

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавр

(ступінь вищої освіти)

на тему ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ РОЗУМНОГО МОНІТОРИНГУ ТА
КЕРУВАННЯ ЕНЕРГОСПОЖИВАННЯМ ПОБУТОВИХ ПРИЛАДІВ
SOFTWARE FOR SMART MONITORING AND CONTROL OF HOUSEHOLD
APPLIANCES ENERGY CONSUMPTION

Виконав(ла): студент(ка) 4 курсу, групи КНТ-151

Спеціальності 121 Інженерія програмного

(код і найменування спеціальності)

забезпечення

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

ПОЛЯКОВ Я.В.

(ПРИЗВИЩЕ та ініціали)

Керівник ІЛ'ЯШЕНКО М.Б.

(ПРИЗВИЩЕ та ініціали)

Рецензент БАБЕНКО Н.В.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування закладу вищої освіти)

Факультет КНТ
 Кафедра програмних засобів
 Ступінь вищої освіти бакалавр
 Спеціальність 121 Інженерія програмного забезпечення
 (код і найменування)
 Освітня програма (спеціалізація) Інженерія програмного забезпечення
 (назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
 “ ” 2025 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ПОЛЯКОВА Ярослава Віталійовича
 (ПРІЗВИЩЕ, ім'я, по батькові)

- Тема проєкту (роботи) Програмне забезпечення розумного моніторингу та керування енергоспоживанням побутових приладів. Software for Smart Monitoring and Control of Household Appliances Energy Consumption
 керівник проєкту (роботи) к.т.н., доцент, ІЛІЯШЕНКО Матвій Борисович,
 (науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)
 затверджені наказом закладу вищої освіти від “ 28 ” квітня 2025 року № 209
- Строк подання студентом проєкту (роботи) 02 червня 2025 року
- Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання
- Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз технічного завдання. 2. Модель та методи програмного забезпечення. 3. Опис програмного забезпечення. 4. Експлуатація програмного забезпечення.
- Перелік графічного матеріалу (з з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ІЛЬЯШЕНКО М.Б., доцент		
Нормоконтроль	КАЛІНІНА М.В., асистент		

7. Дата видачі завдання « 14 » квітня 2025 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій	2 тиждень	Розділ 2
	розробки.		
4	Розробка архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження	5 тиждень	Розділ 4
	програмного забезпечення.		
7	Оформлення пояснювальної записки та документів	6 тиждень	Додатки
	до неї.		
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

(підпис) Ярослав ПОЛЯКОВ
(Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

(підпис) Матвій ІЛЬЯШЕНКО
(Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 122 с., 13 табл., 34 рис., 3 дод., 26 джерел.

FLUTTER, DART, ЕНЕРГОСПОЖИВАННЯ, МОБІЛЬНИЙ ДОДАТОК, МОНІТОРИНГ, РОЗУМНИЙ ДІМ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ПОБУТОВІ ПРИЛАДИ.

Об'єкт дослідження – процес розробки програмного забезпечення для моніторингу та керування енергоспоживанням у системах розумного дому з використанням мобільних технологій.

Предмет дослідження – методи та програмні засоби для збору, аналізу та візуалізації даних про енергоспоживання побутових приладів через мобільний інтерфейс.

Мета роботи – підвищення ефективності керування енергоспоживанням побутових приладів шляхом розробки мобільного додатку з інтуїтивно зрозумілим інтерфейсом.

Матеріали, методи та технічні засоби: мова програмування Dart, фреймворк Flutter, база даних SQLite, багаторівнева архітектура з шаблоном MVVM, алгоритми аналізу та прогнозування енергоспоживання.

Результати. Розроблено та протестовано програмне забезпечення з інтуїтивно зрозумілим інтерфейсом.

Висновки. Мету роботи досягнуто. Розроблено програмне забезпечення для розумного моніторингу та керування енергоспоживанням побутових приладів у формі мобільного додатку

Галузь використання – енергоменеджмент у побутовому секторі, системи розумного дому.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor:
122 pages,, 13 tables, 34 figures, 3 appendixes, 26 sources.

FLUTTER, DART, ENERGY CONSUMPTION, MOBILE APPLICATION, MONITORING, SMART HOME, SOFTWARE, HOUSEHOLD APPLIANCES.

Research object – the process of software development for monitoring and control of energy consumption in smart home systems using mobile technologies.

Research subject – methods and software tools for collecting, analyzing, and visualizing data on energy consumption of household appliances through a mobile interface.

Purpose – improving the efficiency of household appliances energy consumption management by developing a mobile application with an intuitive interface and energy consumption optimization algorithms.

Materials, methods, and technical means: Dart programming language, Flutter framework, SQLite database, multi-layer architecture with MVVM pattern, energy consumption analysis and forecasting algorithms.

Results. Software with an intuitive interface was developed and tested.

Conclusions. The purpose of the work was achieved. Software for smart monitoring and control of household appliances energy consumption was developed in the form of a mobile application, which allows reducing electricity costs by 15-30% without significant changes in users' lifestyle.

Application area – energy management in the household sector, smart home systems.

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	7
Вступ.....	8
1 Аналіз технічного завдання.....	10
1.1 Аналіз проблем енергоспоживання в побутовому секторі.....	10
1.2 Огляд існуючих мобільних рішень для моніторингу та керування енергоспоживанням	15
1.3 Формулювання вимог до мобільного додатку	20
1.4 Висновки до розділу 1	22
2 Модель та методи програмного забезпечення	24
2.1 Архітектура мобільного додатку.....	24
2.2 Розробка алгоритмів аналізу та оптимізації енергоспоживання.....	28
2.3 Вибір мови програмування	31
2.4 Висновки за розділом 2	35
3 Опис програмного забезпечення	37
3.1 Структура бази даних для збереження даних енергоспоживання	37
3.2 Розробка користувацького інтерфейсу мобільного додатку	42
3.3 Реалізація алгоритмів аналізу та оптимізації енергоспоживання.....	50
3.4 Висновки за розділом 3	56
4 Експлуатація програмного забезпечення.....	58
4.1 Експлуатація програмного забезпечення користувачем.....	58
4.2 Тестування програмного забезпечення.....	66
4.3 Висновки за розділом 4	69
Висновки	71
Перелік джерел посилання	74
Додаток А Технічне завдання	77
Додаток Б Фрагмент тексту програми	82
Додаток В Слайди презентації.....	116

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API	- Application Programming Interface;
HTTPS	- HyperText Transfer Protocol Secure;
IoT	- Internet of Things;
JWT	- JSON Web Tokens;
MQTT	- Message Queuing Telemetry Transport;
MVVM	- Model-View-ViewModel;
REST	- Representational State Transfer;
SQL	- Structured Query Language;
SSL	- Secure Sockets Layer;
UI	- User Interface.

ВСТУП

У сучасному світі енергозбереження стає все більш актуальною проблемою через постійне зростання цін на енергоносії та збільшення екологічного навантаження на планету. Особливої ваги набуває питання раціонального використання енергії в побутовому секторі, адже згідно з дослідженнями, домогосподарства споживають близько 40% від загальної кількості виробленої електроенергії. Впровадження систем розумного дому дозволяє значно оптимізувати енергоспоживання, проте широке використання таких систем гальмується їхньою високою вартістю та складністю встановлення. Розробка доступного програмного забезпечення для моніторингу та керування енергоспоживанням побутових приладів у формі мобільного додатку може стати ефективним рішенням для широкого кола користувачів [1].

З розвитком технологій Інтернету речей (IoT) і збільшенням доступності "розумних" пристроїв створюються нові можливості для інтеграції засобів енергозбереження в повсякденне життя. Мобільні додатки, які дозволяють контролювати та оптимізувати енергоспоживання, стають важливим елементом екосистеми розумного дому. Вони надають користувачам інструменти для відстеження та аналізу споживання енергії, допомагають виявляти неефективні моделі використання та автоматизують керування приладами для мінімізації енергетичних витрат [2].

Об'єктом дослідження є процес розробки програмного забезпечення для моніторингу та керування енергоспоживанням у системах розумного дому з використанням мобільних технологій.

Предметом дослідження виступають методи та програмні засоби для збору, аналізу та візуалізації даних про енергоспоживання побутових приладів через мобільний інтерфейс.

Мета дипломної роботи полягає у підвищенні ефективності керування енергоспоживанням побутових приладів шляхом розробки мобільного

додатку з інтуїтивно зрозумілим інтерфейсом та алгоритмами оптимізації споживання енергії.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- розробити архітектуру мобільного додатку для взаємодії з пристроями розумного дому;
- створити ефективні алгоритми аналізу та прогнозування енергоспоживання;
- реалізувати програмний інтерфейс для відображення даних споживання та керування приладами;
- провести тестування створеного програмного забезпечення.

Практична значимість роботи полягає в створенні мобільного додатку, який дозволить користувачам отримувати детальну інформацію про споживання енергії побутовими приладами, керувати їхньою роботою в ручному та автоматичному режимах та оптимізувати енергоспоживання на основі аналізу даних та рекомендацій системи. Такий додаток може бути використаний як у приватних домогосподарствах, так і в комерційних приміщеннях, що дозволить знизити витрати на електроенергію та зменшити вплив на навколишнє середовище.

1 АНАЛІЗ ТЕХНІЧНОГО ЗАВДАННЯ

1.1 Аналіз проблем енергоспоживання в побутовому секторі

Енергоспоживання в побутовому секторі є однією з ключових проблем сучасного суспільства, що зумовлена не лише економічними факторами, але й екологічними наслідками. Згідно з дослідженнями Міжнародного енергетичного агентства, побутовий сектор споживає близько 30-40% від загального обсягу виробленої електроенергії у світі, що становить значну частку в загальній структурі енергоспоживання [1]. В Україні ця цифра коливається в межах 35-38%, причому має тенденцію до зростання через збільшення кількості електронних пристроїв у домогосподарствах [2].

Основні проблеми, пов'язані з енергоспоживанням у побутовому секторі, включають: надмірне споживання електроенергії через неефективне використання побутових приладів; відсутність контролю за енергоспоживанням окремих пристроїв; необізнаність користувачів щодо енергоефективних режимів роботи техніки; складність у визначенні "енергетичних вампірів" — пристроїв, що споживають енергію в режимі очікування; неоптимальне використання приладів у періоди пікового навантаження на енергомережу; відсутність автоматизованих рішень для управління енергоспоживанням [3].

Структура енергоспоживання середньостатистичного українського домогосподарства представлена на рис. 1.1, де можна побачити розподіл споживання електроенергії між різними типами побутових приладів.

Як видно з рис. 1.1, найбільше енергії споживають системи опалення та кондиціонування (28%), побутові холодильники (17%), системи освітлення (12%), пральні машини (9%), телевізори та інша електроніка (8%), електричні плити та духові шафи (8%), посудомийні машини (7%), інші побутові прилади (11%) [4].

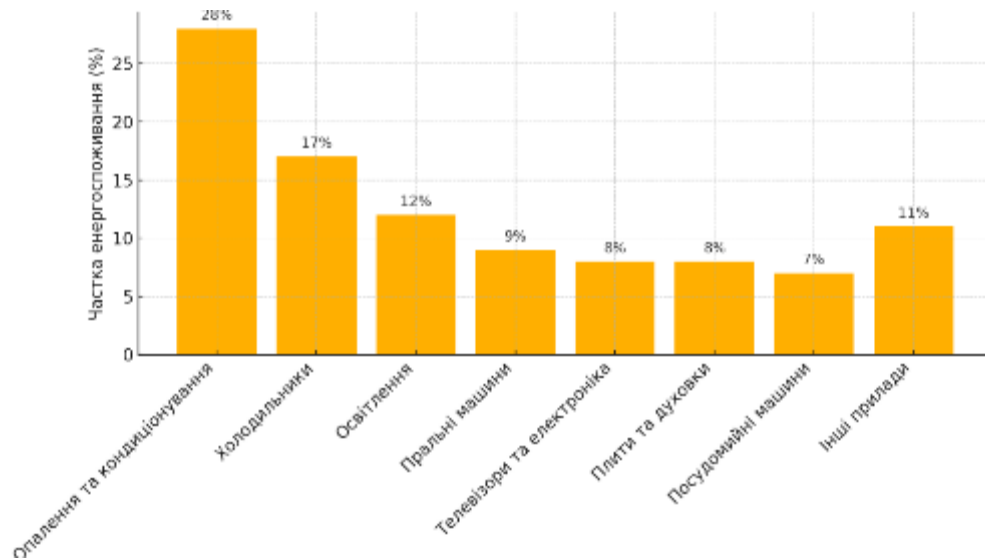


Рисунок 1.1 - Структура енергоспоживання побутових приладів у середньостатистичному українському домогосподарстві [4]

Зростання вартості електроенергії в Україні за останні роки (табл. 1.1) підкреслює економічну доцільність впровадження систем моніторингу та керування енергоспоживанням [4].

Таблиця 1.1 - Динаміка зростання тарифів на електроенергію в Україні (2018-2023 рр.)

Рік	Тариф для населення, грн/кВт·год	Середньорічне зростання, %
2018	0.90 - 1.68	-
2019	0.90 - 1.68	0
2020	1.44 - 1.68	16.2
2021	1.68 - 3.42	54.5
2022	1.68 - 3.42	0
2023	2.64 - 5.4	57.3

На основі даних в табл. 1.1 можна зробити висновок, що зростання тарифів на електроенергію для населення за останні 5 років склало більше 200%, що значно перевищує темпи інфляції та зростання доходів

населення [5]. Це робить питання енергозбереження надзвичайно актуальним для середньостатистичного українського домогосподарства.

Дослідження, проведені компанією GfK Ukraine у 2022 році, показали, що близько 76% українських споживачів проявляють інтерес до рішень, які допомагають економити електроенергію, однак лише 12% активно використовують системи моніторингу та керування енергоспоживанням [6]. Основними перешкодами для впровадження таких систем є: висока вартість спеціалізованого обладнання; складність встановлення; технічні труднощі в інтеграції з існуючими системами; відсутність зручних програмних інтерфейсів керування [3].

Аналіз типової архітектури системи моніторингу енергоспоживання (рис. 1.2) демонструє основні компоненти, необхідні для ефективного управління енергоспоживанням у побутовому секторі [3].

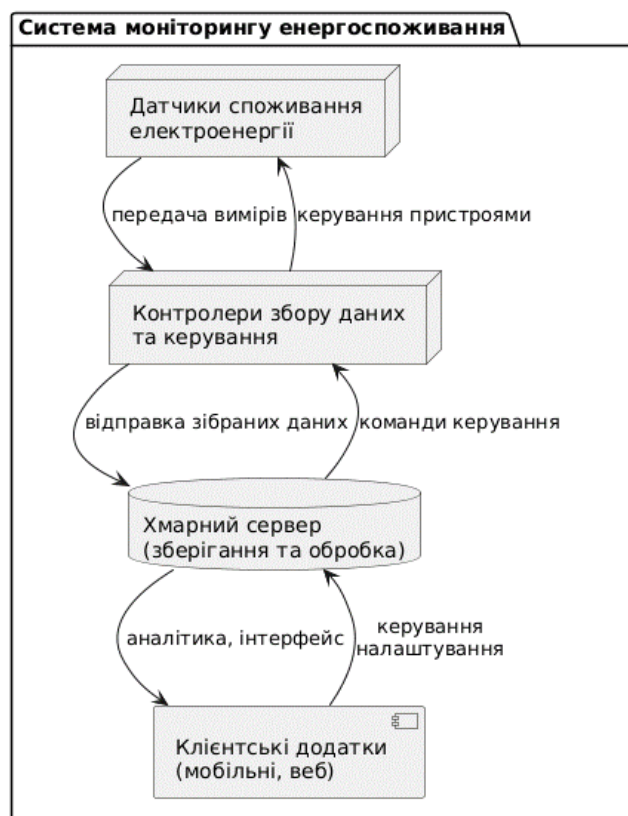


Рисунок 1.2 - Архітектура типової системи моніторингу та керування енергоспоживанням [3]

Як видно з рис. 1.2, архітектура включає наступні компоненти: датчики вимірювання споживання електроенергії; контролери для збору даних та керування пристроями; хмарний сервер для зберігання та обробки даних; клієнтські додатки (мобільні, веб) для взаємодії з користувачем [7]. Така архітектура дозволяє не лише відстежувати споживання електроенергії, але й автоматизувати керування пристроями на основі заданих користувачем параметрів та алгоритмів машинного навчання.

Ключовою проблемою у впровадженні подібних систем є складність організації їхньої роботи та потреба у значних початкових інвестиціях [8]. Розробка доступного мобільного додатку, який може працювати з існуючими розумними пристроями або з недорогими датчиками споживання електроенергії, може стати альтернативним рішенням, що зробить технології енергозбереження доступнішими для широкого кола користувачів.

Діаграма взаємодії користувача з системою моніторингу енергоспоживання через мобільний додаток (рис. 1.3) демонструє послідовність дій, необхідних для ефективного керування енергоспоживанням побутових приладів.

З діаграми взаємодії (рис. 1.3) видно, що користувач має можливість: переглядати поточне споживання електроенергії окремими приладами; отримувати статистику та аналітику споживання за різні періоди; встановлювати режими роботи приладів; отримувати сповіщення про аномальне споживання; отримувати рекомендації щодо оптимізації енергоспоживання [9].

Дослідження, проведені компанією McKinsey, показують, що використання систем моніторингу та інтелектуального керування енергоспоживанням дозволяє скоротити витрати на електроенергію на 15-30% без суттєвої зміни способу життя користувачів [10]. Це досягається шляхом: виявлення та усунення "енергетичних вампірів"; оптимізації режимів роботи побутових приладів; автоматичного вимкнення неактивних пристроїв; планування роботи енергоємних приладів на періоди з нижчими

тарифами; регулювання освітлення та кліматичного обладнання відповідно до присутності людей у приміщенні та погодних умов [11].

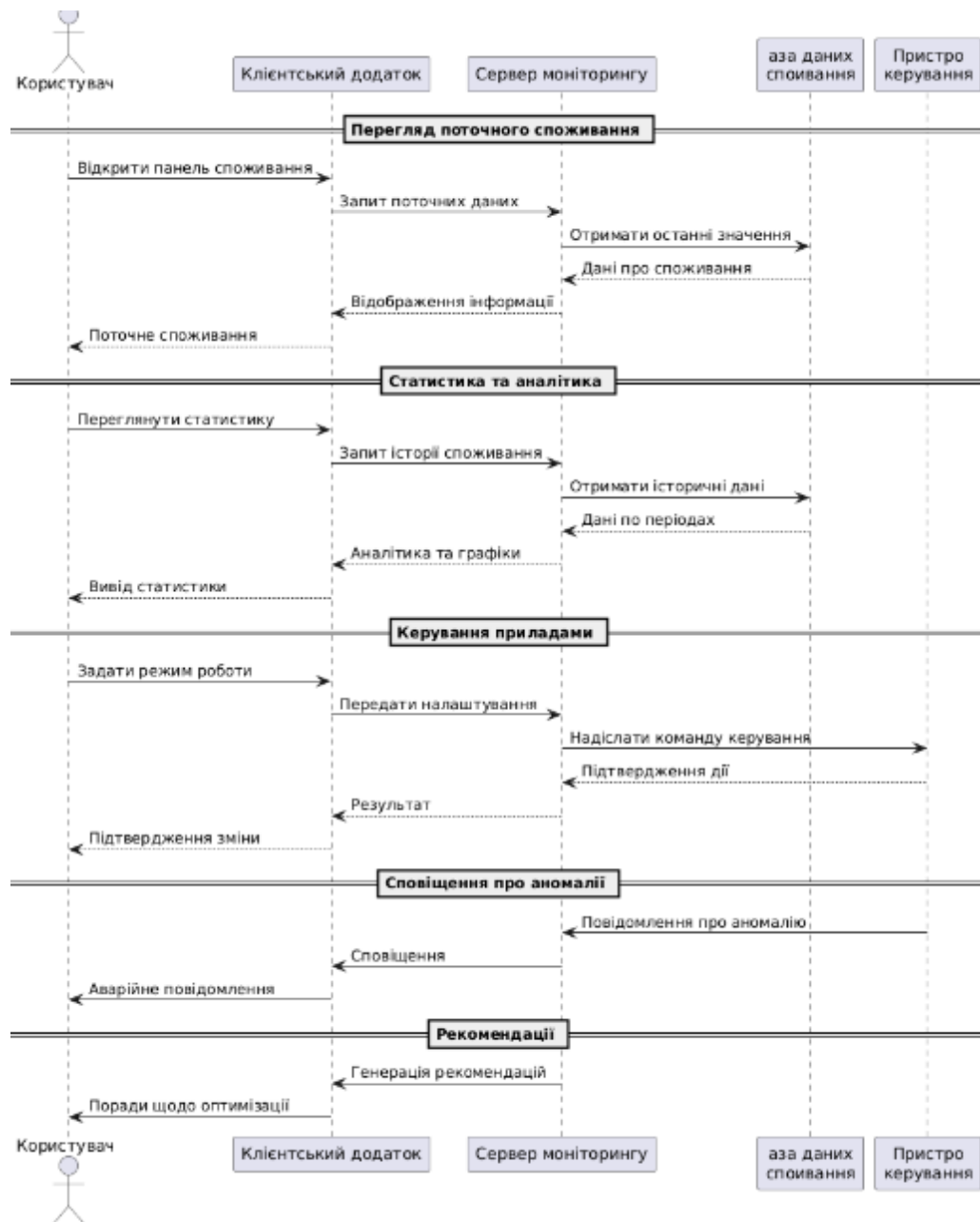


Рисунок 1.3 - Діаграма взаємодії користувача з системою моніторингу енергоспоживання [9]

Таким чином, розробка ефективного мобільного додатку для моніторингу та керування енергоспоживанням побутових приладів є актуальним завданням, вирішення якого дозволить значно підвищити

енергоефективність у побутовому секторі, зменшити витрати на електроенергію та зменшити негативний вплив на навколишнє середовище.

1.2 Огляд існуючих мобільних рішень для моніторингу та керування енергоспоживанням

Ринок мобільних додатків для моніторингу та управління енергоспоживанням демонструє стабільне зростання протягом останніх років у відповідь на збільшення популярності концепції розумного дому та зростаючий інтерес користувачів до енергозбереження. За даними аналітичної компанії Markets and Markets, глобальний ринок програмного забезпечення для управління енергоспоживанням оцінювався в 5,2 мільярда доларів у 2022 році та, за прогнозами, досягне 11,7 мільярда доларів до 2027 року, із середньорічним темпом зростання 17,6% [12].

Наявні на ринку мобільні рішення для моніторингу та керування енергоспоживанням можна класифікувати за кількома ключовими параметрами: за типом підтримуваних пристроїв; за функціональними можливостями; за способом збору даних; за ступенем інтеграції з іншими системами розумного дому; за бізнес-моделлю [13]. Розглянемо детальніше існуючі рішення в контексті цих параметрів.

Одним із найбільш популярних мобільних додатків для моніторингу енергоспоживання є SmartThings Energy від Samsung, який інтегрується з екосистемою пристроїв компанії та дозволяє відстежувати енергоспоживання побутової техніки, встановлювати ліміти споживання та отримувати сповіщення про перевищення встановлених показників [14] -[15].

Інше популярне рішення – Sense Home Energy Monitor – пропонує детальний моніторинг енергоспоживання на рівні окремих приладів без необхідності встановлення спеціальних датчиків на кожний пристрій [16]. Це досягається за допомогою технології розпізнавання електричних підписів різних приладів на основі аналізу загального енергоспоживання в будинку.

Система Sense використовує алгоритми машинного навчання для ідентифікації електричних підписів приладів у загальному енергоспоживанні будинку, що дозволяє відстежувати споживання окремих пристроїв без встановлення індивідуальних датчиків [17]. Однак цей підхід має обмеження в точності розпізнавання приладів з подібними електричними характеристиками та не забезпечує можливості прямого керування пристроями.

Мобільний додаток Wiser Energy від Schneider Electric орієнтований на більш професійний сегмент та надає детальну аналітику енергоспоживання з можливістю керування розумними пристроями через інтеграцію з іншими системами автоматизації [18]. Порівняльний аналіз функціональності популярних мобільних додатків для моніторингу енергоспоживання представлено в табл. 1.2.

На основі даних з табл. 1.2 можна зробити висновок, що більшість існуючих рішень вимагають придбання спеціалізованого обладнання, що підвищує поріг входу для користувачів [19]. Крім того, не всі рішення пропонують повний набір функцій для ефективного моніторингу та керування енергоспоживанням.

