Міністерство освіти і науки України
Національний університет водного господарства та природокористування
Навчально-науковий інститут кібернетики, інформаційних технологій та інженерії
Кафедра комп'ютерних технологій та економічної кібернетики

Допущено до захисту:
Завідувач кафедри
комп'ютерних технологій та
економічної кібернетики
д. е. н., проф. П. М. Грицюк
_____
«__»_____20__р

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття ступеня «бакалавр»
за освітньо-професійною програмою
«Інформаційні технології в бізнесі»
спеціальності 126 «Інформаційні системи та технології»

на тему**: «Інформаційна система управління розумним домом»**

Виконав:
здобувач освіти 4 курсу,
групи ІСТ-41
Пінкевич Дмитро Сергійович
(прізвище, ім'я, по – батькові)

Керівник:
д.е.н., професор Грицюк П. М.
(науковий ступінь, вчене звання прізвище та ініціали)

Рецензент:
к.е.н., доцент Волошин В. С.
(науковий ступінь, вчене звання прізвище та ініціали)

Рівне – 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет водного господарства та природокористування
Навчально-науковий інститут кібернетики, інформаційних технологій та інженерії
Кафедра комп'ютерних технологій та економічної кібернетики
Освітньо-кваліфікаційний рівень – бакалавр
Освітньо-професійна програма
«Інформаційні технології в бізнесі»
Спеціальність 126 «Інформаційні системи та технології»

ЗАТВЕРДЖУЮ
Завідувач кафедри
комп'ютерних технологій та
економічної кібернетики
д. е. н., проф. П. М. Грицюк
_____
«__»_____20__р

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Пінкевича Дмитра Сергійовича
(прізвище, ім'я, по батькові)

1. Тема роботи: *Інформаційна система управління розумним домом*

керівник роботи: д.е.н., професор Грицюк П. М.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджена наказом по університету від _____

2.Термін здачі студентом закінченої роботи: 08 червня 2026 р.

3. Вихідні дані до роботи: *Технічне завдання на систему, перелік типів пристроїв і метрик телеметрії, вимоги до вебзастосунку та бази даних*

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *аналіз предметної області та аналогів, обґрунтування технологій, проєктування архітектури й бази даних, реалізація та тестування системи*

5. Перелік графічного матеріалу:

- *Діаграма архітектури системи*
- *ER-модель бази даних*
- *Діаграма прецедентів (use-case)*
- *Діаграма потоку телеметрії*
- *Скриншоти інтерфейсу системи*
- *Порівняльна таблиця систем-аналогів*

6. Консультація розділів роботи

	Розділ	Прізвище, ініціали та посада консультанта	Завдання видав (підпис, дата)	Завдання прийняв (підпис, дата)
	Розділ I	Грицюк П. М.		
	Розділ II	Грицюк П. М.		
	Розділ III	Грицюк П. М.		

7. Дата видачі завдання: 19 січня 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Термін виконання етапів роботи	Примітка
	Постановка задачі, загальна характеристика пріоритетних питань	*19.01.26-25.01.26*	
1	*Огляд літератури та пошук даних*	*26.01.26-08.02.26*	
2	*Аналіз предметної області*	*09.02.26–15.02.26*	
3	*Проєктування архітектури та бази даних*	*16.02.26-01.03.26*	
4	*Специфікація вимог до програмного забезпечення*	*02.03.26-22.03.26*	
5	*Програмна реалізація інформаційної системи*	*23.03.26-26.04.26*	
6	*Розгортання, тестування та оптимізація системи*	*27.04.26-17.05.26*	
	Висновки та перспективи подальших досліджень	*18.05.26-24.05.26*	
	Підготовка презентації роботи	*25.05.26-31.05.26*	
П	*Представлення роботи та інших документів до захисту*	*08.06.2026*	

Студент _____
(підпис) (прізвище, та ініціали)

Керівник роботи _____
(підпис) (прізвище, та ініціали)

ЗАЯВА
щодо самостійності виконання випускної кваліфікаційної роботи

Я, _____(ПІБ), студент(ка) _____курсу ___ групи ___ ННІ _____

ЗАЯВЛЯЮ:

моя випускна кваліфікаційна робота на тему «_____ _____», яка надається в екзаменаційну комісію із захисту кваліфікаційних робіт зі спеціальності «_____» для захисту, виконана самостійно і не містить плагіату.

Всі запозичення з друкованих та електронних джерел, у тому числі із захищених раніше випускових кваліфікаційних робіт мають відповідні посилання.

Я не використовував(ла) шахрайські методи маніпуляції з текстом (заміна букв, інтервали, мікропробіли, білі знаки, парафрази та ін.).

Інструменти штучного інтелекту використовував(ла) без порушення академічної доброчесності.

Я ознайомлений(а) з чинним Порядком перевірки навчальних, кваліфікаційних, навчально-методичних та наукових робіт на наявність ознак академічного плагіату в НУВГП та Положенням про академічну доброчесність в НУВГП, за яким виявлення плагіату є підставою для відмови в допуску моєї роботи до захисту та застосування відповідних санкцій (академічної відповідальності).

StrikePlagiarism.com | Національний університет водного господарства та природокористування | Документ прийнятий

Метадані

ДОКУМЕНТ

Заголовок
Кваліфікаційна робота.docx

Автор	Науковий керівник / Експерт	ІД документу
Пінкевич Дмитро Сергійович	**Пінкевич Дмитро Сергійович**	**334253836**

ОРГАНІЗАЦІЯ

Назва організації	підрозділ
National University of Water and Environmental Engineering	**National University of Water and Environmental Engineering**

ЗВІТ

Дата звіту	Дата редагування
6/10/2026	**---**

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

6.57% (6.57%) КП 1

3.37% (3.37%) КЦ

9576	78024
Кількість слів	Кількість символів

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв	0
Інтервали	0
Мікропробіли	0
Білі знаки	0
Парафрази (SmartMarks)	39

Джерела

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

АКТ
перевірки випускної кваліфікаційної роботи
автора _____
на рівень запозичень та можливих маніпуляцій з текстом

Відповідно до даних системи StrikePlagiarism файл «_____»

містить: ____ % коефіцієнта подібності; ____% коефіцієнта цитованості; ____ замінених букв; ____ інтервалів; ____ мікропробілів; ____ білих знаків; ____ % ймовірність використання ШІ.

За результатами перевірки засвідчую, що:
автор кваліфікаційної роботи _____:

- самостійно виконав(ла) кваліфікаційну роботу;
- коректно посилався(лась) на використані інформаційні джерела;
- в роботі відсутні ознаки академічної недоброчесності.

Керівник кваліфікаційної роботи *підпис* _____
(ПІП)

РЕФЕРАТ

Кваліфікаційна робота бакалавра: 56 с., 27 рис., 5 табл., 39 літературних джерел.

Актуальність теми. Розвиток інтернету речей та масове поширення «розумних» побутових пристроїв зробили автоматизацію житла одним із найдинамічніших напрямів інформаційних технологій. Наявні рішення або є закритими пропрієтарними екосистемами з обмеженою кастомізацією, або потребують значної технічної підготовки для розгортання. Це зумовлює актуальність створення прозорої, гнучкої та зручної веборієнтованої інформаційної системи управління розумним домом із підтримкою моніторингу в реальному часі, аналітики та автоматизації.

Об'єктом дослідження є процеси моніторингу та керування пристроями домашньої автоматизації, що охоплюють збір телеметрії, її візуалізацію, оброблення подій і автоматичне реагування.

Предметом дослідження є методи, моделі й програмні засоби побудови веборієнтованої інформаційної системи управління розумним домом.

Метою роботи є розроблення та реалізація веборієнтованої інформаційної системи управління розумним домом, яка забезпечує ієрархічну організацію об'єктів керування, збір і візуалізацію телеметрії в реальному часі, порогові сповіщення, сценарії автоматизації та розмежування доступу за ролями.

У роботі проаналізовано предметну область та наявні аналоги, обґрунтовано вибір технологічного стека, спроєктовано архітектуру системи й модель бази даних з одинадцяти таблиць, реалізовано серверну частину (REST API, рушії емулятора телеметрії, оцінювача алертів і сценаріїв, канал реального часу), клієнтську частину (односторінковий застосунок з аналітикою) та механізми безпеки на основі JWT і рольового доступу.

КЛЮЧОВІ СЛОВА: РОЗУМНИЙ ДІМ, ІНТЕРНЕТ РЕЧЕЙ, IOT, ТЕЛЕМЕТРІЯ, ВЕБЗАСТОСУНОК, REST API, WEBSOCKET, POSTGRESQL, REACT, NODE.JS.

ЗМІСТ

ВСТУП

Актуальність теми. Протягом останнього десятиліття інтернет речей (IoT) перетворився з нішевої технології на масове явище: кількість підключених пристроїв у світі вимірюється десятками мільярдів, а обсяг ринку розумного дому щороку зростає двозначними темпами. Сучасне житло дедалі частіше оснащується датчиками, освітлювальними та кліматичними системами, засобами безпеки, які потребують централізованого керування. Водночас наявні платформи мають істотні обмеження — від залежності пропрієтарних екосистем від хмарних сервісів виробника до високого порога входження відкритих рішень. Це обґрунтовує актуальність розроблення власної веборієнтованої інформаційної системи, яка поєднує прозору архітектуру, зручний інтерфейс, моніторинг у реальному часі та можливість демонстрації функціональності без фізичного обладнання.

Об'єкт дослідження — процеси моніторингу та керування пристроями розумного дому, що охоплюють збір і збереження телеметрії, її візуалізацію, оброблення порогових подій та автоматичне реагування на зміни стану середовища.

Предмет дослідження — методи, моделі та програмні засоби побудови веборієнтованої інформаційної системи управління розумним домом із підтримкою реального часу.

Мета роботи — розробити та реалізувати веборієнтовану інформаційну систему управління розумним домом, що забезпечує ієрархічну організацію об'єктів керування, збір і візуалізацію телеметрії в реальному часі, порогові сповіщення, сценарії автоматизації та розмежування доступу за ролями користувачів.

Для досягнення мети поставлено такі завдання:

- проаналізувати предметну область, концепцію розумного дому та інтернету речей, а також наявні системи-аналоги;
- обґрунтувати вибір архітектурного стилю й технологічного стека реалізації;

- спроєктувати архітектуру системи, рольову модель і модель бази даних;
- реалізувати серверну частину: REST API, рушії емулятора телеметрії, оцінювача алертів і сценаріїв автоматизації, канал зв'язку реального часу;
- реалізувати клієнтську частину: односторінковий вебзастосунок із моніторингом, аналітикою та керуванням пристроями;
- забезпечити безпеку системи на основі автентифікації JWT та рольового розмежування доступу;
- перевірити працездатність системи на демонстраційних даних і сформулювати напрями подальшого розвитку.

Методи дослідження. У роботі використано методи системного аналізу предметної області, об'єктно-орієнтованого та структурного проєктування, методи проєктування реляційних баз даних, а також методи розроблення вебзастосунків на основі клієнт-серверної архітектури.

Практичне значення. Розроблена система може використовуватися як основа для побудови реальних рішень домашньої автоматизації, а також як навчально-демонстраційна платформа завдяки вбудованому емулятору телеметрії, що не потребує фізичних пристроїв.

Структура роботи. Робота складається зі вступу, двох розділів, висновків, списку використаної літератури та додатків. У першому розділі проаналізовано предметну область і обґрунтовано вибір технологій. У другому розділі описано проєктування та реалізацію системи.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ТЕХНОЛОГІЙ СИСТЕМ РОЗУМНОГО ДОМУ

1.1. Концепція розумного дому та інтернету речей: значення й сучасний стан

Стрімкий розвиток інформаційно-комунікаційних технологій та здешевлення мікроелектроніки протягом останнього десятиліття зумовили масове проникнення обчислювальних пристроїв у повсякденне життя людини. Якщо раніше обчислювальна техніка була зосереджена в персональних комп'ютерах і серверах, то сьогодні мережеві можливості отримують побутові прилади, освітлювальні системи, засоби безпеки та інженерні комунікації житла. Сукупність таких технологій формує концепцію розумного дому, яка перетворює традиційне помешкання на кероване інформаційне середовище, здатне адаптуватися до потреб мешканців.

Розумний дім (англ. smart home, home automation) — це житло, оснащене мережею взаємопов'язаних електронних пристроїв, датчиків і виконавчих механізмів, які під керуванням централізованого програмного забезпечення дають змогу автоматизовано контролювати та керувати освітленням, мікрокліматом, енергоспоживанням, безпекою й іншими функціями приміщення. Ключовою ознакою розумного дому є не просто наявність окремих «розумних» пристроїв, а їх інтеграція в єдину систему, що забезпечує спільну роботу, обмін даними та узгоджене реагування на події.

Інтернет речей (англ. Internet of Things, IoT) є технологічним фундаментом розумного дому. Під цим терміном розуміють мережу фізичних об'єктів, оснащених сенсорами, програмним забезпеченням і засобами зв'язку, які збирають та обмінюються даними через мережу Інтернет без безпосередньої участі людини. У контексті житла IoT-пристрої (термостати, лампи, датчики руху, лічильники електроенергії, датчики якості повітря та протікання, розумні замки) виступають джерелами телеметрії й одночасно виконавчими елементами, якими можна керувати дистанційно.

З архітектурного погляду будь-яка система розумного дому містить кілька функціональних складників. Сенсорний рівень утворюють датчики, що вимірюють фізичні параметри середовища (температуру, вологість, освітленість, концентрацію вуглекислого газу, наявність руху тощо). Виконавчий рівень становлять актуатори — пристрої, що змінюють стан середовища. Рівень зв'язку забезпечує передавання даних між пристроями та центром керування. Рівень керування реалізується контролером або програмною платформою, яка приймає телеметрію, застосовує логіку автоматизації та формує керувальні команди. Нарешті, рівень подання відповідає за взаємодію з користувачем через вебінтерфейс або мобільний застосунок. Узагальнену багаторівневу структуру системи наведено на рис. 1.1.

Рис. 1.1. Узагальнена багаторівнева структура системи розумного дому

Обмін даними у розумному домі здійснюється за допомогою різних протоколів зв'язку. Найпоширенішими є Wi-Fi (висока пропускна здатність, але значне енергоспоживання), Bluetooth Low Energy (низьке енергоспоживання, мала дальність), а також спеціалізовані бездротові протоколи Zigbee та Z-Wave, орієнтовані на створення самоорганізовуваних mesh-мереж із малопотужних пристроїв. Окремої уваги заслуговують протокол Thread і галузевий стандарт Matter, який підтримують провідні виробники і який покликаний забезпечити сумісність пристроїв різних брендів. На прикладному рівні для передавання

телеметрії широко застосовують легкий протокол MQTT, що працює за моделлю «видавець — передплатник».

Актуальність тематики розумного дому підтверджується динамікою відповідного ринку. За оцінками аналітичних агенцій, обсяг світового ринку розумного дому у 2024–2025 роках становив від 104 до 156 млрд доларів США, а прогнозовані темпи його зростання сягають 12–27 % на рік залежно від методики оцінювання [1, 2, 3]. Загальна кількість підключених IoT-пристроїв у світі перевищує 25 млрд активних з'єднань і продовжує стрімко зростати [4]. Це свідчить про сталий попит на технології автоматизації житла та про перспективність розроблення відповідного програмного забезпечення.

Функціональне призначення систем розумного дому охоплює моніторинг (безперервний збір і відображення показників), дистанційне керування пристроями, автоматизацію на основі сценаріїв, забезпечення безпеки та підвищення енергоефективності. За способом організації обчислень розрізняють хмарні системи, що покладаються на сервери виробника, та локальні системи, які виконують обробку на власному обладнанні користувача й забезпечують вищу приватність. Незалежно від підходу, критично важливими залишаються питання захисту персональних даних та кібербезпеки, оскільки IoT-пристрої збирають чутливу інформацію про поведінку мешканців.

Системи розумного дому можна класифікувати за кількома ознаками. За способом організації обчислень вони поділяються на хмарні, локальні та гібридні. За ступенем відкритості — на пропрієтарні та з відкритим кодом. За типом інтерфейсу взаємодії — на керовані голосом, через мобільний застосунок або вебінтерфейс. За масштабом — на рішення для окремої квартири, приватного будинку чи багатоквартирного комплексу. Така класифікація (рис. 1.2) допомагає визначити місце розроблюваної системи: це веборієнтоване, відкрите для модифікації рішення з вебінтерфейсом, орієнтоване на рівень окремого помешкання.

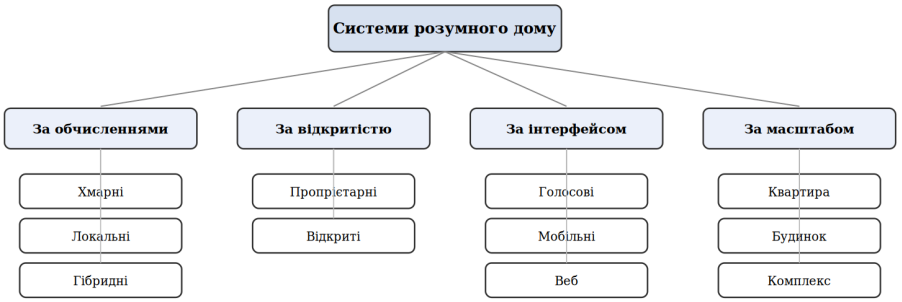

Рис. 1.2. Класифікація систем розумного дому за основними ознаками

Концепція розумного дому пройшла помітний еволюційний шлях. Перші системи домашньої автоматизації, що з'явилися наприкінці XX століття, базувалися на дротових рішеннях і пропрієтарних протоколах, були дорогими та орієнтованими переважно на елітне житло. Поширення бездротових технологій, здешевлення мікроконтролерів і поява смартфонів як універсального пульта керування зробили розумний дім доступним для масового споживача. Сучасний етап характеризується переходом від ізольованих «острівців» автоматизації до інтегрованих платформ, об'єднаних спільними стандартами сумісності.

