**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**
**БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ**
**УКРАЇНИ**

**Факультет інформаційних технологій**

| | |
|---|---|
| **ПОГОДЖЕНО** | **ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ** |
| **Декан факультету** | **Завідувач кафедри комп'ютерних наук** |
| _____**Ігор БОЛБОТ** | _____**Белла ГОЛУБ** |
| (підпис) | (підпис) |
| «___» _____2026 р. | «___» _____2026 р. |

# БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

**на тему: «IoT-система контролю мікроклімату у приміщеннях з візуалізацією даних у реальному часі»**

Спеціальність «121 Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення»

**Гарант освітньої програми**
к.т.н., доцент _____ **Ганна ВАЙГАНГ**
(підпис)

**Керівник бакалаврської кваліфікаційної роботи**
(науковий ступінь та вчене звання) _____ **Роман РУДЕНСЬКИЙ**
(підпис)

**Виконав** _____ **Валентин МАЙБОРОДА**
(підпис)

**Київ-2026**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ**
**Факультет інформаційних технологій**

**ЗАТВЕРДЖУЮ**
**Завідувач кафедри комп'ютерних наук**
к.т.н., доцент _____ Белла ГОЛУБ
(підпис)
«___» _____ 2025 р.

**ЗАВДАННЯ**
**ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ**
**ЗДОБУВАЧУ**
**Майбороді Валентину Андрійовичу**
(прізвище, ім'я, по батькові)

Спеціальність: 121 – «Інженерія програмного забезпечення»
Освітньо-професійна програма: «Інженерія програмного забезпечення»
Тема бакалаврської кваліфікаційної роботи
«IoT-система контролю мікроклімату у приміщеннях з візуалізацією даних у реальному часі».
затверджена наказом від «_19_» ___грудня___ 2025 р. № _3196_ «С»
Термін подання завершеної роботи на кафедру «_22_» ____травня___ 2026 р.
Вихідні дані до бакалаврської кваліфікаційної роботи: персональний комп'ютер з ОС Windows 10/11; локальний демонстраційний MQTT-брокер; програмний симулятор IoT-вузла; база даних SQLite для локального режиму та можливість міграції на InfluxDB; мова програмування Python; фреймворк Streamlit; результати аналізу предметної області та нормативні діапазони параметрів мікроклімату.
Перелік питань, які підлягають дослідженню: системний аналіз предметної області; аналіз функціональних та нефункціональних вимог; моделювання предметної області й програмної архітектури; проєктування інформаційної бази; розробка та реалізація програмних модулів збору, передачі, збереження, аналізу та візуалізації телеметрії; тестування системи; визначення вимог до експлуатації.
Перелік графічних документів: контекстна діаграма; діаграма варіантів використання; ER-діаграма; діаграми класів, пакетів, компонентів, розміщення; алгоритм обробки телеметрії; скріншоти інтерфейсу та результатів тестування.
Дата видачі завдання «_19_» ____грудня____ 2025 р.

**Керівник бакалаврської кваліфікаційної роботи** _____ **Роман РУДЕНСЬКИЙ**
(підпис)

**Завдання взяв до виконання** _____ **Валентин МАЙБОРОДА**
(підпис)

# ЗМІСТ

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

| Позначення | Пояснення |
|---|---|
| API | Application Programming Interface, інтерфейс програмної взаємодії |
| CSV | Comma-Separated Values, табличний формат експорту даних |
| DB | Database, база даних |
| ESP32 | мікроконтролер з вбудованим Wi-Fi/Bluetooth модулем |
| GPIO | General Purpose Input/Output, універсальний цифровий вивід |
| HTTP | HyperText Transfer Protocol |
| IoT | Internet of Things, Інтернет речей |
| JSON | JavaScript Object Notation, формат структурованих даних |
| MQTT | Message Queuing Telemetry Transport, протокол обміну телеметрією |
| NDIR | Non-Dispersive Infrared, інфрачервоний принцип вимірювання CO2 |
| QoS | Quality of Service, рівень гарантії доставки повідомлень |
| SQL | Structured Query Language, мова запитів до баз даних |
| UI | User Interface, інтерфейс користувача |
| UML | Unified Modeling Language, уніфікована мова моделювання |

# ВСТУП

У сучасних умовах цифрової трансформації програмні системи моніторингу середовища стають важливою складовою інфраструктури житлових, освітніх та офісних приміщень. Параметри мікроклімату безпосередньо впливають на самопочуття людини, продуктивність праці, якість навчального процесу та енергоефективність будівель. Традиційний ручний контроль температури, вологості й якості повітря не забезпечує достатньої оперативності, не формує історії вимірювань і не дозволяє вчасно приймати рішення щодо вентиляції або зміни режимів експлуатації приміщення.

Актуальність роботи полягає у необхідності створення доступного програмного рішення, здатного об’єднати збір телеметричних даних, їх передавання за мережевим протоколом, збереження в інформаційній базі та представлення користувачу у зрозумілій формі. Для спеціальності 121 «Інженерія програмного забезпечення» важливо, що об’єктом розробки є не окремий датчик або електронна схема, а комплекс програмних модулів, які забезпечують життєвий цикл даних: від формування телеметрії до аналітичного звіту та команд керування.

Метою бакалаврської кваліфікаційної роботи є проєктування та реалізація програмного забезпечення IoT-системи контролю мікроклімату у приміщеннях з візуалізацією даних у реальному часі. Розроблена система повинна забезпечувати приймання телеметрії, перевірку коректності даних, збереження історії вимірювань, обчислення статусу комфорту, прогнозування наближення показників до граничних зон, формування керуючих команд і відображення результатів у веб-інтерфейсі.

Для досягнення мети необхідно розв’язати такі завдання: проаналізувати предметну область і існуючі рішення; сформувати вимоги до програмної системи; розробити моделі предметної області та програмної архітектури; спроєктувати інформаційну базу; реалізувати симулятор IoT-вузла, MQTT-подібний обмін повідомленнями, серверну частину, модуль керування та

dashboard; додати контроль якості вхідних даних і прогнозну оцінку зміни CO2 та температури; провести тестування й описати склад інсталяційного пакету.

Об’єктом дослідження є процеси програмного моніторингу параметрів мікроклімату в системах Інтернету речей. Предметом дослідження є методи, моделі та програмні засоби збору, передавання, збереження, аналізу і візуалізації телеметричних даних у локальній IoT-системі.

Методи та технології, використані у роботі: системний аналіз для формування вимог; UML-моделювання для опису архітектури; ER-моделювання для проєктування інформаційної бази; MQTT-подібна модель publish/subscribe для обміну повідомленнями; формат JSON для структурування телеметрії; Python для реалізації серверної логіки; SQLite для локального збереження даних; Streamlit і Plotly для побудови веб-інтерфейсу; розрахунок тренду для короткострокового прогнозування; модульне тестування для перевірки логіки керування та валідації вхідних даних.

Апробація програмного додатку здійснювалася шляхом локального запуску програмного стенда без фізичного обладнання. Симулятор IoT-вузла формував телеметрію, локальний MQTT-подібний брокер передавав повідомлення, серверний модуль виконував валідацію та записував дані до бази, а dashboard відображав поточні показники, графіки, журнал вимірювань, прогнозні значення, керуючі дії та результати тестування. Такий спосіб апробації відповідає умовам, коли реальні ESP32 та сенсори відсутні, але необхідно підтвердити працездатність програмної архітектури.

Пояснювальна записка складається з анотації українською та англійською мовами, переліку умовних позначень, вступу, чотирьох розділів, висновків, списку використаних джерел і додатків. У першому розділі виконано системний аналіз предметної області. У другому розділі розроблено інформаційне та програмне забезпечення на рівні моделей. У третьому розділі описано реалізацію програмних модулів. У четвертому розділі наведено тестування, вимоги до запуску й склад інсталяційного пакету.

# 1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

## 1.1 Опис предметної області

Предметна область системи охоплює моніторинг стану повітря в закритих приміщеннях за допомогою програмно керованих сенсорних вузлів. У класичному варіанті така система включає датчики температури, вологості, атмосферного тиску та концентрації CO2, мікроконтролерний вузол, мережевий протокол передачі, серверну частину, базу даних і користувацький інтерфейс. У межах цієї кваліфікаційної роботи фізичний сенсорний вузол замінено симулятором, але всі інші програмні процеси реалізовано як у реальній IoT-архітектурі.

Важливість контролю мікроклімату пояснюється тим, що якість повітря не завжди може бути оцінена суб’єктивно. Наприклад, зростання концентрації CO2 в аудиторії або офісі часто відбувається поступово, користувачі помічають лише втому або зниження концентрації, але не мають інструмента для аналізу реальної причини. Програмна система має перетворювати вимірювання на зрозумілу інформацію: показники, графіки, статус комфорту, журнал подій і рекомендації.

З точки зору програмної інженерії, система мікроклімату є прикладом інформаційно-управляючої системи. Вона отримує вхідні дані, перевіряє їх коректність, зберігає, аналізує та формує вихідну реакцію. Вихідною реакцією може бути повідомлення користувачу, автоматичний звіт, CSV-файл або команда керування виконавчим пристроєм. Саме тому основний акцент роботи перенесено з апаратної збірки на модулі програмного забезпечення.

Типовими користувачами системи є оператор приміщення, адміністратор програмного стенда та технічний фахівець, який може розгортати систему або аналізувати журнали. Оператор працює переважно з веб-інтерфейсом, адміністратор налаштовує пороги й контролює запуск сервісів, а технічний фахівець аналізує помилки, пакети даних і результати тестування.


Рис. 1.1 Контекстна модель програмної системи

Контекстна модель на рис. 1.1 показує межі програмної системи та зовнішні взаємодії. Центральними програмними компонентами є MQTT-брокер, сервер обробки, база даних і веб-панель. Симулятор IoT-вузла виконує роль джерела телеметрії, а користувач отримує інформацію через браузер.

## 1.2 Аналіз вимог до програмної системи

Оскільки фізичне обладнання відсутнє, окремою вимогою є підтримка демонстраційного режиму. Демонстраційний режим не повинен бути просто випадковим графіком у dashboard; він має відтворювати архітектуру реальної IoT-системи. Тому симулятор надсилає дані через локальний MQTT-подібний канал, сервер приймає та перевіряє їх, база даних зберігає історію, а веб-інтерфейс читає дані з бази. Додатково система повинна підтримувати контрольовані сценарії помилок, щоб перевірити реакцію на некоректні або застарілі пакети телеметрії.

Оскільки фізичне обладнання відсутнє, окремою вимогою є підтримка демонстраційного режиму. Демонстраційний режим не повинен бути просто випадковим графіком у dashboard; він має відтворювати архітектуру реальної IoT-системи. Тому симулятор надсилає дані в MQTT, сервер приймає їх, база даних зберігає історію, а веб-інтерфейс читає дані з бази. Це забезпечує відповідність основним сценаріям дипломної роботи без залежності від апаратної платформи.

Таблиця 1.1

Функціональні вимоги до програмної системи

| Код | Вимога | Критерій виконання |
|---|---|---|

| | | |
|---|---|---|
| FR-01 | Система повинна приймати телеметричні дані у форматі JSON | Сервер приймає повідомлення з полями device_id, location, timestamp, temperature, humidity, pressure, co2 |
| FR-02 | Система повинна перевіряти коректність вхідних даних | Некоректні повідомлення не записуються до бази та фіксуються у журналі подій |
| FR-03 | Система повинна зберігати історію вимірювань | Кожен валідний пакет зберігається у таблиці measurements |
| FR-04 | Система повинна відображати поточні показники | Dashboard показує останні значення температури, вологості, тиску, CO2, статус ризику та керуючу дію |
| FR-05 | Система повинна будувати графіки за період | Користувач бачить динаміку параметрів у часовому інтервалі |
| FR-06 | Система повинна формувати керуючі дії | При перевищенні CO2 формується fan_on, після стабілізації - fan_off, а при прогнозованому наближенні до порога - fan_pre_on |
| FR-07 | Система повинна експортувати дані | Користувач може завантажити CSV-файл або текстову виписку |
| FR-08 | Система повинна містити тести | Модульні тести перевіряють валідацію, гістерезис, прогнозне керування та реакцію на некоректні пакети |
| FR-09 | Система повинна підтримувати прогнозне керування | За останніми вимірюваннями розраховується темп зміни CO2 і температури та прогноз на 5 хвилин |
| FR-10 | Симулятор повинен підтримувати | Доступні режими нормальної роботи, |

| | | |
|---|---|---|
| | контрольовані сценарії відмов | стрибка CO2, шумного каналу, періодичних відмов, значень поза діапазоном і застарілої часової мітки |

Таблиця 1.2

Нефункціональні вимоги до програмної системи

| Код | Вимога | Пояснення |
|---|---|---|
| NFR-01 | Переносимість | Проєкт має запускатися на Windows 10/11 з portable Python або встановленим Python |
| NFR-02 | Модульність | Код поділено на app, server, simulator, iot, firmware, tests |
| NFR-03 | Зрозумілість інтерфейсу | Головні показники відображаються у вигляді метрик, графіків і таблиці логів |
| NFR-04 | Надійність | У журналі фіксуються запуск системи, приймання даних, помилки валідації та керуючі дії |
| NFR-05 | Масштабованість | Архітектура дозволяє замінити SQLite на InfluxDB та симулятор на реальний ESP32 |
| NFR-06 | Тестованість | Ключові функції не залежать від UI і перевіряються модульними тестами |
| NFR-07 | Стійкість до некоректних даних | Помилковий пакет не зупиняє роботу системи і не потрапляє до таблиці вимірювань |

Вимоги в табл. 1.1 та табл. 1.2 відображають пріоритети саме програмної інженерії. Найважливішим є не факт наявності датчика, а наявність відтворюваного програмного процесу, який можна запустити, протестувати, продемонструвати й розширити.

## 1.3 Моделювання предметної області

Моделювання предметної області дозволяє представити систему до початку реалізації та зменшити ризик невідповідності між задумом і програмним продуктом. У цій роботі застосовано UML-моделювання, яке дає змогу описати користувацькі сценарії, структуру модулів, залежності між компонентами та розміщення програмних частин на вузлах виконання.

