МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ВОДНОГО ГОСПОДАРСТВА ТА ПРИРОДОКОРИСТУВАННЯ
Навчально-науковий інститут кібернетики,
інформаційних технологій та інженерії
Кафедра комп'ютерних наук та прикладної математики

"До захисту допущена"
Завідувач кафедри комп'ютерних
наук та прикладної математики
Турбал Юрій Васильович
_____ *20 р.*

КВАЛІФІКАЦІЙНА РОБОТА

«*Проектування та розробка прототипу охоронної системи будинку з використанням Arduino та ESP32*»

Виконав: *Дяченко Богдан Володимирович*
_____
(прізвище, ім'я, по батькові) *(підпис)*
група ІПЗ-41
Керівник: *ст.викладач Зубик Я. Я.*
_____
(науковий ступінь, вчене звання, посада, прізвище, ініціали) *(підпис)*

Рівне 2025

Зміст

Реферат

Кваліфікаційна бакалаврська робота присвячена розробці та впровадженню прототипу охоронної системи будинку з використанням платформи ESP32, Arduino та мобільного застосунку на Android. Метою дослідження є створення доступного, функціонального та автономного рішення для забезпечення безпеки житла шляхом моніторингу ключових параметрів: відкриття дверей, наявності газу в повітрі та фіксації просторових змін за допомогою ультразвукового сенсора.

У роботі проаналізовано сучасні рішення на ринку систем безпеки, виконано порівняння апаратних платформ Arduino та ESP32, розроблено архітектуру охоронної системи з урахуванням її масштабованості та енергоефективності. Система реалізована на базі мікроконтролера ESP32, до якого підключено газовий, магнітний та ультразвуковий сенсори. Дані передаються на сервер, розгорнутий на платформі Render, з використанням Flask та бази даних PostgreSQL. Сервер зберігає та обробляє інформацію, забезпечує авторизацію користувачів та взаємодію з мобільним застосунком.

Мобільний додаток, створений на мові Kotlin, дозволяє користувачеві переглядати поточні дані з пристрою, додавати та редагувати пристрої, а також переглядати історію подій. У роботі детально описано функціонал усіх компонентів, з прикладами фрагментів коду, алгоритмами взаємодії, скріншотами інтерфейсу застосунку та схемами підключення.

Обсяг дипломної роботи становить 56 сторінок, вона містить 4 розділи, 10 рисунків, 4 таблиці , 2 додатки.

Охоронна система, ESP32, Arduino, мобільний додаток, Android, сенсори, Flask, сервер, база даних, інтернет речей (IoT), дистанційний моніторинг, PostgreSQL.

ВСТУП

У сучасному світі, де кількість приватних помешкань, дачних будинків та офісних приміщень постійно зростає, питання безпеки об'єктів нерухомості набуває все більшого значення. Особливої актуальності воно набуває у контексті зростання злочинності, несанкціонованих проникнень та пожежонебезпеки. У зв’язку з цим суттєво зростає попит на надійні, автономні та розумні системи охорони.

Завдяки стрімкому розвитку Інтернету речей (Internet of Things, IoT) з’явилася можливість будувати доступні й ефективні системи моніторингу та керування, які здатні працювати у реальному часі, взаємодіяти з користувачем через мобільні додатки та забезпечувати віддалене сповіщення про загрози. Комбінація мікроконтролерів Arduino та потужних Wi-Fi модулів ESP32 відкриває нові горизонти для створення інтелектуальних охоронних систем із можливістю швидкого розгортання, масштабованості та зниження вартості реалізації.

Мета і завдання дослідження

Метою моєї кваліфікаційної роботи є **проєктування та розробка прототипу охоронної системи для приватного будинку** на основі мікроконтролерів Arduino та ESP32 з використанням сенсорів, web-сервера, мобільного Android-застосунку та технологій бездротового зв’язку.

Для досягнення поставленої мети необхідно виконати такі завдання:

- провести аналіз існуючих рішень у галузі охоронних систем та IoT;
- дослідити технічні особливості Arduino, ESP32, а також відповідні протоколи передачі даних;
- розробити архітектуру системи з урахуванням потреб користувача;
- реалізувати програмне забезпечення для мікроконтролера ESP32, серверну частину на Flask та мобільний застосунок Android;

- забезпечити зв’язок між усіма компонентами системи з урахуванням вимог безпеки;
- протестувати систему в реальних умовах та надати оцінку її ефективності.

Об'єкт і предмет дослідження

Об’єктом дослідження є процес забезпечення фізичної безпеки житлових приміщень за допомогою технічних засобів контролю та оповіщення.

Предметом дослідження є розробка інтегрованої охоронної системи, побудованої на базі мікроконтролерної платформи ESP32, сенсорних пристроїв, Android-додатку та web-сервера, що забезпечує взаємодію користувача з пристроєм у режимі реального часу.

Методи дослідження

У процесі виконання роботи застосовувалися наступні методи:

- метод системного аналізу — для визначення вимог до функціоналу охоронної системи;
- порівняльний аналіз — при виборі апаратної та програмної платформи;
- моделювання та прототипування — при побудові схеми й реалізації системи;
- методи об’єктно-орієнтованого проєктування — для розробки структури застосунку;
- тестування та інструментальні вимірювання — для оцінки працездатності системи у реальних умовах.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ОСНОВИ СТВОРЕННЯ ІНТЕЛЕКТУАЛЬНИХ ОХОРОННИХ СИСТЕМ З ВИКОРИСТАННЯМ ARDUINO ТА ESP32

1.1. Інтернет речей як основа побудови сучасних охоронних систем

Інтернет речей (англ. Internet of Things, **IoT**) — це концепція з'єднання фізичних пристроїв із мережею Інтернет для обміну даними та автоматизованого керування. IoT дозволяє створювати адаптивні системи, які реагують на зовнішнє середовище в режимі реального часу. Ці технології стали надзвичайно популярними у сферах «розумного дому», енергетики, промислової автоматизації та, зокрема, охоронних систем.

Охоронні системи, засновані на технологіях IoT, мають такі ключові переваги:

- віддалений доступ — можливість контролювати стан об'єкта з будь-якої точки світу;
- автоматизація реакцій — миттєва активація сирен, надсилання сповіщень;
- гнучке налаштування — конфігурування параметрів через мобільні застосунки або веб-інтерфейси;
- інтеграція з іншими сервісами — наприклад, хмарні бази даних, Telegram-боти тощо.

Таким чином, IoT не просто збагачує охоронну інфраструктуру новими функціями, а трансформує саму модель взаємодії користувача з безпековими механізмами.

1.2. Аналіз ринку сучасних охоронних систем

Сфера охоронних систем є однією з найбільш динамічних і технічно розвинених галузей сучасного ринку електроніки. Зі зростанням урбанізації, підвищенням рівня життя населення та актуальністю питань безпеки

приватних об'єктів, попит на засоби контролю доступу, детекції вторгнень, відеоспостереження та оповіщення зростає щороку.

На сьогодні ринок охоронних систем умовно поділяється на два основних сегменти:

- рофесійні (промислові або корпоративні) охоронні комплекси, які вимагають монтажу сертифікованими фахівцями, інтеграції з централізованими пультами охорони, використовують спеціалізоване обладнання (GSM-модулі, LoRa-зв’язок, сигнали тривоги, датчики руху, відкриття, диму, витоку газу тощо);
- поживчі (домашні) охоронні системи — рішення для самостійної інсталяції, які часто підтримують підключення через Wi-Fi або Bluetooth та мають мобільний застосунок для керування й моніторингу.

До найвідоміших комерційних продуктів на українському та європейському ринку належать:

- jax Systems (Україна) — одна з провідних компаній у галузі, пропонує комплексні бездротові охоронні системи з високим рівнем шифрування, професійною пультовою охороною, мобільним застосунком та власною хмарною інфраструктурою;
- ing (США) — рішення компанії Amazon, орієнтовані на домашніх користувачів. Відомі завдяки відеодомофонам та камерам спостереження, що інтегруються з Alexa;
- iaomi Smart Home (Китай) — екосистема пристроїв для розумного дому, які включають датчики відкриття дверей, руху, камери, сигналізації тощо. Висока доступність і простота налаштування;

-

P-Link Tapo, EZVIZ, Hikvision — доступні варіанти для споживчого ринку, які підтримують мобільні застосунки, сповіщення про тривогу, відеозапис на хмару або локально.

Ціновий діапазон комерційних охоронних рішень коливається від 7000 грн за простий комплект до 20000 грн і вище — залежно від кількості пристроїв, підтримки GSM або Ethernet, можливостей інтеграції з хмарою або пультами охорони.

Попри очевидні переваги готових рішень, вони мають низку недоліків:

-

акрита архітектура, що унеможливлює модифікацію або інтеграцію з іншими пристроями;

-

исока вартість, особливо при масштабуванні на кілька об’єктів.

На цьому тлі дедалі популярнішими стають DIY-рішення (Do It Yourself), побудовані на відкритих платформах Arduino, ESP8266, ESP32, Raspberry Pi, що дозволяють створювати адаптивні, налаштовувані та недорогі охоронні системи з можливістю інтеграції з мобільними застосунками, власними API, локальними серверами.

Такий підхід дає змогу студентам, інженерам і розробникам:

-

либше зрозуміти принципи роботи охоронної системи;

-

еалізувати авторські функції (наприклад, затримку сирени, керування через Wi-Fi, вивід на LCD тощо);

-

е залежати від сторонніх сервісів;

-

аощадити кошти при розробці прототипів для реальних об’єктів.

