонованого доступу при одночасній зручності для користувачів.

Проектування інтерфейсу користувача виконано з урахуванням принципів UX/UI дизайну, що забезпечує інтуїтивно зрозумілий та ефективний доступ до функцій системи. Веб-інтерфейс, побудований на основі React, Redux та Material-UI, надає користувачам можливість управління просторами, пристроями з будь-якого пристрою через веб-браузер.

Запропонована архітектура платформи безпечного простору є універсальною та гнучкою, що дозволяє адаптувати її для різних типів приміщень та різноманітних сценаріїв використання. Вона забезпечує надійний моніторинг, своєчасне виявлення інцидентів та ефективне сповіщення користувачів, що відповідає основним вимогам до сучасних систем безпеки.

РОЗДІЛ 3.
РЕАЛІЗАЦІЯ ПЛАТФОРМИ БЕЗПЕЧНОГО ПРОСТОРУ

Даний розділ присвячено практичній реалізації платформи безпечного простору на основі архітектурних рішень та вимог, описаних у попередніх розділах. Розглянуто технічні аспекти розробки серверної та клієнтської частини, реалізацію механізмів обробки подій від датчиків і камер, систему сповіщень про інциденти, а також інтеграцію з зовнішніми системами.

Реалізація платформи здійснена з використанням сучасних технологій та підходів до розробки програмного забезпечення, що забезпечує надійність, масштабованість та зручність використання системи. Особлива увага приділена аспектам безпеки даних та механізмам виявлення потенційно небезпечних ситуацій у режимі реального часу.

3.1 Розробка серверної частини платформи

3.1.1 Структура проекту та організація коду

Розробка серверної частини платформи безпечного простору базується на архітектурних рішеннях, описаних у Розділі 2. Для забезпечення модульності, масштабованості та зручності підтримки проект організовано відповідно до принципів чистої архітектури з чітким розділенням рівнів та відповідальності.

Структура проекту спроектована за модульним принципом і включає наступні основні компоненти:

1. API модуль - відповідає за обробку HTTP-запитів, валідацію вхідних даних та маршрутизацію запитів до відповідних сервісів.

2. Сервісний рівень - містить бізнес-логіку застосунку, включаючи обробку подій від датчиків, виявлення інцидентів та управління сповіщеннями.

3. Рівень доступу до даних - забезпечує взаємодію з базою даних через репозиторії, які інкапсулюють логіку запитів.

4. Моделі та схеми даних - визначають структуру даних у системі та забезпечують їх перевірку.

5. Модуль фонових завдань - обробляє події, які надходять від пристроїв, та виконує інші асинхронні операції.

6. Модуль конфігурації та безпеки - містить налаштування застосунку та логіку автентифікації та авторизації.

Така структура забезпечує чітке розділення обов'язків та спрощує подальшу підтримку та розширення системи. Наприклад, вся логіка роботи з подіями зосереджена в спеціалізованих модулях, що дозволяє легко модифікувати та розширювати функціональність обробки подій без впливу на інші компоненти системи.

3.1.2 Організація даних у системі

На основі проектування бази даних, описаного в Розділі 2.3, були розроблені моделі даних, які відображають основні сутності системи. Ключовими елементами даних у системі є:

1. Користувачі (Users) - містять інформацію про зареєстрованих користувачів системи, включно з їхніми обліковими даними та контактною інформацією для надсилання сповіщень.

2. Простори (Spaces) - представляють фізичні приміщення або території, які контролюються системою. Кожен простір має власника та тип (квартира, будинок, офіс, тощо).

3. Хаби (Hubs) - центральні пристрої, які координують роботу датчиків у певному просторі. Хаб містить унікальний API-ключ для безпечного зв'язку з платформою.

4. Пристрої (Devices) - датчики та камери, підключені до хабів. Кожен пристрій має тип, унікальний ідентифікатор Zigbee та налаштування.

5. Події (Events) - записи про дані, отримані від пристроїв, які можуть свідчити про певні зміни у просторі (рух, відкриття дверей, задимлення тощо).

6. Інциденти (Incidents) - події, класифіковані як потенційно небезпечні ситуації, які вимагають уваги або реакції користувачів.

7. Сповіщення (Notifications) - повідомлення, які надсилаються користувачам про виявлені інциденти через різні канали зв'язку.

3.1.3 Реалізація API інтерфейсу

API інтерфейс платформи реалізований за допомогою фреймворку FastAPI, який забезпечує високу продуктивність, автоматичну валідацію даних та генерацію документації за стандартом OpenAPI. Основні групи ендпоінтів включають:

1. Автентифікація та управління користувачами - ендпоінти для реєстрації, автентифікації та управління профілями користувачів:

- POST /api/users - реєстрація нового користувача
- POST /api/token - автентифікація та отримання токенів доступу
- GET /users/me - отримання інформації про поточного користувача
- PUT /users/me – оновлення інформації у поточного користувача

2. Управління просторами - ендпоінти для створення та налаштування просторів:

- POST /spaces - створення нового простору
- GET /spaces - отримання списку просторів користувача
- GET /spaces/{space_id} - отримання детальної інформації про простір
- PUT /spaces/{space_id} - оновлення налаштувань простору
- DELETE /spaces/{space_id} - видалення простору

3. Управління пристроями - ендпоінти для додавання та налаштування пристроїв у просторах:

- POST /spaces/{space_id}/devices - додавання нового пристрою
- GET /spaces/{space_id}/devices - отримання списку пристроїв у просторі
- GET /devices/{device_id} - отримання інформації про пристрій
- PUT /devices/{device_id} - оновлення налаштувань пристрою
- DELETE /devices/{device_id} - видалення пристрою

4. Робота з подіями та інцидентами - ендпоінти для отримання інформації про події та управління інцидентами:

- GET /spaces/{space_id}/events - отримання історії подій у просторі

- GET /spaces/{space_id}/incidents - отримання списку інцидентів у просторі

- GET /incidents/{incident_id} - отримання детальної інформації про інцидент

- PUT /incidents/{incident_id}- оновлення статусу інциденту (підтвердження/вирішення)

5. Налаштування сповіщень - ендпоінти для управління налаштуваннями сповіщень:

- GET /notification-settings - отримання налаштувань сповіщень користувача

- PUT /notification-settings - оновлення налаштувань сповіщень

Всі ендпоінти, крім автентифікації та реєстрації, вимагають авторизації за допомогою JWT-токена, який передається в заголовку Authorization.

3.2 Реалізація обробки подій від датчиків

3.2.1 Архітектура підсистеми обробки подій

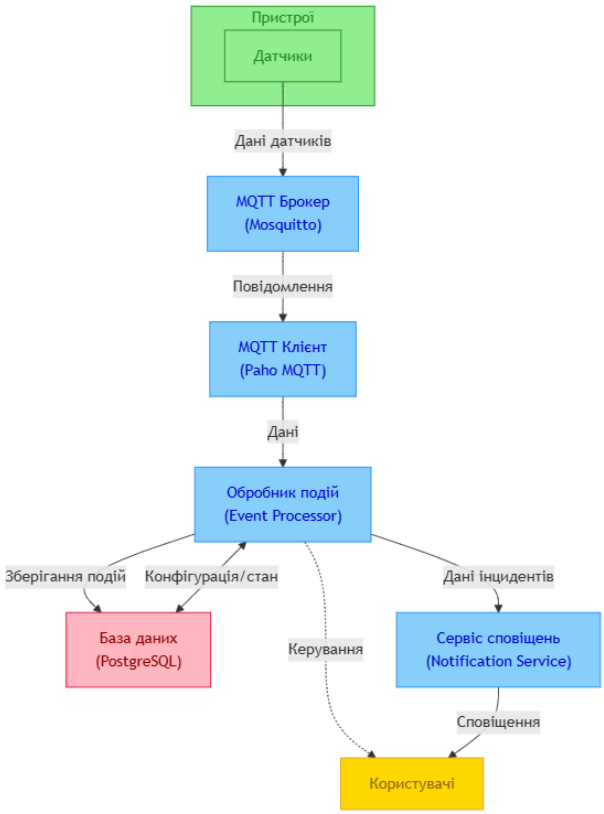

Рисунок 3.1 - Схема взаємодії компонентів підсистеми обробки подій.

Підсистема обробки подій складається з наступних компонентів:

1. MQTT брокер - центральний компонент, який забезпечує обмін повідомленнями між пристроями (датчиками і камерами) та серверною частиною платформи.

2. MQTT клієнт - компонент серверної частини, який підписується на теми MQTT та отримує повідомлення від пристроїв.

3. Обробник подій - компонент, який обробляє отримані повідомлення, класифікує події та виявляє інциденти.

4. Сервіс сповіщень - компонент, який відправляє сповіщення користувачам про виявлені інциденти.

3.2.2 Алгоритм обробки даних від датчиків

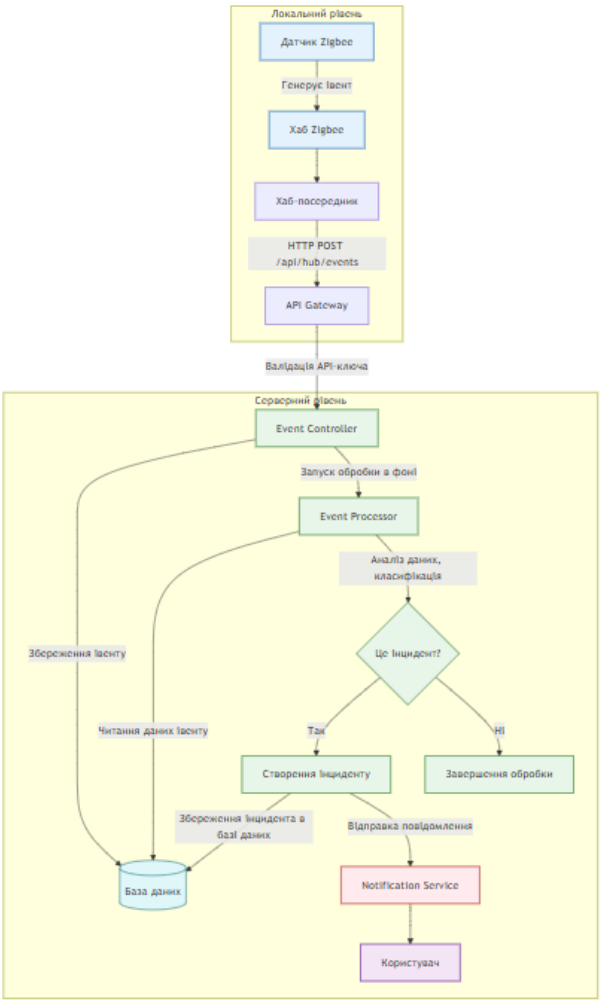

Рисунок 3.2 - Алгоритм обробки даних від датчиків.

Основні етапи алгоритму:

1. Отримання повідомлення - дані від датчика надходять до MQTT брокера і передаються MQTT клієнту, який підписаний на відповідну тему.

2. Збір та перевірка даних - перевірка формату повідомлення та наявності необхідних полів. Якщо повідомлення некоректне, воно ігнорується.

3. Ідентифікація пристрою - визначення, від якого пристрою надійшло повідомлення, та перевірка його наявності в системі.

4. Збереження події - створення запису про подію в базі даних з усіма отриманими параметрами.

5. Аналіз події - аналіз даних події для визначення, чи представляє вона потенційну небезпеку.

6. Класифікація інциденту - якщо подія класифікована як потенційно небезпечна, визначається тип та серйозність інциденту.

7. Створення інциденту - створення запису про інцидент в базі даних.

8. Відправка сповіщень - повідомлення користувачів про виявлений інцидент через налаштовані канали зв'язку.

Для різних типів датчиків використовуються спеціалізовані алгоритми аналізу даних. Наприклад:

- Датчики диму/пожежі: аналізуються значення концентрації диму/температури, і якщо вони перевищують встановлені пороги, створюється інцидент типу "пожежа".
- Датчики затоплення: при виявленні води створюється інцидент "затоплення".

3.3 Реалізація клієнтської частини

3.3.1 Архітектура клієнтської частини

Клієнтська частина платформи побудована за принципом SPA (Single Page Application) з використанням наступного технологічного стеку:

- React - бібліотека для побудови інтерактивних користувацьких інтерфейсів.
- Redux - бібліотека для управління станом застосунку.
- React Router - бібліотека для реалізації маршрутизації у React-застосунку.

- Material-UI - бібліотека компонентів, реалізованих відповідно до принципів Material Design.

- Axios - бібліотека для виконання HTTP-запитів до API.

Архітектура клієнтської частини побудована за принципом модульності та включає наступні основні компоненти:

1. Компоненти інтерфейсу - реалізують відображення різних елементів інтерфейсу.

2. Контейнери - з'єднують компоненти інтерфейсу з даними та діями.

3. Дії (Actions) - описують дії, які можуть бути виконані в системі.

4. Редуктори (Reducers) - визначають, як змінюється стан застосунку у відповідь на дії.

5. Сервіси - реалізують взаємодію з API серверної частини.

6. Утиліти - містять допоміжні функції та константи.

3.3.2 Основні екрани інтерфейсу користувача

Інтерфейс користувача платформи включає наступні основні екрани:

1. Головна сторінка - надає загальний огляд стану просторів користувача, відображає активні інциденти та останні події.

