Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук

Бакалаврська кваліфікаційна робота на тему:

РОЗРОБКА ДОДАТКУ «РОЗУМНИЙ БУДИНОК»

Виконав: студент 4 курсу, групи 4КН-21б
спеціальності 122 Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Обідник Р. К.
(прізвище та ініціали)
Керівник: к.т.н. ст. викл. каф. КН
Петришин С. І.
(прізвище та ініціали)
«04» червня 2025 р.

Рецензент: к.т.н. доц. каф. САІТ
Варчук І.В.
(прізвище та ініціали)
«05» червня 2025 р.

Допущено до захисту
Зав. кафедри Яровий А.А. «__»
06 червня 2025 р.

Вінниця ВНТУ – 2025 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Галузь знань – 12 «Інформаційні технології»
Спеціальність – 122 Комп'ютерні науки
Освітньо-професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
проф., д.т.н. Яровий А. А.
«05» травня 2025 р

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
Обіднику Роману Костянтиновичу

1. Тема роботи: Розробка додатку «Розумний будинок».
Керівник роботи: Петришин Сергій Іванович, к.т.н.
Затверджені наказом вищого навчального закладу "20" січня 2025 року № 97
2. Термін подання студентом роботи 30 травня 2025 року
3. Вихідні дані до роботи: -
   - Мова програмування: Dart
   - Фреймворк для клієнтської частини: Flutter
   - Серверна частина: Python (FastAPI)
   - База даних: PostgreSQL
   - Архітектура: клієнт-серверна з локальною емуляцією пристроїв
   - Емуляція пристроїв: програмна симуляція сенсорів (температура, освітлення, рух)
   - Середовище розробки: Android Studio
4. Зміст текстової частини (перелік питань, які потрібно розробити):
Вступ; Аналіз предметної області додатку «Розумний будинок»; Проєктування додатку «Розумний будинок»; Програмна реалізація додатку «Розумний будинок»; Висновки та аналіз отриманих результатів; Висновки; Список використаних джерел; Додатки.
5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень):
Структура архітектури додатку; UML-діаграми (прецедентів, послідовностей, класів); Екрани інтерфейсу користувача додатку; Приклади симуляції роботи системи; Фрагменти коду та конфігураційних файлів.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Петришин С. І., к.т.н.		

7. Дата видачі завдання 05 травня 2025 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Аналіз предметної області додатку «Розумний будинок»	05.05.2025р.	07.05.2025р.	
2	Проектування додатку «Розумний будинок»	08.05.2025р.	11.05.2025р.	
3	Програмна реалізація додатку «Розумний будинок»	12.05.2025р.	15.05.2025р.	
4	Висновки та аналіз отриманих результатів	16.05.2025р.	26.05.2025р.	
5	Оформлення додатків	27.05.2025р.	29.05.2025р.	
6	Розробка інструкції користувача	30.05.2025р.	01.06.2025р.	
7	Оформлення матеріалів до захисту БКР	02.06.2025р.	04.06.2025р.	

Студент _____ **Обідник Р. К.**
(підпис) *(прізвище та ініціали)*

Керівник роботи _____ **Петришин С. І.**
(підпис) *(прізвище та ініціали)*

АНОТАЦІЯ

УДК 004.4

Обідник Р. К. Розробка додатку «Розумний будинок». Бакалаврська дипломна робота зі спеціальності 122 – комп’ютерні науки, освітня програма – комп’ютерні науки. Вінниця: ВНТУ, 2025.

Бакалаврська дипломна робота складається з 78 сторінок формату А4, на яких є 39 рисунків, 9 таблиць, список використаних джерел містить 29 найменувань.

У даній бакалаврській дипломній роботі розглянуто процес розробки мобільного застосунку для керування системою «Розумний будинок». Проведено аналіз предметної області та аналогічних систем, визначено вимоги до функціональності майбутнього додатку. Описано структуру системи, виконано проєктування її архітектури за допомогою UML-діаграм та реалізовано ключовий функціонал засобами Flutter. Серверну частину побудовано за допомогою Python (FastAPI), а для збереження даних використано PostgreSQL.

Додаток підтримує емуляцію роботи пристроїв розумного дому (освітлення, температура, рух), що дозволяє протестувати логіку взаємодії без фізичного обладнання. Проведено тестування функціоналу, підготовлено інструкцію користувача та оформлено всі супровідні матеріали.

Ключові слова: розумний будинок, мобільний застосунок, Flutter, клієнт-сервер, FastAPI, PostgreSQL, UML, емуляція пристроїв.

ABSTRACT

Obidnyk R. K. Development of a “Smart Home” Application. Bachelor’s thesis in specialty 122 – Computer Science, educational program – Computer Science. Vinnytsia: VNTU, 2025.

The bachelor’s thesis consists of 78 A4 pages, including 39 figures and 9 tables. The list of references includes 29 sources.

This thesis examines the process of developing a mobile application for managing a smart home system. The purpose of the work is to create an intuitive and user-friendly interface that enables remote monitoring and control of environmental conditions such as lighting, temperature, and motion.

The system is built using a client-server architecture. The client-side is developed with Flutter (Dart language), while the server-side is implemented using Python (FastAPI). Device control is simulated through software-based emulation, allowing for demonstration without the need for physical equipment.

The work includes an analysis of the problem domain and existing analogs, system design using UML diagrams, implementation of core functionality, testing, and preparation of user documentation. The results of the project demonstrate the feasibility of simulating smart home device management using mobile technologies.

Keywords: smart home, mobile application, Flutter, client-server architecture, FastAPI, PostgreSQL, UML, device emulation.

ЗМІСТ

ВСТУП

Актуальність. Сучасні технології відіграють важливу роль у підвищенні комфорту, безпеки та енергоефективності житлових приміщень. Одним із напрямів такого розвитку є впровадження систем «Розумний будинок». Вони дозволяють автоматизувати керування освітленням, опаленням, системами безпеки та іншими пристроями в оселі.

Зі стрімкою популяризацією мобільних технологій та зростанням кількості "розумних" пристроїв у нашому повсякденному житті, постійно зростає потреба у створенні зручних та інтуїтивно зрозумілих мобільних додатків для ефективного управління цими девайсами. Користувачі очікують можливості контролювати освітлення, клімат, побутову техніку та інші елементи розумного будинку безпосередньо зі своїх смартфонів.

Проте, розробка та демонстрація таких додатків стикаються з суттєвими викликами. Повноцінна взаємодія з системами розумного будинку зазвичай вимагає наявності спеціального фізичного обладнання. Це може бути дорого, громіздко та ускладнює процеси розробки, тестування, демонстрації функціоналу.

Саме тому одним із найбільш ефективних підходів до вирішення цієї проблеми є використання емуляції пристроїв. Емуляція дозволяє створити віртуальне середовище, що імітує поведінку та функціональність реальних розумних пристроїв. Це означає, що розробники можуть тестувати та налагоджувати додатки без необхідності фізично підключати кожен пристрій, демонструвати повний функціонал системи будь-де і будь-коли, не транспортуючи дороге та крихке обладнання.

Отже, емуляція розумних пристроїв стає невід'ємним інструментом, що значно спрощує розробку, тестування та презентацію мобільних додатків для управління сучасними IoT-системами, дозволяючи зосередитися на користувацькому досвіді та інноваційних функціях.

Метою дослідження є розширення функціональних можливостей програмного забезпечення для керування інтелектуальними системами в рамках концепції «Розумного будинку».

Об’єктом дослідження є процес керування інтелектуальними системами в рамках концепції «Розумного будинку».

Предметом дослідження є програмні засоби для керування інтелектуальними системами в рамках концепції «Розумного будинку».

Задачі дослідження:

1. Обґрунтувати доцільність створення додатку для «розумного будинку» з можливістю симуляції пристроїв.
2. Проаналізувати існуючі аналоги та технології.
3. Спроєктувати архітектуру системи.
4. Реалізувати клієнтську та серверну частини.
5. Провести тестування та оцінити ефективність розробленого рішення.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ СТВОРЕННЯ «РОЗУМНОГО БУДИНКУ»

1.1 Поняття та принципи функціонування систем «Розумний будинок»

Системи «Розумний будинок» (англ. *Smart Home*) є сучасними технологічними рішеннями, які дозволяють автоматизувати керування побутовими процесами з метою підвищення комфорту, безпеки та енергоефективності житла. В основі таких систем лежить поєднання програмного забезпечення, мікроконтролерів, сенсорів, виконавчих механізмів і засобів комунікації між ними.

Основна ідея системи розумного будинку полягає в тому, щоб створити єдину мережу пристроїв, які можуть обмінюватися даними між собою та з центральним контролером, що керується користувачем. Це дозволяє автоматизувати такі дії, як вмикання світла при русі в кімнаті, регулювання температури в залежності від часу доби, виявлення відкритих вікон або навіть дистанційне керування побутовими приладами через мобільний додаток.

Компоненти систем «Розумний будинок» поділяються на кілька основних категорій:

- Сенсори — пристрої для збору даних про стан навколишнього середовища (температура, вологість, рух, освітленість тощо);
- Виконавчі механізми — пристрої, які реагують на команди системи (розетки, реле, моторизовані штори, освітлення);
- Комунікаційні модулі — забезпечують обмін даними (Wi-Fi, Bluetooth, Zigbee, Z-Wave);
- Центральний контролер або сервер — обробляє дані і керує логікою взаємодії;
- Інтерфейс користувача — застосунок, за допомогою якого користувач взаємодіє із системою.

Принцип роботи таких систем полягає у зборі даних із сенсорів, обробці цих даних на сервері або в контролері згідно з закладеними алгоритмами, та виконанні відповідних дій. Наприклад, якщо сенсор фіксує зниження температури нижче певного рівня — система може ввімкнути обігрівач.

Особливу популярність останніми роками отримали мобільні додатки, які дозволяють здійснювати керування системою не тільки локально, а й віддалено — з будь-якої точки світу, за умови підключення до Інтернету. Саме тому мобільний застосунок є ключовим компонентом сучасної системи «розумного будинку».

У рамках цієї дипломної роботи реалізація системи відбуватиметься у вигляді мобільного застосунку, розробленого на технології Flutter, із серверною частиною на FastAPI, та емуляцією пристроїв — що дозволить протестувати функціональність навіть без фізичного обладнання.

Таким чином, розробка подібного застосунку дозволяє не лише демонструвати сучасні технології керування побутовими приладами, а й є важливим кроком у розвитку програмних рішень у галузі автоматизації.

1.2 Аналіз існуючих рішень та аналогів

У сучасному світі існує велика кількість програмних рішень для автоматизації житла, керування пристроями та побудови інтелектуальних систем “розумний будинок”. До найпопулярніших серед них належать як комерційні екосистеми від великих IT-компаній, так і платформи з відкритим кодом для ентузіастів.

Google Home — це централізована платформа для керування пристроями розумного дому, розроблена компанією Google. Вона підтримує керування різними побутовими пристроями: лампами, розетками, термостатами, відеокамерами, аудіосистемами тощо. Користувачі мають можливість створювати сценарії автоматизації («рутини»), які виконуються за певних умов,

наприклад, автоматичне вмикання світла при поверненні додому або регулювання температури за розкладом.

Головна особливість Google Home — інтеграція з голосовим помічником Google Assistant, який дозволяє керувати пристроями за допомогою голосових команд. Додаток також забезпечує віддалений доступ: користувач може змінювати параметри чи отримувати повідомлення про події з будь-якого місця світу через інтернет. До переваг системи належить інтуїтивно зрозумілий інтерфейс, широка підтримка пристроїв різних виробників та простота налаштування. Серед недоліків — потреба у Google-акаунті, обмежена можливість налаштування для просунутих користувачів і залежність від стабільного інтернет-з’єднання (рис. 1.1).

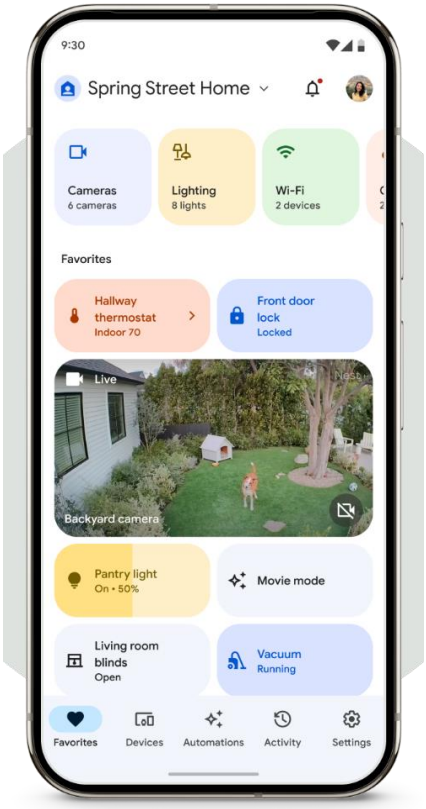


Рисунок 1.1 – Інтерфейс Google Home

Amazon Alexa — це комплексна платформа для створення екосистеми розумного дому, розроблена компанією Amazon. Основна взаємодія відбувається

через спеціальні пристрої-динаміки Amazon Echo, але керування можливо також через мобільний застосунок. Alexa дозволяє не лише керувати освітленням, температурою, розетками та іншими пристроями, але й підтримує тисячі так званих «навичок» (skills) — додаткових функцій, які можна підключати для розширення можливостей.

Важливою перевагою Alexa є голосове керування й можливість створювати складні автоматизовані сценарії. Користувач може комбінувати кілька дій у рамках однієї рутини, наприклад, вимикати все світло, знижувати температуру і зачиняти двері однією командою. Alexa інтегрується з великою кількістю пристроїв різних брендів та сервісів, включаючи потокове аудіо, календарі, нагадування, а також підтримує інтеграцію з IFTTT (If This Then That) для розширеної автоматизації.

До недоліків системи належить висока залежність від хмарних сервісів Amazon, вимога до стабільного підключення до Інтернету та ризики щодо конфіденційності даних (оскільки частина даних зберігається на сторонніх серверах) (рис. 1.2).

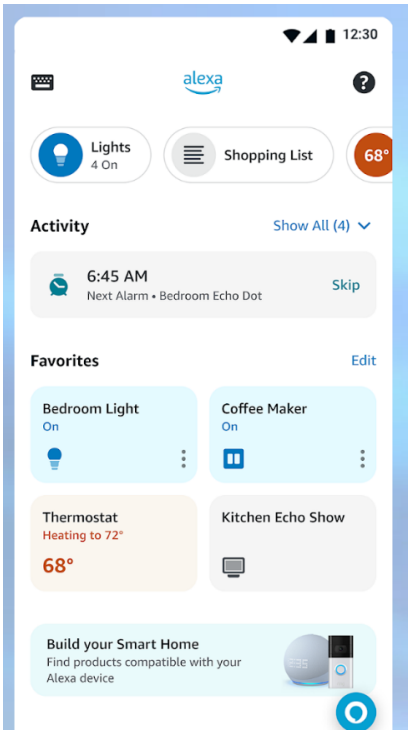


Рисунок 1.2 – Інтерфейс Amazon Alexa

Apple HomeKit — екосистема для управління пристроями розумного дому, вбудована в операційні системи iOS, iPadOS та macOS. Додаток «Дім» дозволяє об’єднувати всі сумісні пристрої (датчики, розетки, лампи, замки, штори, камери тощо) в єдину систему, якою можна керувати як вручну, так і за допомогою голосового асистента Siri.

Особливістю HomeKit є акцент на безпеку: всі пристрої повинні підтримувати стандарт Apple HomeKit, а дані користувачів захищені наскрізним шифруванням. Користувач може створювати автоматизації, які виконуватимуться за певних умов (наприклад, час доби, присутність користувача вдома, сигнали від датчиків). Система підтримує інтеграцію із домашніми хабами (HomePod, Apple TV), що дозволяє керувати будинком дистанційно.

