

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

**«Додаток для моніторингу стану та ефективності
сонячної енергосистеми й акумуляторної батареї»**

Виконав студент групи ІПЗк-23-1
ЯКУСЕВИЧ Владислав Олександрович

Керівник роботи:
ГОДЛЕВСЬКИЙ Юрій Олександрович

Рецензент:
ЧИЖМОТРЯ Олена Геннадіївна

Житомир – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення
Тетяна Вакалюк

«26» лютого 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ЯКУСЕВИЧ Владислав Олександрович**

Керівник роботи: **ГОДЛЕВСЬКИЙ Юрій Олександрович**

Тема роботи: **«Додаток для моніторингу стану та ефективності сонячної енергосистеми й акумуляторної батареї»,**

затверджена Наказом закладу вищої освіти від **«14» лютого 2025 р., №61/с Д**

Вихідні дані для роботи: Об'єктом дослідження є реалізація моніторингу роботи енергогенерувальних та накопичувальних систем.

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Годлевський Ю.О.	27.02.2025р.	27.02.2025р.
2	Годлевський Ю.О.	27.02.2025р.	27.02.2025р.
3	Годлевський Ю.О.	27.02.2025р.	27.02.2025р.

РЕФЕРАТ

Бакалаврська випускна кваліфікаційна робота складається із розробки мобільного застосунку для моніторингу стану та ефективності сонячною панеллю і акумулятором, а також супровідної пояснювальної записки. Документ містить 51 сторінок, включає 35 ілюстрації та 11 таблиць.

Основною метою дослідження є створення мобільної системи, яка дозволяє користувачу в режимі реального часу контролювати робочі параметри сонячної енергетичної установки, включаючи напругу, струм та рівень заряду акумулятора.

Архітектура вирішення завдання базується на використанні протоколу MQTT для обміну даними між пристроями, бази даних InfluxDB для зберігання телеметричних даних, а також бекенд-сервісу на основі NestJS. Мобільний застосунок розроблено з використанням сучасних фреймворків, що забезпечує його кросплатформенність і зручність у користуванні.

Користувач має можливість отримувати актуальні показники напруги, струму та стану заряду батареї.

КЛЮЧОВІ СЛОВА: МОБІЛЬНИЙ ДОДАТОК, MQTT, INFLUXDB, NESTJS, IoT.

					ІПЗ.КР.Б – 121 – 25 – ПЗ				
Змн.	Арк.	№ докум.	Підпис	Дата	Додаток для моніторингу стану та ефективності сонячної енергосистеми й акумуляторної батареї	Літ.	Арк.	Акрушів	
Розроб.		Якусевич В.О.							
Керівник		Годлевський Ю.О.					3	61	
						ФІКТ Гр. ІПЗк-23-1			
Н. Контр.		Чижмoтря О.Г.							
Затверд.		Вакалюк Т.А.							

ABSTRACT

The bachelor's thesis consists of the development of a mobile application for monitoring the status and efficiency of a solar panel and battery, as well as an accompanying explanatory note. The document is 51 pages long, with 35 illustrations and 11 tables.

The main goal of the study is to create a mobile system that allows the user to monitor the operating parameters of a solar power plant in real time, including voltage, current and battery charge level.

The architecture of the solution is based on the use of the MQTT protocol for data exchange between devices, the InfluxDB database for storing telemetry data, and a NestJS-based backend service. The mobile application is developed using modern frameworks, which ensures its cross-platform compatibility and ease of use.

The user can receive up-to-date indicators of voltage, current and battery charge status.

KEYWORDS: MOBILE APPLICATION, MQTT, INFLUXDB, NESTJS, IoT.

					ІІЗ.КР.Б – 121 – 25 – ІІЗ	Арк.
						4
Змн.	Арк.	№ докум.	Підпис	Дата		

ЗМІСТ

РЕФЕРАТ	3
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	6
ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ ТА РОЗРОБКА СИСТЕМИ.....	9
1.1 Постановка задачі	9
1.2 Аналіз аналогів програмного продукту.....	11
1.3 Вибір архітектури веб-орієнтованої системи планування робочого часу	16
1.4 Обґрунтування вибору інструментальних засобів	19
Висновки до першого розділу.....	22
РОЗДІЛ 2. ПРОЕКТУВАННЯ ДОДАТОК ДЛЯ МОНІТОРИНГУ СТАНУ ТА ЕФЕКТИВНОСТІ СОНЯЧНОЇ ЕНЕРГОСИСТЕМИ Й АКУМУЛЯТОРНОЇ БАТАРЕЇ.....	23
2.1 Визначення варіантів використання та об'єктно-орієнтованої структури системи	23
2.2 Розробка бази даних системи	27
2.3 Проектування та реалізація алгоритмів роботи	29
2.4 Реалізація функціоналу додатку	31
Висновок до другого розділу	38
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ДОДАТКОМ.....	39
3.1 Порядок встановлення та налаштування параметрів системи.....	39
3.2 Структура інтерфейсу	40
3.3 Тестування роботи програмного комплексу.....	47
Висновки до третього розділу	49
ВИСНОВКИ	50
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	51
ДОДАТКИ.....	54

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – Application Programming Interface

БД – База даних

SQL – Structured Query Language

CRUD – create, read, update, delete

ORM – Object-relational mapping

Flux – мова запитів для аналізу часових рядів у InfluxDB 2.x основана на концепції потокової обробки даних

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						6
Змн.	Арк.	№ докум.	Підпис	Дата		

ВСТУП

Актуальність теми дослідження зумовлена критичною ситуацією в енергетичній інфраструктурі України, яка зазнає руйнувань внаслідок воєнних дій. Електроенергія є стратегічно важливим ресурсом для населення та економіки, тому розвиток альтернативних джерел енергії, зокрема сонячної енергетики, набуває особливого значення. Сонячні панелі стають надзвичайно актуальними як автономне джерело енергії, особливо у зв'язку із загрозами блекаутів та вимкнення електроенергії.

Мета є створення модульної системи для генерації та акумулювання відновлюваної енергії, об'єднання всіх компонентів в екосистему з управлінням через мобільний додаток, зменшення витрат користувачів і сприяння збереженню довкілля.

Для реалізації поставленої мети необхідно виконати низку взаємопов'язаних завдань, що охоплюють як аналітичні, так і практичні аспекти дослідження. Насамперед передбачається аналіз існуючих рішень у сфері моніторингу та керування сонячними енергетичними установками з метою виявлення їхніх сильних і слабких сторін. На основі отриманих висновків буде спроектовано детальний план розробки програмного комплексу "Storm Energy", що включає визначення його архітектури, ключових функціональних компонентів та взаємозв'язків між ними. Подальшим етапом стане безпосередня розробка програмного забезпечення, яке забезпечить моніторинг основних параметрів сонячної енергетичної системи в режимі реального часу. Завершальною фазою дослідження є проведення тестування створеної системи, оцінка її ефективності та придатності до практичного використання.

Об'єктом дослідження є організація ефективного моніторингу систем генерації та акумулювання відновлюваної енергії.

Предметом дослідження є методи та засоби розробки програмного забезпечення для моніторингу енергетичних систем на основі сонячних панелей і акумуляторів.

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						7
Змн.	Арк.	№ докум.	Підпис	Дата		

У рамках проєкту було використано методології об'єктно-орієнтованого програмування та проєктування на серверній частині, разом із застосуванням принципів розробки дизайну інтерфейсу на клієнтській стороні.

На даний момент мобільний застосунок створений для особистого користування. За необхідності в майбутньому, його можна буде вдосконалити, додавши нові опції або поліпшивши функціонал вже існуючого.

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						8
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1. АНАЛІЗ ТА РОЗРОБКА СИСТЕМИ

1.1 Постановка задачі

У сучасних умовах підвищеного попиту на автономні джерела електропостачання особливого значення набувають рішення, що базуються на використанні відновлюваних джерел енергії, зокрема сонячної. З огляду на часті перебої в енергопостачанні, зростання тарифів на електроенергію, а також загострення екологічних проблем, сонячні енергетичні установки стають невіддільною складовою енергетичної незалежності приватних домогосподарств, комерційних структур та виробничих об'єктів. Проте ефективність роботи таких установок безпосередньо залежить від можливості оперативного моніторингу стану системи, акумуляції енергії та аналізу параметрів її функціонування в режимі реального часу.

У цьому контексті постає актуальне завдання – створення мобільного додатку, здатного забезпечити користувача повноцінною інформаційною підтримкою під час експлуатації сонячної енергетичної системи. Такий програмний продукт має не лише надавати відображення основних технічних параметрів, але й інтегруватися із фізичними компонентами установки, забезпечувати надійне збирання, зберігання, обробку й аналіз даних у режимі реального часу. Крім того, система повинна бути зручною, інтуїтивно зрозумілою, адаптивною до різних моделей обладнання та мати можливість масштабування відповідно до потреб кінцевих користувачів.

Дослідження зосереджується на розробці програмного комплексу "Storm Energy", що слугуватиме інструментом для постійного контролю за генерацією та накопиченням електроенергії від сонячної панелі, відображення поточного стану акумулятора, а також фіксації рівня енергоспоживання. У процесі реалізації даної роботи передбачається здійснити повноцінне проектування архітектури інформаційної системи, розробку клієнтського мобільного додатку, налагодження зв'язку між клієнтською частиною та серверною, а також інтеграцію з джерелом даних на базі MQTT-брокера. Особлива увага приділяється обробці телеметрич-

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						9
Змн.	Арк.	№ докум.	Підпис	Дата		

телеметричних даних, що надходять від системи у вигляді потоків, та їх подальшому збереженню в базі даних типу time series, зокрема InfluxDB.

Зважаючи на те, що в ході виконання роботи не було можливості підключити реальну сонячну установку, джерелом телеметричних даних слугує генератор випадкових чисел, реалізований на стороні сервера. Він формує типові за структурою повідомлення, що відповідають очікуваному формату від реального обладнання, що дає змогу повністю протестувати логіку обробки та візуалізації даних у клієнтському додатку.

З-поміж основних завдань, які необхідно вирішити в межах дослідження, варто виокремити аналіз функціональних і нефункціональних вимог до системи, розробку алгоритмів збору, фільтрації й узагальнення даних з сенсорів, побудову програмної архітектури з чітким розмежуванням модулів за принципом контролер-сервіс-репозиторій (у межах серверної частини), а також створення інтерфейсу користувача, що дозволяє візуалізувати всі ключові параметри роботи енергосистеми. Важливим етапом розробки є реалізація модуля автентифікації користувачів, підтримка реєстрації енергетичних систем за допомогою QR-кодів, а також реалізація механізмів перегляду історичних даних і відображення прогнозованих показників енерговиробництва.

Розробка такого застосунку потребує ретельного обґрунтування вибору програмних засобів і технологій. На боці клієнта доцільно використати фреймворк React Native з Expo для кросплатформеного розвитку мобільного застосунку. Серверну логіку реалізовано засобами NestJS, що відповідає вимогам модульності, масштабованості та підтримує інтеграцію з брокером повідомлень MQTT. Для забезпечення високої продуктивності при роботі з телеметрією використовуються механізми сокет-з'єднання (WebSocket), що дозволяє отримувати оперативні дані без надлишкового навантаження на систему.

Завершальним етапом дослідження стане тестування програмного комплексу в умовах, наближених до реальної експлуатації, що дозволить виявити можливі недоліки, провести оптимізацію внутрішніх процесів і сформулювати практичні рекомендації щодо впровадження системи. Очікується, що реалізація

					ІІЗ.КР.Б – 121 – 25 – ІІЗ	Арк.
						10
Змн.	Арк.	№ докум.	Підпис	Дата		

зазначених функціональних компонентів дозволить сформувати сучасний інструмент для підвищення ефективності використання відновлюваних джерел енергії, забезпечення стабільного електропостачання та зменшення енергозалежності на побутовому й промисловому рівнях.

1.2 Аналіз аналогів програмного продукту

Перед початком розробки нового програмного забезпечення важливо провести всебічний аналіз поточного стану ринку. Вивчення аналогічних продуктів дає змогу виявити програмні рішення з подібним функціоналом, з'ясувати їхні механізми дії, а також встановити слабкі місця та невирішені потреби користувачів. Це є ключовим моментом для генерації інноваційних ідей, спрямованих на поліпшення якості продукту та формування конкурентної позиції. У наступному розділі ми детально проаналізуємо основні програмні аналоги, оцінимо їхні переваги та недоліки, а також визначимо потенційні відмінності нашої розробки від наявних на ринку рішень.

