МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ НА REACT NATIVE ДЛЯ АНАЛІЗУ ПОТЕНЦІЙНОГО ДОБОВОГО ВИРОБНИЦТВА СОНЯЧНОЇ ЕНЕРГІЇ ТА ОЦІНКИ МОЖЛИВОСТЕЙ ЇЇ ВИКОРИСТАННЯ У ПОБУТІ»**

Виконала: студентка 4 курсу, групи 6.1221
спеціальності 122 Комп'ютерні науки
(шифр і назва спеціальності)
освітньої програми Комп'ютерні науки
(назва освітньої програми)
К.А. Пазинич
(ініціали та прізвище)
Керівник завідувачка кафедри комп'ютерних наук, професор, д.т.н. Шило Г.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент завідувач кафедри програмної інженерії, доцент, к.ф.-м.н. Лісняк А.О.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра комп'ютерних наук

Рівень вищої освіти бакалавр

Спеціальність 122 Комп'ютерні науки
(шифр і назва)

Освітня програма Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук,
д.т.н., професор

_____ Шило Г.М.
(підпис)

" 23 " 12 2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТЦІ

Пазинич Катерині Андріївні
(прізвище, ім'я та по-батькові)

1. Тема роботи Розробка мобільного застосунку на React Native для аналізу потенційного добового виробництва сонячної енергії та оцінки можливостей її використання у побуті
керівник роботи Шило Галина Миколаївна, д.т.н., професор
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 20 » грудня 2024 року № 2001-с

2. Строк подання студентом роботи 02.06.2025

3. Вихідні дані до роботи
1. Середньодобові значення інсоляції для областей України за місяцями.
2. Типи, параметри та характеристики сонячних панелей.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Аналіз предметної області.
2. Розробка архітектури проєкту.
3. Розробка алгоритмів та кодова реалізація застосунку.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
презентація

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 23.12.2024

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	30.12.2024	
2.	Збір вихідних даних.	14.01.2025	
3.	Обробка методичних та теоретичних	11.02.2025	
	джерел.		
4.	Розробка першого та другого розділу.	02.03.2025	
5.	Розробка третього розділу.	23.03.2025	
6.	Оформлення та нормоконтроль	20.04.2025	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	18.06.2025	

Студент _____ К.А. Пазинич
(підпис) (ініціали та прізвище)

Керівник роботи _____ Г.М. Шило
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О.Г.Спиця
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка мобільного застосунку на React Native для аналізу потенційного добового виробництва сонячної енергії та оцінки можливостей її використання у побуті» 87 с., 22 рис., 1 таблиця, 16 джерел, 9 додатків.

API, EXPO, EXPRESS, JAVA SCRIPT, MONGODB, NODE.JS, REACT NATIVE, АВТОРИЗАЦІЯ, АЛГОРИТМ, МОБІЛЬНИЙ ЗАСТОСУНОК, СОНЯЧНА ЕНЕРГІЯ.

Об’єкт дослідження: Процеси оцінювання потенціалу сонячної енергії.

Предмет дослідження: Алгоритми розрахунку добового виробництва сонячної енергії та розподілення її між девайсами.

Мета роботи: Розробка мобільного застосунку для розрахунку потенційного добового виробництва сонячної енергії та визначення способів її розподілення між побутовими приладами за вказаними пріоритетами.

У ході роботи було використано метод аналізу наявних застосунків розрахунку виробництва енергії сонячними панелями. Застосовано метод порівняння для аналізу та вибору кращих технологій розробки проєкту. Також використано метод експерименту для тестування функціональності та інтерфейсу додатку.

У кваліфікаційній роботі розроблено мобільний застосунок за допомогою технології React Native для розрахунку виробництва сонячної енергії з урахуванням географічних координат, потужності сонячних панелей та погодних умов. Програмний продукт використовує дані, отримані від сторонніх API, для точного прогнозування виробництва енергії та оптимального її розподілення між побутовими приладами.

Мобільний застосунок може використовуватись як самостійний інструмент для фізичних осіб, домогосподарств або бізнесів, що планують

встановлення сонячних панелей, а також може бути впроваджений як один із компонентів екосистеми енергетичних компаній.

Отримані результати роботи можуть бути використані для подальшого вдосконалення мобільних застосунків та сервісів компаній, що працюють з відновлюваними джерелами енергії. Також у майбутньому можливе впровадження додаткових функцій, наприклад, інтеграцію з розумними домашніми системами.

SUMMARY

Bachelor's Qualification Thesis «Development of a mobile application on React Native for analysing the potential daily production of solar energy and estimating the possibilities of its use in the household» 87 pages, 22 figures, 1 table, 16 references, 9 supplements.

API, EXPO, EXPRESS, JAVA SCRIPT, MONGODB, NODE.JS, REACT NATIVE, AUTHORIZATION, ALGORITHM, MOBILE APPLICATION, SOLAR ENERGY.

The object of the study is the processes of solar energy potential evaluation.

The subject of the study are algorithms for calculating daily solar energy production and its distribution among devices.

The aim of the study is to develop a mobile application for calculating the potential daily solar energy production and determining how to distribute it among household appliances according to the specified priorities.

In the course of the work, we used the method of analysing existing applications for calculating energy production by solar panels. The comparative method was used to analyse and select the best project development technologies. An experiment method was also used to test the functionality and interface of the application.

In the qualification work, a mobile application was developed using React Native technology to calculate solar energy production based on geographic coordinates, solar panel power, and weather conditions. The software product uses data received from third-party APIs to accurately predict energy production and optimize its distribution among household appliances.

The mobile application can be used as a standalone tool for individuals, households, or businesses planning to install solar panels, and can also be implemented as part of the ecosystem of energy companies.

The results of this work can be used to further improve mobile applications and services for companies working with renewable energy sources. Additional features, such as integration with smart home systems, may also be introduced in the future.

ЗМІСТ

Завдання на кваліфікаційну роботу……..……..…..2
Реферат 4
Summary 6
Скорочення та умовні познаки 10
Вступ 11
1 Аналіз предметної області 13
1.1 Аналіз підходів до прогнозування виробництва сонячної енергії 13
1.2 Дослідження існуючих застосунків та веб-сервісів 17
1.2.1 Сервіс PVWatts Calculator 18
1.2.2 Сервіс PVGIS (Photovoltaic Geographical Information System) 19
1.2.3 Застосунок mySolarEdge 20
1.3 Вимоги до застосунку 23
1.4 Аналіз методів досліджень і вибір технологій 25
1.4.1 Методи дослідження 25
1.4.2 Технології серверної частини Node.js та Express.js 26
1.4.3 База даних MongoDB 27
1.4.4 Технології клієнтської частини React Native та Expo CLI (Command Line Interface) 28
1.5 Постановка завдань дослідження 29
2 Розробка архітектури проєкту 32
2.1 База даних 32
2.2 Архітектура Backend-частини 33
2.3 Архітектура Frontend-частини 36
2.4 Організація зв’язку між серверною та клієнтською сторонами 40
3 Розробка алгоритмів та кодова реалізація застосунку 41
3.1 Розробка алгоритму прогнозування добового виробництва енергії 41
3.2 Розробка алгоритму розподілу енергії між приладами 43

3.3 Реалізація Backend-частини 45
3.4 Розробка Frontend-частини 48
3.4.1 Навігація та організація сторінок 48
3.4.2 Відображення карти та робота з геоданими 50
3.4.3 Запити до API та обробка даних 52
3.4.4 Управління станом за допомогою Zustand 53
3.4.5 Реалізація UI-компонентів 55
3.5 Тестування роботи додатку 57
Висновки 60
Перелік посилань 62
Додаток А Програмний код алгоритму обрахунку виробленої сонячної енергії 64
Додаток Б Програмний код алгоритму розподілу енергії 66
Додаток В Основний файл backend-частини app.js 68
Додаток Г Файл-маршрутизатор на серверній частині для приладів 69
Додаток Д Програмна реалізація компонента Map 72
Додаток Е Програмний код користувацького хука usePanelCard 78
Додаток Ж Приклад використання анімації у компоненті DevicesList 80
Додаток И Знімки екранів програми з розрахунками енергії з 2 по 6 квітня . 82
Додаток К Знімки розрахунків енергії від сервісу PVWatts Calculator з 2 по 6 квітня 85

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

БД	База даних
ККД	Коефіцієнт корисної дії
API	Application Programming Interface
CLI	Command Line Interface
CSV	Comma-Separated Values
iOS	iPhone Operating System
JSON	JavaScript Object Notation
NASA	National Aeronautics and Space Administration
NoSQL	Not only SQL
PDF	Portable Document Format
REST	Representational State Transfer
SQL	Structured Query Language
UI	User Interface

ВСТУП

У сучасному світі, де альтернативні та природні джерела енергії набувають все більшого значення, використання сонячної енергія є одним із способів покращення екологічної ситуації нашої планети. Проте багато користувачів сонячних панелей можуть стикнутись з декількома проблемами і однією з них є відсутність зручного інструменту для розрахунку виробленої енергії та її оптимального використання. Багато людей не знають чи не можуть розрахувати кількість енергії, яку можна згенерувати за день, та які є способи розподілення її між побутовими приладами, або скільки часу кожен прилад може працювати від цієї енергії. Наслідком цього може бути неефективне використання ресурсів і ймовірна можливість додаткових витрат на енергію з інших джерел.

Актуальність наукових досліджень у галузі оптимізації використання відновлюваних джерел енергії полягає в тому, що вони допомагають користувачам правильно оцінити, скільки енергії можуть отримати, і як краще її використовувати для побутових потреб. Метою даної роботи є створення базового мобільного застосунку, який буде розраховувати потенційне вироблення сонячної енергії на основі географічного розташування, кліматичних умов та потужності панелей, а також пропонувати оптимальний розподіл цієї енергії між побутовими приладами.

До виконання кваліфікаційної роботи були поставлені такі завдання:

– аналіз наявних програмних продуктів для розрахунку вироблення сонячної енергії;
– визначення алгоритмів для оптимального розподілення виробленої енергії між побутовими приладами;
– створення мобільного застосунку за допомогою технології React Native з інтуїтивно зрозумілим інтерфейсом для користувачів, який

дозволяє відстежувати виробництво енергії та планувати її споживання;

– інтеграція даних з API (Application Programming Interface) для автоматичного отримання інформації про погоду та дані приладів;
– проведення тестування функціональності реалізованих алгоритмів та стабільності й коректності роботи розробленого застосунку.

Мобільний продукт буде розроблений на базі технологій React Native, що зробить наш продукт кросплатформеним і надасть змогу використовувати його як на Android, так і на iOS (iPhone Operating System) девайсах. Для зберігання даних про користувачів, їх сонячних панелей, також приладів та їхнє споживання, використовуватимуться сервіси MongoDB, а для прогнозування виробництва сонячної енергії – сторонні API для отримання погодних і кліматичних умов. Для серверної частини буде використовуватись Node.js, що дозволить виконувати складні обчислення та забезпечить ефективну обробку запитів від клієнтської частини застосунку.

Результати цієї роботи матимуть як теоретичне, так і практичне значення. Використовуючи розроблений застосунок, користувачі зможуть раціонально використовувати енергію, підвищуючи ефективність домашніх систем і знижуючи витрати на енергію. Це також сприятиме підвищенню інтересу до використання відновлювальних джерел енергії, що є важливим кроком на шляху до більш екологічного способу життя. У майбутньому застосунок може бути розширений для обробки інших альтернативних джерел енергії та нових алгоритмів для більш точного прогнозування та розподілу енергії.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз підходів до прогнозування виробництва сонячної енергії

Протягом останніх десятиліть спостерігається інтенсивний розвиток досліджень щодо використання відновлюваних джерел енергії, особливо сонячної. В Україні розвиток альтернативної енергетики визначено як один із пріоритетних напрямків енергетичної стратегії до 2035 року [1]. Потенціал сонячної енергетики в країні є значним, а її впровадження може суттєво знизити залежність від традиційних енергоресурсів та покращити стан навколишнього середовища [2].

У сучасній літературі розглядаються різні підходи до прогнозування виробництва енергії сонячними панелями. Дослідники пропонують як статичні моделі, що ґрунтуються на історичних даних, так і фізичні, які враховують метеорологічні умови, географічне розташування та технічні характеристики панелей. Деякі автори зосереджуються на підвищенні точності прогнозування за допомогою машинного навчання, нейронних мереж та штучного інтелекту, що і створює собою ще одну модель прогнозування – адаптивну [3].

Фізичний метод прогнозування виробництва енергії сонячними панелями представляється найбільш оптимальним вибором для нашого дослідження. На відміну від статистичних та адаптивних моделей, фізичний метод враховує конкретні технічні характеристики фотоелектричних модулів, а також комплексний вплив метеорологічних параметрів, включаючи інтенсивність сонячного випромінювання, температуру повітря та хмарність. Такий підхід дозволяє отримати більш надійні та точні прогнози, особливо для приватних та нових сонячних електростанцій, де історичні дані виробництва відсутні або обмежені.

Для виконання завдання нашої роботи було проведено пошук та аналіз необхідних вихідних даних. Зокрема, для оцінки потенціалу сонячної енергії в

різних регіонах України використано дані щодо середньодобової величини сонячної інсоляції з розподілом за місяцями року [4]. Ці дані представлені у вигляді таблиць із середніми показниками за спостереженнями NASA (National Aeronautics and Space Administration) за останні 22 роки та вимірюються у кВт· год/м²/день (рис. 1.1). Такі відомості є фундаментальними для прогнозування потенційного виробництва електроенергії в різних областях країни та в різні пори року.

Міста/Місяці	січень	лютий	березень	квітень	травень	червень	липень	серпень	вересень	жовтень	листопад	грудень	Сер.
Вінниця	1.07	1.89	2.94	3.92	5.19	5.30	5.16	4.68	3.21	1.97	1.10	0.90	3.11
Дніпропетровськ	1.21	1.99	2.98	4.05	5.55	5.57	5.70	5.08	3.66	2.27	1.20	0.96	3.36
Донецьк	1.21	1.99	2.94	4.04	5.48	5.55	5.66	5.09	3.67	2.24	1.23	0.96	3.34
Житомир	1.01	1.82	2.87	3.88	5.16	5.19	5.04	4.66	3.06	1.87	1.04	0.83	3.04
Запоріжжя	1.21	2.00	2.91	4.20	5.62	5.72	5.88	5.18	3.87	2.44	1.25	0.95	3.44
Івано-Франківськ	1.19	1.93	2.84	3.68	4.54	4.75	4.76	4.40	3.06	2.00	1.20	0.94	2.94
Київ	1.07	1.87	2.95	3.96	5.25	5.22	5.25	4.67	3.12	1.94	1.02	0.86	3.10
Кропивницький	1.20	1.95	2.96	4.07	5.47	5.49	5.57	4.92	3.57	2.24	1.14	0.96	3.30
Луцьк	1.02	1.77	2.83	3.91	5.05	5.08	4.94	4.55	3.01	1.83	1.05	0.79	2.99
Луганськ	1.23	2.06	3.05	4.05	5.46	5.57	5.65	4.99	3.62	2.23	1.26	0.93	3.34
Львів	1.08	1.83	2.82	3.78	4.67	4.83	4.83	4.45	3.00	1.85	1.06	0.83	2.92
Миколаїв	1.25	2.10	3.07	4.38	5.65	5.85	6.03	5.34	3.93	2.52	1.36	1.04	3.55
Одеса	1.25	2.11	3.08	4.38	5.65	5.85	6.04	5.33	3.93	2.52	1.36	1.04	3.55
Полтава	1.18	1.96	3.05	4.00	5.40	5.44	5.51	4.87	3.42	2.11	1.15	0.91	3.25
Рівне	1.01	1.81	2.83	3.87	5.08	5.17	4.98	4.58	3.02	1.87	1.04	0.81	3.01
Суми	1.13	1.93	3.05	3.98	5.27	5.32	5.38	4.67	3.19	1.98	1.10	0.86	3.16
Сімферополь	1.27	2.06	3.05	4.30	5.44	5.84	6.20	5.34	4.07	2.67	1.55	1.07	3.58
Тернопіль	1.09	1.86	2.85	3.85	4.84	5.00	4.93	4.51	3.08	1.91	1.09	0.85	2.99
Ужгород	1.13	1.91	3.01	4.03	5.01	5.31	5.25	4.82	3.33	2.02	1.19	0.88	3.16
Харків	1.19	2.02	3.05	3.92	5.38	5.46	5.56	4.88	3.49	2.10	1.19	0.90	3.26
Херсон	1.30	2.13	3.08	4.36	5.68	5.76	6.00	5.29	4.00	2.57	1.36	1.04	3.55
Хмельницький	1.09	1.86	2.87	3.85	5.08	5.21	5.04	4.58	3.14	1.98	1.10	0.87	3.06
Черкаси	1.15	1.91	2.94	3.99	5.44	5.46	5.54	4.87	3.40	2.13	1.09	0.91	3.24
Чернігів	0.99	1.80	2.92	3.96	5.17	5.19	5.12	4.54	3.00	1.86	0.98	0.75	3.03
Чернівці	1.19	1.93	2.84	3.96	4.54	4.75	4.76	4.40	3.06	2.00	1.20	0.94	2.94

Рисунок 1.1 – Середньодобовий показник інсоляції за спостереженнями NASA за останні 22 роки, кВт· год/м²/день

Окрім цього, було проаналізовано технічні характеристики, типи та параметри сонячних панелей, доступних на ринку України. В результаті аналізу визначено, що ключовими параметрами для прогнозування виробництва електроенергії є потужність панелей та їх тип. Дослідження показало, що тип сонячних панелей суттєво впливає на кількість потенційної виробленої енергії через різні ККД (коефіцієнти корисної дії), у середньому, монокристалічні панелі мають ККД – 21%, полікристалічні – 16%, а аморфні (тонкоплівкові) – 11% [5].

Окрім вище зазначених параметрів, на ефективність роботи фотоелектричних систем, також можуть впливати інші різні фактори. У наукових роботах [6, 7, 8], зазначається, що у цей перелік входять наступні: інтенсивність сонячного випромінювання, кут падіння сонячних променів, географічні координати місця встановлення, орієнтацію панелей у просторі, їх нахил відносно горизонту, затінення, запиленість поверхні панелей, деградацію матеріалів з часом, температуру навколишнього середовища, вологість повітря, а також технічні характеристики інверторів та системи передачі енергії.

