Форма № Н-9.02

Міністерство освіти і науки України

Відокремлений структурний підрозділ «Тернопільський фаховий коледж Тернопільського національного технічного університету імені Івана Пулюя»

(повне найменування вищого навчального закладу)

Відділення інформаційних технологій, менеджменту, туризму та підготовки іноземних громадян

(назва відділення)

Циклова комісія комп'ютерної інженерії

(повна назва циклової комісії)

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи

фахового молодшого бакалавра

(освітньо-професійного ступеня)

на тему: **Розробка серверу баз даних для збереження та синхронізації даних IoT пристроїв**

Виконав: студент IV курсу, групи КІ-412

Спеціальності 123 Комп'ютерна інженерія

(шифр і назва спеціальності)

_____ Станіслав МОСКАЛЬ

(ім'я та прізвище)

Керівник Наталя ДЗЮБАТА

(ім'я та прізвище)

Рецензент _____

(ім'я та прізвище)

Тернопіль – 2026

ВІДОКРЕМЛЕНИЙ СТРУКТУРНИЙ ПІДРОЗДІЛ
«ТЕРНОПІЛЬСЬКИЙ ФАХОВИЙ КОЛЕДЖ
ТЕРНОПІЛЬСЬКОГО НАЦІОНАЛЬНОГО ТЕХНІЧНОГО УНІВЕРСИТЕТУ
імені ІВАНА ПУЛЮЯ»

Відділення **інформаційних технологій, менеджменту, туризму та підготовки іноземних громадян**
Циклова комісія **комп'ютерної інженерії**
Освітньо-професійний ступінь **фаховий молодший бакалавр**
Освітньо-професійна програма: **Обслуговування програмних систем і комплексів**
Спеціальність: **123 Комп'ютерна інженерія**
Галузь знань: **12 Інформаційні технології**

ЗАТВЕРДЖУЮ
Голова циклової комісії
комп'ютерної інженерії
_____ Андрій ЮЗЬКІВ
"30" березня 2026 року

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

Москалю Станіславу Віталійовичу
(прізвище, ім'я, по батькові)

Тема кваліфікаційної роботи: **Розробка серверу баз даних для збереження та синхронізації даних IoT пристроїв**

керівник роботи Дзюбата Наталія Миколаївна
(прізвище, ім'я, по батькові)

затверджені наказом Відокремленого структурного підрозділу «Тернопільський фаховий коледж Тернопільського національного технічного університету імені Івана Пулюя» від 27.03.2026р № 4/9-167.

2. Строк подання студентом роботи: 15 червня 2026 року.

3. Вихідні дані до роботи: технічне завдання; структура бази даних теплиці; API сервісів (ESP32/Wokwi, PHP REST API, MySQL 8.0, phpMyAdmin, Docker/Vagrant); стандарти ЄСПД (ГОСТ 19.701, 19.201, 19.402); формат JSON, REST API.

4. Зміст розрахунково-пояснювальної записки: Загальний розділ. Розробка технічного та робочого проєкту. Спеціальний розділ. Економічна частина. Охорона праці, техніка безпеки та екологічні вимоги.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
   - Структурна схема;
   - ER-діаграма бази даних системи;
   - Блок схема алгоритму;
   - таблиця техніко-економічних показників.

6. Консультанти розділів роботи

Розділ	Ім'я, прізвище та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний розділ	Богдана МАРТИНЮК викладач		
Охорона праці та безпека життєдіяльності	Володимир ШТОКАЛО викладач		

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Отримання і аналіз технічного завдання	31.03	
2	Збір і узагальнення інформації	08.05	
3	Написання першого розділу	15.05	
4	Розробка технічного та робочого проекту	22.05	
5	Написання спеціального розділу	28.05	
6	Розрахунок економічної частини	1.06	
7	Написання розділу охорони праці	3.06	
8	Виконання графічної частини	8.06	
9	Оформлення проєкту	10.06	
10	Погодження нормоконтролю	11.06	
11	Попередній захист роботи	12.06	
12	Захист кваліфікаційної роботи		

7. Дата видачі завдання: 31 березня 2026 року

Студент _____ Станіслав МОСКАЛЬ
(підпис) (ім'я та прізвище)

Керівник роботи _____ Наталя ДЗЮБАТА
(підпис) (ім'я та прізвище)

АНОТАЦІЯ

Москаль С.В Розробка серверу баз даних для збереження та синхронізації даних IoT пристроїв: кваліфікаційна робота на здобуття освітньо-професійного ступеня фахового молодшого бакалавра, за спеціальністю 123 Комп'ютерна інженерія. Тернопіль: ВСП «ТФК ТНТУ», 2026. – ___ с.
Метою роботи є проєктування та реалізація реляційної бази даних і REST API серверу для системи моніторингу та дистанційного керування IoT-пристроями розумної теплиці. Обґрунтовано вибір технологій (MySQL 8.0, PHP 8.2, Docker, Vagrant), спроєктовано структуру бази даних із таблицями для зберігання показників датчиків, журналу подій, системи сповіщень, керування пристроями та автентифікації користувачів. Реалізовано механізми погодинної агрегації даних через Event Scheduler, автоматичного формування алертів через тригери MySQL, автентифікацію на основі API-ключів з розмежуванням прав доступу. Наведено інструкцію з розгортання системи засобами контейнеризації Docker та віртуалізації Vagrant, методику тестування, економічне обґрунтування та заходи з охорони праці.
Робота містить графічну частину на 4 аркушах А1 та пояснювальну записку обсягом 71 аркушів, що містить 5 таблиць і 20 рисунків.
Ключові слова: база даних, MySQL, REST API, PHP, IoT, розумна теплиця, Docker, Vagrant, автентифікація, API-ключ, тригер, агрегація даних, охорона праці.

ANNOTATION

Moskal. S.V. Development of a database server for storing and synchronising data from IoT devices Development of a database server for storing and synchronising data from IoT devices: qualification thesis for the degree of Professional Junior Bachelor, specialty 123 Computer Engineering. Ternopil: VSP "TFK TNTU", 2026. – ___ p.
The aim of the work is to design and implement a relational database and REST API server for a system of monitoring and remote control of smart greenhouse IoT devices. The selection of technologies (MySQL 8.0, PHP 8.2, Docker, Vagrant) is justified, the database structure is designed with tables for storing sensor readings, event logs, alert systems, device management, and user authentication. Mechanisms for hourly data aggregation via Event Scheduler, automatic alert generation via MySQL triggers, and API key-based authentication with role-based access control are implemented. Deployment instructions using Docker containerization and Vagrant virtualization, a testing methodology, economic justification, and occupational safety measures are provided.
The work includes a graphic section of 4 A1 sheets and an explanatory note of 71 pages containing 5 tables and 20 figures.
Keywords: database, MySQL, REST API, PHP, IoT, smart greenhouse, Docker, Vagrant, authentication, API key, trigger, data aggregation, occupational safety.

ЗМІСТ

ВСТУП

Стрімкий розвиток інформаційних технологій та Інтернету речей (IoT) докорінно змінює підходи до автоматизації у найрізноманітніших сферах людської діяльності. Одним із найперспективніших напрямків інтеграції цих технологій є сучасний агропромисловий комплекс та системи "Розумного дому" (Smart Home). Концепція автоматизованого вирощування рослин, відома як "Розумна теплиця", передбачає створення повністю автономного середовища, яке здатне без втручання людини збирати показники мікроклімату та приймати рішення щодо керування виконавчими механізмами: системами освітлення, поливу, обігріву та вентиляції.

Будь-яка комплексна система Інтернету речей складається з трьох ключових рівнів: апаратного забезпечення (мікроконтролерів та датчиків), серверної частини (баз даних та API) і клієнтського програмного забезпечення (мобільних або веб-додатків). Головною проблемою, яка виникає при проектуванні таких систем, є організація безперервного, надійного та безпечного обміну даними між фізичними пристроями та користувачем. Мікроконтролери генерують величезні масиви телеметричної інформації (з частотою відправки кожні декілька секунд), яку необхідно не лише прийняти, але й оптимізувати, зберегти та синхронізувати для миттєвого відображення на мобільному пристрої.

Дана кваліфікаційна робота є складовою частиною комплексного розроблення системи розумної теплиці, над яким працює група студентів. Комплексна система поділена на три логічні модулі: розробка апаратної системи автоматизованого керування мікрокліматом (апаратний рівень), розробка мобільного інтерфейсу для віддаленого керування (клієнтський рівень) та розробка серверної архітектури.

Метою даної кваліфікаційної роботи є розв’язання задачі збереження та синхронізації інформації шляхом розробки оптимізованої реляційної бази даних та програмного інтерфейсу (API) розумної теплиці.

Для досягнення поставленої мети у кваліфікаційній роботі необхідно вирішити наступні завдання: – провести аналіз предметної області та існуючих технологій побудови баз даних для систем Інтернету речей; – спроектувати логічну та фізичну структуру реляційної бази даних для збереження показників датчиків (температури, вологості, освітленості), стану виконавчих пристроїв та черг команд; – розробити алгоритми агрегації даних (погодинної та денної статистики) для оптимізації дискового простору сервера та пришвидшення роботи бази даних; – створити серверну частину (REST API), яка забезпечить авторизований обмін даними між апаратним мікроконтролером теплиці та мобільним додатком користувача у форматі JSON; – провести тестування розробленого програмного забезпечення на предмет цілісності даних та стійкості до навантажень.

Актуальність теми кваліфікаційної роботи зумовлена високою вартістю та закритістю комерційних систем "Розумного дому". Більшість готових рішень на ринку використовують пропрієтарні сервери, що унеможливлює розширення функціоналу або безкоштовне використання системи у довгостроковій перспективі. Розробка власної архітектури на базі відкритих веб-стандартів (PHP та MySQL) дозволить створити універсальний, незалежний та гнучкий продукт.

Галузь використання розробленого програмного забезпечення охоплює автоматизацію домашніх міні-теплиць, оранжерей, фермерських господарств малого масштабу, а також використання як навчального макета для дослідження процесів автоматизації екосистем. Результатом роботи стане готова серверна платформа, здатна ефективно агрегувати статистику, миттєво обробляти команди користувача та генерувати автоматичні тривожні сповіщення (алерти) у разі відхилення показників мікроклімату від норми.

1 ЗАГАЛЬНИЙ РОЗДІЛ

1.1 Аналітичний огляд існуючих рішень

Стрімкий розвиток концепції Інтернету речей (IoT) докорінно трансформує підходи до проектування інформаційних систем у найрізноманітніших галузях. Термін «Інтернет речей», уперше запропонований Кевіном Ештоном у 1999 році, описує мережу фізичних об'єктів, оснащених датчиками, програмним забезпеченням та засобами зв'язку, які обмінюються даними між собою та з хмарними серверами без безпосередньої участі людини. За прогнозами аналітичної компанії Statista, кількість підключених IoT-пристроїв у світі до 2030 року перевищить 29 мільярдів одиниць, а обсяг згенерованих ними даних зростатиме експоненційно. Проектування програмного забезпечення для «Розумної теплиці» є типовою задачею IoT, де мікроконтролери, зокрема ESP32 виробництва компанії Espressif Systems, виступають у ролі периферійних вузлів (Edge Nodes), що генерують безперервний потік телеметричних даних — температуру повітря, відносну вологість, рівень освітленості та вологість ґрунту. Головним завданням серверної частини такої системи є прийом, структурування, надійне збереження та подальша синхронізація цих даних із користувацьким інтерфейсом у режимі, максимально наближеному до реального часу.

Типова архітектура IoT-системи складається з трьох фундаментальних рівнів. Нижній рівень, який називається рівнем сприйняття (Perception Layer), охоплює фізичні датчики та виконавчі механізми, що безпосередньо взаємодіють із навколишнім середовищем теплиці. Середній рівень — мережевий (Network Layer) — відповідає за транспортування зібраних даних від периферійних вузлів до центрального сервера за допомогою протоколів бездротового зв'язку Wi-Fi, Bluetooth Low Energy або LoRaWAN. Верхній рівень — прикладний (Application Layer) — включає серверну інфраструктуру, бази

даних та користувацькі інтерфейси, що забезпечують візуалізацію інформації та віддалене керування системою. Саме від якості проектування верхнього, прикладного рівня значною мірою залежить загальна надійність, швидкодія та зручність експлуатації всієї системи автоматизації. Для вирішення цього комплексу завдань необхідно здійснити аналіз та вибір оптимальних рішень за трьома основними напрямками: технологія розробки серверної логіки (бекенду), система управління базами даних (СУБД) та протокол передачі інформації між компонентами системи.

Окремою і надзвичайно актуальною проблемою при проектуванні IoT-систем є забезпечення інформаційної безпеки. На відміну від традиційних веб-додатків, де взаємодія відбувається виключно між браузером користувача та сервером, в екосистемі Інтернету речей до сервера підключаються десятки або сотні автономних фізичних пристроїв, кожен з яких потенційно може бути скомпрометований зловмисником. Атаки типу «людина посередині» (Man-in-the-Middle), підміна телеметричних даних або несанкціоноване надсилання команд керування здатні завдати реальної фізичної шкоди — наприклад, вивести з ладу систему поливу або спричинити перегрів теплиці. Тому вже на етапі вибору технологічного стеку необхідно передбачити механізми автентифікації пристроїв, шифрування трафіку та розмежування прав доступу, що суттєво впливає на архітектурні рішення серверної частини.

Аналіз технологій розробки серверної логіки (API) є першим і, мабуть, визначальним кроком проектування, оскільки саме від обраної мови програмування та її екосистеми залежить швидкість розробки, можливості масштабування, вартість обслуговування та доступність хостингових платформ. Серверна логіка відповідає за обробку вхідних запитів, авторизацію пристроїв та користувачів, валідацію даних та взаємодію з базою даних. На сучасному ринку домінують кілька підходів до побудови бекенду для IoT-проектів, кожен з яких має свої переваги та обмеження.

Серверна логіка відповідає за обробку вхідних запитів, авторизацію пристроїв та користувачів, валідацію даних та взаємодію з базою даних. На сучасному ринку домінують кілька підходів до побудови бекенду.

Одним із найпопулярніших середовищ виконання серверного коду є Node.js — платформа, побудована на високопродуктивному рушії V8, розробленому компанією Google для браузера Chrome. Основною перевагою Node.js є його асинхронна подієво-орієнтована архітектура (Event Loop), що робить його ідеальним інструментом для додатків реального часу, таких як чати, ігрові сервери та системи потокової передачі даних. У контексті IoT Node.js чудово підтримує постійні з'єднання через протокол WebSockets, що дозволяє організувати миттєву двосторонню комунікацію між сервером та клієнтськими пристроями. Фреймворк Express.js, побудований на базі Node.js, забезпечує зручне створення RESTful API з мінімальним обсягом коду. Однак для невеликих домашніх IoT-проектів Node.js має суттєвий недолік: він вимагає налаштування віртуального виділеного сервера (VPS) або використання специфічних хмарних платформ (Heroku, AWS Lambda), що значно збільшує вартість утримання системи та потребує від розробника додаткових навичок системного адміністрування.

Іншим потужним кандидатом для реалізації серверної логіки IoT-систем є мова програмування Python із використанням сучасних фреймворків Django або FastAPI. Python de facto є стандартом у сфері аналізу даних, машинного навчання та штучного інтелекту, що відкриває унікальні перспективи для інтелектуальної обробки даних розумної теплиці. Зокрема, з використанням бібліотек TensorFlow або scikit-learn у майбутньому можна було б реалізувати прогнозування потреби рослин у поливі на основі аналізу часових рядів або автоматичне розпізнавання хвороб за фотографіями листя. Фреймворк FastAPI, заснований на стандарті ASGI, забезпечує асинхронну обробку запитів із автоматичною генерацією інтерактивної документації OpenAPI (Swagger). Водночас для класичного управління мікрокліматом теплиці, де основний потік

даних зводиться до прийому числових показників та відправки команд керування, весь арсенал Data Science може бути надлишковим, а розгортання Python-додатку на бюджетних хостингах залишається значно складнішим порівняно з іншими варіантами.

Третім і, як буде обґрунтовано далі, оптимальним варіантом для даного проекту є мова програмування PHP. Попри те, що PHP часто асоціюють із застарілими веб-рішеннями, сучасні версії цієї мови (7.4, 8.0, 8.1, 8.2) зазнали кардинального оновлення продуктивності завдяки вбудованому JIT-компілятору (Just-In-Time) та суворій типізації. Окрім того, PHP залишається найпоширенішою мовою серверного програмування у світі: за даними W3Techs, станом на 2024 рік приблизно 76,5% усіх веб-сайтів із відомою серверною технологією використовують саме PHP. Головна перевага PHP у контексті IoT-прототипів та домашньої автоматизації полягає у безпрецедентній простоті розгортання. PHP-скрипти нативно підтримуються абсолютною більшістю хостинг-провайдерів, включаючи безкоштовні. Класична синхронна модель обробки HTTP-запитів, схему якої наведено на рисунку 1.1 (див. рис. 1.1), у випадку IoT-телеметрії є цілком достатньою, оскільки мікроконтролер ініціює з'єднання самостійно із фіксованою періодичністю. Отже, PHP дозволяє реалізувати надійний REST API з мінімальними фінансовими витратами та зусиллями на адміністрування.

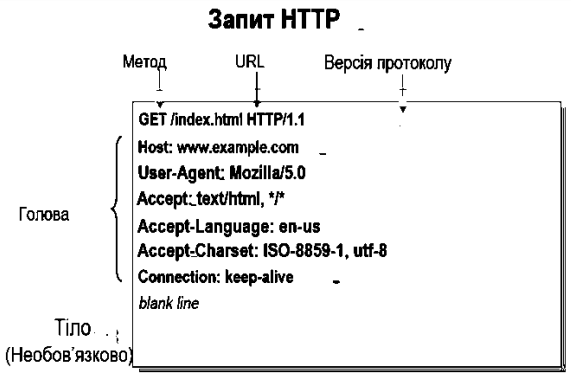


Рисунок 1.1 – схема обробки HTTP-запитів сервером

Враховуючи вимоги до універсальності, економічності та легкості підтримки, для реалізації серверної логіки розумної теплиці було обрано мову програмування PHP. Вона повністю задовольняє потребу у створенні швидкого REST API для прийому даних формату JSON від мікроконтролерів та забезпечує миттєве розгортання на будь-якій доступній хостинговій платформі. Порівняльний аналіз трьох розглянутих технологій засвідчив, що Node.js та Python пропонують потужніші можливості для побудови складних розподілених систем, проте для конкретного завдання домашньої автоматизації ці переваги є надлишковими і не компенсують значно вищих витрат на інфраструктуру.