Основний користувацький сценарій полягає у перегляді поточних показників мікроклімату. Система отримує телеметрію, автоматично оновлює графіки, показує статус комфорту і дозволяє сформувати звіт. Адміністратор, на відміну від звичайного користувача, контролює запуск компонентів, переглядає журнали та може змінити пороги спрацювання. Симулятор IoT-вузла виступає технічним актором, який передає дані та отримує команди керування.

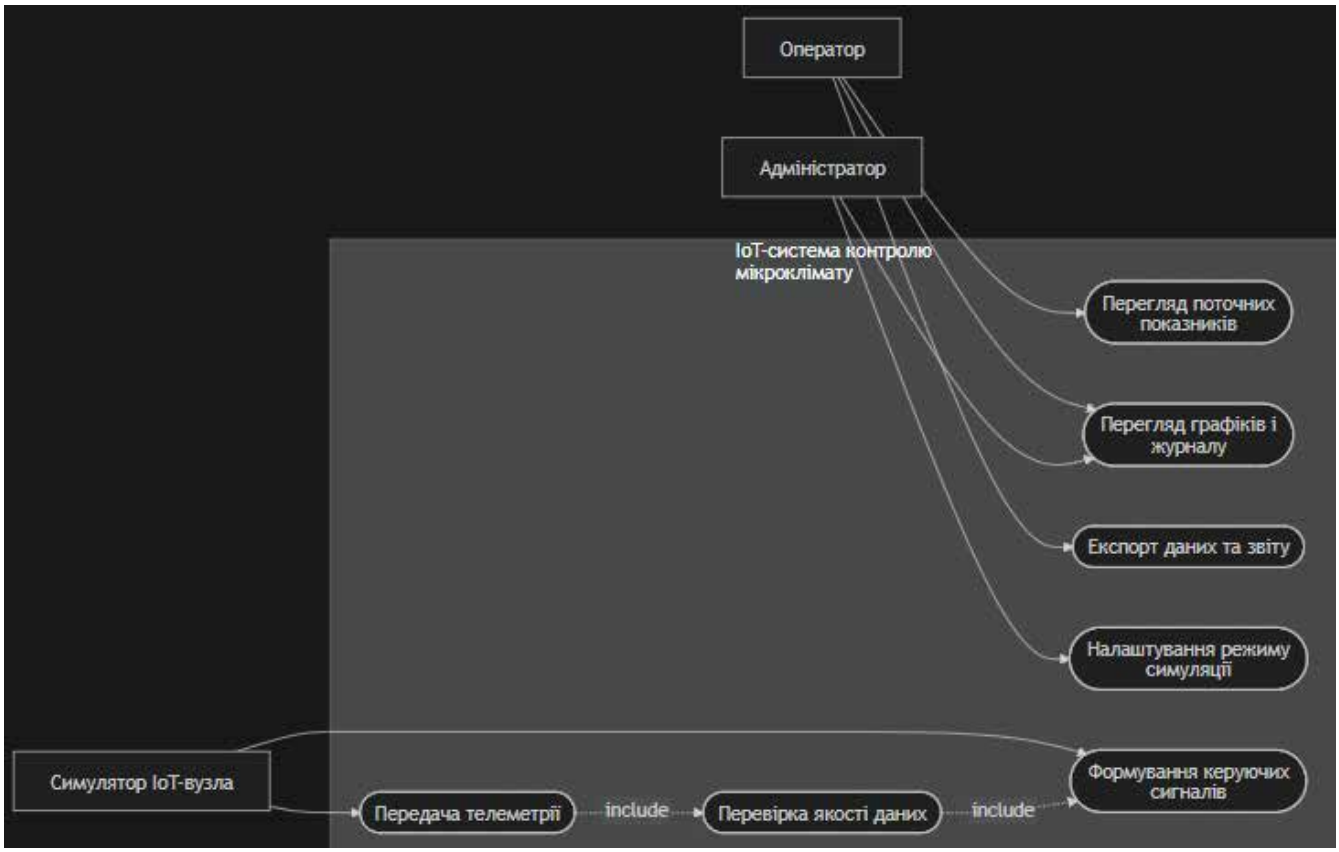


Рис. 1.2 Діаграма варіантів використання програмної системи

Діаграма варіантів використання на рис. 1.2 фіксує взаємодію користувача, адміністратора та симулятора з системою. Вона показує, що програмний продукт має не лише відображати дані, а й підтримувати повний цикл обробки: приймання, перевірка, збереження, аналіз, звітність та керування.

## 1.4 Огляд інформаційних джерел та існуючих рішень

Існуючі рішення для моніторингу мікроклімату можна умовно поділити на три групи: промислові SCADA-системи, готові Smart Home платформи та відкриті IoT-рішення на базі мікроконтролерів. Промислові системи характеризуються високою надійністю, але мають значну вартість, складність впровадження та надлишковість для невеликих приміщень. Smart Home рішення зручні для користувача, але часто мають закритий код і обмежені можливості налаштування.

Відкриті IoT-рішення на базі ESP32, MQTT, Python та веб-панелей є найбільш придатними для бакалаврської кваліфікаційної роботи з інженерії програмного забезпечення. Вони дозволяють показати повний цикл розробки: постановку вимог, моделювання, реалізацію модулів, тестування та підготовку інсталяційного пакету. Крім того, такі системи легко демонструвати у локальному режимі без хмарних сервісів і без фізичних сенсорів.

Таблиця 1.3

Порівняння підходів до побудови систем моніторингу

| **Критерій** | **SCADA** | **Smart Home** | **Власна IoT-система** |
|---|---|---|---|
| Вартість | Висока | Середня | Низька або середня |
| Гнучкість | Середня | Низька | Висока |
| Доступ до коду | Обмежений | Зазвичай відсутній | Повний |
| Придатність для навчального проєкту | Низька | Середня | Висока |
| Можливість симуляції | Обмежена | Обмежена | Повна |
| Основні технології | Modbus, OPC, SCADA | Zigbee, BLE, хмара | MQTT, Python, SQLite/InfluxDB, Streamlit |

За результатами порівняння доцільно обрати власну IoT-систему. Вона не поступається готовим рішенням за базовими функціями моніторингу, але краще відповідає вимогам кваліфікаційної роботи, оскільки дозволяє

продемонструвати власну програмну реалізацію, структуру коду, інформаційну базу та протокол тестування.

## 1.5 Постановка завдання

Вхідними даними системи є JSON-повідомлення телеметрії, що містить ідентифікатор пристрою, місце розташування, часову мітку, температуру, вологість, тиск і рівень CO2. Вихідними даними є поточні показники в інтерфейсі, графіки динаміки, таблиця останніх вимірювань, CSV-файл, текстовий звіт, журнали подій, рівень ризику, прогнозні значення та команди керування вентиляцією або рекомендації щодо температурного режиму.

Окрім реактивної логіки керування, система повинна підтримувати елементи прогнозного керування. Для цього за останніми валідними вимірюваннями визначається темп зміни CO2 та температури, після чого обчислюється прогноз на наступні 5 хвилин. Якщо прогноз показує наближення параметра до граничної зони, система формує дію заздалегідь: для CO2 - попереджувальне ввімкнення примусової вентиляції, для температури - зменшення інтенсивності обігріву або підготовка кондиціонування.

Вхідними даними системи є JSON-повідомлення телеметрії, що містить ідентифікатор пристрою, часову мітку, температуру, вологість, тиск і рівень CO2. Вихідними даними є поточні показники в інтерфейсі, графіки динаміки, таблиця останніх вимірювань, CSV-файл, текстовий звіт, журнали подій та команди керування вентиляцією.

Критеріями успішності є: запуск усіх модулів локально; отримання телеметрії через MQTT-подібний канал; запис валідних даних у базу; відхилення некоректних пакетів; відображення показників у dashboard; формування команд fan_on, fan_off і fan_pre_on за порогами та прогнозом; проходження тестів; наявність опису інсталяційного пакету та інструкції запуску.

Критеріями успішності є: запуск усіх модулів локально; отримання телеметрії через MQTT; запис даних у базу; відображення показників у

dashboard; формування команд fan_on/fan_off за порогами; проходження тестів; наявність опису інсталяційного пакету та інструкції запуску.

# 2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 2.1 Логічна модель даних у вигляді ER-діаграми

Інформаційне забезпечення системи призначене для збереження історії вимірювань, ідентифікації пристроїв, фіксації подій та підтримки порогів керування. Дані мікроклімату є часовими рядами, тому для виробничого режиму доцільно використовувати InfluxDB. Водночас для локального демонстраційного стенда обрано SQLite, оскільки вона не потребує окремого сервера, легко переноситься разом з проєктом і повністю достатня для демонстрації функціональності.

Логічна модель містить таблиці devices, measurements, events і control_settings. Таблиця devices описує зареєстровані джерела телеметрії. Таблиця measurements зберігає значення параметрів мікроклімату, прогнозні значення, рівень ризику та сформовану керуючу дію. Таблиця events фіксує помилки, попередження і команди керування. Таблиця control_settings зберігає порогові значення, які використовуються логікою прийняття рішень.

Для підтримки прогнозного керування таблицю measurements розширено службовими полями, які зберігають прогнозований рівень CO2 і температуру, темпи їх зміни, орієнтовний час до досягнення порогових значень, сформовану керуючу дію, пояснення причини керування та рівень ризику. Це дає змогу аналізувати не лише фактичний стан мікроклімату, а й очікувану динаміку параметрів.

Модель є розширюваною: за необхідності до неї можна додати таблиці users, rooms або sensor_types. Проте для поточного етапу розробки надмірне ускладнення не потрібне, оскільки головним завданням є підтвердження повного програмного циклу обробки телеметрії.

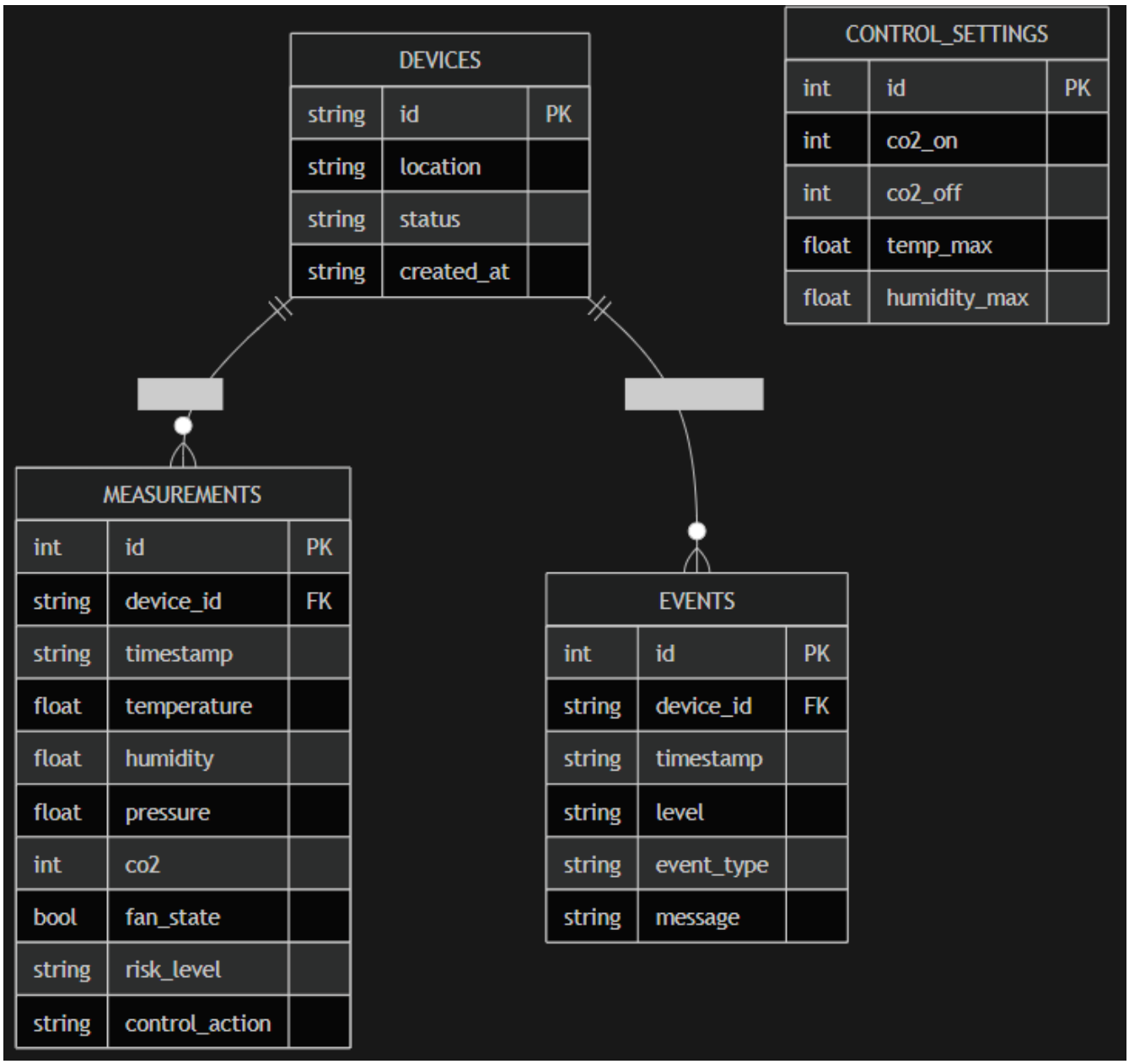



Рис. 2.1 ER-діаграма інформаційної бази системи

Таблиця 2.1

Опис основних сутностей інформаційної бази

| Сутність | Призначення | Ключові поля |
|---|---|---|
| devices | Ідентифікація джерел телеметрії | id, location, status, created_at |
| measurements | Збереження показників мікроклімату та результатів прогнозного керування | id, device_id, timestamp, temperature, humidity, pressure, co2, fan_state, predicted_co2, predicted_temperature, risk_level, climate_action |

| | | |
|---|---|---|
| events | Журналювання системних подій, помилок і керуючих дій | id, device_id, timestamp, level, event_type, message |
| control_settings | Збереження порогів керування | co2_on, co2_off, temp_max, humidity_max |

Модель на рис. 2.1 відповідає вимогам логічного проєктування, оскільки відокремлює довідкову інформацію про пристрій від багаторазових вимірювань. Зв’язок devices - measurements має кардинальність один-до-багатьох, адже один сенсорний вузол може формувати багато записів телеметрії.