Smart MAIC[1] – це мобільний веб-ресурс (рис. 1.1), що дозволяє відстежувати енергоспоживання та керувати під'єднаними приладами. З його допомогою можна контролювати в реальному часі параметри: напругу, силу струму та загальне енерговитрачання. Користувачі можуть переглядати перелік пристроїв, їхній поточний стан, а також дізнаватися деталі. Платформа містить цифрові віджети, які показують актуальну інформацію про стан мережі. Цей продукт розроблено спеціально для бізнес-клієнтів.

Серед основних переваг платформи варто виокремити її простий та зрозумілий інтерфейс, що полегшує взаємодію користувача з функціоналом системи. Важливою особливістю є підтримка режиму роботи в реальному часі, що забезпечує актуальність отримуваної інформації. Система також дозволяє здійснювати ефективне управління підключеними пристроями та отримувати розширену інформацію про їхній стан, що є особливо корисним у корпоративному середовищі. Гнучка система налаштувань дає змогу адаптувати платформу відповідно до конкретних вимог підприємства, що робить її придатною для різних сценаріїв використання в комерційному секторі.

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						11
Змн.	Арк.	№ докум.	Підпис	Дата		

Разом із тим, система має і певні недоліки. Зокрема, дизайн інтерфейсу сприймається як застарілий і менш привабливий, що може негативно вплинути на користувацький досвід. Відсутність розвиненого модуля аналітики обмежує можливості для глибокого аналізу даних, а при значному навантаженні система може втрачати стабільність і демонструвати зниження продуктивності. Постійна потреба у стабільному підключенні до інтернету також зменшує гнучкість у використанні. Крім того, наявність складної технічної термінології створює труднощі для недосвідчених користувачів, а мобільна версія платформи не завжди має достатній рівень оптимізації.

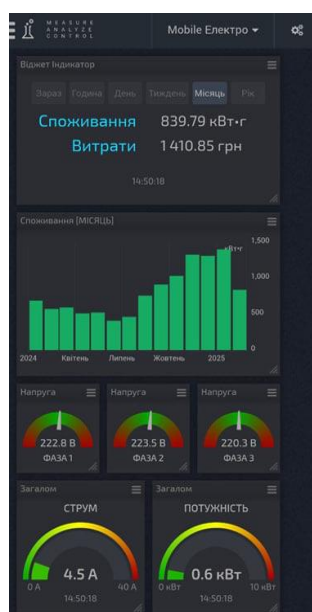


Рис. 1.1 – Сторінка моніторингу електроспоживання

mySolarEdge[2] – це інструмент для моніторингу і вдосконалення роботи сонячних панелей (рис. 1.2). Завдяки йому ви зможете в реальному часі стежити за поточною потужністю, виробленою енергією та ефективністю вашої системи. Програма аналізує дані за різні періоди, надає рекомендації щодо налаштувань і очищення панелей, а також відправляє сповіщення про необхідність технічного обслуговування. Підходить як для домашніх, так і для комерційних користувачів. Інструмент має зручний інтерфейс, а в преміум-версії доступні розширені функції.

Серед переваг цього рішення слід зазначити можливість відстеження параметрів роботи системи в режимі реального часу, що дозволяє оперативно реа-

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		12

гувати на зміни у виробництві енергії. Додаток підтримує аналіз даних за певні проміжки часу, що сприяє виявленню закономірностей у роботі системи та оцінці її ефективності. Користувачі отримують рекомендації щодо оптимізації нахилу панелей та інших параметрів, що впливають на загальну продуктивність, а також автоматичні нагадування про необхідність технічного обслуговування. Додаток забезпечує взаємодію з інверторами та іншими компонентами енергосистеми, що дозволяє комплексно оцінювати її стан. Крім того, інтерфейс програми є інтуїтивно зрозумілим і зручним для користувача.

Разом із тим, застосунок має низку обмежень. Зокрема, безкоштовна версія програми надає обмежений набір функцій, тоді як повний спектр можливостей доступний лише у платному варіанті. Робота додатку залежить від сумісності з конкретним обладнанням, що звужує його універсальність. Для забезпечення повної функціональності необхідне постійне підключення до мережі Інтернет. У деяких випадках точність відображуваних даних може викликати сумніви, що обумовлено як особливостями вимірювального обладнання, так і специфікою обробки інформації. Окрім цього, початківці можуть стикатися з труднощами у використанні окремих функцій через специфічну термінологію або складність налаштувань.

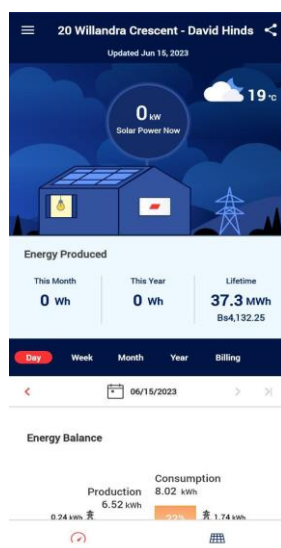


Рис. 1.2 – Сонячна статистика

PV Solar Forecast – це мобільний застосунок (рис. 1.3) для власників сонячних електростанцій, який дозволяє в реальному часі відслідковувати поточну погоду, температуру, а також вироблення електроенергії.

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		13

На головному екрані показано поточний стан станції, прогноз вироблення по годинах і графік добового виробітку, що допомагає швидко оцінити ефективність роботи сонячних панелей. Інтерфейс простий і зручний для швидкого доступу до основної інформації.

До основних переваг застосунку належить надання актуальних даних як про погодні умови, так і про вироблення електроенергії, що є критично важливим для користувачів у плануванні споживання. Простий та лаконічний інтерфейс полегшує навігацію й забезпечує швидкий доступ до необхідної інформації. Додаток також підтримує відображення графічних елементів, зокрема графіків зміни показників упродовж доби, що візуалізує динаміку роботи сонячної станції. Прогноз вироблення на основі погодних даних дає змогу користувачеві адаптувати використання енергії до очікуваних умов.

Водночас слід зазначити й деякі обмеження функціональності системи. Застосунок не адаптується до індивідуальних характеристик конкретної сонячної станції, що може знижувати точність відображуваних прогнозів. Крім того, доступна аналітика є доволі базовою та не дозволяє проводити глибокий аналіз історичних даних або порівнянь. Можливі затримки в оновленні інформації також впливають на оперативність ухвалення рішень. Окремо варто зауважити на відсутності можливості експорту або синхронізації даних з іншими платформами, що обмежує інтеграційний потенціал застосунку.

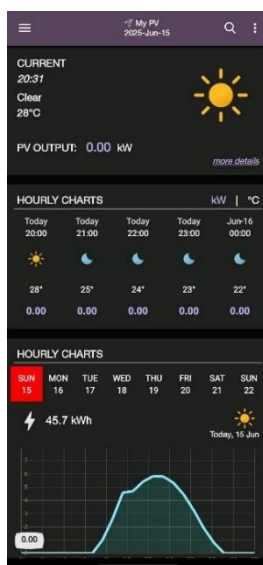


Рис. 1.3 – Логотип додатку PV Solar Forecast

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		14

На основі отриманих даних, складемо порівняльну таблицю (табл. 1.1).

Таблиця 1.1

Порівняльна таблиця аналогів

Критерій	Smart MAIC	mySolarEdge	PV Solar Forecast	StormEnergy
Функціональність	+	+	+	+
Зручність інтерфейсу	-	+	+	+
Аналітика даних	-	+	+	+-
Підтримка пристроїв	+	+	+-	+
Можливості для новачків	-	-	-	+
Залежність від інтернету	-	-	+	+
Вартість (безкоштовний)	+	-	-	+

Після проведеного порівняння основних функціональних характеристик розглянутих програмних продуктів видно, що StormEnergy поєднує в собі ключові переваги конкурентів, при цьому усуваючи їхні недоліки. За функціональністю всі системи забезпечують базовий моніторинг та управління енергоспоживанням, але StormEnergy вирізняється більш інтуїтивним та дружнім інтерфейсом, що робить її доступною навіть для новачків.

Крім того, StormEnergy пропонує розширені аналітичні можливості, яких не вистачає деяким аналогам, і підтримує широкий спектр пристроїв. Важливо, що для роботи платформи потрібне стабільне інтернет-з'єднання, однак це дозволяє забезпечити актуальність даних і швидкість оновлення. Завдяки безкоштовній доступності та комплексному функціоналу StormEnergy має всі шанси стати конкурентоспроможним рішенням на ринку.

1.3 Вибір архітектури веб-орієнтованої системи планування робочого часу

У клієнт-серверній архітектурі клієнт надсилає запит на сервер, який обробляє його та повертає результат клієнту. У серверній частині REST API є центральним компонентом, який відповідає за обробку запитів клієнта, управління ресурсами та повернення відповідей.

API (Application Programming Interface) – це набір правил, що дозволяє програмам спілкуватися одна з одною. Розробник створює API на сервері та дозволяє клієнтам звертатися до нього.

REST (Representational State Transfer) – це архітектурний підхід, що визначає, як API мають виглядати. Цьому набору правил і слідує розробник під час створення свого застосунку. Одне з цих правил каже, що при зверненні до певної адреси ви повинні отримувати певний набір даних (ресурс).

Кожна адреса – маршрут, пакет даних – запит, у той час як результуючий ресурс – відповідь[4].

У REST використовуються HTTP-методи для визначення дій, які виконуються над ресурсами[5]:

GET – отримати дані ресурсу;

POST – створити новий ресурс;

PUT – повністю оновити ресурс або створити новий, якщо його не існує;

PATCH – частково оновити ресурс.

DELETE – видалити ресурс.

WebSocket – це технологія, яка дозволяє встановити двостороннє постійне з'єднання між клієнтом (наприклад, браузером чи мобільним додатком) і сервером. Завдяки цьому з'єднанню обидві сторони можуть обмінюватися повідомленнями в режимі реального часу без потреби у повторному запиті або опитуванні сервера[6].

MQTT – це легкий та відкритий протокол обміну повідомленнями, створений для швидкої та стабільної передачі даних між пристроями в умовах обмеженого каналу зв'язку. Його архітектура базується на моделі «публікація-підписка», що дозволяє легко масштабувати систему та забезпечує гнучкий обмін

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						16
Змн.	Арк.	№ докум.	Підпис	Дата		

даними. У типовій конфігурації беруть участь брокер, клієнти та видавець даних[7].

Між клієнтом та сервером потрібно створювати передачу даних, і є різні формати передачі. Один із форматів це JSON.

JSON (JavaScript Object Notation) – найпоширеніший та найпопулярніший формат для передачі структурованих даних через REST API через його простоту та легкість для аналізу як клієнтами, так і серверами.

В архітектурі REST API питання автентифікації є надзвичайно важливим, оскільки необхідно захищати API від несанкціонованого доступу. Один із варіантів це веб-токени JSON. Облікові дані та інші параметри доступу передаються у вигляді структур даних JSON. Ці токени доступу можуть бути криптографічно підписані, що робить їх одним з найефективніших способів контролю доступу до REST API[8].

Контейнеризація – це процес розгортання програмного забезпечення разом із усіма його необхідними компонентами, такими як код, бібліотеки, фреймворки та інше, у вигляді ізолюваного контейнера[9].

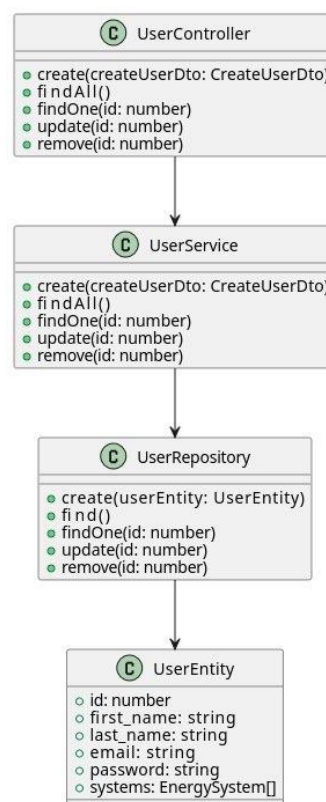


Рис. 1.4 – Патерн Controller-Service-Repository

На цій діаграмі зображено (рис. 1.4), як у додатку організована робота з користувачами за допомогою патерну Controller-Service-Repository. Вся взаємодія починається з UserController, який приймає запити від клієнта й пересилає їх у UserService. Сервіс відповідає за бізнес-логіку, обробляє дані та звертається до UserRepository для збереження або отримання інформації з бази даних. Репозиторій безпосередньо працює з сутністю UserEntity, що описує структуру користувача у системі: містить його ім'я, прізвище, email, пароль і список енергетичних систем. Така архітектура забезпечує чіткий розподіл обов'язків та робить код чітким і зрозумілим[10][11].

