НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці

ДО ЗАХИСТУ ДОПУЩЕНО:
В. о. завідувача кафедри
_____ Олександр КОВАЛЬ
«___»_____202_ р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення інтелектуальних кібер-фізичних систем в енергетиці»

спеціальності 121 Інженерія програмного забезпечення
на тему: «Мобільний застосунок керування електроенегією споживача»

Виконала:
студентка IV курсу, групи ТВ-11
Барабаш Маріна Володимирівна _____ _____
(прізвище, ім'я, по батькові) (підпис)

Керівник:
Професор, д.н.т., Федорова Н. В. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент:
_____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студентка _____

Київ — 2025

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці
Рівень вищої освіти перший (бакалаврський)
Спеціальності: 121 Інженерія програмного забезпечення
Освітньо-професійна програма: «Інженерія програмного забезпечення інтелектуальних кібер-фізичних систем в енергетиці»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Олександр КОВАЛЬ
(підпис)
«___»_____202_ р.

ЗАВДАННЯ
на дипломну роботу студенту
Барабаш Маріні Володимирівні
(прізвище, ім’я, по батькові)

1. Тема роботи
Мобільний застосунок керування електроенергією користувача
керівник роботи Федорова Наталя Володимирівна, д.н.т., професор
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом по університету від «___»_____2024 р. №_____

2. Строк подання студентом роботи 12 червня 2025 р.
3. Вихідні дані до роботи мова програмування Kotlin, середовище розробки Android Studio, мова програмування Python, середовище розробки VS Code.
4.Зміст пояснювальної записки (перелік питань, які потрібно розробити) розробити систему, що надаватиме рекомендації користувачу для використання електроенергії, прогнозуватиме витрати та повідомлятиме про відключення електроенергії; розробити мобільний застосунок з інтеграцією створеної системи зі зручним інтерфейсом.

5. Перелік ілюстративного матеріалу Діаграма взаємодії компонентів системи, Діаграма прецедентів програмного застосунку, Діагарама бази даних. _____
6. Дата видачі завдання «___»_____2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Строки виконання етапів роботи	Примітки
1.	Отримання завдання	01.10.2024	Виконано
2.	Дослідження предметної області	01.10.2024–01.11.2024	Виконано
3.	Дослідження існуючих рішень	01.11.2024–01.12.2024	Виконано
4.	Постановка вимог до проєктування системи	14.04.2025–27.04.2025	Виконано
5.	Розробка програмного продукту	27.04.2025–11.05.2025	Виконано
6.	Тестування програмного продукту	12.05.2025–18.05.2025	Виконано
7.	Захист програмного продукту	14.05.2025	Виконано
8.	Оформлення дипломної роботи	15.05.2025 – 25.05.2025	Виконано
9.	Передзахист		Виконано
10.	Захист		

Студент _____ _____Барабаш М.В._____
(підпис) (прізвище та ініціали,)

Керівник роботи _____ _____Федорова Н. В._____
(підпис) (прізвище та ініціали,

РЕФЕРАТ

Структура та обсяг дипломної роботи. Робота містить 50 сторінки, 24 рисунків, 1 додатот та 8 посилань.

Метою роботи є забезпечення оптимального використання електроенергії шляхом розробки мобільного застосунку для керування споживанням електроенергії.

Для досягнення поставленої мети виконано такі завдання:

- проаналізовано наявні рішення для поставленої задачі;
- спроектовано архітектуру системи;
- розроблено методики для надання користувачу рекомендацій;
- обрано модель прогнозування електроенергії.

Розроблено мобільний застосунок, що в режимі реального часу здатний передавати дані з лічильника будинка, а також попереджати про планові відключення електроенергії. Крім цього, система надає рекомендації щодо дій користувача для ефективного використання енергомережі.

Практичне значення одержаних результатів полягає у створенні програмного застосунку, що допоможе користувачам використовувати оптимально електроенергію з врахуванням навантажень на мережу та відключень електроенергії.

Ключові слова: мобільний застосунок, оптимальне використання, електроенергія, рекомендації, аналіз даних, прогнозування.

ABSTRACT

Structure and scope of the thesis. The work contains 50 pages, 24 figures, 1 appendic and 8 references.

The aim of the work is to ensure optimal use of electricity by developing a mobile application for managing electricity consumption.

To achieve this goal, the following tasks were performed:

- existing solutions for the task were analyzed;
- the system architecture was designed;
- methods were developed for providing recommendations to the user;
- an electricity forecasting model was selected.

A mobile application was developed that is capable of transmitting data from the house meter in real time, as well as warning about planned electricity outages. In addition, the system provides recommendations on user actions for the effective use of the power grid.

The practical significance of the results obtained is in creating a software application that will help users use electricity optimally, taking into account network loads and power outages.

Keywords: mobile application, optimal use, electricity, recommendations, data analysis, forecasting.

ЗМІСТ

ВСТУП 4
1 ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ 6
Висновки до розділу 1 8
2 АНАЛІЗ НАЯВНИХ РІШЕНЬ 9
Висновки до розділу 2 10
3 ЗАСОБИ РОЗРОБКИ 12
3.1 Проектування архітектури системи 12
3.2 Вибір мови програмування та фреймворків 15
3.3 Вибір СКБД 17
3.4 Вибір моделі прогнозування 18
3.5 Вибір бібліотек 19
3.6 Вибір середовища розробки 20
Висновки до розділу 3 22
4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ 23
4.1 Основні компоненти системи 23
4.2 Моделювання взаємодії між користувачем та системою 25
4.3 Моделювання бази даних 27
4.4 Побудова моделі для прогнозування енергоспоживання 30
4.5 HTTP-запити та підключення серверної частини до мобільного застосунку 32
4.6 Тестування системи за допомогою unit-тестів 33
Висновки до розділу 4 34
5 МЕТОДИКА РОБОТИ ЗАСТОСУНКУ З УРАХУВАННЯМ РЕЖИМІВ ФУНКЦІОНУВАННЯ 36

Висновки до розділу 5 40
6 РОБОТА КОРИСТУВАЧА З СИСТЕМОЮ 42
Висновки до розділу 6 47
ВИСНОВКИ 49
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 50
ДОДАТОК А 51

ВСТУП

З кожним етапом розвитку людство все більше і більше вимагало ресурсів для свого існування. На сьогоднішній день, коли люди досягли етапу технологічного розвитку зросла й потреба у використанні енергоресурсів, зокрема електроенергії. Її споживання досягає десятків тисяч терават-годин кожного року усім населенням планети Земля. І ця кількість буде зростати з розвитком промисловості, урбанізацією міст, цифрових технологій та й збільшення самої кількості людей.

Надмірне та неефективне споживання електроенергії впливає не тільки на навколишнє середовище, а й на соціальну, економічну та моральну сфери життя людства. Тому із підвищенням потреб у електропостачанні зростають й вимоги до ефективного та ощадного використання енергії, як на рівні підприємств так і на рівні індивідуальних споживачів. Розумне використання електроенергії – перший крок до енергоефективного, екологічного та безпечного майбутнього людства.

У зв’язку з цим з’являється не просте питання – як оптимально використовувати електроенергію? Важливо впроваджувати інноваційні підходи до контролю та управління споживанням усіма можливими сучасними методами.

В умовах сучасності, коли розвиток цифрових технологій зростає, зростає й можливість використання різних засобів для оптимального користування електроенергією. Смартфони вже давно вкорінилися у життя людства. Тепер це не просто пристрій для зв’язку, а міні-комп’ютери через які можна навчатися, спілкуватися, працювати, розважатися, відпочивати та керувати процесами у нашому повсякденному житті. Вони є хорошим засобом для відстежування енерговитрат.

Індивідуальним споживачам електроенергії важко слідкувати за своїми витратами та навіть не мають повного уявлення про їхні обсяги та структуру споживання, не кажучи вже про поняття навантаження мережі в одному будинку.

Для цього мешканцям багатоповерхівок варто надавати поради для ефективного використання, враховуючи не лише їхні витрати, а й витрати їхніх сусідів.

Важливо надати технологію, що дозволить їм робити це без великих складнощів. З появою смарт-лічильників збільшилась можливість для розробки програмних продуктів, що дозволяють відстежувати витрати в реальному часі, переглядати та робити прогнози про використання електроенергії. Такі технології можуть навіть надавати рекомендації щодо зниження споживання та ефективного використання енергії.

1 ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ

В умовах зростаючих тарифів на електроенергію, енергетичних криз та зниження доступності енергоносіїв, споживачам потрібен ефективний інструмент, який би надавав актуальну інформацію про витрати енергії та рекомендації для їх зменшення. Мета роботи полягає в забезпеченні оптимального використання електроенергії індивідуальними споживачами шляхом створення мобільного застосунку, що дозволяє здійснювати контроль, моніторинг і прогнозування енергоспоживання.

Об’єктом дослідження є процес керування споживанням електроенергії на рівні індивідуальних споживачів — тобто фізичних осіб, домогосподарств або малих підприємств, які прагнуть зменшити свої витрати та раціонально використовувати ресурси. Ці користувачі зазвичай не мають професійних знань у сфері енергозбереження, тому засоби керування мають бути максимально простими, наочними та автоматизованими.

Предметом дослідження виступає методика керування електроспоживанням за допомогою мобільного застосунку, який взаємодіє з цифровими приладами обліку, зокрема, смарт-лічильниками, виконує аналіз зібраних даних та реалізує функції прогнозування, візуалізації та сповіщення. Особлива увага приділяється розробці алгоритмів, що дозволяють не лише відображати поточні витрати електроенергії, а й попереджати перевищення допустимих лімітів споживання на основі щоденних прогнозів.

Мобільний застосунок для керування споживанням електроенергії індивідуального користувача створено передбачає забезпечення раціонального використання ресурсів, зниження витрат, а також своєчасного інформування про загрози перебоїв постачання. Його робота охоплює кілька режимів, кожен з яких передбачає окрему логіку обробки даних, інтерфейсну поведінку та механізми взаємодії з користувачем і джерелами даних.

У межах предметної області розглядаються:

- принципи безпечного збереження та передавання даних;
- використання бібліотек і фреймворків для обробки даних;
- способи виведення рекомендацій користувачу;
- можливість адаптації сповіщень під різні режими (до прикладу, аварійні відключення, блекаути).

Сучасні системи обліку електроенергії, такі як смарт-лічильники, відкривають широкі можливості для автоматизації процесу енергоспоживання. Застосунок, який реалізує ці можливості, має на меті допомогти користувачам не лише контролювати, а й аналізувати свою поведінку щодо споживання енергії, а також коригувати її згідно з зовнішніми критеріями.

Мобільний застосунок для керування електроенергією споживача дозволить не лише отримувати дані про її використання жителями будинку з періоду встановлення лічильника, а й прогнозувати майбутні витрати на основі історичних даних та сезонних змін.

Як спосіб оптимального користування електроенергією система надаватиме поради щодо зменшення використання електроприладів в разі перевищення рекомендованого ліміту. Пік використання електроенергії припадає на час, коли всі повертаються з роботи або ж вихідні дні. У цей період споживачі намагаються контролювати своє використання енергії, вимикаючи прилади. Програма відправлятиме сповіщення з рекомендаціями про відключення певної побутової техніки наприклад електрообігрівача, кондиціонера чи зволожувача повітря. Або ж навпаки у дні з низьким рівнем споживання користувачу буде запропоновано певний енерговитратний процес, до приклад зарядку електромобіля. Це допоможе зменшити навантаження на мережу у дні з високим споживанням користувачами мережі.

Заради зменшення фінансових витрат буде надсилатись порада з денними та нічними тарифами.

В наш час, коли стаються часті відключення електроенергії, важливо знати про час енергопостачання та його зупинки, щоб будувати плани на свій робочий та особистий час. Повідомлення від системи про перепідключення до альтернативного джерела енергії дозволить користувачам мати постійне електропостачання.

Таким чином, предметна область охоплює перетин кількох дисциплін: інформаційних технологій, енергетики та економіки. Актуальність обраної тематики пояснюється глобальними тенденціями енергозбереження, а також бажанням користувачів знизити фінансове навантаження на власні бюджети.