## 2.2 Діаграма класів та кооперацій

Діаграма класів у цій роботі подається як діаграма програмних модулів, оскільки реалізація виконана мовою Python і частина логіки винесена в окремі файли замість класичних об’єктно-орієнтованих класів. Такий підхід не суперечить меті моделювання: діаграма показує відповідальність окремих частин програми та залежності між ними.

Ключовими модулями є MicroclimateRuntime, MinimalMQTTBroker, ESP32Simulator, TelemetrySubscriber, DatabaseRepository, ControlLogic і Dashboard. Разом вони утворюють кооперацію, в якій симулятор генерує пакет, локальний publish/subscribe-брокер передає його серверному підписнику, база зберігає валідні дані, логіка керування оцінює пороги та прогноз, а dashboard читає останні дані для відображення.

Модуль ControlLogic поєднує реактивне та прогнозне керування. Реактивна частина використовує гістерезис для уникнення частого перемикання вентиляції. Прогнозна частина оцінює темп зміни CO2 і температури за останніми валідними вимірюваннями та визначає, чи може параметр досягти порогового значення протягом наступних 5 хвилин.

Для зменшення зв’язаності модулі не повинні напряму звертатися один до одного без потреби. Наприклад, dashboard не отримує дані від симулятора, а читає їх через інформаційну базу. Це дозволяє замінити джерело даних без зміни інтерфейсу користувача.

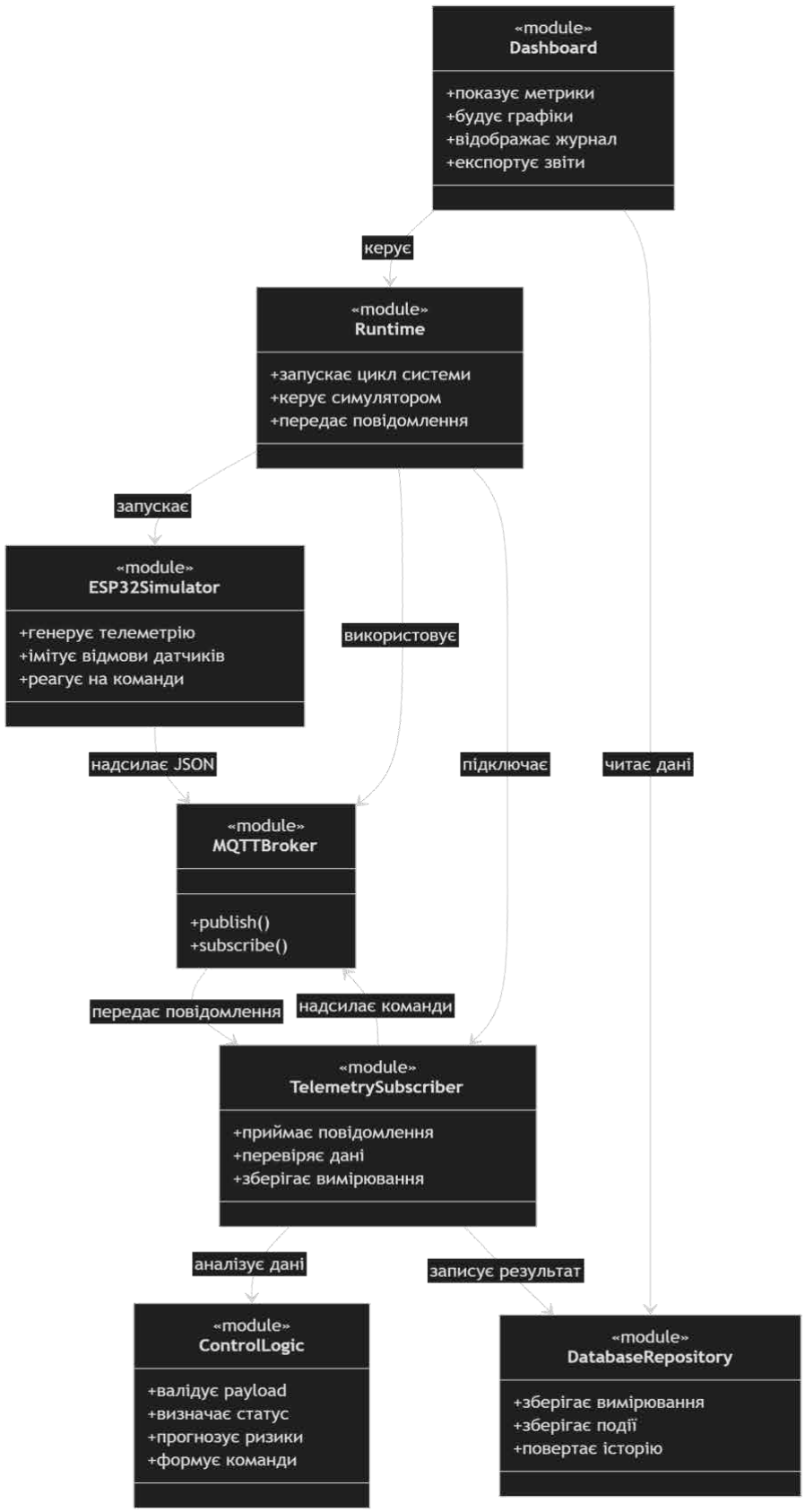



Рис. 2.2 Діаграма класів/модулів програмної системи

Таблиця 2.2

Відповідальність програмних модулів

| Модуль | Файл | Відповідальність |
|---|---|---|

| | | |
|---|---|---|
| Dashboard | app/dashboard.py | Візуалізація показників, графіків, журналу, прогнозу, експорту та звіту |
| MicroclimateRuntime | server/runtime.py | Запуск фонового контуру симулятор - брокер - сервер - база - dashboard |
| TelemetrySubscriber | server/mqtt_subscriber.py | Приймання повідомлень, валідація JSON, запис у базу та передавання даних до модуля керування |
| DatabaseRepository | server/db.py | Ініціалізація SQLite, вставка вимірювань, подій, прогнозних значень і запити історії |
| ControlLogic | server/control_logic.py | Валідація пакетів, оцінка статусу комфорту, гістерезисне керування вентиляцією, розрахунок прогнозу CO2 і температури, формування попереджувальних дій |
| ESP32Simulator | simulator/esp32_simulator.py | Генерація телеметрії, імітація відмов датчиків і реакція на керуючі команди |
| MQTTBroker | iot/mqtt_minimal.py | Локальний MQTT-подібний publish/subscribe-канал для демонстрації |

## 2.3 Діаграма пакетів

Пакетна структура проєкту визначає фізичну організацію коду в каталогах. Це важливо для супроводження, оскільки кожен каталог має окрему відповідальність і може розвиватися незалежно. Наприклад, каталог app містить лише інтерфейс, server - серверну логіку, simulator - модель пристрою, tests - перевірки, firmware - ескіз прошивки для майбутнього реального ESP32.

Такий поділ також полегшує демонстрацію програмного продукту на попередньому захисті. Замість одного великого файлу користувач бачить структурований інсталяційний пакет, де кожний модуль має зрозуміле призначення. Це відповідає вимогам методичних вказівок щодо визначення складу інсталяційного пакету та демонстрації програмної системи.

Пакетний підхід зменшує ризик помилок при розширенні. Наприклад, якщо надалі потрібно додати автентифікацію користувачів, це можна зробити в окремому модулі, не змінюючи симулятор або MQTT-брокер.

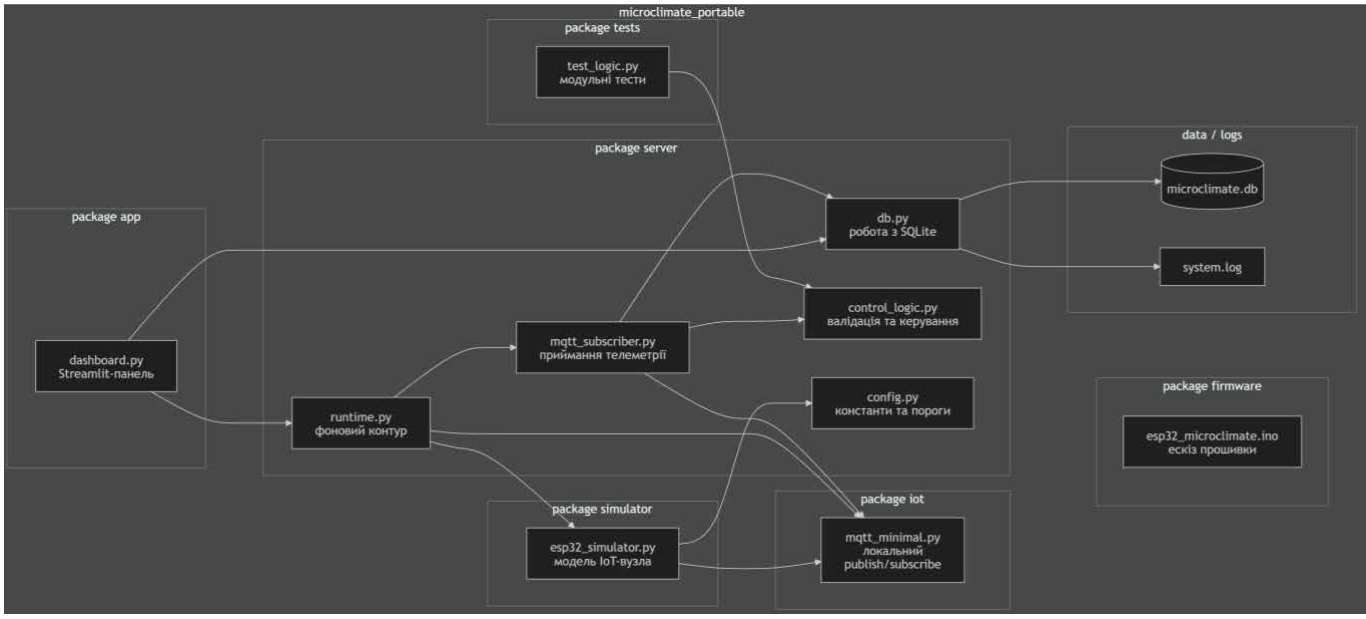



Рис. 2.3 Діаграма пакетів програмного проєкту

## 2.4 Діаграма компонентів

У межах розробки виділяються компонент симулятора, локальний MQTT-подібний брокер, сервер телеметрії, база даних, веб-інтерфейс і компонент керування з гістерезисом та прогнозом. Кожен компонент має чіткий інтерфейс взаємодії: топіки обміну повідомленнями, SQL-запити або HTTP-доступ до Streamlit.

Наявність MQTT-брокера між джерелом даних і сервером зменшує зв’язаність компонентів. Симулятор не знає, як саме дані зберігаються або відображаються; він лише публікує телеметрію. Сервер не знає, чи джерело даних є реальним ESP32 або програмною моделлю; він отримує однаковий

JSON-пакет. Це підвищує масштабованість і спрощує майбутню заміну симулятора на обладнання.

Компонентний поділ також підвищує тестованість. Окремо можна перевірити валідацію пакета, окремо - логіку гістерезису, окремо - запити до бази, а окремо - візуальний рівень. Така декомпозиція відповідає принципам інженерії програмного забезпечення.

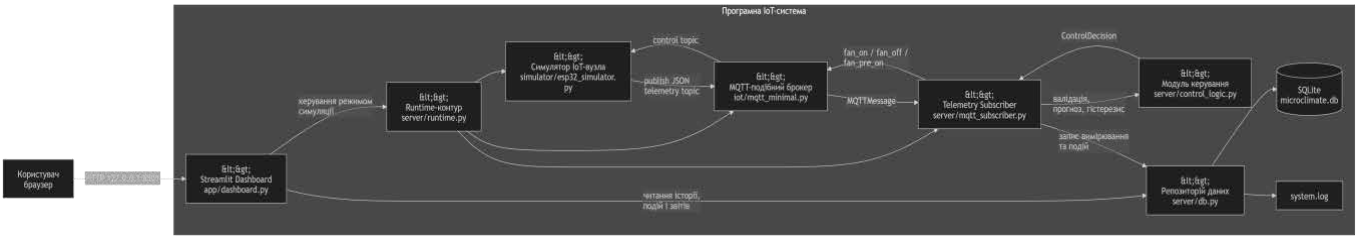



Рис. 2.4 Діаграма компонентів IoT-системи

## 2.5 Діаграма розміщення

Діаграма розміщення необхідна для опису топології системи та вимог до вузлів виконання. У демонстраційному режимі всі компоненти можуть працювати на одній робочій станції Windows: локальний брокер, сервер, симулятор, база даних і Streamlit dashboard. Користувач відкриває інтерфейс у браузері за адресою 127.0.0.1:8501.

У виробничому режимі система може бути розгорнута інакше: MQTT-брокер та InfluxDB працюють у Docker-контейнерах або на окремому локальному сервері, реальний ESP32 передає телеметрію через Wi-Fi, а користувачі відкривають dashboard з робочих станцій або мобільних пристроїв. Така модель показує, що локальна демонстрація не обмежує майбутню експлуатацію.

Використання локального розгортання є виправданим для захисту, оскільки зменшує залежність від інтернету, сторонніх сервісів і реального обладнання. При цьому програмний продукт має достатньо повну структуру для демонстрації вимог, моделей, тестів і інсталяційного пакету.