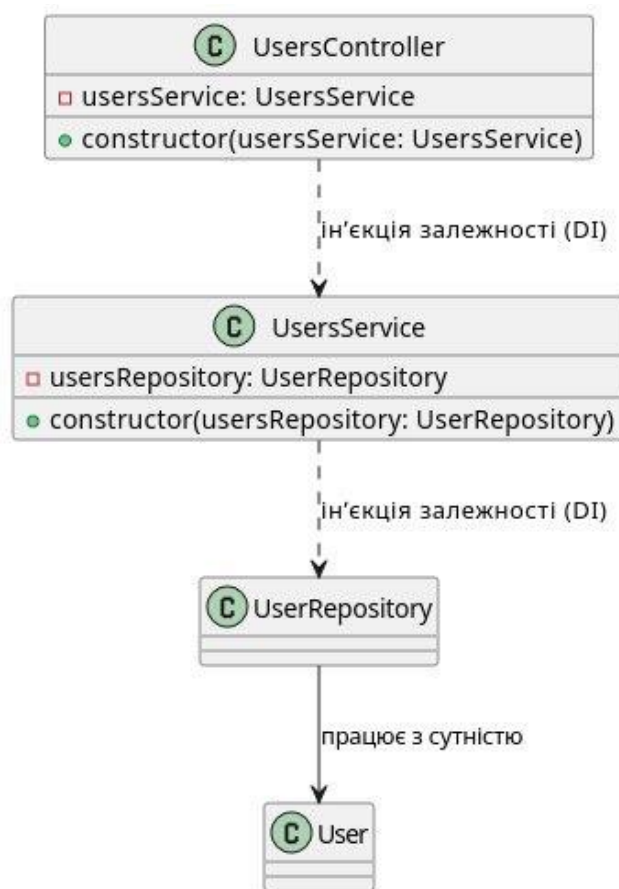


Рис. 1.5 – Патерн Dependency injection

Ця діаграма класів ілюструє (рис. 1.5), як працює ін'єкція залежностей у модулі користувачів. Контролер не створює сервіс вручну, а отримує готовий об'єкт UserService через DI – це робить NestJS. Сервіс, у свою чергу, так само через DI отримує репозиторій UserRepository для роботи з даними. UserRepository вже взаємодіє із сутністю User, яка описує структуру користувача.

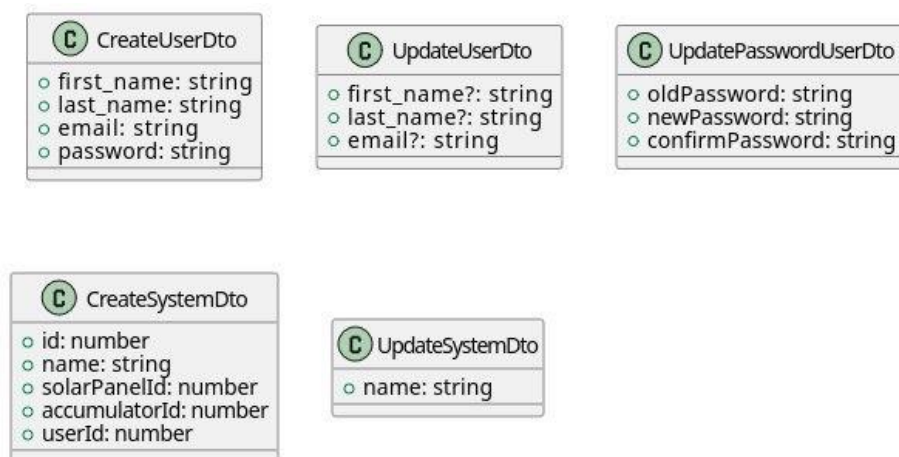


Рис. 1.6 – Патерн Data Transfer Object

Ця діаграма показує (рис. 1.6), як у системі для кожної дії використовується свій окремий об'єкт DTO – наприклад, для реєстрації, зміни профілю чи додавання енергосистеми. Кожен DTO визначає лише ті поля, які дійсно потрібні для конкретної операції, тому взаємодія між клієнтом і сервером стає максимально чіткою та безпечною[12].

1.4 Обґрунтування вибору інструментальних засобів

Для розробки мобільного додатку важливо вибрати відповідні інструменти та технології, які відповідатимуть вимогам.

Інструменти для розробки:

Visual Studio Code (VS Code) – це безкоштовний редактор коду, створений на основі фреймворку Electron. Дозволяє розробляти багато платформні десктопні застосунки, використовуючи веб-технології.

Головна перевага VS Code – універсальність. Редактор не обмежений конкретною мовою програмування. З його допомогою можна створювати веб-сайти, мобільні додатки, працювати з базами даних і тестувати сервіси. Велика бібліотека плагінів розширює функціонал редактора. А якщо потрібний модуль знайти не вдалося, його можна створити самостійно[13].

Docker – це платформа, що дає змогу створювати контейнеризовані середовища, у яких програмне забезпечення разом із його залежностями та налаштуваннями може бути упаковане, перенесене та запущене на будь-якій цільовій системі[14].

Postman – це програмне забезпечення для тестування та розробки API, що виконує роль HTTP-клієнта з інтуїтивно зрозумілим інтерфейсом. Він дозволяє створювати й надсилати HTTP-запити, налаштовувати параметри взаємодії з API та отримувати структуровані відповіді. Інструмент підтримує організацію запитів у колекції, роботу з різними середовищами та додавання контрольних точок для перевірки результатів. Завдяки функції Collection Runner можливе автоматизоване тестування, що спрощує перевірку стабільності API[15].

Є 4 основних дієслова HTTP, які використовуються в запитах для взаємодії з ресурсами в системі REST:

GET – отримати певний ресурс (за ідентифікатором) або набір ресурсів;

POST – створити новий ресурс;

PUT – оновити певний ресурс (за ідентифікатором);

DELETE – видалити певний ресурс за ідентифікатором.

Технології:

Для клієнтської частини буде використовуватися:

React Native – це фреймворк із відкритим вихідним кодом, який дозволяє розробникам створювати власні мобільні програми за допомогою JavaScript або TypeScript[16].

React Native вирізняється можливістю повторного використання коду між Android та iOS, що дозволяє суттєво зекономити час розробки. Завдяки нативним компонентам забезпечується висока продуктивність додатків. Підтримка гарячого перезавантаження дає змогу оперативно вносити зміни без повного перезапуску. Розвинена екосистема та активна спільнота сприяють швидкому вирішенню типових завдань, а кросплатформенний підхід робить цей фреймворк економічно ефективним.

CSS – це мова таблиць стилів, яка використовується для опису подання документа, написаного в HTML або XML. CSS описує, як елементи мають відображатися на екрані, на папері, у мовленні чи на інших носіях[17].

JavaScript – це кросплатформена об'єктно-орієнтована мова програмування, яка використовується для створення інтерактивних вебсторінок, зокрема аніма-

					ІІЗ.КР.Б – 121 – 25 – ІІЗ	Арк.
						20
Змн.	Арк.	№ докум.	Підпис	Дата		

цій, кнопок та динамічного контенту. У серверному середовищі, наприклад з використанням Node.js, вона дозволяє реалізовувати розширену функціональність, таку як обмін даними в реальному часі. У браузері JavaScript може взаємодіяти з об'єктами сторінки, надаючи повний програмний контроль над інтерфейсом[18].

TypeScript – це надбудова над JavaScript, яка додає підтримку типів. Іншими словами, він працює як звичайний JavaScript, але дозволяє точніше описувати структуру коду. Це допомагає уникати помилок ще до запуску програми та робить роботу з великими проєктами значно зручнішою[19].

Застосовані технології дають змогу розробити сучасний і зручний клієнтський додаток, використовуючи передові підходи, які забезпечують надійність та високу якість програмного забезпечення.

Для серверної частини буде використовуватися:

Node.js – це однопоточне кросплатформове середовище виконання з відкритим вихідним кодом і бібліотека, яка використовується для запуску веб-додатків, написаних на JavaScript, поза браузером клієнта[20].

Nestjs – це платформа для створення ефективних, масштабованих серверних програм Node.js. Він використовує прогресивний JavaScript, створений і повністю підтримує TypeScript і поєднує елементи ООП[21].

Socket.IO – це сучасна технологія, яка забезпечує швидкий і двосторонній обмін даними між сервером і клієнтом у режимі реального часу[22].

TypeORM – це ORM, який працює на різних платформах, включно з NodeJS, браузерами та мобільними додатками. Він підтримує сучасний JavaScript і TypeScript, полегшуючи роботу з базами даних як у невеликих, так і у великих проєктах. TypeORM допомагає зосередитись на логіці програми, спрощуючи управління даними[23].

PostgreSQL – це об'єктно-реляційна система управління базами даних, яка першою впровадила низку інноваційних рішень, що згодом стали стандартом у багатьох комерційних базах даних[24].

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						21
Змн.	Арк.	№ докум.	Підпис	Дата		

InfluxDB – це спеціалізована система баз даних, створена для зберігання, обробки та візуалізації часових рядів даних, тобто інформації, яка змінюється з часом і прив’язана до конкретних моментів. Такий підхід дозволяє ефективно працювати з показниками, що постійно оновлюються – наприклад, даними від сенсорів або систем моніторингу[25].

Обране технологічне рішення ідеально підходить для реалізації серверної частини, адже дозволяє вибудувати стійку та масштабовану архітектуру, яка легко справляється з обробкою даних у режимі реального часу. Система забезпечує стабільну взаємодію з клієнтським додатком, оперативну передачу інформації та гнучкість у налаштуванні логіки застосунку.

Висновки до першого розділу

У першому розділі було сформульовано основні задачі, які необхідно вирішити для розробки ефективної системи моніторингу сонячною енергетичною установкою "Storm Energy". Проведено детальний аналіз існуючих програмних аналогів, на основі якого визначено переваги та недоліки конкурентів. Це дозволило обґрунтовано визначити унікальні риси запропонованого програмного рішення.

На основі проведеного аналізу було обрано клієнт-серверну архітектуру з REST API, WebSocket і MQTT, що забезпечує надійний обмін даними у реальному часі. Розглянуто ключові архітектурні патерни – Controller-Service-Repository, Dependency Injection та Data Transfer Object – що гарантують модульність, масштабованість і підтримуваність коду.

Також обґрунтовано вибір технологічного стеку як для клієнтської, так і для серверної частини, що дає змогу реалізувати продуктивне, адаптивне та масштабоване рішення, яке відповідає сучасним вимогам до подібного класу систем.

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						22
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 2. ПРОЕКТУВАННЯ ДОДАТОК ДЛЯ МОНІТОРИНГУ СТАНУ ТА ЕФЕКТИВНОСТІ СОНЯЧНОЇ ЕНЕРГОСИСТЕМИ Й АКУМУЛЯТОРНОЇ БАТАРЕЇ

2.1 Визначення варіантів використання та об'єктно-орієнтованої структури системи

У цьому розділі розглянуто про функціонал, втілений у застосунку для спостереження за станом і показниками ефективності сонячної енергосистеми та акумулятора. Головне завдання системи – надати користувачеві зручний засіб для контролю параметрів виробництва енергії, оцінки продуктивності та своєчасного виявлення неполадок у роботі обладнання. Для цього була побудовано діаграму варіантів використання (рис. 2.1).