Висновки до розділу 1

У цьому розділі було охарактеризовано предметну область, пов’язану з керуванням енергоспоживанням на рівні індивідуальних користувачів. Було обґрунтовано актуальність створення інструменту, що дозволяє ефективно контролювати, аналізувати та прогнозувати витрати електроенергії, особливо в умовах зростання тарифів, частих відключень електропостачання та необхідності енергозбереження.

2 АНАЛІЗ НАЯВНИХ РІШЕНЬ

В умовах зростаючого навантаження на енергетичні мережі, збільшення вартості електроенергії, а також необхідності зниження викидів парникових газів, особливого значення набуває раціональне та оптимальне використання електричної енергії. Застосування сучасних інформаційних технологій у побутовому секторі дозволяє автоматизувати контроль за витратами електроенергії та допомагає користувачам приймати ефективні рішення щодо її споживання. У цьому контексті мобільні застосунки виступають як зручний інструмент для індивідуального керування електроспоживанням у реальному часі.

За мету було поставлено розробку мобільного застосунку, що забезпечує оптимальне використання електроенергії споживачами за рахунок збору, обробки, аналізу даних та формування рекомендацій для оптимального використання електроенергії. Для реалізації цієї мети варто вирішити декілька технічних та дослідницьких питань.

Варто провести аналіз вже існуючих програмних рішень, створених на сьогоднішній день, аби краще сформувати завдання перед розробкою. Візьмемо до прикладу три інструменти запропонованих ринком для керування електроенергією: Smappe, Green Energy Management і My Energy.

Smappe – це смарт-платформа для управління моніторингу енергоспоживання, що інтегрується зі смарт-лічильником. Її головна задача – надавати доступ до даних з нього в реальному часі. До того ж, ця система допомагає користувачам визначати та оптимізувати витрати.

Її головними перевагами є інтуїтивний інтерфейс, аналіз в реальному часі та можливість взаємодіяти з різними пристроями. На противагу цьому смарт-платформа залежна від точності даних. Точні показники програми завжди потребують правильності роботи смарт-лічильників, які можуть мати обмежену точність. Також деякі функції цієї системи не доступні в усіх регіонах світу. Наприклад інтеграція з локальними енергетичними мережами.

Green Energy Management – це система, що дозволяє користувачам контролювати та оптимізувати споживання електроенергії. Вона орієнтована на людей, що бажають зменшити забруднення навколишнього середовища. З врахуванням використання відновлювальних джерел енергії, платформа надає споживачам можливість слідкувати за своїми витратами електрики й рекомендації для оптимізації енергоспоживання.

Головними перевагами цього продукту є надання пропозицій для підвищення енергоефективності на основі даних та детальний аналіз витрат. В той час, найбільшим недоліком системи є складність інтеграції з іншими енергетичними системами, особливо в старих будинках.

My Energy – цей мобільний застосунок отримує дані зі смарт-лічильника і надає звіти про витрати електроенергії та допомагають спожтвачам оптимізувати використання. Система надає можливість порівнювати свої витрати з витратами інших користувачів, що має на меті – мотивування до економії енергії.

Легкість у використанні та налаштуванні додатку є його позитивними сторонами, а от відсутність глибокого аналізу даних та їх не точність навпаки – негативними.

Висновки до розділу 2

У цьому розділі було проведено аналіз трьох існуючих програмних рішень у сфері моніторингу та оптимізації енергоспоживання: Smappe, Green Energy Management та My Energy. Розгляд функціоналу, переваг та недоліків кожного з них дозволив виявити як загальні тенденції розвитку подібних систем, так і конкретні прогалини, які залишаються не закритими поточними рішеннями.

Зокрема, більшість існуючих продуктів зосереджуються на:

- зборі даних у реальному часі,
- наданні рекомендацій щодо економії,
- підтримці відновлювальних джерел енергії,

- мотивації користувача через порівняння з іншими.

Втім, також були виявлені обмеження:

- залежність від точності стороннього обладнання (смарт-лічильників),
- складність інтеграції в застарілу інфраструктуру,
- обмеженість функціоналу в окремих регіонах,
- поверхневий або неточний аналіз енергоспоживання.

Отже, зроблений аналіз підтверджує актуальність та доцільність розробки власного мобільного застосунку, який би поєднував сильні сторони наявних рішень та усував їх ключові недоліки. Особлива увага в майбутньому проєкті буде приділена адаптації до локальних умов, точному прогнозуванню, персоналізованим рекомендаціям, інтеграції з сонячними панелями та зрозумілому інтерфейсу.

3 ЗАСОБИ РОЗРОБКИ

Для реалізації програмного продукту варто обрати сучасні інструменти та технології для його реалізації. Правильний їх підбір забезпечить ефективну розробку та зручне тестування. Засоби були підібрано спеціально під вимоги до проекту з урахуванням всіх можливостей застосунку.

3.1 Проектування архітектури системи

Архітектура системи має клієнт-серверну структуру, що розділена між двома важливими частинами – backend та frontend. На рисунку 3.1 зображено їхню взаємодію.

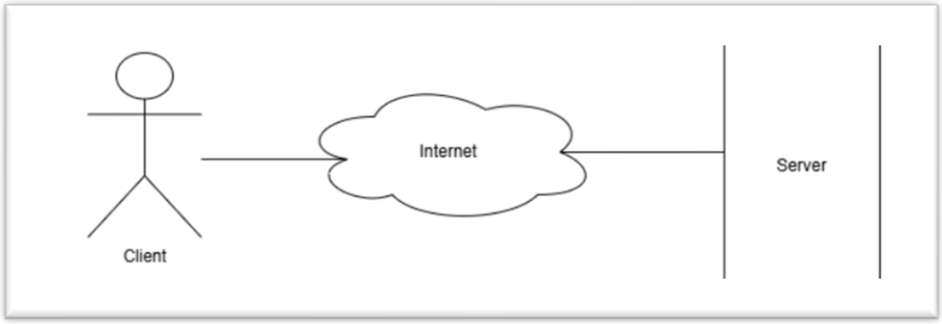

Рисунок 3.1 – Архітектура клієнт-сервер

Серверна частина побудована наближено за принципом Model-View-Controller (MVC) – найпопулярніший та найпоширеніший підхід у розробці backend для веб-застосунків.

MVC — це архітектурний шаблон програмного забезпечення, який широко застосовується для побудови інтерфейсів користувача. Його суть полягає в поділі програми на три основні компоненти: модель, представлення та контролер. Коли користувач здійснює певну дію (наприклад, вводить дані або натискає кнопку), контролер обробляє цю подію, при потребі оновлює модель, а також визначає, яке представлення потрібно показати у відповідь. Такий підхід забезпечує логічну організацію коду та спрощує його підтримку. [1]

На рисунку 3.2 зображено зв’язок між компонентами цього патерну.

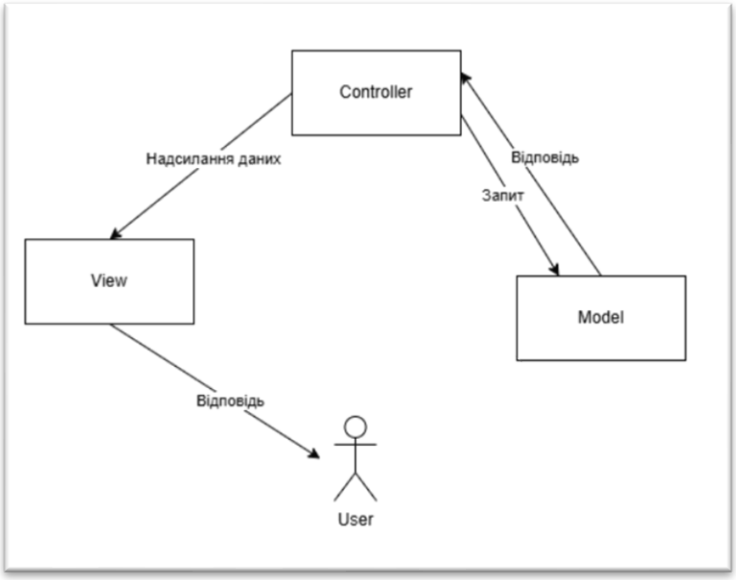

Рисунок 3.2 – Архітектурний шаблон MVC

Ця архітектура чітко відділяє структуру даних, обробку запитів та логіку взаємодії між компонентами системи. Так, як FastApi не вимагає чіткого прив’язування до принципу MVC, у структурі були внесені певні зміни.

Модель – це компонент, що відповідає за опис структури даних та взаємодію з базою даних. Їхню роль в проекті виконують моделі створені завдяки технології ORM, що полегшують роботу з БД, замінивши пряме звернення до SQL-запитів створеними об’єктами, чим полегшує розробку коду, спрощує його та дозволяє здійснювати контроль типів. Для роботи з кожною таблицею з бази даних створено клас Python, що містять відповідні поля, їхні типи та зв’язки між різними таблицями. [6]

Контролери – це маршрутизатори, які визначають endpoints REST API. У розробленому проекті контролери згруповані в окремі модулі, де для кожної функціональної зони визначено набір HTTP-запитів. Ці запити отримуються від клієнтської сторони, викликають певні сервіси та виводять відповідь у вигляді JSON. Головна задача контролерів виступати посередниками між користувачем та бізнес-логікою застосунку.

Сервіси – це не стандартний компонент MVC підходу розробки архітектури програмного продукту, однак його було застосовано на серверній частині розроблюваного застосунку. Вони виконують функції для реалізації специфічних завдань продукту, такі як обробку даних, прогнозування, аналітику та створення рекомендацій. Ця частина архітектури системи дозволяє уникати надмірного навантаження на контролери, зберігати логіку в одному місці та повторно використовувати її в різних частинах програми.

У проекті компонент вигляду (View) не використане у вигляді HTML або графічного представлення, а як JSON-відповідь, створені на основі роботи з сервером. Завдяки йому, кожен запит повертає повертає строго визначену відповідь, що дозволяє клієнтській частині надійно відображати її у мобільному застосунку, не турбуючись про нестабільність або змінність формату.

Отже, архітектура бекенду побудована на принципах чіткої відповідальності кожного шару. Моделі забезпечують доступ до даних, контролери приймають запити та делегують обробку, сервіси реалізують бізнес-логіку, а JSON-відповіді є уніфікованим способом представлення результатів. Такий підхід сприяє кращій підтримці коду, полегшує розширення функціоналу та підвищує загальну стабільність і якість системи.

У клієнтській частині для побудови архітектури використовується принцип Model-View-ViewModel (MVVM) – одного з найпоширеніших способів побудови мобільних застосунків. Цей спосіб допомагає прямо розділити завдання між компонентами інтерфейсу та логікою керування станом. Цей патерн містить у собі три основні компоненти – вигляд, модель та модель представлення. На рисунку 3.3 зображено зв’язок між компонентами MVVM.

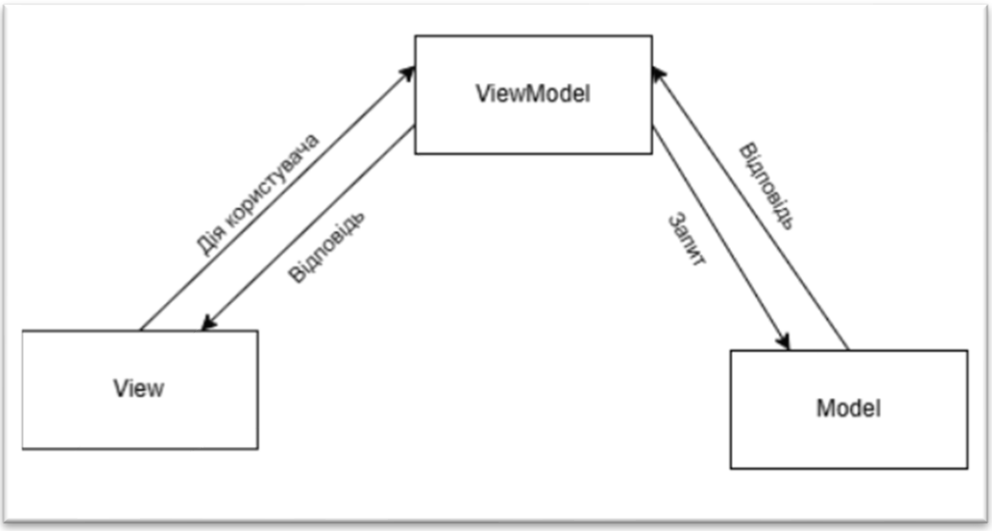

Рисунок 3.3 – Архітектурний шаблон MVVM

Компонент вигляду створений завдяки Jetpack Compose. Кожен екран створений завдяки анотації @Composable є окремим блоком зі спеціальним функціоналом.