До переваг відносять простоту налаштування для власників пристроїв Apple, безпеку та зручний інтерфейс. Недоліки: обмежена кількість сумісних пристроїв, необхідність наявності продуктів Apple та вища вартість сумісного обладнання (рис. 1.3).

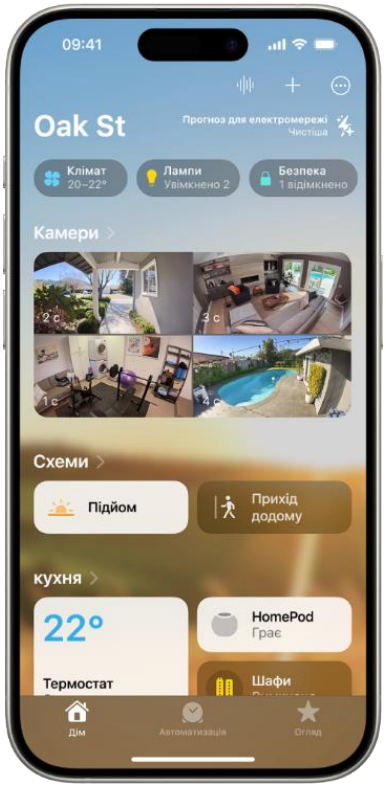


Рисунок 1.3 – Інтерфейс Apple HomeKit

Home Assistant — одна з найпотужніших і найгнучкіших платформ з відкритим кодом для побудови розумного дому. Вона працює на різних платформах (Linux, Windows, Raspberry Pi, Docker тощо) і дозволяє інтегрувати понад тисячу різних пристроїв та сервісів. Система побудована на Python, має активну спільноту, розвинуту документацію й численні додатки.

Особливість Home Assistant — можливість створювати складні сценарії автоматизації, гнучко налаштовувати логіку і зовнішній вигляд панелі керування (Lovelace UI). Всі налаштування можна виконувати як через веб-інтерфейс, так і через текстові файли (YAML). Користувачі отримують повний контроль над системою і даними, оскільки все працює локально, без необхідності відправляти дані у хмару.

До переваг відносять універсальність, велику кількість інтеграцій та можливість адаптації під будь-які потреби. Серед недоліків: складність початкового налаштування, потреба у певних знаннях програмування та мережевих технологій (рис. 1.4).

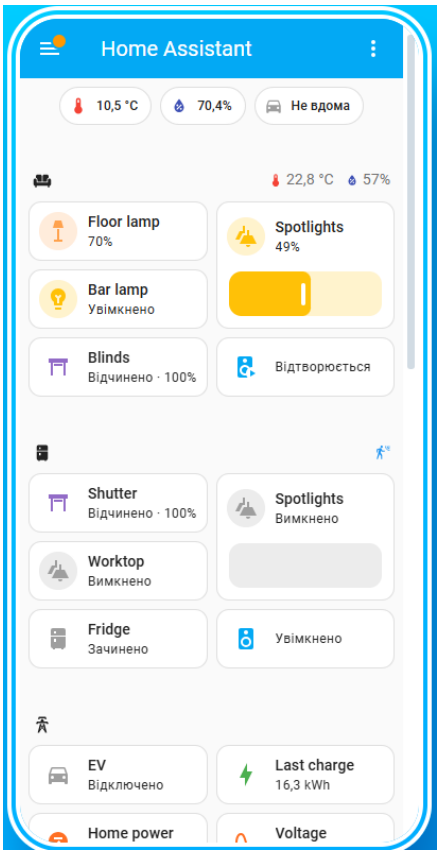


Рисунок 1.4 – Інтерфейс Home Assistant

OpenHAB — ще одна потужна платформа з відкритим кодом, орієнтована на інтеграцію пристроїв із різних екосистем та виробників. Система підтримує численні протоколи зв’язку (Zigbee, Z-Wave, MQTT, KNX тощо) та надає засоби для створення сценаріїв автоматизації, інтерфейсів керування й віджетів.

Платформа є кросплатформною і працює на різних операційних системах, у тому числі на мінікомп’ютерах. Інтерфейс OpenHAB дає можливість користувачу створювати власні панелі для керування системою, переглядати історію подій, додавати інтеграції та компоненти через маркети додатків.

Основна перевага OpenHAB — її універсальність і підтримка великої кількості протоколів. Система також підходить для досвідчених користувачів, які бажають налаштувати нестандартні сценарії. Недоліки: складніша початкова конфігурація, менш інтуїтивний інтерфейс для новачків (рис. 1.5).

Рисунок 1.5 – Інтерфейс OpenHAB

Domoticz — легка у використанні платформа з відкритим кодом, яка не потребує значних апаратних ресурсів. Вона підтримує інтеграцію із

різноманітними пристроями й датчиками (Z-Wave, 433 MHz, MQTT тощо), а також зовнішні сервіси для автоматизації.

Domoticz вирізняється простим і швидким налаштуванням, доступним через веб-інтерфейс. Система дозволяє створювати правила й сценарії для автоматизації, переглядати логи подій, інтегрувати систему з мобільними додатками чи сторонніми сервісами.

Головна перевага Domoticz — простота та низькі вимоги до «заліза», що робить її ідеальним варіантом для домашніх серверів та експериментів. До недоліків належать обмежена кількість інтерфейсних налаштувань та менша гнучкість у порівнянні з Home Assistant чи OpenHAB (рис. 1.6).

Рисунок 1.6 – Інтерфейс Domoticz

Порівняльна характеристика функціональності систем наведена у таблиці 1.1.

Таблиця 1.1 – Порівняльний аналіз функціональних можливостей систем «розумний будинок»

Назва системи	Платформа	Відкритість	Необхідність фізичних пристроїв	Гнучкість налаштувань	Легкість налаштування
Google Home	Android, iOS	Закрита	Так	Середня	Висока
Amazon Alexa	Android, iOS	Закрита	Так	Середня	Висока
Apple HomeKit	iOS, MacOS	Закрита	Так	Низька	Висока
Home Assistant	Кросплатформна	Відкрита	Так	Висока	Середня
OpenHAB	Кросплатформна	Відкрита	Так	Висока	Середня
Domoticz	Кросплатформна	Відкрита	Так	Середня	Середня
Власна розробка	Flutter, FastAPI	Відкрита	Ні (емуляція)	Висока	Висока

На основі проведеної порівняльної характеристики (таблиця 1.1) можна зробити висновок, що комерційні рішення, такі як Google Home, Amazon Alexa та Apple HomeKit, орієнтовані на масового користувача, вирізняються простотою налаштування та інтеграції, але мають закритий код і вимагають наявності сумісних фізичних пристроїв. Це обмежує можливості розширення функціоналу та не дозволяє використовувати такі системи для навчальних або експериментальних цілей без додаткових фінансових витрат.

У той же час відкриті платформи — Home Assistant, OpenHAB і Domoticz — пропонують значно ширший функціонал і гнучкість у налаштуванні, проте їх встановлення та конфігурація вимагають певних технічних знань. Всі ці системи також передбачають обов’язкове використання фізичних пристроїв для повноцінної роботи.

Власна розробка, яка пропонується у цій бакалаврській роботі, відрізняється тим, що дозволяє повністю емуляторувати роботу пристроїв, забезпечуючи максимальну гнучкість і простоту налаштування без необхідності

купувати додаткове обладнання. Це робить її ідеальним вибором для навчання, тестування та демонстрації можливостей системи «розумний будинок» у віртуальному середовищі.

1.3 Постановка задачі та формування вимог до системи

На підставі аналізу сучасних технологій розумного будинку, а також порівняння основних комерційних та відкритих платформ, визначено основні проблеми та виклики для розробників подібних рішень. Більшість існуючих систем вимагають наявності фізичних пристроїв, мають складні умови розгортання або закриту архітектуру, що обмежує можливість навчального чи демонстраційного використання без додаткових фінансових та технічних ресурсів.

Відтак, перед розробником постає задача створення системи, яка дозволить:

- забезпечити інтуїтивно зрозумілий та сучасний інтерфейс для користувача;
- керувати пристроями розумного будинку з використанням мобільного застосунку;
- реалізувати повноцінну емуляцію сенсорів та виконавчих механізмів без потреби в реальному обладнанні;
- надати можливість легкої інтеграції з реальними пристроями у майбутньому;
- забезпечити зберігання й обробку даних у безпечний спосіб, з підтримкою багатокористувацької роботи.

Головною метою дипломної роботи є розробка мобільного додатку для керування системою «Розумний будинок» з архітектурою клієнт–сервер і можливістю симуляції пристроїв. Система повинна бути зручною для користувача, розширюваною та придатною для навчального чи демонстраційного використання.

Для досягнення цієї мети необхідно виконати такі задачі:

- Провести аналіз існуючих систем і визначити їх переваги та недоліки.
- Розробити функціональні та нефункціональні вимоги до власного застосунку.
- Спроєктувати архітектуру клієнтської та серверної частин із чітко визначеними ролями.
- Реалізувати інтерфейс користувача для мобільної платформи.
- Створити серверну частину, здатну взаємодіяти з клієнтом та забезпечувати симуляцію пристроїв.
- Реалізувати можливість зберігання стану пристроїв, сценаріїв автоматизації та історії подій.
- Провести тестування працездатності та зручності використання системи.

На основі поставленої задачі та з урахуванням досвіду аналізу аналогічних систем, до розроблюваного застосунку висуваються такі вимоги. Формалізовані функціональні та нефункціональні вимоги до системи наведено в таблиці 1.2.

Таблиця 1.2 – Основні функціональні та нефункціональні вимоги до системи «розумний будинок»

№	Вимога	Тип вимоги	Опис
1	Авторизація користувача	Функціональна	Користувач може створювати обліковий запис та входити
2	Перегляд та керування пристроями	Функціональна	Можливість додавати, видаляти, керувати пристроями
3	Симуляція роботи сенсорів	Функціональна	Система емулює сигнали температури, руху, освітлення
4	Створення сценаріїв автоматизації	Функціональна	Користувач створює сценарії для автоматизації
5	Push-сповіщення	Функціональна	Система надсилає повідомлення про важливі події
6	Кросплатформність мобільного застосунку	Нефункціональна	Підтримка Android та iOS через Flutter
7	Безпечне зберігання персональних даних	Нефункціональна	Всі особисті дані користувачів захищені
8	Зберігання історії подій	Функціональна	Система фіксує всі зміни стану пристроїв
9	Легкість інтеграції нових пристроїв та сценаріїв	Нефункціональна	Гнучка архітектура для подальшого розширення системи

Таким чином, сформульовані вимоги забезпечують можливість створення універсальної, гнучкої та сучасної системи керування «розумним будинком» із використанням мобільного застосунку та серверної частини з емуляцією пристроїв. Це дозволить як демонструвати можливості сучасних технологій автоматизації побуту, так і застосовувати систему у навчальному або дослідницькому середовищі.

1.4 Висновок розділу 1

У першому розділі було здійснено всебічний аналіз предметної області та існуючих рішень у сфері систем управління «розумним будинком». Було визначено, що дана тема є актуальною з огляду на потреби автоматизації побуту, енергоефективності та зручності користувачів.

Розглянуто приклади популярних мобільних застосунків (Google Home, Amazon Alexa, Tuya Smart тощо), на основі чого виокремлено ключові функціональні можливості та визначено переваги і недоліки кожного з них.

Здійснено обґрунтований вибір стеку технологій для реалізації: клієнтську частину розроблятиметься з використанням Flutter (Dart), серверну — на FastAPI (Python), а для бази даних обрано PostgreSQL. Визначено попередню архітектуру системи, технічні вимоги, функціональні блоки та середовище розробки.

Проведений аналіз дозволив сформувати чітке уявлення про необхідну структуру, обсяг роботи та інструменти реалізації, що створює міцну основу для подальшого проєктування і створення додатку.

2 ПРОЄКТУВАННЯ ДОДАТКУ «РОЗУМНИЙ БУДИНОК»

2.1 Загальна архітектура клієнт–серверної системи

Для створення сучасної, масштабованої та гнучкої системи «Розумний дім» було обрано архітектуру «клієнт–сервер». Такий підхід дозволяє розділити програмний комплекс на дві основні частини — клієнтську (користувацьку) та серверну (логічну), що підвищує стабільність роботи, спрощує розгортання і дозволяє зручно розширювати функціонал у майбутньому.

Архітектура «клієнт–сервер» передбачає розділення програмного комплексу на дві основні, чітко визначені частини:

- клієнтська частина (користувацька);
- серверна частина (логічна).

Загалом, архітектура «клієнт–сервер» є оптимальним вибором для систем «Розумний дім», оскільки вона забезпечує надійність, гнучкість та можливість постійного розвитку, відповідаючи вимогам сучасного технологічного середовища.

Компоненти системи:

Система складається з таких основних компонентів:

1. Клієнтська частина (мобільний застосунок): створена на Flutter (Dart), забезпечує користувацький інтерфейс для керування пристроями, налаштування сценаріїв та перегляду історії подій.
2. Серверна частина: реалізована на Python-фреймворку FastAPI, виконує бізнес-логіку, керує даними, моделює пристрої, забезпечує безпеку.
3. База даних: використовується PostgreSQL для зберігання інформації про користувачів, пристрої, події, сценарії автоматизації.
4. Симульовані пристрої: програмно моделюють роботу пристроїв у режимі навчання чи тестування.

Взаємодія основних модулів системи зображена на архітектурній схемі (рис. 2.1):

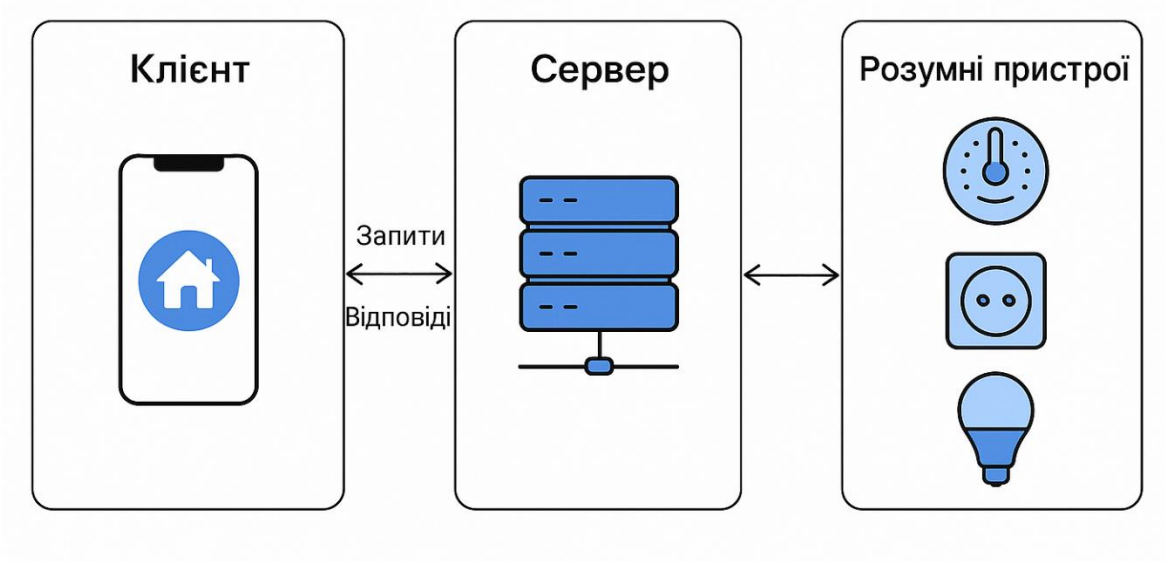


Рисунок 2.1 – Загальна архітектура клієнт–серверної системи «Розумний будинок»

На схемі показано, як клієнт надсилає запити до сервера, а сервер, у свою чергу, взаємодіє з базою даних та модулем симуляції пристроїв. Усі дії користувача, наприклад керування пристроями, створення сценаріїв чи перегляд історії, здійснюються через мобільний додаток, який отримує та відправляє дані до серверної частини.