Наступним критичним аспектом проектування серверної архітектури є вибір системи управління базами даних (СУБД). Збереження даних телеметрії є специфічним завданням, що висуває підвищені вимоги до пропускної здатності на запис та ефективності агрегаційних запитів. При частоті оновлення кожні 30 секунд один пристрій генерує понад 2800 записів на добу, або приблизно 86 000 записів на місяць. Якщо система обслуговує декілька теплиць одночасно, це число зростає пропорційно. Отже, обрана СУБД повинна не лише витримувати таке навантаження на запис, але й ефективно обробляти запити на вибірку для побудови графіків та обчислення статистичних показників.

Сучасний ринок систем управління базами даних пропонує три принципово різних підходи до зберігання IoT-телеметрії, кожен з яких оптимізований під специфічний патерн використання.

Першим із розглянутих підходів є використання спеціалізованих СУБД часових рядів (Time-Series DBMS), представниками яких є InfluxDB, TimescaleDB та Prometheus. Архітектура цих баз даних принципово відрізняється від класичних реляційних систем: вони оптимізовані для зберігання послідовностей числових значень, прив'язаних до міток часу (timestamps). Механізми автоматичного стиснення (compression), зменшення дискретизації (downsampling) та надшвидкі агрегаційні функції роблять такі СУБД ідеальним інструментом для зберігання сирих даних сенсорів. Наприклад,

InfluxDB здатна обробляти мільйони точок за секунду, що значно перевищує потреби домашньої теплиці. Водночас головним обмеженням цього класу баз даних є відсутність підтримки класичних реляційних зв'язків між таблицями. СУБД часових рядів не підходять для зберігання профілів користувачів, складних налаштувань порогів спрацювання автоматики, списків зареєстрованих пристроїв, черг команд керування та журналів авторизації — тобто всього того, що виходить за межі суто числової телеметрії.

Другим розглянутим варіантом є нереляційні, документо-орієнтовані СУБД (NoSQL), найвідомішим представником яких є MongoDB. Документо-орієнтовані бази зберігають інформацію у форматі JSON-подібних документів (BSON — Binary JSON у випадку MongoDB), що забезпечує високу гнучкість моделі даних. Схема документа може змінюватися динамічно, без необхідності виконання SQL-міграцій, що є зручним при частих змінах набору датчиків на ранніх етапах прототипування. Проте для системи автоматизації теплиці, де структура даних чітко визначена на етапі проектування і залишається стабільною впродовж усього життєвого циклу продукту, надмірна гнучкість NoSQL може обернутися проблемою: відсутність жорсткої схеми підвищує ризик порушення цілісності даних на етапі аналітичних запитів. Крім того, складні агрегаційні операції в MongoDB реалізуються через специфічний Aggregation Pipeline, який є менш інтуїтивним та менш оптимізованим порівняно зі стандартною мовою SQL.

Третій, класичний підхід полягає у використанні реляційних СУБД (RDBMS), найпоширенішими представниками яких є MySQL та PostgreSQL. Ці системи використовують структуровану мову запитів SQL та зберігають дані у жорстко типізованих таблицях, пов'язаних між собою за допомогою зовнішніх ключів (Foreign Keys). Головна перевага реляційних СУБД — забезпечення транзакційної цілісності відповідно до принципів ACID (Atomicity, Consistency, Isolation, Durability), що гарантує відсутність часткових або суперечливих записів навіть за умов одночасного звернення кількох клієнтів. У контексті

розумної теплиці СУБД MySQL дозволяє спроектувати комплексну архітектуру, яка об'єднує в єдиній базі таблиці облікових записів користувачів, криптографічні ключі для авторизації, черги команд керування пристроями, налаштування порогів спрацювання автоматики, журнали подій, денну та погодинну агреговану статистику, а також, власне, самі покази датчиків (див. рис. 1.2). Схематичне порівняння реляційної та документо-орієнтованої моделей зберігання наведено на рисунку 1.2.

Проблему швидкого накопичення великих масивів даних у реляційній СУБД MySQL можна ефективно вирішити засобами самої бази даних. MySQL версії 8.0 та вище підтримує механізм збережених процедур (Stored Procedures) — попередньо скомпільованих SQL-програм, що виконуються безпосередньо у ядрі СУБД без участі зовнішніх скриптів. Додатково MySQL надає планувальник подій (Event Scheduler) — вбудований аналог системного cron-планувальника, який дозволяє автоматично запускати збережені процедури за жорстко визначеним розкладом.

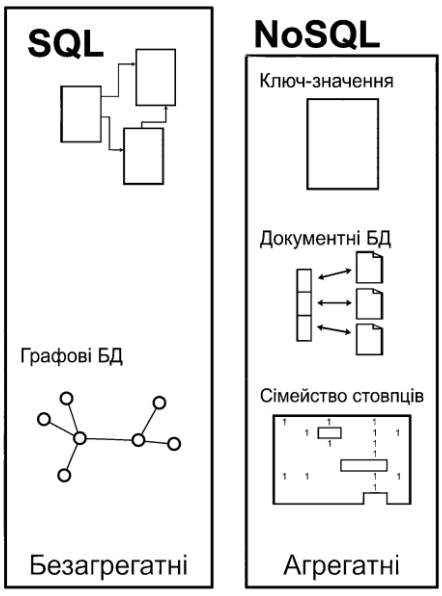


Рисунок 1.2 – Схематичне порівняння структури реляційної БД та документо-орієнтованої NoSQL БД

Завдяки комбінації цих двох механізмів можна реалізувати автоматичну агрегацію сирих даних у погодинну та денну статистику, паралельно видаляючи застарілі записи. Такий підхід забезпечує контрольоване зростання обсягу бази даних незалежно від кількості підключених пристроїв та інтенсивності генерації телеметрії. Саме тому MySQL є найбільш раціональним та збалансованим вибором СУБД для даного проекту.

Важливим аспектом оптимізації продуктивності реляційної бази даних є правильне проектування індексів. Індекс у MySQL — це допоміжна структура даних (зазвичай B-Tree або B+Tree), яка дозволяє СУБД знаходити потрібні рядки без повного перебору всієї таблиці (Full Table Scan). Для таблиці сирої телеметрії, що накопичує тисячі записів щодоби, критично важливим є індекс за полем часової мітки (created_at), який прискорює вибірку останніх записів для відображення на дашборді та побудови графіків. Додатково створюється композитний індекс за полями device_id та created_at, що оптимізує запити в архітектурі Multi-device, де фільтрація одночасно відбувається за ідентифікатором пристрою та часовим діапазоном. Для таблиці команд індексується поле status, оскільки найчастішим запитом є вибірка команд зі статусом «pending». Правильно спроектовані індекси дозволяють обробляти аналітичні запити за мілісекунди навіть при обсязі таблиці в сотні тисяч записів.

Завершальним етапом аналітичного огляду є вибір протоколу прикладного рівня для синхронізації інформації між апаратною теплицею, центральним сервером та мобільним додатком. Від обраного протоколу залежить затримка доставки даних, надійність каналу зв'язку та складність реалізації як на стороні мікроконтролера, так і на стороні серверного програмного забезпечення.

Сучасна практика IoT-розробки пропонує два домінуючих протоколи прикладного рівня, кожен з яких оптимізований під конкретний патерн обміну повідомленнями.

Першим розглянутим кандидатом є протокол MQTT (Message Queuing Telemetry Transport) — легковагий протокол обміну повідомленнями, створений

компанією IBM у 1999 році спеціально для каналів зв'язку з обмеженою пропускною здатністю. MQTT реалізує модель «видавець-підписник» (publish-subscribe), у якій IoT-пристрої публікують дані до іменованих тем (topics), а підписники автоматично отримують повідомлення з відповідних тем без необхідності постійного опитування сервера. Протокол вимагає мінімум накладних витрат на мережевому рівні, підтримує три рівні гарантії доставки (QoS 0, 1, 2) та механізм «останнього волевиявлення» (Last Will and Testament), який автоматично повідомляє інших учасників мережі про відключення пристрою. Однак для функціонування MQTT необхідний окремий програмний посередник — брокер (наприклад, Eclipse Mosquitto або HiveMQ), що ускладнює загальну архітектуру системи, збільшує кількість компонентів, які потрібно підтримувати, та робить систему залежною від додаткового серверного програмного забезпечення.

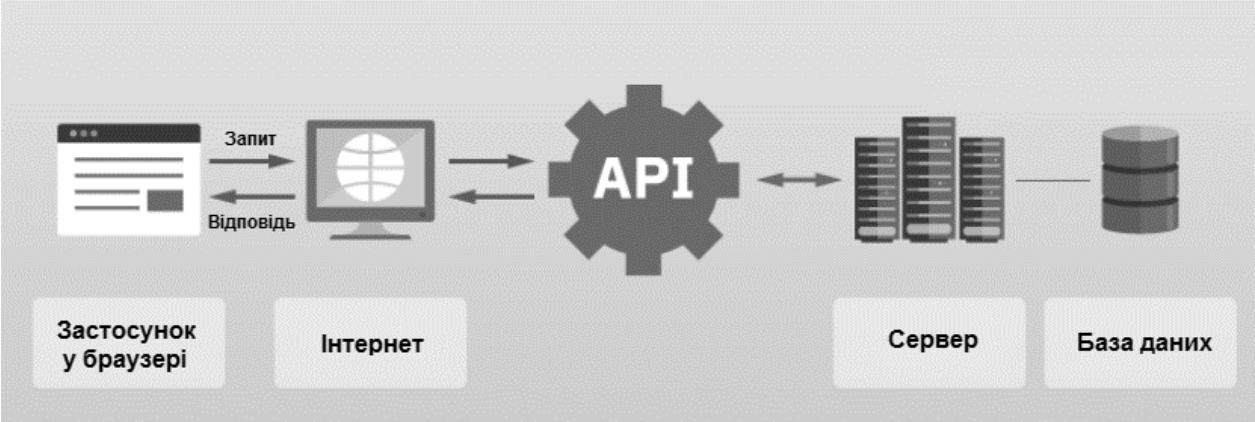


Рисунок 1.3 – Схема взаємодії клієнта та сервера за протоколом HTTP за архітектурою REST API

Альтернативним і значно простішим підходом є використання протоколу HTTP/HTTPS із дотриманням архітектурного стилю REST (Representational State Transfer), загальну схему якого наведено на рисунку 1.3 (див. рис. 1.3). REST API організує взаємодію за класичною моделлю «запит-відповідь»: клієнт (ESP32

або мобільний додаток) самостійно ініціює HTTP-з'єднання з сервером, передає серіалізовані дані у форматі JSON методом POST або отримує інформацію методом GET, після чого з'єднання негайно закривається. Незважаючи на дещо більший обсяг службових HTTP-заголовків порівняно з бінарними пакетами MQTT, архітектура REST API є максимально стандартизованою, легко тестується за допомогою будь-якого HTTP-клієнта (наприклад, Postman або cURL) та ідеально інтегрується з PHP-скриптами, які нативно працюють всередині HTTP-серверів Apache або Nginx. Для системи домашньої теплиці з одним-двома пристроями різниця в латентності між MQTT та HTTP є нехтовно малою та не впливає на якість управління мікрокліматом.

Синхронізація стану виконавчих механізмів, наприклад увімкнення фітолампи або відкриття клапана поливу, за допомогою REST API реалізується через архітектурний патерн «черги команд» (Command Queue). Суть цього механізму полягає в тому, що клієнтський мобільний додаток відправляє команду на сервер, де вона зберігається у спеціальній таблиці бази даних зі статусом «очікує виконання». Мікроконтролер теплиці під час чергового сеансу зв'язку із сервером, який відбувається кожні 30 секунд, разом із відправкою свіжої телеметрії автоматично запитує список невиконаних команд. Сервер віддає ці команди та негайно оновлює їхній статус на «виконано», запобігаючи повторному зчитуванню. Такий асинхронний підхід забезпечує надійну доставку інструкцій навіть за умов нестабільного з'єднання, оскільки команда зберігатиметься в базі до тих пір, поки мікроконтролер не підтвердить її отримання.

Проведений аналітичний огляд існуючих рішень засвідчив, що для розробки бази даних та системи синхронізації домашньої розумної теплиці найбільш доцільним є використання стеку технологій PHP + MySQL + HTTP (REST API). Цей стек забезпечує оптимальний баланс між продуктивністю, надійністю збереження структурованої інформації та простотою розгортання на будь-яких доступних хостингових платформах, включаючи безкоштовні.

Обрана реляційна СУБД MySQL надає потужний внутрішній інструментарій для автоматизації обслуговування даних, а протокол HTTP ідеально інтегрується з мовою PHP, що усуває потребу у додаткових серверних компонентах, таких як MQTT-брокери. Наступним етапом роботи є формування детального технічного завдання на розробку бази даних та серверного програмного забезпечення на основі обраного технологічного стеку

Таким чином, проведений аналіз трьох технологічних напрямків — серверної мови програмування, СУБД та протоколу обміну даними — дозволив сформувати цілісний та збалансований технологічний стек для реалізації серверної підсистеми розумної теплиці. Обрана комбінація PHP + MySQL + REST API не є абсолютним лідером за жодним із окремих критеріїв (швидкодія, гнучкість, реальний час), проте саме ця комбінація забезпечує найкращий сукупний результат за сумою параметрів: вартість розгортання, простота підтримки, доступність хостингу, зрілість екосистеми, безпека зберігання даних та широка документованість. Це підтверджується тим, що абсолютна більшість домашніх IoT-проектів, описаних у відкритих джерелах, використовують саме подібну архітектуру.

1.2 Технічне завдання

Даний підрозділ є основним нормативним документом, що встановлює вимоги до розробки серверної частини (бази даних та програмного інтерфейсу API) для системи автоматизованого керування мікрокліматом «Розумна теплиця». Структура технічного завдання сформована з урахуванням стандартів розробки програмної документації та охоплює призначення розробки, функціональні вимоги, вимоги до надійності й безпеки, а також специфікації програмної та апаратної сумісності.

1.2.1 Призначення та сфера застосування розробки

Повна назва розроблюваного програмного продукту: «Підсистема збереження, обробки та синхронізації даних комплексної системи Розумної теплиці». Програмний продукт призначений для централізованого керування інформаційними потоками між апаратним забезпеченням теплиці (мікроконтролерами з підключеними датчиками) та користувацьким інтерфейсом (мобільним додатком або веб-панеллю).

Основними цілями створення підсистеми є: забезпечення безперервного цілодобового прийому телеметричної інформації від мікроконтролерів із частотою запитів до одного разу на 30 секунд, надійне збереження сирих даних мікроклімату та їх подальша алгоритмічна агрегація для економії дискового простору сервера, організація захищеного каналу зв'язку для авторизації пристроїв та користувачів, маршрутизація та буферизація керуючих команд між мобільним додатком та контролером теплиці, а також автоматичний моніторинг показників на стороні сервера для генерації тривожних сповіщень (алертів) у разі виникнення критичних ситуацій.

Сфера застосування розроблюваної підсистеми охоплює автоматизацію домашніх теплиць, невеликих фермерських господарств, оранжерей, навчальних лабораторій з дослідження процесів автоматизації екосистем, а також використання як базової серверної платформи для розгортання масштабованих агропромислових IoT-рішень.

1.2.2 Вимоги до функціональних характеристик

Розроблювана база даних та серверний API повинні забезпечувати виконання комплексу функцій, які можна класифікувати за чотирма основними напрямками.

У сфері керування користувачами та безпеки система повинна забезпечувати повний цикл реєстрації нових користувачів із валідацією вхідних даних, включаючи перевірку унікальності логіна та вимоги до мінімальної довжини пароля. Авторизація користувачів має здійснюватися з використанням алгоритму хешування bcrypt, що унеможливлює відновлення пароля навіть у разі компрометації бази даних. Для ідентифікації мобільних додатків та фізичних мікроконтролерів система повинна автоматично генерувати унікальні криптографічні токени (API-ключі формату SHA-256). Розмежування прав доступу реалізується на рівні бази даних через механізм дозволів, що включає чотири рівні: читання, запис, керування та адміністрування. Усі спроби авторизації, як успішні, так і невдалі, фіксуються у спеціалізованому журналі подій бази даних із записом IP-адреси та ідентифікатора агента клієнта.

У частині обробки та збереження даних телеметрії система повинна забезпечувати прийом масивів даних від датчиків через HTTP POST-запити. Сервер обробляє такі параметри мікроклімату: температуру повітря в градусах Цельсія, вологість повітря у відсотках, рівень освітленості у відсотках від максимального значення, а також вологість ґрунту мінімум у трьох незалежних зонах поливу. Окрім кліматичних показників, кожен пакет містить поточний стан виконавчих механізмів — статус вентилятора, фітолампи та трьох сервоприводів (клапанів поливу). Для кожного отриманого пакету СУБД автоматично генерує часову мітку з точністю до секунди. Архітектура підтримує багатопристроєвість (Multi-device): кожен запис прив'язується до конкретного ідентифікатора пристрою, що дозволяє обслуговувати кілька теплиць із одного серверного інстансу.

Функції агрегації та автоматизації реалізуються безпосередньо на рівні СУБД, без участі зовнішніх скриптів. Система повинна автоматично обчислювати середні, мінімальні та максимальні показники мікроклімату за кожну годину шляхом виклику збережених процедур, а також формувати денну статистику, що включає загальну кількість вимірювань, тривалість роботи

вентиляторів та ламп у секундах і кількість сеансів поливу. Додатково реалізується алгоритмічне виявлення аномальних значень датчиків, таких як різкі стрибки показників або обриви зв'язку. Для запобігання переповненню дискового простору система автоматично очищує застарілі сирі дані, вік яких перевищує конфігурований поріг (за замовчуванням — 30 днів).

Функції синхронізації та маршрутизації команд передбачають прийом від клієнтського додатку інструкцій керування виконавчими механізмами та їх розміщення у спеціальній таблиці черги зі статусом очікування. Мікроконтролер під час кожного сеансу зв'язку з сервером отримує список невиконаних команд, після чого їхній статус автоматично змінюється на «виконано». Система також забезпечує збереження та синхронізацію індивідуальних налаштувань кожної теплиці — порогів температури, критичних рівнів вологості ґрунту, режимів роботи (автоматичний або ручний).

1.2.3 Вимоги до надійності та безпеки

Забезпечення цілісності даних є пріоритетною вимогою до серверної архітектури. База даних повинна використовувати рушій зберігання InnoDB, який підтримує ACID-транзакції та механізм зовнішніх ключів (Foreign Keys). Каскадне видалення облікового запису користувача повинно автоматично знищувати всі пов'язані з ним API-ключі та налаштування пристроїв. Відмовостійкість забезпечується обов'язковою обробкою помилок з'єднання з базою даних у блоках try-catch: у разі недоступності СУБД скрипти API повинні повертати клієнту HTTP-статус 500 (Internal Server Error) із відповідним повідомленням у форматі JSON, не допускаючи виведення системних трасувань або структури таблиць.