Важливою тенденцією є зміщення обчислень ближче до джерела даних — так звані граничні обчислення (edge computing), коли частина оброблення телеметрії виконується безпосередньо на контролері або шлюзі, а не в хмарі. Це зменшує затримки, знижує навантаження на канали зв'язку та підвищує приватність. Паралельно зростає роль методів штучного інтелекту й машинного навчання, які застосовують для прогнозування споживання ресурсів, виявлення аномалій у поведінці пристроїв та формування адаптивних сценаріїв, що враховують звички мешканців.

Окремою критично важливою проблемою систем розумного дому є кібербезпека та захист приватності. Оскільки IoT-пристрої збирають чутливі дані про спосіб життя мешканців і отримують фізичний контроль над приміщенням, вони стають привабливою ціллю для зловмисників. До типових загроз належать перехоплення незахищеного трафіку, використання стандартних або слабких паролів, відсутність оновлень мікропрограм та

небезпечне зберігання облікових даних [39]. Тому під час проєктування програмної платформи розумного дому необхідно закладати механізми надійної автентифікації, шифрування каналів передавання даних, розмежування доступу та валідації вхідної інформації.

Отже, розумний дім є складною інформаційною системою, ядром якої виступає програмна платформа керування. Саме вона визначає, наскільки зручно, безпечно й ефективно користувач взаємодіє з пристроями. Тому подальший аналіз доцільно зосередити на наявних програмних рішеннях для управління розумним домом.

1.2. Огляд наявних систем і аналогів управління розумним домом

На сучасному ринку представлено значну кількість програмних платформ для управління розумним домом. Їх умовно поділяють на дві великі групи: пропрієтарні екосистеми великих технологічних компаній і відкриті (open-source) рішення, що розвиваються спільнотою.

Пропрієтарні екосистеми. До найпоширеніших належать платформи Google Home (Nest), Amazon Alexa, Apple HomeKit та Samsung SmartThings. Платформи Google та Amazon будуються навколо голосових асистентів і хмарних сервісів та активно інтегрують технології штучного інтелекту [8]. Apple HomeKit вирізняється підвищеною увагою до приватності: значна частина обробки команд виконується локально, а відеопотік захищається наскрізним шифруванням [10]. Samsung SmartThings позиціонується як найбільш сумісна екосистема завдяки глибокій підтримці стандартів Matter і Thread, що дозволяє об'єднувати пристрої різних брендів [9].

Спільною перевагою пропрієтарних екосистем є простота початкового налаштування, відшліфований інтерфейс і широка підтримка серійних пристроїв. Водночас вони мають істотні недоліки: залежність від хмарної інфраструктури постачальника, закритість програмного коду, що унеможливлює глибоку кастомізацію, та прив'язку до конкретної екосистеми (vendor lock-in).

Поява стандарту Matter частково пом'якшує проблему сумісності, проте його впровадження відбувається повільніше за очікуване [9].

Відкриті рішення. Альтернативою є платформи з відкритим кодом, серед яких найпопулярнішою станом на 2025 рік є Home Assistant. Ця система працює локально на власному обладнанні користувача, підтримує понад тисячу інтеграцій і не потребує платної підписки [5, 6]. Кількість її інсталяцій перевищує два мільйони, а у розробку залучені десятки тисяч учасників [5]. Іншими відомими відкритими платформами є openHAB (написана мовою Java) та легковажна Domoticz [6].

Головними перевагами відкритих рішень є повний контроль користувача над даними, незалежність від хмарних сервісів, відсутність абонентської плати та широкі можливості налаштування. Проте ці переваги досягаються ціною підвищеної складності розгортання й підтримки, що робить поріг входження зависоким для пересічного користувача.

Для наочного порівняння розглянутих груп систем за ключовими критеріями зведемо їх характеристики у таблицю 1.1.

Таблиця 1.1. Порівняння пропрієтарних та відкритих систем управління розумним домом

Критерій	Пропрієтарні екосистеми	Відкриті рішення
Розгортання	Просте, «з коробки»	Складне, потребує адміністрування
Обробка даних	Переважно хмарна	Локальна, на обладнанні користувача
Приватність	Залежить від виробника	Висока, дані не залишають мережу
Кастомізація	Обмежена рамками платформи	Практично необмежена
Вартість	Підписка, прив'язка до брендів	Безкоштовно, потрібне обладнання
Поріг входження	Низький	Високий

Окрім згаданих, на ринку представлені й інші рішення. Платформа ioBroker орієнтована на інтеграцію та зберігання даних із підтримкою численних адаптерів; Domoticz вирізняється низькими вимогами до ресурсів і

простотою; екосистема Tuya Smart надає хмарну платформу для великої кількості бюджетних пристроїв різних виробників. Спільним для відкритих і напіввідкритих рішень є акцент на інтеграції наявного обладнання, тоді як питання прозорої структури даних і навчальної доступності коду залишаються другорядними.

Слід окремо зауважити тенденцію до конвергенції екосистем. Завдяки впровадженню стандарту Matter та мережевого протоколу Thread межі між платформами поступово стираються: сертифікований пристрій теоретично здатний працювати одночасно в кількох екосистемах, а технологія багатоадміністративного доступу (multi-admin) дозволяє керувати одним пристроєм з різних застосунків [8, 9]. Попри це, повна безшовна сумісність ще не досягнута, оскільки підтримка окремих категорій пристроїв (наприклад, камер) у стандарті лишається обмеженою, а застаріле обладнання здебільшого не піддається оновленню.

Варто також враховувати, що розглянуті платформи орієнтовані передусім на інтеграцію наявного серійного обладнання й керування ним, а не на надання прозорого, відкритого для модифікації програмного ядра з гнучкою багатокористувацькою моделлю. Для навчальних, дослідницьких і демонстраційних завдань, а також для випадків, коли потрібен повний контроль над логікою оброблення даних і структурою бази даних, ці рішення виявляються або надлишково складними, або надмірно закритими.

Щоб об'єктивно визначити місце розроблюваної системи серед наявних рішень, доцільно зіставити її з найпоширенішими платформами за критеріями, що є визначальними для поставлених у роботі завдань: відкритість архітектури, спосіб обробки даних (локальний чи хмарний), наявність веборієнтованого інтерфейсу, гнучкість рольової моделі, наявність вбудованої аналітики та можливість демонстрації роботи без фізичного обладнання.

Пропрієтарні екосистеми Google Home та Amazon Alexa забезпечують зручне голосове керування й широку підтримку серійних пристроїв, проте є закритими,

повністю залежними від хмари виробника та не передбачають прозорого доступу до структури даних чи розвиненої багатокористувацької адміністративної моделі [8]. Apple HomeKit надає високий рівень приватності завдяки локальній обробці команд, але прив'язана до екосистеми Apple і так само закрита для модифікації [10]. Samsung SmartThings вирізняється найкращою сумісністю пристроїв різних виробників завдяки підтримці стандартів Matter і Thread, однак залишається пропрієтарною платформою з обмеженими можливостями кастомізації [9].

Відкриті платформи Home Assistant та openHAB, навпаки, надають користувачеві повний контроль, локальну обробку даних, підтримують велику кількість інтеграцій і не потребують абонентської плати [5, 6]. Проте вони орієнтовані насамперед на інтеграцію наявного фізичного обладнання та керування ним, мають високий поріг входження й не містять вбудованого засобу демонстрації роботи за відсутності реальних пристроїв.

Розроблювана система «Smart Home» свідомо посідає окрему нішу. Вона є відкритою для модифікації веборієнтованою платформою з прозорою структурою даних, дворівневою рольовою моделлю «адміністратор — клієнт», вбудованою аналітикою з кількома типами візуалізації та емулятором телеметрії, що дозволяє відтворити повний цикл роботи без під'єднання фізичного обладнання. На відміну від масових пропрієтарних екосистем, вона не прив'язана до конкретного виробника й не залежить від хмарних сервісів сторонніх компаній, а на відміну від відкритих платформ загального призначення — простіша для опанування й орієнтована на навчально-демонстраційне та дослідницьке застосування.

Аналіз наявних рішень засвідчує наявність ніші: жодна з груп не пропонує водночас прозорої, легкої для опанування архітектури з гнучкою рольовою моделлю та можливістю демонстрації повного циклу роботи без реального обладнання. Це обґрунтовує доцільність розроблення власної веборієнтованої інформаційної системи, яка реалізує повну ієрархію об'єктів

керування, збір і візуалізацію телеметрії в реальному часі, правила сповіщень, сценарії автоматизації, вбудований емулятор пристроїв та чітке розмежування ролей адміністратора й клієнта.

1.3. Обґрунтування вибору технологій та архітектурного стилю реалізації

Якість, масштабованість і підтримуваність системи значною мірою визначаються обґрунтованістю вибору технологій. З огляду на сформульовані вимоги, обрано клієнт-серверну архітектуру з поділом на односторінковий вебзастосунок на боці клієнта та сервер застосунків, що надає програмний інтерфейс REST і канал двостороннього зв'язку на основі технології WebSocket. Доступ до бази даних опосередковується об'єктно-реляційним відображенням.

Клієнтська частина. Для реалізації інтерфейсу обрано бібліотеку React у поєднанні з мовою TypeScript і складальним інструментом Vite. React забезпечує компонентний підхід і має одну з найбільших екосистем; TypeScript додає статичну типізацію; Vite забезпечує швидке складання й миттєве оновлення модулів. Для оформлення використано утилітарний CSS-фреймворк TailwindCSS, а для побудови графіків — бібліотеку ECharts.

Серверна частина. Сервер реалізовано на платформі Node.js з вебфреймворком Express та мовою TypeScript. Вибір Node.js зумовлений можливістю використання єдиної мови на клієнті й сервері та подієво-орієнтованою неблокувальною моделлю введення-виведення, ефективною для застосунків реального часу. Express є мінімалістичним і гнучким фреймворком для побудови маршрутів і проміжного програмного забезпечення.

База даних. Обрано реляційну СКБД PostgreSQL з огляду на структурований та ієрархічний характер предметної області, важливість цілісності даних, підтримку зовнішніх ключів із каскадними операціями й транзакційність. Тип даних JSONB дає змогу гнучко зберігати параметри сценаріїв, поєднуючи переваги реляційної та документоорієнтованої моделей.

Об'єктно-реляційне відображення. Взаємодію з базою даних організовано за допомогою ORM Prisma, що забезпечує типобезпечні запити, автоматичну генерацію типів за схемою та вбудований механізм міграцій, завдяки чому опис схеми стає єдиним джерелом істини.

Зв'язок у реальному часі. Для оновлення інтерфейсу без повторних запитів застосовано технологію WebSocket через бібліотеку Socket.IO, що забезпечує постійний двосторонній канал. Ізоляція подій між користувачами реалізується через механізм кімнат.

Автентифікація та безпека. Безпеку доступу побудовано на основі веб-токенів JWT із поділом на токени доступу та оновлення, а паролі зберігаються у вигляді bcrypt-хешів. Це забезпечує захищену автентифікацію без зберігання стану сесії та рольове розмежування доступу.

Узагальнення обраного технологічного стека наведено у таблиці 1.2.

Таблиця 1.2. Технологічний стек розроблюваної системи

Шар	Технологія	Призначення
Клієнт	React, TypeScript, Vite, TailwindCSS, ECharts	SPA-інтерфейс, візуалізація, аналітика
Сервер	Node.js, Express, TypeScript	REST API, бізнес-логіка
Реальний час	Socket.IO (WebSocket)	Двосторонній обмін, live-оновлення
Доступ до даних	Prisma ORM	Типобезпечні запити, міграції
База даних	PostgreSQL	Зберігання, цілісність, транзакції
Безпека	JWT, bcrypt	Автентифікація, рольовий доступ

Вибір кожної технології здійснювався після порівняння з поширеними альтернативами. Серед бібліотек і фреймворків для побудови інтерфейсу, окрім React, розглядалися Angular та Vue.js [31, 32]. Angular є повноцінним фреймворком зі значним обсягом готових рішень, проте має вищий поріг входження й більшу «вагу»; Vue.js простіший, але має меншу екосистему. React обрано як компроміс між гнучкістю, розміром спільноти та доступністю компонентних бібліотек, що особливо важливо для побудови насиченого аналітичного інтерфейсу.

Для серверної частини альтернативами Node.js були екосистеми Python (фреймворк Django) та Java (Spring Framework) [34, 35]. Django пропонує швидку розробку та вбудовану адміністративну панель, а Spring — високу надійність для корпоративних систем, проте обидва передбачають іншу мову програмування на сервері. Перевагу віддано Node.js саме через єдиний мовний простір (TypeScript) на клієнті й сервері та природну придатність подієвої моделі для застосунків реального часу з інтенсивним введенням-виведенням.

Під час вибору сховища даних реляційну СКБД PostgreSQL порівнювали з MySQL та документоорієнтованою MongoDB [36, 37]. MongoDB зручна для слабоструктурованих даних, проте поступається у підтримці складних зв'язків і транзакційної цілісності, критичних для ієрархічної моделі розумного дому. MySQL близька за можливостями, але PostgreSQL вирізняється потужнішою підтримкою типу JSONB і розширених індексів. Для організації доступу до даних замість ORM Prisma розглядалися TypeORM і Sequelize; перевагу надано Prisma завдяки декларативній схемі, автоматичній генерації типів і зручному механізму міграцій.

Нарешті, для організації програмного інтерфейсу архітектурний стиль REST порівнювали з підходом GraphQL [38]. GraphQL дає клієнту гнучкість у формуванні запитів, проте ускладнює кешування й моніторинг та є надлишковим для відносно стабільного набору ресурсів цієї системи. Тому обрано REST як простіший, добре стандартизований і достатній для поставлених завдань підхід, доповнений каналом WebSocket для подій реального часу.

Таким чином, обраний технологічний стек і клієнт-серверна архітектура з підтримкою реального часу повною мірою відповідають сформульованим загальним вимогам до системи.

1.4. Аналіз та формування вимог до системи

На підставі проведеного аналізу предметної області та наявних рішень сформульовано вимоги до розроблюваної інформаційної системи. Їх поділено

на функціональні, що описують поведінку системи та її можливості, і нефункціональні, що визначають якісні характеристики, зокрема продуктивність, безпеку, зручність використання й масштабованість відповідно до моделі якості програмного забезпечення [38].

Користувачі системи (актори). Виокремлено двох основних акторів. Адміністратор відповідає за керування обліковими записами клієнтів, перегляд статистики та, за потреби, вхід від імені клієнта для діагностики. Клієнт (користувач) є власником домену розумного дому й працює з будинками, кімнатами, пристроями, телеметрією, аналітикою, сценаріями та сповіщеннями. Розмежування цих ролей є однією з ключових вимог. Взаємодію акторів із системою відображає діаграма прецедентів на рис. 1.3.

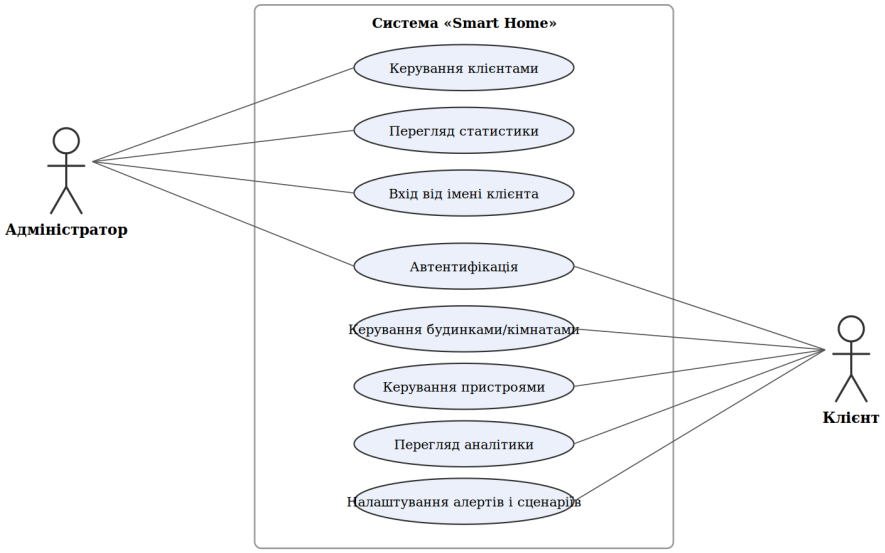

Рис. 1.3. Діаграма прецедентів (use-case) системи

Перелік основних функціональних вимог наведено у таблиці 1.3.

Таблиця 1.3. Функціональні вимоги до системи

Код	Вимога	Опис
ФВ-1	Автентифікація та ролі	Реєстрація, вхід, розмежування доступу адміністратора й клієнта
ФВ-2	Керування об'єктами	CRUD-операції над будинками, кімнатами та пристроями
ФВ-3	Збір телеметрії	Отримання та збереження показників пристроїв

Код	Вимога	Опис
ФВ-4	Моніторинг у реальному часі	Оновлення показників та подій без перезавантаження сторінки
ФВ-5	Аналітика	Агрегація даних і побудова графіків різних типів
ФВ-6	Порогові сповіщення	Створення правил і автоматичне відстеження їх спрацювання
ФВ-7	Сценарії автоматизації	Тригери за часом, сенсором або вручну та виконання дій
ФВ-8	Адміністрування	Перегляд клієнтів, статистики та керування обліковими записами

Нефункціональні вимоги. До системи висунуто такі якісні вимоги: продуктивність — оновлення показників у реальному часі з мінімальною затримкою; безпека — хешування паролів, токенна автентифікація, перевірка належності даних і валідація введення; зручність використання — адаптивний інтерфейс із підтримкою мобільних пристроїв та кастомізацією теми; масштабованість — модульна архітектура, що допускає розширення набору пристроїв і функцій; надійність — стійкість сервісів до помилок та збереження цілісності даних. Сформульовані вимоги стали основою для проєктування та реалізації системи, що розглядаються у другому розділі.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ «SMART HOME»

2.1. Архітектура системи та модель даних

Система побудована за трирівневою клієнт-серверною архітектурою. Перший рівень — клієнтський односторінковий вебзастосунок, що виконується у браузері й відповідає за подання даних та взаємодію з користувачем. Другий рівень — сервер застосунків на платформі Node.js, який реалізує бізнес-логіку, надає програмний інтерфейс REST і канал реального часу. Третій рівень — реляційна база даних PostgreSQL, доступ до якої здійснюється через ORM Prisma. Загальну схему взаємодії компонентів наведено на рис. 2.1.

