Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка програмного забезпечення для моніторингу енергоспоживання комерційних будівель на базі трирівневої архітектури з використанням платформ .NET та Angular

Виконав(ла): студент(ка) IV курсу, групи СП-41
спеціальності 121 – Інженерія програмнго забезпечення
(шифр і назва спеціальності)

(підпис) Білінський М.Т. (прізвище та ініціали)

Керівник (підпис) Коноваленко І.В. (прізвище та ініціали)

Нормоконтроль (підпис) Стоянов Ю.М. (прізвище та ініціали)

Завідувач кафедри (підпис) Петрик М.Р (прізвище та ініціали)

Рецензент (підпис) (прізвище та ініціали)

Тернопіль 2026

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ проф. Петрик М.Р.
(підпис) (прізвище та ініціали)
« 6 » квітня 2026р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавр
(назва освітнього ступеня)

за спеціальністю 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

студенту Білінський Максим Тарасович
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка програмного забезпечення для моніторингу енергоспоживання комерційних будівель на базі трирівневої архітектури з використанням .NET та Angular

Керівник роботи Коноваленко Ігор Володимирович канд.техн.наук, доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від « *6* » *квітня* 2026 року № _____

2. Термін подання студентом завершеної роботи 22.06.2026

3. Вихідні дані до роботи Технічне завдання на розробку вебзастосунку для моніторингу енергоспоживання. Технологічний стек: ASP.NET Core 8, Angular, PostgreSQL 16… Архітектурний підхід. Інструменти тестування: xUnit, Moq. Середовище розробки: Visual Studio

4. Зміст роботи (перелік питань, які потрібно розробити)
Аналіз предметної області та визначення вимог до системи; Опис акторів, варіантів використання; Проектування бази даних та UML діаграм системи; Обґрунтування архітектурного підходу Clean Architecture та технологічного стеку; Реалізація серверної частини; Реалізація фонового сервісу симуляції та механізму виявлення аномалій; Реалізація клієнтської частини; Налаштування контейнерного розгортання; Тестування системи та аналіз результатів

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
Рисунок 1.1; Рисунок 1.2; Рисунок 1.3; Рисунок 1.4; Рисунок 1.5; Рисунок 1.6; Рисунок 3.1; Рисунок 3.2; Рисунок 3.3; Рисунок 3.4; Рисунок 3.5; Рисунок 3.6; Рисунок 3.7; Рисунок 3.8; Рисунок 3.9 ;Рисунок А.1; Рисунок А.2; Рисунок А.3.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці			
Нормоконтроль	Стоянов Ю.М		

7. Дата видачі завдання 6 квітня 2026 р.

КАЛЕНДАРНИЙ ПЛАН

з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної області, визначення вимог, опис акторів та use case	03.04.2026-09.04.2026	*Виконано*
2	Проектування доменної моделі, бази даних та UML діаграм	10.04.2026-20.04.2026	*Виконано*
3	Розробка базового каркасу серверної частини	21.04.2026-28.04.2026	*Виконано*
4	Реалізація механізму авторизації. Реалізація CRUD модулів	29.04.2026-04.05.2026	*Виконано*
5	Реалізація фонового сервісу симуляції та механізму алертів,та аналітики, дашборду	05.05.2026-11.05.2026	*Виконано*
6	Розробка клієнтської частини на Angular	12.05.2026-17.05.2026	*Виконано*
7	Налаштування Docker Compose та контейнерного розгортання	18.05.2026-23.05.2026	*Виконано*
8	Розробка та виконання тестів. Аналіз результатів тестування, виправлення виявлених недоліків	24.05.2026-30.05.2026	*Виконано*
9	Написання розділу з безпеки життєдіяльності та охорони праці	01.06.2026-06.06.2026	*Виконано*
10	Оформлення пояснювальної записки та підготовка до захисту	07.06.2026-12.06.2026	*Виконано*
11	Нормоконтроль		
12	Перевірка на академічний плагіат	08.06.2026-14.06.2026	
13	Попередній захист кваліфікаційної роботи бакалавра	15.06.2026 – 21.06.2026	
14	Захист кваліфікаційної роботи бакалавра		

Студент _____ Білінський М.Т
(підпис) (прізвище та ініціали)

Керівник роботи _____ Коноваленко І.В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Розробка програмного забезпечення для моніторингу енергоспоживання комерційних будівель на базі трирівневої архітектруи на базі .NET та Angular// Кваліфікаційна робота освітнього рівня «Бакалавр» // Білінський Максим // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра програмної інженерії // Тернопіль, 2026 // С. 75, рис. – 12, табл. – 20, част. – 4, додат. – 2, бібліогр. – 36.

Ключові слова: енергоспоживання; вебзастосунок; ASP.NET Core; Angular; PostgreSQL; Clean Architecture; JWT; симуляція; аналітика; алерти.

Кваліфікаційна робота присвячена розробці вебзастосунку Energy Monitoring для моніторингу енергоспоживання.

У першому розділі проведено аналіз вимог до системи, досліджено предметну область та визначено функціональні й нефункціональні вимоги. У другому розділі описано реалізацію вебзастосунку, обґрунтовано вибір технологій та архітектурних рішень. У третьому розділі проведено тестування системи та оцінено результати її роботи. У четвертому розділі розглянуто питання безпеки життєдіяльності під час експлуатації програмного забезпечення.

Вебзастосунок реалізовано з використанням Angular, ASP.NET Core, Entity Framework Core та PostgreSQL. Для авторизації використано JWT, а для контейнерного розгортання — Docker Compose.

Результатом роботи є вебзастосунок Energy Monitoring для обліку та аналізу енергоспоживання, виявлення аномалій і формування сповіщень. Система відповідає вимогам функціональності та безпеки.

Об'єкт дослідження: процес моніторингу та аналізу енергоспоживання в будівлях і окремих пристроях.

ABSTRACT

Development for monitoring energy consumption of commercial buildings based on a three-tier architecture based on .NET and Angular // Bachelor Qualification Thesis // Maksym Bilinskyi // Ternopil Ivan Puluj National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Software Engineering // Ternopil, 2026 // P. 75, Fig. – 12, Tab. – 20, Parts – 4, App. – 2, Ref. – 36.

Keywords: energy consumption; web application; ASP.NET Core; Angular; PostgreSQL; Clean Architecture; JWT; simulation; analytics; alerts.

The bachelor qualification thesis is devoted to the development of the Energy Monitoring web application for energy consumption monitoring.

The first chapter presents the analysis of system requirements, examines the subject area, and defines functional and non-functional requirements. The second chapter describes the implementation of the web application, including the selected technologies and architectural solutions. The third chapter covers system testing and evaluation of the obtained results. The fourth chapter addresses occupational health and safety issues related to software operation.

The web application is implemented using Angular, ASP.NET Core, Entity Framework Core, and PostgreSQL. JWT is used for authentication, while Docker Compose is applied for containerized deployment.

The result of the thesis is the Energy Monitoring web application for energy consumption accounting and analysis, anomaly detection, and alert generation. The system meets functional and security requirements.

Research object: the process of monitoring and analyzing energy consumption in buildings and individual devices.

ЗМІСТ

ВСТУП

Електроенергія стала одним із ресурсів, який потрібно не лише оплачувати, а й постійно контролювати. Для офісу, навчального корпусу, лабораторії або невеликого виробничого приміщення важливо розуміти, які пристрої споживають найбільше, у який час виникають піки навантаження і чи є в системі підозрілі відхилення. Простого місячного рахунку за електроенергію для цього недостатньо. Він показує результат, але не пояснює причину.

Ідея цієї роботи полягає у створенні програмної системи, яка наближає користувача до даних. Energy Monitoring дає змогу описати будівлі, пристрої та сенсори, після чого система починає формувати показники споживання. У межах бакалаврського проєкту фізичні лічильники не використовуються. Їх замінено віртуальними сенсорами і фоновим сервісом симуляції, який кожні 10 секунд генерує нові значення потужності, енергії, напруги, струму, коефіцієнта потужності та вартості.

Актуальність теми пояснюється кількома причинами. По-перше, сучасні будівлі мають багато електричних пристроїв, а ручний контроль стає незручним. По-друге, аналітика енергоспоживання допомагає знаходити неочевидні втрати. По-третє, швидке сповіщення про аномальне навантаження дає змогу реагувати раніше, ніж проблема стане фінансово або технічно відчутною. Для програмної інженерії така система цікава ще й тим, що поєднує предметну область, веброзробку, бази даних, фонові процеси, авторизацію і тестування.

Метою роботи є розроблення вебзастосунку Energy Monitoring для моніторингу енергоспоживання з підтримкою симуляції показників у реальному часі, аналітики, сповіщень та звітів. Система має бути зрозумілою для користувача, придатною для розгортання в контейнерах і побудованою так, щоб її можна було розширювати в майбутньому.

Для досягнення мети потрібно виконати такі завдання: проаналізувати предметну область, визначити вимоги до системи, описати акторів та варіанти

використання, спроєктувати доменну модель і базу даних, обрати архітектурний підхід, реалізувати backend, frontend і фонову симуляцію, підготувати механізм алертів, реалізувати аналітичні екрани, описати розгортання та перевірити працездатність системи.

Об’єктом дослідження є процес моніторингу та аналізу енергоспоживання в будівлях. Предметом дослідження є програмні засоби для збору, симуляції, збереження, обробки та візуалізації енергетичних показників у вебсистемі.

Практичне значення роботи полягає в тому, що створена система може бути використана як навчальний або демонстраційний прототип для подальшого підключення реальних сенсорів. Вона вже містить основні частини, які потрібні подібним рішенням: структуру об’єктів, історію показників, обчислення вартості, аналітику, алерти, ролі користувачів і контейнерне розгортання.

У роботі використано ASP.NET Core 8 для серверної частини, Angular для клієнтського інтерфейсу, PostgreSQL 16 для збереження даних, Entity Framework Core для доступу до бази даних, JWT для авторизації, Docker Compose для запуску декількох сервісів і xUnit з Moq для тестування окремих сервісів.

Структура роботи складається з трьох основних розділів. У першому розділі описано розробку програмної системи від аналізу вимог до використання. У другому розділі подано тестування. У третьому розділі розглянуто питання безпеки життєдіяльності та охорони праці для робочого місця розробника.

1 РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИ

Розробка сучасної вебсистеми моніторингу енергоспоживання вимагає комплексного підходу, що охоплює всі етапи програмної інженерії — від аналізу вимог і проєктування до реалізації та розгортання. Кожен із цих етапів є невід'ємною складовою процесу створення надійного програмного продукту, здатного задовольняти як поточні потреби користувачів, так і вимоги щодо подальшого масштабування системи.

У межах цього розділу послідовно розглядаються ключові аспекти розробки системи Energy Monitoring. Спочатку проводиться детальний аналіз предметної області та формулюються вимоги до системи, після чого виконується її проєктування із застосуванням загальноприйнятих практик об'єктно-орієнтованого аналізу. Проєктне рішення включає опис архітектурних підходів, побудову UML-діаграм та визначення структури бази даних, що в сукупності забезпечує цілісне уявлення про систему ще до початку реалізації.

Прийняті на цьому етапі рішення безпосередньо впливають на якість кінцевого програмного продукту. Правильно проаналізована предметна область і коректно сформульовані вимоги дозволяють уникнути суперечностей у процесі реалізації, а обґрунтований вибір архітектурних підходів забезпечує можливість розширення системи без необхідності її повного перепроєктування. Саме тому аналізу та проєктуванню приділяється особлива увага як фундаментальному підґрунтю всієї подальшої розробки.

1.1 Аналіз вимог

Перед початком розробки необхідно чітко визначити предметну область, сформулювати задачу, окреслити коло акторів та їхніх сценаріїв взаємодії, а також встановити словник термінів, єдиний для всіх рівнів системи. Такий підхід забезпечує узгодженість між аналізом, проєктуванням і реалізацією, знижує ризик

виникнення суперечностей у вимогах і полегшує подальший супровід системи. Аналіз охоплює як функціональні аспекти — тобто те, що система повинна робити, — так і нефункціональні вимоги щодо продуктивності, безпеки, надійності та масштабованості.

1.1.1 Аналіз предметної області

Предметна область моніторингу енергоспоживання охоплює комплекс задач, пов’язаних із обліком електричної енергії, аналізом режимів роботи обладнання, оцінкою вартості споживання та своєчасним виявленням відхилень від нормальних значень. У реальних умовах подібні системи інтегруються з різноманітними джерелами даних, зокрема розумними лічильниками, контролерами, промисловими сенсорами, інтелектуальними розетками та іншими IoT-пристроями. У межах навчального проєкту доцільно відокремити логіку обробки даних від фізичного обладнання, використовуючи програмну симуляцію. Такий підхід дозволяє дослідити архітектуру системи, структуру бази даних та аналітичні можливості без необхідності залучення реального апаратного середовища .

Основною структурною одиницею системи є будівля, яка виступає базовим контейнером для всіх інших об’єктів. Вона описується набором параметрів, що включає власника, назву, адресу, функціональний тип, площу, кількість поверхів, часові межі експлуатації та тариф на електроенергію. Внутрішня структура будівлі може деталізуватися за допомогою кімнат, що дозволяє більш точно визначати місце розташування обладнання. До будівель або окремих кімнат прив’язуються пристрої, які є безпосередніми споживачами енергії. Кожен пристрій характеризується технічними параметрами та пов’язується із сенсором, що відповідає за формування вимірювань.

Ключовим елементом аналітичної підсистеми є показник енергоспоживання. Він являє собою структурований запис, який містить часову мітку, значення потужності, обсяг спожитої енергії, параметри електричної мережі (напруга, струм, частота), коефіцієнт потужності, а також обчислену вартість і ознаку аномалії.

Особливістю таких даних є їх інтенсивне накопичення, що створює значне навантаження на систему зберігання. Навіть при відносно невеликій кількості пристроїв загальний обсяг записів швидко зростає, тому ефективність роботи системи значною мірою залежить від правильної організації запитів, індексації та механізмів агрегації даних .

Окрім збору та збереження даних, система повинна забезпечувати їх інтерпретацію. Для цього використовується механізм алертів, який дозволяє оперативно реагувати на потенційно небезпечні або нетипові ситуації. Алерти можуть формуватися у випадках перевищення допустимих значень, відсутності сигналу від сенсора або інших відхилень від очікуваної поведінки системи. У розробленому рішенні реалізовано базовий механізм виявлення аномалій, який ґрунтується на порівнянні потужності з номінальним значенням пристрою. Такий підхід забезпечує просту, але ефективну модель контролю стану системи .

З точки зору користувача, система повинна не лише накопичувати інформацію, а й надавати її у зручному для аналізу вигляді. Основна увага приділяється швидкому отриманню відповідей на практичні запитання, пов’язані з поточним станом споживання, витратами, навантаженням та наявністю проблемних ситуацій. Для цього реалізовано дашборд, сторінки аналітики, модуль сповіщень та інструменти формування звітів.

У процесі аналізу предметної області окремо виділено низку функціональних аспектів, кожен з яких має власні особливості реалізації.

Аспект обліку будівель передбачає створення та управління основними об’єктами системи. На відміну від простого введення даних, система повинна забезпечувати перевірку прав доступу, коректне збереження параметрів та формування зв’язків із підлеглими сутностями. Оскільки будівля є базовою одиницею, від її цілісності залежить коректність усієї структури даних.

Облік кімнат дозволяє деталізувати структуру будівлі та забезпечити більш точне представлення інформації. У цьому випадку важливо не лише зберігати дані,

але й контролювати їхню належність до конкретної будівлі, а також забезпечувати узгодженість при зміні або видаленні записів.

Облік пристроїв є більш складним, оскільки включає як зберігання характеристик обладнання, так і інтеграцію з іншими компонентами системи. Пристрій виступає джерелом даних, тому його параметри безпосередньо впливають на результати аналітики та виявлення аномалій.

Підключення сенсорів є критичним етапом, що забезпечує зв’язок між фізичною або віртуальною моделлю пристрою та потоком даних. Система повинна враховувати стан сенсора, періодичність передачі даних і можливі збої у його роботі.

Обробка потоку показників є однією з найбільш ресурсоємних частин системи. Вона включає генерацію або прийом даних, їх збереження, фільтрацію та підготовку до подальшого аналізу. Висока частота оновлення вимагає оптимізації алгоритмів та структури зберігання.

Аналіз аномальних значень спрямований на виявлення ситуацій, що виходять за межі нормального функціонування системи. Для цього використовуються порогові значення та додаткові перевірки, що дозволяють уникнути помилкових спрацювань.

Економічна оцінка споживання дозволяє перевести технічні показники у фінансову площину. Це дає змогу користувачу оцінити витрати та приймати обґрунтовані рішення щодо оптимізації використання енергії.

Історичний аналіз забезпечує можливість дослідження змін показників у часі. Він використовується для виявлення закономірностей, прогнозування та оцінки ефективності роботи системи.

Роль адміністратора передбачає розширені функції управління системою. Адміністратор має доступ до загальних даних, може керувати користувачами та контролювати стан системи в цілому, що вимагає додаткових механізмів безпеки.

Контейнерне розгортання є технічним аспектом, який забезпечує зручність запуску та масштабування системи. Воно дозволяє стандартизувати середовище виконання та спростити процес впровадження.

Спільною вимогою для всіх зазначених аспектів є забезпечення високої продуктивності. Система повинна мінімізувати обсяг переданих даних, використовувати агреговані запити та оптимізовані структури зберігання, що дозволяє забезпечити швидку роботу інтерфейсу та стабільну обробку великих обсягів інформації .

1.1.2 Постановка задачі

Потрібно розробити вебсистему, яка дозволяє користувачам керувати об’єктами енергомоніторингу та отримувати зрозумілу аналітику. Система має підтримувати реєстрацію, вхід, захищені запити, ролі User і Admin, CRUD операції для будівель, кімнат і пристроїв, підключення сенсорів, автоматичну генерацію показників, виявлення аномалій, перегляд алертів, аналітичний дашборд і формування звітів.

Серверна частина повинна бути реалізована як REST API. Це спрощує взаємодію з Angular клієнтом і залишає можливість додати інші клієнти в майбутньому. Усі запити, які змінюють або читають приватні дані, мають перевіряти JWT токен. Для адміністративних операцій потрібна додаткова перевірка ролі [2, 12].

Симуляція показників має працювати без участі користувача. Для цього використовується фоновий сервіс ASP.NET Core. Він періодично отримує активні підключені сенсори, розраховує потужність з урахуванням категорії пристрою, робочих годин будівлі і випадкового шуму, після чого зберігає записи в базу даних. Якщо виникає аномалія, сервіс формує алерт [4].