Враховуючи потребу в розробці мобільного застосунку, що має бути доступним для широкого кола користувачів без спеціалізованих знань та обладнання, було прийнято рішення використати спрощену методику прогнозування, яка, тим не менш, забезпечить прийнятну точність результатів для побутового використання. Наш підхід передбачає фокусування на ключових параметрах, що мають найбільший вплив на продуктивність сонячних панелей.

Для реалізації прогнозування будемо враховувати такі основні фактори: середньоденну інсоляцію для конкретної області України за обраний місяць, температуру навколишнього середовища, хмарність, потужність фотоелектричних модулів, їх кількість та тип, що визначає їх ККД. Для забезпечення більшої точності прогнозування та інформативності демонстрації результатів ми реалізуємо погодинні розрахунки протягом світлового дня.

Вплив температури повітря будемо вираховувати за формулою 1.1. Таке рішення базується на дослідженнях [9], які показують, що при зниженні температури від оптимальної (25°C - визначений стандарт) з кожним градусом ефективність панелей збільшується на 0,5%, а при підвищенні температури понад оптимальної з кожним градусом, відповідно, відбувається зниження ККД фотоелементів на такий самий показник.

$$T_{impact} = 1 - \frac{k \times (T - T_{opt})}{100}, \tag{1.1}$$

де T_{impact} – коефіцієнт впливу температури на ефективність панелей;

k = 0.5 – температурний коефіцієнт потужності модуля, %;

T_{opt} = 25 – оптимальна температура визначена стандартом,°C;

T – температура навколишнього середовища, °C.

Хмарність істотно впливає на кількість сонячного випромінювання, що досягає поверхні фотоелектричних модулів. Згідно джерела [10], в умовах повністю похмурої погоди енергетичний вихід від панелей становить близько 20% від того, який міг би бути досягнутий при безхмарному небі. Зважаючи на вищесказане, вплив хмарності будемо вираховувати за наступною формулою:

$$C_{impact} = 1 - k \times \frac{C}{100}, \quad (1.2)$$

де C_{impact} – коефіцієнт впливу хмарності на ефективність панелей;

k = 0.8 – коефіцієнт пропорційності, що характеризує ступінь зниження ефективності при збільшенні хмарності;

C – показник хмарності для конкретної локації, %.

Визначивши всі ключові коефіцієнти впливу на ефективність фотоелектричних модулів, пропонуємо спрощену комплексну математичну модель для обчислення прогнозованого обсягу виробленої сонячної електроенергії за одну годину. Ця модель враховує вплив інсоляції, температури, хмарності, технічних характеристик фотоелектричних модулів та інших визначальних параметрів. Загальну формулу для розрахунку погодинного виробництва електроенергії можна представити наступним чином:

$$E_h = I_h \times P \times N \times \eta \times T_{impact} \times C_{impact}, \quad (1.3)$$

де E_h – прогнозована кількість виробленої електроенергії за годину, кВт· год;

I_h – середньодобове значення інсоляції для області, кВт· год/м²;

P – потужність однієї сонячної панелі, Вт;

N – кількість панелей у системі;

η – коефіцієнт корисної дії панелі відповідно до її типу;

T_{impact} – коефіцієнт впливу температури, розрахований за формулою (1.1);

C_{impact} – коефіцієнт впливу хмарності, розрахований за формулою (1.2).

Таким чином, проаналізувавши підходи до прогнозування виробництва сонячної енергії, нами було обрано фізичний метод як найбільш оптимальний для реалізації в мобільному застосунку. Розроблена математична модель враховує ключові фактори впливу на продуктивність фотоелектричних модулів, а запропонована формула (1.3) дозволяє з оптимальною для нас точністю прогнозувати погодинне виробництво електроенергії, що є важливим для оцінки ефективності роботи сонячних електростанцій. Створена модель стане основою для програмної реалізації алгоритму прогнозування у мобільному застосунку, забезпечуючи простоту використання для широкого кола користувачів при збереженні необхідної точності розрахунків.

1.2 Дослідження існуючих застосунків та веб-сервісів

Наразі існує велика кількість сервісів та застосунків для розрахунку сонячної енергії, та не всі вони є безкоштовними у використанні, або ж деякі з них доступні лише клієнтам певних компаній. Проте ми дослідили сервіси та застосунки, які є доступними для нас та не вимагають грошових затрат.

Розглянуто три сервіси для розрахунку виробництва сонячної енергії: PVWatts Calculator, PVGIS (Photovoltaic Geographical Information System) та застосунок mySolarEdge. Ці інструменти пропонують комплексні можливості для аналізу продуктивності сонячних систем у різних регіонах світу.

1.2.1 Сервіс PVWatts Calculator

PVWatts Calculator – веб-сервіс, розроблений Національною лабораторією відновлюваної енергії США (NREL), призначений для оцінки виробництва електроенергії фотоелектричними системами (рис. 1.2). Він дозволяє користувачам визначити потенційне виробництво сонячної енергії на основі місця розташування, параметрів системи та метеорологічних даних.

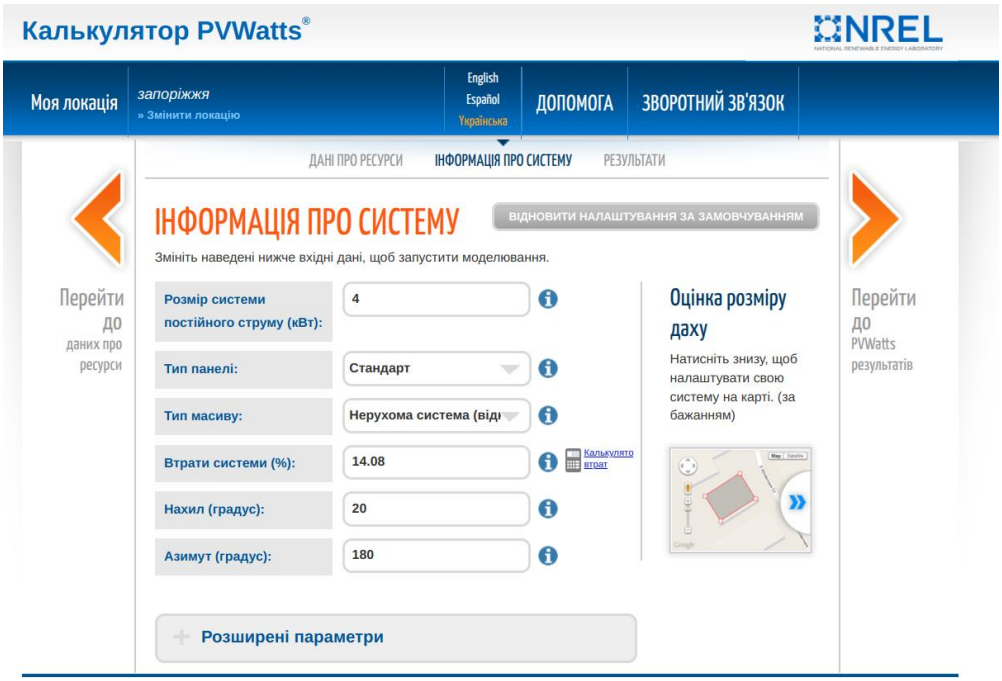


Рисунок 1.2 – Веб-сервіс PVWatts Calculator

Переваги:

– інтуїтивно зрозумілий інтерфейс;
– розрахунок місячних та погодинних результатів та можливість їх експорту у форматі CSV (Comma-Separated Values);
– безкоштовний доступ до всіх функцій;
– має вбудовану розроблену сторінку українською мовою;

Недоліки:

– використання історичних даних, а не даних реального часу;
– можлива недостатня кількість даних про обраний регіон.

Отже, PVWatts Calculator є непоганим інструментом для попереднього оцінювання потенціалу сонячної генерації. Сервіс забезпечує достатню точність для початкового планування, проте, для професійного проектування великих систем може знадобитися більш широкий функціонал з розширеними можливостями моделювання та більша точність розрахунків.

1.2.2 Сервіс PVGIS (Photovoltaic Geographical Information System)

PVGIS – комплексний веб-сервіс для оцінки ефективності фотоелектричних систем, розроблений Об'єднаним дослідницьким центром Європейської Комісії (рис. 1.3). Сервіс надає набір інструментів для розрахунку виробництва сонячної енергії на основі супутникових даних та наземних вимірювань. PVGIS включає кілька модулів: основний калькулятор виробництва, доступ до погодинних метеорологічних даних, генератор типового метеорологічного року та детальну інформацію про сонячну радіацію.

Переваги:

– висока точність даних для європейського регіону;
– детальне врахування втрат енергії (температура, відбиття тощо);
– розширені можливості візуалізації результатів (графіки, карти);
– генерація професійних PDF-звітів (Portable Document Format);

Недоліки:

– складний інтерфейс;
– менша точність даних для деяких неєвропейських регіонів;
– відсутність погодинних результатів.

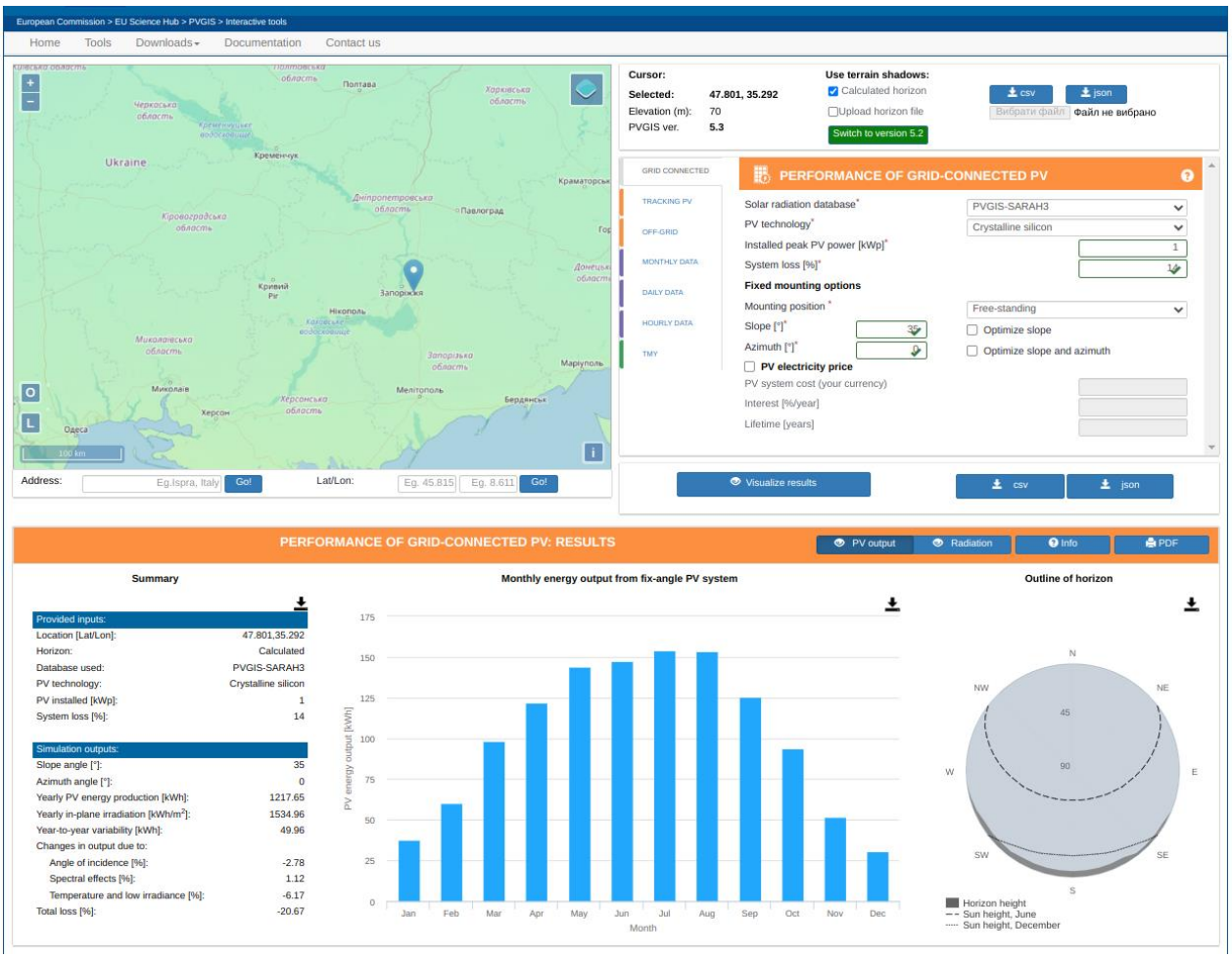


Рисунок 1.3 – Веб-сервіс PVGIS

Отже, PVGIS є потужним інструментом для професійної оцінки продуктивності фотоелектричних систем. Сервіс надає значно більше можливостей для детального аналізу та моделювання порівняно з PVWatts, також він візуалізує результати розрахунків.

1.2.3 Застосунок mySolarEdge

mySolarEdge – спеціалізований застосунок від компанії SolarEdge Technologies, призначений для моніторингу та управління вже встановленими фотоелектричними системами на базі обладнання SolarEdge (рис. 1.4). На відміну від PVWatts і PVGIS, які фокусуються на проектуванні та оцінці потенціалу, mySolarEdge працює з реальними даними існуючих систем. Застосунок доступний як у мобільній версії (iOS, Android), так і у веб-

інтерфейсі, і забезпечує контроль продуктивності системи в режимі реального часу, історичний аналіз даних, діагностику проблем та економічну оцінку.

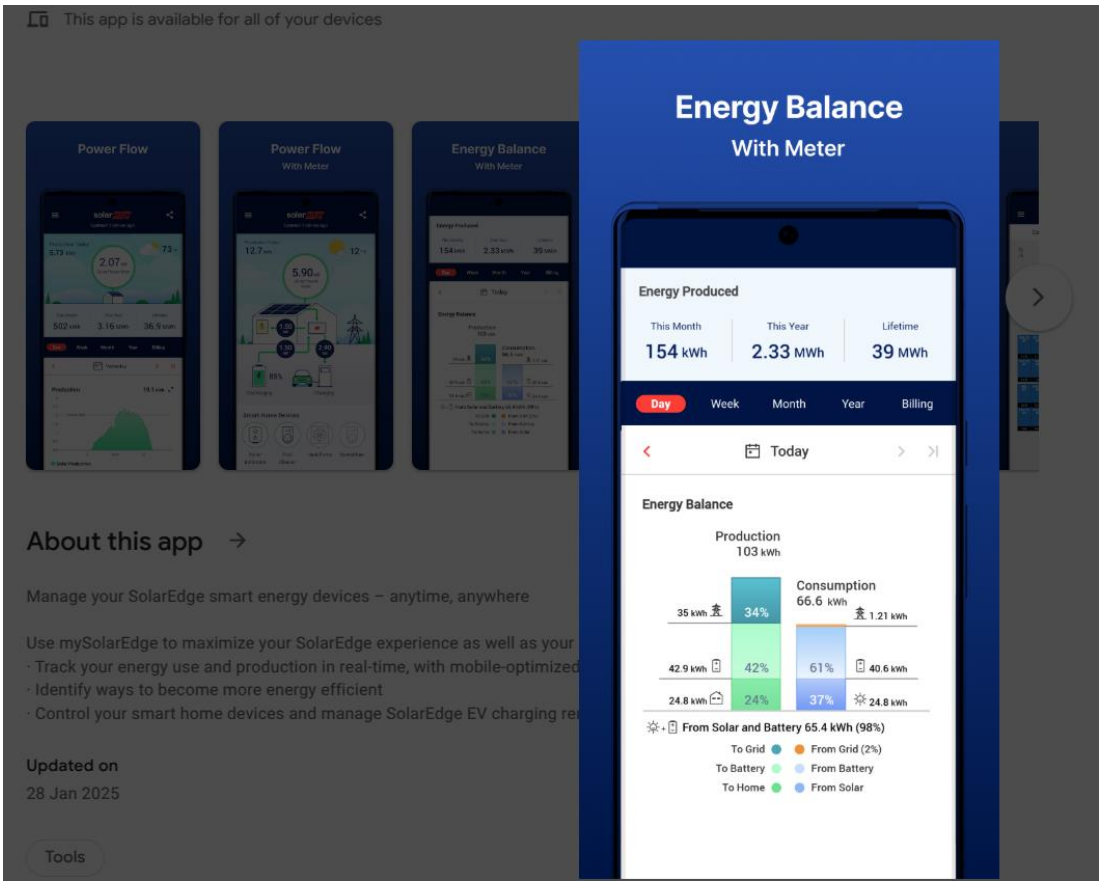


Рисунок 1.4 – Застосунок mySolarEdge

Переваги:

– моніторинг у реальному часі з високою точністю;
– надає детальний аналіз;
– інтегровані інструменти для оптимізації власного споживання;
– зручний мобільний доступ до всіх функцій;
– розширені можливості для аналізу історичних даних;

Недоліки:

– працює виключно з обладнанням SolarEdge;
– орієнтований на моніторинг існуючих систем, а не на проектування нових;
– потребує наявності інтернет-підключення для збору та аналізу даних;

– деякі розширені функції можуть вимагати додаткової оплати.

Можемо зробити висновок, що mySolarEdge є ефективним інструментом для власників сонячних електростанцій на базі обладнання SolarEdge, що дозволяє максимізувати продуктивність системи через детальний моніторинг, діагностику та оптимізацію. Застосунок доповнює інструменти проектування, такі як PVWatts та PVGIS, забезпечуючи контроль над системою протягом всього життєвого циклу.

Для більш наглядного результату проведеного порівняльного аналізу аналогів, створили таблицю 1.1, яка демонструє ключові характеристики та їх значення відповідно до кожного розглянутого продукту.

