НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри

_____комп'ютерних наук_____

(назва кафедри)

_____ /_____Голуб Б.Л._____/

(підпис) (ПІБ)

"_02_" __червня___ 2025 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему

«Програмне забезпечення освітлення в розрізі розумного будинку»

Спеціальність 121 – «Інженерія програмного забезпечення»

Гарант освітньої програми

_____кандидат технічних наук_____ _____ ____Вайганг Г.О_____

(науковий ступінь та вчене звання) (підпис) (ПІБ)

Керівник бакалаврської кваліфікаційної роботи

_____д.т.н., професор_____ _____ __Бушма О.В_____

(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконала _____ _____Поліщук Ольга Василівна_____

(підпис) (ПІБ студента)

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри
___комп'ютерних наук_____
(назва кафедри)

_____ _____ **Голуб Б.Л.**
(науковий ступінь, вчене звання) (підпис) (ПІБ)

"16_" __грудня__ 2025 р.

З А В Д А Н Н Я

на виконання бакалаврської кваліфікаційної роботи студентці

_____Поліщук Ользі Василівні_____
(прізвище, ім'я, по батькові)

Спеціальність 121 – «Інженерія програмного забезпечення»

Тема бакалаврської кваліфікаційної роботи Програмне забезпечення освітлення в розрізі розумного будинку

затверджена наказом ректора НУБіП України від "16_" __грудня___2024р. №_2248.с__

Термін подання завершеної роботи на кафедру___25.05.2025___
(рік, місяць, число)

Вихідні дані до бакалаврської кваліфікаційної роботи – Модель розумного будинку

Перелік питань, які потрібно розробити:

Вступ і постановка проблеми, цілі та сфера застосування, огляд літератури, методологія, архітектура та дизайн системи, впровадження, оцінка та результати_____

Дата видачі завдання "16" грудня__ 2024 р.

Керівник бакалаврської кваліфікаційної роботи _____ Бушма О.В_____
(підпис) (прізвище та ініціали)

Завдання прийняла до виконання _____ _____**Поліщук О.В.**_____
(підпис) (прізвище та ініціали студента)

ЗМІСТ

ВСТУП

Актуальність.У сучасних умовах стрімкої цифровізації побутового середовища зростає попит на інтелектуальні системи керування, що забезпечують комфорт, безпеку та енергоефективність житла. Однією з ключових проблем, з якою стикаються користувачі, є неефективне управління освітленням, що призводить до зайвих витрат електроенергії та зниження рівня зручності. Традиційні методи вмикання та вимикання світла не відповідають сучасним вимогам до автоматизації та інтеграції в єдину цифрову екосистему. Саме тому розробка інтелектуальної системи освітлення є актуальним завданням, яке спрямоване на вирішення проблеми оптимального керування домашнім середовищем з урахуванням індивідуальних потреб користувачів.

Мета розробки.Метою дипломного проєкту є створення веб-орієнтованого застосунку для інтелектуального керування системою освітлення в рамках концепції "розумного дому". Запропоноване рішення дозволяє реалізувати як базові функції (вмикання/вимикання освітлення), так і розширені можливості: налаштування яскравості, колірної температури, створення сценаріїв роботи на основі розкладу тощо. Основне завдання — продемонструвати ефективну інтеграцію сучасних програмно-апаратних рішень для побудови гнучкої, масштабованої та енергоефективної системи управління освітленням.

Методи та технології.У процесі розробки було використано такі технології:

- **Мова програмування Python** — для логіки керування;
- **MQTT-брокер Mosquitto** — як медіатор обміну повідомленнями між веб-інтерфейсом та фізичними пристроями;
- **HTML/CSS та JavaScript** — для створення візуального інтерфейсу;
- **Podman** — для контейнеризації застосунку;
- **PostgreSQL** — для зберігання параметрів системи та історії керування;

Апробація.Опубліковані тези “Програмне забезпечення освітлення в розрізі розумного будинку” на міжнародній науково-практичній конференції студентів,

аспірантів “Актуальні питання розвитку науки та техніки в умовах глобалізації” (місто Боярка. 14.05.2025)

Структура записки. Дипломна записка складається з 64 сторінок, включає вступ, чотири основних розділів, висновки, список використаних джерел та додатки.

- **У першому розділі** подано загальну характеристику розумного дому та огляд рішень у сфері автоматизації освітлення.
- **У другому розділі** розглянуто вимоги до системи, обґрунтовано вибір методів та технологій.
- **У третьому розділі** описано процес розробки програмного забезпечення, архітектуру системи та реалізацію MQTT-взаємодії.
- **У четвертому розділі** наведено приклади сценаріїв використання, результати тестування та аналіз ефективності.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Сучасна епоха цифрової трансформації кардинально переформатувала парадигму організації житлового простору, впроваджуючи інноваційні технологічні рішення у повсякденне життя. Магістральним напрямком цієї еволюції виступає концепція **"інтелектуалізованого житла"** (*smart home environment*) — комплексної екосистеми взаємопов'язаних пристроїв, що функціонують у синергетичній взаємодії для оптимізації побутових процесів, підвищення комфортності проживання та раціоналізації енергетичних витрат. Серед фундаментальних підсистем такого середовища виокремлюються модулі регулювання мікроклімату, забезпечення безпеки, моніторингу енергоспоживання та, що особливо значимо, **автоматизовані комплекси керування світловим режимом приміщень**.

Автоматизований комплекс керування світловим режимом посідає пріоритетне місце в ієрархії компонентів інтелектуалізованого житла через свій безпосередній вплив на психофізіологічний комфорт мешканців, рівень захищеності об'єкта та економічну ефективність експлуатації будівлі. Інтеграція датчиків різного функціонального призначення, мікропроцесорних контролерів та централізованої системи управління створює можливість для динамічної адаптації освітлювального режиму відповідно до об'єктивних факторів навколишнього середовища, біометричних показників присутності людей, часового контексту та індивідуалізованих користувацьких преференцій. Як приклад практичної імплементації можна навести автоматичну активацію освітлювальних приладів при детектуванні руху в приміщенні або градієнтне зниження інтенсивності світлового потоку у вечірній час для формування атмосфери релаксації.

Особливої значущості набуває фактор **оптимізації енергоспоживання** — один із визначальних стимулів для імплементації подібних технологічних рішень. Автоматизація процесів освітлення мінімізує ймовірність нераціонального

використання електроенергії, коли освітлювальні прилади функціонують у незайнятих приміщеннях, а також забезпечує оптимальний баланс між природним та штучним освітленням через аналіз даних, отриманих від фотоелектричних сенсорів.

Проєктування та реалізація автоматизованого комплексу керування світловим режимом вимагає урахування комплексу технічних, функціональних та ергономічних аспектів. Зокрема, критично важливими є:

- **Багатовекторність механізмів керування** — забезпечення можливостей як для мануального втручання, так і для автономної реалізації алгоритмізованих сценаріїв освітлення;
- **Диференційована система доступу** — впровадження ієрархії прав для розмежування функціональних можливостей між адміністративним персоналом та рядовими користувачами системи;
- **Агрегація та аналітична обробка сенсорних даних** — комплексне опрацювання інформації щодо рівня освітленості, біометричних показників присутності, часових параметрів;
- **Імплементація протоколів захисту інформації** — враховуючи потенційну вразливість системи при підключенні до глобальної мережі;
- **Кросплатформна інтероперабельність** — забезпечення сумісності з іншими компонентами екосистеми розумного дому (системами відеоспостереження, механізованими віконними конструкціями, охоронними комплексами).

Фундаментальне значення має архітектура **інтерфейсу взаємодії з користувачем**. Пропонована система передбачає розробку веб-орієнтованого інтерфейсу на базі високопродуктивного фреймворку Flask (Python), що надаватиме адміністратору інструментарій для параметризації системи, реєстрації нових периферійних пристроїв, а також візуалізації журналів системних подій та

моніторингу поточного стану освітлювальної інфраструктури в режимі реального часу.

Інформаційний обмін між компонентами системи реалізовуватиметься через **протокол MQTT (Message Queuing Telemetry Transport)** — оптимізований, асинхронний комунікаційний протокол, що демонструє виняткову релевантність для середовищ Інтернету речей завдяки своїй стабільності функціонування, високій пропускній здатності та мінімальним вимогам до обчислювальних ресурсів.

В якості інформаційного репозиторію проєктується використання **реляційної системи управління базами даних PostgreSQL**, що забезпечуватиме структуроване зберігання даних про суб'єктів системи (користувачів), об'єкти керування (освітлювальні прилади), сценарії автоматизації та хронологію їх застосування. Такий архітектурний підхід гарантуватиме ефективне управління інформаційними потоками, високі показники продуктивності та перспективи горизонтального масштабування інфраструктури.

Підсумовуючи, автоматизований комплекс керування світловим режимом як органічний елемент екосистеми інтелектуалізованого житла реалізує мультифункціональний спектр завдань — від підвищення енергоефективності експлуатації будівлі до оптимізації комфортності проживання та автоматизації рутинних дій мешканців, що зумовлює його виключну актуальність у контексті сучасних підходів до організації житлового простору.

1.2 Аналіз вимог до програмної системи

Загальні положення та функціональні вимоги. Проєктування інтелектуальної системи освітлення у розумному будинку вимагає точного визначення функціональних можливостей, на які орієнтуватиметься програмна реалізація. Сформульовані вимоги дозволяють визначити межі проєкту, структуру системи, порядок взаємодії її компонентів та майбутні сценарії використання.

Основні функціональні можливості системи:

- **Аутентифікація та авторизація користувачів:**
  - Передбачено реєстрацію користувачів у системі з подальшим входом до облікового запису.
  - Користувачі поділяються на три категорії: головний користувач (власник), звичайний користувач, адміністратор.
  - Для кожної категорії задається окремий рівень прав доступу.
- **Керування освітленням у приміщеннях:**
  - Увімкнення та вимкнення світильників через веб-інтерфейс.
  - Можливість регулювання яскравості (у разі підтримки пристроєм).
  - Підтримка керування декількома зонами освітлення.
- **Розклад та сценарії освітлення (планується до реалізації):**
- Система передбачає можливість створення сценаріїв керування освітленням, які враховують час доби або зовнішні події (наприклад, “ранкове світло”, “нічний режим”).
- Передбачається реалізація прив’язки сценаріїв до конкретного розкладу або до сигналів від датчиків (руху, освітленості).
- У подальших версіях системи буде забезпечено інтерфейс для створення, редагування та активації таких сценаріїв користувачем.
- **Автоматизація на основі сенсорних даних:**
  - Взаємодія з датчиками освітленості, руху та присутності.

- Автоматичне керування світлом залежно від рівня зовнішнього освітлення або наявності людей у приміщенні.

- **Моніторинг роботи системи:**
  - Відображення актуального стану освітлювальних пристроїв.
  - Інформування про події в системі (вмикання/вимикання, спрацювання автоматичних сценаріїв).
  - Можливість перегляду історії змін.
- **Ведення логів:**
  - Протоколювання дій користувачів і системних подій.
  - Зберігання інформації про помилки, збої та інші виняткові ситуації.
  - Планується створення окремого розділу веб-інтерфейсу для перегляду журналу подій (доступ для адміністратора)

- **Керування пристроями освітлення:**
  - Підключення нових пристроїв до системи.
  - Налаштування параметрів для кожного пристрою: назва, зона, яскравість, реакція на сенсори.
  - Виявлення несправностей пристроїв і повідомлення відповідального користувача.
- **Інтерфейс користувача:**
  - Простий, зрозумілий веб-інтерфейс, адаптований для користування на стаціонарних комп’ютерах.
  - Підтримка англійської мови інтерфейсу.
  - Візуальна навігація між кімнатами, зонами та пристроями.

Нефункціональні та системні вимоги

Нефункціональні вимоги визначають якісні характеристики системи, що не залежать безпосередньо від функціональності, але мають важливе значення для її

надійності, масштабованості, зручності використання та безпеки. Нижче наведено основні вимоги до системи інтелектуального освітлення у розумному будинку.

Надійність та відмовостійкість.

- Система повинна функціонувати стабільно протягом тривалого часу без необхідності постійного втручання користувача.
- У разі втрати зв’язку з окремими компонентами (наприклад, датчиком чи контролером) система повинна фіксувати помилку, зберігати стабільний стан та відновлюватися автоматично після відновлення зв’язку.
- Очікуваний рівень доступності системи — не нижче 95% часу роботи.

Масштабованість.

- Архітектура має дозволяти додавання нових пристроїв освітлення, користувачів та сенсорів без потреби в кардинальних змінах програмного коду.
- Передбачено можливість розширення бази даних та MQTT-інфраструктури без значного зниження продуктивності.
- У майбутньому планується підтримка розподіленої системи на декілька фізичних приміщень (наприклад, багатокімнатні апартаменти або офісні зони).

Продуктивність.

- Затримка між надходженням команди (через інтерфейс або MQTT) і реакцією пристрою не повинна перевищувати 1–2 секунд.
- Підтримка роботи з одночасно активними запитами від кількох користувачів (у межах локальної мережі).
- Швидкість обробки запитів до бази даних має відповідати обсягам середньої домашньої мережі (до 50 пристроїв).

Безпека та захист даних.

- Обов’язкове використання механізмів аутентифікації для входу до системи.

- Веб-інтерфейс повинен бути захищений протоколом HTTPS.
- Планується реалізація обмеження доступу до MQTT-брокера за допомогою авторизації з унікальними обліковими даними для кожного пристрою.
- Дані в базі повинні бути захищені від несанкціонованого доступу за допомогою політик користувацьких прав.
- У подальшому передбачається впровадження журналу безпеки для виявлення потенційних загроз.

Зручність та ергономіка.

- Інтерфейс має бути інтуїтивно зрозумілим, навіть для користувачів без технічної підготовки.
- Сторінки інтерфейсу повинні відображатися коректно у всіх сучасних браузерах (Chrome, Firefox, Edge).
- Передбачається використання адаптивної верстки (тільки для стаціонарних екранів, без підтримки мобільних пристроїв).

Технологічна незалежність та відкритість.

