

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ
СІКОРСЬКОГО»**

Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці

ДО ЗАХИСТУ ДОПУЩЕНО

В.о. завідувача кафедри

_____ Олександр КОВАЛЬ

«___» _____ 2026 р.

Дипломна робота

на здобуття ступеня бакалавра

**за освітньо-професійною програмою «Інженерія програмного забезпечення
інтелектуальних кібер-фізичних систем в енергетиці»
спеціальності 121 Інженерія програмного забезпечення**

**на тему: «Система моніторингу генерації та споживання електроенергії в
SmartEnergy Lab»**

Виконав:

студент IV курсу, групи ТВ-з21

Монастирний Віктор Ігорович

(прізвище, ім'я, по батькові)

_____ (підпис)

Керівник:

д.т.н., професор кафедри ІПЗЕ

Федорова Наталія Володимирівна

(посада, науковий ступінь, вчене звання, прізвище та ініціали)

_____ (підпис)

Рецензент:

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

_____ (підпис)

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

Студент _____

(підпис)

Київ – 2026

**Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»**

Навчально-науковий інститут атомної та теплової енергетики

Кафедра інженерії програмного забезпечення в енергетиці

Рівень вищої освіти перший (бакалаврський)

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення
інтелектуальних кібер-фізичних систем в енергетиці»

ЗАТВЕРДЖУЮ

В.о. завідувача кафедри

_____ Олександр КОВАЛЬ

«___» _____ 2026р.

ЗАВДАННЯ

на дипломну роботу студенту

_____ Монастирному Віктору Ігоровичу

(прізвище, ім'я, по батькові)

1. Тема роботи: Система моніторингу генерації та споживання електроенергії в SmartEnergy Lab

керівник роботи: Федорова Наталія Володимирівна, д.т.н., професор кафедри ІПЗЕ
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «28» травня 2026 р. № 1993-с

2. Строк подання студентом роботи «10» червня 2026 р.

3. Вихідні дані до роботи: наукова та технічна література, доступне обладнання в лабораторії, середовище розробки WebStorm

4. Зміст (дипломної роботи) пояснювальної записки (перелік завдань, які потрібно розробити): Проаналізувати існуючі подібні smart-системи, визначити компоненти системи, створити функціональну схему системи, створити програмний продукт, провести тестування

5. Перелік ілюстративного матеріалу: функціональна електрична схема системи, Use Case діаграма, діаграма архітектури системи, інтерфейс програмної системи

6. Дата видачі завдання «31» жовтня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Строки виконання етапів роботи	Примітка
1	Отримання завдання	31.10.2025	
2	Дослідження предметної області	01.11.2025 – 31.11.2025	
3	Дослідження існуючих рішень	01.12.2025 – 31.12.2025	
4	Постановка вимог до проєктування системи	01.01.2026 – 31.01.2026	
5	Розробка програмного продукту	01.02.2026 – 03.05.2026	
6	Тестування	04.05.2026 – 10.05.2026	
7	Захист програмного продукту	11.05.2026 – 15.05.2026	
8	Оформлення дипломної роботи	18.05.2026 – 31.05.2026	
9	Передзахист	01.06.2026 – 05.06.2026	
10	Захист	15.06.2026 – 19.06.2026	

Студент

(підпис)Віктор МОНАСТИРНИЙ

(ім'я, ПРІЗВИЩЕ)

Керівник роботи

(підпис)Наталія ФЕДОРОВА

(ім'я, ПРІЗВИЩЕ)

РЕФЕРАТ

Структура та обсяг дипломної роботи. Робота містить 58 сторінок, 32 рисунок, 2 додатки та 1 посилання.

Мета роботи: Розробити сучасну кібер-фізичну систему моніторингу генерації і споживання електроенергії.

Методи дослідження: аналіз, моделювання, абстрагування, емпіричні методи.

Практичне значення одержаних результатів полягає в тому, що система може бути використана є власною розробкою і не прив'язана до архітектури конкретних постачальників екосистем «розумних пристроїв».

Проект впроваджено на кафедральному сервері і він може бути використаний для демонстраційного стенду КПП; проєкт відповідає вимогам Smart Energy.

Ключові слова: *моніторинг, датчики, контролери, сонячні панелі, електрика, кіберфізична система.*

ABSTRACT

Structure and scope of the thesis. The work contains 58 pages, 32 figures, 2 appendices and 1 reference.

The purpose of the work: Develop a modern cyber-physical system for monitoring electricity generation and consumption.

The practical significance of the results obtained lies in the fact that the system can be used as its own development and is not tied to the architecture of specific providers of ecosystems of "smart devices".

The project is implemented on the departmental server and it can be used for the demonstration stand of KPI; the project meets the requirements of Smart Energy.

Keywords: monitoring, sensors, controllers, solar panels, electricity, cyber-physical system.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ.....	8
ВСТУП.....	9
1 АНАЛІЗ ПОДІБНИХ ІСНУЮЧИХ СИСТЕМ	13
1.1 Проєкт з відкритим кодом Home Assistant	13
1.2 Компанія Victron Energy	15
1.3 Компанія Shelly.....	16
1.4 Компанія Loxone.....	17
Висновки до розділу 1	18
2 АПАРАТНЕ МОДЕЛЮВАННЯ СИСТЕМИ МОНІТОРИНГУ	20
2.1 Фізичні принципи вимірювання напруги та струму.....	20
2.2 Розрахунок потужності та обліку енергії.....	21
2.3 Апаратні інтерфейси та мережеві протоколи передачі даних	22
2.4 Апаратне забезпечення	23
2.4.1 Контролер ESP32	24
2.4.2 Перетворювач LM2596	24
2.4.3 Датчик напруги та струму INA226	25
2.4.4 Модуль моніторингу електроенергії PZEM-017	26
2.4.5 Модуль-конвертер зв'язку (TTL to RS485)	26
2.4 Побудова функціональної електричної схеми	27
Висновки до розділу 2.....	30
3 ЗАСОБИ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ	31
3.1 Платформа і стек програмного забезпечення.....	31
3.2 Мови програмування.....	32
3.3.1 Typescript	32
3.3.2 INO (мова Arduino).....	33
3.3.3 JSON (JavaScript Object Notation)	33
3.4 Серверна частина (Backend).....	33
3.5 База даних InfluxDB	34
3.6 MQTT-брокер.....	35
3.7 Клієнтська частина (Frontend).....	36
3.8 Контейнеризатор Docker.....	37

Висновки до розділу 3.....	37
4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ.....	38
4.1 Архітектура програмної системи.....	38
4.2 Апаратне програмне забезпечення	39
4.3 Реалізація серверної частини (Backend) та обробка поточкових даних	42
4.4 Розробка клієнтського інтерфейсу користувача (Frontend)	44
4.4.1 Головна панель	45
4.4.2 Панелі аналітики.....	47
4.4.3 Панель порівняння параметрів.....	48
4.5 Тестування розробленої системи	49
4.5.1 Поведінка системи в різних ситуаціях	49
4.5.2 Додавання історичних даних.....	51
Висновки до розділу 4.....	53
5 РОБОТА КОРИСТУВАЧА З ПРОГРАМНИМ ПРОДУКТОМ.....	54
5.1 Порядок розгортання та запуску програмного комплексу	54
5.2 Експлуатація головної панелі.....	55
5.2 Експлуатація панелі порівняння	56
Висновки до розділу 5.....	57
ВИСНОВКИ.....	58
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	59
ДОДАТОК А Лістинг розробленої системи.....	61
ДОДАТОК Б Презентація.....	75

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

BMS	Battery Management System, система контролю батареєю
MPPT- контролер	Maximum Power Point Tracking, контролер відстеження максимальної точки потужності
RS	Recommended Standard, стандарт для передачі даних (напр. RS-485)
UART	Universal Asynchronous Receiver-Transmitter, протокол передачі даних
СУБД	Система управління базами даних
ШІМ	широотно-імпульсна модуляція

ВСТУП

Сучасний етап розвитку енергетики характеризується активною інтеграцією відновлюваних джерел енергії (ВДЕ), зокрема сонячних електростанцій (СЕС), у локальні та глобальні енергетичні мережі. Проте генерація з таких джерел є нестабільною та залежною від зовнішніх чинників (метеорологічних умов, часу доби). [1, 4] Для контролю виробництва та споживання електроенергії, а також для оптимізації роботи акумуляторних батарей, виникає гостра потреба у високоточних цифрових інструментах моніторингу в реальному часі. Лабораторне середовище "SmartEnergy Lab" виконує роль інститутської мережі, що поєднує генерацію від сонячних панелей, накопичувачі енергії (акумуляторні батареї) та різноманітні комунально-побутові чи лабораторні навантаження. Ефективне керування та дослідження процесів у такій системі неможливі без автоматизованого збору даних. [4] Наявність надійного програмно-апаратного комплексу для моніторингу дозволяє не лише накопичувати статистику для подальшого аналізу, а й створювати алгоритми прогнозування та оптимізації енергоспоживання.

Актуальність роботи

У лабораторії КПІ є сонячні панелі, які встановлені на даху будівлі, є mppt-контролер, батарея, інвертор, а також інші кінцеві пристрої, які слугують споживачами. При ясній погоді, сонячні панелі генерують електроенергію, яка по підключеним дротам надходить до mppt-контролера. Контролер всередині коригує вихідні струм і напругу, налаштовуючи їх так, щоб ефективно заряджати підключену батарею. Енергія з батареї передається на інвертора, до якого підключені потім інші побутові пристрої. Хоча система працює справно, однак вона досить проста і не має можливості моніторингу. Тобто користувач системи не має змоги бачити, що відбувається в системі. [5] Користувач не може визначити скільки енергії і коли згенеровано, де і як вона витратилася; не бачить рівень заряду батареї; не може проводити аналітику, наприклад порівнюючи зима – літо, ранок – вечір. Неможливо оцінити вигоду і ефективність системи.

Мета

Розробити сучасну кібер-фізичну систему моніторингу генерації і споживання електроенергії.

Об'єкт дослідження

Процес модифікації наявної системи генерації до рівня сучасної smart-системи.

Предмет дослідження

Методика створення розумної системи моніторингу за допомогою датчиків, мікроконтролерів, програмування та електротехніки.

Методи дослідження

Для створення сучасної системи моніторингу необхідно провести комплексне дослідження тем моніторингу та обробки даних. Під час виконання роботи було використано наступні методи досліджень:

- Аналіз: прочитані і переглянуті наукові статті, публікації, технічні документації, відкриті репозиторії з кодом
- Моделювання: створення прототипів системи, датчиків, контролерів, рядів даних для створення програми і перевірки її працездатності
- Абстрагування: система моніторингу концентрується лише на основних параметрах електричної системи; несуттєві фактори і параметри системи, що не мають вплив на моніторинг, були відкинуті
- Емпіричні методи: моніторингова система на практиці вимірює і оброблює показники, такі як: напруга, сила струму, потужність, заряд батареї; ці показники потім можна подивитися на графіках у різному часі і порівняти між собою

Успішне застосування всіх цих методів — ключова основа сучасних моніторингових систем.

Завдання

Для успішного виконання роботи було поставлено наступні завдання:

- Проаналізувати існуючі подібні smart-системи
- Визначити оптимальну номенклатуру, необхідну кількість та технічні характеристики сенсорів з огляду на їхню вартість, доступність і доцільність.
- Обґрунтувати способи фізичного підключення датчиків, а також вибір комунікаційних інтерфейсів та протоколів передачі даних у системі.
- Спроекувати та реалізувати архітектуру програмного забезпечення для збору, первинної обробки й довготривалого збереження часових рядів даних із можливістю оперативного доступу до них
- Розробити інтуїтивно зрозумілий, адаптивний та сучасний веб-інтерфейс

Наукова новизна

Існують різні системи моніторингу сонячної генерації електроенергії. Абсолютна більшість з них – розробки і програмні продукти компаній, які займаються продажем і встановленням сонячних панелей та їх компонентів, або систем розумного будинку. Це комерційні рішення, які створені спеціально під прилади конкретної компанії, не мають відкритого програмного коду і не надають можливості вносити зміни чи додавати сторонні пристрої від інших виробників. Тобто такі рішення накладають сильні обмеження екосистеми.

Мій моніторинг буде спрямований на відкритість та результативність: не буде штучних апаратних чи програмних обмежень, система прийматиме дані з будь-яких пристроїв (датчиків).

Практичне значення

Результат моєї роботи можна використати як повноцінну незалежну систему моніторингу, так і в рамках демонстраційного стенду, або в поєднанні з іншими роботами напрямку Smart Energy.

Розроблена система моніторингу зможе бути використана для будь-яких генераторів і кінцевих пристроїв. Однак для цього може знадобитися більше датчиків.

1 АНАЛІЗ ПОДІБНИХ ІСНУЮЧИХ СИСТЕМ

Системи моніторингу активно вбудовуються на різних підприємствах та приватними користувачами. Особливо популярність їм додають нові електромобілі, які потребують електроенергію. Багато виробників сонячних панелей рекламують свій продукт саме в контексті зарядних станцій для електричних автомобілей та значних коштах, які люди можуть на цьому зекономити, не купуючи паливе. Такі панелі часто вбудовуються в систему «розумний дім» або інші комерційні системи моніторингу. Є також системи, які дозволяють передати надлишок енергії в загальну систему електромережі і заробити гроші. Зазвичай це робиться системою автоматично; часто такі системи надають свій додаток для управління процесами в будинку.

В цьому розділі будуть наведені найвідоміші системи, які дотичні до теми моєї роботи.

1.1 Проєкт з відкритим кодом Home Assistant

Home Assistant — це безкоштовна платформа з відкритим вихідним кодом для автоматизації «розумного будинку», яка виступає в ролі центральної системи керування та локального хабу для IoT-пристроїв. Основний фокус платформи спрямований на локальний контроль, конфіденційність даних користувача та максимальну незалежність від сторонніх хмарних сервісів. Головне меню демонстраційного варіанту панелі керування Home Assistant зображено на рисунку 1.1.

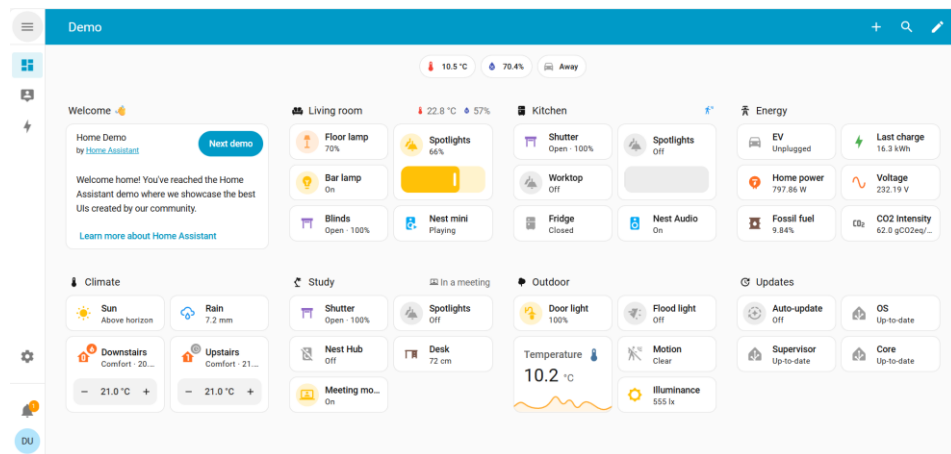


Рисунок 1.1 – Демо-вигляд віджетів Home Assistant

Open Source проект для налаштування системи «розумного будинку» власноруч. Надає власний програмний інтерфейс з великою кількістю цікавих віджетів.

У контексті систем моніторингу енергетики Home Assistant має такі ключові характеристики:

- **Універсальна інтеграція:** Платформа підтримує тисячі різноманітних пристроїв та протоколів (зокрема, MQTT, Wi-Fi, Bluetooth). Вона дозволяє легко об'єднувати в одну мережу вимірювальні датчики, мікроконтролери, розумні розетки та інвертори сонячних електростанцій.
- **Вбудована панель Energy Dashboard:** Спеціалізований інструмент для моніторингу енергетичних потоків у реальному часі. Панель дозволяє наочно візуалізувати обсяги генерації сонячних панелей, споживання електроенергії будинком або лабораторією, а також процеси заряду й розряду акумуляторних батарей (як-от відстеження стану SoC).
- **Автоматизація:** платформа дозволяє створювати складні логічні ланцюжки керування навантаженням. Наприклад, автоматично вмикати певні лабораторні прилади або системи підігріву у періоди пікової генерації сонячної енергії або за умови досягнення акумулятором заданого рівня заряду.
- **Кросплатформеність та адаптивність:** Інтерфейс користувача побудований на сучасних вебтехнологіях, є повністю адаптивним під будь-які

екрани (від смартфонів до великих настільних моніторів) і швидко реагує на події в мережі.

1.2 Компанія Victron Energy

Victron Energy – це провідний світовий виробник високоякісного обладнання для автономних, гібридних та резервних систем енергопостачання. Продукція компанії широко використовується в комерційному автотранспорті, а також у стаціонарних сонячних електростанціях та локальних мікромережах. Приклади їх приладів зображені на рисунку 1.2. Окрім силового обладнання (інверторів, зарядних пристроїв), компанія відома своєю потужною екосистемою цифрового моніторингу та керування енергетичними потоками.