Саме в контексті цього тренду і було вирішено розробити прототип охоронної системи будинку на базі мікроконтролера ESP32, з мобільним Android-застосунком та серверною частиною на **Flask**.

1.3. Платформи Arduino та ESP32: порівняльна характеристика

У процесі проєктування прототипу охоронної системи було розглянуто кілька варіантів апаратних платформ для реалізації основного логічного модуля. Найбільш відомими серед доступних відкритих рішень є Arduino та ESP32. Обидві платформи активно застосовуються в освітніх, дослідницьких і аматорських проєктах завдяки простоті використання, доступності документації та підтримці великої кількості сенсорів і периферійних пристроїв.

Для кращого розуміння технічних переваг і обмежень кожної з платформ далі наведено порівняльну таблицю:

Таблиця 1.1 – Порівняльна характеристика платформ Arduino Uno та ESP32

Характеристика	**Arduino Uno (ATmega328P)**	**ESP32 (ESP32-WROOM-32D)**
Архітектура	8-біт	32-біт (Tensilica Xtensa)
Тактова частота	16 МГц	до 240 МГц (двоядерний процесор)
Оперативна пам’ять (SRAM)	2 КБ	до 520 КБ
Постійна пам’ять (Flash)	32 КБ	4 МБ (або більше, залежно від модуля)
Вбудований Wi-Fi	Немає	Є (802.11 b/g/n)

Вбудований Bluetooth	Немає	Є (BLE + Classic)
Кількість GPIO-пінів	14 цифрових, 6 аналогових	до 34 GPIO
Живлення	5 В	3.3 В
Вартість	Низька (~5–8 $)	Середня (~5–10 $)
Споживання енергії	Більше (немає режимів сну)	Менше (є глибокий сон)
Придатність для IoT	Обмежена	Висока

З наведених у таблиці даних видно, що ESP32 суттєво перевершує Arduino Uno за ключовими характеристиками, що є критично важливими для побудови сучасної охоронної системи: зокрема, наявність вбудованого Wi-Fi-модуля, вища продуктивність, більший обсяг оперативної пам’яті та підтримка режимів енергозбереження. Arduino Uno може використовуватись для простих задач, однак для розробки повноцінної IoT-системи вона є недостатньо потужною.

У ході попередніх експериментів було також встановлено, що спроба використання Arduino Uno паралельно з ESP32 призводить до нестабільності роботи, пов’язаної з конфліктами живлення та UART-з’єднань. Окрім того, Arduino Uno не підтримує бездротовий зв’язок «з коробки», що вимагало б додаткових модулів і ускладнювало конструкцію системи. З огляду на ці недоліки було прийнято рішення повністю відмовитися від Arduino Uno на користь ESP32.

Таким чином, аналіз технічних параметрів та практичний досвід підтверджують доцільність використання ESP32 як основної платформи в рамках даного проєкту охоронної системи.

1.4. Протоколи зв’язку у IoT-середовищі

Однією з ключових особливостей сучасних систем Інтернету речей (IoT) є здатність пристроїв до безперервної взаємодії між собою та із зовнішніми службами — мобільними застосунками, серверами, базами даних, веб-інтерфейсами. Саме ця властивість реалізується завдяки **протоколам зв'язку**, які визначають спосіб обміну інформацією, її формат, правила маршрутизації та безпеки.

У межах цього проєкту були застосовані одразу кілька рівнів протоколів зв'язку, що дозволило забезпечити ефективну та надійну комунікацію між компонентами системи: **ESP32**, **Flask API-сервером**, **мобільним Android-застосунком**. Нижче наведено огляд ключових протоколів.

1.4.1. Вибір технології передачі даних

Усі пристрої в системі взаємодіють через **локальну Wi-Fi-мережу**. Саме Wi-Fi було обрано як базову транспортну технологію з наступних причин:

- ідтримується ESP32 без додаткових модулів;
- абезпечує достатню пропускну здатність для передачі JSON-повідомлень у реальному часі;
- озволяє швидко інтегруватися у вже наявну домашню мережу;
- ідтримує як режим **STA (Station)** — підключення як до роутера, так і **режим AP (Access Point)** — для первинного налаштування пристрою через Android.

У проєкті реалізовано обидва режими. Після конфігурації ESP32 автоматично перемикається з **AP** у **STA**-режим, підключаючись до збереженої Wi-Fi-мережі користувача.

1.4.2. HTTPS-зв'язок та централізоване зберігання даних

У архітектурі проєкту було реалізовано модель, в якій мікроконтролер ESP32 не підтримує безпосередній зв'язок із мобільним застосунком. Натомість, з метою підвищення надійності та масштабованості, було впроваджено схему з передачею даних на центральний сервер, що виконує роль проміжної ланки між пристроєм і користувачем.

Алгоритм взаємодії виглядає так:

-

ікроконтролер ESP32 періодично формує JSON-повідомлення, яке містить показники з сенсорів (gasLevel, distance, alarm тощо), додає до нього device_token для ідентифікації пристрою та надсилає HTTPS POST-запит на ендпоїнт /push_data **Flask**-сервера;

-

lask-сервер обробляє вхідний запит, зберігає payload у таблиці device_data бази PostgreSQL, а також фіксує час надходження даних (created_at);

-

ndroid-додаток, після авторизації користувача, надсилає HTTPS GET-запит до ендпоїнта /get_device_data з токеном авторизації та параметром device_token;

-

ервер перевіряє токен, переконується у праві користувача бачити цей пристрій та повертає останні 20 записів у вигляді масиву JSON-об'єктів.

Переваги запропонованої архітектури полягають у наступному:

-

ентралізоване зберігання — усі показники агрегуються на сервері, що спрощує подальший аналіз і побудову графіків;

- езпечний доступ — кожен запит перевіряється за допомогою JWT, що унеможливлює несанкціонований доступ;
- одульність — пристрої можуть передавати дані незалежно від наявності підключеного мобільного застосунку;
- озширюваність — легко додати веб-інтерфейс або аналітичний модуль, не змінюючи логіку ESP32 чи Android.

Така модель відповідає сучасним принципам побудови IoT-систем і може бути адаптована для промислового використання (наприклад, для моніторингу параметрів середовища, споживання енергії тощо).

1.4.3. Аутентифікація за допомогою JWT

З розвитком інформаційних технологій та стрімким поширенням Інтернету речей (IoT) питання забезпечення інформаційної безпеки набуває дедалі більшої актуальності. В умовах, коли тисячі пристроїв взаємодіють у мережі, надсилаючи та приймаючи чутливі дані, необхідно впроваджувати механізми, які забезпечують автентичність користувача, контроль доступу до ресурсів та захист від несанкціонованих втручань. Одним із таких сучасних механізмів, який активно використовується в IoT-рішеннях, є технологія **JSON Web Token (JWT)**.

JSON Web Token — це відкритий стандарт (RFC 7519), що визначає компактний і самодостатній спосіб передачі інформації між сторонами у вигляді JSON-об'єкта. Ця інформація може бути перевірена і довірена завдяки цифровому підпису. В контексті даного проєкту JWT використовується для забезпечення автентифікації користувача в мобільному застосунку, що взаємодіє з сервером.

Механізм роботи токена полягає в наступному. Коли користувач проходить процедуру авторизації (вводить логін і пароль), серверна частина на **Flask** перевіряє правильність введених даних. У разі успішної автентифікації, сервер генерує унікальний токен, до якого вбудовується певний набір інформації: унікальний ідентифікатор користувача, мітка часу створення, термін дії токена, а також інші параметри, які можна налаштувати на етапі генерації.

Отриманий токен має тричастинну структуру: header, payload і signature. Header вказує, який алгоритм використовувався для підпису (наприклад, HS256), payload містить фактичні дані, які передаються (sub, iat, exp), а signature є криптографічним підписом, який дозволяє перевірити цілісність токена.

Приклад:

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJhZG1pbiIsImV4cCI6MTcwNjE3Nzc4OX0.v89vB5IVTMyUv0FdWjPu_QHa5ZrFvZ1M3jVFGhN5mgo

Після отримання токена, мобільний застосунок зберігає його у захищеному сховищі (в даному випадку — Room або SharedPreferences) та надсилає до сервера у всіх наступних запитах у вигляді HTTPS-заголовка Authorization: Bearer <**jwttoken**>. Це дозволяє уникнути повторної автентифікації на кожному кроці, а також підтримує сесію між користувачем та сервером упродовж дії токена.

На стороні сервера усі запити, які вимагають авторизації, обгортаються у спеціальний декоратор **@token_required**, що виконує перевірку дійсності токена. У разі виявлення недійсного або простроченого токена, запит завершується з відповідним повідомленням про помилку 401 (Unauthorized), що відповідає стандартам безпечної розробки .

Перевагами використання JWT у цьому проєкті є:

- амодостатність токена — відсутність необхідності зберігати сесію на стороні сервера;
- ростота інтеграції — JSON-структура легко обробляється засобами Python і Kotlin;
- асштабованість — можливість реалізації системи з великою кількістю користувачів без збільшення навантаження на серверну частину;
- ідтримка сучасних стандартів безпеки — алгоритми шифрування (наприклад, HMAC-SHA256), можливість встановлення строку дії токена, тощо.