Захист від несанкціонованого доступу реалізується через обов'язкову передачу валідного криптографічного токена у HTTP-заголовку X-API-Key для

всіх ендпоінтів, що стосуються зміни стану теплиці або зчитування історичних даних. Єдиним винятком є публічні ендпоінти авторизації та реєстрації.

Обробка вхідних даних повинна здійснюватися з дотриманням принципу «ніколи не довіряй вхідним даним». Усі значення, що надходять від зовнішніх клієнтів через системний потік php://input або параметри GET/POST, проходять процедуру десеріалізації та валідації типів, включаючи примусове приведення до числових типів float та integer. Для роботи з базою даних використовуються виключно параметризовані підготовлені запити (Prepared Statements) бібліотеки PDO, що повністю виключає можливість проведення атак типу SQL-ін'єкцій.

1.2.4 Вимоги до інформаційної та програмної сумісності

Обмін даними між апаратним забезпеченням, сервером та мобільним додатком здійснюється за протоколом HTTP/1.1 (або HTTPS при наявності SSL-сертифіката) з використанням архітектурного стилю REST. Як єдиний формат представлення вхідних та вихідних даних використовується JSON (JavaScript Object Notation) із обов'язковим кодуванням символів UTF-8, що забезпечує коректне збереження кириличних назв теплиць та текстових повідомлень.

Для забезпечення сумісності з мобільними гібридними додатками (PWA) та веб-клієнтами серверний API повинен явно віддавати HTTP-заголовки Access-Control-Allow-Origin: * та підтримувати обробку попередніх запитів методом OPTIONS відповідно до специфікації Cross-Origin Resource Sharing (CORS).

1.2.5 Вимоги до складу і параметрів технічних засобів

Розроблювана система повинна функціонувати у гетерогенному мережевому середовищі без жорстких вимог до високопродуктивного обладнання, що є ключовою перевагою обраного технологічного стеку.

Серверне програмне забезпечення потребує операційної системи GNU/Linux (Ubuntu 20.04 або новіше, Debian, CentOS) або іншої UNIX-подібної ОС. Веб-сервер Apache 2.4 або Nginx забезпечує обробку HTTP-запитів та передачу їх інтерпретатору PHP версії не нижче 7.4 із встановленими розширеннями PDO, pdo_mysql та json. Система управління базами даних MySQL версії 8.0 або вище (або сумісний форк MariaDB 10.4+) повинна підтримувати виконання подій, тобто планувальник Event Scheduler має бути переведений у активний стан. Структурну схему взаємодії програмних та апаратних компонентів системи наведено на рисунку 1.4 (див. рис. 1.4).

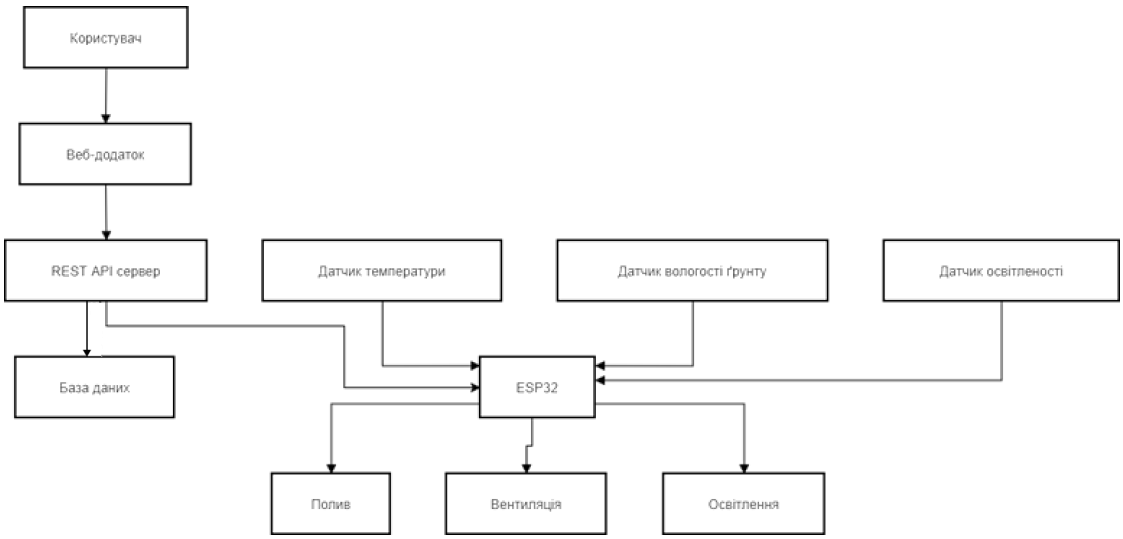


Рисунок 1.4 – Структурна схема взаємодії програмних та апаратних компонентів системи за REST API

Мінімальні параметри серверного апаратного забезпечення (VPS або Shared Hosting) для обслуговування до 100 активних пристроїв включають: процесор із частотою від 1 ГГц (одне ядро), обсяг оперативної пам'яті від 512 МБ, дисковий простір на твердотільному накопичувачі (SSD) від 1 ГБ для розміщення скриптів та файлів бази даних. Пропускна здатність інтернет-каналу сервера повинна становити не менше 10 Мбіт/с для уникнення затримок під час одночасного звернення мікроконтролерів.

Клієнтська апаратна частина системи, що розміщується безпосередньо у теплиці, будується на базі мікроконтролера ESP32 або його аналогів із вбудованим модулем Wi-Fi (стандарту 802.11 b/g/n). Апаратна платформа має забезпечувати керування виконавчими механізмами (системами поливу, вентиляції та освітлення) через блоки реле з відповідною напругою та струмом комутації.

Для забезпечення безперебійної передачі даних на сервер об’єкт автоматизації має бути оснащений локальним маршрутизатором із підключенням до мережі Інтернет. Маршрутизатор повинен забезпечувати стабільне покриття Wi-Fi мережі у зоні розміщення мікроконтролера.

2 РОЗРОБКА ТЕХНІЧНОГО ТА РОБОЧОГО ПРОЕКТУ

2.1 Розробка структури бази даних

Основою серверної частини системи «Розумна теплиця» є оптимізована реляційна база даних, що працює під керуванням СУБД MySQL версії 8.0. Вона спроектована з дотриманням правил нормалізації до третьої нормальної форми (3NF), що дозволяє уникнути надмірності інформації, забезпечити транзакційну цілісність та мінімізувати ризик аномалій при оновленні та видаленні записів. Нормалізація — це процес організації атрибутів і таблиць реляційної бази даних таким чином, щоб мінімізувати дублювання даних та забезпечити логічну залежність між таблицями виключно через механізм зовнішніх ключів. База отримала назву smart_greenhouse і концептуально поділена на декілька логічних підсистем, які тісно взаємодіють між собою. Для кодування символів обрано набір utf8mb4 із зіставленням utf8mb4_unicode_ci, що забезпечує повну підтримку Юнікоду, включаючи кириличні символи в назвах теплиць та текстових сповіщеннях.

Теорія нормалізації реляційних баз даних, запропонована Едгаром Коддом у 1970 році, визначає послідовні рівні організації даних, кожен з яких усуває певний клас аномалій. Перша нормальна форма (1NF) вимагає, щоб кожне поле таблиці містило атомарне (неподільне) значення. У контексті розроблюваної бази даних це означає, що показники трьох зон поливу зберігаються в окремих полях soil1, soil2 та soil3, а не у вигляді масиву або серіалізованого рядка. Друга нормальна форма (2NF) усуває часткову залежність неключових атрибутів від складеного первинного ключа, що досягається використанням сурогатних автоінкрементних ключів у більшості таблиць. Третя нормальна форма (3NF) вимагає відсутності транзитивних залежностей: наприклад, назва теплиці зберігається виключно у таблиці devices і не дублюється у таблицях телеметрії

чи команд, а при потребі отримується через операцію JOIN за зовнішнім ключем device_id.

Підсистема ідентифікації та безпеки базується на двох взаємопов'язаних таблицях: users та api_keys. Таблиця users зберігає облікові записи власників теплиць, включаючи унікальні логіни, паролі, зашифровані за алгоритмом bcrypt (з вартістю хешування cost=10), та рівні доступу, які визначаються полем role типу ENUM із трьома можливими значеннями: admin, user та viewer. Таблиця api_keys реалізує модель авторизації на основі токенів: кожен ключ являє собою 64-символьний рядок, згенерований криптографічно безпечним генератором випадкових чисел. Ключі класифікуються за типом (device — для мікроконтролерів, app — для мобільних додатків) та мають набір гранулярних дозволів, реалізований через тип SET у MySQL. Зв'язок між таблицями забезпечується зовнішнім ключем із каскадним видаленням (ON DELETE CASCADE), що автоматично знищує всі ключі при видаленні облікового запису. Усі події авторизації фіксуються в журналі auth_log із записом IP-адреси та ідентифікатора агента клієнта.

Принципи ACID, на яких базується рушій InnoDB, забезпечують чотири фундаментальних гарантії при роботі з транзакціями. Атомарність (Atomicity) гарантує, що кожна операція запису телеметрії або створення команди виконується повністю або не виконується взагалі — часткових записів у базі не існує. Узгодженість (Consistency) забезпечує, що після кожної транзакції база даних переходить з одного валідного стану в інший, зберігаючи цілісність зовнішніх ключів та обмежень типів. Ізольованість (Isolation) означає, що одночасні запити від кількох мікроконтролерів не інтерферують між собою: кожна транзакція бачить узгоджений знімок даних. Довговічність (Durability) гарантує, що після підтвердження транзакції дані зберігаються на диску навіть у разі раптового відключення електроживлення сервера. Ці гарантії є особливо важливими для IoT-систем, де втрата навіть одного пакету телеметрії може

призвести до некоректного спрацювання алгоритмів автоматичного керування мікрокліматом.

Для підтримки масштабованості та можливості керування кількома теплицями з одного облікового запису реалізовано архітектуру Multi-device на основі таблиці devices. Ця таблиця є центральним реєстром фізичних мікроконтролерів, де кожен запис має унікальний текстовий ідентифікатор (наприклад, greenhouse_01), що використовується як зовнішній ключ у більшості інших таблиць системи. Найважливішою таблицею з точки зору обсягу даних є sensor_data — таблиця безперервного накопичення сирої телеметрії. Кожні 30 секунд до неї записуються значення температури повітря (тип DECIMAL(5,1)), вологості повітря, рівня освітленості, вологості ґрунту по трьох зонах поливу, а також поточний стан п'яти виконавчих реле (вентилятор, фітолампа та три клапани поливу), представлений типом TINYINT(1). Для оптимізації швидкості запитів на вибірку таблиця проіндексована за полем created_at, що забезпечує швидке зчитування останніх записів.

Column	Type	Function	Null	Value
id	bigint unsigned			9
device_id	varchar(32)			greenhouse_01
temperature	decimal(5,1)			36.3
humidity	decimal(5,1)			40.0
lux	decimal(5,1)			24.0
soil1	decimal(5,1)			100.0
soil2	decimal(5,1)			100.0
soil3	decimal(5,1)			100.0
fan	tinyint(1)			1
lamp	tinyint(1)			1
servo1	tinyint(1)			0
servo2	tinyint(1)			0
servo3	tinyint(1)			0

Рисунок 2.1 — Структура таблиці збереження телеметрії sensor_data

Оскільки масив сирої телеметрії зростає зі швидкістю понад 2800 записів на добу на один пристрій, у базі спроектовано два рівні агрегації: погодинну (таблиця sensor_data_hourly) та денну (таблиця daily_stats). Погодинна агрегація зберігає середні, мінімальні та максимальні значення кожного датчика за одну годину, а також кількість вимірювань, на основі яких обчислено статистику. Денна агрегація додатково включає сумарний час роботи вентиляторів та ламп, кількість сеансів поливу і кількість згенерованих тривожних сповіщень. Для маршрутизації команд керування використовується таблиця commands, яка виконує роль асинхронної черги: клієнтський додаток створює запис із зазначенням цільового пристрою та бажаного стану, а мікроконтролер зчитує невиконані команди при наступному сеансі зв'язку. Життєвий цикл команди відображається полем status типу ENUM зі значеннями pending, executed, expired та cancelled. Індивідуальні параметри спрацювання автоматики для кожної теплиці розміщені у таблиці settings з композитним первинним ключем (device_id, setting_key). Окрім того, у базі присутні таблиці для зберігання профілів рослин (plant_profiles), сесій поливу (irrigation_sessions), обліку енергоспоживання (energy_usage), моніторингу здоров'я системи (system_health) та автоматично виявлених аномалій (anomalies).

Окрім базових таблиць, у структурі бази даних реалізовано систему представлень (Views) — віртуальних таблиць, що зберігають попередньо визначені SQL-запити і дозволяють спростити доступ до часто запитуваних даних. Представлення v_latest_data повертає останній запис телеметрії для кожного зареєстрованого пристрою, v_active_alerts — список непрочитаних тривожних сповіщень, v_pending_commands — чергу команд, що очікують на виконання, а v_today_stats — агреговану статистику за поточну добу. Використання представлень не лише скорочує обсяг PHP-коду, але й підвищує безпеку, оскільки зовнішні клієнти працюють з обмеженим набором полів, не маючи прямого доступу до службових колонок таблиць. Додатково розроблено представлення для аналізу ефективності поливу по зонах (v_irrigation_stats),

енергоспоживання за тиждень (v_weekly_energy), здоров'я системи (v_system_health) та нерозв'язаних аномалій (v_open_anomalies), що забезпечують комплексний моніторинг стану всієї екосистеми теплиці.

Для наочного відображення архітектури та зв'язків між усіма описаними сутностями була розроблена ER-діаграма (Entity-Relationship Diagram), яку наведено на рисунку 2.2 (див. рис. 2.2). Діаграма демонструє зв'язки типу «один-до-багатьох» між таблицями devices та sensor_data, commands, alerts, а також зв'язок між users та api_keys із каскадним видаленням.

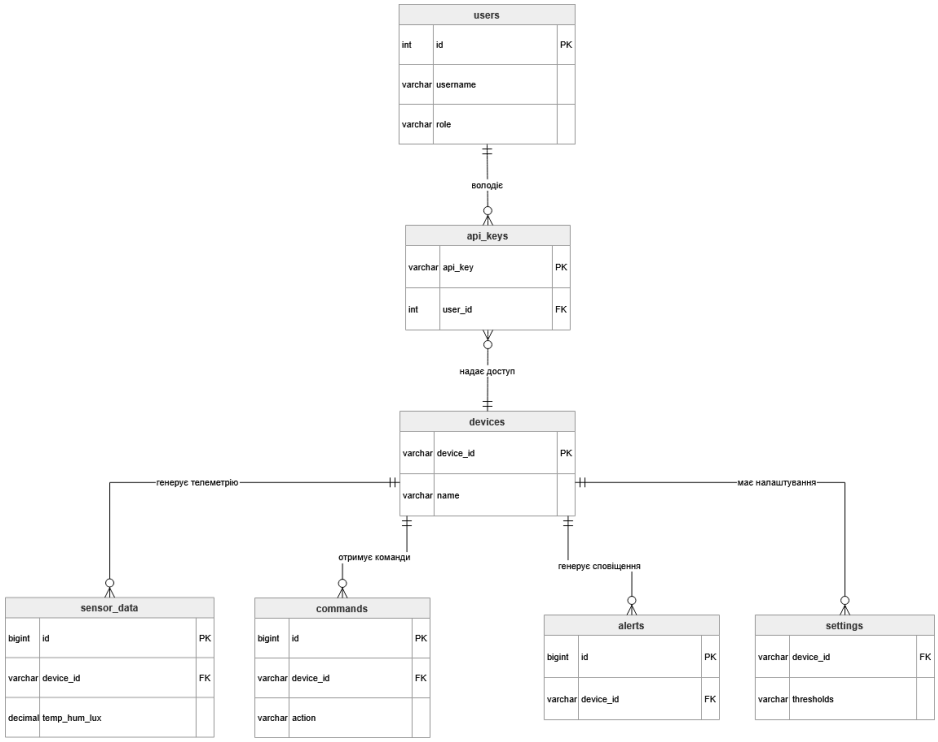


Рисунок 2.2 — ER-діаграма бази даних (схема зв'язків між таблицями)

2.2 Опис методів організації вхідних та вихідних даних

Проектування ефективної системи автоматизації розумної теплиці вимагає детальної структуризації всіх інформаційних потоків, що циркулюють між фізичним обладнанням, сервером та мобільним інтерфейсом користувача. Організація вхідних і вихідних даних базується на архітектурному стилі REST

API, який передбачає чітке розділення операцій на ресурсно-орієнтовані ендпоінти з використанням стандартних HTTP-методів GET та POST. Як єдиний універсальний засіб представлення інформаційних пакетів обрано текстовий формат JSON (JavaScript Object Notation). Цей формат дозволяє будувати складні деревоподібні структури даних, які легко піддаються серіалізації за допомогою вбудованої бібліотеки ArduinoJson на стороні мікроконтролера ESP32 та десеріалізації у мобільному додатку, зберігаючи при цьому мінімальний обсяг службових заголовків порівняно з альтернативними форматами, такими як XML або Protocol Buffers.

Вхідні дані, що надходять від апаратного контролера теплиці, являють собою безперервний потік телеметричних показників, який передається на сервер за допомогою HTTP-методу POST. Кожен такий інформаційний пакет містить повний набір числових значень, знятих із датчиків мікроклімату у реальному часі. До цих даних відносяться температура повітря, вологість навколишнього середовища, рівень штучного або природного освітлення, а також показники вологості ґрунту, отримані одночасно з трьох різних зон вирощування рослин. Окрім суто кліматичних параметрів, мікроконтролер обов'язково додає до вхідного пакету поточний статус своїх виконавчих пристроїв, повідомляючи сервер про те, чи увімкнена в даний момент фітолампа, чи працює вентилятор та в якому стані перебувають сервоприводи або соленоїдні клапани кожної з трьох зон поливу. Усі ці параметри передаються у тілі запиту і зчитуються PHP-скриптом через спеціалізований системний потік, після чого проходять обов’язкову перевірку на відповідність типам даних.

Невід'ємною частиною вхідного потоку від апаратної частини є службова інформація, необхідна для безпеки та ідентифікації. Разом із масивом телеметрії мікроконтролер передає унікальний ідентифікатор пристрою та криптографічний токен доступу, який розміщується у HTTP-заголовку запиту під назвою X-API-Key. Наявність цього ключа дозволяє серверній логіці миттєво відсікати неавторизовані запити, захищаючи базу даних від підміни показників

або зловмисного переповнення. Сервер аналізує отриманий ключ, перевіряє його статус у таблиці активних токенів і лише після успішної автентифікації дозволяє транзакцію запису в основну таблицю телеметрії. При цьому на рівні сервера відбувається автоматичне збагачення вхідного пакету: СУБД самостійно генерує унікальний ідентифікатор запису та додає точну часову мітку отримання інформації.

