БАКАЛАВРСЬКА РОБОТА

БР.КІ–36.00.000 ПЗ

Група КІ-22-2

Коваль Олександр

2026

Міністерство освіти і науки України
Івано-Франківський Національний Технічний
Університет Нафти і Газу
Факультет інформаційних технологій
Кафедра комп'ютерних систем і мереж

Коваль Олександр Андрійович

УДК 004.415.2+681.586

БАКАЛАВРСЬКА РОБОТА

Розробка мікроконтролерної системи періодичного моніторингу параметрів повітря приміщень на базі ESP32 та Wi-Fi Fingerprinting

Комп'ютерна інженерія

123 – Комп'ютерна інженерія

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

Здобувач освітнього ступеня бакалавра _____ *Коваль О. А.*
(підпис, ініціали та прізвище здобувача)

Науковий керівник _____ *Кропивницька В. Б. , доцент*
(підпис, ПІБ, науковий ступінь, вчене звання)

Допущено до захисту
Завідувач кафедри КСМ

д.т.н., доцент _____ *Мельничук С. І.*
(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківськ – 2026 рік

Івано-Франківський національний технічний університет нафти і газу

Факультет *Інформаційних технологій*

Кафедра *Комп'ютерних систем і мереж*

Освітньо-кваліфікаційний рівень *бакалавр*

Спеціальність *123 – Комп'ютерна інженерія*

ЗАТВЕРДЖУЮ:
Зав. кафедрою КСМ
С.І. Мельничук
« 21 квіт*ня 2026*року

З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Ковалю Олександру Андрійовичу
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи*) Розробка мікроконтролерної системи періодичного моніторингу параметрів повітря приміщень на базі ESP32 та Wi-Fi Fingerprinting*

керівник проекту (роботи) *Кропивницька В. Б., доцент* .

затверджені наказом вищого навчального закладу від *21.04.2026 № 180/7*

2. Строк подання студентом проекту (роботи) *11 червня 2026р.*

3. Вихідні дані до роботи *Методичні вказівки, технічна література* .

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) *1. Аналіз існуючих методів моніторингу мікроклімату та позиціонування в приміщеннях. Постановка задачі. 2. Розробка структури системи на базі ESP32: апаратна частина, алгоритм Wi-Fi Fingerprinting, база даних. 3. Програмна реалізація: прошивка ESP32, серверний застосунок Python/Flask, веб-інтерфейс. 4. Тестування системи та аналіз результатів.*

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

6. Консультанти розділів роботи

7. Дата видачі завдання *02 лютого 2026р.*.

№ з/п	Назва етапів дипломного проекту (роботи)	Термін виконання етапів проекту (роботи)	Примітка
1	*Аналіз предметної області, огляд існуючих технологій indoor-позиціонування та обґрунтування вибору апаратних компонентів (ESP32, DHT22).*	*Лютий 2026р*	
2	*Проєктування електричної схеми, збірка апаратного вузла та написання програмного забезпечення (C++)*	*Березень 2026р*	
3	*Розробка структури бази даних SQLite створення серверного застосунку (Python/Flask) та реалізація математичного алгоритму Wi-Fi Fingerprinting.*	*Квітень 2026р*	
4	*Створення клієнтського веб-інтерфейсу, проведення натурного та функціонального тестування системи у реальних умовах, оцінка точності локалізації.*	*Травень 2026р*	
5	*Підведення висновків та оформлення роботи*	*Червень 2026р*	

Студент _____ *Коваль О.А.*
(підпис) (прізвище та ініціали)

Керівник роботи _____ *Кропивницька В.Б.*

АНОТАЦІЯ

Актуальність роботи зумовлена необхідністю забезпечення якісного моніторингу мікроклімату у приміщеннях для підвищення комфорту та продуктивності людей. Поставлено задачу розробки вбудованої системи, що здійснює вимірювання температури та вологості повітря з одночасним визначенням місцезнаходження датчика в приміщенні методом Wi-Fi Fingerprinting. Розроблено апаратну частину на базі мікроконтролера ESP32, програмне забезпечення мікроконтролера (Arduino/C++) та серверний додаток (Python/Flask) з веб-інтерфейсом і базою даних SQLite. Реалізовано алгоритм позиціонування, що порівнює поточні RSSI-значення Wi-Fi точок доступу з навченою базою «відбитків». Результати: система здатна визначати поточну кімнату у квартирі з 3 кімнат і відображати показники на веб-панелі в режимі реального часу.

Ключові слова: esp32, моніторинг параметрів повітря, wi-fi fingerprinting, розпізнавання приміщень, flask, sqlite, rssi, мікроклімат, iot, arduino.

ABSTRACT

The relevance of the work is determined by the need for high-quality indoor microclimate monitoring to improve human comfort and productivity. The task of developing an embedded system that measures temperature and air humidity while simultaneously determining the sensor location using the Wi-Fi Fingerprinting method has been set. The hardware part based on the ESP32 microcontroller, microcontroller firmware (Arduino/C++), and a server application (Python/Flask) with a web interface and SQLite database have been developed. A positioning algorithm has been implemented that compares current RSSI values of Wi-Fi access points with a trained fingerprint database. Results: the system is capable of determining the current room among 3 rooms and displaying readings on a web panel in real time.

Key Words: Esp32, Air Parameter Monitoring, Wi-Fi Fingerprinting, Room Recognition, Flask, Sqlite, Rssi, Microclimate, Iot, Arduino.

ЗМІСТ

ПЕРЕЛІК ОСНОВНИХ ПОЗНАЧЕНЬ І СКОРОЧЕНЬ

ESP32	– двоядровий мікроконтролер з інтегрованими модулями Wi-Fi та Bluetooth
Wi-Fi Fingerprinting	– метод позиціонування на основі бібліотеки відбитків Wi-Fi сигналів
RSSI	– Received Signal Strength Indicator – індикатор рівня прийнятого сигналу
IoT	– Internet of Things – Інтернет речей
HTTP	– Hypertext Transfer Protocol – протокол передачі гіпертексту
JSON	– JavaScript Object Notation – текстовий формат обміну даними
SQLite	– вбудована реляційна база даних
API	– Application Programming Interface – інтерфейс програмування застосунків
BSSID	– Basic Service Set Identifier – унікальний ідентифікатор точки доступу Wi-Fi
DHT22	– цифровий датчик температури та вологості
АЦП	– аналогово-цифровий перетворювач
ПЗ	– програмне забезпечення
БД	– база даних

ВСТУП

Якість повітряного середовища у приміщеннях безпосередньо впливає на здоров'я, самопочуття та продуктивність людей, які проводять значну частину часу в закритих просторах. За статистикою ВООЗ, погані умови мікроклімату є причиною низки захворювань дихальних шляхів і зниження когнітивних функцій [2]. У зв'язку з цим системи автоматичного моніторингу параметрів мікроклімату набувають дедалі більшого значення як у промисловому, так і у побутовому секторі.

Традиційні підходи до моніторингу передбачають встановлення стаціонарних датчиків у фіксованих точках. Проте сучасні вимоги диктують необхідність знати не лише значення параметрів, а й прив'язку вимірювань до конкретного місця в приміщенні. Технологія Wi-Fi Fingerprinting дозволяє вирішити цю задачу без залучення додаткової позиціонувальної інфраструктури, використовуючи лише наявну бездротову мережу.

Мікроконтролер ESP32 від Espressif Systems є оптимальним вибором для подібних задач завдяки наявності інтегрованих модулів Wi-Fi і Bluetooth, достатньої обчислювальної потужності, низького енергоспоживання та широкої екосистеми розробки [13]. Поєднання ESP32 з датчиком DHT22 та алгоритмом Wi-Fi Fingerprinting дозволяє створити компактну, автономну та відносно дешеву систему моніторингу мікроклімату з функцією визначення місцеположення[3].

Актуальність теми - Автоматизований моніторинг параметрів мікроклімату з прив'язкою до кімнати є затребуваним у «розумних будинках», медичних закладах, офісних та навчальних приміщеннях. Наявні комерційні рішення є дорогими або не надають API для інтеграції. Відкрита система на базі ESP32 заповнює цю нішу.

Об'єкт дослідження - процес вимірювання та обробки параметрів мікроклімату приміщень у системах IoT.

Предмет дослідження - методи позиціонування на основі Wi-Fi Fingerprinting та апаратно-програмні засоби їх реалізації на платформі ESP32.

Мета роботи - розробити мікроконтролерну систему, що здійснює автоматичний збір даних про температуру та вологість повітря і визначає поточне приміщення методом Wi-Fi Fingerprinting з передачею даних на веб-сервер.

Для досягнення мети необхідно вирішити такі завдання:

- проаналізувати існуючі методи позиціонування в приміщеннях та засоби моніторингу мікроклімату;

- обрати і обґрунтувати апаратну платформу та програмний стек;

- розробити структурну та функціональну схеми системи;

- реалізувати прошивку мікроконтролера ESP32;

- розробити серверний додаток та веб-інтерфейс;- реалізувати та верифікувати алгоритм Wi-Fi Fingerprinting;- протестувати систему в умовах реального приміщення.

Методи дослідження: аналіз літературних джерел та технічної документації (огляд існуючих методів); метод «відбитків» Wi-Fi (Wi-Fi Fingerprinting) для позиціонування; статистичний метод найменших середніх квадратичних відхилень для вибору кімнати; макетування та натурний експеримент для перевірки точності системи.

Практичне значення отриманих результатів полягає у можливості безпосереднього використання розробленої системи у побуті та малому бізнесі для моніторингу мікроклімату. Вихідний код є відкритим і придатним для масштабування та адаптації під інші завдання IoT.

1 АНАЛІЗ МЕТОДІВ МОНІТОРИНГУ МІКРОКЛІМАТУ ТА ПОЗИЦІОНУВАННЯ В ПРИМІЩЕННЯХ

1.1 Параметри мікроклімату приміщень та їх вплив на людину

Поняття «мікроклімат приміщення» охоплює комплекс взаємопов'язаних фізичних та хімічних параметрів: температуру повітря, відносну вологість, швидкість руху повітря, концентрацію діоксиду вуглецю (CO_2), вміст дрібнодисперсних частинок (PM2.5/PM10) та концентрацію чадного газу (CO). Сукупність цих факторів безпосередньо визначає якість повітряного середовища і самопочуття людей, що перебувають у приміщенні.

Нормативна база для виробничих приміщень. Основним документом, що регламентує мікрокліматичні умови на виробництві в Україні, є ДСН 3.3.6.042-99 «Санітарні норми мікроклімату виробничих приміщень» (постанова МОЗ України від 01.12.1999 № 42) . Для операторської діяльності та роботи в залах обчислювальної техніки документ встановлює оптимальні умови: температура повітря 22-24°C, відносна вологість 60-40%, швидкість руху повітря не більше 0,1 м/с [1]. Норми диференційовані залежно від важкості праці та пори року; допустимі значення передбачають певні відхилення від оптимальних за умов технологічної неможливості їх забезпечення.

Нормативна база для житлових приміщень. Для квартир та житлових будинків застосовуються інші нормативні документи. Головним з них є ДБН В.2.5-67:2013 «Опалення, вентиляція та кондиціонування» (Мінрегіон України), який встановлює нормативні параметри мікроклімату для житлових, громадських та адміністративно-побутових будівель, наводячи таблиці оптимальних і допустимих значень температури, вологості та швидкості руху повітря [2]. Зокрема, для житлових приміщень встановлено температуру 18-22°C (залежно від призначення кімнати) та відносну вологість 30-60%. При цьому у теплий період року параметри мікроклімату не нормуються для житлових будинків, крім приміщень з системами кондиціонування та охолодження

повітря [2]. Поряд із ДБН В.2.5-67:2013 діють Державні санітарні правила та норми (ДСанПіН), що регулюють гігієнічні вимоги до утримання житла: норма швидкості руху повітря для житлових приміщень становить 0,15-0,2 м/с у холодний та до 0,3 м/с у теплий період, вологість - 40-60% [3].

Інші параметри якості повітря. Сучасні нормативні підходи враховують не лише температурно-вологісний режим, а й хімічний склад повітря. Відповідно до ДБН В.2.5-67:2013, встановлена класифікація за рівнем концентрації CO_2: оптимальні умови (до 800 ppm) забезпечують найкраще самопочуття та продуктивність, допустимі (до 1000 ppm) вимагають напруження організму та адаптації [2]. За гігієнічними нормативами МОЗ України, повітря населених людьми приміщень вважається чистим, якщо концентрація CO_2 не перевищує гранично допустимих 0,1% (1000 ppm) за Флюге [4]. Щодо дрібнодисперсних частинок, рекомендації ВООЗ встановлюють для PM2.5 середньодобовий рівень 25 мкг/м³; у приміщеннях зазвичай також контролюють рівень формальдегіду та частинки PM10 .

Варто підкреслити, що динаміка зміни зазначених параметрів у закритих просторах має комплексний і взаємопов'язаний характер. Наприклад, коливання температури суттєво впливають на суб'єктивне теплове сприйняття вологості людиною, а недостатня вентиляція приміщення призводить до одночасного накопичення як вуглекислого газу, так і водяної пари внаслідок життєдіяльності. Це зумовлює необхідність переходу від епізодичних ручних замірів до безперервного інструментального моніторингу, що здатний фіксувати флуктуації фізико-хімічних показників повітряного середовища у фоновому режимі в реальному часі.

Наведені вище нормативні вимоги стосуються різних типів приміщень і відрізняються як за діапазонами допустимих значень, так і за переліком контрольованих параметрів. Для зручності порівняння ключові показники зведено в таблицю 1.1. Слід зазначити, що для житлових приміщень нормування є дещо менш жорстким ніж для виробничих.

Таблиця 1.1 – Нормовані параметри мікроклімату для різних типів приміщень

Параметр	Виробниче приміщення (ДСН 3.3.6.042-99)	Житлове приміщення (ДБН В.2.5-67:2013 / ДСанПіН)
Температура, холодний період	22–24°C (опт.); до 28°C (доп.)	18–22°C (опт.)
Температура, теплий період	23–25°C (опт.)	не нормується (без кондиціонування)
Відносна вологість	40–60%	30–60% (опт.)
Швидкість руху повітря	0,1–0,4 м/с	0,15–0,2 м/с (до 0,3 м/с влітку)
CO_2 (ГДК)	до 1000 ppm	700–1000 ppm
PM2.5	25 мкг/м³ (добова, ВООЗ)	25 мкг/м³ (добова, ВООЗ)

Вплив відхилень від норм на здоров'я. Порушення будь-якого з наведених параметрів негативно позначається на самопочутті та продуктивності. Підвищена вологість (понад 70%) сприяє розвитку цвілі та алергічних реакцій, занадто сухе повітря (нижче 30%) подразнює слизові оболонки та підвищує ризик респіраторних захворювань [3]. При концентрації CO_2 понад 1000 ppm у людей виникають головний біль, втома та зниження концентрації уваги [4]. Перевищення допустимого вмісту PM2.5 збільшує ризик серцево-судинних і респіраторних захворювань при тривалому впливі. Температура поза межами комфортних значень знижує концентрацію уваги та продуктивність праці на 10-15% .

Обґрунтування фокусу дослідженя. Незважаючи на багатопараметричний характер мікроклімату, саме температура та відносна вологість є найбільш пріоритетними показниками для повсякденного моніторингу в житловому приміщенні. По-перше, ці параметри безпосередньо визначають теплове відчуття людини і потребують постійного контролю протягом усього року. По-друге, для їх вимірювання існують доступні, надійні та недорогі цифрові датчики, зокрема DHT22, що забезпечують достатню для побутового застосування точність (~0,5°C, ~2% ВВ). По-третє, саме температура та вологість є тими

параметрами, якими мешканці можуть практично управляти засобами опалення, вентиляції або зволоження повітря. Моніторинг решти параметрів (CO_2, PM2.5, CO) потребує значно дорожчих сенсорів і складніших систем реагування, що виходить за рамки поставленого завдання. Таким чином, постійний автоматизований контроль температури та вологості є необхідною і практично досяжною умовою забезпечення комфортного та здорового середовища в квартирі.

1.2 Огляд та порівняльний аналіз наявних рішень моніторингу мікроклімату

Компанія Netatmo пропонує розумну метеостанцію з хмарним сервісом та мобільним додатком. Пристрій вимірює температуру, вологість, CO_2 та рівень шуму і передає дані через Wi-Fi на сервери Netatmo. Перевагою є готовий продукт з красивим інтерфейсом та інтеграцією з Apple HomeKit та Amazon Alexa. Недоліком є залежність від хмарного сервісу третьої сторони: при відключенні серверів Netatmo система перестає функціонувати, а API обмежений та підлягає змінам без попередження [3]. Вартість базового комплекту - близько $100.

Система Xiaomi Mi Home включає датчики температури та вологості LYWSD03MMC з підключенням через Bluetooth Low Energy до шлюзу Xiaomi Gateway. Перевагами є низька вартість датчика (~$5), компактний форм-фактор та тривалий заряд батареї (до 12 місяців). Недоліками є закрита екосистема, відсутність прямого Wi-Fi і залежність від хмари Mi Home. Інтеграція з Home Assistant можлива через сторонні бібліотеки, але нестабільна.

Продукти компанії Sensirion серії SHT відрізняються промисловою точністю (~0.2°C, ~1.5% ВВ) та широким діапазоном робочих температур. Вони використовуються у медичній, лабораторній та промисловій апаратурі.

Однак ці рішення вимагають значно складнішого апаратного інтегрування, коштують у 5-10 разів дорожче за DHT22 і не мають вбудованої мережевої підтримки.

У спільноті розробників IoT популярними є рішення на базі ESP8266 (NodeMCU), ESP32 та Arduino Uno з датчиками DHT11/DHT22 або BME280. Типова реалізація включає мікроконтролер, датчик, OLED-дисплей та передачу даних на локальний сервер або в хмару. Найпоширенішими проектами є: ESPHome - компонентна система для Home Assistant; Tasmota - відкрита прошивка для ESP-пристроїв з підтримкою MQTT; самостійні Arduino-скетчі з Blynk або ThingSpeak.