Рисунок 1.2 – Основна продукція Victron Energy

У контексті систем моніторингу та побудови інтелектуальних кібер-фізичних систем Victron Energy має такі ключові особливості:

- **Екосистема GX-пристроїв:** Виступають у ролі локального «мозку» та комунікаційного хабу системи. Ці пристрої збирають інформацію з усіх

підключених компонентів (MPPT-контролерів, інверторів, шунтів батарей), обробляють її в реальному часі та передають на сервери або локальні дашборди.

- Підтримка протоколів: Обладнання Victron використовує відкритий протокол Modbus TCP, що дозволяє легко інтегрувати його із зовнішніми системами автоматизації, промисловими контролерами або стороннім програмним забезпеченням. Також підтримуються легковагі протоколи, такі як MQTT, для передачі даних у хмарні чи локальні бази даних часових рядів.

- Хмарне сховище: безкоштовна хмарна платформа для віддаленого моніторингу та аналітики. Дає детальну візуалізацію процесів генерації, споживання та накопичення енергії у вигляді інтерактивних графіків та схем, дозволяючи відстежувати ефективність системи, стан акумуляторів (SoC) та аналізувати історичні дані за будь-який період.

- Операційна система на базі Linux: можна модифікувати, додавати власні скрипти (наприклад, на Python), створювати кастомні інтерфейси або інтегрувати обладнання з платформами розумного будинку (наприклад Home Assistant або Node-RED).

- Проведення вимірів: використання спеціалізовані шунти (SmartShunt) та системи BMS, які фіксують напругу, силу струму, активну потужність та дозволяють вираховувати точний залишок ємності акумуляторів, мінімізуючи похибки та відкидаючи непотрібні дані моніторингу.

1.3 Компанія Shelly

Shelly – це один із найпопулярніших у світі виробників компактних IoT-пристроїв та інтелектуальних рішень для автоматизації житлових і комерційних об'єктів. Продукція компанії відома в Європі завдяки мініатюрності їх пристроїв, високої надійності та унікальної відкритості програмного забезпечення. Приклад інтерфейсу мобільного застосунку для моніторингу від цієї компанії зображено на рисунку 1.3.



Рисунок 1.3 – Мобільний застосунок Shelly

Компанія пропонує встановлення свого контролера безпосередньо в енергощиток квартири/будинку. Їх система може взаємодіяти з розумними пристроями, а також контролювати процес генерації енергії сонячними панелями та зарядку електромобіля.

1.4 Компанія Loxone

Loxone – це австрійська компанія, яка є одним із світових лідерів у сфері професійної автоматизації житлових та комерційних будівель. На відміну від концепції «розумного будинку», що збирається з окремих Wi-Fi-гаджетів, Loxone використовує монолітну архітектуру. Платформа об'єднує всі інженерні системи будівлі (освітлення, опалення, вентиляцію, кондиціонування) в єдину кібер-фізичну екосистему.

технічної складової і програмного коду їх продуктів. Хоча і є відкриті проєкти, такі як Smart Home, все одно вони ставлять розробника у певні рамки своєї архітектури.

З огляду існуючих систем стає зрозуміло, як виглядають сучасні смарт системи і на якій архітектурі вони побудовані.

2 АПАРАТНЕ МОДЕЛЮВАННЯ СИСТЕМИ МОНІТОРИНГУ

Для створення ефективної системи моніторингу сонячної генерації необхідно розуміти фізичні процеси вимірювання електричних величин та особливості взаємодії компонентів у колі постійного струму.

2.1 Фізичні принципи вимірювання напруги та струму

В основі роботи будь-якого цифрового датчика електроенергії лежить перетворення аналогових величин (напруги та струму) у цифровий сигнал, придатний для обробки мікроконтролером.

Вимірювання напруги: здійснюється за принципом паралельного підключення вимірювального вузла до ділянки кола. Для високих рівнів напруги використовуються резистивні дільники напруги, які знижують потенціал до рівнів, безпечних для аналого-цифрового перетворювача (АЦП). Промислові датчики, такі як PZM-017 працюють саме по такому принципу.

Вимірювання струму: найбільш поширеним і надійним методом у системах низької та середньої потужності є метод вимірювання падіння напруги на шунті. Фактично вимірюється знову напруга, а потім ділиться на відомий опір шунта і за законом Ома ми дізнаємося силу струму на ділянці кола, а отже і у самому колі. [6] Такий спосіб також є безпечним: Вольтметр має високий внутрішній опір і фізично не пропустить крізь себе великий струм і не згорить при неправильному підключенні. На відміну від нього, Амперметр згорить, якщо випадково підключити його паралельно до якогось компонента.

Вимірювання струму (альтернатива): альтернативно можна вимірювати струм (напряму) за допомогою датчика Холла. Це стандартна практика в мікроелектроніці, однак він не рекомендований для систем Smart Energy, бо:

- Такий датчик не здатний працювати зі струмом більше приблизно 10А

- Він дуже вразливий до зовнішнього електромагнітного впливу, який виникає від дротів та інших пристроїв

2.2 Розрахунок потужності та обліку енергії

Прийнято класифікувати потреби пристроїв в електроенергії, покладаючись на їх потужність. Потужність можна розрахувати за формулою (2.1) [2]:

$$P = U \cdot I \quad (2.1)$$

де P – потужність, U - напруга, I – сила струму

Для акумулятора доцільно знати його ємність (рахується за формулою 2.2) або кількість збереженої енергії (за формулою 2.3).

$$J = \int_{t_0}^{t_1} I(t) \cdot dt \quad (2.2)$$

де J – ємність, $I(t)$ – сила струму (в залежності від часу t);

t_0 – час початку спостереження, t_1 – час кінця спостереження

$$E = \int_{t_0}^{t_1} P(t) \cdot dt \quad (2.3)$$

де E – к-сть енергії, $P(t)$ – потужність (в залежності від часу t);

t_0 – час початку спостереження, t_1 – час кінця спостереження

Варто зазначити, що основна робота батареї полягає у накопиченні заряду, тобто: зарядці або розрядці.

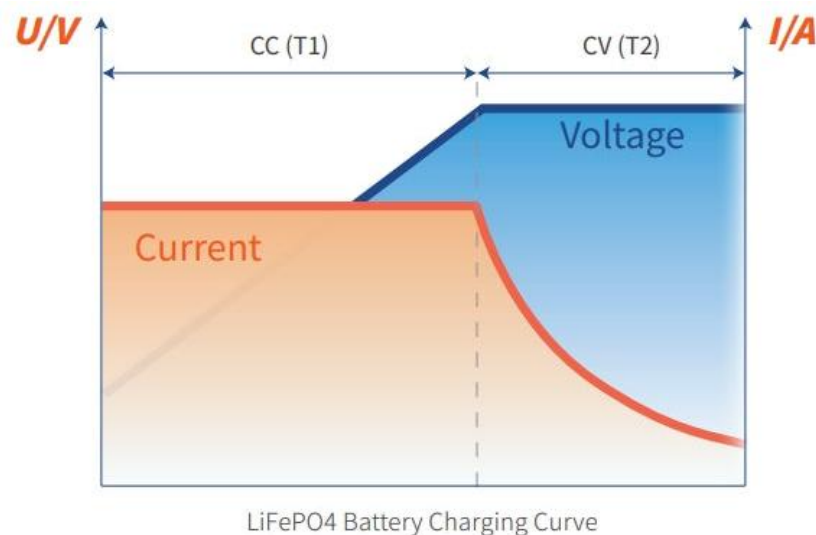


Рисунок 2.1 – Графік залежності струму і напруги для LiFePO4 акумулятора

Чим більше енергії в батареї, тим повільніше вона заряджається і тим менші струм і напруга на ній. Чим менше енергії в батареї, тим повільніше вона розряджається і тим менші струм і напругу видає. Тобто значення струму і напруги не постійні, а змінюються з часом. Графіки, зображені на рисунку 2.1 актуальні для більшості видів акумуляторів. CC означає постійний струм, а CV – постійну напругу (Constant Current / Constant Voltage). [12]

2.3 Апаратні інтерфейси та мережеві протоколи передачі даних

Надійний збір даних з датчиків та їх передача на сервер є критично важливим етапом. Для цього використано комплекс протоколів різного рівня:

- **UART (Universal Asynchronous Receiver-Transmitter) та TTL (Transistor-Transistor Logic):** UART забезпечує асинхронну послідовну передачу даних між мікроконтролером (наприклад, ESP32) та периферійними пристроями. Використання логіки TTL дозволяє інтегрувати прості датчики та модулі, що працюють з низьковольтними рівнями сигналів (зазвичай 3.3 В або 5 В), забезпечуючи базовий рівень комунікації на платі. [8]

- **RS485:** Це стандарт напівдуплексної передачі даних, який ідеально підходить для зчитування показників з промислових датчиків енергії. На відміну від звичайного UART, RS485 використовує диференціальний сигнал, що робить його надзвичайно стійким до електромагнітних перешкод, які неминуче виникають біля інверторів та силових ліній сонячної станції. [7, 8] Це дозволяє передавати дані на значні відстані без втрат. [8]
- **TCP/IP (Transmission Control Protocol / Internet Protocol):** Базовий мережевий протокол, що використовується для передачі даних від мікроконтролера до веб-сервера через Wi-Fi. TCP гарантує доставку пакетів даних без втрат, що важливо для збереження цілісності метрик енергоспоживання.
- **MQTT (Message Queuing Telemetry Transport):** Легковаговий протокол обміну повідомленнями за шаблоном "видавець-підписник" (publish/subscribe). Він є стандартом де-факто для Інтернету речей (IoT). У цій системі мікроконтролер виступає "видавцем", який регулярно публікує дані про напругу та струм, а серверна частина системи підписується на ці оновлення. MQTT ідеально працює в умовах нестабільного з'єднання та потребує мінімум ресурсів мікроконтролера.

2.4 Апаратне забезпечення

Перед початком роботи над системою, в лабораторії КПІ кафедри вже була працююча система з наступних компонентів:

- Сонячна панель
- МРТТ-контролер
- Свинцева батарея VARTA V8
- Інвертор з номінальною потужністю 2000 Вт

В процесі виконання роботи було проаналізовано і обрано додаткові компоненти, для того щоб зробити систему «розумною» і мати змогу зчитувати дані. Для цього необхідно мати датчики для вимірів параметрів (напр. струм і

напруга), контролер, який керує даними (збирає і пересилає їх) і пристроями (отримує команди ззовні і вмикає потрібний функціонал), а також допоміжні компоненти, такі як перетворювачі напруги чи перехідники для протоколів.

2.4.1 Контролер ESP32

Для сучасної «розумної» системи було прийнято рішення будувати електричну систему на основі контролера ESP32. Це дозволило:

- забезпечити нативну підтримку Wi-Fi для передачі даних за протоколом MQTT;
- використовувати обчислювальну потужність для попередньої обробки сигналів з датчиків;
- реалізувати стабільне з'єднання з локальним сервером.

Так як ESP32 працює з живленням і сигналами напругою 3.3 Вольт, необхідно було це також врахувати при виборі інших компонентів. Наприклад, спеціально через це в систему був доданий компонент LM2596, який понижує напругу з 5 Вольт (з виходу додаткових портів контролера) до 3.3 Вольт.

2.4.2 Перетворювач LM2596

У кібер-фізичних системах моніторингу стабільність роботи мікроконтролерів та периферійних цифрових інтерфейсів критично залежить від якості та номіналу напруги живлення. [3] Системний MPPT-контролер може видавати мінімальну напругу 5 Вольт через порт для підключення пристроїв. Однак ESP32 працює тільки з 3.3В і може згоріти, якщо на нього подати занадто високу напругу. Хоча на ESP32 є роз'єм для живлення від 5В, однак це має наступні незручності:

- Певні інші компоненти, такі як датчики, пересилають сигнали на ESP32 з тим вольтажем, до якого вони підключені, а отже для них теж критично

отримувати напругу 3.3 Вольт, інакше їх сигнали будуть занадто сильними і можуть пошкодити ESP32. [13]

- Якщо контролер ESP32 отримує живлення напругою 5 Вольт, то зайву напругу (1.7 Вольт) він розвіює за допомогою резистивного дільника напруг. Отже зайва напруга 1.7В перетворюється в тепло на резисторі, що додатково нагріває контролер і є в цілому не бажаним. [9]

Через наведені причини, для вирішення цієї задачі в апаратній схемі комплексу було застосовано імпульсний понижувальний DC-DC перетворювач на базі інтегральної мікросхеми LM2596.

Вибір цього компонента зумовлений його високою надійністю, простотою інтеграції та низькою вартістю. На відміну від класичних лінійних стабілізаторів (наприклад, серії 7805), які розсіюють надлишкову потужність у вигляді тепла, LM2596 працює за принципом ШІМ з частотою перемикання 150 кГц. Цей пристрій працює таким чином, що він відкриває і закриває прохід для напруги через себе так, щоб в середньому за певний часовий проміжок на виході була задана кількість напруги. Щоб згладити вихідну напругу використовуються котушки і конденсатори. [10] Це дозволяє досягти коефіцієнта корисної дії (ККД) до 92%, що мінімізує нагрівання всередині та зменшує власне енергоспоживання системи моніторингу.

2.4.3 Датчик напруги та струму INA226

Датчик INA226 було обрано для прецизійного моніторингу низьковольтних ланцюгів постійного струму (наприклад, окремих ліній сонячних панелей або акумуляторних батарей). Основні технічні переваги:

- **Висока точність:** пристрій дозволяє вимірювати напругу шини до 36 В та падіння напруги на шунті з похибкою не більше 0.1%.
- **Інтерфейс I2C:** завдяки підтримці шини I2C, датчик легко підключається до мікроконтролера ESP32, дозволяючи програмно налаштовувати час усереднення та режим роботи.

- **Обчислення потужності:** INA226 має вбудований математичний двигун, який автоматично розраховує потужність у ватах, що знижує обчислювальне навантаження на основний процесор системи.

4.1.4 Модуль моніторингу електроенергії PZEM-017

Для вимірювання параметрів у потужніших ланцюгах постійного струму та ведення обліку спожитої енергії було інтегровано модуль PZEM-017. Його роль у системі обумовлена наступними характеристиками:

- **Діапазон вимірювань:** модуль підтримує роботу з напругою до 300 В та великими струмами (залежно від зовнішнього шунта), що робить його ідеальним для моніторингу загальної генерації системи або потужних інверторів.
- **Протокол RS485/Modbus-RTU:** використання промислового стандарту передачі даних забезпечує високу завадостійкість на довгих лініях зв'язку. Це критично важливо в умовах лабораторії, де силове обладнання може створювати значні електромагнітні шуми.
- **Енергонезалежна пам'ять:** PZEM-017 самостійно накопичує дані про спожиту/згенеровану енергію (Вт·год). Навіть при тимчасовому зникненні живлення або перезавантаженні мікроконтролера, дані про загальний виробіток залишаються збереженими в пам'яті модуля.

2.4.5 Модуль-конвертер зв'язку (TTL to RS485)

Передача даних між вимірювальними компонентами та центральним обчислювачем часто ускладнюється. Оскільки інтегральний датчик PZEM-017 використовує для обміну інформацією промисловий інтерфейс RS-485, а апаратне ядро системи на базі мікроконтролера ESP32 оперує стандартними логічними рівнями TTL, виникає проблема апаратної несумісності фізичних рівнів. Для взаємодії цих пристроїв у загальну електричну схему інтегровано спеціалізований модуль-конвертер зв'язку TTL to RS485.

Вибір цього модуля зумовлений його архітектурною специфікою — можливістю автоматичного керування передачею даних. На відміну від класичних мікросхем типу MAX485, які вимагають виділення додаткового виводу (GPIO) мікроконтролера для перемикання режимів приймання/передачі (лінії RE/DE), цей модуль здійснює комутацію автоматично на апаратному рівні. Це дозволяє вивільнити обмежені апаратні ресурси ESP32 та спростити програмний код прошивки.

З боку TTL-інтерфейсу модуль підключається до виділеного апаратного послідовного порту мікроконтролера ESP32: вивід TXD конвертера з'єднується з GPIO16 (RX2), а вивід RXD — з GPIO17 (TX2). Живлення модуля здійснюється від стабілізованої лінії 3.3В, що виходить з перетворювача LM2596. З протилежного боку диференційна лінія зв'язку підключається до клем А та В датчика PZEM-017 за допомогою кабелю типу «вита пара». Це забезпечує надійне зчитування регістрів по логічному протоколу Modbus RTU без втрати або спотворення бітів телеметрії навіть в умовах сильного електромагнітного фону від інвертора чи інших пристроїв.

2.4 Побудова функціональної електричної схеми

Зважаючи на наведені електричні електричні основи, формули, протоколи передачі даних та компоненти, розглянуті в попередніх підрозділах було створено функціональну електричну схему, зображену на рисунку 2.2.