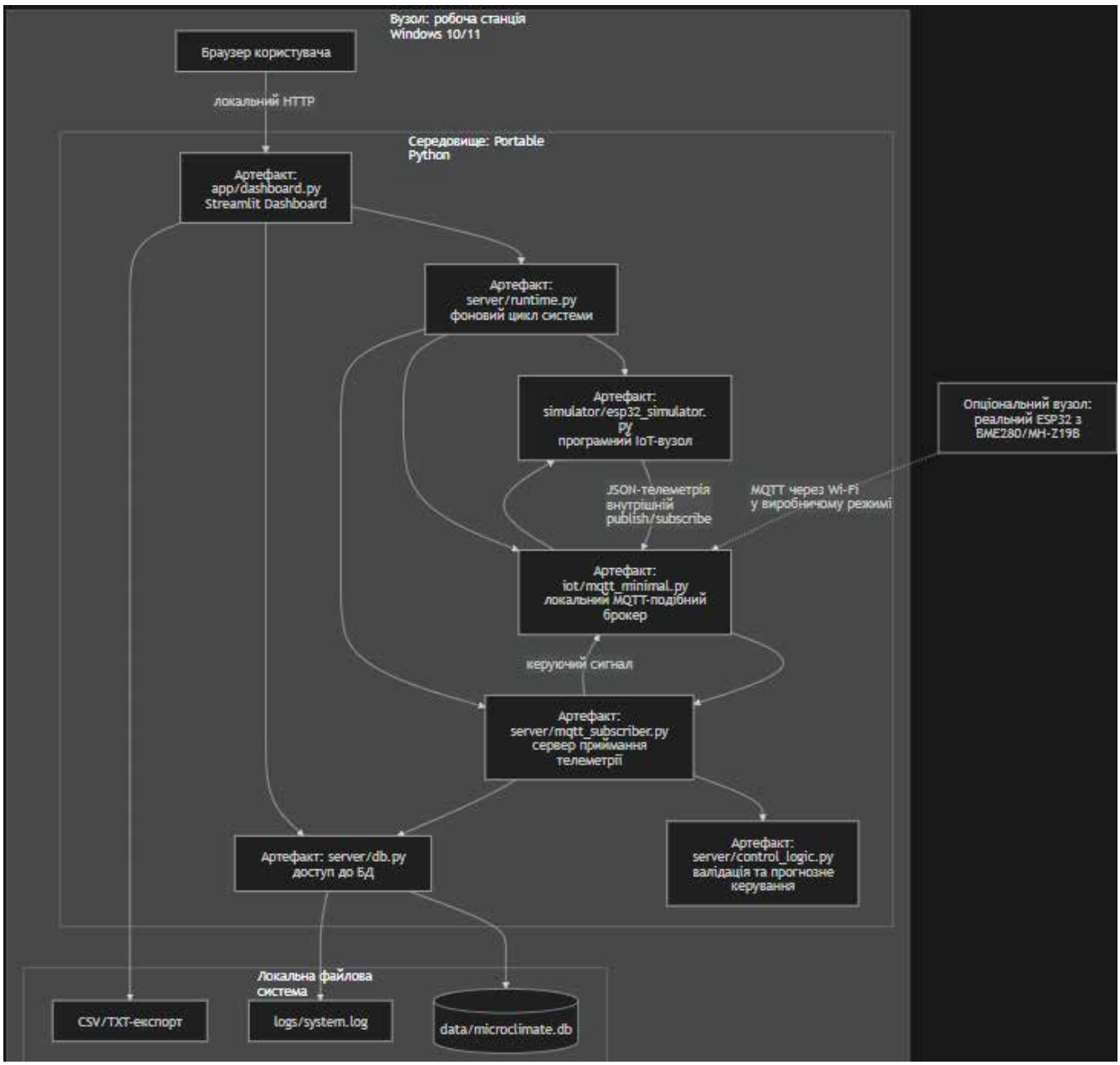



Рис. 2.5 Діаграма розміщення програмної системи

## 2.6 Проєктування алгоритму обробки телеметрії

Алгоритм роботи серверної частини починається з очікування повідомлення у топіку телеметрії. Після отримання пакета виконується декодування JSON і перевірка наявності обов’язкових полів, типів значень, коректності часової мітки та допустимих діапазонів. Якщо дані некоректні, система не записує їх до таблиці measurements, а формує запис у журналі events. Якщо пакет валідний, сервер зберігає вимірювання та передає значення до модуля керування.

Модуль керування оцінює показники за встановленими порогами та останніми валідними вимірюваннями. Для CO2 використовується гістерезис: команда fan_on формується при досягненні верхнього порогу, а fan_off - лише після зниження нижче нижнього порогу. Додатково розраховується темп зміни CO2 і температури, після чого формується прогноз на наступні 5 хвилин. Якщо прогноз показує наближення до граничної зони, система може сформувати попереджувальну дію до фактичного перевищення.

Відображення даних у dashboard відбувається незалежно від приймання повідомлень. Інтерфейс запитує історію з бази і будує графіки за останні вимірювання, показує рівень ризику, прогнозні значення та сформовану керуючу дію. Такий поділ дозволяє серверній логіці продовжувати роботу навіть тоді, коли користувач не відкрив веб-інтерфейс.

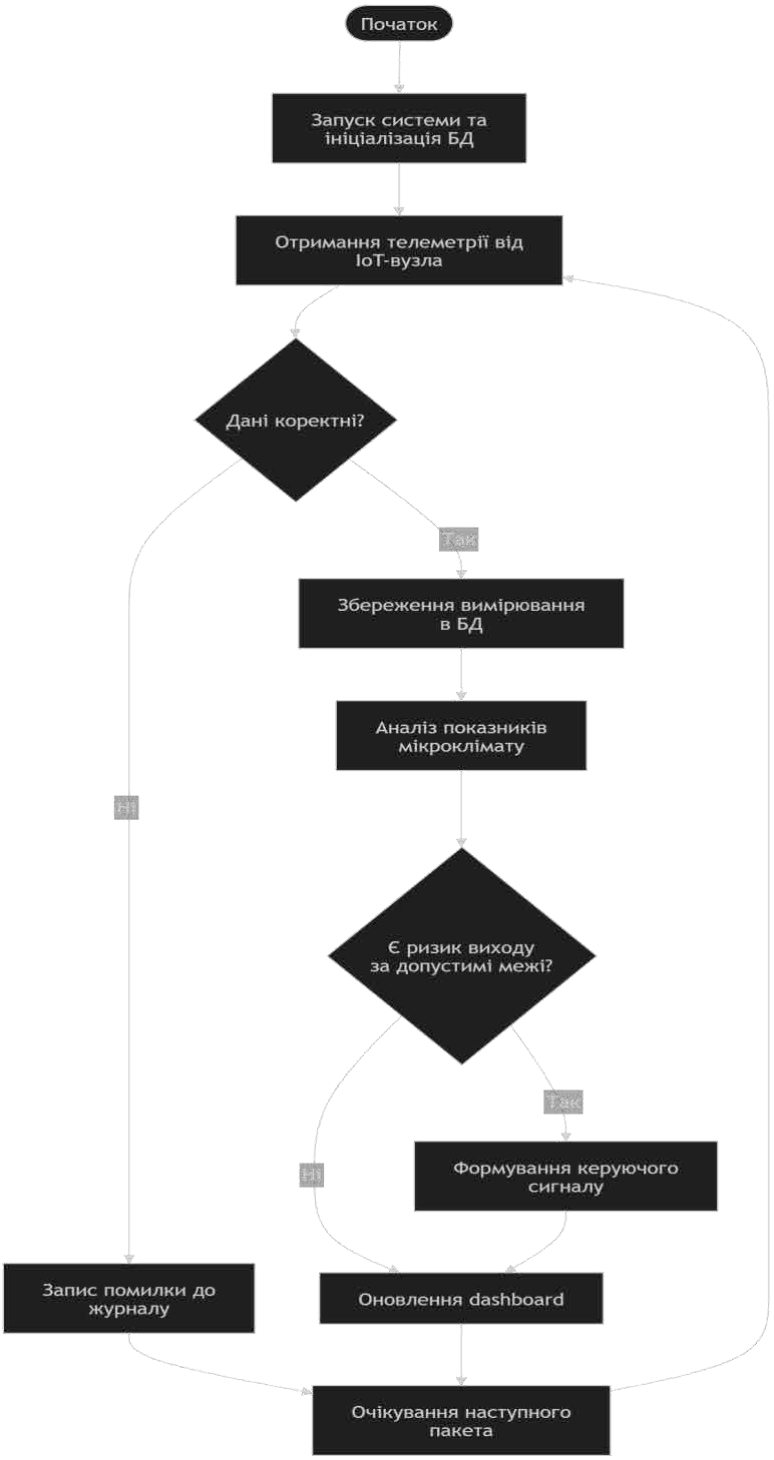



Рис. 2.6 Блок-схема алгоритму обробки телеметрії та керування

Таблиця 2.3

Порогові значення для оцінки параметрів

| Параметр | Нормальний діапазон | Дія при відхиленні |
|---|---|---|
| Температура | 21-25 °C | При наближенні до верхньої межі - рекомендація зменшити обігрів або підготувати кондиціонування; при |

| | | |
|---|---|---|
| | | перевищенні - увімкнути кондиціонування |
| Вологість | 40-60 % | Виведення попередження про сухе або надмірно вологе повітря |
| CO2 | до 800 ppm - комфортно; 800-1000 ppm - зона уваги; >=1000 ppm - погіршення якості повітря | Формування fan_on, fan_off або попереджувальної команди fan_pre_on за прогнозом |
| Тиск | інформаційний показник | Використовується для аналітики та звіту |

## 2.7 Проєктування інтерфейсу користувача

Інтерфейс користувача проєктується як веб-панель моніторингу, доступна через браузер. Основний екран має містити блок поточних метрик, графіки динаміки, текстову виписку, кнопки експорту та журнал останніх вимірювань. Такий підхід відповідає принципу прозорості інтерфейсу: користувач одразу бачить головну інформацію, а деталізація доступна нижче на сторінці.

Вибір Streamlit обґрунтований тим, що фреймворк дозволяє швидко перетворити Python-код у веб-додаток, не створюючи окремий фронтенд на JavaScript. Для бакалаврської роботи це дає можливість зосередитися на програмній логіці, потоках даних і тестуванні, одночасно отримавши повноцінний інтерактивний dashboard.

Інтерфейс не повинен перевантажувати користувача другорядними елементами. Тому на першому екрані розміщено лише найважливіші показники: температура, вологість, CO2 і статус середовища. Інші елементи - графіки, звіт, журнал і тести - розташовані нижче.

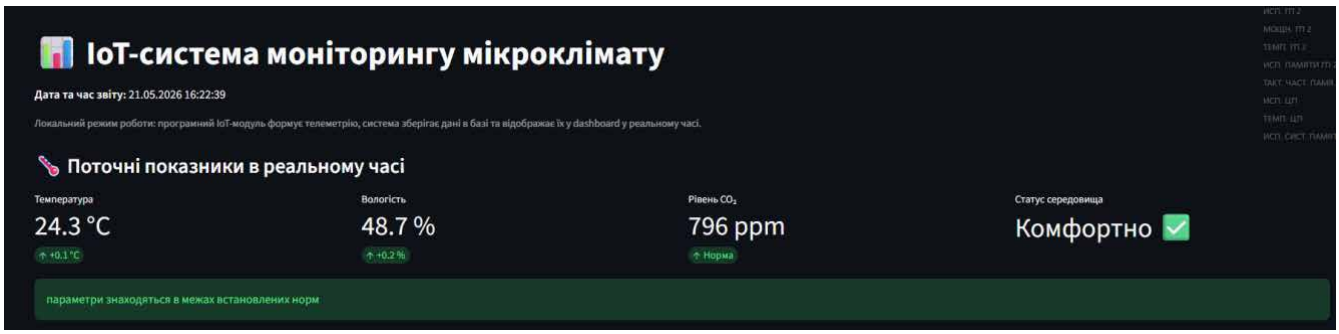


Рис. 2.7 Прототип головного екрана dashboard

Прототип головного екрана на рис. 2.7 демонструє розміщення основних метрик: температури, вологості, CO2 і статусу середовища. У фінальній реалізації ці значення беруться з бази даних, а не генеруються безпосередньо у файлі dashboard.

# 3 РОЗРОБКА ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 3.1 Вибір системи управління інформаційною базою

Для локального демонстраційного режиму обрано SQLite. Основними перевагами цього вибору є відсутність необхідності встановлення окремого сервера, простота перенесення файлу бази разом з проєктом, достатня продуктивність для навчального стенда та підтримка SQL. База створюється автоматично при першому запуску серверного модуля, що знижує кількість ручних дій для користувача.

У дипломній роботі також передбачено можливість переходу на InfluxDB. Це важливо, оскільки InfluxDB є спеціалізованою базою часових рядів і краще підходить для виробничих IoT-систем з великою кількістю вимірювань. Тому у складі інсталяційного пакету передбачено docker-compose.yml, який дозволяє підняти Mosquitto та InfluxDB для розширеного режиму.

Розмежування локального і виробничого режимів є обґрунтованим інженерним рішенням. SQLite використовується для демонстрації без зайвих залежностей, а InfluxDB залишається цільовою платформою для масштабування. Завдяки модульній структурі заміна механізму збереження не потребує переписування dashboard або симулятора.

Таблиця 3.1

Порівняння SQLite та InfluxDB для проєкту

| **Критерій** | **SQLite** | **InfluxDB** |
|---|---|---|
| Тип | Вбудована реляційна БД | База часових рядів |
| Складність встановлення | Мінімальна | Потребує сервера або Docker |
| Придатність для демонстрації | Висока | Середня |
| Придатність для масштабування | Середня | Висока |
| Файл інсталяції | microclimate.db | docker-compose.yml + контейнер |
| Використання у роботі | Основний локальний режим | Production-варіант |

## 3.2 Розробка інформаційної бази

Інформаційна база реалізована у модулі server/db.py. Модуль відповідає за створення каталогу data, ініціалізацію таблиць, відкриття підключення, запис

вимірювань і отримання історії для dashboard. Така інкапсуляція дозволяє приховати деталі SQL-запитів від інших частин програми.