Модель представлення виконує роль зв'язку між двома іншими компонентами архітектури системи представлення та моделю. У ній міститься логіка обробки подій, збереження станів системи та здійснюються виклики до серверної частини.

В той же час модель виконує роль для отримання та обробку даних. Головною їхньою задачею в проекті є виконання HTTP-запитів, які перетворюють відповіді від API у внутрішні моделі та передають їх до моделей представлення.

Цей патерн дозволяє забезпечити високу реактивність та плавність інтерфейсу, зручність тестування та масштабовану структру проекту.

3.2 Вибір мови програмування та фреймворків

Архітектура проекту побудована таким чином, що обидві частини — серверна (бекенд) і клієнтська (мобільний застосунок) — реалізовані із застосуванням сучасних мов програмування і актуальних фреймворків. Вибір був

зроблений на користь Python з FastAPI для серверної частини та Kotlin з Jetpack Compose для мобільного застосунку.

Python є однією з найпопулярніших мов програмування у світі, що вирізняється простим синтаксисом, високою читабельністю коду та потужною екосистемою бібліотек. Завдяки цьому Python є ідеальним вибором для швидкої розробки серверної логіки, побудови REST API, обробки даних та реалізації алгоритмів машинного навчання. У контексті цього проекту Python було обрано через кілька важливих причин:

- швидкий старт: Python дозволяє дуже швидко створити прототип і поступово масштабувати його до повноцінного серверного застосунку;
- підтримка багатьох бібліотек: для роботи з базами даних, для авторизації, для аналітики та побудови прогнозів;
- придатність до інтеграції з іншими мовами та мікросервісами.

FastAPI, як фреймворк дозволяє реалізувати багатошарову архітектуру з чітким поділом відповідальностей: моделі (SQLAlchemy), контролери (роутери), сервіси (бізнес-логіка), схеми (Pydantic). Усе це робить код структурованим, масштабованим і зручним для підтримки. [7]

Для реалізації клієнтської частини було обрано мову програмування Kotlin та фреймворк Jetpack Compose. Такий вибір зумовлений як технічними характеристиками платформи Android, так і сучасними тенденціями мобільної розробки. Kotlin на сьогодні є офіційною мовою для Android-застосунків, яка активно підтримується компанією Google. Це мова, що поєднує в собі простоту, потужність та безпечність. Її синтаксис є лаконічним, але водночас достатньо виразним, щоб забезпечити читабельність та підтримуваність коду навіть у великих проєктах. У порівнянні з Java, Kotlin дозволяє зменшити обсяг коду, зменшити ймовірність помилок завдяки системі нульової безпеки (null-safety), а також забезпечити більш гнучку структуру програмних модулів за рахунок використання функціонального стилю. [8]

Важливою перевагою Kotlin у межах цього проекту є його тісна інтеграція з системою Android і можливість ефективно працювати з усіма стандартними інструментами, такими як ViewModel, Navigation, LiveData, Coroutines та іншими компонентами Jetpack. У той же час Kotlin добре підходить для реактивної розробки, де важливою є можливість відстеження змін стану та миттєвого оновлення інтерфейсу користувача.

Jetpack Compose, який виступає як фреймворк для побудови інтерфейсу, є відносно новим, але вже встиг стати основним підходом до створення UI в Android-застосунках. Його поява стала логічним розвитком декларативних парадигм у мобільній розробці. На відміну від традиційного способу побудови інтерфейсу на основі XML-розмітки, Jetpack Compose дозволяє описувати вигляд застосунку безпосередньо в коді Kotlin. Це значно пришвидшує розробку, оскільки виключає необхідність синхронізації між кодом і шаблонами розмітки. Інтерфейс створюється за допомогою функцій, позначених анотацією @Composable, які можуть бути вкладені одна в одну, утворюючи ієрархію елементів.

3.3 Вибір СКБД

Для збереження, обробки та аналізу структурованих даних у межах цього проєкту використовується реляційна система керування базами даних PostgreSQL. Це сучасна, високопродуктивна, надійна і відкрита СКБД, що підтримує широкий спектр функціональних можливостей, необхідних для реалізації складних інформаційних систем. PostgreSQL має гнучку архітектуру, що дозволяє ефективно працювати як у локальному середовищі, так і в хмарі, легко масштабується, забезпечує високу безпеку даних та відповідає промисловим стандартам.[3]

У проекті PostgreSQL виступає основним сховищем усіх ключових даних: інформації про користувачів, лічильники, історію енергоспоживання, відключення електроенергії, повідомлення, а також аналітичні дані. Її використання дозволяє будувати логічні зв’язки між таблицями, визначати обмеження цілісності,

здійснювати складні запити з об’єднаннями, групуванням, агрегацією й фільтрацією. Це особливо важливо для забезпечення надійної роботи застосунку, який постійно оновлює дані в реальному часі та виконує численні аналітичні обчислення.

У контексті розгортання PostgreSQL є надзвичайно зручною: вона легко налаштовується як локально для розробки, так і у хмарному середовищі. Це дозволяє не лише розгортати систему у продакшн-середовищі, а й автоматизувати її оновлення, резервне копіювання та міграцію без втрати даних. [3]

Завдяки своїй потужності, надійності та сумісності з обраною серверною платформою вона дозволяє забезпечити стабільну роботу системи, швидкий доступ до даних і можливість легкої адаптації до майбутніх змін або розширення функціоналу.

3.4 Вибір моделі прогнозування

Для задачі прогнозування споживання електроенергії було обрано модель градієнтного бустингу на основі дерев рішень, реалізовану в бібліотеці LightGBM у вигляді класу LGBMRegressor. Дана модель поєднує високу точність з ефективною обробкою великих обсягів даних і складних зв’язків між ознаками.

Дерева рішень дозволяють візуалізувати шлях прийняття рішення та легко зрозуміти, які ознаки мають найбільший вплив на результат. Це особливо важливо для прикладних задач, де користувачам важливо розуміти логіку прогнозу. [5]

Енергоспоживання залежить від багатьох факторів (час доби, день тижня), і ці залежності зазвичай є нелінійними. Дерева рішень добре справляються з виявленням і моделюванням таких складних зв’язків без необхідності попереднього перетворення ознак.

На відміну від лінійних моделей, дерева рішень менш чутливі до викидів у даних, що підвищує стабільність прогнозу при наявності реальних, іноді неідеальних даних.

Завдяки гістрограмній реалізації та підтримці багатопотоковості, LightGBM значно швидший за класичні алгоритми бустингу при збереженні високої точності, що дозволяє ефективно працювати навіть із великими наборами даних.

3.5 Вибір бібліотек

У сучасному програмному забезпеченні використання сторонніх бібліотек є не просто зручністю, а необхідністю. Завдяки цьому можна суттєво знизити час розробки, зменшити кількість помилок і зосередитися на реалізації бізнес-логіки замість повторного вирішення вже вирішених задач. У межах реалізації цього проекту були обрані бібліотеки, які дозволяють забезпечити стабільну роботу клієнтської та серверної частин, а також їхню ефективну взаємодію між собою.

Для реалізації бекенду основною платформою обрано FastAPI, і відповідно бібліотеки підбиралися з урахуванням сумісності з асинхронною архітектурою, підтримки REST API, безпечного збереження даних та роботи з базою PostgreSQL.

Для роботи з базою даною використовується потужна ORM-бібліотека, що дозволяє працювати з реляційною базою даних у вигляді Python-класів – SQLAlchemy. Вона забезпечує абстракцію над SQL-запитами, підтримує каскадні зв’язки, транзакції та автоматичне оновлення структури таблиць.

Аби полегшити формування структури запитів та їх відповідей було застосовано бібліотеку Pydantic – бібліотека для валідації даних, що тісно інтегрована з FastAPI. Використовуючи її для створення класів запитів гарантується правильність вхідних параметрів на рівні API.

Uvicorn – високопродуктивний ASGI-сервер, що використовується для запуску FastAPI-застосунку. Завдяки своїй асинхронній природі, Uvicorn дозволяє ефективно обробляти велику кількість одночасних підключень. [4]

Для керування міграціями бази даних було встановлено та застосовано Alembic. З його допомогою можна безпечно вносити зміни до структури таблиць

без втрати існуючих даних. Використовуючи її гарантується безпечна робота з PostgreSQL.

У проекті відбувається робота не тільки з базою даних, а й їхня обробка та прогнозування. Pandas є базовою бібліотекою для роботи з табличними даними у Python. Вона дозволяє зручно обробляти історію енергоспоживання у вигляді DataFrame – двовимірної таблиці з індексацією за датами, номерами лічильників та значеннями енергії. Бібліотека надає функції для агрегації даних за днями, тижнями, місяцями, фільтрації аномалій, обчислення статистичних показників (середнє, медіана, максимум, тощо), а також для побудови зведених таблиць та групування за ознаками. У проекті Pandas використовується як основа для попередньої обробки даних перед передачею у моделі прогнозування.

NumPy – це бібліотека для високопродуктивних обчислень з багатовимірними масивами. Хоча її часто використовують як “двигун” для Pandas, у задачах прогнозування NumPy важлива для математичних перетворень, нормалізації, роботи з часовими рядами у вигляді масивів і векторизованих обчислень. У контексті проекту вона корисна для обчислення коефіцієнтів сезонності, трендів та лагів, що передують використанню моделі.

Scikit-learn – одна з найпопулярніших бібліотек для машинного навчання, а LightGBM – це високопродуктивна бібліотека для градієнтного бустингу, яка дозволяє будувати точні моделі для задач регресії, класифікації та ранжування. Модель добре справляється з великими обсягами даних, підтримує категоріальні ознаки, працює швидко та має вбудовані методи оцінки важливості ознак.

3.6 Вибір середовища розробки

У процесі реалізації програмного забезпечення було використано два основні середовища розробки — Android Studio та Visual Studio Code. Їх вибір зумовлений специфікою реалізації клієнтської та серверної частин застосунку.

Для розробки мобільного застосунку, призначеного для платформи Android, було використано Android Studio — офіційне інтегроване середовище розробки (IDE).

Основні переваги Android Studio:

- Повна інтеграція з Android SDK та Jetpack Compose — сучасним підходом до створення інтерфейсів.
- Зручний редактор інтерфейсів з підтримкою live preview.
- Вбудований емулятор Android для тестування застосунку без фізичного пристрою.
- Інтеграція з системами контролю версій, зокрема Git.
- Підтримка Kotlin — основної мови розробки в даному проєкті.

Android Studio надала всі необхідні інструменти для створення адаптивного інтерфейсу користувача, реалізації бізнес-логіки, обробки мережевих запитів та інтеграції з серверною частиною.

Для реалізації серверної частини застосунку, написаної на мові Python з використанням фреймворку FastAPI, було використано редактор коду Visual Studio Code (VS Code). Це легкий, але потужний редактор з розширеною підтримкою Python через офіційне розширення.

Основні причини вибору VS Code:

- Підтримка інтерактивного дебагінгу серверної логіки.
- Вбудований термінал, що спрощує запуск серверу, роботу з базою даних та інструментами керування пакетами (наприклад, pip).
- Розширення для FastAPI, Pylance, SQL тощо — значно підвищують ефективність роботи.
- Легка інтеграція з Git та можливість швидкого переходу між гілками і комітами.