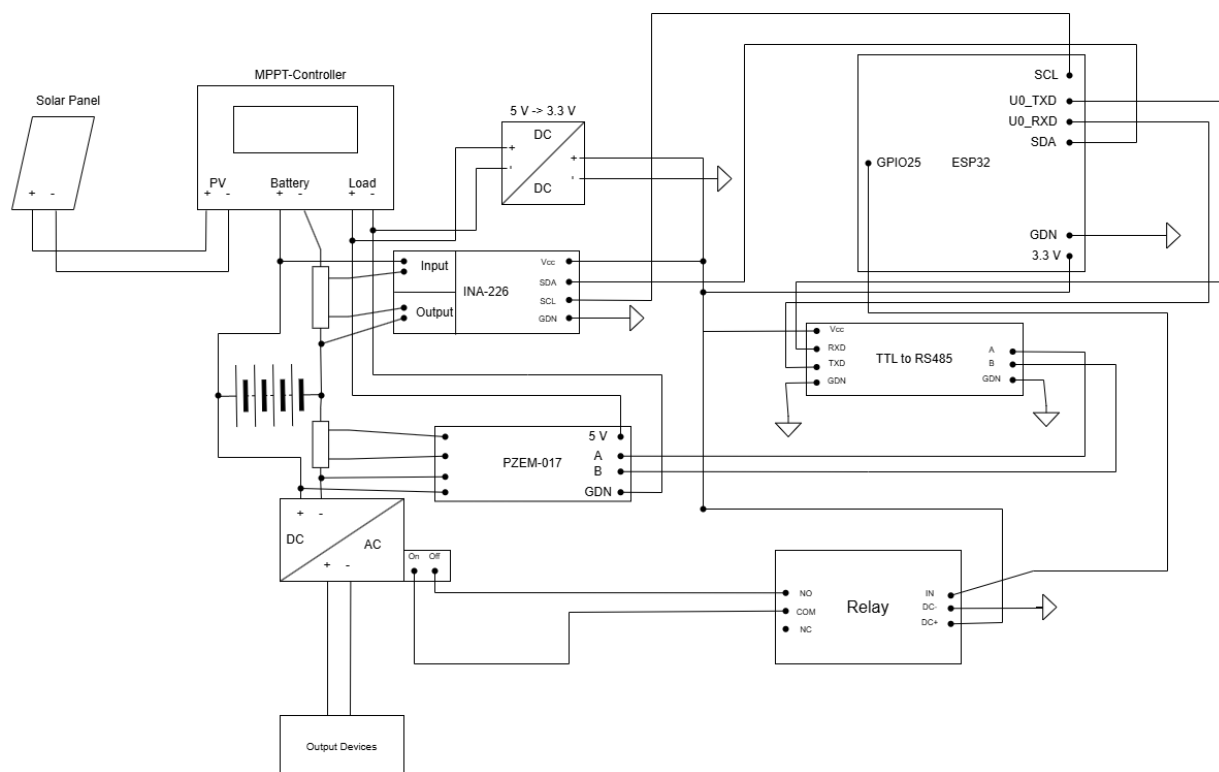


Рисунок 2.2 – функціональний вигляд електричної розробленої схеми

Основою апаратної архітектури є мікроконтролер ESP32, який виконує роль центрального обчислювального вузла. Стабільне живлення логічної частини напругою 3.3 Вольт забезпечується за допомогою LM2596, який підключається до виходу MPPT-контролера 5 Вольт і понижує вихідну напругу до 3.3 Вольт для безпечного функціонування контролера і інших датчиків.

Моніторинг системи розділено на два незалежні канали вимірювання:

- Канал від MPPT-контролера до акумуляторної батареї (загальний притік енергії): Побудований на базі вимірювального модуля INA226, який підключається однією частиною паралельно до акумулятора, а іншою – послідовно через додатковий зовнішній шунт. Модуль здійснює двонаправлений моніторинг напруги батареї та падіння напруги на шунті (для обчислення струму), що дозволяє фіксувати як процеси заряджання, так і розряджання АКБ. [11] Обмін даними з ESP32 відбувається через синхронну шину I2C за допомогою ліній SDA та SCL, які заживлені від робочої напруги 3.3 В.

- Канал від батареї до інвертора (загальний відтік енергії): Реалізований на базі інтегрального датчика PZEM-017, який підключається і вимірює струм і напругу, які йдуть від батареї до інвертора. Так як значення струму можуть бути досить великі, вимір струму відбувається через шунт. Оскільки PZEM-017 і ESP32 мають різні інтерфейси передачі даних, між ними стоїть модуль-конвертер TTL to RS485. Сам PZEM-017 заживлений безпосередньо від 5В, які виходять з MPPT-контролера, оскільки 3.3В для нього замало. [10]

Для кращого сприйняття і розуміння, було також створено концептуальну модель системи на рисунку 2.3, яка дає більш реальний погляд на стан підключень і поєднань пристроїв.

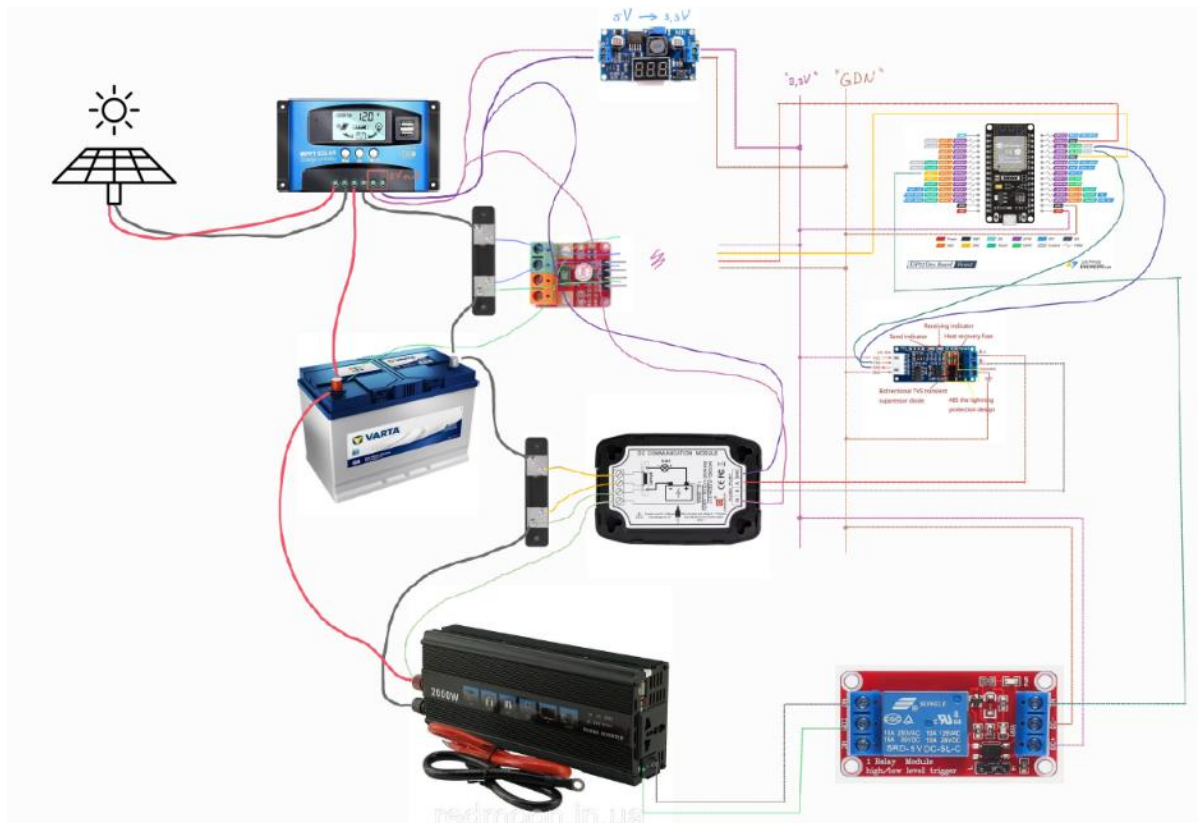


Рисунок 2.3 – Концептуальна модель системи

Розроблена топологія схеми забезпечує повне розділення силових енергетичних потоків від слабкострумівих інформаційних ліній, що гарантує високу завадостійкість, точність зчитування метрик та загальну електричну безпеку кібер-фізичного комплексу в SmartEnergy Lab.

Висновки до розділу 2

У другому розділі було детально розглянуто та обґрунтовано електротехнічні основи розробки системи моніторингу. Визначено ключові фізичні принципи вимірювання напруги за та сили струму. Наведено математичні формули для розрахунку потужності, ємності та споживаної/генерованої енергії. Оглянуто необхідні компоненти і побудовано функціональну електричну схему.

3 ЗАСОБИ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Платформа і стек програмного забезпечення

При розробці кібер-фізичної системи моніторингу критично важливим етапом є вибір платформи програмного забезпечення. Традиційні десктопні рішення або монолітні системи локального керування суттєво обмежують мобільність користувача та масштабованість системи. Для забезпечення кросплатформеності, можливості одночасного доступу багатьох користувачів з різних девайсів та інтерактивності було обрано підхід (стек) **Web Fullstack**, який зокрема передбачає розділення системи на серверну (Backend) та клієнтську (Frontend) частини, які взаємодіють через API.

Вибір підходу Web Fullstack має низку суттєвих переваг в порівнянні з іншими можливими архітектурними рішеннями. Основними перевагами є:

- **Єдиний технологічний стек:** використання однієї мови програмування для розробки як клієнтської частини (Frontend), так і серверної логіки (Backend) суттєво спрощує розробку проєкту. Строга типізація мови TypeScript на всіх рівнях мінімізує ризик виникнення помилок при передачі структурованих JSON-даних від сервера до клієнта.

- **Платформа Node.js** працює на базі рушія V8 та використовує Event Loop (асинхронну подієво-орієнтовану модель введення-виведення). Event Loop – це як нескінченний цикл всередині коду, який відстежує і перехоплює всі події (івенти) для їх виконання і обробки. Для систем моніторингу, де дані надходять безперервно у вигляді потоку метрик, здатність сервера обробляти тисячі одночасних неблокуючих з'єднань є критично важливою. Це гарантує, що система не «зависатиме» під час паралельного запису даних у базу даних та обслуговування HTTP-запитів користувачів.

- **Кросплатформеність:** так як розроблений програмний продукт функціонує у веб-браузері, кінцевому користувачу не потрібно встановлювати специфічне програмне забезпечення чи драйвери. Інтерфейс однаково коректно працює на персональних комп'ютерах під керуванням Windows/Linux, інформаційних табло, а також на мобільних пристроях і планшетах завдяки адаптивній верстці.
- **Ефективна інтеграція з IoT-протоколами:** стек технологій Web Fullstack має нативну підтримку роботи з сучасними мережевими протоколами з низькою затримкою, такими як WebSockets та MQTT. Це дозволяє реалізувати концепцію актуальності даних, коли виміряні датчиками PZEM-017 та INA226 параметри напруги й струму миттєво відображаються на графіках користувача, минаючи стадію постійного примусового оновлення сторінки браузера.
- **Гнучкість візуалізації та аналітики:** екосистема веб-розробки має відкриту бібліотеку для побудови інтерактивних графіків Recharts. Це дозволяє реалізувати складні інтерфейсні рішення для аналізу даних виробництва і споживання електроенергії, динамічного масштабування графіків та кастомізації віджетів під конкретні умови і задачі.

3.2 Мови програмування

3.3.1 Typescript

Для розробки серверної частини (Backend) та користувацького вебінтерфейсу (Frontend) було обрано мову програмування TypeScript. Вона є строго типізованим розширенням мови JavaScript; компілюється в чистий JavaScript для виконання в середовищі Node.js або браузері. Вона підтримується у всіх сучасних середовищах розробки і містить увесь функціонал мови JavaScript. Ще перевагою є те, що ці дві мови абсолютно сумісні одна з одною.

3.3.2 INO (мова Arduino)

Для програмування логіки роботи мікроконтролера ESP32, який безпосередньо взаємодіє з цифровими датчиками, було використано мову Arduino (файл-розширення .ino). Ця створена мова на основі мов C/C++, але використовує специфічний апаратний фреймворк Arduino Core. Фактично ця мова є розширенням мови C++, з певними власними бібліотеками та структурами.

3.3.3 JSON (JavaScript Object Notation)

Для організації безперебійного, легковагового та зрозумілого обміну інформацією між фізичним рівнем (ESP32), сервером збору даних та клієнтським дашбордом застосовано текстовий формат структуризації даних JSON. Хоча JSON не є сам по собі мовою програмування, однак він тісно пов'язаний з JavaScript і TypeScript. Мала вага пакетів у порівнянні з важкими XML-структурами дозволяє транслювати телеметрію з високою частотою, не перевантажуючи локальну мережу.

3.4 Серверна частина (Backend)

Backend відповідає за прийом даних від контролерів, їх обробку та надання доступу до них для клієнтського додатку. Під час створення серверної частини було використано наступні засоби розробки

- **Express.js:** Мінімалістичний та гнучкий веб-фреймворк для середовища Node.js. Він використаний для створення REST API, яке обробляє запити від браузера користувача (отримання історичних даних, статус системи).
- **Dotenv:** Модуль для завантаження змінних середовища з файлу .env. У системі моніторингу він забезпечує безпеку: всі конфігураційні дані (паролі до баз даних, ключі доступу до MQTT-брокера, порти) зберігаються окремо від вихідного коду, що відповідає сучасним стандартам кібербезпеки.

- **mqtt:** це бібліотека є високопродуктивним клієнтом для протоколу MQTT, написаним повністю на JavaScript з нативною підтримкою типізації TypeScript, що дозволяє надсилати та отримувати дані через MQTT -брокер.

3.5 База даних InfluxDB

Дані надходять безперервно через однакові невеликі проміжки часу. Кожен запис містить часову мітку (timestamp) та набір вимірних параметрів. Тобто при плануванні/взаємодії з такою системою треба передбачати, що даних буде дуже багато, і що вони будуть надходити дуже часто.

Тому збереження таких даних у класичних послідовних нескінченних ланцюгів даних у реляційних базах даних (наприклад, MySQL або PostgreSQL) є неефективним через швидке зростання обсягів таблиць та уповільнення виконання аналітичних запитів, наприклад при порівнянні режимів даних генерації за різні проміжки часу. Необхідна також зручна агрегація даних (напр. обчислення суми, середнього значення).

Тому для збереження таких даних було обрано спеціалізовану базу даних часових рядів – **InfluxDB**. Ця СУБД оптимізована для швидкого запису та читання величезних масивів даних з часовими мітками. Це дозволяє миттєво формувати вибірки даних для графіків, щоб продемонструвати їх користувачу. Також до цієї бази даних є бібліотеки, які дозволяють легко виконувати операції читання та запису даних в програмному коді.

Структура логічного запису метрики в цій базі даних формалізується таким чином:

$$\text{DataPoint} = \langle \text{Timestamp}, \text{Measurement}, \text{Tags}, \text{Fields} \rangle \quad (3.1)$$

де:

- **Timestamp:** Час фіксації параметрів (в мілісекундах)
- **Measurement:** назва вимірювання

- Tags (Індексовані метадані): Ідентифікатори вузлів вимірювання (наприклад, назва девайсу) для швидкого фільтрування даних за конкретними параметрами
- Fields: значення метрик – наприклад напруга (voltage), струм (current), потужність (power).

Саме така структура ідеально підходить для розроблюваної системи моніторингу. Також influxDB має зручну адмін панель керування, в якій також можна переглянути всі збережені дані і за необхідності завантажити їх.

3.6 MQTT-брокер

Постійне опитування сервера мікроконтролером створює надлишкові накладні витрати на мережевий трафік та обчислювальні ресурси ESP32. Тому для передачі даних від ESP32 до бекенду і фронтенду було додано MQTT-брокер Eclipse Mosquitto. Суть в тому, що брокер це універсальний шлюз передачі даних, а інші компоненти системи – його підписники, які публікують/слухають повідомлення.

Eclipse Mosquitto – це відкритий (open-source) брокер повідомлень, який реалізує протоколи MQTT версій 5.0, 3.1.1 та 3.1. Він є стандартом де-факто в індустрії Інтернету речей (IoT) та кібер-фізичних системах завдяки своїй компактності, швидкодії та мінімальним вимогам до оперативної пам'яті сервера.

Брокер підтримує функцію збереження останнього надісланого повідомлення у топіку. Коли веб-інтерфейс або новий мікросервіс підключається до системи, йому не потрібно чекати наступного циклу вимірювання від ESP32 – Mosquitto миттєво віддає останній збережений стан системи, що прискорює ініціалізацію інтерфейсу користувача.

У межах розроблюваної системи Mosquitto розгортається як ізольований Docker-контейнер. Конфігураційні файли `mosquitto.conf`, `mosquitto.acl` дозволяють чітко розмежувати права доступу, налаштувати автентифікацію за логіном і

паролем для ESP32 та серверного бекенду, а також виділити стандартний порт 1883 для внутрішнього обміну даними.