Таким чином, застосування JWT у реалізації мобільного клієнта охоронної системи є доцільним і сучасним рішенням, яке забезпечує достатній рівень захисту персональних даних користувача та дозволяє реалізувати доступ до пристроїв лише для авторизованих осіб. Це особливо важливо в умовах, коли система має справу із чутливою інформацією — наприклад, статусом безпеки помешкання, датою і часом спрацювання тривоги або показниками газового сенсора.

За допомогою даного підходу було досягнуто балансу між безпекою, продуктивністю та зручністю використання, що є критично важливим при розробці сучасних IoT-рішень.

1.5. База даних і структура збереження інформації

Для зберігання інформації про пристрої, користувачів і отримані дані використовується реляційна база PostgreSQL. Структура бази включає щонайменше три таблиці:

- **sers —** зберігає ім’я користувача (username) і захешований пароль (password) за допомогою bcrypt;
- **evices: —** містить список зареєстрованих пристроїв, їхні імена, IP-адреси, SSID, токени та прив'язку до користувача (user_id);
- **evice_data —** основна таблиця для зберігання показників, яку наповнює ESP32.

Розділ бази даних зображено на рисунку 1.1.

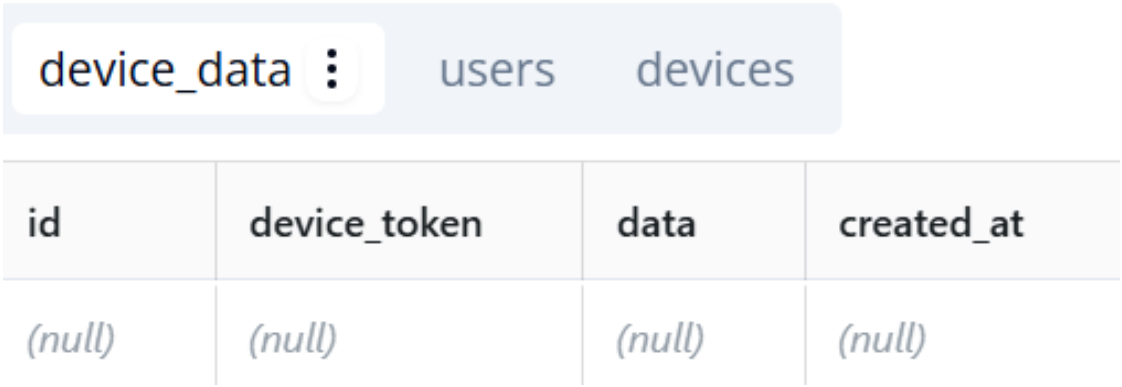

Рис. 1.1. Скріншот бази даних

Поле data містить повний JSON-об’єкт зі зчитаними значеннями сенсорів. Такий підхід (NoSQL у SQL) забезпечує гнучкість і можливість розширення структури даних без необхідності зміни схеми БД.

1.6. Сенсори та схема підключення

Газовий сенсор (Flying Fish)

Це аналоговий сенсор на основі модулів типу MQ-9, що реагує на концентрацію горючих газів. Вихідний сигнал — аналогова напруга, яку ESP32 читає через ADC.

Рис. 1.2. Газовий сенсор (Flying Fish)

Ультразвуковий сенсор (HC-SR04)

Призначений для вимірювання відстані до об’єктів. Робота базується на часі затримки ультразвукової хвилі:

- **RIG** — сигнал запуску імпульсу;
- **CHO** — сигнал відлуння, що повертається до датчика.

На рисунку 1.3 зображено ультразвуковий сенсор HC-SR04, який визначає відстань до об’єкта шляхом аналізу часу проходження імпульсу між виводами TRIG та ECHO.

Рис. 1.3. Ультразвуковий сенсор (HC-SR04)

Датчик відкриття дверей (геркон)

Працює як цифровий перемикач: якщо двері зачинені — ланцюг замкнутий, якщо відкриті — розімкнений. Читається як логічний HIGH або LOW.

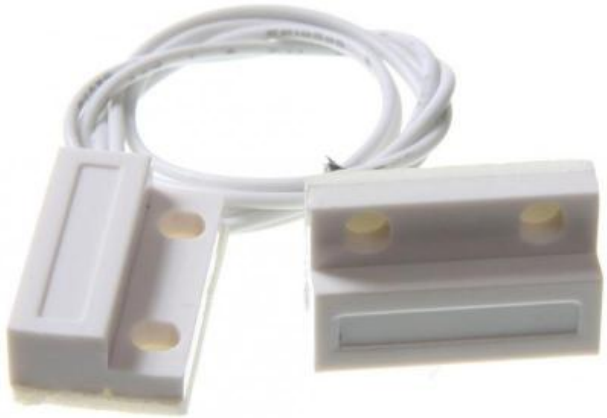

Рис. 1.4. Датчик відкриття дверей

РОЗДІЛ 2
ПРОЄКТУВАННЯ ПРОТОТИПУ ОХОРОННОЇ СИСТЕМИ НА ОСНОВІ ESP32

2.1. Загальна архітектура системи

Побудова прототипу охоронної системи була реалізована з використанням трирівневої архітектури, що включає апаратний рівень (ESP32 з підключеними сенсорами), серверну частину (**Flask** API, розгорнуту на Render), а також клієнтську частину у вигляді мобільного застосунку для Android.

На першому рівні — мікроконтролер ESP32, який виконує зчитування даних з сенсорів: газового датчика Flying Fish, ультразвукового датчика HC-SR04 для визначення відстані до об’єкта, а також магнітного геркона, що реагує на відкриття дверей. Отримані дані упаковуються у формат JSON і передаються через HTTPS POST-запит на серверну частину.

На другому рівні функціонує сервер, реалізований за допомогою мови Python та фреймворку **Flask**. Сервер виконує прийом даних, їх валідацію, обробку та збереження у базу даних PostgreSQL. Для зберігання структурованої сенсорної інформації використовується тип JSONB, що

дозволяє гнучко працювати з довільними структурами без створення окремих полів для кожного показника.

Третій рівень — мобільний застосунок, розроблений на платформі Android з використанням Kotlin. Застосунок надає користувачу можливість переглядати статус системи, історію спрацювань сенсорів, підключати нові пристрої, а також змінювати їхні налаштування (наприклад, Wi-Fi SSID, токен доступу тощо). Взаємодія додатку з сервером здійснюється через HTTPS-запити, передбачено авторизацію користувача через JWT.

На рисунку 2.1 подано логічну схему взаємодії компонентів системи.

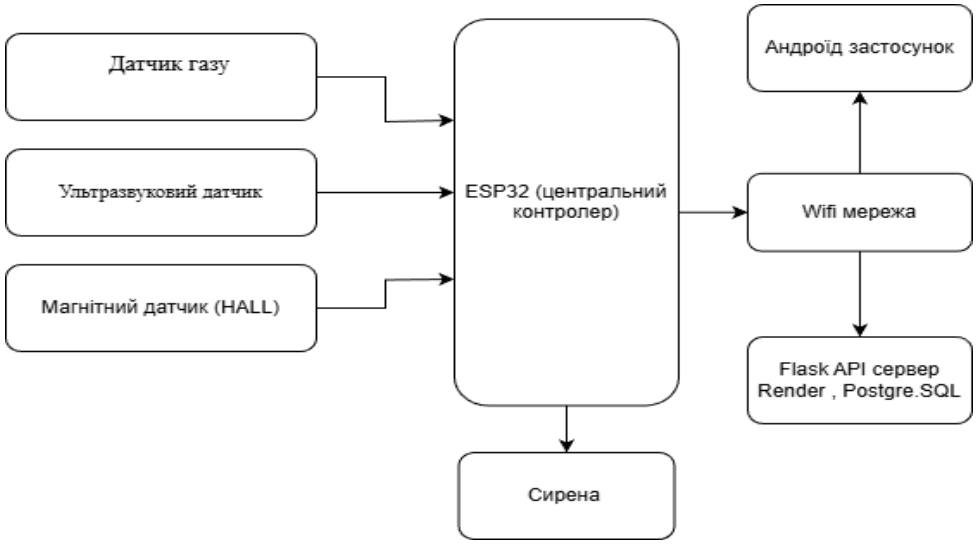

Рис. 2.1. Схема взаємодії компонентів системи

Такий підхід дозволяє забезпечити гнучкість, масштабованість і стабільність роботи системи. Кожен з елементів системи працює незалежно, взаємодіючи через API. У випадку втрати зв’язку мобільний застосунок не залежить напряму від ESP32, оскільки працює лише з сервером, який зберігає всю інформацію.

2.2. Вибір апаратного забезпечення

Для реалізації прототипу охоронної системи було обрано набір апаратних компонентів, які забезпечують взаємодію з навколишнім середовищем, обробку даних та відображення стану системи. Усі елементи підібрано з урахуванням доступності, простоти інтеграції та низького енергоспоживання.Дивитись додаток А.1 — таблицю апаратне забезпечення.

Усі компоненти обрано з урахуванням простоти інтеграції, доступності та низького енергоспоживання.

2.3. Реалізація обробки подій та передача даних через ESP32

На мікроконтролері ESP32 реалізовано основну логіку взаємодії з підключеними сенсорами та передачі даних на сервер. Пристрій працює в режимі постійного моніторингу стану навколишнього середовища та подій у приміщенні.