Використання Visual Studio Code дозволило ефективно реалізувати API для обміну даними з мобільним застосунком, забезпечити обробку запитів, автентифікацію користувачів, та обробку даних зі смарт-лічильника.

Висновки до розділу 3

У цьому розділі було обґрунтовано вибір технологій, архітектурних підходів, мов програмування, бібліотек, СКБД, моделей прогнозування та середовищ розробки, які були використані при реалізації інформаційної системи.

Загалом, обрані засоби розробки продемонстрували свою ефективність і відповідність поставленим вимогам, що дозволило успішно реалізувати функціональну, надійну та масштабовану інформаційну систему.

4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Після вибору інструментів розробки важливим етапом є створення структурованої архітектури програмного продукту. Це включає побудову логічної моделі, визначення компонентів системи, їх взаємодії.

4.1 Основні компоненти системи

Програмна система розумного обліку енергії складається з декількох ключових компонентів, які взаємодіють між собою для забезпечення повного циклу збору, обробки, аналізу та візуалізації даних. На рисунку 4.1 зображено основні компоненти системи та їх взаємодію.

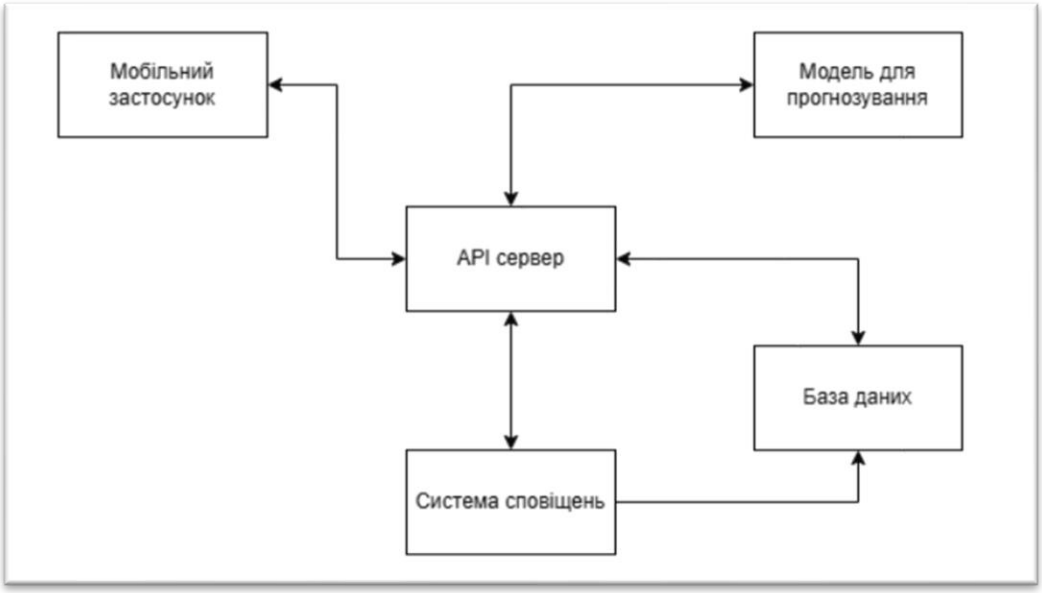

Рисунок 4.1 – Схема взаємодії основних компонентів системи

Центральним елементом є API-сервер, реалізований на основі FastAPI. Саме через нього відбувається вся комунікація між користувачем та внутрішніми службами системи. Сервер приймає запити від мобільного застосунку, обробляє їх, звертається до бази даних, виконує бізнес-логіку і формує відповіді, які повертаються клієнту.

Мобільний застосунок, написаний за допомогою Kotlin і Jetpack Compose, виступає головним інтерфейсом взаємодії користувача з системою. Через нього користувач може переглядати показники енергії, отримувати повідомлення, бачити прогнози споживання та взаємодіяти з власним лічильником. Для збереження важливої інформації локально у мобільному застосунку використовується локальне сховище, побудоване на технології DataStore. Це дозволяє працювати навіть у випадках тимчасової втрати підключення до мережі.

База даних, реалізована на PostgreSQL, відповідає за збереження всіх структурованих даних, включаючи користувачів, лічильники, історичні показники енергоспоживання, планові відключення, повідомлення та інші об’єкти. Усі запити на зчитування чи запис здійснюються API-сервером через ORM SQLAlchemy, що дозволяє ефективно працювати з реляційними зв’язками, зокрема багато до багатьох.

Модель прогнозування, реалізована з використанням бібліотеки LightGBM, отримує вхідні дані з бази, формує прогноз добового споживання і повертає результати на API-сервер. На основі цього прогнозу система може здійснювати аналітику, попереджати користувача про потенційне перевищення ліміту або рекомендувати оптимальні години для використання енергоємних приладів.

Система сповіщень є важливою ланкою у зворотному зв’язку між сервером і користувачем. Вона автоматично створює повідомлення у разі перевищення прогнозованого споживання або при наближенні до запланованого відключення електроенергії. Ці повідомлення зберігаються у базі та надсилаються користувачеві через мобільний застосунок.

Усі ці компоненти утворюють єдину інформаційну систему, що дозволяє користувачу не лише контролювати власне споживання електроенергії, а й отримувати корисні поради, аналіз тенденцій та бути заздалегідь попередженим про ризики чи незручності.

4.2 Моделювання взаємодії між користувачем та системою

Для візуалізації взаємодії користувача з функціональністю програмного забезпечення побудовано діаграму прецедентів, що відображає основні сценарії використання системи. На рисунку 4.2 подано діаграму, яка ілюструє доступні користувачу дії та взаємодію з системою.

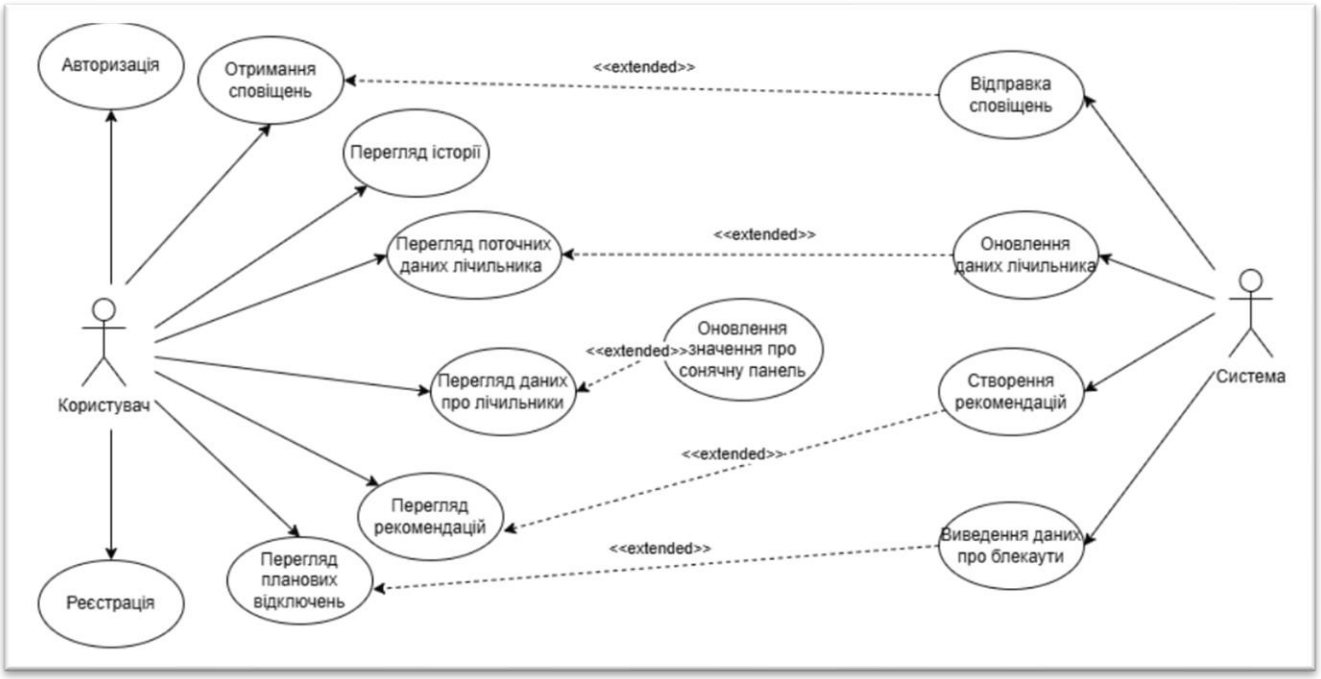

Рисунок 4.2 – Діаграма прецедентів

Після проходження етапу реєстрації та авторизації користувач отримує доступ до ключових функцій системи. Основні дії включають перегляд поточних даних лічильника, які дають змогу контролювати рівень енергоспоживання у реальному часі, а також перегляд історії показників, що дозволяє аналізувати динаміку споживання за попередні періоди (тиждень, місяць, рік).

Система також передбачає перегляд загальної інформації про лічильники, зокрема серійні номери, адреси та технічні характеристики. Цей сценарій містить розширення — оновлення значення про сонячну панель, яке може бути необхідним для уточнення розрахунків рекомендацій.

Ще однією важливою функцією є перегляд рекомендацій, які система формує на основі аналізу поточного та прогнозованого енергоспоживання. Цей прецедент, у свою чергу, активує процес створення рекомендацій системою.

Крім того, користувач має змогу отримувати сповіщення про критичні події, що реалізується через взаємодію з підсистемою повідомлень. У відповідь система може надсилати відповідні сповіщення, наприклад, при досягненні порогу споживання. Виведення інформації про заплановані відключення або нештатні ситуації (наприклад, блекаути) також реалізується окремим прецедентом, який може активуватись як на основі даних, так і за ініціативою користувача.

Діаграма також відображає функціональність, що стосується автоматичних дій з боку системи. Наприклад, оновлення даних лічильника, створення рекомендацій або відправлення повідомлень не ініціюються напряму користувачем, але тісно пов’язані з його активністю. Це підкреслює наявність фонового інтелектуального аналізу, який забезпечує своєчасну реакцію системи на зміни в поведінці споживання.

Ключовим елементом є взаємозв’язок між прецедентами користувача і діями системи. Це забезпечується через розширення, які демонструють логічну залежність: наприклад, при перегляді поточних даних система може автоматично оновити інформацію з лічильника, а при зверненні до рекомендацій — ініціювати їх актуалізацію.

Загалом, побудована діаграма дозволяє повною мірою оцінити логіку взаємодії користувача з програмним забезпеченням. Вона забезпечує наочне представлення архітектури поведінки системи, сприяє розумінню внутрішніх процесів і служить основою для подальшого проєктування інтерфейсу користувача та внутрішніх алгоритмів.

4.3 Моделювання бази даних

Одним з ключових компонентів інформаційної системи є база даних, яка забезпечує надійне зберігання, ефективний доступ та логічно структуровану організацію інформації. У межах даного проекту було реалізовано реляційну базу даних, яка слугує основою для зберігання інформації про користувачів, енергетичні лічильники, показники споживання електроенергії, повідомлення системи та графіки відключення електропостачання. Для визначення структури бази даних використовувалась ORM-бібліотека SQLAlchemy, що дозволяє описати моделі на рівні класів мови Python та автоматично відображати їх у вигляді відповідних таблиць у СУБД PostgreSQL. На рисунку 4.3 зображено схему змодельованої бази даних.