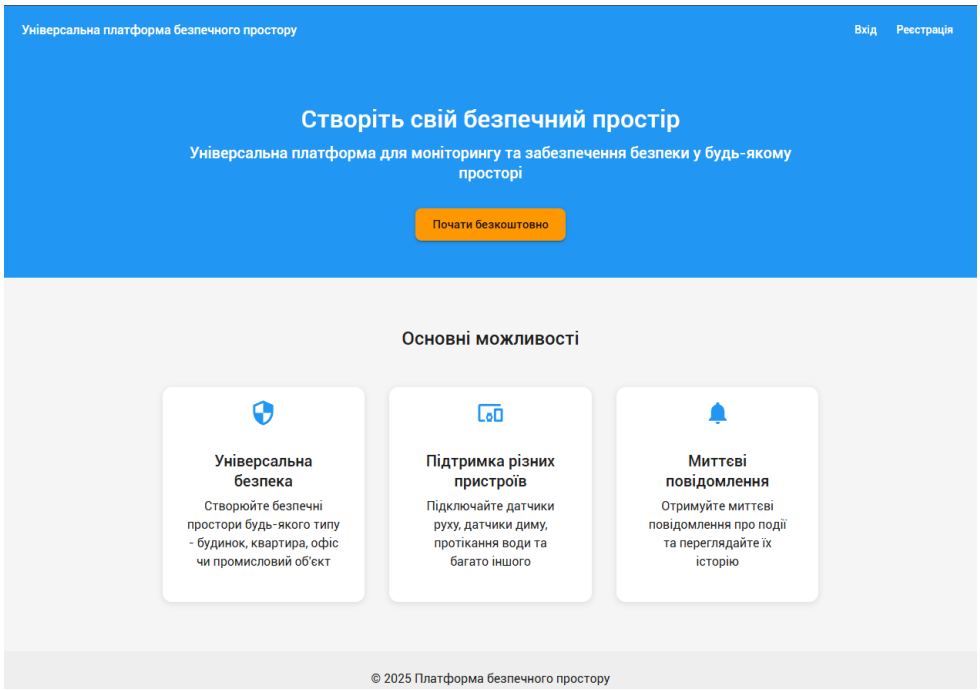

Рисунок 3.3 – Головна сторінка.

2. Панель керування - надає загальний огляд стану просторів користувача, відображає активні інциденти та останні події.

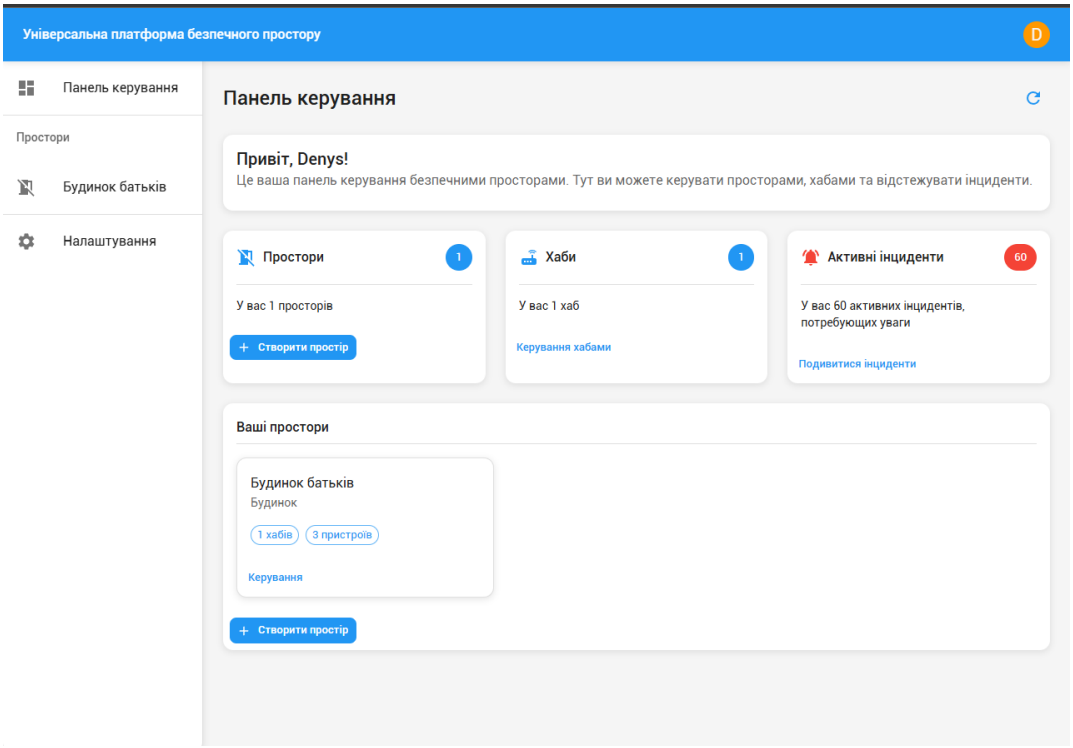

Рисунок 3.4 – Панель керування.

3. Сторінка простору - відображає детальну інформацію про конкретний простір, включаючи список пристроїв, історію подій та інцидентів.

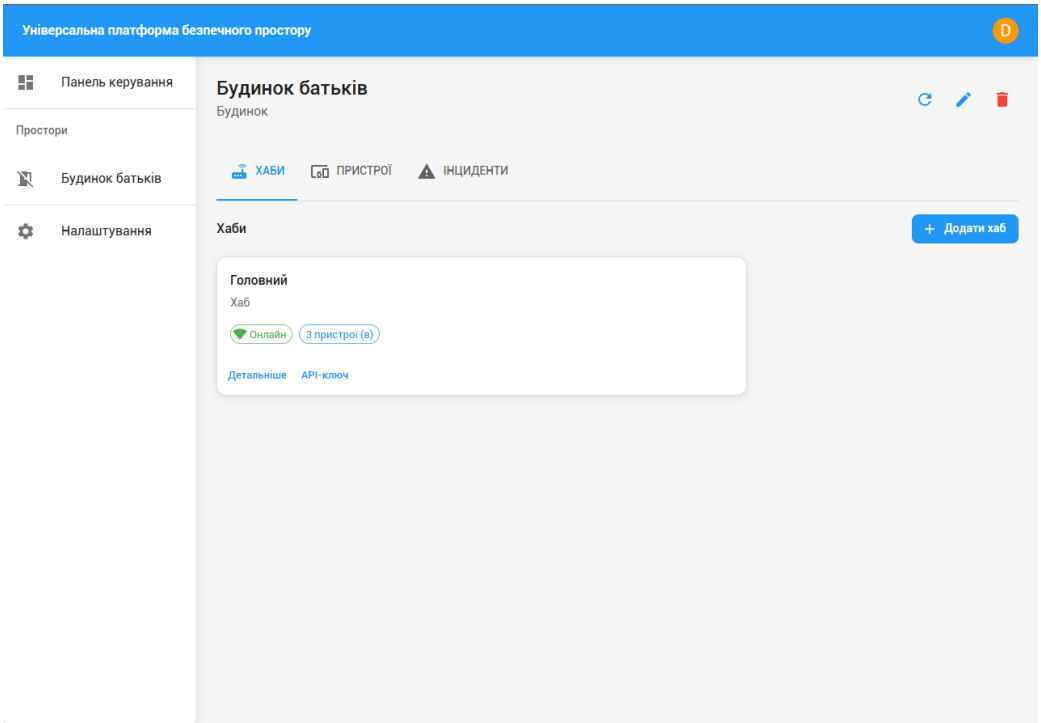

Рисунок 3.5 – Дизайн сторінки простору.

4. Сторінка хабу - надає інформацію про конкретний хаб, його пристрої, дає можливість для отримання API-ключа щоб підключити його до системи.

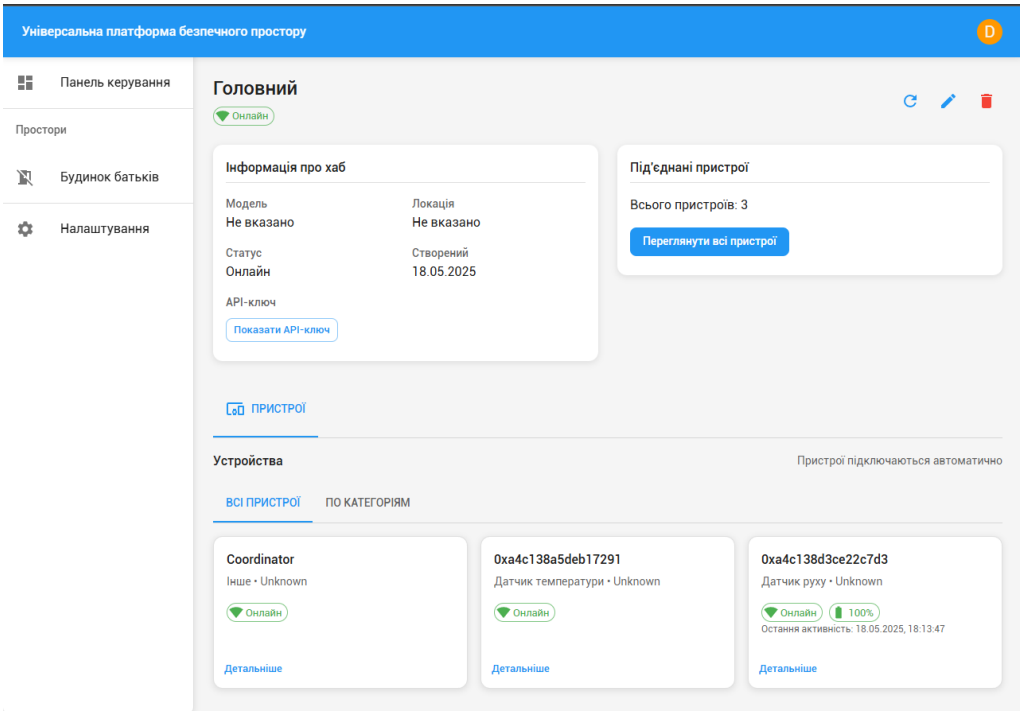

Рисунок 3.6 – Дизайн сторінки хабу.

5. Сторінка пристрою - надає інформацію про конкретний пристрій, його стан та налаштування.

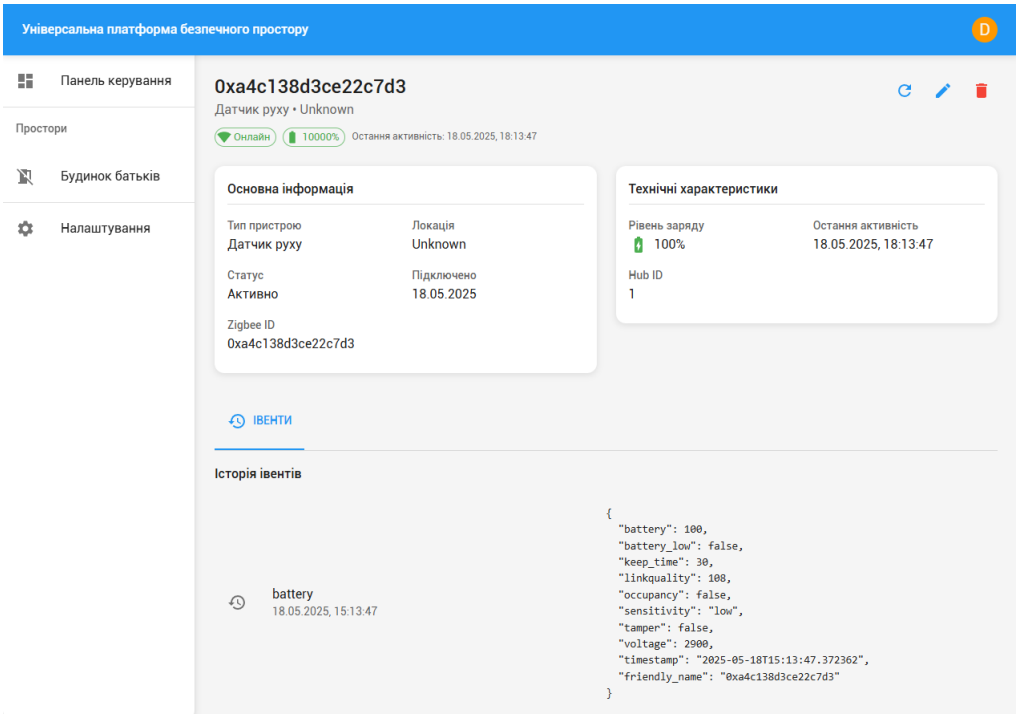

Рисунок 3.7 – Дизайн сторінки пристрою.

6. Сторінка інцидентів - відображає список інцидентів з можливістю фільтрації та сортування, а також детальну інформацію про кожен інцидент.

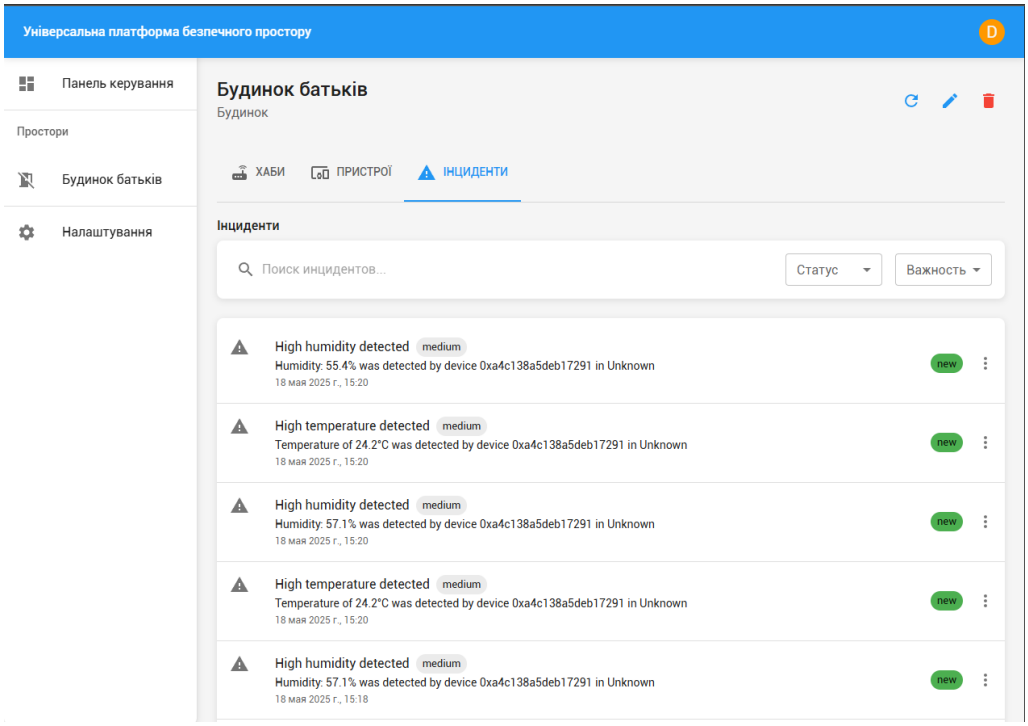

Рисунок 3.8 – Дизайн сторінки простору.