Вихідні дані, які сервер формує у відповідь на запит мікроконтролера, мають комбіновану структуру та виконують роль зворотного зв'язку. Після успішного збереження телеметрії серверне API виконує вибірку з бази даних і надсилає апаратному комплексу JSON-пакет, що містить чергу команд керування, які очікують на виконання. Цей вихідний пакет включає унікальні ідентифікатори команд, назви цільових пристроїв, для яких призначена дія, та конкретні значення станів, які необхідно прийняти. Окрім черги команд, сервер передає мікроконтролеру актуальні порогові значення автоматики, такі як максимальна та мінімальна дозволена температура або критичні рівні сухості ґрунту, що дозволяє апаратній частині коригувати свою автономну логіку роботи навіть у разі тимчасової втрати зв'язку із сервером. Після успішної віддачі цього пакету сервер автоматично змінює статус згаданих команд у базі даних на виконані.

Організація інформаційного обміну з мобільним додатком користувача має дещо інший характер, оскільки додаток вимагає різноманітних вихідних структур залежно від відкритого екрана інтерфейсу. Найбільш об'ємним вихідним потоком є дані для головного екрана або дашборду. Сервер формує комплексний JSON-об'єкт, всередині якого згруповано останній актуальний запис із датчиків, поточний режим роботи всієї системи (автоматичний або ручний), кількість непрочитаних критичних сповіщень про аварії, а також детальний список і мережевий статус усіх зареєстрованих у користувача теплиць. Окремо всередині цього ж об'єкта передається масив денної статистики, який містить попередньо розраховані середні, мінімальні та

максимальні показники клімату за поточну добу, що дозволяє мобільному додатку миттєво відображати аналітику без виконання складних розрахунків на стороні смартфона.

Для побудови графіків у мобільному додатку організовано окремий вихідний потік історичних даних, структура якого гнучко змінюється залежно від обраного користувачем часового періоду. Якщо додаток запитує історію за останні двадцять чотири години, сервер повертає масив сирих точок телеметрії, оптимізований за допомогою математичного кроку вибірки для уникнення перевантаження графічного процесора мобільного пристрою. У разі запиту історії за тиждень або місяць, серверне API звертається до таблиць погодинної та щоденної агрегації, формуючи компактний вихідний JSON-масив, що містить виключно усереднені тренди (див рис2.3). Нарешті, вхідні дані від мобільного додатка до сервера включають POST-запити на ручну зміну стану реле або оновлення конфігураційних порогів, де кожне значення валідується сервером на відповідність допустимим діапазонам перед оновленням записів у таблиці налаштувань.

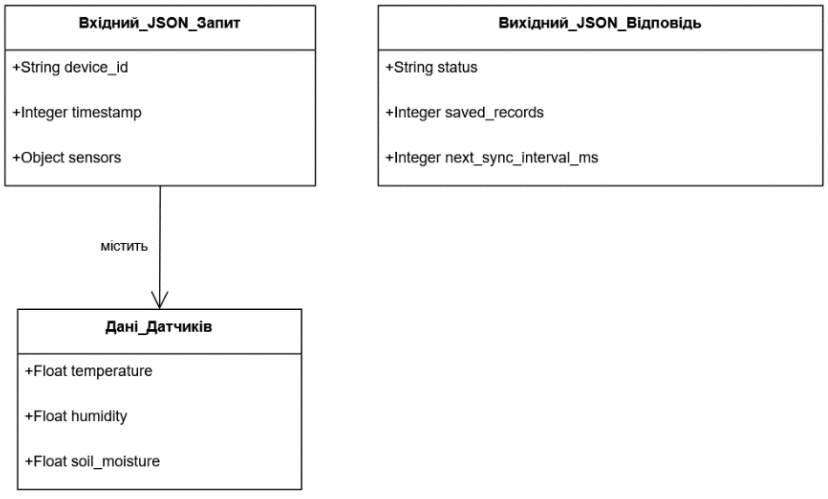


Рисунок 2.3 — Схема структури вхідних та вихідних JSON-пакетів при обміні даними

2.3 Програмування серверної частини та бази даних

Реалізація програмної частини серверної підсистеми розумної теплиці виконана мовою програмування PHP сьомої версії у поєднанні з мовою структурованих запитів SQL. Процес програмування було розділено на кілька логічних етапів: розробка конфігураційного модуля, програмування ядра маршрутизації API, реалізація криптографічних механізмів авторизації, написання обробників для телеметрії та команд, а також програмування внутрішніх алгоритмів на рівні системи управління базами даних. Для забезпечення максимальної швидкодії та гнучкості архітектури було прийнято рішення відмовитися від використання важких сторонніх фреймворків на користь чистого PHP з використанням вбудованих об'єктно-орієнтованих бібліотек.

2.3.1 Розробка конфігураційного модуля та підключення до бази даних

Будь-яка масштабована програмна система вимагає чіткого відокремлення конфігураційних параметрів від основної бізнес-логіки. Для реалізації цього принципу було створено окремий програмний файл конфігурації. У цьому файлі запрограмовано механізм гнучкого перемикання середовищ розробки за допомогою визначення глобальної константи режиму роботи системи. Це архітектурне рішення дозволяє програмному коду автоматично адаптуватися під локальний веб-сервер (наприклад, XAMPP) під час тестування функціоналу, або під параметри бойового хостингу під час розгортання в мережі Інтернет. Залежно від значення цієї константи, умовний оператор ініціалізує відповідні реквізити доступу до системи управління базами даних: адресу сервера, ім'я бази даних, логін користувача та його пароль. Лістинг коду конфігураційного модуля із розгалуженням середовищ розробки наведено на рисунку 2.4 (див. рис. 2.4).

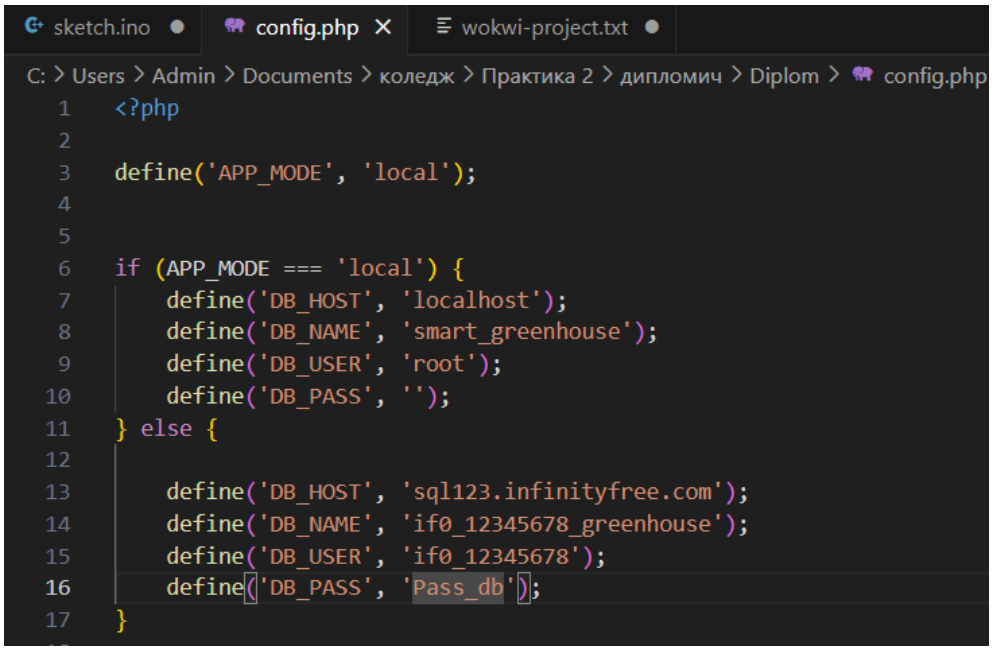


Рисунок 2.4 — Лістинг коду: Файл config.php з налаштуваннями констант

Безпосереднє підключення до бази даних MySQL запрограмовано з використанням сучасного розширення PHP Data Objects (PDO), яке надає абстрактний рівень доступу до даних. Ініціалізація об'єкта підключення відбувається всередині блоку перехоплення виняткових ситуацій (try-catch), що дозволяє уникнути виведення системних помилок користувачу у разі падіння бази даних.

При створенні екземпляра класу PDO примусово задається кодування UTF-8, що є критично важливим для коректного збереження кириличних назв теплиць та текстових тривожних сповіщень. Крім того, програмно вимикається режим емуляції підготовлених запитів, що змушує ядро MySQL самостійно компілювати SQL-вирази, тим самим забезпечуючи абсолютний захист від атак типу SQL-ін'єкцій.

Якщо підключення зазнає невдачі, скрипт негайно припиняє роботу і віддає клієнту серіалізований JSON-об'єкт із повідомленням про тимчасову недоступність сервісу.

2.3.2 Програмування ядра маршрутизації та допоміжних функцій

Основним вхідним вузлом розробленої системи є файл маршрутизації програмного інтерфейсу. Його виконання починається з налаштування параметрів HTTP-відповіді. Оскільки до серверної частини можуть звертатися клієнти з різних доменів або безпосередньо з мобільних пристроїв, скрипт програмно встановлює заголовки, які дозволяють політику спільного використання ресурсів з різними джерелами (CORS). Також встановлюється заголовок типу контенту application/json, який вказує браузерам та мікроконтролерам на те, що всі відповіді сервера будуть виключно у форматі об'єктів JavaScript.

Для уникнення дублювання коду було запрограмовано ряд допоміжних функцій. Функція форматування відповіді приймає числовий HTTP-код стану та масив даних, після чого встановлює код на рівні веб-сервера, перетворює масив у JSON-рядок і завершує виконання скрипта. Інша допоміжна функція відповідає за перевірку методу запиту (GET або POST), відхиляючи всі звернення, які не відповідають архітектурі конкретного ендпоінта. Фрагмент коду із допоміжними функціями форматування JSON-відповідей наведено на рисунку 2.5 (див. рис. 2.5).

Для прийому складних структур даних від мікроконтролера була запрограмована окрема функція читання вхідного потоку. Замість використання стандартних глобальних масивів PHP, які підходять лише для звичайних веб-форм, функція зчитує сирі байти безпосередньо з системного потоку вводу php://input.

Після цього алгоритм здійснює спробу десеріалізації цих байтів. Якщо вхідні дані мають порушену структуру або не є валідним JSON-об'єктом, функція перериває виконання і повертає клієнту помилку з кодом 400 (Bad Request).

```php
function respond(int $code, array $data): void {
    http_response_code($code);
    echo json_encode($data, JSON_UNESCAPED_UNICODE);
    exit;
}

function requireMethod(string $method): void {
    if ($_SERVER['REQUEST_METHOD'] !== $method) {
        respond(405, ['error' => "Метод має бути $method"]);
    }
}

function getJsonInput(): array {
    $raw = file_get_contents('php://input');
    $data = json_decode($raw, true);
    if (!is_array($data)) {
        respond(400, ['error' => 'Невалідний JSON']);
    }
    return $data;
}

function logEvent(string $source, string $type, string $desc): void {
    global $pdo;
    $stmt = $pdo->prepare("
        INSERT INTO event_log (event_source, event_type, description)
        VALUES (:src, :type, :desc)
    ");
    $stmt->execute(['src' => $source, 'type' => $type, 'desc' => $desc]);
}
```

Рисунок 2.5 — Лістинг коду: Допоміжні функції respond() та requireMethod()

2.3.3 Програмування алгоритмів автентифікації та перевірки прав доступу

Захист даних є ключовою вимогою до IoT-систем, тому значну увагу було приділено програмуванню модулів автентифікації. Більшість маршрутів системи є закритими і вимагають наявності криптографічного токена. Алгоритм безпеки розпочинається зі спроби витягти ключ доступу з HTTP-заголовків запиту (X-API-Key). Якщо заголовок відсутній, програма перевіряє наявність ключа у параметрах URL-адреси. Якщо ключ не знайдено в жодному з джерел, скрипт генерує помилку доступу з кодом 401 (Unauthorized). Після успішного отримання рядка ключа, програма формує параметризований SQL-запит до таблиці ключів, об'єднуючи її з таблицею користувачів. Лістинг коду обробника автентифікації та валідації API-ключа наведено на рисунку 2.6 (див. рис. 2.6).

```php
$action    = $_GET['action'] ?? '';
$deviceId  = $_GET['device_id'] ?? $_POST['device_id'] ?? 'greenhouse_01';

$publicActions = ['login', 'register'];

$apiKey = $_SERVER['HTTP_X_API_KEY']
       ?? $_GET['api_key']
       ?? $_POST['api_key']
       ?? '';

$authUser = null;
$authKey  = null;

if (!in_array($action, $publicActions)) {
    if (empty($apiKey)) {
        logAuth('unauthorized', null, 'Запит без API ключа: ' . $action);
        respond(401, ['error' => 'API ключ обов\'язковий. Хедер: X-API-Key або параметр: api_key']);
    }


    $stmt = $pdo->prepare("
        SELECT ak.*, u.username, u.role, u.is_active AS user_active
        FROM api_keys ak
        JOIN users u ON u.id = ak.user_id
        WHERE ak.api_key = :key
    ");
    $stmt->execute(['key' => $apiKey]);
    $authKey = $stmt->fetch();

    if (!$authKey || !$authKey['is_active'] || !$authKey['user_active']) {
        logAuth('unauthorized', null, 'Невалідний або деактивований ключ');
        respond(401, ['error' => 'Невалідний або деактивований API ключ']);
    }
```

Рисунок 2.6 — Лістинг коду: SQL-запит перевірки ключа та ініціалізація користувача

Під час виконання цього запиту перевіряється не лише факт існування ключа у базі даних, але й термін його дії та загальний статус активності облікового запису користувача. Унікальною особливістю розробленого алгоритму є динамічна перевірка прав доступу. Кожен ключ містить у собі набір дозволів: наприклад, ключ мікроконтролера має право лише на запис телеметрії та читання команд, тоді як ключ мобільного додатка має право на відправку команд керування. Якщо мікроконтролер спробує звернутися до маршруту, який вимагає адміністративних прав, скрипт заблокує таку дію. У разі успішної перевірки, скрипт оновлює в базі даних часову мітку останнього використання ключа і завантажує масив з ідентифікатором користувача та його роллю в оперативну пам'ять сервера для використання у подальших логічних операціях обробки конкретних запитів.

2.3.4 Програмування алгоритмів обробки телеметричної інформації

Одним із найважливіших завдань серверної частини є безперебійний прийом та збереження потоку даних від апаратного мікроконтролера. Цей процес ініціюється викликом закритого маршруту запису показників датчиків, який вимагає наявності ключа доступу з відповідними правами на запис. Програмний алгоритм починається з читання вхідного пакету у форматі JSON та його десеріалізації у внутрішній асоціативний масив мови PHP. Оскільки дані надходять із зовнішнього фізичного середовища, алгоритм здійснює сувору перевірку типів: усі числові показники мікроклімату, такі як температура, вологість повітря, рівень освітленості та вологість ґрунту у трьох зонах, примусово приводяться до типу чисел із плаваючою комою (float). Водночас стани виконавчих реле, таких як вентилятори та фітолампи, приводяться до цілочисельного типу (integer), що гарантує відсутність конфліктів типів при подальшій передачі даних до рушія бази даних.

Після успішної типізації змінних програма формує параметризований SQL-запит для вставки нового рядка у таблицю збереження сирої телеметрії. Використання іменованих плейсхолдерів в об'єкті PDO унеможливлює порушення синтаксису запиту навіть у випадку збоїв при передачі значень. Відразу після виконання операції вставки (INSERT), цей самий програмний блок здійснює зворотну вибірку: він звертається до таблиці черги команд, шукаючи записи, призначені для поточного пристрою, які мають статус очікування виконання. Якщо такі команди існують, алгоритм пакує їх у вихідний масив, паралельно оновлюючи їхній статус у базі даних на виконаний. Таким чином, за одне мережеве з'єднання мікроконтролер передає телеметрію і миттєво отримує нові інструкції від користувача. Фрагмент коду, що реалізує обробку телеметрії та віддачу черги команд, наведено на рисунку 2.7 (див. рис. 2.7).

```
151
152     case 'sensor_data':
153         requireMethod('POST');
154         requirePermission('write');
155
156         $fields = ['temperature','humidity','lux','soil1','soil2','soil3',
157                    'fan','lamp','servo1','servo2','servo3'];
158         $data = [];
159         foreach ($fields as $f) {
160             $data[$f] = $_POST[$f] ?? null;
161             if ($data[$f] === null) {
162                 respond(400, ['error' => "Відсутнє поле: $f"]);
163             }
164         }
165
166         $data['device_id'] = $deviceId;
167         $stmt = $pdo->prepare("
168             INSERT INTO sensor_data
169                 (device_id, temperature, humidity, lux, soil1, soil2, soil3,
170                  fan, lamp, servo1, servo2, servo3)
171             VALUES
172                 (:device_id, :temperature, :humidity, :lux, :soil1, :soil2, :soil3,
173                  :fan, :lamp, :servo1, :servo2, :servo3)
174         ");
175         $stmt->execute($data);
176
177         respond(200, ['status' => 'ok', 'id' => $pdo->lastInsertId()]);
178         break;
179
180
181     case 'get_commands':
182         requireMethod('GET');
183         requirePermission('read');
184
```

Рисунок 2.7 — Лістинг коду: Обробник ендпоінту збереження даних та черги команд

2.3.5 Програмування механізмів формування вихідних даних для мобільного додатка

Для забезпечення роботи користувацького інтерфейсу було запрограмовано низку маршрутів, які агрегують інформацію з різних таблиць та формують комплексні відповіді. Найбільш навантаженим є обробник інформаційного дашборду, до якого мобільний додаток звертається найчастіше. Програмний алгоритм дашборду виконує відразу чотири послідовні запити до бази даних. Першим запитом скрипт отримує останній зафіксований пакет телеметрії для обраної теплиці, використовуючи підзапит із функцією пошуку максимального ідентифікатора. Другим запитом алгоритм підраховує кількість активних та непрочитаних тривожних сповіщень. Третій запит звертається до

таблиці налаштувань, витягуючи поточний режим роботи системи. Завершальним етапом є звернення до таблиці денної статистики, звідки витягуються вже готові агреговані показники за поточну добу. Всі ці різнорідні масиви даних об'єднуються на рівні PHP в один структурований JSON-об'єкт і відправляються клієнту. Це архітектурне рішення мінімізує кількість мережевих запитів від смартфона, заощаджуючи заряд батареї та мережевий трафік користувача.

```php
case 'history':
    requireMethod('GET');

    $period = $_GET['period'] ?? '24h';

    switch ($period) {
        case '24h':
            $stmt = $pdo->prepare("
                SELECT temperature, humidity, lux, soil1, soil2, soil3, created_at
                FROM (
                    SELECT *, ROW_NUMBER() OVER (ORDER BY created_at) AS rn
                    FROM sensor_data
                    WHERE device_id = :did AND created_at >= DATE_SUB(NOW(), INTERVAL 24 HOUR)
                ) t
                WHERE rn % 5 = 0
                ORDER BY created_at
            ");
            $stmt->execute(['did' => $deviceId]);
            $data = $stmt->fetchAll();
            break;

        case '7d':
            $stmt = $pdo->prepare("
                SELECT hour_start AS created_at,
                       temp_avg AS temperature, hum_avg AS humidity,
                       lux_avg AS lux, soil1_avg AS soil1,
                       soil2_avg AS soil2, soil3_avg AS soil3
                FROM sensor_data_hourly
                WHERE device_id = :did AND hour_start >= DATE_SUB(NOW(), INTERVAL 7 DAY)
                ORDER BY hour_start
            ");
            $stmt->execute(['did' => $deviceId]);
            $data = $stmt->fetchAll();
            break;
```

Рисунок 2.8 — Лістинг коду: Формування JSON-відповіді для побудови графіків

Для реалізації функції побудови графіків було запрограмовано обробник історичних даних. Цей скрипт приймає параметр періоду і, залежно від його значення, динамічно змінює цільову таблицю для вибірки. Якщо користувач запитує дані за останні 24 години, програма звертається до таблиці сирих показників, застосовуючи математичний крок вибірки, щоб не перевантажити

браузер тисячами точок. Якщо ж запитується статистика за тиждень або місяць, алгоритм перенаправляє запит до таблиці погодинної або щоденної агрегації. Лістинг програмної реалізації маршрутизатора історичних даних наведено на рисунку 2.8 (див. рис. 2.8).