3.7 Клієнтська частина (Frontend)

Клієнтська частина — це інтерфейс, з яким безпосередньо взаємодіє користувач системи для аналізу даних. Під час створення клієнтської частини були використані наведені засоби розробки:

- **React:** Найпопулярніший фреймворк для створення користувацьких інтерфейсів. Завдяки компонентному підходу, інтерфейс моніторингу розбито на незалежні блоки (картки поточного стану, панелі графіків, налаштування). Це дозволяє інтерфейсу оновлюватися динамічно без перезавантаження сторінки, миттєво реагуючи на нові дані по MQTT.
- **Tailwind CSS:** Фреймворк для стилізації на основі утилітарних класів (utility-first). Він дозволив швидко створити сучасний, професійний та повністю адаптивний дизайн без довгого пропису CSS. Інтерфейс коректно відображається як на великих екранах (для демонстраційного стенду в лабораторії), так і на екранах мобільних телефонів.
- **Recharts:** Спеціалізована бібліотека для побудови діаграм у React-додатках. Саме за допомогою неї реалізовано ключову функцію системи — візуалізацію аналітики. Бібліотека дозволяє будувати плавні, інтерактивні графіки генерації та споживання, де користувач може навести курсор на певну точку і побачити точні показники енергії в конкретний момент часу.
- **Lucide-react:** Бібліотека векторних іконок, яка використовується для покращення навігації та візуального сприйняття інтерфейсу (іконки батареї, сонячних панелей, тощо). Забезпечує професійний стилістичний вигляд системи.

3.8 Контейнеризатор Docker

Оскільки проектована система моніторингу містить кілька різнорідних сервісів: базу даних часових рядів InfluxDB, MQTT-брокер Mosquitto, фоновий сервіс збору телеметрії та аналітичний Backend API, виникає потреба в інструменті, який забезпечить ізоляцію, легке масштабування та швидке розгортання всього стеку. Для вирішення цих завдань було обрано платформу контейнеризації Docker.

Docker – це програмне забезпечення з відкритим кодом, яке автоматизує розгортання застосунків у вигляді ізольованих контейнерів. На відміну від класичних технологій віртуалізації (апаратних віртуальних машин), Docker не потребує запуску повноцінної гостьової операційної системи для кожного сервісу. Замість цього контейнери використовують спільне ядро хостової ОС (завдяки механізмам ядра Linux: cgroups та namespaces), що забезпечує мінімальні накладні витрати на процесорну потужність та оперативну пам'ять сервера.

Висновки до розділу 3

У третьому розділі було проведено комплексний аналіз та обґрунтовано вибір інструментальних засобів, платформ і мов програмування, необхідних для створення системи моніторингу в реальному часі. Обґрунтовано вибір архітектурного підходу Web Fullstack, визначено необхідні мови програмування, обрано базу даних, обрано і аргументовано брокер, наведено необхідний стек технологій для фронтенду і бекенду.

4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Розробка програмної системи моніторингу Smart Energy пройшла шлях від створення базових прототипів на базі мікроконтролерів до побудови комплексної Fullstack-платформи. Основним орієнтиром під час реалізації була масштабованість системи, професійний вигляд та можливість роботи в режимі реального часу.

4.1 Архітектура програмної системи

Для розробки і використання системи моніторингу була спроектована архітектура, зображена на рис 4.1.

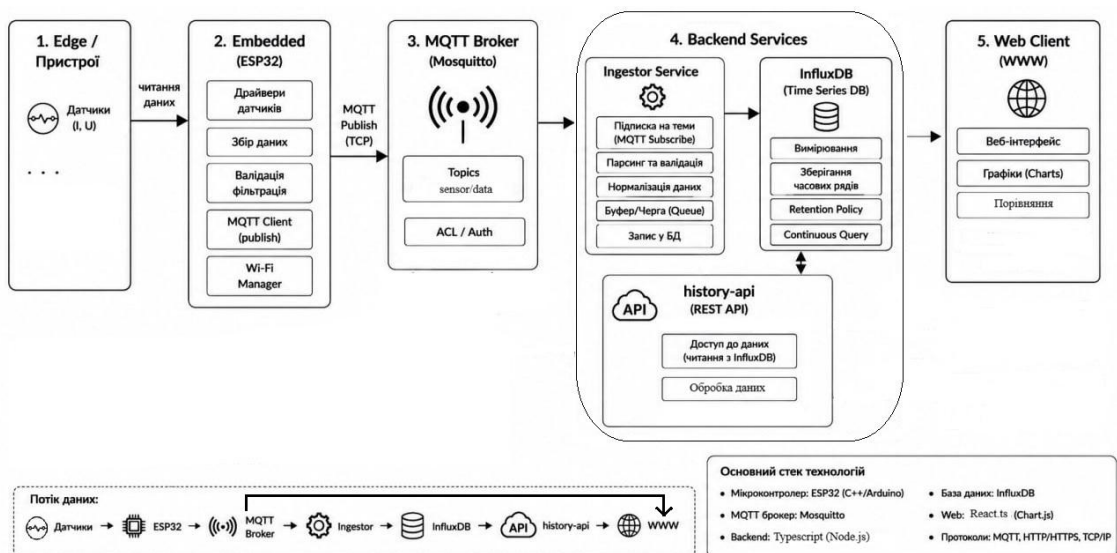


Рисунок 4.1 – Архітектура системи моніторингу

Опис архітектури:

- Кінцевими пристроями є датчики. Вони збирають всю інформацію і передають її на ESP32 – головний контролер системи;
- ESP32 зв'язується з MQTT-брокером і передає дані йому;
- MQTT-брокер передає дані далі на бекенд (ingestor service), а також напряму у веб-клієнт, щоб користувач одразу бачив дані, не очікуючи відповіді від бази даних.

- Ingestor нормалізує отримані дані і вносить їх в базу даних influxDB. Варто зазначити, що також через вимоги influxDB, ingestor не одразу вносить дані в базу даних, а спочатку збирає невеликий набір цих даних і лише потім вони всі разом вносяться в influxDB. Через це, наприклад, неможливо зчитати актуальні дані одразу з бази даних у веб клієнт;
- Компонент history-арі підключається до бази даних і переглядає дані. Веб клієнт робить запити до цього компоненту, щоб отримати історичні дані за минулі періоди (наприклад за останні 5 хв, місяць чи рік). Також можна перевірити статус запису даних в базу даних;
- Веб клієнт взаємодіє з користувачем і показує йому відповідні статуси, дані і діаграми.

4.2 Апаратне програмне забезпечення

Для контролера ESP32 був написаний програмний код за допомогою мови Arduino. В самому початку визначено параметри пристрою для підключення до мережі WiFi та MQTT-брокера. Параметри та вказівки до них зображено на рисунку 4.2.

```
// Налаштування мережі та брокера
// ssid - Назва мережі WiFi
// password - Пароль мережі WiFi
// mqtt_broker - Адреса брокера
// client_id - Назва клієнта (цього пристрою) для брокера
const char* ssid = ""; // Назва мережі WiFi
const char* password = ""; // Пароль мережі WiFi
const char* mqtt_broker = "192.168.0.100"; // Адреса брокера, наприклад, 192.168.1.100 (або сайт)
const char* client_id = "ESP32_PHYSICAL_DATA_SIMULATOR"; // Назва клієнта (цього пристрою) для брокера
const char* mqtt_username = ""; // ім'я для підключення до брокера
const char* mqtt_password = ""; // пароль для підключення до брокера
```

Рисунок 4.2 – Параметри налаштування підключення до мережі і брокера

Особливо вразливі параметри, такі як назва і пароль мережі або пароль для підключення до брокера були приховані на рисунку 4.2.

Для отримання актуального часу використовується сервер pool.ntp.org. Він гарантує актуальність і точність часу вимірювань. Отримання актуального часу продемонстровано на рисунку 4.3.

```
24 WiFiUDP ntpUDP;  
25 const int UTC_0_shift = 0;  
26 NTPClient timeClient(ntpUDP, "pool.ntp.org", UTC_0_shift);
```

Рисунок 4.3 – Отримання актуального часу з серверу pool.ntp.org

Так як фізичних датчиків в лабораторії немає, їх дані емулюються. Була спроба використати спеціалізовані бібліотеки для формування JSON-форми для пересилання даних, однак вони працювали некоректно. Тому був обраний метод безпосереднього створення JSON-строки і її відправки, зображений на рисунку 4.4.

```
93 unsigned long long timestampMs = ((unsigned long long) timeClient.getEpochTime()) * 1000;  
94  
95 String payload = "{";  
96 payload += "\"status\": \"online\",";  
97 payload += "\"timestamp\":" + String(timestampMs) + ",";  
98  
99 // Блок Solar  
100 payload += "\"solar\":{" + "\"v\":" + String(solarV) + "," + "\"i\":" + String(solarI) + "},"";  
101  
102 // Блок Charge  
103 payload += "\"charge\":{" + "\"v\":" + String(chargeV) + "," + "\"i\":" + String(chargeI) + "},"";  
104  
105 // Блок Discharge  
106 payload += "\"discharge\":{" + "\"v\":" + String(dischargeV) + "," + "\"i\":" + String(dischargeI) + "},"";  
107  
108 // Блок Battery  
109 payload += "\"battery\":{" + "\"charge\":" + String(batteryCharge) + "},"";  
110  
111 // Блок Devices (приклад з декількома пристроями)  
112 payload += "\"devices\":{";  
113 payload += "\"dev1\":{" + "\"v\":" + String(230) + "," + "\"i\":" + String(0.5 + (random(0, 50) / 100.0)) + "},"";  
114 payload += "\"dev2\":{" + "\"v\":" + String(230) + "," + "\"i\":" + String(1.2 + (random(0, 80) / 100.0)) + "},"";  
115 payload += "\"dev3\":{" + "\"v\":" + String(0) + "," + "\"i\":" + String(0) + "},"";  
116 payload += "},"";  
117  
118 payload += "}";  
119 client.publish("sensor/data", payload.c_str(), false);  
120 Serial.println("Повний пакет відправлено:");
```

Рисунок 4.4 – формування і відправка даних

Однак все ж треба передбачити можливість зчитування даних з датчиків, тому ця програмна реалізація теж була додана окремо, в демонстраційному вигляді. Зчитування даних з датчика INA-226 наведено на рисунку 4.5, а з датчика PZEM-017 на рисунку 4.6.

```

34 float shuntVoltage_mV = 0.0;
35 float busVoltage_V = 0.0;
36 float current_mA = 0.0;
37 float power_mW = 0.0;
38
39 ina226.readAndClearFlags();
40
41 shuntVoltage_mV = ina226.getShuntVoltage_mV();
42 busVoltage_V = ina226.getBusVoltage_V();
43 shuntVoltage_mV = ina226.getShuntVoltage_mV();
44 current_mA = ina226.getCurrent_mA();
45
46 power_mW = ina226.getBusPower();
47
48 Serial.print("Bus Voltage: "); Serial.print(busVoltage_V); Serial.println(" V");
49 Serial.print("Shunt Voltage: "); Serial.print(shuntVoltage_mV); Serial.println(" mV");
50 Serial.print("Current: "); Serial.print(current_mA); Serial.println(" mA");
51 Serial.print("Power: "); Serial.print(power_mW); Serial.println(" mW");
52 Serial.println("-----");

```

Рисунок 4.5 – зчитування даних з датчика INA-226

```

28 // Request 6 registers from the PZEM
29 uint8_t result = node.readInputRegisters(0x0000, 6);
30 Serial.print("Result: "); Serial.println(result, HEX);
31 // Check if the result is 0x00 (E0)
32 if (result == 0x00) {
33     // SUCCESS: The PZEM responded with data!
34
35     // Voltage is in register 0 (0.01V resolution)
36     float voltage = node.getResponseBuffer(0x00) / 100.0;
37
38     // Current is in register 1 (0.01A resolution)
39     float current = node.getResponseBuffer(0x01) / 100.0;
40
41     // Power is 32-bit (Registers 2 and 3)
42     uint32_t power_raw = ((uint32_t)node.getResponseBuffer(0x03) << 16) | node.getResponseBuffer(0x02);
43     float power = power_raw / 10.0;
44
45     // Energy is 32-bit (Registers 4 and 5)
46     uint32_t energy = ((uint32_t)node.getResponseBuffer(0x05) << 16) | node.getResponseBuffer(0x04);
47
48     // Print the actual values to the console
49     Serial.println("--- PZEM-017 MEASUREMENTS ---");
50     Serial.print("Voltage: "); Serial.print(voltage); Serial.println(" V");
51     Serial.print("Current: "); Serial.print(current); Serial.println(" A");
52     Serial.print("Power: "); Serial.print(power); Serial.println(" W");
53     Serial.print("Energy: "); Serial.print(energy); Serial.println(" Wh");
54     Serial.println("-----");
55
56 } else {
57     // ONLY print this if the code is NOT 0 (E0)
58     Serial.print("Actual Error Code: 0x");
59     Serial.println(result, HEX);
60 }

```

Рисунок 4.6 – зчитування даних з датчика PZEM-017

Обидва датчики були протестовані і показали високу точність результатів вимірювань.

4.3 Реалізація серверної частини (Backend) та обробка потокових даних

Серверна частина системи (Backend) реалізована на базі середовища Node.js з використанням фреймворку Express. Вона поділена на 2 незалежних компоненти: ingestor, history-api.

Компонент ingestor:

- Отримує дані з mqtt-брокера (рис. 4.7)
- Приводить їх до необхідного вигляду (рис 4.8) і записує дані в базу даних (рис. 4.9)

```
const writeApi : WriteApi = influxDB.getWriteApi(INFLUX_ORG!, INFLUX_BUCKET!, { precision: "ms" })
const mqttClient : MqttClient = mqtt.connect(MQTT_URL!, {
  clientId: clientId,
  username: MQTT_USER,
  password: MQTT_PASSWORD,
  clean: true // Брокер видалятиме підписки та незавершені повідомлення при відключенні //!
})

// Підключення до MQTT
mqttClient.on("connect", () : void => {
  console.log("Ingestor: підключено до MQTT брокера!")
  mqttClient.subscribe(
    { topicObject: "sensor/data",
      err : Error | null => {
        if (!err) {
          console.log("Ін'єктор підписався на sensor/data (MQTT)")
        }
      }
    }
  )
})
```

Рисунок 4.7 – Визначення API запису даних в базу даних і підключення до брокера в компоненті ingestor

```
// Обробка повідомлень
mqttClient.on( event: "message", (topic :string , message :Buffer<ArrayBufferLike> ) :void => { & Viktor
    const payload :string = message.toString()
    if (topic == "sensor/data") {
        try {
            const data :any = JSON.parse(payload)

            if(data.status == "offline"){
                console.log( ...data: "Контролер не в мережі (офлайн)"
                return
            }

            const point :Point = new Point(DATA_SAVE_TOPIC)
                .tag( name: "deviceId", value: "esp32")
                .floatField( name: "solar_v", data.solar?.v || 0)
                .floatField( name: "solar_i", data.solar?.i || 0)
                .floatField( name: "charge_v", data.charge?.v || 0)
                .floatField( name: "charge_i", data.charge?.i || 0)
                .floatField( name: "discharge_v", data.discharge?.v || 0)
                .floatField( name: "discharge_i", data.discharge?.i || 0)
                .floatField( name: "solar_p", (data.solar?.v || 0) * (data.solar?.i || 0))
                .floatField( name: "charge_p", (data.charge?.v || 0) * (data.charge?.i || 0))
                .floatField( name: "discharge_p", (data.discharge?.v || 0) * (data.discharge?.i || 0))
                .floatField( name: "battery_charge", data.battery?.charge || 0)
                .timestamp(new Date(data.timestamp))

            // Write device data if needed, or just specific device fields
            if (data.devices) {
                Object.keys(data.devices).forEach(devId :string => {
                    point.floatField( name: `${devId}_v`, data.devices[devId].v);
                    point.floatField( name: `${devId}_i`, data.devices[devId].i);
                    point.floatField( name: `${devId}_p`, data.devices[devId].v * data.devices[devId].i);
                });
            }
        }
    }
})
```

Рисунок 4.8 – Обробка і нормалізація вхідних даних з ESP32 у компоненті ingestor

```
// Збереження даних в InfluxDB
writeApi.writePoint(point)

// Лог
console.log(new Date(data.timestamp).toLocaleString( locales: "uk-UA"), "Solar P:", (data.solar?.p || 0))
} catch (error) {
    console.error( ...data: "Помилка парсингу:", error instanceof Error ? error.message : error)
    console.error( ...data: "Payload:", payload)
}
```

Рисунок 4.9 – збереження даних в базі даних

Компонент history-api:

- Підключається до бази даних і відповідає на API-запити фронтенду
- Запитом «/api/history» обробляє передані з фронтенду параметри, такі як початкова дата, кінцева дата, розмір вікна агрегації, бере відповідні дані з бази даних і повертає їх у відповіді. Приклад обробки запиту наведено на рис. 4.10.