Переваги клієнт–серверної архітектури для «Розумного дому»:

- Масштабованість: легко додавати нові функції, пристрої, користувачів.
- Безпека: персональні дані зберігаються централізовано, доступ контролюється сервером.
- Підтримка різних платформ: можливість розробки кросплатформного клієнта (Android/iOS/Web).
- Гнучкість у тестуванні: підтримка емуляції, можливість створення тестових сценаріїв.
- Простота адміністрування: централізоване оновлення логіки і налаштувань.

Приклад сценарію взаємодії

У таблиці 2.1 наведено типовий сценарій взаємодії користувача, клієнтського застосунку і серверної частини системи.

Таблиця 2.1 – Приклад сценарію взаємодії клієнта та серверної частини системи

Крок	Дія користувача	Взаємодія клієнта із сервером	Дія системи
1	Авторизація	Надсилає логін і пароль	Сервер перевіряє, повертає токен
2	Додає пристрій у кімнату	Передає дані про пристрій	Сервер зберігає у БД
3	Керує пристроєм	Надсилає команду (вкл./викл./регулювання)	Сервер змінює стан, оновлює БД
4	Створює сценарій автоматизації	Передає параметри сценарію	Сервер зберігає сценарій
5	Переглядає історію подій	Запитує історію дій	Сервер повертає журнал подій

Обрана архітектура забезпечує розподіл обов’язків, надійність та масштабованість, зручність користування і подальшого розвитку системи. Такий підхід ідеально підходить для навчальних, демонстраційних і практичних цілей, що відповідає темі та вимогам бакалаврської кваліфікаційної роботи.

2.2 Вибір засобів реалізації

Від правильного вибору технологій та інструментів залежить не лише функціональність, а й надійність, масштабованість і підтримка проєкту в майбутньому. Оскільки додаток «Розумний дім» розробляється як навчальний,

демонстраційний і потенційно розширюваний, критеріями відбору інструментів стали: сучасність, підтримка спільнотою, легкість старту та кросплатформність.

Аналіз та порівняння основних технологій

Для кожної компоненти системи (клієнт, сервер, база даних, сповіщення) було розглянуто кілька популярних альтернатив. Оцінювання проводилося за критеріями: легкість розгортання, продуктивність, підтримка мобільних платформ, можливість масштабування, документованість та навчальна цінність.

Порівняльна характеристика основних сучасних технологій, які розглядалися для розробки системи, наведена в таблиці 2.2.

Таблиця 2.2 – Порівняльна характеристика технологій

Компонент	Варіанти	Обране рішення	Аргументація вибору
Мобільний клієнт	Android (Kotlin), iOS (Swift), React Native, Flutter	Flutter (Dart)	Єдина кодова база для Android/iOS, активна спільнота, простота UI
Сервер	Node.js, Flask, Django, FastAPI	FastAPI (Python)	Висока швидкість, простота REST API, підтримка асинхронності
База даних	SQLite, MySQL, PostgreSQL	PostgreSQL	Надійність, гнучка робота зі зв’язками, зручний для складних структур
Push-сповіщення	OneSignal, Firebase	Firebase	Легкість інтеграції, документація, підтримка Flutter
Хмарне розгортання	Heroku, DigitalOcean, AWS	Heroku	Безкоштовний старт, автоматизація деплойменту

Чому саме Flutter/Dart для клієнтської частини?

- Кросплатформність: один код працює на Android та iOS.

- Простота створення UI: багато готових віджетів і гнучке налаштування стилю.
- Активна спільнота: регулярні оновлення, велика база знань.
- Можливість тестування без заліза: зручно робити навчальні симуляції, не маючи фізичних пристроїв.

Вигляд логотипа Flutter показано на рисунку 2.2.

Рисунок 2.2 – Логотип Flutter

Чому FastAPI для серверної частини?

- Висока продуктивність: побудований на асинхронних запитах.
- Докладна документація: автоматична генерація Swagger UI.
- Гнучкість: легко додавати нові маршрути, підтримує WebSocket для “живого” зв’язку з клієнтом.
- Навчальна цінність: синтаксис Python зрозумілий для початківців і досвідчених розробників.

Схематичне зображення FastAPI наведено на рисунку 2.3.

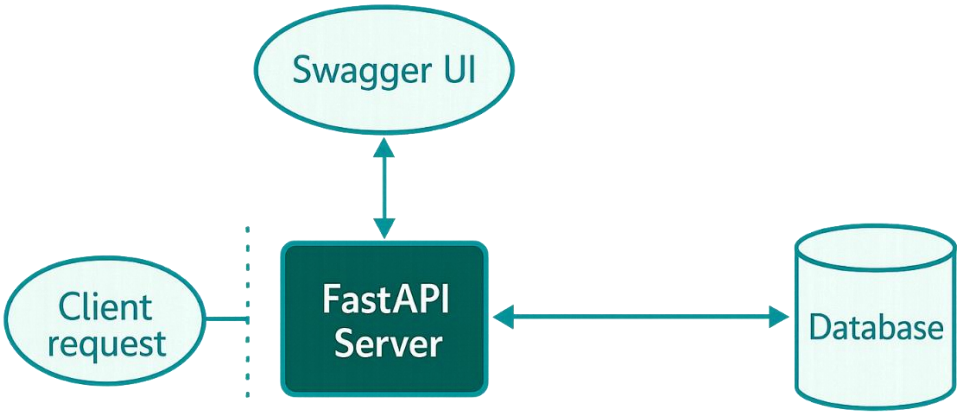


Рисунок 2.3 – Схематична структура FastAPI

Чому саме PostgreSQL для бази даних?

- Розширюваність: підтримує складні реляційні зв’язки, сховища JSON, геодані, історію.
- Сумісність: легко під’єднується до FastAPI.
- Стійкість до навантаження: підходить для будь-яких обсягів даних — від домашніх проєктів до комерційних систем.
- Безкоштовність і відкритість коду.

Вигляд логотипа PostgreSQL зображено на рисунку 2.4.

Рисунок 2.4 – Логотип PostgreSQL

Схематична структура взаємодії

На рисунку 2.5 зображено, як компоненти системи взаємодіють між собою у рамках розробки та використання.

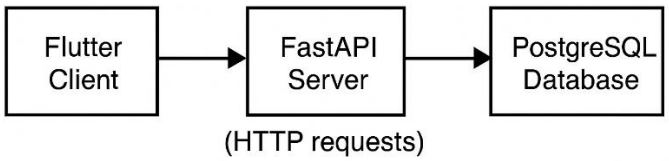


Рисунок 2.5 – Схема взаємодії Flutter-клієнта, FastAPI-сервера та PostgreSQL БД

Узагальнений технологічний стек системи представлено в таблиці 2.3.

Таблиця 2.3 – Технологічний стек системи

Рівень	Технологія	Опис
Мобільний клієнт	Flutter (Dart)	Кросплатформний мобільний додаток
Сервер	FastAPI (Python)	REST API, логіка системи
База даних	PostgreSQL	Сховище даних, історія
Сповіщення	Firebase	Push-повідомлення
Хостинг	Heroku	Хмарне розгортання

Фрагмент інтерфейсу мобільного застосунку наведено на рисунку 2.6.

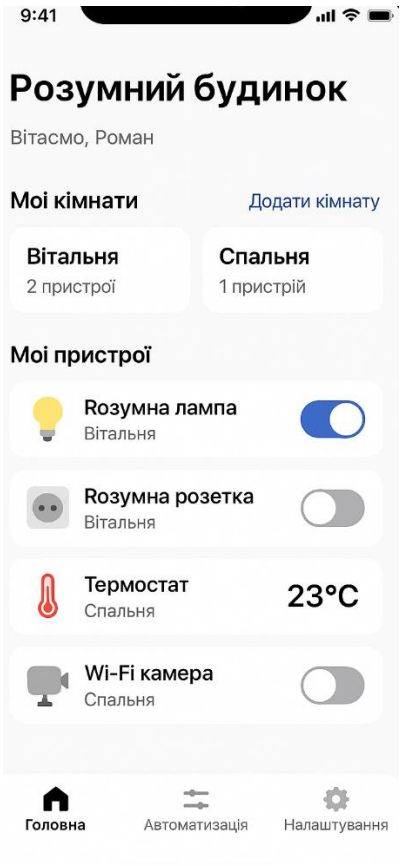


Рисунок 2.6 – Фрагмент інтерфейсу мобільного застосунку

Вибраний набір технологій відповідає сучасним вимогам до надійності, зручності, гнучкості і навчанню, а також дозволяє швидко протестувати систему та внести зміни без серйозних витрат.

2.3 Проєктування інтерфейсу користувача

Інтерфейс користувача (UI) мобільного застосунку «Розумний будинок» є без перебільшення ключовим елементом взаємодії користувача з системою. Це не просто набір кнопок та меню; це міст, що з'єднує складні технології розумного дому з повсякденними потребами та бажаннями людини. Його дизайн та функціональність безпосередньо впливають на те, наскільки ефективно, приємно та зручно користувачі зможуть керувати своїм домом.

Вимоги до інтерфейсу

– Інтерфейс повинен забезпечувати:

– зручну навігацію між основними екранами;
– швидкий доступ до керування пристроями;
– інтуїтивну реєстрацію, вхід і редагування профілю;
– підтримку темного та світлого режимів;
– адаптивність під різні екрани смартфонів.

Інтерфейс створено за допомогою Flutter — кросплатформного фреймворку, який дозволяє створювати сучасні, анімовані та адаптивні UI-компоненти з мінімальними зусиллями.

Особливо важливо, щоб усі ці якості зберігалися навіть для користувачів без технічної підготовки. Люди, які не є експертами в галузі технологій, повинні мати змогу легко та впевнено керувати своїм розумним будинком. Це вимагає від розробників глибокого розуміння потреб кінцевого користувача та постійного тестування інтерфейсу на різних групах.

Порівняльні приклади макетів наведено на рисунках 2.7–2.11.

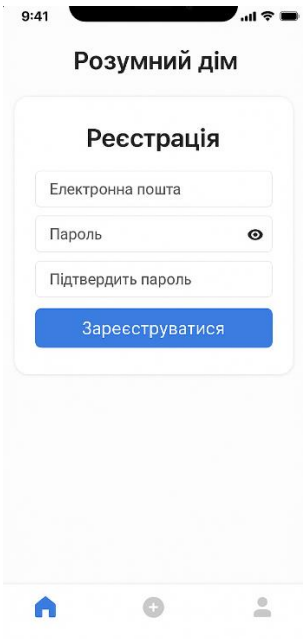


Рисунок 2.7 – Екран реєстрації користувача

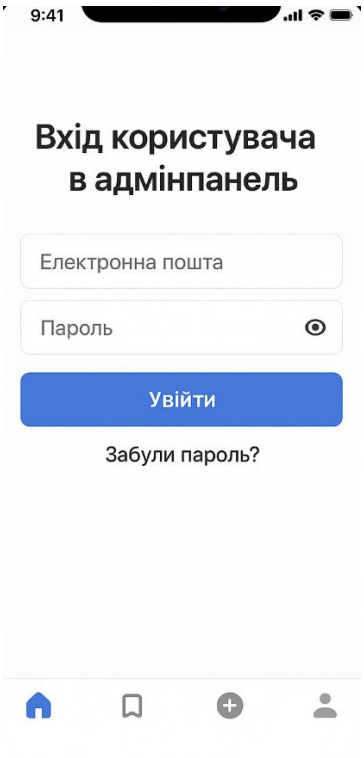


Рисунок 2.8 – Екран входу в систему

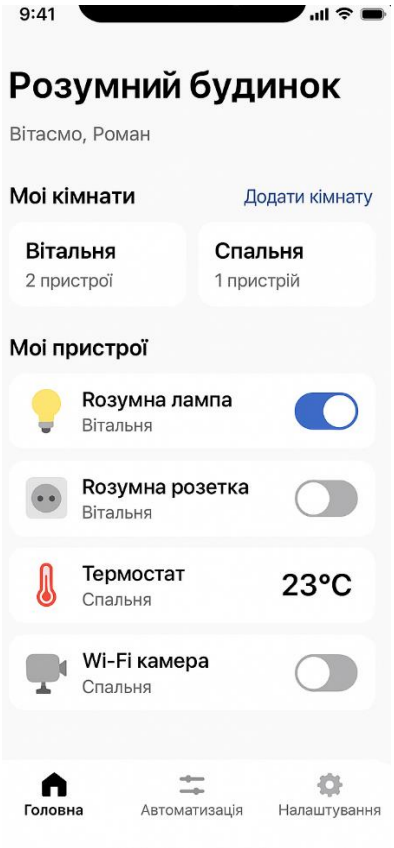


Рисунок 2.9 – Головна сторінка додатку «Розумний будинок»

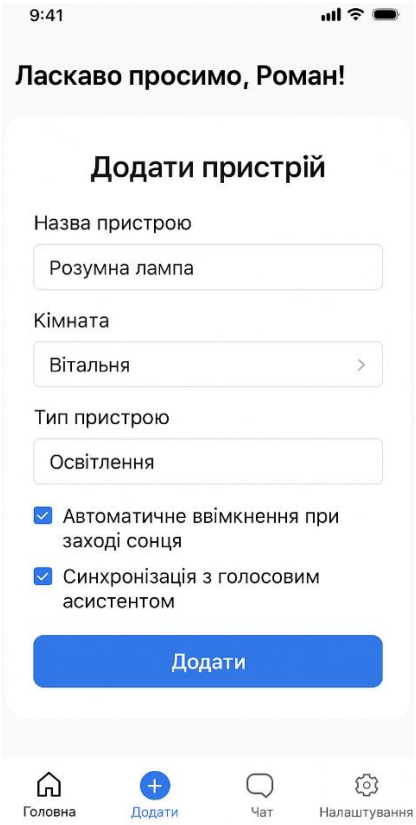


Рисунок 2.10 – Інтерфейс додавання нового пристрою

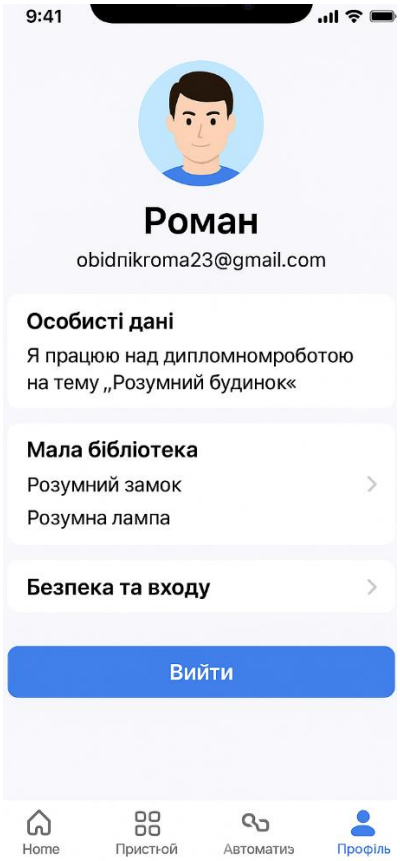


Рисунок 2.11 – Інтерфейс профілю користувача

Основні UI-компоненти зведені у таблицю 2.4

Таблиця 2.4 – Основні елементи користувацького інтерфейсу

Елемент інтерфейсу	Опис
TextField	Поле для введення (електронна пошта, пароль)
ElevatedButton	Основна кнопка дій (реєстрація, вхід, додати)
BottomNavigationBar	Нижнє меню для навігації між розділами
Switch / Checkbox	Керування опціями пристрою
Card	Відображення блоку з пристроєм
AppBar	Верхній заголовок з назвою сторінки

Інтерфейс реалізовано згідно з принципами UI/UX:

- Простота – немає зайвих елементів або кольорів.
- Єдність стилю – усі екрани оформлені в одному візуальному стилі.
- Інтуїтивність – всі кнопки мають зрозумілі підписи та іконки.
- Адаптивність – підлаштування під розміри екрана.