- Усі компоненти реалізуються з використанням відкритих стандартів і безкоштовних бібліотек (Flask, paho-mqtt, PostgreSQL, SQLAlchemy).
- Система розгортається локально або в приватній мережі без залежності від хмарних сервісів.

Логування та аудит (планується).

- У перспективі планується впровадження підсистеми збору логів для аналізу роботи системи.
- Зберігання журналів дій у базі даних дозволить виконувати аудит дій користувачів та діагностику збоїв.

1.3 Огляд інформаційних джерел та існуючих рішень

У сфері автоматизованого освітлення вже наявні численні технічні рішення, що дають змогу керувати освітленням у побутових або комерційних приміщеннях. Серед найвідоміших варто виокремити такі системи:

- **Philips Hue** — одна з найпоширеніших платформ для автоматизації освітлення, яка надає змогу дистанційно змінювати освітлення за допомогою мобільного застосунку або також голосових асистентів (Google Assistant, Amazon Alexa). Крім того, вона може підтримувати створення сценаріїв освітлення та інтеграцію з іншими пристроями системи «розумного дому». Водночас система має суттєві недоліки, серед яких — висока ціна, значна залежність від стабільного інтернет-з’єднання та обмежена підтримка локалізованих інтерфейсів [1].
- **Xiaomi Yeelight** — бюджетний варіант освітлювальних рішень із базовими функціями автоматизації. Хоча система дозволяє створювати сценарії й таймери, їй бракує гнучких механізмів управління ролями користувачів. Також деякі користувачі висловлюють занепокоєння щодо конфіденційності, оскільки дані проходять через хмарну інфраструктуру, розташовану у Китаї [2].
- **LIFX** — рішення, що дозволяє підключати лампи безпосередньо до Wi-Fi без використання проміжних хабів. Незважаючи на простоту налаштування, ця система не забезпечує стабільної роботи при збільшенні кількості підключених пристроїв, а також має обмежені можливості масштабування [3].
- **Bosch Smart Home Lighting** — високотехнологічна система, інтегрована у загальну екосистему Bosch. Основними її недоліками є висока вартість та потреба у використанні виключно фірмового обладнання, що обмежує гнучкість користувачів [4].
-

1.3.1 Порівняльна характеристика та переваги запропонованої системи. На відміну від вищеописаних комерційних систем, запропоноване рішення має ряд ключових переваг:

- **Відкритий код і масштабованість:** застосування відкритих протоколів зв'язку (MQTT), зокрема брокера Mosquitto, а також таких технологій як Flask і PostgreSQL, забезпечує гнучкість і можливість масштабування без прив'язки до конкретного вендора.
- **Гнучка система доступу:** проєктована система реалізує чітку рольову модель доступу (власник, адміністратор, звичайний користувач), що дозволяє налаштувати права відповідно до потреб.
- **Стійкість до втрати інтернет-зв'язку:** навіть у випадку відсутності доступу до мережі інтернет система продовжує працювати в локальній мережі, забезпечуючи обмін повідомленнями між пристроями через MQTT.
- **Економічність:** завдяки використанню безкоштовного програмного забезпечення й сумісності з доступними мікроконтролерами (ESP8266/ESP32), загальні витрати на впровадження системи значно нижчі порівняно з комерційними аналогами.

1.4 Постановка задачі

У сучасних умовах популяризації концепції «розумного дому» все більшої актуальності набувають системи, здатні автоматично регулювати домашнє середовище без постійного втручання людини. Одним із таких аспектів є інтелектуальне управління освітленням, яке дозволяє не лише зекономити електроенергію, але й створити комфортні умови проживання. Проте більшість доступних на ринку рішень мають обмеження: вони або дорогі, або жорстко прив'язані до конкретних виробників і не надають користувачу достатньої гнучкості у налаштуваннях.

Мета цього проєкту — створення прототипу системи інтелектуального керування освітленням для розумного будинку, яка буде адаптивною, розширюваною і орієнтованою на користувача. Система повинна автоматично

реагувати на зміни рівня природного освітлення в кімнатах та враховувати встановлений графік або переваги мешканців.

Щоб реалізувати цю ідею, необхідно виконати наступні завдання:

- **Розробити веб-інтерфейс** для взаємодії користувача з системою (на базі Flask), який дозволить керувати пристроями, створювати графіки, налаштовувати профілі доступу тощо.
- **Забезпечити збирання та обробку даних** з датчиків освітлення в реальному часі.
- **Реалізувати алгоритми автоматизації**: на основі заданого розкладу, рівня освітленості та ролі користувача система повинна вмикати чи вимикати світло в кімнатах.
- **Створити рольову модель доступу**, яка передбачає різні рівні керування (власник, адміністратор, звичайний користувач).
- **Інтегрувати протокол MQTT** для надійного обміну повідомленнями між компонентами системи та пристроями освітлення.
- **Налаштувати систему зберігання даних** у базі PostgreSQL — для збереження інформації про пристрої, користувачів, логів тощо.
- **Забезпечити масштабованість системи**, щоб у майбутньому можна було легко додавати нові пристрої або функціонал.
- **Задокументувати архітектуру**, включаючи схеми, діаграми прецедентів і послідовності, щоб сторонній розробник міг швидко зрозуміти логіку роботи.
- **Передбачити можливість розширення системи** у майбутньому (наприклад, додати мобільний застосунок або голосове керування — на етапі планування).

1.5 Моделювання предметної області

У процесі розробки інформаційної системи важливо не лише розуміти її функціональність, а й мати чітке уявлення про структуру, поведінку та взаємодію між компонентамиПід час створення інформаційних систем необхідно не тільки розуміти їх функціональні можливості, але й мати ясне розуміння архітектури, динаміки роботи та способів взаємодії між елементами системи. **З цією метою використовується уніфікована мова моделювання UML (Unified Modeling Language) — стандартизований інструмент для візуального опису систем.** UML дозволяє створювати узгоджені моделі програмної архітектури, що покращує процеси аналізу, планування та взаємодії між членами команди розробників. UML включає різноманітні типи діаграм для відображення різних граней системи. До основних належать діаграми сценаріїв використання (use case), діаграми взаємодії (sequence), діаграми структури класів, діаграми бізнес-процесів (activity) та багато інших.Мова UML охоплює низку діаграм, які дозволяють описати різні аспекти системи. Серед них виділяють діаграми варіантів використання (прецедентів), діаграми послідовності, діаграми класів, діаграми активності та інші. У межах даного проєкту було побудовано діаграму прецедентів для представлення функціональності системи з погляду користувачів та діаграму послідовності, що ілюструє хронологію взаємодії компонентів системи.

1.5.1 Діаграма прецедентів. Діаграма прецедентів демонструє, які дії можуть виконувати користувачі в системі, яку функціональність їм надано та як саме вони взаємодіють із системою освітлення розумного будинку. Основними елементами такої діаграми є актори (користувачі системи) та прецеденти (варіанти використання), що зображають функціональні можливості.

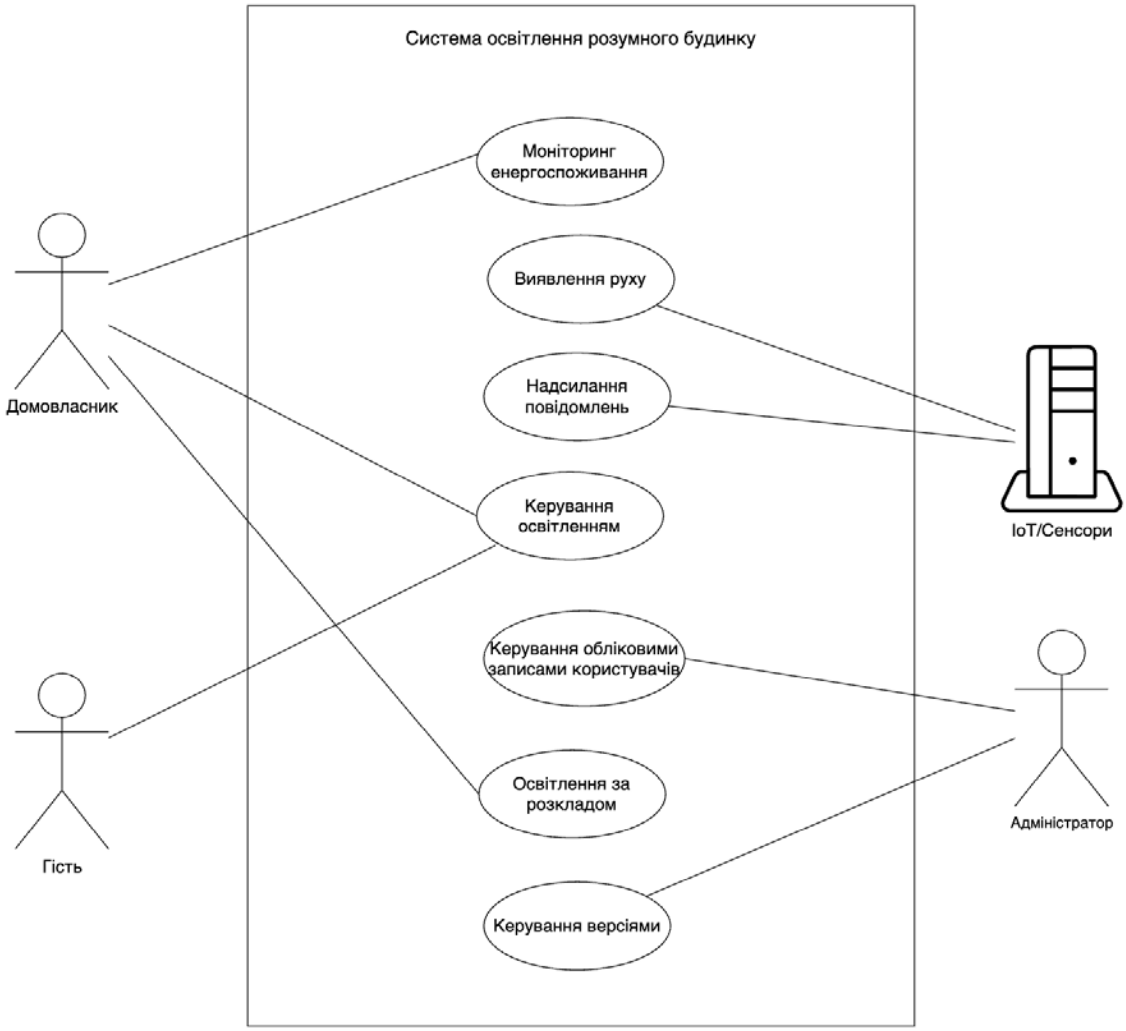

Рис.1 – Діаграма прецедентів

У представленій діаграмі (рис. 1) основними акторами є:

- **Домовласник** — головний користувач, який може контролювати освітлення, переглядати інформацію про енергоспоживання, отримувати повідомлення, керувати розкладом освітлення та обліковими записами;
- **Гість** — має обмежений доступ до функціональності, зокрема, до елементарного керування світлом;
- **Адміністратор** — відповідає за конфігурацію системи, управління версіями та контроль облікових записів;

- **IoT/Сенсори** — технічні засоби, що фіксують рух, забезпечують автоматизацію та збирання даних.

Основні варіанти використання включають:

- **Моніторинг енергоспоживання** — дозволяє отримувати звіти про витрати енергії;
- **Виявлення руху** — автоматично активує світло в залежності від присутності людей;
- **Надсилання повідомлень** — сповіщення про події або зміни в системі;
- **Керування освітленням** — ручне або автоматизоване вмикання/вимикання світла;
- **Освітлення за розкладом** — можливість встановлення часових сценаріїв;
- **Керування обліковими записами** — додавання, редагування та видалення користувачів;
- **Керування версіями** — оновлення системи й налаштування пристроїв.

Ця діаграма дозволяє швидко оцінити, які функції доступні кожному типу користувача, і чітко окреслює межі відповідальності та можливості в системі.

1.5.2 Діаграма послідовності. Діаграма послідовності (рис.2) демонструє хронологічну взаємодію між учасниками системи при виконанні певних сценаріїв. Вона дозволяє детально простежити, як система реагує на запити користувача та як компоненти обмінюються повідомленнями.