- За запитом «/api/status» система перевіряє в базі даних останню інформацію про збережені дані і повертає час, коли були збережені останні дані, і інші суміжні дані

```
// Обробник запитів на отримання історичних даних
app.get( path: '/api/history', async (req: Request<I>, any, any, ParsedQs, Record<string, any>, number...) : Promise<void> => { & Viktor
  console.log( message: 'Incoming request: ${req.url}');
  try {
    // параметри запитів до influxDB
    const range :string = (req.query.range as string) || '1h';
    const window = req.query.window as string;
    const offset = req.query.offset as string;
    const start = req.query.start as string;
    const end = req.query.end as string;

    let query :string = '';
    if (start && end) {
      console.log( message: "START&END category")
      const windowClause :string = window ? '> aggregateWindow(every: ${window}, fn: mean, createEmpty: false ${offset ? ", offset: "+ offset:""})' : '';
      query = `
        from(bucket: "${INFLUX_BUCKET}")
        |> range(start: time(v: "${start}"), stop: time(v: "${end}"))
        |> filter(fn: (r) => r["_measurement"] == "${DATA_SAVE_TOPIC}")
        ${windowClause}
        |> pivot(rowKey:["_time"], columnKey: ["_field"], valueColumn: "_value")
        |> drop(columns: ["_start", "_stop", "_measurement", "deviceId"])
        |> sort(columns: ["_time"])
      `;
    } else {
```

Рисунок 4.10 – приклад обробки запитів до history-api

Бекенд-модулі є самостійними API-сервісами. При виході з ладу одного з модулів, це не завадить роботі іншого, а користувач буде проінформований через користувацький інтерфейс.

4.4 Розробка клієнтського інтерфейсу користувача (Frontend)

Frontend системи побудований як сучасний сайт на базі React. Головною метою було створення інтуїтивно зрозумілого та адаптивного інтерфейсу. Професійний інтерфейс є зручним у використанні, а також привертає увагу відвідувачів стенду.

Програмна реалізація інтерфейсу включає сторінки:

- Головна панель (Dashboard)
- Панелі аналітики (Генерація vs Споживання енергії, Сонячні панелі, МРРТ, Загальне споживання)
- Панель порівняння параметрів

4.4.1 Головна панель

Вигляд головної панелі зображено на рисунках 4.11-4.15.

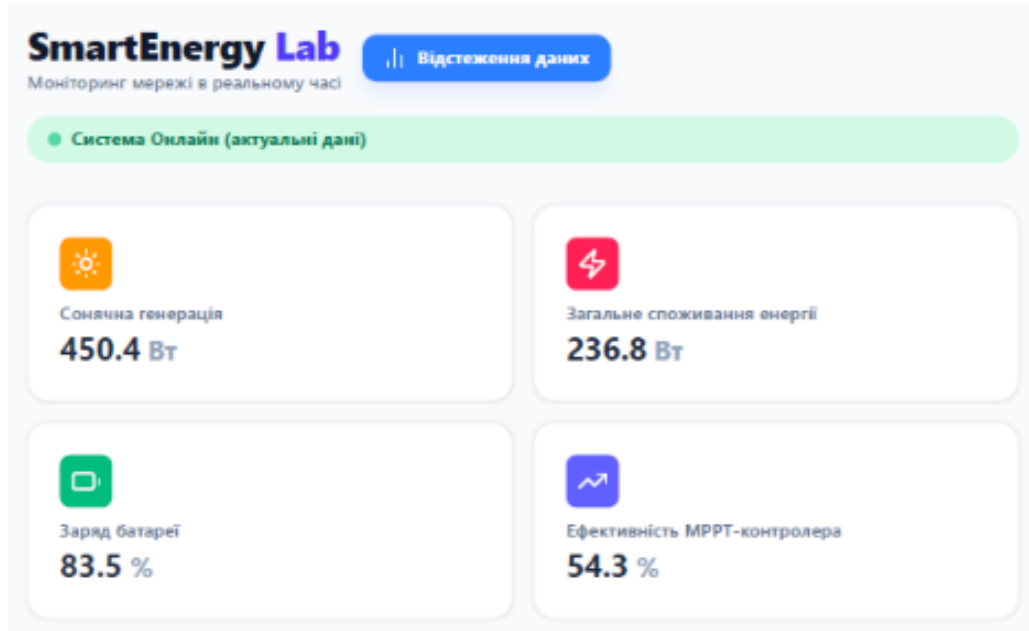


Рисунок 4.11 – Верхня частина головної панелі

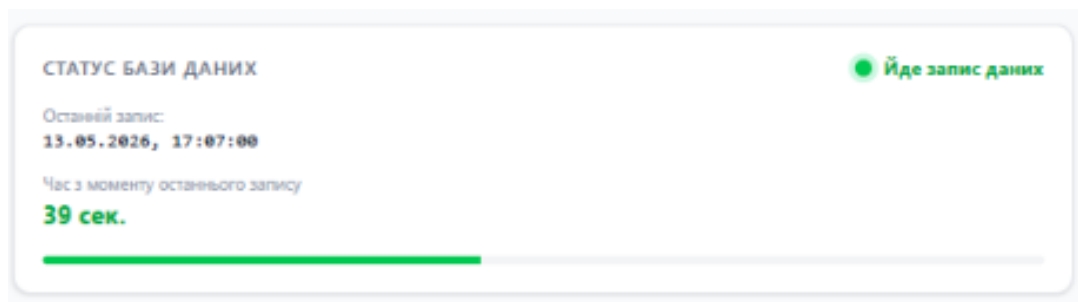


Рисунок 4.12 – Розділ статусу бази даних на головній панелі



Рисунок 4.13 – Головна діаграма загальної генерації і споживання

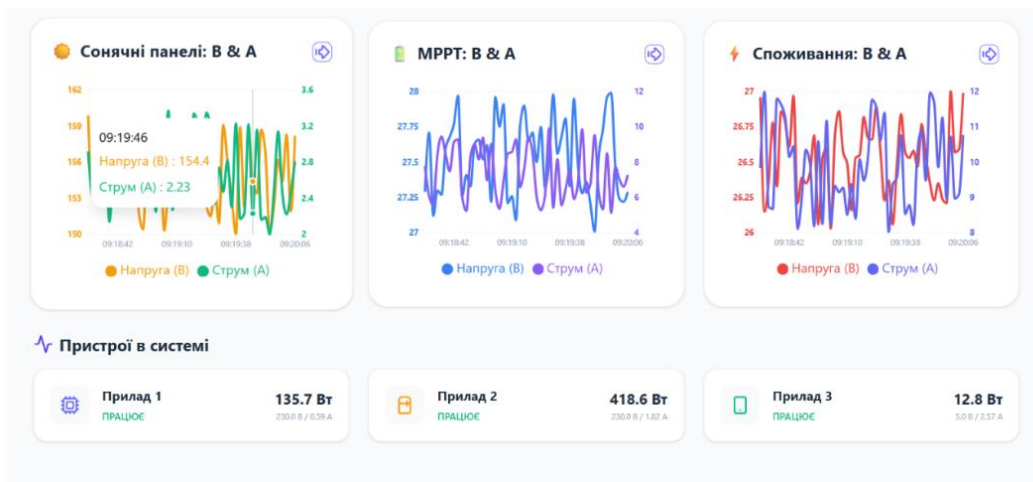


Рисунок 4.14 – Допоміжні діаграми для сонячних панелей, MPPT-контролера, споживання електроенергії

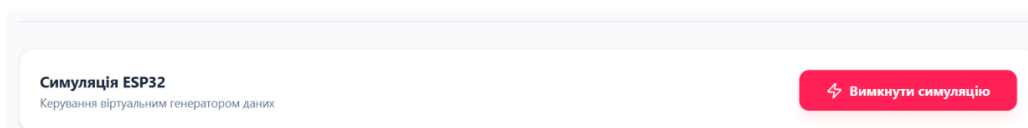


Рисунок 4.15 – Кнопка ввімкнення/вимкнення симуляції девайсу ESP32

Інтерфейс панелі має модульну структуру і включає наступні ключові блоки:

- У верхньому меню знаходиться логотип Smart Energy, вкладка для відстеження даних і компонент для відстеження статусу підключення системи моніторингу (онлайн/офлайн) – на рисунку 4.11.
- Нижче знаходяться **ключові метрики**: 4 окремі картки відображають поточну «Сонячну генерацію» (у Вт), «Загальне споживання енергії» та «Заряд батареї» у відсотках, ефективність MPPT-контролера. Ці картки знаходяться на рисунку 4.11.
- Блок «Статус бази даних» інформує користувача про час останнього успішного запису та підтверджує процес збереження інформації («Йде запис даних»). Вид блоку зображений на рисунку 4.12.

- Головний порівняльний аналіз: «Генерація vs Споживання енергії» зображений на рисунку 4.13. На одній часовій шкалі відображаються дві лінії: сонячна генерація (помаранчевий колір) та споживання (червоний колір). Це дозволяє користувачеві миттєво оцінити профіцит або дефіцит енергії.
- Індикація напруги: круговий стрілочний індикатор, який акцентує увагу на «Напрузі в системі» (наприклад, 26.5 В) та поточному загальному значенні струму (рис. 4.13).
- Сонячні панелі, МРРТ та Споживання: для кожного вузла побудовано окремі графіки «В & А» (Вольти та Амperi). Використання двох осей Y дозволяє одночасно відстежувати коливання напруги та сили струму на одному полі. Графіки підтримують спливаючі підказки, що відображають точні значення в обраній точці часової шкали; відображення параметрів графіків може бути відключене при натисканні на параметр. Вид графіків зображено на рисунку 4.14.
- Розділ «Пристрої в системі» надає інформацію про кінцевих споживачів: їх потужність, силу струму і напругу (рис. 4.14, нижня частина).
- Останнім розділом є режим «симуляція ESP32», в якому користувач може запустити або зупинити симульований програмно контролер ESP32 (рис. 4.15).

4.4.2 Панелі аналітики

Панелі аналітики є розширеними версіями діаграм на головній сторінці. З кожної діаграми можна перейти у відповідну розширену її версію на окремій сторінці аналітики. Приклад панелі для діаграми «МРРТ В & А» зображено на рисунку 4.17.

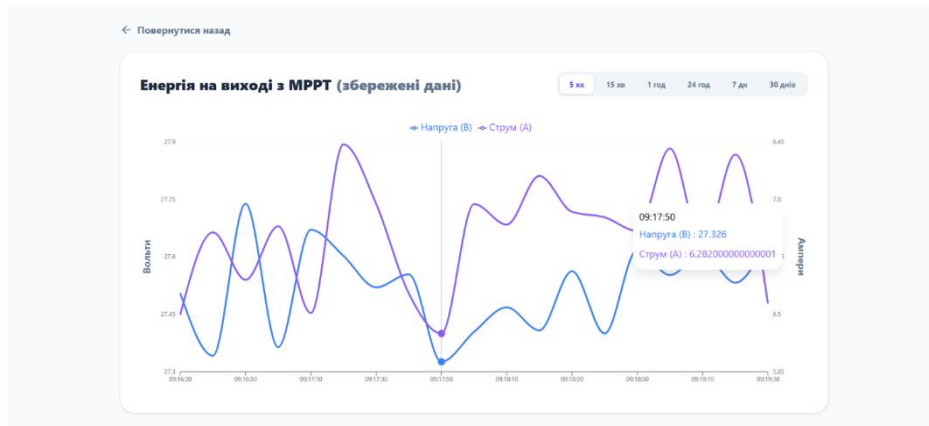


Рисунок 4.17 – Вигляд розширеної панелі для перегляду даних МРРТ-контролера

Розширені діаграми дозволяють переглядати дані впродовж останніх 5хв, 15хв, 1 год, 24год, 7 днів. Компоненти діаграм також можна вимикати (щоб не показувати їх), або знову вмикати. Якщо в базі даних немає збережених даних на часовий проміжок, користувач побачить відповідне повідомлення. Якщо не вдалося отримати дані з бази даних, користувач побачить це повідомлення також.

4.4.3 Панель порівняння параметрів

Для проведення глибокого ретроспективного аналізу роботи енергосистеми було розроблено спеціалізований розділ «Огляд і порівняння параметрів», зображений на рисунку 4.18. На відміну від головної панелі, яка фокусується на поточному стані, ця сторінка призначена для виявлення закономірностей та кореляцій між різними показниками за обрані періоди часу.

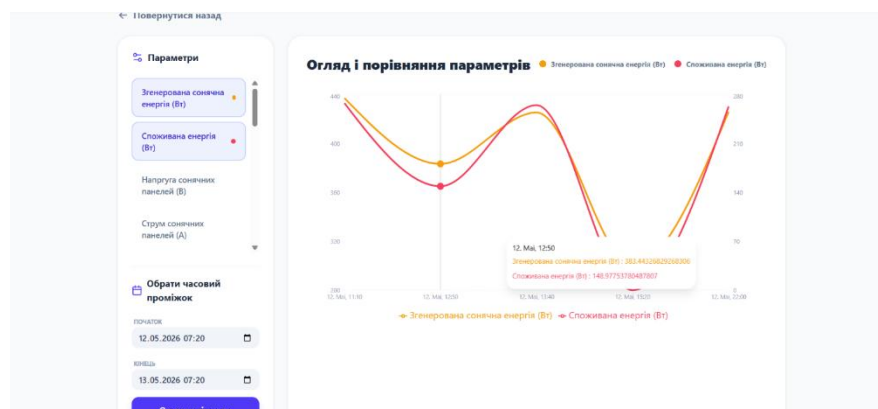


Рисунок 4.18 – Розділ «Огляд і порівняння параметрів»

Ліва частина інтерфейсу містить гнучку систему фільтрів, що дозволяє користувачеві самостійно формувати склад діаграми. Користувач може обрати будь-яку комбінацію з доступних метрик. Також знизу користувач повинен обрати проміжок часу, з якого він хоче обрати дані для перегляду. Якщо користувач обрере дату, яка ще не настала, або якщо дата початку буде пізніше дати кінця, він не зможе оновити діаграму і побачить підпис знизу про цю проблему.

Центральну частину сторінки займає розгорнута часова діаграма, побудована за допомогою бібліотеки Recharts. Графік дозволяє наочно побачити взаємозв'язок між параметрами та дослідити їх (разом або індивідуально). Кожен параметр має свій унікальний колір для зручності.

4.5 Тестування розробленої системи

Основне завдання системи моніторингу – збирати і демонструвати дані користувачу в зручному вигляді. Тому в такому випадку найкраще підходить саме ручне тестування, адже unit-тести не зможуть визначити, наскільки правильно відображається певний параметр чи графік.

4.5.1 Поведінка системи в різних ситуаціях

На рисунку 4.19 зображено вигляд сповіщення про базу даних, коли система довго була вимкнена.

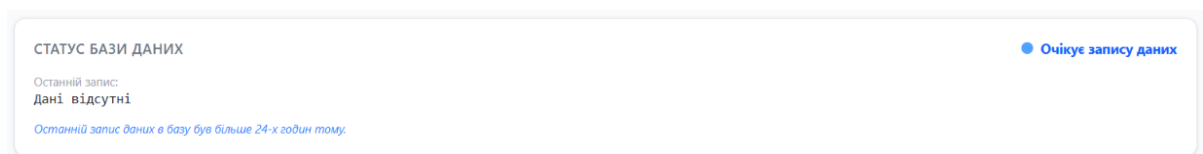


Рисунок 4.19 – Сповіщення про довгу відсутність записів

На рисунку 4.20 зображено вигляд системи користувача, коли система повністю вимкнена. Користувач може побачити і зрозуміти стан системи.