Інтерфейс користувача розроблений так, щоб забезпечити просту та комфортну роботу з додатком. Завдяки використанню Flutter, застосунок має сучасний вигляд і функціональність, яка відповідає актуальним вимогам до мобільних рішень для «розумного дому».

2.4 Проєктування бази даних

Для ефективного збереження та обробки інформації у мобільному додатку «Розумний будинок» необхідна надійна, структурована та масштабована база даних. Основні вимоги до неї:

– підтримка реляційної моделі;
– цілісність та унікальність даних;
– можливість масштабування;
– простота в інтеграції з сервером FastAPI.

Зважаючи на ці фактори, для реалізації використовується PostgreSQL — потужна СУБД з відкритим кодом, що забезпечує високу надійність, транзакційність та гнучке управління зв’язками між таблицями (рис. 2.12).

У базі даних передбачено наступні основні таблиці (табл. 2.5):

– Users – зберігає інформацію про користувачів.
– Devices – містить дані про пристрої, що підключені до системи.
– Rooms – кімнати, у яких розміщено пристрої.
– DeviceTypes – категорії пристроїв (освітлення, клімат, охорона тощо).
– Settings – індивідуальні параметри автоматизації для кожного пристрою.
– Logs – журнал подій (історія увімкнень/вимкнень).

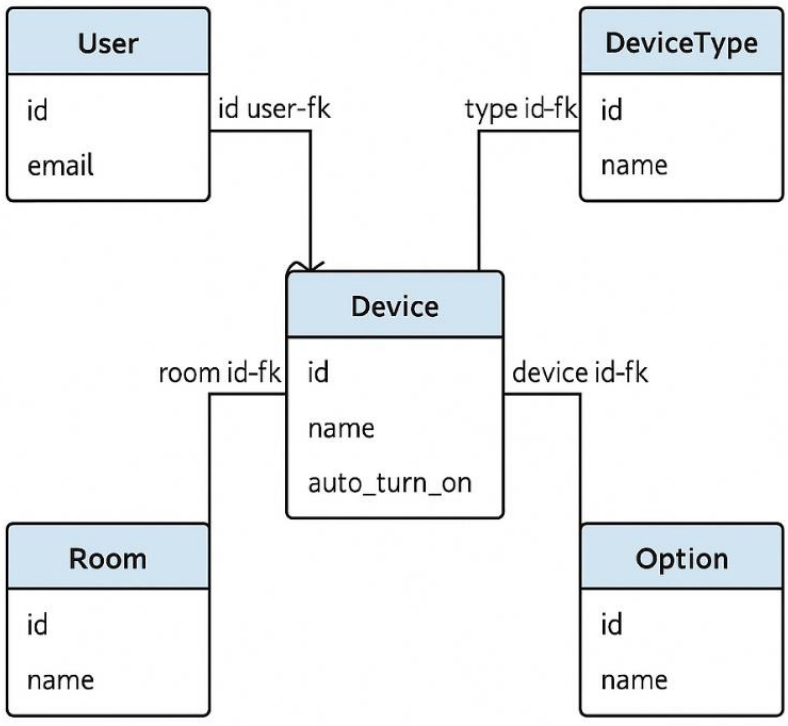


Рисунок 2.12 – Схема зв’язків між таблицями бази даних (ER-діаграма)

Таблиця 2.5 – Основні таблиці бази даних

Таблиця	Ключові поля	Призначення
Users	id, email, password, name	Дані користувача
Devices	id, name, room_id, type_id, user_id	Інформація про кожен пристрій
Rooms	id, name, user_id	Перелік кімнат для користувача
DeviceTypes	id, name	Категорії пристроїв
Settings	id, device_id, param_name, param_value	Індивідуальні налаштування пристроїв
Logs	id, device_id, timestamp, action	Історія подій (вкл./викл./помилка тощо)

Проєктування бази даних є одним із найважливіших етапів у розробці будь-якої складної програмної системи, зокрема мобільного застосунку «Розумний будинок». Цей процес передбачає чітке визначення всіх необхідних сутностей (тобто об'єктів або категорій інформації, які потрібно зберігати) та їхніх взаємозв'язків. Ретельне проєктування гарантує структуроване зберігання даних, що є запорукою ефективної роботи системи, а також забезпечує можливість подальшого масштабування. Це означає, що база даних зможе обробляти зростаючі обсяги інформації та підтримувати більшу кількість користувачів і пристроїв у майбутньому без значних змін у своїй архітектурі.

Для реалізації цієї системи була обрана система управління базами даних (СУБД) PostgreSQL.

Завдяки своїм характеристикам, PostgreSQL є критично важливим елементом для стабільної та ефективної роботи мобільного застосунку в реальному часі. Він забезпечує швидкий доступ до інформації про стан

пристроїв, оперативне виконання команд та надійне зберігання всіх даних, що є фундаментом для комфортного та безпечного управління розумним будинком.

2.5 Висновок розділу 2

У другому розділі було здійснено поетапне проєктування клієнт-серверної системи управління «розумним будинком». Спочатку було описано загальну архітектуру взаємодії між компонентами системи: мобільним клієнтом, серверною частиною та базою даних.

Детально обґрунтовано вибір засобів реалізації: фреймворку Flutter для створення зручного мобільного інтерфейсу, FastAPI для розробки гнучкого та продуктивного API, PostgreSQL як основи для надійного зберігання даних. У підрозділі 2.3 спроєктовано інтерфейс користувача із використанням принципів UX/UI-дизайну та згенеровано відповідні макети для ключових екранів. У підрозділі 2.4 представлено логічну модель бази даних та ER-діаграму, що демонструє зв’язки між основними сутностями.

Таким чином, результати другого розділу формують повну проєктну документацію, яка дозволяє перейти до наступного етапу — реалізації застосунку.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ДОДАТКУ «РОЗУМНИЙ БУДИНОК»

3.1 Розробка клієнтської частини (Flutter)

Розробка клієнтської частини системи «Розумний будинок» – тобто того інтерфейсу, з яким безпосередньо взаємодіє користувач – була реалізована за допомогою сучасного та високоефективного фреймворку Flutter. Flutter, розроблений Google, є потужним інструментом для створення мобільних додатків, що пропонує низку значних переваг, особливо для проєктів, які вимагають присутності на декількох платформах.

Ключова особливість Flutter полягає в його здатності створювати кросплатформенні мобільні додатки для Android та iOS з єдиною кодовою базою. Це означає, що розробникам не потрібно писати окремий код для кожної операційної системи. Замість цього, єдиний набір коду використовується для збірки нативних додатків, які бездоганно працюють як на пристроях Android, так і на iPhone.

Вибір інструментів

- Flutter був обраний у якості основного інструменту з наступних причин:
- Кросплатформеність: один код для Android та iOS;
- Висока швидкодія завдяки компіляції у рідний код;
- Велика кількість готових віджетів та можливість гнучкої кастомізації інтерфейсу;
- Підтримка гнучкої архітектури з використанням патернів BLoC, Provider або Riverpod;
- Швидкий Hot Reload, що значно полегшує розробку.

Інтегроване середовище розробки: Android Studio з плагінами для Flutter та Dart.

Основна структура інтерфейсу

Головні екрани додатку реалізовано у відповідності до функціональних вимог системи «Розумний будинок». Нижче подано опис основних частин інтерфейсу:

1. Екран реєстрації користувача містить поля введення електронної пошти, пароля та кнопки підтвердження. При валідації даних здійснюється запит до серверу FastAPI для реєстрації.

2. Екран входу (автентифікація) реалізовано механізм збереження токена автентифікації за допомогою пакету shared_preferences, що дозволяє уникати повторного входу.

3. Головний екран виводить список підключених пристроїв користувача з можливістю перегляду їх статусу та вмикання/вимикання. Здійснюється періодичне оновлення статусів через API.

4. Екран додавання пристрою дозволяє вказати тип пристрою, його назву та кімнату. Запит відправляється до серверної частини, де зберігається у базі даних PostgreSQL.

5. Екран профілю користувача містить інформацію про акаунт, налаштування та можливість виходу із системи.

6. Навігація в додатку реалізовано нижню панель навігації (BottomNavigationBar), яка дозволяє швидко перемикатися між головними екранами: Головна, Додати пристрій, Профіль.

Вибір Flutter для клієнтської частини системи «Розумний будинок» є стратегічним рішенням, що забезпечує ефективність розробки, уніфікований досвід для користувачів та оптимізацію ресурсів на підтримку проєкту.

У реалізації клієнтської частини було використано низку сторонніх пакетів з pub.dev:

- http – для відправлення запитів до FastAPI;
- provider – для управління станом додатку;
- flutter_secure_storage – для безпечного зберігання токена;
- shared_preferences – для зберігання базових налаштувань;

- fluttertoast – для показу повідомлень користувачеві;
- form_field_validator – для валідації форм.

Приклад коду: запит до серверу

dart

```
final response = await http.post(
  Uri.parse('http://yourapi.com/login'),
  body: {
    'email': emailController.text,
    'password': passwordController.text,
  },
);

if (response.statusCode == 200) {
  final token = jsonDecode(response.body)['token'];
  await storage.write(key: 'auth_token', value: token);
}
```

Особливості реалізації:

- всі екрани спроєктовані згідно з принципами адаптивного дизайну, що дозволяє коректне відображення на екранах різного розміру;
- для взаємодії з сервером реалізовано асинхронні функції, які не блокують основний потік;
- у разі відсутності підключення до інтернету, користувач отримує відповідне повідомлення.

Безпека (табл. 3.1)

- дані користувача зберігаються в зашифрованому сховищі;
- всі запити здійснюються через HTTPS (на етапі розробки – через HTTP на локальному сервері);
- передбачено механізм виходу з додатку, що очищає локальні дані.

Таблиця 3.1 – Огляд застосованих пакетів у Flutter-проєкті

№ з/п	Назва пакету	Версія	Призначення
1	http	^0.13.6	Використовується для виконання HTTP-запитів до API серверу.
2	provider	^6.0.5	Управління станом додатку.
3	flutter_svg	^2.0.7	Відображення SVG-іконок у інтерфейсі користувача.
4	shared_preferences	^2.2.1	Локальне збереження даних (токенів, налаштувань).
5	go_router	^10.2.0	Маршрутизація між екранами у додатку.
6	flutter_hooks	^0.20.1	Альтернатива StatefulWidget для гнучкішого управління станом.
7	flutter_dotenv	^5.1.0	Завантаження конфігурацій з .env файлів.
8	cupertino_icons	^1.0.6	Системні іконки iOS стилю.

3.2 Розробка серверної частини (FastAPI)

Для реалізації серверної частини системи «Розумний будинок» було обрано веб-фреймворк FastAPI, який забезпечує високу продуктивність, простоту у використанні та гнучкість при створенні RESTful API. Завдяки асинхронній архітектурі та підтримці стандарту OpenAPI, FastAPI є сучасним рішенням для побудови надійних серверних додатків.

Основні функції серверної частини:

1. Обробка HTTP-запитів від клієнтського додатку (Flutter);
2. Реєстрація та автентифікація користувачів;
3. Керування підключеними пристроями;

4. Зберігання даних у базі PostgreSQL;

5. Взаємодія з системою емуляції пристроїв (MQTT / віртуальні ендпоїнти);

6. Захист та шифрування даних.

Структура проєкту FastAPI

plaintext

КопіюватиРедагувати

smart_home_api/

main.py

models/

user.py

device.py

routes/

auth.py

devices.py

schemas/

user.py

device.py

database/

session.py

utils/

security.py

requirements.txt

Таке структурування дозволяє розділити логіку додатку на окремі модулі та забезпечити легкість у супроводі та масштабуванні системи.

Реєстрація та авторизація

Використовується механізм JWT (JSON Web Token) для авторизації користувачів. Токен генерується при успішному вході і надсилається клієнту. Кожен подальший запит має містити токен у заголовку Authorization.

Приклад ендпоїнту авторизації:

```
python
@router.post("/login")
def login(request: OAuth2PasswordRequestForm = Depends(), db: Session = Depends(get_db)):
    user = authenticate_user(request.username, request.password, db)
    if not user:
        raise HTTPException(status_code=400, detail="Invalid credentials")
    access_token = create_access_token(data={"sub": user.email})
    return {"access_token": access_token, "token_type": "bearer"}
```

API для роботи з пристроями

Реалізовано CRUD-функціонал:

– GET /devices – отримати список пристроїв користувача;

– POST /devices – додати новий пристрій;

– PUT /devices/{id} – оновити дані пристрою;

– DELETE /devices/{id} – видалити пристрій.

З'єднання з базою PostgreSQL (рис. 3.5).

FastAPI використовує ORM-бібліотеку SQLAlchemy, яка забезпечує взаємодію з базою даних на рівні Python-об’єктів (рис. 3.6). Підключення до бази:

```
python
DATABASE_URL = "postgresql://user:password@localhost/smart_home"
engine = create_engine(DATABASE_URL)
SessionLocal = sessionmaker(bind=engine)
```

Для повноцінної демонстрації та тестування системи «Розумний дім» без потреби у фізичному обладнанні, центральну роль відіграє спеціалізований модуль емуляції. Цей модуль є справжнім віртуальним оркестром, що дозволяє відтворювати роботу реальних пристроїв з високим ступенем достовірності.

В основі функціонування модуля лежить можливість кожного емульованого пристрою отримувати та змінювати свій статус. Це означає, що віртуальне світло може бути "увімкнене" або "вимкнене", віртуальний термостат може "змінювати" температуру, а віртуальний датчик руху – "фіксувати" рух. Користувачі мобільного застосунку можуть взаємодіяти з цими емульованими пристроями так само, як і з реальними, надсилаючи команди та отримуючи

зворотний зв'язок (табл. 3.2). Це створює ілюзію повної функціональності системи, що є безцінним для розробки, тестування та демонстрації (рис. 3.7).