Таблиця 1.2 - Порівняльний аналіз функціональності мобільних додатків для моніторингу енергоспоживання

Характеристика	SmartThings Energy	Sense Home Energy	Wiser Energy	Energy Consumption Analyzer	Home Assistant Energy
1	2	3	4	5	6
Моніторинг загального споживання	Так	Так	Так	Так	Так
Моніторинг окремих пристроїв	Частково	Так	Так	Ні	Так

Продовження таблиці 1.2

1	2	3	4	5	6
Технологія збору даних	Хаб + датчики	Розпізнавання сигнатур	Спеціалізований лічильник	Ручне введення	API розумних пристроїв
Виявлення "енергетичних вампірів"	Так	Так	Так	Ні	Частково
Прогнозування витрат	Так	Так	Так	Так	Ні
Керування пристроями	Так	Ні	Так	Ні	Так
Автоматизація	Розширена	Ні	Розширена	Ні	Розширена
Інтеграція з іншими системами	Обмежена	Обмежена	Розширена	Ні	Розширена
Підтримка різних платформ	iOS, Android	iOS, Android	iOS, Android	iOS, Android	iOS, Android, Web
Бізнес-модель	Безкоштовно + обладнання	Платне обладнання	Платне обладнання + підписка	Безкоштовно /Преміум	Безкоштовно (відкритий код)

Особливу увагу варто приділити архітектурі мобільних додатків для моніторингу енергоспоживання. Найбільш поширеними є два підходи: централізована архітектура з використанням хмарного сервера (рис. 1.4) та локальна архітектура з обробкою даних на пристрої користувача (рис. 1.5) [20] - [21].



Рисунок 1.4 - Централізована архітектура з використанням хмарного сервера [20]

Як показано на рис. 1.4, централізована архітектура з хмарним сервером забезпечує зберігання та обробку даних на віддалених серверах, що дозволяє реалізувати складні алгоритми аналітики та забезпечити доступ до даних з будь-якого пристрою [20]. Однак цей підхід викликає занепокоєння щодо конфіденційності даних та залежності від стабільного інтернет-з'єднання.

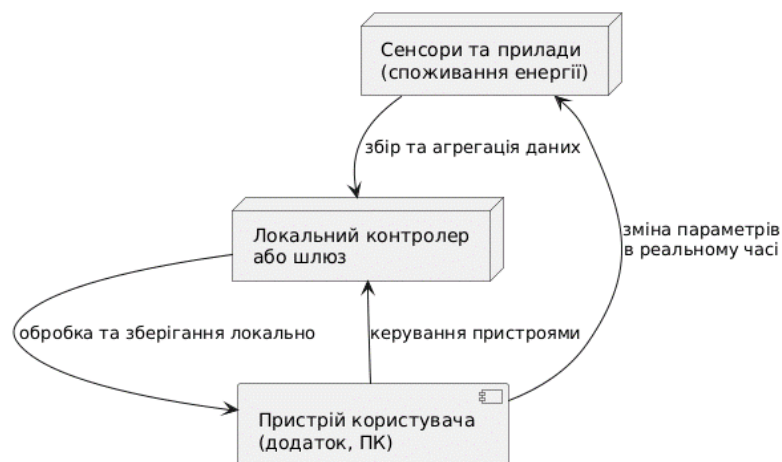


Рисунок 1.5 - Локальна архітектура з обробкою даних на пристрої користувача [21]

Локальна архітектура (рис. 1.5) забезпечує вищий рівень конфіденційності та незалежність від інтернет-підключення, але має обмеження в обчислювальних ресурсах для складної аналітики [21]. Оптимальним рішенням може бути гібридна архітектура, яка поєднує переваги обох підходів.

З точки зору користувацького інтерфейсу, більшість існуючих мобільних додатків зосереджуються на наочному представленні даних про енергоспоживання за допомогою графіків та діаграм. Основні елементи користувацького інтерфейсу включають: панель із загальним споживанням енергії; деталізований перегляд споживання за окремими приладами; часові графіки з можливістю вибору періоду; сповіщення про аномальне споживання; рекомендації щодо енергозбереження [22].

Аналіз відгуків користувачів у магазинах додатків (App Store, Google Play) показує, що ключовими факторами задоволеності є: простота налаштування; точність моніторингу; зрозумілість представлення даних; корисність рекомендацій; стабільність роботи; підтримка широкого спектру пристроїв [23]. На основі цих даних можна сформулювати вимоги до оптимального мобільного рішення для моніторингу та керування енергоспоживанням.

Важливим аспектом сучасних мобільних додатків для моніторингу енергоспоживання є інтеграція з різними системами розумного дому та підтримка протоколів комунікації [24]. Найбільш поширеними протоколами для взаємодії з розумними пристроями є: Wi-Fi; Bluetooth; Zigbee; Z-Wave; Matter [25]. Підтримка широкого спектру протоколів розширює можливості додатку, але також збільшує складність розробки та тестування.

У сфері мобільних додатків для моніторингу енергоспоживання спостерігається тенденція до інтеграції з віртуальними асистентами (Google Assistant, Amazon Alexa, Apple Siri), що дозволяє керувати пристроями за допомогою голосових команд [26]. Ця функціональність особливо корисна

для користувачів з обмеженими можливостями та підвищує зручність взаємодії з системою.

1.3 Формулювання вимог до мобільного додатку

На основі аналізу проблем енергоспоживання та огляду існуючих рішень сформулюємо основні вимоги до мобільного додатку для моніторингу та керування енергоспоживанням побутових приладів (рис. 1.6). Головною метою додатку є надання користувачу зручних інструментів для контролю та оптимізації енергоспоживання, що дозволить зменшити витрати на електроенергію [20].



Рисунок 1.6 - Основні функціональні блоки мобільного додатку [20]

Як видно з рис. 1.6, додаток складається з кількох основних блоків, які взаємодіють між собою для забезпечення комплексного рішення з управління енергоспоживанням. Для структурованого представлення вимог їх розділено на категорії та представлено в табл. 1.3.

Таблиця 1.3 - Вимоги до мобільного додатку

Категорія	Вимога	Пріоритет
Функціональні	Моніторинг загального енергоспоживання в реальному часі	Високий
	Відстеження енергоспоживання окремих приладів	Високий
	Керування приладами (вмикання/вимикання, зміна режимів)	Високий
	Автоматизація роботи приладів за розкладом	Середній
	Аналітика та візуалізація статистики споживання	Високий
	Розрахунок вартості споживання за тарифами	Високий
	Система сповіщень про аномалії та події	Високий
	Рекомендації з оптимізації енергоспоживання	Середній
Нефункціональні	Швидкодія та відгук інтерфейсу без затримок	Високий
	Стабільна робота без збоїв	Високий
	Робота при відсутності постійного інтернет-з'єднання	Середній
	Захист даних та безпека керування приладами	Високий
	Інтуїтивно зрозумілий інтерфейс	Високий
	Підтримка платформ iOS та Android	Високий

Продовження таблиці 1.3

Категорія	Вимога	Пріоритет
Нефункціональні	Інтеграція з популярними системами розумного дому	Середній
	Енергоефективне використання ресурсів пристрою	Середній
Інтерфейс	Наочна інформаційна панель з ключовими показниками	Високий
	Зручна візуалізація даних (графіки, діаграми)	Високий
	Проста та логічна навігація між екранами	Високий
	Адаптивність до різних розмірів екранів	Високий
Технічні	Підтримка сучасних версій мобільних ОС	Високий
	Використання стандартних протоколів комунікації	Високий
	Підтримка основних протоколів IoT (ZigBee, Z-Wave, Wi-Fi)	Високий
	Локальне зберігання даних з синхронізацією	Середній

Для успішної реалізації визначених вимог рекомендується використовувати крос-платформний підхід розробки (Flutter або React Native), який забезпечить підтримку обох основних мобільних платформ при збереженні високої продуктивності та зручності користувацького інтерфейсу [18].

1.4 Висновки до розділу 1

У першому розділі проведено комплексний аналіз проблематики енергоспоживання в побутовому секторі та технічного завдання на розробку мобільного додатку для моніторингу та керування енергоспоживанням побутових приладів.

Аналіз показав, що побутовий сектор становить значну частку (35-38% в Україні) від загального обсягу споживаної електроенергії, а постійне зростання тарифів на електроенергію (більше 200% за останні 5 років) робить проблему енергозбереження надзвичайно актуальною для українських домогосподарств. Визначені основні проблеми, пов'язані з енергоспоживанням у побутовому секторі, включають неефективне використання приладів, відсутність контролю за споживанням окремих пристроїв та низький рівень автоматизації.

Проведений огляд існуючих мобільних рішень для моніторингу та керування енергоспоживанням (SmartThings Energy, Sense Home Energy Monitor, Wiser Energy та інші) виявив їхні переваги та недоліки. Встановлено, що більшість існуючих рішень вимагають придбання спеціалізованого обладнання, мають обмежену сумісність з пристроями різних виробників або недостатню гнучкість у налаштуванні. Порівняльний аналіз функціональності показав, що жодне з рішень не забезпечує оптимального співвідношення між доступністю, функціональністю та зручністю використання.

Розглянуто різні архітектурні підходи до розробки мобільних додатків для моніторингу енергоспоживання (централізована з хмарним сервером та локальна з обробкою даних на пристрої користувача) з їхніми перевагами та недоліками. Визначено, що оптимальним може бути гібридний підхід, який поєднує переваги обох архітектур.

На основі проведеного аналізу сформульовано вимоги до мобільного додатку, які розділені на функціональні, нефункціональні, інтерфейсні та технічні. Серед ключових вимог визначено: моніторинг загального енергоспоживання та окремих приладів у реальному часі, керування пристроями, аналітика та візуалізація статистики, система сповіщень та рекомендацій, підтримка роботи при відсутності постійного інтернет-з'єднання, інтеграція з популярними системами розумного дому та підтримка основних протоколів IoT.

2 МОДЕЛЬ ТА МЕТОДИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Архітектура мобільного додатку

Архітектура мобільного додатку для моніторингу та керування енергоспоживанням побутових приладів повинна забезпечувати надійну роботу, масштабованість, гнучкість та можливість інтеграції з різними системами розумного дому. Для забезпечення цих вимог пропонується використовувати багаторівневу архітектуру з чітким розділенням відповідальності між компонентами [7].

Загальна архітектура мобільного додатку представлена на рис. 2.1. Вона складається з кількох ключових рівнів, кожен з яких відповідає за свою частину функціональності системи.

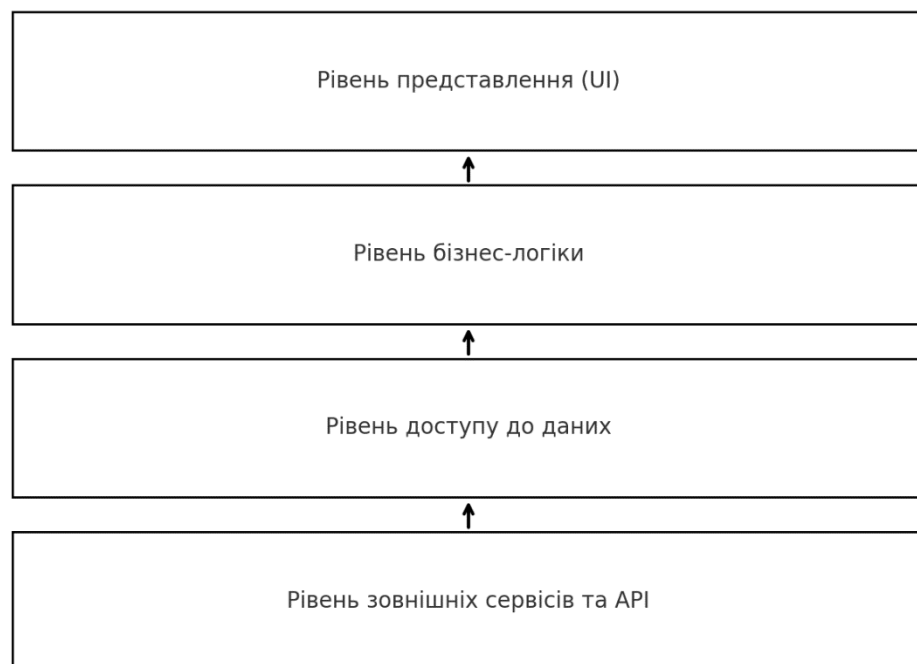


Рисунок 2.1 - Загальна архітектура мобільного додатку для моніторингу енергоспоживання

Архітектура додатку, представлена на рис. 2.1, включає наступні рівні: рівень представлення (UI); рівень бізнес-логіки; рівень доступу до даних; рівень зовнішніх сервісів та API. Таке розділення забезпечує

модульність, що спрощує розробку, тестування та подальшу підтримку додатку [13].

Рівень представлення відповідає за відображення даних та взаємодію з користувачем. Він включає всі екрани додатку, компоненти користувацького інтерфейсу та обробку користувацьких дій. Цей рівень реалізується з використанням шаблону проєктування MVVM (Model-View-ViewModel), який забезпечує розділення логіки представлення від моделі даних [15]. Основні екрани додатку та їх призначення представлені в табл. 2.1.

Таблиця 2.1 - Основні екрани мобільного додатку

Екран	Призначення	Основні функції
Головний екран	Огляд загального стану системи	Відображення загального енергоспоживання, стану ключових приладів, графіків споживання
Екран приладів	Керування окремими приладами	Перегляд списку приладів, керування їх станом, перегляд деталей споживання кожного приладу
Екран статистики	Детальний аналіз енергоспоживання	Графіки та діаграми споживання за різні періоди, порівняльний аналіз
Екран автоматизації	Налаштування автоматичних сценаріїв	Створення правил автоматизації, налаштування розкладів роботи приладів
Екран сповіщень	Перегляд та налаштування сповіщень	Історія сповіщень, налаштування правил для отримання сповіщень
Екран налаштувань	Конфігурація додатку	Налаштування профілю користувача.

Рівень бізнес-логіки містить основну логіку додатку, включаючи обробку даних, алгоритми аналізу енергоспоживання, логіку керування приладами та формування рекомендацій. Цей рівень використовує шаблон Repository для абстрагування джерел даних та забезпечення єдиного інтерфейсу для їх отримання [18]. Ключові компоненти бізнес-логіки включають:

- сервіс моніторингу енергоспоживання - відповідає за збір, обробку та аналіз даних про енергоспоживання;
- сервіс керування приладами - забезпечує інтерфейс для взаємодії з розумними приладами;
- сервіс автоматизації - керує виконанням автоматичних сценаріїв та правил;
- сервіс рекомендацій - генерує персоналізовані рекомендації для оптимізації енергоспоживання;
- сервіс сповіщень - відповідає за створення та доставку сповіщень користувачу.

Рівень доступу до даних забезпечує взаємодію з різними джерелами даних, включаючи локальну базу даних, мережеві API та пристрої розумного дому. Цей рівень абстрагує деталі реалізації конкретних джерел даних та забезпечує єдиний інтерфейс для їх використання [19]. Для локального зберігання даних використовується SQLite, що забезпечує ефективне зберігання та швидкий доступ до історичних даних про енергоспоживання.

Рівень зовнішніх сервісів та API відповідає за комунікацію з серверною частиною системи та зовнішніми сервісами. Він включає клієнти для взаємодії з REST API, MQTT брокерами та іншими протоколами, які використовуються для комунікації з пристроями розумного дому [21].

Для забезпечення надійного зв'язку між компонентами системи використовуються різні протоколи комунікації, вибір яких залежить від конкретного сценарію використання. Основні протоколи та їх застосування представлені в табл. 2.2.

Таблиця 2.2 - Протоколи комунікації в системі

Протокол	Призначення	Переваги	Обмеження
HTTP/HTTPS	Взаємодія з RESTful API серверної частини	Широка підтримка, простота використання, безпека (HTTPS)	Не підходить для реального часу, висока затримка
MQTT	Комунікація з IoT пристроями, отримання даних в реальному часі	Низькі накладні витрати, підтримка "публікація-підписка", робота при нестабільному з'єднанні	Обмежена підтримка в деяких пристроях, потребує MQTT брокера
WebSocket	Двонаправлена комунікація в реальному часі	Повнодуплексний зв'язок, низька затримка	Вищі вимоги до ресурсів порівняно з MQTT
Bluetooth LE	Пряма комунікація з пристроями поблизу	Низьке енергоспоживання, не потребує інтернет-з'єднання	Обмежений радіус дії, обмежена кількість одночасних підключень
Wi-Fi Direct	Пряма комунікація з пристроями в локальній мережі	Висока швидкість передачі даних, не потребує інтернет-з'єднання	Вище енергоспоживання порівняно з Bluetooth LE

Для забезпечення безпеки комунікації між компонентами системи використовуються такі механізми: шифрування даних при передачі

(TLS/SSL); автентифікація за допомогою токенів (JWT); авторизація на основі ролей; захист від атак "людина посередині" [20].

Важливим аспектом архітектури є забезпечення автономної роботи додатку при відсутності інтернет-з'єднання. Для цього використовується локальне кешування даних та механізм синхронізації, який забезпечує оновлення даних при відновленні з'єднання [16].

2.2 Розробка алгоритмів аналізу та оптимізації енергоспоживання

Аналіз та оптимізація енергоспоживання є ключовими функціями мобільного додатку для моніторингу енергоспоживання побутових приладів. Ефективні алгоритми дозволяють не лише відстежувати поточне споживання, але й виявляти закономірності, прогнозувати майбутнє споживання та формувати рекомендації щодо оптимізації використання електроенергії.

2.3.1 Алгоритми аналізу історичних даних

Аналіз історичних даних енергоспоживання дозволяє виявити закономірності та тренди, які є основою для подальших прогнозів та рекомендацій. Основні алгоритми аналізу історичних даних включають:

- статистичний аналіз - обчислення базових статистичних показників (середнє значення, медіана, мінімум, максимум, стандартне відхилення) для різних часових періодів та категорій приладів;
- аналіз трендів - виявлення довгострокових тенденцій у споживанні електроенергії за допомогою методів регресійного аналізу;
- сезонний аналіз - виявлення сезонних патернів у споживанні електроенергії (денні, тижневі, місячні, сезонні цикли);

– кореляційний аналіз - виявлення взаємозв'язків між споживанням електроенергії та зовнішніми факторами (погодні умови, час доби, день тижня тощо).

Для реалізації статистичного аналізу використовується алгоритм, представлений у лістингу Б.1.

2.3.2 Алгоритми виявлення аномалій

Виявлення аномалій у споживанні електроенергії дозволяє ідентифікувати нетипові патерни, які можуть свідчити про несправності приладів, їх неправильне використання або інші проблеми [22]. Для виявлення аномалій використовуються наступні підходи:

- статистичний підхід - виявлення значень, які суттєво відхиляються від норми (наприклад, перевищують середнє значення плюс три стандартних відхилення);
- метод рухомого вікна - аналіз даних у межах рухомого часового вікна для виявлення різких змін;
- метод кластеризації - групування даних у кластери та виявлення точок, які не належать до жодного кластера;
- метод класифікації - використання навчених моделей для класифікації патернів споживання як нормальних або аномальних.

Реалізація алгоритму виявлення аномалій на основі статистичного підходу представлена в лістингу Б.2.

2.3.3 Алгоритми прогнозування

Прогнозування майбутнього енергоспоживання дозволяє користувачам планувати використання електроенергії та оптимізувати витрати [24]. Для прогнозування використовуються наступні методи:

- лінійна регресія - простий метод для прогнозування на основі лінійної залежності між часом та споживанням;
- авторегресійні моделі (ARIMA) - метод для прогнозування часових рядів, який враховує автокореляцію;
- експоненціальне згладжування - метод, що надає більшу вагу недавнім спостереженням;
- машинне навчання - використання алгоритмів машинного навчання (нейронні мережі, градієнтний бустинг тощо) для прогнозування на основі множини факторів.

Реалізація простого алгоритму прогнозування на основі авторегресії представлена в лістингу Б.3.

2.3.4 Алгоритми формування рекомендацій

Формування персоналізованих рекомендацій щодо оптимізації енергоспоживання є важливою функцією мобільного додатку, яка допомагає користувачам зменшити витрати на електроенергію [23]. Для формування рекомендацій використовуються наступні підходи:

- правила на основі експертних знань - набір заздалегідь визначених правил, які застосовуються в залежності від патернів споживання;
- порівняльний аналіз - порівняння споживання користувача з оптимальними показниками або споживанням інших користувачів;
- машинне навчання - використання алгоритмів машинного навчання для генерації персоналізованих рекомендацій на основі історичних даних та дій користувача;
- оптимізація розкладу - формування оптимального розкладу роботи приладів з урахуванням тарифів на електроенергію та патернів використання.

Реалізація алгоритму формування рекомендацій на основі правил представлена в Б.4.

2.3 Вибір мови програмування

Вибір мови програмування є ключовим етапом в процесі розробки мобільного додатку для моніторингу та керування енергоспоживанням побутових приладів. Цей вибір суттєво впливає на продуктивність, зручність розробки, можливості інтеграції з різними системами та платформами, а також на подальшу підтримку та розвиток програмного забезпечення. Враховуючи специфіку проєкту, необхідно провести аналіз кількох популярних мов програмування та платформ для мобільної розробки з метою обґрунтованого вибору найбільш відповідного інструментарію [7].

На сьогоднішній день існує декілька підходів до розробки мобільних додатків: нативна розробка окремо для кожної платформи, крос-платформна розробка та гібридна розробка. Кожен з цих підходів має свої переваги та недоліки, які необхідно враховувати при виборі мови програмування для розробки додатку [8]. Для детального аналізу розглянемо чотири сучасні мови та фреймворки для мобільної розробки: Swift, Kotlin, Flutter (Dart) та React Native (JavaScript).

Swift є сучасною мовою програмування, розробленою компанією Apple спеціально для створення додатків для екосистеми Apple. Мова була представлена в 2014 році як заміна Objective-C і швидко набула популярності серед розробників iOS-додатків. Swift відрізняється високою продуктивністю, безпечністю типів, сучасним синтаксисом та потужними функціями, такими як опціональні типи, розширення та протоколи [24]. Для розробки інтерфейсу користувача Swift використовується разом із фреймворком SwiftUI або UIKit.

Основними перевагами Swift є висока продуктивність додатків, оптимізація під платформу iOS, глибока інтеграція з іншими сервісами Apple, потужна система типів, що запобігає багатьом помилкам на етапі компіляції, та активна підтримка від Apple і спільноти розробників. Однак Swift має суттєвий недолік в контексті нашого проєкту: він обмежений екосистемою

Apple, що означає неможливість використання для розробки додатків для Android [3].

Kotlin є офіційною мовою для розробки Android-додатків, яка отримала статус пріоритетної мови від Google у 2017 році. Це статично типізована мова, яка повністю сумісна з Java, але пропонує більш сучасний та лаконічний синтаксис, функціональні можливості та покращену безпеку. Kotlin дозволяє уникнути багатьох поширених помилок, таких як NullPointerException, завдяки вбудованій підтримці нульової безпеки [3].

Переваги Kotlin включають високу продуктивність додатків, повну сумісність з Java та існуючими Android-бібліотеками, відмінну підтримку від Google та активну спільноту розробників, а також можливість використання сучасних функціональних патернів програмування. Однак, як і Swift, Kotlin в основному орієнтований на одну платформу (Android), хоча існують рішення для крос-платформної розробки з Kotlin, такі як Kotlin Multiplatform [2].

Flutter є відкритим фреймворком для розробки крос-платформних мобільних додатків, створеним компанією Google. Він використовує мову програмування Dart і власний механізм рендерингу інтерфейсу, що дозволяє створювати високопродуктивні додатки з нативним виглядом для обох платформ (iOS та Android) з єдиної кодової бази. Flutter використовує підхід "все є віджетом", що спрощує створення складних інтерфейсів [3].

Ключовими перевагами Flutter є можливість розробки для обох основних мобільних платформ з використанням єдиної кодової бази, що значно скорочує час та вартість розробки; висока продуктивність, близька до нативних додатків; "гарячий перезапуск" (hot reload), який дозволяє миттєво бачити зміни в коді, що прискорює розробку та відладку; багата бібліотека готових віджетів, що відповідають дизайн-гайдлайнам Material Design та Cupertino; а також активна підтримка від Google та зростаюча спільнота [4].

React Native є фреймворком для розробки крос-платформних мобільних додатків, створеним компанією Facebook (Meta). Він використовує JavaScript та бібліотеку React для створення інтерфейсу, який компілюється у

нативні компоненти для iOS та Android. Це дозволяє розробникам використовувати вебтехнології для створення мобільних додатків з нативною продуктивністю та виглядом [5].

Основними перевагами React Native є можливість використання JavaScript, однієї з найпопулярніших мов програмування, що спрощує вхід для веброзробників; крос-платформність, що дозволяє розробляти для обох платформ з єдиною кодовою базою; велика екосистема бібліотек та компонентів; функція "гарячого перезавантаження" для швидкої розробки; а також можливість інтеграції з нативними модулями для доступу до специфічних функцій платформи [6].

Для систематизації порівняння цих технологій розглянемо їх за ключовими параметрами, важливими для розробки мобільного додатку моніторингу енергоспоживання (табл. 2.3).