Переваги DIY-підходу: повна відкритість та кастомізація; низька вартість (ESP32 + DHT22 = 300 грн); відсутність залежності від сторонніх сервісів; можливість локального розгортання. Недоліком є те, що більшість існуючих DIY-проектів не включає функції автоматичного визначення місцезнаходження датчика в приміщенні, що є ключовою відмінністю даної роботи [4].

ThingSpeak - хмарна платформа для IoT від MathWorks, що дозволяє безкоштовно зберігати та візуалізувати до 3 мільйонів повідомлень на рік. Підтримує MATLAB-аналіз даних безпосередньо в хмарі. Обмеженням є максимальна частота оновлення 15 секунд для безкоштовного тарифу та залежність від зовнішнього сервісу.

Home Assistant - відкрита платформа автоматизації «розумного будинку» з можливістю локального розгортання. Підтримує тисячі інтеграцій, включаючи ESP32 через ESPHome. Перевагою є повна локальна робота без хмари, активна спільнота та регулярні оновлення. Недоліком є висока складність початкового налаштування для непідготовленого користувача.

Blynk - мобільна платформа IoT з drag-and-drop конструктором інтерфейсу. Надає готові віджети (кнопки, графіки, індикатори) без написання фронтенд-коду. Blynk 2.0 перейшов на підписну модель, що обмежує безкоштовне використання п'ятьма пристроями.

Таким чином, проведений огляд інженерних рішень демонструє чіткий поділ між закритими комерційними екосистемами та відкритими DIY-платформами. Комерційні продукти пропонують високу надійність «з коробки», але обмежують користувача залежністю від хмарних серверів і закритим вихідним кодом. Водночас DIY-рішення надають повну автономність, проте вимагають значних зусиль для інтеграції нестандартних функцій, зокрема локального позиціонування. Для систематизації та наочного зіставлення ключових характеристик розглянутих систем доцільно сформувати порівняльну таблицю.

Таблиця 1.2 - Порівняльний аналіз наявних рішень моніторингу мікроклімату

Критерій	Netatmo	Xiaomi Mi Home	Sensirion SHT	DIY ESP32 (DIY)	Дана робота
Вартість ($)	~100	~10–30	~50–200	~5	~5
Точність температури	~0.3°C	~0.5°C	~0.2°C	~0.5°C (DHT22)	~0.5°C (DHT22)
Локальна робота	Ні (хмара)	Частково	Так	Так	Так
Відкритий API	Обмежений	Ні	Так	Так	Так
Визначення кімнати	Ні	Ні	Ні	Зазвичай ні	Так (Wi-Fi FP)
Простота розгортання	Висока	Висока	Низька	Середня	Середня
Масштабованість	Обмежена	Обмежена	Так	Так	Так

Аналіз таблиці 1.2 показує, що жодне з існуючих рішень не поєднує низьку вартість, локальну роботу, відкритий API та автоматичне визначення кімнати. Дана робота заповнює цю нішу, пропонуючи повністю відкрите рішення на базі ESP32 з підтримкою Wi-Fi Fingerprinting.

1.3 Методи визначення місцезнаходження в приміщеннях

Системи позиціонування в приміщеннях (Indoor Positioning Systems, IPS) поділяються на кілька категорій залежно від використовуваної фізичної основи та необхідної інфраструктури.

Ультраширокосмугові системи (UWB) забезпечують точність позиціонування 10-30 см завдяки вимірюванню часу прольоту коротких імпульсів між мітками та якорями. Прикладами є Pozyx, DecaWave DWM1000, Apple AirTag. Вимагають розгортання мінімум 3-4 спеціалізованих якорів у кожному приміщенні та коштують від $50 на вузол [5].

Bluetooth Low Energy (BLE) Beacons - маяки iBeacon або Eddystone передають рекламні пакети, за рівнем сигналу яких можна оцінити відстань. Точність 1-3 м, але вимагає встановлення маяків у кожній зоні та наявності BLE-сканера. Використовується у великих торгових центрах та музеях [6].

Ультразвукове позиціонування засноване на вимірюванні часу поширення ультразвукового сигналу між передавачами та приймачами. Забезпечує точність до 1-5 см, але вразливе до відлунь, перешкод та температурних змін. Практично не використовується у побутових застосуваннях [7].

Wi-Fi Fingerprinting (метод «відбитків Wi-Fi») ґрунтується на тому, що в кожній точці приміщення спостерігається унікальний «відбиток» - набір значень RSSI від видимих точок доступу. На першому етапі (навчання) збираються зразки RSSI для кожної референсної позиції (кімнати). На другому (визначення) поточний набір RSSI порівнюється з базою, і обирається найближча позиція за метрикою близькості [8,10].

Переваги Wi-Fi Fingerprinting: використовує наявну Wi-Fi інфраструктуру без додаткового обладнання; забезпечує роздільну здатність на рівні кімнат (3-5 м); реалізується на ESP32 вартістю ~$5. Недоліки: точність залежить від стабільності Wi-Fi середовища; вимагає повторного навчання при зміні топології мережі або розташування меблів; часова мінливість RSSI ускладнює точне визначення позиції [9,10].

Порівняно з UWB та BLE, Wi-Fi Fingerprinting є оптимальним рішенням для задачі визначення кімнати, оскільки не вимагає додаткової інфраструктури і реалізується на ESP32, що вже має вбудований Wi-Fi модуль.

1.4 Постановка задачі

На основі проведеного аналізу сформульована така задача: розробити систему, що включає вузол на ESP32 з датчиком DHT22, здатний з заданою частотою (1 раз на 15 секунд) вимірювати температуру та вологість повітря і сканувати Wi-Fi мережі, а також передавати отримані дані на локальний HTTP-сервер (Python/Flask). Сервер має зберігати «відбитки» Wi-Fi для кожної кімнати у базі даних SQLite, реалізовувати алгоритм визначення поточної кімнати методом Wi-Fi Fingerprinting та надавати веб-інтерфейс для відображення поточних показників і управління навчанням системи.

Система повинна задовольняти таким вимогам: час відгуку веб-інтерфейсу - не більше 2 с; точність визначення кімнати - не менше 70% при стабільному Wi-Fi оточенні; вартість апаратної частини - не більше 500 грн; споживання струму вузлом - не більше 250 мА.

2 РОЗРОБКА СТРУКТУРИ ТА АЛГОРИТМІВ МІКРОКОНТРОЛЕРНОЇ СИСТЕМИ

2.1 Розробка загальної архітектури системи

Розроблена система складається з трьох основних компонентів: вузла збору даних на базі ESP32, серверного застосунку на Python/Flask та бази даних SQLite. Взаємодія між компонентами здійснюється за допомогою протоколу HTTP з передачею даних у форматі JSON.

Вузол ESP32 виконує дві функції: вимірювання температури та вологості за допомогою датчика DHT22 і сканування навколишніх Wi-Fi мереж для збору RSSI-значень. Зібрані дані упаковуються у JSON-об'єкт та відправляються на сервер методом HTTP POST.

Серверний застосунок приймає дані від вузла, зберігає або оновлює базу «відбитків» під час фази навчання і визначає поточну кімнату в режимі роботи. Веб-інтерфейс відображає актуальні показники мікроклімату та поточну кімнату.

Для наочного відображення описаних вище зв'язків було розроблено логічну схему системи (рис. 2.1). Головною ідеєю такої архітектури стало чітке розмежування відповідальності між рівнями. Замість того, щоб перевантажувати периферійний вузол обчисленнями, було вирішено делегувати всю ресурсомістку математику алгоритму позиціонування та операції з пам'яттю на потужніший серверний рівень.

Як видно зі схеми, інформаційний потік має чітку спрямованість. Сирі показники мікроклімату та радіочастотні відбитки транслюються від апаратної частини до сервера, де відразу обробляються або записуються у БД, залежно від поточного режиму. При цьому веб-інтерфейс повністю абстрагований від низькорівневого обміну датчиків. Він слугує лише вітриною для готових результатів та інструментом керування.

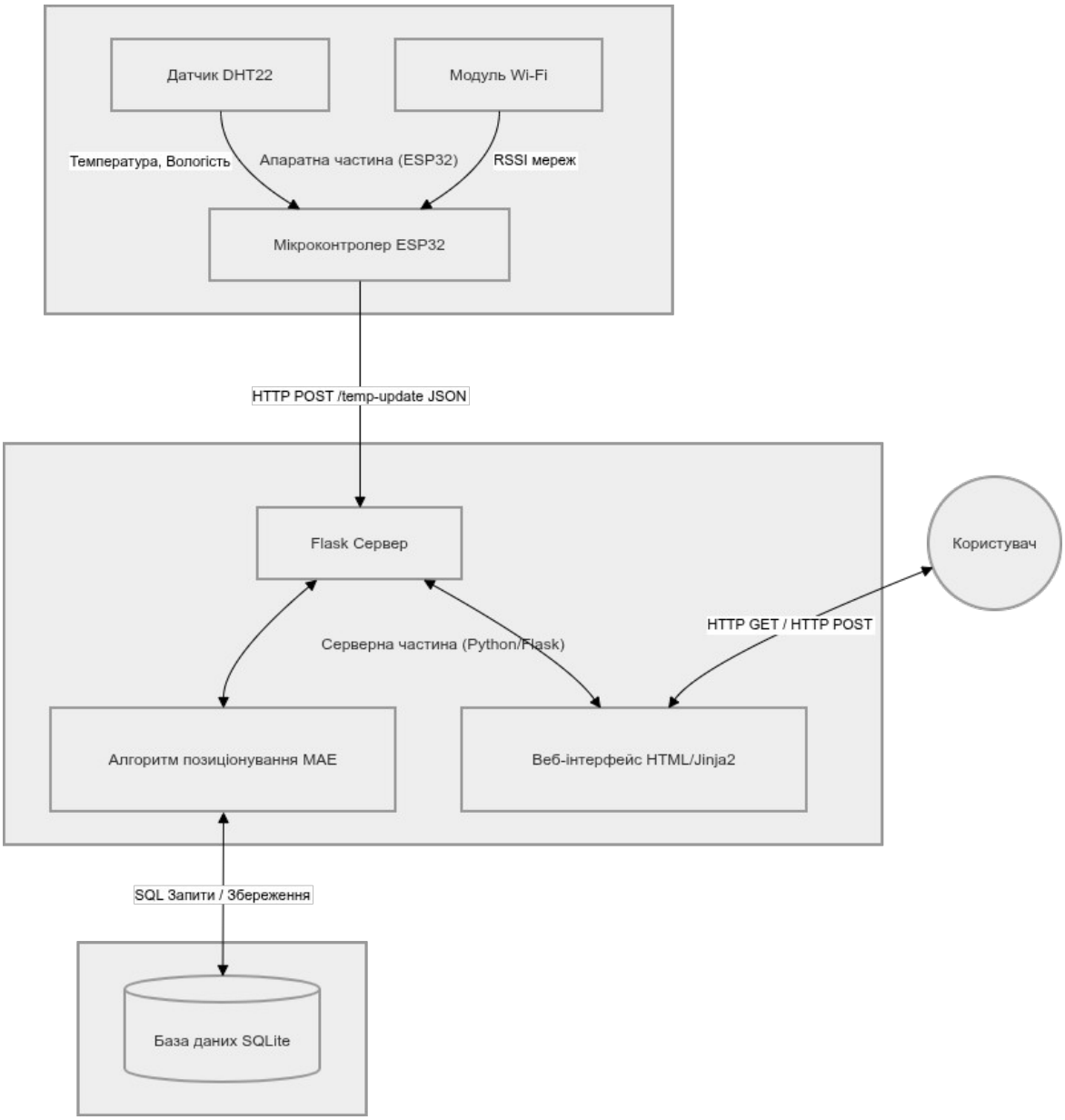

Рисунок 2.1 — Логічна схема системи

Такий модульний підхід робить систему гнучкою: він дозволяє в майбутньому змінювати алгоритми обробки на сервері чи структуру бази даних без жодного втручання у прошивку самого мікроконтролера. Більш детальний опис апаратної реалізації, логіки алгоритму та структури зберігання даних розглядається у наступних підрозділах роботи.

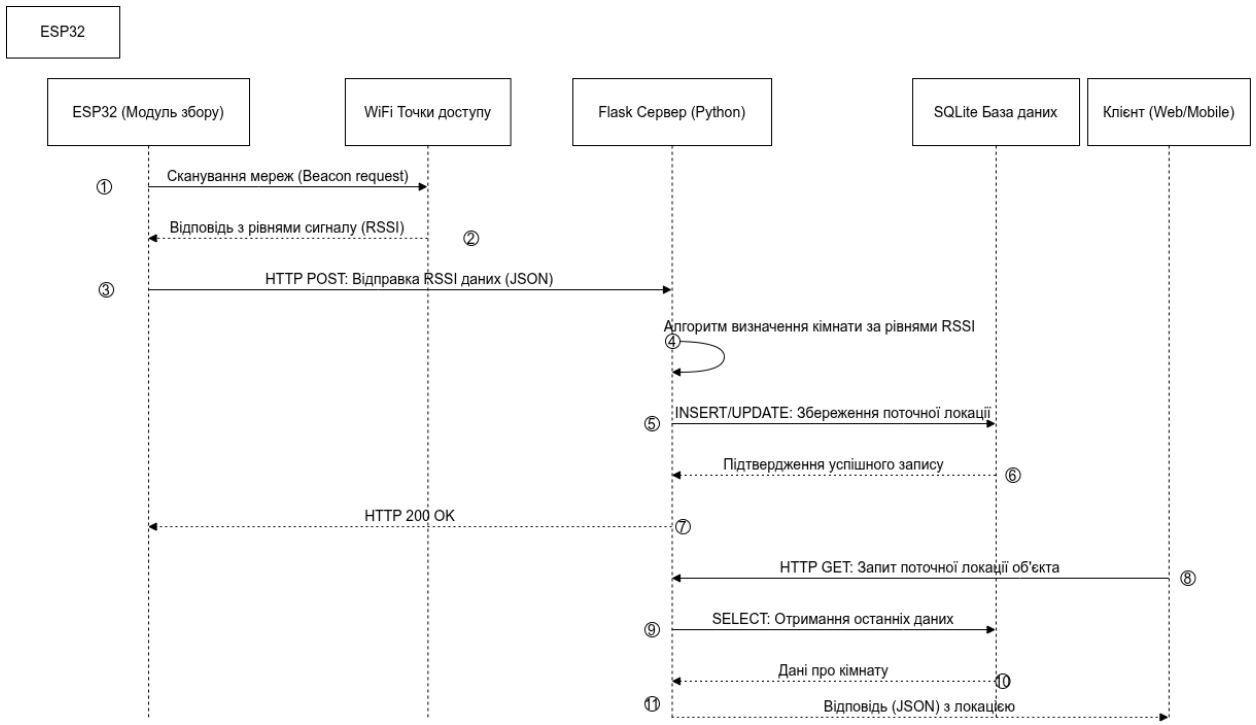

Рисунок 2.2 - Діаграма послідовності системи позиціонування Smart Home

На рисунку 2.2 наведено діаграму послідовностей, яка зображає логіку інформаційної взаємодії компонентів розробленої системи позиціонування об'єктів у приміщенні.

2.2 Проєктування сценаріїв взаємодії користувача та вузла ESP32 із системою

Будь-який процес проєктування програмного забезпечення вимагає чіткого визначення функціональних меж системи та її поведінки у відповідь на зовнішні запити. У контексті розроблюваної платформи позиціонування та моніторингу мікроклімату критично важливо розмежувати ролі ініціаторів цих подій. З одного боку, серверна частина має забезпечувати зручний інтерфейс для людини, яка аналізує фінальні результати та керує режимом збору еталонних даних. З іншого боку, невід'ємним компонентом комплексу є мікроконтролер, який виступає автономним агентом і генерує безперервний потік

телеметрії. Для формалізації цієї асинхронної взаємодії та наочного представлення вимог на етапі логічного проєктування доцільно використовувати методологію об'єктно-орієнтованого моделювання UML.

Для безпосереднього опису функціональних вимог та сценаріїв взаємодії із розробленим сервером було побудовано діаграму прецедентів (Use Case), яку представлено на рисунку 2.3.