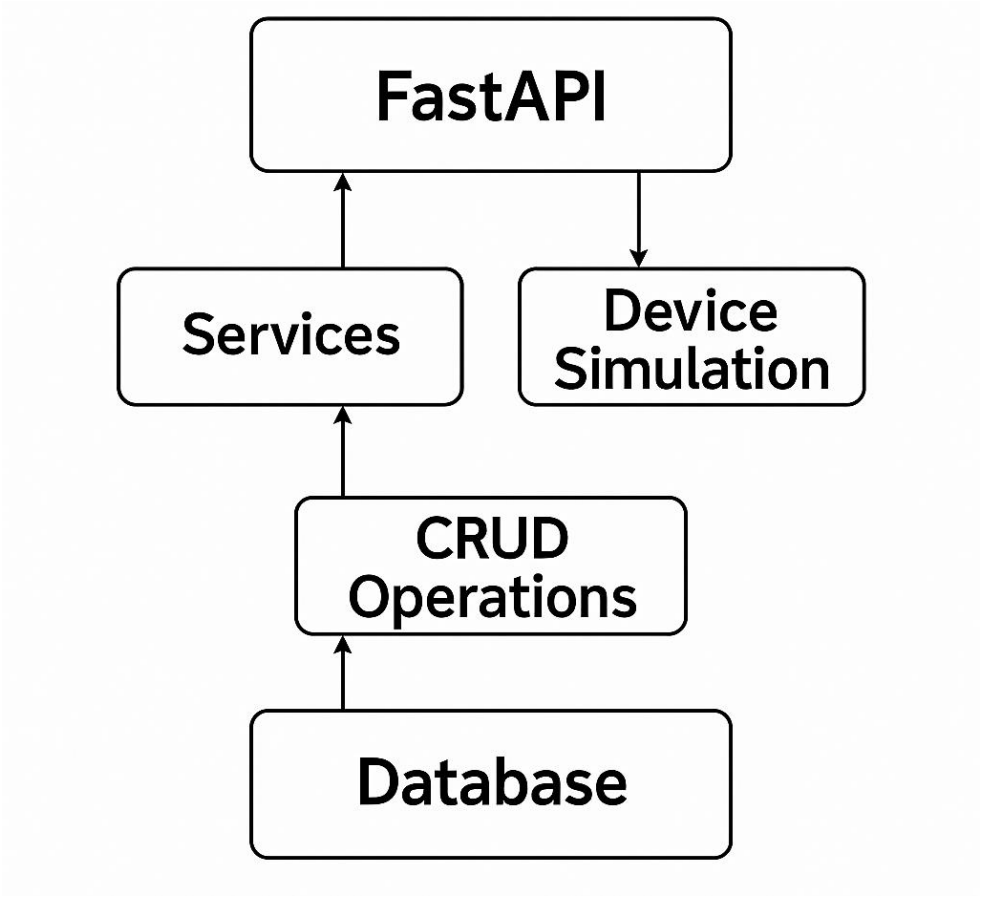


Рисунок 3.5 – Схематична структура серверної частини FastAPI

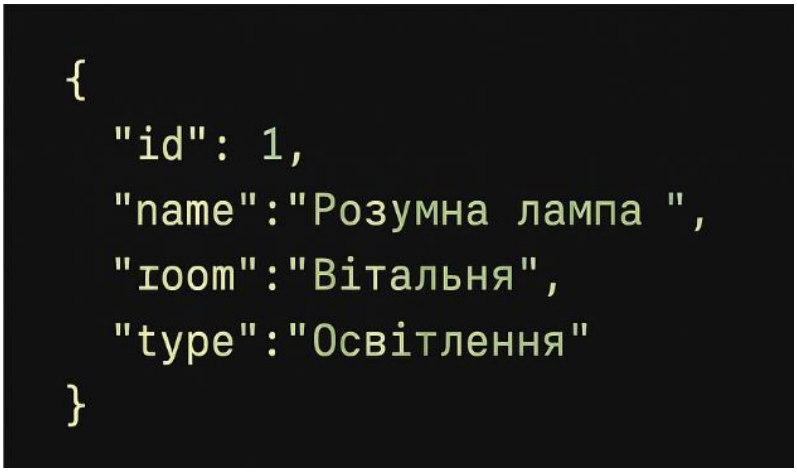


Рисунок 3.6 – Приклад відповіді API при додаванні пристрою

Рисунок 3.7 – Послідовність автентифікації користувача

Таблиця 3.2 – Основні ендпоїнти API та їх призначення

Ендпоїнт	Метод	Опис
/users/register	POST	Реєстрація нового користувача
/users/login	POST	Авторизація користувача та отримання токена
/users/me	GET	Отримання даних поточного авторизованого користувача
/devices	GET	Отримання списку пристроїв користувача
/devices/{id}	PUT / DELETE	Оновлення або видалення пристрою за ID
/devices/toggle/{id}	POST	Зміна стану (вкл/викл) пристрою

Усі паролі зберігаються у базі у вигляді хешів (з використанням bcrypt). Всі маршрути, окрім реєстрації та входу, захищені авторизацією. Також впроваджено механізм обмеження запитів (Rate Limiting) для запобігання атакам типу brute-force.

Для перевірки роботи ендпоїнтів використовувався Postman та Swagger UI, який автоматично генерується у FastAPI.

У результаті розробки серверної частини створено повноцінний API, що забезпечує інтеграцію між клієнтським додатком і базою даних, а також дозволяє гнучко керувати обліковими записами та пристроями системи «Розумний будинок».

3.3 Реалізація емуляції пристроїв

Оскільки під час розробки додатку «Розумний будинок» не було фізичного доступу до реальних пристроїв, важливим завданням стало створення емулятора пристроїв, що імітує їхню поведінку у віртуальному середовищі. Такий підхід дозволив розробити та протестувати основну функціональність системи без використання апаратного забезпечення.

Мета емуляції

- Відлагодження логіки клієнтського додатку;
- Перевірка обробки API-запитів сервером;
- Моделювання реального сценарію взаємодії користувача з пристроями;
- Тестування системи автоматичного оновлення статусу пристроїв.

Принцип роботи

Кожен віртуальний пристрій має:

- Унікальний ідентифікатор (UUID),
- Тип (лампа, розетка, датчик температури),
- Поточний стан (ввімкнено/вимкнено, значення температури тощо),
- Поведінку (змінює свій стан з певною періодичністю або за запитом).

Емулятор реалізовано як окремий Python-модуль, що працює асинхронно паралельно з основним сервером.

Структура модулю емуляції

plaintext

КопіюватиРедагувати

emulator/

__init__.py

manager.py # керування списком пристроїв

simulator.py # логіка генерації подій

devices_template.json

Основна логіка емуляції

python

КопіюватиРедагувати

```
import asyncio
import random

devices = {
    "lamp-1": {"type": "lamp", "state": False},
    "temp-1": {"type": "sensor", "value": 22.5},
}

async def simulate_devices():
    while True:
        for device_id, device in devices.items():
            if device["type"] == "lamp":
                device["state"] = random.choice([True, False])
            elif device["type"] == "sensor":
                device["value"] = round(random.uniform(20.0, 25.0), 1)
        await asyncio.sleep(5)
```

Емулятор кожні 5 секунд оновлює стан пристроїв та зберігає їх у базу даних через сервер FastAPI, використовуючи внутрішні API-запити (рис. 3.8).

Зв’язок із сервером

Емуляція виконує запити до внутрішніх маршрутів типу (рис. 3.9):

http

POST /api/emulate/device_status

Це дозволяє безпосередньо впливати на модель пристрою в базі PostgreSQL.

Користь для тестування

Завдяки реалізованому симулятору (табл. 3.3) розробники отримали можливість:

– Побачити в реальному часі зміну статусу пристроїв у додатку (Flutter);
– Перевірити стабільність роботи API при частих змінах даних;
– Підготуватися до подальшої інтеграції з реальним апаратним забезпеченням (наприклад, через MQTT).

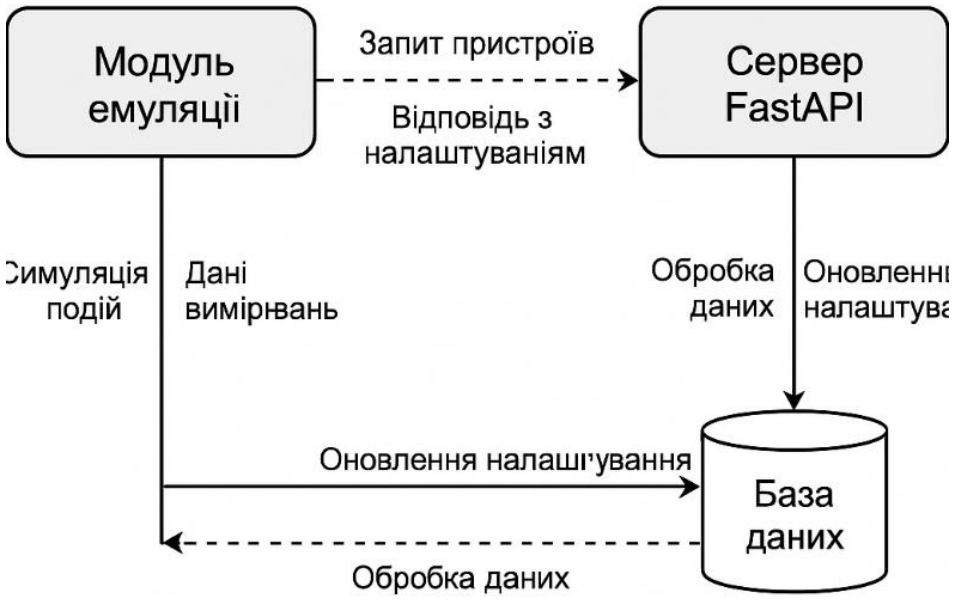


Рисунок 3.8 – Схема взаємодії модуля емуляції з сервером FastAPI

```
INFO 2025-06-01T12:00:02.630360
INFO [emulator:status]  Changed
status of 'Розумна лампа' in 'Вітальня'
to on
INFO 2025-06-01T12:00:03.632859
INFO [emulator:status]  Changed
status of 'Розумна розетка' in
Вітальня' to off
INFO 2025-06-01T12:00:04.634606
INFO [emulator:status]  Changed
status of 'Термостат' in 'Спальня' to
20
```

Рисунок 3.9 – Приклад логів зміни статусу пристроїв

Таблиця 3.3 – Типи віртуальних пристроїв та їхня поведінка

Тип пристрою	Опис	Типова поведінка
Сенсор температури	Віртуальний пристрій, який імітує зміни температури у приміщенні	Передає значення температури кожні 10 секунд, змінюючи значення в межах ±2°C
Сенсор руху	Детектор, який імітує виявлення руху	Надсилає сигнал при виявленні руху (випадково через певний інтервал)
Розумна лампа	Світильник з можливістю дистанційного вмикання/вимикання	Змінює свій стан при отриманні команди з API
Розетка	Імітує вмикання та вимикання електроживлення	Увімкнення/вимкнення по команді; може включати таймер
Датчик вологості	Вимірює вологість повітря	Надсилає оновлення кожні 15 секунд з варіацією вологості

Реалізація модуля емуляції пристроїв стала важливим етапом у розробці системи «Розумний будинок». Вона дозволила перевірити повноцінну роботу архітектури клієнт–сервер–база даних без наявності реального обладнання. У майбутньому такий підхід може бути масштабований для моделювання тисяч віртуальних пристроїв, а також використаний як тестове середовище при впровадженні нових функцій.

3.4 Інтеграція з базою даних PostgreSQL

База даних є центральним компонентом системи «Розумний будинок», оскільки вона зберігає всю ключову інформацію: дані користувачів, стан пристроїв, логування подій, параметри конфігурацій тощо. У проєкті використовується реляційна система управління базами даних PostgreSQL, яка забезпечує надійність, масштабованість та безпечну обробку запитів.

Обґрунтування вибору PostgreSQL

PostgreSQL було обрано з наступних причин:

- Висока продуктивність при роботі з великими обсягами даних;
- Підтримка транзакцій і зв’язків між таблицями (foreign keys);
- Гнучкий синтаксис запитів SQL;
- Простота інтеграції з Python через SQLAlchemy;
- Наявність розширень (PostGIS, pg_cron тощо);
- Відкрите програмне забезпечення з активною спільнотою.

Структура бази даних

Розроблена база даних містить кілька ключових таблиць (рис. 3.10):

- users – зберігає інформацію про користувачів;
- devices – містить перелік пристроїв, прив’язаних до користувача;
- device_status – оновлювані параметри кожного пристрою;
- events – журнал змін стану пристроїв;
- settings – конфігураційні параметри пристрою;
- logs – логування всіх дій користувача.

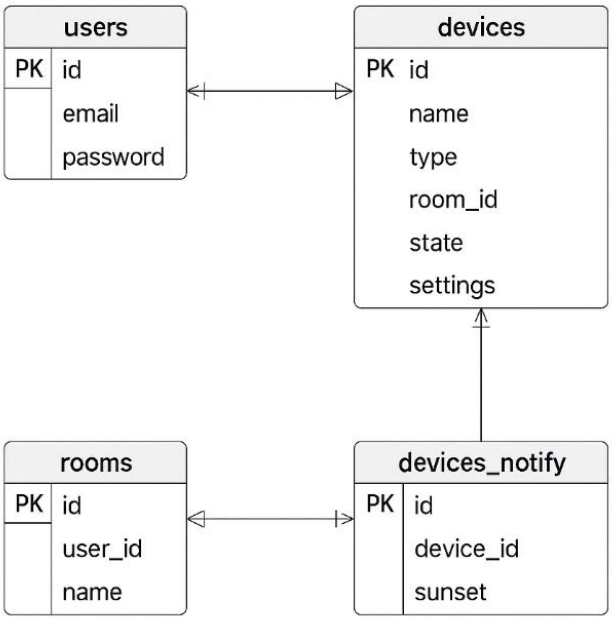


Рисунок 3.10 – ER-діаграма бази даних

Приклад створення таблиці

sql

```
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    email VARCHAR(100) UNIQUE NOT NULL,
    hashed_password TEXT NOT NULL,
    full_name VARCHAR(100),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Інтеграція через SQLAlchemy

Для взаємодії з базою даних використовується ORM SQLAlchemy, що дозволяє описувати таблиці через Python-класи, не використовуючи сирий SQL у коді додатку (рис. 3.11).

Приклад моделі пристрою:

python

```
class Device(Base):
```

```
__tablename__ = 'devices'
id = Column(Integer, primary_key=True, index=True)
name = Column(String)
user_id = Column(Integer, ForeignKey("users.id"))
status = Column(String)
created_at = Column(DateTime, default=datetime.utcnow)
```

Ініціалізація сесії:

python