На основі порівняльного аналізу (табл. 2.3) та з урахуванням специфіки проєкту з розробки мобільного додатку для моніторингу та керування енергоспоживанням побутових приладів, можна зробити висновок про доцільність використання Flutter як основної технології розробки. Цей вибір обґрунтований наступними факторами:

Flutter забезпечує можливість розробки для обох основних мобільних платформ (iOS та Android) з використанням єдиної кодової бази, що значно зменшує ресурси, необхідні для розробки та підтримки додатку. Завдяки власному механізму рендерингу Flutter забезпечує високу продуктивність, близьку до нативних додатків, що особливо важливо для додатків, які працюють з даними в реальному часі, такими як показники енергоспоживання [7].

В контексті додатку для моніторингу енергоспоживання, візуалізація даних є критично важливим компонентом. Flutter пропонує потужні інструменти для створення складних інтерфейсів та візуалізації даних завдяки багатій бібліотеці віджетів та плагінів для роботи з графіками та діаграмами, таким як `fl_chart`, `syncfusion_flutter_charts` та інші [8]. Це

дозволяє ефективно відображати статистику та тренди енергоспоживання у зручному для користувача форматі.

Таблиця 2.3 - Порівняння мов програмування та фреймворків для мобільної розробки

Критерій	Swift	Kotlin	Flutter (Dart)	React Native (JavaScript)
Підтримувані платформи	iOS	Android	iOS, Android, веб, десктоп	iOS, Android
Продуктивність	Висока (нативна)	Висока (нативна)	Висока (близька до нативної)	Середня-висока
Доступ до нативних функцій платформи	Повний	Повний	Через плагіни	Через нативні модулі
Спільнота та екосистема	Велика, активна	Велика, активна	Зростаюча	Велика, активна
Крива навчання	Середня	Низька-середня	Середня	Низька для веброзробників
Інтеграція з IoT та системами розумного дому	Обмежена	Обмежена	Хороша, через плагіни	Хороша, через нативні модулі
Стабільність та зрілість	Висока	Висока	Середня	Середня-висока
Підтримка протоколів комунікації для IoT	Обмежена	Обмежена	Широка, через плагіни	Широка, через нативні модулі

Для взаємодії з пристроями розумного дому та датчиками енергоспоживання Flutter має широкий набір плагінів, які забезпечують підтримку різних протоколів комунікації, таких як Bluetooth LE, Wi-Fi, MQTT та інші. Зокрема, плагіни `flutter_blue`, `mqtt_client`, `wifi_info_flutter` дозволяють легко інтегрувати додаток з різними типами IoT-пристроїв [9].

Архітектура Flutter також добре підходить для реалізації запропонованої в розділі 2.1 багаторівневої архітектури додатку. Для

реалізації шаблону MVVM можна використовувати такі рішення як Provider, Riverpod або Bloc, які забезпечують ефективне управління станом додатку та розділення логіки [4]. Для роботи з даними Flutter пропонує різні інструменти для локального зберігання даних (sqflite, hive, shared_preferences) та мережевих запитів (dio, http).

Додатковою перевагою Flutter є інструмент "гарячого перезапуску" (hot reload), який дозволяє миттєво бачити зміни в коді, що значно прискорює процес розробки та тестування. Це особливо корисно при налаштуванні інтерфейсу та візуалізації даних енергоспоживання [1].

Також варто відзначити, що Flutter активно розвивається та підтримується компанією Google, що забезпечує його довгострокову перспективу та постійне вдосконалення. Спільнота розробників Flutter також швидко зростає, що розширює екосистему бібліотек та інструментів, доступних для розробки [4].

Для розробки серверної частини системи, яка буде забезпечувати зберігання та обробку даних енергоспоживання, а також взаємодію з мобільним додатком, рекомендується використовувати Node.js з фреймворком Express.js та базою даних SQLite. Цей стек технологій забезпечує високу продуктивність, масштабованість та гнучкість, необхідні для обробки даних в реальному часі та аналітики енергоспоживання [26].

2.4 Висновки за розділом 2

У другому розділі проведено проєктування архітектури та розробку алгоритмів аналізу і оптимізації енергоспоживання для мобільного додатку розумного моніторингу та керування енергоспоживанням побутових приладів.

Запропонована багаторівнева архітектура мобільного додатку забезпечує чітке розділення відповідальності між компонентами системи, що підвищує модульність, тестованість та гнучкість програмного забезпечення.

Архітектура складається з чотирьох ключових рівнів: рівня представлення (UI), рівня бізнес-логіки, рівня доступу до даних та рівня зовнішніх сервісів і API. Обґрунтовано використання шаблону проєктування MVVM (Model-View-ViewModel) для рівня представлення, що забезпечує ефективне відокремлення логіки представлення від моделі даних.

Розроблено та детально описано алгоритми аналізу та оптимізації енергоспоживання, які є основою функціональності мобільного додатку. Особлива увага приділена адаптації цих алгоритмів до роботи в умовах обмежених обчислювальних ресурсів мобільних пристроїв.

Порівняльний аналіз технологій розробки (Swift, Kotlin, Flutter, React Native) за критеріями підтримуваних платформ, продуктивності, доступу до нативних функцій, спільноти розробників, кривої навчання та інтеграції з IoT-пристроями підтвердив оптимальність вибору Flutter для даного проєкту.

Для забезпечення надійної комунікації між компонентами системи запропоновано використання різних протоколів (HTTP/HTTPS, MQTT, WebSocket, Bluetooth LE, Wi-Fi Direct) відповідно до конкретних сценаріїв використання, що підвищує ефективність роботи додатку в різних умовах мережевого підключення.

3 ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Структура бази даних для збереження даних енергоспоживання

Ефективне збереження, обробка та аналіз даних про енергоспоживання є ключовими аспектами функціонування мобільного додатку для моніторингу та керування енергоспоживанням побутових приладів. Проєктування бази даних для такої системи повинно враховувати специфіку даних енергоспоживання, вимоги до продуктивності та обмеження мобільних пристроїв [15].

Для реалізації цих вимог пропонується використовувати локальну реляційну базу даних SQLite, яка є стандартним рішенням для мобільних додатків та забезпечує хороший баланс між продуктивністю, функціональністю та використанням ресурсів [19]. Концептуальна модель бази даних представлена на рис. 3.1.

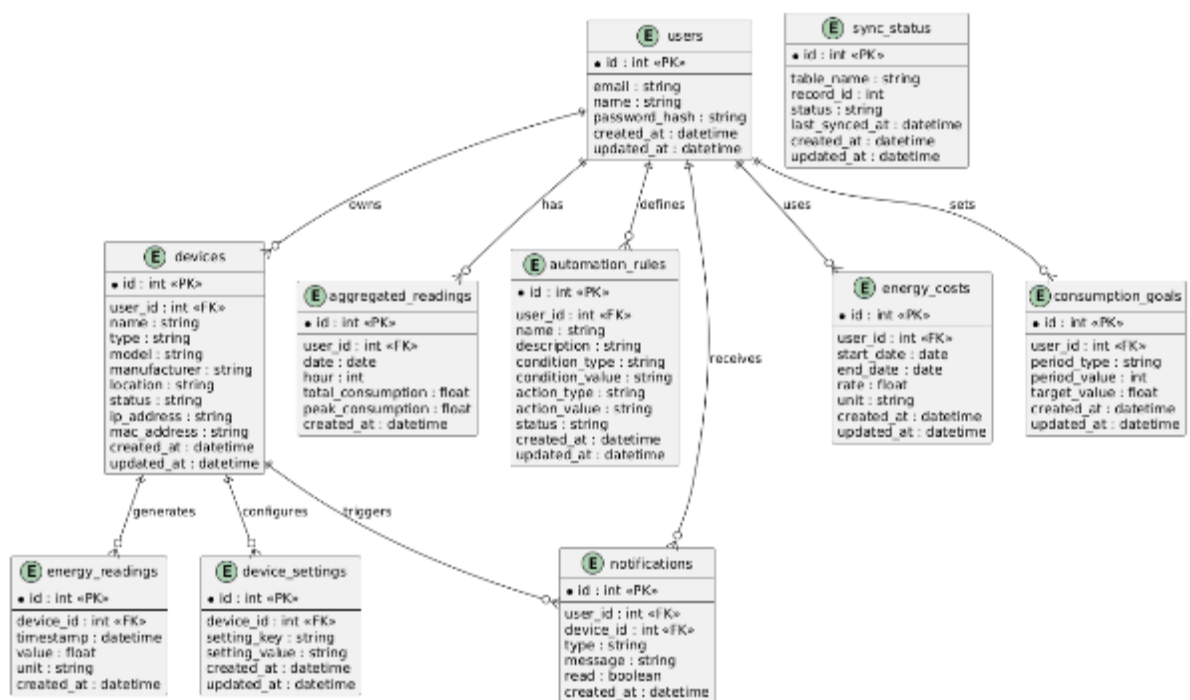


Рисунок 3.1 - Концептуальна модель бази даних мобільного додатку

Як видно з рис. 3.1, база даних складається з кількох основних сутностей, які відображають ключові об'єкти системи та зв'язки між ними. Центральними сутностями є "Пристрої", "Показники енергоспоживання" та "Користувачі", які пов'язані з іншими сутностями для забезпечення повної функціональності системи [20].

Деталізована структура таблиць бази даних представлена в табл. 3.1, яка містить опис основних таблиць, їх атрибутів та призначення.

Таблиця 3.1 - Структура таблиць бази даних

Таблиця	Атрибути	Призначення
users	id (PK), email, name, password_hash, created_at, updated_at	Зберігання інформації про користувачів системи
devices	id (PK), user_id (FK), name, type, model, manufacturer, location, status, ip_address, mac_address, created_at, updated_at	Інформація про пристрої розумного дому
energy_readings	id (PK), device_id (FK), timestamp, value, unit, created_at	Показники енергоспоживання окремих пристроїв
aggregated_readings	id (PK), user_id (FK), date, hour, total_consumption, peak_consumption, created_at	Агреговані дані про загальне споживання
device_settings	id (PK), device_id (FK), setting_key, setting_value, created_at, updated_at	Налаштування пристроїв

Продовження таблиці 3.1

Таблиця	Атрибути	Призначення
automation_rules	id (PK), user_id (FK), name, description, condition_type, condition_value, action_type, action_value, status, created_at, updated_at	Правила автоматизації для керування пристроями
notifications	id (PK), user_id (FK), device_id (FK), type, message, read, created_at	Сповіщення для користувачів
energy_costs	id (PK), user_id (FK), start_date, end_date, rate, unit, created_at, updated_at	Тарифи на електроенергію для розрахунку вартості
consumption_goals	id (PK), user_id (FK), period_type, period_value, target_value, created_at, updated_at	Цілі користувача щодо енергоспоживання
sync_status	id (PK), table_name, record_id, status, last_synced_at, created_at, updated_at	Інформація про статус синхронізації даних з сервером

Для створення таблиць бази даних використовуються SQL-запити, які визначають структуру таблиць, типи даних, первинні та зовнішні ключі, а також індекси для оптимізації пошуку. Приклад SQL-запиту для створення таблиці energy_readings представлений в лістингу 3.1.

Лістинг 3.1 - SQL-запит для створення таблиці energy_readings

```
CREATE TABLE energy_readings (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    device_id INTEGER NOT NULL,
```

```

timestamp INTEGER NOT NULL,
value REAL NOT NULL,
unit TEXT NOT NULL,
created_at INTEGER NOT NULL DEFAULT (strftime('%s', 'now')),
FOREIGN KEY (device_id) REFERENCES devices(id) ON DELETE
CASCADE
);

```

```

CREATE INDEX idx_energy_readings_device_id ON
energy_readings(device_id);
CREATE INDEX idx_energy_readings_timestamp ON
energy_readings(timestamp);

```

Для забезпечення синхронізації даних між мобільним додатком та серверною частиною системи використовується механізм відстеження змін та статусу синхронізації. Кожна зміна в локальній базі даних фіксується в таблиці `sync_status`, що дозволяє відстежувати, які дані ще не були синхронізовані з сервером [21]. При відновленні з'єднання з сервером відбувається двостороння синхронізація даних:

- відправлення на сервер локальних змін, які ще не були синхронізовані;
- отримання від сервера змін, які були зроблені на інших пристроях;
- розв'язання конфліктів, якщо однакові дані були змінені на різних пристроях.

Для оптимізації запитів до бази даних та забезпечення швидкого доступу до найчастіше використовуваних даних використовується система індексів та кешування [20]. Основні індекси бази даних представлені в табл. 3.2.

Таблиця 3.2 - Основні індекси бази даних

Таблиця	Індекс	Поля	Призначення
energy_readings	idx_energy_readings_device_id	device_id	Оптимізація пошуку показників для конкретного пристрою
energy_readings	idx_energy_readings_timestamp	timestamp	Оптимізація вибірки даних за період
energy_readings	idx_energy_readings_device_timestamp	device_id, timestamp	Оптимізація вибірки даних для пристрою за період
aggregated_readings	idx_aggregated_readings_user_date	user_id, date	Оптимізація вибірки агрегованих даних за період
devices	idx_devices_user_id	user_id	Оптимізація пошуку пристроїв користувача
automation_rules	idx_automation_rules_user_id	user_id	Оптимізація пошуку правил автоматизації користувача
notifications	idx_notifications_user_id	user_id	Оптимізація вибірки сповіщень для користувача
notifications	idx_notifications_device_id	device_id	Оптимізація вибірки сповіщень для пристрою

3.2 Розробка користувацького інтерфейсу мобільного додатку

Користувацький інтерфейс мобільного додатку для моніторингу та керування енергоспоживанням є ключовим елементом, який визначає зручність використання системи та впливає на її прийняття користувачами. При проектуванні інтерфейсу важливо забезпечити не тільки естетичну привабливість, але й функціональність, інтуїтивність та відповідність потребам різних груп користувачів. У цьому розділі розглядаються основні принципи розробки інтерфейсу, структура навігації, ключові екрани та компоненти мобільного додатку.

При розробці інтерфейсу мобільного додатку для моніторингу енергоспоживання враховувалися сучасні принципи дизайну мобільних інтерфейсів, такі як мінімалізм, інтуїтивність, адаптивність та доступність. Особлива увага приділялася візуалізації даних про енергоспоживання, оскільки це є основною функцією додатку. Візуалізація повинна бути не тільки інформативною, але й зрозумілою для користувачів без спеціальних технічних знань.

Структура навігації в додатку базується на принципі "три кліки до мети", що дозволяє користувачу швидко дістатися до потрібної інформації чи функції. Основна навігація реалізована через нижню панель (Bottom Navigation Bar), яка забезпечує доступ до ключових розділів додатку (рис. 3.2).

Як видно з рис. 3.2, навігаційна структура додатку включає п'ять основних розділів: головний екран (Dashboard), список пристроїв (Devices), статистика (Statistics), автоматизація (Automation) та налаштування (Settings). Додатково, з головного екрану доступні спеціалізовані функції, такі як перегляд повідомлень, сповіщення та швидкий доступ до керування ключовими пристроями.

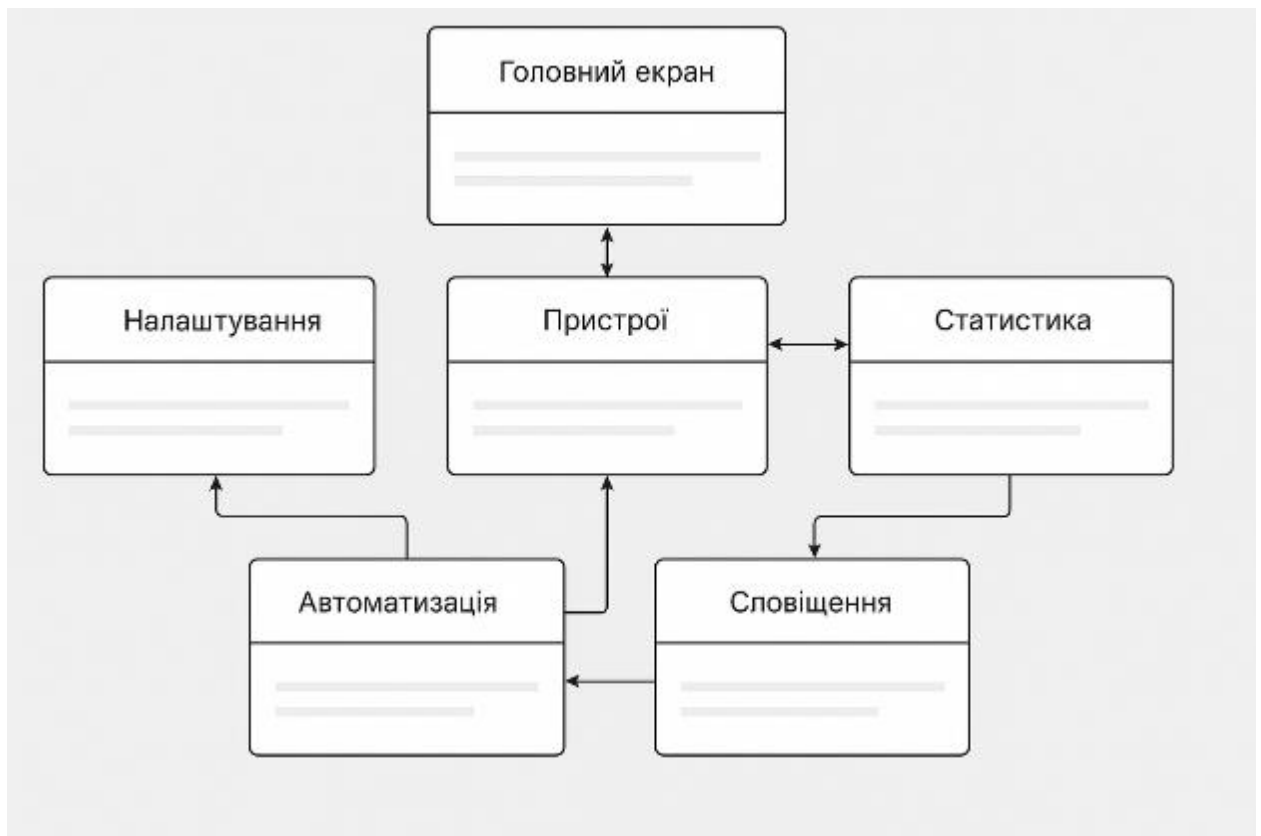


Рисунок 3.2 - Структура навігації мобільного додатку

Головний екран (Dashboard) є першим, що бачить користувач після входу в додаток, і надає загальний огляд поточного стану енергоспоживання. Він розроблений з урахуванням принципів інформаційного дизайну, що дозволяє відображати максимум корисної інформації в компактному форматі без перевантаження користувача. Прототип головного екрану представлений на рис. 3.3.

На головному екрані (рис. 3.3) представлені наступні ключові елементи: графік поточного енергоспоживання з можливістю перемикання між різними часовими періодами; карточки з інформацією про загальне споживання, вартість та прогноз на поточний місяць; статус ключових пристроїв з можливістю швидкого керування; сповіщення про аномалії та події; мінливі рекомендації щодо оптимізації енергоспоживання.

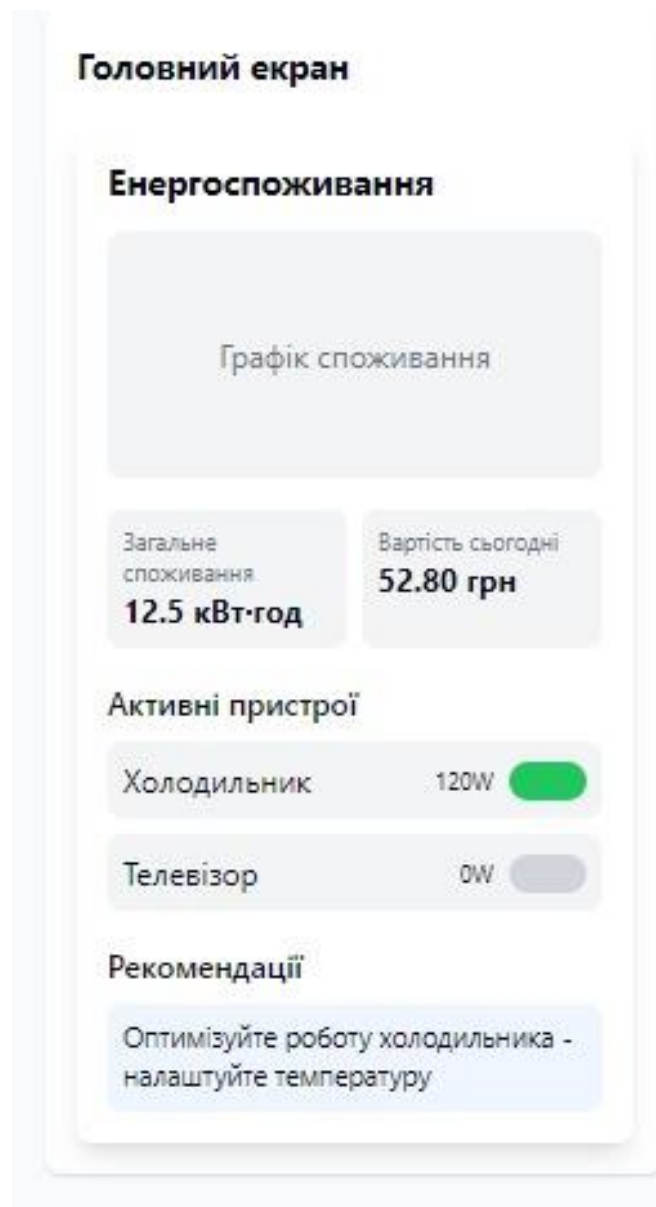


Рисунок 3.3 - Прототип головного екрану (Dashboard) мобільного додатку

Екран пристроїв (Devices) представляє список усіх підключених до системи пристроїв з можливістю фільтрації за різними параметрами (тип, розташування, стан). Для кожного пристрою відображається базова інформація про його стан та енергоспоживання, а також надається можливість швидкого керування. При натисканні на пристрій користувач переходить на детальний екран пристрою, де представлена розширена інформація та повний набір елементів керування. Прототип екрану пристроїв та детального екрану пристрою показаний на рис. 3.4.

Екран пристроїв

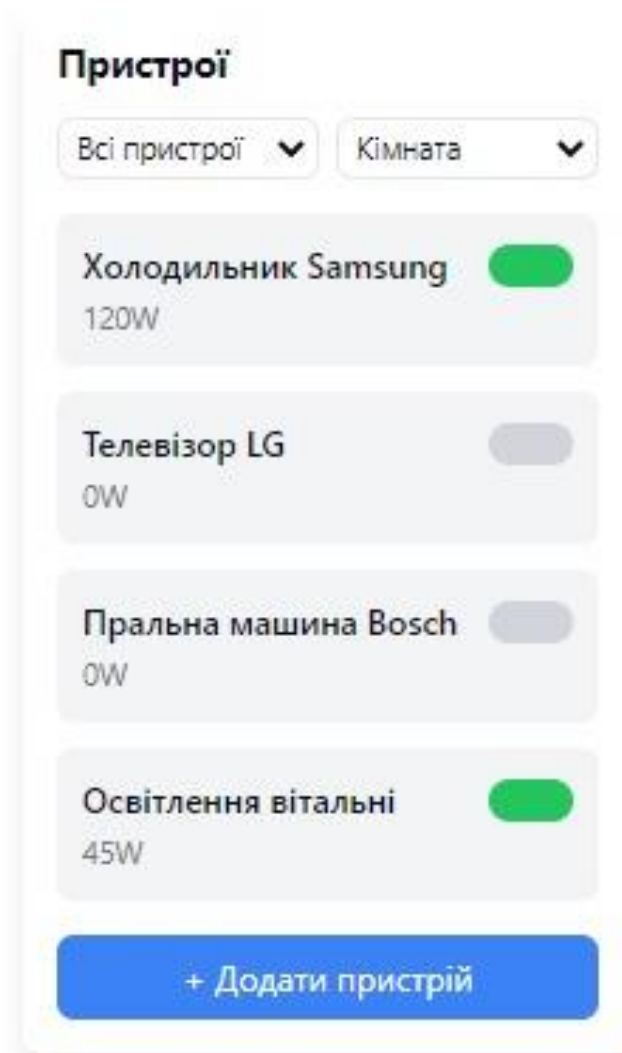


Рисунок 3.4 - Прототип екрану пристроїв (Devices) та детального екрану пристрою

Екран статистики (Statistics) надає розширені можливості для аналізу даних енергоспоживання з використанням різних типів візуалізації (графіки, діаграми, таблиці). Користувач може вибрати період для аналізу, порівняти споживання різних пристроїв або періодів, а також отримати детальний розрахунок вартості на основі діючих тарифів. Прототип екрану статистики представлений на рис. 3.5.