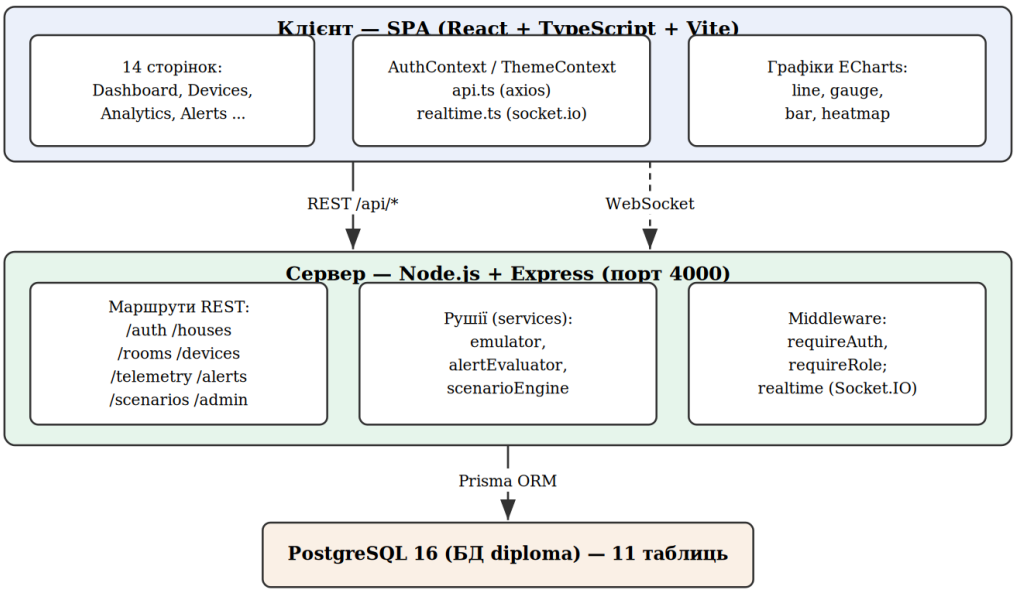

Рис. 2.1. Архітектура інформаційної системи «Smart Home»

Взаємодія клієнта із сервером відбувається двома каналами: синхронні запити виконуються через REST API (за допомогою HTTP-клієнта axios), а оновлення в реальному часі надходять через постійне WebSocket-з'єднання. На етапі розробки інструмент Vite виконує функцію проксі, переспрямовуючи запити клієнта на сервер.

Рольова модель. Система реалізує дворівневу рольову модель. Роль адміністратора створюється автоматично під час першого запуску й призначена для керування обліковими записами клієнтів, перегляду статистики та входу від

імені клієнта (impersonation). Роль клієнта (користувача) реєструється самостійно й володіє повним доменом розумного дому: будинками, кімнатами, пристроями, телеметрією, аналітикою, сценаріями й алертами. Адміністратор навмисно не має доступу до доменних ресурсів клієнтів, що відповідає принципу розмежування обов'язків.

Модель даних. База даних містить одинадцять таблиць, пов'язаних відношеннями зовнішніх ключів. Основу становить ієрархічний ланцюг володіння «користувач — будинок — кімната — пристрій — телеметрія», для якого налаштовано каскадне видалення: вилучення користувача автоматично прибирає всі підпорядковані записи. Перелік таблиць наведено у таблиці 2.1.

Таблиця 2.1. Перелік таблиць бази даних

№	Таблиця	Призначення
1	users	Користувачі та ролі (admin/user), тема інтерфейсу, аватар
2	houses	Будинки, що належать користувачу
3	rooms	Кімнати в будинку, план кімнати
4	devices	Пристрої в кімнаті, тип, статус, координати на плані
5	telemetry	Сирі вимірювання показників пристроїв
6	telemetry_aggregated	Погодинні та подобові агрегати показників
7	device_logs	Історія керування пристроями
8	scenarios	Сценарії автоматизації (тригери, дії)
9	alerts	Правила порогових сповіщень (рівень будинку)
10	alert_events	Події перетину порогів (відкриті/закриті)
11	notifications	Сповіщення від дій сценаріїв

Центральною сутністю моделі є користувач (таблиця users), який, окрім облікових даних і ролі, зберігає індивідуальні налаштування інтерфейсу — обрану тему та посилання на аватар. Користувачу належать будинки (houses), кожен з яких містить кімнати (rooms), а кімнати — пристрої (devices). Пристрій характеризується типом, поточним статусом, ознакою доступності та нормалізованими координатами розташування на плані кімнати. Виміряні показники зберігаються у таблиці telemetry, де кожен запис містить тип метрики, значення з фіксованою точністю, одиницю вимірювання та позначку часу.

Допоміжні таблиці розширюють базову модель. Таблиця telemetry_aggregated зберігає погодинні та подобові агрегати показників, що пришвидшує побудову аналітичних графіків за тривалі періоди. Таблиця device_logs фіксує історію керування пристроями, а scenarios описує сценарії автоматизації, де параметри тригерів і перелік дій зберігаються у форматі JSONB, що забезпечує гнучкість без зміни схеми. Правила порогових сповіщень містяться у таблиці alerts на рівні будинку, а власне факти перетину порогів — у таблиці alert_events, яка зберігає значення першого спрацювання, останнє зафіксоване значення, час відкриття та закриття події й причину закриття. Сповіщення від дій сценаріїв накопичуються у таблиці notifications.

Цілісність даних забезпечується системою зовнішніх ключів. Уздовж ланцюга володіння застосовано каскадне видалення, тоді як для таблиць device_logs і notifications використано стратегію встановлення порожнього значення (SET NULL) під час видалення пов'язаного об'єкта, що дозволяє зберегти історичні записи. Схема бази даних розвивалася ітеративно й налічує шість міграцій: початкове створення схеми, додавання планів кімнат і координат пристроїв, вилучення типу пристрою «камера», додавання таблиці подій алертів, аватара користувача та збереження теми оформлення. Повний опис схеми у вигляді декларації Prisma наведено в Додатку А.

Для опису предметної області застосовано систему перелічень (enum), що забезпечує контроль допустимих значень на рівні бази даних: типи пристроїв (термостат, лампа, датчик руху, лічильник, датчик якості повітря, датчик протікання, розумний замок), типи метрик (температура, вологість, потужність, рух, рівень CO_2, освітленість, протікання), умови спрацювання алертів (більше, менше, дорівнює тощо), типи тригерів сценаріїв (час, сенсор, ручний) та типи сповіщень. Для прискорення типових запитів створено індекси, зокрема складений індекс за полями «пристрій — час» на таблиці телеметрії, що є гарячим шляхом для вибірки останніх значень. Логічну структуру бази даних та зв'язки між таблицями ілюструє рис. 2.2.

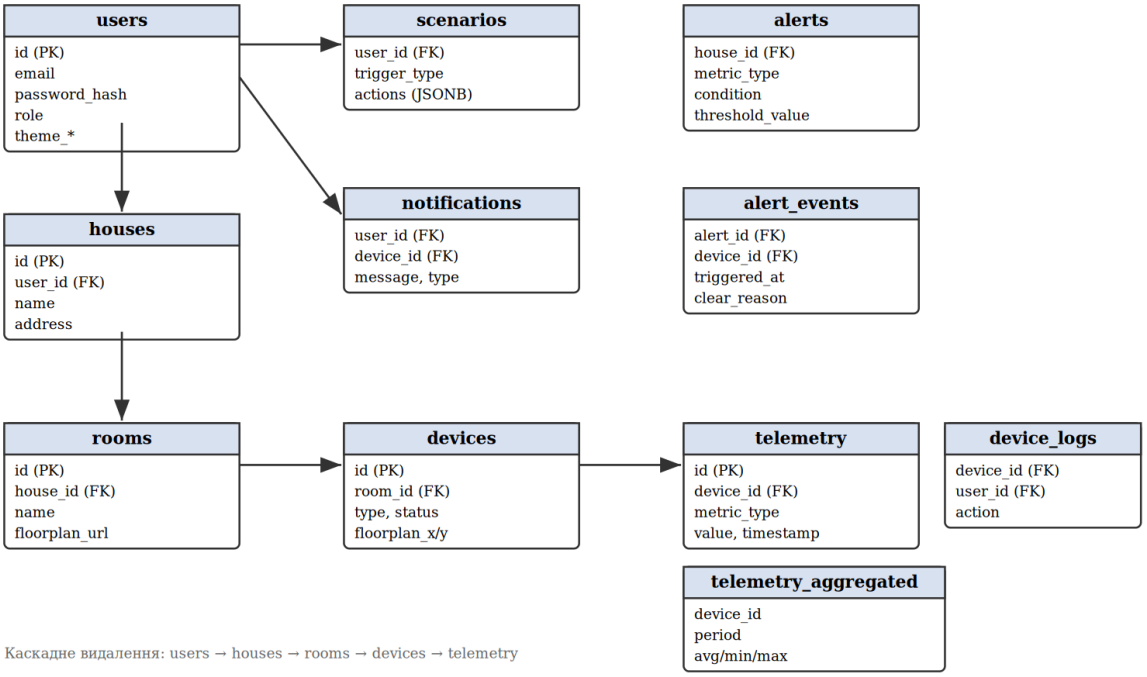

Рис. 2.2. Логічна модель (ER-діаграма) бази даних системи

Потоки даних. Ключовим у системі є потік телеметрії, що працює в реальному часі. Емулятор з визначеним інтервалом генерує нові значення для пристроїв користувача; згенеровані дані пакетно записуються до таблиці телеметрії; після запису викликається оцінювач алертів, який перевіряє пороги й за потреби відкриває або закриває події; далі спрацьовує перевірка сенсорних сценаріїв; нарешті сервер надсилає клієнту WebSocket-подію, і інтерфейс оновлює графіки без повторного запиту. Послідовність оброблення показано на рис. 2.3.

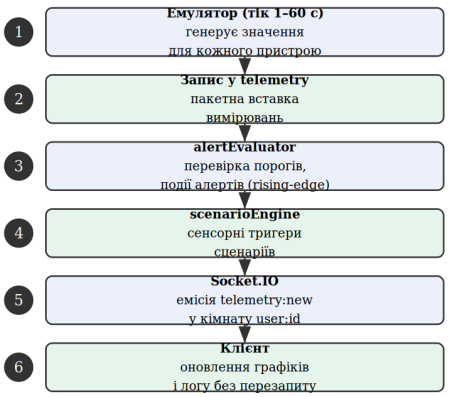

Рис. 2.3. Потік оброблення телеметрії в реальному часі

2.2. Реалізація серверної частини

Серверну частину організовано за модульним принципом. Точка входу налаштовує застосунок Express, монтує маршрути, ініціалізує канал реального часу та забезпечує коректне завершення роботи. Доступ до бази даних інкапсульовано в єдиному екземплярі клієнта Prisma. Окремі модулі відповідають за автентифікацію (генерація й перевірка токенів, проміжне програмне забезпечення), завантаження файлів та три рушії бізнес-логіки.

REST API. Програмний інтерфейс згруповано за ресурсами: автентифікація, будинки, кімнати, пристрої, телеметрія, алерти, сценарії, емулятор та адміністрування. Усі доменні маршрути захищені перевіркою автентифікації та належності ресурсу поточному користувачу. Основні групи ендпоінтів наведено у таблиці 2.2.

Таблиця 2.2. Основні групи ендпоінтів REST API

Група	Доступ	Призначення
/auth	усі / користувач	Реєстрація, вхід, оновлення токена, профіль, тема, аватар
/houses, /rooms, /devices	користувач	CRUD-операції над ієрархією об'єктів
/telemetry	користувач	Лог телеметрії з фільтрами, пакетна вставка
/alerts	користувач	Правила, активні події, історія, зняття алертів
/scenarios	користувач	CRUD сценаріїв, ручний запуск
/emulator	користувач	Запуск, зупинка та статус емулятора
/admin	адміністратор	Список клієнтів, деталі, видалення, impersonation

Уся вхідна валідація реалізована за допомогою бібліотеки Zod, яка перевіряє структуру й обмеження вхідних даних (довжину рядків, типи, обов'язковість полів). Для кожного запиту виконується перевірка належності ресурсу: дані фільтруються за ідентифікатором поточного користувача, тому клієнт отримує доступ виключно до власних об'єктів.

Рушій емулятора телеметрії є внутрішнім сервісом, що працює окремо для кожного користувача й підтримує таблицю таймерів з інтервалом від 1 до 60

секунд. На кожному кроці він завантажує пристрої користувача, генерує наступне значення за допомогою реалістичних генераторів, специфічних для типу пристрою (наприклад, для термостата — випадкове блукання з обмеженням діапазону, для датчика руху — подія з певною ймовірністю), пакетно записує дані до бази, після чого викликає оцінювач алертів і перевірку сенсорних сценаріїв та надсилає WebSocket-подію. Сервіс стійкий до помилок бази даних і самостійно очищає стан видалених пристроїв.

Оцінювач алертів викликається після кожного запису телеметрії. Він дедуплікує показники до останнього для кожної пари «пристрій — метрика» та для кожного активного правила обчислює виконання умови порівняння значення з порогом. Застосовано логіку фронту наростання (rising-edge): подію алерту створюють лише в момент першого перетину порога; доки умова виконується, оновлюється остання зафіксована величина; коли умова перестає виконуватися, подію автоматично закривають. Користувач також може закрити подію вручну.

Рушій сценаріїв виконує масив дій сценарію — зміну статусу пристрою (із записом до журналу керування) та створення сповіщення. Підтримуються три типи тригерів: часові (планувальник з кроком в одну хвилину), сенсорні (спрацьовують за логікою фронту наростання, коли умова стала істинною) та ручні (запуск користувачем). Такий поділ дозволяє гнучко описувати правила автоматизації.

Реалістичність згенерованих даних забезпечується окремими генераторами для кожного типу пристрою. Для термостата значення температури формується методом випадкового блукання з невеликим кроком та обмеженням у межах комфортного діапазону; для лампи рівень освітленості коливається навколо базового значення; лічильник електроенергії за вимкненого стану дає близьке до нуля споживання, а за ввімкненого — поступово зростаюче значення з імовірнісними стрибками навантаження; датчик якості повітря моделює поступове накопичення вуглекислого газу з періодичним різким падінням, що імітує провітрювання. Бінарні пристрої — датчик руху,

розумний замок і датчик протікання — генерують подію з певною, специфічною для кожного типу ймовірністю. Такий підхід дозволяє демонструвати правдоподібну поведінку системи та коректно перевіряти спрацювання алертів і сценаріїв.

Канал реального часу реалізовано на основі Socket.IO поверх того самого HTTP-сервера. Автентифікація виконується на етапі рукостискання: клієнт передає токен доступу, який перевіряється сервером, після чого з'єднання приєднується до персональної кімнати користувача. Це гарантує, що події телеметрії та алертів отримує лише їх власник.

Безпека. Паролі зберігаються у вигляді bcrypt-хешів. Автентифікація побудована на парі токенів JWT: короткоживучому токені доступу (15 хвилин) та довготривалому токені оновлення (7 днів). Проміжне програмне забезпечення перевіряє наявність і дійсність токена та відповідність ролі, повертаючи помилку доступу в разі невідповідності. Завантаження файлів (аватари, плани кімнат) обмежене за розміром і типом.

Обробка помилок і журналювання. Серверні обробники запитів побудовано так, щоб помилки не призводили до аварійного завершення процесу: винятки перехоплюються, а клієнту повертається коректний код стану HTTP разом із описом проблеми. Для структурованого журналювання застосовано бібліотеку pino, яка фіксує події роботи сервера у машинозчитуваному форматі, що спрощує діагностику. Рушії реального часу спроєктовано стійкими до тимчасових збоїв бази даних: у разі помилки запис відповідної ітерації пропускається, а робота сервісу продовжується.

Завантаження файлів. Завантаження зображень (аватарів користувачів і планів кімнат) реалізовано за допомогою проміжного програмного забезпечення multer. Встановлено обмеження на розмір файлів та перелік дозволених типів (PNG, JPEG, WebP), а збережені файли віддаються як статичні ресурси за непередбачуваними іменами, що знижує ризик несанкціонованого доступу. Така перевірка типу й розміру є частиною загальної політики валідації вхідних даних.

Ініціалізація та узгодженість стану. Під час запуску сервера виконуються ідемпотентні процедури початкового наповнення: створення облікового запису адміністратора та демонстраційного клієнта разом із набором показових даних. Ідемпотентність означає, що повторний запуск не призводить до дублювання записів. Життєвий цикл автентифікації побудовано так: після входу клієнт отримує пару токенів, зберігає їх та супроводжує кожен захищений запит заголовком авторизації; у разі завершення строку дії токена доступу клієнт прозоро отримує новий за допомогою токена оновлення, не перериваючи роботу користувача. Завершення роботи сервера виконується коректно, із зупиненням таймерів емулятора та закриттям з'єднань.

2.3. Реалізація клієнтської частини та функціональні можливості

Клієнтську частину реалізовано як односторінковий застосунок із клієнтською маршрутизацією, що налічує чотирнадцять сторінок: вхід і реєстрація, головна панель, сторінки будинків, кімнат і пристроїв, аналітика, лог телеметрії, сценарії, алерти, налаштування та адміністративні сторінки. Доступ до маршрутів контролюється компонентами захисту, які перевіряють автентифікацію та роль користувача.

Керування станом. Глобальний стан автентифікації зберігається у відповідному контексті: токени розміщуються у локальному сховищі браузера, а перехоплювач HTTP-запитів автоматично оновлює токен доступу в разі його завершення, дедуплікуючи одночасні спроби оновлення. Окремий контекст відповідає за тему оформлення, яка зберігається індивідуально для кожного користувача. Типізований клієнт API інкапсулює всі звернення до сервера разом із доменними типами даних.