```
engine = create_engine(DATABASE_URL)
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
```

Обробка запитів із FastAPI

При зверненні до API (наприклад, GET /devices), FastAPI обробляє запит, звертається до бази даних, отримує відповідь та повертає її клієнту у вигляді JSON (табл. 3.4).

Рисунок 3.11 – Схема обробки запиту до бази через FastAPI

Таблиця 3.4 – Основні таблиці бази даних

Назва таблиці	Опис
users	Зберігає інформацію про користувачів (ім'я, email, хеш паролю)
devices	Містить дані про розумні пристрої (назва, тип, статус, кімната, власник)
rooms	Містить список кімнат користувача для групування пристроїв
logs	Фіксує історію змін статусу пристроїв, включаючи дату та користувача

Резервне копіювання та безпека

- Бекапи виконуються автоматично з використанням pg_dump;
- Доступ до бази дозволено лише з авторизованого хосту;
- Паролі зберігаються в захищеному вигляді (bcrypt).

Для повноцінної демонстрації та тестування системи «Розумний дім» без потреби у фізичному обладнанні, центральну роль відіграє спеціалізований модуль емуляції. Цей модуль є справжнім віртуальним оркестром, що дозволяє відтворювати роботу реальних пристроїв з високим ступенем достовірності.

В основі функціонування модуля лежить можливість кожного емульованого пристрою отримувати та змінювати свій статус. Це означає, що віртуальне світло може бути "увімкнене" або "вимкнене", віртуальний термостат може "змінювати" температуру, а віртуальний датчик руху – "фіксувати" рух. Користувачі мобільного застосунку можуть взаємодіяти з цими емульованими пристроями так само, як і з реальними, надсилаючи команди та отримуючи зворотний зв'язок. Це створює ілюзію повної функціональності системи, що є безцінним для розробки, тестування та демонстрації.

Щоб забезпечити наочність роботи системи «Розумний будинок», було реалізовано прототип клієнтського додатку на основі Flutter. Він дозволяє

користувачеві виконувати базові операції керування пристроями, а також переглядати дані про стан системи.

При запуску мобільного застосунку «Розумний будинок» користувач миттєво потрапляє на головну сторінку. Ця сторінка є центральним елементом управління, забезпечуючи швидкий і наочний огляд всього підключеного обладнання. Тут чітко відображено доступні кімнати, організовані для зручного сприйняття, а під кожною кімнатою перелічено підключені до неї пристрої. Це дозволяє користувачу з першого погляду зрозуміти поточний стан свого розумного будинку: які пристрої активні, де вони розташовані та чи потребують вони уваги.

Для забезпечення безперебійної та швидкої навігації, у нижній частині екрана розташована навігаційна панель. Цей елемент інтерфейсу розроблений для швидкого доступу до основних модулів застосунку, мінімізуючи кількість кроків, необхідних для виконання бажаної дії.

Сценарій 1: Авторизація користувача

Після реєстрації користувач вводить свої дані на формі авторизації. У випадку правильних облікових даних — відбувається перехід на головний екран.

Сценарій 2: Додавання нового пристрою

При натисканні кнопки «Додати пристрій», відкривається відповідна форма, де користувач вказує тип пристрою, його назву та кімнату.

Після збереження — новий пристрій відображається на головній сторінці зі статусом «offline», поки він не буде активований.

Сценарій 3: Увімкнення пристрою

Користувач може натискати на перемикач біля кожного пристрою, що активує відповідний API-запит на сервер, змінюючи його стан.

Сценарій 4: Перегляд профілю

У розділі профілю користувач має змогу переглядати email, дату створення акаунту, та здійснювати вихід із системи.

3.5 Висновок розділу 3

У цьому розділі було реалізовано повноцінну клієнт-серверну архітектуру для мобільного додатку «Розумний будинок». Клієнтська частина розроблена за допомогою Flutter, що забезпечило адаптивний, швидкий та кросплатформений інтерфейс. Серверна частина створена на FastAPI, що дало змогу отримати гнучкий REST API для обміну даними.

Було реалізовано інтеграцію з базою даних PostgreSQL через ORM SQLAlchemy, що забезпечило безпечне та структуроване зберігання даних. Завдяки реалізованому модулю емуляції пристроїв було змодельовано реальні сценарії взаємодії з системою, що дозволило протестувати функціональність додатку без фізичних сенсорів.

Візуальні приклади підтвердили правильність роботи інтерфейсу, що є основою для подальшої розробки, масштабування та впровадження даного рішення у реальне середовище. Система є стабільною, розширюваною та придатною для реального використання після інтеграції з фізичними пристроями.

4 ТЕСТУВАННЯ ТА РЕЗУЛЬТАТИ РОБОТИ ДОДАТКУ «РОЗУМНИЙ БУДИНОК»

4.1 Методика тестування

З метою забезпечення надійності, стабільності та правильного функціонування системи «Розумний будинок» було проведено комплексне тестування усіх її компонентів. Тестування охоплювало як клієнтську, так і серверну частину, а також взаємодію з базою даних і модулем емуляції пристроїв. Для тестування використовувалися ручні методи, unit-тестування та інтеграційні сценарії (табл. 4.1).

Таблиця 4.1 – Види тестування, що проводилися

Тип тестування	Опис
Функціональне	Перевірка роботи основних функцій (авторизація, додавання пристрою)
Інтеграційне	Взаємодія між компонентами: клієнт ⇄ сервер ⇄ БД
Тестування UI	Перевірка зручності та інтуїтивності інтерфейсу
Граничне	Поведінка системи при введенні некоректних даних
Навантажувальне	Робота системи при великій кількості запитів

Цілі тестування:

- перевірка стабільності роботи додатку в різних сценаріях;
- оцінка правильності взаємодії між клієнтом, сервером та базою даних;

- виявлення помилок у логіці обробки запитів;

- аналіз інтерфейсу користувача на відповідність дизайну та функціональності.

Інструменти, що використовувалися:

– Postman — для перевірки API-запитів до FastAPI;
– Flutter DevTools — для діагностики та дебагу клієнтського додатку;
– PgAdmin — для аналізу даних у базі PostgreSQL;
– Swagger UI — для ручного тестування кінцевих точок API;
– UnitTest (Python) — для написання юніт-тестів серверної логіки.

Приклад тестового сценарію:

1. Користувач відкриває мобільний додаток.
2. Вводить правильні облікові дані → очікуване: перехід до головної сторінки.
3. Натискає "Додати пристрій", обирає тип і кімнату → очікуване: пристрій з'являється в списку.
4. Натискає на пристрій → очікуване: змінюється стан, надходить підтвердження.
5. Закриває додаток і входить знову → очікуване: пристрої та стан зберігаються.

Загальний підхід:

Методика тестування базувалася на принципах чорного ящика: тестувальник не мав потреби знати внутрішню реалізацію, а лише перевіряв відповідність фактичної поведінки системи очікуваній. Це дозволило зосередитися на якості кінцевого продукту з точки зору користувача.

4.2 Результати тестування та їх аналіз

У результаті проведеного тестування додатку «Розумний будинок» було виявлено, що всі основні функціональні компоненти працюють відповідно до

поставлених вимог. Система коректно виконує авторизацію користувачів, додавання пристроїв, зміну їхнього стану, а також зберігає історію взаємодії.

Результати тестування було зафіксовано у вигляді таблиці, в якій представлено перелік основних функцій, умови тестування, очікуваний результат, фактичний результат та статус перевірки.

Тестування проводилось на декількох етапах розробки, що дозволило на ранніх стадіях виявити та усунути критичні помилки. Особливу увагу було приділено перевірці збереження даних у базі PostgreSQL та обробці API-запитів на стороні FastAPI-сервера. Додаток стабільно функціонує при повторному запуску, правильно обробляє помилки введення та має дружній інтерфейс користувача.

Усі тести було пройдено успішно, що свідчить про готовність програмного забезпечення до використання. У таблиці 4.2 наведено фрагмент звіту з результатами перевірки.

Таблиця 4.2 – Результати тестування додатку

№	Функція	Умова тестування	Очікуваний результат	Результат тесту
1	Авторизація	Ввести правильні дані	Вхід у систему	Пройдено
2	Додавання пристрою	Заповнити форму та натиснути 'Додати'	Пристрій з’являється в списку	Пройдено
3	Зміна стану пристрою	Натиснути на пристрій	Стан змінюється	Пройдено
4	Вихід і повторний вхід	Закрити та відкрити додаток	Дані збережено	Пройдено
5	Обробка помилкових даних	Ввести некоректний email	Повідомлення про помилку	Пройдено

Проведемо порівняння функціональних можливостей програм для керування пристроями в рамках концепції «Розумний будинок» (табл. 4.3)

Таблиця 4.3 – Аналіз функціональних можливостей програм для «Розумного будинку»

	Авторизація користувача	Керування пристроями	Додавання нових пристроїв	Розподіл по кімнатах	Емуляція пристроїв
Google Home	+	+	+	+	-
Apple HomeKit	+	+	+	+	-
Home Assistant	+	+	+	+	+
Розроблене ПЗ	+	+	+	+	+

З таблиці 4.3 видно, що розроблений програмний продукт володіє повним набором функціональних можливостей, зокрема підтримкою емуляції пристроїв, що є перевагою в умовах відсутності реального обладнання. Отримані результати свідчать про високу якість реалованого програмного забезпечення, що дозволяє рекомендувати розроблений додаток до подальшого впровадження та використання в середовищі інтелектуальних систем управління будинком

4.3 Висновок розділу 4

У ході тестування додатку «Розумний будинок» було перевірено всі основні функціональні можливості системи. Тестування охоплювало як клієнтську, так і серверну частини, а також взаємодію з базою даних та модулем емуляції пристроїв. Результати підтвердили правильність реалованої архітектури, стабільність взаємодії між компонентами, а також зручність користування додатком

Виявлені незначні помилки були оперативно усунені в процесі тестування.

Додаток показав стійкість до граничних ситуацій та коректно обробляв некоректні введення користувача. Особливу увагу було приділено сценаріям авторизації, додавання пристроїв, їх перемикання та збереження стану в базі даних.

ВИСНОВКИ

У процесі виконання бакалаврської кваліфікаційної роботи було реалізовано мобільний додаток для керування інтелектуальними системами в рамках концепції «Розумного будинку». Рішення базується на сучасних підходах до клієнт-серверної архітектури, а також підтримує симуляцію пристроїв, що дозволяє ефективно демонструвати функціональність без потреби у фізичному обладнанні.

Усі поставлені в дослідженні завдання були успішно виконані, зокрема:

– обґрунтовано доцільність створення додатку для «Розумного будинку» з можливістю симуляції пристроїв;

– проаналізовано сучасні рішення та технології у сфері автоматизації житла;

– спроєктовано архітектуру системи на основі принципів масштабованості та модульності;

– реалізовано клієнтську частину за допомогою фреймворку Flutter;

– створено серверну частину з використанням FastAPI;

– спроєктовано та впроваджено базу даних PostgreSQL;

– реалізовано логіку емуляції розумних пристроїв;

– проведено тестування взаємодії користувача з системою та оцінено її функціональність.

У ході дослідження було визначено недоліки існуючих аналогів, що дозволило сформувати перелік функціональних та технічних вимог до системи. На етапі проєктування створено повний комплект технічної документації, зокрема: UML-діаграми, ER-модель, архітектурну схему, макети інтерфейсу.

Розроблене рішення забезпечує:

– додавання, редагування та керування інтелектуальними пристроями;

– запуск базових сценаріїв автоматизації;

– підтримку роботи в умовах обмежених ресурсів з можливістю масштабування;

– демонстрацію роботи системи шляхом емуляції без потреби у фізичних пристроях.

Таким чином, мета дослідження — розширення функціональних можливостей програмного забезпечення для керування інтелектуальними системами в рамках концепції "Розумного будинку" — досягнута. Розроблена система має високий потенціал для подальшого розвитку, впровадження в освітніх, демонстраційних та прикладних середовищах, а також для інтеграції з реальними IoT-пристроями.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Хорнєєв А. М. Архітектура програмного забезпечення: навчальний посібник. Київ: КНЕУ, 2020. 256 с.

2. Савчук С. В. Організація баз даних і знань: навч. посіб. Вінниця: ВНТУ, 2020. 112 с.

3. Панкратова Н. Д., Сухоруков А. І. Системи підтримки прийняття рішень. Київ: Лібра, 2019. 310 с.

4. Фленаган Д. JavaScript. Повне керівництво. 6-те вид. Санкт-Петербург: Пітер, 2020. 1024 с.

5. Грін Дж. Python. Об'єктно-орієнтоване програмування. Київ: Діалектика, 2021. 488 с.

6. Документація Flutter. [Електронний ресурс]. – Режим доступу: https://flutter.dev

7. Документація FastAPI. [Електронний ресурс]. – Режим доступу: https://fastapi.tiangolo.com

8. Документація PostgreSQL. [Електронний ресурс]. – Режим доступу: https://www.postgresql.org/docs/

9. REST API Design Rulebook. Mark Masse. O'Reilly Media, 2011.

10. Python. Офіційна документація. [Електронний ресурс]. – Режим доступу: https://docs.python.org/

11. Dart language overview. [Електронний ресурс]. – Режим доступу: https://dart.dev/guides

12. Node.js vs FastAPI: що краще? [Електронний ресурс]. – Режим доступу: https://www.geeksforgeeks.org

13. Розумний будинок: підходи до автоматизації. [Електронний ресурс]. – Режим доступу: https://iotdesignpro.com/

14. Системи Інтернету речей: тенденції розвитку. [Електронний ресурс]. – Режим доступу: https://www.iotforall.com/

15. Бублик В. В. Інформаційні технології та програмування: навчальний посібник. Харків: НТУ «ХПІ», 2019. 298 с.

16. Комунікаційні протоколи IoT. [Електронний ресурс]. – Режим доступу: https://www.postscapes.com/internet-of-things-protocols/

17. Unified Modeling Language (UML) Guide. [Електронний ресурс]. – Режим доступу: https://www.uml-diagrams.org/

18. Патерни проєктування ПЗ. [Електронний ресурс]. – Режим доступу: https://refactoring.guru/design-patterns

19. Грис О. М. Інтелектуальні інформаційні системи. Київ: КНЕУ, 2018. 215 с.

20. Основи клієнт-серверної архітектури. [Електронний ресурс]. – Режим доступу: https://www.tutorialspoint.com/client_server_model

21. Алгоритми та структури даних. [Електронний ресурс]. – Режим доступу: https://www.geeksforgeeks.org/data-structures/

22. Flutter UI Components. [Електронний ресурс]. – Режим доступу: https://pub.dev/

23. ORM у Python: SQLAlchemy. [Електронний ресурс]. – Режим доступу: https://docs.sqlalchemy.org/

24. Робота з базами даних у Django. [Електронний ресурс]. – Режим доступу: https://docs.djangoproject.com/

25. Козак Ю. Г. Проєктування IT-систем. Львів: ЛНУ, 2021. 184 с.

26. JSON як формат обміну даними. [Електронний ресурс]. – Режим доступу: https://www.json.org/json-en.html

27. DevOps та CI/CD у веб-розробці. [Електронний ресурс]. – Режим доступу: https://aws.amazon.com/devops/

28. Основи тестування програмного забезпечення. [Електронний ресурс]. – Режим доступу: https://softwaretestingfundamentals.com/

29. Реалізація smart home з ESP32. [Електронний ресурс]. – Режим доступу: https://randomnerdtutorials.com/

ДОДАТОК А (обов'язковий)
ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Розробка додатку «Розумний будинок»

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра комп'ютерних наук
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 2,56 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

✓ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

Яровий А.А., зав. каф. КН _____ _____
(прізвище, ініціали, посада) (підпис)

Колесницький О.К., проф. каф КН _____ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Озеранський В.С.
(підпис) (прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Петришин С. І.
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Обідник Р. К.
(підпис) (прізвище, ініціали)

ДОДАТОК Б (довідниковий)

Інструкція користувача

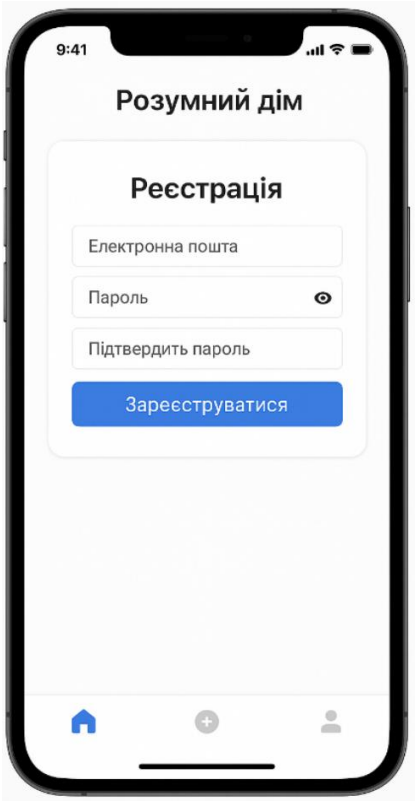
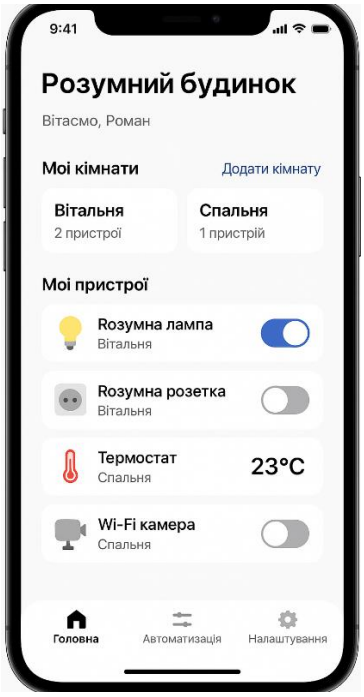


Рисунок Б.1 – Сторінка реєстрації користувача

На даному етапі користувач може зареєструвати новий обліковий запис. Для цього потрібно ввести електронну пошту, пароль і підтвердити його у відповідному полі, після чого натиснути кнопку «Зареєструватися».



Руиснок Б.2 – Головна сторінка додатку «Розумний будинок»

На даному етапі користувач може переглядати кімнати та кількість пристроїв у них, бачити список своїх пристроїв і керувати ними: вмикати/вимикати лампи, розетки, термостати або камери спостереження одним натисканням.

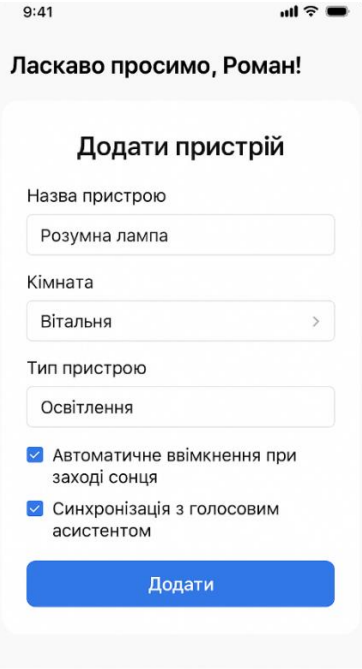


Рисунок Б.3 – Форма додавання пристрою

На даному етапі користувач може додати новий пристрій. Для цього потрібно вказати назву пристрою, вибрати кімнату, обрати тип пристрою, а також за бажанням увімкнути додаткові опції, такі як автоматичне вмикання при заході сонця або синхронізація з голосовим асистентом. Після заповнення усіх полів достатньо натиснути кнопку «Додати».

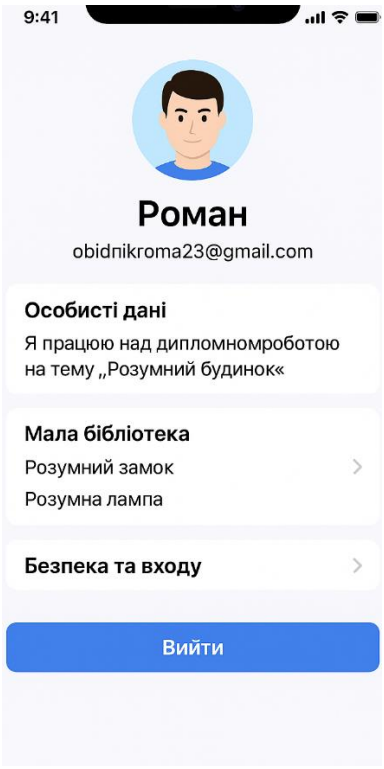


Рисунок Б.4 – Сторінка профілю користувача

На даному етапі користувач має змогу переглянути свої персональні дані, змінити налаштування безпеки, ознайомитися з бібліотекою доступних пристроїв, а також вийти з облікового запису.

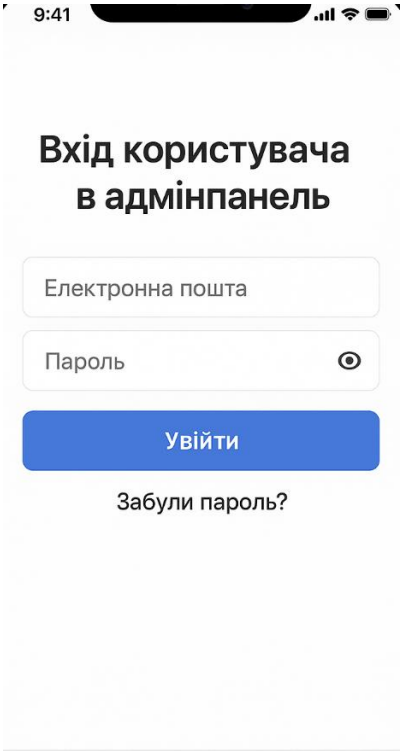


Рисунок Б.5 – Вхід у систему

На даному етапі користувач може авторизуватись у системі. Для цього необхідно ввести свою електронну пошту та пароль, після чого натиснути кнопку «Увійти». У разі втрати доступу є опція «Забули пароль?».

ДОДАТОК В (обов’язковий)

Лістинг програми