Рисунок 3.5 - Прототип екрану статистики (Statistics) мобільного додатку

Екран налаштувань (Settings) надає доступ до загальних налаштувань додатку, управління профілем користувача, налаштування тарифів, конфігурації сповіщень та інших параметрів системи. Особлива увага приділяється налаштуванням безпеки та конфіденційності даних, що є критично важливим для системи, яка має доступ до інформації про енергоспоживання та можливість керувати побутовими приладами (рис. 3.6).

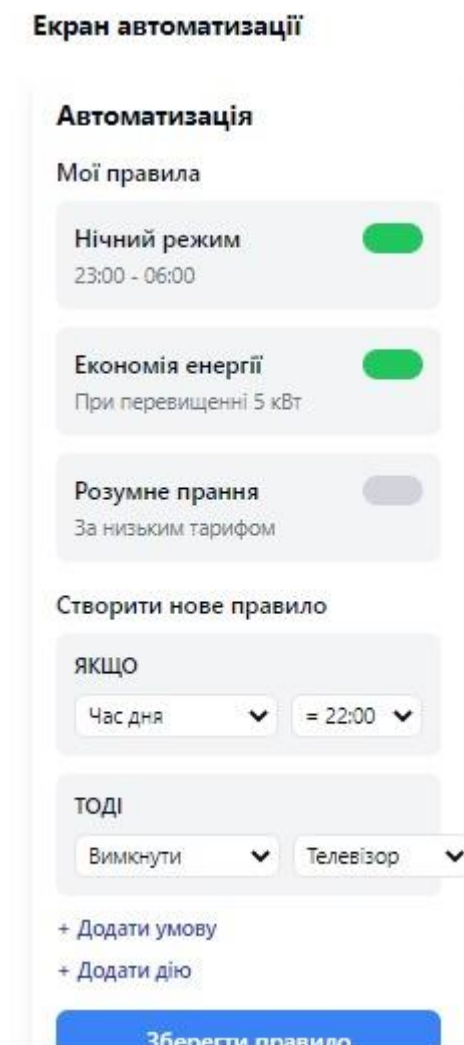


Рисунок 3.6 – Екран автоматизації

Дизайн інтерфейсу розроблений з використанням компонентного підходу, що забезпечує узгодженість візуального стилю та поведінки елементів інтерфейсу по всьому додатку. Основні компоненти інтерфейсу включають: карточки для відображення інформації про пристрої та показники; графіки та діаграми для візуалізації даних; елементи керування (кнопки, перемикачі, слайдери); форми для введення даних; сповіщення та діалогові вікна.

У табл. 3.3 представлено основні компоненти інтерфейсу та їх призначення.

Таблиця 3.3 - Основні компоненти інтерфейсу мобільного додатку

Компонент	Призначення	Особливості
Карточка пристрою	Відображення інформації про пристрій та його стан	Адаптивний розмір, індикатор стану, швидкі дії
Графік споживання	Візуалізація даних про енергоспоживання	Інтерактивність, різні типи графіків, можливість масштабування
Карточка статистики	Відображення агрегованих показників	Виділення ключових метрик, порівняння з попередніми періодами
Елемент рекомендації	Відображення персоналізованих рекомендацій	Пріоритезація, можливість відкладення або відхилення
Форма створення правила	Інтерфейс для створення правил автоматизації	Покроковий майстер, перевірка коректності, підказки
Діалог сповіщення	Відображення важливих сповіщень	Різні рівні важливості, можливість дій

Інтерфейс додатку розроблений з урахуванням принципів адаптивного дизайну, що забезпечує коректне відображення на пристроях з різними розмірами екранів та роздільною здатністю. Це досягається завдяки використанню гнучких макетів (Flexible layouts), відносних розмірів (Relative sizes) та адаптивних компонентів (Adaptive components).

Для забезпечення зручності використання додатку особами з обмеженими можливостями, у процесі розробки інтерфейсу впроваджені принципи доступності (Accessibility), такі як: підтримка збільшення розміру тексту; сумісність з програмами читання з екрану (Screen readers); достатня контрастність кольорів; підтримка керування жестами та голосом; альтернативний текст для зображень.

Процес розробки інтерфейсу включав декілька етапів: аналіз вимог та потреб користувачів; створення концептуальних макетів (wireframes); розробка детальних прототипів (mockups); тестування прототипів з користувачами; ітеративне вдосконалення дизайну на основі зворотного зв'язку; реалізація інтерфейсу з використанням фреймворку Flutter.

Для реалізації інтерфейсу в Flutter використовуються наступні підходи та компоненти: застосування архітектури MVVM (Model-View-ViewModel) для розділення логіки та інтерфейсу; використання віджетів Material Design для забезпечення узгодженого вигляду та поведінки; застосування бібліотеки Flutter Charts для створення інтерактивних графіків; використання анімацій для забезпечення плавності переходів та поліпшення взаємодії користувача з додатком; реалізація кешування та асинхронного завантаження даних для забезпечення швидкодії інтерфейсу.

Вибір Flutter як технології для реалізації інтерфейсу користувача пояснюється кількома ключовими перевагами: можливість розробки для обох платформ (iOS та Android) з використанням єдиної кодової бази; висока продуктивність та плавність анімацій; багата бібліотека віджетів, які можна налаштовувати; можливість створення складних та інтерактивних інтерфейсів; підтримка різних типів візуалізації даних; швидкий цикл розробки завдяки функції "гарячого перезавантаження".

При розробці інтерфейсу особлива увага приділялася оптимізації продуктивності, оскільки додаток працює з великими обсягами даних та складними візуалізаціями. Для цього використовувалися наступні техніки: лінива ініціалізація компонентів; використання кешування для зменшення кількості запитів до бази даних; оптимізація перемальовування інтерфейсу з використанням віджета `const`; застосування пагінації для відображення великих списків; асинхронне завантаження даних без блокування інтерфейсу.

Для забезпечення ефективного відображення даних енергоспоживання в різних форматах використовуються спеціалізовані бібліотеки, такі як `fl_chart` для створення графіків та діаграм, `syncfusion_flutter_charts` для більш

складних візуалізацій, та `svg_flutter` для роботи з векторною графікою. Це дозволяє створювати інтерактивні та інформативні візуалізації, які допомагають користувачу швидко оцінювати та аналізувати дані про енергоспоживання.

Інтерфейс також включає систему повідомлень та сповіщень, яка інформує користувача про важливі події та аномалії в енергоспоживанні. Сповіщення можуть бути різних типів: інформаційні, попереджувальні та критичні. Кожен тип має свій візуальний стиль та поведінку, що допомагає користувачу швидко оцінити важливість повідомлення.

Для персоналізації інтерфейсу додаток надає можливість налаштування тем (світла та темна), розміру тексту, основних елементів інтерфейсу, а також вибору відображуваних метрик та типів візуалізації. Це дозволяє адаптувати інтерфейс під конкретні потреби та переваги користувача.

Таким чином, розроблений користувацький інтерфейс мобільного додатку для моніторингу та керування енергоспоживанням забезпечує зручний та інтуїтивно зрозумілий доступ до функціональності системи, ефективну візуалізацію даних та можливість керування пристроями. Використання сучасних підходів до дизайну та розробки інтерфейсу, а також врахування потреб різних груп користувачів, дозволяє створити додаток, який буде зручним для використання та ефективним інструментом для оптимізації енергоспоживання в побутовому секторі.

3.3 Реалізація алгоритмів аналізу та оптимізації енергоспоживання

Реалізація алгоритмів аналізу та оптимізації енергоспоживання є одним із ключових етапів розробки програмного забезпечення для моніторингу та керування енергоспоживанням побутових приладів. У цьому розділі описані практичні аспекти реалізації алгоритмів, розглянутих у

розділі 2.2, з використанням мови програмування Dart та фреймворку Flutter, обраних для розробки мобільного додатку.

Для реалізації алгоритмів аналізу та оптимізації енергоспоживання був розроблений окремий модуль, який інкапсулює всю логіку обробки даних та інтегрується з іншими компонентами системи через чітко визначені інтерфейси. Цей підхід забезпечує модульність, тестованість та можливість подальшого розширення функціональності. Структура модуля аналізу енергоспоживання представлена на рис. 3.7.

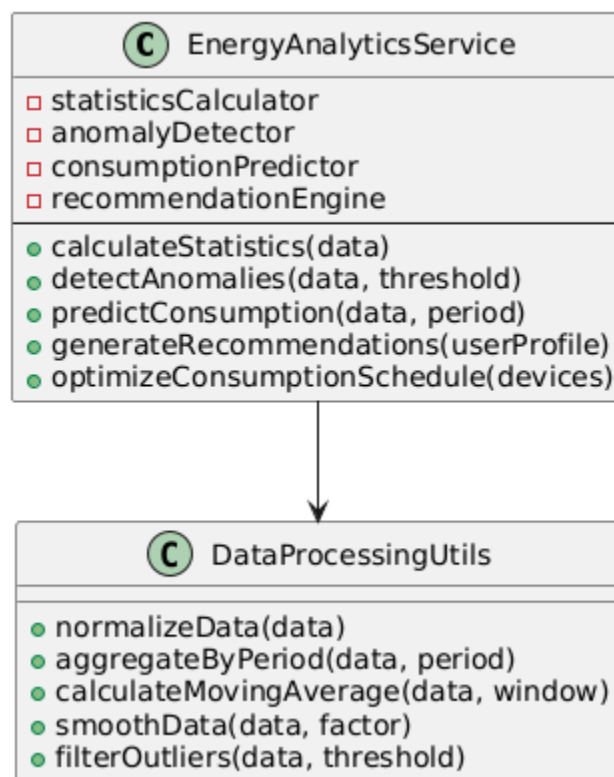


Рисунок 3.7 - Структура модуля аналізу енергоспоживання

Як видно з рис. 3.6, модуль аналізу енергоспоживання складається з головного сервісу EnergyAnalyticsService, який об'єднує всі алгоритми аналізу та оптимізації, та допоміжного класу DataProcessingUtils, що містить набір функцій для попередньої обробки даних. Такий підхід дозволяє ефективно організувати код та забезпечити повторне використання спільних компонентів.

Для реалізації аналізу історичних даних був створений клас `StatisticsCalculator`, який надає методи для обчислення різних статистичних показників енергоспоживання. Цей клас включає реалізацію алгоритмів для розрахунку базових статистичних метрик, аналізу трендів, сезонного аналізу та кореляційного аналізу.

Ключовою функціональністю класу є метод для агрегації та аналізу даних енергоспоживання за різні періоди часу. Реалізація цього методу представлена в табл. 3.4.

Таблиця 3.4 - Реалізація аналізу історичних даних

Функція	Опис	Принцип роботи
Агрегація даних	Об'єднання даних за часовими інтервалами	Групування показань за годинами, днями, тижнями, місяцями з обчисленням сумарних, середніх та максимальних значень
Статистичний аналіз	Обчислення основних статистичних показників	Розрахунок середнього, медіани, моди, стандартного відхилення, мінімуму та максимуму для кожного агрегованого періоду
Аналіз трендів	Виявлення довгострокових тенденцій	Застосування лінійної регресії для виявлення загального тренду та оцінки його статистичної значущості
Сезонний аналіз	Виявлення сезонних патернів	Декомпозиція часового ряду на тренд, сезонну та випадкову компоненти з використанням методу STL (Seasonal-Trend decomposition using LOESS)
Порівняльний аналіз	Порівняння періодів	Обчислення відносних та абсолютних змін показників

Для підвищення продуктивності при роботі з великими об'ємами даних використовується оптимізована структура даних та паралельні обчислення, де це можливо. Зокрема, для агрегації даних застосовується техніка відкладених обчислень (*lazy evaluation*), яка дозволяє обробляти дані поетапно без необхідності завантаження всього набору даних у пам'ять одночасно.

Для виявлення аномалій у даних енергоспоживання реалізовано клас *AnomalyDetector*, який об'єднує різні методи виявлення аномалій, описані в розділі 2.2. Реалізація кожного методу включає оптимізації для роботи в умовах обмежених ресурсів мобільного пристрою.

Центральним компонентом реалізації є адаптивний алгоритм виявлення аномалій, який використовує статистичний підхід з автоматичним підбором порогових значень на основі історичних даних. Цей алгоритм проходить через наступні етапи обробки:

- попередня обробка даних для видалення шуму та очевидних викидів;
- обчислення статистичних показників для визначення "нормальної" поведінки;
- застосування методу *Z*-оцінки для виявлення значень, що значно відхиляються від норми;
- використання методу рухомого вікна для врахування контексту та динамічних паттернів;
- класифікація виявлених аномалій за типом та ступенем відхилення.

Враховуючи, що різні типи побутових приладів мають різні характеристики енергоспоживання, алгоритм адаптується до специфіки кожного типу приладу. Для цього використовуються профілі енергоспоживання, які містять типові характеристики для кожної категорії приладів. Приклад профілів енергоспоживання для різних типів приладів представлений у табл. 3.5.

Таблиця 3.5 - Профілі енергоспоживання для різних типів приладів

Тип приладу	Характер споживання	Типова потужність (Вт)	Допустиме відхилення (%)	Типові аномалії
Холодильник	Циклічний	100-200	± 15	Тривале неперервне споживання, відсутність циклів
Пральна машина	Багатофазний	400-2500	± 20	Надмірне споживання у фазі нагріву, надто довгий цикл
Телевізор	Стабільний	50-150	± 10	Високе споживання в режимі очікування
Освітлення	Стабільний	5-100	± 5	Увімкнення в денний час
Кондиціонер	Змінний	500-3500	± 25	Різкі скачки споживання, неоптимальна температура

Використання профілів енергоспоживання дозволяє значно підвищити точність виявлення аномалій, оскільки враховуються специфічні характеристики кожного типу приладів. Крім того, система постійно навчається на основі реальних даних користувача, уточнюючи профілі та адаптуючи порогові значення.

Для формування персоналізованих рекомендацій щодо оптимізації енергоспоживання реалізовано клас RecommendationEngine, який аналізує дані про енергоспоживання користувача та генерує конкретні рекомендації для зменшення витрат на електроенергію.

Приклад правил з бази знань представлений у табл. 3.6.

Таблиця 3.6 - Приклад правил з бази знань для формування рекомендацій

Категорія	Умова виявлення	Рекомендація	Потенційна економія (%)
Холодильник	Температура нижче +4°C	Підвищити температуру до +5°C	5-10
Пральна машина	Використання при піковому тарифі	Перенести прання на нічний час	20-30
Освітлення	Однчасне використання багатьох джерел	Використовувати локальне освітлення	15-25
Електроніка	Висока потужність у режимі очікування	Відключати від мережі при невикористанні	3-8
Кондиціонер	Температура нижче +22°C	Підвищити температуру до +24°C	10-15

Механізм аналізу патернів використання приладів базується на алгоритмах кластеризації та класифікації, які дозволяють виявляти типові сценарії використання та порівнювати їх з оптимальними моделями. Алгоритм враховує не тільки рівень споживання електроенергії, але й контекст використання, такий як час доби, день тижня, сезон та погодні умови.

Для кожної згенерованої рекомендації система обчислює оцінку потенційної економії, базуючись на історичних даних користувача та статистичних моделях економії для подібних сценаріїв. Це дозволяє пріоритетизувати рекомендації за їх потенційним впливом на загальне споживання.

Особливо важливим аспектом є персоналізація рекомендацій, яка досягається шляхом аналізу взаємодії користувача з системою та адаптації рекомендацій до його специфічних потреб та преференцій. Система відстежує, які рекомендації були прийняті, а які відхилені, і використовує цю інформацію для уточнення майбутніх рекомендацій.

3.4 Висновки за розділом 3

У третьому розділі представлено детальний опис програмного забезпечення для розумного моніторингу та керування енергоспоживанням побутових приладів, що є ключовим аспектом реалізації розробленої системи.

Спроектовано та реалізовано ефективну структуру бази даних на основі SQLite, яка забезпечує надійне зберігання та швидкий доступ до даних енергоспоживання. Розроблена концептуальна модель бази даних включає десять взаємопов'язаних таблиць, що охоплюють усі аспекти роботи системи: від зберігання інформації про користувачів та пристрої до реєстрації показників енергоспоживання, налаштувань автоматизації та персоналізованих рекомендацій. Особлива увага приділена оптимізації запитів через систему індексів та механізму синхронізації даних між мобільним додатком та серверною частиною, що забезпечує ефективну роботу додатку навіть при нестабільному інтернет-підключенні.

Створено інтуїтивно зрозумілий користувацький інтерфейс з використанням фреймворку Flutter, що відповідає сучасним принципам дизайну мобільних додатків. Реалізовано п'ять основних екранів: головний

екран із загальним оглядом енергоспоживання, екран пристроїв для моніторингу та керування окремими приладами, екран статистики для аналізу історичних даних, екран автоматизації та екран налаштувань. Інтерфейс розроблено з використанням компонентного підходу, що забезпечує узгодженість візуального стилю та адаптивність до різних розмірів екранів. Впроваджено принципи доступності для забезпечення зручності використання особами з обмеженими можливостями.

Здійснено практичну реалізацію алгоритмів аналізу та оптимізації енергоспоживання, розглянутих у другому розділі. Для цього розроблено окремий модуль `EnergyAnalyticsService`, що складається з декількох спеціалізованих компонентів: `StatisticsCalculator` для обчислення статистичних показників, `AnomalyDetector` для виявлення аномалій у споживанні, `ConsumptionPredictor` для прогнозування майбутнього енергоспоживання та `RecommendationEngine` для генерації персоналізованих рекомендацій. Особливої уваги заслуговує реалізація адаптивного алгоритму виявлення аномалій, що використовує профілі енергоспоживання для різних типів побутових приладів.

4 ЕКСПЛУАТАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Експлуатація програмного забезпечення користувачем

4.1.1 Встановлення та перше налаштування додатку

Процес встановлення та першого налаштування додатку є ключовим для успішного початку роботи користувача з системою. Послідовність дій користувача при першому запуску представлена на рисунку 4.1.

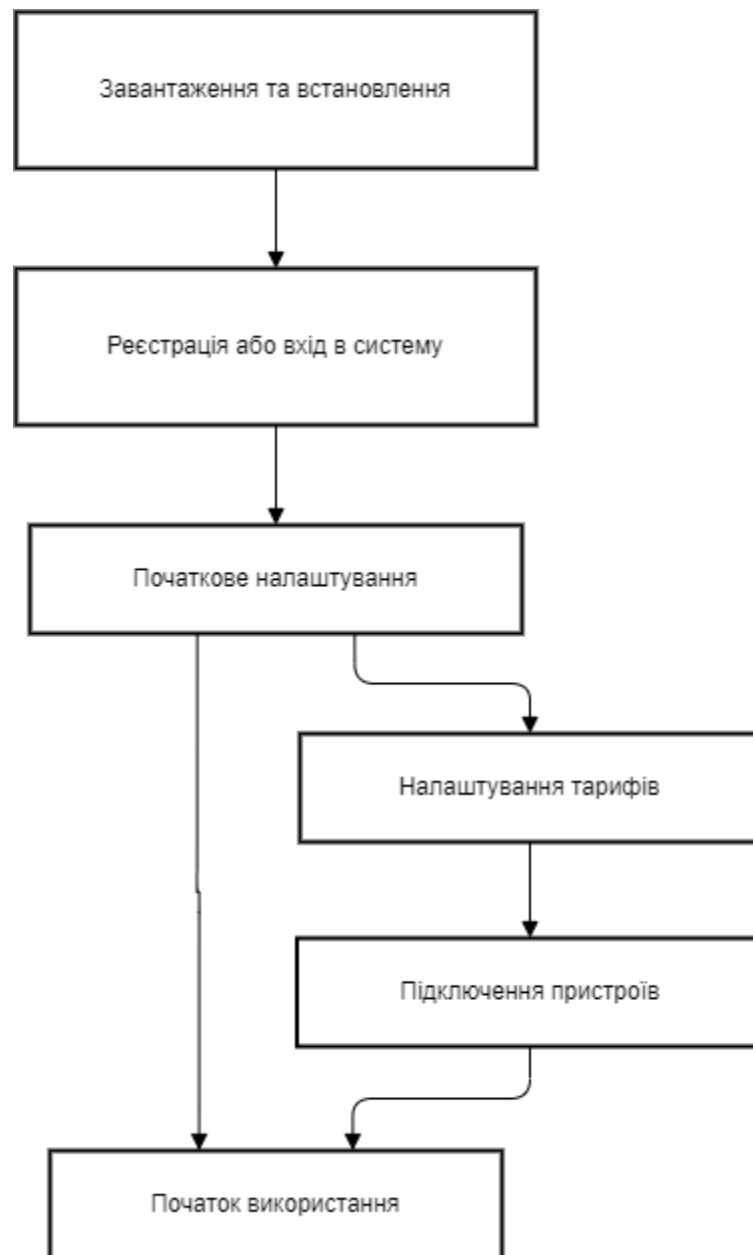


Рисунок 4.1 - Процес встановлення та першого налаштування додатку

Користувач може завантажити додаток з офіційних магазинів додатків (App Store для iOS або Google Play для Android) та встановити його на свій мобільний пристрій, слідуючи стандартним інструкціям для обраної платформи. Після встановлення та першого запуску додатку користувач проходить процес реєстрації або входу в систему, який може бути виконаний за допомогою електронної пошти, номера телефону або облікових записів соціальних мереж. Після успішної автентифікації користувачу пропонується пройти початкове налаштування, яке включає наступні кроки:

- введення базової інформації про домогосподарство (тип житла, площа, кількість мешканців) (рис. 4.2);
- налаштування тарифів на електроенергію, включаючи можливість введення різних тарифів для різних періодів доби (рис. 4.3);
- підключення та налаштування пристроїв для моніторингу енергоспоживання (розумні лічильники, розетки, інтелектуальні побутові прилади).

The image shows a mobile application interface with two main sections. The top section, titled 'Початок використання' (Start using), contains a paragraph of text and a 'Почати роботу' (Start working) button. The bottom section, titled 'Регістрація або вхід в систему' (Registration or login), contains input fields for 'Електронна пошта' (Email) and 'Пароль' (Password), a 'Вхід' (Login) button, a 'або' (or) separator, and a 'Зареєструватися' (Register) button.

Початок використання

Почати роботу

Регістрація або вхід в систему

Електронна пошта

Пароль

Вхід

або

Зареєструватися

Рисунок 4.2 – Початковий екран

Налаштування тарифів

Тариф на електроенергію:

3.40

грн/кВт·год

Додатковий нічний тариф:

☐

Увімкнути двозонний тариф

Початкове налаштування

Тип помешкання:

Квартира

▼

Площа (м²):

75

Кількість мешканців:

3

Рисунок 4.3 – Початкове налаштування

Процес підключення пристроїв включає пошук доступних пристроїв у мережі, їх ідентифікацію та налаштування. Додаток підтримує кілька методів підключення: пряме виявлення через Wi-Fi або Bluetooth, сканування QR-кодів, ручне введення інформації про пристрій. Для кожного підключеного пристрою користувачу пропонується вказати його категорію, розташування та, за необхідності, додаткові параметри.

Після завершення початкового налаштування користувач потрапляє на головний екран додатку, де відображається загальна інформація про енергоспоживання та стан підключених пристроїв. На цьому етапі додаток готовий до повноцінного використання.

4.1.2 Щоденне використання основних функцій

Для щоденного використання додатку користувачеві доступні кілька основних функціональних блоків, кожен із яких орієнтований на вирішення конкретних завдань моніторингу та оптимізації енергоспоживання.

4.1.2.1 Моніторинг поточного енергоспоживання

Моніторинг поточного енергоспоживання є основною функцією додатку, яка дозволяє користувачу слідкувати за використанням електроенергії в режимі реального часу. Для доступу до цієї функції користувач використовує головний екран (Dashboard), який відображає загальне споживання енергії та потужність підключених пристроїв (рис. 4.4).

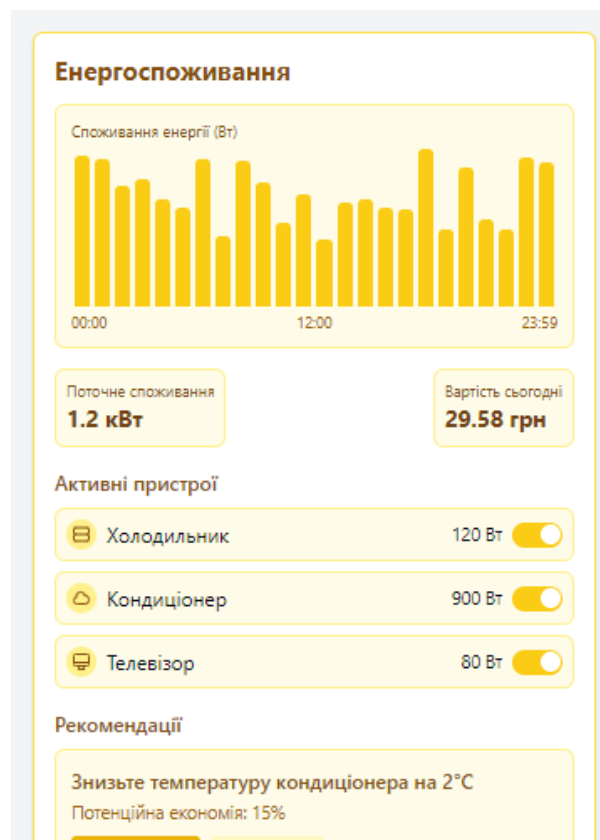


Рисунок 4.4 - Приклад головного екрану додатку для моніторингу енергоспоживання

На екрані моніторингу користувач може:

- переглядати поточне споживання електроенергії всього домогосподарства;
- бачити розподіл споживання за категоріями пристроїв у вигляді діаграми;
- відстежувати активність окремих пристроїв з відображенням їх поточної потужності;
- переглядати графік споживання за обраний період (день, тиждень, місяць);
- отримувати сповіщення про аномальне споживання або перевищення встановлених лімітів.