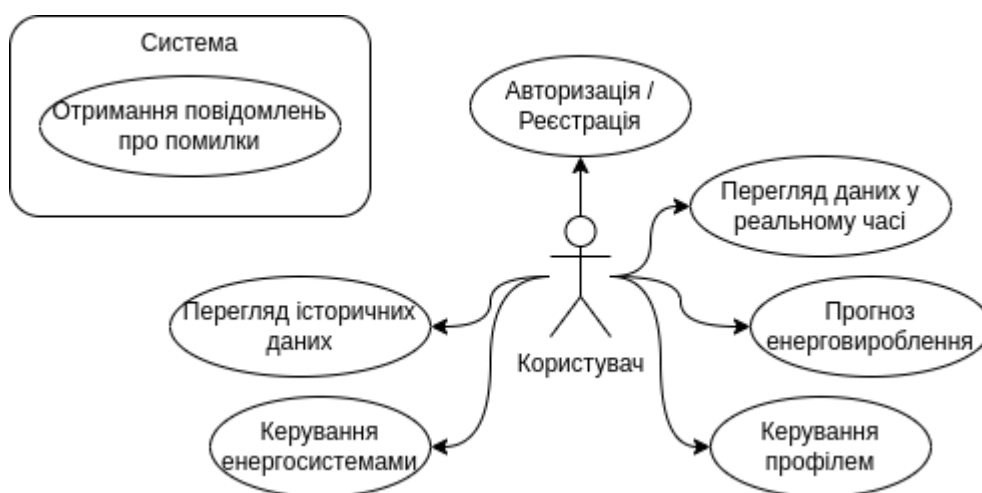


Рис. 2.1 – Діаграма варіантів використання

Опис діаграм варіантів використання, представлених у таблицях 2.1 – 2.7 (табл. 2.1 – 2.7).

Таблиця 2.1

Назва варіанту використання	Авторизація / Реєстрація
Основні актори	Користувач
Короткий опис	Користувач проходить реєстрацію в системі або входить у свій персональний кабінет. Якщо введені дані невірні, він отримує відповідне сповіщення про помилку.

Таблиця 2.2

Назва варіанту використання	Керування профілем
Основні актори	Користувач
Короткий опис	Користувач має можливість переглядати, редагувати та оновлювати свої особисті дані. Система надає підтвердження успішного внесення змін або повідомляє про виниклі помилки.

Таблиця 2.3

Назва варіанту використання	Керування енергосистемами
Основні актори	Користувач
Короткий опис	Користувач має можливість додавати, редагувати та видаляти свої особисті енергосистеми. Система виконує верифікацію введеної інформації та показує відповідні повідомлення у разі помилок або успішної обробки даних.

Таблиця 2.4

Назва варіанту використання	Перегляд даних у реальному часі
Основні актори	Користувач
Короткий опис	Користувач відстежує актуальні показники параметрів енергетичної системи. Уся інформація оновлюється в режимі реального часу.

Таблиця 2.5

Назва варіанту використання	Перегляд історичних даних
Основні актори	Користувач

Продовження таблиці 2.5

Короткий опис	Користувач має можливість обрати конкретний період часу та переглядати історію генерації та споживання енергії у форматі графіків або таблиць.
---------------	--

Таблиця 2.6

Назва варіанту використання	Прогноз енерговироблення
Основні актори	Користувач
Короткий опис	На основі даних з метеостанцій система видає прогноз виробництва енергії, враховуючи поточні погодні фактори.

Таблиця 2.7

Назва варіанту використання	Отримання повідомлень про помилки
Основні актори	Система
Короткий опис	Користувачеві надходять сповіщення про несправності в роботі енергосистеми або додатку.

Основна мета створення додатку є надання користувачам зручного засобу для відстеження стану та ефективності роботи сонячної енергосистеми та акумулятора в режимі реального часу, з доступом до історичних даних і прогнозування енерговиробництва.

З урахуванням бізнес-вимог, визначено, що основною функціональною ціллю є розробка мобільного застосунку, який забезпечуватиме моніторинг параметрів сонячної енергосистеми та акумулятора у реальному часі. У процесі формування вимог до системи враховувались очікування кінцевих користувачів, зокрема щодо зручності інтерфейсу, простоти використання та мобільності платформи. Проєкт орієнтовано на побутових користувачів і представників малого бізнесу, які впроваджують відновлювані джерела енергії у власну інфраст-

руктуру, що зумовлює необхідність створення доступного та інформативного рішення для контролю енерговикористання.

Користувач додатку повинен мати можливість швидко зареєструватися та авторизуватися, щоб отримати доступ до всіх функцій. Після входу він може керувати своїм обліковим записом, додавати власні енергосистеми, а також зручно редагувати чи видаляти їх за потреби. Головна ідея – забезпечити простий і зрозумілий процес керування енергосистемами: користувач може в реальному часі спостерігати за показниками.

Важливо, щоб інтерфейс був інтуїтивно зрозумілим, виглядав сучасно й приємно, а користувачу не доводилося довго шукати потрібні функції чи читати інструкції. Уся робота з додатком має бути максимально простою та логічною – від керування акаунтом до перегляду стану систем.

Продуктивність системи – теж ключовий момент. Усі дані повинні оновлюватися майже миттєво, екрани відкриватися швидко, без затримок. Додаток має стабільно працювати навіть при нестабільному чи повільному інтернеті, не зависати й не «падати», а дані користувача не повинні губитися. Якщо виникають якісь помилки або проблеми з енергосистемами, користувач одразу отримує відповідне сповіщення чи попередження – це дає змогу швидко реагувати на ситуацію.

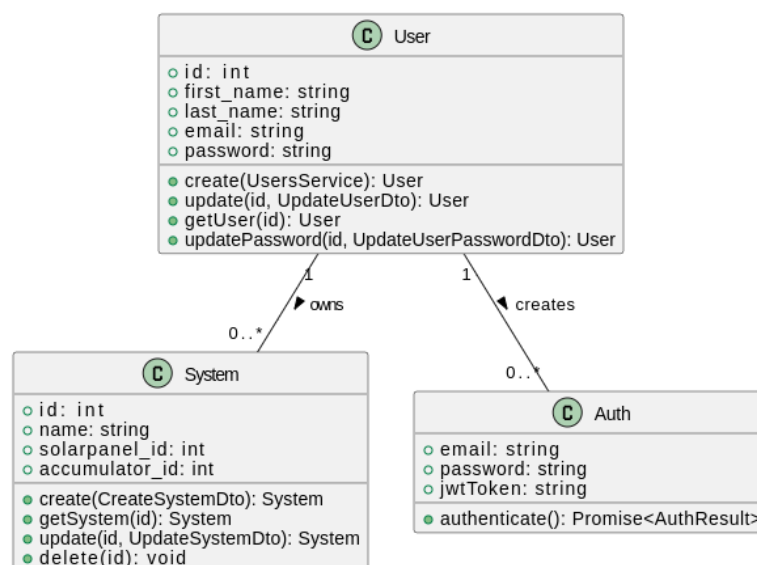


Рис. 2.2 – Діаграма класів

У цій системі (рис. 2.2) кожен користувач має свої персональні дані – ім'я, прізвище, електронну пошту та пароль. Він може створити обліковий запис, увійти в систему через авторизацію, оновлювати свій профіль і змінювати пароль. Авторизація відбувається через введення email і пароля, після чого користувач отримує JWT-токен, який дозволяє безпечно користуватись додатком.

Користувач може створювати одну або кілька енергосистем, кожна з яких має власну назву. Ці системи можна переглядати, оновлювати або видаляти. Уся логіка побудована так, щоб користувач керував тільки своїми системами, а система залишалася стабільною, безпечною й зрозумілою.

2.2 Розробка бази даних системи

У системі використовується база даних, структура якої відповідає сутностям додатку та логіці його роботи. Таблиці створені за допомогою бібліотеки TypeORM, що дозволяє описувати їх у вигляді класів із відповідними анотаціями. Такий підхід спрощує створення, читання та підтримку коду, забезпечуючи зручне об'єктно-реляційне відображення. Взаємодія з базою даних організована через контролери, які приймають зовнішні HTTP-запити та передають їх у сервіси для обробки. Сервіси, своєю чергою, працюють з репозиторіями, що надають доступ до операцій із базою. Така архітектура сприяє модульності, забезпечує гнучке розділення обов'язків і полегшує подальшу підтримку та розвиток функціоналу.

Структура бази включає дві основні таблиці, які відображають ключові логічні зв'язки між даними. Таблиця 2.8 «Users» містить інформацію про користувачів системи, зокрема їхні ідентифікатори, облікові дані та пов'язана із таблицею енергосистем за принципом "один до багатьох", що дає змогу одному користувачеві володіти кількома установками. Таблиця 2.9 «System» призначена для зберігання даних про конкретні енергетичні установки, такі як сонячні панелі та акумулятори, і пов'язана з таблицею користувачів відношенням "багато до одного". Це дозволяє будувати логіку доступу до системи відповідно до обліково-

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						27
Змн.	Арк.	№ докум.	Підпис	Дата		

го запису користувача та організовувати зручне групування елементів за власниками.

Кожна з наведених таблиць описана нижче:

Таблиця 2.8

Опис таблиці Users

Поле	Тип	Призначення
id	int, PK, unique, increment	Унікальний ідентифікатор
first_name	varchar(255)	Ім'я
last_name	varchar(255)	Прізвище
email	varchar(255), unique	Електронна пошта
password	varchar(255)	Пароль

Таблиця 2.9

Опис таблиці System

Поле	Тип	Призначення
id	int, unique	Унікальний ідентифікатор
name	varchar(255)	Ім'я системи
solarpanel_id	int, unique	Ідентифікатор сон. пан.
accumulator_id	int, unique	Ідентифікатор акумулятора
user_id	int, FK	Ідентифікатор користувача

У системі використовується база даних InfluxDB (табл. 2.10), яка оптимізована для зберігання телеметричних даних, що постійно оновлюються. Саме сюди надходять показники стану енергосистеми – зокрема заряд акумулятора, потужність з панелей та споживання з мережі. Дані фіксуються з міткою часу, ідентифікатором користувача та енергосистеми, що дозволяє точно відстежувати зміну параметрів у режимі реального часу. Завдяки цьому користувач бачить актуальні значення в додатку без затримок, а також має змогу переглядати історію зміни даних.

Опис талбиці System

Компонент	Назва	Призначення
Мітка часу	_time	Час запису
Теги	userId	Ідентифікатор користувача
	systemId	Ідентифікатор системи
Поля	current_power	Вироблено потужність
	grid_power	Спожито потужність
	battery_charge	Поточний рівень заряду акумулятора

InfluxDB дає змогу не лише фіксувати значення в реальному часі, а й переглядати історію, будувати графіки та виконувати аналітичні запити. Це забезпечує гнучкість у візуалізації, а також відкриває можливість для прогнозування на основі накопичених даних.

2.3 Проектування та реалізація алгоритмів роботи

Додаток простий у використанні та містить усі необхідні функції, що дозволяє користувачам зручно з ним взаємодіяти. Алгоритм взаємодії між клієнтським додатком та сервером наведено на діаграмі активностей (рис. 2.3), яка ілюструє послідовність обміну даними під час авторизації користувача.

Процес розпочинається з введення електронної пошти та пароля у клієнтському додатку. Після цього ці дані надсилаються на сервер, який обробляє отриманий запит, здійснюючи пошук відповідного запису в базі даних. У разі виявлення облікового запису здійснюється перевірка правильності введеного пароля. Якщо автентифікаційні дані вірні, сервер формує JWT-токен, що підтверджує успішну авторизацію, та надсилає його клієнтському додатку. У разі невідповідності пароля або відсутності користувача в базі, сервер повертає повідомлення про помилку. На основі отриманої відповіді клієнт або зберігає токен для подальшої роботи із захищеними ресурсами, або відображає відповідне повідомлення про помилку користувачеві.