Клієнтський застосунок має бути зручним. Основні сторінки повинні відкриватися через захищені маршрути. Після входу користувач бачить дашборд. Звідти він переходить до будівель, пристроїв, аналітики, алертів і звітів. Для

адміністратора доступна сторінка керування користувачами та перегляду системної статистики [7, 8].

Таблиця 1.1 – Основні функціональні вимоги

Код	**Вимога**	**Пояснення**
FR1	Реєстрація і вхід	Користувач створює обліковий запис і отримує JWT токен після входу.
FR2	Керування будівлями	Користувач додає, редагує, видаляє і переглядає власні будівлі.
FR3	Керування кімнатами	Кімнати створюються в межах будівлі і використовуються для уточнення розміщення пристроїв.
FR4	Керування пристроями	Пристрої мають категорію, статус, пріоритет і номінальну потужність.
FR5	Підключення сенсорів	До пристрою підключається віртуальний сенсор з інтервалом вимірювання.
FR6	Симуляція показників	Фоновий сервіс кожні 10 секунд створює нові записи EnergyReading.
FR7	Алерти	Система створює сповіщення при аномальному споживанні.
FR8	Аналітика	Користувач бачить підсумки, топ пристроїв, пікові години і рекомендації.
FR9	Звіти	Користувач генерує звіт за вибраний період.
FR10	Адміністрування	Admin переглядає користувачів і системні показники.

Таблиця 1.2 – Нефункціональні вимоги

Категорія	**Вимога**
1	2
Продуктивність	API має повертати агреговані дані без передавання всієї історії вимірювань на клієнт.
Безпека	Доступ до приватних даних дозволений лише після перевірки JWT токена.
Масштабованість	Система має підтримувати збільшення кількості будівель, пристроїв і вимірювань.

Продвження таблиці 1.2

1	2
Надійність	Помилки API обробляються централізованим middleware, а фоновий сервіс продовжує роботу після винятків.
Супровід	Код поділений на шари Clean Architecture, що спрощує зміни і тестування.
Розгортання	Система запускається через Docker як набір frontend, api і postgres.

1.1.3 Актанти і варіанти використання

У системі визначено три основні актори: User, Admin і Simulation Service. User є звичайним користувачем, який працює зі своїми будівлями, кімнатами, пристроями, сенсорами, аналітикою, алертами і звітами. Admin має ширші права — він може переглядати всіх користувачів, змінювати їхній статус, переглядати системне зведення та глобальне споживання по всій системі. Слід зазначити, що Admin є розширенням ролі User і має доступ до всього функціоналу звичайного користувача. Simulation Service не є людиною, але виступає окремим актором, оскільки самостійно виконує дії в системі без участі користувача.

Варіанти використання відображають практичні цілі акторів. Для User це реєстрація, вхід у систему, перегляд профілю, повний цикл керування будівлями (перегляд, додавання, редагування, видалення), керування кімнатами та пристроями, у тому числі зміна статусу пристрою й підключення або відключення сенсора, перегляд дашборду, аналітика споживання з деталізацією за категоріями та топом пристроїв, перегляд і вирішення сповіщень, генерація та перегляд звітів. Для Admin це перегляд усіх користувачів, зміна їхнього статусу, перегляд системного зведення та глобального споживання. Для Simulation Service це автоматичне створення показників енергоспоживання та формування сповіщень у разі перевищення порогових значень.

Таке розділення акторів дозволяє уникнути плутанини в правах доступу. Звичайний користувач не може переглядати чужі будівлі або адміністративні дані. Адміністратор має доступ до системних показників, але не може напряму

змінювати історію вимірювань. Фоновий сервіс не взаємодіє з інтерфейсом, проте використовує ті самі сутності бази даних, що й інші актори.

Визначення акторів також відіграє важливу роль у формуванні моделі безпеки системи. Кожен актор взаємодіє із системою через чітко обмежений набір дозволених операцій, що реалізується на рівні серверної частини за допомогою перевірки JWT-токена та ролей. Це означає, що навіть за наявності технічного доступу до API жоден із акторів не може виконати дії, що виходять за межі його повноважень.

Таблиця 1.3 – Актори системи

Актор	**Опис**	**Основні дії**
User	Зареєстрований користувач системи	Керує будівлями, кімнатами, пристроями та сенсорами; переглядає аналітику, дашборд, алерти та звіти; генерує звіти.
Admin	Користувач з адміністративною роллю; розширює права User	Переглядає всіх користувачів, змінює їхній статус, переглядає системне зведення та глобальне споживання.
Simulation Service	Фоновий сервіс серверної частини	Автоматично генерує показники енергоспоживання, виявляє аномалії та створює алерти при перевищенні порогів.

1.1.4 Опис use case

Користувацькі сценарії (Use Case) описують основні варіанти взаємодії акторів із системою Energy Monitoring. Вони дозволяють формалізувати поведінку системи у типових ситуаціях та визначити послідовність виконання дій. Нижче наведено опис ключових сценаріїв, що відображають функціональне охоплення системи.

UC1. Реєстрація користувача. Основним актором є User. Користувач вводить персональні дані: ім'я, прізвище, електронну адресу та пароль. Система перевіряє

унікальність email, виконує хешування пароля та створює новий запис у базі даних. У разі успіху користувач отримує можливість перейти до авторизації.

UC2. Вхід у систему. Основним актором є User. Користувач вводить облікові дані. Сервер перевіряє їх відповідність записам у базі даних і в разі успіху генерує JWT-токен, який використовується для авторизації всіх наступних запитів до API.

UC3. Керування будівлями. Основним актором є User. Сценарій охоплює чотири операції: перегляд списку будівель, додавання нової будівлі із зазначенням назви, адреси, площі, типу, робочих годин і тарифу, редагування параметрів існуючої будівлі та її видалення. Перегляд будівлі включає («include») сценарій «Зведення по будівлі», який розширюється («extend») підрахунком пікових годин споживання.

UC4. Керування кімнатами. Основним актором є User. У межах існуючої будівлі користувач може додавати, редагувати та видаляти кімнати, які використовуються для уточнення розміщення пристроїв.

UC5. Керування пристроями. Основним актором є User. Користувач може додавати пристрої до будівлі або кімнати, задаючи категорію, статус, пріоритет і номінальну потужність, а також редагувати та видаляти їх. Сценарій включає («include») зміну статусу пристрою та підключення або відключення сенсора. Підключення сенсора, своєю чергою, розширюється («extend») автоматичним сповіщенням при перевищенні порогового значення.

UC6. Перегляд дашборду. Основним актором є User. Клієнтська частина виконує запити до сервера й відображає узагальнену інформацію: загальне споживання, витрати, кількість активних алертів, останні показники та рекомендації з оптимізації.

UC7. Аналітика споживання. Основним актором є User. Сценарій включає чотири підпорядковані варіанти використання: топ пристроїв за споживанням, споживання за категорією, рекомендації з економії та перегляд показників. Останній, своєю чергою, включає («include») агреговані дані та Live-моніторинг через SignalR.

UC8. Робота зі сповіщеннями. Основним актором є User. Користувач може переглядати список алертів, позначати їх як прочитані (сценарій включений) та вирішувати їх (сценарій є розширенням). Simulation Service розширює («extend») базовий сценарій перегляду сповіщень, автоматично створюючи нові алерти при перевищенні порогів.

UC9. Робота зі звітами. Основним актором є User. Користувач генерує звіт, вибираючи тип, будівлю та часовий інтервал. Система обчислює агреговані показники та зберігає результат. Сценарій перегляду звітів включає («include») завантаження звіту та розширюється («extend») його видаленням.

UC10. Обробка аномалії. Основним актором є Simulation Service. Під час генерації показників сервіс перевіряє відповідність значення потужності встановленому порогу. При перевищенні виконується перевірка cooldown, і якщо умова виконана — створюється новий алерт.

UC11. Керування користувачами. Основним актором є Admin. Адміністратор переглядає список усіх користувачів, може змінювати статус їхніх облікових записів (активувати або деактивувати), а також переглядати системне зведення та глобальне споживання по всій системі.

Таблиця 1.4 – Деталізація ключового use case підключення пристрою до моніторингу

Поле	**Значення**
Назва	Додавання пристрою і підключення сенсора
Основний актор	User
Передумови	Користувач авторизований, будівля вже створена.
Основний сценарій	Користувач створює пристрій, система зберігає його, користувач підключає сенсор, система активує моніторинг.
Альтернативи	Якщо будівля не належить користувачу, API повертає помилку доступу. Якщо сенсор уже існує, система оновлює стан підключення.
Постумови	Пристрій має активний сенсор.

1.1.5 Словник системи

Словник системи потрібний для того, щоб однаково використовувати терміни в аналізі, коді, інтерфейсі та документації. У Energy Monitoring частина назв залишена англійською, тому що вона безпосередньо відповідає класам і таблицям у коді. Наприклад, Building, Device, VirtualSensor і EnergyReading є одночасно термінами предметної області і назвами сутностей доменного шару [16].

У подальшому розвитку системи словник може бути розширений. Наприклад, якщо з’явиться підтримка реальних IoT пристроїв, треба буде додати поняття gateway, protocol, telemetry packet, calibration та device binding. У межах цієї роботи використано словник, достатній для реалізованого функціоналу.

Таблиця 1.5 – Словник системи

Термін	Значення
Building	Будівля або об’єкт, у межах якого ведеться моніторинг.
Room	Кімната в будівлі, яка уточнює місце розташування пристрою.
Device	Електричний пристрій або група обладнання з номінальною потужністю.
VirtualSensor	Віртуальний сенсор, який імітує лічильник або smart plug.
EnergyReading	Один запис вимірювання енергетичних параметрів.
Alert	Сповіщення про подію, яка потребує уваги користувача.
EnergyThreshold	Порогові значення, за якими система може оцінювати перевищення.
Report	Збережений результат формування звіту за період.
Dashboard	Головний екран з короткими показниками та графіками.
Simulation Service	Фоновий сервіс, який генерує показники без дій користувача.

1.2 Проектування системи

Проектування системи Energy Monitoring охоплює сукупність рішень, які визначають її структуру, поведінку та спосіб організації даних. На цьому етапі

формалізуються архітектурні підходи, обирається модель бази даних, будуються UML діаграми та визначається розподіл відповідальностей між компонентами. Прийняті рішення безпосередньо впливають на якість реалізації: правильно спроектована система є зрозумілою для розробника, стійкою до змін і придатною до розширення. Проектування виконується послідовно — від вибору процесу розробки та організації даних до детального опису архітектури і взаємодії між шарами застосунку.

1.2.1 Вибір процесу розробки

Для розробки системи Energy Monitoring доцільно застосувати ітеративний підхід, оскільки вона складається з великої кількості взаємопов'язаних компонентів, які складно реалізувати коректно в межах одного етапу. Такий підхід передбачає поетапне створення функціоналу з можливістю перевірки результатів після кожної ітерації. На початкових етапах формується доменна модель, що відображає структуру предметної області, після чого реалізується база даних і базові операції роботи з нею. Далі поступово додаються механізми авторизації, обробки даних, симуляції показників, аналітики, алертів і формування звітів. Завершальними етапами є тестування, оптимізація та підготовка системи до розгортання. Така послідовність дозволяє контролювати якість реалізації на кожному кроці та своєчасно виявляти помилки [16].

Ітеративний підхід є особливо ефективним у контексті навчального проєкту, оскільки дає змогу отримувати працездатні результати вже на ранніх стадіях розробки. Це дозволяє не лише перевірити коректність технічних рішень, а й оцінити зручність використання системи. Наприклад, після реалізації модулів будівель і пристроїв можна перевірити роботу API та клієнтських форм, що забезпечують взаємодію користувача із системою. Додавання сенсорів відкриває можливість запуску симуляції, а поява перших даних дозволяє перейти до побудови аналітики. Таким чином, кожен наступний етап базується на вже перевіреній функціональності, що зменшує ризик накопичення помилок.

Процес розробки умовно поділено на кілька послідовних ітерацій. Першою є етап аналізу вимог, на якому визначаються основні функціональні можливості системи та формуються сценарії її використання. Далі виконується проєктування доменної моделі, що включає визначення сутностей, їх атрибутів і зв’язків. Наступним кроком є створення базового каркаса серверної частини, включаючи налаштування архітектури, підключення бази даних і реалізацію базових компонентів. Після цього додається механізм авторизації, що забезпечує ідентифікацію користувачів і контроль доступу.

Подальші ітерації включають реалізацію CRUD-функціоналу для основних сутностей, розробку клієнтського інтерфейсу на Angular, інтеграцію симуляції даних, створення системи алертів, а також реалізацію аналітики та звітів. Завершальні етапи передбачають контейнеризацію системи, її тестування в різних сценаріях використання та підготовку супровідної документації.

Кожен із зазначених етапів має власні критерії завершення, які базуються не на кількості написаного коду, а на фактичній працездатності реалізованого функціоналу. На етапі аналізу вимог таким критерієм є сформований і перевірений перелік функціональних можливостей, що відповідають поставленій задачі. Під час проєктування моделі важливо забезпечити коректність структури даних і узгодженість зв’язків між сутностями.

На етапі розробки серверної частини ключовим показником є стабільна робота API, яка включає коректну обробку запитів, повернення очікуваних результатів і обробку помилкових ситуацій. Для клієнтського інтерфейсу важливою є не лише правильність відображення даних, а й зручність взаємодії користувача із системою.

Під час інтеграції симуляції перевіряється коректність генерації даних і їх відповідність очікуваним сценаріям роботи пристроїв. На етапі тестування оцінюється стабільність роботи всієї системи, включаючи взаємодію між компонентами, обробку помилок і збереження цілісності даних.

Таким чином, використання ітеративного підходу дозволяє не лише організувати процес розробки, а й забезпечити контроль якості на кожному етапі, що є особливо важливим для складних багатокомпонентних систем.

1.2.2 База даних

База даних є однією з найважливіших частин системи. Вона зберігає не лише довідникові дані, такі як користувачі, будівлі, кімнати і пристрої, а й велику історію вимірювань. Для Energy Monitoring обрано PostgreSQL, оскільки ця СУБД добре підходить для надійного збереження структурованих даних, підтримує транзакції, індекси, зв’язки, типи дати і часу, числові типи та стабільно працює в Docker контейнері [9].

Модель бази даних містить десять основних таблиць: Users, Buildings, Rooms, Devices, VirtualSensors, EnergyReadings, Alerts, EnergyThresholds, Reports і AlertCooldowns. Усі основні сутності мають ідентифікатор Guid. Зв’язки реалізовано через зовнішні ключі. Наприклад, Building має UserId, Room має BuildingId, Device має BuildingId і необов’язковий RoomId, VirtualSensor має DeviceId, а EnergyReading містить SensorId, DeviceId і BuildingId.

Окремо варто пояснити таблицю EnergyReadings. Вона навмисно дублює деякі зв’язки, зокрема DeviceId і BuildingId, хоча ці значення можна отримати через Sensor. Це зроблено для спрощення аналітичних запитів. Коли потрібно вибрати показники за будівлею або пристроєм, API може фільтрувати записи напряму, не виконуючи зайві з’єднання таблиць.

Таблиця AlertCooldowns використовується для захисту від надмірної кількості однакових сповіщень. Якщо пристрій короткий час багато разів генерує аномальний стрибок, користувач не повинен отримувати десятки однакових алертів. Cooldown зберігає момент останнього створення сповіщення певного типу для пристрою або будівлі.

Таблиця 1.6 – Основні таблиці бази даних

Таблиця	Призначення
Users	Облікові записи користувачів і ролі.
Buildings	Будівлі користувачів з параметрами площі, адреси, ціни електроенергії та робочих годин.
Rooms	Кімнати всередині будівель.
Devices	Електричні пристрої, які споживають енергію.
VirtualSensors	Віртуальні сенсори, підключені до пристроїв.
EnergyReadings	Історія показників енергоспоживання.
Alerts	Сповіщення про аномалії або інші події.
EnergyThresholds	Порогові значення для контролю споживання.
Reports	Згенеровані звіти.
AlertCooldowns	Обмеження частоти повторних алертів.

Для таблиці EnergyReadings бажано створювати індекси за Timestamp, BuildingId і DeviceId. Це прискорює вибірки за періодами, які найчастіше використовуються на дашборді та сторінці аналітики. Також доречно обмежувати період вибірки на API рівні, щоб користувач випадково не запросив усю історію[19].

У PostgreSQL числові значення потужності та енергії зберігаються як типи, що дають достатню точність для аналітики. Вартість зберігається як decimal або numeric, тому що гроші небажано обчислювати через типи з плаваючою точкою без контролю округлення. Дати зберігаються у форматі UTC, що спрощує роботу з часовими зонами [9].

1.2.3 UML діаграми

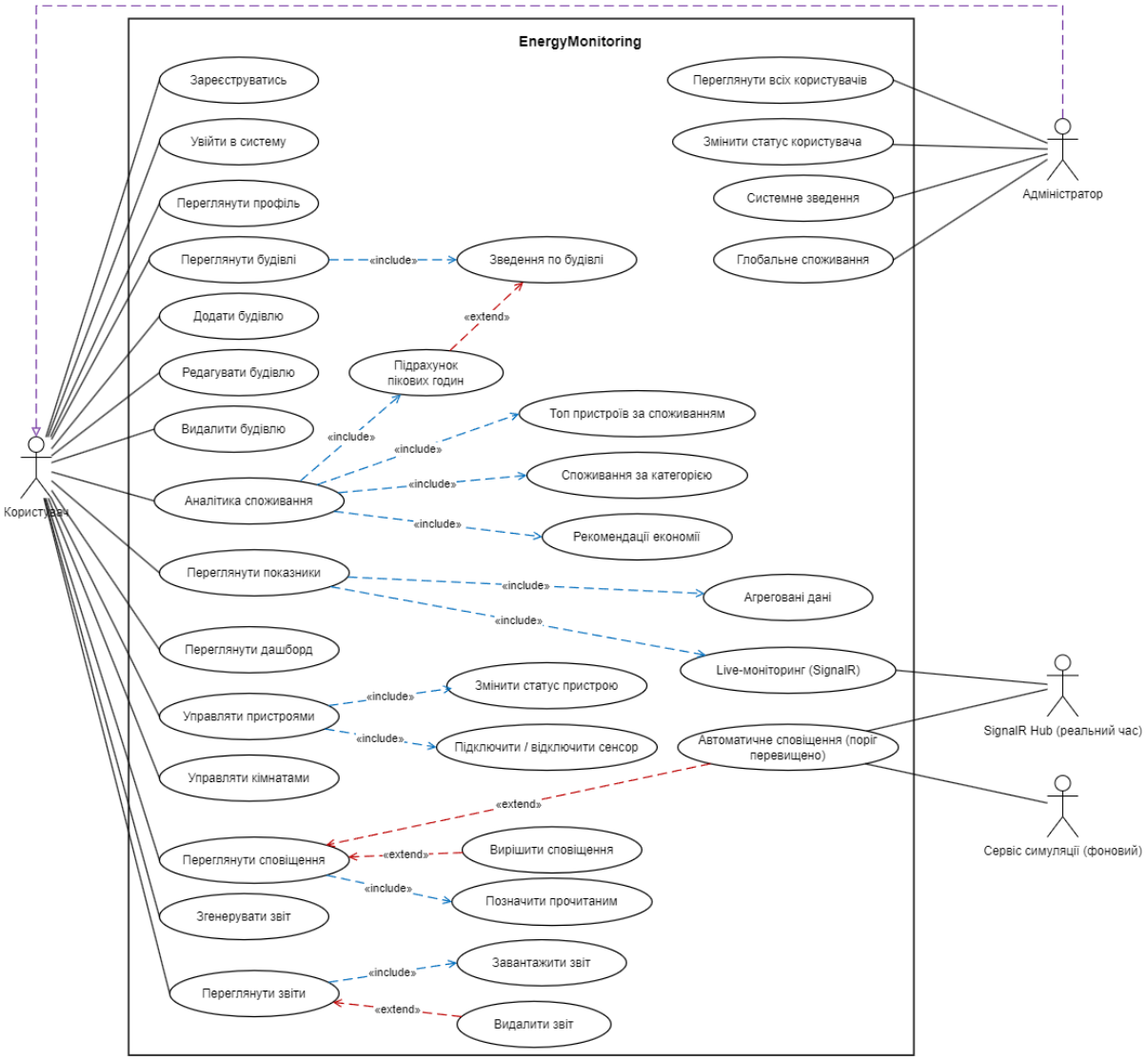

Рисунок 1.1 – Діаграма варіантів використання системи Energy Monitoring

Use Case діаграма показує, які можливості доступні акторам. User працює з основними сутностями системи, Admin керує користувачами і переглядає системні показники, а Simulation Service автоматично генерує показання та перевіряє аномалії. Межа системи відділяє зовнішніх акторів від функцій вебзастосунку.