7. Сторінка налаштувань - дозволяє налаштовувати параметри системи, включаючи налаштування сповіщень.

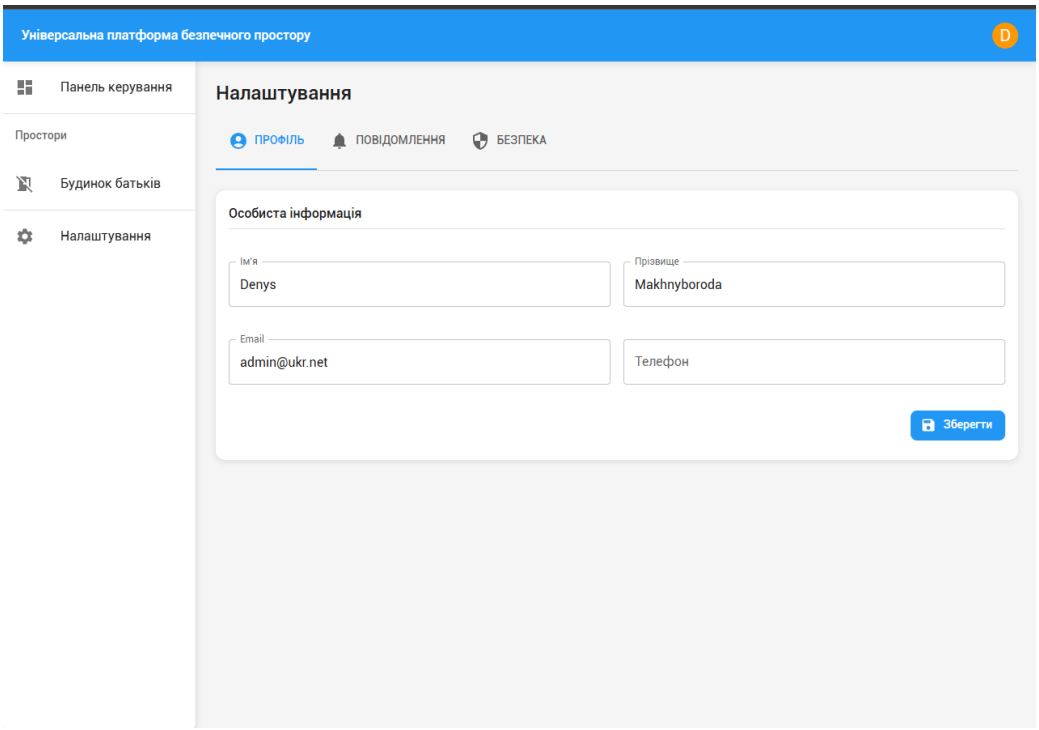

Рисунок 3.9 – Дизайн сторінки налаштувань.

3.4 Інтеграція з зовнішніми системами та пристроями

3.4.1 Підтримка екосистеми Zigbee

Ключовою перевагою розробленої платформи є підтримка протоколу Zigbee, що забезпечує можливість підключення широкого спектру датчиків та пристроїв різних виробників. Протокол Zigbee є відкритим стандартом бездротового зв'язку, що набув значного поширення в сфері IoT та розумних будинків.

Використання Zigbee в якості основного протоколу для взаємодії з пристроями надає наступні переваги:

1. Широке покриття ринку пристроїв - протокол Zigbee підтримується численними виробниками, серед яких Philips Hue, IKEA TRÅDFRI, Xiaomi Aqara, Samsung SmartThings, Sonoff та інші. Це дозволяє інтегрувати в систему більше тисячі різноманітних пристроїв.

2. Різноманітність типів датчиків - завдяки Zigbee платформа може працювати з наступними типами датчиків:

- Датчики руху (PIR-сенсори)
- Датчики відкриття дверей та вікон
- Датчики розбиття скла
- Датчики диму та чадного газу
- Датчики протікання води
- Датчики температури та вологості
- Датчики якості повітря
- Датчики освітленості
- Датчики вібрації та нахилу
- Кнопки тривоги та SOS-сигналізації

3. Mesh-мережева архітектура - Zigbee використовує mesh-топологію, де кожен пристрій з постійним живленням може виступати ретранслятором сигналу для інших пристроїв. Це забезпечує надійне покриття навіть у великих приміщеннях.

4. Енергоефективність - датчики з батарейним живленням можуть функціонувати від декількох місяців до кількох років без заміни батарей.

Для забезпечення роботи з пристроями Zigbee в платформі реалізовано спеціальний програмний модуль, який взаємодіє з координатором Zigbee через інтерфейс USB. Цей модуль відповідає за отримання даних з датчиків та їх перетворення у внутрішній формат системи, та за моніторинг стану пристроїв (рівень заряду батареї, якість зв'язку)

Висновки за розділом 3

У третьому розділі представлено практичну реалізацію універсальної платформи безпечного простору на основі проектних рішень, розроблених у попередньому розділі. Розроблено серверну та клієнтську частини системи, механізми обробки подій від датчиків, а також забезпечено інтеграцію з зовнішніми системами та пристроями.

Реалізація серверної частини платформи виконана із застосуванням сучасних технологій та підходів до розробки програмного забезпечення. Структура проекту організована за модульним принципом з чітким розмежуванням відповідальності між компонентами, що спрощує подальшу підтримку та розширення системи. Використання FastAPI забезпечує високу продуктивність, автоматичну валідацію даних та генерацію документації за стандартом OpenAPI.

Реалізовано повнофункціональний API інтерфейс, який надає клієнтським додаткам доступ до всіх функцій системи, включаючи управління користувачами, просторами, пристроями, подіями та інцидентами. Впроваджено механізми автентифікації та авторизації на основі JWT-токенів, що забезпечує захист від несанкціонованого доступу.

Особливу увагу приділено реалізації підсистеми обробки подій від датчиків, яка є ключовим компонентом платформи безпечного простору. Розроблено архітектуру цієї підсистеми, що включає MQTT брокер, MQTT клієнт, обробник подій та сервіс сповіщень. Реалізовано алгоритм обробки даних від датчиків, який забезпечує ефективне виявлення та класифікацію

інцидентів. Для різних типів датчиків впроваджено спеціалізовані алгоритми аналізу даних, що підвищує точність виявлення потенційно небезпечних ситуацій.

Клієнтська частина платформи реалізована у вигляді одсторінкового додатку (SPA) з використанням React, Redux та Material-UI. Розроблено основні екрани інтерфейсу користувача, включаючи головну сторінку, сторінки простору, пристрою, інцидентів та налаштувань. Інтерфейс надає користувачам інтуїтивно зрозумілий доступ до всіх функцій системи та забезпечує оперативне отримання інформації про стан просторів та виявлені інциденти.

Важливим аспектом реалізації платформи є інтеграція з зовнішніми системами та пристроями. Впроваджено підтримку протоколу Zigbee, що дозволяє підключати широкий спектр датчиків та пристроїв різних виробників. Реалізовано програмний модуль для взаємодії з координатором Zigbee, який забезпечує отримання даних з датчиків, надсилання команд керування виконавчим пристроям та моніторинг стану пристроїв.

Ручне тестування системи показало її надійність, швидкодію та зручність використання, що підтверджує відповідність розробленої платформи поставленим цілям та завданням. Завдяки використанню сучасних технологій та архітектурних рішень, платформа має потенціал для подальшого розвитку та масштабування, що робить її конкурентоспроможним рішенням на ринку систем безпеки.

ВИСНОВКИ

У рамках кваліфікаційної роботи було розроблено універсальну платформу безпечного простору, яка дозволила інтегрувати різні типи датчиків та камер для моніторингу та виявлення інцидентів у режимі реального часу. Робота була спрямована на створення гнучкої системи, що забезпечила ефективне налаштування, моніторинг та сповіщення про потенційно небезпечні ситуації в різноманітних просторах – від житлових приміщень до промислових об'єктів.

Розроблено архітектуру універсальної платформи, яка базувалася на багаторівневому принципі з локальним, транспортним та хмарним рівнями. Обґрунтовано вибір протоколу Zigbee як оптимального рішення для бездротового зв'язку між пристроями завдяки його низькому енергоспоживанню, mesh-архітектурі та широкій сумісності з пристроями різних виробників.

Спроектовано та реалізовано серверну частину платформи з використанням мікросервісної архітектури на базі Python, FastAPI, SQLAlchemy та PostgreSQL. Створено повнофункціональний API інтерфейс, який забезпечив доступ до всіх функцій системи, включаючи управління користувачами, просторами, пристроями та обробку подій.

Створено клієнтську частину у вигляді одностороінкового веб-додатку з використанням React, Redux та Material-UI, що забезпечило інтуїтивний інтерфейс для управління всіма функціями системи.

Реалізовано інтеграцію з екосистемою Zigbee, що дозволило підключати широкий спектр датчиків різних виробників.

Подальший розвиток платформи може включати розширення функціональності за рахунок додавання підтримки інших протоколів IoT та інтеграції з існуючими системами "розумного дому".

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Application Security Market Size, Share Report 2025-2033. *Market Research Company, Reports and Consulting Services/ IMARC*. URL: https://www.imarcgroup.com/application-security-market (Дата звернення: 01.03.2025).
2. Security of the Internet of Things: Vulnerabilities, Attacks, and Countermeasures. *IEEEXplore*. URL: https://ieeexplore.ieee.org/document/8897627 (Дата звернення: 02.03.2025).
3. Multi-Factor Authentication: A Survey. Cryptography. *MDPI*. URL: https://www.mdpi.com/2410-387X/2/1/1 (Дата звернення: 02.03.2025).
4. Зростання біометричних технологій у контролі доступу у 2024 році. *GreenVision - Україна*. URL: https://greenvision.ua/ua/blog/tekhnolohichni-trendy-v-kontroli-fizychnoho-dostupu-rozvytok-biometriyi-shtuchnoho-intelektu-ta-mobilnykh-rishen (Дата звернення: 05.03.2025).
5. Zigbee. *The Fast Mode*. URL: https://www.thefastmode.com/wiki-networking/5641-zigbee
6. Типи девайсів в Zigbee - Wikipedia. *Wikipedia, the free encyclopedia*. URL: https://en.wikipedia.org/wiki/Zigbee#Device_types_and_operating_modes (Дата звернення: 08.03.2025).

ДОДАТКИ

Додаток А

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет імені В. Н. Каразіна

Навчально-науковий інститут комп'ютерних наук та штучного інтелекту
Кафедра комп'ютерних систем та робототехніки
Рівень вищої освіти (освітньо-кваліфікаційний рівень) **Бакалавр**
Галузь знань: 12 – Інформаційні технології
Спеціальність: 123 «Комп'ютерна інженерія»
Освітня програма «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
комп'ютерних
систем та робототехніки
к. ф.-м. н., доц. ХРУСЛОВ М. М.
«02» жовтня 2024року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Махнибороди Дениса Володимировича
(прізвище, ім'я, по батькові студента)

1. Тема роботи____Створення універсальної платформи безпечного простору
керівник роботи __Рало Олександр Миколайович, старший викладач_,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від ***16 квітня 2025 року №4101-5/962***
2. Строк подання студентом роботи ***30 травня 2025 року***

3. Перелік питань, які потрібно розробити
1. Дослідження основних типів систем фізичної безпеки в приміщеннях та громадських просторах.
2. Огляд різних типів датчиків (руху, протікання води, диму, газу) та їх застосування в системах безпеки.
3. Вивчення протоколів бездротового зв'язку для IoT-пристроїв у системах безпеки.
4. Розробка архітектури універсальної платформи для інтеграції різнорідних пристроїв безпеки.
5. Проектування системи сповіщень.
6. Розробка користувацького інтерфейсу для моніторингу та управління пристроями безпеки.

4. План роботи

№ з/п	Назви етапів роботи	Термін виконання етапів роботи
1	Затвердження теми роботи	05.09.2024 - 02.10.2024
2	Аналіз та пошук методичної літератури щодо систем безпеки із використанням IoT-пристроїв	02.10.2024 - 24.10.2024
3	Огляд і аналіз існуючих систем фізичної безпеки та моніторингу в приміщеннях	25.10.2024 - 09.11.2024
4	Визначення функціональних вимог до універсальної платформи безпечного простору	10.11.2024 - 24.11.2024
5	Розробка архітектури платформи для інтеграції різних пристроїв безпеки	25.10.2024 - 08.11.2024
6	Вибір апаратного забезпечення та протоколів для реалізації системи безпеки	09.11.2024 - 29.11.2024
7	Проектування серверної частини для обробки та зберігання даних з пристроїв	30.11.2024- 31.12.2024
8	Розробка клієнтського інтерфейсу (frontend) платформи	01.01.2025- 01.02.2025
9	Розробка системи сповіщень та реагування на інциденти	01.02.2025- 01.03.2025
10	Тестування функціональності та надійності системи	01.03.2025- 15.03.2025
11	Підготовка і оформлення звітних матеріалів. Написання статті за матеріалами кваліфікаційної роботи	15.03.2025- 01.04.2025
12	Підготовка і оформлення звітних матеріалів та додатків кваліфікаційної роботи. Оформлення списку літератури	01.04.2025- 30.04.2025
13	Оформлення пояснювальної записки кваліфікаційної роботи відповідно вимогам до звітів про НДР.	01.05.2025- 30.05.2025
14	Оформлення звіту про переддипломну практику	01.05.2025- 30.05.2025
15	Представлення кваліфікаційної роботи керівнику та рецензенту	30.05.2025

5. Дата видачі завдання ***02 жовтня 2024 року.***

Студент **Махниборода Д.В** _____
ініціали, прізвище підпис

Керівник роботи **Рало О.М.** _____
ініціали, прізвище підпис

Додаток Б

Затверджую

_____

«_____»_____2025 р.