Структура сторінок. Головна панель (Dashboard) надає зведення стану домену: кількість пристроїв, активні сповіщення та швидкі дії. Сторінки будинків, кімнат і пристроїв реалізують повний цикл операцій створення, перегляду, редагування та видалення з каскадним вибором у формах (спершу обирається будинок, потім кімната). Форма кімнати дозволяє завантажити

зображення плану кімнати, а форма пристрою — обрати кімнату й тип. Окремі сторінки відведено логу телеметрії з фільтрами, сценаріям, сповіщенням і налаштуванням.

Візуалізація показників. Для відображення даних застосовано бібліотеку ECharts. Показники подаються лінійним графіком, який доповнюється новими точками в міру надходження телеметрії; для бінарних метрик (рух, протікання) використовується ступеневе подання на осі станів із підписами. Сторінка аналітики надає чотири типи візуалізацій: лінійний графік динаміки, шкальний індикатор із кольоровими зонами для поточного значення, стовпчикову діаграму з розподілом за днями та теплову карту у координатах «години — дні тижня». Користувач може обирати функцію агрегації (середнє, сума, мінімум, максимум, кількість) та інтервал.

Кастомізація інтерфейсу. Система тем налічує п'ять акцентних палітр (синю, смарагдову, фіолетову, бурштинову та рожеву), які визначають колір кнопок, посилань, активних елементів меню й графіків, а також дев'ять основних палітр (п'ять темних і чотири світлих), що задають тло бічної та верхньої панелей. Налаштування зберігаються індивідуально для кожного користувача: під час входу вони підтягуються з облікового запису, а локальне сховище браузера використовується як кеш для першого відображення. Це забезпечує персоналізований і послідовний вигляд інтерфейсу між сеансами.

Адаптивність. Інтерфейс адаптовано до мобільних пристроїв: на вузьких екранах бічна панель навігації перетворюється на висувне меню із затемненням тла, а верхня панель згортає допоміжні елементи, зберігаючи доступ до основних функцій. Це дозволяє повноцінно користуватися системою як на настільному комп'ютері, так і зі смартфона.

Адміністрування та вхід від імені клієнта. Адміністративна частина відображає список клієнтів зі зведеною статистикою та сторінку деталей конкретного клієнта. Функція входу від імені клієнта (impersonation) тимчасово надає адміністратору токени обраного користувача для діагностики; при цьому оригінальні токени адміністратора зберігаються у резервному слоті, а інтерфейс

показує банер швидкого повернення до адміністративного режиму. Така реалізація поєднує зручність підтримки з дотриманням принципу розмежування доступу.

Розроблена система реалізує повний набір запланованої функціональності: автентифікацію та ролі, ієрархічні CRUD-операції, плани кімнат із розміщенням пристроїв, телеметрію, аналітику, порогові алерти з історією, сценарії автоматизації, емулятор, оновлення в реальному часі, кастомізацію теми та адміністративну панель.

2.4. Тестування та демонстрація роботи системи

Перевірку працездатності системи проведено на демонстраційному наборі даних, що автоматично створюється під час першого запуску й містить будинок «Квартира на Подолі» з двома кімнатами, п'ятьма пристроями різних типів, трьома правилами порогових сповіщень і чотирма сценаріями автоматизації. Такий набір дозволяє відтворити повний цикл роботи без під'єднання фізичного обладнання.

Тестування виконувалося за основними сценаріями використання. Перевірено реєстрацію та вхід користувача, розмежування доступу між ролями адміністратора й клієнта, створення та редагування об'єктів ієрархії, запуск емулятора та надходження телеметрії, коректність побудови аналітичних графіків, спрацювання й автоматичне закриття подій алертів, виконання сценаріїв за різними типами тригерів, а також функцію входу адміністратора від імені клієнта. За результатами перевірки підтверджено коректність обліку показників, своєчасне відкриття й закриття подій сповіщень та оновлення інтерфейсу в реальному часі без перезавантаження сторінки.

Процес автентифікації представлено сторінками входу та самостійної реєстрації нового користувача (рис. 2.4, 2.5).

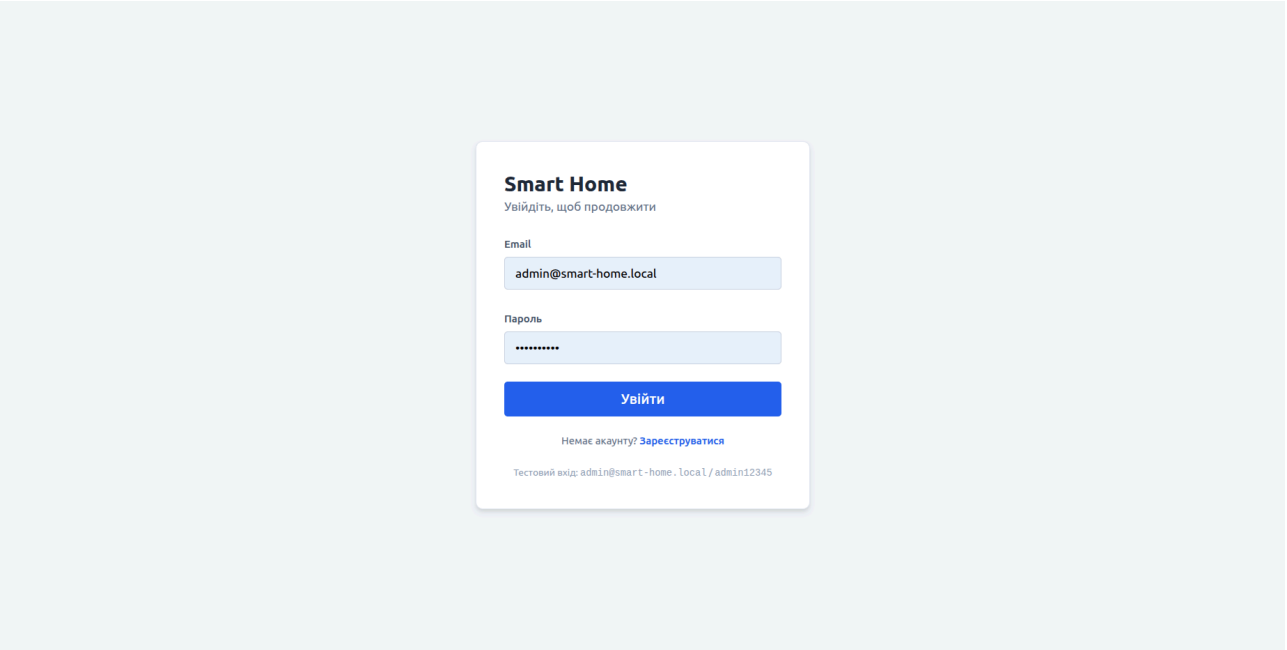

Рис. 2.4. Сторінка входу до системи

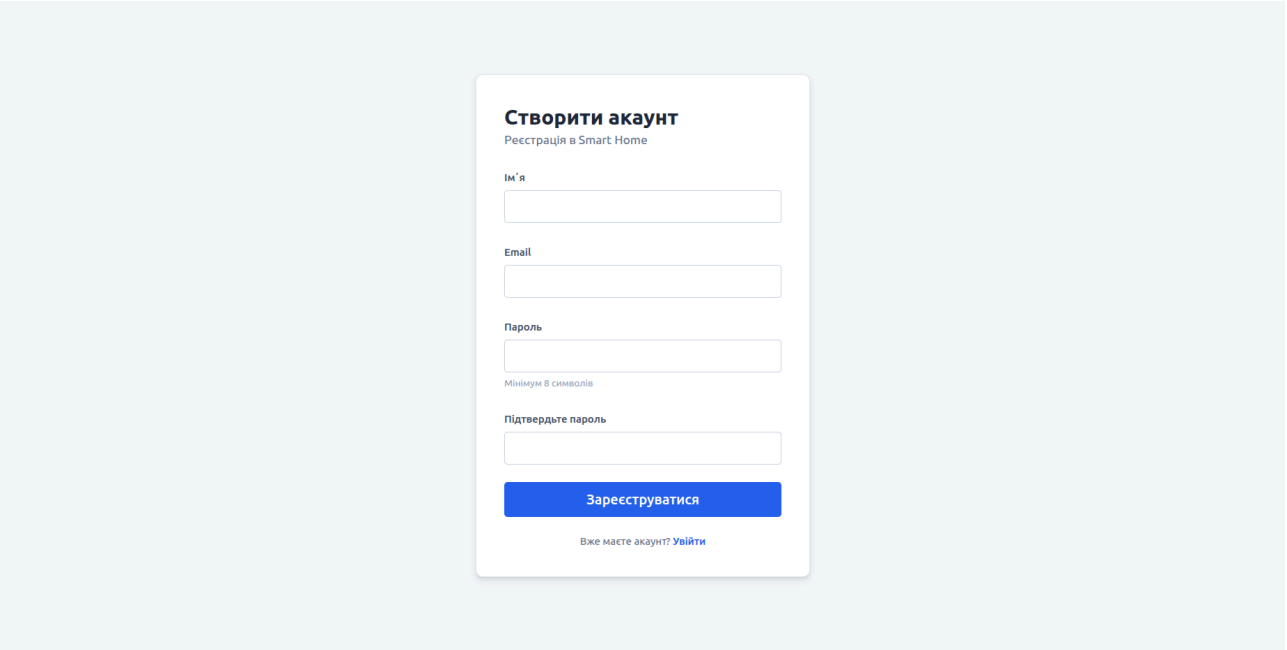

Рис. 2.5. Сторінка реєстрації нового користувача

Після авторизації клієнт потрапляє на головну панель (рис. 2.6), що відображає зведену статистику домену (кількість будинків, кімнат, пристроїв та активних алертів) і поточний стан пристроїв у розрізі кімнат.

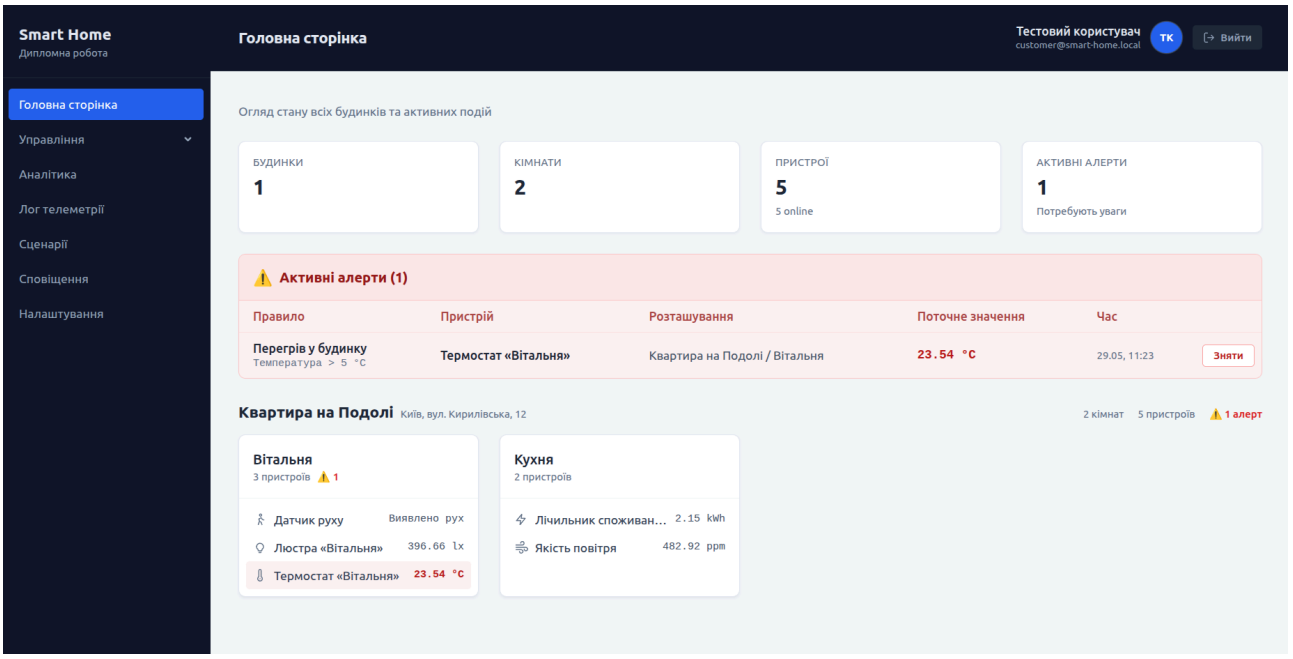

Рис. 2.6. Головна панель (Dashboard) клієнта

Керування ієрархією об'єктів реалізовано на окремих сторінках будинків, кімнат і пристроїв із модальними формами створення (рис. 2.7–2.9). Форма кімнати містить поле завантаження плану, а форма пристрою — каскадний вибір будинку й кімнати та вибір типу пристрою.