Таблиця 1.1 – Порівняння аналогів для розрахунку сонячної енергії

Характеристика	**PVWatts Calculator**	**PVGIS**	**mySolarEdge**
Доступність	Безкоштовний, онлайн	Безкоштовний, онлайн	Безкоштовний (тільки для SolarEdge)
Інтерфейс	Простий, інтуїтивний	Більш складний, багато налаштувань	Зручний мобільний додаток та веб-версія
Географічне покриття	Глобальне	Висока точність у Європі	Лише для SolarEdge
Тип даних	Історичні метеодані	Супутникові та наземні вимірювання	Дані в реальному часі
Потенційний розрахунок	Так	Так	Ні (працює тільки з існуючими системами)
Погодинний прогноз	Є (історичні дані)	Відсутній	Є (моніторинг у реальному часі)
Експорт даних	CSV	PDF-звіти	Є аналітика
Графічне представлення	Таблиця	Таблиці, простий графік	Графіки продуктивності
Акаунт	Не потрібен	Не потрібен	Потрібен

Продовження таблиці 1.1

Характеристика	PVWatts Calculator	PVGIS	mySolarEdge
Збереження власних даних	Немає	Немає	Так
Доступ на різних пристроях	Веб-версія	Веб-версія	Веб + мобільний додаток
Розрахунок для кількох панелей	Ні	Ні	Так
Функція розподілу енергії	Немає	Немає	Є

1.3 Вимоги до застосунку

На основі проведеного аналізу методів прогнозування виробництва сонячної енергії та дослідження існуючих рішень, можемо визначити ключові вимоги до розроблюваного мобільного застосунку. Ці вимоги спрямовані на створення продукту, який буде зручним для широкого кола користувачів, надаватиме точні прогнози та міститиме функціональність, що відповідає потребам власників сонячних панелей та станцій в Україні.

Основні функціональні вимоги до застосунку включають:

– реалізація адаптивних, ефективних та досить точних алгоритмів погодинного прогнозування виробітку сонячної енергії, що враховує інсоляцію, температуру, хмарність та технічні характеристики сонячних панелей;
– реєстрація та авторизація користувача, задля збереження даних;
– забезпечення вибору географічного розташування серед областей України для точного визначення параметрів інсоляції;
– введення даних про кількість та потужність сонячних панелей, а також їх тип (монокристалічні, полікристалічні, аморфні);

- можливість комбінування різних типів панелей для розрахунку сумарної кількості виробленої енергії;
- графічне представлення результатів розрахунків у вигляді зручних для сприйняття діаграм;
- відображення сумарного виробництва енергії за різні періоди часу;
- зберігання історії розрахунків для відстеження динаміки роботи;
- створення записів про користувацькі побутові прилади із вказанням їх споживчої потужності, назви та пріоритету при розподілі енергії;
- реалізація алгоритмів розподілення енергії за пріоритетністю між обраними приладами;
- забезпечення зміни даних про прилади та панелі;
- можливість зміни списку обраних приладів для розподілення енергії.

Нефункціональні вимоги до застосунку мають забезпечити:

- безкоштовне використання повної функціональності програми;
- доступність українською мовою для зручності користувачів;
- інтуїтивно зрозумілий інтерфейс;
- мінімальну кількість кроків для отримання результату розрахунків;
- стабільну роботу застосунку без збоїв із забезпеченням коректності розрахунків.

Порівняно з існуючими аналогами (PVWatts Calculator, PVGIS, mySolarEdge), наш застосунок матиме описані нижче переваги.

Спеціалізація на українському ринку з використанням актуальних даних інсоляції для регіонів України та врахуванням місцевих кліматичних особливостей. Наш мобільний застосунок працюватиме з даними реального часу (метеорологічними даними), що дозволить досягти вищої точності прогнозування виробітку енергії порівняно з рішеннями, що використовують лише історичні дані. Користувачі матимуть змогу вільно обирати та змінювати локацію розташування сонячних панелей, а також налаштовувати технічні параметри системи. Застосунок забезпечуватиме актуальну інформацію про

погодинний і денний виробіток сонячної енергії з візуалізацією результатів у вигляді графіків.

Додатковою перевагою нашого застосунку стане функціонал планування енергоспоживання, який дозволить користувачам розподіляти вироблену енергію між різними електроприладами відповідно до їхніх потреб та пріоритетів.

Таким чином, розроблюваний застосунок відповідатиме потребам як початківців, так і досвідчених користувачів сонячних електростанцій, забезпечуючи простоту використання та достатню точність прогнозування виробництва сонячної енергії в умовах України.

1.4 Аналіз методів досліджень і вибір технологій

На цьому етапі роботи визначимо методи дослідження, які дозволять ефективно розробити та вдосконалити мобільний застосунок для розрахунку та розподілу сонячної енергії. Також оберемо відповідні технології для реалізації програми, орієнтуючись на сучасні підходи до розробки програмного забезпечення.

1.4.1 Методи дослідження

Для успішної реалізації проєкту були обрані наступні методи дослідження:

– метод порівняння – вивчення наявних сервісів, які працюють із сонячною енергією, дослідження їхніх функцій, переваг та недоліків. Це дозволить визначити, які підходи вже використовуються та які можливості ще не реалізовані;
– експериментальний метод – тестування роботи застосунку на реальному мобільному пристрої (Android) з використанням

актуальних географічних і метеорологічних даних. Включає перевірку коректності й стабільності роботи застосунку, також правильності обробки даних;

– опитування – вивчення потреб потенційних користувачів, які використовують або планують використовувати сонячні панелі. Це допоможе визначити найважливіші функції застосунку та адаптувати його під реальні запити ринку;
– формалізація – подання ключових розрахунків у вигляді математичних формул і алгоритмів, що забезпечить чіткість та стандартизований підхід до обчислень.

1.4.2 Технології серверної частини Node.js та Express.js

Серверна частина застосунку реалізована на основі технологій Node.js та Express.js. Node.js – це середовище виконання JavaScript, яке дозволяє запускати серверний код без використання браузера [11]. Воно працює на рушії V8 і відрізняється високою продуктивністю завдяки асинхронній неблокуючій обробці запитів.

Express.js – це мінімалістичний та гнучкий веб-фреймворк для Node.js, що надає набір функцій для веб- та мобільних додатків та значно спрощує створення серверної логіки [12]. Він забезпечує зручний механізм маршрутизації, обробку HTTP-запитів та підтримку middleware-функцій, що дозволяє легко реалізувати автентифікацію, логування та інші необхідні функції.

Однією з головних причин вибору цих технологій є їхня гнучкість та висока продуктивність. Оскільки мобільний застосунок працюватиме з великою кількістю запитів у реальному часі, важливо забезпечити їх швидку обробку. Node.js дозволяє створювати легкі та швидкі сервери, що працюють без зайвих витрат ресурсів.

Також важливим фактором є активна підтримка спільноти та велика кількість готових бібліотек, що прискорює розробку. Express.js має просту та зрозумілу структуру, що полегшує роботу над серверною частиною та дозволяє легко додавати новий функціонал у майбутньому.

1.4.3 База даних MongoDB

MongoDB – це документо-орієнтована система керування базами даних, яка зберігає дані у форматі JSON-подібних (JavaScript Object Notation) документів з динамічними схемами [13]. MongoDB належить до класу NoSQL (Not only SQL) баз даних і розроблена для забезпечення високої доступності та горизонтальної масштабованості.

Основна причина вибору MongoDB – її висока продуктивність та можливість горизонтального масштабування. Оскільки система працює з великими обсягами інформації, важливо мати швидкий доступ до даних, а також можливість розподілу навантаження між серверами. Завдяки механізму реплікації MongoDB забезпечує стійкість до збоїв, що підвищує надійність збереження даних.

Ще одна вагома перевага MongoDB – зручна інтеграція з JavaScript-технологіями, зокрема з Node.js. Оскільки серверна частина застосунку розробляється саме на Node.js, використання MongoDB є оптимальним вибором.

Оскільки у застосунку буде динамічний набір параметрів, що змінюються залежно від потреб користувачів, гнучкість MongoDB дозволяє легко зберігати та оновлювати дані без складної міграції. Це спрощує процес розробки та подальшого вдосконалення програми.

1.4.4 Технології клієнтської частини React Native та Expo CLI (Command Line Interface)

Для розробки клієнтської частини мобільного застосунку обрано фреймворк React Native, оскільки він дозволяє створювати застосунки одразу для двох платформ – Android та iOS – використовуючи одну кодову базу [14]. Це значно скорочує час розробки та спрощує подальшу підтримку програми.

React Native базується на бібліотеці React і використовує мову програмування JavaScript, що робить його зручним для інтеграції з серверною частиною, яка написана на Node.js. Завдяки цьому сервер і клієнтська частина можуть ефективно взаємодіяти між собою.

Однією з ключових переваг React Native є підтримка рідних компонентів мобільних платформ, що дозволяє створювати продуктивні застосунки з інтерфейсом, який відповідає стандартам кожної операційної системи.

Разом із React Native використовується Expo CLI – набір інструментів, бібліотек та сервісів, побудованих навколо React Native, який допомагає створювати, запускати та ітеративно вдосконалювати iOS, Android та веб-застосунки, використовуючи однаковий JavaScript код [15].

Expo надає набір готових API для роботи з сенсорами, push-сповіщеннями та іншими можливостями мобільних пристроїв, що дозволяє значно скоротити час розробки. Завдяки цьому можна швидко реалізовувати функціонал без необхідності писати додатковий код. Ще однією вагомою перевагою Expo є можливість тестування застосунку безпосередньо на реальних пристроях без складних налаштувань. Розробляючи програму ми можемо миттєво бачити внесені зміни, що прискорює процес налагодження.

Важливим фактором вибору React Native та Expo є їх активна підтримка з боку спільноти розробників. Ці технології мають відкритий вихідний код, що означає постійний розвиток та покращення. Крім того, завдяки великій кількості готових бібліотек та компонентів, можна легко розширювати функціональність застосунку без необхідності розробляти все з нуля.

Завдяки React Native та Expo розробка мобільного застосунку стає швидшою, ефективнішою та менш витратною. Використання єдиної кодової бази для двох платформ, інтеграція з серверною частиною на Node.js, підтримка рідних компонентів і доступ до готових API дозволяють створити продуктивний, зручний та масштабований застосунок, що відповідатиме сучасним вимогам ринку.

1.5 Постановка завдань дослідження

Для досягнення поставленої мети – розробки мобільного застосунку для розрахунку потенційного добового виробництва сонячної енергії та визначення способів її розподілення між побутовими приладами – склали план, у якому описали основні завдання, які будемо виконувати:

- аналіз існуючих рішень у сфері розрахунку сонячної енергії. Цей пункт передбачає дослідження функціональності наявних мобільних застосунків та веб-сервісів, виявлення їх сильних сторін, технічних обмежень та недоліків. Аналіз користувацького досвіду існуючих рішень, дозволить визначити зручність та привабливість цих програм для створення більш досконалішого застосунку;
- проєктування архітектури застосунку. Необхідно визначити основні компоненти клієнтської та серверної частини, розробити схему їхньої взаємодії та створити оптимальну модель даних. Важливими аспектами є проєктування REST (Representational State Transfer) API для ефективного обміну інформацією та забезпечення стабільної роботи системи;
- інтеграція з зовнішніми сервісами. Для функціонування застосунку будемо використовувати зовнішнє API для отримання прогнозу погоди. Важливо реалізувати ефективну синхронізацію даних із сервісом погоди для актуального прогнозування енерговиробництва.

- розробка алгоритму для прогнозування добового виробництва енергії. Розрахунок потенційної генерації енергії залежить від багатьох факторів: рівня інсоляції, погодних умов, технічних характеристик панелей. Необхідно створити алгоритм, що враховуватиме всі ці параметри та використовуватиме раніше визначені формули 1.1–1.3;
- розробка алгоритму розподілу виробленої енергії. Важливим завданням є створення алгоритму, який враховує пріоритетність електроприладів, задану користувачем, та розподілить доступну енергію між ними. Тобто такий підхід до управління енергоспоживанням повинен бути адаптивним і гнучким, що дозволить користувачам оптимально використовувати вироблену енергію відповідно до своїх потреб;
- створення інтерфейсу та розробка мобільного застосунку. Розробка інтуїтивно зрозумілого інтерфейсу є ключовим завданням. На основі аналізу вже існуючих програм та потреб користувачів, необхідно створити прототипи основних сторінок. Важливо передбачити систему візуалізації результатів у вигляді графіків або діаграм, щоб зробити дані доступними для користувачів з різним рівнем технічної підготовки. Окрім того, використовуючи сучасні технології, потрібно розробити сам мобільний застосунок, створити зручну навігацію, реалізувати потрібний функціонал для кожного елемента та створити локальне сховище;
- реалізація системи персоналізованих налаштувань. Застосунок повинен дозволяти користувачам налаштовувати параметри відповідно до їхніх потреб. Потрібно розробити механізм реєстрації та авторизації, а також систему збереження індивідуальних налаштувань;
- тестування та оптимізація додатку. Тестування є важливим етапом, тому будемо особисто тестувати застосунок на власному девайсі, використовуючи необхідні програми, що дозволять це зробити. Важливо буде перевірити справність роботи всіх розроблених

функцій, коректність відповідей від зовнішніх сервісів, валідацію введених даних та іншої реалізованої логіки.

Розроблений план дозволить поступово вирішити всі завдання та розробити мобільний застосунку для прогнозування і розподілу сонячної енергії, тобто досягти поставленої мети.

2 РОЗРОБКА АРХІТЕКТУРИ ПРОЄКТУ

2.1 База даних

У попередньому розділі було обрано базу даних MongoDB як оптимальне рішення для розроблюваного мобільного застосунку прогнозування виробітку сонячної енергії завдяки гнучкості схеми даних та високій продуктивності при роботі з нерегулярними даними. На відміну від реляційних баз даних, MongoDB як NoSQL-рішення не вимагає чіткого визначення структури даних заздалегідь, що дозволяє легко адаптуватися до змінних вимог проєкту без необхідності перебудови схеми.

Проаналізувавши функціональні вимоги до застосунку, було виділено основні сутності, які потребують зберігання та обробки: користувачі (user), географічні локації (location), регіони (region), сонячні панелі (panel), типи панелей (panelType) та побутові прилади (appliance). Структура бази даних буде проєктуватись з урахуванням необхідності збереження та оновлення даних про користувачів, їхнє обладнання та географічне розташування. Значення сонячної інсоляції для різних областей України та типи панелей з інформацією про їх ККД, як вихідні дані, будуть зберігатись у відповідних колекціях.

Колекція користувачів (user) буде містити основну інформацію про зареєстрованих користувачів системи: унікальний ідентифікатор (id), ім'я (name), хеш паролю (password_hash) та електронну пошту (email). Ця колекція буде ключовою у схемі даних, оскільки пов'язана з колекціями локацій, панелей та приладів через зовнішні ключі.

Колекція областей (region) буде містити інформацію про області України з відповідними даними про місячну (monthlyInsolation) та річну (yearlyInsolation) інсоляцію.

Колекція локацій (location) зберігатиме дані про місця встановлення сонячних панелей користувачів: унікальний ідентифікатор (id), прив'язку до

області (regionId), географічні координати (coordinates), масив даних про щоденний виробіток енергії (dailyEnergyProduced) та посилання на користувача (userId).

Для зберігання інформації про сонячні панелі створимо колекцію panel, яка включатиме ідентифікатор (id), тип панелі (typeId), потужність (power), їх кількість (number) та посилання на користувача (userId). Колекція panelType, у свою чергу, буде містити дані про різні типи сонячних панелей з їх характеристиками, зокрема тип (type) та коефіцієнт ефективності (efficiency).

Колекція побутових приладів (appliance) зберігатиме інформацію про електроприлади користувачів: ідентифікатор (id), назву (name), пріоритет при розподіленні енергії (importance: "високий", "середній", "низький"), споживану потужність (power) та посилання на користувача (userId).

В результаті проєктування було створено UML-діаграму бази даних наведену на рис. 2.1, яка відображає всі сутності та зв'язки між ними. Сутність користувача пов'язана з сутностями локацій, панелей та приладів за допомогою зовнішніх ключів, так само маємо зв’язок між локаціями та регіонами і між панелями та їх типами. Така структура забезпечує ефективне зберігання даних та швидкий доступ до них при виконанні операцій прогнозування виробітку сонячної енергії та розподілу енергії між приладами.

Використання MongoDB дозволяє також ефективно масштабувати базу даних при збільшенні кількості користувачів та обсягу даних без суттєвого зниження продуктивності, що важливо для подальшого розвитку застосунку та розширення його функціональності.

2.2 Архітектура Backend-частини

Backend-частина додатку буде розроблена у вигляді REST API, реалізованого за допомогою Node.js з використанням фреймворку Express.js.

Такий вибір технологій дозволить створити гнучкий та ефективний інтерфейс для взаємодії з клієнтською частиною.

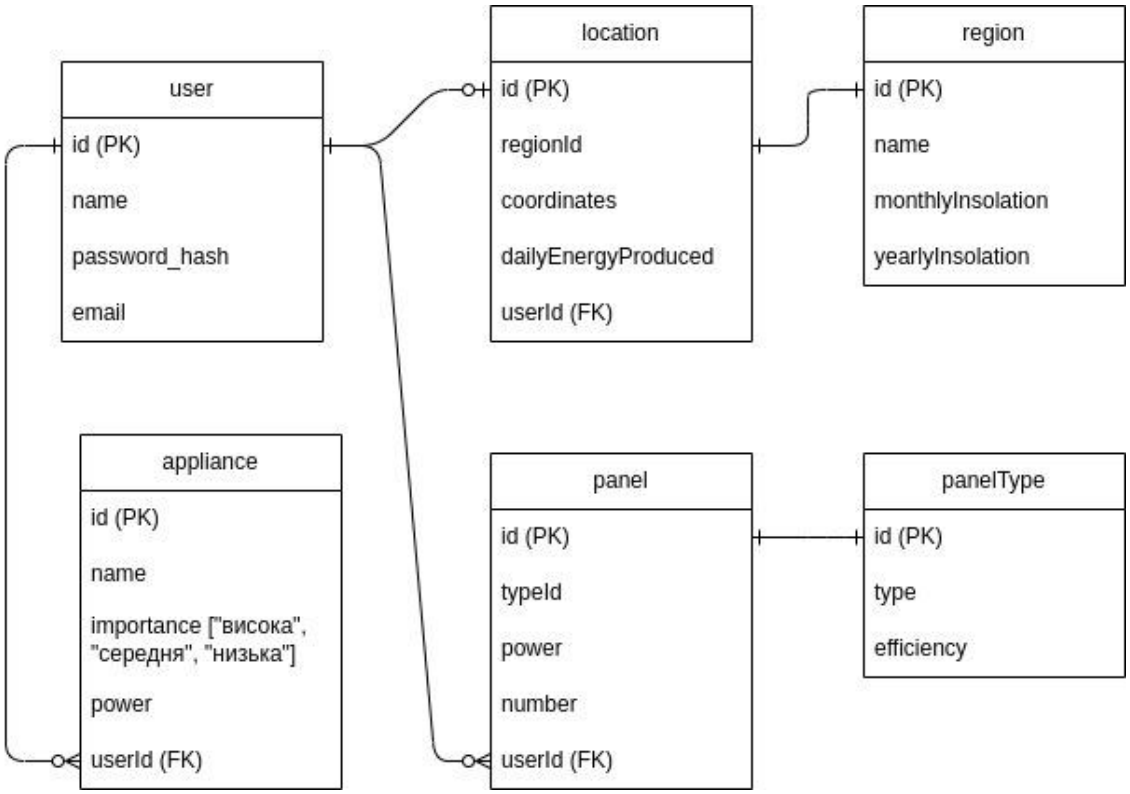


Рисунок 2.1 – UML-діаграма бази даних

Центральним елементом архітектури стане модель "запит-відповідь", де сервер оброблятиме HTTP-запити та повертатиме дані у форматі JSON. Такий підхід забезпечує зручну інтеграцію з клієнтським застосунком і робить API універсальним для розширення. Використання стандартів REST дозволяє створити логічну та зрозумілу структуру ендпоінтів і методів. Послідовність обробки запитів у системі відображена на рис. 2.2.