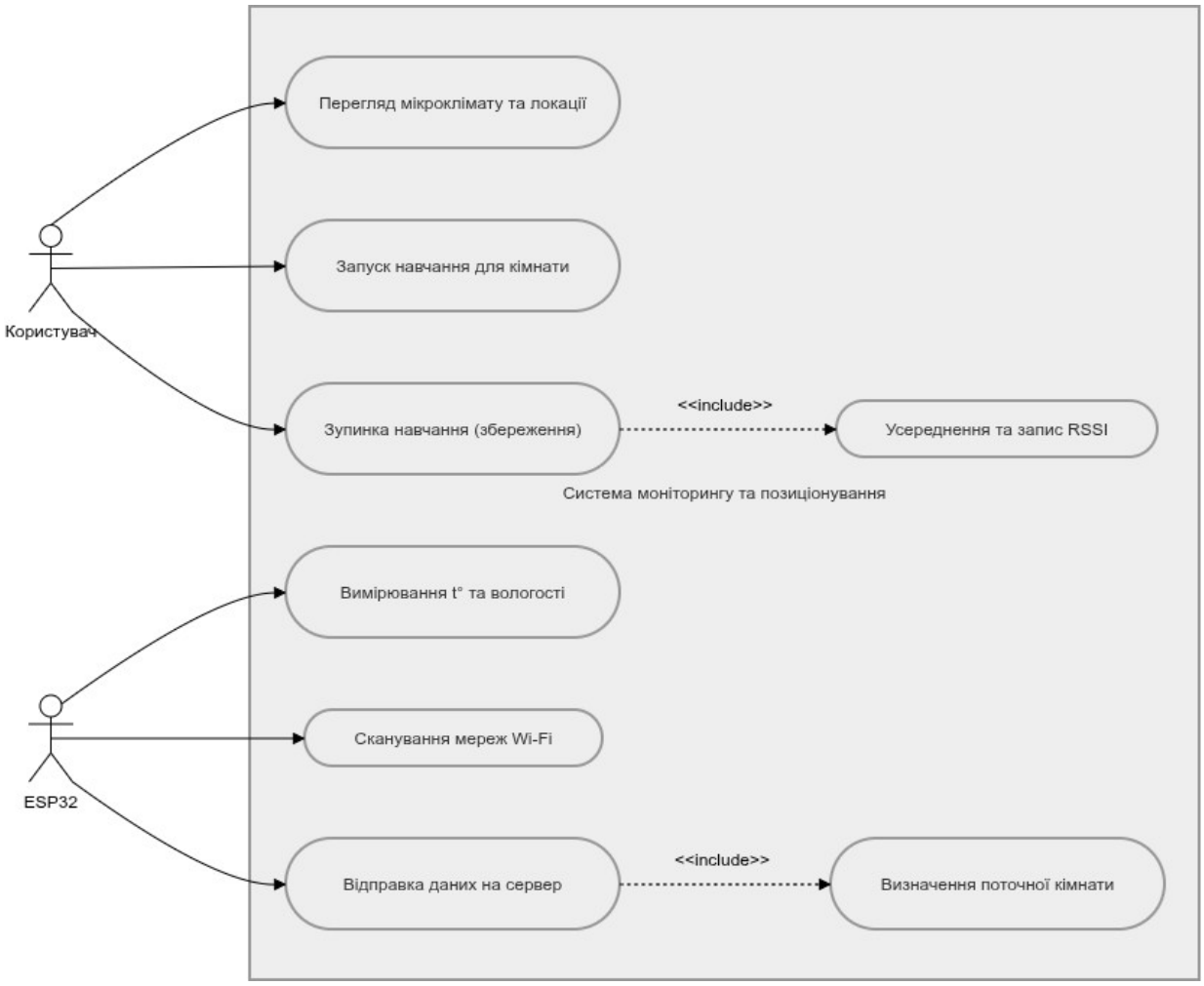

Рисунок 2.3 — Діаграма прецедентів (use case) мікроконтролерної системи

Побудована модель містить двох незалежних акторів: Користувача та Вузол ESP32. Такий розподіл відображає реальну природу системи, де людина взаємодіє з нею епізодично через вебінтерфейс, тоді як апаратний вузол функціонує автономно і безперервно у фоновому режимі.

Актор Користувач реалізує три основні прецеденти. Перший прецедент полягає у перегляді поточних показників мікроклімату та визначеної кімнати на головній сторінці інформаційної панелі. Другий прецедент відповідає за запуск навчання для обраної кімнати, що переводить сервер у режим накопичення RSSI-відбитків шляхом звернення до відповідного програмного маршруту. Третій прецедент ініціює зупинку навчання. Цей прецедент реалізує залежність відносно прихованого серверного механізму: після отримання команди зупинки сервер обов'язково виконує математичне усереднення накопичених RSSI-значень по кожній точці доступу та зберігає результати до таблиці бази даних SQLite. Оскільки без цього кроку процес навчання не матиме практичного результату, дана залежність є строго обов'язковою.

Актор Вузол ESP32 реалізує два власні прецеденти. Базовим прецедентом є надсилання зібраних даних на сервер, що виконується автоматично через задані інтервали часу незалежно від дій користувача. Цей процес включає зчитування температури та вологості з датчика DHT22, сканування доступних Wi-Fi мереж із застосуванням фільтрації слабких сигналів, формування JSON-пакета та його надсилання через HTTP-запит. Другий прецедент апаратного вузла стосується визначення поточної кімнати на стороні сервера. Ця дія реалізується через обов'язкову залежність від базового прецеденту передачі даних і виконується автоматично при кожному успішному отриманні пакета у штатному режимі роботи, проте повністю ігнорується системою під час активного режиму навчання.

Такий логічний розподіл прецедентів між акторами наочно демонструє ключовий архітектурний принцип розробленого комплексу: вся важка обчислювальна логіка зосереджена на стороні сервера, тоді як апаратний модуль виконує виключно функції збору та передачі сирих даних, залишаючи користувачеві зручний інтерфейс для спостереження та керування алгоритмами.

2.3 Проєктування апаратної частини на базі ESP32 та DHT22

Основою апаратної частини системи є мікроконтролерна плата розробника на базі модуля ESP32-WROOM-32U від компанії Espressif Systems (рисунок 2.4). Серцем модуля є двоядерний процесор Xtensa 32-bit LX6 з тактовою частотою до 240 МГц. Ключовою особливістю та перевагою саме модифікації «U» для даної задачі є відсутність вбудованої на плату (PCB) антени та наявність високочастотного роз'єму U.FL (IPEX) для підключення зовнішньої антени.

Використання зовнішньої антени є критично важливим апаратним рішенням для систем Wi-Fi Fingerprinting. Це дозволяє мінімізувати вплив електромагнітних завад від самої мікроконтролерної плати, збільшити радіус впевненого прийому слабких сигналів (до -90 дБм) та значно підвищити стабільність показників RSSI, що прямо впливає на загальну точність класифікації приміщень. Модуль підтримує стандарти бездротового зв'язку Wi-Fi 802.11 b/g/n, має 34 виводи загального призначення (GPIO) та підтримує апаратні інтерфейси SPI, I2C, UART, PWM і 18-канальний 12-бітний АЦП [10].

Рисунок 2.4 — Мікроконтролер esp 32

Детальніше про піни мікроконтролера розписано на рисунку 2.5.

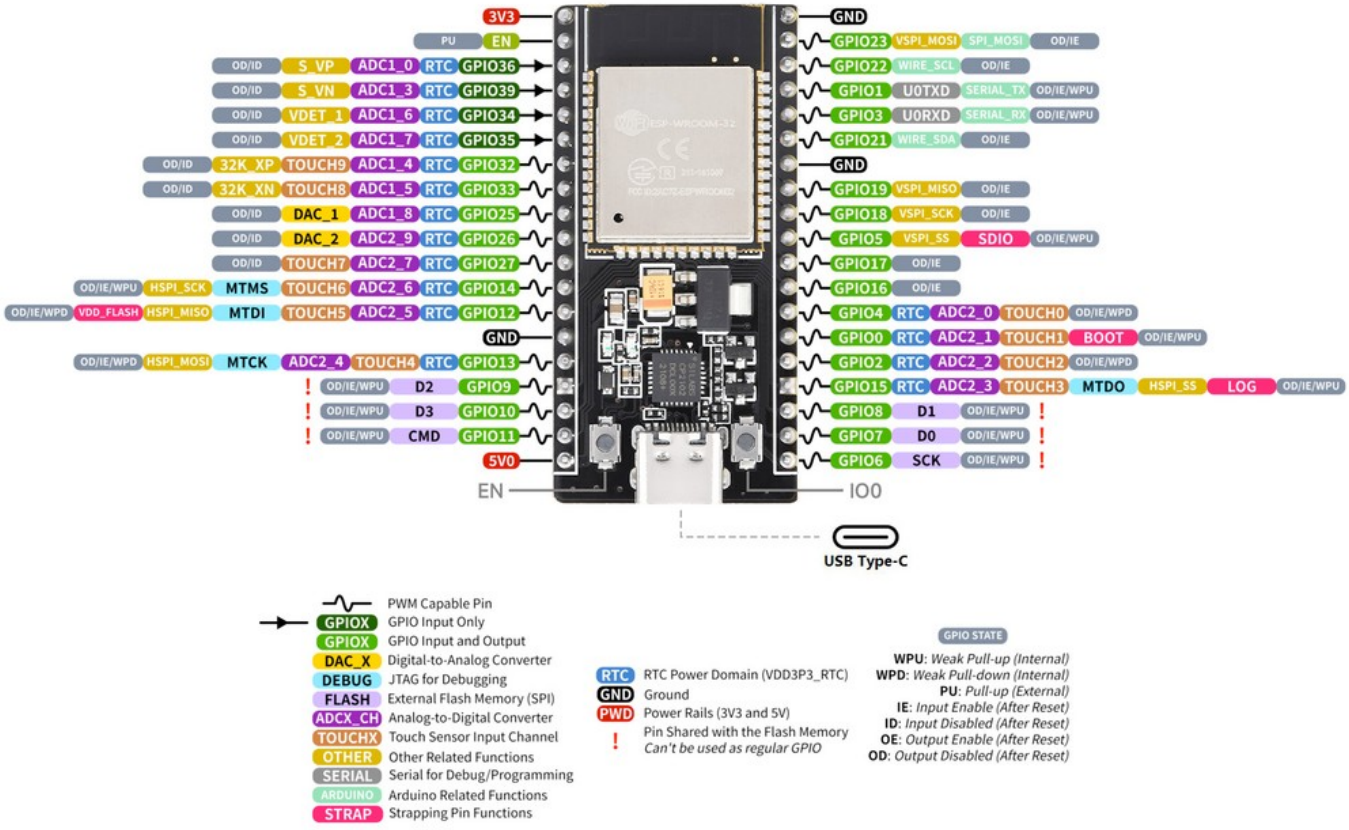

Рисунок 2.5 — Графічне представлення пінів esp32

Для реалізації апаратної частини системи ключовими вимогами до мікроконтролера були: підтримка сканування Wi-Fi мереж у режимі клієнта (STA) без розриву поточного з'єднання із сервером, наявність інтерфейсів вводу-виводу для підключення периферійних пристроїв, а також підтримка режимів зниженого енергоспоживання (Light Sleep) з можливістю апаратного пробудження за таймером для забезпечення енергоефективної циклічної роботи пристрою.

Як основний інструмент моніторингу мікроклімату обрано модуль на базі сенсора AM2302 (аналог DHT22) - цифрового датчика температури та вологості з однопровідним послідовним інтерфейсом (рисунок 2.6). Його технічні характеристики повністю задовольняють вимоги розроблюваної системи: діапазон вимірювання температури становить від -40 до +80°C з точністю ~0.5°C; діапазон вимірювання відносної вологості - від 0 до 100% з

точністю ~2-5%; робоча напруга живлення складає 3.3-5.5 В. Датчик характеризується часом відповіді близько 2 секунд і формує 40-бітне слово даних, що містить 16 біт значень температури, 16 біт значень вологості та 8 біт контрольної суми. Такий формат передачі значно спрощує цифрову обробку сигналу та виключає похибки, характерні для аналого-цифрового перетворення [11].

Рисунок 2.6 — модуль DHT22

Для з'єднання датчика мікроклімату із мікроконтролером ESP32 використовується контакт GPIO4. У розробленій системі застосовано готовий апаратний модуль сенсора, який уже містить інтегрований на платі підтягуючий резистор лінії даних. Це дозволяє підключати модуль безпосередньо до портів мікроконтролера без використання додаткових зовнішніх електронних компонентів, що спрощує загальну принципову схему та підвищує надійність вузла. Програмна реалізація протоколу обміну даними здійснюється за допомогою спеціалізованої бібліотеки DHT від компанії Adafruit.

На основі обраних електронних компонентів та визначеної логіки портів вводу-виводу було спроєктовано загальну електричну принципову схему апаратного вузла, яка наочно демонструє маршрутизацію ліній живлення та підключення сигнального контакту датчика (рис. 2.7).

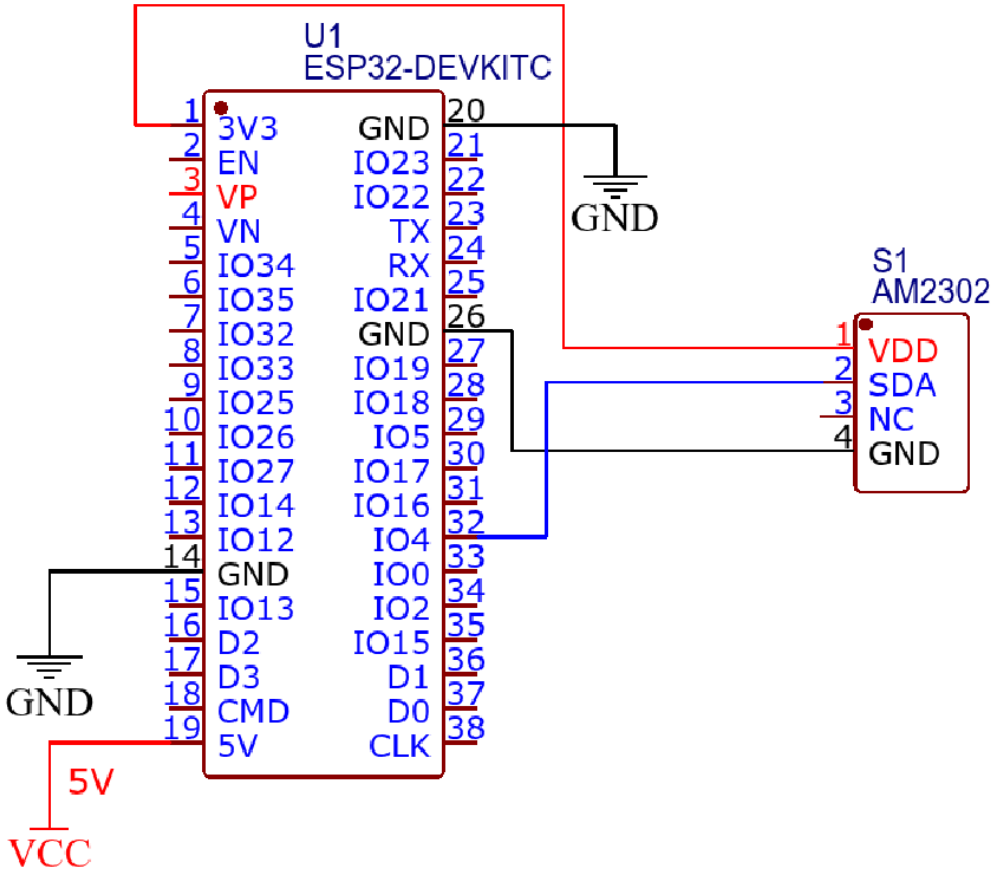

Рисунок 2.7 — Схема підключення мікроконтролерної системи

Таблиця 2.1 - Перелік елементів до схеми електричної принципової

Поз. позначення	Найменування	Кількість	Примітка
U1	Мікроконтролерний модуль ESP32-WROOM-32 (DevKitC)	1	
S1	Модуль датчика температури та вологості AM2302	1	З вбудованим підтягуючим резистором
USB1	Гніздо Micro-USB тип B (для монтажу на плату)	1	Роз'єм живлення 5 В

Сформована апаратна база у вигляді мікроконтролера ESP32 та модуля мікроклімату AM2302 утворює закінчений апаратний вузол системи моніторингу. Затверджена схема підключення та перелік елементів є достатніми для фізичного макетування пристрою, що дозволяє перейти до проєктування та написання мікропрограмного забезпечення для керування збором та передачею даних.

2.4 Програмна реалізація алгоритму Wi-Fi Fingerprinting

У фазі навчання оператор розміщує пристрій у цільовій кімнаті та ініціює процедуру збору еталонних даних через веб-інтерфейс. Протягом заданого проміжку часу мікроконтролер виконує серію ітеративних сканувань радіоефіру та передає пакети виявлених точок доступу (BSSID та RSSI) на сервер. На відміну від базових систем класифікації, які оперують лише середнім арифметичним, реалізований алгоритм додатково розраховує дисперсію та стандартне відхилення сигналу для кожної точки доступу. Це дозволяє враховувати стабільність радіосередовища під час формування еталонного "відбитка" приміщення. Етапи роботи системи зображено на рисунках 2.8, 2.9.

Математично середнє значення рівня сигналу $\overline{RSSI}(k,i)$ та його σ_k стандартне відхилення *i*-тої точки доступу у кімнаті *k* обчислюються за формулами:

$$\overline{RSSI_k}(i)=\frac{1}{N}\sum_{j=1}^{N} RSSI_j(i) \tag{2.1}$$

$$\sigma_k=\sqrt{\frac{1}{N}\sum_{j=1}^{N}\left(RSSI_j(i)-\overline{RSSI_k}(i)\right)^2} \tag{2.2}$$

де $\overline{RSSI_k}(i)$ – середнє значення сили сигналу (RSSI) для *i*-тої точки доступу у кімнаті *k* ;

N - загальна кількість сканувань під час фази навчання;

j - порядковий номер (індекс) поточного сканування;

$RSSI_j(i)$ - значення RSSI від *i*-тої точки доступу у *j*-тому скануванні;

i - порядковий номер (індекс) точки доступу;

k - ідентифікатор (номер) цільової кімнати.

У робочому режимі (фаза класифікації) сервер отримує поточний вектор RSSI від вузла та обчислює міру "відстані" (несхожості) між поточним станом

радіоефіру та збереженими еталонними відбитками кожної кімнати з бази даних. Для підвищення точності локалізації реалізовано модифікований алгоритм на основі зваженої квадратичної відстані зі штрафуванням за зниклі мережі.

Цільова функція оцінки (Score) *D(k)* для еталонної кімнати *k* обчислюється за формулою:

$$D(k)=\frac{\sum_{i=1}^{M} W_k(i)\cdot \Delta RSSI(i)^2}{\sum_{i=1}^{M} W_k(i)} \qquad (2.3)$$

де *D(k)* - обчислена відстань (міра розбіжності) між поточним вектором радіосигналів і еталонним відбитком кімнати *k*;

M - загальна кількість еталонних точок доступу, що були зафіксовані для кімнати *k* під час навчання;

Δ*RSSI(i)* - різниця між поточним виміряним рівнем сигналу та еталонним (очікуваним) середнім рівнем сигналу для і-тої точки доступу. Згідно з твоїм алгоритмом, це значення також включає розрахований штраф, якщо еталонна мережа відсутня у поточному радіоефірі.

i - порядковий номер (індекс) спільної точки доступу;

$W_k(i)$ - ваговий коефіцієнт надійності і-тої точки доступу, який обернено пропорційний до її стандартного відхилення:

$$W_k(i)=\frac{1}{\sigma_k(i)+1} \qquad (2.4)$$

де σ_k - стандартне відхилення рівня радіосигналу для і-тої точки доступу в кімнаті k. Цей параметр розраховується під час навчання і показує, наскільки сильно коливався сигнал цієї конкретної Wi-Fi мережі (її стабільність).