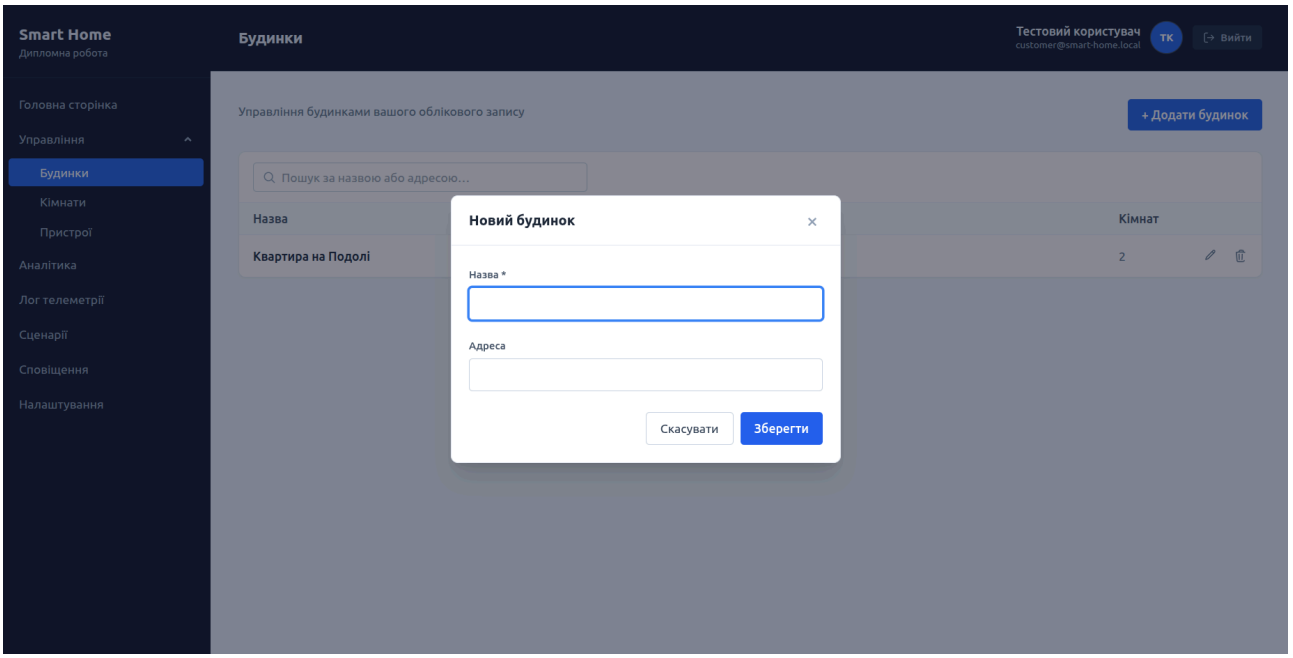

Рис. 2.7. Керування будинками та форма додавання будинку

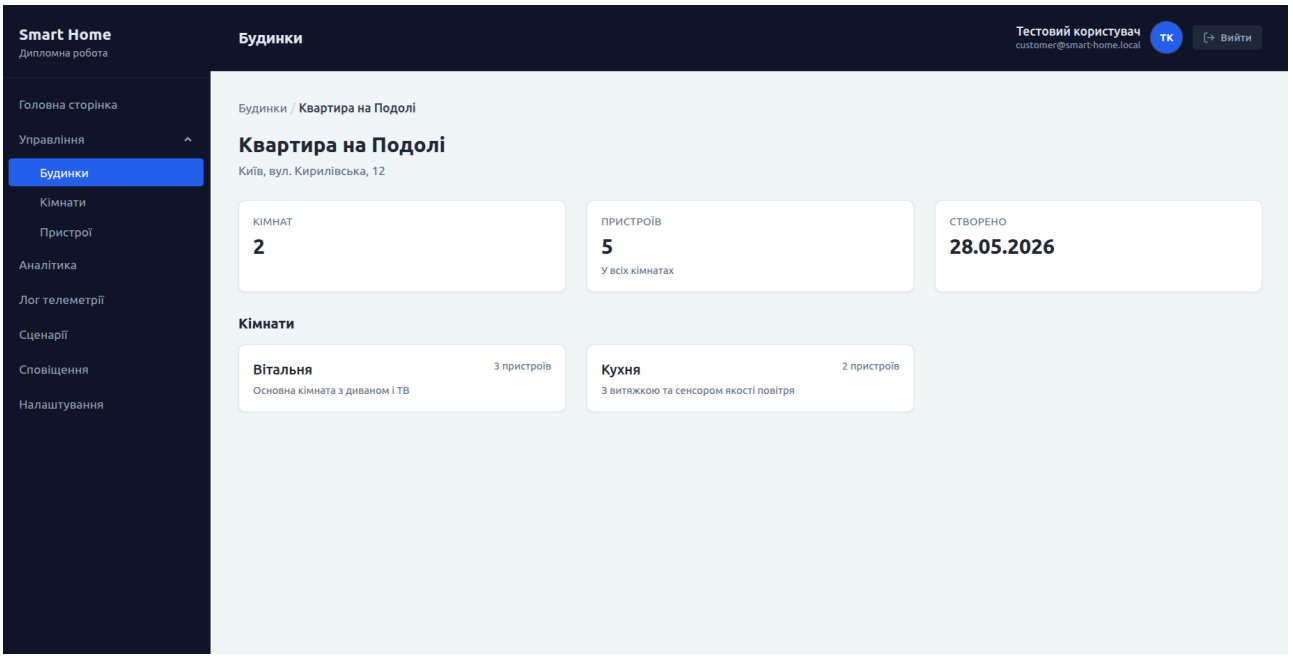

Рис. 2.8. Детальна сторінка будинку

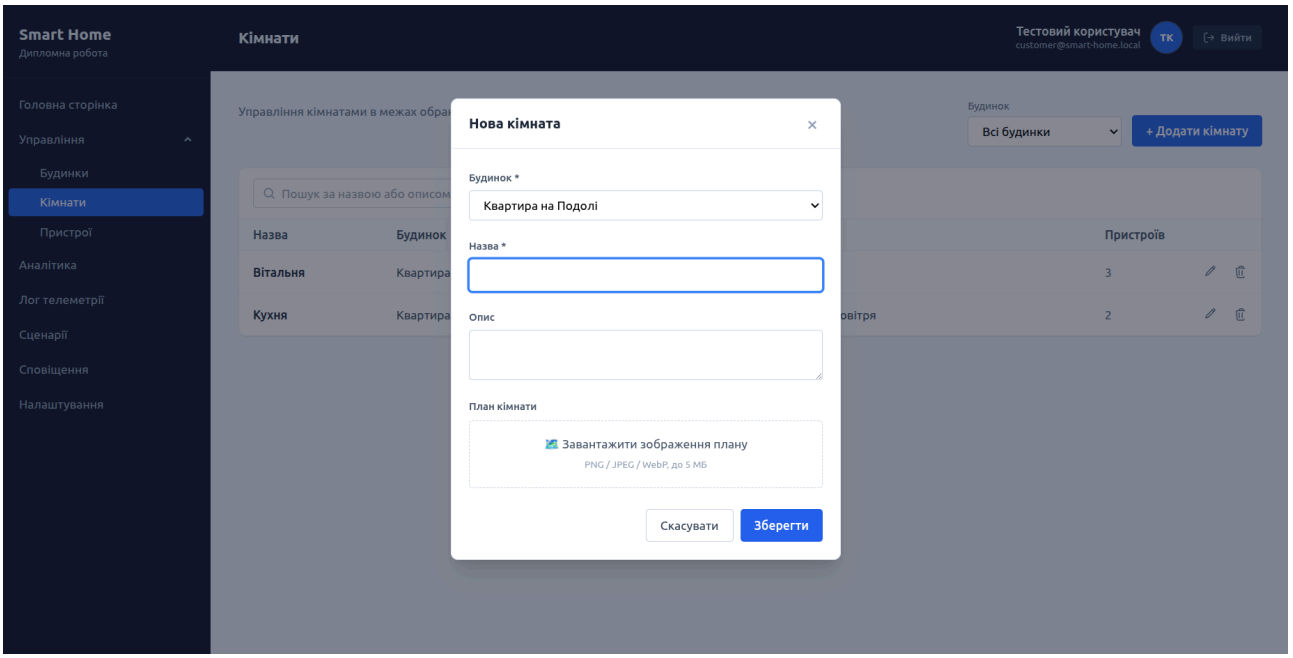

Рис. 2.9. Керування кімнатами та форма створення кімнати з планом

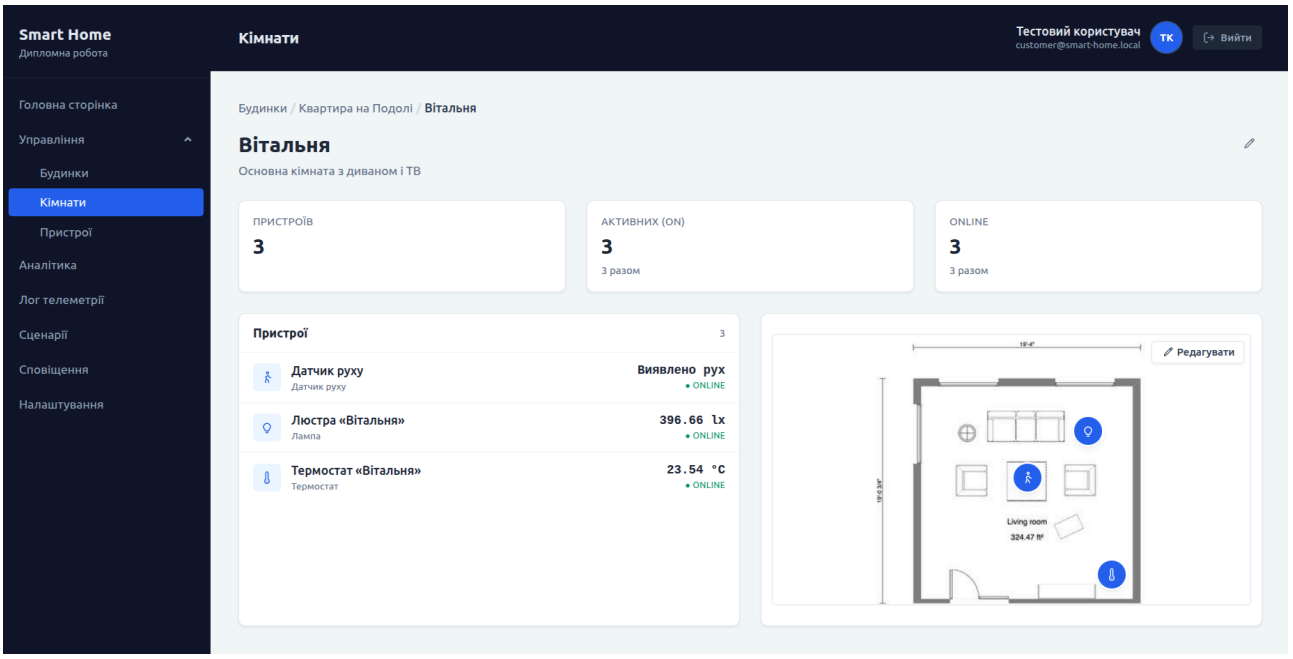

Рис. 2.10 Детальна сторінка поверху плану

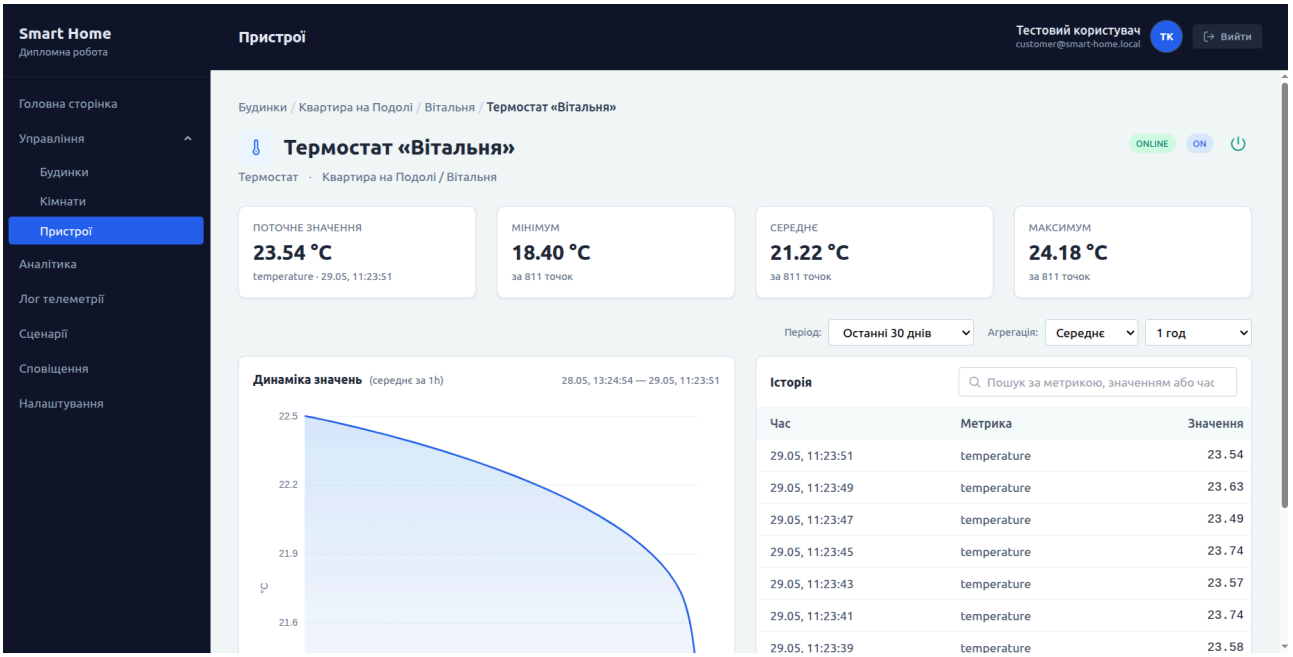

Рис. 2.11. Детальна сторінка пристрою

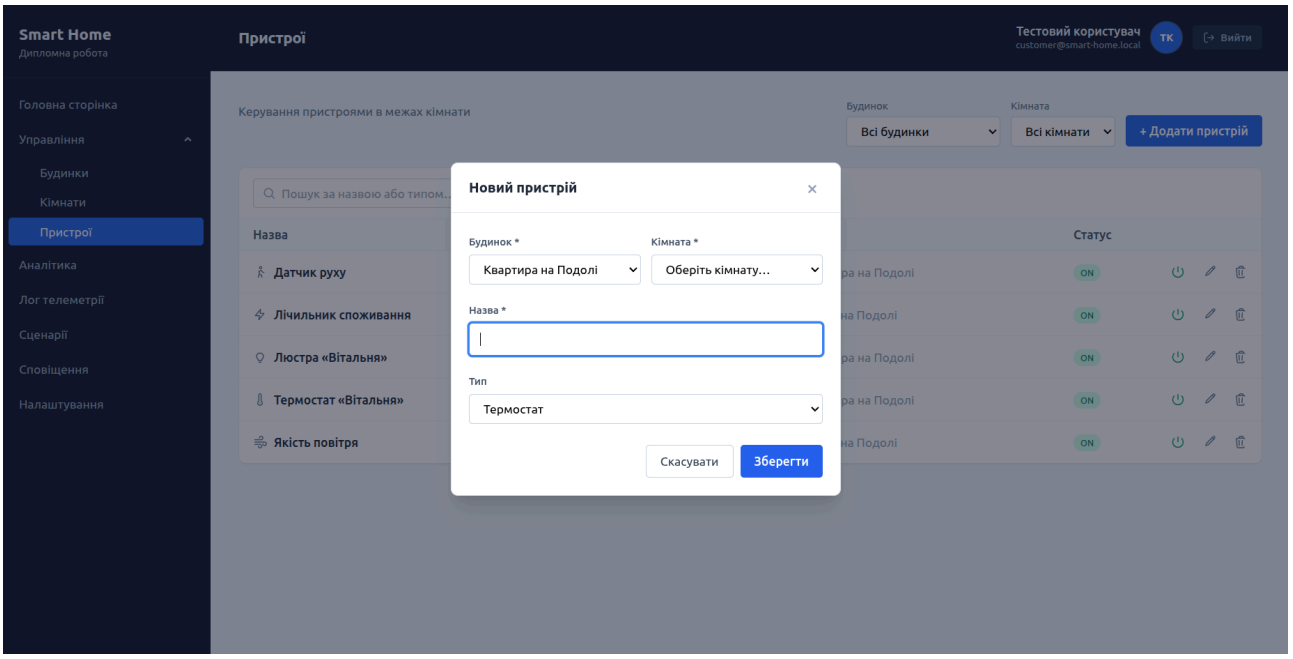

Рис. 2.12. Керування пристроями та форма додавання пристрою

Аналітичний модуль (рис. 2.10) поєднує панель параметрів (пристрій, метрика, період, агрегація) з чотирма типами візуалізації: лінійним графіком динаміки значень, шкальним індикатором поточного стану, стовпчиковою діаграмою розподілу за днями та тепловою картою активності у координатах «години — дні тижня».

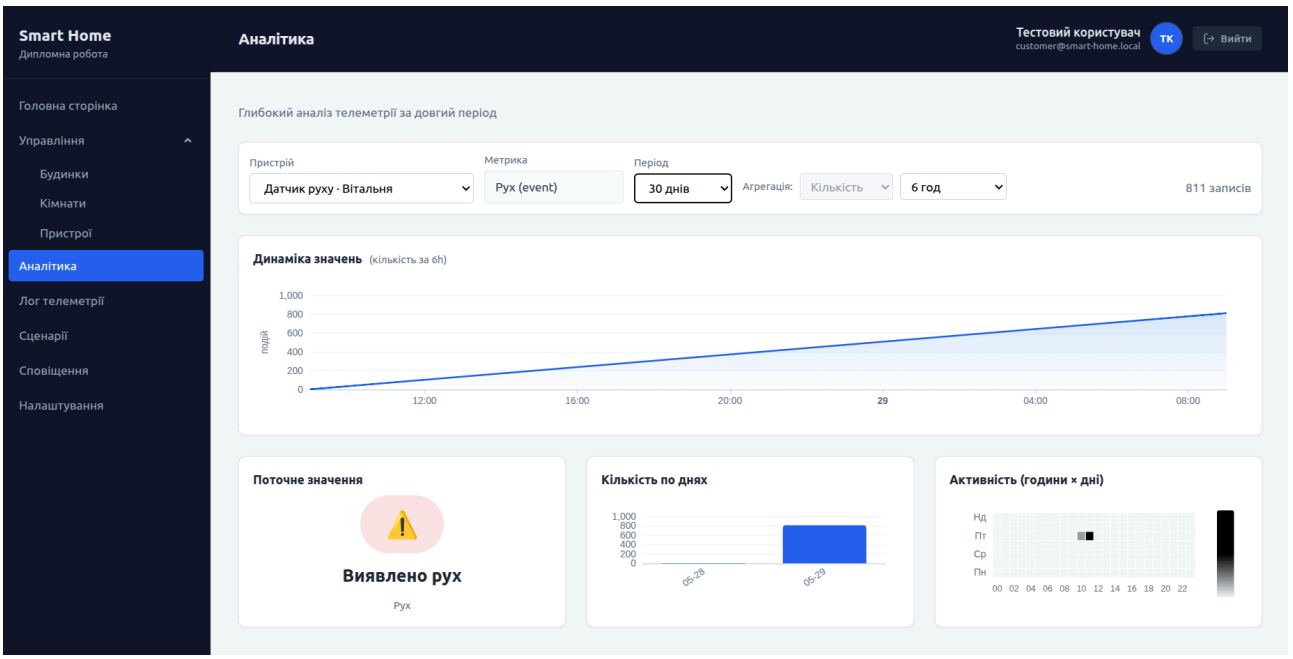

Рис. 2.13. Сторінка аналітики з чотирма типами візуалізації

Лог телеметрії (рис. 2.11) відображає стрічку показників від усіх пристроїв користувача з фільтрами за пристроєм, метрикою та періодом, пошуком і режимом автооновлення в реальному часі.

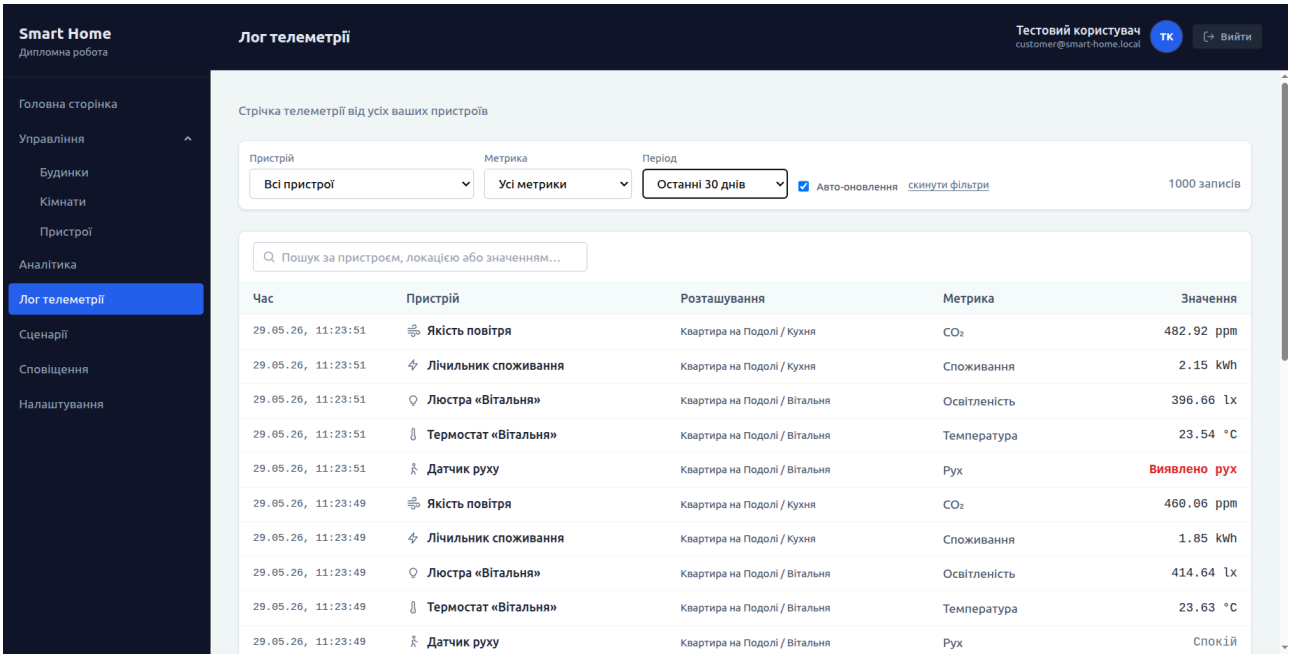

Рис. 2.14. Лог телеметрії з фільтрами та автооновленням

Сторінка сценаріїв (рис. 2.12) дозволяє створювати правила автоматизації: задати назву, обрати тип тригера (вручну, час або сенсор) і сформувати перелік дій — перемикання пристрою чи надсилання сповіщення.