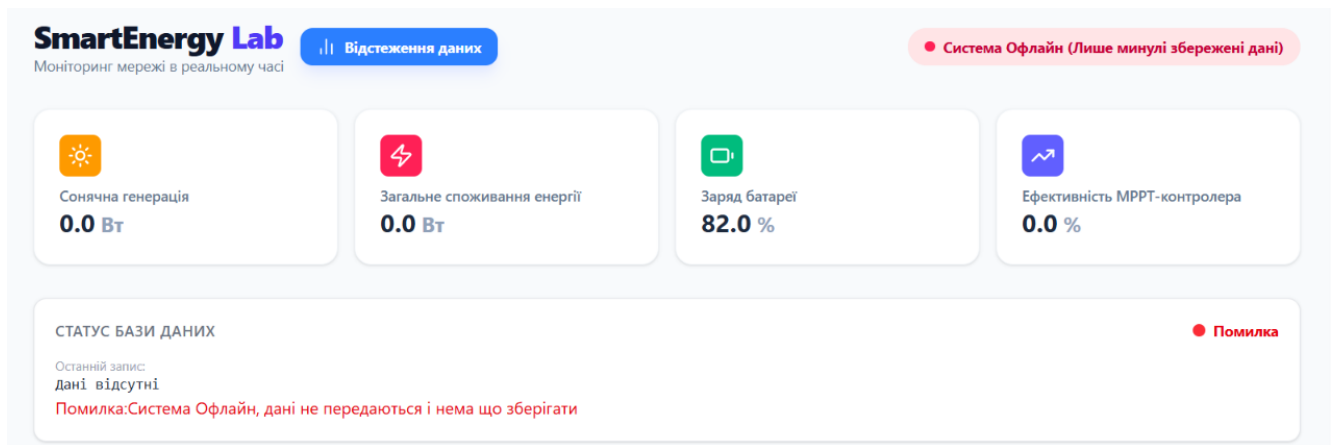


Рисунок 4.20 – Інтерфейс користувача, коли система офлайн

Якщо проблема стається на сервері і не можна отримати дані, користувач буде про це проінформований (рис. 4.21)

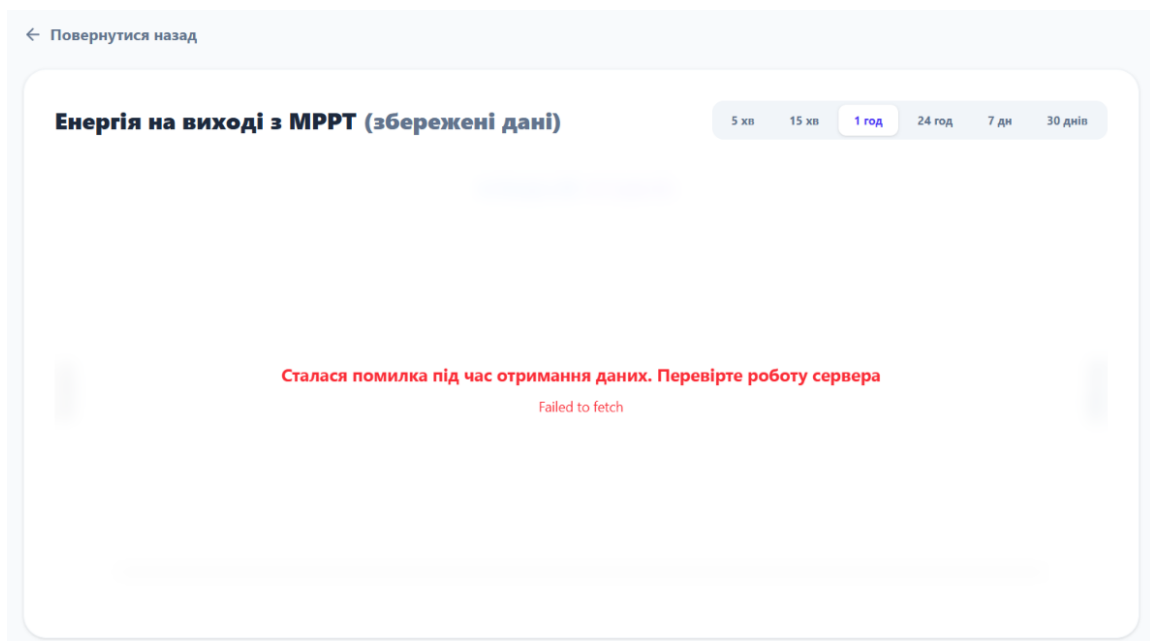


Рисунок 4.21 – Проблема з отриманням даних (на сервері вимкнено модуль history-api)

Так як серверні модулі ingestor, history-api не пов'язані між собою, відмова одного з них не означає повну відмову системи. При усуненні проблем і перезавантаженні сторінки, сайт працює далі у нормальному режимі.

4.5.2 Додавання історичних даних

Для того, щоб перевірити роботу компонентів «панель аналітики» і «панель порівняння параметрів» було створено спеціальний скрипт `influxDBDateGenerator.ts` для того, щоб генерувати великі обсяги даних (рис. 4.22), які потім вносяться ручним способом в базу даних (рис. 4.23), перевіряється їх видимість і відображення в системі (рис. 4.24). Коли ці дані стають непотрібні, їх можна видалити з бази даних (рис. 4.25).

```
1055 , ,5,2026-05-07T20:50:05.627Z,2026-05-08T20:50:05.627Z,2026-05-07T20:59:15.627Z,175.22975467207695,dev1_p,sensor-data,esp32
1056 , ,6,2026-05-07T20:50:05.627Z,2026-05-08T20:50:05.627Z,2026-05-07T20:59:15.627Z,230.00000000000000,dev1_v,sensor-data,esp32
1057 , ,7,2026-05-07T20:50:05.627Z,2026-05-08T20:50:05.627Z,2026-05-07T20:59:15.627Z,1.50609232340919,dev2_i,sensor-data,esp32
1058 , ,8,2026-05-07T20:50:05.627Z,2026-05-08T20:50:05.627Z,2026-05-07T20:59:15.627Z,362.99787798764714,dev2_p,sensor-data,esp32
1059 , ,9,2026-05-07T20:50:05.627Z,2026-05-08T20:50:05.627Z,2026-05-07T20:59:15.627Z,230.00000000000000,dev2_v,sensor-data,esp32
1060 , ,10,2026-05-07T20:50:05.627Z,2026-05-08T20:50:05.627Z,2026-05-07T20:59:15.627Z,1.96761928747569,dev3_i,sensor-data,esp32
1061 , ,11,2026-05-07T20:50:05.627Z,2026-05-08T20:50:05.627Z,2026-05-07T20:59:15.627Z,11.24270550112809,dev3_p,sensor-data,esp32
1062 , ,12,2026-05-07T20:50:05.627Z,2026-05-08T20:50:05.627Z,2026-05-07T20:59:15.627Z,5.00000000000000,dev3_v,sensor-data,esp32
1063 , ,13,2026-05-07T20:50:05.627Z,2026-05-08T20:50:05.627Z,2026-05-07T20:59:15.627Z,10.90596396649832,discharge_i,sensor-data,esp32
1064 , ,14,2026-05-07T20:50:05.627Z,2026-05-08T20:50:05.627Z,2026-05-07T20:59:15.627Z,284.72970300306548,discharge_p,sensor-data,esp32
1065 , ,15,2026-05-07T20:50:05.627Z,2026-05-08T20:50:05.627Z,2026-05-07T20:59:15.627Z,27.51406256849442,discharge_v,sensor-data,esp32
1066 , ,16,2026-05-07T20:50:05.627Z,2026-05-08T20:50:05.627Z,2026-05-07T20:59:15.627Z,0.00000000000000,solar_i,sensor-data,esp32
1067 , ,17,2026-05-07T20:50:05.627Z,2026-05-08T20:50:05.627Z,2026-05-07T20:59:15.627Z,0.00000000000000,solar_p,sensor-data,esp32
1068 , ,18,2026-05-07T20:50:05.627Z,2026-05-08T20:50:05.627Z,2026-05-07T20:59:15.627Z,0.00000000000000,solar_v,sensor-data,esp32
1069 , ,0,2026-05-07T20:50:05.627Z,2026-05-08T20:50:05.627Z,2026-05-07T20:59:25.627Z,84.42999999999971,battery_charge,sensor-data,esp32
```

Рисунок 4.22 – Приклад згенерованих даних

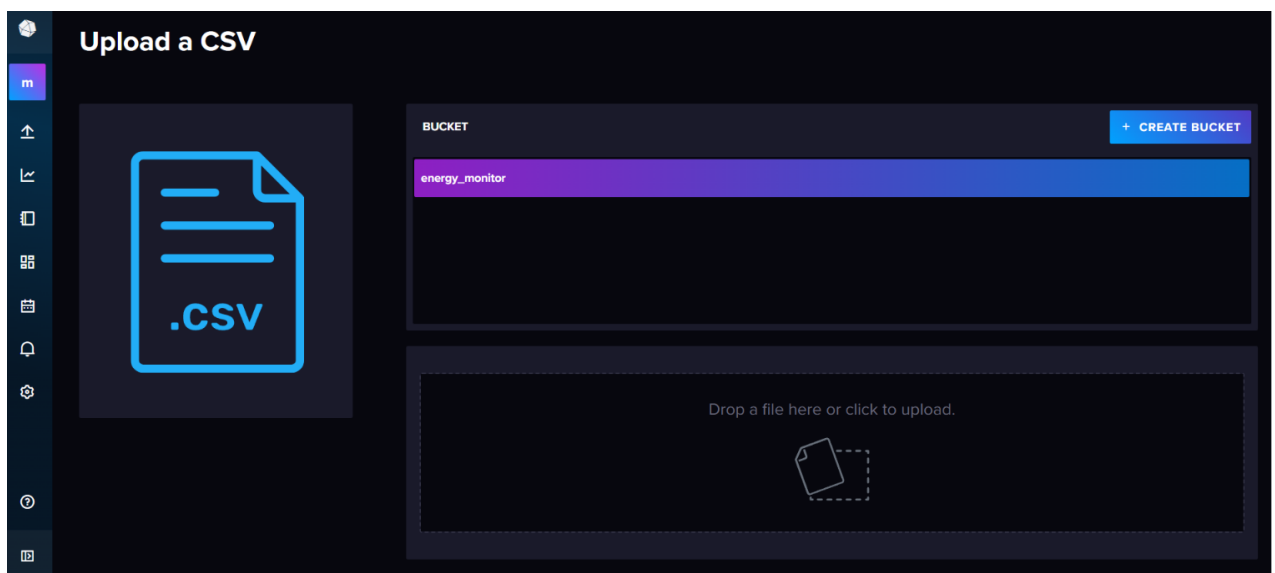


Рисунок 4.23 – Внесення згенерованих даних в influxDB

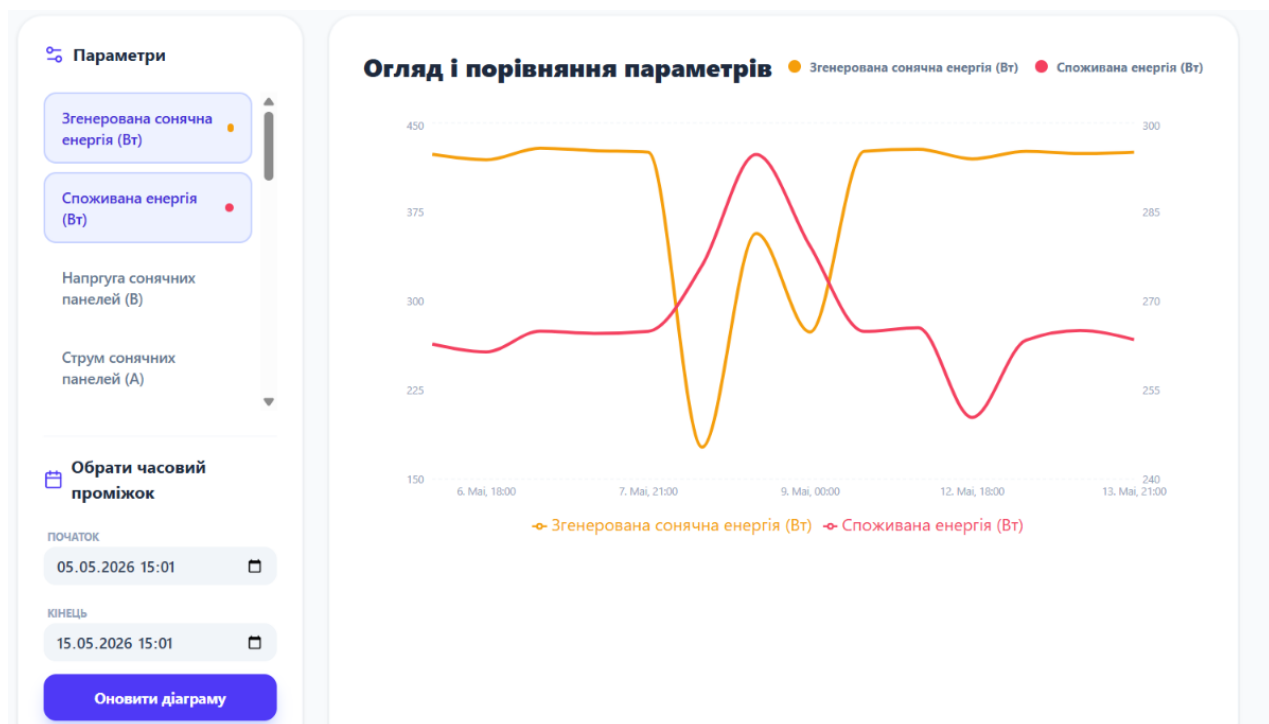


Рисунок 4.24 – Перегляд внесених даних

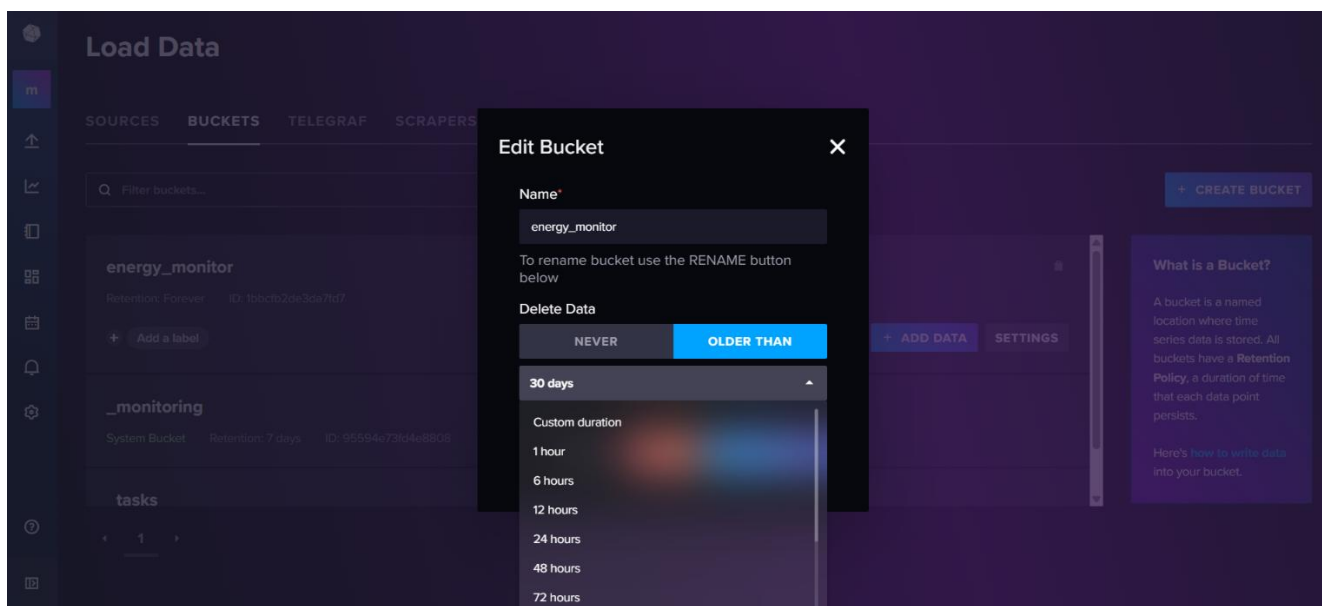


Рисунок 4.25 – Видалення внесених тестових даних

Такі методи тестування були обрані з причини їх швидкості, легкості проведення, доступності і наочності.

Висновки до розділу 4

У четвертому розділі представлено детальний опис практичної реалізації розробленого програмного продукту. У підрозділі 4.1 описано загальний концепт і архітектуру програмного забезпечення. У розділі 4.2 описано програмування апаратної частини (контролера ESP32). У розділі 4.3 описується принцип роботи модулів бекенду, а в розділі 4.4 наведено і описано користувацький інтерфейс. В розділі 4.5 проведено тестування системи та перевірено її працездатність в нормальному, а також при виникненні неполадок.

5 РОБОТА КОРИСТУВАЧА З ПРОГРАМНИМ ПРОДУКТОМ

Розроблений програмно-апаратний комплекс для моніторингу процесів генерації та споживання електричної енергії в умовах лабораторії «SmartEnergy Lab» функціонує як автономне кібер-фізичне рішення. Взаємодія оператора, дослідника або адміністратора з розробленою системою здійснюється через веб-орієнтований графічний інтерфейс користувача. У цьому розділі наведено детальні інструкції щодо розгортання, конфігурації, повсякденної експлуатації системи та проведення аналізу накопичених часових рядів телеметрії.

5.1 Порядок розгортання та запуску програмного комплексу

Для запуску програмного комплексу необхідно:

1. Завантажити вихідний програмний код проєкту.
2. Відкрити проєкт локально, за необхідності змінити певні конфігурації в файлі `docker-compose.yml` під свої потреби (напр. змінити зовнішні порти чи паролі)
3. Зібрати і запустити проєкт командою: `docker-compose up -d`
4. Відкрити локальний хост і перейти за вказаним в `docker-compose.yml` портом фронтенду (за стандартом це порт 5173).

Якщо не має можливості скачати серверну частину, її можна підключити в `docker-compose.yml` як окремі зовнішні образи. Ці образи можна знайти на DockerHub за відповідними назвами:

- `vicmony/monitoring-monastyrnyi-history-api`
- `vicmony/monitoring-monastyrnyi-ingestor`
- `vicmony/monitoring-monastyrnyi-simulators`

При цьому `simulators` це додатковий скрипт, який симулює ESP32. Його не треба встановлювати, якщо для системи не потрібна симуляція передачі даних. Якщо є плата ESP32, то її треба запрограмувати за допомогою коду `esp32.ino`, який міститься в директорії «`firmware`» цього проекту.