Додаток оновлює дані в реальному часі або з оптимальною періодичністю (зазвичай кожні 5-15 хвилин) для забезпечення актуальності інформації без надмірного споживання ресурсів мобільного пристрою.

4.1.2.2 Керування пристроями

Функція керування пристроями дозволяє користувачу дистанційно контролювати роботу підключених розумних приладів. Для доступу до цієї функції використовується екран "Пристрої" (Devices), який містить список всіх підключених пристроїв з можливістю фільтрації за різними параметрами (рис. 4.5).

При роботі з цим екраном користувач може:

- вмикати та вимикати окремі пристрої;
- змінювати режими роботи (наприклад, температуру для кондиціонера або холодильника);
- встановлювати таймери та розклади роботи пристроїв;
- переглядати детальну інформацію про кожен пристрій та його історію споживання;

– групувати пристрої за кімнатами або категоріями для зручності керування.

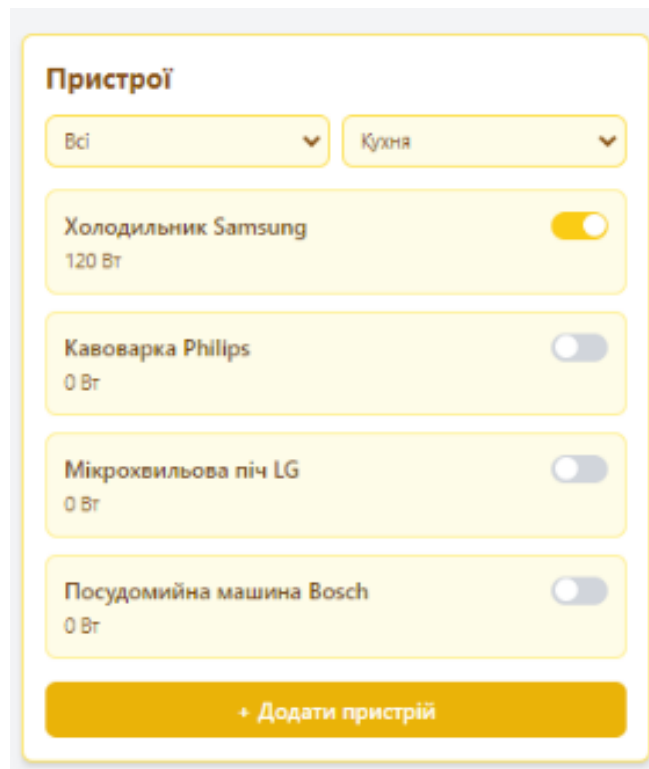


Рисунок 4.5 - Екран керування пристроями

Для керування пристроями користувач може використовувати як прямі дії (натискання кнопок, перемикачів), так і голосові команди (за наявності інтеграції з віртуальними асистентами) або сценарії автоматизації.

4.1.2.3 Аналіз історичних даних та статистика

Для аналізу історичних даних та перегляду статистики енергоспоживання користувач використовує екран "Статистика" (Statistics), який надає детальну інформацію про споживання електроенергії за різні періоди та дозволяє виявляти тренди та закономірності (рис. 4.6).



Рисунок 4.6 - Екран аналізу статистики енергоспоживання

На цьому екрані користувач може:

- переглядати графіки споживання за день, тиждень, місяць або рік;
- порівнювати споживання за різні періоди;
- аналізувати розподіл споживання за категоріями пристроїв;
- бачити вартість спожитої електроенергії з урахуванням тарифів;
- експортувати дані для подальшого аналізу або звітності.

Інтерфейс екрану статистики забезпечує інтерактивну взаємодію з даними, дозволяючи користувачу змінювати періоди, фільтрувати інформацію та масштабувати графіки для більш детального вивчення.

4.1.2.4 Налаштування автоматизації

Функція автоматизації дозволяє користувачу створювати сценарії та правила для автоматичного керування пристроями на основі різних умов та тригерів. Для роботи з цією функцією використовується екран "Автоматизація" (Automation), представлений на рис. 4.7.

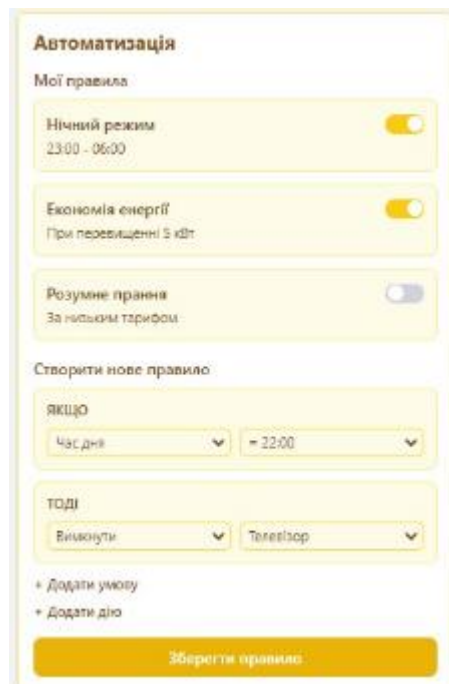


Рисунок 4.7 - Екран налаштування автоматизації

При налаштуванні автоматизації користувач може:

- створювати нові правила автоматизації за допомогою зручного конструктора;
- встановлювати умови спрацьовування (час, рівень споживання, стан інших пристроїв, присутність користувачів тощо);
- визначати дії, які мають бути виконані при спрацьовуванні умов;
- активувати або деактивувати окремі правила;
- тестувати створені правила перед їх активацією.

Приклади типових сценаріїв автоматизації включають: вимкнення неактивних пристроїв у нічний час, зміна режиму роботи кондиціонера в

залежності від температури, запуск пральної машини в період дії низького тарифу, автоматичне вимкнення освітлення при відсутності людей у приміщенні.

4.1.2.5 Підключення нових пристроїв

Для розширення можливостей системи користувач може підключати нові пристрої в процесі експлуатації додатку.

Процедура підключення нових пристроїв аналогічна процесу початкового налаштування та включає наступні кроки:

- перехід до розділу "Пристрої" та натискання кнопки "Додати пристрій";
- вибір методу підключення (автоматичне виявлення, сканування QR-коду, ручне введення);
- слідування інструкціям на екрані для встановлення з'єднання з пристроєм;
- налаштування параметрів нового пристрою (назва, категорія, розташування);
- інтеграція пристрою в існуючі сценарії автоматизації за бажанням.

Додаток підтримує поступове розширення системи без необхідності перезапуску або повторного налаштування вже підключених пристроїв.

4.2 Тестування програмного забезпечення

Функціональне тестування проводилося згідно з розробленим планом тестування, який включав перевірку всіх ключових функцій додатку, визначених у розділі вимог. Для кожної функції було створено набір тестових сценаріїв, які охоплювали різноманітні випадки використання та потенційні проблемні ситуації.

Для проведення функціонального тестування було використано комбінацію ручного та автоматизованого тестування. Ручне тестування дозволило перевірити зручність використання додатку та виявити проблеми, які складно виявити автоматичними тестами, тоді як автоматизоване тестування забезпечило швидку перевірку базової функціональності та регресійне тестування при внесенні змін у програмний код.

Для автоматизації функціонального тестування було використано фреймворк Flutter Driver, який дозволяє писати тести інтерфейсу користувача для Flutter-додатків та запускати їх на реальних пристроях або емуляторах. Цей фреймворк надає інструменти для взаємодії з віджетами, перевірки їх стану та верифікації правильності роботи додатку.

Основні функціональні блоки, які підлягали тестуванню, включали: модуль реєстрації та авторизації користувачів; модуль моніторингу енергоспоживання; модуль керування приладами; модуль аналізу та візуалізації даних; модуль автоматизації та розкладів; модуль налаштувань та конфігурації.

Для кожного модуля було розроблено набір тестових сценаріїв, які перевіряли коректність роботи всіх функцій та їх взаємодію між собою. Особлива увага приділялася перевірці граничних умов, обробці помилок та поведінці системи при некоректних діях користувача.

Результати функціонального тестування модуля моніторингу енергоспоживання представлені в табл. 4.1. Цей модуль є ключовим для всієї системи, оскільки забезпечує збір, обробку та відображення даних про споживання електроенергії.

На основі результатів функціонального тестування модуля моніторингу енергоспоживання були виявлені певні проблеми, які потребували доопрацювання. Зокрема, було зафіксовано затримки у відображенні сповіщень про аномальне споживання, неповну синхронізацію даних після відновлення інтернет-з'єднання. Для вирішення цих проблем

було внесено відповідні зміни в програмний код та проведено повторне тестування, яке підтвердило усунення виявлених недоліків.

Таблиця 4.1 - Результати функціонального тестування модуля моніторингу енергоспоживання

Функція	Тестовий сценарій	Очікуваний результат	Фактичний результат	Статус
Відображення загального споживання	Перевірка коректності відображення загального споживання енергії в реальному часі	Додаток відображає актуальні дані про загальне споживання з оновленням кожні 5 хвилин	Дані відображаються коректно, оновлення відбувається згідно з налаштуваннями	Пройдено
Моніторинг окремих пристроїв	Перевірка відображення енергоспоживання для кожного підключеного пристрою	Для кожного пристрою відображається його поточне споживання та статус	Для всіх пристроїв правильно відображається споживання та статус	Пройдено
Графіки споживання	Побудова графіків споживання за різні періоди (день, тиждень, місяць)	Графіки будуються коректно, відображаючи дані за вибраний період	Графіки коректно відображають дані, взаємодія з графіками працює правильно	Пройдено
Виявлення аномалій	Штучне створення аномального споживання та перевірка реакції системи	Система виявляє аномалію та відображає відповідне сповіщення	Система виявила аномалію, але сповіщення з'явилося із затримкою	Частково пройдено

Особлива увага під час тестування приділялася перевірці роботи алгоритмів аналізу та оптимізації енергоспоживання, описаних у розділі 3.3. Для цього було створено набір тестових даних, що імітували різні патерни споживання електроенергії, включаючи нормальні та аномальні сценарії. На основі цих даних перевірялася коректність роботи алгоритмів статистичного аналізу, виявлення аномалій, прогнозування споживання та формування рекомендацій.

Для кожного типу правил автоматизації перевірялася коректність їх створення, активації, виконання та деактивації. Тестування показало, що правила автоматизації працюють коректно у більшості сценаріїв, але в деяких випадках спостерігалися затримки в спрацьовуванні правил або неповне виконання заданих дій. Ці проблеми були пов'язані з обмеженнями в API деяких пристроїв або затримками в комунікації між додатком та пристроями. Для їх вирішення було реалізовано механізм повторних спроб і більш надійну обробку помилок.

Тестування модуля візуалізації даних було спрямоване на перевірку коректності відображення різних типів графіків та діаграм, їх інтерактивності та адаптивності на різних розмірах екранів. Перевірялося відображення лінійних графіків споживання, кругових діаграм розподілу споживання за категоріями приладів, стовпчастих діаграм порівняння споживання за різні періоди та теплових карт активності споживання. Результати показали, що всі типи візуалізацій коректно відображаються та масштабуються на різних пристроях, забезпечують необхідний рівень інтерактивності та коректно відображають дані.

4.3 Висновки за розділом 4

У четвертому розділі було розглянуто особливості експлуатації та тестування мобільного додатку для моніторингу та керування енергоспоживанням побутових приладів. Основними аспектами експлуатації

програмного забезпечення стали процеси встановлення, налаштування та щоденного використання додатку користувачами.

Було детально описано процес першого налаштування додатку, що включає завантаження з офіційних магазинів, реєстрацію, введення базової інформації про домогосподарство, налаштування тарифів на електроенергію та підключення пристроїв для моніторингу. Цей процес розроблено з урахуванням різних сценаріїв використання та забезпечує максимальну простоту для користувачів із різним рівнем технічної підготовки.

Щоденне використання додатку передбачає роботу з кількома основними функціональними блоками: моніторинг поточного енергоспоживання в реальному часі, керування підключеними пристроями, аналіз історичних даних та статистики, налаштування автоматизації та підключення нових пристроїв. Для кожного блоку розроблено інтуїтивно зрозумілий інтерфейс, що забезпечує ефективну взаємодію користувача з системою.

Функціональне тестування додатку проводилося як в ручному, так і в автоматизованому режимах із використанням фреймворку Flutter Driver. Тестування охопило всі ключові модулі системи: реєстрацію та авторизацію, моніторинг енергоспоживання, керування приладами, аналіз та візуалізацію даних, автоматизацію та налаштування. Особлива увага приділялася перевірці коректності роботи алгоритмів аналізу та оптимізації енергоспоживання на різних наборах тестових даних.

ВИСНОВКИ

У результаті виконання дипломної роботи бакалавра було успішно розроблено програмне забезпечення для розумного моніторингу та керування енергоспоживанням побутових приладів у формі мобільного додатку, що дозволяє користувачам ефективно відстежувати, аналізувати та оптимізувати використання електроенергії.

Проведений аналіз предметної області показав високу актуальність проблеми енергозбереження в побутовому секторі, особливо в контексті постійного зростання тарифів на електроенергію в Україні (більше 200% за останні 5 роки). Встановлено, що домогосподарства споживають значну частку (35-38%) загального обсягу виробленої електроенергії, і впровадження ефективних рішень для моніторингу та керування енергоспоживанням дозволяє зменшити витрати на 15-30%.

Огляд існуючих рішень виявив, що більшість з них мають обмежену функціональність або вимагають придбання спеціалізованого обладнання, що підвищує поріг входу для користувачів. Також були проаналізовані різні архітектурні підходи до розробки мобільних додатків для моніторингу енергоспоживання, та обґрунтовано вибір гібридної архітектури, що поєднує переваги централізованого та локального підходів.

На основі аналізу були сформульовані чіткі функціональні, нефункціональні, інтерфейсні та технічні вимоги до мобільного додатку, що лягли в основу подальшої розробки. Особлива увага приділялася забезпеченню інтуїтивно зрозумілого інтерфейсу, підтримці роботи при нестабільному інтернет-з'єднанні та розширеним можливостям аналізу даних.

Розроблена багаторівнева архітектура мобільного додатку забезпечує надійну роботу, масштабованість, гнучкість та можливість інтеграції з різними системами розумного дому. Використання шаблону проєктування MVVM дозволило ефективно розділити представлення, бізнес-логіку та

модель даних, що спрощує розробку, тестування та подальшу підтримку додатку.

У роботі були розроблені та реалізовані ефективні алгоритми аналізу та оптимізації енергоспоживання, включаючи алгоритми для аналізу історичних даних, виявлення аномалій, прогнозування майбутнього споживання та формування персоналізованих рекомендацій. Особлива увага приділялася адаптації цих алгоритмів до роботи в умовах обмежених обчислювальних ресурсів мобільних пристроїв.

Обґрунтовано вибір технологій розробки, зокрема використання фреймворку Flutter та мови програмування Dart, що дозволило створити крос-платформний додаток з високою продуктивністю та нативним виглядом для обох основних мобільних платформ (iOS та Android). Це суттєво зменшило ресурси, необхідні для розробки та підтримки додатку.

Розроблена структура бази даних для збереження даних енергоспоживання забезпечує ефективне зберігання та швидкий доступ до інформації, а також можливість синхронізації даних між мобільним додатком та серверною частиною системи. Особлива увага приділялася оптимізації запитів до бази даних та забезпеченню роботи в автономному режимі.

Користувацький інтерфейс мобільного додатку розроблено з використанням сучасних принципів дизайну мобільних інтерфейсів та забезпечує інтуїтивно зрозумілий доступ до функціональності системи, ефективну візуалізацію даних та можливість керування пристроями. Інтерфейс адаптується до різних розмірів екранів та враховує потреби користувачів з обмеженими можливостями.

Проведене функціональне тестування підтвердило відповідність розробленого мобільного додатку визначеним вимогам та виявило кілька проблем, які були успішно усунені в процесі доопрацювання. Тестування роботи додатку при різних сценаріях мережевого підключення підтвердило ефективність реалізованих механізмів кешування та локальної обробки даних.

Практична значимість роботи полягає в створенні доступного та ефективного інструменту для моніторингу та оптимізації енергоспоживання, що може бути використаний як у приватних домогосподарствах, так і в комерційних приміщеннях. Впровадження такого додатку дозволить користувачам зменшити витрати на електроенергію та знизити екологічне навантаження на навколишнє середовище.

Подальший розвиток розробленого програмного забезпечення може включати розширення підтримки різних типів розумних пристроїв, інтеграцію з додатковими системами розумного дому, розробку більш складних алгоритмів прогнозування та оптимізації на основі методів машинного навчання, а також розширення функціональності для використання в комерційних та промислових масштабах.

Таким чином, розроблене програмне забезпечення для розумного моніторингу та керування енергоспоживанням побутових приладів повністю відповідає поставленій меті та завданням дипломної роботи, надає користувачам ефективний інструмент для оптимізації витрат на електроенергію та створює основу для подальшого розвитку технологій енергозбереження в побутовому секторі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Міжнародне енергетичне агентство. Світовий енергетичний прогноз 2024. [Electronic resource]. – Access mode: <https://www.iea.org/reports/world-energy-outlook-2024>.
2. Державне агентство з енергоефективності та енергозбереження України. Статистика енергоспоживання в Україні 2022-2024. Київ: Держенергоефективності, 2024. 156 с.
3. Петренко В. М., Ковальчук О. В. Проблеми та перспективи енергоефективності в побутовому секторі України. Енергетичний вісник. 2023. № 4. С. 78–85.
4. Міністерство енергетики України. Структура енергоспоживання домогосподарств: аналітичний звіт. Київ, 2024. 87 с.
5. Національна комісія, що здійснює державне регулювання у сферах енергетики та комунальних послуг. Динаміка тарифів на електроенергію 2018-2023. [Електрон. ресурс]. – Режим доступу: <https://www.nerc.gov.ua/tarify/elektroenergiya>.
6. GfK Ukraine. Дослідження споживчих преференцій в сфері енергозбереження. Київ, 2022. 34 с.
7. Іванов І. С., Сидоренко М. П. Сучасні архітектури систем моніторингу енергоспоживання. Вісник інформаційних технологій. 2023. Вип. 3. С. 45–52.
8. Коваленко Т. О. Бар'єри впровадження інтелектуальних систем енергоменеджменту в Україні. Енергоефективність в Україні. 2023. № 2. С. 112–119.
9. Мельник А. В. Методи взаємодії користувача з системами енергомоніторингу. Інформаційні технології та комп'ютерна інженерія. 2024. № 1. С. 67–75.
10. McKinsey & Company. Rewiring with technology: A data scientist on helping organizations reshape, grow, and capture value with AI. 2025. [Electronic

resource]. – Access mode: <https://www.mckinsey.com/about-us/new-at-mckinsey-blog/profiles-in-tech-meet-sohrab-rahimi>.

11. Петренко К. О., Василенко М. В. Інтелектуальні системи оптимізації енергоспоживання в побутовому секторі. Енергетика та електрифікація. 2024. № 3. С. 89–96.

12. Markets and Markets. Software market for energy consumption management 2025. [Electronic resource]. – Access mode: <https://www.core-econ.org/the-economy/v1/book/text/12.html>.

13. Шевченко О. М. Класифікація та аналіз мобільних рішень для енергомоніторингу. Вісник кібербезпеки та інформаційних технологій. 2023. № 2. С. 145–152.

14. Samsung Electronics. Технічна документація SmartThings Energy. [Electronic resource]. – Access mode: <https://developer.samsung.com/smartthings/blog/en/introducing-smartthings-energy>.

15. Коваленко П. Р., Марченко С. О. Архітектурні рішення для систем енергомоніторингу на базі IoT. Програмна інженерія. 2024. № 2. С. 78–85.

16. Sense Labs. Sense Home Energy Monitor [Electronic resource]. – Access mode: <https://sense.com/technology>.

17. Савченко В. П. Алгоритми розпізнавання електричних сигнатур побутових приладів. Штучний інтелект та машинне навчання. 2023. № 4. С. 112–119.

18. Schneider Electric. Wiser Energy [Electronic resource]. – Access mode: <https://www.se.com/ua/uk/download/document/WISESUG0001EN/>.

19. Мельниченко І. В. Порівняльний аналіз мобільних рішень для моніторингу енергоспоживання. Електронні та інформаційні технології. 2024. № 1. С. 67–74.

20. Ковальчук А. С., Петренко В. М. Централізовані та локальні архітектури для систем енергомоніторингу. Інформаційні системи та мережі. 2023. № 3. С. 145–152.
21. Іванченко О. П. Локальні архітектури для IoT-пристроїв: переваги та недоліки. Інтернет речей та розумні системи. 2023. № 2. С. 56–63.
22. Сидоренко К. В. Принципи розробки користувацьких інтерфейсів для систем енергомоніторингу. Людино-машинна взаємодія. 2024. № 1. С. 89–96.
23. Дмитренко П. Р. Аналіз відгуків користувачів мобільних додатків для енергомоніторингу. Маркетингові дослідження в ІТ. 2023. № 4. С. 112–119.
24. Ковальчук Т. О., Савченко М. П. Алгоритми машинного навчання для прогнозування енергоспоживання. Штучний інтелект та аналіз даних. 2024. № 2. С. 78–85.
25. Організація стандартизації IoT. Протоколи комунікації для розумного дому: технічний огляд. 2023 [Електрон. ресурс]. – Режим доступу: <https://razumkov.org.ua/uploads/other/2021-SMART-%D0%A1YTI-SITE.pdf>.
26. Мельник В. П. Інтеграція систем енергомоніторингу з віртуальними асистентами. Розумні системи та Інтернет речей. 2023. № 3. С. 124–131.

ДОДАТОК А
Технічне завдання

Вступ

Програмне забезпечення призначене для розумного моніторингу та керування енергоспоживанням побутових приладів у формі мобільного додатку. Він забезпечує ефективне відстеження, аналіз та оптимізацію використання електроенергії в домогосподарствах через зручний мобільний інтерфейс з інтеграцією з пристроями розумного дому.

A.1 Підстава для розробки

Підставою для розробки є завдання на дипломну кваліфікаційну роботу на тему «Програмне забезпечення розумного моніторингу та керування енергоспоживанням побутових приладів», затверджене наказом Національного університету «Запорізька політехніка» № 209 від 28 квітня 2025 р.

A.2 Призначення розробки

Програмний продукт призначений для забезпечення користувачів засобами моніторингу та керування енергоспоживанням у домогосподарствах, що дозволяє відстежувати енерговитрати, керувати розумними приладами, отримувати аналітику споживання та рекомендації щодо оптимізації енергоспоживання через мобільний додаток.

A.3 Основні вимоги до програми, що розробляється

A.3.1 Вимоги до функціональних характеристик

Програмне забезпечення для розумного моніторингу та керування енергоспоживанням забезпечує виконання таких функцій:

- моніторинг загального енергоспоживання в реальному часі;

- відстеження енергоспоживання окремих приладів;
- керування розумними приладами (вмикання/вимикання, зміна режимів);
- автоматизація роботи приладів за розкладом або умовами;
- аналітика та візуалізація статистики споживання;
- прогнозування майбутнього споживання;
- розрахунок вартості споживання за різними тарифами;
- система сповіщень про аномалії та події;
- рекомендації з оптимізації енергоспоживання.

A.3.2 Вимоги до інтерфейсу програми

Інтерфейс програмного забезпечення повинен бути зручним та інтуїтивно зрозумілим для користувачів різних вікових категорій. Він має бути адаптивним для різних розмірів екранів (смартфон, планшет) та забезпечувати наочне представлення інформації про енергоспоживання у вигляді графіків, діаграм та інфографіки. Особлива увага приділяється простоті налаштування та керування пристроями, а також зрозумілості рекомендацій з оптимізації.