Так як для зберігання даних буде використано базу даних MongoDB, то взаємодія з нею буде відбуватиметься через бібліотеку Mongoose, яка надасть функціонал для моделювання даних та виконання операцій.

Архітектура серверної частини базуватиметься на принципі модульності. Для кожної сутності системи (user, panel, location, тощо) існуватиме власна модель та набір маршрутів. Така організація коду полегшить підтримку та

масштабування системи, дозволяючи працювати з окремими модулями незалежно.

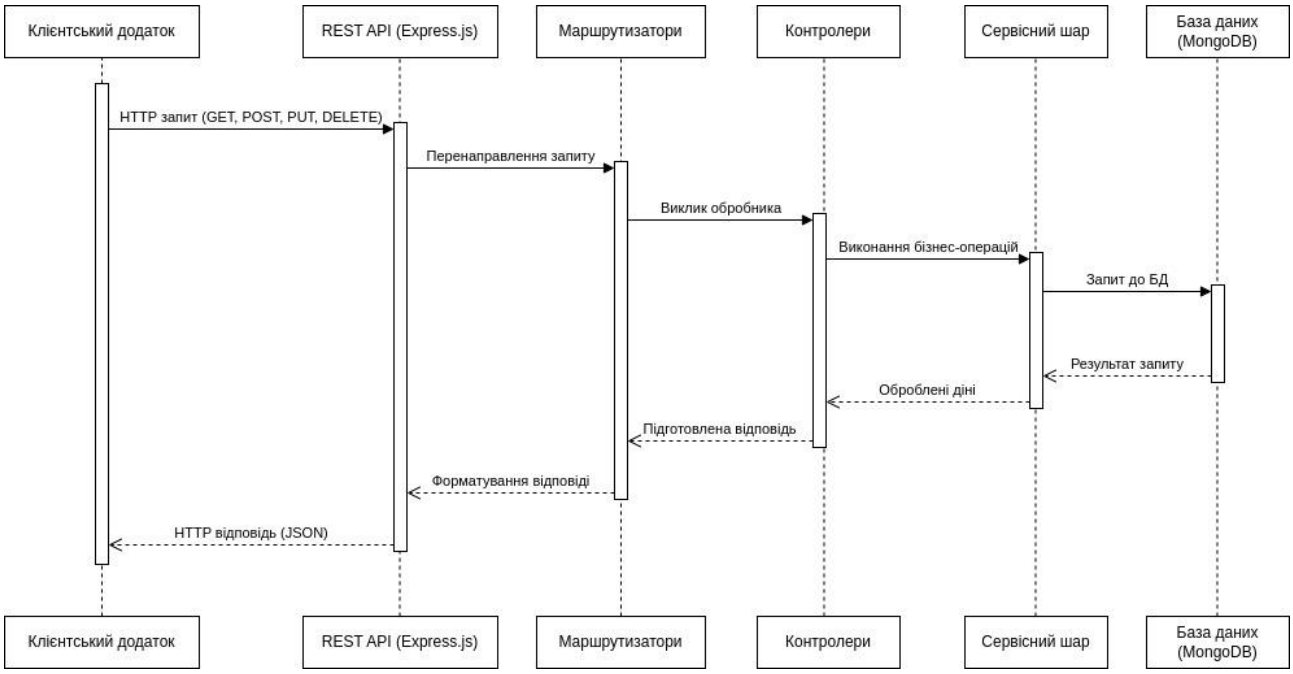


Рисунок 2.2 – Діаграма послідовності обробки запитів на серверній стороні

Система аутентифікації та авторизації користувачів будуватиметься на основі JSON Web Tokens (JWT). Цей механізм забезпечить безпечну передачу інформації про користувача між клієнтом та сервером без необхідності зберігати стан сесії на сервері. Такий підхід є оптимальним для REST архітектури та сприяє масштабованості системи.

Для обробки помилок буде впроваджено централізований механізм, який класифікуватиме помилки за типами та повертатиме клієнту стандартизовані відповіді з відповідними HTTP-статусами. Це покращить відлагодження та користувацький досвід, надаючи зрозумілі повідомлення про помилки.

Безпека системи забезпечуватиметься на кількох рівнях. Окрім JWT для аутентифікації, будуть впроваджені механізми валідації вхідних даних, та хешування паролів. Ці заходи дозволять забезпечити конфіденційність даних користувачів та цілісність системи.

Для підтримки гнучкості налаштувань та полегшення розгортання в різних середовищах, конфігурація системи базуватиметься на змінних

оточення. Цей підхід дозволить змінювати параметри системи без внесення змін у код, що особливо важливо при переході між середовищами розробки, тестування та подальшої експлуатації.

Архітектура передбачатиме асинхронну обробку запитів, що дозволить системі ефективно працювати з великою кількістю одночасних з'єднань. Завдяки подійно-орієнтованій моделі виконання Node.js, сервер оброблятиме багато запитів паралельно без створення окремих потоків.

Структура проєкту слідуватиме принципам розділення відповідальності, де моделі даних, логіка маршрутизації та допоміжні функції будуть чітко відокремлені. Такий підхід полегшить розуміння коду, тестування окремих компонентів та внесення змін у майбутньому.

2.3 Архітектура Frontend-частини

На основі проведеного аналізу вимог, у розділі 1.3, була розроблена діаграма класів аналізу (див. рис. 2.3), яка відображає концептуальну структуру системи нашого застосунку. Діаграма включає три основні види класів аналізу: граничні класи, що відповідають за взаємодію користувача з системою; керуючі класи, що координують роботу системи; та класи-сутності, що представляють довгоживучу інформацію системи. Створена модель аналізу дозволила уточнити внутрішню архітектуру системи, визначити основні компоненти та взаємозв'язки між ними, що стало підґрунтям для подальшого проєктування та розробки застосунку.

Frontend-частина розробляємого мобільного застосунку буде реалізована з використанням фреймворку React Native та інструментарію Expo, що дозволить створити універсальний крос-платформний мобільний додаток з високою продуктивністю та позитивним користувацьким досвідом. Архітектурне рішення базується на принципах компонентно-орієнтованого

підходу, що забезпечить модульність, перевикористання коду та спрощення процесу розробки.

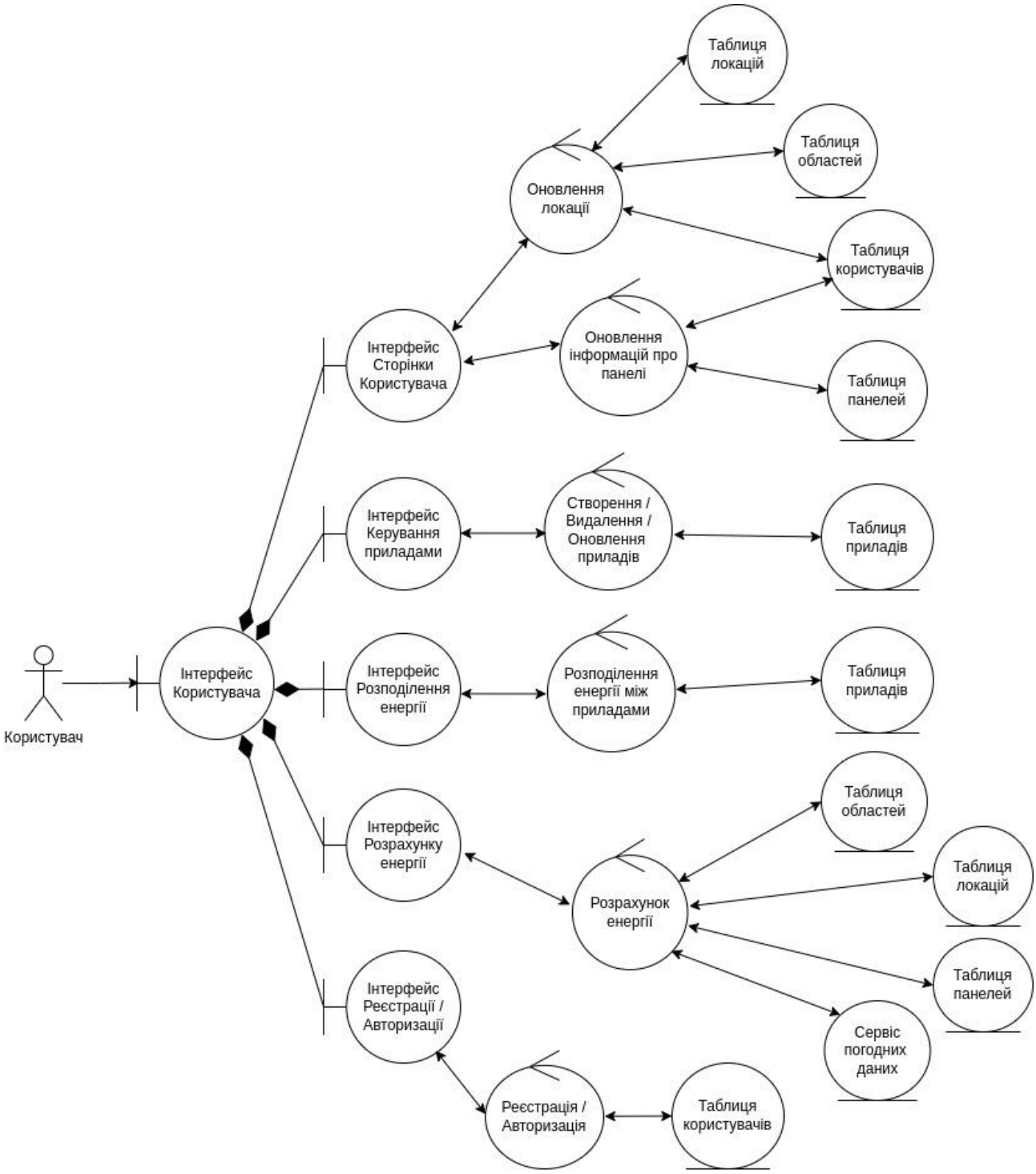


Рисунок 2.3 – Діаграма класів аналізу (Analysis-level class diagram)

Центральним елементом архітектури виступатиме навігаційна система, реалізована за допомогою бібліотеки Expo Router. Така структура забезпечить

інтуїтивну навігацію між екранами додатку, дозволяючи користувачам легко переміщуватися між інтерфейсами авторизації та сторінками розрахунку енергії, пристроїв, розподілу енергії й профілю користувача. Кожен маршрут матиме власний компонент, відповідальний за відображення конкретного екрану, що відповідає принципу єдиної відповідальності.

Для управління станом додатку буде впроваджено бібліотеку Zustand, яка забезпечить ефективний та легкий у використанні механізм керування даними. На відміну від більш складних рішень, таких як Redux, Zustand надасть більш прямолінійний підхід до управління станом, зменшуючи кількість шаблонного коду та спрощуючи архітектуру. Цей вибір обумовлений потребою у гнучкій та легкій системі, здатній забезпечити реактивність інтерфейсу без надмірних накладних витрат.

Система аутентифікації базуватиметься на JWT-токенах, які зберігатимуться у захищеному сховищі AsyncStorage. Це забезпечить збереження сесії користувача між запусками додатку та дозволить реалізувати механізм автоматичного входу. Для керування формами аутентифікації (вхід та реєстрація) буде використано бібліотеку React Hook Form у поєднанні з Yup для валідації даних, що значно спростить роботу з формами та забезпечить надійну перевірку вхідних даних.

Інтерфейс застосунку базуватиметься як на компонентах бібліотек React Native Paper та React Native Elements, які надають готові UI-компоненти (User Interface), що відповідають сучасним стандартам дизайну, так і на користувацьких компонентах, що розроблятимуться з використанням чистого React Native із застосуванням необхідних стилів. Для створення складних анімацій та інтерактивних елементів буде використана бібліотека Reanimated.

Для візуалізації даних про енергоспоживання та ефективність сонячних панелей буде впроваджено бібліотеку React Native Gifted Charts, яка дає широкий вибір інтерактивних графіків та діаграм. Ця бібліотека дозволить наочно представити тенденції зміни значень даних, допомагаючи користувачам аналізувати ефективність своїх сонячних систем. Для роботи з географічними

даними та відображення розташування панелей буде використано бібліотеку React Native Maps.

Архітектура frontend-частини передбачатиме ефективну обробку асинхронних операцій, таких як запити до API, визначення геолокація та інші. Для реалізації таких процесів застосовуватимуться асинхронні React-хуки та шаблони обробки станів завантаження і помилок. Це дозволить зберігати стабільність інтерфейсу та забезпечити нормальну взаємодію користувача з додатком навіть при поганому інтернет-з'єднанні або затримках у відповіді сервера. Для сповіщення користувача про різні події та статуси операцій буде використано функціонал бібліотеки React Native Toast Message.

Структура проєкту організована відповідно до парадигми feature-based architecture, де код групується за функціональними можливостями, а не технологічними шарами. Ключові директорії проєкту включатимуть app (маршрути та екрани), components (перевикористовувані UI-компоненти), hooks (користувацькі React-хуки), assets (шрифти, зображення, налаштування теми), constatnts (глобальні незмінні дані), services (взаємодія з API), store (управління станом) та utils (допоміжні функції).

Для забезпечення якості коду буде використано набір інструментів для статичного аналізу та форматування, включаючи ESLint та Prettier. Ці інструменти дозволять підтримувати визначений правилами стиль коду та виявляти потенційні проблеми на ранніх етапах розробки. Конфігурація ESLint включатиме правила, специфічні для React Native, що допоможе дотримуватися найкращих практик та уникати поширених помилок при розробці мобільних додатків.

Реактивне оновлення інтерфейсу при зміні даних забезпечуватиметься завдяки використанню хуків React та селекторів Zustand. Цей підхід дозволить компонентам автоматично реагувати на зміни у стані додатку, оновлюючись лише коли це необхідно, що позитивно вплине на продуктивність та плавність роботи інтерфейсу. Така архітектура сприятиме створенню інтуїтивно зрозумілого та позитивного користувацького досвіду, який є важливим

аспектом для успішного впровадження додатку серед людей, які планують або вже використовують сонячні панелі.

2.4 Організація зв’язку між серверною та клієнтською сторонами

Для реалізації комунікації між React Native додатком та серверною частиною на базі Express.js обрано бібліотеку Axios, яка стане основним інструментом виконання HTTP-запитів. Axios має наступні переваги над стандартним Fetch API: гнучкий контроль запитів і відповідей, деталізована інформація про помилки, автоматичне перетворення JSON-даних, а також спрощене додавання авторизаційних токенів до заголовків запитів, що є важливим компонентом при імплементації JWT-автентифікації.

Структура API-викликів буде організована відповідно до модульного принципу, де окремі сервіси відповідатимуть за взаємодію з відповідними ресурсами сервера. Такий підхід забезпечуватиме чітке розмежування відповідальності та полегшить підтримку коду в процесі розвитку проєкту.

На етапі розробки та тестування API-взаємодії буде використано інструмент Postman. За допомогою цього інструменту буде створено набір тестових запитів для перевірки роботи всіх ендпоінтів сервера. Postman дозволить швидко виявляти та виправляти помилки в API ще до початку інтеграції з мобільним додатком, що значно пришвидшить процес розробки. Крім того, створені в Postman колекції запитів можуть бути використані для подальшого тестування при внесенні змін у серверну частину.

Для підвищення надійності комунікації між клієнтом і сервером буде впроваджено механізми обробки помилок. На клієнтській стороні за допомогою перехоплювачів Axios буде реалізовано централізовану систему обробки помилок, яка класифікуватиме різні типи мережевих та серверних помилок та відображатиме відповідні повідомлення користувачам.

3 РОЗРОБКА АЛГОРИТМІВ ТА КОДОВА РЕАЛІЗАЦІЯ ЗАСТОСУНКУ

3.1 Розробка алгоритму прогнозування добового виробництва енергії

У розроблюваному мобільному застосунку для аналізу потенційного добового виробництва сонячної енергії основним обчислювальним рушієм є алгоритм розрахунку енергії, що виробляє система сонячних панелей впродовж дня. За основу розробки цього алгоритму було взято формулу погодинного виробництва електроенергії (формула 1.3), яку ми описали у розділі 1.1. У процесі реалізації було враховано вплив різних факторів, зокрема рівень сонячної інсоляції, тип та ефективність сонячних панелей, хмарність, а також температурні коливання, що можуть впливати на продуктивність фотоелектричних елементів.

Алгоритм реалізовано у вигляді власного React-хука useSolarEnergyCalculator, який обчислює прогнозовану генерацію електроенергії на основі параметрів користувача та погодних даних (додаток А). Він складається з декількох логічних етапів: визначення ефективності кожного типу сонячних панелей, розрахунку потужності встановлених панелей, отримання даних про інсоляцію та погодних параметрів, а також проведення погодинного аналізу виробництва енергії.

На першому етапі алгоритм отримує список встановлених сонячних панелей із usePanelsStore та дані про їх типи з usePanelTypesStore. Оскільки різні панелі мають різну ефективність, спочатку відбувається їх групування за типом: монокристалічні, полікристалічні та аморфні. Для кожного типу визначається коефіцієнт ефективності, який зберігається у змінній efficiency.