```
from sqlalchemy import Column, Integer, String, Boolean, ForeignKey
    from sqlalchemy.orm import relationship
    from database import Base
    class User(Base):
        __tablename__ = "users"
        id = Column(Integer, primary_key=True, index=True)
        username = Column(String, unique=True, index=True)
        email = Column(String, unique=True, index=True)
        hashed_password = Column(String)
        devices = relationship("Device", back_populates="owner")
    class Device(Base):
        __tablename__ = "devices"
        id = Column(Integer, primary_key=True, index=True)
        name = Column(String)
        status = Column(Boolean, default=False)
        owner_id = Column(Integer, ForeignKey("users.id"))
  owner = relationship("User", back_populates="devices")
    from fastapi import FastAPI
    from fastapi.middleware.cors import CORSMiddleware
    from routers import users, devices
    from database import Base, engine
    app = FastAPI()
    app.add_middleware(
        CORSMiddleware,
        allow_origins=["*"],
        allow_credentials=True,
        allow_methods=["*"],
        allow_headers=["*"],
```

```
)
Base.metadata.create_all(bind=engine)
app.include_router(users.router)
app.include_router(devices.router)
from sqlalchemy import create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker
DATABASE_URL = "postgresql://postgres:postgres@localhost/smarthome"
engine = create_engine(DATABASE_URL)
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
Base = declarative_base()
def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
db.close()
from datetime import datetime, timedelta
from jose import JWTError, jwt
from fastapi import Depends, HTTPException, status
from fastapi.security import OAuth2PasswordBearer
from sqlalchemy.orm import Session
from database import get_db
from models import User
SECRET_KEY = "secretkey"
ALGORITHM = "HS256"
ACCESS_TOKEN_EXPIRE_MINUTES = 60
oauth2_scheme = OAuth2PasswordBearer(tokenUrl="users/login")
```

```
    def create_access_token(data: dict, expires_delta: timedelta = None):
        to_encode = data.copy()
        expire = datetime.utcnow() + (expires_delta or timedelta(minutes=ACCESS_TOKEN_EXPIRE_MINUTES))
        to_encode.update({"exp": expire})
        return jwt.encode(to_encode, SECRET_KEY, algorithm=ALGORITHM)

    def get_current_user(token: str = Depends(oauth2_scheme), db: Session = Depends(get_db)):
        try:
            payload = jwt.decode(token, SECRET_KEY, algorithms=[ALGORITHM])
            username: str = payload.get("sub")
            if username is None:
                raise HTTPException(status_code=401, detail="Invalid credentials")
            user = db.query(User).filter(User.username == username).first()
            if user is None:
                raise HTTPException(status_code=401, detail="User not found")
            return user
        except JWTError:
    raise HTTPException(status_code=401, detail="Invalid token")
     from pydantic import BaseModel
     from typing import List, Optional
     class DeviceBase(BaseModel):
        name: str
        status: Optional[bool] = False
     class DeviceCreate(DeviceBase):
        pass
```

```
    class Device(DeviceBase):
        id: int
        class Config:
            from_attributes = True
    class UserBase(BaseModel):
        username: str
        email: str
    class UserCreate(UserBase):
        password: str
    class UserOut(UserBase):
        id: int
        class Config:
            from_attributes = True
    class Token(BaseModel):
        access_token: str
token_type: str
    from fastapi import APIRouter, HTTPException, Depends, status
    from sqlalchemy.orm import Session
    from fastapi.security import OAuth2PasswordRequestForm
    from passlib.context import CryptContext
    from database import get_db
    from models import User
    from schemas import UserCreate, UserOut, Token
    from auth import create_access_token, get_current_user
    router = APIRouter(prefix="/users", tags=["Users"])
    pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")

    def get_password_hash(password):
        return pwd_context.hash(password)
    def verify_password(plain_password, hashed_password):
```