Технічне завдання

на розробку системи

«Універсальної платформи безпечного простору».

1.	Введення	1.1 Назва роботи – Створення універсальної платформи безпечного простору. 1.2. Галузь застосування: Інформаційні технології, системи безпеки, моніторинг приміщень та просторів.
2.	Підстава для розробки	2.1. Навчальний план за спеціальністю 123 – Комп’ютерна інженерія 2.2. Завдання на кваліфікаційну роботу бакалавра №4101-5/962 від «16» квітня 2025 року
3.	Призначення розробки	3.1. Мета розробки: Створення універсальної платформи для безпечних просторів, яка дозволяє інтегрувати різні типи датчиків для моніторингу та виявлення інцидентів у режимі реального часу. 3.2. Призначення розробки: Розробка гнучкої системи, що забезпечує ефективне налаштування, моніторинг та сповіщення про потенційно небезпечні ситуації в різноманітних просторах - від житлових приміщень до промислових об'єктів. 3.3. Вхідні дані: Дані від підключених датчиків (датчики руху, відкриття, розбиття скла, диму, газу, протікання води, температури/вологості, тощо) Конфігураційні параметри просторів та пристроїв

		3.4. Вихідні дані розробки: Інформація про виявлені інциденти та порушення безпеки Сповіщення через веб-інтерфейс Історія подій та інцидентів Статистика та аналітика щодо безпеки просторів
4.	Технічні вимоги до програмного виробу	4.1. Функціональні вимоги: – Створення та налаштування різних типів просторів (будинок, квартира, офіс, склад, тощо) – Додавання, налаштування та видалення датчиків та камер – Підтримка різних типів датчиків (руху, відкриття, розбиття скла, диму, газу, протікання води, тощо) – Моніторинг стану пристроїв (рівень заряду батареї, якість з'єднання) – Неперервний збір та аналіз даних з датчиків та камер – Виявлення потенційно небезпечних ситуацій згідно з налаштованими правилами – Класифікація інцидентів за типом та рівнем критичності 4.2. Нефункціональні вимоги: – Забезпечення безперебійної роботи системи 24/7 – Автоматичне відновлення після збоїв – Можливість розширення системи від невеликих до великих об'єктів – Підтримка одночасного підключення сотень пристроїв без погіршення продуктивності – Гнучке налаштування для різних типів приміщень та сценаріїв використання 4.3. Вимоги до інтеграції:

		– Підтримка пристроїв, що працюють за протоколом Zigbee – Забезпечення надійного зв'язку та низького енергоспоживання – Можливість інтеграції з існуючими системами безпеки та автоматизації – API для інтеграції з зовнішніми сервісами 4.4. Вимоги до безпеки: – Надійна автентифікація та авторизація користувачів – Захист від несанкціонованого доступу – Регулярні оновлення для усунення вразливостей
5.	Вимоги до програмної документації	Документацією до виробу «Система розпізнавання перешкод для автопілоту» вважати: 1) Опис основних вимог та функціональності системи (представити у вигляді Додатку Б пояснювальної записки до кваліфікаційної роботи). 2) Програму і методику випробувань розробленої програми (представити як додаток В до пояснювальної записки до кваліфікаційної роботи). 3) Опис розробленої системи (представити в розділі 3 пояснювальної записки до кваліфікаційної роботи).
6.	Вимоги до техніко-економічних показників	Документацією до виробу «Система розпізнавання перешкод для автопілоту» вважати: 1) Технічне завдання на розробку універсальної платформи безпечного простору (представити у вигляді Додатку Б до пояснювальної записки до кваліфікаційної роботи). 2) Опис розробленої універсальної платформи безпечного простору (представити в розділі 3 пояснювальної записки до кваліфікаційної роботи). 3) Джерела базової інформації.

<table>
<tr><td>7.</td><td>Стадії і етапи розробки</td><td>Дата</td><td>Назва етапу</td></tr>
<tr><td></td><td></td><td>02.10.2024 - 24.10.2024</td><td>Аналіз та пошук методичної літератури щодо систем безпеки із використанням IoT-пристроїв</td></tr>
<tr><td></td><td></td><td>25.10.2024 - 09.11.2024</td><td>Огляд і аналіз існуючих систем фізичної безпеки та моніторингу в приміщеннях</td></tr>
<tr><td></td><td></td><td>10.11.2024 - 24.11.2024</td><td>Розробка технічного завдання.</td></tr>
<tr><td></td><td></td><td>25.10.2024 - 08.11.2024</td><td>Розробка архітектури платформи для інтеграції різних пристроїв безпеки</td></tr>
<tr><td></td><td></td><td>30.11.2024 - 01.03.2025</td><td>Проектування серверної частини для обробки та зберігання даних з пристроїв. Розробка клієнтського інтерфейсу платформи</td></tr>
<tr><td></td><td></td><td>01.03.2025 - 15.03.2025</td><td>Тестування функціональності та надійності системи
Підготовка технічного звіту</td></tr>
<tr><td></td><td></td><td>15.03.2025 – 27.05.2025</td><td>Розробка пояснювальної записки.</td></tr>
<tr><td></td><td></td><td>31.03.2025 – 27.05.2025</td><td>Представлення кваліфікаційної роботи керівнику та рецензенту.</td></tr>
<tr><td></td><td></td><td>15.05.2025 – 31.05.2025</td><td>Оформлення пояснювальної записки та підготовка презентації.</td></tr>
<tr><td>8.</td><td>Порядок контролю і приймання програмного продукту</td><td colspan="2">– Щотижневі звіти про хід розробки
– Демонстрація проміжних результатів кожні два тижні
– Перевірка відповідності реалізованої системи функціональним та нефункціональним вимогам
– Тестування системи в різних умовах та сценаріях використання
– Підготовка документації користувача та адміністратора</td></tr>
</table>

Виконавець
Студент групи КІ-41
Махниборода Д. В.

Замовник
Старший викладач
Рало О.М.

Додаток В

Програма і методика випробувань програмного виробу

«Універсальна платформа безпечного простору»

1. Об’єкт випробувань

1.1. Назва розробленого програмного виробу: «Універсальна платформа безпечного простору».

1.2. Галузь застосування: Інформаційні технології, системи безпеки, моніторинг приміщень та просторів.

1.3. Перераховані відомості запозичуються з відповідних розділів Технічного завдання.

2. Мета випробувань

Перевірка відповідності функціональних можливостей системи заявленим функціональним можливостям в технічному завданні (Додаток Б до пояснювальної записки до кваліфікаційної роботи).

3. Загальні положення

3.1. Підстави для проведення випробувань

Підставою для проведення випробувань є наказ про призначення атестаційної комісії.

3.2. Місце і тривалість випробувань

Приймальні (приймально-здавальні) випробування проводяться дистанційно в період роботи атестаційної комісії.

3.3. Обсяг випробувань

Приймальні випробування програмного виробу проводяться в обсязі відповідному цієї програми і методики випробувань.

3.4. Організації, які беруть участь у випробуваннях

Приймальні випробування проводяться атестаційною комісією напередодні засідання (або в процесі засідання) за участю Замовника, Виконавці та інших осіб, присутніх на засіданні.

4. Вимоги до програми або програмного виробу

4.1. Функціональні вимоги:

- Можливість створення та налаштування різних типів просторів (будинок, квартира, офіс, склад, тощо).
- Додавання, налаштування та видалення датчиків та камер.
- Підтримка різних типів датчиків (руху, відкриття, розбиття скла, диму, газу, протікання води, тощо).
- Моніторинг та виявлення інцидентів у режимі реального часу.
- Сповіщення користувачів про виявлені інциденти через веб-інтерфейс.
- Збереження історії всіх подій та інцидентів.

4.2. Нефункціональні вимоги:

- Система повинна бути надійною і функціональною у різних умовах експлуатації, забезпечуючи безперебійну роботу 24/7.
- Система повинна зберігати базову функціональність навіть при відсутності інтернет-з'єднання.
- Система повинна забезпечувати шифрування даних при передачі та зберіганні.
- Система повинна мати надійну автентифікацію та авторизацію користувачів.
- Система повинна швидко реагувати на події - затримка між виявленням події та відправкою сповіщення не повинна перевищувати 3 секунди.
- Система повинна підтримувати масштабування від невеликих до великих об'єктів.

4.3. Вимоги до інтеграції:

- Система повинна підтримувати пристрої, що працюють за протоколом Zigbee.
- Система повинна забезпечувати можливість інтеграції з існуючими системами безпеки та автоматизації.

- Система повинна надавати API для інтеграції з зовнішніми сервісами.

5. Вимоги до програмної документації

Документацією до виробу «Універсальна платформа безпечного простору» вважати:

Технічне завдання на розробку системи (представити у вигляді Додатку Б пояснювальної записки до кваліфікаційної роботи).

Програму і методику випробувань розробленої програми (представити як додаток В до пояснювальної записки до кваліфікаційної роботи).

Опис розробленої системи (представити в розділі 3 пояснювальної записки до кваліфікаційної роботи).

6. Засоби і порядок випробувань

6.1. Засоби випробувань

Випробування проводяться на технічних засобах, серед яких сервер для розгортання бекенду, персональний комп'ютер для тестування веб-інтерфейсу, тестові датчики (датчик руху, датчик відкриття дверей, датчик протікання води, датчик диму, датчик температури/вологості) та координатор Zigbee для інтеграції з датчиками.

Випробування проводяться з використанням програмних засобів, серед яких Python, FastAPI, PostgreSQL, MQTT, React, Redux, Postman.

6.2. Порядок проведення випробувань

Як правило, випробування проводяться в два етапи:

- Ознайомчий (1-й етап);
- Власне випробування програмного виробу (2-й етап).

Перелік перевірок, що проводяться на 1 етапі випробувань, включає в себе:

Перевірка функціональності:

Управління користувачами:

– Реєстрація нового користувача: система дозволяє створити новий обліковий запис з унікальним email.

- Автентифікація користувача: система перевіряє облікові дані та надає доступ до функцій відповідно до прав користувача.
- Зміна даних профілю: система дозволяє користувачу оновлювати свої персональні дані.

Управління просторами:

- Створення нового простору: система дозволяє створювати нові простори різних типів.
- Отримання списку просторів: система відображає всі простори, що належать користувачу.
- Редагування налаштувань простору: система дозволяє змінювати параметри простору.
- Видалення простору: система дозволяє видаляти простори.

Управління пристроями:

- Додавання нового пристрою: система дозволяє додавати нові пристрої до простору.
- Отримання списку пристроїв: система відображає всі пристрої, що належать до простору.
- Налаштування параметрів пристрою: система дозволяє змінювати параметри пристрою.
- Видалення пристрою: система дозволяє видаляти пристрої з простору.
- Обробка подій та інцидентів:
- Генерація тестової події: система створює запис про подію при отриманні даних від датчика.
- Виявлення інциденту: система класифікує події як інциденти згідно з налаштованими правилами.
- Відправка сповіщення: система надсилає сповіщення про інцидент через налаштовані канали.

– Обробка інциденту: система дозволяє користувачу змінювати статус інциденту.

Перевірка інтеграції:

Підтримка Zigbee: система може взаємодіяти з пристроями Zigbee через координатор.

API інтерфейс: система надає доступ до функціональності через API.

Для роботи необхідні:

- Сервер з розгорнутою серверною частиною платформи.
- Комп'ютер з доступом до веб-інтерфейсу платформи.
- Тестові датчики, підключені через координатор Zigbee.

Порядок проведення випробувань:

– Розгортання серверної частини платформи на тестовому сервері.

– Запуск веб-інтерфейсу та перевірка доступу до нього.

– Реєстрація тестового користувача та вхід у систему.

– Тестування API інтерфейсу.

Для тестування системи було зібрано просту екосистему Zigbee, яка складалася з 3 основних модулів:

Таблиця В.1

Компоненти системи

Товар	Кількість	Сума(грн)
Адаптер SONOFF Zigbee 3.0	1 шт.	990
Температурний датчик Zigbee IH-K009	1 шт.	340
Датчик руху TUYA	1 шт.	350
		Всього: 1680

Тест:

Перевірка реєстрації та автентифікації користувача.

Перевірка створення та налаштування простору.

Перевірка створення нового хабу.

Перевірка програми-посередника.

Перевірка API інтерфейсу.