Наступним кроком є обчислення загальної потужності панелей. Алгоритм перебирає всі додані панелі, визначає їх кількість та номінальну потужність,

після чого множить ці показники на відповідний коефіцієнт ефективності, отриманий на попередньому етапі. Результат обчислень зберігається у змінній panelsPower, яка містить три значення – окремо для кожного типу панелей.

Після цього застосунок отримує значення денної середньої місячної інсоляції для поточного місця розташування. Ці дані беруться із useLocationStore і зберігаються у змінній insolation. Вони є ключовими для розрахунку кількості енергії, яку можуть виробити панелі протягом дня.

Основна частина алгоритму реалізована у функції calculateHourlyEnergy. Вона приймає масиви з даними про погодинну хмарність (hourlyClouds), температурні показники (hourlyTemp), а також час сходу (sunrise) і заходу (sunset) сонця. Ця функція працює за циклічним принципом, проходячи по кожній годині від світанку до заходу та виконуючи послідовні розрахунки енергії.

На кожному часовому відрізку алгоритм визначає рівень хмарності, який впливає на ефективність генерації. Використовується формула 1.2, яка у програмі має вигляд: cloudImpact = 1 - CLOUD_COEF * (cloudiness / 100), де CLOUD_COEF є змінною, що визначає, наскільки сильно хмарність зменшує виробництво енергії. Це дозволяє більш гнучко налаштовувати вплив цього фактора, наприклад, адаптуючи його до різних регіональних умов або оновлюючи значення в майбутніх версіях програми.

Додатково враховується температура, яка також може знижувати продуктивність панелей. Це реалізовано через корекційний коефіцієнт tempImpact, який базується на відхиленні температури від стандартного значення TEMP_STANDART. Вплив температури обчислюється за формулою 1.1, яка має наступний вигляд у коді програми: 1 - (TEMP_COEF * (hourlyTemp[hour] - TEMP_STANDART)) / 100, де TEMP_COEF визначає, наскільки сильно змінюється ефективність панелей залежно від температури навколишнього середовища.

З урахуванням цих факторів обчислюється кінцева кількість енергії, яку виробляє кожен тип панелей за конкретну годину. Вона розраховується за

формулою energy = insolation * cloudImpact * tempImpact * panelsPower, де додатково враховується частка часу, що припадає на кожну годину. Отримані значення зберігаються у масиві hourlyEnergy, де кожен елемент містить часовий інтервал, рівень хмарності, температуру та розраховану кількість виробленої енергії.

Щоб забезпечити коректну візуалізацію даних у графіку, до масиву додається стартова точка з нульовою енергією, що відповідає моменту сходу сонця. Це потрібно для плавного відображення початку генерації. Всі підсумкові розрахунки зберігаються у стані allEnergy, що дозволяє застосунку оперативно оновлювати загальну суму виробленої енергії та відображати її у відповідних компонентах інтерфейсу.

Таким чином, алгоритм забезпечує деталізований погодинний аналіз генерації електроенергії, враховуючи основні зовнішні фактори. Завдяки гнучкій реалізації його можна легко адаптувати під різні географічні умови, типи панелей та додаткові параметри, що можуть бути корисними для користувачів, які планують або вже використовують сонячну енергетику.

3.2 Розробка алгоритму розподілу енергії між приладами

Для оптимального використання доступної енергії між побутовими пристроями було розроблено алгоритм розподілу, який враховує як пристрої з фіксованим часом роботи, так і ті, що можуть працювати гнучко в межах доступної потужності. Основна мета алгоритму — рівномірно розподілити енергію серед пристроїв, враховуючи їхню важливість, потужність та наявні обмеження.

Алгоритм реалізований у вигляді функції calculateDistribution, яка приймає на вхід список вибраних пристроїв та загальну кількість енергії, яку потрібно розподілити (додаток Б). Першим кроком є перевірка вхідних параметрів: якщо список пристроїв порожній або доступна енергія не задана,

функція одразу повертає порожній масив. Це запобігає помилкам у подальших розрахунках та забезпечує коректну обробку вихідних даних.

Далі виконується перетворення значення доступної енергії з кіловат у вати. Потім здійснюється розподіл пристроїв на дві основні категорії: ті, що мають фіксований час роботи (fixedDevices), та ті, що можуть змінювати свою тривалість роботи (dynamicDevices). Пристрої з фіксованим часом мають пріоритет, оскільки вони повинні працювати визначену кількість годин або хвилин на день.

Для кожного пристрою з цієї групи розраховується спожита енергія на основі його потужності та тривалості роботи. Використовується формула deviceEnergy = (power * (workingHours * 60 + workingMinutes)) / 60 / 1000, що дозволяє перевести потужність пристрою у відповідну кількість спожитої енергії. Після обчислення загального енергоспоживання фіксованих пристроїв алгоритм визначає залишкову енергію, яку можна розподілити між пристроями, які залишилися.

Якщо після врахування приладів із зазначеним часом доступної енергії не залишається, всі інші пристрої отримують нульовий час роботи та енергоспоживання. У такому випадку алгоритм завершує роботу і повертає оновлений список девайсів із відповідними нульовими параметрами. Якщо ж залишкова енергія є, розпочинається її розподіл між пристроями без фіксованого часу.

Для цього спочатку визначається максимальний можливий час роботи кожного пристрою (maxWorkingHours і maxWorkingMinutes) шляхом ділення залишкової енергії на потужність пристрою. Це дозволяє дізнатися, скільки максимум часу може працювати пристрій за наявних умов.

Далі пристрої групуються за рівнем важливості: висока, середня та низька. Це необхідно для пріоритетного розподілу енергії між пристроями, які є більш критичними для роботи. Для кожної групи встановлюються коефіцієнти розподілу: 50% для високої важливості, 35% для середньої та 15% для низької.

Якщо певні групи відсутні, коефіцієнти коригуються відповідно, щоб у сумі вони складали 100%.

Після цього залишкова енергія розподіляється між групами пристроїв пропорційно їхнім коефіцієнтам. Отримана для кожної групи енергія рівномірно розподіляється між пристроями, що до неї входять. Далі для кожного пристрою виконується розрахунок часу роботи на основі виділеної енергії за допомогою функції calculateWorkingTime. Вона визначає години та хвилини роботи пристрою за формулою workingTime = allocatedEnergy / power, що дозволяє точно визначити, скільки часу пристрій може працювати, виходячи з отриманої енергії.

Оновлені дані зберігаються у новому масиві, що містить як пристрої з фіксованим часом, так і пристрої, яким було надано певну частку доступної енергії. У підсумку алгоритм дозволяє ефективно розподілити ресурси, забезпечуючи роботу найважливіших пристроїв та рівномірний розподіл енергії між рештою приладів.

3.3 Реалізація Backend-частини

Реалізація серверної частини проєкту базується на використанні Node.js з фреймворком Express і базою даних MongoDB. Основні компоненти включають моделі даних, маршрутизатори для обробки HTTP-запитів, middleware для автентифікації та обробки помилок, а також утиліти для валідації. Взаємодія з базою даних здійснюється за допомогою бібліотеки Mongoose.

Основний файл app.js (додаток В) відповідає за ініціалізацію серверу, підключення бази даних та реєстрацію middleware. Використовується dotenv для керування змінними середовища, cors для безпечного доступу до API з інших доменів, morgan для логування запитів та authJwt для перевірки автентифікації. Крім того, реалізований загальний обробник помилок, який

дозволяє стандартизовано відповідати на можливі проблеми при виконанні запитів.

Для кожної сутності у базі даних створені відповідні моделі. Наприклад, модель appliance.js (див. рис. 3.1) визначає схему приладів, зокрема їхню назву, потужність, рівень важливості (пріоритет) та зв’язок із користувачем через userId. Використовується механізм валідації для перевірки правильності даних, таких як допустимі діапазони потужності пристроїв. Подібним чином реалізовані й інші моделі, що відповідають за панелі, типи панелей, локації, регіони та користувачів.

```
const mongoose = require("mongoose");  870.8k (gzipped: 233.1k)
const { Schema, model, Types } = mongoose;

const importanceType = {
  high: "висока",
  medium: "середня",
  low: "низька",
};

const applianceSchema = new Schema({
  name: {
    // Назва приладу
    type: String,
    required: true,
  },
  power: {
    // Значення потужності
    type: Number,
    required: true,
    min: [0, "Значення потужності повинно бути більшим за 0"],
    max: [5000, "Значення потужності повинно бути меншим за 5000"], // Максимальна потужність приладу у Вт
  },
  importance: {
    // Рівень важливості приладу
    type: String,
    enum: Object.values(importanceType),
    required: true,
    default: importanceType.medium,
  },
  userId: {
    // Посилання на документ у колекції users
    type: Types.ObjectId,
    ref: "User",
    required: true,
  },
});

Tabnine | Edit | Test | Explain | Document
applianceSchema.virtual("id").get(function () {
  return this._id.toHexString();
});

applianceSchema.set("toJSON", {
  virtuals: true,
});

exports.Appliance = model("Appliance", applianceSchema);
```

Рисунок 3.1 – Вміст файлу appliance.js

Кожна модель має відповідний маршрутизатор, що забезпечує обробку запитів до API. Наприклад, у маршрутизаторі для приладів реалізовані роутери для отримання всіх приладів, додавання нових, оновлення та видалення (додаток Г). Використовуються методи find(), save(), findByIdAndUpdate() та findByIdAndDelete(), що дозволяють ефективно взаємодіяти з базою даних. Таким же чином, за відмінності лише певних роутерів, реалізовані і маршрутизатори для інших моделей, що відповідають колекціям у БД (базі даних). Підключення маршрутизаторів здійснюється у app.js шляхом додавання відповідних middleware для кожного API-ендпоінту (додаток В).

Для забезпечення автентифікації та авторизації використовується middleware authJwt, що перевіряє JWT-токен у заголовках запитів. Якщо токен відсутній або недійсний, сервер повертає відповідну помилку, і запит не виконується. Крім того, реалізований errorHandler, який дозволяє обробляти винятки та повертати уніфіковані відповіді у разі помилок, що спрощує діагностику та підтримку API.

Для збереження та обробки конфіденційних даних, таких як параметри підключення до бази даних, використовується .env файл. MongoDB підключається через Mongoose, а сам процес підключення реалізований у app.js (додаток В). У разі успішного з’єднання виводиться відповідне повідомлення в консоль, а у разі помилки — відбувається її обробка.

Окремо реалізовані утиліти для валідації вхідних даних у каталозі utils. Вони дозволяють перевіряти коректність параметрів перед додаванням чи оновленням записів у базі. Це зменшує ризик помилок при роботі сервера та дозволяє повертати зрозумілі повідомлення користувачам API.

Таким чином, серверна частина забезпечує модульну, гнучку та захищену архітектуру, що дозволяє легко розширювати функціональність проєкту, додаючи нові моделі та маршрути. Використання middleware, механізмів автентифікації та валідації даних робить API стабільним та безпечним для взаємодії з клієнтською частиною.

3.4 Розробка Frontend-частини

Розробка frontend-частини мобільного застосунку здійснювалася з використанням фреймворку React Native та інструментарію Expo. Основний принцип побудови архітектури ґрунтувався на компонентному підході, що дозволило створювати перевикористовувані елементи інтерфейсу та підтримувати їх у чистому й структурованому вигляді.

Застосунок організовано за feature-based підходом, що передбачає групування коду за функціональністю та спрощує підтримку та масштабування проєкту. Основні каталоги включають: app (сторінки), components (UI-компоненти), hooks (власні хуки), services (робота з API), store (керування станом через Zustand), utils (допоміжні функції), assets (зображення, шрифти, тема), constants (глобальні змінні).

У наступних підрозділах детальніше розглянемо важливі аспекти реалізації інтерфейсу, включаючи навігацію, відображення карти, обробку даних та реалізацію UI-компонентів.

3.4.1 Навігація та організація сторінок

У розроблюваному мобільному застосунку реалізація навігації відбувається за допомогою бібліотеки Expo Router, що забезпечує файлово-орієнтовану систему навігації. Це дозволяє організовувати маршрути, використовуючи структуру директорій, де кожен файл або каталог відповідає певному екрану або групі екранів.

При старті застосунку Expo Router автоматично визначає початковий маршрут, шукаючи перший index.jsx файл, який відповідає кореневому URL «/». У нашому проєкті файл app/index.jsx виконує функцію початкового маршруту та визначає, на який екран спрямувати користувача залежно від його

статусу авторизації. Якщо користувач увійшов у систему, відбувається перенаправлення на вкладку home, якщо ні – на auth.

Файл _layout.jsx виконує роль контейнера для всіх сторінок. Це своєрідний кореневий макет, який являє собою точку входу для навігації (див. рис. 3.2). Він також ініціалізує систему тем та сповіщень у застосунку. У ньому використовується Stack навігації, що дозволяє керувати переходами між екранами без явного налаштування маршрутів.

```
export default function Layout() {
  return (
    <PaperProvider theme={MyLightTheme}>
      <MainLayout />
      <Toast config={toastConfig} position="top" />
    </PaperProvider>
  );
}

Tabnine | Edit | Explain | Complexity is 8 It's time to do something...
const MainLayout = () => {
  useAuthHandler();
  const [fontsLoaded, setFontsLoaded] = useState(false);

  useEffect(() => {
    const loadResources = async () => {
      await loadFonts();
      setFontsLoaded(true);
    };
    loadResources();
  }, []);

  if (!fontsLoaded) {
    return <LoadingScreen />;
  }

  return <Stack screenOptions={{ headerShown: false }}></Stack>;
};
```

Рисунок 3.2 – Вміст файлу app/_layout.jsx

Головні вкладки знаходяться у папці app/(tabs) й містять окремі екрани, такі як home, devices, energy_distribution та profile. Вони організовані як окремі каталоги, що дозволяє легко розширювати функціонал без зміни навігаційної логіки.

Для реалізації екранної навігації використовується власний анімований компонент MyTabBar. Він відповідає за нижню панель вкладок, змінюючи положення індикатора активної сторінки під час перемикання між екранами

застосунку. Анімація реалізована за допомогою useSharedValue та withSpring(), що забезпечує плавні переходи.

Окремо виділена структура для аутентифікації в директорії app/auth/. Вона містить екрани login та register, а також загальний файл index.js, який спрямовує користувача на потрібний розділ. Завдяки цьому система аутентифікації працює окремо від основних вкладок.

Отже, така організація дозволяє підтримувати чітку структуру, спрощує розширення застосунку та забезпечує зручну роботу з навігацією, використовуючи автоматичне визначення маршрутів за файловою структурою.

3.4.2 Відображення карти та робота з геоданими

Для можливості користувачеві самостійно та інтерактивно обирати місце розташування своїх сонячних панелей, ми інтегрували карту у наш додаток. Цей функціонал реалізовано за допомогою бібліотеки React Native Maps, яка дозволяє взаємодіяти з картою та маркером на ній.

Основним способом вибору локації є ручне встановлення координат, яке реалізовано у власному компоненті Map (додаток Д). Користувач має можливість переміщати маркер на карті, для чого використовується компонент Marker, що підтримує подію onDragEnd, яка фіксує нові координати після завершення переміщення. Такий підхід дозволяє йому точно вказати місце розташування сонячних панелей забезпечуючи максимальну гнучкість, оскільки користувач самостійно визначає потрібну точку без введення числових координат.

Альтернативним способом є автоматичне визначення місцезнаходження, що виконується через Location API. Використовуючи вбудовані механізми бібліотеки Expo Location (див. рис. 3.3), додаток отримує поточні координати пристрою, які одразу передаються у стан програми. Це дозволяє користувачеві швидко вказати своє місце розташування без взаємодії з картою.

```
export const getCurrentPosition = async () => {
  const { coords } = await Location.getCurrentPositionAsync();
  return coords;
};
```

Рисунок 3.3 – Функція визначення координат з використанням Expo Location

Після визначення локації система виконує реверсне геокодування через функцію getRegionName() (див. рис. 3.4), яка спочатку перевіряє, чи координати належать до Автономної Республіки Крим, і у такому випадку повертає відповідну назву. В іншому разі функція використовує Location.reverseGeocodeAsync() для отримання адміністративного регіону, після чого зіставляє його з локалізованими назвами областей у створеному власноруч словнику regionsUkr. Якщо геокодування проходить успішно, координати разом із назвою регіону зберігаються у глобальному стані додатку через useLocationStore() та передаються до бази даних для подальшого використання.

```
import * as Location from "expo-location";
import { regionsUkr } from "../../constants/ukrainianRegions";

Complexity is 12 You must be kidding
export const getRegionName = async (latitude, longitude) => {
  if (
    latitude >= 44.3864 &&
    latitude <= 46.1927 &&
    longitude >= 32.4919 &&
    longitude <= 36.6209
  ) {
    return "Автономна Республіка Крим";
  }

  const address = await Location.reverseGeocodeAsync({ latitude, longitude });
  if (!address || address.length === 0) {
    throw new Error("Не вдалося визначити область для заданих координат.");
  }

  const { region } = address[0] || {};

  return regionsUkr[region] || null;
};
```

Рисунок 3.4 – Функція getRegionName()

Окрім основного функціоналу вибору локації, у додатку передбачені додаткові можливості, що покращують взаємодію з картою. Зокрема, реалізовано функцію скидання карти, яка повертає маркер у початкове положення, дозволяючи користувачеві швидко змінити вибране місце. Також передбачено підтвердження місцезнаходження, яке виконується через діалогове вікно, що відображає вибрані координати та назву області.

Окрему увагу приділено обробці помилок, які можуть виникати під час взаємодії з геоданими. Наприклад, якщо з якихось причин неможливо визначити назву регіону для заданих координат, система повідомляє про це користувача, пропонуючи можливі варіанти виправлення.

Завдяки використанню інтерактивної карти та підтримці автоматичного визначення місцезнаходження через Location API, додаток забезпечує зручний і гнучкий механізм вибору географічних координат. Це підвищує точність введених даних та покращує загальний користувацький досвід.

3.4.3 Запити до API та обробка даних

У системі реалізували механізм взаємодії з API для отримання, додавання, зміни та видалення даних. Для виконання HTTP-запитів використовуємо бібліотеку Axios, яку налаштовуємо через окремий конфігураційний файл axiosConfig.js. У цьому файлі встановлюємо базову URL-адреса API, яка формується на основі змінних середовища.