Різниця рівнів сигналу ΔRSSI(i) розраховується з урахуванням наявності мережі у поточному скануванні. Якщо і-та еталонна мережа присутня у по-

точному радіоефірі, береться абсолютна різниця між еталонним і поточним значенням. Якщо ж мережа відсутня (зникла з радіовидимості), алгоритм застосовує "штраф", припускаючи, що рівень її сигналу впав до апаратного мінімуму (-100 дБм):

$$\Delta RSSI(i)=\begin{cases}\left|RSSI_{curr}(i)-\overline{RSSI_k}(i),\right| & Якщо\ BSSID\ присутній\\ -100-\overline{RSSI_k}(i), & Якщо\ BSSID\ відсутній\end{cases} \quad (2.5)$$

Для захисту від аномальних класифікацій запроваджено поріг покриття, якщо відсоток виявлених еталонних мереж становить менше 30%, кімната автоматично виключається з вибірки. З-поміж валідних кандидатів обирається кімната з мінімальним значенням зваженого відхилення. Цей алгоритм базується на методі зваженої відстані, описаному в [10,22].

Окрім математичної класифікації одиничного вектора, система використовує метод "ковзного вікна" розміром 3 ітерації [22]. Остаточне рішення про зміну локації приймається на основі найбільш часто зустрічуваної кімнати в історії останніх вимірювань, що ефективно нівелює "глітчі" радіосигналу.

Рисунок 2.8 - Блок схема алгоритму запам’ятовування кімнати

Процес ініціюється оператором, який через веб-інтерфейс обирає цільову кімнату. Відбувається переадресація на маршрут /start_training, що переводить сервер у стан активного запису (глобальна змінна training ініціалізується як True). Далі алгоритм переходить у циклічну фазу буферизації. Апаратний модуль безперервно здійснює HTTP POST-запити, передаючи поточні масиви BSSID та RSSI. Сервер накопичує ці пакети у тимчасовій структурі даних. Цикл збору триває до ініціації команди /stop_training. Після її отримання циклічний прийом завершується, накопичений масив даних проходить етап усереднення та розрахунку стандартного відхилення, після чого сформований еталонний відбиток зберігається у реляційну базу даних SQLite, і алгоритм припиняє свою роботу.

Рисунок 2.9 — Блок схема алгоритму роботи програми

Робочий цикл ініціюється мікроконтролером, який передає масив зібраних телеметричних даних (показники мікроклімату та вектори RSSI). Сервер приймає цей пакет через кінцеву точку /temp-update. Далі відбувається виклик функції compute_room_probabilities, яка виконує ітеративне порівняння поточного радіоефіру з еталонними записами кожної кімнати у базі даних (з урахуванням штрафних коефіцієнтів та вагових функцій). Сире значення класифікації додається до програмної черги історії вимірювань (ковзного вікна). Фінальний результат визначається методом статистичного голосування за останні три ітерації і записується у глобальний словник станів сервера. На завершальному етапі оновлена інформація стає доступною для клієнтського веб-інтерфейсу (через фонове оновлення), після чого ітерація обробки успішно завершується.

2.5 Проєктування структури бази даних SQLite

Для зберігання еталонних радіовідбитків (Wi-Fi Fingerprints) та інформації про приміщення у розробленій системі позиціонування використовується система керування базами даних SQLite. Це компактна вбудована СКБД, яка працює локально і не вимагає окремого серверного процесу чи складного адміністрування. Усі робочі дані проєкту зберігаються у єдиному файлі безпосередньо на диску пристрою.

Таке архітектурне рішення зумовлене специфікою роботи локальних серверів, де необхідно мінімізувати споживання оперативної пам'яті. На відміну від громіздких рішень типу PostgreSQL чи MySQL, SQLite не створює додаткових фонових процесів. Оскільки база даних працює в тому ж адресному просторі, що й серверна частина на Flask, мережеві затримки на звернення до даних повністю відсутні. Це є критично важливим фактором для функції класифікації приміщень, яка повинна виконувати велику кількість математичних порівнянь поточних радіовідбитків з еталонними у режимі реального часу. Додатковою перевагою є нативна підтримка SQLite у мові Python, що дозволяє

працювати з даними через вбудовану бібліотеку без встановлення сторонніх драйверів.

Логічна структура бази складається з двох пов'язаних таблиць: таблиці збережених приміщень та таблиці векторів RSSI для відповідних точок доступу. Важливою архітектурною особливістю поточної реалізації є динамічне керування простором. Назви кімнат не закодовані жорстко у програмі і не вимагають ручного внесення адміністратором у базу. Замість цього розроблено програмний інтерфейс, який дозволяє системі або користувачеві гнучко створювати нові записи для приміщень на етапі налаштування.

Для забезпечення цілісності даних та оптимізації пошукових запитів застосовано базові принципи реляційної нормалізації. Фізично така архітектура реалізована шляхом розділення загального масиву інформації на незалежний довідник локацій та безпосередній архів вимірювань, що унеможливлює появу аномалій при оновленні чи видаленні даних.

Повна програмна реалізація модуля роботи з базою даних, що включає ініціалізацію таблиць, увімкнення підтримки зовнішніх ключів (Foreign Keys) та логіку додавання нових приміщень, наведена у Додатку Б. Далі наведено детальний опис кожної структурної одиниці бази даних.

rooms - базова таблиця, що містить у собі унікальний перелік усіх кімнат, доданих до системи для подальшого позиціонування.

room_data - таблиця, що зберігає зібрані еталонні радіовідбитки (рівні сигналів Wi-Fi мереж) для кожної кімнати та конкретного пристрою сканування.

Таблиця 2.2 - Опис таблиці rooms

Поле	Розмір поля	Тип даних	Значення
id	4	INTEGER	Первинний ключ (автоінкремент). Унікальний ідентифікатор кімнати
room_name	50	TEXT	Назва кімнати. Має обмеження UNIQUE для уникнення дублювання записів

Таблиця 2.3 - Опис таблиці room_data

Поле	Розмір поля	Тип даних	Значення
id	4	INTEGER	Первинний ключ (автоінкремент). Унікальний ідентифікатор запису
room_id	4	INTEGER	Зовнішній ключ, з'єднання з таблицею rooms (відношення «один до багатьох»)
device	17	TEXT	MAC-адреса клієнтського пристрою (ESP32), для якого зібрано еталонні дані
bssid	17	TEXT	MAC-адреса знайденої точки доступу (маршрутизатора), що транслює Wi-Fi сигнал
rssi	8	REAL	Середнє математичне значення сили сигналу (RSSI) для конкретної точки доступу
std	8	REAL	Стандартне відхилення (дисперсія) рівня сигналу під час збору еталона, що використовується як ваговий коефіцієнт

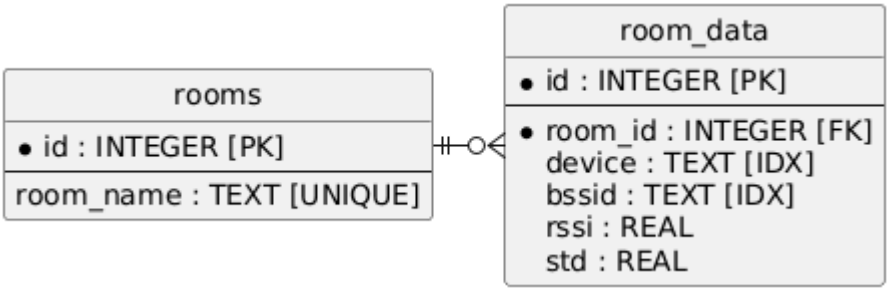

Рисунок 2.10 — Структура бази даних

Зв’язок між таблицями реалізовано за допомогою зовнішнього ключа room_id, що забезпечує цілісність даних і дозволяє однозначно пов’язати кожен Wi-Fi-відбиток із конкретною кімнатою. Використання унікального індексу для назв приміщень запобігає дублюванню записів, а індекси для полів device та bssid підвищують ефективність пошуку під час порівняння поточних RSSI-значень із еталонними даними. У межах реалізованої системи база даних використовується насамперед для підтримки алгоритму Wi-Fi Fingerprinting, тоді як поточні значення температури та вологості передаються на сервер і відображаються у веб-інтерфейсі без накопичення історії вимірювань. У

подальшому структура може бути розширена шляхом додавання таблиць для збереження температури, вологості, часу вимірювання та журналу результатів позиціонування.

2.6 Протокол взаємодії мікроконтролера ESP32 з сервером

Для організації взаємодії між апаратним вузлом на базі ESP32 та серверним застосунком обрано протокол HTTP з передачею даних у форматі JSON. Вибір HTTP обумовлений його повсюдною підтримкою в екосистемі розробки ESP32 через стандартну бібліотеку HTTPClient.h, а також простотою інтеграції з веб-сервером Flask без необхідності розгортання брокерів повідомлень або додаткової мережевої інфраструктури (як це вимагається, наприклад, у протоколі MQTT) [24,20]. JSON обрано як формат серіалізації даних завдяки його текстовій природі, зручності відлагодження та нативній підтримці в оптимізованій бібліотеці ArduinoJson для мікроконтролера.

Вузол ESP32 ініціює з'єднання та надсилає HTTP POST запит на кінцеву точку (endpoint) /temp-update з тілом у форматі JSON наступної структури:

```
{
  "device": "24:6F:28:AB:CD:EF",
  "temperature": 22.5,
  "humidity": 48.3,
  "rssi_list": [
    {"AA:BB:CC:DD:EE:FF": -65},
    {"11:22:33:44:55:66": -72}
  ]
}
```

Поле device містить MAC-адресу вбудованого Wi-Fi-чіпу ESP32, що слугує унікальним апаратним ідентифікатором вузла і дозволяє серверу підтримувати незалежні бази відбитків та історії вимірювань для кількох пристроїв одночасно. Поле temperature містить виміряне значення температури у градусах Цельсія, поле humidity - відносну вологість у відсотках. У разі

апаратного збою датчика DHT22 (відсутність відповіді або помилка контрольної суми), ці поля приймають системне маркерне значення -999, що сигналізує серверу про невалідність показників мікроклімату в поточному пакеті. Поле rssi_list є масивом об'єктів, кожен з яких являє собою пару BSSID-RSSI: ключем виступає MAC-адреса виявленої точки доступу Wi-Fi, значенням - усереднений рівень радіосигналу у дБм (від'ємне ціле число). Кількість елементів у масиві відповідає кількості видимих точок доступу у момент сканування (з рівнем сигналу вище -90 дБм) і може динамічно змінюватися.

Сервер виконує сувору валідацію отриманого JSON-документа на наявність обов'язкових полів. У разі успішного парсингу даних сервер відповідає HTTP-статусом 200 OK та мінімалістичним підтвердженням {"status": "success"}. У випадку отримання пошкоджених даних або відсутності ключових ідентифікаторів, сервер повертає статус 400 Bad Request із відповідним повідомленням про помилку (наприклад, {"status": "error", "message": "Missing device identifier"}).

Така одностороння архітектура телеметрії дозволяє максимально скоротити час очікування відповіді на стороні ESP32. Мікроконтролеру не потрібно витрачати процесорний час та оперативну пам'ять на десеріалізацію зворотної відповіді з назвою кімнати - він лише фіксує статус доставки пакета і негайно переходить у режим енергозбереження (Light Sleep), тоді як уся складна математична обробка та рендеринг результатів для кінцевого користувача делегуються серверному застосунку.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ МІКРОКОНТРОЛЕРНОЇ СИСТЕМИ МОНІТОРИНГУ

3.1 Реалізація прошивки мікроконтролера ESP32

Прошивку мікроконтролера реалізовано в середовищі розробки Arduino IDE 2.3 із використанням офіційного пакету підтримки плат ESP32 від Espressif Systems (esp32 Arduino Core). Для забезпечення функціонування системи використано розширений набір спеціалізованих бібліотек. Керування бездротовим модулем і сканування радіоефіру здійснюється за допомогою стандартної бібліотеки WiFi.h, а для низькорівневого налаштування параметрів живлення Wi-Fi тракту (вимикання енергозбереження під час активної фази сканування) застосовано системний заголовок esp_wifi.h. Зчитування показників мікроклімату реалізовано через бібліотеку DHT.h (Adafruit DHT sensor library). Серіалізацію зібраних даних у формат JSON забезпечує оптимізована бібліотека ArduinoJson.h, що дозволяє уникнути фрагментації пам'яті (Heap Fragmentation), яка виникає під час ручної конкатенації об'єктів типу String. Відправлення сформованих пакетів до серверного застосунку виконується модулем HTTPClient.h. Для підвищення енергоефективності пристрою стандартні функції програмної затримки замінено на апаратний режим зниженого енергоспоживання (Light Sleep Mode) за допомогою системної бібліотеки esp_sleep.h, що дозволяє суттєво зменшити струм споживання мікроконтролера в інтервалах між циклами сканування та передавання даних.

У функції setup() виконується ініціалізація послідовного порту зі швидкістю 115200 бод для відлагодження, підключення до точки доступу Wi-Fi та ініціалізація датчика DHT22 (див. Додаток А). Підключення до мережі реалізовано через блокуючий цикл:

```
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
}
```

Після успішного підключення виконується затримка 2000 мс (delay(2000)) для термічної стабілізації датчика DHT22 - без цієї паузи перше зчитування може повернути некоректне значення через неусталений тепловий режим сенсора.

Ідентифікатор пристрою формується динамічно через функцію WiFi.macAddress(), що повертає унікальну MAC-адресу вбудованого Wi-Fi чіпу у форматі XX:XX:XX:XX:XX:XX. Це забезпечує автоматичну унікальність ідентифікатора без ручного налаштування при розгортанні кількох вузлів - сервер розрізняє пристрої саме за цим полем при збереженні та порівнянні RSSI-відбитків у базі даних.

Основний цикл loop() виконує чотири послідовні операції. Спочатку зчитуються показники вологості та температури з DHT22:

```
humidity    = dht.readHumidity();
temperature = dht.readTemperature();
```

Для забезпечення надійності системи реалізовано механізм обробки апаратних помилок зчитування. У разі порушення зв'язку з датчиком або збою контрольної суми, функції бібліотеки повертають спеціальне значення NAN (Not A Number). Для перевірки валідності отриманих даних застосовується стандартна математична функція isnan(). Якщо виявлено помилку зчитування, відповідна змінна примусово ініціалізується значенням -999, що виступає системним маркером відсутності даних. Серверний застосунок під час обробки JSON-пакета розпізнає цей маркер і відображає прочерк у веб-інтерфейсі замість виведення некоректних числових значень:

```
if (isnan(humidity)) {
  humidity = -999;
}
if (isnan(temperature)) {
  temperature = -999;
}
```

Далі виконується сканування доступних Wi-Fi-мереж за допомогою функції WiFi.scanNetworks(). Особливістю мікроконтролера ESP32 є можливість виконувати сканування радіоефіру без розриву поточного з'єднання з точкою доступу в режимі станції (STA), що дозволяє підтримувати стабільний зв'язок із сервером. Для підвищення достовірності даних та згладжування апаратних флуктуацій радіосигналу алгоритм виконує п'ять послідовних ітерацій сканування з інтервалом 200 мс. Для оптимізації загального часу активної роботи мікроконтролера, час прослуховування кожного окремого Wi-Fi-каналу апаратно обмежено до 150 мс.

До результуючого набору потрапляють точки доступу, рівень сигналу (RSSI) яких дорівнює або перевищує порогове значення -90 дБм. Такий розширений поріг чутливості дозволяє алгоритму враховувати віддалені точки доступу, що підвищує точність Wi-Fi-відбитка для великих приміщень. Після кожної ітерації виділена під результати сканування оперативна пам'ять примусово вивільняється функцією WiFi.scanDelete() для запобігання витокам пам'яті (Memory Leaks).

На основі накопичених та усереднених значень формується JSON-документ. Для цього використовується бібліотека ArduinoJson сьомої версії з автоматичним розподілом пам'яті (JsonDocument), що гарантує синтаксичну правильність пакета даних та уникає проблеми фрагментації купи (Heap Fragmentation):

```
std::map<String, std::vector<int>> rssiAccum;

  unsigned long startTime = millis();
  for (int s = 0; s < 5; s++) {
    int n = WiFi.scanNetworks(false, false, false, 150);
    for (int i = 0; i < n; i++) {
      if (WiFi.RSSI(i) >= -90) {
        rssiAccum[WiFi.BSSIDstr(i)].push_back(WiFi.RSSI(i));
      }
    }
    WiFi.scanDelete();
    delay(200);
  }

  JsonDocument doc;
```

```
  doc["device"] = WiFi.macAddress();
  doc["temperature"] = temperature;
  doc["humidity"] = humidity;

  JsonArray rssi_array = doc["rssi_list"].to<JsonArray>();

  for (auto &pair : rssiAccum) {
    int sum = 0;
    for (int v : pair.second) sum += v;
    int avg = sum / pair.second.size();

    JsonObject net = rssi_array.add<JsonObject>();
    net[pair.first] = avg;
  }

  String json;
  serializeJson(doc, json);
```

Після успішної серіалізації даних ініціюється мережева взаємодія із серверним застосунком. За допомогою екземпляра класу HTTPClient встановлюється з'єднання за адресою, що визначена глобальною константою serverURL (у поточній конфігурації - http://192.168.0.105:5000/temp-update). Для забезпечення коректної обробки та парсингу даних на стороні Flask-сервера, до HTTP-заголовка обов'язково додається специфікатор формату передачі даних Content-Type: application/json. Повна структура JSON-повідомлення наведена у підрозділі 2.6.