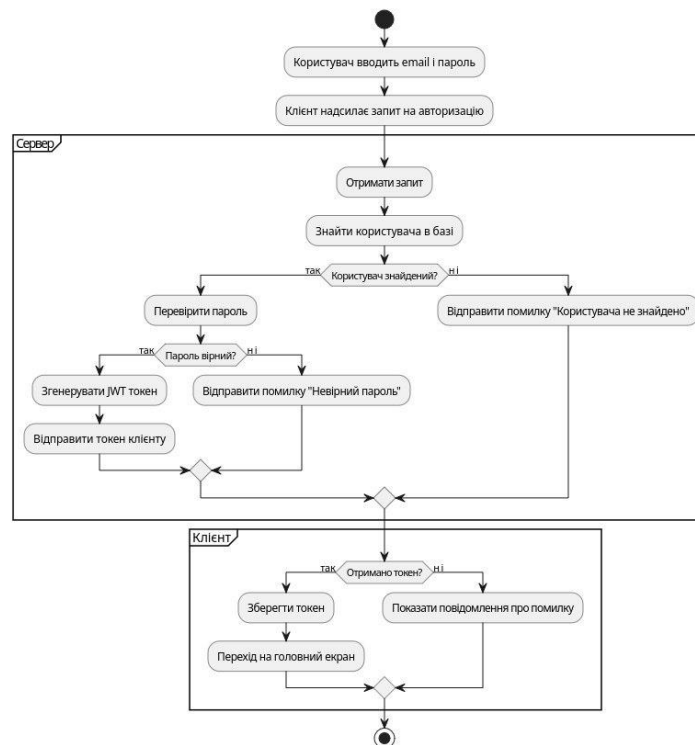


Рис. 2.3 – Діаграма активності взаємодія користувача з мобільним додатком

Ця діаграма чітко показує (рис. 2.3) послідовність дій під час авторизації, а також взаємодію між клієнтським додатком і сервером. Такий підхід дозволяє забезпечити безпечний вхід у систему та зручну обробку помилок на стороні користувача.

Щоб користувач бачив актуальні дані без необхідності оновлювати екран вручну, додаток встановлює WebSocket-з'єднання з сервером. Після відкриття головного екрану клієнт автоматично надсилає параметри своєї енергосистеми, і сервер періодично надсилає у відповідь актуальні показники – зокрема, заряд акумулятора, потужність з панелей або мережі. Оновлення відбувається автоматично через короткі інтервали часу, і вся інформація оновлюється прямо в інтерфейсі користувача.

Алгоритм взаємодії між клієнтом і сервером передбачає встановлення з'єднання з WebSocket-сервером, після чого клієнт передає ідентифікатори користувача та відповідної енергосистеми. Отримавши запит, сервер звертається до бази даних або джерела телеметрії, отримує актуальні параметри та надсилає їх назад клієнту. У свою чергу, клієнтський додаток відображає отримані значення у користувацькому інтерфейсі. Через заданий проміжок часу цей процес повторю-

ється, що забезпечує постійну актуалізацію інформації без дій з боку користувача.

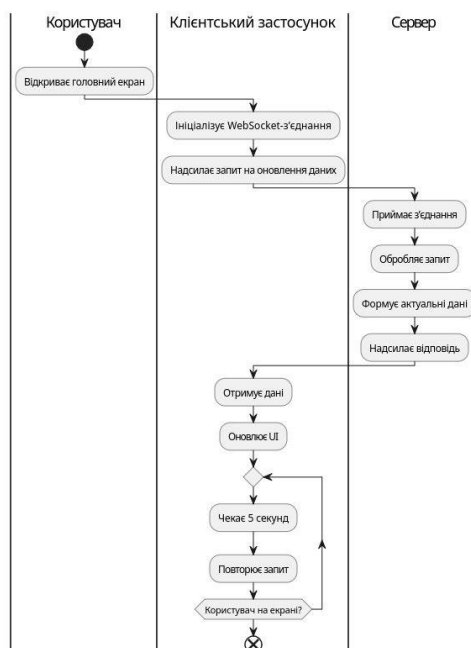


Рис. 2.4 – Діаграма активності оновлення даних в реальному часі

На діаграмі активностей представлено (рис. 2.4) алгоритм отримання оновлених даних від сервера через WebSocket при відкритті головного екрану. Взаємодія починається з ініціалізації з'єднання та надсилання запиту на оновлення. Сервер приймає запит, обробляє його та повертає клієнту актуальні дані моніторингу.

Після отримання відповіді інтерфейс користувача оновлюється. Далі клієнт кожні 5 секунд повторює запит, якщо користувач продовжує перебувати на цьому екрані. Таким чином, забезпечується безперервне оновлення показників у реальному часі без потреби ручного оновлення.

2.4 Реалізація функціоналу додатку

Розглянемо ключові етапи створення мобільного додатку для моніторингу сонячної енергетичної системи.

Додавання нової енергосистеми (рис. 2.5) реалізовано на серверній частині за допомогою методу, що отримує дані про систему, асоціює їх із конкретним користувачем і зберігає у базі даних. Перед створенням запису виконується перевірка існування користувача, після чого створюється екземпляр системи, який зберігається через репозиторій. У випадку помилки обробки запиту повертається

виняток із відповідним повідомленням. Такий підхід дозволяє гарантувати цілісність даних і обробляти нештатні ситуації без зупинки сервісу.



```

1  async create(createSystemDto: CreateSystemDto): Promise<System> {
2      const user = await this.userService.findOne(createSystemDto.userId);
3
4      try {
5          const system = this.systemRepository.create({
6              ...createSystemDto,
7              user,
8          });
9
10         return await this.systemRepository.save(system);
11     } catch (error) {
12         throw new InternalServerErrorException('Failed to create system.');
```

Рис. 2.5 – Метода створення системи на стороні сервера

У мобільному додатку реалізовано зручний механізм сканування QR-коду, який дозволяє користувачеві швидко додати нову енергосистему без необхідності ручного введення даних.

Цей фрагмент коду (рис. 2.6) перевіряє, чи надано доступ до камери. Якщо дозволу немає, користувач бачить повідомлення з кнопкою для надання доступу; якщо об'єкт дозволу ще не отримано, інтерфейс взагалі не відображається.



```

1  const navigation = useNavigation<ScanQRScreenNavigationProp>();
2  const [permission, requestPermission] = useCameraPermissions();
3  const [scanned, setScanned] = useState(false);
4
5  if (!permission) return <View />;
6
7  if (!permission.granted) {
8      return (
9          <View style={styles.messageContainer}>
10             <Text style={styles.message}>Нам потрібен дозвіл на використання камери</Text>
11             <PrimarySubmit title="Надати доступ до камери" onPress={requestPermission} />
12         </View>
13     );
14 }
```

Рис. 2.6 – Перевірка доступу до камери

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		32

Цей код (рис. 2.7) обробляє подію сканування QR-коду: зчитує дані, відправляє їх POST-запитом на сервер для додавання нової енергосистеми, і залежно від відповіді показує користувачу повідомлення про успіх або помилку. У разі помилки дозволяє повторити сканування.



```

1  const handleBarcodeScanned = async (result: BarcodeScanningResult) => {
2    if (scanned) return;
3
4    try {
5      setScanned(true);
6      const qrData = JSON.parse(result.data);
7      const response = await fetch(
8        "http://192.168.50.107:3000/system",
9        {
10       method: 'POST',
11       headers: { 'Content-Type': 'application/json', },
12       body: JSON.stringify(qrData),
13     }
14   );
15
16   if (!response.ok) throw new Error('Щось пішло не так');
17
18   Alert.alert(
19     "Успішно", "Систему додано!",
20     [{
21       text: "OK",
22       onPress: () => navigation.navigate("AddEnergySystem", { qrData }),
23     },]
24   );
25 } catch (error) {
26   console.error("Error sending QR data:", error);
27   Alert.alert(
28     "Помилка", "QR-код некоректний або сервер не відповідає",
29     [{ text: "OK", onPress: () => setScanned(false) } ]
30   );
31   Alert.alert('Помилка', (error as Error).message);
32 }
33 };

```

Рис. 2.7 – Обробка сканування QR-коду

На цьому екрані (рис. 2.8) користувач бачить вікно з активною камерою та підказкою, куди саме потрібно розмістити QR-код для успішного сканування. Додатково використовується графічна рамка для зручності наведення. Завдяки такому оформленню інтерфейсу користувачу не потрібно здогадуватись про правильне положення коду: всі необхідні елементи відразу спрямовують увагу та спрощують виконання завдання.

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		33



```

1  return (
2    <View style={styles.scannerContainer}>
3      <CameraView
4        style={styles.camera}
5        barcodeScannerSettings={{ barcodeTypes: ["qr"] }}
6        onBarcodeScanned={handleBarcodeScanned}
7      >
8        <Text style={styles.scannerTitle}>Помістіть QR-код у рамку</Text>
9        <Image source={require('../../assets/scaner-mask.png')} style={styles.scanerMask} />
10      </CameraView>
11    </View>
12  );

```

Рис. 2.8 – Інтерфейс сканування QR-коду

На сервері реалізовано логіку запису телеметричних показників енергосистеми у базу даних InfluxDB приклад, який наведено (рис. 2.9). Після отримання даних формується спеціальна структура, яка містить теги – ідентифікатори користувача та системи, а також числові поля з поточною потужністю, рівнем заряду акумулятора та споживанням електроенергії з мережі.

Ця структура передається до бази даних, де вона зберігається як окрема точка часових рядів. Завдяки примусовій синхронізації, дані одразу записуються у сховище, що дозволяє забезпечити актуальність інформації в інтерфейсі користувача. У разі виникнення помилки формується повідомлення про її причину, що спрощує діагностику.



```

1  async writeData(handleData: BodyHandleData) {
2    await this.writeSolarpanelData(handleData);
3  }
4  return { message: 'Data written successfully' };
5  }
6
7  private async writeSolarpanelData(handleData: BodyHandleData): Promise<void> {
8    try {
9      const point = new Point('system')
10       .tag('userId', String(handleData.userId))
11       .tag('systemId', String(handleData.systemId))
12       .floatField('current_power', handleData.currentPower)
13       .floatField('grid_power', handleData.gridPower)
14       .floatField('battery_charge', handleData.percentage);
15
16      this.writeApi.writePoint(point);
17
18      await this.writeApi.flush();
19    } catch (error) {
20      throw new InternalServerErrorException('Error solar panel. ');
21    }
22  }

```

Рис. 2.9 – Методи запису даних в базу даних

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						34
Змн.	Арк.	№ докум.	Підпис	Дата		

Для отримання останніх телеметричних показників енергосистеми з бази даних InfluxDB формується запит мовою Flux (рис. 2.10), у якому вказується джерело даних, часовий діапазон та фільтри за типом пристрою, ідентифікаторами користувача й енергосистеми, а також необхідними параметрами – такими як заряд акумулятора, потужність від сонячної панелі та споживання з мережі.

Після цього дані групуються, сортуються у зворотному хронологічному порядку та обмежуються до останнього значення. Отриманий результат обробляється, перетворюється у зручний формат і надсилається клієнту для відображення на екрані.



```

1  async queryRealDataTime(userId: number, systemId: number, bodyFluxQuery: BodyFluxQueryRealTime) {
2    try {
3      const fluxQuery =
4        `from(bucket: "${process.env.INFLUX_BUCKET}")
5          |> range(start: 0)
6          |> filter(fn: (r) =>
7            r._measurement == "${bodyFluxQuery.typeProject}" and
8            r.userId == "${userId}" and
9            r.systemId == "${systemId}" and
10           (r._field == "battery_charge" or r._field == "current_power" or r._field == "grid_power")
11          )
12          |> group(columns: ["_field"])
13          |> sort(columns: ["_time"], desc: true)
14          |> limit(n: 1)`;
15
16     const myQuery = async () => {
17       const result: Record<string, number> = {};
18
19       for await (const { values, tableMeta } of this.queryApi.iterateRows(fluxQuery)) {
20         const o = tableMeta.toObject(values);
21         result[o._field] = o._value;
22       }
23
24       return result;
25     };
26
27     return myQuery();
28   } catch (error) {
29     throw new NotAcceptableException('error. ');
30   }
31 }

```

Рис. 2.10 – Отримання останніх даних

Під час автентифікації (рис. 2.11) система спочатку перевіряє, чи існує користувач із вказаною електронною адресою. Якщо такий знайдений, виконується звірка пароля. У разі успішної перевірки формується токен доступу

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						35
Змн.	Арк.	№ докум.	Підпис	Дата		