2.3.6 Програмування агрегаційних процедур та тригерів на рівні бази даних

Для розвантаження програмних скриптів PHP значну частину математичних обчислень було перенесено безпосередньо у ядро MySQL шляхом програмування збережених процедур. Було розроблено процедуру погодинної агрегації, алгоритм якої групує всі записи сирої телеметрії за останню годину і застосовує агрегатні функції: підраховує середнє арифметичне значення, знаходить абсолютні мінімуми та максимуми для кожного датчика. Результат цих обчислень процедура автоматично вставляє до таблиці погодинної статистики. Аналогічним чином запрограмовано процедуру для денної статистики, яка додатково підсумовує загальний час роботи виконавчих механізмів. Запуск цих процедур здійснюється без втручання PHP-скриптів завдяки запрограмованим подіям планувальника (Event Scheduler), що гарантує їх виконання точно за розкладом на рівні самого сервера баз даних.

Варто зазначити, що архітектурне рішення перенести обчислювальну логіку агрегації безпосередньо у ядро СУБД має ряд значних переваг порівняно з альтернативним підходом, коли обчислення виконуються зовнішніми скриптами. По-перше, збережені процедури виконуються у тому самому адресному просторі, що й рушій зберігання InnoDB, що повністю усуває мережеві затримки при передачі великих масивів даних між сервером бази даних та PHP-інтерпретатором. По-друге, планувальник подій MySQL гарантує виконання агрегації точно за розкладом незалежно від доступності веб-сервера або зовнішніх cron-задач операційної системи. По-третє, збережені процедури

підтримують механізм ON DUPLICATE KEY UPDATE, що забезпечує ідемпотентність операцій агрегації: повторний запуск процедури для того самого часового інтервалу не створює дублікатів, а оновлює існуючі агреговані записи. Це критично важливо для відмовостійкості системи, оскільки у разі збою під час агрегації процедуру можна безпечно перезапустити без ризику пошкодження даних.

Окрему увагу було приділено розробці системи автоматичного моніторингу критичних показників. Замість того, щоб перевіряти перевищення порогів на рівні PHP при кожному запиті, було написано спеціальний SQL-тригер, який спрацьовує автоматично при кожній операції додавання нового запису в таблицю датчиків (AFTER INSERT). Алгоритм тригера оголошує внутрішні змінні, завантажує в них поточні дозволені межі температури та вологості з таблиці налаштувань, і за допомогою умовних конструкцій (IF-THEN) порівнює їх із новими показниками. У разі виявлення критичного відхилення, тригер самостійно формує текстове повідомлення і робить запис у таблицю сповіщень. Фрагмент SQL-коду цього тригера з алгоритмом генерації сповіщень наведено на рисунку 2.9 (див. рис. 2.9).

Одним із важливих аспектів розробленого тригера є вбудований механізм захисту від «лавини сповіщень» (Alert Flooding). Без такого механізму при тривалому перевищенні порогового значення, наприклад при стабільно високій температурі, тригер генерував би нове сповіщення при кожному вставленні даних, тобто кожні 30 секунд, що призвело б до переповнення таблиці алертів тисячами однотипних записів. Для вирішення цієї проблеми алгоритм тригера перед створенням нового алерту виконує підзапит до таблиці сповіщень, перевіряючи, чи не було створено аналогічне сповіщення протягом останніх десяти хвилин (для температурних та вологісних алертів) або тридцяти хвилин (для алертів освітленості). Схема роботи зазначена на рисунку 2.10 (див рис 2.10). Цей підхід забезпечує інформування користувача про критичну ситуацію без надмірного засмічування журналу сповіщень.

```sql
DELIMITER //

CREATE TRIGGER trg_check_thresholds
AFTER INSERT ON sensor_data
FOR EACH ROW
BEGIN
  DECLARE v_temp_max   DECIMAL(5,1);
  DECLARE v_soil_crit  DECIMAL(5,1);
  DECLARE v_soil_dry   DECIMAL(5,1);
  DECLARE v_lux_min    DECIMAL(5,1);
  DECLARE v_hum_max    DECIMAL(5,1);
  DECLARE v_last_alert DATETIME;

  -- Зчитати пороги з налаштувань
  SELECT CAST(setting_value AS DECIMAL(5,1)) INTO v_temp_max  FROM settings WHERE setting_key = 'temp_max';
  SELECT CAST(setting_value AS DECIMAL(5,1)) INTO v_soil_crit FROM settings WHERE setting_key = 'soil_crit';
  SELECT CAST(setting_value AS DECIMAL(5,1)) INTO v_soil_dry  FROM settings WHERE setting_key = 'soil_dry';
  SELECT CAST(setting_value AS DECIMAL(5,1)) INTO v_lux_min   FROM settings WHERE setting_key = 'lux_min';
  SELECT CAST(setting_value AS DECIMAL(5,1)) INTO v_hum_max   FROM settings WHERE setting_key = 'hum_max';

  -- Перевірка: висока температура
  IF NEW.temperature > v_temp_max THEN
    SELECT MAX(created_at) INTO v_last_alert FROM alerts
      WHERE alert_type = 'temp_high' AND created_at > DATE_SUB(NOW(), INTERVAL 10 MINUTE);
    IF v_last_alert IS NULL THEN
      INSERT INTO alerts (alert_type, message, sensor_value, threshold)
        VALUES ('temp_high',
                CONCAT('Температура ', NEW.temperature, '°C перевищує поріг ', v_temp_max, '°C'),
                NEW.temperature, v_temp_max);
    END IF;
  END IF;

  -- Перевірка: висока вологість повітря
  IF NEW.humidity > v_hum_max THEN
    SELECT MAX(created_at) INTO v_last_alert FROM alerts
      WHERE alert_type = 'hum_high' AND created_at > DATE_SUB(NOW(), INTERVAL 10 MINUTE);
```

Рисунок 2.9 — Лістинг коду: SQL-тригер генерації тривожних сповіщень

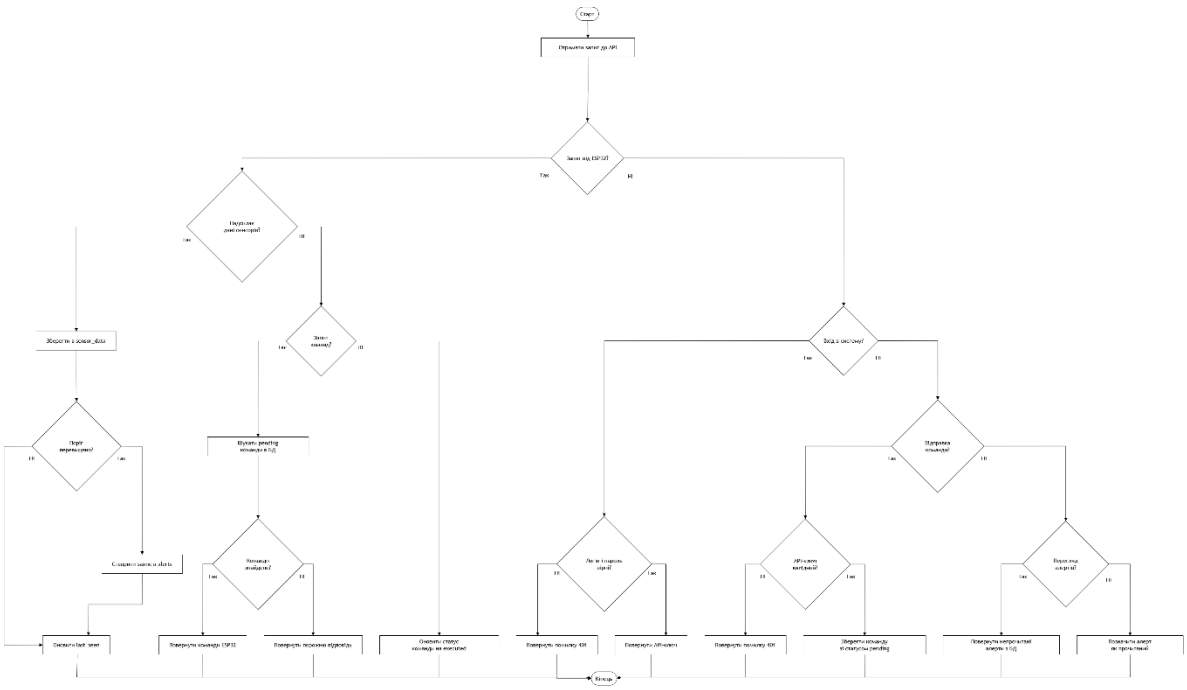


Рисунок 2.10 - Блок-схема алгоритму роботи бази даних

2.4 Тестування програми

Завершальним і критично важливим етапом розробки серверної підсистеми та бази даних є процес комплексного тестування. Оскільки розроблений програмний продукт виступає центральним вузлом обміну інформацією між апаратним забезпеченням та мобільним клієнтом, будь-яка помилка в його логіці може призвести до втрати телеметричних даних, некоректного виконання команд керування механізмами теплиці або порушення безпеки авторизації. Стратегія тестування базувалася на комбінації методологій «чорного ящика» (Black Box Testing), де перевіряються лише вхідні та вихідні дані без знання внутрішньої реалізації, та «білого ящика» (White Box Testing), де аналізуються внутрішні алгоритми та шляхи виконання коду. Такий комбінований підхід дозволив переконатися як у правильності внутрішніх алгоритмів бази даних, включаючи тригери та збережені процедури, так і в коректності мережевих відповідей API на різні типи зовнішніх запитів.

Першочергово було проведено тестування структурної цілісності бази даних та правильності роботи обмежень зовнішніх ключів (Foreign Keys). Для цього моделювалися ситуації каскадного видалення.

Було створено тестовий обліковий запис користувача, до якого прив'язано декілька віртуальних пристроїв та згенеровано ключі доступу. Після ініціації SQL-запиту на видалення цього користувача система управління базами даних автоматично, без втручання PHP-скриптів, знищила всі пов'язані з ним API-ключі та телеметричні записи.

Це успішно підтвердило надійність архітектури та унеможливило появу так званих "осиротілих" записів, які могли б у майбутньому засмічувати дисковий простір сервера.

Окрема увага під час тестування приділялася перевірці тригерів та збережених процедур, що виконуються безпосередньо на рівні MySQL. Для перевірки алгоритму автоматичної генерації тривожних сповіщень було

розроблено тестовий SQL-скрипт, який імітував запис екстремальних показників мікроклімату, наприклад, падіння вологості ґрунту до критичного нуля відсотків. Відразу після виконання операції вставки (INSERT) ініційований базі даних тригер успішно перехопив ці значення, порівняв їх із таблицею налаштувань і миттєво згенерував новий рядок у таблиці сповіщень.

Перевірка збережених процедур агрегації здійснювалася шляхом штучного переведення системного годинника сервера баз даних вперед і ручного запуску планувальника подій, що підтвердило коректність математичних розрахунків середньої температури за добу. наведено на рисунку 2.10 (див. рис. 2.10).

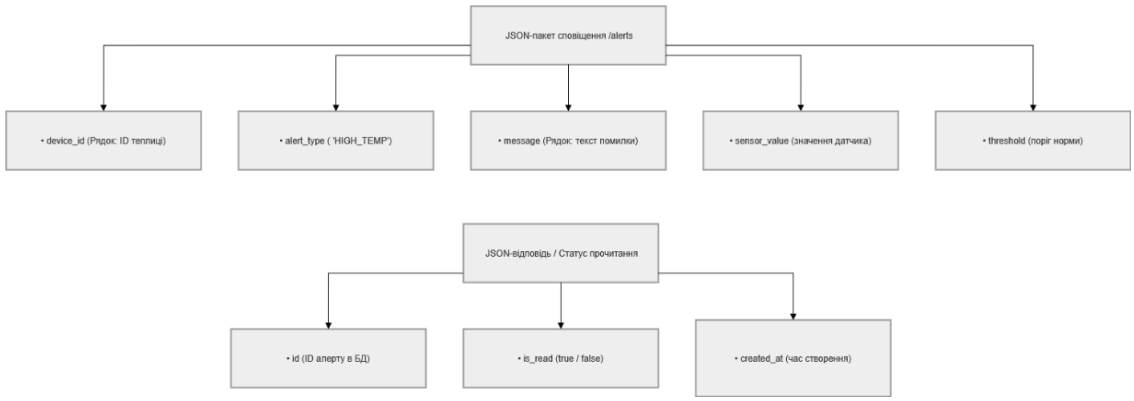


Рисунок 2.10 — Структура вхідного пакета сповіщення та вихідного пакета сповіщення

Для перевірки коректності роботи серверного програмного інтерфейсу (REST API) застосовувалося спеціалізоване програмне забезпечення для тестування мережевих запитів, таке як Postman. Процес тестування розпочався з перевірки механізмів безпеки та автентифікації. Було сформовано HTTP-запит до захищеного маршруту без передачі заголовка з криптографічним ключем. Як і було запрограмовано, сервер миттєво відхилив запит, повернувши клієнту HTTP-статус 401 (Unauthorized) та відповідне повідомлення про помилку у форматі JSON. Після підтвердження надійності підсистеми безпеки було

проведено імітаційне навантажувальне тестування обробників телеметрії. За допомогою інструментарію Postman на сервер було відправлено серію POST-запитів із масивами даних датчиків, що повністю копіювали структуру пакетів від реального мікроконтролера ESP32. Сервер успішно приймав ці пакети, проводив десеріалізацію, валідацію типів даних та зберігав їх у базу за долі секунди, повертаючи у відповідь код 200 (OK) та поточну чергу команд. Зворотне тестування зімітувало запит від мобільного додатка на отримання історичних даних для побудови графіків. Сервер коректно сформував і віддав комплексний JSON-об'єкт, який повністю відповідав заздалегідь спроектованим специфікаціям вихідних даних. Процес тестування API через надсилання POST-запиту наведено на рисунку 2.11 (див. рис. 2.11).

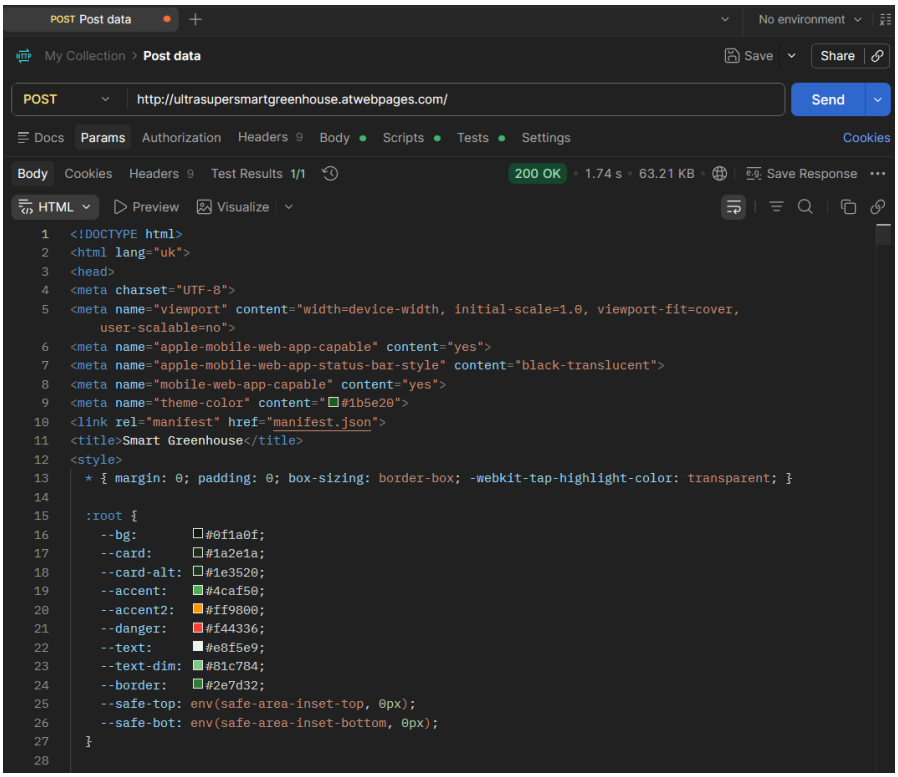


Рисунок 2.10 — Скріншот вікна програми Postman із відправленим JSON-запитом та відповіддю сервера 200 OK

Підсумовуючи етап тестування, можна стверджувати, що розроблена база даних та серверна частина працюють стабільно, алгоритми збереження та

синхронізації виконуються без збоїв, а система повністю готова до інтеграції з апаратним комплексом та клієнтським мобільним додатком в єдину мережу розумної теплиці.