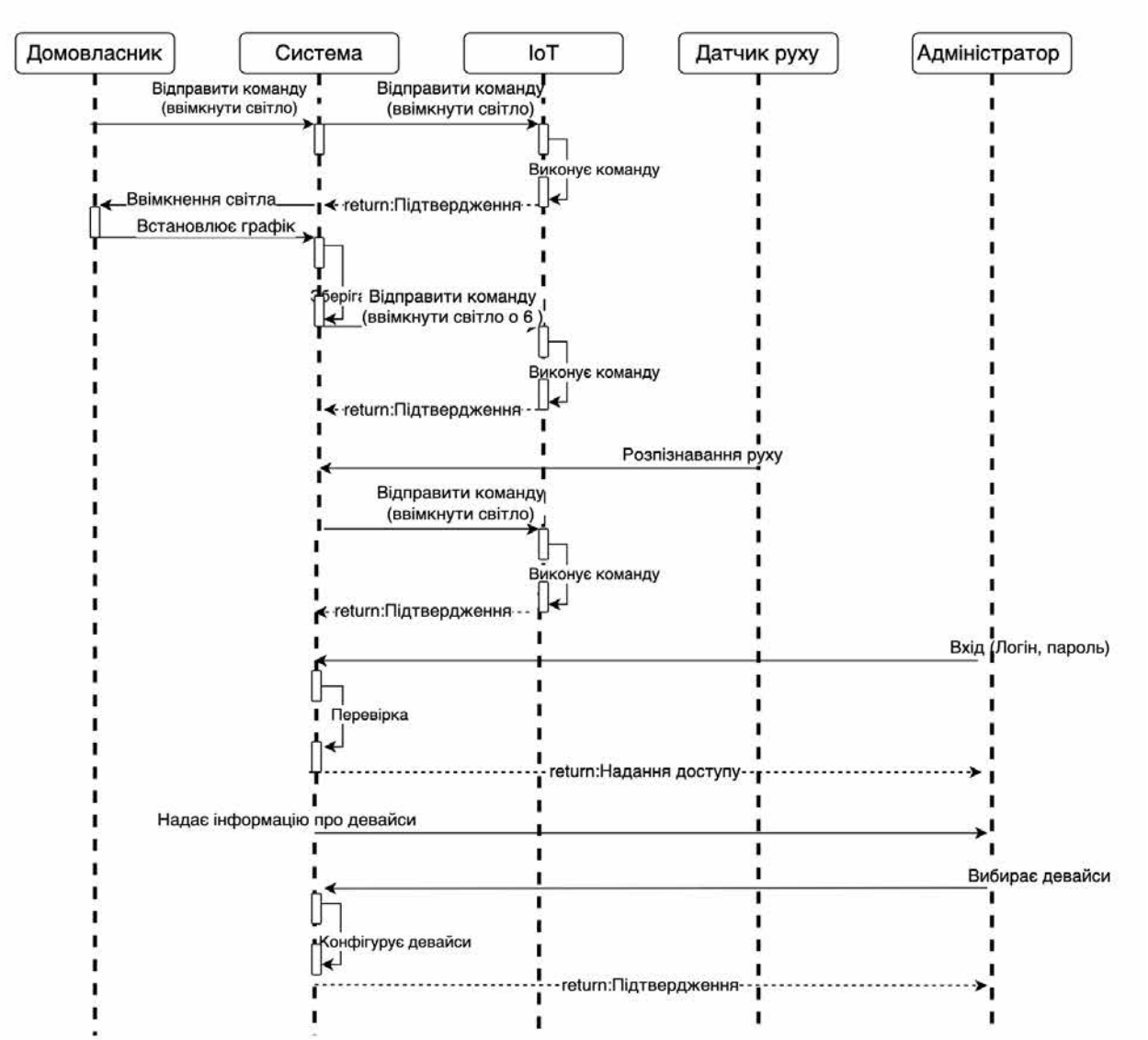

Рис.2 – Діаграма послідовності

У діаграмі моделюється типовий сценарій використання системи:

- **Домовласник** ініціює дію — надсилає команду на ввімкнення світла.
- **Система** обробляє запит і передає команду до IoT-пристрою.
- **IoT-модуль** виконує команду, вмикає світло і надсилає підтвердження назад у систему.
- **Система** може встановити розклад на автоматичне ввімкнення/вимкнення освітлення.
- При виявленні руху **датчик** ініціює відповідну подію, яка також передається до IoT-модуля, що реагує командою керування освітленням.

- **Адміністратор** виконує вхід у систему з обліковими даними, отримує доступ до панелі керування, обирає потрібний пристрій та змінює його конфігурацію.

Ця діаграма наочно відображає ланцюг повідомлень, послідовність дій, підтвердження виконання команд та умови, за яких відбувається активація функціоналу. Завдяки цьому можна чітко визначити, які компоненти беруть участь у кожному етапі процесу.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних у вигляді ER-діаграми

Логічна модель даних задає концептуальну схему бази даних і зазвичай представлена ER-діаграмою (діаграмою «сутність–зв’язок»). ER-діаграма показує основні сутності системи та зв’язки між ними.Сутності відображають об’єкти або поняття реального світу (наприклад, «Світильник», «Сенсор», «Кімната», «Користувач»), а зв’язки демонструють їхні асоціації (наприклад, “кімната містить світильники” або “користувач керує пристроєм”). Атрибути сутності описують її властивості (наприклад, характеристики світильника: яскравість, стан увімкнення; або датчика: тип, поточне значення освітленості).

- Сутності (Entities): об’єкти або концепції, що моделюються в системі (наприклад, «Світильник», «Кімната», «Користувач»).
- Зв’язки (Relationships): асоціації між сутностями (наприклад, «Кімната–Світильник» у співвідношенні один-до-багатьох), що відображають логічні залежності.
- Атрибути (Attributes): властивості сутностей (наприклад, характеристики світильника: яскравість, колір; датчика: тип, значення).

Таким чином ER-діаграма створює концептуальну модель даних високого рівня, що задає основу для подальшого проєктування бази даних. Така модель допомагає планувати схему зберігання даних і перевіряти правильність структури перед реалізацією системи. Вважається, що ER-діаграми особливо корисні для візуалізації структури та зв’язків у системах «Розумного дому» завдяки цьому розробники та аналітики отримують чітке уявлення про архітектуру даних і можуть оптимізувати рішення, що масштабуються з урахуванням потреб користувачів.

Логічна модель представлена на рис.3

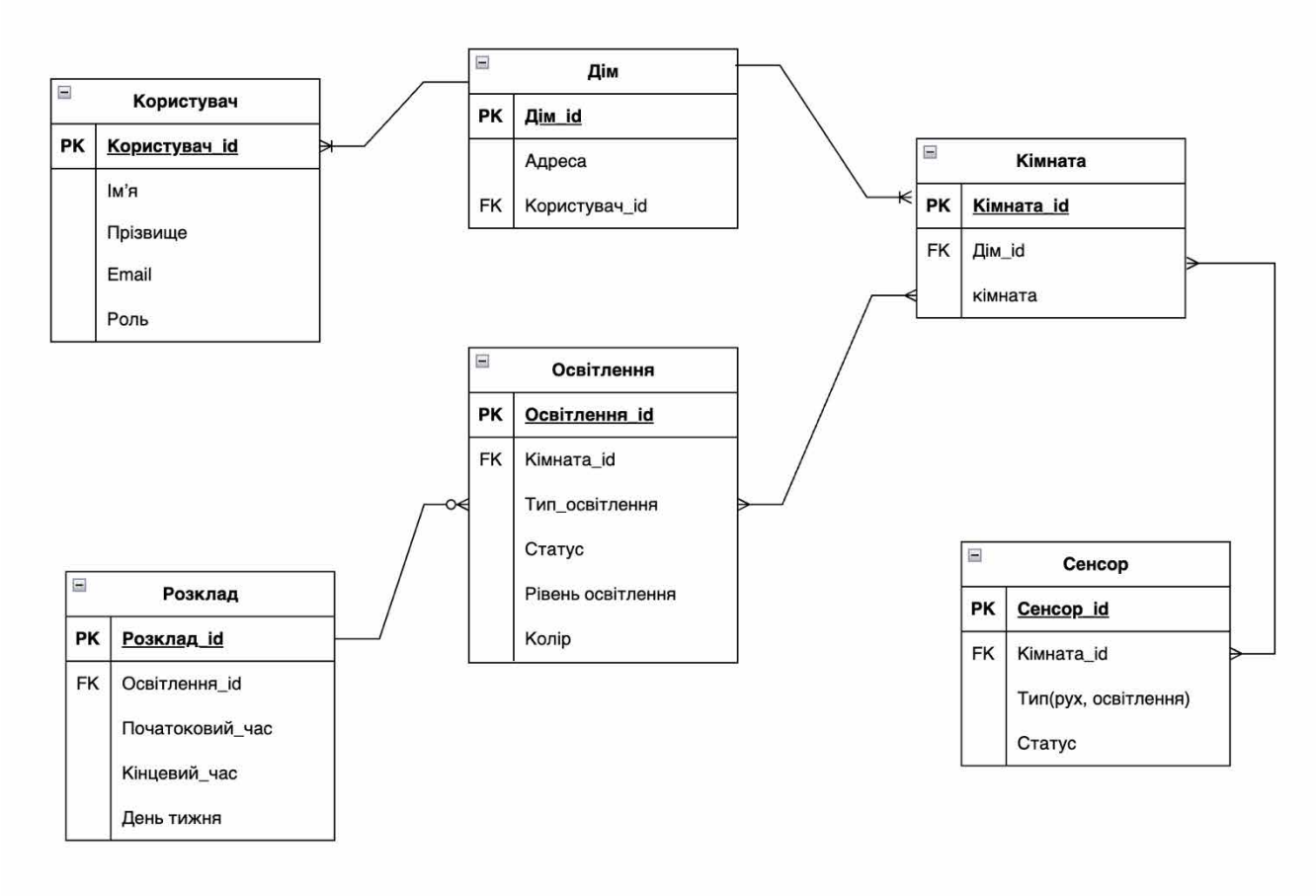

Рис.3 – ER діаграма

Ця ER-діаграма відображає структуру бази даних, яка складається з таких основних сутностей:

- **Users (Користувачі)**
  - **Атрибути:** user_id (PK), name, surname, email (унікальний), user_type
  - **Відношення:** Один користувач може володіти багатьма будинками (1:N).
- **Houses (Будинки)**
  - **Атрибути:** house_id (PK), address, user_id (FK)
  - **Відношення:** Один будинок належить одному користувачу, але може містити багато кімнат (1:N).

- **Rooms (Кімнати)**
  - **Атрибути:** room_id (PK), house_id (FK), room_name
  - **Відношення:** Одна кімната належить одному будинку, але може містити багато світильників (1:N).
  - **Відношення:** Одна кімната має один сенсор (1:1).
- **Lights (Світильники)**
  - **Атрибути:** light_id (PK), room_id (FK), light_type, status, intensity, color
  - **Відношення:** Один світильник належить одній кімнаті, але може мати багато розкладів (1:N).
- **Schedules (Розклади)**
  - **Атрибути:** schedule_id (PK), light_id (FK), start_time, end_time, day_of_week
  - **Відношення:** Один розклад належить одному світильнику (N:1).
- **Sensors (Сенсори)**
  - **Атрибути:** sensor_id (PK), room_id (FK), sensor_type, status
  - **Відношення:** Один сенсор прив'язаний до однієї кімнати (1:1).

Основні зв'язки між таблицями:

- **`Користувачі (Users)` → `Будинки (Houses)`**: Один користувач може мати кілька будинків.
- **`Будинки (Houses)` → `Кімнати (Rooms)`**: Один будинок містить кілька кімнат.
- **`Кімнати (Rooms)` → `Світильники (Lights)`**: Одна кімната може містити кілька світильників.
- **`Світильники (Lights)` → `Розклади (Schedules)`**: Один світильник може мати кілька розкладів.
- **`Кімнати (Rooms)` → `Сенсори (Sensors)`**: Кожна кімната має лише один сенсор.

Ключові особливості:

- **Каскадне видалення:** Видалення користувача, будинку, кімнати чи світильника призводить до видалення пов'язаних записів.
- **Унікальність:** Поле email у таблиці користувачів унікальне, що запобігає повторенню облікових записів.
- **Цілісність:** Забезпечується через використання зовнішніх ключів (FK) і обмежень, таких як CHECK для поля intensity у світильниках.

2.2 Діаграма класів та кооперацій

Діаграма класів— це UML-діаграма статичної структури, яка описує класи з їхніми атрибутами, методами та зв’язки між ними.

Клас у ній є шаблоном для створення об’єктів; кожен клас містить набір атрибутів (дані) і методів (функції, операції). Діаграма показує, як класи взаємопов’язані: зазвичай за допомогою асоціацій (зв’язків), узагальнення (спадкування), а також агрегації/композиції, що виражають відношення «частина-ціле». Наприклад, на діаграмі класів для системи освітлення можна бачити класи «Light» (світильник) з атрибутами (яскравість, стан) та методами (turnOn(), adjustBrightness()), і клас «Room» з атрибутом roomNumber, пов’язані асоціацією “містить”.

У UML-діаграмі класів зазвичай виділяють такі елементи:

- Клас (Class): шаблон об’єкта з назвою, списком атрибутів та методів.
- Асоціація (Association): зв’язок між двома класами, що показує їх взаємозалежність (позначається суцільною лінією, може містити роль і кратність).
- Наслідування (Generalization): ієрархічний зв’язок “підклас–надклас”, що означає, що один клас успадковує властивості іншого.
- Агрегація/Композиція: спеціальні зв’язки «частина-ціле», де композиція означає сильну залежність об’єктів (частини не існують поза цілим).

Класова діаграма забезпечує високорівневий огляд архітектури системи, полегшує документування структури програмного забезпечення і є фундаментальним інструментом об’єктно-орієнтованого проєктування.

Вона дозволяє розробникам спільно обговорювати дизайн, визначати взаємозв’язки між модулями та перевіряти повноту моделі. Наприклад, класова діаграма допомагає наочно спланувати, які класи будуть у ПЗ керування освітленням, які атрибути й методи вони матимуть і як вони між собою взаємодіють.

Діаграми кооперацій (комунікацій) демонструють поведінковий аспект: вони ілюструють, як конкретні об’єкти (екземпляри класів) спілкуються між собою для виконання певного сценарію (use case). На таких діаграмах зображаються об’єкти та передані ними повідомлення, що показує послідовність дій і ролі кожного об’єкта в реалізації сценарію. У результаті кооперaційна діаграма допомагає визначити обов’язки класів і перевірити, що у класів є необхідні методи та інтерфейси для виконання функцій системи. Наприклад, діаграма комунікацій може показувати, як об’єкт контролера освітлення надсилає повідомлення об’єкту лампи для увімкнення світла, а потім записує цю подію в лог.