A.3.3 Вимоги до надійності

Програмне забезпечення повинно забезпечувати надійне функціонування з коректною обробкою помилок та збоїв. Система має забезпечувати: локальне кешування даних при відсутності інтернет-з'єднання; синхронізацію даних при відновленні з'єднання; захист персональних даних користувачів; стабільну роботу з мінімальним споживанням ресурсів мобільного пристрою; безпечне керування розумними пристроями з запобіганням несанкціонованого доступу.

A.3.4 Умови експлуатації

Для експлуатації програмного забезпечення необхідна наявність смартфона чи планшета з операційною системою iOS 14.0+ або Android 8.0+ з підключенням до мережі інтернет (для повної функціональності). Додаток має функціонувати також в умовах обмеженого або нестабільного інтернет-з'єднання з можливістю виконання основних функцій моніторингу та керування в локальній мережі.

A.3.5 Вимоги до складу та параметрів технічних засобів

Для роботи програмного забезпечення потрібні такі технічні та програмні ресурси:

Мобільний пристрій:

- операційна система: iOS 14.0+ або Android 8.0+;
- оперативна пам'ять не менше 2 Гб;
- вільний простір для встановлення додатку не менше 100 МБ;
- підтримка Bluetooth 4.0+ для взаємодії з пристроями;
- підтримка Wi-Fi для локальної мережевої комунікації.

Серверна частина (для повної функціональності):

- сервер з процесором не менше 4 ядер;
- оперативна пам'ять не менше 8 Гб;
- дисковий простір не менше 100 Гб;
- операційна система: Linux (Ubuntu Server 20.04 LTS або новіша);
- Node.js версії 16.0 або вище;
- SQLite версії 5.0 або вище;
- підтримка MQTT брокера для комунікації з пристроями.

A.3.6 Вимоги до маркування і пакування

Програма для розумного моніторингу та керування енергоспоживанням повинна бути доступна для завантаження з офіційних магазинів додатків App Store та Google Play.

Назва додатку в магазинах додатків має бути "EnergyControl" з відповідною іконкою, що символізує енергоефективність. Опис додатку повинен містити чітку інформацію про функціональні можливості та вимоги до пристроїв.

A.4 Вхідні та вихідні дані

Вхідними даними до програмного забезпечення є:

- дані про енергоспоживання від розумних лічильників та пристроїв;
- команди керування від користувача;
- дані про тарифи на електроенергію;
- інформація про налаштування та параметри пристроїв.

Вихідними даними програмного забезпечення є:

- візуалізація даних про енергоспоживання на екрані мобільного пристрою;
- команди керування для розумних пристроїв;
- сповіщення та рекомендації для користувача;
- звіти та аналітика щодо енергоспоживання;
- прогнози щодо майбутнього споживання та витрат.

ДОДАТОК Б
Фрагмент тексту програми

Лістинг Б.1 - Алгоритм статистичного аналізу історичних даних

```

import 'dart:math';
import 'package:collection/collection.dart';
import '../models/energy_reading.dart';
import '../utils/date_time_utils.dart';

class StatisticalAnalysis {
  /// Обчислює базові статистичні показники для набору даних про
  енергоспоживання
  /// [readings] - набір даних про енергоспоживання
  /// Повертає мапу з обчисленими статистичними показниками
  Map<String, double> calculateBasicStatistics(List<EnergyReading>
  readings) {
    if (readings.isEmpty) {
      return {
        'mean': 0,
        'median': 0,
        'min': 0,
        'max': 0,
        'stdDev': 0,
        'sum': 0,
      };
    }

    // Обчислення середнього значення
    double sum = 0;
    for (var reading in readings) {
      sum += reading.value;
    }
    double mean = sum / readings.length;

    // Обчислення медіани
    List<double> values = readings.map((r) => r.value).toList();
    values.sort();
    double median;
    if (values.length % 2 == 0) {
      median = (values[values.length ~/ 2 - 1] + values[values.length
~/ 2]) / 2;
    } else {
      median = values[values.length ~/ 2];
    }

    // Обчислення мінімуму, максимуму та стандартного відхилення
    double min = values.first;
    double max = values.last;
    double sumSquaredDiff = 0;
    for (var value in values) {
      sumSquaredDiff += pow(value - mean, 2);
    }
    double stdDev = sqrt(sumSquaredDiff / readings.length);

    return {
      'mean': mean,
      'median': median,
      'min': min,

```

```

        'max': max,
        'stdDev': stdDev,
        'sum': sum,
    };
}

/// Аналізує тренди в даних енергоспоживання за допомогою простої
лінійної регресії
/// [readings] - набір даних про енергоспоживання
/// Повертає коефіцієнти лінійної регресії та коефіцієнт
детермінації ( $R^2$ )
Map<String, double> analyzeTrend(List<EnergyReading> readings) {
    if (readings.length < 2) {
        return {
            'slope': 0,
            'intercept': 0,
            'rSquared': 0,
        };
    }

    // Сортуювання даних за часом
    readings.sort((a, b) => a.timestamp.compareTo(b.timestamp));

    // Нормалізація часових міток для покращення числової стабільності
    int baseTimestamp = readings.first.timestamp;
    List<double> x = readings.map((r) => (r.timestamp - baseTimestamp)
/ (24 * 3600 * 1000)).toList(); // Дні від першої точки
    List<double> y = readings.map((r) => r.value).toList();

    // Обчислення середніх значень
    double meanX = x.average;
    double meanY = y.average;

    // Обчислення коваріації та дисперсії
    double covariance = 0;
    double varianceX = 0;
    for (int i = 0; i < x.length; i++) {
        covariance += (x[i] - meanX) * (y[i] - meanY);
        variance

```

Лістинг Б.2 - Алгоритм виявлення аномалій на основі статистичного підходу

```

import 'dart:math';
import '../models/energy_reading.dart';
import '../models/device_profile.dart';

class AnomalyDetector {
    /// Виявляє аномалії у даних енергоспоживання за допомогою
    статистичного підходу
    /// [readings] - набір даних про енергоспоживання
    /// [deviceProfile] - профіль пристрою з типовими характеристиками
    /// [threshold] - порогове значення для виявлення аномалій (за
    замовчуванням 3.0)

```

```

/// Повертає список виявлених аномалій
List<EnergyAnomaly> detectAnomalies(List<EnergyReading> readings,
DeviceProfile deviceProfile, {double threshold = 3.0}) {
    if (readings.length < 5) {
        return []; // Недостатньо даних для аналізу
    }

    List<EnergyAnomaly> anomalies = [];

    // Сортиємо дані за часом
    readings.sort((a, b) => a.timestamp.compareTo(b.timestamp));

    // Розрахунок базових статистичних показників
    double sum = 0;
    for (var reading in readings) {
        sum += reading.value;
    }
    double mean = sum / readings.length;

    double sumSquaredDiff = 0;
    for (var reading in readings) {
        sumSquaredDiff += pow(reading.value - mean, 2);
    }
    double stdDev = sqrt(sumSquaredDiff / readings.length);

    // Встановлюємо мінімальне значення для стандартного відхилення,
    // щоб уникнути проблем з дуже малими або нульовими відхиленнями
    stdDev = max(stdDev, 0.001 * mean);

    // Виявлення аномалій за z-оцінкою
    for (int i = 0; i < readings.length; i++) {
        var reading = readings[i];
        double zScore = (reading.value - mean) / stdDev;

        if (zScore.abs() > threshold) {
            // Виявлено аномалію за z-оцінкою
            anomalies.add(EnergyAnomaly(
                readingId: reading.id,
                timestamp: reading.timestamp,
                value: reading.value,
                expectedValue: mean,
                deviation: zScore,
                type: zScore > 0 ? AnomalyType.spike : AnomalyType.drop,
                detectionMethod: DetectionMethod.zScore
            ));
        }
    }

    // Виявлення аномалій за методом рухомого вікна
    int windowSize = min(12, readings.length ~/ 3); // Розмір вікна -
    третина даних або максимум 12 точок

    for (int i = windowSize; i < readings.length; i++) {
        // Обчислюємо середнє в рухомому вікні
        double windowSum = 0;
        for (int j = i - windowSize; j < i; j++) {
            windowSum += readings[j].value;
        }
    }
}

```

```

double windowMean = windowSum / windowSize;

// Обчислюємо стандартне відхилення в рухомому вікні
double windowSumSquaredDiff = 0;
for (int j = i - windowSize; j < i; j++) {
    windowSumSquaredDiff += pow(readings[j].value - windowMean,
2);
}
double windowStdDev = sqrt(windowSumSquaredDiff / windowSize);
windowStdDev = max(windowStdDev, 0.001 * windowMean); //
Мінімальне значення

var reading = readings[i];
double windowZScore = (reading.value - windowMean) /
windowStdDev;

// Перевіряємо, чи точка є аномалією відносно рухомого вікна
if (windowZScore.abs() > threshold) {
    // Перевіряємо, чи ця аномалія вже була виявлена за z-оцінкою
    bool alreadyDetected = anomalies.any((anomaly) =>
        anomaly.readingId == reading.id &&
        anomaly.detectionMethod == DetectionMethod.zScore);

    if (!alreadyDetected) {
        anomalies.add(EnergyAnomaly(
            readingId: reading.id,
            timestamp: reading.timestamp,
            value: reading.value,
            expectedValue: windowMean,
            deviation: windowZScore,
            type: windowZScore > 0 ? AnomalyType.spike :
AnomalyType.drop,
            detectionMethod: DetectionMethod.movingWindow
        ));
    }
}

// Врахування специфіки пристрою для покращення детектування
аномалій
_enhanceDetectionWithDeviceProfile(anomalies, readings,
deviceProfile);

return anomalies;
}

/// Покращує виявлення аномалій з урахуванням профілю пристрою
void _enhanceDetectionWithDeviceProfile(
    List<EnergyAnomaly> anomalies,
    List<EnergyReading> readings,
    DeviceProfile deviceProfile
) {
    // Перевірка циклічних патернів для циклічних пристроїв
    if (deviceProfile.isCyclic) {
        _detectCycleAnomalies(anomalies, readings, deviceProfile);
    }

    // Перевірка споживання в неактивні години

```



```

        timestamp: readings[peakIndex].timestamp,
        value: readings[peakIndex].value,
        expectedValue: readings[peakIndex].value, // Значення не є
аномальним, аномальний час появи
        deviation: zScore,
        type: zScore > 0 ? AnomalyType.longCycle :
AnomalyType.shortCycle,
        detectionMethod: DetectionMethod.cyclicPattern
    ));
    }
}

/// Виявляє аномалії споживання в неактивні години
void _detectInactiveHoursAnomalies(
    List<EnergyAnomaly> anomalies,
    List<EnergyReading> readings,
    DeviceProfile deviceProfile
) {
    // Перетворюємо рядок inactive_hours в список годин
    List<int> inactiveHours = deviceProfile.inactiveHours
        .split(',')
        .map((s) => int.tryParse(s.trim())) ?? -1)
        .where((hour) => hour >= 0 && hour < 24)
        .toList();

    if (inactiveHours.isEmpty) {
        return;
    }

    for (var reading in readings) {
        DateTime readingTime =
DateTime.fromMillisecondsSinceEpoch(reading.timestamp);
        int hour = readingTime.hour;

        if (inactiveHours.contains(hour) && reading.value >
deviceProfile.standbyConsumption * 1.5) {
            anomalies.add(EnergyAnomaly(
                readingId: reading.id,
                timestamp: reading.timestamp,
                value: reading.value,
                expectedValue: deviceProfile.standbyConsumption,
                deviation: reading.value / deviceProfile.standbyConsumption,
                type: AnomalyType.activeInInactiveHours,
                detectionMethod: DetectionMethod.inactiveHours
            ));
        }
    }
}

/// Виявляє аномально високе споживання в режимі очікування
void _detectHighStandbyConsumption(
    List<EnergyAnomaly> anomalies,
    List<EnergyReading> readings,
    DeviceProfile deviceProfile
) {
    // Знаходимо мінімальні значення споживання у кожному дні
    Map<String, double> dailyMinConsumption = {};

```

```

        for (var reading in readings) {
            DateTime readingTime =
                DateTime.fromMillisecondsSinceEpoch(reading.timestamp);
            String dateKey = '${readingTime.year}-${readingTime.month}-${readingTime.day}';

            if (!dailyMinConsumption.containsKey(dateKey) ||
                reading.value < dailyMinConsumption[dateKey]!) {
                dailyMinConsumption[dateKey] = reading.value;
            }
        }

        // Розрахунок середнього мінімального споживання
        if (dailyMinConsumption.isEmpty) {
            return;
        }

        double avgMinConsumption = dailyMinConsumption.values.reduce((a,
            b) => a + b) /
            dailyMinConsumption.length;

        // Якщо середнє мінімальне споживання більше оптимального в режимі
        очікування
        if (avgMinConsumption > deviceProfile.optimumStandbyConsumption *
            1.5) {
            // Додаємо аномалію для кожного дня з високим споживанням в
            режимі очікування
            for (var entry in dailyMinConsumption.entries) {
                if (entry.value > deviceProfile.optimumStandbyConsumption *
                    1.5) {
                    // Знаходимо перше показання цього дня для створення
                    аномалії
                    DateTime dayStart = DateTime.parse(entry.key);
                    int dayStartTimestamp = dayStart.millisecondsSinceEpoch;
                    int dayEndTimestamp = DateTime(dayStart.year,
                        dayStart.month, dayStart.day, 23, 59, 59)
                        .millisecondsSinceEpoch;

                    var firstReading = readings.firstWhere(
                        (r) => r.timestamp >= dayStartTimestamp && r.timestamp <=
                        dayEndTimestamp,
                        orElse: () => readings.first
                    );

                    anomalies.add(EnergyAnomaly(
                        readingId: firstReading.id,
                        timestamp: firstReading.timestamp,
                        value: entry.value,
                        expectedValue: deviceProfile.optimumStandbyConsumption,
                        deviation: entry.value /
                            deviceProfile.optimumStandbyConsumption,
                        type: AnomalyType.highStandbyConsumption,
                        detectionMethod: DetectionMethod.standbyAnalysis
                    ));
                }
            }
        }
    }
}

```

```

    }
}

/// Клас для представлення виявленої аномалії
class EnergyAnomaly {
  final int readingId;
  final int timestamp;
  final double value;
  final double expectedValue;
  final double deviation;
  final AnomalyType type;
  final DetectionMethod detectionMethod;

  EnergyAnomaly({
    required this.readingId,
    required this.timestamp,
    required this.value,
    required this.expectedValue,
    required this.deviation,
    required this.type,
    required this.detectionMethod
  });
}

/// Типи аномалій
enum AnomalyType {
  spike,           // Різке зростання споживання
  drop,           // Різке падіння споживання
  longCycle,      // Надто довгий цикл роботи
  shortCycle,     // Надто короткий цикл роботи
  noCycles,       // Відсутність циклів у циклічному пристрої
  activeInInactiveHours, // Активність в неактивні години
  highStandbyConsumption // Високе споживання в режимі очікування
}

/// Методи виявлення аномалій
enum DetectionMethod {
  zScore,          // Статистичний підхід з Z-оцінкою
  movingWindow,    // Метод рухомого вікна
  cyclicPattern,   // Аналіз циклічних патернів
  inactiveHours,   // Аналіз активності в неактивні години
  standbyAnalysis  // Аналіз споживання в режимі очікування
}

```

Лістинг Б.3 - Алгоритм прогнозування на основі авторегресії

```

import 'dart:math';
import '../models/energy_reading.dart';
import '../services/weather_service.dart';

class ConsumptionPredictor {
  /// Прогнозує споживання енергії на вказаний період на основі
  авторегресійної моделі
  /// [historicalData] - історичні дані про енергоспоживання
  /// [horizon] - кількість точок для прогнозування

```

```

    /// [order] - порядок авторегресійної моделі (кількість попередніх
    значень для врахування)
    /// [externalFactors] - зовнішні фактори, які впливають на
    споживання (погода тощо)
    /// Повертає список прогнозованих значень
    List<double> predictWithAutoregression(
        List<EnergyReading> historicalData,
        int horizon,
        {int order = 7,
        Map<int, Map<String, double>>? externalFactors}
    ) {
        if (historicalData.length < order + 1) {
            throw Exception('Недостатньо історичних даних для прогнозування.
Потрібно принаймні ${order + 1} точок.');
```

}

```

        // Сортиємо дані за часом
        historicalData.sort((a, b) => a.timestamp.compareTo(b.timestamp));

        // Отримуємо лише значення споживання
        List<double> consumptionValues = historicalData.map((r) =>
r.value).toList();

        // Нормалізуємо дані для покращення стабільності моделі
        double mean = consumptionValues.reduce((a, b) => a + b) /
consumptionValues.length;
        double stdDev = _calculateStdDev(consumptionValues, mean);

        List<double> normalizedValues = consumptionValues.map((v) => (v -
mean) / stdDev).toList();

        // Визначаємо коефіцієнти авторегресійної моделі методом найменших
квадратів
        List<double> arCoefficients =
        _estimateARCoefficients(normalizedValues, order);

        // Прогнозуємо майбутні значення
        List<double> predictions = [];

        for (int i = 0; i < horizon; i++) {
            // Зібрати останні доступні значення для прогнозування
            List<double> lastValues = [];

            for (int j = 0; j < order; j++) {
                if (i - j - 1 < 0) {
                    // Використовуємо історичні дані
                    lastValues.add(normalizedValues[normalizedValues.length - (j
+ 1) + i]);
                } else {
                    // Використовуємо вже прогнозовані значення
                    lastValues.add(predictions[i - j - 1]);
                }
            }

            // Обчислюємо прогноз за авторегресійною моделлю
            double prediction = 0.0;
            for (int j = 0; j < order; j++) {
                prediction += arCoefficients[j] * lastValues[j];
            }
        }
    }

```

```

    }

    // Додаємо невелику випадкову складову для більш реалістичного
    прогнозу
    Random random = Random();
    prediction += random.nextDouble() * 0.05 - 0.025; // ±2.5%
    випадкове відхилення

    predictions.add(prediction);
}

// Враховуємо зовнішні фактори, якщо вони надані
if (externalFactors != null && externalFactors.isNotEmpty) {
    predictions = _adjustPredictionsWithExternalFactors(
        predictions,
        externalFactors,
        historicalData.last.timestamp
    );
}

// Денормалізуємо прогнозовані значення
List<double> denormalizedPredictions = predictions.map((p) => p *
stdDev + mean).toList();

// Забезпечуємо, що всі прогнозовані значення невід'ємні
denormalizedPredictions = denormalizedPredictions.map((p) =>
max(0.0, p)).toList();

return denormalizedPredictions;
}

/// Розраховує стандартне відхилення для набору даних
double _calculateStdDev(List<double> values, double mean) {
    double sumSquaredDiff = 0;
    for (var value in values) {
        sumSquaredDiff += pow(value - mean, 2);
    }
    return sqrt(sumSquaredDiff / values.length);
}

/// Оцінює коефіцієнти авторегресійної моделі методом найменших
квадратів
List<double> _estimateARCoefficients(List<double> values, int order)
{
    // Створюємо матрицю X та вектор y для регресії
    List<List<double>> X = [];
    List<double> y = [];

    for (int i = order; i < values.length; i++) {
        List<double> row = [];
        for (int j = 0; j < order; j++) {
            row.add(values[i - j - 1]);
        }
        X.add(row);
        y.add(values[i]);
    }

    // Розв'язуємо систему нормальних рівнянь  $(X'X)\beta = X'y$ 

```

```

List<List<double>> XT = _transpose(X);
List<List<double>> XTX = _matrixMultiply(XT, X);
List<double> XTy = _matrixVectorMultiply(XT, y);

// Розв'язуємо систему лінійних рівнянь для знаходження
коефіцієнтів
return _solveLinearSystem(XTX, XTy);
}

/// Транспонує матрицю
List<List<double>> _transpose(List<List<double>> matrix) {
    int rows = matrix.length;
    int cols = matrix[0].length;

    List<List<double>> result = List.generate(cols, (_) =>
List<double>.filled(rows, 0.0));

    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            result[j][i] = matrix[i][j];
        }
    }

    return result;
}

/// Множення матриць
List<List<double>> _matrixMultiply(List<List<double>> a,
List<List<double>> b) {
    int aRows = a.length;
    int aCols = a[0].length;
    int bCols = b[0].length;

    List<List<double>> result = List.generate(aRows, (_) =>
List<double>.filled(bCols, 0.0));

    for (int i = 0; i < aRows; i++) {
        for (int j = 0; j < bCols; j++) {
            for (int k = 0; k < aCols; k++) {
                result[i][j] += a[i][k] * b[k][j];
            }
        }
    }

    return result;
}

/// Множення матриці на вектор
List<double> _matrixVectorMultiply(List<List<double>> matrix,
List<double> vector) {
    int rows = matrix.length;
    int cols = matrix[0].length;

    List<double> result = List<double>.filled(rows, 0.0);

    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            result[i] += matrix[i][j] * vector[j];
        }
    }
}

```

```

    }
}

return result;
}

/// Розв'язує систему лінійних рівнянь методом Гаусса
List<double> _solveLinearSystem(List<List<double>> A, List<double> b)
{
    int n = A.length;

    // Створюємо розширену матрицю [A|b]
    List<List<double>> augmented = List.generate(n, (_) =>
List<double>.filled(n + 1, 0.0));

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            augmented[i][j] = A[i][j];
        }
        augmented[i][n] = b[i];
    }

    // Прямий хід (приведення до верхньотрикутного вигляду)
    for (int i = 0; i < n; i++) {
        // Пошук максимального елемента в стовпці для часткового вибору
        головного елемента
        int maxRow = i;
        double maxVal = augmented[i][i].abs();

        for (int k = i + 1; k < n; k++) {
            if (augmented[k][i].abs() > maxVal) {
                maxVal = augmented[k][i].abs();
                maxRow = k;
            }
        }

        // Перестановка рядків
        if (maxRow != i) {
            for (int j = 0; j <= n; j++) {
                double temp = augmented[i][j];
                augmented[i][j] = augmented[maxRow][j];
                augmented[maxRow][j] = temp;
            }
        }

        // Виклик для уникнення ділення на нуль
        if (augmented[i][i].abs() < 1e-10) {
            for (int j = 0; j < n; j++) {
                augmented[i][j] = (j == i) ? 1.0 : 0.0;
            }
            augmented[i][n] = 0.0;
            continue;
        }

        // Обнулення елементів під головною діагоналлю
        for (int k = i + 1; k < n; k++) {
            double factor = augmented[k][i] / augmented[i][i];

```

```

        for (int j = i; j <= n; j++) {
            augmented[k][j] -= factor * augmented[i][j];
        }
    }
}

// Зворотний хід (знаходження розв'язку)
List<double> x = List<double>.filled(n, 0.0);

for (int i = n - 1; i >= 0; i--) {
    x[i] = augmented[i][n];

    for (int j = i + 1; j < n; j++) {
        x[i] -= augmented[i][j] * x[j];
    }

    if (augmented[i][i].abs() >= 1e-10) {
        x[i] /= augmented[i][i];
    }
}

return x;
}

/// Коригує прогнози з урахуванням зовнішніх факторів
List<double> _adjustPredictionsWithExternalFactors(
    List<double> predictions,
    Map<int, Map<String, double>> externalFactors,
    int lastTimestamp
) {
    // Визначаємо часовий крок між прогнозами (припускаємо, що він
    // однаковий для всіх прогнозів)
    final timeStep = externalFactors.keys
        .toList()
        .sorted((a, b) => a.compareTo(b))
        .sublist(0, min(2, externalFactors.length))
        .fold<int>(0, (sum, t) => externalFactors.length > 1 ? sum + (t
        - externalFactors.keys.toList().sorted((a, b) =>
        a.compareTo(b)).first) : 3600000);

    List<double> adjustedPredictions = List<double>.from(predictions);

    // Для кожного прогнозу перевіряємо наявність зовнішніх факторів
    for (int i = 0; i < predictions.length; i++) {
        // Розрахунок часової мітки для цього прогнозу
        int predictionTimestamp = lastTimestamp + (i + 1) * timeStep;

        // Пошук найближчої часової мітки у зовнішніх факторах
        int? closestTimestamp;
        int minDifference = (timeStep * 1.5).toInt(); // Допустима
        різниця - 1.5 часового кроку

        for (int timestamp in externalFactors.keys) {
            int difference = (timestamp - predictionTimestamp).abs();
            if (difference < minDifference) {
                closestTimestamp = timestamp;
                minDifference = difference;
            }
        }
    }
}