Для автентифікації запитів використовується механізм додавання токена доступу до заголовків кожного запиту. Токен зберігається у локальному сховищі (asyncStorage) і, якщо він є, додається до заголовка Authorization у форматі Bearer token.

Також налаштували обробку помилок у відповідях API. Якщо сервер повертає статус 401 Unauthorized, що означає недійсний або відсутній токен, система автоматично виконує вихід користувача. Це реалізовано через функцію handleLogout, яка видаляє токен та інші дані користувача з локального сховища,

а також сповіщає інші частини програми про вихід. Такий підхід забезпечує безпеку взаємодії з API та унеможливлює доступ до захищених ресурсів без авторизації.

Для роботи з конкретними сутностями API передбачені окремі модулі, що містяться у папці apis:

– panels.js – обробка запитів, пов'язаних з сонячними панелями (отримання списку панелей користувача, додавання, редагування та видалення панелі);
– auth.js – виконання запитів, пов'язаних з автентифікацією користувача (вхід, вихід, оновлення токена тощо);
– devices.js – керування пристроями;
– panelTypes.js – отримання та керування типами панелей;
– locations.js – обробка запитів, пов'язаних з геолокацією користувача.

Крім того, у файлі weather.js реалізували запити до зовнішнього сервісу погоди OpenWeather Map [16]. Отримана відповідь містить прогноз погоди, включаючи інформацію про хмарність, температуру та час сходу і заходу сонця. Дані отримуємо у форматі JSON, після чого виконуємо їх обробку та використовуємо для відображення погодних умов, а також розрахунку ефективності роботи сонячних панелей.

Таким чином, архітектура API-запитів у системі забезпечує структуроване та ефективне управління даними через окремі модулі, що спрощує розширюваність та підтримку коду.

3.4.4 Управління станом за допомогою Zustand

У нашому мобільному застосунку для керування станом на стороні frontend використовується бібліотека Zustand. Вона є легкою та продуктивною альтернативою Redux і забезпечує зручний API для створення глобальних сховищ без зайвої складності. Розглянемо основні сховища, що використовуються у додатку.

AuthStore відповідає за аутентифікацію користувачів та управління їхніми сесіями. Він зберігає інформацію про поточного користувача, токен доступу та дозволяє виконувати операції входу, виходу та оновлення сесії. Також це сховище обробляє можливі помилки автентифікації та зберігає статус завантаження під час запитів на сервер.

DistributeStore керує розподілом енергетичних ресурсів між приладами у додатку. Він містить дані про поточний стан розподілу енергії та дозволяє оновлювати девайси для розподілу, їх показники та загальну кількість енергії. Це допомагає візуалізувати та аналізувати розподіл енергії між різними пристроями або користувачами в системі.

DevicesStore і PanelsStore відповідають за управління списками пристроїв та сонячних панелей відповідно. Вони дозволяють додавати, видаляти та оновлювати пристрої та панелі, а також зберігати їхні характеристики. Завдяки цим сховищами додаток підтримує зручну роботу з даними про девайси, сонячні станції та їх налаштуванням.

WeatherStore і PanelTypesStore містять інформацію, яка впливає на енергетичні розрахунки. WeatherStore забезпечує отримання та збереження даних про поточні погодні умови, що важливо для розрахунку ефективності сонячних панелей. PanelTypesStore зберігає інформацію про типи сонячних панелей, їхню потужність та характеристики, що дозволяє правильно розраховувати енергетичний потенціал системи.

LocationAndMapStore відповідає за управління геолокацією користувача та інтеграцію з картою. Він дозволяє отримувати поточне місцезнаходження, оновлювати координати та зберігати інформацію про обране місце. Також сховище забезпечує роботу з маркерами на карті та обробку дозволів на використання геолокації користувача.

Таким чином, використання Zustand дозволяє оптимізувати управління станом у додатку, забезпечуючи зручний та продуктивний спосіб роботи з глобальними даними. Завдяки поділу глобального сховища додатку на окремі модулі з певним функціоналом, код залишається структурованим і легко

підтримуваним. Це спрощує розширення функціональності додатка та дозволяє ефективно працювати з даними в реальному часі.

3.4.5 Реалізація UI-компонентів

Реалізація UI-компонентів у проєкті базується на принципах модульності, інтерактивності та можливості перевикористовувати. Кожен компонент виконує чітко визначену функцію, забезпечуючи зручність взаємодії з користувачем та ефективне управління даними. Компоненти мають добре структуровану логіку та розподілені функції обробки стану, що спрощує їхнє використання та підтримку.

Для управління станом компонентів активно використовуються користувацькі хуки, наприклад, usePanelCard (додаток Е), який інкапсулює логіку редагування, видалення та зміни даних. Такий підхід дозволяє зменшити кількість коду в основному компоненті, зробивши його більш читабельним і зрозумілим. Крім того, використання вбудованих хуків управління станом, таких як useState та useSharedValue, та глобальних, створених нами, сховищ (PanelsStore, DistributeStore та інші) забезпечує ефективний обмін даними між різними частинами застосунку.

Важливу роль у проєкті відіграють анімації, реалізовані через react-native-reanimated. Використання FadeIn, FadeOut та LinearTransition дозволяє створювати плавні ефекти появи та зникнення елементів, а також забезпечує природні переходи між станами інтерфейсу. Наприклад, у компоненті DevicesList (додаток Ж) анімація додає ефект поступового появлення пристроїв на екрані. Ми використовуємо Animated.View, що застосовує FadeIn, що створює ефект плавного масштабування та зсуву, роблячи появу пристроїв більш природною. Аналогічно, при видаленні елементів використовується FadeOut, що забезпечує гармонійне зникнення карток.

Графічні елементи представлені не лише класичними контейнерами та кнопками, а й складнішими інтерактивними компонентами. Для візуалізації

даних використовується LineChart з бібліотеки react-native-gifted-charts, який дозволяє створювати анімовані графіки з динамічним оновленням даних. Також було інтегровано у наш застосунок як анімовані так і звичайні іконки, що надають візуальний акцент на важливих елементах інтерфейсу та покращують його сприйняття.

Окрему увагу приділено діалоговим вікнам та сповіщенням. Для підтвердження дій і попереджень використовується AwesomeAlert з бібліотеки react-native-awesome-alerts, що надає вже стилізований компонент для повідомлень з кнопками підтвердження. Додатково, для відображення сповіщень про успішні чи помилкові дії інтегровано та налаштовано компонент Toast з react-native-toast-message, що дозволяє швидко інформувати користувача про результати його дій.

Візуальна складова компонентів реалізована з використанням StyleSheet та розробленої теми з визначеною кольоровою гамою (MyLightTheme), що сприяє узгодженому вигляду інтерфейсу. Проте також ми використовуємо готові рішення компонентів за рахунок бібліотеки react-native-paper, що надає вже розроблені елементи матеріального дизайну, такі як кнопки, текстові поля та картки. Це дозволяє трохи зменшити кількість самостійно-розроблюваних стилів та таким чином оптимізувати процес розробки frontend-частини застосунку.

Загалом, UI-компоненти реалізовані з урахуванням сучасних підходів до розробки мобільних інтерфейсів. Завдяки чіткій структурі, розподілу логіки між окремими модулями та використанню сучасних підходів до стилізації та анімації, забезпечується зручність користування та ефективне управління даними.

3.5 Тестування роботи додатку

Роботу застосунку для аналізу потенційного добового виробництва сонячної енергії та оцінки можливостей її використання у побуті було здійснено з допомогою мобільного додатку Expo Go. Для всебічної перевірки функціональності та швидкодії додатку використовувались як емулятор Android Studio, так і реальний пристрій. В емуляторі було створено віртуальний девайс з операційною системою Android 15 та роздільною здатністю екрану 1080 на 2280 пікселів, що дозволило відтворити умови використання на сучасних смартфонах середнього та високого класу.

Для перевірки роботи додатку на реальному пристрої було використано смартфон Xiaomi Redmi Note 13, на якому встановлено додаток Expo Go. Такий підхід дозволив оцінити поведінку застосунку в реальних умовах використання та виявити потенційні проблеми, які могли бути непомітні при тестуванні на емуляторі. Також при тестуванні було приділено увагу швидкості обробки запитів, відображенню інтерфейсу та коректності проведених розрахунків на реальному пристрої.

У процесі тестування перевірялась коректність роботи усіх основних функцій додатку. В першу чергу, було протестовано функцію отримання геолокаційних даних, яка є ключовою для розрахунку потенціалу сонячної енергії. Тестування показало, що додаток коректно отримує дані про місцезнаходження користувача як при використанні GPS (на реальному пристрої), так і при заданні координат за допомогою карти. Перевірено також коректність валідації введених даних та відображення повідомлень про помилки при некоректному введенні.

Наступним етапом тестування стала перевірка алгоритмів розрахунку потенційного добового виробництва сонячної енергії. Для цього обрали 5 днів (2-6 квітня 2025 року) впродовж яких будемо тестувати застосунок, задали місце розташування панелей та їх параметри (див. рис. 3.5). Кожного дня запускали застосунок та розраховували потенційну вироблену енергію для

заданої сонячної системи (додаток И), у результаті загальна сума такої енергії за визначений період склала 56,1 кВт.

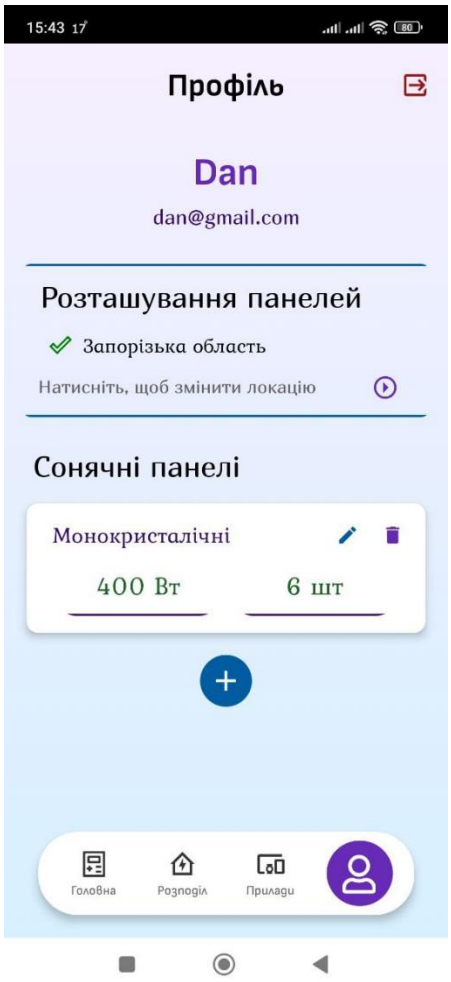


Рисунок 3.5 – Знімок екрану з відображенням налаштованих даних у застосунку

Для оцінки точності результатів, використали дані (з 2 по 6 квітня), отримані за допомогою перевіреного сервісу PVWatts Calculator, який надав нам інформацію про погодинний виробіток енергії сонячними панелями, з відповідними характеристиками, у вигляді таблиці (додаток К). Переглянувши результати, вирахували, що сума потенційної виробленої енергії впродовж 5 днів становить 57,23 кВт. Результати розрахунків нашого застосунку порівнювали з еталонними значеннями (від сервісу PVWatts) і отриманими, що відхилення результатів дорівнює приблизно 2%, що є оптимальним для даного типу застосунку. Проте даний показник не можна вважати цілком точним, так

як тестовий період був надто коротким для повноцінної оцінки точності алгоритму. Для більш об'єктивної оцінки необхідно проводити тривале тестування протягом всього року, щоб врахувати сезонні зміни клімату, кута падіння сонячних променів та інших факторів, що впливають на продуктивність сонячних панелей.

Важливим аспектом тестування була перевірка візуалізації отриманих результатів. Тестування показало, що графіки добового виробництва сонячної енергії відображаються коректно як на емуляторі, так і на реальному пристрої. Інтерфейс дозволяє користувачу легко аналізувати дані, а кольорова схема забезпечує зрозуміле представлення інформації. В ході тестування було виявлено незначні проблеми з відображенням маркера на карті, при використанні віртуального девайсу, які були оперативно усунені.

Завершальним етапом тестування стала перевірка зручності використання додатку. Для цього було залучено кілька осіб, яким дали можливість користуватися застосунком без попереднього знайомства з інструкцією. Результати показали, що інтерфейс додатку є інтуїтивно зрозумілим, а основні функції легко доступні. Більшість користувачів змогли успішно виконати всі запропоновані завдання без сторонньої допомоги, що свідчить про високу зручність використання розробленого додатку.

За результатами проведеного тестування можна зробити висновок, що розроблений додаток для аналізу потенційного добового виробництва сонячної енергії повністю відповідає поставленим вимогам. Додаток демонструє стабільну роботу як на емуляторі Android Studio, так і на реальному пристрої Xiaomi Redmi Note 13. Функціонал застосунку працює коректно, а інтерфейс забезпечує зручне та ефективне використання. Виявлені в процесі тестування недоліки були успішно усунені, що дозволило підвищити загальну якість продукту.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було розроблено мобільний застосунок на базі технології React Native для аналізу потенційного добового виробництва сонячної енергії та оцінки можливостей її використання у побуті. Створений програмний продукт дозволяє користувачам розраховувати кількість енергії, яку можуть виробити сонячні панелі, та ефективно планувати її розподіл між побутовими приладами відповідно до встановлених пріоритетів.

Аналіз існуючих рішень у сфері розрахунку сонячної енергії виявив, що наявні на ринку застосунки не забезпечують комплексного підходу до проблеми оптимального використання сонячної енергії в побуті, або мають обмежений функціонал для персоналізованого налаштування. Розроблений застосунок вирішує цю проблему, надаючи користувачам повний спектр інструментів для аналізу та планування енергоспоживання.

У процесі розробки було спроєктовано ефективну архітектуру застосунку, що складається з клієнтської та серверної частин. Клієнтська частина реалізована за допомогою технології React Native, що забезпечує кросплатформеність і дозволяє використовувати додаток як на пристроях Android, так і на iOS. Серверна частина розроблена на Node.js, що забезпечує ефективну обробку запитів та виконання складних обчислень. Для зберігання даних про користувачів, їх сонячні панелі та прилади використовується нереляційна база даних MongoDB.

Успішно реалізовано інтеграцію з зовнішніми сервісами для отримання актуальних даних про погодні умови, що дозволяє точно прогнозувати виробництво сонячної енергії. Розроблені алгоритми враховують такі фактори, як географічне розташування, потужність сонячних панелей, рівень інсоляції та погодні умови, що забезпечує високу точність прогнозів.

Особливу увагу було приділено розробці алгоритму розподілу виробленої енергії між побутовими приладами. Система дозволяє користувачам

встановлювати пріоритети для різних приладів та автоматично розподіляє доступну енергію відповідно до цих налаштувань. Такий підхід допомагає оптимізувати використання енергії та зменшити залежність від традиційних джерел енергопостачання.

Інтерфейс застосунку розроблено з урахуванням принципів інтуїтивної зрозумілості та зручності використання. Система візуалізації результатів у вигляді графіків та діаграм робить дані доступними для користувачів з різним рівнем технічної підготовки.

Розроблений мобільний застосунок має значну практичну цінність і може використовуватись як самостійний інструмент для фізичних осіб, домогосподарств або бізнесів, що планують встановлення сонячних панелей. Крім того, він може бути інтегрований як компонент у більш широкі екосистеми енергетичних компаній. Застосунок сприяє раціональному використанню відновлюваних джерел енергії, що є важливим кроком на шляху до екологічно сталого розвитку.

ПЕРЕЛІК ПОСИЛАНЬ

1. Енергетична стратегія України на період до 2035 року. Київ, 2015. 48 с. URL : https://niss.gov.ua/sites/default/files/2015-04/Energy%20Strategy.pdf (дата звернення : 10.03.2025).

2. 1. Відновлювані джерела енергії / за заг. ред. С. О. Кудрі. Київ : Інститут відновлюваної енергетики НАНУ, 2020. 392 с. URL : https://www.ive.org.ua/wp-content/uploads/Monografia_final_21.12.2020.pdf (дата звернення : 10.03.2025).

3. Матушкін Д. С. Огляд сучасних методів прогнозування сонячної енергії. Матеріали аспірантських читань пам’яті професора Артура Праховника присвячені 75-річчю ІЕЕ, м. Київ, 10–11 берез. 2021 р. 2021. С. 34–38. URL : https://ela.kpi.ua/server/api/core/bitstreams/c1cc605e-8c22-4311-8124-d2bb1052b750/content (дата звернення : 12.03.2025).

4. Інсоляція : вплив на виробництво електроенергії сонячними панелями. Solar Garden. URL : https://www.solargarden.com.ua/insolyatsiya-vplyv-na-vyrobnytstvo-elektroenergiyi-sonyachnymy-panelyamy/ (дата звернення : 10.03.2025).

5. Ефективність сонячних панелей. Solar Garden. URL : https://www.solargarden.com.ua/vid-chogo-zalezhyt-kkd-sonyachnyh-panelej/ (дата звернення : 10.03.2025).

6. Steve Ransome. 4ep1.1 How well do PV modelling algorithms really predict performance? *22nd European PVSEC Milan*. 2007. URL : https://www.steveransome.com/PUBS/2007Milan_4EP_1_1_paper.pdf (дата звернення : 10.03.2025).

7. Normalised efficiency of photovoltaic systems: Going beyond the performance ratio. *Bert Herteleer et al*. 2017. URL : https://www.researchgate.net/publication/319422942_Normalised_efficiency_of_photovoltaic_systems_Going_beyond_the_performance_ratio (дата звернення : 10.03.2025).

8. Energy Yield Calculation Of The Grid Connected Photovoltaic Power System. *A. M. Omar et al. International Conference on Renewable Energy Sources (RES 14), Boston, 2 July 2014*. URL : https://www.researchgate.net/publication/276072524_Energy_Yield_Calculation_Of_The_Grid_Connected_Photovoltaic_Power_System (дата звернення : 10.03.2025).

9. Swapnil Dubey, Jatin Narotam Sarvaiya, Bharath Seshadri. Temperature Dependent Photovoltaic (PV) Efficiency and Its Effect on PV Production in the World – A Review. *Energy Procedia*. 2013. Vol. 33. P. 311–321. URL : https://doi.org/10.1016/j.egypro.2013.05.072 (дата звернення : 11.03.2025).