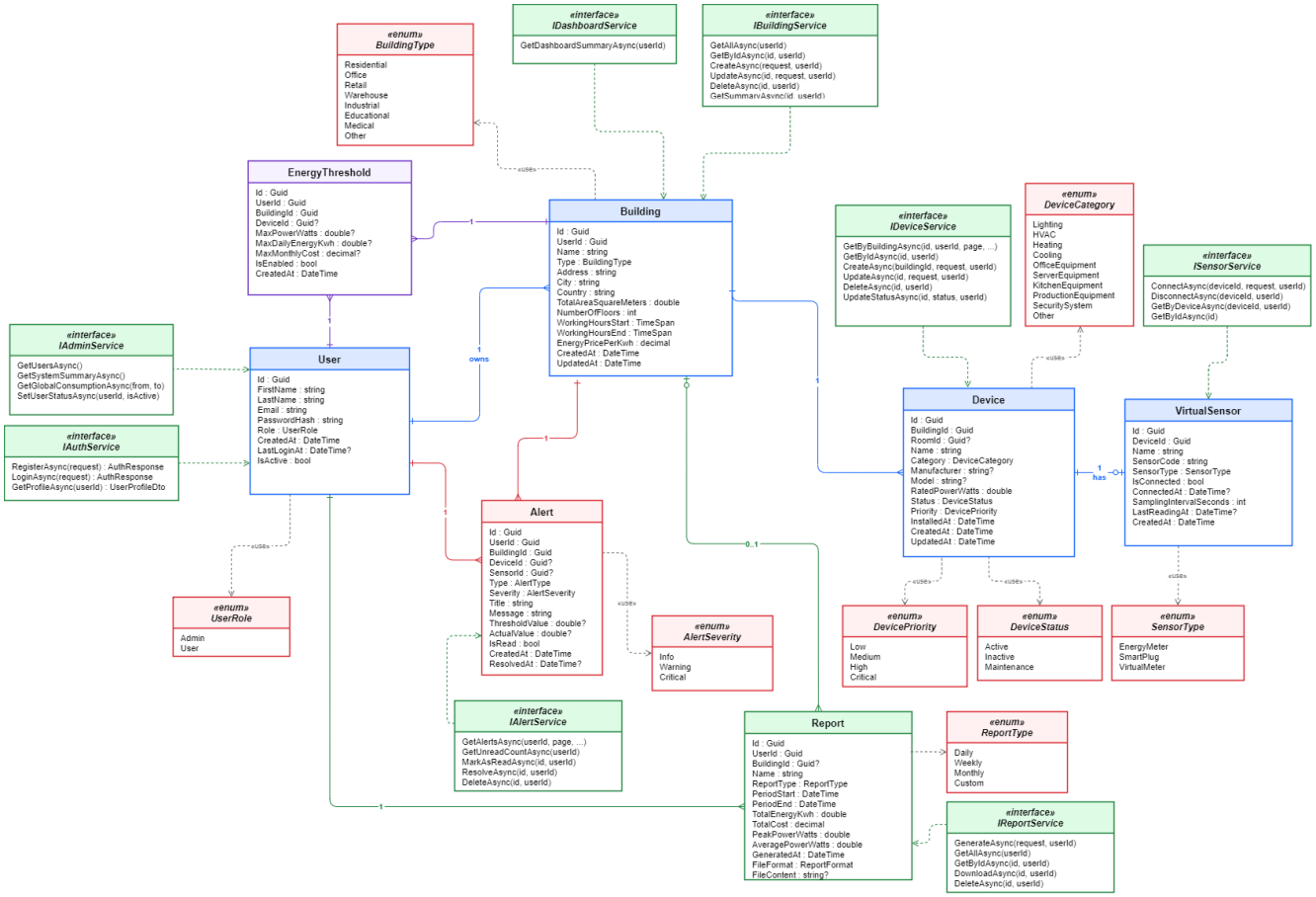

Рисунок 1.2 – Діаграма класів системи Energy Monitoring

Діаграма класів описує доменну модель. Основний ланцюжок зв’язків має вигляд User, Building, Device і VirtualSensor. Додаткові сутності Alert, EnergyThreshold, Report і Alert підтримують сповіщення, правила контролю та звітність. Інтерфейси сервісів показують, що бізнес операції винесено з контролерів.

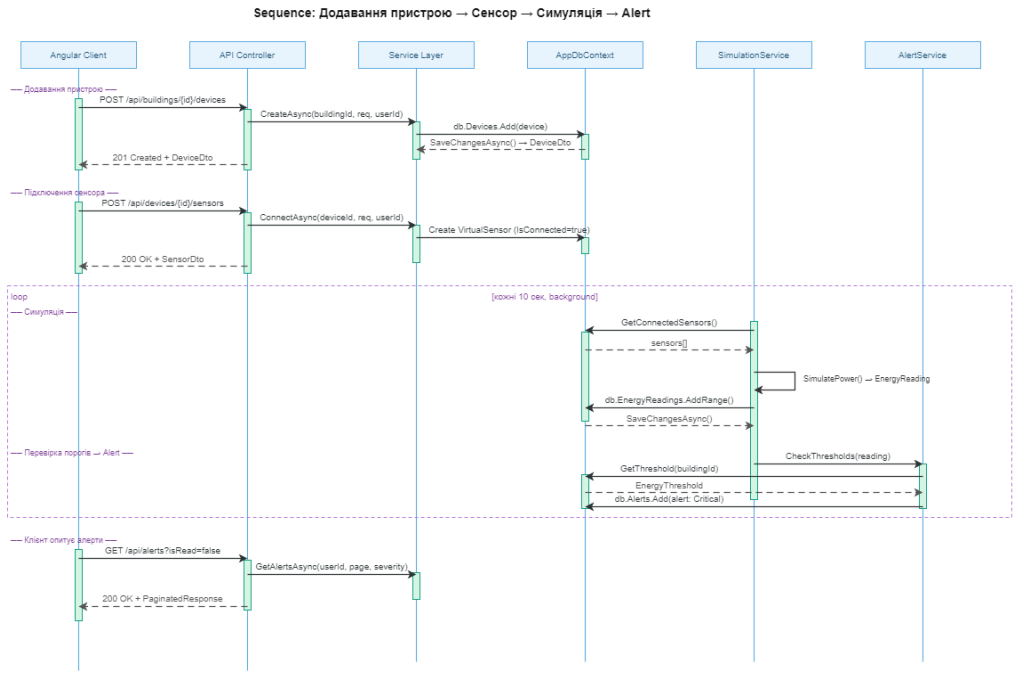

Рисунок 1.3 – Діаграма послідовності системи Energy Monitoring

Діаграма послідовності деталізує сценарій додавання пристрою, підключення сенсора, симуляції показників і створення алерта. Вона показує, що клієнт не працює з базою напряму.

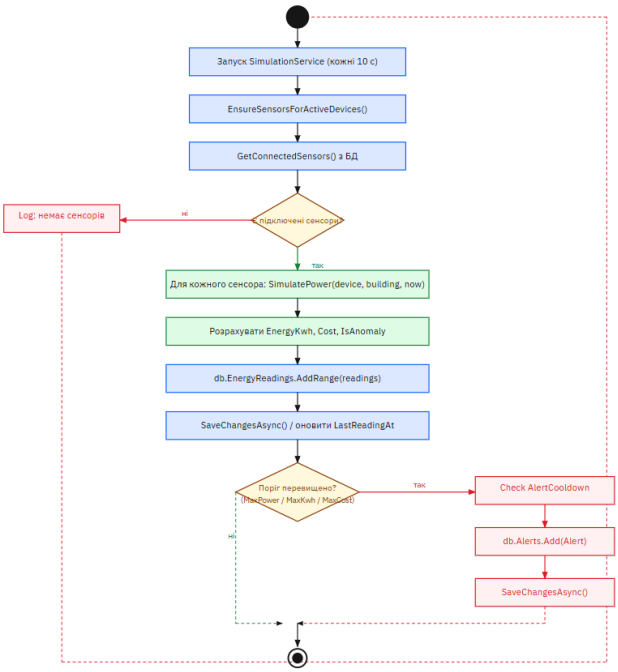

Рисунок 1.4 – Діаграма активності системи Energy Monitoring

Діаграма активності описує цикл роботи Simulation Service. Сервіс запускається, шукає активні пристрої без сенсорів, отримує підключені сенсори, генерує показники, зберігає їх і перевіряє пороги. Якщо поріг перевищено, створюється алерт з урахуванням cooldown.

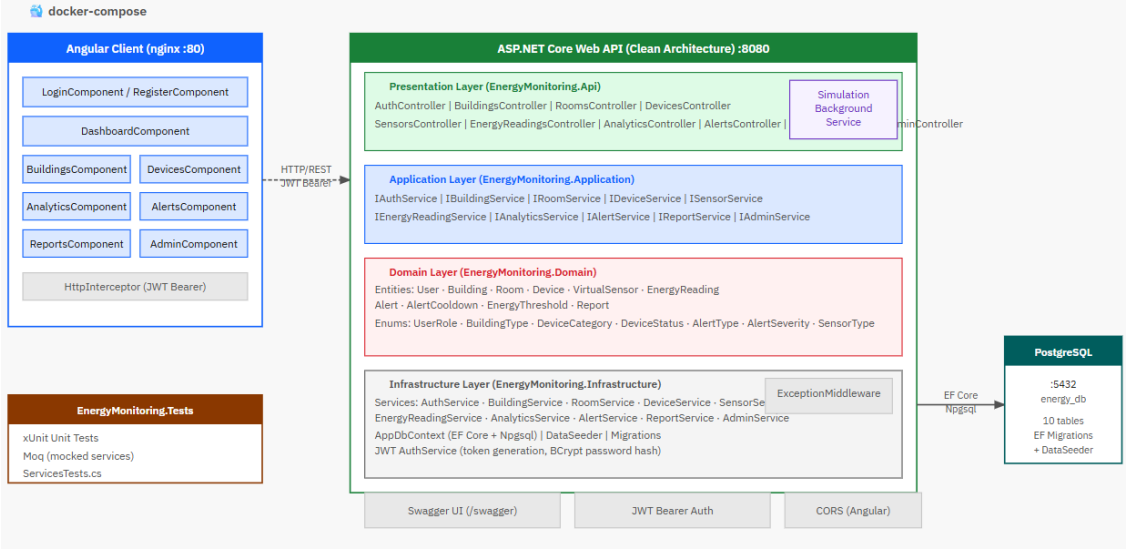

Рисунок 1.5 – Діаграма архітектури системи Energy Monitoring

Архітектурна діаграма показує контейнерну структуру та шари Clean Architecture. Angular клієнт звертається до ASP.NET Core API через HTTP. API складається з Presentation, Application, Domain та Infrastructure шарів. PostgreSQL працює як окремий контейнер.

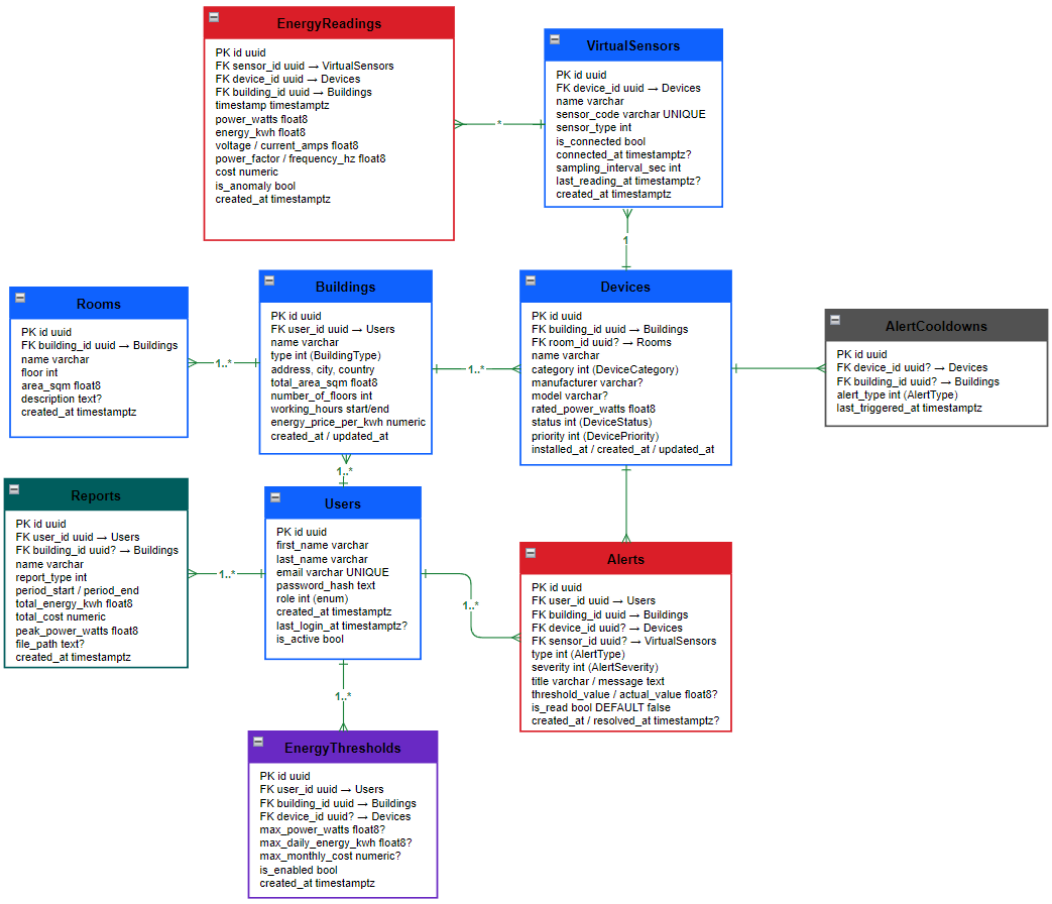

Рисунок 1.6 – Схема бази даних системи Energy Monitoring

Схема бази даних показує таблиці, первинні ключі, зовнішні ключі та основні поля. Вона підтверджує, що історія вимірювань пов’язана з сенсором, пристроєм і будівлею, а алерти та звіти прив’язані до користувача.

1.2.4 Архітектура Clean Architecture

Архітектурний підхід Clean Architecture було обрано з урахуванням перспектив подальшого розвитку системи. Оскільки застосунок передбачає можливість інтеграції з реальними сенсорами, розширення аналітики, зміну механізмів генерації звітів або додавання нових каналів взаємодії (наприклад, через SignalR), важливо забезпечити слабку зв’язаність між компонентами. У випадку, якщо бізнес-логіка буде тісно інтегрована з контролерами або кодом доступу до даних, внесення змін стане складним і ризикованим. Розділення системи на окремі

шари дозволяє локалізувати зміни та мінімізувати їхній вплив на інші частини застосунку [14].

У межах проєкту реалізовано чотиришарову структуру. Шар Domain містить основні сутності предметної області та допоміжні переліки. Він не залежить від інших частин системи і відображає чисту бізнес-модель. Шар Application відповідає за опис сценаріїв використання через DTO та інтерфейси сервісів. Шар Infrastructure реалізує технічні деталі, включаючи доступ до бази даних, механізми авторизації, ініціалізацію даних та роботу з міграціями. Шар API виступає точкою входу в систему, забезпечуючи обробку HTTP-запитів, налаштування middleware, документацію через Swagger, а також інтеграцію з механізмами безпеки.

Важливою характеристикою архітектури є напрямок залежностей, який завжди спрямований до внутрішніх шарів. Це означає, що зовнішні компоненти залежать від внутрішніх, але не навпаки. Завдяки цьому доменну модель можна аналізувати незалежно від конкретної технології зберігання даних або клієнтської частини. Такий підхід підвищує зрозумілість коду та спрощує його підтримку [14].

У рамках цієї архітектури контролери виконують мінімальний обсяг роботи. Вони відповідають за прийом HTTP-запитів, вилучення необхідних параметрів (наприклад, ідентифікатора користувача з токена) та передачу керування відповідним сервісам. Уся складна логіка, включаючи перевірки прав доступу, обробку даних і формування результатів, зосереджена на рівні Application та Infrastructure. Це дозволяє тестувати функціональність без необхідності запуску вебсервера.

Кожен із шарів має чітко визначену роль у системі.

Рівень Domain відповідає за опис основних бізнес-сутностей і правил їх взаємодії. У цьому шарі не допускається використання залежностей від бази даних або зовнішніх бібліотек. Його основне завдання — забезпечити чисту і незалежну модель предметної області, яку можна використовувати незалежно від конкретної реалізації інфраструктури.

Рівень Application виступає посередником між доменною моделлю та зовнішніми компонентами. Він визначає інтерфейси сервісів, описує сценарії використання системи та формує структури передачі даних. Саме тут реалізується логіка, яка координує взаємодію між різними частинами системи, не прив’язуючись до конкретних технологій.