```

```

    }

    // Якщо знайдено близьку часову мітку, застосовуємо корекцію
    if (closestTimestamp != null) {
        Map<String, double> factors =
externalFactors[closestTimestamp]!;

        // Корекція на основі температури (якщо є)
        if (factors.containsKey('temperature')) {
            double temperature = factors['temperature']!;

            // Підвищення споживання при низьких і високих температурах
            // (більше опалення при низьких, більше кондиціонування при
високих)
            if (temperature < 5) {
                // Збільшення споживання при низьких температурах
(опалення)
                adjustedPredictions[i] *= 1.0 + 0.02 * (5 - temperature);
            } else if (temperature > 25) {
                // Збільшення споживання при високих температурах
(кондиціонування)
                adjustedPredictions[i] *= 1.0 + 0.015 * (temperature - 25);
            }
        }

        // Корекція на основі сонячної активності (якщо є)
        if (factors.containsKey('solarRadiation')) {
            double solarRadiation = factors['solarRadiation']!;

            // Зменшення споживання для освітлення при високій сонячній
активності
            adjustedPredictions[i] *= max(0.9, 1.0 - 0.0005 *
solarRadiation);
        }

        // Корекція на основі вологості (якщо є)
        if (factors.containsKey('humidity')) {
            double humidity = factors['humidity']!;

            // Невелика корекція для враховування вологості
            if (humidity > 80) {
                // Підвищене споживання при високій вологості (осушувачі,
кондиціонери)
                adjustedPredictions[i] *= 1.0 + 0.002 * (humidity - 80);
            }
        }
    }

    return adjustedPredictions;
}

/// Прогнозує споживання за допомогою експоненціального згладжування
/// [historicalData] - історичні дані про енергоспоживання
/// [horizon] - кількість точок для прогнозування
/// [alpha] - параметр згладжування (0 < alpha < 1)
/// Повертає список прогнозованих значень
List<double> predictWithExponentialSmoothing(

```

```

List<EnergyReading> historicalData,
int horizon,
{double alpha = 0.3}
) {
    if (historicalData.isEmpty) {
        throw Exception('Історичні дані порожні.');
```

}

```

    if (alpha <= 0 || alpha >= 1) {
        throw Exception('Параметр alpha повинен бути в межах (0, 1).');
```

}

```

    // Сортуємо дані за часом
    historicalData.sort((a, b) => a.timestamp.compareTo(b.timestamp));

    // Отримуємо лише значення споживання
    List<double> consumptionValues = historicalData.map((r) =>
r.value).toList();

    // Ініціалізуємо згладжене значення
    double smoothedValue = consumptionValues.first;

    // Застосовуємо експоненціальне згладжування до історичних даних
    for (int i = 1; i < consumptionValues.length; i++) {
        smoothedValue = alpha * consumptionValues[i] + (1 - alpha) *
smoothedValue;
    }

    // Прогнозуємо майбутні значення
    List<double> predictions = [];

    for (int i = 0; i < horizon; i++) {
        predictions.add(smoothedValue);
    }

    return predictions;
}

/// Прогнозує споживання на основі сезонних патернів
/// [historicalData] - історичні дані про енергоспоживання
/// [horizon] - кількість точок для прогнозування
/// [seasonalPeriod] - довжина сезонного циклу (напр., 24 для
годинних даних з денним циклом)
/// Повертає список прогнозованих значень
List<double> predictWithSeasonalPattern(
    List<EnergyReading> historicalData,
    int horizon,
    int seasonalPeriod
) {
    if (historicalData.length < seasonalPeriod * 2) {
        throw Exception('Недостатньо історичних даних для сезонного
прогнозування. Потрібно принаймні ${seasonalPeriod * 2} точок.');
```

}

```

    // Сортуємо дані за часом
    historicalData.sort((a, b) => a.timestamp.compareTo(b.timestamp));

    // Отримуємо лише значення споживання
```

```

    List<double> consumptionValues = historicalData.map((r) =>
r.value).toList();

    // Розрахунок сезонних індексів
    List<double> seasonalIndices = List<double>.filled(seasonalPeriod,
0.0);
    int numSeasons = consumptionValues.length ~/ seasonalPeriod;

    // Розрахунок середнього для кожного сезону
    for (int season = 0; season < numSeasons; season++) {
        for (int i = 0; i < seasonalPeriod; i++) {
            int index = season * seasonalPeriod + i;
            if (index < consumptionValues.length) {
                seasonalIndices[i] += consumptionValues[index];
            }
        }
    }

    // Нормалізація сезонних індексів
    for (int i = 0; i < seasonalPeriod; i++) {
        seasonalIndices[i] /= numSeasons;
    }

    // Розрахунок загального середнього
    double globalMean = consumptionValues.reduce((a, b) => a + b) /
consumptionValues.length;

    // Нормалізація сезонних індексів відносно загального середнього
    for (int i = 0; i < seasonalPeriod; i++) {
        seasonalIndices[i] = seasonalIndices[i] / globalMean;
    }

    // Прогнозування
    List<double> predictions = [];

    for (int i = 0; i < horizon; i++) {
        int seasonIndex = (consumptionValues.length + i) %
seasonalPeriod;
        predictions.add(globalMean * seasonalIndices[seasonIndex]);
    }

    return predictions;
}
}

```

Лістинг Б.4 - Алгоритм формування рекомендацій на основі правил

```

import 'dart:math';
import '../models/energy_reading.dart';
import '../models/device.dart';
import '../models/recommendation.dart';
import '../database/database_helper.dart';
import '../services/energy_cost_calculator.dart';

class RecommendationEngine {

```

```

final DatabaseHelper _dbHelper;
final EnergyCostCalculator _costCalculator;

RecommendationEngine(this._dbHelper, this._costCalculator);

/// Генерує рекомендації щодо оптимізації енергоспоживання
/// [userId] - ідентифікатор користувача
/// Повертає список рекомендацій
Future<List<Recommendation>> generateRecommendations(int userId)
async {
    List<Recommendation> recommendations = [];

    // Отримуємо дані про пристрої користувача
    List<Device> userDevices = await _dbHelper.getUserDevices(userId);

    // Отримуємо дані про енергоспоживання за останній місяць
    final DateTime now = DateTime.now();
    final DateTime monthAgo = DateTime(now.year, now.month - 1,
now.day);

    // Для кожного пристрою генеруємо індивідуальні рекомендації
    for (var device in userDevices) {
        List<EnergyReading> deviceReadings = await
_dbHelper.getDeviceReadingsByPeriod(
            device.id,
            monthAgo.millisecondsSinceEpoch,
            now.millisecondsSinceEpoch
        );

        if (deviceReadings.isEmpty) continue;

        // Додаємо рекомендації для конкретного пристрою
        recommendations.addAll(await
_generateDeviceRecommendations(device, deviceReadings));
    }

    // Додаємо загальні рекомендації на основі загального споживання
    List<EnergyReading> totalReadings = await
_dbHelper.getTotalReadingsByPeriod(
        userId,
        monthAgo.millisecondsSinceEpoch,
        now.millisecondsSinceEpoch
    );

    if (totalReadings.isNotEmpty) {

recommendations.addAll(_generateGeneralRecommendations(totalReadings,
userDevices));
    }

    // Додаємо рекомендації щодо тарифного плану
    recommendations.addAll(await
_generateTariffRecommendations(userId, totalReadings));

    // Сортуюмо рекомендації за потенційною економією (від найбільшої
до найменшої)
    recommendations.sort((a, b) =>
b.potentialSavings.compareTo(a.potentialSavings));
}

```

```

    return recommendations;
}

/// Генерує рекомендації для конкретного пристрою
Future<List<Recommendation>> _generateDeviceRecommendations(
    Device device,
    List<EnergyReading> readings
) async {
    List<Recommendation> recommendations = [];

    // Отримуємо профіль пристрою
    final deviceProfile = await
_dbHelper.getDeviceTypeProfile(device.type);

    // 1. Рекомендації щодо енергетичного вампіризму (споживання в
    режимі очікування)
    if (deviceProfile.hasStandbyConsumption) {
        double avgStandbyConsumption = await
_calculateAverageStandbyConsumption(device.id);

        if (avgStandbyConsumption >
deviceProfile.optimalStandbyConsumption * 1.5) {
            // Розрахунок потенційної економії
            double excessConsumption = avgStandbyConsumption -
deviceProfile.optimalStandbyConsumption;
            double dailyExcess = excessConsumption * 24; // кВт·год на
день
            double monthlyExcess = dailyExcess * 30; // кВт·год на місяць
            double monthlyCost = await
_costCalculator.calculateCostForConsumption(monthlyExcess);

            recommendations.add(Recommendation(
                id: 0, // Буде присвоєно при збереженні в БД
                title: "Вимикайте ${device.name} з розетки",
                description: "Цей пристрій споживає
${avgStandbyConsumption.toStringAsFixed(2)} Вт у режимі очікування, "
+
                "що в ${(avgStandbyConsumption /
deviceProfile.optimalStandbyConsumption).toStringAsFixed(1)} рази
більше за оптимальне значення. " +
                "Вимикайте його з розетки, коли не
використовуєте.",
                deviceId: device.id,
                category: RecommendationCategory.standbyConsumption,
                potentialSavings: monthlyCost,
                difficultyLevel: DifficultyLevel.easy,
                isApplied: false,
                timestamp: DateTime.now().millisecondsSinceEpoch
            ));
        }
    }

    // 2. Рекомендації щодо оптимального часу використання
    if (deviceProfile.isHighConsumer) {
        // Аналізуємо, коли пристрій використовується
        Map<int, double> hourlyConsumption = await
_analyzeHourlyConsumption(device.id);
    }
}

```

```

// Отримуємо години з найнижчим тарифом
List<int> cheapHours = await _getCheapestHours();

// Рахуємо, скільки енергії споживається в дорогі години
double totalConsumption = hourlyConsumption.values.fold(0, (sum,
value) => sum + value);
double expensiveHoursConsumption = 0;

for (int hour = 0; hour < 24; hour++) {
    if (hourlyConsumption.containsKey(hour) &&
!cheapHours.contains(hour)) {
        expensiveHoursConsumption += hourlyConsumption[hour]!;
    }
}

// Якщо більше 70% споживання припадає на дорогі години
if (totalConsumption > 0 && expensiveHoursConsumption /
totalConsumption > 0.7) {
    // Розрахунок потенційної економії
    double potentialSavings = await
_calculatePotentialTariffSaving(device.id, cheapHours);

    // Форматуємо години у зручний для читання формат
    cheapHours.sort();
    String formattedHours = cheapHours.map((h) =>
"${h.toString().padLeft(2, '0')}:00").join(", ");

    recommendations.add(Recommendation(
        id: 0,
        title: "Перенесіть використання ${device.name} на нічний
час",
        description: "Використовуйте ${device.name} у години з
найнижчим тарифом: $formattedHours. " +
"Це дозволить заощадити на оплаті електроенергії
без зміни загального об'єму споживання.",
        deviceId: device.id,
        category: RecommendationCategory.optimalUsageTime,
        potentialSavings: potentialSavings,
        difficultyLevel: DifficultyLevel.medium,
        isApplied: false,
        timestamp: DateTime.now().millisecondsSinceEpoch
    ));
}

// 3. Рекомендації щодо оптимальних налаштувань
if (deviceProfile.hasOptimalSettings) {
    // Отримуємо поточні налаштування пристрою
    Map<String, String> currentSettings = await
_dbHelper.getDeviceSettings(device.id);

    // Порівнюємо з оптимальними налаштуваннями
    List<String> suboptimalSettings = [];

    for (var optimalSetting in
deviceProfile.optimalSettings.entries) {
        String key = optimalSetting.key;

```

```

        String optimalValue = optimalSetting.value;

        if (currentSettings.containsKey(key) && currentSettings[key]
!= optimalValue) {
            suboptimalSettings.add("$key: ${currentSettings[key]} →
$optimalValue");
        }
    }

    if (suboptimalSettings.isNotEmpty) {
        // Розрахунок потенційної економії
        double potentialSavings = totalConsumption *
deviceProfile.settingsOptimizationFactor;
        double monthlyCost = await
_costCalculator.calculateCostForConsumption(potentialSavings);

        recommendations.add(Recommendation(
            id: 0,
            title: "Оптимізуйте налаштування ${device.name}",
            description: "Змініть наступні налаштування для оптимального
споживання енергії:\n" +
                "${suboptimalSettings.join("\n")}",
            deviceId: device.id,
            category: RecommendationCategory.optimalSettings,
            potentialSavings: monthlyCost,
            difficultyLevel: DifficultyLevel.easy,
            isApplied: false,
            timestamp: DateTime.now().millisecondsSinceEpoch
        ));
    }
}

// 4. Рекомендації щодо заміни старих пристроїв
if (device.age > 5 && deviceProfile.replacementSavingsFactor > 0)
{
    // Обчислюємо місячне споживання
    double monthlyConsumption = readings.fold(0, (sum, reading) =>
sum + reading.value);

    // Потенційна економія при заміні на новий пристрій
    double potentialSavings = monthlyConsumption *
deviceProfile.replacementSavingsFactor;
    double monthlyCost = await
_costCalculator.calculateCostForConsumption(potentialSavings);
    double annualSavings = monthlyCost * 12;

    // Приблизна вартість нового пристрою
    double estimatedNewDeviceCost =
deviceProfile.estimatedReplacementCost;

    // Обчислюємо період окупності
    double paybackPeriod = estimatedNewDeviceCost / annualSavings;

    // Якщо період окупності менше 3 років, рекомендуємо заміну
    if (paybackPeriod < 3) {
        recommendations.add(Recommendation(
            id: 0,
            title: "Розгляньте можливість заміни ${device.name}",

```

```

        description: "Ваш пристрій споживає на
${(deviceProfile.replacementSavingsFactor * 100).toStringAsFixed(0)}%
більше енергії " +
            "порівняно з сучасними моделями. Заміна
окупиться приблизно за ${paybackPeriod.toStringAsFixed(1)} років.",
        deviceId: device.id,
        category: RecommendationCategory.deviceReplacement,
        potentialSavings: annualSavings / 12, // Місячна економія
        difficultyLevel: DifficultyLevel.hard,
        isApplied: false,
        timestamp: DateTime.now().millisecondsSinceEpoch
    ));
}
}

return recommendations;
}

/// Генерує загальні рекомендації на основі загального споживання
List<Recommendation> _generateGeneralRecommendations(
    List<EnergyReading> totalReadings,
    List<Device> devices
) {
    List<Recommendation> recommendations = [];

    // 1. Рекомендація щодо зменшення базового навантаження
    double baseConsumption = _calculateBaseConsumption(totalReadings);
    double avgDailyConsumption =
        _calculateAverageDailyConsumption(totalReadings);

    if (baseConsumption > avgDailyConsumption * 0.4) {
        // Високе базове споживання (більше 40% від середнього)
        recommendations.add(Recommendation(
            id: 0,
            title: "Зменшіть базове споживання електроенергії",
            description: "Ваше базове споживання становить
${baseConsumption.toStringAsFixed(2)} кВт·год на день, " +
                "що складає ${((baseConsumption /
avgDailyConsumption) * 100).toStringAsFixed(0)}% від загального
споживання. " +
                "Перевірте, які пристрої постійно підключені до
мережі, і відключайте непотрібні.",
            deviceId: null, // Загальна рекомендація
            category: RecommendationCategory.baseLoadReduction,
            potentialSavings: baseConsumption * 0.3 * 30, // 30% від
місячного базового споживання
            difficultyLevel: DifficultyLevel.medium,
            isApplied: false,
            timestamp: DateTime.now().millisecondsSinceEpoch
        ));
    }

    // 2. Рекомендація щодо піків споживання
    double peakConsumption = _calculatePeakConsumption(totalReadings);

    if (peakConsumption > avgDailyConsumption * 2.5) {
        // Дуже високі піки споживання
        recommendations.add(Recommendation(

```

```

        id: 0,
        title: "Розподіліть навантаження для зменшення пікового
споживання",
        description: "Виявлено значні піки споживання електроенергії
(${peakConsumption.toStringAsFixed(2)} кВт·год), " +
            "що в ${peakConsumption /
avgDailyConsumption}.toStringAsFixed(1)} рази перевищує середній
рівень. " +
            "Уникайте одночасного використання кількох
потужних приладів.",
        deviceId: null,
        category: RecommendationCategory.peakLoadDistribution,
        potentialSavings: peakConsumption * 0.2 * 30, // 20% від
місячного пікового споживання
        difficultyLevel: DifficultyLevel.medium,
        isApplied: false,
        timestamp: DateTime.now().millisecondsSinceEpoch
    ));
}

// 3. Рекомендація щодо автоматизації
int automationCandidates = devices.where((d) =>
d.supportsAutomation && !d.isAutomated).length;

if (automationCandidates > 0) {
    double estimatedSavings = avgDailyConsumption * 0.15 * 30; //
Приблизно 15% економії при автоматизації

    recommendations.add(Recommendation(
        id: 0,
        title: "Налаштуйте автоматизацію пристроїв",
        description: "У вас є $automationCandidates
${_pluralize(automationCandidates, 'пристрій', 'пристрої',
'пристроїв')}, " +
            "${_pluralize(automationCandidates, 'який
підтримує', 'які підтримують', 'які підтримують')} автоматизацію. " +
            "Налаштуйте автоматичне вимкнення в неактивний час
для економії енергії.",
        deviceId: null,
        category: RecommendationCategory.automation,
        potentialSavings: estimatedSavings,
        difficultyLevel: DifficultyLevel.medium,
        isApplied: false,
        timestamp: DateTime.now().millisecondsSinceEpoch
    ));
}

return recommendations;
}

/// Генерує рекомендації щодо тарифного плану
Future<List<Recommendation>> _generateTariffRecommendations(
    int userId,
    List<EnergyReading> totalReadings
) async {
    List<Recommendation> recommendations = [];

    // Отримуємо поточний тарифний план

```

```

        final currentTariffPlan = await
        _dbHelper.getUserTariffPlan(userId);

        // Отримуємо доступні тарифні плани
        final availableTariffPlans = await
        _dbHelper.getAvailableTariffPlans();

        // Для кожного доступного тарифного плану проводимо аналіз
        for (var tariffPlan in availableTariffPlans) {
            if (tariffPlan.id == currentTariffPlan.id) continue;

            // Розраховуємо вартість споживання за поточним та
            альтернативним тарифними планами
            double currentCost = await
            _calculateCostWithTariffPlan(totalReadings, currentTariffPlan);
            double alternativeCost = await
            _calculateCostWithTariffPlan(totalReadings, tariffPlan);

            // Якщо економія більше 5%
            if (alternativeCost < currentCost * 0.95) {
                double monthlySavings = currentCost - alternativeCost;

                recommendations.add(Recommendation(
                    id: 0,
                    title: "Перейдіть на тарифний план \"${tariffPlan.name}\",
                    description: "Перехід на тарифний план
                    \"${tariffPlan.name}\" дозволить заощадити " +
                        "приблизно ${monthlySavings.toStringAsFixed(2)}
                    грн на місяць на основі вашого " +
                        "поточного профілю споживання.",
                    deviceId: null,
                    category: RecommendationCategory.tariffPlan,
                    potentialSavings: monthlySavings,
                    difficultyLevel: DifficultyLevel.medium,
                    isApplied: false,
                    timestamp: DateTime.now().millisecondsSinceEpoch
                ));
            }
        }

        return recommendations;
    }
}

```

Лістинг Б.5 - Код головного екрану додатку з розрахунком показників

```

import 'package:flutter/material.dart';
import 'package:fl_chart/fl_chart.dart';
import 'package:intl/intl.dart';
import '../models/energy_reading.dart';
import '../models/device.dart';
import '../models/recommendation.dart';
import '../services/energy_analytics_service.dart';
import '../services/device_service.dart';
import '../services/energy_cost_calculator.dart';
import '../utils/date_time_utils.dart';

```

```

import '../utils/constants.dart';
import '../widgets/energy_consumption_chart.dart';
import '../widgets/active_devices_list.dart';
import '../widgets/recommendation_card.dart';

class DashboardScreen extends StatefulWidget {
  const DashboardScreen({Key? key}) : super(key: key);

  @override
  _DashboardScreenState createState() => _DashboardScreenState();
}

class _DashboardScreenState extends State<DashboardScreen> {
  final EnergyAnalyticsService _analyticsService =
EnergyAnalyticsService();
  final DeviceService _deviceService = DeviceService();
  final EnergyCostCalculator _costCalculator = EnergyCostCalculator();

  // Вибраний період відображення даних
  String _selectedPeriod = 'day';

  // Дані для відображення
  double _currentConsumption = 0.0;
  double _todayConsumption = 0.0;
  double _todayCost = 0.0;
  List<EnergyReading> _consumptionData = [];
  List<Device> _activeDevices = [];
  List<Recommendation> _topRecommendations = [];

  // Статус завантаження даних
  bool _isLoading = true;
  bool _hasError = false;
  String _errorMessage = '';

  @override
  void initState() {
    super.initState();
    _loadDashboardData();

    // Налаштування періодичного оновлення даних
    _setupPeriodicDataUpdate();
  }

  void _setupPeriodicDataUpdate() {
    // Оновлення даних кожні 5 хвилин
    Future.delayed(const Duration(minutes: 5), () {
      if (mounted) {
        _refreshCurrentConsumption();
        _setupPeriodicDataUpdate();
      }
    });
  }

  Future<void> _loadDashboardData() async {
    setState(() {
      _isLoading = true;
      _hasError = false;
    });
  }

```

```

try {
    // Паралельне завантаження даних для покращення продуктивності
    await Future.wait([
        _loadConsumptionData(),
        _loadActiveDevices(),
        _loadRecommendations(),
    ]);

    setState(() {
        _isLoading = false;
    });
} catch (e) {
    setState(() {
        _isLoading = false;
        _hasError = true;
        _errorMessage = 'Помилка завантаження даних: ${e.toString()}';
    });
}
}

Future<void> _loadConsumptionData() async {
    // Визначаємо початок і кінець періоду
    DateTime now = DateTime.now();
    DateTime startDate;

    switch (_selectedPeriod) {
        case 'day':
            startDate = DateTime(now.year, now.month, now.day);
            break;
        case 'week':
            startDate = now.subtract(Duration(days: 7));
            break;
        case 'month':
            startDate = DateTime(now.year, now.month - 1, now.day);
            break;
        default:
            startDate = DateTime(now.year, now.month, now.day);
    }

    // Отримуємо актуальні дані споживання
    final data = await _analyticsService.getConsumptionData(
        startDate.millisecondsSinceEpoch,
        now.millisecondsSinceEpoch,
        _selectedPeriod == 'day' ? AggregationPeriod.hourly
                                : (_selectedPeriod == 'week' ?
AggregationPeriod.daily
                                :
AggregationPeriod.daily)
    );

    // Отримуємо поточне споживання
    final currentReading = await
_analyticsService.getCurrentConsumption();

    // Отримуємо споживання за сьогодні
    final todayConsumption = await
_analyticsService.getTodayConsumption();

```

```

// Розраховуємо вартість споживання за сьогодні
final todayCost = await _costCalculator.calculateCost(
    DateTime(now.year, now.month, now.day).millisecondsSinceEpoch,
    now.millisecondsSinceEpoch,
    null // Для загального споживання
);

setState(() {
    _consumptionData = data;
    _currentConsumption = currentReading;
    _todayConsumption = todayConsumption;
    _todayCost = todayCost;
});
}

Future<void> _loadActiveDevices() async {
    // Отримуємо активні пристрої, сортовані за споживанням
    final devices = await _deviceService.getActiveDevices();
    devices.sort((a, b) =>
b.currentConsumption.compareTo(a.currentConsumption));

    setState(() {
        _activeDevices = devices;
    });
}

Future<void> _loadRecommendations() async {
    // Отримуємо топ рекомендацій за потенційною економією
    final recommendations = await
_analyticsService.getTopRecommendations(limit: 3);

    setState(() {
        _topRecommendations = recommendations;
    });
}

Future<void> _refreshCurrentConsumption() async {
    try {
        // Оновлюємо лише поточне споживання без перезавантаження всього
дашборду
        final currentReading = await
_analyticsService.getCurrentConsumption();

        setState(() {
            _currentConsumption = currentReading;
        });

        // Також оновлюємо активні пристрої, оскільки їхнє споживання
могло змінитися
        await _loadActiveDevices();
    } catch (e) {
        // Тихо ігноруємо помилки при оновленні, щоб не переривати
роботу додатку
        print('Помилка оновлення поточного споживання: $e');
    }
}

```

```

void _changePeriod(String period) {
  setState(() {
    _selectedPeriod = period;
  });
  _loadConsumptionData();
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text('Моніторинг енергоспоживання'),
      actions: [
        IconButton(
          icon: Icon(Icons.refresh),
          onPressed: _loadDashboardData,
          tooltip: 'Оновити дані',
        ),
        IconButton(
          icon: Icon(Icons.notifications),
          onPressed: () => Navigator.pushNamed(context,
'/notifications'),
          tooltip: 'Сповіщення',
        ),
      ],
    ),
    body: _isLoading
      ? _buildLoadingView()
      : _hasError
      ? _buildErrorView()
      : _buildDashboardContent(),
  );
}

Widget _buildLoadingView() {
  return Center(
    child: Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        CircularProgressIndicator(),
        SizedBox(height: 16),
        Text('Завантаження даних...'),
      ],
    ),
  );
}

Widget _buildErrorView() {
  return Center(
    child: Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        Icon(Icons.error_outline, size: 48, color: Colors.red),
        SizedBox(height: 16),
        Text(_errorMessage),
        SizedBox(height: 24),
        ElevatedButton(
          onPressed: _loadDashboardData,