10. How does wet weather affect the efficiency of solar panels. Tongwei. 2024. URL : https://en.tongwei.cn/blog/452.html (дата звернення : 11.03.2025).

11. OpenJS Foundation. About Node.js®. Node.js. URL : https://nodejs.org/en/about/ (дата звернення : 21.03.2025).

12. OpenJS Foundation. Express - Node.js web application framework. Express.js. URL : https://expressjs.com/ (дата звернення : 21.03.2025).

13. MongoDB, Inc. What Is MongoDB. MongoDB. URL : https://www.mongodb.com/what-is-mongodb (дата звернення : 21.03.2025).

14. Meta Platforms, Inc. React Native - A framework for building native apps using React. React Native. URL : https://reactnative.dev/ (дата звернення : 21.03.2025).

15. 650 Industries, Inc. Expo Documentation. Expo. URL : https://docs.expo.dev/ (дата звернення : 21.03.2025).

16. OpenWeather. Weather API. OpenWeatherMap. URL : https://openweathermap.org/api (дата звернення: 31.03.2025).

ДОДАТОК А

Програмний код алгоритму обрахунку виробленої сонячної енергії

```
export default function useSolarEnergyCalculator() {
  const { location } = useLocationStore();
  const { panels } = usePanelsStore();
  const { panelTypes } = usePanelTypesStore();
  const [allEnergy, setAllEnergy] = useState(0);

  const efficiency = useMemo(() => {
    if (!panels?.length || !panelTypes?.length) return null;

    const effMono = panelTypes.find((pT) => pT.type === "Монокристалічні").efficiency / 100;
    const effPoli = panelTypes.find((pT) => pT.type === "Полікристалічні").efficiency / 100;
    const effAmo = panelTypes.find((pT) => pT.type === "Аморфні").efficiency / 100;

    return { monocristal: effMono, policristal: effPoli, amorfni: effAmo, };
  }, [panels, panelTypes]);

  const panelsPower = useMemo(() => {
    if (!panels?.length || !efficiency) return null;

    return panels.reduce(
      (acc, panel) => {
        const totalPower = (+panel.number * +panel.power) / 1000;
        if (panel.type === "Монокристалічні") {
          acc.monocristal += totalPower * efficiency.monocristal;
        } else if (panel.type === "Полікристалічні") {
          acc.policristal += totalPower * efficiency.policristal;
        } else if (panel.type === "Аморфні") {
          acc.amorfni += totalPower * efficiency.amorfni;
        }
        return acc;
      },
      { monocristal: 0, policristal: 0, amorfni: 0 }
    );
  }, [panels, efficiency]);

  const insolation = useMemo(
    () => location?.monthlyInsolation?.[new Date().getMonth()] || 0,
    [location]
  );

  const calculateHourlyEnergy = useCallback(
    (hourlyClouds, hourlyTemp, sunrise, sunset) => {
      const hourlyEnergy = [];
      const [sunriseHour, sunriseMinute] = sunrise.split(":").map(Number);
      const [sunsetHour, sunsetMinute] = sunset.split(":").map(Number);

      let totalEnergyAccumulator = 0;

      for (let hour = sunriseHour; hour <= sunsetHour; hour++) {
        let startMinute = hour === sunriseHour ? sunriseMinute : 0;
        let endMinute = hour === sunsetHour ? sunsetMinute : 60;
```

```
    const cloudiness = hourlyClouds[hour];
    const cloudImpact = 1 - CLOUD_COEF * (cloudiness / 100);
    const tempImpact =
      1 - (TEMP_COEF * (hourlyTemp[hour] - TEMP_STANDART)) / 100;

    const periodResult =
      insolation * cloudImpact * tempImpact *
      ((endMinute - startMinute) / 60);
    const energyMono = panelsPower.monocristal * periodResult;
    const energyPoly = panelsPower.policristal * periodResult;
    const energyAmorph = panelsPower.amorfni * periodResult;

    const totalEnergy = energyMono + energyPoly + energyAmorph;
    totalEnergyAccumulator += totalEnergy;

    hourlyEnergy.push({
      interval: `${hour}:${startMinute.toString().padStart(2, "0")}-${formatTime(hour, endMinute)}`,
      cloudiness,
      temp: hourlyTemp[hour],
      energy: totalEnergy.toFixed(3),
    });
  }

  const startEnergy = {
    interval: `${sunrise}`,
    cloudiness: hourlyClouds[sunriseHour] || 0,
    energy: 0,
    temp: hourlyTemp[sunriseHour] || 0,
  };

  const energyForGraph = [startEnergy, ...hourlyEnergy];

  setAllEnergy(totalEnergyAccumulator.toFixed(2));

  return {
    hourlyEnergy: energyForGraph, allEnergy: totalEnergyAccumulator.toFixed(2),
  };
 },
 [insolation, panelsPower]
);

return { calculateHourlyEnergy, allEnergy };
        }
```

ДОДАТОК Б

Програмний код алгоритму розподілу енергії

```
export const calculateDistribution = ( selectedDevices, totalDistributeEnergy) => {
 if (!selectedDevices?.length || !totalDistributeEnergy) return [];

 const totalEnergy = totalDistributeEnergy * 1000;

 // Фільтруємо пристрої з фіксованим часом
 const fixedDevices = selectedDevices.filter((dev) => dev.fixedTime);
 let fixedEnergyTotal = 0;

 fixedDevices.forEach((dev) => {
  dev.deviceEnergy = parseFloat(
   (dev.power * (dev.workingHours * 60 + dev.workingMinutes)) / 60 / 1000
  ).toFixed(3);
  fixedEnergyTotal += dev.deviceEnergy * 1000;
 });

 const remainingEnergy = totalEnergy - fixedEnergyTotal;

 // Фільтруємо пристрої без фіксованого часу
 const dynamicDevices = selectedDevices.filter((dev) => !dev.fixedTime);

 // Якщо енергії на інші пристрої не лишилось
 if (remainingEnergy <= 0) {
  const zeroTimeDevices = dynamicDevices.map((device) => ({
   ...device, workingHours: 0, workingMinutes: 0, deviceEnergy: 0, allocatedEnergy: 0,
  }));
  return [...fixedDevices, ...zeroTimeDevices];
 }

 // Перераховуємо maxWorkingHours та maxWorkingMinutes для кожного пристрою
 dynamicDevices.forEach((device) => {
  const maxWorkingTime = remainingEnergy / device.power;
  device.maxWorkingHours = Math.floor(maxWorkingTime);
  device.maxWorkingMinutes = Math.round(
   (maxWorkingTime - device.maxWorkingHours) * 60
  );
 });

 // Групуємо інші пристрої за пріоритетністю
 const groupedDevices = {
  high: dynamicDevices.filter((dev) => dev.importance === "висока"),
  medium: dynamicDevices.filter((dev) => dev.importance === "середня"),
  low: dynamicDevices.filter((dev) => dev.importance === "низька"),
 };

 let distributionRatios = { high: 0.5, medium: 0.35, low: 0.15 };
 const activeGroups = Object.keys(groupedDevices).filter(
  (key) => groupedDevices[key].length > 0
 );
 const totalActiveRatio = activeGroups.reduce(
  (sum, key) => sum + distributionRatios[key], 0 );
```

```
 activeGroups.forEach((key) => {
  distributionRatios[key] = distributionRatios[key] / totalActiveRatio;
 });

 // Розподіляємо енергію серед пристроїв, які не мають фіксованого часу
 const energyDistribution = {
  high:
   (remainingEnergy * distributionRatios.high) /
   (groupedDevices.high.length || 1),
  medium:
   (remainingEnergy * distributionRatios.medium) /
   (groupedDevices.medium.length || 1),
  low:
   (remainingEnergy * distributionRatios.low) /
   (groupedDevices.low.length || 1),
 };

 const updatedDevices = dynamicDevices.map((device) => {
  const importanceKey =
   device.importance === "висока"
    ? "high"
    : device.importance === "середня"
     ? "medium"
     : "low";
  const allocatedEnergy = energyDistribution[importanceKey];
  const { hours, minutes } = calculateWorkingTime(device, allocatedEnergy);
  const deviceUseEnergy = parseFloat(
   (device.power * (hours * 60 + minutes)) / 60 / 1000
  ).toFixed(3);

  return { ...device, allocatedEnergy, deviceEnergy: deviceUseEnergy, workingHours: hours, workingMinutes:
minutes, };
 });

 return [...fixedDevices, ...updatedDevices];
};

const calculateWorkingTime = (device, allocatedEnergy) => {
 if (!device.power || device.power <= 0) return { hours: 0, minutes: 0 };

 const workingTime = allocatedEnergy / device.power;
 const hours = Math.floor(workingTime);
 const minutes = Math.round((workingTime - hours) * 60);
 return { hours, minutes };
};
```

ДОДАТОК В

Основний файл backend-частини app.js

```
const express = require("express");
const morgan = require("morgan");
const app = express();
require("dotenv").config();
const mongoose = require("mongoose");
const cors = require("cors");
const authJwt = require("./helpers/jwt");
const { errorHandler } = require("./helpers/error-handler");
app.use(cors());
app.options("*", cors());

const port = process.env.PORT || 4000;
const api = process.env.API_URL;
const mongoURI = process.env.MONGO_URI;

const appliancesRouter = require("./routers/appliances");
const usersRouter = require("./routers/users");
const panelsRouter = require("./routers/panels");
const locationsRouter = require("./routers/locations");
const regionsRouter = require("./routers/regions");
const panelTypesRouter = require("./routers/panelTypes");

app.use(express.json());
app.use(morgan("tiny"));
app.use(authJwt());

app.use(`${api}/appliances`, appliancesRouter);
app.use(`${api}/users`, usersRouter);
app.use(`${api}/panels`, panelsRouter);
app.use(`${api}/locations`, locationsRouter);
app.use(`${api}/regions`, regionsRouter);
app.use(`${api}/paneltypes`, panelTypesRouter);

app.use(errorHandler);

mongoose
  .connect(mongoURI, {
    dbName: "solar-panels",
  }).then(() => {
    console.log("Database connected");
  }).catch((err) => {
    console.error("Error connecting to the database:", err);
  });

app.listen(port, () => {
  console.log(`Example app listening on port ${port}`);
});
```

ДОДАТОК Г

Файл-маршрутизатор на серверній частині для приладів

```
const express = require("express");
const router = express.Router();
const { Appliance } = require("../models/appliance");
const { AppError, ERROR_TYPES } = require("../helpers/error-handler");
const { User } = require("../models/user");

const importanceValues = ["низька", "середня", "висока"];

router.get("/", async (req, res, next) => {
  try {
    const appliances = await Appliance.find();
    if (!appliances.length) {
      throw new AppError(ERROR_TYPES.NOT_FOUND, "Прилади не знайдено");
    }
    res.status(200).json(appliances);
  } catch (error) {
    next(error);
  }
});

router.get("/:id", async (req, res, next) => {
  try {
    const appliance = await Appliance.findById(req.params.id);
    if (!appliance) {
      throw new AppError(ERROR_TYPES.NOT_FOUND, "Прилад не знайдено");
    }
    res.status(200).json(appliance);
  } catch (error) {
    next(error);
  }
});

router.get("/userId/:userId", async (req, res, next) => {
  try {
    const { userId } = req.params;

    const user = await User.findById(userId);
    if (!user) {
      throw new AppError(
        ERROR_TYPES.NOT_FOUND,
        "Користувача з таким id не знайдено"
      );
    }

    const appliances = await Appliance.find({ userId });

    res.status(200).json(appliances);
  } catch (error) {
    next(error);
  }
});
```

```
router.post("/", async (req, res, next) => {
 try {
  const { name, power, importance, userId } = req.body;
  if (!name || !power || !importance || !userId) {
   throw new AppError(
    ERROR_TYPES.VALIDATION_ERROR,
    "Відсутні обов'язкові поля"
   );
  }

  if (!importanceValues.includes(importance)) {
   throw new AppError(
    ERROR_TYPES.VALIDATION_ERROR,
    "Невірне значення пріоритету приладу"
   );
  }

  const user = await User.findById(userId);
  if (!user) {
   throw new AppError(
    ERROR_TYPES.NOT_FOUND,
    "Користувача з таким id не знайдено"
   );
  }

  const newAppliance = new Appliance({
   name,
   power,
   importance,
   userId,
  });

  const savedAppliance = await newAppliance.save();
  res.status(201).json(savedAppliance);
 } catch (error) {
  next(error);
 }
});

router.put("/:id", async (req, res, next) => {
 try {
  const { name, power, importance } = req.body;
  if (importance && !importanceValues.includes(importance)) {
   throw new AppError(
    ERROR_TYPES.VALIDATION_ERROR,
    "Невірне значення пріоритету приладу"
   );
  }

  const updatedAppliance = await Appliance.findByIdAndUpdate(
   req.params.id,
   { name, power, importance },
   { new: true, runValidators: true }
  );

  if (!updatedAppliance) {
   throw new AppError(ERROR_TYPES.NOT_FOUND, "Прилад не знайдено");
  }

  res.status(200).json(updatedAppliance);
 } catch (error) {
```

```
    next(error);
  }
});

router.delete(`/:id`, async (req, res, next) => {
  const applianceId = req.params.id;
  try {
    const appliance = await Appliance.findByIdAndDelete(applianceId);

    if (!appliance) {
      throw new AppError(ERROR_TYPES.NOT_FOUND, "Прилад не знайдено");
    }

    return res.status(200).json({
      success: true,
      message: "The appliance was removed",
    });
  } catch (error) {
    next(error);
  }
});

module.exports = router;
```

ДОДАТОК Д

Програмна реалізація компонента Map

```
import { useRouter } from "expo-router";
import MapView, { Marker, PROVIDER_GOOGLE } from "react-native-maps";
import { SafeAreaView, StyleSheet, Text, View } from "react-native";
import { ActivityIndicator, FAB } from "react-native-paper";
import { useState, useEffect, useCallback } from "react";
import ErrorText from "./UI/ErrorText";
import { MyLightTheme } from "../assets/theme/global";
import LoadingScreen from "./UI/LoadingScreen";
import { showToast } from "../utils/showToast";
import CustomAlert from "./UI/CustomAlert";
import { getRegionName } from "../store/utils/locationUtils";
import useLocationStore from "../store/locationAndMapStore";

const INITIAL_COORDS = {
 latitude: 50.4501,
 longitude: 30.5234,
 latitudeDelta: 20,
 longitudeDelta: 20,
};

export default function Map({ forRegister }) {
 const router = useRouter();

 const {
  location,
  markerCoords,
  isLoading,
  error,
  clearError,
  setCoordinatesAndFetchAddress,
  setRegisterLocationWithMap,
  setMapMarkerCoordinates,
 } = useLocationStore();

 const [currentCoords, setCurrentCoords] = useState(
  location
   ? {
     latitude: location.latitude,
     longitude: location.longitude,
     latitudeDelta: 20,
     longitudeDelta: 20,
    }
   : markerCoords
 );
 const [isSavingLocation, setIsSavingLocation] = useState(false);
 const [refreshing, setRefreshing] = useState(false);

 const [locationAlert, setLocationAlert] = useState({
  show: false,
  message: "",
  onConfirm: null,
```

```
  onCancel: null,
});
const [showErrorAlert, setShowErrorAlert] = useState(false);
const closeErrorAlert = () => setShowErrorAlert(false);

const onRefresh = useCallback(() => {
  setRefreshing(true);
  clearError();

  setTimeout(async () => {
    try {
      setCurrentCoords(INITIAL_COORDS);
      setMapMarkerCoordinates(INITIAL_COORDS);
    } catch (error) {
      showToast("error", "Не вдалося оновити дані карти.");
    } finally {
      setRefreshing(false);
    }
  }, 1000);
}, [clearError, setMapMarkerCoordinates]);

useEffect(() => {
  clearError();
  if (!currentCoords && location) {
    setCurrentCoords({
      latitude: location.latitude,
      longitude: location.longitude,
      latitudeDelta: 20,
      longitudeDelta: 20,
    });
  } else if (!currentCoords && !location) {
    setCurrentCoords(markerCoords);
  }
}, []);

const onRegionChangeComplete = (region) => {
  setCurrentCoords(region);
};

const saveLocation = useCallback(async () => {
  try {
    setIsSavingLocation(true);
    setShowErrorAlert(false);

    const region = await getRegionName(
      currentCoords.latitude,
      currentCoords.longitude
    );

    if (region) {
      setMapMarkerCoordinates(currentCoords);

      if (forRegister) {
        await setRegisterLocationWithMap({
          ...currentCoords,
          regionName: region,
        });
      } else {
        await setCoordinatesAndFetchAddress({
          ...currentCoords,
          regionName: region,
        });
```

```
      }

      setLocationAlert({
        show: true,
        message: `Широта: ${currentCoords?.latitude}, \nДовгота: ${currentCoords?.longitude}, \nВаша область: ${region}`,
        onConfirm: () => {
          setLocationAlert((prev) => ({
            ...prev,
            show: false,
          }));
          router.back();
        },
        onCancel: () => {
          setLocationAlert(() => ({ ...locationAlert, show: false }));
        },
      });
    } else {
      setShowErrorAlert(true);
      return;
    }
  } catch (err) {
    showToast("error", "Не вдалося зберегти місцезнаходження.");
  } finally {
    setIsSavingLocation(false);
  }
}, [
  currentCoords,
  router,
  locationAlert,
  forRegister,
  setCoordinatesAndFetchAddress,
  setMapMarkerCoordinates,
  setRegisterLocationWithMap,
]);

if (!currentCoords || !markerCoords) {
  return <LoadingScreen />;
}

return (
  <SafeAreaView style={styles.container}>
    <>
      {!error && (
        <MapView
          style={styles.map}
          onRegionChangeComplete={onRegionChangeComplete}
          initialRegion={currentCoords}
          region={currentCoords}
          provider={PROVIDER_GOOGLE}
        >
          <Marker
            coordinate={{
              latitude: currentCoords.latitude,
              longitude: currentCoords.longitude,
            }}
            title="Your Location"
            style={styles.marker}
            draggable
            onDragEnd={(e) => {
              const newCoords = {
                latitude: e.nativeEvent.coordinate.latitude,
```

```
            longitude: e.nativeEvent.coordinate.longitude,
          };
          if (
            newCoords.latitude !== currentCoords.latitude ||
            newCoords.longitude !== currentCoords.longitude
          ) {
            setCurrentCoords({ ...currentCoords, ...newCoords });
          }
        }}
      />
    </MapView>
  )}

  {error && (
    <View style={styles.loadingOverlay}>
      <ErrorText error={error} />
    </View>
  )}

  {(isLoading || isSavingLocation || refreshing) && (
    <View style={styles.loadingOverlay}>
      <ActivityIndicator size="large" color={MyLightTheme.colors.white} />
      <Text style={styles.loadingText}>Завантаження...</Text>
    </View>
  )}

  <FAB
    onPress={() => {
      router.back();
    }}
    icon={"arrow-left"}
    style={styles.btnBack}
    customSize={46}
  />
  <FAB
    onPress={saveLocation}
    style={styles.btnSave}
    label="Зберегти"
    icon={"check"}
    customSize={48}
  />
  <FAB
    onPress={onRefresh}
    icon={"reload"}
    style={styles.reloadBtn}
    customSize={46}
  />
  {showErrorAlert && (
    <CustomAlert
      showAlert={showErrorAlert}
      title={"Помилка"}
      message={
        "Не вдалося визначити область для заданих координат. Будь ласка, перевірте, чи відповідає розташування прапорця на карті одній з області України."
      }
      onConfirm={closeErrorAlert}
    />
  )}
  {locationAlert.show && (
    <CustomAlert
      showAlert={locationAlert.show}
      title={"Ваше місцезнаходження"}
```

```
          message={locationAlert.message}
          showCancelButton
          onConfirm={() => {
           if (locationAlert.onConfirm) {
            locationAlert.onConfirm();
           }
          }}
          onCancel={() => {
           if (locationAlert.onCancel) {
            locationAlert.onCancel();
           }
          }}
         />
       )}
      </>
    </SafeAreaView>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
  },
  map: {
    flex: 1,
  },
  btnSave: {
    right: 0,
    position: "absolute",
    top: 0,
    margin: 16,
    backgroundColor: MyLightTheme.colors.white,
    borderWidth: 3,
    borderColor: MyLightTheme.colors.greenDark,
  },
  btnBack: {
    position: "absolute",
    top: 0,
    margin: 16,
    marginTop: 19,
    backgroundColor: MyLightTheme.colors.white,
  },
  reloadBtn: {
    right: 0,
    position: "absolute",
    bottom: 0,
    margin: 16,
    backgroundColor: MyLightTheme.colors.white,
    borderWidth: 3,
    borderColor: MyLightTheme.colors.primaryDark,
  },
  marker: {
    width: 50,
    height: 50,
  },
  loadingOverlay: {
    position: "absolute",
    top: 0,
    left: 0,
    bottom: 0,
    backgroundColor: MyLightTheme.colors.darkTransparent,
    justifyContent: "center",
```

```
    alignItems: "center",
  },
  loadingText: {
    color: MyLightTheme.colors.white,
    fontSize: 16,
    marginTop: 10,
    fontWeight: "bold",
  },
});
```