Основна таблиця measurements зберігає часові мітки та числові параметри мікроклімату. Для запитів за періодом використовується поле timestamp. У разі розширення системи можна додати індекси для device_id і timestamp, що дозволить швидко будувати графіки за конкретним пристроєм або приміщенням.

Події системи доцільно зберігати окремо від вимірювань. Це дозволяє не змішувати телеметричні дані з технічними повідомленнями, такими як помилки валідації, підключення сервера, втрата пакета або формування команди fan_on. Такий поділ підвищує якість аналізу і спрощує налагодження.

## 3.3 Вибір інструментарію для створення прикладного програмного забезпечення

Основною мовою реалізації обрано Python. Вона має розвинену екосистему для роботи з даними, веб-інтерфейсами, мережевими протоколами й тестуванням. Для бакалаврської роботи Python також зручний тим, що дозволяє швидко реалізувати окремі модулі та продемонструвати їхню взаємодію.

Для створення dashboard використано Streamlit. Він дає можливість реалізувати метрики, графіки, таблиці та кнопки експорту без створення окремого клієнтського застосунку. Для побудови графіків застосовано Plotly, оскільки він підтримує інтерактивне масштабування та зручне відображення часових рядів.

Для обміну повідомленнями у локальному режимі реалізовано мінімальний MQTT-подібний publish/subscribe-канал у файлі iot/mqtt_minimal.py. Це рішення прийнято для зменшення залежностей і можливості запуску без встановлення Mosquitto. Водночас структура повідомлень і топіків зберігає логіку реального MQTT-підходу, тому надалі локальний брокер можна замінити на Mosquitto.

Таблиця 3.2

Обраний програмний стек

| Компонент | Технологія | Обґрунтування |
|---|---|---|

| | | |
|---|---|---|
| Мова програмування | Python | Швидка розробка, підтримка роботи з даними, тестування |
| Інтерфейс | Streamlit | Просте створення веб-dashboard без окремого frontend |
| Графіки | Plotly | Інтерактивна візуалізація часових рядів |
| Локальна БД | SQLite | Мінімальні залежності, файлова база для демонстрації |
| Production-БД | InfluxDB | Оптимізація для часових рядів |
| Протокол | MQTT | Легка модель publish/subscribe для IoT |
| Тестування | unittest | Вбудований засіб Python для модульних перевірок |

## 3.4 Реалізація MQTT-брокера та обміну повідомленнями

Модель publish/subscribe є центральним каналом обміну між джерелом телеметрії та серверною частиною. У локальному режимі використано файл iot/mqtt_minimal.py, який реалізує внутрішній MQTT-подібний брокер для передавання повідомлень між симулятором, серверним модулем і контуром керування. Цього достатньо для демонстрації ключової архітектурної ідеї - відокремлення джерела даних від отримувача.

Дані передаються у топіку microclimate/device001/telemetry, а команди керування - у топіку microclimate/device001/control. Формат повідомлення телеметрії є JSON-об’єктом. Така структура є зрозумілою, розширюваною і може бути використана як симулятором, так і реальним ESP32. Наприклад, додавання нового поля pm25 не потребуватиме зміни принципу взаємодії.

Серверний модуль підписується на телеметричний топік, а симулятор підписується на топік команд. У результаті формується двосторонній канал: пристрій передає вимірювання, а сервер повертає команду керування. Це відповідає концепції інформаційно-управляючої системи.

Таблиця 3.3

Формат MQTT-повідомлення телеметрії

| Поле | Тип | Призначення |
|---|---|---|
| device_id | string | Ідентифікатор джерела даних |
| location | string | Назва приміщення або місця розташування вузла |

| | | |
|---|---|---|
| timestamp | string | Час формування пакета у форматі ISO |
| temperature | float | Температура повітря, °C |
| humidity | float | Відносна вологість, % |
| pressure | float | Атмосферний тиск, мм рт. ст. |
| co2 | integer | Концентрація CO2, ppm |

## 3.5 Реалізація симулятора IoT-вузла

Симулятор IoT-вузла реалізовано у файлі simulator/esp32_simulator.py. Він виконує роль програмної моделі ESP32 з умовними датчиками BME280 і MH-Z19B. Симулятор формує пакет телеметрії кожні 5 секунд. До складу пакета входять поля device_id, location, timestamp, temperature, humidity, pressure, co2. Передавання виконується через локальний MQTT-подібний канал, що дозволяє перевірити архітектуру publish/subscribe без встановлення зовнішнього брокера.

Значення температури, вологості, тиску та CO2 генеруються як плавні коливання з невеликою випадковою складовою. Рівень CO2 залежить від умовного режиму зайнятості приміщення та стану вентиляції: при відсутності вентиляції концентрація поступово зростає, а після формування команди керування знижується. Температура змінюється з урахуванням керуючої дії: при рекомендації кондиціонування або зменшення обігріву цільова температура знижується.

Для перевірки стійкості системи симулятор підтримує кілька режимів: нормальний режим, стрибок CO2, шумний канал, періодичні відмови, значення поза діапазоном і застарілу часову мітку. Такі режими дають змогу перевірити не тільки стандартне отримання телеметрії, а й реакцію серверної частини на пропущені поля, некоректні типи даних, фізично неможливі значення та неактуальні пакети.

Окремо у каталозі firmware розміщено файл esp32_microclimate.ino. Він не потрібен для локального запуску, але демонструє, як програмний протокол може

бути перенесений на реальне обладнання. У прошивці передбачено підключення до Wi-Fi, роботу з MQTT, читання BME280/MH-Z19B і керування GPIO для реле.

Симулятор не є метрологічно точною моделлю фізичних сенсорів. Він не моделює калібрування, старіння датчиків, прогрів сенсора, електричні шуми живлення або реальні характеристики конкретних BME280 і MH-Z19B. Його призначення полягає у перевірці програмної частини системи: приймання даних, валідації, журналювання, прогнозного керування та візуалізації.

## 3.6 Реалізація серверного модуля обробки даних

Серверний модуль server/mqtt_subscriber.py є центральною частиною програмної системи. Його основна функція - отримати повідомлення телеметрії, перевірити його структуру, записати валідні дані в інформаційну базу та викликати модуль керування. Саме цей модуль забезпечує перехід від сирої телеметрії до керуючих рішень.

Валідація даних включає перевірку наявності обов’язкових полів, відповідності device_id очікуваному пристрою, коректності часової мітки, числового типу показників і допустимих діапазонів температури, вологості, тиску й CO2. Наприклад, відхиляються пакети без поля co2, з humidity = None, з temperature = "sensor_error", з CO2 поза очікуваним діапазоном або із застарілою часовою міткою.

Якщо пакет не проходить перевірку, він не впливає на стан системи і не записується до таблиці measurements. Замість цього у таблиці events фіксується причина відхилення. Після успішного запису вимірювання сервер передає значення до control_logic.py, який формує реактивну або прогнозну керуючу дію та за потреби публікує команду назад у канал керування.

## 3.7 Реалізація модуля автоматичного керування

Модуль автоматичного керування server/control_logic.py реалізує правила оцінки стану мікроклімату. Він складається з двох частин: реактивної та прогнозної. Реактивна частина використовує гістерезис для керування

вентиляцією: при досягненні CO2 рівня 1000 ppm формується команда fan_on, а після зниження до 800 ppm - fan_off. Такий підхід запобігає частому перемиканню виконавчого пристрою.

Прогнозна частина оцінює інерційність зміни параметрів. За останніми валідними вимірюваннями обчислюється темп зміни CO2 у ppm/хв та темп зміни температури у °C/хв. Далі система прогнозує значення на наступні 5 хвилин. Якщо прогноз показує наближення CO2 до порога 1000 ppm, система формує попереджувальну команду fan_pre_on, тобто вентиляція вмикається до фактичного виходу показника у червону зону.

Для температури система формує не дискретний сигнал реле, а керуючу рекомендацію. Якщо температура прогнозовано наближається до верхньої межі комфорту, формується дія “зменшити інтенсивність обігріву / підготувати кондиціонування”. Якщо верхню межу вже перевищено, дія змінюється на “увімкнути кондиціонування”.

Таблиця 3.4

Логіка прийняття рішень модулем керування

| Умова | Попередній стан | Команда | Пояснення |
|---|---|---|---|
| CO2 >= 1000 ppm | fan_state = false | fan_on | Потрібне примусове провітрювання або вентиляція |
| CO2 <= 800 ppm | fan_state = true | fan_off | Повітря повернулося до прийнятного рівня |
| 800 < CO2 < 1000 ppm | будь-який | немає | Зона гістерезису, стан вентиляції не змінюється |
| Прогноз CO2 наближається до 1000 ppm | fan_state = false | fan_pre_on | Попереджувальне ввімкнення вентиляції з урахуванням інерційності |
| Температура >= 25 °C | будь-який | Увімкнути кондиціонування | Температура перевищила верхню межу комфорту |

| | | | |
|---|---|---|---|
| Температура прогнозовано наближається до 25 °C | будь-який | Зменшити обігрів / підготувати кондиціонування | Керуюча дія формується до фактичного перевищення |
| Температура <= 21 °C | будь-який | Підвищити інтенсивність обігріву | Температура нижча за комфортний діапазон |

## 3.8 Реалізація Streamlit-панелі моніторингу

Streamlit-панель реалізована у файлі app/dashboard.py. Порівняно з початковим варіантом, де дані генерувалися безпосередньо в інтерфейсі, нова версія читає показники з бази даних. Це принципово важливо, оскільки dashboard тепер є лише рівнем представлення, а не джерелом телеметрії.

Інтерфейс поділено на кілька блоків: поточні показники, графіки, прогнозне керування, звіт, експорт даних, журнал останніх вимірювань і результати тестування. Така структура відповідає сценаріям користувача: спочатку швидкий контроль стану, потім аналіз динаміки, після цього формування звіту або перевірка деталей.

У панель додано блок прогнозного керування та інерційності. У ньому відображаються прогноз CO2 на 5 хвилин, прогноз температури на 5 хвилин, орієнтовний час до досягнення порогів, рівень ризику та поточна керуюча дія. Окремо побудовано графіки фактичних і прогнозованих значень CO2 та температури.

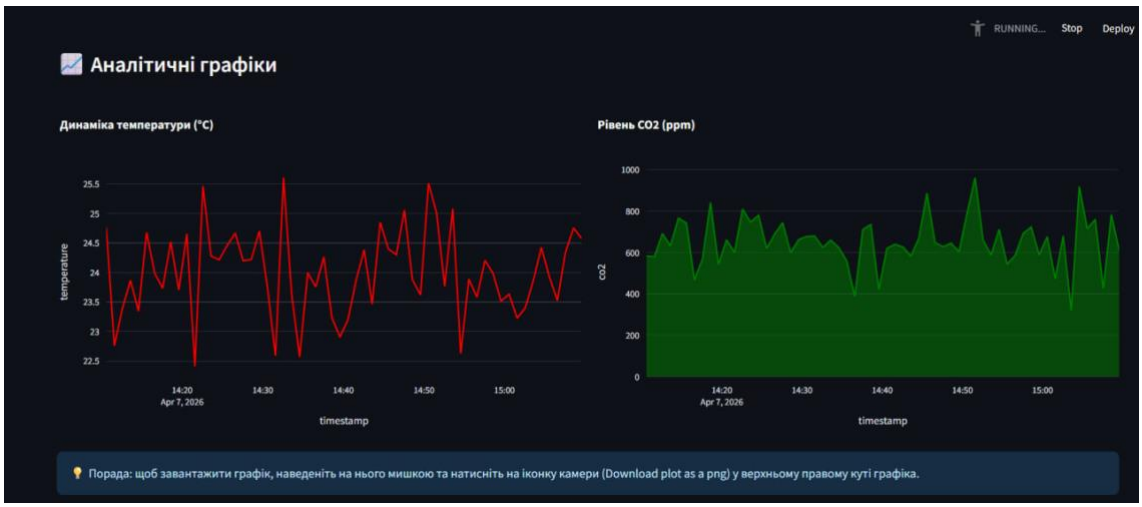



Рис. 3.1 Аналітичні графіки у Streamlit-панелі

На рис. 3.1 показано приклад графіків температури та CO2. Вони дозволяють не тільки побачити поточне значення, а й визначити тренд. Наприклад, поступове зростання CO2 може вказувати на недостатню вентиляцію навіть до досягнення критичного порогу.