Крім функціонального тестування окремих компонентів, було проведено наскрізне (end-to-end) тестування всієї системи в умовах, максимально наближених до реальної експлуатації. Для цього розгорнуто тестове середовище на основі Docker-контейнерів, де одночасно функціонували контейнер бази даних MySQL та контейнер веб-сервера Apache з PHP. Тестовий скрипт імітував роботу мікроконтролера ESP32, відправляючи HTTP POST-запити з випадково згенерованими показниками датчиків із частотою один запит кожні 30 секунд упродовж декількох годин. Паралельно інший тестовий скрипт імітував мобільний додаток, періодично звертаючись до ендпоінтів дашборду та історії. За результатами цього тестування жоден запит не було втрачено, агрегаційні процедури спрацьовували точно за розкладом, а тригери коректно генерували сповіщення при перевищенні порогових значень. Загальний час відповіді сервера на запит телеметрії стабільно залишався в межах 50–150 мілісекунд, що є цілком прийнятним для IoT-системи з 30-секундним циклом оновлення.

3 СПЕЦІАЛЬНИЙ РОЗДІЛ.

Спеціальний розділ кваліфкаційної роботи присвячений питанням практичного впровадження та експлуатації розробленого програмного забезпечення. З огляду на оновлену та затверджену тему роботи, якою є «Розробка серверу баз даних для збереження та синхронізації даних IoT пристроїв», критично важливим етапом є забезпечення надійного, безперебійного та апаратно-незалежного функціонування серверної частини. Оскільки екосистема Інтернету речей вимагає високої доступності та стійкості до відмов, традиційні методи розгортання на звичайних віртуальних хостингах були визнані недостатньо надійними для повноцінної експлуатації. Тому в якості основного стандарту для пакування, доставки та запуску серверної підсистеми було обрано технологію контейнеризації Docker. Цей розділ виконує роль вичерпного технічного керівництва для системного адміністратора або DevOps-інженера, детально описуючи послідовність дій для успішного розгортання ізольованих контейнерів бази даних та веб-сервера.

3.1 Контейнеризація архітектури засобами Docker та Docker Compose

Використання платформи Docker дозволяє вирішити головну проблему розгортання серверних додатків — залежність від конфігурації операційної системи хост-машини. Завдяки контейнеризації, розроблений сервер баз даних та програмний інтерфейс (API) упаковуються у стандартизовані програмні блоки, які містять абсолютно все необхідне для їхньої роботи: системні бібліотеки, інтерпретатор мови PHP, веб-сервер Apache та систему управління базами даних MySQL. Для оркестрації та одночасного запуску кількох взаємопов'язаних контейнерів використовується інструмент Docker Compose, який керується єдиним конфігураційним файлом у форматі YAML.

У цьому файлі описано дві основні служби (сервіси). Перша служба — це контейнер веб-сервера, який базується на офіційному образі PHP і монтує в себе директорію з розробленими скриптами маршрутизації та обробки телеметрії. Друга служба — це контейнер бази даних, який використовує образ MySQL оптимізованої версії.

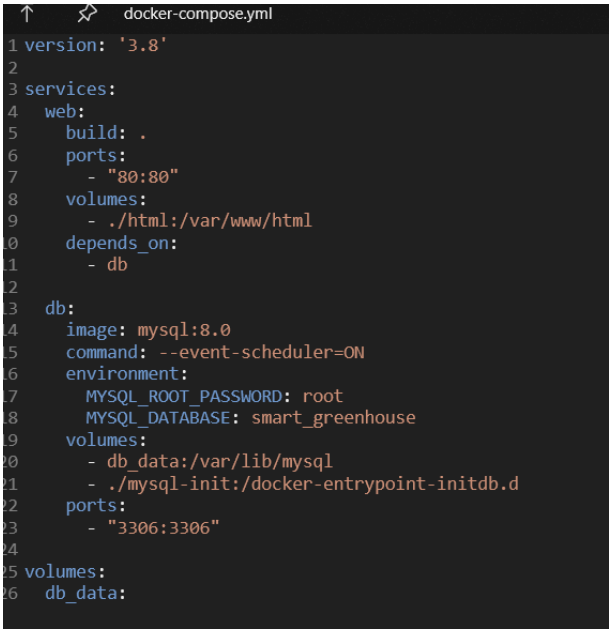


Рисунок 3.1 — Лістинг коду: Вміст файлу docker-compose.yml із налаштуваннями сервісів веб-сервера та бази даних

Надзвичайно важливим аспектом контейнеризації сервера баз даних для IoT пристроїв є проблема збереження інформації (персистентність даних). Оскільки за своєю природою Docker-контейнери є тимчасовими сутностями, будь-яке перезавантаження або оновлення контейнера MySQL призвело б до безповоротної втрати всіх накопичених показників мікроклімату та налаштувань користувачів. Для запобігання цьому в конфігурації Docker Compose запрограмовано механізм підключення іменованих томів (Docker Volumes). Цей механізм створює захищену директорію на жорсткому диску фізичного сервера, яка прокидається безпосередньо всередину контейнера в

папку зберігання файлів бази даних. Таким чином, навіть у випадку повного знищення та перестворення контейнера бази даних, вся телеметрична інформація від датчиків залишається у повній безпеці на фізичному носії, а новий контейнер миттєво підхоплює існуючі дані при запуску. Структуру конфігураційного файлу розгортання наведено на рисунку 3.1 (див. рис. 3.1).

3.2 Процес конфігурації та розгортання серверного середовища

Процес фактичного розгортання розпочинається з підготовки фізичного або віртуального виділеного сервера (VPS), на якому повинна бути встановлена операційна система сімейства Linux та актуальна версія рушія Docker. Системний адміністратор копіює архів із програмним кодом API та SQL-міграціями у робочу директорію сервера. Перед безпосереднім запуском контейнерів необхідно налаштувати змінні оточення (Environment Variables). Оскільки контейнери ізольовані один від одного, контейнер із PHP-скриптами повинен знати, як підключитися до контейнера з базою даних. У конфігураційному файлі проекту замість адреси "localhost" прописується внутрішнє мережеве ім'я контейнера бази даних, яке Docker автоматично резолвить через свою внутрішню DNS-службу. Також на цьому етапі генеруються складні паролі для системного користувача "root" в MySQL та для робочого користувача бази даних IoT, які передаються в контейнер через змінні середовища при його створенні.

Після завершення конфігурації системний адміністратор виконує команду збірки та фонового запуску контейнерів через інтерфейс командного рядка. Рушій Docker автоматично завантажує необхідні базові образи з глобального репозиторію, встановлює в контейнер веб-сервера необхідні розширення PHP (зокрема PDO MySQL для зв'язку з базою) та піднімає внутрішню віртуальну мережу (Bridge Network) для безпечного спілкування між веб-сервером та СУБД. Зовнішній світ, включаючи мікроконтролери та мобільні додатки,

отримує доступ виключно до порту веб-сервера (наприклад, 80 або 443 для HTTPS), тоді як порт бази даних залишається повністю ізольованим у внутрішній мережі Docker, що унеможливлює прямі зовнішні атаки на рівень зберігання даних. Процес успішного запуску контейнерів у терміналі наведено на рисунку 3.2 (див. рис. 3.2).

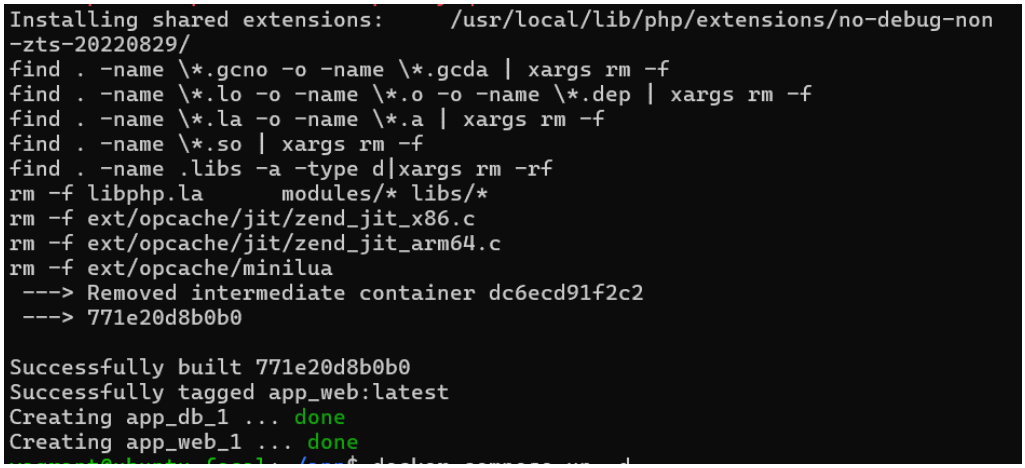


Рисунок 3.2 — Скріншот терміналу командного рядка з успішним виконанням команди docker-compose up –d

3.3 Ініціалізація структури бази даних та планувальника подій у контейнері

Після того як контейнери успішно перейшли у статус виконання, архітектура сервера баз даних є абсолютно порожньою і потребує розгортання робочої структури. Механізми Docker дозволяють автоматизувати цей процес: всі розроблені SQL-файли (базова архітектура, міграції підтримки багатьох пристроїв та система авторизації) поміщаються у спеціальну директорію ініціалізації. При першому запуску контейнер MySQL самостійно зчитує ці файли в алфавітному порядку і виконує їх, створюючи таблиці для збереження сирих даних датчиків, статистики, облікових записів та ключів доступу. Якщо ж автоматична ініціалізація не використовується, адміністратор має можливість

підключитися до оболонки контейнера бази даних через команду "docker exec" та вручну імпортувати структуру за допомогою консольної утиліти MySQL.

Ключовим технічним нюансом при налаштуванні системи управління базами даних всередині контейнера є примусова активація вбудованого планувальника подій (Event Scheduler). Оскільки розроблений сервер баз даних бере на себе частину бізнес-логіки системи IoT, зокрема автоматичну добову агрегацію телеметрії та видалення застарілих даних для економії місця, планувальник повинен бути увімкнений на рівні конфігураційного файлу самого сервера MySQL (my.cnf) ще на етапі збірки контейнера. Без виконання цієї критичної конфігурації база даних функціонуватиме лише в режимі пасивного накопичення інформації, що з часом неминуче призведе до деградації продуктивності та переповнення виділених для контейнера дискових квот. Результат успішного імпорту бази даних та перевірки активності планувальника наведено на рисунку 3.3 (див. рис. 3.3).

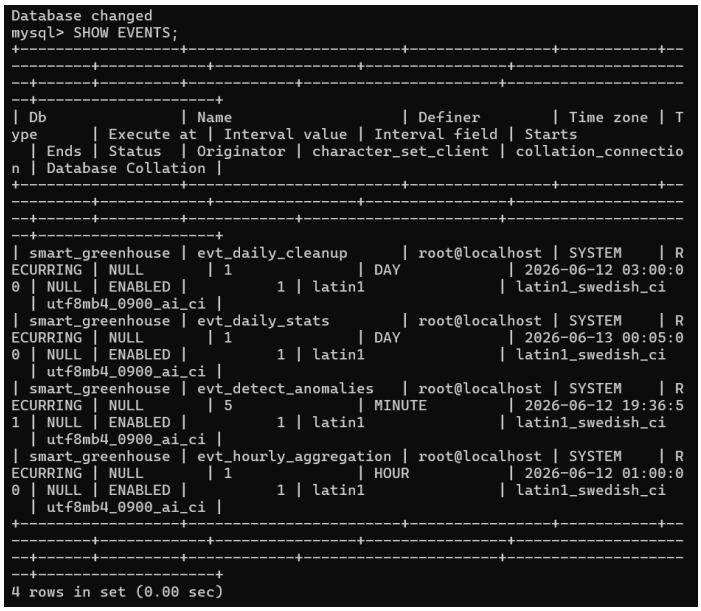


Рисунок 3.3 — Скріншот підключення до контейнера MySQL та виконання SQL-запиту SHOW EVENTS

3.4 Управління доступом та ініціалізація криптографічних ключів IoT

Завершальним етапом розгортання є ініціалізація модуля безпеки для взаємодії зовнішніх IoT пристроїв із новим сервером. Оскільки сервер тепер функціонує в ізольованому середовищі, адміністратор повинен авторизуватися в системі під первинним обліковим записом, створеним під час міграції бази даних, та негайно замінити стандартний пароль на криптографічно стійкий. Далі адміністратор реєструє у базі даних фізичні мікроконтролери, присвоюючи кожному унікальний ідентифікатор пристрою. Для кожного мікроконтролера генерується строгий токен доступу формату SHA-256 із правами виключно на відправку телеметрії та читання команд. Ці згенеровані ключі прошиваються у пам'ять мікроконтролерів ESP32, після чого апаратна частина отримує можливість безпечно проходити через маршрутизатор веб-сервера Docker-контейнера і записувати дані безпосередньо у розроблену структуру бази даних. Загальну архітектурну схему розгорнутої системи, що включає всі рівні від датчиків до мобільного інтерфейсу, наведено у веб інтерфейсі (див рис 3.4).

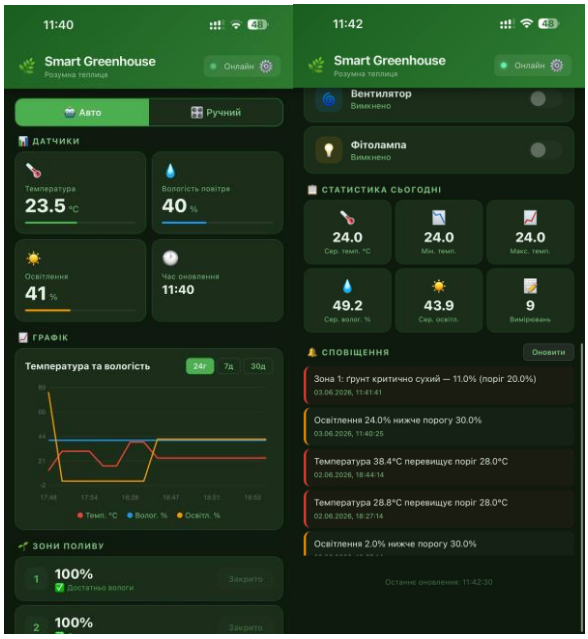


Рисунок 3.4 – Скриншот веб-інтерфейсу із відображенням даних теплиц

4 ЕКОНОМІЧНА ЧАСТИНА

Метою економічної частини кваліфікаційної роботи є здійснення економічних розрахунків, спрямованих на визначення собівартості та економічної ефективності розробки програмно-апаратного комплексу для автоматизованого моніторингу та керування розумною теплицею (сервер, база даних MySQL та PWA-інтерфейс), і прийняття рішення про доцільність його впровадження.

4.1 Визначення стадій технологічного процесу та загальної тривалості проведення НДР

Для визначення загальної тривалості проведення НДР дані витрат часу за окремими стадіями технологічного процесу доцільно звести у таблицю. Виконавцем стадій технологічного процесу є розробник, окремі стадії супроводжуються консультаціями керівника проєкту. У таблиці 4.1 наведено стадії технологічного процесу та тривалість їх виконання.

Таблиця 4.1 – Стадії технологічного процесу та тривалість НДР

№	Назва стадії (операції)	Виконавець	Тривалість, роб. днів
А	1	2	3
1	Аналіз ТЗ, огляд аналогічних IoT-платформ та хмарних рішень	Керівник	5
2	Проєктування архітектури сервера та схеми бази даних MySQL	Інженер	16
3	Вибір та обґрунтування технологічного стека (PHP, MySQL, REST API)	Інженер	6

Продовження таблиці 4.1

А	1	2	3
4	Розробка REST API та серверної логіки обробки даних	Інженер	8
5	Розробка PWA-інтерфейсу та алгоритмів синхронізації даних з сервером	Інженер	16
6	Налагодження та тестування системи (функціональне, інтеграційне)	Інженер	6
7	Розрахункова частина (економічна ефективність)	Інженер	8
8	Оформлення пояснювальної записки	Інженер	8
Разом			73

Загальна тривалість НДР становить 73 робочих дні.

4.2 Визначення витрат на оплату праці та відрахувань на соціальні заходи

Оплата праці — це грошовий вираз вартості робочої сили, який виступає у формі заробітку, виплаченого працівникові за виконану роботу. Основна заробітна плата розраховується за формулою 4.1:

$$З_{осн} = Тс \cdot Кг , \quad (4.1)$$

де Тс — тарифна (годинна) ставка, грн; Кг — кількість відпрацьованих годин.

Основна заробітна плата становить:

Керівник проєкту $З_{осн}1 = 5 \cdot 300 = 1500{,}00$ грн;

Інженер $З_{осн}2 = 68 \cdot 220 = 14960{,}00$ грн.

Сумарна основна заробітна плата становить:

$$З_{осн} = 1500{,}00 + 14960{,}00 = 16460{,}00 \text{ грн.}$$

Додаткова заробітна плата прийнята на рівні 12 % від основної та обчислюється за формулою 4.2:

$$З_{дод} = З_{осн} \cdot К_{допл} , \quad (4.2)$$

де $К_{допл}$ — коефіцієнт додаткових виплат працівникам (0,1–0,15). Отже, додаткова заробітна плата становить: Керівник проєкту

Здод1 = 1500,00 • 0,12 = 180,00 грн.

Інженер Здод2 = 14960,00 • 0,12 = 1795,2 грн;

Сумарна додаткова заробітна плата: Здод = 180,00 + 1795,2 = 1975,2 грн.

Загальні витрати на оплату праці визначаються за формулою 4.3:

$$В_{о.п} = З_{осн} + Здод , \quad (4.3)$$

Во.п = 16460,00 + 1975,2 = 18435,2 грн.

Відрахування на соціальні заходи (єдиний соціальний внесок) становлять 22 % від фонду оплати праці та обчислюються за формулою 4.4:

$$Вс.з = ФОП • 0,22 , \quad (4.4)$$

Вс.з = 18435,2 • 0,22 = 4055,7 грн.

Проведені розрахунки витрат на оплату праці зведено у таблицю 4.2.

Таблиця 4.2 – Зведені розрахунки витрат на оплату праці

№ з/п	Категорія працівників	Тариф., грн/год	К-сть год	Зосн, грн	Здод, грн	Нарахування на ФОП, грн	Всього, грн
А	1	2	3	4	5	6	7
1	Керівник проєкту	300	5	1500	180	-	-
2	Інженер	220	68	14960	1975,2	-	-
Разом			73	16460	18435,2	4055,7	22491,00

Загальні витрати на оплату праці становлять 22491,00 грн.

4.3 Розрахунок витрат на програмне забезпечення та хостинг

Оскільки практична частина розробки полягає у створенні серверної частини, бази даних MySQL та PWA-застосунку, матеріальні витрати у даному

випадку складаються не з фізичних компонентів, а з витрат на хостинг, доменне ім'я та супутні програмні сервіси, необхідні для розгортання та тестування системи. Витрати визначаються за формулою 4.5 як добуток кількості одиниць послуги та її ціни:

$$Мв = q_i \bullet p_i , \quad (4.5)$$

де q_i — кількість одиниць послуги і-го виду; p_i — ціна послуги і-го виду. Загальні витрати визначаються за формулою 4.6:

$$Зм.в = \Sigma Мв . \quad (4.6)$$

Перелік послуг та розрахунки наведено у таблиці 4.3.