Після ввімкнення ESP32 ініціалізує Wi-Fi-з’єднання, використовуючи збережені параметри (SSID, пароль), які було раніше передано з мобільного додатку. У разі відсутності збережених налаштувань пристрій переходить у режим точки доступу, дозволяючи користувачу ввести потрібну конфігурацію через локальну веб-сторінку.

Після успішного з'єднання з мережею ESP32 активує підключені сенсори:

-

MQ-9 — газовий датчик, який працює в цифровому режимі: він надсилає сигнал у разі перевищення допустимого рівня газу в повітрі.

-

ерконовий датчик спрацьовує при відкриванні дверей, змінюючи логічний рівень на відповідному піні.

-

C-SR04 визначає відстань до об'єкта за допомогою ультразвуку — розрахунок здійснюється за часом проходження імпульсу.

Обробка подій реалізована в циклі loop(). При виявленні будь-якої зміни (наприклад, відкриття дверей або наявність газу) пристрій формує JSON-структуру, яка містить:

- ип події (газ, двері, рух);
- татус події (наприклад, “відкрито” / “в нормі”);
- нікальний токен пристрою (для ідентифікації на сервері);
- асову мітку.

Ця структура надсилається на сервер через HTTPS POST-запит на маршрут /data. Для цього використовується бібліотека HTTPClient.h. У коді передбачено механізм повторної спроби відправлення у разі помилки або відсутності підключення.

Такий підхід дозволяє мінімізувати навантаження на ESP32: дані не зберігаються локально, а одразу передаються на сервер, де й обробляються. Це забезпечує високу швидкодію та простоту в масштабуванні.

2.4. Схема з’єднання апаратних компонентів

На апаратному рівні охоронна система базується на використанні мікроконтролера ESP32, до якого підключено кілька ключових сенсорів і виконавчих пристроїв. Ці компоненти забезпечують функціонування основних модулів контролю: виявлення диму або витоку газу, контролю за відкриттям дверей, визначення відстані до об'єктів, а також звукового сповіщення про загрозу. Окремо підключено клавіатуру 4×4 для керування режимом охорони, а також LCD-дисплей з інтерфейсом I2C (опційно — для локального виводу інформації).

Фізичне з'єднання компонентів реалізовано з дотриманням вимог сумісності по живленню (ESP32 працює на логіці 3.3В, а деякі сенсори — на 5В) та оптимального розведення виводів.

Див. додаток А.2 — таблицю відповідності компонентів і пінів ESP32 згідно з вихідним кодом прошивки.

Кожен сенсор передає сигнал на окремий цифровий пін, що дозволяє обробляти події незалежно. Газовий сенсор Flying Fish має вбудований аналогово-цифровий перетворювач і підключається до цифрового входу D34, реагуючи на перевищення порогу концентрації газу. Геркон підключено до піну D16 і цей датчик змінює свій логічний стан при відкриванні дверей. Для HC-SR04 використовуються пари пінів Trig (D5) та Echo (D18), а обчислення відстані здійснюється у циклі loop() за часом проходження сигналу.

Окрема увага приділена клавіатурі — для підключення 4×4 матриці задіяно вісім пінів ESP32 (4 рядки, 4 стовпці). Опитування клавіш реалізується через бібліотеку Keypad, що дозволяє реагувати на введення коду користувачем для активації чи деактивації системи.

Такий спосіб підключення забезпечує надійну роботу системи, уникнення конфліктів на пін-контактах та дозволяє легко масштабувати рішення — наприклад, додати ще один ESP32 або додаткові сенсори без суттєвих змін у схемі.

2.5. Реалізація серверної частини на Flask

Серверна частина охоронної системи реалізована за допомогою Python-фреймворку **Flask**. Вона виступає в ролі посередника між мікроконтролером ESP32, який надсилає дані про стан середовища, та мобільним додатком користувача, який отримує ці дані у вигляді журналів подій та системних повідомлень. Сервер забезпечує обробку HTTP-запитів, авторизацію користувачів за допомогою JWT-токенів, а також взаємодію з базою даних PostgreSQL для зберігання інформації про пристрої та події.

Сервер розгорнуто на хмарній платформі Render, що дозволяє забезпечити стабільну доступність сервісу без необхідності самостійного обслуговування інфраструктури.

Основні функціональні можливості серверної частини:

- обробка реєстрації та автентифікації користувачів;
- прийом телеметричних даних з ESP32 через /push_data;
- збереження подій у форматі JSONB у таблиці device_data;
- надання клієнтському застосунку доступу до журналів пристроїв;
- додавання, редагування та оновлення налаштувань пристроїв (SSID, IP, назва тощо);
- перевірка статусу системи (systemAlive).

База даних PostgreSQL містить такі основні таблиці:

- sers: зберігає облікові записи користувачів;
- evices: зберігає прив’язані до користувачів пристрої;
- evice_data: містить журнали подій, що надходять з ESP32.

Основна точка взаємодії з ESP32 — маршрут /push_data, який приймає POST-запит з наступною структурою:

{"device_token": "8545534",
"payload": { "gas": "ok", "distance": "87","door":"closed","systemAlive": "ok }}

На сервері ці дані зберігаються як JSONB, що дозволяє зручно здійснювати вибірки по ключам, зокрема для перевірки наявності події systemAlive або перегляду останніх журналів подій.

Нижче представлено скріншот (рис. 2.2) коду обробки цього запиту.

```python
@app.route('/push_data', methods=['POST'])
def push_data():
    data = request.json
    device_token = data.get('device_token')
    payload = data.get('payload')
    if not device_token or not payload:
        return jsonify({'status': 'error', 'message': 'Missing data'}), 400

    conn = get_db_connection()
    cur = conn.cursor()
    cur.execute(
        "INSERT INTO device_data (device_token, data) VALUES (%s, %s)",
        (device_token, json.dumps(payload))
    )
    conn.commit()
    cur.close()
    conn.close()
    return jsonify({'status': 'success'})
```

Рис. 2.2. Обробка запиту /push_data

Отримання подій здійснюється за маршрутом /get_device_data (авторизований доступ). Сервер перевіряє, чи цей device_token належить поточному користувачу, і у разі успіху надає останні 20 подій.

Авторизація реалізована на основі JWT (JSON Web Tokens). Після реєстрації або входу користувач отримує токен, який надалі передається у заголовках запитів Authorization. Сервер розшифровує токен та дозволяє доступ лише авторизованим користувачам.

Окремий маршрут /update_device_settings дозволяє оновити такі параметри пристрою, як ім’я, SSID та пароль Wi-Fi. Це реалізовано через метод POST з динамічною перевіркою присутності параметрів.Функція моніторингу життєздатності системи реалізована у вигляді запиту до останнього запису з ключем "systemAlive": "ok". Якщо останній сигнал від ESP32 не містить такого поля — система вважається неактивною.

Таким чином, серверна частина побудована з дотриманням принципів безпеки, масштабованості та модульності. Вона є ключовим елементом у

передачі, зберіганні та фільтрації даних між апаратною частиною та користувачем.

2.6. Реалізація Android-додатку

Android-додаток є ключовим елементом взаємодії користувача з охоронною системою. Він забезпечує автентифікацію, реєстрацію нових користувачів, перегляд та зміну налаштувань пристроїв, а також отримання оновленої інформації про стан об’єкта охорони. Застосунок написаний мовою Kotlin з використанням бібліотек Jetpack (Room, LiveData, ViewModel) і AndroidX, що дозволяє дотримуватися принципів сучасної архітектури Android-додатків.

Загальна структура додатку включає наступні компоненти:

- ctivity та Fragment-и для інтерфейсу;
- oom-базу для локального зберігання пристроїв і журналів;
- etrofit/WebSocketService та HomePage для мережевої взаємодії;
- iewModel-и для роботи з даними;
- ласні компоненти відображення (RecyclerView Adapter-и);
- haredPreferences для зберігання токенів авторизації.

Інтерфейс поділено на такі основні розділи:

- головна сторінка (HomeFragment): відображає список підключених пристроїв з їхнім поточним статусом;

- журнали подій (GasLogFragment, DistanceLogFragment, DoorLogFragment): відображають історію станів відповідно до типу датчика;
- екран додавання нового пристрою (AddDeviceFragment);
- екран редагування налаштувань пристрою (EditDeviceSettings);
- екран входу та реєстрації (LoginActivity);
- навігаційна панель (NavBar.kt) забезпечує швидкий перехід між основними розділами.

Реєстрація користувача відбувається через POST-запит до /register, а вхід — до /login. У відповідь користувач отримує JWT-токен, який зберігається локально у SharedPreferences і використовується для подальших запитів до серверу.

Кожен пристрій у додатку представлений як об'єкт Device.kt і зберігається у локальній Room-базі через DeviceDao.kt. Отримання та оновлення даних здійснюється через DeviceRepository.kt, що виступає в ролі проксі між ViewModel та мережею або локальним сховищем.

Отримання журналів подій реалізовано через маршрут /get_device_data. Відповідні фрагменти (GasLogFragment.kt, DistanceLogFragment.kt, DoorLogFragment.kt) надсилають запити із JWT-токеном у заголовку й отримують масив подій у форматі JSON. Ці події конвертуються у структури EventLog.kt та передаються до адаптерів RecyclerView для візуалізації.

Для зручності користувача реалізована можливість редагування параметрів пристрою (назва, SSID, пароль Wi-Fi) безпосередньо з інтерфейсу. Компонент EditDeviceSettings.kt оновлює поля локально та надсилає відповідний запит до /update_device_settings.