Перевірка реєстрації та автентифікації користувача:

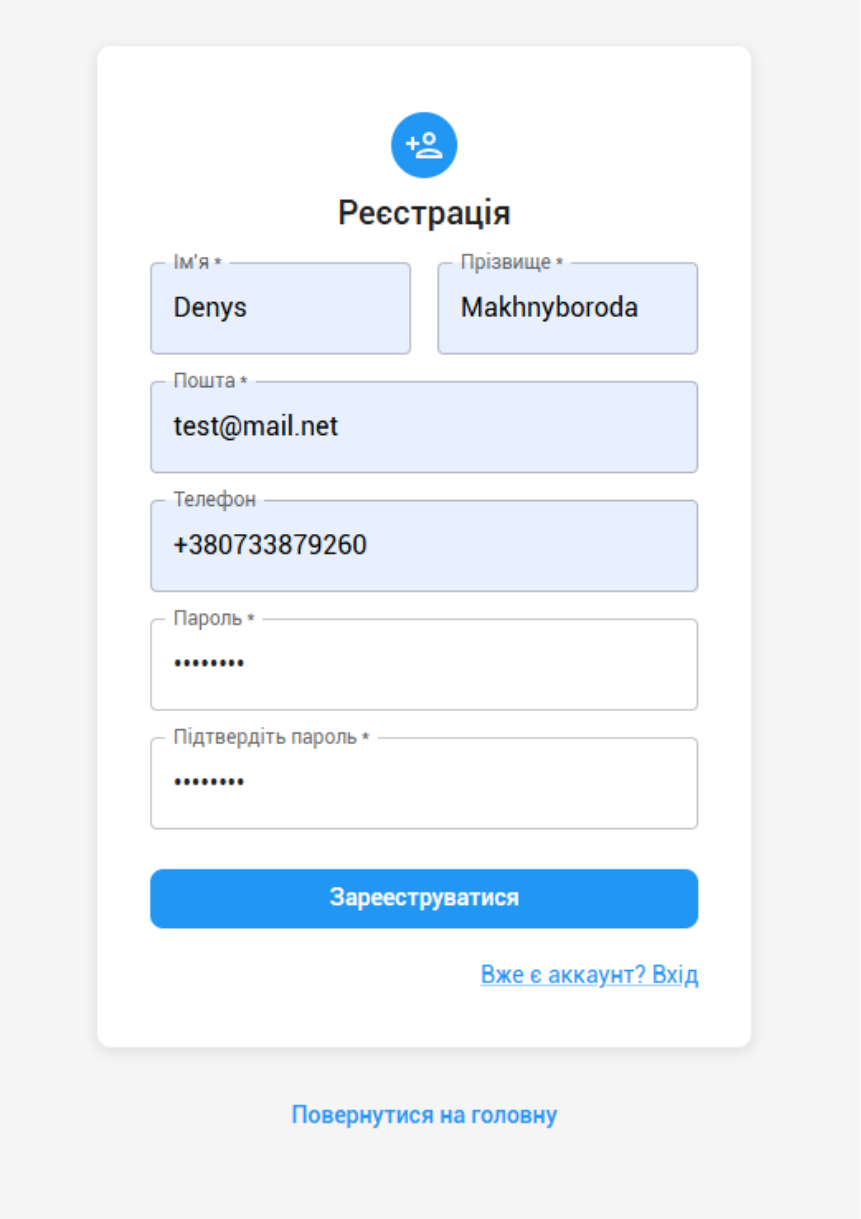

Рис. В.1 - Тестова реєстрація користувача.

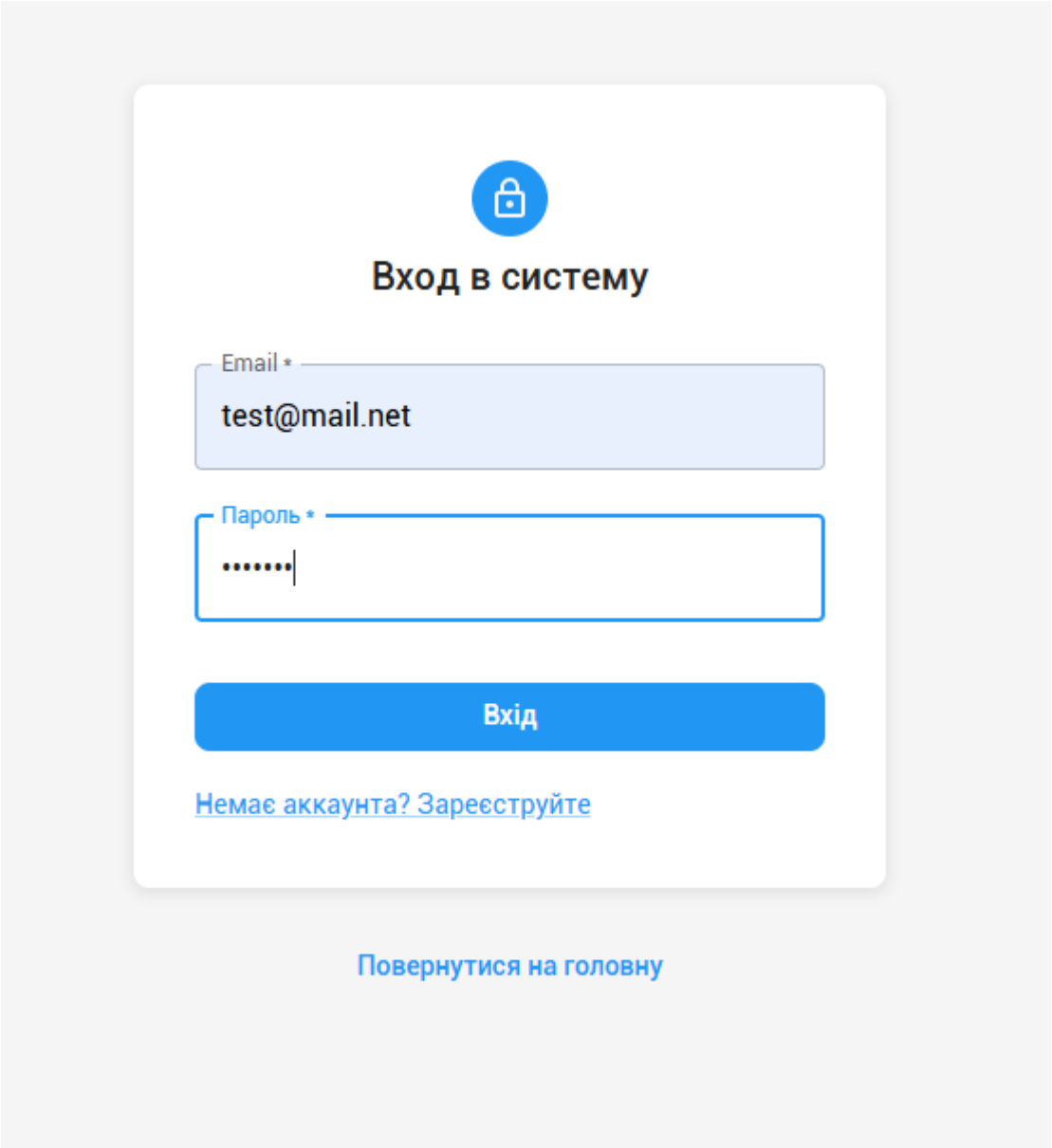

Рис. В.2 - Тестова авторизація користувача.

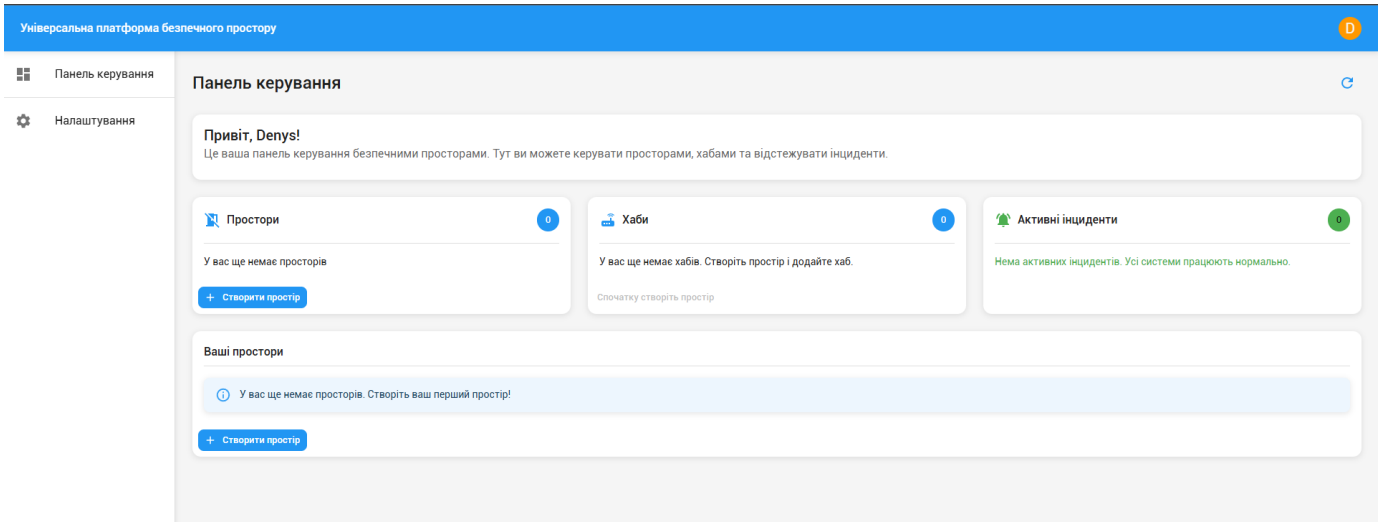

Рис. В.3 - Успішна авторизація.

Перевірка створення та налаштування простору:

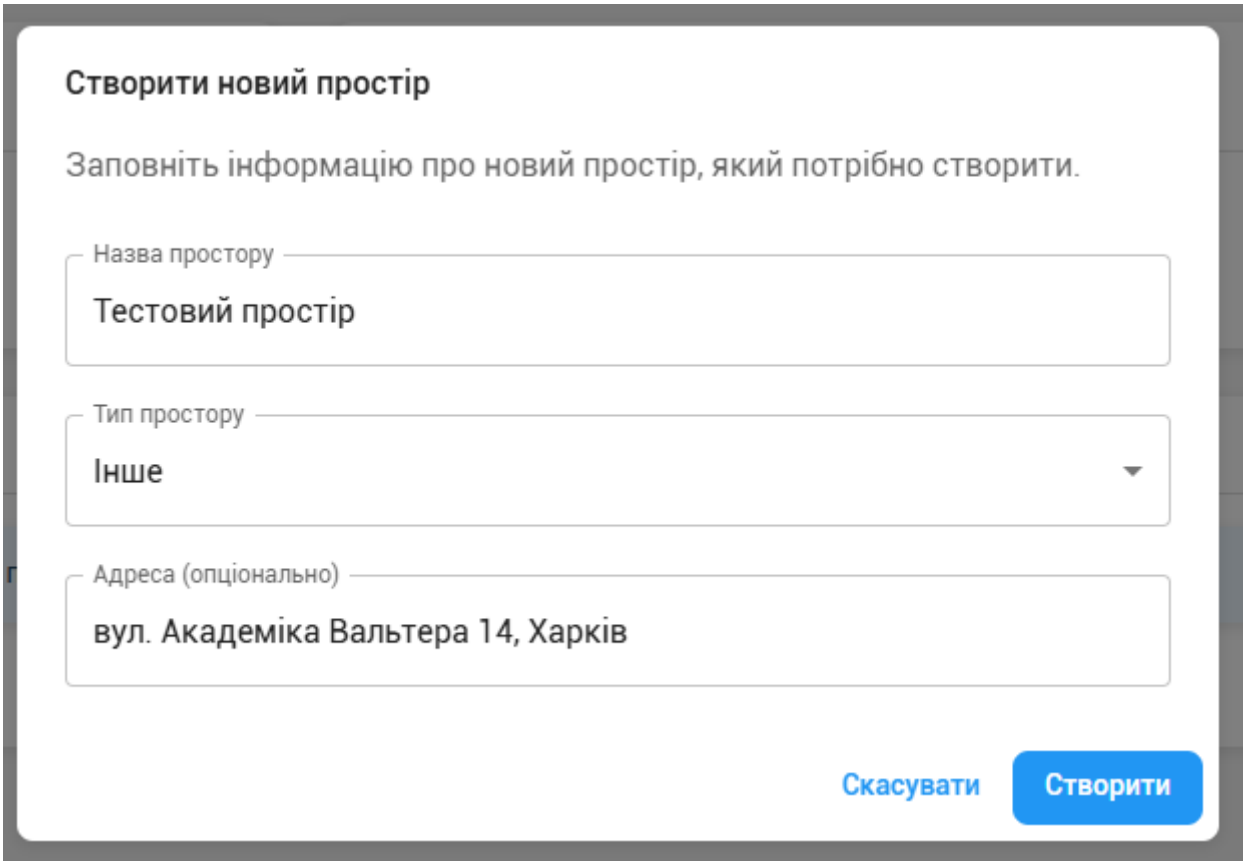

Рис. В.4 - Створення простору.