Діагарма класів зображена на рис.4

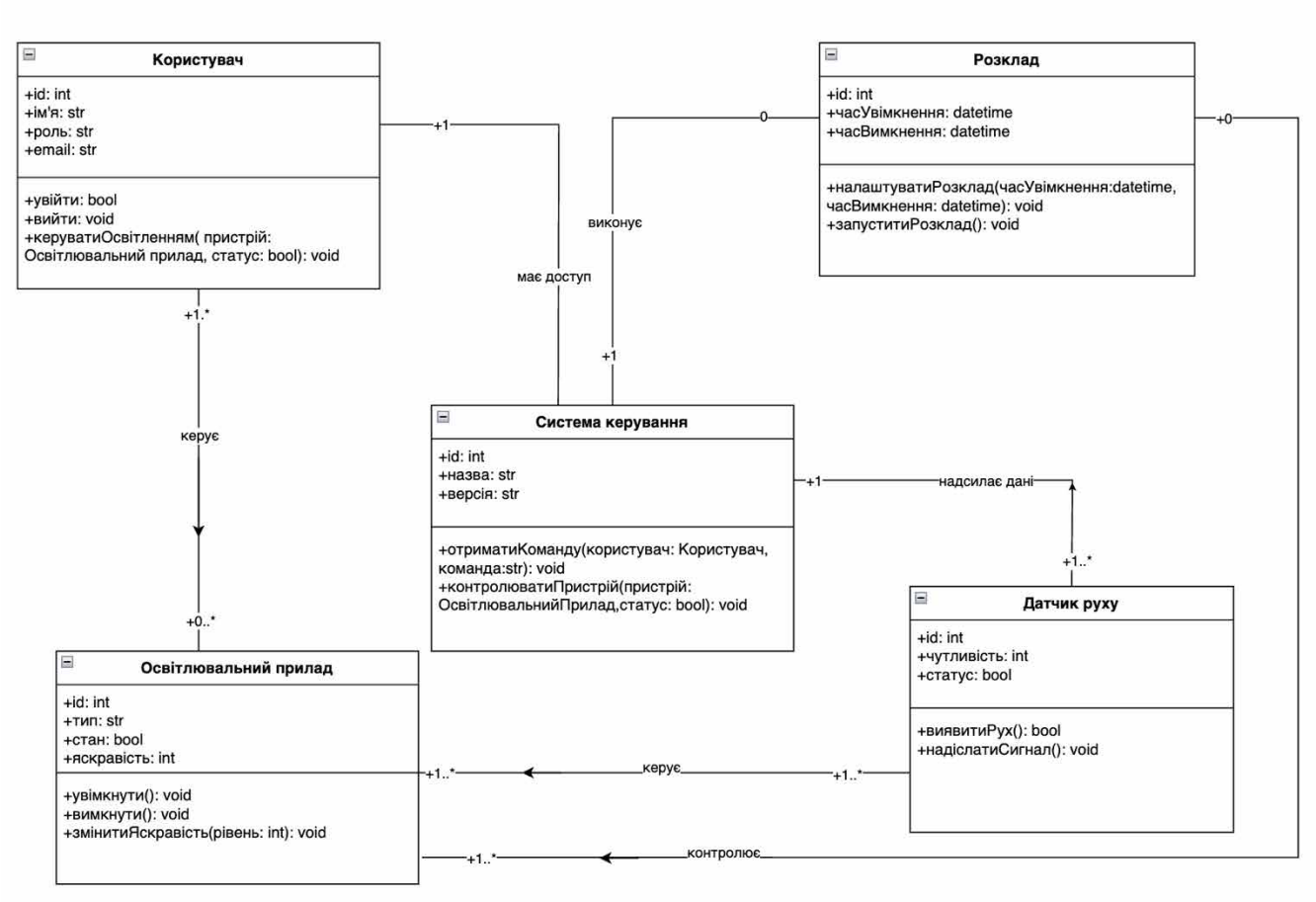

Рис.4 – Діаграма класів

Ця UML-діаграма класів відображає структуру системи розумного освітлення, зокрема взаємозв’язки між основними класами, їхні атрибути, методи, рівні видимості та кратність зв’язків. Основні класи: • Користувач – взаємодіє із системою, входить у неї, керує освітленням. • Система керування – обробляє команди користувача та керує пристроями. • Освітлювальний прилад – пристрій, який можна вмикати, вимикати та змінювати яскравість. • Датчик руху – виявляє рух та надсилає сигнал до системи керування. • Розклад – дозволяє автоматично змінювати стан освітлення за заданим часом. Діаграма містить асоціації між класами, наприклад, Користувач може керувати Освітлювальними приладами, а Система керування взаємодіє як із датчиками, так і з розкладом. Також показані рівні видимості (+ public, - private) та кратність (наприклад, один користувач може керувати кількома пристроями).

2.3 Діаграма пакетів

Діаграма пакетів (рис.5) є UML-структурною діаграмою, яка ілюструє організацію великих систем у вигляді взаємопов’язаних пакетів (модулів, підсистем). Вона показує, як класи та інші елементи групуються у пакети (своєрідні «папки»), а також залежності між цими пакетами. Завдяки цьому можна побачити загальну ієрархію системи та її логічну структуру: наприклад, виділити окремі пакети для інтерфейсу, бізнес-логіки, роботи з даними тощо.

- Структура проєкту: діаграма пакетів показує ієрархію пакетів у системі (пакети можуть вкладатися один в одного), відображаючи основні підсистеми чи шари. Це дозволяє бачити верхньорівневу архітектуру системи.
- Групування елементів: пакети використовують для об’єднання взаємопов’язаних класів та компонентів у логічні модулібщо спрощує складні діаграми класів. Наприклад, усі класи, що відповідають за керування пристроями, можуть входити до одного пакету.
- Відображення залежностей: діаграма показує залежності між пакетами (стрілками залежності); якщо один пакет залежить від іншого, зміни в першому можуть впливати на другий.Це допомагає відстежувати, які частини системи будуть затронуті у разі рефакторингу чи масштабування.

У цілому пакетні діаграми покращують організацію великої системи, розбиваючи її на модулі чи підсистеми. Це полегшує проєктування, читабельність та підтримку коду, особливо коли йдеться про складне ПЗ керування освітленням, що може містити безліч класів і компонентів.

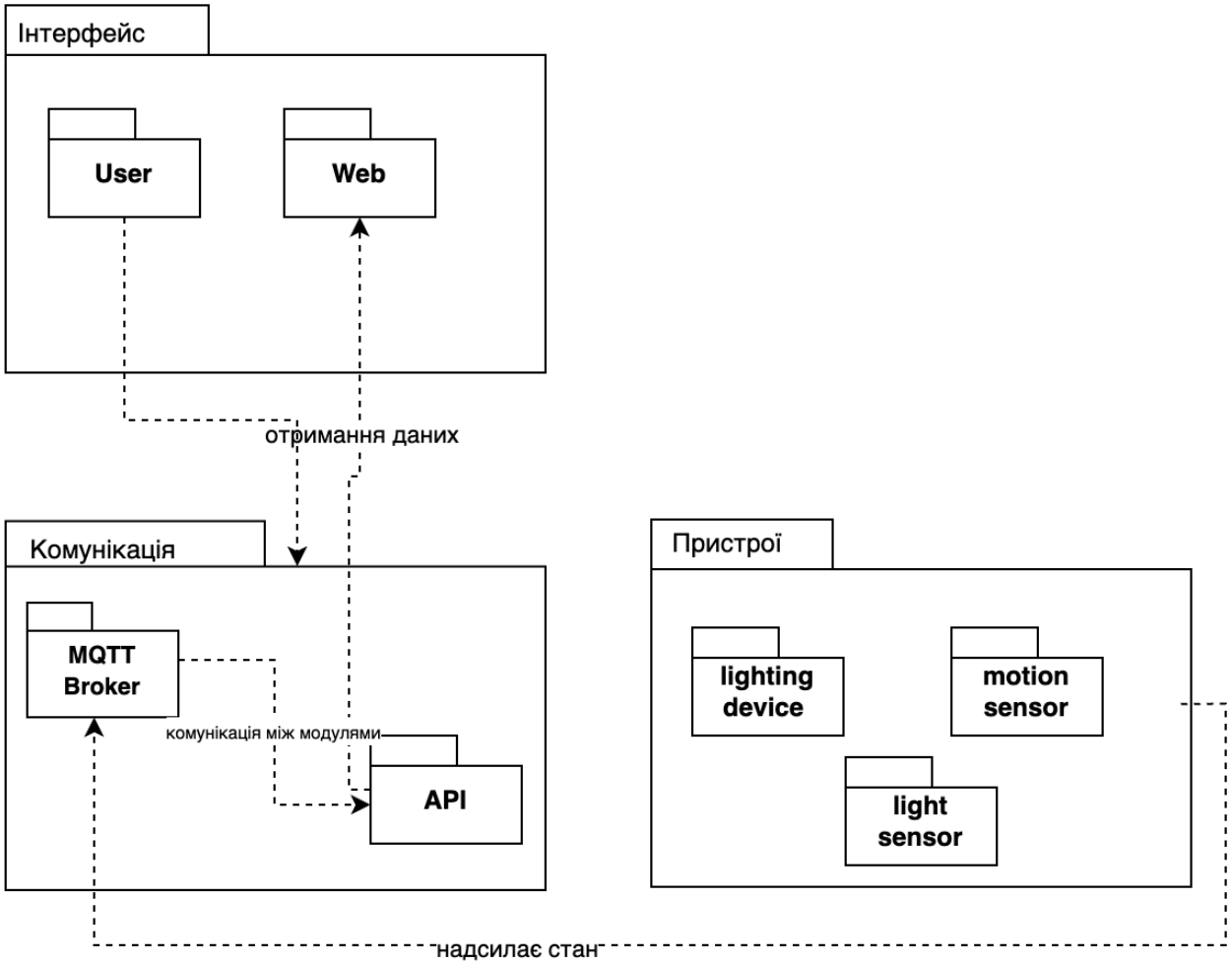

Рис.5 – Діаграма пакетів

Інтерфейс → Система керування

- Користувачі (User, Admin) та Web-інтерфейс надсилають команди до schedule.

Система керування → Пристрої

- schedule керує пристроями (lighting device, motion sensor, light sensor).

Пристрої → Система керування

- Пристрої надсилають дані про свій стан назад до системи керування.

Система керування → Комунікація

- Передає дані через API та MQTT Broker.

Комунікація → Пристрої

API та MQTT Broker забезпечують зв’язок між модулями та пристроями

2.4 Діаграма компонентів

Діаграма компонентів описує фізичну структуру системи – тобто показує, з яких виконуваних модулів, служб або апаратних блоків складається система, і як вони взаємодіють між собою.У ній компоненти (компонент – це, за UML, будь-який модуль із явно визначеними інтерфейсами, наприклад бібліотека, сервіс чи апаратний пристрій) позначаються спеціальними піктограмами, а залежності або інтерфейсні зв’язки між ними – пунктирними лініями зі стрілками. Діаграма компонентів акцентує увагу на реалізації системи: вона допомагає візуалізувати, які частини коду чи обладнання взаємодіють (наприклад, які модулі викликають сервіси інших модулів) і які зовнішні інтерфейси потребуються.

У проєктах керування освітленням у розумному будинку типовими компонентами можуть бути:

- Контролер освітлення: програмно-апаратний компонент, що виконує керування лампами (регулює яскравість, відстежує сигнали сенсорів, виконує обчислювальні алгоритми). Наприклад, це може бути розподілений контролер у центральній панелі або вбудований мікроконтролер в обладнанні.
- Світильники (лампи): апаратні пристрої з вбудованою електронікою управління (наприклад, «розумні» LED-лампи або світильники), що виконують команди від контролера. У компонентній діаграмі вони представлені як окремі блоки-компоненти.
- Інтерфейс користувача: програмний компонент (мобільний чи веб-застосунок), через який користувач керує системою. На діаграмі це окремий модуль, який взаємодіє із сервісами контролера освітлення.
- Сервер (бекенд): програмна підсистема (локальний або хмарний сервер) для зберігання стану системи, налаштувань користувача та даних (база даних), а

також для обробки високорівневих логік (наприклад, сценаріїв освітлення, аутентифікація).

- Датчики і мережеві модулі: апаратні компоненти (датчики освітленості, руху та ін.) і комунікаційні модулі (наприклад, Wi-Fi/ ZigBee-чипи), що забезпечують збір даних про середовище та обмін повідомленнями між пристроями.

Кожен з цих елементів може мати індивідуальні точки взаємодії (такі як REST API, фізичні порти підключення) та залежності від інших частин системи (використання функціональності інших модулів). Діаграма компонентів відображає ці взаємозв'язки та сприяє розумінню розподілу функцій між підсистемами, а також допомагає у плануванні впровадження та об'єднання різних елементів освітлювальної системи.

Ось приклад моєї діаграми компонентів

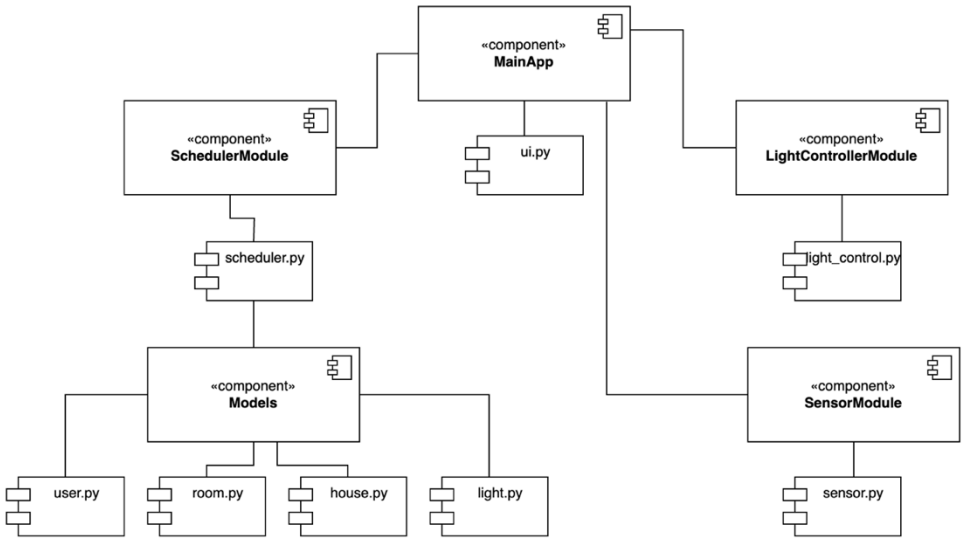

Рис.6

MainApp.

- Основний компонент, що відповідає за загальне управління системою та взаємодію з користувачем.
- Містить модуль **ui.py**, який, реалізує графічний інтерфейс.

SchedulerModule.

- Відповідає за планування та керування розкладом роботи освітлення.
- Містить модуль **scheduler.py**.

Models.

- Основний компонент для опису основних сутностей системи.
- Включає:
  - **user.py** — модель користувача;
  - **room.py** — модель кімнати;
  - **house.py** — модель будинку;
  - **light.py** — модель освітлення.
- Ці моделі використовуються як для збереження даних, так і для взаємодії з іншими компонентами.

LightControllerModule.

- Відповідає за контроль і управління світловими пристроями.
- Містить модуль **light_control.py**, який може реалізовувати функції ввімкнення, вимкнення та регулювання яскравості освітлення.