Відправлення сформованого пакета здійснюється методом HTTP POST. Особливістю реалізованого алгоритму є вбудоване апаратне профілювання мережевих затримок. Мікроконтролер за допомогою таймера millis() фіксує час, витрачений на маршрутизацію HTTP-запиту, а також загальний час активного робочого циклу. Ці часові метрики, разом із кодом статусу відповіді сервера (або детальним описом помилки у разі збою), виводяться у послідовний інтерфейс (Serial Monitor) для апаратного відлагодження:

```
HTTPClient http;
  http.begin(serverURL);
  http.addHeader("Content-Type", "application/json");

  // Вимірювання часу виконання HTTP-запиту
  unsigned long startHttp = millis();
  int httpResponseCode = http.POST(json);
  unsigned long endHttp = millis();

  Serial.print("HTTP Time: ");
  Serial.println(endHttp - startHttp);
```

```
  // Обробка відповіді сервера
  if (httpResponseCode > 0) {
    Serial.printf("Response: %d\n", httpResponseCode);
  } else {
    Serial.printf("Error: %s\n", http.errorToString(httpResponseCode).c_str());
  }
  http.end();

  // Логування тривалості повного циклу
  Serial.printf("Повний цикл: %lu мс\n", millis() - loopStart);
```

Після успішного відправлення HTTP POST-запиту мікроконтролер конфігурує таймер пробудження та переходить у режим зниженого енергоспоживання (Light Sleep Mode) на 3 секунди, що дозволяє суттєво заощадити заряд джерела живлення між циклами вимірювання без втрати з'єднання з локальною мережею:

```
esp_sleep_enable_timer_wakeup(3000 * 1000ULL);
esp_light_sleep_start();
```

Повний вихідний код прошивки наведено у Додатку А.

3.2 Реалізація серверного застосунку Python/Flask

Серверний застосунок реалізовано на мові Python 3.10 з використанням мікрофреймворку Flask [26]. Структурно застосунок розділено на два основні модулі: app.py - головний модуль веб-сервера з визначенням маршрутів (API endpoints) та управлінням глобальним станом; wifi_check.py - модуль взаємодії з базою даних SQLite та реалізації математичного алгоритму класифікації (Wi-Fi Fingerprinting). Повний вихідний код серверного застосунку наведено у Додатку Б.

Для забезпечення масштабованості та можливості одночасної роботи з кількома мікроконтролерами ESP32, замість одиничних глобальних змінних використано словники device_data та device_history. Словник device_data зберігає актуальні показники температури, вологості, назву кімнати та мітку часу останньої активності (time.time()) унікально для кожної MAC-адреси.

Глобальний стан режиму навчання контролюється змінними training, target_room, current_device та буфером rssi_buffer.

Основний маршрут POST /temp-update приймає JSON-пакети від апаратних вузлів. На цьому етапі реалізовано сувору валідацію вхідних даних для запобігання критичним збоям сервера (HTTP 400 Bad Request). Залежно від стану змінної training, сервер або накопичує "сирі" дані у буфер, або виконує позиціонування пристрою. Важливою інновацією алгоритму є застосування методу "ковзного вікна" (Sliding Window) на базі структури даних deque з обмеженою довжиною (3 останні вимірювання). Замість миттєвої зміни локації, кінцевий результат визначається на основі найбільш частого значення (моди) в історії вимірювань за допомогою класу Counter. Це ефективно усуває проблему хибних "стрибків" інтерфейсу, викликаних короткочасними апаратними флуктуаціями радіоефіру:

```
@app.route('/temp-update', methods=['POST'])
def receive_data():
    global device_data, training, current_device, rssi_buffer, device_history
    data = request.get_json()
                if not data or not data.get('device') or not
isinstance(data.get('rssi_list'), list):
        return jsonify({"status": "error", "message": "Invalid data"}), 400
    device = data.get('device')
    rssi_list = data.get('rssi_list')
    # Оновлення телеметрії пристрою
    device_data[device] = {
        'temperature': data.get('temperature'),
        'humidity':    data.get('humidity'),
        'room':        'Unknown',
        'last_seen':   time.time()
    }
    if training:
        if not current_device: current_device = device
        if device == current_device: rssi_buffer.append(rssi_list)
    else:
        # Розрахунок кімнати та застосування програмного згладжування
        raw_room = wifi_check.compute_room_probabilities(rssi_list, device)
        if device not in device_history:
```

```
            device_history[device] = deque(maxlen=3)
        device_history[device].append(raw_room)
        most_common_room = Counter(device_history[device]).most_common(1)[0][0]
        device_data[device]['room'] = most_common_room
    return jsonify({"status": "success"})
```

Маршрути GET /start_training/<room> та GET /stop_training відповідають за управління життєвим циклом збору еталонних даних. При зупинці навчання накопичений масив передається до функції store_training_data(). На відміну від базових систем, які зберігають лише середнє арифметичне значення сигналу, розроблений алгоритм додатково обчислює дисперсію та стандартне відхилення (standard deviation) для кожної точки доступу. Це дозволяє системі оцінювати стабільність сигналу під час навчання і враховувати цей показник як "вагу" (weight) при подальшій класифікації:

```
final_entries = []
    for bssid, rssi_values in temp_list.items():
        avg_rssi = sum(rssi_values) / len(rssi_values)
                variance = sum((x - avg_rssi)**2 for x in rssi_values) /
len(rssi_values)
        std_rssi = math.sqrt(variance) if variance > 0 else 1.0
        final_entries.append((room_id, device, bssid, avg_rssi, std_rssi))
```

Функція compute_room_probabilities() реалізує модифікований алгоритм класифікації. Замість порівняння лише видимих точок доступу, система здійснює інверсний пошук: алгоритм ітерує по еталонних записах (expected data) кожної кімнати з бази даних і перевіряє їх наявність у поточному скануванні.

У разі виявлення еталонної мережі, обчислюється абсолютна різниця сигналів, зважена на обернено пропорційне стандартне відхилення (weight = 1.0 / (std + 1.0)). Якщо еталонна мережа відсутня у поточному радіоефірі, алгоритм застосовує "штраф", припускаючи, що рівень її сигналу впав до критичного мінімуму (-100 дБм). Для додаткового захисту від хибних спрацювань введено поріг покриття (Coverage): якщо кількість збігів між поточними та очікуваними

мережами становить менше 30% (coverage < 0.3), така кімната автоматично виключається з кандидатів:

```
for bssid, data in expected_bssids.items():
            expected_rssi = data['rssi']
            std = data['std']
            weight = 1.0 / (std + 1.0) # Ваговий коефіцієнт надійності мережі
            if bssid in current_scan:
                actual_rssi = current_scan[bssid]
                diff = abs(actual_rssi - expected_rssi)
                matched_count += 1
            else:
                # Штраф за відсутність очікуваної мережі
                diff = abs(-100 - expected_rssi)
            total_diff += diff * weight
            total_weight += weight
    coverage = matched_count / expected_count if expected_count > 0 else 0
    avg_diff = total_diff / total_weight if total_weight > 0 else float('inf')

  # Фільтрація аномалій
      if coverage >= 0.3:
      room_scores[room_id] = avg_diff
```

Результуючим приміщенням вважається кімната з мінімальним зваженим відхиленням (avg_diff). Такий комплексний підхід, що поєднує штрафи за зниклі мережі, вагу стандартного відхилення та згладжування часових рядів, забезпечує високу точність та стабільність визначення локації IoT-пристрою в умовах нестабільного радіосередовища.

3.3 Реалізація веб-інтерфейсу системи моніторингу

Веб-інтерфейс системи реалізовано у вигляді трьох HTML-шаблонів Flask (Jinja2): base.html - базовий шаблон з навігацією та підключенням стилів, index.html та training.html, що наслідують базовий через механізм блоків {% extends %} та {% block %}. Стилізацію винесено в окремий файл static/main.css,

що відповідає стандартній структурі Flask-проєкту та забезпечує кешування стилів браузером без повторного завантаження при переході між сторінками.

Для типографіки використано шрифтові гарнітури DM Sans та DM Mono, що завантажуються через Google Fonts API. DM Sans застосовується для основного тексту та елементів інтерфейсу, тоді як DM Mono використовується для відображення числових показників (температура, вологість) та технічних міток, що візуально підкреслює їх машинну природу. Підключення шрифтів реалізовано у base.html:

```
<link href="https://fonts.googleapis.com/css2?
  family=DM+Sans:wght@300;400;500;600
  &family=DM+Mono:wght@400;500
  &display=swap" rel="stylesheet">
```

Дизайн інтерфейсу виконано у світлій кольоровій схемі з використанням CSS-змінних у блоці :root файлу main.css. Основний фон сторінки - #ffffff, поверхні карток - #f8f9fa, межі елементів - #e2e4e9. Акцентний колір #2563eb використовується для активних посилань, кнопок та індикаторів стану. Зелений #059669 застосовується для індикатора активного з'єднання та показника температури. Використання CSS-змінних дозволяє змінити всю кольорову схему в одному місці без редагування окремих компонентів.

Сторінка index.html є головною панеллю моніторингу. Оскільки система підтримує роботу з кількома апаратними вузлами одночасно, інформаційні картки формуються динамічно для кожного підключеного пристрою за допомогою циклу Jinja2 {% for device_id, data in devices.items() %}. Показники температури, вологості та поточної кімнати виводяться у трьох окремих картках у CSS Grid-сітці з трьома рівними колонками (grid-template-columns: repeat(3, 1fr)). Кожна картка містить мітку у верхній частині та числове значення з одиницями виміру. Також реалізовано візуальний індикатор стану мережевої активності (онлайн/офлайн), що розраховується на основі мітки часу

останнього отриманого пакета. Умовний рендеринг кліматичних показників реалізовано через шаблонізатор Jinja2:

```
{% if latest_temperature %}
  <div class="stat-value success">
    {{ "%.1f"|format(latest_temperature|float) }}
    <span class="stat-unit">°C</span>
  </div>
{% else %}
  <div class="stat-value stat-null">—</div>
{% endif %}
```

Якщо дані ще не надійшли від пристрою або значення дорівнює -999 (маркер помилки датчика), замість числа відображається символ «-». Сторінка автоматично оновлюється кожні 16 секунд засобами мета-тегу:

```
<meta http-equiv="refresh" content="16">
```

Вигляд головної сторінки панелі моніторингу наведено на рисунку 3.1.

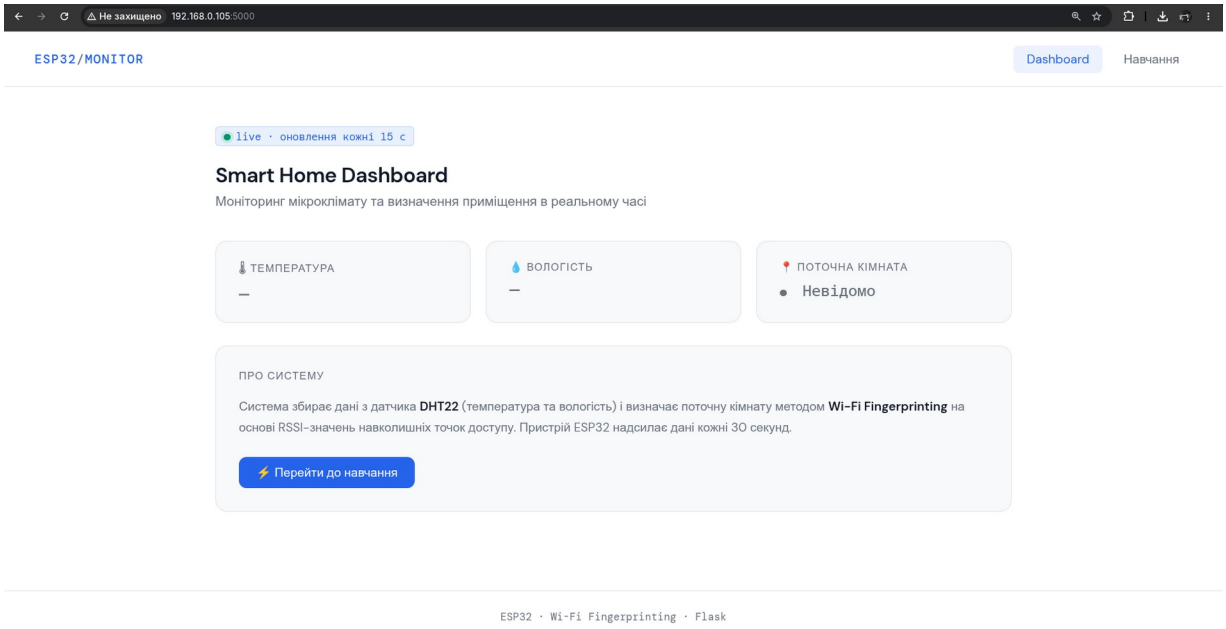

Рисунок 3.1 - Вигляд головної сторінки

Сторінка training.html призначена для керування режимом навчання та управління списком кімнат. На відміну від попередньої версії з

захардкодженими кнопками, кімнати завантажуються динамічно з бази даних і передаються у шаблон через маршрут /training_page:

```
@app.route('/training_page', methods=['GET'])
def training_page():
    rooms = wifi_check.get_all_rooms()
    return render_template('training.html', rooms=rooms)
```

У шаблоні training.html список кімнат відображається динамічно через цикл Jinja2. Поруч з кожною кімнатою розміщено кнопку видалення для зручного управління списком:

```
{% for room_id, room_name in rooms %}
  <a href="{{ url_for('start_training',
              room=room_name) }}"
     class="room-chip">{{ room_name }}</a>
  <a href="{{ url_for('delete_room',
              room_id=room_id) }}">X</a>
{% endfor %}
```

Форма додавання нової кімнати дозволяє задати довільну назву - це забезпечує портативність системи: при розгортанні в новому приміщенні (наприклад, в університеті) достатньо додати відповідні кімнати через інтерфейс без зміни коду. Обробник маршруту /add_room отримує назву з форми та викликає wifi_check.add_room():

```
@app.route('/add_room', methods=['POST'])
def add_room():
    room_name = request.form.get('room_name','').strip()
    if room_name:
        wifi_check.add_room(room_name)
    return redirect(url_for('training_page'))
```

Навігаційна панель у base.html містить логотип системи «ESP32/Monitor» виконаний шрифтом DM Mono та два посилання: Dashboard і Навчання.

Активний розділ підсвічується акцентним кольором з напівпрозорим фоном, що визначається динамічно через значення request.endpoint у шаблоні Jinja2:

```
<a href="{{ url_for('get_data') }}"
   {% if request.endpoint == 'get_data' %}
   class="active"{% endif %}>Dashboard</a>
```

Вигляд сторінки навчання з формою додавання кімнат наведено на рисунку 3.2.

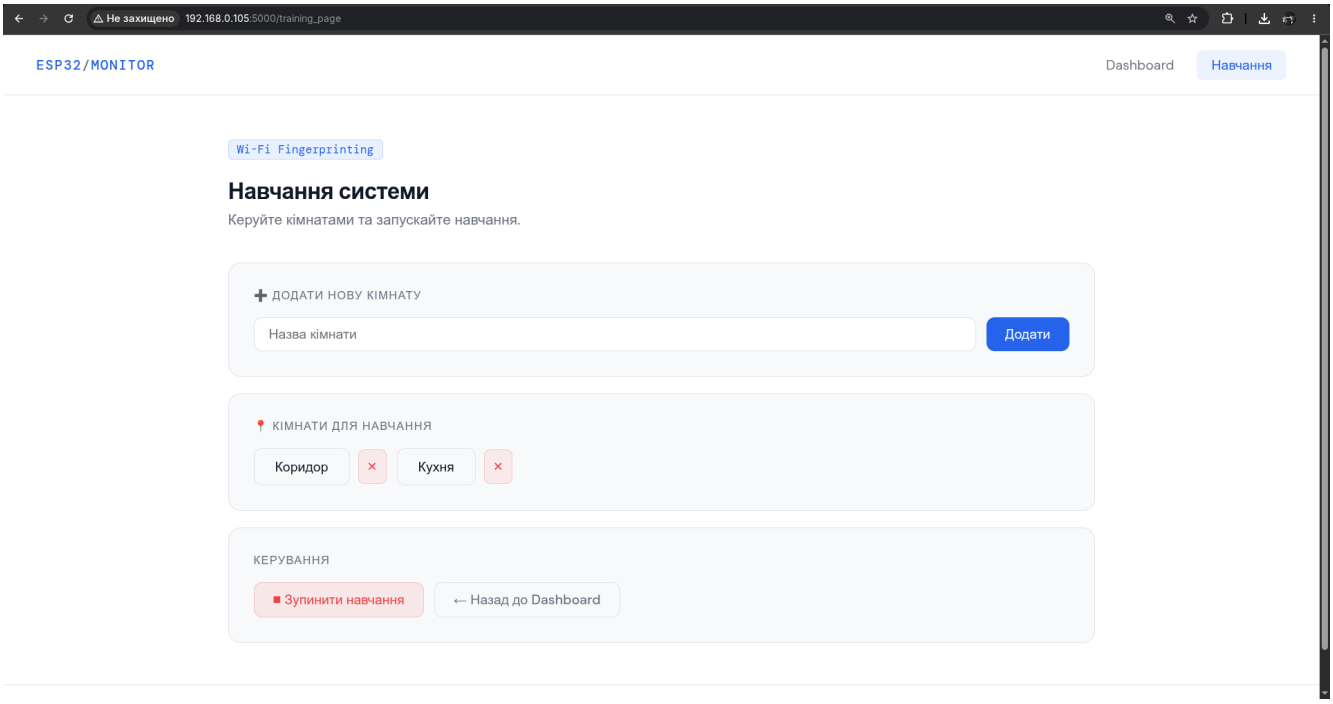

Рисунок 3.2 — Вигляд сторінки навчання

Таким чином, розроблений вебінтерфейс забезпечує інтуїтивно зрозумілу взаємодію користувача з програмно-апаратним комплексом. Використання сучасних вебтехнологій (шаблонізатора Jinja2 та CSS Grid) дозволило створити легкий і чутливий клієнтський додаток. Відмова від жорсткого кодування параметрів на користь динамічного управління через базу даних суттєво підвищує універсальність системи, дозволяючи легко масштабувати її та розгортати у нових приміщеннях без жодних втручань у вихідний код серверної частини.