Рівень Infrastructure забезпечує реалізацію технічних деталей. Він включає роботу з базою даних через ORM, реалізацію сервісів, підключення сторонніх бібліотек і взаємодію із зовнішніми системами. У цьому шарі допускається використання конкретних технологій, оскільки він ізольований від доменної логіки.

Рівень Presentation (API) відповідає за взаємодію із зовнішніми клієнтами. Він обробляє HTTP-запити, виконує первинну валідацію даних і повертає відповіді у визначеному форматі. Основне завдання цього шару — забезпечити зручний і стандартизований інтерфейс доступу до функціоналу системи.

Окремо варто розглянути клієнтську частину, реалізовану на Angular. Вона відповідає за відображення даних і взаємодію з користувачем, але не повинна містити критичну бізнес-логіку. Наприклад, клієнт може приховати певні елементи інтерфейсу, однак остаточне рішення щодо доступу до даних повинно прийматися на сервері.

Рівень бази даних також має обмежену відповідальність і використовується виключно для збереження та вибірки даних. У ньому не реалізується складна бізнес-логіка, оскільки це ускладнює тестування та знижує гнучкість системи.

Такий розподіл відповідальностей дозволяє уникнути змішування логіки між шарами, підвищує якість коду та забезпечує можливість подальшого розвитку системи без суттєвих змін у вже реалізованому функціоналі.

2 КОНСТРУЮВАННЯ ТА ВИКОРИСТАННЯ СИСТЕМИ

Конструювання та використання системи є завершальним етапом розробки, на якому абстрактні проєктні рішення набувають конкретної форми у вигляді працюючого програмного продукту. Саме на цьому етапі визначається, наскільки точно реалізований код відповідає сформульованим вимогам і наскільки ефективно обрані технології вирішують задачі предметної області. Від якості конструювання залежить не лише поточна функціональність системи, а й зручність її подальшого супроводу та розширення.

У межах цього розділу послідовно розглядаються практичні аспекти реалізації системи Energy Monitoring. Спочатку описується технологічний стек та обґрунтовується вибір конкретних інструментів і платформ, після чого детально висвітлюється організація бази даних, реалізація основних модулів серверної та клієнтської частин, а також механізм симуляції показників і генерації алертів. Окрема увага приділяється практичним питанням розгортання системи та типовим сценаріям її використання.

2.1 Конструювання системи

Конструювання системи охоплює практичну реалізацію рішень, прийнятих на етапі проєктування. У цьому підрозділі розглядаються використані технології та інструменти, організація бази даних, побудова основних модулів серверної і клієнтської частин, а також механізм симуляції показників та генерації алертів. Кожне технічне рішення обґрунтовується з урахуванням вимог до системи та особливостей предметної області.

2.1.1 Технології ASP.NET Core, Angular

Backend реалізовано на ASP.NET Core 8. Ця платформа підходить для створення продуктивних веб API, має вбудовану систему dependency injection,

middleware конвеєр, підтримку контролерів, фонових сервісів, авторизації, Swagger і хорошу інтеграцію з Entity Framework Core. Для Energy Monitoring це важливо, оскільки серверна частина одночасно обробляє HTTP запити і виконує періодичну симуляцію [1].

ASP.NET Core контролери реалізують REST ендпоїнти. Наприклад, AuthController відповідає за реєстрацію і вхід, BuildingsController за будівлі, DevicesController за пристрої, SensorsController за сенсори, AnalyticsController за аналітику, AlertsController за алерти, ReportsController за звіти, а AdminController за адміністративні дії. Такий поділ полегшує пошук коду і відповідає логічним модулям системи [2].

Angular використано для frontend частини. Клієнт має модульну структуру: core містить auth service, guards, interceptors, models і API сервіси, а features містить окремі сторінки. Компоненти відповідають за відображення даних, форми, фільтри і взаємодію з користувачем. API сервіси інкапсулюють HTTP запити, тому компоненти не дублюють URL адреси і заголовки [7].

JWT авторизація працює так: після входу сервер повертає токен. Angular зберігає його і AuthInterceptor додає заголовок Authorization до наступних запитів. На backend JwtBearer middleware перевіряє підпис, issuer, audience і термін дії. Якщо токен недійсний, захищений ендпоїнт не виконується [5, 18].

У вимогах до системи згадано SignalR як механізм real-time логіки. У поточній реалізації основна поведінка побудована через фоновий сервіс і періодичне оновлення клієнта. Це означає, що система вже має імітацію роботи в реальному часі на сервері, але миттєве push повідомлення на клієнт можна додати наступною ітерацією через SignalR Hub. Такий розвиток не суперечить архітектурі, бо алерти вже зберігаються як окрема сутність [3].

Таблиця 2.1 – Технологічний стек системи

Компонент	**Технологія**	**Призначення**
1	2	3

Продвження таблиці 2.1

1	2	3
Backend	ASP.NET Core 8	REST API, middleware, DI, hosted service.
Frontend	Angular	SPA інтерфейс, маршрути, компоненти, форми.
База даних	PostgreSQL 16	Збереження користувачів, об’єктів, показників, алертів і звітів.
ORM	Entity Framework Core	Міграції, DbContext, LINQ запити.
Авторизація	JWT Bearer	Захист API і передавання ролі користувача.
Контейнери	Docker Compose	Запуск frontend, backend і PostgreSQL.
Тести	xUnit, Moq	Модульна перевірка сервісів.
Документація API	Swagger/OpenAPI	Перевірка ендпоїнтів і контрактів.

2.1.2 СУБД PostgreSQL

PostgreSQL у системі використовується як основне сховище даних. Вибір цієї системи управління базами даних обумовлений її високою надійністю, відкритістю, широкими можливостями роботи зі складними SQL-запитами, а також хорошою сумісністю з контейнеризованими середовищами. Для задач енергомоніторингу особливо важливою є ефективна обробка часових рядів, підтримка фільтрації за періодами та виконання агрегованих обчислень, що PostgreSQL забезпечує на високому рівні [9].

Взаємодія із базою даних реалізована за допомогою Entity Framework Core, який дозволяє описувати структуру даних через об’єктну модель мовою C#. Такий підхід спрощує розробку, оскільки дає змогу працювати з базою на рівні класів, а також автоматично генерувати схему за допомогою механізму міграцій. У проєкті використовується початкова міграція, що створює таблиці, визначає зв’язки між ними та накладає обмеження цілісності. Під час запуску серверної частини міграції застосовуються автоматично, після чого виконується ініціалізація початкових даних за допомогою компонента DataSeeder [6, 24].

Для забезпечення ізольованого та відтворюваного середовища виконання PostgreSQL розгортається у контейнері Docker. У конфігураційному файлі docker-compose.yml визначено параметри запуску, включаючи образ postgres:16-alpine, назву бази даних, облікові дані користувача, мережеві порти, а також механізм збереження даних через томи. Додатково налаштовано перевірку готовності сервісу (healthcheck), що дозволяє backend-компоненту коректно очікувати запуску бази перед встановленням з'єднання, зменшуючи ймовірність помилок під час старту системи [10, 11].

Особливу увагу приділено таблиці EnergyReadings, яка є основним джерелом аналітичних даних і характеризується високою швидкістю зростання. У реальних умовах така таблиця може досягати значних обсягів, тому для промислового використання доцільно передбачити додаткові механізми оптимізації. До них належать партиціювання за часовими інтервалами, архівація застарілих даних та використання матеріалізованих представлень для збереження результатів частих агрегованих запитів. У межах бакалаврського проєкту реалізовано базову модель, однак її структура не обмежує можливість подальшого масштабування [9, 19].

Під час проєктування бази даних було враховано низку важливих аспектів, що впливають на продуктивність системи.

Зокрема, індексація за часовими полями є критичною для забезпечення швидкого доступу до даних за певний період. Оскільки більшість аналітичних запитів пов'язана з часовими інтервалами, правильний вибір індексів дозволяє суттєво зменшити час виконання запитів та уникнути повного сканування таблиці.

Фільтрація за будівлею або пристроєм також відіграє важливу роль, адже користувач зазвичай працює з конкретними об'єктами. Для цього доцільно використовувати комбіновані індекси, що враховують як часові параметри, так і ідентифікатори сутностей, забезпечуючи ефективну вибірку даних навіть при великому обсязі інформації.

Агрегація даних за певними інтервалами часу, наприклад за годинами або днями, є основою для побудови аналітичних графіків і звітів. Вона потребує

оптимізації запитів, оскільки обчислення на великій кількості записів можуть значно впливати на швидкодію системи. Використання попередньо обчислених значень або матеріалізованих представлень дозволяє зменшити навантаження на базу.

Збереження вартості енергоспоживання вимагає точності обчислень, тому відповідні поля реалізуються із використанням числових типів із фіксованою точністю. Це дозволяє уникнути помилок округлення та забезпечує коректність фінансових розрахунків.

Організація зв’язків між таблицями реалізована через зовнішні ключі, що гарантує цілісність даних. Водночас при проєктуванні враховано баланс між нормалізацією та продуктивністю, оскільки надмірна кількість з’єднань може ускладнювати виконання запитів.

Механізм міграцій схеми бази даних забезпечує контроль за її еволюцією. Він дозволяє відслідковувати зміни структури, автоматизувати їх застосування та забезпечувати узгодженість між різними середовищами розгортання. Це особливо важливо в умовах поступового розвитку системи та додавання нового функціоналу.

У сукупності зазначені підходи формують основу для побудови ефективної, масштабованої та надійної системи збереження даних, здатної обробляти значні обсяги інформації без втрати продуктивності.

2.1.3 Реалізація основних модулів

Система Energy Monitoring реалізована за модульним принципом, у межах якого кожен компонент відповідає за окрему частину функціональності. Такий підхід дозволяє ізолювати бізнес-логіку, спростити тестування та забезпечити можливість подальшого розширення системи без значних змін у вже реалізованих модулях.

Модуль авторизації відповідає за роботу з обліковими записами користувачів. У його межах реалізовано процеси реєстрації, входу в систему та отримання інформації про поточного користувача. Паролі зберігаються у

зашифрованому вигляді з використанням алгоритму BCrypt, а для взаємодії з API застосовуються JWT-токени. Уся логіка перевірки облікових даних і генерації токенів винесена у сервісний шар, що дозволяє відокремити її від HTTP-рівня та спростити тестування [5, 18, 23].

Модуль управління будівлями реалізує базові операції роботи з основними об'єктами системи. Користувач може створювати, редагувати та видаляти будівлі, а також отримувати їх список. Кожна будівля жорстко прив'язана до власника, що дозволяє обмежити доступ до даних. Окрім збереження інформації, модуль формує узагальнені показники, які використовуються для відображення на дашборді.

Модуль кімнат розширює функціональність будівель, додаючи можливість структуризації внутрішнього простору. Кімнати створюються в межах конкретної будівлі та дозволяють точніше описати розміщення обладнання. Основна увага приділяється підтримці цілісності зв'язків і коректній обробці операцій створення та видалення.

Модуль пристроїв забезпечує роботу з обладнанням, що генерує споживання енергії. У ньому зберігаються основні характеристики пристрою, такі як категорія, номінальна потужність, статус роботи та пріоритет. Пристрої можуть бути прив'язані як до будівлі, так і до конкретної кімнати, що забезпечує гнучкість структури. Після створення пристрій може бути використаний для підключення сенсора.

Модуль сенсорів відповідає за підключення та управління віртуальними сенсорами. Він виступає зв'язуючою ланкою між пристроями та системою генерації даних. При активації сенсора запускається фоновий процес, який моделює надходження показників. Логіка модуля враховує поточний стан сенсора та забезпечує його коректну взаємодію з іншими компонентами системи.

Модуль показників забезпечує доступ до даних енергоспоживання. Він дозволяє отримувати як історичні записи, так і актуальні значення, а також формувати агреговані результати за обраний період. Підтримується режим

відображення останніх даних, що дозволяє використовувати систему для моніторингу в реальному часі.

Модуль аналітики виконує обробку накопичених даних і формує узагальнену інформацію. У його межах реалізовано побудову дашборду, визначення найбільш енерговитратних пристроїв, аналіз пікових навантажень і розподіл споживання за категоріями. Додатково передбачено формування рекомендацій щодо оптимізації використання ресурсів, що підвищує практичну цінність системи [25, 26].

Модуль алертів призначений для інформування користувача про події, що виходять за межі нормального функціонування системи. Він забезпечує створення, зберігання та відображення повідомлень, а також дозволяє змінювати їхній статус. Особливістю є запобігання дублюванню алертів у випадку повторних аномалій.

Модуль звітів реалізує формування аналітичних звітів за заданий період. Він обчислює підсумкові показники, зберігає результати та забезпечує можливість їх подальшого перегляду або завантаження. Це дозволяє використовувати систему не лише для оперативного моніторингу, а й для підготовки узагальненої інформації.

Модуль адміністратора забезпечує управління користувачами та контроль стану системи. Адміністратор має доступ до списку користувачів, може змінювати їхній статус, а також переглядати загальні показники споживання. Для цього модуля реалізовано додаткові перевірки доступу, що гарантують безпеку системи.

У цілому архітектура системи побудована відповідно до принципу розділення відповідальностей. Контролери виконують роль інтерфейсу взаємодії з клієнтом, тоді як основна бізнес-логіка зосереджена у сервісному шарі. Такий підхід забезпечує гнучкість, спрощує тестування та створює умови для подальшого розвитку системи. Взаємодія між модулями здійснюється через чітко визначені інтерфейси, що унеможливлює пряму залежність між конкретними реалізаціями та знижує ризик виникнення побічних ефектів при внесенні змін. Завдяки цьому кожен модуль може розвиватися незалежно, а нові функціональні можливості можуть додаватися без необхідності суттєвого перегляду вже наявного коду.

Таблиця 2.2 – API модулі та відповідні контролери

Модуль	Контролер	Приклад ендпоїнта
Auth	AuthController	POST /api/auth/login
Buildings	BuildingsController	GET /api/buildings
Rooms	RoomsController	GET /api/buildings/{buildingId}/rooms
Devices	DevicesController	GET /api/buildings/{buildingId}/devices
Sensors	SensorsController	POST /api/devices/{deviceId}/sensor/connect
Readings	EnergyReadingsController	GET /api/readings/latest
Analytics	AnalyticsController	GET /api/analytics/dashboard
Alerts	AlertsController	GET /api/alerts
Reports	ReportsController	POST /api/reports/generate
Admin	AdminController	GET /api/admin/users

2.1.4 Симуляція і алерти

Симуляція є однією з головних особливостей Energy Monitoring. Вона дозволяє демонструвати поведінку системи без фізичних сенсорів. У backend реалізовано клас EnergyReadingSimulationService, який наслідує BackgroundService. Після запуску застосунку сервіс входить у цикл, викликає GenerateReadingsAsync, обробляє можливі помилки і чекає 10 секунд до наступної ітерації [4].

Перед генерацією сервіс перевіряє активні пристрої без сенсорів. Якщо такі є, для них автоматично створюється VirtualSensor з інтервалом 10 секунд. Це зручно для демонстрації, бо користувач може додати пристрій і система швидко почне формувати дані. У реальному продукті автоматичне створення сенсора можна замінити ручним прив’язуванням фізичного пристрою.

Для кожного підключеного сенсора сервіс отримує пристрій і будівлю. Якщо пристрій неактивний, він пропускається. Далі визначається кількість секунд від останнього показника, розраховується інтервал у годинах, симулюється

потужність, обчислюється енергія, напруга, струм, коефіцієнт потужності, частота і вартість. Запис додається до списку, а LastReadingAt сенсора оновлюється.

Алгоритм SimulatePower враховує категорію пристрою і робочий час будівлі. Освітлення в робочий час споживає значно більше, ніж уночі. HVAC, heating і cooling мають власні діапазони. ServerEquipment працює майже постійно. KitchenEquipment має піки в ранковий і обідній час. Додатково з малою ймовірністю додається стрибок, який може створити аномалію.

Аномалія визначається просто і зрозуміло: якщо потужність більша за 125 відсотків номінальної потужності пристрою, IsAnomaly отримує значення true. Після цього TryCreateAlertAsync перевіряє таблицю AlertCooldowns. Якщо за останні 30 хвилин для цього пристрою вже був алерт AbnormalSpike, новий алерт не створюється. Якщо cooldown не блокує подію, система додає Alert і оновлює cooldown [28].

Цей механізм є базовим, але логічним. Він не претендує на складне машинне навчання, проте добре показує, як у системі енергомоніторингу можна перейти від сирих показників до події, яка має значення для користувача. У майбутньому правило можна замінити статистичним профілем пристрою або моделлю прогнозування.

1. Фоновий сервіс запускається разом з API.
2. Сервіс перевіряє активні пристрої і створює сенсори за потреби.
3. Отримуються всі підключені сенсори.
4. Для кожного сенсора розраховується новий EnergyReading.
5. Записи зберігаються у PostgreSQL.
6. Якщо значення аномальне, створюється Alert з урахуванням cooldown.
7. Клієнт отримує нові дані через API під час періодичного оновлення.

У процесі моделювання енергоспоживання ключову роль відіграють окремі показники, які формують узгоджену систему даних. Кожен із них використовується не ізольовано, а як частина загальної логіки обчислень і відображення інформації.

Завдяки цьому симуляція не зводиться до генерації випадкових значень, а відтворює поведінку реальних електроприладів.

Показник потужності є базовим параметром, що характеризує миттєве навантаження пристрою. Саме на його основі формується уявлення про поточний стан споживання. Потужність використовується як для відображення користувачу в реальному часі, так і як вхідна величина для інших розрахунків, зокрема визначення енергоспоживання та виявлення перевантажень.

Енергія є інтегральним показником, який накопичується у часі на основі значень потужності. Вона дозволяє оцінити загальне споживання за певний період і є основою для подальших фінансових розрахунків. Саме значення енергії використовується при визначенні витрат, що робить цей показник ключовим для аналітики.

Вартість споживання формується на основі обсягу використаної енергії та тарифу, встановленого для конкретної будівлі. Цей показник є похідним, однак має практичну значущість для користувача, оскільки дозволяє оцінити економічний ефект від використання електроенергії та приймати рішення щодо оптимізації витрат.

Ознака аномалії використовується для виявлення нетипових ситуацій у роботі обладнання. Вона визначається шляхом порівняння фактичної потужності з номінальним значенням пристрою. У випадку перевищення встановленого порогу система фіксує відхилення, що може свідчити про несправність або неефективну роботу обладнання.

Параметр cooldown використовується для регулювання частоти генерації подій або повторного виникнення аномалій. Він дозволяє уникнути ситуацій, коли одна й та сама проблема генерує надмірну кількість сповіщень за короткий проміжок часу. Таким чином, забезпечується більш стабільна і контрольована робота системи моніторингу.