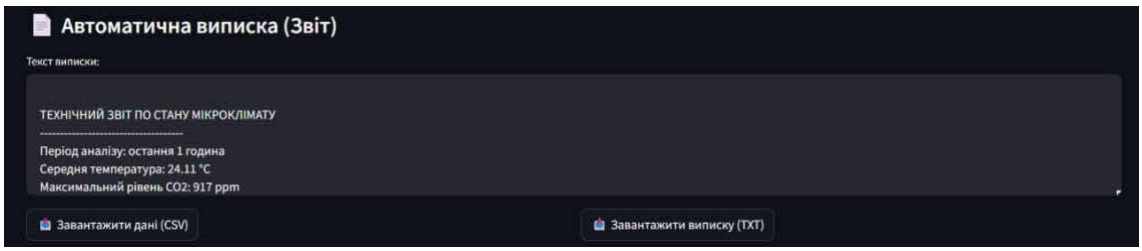
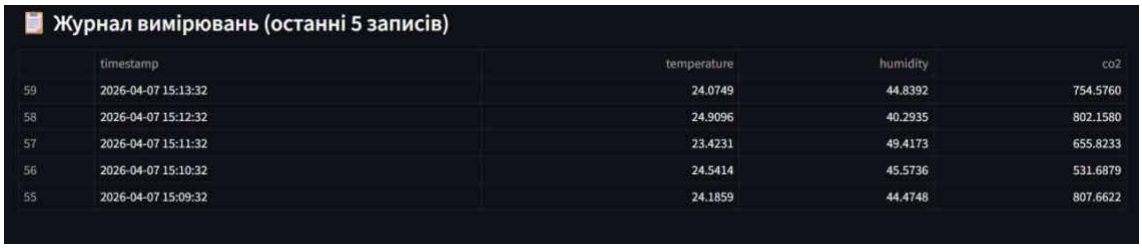



Рис. 3.2 Блок автоматичного формування виписки

Блок виписки на рис. 3.2 автоматично формує короткий текстовий висновок за період. Це корисно для звітності, адже користувач отримує не лише числові дані, а й готове пояснення стану приміщення.




| | timestamp | temperature | humidity | co2 |
|---|---|---|---|---|
| 59 | 2026-04-07 15:13:32 | 24.0749 | 44.8392 | 754.5760 |
| 58 | 2026-04-07 15:12:32 | 24.9096 | 40.2935 | 802.1580 |
| 57 | 2026-04-07 15:11:32 | 23.4231 | 49.4173 | 655.8233 |
| 56 | 2026-04-07 15:10:32 | 24.5414 | 45.5736 | 531.6879 |
| 55 | 2026-04-07 15:09:32 | 24.1859 | 44.4748 | 807.6622 |

Рис. 3.3 Журнал останніх вимірювань

## 3.9 Журналювання, обробка помилок і безпека

Журналювання реалізовано для фіксації ключових подій роботи системи. До журналу можуть потрапляти повідомлення про запуск фонового контуру,

приймання телеметрії, помилки валідації, відхилені пакети, прогнозовані ризики та формування керуючих дій fan_on, fan_off або fan_pre_on. Наявність журналу є важливою умовою супроводження програмного продукту.

Обробка помилок побудована так, щоб некоректне повідомлення не призводило до аварійного завершення сервера. Це особливо важливо для IoT-систем, де джерела даних можуть періодично втрачати зв’язок або надсилати неповні пакети. Сервер повинен ігнорувати помилкове повідомлення, зафіксувати проблему і продовжити роботу.

У локальному демонстраційному режимі питання автентифікації не є критичним, оскільки всі компоненти працюють на 127.0.0.1. Для виробничого режиму необхідно додати пароль до MQTT-брокера, обмежити мережевий доступ до бази даних, використовувати змінні середовища для конфігурації та забезпечити резервне копіювання історії вимірювань.

# 4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

## 4.1 Тестування системи

Тестування програмної системи виконувалося на рівні модулів та інтеграції. Модульне тестування перевіряє окремі функції: валідацію JSON-пакетів, оцінку комфорту, спрацювання гістерезису, прогнозне керування, формування команд fan_on/fan_off/fan_pre_on та реакцію на некоректні пакети. Інтеграційне тестування перевіряє послідовність роботи всього стенда: симулятор - MQTT-подібний канал - сервер - база - dashboard.

Для запуску тестів використовується файл run_tests.bat або команда python -m unittest discover -s tests -v. У поточній версії перевіряється 13 модульних тестів. Наявність тестів є важливою вимогою для попереднього захисту, оскільки дозволяє підтвердити не лише візуальну роботу інтерфейсу, а й коректність внутрішньої логіки програми.

Критерієм успішного тестування є не тільки відсутність помилок у консолі, а й відповідність програмного продукту постановці завдання. Тому перевіряються ключові сценарії: отримання телеметрії, запис до бази, відображення у браузері, експорт даних, реакція на перевищення CO2, попереджувальне керування за прогнозом, відхилення некоректних пакетів і коректність гістерезису.

Таблиця 4.1

Протокол функціонального тестування

| № | Сценарій | Очікуваний результат | Статус |
|---|---|---|---|
| 1 | Запуск start_dashboard.bat | Відкривається Streamlit-панель і запускається фоновий контур системи | Пройдено |
| 2 | Запуск симулятора IoT-вузла | Симулятор формує JSON-повідомлення кожні 5 секунд | Пройдено |

| | | | |
|---|---|---|---|
| 3 | Приймання телеметрії сервером | Сервер отримує пакет і виконує валідацію | Пройдено |
| 4 | Запис валідного вимірювання у базу | У таблиці measurements з’являється новий запис | Пройдено |
| 5 | Некоректний пакет телеметрії | Пакет відхиляється, причина записується у events | Пройдено |
| 6 | Перевищення CO2 | Формується команда fan_on | Пройдено |
| 7 | Зниження CO2 нижче порогу | Формується команда fan_off | Пройдено |
| 8 | Прогнозоване наближення CO2 до порога | Формується попереджувальна команда fan_pre_on | Пройдено |
| 9 | Зростання температури | Формується рекомендація зменшити обігрів або підготувати кондиціонування | Пройдено |
| 10 | Експорт даних | Користувач отримує CSV-файл або текстову виписку | Пройдено |

Таблиця 4.2

Модульні тести програмної логіки

| Файл тесту | Що перевіряється | Очікуваний результат |
|---|---|---|
| tests/test_logic.py | Гістерезис вентиляції fan_on/fan_off і збереження стану у зоні гістерезису | Рішення відповідають порогам 1000/800 ppm |
| tests/test_logic.py | Прогнозне керування CO2 і температурою | Формується fan_pre_on або температурна |

| | | |
|---|---|---|
| | | керуюча дія при прогнозованому ризику |
| tests/test_logic.py | Валідація payload: обов’язкові поля, типи, діапазони, timestamp | Некоректні пакети відхиляються, валідний пакет приймається |

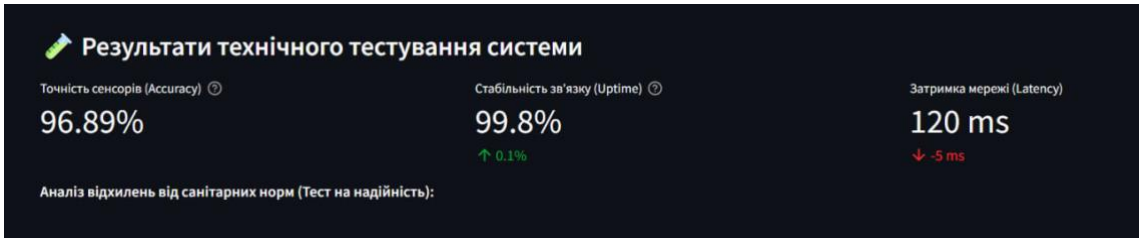



Рис. 4.1 Приклад відображення результатів технічного тестування

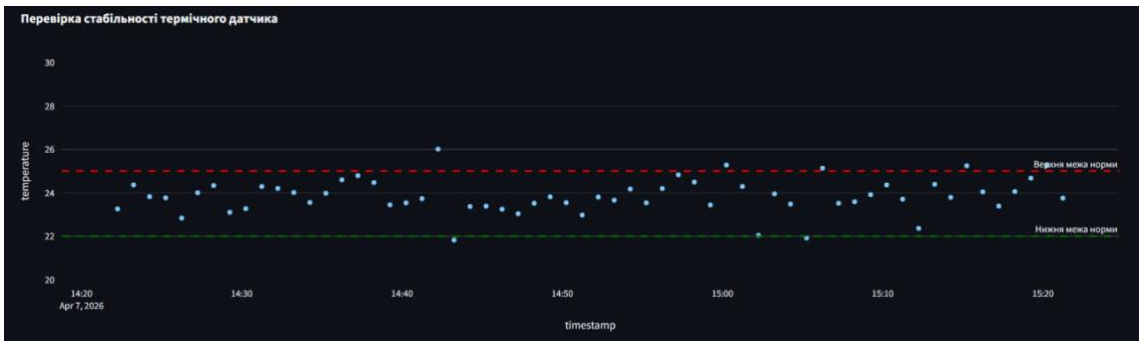



Рис. 4.2 Графік перевірки стабільності показників

Рис. 4.1 та рис. 4.2 використовуються як приклад представлення результатів тестування в інтерфейсі. У фінальній версії бажано формувати такі значення на основі тестових запусків і журналів, а не задавати їх вручну.

## 4.2 Вимоги до апаратного та програмного забезпечення

Для локального запуску програмного стенда достатньо персонального комп’ютера або ноутбука з операційною системою Windows 10/11. Оскільки фізичне обладнання відсутнє, вимоги до ESP32 і сенсорів є опціональними. Усі основні програмні сценарії можуть бути продемонстровані за допомогою симулятора.

Програмні вимоги включають Python, бібліотеки Streamlit, pandas і Plotly. Для portable-версії Python може знаходитися безпосередньо в каталозі проєкту.

У цьому випадку користувачеві не потрібно окремо встановлювати Python у систему. Для розширеного режиму з InfluxDB потрібен Docker Desktop.

У разі використання реального обладнання додатково потрібні ESP32, датчики BME280 і MH-Z19B, джерело живлення та мережа Wi-Fi. Проте ці компоненти не є обов'язковими для демонстрації програмної частини, оскільки симулятор відтворює формат даних і MQTT-взаємодію.

Таблиця 4.3

Мінімальні вимоги до запуску локального стенда

| **Категорія** | **Вимога** |
|---|---|
| Операційна система | Windows 10/11 |
| Процесор | 2 ядра або більше |
| Оперативна пам'ять | 4 ГБ або більше |
| Диск | не менше 500 МБ вільного місця для проєкту та бази |
| Python | 3.10 або новіший / portable Python |
| Браузер | Google Chrome, Microsoft Edge або інший сучасний браузер |
| Мережа | локальний доступ до 127.0.0.1; інтернет не обов'язковий для демо |

Таблиця 4.4

Опціональні вимоги для виробничого режиму

| **Компонент** | **Вимога** |
|---|---|
| Мікроконтролер | ESP32 DevKit або сумісна плата |
| Датчики | BME280, MH-Z19B |
| Виконавчий пристрій | Релейний модуль або транзисторний ключ для вентилятора |
| MQTT-брокер | Mosquitto |
| База часових рядів | InfluxDB |
| Контейнеризація | Docker Desktop |

## 4.3 Склад інсталяційного пакету

Інсталяційний пакет має бути зрозумілим для користувача та містити всі файли, необхідні для запуску. У цьому проєкті основним файлом запуску є start_dashboard.bat, який запускає Streamlit-панель і фоновий контур роботи стенда. Це спрощує демонстрацію на захисті, оскільки не потрібно окремо запускати брокер, сервер і симулятор у різних вікнах.

Склад пакету також включає каталог firmware. Він не використовується у локальному режимі, але підтверджує готовність архітектури до роботи з реальним ESP32. Каталог docs містить опис відповідності програмного проєкту вимогам дипломної роботи, а tests - модульні тести.

Таблиця 4.5

Склад інсталяційного пакету програмної системи

| Елемент | Призначення |
|---|---|
| app/dashboard.py | Streamlit-панель моніторингу |
| server/runtime.py | Фоновий контур роботи стенда |
| server/mqtt_subscriber.py | Сервер приймання, валідації телеметрії та формування команд |
| server/control_logic.py | Валідація пакетів, гістерезисне та прогнозне керування |
| server/db.py | Робота з SQLite, таблицями measurements, events, devices і control_settings |
| server/config.py | Константи, пороги, топіки та шляхи до файлів |
| simulator/esp32_simulator.py | Симулятор ESP32/BME280/MH-Z19B із режимами відмов |
| iot/mqtt_minimal.py | Локальний MQTT-подібний publish/subscribe-брокер |
| tests/test_logic.py | Модульні тести логіки керування, прогнозування та валідації |
| firmware/esp32_microclimate.ino | Ескіз прошивки для майбутнього фізичного ESP32 |
| untitled4.py | Сумісний launcher для Streamlit |
| start_dashboard.bat | Запуск локальної демонстраційної системи |
| run_tests.bat | Запуск модульних тестів |

## 4.4 Інструкція запуску програмного продукту

Для запуску локальної системи користувач розпаковує архів проєкту в окремий каталог. Якщо у проєкті є portable Python, додаткове встановлення Python не потрібне. Якщо portable Python відсутній, необхідно встановити Python 3.10 або новіший і виконати встановлення залежностей з requirements.txt.

Найпростіший спосіб запуску - виконати start_dashboard.bat. Цей файл відкриває адресу http://127.0.0.1:8501 і запускає Streamlit dashboard. Фоновий контур MicroclimateRuntime стартує разом із панеллю: симулятор формує телеметрію, локальний брокер передає повідомлення, серверний модуль виконує валідацію, а база даних зберігає історію вимірювань.

Для перевірки програмної логіки окремо від інтерфейсу використовується run_tests.bat. Він запускає модульні тести з каталогу tests і дозволяє переконатися, що логіка гістерезису, прогнозного керування та валідації працює коректно.

Таблиця 4.6

Послідовність запуску системи

| Крок | Дія | Очікуваний результат |
|---|---|---|
| 1 | Розпакувати архів проєкту або застосувати патч до portable-версії | Структура каталогів доступна на диску |
| 2 | Запустити start_dashboard.bat | Відкривається браузер і запускається Streamlit-панель |
| 3 | Дочекатися старту фонового контуру | У базі з’являються стартові та нові вимірювання |
| 4 | Переглянути поточні показники, графіки та журнал | Інтерфейс відображає телеметрію, рівень ризику і керуючу дію |
| 5 | За потреби змінити режим симуляції | Система перевіряє нормальні або аварійні сценарії |
| 6 | Запустити run_tests.bat | Успішно проходять модульні тести |

## 4.5 Рекомендації щодо впровадження

Для впровадження системи у реальному приміщенні потрібно замінити симулятор на ESP32 з датчиками. При цьому формат JSON, топіки обміну повідомленнями та серверна логіка залишаються незмінними. Такий підхід мінімізує обсяг доробок і підтверджує правильність модульної архітектури.

У виробничому режимі бажано використовувати Mosquitto як MQTT-брокер, InfluxDB як базу часових рядів і окремий сервер або міні-комп’ютер для постійної роботи. Також необхідно додати механізми автентифікації, резервного копіювання, контроль доступу до dashboard і моніторинг стану процесів.