3.4 Функціональне тестування системи в умовах приміщення

Функціональне тестування системи (зображену на рисунку 3.3) проводилося у житловому приміщенні, що складалося з 3 окремих зон: головної спальні, другої спальні та кухні. У зоні тестування було виявлено 4 точки доступу Wi-Fi, сигнали яких використовувалися для формування еталонних Wi-Fi-відбитків приміщень.

Для кожної кімнати було виконано етап навчання тривалістю 10 хвилин, упродовж якого система накопичувала результати сканування Wi-Fi-мереж у різних точках відповідного приміщення. У результаті для кожної кімнати було сформовано набір еталонних RSSI-значень, що надалі використовувався серверним застосунком для порівняння з поточними даними, отриманими від мікроконтролера ESP32.

Метою тестування було не метрологічне визначення похибки позиціонування, а перевірка працездатності алгоритму класифікації приміщення за Wi-Fi-відбитком. Оскільки система визначає не координати об’єкта, а належність поточного набору RSSI-значень до однієї з попередньо навчених кімнат, для оцінювання результату використано показник частки правильних визначень приміщення.

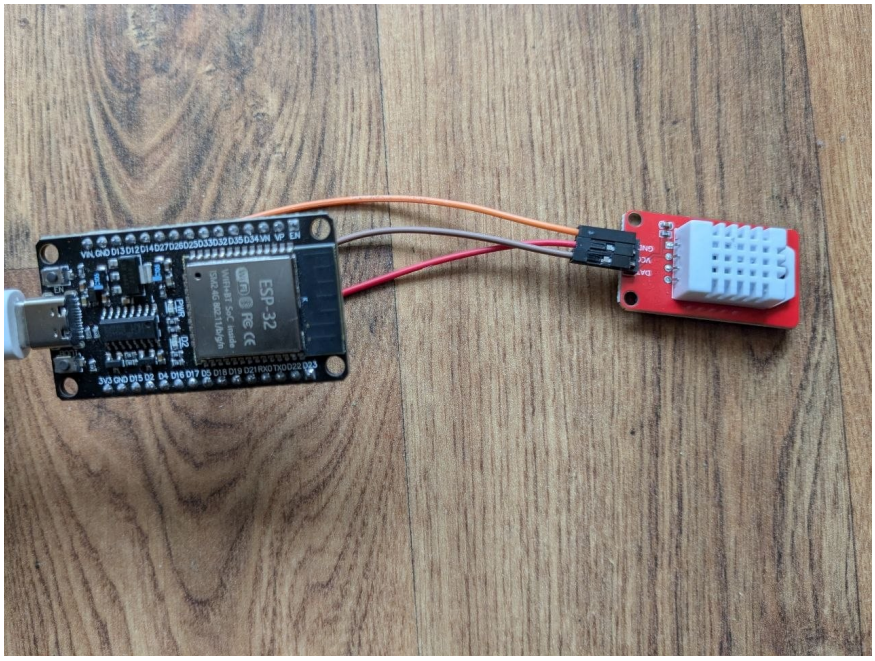

Рисунок 3.3 — Зовнішній вигляд складеної системи

Для окремої кімнати частка правильних визначень A розраховувалася за формулою:

$$\frac{T}{N} * 100\% \tag{3.1}$$

де A - точність визначення кімнати;

T - кількість правильно класифікованих вимірювань для кімнати;

N - загальна кількість тестових вимірювань кімнати.

Загальний показник правильності визначення приміщення A_{total} для всієї системи обчислювався як відношення сумарної кількості правильних визначень до загальної кількості контрольних спостережень::

$$\frac{\sum_{i=1}^{k} T_i}{\sum_{i=1}^{k} N_i} * 100\% \tag{3.2}$$

де A_{total} – загальна точність роботи системи класифікації, %;

k – загальна кількість досліджуваних кімнат у приміщенні;

i – порядковий номер кімнати (індекс сумування);

T_i – кількість правильно класифікованих вимірювань для і-тої кімнати;

N_i – загальна кількість тестових вимірювань, проведених в і-тій кімнаті.

Для перевірки роботи алгоритму було проведено 30 контрольних спостережень, по 10 для кожної кімнати. Під час кожного спостереження мікроконтролер ESP32 виконував сканування Wi-Fi-мереж, передавав отримані RSSI-значення на сервер, після чого сервер порівнював їх з еталонною базою відбитків і визначав найбільш імовірну кімнату.

Результати тестування показали, що з 30 контрольних спостережень система правильно визначила приміщення у 25 випадках. Отже, у межах проведеного експерименту частка правильних визначень становила 83 відсотки. Найнижчий результат отримано для кухні - 70%, що може пояснюватися

меншою кількістю стабільних Wi-Fi мереж у цій зоні, малою загальною площею приміщення та високою подібністю RSSI-профілів із суміжними кімнатами (наприклад, коридором). Крім того, наявність у спальні специфічних перешкод, таких як масивні дерев'яні меблі (шафи) або великі дзеркальні поверхні, створює ефект багатопроменевого поширення радіохвиль (multipath propagation). Це призводить до нелінійних відображень, поглинання частини спектра та локальних флуктуацій рівня сигналу, що ускладнює роботу математичного алгоритму класифікації.

Отримані результати підтверджують працездатність реалізованого алгоритму Wi-Fi Fingerprinting для розпізнавання кімнат у заданих умовах тестування. Водночас наведений показник слід розглядати як експериментальну оцінку правильності класифікації, а не як метрологічну характеристику системи. Для отримання статистично стійкішої оцінки доцільно збільшити кількість контрольних спостережень, провести тестування в різний час доби та врахувати можливий вплив зміни Wi-Fi-середовища, переміщення людей і зміни розташування побутових предметів.

Таблиця 3.1 – Результати функціональної перевірки правильності визначення приміщення

Кімната	Кількість контрольних спостережень	Кількість правильних визначень	Частка правильних визначень, %
Спальня головна	10	9	90
Спальня друга	10	9	90
Кухня	10	7	70

Для перевірки працездатності основного інформаційного каналу системи було проведено функціональне тестування передавання даних від мікроконтролера ESP32 до серверного застосунку Python/Flask (табл. 3.2). Під час тестування перевірялося послідовне виконання ключових операцій: підключення ESP32 до Wi-Fi-мережі, зчитування даних із датчика DHT22, сканування Wi-Fi-точок доступу, передавання JSON-пакета на сервер та відображення отриманих значень у веб-інтерфейсі.

Таблиця 3.2 - Результати функціонального тестування передавання даних

№ тесту	Очікуваний результат	Фактичний результат	Статус
1	ESP32 підключається до Wi-Fi	Підключення виконано	Виконано
2	ESP32 зчитує температуру і вологість	Дані отримано від DHT22	Виконано
3	ESP32 сканує доступні Wi-Fi-мережі	Отримано список BSSID та RSSI	Виконано
4	Дані передаються HTTP POST-запитом	Сервер повернув відповідь 200 OK	Виконано
5	Веб-інтерфейс оновлює поточні значення	Дані відображено на сторінці	Виконано

За результатами проведеного функціонального тестування встановлено, що мікроконтролер ESP32 коректно підключається до локальної Wi-Fi-мережі, зчитує показники температури та вологості з датчика DHT22, виконує сканування доступних Wi-Fi-точок доступу та формує набір даних, необхідний для подальшого визначення приміщення. Передавання даних на серверний застосунок Python/Flask здійснюється за допомогою HTTP POST-запиту у форматі JSON. Сервер успішно приймає отримані дані, обробляє їх і повертає відповідь про успішне виконання запиту.

Також підтверджено, що веб-інтерфейс системи відображає актуальні значення температури, вологості та результат визначення приміщення. Отже, проведене тестування підтвердило працездатність основного інформаційного каналу системи: від зчитування даних на стороні ESP32 до їх передавання, обробки на сервері та відображення у веб-інтерфейсі. При цьому перевірка показників температури та вологості мала функціональний характер і була спрямована на підтвердження факту отримання та передавання даних від датчика DHT22; метрологічна оцінка похибки вимірювання в межах цього тестування не проводилася.

Для перевірки коректності режиму навчання було виконано тестування процесу формування еталонних Wi-Fi-відбитків для окремих приміщень. Метою цього тестування було підтвердити, що після запуску навчання серверний застосунок переходить у режим накопичення RSSI-значень, отримує дані від мікроконтролера ESP32, виконує їх усереднення після завершення навчання та зберігає сформовані еталонні записи у базі даних SQLite.

Під час тестування послідовно перевірялися основні етапи режиму навчання: вибір кімнати у веб-інтерфейсі, запуск навчання, приймання RSSI-даних від ESP32, завершення навчання, запис усереднених значень RSSI до таблиці room_data та подальше використання цих записів для порівняння з поточними Wi-Fi-відбитками.

Таблиця 3.3 – Результати тестування коректності режиму навчання

№ тесту	Дія	Очікуваний результат	Фактичний результат	Статус
1	Вибір кімнати у веб-інтерфейсі	Сервер отримує назву кімнати для навчання	Кімнату обрано	Виконано
2	Запуск режиму навчання	Сервер переходить у режим накопичення RSSI-значень	Режим навчання активовано	Виконано
3	Надсилання даних від ESP32	Сервер приймає BSSID та RSSI виявлених Wi-Fi-точок	Дані приймаються сервером	Виконано
4	Накопичення кількох сканувань	RSSI-значення зберігаються у тимчасовому наборі даних	Дані накопичуються	Виконано
5	Завершення навчання	Сервер обчислює середні RSSI для кожного BSSID	Усереднені значення сформовано	Виконано
6	Запис результатів у SQLite	У таблиці room_data з’являються записи для обраної кімнати	Дані записано в БД	Виконано
7	Повторне визначення кімнати	Система використовує збережені Wi-Fi-відбитки для порівняння	Кімната визначається на основі записів БД	Виконано

Результати тестування підтвердили коректність роботи режиму навчання. Після запуску навчання система приймала від ESP32 дані про виявлені Wi-Fi-точки доступу, зокрема їх BSSID та RSSI-значення. Після завершення навчання серверний застосунок виконував усереднення накопичених RSSI-значень і записував сформовані еталонні Wi-Fi-відбитки до таблиці room_data бази даних SQLite.

Коректність навчання визначалася за такими ознаками: для обраної кімнати створювався або використовувався відповідний запис у таблиці rooms; у таблиці room_data з'являлися записи з прив'язкою до цієї кімнати через поле room_id; для кожної виявленої точки доступу зберігалися BSSID та усереднене значення RSSI; після завершення навчання ці дані використовувалися алгоритмом Wi-Fi Fingerprinting для подальшого визначення приміщення. Отже, режим навчання забезпечує формування еталонної бази Wi-Fi-відбитків, необхідної для роботи алгоритму позиціонування.

Для комплексної оцінки продуктивності розробленої системи було додатково проведено тестування часових характеристик (Performance Testing) ключових етапів збору, передавання та обробки даних. Метою цього етапу було визначення апаратних та мережевих затримок у системі, а також підтвердження її здатності працювати в режимі, наближеному до реального часу (Near Real-Time), без переповнення оперативної пам'яті мікроконтролера.

Вимірювання часу виконання на стороні мікроконтролера ESP32 здійснювалося за допомогою виклику вбудованої функції апаратного таймера `millis()`. Зокрема, для визначення тривалості повного циклу сканування Wi-Fi-мереж (з урахуванням фільтрації прихованих мереж та очищення результатів з пам'яті) і тривалості HTTP POST-запиту було імплементовано наступний програмний механізм логування:

```
//Заміряємо сканування мереж
unsigned long startScan = millis();
for (int s = 0; s < 3; s++) {
  int n = WiFi.scanNetworks();
  WiFi.scanDelete();
```

```
  delay(500);
}
unsigned long scanTime = millis() - startScan;
Serial.printf("Час сканування Wi-Fi: %lu мс\n", scanTime);
//Заміряємо час відправки і отримання запиту
unsigned long startHttp = millis();
int httpResponseCode = http.POST(json);
unsigned long httpTime = millis() - startHttp;
```

На стороні серверного застосунку (Python/Flask) вимірювання часу, витраченого на обробку отриманого JSON-пакета, доступ до бази даних SQLite та виконання математичного розрахунку реалізовано за допомогою стандартної бібліотеки time:

```
import time
@app.route('/temp-update', methods=['POST'])
def receive_data():
    start_time = time.time()
    process_time = (time.time() - start_time) * 1000
    print(f"Час обробки на сервері: {process_time:.2f} мс")
    return jsonify({"status": "success"})
```

Узагальнені результати вимірювання часових характеристик для повного циклу роботи системи наведено в таблиці 3.4. (Дані отримано шляхом усереднення результатів 20 послідовних ітерацій роботи пристрою).

Таблиця 3.4 – Часові характеристики виконання основних операцій системи

Параметр	Отримане значення	Параметр
Час сканування Wi-Fi-мереж	14-15 с	Час сканування Wi-Fi-мереж
Час HTTP POST-запиту до сервера	20-40 мс	Час HTTP POST-запиту до сервера
Повний цикл вимірювання та передавання	14-16 с	Повний цикл вимірювання та передавання
Інтервал оновлення веб-інтерфейсу	30 с	Інтервал оновлення веб-інтерфейсу

Аналіз отриманих часових характеристик свідчить про те, що найбільш ресурсоємною та тривалою операцією є апаратне сканування радіоефіру модулем ESP32, що зумовлено фізичними особливостями мультиплексування Wi-Fi каналів. Натомість програмна обробка даних на сервері (пошук еталонів у БД, застосування алгоритму класифікації та фільтрація хибних стрибків) виконується високоефективно і займає в середньому менше 20 мілісекунд. Треба відмітити що на такий великий період сканування мереж впливають затримка в 2500 мс, а також усереднення п’яти значень сигналів на esp.

Загальний активний час роботи мікроконтролера становить близько 15 секунд, що гарантовано вкладається у виділений часовий слот до наступного оновлення веб-інтерфейсу (30 секунд). Це забезпечує стабільну роботу системи без накопичення черги запитів та переповнення буферів пам'яті.

3.5 Аналіз результатів тестування та напрями вдосконалення системи

Розроблена система успішно вирішує поставлену задачу: здійснює моніторинг температури та вологості і визначає поточну кімнату з точністю 83%. До переваг системи слід віднести мінімальну вартість апаратної частини (ESP32 ~ 250 грн, DHT22 ~ 80 грн), повну відсутність залежності від хмарних сервісів, відкритий вихідний код та простоту розгортання у нових умовах завдяки динамічному управлінню кімнатами через веб-інтерфейс.

Практичну портативність системи підтверджено можливістю перенесення у нове приміщення (наприклад, аудиторія університету) без зміни коду: достатньо підключити ESP32 до нової Wi-Fi мережі (змінивши SSID та IP сервера у прошивці), додати нові кімнати через сторінку навчання і провести збір відбитків. Процедура навчання для трьох кімнат займає приблизно 30 хвилин.

Виявлені обмеження і шляхи їх усунення:

- при зміні Wi-Fi оточення (зміна розташування маршрутизаторів, поява нових точок доступу) необхідне повторне навчання - це можна автоматизувати фоновою перевіркою стабільності поточних відбитків;

- для підвищення точності в суміжних приміщеннях (туалет / коридор у даному тесті) доцільно збільшити кількість зразків при навчанні до 60-100 сканувань та використовувати кілька точок збору у різних частинах кімнати;

- для подальшого збільшення часу автономного живлення від акумулятора доцільно перейти з поточного режиму Light Sleep на режим глибокого сну ESP32 (esp_deep_sleep_start()). Хоча це вимагатиме повторного підключення до Wi-Fi на кожній ітерації, такий підхід знизить споживання струму в режимі очікування до мікроамперних значень (~10 мкА);

- для підвищення точності алгоритму позиціонування можна застосувати методи машинного навчання, такі як k-найближчих сусідів (kNN) або нейронні мережі, замість поточної зваженої квадратичної відстані - це дозволить системі краще адаптуватися до нелінійних змін радіосередовища та враховувати складні статистичні розподіли відбитків [22,23].

ВИСНОВКИ

У ході виконання бакалаврської роботи повністю вирішено поставлені науково-технічні завдання та отримано комплексні результати, що мають безпосереднє практичне значення для розвитку локальних систем Інтернету речей. У першу чергу, було проведено глибокий аналітичний огляд наявних методів моніторингу параметрів повітряного середовища та бездротового позиціонування об'єктів у закритих просторах, де супутникові навігаційні системи є неефективними. На основі порівняння технологій Bluetooth Low Energy, Ultra-Wideband та Wi-Fi було теоретично обґрунтовано, що метод Wi-Fi Fingerprinting є найбільш раціональним для задачі розрізнення кімнат у житлових та навчальних корпусах. Це зумовлено можливістю повноцінного використання вже розгорнутої локальної мережевої інфраструктури, високою проникаючою здатністю радіохвиль частотного діапазону 2,4 ГГц у капітальних будівлях та повною відсутністю потреби у закупівлі додаткових апаратних маяків.

Для технічної реалізації системи було обрано та всебічно обґрунтовано апаратну платформу на базі енергоефективного двоядерного мікроконтролера ESP32 модифікації ESP32-WROOM-32U у зв'язці з цифровим сенсором мікроклімату DHT22. Використання саме цієї модифікації обчислювального модуля із зовнішньою штирьовою антеною дозволило суттєво стабілізувати показники сили сигналу (RSSI), значно зменшити вплив багатопроменевого поширення хвиль від перешкод та мінімізувати просторові флуктуації радіокарти приміщення. Модуль ESP32 успішно поєднує в собі вбудований Wi-Fi стек, достатній обсяг оперативної пам'яті для тимчасової буферизації масивів RSSI та низьку вартість, що робить його оптимальним вузлом для систем IoT-моніторингу з інтегрованою функцією радіолокації.