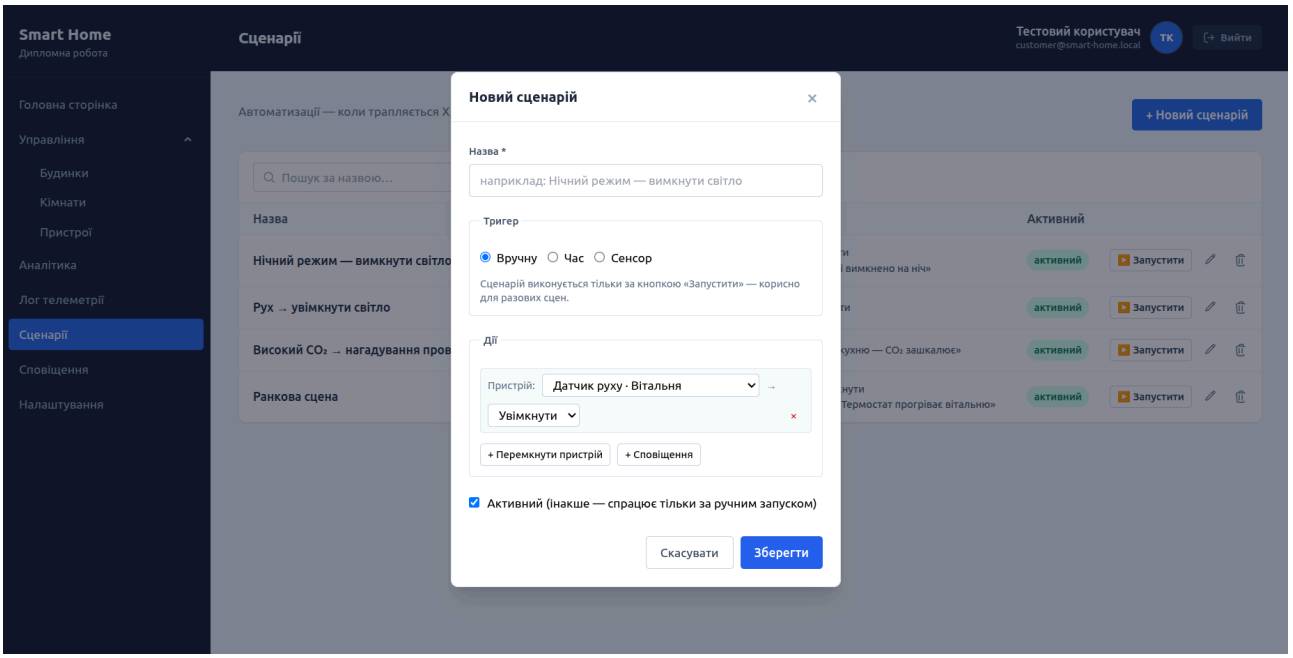

Рис. 2.15. Створення сценарію автоматизації

Підсистему сповіщень організовано у три вкладки — активні події, історія спрацювань та правила порогів (рис. 2.13–2.15). Активна подія містить поточне значення й тривалість, історія — пікове значення та спосіб завершення, а вкладка правил дозволяє створювати порогові умови для метрик.

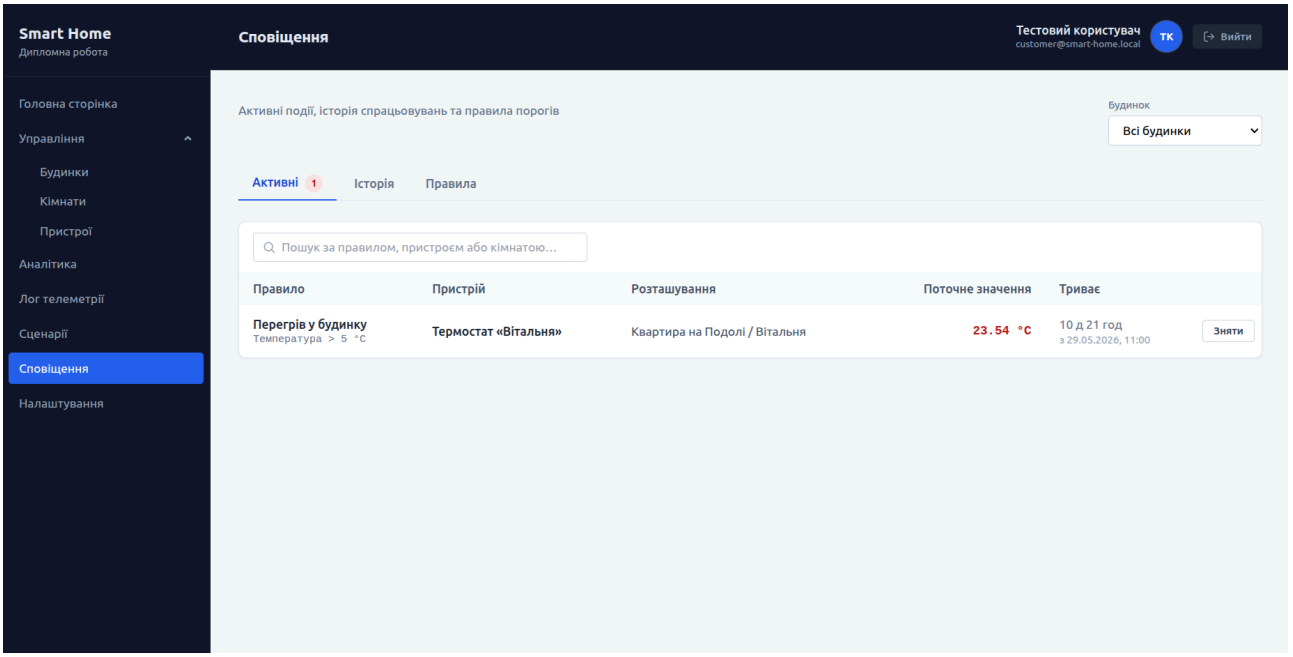

Рис. 2.16. Сповіщення: вкладка активних подій

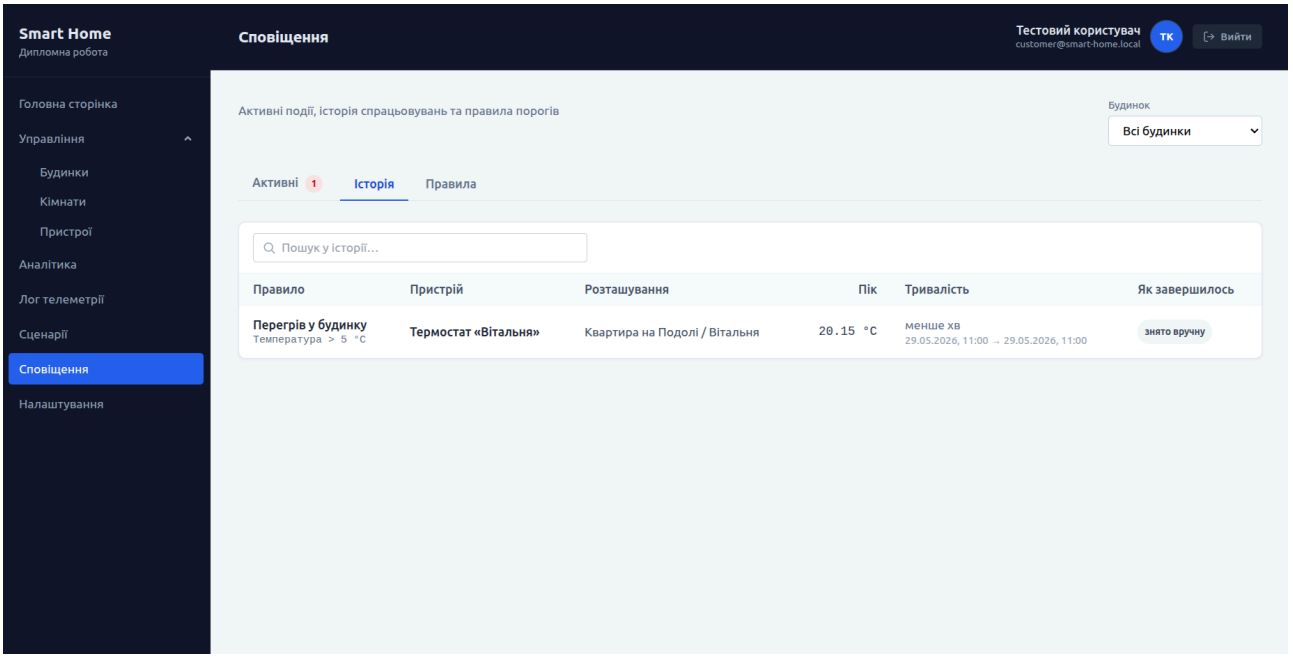

Рис. 2.17. Сповіщення: історія спрацювань

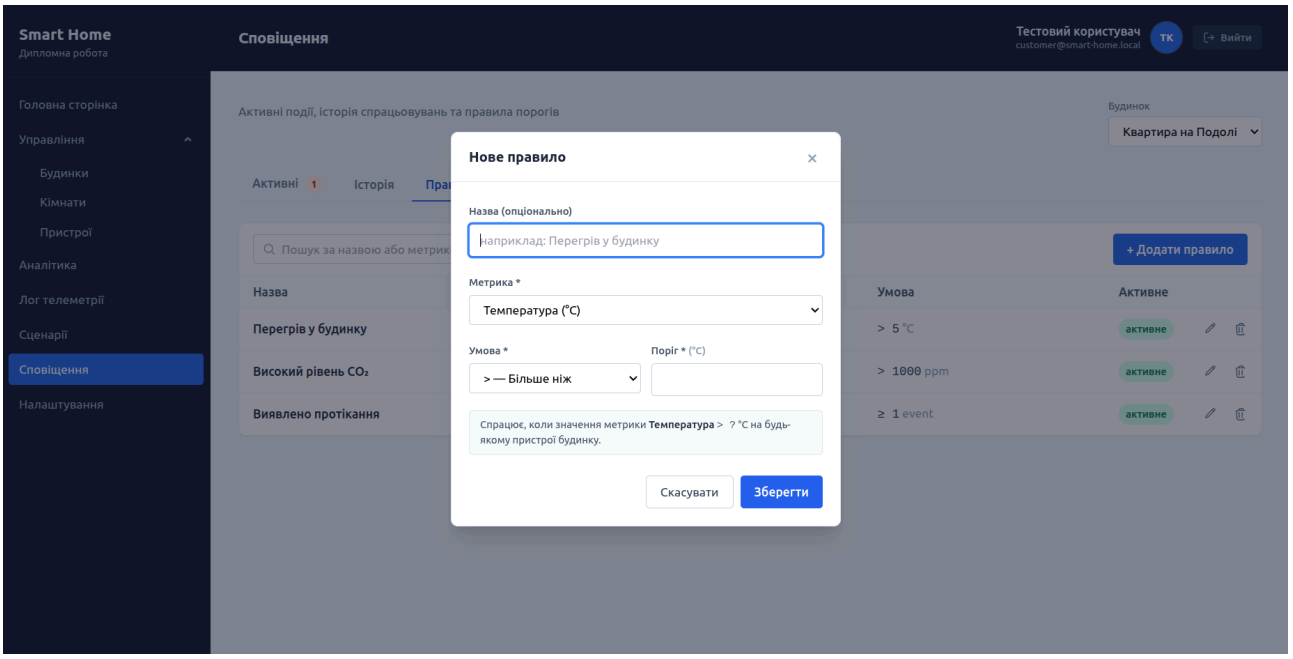

Рис. 2.18. Сповіщення: правила порогів і форма створення правила

Сторінка налаштувань (рис. 2.16) об'єднує профіль користувача з аватаром, керування емулятором телеметрії (інтервал, статус, статистика тіків) та вибір теми оформлення.

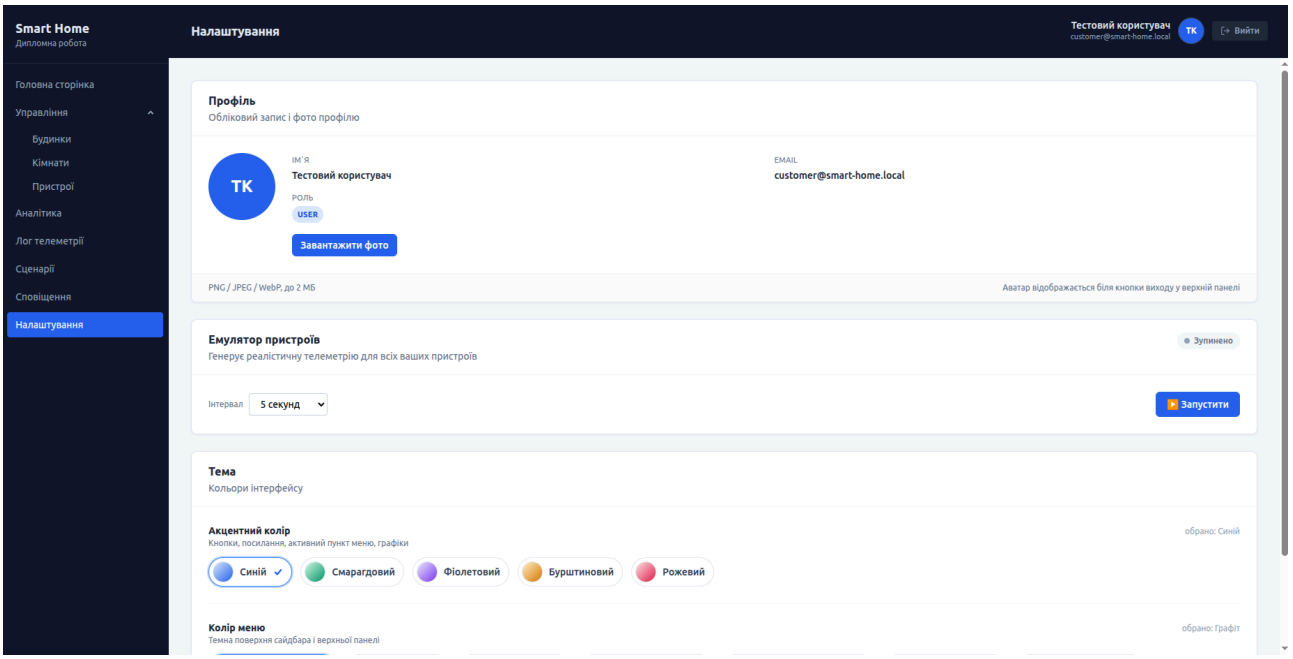

Рис. 2.19. Налаштування: профіль, емулятор телеметрії та тема

Адміністративну частину системи ілюструють рис. 2.17–2.19: список зареєстрованих користувачів зі зведеною статистикою та дією «Увійти як», сторінка деталей користувача й окрема сторінка налаштувань адміністратора з вибором теми.