Останнє значення сенсора відіграє важливу роль у забезпеченні безперервності даних. Воно використовується як точка відліку для генерації

наступних значень, що дозволяє уникнути різких і нереалістичних змін у показниках. Завдяки цьому симуляція виглядає більш природною та наближеною до реальних умов.

Робочі години будівлі визначають часові рамки, у межах яких відбувається активне споживання енергії. Цей параметр впливає на характер генерації даних, оскільки в неробочий час рівень навантаження знижується. Врахування цього фактору дозволяє моделювати більш реалістичні сценарії використання ресурсів.

У сукупності всі зазначені показники формують взаємопов’язану систему, в якій кожен елемент впливає на інші. Такий підхід дозволяє створити симуляцію, що не лише генерує дані, а й відображає логіку реального енергоспоживання, забезпечуючи коректність аналітики та практичну цінність отриманих результатів.

2.2 Використання системи

Цей підрозділ описує практичні аспекти розгортання та експлуатації системи Energy Monitoring. Розглядається запуск системи через Docker Compose, типові сценарії взаємодії користувача з інтерфейсом, а також результати верифікації, що підтверджують відповідність реалізованого функціоналу поставленим вимогам.

2.2.1 Розгортання Docker

Для розгортання системи використовується Docker Compose. Це дозволяє запускати PostgreSQL, ASP.NET Core API і Angular frontend однією командою. Такий спосіб особливо зручний для демонстрації дипломного проєкту, тому що не потрібно окремо налаштовувати базу даних, порти і залежності на кожному комп’ютері [10, 11].

Файл docker-compose.yml описує три сервіси. Сервіс postgres використовує образ postgres:16-alpine, відкриває порт 5432 і має healthcheck через pg_isready. Сервіс api збирається з backend Dockerfile, отримує рядок підключення через змінні

середовища, відкриває порт 5000 і залежить від готовності postgres. Сервіс frontend збирається з Angular Dockerfile, працює через nginx і відкриває порт 4200.

Після запуску docker-compose up --build користувач може відкрити frontend за адресою http://localhost:4200, backend API за адресою http://localhost:5000, а Swagger UI за адресою http://localhost:5000/swagger. PostgreSQL доступний на localhost:5432. Для тестування передбачено seed облікових записів Admin і User.

Під час запуску API автоматично застосовує міграції і виконує seed. Це важливо, бо новий користувач проєкту не повинен вручну створювати таблиці. Якщо база порожня, схема створюється автоматично, а демонстраційні дані додаються в процесі старту.

Таблиця 2.3 – Сервіси Docker Compose

Сервіс	**Порт**	**Призначення**
postgres	5432	База даних PostgreSQL.
api	5000	ASP.NET Core Web API.
frontend	4200	Angular SPA через nginx.

2.2.2 Сценарії використання

Новий користувач відкриває сторінку реєстрації, створює обліковий запис, входить у систему і потрапляє на дашборд. Спочатку дашборд може не містити даних, бо користувач ще не створив будівлі та пристрої.

Користувач створює будівлю, наприклад офіс. Він вказує площу, адресу, робочий час і ціну за кіловат-годину. Після цього система може правильно розраховувати вартість споживання для всіх пристроїв цієї будівлі.

Користувач додає кімнати: серверну, кабінет, кухню або аудиторію. Кімнати допомагають краще організувати пристрої і в майбутньому можуть використовуватися для локальної аналітики.

Користувач додає пристрої: освітлення, кондиціонер, серверну шафу, принтер або кухонне обладнання. Для кожного пристрою вибирається категорія і номінальна потужність. Саме ці дані впливають на симуляцію.

Користувач підключає сенсор. Після цього фоновий сервіс починає створювати показники. Через деякий час на дашборді з’являються графіки, останні значення і підсумки.

Якщо симулятор згенерує стрибок, який перевищує 125 відсотків номінальної потужності, система створить алерт. Користувач побачить його на сторінці алертів і зможе позначити прочитаним або закрити.

Користувач переходить на сторінку аналітики, вибирає будівлю і період. Система показує динаміку споживання, топ пристроїв, розподіл за категоріями та пікові години.

Користувач генерує звіт за день, тиждень або місяць. Звіт містить загальну енергію, вартість, пікову потужність і середнє споживання.

2.2.3 Верифікація

Верифікація системи виконувалася через перевірку відповідності реалізації поставленим вимогам. Для цього використовувалися ручні сценарії у вебінтерфейсі, Swagger для перевірки API, запуск Docker Compose, перегляд записів у базі даних і модульні тести. Важливо було перевірити не лише позитивні сценарії, а й помилки, наприклад неправильний пароль, відсутність токена або спробу доступу до чужих даних [16, 17].

Окремо перевірено роботу симуляції. Після додавання активного пристрою і підключення сенсора в таблиці EnergyReadings мають з’являтися нові записи. Інтервал створення записів відповідає 10 секундам. Якщо пристрій переведено в неактивний стан, симулятор пропускає його. Якщо потужність перевищує поріг аномалії, створюється запис Alert.

Дашборд перевірявся через порівняння даних API і відображення в Angular. Сторінка не повинна ламатися, якщо даних ще немає. Це важливо для нового

користувача. Також перевірено періодичне оновлення, яке виконується через interval на клієнті.

Верифікація розгортання включала запуск контейнерів з нуля. Після команди docker-compose up --build мають успішно стартувати postgres, api і frontend. API має застосувати міграції, Swagger має відкриватися, а frontend має виконувати запити до backend.

Таблиця 2.4 – Перевірка вимог

Вимога	**Спосіб перевірки**	**Очікуваний результат**
Авторизація	Вхід з правильним і неправильним паролем	Токен видається тільки для правильних даних.
CRUD будівель	Створення, редагування, видалення	Дані змінюються і належать поточному користувачу.
Симуляція	Очікування після підключення сенсора	У базі з’являються EnergyReading.
Алерти	Створення аномального значення	З’являється Alert без дублювання через cooldown.
Аналітика	Запит dashboard і сторінки analytics	Повертаються агреговані дані.
Docker	Запуск docker-compose up --build	Усі сервіси стартують і доступні на своїх портах.

3 ТЕСТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

Тестування є невід'ємною складовою процесу розробки програмного забезпечення, яка дозволяє підтвердити відповідність реалізованої системи поставленим вимогам і виявити потенційні недоліки до введення продукту в експлуатацію. Для складних багатокомпонентних систем, таких як Energy Monitoring, якість тестування безпосередньо визначає надійність кінцевого продукту, оскільки помилки в окремих модулях можуть негативно впливати на роботу всієї системи в цілому.

У межах цього розділу описується підхід до тестування системи Energy Monitoring, що охоплює декілька рівнів перевірки. Спочатку формулюється план тестування, який визначає цілі, методи та інструменти перевірки кожного компонента. Далі розглядається розробка конкретних тестових сценаріїв для автоматизованого та мануального тестування, після чого наводяться результати перевірки основних функціональних модулів із детальним описом виконаних дій та отриманих результатів.

3.1 План тестування

Тестування системи Energy Monitoring спрямоване на перевірку коректності роботи основних модулів, стабільності API, правильності реалізації механізмів авторизації, надійності симуляції та відповідності інтерфейсу очікуваним сценаріям використання. Оскільки система складається з серверної частини, клієнтського застосунку та бази даних, процес тестування охоплює декілька рівнів і включає як автоматизовані, так і ручні перевірки.

План тестування передбачає використання модульних тестів для перевірки окремих сервісів, інтеграційного тестування API за допомогою Swagger або HTTP-клієнта, мануального тестування інтерфейсу користувача, перевірки розгортання в

середовищі Docker, а також аналізу негативних сценаріїв. Для модульного тестування застосовано бібліотеки xUnit та Moq, які дозволяють ізолювати бізнес-логіку сервісів і перевіряти її без необхідності запуску всієї системи [21, 22].

Мануальне тестування відіграє важливу роль у перевірці системи, оскільки дозволяє оцінити її роботу з точки зору кінцевого користувача. На відміну від автоматизованих тестів, які перевіряють окремі функції, ручна перевірка дає змогу проаналізувати повний сценарій взаємодії, включаючи навігацію, відображення даних, зручність використання та обробку помилок. Тестування виконувалося у браузері шляхом роботи з Angular-інтерфейсом із паралельним контролем запитів до API.

Особливу увагу приділено перевірці роботи симуляції, яка функціонує у фоновому режимі. Помилки в цьому компоненті можуть не проявлятися одразу, але впливають на результати аналітики. У процесі тестування аналізувалося створення записів показників, частота їх генерації, оновлення поля LastReadingAt, а також коректність формування алертів. Виявлено, що навіть незначні відхилення у логіці симуляції можуть призводити до некоректних результатів у звітах і на дашборді.

Окремим напрямом тестування є перевірка безпеки системи. Запити без токена авторизації не повинні надавати доступ до захищених ресурсів, а користувачі не повинні мати можливість отримувати дані, що належать іншим користувачам. Також перевірено, що адміністративні ендпоїнти доступні лише для ролі Admin. Навіть у випадку, коли клієнтська частина приховує відповідні елементи інтерфейсу, серверна частина виконує обов’язкову перевірку прав доступу [27, 28].

Таблиця 3.1 – План тестування

Рівень	**Що перевіряється**	**Інструменти**
1	2	3

Продовження таблиці 3.1

1	2	3
Модульний	Окремі сервіси і методи	xUnit, Moq
API	Ендпоїнти, коди відповіді, DTO	Swagger, HTTP клієнт
Інтерфейс	Форми, маршрути, відображення даних	Ручне тестування у браузері
База даних	Міграції, зв’язки, записи	PostgreSQL, EF Core
Симуляція	Створення показників і алертів	Логи, база даних, API
Розгортання	Контейнери і порти	Docker Compose

Тестування системи виконувалося поетапно для кожного функціонального модуля з використанням однакового підходу: спочатку перевірялися коректні (позитивні) сценарії використання, після чого аналізувалася поведінка системи у випадку помилкових або некоректних вхідних даних. Така стратегія дозволяє спочатку підтвердити базову працездатність функціоналу, а потім оцінити його стійкість до некоректних дій користувача.

У процесі мануального тестування було перевірено роботу всіх основних модулів системи. Для модуля авторизації аналізувалася коректність реєстрації та входу, а також реакція системи на помилкові облікові дані. У модулі будівель перевірялися операції створення, редагування і видалення об’єктів та контроль доступу до них. Модулі кімнат і пристроїв тестувалися з точки зору правильності побудови ієрархії та збереження характеристик обладнання.

У модулі управління будівлями тестувалися операції створення, редагування та видалення об’єктів, а також доступ до них з боку різних користувачів. Додатково перевірялися сценарії, пов’язані з відсутністю необхідних полів, некоректними значеннями або спробами доступу до чужих даних.

Тестування модуля кімнат передбачало перевірку коректності створення ієрархії всередині будівлі. Окремо аналізувалися випадки додавання кімнат до неіснуючих будівель або некоректного видалення елементів, що могло призвести до порушення зв’язків між сутностями.

У модулі пристроїв перевірялася правильність збереження характеристик обладнання та його прив’язки до відповідних об’єктів. Негативні сценарії включали спроби створення пристроїв без обов’язкових параметрів або з некоректними значеннями технічних характеристик.

Для модуля сенсорів тестування включало перевірку процесів підключення та відключення сенсорів, а також реакцію системи на спроби виконання цих дій у некоректних умовах, наприклад при відсутності пов’язаного пристрою.

Модуль показників перевірявся з точки зору отримання історичних даних, актуальних значень та агрегованої інформації. Окремо аналізувалися випадки запитів із некоректними параметрами, такими як неправильні часові інтервали або відсутність даних.

У модулі аналітики тестування було спрямоване на перевірку коректності обчислень і формування узагальненої інформації. Додатково перевірялася поведінка системи у випадку недостатньої кількості даних або некоректних вхідних параметрів для побудови аналітики.

Модуль алертів перевірявся з точки зору відображення сповіщень, зміни їхнього статусу та обробки сценаріїв із відсутніми або некоректними ідентифікаторами повідомлень.

Для модуля звітів тестування включало перевірку генерації звітів за різні періоди, збереження метаданих та коректність завантаження результатів. Негативні сценарії охоплювали випадки некоректно заданих параметрів або відсутності даних для формування звіту.

У модулі адміністрування перевірялися функції перегляду користувачів, зміни їхнього статусу та доступу до системної інформації. Особливу увагу приділено перевірці обмежень доступу, щоб запобігти виконанню адміністративних дій неавторизованими користувачами.

Таким чином, поєднання автоматизованого та мануального тестування дозволило не лише перевірити окремі компоненти системи, а й оцінити її поведінку

в реальних умовах використання. Це забезпечує більш повну перевірку якості програмного забезпечення та підвищує його надійність.

3.2 Розробка тестів

Розробка тестів виконувалася з урахуванням того, що найважливіша бізнес логіка міститься в сервісах. Контролери повинні бути простими, тому їх достатньо перевіряти через API сценарії. Сервіси потребують детальніших тестів, бо саме вони створюють сутності, перевіряють права доступу, виконують агрегації і повертають DTO.

Для модульних тестів використано xUnit. Moq застосовується для створення підроблених залежностей там, де потрібно ізолювати об’єкт тестування. У випадках, де логіка тісно пов’язана з EF Core, можна використовувати тестову базу або in-memory провайдер. У межах цієї роботи основний акцент зроблено на перевірці сервісного шару і сценаріїв, які легко відтворити [21, 22].

Тести повинні мати зрозумілі назви. Добра назва описує умову, дію і очікуваний результат. Наприклад, Login_WithValidCredentials_ReturnsToken або CreateBuilding_WhenUserExists_CreatesBuilding. Такий стиль допомагає швидко зрозуміти, що саме зламалося, якщо тест не проходить.

Окремо підготовлено ручні тест кейси. Вони потрібні для перевірки UI, бо автоматичні unit тести не покажуть, чи зручно користувачу працювати зі сторінкою, чи правильно відображаються порожні стани, чи не перекриваються елементи інтерфейсу і чи зрозумілі повідомлення про помилки.

Таблиця 3.2 – Приклади тест кейсів

ID	**Сценарій**	**Кроки**	**Очікуваний результат**
1	2	3	4

Продовження таблиці 3.2

1	2	3	4
TC1	Успішний вхід	Ввести email і пароль тестового користувача, натиснути Login	Користувач переходить на dashboard, токен збережено.
TC2	Помилка входу	Ввести неправильний пароль	Система показує повідомлення про помилку, токен не створюється.
TC3	Створення будівлі	Заповнити форму будівлі і зберегти	Будівля з’являється у списку.
TC4	Додавання пристрою	Вибрати будівлю, ввести пристрій і потужність	Пристрій збережено і доступний для сенсора.
TC5	Підключення сенсора	Натиснути Connect sensor для пристрою	Створюється VirtualSensor зі станом connected.
TC6	Симуляція	Зачекати понад 10 секунд після підключення сенсора	З’являється новий EnergyReading.
TC7	Алерт	Згенерувати або дочекатися аномального стрибка	У списку алертів з’являється сповіщення.
TC8	Звіт	Створити звіт за період	Звіт збережено і доступний для перегляду або завантаження.
TC9	Admin сторінка	Увійти як User і перейти на admin route	Доступ заборонено або сторінка недоступна.
TC10	Docker запуск	Виконати docker-compose up --build	Усі сервіси стартують без критичних помилок.

Під час тестування системи окрему увагу було приділено допоміжним аспектам, які не пов’язані безпосередньо з основною бізнес-логікою, однак суттєво впливають на зручність використання та стабільність роботи інтерфейсу. Такі перевірки дозволяють виявити дрібні, але помітні для користувача недоліки.

Аспект валідації форм досліджувався з точки зору коректності введення даних користувачем. Перевірялися сценарії введення некоректних значень, пропущених обов’язкових полів та неправильного формату даних. Важливим

критерієм була наявність зрозумілих повідомлень про помилки, які дозволяють швидко виправити введені дані без необхідності повторного заповнення форми.

Перевірка ролей і прав доступу була спрямована на забезпечення безпеки системи. Аналізувалися ситуації, коли користувач намагався отримати доступ до функціоналу, який йому не дозволений. Очікуваною поведінкою системи є обмеження доступу та повернення відповідного повідомлення без розкриття внутрішніх деталей реалізації.

Аспект обробки порожніх списків тестувався для забезпечення коректного відображення інтерфейсу у випадках відсутності даних. Замість відображення пустих таблиць або некоректних елементів інтерфейсу система повинна інформувати користувача про відсутність записів, наприклад через повідомлення або спеціальний стан сторінки.

Пагінація перевірялася з точки зору правильності навігації між сторінками даних. Аналізувалися сценарії переходу вперед і назад, коректність відображення кількості елементів та поведінка при досягненні кінця списку. Також перевірялося, чи не виникають дублювання або пропуски записів при перемиканні сторінок.

Фільтрація за датою тестувалася для перевірки точності вибірки даних за заданий період. Особлива увага приділялася граничним значенням, таким як початок і кінець діапазону, а також коректності обробки некоректно введених дат.

Обробка помилок API була важливим аспектом з точки зору взаємодії з користувачем. У випадках, коли сервер повертає помилку або недоступний, інтерфейс повинен коректно відобразити повідомлення без технічних деталей, таких як stack trace, які можуть бути незрозумілими або небезпечними.

Повторне оновлення дашборду тестувалося для забезпечення стабільності відображення даних у динамічному режимі. Перевірялося, чи правильно оновлюється інформація без перезавантаження сторінки, а також чи не виникають конфлікти або затримки при частих запитах до сервера.

Збереження звітів аналізувалося з точки зору коректності формування та подальшого доступу до створених документів. Перевірялося, чи зберігаються

метадані, чи доступні звіти після створення та чи не виникають помилки при повторному завантаженні.

У результаті проведеного тестування було підтверджено, що навіть другорядні аспекти системи суттєво впливають на загальне враження від роботи з нею. Їх опрацювання дозволяє підвищити якість інтерфейсу, зробити систему більш зрозумілою для користувача та зменшити кількість потенційних помилок під час експлуатації.

3.3 Мануальне тестування

На початковому етапі було перевірено механізм реєстрації та авторизації користувачів. У процесі тестування здійснювалося створення нового облікового запису із заповненням відповідних полів форми, після чого виконувалася процедура входу в систему. У разі успішної авторизації сервер генерує JWT-токен, який використовується для подальшої взаємодії з API.