Фонову взаємодію з WebSocket-сервером (у перспективі для push-оновлень) реалізовано у WebSocketService.kt — цей компонент відповідає за з'єднання та прослуховування каналу в режимі реального часу. Наразі основна взаємодія відбувається через періодичне опитування HTTP-запитами.

Інтерфейс додатку адаптований до темної теми, компонування виконано за допомогою ConstraintLayout. Основні візуальні елементи (карти пристроїв, кнопки, поля вводу) оформлено з використанням Material Design.

Усі основні події та статуси системи представлені в лаконічному, але інформативному вигляді. Користувач може легко побачити останній стан кожного пристрою та переглянути історію змін. Всі дані, що надходять від ESP32 через сервер, структуровані за типом події та мають часову прив’язку.

Таким чином, Android-додаток виконує роль інтерактивного центру керування та моніторингу охоронної системи, забезпечуючи повноцінну взаємодію користувача з усіма компонентами системи.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ ПРОТОТИПУ ОХОРОННОЇ СИСТЕМИ: ПРОГРАМНА ЧАСТИНА

3.1. Загальна структура проєкту

Програмна частина охоронної системи базується на архітектурі клієнт-сервер, де основним засобом взаємодії є HTTPS-запити з передачею структурованих даних у форматі JSON. Центральним вузлом цієї взаємодії є сервер, реалізований на основі мікрофреймворку **Flask**, який приймає та обробляє запити як від пристрою ESP32, так і від Android-додатку. Сервер розгорнутий на хмарній платформі Render.com, що дозволяє забезпечити постійну доступність та масштабованість системи.

Система включає наступні компоненти:

-

ікроконтролер ESP32, який фіксує події від фізичних сенсорів (газ, рух, відкриття дверей) і відправляє дані на сервер через HTTPS POST-запити;

-

ерверна частина, яка приймає, обробляє, зберігає дані та відповідає на запити від мобільного додатку;

-

обільний застосунок на Android, що здійснює авторизацію користувача, відображає інформацію про стан пристроїв, дозволяє налаштовувати параметри та переглядати журнали подій.

Приклад обміну даними між ESP32 і сервером:

-

SP32 формує структуру типу JSONB та надсилає її на сервер через POST-запит на маршрут /push_data. На сервері ці дані зберігаються у таблиці device_data з полем типу JSONB, що дозволяє обробляти структуровану інформацію.

-

обільний застосунок, зі свого боку, здійснює GET-запити на маршрут /get_device_data з параметром device_token, вказуючи ідентифікатор пристрою, а також передає JWT-токен у заголовку Authorization. У відповідь сервер повертає масив подій із зазначенням часу створення кожної події.

Потік взаємодії виглядає так:

ESP32 ⟶ [POST] /push_data ⟶ сервер

Android-додаток ⟶ [GET] /get_device_data ⟶ сервер

Користувач ⟶ [POST] **/login**, **/register**, **/add_device ⟶ сервер**

Інші функціональні можливості включають:

- передачу IP-адреси пристрою при старті або перезапуску ESP32;
- оновлення параметрів пристрою (SSID, пароль) через окремий маршрут /update_device_settings;
- асоціацію кожного пристрою з конкретним користувачем;
- збереження історії подій у базі PostgreSQL з фільтрацією по device_token.

Такий підхід забезпечує чітку та зрозумілу логіку обміну даними, гнучкість масштабування та можливість розширення функціоналу в майбутньому.

3.2. Код ESP32 — сенсори, сирена, клавіатура

Програмна частина мікроконтролера ESP32 реалізована на мові C++ з використанням Arduino SDK та бібліотек, призначених для роботи з мережею, сенсорами, клавіатурою та дисплеєм. Основне завдання цього компонента це зчитування даних з підключених датчиків, та реагування на потенційно небезпечні події, формування звітів та передача їх на сервер, а також забезпечення базової взаємодії з користувачем через LCD-дисплей та клавіатуру.

3.2.1. Підключення бібліотек та глобальні змінні

На початку коду підключаються бібліотеки, необхідні для реалізації логіки пристрою:

- iFi.h — для підключення до Wi-Fi-мережі;
- references.h — для збереження та зчитування налаштувань мережі у flash-пам’яті;
- rduinoJson — для формування структури даних перед відправленням;
- eypad.h — для роботи з клавіатурою 4x4;
- iquidCrystal_I2C — для взаємодії з LCD-дисплеєм по шині I2C;
- TTPClient — для надсилання HTTP POST-запитів до сервера.

Оголошуються глобальні змінні, які визначають пін-конфігурацію, мережеві параметри та елементи інтерфейсу:

```
Preferences preferences;
String ssid, password;
const char* apSSID = "ESP3";
const char* apPas = "88556524";
```

Оголошення основних пінів сенсорів:

```
const int gasSensorPin = 34;
const int trigPin = 5;
```

const int echoPin = 18;

#define DOOR_SENSOR_PIN 16

#define SIREN_PIN 4

Оголошення пінів для блоку клавіатури:

byte rowPins[ROWS]={13, 12, 14, 27};

byte colPins[COLS]={26, 25, 33, 32};

3.2.2. Ініціалізація мережі

ESP32 при запуску перевіряє, чи збережені параметри бездротової мережі (SSID та пароль). Якщо вони відсутні або не вдається підключитися, створюється точка доступу (Access Point) з можливістю налаштування параметрів через мобільний застосунок.

Реалізація сканування сенсорів та логіки тривоги ESP32 постійно опитує підключені сенсори:

- газовий сенсор MQ-9 (модуль Flying Fish) — визначає рівень концентрації газу на основі аналогового сигналу;
- ультразвуковий сенсор HC-SR04 — використовується для виявлення руху або зміни положення об'єктів шляхом вимірювання відстані;
- магнітний герконовий сенсор — реагує на відкривання дверей, змінюючи логічний рівень на цифровому вході.

На основі цих даних формується об’єкт JSON, що містить поля:

{ "device_token": "...",

"payload": { "gasLevel": ...,

"motionDetected": ...,

"doorOpen": ...,

"systemAlive": "ok"

}}

Ці дані надсилаються на сервер через маршрут /push_data методом POST. У випадку виявлення небезпечної ситуації (відкриття дверей або перевищення газу) вмикається сирена та дані передаються на сервер.

Вся логіка охоплює:

-
ункції зчитування сенсорів;

-
бробку вводу з клавіатури;

-
ктивацію/деактивацію сигналізації;

-
иведення статусу на LCD;

-
еріодичну перевірку з'єднання та alive-повідомлень.

3.2.3. Стійкість до втрати з’єднання

Однією з важливих переваг запропонованої реалізації є передбачена логіка обробки втрати інтернет-з’єднання або збою Wi-Fi-підключення. У разі неможливості підключитися до збереженої мережі ESP32 автоматично перемикається в режим точки доступу, що дозволяє користувачу повторно передати правильні параметри Wi-Fi без потреби перепрошивки.

Окрім цього, дані сенсорів не передаються одразу по події, а акумулюються в окремій структурі перед формуванням JSON-запиту. Це дозволяє уникнути зайвого навантаження на сервер і реалізувати періодичну синхронізацію навіть при нестабільному інтернеті.

Гнучкість обробки вхідних даних на сервері також дозволяє не втратити контекст подій: навіть якщо пристрій кілька хвилин не був на зв’язку, у запиті все одно буде передано історичні дані, включаючи час останньої активності.

Інтерактивність та контроль

Інтеграція клавіатури, сирени та LCD-дисплею перетворює ESP32 з пасивного сенсорного вузла на повноцінний пристрій локального реагування. У випадку втрати зв’язку з сервером, ESP32 здатен самостійно активувати сирену та інформувати користувача через дисплей про критичну ситуацію. Такий підхід підвищує надійність системи у разі DDoS-атаки, падіння сервера чи аварійного вимкнення інтернету.

У результаті реалізована на ESP32 логіка охоплює повний цикл: від зчитування сенсорних даних, локального реагування, формування структурованого повідомлення у форматі JSON до його надсилання на сервер за протоколом HTTP. Передбачено обробку помилок, підтримку декількох режимів зв’язку, а також інтерфейс користувача на базі клавіатури та дисплея.

Реалізація є гнучкою, розширюваною (можна підключити додаткові сенсори) і стійкою до типових збоїв IoT-пристроїв.

3.3. Реалізація мобільного застосунку для взаємодії з системою

Застосунок, розроблений на платформі Android з використанням мови Kotlin, виконує функцію основного інтерфейсу користувача для взаємодії з охоронною системою. Його архітектура включає елементи авторизації, реєстрації, керування пристроями, перегляду журналу подій та редагування налаштувань. Завдяки цьому користувач має змогу повноцінно контролювати систему з мобільного телефону.

Розглянемо ключові компоненти застосунку та їхню реалізацію.

3.3.1. Реєстрація та вхід користувача

Процес аутентифікації побудований на основі передачі логіна і пароля на сервер через HTTPS-запит. У файлі LoginActivity.kt реалізована форма входу, яка надсилає запит типу POST на маршрут /login. У відповідь сервер повертає JWT-токен, що зберігається локально:

val requestBody = JSONObject()

```
requestBody.put("username", username)
requestBody.put("password", password)
val request = JsonObjectRequest(Request.Method.POST,
"$BASE_URL/login", requestBody,
    {
response ->val token = response.getString("token")
    saveTokenToPrefs(token)
      }, { error ->
    showErrorToast("Помилка входу") })
```

Аналогічно працює механізм реєстрації нового користувача (маршрут /register), реалізований у тій же активності.