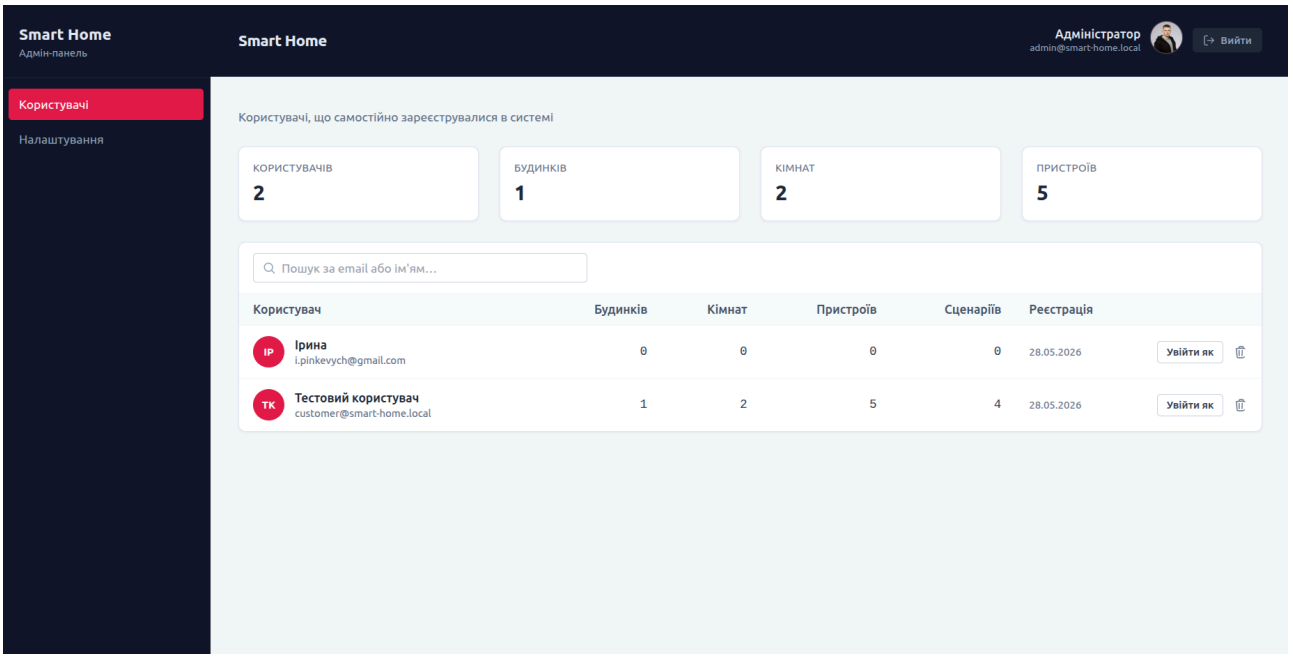

Рис. 2.20. Адмін-панель: список користувачів

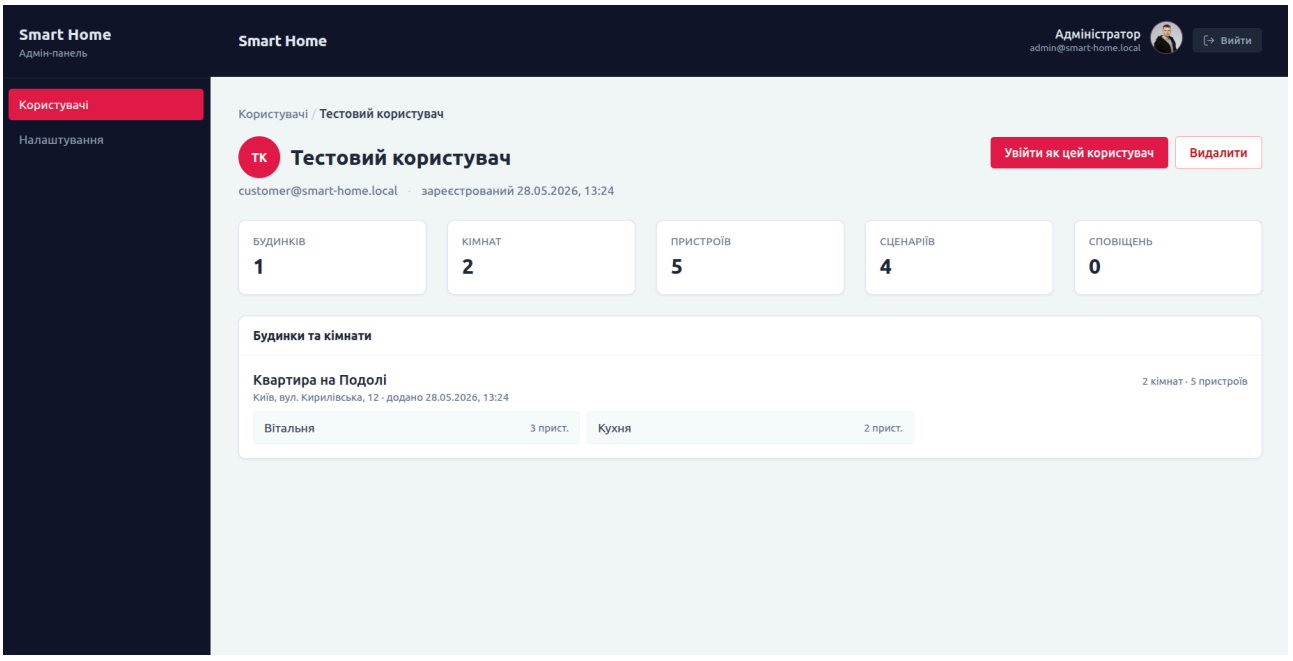

Рис. 2.21. Адмін-панель: деталі користувача

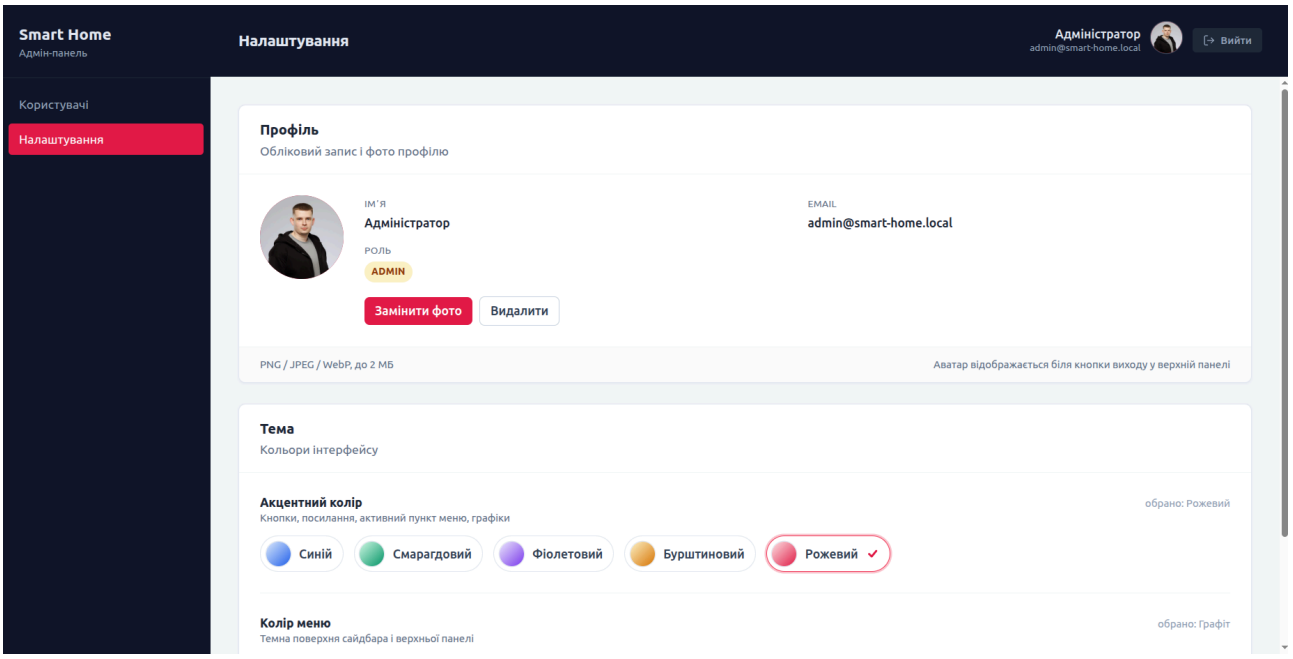

Рис. 2.22. Адмін-панель: налаштування та вибір теми

Функцію входу від імені клієнта (impersonation) демонструє рис. 2.20: адміністратор переглядає інтерфейс обраного користувача, а вгорі сторінки відображається банер «Адмін-режим» із кнопкою повернення до адміністративного облікового запису.

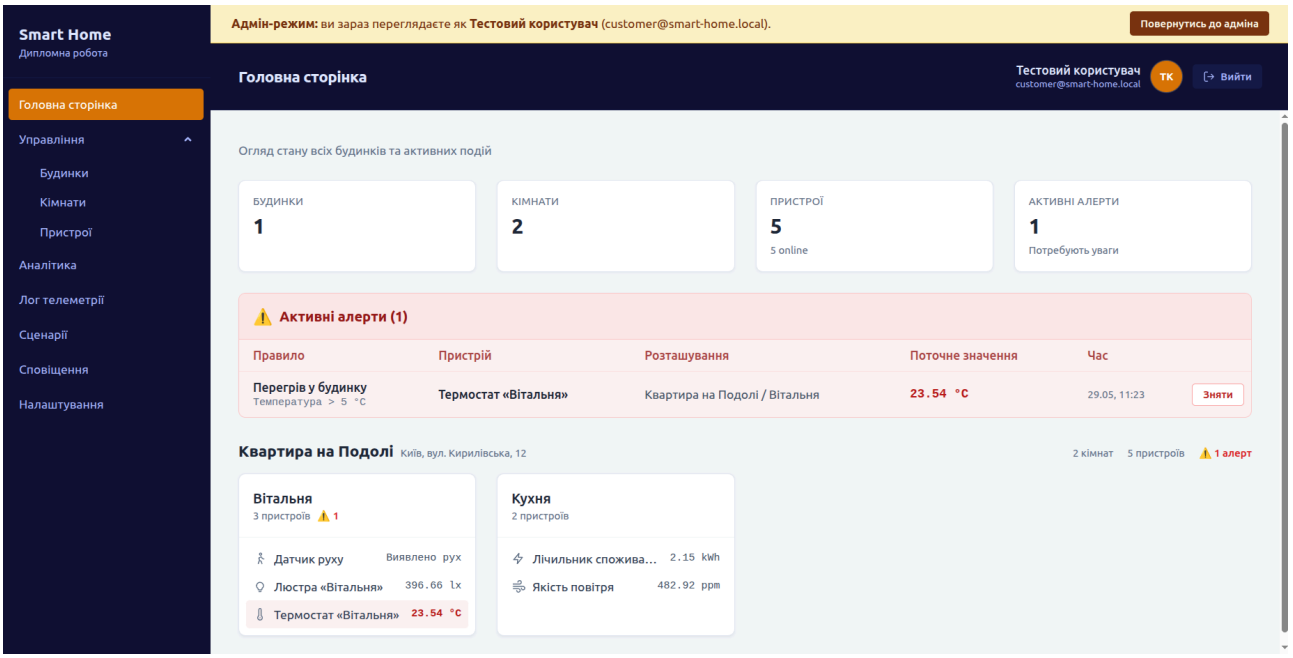

Рис. 2.23. Перегляд системи в режимі impersonation

Проведене тестування підтвердило відповідність реалізованої системи сформульованим у підрозділі 1.4 функціональним і нефункціональним вимогам,

а також стабільність роботи рушіїв реального часу під час безперервної генерації телеметрії.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи розроблено та реалізовано веборієнтовану інформаційну систему управління розумним домом. Проаналізовано предметну область, концепцію розумного дому та інтернету речей, що дало змогу сформулювати вимоги до функціональності системи.

Проведений огляд наявних рішень показав, що пропрієтарні екосистеми обмежують користувача у контролі та кастомізації й залежать від хмарних сервісів, тоді як відкриті платформи мають високий поріг входження. Це обґрунтувало доцільність розроблення власної системи з прозорою архітектурою та вбудованим емулятором телеметрії.

Обґрунтовано вибір технологічного стека (React, TypeScript, Node.js, Express, PostgreSQL, Prisma, Socket.IO) і клієнт-серверної архітектури з підтримкою реального часу. Спроєктовано архітектуру системи, дворівневу рольову модель та модель бази даних з одинадцяти таблиць із каскадним видаленням по ланцюгу володіння.

Реалізовано серверну частину з програмним інтерфейсом REST, трьома рушіями бізнес-логіки (емулятор телеметрії, оцінювач алертів, рушій сценаріїв) та каналом реального часу, а також клієнтську частину у вигляді односторінкового застосунку з аналітикою та керуванням пристроями. Безпеку забезпечено автентифікацією на основі JWT і рольовим розмежуванням доступу.

За результатами перевірки на демонстраційних даних підтверджено, що розроблена система дозволяє:

2. вести ієрархічний облік об'єктів розумного дому з контролем належності даних користувачу;
3. збирати й відображати телеметрію в реальному часі без повторних запитів до сервера;
4. автоматично виявляти перетин порогових значень і керувати подіями сповіщень;

5. виконувати сценарії автоматизації за часовими, сенсорними та ручними тригерами;
6. демонструвати повний цикл роботи без фізичного обладнання завдяки вбудованому емулятору.

Проведене тестування на демонстраційному наборі даних підтвердило відповідність системи сформульованим функціональним і нефункціональним вимогам, зокрема щодо продуктивності оновлень у реальному часі, безпеки доступу та зручності використання. Реалізовані механізми обробки помилок і структурованого журналювання забезпечують стійкість роботи під час безперервної генерації телеметрії.

Таким чином, мету роботи досягнуто. Подальший розвиток системи можливий у напрямах інтеграції з реальними IoT-пристроями через протокол MQTT, розроблення мобільного застосунку, додавання модулів машинного навчання для прогнозування показників та оптимізації енергоспоживання.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

7. SkyQuest. IoT Smart Homes Market Size, Share, Forecast: вебсайт. URL: https://www.skyquestt.com/report/iot-smart-homes-market (дата звернення: 29.05.2026).
8. Fortune Business Insights. Smart Home Market Size, Share & Growth: вебсайт. URL: https://www.fortunebusinessinsights.com/industry-reports/smart-home-market-101900 (дата звернення: 29.05.2026).
9. Grand View Research. Smart Homes Industry Report, 2030: вебсайт. URL: https://www.grandviewresearch.com/industry-analysis/smart-homes-industry (дата звернення: 29.05.2026).
10. Straits Research. Smart Home Market (GSMA Intelligence): вебсайт. URL: https://straitsresearch.com/report/smart-home-market (дата звернення: 29.05.2026).
11. Smart Home Digest. Smart Home Ecosystems Compared (2026): вебсайт. URL: https://smarthomedigest.com/articles/smart-home-ecosystems-compared-2026-alexa-vs-google-home-vs-homekit-vs-home-assistant (дата звернення: 29.05.2026).
12. CEDIA. The Best Open Source Smart Home Tools: вебсайт. URL: https://cedia.org/homeowners/knowledge/best-open-source-smart-home-tools/ (дата звернення: 29.05.2026).
13. Eufy. Top Open Source Home Automation Tools: вебсайт. URL: https://www.eufy.com/blogs/home/open-source-smart-home (дата звернення: 29.05.2026).
14. TS2. Smart Home Showdown 2025: Google Nest vs Alexa, HomeKit & the Rest: вебсайт. URL: https://ts2.tech/en/smart-home-showdown-2025-google-nest-vs-alexa-homekit-the-rest-which-ecosystem-rules/ (дата звернення: 29.05.2026).

15. Apple HomeKit vs Google Home vs Alexa vs SmartThings in 2026: вебсайт. URL: https://ththeater.com/apple-homekit-vs-google-home-vs-alexa-vs-smartthings/ (дата звернення: 29.05.2026).
16. AmpVortex. Comparison Guide for Mainstream Smart Home Integration Platforms: вебсайт. URL: https://www.ampvortex.com/comparison-and-selection-guide-for-mainstream-smart-home-integration-platforms/ (дата звернення: 29.05.2026).
17. Connectivity Standards Alliance. Matter: the foundation for connected things: вебсайт. URL: https://csa-iot.org/all-solutions/matter/ (дата звернення: 29.05.2026).
18. OASIS. MQTT Version 5.0 Specification: вебсайт. URL: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html (дата звернення: 29.05.2026).
19. Zigbee Alliance. Zigbee Specification: вебсайт. URL: https://csa-iot.org/all-solutions/zigbee/ (дата звернення: 29.05.2026).
20. Thread Group. Thread Network Technology: вебсайт. URL: https://www.threadgroup.org/ (дата звернення: 29.05.2026).
21. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures: дис. … PhD. University of California, Irvine, 2000. 162 p.
22. React: офіційна документація. URL: https://react.dev (дата звернення: 29.05.2026).
23. TypeScript: офіційна документація. URL: https://www.typescriptlang.org/docs/ (дата звернення: 29.05.2026).
24. Vite: офіційна документація. URL: https://vitejs.dev/guide/ (дата звернення: 29.05.2026).
25. TailwindCSS: офіційна документація. URL: https://tailwindcss.com/docs (дата звернення: 29.05.2026).
26. Apache ECharts: офіційна документація. URL: https://echarts.apache.org/en/index.html (дата звернення: 29.05.2026).

27. Node.js: офіційний сайт. URL: https://nodejs.org (дата звернення: 29.05.2026).
28. Express: офіційна документація. URL: https://expressjs.com (дата звернення: 29.05.2026).
29. Prisma: офіційна документація. URL: https://www.prisma.io/docs (дата звернення: 29.05.2026).
30. Socket.IO: офіційна документація. URL: https://socket.io/docs/ (дата звернення: 29.05.2026).
31. Zod: TypeScript-first schema validation: вебсайт. URL: https://zod.dev (дата звернення: 29.05.2026).
32. PostgreSQL 16: офіційна документація. URL: https://www.postgresql.org/docs/16/ (дата звернення: 29.05.2026).
33. PostgreSQL: JSON Types (JSONB): вебсайт. URL: https://www.postgresql.org/docs/current/datatype-json.html (дата звернення: 29.05.2026).
34. Jones M., Bradley J., Sakimura N. RFC 7519: JSON Web Token (JWT). IETF, 2015. URL: https://datatracker.ietf.org/doc/html/rfc7519 (дата звернення: 29.05.2026).
35. Provos N., Mazières D. A Future-Adaptable Password Scheme (bcrypt). USENIX, 1999. URL: https://www.usenix.org/legacy/events/usenix99/provos.html (дата звернення: 29.05.2026).
36. Statista. Number of IoT connected devices worldwide: вебсайт. URL: https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/ (дата звернення: 29.05.2026).
37. Angular: офіційна документація. URL: https://angular.dev (дата звернення: 29.05.2026).
38. Vue.js: офіційна документація. URL: https://vuejs.org/guide/introduction.html (дата звернення: 29.05.2026).

39. GraphQL: специфікація мови запитів. URL: https://graphql.org/learn/ (дата звернення: 29.05.2026).
40. Django: офіційна документація. URL: https://docs.djangoproject.com/ (дата звернення: 29.05.2026).
41. Spring Framework: офіційна документація. URL: https://spring.io/projects/spring-framework (дата звернення: 29.05.2026).
42. MySQL 8.0 Reference Manual: вебсайт. URL: https://dev.mysql.com/doc/ (дата звернення: 29.05.2026).
43. MongoDB: офіційна документація. URL: https://www.mongodb.com/docs/ (дата звернення: 29.05.2026).
44. ISO/IEC 25010:2011. Systems and software engineering — SQuaRE — System and software quality models. Geneva: ISO, 2011.
45. OWASP Top Ten Web Application Security Risks: вебсайт. URL: https://owasp.org/www-project-top-ten/ (дата звернення: 29.05.2026).

ДОДАТКИ

У додатках наведено основні фрагменти програмного коду системи. Лістинги відтворюють структуру та логіку реалізації; повний вихідний код розміщено у репозиторії проєкту.