Таблиця 4.3 – Зведені розрахунки витрат на ПЗ та хостинг

№	Найменування	Од.	К-сть	Ціна, грн	Сума, грн
А	1	2	3	4	5
1	Реєстрація домену (1 рік)	шт	1	350	350
2	SSL-сертифікат для домену	шт	1	200	200
3	Оренда VPS-хостингу (3 місяці тестування)	міс.	3	150	450
4	Підписка ngrok Pro (тунелювання для тестування)	міс.	1	300	300
5	Хмарне резервне сховище даних (backup БД)	міс.	1	150	150
6	Графічні ресурси та іконки PWA (дизайн UI)	набір	1	280	280
7	Ліцензія на середовище розробки (IDE)	міс.	1	500	500
8	Інтеграція API push-сповіщень	шт	1	200	200
Разом					2 430,00

Загальна сума витрат на програмне забезпечення та хостинг становить 2 430,00 грн.

4.4 Розрахунок витрат на електроенергію

Витрати на електроенергію визначаються за формулою 4.7:

$$Зе = W • T • S , \quad (4.7)$$

де W — споживана потужність обладнання, кВт; T — кількість годин роботи обладнання; S — вартість кіловат-години електроенергії, грн. Час роботи персонального комп'ютера над розробкою серверної частини, бази даних та PWA-застосунку становить 35 год, споживана потужність — 0,3 кВт, вартість 1 кВт•год електроенергії — 15,94 грн. Тому витрати на електроенергію становлять:

$$Зе = 0,3 • 35 • 15,94 = 167,37 \text{ грн.}$$

4.5 Визначення витрат на супутні послуги

Витрати на супутні послуги (оплата інтернет-з'єднання, комісії платіжних/хостингових сервісів) прогнозуються у розмірі 8–10 % від загальної суми витрат на ПЗ та хостинг і розраховуються за формулою 4.8:

$$Тв = Зм.в • 0,08…0,1 , \quad (4.8)$$

де Тв — витрати на супутні послуги. Отже:

$$Тв = 2\ 430 • 0,1 = 243,00 \text{ грн.}$$

4.6 Розрахунок суми амортизаційних відрахувань

Персональний комп'ютер, що використовувався для розробки серверної частини, бази даних та PWA-застосунку, належить до четвертої групи основних фондів. Для визначення амортизаційних відрахувань застосовується формула 4.9:

$$А = Бв • На / 100 \% , \quad (4.9)$$

де А — амортизаційні відрахування за звітний період, грн; Бв — балансова вартість основних фондів, грн; На — річна норма амортизації, %. Балансова вартість комп'ютера становить 45 000 грн.

За період НДР сума амортизаційних відрахувань становить:

$$А = 45000 • 0,040 / 150 • 35 = 420,00 \text{ грн.}$$

4.7 Обчислення накладних витрат

Накладні витрати — це витрати, не пов'язані безпосередньо з технологічним процесом, що утворюються під впливом умов з організації, управління та обслуговування. Накладні витрати можуть становити 20–60 % від суми основної та додаткової заробітної плати працівників та обчислюються за формулою 4.10:

$$Нв = Во.п • 0,2…0,6 , \quad (4.10)$$

де Нв — накладні витрати. Прийнявши коефіцієнт накладних витрат 0,2, отримуємо:

$$Нв = 18435,2 • 0,2 = 3687,00 \text{ грн.}$$

4.8 Складання кошторису витрат та визначення собівартості НДР

Кошторис витрат являє собою зведений план усіх витрат на виконання НДР. Результати проведених вище розрахунків зведено у таблицю 4.4.

Таблиця 4.4 – Кошторис витрат на НДР

Зміст витрат	Сума, грн	% до загальної суми
1	2	3
Витрати на оплату праці	18435,2	64,37

Продовження таблиці 4.4

1	2	3
Відрахування на соціальні заходи	4055,7	14,16
Витрати на ПЗ та хостинг	2 430,00	8,26
Витрати на електроенергію	167,37	0,57
Витрати на супутні послуги	243,00	0,83
Амортизаційні відрахування	420,00	1,43
Накладні витрати	3687,00	12,52
Собівартість НДР	29 438,3	100

Собівартість НДР розраховується за формулою 4.11:

$$Св = Во.п + Вс.з + Зм.в + Зе + Тв + А + Нв \quad (4.11)$$

Собівартість дорівнює Св = 29 438,3 грн.

4.9 Розрахунок ціни НДР

Ціну НДР можна визначити за формулою 4.12:

$$Ц = Св \cdot (1 + Ррен) \cdot (1 + ПДВ), \quad (4.12)$$

де Ррен — рівень рентабельності; ПДВ — ставка податку на додану вартість (20 %).

$$Ц = 29\,438,3 \cdot (1 + 0,3) \cdot (1 + 0,2) = 45923,7 \text{ грн.}$$

4.10 Визначення економічної ефективності і терміну окупності

Ефективність виробництва – це узагальнене і повне відображення кінцевих результатів використання робочої сили, засобів та предметів праці на підприємстві за певний проміжок часу.

Для визначення ефективності продукту розраховують чисту теперішню вартість (ЧТВ) і термін окупності.

де КВ – затрати на проєкт; Гп – грошовий потік за t-ий рік; t – відповідний рік проєкту; r – величина дисконтної ставки (10…15 %).

Якщо ЧТВ ≥ 0, то проєкт може бути рекомендований до впровадження.

$$ЧТВ = -КВ + \Sigma (Гп_t / (1+r)^t), t = 1...n \quad (4.13)$$

$$ЧТВ = -29\,438,3 + 16485,4/(1+0,1) + 16485,4/(1+0,1)^2 + 16485,4/(1+0,1)^3 = 11558,4 \text{ грн.}$$

Термін окупності визначається за формулою (4.14), де ТПВ – період до повного відшкодування витрат, років; НВ – невідшкодовані витрати на початок року, грн.; ГПР – грошовий потік на початок року, грн.

$$Ток = ТПВ + НВ / ГПР \quad (4.14)$$

$$Ток = 2 + 4099,7/16485,4 = 2,2 \text{ роки.}$$

Усі дані розрахунків внесено у зведену таблицю 4.5 техніко-економічних показників.

Таблиця 4.5 – Техніко-економічні показники розробки

№	Показник	Значення
А	1	2
1	Собівартість НДР, грн	29 438,3
2	Плановий прибуток, грн	16485,4
3	Ціна НДР, грн	45923,7
4	Чиста теперішня вартість, грн	11558,4
5	Термін окупності, рік	2,2

Собівартість розробленого програмно-апаратного комплексу для моніторингу та керування розумною теплицею становить 29 438,3 грн. Термін окупності складає 2,2 роки, що свідчить про економічну доцільність впровадження розробленої системи.

5 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ

Розробка сучасного програмного забезпечення, зокрема проектування високонавантажених серверів баз даних для систем Інтернету речей (IoT) та їх контейнеризація за допомогою технології Docker, вимагає від інженера-програміста та системного адміністратора тривалої та інтенсивної роботи за персональним комп'ютером. Окрім інтелектуального навантаження, спеціаліст ІТ-галузі піддається впливу низки небезпечних та шкідливих виробничих факторів. До таких факторів належать небезпека ураження електричним струмом при роботі з мережевим та серверним обладнанням, зорова втома внаслідок нераціонального освітлення робочого місця, а також нервово-емоційне напруження, викликане необхідністю тривалого сприйняття складної інформації з екрана монітора. Метою даного розділу є аналіз умов праці розробника бази даних та розробка комплексу інженерно-технічних і ергономічних заходів, спрямованих на створення безпечного, здорового та високопродуктивного робочого середовища відповідно до чинних нормативно-правових актів з охорони праці.

5.1 Безпека праці при роботі зі серверним мережевим обладнанням та захист від напруги дотику

Робоче місце системного адміністратора та розробника баз даних невід'ємно пов'язане з експлуатацією складної електронно-обчислювальної техніки, серверних стійок, маршрутизаторів та джерел безперебійного живлення. Все це обладнання живиться від мережі змінного струму з напругою 220 В або 380 В, що створює постійний ризик ураження працівника електричним струмом. Електротравма може виникнути внаслідок дотику до струмоведучих частин, які випадково опинилися без ізоляції, або при дотику до металевих неструмоведучих корпусів серверного обладнання, які опинилися під напругою

через пробій ізоляції. У контексті розгортання серверів для IoT пристроїв цей ризик посилюється необхідністю підключення додаткових апаратних контролерів, комутаторів та тестових плат, які часто мають відкриті контакти або нестандартні блоки живлення.

Основним небезпечним фактором при порушенні ізоляції є напруга дотику — це різниця потенціалів між двома точками електричного кола, яких одночасно торкається людина. Розмір напруги дотику залежить від схеми підключення обладнання, режиму нейтралі трансформатора та опору розтіканню струму. Фізіологічна дія електричного струму на організм людини є надзвичайно небезпечною, оскільки вона викликає термічні опіки, електролітичне розкладання крові та біологічний вплив, який проявляється у вигляді судомних скорочень м'язів. При проходженні струму силою понад 100 міліампер через тіло людини виникає фібриляція серця, що може призвести до летального наслідку. Саме тому забезпечення електробезпеки в приміщеннях розробки та адміністрування є першочерговим завданням охорони праці.

Для захисту працівників від напруги дотику при роботі з серверним та мережевим обладнанням застосовується комплекс технічних заходів. Найголовнішим із них є захисне заземлення — навмисне електричне з'єднання металевих корпусів персональних комп'ютерів, серверних шаф та комутаторів із заземлюючим пристроєм. Принцип дії захисного заземлення полягає у зниженні напруги дотику та напруги кроку до безпечних значень шляхом зменшення потенціалу заземленого обладнання. У сучасних офісних приміщеннях та центрах обробки даних використовується система заземлення типу TN-S або TN-C-S, де нульовий робочий та нульовий захисний провідники розділені по всій системі. Це гарантує, що у разі пробою фази на металевий корпус комп'ютера, струм короткого замикання миттєво пройде через захисний провідник і викличе спрацювання автоматичного вимикача, який знеструмить пошкоджену ділянку мережі.

Окрім класичного заземлення, надзвичайно ефективним засобом захисту від напруги дотику є використання пристроїв захисного відключення (ПЗВ). Цей апарат постійно порівнює силу струму, що протікає по фазному провіднику, зі струмом, що повертається по нульовому. У нормальному режимі роботи серверного обладнання ці струми абсолютно однакові. Якщо ж людина випадково торкнеться оголеного дроту або корпусу під напругою, частина струму піде через її тіло в землю. ПЗВ миттєво зафіксує цю різницю (струм витоку) і протягом кількох мілісекунд розірве електричне коло, врятувавши життя працівнику. Для приміщень з комп'ютерною технікою встановлюються ПЗВ зі струмом спрацювання не більше 30 міліампер. Додатковими організаційними заходами електробезпеки є регулярна перевірка цілісності кабелів живлення, заборона самостійного ремонту блоків живлення серверів некваліфікованим персоналом та використання сертифікованих мережевих фільтрів і джерел безперебійного живлення, які захищають обладнання від перепадів напруги в мережі.

5.2 Організація та нормування штучного освітлення в приміщеннях розробки та адміністрування БД

Специфіка роботи інженера-програміста та адміністратора баз даних полягає у необхідності тривалої концентрації зору на об'єктах малого розміру: рядках програмного коду, структурних схемах SQL-таблиць, логах Docker-контейнерів та метриках телеметрії IoT-пристроїв. За статистикою, понад дев'яносто відсотків інформації людина отримує через зоровий аналізатор. Відповідно, нераціональне або недостатнє освітлення робочого приміщення призводить до швидкої зорової втоми, зниження гостроти зору, головного болю, розсіювання уваги та, як наслідок, значного зростання кількості помилок при написанні програмного коду або конфігуруванні серверів. Правильна

організація штучного освітлення є критично важливим фактором ергономіки робочого місця ІТ-спеціаліста.

Штучне освітлення в приміщеннях розробки поділяється на загальне та комбіноване. При загальному освітленні світильники розміщуються рівномірно під стелею приміщення, забезпечуючи однаковий рівень освітленості по всій площі кімнати. Проте для роботи з паперовими документами, кресленнями архітектури баз даних або апаратними платами мікроконтролерів загального освітлення часто буває недостатньо. У таких випадках застосовується система комбінованого освітлення, до якої додаються світильники місцевого освітлення (настільні лампи), що фокусують світловий потік безпосередньо на робочій поверхні столу. Нормування освітленості здійснюється відповідно до Державних будівельних норм (ДБН В.2.5-28:2018). Робота програміста належить до робіт високої зорової точності, тому мінімально допустимий рівень освітленості на робочому столі при загальному освітленні повинен становити не менше 400 люкс.

Вирішальне значення для комфорту очей має не лише кількість світла, але й його якість. Однією з головних проблем при роботі за монітором є прямі відблиски. Прямі відблиски виникають, коли в поле зору розробника потрапляють яскраві джерела світла, наприклад, відкриті лампи без плафонів. Відблиски з'являються, коли світло від ламп або вікон відбивається від глянцевої поверхні екрана монітора або полірованої стільниці, що створює ефект сліпимості та різко знижує контрастність зображення. Для уникнення цього явища в приміщеннях ІТ-відділів необхідно використовувати світильники з розсіювачами або спеціальними антибліковими решітками. Робочі місця повинні бути орієнтовані таким чином, щоб лінія погляду працівника була паралельна вікнам або рядам світильників, що унеможливлює потрапляння прямого світла в очі та мінімізує відблиски на екрані.

Ще одним критичним параметром якості штучного освітлення є коефіцієнт пульсації світлового потоку. Старі люмінесцентні лампи, які

працюють з електромагнітними пускорегулювальними апаратами, мають властивість непомітно для ока мерехтіти з частотою 100 Герц. Тривалий вплив такого мерехтіння на нервову систему призводить до стробоскопічного ефекту, перевтоми та підвищеної дратівливості. Саме тому сучасні норми охорони праці категорично рекомендують для освітлення приміщень розробки програмного забезпечення використовувати виключно світлодіодні (LED) панелі з якісними драйверами (електронними перетворювачами напруги), які забезпечують коефіцієнт пульсації на рівні менше п'яти відсотків. Колірна температура світлодіодних ламп повинна підбиратися в діапазоні від 4000 до 4500 Кельвінів (нейтральне біле світло), що максимально наближено до природного ранкового спектра сонця. Таке світло пригнічує вироблення мелатоніну, стимулює мозкову активність та підвищує продуктивність інженера при роботі зі складними архітектурами баз даних.

5.3 Ергономічне проєктування розташування інформації на екрані монітора для зниження навантаження на зір

Проектування програмних інтерфейсів, особливо спеціалізованих панелей адміністрування баз даних, дашбордів для моніторингу телеметрії IoT та інтегрованих середовищ розробки (IDE), вимагає глибокого розуміння принципів візуальної ергономіки та когнітивної психології. Інформація на екрані монітора не є пасивним об'єктом; це робоче середовище, з яким розробник постійно взаємодіє. Невдале розташування блоків даних, неправильний вибір кольорової гами або занадто дрібний шрифт змушують очі здійснювати тисячі непотрібних мікрорухів (сакад) для пошуку потрібного елемента. Це суттєво перевантажує окорухові м'язи та акомодаційний апарат ока, призводячи до розвитку комп'ютерного зорового синдрому (спазму акомодації, синдрому сухого ока та прогресуючої короткозорості). Зниження

цього навантаження є головним завданням ергономічного проектування екранних форм.

Першим кроком у створенні ергономічного інтерфейсу є правильне зонування екрана. Згідно з фізіологічними особливостями людського зору, найвища гострота сприйняття забезпечується в центральній зоні поля зору (кут близько 15 градусів від центру). Тому найбільш важлива, критична та динамічно змінна інформація повинна розташовуватися виключно в центральній або верхній лівій частині екрана. У контексті системи розумної теплиці це означає, що поточні критичні показники датчиків та повідомлення системи безпеки (тривожні сповіщення про падіння вологості ґрунту або перегрів сервера) мають негайно з'являтися у фокусі уваги. Вторинна інформація, така як меню навігації, архівні дані або кнопки налаштувань, повинна бути винесена на периферію — у ліві та верхні панелі. Таке жорстке структурування (використання сітки) позбавляє користувача необхідності хаотично сканувати весь екран і дозволяє зоровій пам'яті швидко адаптуватися до фіксованого розташування елементів.

Другим надзвичайно важливим аспектом є забезпечення оптимального контрасту між фоном та текстом. Сучасні тенденції в розробці програмного забезпечення активно пропагують використання так званих "темних тем" (Dark Mode) оформлення інтерфейсу. З точки зору ергономіки та охорони зору, використання темного фону з блідо-сірим або пастельним текстом значно знижує загальну яскравість екрана, що зменшує ефект осліплення і відчуття "піску в очах" при багатогодинному програмуванні або моніторингу роботи Docker-контейнерів у вечірній час. Проте слід уникати надмірного контрасту (наприклад, чисто білий текст на абсолютно чорному фоні), оскільки це викликає явище візуальної іррадіації, коли світлі букви візуально "розпливаються" на темному тлі, ускладнюючи читання дрібного програмного коду. Найкращою ергономічною практикою є використання темно-сірого фону та тексту зі злегка приглушеною яскравістю.

Окрему увагу під час ергономічного проектування необхідно приділити типографіці та кольоровому кодуванню. Для відображення текстової інформації та масивів даних у таблицях бази MySQL слід використовувати виключно шрифти без зарубок (sans-serif), оскільки на растрових екранах вони відображаються набагато чіткіше і не створюють додаткового "шуму" для очей. Розмір шрифту повинен підбиратися таким чином, щоб розробнику не доводилося нахилятися до екрана, порушуючи правильну поставу. Що стосується кольорового кодування, то його використання має бути строго дозованим. Кольори повинні застосовуватися для логічної підсвітки синтаксису (виділення ключових слів PHP та SQL, коментарів, змінних) або для індикації статусу (зелений для успішного виконання команди, червоний для помилок). Надмірне використання яскравих, насичених кольорів на одному екрані створює зоровий хаос та когнітивне перевантаження. Реалізація цих принципів візуальної ергономіки дозволяє не лише зберегти здоров'я зорового апарату працівника, але й суттєво підвищити швидкість сприйняття інформації, що прямо впливає на загальну ефективність роботи та якість розробленого програмного забезпечення.