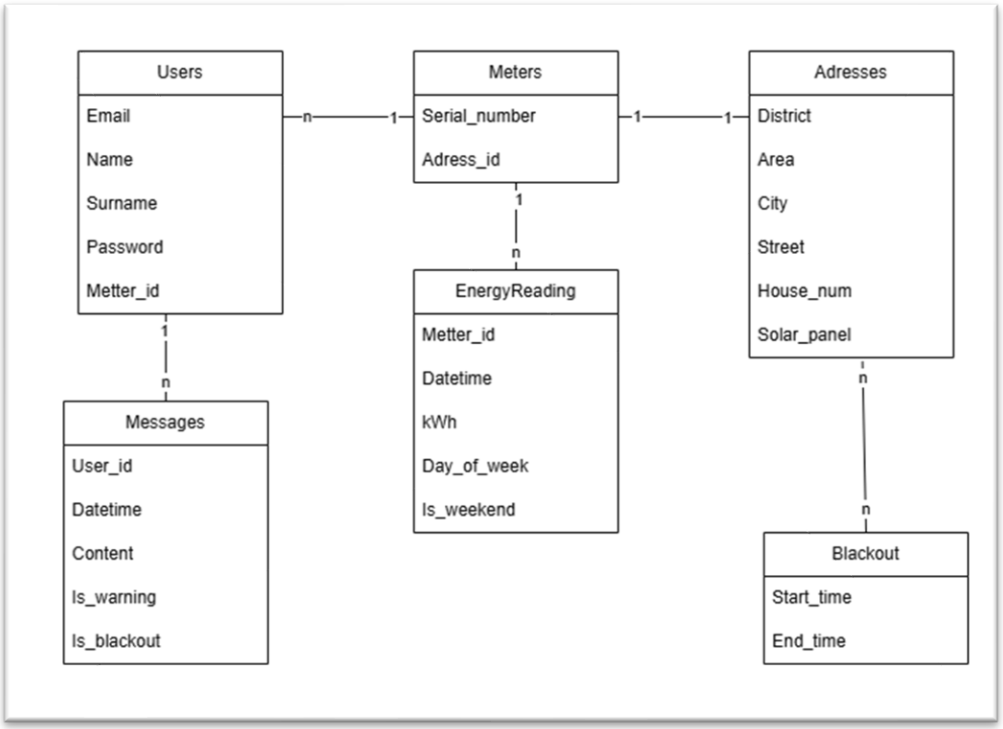

Рисунок 4.3 – Модель бази даних

Центральним елементом системи є користувач — особа, що взаємодіє з програмним забезпеченням. Для цього було створено окрему таблицю користувачів, яка містить такі ключові поля, як адреса електронної пошти, ім’я,

прізвище та захешований пароль. Адреса електронної пошти має бути унікальною для кожного користувача та використовується для ідентифікації під час автентифікації.

Оскільки кожен користувач може мати доступ до одного лічильника електроенергії, а кожен лічильник може бути прив’язаний до кількох користувачів, між таблицями користувачів та лічильників реалізовано зв’язок один-до-багатьох.

Таблиця лічильників є однією з найважливіших у структурі бази даних. Вона зберігає докладну інформацію про кожен пристрій обліку споживання енергії. Серед ключових атрибутів — серійний номер (який має бути унікальним) та прив’язку до таблиці адрес, яка має свої власні поля – адміністративна прив’язка (район, місто), а також адреса розташування (вулиця, номер будинку, номер квартири). Додатково передбачено логічний індикатор наявності сонячної панелі — це дозволяє у подальшому враховувати можливість автономного живлення об’єкта.

Кожен лічильник має зв’язки з двома іншими сутностями: з показниками енергоспоживання та з графіками відключення електропостачання. Обидва зв’язки мають тип один-до-багатьох, тобто кожен лічильник може мати багато пов’язаних записів відповідного типу.

Показники енергоспоживання є основним типом даних, що використовується для аналітики та побудови прогнозів. Кожен запис у цій таблиці пов’язаний з конкретним лічильником та містить такі поля, як обсяг спожитої електроенергії (у кіловат-годинах), дата та час фіксації показника, день тижня та індикатор вихідного дня.

Для підтримки цілісності даних та запобігання появі «сирітських» записів передбачено каскадне видалення: при видаленні лічильника всі пов’язані з ним показники споживання автоматично видаляються з бази даних.

У рамках системи реалізовано механізм надсилання повідомлень користувачам. Це можуть бути як загальні інформаційні повідомлення, так і специфічні попередження про небезпеку або повідомлення про планові

відключення електроенергії. Таблиця повідомлень включає текст повідомлення, дату його створення, а також бінарні індикатори, які вказують на тип повідомлення (звичайне, попередження, або повідомлення про планові відключення електроенергії). Кожне повідомлення пов’язане з конкретним користувачем, що дозволяє персоналізувати комунікацію.

Архітектурно реалізовано двосторонній зв’язок між таблицею користувачів і таблицею повідомлень. Видалення користувача спричиняє автоматичне видалення всіх його повідомлень, що відповідає принципам цілісності й економії ресурсів.

Ще однією важливою частиною бази даних є інформація про планові відключення електроенергії — періоди, протягом яких електропостачання до певного об’єкта буде призупинено. Для цього створено окрему таблицю, яка містить час початку та завершення відключення, а також прив’язку до конкретної адреси. Таке моделювання дозволяє здійснювати попередній аналіз впливу відключень на споживання енергії та генерувати попереджувальні повідомлення користувачам про заплановані збої.

Створена база даних повністю підтримує концепцію нормалізованого зберігання інформації, уникнення дублювання даних та забезпечення логічної зв’язаності між об’єктами системи. Реалізовані зв’язки відображають реальні взаємовідносини між користувачами, пристроями обліку, показниками споживання та інфраструктурними подіями. Особливістю моделі є підтримка як зв’язків «один до багатьох» (наприклад, між лічильниками та показниками споживання), так і зв’язку «багато до багатьох» (між користувачами та лічильниками), що дозволяє гнучко масштабувати систему при збільшенні кількості користувачів і пристроїв.

Крім того, використання механізмів каскадного видалення, двосторонніх зв’язків та чіткої типізації полів дозволяє досягти високої надійності при роботі з даними. Така структура забезпечує ефективне функціонування системи в умовах реального навантаження, збереження даних історичного споживання та формування основи для подальшої аналітики, прогнозування та взаємодії з користувачем.

4.4 Побудова моделі для прогнозування енергоспоживання

У рамках даного проекту було розроблено алгоритм прогнозування енергоспоживання з використанням інструментів машинного навчання, що базується на історичних погодинних даних.

Першим і ключовим етапом у реалізації цього підходу стало формування якісного датасету. Джерелом даних виступили архіви погодинного споживання електроенергії у кіловат-годинах. Дані, отримані у вигляді табличних файлів, були оброблені таким чином, щоб зберегти лише ті записи, які містили повну інформацію щодо ключових параметрів, що впливають на споживання. Таким чином, з метою забезпечення достовірності результатів, із вибірки було видалено всі рядки з відсутніми або некоректними значеннями.

Наступним кроком стала побудова нових інформативних ознак, що несуть у собі потенціал покращити здатність моделі до узагальнення. Серед них – календарні та часові характеристики, які відображають ритмічну структуру енергоспоживання. Зокрема, було враховано місяць, день місяця, годину доби, день тижня та інформацію про те, чи є поточний день вихідним. Така деталізація дозволила моделі ефективніше виявляти повторювані шаблони, що зумовлені поведінковими чинниками користувачів.

Особливо важливими у контексті часових рядів є ознаки, що враховують попередні значення цільової змінної. Для створення системи було реалізовано кілька таких підходів. По-перше, враховувалося споживання, що мало місце 30 днів тому, тобто модель отримувала доступ до віддаленої в часі інформації. По-друге, було використано ковзне середнє за попередні сім днів, що дало змогу оцінити короткотермінові тенденції. Таким чином, модель отримувала доступ як до миттєвих, так і до усереднених значень, що дозволяє гнучко поєднувати коротко- та довгострокові залежності.

На основі сформованого набору ознак було проведено підготовку вибірки для навчання моделі. Цільовою змінною виступало фактичне значення

енергоспоживання. Для об'єктивної оцінки продуктивності алгоритму усі дані було розділено на тренувальний та тестовий набори, що дозволяло перевірити здатність моделі узагальнювати нові, невідомі їй спостереження. Розділення виконувалося випадковим чином з фіксованим параметром ініціалізації генератора псевдовипадкових чисел, щоб забезпечити відтворюваність експериментів.

У якості моделі машинного навчання було обрано алгоритм градієнтного бустингу LightGBM, який є сучасним рішенням для задач регресії та класифікації, де потрібно досягти високої точності на структурованих табличних даних.

Після навчання модель була використана для прогнозування значень енергоспоживання на тестовій вибірці. Для оцінки точності прогнозу було використано середньоквадратичну помилку (RMSE), яка дозволяє визначити середнє відхилення прогнозованих значень від фактичних. Ця метрика є чутливою до великих похибок, що робить її особливо корисною для задач, де недопустимі суттєві відхилення у прогнозах.

Особливу увагу було приділено оцінці моделі не лише на рівні окремих годин, але й у розрізі щоденного споживання. З цією метою отримані прогнозні значення були агреговані за календарними днями, що дозволило проаналізувати відхилення у більш загальному вигляді. Таке групування допомагає уникати випадкових флуктуацій, властивих погодинним даним, і краще відображає загальний характер споживання.

У процесі аналізу були порівняні щоденні сумарні прогнози з відповідними фактичними значеннями. Це дозволило виявити загальну ефективність моделі у довготривалій перспективі, а також з'ясувати, наскільки точно вона може передбачити загальне навантаження на мережу протягом дня. Після чого було обчислено добовий RMSE, що є похідною метрикою, але дозволяла виявити додаткові закономірності в точності прогнозування на більш високому часовому рівні.

На завершальному етапі отриману модель було серіалізовано, тобто збережено у вигляді файлу, придатного для подальшого використання у веб-

сервісах або локальних програмах без необхідності повторного навчання. Такий підхід дозволяє інтегрувати модель у систему.

У підсумку, розроблений алгоритм забезпечує надійне та гнучке прогнозування енергоспоживання на основі багатьох факторів, серед яких – календарні та історичні. Його перевага полягає у здатності враховувати як миттєві умови, так і тренди минулих періодів.

4.5 HTTP-запити та підключення серверної частини до мобільного застосунку

Взаємодія між мобільним застосунком і серверною частиною здійснюється за допомогою протоколу HTTP через REST API. Кожна дія користувача, що вимагає доступу до даних або взаємодії з бізнес-логікою, ініціює відповідний HTTP-запит до API-сервера, реалізованого за допомогою фреймворку FastAPI.

З боку клієнта, всі запити надсилаються з використанням бібліотеки Retrofit, яка інтегрується з ViewModel і забезпечує обробку результатів у вигляді асинхронних потоків. Наприклад, при авторизації користувач надсилає POST-запит, передаючи облікові дані, а у відповідь отримує JWT-токен. Цей токен зберігається локально в DataStore і додається до заголовків наступних запитів забезпечуючи захист усіх приватних маршрутів.

Типовими запитами є GET-запити для отримання поточних або історичних даних, POST-запити для внесення нових показників або створення повідомлень, а також PATCH-запити для оновлення параметрів лічильника.

Усі маршрути на сервері мають чітко визначені схеми запиту і відповіді через Pydantic-моделі, що дозволяє клієнту отримувати уніфіковані та перевірені відповіді з API. Сервер також виконує валідацію даних, обробку помилок, ведення логів та автентифікацію.

Таким чином, HTTP-зв’язок між клієнтом і сервером побудований на сучасних принципах REST-архітектури, що забезпечує стабільну, масштабовану та безпечну інтеграцію мобільного застосунку з серверною логікою.

4.6 Тестування системи за допомогою unit-тестів

Юніт-тестування є важливою складовою процесу розробки програмного забезпечення, яка дозволяє перевірити функціональність окремих одиниць коду (функцій, методів, класів) в ізоляції. Метою є виявлення помилок на ранніх етапах, підвищення надійності системи та забезпечення впевненості в тому, що зміни у коді не спричиняють порушення вже реалізованої логіки.

Тестування виконується без взаємодії з реальними зовнішніми сервісами чи базою даних, що дозволяє:

- отримати швидкий зворотній зв’язок;
- уникати побічних ефектів;
- здійснювати незалежну перевірку поведінки компонентів.

У межах проекту було реалізовано юніт-тестування ключових модулів серверної частини FastAPI-застосунку.