У межах теоретичного забезпечення проєкту розроблено структурну схему взаємодії компонентів та оригінальний модифікований алгоритм Wi-Fi Fingerprinting. На відміну від базових систем, розроблена математична модель базується на обчисленні зваженої квадратичної відстані з урахуванням

стандартного відхилення (дисперсії) еталонних сигналів та застосуванням математичних штрафів за відсутні мережі. Алгоритм не вимагає розгортання важких сторонніх фреймворків, що дозволило ефективно реалізувати логіку порівняння сигнатур безпосередньо на стороні сервера за допомогою стандартних засобів мови Python та вбудованої реляційної бази даних SQLite. Структура бази даних була повністю оптимізована для швидкого збереження унікальних адрес точок доступу (BSSID) та вагових коефіцієнтів із жорсткою прив'язкою до ідентифікаторів конкретних кімнат під час фази калібрування системи.

Програмна реалізація проєкту охоплює два ключові рівні, а саме мікропрограмне забезпечення для периферійних вузлів та центральний серверний застосунок. Прошивку мікроконтролера ESP32 розроблено на базі мови C++ у середовищі Arduino IDE з використанням функцій багаторазового асинхронного сканування ефіру та інкапсуляції даних телеметрії в компактні пакети формату JSON для їх подальшої відправки через HTTP POST-запити. Серверна частина успішно реалізована на базі легковажного фреймворку Python/Flask, який надійно обробляє координаційні ендпоінти для оновлення поточної телеметрії та інтерактивного керування базою відбитків. Створений динамічний вебінтерфейс користувача на основі шаблонів HTML5 та рушія Jinja2 підтримує одночасну роботу з кількома апаратними вузлами, відображаючи поточні показники температури й вологості, статус мережевої активності та назву локалізованої кімнати в режимі реального часу.

Важливим етапом роботи стало комплексне натурне тестування розробленого програмно-апаратного комплексу в умовах реального приміщення з високим рівнем завад від сусідніх бездротових мереж. За результатами тривалих експериментів загальна точність автоматичного визначення поточної кімнати склала 83%, що впевнено перевищує початково задану технічну вимогу у 70% і підтверджує стійкість обраної метрики відхилень та методу статистичного згладжування («ковзного вікна») в умовах динамічної зміни радіоефіру.

Водночас виміряні сенсором DHT22 параметри температури та відносної вологості повітря повністю відповіли його офіційним паспортним характеристикам, що свідчить про належну точність збору кліматичних метрик.

Практична цінність виконаної бакалаврської роботи полягає у створенні повністю автономного, завершеного та відкритого IoT-рішення, яке є незалежним від сторонніх комерційних хмарних сервісів і забезпечує локальне збереження конфіденційних даних користувача. Розроблена система повністю готова до безпосереднього розгортання в побутових умовах, автоматизованих офісних просторах, об'єктах малого бізнесу чи навчальних лабораторіях кафедри. Весь вихідний код серверної та клієнтської частин детально задокументовано та структуровано, що відкриває широкі можливості для його подальшої адаптації під специфічні інженерні завдання та сторонні платформи.

На основі отриманих результатів окреслено перспективні напрями подальших досліджень, які дозволять суттєво масштабувати та модернізувати систему в майбутньому. Серед них виділяється перехід до використання класичних алгоритмів машинного навчання, таких як метод k-найближчих сусідів (kNN) або штучні нейронні мережі, для досягнення вищої точності позиціонування всередині складних залів. З погляду апаратного забезпечення планується перехід від реалізованого наразі режиму зниженого енергоспоживання (Light Sleep) до режиму глибокого сну (Deep Sleep) мікроконтролера для критичної мінімізації струму споживання, що дозволить перевести периферійні вузли на тривале автономне живлення від малогабаритних акумуляторів. Також передбачено розширення аналітичного функціоналу системи шляхом повноцінної інтеграції за протоколом MQTT у відкриту екосистему Home Assistant та підключення додаткових сенсорів вуглекислого газу й летких органічних сполук для розширення комплексного контролю мікроклімату.

ПЕРЕЛІК ПОСИЛАНЬ НА ДЖЕРЕЛА

1. Санітарні норми мікроклімату виробничих приміщень : ДСН 3.3.6.042-99. Чинний від 1999-12-01. Київ : МОЗ України, 1999. 16 с.

2. Semerjian L. et al. Low-Cost IoT-based Indoor Air Quality Monitoring. Journal of Architectural Engineering Technology. 2024. DOI: 10.1080/24751448.2024.2405403

3. Netatmo Smart Home Weather Station. Netatmo. URL: https://www.netatmo.com/en-gb/weather/weatherstation (дата звернення: 10.03.2026).

4. Farahsari P.S., Farahzadi A., Rezazadeh J., Bagheri A. A survey on indoor positioning systems for IoT-based applications. IEEE Internet of Things Journal. 2022. Vol. 9, no. 10. P. 7680–7699. DOI: 10.1109/JIOT.2021.3130617

5. Elsanhoury M. et al. Precision Positioning for Smart Logistics Using Ultra-Wideband Technology-Based Indoor Navigation: A Review. IEEE Access. 2022. Vol. 10. P. 44413–44445. DOI: 10.1109/ACCESS.2022.3168981

6. Muñoz-Organero M. et al. On the accuracy of BLE indoor localization systems: An assessment survey. Computers and Electrical Engineering. 2024. Vol. 118. DOI: 10.1016/j.compeleceng.2024.109455

7. Zafari F., Gkelias A., Leung K. K. A survey of indoor localization systems and technologies. IEEE Communications Surveys & Tutorials. 2019. Vol. 21, no. 3. P. 2568–2599.

8. Priyantha N. B., Chakraborty A., Balakrishnan H. The Cricket location-support system. Proceedings of ACM MobiCom'00, New York, 2000. P. 32–43. [класичне першоджерело — залишено]

9. Bahl P., Padmanabhan V. N. RADAR: An in-building RF-based user location and tracking system. Proceedings of IEEE INFOCOM'00, Tel Aviv, 2000. Vol. 2. P. 775–784. [класичне першоджерело — залишено]

10. Dai J. et al. A Survey of Latest Wi-Fi Assisted Indoor Positioning on Different Principles. Sensors. 2023. Vol. 23, no. 18. Art. 7961. DOI: 10.3390/s23187961

11. Feng X., Nguyen K. A., Luo Z. A review of open access Wi-Fi fingerprinting datasets for indoor positioning. IEEE Access. 2024. DOI: 10.1109/ACCESS.2024.3354987

12. IEEE Std 802.11-2020. Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. New York : IEEE, 2021. 4379 p.

13. ESP32 Technical Reference Manual. Version 5.1 / Espressif Systems. Shanghai : Espressif, 2023. 1100 p.

14. ESP32-WROOM-32U Datasheet. Version 3.2. Shanghai : Espressif Systems, 2022. 41 p.

15. DHT22/AM2302 Digital-output relative humidity & temperature sensor/module Datasheet. Guangzhou : Aosong Electronics, 2022. 4 p.

16. ESP32 Sleep Modes API Reference. Espressif Systems. URL: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/sleep_modes.html (дата звернення: 21.03.2026).

17. DHT sensor library for Arduino. GitHub. URL: https://github.com/adafruit/DHT-sensor-library (дата звернення: 15.03.2026).

18. Stroustrup B. The C++ Programming Language. 4th ed. Boston : Addison-Wesley Professional, 2013. 1368 p.

19. Arduino Reference. Arduino. URL: https://www.arduino.cc/reference/en/ (дата звернення: 12.03.2026).

20. Bray T. The JavaScript Object Notation (JSON) Data Interchange Format : RFC 8259. Internet Engineering Task Force, 2017. 16 p.

21. ArduinoJson: Efficient JSON serialization for C++. ArduinoJson. URL: https://arduinojson.org/ (дата звернення: 16.03.2026).

22. Singh N., Choe S., Punmiya R. Machine learning based indoor localization using Wi-Fi RSSI fingerprints: An overview. IEEE Access. 2021. Vol. 9. P. 127150–127174. DOI: 10.1109/ACCESS.2021.3112220

23. Murphy K.P. Probabilistic Machine Learning: An Introduction. Cambridge, MA : MIT Press, 2022. 864 p. ISBN 978-0262046824. URL: https://probml.github.io/pml-book/book1.html

24. Fielding R. et al. HTTP Semantics : RFC 9110. Internet Engineering Task Force, 2022. URL: https://www.rfc-editor.org/rfc/rfc9110

25. Python 3.10 Documentation. Python Software Foundation. URL: https://docs.python.org/3.10/ (дата звернення: 15.03.2026).

26. Flask Documentation. Version 3.1.x. Pallets, 2024. URL: https://flask.palletsprojects.com/en/3.1.x/ (дата звернення: 18.03.2026).

27. SQLite Documentation. SQLite. URL: https://www.sqlite.org/docs.html (дата звернення: 20.03.2026).

28. Jinja2 Documentation. Pallets. URL: https://jinja.palletsprojects.com/ (дата звернення: 18.03.2026).

29. CSS Grid Layout. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_grid_layout (дата звернення: 18.03.2026).

30. Home Assistant Documentation. Home Assistant. URL: https://www.home-assistant.io/docs/ (дата звернення: 22.03.2026).

ДОДАТКИ

ДОДАТОК А

Вихідний код прошивки esp32 (esp32_monitor_firmware.ino)

```
#include <WiFi.h>
#include <DHT.h>
#include <HTTPClient.h>
#include "esp_wifi.h"
#include <map>
#include <vector>
#include <ArduinoJson.h>
#include "esp_sleep.h"

#define DHTPIN 4 // Сенсор підключений до GPIO4
#define DHTTYPE DHT22 // Тип сенсору DHT22

const char *ssid = "Pentagon";
const char *password = "alabaj2005";
const char *serverURL = "http://192.168.0.105:5000/temp-update";

DHT dht(DHTPIN, DHTTYPE);
float humidity;
float temperature;

void setup()
{
Serial.begin(115200);
delay(10);

// Починаємо з'єднання з WiFi

Serial.println();
Serial.println();
Serial.print("Connecting to ");
Serial.println(ssid);

WiFi.mode(WIFI_AP_STA);
esp_wifi_set_ps(WIFI_PS_NONE);
WiFi.setTxPower(WIFI_POWER_19_5dBm);
WiFi.begin(ssid, password);

while (WiFi.status() != WL_CONNECTED)
{
delay(500);
Serial.print(".");
}

Serial.println("");
Serial.println("WiFi connected.");
Serial.println("IP address: ");
Serial.println(WiFi.localIP());

dht.begin();
delay(5000);
}

void loop()
{
unsigned long loopStart = millis();
```

```
// Зчитування DHT з перевіркою на помилки
humidity = dht.readHumidity();
temperature = dht.readTemperature();
if (isnan(humidity) || isnan(temperature))
{
humidity = -999;
temperature = -999;
}

std::map<String, std::vector<int>> rssiAccum;

unsigned long startTime = millis();
for (int s = 0; s < 5; s++)
{
int n = WiFi.scanNetworks(false, false, false, 150);

for (int i = 0; i < n; i++)
{
// Поріг чутливості -90 дозволить бачити далекі кімнати
if (WiFi.RSSI(i) >= -90)
{
rssiAccum[WiFi.BSSIDstr(i)].push_back(WiFi.RSSI(i));
}
}
WiFi.scanDelete();
delay(200);
}
unsigned long endTime = millis();
Serial.print("Scan Time: ");
Serial.println(endTime - startTime);

// 3. Формування JSON
JsonDocument doc;
doc["device"] = WiFi.macAddress();
doc["temperature"] = temperature;
doc["humidity"] = humidity;

JsonArray rssi_array = doc["rssi_list"].to<JsonArray>();

for (auto &pair : rssiAccum)
{
int sum = 0;
for (int v : pair.second)
sum += v;
int avg = sum / pair.second.size();

JsonObject net = rssi_array.add<JsonObject>();
net[pair.first] = avg;
}

String json;
serializeJson(doc, json); // Конвертуємо об'єкт у готовий рядок

// Відправка HTTP
HTTPClient http;
http.begin(serverURL);
http.addHeader("Content-Type", "application/json");

unsigned long startHttp = millis();
int httpResponseCode = http.POST(json);
unsigned long endHttp = millis();
```

```
Serial.print("HTTP Time: ");
Serial.println(endHttp - startHttp);

if (httpResponseCode > 0)
{
Serial.printf("Response: %d\n", httpResponseCode);
}
else
{
Serial.printf("Error: %s\n", http.errorToString(httpResponseCode).c_str());
}
http.end();

Serial.printf("Повний цикл: %lu мс\n", millis() - loopStart);

esp_sleep_enable_timer_wakeup(3000 * 1000ULL); // мікросекунди
esp_light_sleep_start();
}
```

ДОДАТОК Б

Вихідний код серверного застосунку (app.py, wifi_check.py)

Файл app.py:

```
from flask import Flask, redirect, request, jsonify, render_template, url_for
import sqlite3
import wifi_check
import time
from collections import deque, Counter

app = Flask(__name__)

# Словник де зберігаються останні дані від кожного пристрою.
# Ключ - MAC-адреса ESP32, значення - температура, вологість, кімната і час останнього запиту
device_data = {}

# Буфер що накопичує RSSI-скани під час навчання
# Скидається при старті кожного нового навчання
rssi_buffer = []

# Історія визначених кімнат для кожного пристрою
# Використовується для згладжування - беремо найчастіше значення з останніх N визначень
device_history = {}

# Прапор активного режиму навчання і супутні змінні
training = False
target_room = None
current_device = None

# Ініціалізація БД при старті — створює таблиці якщо їх ще немає
wifi_check.initialize_db()


@app.route('/temp-update', methods=['POST'])
def receive_data():
global device_data, training, current_device, rssi_buffer, device_history

data = request.get_json()
if not data:
return jsonify({"status": "error", "message": "Invalid JSON"}), 400

device = data.get('device')
rssi_list = data.get('rssi_list')

# Перевіряємо що пристрій надіслав коректний ідентифікатор і список мереж
# Без цього не можемо ні навчати ні визначати кімнату
if not device or not isinstance(device, str):
return jsonify({"status": "error",
"message": "Missing device identifier"}), 400
if not isinstance(rssi_list, list):
return jsonify({"status": "error",
"message": "Missing or invalid rssi_list"}), 400

# Оновлюємо запис пристрою. Кімнату поки ставимо Unknown -
# нижче або перезапишемо з алгоритму, або залишимо якщо йде навчання
device_data[device] = {
```

```
'temperature': data.get('temperature'),
'humidity': data.get('humidity'),
'room': device_data.get(device, {}).get('room', 'Unknown'),
'last_seen': time.time()
}

if training:
# Перший пристрій що надіслав дані під час навчання стає
# "навчальним" — решта ігнорується щоб не мішати відбитки
if not current_device:
current_device = device
if device == current_device:
rssi_buffer.append(rssi_list)

if not training:
# Визначаємо кімнату і додаємо результат в історію
# Беремо найчастіше значення з останніх 3 визначень -
# це усуває одиночні хибні спрацювання
raw_room = wifi_check.compute_room_probabilities(rssi_list, device)
if device not in device_history:
device_history[device] = deque(maxlen=3)
device_history[device].append(raw_room)
most_common_room = Counter(
device_history[device]).most_common(1)[0][0]
device_data[device]['room'] = most_common_room

return jsonify({"status": "success"})


@app.route('/start_training/<room>')
def start_training(room):
global training, target_room, rssi_buffer, current_device

# Скидаємо буфер і current_device щоб навчання починалось чисто
# current_device визначиться автоматично при першому POST від ESP32
training = True
target_room = room
rssi_buffer = []
current_device = None
return redirect(url_for('training_page'))


@app.route('/stop_training')
def stop_training():
global training, rssi_buffer, target_room, current_device

# Зберігаємо зібрані відбитки тільки якщо є що зберігати
# Якщо навчання зупинили одразу - просто скидаємо стан
if training and rssi_buffer and current_device:
wifi_check.store_training_data(
target_room, current_device, rssi_buffer)

training = False
rssi_buffer = []
target_room = None
current_device = None
return redirect(url_for('training_page'))


@app.route('/', methods=['GET'])
def get_data():
```

```
# Передаємо поточний час щоб шаблон міг визначити
# чи пристрій онлайн (last_seen не старше 30 секунд)
return render_template('index.html',
devices=device_data,
now=time.time())


@app.route('/add_room', methods=['POST'])
def add_room():
room_name = request.form.get('room_name', '').strip()
if room_name:
wifi_check.add_room(room_name)
return redirect(url_for('training_page'))


@app.route('/delete_room/<int:room_id>')
def delete_room(room_id):
# Видаляємо кімнату разом з усіма її RSSI-відбитками в БД
wifi_check.delete_room(room_id)
return redirect(url_for('training_page'))


@app.route('/training_page', methods=['GET'])
def training_page():
rooms = wifi_check.get_all_rooms()
return render_template('training.html',
rooms=rooms,
training=training,
target_room=target_room)


if __name__ == '__main__':
app.run(debug=True, port=5000, host='0.0.0.0')
```