SensorModule.

- Забезпечує інтеграцію з сенсорами, які можуть використовуватись для автоматичного керування освітленням.
- Містить модуль **sensor.py**, що може обробляти вхідні дані від датчиків руху, освітленості або присутності.

Зв'язки між компонентами демонструють, як ці модулі взаємодіють один з одним для забезпечення повної функціональності системи. Наприклад, **SchedulerModule** використовує моделі з **Models** для планування, а **LightControllerModule** може отримувати дані від **SensorModule** для адаптивного управління освітленням.

3 РОЗРОБКА ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Система управління інформаційною базою

Інформаційна база даних є ключовим компонентом будь-якої сучасної інтелектуальної системи, включаючи систему освітлення для розумного будинку. Вона забезпечує збереження, організацію, обробку та доступ до структурованих даних, що є критично важливим для ефективної роботи всіх компонентів системи.

Основою для створення такої бази у моєму проєкті є реляційна система управління базами даних (СУБД) PostgreSQL. Це одна з найбільш популярних і надійних СУБД з відкритим кодом, яка підтримує потужні можливості для роботи з великими обсягами даних, забезпечує високу продуктивність, гнучкість масштабування та безпеку. Завдяки підтримці ACID-властивостей (атомарність, узгодженість, ізоляція, надійність), PostgreSQL гарантує цілісність даних навіть у випадку збоїв системи або втрати з'єднання (Крейг, Брук, 2023). Крім того, ця СУБД надає розширені можливості для створення індексів, оптимізації запитів та використання тригерів, що є особливо важливим для інтелектуальних систем з великим потоком даних.

У системі розумного будинку використання реляційної бази PostgreSQL виправдане необхідністю централізованого зберігання даних про користувачів, будинки, кімнати, сенсори, розклади роботи освітлення та інші елементи. Така організація даних спрощує їх аналіз, обробку та інтеграцію з іншими модулями системи. Основні переваги реляційної моделі в контексті розумного будинку включають:

- Структурованість даних — дані організовані у вигляді таблиць, зв'язаних між собою відношеннями, що спрощує пошук та оновлення інформації.
- Високий рівень цілісності даних завдяки підтримці первинних і зовнішніх ключів, а також обмежень типу даних (Паттерсон, Стоунбрекер, 2021).

- Підтримка транзакцій з гарантованою цілісністю даних завдяки ACID-властивостям (Гайдер, Рагураман, 2022).
- Гнучкі можливості запитів через мову SQL (Structured Query Language), що дозволяє ефективно здійснювати вибірку, фільтрацію, агрегування та сортування даних.

Структура бази даних для системи розумного освітлення розроблена з урахуванням потреб у масштабованості, безпеці та зручності доступу до даних для подальшого аналізу. Основні таблиці бази даних включають:

- Таблиця 'users' — зберігає інформацію про користувачів системи, включаючи унікальний ID, ім'я, прізвище, електронну пошту та тип користувача.
- Таблиця 'houses' — містить дані про будинки, що належать користувачам, з посиланням на таблицю 'users'.
- Таблиця 'rooms' — зберігає інформацію про кімнати в будинках, зв'язана з таблицею 'houses'.
- Таблиця 'lights' — описує освітлювальні пристрої, включаючи тип світла, інтенсивність, колір та статус, зв'язана з таблицею 'rooms'.
- Таблиця 'schedules' — зберігає розклади роботи освітлення, зв'язана з таблицею 'lights'.
- Таблиця 'sensors' — містить дані про сенсори, що встановлені в кімнатах, зв'язана з таблицею 'rooms'.

Крім традиційної реляційної моделі, система також використовує протокол MQTT (Message Queuing Telemetry Transport) для передачі даних у реальному часі між сенсорами, контролерами та центральною базою даних. MQTT є легким протоколом, оптимізованим для роботи у середовищах з низькою пропускною здатністю та високою затримкою, що робить його ідеальним для пристроїв Інтернету речей (IoT). Він підтримує асинхронний обмін повідомленнями між клієнтами через брокер (Mosquitto), що дозволяє оперативно реагувати на зміну умов у будинку (Hunkeler, Truong, Stanford-Clark, 2022).

Така інтеграція реляційної бази даних з протоколом MQTT створює надійну платформу для зберігання та обробки даних, необхідних для автоматизації освітлення та інших функцій розумного будинку.

3.2 Розробка інформаційної бази

Фізична модель була розроблена на основі логічної моделі яка була представлена на рисунку 3.

Отриману схему бази даних маємо на рисунку 7. Структура таблиць вказана на рисунках 8 - 13

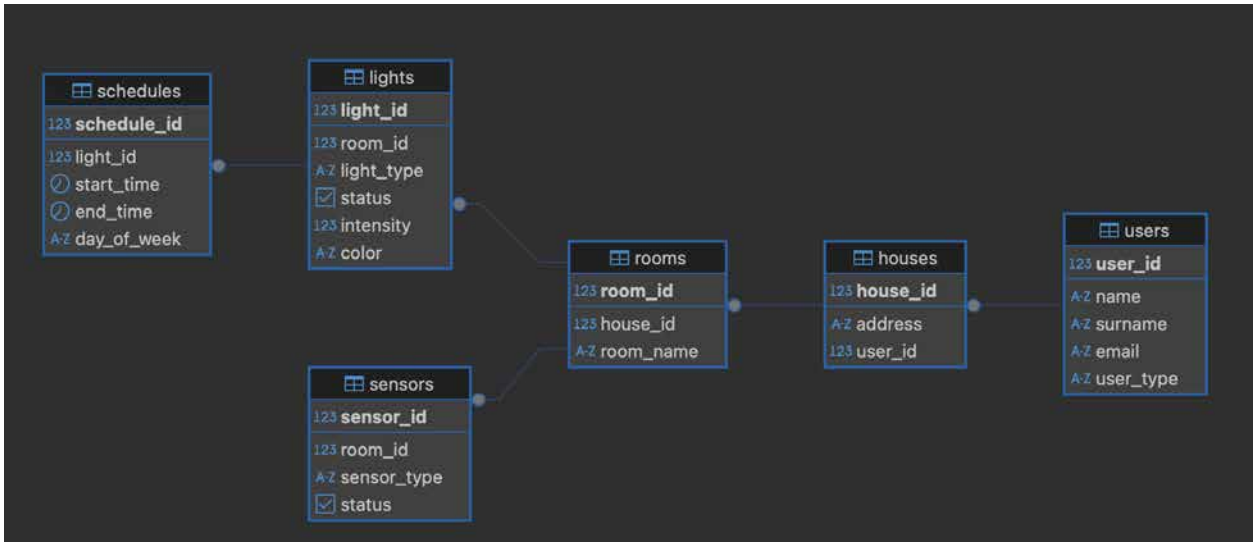

Рис.7

Створена таблиця Housres на рис.8

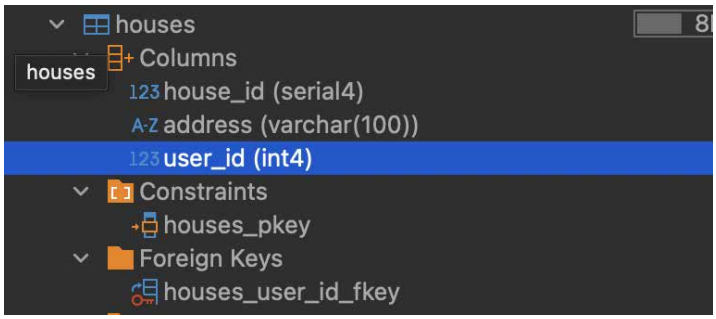

Рис.8

Створена таблиця lights на рис.9

Рис.9

Створена таблиця rooms на рис.10

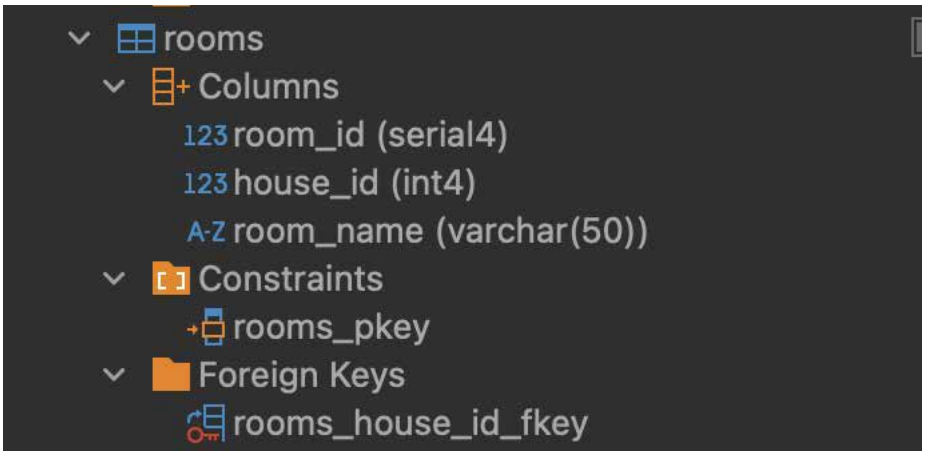

Рис.10

Створена таблиця schedulers на рис.11

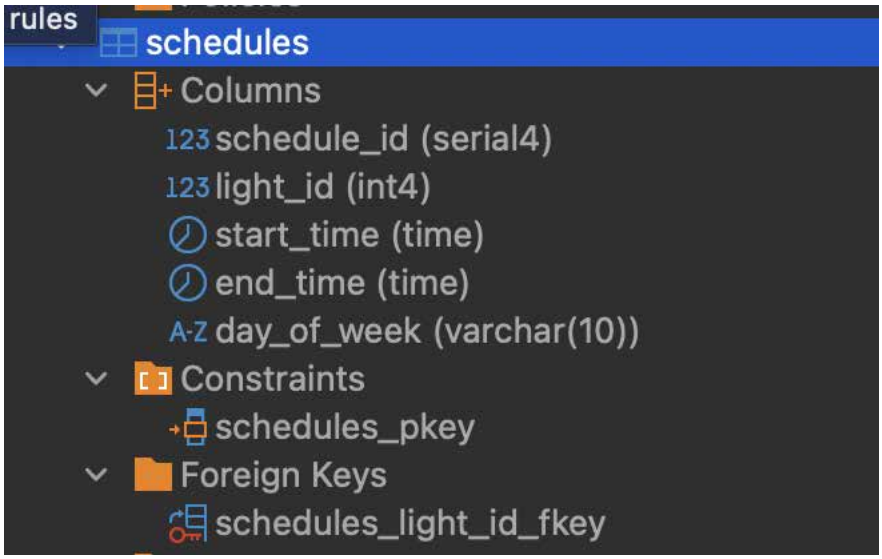

Рис.11

Створена таблиця sensors на рис.12

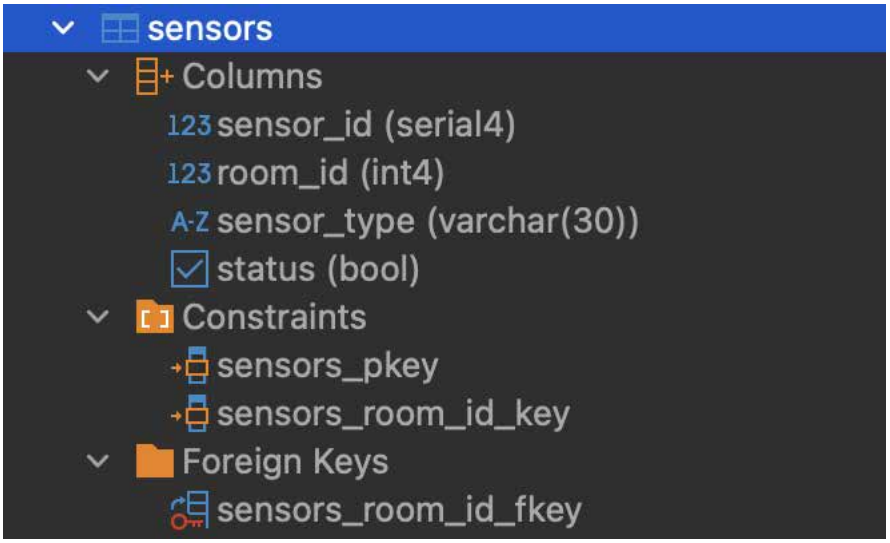

Рис.12

Створена таблиця users на рис.13

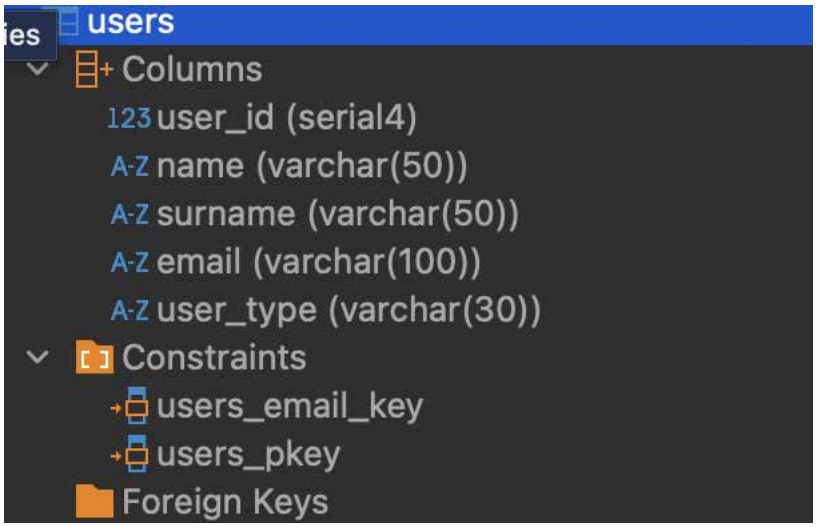

Рис.13

3.3 Вибір інструментарію для створення прикладного програмного забезпечення

Дипломний проєкт фокусується на розробці програмного забезпечення системи освітлення в розрізі розумного будинку з акцентом на сучасний стек технологій для легшої підтримки та масштабованості.