на основі ідентифікатора, email та імені користувача. Сформований JWT-токен повертається разом із основною інформацією про користувача для подальшого використання в клієнтському додатку.



```

1  async authenticate(signInData: LoginAuthDto): Promise<AuthResult> {
2      const user = await this.validateUser(signInData);
3
4      if (!user) {
5          throw new UnauthorizedException('NOT FOUND. ');
6      }
7
8      return this.signIn(user);
9  }
10
11 async validateUser(signInData: LoginAuthDto): Promise<UserData | null> {
12     const user = await this.userService.findUserByEmail(signInData.email, true);
13
14     if (user) {
15         await this.userService.chekingPassword(signInData.password, user.password);
16
17         return {
18             userId: user.id,
19             email: user.email,
20             firstName: user.first_name,
21             lastName: user.last_name,
22         }
23     }
24
25     return null;
26 }
27
28 async signIn(user: UserData) {
29     const tokenPayload = {
30         sub: user.userId,
31         email: user.email,
32         firstName: user.firstName,
33         lastName: user.lastName
34     }
35
36     const accessToken = await this.jwtService.signAsync(tokenPayload);
37
38     return {
39         accessToken,
40         userId: user.userId,
41         email: user.email,
42         firstName: user.firstName,
43         lastName: user.lastName
44     }
45 }

```

Рис. 2.11 – Автентифікація та створення JWT-токена

На етапі обробки запиту (рис. 2.12) система перевіряє наявність токена у заголовку авторизації. Якщо токен не знайдено – викликається помилка неавтори-

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						36
Змн.	Арк.	№ докум.	Підпис	Дата		

ризованого доступу. У разі наявності токена відбувається його перевірка. Якщо він дійсний, з нього зчитується інформація про користувача, яка додається до об'єкта запиту для подальшої обробки.



```

1  async canActivate(context: ExecutionContext) {
2    const request = context.switchToHttp().getRequest();
3    const authorization = request.headers.authorization;
4    const token = authorization?.split(' ')[1];
5
6    if (!token) {
7      throw new UnauthorizedException('NOT HAVE A TOKEN');
8    }
9
10   try {
11     const tokenPayload = await this.jwtService.verifyAsync(token);
12
13     request.user = {
14       userId: tokenPayload.sub,
15       email: tokenPayload.email,
16     };
17
18     return true;
19   } catch (error) {
20     throw new UnauthorizedException('VERIFY ERROR');
21   }
22 }

```

Рис. 2.12 – Валідація JWT-токена

У системі (рис. 2.13) використовуються спеціальні структури для того, щоб впорядкувати й зробити більш зручним обмін даними між клієнтом та сервером. Це дозволяє чітко визначити, яка інформація надсилається або запитується, і забезпечує коректну обробку на всіх етапах.



```

1  export interface BodyHandleData {
2    userId: number;
3    systemId: number;
4    currentPower?: number;
5    gridPower?: number;
6    percentage?: number;
7  };
8
9  export interface BodyFluxQueryRealTime {
10   typeProject: string;
11 };
12

```

Рис. 2.13 – Інтерфейси

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						37
Змн.	Арк.	№ докум.	Підпис	Дата		

Висновок до другого розділу

У цьому розділі було детально розглянуто, як саме побудований і працює мобільний додаток для моніторингу сонячної енергетичної системи та акумуляторної батареї. Було окреслено ключові сценарії використання, які покривають увесь цикл взаємодії користувача з системою – від реєстрації до перегляду прогнозів.

Архітектура додатку реалізована на основі перевірених підходів, таких як контролери, сервіси та репозиторії. База даних побудована на двох рівнях: класична реляційна та InfluxDB – для роботи з телеметричними даними в реальному часі.

Окрему увагу приділено реальній роботі додатку: реалізовано алгоритми обміну даними, авторизації, збереження показників та оновлення інтерфейсу без затримок. Передбачено також механізми автоматичного повідомлення про помилки.

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						38
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ДОДАТКОМ

3.1 Порядок встановлення та налаштування параметрів системи

Розгортання та конфігурація системи передбачає поетапне налаштування кожного її компонента окремо, а також узгодження роботи усіх частин у єдиному цілісному комплексі.

Для наочного представлення архітектури програмного комплексу та процесу його розгортання була підготовлена діаграма розгортання (рис. 3.1).

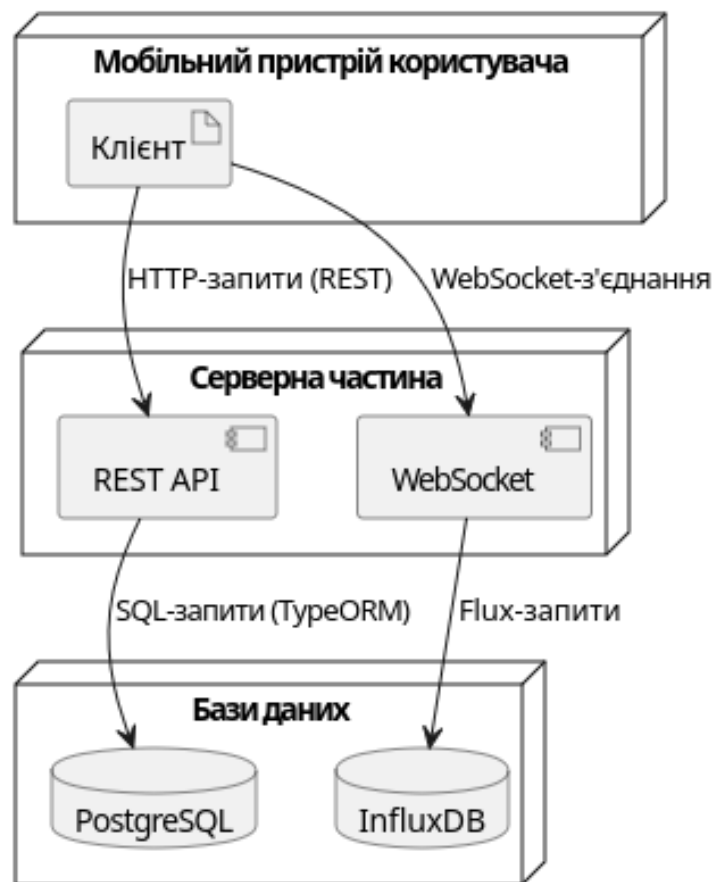


Рис. 3.1 – Діаграма розгортання додатку

Діаграма показує основні компоненти системи та їх взаємодію під час роботи мобільного додатку моніторингу енергосистем. Клієнтський додаток на мобільному пристрої спілкується із сервером через HTTP-запити для стандартних операцій та через WebSocket для отримання даних у реальному часі. Сервер складається з REST API і WebSocket, які працюють із двома базами даних: PostgreSQL для збереження інформації про користувачів і системи, а також InfluxDB – для зберігання часових даних моніторингу.

3.2 Структура інтерфейсу

Користувач взаємодіє із системою через мобільний додаток. Після запуску додатку на екрані відразу відображається сторінка реєстрації (рис. 3.2).

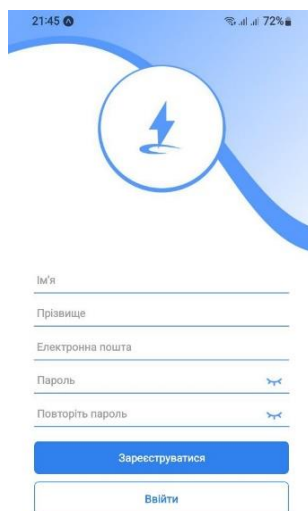


Рис. 3.2 – Сторінка реєстрації

Користувач також має можливість увійти у свій існуючий акаунт, використовуючи електронну пошту та пароль, якщо він вже зареєстрований в системі (рис. 3.3).

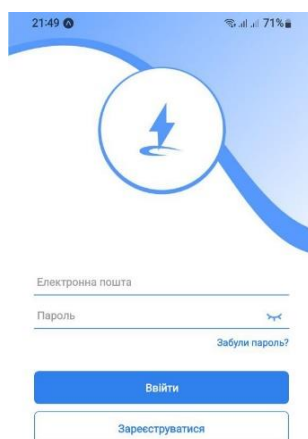


Рис. 3.3 – Сторінка авторизації

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						40
Змн.	Арк.	№ докум.	Підпис	Дата		

Після успішної реєстрації користувач потрапляє на головний екран додатку, де відображається основна інформація та доступні функції для подальшої роботи з системою (рис. 3.4).



Рис. 3.4 – Головна сторінка

Головний екран показує додані енергосистеми або повідомляє про їх відсутність. Знизу є панель навігації з кнопками для переходу на головний екран, профіль і додавання системи через сканер QR-коду. Верхня плаваюча кнопка також відкриває екран додавання системи.

На профільній сторінці користувач бачить своє ім'я та аватар, а також посилання для налаштування профілю. Нижче відображається список його енергосистем. Якщо системи ще не додані, відображається повідомлення про необхідність додати систему (рис. 3.5).



Рис. 3.5 – Профільна сторінка

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		41

На сторінці налаштувань профілю (рис. 3.6) користувач має змогу оновити свої особисті дані, такі як ім'я, прізвище та електронна адреса. Тут також передбачена можливість змінити пароль для підвищення безпеки акаунта. Крім того, користувач може швидко вийти зі свого облікового запису або, за потреби, повністю видалити акаунт із системи.

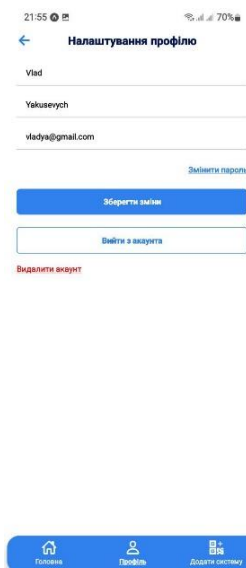


Рис. 3.6 – Сторінка налаштування

Сторінка зміни пароля (рис. 3.7) дозволяє користувачу ввести старий пароль, а також новий пароль і підтвердити його повторним введенням. Для зручності і безпеки поля мають можливість приховувати введені символи. Після заповнення необхідних полів користувач може натиснути кнопку «Змінити пароль» для збереження оновлених даних.



Рис. 3.7 – Сторінка зміни пароля

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						42
Змн.	Арк.	№ докум.	Підпис	Дата		

При переході до додавання системи (рис. 3.8) користувач потрапляє на екран сканування QR-коду. Тут активується камера пристрою, і користувачу пропонується помістити QR-код у рамку для автоматичного зчитування. Інтерфейс мінімалістичний і зручний, щоб забезпечити швидкий та точний процес сканування.

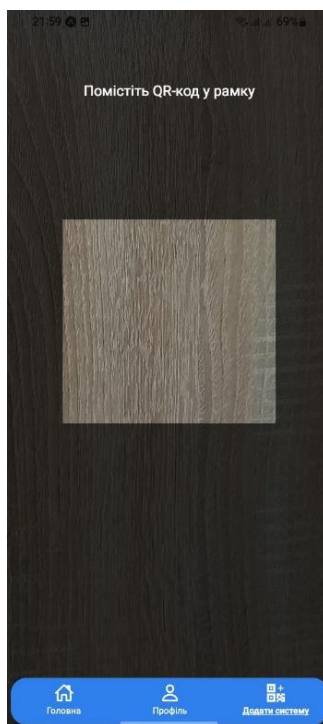


Рис. 3.8 – Екран сканування Qr-код

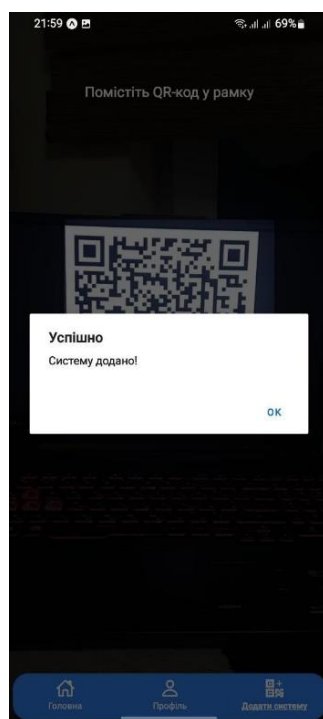


Рис. 3.9 – Успішне сканування Qr-коду

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						43
Змн.	Арк.	№ докум.	Підпис	Дата		

Після успішного сканування QR-коду (рис. 3.9) користувач переходить на екран додавання енергосистеми (рис. 3.10), де він може вказати назву для нової системи. Тут доступні кнопки для підтвердження додавання або відміни операції.