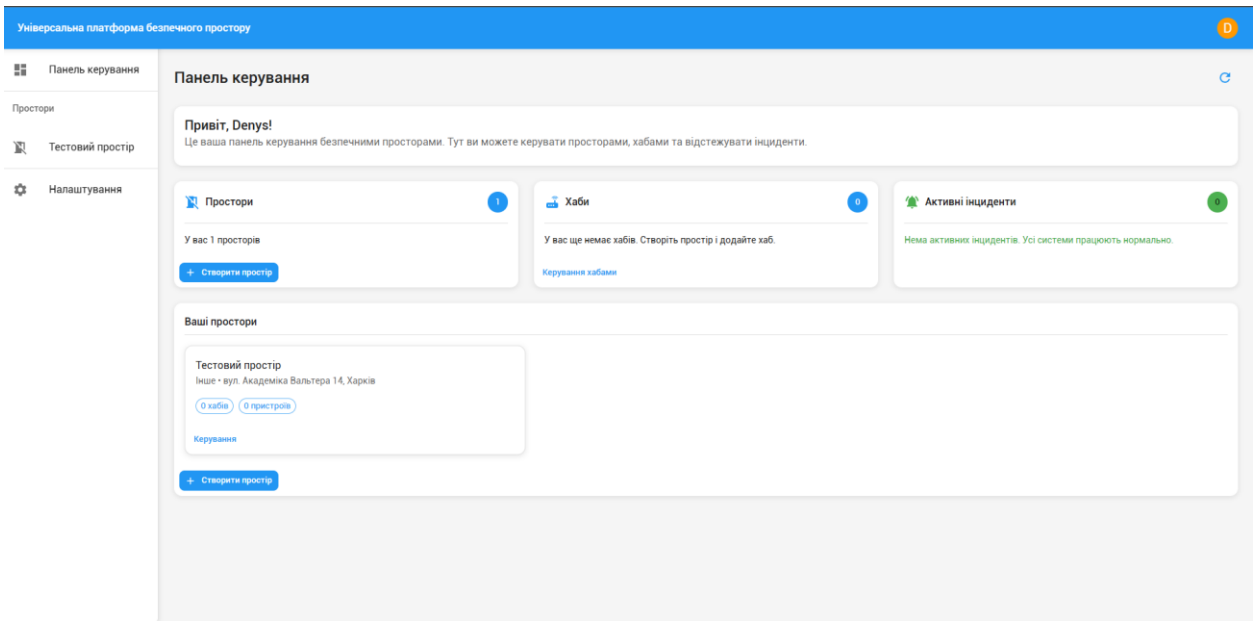

Рис. В.5 - Відображення створеного простору.

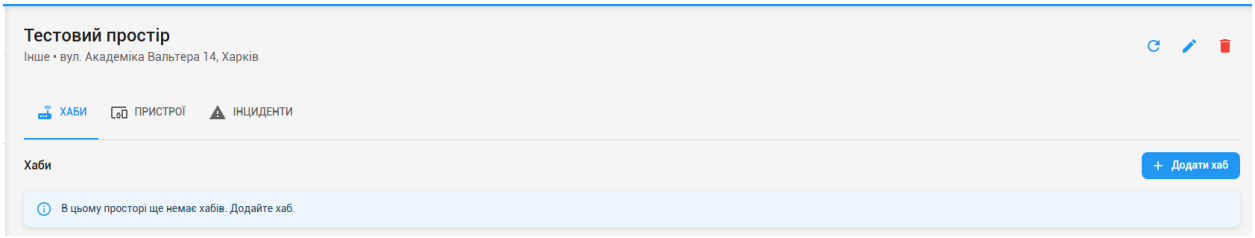

Рис. В.6 - Налаштування тестового простору.

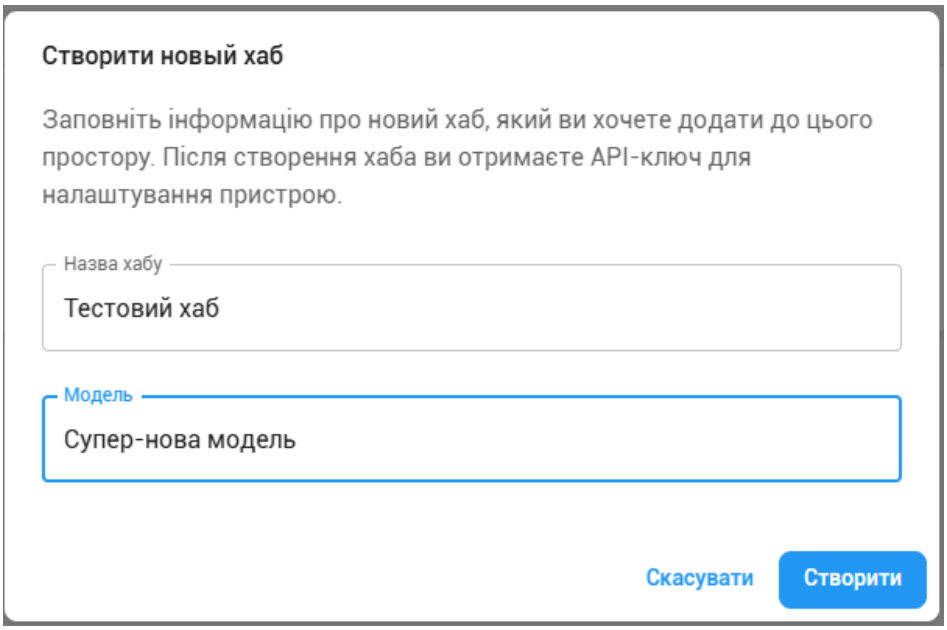

Рис. В.7 - Створення нового хабу.

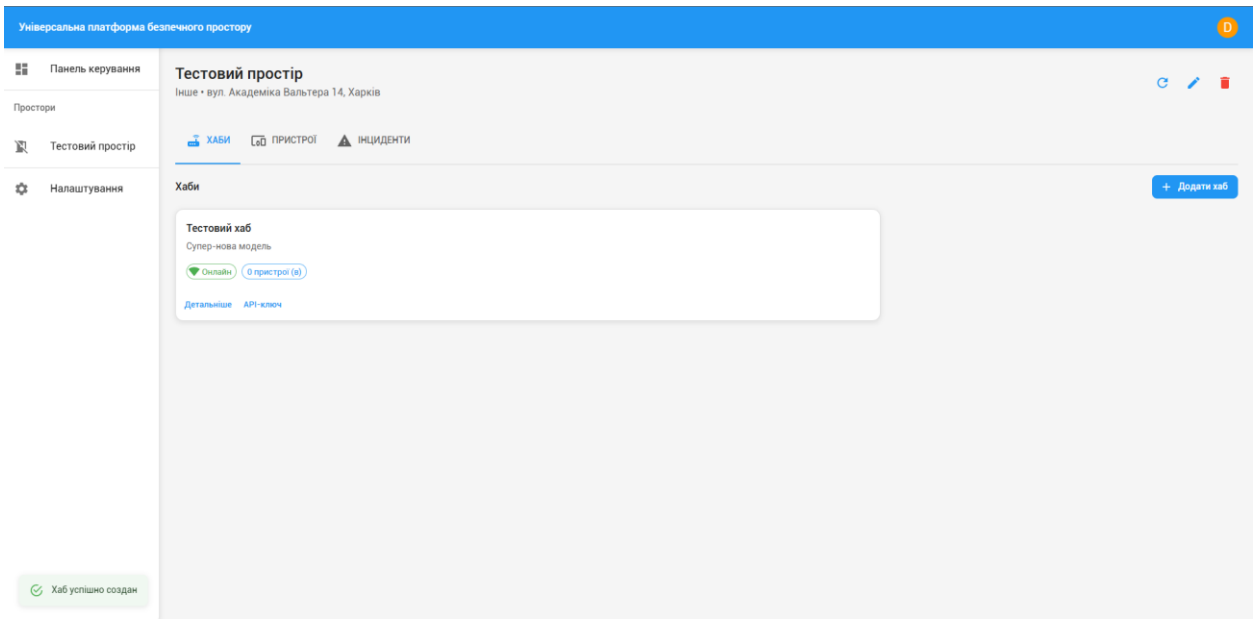

Рис. В.8 - Перегляд тестового простору з новим хабом.

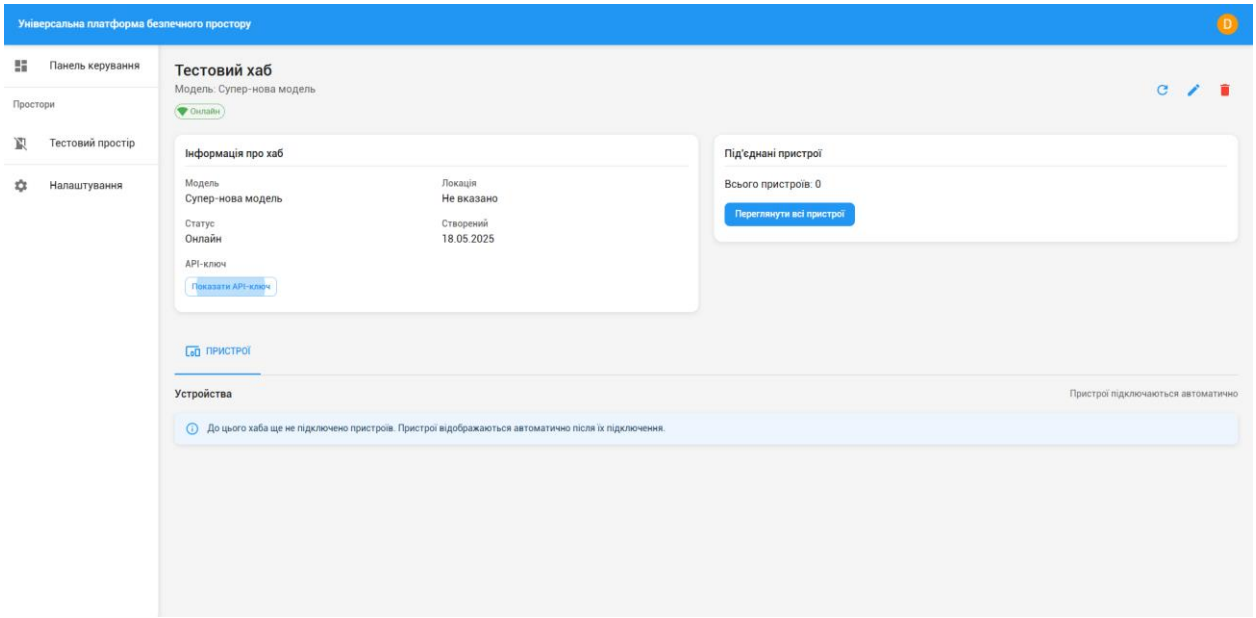

Рис. В.9 - Перегляд тестового хабу.

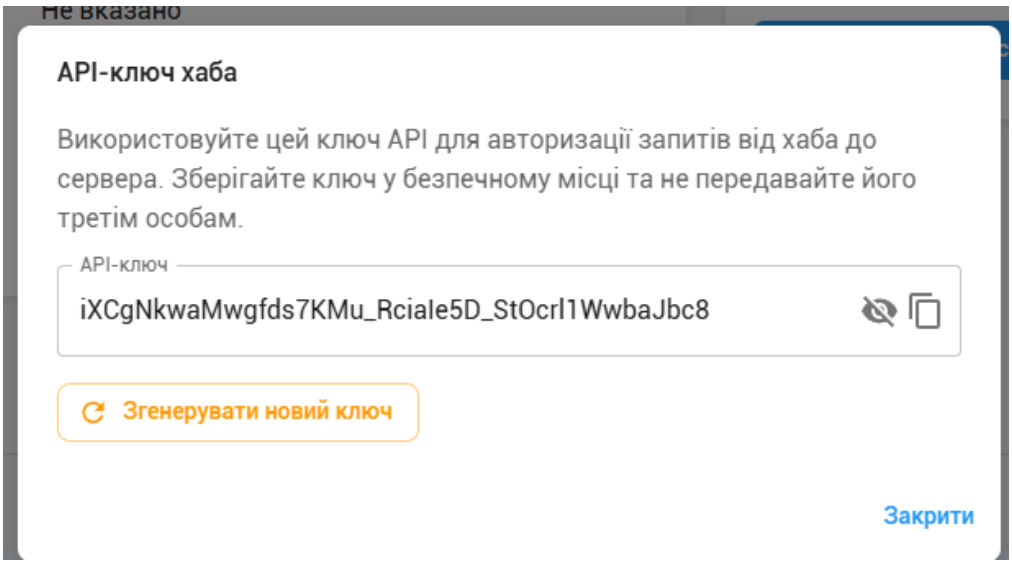

Рисунок В.10 - Створення нового АПІ-ключа.

```
API_URL = 'http://localhost:8000/api'
API_KEY = 'iXCgNkwaMwgfds7KMu_RciaIe5D_StOcrl1WwbaJbc8'
MQTT_BROKER = '127.0.0.1'
MQTT_PORT = 1883
MQTT_TOPIC = 'zigbee2mqtt'
MQTT_LOGIN = 'admin@mail.com'
MQTT_PASSWORD = '123456'
```

Рисунок В.11 - Налаштування АПІ-ключа в програмі-посередника.

Тепер можемо запустити програму-посередника

```
2025-05-18 15:44:53,479 [INFO] Connecting with username: admin@mail.com
2025-05-18 15:44:53,479 [INFO] Starting Safe Space Hub
2025-05-18 15:44:53,479 [INFO] API URL: http://localhost:8000/api
2025-05-18 15:44:53,479 [INFO] MQTT Broker: 127.0.0.1:1883
2025-05-18 15:44:53,479 [INFO] MQTT Topic: zigbee2mqtt
2025-05-18 15:44:53,510 [INFO] Connecting to MQTT broker 127.0.0.1:1883
2025-05-18 15:44:53,511 [INFO] Connected to MQTT broker with result code 0
2025-05-18 15:44:53,512 [INFO] Subscribed to bridge devices topic: zigbee2mqtt/bridge/devices
2025-05-18 15:44:53,512 [INFO] Subscribed to all device topics: zigbee2mqtt/+
2025-05-18 15:44:53,512 [INFO] Requested device list from zigbee2mqtt: zigbee2mqtt/bridge/request/devices
2025-05-18 15:44:53,513 [INFO] Received information about 3 devices from zigbee2mqtt
2025-05-18 15:44:53,513 [INFO] Device information processed successfully
2025-05-18 15:44:53,513 [INFO] Registering 3 devices in the platform API
2025-05-18 15:44:53,560 [INFO] Successfully registered device: Coordinator
2025-05-18 15:44:53,566 [INFO] Successfully registered device: 0xa4c138a5deb17291
2025-05-18 15:44:53,572 [INFO] Successfully registered device: 0xa4c138d3ce22c7d3
```

Рисунок В.12 - Логи програми-посередника.