Вибраний інструментарій для створення прикладного програмного забезпечення включає:

- **Python** – використовується для розробки backend-частини завдяки своїй високій читабельності, великій кількості бібліотек для обробки даних та легкій інтеграції з іншими компонентами системи, включаючи MQTT і PostgreSQL.
- **HTML, CSS, JavaScript** – основа для frontend-частини, що забезпечує інтерактивний інтерфейс користувача, з можливістю швидкої адаптації до різних пристроїв та платформ.
- **MQTT (Mosquitto)** – це компактний протокол для обміну даними в режимі реального часу, спеціально адаптований для IoT-пристроїв з обмеженим енергоспоживанням та нестабільним з'єднанням, що робить його оптимальним вибором для сенсорних мереж у системах розумного дому.

- **PostgreSQL** – реляційна база даних для зберігання структурованих даних, забезпечує високу продуктивність, безпеку та цілісність даних завдяки підтримці ACID-властивостей.
- **GitHub** – віддалене сховище для збереження змін у коді, що полегшує спільну роботу та забезпечує доступ до історії версій.
- **Git** – система контролю версій, яка дозволяє ефективно відстежувати зміни у проєкті, створювати гілки для нових функцій та керувати конфліктами у коді.
- **GitHub Actions** – е платформа для автоматизації робочих процесів CI/CD (континуальної інтеграції та розгортання), яка забезпечує можливість налаштування автоматичного тестування, компіляції та впровадження проєкту через конфігураційні файли у форматі YAML.
- **Podman** – контейнерний рушій, що забезпечує запуск контейнерів у безпечному середовищі без необхідності використання фонових демонів. Він підтримує сумісність із Docker, спрощуючи перехід між платформами, та забезпечує високу безпеку завдяки rootless режиму, що дозволяє виконувати контейнери від імені звичайних користувачів. Podman також забезпечує розширені можливості мережевої ізоляції, управління контейнерами та збереження образів, що робить його ефективним інструментом для розробки, тестування та розгортання мікросервісних архітектур.

Цей набір технологій був обраний з урахуванням вимог до продуктивності, масштабованості, гнучкості та зручності підтримки проєкту. Така комбінація дозволяє створити ефективну, стабільну та легко масштабовану систему розумного освітлення, що може бути розширена новими функціями у майбутньому.

3.4 Архітектура програмного забезпечення

Основні підходи до архітектури ПЗ

- **Монолітна архітектура:** у цьому підході всі компоненти програми об’єднані в єдине ціле. Уся логіка — від інтерфейсу користувача до роботи з базою даних — міститься в одному кодовому блоц.

Це полегшує початкову розробку та тестування програми, адже вся система розгортається однією збіркою.Однак у міру зростання обсягу коду підтримка та масштабування такого рішення ускладнюються.

- **Клієнт-серверна архітектура:** припускає розділення системи на клієнтську частину (інтерфейс для користувача) і серверну (логіка обробки запитів і доступ до даних). Така модель робить чітке розмежування обов’язків: клієнт формує запит і відображає результати, сервер його обробляє і зберігає дані.Це дозволяє оновлювати сервер незалежно від клієнта, проте вимагає постійного зв’язку між ними.
- **Багаторівнева (N-tier) архітектура:** розділяє систему на кілька логічних шарів (рівнів), кожен зі своєю роллю. Зазвичай виділяють рівень презентації (інтерфейс користувача), рівень бізнес-логіки та рівень даних.Таке розподілення підвищує гнучкість і масштабованість системи, оскільки кожен шар можна розвивати та масштабувати незалежно.Наприклад, зміни в інтерфейсі не зачіпатимуть бізнес-логіку, а питання зберігання даних вирішуються в окремому шарі.
- **Мікросервісна архітектура:** передбачає розбиття системи на набір дрібних автономних сервісів, кожен із власною функціональністю.Кожний мікросервіс можна незалежно розгортати, масштабувати або оновлювати, що зручно для великих розподілених проектів.Недоліками є складність у налагодженні комунікації між сервісами і необхідність ретельно налаштовувати взаємодію та безпеку між ними.
- **Подійно-орієнтована архітектура (Event-driven):** базується на системі подій та повідомлень, які циркулюють між елементами системи. При виникненні певної події (як-от отримання нових показань від датчика) відповідні сервіси, що підписані на цю подію, автоматично активуються та відповідають на неї. Це забезпечує високий рівень автономності між постачальниками та отримувачами інформації: модулі комунікують через

систему повідомлень без прямої залежності один від одного. Даний підхід характеризується хорошою масштабованістю та відповідає специфіці IoT-середовища, де датчики та виконавчі механізми обмінюються асинхронними сигналами.

Обґрунтування вибору архітектури. Для проєкту системи розумного освітлення оптимальним є **багаторівневий клієнт-серверний підхід з подійною комунікацією**. Система містить веб-інтерфейс (frontend), серверну частину на Python (backend) і базу даних, що відповідає стандартній клієнт-серверній моделі.Розподіл на шари (інтерфейс – логіка – дані) підвищує гнучкість системи і спрощує її підтримку: наприклад, зміну користувацького інтерфейсу можна здійснити окремо від змін у бізнес-логіці або структурі БД. Монолітний підхід (з внутрішньою багаторівневою організацією) застосовується всередині серверної частини, оскільки проект відносно невеликий. Це спрощує розробку та розгортання (усі модулі розміщені в одній збірці). Натомість глобальної мікросервісної організації система не потребує, оскільки складність і масштаб поки що невеликі.

Водночас для зв’язку з фізичними пристроями використовується **протокол MQTT**, що надає подійно-орієнтований шар обміну повідомленнями. Сенсори та контролери світла працюють як MQTT-клієнти, публікуючи стан та отримуючи команди через брокер (Mosquitto). Як зазначають експерти, MQTT призначений для зв’язку IoT-пристроїв у ресурсомісткому середовищі і забезпечує ефективну передачу повідомлень.Він працює за публікаційно-підписувальною моделлю і «витягує» події до клієнтів у реальному часі.Такий підхід дозволяє розв’язати компонентну зв’язність і легко масштабувати систему: нові сенсори або контролери можна додавати, просто підписавши їх на відповідні теми. Таким чином, поєднання класичної клієнт-серверної моделі з MQTT-комунікацією дає баланс між простотою розробки та гнучкістю IoT-рішень.

На компонентній діаграмі системи розумного освітлення показано основні модулі проекту і їх взаємодію. Центральним є модуль **MainApp** (серверна частина), який містить бізнес-логіку і забезпечує роботу UI (фронтенд). До складу MainApp входять **Models** (класи даних: User, Room, Light тощо), **SchedulerModule** (планувальник задач, реалізований у scheduler.py), **LightControllerModule** (керування освітленням, light_control.py) та **SensorModule** (обробка даних сенсорів, sensor.py). Інтерфейс користувача реалізований через файл ui.py у складі MainApp. Діаграма відображає, що модулі взаємодіють через внутрішні виклики і обмін повідомленнями; наприклад, дані від сенсорів оброблюються SensorModule і зберігаються в базі, а планувальник SchedulerModule генерує команди увімкнення/вимкнення світла за розкладом.

- **Фронтенд (HTML/CSS/JS):** користувацький веб-інтерфейс працює в браузері, відображає інформацію про поточний стан системи (освітленість, графік роботи світла тощо) і приймає команди від користувача (наприклад, увімкнути світло). Він посилає HTTP-запити до бекенду для отримання даних або запуску дії.
- **Бекенд (Python):** серверна частина (модуль MainApp) реалізує REST-API або websockets для зв’язку з фронтендом, містить бізнес-логіку системи та керує пристроями. Тут знаходяться модулі моделей (таблиці БД), логіка планувальника розкладів, обробники MQTT-повідомлень від сенсорів, генерація команд для контролерів тощо. Бекенд підписується на теми брокера MQTT для отримання свіжих даних сенсорів і публікує повідомлення-команди контролерам світла.
- **База даних PostgreSQL:** служить сховищем інформації про конфігурацію будинку, користувачів, стан приміщень і світла, історію показників сенсорів і розкладів. Бекенд читає і записує ці дані, реалізуючи шар доступу до даних.
- **Брокер MQTT (Mosquitto):** центральний компонент для обміну подіями між пристроями. Сенсори публікують на нього дані про вимірювання

(освітленість, рух тощо), бекенд – свої повідомлення про виконані операції, а контролери світла – отримують через нього команди від бекенду. Використання брокера забезпечує асинхронну модель передачі і слабке зв’язування учасників системи.

- **Сенсори (Sensors):** фізичні або віртуальні пристрої, які збирають дані про навколишнє середовище. Вони підключаються до MQTT-брокера і періодично відправляють вимірювання (напр., рівень освітленості, рух) на конкретні теми. Бекенд на їхній основі приймає рішення (наприклад, вимкнути світло при яскравому денному освітленні).
- **Контролери світла (Light Controllers):** виконавчі пристрої, що керують живленням освітлювальних приладів. Контролери підписані на теми команд від бекенду; коли надходить повідомлення на MQTT (наприклад, “увімкнути світло в кімнаті”), вони виконують відповідну дію.
- **Система розкладів:** реалізована всередині SchedulerModule бекенду. Вона містить часові алгоритми, що періодично запускають події, пов’язані зі зміною стану освітлення (за заздалегідь визначеним розкладом). При настанні запланованого часу SchedulerModule формує команду на перемикання світла та відправляє її на виконання через центральний контролер або публікує відповідне повідомлення через MQTT.

Таким чином, компоненти системи взаємодіють за моделлю «клієнт-сервер + події»: користувацький клієнт звертається до бекенду за даними, бекенд керує апаратними модулями і зберігає результати в БД, а сенсори й виконавчі пристрої обмінюються повідомленнями через MQTT-брокер. Подібна архітектура відповідає потребам системи розумного будинку, поєднуючи зрозумілу шарову структуру та ефективну обробку подій і даних.

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

4.1 Тестування системи

Сторінка логіну системи показана на рис.14

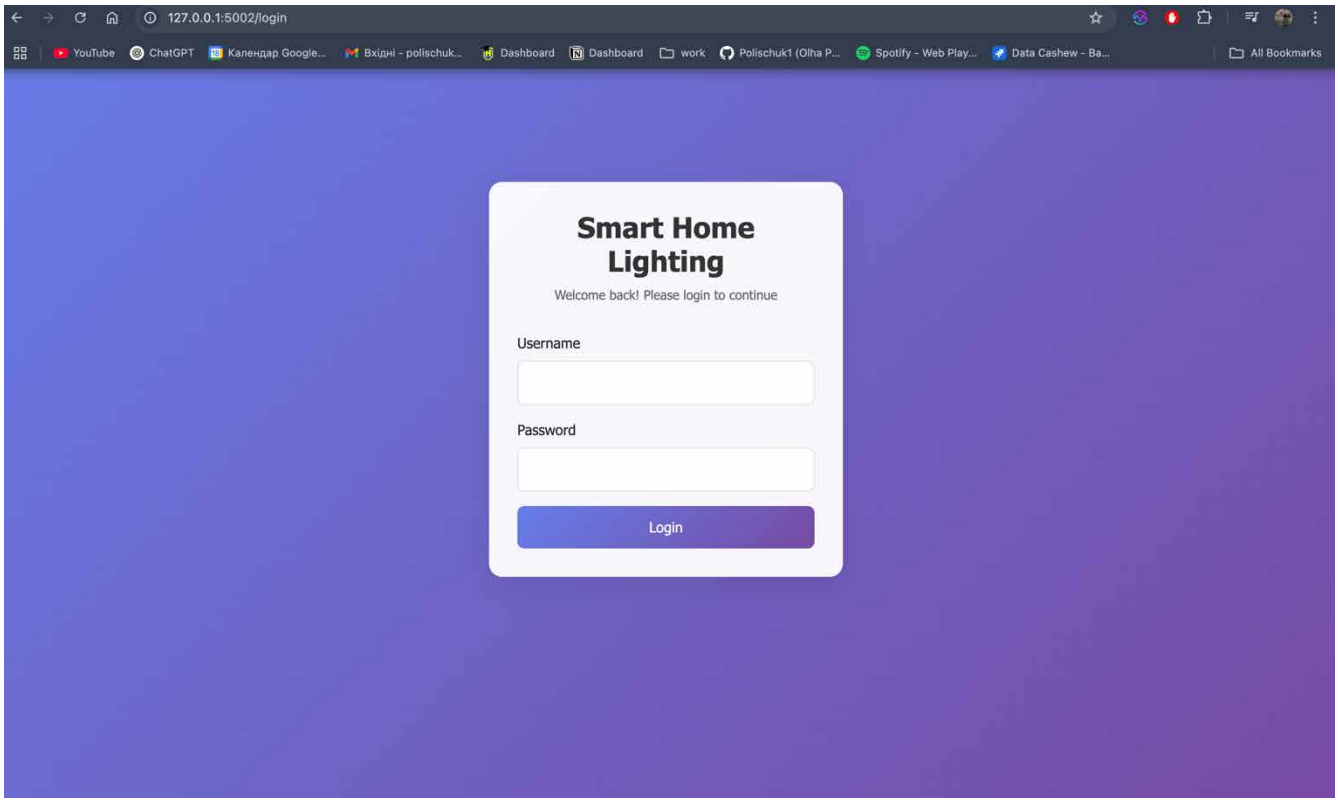

Рис.14

Спробуємо зайти під ім’ям користувача admin та з неправильним паролем та на рис.15 можемо побачити, що система нас не пустила