Тестуються ендпойнти реєстрації та входу користувача, перевірка правильності валідації, відповідність структури токена, а також обробка помилок у разі неправильних даних.

Юніт-тести перевіряють логіку додавання наявного лічильника до облікового запису користувача, перевірку серійного номера та наявності такого лічильника в базі.

Тестуються функції, що забезпечують доступ до агрегованих даних — за день, тиждень, місяць або рік. У тестах імітуються типові запити до API та перевіряється коректність структури відповіді, наявність усіх очікуваних полів та правильність агрегацій.

Юніт-тести перевіряють правильність обробки запитів до модулів аналітики: генерацію графіків за днями тижня, сезонами, місяцями року, останніми 30 днями тощо. Для тестування використовуються ізольовані підготовлені дані.

Перевіряється логіка виклику модуля прогнозування, порівняння з пороговими значеннями та формування текстових порад користувачеві. Тести імітують ситуації перевищення споживання та перевіряють, що рекомендації формуються коректно.

Висновки до розділу 4

У цьому розділі було детально описано архітектуру та програмну реалізацію інформаційної системи для моніторингу та прогнозування енергоспоживання. Система реалізована за сучасними принципами клієнт-серверної архітектури, що забезпечує гнучкість, масштабованість та розподілення навантаження між мобільним застосунком та серверною частиною.

Описано ключові компоненти системи: мобільний додаток на Kotlin з Jetpack Compose, серверна логіка на FastAPI, реляційна база даних PostgreSQL, модуль машинного навчання з використанням LightGBM, а також система сповіщень і аналітики. Взаємодія між компонентами реалізована через REST API з використанням сучасних бібліотек, таких як Retrofit, SQLAlchemy, Pandas та інші.

Особливу увагу приділено структурі та зв’язкам у базі даних, які відображають реальні взаємозв’язки між користувачами, лічильниками, показниками енергоспоживання та повідомленнями. Було реалізовано логіку прогнозування енергоспоживання, яка базується на історичних даних, часових ознаках та ковзних середніх, що забезпечує точність прогнозу на добовому рівні.

Також розглянуто механізм HTTP-взаємодії між клієнтом і сервером, що побудований на принципах REST-архітектури, та реалізовано юніт-тестування основних компонентів серверної частини. Це дозволяє гарантувати стабільність та

правильність роботи окремих модулів, підвищує надійність і полегшує подальшу підтримку системи.

У результаті розробки створено повноцінний прототип системи інтелектуального обліку енергії, що охоплює весь цикл: від збору даних до аналітики, прогнозування та взаємодії з користувачем через мобільний інтерфейс.

5 МЕТОДИКА РОБОТИ ЗАСТОСУНКУ З УРАХУВАННЯМ РЕЖИМІВ ФУНКЦІОНУВАННЯ

Застосунок працює в кількох режимах, які визначаються в залежності від зовнішніх умов, показників енергоспоживання, наявності електропостачання та часу доби. Кожен режим має власну логіку обробки даних, формування повідомлень та взаємодії з користувачем. Нижче наведено детальний опис роботи системи в кожному з таких режимів.

У нормальному режимі система функціонує у штатному режимі без обмежень. Користувач має доступ до всіх функцій: перегляд поточних і історичних даних споживання, отримання рекомендацій, перегляд прогнозу, повідомлень та відключень. Постійно надходять нові дані від лічильника, які зберігаються у базі даних кожну годину та передаються моделі прогнозування. Всі розрахунки виконуються автоматично, а користувач може переглядати результати в мобільному застосунку в режимі реального часу. У цей період система також оцінює, чи споживання користувача відповідає прогнозованим значенням, і якщо ні — надсилає відповідні рекомендації.

Ці рекомендації формуються на основі історичних даних попередньо проаналізованих системою, якщо день тижня визначений як час з великими витратами користувачу буде виведено сповіщення про перенесення певного виду роботи на інший день з більш енергоефективним використанням. На рисунку 5.1 зображено сповіщення такого типу.

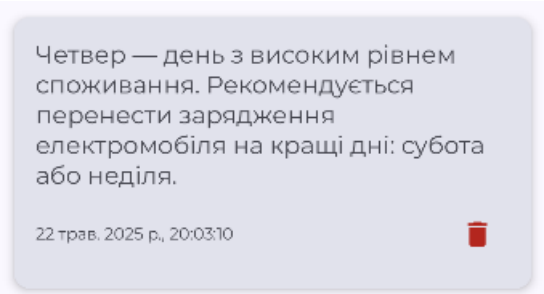

Рисунок 5.1 – Рекомендація перенести завдання з високими енергетичними затратами на інший день

У випадку, коли перехід ліміту споживання електроенергії припадає на середньостатистичний день, але час з піковими навантаженнями система надсилає сповіщення про перенесення дрібної роботи, як от увімкнення бойлера на час з найменшими витратами. На рисунку 5.2 відображено рекомендацію відкласти використання бойлера з восьмої вечора, яка очевидно є піковою годиною навантаження на електромережу, на годину з низькими енергозатратами – п’яту ранку. Крім того, ці завдання слідкують за сезонністю задля уникнення рекомендацій, по типу «вимкнення обігрівача» влітку.

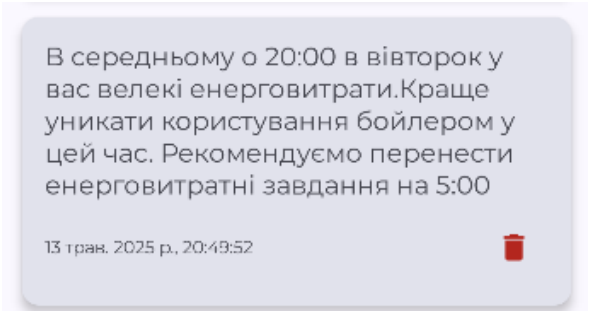

Рисунок 5.2 – Рекомендація перенести завдання з високими енергетичними затратами на іншу годину

Якщо користувач перейшов ліміт у день з оптимальним використанням електроенергії, тоді система реагує у вигляді рекомендації протилежній до першого варіанту. Користувач отримує пораду виконати більше енергоємних завдань.

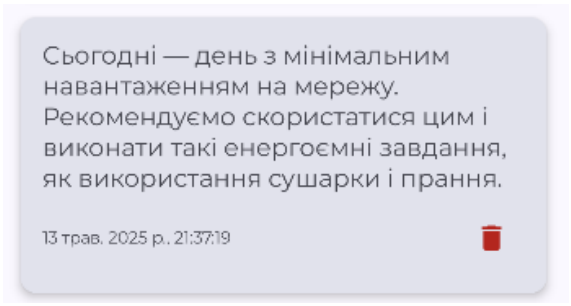

Рисунок 5.3 – Рекомендація виконати більше енергозатратних завдань

Режим оптимального споживання згідно тарифів активується в залежності від часу доби. Система має вбудовану сітку тарифних зон: нічний та денний, що задано в системі. Після отримання прогнозу на добу і раптового перевищення

ліміту система порівнює планове споживання з тарифними зонами і формує рекомендації. На рисунку 5.3 показано приклад сповіщення при денному тарифі. Користувачу пропонують перенести використання пральною машиною чи бойлером за нічного тарифу.

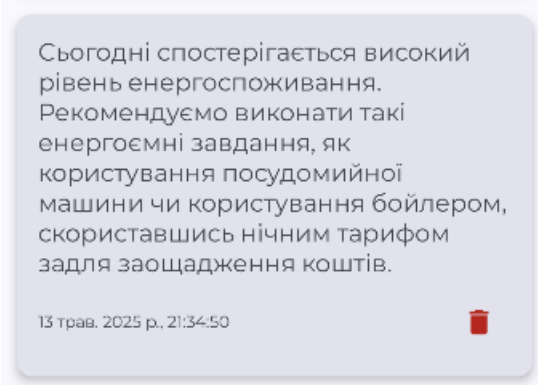

Рисунок 5.4 – Рекомендація використати електроприлади у період нічного режиму

Коли діє нічний тариф, система навпаки буде заохочувати до збільшення кількості використання енергоприладів для заощадження коштів споживача (рисунок 5.6). Такі підказки допомагають зменшити економічні витрати на електроенергію та підвищують енергоефективність.

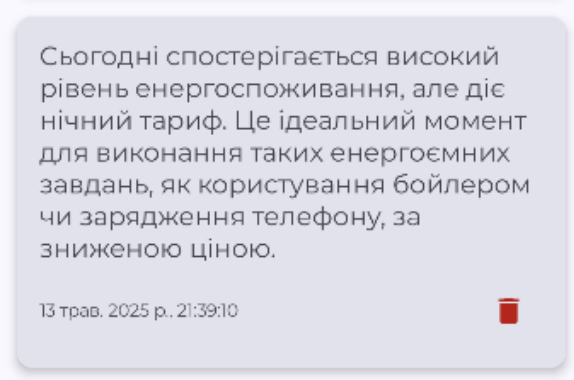

Рисунок 5.5 – Рекомендація скористатись нічним тарифом

Аварійне відключення електроенергії – за цього режиму лічильник більше не оновлює свої значення. Користувачу виводиться повідомлення про аварійне відключення електроенергії (рисунок 5.6).

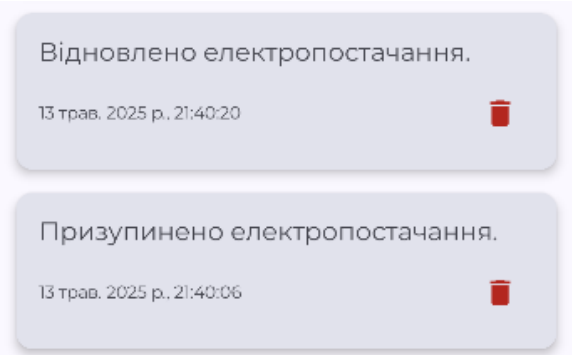

Рисунок 5.6 – Сповіщення про припинення та відновлення електропостачання

За умови, що у будинку є сонячна панель йому пропонується переключитися на альтеративне джерело живлення електроенергії (рисунок 5.7). Якщо електропостачання відновлюється виводиться відповідне цьому повідомлення (рисунок 5.6).

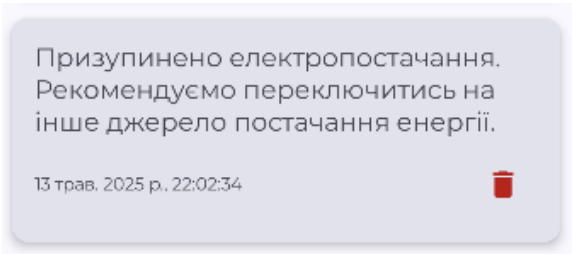

Рисунок 5.7 – Сповіщення з рекомендацією перемкнутися на використання сонячної панелі

Режим планових відключень визначається на основі даних про планові відключення, що надходять у базу заздалегідь. У цьому режимі користувач бачить попередження з вказаними точними часом початку відключення (рисунок 5.8).

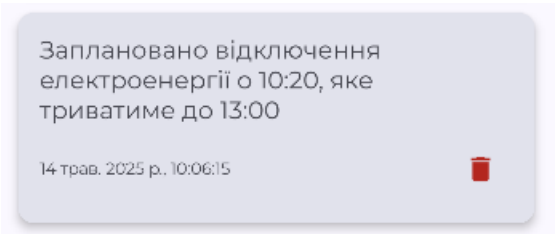

Рисунок 5.8 – Сповіщення про відключення електропостачання

Якщо лічильник має позначку про наявність сонячної панелі, система додатково рекомендує переключитися на альтернативне джерело живлення (рисунок 5.9).

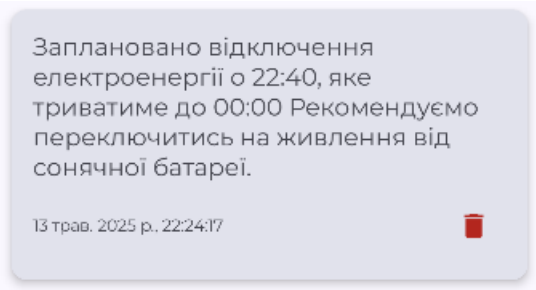

Рисунок 5.9 – Рекомендація переходу на інше джерело живлення

Таким чином, застосунок динамічно адаптується до змін умов та автоматично перемикається між режимами залежно від ситуації. Це забезпечує користувачеві актуальну інформацію, підвищує надійність системи, а також сприяє відповідальному ставленню до споживання енергії.