ДОДАТОК Е

Програмний код користувацького хука usePanelCard

```
import { useState } from "react";
import { showToast } from "../../utils/showToast";
import usePanelsStore from "../../store/panelsStore";

export const usePanelCard = ({ id, power, number, typeId, refresh }) => {
  const [panelData, setPanelData] = useState({
    power: power || null,
    number: number || null,
    typeId: typeId || null,
  });
  const [reductOpen, setReductOpen] = useState(false);
  const [deleteOpen, setDeleteOpen] = useState(false);
  const [showAlert, setShowAlert] = useState(false);

  const { changePanel, deletePanel } = usePanelsStore();

  const handleReductPanel = () => {
    setReductOpen((prev) => !prev);
  };

  const handleDeleteDialog = () => {
    setDeleteOpen((prev) => !prev);
  };

  const handleConfirm = () => setShowAlert(false);

  const handleSaveChanges = async () => {
    try {
      const power = Number(panelData.power);
      const number = Number(panelData.number);

      if (
        isNaN(power) ||
        power <= 0 ||
        isNaN(number) ||
        number <= 0 ||
        !typeId
      ) {
        setShowAlert(true);
        return;
      }

      await changePanel({ id, ...panelData });
      showToast("success", "Дані успішно оновлені");
      setReductOpen(false);
      await refresh();
    } catch (error) {
      showToast("error", "Помилка при оновленні даних");
    }
  };
```

```
  const handleDelete = async () => {
    try {
      await deletePanel(id);
      showToast("success", "Панель успішно видалена");
      setDeleteOpen(false);
      await refresh();
    } catch (e) {
      showToast("error", "Помилка при видалення панелі");
    }
  };

  const closeReductDialog = () => setReductOpen(false);
  const closeDeleteDialog = () => setDeleteOpen(false);

  return {
    panelData,
    setPanelData,
    reductOpen,
    deleteOpen,
    showAlert,
    handleReductPanel,
    handleDeleteDialog,
    handleConfirm,
    handleSaveChanges,
    handleDelete,
    closeReductDialog,
    closeDeleteDialog,
  };
};
```

ДОДАТОК Ж

Приклад використання анімації у компоненті DevicesList

```
import React from "react";
import { View, StyleSheet } from "react-native";
import DeviceCard from "./DeviceCard";
import { usePathname } from "expo-router";
import DeviceDistributionCard from "../distribution/DeviceCard";
import Animated, {
  FadeIn,
  FadeOut,
  LinearTransition,
} from "react-native-reanimated";

export default function DevicesList({
  devices,
  refresh,
  toggleSelect,
  selected,
}) {
  const currentPath = usePathname();

  if (currentPath === "/devices") {
    return (
      <View style={styles.devicesContainer}>
        {devices?.map((device) => {
          return <DeviceCard data={device} key={device.id} refresh={refresh} />;
        })}
      </View>
    );
  }

  if (currentPath === "/energy_distribution/list") {
    return (
      <Animated.View
        style={styles.disrtibutionContainer}
        entering={FadeIn.duration(600).springify()}
        layout={LinearTransition.springify().mass(0.8)}
      >
        {devices?.map((device) => {
          return (
            <Animated.View
              key={device.id}
              entering={FadeIn.delay(100)
                .duration(600)
                .withInitialValues({
                  opacity: 0,
                  transform: [{ scale: 0.8 }, { translateX: 50 }],
                })
                .springify()}
              exiting={FadeOut.duration(500)
                .withInitialValues({
                  opacity: 1,
                  transform: [{ scale: 1 }, { translateX: 0 }],
```

```
            })
            .springify()}
           layout={LinearTransition.springify().mass(0.8)}
          >
           <DeviceDistributionCard
            data={device}
            toggleSelect={toggleSelect}
            selected={selected}
           />
          </Animated.View>
         );
        })}
      </Animated.View>
    );
  }
}

const styles = StyleSheet.create({
  devicesContainer: {
    flexWrap: "wrap",
    flexDirection: "row",
    width: "100%",
    justifyContent: "center",
    paddingTop: 20,
  },
  disrtibutionContainer: {
    flexDirection: "column",
    width: "100%",
    paddingHorizontal: 8,
    justifyContent: "center",
  },
});
```

ДОДАТОК И

Знімки екранів програми з розрахунками енергії з 2 по 6 квітня



Рисунок И.1 – Розрахунок потенційної виробленої енергії на 2 квітня 2025 року, визначеної розробленим додатком



Рисунок И.2 – Розрахунок потенційної виробленої енергії на 3 квітня 2025 року, визначеної розробленим додатком



Рисунок И.3 – Розрахунок потенційної виробленої енергії на 4 квітня 2025 року, визначеної розробленим додатком



Рисунок И.4 – Розрахунок потенційної виробленої енергії на 5 квітня 2025 року, визначеної розробленим додатком



Рисунок И.5 – Розрахунок потенційної виробленої енергії на 6 квітня 2025 року, визначеної розробленим додатком

ДОДАТОК К

Знімки розрахунків енергії від сервісу PVWatts Calculator з 2 по 6 квітня

Місяць	День	Година	Температура навколишнього середовища (C)	Швидкість вітру (m/s)	Альбедо	Опромінення на поверхні системи (W/m2)	Температура сонячної батареї (C)	Виробництво постійного струму (W)	Добове виробництво струму (W)
4	2	0	-2,5	2,7	0,2	0	-2,5	0	
4	2	1	-2,9	2,7	0,2	0	-2,9	0	
4	2	2	-3,3	2,7	0,2	0	-3,3	0	
4	2	3	-3,5	2,9	0,2	0	-3,5	0	
4	2	4	-3,4	3	0,2	0	-3,4	0	
4	2	5	-2,8	3,4	0,2	0	0	0	
4	2	6	-0,8	4,1	0,2	9,272	-0,642	20,091	
4	2	7	0,9	4,5	0,2	30,665	1,396	69,828	
4	2	8	1,8	4,3	0,2	182,213	4,823	432,407	
4	2	9	2,5	4	0,2	436,663	10,049	1032,793	
4	2	10	3,3	3,6	0,2	329,475	9,329	770,28	
4	2	11	4,3	3,2	0,2	471,829	13,5	1094,951	
4	2	12	5,2	2,9	0,2	408,203	13,565	943,866	
4	2	13	5,8	2,8	0,2	374,301	13,612	867,236	
4	2	14	6	2,9	0,2	452,822	15,313	1055,728	
4	2	15	5,8	2,9	0,2	191,006	9,734	448,18	
4	2	16	4,9	2,6	0,2	201,238	9,259	469,767	
4	2	17	3,1	1,6	0,2	3,705	3,196	7,069	
4	2	18	1,3	1,3	0,2	0	1,3	0	
4	2	19	0,6	1,3	0,2	0	0,6	0	
4	2	20	0	1,3	0,2	0	0	0	
4	2	21	-0,4	1,3	0,2	0	-0,4	0	
4	2	22	-0,5	1,2	0,2	0	-0,5	0	
4	2	23	-0,6	1,1	0,2	0	-0,6	0	7212,196

Рисунок К.1 – Дані розрахунку потенційної виробленої енергії на 2 квітня, отримані за допомогою сервісу PVWatts Calculator

Місяць	День	Година	Температура навколишнього середовища (C)	Швидкість вітру (m/s)	Альбедо	Опромінення на поверхні системи (W/m2)	Температура сонячної батареї (C)	Виробництво постійного струму (W)	Добове виробництво струму (W)
4	3	0	-0,6	0,9	0,2	0	-0,6	0	
4	3	1	-0,6	0,7	0,2	0	-0,6	0	
4	3	2	-0,5	0,5	0,2	0	-0,5	0	
4	3	3	-0,4	0,4	0,2	0	-0,4	0	
4	3	4	-0,3	0,4	0,2	0	-0,3	0	
4	3	5	0	0,5	0,2	0	0	0	
4	3	6	1,6	0,7	0,2	159,913	6,673	328,008	
4	3	7	3,9	0,7	0,2	395,479	16,984	923,916	
4	3	8	5,6	0,8	0,2	623,116	25,723	1431,857	
4	3	9	6,7	1,2	0,2	814,613	30,419	1823,989	
4	3	10	7,8	1,6	0,2	940,19	32,733	2073,285	
4	3	11	8,7	2,2	0,2	991,575	31,913	2186,634	
4	3	12	9,4	2,7	0,2	982,171	30,358	2180,549	
4	3	13	9,8	3,1	0,2	891,275	27,577	2010,097	
4	3	14	9,8	3,5	0,2	719,51	23,279	1660,603	
4	3	15	9,4	3,7	0,2	516,764	18,81	1221,236	
4	3	16	8,2	3,4	0,2	287,761	13,591	642,71	
4	3	17	5,4	2,4	0,2	85,49	7,165	149,575	
4	3	18	3,1	2,8	0,2	0	3,1	0	
4	3	19	2,7	4	0,2	0	2,7	0	
4	3	20	2,1	4,8	0,2	0	2,1	0	
4	3	21	1,1	4,7	0,2	0	1,1	0	
4	3	22	0,6	4,5	0,2	0	0,6	0	
4	3	23	0,4	4,3	0,2	0	0,4	0	16632,459

Рисунок К.2 – Дані розрахунку потенційної виробленої енергії на 3 квітня, отримані за допомогою сервісу PVWatts Calculator

Місяць	День	Година	Температура навколишнього середовища (C)	Швидкість вітру (m/s)	Альбедо	Опромінення на поверхні системи (W/m2)	Температура сонячної батареї (C)	Виробництво постійного струму (W)	Добове виробництво струму (W)
4	4	0	0,3	4,3	0,2	0	0,3	0	
4	4	1	0,2	4,4	0,2	0	0,2	0	
4	4	2	0,5	4,8	0,2	0	0,5	0	
4	4	3	1	5	0,2	0	1	0	
4	4	4	1,2	4,9	0,2	0	1,2	0	
4	4	5	1,5	5,1	0,2	0	0	0	
4	4	6	2	5,4	0,2	2,78	2,04	5,649	
4	4	7	2,3	6	0,2	111,598	3,803	264,049	
4	4	8	3,1	6,7	0,2	89,339	4,213	207,441	
4	4	9	4,5	7,3	0,2	93,392	5,593	214,169	
4	4	10	5,7	7,5	0,2	179,528	7,757	415,211	
4	4	11	6,6	7,4	0,2	196,294	8,87	452,396	
4	4	12	7,3	7	0,2	312,672	11,064	725,024	
4	4	13	7,9	6,5	0,2	223,843	10,739	516,734	
4	4	14	8,2	5,8	0,2	158,616	10,377	365,395	
4	4	15	8,2	4,9	0,2	92,706	9,622	212,203	
4	4	16	7,9	4	0,2	9,276	8,061	19,494	
4	4	17	7,5	3,2	0,2	3,705	7,571	7,029	
4	4	18	7,1	2,8	0,2	0	7,1	0	
4	4	19	6,6	2,5	0,2	0	6,6	0	
4	4	20	5,6	2,3	0,2	0	5,6	0	
4	4	21	4,5	2,5	0,2	0	4,5	0	
4	4	22	3,9	3,2	0,2	0	3,9	0	
4	4	23	3,1	3,6	0,2	0	3,1	0	3404,794

Рисунок К.3 – Дані розрахунку потенційної виробленої енергії на 4 квітня, отримані за допомогою сервісу PVWatts Calculator

Місяць	День	Година	Температура навколишнього середовища (C)	Швидкість вітру (m/s)	Альбедо	Опромінення на поверхні системи (W/m2)	Температура сонячної батареї (C)	Виробництво постійного струму (W)	Добове виробництво струму (W)
4	5	0	1,9	3,3	0,2	0	1,9	0	
4	5	1	1	3,4	0,2	0	1	0	
4	5	2	0,4	3,7	0,2	0	0,4	0	
4	5	3	0	4,1	0,2	0	0	0	
4	5	4	-0,2	4,3	0,2	0	-0,2	0	
4	5	5	0,1	4,7	0,2	0	0	0	
4	5	6	1,5	5,2	0,2	35,556	2,025	80,69	
4	5	7	3,3	5,9	0,2	309,827	7,521	745,808	
4	5	8	4,7	6,2	0,2	497,456	11,26	1193,328	
4	5	9	6	6,2	0,2	693,822	15,137	1635,658	
4	5	10	7,3	6,1	0,2	938,676	19,799	2170,164	
4	5	11	8,3	5,9	0,2	996,016	21,857	2279,706	
4	5	12	9	5,8	0,2	819,77	20,273	1884,684	
4	5	13	9,5	5,8	0,2	735,696	19,625	1701,762	
4	5	14	9,5	5,8	0,2	558,643	17,194	1306,815	
4	5	15	9	5,5	0,2	549,562	16,877	1310,719	
4	5	16	7,7	4,8	0,2	315,382	12,532	704,789	
4	5	17	5	3,2	0,2	96,001	6,671	156,299	
4	5	18	1,9	2,2	0,2	0	1,9	0	
4	5	19	0,7	2,1	0,2	0	0,7	0	
4	5	20	-0,2	2	0,2	0	-0,2	0	
4	5	21	-1,1	1,8	0,2	0	-1,1	0	
4	5	22	-1,8	1,7	0,2	0	-1,8	0	
4	5	23	-2,2	1,6	0,2	0	-2,2	0	15170,422

Рисунок К.4 – Дані розрахунку потенційної виробленої енергії на 5 квітня, отримані за допомогою сервісу PVWatts Calculator

Місяць	День	Година	Температура навколишнього середовища (C)	Швидкість вітру (m/s)	Альбедо	Опромінення на поверхні системи (W/m2)	Температура сонячної батареї (C)	Виробництво постійного струму (W)	Добове виробництво струму (W)
4	6	0	-2,3	1,5	0,2	0	-2,3	0	
4	6	1	-2,2	1,3	0,2	0	-2,2	0	
4	6	2	-2,1	1,2	0,2	0	-2,1	0	
4	6	3	-2,1	1,2	0,2	0	-2,1	0	
4	6	4	-2	1,1	0,2	0	-2	0	
4	6	5	-1,1	1	0,2	0	0	0	
4	6	6	1,3	1,3	0,2	176,804	6,133	368,854	
4	6	7	3,9	1,4	0,2	413,924	15,395	976,308	
4	6	8	6,2	1,7	0,2	642,492	22,922	1490,881	
4	6	9	7,6	1,8	0,2	829,138	28,681	1866,877	
4	6	10	9	1,9	0,2	941,851	32,423	2077,626	
4	6	11	10,2	1,9	0,2	900,438	32,559	1975,576	
4	6	12	11,2	2	0,2	808,777	30,872	1786,669	
4	6	13	11,9	1,9	0,2	674,929	28,653	1503,741	
4	6	14	12,1	1,8	0,2	526,586	25,464	1191,242	
4	6	15	11,8	1,8	0,2	399,934	21,973	921,077	
4	6	16	10,7	1,6	0,2	239,196	17,003	538,684	
4	6	17	8	1	0,2	57,392	9,729	122,826	
4	6	18	5,8	1,3	0,2	0	5,8	0	
4	6	19	4,3	1,5	0,2	0	4,3	0	
4	6	20	2,8	1,6	0,2	0	2,8	0	
4	6	21	1,6	1,6	0,2	0	1,6	0	
4	6	22	0,6	1,5	0,2	0	0,6	0	
4	6	23	0,1	1,5	0,2	0	0,1	0	14820,361

Рисунок К.5 – Дані розрахунку потенційної виробленої енергії на 6 квітня, отримані за допомогою сервісу PVWatts Calculator