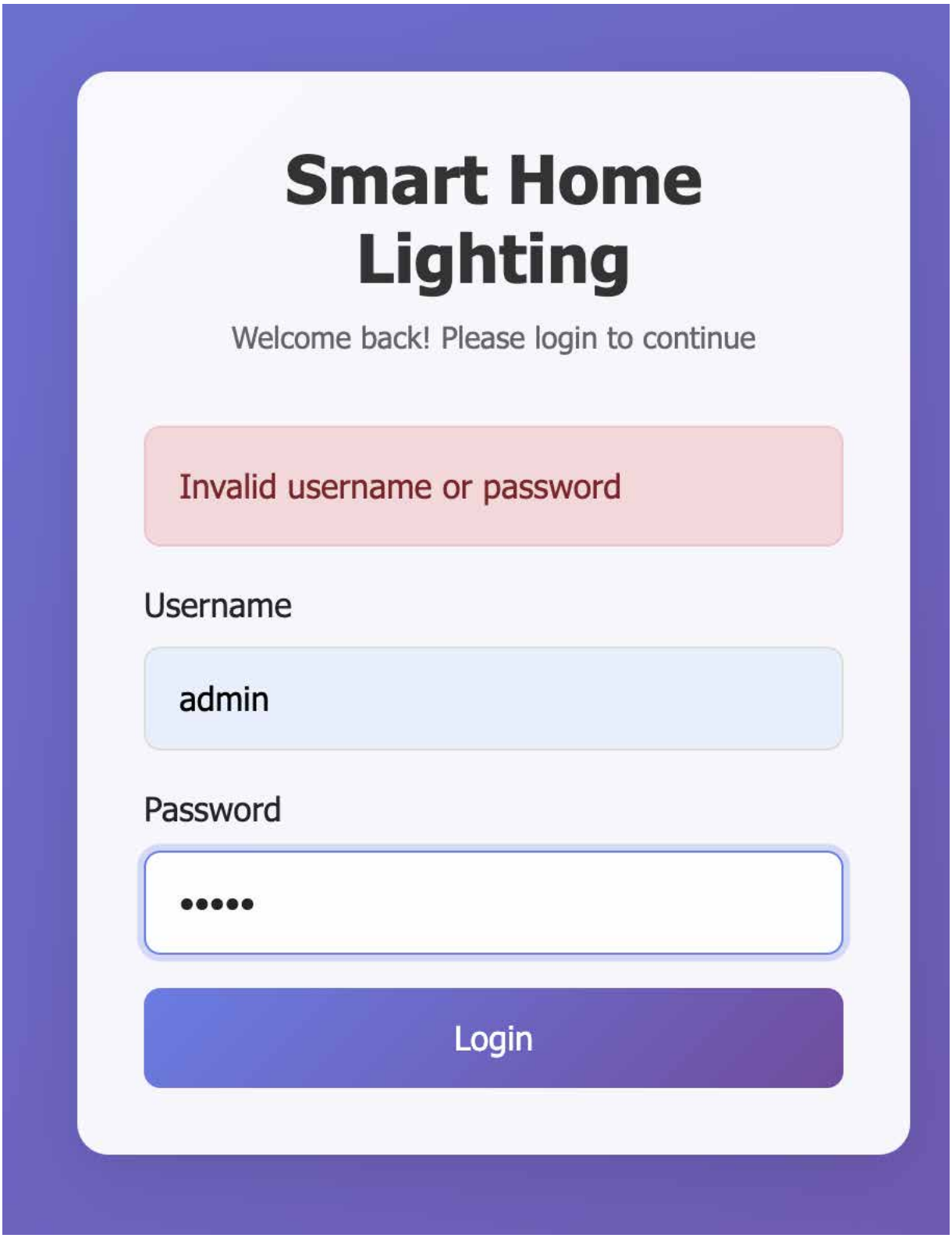
Smart Home Lighting
Welcome back! Please login to continue
Invalid username or password
Username
admin
Password
•••••
Login

Рис.15

Спробуємо знову та тепер з правильними креденшилами, на рис.16 можемо побачити, що ми отримали доступ до системи.

Адмністратор має право додавати нові девайси, сенсори та конфігурувати їх.

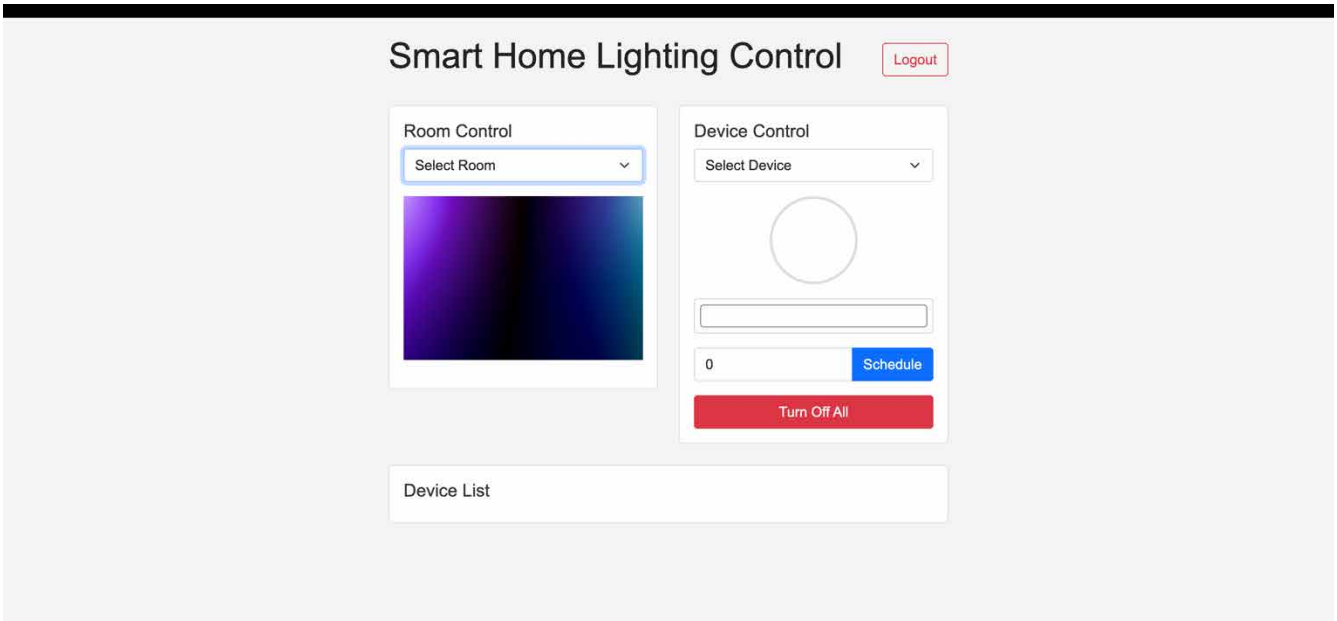

Рис.16

Також можемо зайти в середину контейнера та зробити імітацію додавання нового пристрою в систему, ми додали новий девайс на рис.17 до ванної кімнати

Рис.17

Можемо побачити відображення на головній сторінці на рис.18

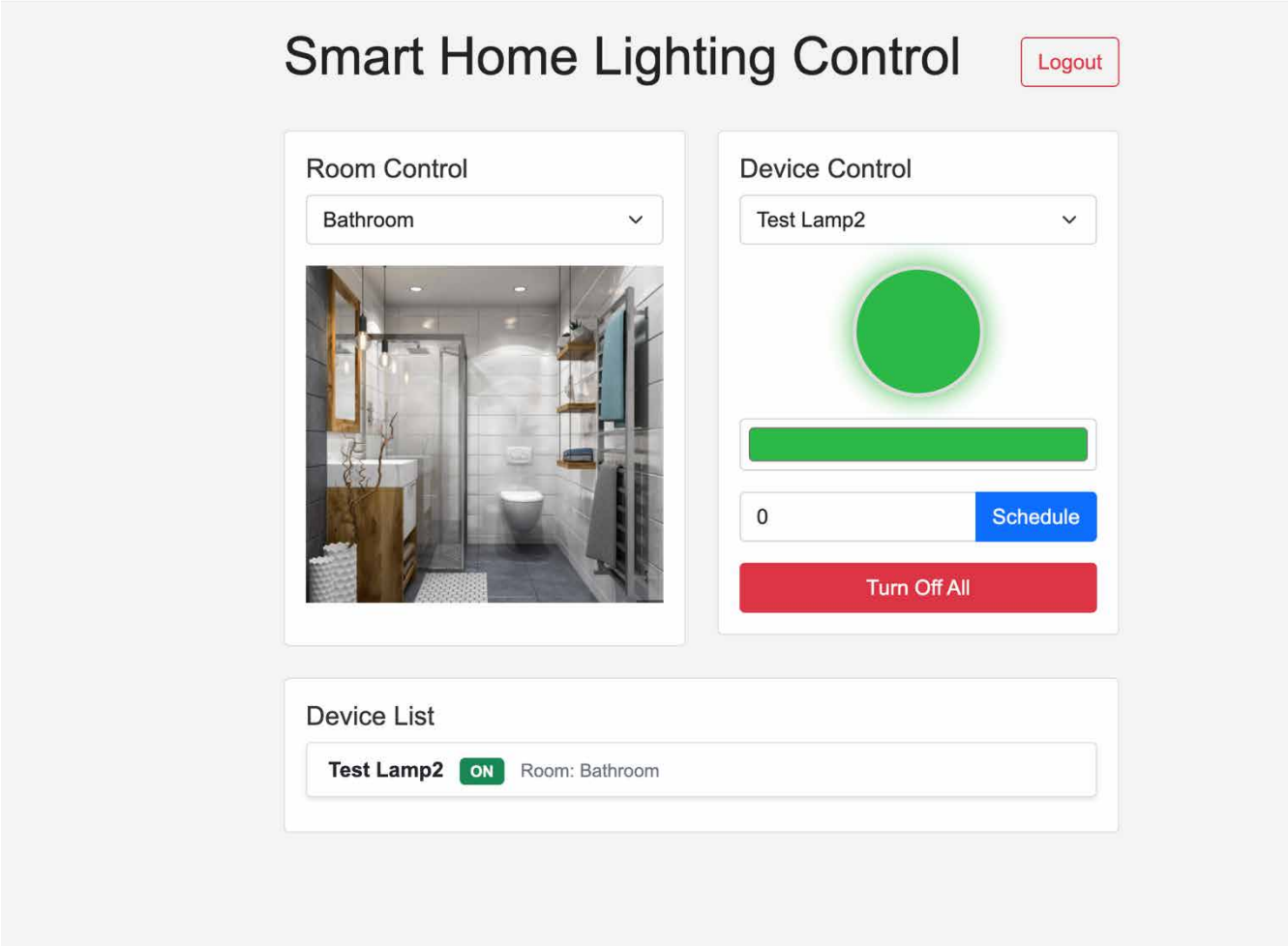

Рис.18

Також маємо функціонал який надає нам можливість запланувати графік вмикання/вимикання пристроїв як показано на рис.19

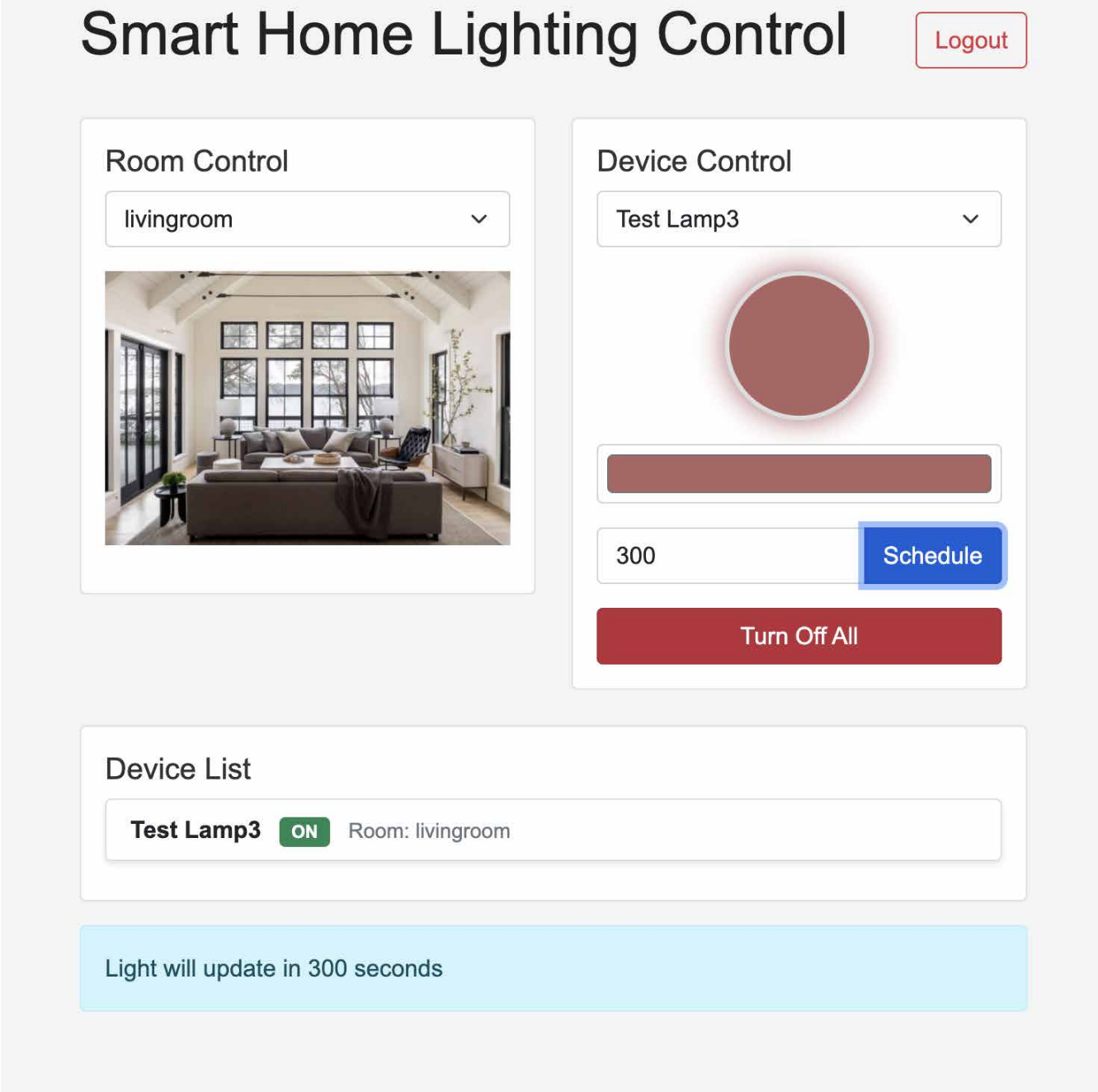

Рис.19

Також якщо лампа/девайс має функцію зміни кольору наша система зможе вибрати колір, наприклад для дитячої кімнати, зображено на рис.20