Висновки до розділу 5

У цьому розділі було описано логіку роботи мобільного застосунку в різних режимах функціонування, зумовлених як зовнішніми факторами (аварійні або планові відключення), так і внутрішніми показниками (рівень споживання, тарифна зона, наявність альтернативного живлення).

Розроблена система демонструє гнучку поведінку в умовах зміни ситуації, динамічно адаптується до показників споживання та часу доби, що дозволяє своєчасно інформувати користувача про важливі події. Застосунок забезпечує не лише доступ до поточних і прогнозованих даних, а й виступає як асистент енергоефективності, формуючи персоналізовані рекомендації залежно від поведінки користувача.

Режими роботи охоплюють:

- нормальне функціонування з повним доступом до даних та рекомендацій;

- автоматичні поради щодо перенесення завдань залежно від навантаження;
- реакцію на перевищення прогнозу;
- адаптацію до тарифних зон для зменшення витрат;
- аварійні та планові сценарії з відповідними повідомленнями й порадами щодо використання сонячної панелі.

Таким чином, застосунок виконує не лише роль моніторингового інструменту, а й стає активним учасником енергетичного менеджменту в домогосподарстві, сприяючи підвищенню обізнаності користувача, економії ресурсів і сталому використанню енергії.

6 РОБОТА КОРИСТУВАЧА З СИСТЕМОЮ

Інтерфейс користувача є ключовою частиною програми через який споживач взаємодіє з системою. При першому вході в систему користувач отримує вибір між авторизацією та реєстрацією. За умови вибору останнього на екрані виводиться сторінка з п’ятьма полями, які варто заповнити для створення облікового запису. На рисунку 6.1 зображено повну реалізацію сторінки «Реєстрації».

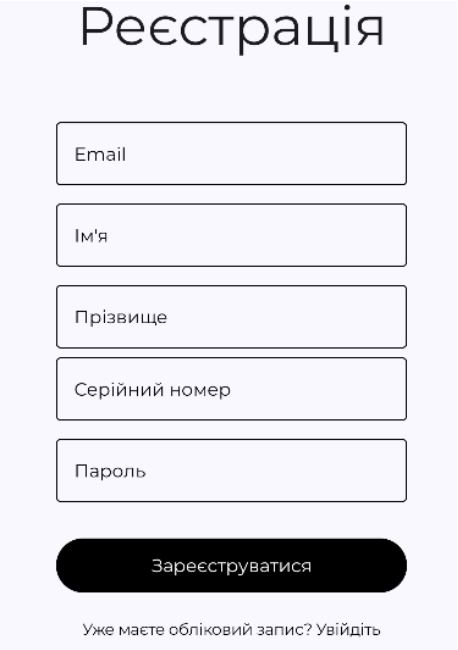

Рисунок 6.1 – Реєстрація користувача

Для реєстрації потрібні п’ять основних даних – пошта, ім’я, прізвище, серійний номер лічильника та пароль. Кнопка «Зареєструватися» створює новий обліковий запис користувача.

Під кнопкою є перехід на сторінку авторизацію. Її відображення на екрані зображено на рисунку 6.2.

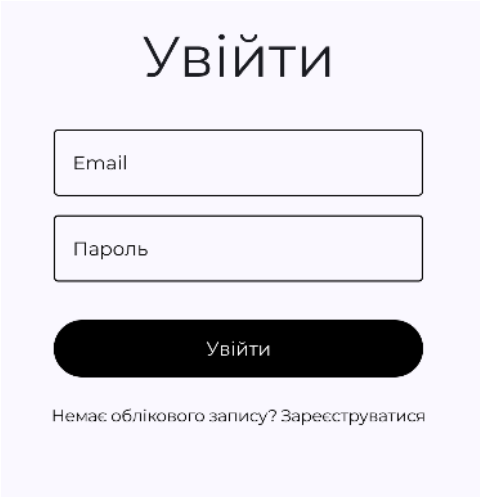

Рисунок 6.2 – Вхід в обліковий запис

Для входу в систему потрібно вести електронну пошту та пароль. Після натискання кнопки входу користувача одразу ж перекидає на головну сторінки. Сторінка, що відповідає за виведення даних лічильника в реальному часі. На рисунку 6.3 відображено екран головної сторінки. Одразу ж перед користувачем з’являється навігаційна панель для переходу між сторінками та шапка з серійним номером лічильника та адресою будинку в якому він встановлений.

Рисунок 6.3 – Головна сторінка

На головній сторінці виставлено значення лічильника, що оновлюється в реальному часі. Під ним виведено рекомендовані витрати на поточний місяць між споживачами одного будинку до якого прив’язаний лічильник. Також на сторінці виведена таблиця планових відключень на поточний день.

Дані про лічильник користувача також виводяться на сторінці облікового запису (рисунок 6.4) З цієї сторінки користувач може бачити свої дані та вийти з системи.

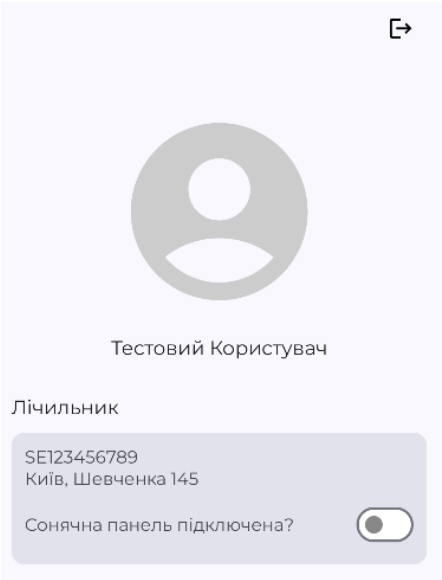

Рисунок 6.4 – Сторінка профілю

Однією зі сторінок мобільного застосунку є сторінка історії споживання (рисунок 6.5). На ній користувачу продемонстровані графіки та таблиці розрахованої аналітики за рік – згруповані дані за кожен місяць останнього року. За місяць – дані за кожен день за попередній місяць до сьогодні та за тиждень – виводить значення за останній тиждень завершуючи сьогоднішнім днем. Аби побачити дані за певною категорією, потрібно переключити на відповідну кнопку «Рік», «Місяць», «Тиждень».

Рисунок 6.5 – Переключання кнопок на сторінці історії споживання

Коли користувач натискає на навігаційній панелі на дзвіночок, то перед ним відкривається сторінка сповіщення (рисунок 6.6). Всі сповіщення відправлені системою відображаються у цьому блоці. Надається дозвіл видаляти ці повідомлення за потреби.

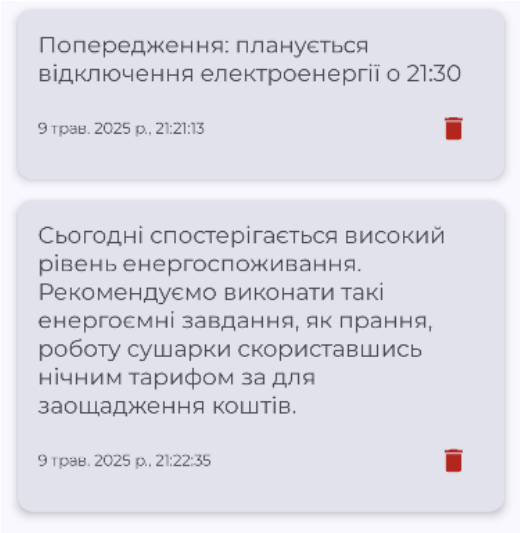

Рисунок 6.6 – Сторінка сповіщень

Перейшовши на сторінку «Графіки» користувач побачить перед собою спочатку графіки використання електроенергії за минулий та позаминулий період часу з вибором: тиждень, місяць, рік. На рисунку 6.7 зображені всі три графіки

порівняння використання енергії. Червоним зазначено дані про попередній термін, а чорним – поточний період часу.

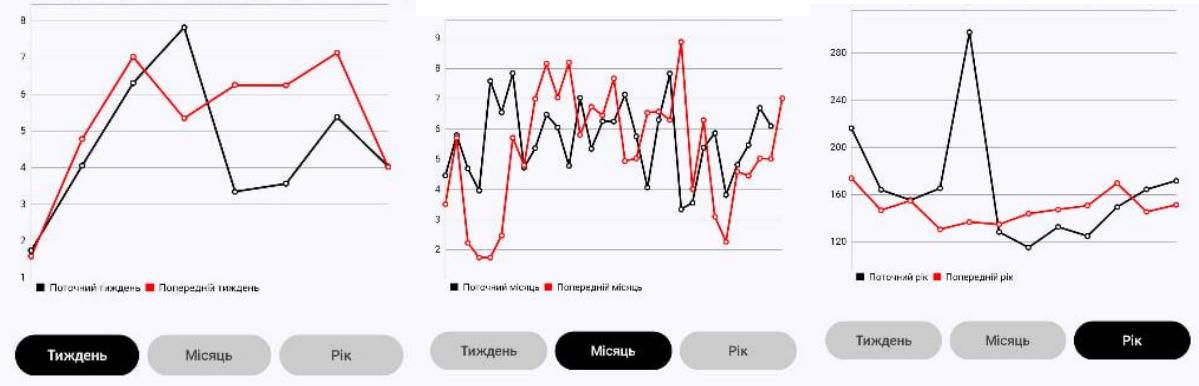

Рисунок 6.7 – Графіки порівняння використання електроенергії за тиждень, місяць та рік

На тій же сторінці показано графік споживання середньостатистичного споживання по годинам (рисунок 6.8). Крім того виводиться рекомендація, яка повідомляє користувача про найраціональніше використання електроенергії протягом дня. В годину з найвищими енерговитратами система рекомендує зменшити навантаження і перенести енергоємні завдання на годину з найнижчими витратами.

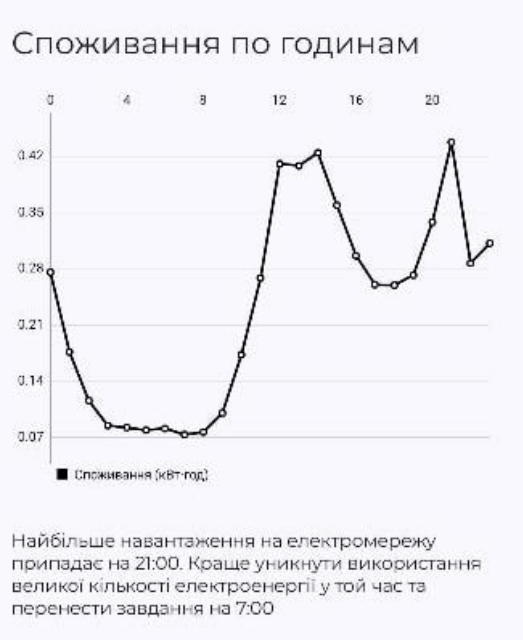

Рисунок 6.8 – Графік споживання по годинам

На рисунку 6.9 зображено такий самий графік, але з врахуванням днів тижня. Система знаходить день з найвищими витратами електроенергії та просить користувача зменшити навантаження на мережу в той день, перенісши енергоємні завдання на інший день тижня, визначеним, як найбільш енергоефективним.