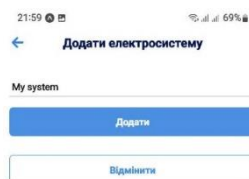


Рис. 3.10 – Додавання енергосистеми

Після додавання енергосистеми користувач бачить повідомлення про успішне завершення операції (рис. 3.11). Після підтвердження повідомлення система автоматично оновлює список, і на головному екрані з'являються картки з інформацією про додані пристрої, такі як сонячна панель та акумулятор, із відображенням їх поточних параметрів – потужності, заряду акумулятора та потужності з мережі (рис. 3.12).

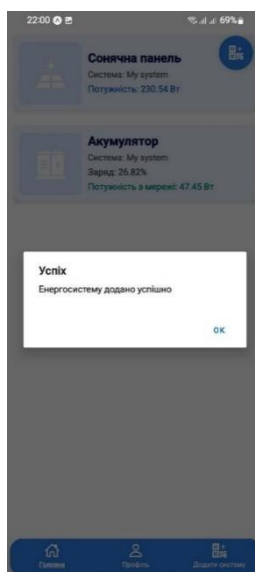


Рис. 3.11 – Успішне повідомлення додавання системи

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		44

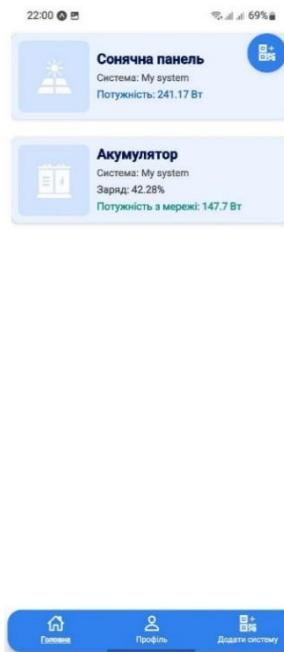


Рис. 3.12 – Головна сторінка зі системами та показниками у реальному часі

На головному екрані користувача представлений список доданих енергосистем із основною інформацією про кожну з них (рис. 3.13). Кожна система відображається у вигляді окремої картки. Користувач може взаємодіяти з цими картками, наприклад, видаляти непотрібні системи через контекстне меню, що викликається натисканням на відповідний значок (рис. 3.14). Після видалення система підтверджує успішне виконання операції відповідним повідомленням (рис. 3.15).

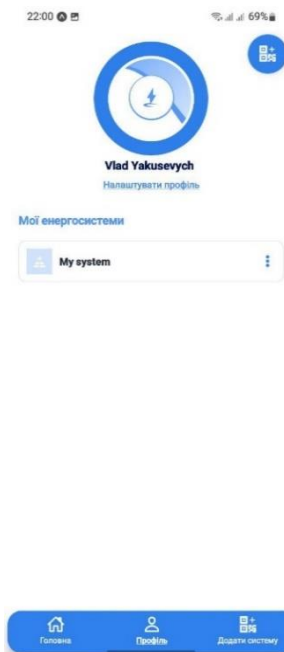


Рис. 3.13 – Головна сторінка з системами

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		45

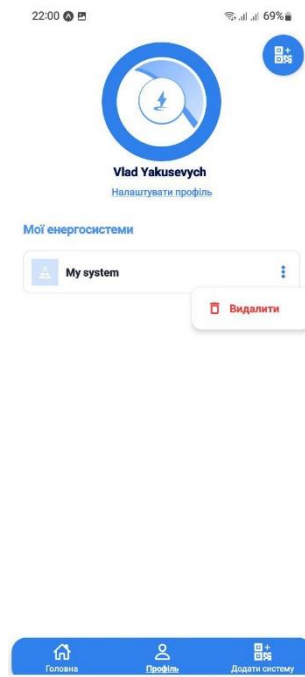


Рис. 3.14 – Головна сторінка з видаленням системи

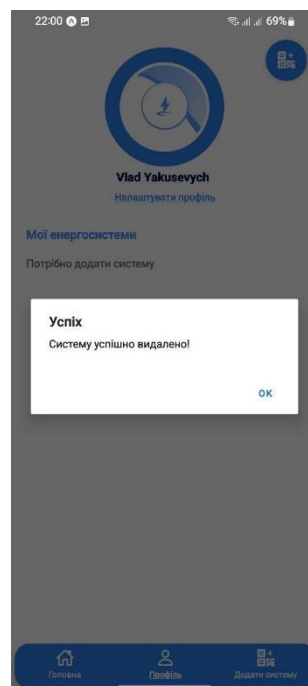


Рис. 3.15 – Успішне підтвердження видаленої системи

На екрані показано спливаюче вікно (рис. 3.16), що відображає детальну інформацію про роботу сонячної панелі. У верхній частині вказано тип пристрою – сонячна панель. Центральний елемент інтерфейсу – графік, який демонструє зміну потужності за декілька останніх вимірювань: по вертикальній осі подано значення потужності у ватах, а по горизонтальній – порядкові номери вимірювань. Під графіком окремо виділено поточне значення потужності великим синім

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						46
Змн.	Арк.	№ докум.	Підпис	Дата		

шрифтом для зручності сприйняття. Внизу розташована кнопка для закриття вікна, що дозволяє швидко повернутися до основного інтерфейсу.

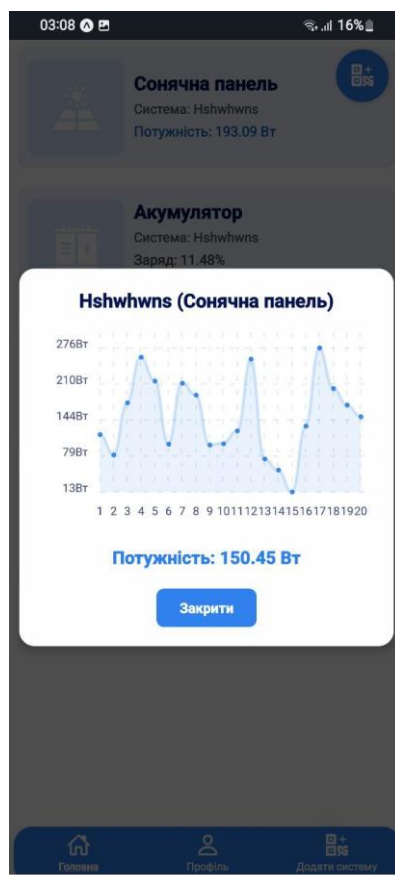


Рис. 3.16 – Графік відображення останніх показників «Сонячної панелі»

3.3 Тестування роботи програмного комплексу

З метою перевірки коректності функціонування розробленого застосунку було проведено низку базових тестувань, спрямованих на охоплення основних сценаріїв його використання. Зокрема, було здійснено тестування процесу реєстрації нового користувача, під час якого перевірялась здатність системи обробляти валідні вхідні дані. Користувач вводив коректну електронну адресу, заповнював усі необхідні поля форми та ініціював реєстрацію. На цьому етапі система здійснювала перевірку введених даних на відповідність встановленим вимогам перед їх передачею на сервер. За результатами тесту підтверджено, що процедура реєстрації відбувається без помилок, а користувач успішно авторизується та автоматично перенаправляється на головний екран застосунку.

Було проведено тестування функціоналу видалення енергосистеми, зосереджене на перевірці здатності користувача ініціювати та завершити процес

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						47
Змн.	Арк.	№ докум.	Підпис	Дата		

видалення елемента зі списку на головному екрані застосунку. У рамках перевірки користувач обирав одну з доступних систем та підтверджував її видалення через контекстне меню інтерфейсу. Система обробляла запит, після чого відображався оновлений список без зазначеного елемента. Результати тестування підтвердили, що обрана енергосистема успішно вилучається з переліку, а головний екран автоматично оновлюється відповідно до змін, забезпечуючи коректне відображення актуального стану.

У процесі тестування функціоналу сканування QR-коду було перевірено здатність застосунку обробляти дані, закодовані в графічному форматі, з метою подальшого додавання нової енергосистеми. Користувач, перебуваючи на відповідному екрані, ініціював сканування за допомогою камери мобільного пристрою. Система мала зчитати QR-код, розпізнати зашифровану в ньому інформацію та автоматично перейти до етапу заповнення форми додавання. Під час тестування виявлено, що у більшості випадків процедура проходить успішно – код зчитується, дані обробляються, а користувач безперешкодно переходить до наступного етапу. Водночас зафіксовані окремі випадки, коли система не змогла коректно розпізнати певні коди або відображала повідомлення про помилку, що свідчить про необхідність додаткової обробки виняткових ситуацій.

У межах тестування функціональності зміни пароля було перевірено здатність системи забезпечувати безпечне та коректне оновлення облікових даних користувача через інтерфейс налаштувань профілю. Користувач вводив поточний пароль, новий пароль і підтверджував його, після чого ініціював зміну. Система здійснювала валідацію введених значень, перевіряючи відповідність вимогам безпеки. У результаті тестування встановлено, що процес оновлення пароля виконується стабільно, без виникнення помилок, а інтерфейс програми надає повідомлення про успішне завершення операції, що підтверджує правильність реалованої логіки.

Тестування режиму моніторингу енергосистем у реальному часі мало на меті оцінити стабільність та своєчасність оновлення критичних показників, таких як заряд акумулятора, потужність сонячної панелі та рівень споживання енергії.

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						48
Змн.	Арк.	№ докум.	Підпис	Дата		

Користувач, перебуваючи на головному екрані застосунку, мав змогу спостерігати за динамікою змін цих параметрів, які повинні оновлюватись автоматично з інтервалом в одну секунду. За результатами тесту встановлено, що в більшості випадків система коректно відображає поточний стан енергосистеми, однак періодично спостерігались затримки оновлення або повна його відсутність. Це призводило до відображення застарілих даних і, відповідно, ускладнювало оперативне реагування на зміни в роботі системи, що свідчить про потребу в удосконаленні механізмів синхронізації даних у реальному часі.

Висновки до третього розділу

У третьому розділі було детально розглянуто, як користувач взаємодіє з додатком, як виглядає інтерфейс і що відбувається під час встановлення та роботи системи. Особливу увагу приділено зручності користування – від моменту першого запуску до управління енергосистемами й перегляду телеметрії. Уся структура додатку побудована так, щоб навіть новий користувач інтуїтивно розумів, як додати систему, змінити налаштування або переглянути дані в реальному часі.

Було також протестовано ключові сценарії використання – реєстрацію, авторизацію, сканування QR-коду, додавання і видалення енергосистем, моніторинг у реальному часі. Тестування підтвердило працездатність основного функціоналу, хоча в деяких випадках було зафіксовано незначні затримки при оновленні даних, що потребує подальшої оптимізації. Загалом, додаток продемонстрував стабільну роботу, зрозумілу навігацію та позитивний користувацький досвід, що є важливим кроком на шляху до впровадження системи в реальні умови.

ВИСНОВКИ

У межах цієї роботи було створено повноцінну систему для моніторингу сонячною енергетичною установкою, яка поєднує зручний мобільний додаток, надійний серверний функціонал та ефективну обробку даних у режимі реального часу. Враховуючи актуальність використання відновлюваних джерел енергії, особливо в умовах нестабільного електропостачання, розроблена система має не лише практичну цінність, а й соціальну значущість.

На етапі аналізу було вивчено потреби майбутніх користувачів, розглянуто аналогічні програмні продукти й визначено оптимальний підхід до побудови архітектури. Створена система має чітку структуру, де кожен компонент виконує свою роль – сервер відповідає за логіку, база даних зберігає структуровану інформацію, а InfluxDB забезпечує швидкий доступ до телеметрії. Усе це поєднується в мобільному додатку, який дозволяє у зручному форматі бачити стан обладнання, контролювати ефективність, додавати нові пристрої та своєчасно реагувати на проблеми.

Інтерфейс продуманий таким чином, щоб ним було легко користуватись навіть людям без технічного досвіду. Функції авторизації, управління профілем, сканування QR-кодів, перегляд історії та моніторинг у реальному часі працюють як єдиний цілісний механізм. Проведене тестування підтвердило, що додаток справляється з основними завданнями, хоча деякі аспекти можуть бути вдосконалені в подальших версіях.

У результаті реалізовано не просто технічне рішення, а інструмент, який дійсно може допомогти людям ефективно використовувати енергію, економити ресурси та робити свій внесок у розвиток сталої енергетики. Це демонструє, як сучасні цифрові технології можуть працювати на користь екології, економіки й комфорту користувачів.