3.3.2. Пристрій та взаємодія з ним

Після входу на головний екран (HomePage.kt) користувач бачить підключений пристрій. Ці дані завантажуються з сервера (маршрут /get_devices), де кожен пристрій містить IP-адресу, SSID та ім'я:

```
val request= object : JsonArrayRequest(Method.GET,
"$BASE_URL/get_devices", null, {
 response ->
val devices = parseDevices(response) deviceAdapter.submitList(devices)
 }, {
error -> Log.e("API", "Помилка завантаження пристроїв", error)
 }) {
override fun getHeaders(): MutableMap<String, String>
{ val headers = HashMap<String, String>()
headers["Authorization"] = "Bearer $token" return headers } }
```

Особливістю реалізації є введення відкладеного спрацювання сирени після відкриття дверей. Користувач має 30 секунд, щоб ввести правильний

код на клавіатурі, після чого сирена не активується. У протилежному випадку — сигналізація спрацьовує. Така логіка реалізується перевіркою значення door sensor (зчитується з піну 16) та ініціалізацією таймера. Якщо введено правильний пароль (порівняння з EEPROM), спрацювання системи оповіщення.

3.3.3. Додавання пристрою

Через AddDeviceFragment.kt реалізовано інтерфейс для додавання нового пристрою: користувач вводить SSID, пароль, ім’я пристрою, після чого ці дані передаються через маршрут /add_device. Після авторизації користувач має змогу додати новий пристрій до системи, вказавши його параметри, як-от токен, IP-адресу та назву.

3.3.4. Перегляд журналу подій

Кожен пристрій має окремий журнал подій за який відповідає (EventLog.kt), який завантажується з маршруту /get_device_data. У фрагментах DoorLogFragment.kt, GasLogFragment.kt та DistanceLogFragment.kt реалізовані різні типи логів залежно від сенсора.

Ілюстровано список останніх повідомлень від пристрою на основі інформації, збереженої в базі даних.

Рис. 3.1. Демонстрація логів рівня газу.

Відображення подій реалізовано за допомогою RecyclerView у fragment_log_list.xml, а окремі записи — через item_log.xml.

3.3.5. Редагування налаштувань пристрою

EditDeviceSettings.kt дозволяє змінити ім’я пристрою, Wifi SSID та пароль. Дані передаються через маршрут /update_device_settings. Після А.2редагування — зміни синхронізуються з базою даних, і при наступному запуску ESP32 отримає оновлені параметри Wi-Fi. Як показано в додатку Б.1.

3.3.6. Локальна база даних (Room)

Для кешування даних використано бібліотеку Room: файли Device.kt, DeviceDao.kt, AppDatabase.kt дозволяють зберігати пристрої в локальній базі. Це дає змогу переглядати останній стан навіть без підключення до інтернету.

Інтерфейс користувача

Використано Material Design компоненти — MaterialCardView, NavigationBar, різні іконки, тема інтерфейсу змінюється відповідно до вибраної в налаштуваннях телефону. Фрагмент home_fragment.xml є

основним контейнером інтерфейсу. Всі елементи адаптовані під екрани різних розмірів.

Мобільний застосунок виступає повноцінним контролером системи: від аутентифікації до перегляду подій, керування налаштуваннями та додавання пристроїв. Весь обмін даними реалізований через захищені запити з використанням JWT-токенів. Архітектура розширювана та придатна для масштабування.

3.4. Реалізація серверної частини (Flask API).

У цьому розділі описано реалізацію серверної частини програмного забезпечення системи, що відповідає за обробку даних від пристрою ESP32, їх збереження в базі даних та взаємодію з мобільним застосунком. Сервер побудований на базі фреймворку Flask і підтримує авторизацію користувачів, маршрутизацію запитів та обробку подій.

3.4.1. Архітектура Flask-сервера

Серверна частина реалізована з використанням фреймворку **Flask** — мікрофреймворку на мові програмування Python, що забезпечує легкість розгортання, масштабованість та зручну інтеграцію з базами даних. Основне призначення серверної частини — приймати дані від пристрою ESP32, зберігати їх у структурованому вигляді в базі даних PostgreSQL та надавати доступ до них мобільному застосунку користувача.

Структура проєкту дотримується принципів розділення обов’язків:

- крема логіка аутентифікації користувачів;
- бробка запитів від пристроїв та клієнтів;
- аршрути для зчитування та оновлення інформації;

- онфігурація бази даних і ініціалізація таблиць.

Ключовими бібліотеками, що використовуються у проєкті, є:

- **lask** — для створення веб-додатку;
- **sycopg2** — для взаємодії з PostgreSQL;
- **lask_cors** — для підтримки CORS (Cross-Origin Resource Sharing);
- **crypt** — для хешування паролів;
- **wt** — для реалізації токен-базованої автентифікації;
- **unctools.wraps** — для декоратора @token_required.

Для гнучкості та адаптивності до середовища хостингу (Render.com), використовуються змінні середовища (DATABASE_URL, SECRET_KEY), які забезпечують безпечне зберігання конфіденційних параметрів.

Таким чином, архітектура сервера відповідає вимогам до сучасних IoT-рішень: розподілена логіка, захищена передача даних, розширюваність функціоналу та інтеграція з фронтендом і фізичними пристроями.

3.4.2. Структура бази даних PostgreSQL

Серверна частина охоронної системи використовує реляційну базу даних PostgreSQL для зберігання користувацької інформації, параметрів пристроїв та журналів отриманих даних. Такий вибір зумовлений високою надійністю, стабільністю та широкими можливостями PostgreSQL щодо роботи з JSONB-полями, що актуально для зберігання структурованих показників, які надходять від пристрою ESP32.

У базі даних реалізовано такі основні таблиці:

users — зберігає облікові записи користувачів системи:

- d (SERIAL PRIMARY KEY) — унікальний ідентифікатор.
- sername (TEXT) — логін користувача.
- assword (TEXT) — пароль користувача.

devices — містить інформацію про пристрої, прив’язані до конкретних користувачів:

- d — ідентифікатор пристрою.
- evice_token — унікальний токен пристрою (використовується ESP32 для автентифікації).
- evice_name, ip_address, ssid, wifi_password — параметри підключення та ідентифікації.
- ser_id — зовнішній ключ, що вказує на користувача.
- ast_seen — мітка часу останнього зв'язку.

device_data — таблиця з журналом подій від пристроїв:

- d — унікальний запис.
- evice_token — прив’язка до пристрою.

-

ata — поле типу JSONB, у якому містяться показники датчиків.

-

reated_at — мітка часу отримання даних.

Така структура дає змогу гнучко масштабувати систему: з одного боку, забезпечується нормалізований зв’язок між користувачами та їх пристроями, а з іншого — зберігання сенсорних даних у вигляді JSONB дозволяє зберігати довільну кількість параметрів без необхідності модифікувати схему бази даних у майбутньому.

Крім того, структура дозволяє зручно реалізовувати запити для відображення останніх подій, аналітики активності пристроїв та перевірки працездатності системи в цілому. Наприклад, мобільний застосунок може запитати останні 20 подій для пристрою, або перевірити останній запис із ознакою systemAlive = "ok" — і все це за допомогою оптимізованого запиту до відповідного JSONB-поля.

Початкова ініціалізація бази даних виконується через функцію init_db(), яка автоматично створює необхідні таблиці. Також реалізовано CLI-команду та окремий ендпоінт /init_db, що дозволяє ініціалізувати базу віддалено під час розгортання на сервері (наприклад, на платформі Render).

3.4.3. Авторизація та безпека

Одним із ключових аспектів безпечної взаємодії користувача з охоронною системою є контроль доступу та захист персональних даних. Для досягнення цього у серверній частині реалізовано механізм авторизації з використанням JSON Web Token (JWT) та хешування паролів за допомогою бібліотеки bcrypt.

3.4.4. Хешування паролів

Під час реєстрації новий пароль користувача не зберігається у відкритому вигляді. Замість цього використовується алгоритм хешування bcrypt:

- ароль шифрується методом bcrypt з автоматично згенерованою (salt);
- еш зберігається у полі password таблиці users;
- ід час авторизації введений пароль проходить через ту саму процедуру хешування, і результат порівнюється з тим, що збережено в базі.

Це дозволяє надійно захистити облікові дані навіть у разі компрометації бази даних.

Авторизація за допомогою JWT

Після успішного входу користувач отримує токен доступу у форматі JWT (JSON Web Token), який містить зашифровану інформацію про користувача:

Тіло токена включає такі дані:

- ub (subject) — ім’я користувача;
- at (issued at) — час видачі;
- xp (expiration) — час завершення дії токена (в поточній реалізації — 2 години).

Токен підписується секретним ключем, заданим у конфігурації **Flask**-додатку (SECRET_KEY), та надсилається клієнту у відповіді на /login.

Надалі клієнт повинен додавати цей токен у заголовок Authorization при кожному запиті до захищених маршрутів, наприклад:

Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6…

Перевірка авторизації

Функція-декоратор @token_required виконує такі дії:

- итягує токен із заголовка HTTPS-запиту;
- озшифровує його за допомогою SECRET_KEY;
- еревіряє дійсність (строк дії, цілісність, наявність у базі користувача);
- разі успіху передає ім’я користувача (current_user) у відповідну функцію маршруту.