Додаток А. Схема бази даних (Prisma schema)

```
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}

enum UserRole            { admin  user }
enum DeviceType          { thermostat lamp motion_sensor power_meter air_quality
water_leak smart_lock }
enum DeviceStatus        { on  off }
enum MetricType          { temperature humidity power motion co2 light_level water_leak }
enum AggregationPeriod   { hour  day }
enum ScenarioTriggerType { time  sensor  manual }
enum AlertCondition      { gt  lt  eq  gte  lte }
enum NotificationType    { warning  info  alert }

model User {
  id            String   @id @default(uuid())
  email         String   @unique
  passwordHash  String   @map("password_hash")
  fullName      String   @map("full_name")
  role          UserRole @default(user)
  avatarUrl     String?  @map("avatar_url")
  themeBrand    String?  @map("theme_brand")
  themePrimary  String?  @map("theme_primary")
  createdAt     DateTime @default(now()) @map("created_at")
  houses        House[]
  scenarios     Scenario[]
  notifications Notification[]
  deviceLogs    DeviceLog[]
  @@map("users")
}

model House {
  id      Int    @id @default(autoincrement())
  userId  String @map("user_id")
  name    String
  address String?
  user    User   @relation(fields: [userId], references: [id], onDelete: Cascade)
  rooms   Room[]
  alerts  Alert[]
  @@index([userId])
  @@map("houses")
}

model Room {
  id          Int     @id @default(autoincrement())
  houseId     Int     @map("house_id")
  name        String
  description String?
  floorplanUrl String? @map("floorplan_url")
  house       House   @relation(fields: [houseId], references: [id], onDelete: Cascade)
  devices     Device[]
  @@index([houseId])
  @@map("rooms")
```

```
}

model Device {
  id          String       @id @default(uuid())
  roomId      Int          @map("room_id")
  name        String
  type        DeviceType
  status      DeviceStatus @default(off)
  isOnline    Boolean      @default(true) @map("is_online")
  floorplanX  Float?       @map("floorplan_x")
  floorplanY  Float?       @map("floorplan_y")
  room        Room         @relation(fields: [roomId], references: [id], onDelete:
Cascade)
  telemetry   Telemetry[]
  @@index([roomId])
  @@map("devices")
}

model Telemetry {
  id         BigInt     @id @default(autoincrement())
  deviceId   String     @map("device_id")
  metricType MetricType @map("metric_type")
  value      Decimal    @db.Decimal(14, 4)
  unit       String
  timestamp  DateTime   @default(now())
  device     Device     @relation(fields: [deviceId], references: [id], onDelete:
Cascade)
  @@index([deviceId, timestamp(sort: Desc)])
  @@index([metricType, timestamp(sort: Desc)])
  @@map("telemetry")
}

model Alert {
  id             Int            @id @default(autoincrement())
  houseId        Int            @map("house_id")
  name           String
  metricType     MetricType     @map("metric_type")
  condition      AlertCondition
  thresholdValue Decimal        @map("threshold_value") @db.Decimal(14, 4)
  isActive       Boolean        @default(true) @map("is_active")
  house          House          @relation(fields: [houseId], references: [id], onDelete:
Cascade)
  events         AlertEvent[]
  @@index([houseId, metricType])
  @@map("alerts")
}

model AlertEvent {
  id             Int       @id @default(autoincrement())
  alertId        Int       @map("alert_id")
  deviceId       String    @map("device_id")
  triggerValue   Decimal   @map("trigger_value") @db.Decimal(14, 4)
  latestValue    Decimal   @map("latest_value")  @db.Decimal(14, 4)
  triggeredAt    DateTime  @default(now()) @map("triggered_at")
  lastSeenAt     DateTime  @map("last_seen_at")
  clearedAt      DateTime? @map("cleared_at")
  clearReason    String?   @map("clear_reason")
  alert          Alert     @relation(fields: [alertId], references: [id], onDelete:
Cascade)
  @@index([alertId, clearedAt, triggeredAt(sort: Desc)])
  @@map("alert_events")
}

model Scenario {
  id           Int                @id @default(autoincrement())
  userId       String             @map("user_id")
  name         String
  triggerType  ScenarioTriggerType @map("trigger_type")
  triggerValue Json?              @map("trigger_value")
  actions      Json
  isActive     Boolean            @default(true) @map("is_active")
```

```
  user         User                @relation(fields: [userId], references: [id],
onDelete: Cascade)
  @@index([userId])
  @@map("scenarios")
}

model Notification {
  id         Int              @id @default(autoincrement())
  userId     String           @map("user_id")
  deviceId   String?          @map("device_id")
  message    String
  type       NotificationType
  isRead     Boolean          @default(false) @map("is_read")
  createdAt  DateTime         @default(now()) @map("created_at")
  user       User             @relation(fields: [userId], references: [id], onDelete:
Cascade)
  @@index([userId, createdAt(sort: Desc)])
  @@map("notifications")
}

model DeviceLog {
  id         Int      @id @default(autoincrement())
  deviceId   String   @map("device_id")
  userId     String?  @map("user_id")
  action     String
  oldValue   String?  @map("old_value")
  newValue   String?  @map("new_value")
  timestamp  DateTime @default(now())
  user       User?    @relation(fields: [userId], references: [id], onDelete: SetNull)
  @@map("device_logs")
}

model TelemetryAggregated {
  id          BigInt            @id @default(autoincrement())
  deviceId    String            @map("device_id")
  metricType  MetricType        @map("metric_type")
  period      AggregationPeriod
  avgValue    Decimal           @map("avg_value") @db.Decimal(14, 4)
  minValue    Decimal           @map("min_value") @db.Decimal(14, 4)
  maxValue    Decimal           @map("max_value") @db.Decimal(14, 4)
  periodStart DateTime          @map("period_start")
  @@unique([deviceId, metricType, period, periodStart])
  @@map("telemetry_aggregated")
}
```

Додаток Б. Оцінювач порогових сповіщень (alertEvaluator.ts)

```
import { prisma } from "../db";
import { emitToUser } from "../realtime";

interface Reading { deviceId: string; metricType: string; value: number; unit: string; }

// rising-edge оцінка порогів після кожного запису телеметрії
export async function evaluateForReadings(userId: string, readings: Reading[]) {
  // лишаємо лише останнє значення для кожної пари (пристрій, метрика)
  const latest = new Map<string, Reading>();
  for (const r of readings) latest.set(r.deviceId + ":" + r.metricType, r);

  const alerts = await prisma.alert.findMany({
    where: { isActive: true, house: { user: { id: userId } } },
  });

  for (const a of alerts) {
    for (const r of latest.values()) {
      if (r.metricType !== a.metricType) continue;
      const hit = compare(r.value, a.condition, Number(a.thresholdValue));
      const open = await prisma.alertEvent.findFirst({
        where: { alertId: a.id, deviceId: r.deviceId, clearedAt: null },
      });

      if (hit && !open) {
```

```
        const ev = await prisma.alertEvent.create({
          data: {
            alertId: a.id, deviceId: r.deviceId,
            triggerValue: r.value, latestValue: r.value,
            lastSeenAt: new Date(),
          },
        });
        emitToUser(userId, "alert:event", { kind: "opened", event: ev });
      } else if (hit && open) {
        await prisma.alertEvent.update({
          where: { id: open.id },
          data: { latestValue: r.value, lastSeenAt: new Date() },
        });
      } else if (!hit && open) {
        const ev = await prisma.alertEvent.update({
          where: { id: open.id },
          data: { clearedAt: new Date(), clearReason: "auto" },
        });
        emitToUser(userId, "alert:event", { kind: "cleared", event: ev });
      }
    }
  }
}

function compare(v: number, cond: string, t: number): boolean {
  switch (cond) {
    case "gt":  return v >  t;
    case "lt":  return v <  t;
    case "gte": return v >= t;
    case "lte": return v <= t;
    case "eq":  return v === t;
    default:    return false;
  }
}
```

Додаток В. Проміжне програмне забезпечення автентифікації (middleware.ts)

```
import { Request, Response, NextFunction } from "express";
import { verifyAccessToken } from "./jwt";

export interface AuthedRequest extends Request {
  user?: { sub: string; email: string; role: string };
}

// перевірка Bearer-токена
export function requireAuth(req: AuthedRequest, res: Response, next: NextFunction) {
  const header = req.headers.authorization;
  if (!header?.startsWith("Bearer ")) {
    return res.status(401).json({ error: "Не авторизовано" });
  }
  try {
    req.user = verifyAccessToken(header.slice(7));
    next();
  } catch {
    return res.status(401).json({ error: "Недійсний токен" });
  }
}

// перевірка ролі
export function requireRole(...roles: string[]) {
  return (req: AuthedRequest, res: Response, next: NextFunction) => {
    if (!req.user || !roles.includes(req.user.role)) {
      return res.status(403).json({ error: "Недостатньо прав" });
    }
    next();
  };
}
```

Додаток Г. Рушій емулятора телеметрії (emulator.ts)

```
import { prisma } from "../db";
import { emitToUser } from "../realtime";
import { evaluateForReadings } from "./alertEvaluator";
import { evaluateSensorScenarios } from "./scenarioEngine";

const timers = new Map<string, NodeJS.Timeout>();

export function startEmulator(userId: string, intervalMs = 5000) {
  stopEmulator(userId);
  const period = Math.min(60000, Math.max(1000, intervalMs));
  const timer = setInterval(() => tick(userId).catch(console.error), period);
  timers.set(userId, timer);
}

export function stopEmulator(userId: string) {
  const t = timers.get(userId);
  if (t) { clearInterval(t); timers.delete(userId); }
}

export function emulatorStatus(userId: string) {
  return { running: timers.has(userId) };
}

async function tick(userId: string) {
  const devices = await prisma.device.findMany({
    where: { room: { house: { userId } } },
    include: { telemetry: { orderBy: { timestamp: "desc" }, take: 1 } },
  });

  const readings = devices.map((d) => generate(d));
  if (readings.length === 0) return;

  await prisma.telemetry.createMany({
    data: readings.map((r) => ({
      deviceId: r.deviceId, metricType: r.metricType as any,
      value: r.value, unit: r.unit,
    })),
  });

  await evaluateForReadings(userId, readings);
  await evaluateSensorScenarios(userId, readings);

  for (const r of readings) {
    emitToUser(userId, "telemetry:new", {
      deviceId: r.deviceId,
      points: [{ metricType: r.metricType, value: r.value, unit: r.unit, timestamp: new
Date() }],
    });
  }
}

function rand(min: number, max: number) { return Math.random() * (max - min) + min; }
function clamp(v: number, lo: number, hi: number) { return Math.min(hi, Math.max(lo, v));
}

// генерація наступного значення залежно від типу пристрою
function generate(device: any) {
  const prev = Number(device.telemetry[0]?.value ?? 0);
  switch (device.type) {
    case "thermostat":
      return reading(device, "temperature", clamp(prev + rand(-0.6, 0.6), 18, 26), "°C");
    case "lamp":
      return reading(device, "light_level", clamp(prev + rand(-60, 60), 100, 600), "lx");
    case "power_meter": {
      const v = device.status === "off" ? rand(0, 0.05)
              : clamp(prev + rand(-0.3, 0.3) + (Math.random() < 0.08 ? 0.8 : 0), 0, 12);
      return reading(device, "power", v, "kWh");
    }
    case "air_quality": {
      const v = clamp(prev + 40 - (Math.random() < 0.05 ? 200 : 0), 400, 2000);
      return reading(device, "co2", v, "ppm");
```

```
    }
    case "motion":
    case "motion_sensor":
      return reading(device, "motion", Math.random() < 0.15 ? 1 : 0, "");
    case "smart_lock":
      return reading(device, "motion", Math.random() < 0.05 ? 1 : 0, "");
    case "water_leak":
      return reading(device, "water_leak", Math.random() < 0.01 ? 1 : 0, "");
    default:
      return reading(device, "temperature", prev, "");
  }
}

function reading(device: any, metricType: string, value: number, unit: string) {
  return { deviceId: device.id, metricType, value: Number(value.toFixed(2)), unit };
}
```

Додаток Д. Компонент сторінки пристрою з графіком у реальному часі (DeviceDashboard.tsx)

```
import { useEffect, useState } from "react";
import { useParams } from "react-router-dom";
import ReactECharts from "echarts-for-react";
import { api, Telemetry } from "../api";
import { socket } from "../realtime";

export default function DeviceDashboard() {
  const { id } = useParams<{ id: string }>();
  const [points, setPoints] = useState<Telemetry[]>([]);

  useEffect(() => {
    if (!id) return;
    // початкова серія показників
    api.getDeviceTelemetry(id).then(setPoints);

    // живі оновлення через WebSocket
    const onNew = (msg: { deviceId: string; points: Telemetry[] }) => {
      if (msg.deviceId !== id) return;
      setPoints((prev) => [...prev, ...msg.points].slice(-200));
    };
    socket.on("telemetry:new", onNew);
    return () => { socket.off("telemetry:new", onNew); };
  }, [id]);

  const option = {
    tooltip: { trigger: "axis" },
    xAxis: { type: "time" },
    yAxis: { type: "value" },
    series: [{
      type: "line",
      showSymbol: false,
      data: points.map((p) => [new Date(p.timestamp).getTime(), Number(p.value)]),
    }],
  };

  return (
    <div className="p-4">
      <h1 className="text-xl font-semibold mb-4">Показники пристрою</h1>
      <ReactECharts option={option} style={{ height: 360 }} />
    </div>
  );
}
```

Додаток Е. Рушій сценаріїв автоматизації (scenarioEngine.ts)

```
import { prisma } from "../db";
import { emitToUser } from "../realtime";

const sensorState = new Map<string, boolean>();
```

```
// виконання масиву дій сценарію
export async function runScenario(id: number, source: string) {
  const sc = await prisma.scenario.findUnique({ where: { id } });
  if (!sc || !sc.isActive) return;
  const actions = (sc.actions as any[]) ?? [];

  for (const a of actions) {
    if (a.type === "set_device_status") {
      const dev = await prisma.device.update({
        where: { id: a.deviceId },
        data: { status: a.status },
      });
      await prisma.deviceLog.create({
        data: { deviceId: dev.id, userId: sc.userId, action: "scenario:" + source,
                newValue: a.status },
      });
    } else if (a.type === "notify") {
      await prisma.notification.create({
        data: { userId: sc.userId, message: a.message, type: a.level ?? "info" },
      });
      emitToUser(sc.userId, "notification:new", { message: a.message });
    }
  }
}

// планувальник часових тригерів (тік раз на хвилину)
export function startScenarioScheduler() {
  setInterval(async () => {
    const now = new Date();
    const scenarios = await prisma.scenario.findMany({
      where: { isActive: true, triggerType: "time" },
    });
    for (const sc of scenarios) {
      const t = sc.triggerValue as any;
      if (t?.hour === now.getHours() && t?.minute === now.getMinutes()) {
        await runScenario(sc.id, "time");
      }
    }
  }, 60000);
}

// сенсорні тригери за логікою фронту наростання
export async function evaluateSensorScenarios(userId: string, readings: any[]) {
  const scenarios = await prisma.scenario.findMany({
    where: { isActive: true, triggerType: "sensor", userId },
  });
  for (const sc of scenarios) {
    const t = sc.triggerValue as any;
    const r = readings.find((x) => x.metricType === t.metricType);
    if (!r) continue;
    const cond = compare(r.value, t.condition, t.value);
    const key = sc.id + ":" + r.deviceId;
    const prev = sensorState.get(key) ?? false;
    if (cond && !prev) await runScenario(sc.id, "sensor"); // rising edge
    sensorState.set(key, cond);
  }
}

function compare(v: number, c: string, t: number) {
  return c === "gt" ? v > t : c === "lt" ? v < t
       : c === "gte" ? v >= t : c === "lte" ? v <= t : v === t;
}
```

Додаток Ж. Маршрут керування пристроями (routes/devices.ts)

```
import { Router } from "express";
import { z } from "zod";
import { prisma } from "../db";
import { requireAuth, requireRole, AuthedRequest } from "../auth/middleware";

const router = Router();
```

```
router.use(requireAuth, requireRole("user"));

const deviceSchema = z.object({
  roomId: z.number().int(),
  name: z.string().trim().min(1).max(100),
  type: z.enum(["thermostat", "lamp", "motion_sensor", "power_meter",
                "air_quality", "water_leak", "smart_lock"]),
});

// список пристроїв поточного користувача
router.get("/", async (req: AuthedRequest, res) => {
  const devices = await prisma.device.findMany({
    where: { room: { house: { userId: req.user!.sub } } },
    include: { room: true },
  });
  res.json(devices);
});

// створення пристрою з перевіркою власності кімнати
router.post("/", async (req: AuthedRequest, res) => {
  const parsed = deviceSchema.safeParse(req.body);
  if (!parsed.success) return res.status(400).json({ error: parsed.error.issues });

  const room = await prisma.room.findFirst({
    where: { id: parsed.data.roomId, house: { userId: req.user!.sub } },
  });
  if (!room) return res.status(404).json({ error: "Кімнату не знайдено" });

  const device = await prisma.device.create({ data: parsed.data });
  res.status(201).json(device);
});

// зміна статусу/назви пристрою
router.patch("/:id", async (req: AuthedRequest, res) => {
  const device = await prisma.device.findFirst({
    where: { id: req.params.id, room: { house: { userId: req.user!.sub } } },
  });
  if (!device) return res.status(404).json({ error: "Пристрій не знайдено" });

  const updated = await prisma.device.update({
    where: { id: device.id },
    data: { name: req.body.name ?? device.name, status: req.body.status ?? device.status
},
  });
  res.json(updated);
});

// видалення пристрою
router.delete("/:id", async (req: AuthedRequest, res) => {
  const device = await prisma.device.findFirst({
    where: { id: req.params.id, room: { house: { userId: req.user!.sub } } },
  });
  if (!device) return res.status(404).json({ error: "Пристрій не знайдено" });
  await prisma.device.delete({ where: { id: device.id } });
  res.status(204).end();
});

export default router;
```

Додаток З. Типізований клієнт REST API (api.ts)

```
import axios from "axios";

export interface Device { id: string; name: string; type: string; status: string; }
export interface Telemetry { metricType: string; value: number; unit: string; timestamp:
string; }

const http = axios.create({ baseURL: "/api" });

// додаємо токен доступу до кожного запиту
http.interceptors.request.use((config) => {
```

```
  const token = localStorage.getItem("accessToken");
  if (token) config.headers.Authorization = "Bearer " + token;
  return config;
});

// автоматичне оновлення токена при 401 (single-flight)
let refreshing: Promise<string> | null = null;
http.interceptors.response.use(undefined, async (error) => {
  if (error.response?.status === 401 && !error.config._retry) {
    error.config._retry = true;
    refreshing = refreshing ?? refreshToken();
    const fresh = await refreshing;
    refreshing = null;
    error.config.headers.Authorization = "Bearer " + fresh;
    return http(error.config);
  }
  return Promise.reject(error);
});

async function refreshToken(): Promise<string> {
  const refresh = localStorage.getItem("refreshToken");
  const { data } = await axios.post("/api/auth/refresh", { refresh });
  localStorage.setItem("accessToken", data.accessToken);
  return data.accessToken;
}

export const api = {
  login: (email: string, password: string) =>
    http.post("/auth/login", { email, password }).then((r) => r.data),
  getDevices: () => http.get<Device[]>("/devices").then((r) => r.data),
  getDeviceTelemetry: (id: string) =>
    http.get<Telemetry[]>("/devices/" + id + "/telemetry").then((r) => r.data),
  setDeviceStatus: (id: string, status: string) =>
    http.patch("/devices/" + id, { status }).then((r) => r.data),
  getActiveAlerts: () => http.get("/alerts/active").then((r) => r.data),
};
```