На них ми можемо побачити що зареєструвалися 3 девайси, тепер подивимося на них в системі

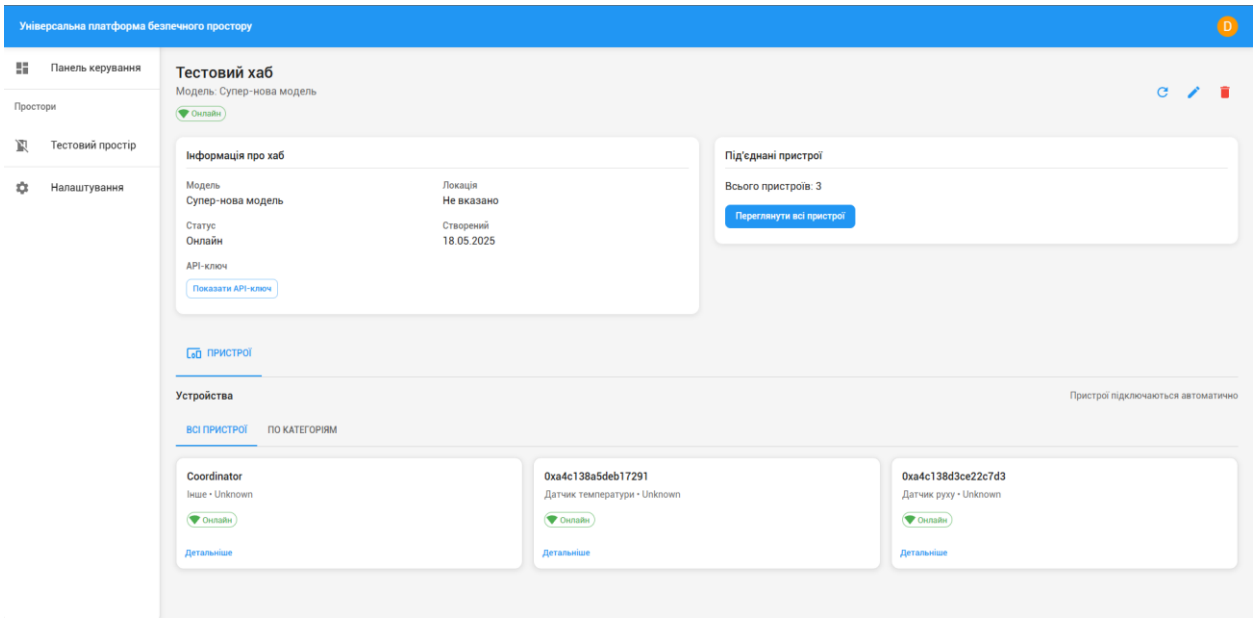

Рисунок В.13 - Вигляд хабу з девайсами.

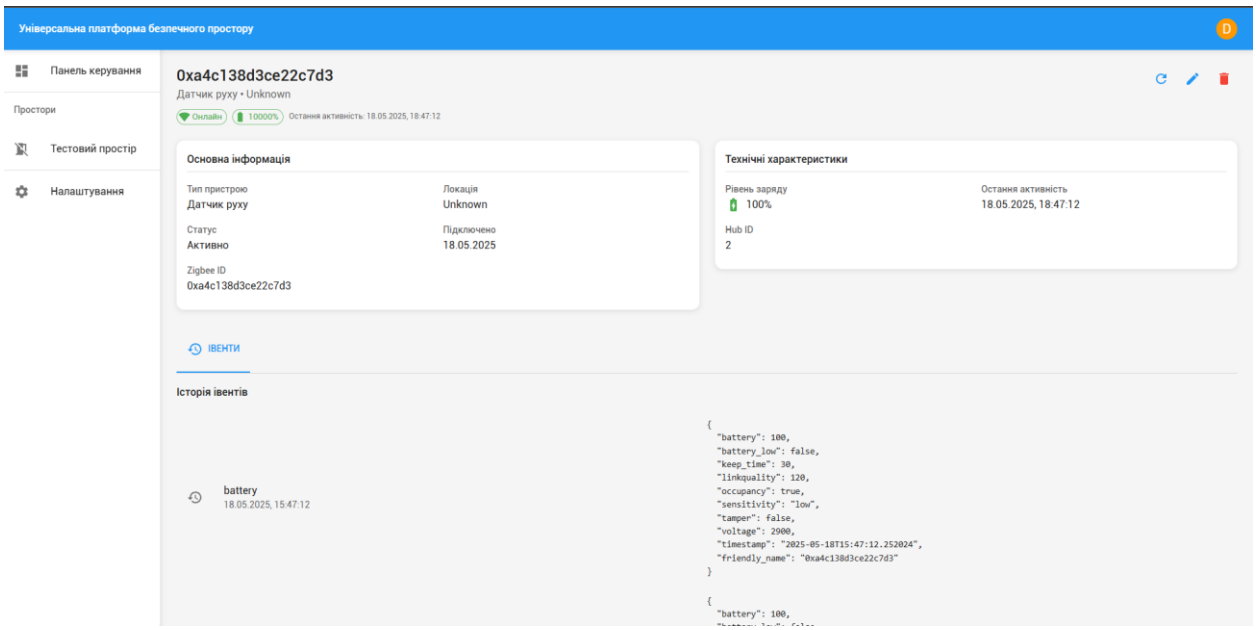

Рисунок В.14 - Перегляд девайсу з івентами.

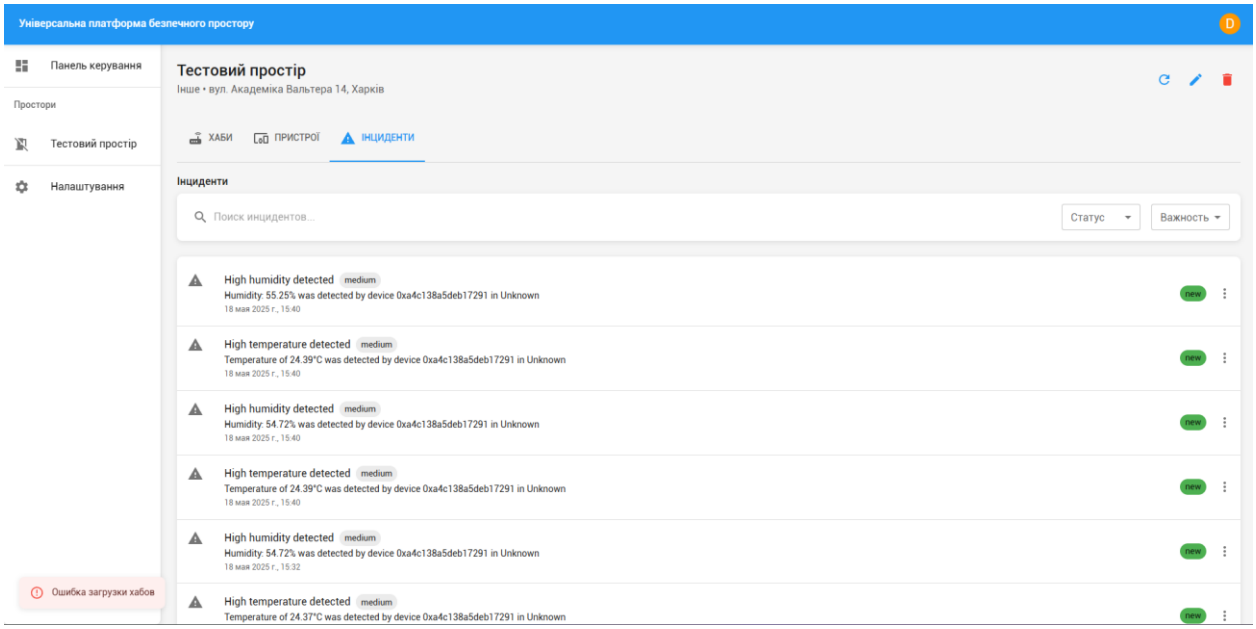

Рисунок В.15 - Перегляд інцидентів.

Перевірка API інтерфейсу:

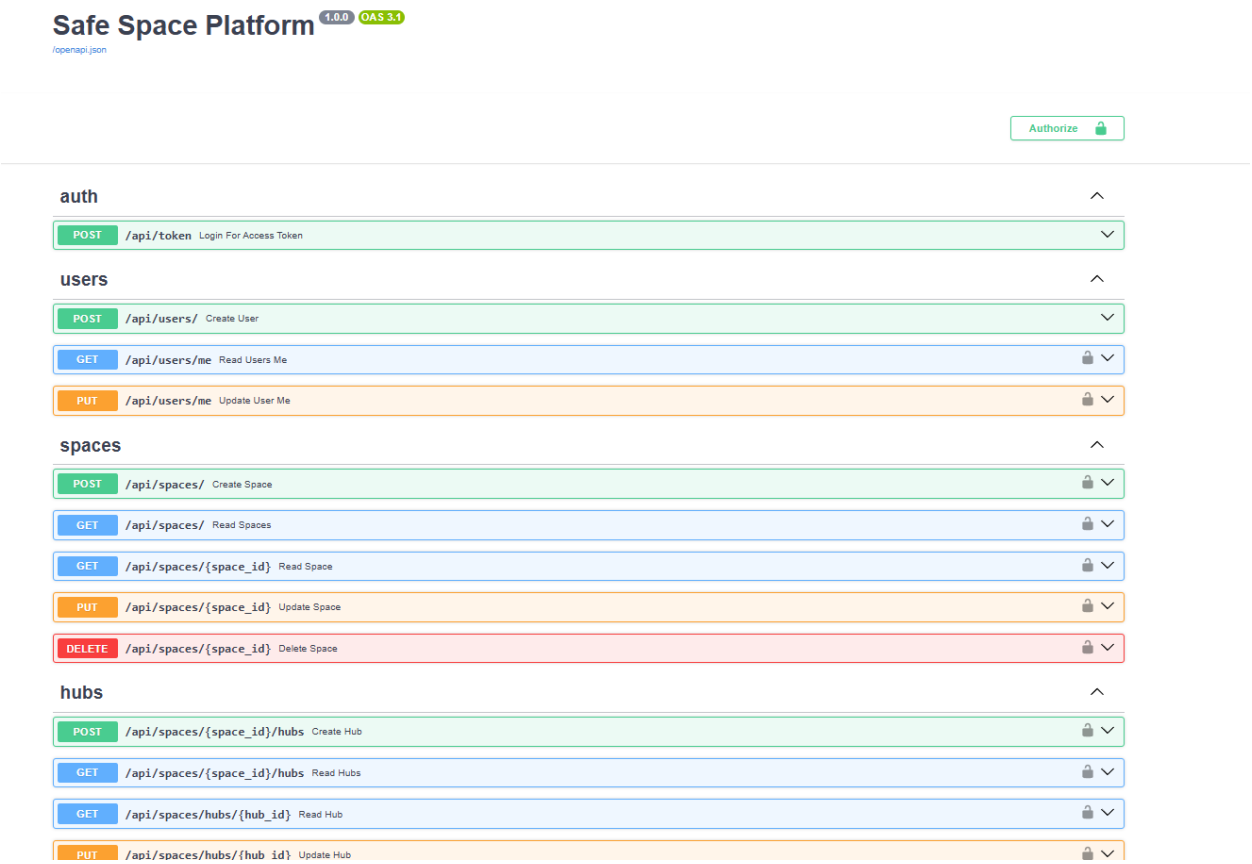

Рисунок В.16 - АПІ-Інтерфейс.

Висновки: тестування продукту пройшло успішно, все відпрацювало стабільно.

Виконавець: студент групи КІ-41, Махниборода Д.В. _____

Додаток Г

СТРУКТУРА БАЗИ ДАНИХ ПЛАТФОРМИ БЕЗПЕЧНОГО ПРОСТОРУ

Таблиця Users:

User			
int	id	PK	
string	email		unique
string	hashed_password		
string	first_name		
string	last_name		
string	phone		
bool	is_active		
datetime	created_at		
datetime	updated_at		

Рисунок Г.1 – Структура бази даних користувачів.

Таблиця Spaces:

Space			
int	id	PK	
string	name		
enum	type		home, apartment, office, warehouse, other
string	address		
int	owner_id	FK	
datetime	created_at		
datetime	updated_at		

Рисунок Г.2 – Структура бази даних просторів.

Таблиця Hubs:

Hub			
int	id	PK	
string	name		
string	model		
string	api_key		unique
bool	is_active		
datetime	last_connection		
string	ip_address		
int	space_id	FK	
datetime	created_at		
datetime	updated_at		

Рисунок Г.3 – Структура бази даних хабів.

Таблиця Devices:

Device			
int	id	PK	
string	name		
enum	type		motion_sensor, door_sensor, etc
string	zigbee_id		unique
string	location		
bool	is_active		
float	battery_level		
json	config		
datetime	last_seen		
int	hub_id	FK	
int	space_id	FK	
datetime	created_at		
datetime	updated_at		

Рисунок Г.4 – Структура бази даних девайсів.

Таблиця Events:

Event			
int	id	PK	
enum	type		motion_detected, door_opened, smoke_detected, etc
json	data		
bool	processed		
int	device_id	FK	
datetime	created_at		

Рисунок Г.5 – Структура бази даних подій.

Таблиця Incidents:

Incident			
int	id	PK	
string	title		
string	description		
enum	status		new, acknowledged, resolved, false_alarm
enum	severity		low, medium, high, critical
json	data		
int	event_id	FK	
datetime	created_at		
datetime	updated_at		
datetime	resolved_at		

Рисунок Г.6 – Структура бази даних інцидентів.