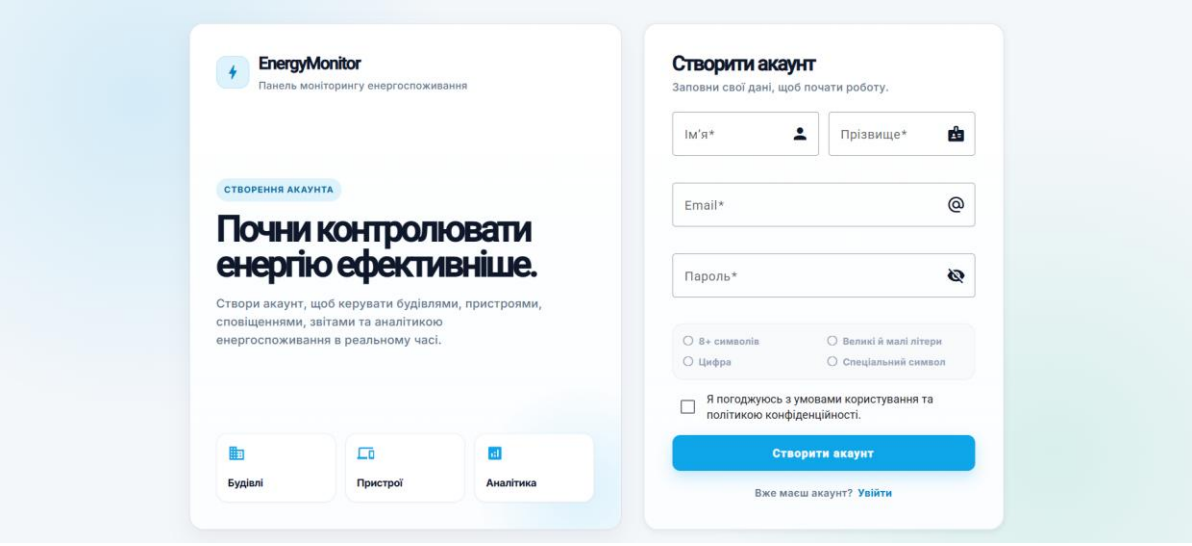

Рисунок 3.1 – Форма реєстрації користувача

Було перевірено коректність обробки введених даних, реакцію системи на неправильні облікові дані, а також збереження та використання токена доступу.

Система повинна забезпечувати захищений доступ до ресурсів і не дозволяти виконання запитів без авторизації.

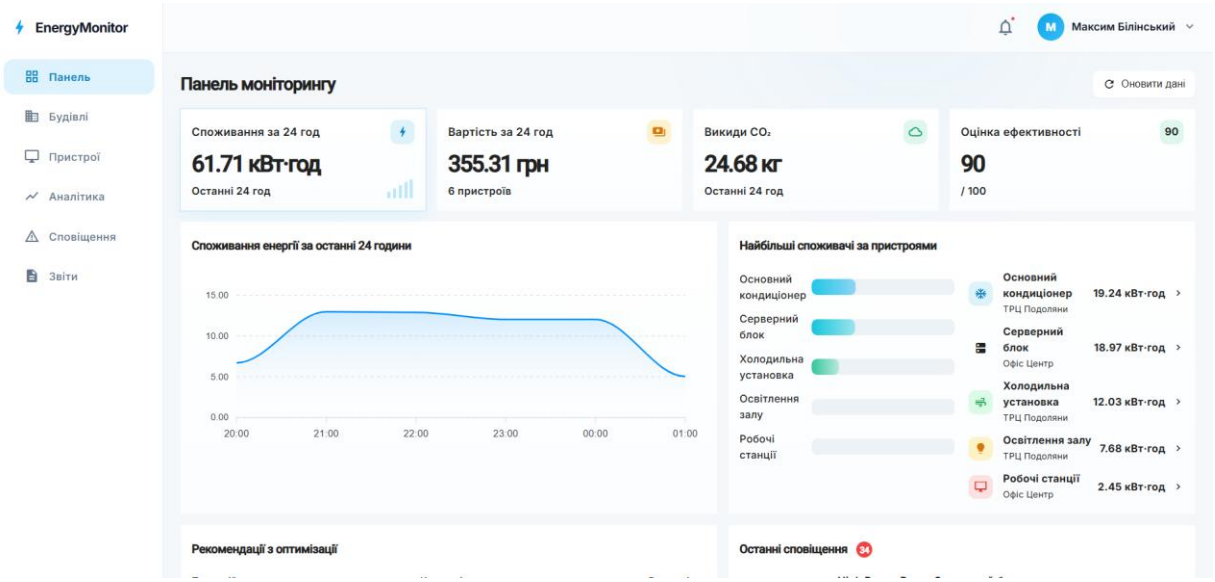

Рисунок 3.2 – Результат успішної авторизації користувача

Після входу в систему було протестовано функціонал управління будівлями. Користувач має можливість створювати нові об’єкти, редагувати їх параметри та видаляти. У процесі перевірки було встановлено, що дані коректно зберігаються в базі та відображаються у списку.

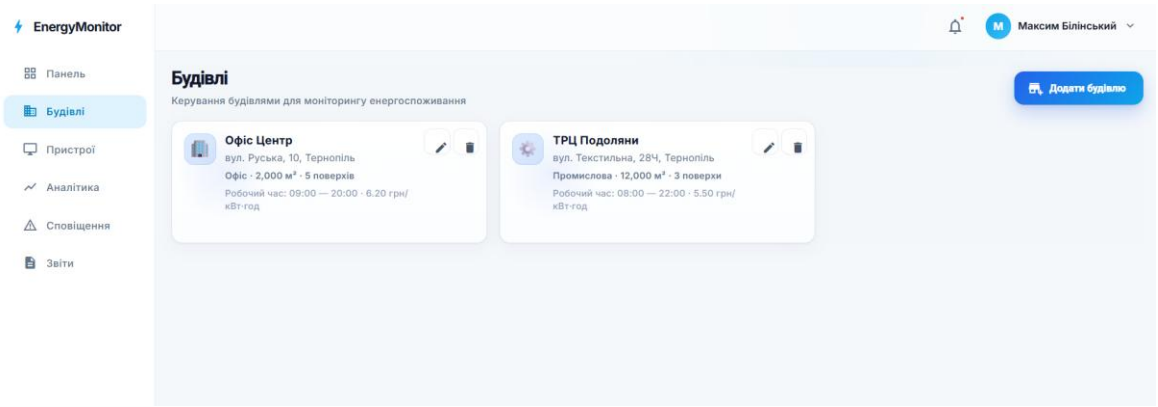

Рисунок 3.3 – Список будівель користувача

Окремо було перевірено форму створення будівлі, яка повинна забезпечувати введення необхідних параметрів та їх валідацію перед збереженням.

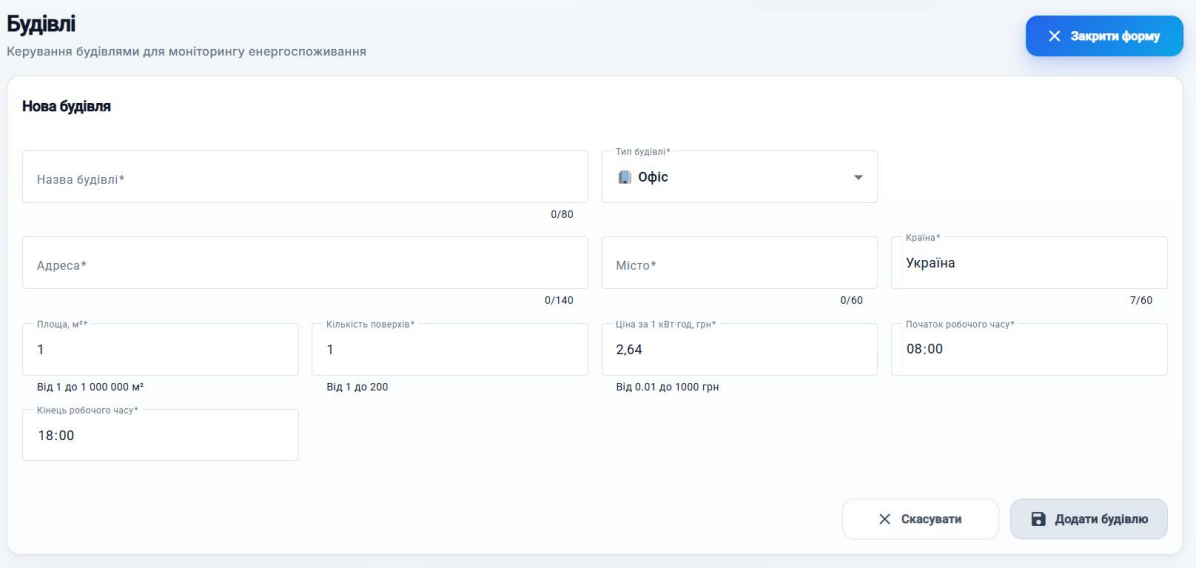

Рисунок 3.4 – Форма створення будівлі

Наступним етапом було тестування роботи з пристроями. У процесі тестування було перевірено додавання пристроїв із відповідними характеристиками.

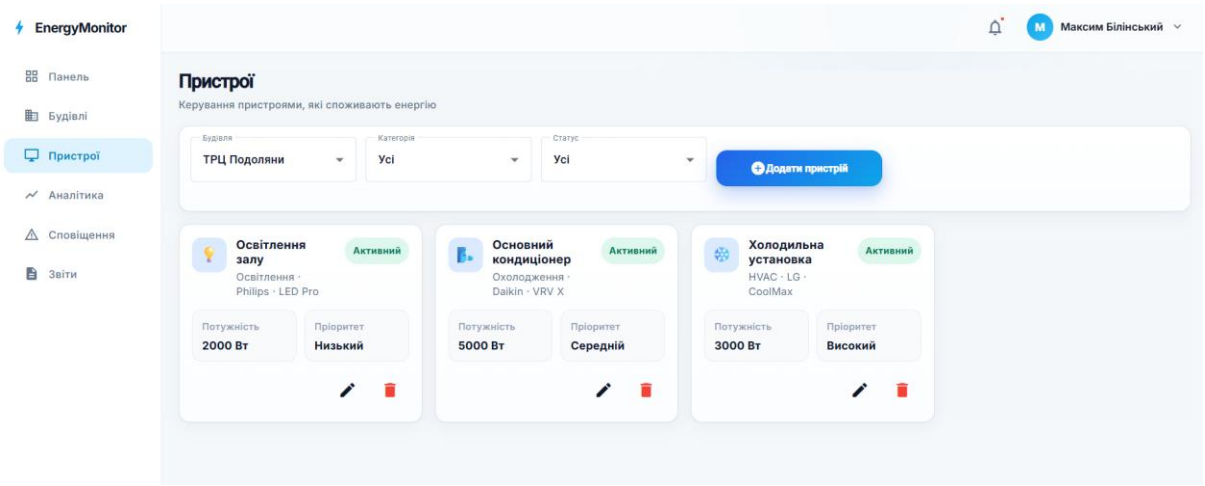

Рисунок 3.5 – Список пристроїв

Окремо було протестовано дашборд, який відображає узагальнену інформацію про стан системи. Було перевірено правильність обчислення сумарного споживання, вартості та кількості активних алертів.

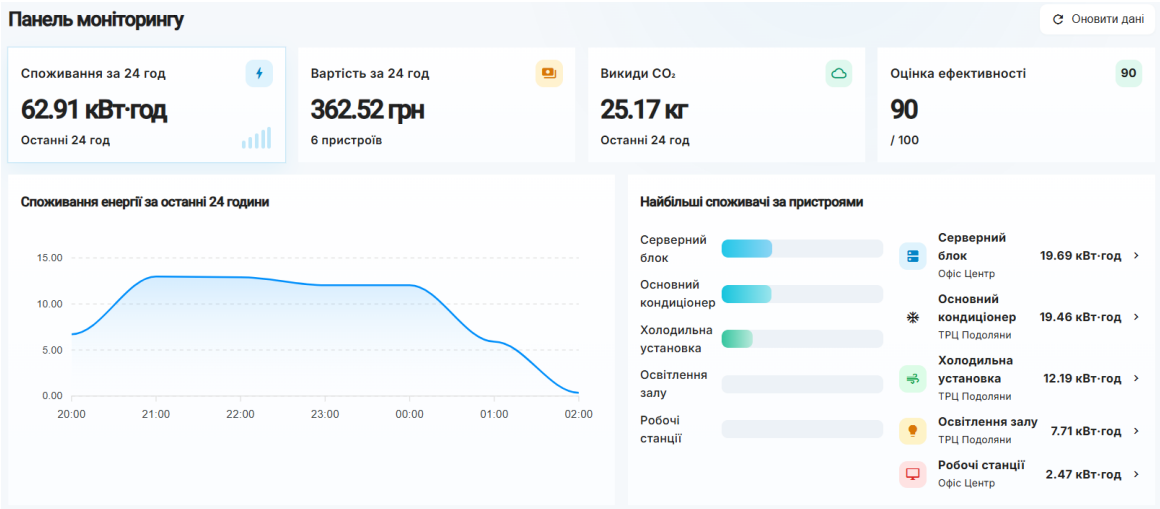

Рисунок 3.6 – Дашборд системи енергомоніторингу

Тестування аналітичного модуля передбачало перевірку формування узагальнених даних за обраний період. Було проаналізовано коректність фільтрації за датою та точність обчислень.

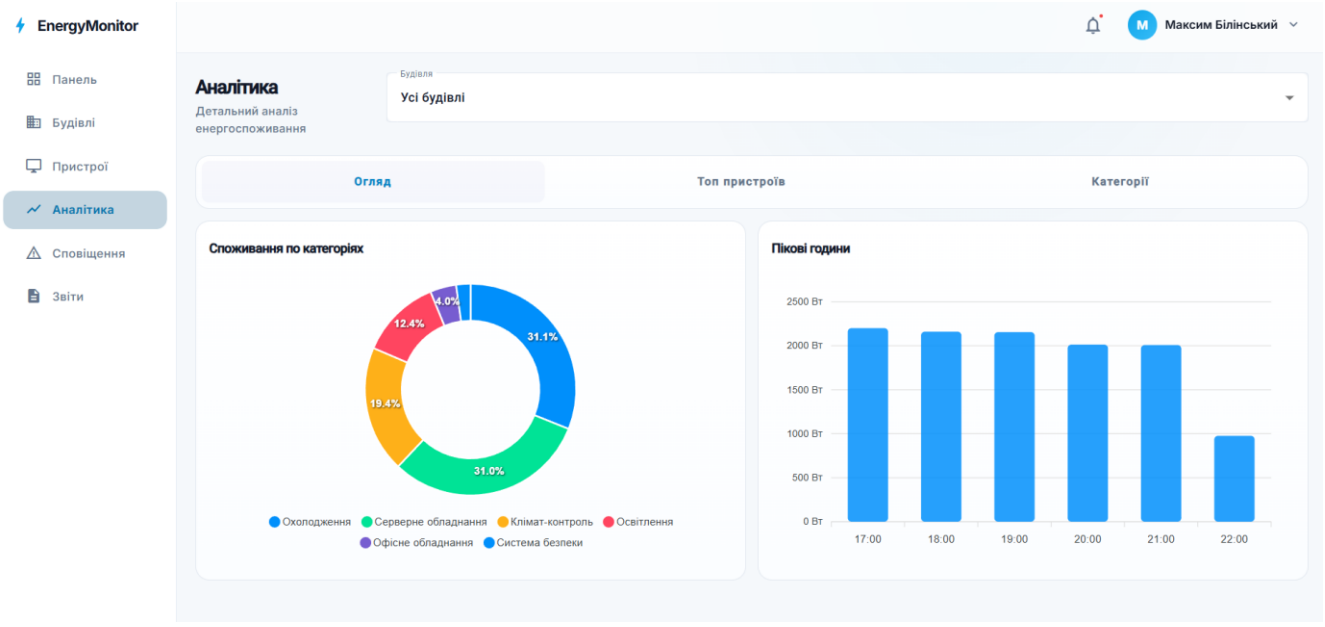

Рисунок 3.7 – Сторінка аналітики

У процесі тестування системи алертів було перевірено створення повідомлень у випадках виникнення аномальних значень та їх відображення у списку.

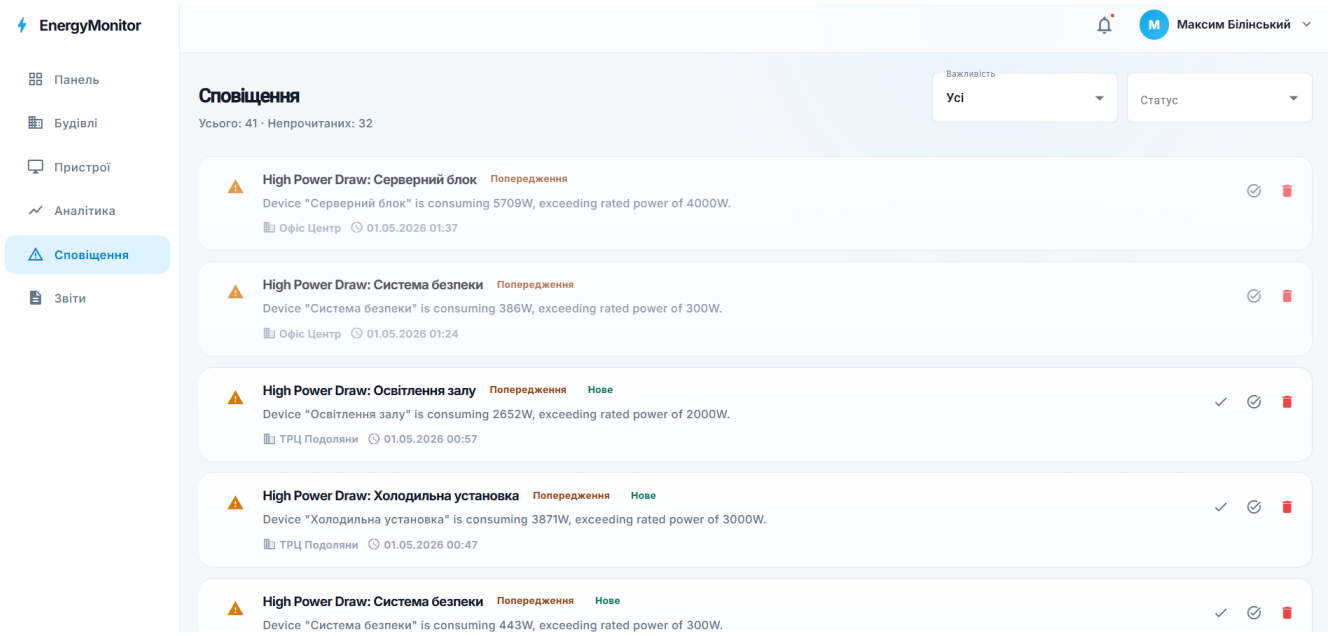

Рисунок 3.8 – Список алертів

Наступним етапом було тестування функціоналу формування звітів. Було перевірено можливість створення звітів за обраний період та їх повторного перегляду.