5.2 Експлуатація головної панелі

Після відкриття вебзастосунку користувач потрапляє на головну панель (Dashboard), яка працює в режимі реального часу. Інтерфейс автоматично адаптується до моніторів персональних комп'ютерів або мобільних пристроїв.

Можливості користувача зображені на рисунку 5.1.

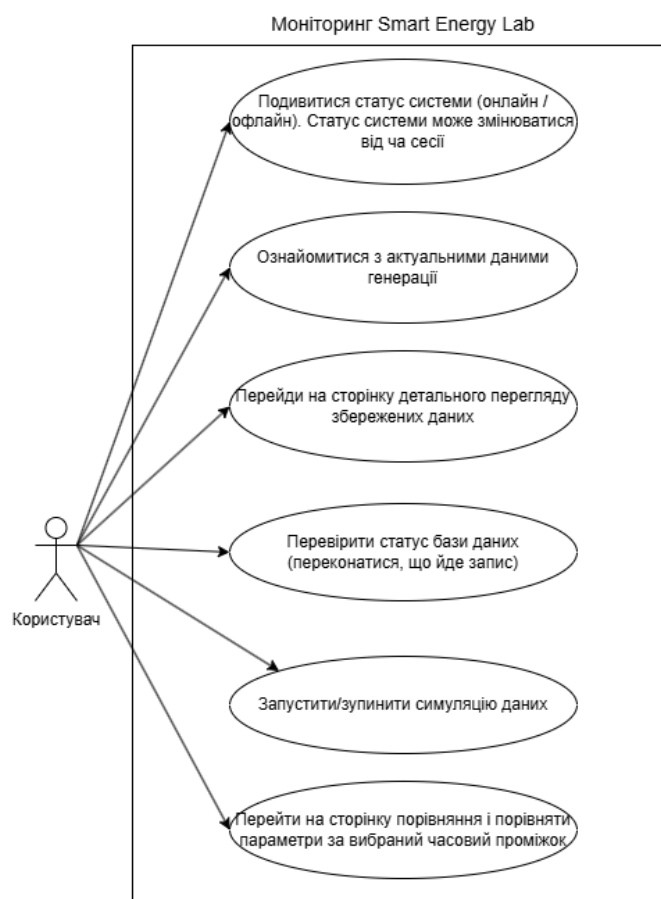


Рисунок 5.1 – Дії користувача в системі

Оновлення даних на екрані відбувається асинхронно без перезавантаження сторінки браузера, що забезпечується постійною обробкою потоку MQTT-

повідомлень від контролера. Система є інтуїтивно зрозумілою, всі графіки і компоненти мають імена, які їх чітко і зрозуміло описують.

Усі графічні елементи є чутливими до дій користувача: при наведенні курсору миші на тренд з'являється плаваюче вікно (Tooltip) із точними фізичними значеннями та часовою міткою, а клік по легенді дозволяє тимчасово приховати обрану криву для детального вивчення інших параметрів.

Якщо в системі щось йде не так (наприклад втрачається зв'язок з ESP32), то користувач бачить це в системі.

5.2 Експлуатація панелі порівняння

Для проведення ретроспективного аналізу енергетичних процесів, або порівняння ефективності обладнання в різні календарні періоди призначена вкладка «Огляд і порівняння параметрів».

Порядок роботи з цим модулем складається з таких кроків:

- **Вибір досліджуваних величин:** У лівій бічній панелі користувач за допомогою прапорців (checkbox) відзначає метрики, які необхідно зіставити на одному екрані (наприклад, одночасно обирає «Напруга сонячних панелей» та «Струм сонячних панелей»). Одночасно можна обрати до 2-х параметрів.

- **Встановлення часових меж:** За допомогою інтегрованого календаря вказується початкова та кінцева дата й час досліджуваного історичного інтервалу. Якщо часовий проміжок некоректний, користувач не зможе оновити діаграму і побачить сповіщення про причину.

Після натискання кнопки «Оновити діаграму» система генерує комплексний аналітичний графік порівняння обраних параметрів за заданий проміжок часу.

Висновки до розділу 5

У п'ятому розділі було детально розглянуто практичні аспекти взаємодії користувача з розробленим програмним забезпеченням системи моніторингу SmartEnergy Lab. Обґрунтовано та описано процедуру контейнеризації серверної інфраструктури за допомогою Docker, що мінімізує час на розгортання та налаштування проєкту на новому обладнанні. Наведено покрокові інструкції користувача щодо навігації по інтерактивній головній панелі та проведення ретроспективного аналізу історичних даних.

ВИСНОВКИ

У ході виконання роботи було спроектовано та програмно реалізовано комплексну кібер-фізичну систему моніторингу генерації та споживання електроенергії для лабораторії «Smart Energy Lab». Розроблена система відповідає всім поставленим вимогам щодо сумісності з розробками напрямку Smart Energy.

Було досліджено та описано апаратну побудову розумної мережі моніторингу, розглянуто необхідні і суміжні компоненти, такі як датчики струму і напруги, мікроконтролер, перетворювач струму та конвертер сигналів, для доповнення існуючої системи.

Реалізовано надійну і сучасну архітектуру збереження даних за допомогою бази даних часових рядів InfluxDB. Це дозволило оптимізувати збереження великих масивів метрик, що надходять у реальному часі і забезпечило високу швидкість виконання запитів при побудові історичних звітів та проведенні аналізу за тривалі періоди часу.

Було створено архітектуру збору і передачі інформації від пристроїв до системи моніторингу, досліджено протоколи передачі даних. Було запущено MQTT-брокер, налаштовані мережеві порти сервера, запрограмована плата esp32. Створено 2 серверних модуля для збереження і видачі даних.

Розроблено адаптивний сучасний інтерфейс користувача. За допомогою бібліотек візуалізовано складні енергетичні процеси, що значно спрощує контроль за станом системи. Користувач бачить в інтерфейсі все, що відбувається в системі.

Також було описано, як встановити і запустити цю систему, а також написані експлуатаційні рекомендації.

Підсумовуючи, розроблений програмно-апаратний комплекс є завершеним рішенням, яке дозволяє не лише відстежувати поточні параметри енергосистеми, а й аналізувати її ефективність, та помічати неполадки в системі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бацала Я. В., Гладь І. В., Николин У. М. Аналіз показників якості електроенергії сонячної електростанції. Розвідка та розробка нафтових і газових родовищ. 2013. № 4(49). С. 81–89.
2. Денисюк С. П., Базюк Т. М. Оптимальний відбір потужності в системах електропостачання. Енергетика: економіка, технології, екологія. 2013. № 2. С. 50–58.
3. Деревянко Д. Г. Особливості оцінки запасу стійкості локальних систем з різнорідними джерелами генерації. Енергетика: економіка, технології, екологія. 2013. Спецвипуск. С. 15–18.
4. Праховник А. В. Від управління електроспоживанням до енергетики сталого розвитку. Вісник Національного технічного університету України «Київський політехнічний інститут». Серія «Гірництво». 2010. Вип. 19. С. 108–119.
5. Технологія забезпечення живучості територіально розподілених інформаційних комп'ютерних систем в єдиному інформаційному просторі / А. Г. Додонов та ін. Реєстрація, зберігання і обробка даних. 2024. Т. 26, № 1. С. 122–143.
6. Залужний В. Ф. Показники ефективності (якості) комплексної системи забезпечення живучості розподілених автоматизованих систем організаційного управління силами та засобами. Реєстрація, зберігання і обробка даних. 2025. Т. 27, № 2. С. 42–50.
7. Ковальчук О. А., Пилипенко Ю. М. Система управління сонячними панелями. Технології та дизайн. 2020. № 1. С. 1–9.
8. Solar Data Loggers. Solar Power World. 2013. URL: <http://www.solarpowerworldonline.com/2013/11/> (дата звернення: 06.06.2026).
9. Pokalekar S., Patil R. Development and implementation of Solar Panel based power management system. SSRG International Journal of Mechanical Engineering. 2017. P. 82–84.

10. Liptak B. G., Eren H. Process Software and Digital Networks. Instrument Engineers' Handbook. Vol. 3. 3rd ed. CRC Press, 2018. 1134 p.
11. Umoh V. B., Obot J. J., Ekpe U. M. Development Of A Smart Solar Energy Management System. International Journal of Advanced Research and Publications. 2019. Vol. 3, iss. 8. P. 1–5.
12. Vishwakarma V. K., Srivastava S., Mishra V. Strategies of battery management systems in electric vehicles: a review. International Journal of Modelling and Simulation. 2025. P. 1–31.
13. Ahmad W., Hasan O., Pervez U., Qadir J. Reliability modeling and analysis of communication networks. Journal of Network and Computer Applications. 2017. Vol. 100. P. 191–215.

ДОДАТОК А Лістинг розробленої системи

backend/history-api/index.js:

```
import express from 'express';
import cors from 'cors';
import { InfluxDB } from '@influxdata/influxdb-client';
import * as dotenv from 'dotenv';

dotenv.config(); // Беремо секретні дані з .env

const app = express();
app.use(cors());
app.use(express.json());

const {
  INFLUX_URL,
  INFLUX_TOKEN,
  INFLUX_ORG,
  INFLUX_BUCKET,
  PORT = 3000,
  DATA_SAVE_TOPIC="sensor-data"
} = process.env;

const influxDB = new InfluxDB({ url: INFLUX_URL!, token: INFLUX_TOKEN! });
const queryApi = influxDB.getQueryApi(INFLUX_ORG!);

// Обробник запитів на отримання історичних даних
app.get('/api/history', async (req, res) => {
  console.log(`Incoming request: ${req.url}`);
  try {
    // параметри запитів до influxDB
    const range = (req.query.range as string) || '1h';
    const window = req.query.window as string;
    const offset = req.query.offset as string;
    const start = req.query.start as string;
    const end = req.query.end as string;

    let query = "";
    if (start && end) {
      console.log("START&END category")
      const windowClause = window ? `> aggregateWindow(every: ${window}, fn: mean, createEmpty: false ${offset ?
", offset: "+ offset:""})` : "";
      query = `
        from(bucket: "${INFLUX_BUCKET}")
          > range(start: time(v: "${start}"), stop: time(v: "${end}"))
          > filter(fn: (r) => r["_measurement"] === "${DATA_SAVE_TOPIC}")
          ${windowClause}
          > pivot(rowKey:["_time"], columnKey: ["_field"], valueColumn: "_value")
          > drop(columns: ["_start", "_stop", "_measurement", "deviceId"])
          > sort(columns: ["_time"])
      `;
    } else {
      console.log("ELSE category")
      const startRange = range === 'all' ? '0' : `-${range}`;
      const windowClause = window ? `> aggregateWindow(every: ${window}, fn: mean, createEmpty: false ${offset ?
", offset: "+ offset:""})` : "";
      query = `
        from(bucket: "${INFLUX_BUCKET}")
          > range(start: ${startRange})
```

```

    |> filter(fn: (r) => r["_measurement"] == "${DATA_SAVE_TOPIC}")
    ${windowClause}
    |> pivot(rowKey:["_time"], columnKey: ["_field"], valueColumn: "_value")
    |> drop(columns: ["_start", "_stop", "_measurement", "deviceId"])
    |> sort(columns: ["_time"])
  `;
}

const result: any[] = [];
let isResponded = false;

queryApi.queryRows(query, {
  next(row, tableMeta) {
    result.push(tableMeta.toObject(row));
  },
  error(error) {
    console.error("Query error:", error);
    if (!isResponded) {
      isResponded = true;
      res.status(500).json({ error: 'Query failed', details: error.message });
    }
  },
  complete() {
    // Переназиваю _time на time для сумісності з фронтендом (згідно з старндартами розробки щодо _полів)
    const formatted = result.map(r => {
      const { _time, ...rest } = r;
      return {
        time: _time,
        ...rest
      };
    });
    if (!isResponded) {
      isResponded = true;
      res.json(formatted);
    }
  },
});
} catch (error: any) {
  console.error("Internal error:", error);
  res.status(500).json({ error: 'Internal server error', details: error.message });
}
});

// Endpoint to check ingestor status
app.get('/api/status', async (req, res) => {
  try {
    const query = `
      from(bucket: "${INFLUX_BUCKET}")
      |> range(start: -24h)
      |> filter(fn: (r) => r["_measurement"] == "${DATA_SAVE_TOPIC}")
      |> last()
    `;

    let lastRecord: any = null;
    queryApi.queryRows(query, {
      next(row, tableMeta) {
        lastRecord = tableMeta.toObject(row);
      },
      error(error) {
        console.error("Status check error:", error);
      }
    });
  }
});

```

```

        res.status(500).json({ status: 'error', errorDetails: "Помилка отримання даних з бази даних. Перевірте базу
даних!" });
    },
    complete() {
        if (lastRecord) {
            const lastTime = new Date(lastRecord._time).getTime();
            const now = Date.now();
            const diff = (now - lastTime) / 1000; // seconds

            // If data was received in the last 70 seconds, it's active
            const isActive = diff <= 70;

            res.json({
                status: isActive ? 'active' : 'waiting',
                lastRecordTime: lastRecord._time,
                secondsSinceLastRecord: diff
            });
        } else {
            res.json({
                status: 'waiting',
                lastRecordTime: null,
                secondsSinceLastRecord: null,
                message: 'Останній запис даних в базу був більше 24-х годин тому.'
            });
        }
    },
    });
} catch (error: any) {
    console.error("Internal error in status check:", error);
    res.status(500).json({ error: 'Internal server error', errorDetails: "Помилка виконання запиту до сервера!" });
}
});

// Запуск API
app.listen(PORT, () => {
    console.log(`History API listening on port ${PORT}`);
});

```

backend/ingestor/index.js:

```

import mqtt from "mqtt"
import { InfluxDB, Point } from "@influxdata/influxdb-client"
import * as dotenv from "dotenv"

dotenv.config()
const {
    MQTT_URL,
    MQTT_USER,
    MQTT_PASSWORD,
    INFLUX_URL,
    INFLUX_TOKEN,
    INFLUX_ORG,
    INFLUX_BUCKET,
    DATA_SAVE_TOPIC="sensor-data"
} = process.env

const clientId = "INGESTOR"

// ініціалізація клієнта
const influxDB = new InfluxDB({

```

```

    url: INFLUX_URL!,
    token: INFLUX_TOKEN!
  })

const writeApi = influxDB.getWriteApi(INFLUX_ORG!, INFLUX_BUCKET!, "ms")
const mqttClient = mqtt.connect(MQTT_URL!, {
  clientId: clientId,
  username: MQTT_USER,
  password: MQTT_PASSWORD,
  clean: true // Брокер видалятиме підписки та незавершені повідомлення при відключенні //!
})

// Підключення до MQTT
mqttClient.on("connect", () => {
  console.log("Ingestor: підключено до MQTT брокера!")
  mqttClient.subscribe(
    "sensor/data",
    err => {
      if (!err) {
        console.log("Ін'єктор підписався на sensor/data (MQTT)")
      }
    }
  )
})

// Обробка повідомлень
mqttClient.on("message", (topic, message) => {
  const payload = message.toString()
  if (topic === "sensor/data") {
    try {
      const data = JSON.parse(payload)

      if(data.status === "offline"){
        console.log("Контролер не в мережі (офлайн)")
        return
      }
      const point = new Point(DATA_SAVE_TOPIC)
        .tag("deviceId", "esp32")
        .floatField("solar_v", data.solar?.v || 0)
        .floatField("solar_i", data.solar?.i || 0)
        .floatField("charge_v", data.charge?.v || 0)
        .floatField("charge_i", data.charge?.i || 0)
        .floatField("discharge_v", data.discharge?.v || 0)
        .floatField("discharge_i", data.discharge?.i || 0)
        .floatField("solar_p", (data.solar?.v || 0) * (data.solar?.i || 0))
        .floatField("charge_p", (data.charge?.v || 0) * (data.charge?.i || 0))
        .floatField("discharge_p", (data.discharge?.v || 0) * (data.discharge?.i || 0))
        .floatField("battery_charge", data.battery?.charge || 0)
        .timestamp(new Date(data.timestamp))

      // Write device data if needed, or just specific device fields
      if (data.devices) {
        Object.keys(data.devices).forEach(devId => {
          point.floatField(`${devId}_v`, data.devices[devId].v);
          point.floatField(`${devId}_i`, data.devices[devId].i);
          point.floatField(`${devId}_p`, data.devices[devId].v * data.devices[devId].i);
        });
      }

      // Збереження даних в InfluxDB
    } catch (err) {
      console.error("Error parsing message: ", err)
    }
  }
})

```

```

writeApi.writePoint(point)

// Лог
console.log(new Date(data.timestamp).toLocaleString("uk-UA"), "Solar P:", (data.solar?.v * data.solar?.i ||
0).toFixed(1), "Load P:", (data.discharge?.v * data.discharge?.i || 0).toFixed(1))
} catch (error) {
  console.error("Помилка парсингу:", error instanceof Error ? error.message : error)
  console.error("Payload:", payload)
}
}
})

mqttClient.on("error", err => {
  console.error("MQTT Error: ", err.message)
})

// При перериванні програми (Ctrl+C) виконується наступне:
process.on("SIGINT", () => {
  console.log("Зупинка ін'єктора")
  writeApi.close()
  .then(() => {
    console.log("Запис в InfluxDB припинено, з'єднання завершено")
    process.exit(0)
  })
  .catch(err => {
    console.error("Проблема під час завершення з'єднання в InfluxDB", err)
    process.exit(1)
  })
})