Якщо токен відсутній або недійсний, сервер повертає повідомлення про помилку 401 Unauthorized.

Цей механізм дозволяє відділити аутентифікацію (вхід) від авторизованої роботи з системою, а також зручно розмежовувати доступ до персональних пристроїв та даних у базі.

Безпека при передачі даних

Оскільки ESP32 і сервер обмінюються даними через HTTP-запити, рекомендовано використовувати HTTPS-протокол при розгортанні . Це дозволяє захистити трафік від перехоплення (man-in-the-middle attack), що особливо важливо під час передачі облікових даних і конфіденційних налаштувань пристроїв.

3.4.5. Огляд основних ендпоінтів API

У рамках серверної частини охоронної системи реалізовано набір REST-орієнтованих маршрутів (ендпоінтів), які забезпечують взаємодію між клієнтською частиною (мобільний застосунок та пристрій ESP32) і сервером. Усі основні маршрути згруповано за функціональністю:

Реєстрація та автентифікація:

- OST /register — реєстрація нового користувача. Приймає JSON із логіном та паролем, зберігає хеш пароля у базі;
- OST /login — автентифікація користувача. Повертає JWT-токен у випадку успішного входу;

Робота з пристроями:

- OST /add_device — додає новий пристрій, пов’язаний з користувачем. Приймає JSON з даними пристрою (токен, IP, SSID тощо). Захищений токеном;
- ET /get_devices — повертає список усіх пристроїв, зареєстрованих для поточного користувача. Захищений токеном;
- OST /update_device_ip — оновлює IP-адресу пристрою за його токеном (використовується ESP32 після підключення до мережі);
- OST /update_device_settings — оновлює параметри пристрою (ім’я, SSID, пароль). Захищений токеном.

Отримання даних від пристроїв:

- OST /push_data — приймає дані від ESP32 у форматі JSON. Поле payload містить інформацію з сенсорів (двері, газ, відстань);
- ET /get_device_data — повертає останні 20 записів журналу з таблиці device_data для заданого пристрою. Захищений токеном;
- ET /get_system_alive — перевірка працездатності пристрою. Повертає останню мітку часу, коли було отримано systemAlive: ok.

Технічна частина

- OST /init_db — ініціалізація структури бази даних. Створює таблиці, якщо вони ще не існують. Використовується під час розгортання;
- OST /test_add_device — допоміжний маршрут для налагодження, повертає отриманий JSON-запит без запису в базу.

Сервер зберігає отриманий payload у таблиці device_data, прикріплюючи поточний timestamp.

Ці маршрути утворюють повноцінне API для зчитування, збереження та керування параметрами охоронної системи в режимі реального часу. Додатково, архітектура дозволяє масштабування — зокрема, можливість розширити набір подій, додати нові типи пристроїв або аналітичні сервіси без зміни базової логіки.

3.5. Реалізація мобільного додатку

Для забезпечення зручного керування охоронною системою та перегляду стану сенсорів розроблено мобільний додаток під операційну

систему Android. Застосунок реалізовано мовою Kotlin з використанням бібліотек Android Jetpack, ViewModel, LiveData та Room, що забезпечують реактивність, стійкість до поворотів екрана і чисту архітектуру.

Інтерфейс користувача побудовано на основі фрагментів, які перемикаються через нижню навігаційну панель (NavBar.kt). Основні компоненти додатка:

- кран входу та реєстрації (LoginActivity.kt);
- оловна панель з пристроями (HomePage.kt);
- урнал подій (фрагменти GasLogFragment.kt, DistanceLogFragment.kt, DoorLogFragment.kt);
- кран додавання нового пристрою (AddDeviceFragment.kt);
- алаштування пристрою (EditDeviceSettings.kt).

3.5.1. Архітектура з Room та ViewModel

Для зберігання локальних даних використовується база даних Room (AppDatabase.kt), яка взаємодіє з класами:

- evice.kt — модель пристрою;
- eviceDao.kt — DAO-інтерфейс для роботи з таблицею пристроїв;
- eviceRepository.kt — проміжний шар між ViewModel та DAO;

-

eviceViewModel.kt — ViewModel, що зберігає дані для життєвого циклу фрагментів.

Ця архітектура забезпечує ефективне управління станом застосунку, а також підтримку оновлення інтерфейсу при зміні даних.

3.5.2. Підключення до серверу

Додаток використовує стандартну бібліотеку HTTP (через OkHttp або HttpURLConnection) для відправлення запитів до **Flask** API. При вході або реєстрації користувача отримується JWT-токен, який зберігається локально та додається в заголовок кожного наступного запиту.

Також реалізовано забезпечує отримання з низькою затримкоюв режимі реального часу, зокрема для швидкого відображення змін статусу пристрою (відкриття дверей, перевищення рівня газу тощо).

3.5.3. Оформлення інтерфейсу

Користувацький інтерфейс оформлено за допомогою:

- елементи журналу (item_log.xml);
- вікна конфігурації(fragment_edit_device.xml, add_device_fragment.xml);
- головний екран (home_fragment.xml);
- адаптери по типу (LogAdapter.kt) для прив’язки даних до RecyclerView.

На головному екрані система будинку відображаються у вигляді картинок з назвою, статусом та кнопками для ігнорувань.

3.5.4. Реєстрація та додавання пристрою

Після реєстрації та авторизації користувач потрапляє на головний екран. Кнопка «Додати пристрій +» відкриває форму (add_device_and_settings.xml), у якій необхідно вказати:

- азву пристрою;
- ого унікальний токен (отриманий із LCD екрану ESP32);
- SID і пароль Wi-Fi;
- нші параметри (опційно).

Після підтвердження пристрій реєструється через API /add_device та зберігається в локальну базу.

3.5.5. Робота з журналами

Фрагменти GasLogFragment, DoorLogFragment, DistanceLogFragment використовують логіку, яка запитує останні події з сервера через /get_device_data. Дані відображаються у RecyclerView з часовою міткою, значенням і типом події.

3.6. Візуалізація роботи системи

Ефективна візуалізація стану охоронної системи є ключовою для зручності користувача. Android-застосунок, розроблений у рамках цього проєкту, дозволяє переглядати стан підключеного пристрою, журнал подій, а також оновлювати параметри системи.

3.6.1. Відображення інформації про пристрій

Інтерфейс головного екрана, реалізований у класі HomePage.kt, призначений для демонстрації поточного статусу одного підключеного пристрою.

Основні елементи інтерфейсу включають:

- відображення назви пристрою;
- індикацію статусу з'єднання з мережею (визначається на основі часу останнього оновлення);
- кнопку для переходу до налаштувань пристрою (наприклад, EditDeviceSettings.kt);
- кнопки для переходу до журналів подій, зокрема для датчиків газу, руху та дверей.

Дані про пристрій отримуються через HTTPS-запит до ендпоінту **Flask**-сервера /get_devices із JWT-автентифікацією. У відповідь надходить масив пристроїв, з якого застосунок обробляє перший запис (оскільки підтримується лише один пристрій на користувача).

3.6.2. Перегляд журналів подій

Окремі фрагменти — GasLogFragment.kt, DistanceLogFragment.kt та DoorLogFragment.kt — відповідають за виведення історії подій, що надходять від пристрою. Дані передаються на сервер ESP32 через ендпоінт /push_data, а згодом зчитуються Android-застосунком через запит /get_device_data.

Журнали відображаються у вигляді списку за допомогою адаптера LogAdapter.kt, який наповнює макет item_log.xml.

Кожен запис у журналі містить такі атрибути:

- **ип події** (газ, відстань, двері);
- **начення або статус** зафіксованої події;

-

ас фіксації події.

РОЗДІЛ 4

ТЕСТУВАННЯ ПРОТОТИПУ ОХОРОННОЇ СИСТЕМИ ТА ОЦІНКА ЕФЕКТИВНОСТІ

4.1. Мета тестування

Основною метою тестування було перевірити:

- коректність роботи сенсорів та реакції ESP32;

- надійність зв’язку між ESP32 та Android;
- стабільність мобільного застосунку при різних сценаріях використання;
- правильність запису логів подій;
- відповідність реакцій системи очікуванням користувача (юзабіліті).

Тестування проводилося як у лабораторних умовах, так і в умовах реального приміщення.

4.2. Тестування виявлення подій

4.2.1. Сценарій 1: Витік газу

Умови: до датчика газу піднесено джерело (запальничка без вогню).

Очікуваний результат:

- газовий сенсор фіксує перевищення рівня (значення > 2200).
- ESP32 передає JSON через сервер: {"gasLevel": 2383};
- Android-застосунок змінює статус на «Gas: ALARM», червоний
- колір;
- спрацьовує сирена;
- подія записується у таблицю event_logs.

Фактичний результат:Усі дії виконано правильно. Затримка до реакції: < 1.2 с.

4.2.2. Сценарій 2: Відкриття дверей

Умови: геркон розімкнено вручну (імітація відкриття дверей).

Результат:

-

ерез 30 секунд після відкриття дверей і при умові не введеного паролю передається alarm = 1, doorStatus = "ALARM".

-

а Android — "Door: ALARM", вмикається сирена, лог: "ALARM".

Реакція стабільна, без помилок. Фіксується одразу після зміни статусу піну. Відповідний інтерфейс наведено на рисунку 4.1.

Рис. 4.1. Сповіщення несанкціонованого відкриття дверей

4.2.3. Сценарій 3: Рух у зоні контролю

Умови: перед ультразвуковим сенсором з’являється об’єкт < 200 см.