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						50
Змн.	Арк.	№ докум.	Підпис	Дата		

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Smart MAIC. Demo. URL: <https://dash.smart-maic.com/demo#ath>.
2. SolarEdge. Головна сторінка. URL: <https://www.solaredge.com/en/products/software-tools/mysolaredge>.
3. PV Forecast: Solar Power & Gen. Google Play. Додатки. URL: <https://play.google.com/store/apps/details?id=com.exigo.solcaster>.
4. Zell L. Як правильно працювати з REST API. ITVDN. REST API. 15.05.2015. URL: <https://itvdn.com/ua/blog/article/rest-api-18>.
5. Valentyn V. Повний огляд REST: нюанси, поради, приклади. DOU. Форум. 19.09.2024. URL: <https://dou.ua/forums/topic/50364/>.
6. The WebSocket API (WebSockets). MDN. Web APIs. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API.
7. MQTT. NestJS. Microservices. URL: <https://docs.nestjs.com/microservices/mqtt>.
8. Як забезпечити безпеку REST API. COREWIN. Блог. URL: <https://corewin.ua/blog/rest-api-web-service-security/>.
9. Що таке контейнеризація та в чому її переваги. GigaCloud. Статті. URL: <https://gigacloud.ua/articles/shho-take-kontejneryzacziya-ta-v-chomu-yiyi-perevagy/>.
10. Gomes G. Services, Controllers and Repositories. An Introduction. Medium. 24.07.2024. URL: <https://gabrielgomes61320.medium.com/services-controllers-and-repositories-an-introduction-241ac52cdd93>.
11. Collings T. Controller-Service-Repository. Medium. 10.08.2021. URL: <https://tom-collings.medium.com/controller-service-repository-16e29a4684e5>.
12. Aibin M., Baeldung. The DTO Pattern (Data Transfer Object). Baeldung. Architecture. 25.03.2025. URL: <https://www.baeldung.com/java-dto-pattern>.
13. Михайло П. VS Code (Visual Studio Code): налаштування, розширення, комбінації клавіш. Petrov. Софт. 10.02.2025. URL: <https://petrov.net.ua/visual-studio-code-settings-extensions-hotkeys/>.

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						51
Змн.	Арк.	№ докум.	Підпис	Дат		

- 14.Гайд по Docker: концепція, устрій та принцип роботи. SpaceLab. URL: <https://spacelab.ua/articles/Docker/>.
- 15.@Admin. Використання Postman в тестуванні. QATestLab. Технічні статті. 02.12.2021. URL: <https://training.qatestlab.com/blog/technical-articles/use-postman-in-testing/>.
- 16.Розробка мобільних додатків Android за допомогою React Native. WhileWeb. Блог. 12.07.2023. URL: <https://whileweb.com/uk/blog/rozrobka-mobilnih-dodatkov-android-za-dopomogoyu-react-native/>.
- 17.CSS: Cascading Style Sheets. MDN Web Docs. References. URL: <https://developer.mozilla.org/en-US/docs/Web/CSS>.
- 18.Introduction. MDN. References. URL: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Introduction>.
- 19.TypeScript for JavaScript Programmers. TypeScript. Get Started. URL: <https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html>.
- 20.Що таке Node JS простими словами. DAN.IT. Блог. 16.10.2024. URL: <https://dan-it.com.ua/uk/blog/chto-jeto-takoe-node-js-prostymi-slovami/>.
- 21.NestJS. Docs. URL: <https://docs.nestjs.com/>.
- 22.Introduction. Socket.IO. Documentation. URL: <https://socket.io/docs/v4/>.
- 23.Getting Started. TypeORM. Docs. URL: <https://typeorm.io/docs/getting-started>.
- 24.What Is PostgreSQL?. PostgreSQL. Documentation. URL: <https://www.postgresql.org/docs/current/intro-what-is.html>.
- 25.Get started with InfluxDB. InfluxData. Documentation. URL: <https://docs.influxdata.com/influxdb/v2/get-started/>.
- 26.Anonymous. QRSAY: долаючи прірву. Адаптуємо технологію QR-кодів. DOU. Розробка. 02.04.2021. URL: <https://dou.ua/forums/topic/33157/>.
- 27.Що таке MVC? Переваги та недоліки MVC. AlexHost. 06.12.2024. URL: <https://alexhost.com/uk/faq/shho-take-mvc-perevagy-ta-nedoliky-mvc/>.
- 28.Beaulieu J. F. The Beginner's Guide to Docker. Medium. 28.01.2024. URL: <https://medium.com/@JeffyJeff/the-beginners-guide-to-docker-fa4c4d3181e7>.

					ІПЗ.КР.Б – 121 – 25 – ІІЗ	Арк.
						52
Змн.	Арк.	№ докум.	Підпис	Дат		

29. Marynych I. Порівняння React Native CLI та Expo: що обрати для вашого проєкту. DOU. Різне. 02.02.2025. URL: <https://dou.ua/forums/topic/52308/>.

30. Tkachuk I. Типи баз даних: особливості, відмінності та приклади. DOU. Статті. 13.01.2023. URL: <https://dou.ua/lenta/articles/types-of-databases/>.

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дат		53

ДОДАТКИ

					ІІЗ.КР.Б – 121 – 25 – ІІЗ	Арк.
						54
Змн.	Арк.	№ докум.	Підпис	Дат		

HomeScreen.tsx:

```
import { useEffect, useRef, useState } from "react";
import { FlatList, Image, StyleSheet, Text, TouchableOpacity, View } from "react-native";
import { MaterialCommunityIcons } from '@expo/vector-icons';
import { useNavigation } from "@react-navigation/native";
import type { NativeStackNavigationProp } from "@react-navigation/native-stack";
import { RootStackParamList } from "../types";
import AsyncStorage from "@react-native-async-storage/async-storage";
import { io, Socket } from "socket.io-client";
import EventBus from "../utils/EventBus";
```

```
type NavigationProp = NativeStackNavigationProp<RootStackParamList, 'Home'>;
```

```
interface EnergySystem {
  id: string;
  systemId: number;
  type: 'Сонячна панель' | 'Акумулятор';
  battery: number;
  saved: number;
  spent: number;
  status: 'active' | 'inactive';
  systemName: string;
  currentPower?: number;
  gridPower?: number;
}
```

```
const imageMap: { [key: string]: any } = {
  'Сонячна панель': require('../assets/home/solar_panel.png'),
  'Акумулятор': require('../assets/home/accumulator.png'),
};
```

```
const EnergyCard = ({ system }: { system: EnergySystem }) => {
  const imageSource = imageMap[system.type];
  return (
    <View style={styles.card}>
      <Image source={imageSource} style={styles.image} />
      <View style={styles.cardText}>
        <Text style={styles.title}>{system.type}</Text>
        <Text style={styles.systemName}>Система: {system.systemName}</Text>
        {system.type === 'Сонячна панель' && system.currentPower !== undefined && (
```

```

    <Text style={styles.power}>Потужність: {system.currentPower} Вт</Text>
  )}
  {system.type === 'Акумулятор' && (
    <>
      <Text style={styles.batteryText}>Заряд: {system.battery}%</Text>
      {system.gridPower !== undefined && (
        <Text style={styles.gridPower}>Потужність з мережі: {system.gridPower} Вт</Text>
      )}
    </>
  )}
</View>
</View>
);
};

```

```

export default function HomeScreen() {
  const [systems, setSystems] = useState<EnergySystem[]>([]);
  const navigation = useNavigation<NavigationProp>();
  const socket = useRef<Socket | null>(null);
  const intervals = useRef<NodeJS.Timeout[]>([]);

  useEffect(() => {
    socket.current = io("http://192.168.50.107:3000");

    return () => {
      socket.current?.disconnect();
    };
  }, []);

  const fetchSystems = async () => {
    try {
      const userId = await AsyncStorage.getItem('userId');
      if (!userId) return;

      const response = await fetch(`http://192.168.50.107:3000/users/${userId}`);
      const data = await response.json();

      const formattedSystems: EnergySystem[] = [];

      for (const system of data.systems) {
        const name = system.name || 'Без назви';
        const base = {

```

```

    battery: 0,
    saved: 0,
    spent: 0,
    status: 'active' as const,
    systemName: name,
    systemId: system.id,
  };

  formattedSystems.push({ ...base, id: `solar-${system.id}`, type: 'Сонячна панель' });
  formattedSystems.push({ ...base, id: `accumulator-${system.id}`, type: 'Акумулятор' });
}

setSystems(formattedSystems);
} catch (error) {
  console.error('Помилка завантаження систем:', error);
}
};

useEffect(() => {
  fetchSystems();
  EventBus.on('systemDeleted', fetchSystems);

  return () => {
    EventBus.off('systemDeleted', fetchSystems);
  };
}, []);

useEffect(() => {
  intervals.current.forEach(clearInterval);
  intervals.current = [];

  const setupIntervals = async () => {
    const userId = await AsyncStorage.getItem('userId');
    if (!userId) return;

    const uniqueSystemIds = Array.from(new Set(systems.map(s => s.systemId)));

    uniqueSystemIds.forEach(systemId => {
      const payload = {
        userId: Number(userId),
        systemId,
        typeProject: { typeProject: 'system' }
      };
    });
  };
  setupIntervals();
}, []);

```

```

};

const interval = setInterval(() => {
  socket.current?.emit('queryRealDataTime', payload, (response: any) => {
    if (typeof response === 'object' && response !== null) {
      setSystems(prev =>
        prev.map(s =>
          s.systemId === systemId
            ? {
                ...s,
                battery: response.battery_charge ?? s.battery,
                currentPower: response.current_power ?? s.currentPower,
                gridPower: response.grid_power ?? s.gridPower,
              }
            : s
          )
        );
    }
  });
}, 1000);

intervals.current.push(interval);
});
};

if (systems.length > 0) {
  setupIntervals();
}

return () => {
  intervals.current.forEach(clearInterval);
  intervals.current = [];
};
}, [systems]);

const handlePress = () => {
  navigation.navigate('AddSystem', { qrData: null });
};

return (
  <View style={styles.container}>
    <TouchableOpacity style={styles.circle} onPress={handlePress} activeOpacity={0.7}>

```

```

        <MaterialCommunityIcons name="qrcode-plus" size={24} color="#fff" />
    </TouchableOpacity>

    {systems.length > 0 ? (
      <FlatList
        data={systems}
        keyExtractor={(item) => item.id}
        renderItem={({ item }) => <EnergyCard system={item} />}
        contentContainerStyle={{ paddingBottom: 40 }}
      />
    ) : (
      <View style={styles.noSystemsContainer}>
        <Text style={styles.noSystemsText}>Потрібно додати систему</Text>
      </View>
    )}
  </View>
);
}

```

```

const styles = StyleSheet.create({
  container: {
    flex: 1,
    paddingHorizontal: 10,
    paddingTop: 20,
    paddingBottom: 40,
    backgroundColor: '#fff'
  },
  circle: {
    position: 'absolute',
    top: 40,
    right: 16,
    width: 55,
    height: 55,
    borderRadius: 30,
    backgroundColor: '#2f80ed',
    justifyContent: 'center',
    alignItems: 'center',
    elevation: 5,
    zIndex: 2,
  },
  card: {
    flexDirection: 'row',

```

```
    backgroundColor: '#eaf3ff',
    padding: 10,
    marginVertical: 20,
    marginBottom: 5,
    borderRadius: 12,
    elevation: 4,
    alignItems: 'center'
  },
  cardText: {
    flex: 1,
    paddingHorizontal: 10
  },
  image: {
    width: 100,
    height: 100,
    borderRadius: 10
  },
  title: {
    color: '#031350',
    fontSize: 22,
    fontWeight: 'bold',
    marginBottom: 4
  },
  systemName: {
    fontSize: 15,
    color: '#555',
    marginBottom: 4
  },
  batteryText: {
    fontSize: 16,
    color: '#444',
    marginBottom: 4
  },
  power: {
    fontSize: 16,
    color: '#1565c0',
    marginBottom: 4
  },
  gridPower: {
    fontSize: 16,
    color: '#00897B',
    marginBottom: 4,
```

```
},
noSystemsContainer: {
  flex: 1,
  justifyContent: 'center',
  alignItems: 'center'
},
noSystemsText: {
  fontSize: 18,
  color: '#555'
},
});
```