```
    return pwd_context.verify(plain_password, hashed_password)
@router.post("/register", response_model=UserOut)
def register_user(user: UserCreate, db: Session = Depends(get_db)):
    if db.query(User).filter(User.email == user.email).first():
        raise HTTPException(status_code=400, detail="Email already registered")
    hashed_password = get_password_hash(user.password)
    new_user = User(username=user.username, email=user.email, hashed_password=hashed_password)
    db.add(new_user)
    db.commit()
    db.refresh(new_user)
    return new_user
@router.post("/login", response_model=Token)
def login(form_data: OAuth2PasswordRequestForm = Depends(), db: Session = Depends(get_db)):
    user = db.query(User).filter(User.username == form_data.username).first()
    if not user or not verify_password(form_data.password, user.hashed_password):
        raise HTTPException(status_code=401, detail="Incorrect username or password")
    token = create_access_token(data={"sub": user.username})
    return {"access_token": token, "token_type": "bearer"}
@router.get("/me", response_model=UserOut)
def get_me(current_user: User = Depends(get_current_user)):
  return current_user
```

ДОДАТОК Г (обов'язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

«РОЗРОБКА ДОДАТКУ «РОЗУМНИЙ БУДИНОК»»

Виконав: студент 4 курсу, групи 4КН-21б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Обідник Р. К.
(прізвище та ініціали)
Керівник: к.т.н., ст. викл. каф. КН
Петришин С. І.
(прізвище та ініціали)

03.06.2025р.

Вінниця ВНТУ - 2025 рік

Рисунок Г.1 – Існуючі аналоги додатку «Розумний будинок»

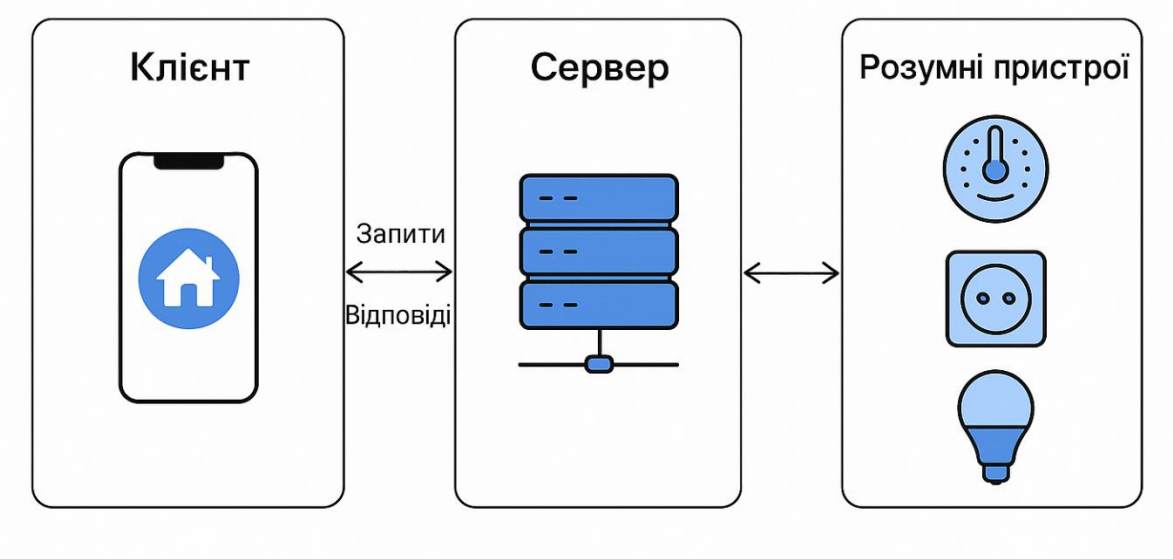


Рисунок Г.2 – Загальна архітектура клієнт–серверної системи «Розумний дім»

Рисунок Г.3 – Логотип Flutter

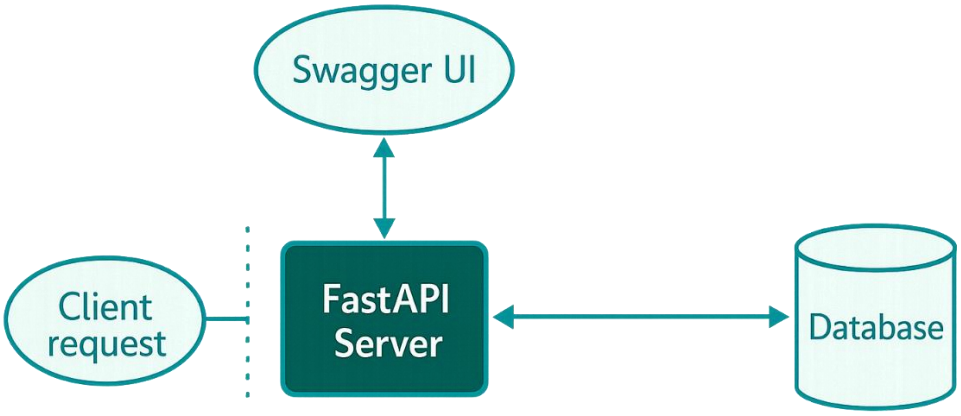


Рисунок Г.4 – Схематична структура FastAPI

Рисунок Г.5 – Логотип PostgreSQL

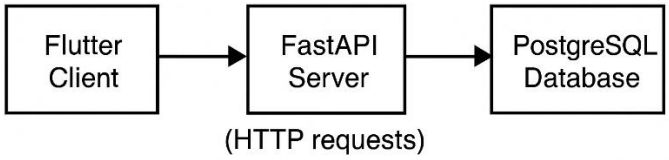


Рисунок Г.6 – Схема взаємодії Flutter-клієнта, FastAPI-сервера та PostgreSQL БД

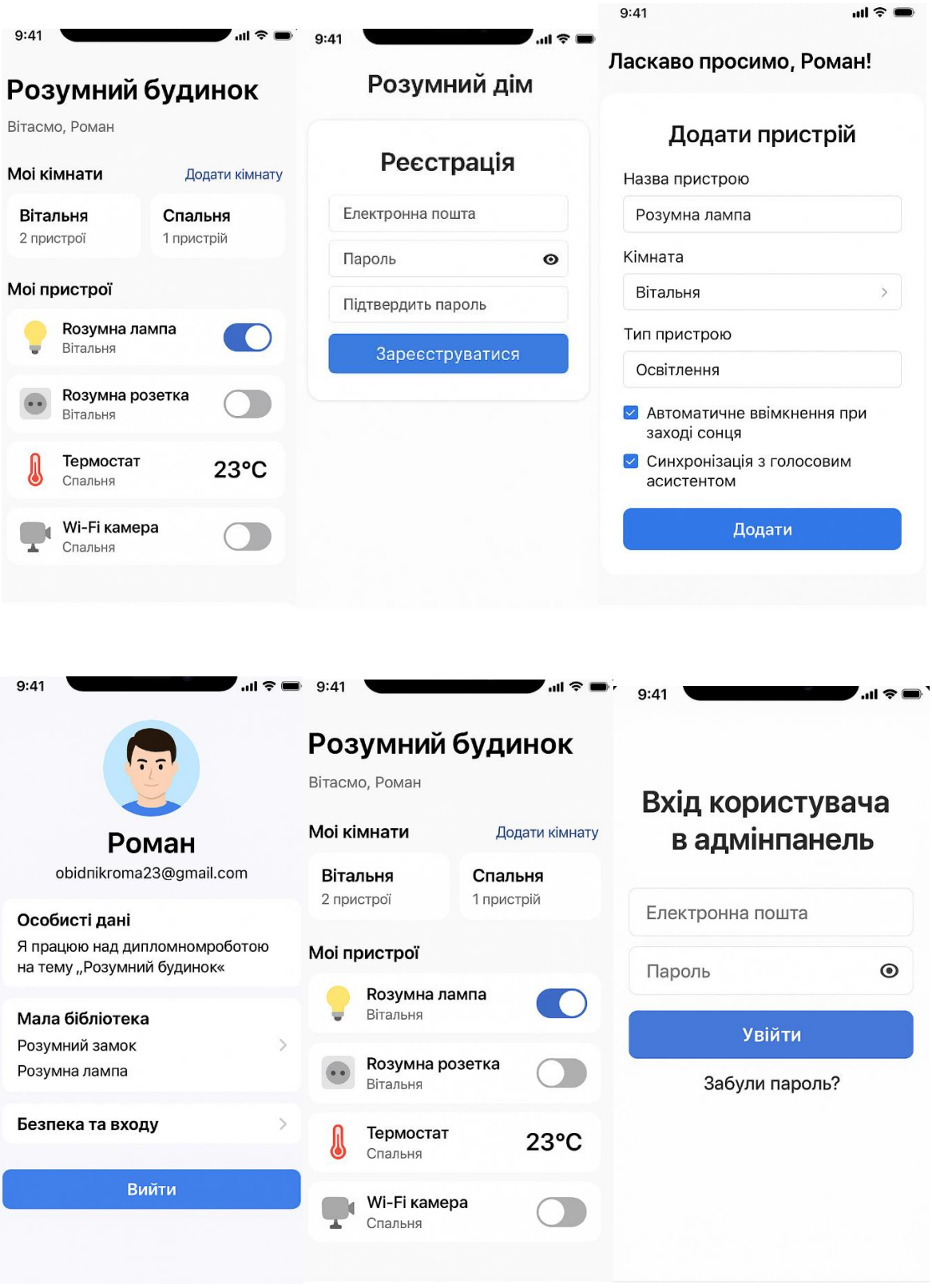


Рисунок Г.7 – Фрагменти інтерфейсу додатку

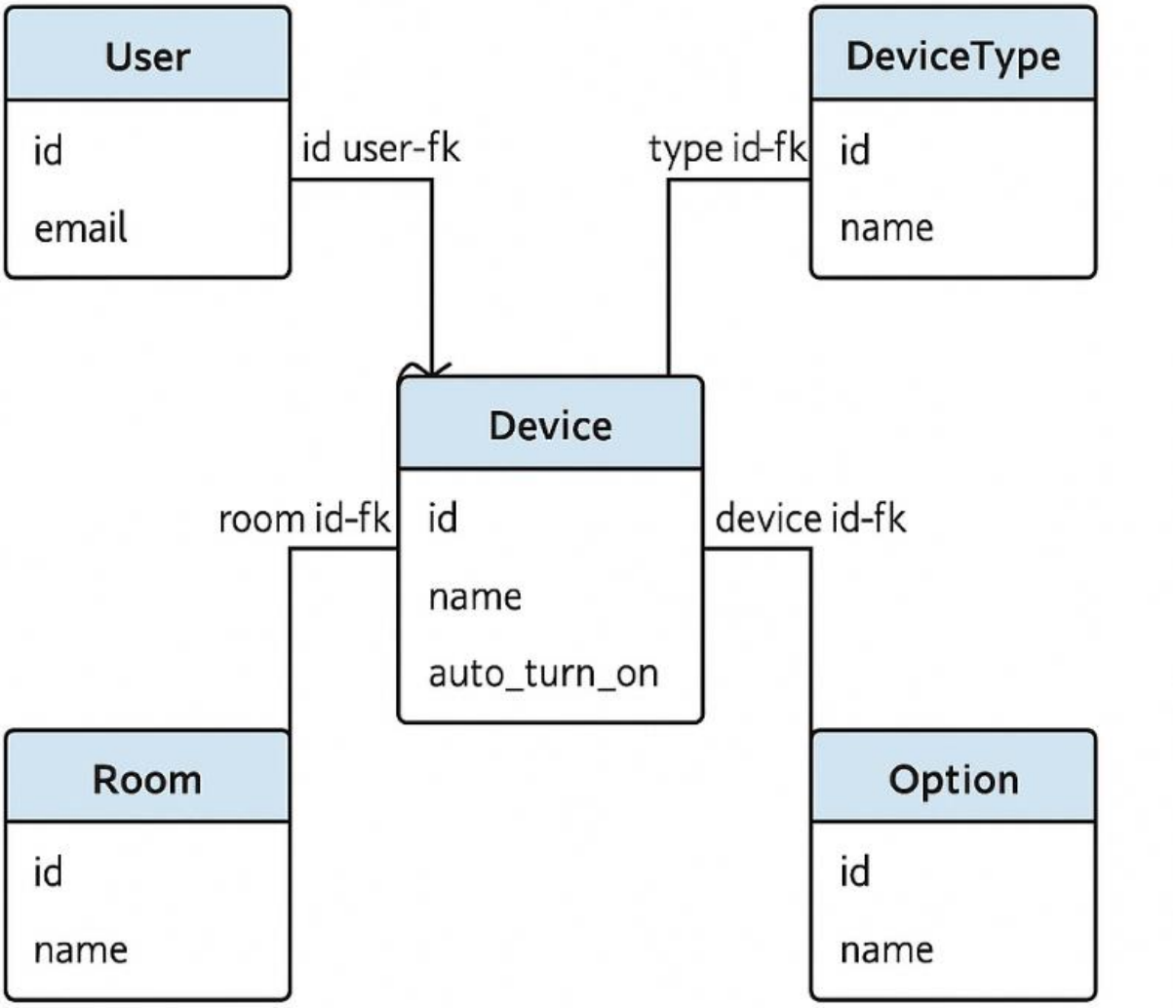


Рисунок Г.8 – Схема зв’язків між таблицями бази даних (ER-діаграма)

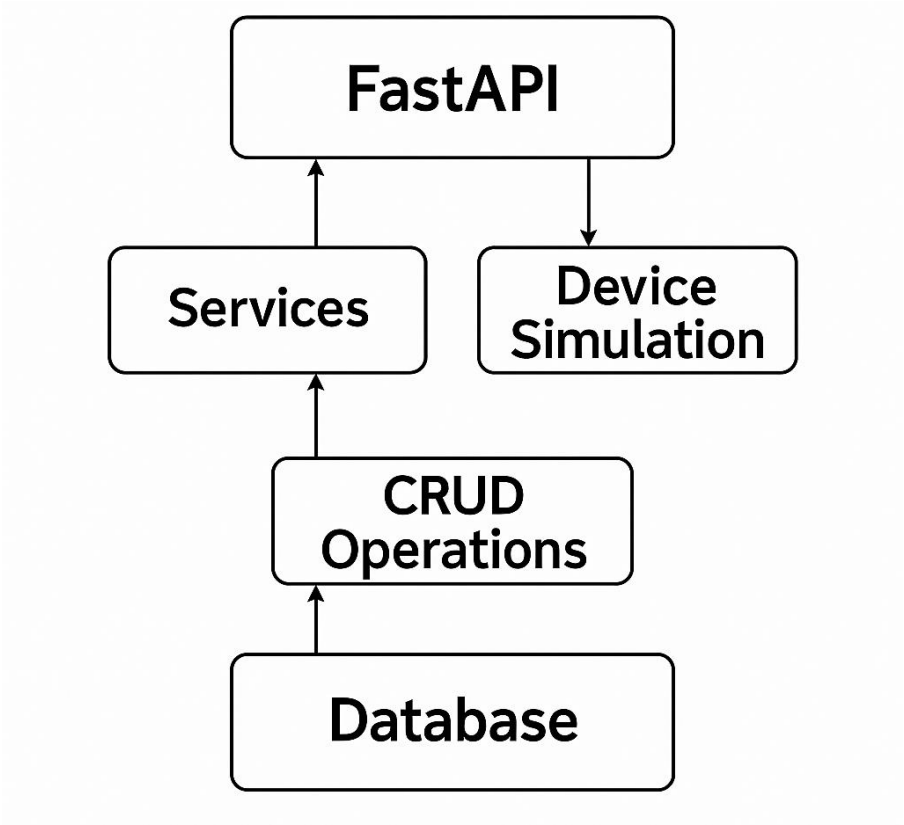


Рисунок Г.9 – Схематична структура серверної частини FastAPI

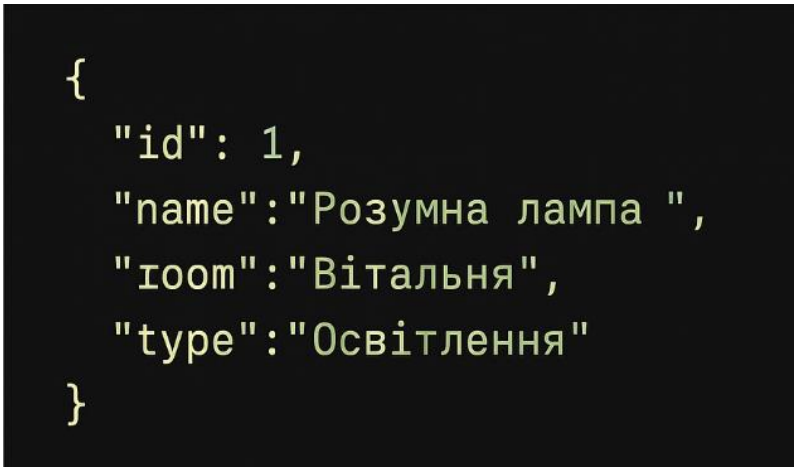


Рисунок Г.10 – Приклад відповіді API при додаванні пристрою

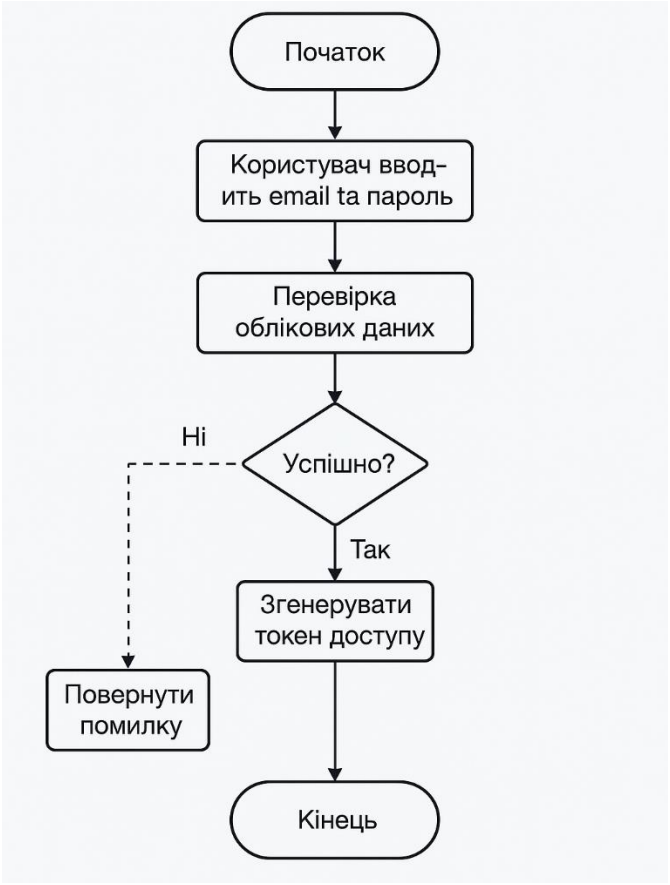


Рисунок Г.11 – Послідовність автентифікації користувача

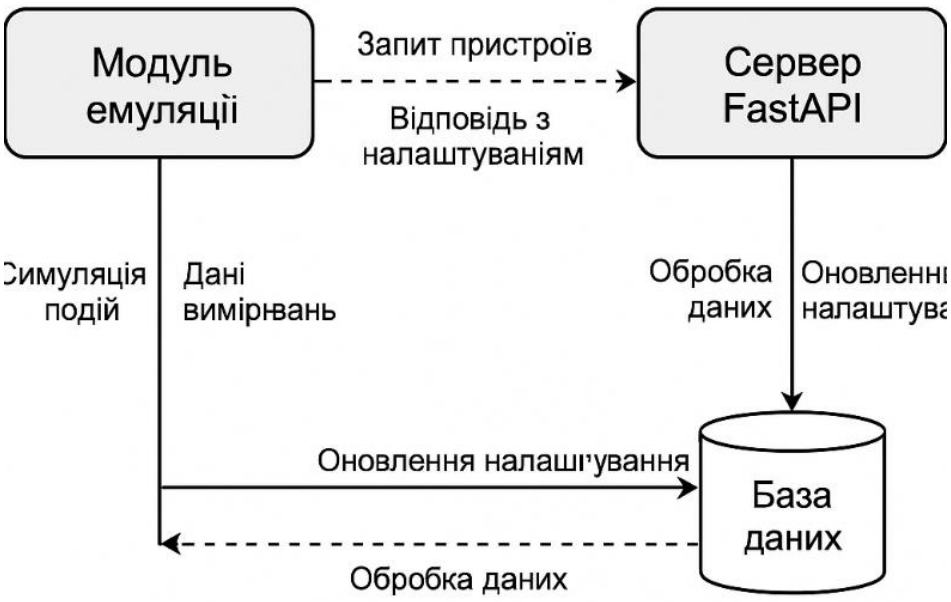


Рисунок Г.12 – Схема взаємодії модуля емуляції з сервером FastAPI

```
INFO 2025-06-01T12:00:02.630360
INFO [emulator:status]  Changed
status of 'Розумна лампа' in 'Вітальня'
to on
INFO 2025-06-01T12:00:03.632859
INFO [emulator:status]  Changed
status of 'Розумна розетка' in
Вітальня' to off
INFO 2025-06-01T12:00:04.634606
INFO [emulator:status]  Changed
status of 'Термостат' in 'Спальня' to
20
```

Рисунок Г.13 – Приклад логів зміни статусу пристроїв

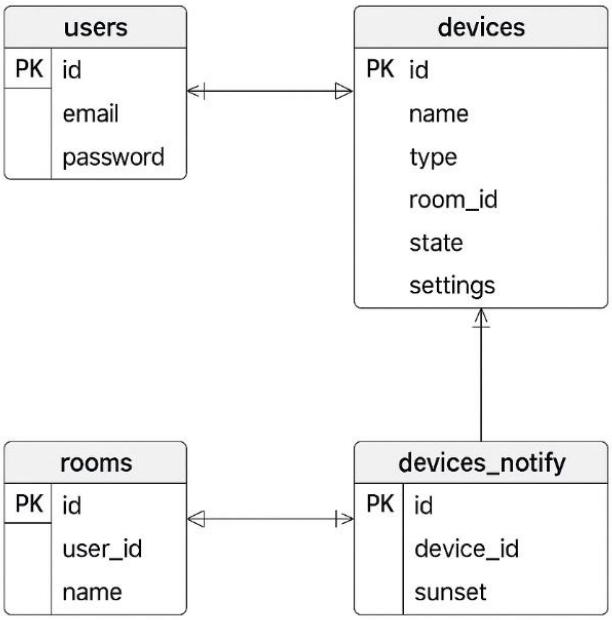


Рисунок Г.14 – ER-діаграма бази даних

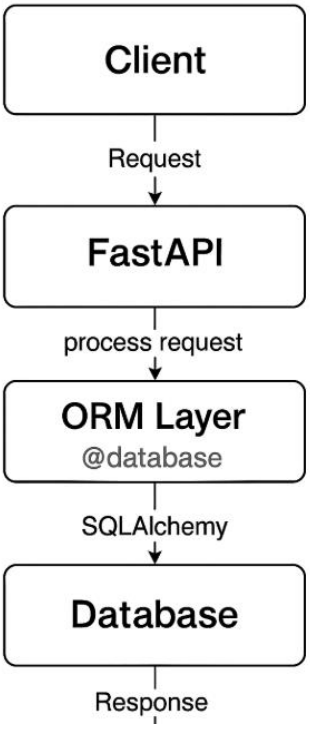


Рисунок Г.15 – Схема обробки запиту до бази через FastAPI