Результат:

- ESP32 фіксує значення відстані, надсилає тривогу;
- У застосунку виводиться "Distance: ALARM", сирена активується.

Як видно з інтерфейсу, повідомлення про тривогу відображається у відповідному блоці. Реакція системи — миттєва та безпомилкова.

Вигляд сторінки наведено на рисунку 4.2 (наступна сторінка).

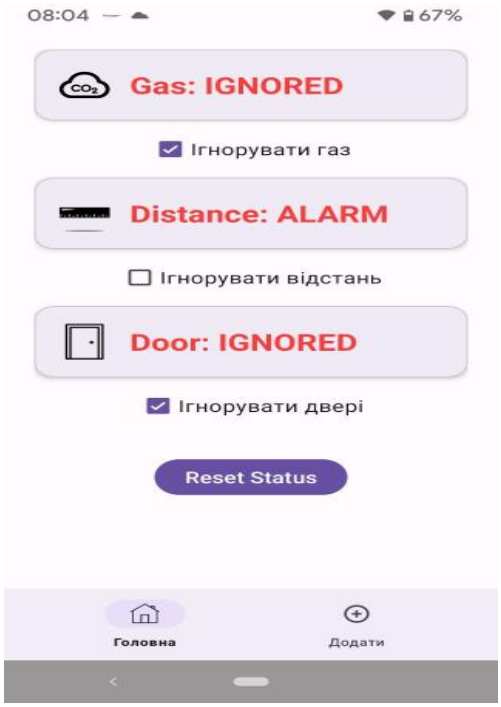

Рис. 4.2. Сповіщення через прохід перед сенсором

Статус:

Тест пройдено.

4.3. Тестування логування

4.3.1. Сценарій: Перегляд логів

Усі події записуються у локальну БД Room. Логи фільтруються по типу подій (gas, distance, door) і даті.

Функціонал:

- ільтрація по даті (через DatePicker);
- чищення логів кнопкою;
- ідображення часу і значення (із кольоровим маркуванням).

Реалізація повністю відповідає вимогам: UX простий, адаптований до середньстатистичного користувача.

4.4. Перевірка взаємодії з сервером

4.4.1. Реєстрація пристрою

Запит /add_device з параметрами:

{ "device_token": "123-456",

"device_name": "MyESP",

"ip_address": "192.168.0.101",

"ssid": "MyWiFi",

"wifi_password": "pass"}

Результат: Після запиту — з'являється повідомлення про успішне додавання пристрою до акаунту . На стороні сервера ми можемо побачити успішне виконання запиту за логом (10.201.224.65 - - [13/Jun/2025 06:14:26] "POST /add_device HTTP/1.1" 200 -).

4.4.2. Перевірка авторизації

Після реєстрації вводимо свіої дані та авторизуємось в додаток .JWT-токен успішно перевіряється, зберігається в локальній БД під ім'ям token. При відсутності токена мобільний застосунок не дозволяє взаємодію з сервером. Приклад вигляду сторінки наведено нижче:

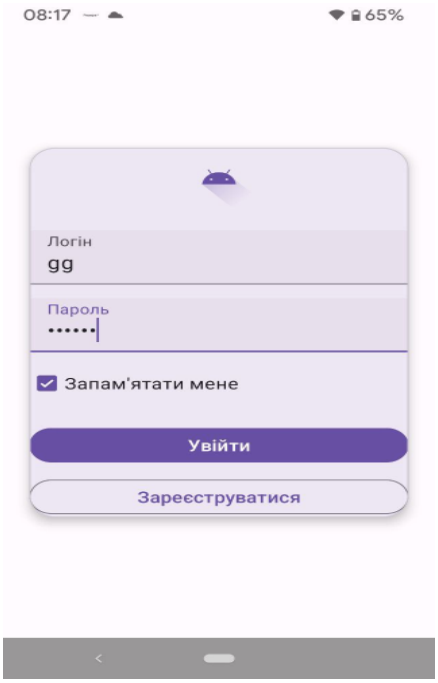

Рис. 4.3. Сторінка авторизації в застосунок

4.5. Тестування UX/UI

4.5.1. Інтерфейс

- **авігація**: реалізована через Navigation Component.
- **олірна індикація**: зелений (OK), червоний (ALARM) та (IGNORED).
- **нопка скидання**: дозволяє вимкнути сирену вручну.

Загальний інтерфейс адаптовано до користувачів без технічних знань. Тексти підписів українською, лаконічні. UX оцінено як "зрозумілий, функціональний".

Таблиця 4.5. Загальна оцінка ефективності системи

Критерій	Оцінка	Примітка
Реакція на події	< 2 с.	Залежить від з'єднання
Виявлення сенсорів	Надійне	Газ > 1300, Відстань < 200, Двері: open

Витрати ресурсів	Низькі	Працює на ESP32 із середнім енергоспоживанням
Юзабіліті застосунку	Високе	Просте керування, логічна навігація
Робота серверної частини	Стабільна	**Flask** + PostgreSQL + JWT
Масштабованість	Підтримується	Можна модифікувати

Проведене тестування UX/UI підтвердило зручність та ефективність взаємодії користувача з мобільним застосунком. Візуальні індикатори та навігація забезпечують швидке орієнтування у станах системи. Система демонструє стабільну роботу серверної частини та швидку реакцію на події, водночас споживаючи мінімальні апаратні ресурси.

ВИСНОВОКИ

У цій кваліфікаційній роботі було реалізовано прототип охоронної системи для приватного будинку з використанням апаратно-програмного комплексу, побудованого на базі **ESP32**, **Arduino-сумісних сенсорів**, **мобільного застосунку Android**, а також **серверної частини на Flask API**.

Досягнуті результати:

- **наліз предметної області** охоронних систем і IoT дав змогу обґрунтувати вибір архітектури та технологій.
- озроблено **архітектуру системи**, що охоплює апаратні та програмні компоненти з чітким розмежуванням відповідальностей.

Реалізовано:

- бір і аналіз даних з трьох типів сенсорів (газ, рух, двері);

- иявлення тривожних подій та запуск сирени;
- синхронний обмін даними між ESP32 та Android;
- єєстрацію та авторизацію користувачів на Flask API-сервері з JWT-аутентифікацією;
- овноцінний Android-застосунок для взаємодії з пристроєм, конфігурації, перегляду логів.

Система пройшла **тестування в реальних умовах**, показавши стабільну роботу і адекватну реакцію на події.

Практична цінність:

- истема може використовуватися як **базова модель для розумного будинку** або для малого офісу.
- артість реалізації низька, а технології — відкриті та легко масштабовані.
- рхітектура підтримує **додавання нових сенсорів** або **інтеграцію з хмарними сервісами**.

Обмеження:

- аразі немає шифрування трафіку.

Перспективи подальшого розвитку:

1. **одавання Push-повідомлень** (наприклад, через Firebase Cloud Messaging).

2.

ифрування зв’язку між ESP32 та сервером (через TLS/SSL).

3.

нтеграція з Telegram-ботом або e-mail-сповіщеннями для додаткових каналів попередження.

4.

асштабування до кількох об’єктів та централізоване адміністрування через веб-інтерфейс.

5.

езервне живлення та автономність — підтримка живлення від батареї з контролем заряду.

Список використаних джерел

Програмування мікроконтролерів ESP32: веб-сервер. URL: https://randomnerdtutorials.com/esp32-web-server-arduino-ide/

PostgreSQL. Документація до системи керування базами даних. URL: https://www.postgresql.org/docs/

Flask (Python microframework). Офіційна документація. URL: https://flask.palletsprojects.com/

W3Schools. JSON Tutorial. URL: https://www.w3schools.com/js/js_json_intro.asp

Android & Kotlin Development Masterclass – Full Course. URL: https://www.youtube.com/watch?v=blKkRoZPxLc&ab_channel=freeCodeCamp.org

How to Use an I2C LCD Display With ESP32. URL: https://lastminuteengineers.com/esp32-i2c-lcd-tutorial/

ДОДАТКИ

Проектування прототипу

Додаток А

Таблиця А.1 Апаратне забезпечення

Компонент	**Модель / Тип**	**Призначення**
Мікроконтролер	ESP32-WROOM-32D	Передача та обробка даних
Датчик газу	Flying Fish (тип MQ-9)	Виявлення газів
Ультразвуковий сенсор	HC-SR04	Контроль руху
Датчик відкриття дверей	Геркон	Фіксація відкриття/закриття
Клавіатура	4x4 Matrix Keypad	Введення коду безпеки
Сирена	Активний бузер	Сповіщення про тривогу
Екран	LCD 1602 + I2C	Вивід статусу системи
Мобільний пристрій	Android 8.0+	Інтерфейс користувача

Таблиця А.2 Відповідності компонентів і пінів ESP32

Назва пристрою	**Сигнал (призначення)**	**Пін ESP32**
Газовий сенсор Flying Fish	DO (цифровий вихід)	34
Ультразвуковий датчик HC-SR04	Trig Echo	5 18
Магнітний датчик (геркон)	DO	16
Сирена	IN (керування)	4
Клавіатура 4×4	Row0 – Row3 , Col0 – Col3	14, 27, 26, 25, 33, 32, 35, 34
LCD-дисплей I2C	SDA / SCL	21 / 22

Додаток Б

Додаток Б.1 Сторінка редагування налаштувань пристрою