Додаток Д

Лістінг основного коду (програми посередника)

```
import json
import time
import requests
import paho.mqtt.client as mqtt
from datetime import datetime

from config import EVENT_TYPES, DEVICE_TYPES


class SafeSpaceHub:
    def __init__(
            self,
            logger,
            api_url,
            api_key,
            mqtt_broker,
            mqtt_port,
            mqtt_prefix="zigbee2mqtt",
            mqtt_username=None,
            mqtt_password=None
    ):

        self.logger = logger

        self.api_url = api_url
        self.api_key = api_key
        self.mqtt_broker = mqtt_broker
        self.mqtt_port = mqtt_port
        self.mqtt_prefix = mqtt_prefix
        self.mqtt_username = mqtt_username
        self.mqtt_password = mqtt_password

        self.devices_cache = {}

        self.zigbee_devices = {}

        self.devices_info_received = False

        client_id = f"zigbee2mqtt_bridge_{int(time.time())}"
        self.mqtt_client = mqtt.Client(client_id=client_id)
        self.mqtt_client.on_connect = self.on_connect
        self.mqtt_client.on_message = self.on_message
        self.mqtt_client.on_disconnect = self.on_disconnect

        if mqtt_username and mqtt_password:
            self.logger.info(f"Connecting with username: {mqtt_username}")
            self.mqtt_client.username_pw_set(mqtt_username, mqtt_password)

        self.session = requests.Session()
        self.session.headers = {"X-API-Key": self.api_key}

        self.connected = False

    def start(self):
        try:
            self.ping_api()

            self.logger.info(f"Connecting to MQTT broker {self.mqtt_broker}:{self.mqtt_port}")
            self.mqtt_client.connect(self.mqtt_broker, self.mqtt_port, 60)

            self.mqtt_client.loop_start()

            try:
                while True:
                    self.ping_api()
                    time.sleep(60)
            except KeyboardInterrupt:
                self.logger.info("Shutting down...")
            finally:
                self.mqtt_client.loop_stop()
                self.mqtt_client.disconnect()
```

```
        except Exception as e:
            self.logger.exception('Failed: ')
            self.logger.error(f"Error starting hub: {e}")
            raise

    def ping_api(self):
        try:
            response = self.session.post(
                f"{self.api_url}/spaces/hub/ping"
            )
            if response.status_code == 200:
                self.logger.debug("Successfully pinged API")
                self.connected = True
            else:
                self.logger.warning(f"Failed to ping API: {response.status_code} {response.text}")
                self.connected = False
        except Exception as e:
            self.logger.error(f"Error pinging API: {e}")
            self.connected = False

    def on_connect(self, client, userdata, flags, rc):
        if rc == 0:
            self.logger.info(f"Connected to MQTT broker with result code {rc}")

            bridge_topic = f"{self.mqtt_prefix}/bridge/devices"
            client.subscribe(bridge_topic)
            self.logger.info(f"Subscribed to bridge devices topic: {bridge_topic}")

            devices_topic = f"{self.mqtt_prefix}/+"
            client.subscribe(devices_topic)
            self.logger.info(f"Subscribed to all device topics: {devices_topic}")

            info_topic = f"{self.mqtt_prefix}/bridge/request/devices"
            client.publish(info_topic, "")
            self.logger.info(f"Requested device list from zigbee2mqtt: {info_topic}")

            client.subscribe(f"{self.mqtt_prefix}/bridge/state")
        else:
            self.logger.error(f"Failed to connect to MQTT broker with result code {rc}")

    def on_disconnect(self, client, userdata, rc):
        if rc != 0:
            self.logger.warning(f"Unexpected disconnection from MQTT broker: {rc}")
        else:
            self.logger.info("Disconnected from MQTT broker")

    def on_message(self, client, userdata, msg):
        try:
            topic = msg.topic

            try:
                payload = msg.payload.decode('utf-8')
                try:
                    payload_data = json.loads(payload)
                except json.JSONDecodeError:
                    payload_data = {}
            except UnicodeDecodeError:
                self.logger.warning(f"Failed to decode message payload: {msg.payload}")
                payload_data = {}

            self.logger.debug(f"Received message from topic {topic}: {payload}")

            if topic == f"{self.mqtt_prefix}/bridge/devices":
                self.handle_bridge_devices(payload)
            elif topic == f"{self.mqtt_prefix}/bridge/state":
                self.handle_bridge_state(payload)
            elif topic.startswith(f"{self.mqtt_prefix}/"):
                self.handle_device_message(topic, payload_data)

        except Exception as e:
            self.logger.error(f"Error processing message: {e}")

    def handle_bridge_devices(self, payload):
        try:
            devices = json.loads(payload)
            self.logger.info(f"Received information about {len(devices)} devices from zigbee2mqtt")
```

```
            for device in devices:
                ieee_address = device.get('ieee_address')
                friendly_name = device.get('friendly_name')

                if ieee_address and friendly_name:
                    self.zigbee_devices[friendly_name] = {
                        'ieee_address': ieee_address,
                        'definition': device.get('definition', {}),
                        'capabilities': device.get('supported', []),
                        'type': device.get('type', 'unknown'),
                        'model': device.get('model_id', 'unknown'),
                        'vendor': device.get('vendor', 'unknown'),
                        'description': device.get('description', 'unknown'),
                        'endpoints': device.get('endpoints', {})
                    }

            self.devices_info_received = True
            self.logger.info("Device information processed successfully")

            # Регистрирует устройства в API
            self.register_zigbee_devices()

        except json.JSONDecodeError:
            self.logger.error(f"Failed to parse bridge devices payload: {payload}")
        except Exception as e:
            self.logger.error(f"Error handling bridge devices: {e}")

    def handle_bridge_state(self, payload):
        try:
            state = payload.strip().lower()
            if state == "online":
                self.logger.info("zigbee2mqtt bridge is online")
            elif state == "offline":
                self.logger.warning("zigbee2mqtt bridge is offline")

        except Exception as e:
            self.logger.error(f"Error handling bridge state: {e}")

    def handle_device_message(self, topic, data):
        try:
            parts = topic.split('/')
            if len(parts) < 2:
                self.logger.warning(f"Invalid topic format: {topic}")
                return

            friendly_name = parts[1]

            device_info = self.zigbee_devices.get(friendly_name)
            if not device_info:
                self.logger.warning(f"Received message from unknown device: {friendly_name}")

                info_topic = f"{self.mqtt_prefix}/bridge/request/devices"
                self.mqtt_client.publish(info_topic, "")
                return

            self.process_device_events(friendly_name, device_info, data)

        except Exception as e:
            self.logger.error(f"Error handling device message: {e}")

    def register_zigbee_devices(self):
        try:
            self.logger.info(f"Registering {len(self.zigbee_devices)} devices in the platform API")

            for friendly_name, device_info in self.zigbee_devices.items():
                zigbee_id = device_info.get('ieee_address')
                if not zigbee_id:
                    self.logger.warning(f"Device {friendly_name} has no IEEE address, skipping
registration")
                    continue

                device_type = self.determine_device_type(device_info)

                device_data = {
                    "name": friendly_name,
                    "type": device_type,
```

```
                    "zigbee_id": zigbee_id,
                    "location": "Unknown",
                    "config": {
                        "model": device_info.get('model', 'unknown'),
                        "vendor": device_info.get('vendor', 'unknown'),
                        "description": device_info.get('description', 'unknown'),
                        "capabilities": device_info.get('capabilities', [])
                    }
                }

                try:
                    response = self.session.post(
                        f"{self.api_url}/spaces/hub/devices",
                        json=device_data
                    )
                    print(response.json())
                    if response.status_code == 201 or response.status_code == 200:
                        self.logger.info(f"Successfully registered device: {friendly_name}")
                        self.devices_cache[zigbee_id] = response.json()
                    else:
                        self.logger.warning(
                            f"Failed to register device {friendly_name}: {response.status_code}
{response.text}")

                except Exception as e:
                    self.logger.error(f"Error registering device {friendly_name}: {e}")

        except Exception as e:
            self.logger.error(f"Error registering zigbee devices: {e}")

    def determine_device_type(self, device_info):
        device_definition = device_info.get('definition', {})
        exposes = device_definition.get('exposes', [])

        device_type_map = {
            'occupancy': 'motion_sensor',
            'contact': 'door_sensor',
            'temperature': 'temperature_sensor',
            'humidity': 'humidity_sensor',
            'illuminance': 'air_quality_sensor',
            'water_leak': 'water_leak_sensor',
            'smoke': 'smoke_detector',
            'vibration': 'window_sensor'
        }

        for feature in exposes:
            feature_type = feature.get('type', '')
            feature_name = feature.get('name', '')

            if feature_type == 'binary' or feature_type == 'numeric':
                if feature_name in device_type_map:
                    return device_type_map[feature_name]

        model = device_info.get('model', '').lower()
        if 'motion' in model or 'occupancy' in model:
            return 'motion_sensor'
        elif 'door' in model or 'contact' in model or 'window' in model:
            return 'door_sensor'
        elif 'temp' in model:
            return 'temperature_sensor'
        elif 'humid' in model:
            return 'humidity_sensor'
        elif 'water' in model or 'leak' in model:
            return 'water_leak_sensor'
        elif 'smoke' in model or 'fire' in model:
            return 'smoke_detector'

        return 'other'

    def process_device_events(self, friendly_name, device_info, data):
        zigbee_id = device_info.get('ieee_address')
        if not zigbee_id:
            self.logger.warning(f"Device {friendly_name} has no IEEE address, cannot process events")
            return

        data_with_timestamp = dict(data)
        data_with_timestamp["timestamp"] = datetime.now().isoformat()
```

```
        data_with_timestamp["friendly_name"] = friendly_name

        if 'occupancy' in data and data['occupancy']:
            self.send_event(zigbee_id, "motion_detected", data_with_timestamp)

        if 'contact' in data and not data['contact']:
            self.send_event(zigbee_id, "door_opened", data_with_timestamp)

        if 'smoke' in data and data['smoke']:
            self.send_event(zigbee_id, "smoke_detected", data_with_timestamp)

        if 'water_leak' in data and data['water_leak']:
            self.send_event(zigbee_id, "water_leak_detected", data_with_timestamp)

        if 'temperature' in data:
            temperature = data['temperature']
            data_with_timestamp["temperature"] = temperature
            self.send_event(zigbee_id, "temperature", data_with_timestamp)

        if 'humidity' in data:
            humidity = data['humidity']
            data_with_timestamp["humidity"] = humidity
            self.send_event(zigbee_id, "humidity", data_with_timestamp)

        if 'battery' in data:
            battery = data['battery']
            data_with_timestamp["battery"] = battery
            self.send_event(zigbee_id, "battery", data_with_timestamp)

    def process_message(self, zigbee_id, event_type, data):
        try:
            device_info = self.get_device_info(zigbee_id)

            if not device_info:
                device_info = self.register_device(zigbee_id, event_type, data)
                if not device_info:
                    self.logger.warning(f"Failed to register device with zigbee_id: {zigbee_id}")
                    return

            api_event_type = self.map_event_type(event_type)
            self.send_event(zigbee_id, api_event_type, data)

        except Exception as e:
            self.logger.error(f"Error processing message for device {zigbee_id}: {e}")

    def get_device_info(self, zigbee_id):
        if zigbee_id in self.devices_cache:
            return self.devices_cache[zigbee_id]

        for friendly_name, device_info in self.zigbee_devices.items():
            if device_info.get('ieee_address') == zigbee_id:
                return device_info

        return None

    def register_device(self, zigbee_id, event_type, data):
        try:
            device_type = self.infer_device_type(event_type, data)

            device_data = {
                "name": f"Device {zigbee_id}",
                "type": device_type,
                "zigbee_id": zigbee_id,
                "location": data.get("location", "Unknown location")
            }

            response = self.session.post(
                f"{self.api_url}/spaces/hub/devices",
                json=device_data
            )

            if response.status_code == 201 or response.status_code == 200:
                device_info = response.json()
                self.devices_cache[zigbee_id] = device_info
                self.logger.info(f"Successfully registered device: {zigbee_id}")
                return device_info
            else:
```

```
                self.logger.warning(f"Failed to register device: {response.status_code} {response.text}")
                return None

        except Exception as e:
            self.logger.error(f"Error registering device {zigbee_id}: {e}")
            return None

    def infer_device_type(self, event_type, data):
        event_to_device = {
            "motion": "motion_sensor",
            "door_open": "door_sensor",
            "window_open": "window_sensor",
            "smoke": "smoke_detector",
            "water_leak": "water_leak_sensor",
            "high_temp": "temperature_sensor",
            "low_temp": "temperature_sensor",
            "high_humidity": "humidity_sensor",
            "low_humidity": "humidity_sensor",
            "poor_air": "air_quality_sensor"
        }

        if event_type in event_to_device:
            return event_to_device[event_type]

        if "device_type" in data:
            device_type = data["device_type"]
            if device_type in DEVICE_TYPES.values():
                return device_type

        return "other"

    def map_event_type(self, raw_event_type):
        if raw_event_type in EVENT_TYPES:
            return EVENT_TYPES[raw_event_type]

        # Проверяем частичное соответствие
        for key, value in EVENT_TYPES.items():
            if key in raw_event_type:
                return value

        return "other"

    def send_event(self, zigbee_id, event_type, data):
        try:
            event_data = {
                "zigbee_id": zigbee_id,
                "type": event_type,
                "data": data
            }

            response = self.session.post(
                f"{self.api_url}/spaces/hub/events",
                json=event_data
            )

            if response.status_code == 201 or response.status_code == 200:
                self.logger.info(f"Successfully sent event: {event_type} for device {zigbee_id}")
            else:
                self.logger.warning(f"Failed to send event: {response.status_code} {response.text}")

        except Exception as e:
            self.logger.error(f"Error sending event for device {zigbee_id}: {e}")
```

Детальний лістінг коду бекенду:

https://github.com/De9Max/safe_space_backend

Детальний лістінг коду фронтенду:

https://github.com/De9Max/safe_space_frontend