Подальший розвиток може включати підтримку кількох приміщень, додавання датчика пилу PM2.5, push-сповіщення на телефон, інтеграцію з Home Assistant або Grafana, а також адаптивне керування вентиляцією залежно від часу доби та кількості людей у приміщенні.

## 4.6 Керівництво користувача програмної системи

Керівництво користувача призначене для оператора, який запускає програмний продукт, переглядає показники мікроклімату та формує звіт. Робота з системою починається з розпакування інсталяційного пакету у локальний каталог. Після цього користувач запускає файл start_dashboard.bat, який ініціює роботу всіх необхідних компонентів.

Головна сторінка dashboard містить блок поточних метрик. Користувач повинен звертати увагу насамперед на температуру, вологість, CO2, рівень ризику і поточну керуючу дію. Якщо інтерфейс показує попередження, користувач може переглянути графіки та визначити, чи є відхилення короткочасним або стабільним.

Графіки призначені для аналізу динаміки. Якщо CO2 поступово зростає, це може свідчити про недостатню вентиляцію. Якщо система прогнозує наближення до порога, вона може сформувати попереджувальну дію до фактичного виходу показника у червону зону.

Кнопка експорту CSV дозволяє зберегти дані у табличному форматі, а блок автоматичної виписки формує короткий текстовий звіт за вибраний період. У разі відсутності нових записів користувач повинен перевірити, чи запущений фоновий контур і чи не відхиляються пакети через помилки валідації.

## 4.7 Аналіз ризиків та обмежень розробленої системи

Основним обмеженням цієї роботи є відсутність фізичного обладнання під час розробки. Через це метрологічна точність реальних сенсорів не підтверджується експериментально. Натомість перевірено програмний цикл обробки даних, який не залежить від конкретного типу датчика.

Симулятор не відтворює всі фізичні особливості сенсорів: прогрів, калібрування, старіння, похибку вимірювання та випадкові перешкоди живлення. Результати симуляції не слід трактувати як реальні вимірювання конкретного приміщення; вони використовуються для перевірки програмної архітектури.

У локальному демонстраційному режимі система не має автентифікації, оскільки всі компоненти працюють на одному комп’ютері. Якщо систему розгортати у мережі, необхідно додати пароль для MQTT-брокера, обмеження доступу до dashboard, HTTPS або VPN, а також резервне копіювання бази даних.

Ще одним ризиком є залежність від локальної конфігурації Python. Для зменшення цього ризику передбачено portable Python, requirements.txt і запуск через .bat-файли. Перед захистом доцільно виконати пробний запуск на тому самому комп’ютері, де відбуватиметься демонстрація.

### 4.8 Перевірка відповідності програмного продукту постановці завдання

Оцінка відповідності показує, що програмний продукт реалізує основний цикл, заявлений у постановці завдання: формування телеметрії, передавання повідомлень, валідацію, збереження у SQLite, візуалізацію, експорт, журналювання та формування керуючих дій.

Вимога приймання телеметрії реалізована через локальний MQTT-подібний обмін між симулятором і сервером. Вимога збереження даних реалізована через SQLite. Вимога візуалізації реалізована через Streamlit dashboard. Вимога автоматичного керування реалізована через control_logic.py і команди fan_on, fan_off та fan_pre_on.

Додатково реалізовано контроль якості вхідних даних і прогнозне керування. Система відхиляє некоректні пакети, фіксує причину у журналі, розраховує темп зміни CO2 і температури, прогнозує значення на 5 хвилин і формує попереджувальні дії при наближенні до граничних зон.

### 4.9 План міграції на реальне обладнання

Міграція на реальне обладнання може виконуватися поетапно. На першому етапі необхідно зібрати апаратний вузол на базі ESP32, BME280 і MH-Z19B. На другому етапі прошивка повинна підключитися до тієї самої Wi-Fi мережі, що й комп’ютер із MQTT-брокером.

На третьому етапі ESP32 починає публікувати телеметрію у тому самому форматі JSON, який використовує симулятор. Якщо реальний ESP32 надсилає ті самі поля, серверний модуль і dashboard залишаються незмінними, що підтверджує коректне відокремлення апаратного і програмного рівнів.

Після підключення реальних датчиків потрібно провести калібрування та порівняння показників з еталонними приладами. Для температури та вологості можна використати побутовий термогігрометр, для CO2 - калібрований NDIR-датчик або лабораторний вимірювач.

Якщо система керує вентилятором або реле, необхідно забезпечити електробезпеку. У студентській демонстрації можна обмежитися програмним станом fan_on/fan_off без підключення до мережі 220 В. Для реального монтажу потрібна гальванічна розв’язка, відповідний корпус і дотримання правил електробезпеки.

# ВИСНОВКИ

У бакалаврській кваліфікаційній роботі виконано проєктування та реалізацію програмного забезпечення IoT-системи контролю мікроклімату у приміщеннях з візуалізацією даних у реальному часі. Роботу побудовано відповідно до вимог інженерії програмного забезпечення: проведено аналіз предметної області, сформовано вимоги, побудовано моделі, розроблено програмні модулі, виконано тестування та визначено склад інсталяційного пакету.

У першому розділі проаналізовано предметну область систем моніторингу мікроклімату, визначено користувачів, вхідні та вихідні дані, функціональні й нефункціональні вимоги. Показано, що власна IoT-система на базі відкритих технологій є доцільною для навчального проєкту, оскільки забезпечує гнучкість, доступ до коду та можливість локальної демонстрації.

У другому розділі виконано проєктування інформаційного та програмного забезпечення. Розроблено ER-діаграму, діаграму класів/модулів, пакетну структуру, діаграму компонентів, діаграму розміщення та блок-схему алгоритму обробки телеметрії. Ці моделі забезпечують цілісне уявлення про архітектуру системи й дозволяють пов’язати текст пояснювальної записки з реальним програмним продуктом.

У третьому розділі описано реалізацію локального демонстраційного стенда. Створено симулятор IoT-вузла, MQTT-подібний локальний брокер, серверний модуль приймання й валідації телеметрії, базу даних SQLite, модуль автоматичного та прогнозного керування і Streamlit dashboard. Окремо підготовлено файл прошивки ESP32 для майбутнього переходу від симуляції до реального обладнання.

У четвертому розділі наведено тестування, вимоги до апаратного та програмного забезпечення, склад інсталяційного пакету й інструкцію запуску. Підтверджено, що система може бути продемонстрована без фізичних датчиків, але при цьому зберігає архітектурну відповідність реальній IoT-системі: телеметрія передається через локальний MQTT-подібний канал, сервер обробляє й перевіряє дані, база зберігає історію, а інтерфейс відображає фактичні та прогнозні результати.

Поставлену мету досягнуто. Розроблене програмне забезпечення є модульним, тестованим і придатним для подальшого розширення. Найближчими напрямами розвитку є підключення реального ESP32, перехід на InfluxDB, додавання автентифікації, підтримка кількох пристроїв і розширення аналітичних можливостей dashboard.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Барало О. В., Самойленко П. Г., Гранат С. Є. Автоматизація технологічних процесів і системи автоматичного керування : навч. посіб. Київ : Аграрна освіта, 2010. 557 с.

2. Говоров П. П., Перепечений В. І., Говорова К. В. Автоматизація керування режимами систем електропостачання та освітлення міст. Вісник Вінницького політехнічного інституту. 2021. № 5. С. 58-63.

3. Бондаренко І. М., Бородин О. В., Карнаушенко В. П. Мікропроцесорні системи контролю та керування : навч. посіб. Харків : ХНУРЕ, 2020.

4. Заячковський А. В. та ін. Моніторинг та використання IoT для автоматизації промислових процесів. Зв'язок. 2025. № 3. С. 3-7.

5. Безкоровайний М. Р. Мікропроцесорна система управління житловим комплексом. Апаратна частина : кваліфікаційна робота. Запоріжжя, 2022.

6. Смірнов В. В., Смірнова Н. В., Пархоменко Ю. М. Програмування мікроконтролерних систем : навч. посіб. Кропивницький, 2021.

7. Лазарович І. М. Реалізація методів сигнальної рандомізації для завадостійкого передавання даних. Вісник Хмельницького національного університету. Технічні науки. 2014. № 6. С. 165-168.

8. Дячук О. Ю. та ін. Комплексний аналіз програмного забезпечення для моделювання та емуляції комп'ютерних мереж. Житомир, 2024.

9. Олійник Т. М. Методи і засоби проектування архітектури комп'ютерних систем з використанням IoT компонентів : магістерська робота. 2018.

10. Дараган В. В. Веб-інтерфейси для моніторингу та управління роботизованими системами в реальному часі. Харків, 2025.

11. Цирульник С. М., Лисенко Г. Л. Проектування мікропроцесорних систем : навч. посіб. Вінниця : ВНТУ, 2012.

12. Kopetz H. Real-Time Systems: Design Principles for Distributed Embedded Applications. Boston : Springer, 1997.

13. Cisco Press. IoT Fundamentals: Networking Technologies, Protocols, and Use Cases for the Internet of Things. Indianapolis : Cisco Press, 2017.

14. Buyya R., Dastjerdi A. V. Internet of Things: Principles and Paradigms. Cambridge : Elsevier, 2016.

15. Ray P. P. A survey on Internet of Things architectures. Journal of King Saud University - Computer and Information Sciences. 2018. Vol. 30, No. 3. P. 291-319.

16. Al-Fuqaha A., Guizani M., Mohammadi M., Aledhari M., Ayyash M. Internet of Things: A survey on enabling technologies, protocols, and applications. IEEE Communications Surveys & Tutorials. 2015. Vol. 17, No. 4. P. 2347-2376.

17. Nauman A. et al. Multimedia Internet of Things: A comprehensive survey. IEEE Access. 2020. Vol. 8. P. 8202-8250.

18. Monk S. Raspberry Pi Cookbook. 4th ed. Sebastopol : O'Reilly Media, 2022.

19. Scherz P., Monk S. Practical Electronics for Inventors. 4th ed. New York : McGraw-Hill Education, 2016.

20. Документація Streamlit [Електронний ресурс]. URL: https://docs.streamlit.io/ (дата звернення: 20.04.2026).

21. Документація SQLite [Електронний ресурс]. URL: https://www.sqlite.org/docs.html (дата звернення: 20.04.2026).

22. Документація Eclipse Mosquitto [Електронний ресурс]. URL: https://mosquitto.org/documentation/ (дата звернення: 20.04.2026).

**ДОДАТОК А**

## Фрагменти лістингу програмних модулів

У додатку наведено скорочені фрагменти основних програмних модулів. Повні файли містяться в інсталяційному пакеті проєкту.

# app/dashboard.py

```
from __future__ import annotations

import sys
import time
from datetime import datetime
from pathlib import Path

import pandas as pd
import plotly.express as px
import streamlit as st

ROOT = Path(__file__).resolve().parents[1]
if str(ROOT) not in sys.path:
    sys.path.insert(0, str(ROOT))

from server import config
from server.control_logic import comfort_status
from server.db import fetch_events, fetch_latest, fetch_measurements, init_db

st.set_page_config(page_title="IoT Microclimate Dashboard", layout="wide")
init_db()

st.title("📊 IoT-система моніторингу мікроклімату")
st.caption("Демо-режим без фізичного ESP32: симулятор датчиків → MQTT → сервер → SQLite → Streamlit dashboard")

with st.sidebar:
    st.header("⚙️ Параметри")
    minutes = st.slider("Період аналізу, хв", min_value=10, max_value=360, value=config.DASHBOARD_DEFAULT_MINUTES, step=10)
    auto_refresh = st.checkbox("Автооновлення кожні 5 секунд", value=True)
    st.markdown("---")
    st.write("**MQTT telemetry:**")
    st.code(config.TELEMETRY_TOPIC, language="text")
    st.write("**MQTT control:**")
    st.code(config.CONTROL_TOPIC, language="text")
    st.write("**База даних:**")
    st.code(str(config.DB_PATH), language="text")

rows = fetch_measurements(minutes=minutes)
latest = fetch_latest()
events = fetch_events(limit=30)

if not rows or latest is None:
    st.warning("Даних ще немає. Запусти `start_all.bat` або окремо: broker → server → simulator → dashboard.")
    st.info("Після запуску симулятора перші записи з'являться приблизно через 5 секунд.")
    st.stop()

df = pd.DataFrame(rows)
df["timestamp"] = pd.to_datetime(df["timestamp"], errors="coerce")
df["received_at"] = pd.to_datetime(df["received_at"], errors="coerce")
df["fan_state_label"] = df["fan_state"].map({0: "Вимкнено", 1: "Увімкнено"})

current_temp = float(latest["temperature"])
current_hum = float(latest["humidity"])
current_co2 = int(latest["co2"])
current_pressure = latest.get("pressure")
current_fan = bool(latest.get("fan_state"))
current_latency = latest.get("latency_ms")
comfort, comfort_reason = comfort_status(current_temp, current_hum, current_co2)

st.markdown(f"**Останнє оновлення dashboard:** {datetime.now().strftime('%d.%m.%Y %H:%M:%S')}")

st.subheader("🌡️ Поточні показники")
col1, col2, col3, col4, col5, col6 = st.columns(6)
col1.metric("Температура", f"{current_temp:.1f} °C")
```

```
col2.metric("Вологість", f"{current_hum:.1f} %")
col3.metric("CO₂", f"{current_co2} ppm", "Вентиляція" if

current_fan else "Моніторинг")
col4.metric("Тиск", "—" if current_pressure is None else f"{float(current_pressure):.1f} мм рт. ст.")
col5.metric("Вентиляція", "Увімкнено" if current_fan else "Вимкнено")
col6.metric("Latency", "—" if current_latency is None else f"{float(current_latency):.0f} ms")

if comfort.startswith("Комфортно"):
    st.success(f"{comfort}. {comfort_reason}")
else:
    st.warning(f"{comfort}. {comfort_reason}")

st.markdown("---")
st.subheader("📈 Графіки часових рядів")
left, right = st.columns(2)

fig_temp = px.line(df, x="timestamp", y="temperature", title="Температура, °C")
fig_temp.add_hrect(y0=config.COMFORT_TEMP_MIN, y1=config.COMFORT_TEMP_MAX, opacity=0.12, line_width=0)
left.plotly_chart(fig_temp, use_container_width=True)

fig_hum = px.line(df, x="timestamp", y="humidity", title="Вологість, %")
fig_hum.add_hrect(y0=config.COMFORT_HUM_MIN, y1=config.COMFORT_HUM_MAX, opacity=0.12, line_width=0)
right.plotly_chart(fig_hum, use_container_width=True)

left2, right2 = st.columns(2)
fig_co2 = px.area(df, x="timestamp", y="co2", title="Рівень CO₂, ppm")
fig_co2.add_hline(y=config.COMFORT_CO2_MAX, line_dash="dash", annotation_text="Комфортна межа 800 ppm")
```