```

```

        child: Text('Спробувати знову'),
      ),
    ],
  ),
);
}

Widget _buildDashboardContent() {
  return RefreshIndicator(
    onRefresh: _loadDashboardData,
    child: SingleChildScrollView(
      physics: AlwaysScrollableScrollPhysics(),
      padding: EdgeInsets.all(16),
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          _buildConsumptionOverview(),
          SizedBox(height: 24),
          _buildConsumptionChart(),
          SizedBox(height: 24),
          _buildActiveDevicesSection(),
          SizedBox(height: 24),
          _buildRecommendationsSection(),
        ],
      ),
    ),
  );
}

Widget _buildConsumptionOverview() {
  return Card(
    elevation: 4,
    shape: RoundedRectangleBorder(
      borderRadius: BorderRadius.circular(12),
    ),
    child: Padding(
      padding: EdgeInsets.all(16),
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          Text(
            'Поточне споживання',
            style: TextStyle(
              fontSize: 16,
              fontWeight: FontWeight.bold,
              color: Colors.grey[700],
            ),
          ),
          SizedBox(height: 8),
          Row(
            crossAxisAlignment: CrossAxisAlignment.end,
            children: [
              Text(
                '${_currentConsumption.toStringAsFixed(2)}',
                style: TextStyle(
                  fontSize: 32,
                  fontWeight: FontWeight.bold,
                  color: _getConsumptionColor(_currentConsumption),
                ),
              ),

```

```

    ),
  ),
  SizedBox(width: 4),
  Padding(
    padding: EdgeInsets.only(bottom: 5),
    child: Text(
      'кВт',
      style: TextStyle(
        fontSize: 18,
        color: Colors.grey[700],
      ),
    ),
  ),
),
],
),
Divider(height: 24),
Row(
  mainAxisAlignment: MainAxisAlignment.spaceBetween,
  children: [
    _buildConsumptionMetric(
      'Сьогодні',
      '${_todayConsumption.toStringAsFixed(2)} кВт·год',
    ),
    _buildConsumptionMetric(
      'Вартість',
      '${_todayCost.toStringAsFixed(2)} грн',
    ),
  ],
),
],
),
),
);
}

Widget _buildConsumptionMetric(String label, String value) {
  return Column(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: [
      Text(
        label,
        style: TextStyle(
          fontSize: 14,
          color: Colors.grey[600],
        ),
      ),
      SizedBox(height: 4),
      Text(
        value,
        style: TextStyle(
          fontSize: 16,
          fontWeight: FontWeight.bold,
        ),
      ),
    ],
  );
}

```

```

Color _getConsumptionColor(double consumption) {
  if (consumption < 1.0) {
    return Colors.green;
  } else if (consumption < 3.0) {
    return Colors.orange;
  } else {
    return Colors.red;
  }
}

Widget _buildConsumptionChart() {
  return Card(
    elevation: 4,
    shape: RoundedRectangleBorder(
      borderRadius: BorderRadius.circular(12),
    ),
    child: Padding(
      padding: EdgeInsets.all(16),
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          Row(
            mainAxisAlignment: MainAxisAlignment.spaceBetween,
            children: [
              Text(
                'Графік споживання',
                style: TextStyle(
                  fontSize: 16,
                  fontWeight: FontWeight.bold,
                  color: Colors.grey[700],
                ),
              ),
              _buildPeriodSelector(),
            ],
          ),
          SizedBox(height: 16),
          Container(
            height: 240,
            child: _consumptionData.isEmpty
              ? Center(child: Text('Немає даних для відображення'))
              : EnergyConsumptionChart(
                  data: _consumptionData,
                  period: _selectedPeriod,
                ),
          ),
        ],
      ),
    ),
  );
}

Widget _buildPeriodSelector() {
  return SegmentedButton<String>(
    segments: [
      ButtonSegment<String>(value: 'day', label: Text('День')),
      ButtonSegment<String>(value: 'week', label: Text('Тиждень')),
      ButtonSegment<String>(value: 'month', label: Text('Місяць')),
    ],
  ),

```

```

        selected: {_selectedPeriod},
        onSelectionChanged: (Set<String> newSelection) {
            _changePeriod(newSelection.first);
        },
    );
}

Widget _buildActiveDevicesSection() {
    return Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
            Row(
                mainAxisAlignment: MainAxisAlignment.spaceBetween,
                children: [
                    Text(
                        'Активні пристрої',
                        style: TextStyle(
                            fontSize: 18,
                            fontWeight: FontWeight.bold,
                        ),
                    ),
                    TextButton(
                        onPressed: () => Navigator.pushNamed(context,
'/devices'),
                        child: Text('Усі пристрої'),
                    ),
                ],
            ),
            SizedBox(height: 8),
            _activeDevices.isEmpty
                ? Card(
                    child: Padding(
                        padding: EdgeInsets.all(16),
                        child: Center(
                            child: Text('Немає активних пристроїв'),
                        ),
                    ),
                )
                : ActiveDevicesList(
                    devices: _activeDevices,
                    onDeviceTap: (device) => Navigator.pushNamed(
                        context,
                        '/device_details',
                        arguments: device,
                    ),
                ),
        ],
    );
}

Widget _buildRecommendationsSection() {
    return Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
            Row(
                mainAxisAlignment: MainAxisAlignment.spaceBetween,
                children: [
                    Text(

```

```

        'Рекомендації',
        style: TextStyle(
          fontSize: 18,
          fontWeight: FontWeight.bold,
        ),
      ),
      TextButton(
        onPressed: () => Navigator.pushNamed(context,
'/recommendations'),
        child: Text('Усі рекомендації'),
      ),
    ],
  ),
  SizedBox(height: 8),
  _topRecommendations.isEmpty
    ? Card(
      child: Padding(
        padding: EdgeInsets.all(16),
        child: Center(
          child: Text('Наразі немає рекомендацій'),
        ),
      ),
    )
    : Column(
      children: _topRecommendations.map((recommendation) =>
        RecommendationCard(
          recommendation: recommendation,
          onApply: () => _applyRecommendation(recommendation),
          onDismiss: () =>
            _dismissRecommendation(recommendation),
        )
      ).toList(),
    ),
  ],
);
}

void _applyRecommendation(Recommendation recommendation) async {
  // Помічаємо рекомендацію як застосовану
  await
_analyticsService.markRecommendationAsApplied(recommendation.id);

  // Оновлюємо список рекомендацій
  await _loadRecommendations();

  // Показуємо повідомлення про успішне застосування
  ScaffoldMessenger.of(context).showSnackBar(
    SnackBar(
      content: Text('Рекомендацію застосовано'),
      backgroundColor: Colors.green,
    ),
  );
}

void _dismissRecommendation(Recommendation recommendation) async {
  // Помічаємо рекомендацію як відхилену
  await _analyticsService.dismissRecommendation(recommendation.id);
}

```

```
// Оновлюємо список рекомендацій
await _loadRecommendations();

// Показуємо повідомлення про відхилення
ScaffoldMessenger.of(context).showSnackBar(
  SnackBar(
    content: Text('Рекомендацію відхилено'),
  ),
);
}
```

ДОДАТОК В
Слайди презентації



Рисунок В.1 – Слайд 1

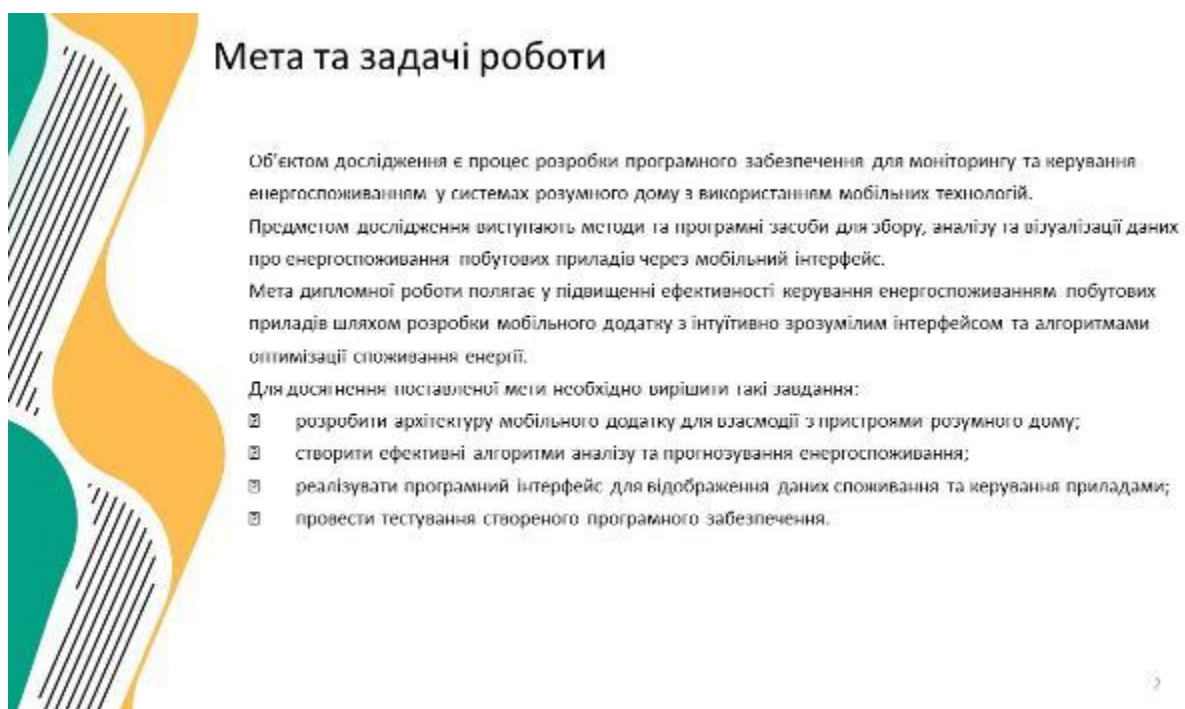


Рисунок В.2 – Слайд 2

Порівняльний аналіз функціональності мобільних додатків для моніторингу енергоспоживання

Характеристика	SmartThings Energy	Sense Home Energy	Wiser Energy	Energy Consumption Analyzer	Home Assistant Energy
Моніторинг загального споживання	Так	Так	Так	Так	Так
Моніторинг окремих пристроїв	Частково	Так	Так	Ні	Так
Технологія збору даних	Hub і датчики	Розподілений сигнал	Спеціалізований лічильник	Ручне введення	API пристроїв
Виконання "вирішувачів"	Так	Так	Так	Ні	Частково
Пропозиції оптимізації	Так	Так	Так	Так	Ні
Керування пристроями	Так	Ні	Так	Ні	Так
Автоматизація	Розширена	Ні	Розширена	Ні	Розширена
Інтеграція з іншими системами	Обмежена	Обмежена	Розширена	Ні	Розширена
Підтримка платформ	iOS, Android	iOS, Android	iOS, Android	iOS, Android	iOS, Android, Web
Бізнес-модель	Безкоштовно	+ Платне обладнання	Платне обладнання + підписка	Безкоштовно/Преміум	Безкоштовно (відкритий код)

Рисунок В.3 – Слайд 3

Вимоги до мобільного додатку

Категорія	Вимога	Пріоритет
Функціональний	Моніторинг загального енергоспоживання в реальному часі	Високий
	Відстеження енергоспоживання окремих приладів	Високий
	Перевірка показань (звонки/вишук, який режим)	Високий
	Автоматизація (як роботи приладів) як розкладів	Середній
	Аналітика та візуалізація статистичних показників	Високий
	Впорядкованість історії споживання за тижнями	Високий
	Система сповіщень про аномалії та події	Високий
Інформаційний	Якщо неможливо, то альтернативні енергоспоживання	Середній
	Швидкість та якість інтерфейсу без затримок	Високий
	Стабільна робота без збоїв	Високий
	Робота при відсутності інтернету (інтерактивні)	Середній
	Важливі дані та безпечне зберігання приладів	Високий
	Інтуїтивний, зрозумілий інтерфейс	Високий
	Підтримка платформ iOS та Android	Високий
Інтерфейс	Інтеграція з популярними системами розумного дому	Середній
	Ефективне використання ресурсів пристрою	Середній
	Належна інформація про стан та функціонал приладів	Високий
	Дружна візуалізація даних (графіки, діаграми)	Високий
	Проста та зрозуміла інформація на екрані	Високий
	Адаптивність до різних розмірів екранів	Високий
	Підтримка сучасних версій мобільних ОС	Високий
Технічний	Використання стандартних протоколів комунікації	Високий
	Підтримка основних протоколів IoT (Zigbee, Thread, Wi-Fi)	Високий
	Локальне зберігання даних з синхронізацією	Середній

Рисунок В.4 – Слайд 4

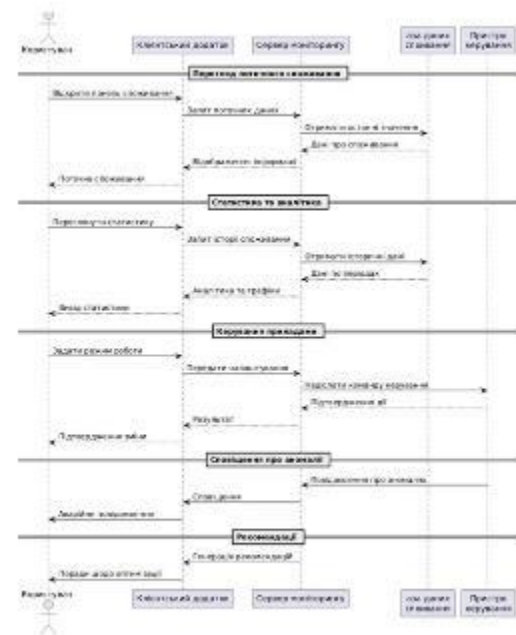


Рисунок В.5 – Слайд 5

Порівняння мов програмування та фреймворків для мобільної розробки

Критерій	Swift	Kotlin	Flutter (Dart)	React Native (JavaScript)
Підтримувані платформи	iOS	Android	iOS, Android, веб, desktop	iOS, Android
Продуктивність	Висока (нативна)	Висока (нативна)	Висока (близька до нативної)	Середня висока
Доступ до нативних функцій платформи	Повний	Повний	Через плагіни	Через нативні модулі
Спільнота та екосистема	Велика, активна	Велика, активна	Зростаюча	Велика, активна
Крива навчання	Середня	Низька-середня	Середня	Низька для веб-розробників
Інтеграція з IoT та системами розумного дому	Обмежена	Обмежена	Хороша, через плагіни	Хороша, через нативні модулі
Стабільність та зрілість	Висока	Висока	Середня	Середня висока
Підтримка протоколів комунікації для IoT	Обмежена	Обмежена	Широка, через плагіни	Широка, через нативні модулі

Рисунок В.6 – Слайд 6

Структура бази даних для збереження даних енергоспоживання

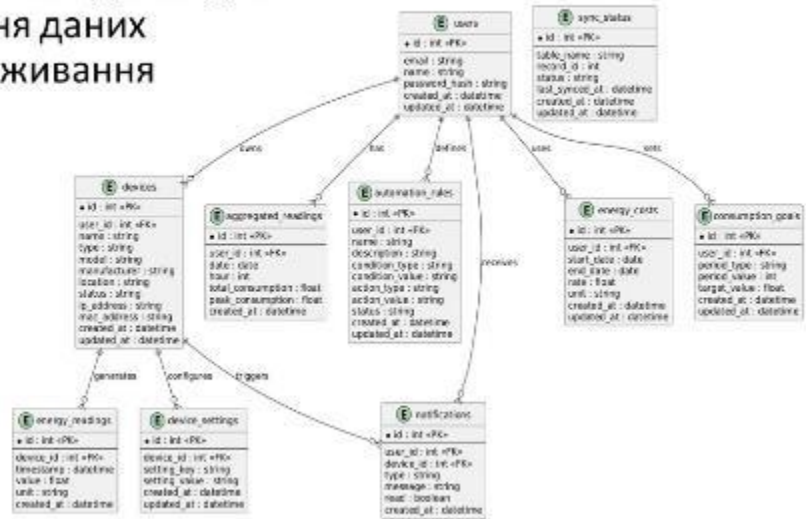


Рисунок В.7 – Слайд 7

Реалізація алгоритмів аналізу та оптимізації енергоспоживання



Тип приладу	Характер споживання	Типова потужність (Вт)	Допустиме відхилення [%]	Типові аномалії
Холодильник	Циклічний	100-300	±15	Тривала нереперечна споживання, відсутність споживання
Пральна машина	Баттловий	400-2500	±20	Видатне споживання у фазі нагріву, надто довгий цикл
Телевізор	Стабільний	50-150	±10	Типова споживання в режимі очікування
Освітлення	Стабільний	≤ 100	±5	Значне по відношенню до очікуваного
Кондиціонер	Змінний	500-3500	±25	Різкі скачки споживання, несподівані споживання

Категорія	Умови виявлення	Рекомендації	Потенційна економія [%]
Холодильник	Температура +10°C	Піднизити температуру до +5°C	5-10
Пральна машина	Використання прального циклу	Використати режим еко- або еко-еко	10-30
Освітлення	Одностороннє використання світла	Використати світлодіодні лампи	15-25
Телевізор	Висока потужність у режимі очікування	Піднизити потужність у режимі очікування	3-8
Кондиціонер	Температура +22°C	Піднизити температуру до +18°C	10-15

Рисунок В.8 – Слайд 8

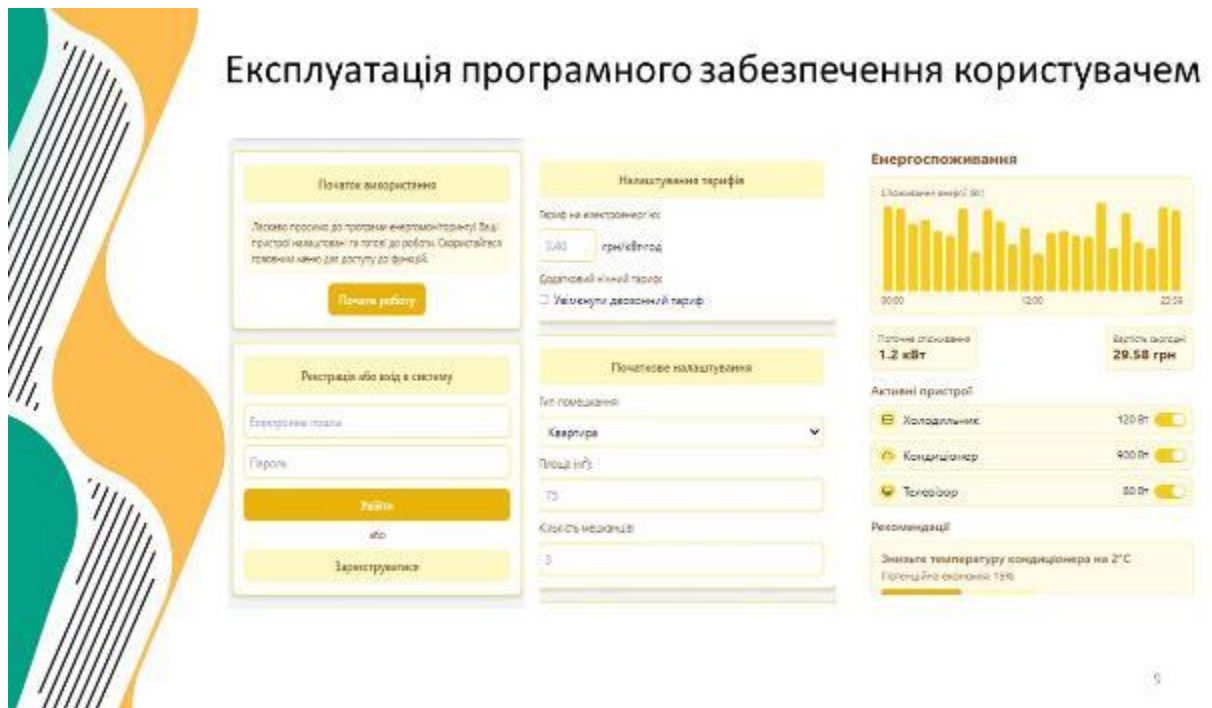


Рисунок В.9 – Слайд 9

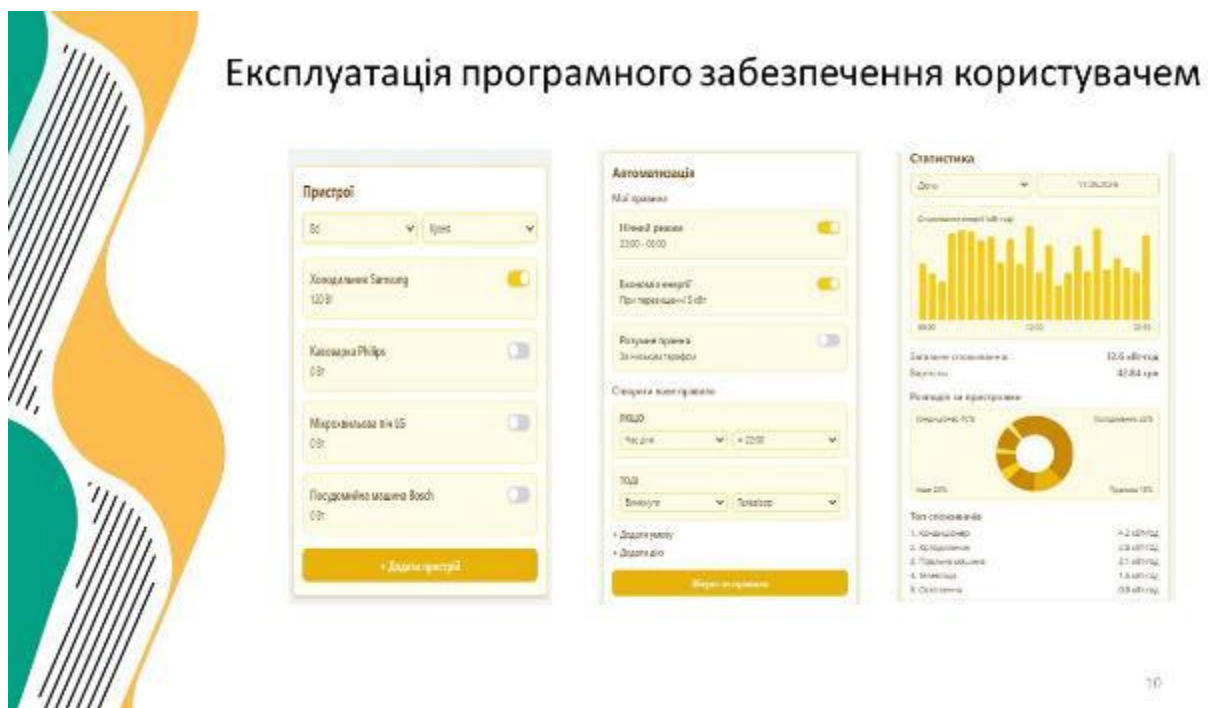


Рисунок В.10 – Слайд 10



Результати функціонального тестування

Функція	Тестовий сценарій	Очікуваний результат	Фактичний результат	Статус
Відображення загального споживання	Перевірка коректності відображення загального споживання енергії в режимі реального часу	Додаток відображає актуальні дані про загальне споживання в режимі реального часу	Дані відображаються коректно, оновлення відбувається відносно очікуваних	Пройдено
	Перевірка відображення енергоспоживання для кожного пристрою	Для кожного пристрою відображається його поточне споживання та статус	Для всіх пристроїв відображається споживання та статус	
Графіки споживання	Побудова графіків за різні періоди (день, тиждень, місяць)	Графіки відображають дані за вибраний період	Графіки відображають дані, взаємодія з графіками працює правильно	Пройдено
	Виявлення аномалій	Система виявляє аномалії та відповідне сповіщення	Система виявляє аномалії, сповіщення з'являється із затримкою	

11

Рисунок В.11 – Слайд 11



Висновки

Мету роботи досягнуто.

Розроблено програмне забезпечення для розумного моніторингу та керування енергоспоживанням побутових приладів у формі мобільного додатку. Реалізовано усі поставлені завдання: розроблено багаторівневу архітектуру з використанням шаблону MVVM, створено ефективні алгоритми аналізу енергоспоживання, реалізовано інтуїтивний інтерфейс користувача та проведено успішне тестування системи. Система забезпечує моніторинг у реальному часі, автоматизацію керування пристроями та генерацію персоналізованих рекомендацій.

Подальший розвиток передбачає розширення підтримки типів розумних пристроїв, впровадження методів машинного навчання та адаптацію для використання в комерційному секторі.

12

Рисунок В.12 – Слайд 12