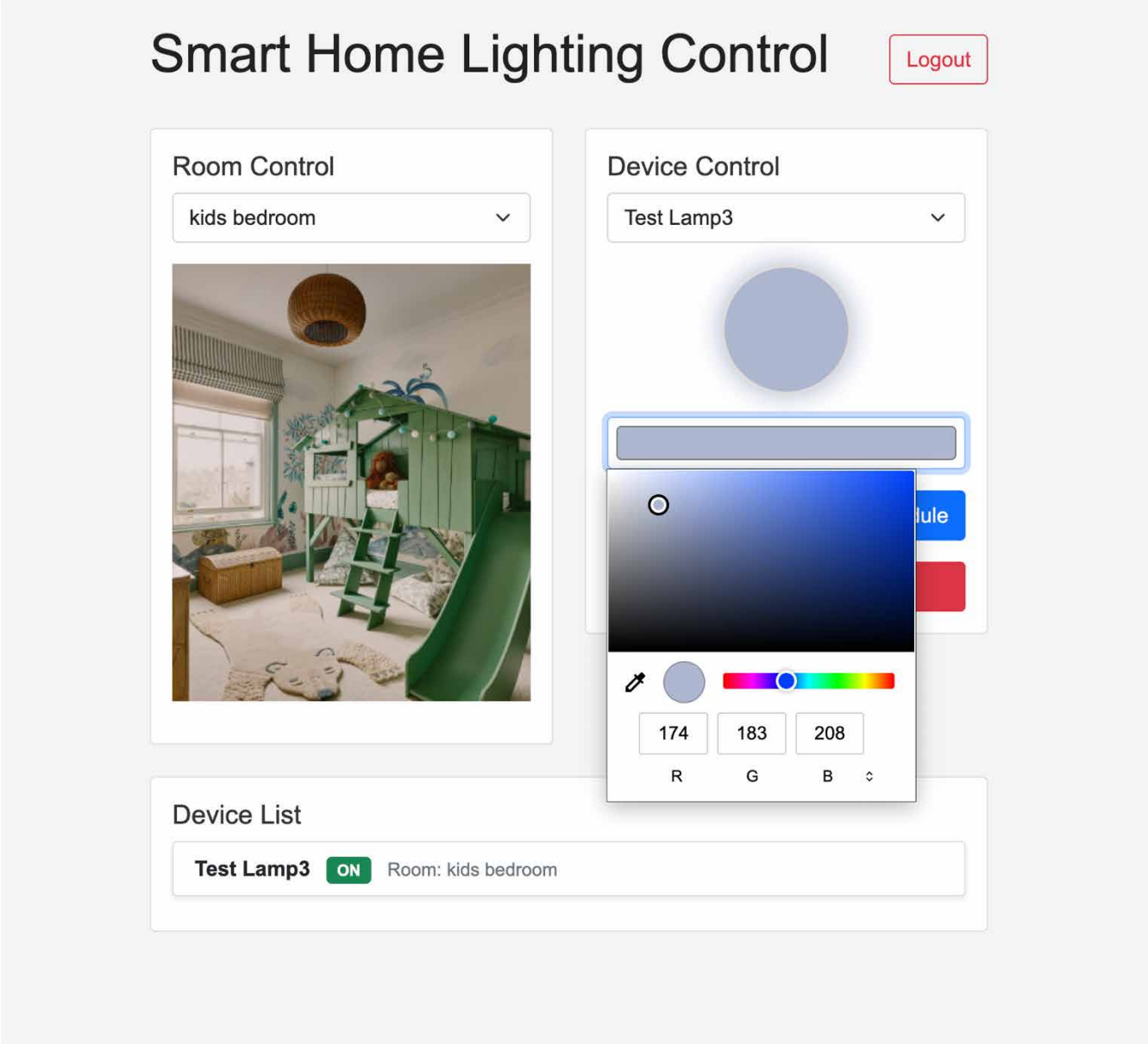

Рис.20

4.2 Вимоги до апаратного та програмного забезпечення

Проєкт програмного забезпечення для системи освітлення розумного будинку передбачає використання сучасних апаратних та програмних засобів для забезпечення високої продуктивності, стабільності роботи та легкості у масштабуванні. У цьому розділі визначені мінімальні вимоги до апаратного та

програмного забезпечення, необхідні для розробки, розгортання та експлуатації системи.

Апаратне забезпечення:

- Центральний сервер або одноплатний комп’ютер (наприклад, Raspberry Pi) для виконання функцій брокера повідомлень та збереження даних:
  1. Процесор з частотою не менше 1,2 ГГц (рекомендовано багатоядерний для кращої продуктивності);
  2. Оперативна пам’ять від 2 ГБ (рекомендовано 4 ГБ для кращої продуктивності);
  3. Мінімум 32 ГБ вільного місця на носії для операційної системи, бази даних та журналів;
  4. Підтримка Wi-Fi, Ethernet або Bluetooth для забезпечення стабільного з’єднання з мережею;
  5. Живлення від мережі 220 В або через акумулятор для мобільних варіантів.
- Мережеве обладнання:
  1. Wi-Fi маршрутизатор з підтримкою протоколу 802.11n/ac для швидкої передачі даних між пристроями;
  2. Датчики освітлення, вимикачі та інші периферійні пристрої з підтримкою протоколів MQTT, Zigbee або Z-Wave;
  3. Необхідний набір розумних ламп, світильників або контролерів з підтримкою IP-з’єднання.
- Клієнтські пристрої:
  1. Комп’ютери, ноутбуки для управління системою через веб-інтерфейс;

Програмне забезпечення:

- Операційна система сервера:

- Linux (рекомендовано Ubuntu або Raspberry Pi OS) для забезпечення стабільної роботи та доступу до широкого спектру інструментів для розробки;
- Можливість налаштування мережевих служб, таких як SSH, FTP та Firewall.

- Система управління базами даних:
  - PostgreSQL – реляційна СУБД для зберігання структурованих даних;
  - Підтримка ACID-транзакцій для забезпечення цілісності та захисту даних.
- Програмне середовище для розробки:
  - Python 3.11+ для створення основного бізнес-логіки та інтеграції з іншими компонентами;
  - Бібліотеки Paho-MQTT, SQLAlchemy, psycopg2 для роботи з брокером MQTT та базою даних;
  - HTML, CSS, JavaScript для створення інтерфейсу користувача.
- Інструменти автоматизації та контролю версій:
  - Git – система контролю версій для відстеження змін у коді;
  - GitHub для віддаленого збереження та спільної роботи над проєктом;
  - GitHub Actions для налаштування CI/CD (безперервної інтеграції та доставки).
- Контейнеризація:
  - Podman для створення, розгортання та керування контейнерами
- Брокер повідомлень:
  - Mosquitto – легкий MQTT-брокер для передачі повідомлень у реальному часі;
  - Підтримка шифрування SSL/TLS для безпечної передачі даних.
- Інші необхідні інструменти:

- MQTT брокер (Mosquitto) для передачі даних між пристроями у реальному часі.
- IDE (Visual Studio Code) для розробки програмного забезпечення.
- YAML для написання GitHub Actions для автоматизації процесів CI/CD.

Ці вимоги забезпечують ефективну роботу програмного забезпечення для системи освітлення розумного будинку, забезпечуючи при цьому високу продуктивність, безпеку та масштабованість.

ВИСНОВКИ

У ході виконання дипломного проєкту було розроблено програмне забезпечення для управління освітленням у розумному будинку. Ця система забезпечує ефективне керування освітленням, автоматизацію процесів, а також високу гнучкість і масштабованість завдяки використанню сучасних технологій, таких як Python для серверної логіки, HTML/CSS/JavaScript для фронтенду, MQTT для реального часу та PostgreSQL для надійного зберігання даних.

Системна архітектура була створена з використанням принципів об'єктно-орієнтованого аналізу та проєктування, що забезпечує модульність, гнучкість і масштабованість рішення. Це дозволяє легко додавати нові функціональні модулі, адаптувати систему під специфічні вимоги користувача та підтримувати її на всіх етапах розробки.

Для забезпечення стабільного середовища розгортання та тестування використано контейнеризацію за допомогою Podman, що значно спрощує управління компонентами системи, забезпечує портативність та ізоляцію сервісів. Інтеграція з IoT-пристроями реалізована за допомогою протоколу MQTT, що забезпечує низькі затримки передачі даних, високу надійність зв'язку та гнучкість налаштування мережевих з'єднань. Це дозволяє легко додавати нові пристрої та сенсори, створювати автоматизовані сценарії управління освітленням та реагувати на зміни в реальному часі.

Збереження структурованих даних реалізовано за допомогою PostgreSQL, що забезпечує високу продуктивність, надійність і безпеку зберігання великих обсягів інформації. Це дозволяє проводити складні аналітичні запити з мінімальними затримками, а також забезпечує цілісність та захист даних.

Крім цього, автоматизація процесів розробки та тестування налаштована за допомогою GitHub Actions, що забезпечує безперервну інтеграцію, швидке виявлення помилок та прискорює процес релізу нових версій. Особлива увага

приділена безпеці даних, включаючи шифрування з'єднань та захист переданих даних, що мінімізує ризики несанкціонованого доступу.

Розроблена система має високу гнучкість і масштабованість, що дозволяє легко адаптувати її для підтримки нових пристроїв, функцій та сценаріїв. Це робить її перспективним рішенням для майбутніх розробок у сфері домашньої автоматизації, дозволяючи створювати більш розумні, ефективні та екологічно дружні будинки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Philips Hue. Philips Hue Smart Lighting — [Електронний ресурс]. — Режим доступу: https://www.philips-hue.com/ (дата звернення: 11.04.2025).
2. Yeelight. Xiaomi Yeelight Smart Light — [Електронний ресурс]. — Режим доступу: https://www.yeelight.com/ (дата звернення: 11.04.2025).
3. LIFX. LIFX – Smarter Light — [Електронний ресурс]. — Режим доступу: https://www.lifx.com/ (дата звернення: 11.04.2025).
4. Bosch Smart Home. Bosch Smart Home Lighting — [Електронний ресурс]. — Режим доступу: https://www.bosch-smarthome.com/ (дата звернення: 11.04.2025).
5. Бочкарьов В.І., Павлов В.В. Управління вимогами в програмній інженерії. — Київ: НАУ, 2021.
6. Object Management Group. UML 2.5.1 Specification — [Електронний ресурс]. — Режим доступу: https://www.omg.org/spec/UML (дата звернення: 11.04.2025).
7. Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. — Addison-Wesley, 2003.
8. Алєксєєв В.П. Об'єктно-орієнтований аналіз і проектування. — Харків: ХНУРЕ, 2020.
9. GeeksforGeeks. How to Design ER Diagrams for Smart Home Automation Systems — [Електронний ресурс]. — Режим доступу: https://www.geeksforgeeks.org/how-to-design-er-diagrams-for-smart-home-automation-systems/ (дата звернення: 27.04.2025).
10. Vertabelo. The Smart Home Data Model — [Електронний ресурс]. — Режим доступу: https://vertabelo.com/blog/the-smart-home-data-model/ (дата звернення: 27.04.2025).

11. Visual Paradigm Online. Class Diagram for Home Automation System — [Електронний ресурс]. — Режим доступу: https://online.visual-paradigm.com/community/share/untitled-1kwgv2e595 (дата звернення: 27.04.2025).
12. Software Ideas Modeler. How to Create a UML Component Diagram — [Електронний ресурс]. — Режим доступу: https://www.softwareideas.net/a/1927/how-to-create-a-uml-component-diagram (дата звернення: 27.04.2025).
13. Lucid Software. UML Package Diagram — [Електронний ресурс]. — Режим доступу: https://lucid.co/templates/uml-package-diagram (дата звернення: 27.04.2025).
14. Lifewire. Choosing a User-Friendly Database Management System (DBMS) — [Електронний ресурс]. — Режим доступу: https://www.lifewire.com/database-management-system-1019609 (дата звернення: 27.04.2025).lifewire.com+1lifewire.com+1
15. Lifewire. What Is BASE in Database Engineering? — [Електронний ресурс]. — Режим доступу: https://www.lifewire.com/abandoning-acid-in-favor-of-base-1019674 (дата звернення: 15.05.2025).lifewire.com
16. Wired. Spime Watch: the Message Queuing Telemetry Transport (MQTT) — [Електронний ресурс]. — Режим доступу: https://www.wired.com/2013/05/spime-watch-the-message-queuing-telemetry-transport-mqtt (дата звернення: 15.05.2025).wired.com
17. ArtofBA. Основні типи архітектури програмного забезпечення — [Електронний ресурс]. — Режим доступу: https://www.artofba.com/uk/post/main-types-of-software-architecture (дата звернення: 15.05.2025).

18. HiveMQ. MQTT Tutorial: An Easy Guide to Getting Started with MQTT — [Електронний ресурс]. — Режим доступу: https://www.hivemq.com/blog/how-to-get-started-with-mqtt/ (дата звернення: 15.05.2025).
19. Amazon Web Services. What is MQTT? — [Електронний ресурс]. — Режим доступу: https://aws.amazon.com/what-is/mqtt/ (дата звернення: 15.05.2025).
20. VPN Unlimited. N-рівнева архітектура (N-tier architecture) — [Електронний ресурс]. — Режим доступу: https://www.vpnunlimited.com/ua/help/cybersecurity/n-tier-architecture (дата звернення: 15.05.2025).
21. PostgreSQL Documentation — [Електронний ресурс]. — Режим доступу: https://www.postgresql.org/docs/ (дата звернення: 19.05.2025).
22. Podman User Guide — [Електронний ресурс]. — Режим доступу: https://podman.io/getting-started/ (дата звернення: 19.05.2025).
23. GitHub Guides — [Електронний ресурс]. — Режим доступу: https://guides.github.com/ (дата звернення: 19.05.2025).
24. Mosquitto MQTT Documentation — [Електронний ресурс]. — Режим доступу: https://mosquitto.org/documentation / (дата звернення: 19.05.2025).