# server/mqtt_subscriber.py

```
from __future__ import annotations

import json
import logging
import sys
from datetime import datetime, timezone
from pathlib import Path
from typing import Any

ROOT = Path(__file__).resolve().parents[1]
if str(ROOT) not in sys.path:
    sys.path.insert(0, str(ROOT))

from iot.mqtt_minimal import MqttClient
from server import config
from server.control_logic import evaluate_control
from server.db import add_event, init_db, save_measurement

fan_state = False
publisher: MqttClient | None = None


def setup_logging() -> None:
    config.LOG_DIR.mkdir(parents=True, exist_ok=True)
    logging.basicConfig(
        level=logging.INFO,
        format="%(asctime)s [%(levelname)s] %(message)s",
        handlers=[
            logging.FileHandler(config.LOG_FILE, encoding="utf-8"),
            logging.StreamHandler(sys.stdout),
        ],
    )


def parse_iso(value: str) -> datetime | None:
    try:
        if value.endswith("Z"):
            value = value[:-1] + "+00:00"
        dt = datetime.fromisoformat(value)
        if dt.tzinfo is None:
            dt = dt.replace(tzinfo=timezone.utc)
        return dt.astimezone(timezone.utc)
    except Exception:
        return None
```

```
def validate_payload(payload: dict[str, Any]) -> list[str]:
    errors: list[str] = []
    required = ["device_id", "timestamp", "temperature", "humidity", "co2"]
    for field in required:
        if field not in payload:
            errors.append(f"відсутнє поле {field}")

    try:
        temp = float(payload.get("temperature"))
        if not (-40 <= temp <= 80):
            errors.append("temperature поза фізично очікуваним діапазоном")
    except Exception:
        errors.append("temperature не є числом")

    try:
        hum = float(payload.get("humidity"))
        if not (0 <= hum <= 100):
            errors.append("humidity має бути 0..100%")
    except Exception:
        errors.append("humidity не є числом")

    try:
        co2 = int(payload.get("co2"))
        if not (0 <= co2 <= 10000):
            errors.append("co2 має бути 0..10000 ppm")
    except Exception:
        errors.append("co2 не є цілим числом")

    if parse_iso(str(payload.get("timestamp", ""))) is None:
        errors.append("timestamp має бути ISO-датою")
    return errors


def publish_control(command: str, reason: str, co2: int, device_id: str) -> None:
    global publisher
    if publisher is None:
        publisher = MqttClient("microclimate_server_control", config.MQTT_HOST, config.MQTT_PORT)


publisher.connect()

    message = {
        "device_id": device_id,
        "command": command,
        "reason": reason,
        "co2": co2,
```

# server/control_logic.py

```
from __future__ import annotations

from dataclasses import dataclass

try:
    from server import config
except ModuleNotFoundError:
    import config  # type: ignore


@dataclass(frozen=True)
class ControlDecision:
    fan_state: bool
    command: str | None
    level: str
    status: str
    reason: str


def comfort_status(temperature: float, humidity: float, co2: int) -> tuple[str, str]:
    problems: list[str] = []
    if not (config.COMFORT_TEMP_MIN <= temperature <= config.COMFORT_TEMP_MAX):
        problems.append("температура поза комфортним діапазоном")
    if not (config.COMFORT_HUM_MIN <= humidity <= config.COMFORT_HUM_MAX):
        problems.append("вологість поза комфортним діапазоном")
    if co2 > config.COMFORT_CO2_MAX:
```

```
        problems.append("підвищений CO2")

    if not problems:
        return "Комфортно ✅", "Усі ключові параметри в межах норми"
    return "Некомфортно ⚠️", "; ".join(problems)


def evaluate_control(temperature: float, humidity: float, co2: int, previous_fan_state: bool) -> ControlDecision:
    fan_state = previous_fan_state
    command: str | None = None
    level = "INFO"
    reason = "Показники в межах заданих порогів"

    # Гістерезис: вмикаємо при CO2 > 1000, вимикаємо лише після падіння нижче 800.
    if co2 >= config.CO2_FAN_ON_PPM and not previous_fan_state:
        fan_state = True
        command = "fan_on"
        level = "WARNING"
        reason = f"CO2={co2} ppm перевищує поріг {config.CO2_FAN_ON_PPM} ppm"
    elif co2 <= config.CO2_FAN_OFF_PPM and previous_fan_state:
        fan_state = False
        command = "fan_off"
        level = "INFO"
        reason = f"CO2={co2} ppm знизився нижче {config.CO2_FAN_OFF_PPM} ppm"

    if temperature >= config.TEMP_WARNING_HIGH:
        level = "WARNING"
        reason = f"Температура={temperature:.1f} °C перевищує {config.TEMP_WARNING_HIGH:.1f} °C"
    elif temperature <= config.TEMP_WARNING_LOW:
        level = "WARNING"
        reason = f"Температура={temperature:.1f} °C нижча за {config.TEMP_WARNING_LOW:.1f} °C"

    if humidity >= config.HUMIDITY_WARNING_HIGH:
        level = "WARNING"
        reason = f"Вологість={humidity:.1f}% перевищує {config.HUMIDITY_WARNING_HIGH:.1f}%"
    elif humidity <= config.HUMIDITY_WARNING_LOW:
        level = "WARNING"
        reason = f"Вологість={humidity:.1f}% нижча за {config.HUMIDITY_WARNING_LOW:.1f}%"

    status, comfort_reason = comfort_status(temperature, humidity, co2)
    if command is None and

status.startswith("Некомфортно"):
        reason = comfort_reason
        level = "WARNING"

    return ControlDecision(
        fan_state=fan_state,
        command=command,
        level=level,
        status=status,
        reason=reason,
    )
```

# simulator/esp32_simulator.py

```
from __future__ import annotations

import json
import math
import random
import sys
import threading
import time
from datetime import datetime, timezone
from pathlib import Path

ROOT = Path(__file__).resolve().parents[1]
if str(ROOT) not in sys.path:
    sys.path.insert(0, str(ROOT))

from iot.mqtt_minimal import MqttClient
```

```
from server import config

fan_state = False
co2_value = 650.0
start_time = time.time()
lock = threading.Lock()


def iso_now() -> str:
    return datetime.now(timezone.utc).isoformat(timespec="milliseconds")


def on_control(topic: str, payload: bytes) -> None:
    global fan_state
    try:
        message = json.loads(payload.decode("utf-8"))
    except json.JSONDecodeError:
        print(f"[SIM] Некоректна команда: {payload!r}")
        return

    command = message.get("command")
    with lock:
        if command == "fan_on":
            fan_state = True
        elif command == "fan_off":
            fan_state = False
    print(f"[SIM] Отримано команду {command}, fan_state={fan_state}. Причина: {message.get('reason', '-')}")


def run_control_listener() -> None:
    client = MqttClient("esp32_simulator_control", config.MQTT_HOST, config.MQTT_PORT)
    client.connect()
    client.subscribe(config.CONTROL_TOPIC)
    print(f"[SIM] Слухаю команди: {config.CONTROL_TOPIC}")
    client.loop_forever(on_control)


def generate_measurement() -> dict[str, object]:
    global co2_value
    elapsed = time.time() - start_time
    day_wave = math.sin(elapsed / 180.0)
    short_wave = math.sin(elapsed / 25.0)

    with lock:
        fan = fan_state

    # Імітація: без вентиляції CO2 поступово росте, з вентиляцією падає.
    if fan:
        co2_value -= random.uniform(15, 35)
    else:
        co2_value += random.uniform(-8, 20)
    co2_value = max(410, min(1500, co2_value))

    temperature = 23.2 + 1.1 * day_wave + 0.25 * short_wave + random.uniform(-0.18, 0.18)
    humidity = 46.0 - 4.0 * day_wave + random.uniform(-1.2, 1.2)
    pressure = 749.0 + 2.5 * math.sin(elapsed / 420.0) + random.uniform(-0.25, 0.25)

    return {
        "device_id": config.DEVICE_ID,
        "location": config.LOCATION,
        "source": "ESP32_SIMULATOR",
        "simulated": True,
        "timestamp": iso_now(),
        "temperature": round(temperature, 2),
        "humidity": round(humidity, 2),
        "pressure": round(pressure, 2),
        "co2": int(co2_value),
        "fan_state": fan,
        "sensor_status": "OK",
    }


def main() -> None:
    listener =

threading.Thread(target=run_control_listener, daemon=True)
```

# firmware/esp32_microclimate.ino

```
#include <WiFi.h>
#include <PubSubClient.h>
#include <ArduinoJson.h>
#include <Wire.h>
#include <Adafruit_BME280.h>
#include <MHZ19.h>

#define DEVICE_ID "esp32_001"
#define LOCATION "Навчальна аудиторія"

const char* WIFI_SSID = "YOUR_WIFI_NAME";
const char* WIFI_PASS = "YOUR_WIFI_PASSWORD";
const char* MQTT_HOST = "192.168.1.100";   // IP ПК або Raspberry Pi з брокером
const int MQTT_PORT = 1883;

const char* TELEMETRY_TOPIC = "microclimate/esp32_001/telemetry";
const char* CONTROL_TOPIC   = "microclimate/esp32_001/control";

const int RELAY_PIN = 25;
const int MHZ19_RX = 16; // ESP32 RX2 <- TX датчика
const int MHZ19_TX = 17; // ESP32 TX2 -> RX датчика

WiFiClient wifiClient;
PubSubClient mqtt(wifiClient);
Adafruit_BME280 bme;
HardwareSerial co2Serial(2);
MHZ19 mhz19;

bool fanState = false;
unsigned long lastSend = 0;
const unsigned long SEND_INTERVAL_MS = 5000;

void setFan(bool enabled) {
  fanState = enabled;
  digitalWrite(RELAY_PIN, enabled ? HIGH : LOW);
}

String isoTimeStub() {
  // Для реального часу бажано додати NTP.
  // Тут передається uptime, а сервер може проставляти received_at.
  return String(millis());
}

void onMqttMessage(char* topic, byte* payload, unsigned int length) {
  StaticJsonDocument<256> doc;
  DeserializationError error = deserializeJson(doc, payload, length);
  if (error) {
    return;
  }

  const char* command = doc["command"] | "";
  if (strcmp(command, "fan_on") == 0) {
    setFan(true);
  } else if (strcmp(command, "fan_off") == 0) {
    setFan(false);
  }
}

void connectWifi() {
  WiFi.mode(WIFI_STA);
  WiFi.begin(WIFI_SSID, WIFI_PASS);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
  }
}

void connectMqtt() {
  while (!mqtt.connected()) {
    if (mqtt.connect(DEVICE_ID)) {
      mqtt.subscribe(CONTROL_TOPIC);
    } else {
      delay(1000);
    }
  }
```

**ДОДАТОК Б**

# Протокол тестування програмного продукту

Таблиця Б.1

Перелік контрольних перевірок

| № | Перевірка | Очікуваний результат | Результат |
|---|---|---|---|
| 1 | Запуск start_all.bat | Усі компоненти стартують без критичних помилок | Позитивний |
| 2 | Отримання телеметрії | Сервер приймає дані з MQTT | Позитивний |
| 3 | Запис до бази | Файл microclimate.db містить нові записи | Позитивний |
| 4 | Відображення dashboard | Дані доступні у браузері | Позитивний |
| 5 | Команда fan_on | Команда формується при перевищенні CO2 | Позитивний |
| 6 | Команда fan_off | Команда формується після зниження CO2 | Позитивний |
| 7 | Запуск модульних тестів | Тести завершуються без помилок | Позитивний |

**ДОДАТОК В**

# Структура інсталяційного пакету

```
microclimate_iot/
├── app/
│   └── dashboard.py
├── server/
│   ├── config.py
│   ├── db.py
│   ├── control  logic.py
│   └── mqtt_subscriber.py
├── simulator/
│   └── esp32_simulator.py
├── iot/
│   └── mqtt_minimal.py
├── firmware/
│   └── esp32_microclimate.ino
├── tests/
│   ├── test_control_logic.py
│   └── test_payload_validation.py
├── start_dashboard.bat
├── run_tests.bat
└── README.md
```

**ДОДАТОК Г**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**
**БІОРЕСУРСІВ І**
**ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ**

**Факультет інформаційних технологій**

**Декларація про академічну доброчесність та використання ШІ**

Я, Майборода Валентин Андрійович, здобувач НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

1. Моя бакалаврська кваліфікаційна робота на тему

«IoT-система контролю мікроклімату у приміщеннях з візуалізацією даних у реальному часі» є результатом моєї особистої праці.

2. Усі запозичення з текстів інших авторів мають відповідні посилання згідно з чинними стандартами.
3. Щодо використання інструментів ШІ зазначаю, що (обрати необхідне):
    - [ ] інструменти генеративного ШІ не використовувалися.
    - [✓ ] інструмент ШІ ChatGPT був використаний для:
        - [ ] перекладу іншомовних джерел;
        - [✓] стилістичного редагування та перевірки граматики;
        - [✓] допомоги у написанні/відлагодженні програмного коду;
        - [ ] інше: _____.

Я розумію, що надання недостовірної інформації може призвести до скасування результатів захисту та відрахування з числа здобувачів НУБіП України.

«21» травня 2026 р. _____ **Валентин МАЙБОРОДА**

(підпис)