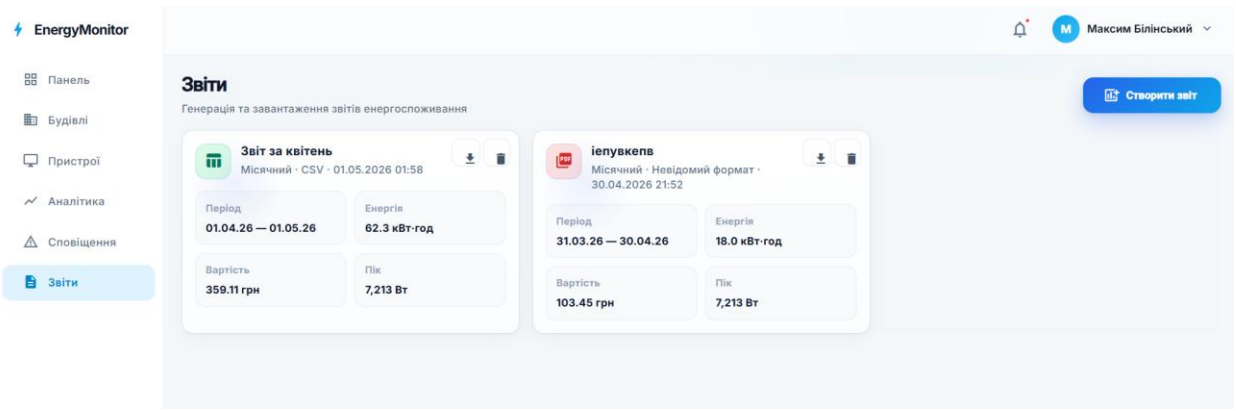

Рисунок 3.9 – Створення звіту

3.4 Аналіз результатів тестування

За результатами тестування система виконує основні вимоги. Користувач може зареєструватися, увійти, створити будівлю, додати кімнати і пристрої, підключити сенсор, отримати показники, переглянути дашборд, побачити алерти і сформувати звіт. Адміністратор має доступ до окремих адміністративних функцій [16].

Симуляція показників працює стабільно. Нові записи створюються з інтервалом 10 секунд для активних пристроїв з підключеними сенсорами. Якщо пристрій неактивний, показники для нього не генеруються. Аномальні стрибки позначаються через IsAnomaly і можуть створювати алерти. Cooldown зменшує кількість повторних повідомлень.

Під час перевірки було виявлено, що поточна реалізація не використовує SignalR для миттєвого push оновлення. Оновлення дашборду виконується через періодичний запит з клієнта. Для дипломного проєкту це прийнятно, бо фонові дані реально генеруються на сервері, а клієнт регулярно їх отримує. Проте для промислової версії доцільно додати SignalR Hub, який буде надсилати повідомлення про новий алерт без очікування наступного polling циклу [3].

Також варто врахувати потенційне зростання таблиці EnergyReadings. У демонстраційній версії обсяг даних контрольований, але при великій кількості пристроїв потрібні індекси, архівація і можливо партиціювання. Це не є помилкою реалізації, але є важливим напрямом розвитку.

Таблиця 3.3 – Підсумок результатів тестування

Блок	**Результат**	**Коментар**
1	2	3
Авторизація	Успішно	JWT видається і перевіряється.
CRUD модулі	Успішно	Основні операції виконуються.

Продовження таблиці 3.3

1	2	3
Симуляція	Успішно	Показники створюються кожні 10 секунд.
Алерти	Успішно	Аномалії створюють сповіщення з cooldown.
Аналітика	Успішно	Дані агрегуються і відображаються.
Розгортання	Успішно	Docker Compose запускає всі сервіси.
SignalR	Потребує розвитку	Архітектурно можливий, але в поточному коді не реалізований.

Під час оцінювання якості розробленої системи було використано низку критеріїв, які розглядалися не ізольовано, а в контексті реальних сценаріїв використання. Такий підхід дозволяє оцінити не лише формальну правильність окремих компонентів, а й їхню узгоджену роботу в межах повного користувацького процесу.

Критерій стабільності API аналізувався з точки зору надійності обробки запитів у різних умовах. Перевірялося, чи коректно система реагує на послідовні запити, чи не виникають збої при навантаженні та чи зберігається узгодженість відповідей. Особлива увага приділялася сценаріям, у яких користувач виконує декілька дій підряд, наприклад отримує аналітику після зміни параметрів.

Зручність інтерфейсу оцінювалася через швидкість виконання типових дій і зрозумілість взаємодії. Аналізувалося, наскільки інтуїтивно користувач може знайти потрібний функціонал, чи не виникає зайвих кроків у процесі роботи та чи є інтерфейс достатньо інформативним для прийняття рішень.

Коректність обчислень перевірялася шляхом порівняння отриманих результатів із очікуваними значеннями. Це стосувалося як базових розрахунків енергоспоживання, так і похідних показників, зокрема вартості та агрегованих значень. Важливою була не лише точність, а й узгодженість результатів у різних частинах системи.

Захист маршрутів оцінювався з позиції безпеки доступу до функціоналу. Перевірялося, чи обмежений доступ до певних ресурсів для неавторизованих користувачів, а також чи правильно застосовуються ролі та права доступу. Такий підхід дозволяє запобігти несанкціонованому використанню системи.

Читабельність коду розглядалася як фактор, що впливає на подальшу підтримку та розвиток проєкту. Оцінювалася структурованість, зрозумілість і логічність реалізації, а також відповідність загальноприйнятим принципам програмування. Це особливо важливо у випадку командної розробки або передачі проєкту іншому розробнику.

4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ ТА ОСНОВИ ОХОРОНИ ПРАЦІ

Розробка вебзастосунку Energy Monitoring для автоматизованого моніторингу енергоспоживання в будівлях і приміщеннях є прикладом застосування сучасних інформаційних технологій у сфері охорони праці та безпеки виробничих процесів.

З огляду на специфіку розробленої системи та умови праці розробника, у даному розділі розглядаються: менеджмент безпеки як системний підхід до управління охороною праці; значення автоматизації виробничих процесів у питаннях охорони праці та роль системи Energy Monitoring як інструменту автоматизованого контролю безпеки.

4.1 Менеджмент безпеки

Менеджмент безпеки — це систематизований процес управління ризиками та небезпеками на підприємстві, що включає планування, організацію, контроль і коригування заходів з охорони праці. В основі менеджменту безпеки лежить система управління охороною праці (СУОП), яка будується за циклом PDCA: Plan (плануй) — Do (виконуй) — Check (перевіряй) — Act (коригуй).

Правову основу менеджменту безпеки в Україні складають: Закон України «Про охорону праці» від 21.11.2002 р.; Кодекс законів про працю України; Кодекс цивільного захисту України від 01.07.2013 р.; Порядок розслідування та обліку нещасних випадків, професійних захворювань та аварій на виробництві від 17.04.2019 р. № 337.

Для підприємства, де розробляється та впроваджується система Energy Monitoring, менеджмент безпеки охоплює весь життєвий цикл: від проектування серверної інфраструктури Docker до щоденної роботи операторів, які користуються вебінтерфейсом для контролю енергоспоживання.

Центральним елементом менеджменту безпеки є кількісна оцінка ризику. Ризик — це усвідомлена кількісна оцінка ймовірності виникнення події з певними небажаними наслідками. Ступінь ризику R визначається як відношення кількості небезпек, що проявилися з негативними наслідками (n), до можливої їх кількості (N) за конкретний період часу:

$$R = n / N. \quad (4.1)$$

Для оцінки ризиків, пов'язаних з розробкою та експлуатацією системи Energy Monitoring, застосовується матричний метод. Кожна небезпека характеризується двома параметрами: категорією серйозності (I–IV) та рівнем ймовірності (A–E). Класифікація наведена в таблицях 4.1 та 4.2.

Таблиця 4.1 — Категорії серйозності небезпеки

Вид	**Категорія**	**Опис нещасного випадку**
Катастрофічна	I	Смерть або руйнування системи
Критична	II	Серйозна травма, стійке захворювання, суттєве пошкодження системи
Гранична	III	Незначна травма, короткочасне захворювання, пошкодження в системі
Незначна	IV	Менш значні, ніж у III категорії, травми та пошкодження

Таблиця 4.2 — Рівні ймовірності небезпеки

Вид	**Рівень**	**Опис**
Часта	A	Велика ймовірність того, що подія відбудеться
Можлива	B	Може трапитися кілька разів за життєвий цикл
Випадкова	C	Іноді може відбутися за життєвий цикл
Віддалена	D	Малоймовірна, але можлива подія протягом життєвого циклу
Неймовірна	E	Настільки малоймовірно, що можна вважати, що ніколи не відбудеться

Застосуємо матричний метод оцінки ризику до двох характерних небезпек робочого місця розробника програмного забезпечення.

Небезпека 1: Ураження електричним струмом при несправності обладнання (пошкоджений кабель живлення сервера, де розгорнуто Docker-контейнери системи). Категорія серйозності — II (критична): можлива серйозна травма або загибель. Рівень ймовірності — D (віддалена): офісне приміщення без підвищеної небезпеки, сучасне обладнання з захистом. Індекс ризику: 2D — небажаний (гранично допустимий).

Небезпека 2: Хронічне перевтомлення розробника внаслідок тривалої роботи з ВДТ без регламентованих перерв (психофізіологічна небезпека). Категорія серйозності — III (гранична): незначна травма, короткочасне захворювання (синдром зорової втоми, м'язово-скелетні розлади). Рівень ймовірності — B (можлива): може траплятися кілька разів за трудову діяльність при недотриманні режиму праці. Індекс ризику: 3B — небажаний. Результати оцінки ризиків та заплановані заходи зменшення наведено в таблиці 4.3.

Таблиця 4.3 — Матриця оцінки ризику та заходи зменшення для небезпек робочого місця розробника

Небезпека	**Категорія серйозності**	**Рівень ймовірності**	**Індекс ризику**	**Клас ризику**	**Заходи зменшення**
Ураження електричним струмом	II Критична	D Віддалена	2D	Небажаний (гранично допустимий)	Захисне заземлення ≤4 Ом; щорічна перевірка ізоляції мегомметром; автоматч. вимикач 10 А
Хронічне перевтомлення розробника при роботі з ВДТ	III Гранична	B Можлива	3B	Небажаний (гранично допустимий)	Регламентовані перерви 15 хв/год; обмеження роботи з ВДТ до 6 год/зміну; контроль освітленості ≥300 лк

Для планування заходів СУОП важливо мати кількісні орієнтири. Виконаємо розрахунок статистичного ризику загибелі на виробництві в Україні за 2020–2022 роки. Протягом цього періоду на дорогах України загинуло 15 300 осіб. Населення України — 38 млн осіб. Статистичний ризик за три роки:

$$R = n / N = 15\,300 / 38\,000\,000 \approx 4{,}03 \times 10^{-4}. \quad (4.2)$$

Річний ризик:

$$R\_рік = R / 3 = 4{,}03 \times 10^{-4} / 3 \approx 1{,}34 \times 10^{-4}. \quad (4.3)$$

Порівняння з нормованими рівнями: у світовій практиці прийнятний індивідуальний ризик загибелі становить 10^{-6} на рік; малим вважається ризик 10^{-8} на рік. Обчислений ризик $1{,}34 \times 10^{-4}$ перевищує прийнятний рівень у 134 рази — це надмірний ризик, що вимагає системних заходів. Саме тому менеджмент безпеки, включаючи автоматизовані системи контролю на зразок Energy Monitoring, є критично важливим.

У рамках СУОП для підприємства, де впроваджується Energy Monitoring, плануються такі заходи: щорічний аналіз стану охорони праці з оцінкою ризиків за матричним методом; проведення інструктажів — вступного, первинного, повторного (двічі на рік) та позапланового; розслідування нещасних випадків відповідно до Порядку № 337; використання даних системи Energy Monitoring для виявлення аномального споживання електроенергії як сигналу можливої несправності обладнання.

4.2. Значення автоматизації виробничих процесів у питаннях охорони праці

Автоматизація виробничих процесів є одним із найефективніших напрямів підвищення рівня охорони праці. Головний принцип: виведення людини з небезпечної зони або скорочення часу її перебування в умовах впливу шкідливих та небезпечних факторів. Згідно зі статистикою, до 75% нещасних випадків на виробництві обумовлені людським фактором — помилками персоналу внаслідок втоми, неуважності або недостатньої кваліфікації. Автоматизований контроль усуває цю причину.

Основні напрями впливу автоматизації на безпеку праці: автоматичний контроль параметрів технологічних процесів (температура, тиск, напруга, струм); автоматична сигналізація при виході параметрів за допустимі межі; автоматичне аварійне відключення обладнання; дистанційне керування в небезпечних зонах; реєстрація та аналіз даних для прогнозування відмов.

Функції системи Energy Monitoring в контексті охорони праці наведено в таблиці 4.4.

Таблиця 4.4 — Функції системи у забезпеченні безпеки праці

Функція системи	**Значення для охорони праці**	**Реалізація в системі**
Автоматичні алерти при перевищенні норм	Миттєве сповіщення без ручного обходу — персонал не перебуває в небезпечній зоні	Модуль сповіщень Angular + SignalR
Моніторинг споживання електроенергії в реальному часі	Виявлення перевантажень, що можуть призвести до пожежі або ураження струмом	API ASP.NET Core, дані з датчиків → PostgreSQL
Аналіз аномалій споживання	Раннє виявлення несправностей обладнання до виходу з ладу	Алгоритми порівняння з baseline у PostgreSQL
Формування звітів та історія подій	Документальна база для розслідування інцидентів (Порядок № 337)	Модуль звітності, експорт даних
Симуляція навантажень	Перевірка граничних режимів без реального ризику для персоналу	Модуль симуляції в Docker-середовищі

Для обґрунтування ефективності автоматизації порівняємо ручний обхід обладнання з автоматизованим моніторингом засобами Energy Monitoring за ключовими показниками охорони праці.

Таблиця 4.5 — Порівняння ручного та автоматизованого контролю безпеки

Критерій	**Ручний контроль (обхід)**	**Автоматизований (Energy Monitoring)**
Частота контролю	1–2 рази на зміну (дискретно)	Безперервно, 24/7
Час виявлення аномалії	До кількох годин	Секунди — хвилини (алерт)
Перебування персоналу в небезпечній зоні	Регулярно, при кожному обході	Відсутнє або мінімальне
Вплив людського фактора	Висока ймовірність помилки при втомі	Виключений (автоматика)
Документування подій	Ручний журнал, можливі помилки	Автоматична реєстрація в PostgreSQL
Реакція на аварію	Залежить від графіку обходу	Миттєве сповіщення відповідального
Психофізіологічне навантаження на персонал	Висока втома від монотонних обходів	Знижене — оператор реагує лише на алерти

Автоматизація не лише усуває людину з небезпечних зон, але й змінює характер праці персоналу. Оператор системи Energy Monitoring виконує переважно розумову діяльність — роботу з комп'ютером (115 кКал/год), аналіз даних, прийняття рішень. Це суттєво відрізняється від фізичної праці електрика (190–220 кКал/год), якого автоматизований моніторинг частково замінює при рутинному контролі.

Добові енерговитрати розробника системи наведено в таблиці 4.6.

Таблиця 4.6 — Добовий хронометраж енерговитрат розробника системи

№	Вид діяльності протягом доби	Витрати часу	Енерговитрата, кКал/год	Разом, кКал
1	Нічний сон	7 год	77	539
2	Ранковий туалет	40 хв	144	96
3	Сніданок	20 хв	103	34
4	Дорога на роботу	40 хв	120	80
5	Програмування (Angular, ASP.NET Core)	3 год	115	345
6	Участь у нараді / код-рев'ю (бесіда сидячи)	1 год	106	106
7	Обід	30 хв	103	52
8	Налагодження Docker-контейнерів, тестування	2 год	115	230
9	Написання документації	1 год	112	112
10	Прийняття рішень з архітектури системи	30 хв	270	135
11	Дорога додому	40 хв	120	80
12	Вечеря	30 хв	103	52
13	Фізичні вправи	1 год	300	300
14	Читання / відпочинок	2 год	90	180
15	Вечірній туалет	30 хв	102	51
16	Нічний сон (залишок)	2 год 10 хв	77	167
	Разом за добу	**24 год**	**—**	**2559**

Як видно з таблиці 4.6, добові енерговитрати розробника складають 2559 кКал, що відповідає легкій розумовій праці. Переважання розумової діяльності (робота з комп'ютером — 5,5 год, 115 кКал/год) над фізичною зумовлює специфічні психофізіологічні ризики: зорову втому, статичне навантаження на хребет, стрес при відлагодженні складних компонентів системи. Автоматизація рутинних операцій за допомогою інструментів CI/CD та Docker Compose дозволяє знизити монотонність праці та зменшити ймовірність помилок, спричинених втомою.

Впровадження автоматизованих систем контролю безпеки регламентується рядом нормативних документів. Відповідно до Закону України «Про охорону праці» роботодавець зобов'язаний впроваджувати сучасні засоби захисту, у тому

числі автоматичні системи контролю небезпечних факторів виробничого середовища. Автоматизовані системи сигналізації та контролю на електроустановках обов'язкові відповідно до ПУЕ для установок підвищеної небезпеки.

Система Energy Monitoring, розроблена в рамках кваліфікаційної роботи, реалізує вимоги щодо автоматичного контролю та реєстрації параметрів електроспоживання, що відповідає сучасним вимогам до систем управління енергоефективністю та безпекою виробничих об'єктів. Ведення автоматизованого журналу подій у базі даних PostgreSQL забезпечує документальну базу для розслідування інцидентів згідно з Порядком розслідування нещасних випадків № 337 від 17.04.2019 р.

Висновки до розділу 4

У розділі розглянуто два взаємопов'язані питання охорони праці в контексті розробки системи Energy Monitoring. В рамках менеджменту безпеки виконано оцінку ризиків матричним методом: небезпека ураження електричним струмом отримала індекс 2D — небажаний ризик; хронічне перевтомлення розробника — індекс 3В — небажаний ризик. Розраховано статистичний річний ризик загибелі на дорогах України ($1{,}34 \times 10^{-4}$), що перевищує світовий прийнятний рівень (10^{-6}) у 134 рази, що підтверджує необхідність системного менеджменту безпеки.