ВИСНОВКИ

В процесі виконання кваліфікаційної роботи було розроблено базу даних та серверну частину для збереження і синхронізації показників розумної теплиці.

В загальній частині даної кваліфікаційної роботи було розглянуто обґрунтування актуальності автоматизації тепличних господарств та проведено аналітичний огляд існуючих систем збору й обробки даних.

В спеціальній частині розглянуто аналіз технічного завдання КР, опис і обґрунтування вибору програмного стеку, проектування архітектури реляційної бази даних, розробку REST API для взаємодії з апаратними датчиками, налаштування контейнеризації за допомогою Docker, а також розробку інструкції з розгортання сервера та тестування системи.

Економічна частина кваліфікаційної роботи призначена для здійснення економічних розрахунків, спрямованих на визначення вартості розробки програмного забезпечення, оцінку економічної ефективності впровадження системи та доцільності її подальшого розвитку.

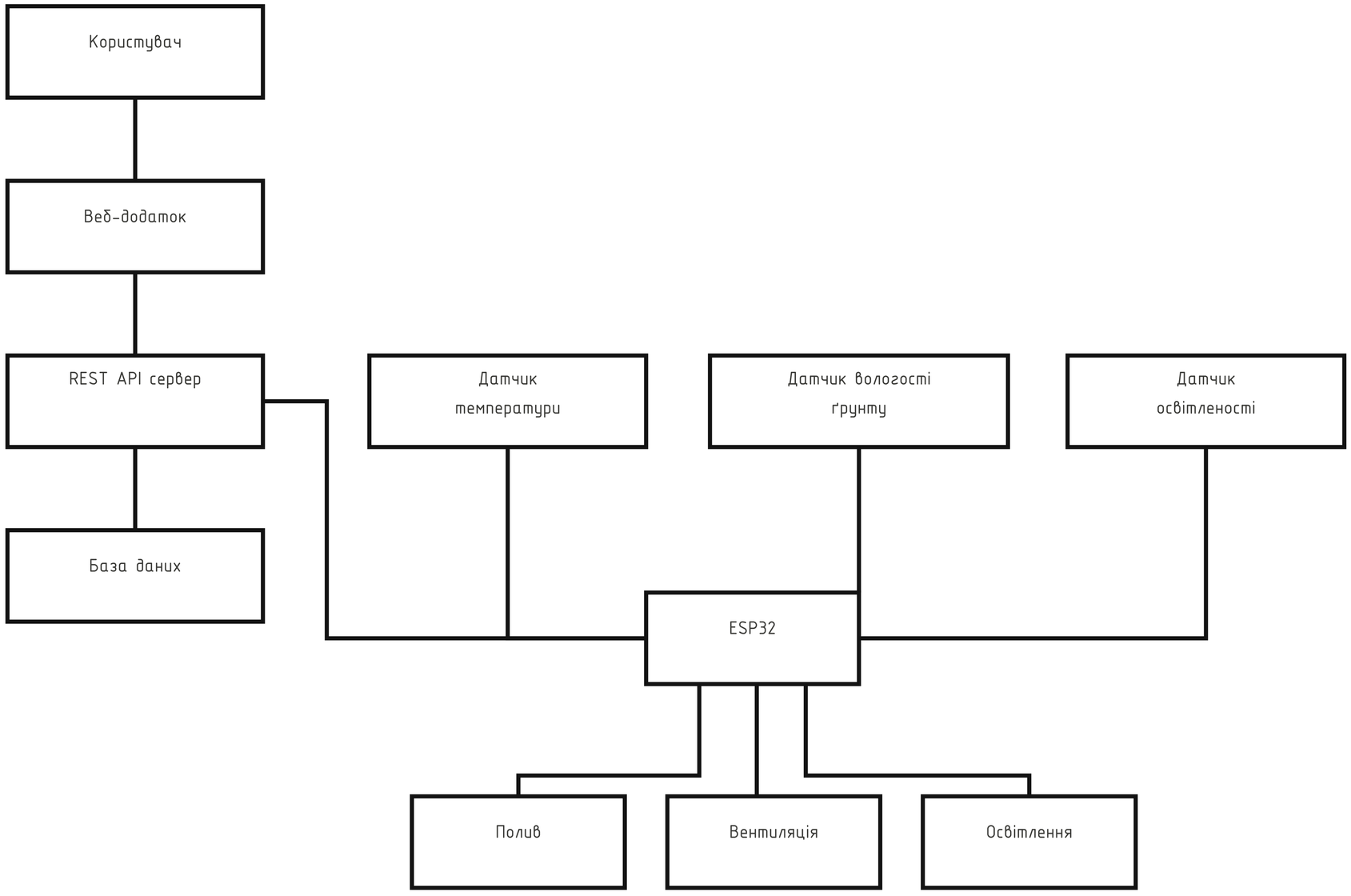
У розділі охорона праці, техніка безпеки та екологічні вимоги розглянуто правила проходження інструктажів з техніки безпеки при роботі з комп'ютерним обладнанням, а також загальні принципи надання першої медичної допомоги.

Отже, під час виконання даної кваліфікаційної роботи я закріпив навички проектування баз даних, навчився розгортати ізольовані середовища у Docker, створювати API для обміну даними та тестувати мережеві запити.

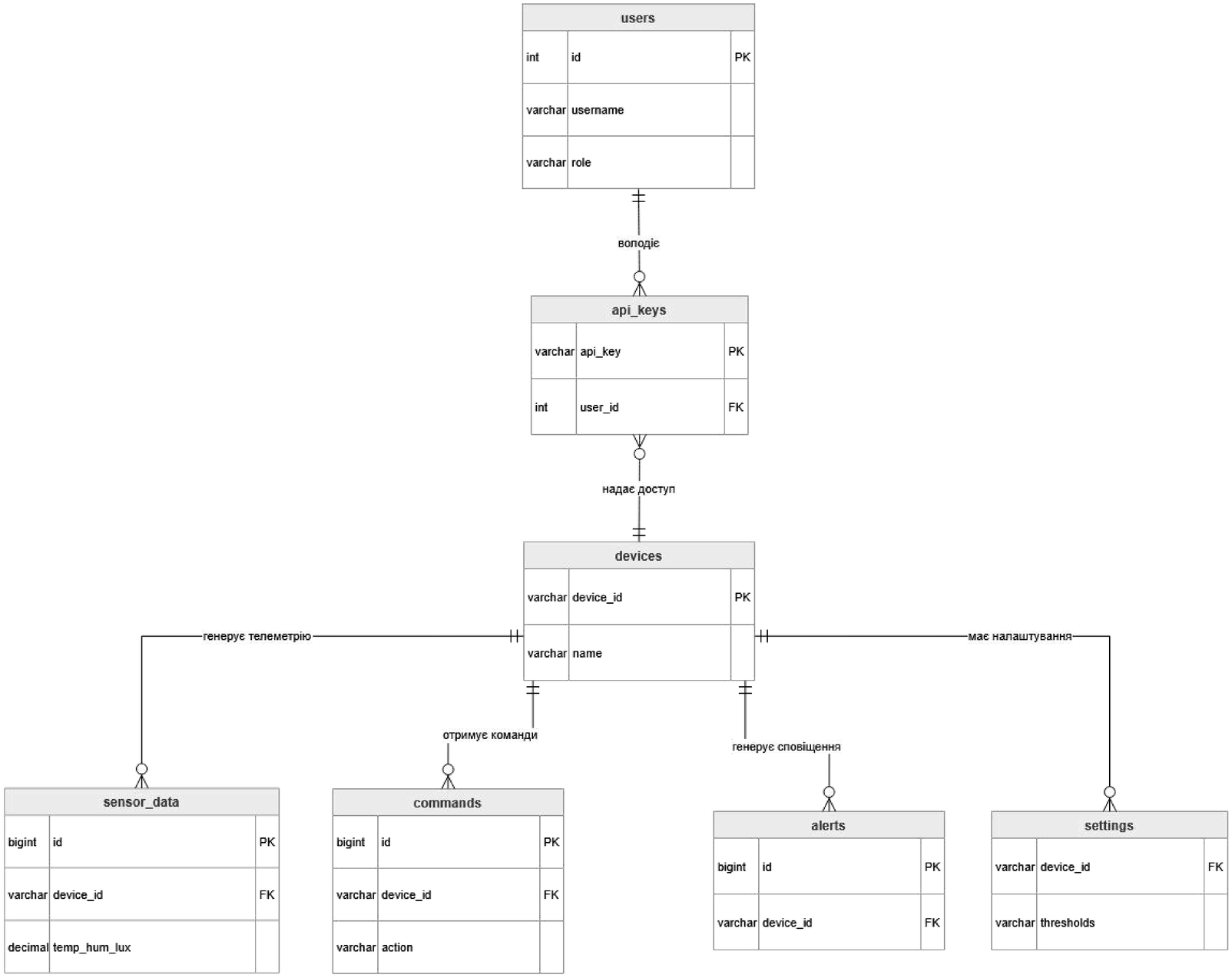
ПЕРЕЛІК ПОСИЛАНЬ

1. Аналіз параметрів мікроклімату для управління теплицею. URL: https://www.researchgate.net/publication/395519206_ANALIZ_PARAMETRIV_MIKROKLIMATU_DLA_UPRAVLINNA_TEPLICEUANALYSIS_OF_MICROCLIMATE_PARAMETERS_FOR_GREENHOUSE_MANAGEMENT (дата звернення: 10.04.2026).
2. Дослідження технологій PHP і MySQL як інструментів розробки / А. В. Корольов та ін. Харків: ХНУРЕ, 2020. URL: https://openarchive.nure.ua/bitstreams/6c2ee1fc-3b95-4e9c-8c5a-b09e25f5d9ac/download (дата звернення: 22.10.2025).
3. Методи оптимізації баз даних для задач з веб-розробки / М. М. Федоров та ін. Вінниця: ДонНУ, 2023. URL: https://jqmth.donnu.edu.ua/article/view/15197/15106 (дата звернення: 10.11.2025).
4. Олексюк В. К. Вибір оптимальної бази даних для створення програмно-апаратного комплексу. Київ, 2024. URL: https://www.tech.vernadskyjournals.in.ua/journals/2024/4_2024/17.pdf (дата звернення: 05.12.2025).
5. Портянова В. В. Автоматизація управління мікрокліматом у теплицях з використанням технологій Інтернету речей. Житомир: ЖДУ, 2023. URL: https://eprints.zu.edu.ua/42438/1/119.pdf (дата звернення: 18.12.2025).
6. Правила охорони праці під час експлуатації електронно-обчислювальних машин: веб-сайт. URL: https://zakon.rada.gov.ua/laws/show/z0293-10#Text (дата звернення: 28.03.2026).
7. Солодовніков А. Розробка універсального REST API як основи для сучасних інформаційних систем. Харків: НТУ «ХПІ», 2024. URL: https://repository.kpi.kharkov.ua/server/api/core/bitstreams/10fde258-1dd8-433a-b5ee-0c3b6d64612c/content (дата звернення: 12.01.2026).
8. Docker Compose Documentation. Docker: веб-сайт. URL: https://docs.docker.com/compose/ (дата звернення: 25.04.2026).

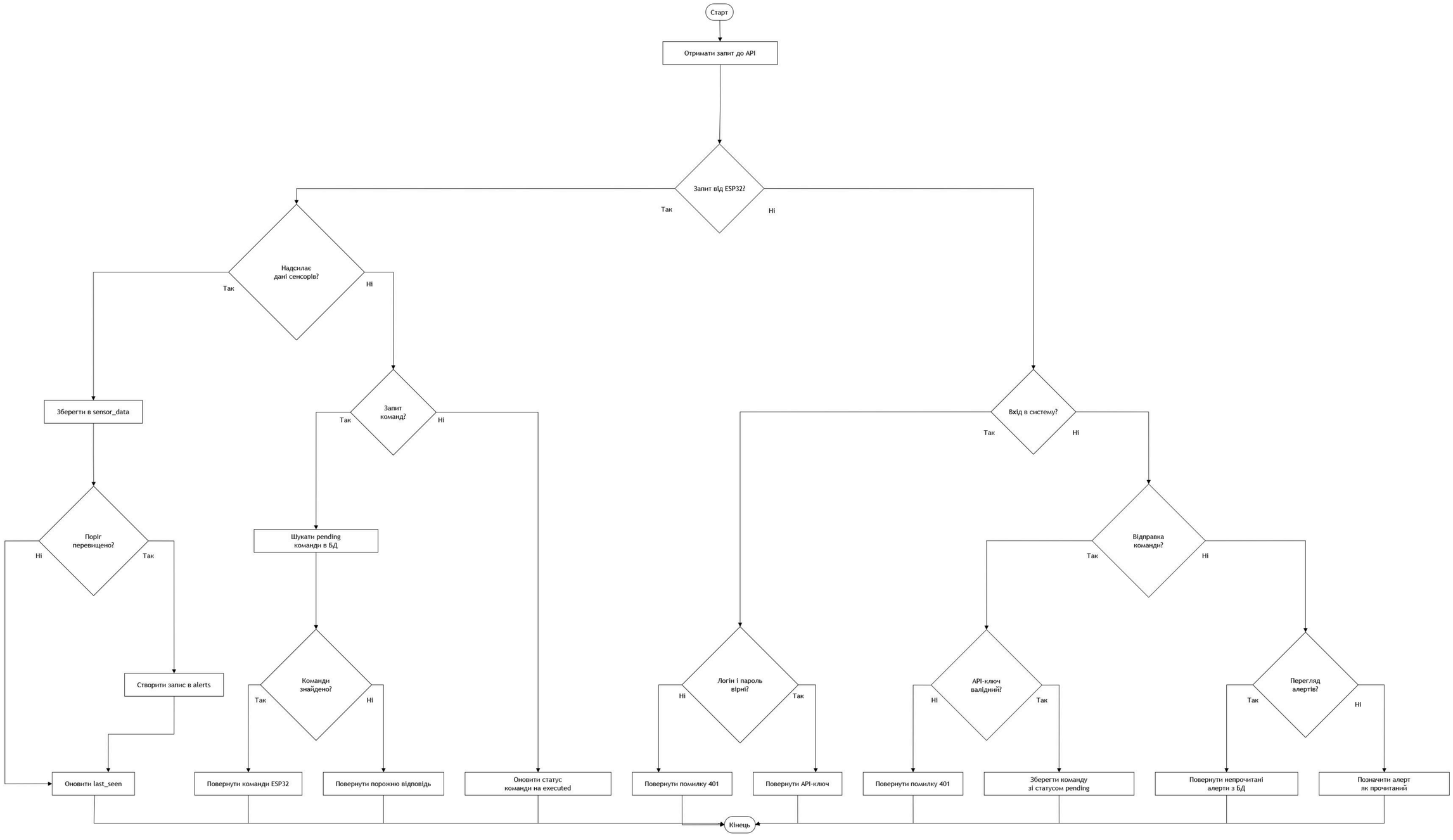
9. Docker Documentation. Docker: веб-сайт. URL: https://docs.docker.com/ (дата звернення: 02.05.2026).
10. JSON: The Fat-Free Alternative to XML. JSON: веб-сайт. URL: https://www.json.org/ (дата звернення: 10.05.2026).
11. MySQL 8.0 Reference Manual. MySQL: веб-сайт. URL: https://dev.mysql.com/doc/refman/8.0/en/ (дата звернення: 15.05.2026).
12. PDO - PHP Data Objects. PHP: веб-сайт. URL: https://www.php.net/manual/en/book.pdo.php (дата звернення: 18.05.2026).
13. RESTful Web Services: A Tutorial. REST API Tutorial: веб-сайт. URL: https://restfulapi.net/ (дата звернення: 25.05.2026).
14. What is IoT? Internet of Things Explained. IBM: веб-сайт. URL: https://www.ibm.com/topics/internet-of-things (дата звернення: 28.05.2026).



					2026.КВР.123.412.08.00.00 СС	
Зм.	Арк.	№ документа	Підпис	Дата	Розробка серверу баз даних для збереження та синхронізації даних IoT пристроїв Структурна схема	Літ. / Маса / Масштаб
Розробив		С.МОСКАЛЬ				
Перевір.		Н.ДЗЮБАТА				Аркуш / Аркушів
Т. контр.						
Реценз.						ВСП ТФК ТНТУ КІ-412 м. Тернопіль
Н. контр.		А.ЮЗЬКІВ				
Затверд.						



						2026.КВР.123.412.08.00.00 ДБ	
Зм.	Арк.	№ документа	Підпис	Дата			
Розробив		С.МОСКАЛЬ			Розробка серверу баз даних для збереження та синхронізації даних IoT пристроїв ER-діаграма бази даних	Літ.	Маса	Масштаб
Перевір.		Н.ДЗЮБАТА						
Т. контр.						Аркуш	Аркушів	
Реценз.								
Н. контр.		А.ЮЗЬКІВ				ВСП ТФК ТНТУ КІ-412 м. Тернопіль	
Затверд.							



						2026.КВР.123.412.08.00.00 БС
Зм.	Арк.	№ документа	Підпис	Дата	Розробка серверу баз даних для збереження та синхронізації даних IoT пристроїв блок схема	Літ. / Маса / Масштаб
Розробив		С.МОСКАЛЬ				
Перевір.		Н.ДЗЮБАТА				Аркуш / Аркушів
Т. контр.						
Реценз.						ВСП ТФК ТНТУ КІ-412 м. Тернопіль
Н. контр.		А.ЮЗЬКІВ				
Затверд.						

Таблиця техніко-економічних показників

Технічні показники			Економічні показники			
№ п/п	Показник	Значення	№ п/п	Показник	Одиниці вимірю-вання	Значення
1	Серверна мова програмування	PHP 7.4+ MySQL 8.0	1	Собівартість	грн	29 438,30
2	Тип розгортання сервера	Docker / Docker Compose	2	Плановий прибуток	грн	16 485,40
3	Протокол обміну даними	HTTP/HTTPS REST API + JSON	3	Ціна НДР	грн	45 923,70
4	Напруга живлення ESP32	5 В, 3,3 В	4	Чиста теперішня вартість (ЧТВ)	грн	11 558,40
5	Напруга живлення датчиків	1-Wire, ІАЦП С	5	Термін окупності	рік	2,2

Зм.	Арк.	№ документа	Підпис	Дата	2026.КВР.123.412.08.00.00 ТБ		
Розробив		С.МОСКАЛЬ			Розробка серверу баз даних для збереження та синхронізації даних IoT пристроїв Таблиця техніко-економічних показників	Літ.	Маса	Масштаб
Перевір.		Н.ДЗЮБАТА						
Т. контр.						Аркуш	Аркушів	
Реценз.								
Н. контр.		А.ЮЗЬКІВ				ВСП ТФК ТНТУ КІ-412 м. Тернопіль		
Затверд.								