Файл wifi_check.py:

```
import sqlite3
from collections import defaultdict
import math

def initialize_db():
con = sqlite3.connect('rooms.db')
cur = con.cursor()
# Вмикаємо підтримку зовнішніх ключів у SQLite
cur.execute("PRAGMA foreign_keys = ON")
# Створення таблиці кімнат
cur.execute("""CREATE TABLE IF NOT EXISTS rooms
(id INTEGER PRIMARY KEY AUTOINCREMENT,
room_name TEXT UNIQUE)""")
con.commit()
# Таблиця для еталонних відбитків (зберігає середній RSSI та стандартне відхилення)
cur.execute("""CREATE TABLE IF NOT EXISTS room_data
(id INTEGER PRIMARY KEY AUTOINCREMENT,
room_id INTEGER, device TEXT,
bssid TEXT, rssi REAL, std REAL,
FOREIGN KEY (room_id) REFERENCES rooms(id))""")
```

```
con.commit()
con.close()

def add_room(room_name):
con = sqlite3.connect('rooms.db')
cur = con.cursor()
# Ігноруємо запит, якщо кімната вже існує
cur.execute("INSERT OR IGNORE INTO rooms (room_name) VALUES (?)",
(room_name,))
con.commit()
con.close()

def delete_room(room_id):
con = sqlite3.connect('rooms.db')
cur = con.cursor()
# Спочатку видаляємо відбитки, потім саму кімнату для збереження цілісності БД
cur.execute("DELETE FROM room_data WHERE room_id=?", (room_id,))
cur.execute("DELETE FROM rooms WHERE id=?", (room_id,))
con.commit()
con.close()

def get_all_rooms():
con = sqlite3.connect('rooms.db')
cur = con.cursor()
cur.execute("SELECT id, room_name FROM rooms")
rooms = cur.fetchall()
con.close()
return rooms

def store_training_data(room, device, rssi_list):
# Перевірка валідності вхідних даних
if not isinstance(rssi_list, list) or not rssi_list:
return

con = sqlite3.connect('rooms.db')
cur = con.cursor()
# Отримуємо ID кімнати
cur.execute("SELECT id FROM rooms WHERE room_name=?", (room,))
res = cur.fetchone()
if not res:
con.close()
return
room_id = res[0]
# Групування значень RSSI за BSSID точок доступу
temp_list = defaultdict(list)

for scan in rssi_list:
if not isinstance(scan, list):
continue
for pack in scan:
if isinstance(pack, dict):
for bssid, rssi in pack.items():
temp_list[bssid].append(rssi)

final_entries = []
for bssid, rssi_values in temp_list.items():
# Розрахунок середнього значення
avg_rssi = sum(rssi_values) / len(rssi_values)
# Розрахунок дисперсії та стандартного відхилення
variance = sum((x - avg_rssi)**2 for x in rssi_values) / len(rssi_values)
# Запобігання діленню на нуль у подальших розрахунках
```

```
std_rssi = math.sqrt(variance) if variance > 0 else 1.0
final_entries.append((room_id, device, bssid, avg_rssi, std_rssi))

# Видалення старих відбитків перед записом нових даних
cur.execute("DELETE FROM room_data WHERE room_id=? AND device=?", (room_id, device))
cur.executemany("INSERT INTO room_data (room_id, device, bssid, rssi, std) VALUES (?, ?, ?, ?, ?)",
final_entries)
con.commit()
con.close()

def compute_room_probabilities(rssi_list, device):
if not isinstance(rssi_list, list) or not rssi_list:
return "Unknown (No RSSI data)"

# Формування словника {BSSID: RSSI} з поточного сканування
current_scan = {}
for pack in rssi_list:
if isinstance(pack, dict):
for bssid, rssi in pack.items():
current_scan[bssid] = rssi

con = sqlite3.connect('rooms.db')
cur = con.cursor()

# Отримання списку кімнат
cur.execute("SELECT id, room_name FROM rooms")
rooms_info = {row[0]: row[1] for row in cur.fetchall()}

# Завантаження еталонних відбитків для конкретного пристрою
cur.execute("SELECT room_id, bssid, rssi, std FROM room_data WHERE device=?", (device,))
expected_data = {}
for room_id, bssid, stored_rssi, std in cur.fetchall():
if room_id not in expected_data:
expected_data[room_id] = {}
expected_data[room_id][bssid] = {'rssi': stored_rssi, 'std': std}
con.close()

if not expected_data:
return "Unknown (No matches in DB)"

room_scores = {}

# Порівняння поточного скану з еталонами кожної кімнати
for room_id, expected_bssids in expected_data.items():
total_diff = 0
total_weight = 0
matched_count = 0
expected_count = len(expected_bssids)

for bssid, data in expected_bssids.items():
expected_rssi = data['rssi']
std = data['std']
# Розрахунок вагового коефіцієнта на основі стабільності сигналу
weight = 1.0 / (std + 1.0)

if bssid in current_scan:
# Розрахунок різниці для знайдених мереж
actual_rssi = current_scan[bssid]
diff = abs(actual_rssi - expected_rssi)
matched_count += 1
else:
```

Кінець додатку Б

```
# Штраф за відсутню мережу (встановлюємо рівень сигналу на апаратний мінімум)
diff = abs(-100 - expected_rssi)
total_diff += diff * weight
total_weight += weight

# Розрахунок відсотка виявлених еталонних мереж
coverage = matched_count / expected_count if expected_count > 0 else 0
avg_diff = total_diff / total_weight if total_weight > 0 else float('inf')

# Відкидання кімнат з низьким рівнем покриття (<30%)
if coverage >= 0.3:
room_scores[room_id] = avg_diff

if not room_scores:
return "Unknown (Low coverage)"

# Вибір кімнати з найменшим зваженим відхиленням
best_room_id = min(room_scores, key=room_scores.get)
return rooms_info.get(best_room_id, "Unknown")

def get_room_name_by_id(room_id):
con = sqlite3.connect('rooms.db')
cur = con.cursor()
cur.execute("SELECT room_name FROM rooms WHERE id=?", (room_id,))
res = cur.fetchone()
con.close()
return res[0] if res else "Unknown"
```

ДОДАТОК В

Вихідний код вигляду сайту (main.css, base.html, index.html, training.html)

Файл main.css:

```
*,
*::before,
*::after {
box-sizing: border-box;
margin: 0;
padding: 0;
}

:root {
--bg: #0f1117;
--surface: #1a1d27;
--border: #2a2d3a;
--accent: #4f8ef7;
--accent2: #7c3aed;
--text: #e8eaf0;
--muted: #6b7280;
--success: #10b981;
--warn: #f59e0b;
--r: 12px;
}

html {
height: 100%;
}

body {
font-family: 'DM Sans', sans-serif;
background: var(--bg);
color: var(--text);
min-height: 100vh;
display: flex;
flex-direction: column;
}

/* nav */
nav {
display: flex;
align-items: center;
justify-content: space-between;
padding: 0 32px;
height: 58px;
border-bottom: 1px solid var(--border);
background: var(--surface);
}

.nav-logo {
font-family: 'DM Mono', monospace;
font-size: 13px;
font-weight: 500;
color: var(--accent);
letter-spacing: 0.08em;
text-transform: uppercase;}
```

```
.nav-logo span {
color: var(--muted);
}

.nav-links {
display: flex;
gap: 8px;
}

.nav-links a {
font-size: 13px;
color: var(--muted);
text-decoration: none;
padding: 6px 14px;
border-radius: 6px;
transition: all 0.15s;
}

.nav-links a:hover {
color: var(--text);
background: var(--border);
}

.nav-links a.active {
color: var(--accent);
background: rgba(79, 142, 247, 0.1);
}

/* main container */
main {
flex: 1;
padding: 40px 32px;
max-width: 900px;
width: 100%;
margin: 0 auto;
}

/* page header */
.page-header {
margin-bottom: 32px;
}

.page-header h1 {
font-size: 22px;
font-weight: 600;
letter-spacing: -0.02em;
margin-bottom: 4px;
}

.page-header p {
font-size: 14px;
color: var(--muted);
}

/* cards */
.card {
background: var(--surface);
border: 1px solid var(--border);
border-radius: var(--r);
padding: 24px;
}
```

```
.card+.card {
margin-top: 16px;
}

/* stat row */
.stats-grid {
display: grid;
grid-template-columns: repeat(3, 1fr);
gap: 16px;
margin-bottom: 24px;
}

.stat-card {
background: var(--surface);
border: 1px solid var(--border);
border-radius: var(--r);
padding: 20px 24px;
}

.stat-label {
font-size: 11px;
font-weight: 500;
text-transform: uppercase;
letter-spacing: 0.08em;
color: var(--muted);
margin-bottom: 8px;
}

.stat-value {
font-family: 'DM Mono', monospace;
font-size: 28px;
font-weight: 500;
color: var(--text);
line-height: 1;
}

.stat-value.accent {
color: var(--accent);
}

.stat-value.success {
color: var(--success);
}

.stat-unit {
font-family: 'DM Mono', monospace;
font-size: 14px;
color: var(--muted);
margin-left: 4px;
}

.stat-null {
color: var(--muted);
font-size: 18px;
}

/* buttons */
.btn {
display: inline-flex;
align-items: center;
gap: 6px;
```

```
font-family: 'DM Sans', sans-serif;
font-size: 13px;
font-weight: 500;
padding: 8px 18px;
border-radius: 8px;
border: none;
cursor: pointer;
text-decoration: none;
transition: all 0.15s;
}

.btn-primary {
background: var(--accent);
color: #fff;
}

.btn-primary:hover {
background: #6b9ff8;
transform: translateY(-1px);
}

.btn-ghost {
background: transparent;
color: var(--muted);
border: 1px solid var(--border);
}

.btn-ghost:hover {
color: var(--text);
border-color: var(--muted);
}

.btn-danger {
background: rgba(239, 68, 68, 0.1);
color: #ef4444;
border: 1px solid rgba(239, 68, 68, 0.2);
}

.btn-danger:hover {
background: rgba(239, 68, 68, 0.2);
}

/* room chips */
.room-grid {
display: flex;
flex-wrap: wrap;
gap: 10px;
}

.room-chip {
font-size: 13px;
font-weight: 500;
padding: 9px 20px;
border-radius: 8px;
border: 1px solid var(--border);
background: var(--surface);
color: var(--text);
cursor: pointer;
text-decoration: none;
transition: all 0.15s;
font-family: 'DM Sans', sans-serif;
```

```
}

.room-chip:hover {
border-color: var(--accent);
color: var(--accent);
background: rgba(79, 142, 247, 0.06);
transform: translateY(-1px);
}

/* badge */
.badge {
display: inline-block;
font-family: 'DM Mono', monospace;
font-size: 11px;
padding: 2px 8px;
border-radius: 4px;
background: rgba(79, 142, 247, 0.12);
color: var(--accent);
border: 1px solid rgba(79, 142, 247, 0.25);
margin-bottom: 16px;
}

/* section label */
.section-label {
font-size: 11px;
font-weight: 500;
text-transform: uppercase;
letter-spacing: 0.08em;
color: var(--muted);
margin-bottom: 14px;
}

/* dot indicator */
.dot {
display: inline-block;
width: 7px;
height: 7px;
border-radius: 50%;
margin-right: 6px;
}

.dot-green {
background: var(--success);
box-shadow: 0 0 0 3px rgba(16, 185, 129, 0.15);
}

.dot-gray {
background: var(--muted);
}

/* actions row */
.actions {
display: flex;
gap: 10px;
align-items: center;
flex-wrap: wrap;
}

footer {
text-align: center;
font-size: 11px;
```

```
color: var(--muted);
padding: 20px;
border-top: 1px solid var(--border);
font-family: 'DM Mono', monospace;
}
```

Файл base.html:

```
<!DOCTYPE html>
<html lang="uk">

<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<link rel="preconnect" href="https://fonts.googleapis.com">
<link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
<link
href="https://fonts.googleapis.com/css2?
family=DM+Sans:wght@300;400;500;600&family=DM+Mono:wght@400;500&display=swap"
rel="stylesheet">
{% block title %}<title>Smart Home</title>{% endblock %}
<link rel="stylesheet" href="{{ url_for('static', filename='main.css') }}">
</head>

<body>
<nav>
<div class="nav-logo">ESP32<span>/</span>Monitor</div>
<div class="nav-links">
<a href="{{ url_for('get_data') }}" {% if request.endpoint=='get_data' %}class="active" {% endif
%}>Dashboard</a>
<a href="{{ url_for('training_page') }}" {% if request.endpoint=='training_page' %}class="active" {% endif
%}>Навчання</a>
</div>
</nav>
<main>
{% block body %}{% endblock %}
</main>
<footer>ESP32 · Wi-Fi Fingerprinting · Flask</footer>
</body>

</html>
```

Файл index.html:

```
{% extends "base.html" %}

{% block title %}
<title>Smart Home Dashboard</title>
<meta http-equiv="refresh" content="16">
{% endblock %}

{% block body %}

<div class="page-header">
<div class="badge">
<span class="dot dot-green"></span>live · оновлення кожні 16 с
```

```
</div>
<h1>Smart Home Dashboard</h1>
<p>Моніторинг мікроклімату та визначення приміщення в реальному часі</p>
</div>

<!-- Stats -->
{% for device_id, data in devices.items() %}
<div class="card" style="margin-bottom:16px;">
{% set is_online = (now - data.last_seen) < 30 %} <div class="section-label">
📡 {{ device_id }}
{% if is_online %}
<span style="font-size:11px; color:#059669; margin-left:8px;">
● онлайн
</span>
{% else %}
<span style="font-size:11px; color:#6b7280; margin-left:8px;">
○ офлайн · {{ ((now - data.last_seen) / 60)|int }} хв тому
</span>
{% endif %}
</div>
<div class="stats-grid">
<div class="stat-card">
<div class="stat-label">🌡️ Температура</div>
<div class="stat-value success">
{% if data.temperature and data.temperature != -999 %}
{{ "%.1f"|format(data.temperature|float) }}
<span class="stat-unit">°C</span>
{% else %}—{% endif %}
</div>
</div>
<div class="stat-card">
<div class="stat-label">💧 Вологість</div>
<div class="stat-value accent">
{% if data.humidity and data.humidity != -999 %}
{{ "%.1f"|format(data.humidity|float) }}
<span class="stat-unit">%</span>
{% else %}—{% endif %}
</div>
</div>
<div class="stat-card">
<div class="stat-label">📍 Кімната</div>
<div class="stat-value" style="font-size:17px;">
{{ data.room }}
</div>
</div>
</div>
</div>
{% else %}
<div class="card">
<p style="color:var(--muted); font-size:13px;">
Пристроїв ще немає — очікуємо підключення ESP32...
</p>
</div>
{% endfor %}

<!-- Info card -->
<div class="card">
<div class="section-label">Про систему</div>
<p style="font-size:13px; color:var(--muted); line-height:1.7;">
Система збирає дані з датчика <strong style="color:var(--text)">DHT22</strong> (температура та вологість)
і визначає поточну кімнату методом
```

Wi-Fi Fingerprinting на основі RSSI-значень навколишніх точок доступу. Пристрій ESP32 надсилає дані кожні 30 секунд.

⚡ Перейти до навчання

{% endblock %}

Файл training.html:

{% extends "base.html" %}
{% block title %}<title>Training Page</title>{% endblock %}
{% block body %}

Wi-Fi Fingerprinting
<h1>Навчання системи</h1>
Керуйте кімнатами та запускайте навчання.

<!-- Статус навчання -->
{% if training %}
<div style="background:rgba(16,185,129,0.1); border:1px solid rgba(16,185,129,0.3);
border-radius:8px; padding:12px 18px; margin-bottom:16px;
display:flex; align-items:center; gap:10px;">
<span style="width:8px;height:8px;border-radius:50%;background:#10b981;
box-shadow:0 0 0 3px rgba(16,185,129,0.2);
animation:pulse 1.5s infinite;">

Навчання активне — збираємо відбитки для:
{{ target_room }}

{% else %}
<div style="background:rgba(107,114,128,0.08); border:1px solid #e2e4e9;
border-radius:8px; padding:12px 18px; margin-bottom:16px;">

⏸ Навчання не активне — оберіть кімнату нижче

{% endif %}

<!-- Додати кімнату -->

➕ Додати нову кімнату
<form action="/add_room" method="POST" style="display:flex; gap:10px; align-items:center;">
<input type="text" name="room_name" placeholder="Назва кімнати" style="flex:1; padding:8px 14px; border-radius:8px;
border:1px solid #e2e4e9; background:#ffffff;
color:#111827; font-size:13px;" required>
<button type="submit" class="btn btn-primary">Додати</button>
</form>

<!-- Список кімнат -->

```
<div class="card" style="margin-bottom:16px;">
<div class="section-label">📍 Кімнати для навчання</div>
{% if rooms %}
<div class="room-grid">
{% for room_id, room_name in rooms %}
<div style="display:flex; align-items:center; gap:8px;">
<a href="{{ url_for('start_training', room=room_name) }}"
class="room-chip {% if training and target_room == room_name %}room-chip-active{% endif %}"
style="flex:1;">
{{ room_name }}
</a>
<a href="{{ url_for('delete_room', room_id=room_id) }}" style="font-size:12px; color:#ef4444; padding:8px;
border-radius:6px; border:1px solid rgba(239,68,68,.2);
background:rgba(239,68,68,.08); text-decoration:none;"
onclick="return confirm('Видалити {{ room_name }}?')">
✕
</a>
</div>
{% endfor %}
</div>
{% else %}
<p style="font-size:13px; color:#6b7280;">
Кімнат ще немає — додайте першу вище.
</p>
{% endif %}
</div>

<!-- Керування -->
<div class="card">
<div class="section-label">Керування</div>
<div class="actions">
<a href="{{ url_for('stop_training') }}" class="btn btn-danger">
⏹ Зупинити навчання
</a>
<a href="{{ url_for('get_data') }}" class="btn btn-ghost">
← Назад до Dashboard
</a>
</div>
</div>

{% endblock %}
```

БІБЛІОГРАФІЧНА ДОВІДКА

Тема дипломної роботи: ***Розробка мікроконтролерної системи періодичного моніторингу параметрів повітря приміщень на базі ESP32 та Wi-Fi Fingerprinting***

Обсяг пояснювальної записки _59_ аркушів:

_9_ таблиць;

_12_ рисунків;

_3_ додаток.

Дата завершення роботи: 10 *червня 2026р*.

Підпис студента- _____ *Коваль О.А.*