Система Energy Monitoring розглянута як інструмент автоматизованого контролю безпеки: безперервний моніторинг 24/7, миттєві алерти при аномаліях споживання та автоматична реєстрація подій у PostgreSQL усувають ключові недоліки ручного контролю і знижують вплив людського фактора — головної причини до 75% виробничих нещасних випадків. Добові енерговитрати розробника складають 2559 кКал, що відповідає легкій розумовій праці з переважанням роботи за комп'ютером.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи розроблено вебзастосунок Energy Monitoring для моніторингу енергоспоживання з симуляцією показників, аналітикою, алертами і звітністю. Система реалізована на основі ASP.NET Core, Angular і PostgreSQL та розгортається через Docker Compose.

Під час аналізу предметної області визначено основні сутності: користувач, будівля, кімната, пристрій, віртуальний сенсор, показник енергоспоживання, алерт, поріг і звіт. На основі цих сутностей сформовано вимоги, акторів, use case і словник системи.

Проєктування виконано з використанням Clean Architecture. Такий підхід дав змогу відокремити доменну модель, контракти сервісів, інфраструктурну реалізацію та API. Завдяки цьому система стала зрозумілішою і більш придатною до розвитку.

База даних PostgreSQL містить таблиці для всіх основних сутностей і підтримує зв’язки через зовнішні ключі. Таблиця EnergyReadings є центральною для аналітики, тому в роботі окремо пояснено питання частих записів, фільтрації та майбутньої оптимізації.

Реалізовано фоновий сервіс симуляції, який кожні 10 секунд генерує показники для підключених сенсорів. Алгоритм враховує категорію пристрою, робочі години будівлі, номінальну потужність і випадкові коливання. При перевищенні 125 відсотків номінальної потужності створюється аномалія і може формуватися алерт.

Клієнтська частина на Angular забезпечує реєстрацію, вхід, роботу з основними об’єктами, дашборд, аналітику, алерти, звіти та адміністративну сторінку. Для захисту запитів використовується JWT токен і HTTP interceptor.

Тестування підтвердило працездатність основних сценаріїв. Перевірено авторизацію, CRUD операції, симуляцію, алерти, аналітику, звіти і Docker запуск. Виявлено напрям розвитку: додавання SignalR для миттєвого push оновлення

клієнта, оскільки в поточній реалізації використовується періодичне оновлення через API.

Практична цінність роботи полягає у створенні повного прототипу, який можна використовувати для демонстрації принципів енергомоніторингу або розвивати до системи з реальними сенсорами. Архітектура не обмежує подальше розширення, тому до системи можна додати IoT протоколи, складніші правила аномалій, ML прогнозування, SignalR сповіщення і розширену звітність [29, 30].

СПИСОК ДЖЕРЕЛ

1. Microsoft Learn. ASP.NET Core documentation [Електронний ресурс]. — Режим доступу: https://learn.microsoft.com/en-us/aspnet/core/

2. Microsoft Learn. Create web APIs with ASP.NET Core [Електронний ресурс]. — Режим доступу: https://learn.microsoft.com/en-us/aspnet/core/web-api/

3. Microsoft Learn. ASP.NET Core SignalR [Електронний ресурс]. — Режим доступу: https://learn.microsoft.com/en-us/aspnet/core/signalr/introduction

4. Microsoft Learn. Background tasks with hosted services in ASP.NET Core [Електронний ресурс]. — Режим доступу: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/host/hosted-services

5. Microsoft Learn. Authentication and authorization in ASP.NET Core [Електронний ресурс]. — Режим доступу: https://learn.microsoft.com/en-us/aspnet/core/security/authentication/

6. Microsoft Learn. Entity Framework Core documentation [Електронний ресурс]. — Режим доступу: https://learn.microsoft.com/en-us/ef/core/

7. Angular documentation [Електронний ресурс]. — Режим доступу: https://angular.dev/

8. Angular. HTTP client guide [Електронний ресурс]. — Режим доступу: https://angular.dev/guide/http

9. PostgreSQL Global Development Group. PostgreSQL 16 Documentation [Електронний ресурс]. — Режим доступу: https://www.postgresql.org/docs/16/

10. Docker Docs. Docker Compose overview [Електронний ресурс]. — Режим доступу: https://docs.docker.com/compose/

11. Docker Docs. Compose file reference [Електронний ресурс]. — Режим доступу: https://docs.docker.com/reference/compose-file/

12. OpenAPI Initiative. OpenAPI Specification [Електронний ресурс]. — Режим доступу: https://spec.openapis.org/oas/latest.html

13. Fowler M. Patterns of Enterprise Application Architecture. — Boston : Addison-Wesley, 2022. — 560 p.

14. Martin R. C. Clean Architecture: A Craftsman’s Guide to Software Structure and Design. — Boston : Prentice Hall, 2017. — 432 p.

15. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. — Boston : Addison-Wesley, 1994. — 395 p.

16. ISO/IEC/IEEE 29148:2018. Systems and software engineering — Life cycle processes — Requirements engineering. — Geneva : ISO, 2018.

17. ISO/IEC 25010:2011. Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models. — Geneva : ISO, 2021.

18. JWT.io. Introduction to JSON Web Tokens [Електронний ресурс]. — Режим доступу: https://jwt.io/introduction

19. PostgreSQL Global Development Group. Documentation: Indexes [Електронний ресурс]. — Режим доступу: https://www.postgresql.org/docs/16/indexes.html

20. Microsoft Learn. ASP.NET Core middleware [Електронний ресурс]. — Режим доступу: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/middleware/

21. xUnit.net documentation [Електронний ресурс]. — Режим доступу: https://xunit.net/

22. Moq documentation [Електронний ресурс]. — Режим доступу: https://github.com/devlooped/moq

23. BCrypt.Net library repository [Електронний ресурс]. — Режим доступу: https://github.com/BcryptNet/bcrypt.net

24. Npgsql Entity Framework Core Provider [Електронний ресурс]. — Режим доступу: https://www.npgsql.org/efcore/

25. Chart.js documentation [Електронний ресурс]. — Режим доступу: https://www.chartjs.org/docs/latest/

26. ReactiveX. RxJS documentation [Електронний ресурс]. — Режим доступу: https://rxjs.dev/

27. OWASP Foundation. JSON Web Token Cheat Sheet [Електронний ресурс]. — Режим доступу:

https://cheatsheetseries.owasp.org/cheatsheets/JSON_Web_Token_for_Java_Cheat_Sheet.html

28. OWASP Foundation. REST Security Cheat Sheet [Електронний ресурс]. — Режим доступу: https://cheatsheetseries.owasp.org/cheatsheets/REST_Security_Cheat_Sheet.html

29. Energy Monitoring System : вихідний код проєкту, наданий для кваліфікаційної роботи. — Тернопіль, 2026.

30. UML діаграми Energy Monitoring System : HTML-файл з діаграмами, наданий для кваліфікаційної роботи. — Тернопіль, 2026.

31. Центр громадського здоров’я МОЗ України. Харчові отруєння та їх профілактика [Електронний ресурс]. — Режим доступу: https://phc.org.ua/

32. Міністерство охорони здоров’я України. Домедична допомога при невідкладних станах [Електронний ресурс]. — Режим доступу: https://moz.gov.ua/

33. Наказ Міністерства охорони здоров’я України «Про затвердження порядків надання домедичної допомоги особам при невідкладних станах» [Електронний ресурс]. — Режим доступу: https://zakon.rada.gov.ua/

34. Закон України «Про охорону праці» [Електронний ресурс]. — Режим доступу: https://zakon.rada.gov.ua/laws/show/2694-12

35. Правила безпечної експлуатації електроустановок споживачів [Електронний ресурс]. — Режим доступу: https://zakon.rada.gov.ua/

36. ДСТУ EN ISO 45001:2019. Системи управління охороною здоров’я та безпекою праці. Вимоги та настанови щодо застосування. — Київ : ДП «УкрНДНЦ», 2019.

ДОДАТКИ

ДОДАТОК А. ІЛЮСТРАЦІЇ ВАРІАНТІВ ВИКОРИСТАННЯ СИСТЕМИ

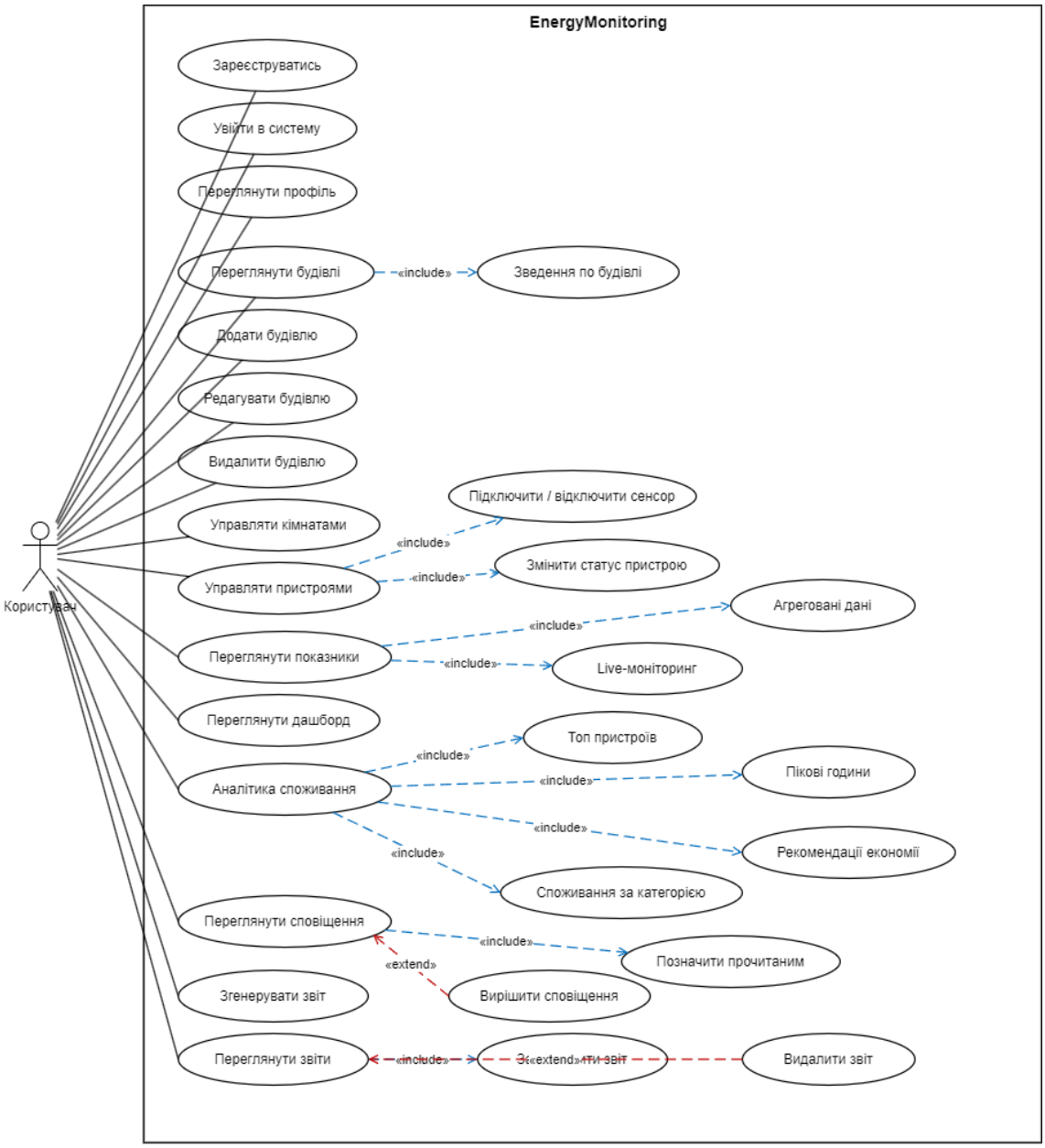

Рисунок А.1 – Діаграма варіантів використання для адміністратора

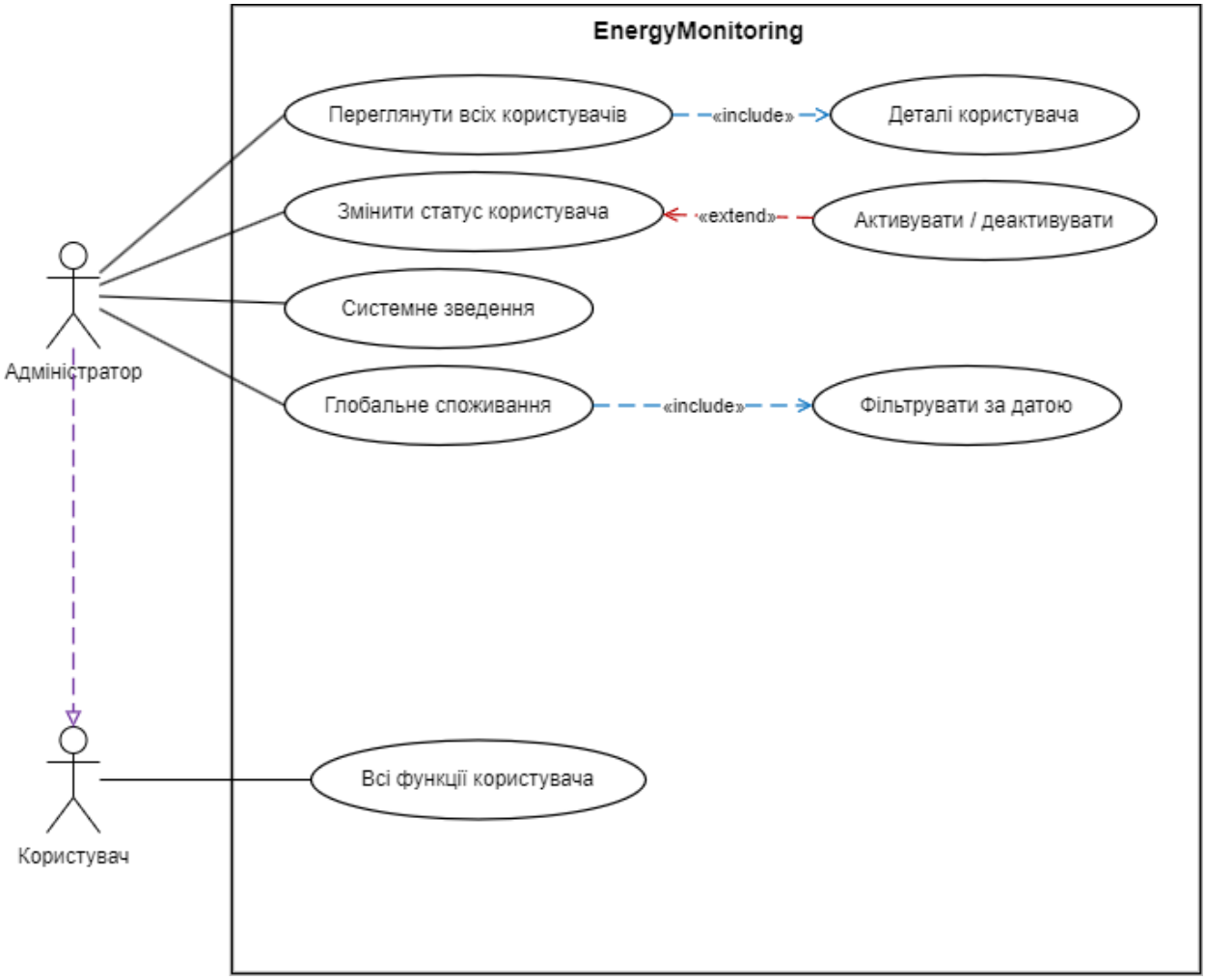

Рисунок А.2 – Діаграма варіантів використання для користувача

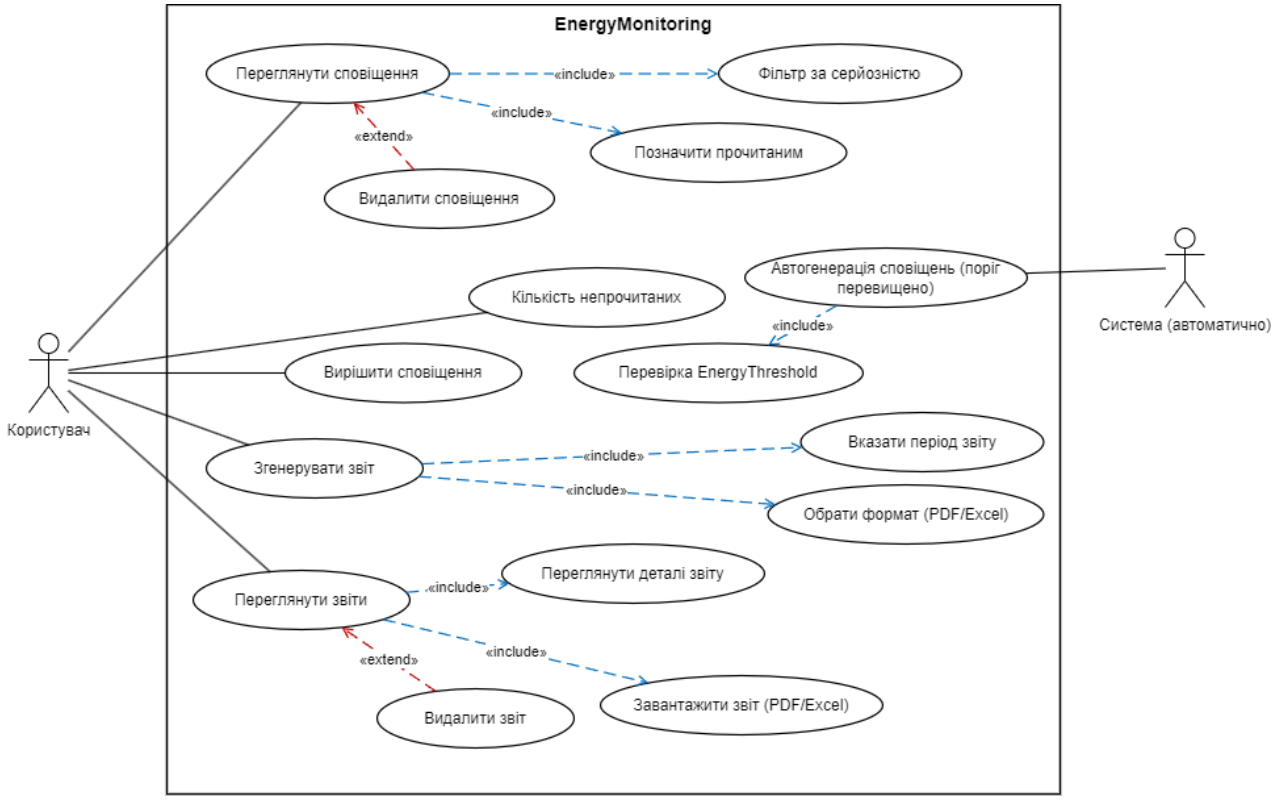

Рисунок А.3 – Діаграма варіантів використання сповіщень та звітів