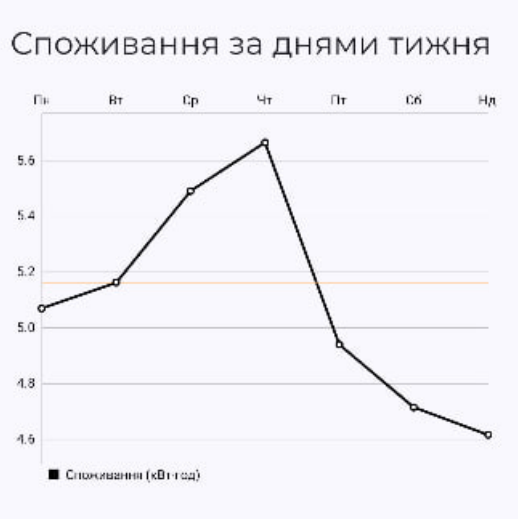

Рисунок 6.9 – Графік споживання по дням тижня

Цей інтерфейс є інтуїтивно зрозумілий для користувача та полегшує йому відслідковування електроспоживання по всьому будинку та його оптимальне використання.

Висновки до розділу 6

У цьому розділі було детально описано інтерфейс користувача та основні сценарії взаємодії з мобільним застосунком. Користувач має змогу швидко зареєструватися або увійти до системи, після чого отримує доступ до повного функціоналу програми: перегляду поточних показників енергоспоживання, аналізу історичних даних, читання рекомендацій та сповіщень, а також ознайомлення з плановими відключеннями.

Інтерфейс реалізовано на основі сучасного підходу Jetpack Compose, що дозволяє досягти високої адаптивності та зручності використання. Всі основні дії

користувача винесено в окремі логічні розділи: головна сторінка, профіль, історія, сповіщення, що забезпечує інтуїтивну навігацію.

Особливу увагу приділено реальному часу оновлення показників та персоналізованим даним користувача: відображення адреси, серійного номера лічильника, динаміки споживання та індивідуальних порад.

Таким чином, користувацький інтерфейс є не лише інформативним, а й інтерактивним інструментом, що дозволяє споживачеві активно керувати своїм енергоспоживанням, отримувати корисні рекомендації та приймати енергоефективні рішення в зручній формі.

ВИСНОВКИ

У процесі виконання проекту була здійснена комплексна розробка мобільного застосунку, призначеного для моніторингу та оптимізації споживання електроенергії індивідуальними користувачами. Було проведено детальний аналіз предметної області, а також вивчено існуючі рішення на ринку енергозбереження. На основі цього аналізу сформовано концепцію програмного продукту, який би поєднував простоту використання з потужними аналітичними можливостями. Було спроєктовано архітектуру клієнт-серверної системи з чітким розділенням логіки на фронтенд і бекенд, реалізовано серверну частину з використанням мови Python та фреймворку FastAPI, а також розроблено клієнтський інтерфейс на Kotlin з використанням Jetpack Compose.

Окрема увага приділялася реалізації функціоналу прогнозування енергоспоживання на основі історичних даних, що дозволило не лише інформувати користувача про поточний стан, а й передбачати майбутні витрати та надавати своєчасні поради. Завдяки впровадженню сучасної моделі машинного навчання LightGBM система здатна формувати точні й інформативні рекомендації. Програмний застосунок підтримує різні режими роботи — нормальний, аварійний, режим планових відключень та режим оптимального споживання згідно тарифів — кожен з яких має свою логіку обробки та взаємодії з користувачем.

У результаті було досягнуто поставленої мети: створено інноваційне рішення, яке забезпечує користувачеві не лише зручний інтерфейс, а й корисну аналітику для прийняття обґрунтованих рішень щодо енергоспоживання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Що таке MVC? Переваги та недоліки MVC. URL: https://alexhost.com/uk/faq/shho-take-mvc-perevagy-ta-nedoliky-mvc/ (дата звернення 12.05.2025)

2. Model-View-ViewModel (MVVM) URL: https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm (дата звернення 12.05.2025)

3. Що таке PostgreSQL і для чого використовується? URL: https://foxminded.ua/postgresql-shcho-tse/ (дата звернення 12.05.2025)

4. FastAPI – Uvicorn URL: https://www.geeksforgeeks.org/fastapi-uvicorn/ (дата звернення 12.05.2025)

5. Hastie, Tibshirani, Friedman, The Elements of Statistical Learning, 2009, 325 с.

6. MVC Framework – Introduction URL: https://www.tutorialspoint.com/mvc_framework/mvc_framework_introduction (дата звернення: 20.05.2025).

7. FastAPI – Introduction URL: https://www.tutorialspoint.com/fastapi/fastapi_introduction (дата звернення: 12.05.2025).

8. Kotlin – Overview URL: https://www.tutorialspoint.com/kotlin/kotlin_overview.htm (дата звернення: 12.05.2025).

ДОДАТОК А

Мобільний застосунок керування електроенегією споживача

Текст програми

Аркушів 6

Київ-2025

recommendation_repo.py

```
from datetime import datetime
from typing import List
from sqlalchemy.orm import Session
from app.utils.classifier import classify_levels
from app.repository import analytics_repo
from app.services.analytics_service import convert_to_chart
import random

postponable_tasks = [
    "прання",
    "зарядження електромобіля",
    "використання сушарки",
    "використання праски",
    "використання пилососа"
]

shiftable_today_tasks = [
    ["використання обігрівача",  "використання кондиціонера"],
    "користування бойлером",
    "зарядження телефону",
    "користування посудомийною машини"
]

def seasonal_first_task(tasks: list) -> list:
    month = datetime.now().month
    if month in [10, 11, 12, 1, 2, 3, 4]:
        first = tasks[0][0]
    else:
        first = tasks[0][1]
    return [first] + tasks[1:]

def get_current_tariff(hour: int) -> str:
    return "night" if hour < 7 or hour >= 23 else "day"
```

```
def generate_today_energy_recommendation(db: Session, meter_id: int) -> List[str]:
    now = datetime.now()
    today_dow = now.weekday()
    current_hour = now.hour
    shiftable_tasks = seasonal_first_task(shiftable_today_tasks)

    dow_data = convert_to_chart(analytics_repo.avg_by_day_of_week_last_year(db, meter_id))
    hour_data = convert_to_chart(analytics_repo.avg_usage_by_hour_last_month(db, meter_id))

    dow_levels = classify_levels(dow_data, 7)
    hour_levels = classify_levels(hour_data, 24)
    low_dow_keys = [k for k, v in dow_levels.items() if v in ["low", "too low"]]
    low_hour_keys = [k for k, v in hour_levels.items() if v in ["low", "too low"]]

    dow_names = ["понеділок", "вівторок", "середа", "четвер", "п’ятниця", "субота", "неділя"]

    dow_level = dow_levels.get(str(today_dow))
    hour_level = hour_levels.get(str(current_hour))

    random_dow = random.sample(low_dow_keys, k=2) if low_dow_keys else None
    random_hour = random.sample(low_hour_keys, k=2) if low_hour_keys else None
    random_postponable_tasks = random.sample(postponable_tasks, k=1)
    random_shiftable_today_tasks = random.sample(shiftable_tasks, k=1)

    messages = ""

    if dow_level in ["hight", "too hight"]:
        messages += (
            f"{dow_names[today_dow].capitalize()} — день з високим рівнем споживання. "
                        f"Рекомендується перенести {random_postponable_tasks[0]} на кращі дні:
{dow_names[int(random_dow[0])]} або {dow_names[int(random_dow[1])]}."
        )

    elif dow_level in ["too low"]:
```

```
        random_tasks = " і ".join(random.sample(postponable_tasks, k=2))
        messages += (
            f"Сьогодні — день з мінімальним навантаженням на мережу. "
            f"Рекомендуємо скористатися цим і виконати такі енергоємні завдання, як {random_tasks}."
        )

    elif dow_level == "average" and hour_level in ["hight", "too hight"]:
        messages += (
            f"В середньому о {current_hour}:00 в {dow_names[today_dow]} у вас великі енерговитрати."
             f"Краще уникати {random_shiftable_today_tasks[0]} у цей час. Рекомендуємо перенести енерговитратні завдання на {random_hour[0]}:00"
        )

    else:

        random_tasks = " чи ".join(random.sample(shiftable_tasks, k=2))
        current_tariff = get_current_tariff(current_hour)

        if current_tariff == "night":
            messages += (
                f"Сьогодні спостерігається високий рівень енергоспоживання, але діє нічний тариф. "
                f"Це ідеальний момент для виконання таких енергоємних завдань, як {random_tasks}, за зниженою ціною."
            )
        else:
            messages += (
                f"Сьогодні спостерігається високий рівень енергоспоживання. "
                f"Рекомендуємо виконати такі енергоємні завдання, як {random_tasks}, скориставшись нічним тарифом, задля заощадження коштів."
            )

    return messages
```

model_forecast.py

```
import pandas as pd
import numpy as np
```

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from lightgbm import LGBMRegressor
import joblib

df = pd.read_csv("./app/data/weather-energy-data-update.csv")
df['Datetime'] = pd.to_datetime(df['Datetime'])

df['month'] = df['Datetime'].dt.month
df['day'] = df['Datetime'].dt.day
df['hour'] = df['Datetime'].dt.hour
df['day_of_week'] = df['Datetime'].dt.dayofweek
df['is_weekend'] = df['day_of_week'] >= 5

df = df.set_index('Datetime').sort_index()
df['lag_30d'] = df['kWh'].shift(24 * 30)
df['rolling_7d'] = df['kWh'].rolling(window=24 * 7).mean()

features = ['month', 'day', 'hour', 'day_of_week', 'is_weekend', 'temp_dry', 'lag_30d', 'rolling_7d']
df = df.dropna(subset=features + ['kWh'])

X = df[features].copy()
X['Datetime'] = df.index
y = df['kWh']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

X_train_model = X_train.drop(columns='Datetime')
X_test_model = X_test.drop(columns='Datetime')

model = LGBMRegressor(random_state=42)
model.fit(X_train_model, y_train)

y_pred = model.predict(X_test_model)
```

```
rmse = np.sqrt(mean_squared_error(y_test, y_pred))

X_test_with_dt = X_test.copy()
X_test_with_dt['kWh_pred'] = y_pred
X_test_with_dt['date'] = X_test_with_dt['Datetime'].dt.date

daily_prediction = X_test_with_dt.groupby('date')['kWh_pred'].sum().reset_index()
daily_prediction.rename(columns={'kWh_pred': 'daily_kWh_pred'}, inplace=True)

actual = X_test_with_dt.copy()
actual['kWh'] = y_test.values
actual['date'] = actual['Datetime'].dt.date
daily_actual = actual.groupby('date')['kWh'].sum().reset_index()
daily_actual.rename(columns={'kWh': 'daily_kWh_actual'}, inplace=True)

daily_comparison = pd.merge(daily_prediction, daily_actual, on='date')
daily_comparison['daily_error'] = abs(daily_comparison['daily_kWh_actual'] - daily_comparison['daily_kWh_pred'])

daily_rmse = np.sqrt(mean_squared_error(
    daily_comparison['daily_kWh_actual'],
    daily_comparison['daily_kWh_pred']
))

joblib.dump(model, './app/models/lgbm_model.pkl')
```

prediction.py

```
import joblib
import pandas as pd
from datetime import datetime, timedelta
from calendar import monthrange

model = joblib.load('./app/models/lgbm_model.pkl')

def generate_future_features_to_month_end(start_date, tempreture, avg_month, avg_week):
    features = ['month', 'day', 'hour', 'day_of_week', 'is_weekend', 'lag_30d', 'rolling_7d']
```

```
year = start_date.year
month = start_date.month
last_day = monthrange(year, month)[1]
end_date = datetime(year, month, last_day, 23)
hours = int((end_date - start_date).total_seconds() // 3600) + 1

datetimes = [start_date + timedelta(hours=i) for i in range(hours)]
df = pd.DataFrame({'Datetime': datetimes})
df['month'] = df['Datetime'].dt.month
df['day'] = df['Datetime'].dt.day
df['hour'] = df['Datetime'].dt.hour
df['day_of_week'] = df['Datetime'].dt.dayofweek
df['is_weekend'] = (df['day_of_week'] >= 5).astype(int)

df['lag_30d'] = avg_month
df['rolling_7d'] = avg_week

df['kWh_pred'] = model.predict(df[features])
df['date'] = df['Datetime'].dt.date
future_daily = df.groupby('date')['kWh_pred'].sum().reset_index()
return future_daily
```