```

firmware/esp32_data_simulator/ esp32_data_simulator.ino:

```

#include <WiFi.h>
#include <PubSubClient.h>
#include <NTPClient.h>
#include <WiFiUdp.h>

// Налаштування мережі та брокера
// ssid - Назва мережі WiFi
// password - Пароль мережі WiFi
// mqtt_broker - Адреса брокера
// client_id - Назва клієнта (цього пристрою) для брокера
const char* ssid = "Vodafone-7C3C"; // Назва мережі WiFi
const char* password = "GD3phtqJBcmQNnQN"; // Пароль мережі WiFi
const char* mqtt_broker = "192.168.0.100"; // Адреса брокера, наприклад, 192.168.1.100 (або сайт)
const char* client_id = "ESP32_PHYSICAL_DATA_SIMULATOR"; // Назва клієнта (цього пристрою) для брокера
const char* mqtt_username = "physical_esp32"; // ім'я для підключення до брокера
const char* mqtt_password = "physical_esp32_pass"; // пароль для підключення до брокера

WiFiClient espClient;
PubSubClient client(espClient);

unsigned long lastMsg = 0;
const long interval = 2000; // 2 секунди

WiFiUDP ntpUDP;
const int UTC_0_shift = 0;
NTPClient timeClient(ntpUDP, "pool.ntp.org", UTC_0_shift);

void setup_wifi() {

```

```

delay(10);
Serial.println();
Serial.print("Connecting to ");
Serial.println(ssid);

WiFi.begin(ssid, password);

while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
}

Serial.println("\nWiFi connected");
}

void reconnect() {
    while (!client.connected()) {
        Serial.print("Attempting MQTT connection...");

        // Налаштування Last Will and Testament (LWT)
        String lastWillPayload = "{\"status\": \"offline\"}";

        if (client.connect(client_id, mqtt_username, mqtt_password, "sensor/data", 1, true, lastWillPayload.c_str())) {
            Serial.println("connected");
            // client.publish("sensor/data", "{\"status\": \"online\"}", true);
        } else {
            Serial.print("failed, rc=");
            Serial.print(client.state());
            Serial.println(" try again in 5 seconds");
            delay(5000);
        }
    }
}

void setup() {
    Serial.begin(115200);
    setup_wifi();
    client.setServer(mqtt_broker, 1883);
    timeClient.begin();
}

void loop() {
    if (!client.connected()) {
        reconnect();
    }
    client.loop();
    timeClient.update();

    unsigned long now = millis();
    if (now - lastMsg > interval) {
        lastMsg = now;

        // Генерація даних (імітація ваших параметрів)
        float solarV = 150.0 + ((float)random(0, 100) / 100.0);
        float solarI = 2.0 + ((float)random(0, 150) / 100.0);

        float chargeV = 27.0 + ((float)random(0, 100) / 100.0);
        float chargeI = 5.0 + ((float)random(0, 500) / 100.0);

        float dischargeV = 26.0 + ((float)random(0, 100) / 100.0);

```

```

float dischargeI = 8.0 + ((float)random(0, 400) / 100.0);

float batteryCharge = 82.0 + ((float)random(0, 50) / 10.0);

unsigned long long timestampMs = ((unsigned long long) timeClient.getEpochTime()) * 1000;

String payload = "{";
payload += "\"status\": \"online\",";
payload += "\"timestamp\": " + String(timestampMs) + ",";

// Блок Solar
payload += "\"solar\": {" + "\"v\": " + String(solarV) + ", \"i\": " + String(solarI) + "},";

// Блок Charge
payload += "\"charge\": {" + "\"v\": " + String(chargeV) + ", \"i\": " + String(chargeI) + "},";

// Блок Discharge
payload += "\"discharge\": {" + "\"v\": " + String(dischargeV) + ", \"i\": " + String(dischargeI) + "},";

// Блок Battery
payload += "\"battery\": {" + "\"charge\": " + String(batteryCharge) + "},";

// Блок Devices (приклад з декількома пристроями)
payload += "\"devices\": {";
payload += "\"dev1\": {" + "\"v\": 230, \"i\": " + String(0.5 + (random(0, 50) / 100.0)) + "},";
payload += "\"dev2\": {" + "\"v\": 230, \"i\": " + String(1.2 + (random(0, 80) / 100.0)) + "},";
payload += "\"dev3\": {" + "\"v\": 0, \"i\": " + String(0) + "},";
payload += "},";

payload += "}";
client.publish("sensor/data", payload.c_str(), false);
Serial.println("Повний пакет відправлено:");
Serial.println(payload);

}
}

```

frontend/App.tsx:

```

import { useState, useEffect, useRef } from 'react';
import { BrowserRouter, Routes, Route } from 'react-router-dom';
import mqtt from 'mqtt';
import {
  Cpu, Smartphone, Refrigerator
} from 'lucide-react';
import type { HistoryPoint, SystemStatus, ConsumerDevice, IngestorStatus } from './types';
import AnalyticsPage from "./components/AnalyticsPage.tsx";
import ComparisonPage from "./components/ComparisonPage.tsx";
import { Dashboard } from './components/Dashboard.tsx';

let globalMqttClient: mqtt.MqttClient | null = null;

const MAX_CHART_POINTS = 50;

const App = () => {
  const [history, setHistory] = useState<HistoryPoint[]>([]);
  const [isMqttConnected, setIsMqttConnected] = useState(false);

```

```

const isMqttConnectedRef = useRef(isMqttConnected);
useEffect(() => {
  isMqttConnectedRef.current = isMqttConnected;
}, [isMqttConnected]);
const [ingestorInfo, setIngestorInfo] = useState<IngestorStatus | null>({
  status: 'waiting',
  lastRecordTime: null,
  secondsSinceLastRecord: null,
  message: 'Очікування на відповідь від бази даних'
});
const lastDataTime = useRef<number>(Date.now());

const [status, setStatus] = useState<SystemStatus>({
  solarPower: 0,
  chargePower: 0,
  dischargePower: 0,
  solarV: 0,
  solarA: 0,
  dischargeV: 0,
  dischargeA: 0,
  batteryCharge: 82,
  batteryTemp: 24.5,
  mpptEfficiency: 0,
});

const [devices, setDevices] = useState<ConsumerDevice[]>([
  { id: 'dev1', name: 'Прилад 1', consumption: 0, voltage: 0, current: 0, icon: Cpu, color: 'text-indigo-500' },
  { id: 'dev2', name: 'Прилад 2', consumption: 0, voltage: 0, current: 0, icon: Refrigerator, color: 'text-amber-500' },
  { id: 'dev3', name: 'Прилад 3', consumption: 0, voltage: 0, current: 0, icon: Smartphone, color: 'text-emerald-500' },
]);

const [isSimulatorOn, setIsSimulatorOn] = useState(true);
const [isSimulatorPending, setIsSimulatorPending] = useState(false);
const lastToggleTime = useRef<number>(0);
const waitingForFirstData = useRef<boolean>(false);

const formatTime = (date: Date) => date.toLocaleTimeString([], { hour: '2-digit', minute: '2-digit', second: '2-digit' });

useEffect(() => {
  fetch('http://localhost:3000/api/history')
    .then(res => res.json())
    .then(data => {
      const historyData = data.map((d: any) => ({
        timestamp: new Date(d.time).getTime(),
        time: formatTime(new Date(d.time)),
        solar: d.solar_p || 0,
        consumption: d.discharge_p || 0,
        solar_v: d.solar_v || 0,
        solar_i: d.solar_i || 0,
        charge_v: d.charge_v || 0,
        charge_i: d.charge_i || 0,
        discharge_v: d.discharge_v || 0,
        discharge_i: d.discharge_i || 0,
      })).sort((a: any, b: any) => a.timestamp - b.timestamp)
        .slice(-MAX_CHART_POINTS);

      setHistory(historyData);
    })
    .catch(err => console.error("History load failed", err));

```

```

// Підключення до MQTT сервера
if (!globalMqttClient) {
  globalMqttClient = mqtt.connect('ws://localhost:9001', {
    clientId: 'smart_ui_' + Math.random().toString(16).substring(2, 8),
    username: 'frontend',
    password: 'frontend_pass',
  });
}

const client = globalMqttClient;
client.on('connect', () => {
  client.subscribe(['sensor/data', 'simulator/status']);
});

const handleMessage = (topic: string, message: Buffer) => {
  try {
    const payloadStr = message.toString();

    if (topic === 'simulator/status') {
      const newState = payloadStr === 'ON';
      setIsSimulatorOn(newState);

      if (newState) {
        // When simulator turns ON, we might still want to wait for first data
        // before clearing the pending state, but let's see if we are in that mode.
        if (!waitingForFirstData.current) {
          setIsSimulatorPending(false);
        }
      } else {
        // When simulator turns OFF, we can clear pending immediately
        setIsSimulatorPending(false);
        waitingForFirstData.current = false;
      }
      return;
    }

    const payload = JSON.parse(payloadStr);

    if (topic === 'sensor/data') {
      if (payload.status === 'online') {
        setIsMqttConnected(true);
        if (waitingForFirstData.current) {
          waitingForFirstData.current = false;
          setIsSimulatorPending(false);
        }
      } else if (payload.status === 'offline') {
        setIsMqttConnected(false);
        setDevices(prev => prev.map(d => ({ ...d, voltage: 0, current: 0, consumption: 0 })));
        return;
      }
      // The check below is redundant now but kept for safety if status is undefined
      if (payload.status === undefined) {
        setIsMqttConnected(true);
      }
      lastDataTime.current = Date.now();

      const pS = (payload.solar?.v || 0) * (payload.solar?.i || 0);
      const pD = (payload.discharge?.v || 0) * (payload.discharge?.i || 0);
      const dataTimestamp = payload.timestamp ? new Date(payload.timestamp) : new Date();
    }
  }
};

```

```

setStatus(prev => ({
  ...prev,
  solarPower: pS,
  solarV: payload.solar?.v || 0,
  solarA: payload.solar?.i || 0,
  chargePower: (payload.charge?.v || 0) * (payload.charge?.i || 0),
  dischargePower: pD,
  dischargeV: payload.discharge?.v || 0,
  dischargeA: payload.discharge?.i || 0,
  batteryCharge: payload.battery?.charge || prev.batteryCharge,
  mpptEfficiency: pS > 0 ? (((payload.charge?.v || 0) * (payload.charge?.i || 0)) / pS) * 100 : 0,
}));

setHistory(prev => {
  const newPoint: HistoryPoint = {
    timestamp: dataTimestamp.getTime(),
    time: formatTime(dataTimestamp),
    solar: pS,
    consumption: pD,
    solar_v: payload.solar?.v || 0,
    solar_i: payload.solar?.i || 0,
    charge_v: payload.charge?.v || 0,
    charge_i: payload.charge?.i || 0,
    discharge_v: payload.discharge?.v || 0,
    discharge_i: payload.discharge?.i || 0
  };
  return [...prev, newPoint].sort((a, b) => a.timestamp - b.timestamp).slice(-MAX_CHART_POINTS);
});

if (payload.devices) {
  setDevices(prev => prev.map(d => {
    const devData = payload.devices[d.id];
    return devData ? { ...d, voltage: devData.v, current: devData.i, consumption: devData.v * devData.i } : d;
  }));
}
} catch (e) { console.error("Data error", e); }
};

client.on('message', handleMessage);

const intervalMqttMonitoring = setInterval(() => {
  if (Date.now() - lastDataTime.current > 5000) {
    setIsMqttConnected(false);
    setDevices(prev => prev.map(d => ({ ...d, voltage: 0, current: 0, consumption: 0 })));
  }
}, 2000);

const intervalIngestorMonitoring = setInterval(() => {
  // Check ingestor status
  if(!isMqttConnectedRef.current){
    setIngestorInfo({
      status: "error",
      errorDetails: "Система Офлайн, дані не передаються і нема що зберігати"
    })
    return
  }
  fetch('http://localhost:3000/api/status')
    .then(res => res.json())
    .then(data => setIngestorInfo(data))

```

```

        .catch(() => setIngestorInfo({
            status: "error",
            errorDetails: "Не вийшло встановити зв'язок з History-API модулем. Стан бази даних невідомий"
        }));
    }, 10000)

    return () => {
        client.removeListener('message', handleMessage);
        clearInterval(intervalMqttMonitoring);
        clearInterval(intervalIngestorMonitoring);
    };
}, []);

const toggleSimulator = () => {
    const now = Date.now();
    if (isSimulatorPending || (now - lastToggleTime.current < 3000)) {
        console.log("Toggle suppressed: pending or cooldown active");
        return;
    }

    lastToggleTime.current = now;
    setIsSimulatorPending(true);

    const nextState = isSimulatorOn ? 'OFF' : 'ON';
    if (nextState === 'ON') {
        waitingForFirstData.current = true;
    }

    globalMqttClient?.publish('simulator/control', nextState, { qos: 1 });
};

return (
    <BrowserRouter>
    <Routes>
    <Route path="/" element={
        <Dashboard
            history={history}
            status={status}
            devices={devices}
            isMqttConnected={isMqttConnected}
            ingestorInfo={ingestorInfo}
            isSimulatorOn={isSimulatorOn}
            isSimulatorPending={isSimulatorPending}
            onToggleSimulator={toggleSimulator}
        />
    } />
    <Route path="/analytics/:type" element={<AnalyticsPage />} />
    <Route path="/comparison" element={<ComparisonPage />} />
    </Routes>
</BrowserRouter>
);
};

export default App;

```

simulators/esp32.ts:

```

import mqtt from "mqtt"
import * as dotenv from "dotenv"

```

```

dotenv.config()
const {BROKER_URL, CLIENT_ID, SENSOR_DATA_TOPIC, MQTT_USER, MQTT_PASSWORD} = process.env

const PUBLISH_INTERVAL_MS = 2000

// Створення mqtt клієнта
const client = mqtt.connect(
  BROKER_URL!,
  {
    clientId: CLIENT_ID,
    username: MQTT_USER,
    password: MQTT_PASSWORD,
    clean: true,
    connectTimeout: 4000,
    will: {
      topic: SENSOR_DATA_TOPIC!,
      payload: Buffer.from(JSON.stringify({status: "offline"})),
      qos: 1,
      retain: true
    }
  }
)

client.on("connect", () => {
  console.log(`${CLIENT_ID} connected to MQTT broker!`)
  setInterval(() => {
    // Генерація даних
    const timestamp = Date.now()

    // Solar Generation
    const solarV = Number((150 + Math.random() * 10).toFixed(1));
    const solarI = Number((2.0 + Math.random() * 1.5).toFixed(2));

    // Battery / MPPT
    const chargeV = Number((27 + Math.random() * 1).toFixed(2));
    const chargeI = Number((5 + Math.random() * 5).toFixed(2));

    // Discharge / Main Load
    const dischargeV = Number((26 + Math.random() * 1).toFixed(2));
    const dischargeI = Number((8 + Math.random() * 4).toFixed(2));

    const batteryCharge = 82 + (Math.random() * 5); // %

    // Споживачі (тут для виду - 3 пристрої)
    const devices = {
      dev1: { v: 230, i: Number((0.5 + Math.random() * 0.5).toFixed(2)) },
      dev2: { v: 230, i: Number((1.2 + Math.random() * 0.8).toFixed(2)) },
      dev3: { v: 5, i: Number((1.0 + Math.random() * 2.0).toFixed(2)) }
    };

    const payload = {
      status: "online",
      timestamp: timestamp,
      solar: { v: solarV, i: solarI },
      charge: { v: chargeV, i: chargeI },
      discharge: { v: dischargeV, i: dischargeI },
      battery: { charge: batteryCharge },
      devices: devices
    }

    // Надсилання (публікування) даних

```

```

    client.publish(
      SENSOR_DATA_TOPIC!,
      JSON.stringify(payload),
      {qos:1}
    )

    // Логуювання:
    console.log(`Відправлено: Solar ${solarV} V, Load ${payload.discharge.i} A, Battery ${batteryCharge.toFixed(1)}%`)
  }, PUBLISH_INTERVAL_MS) // Надсилаємо дані кожні PUBLISH_INTERVAL_MS мілісекунд
})

client.on("error", err => {
  console.log("MQTT помилка:", err)
})

```

simulators/influxDBDataGenerator.ts:

```

// @ts-ignore
import * as fs from 'fs';

/**
 * Генерує дані сонячної генерації (0 вночі, синусоїда вдень)
 */
function getSolarPower(date: Date): number {
  const hour = date.getHours() + date.getMinutes() / 60;
  if (hour < 6 || hour > 20) return 0;
  const maxPower = 500;
  return maxPower * Math.sin((Math.PI * (hour - 6)) / 14) + (Math.random() * 10);
}

function generateFullInfluxCSV(startTime: Date, endTime: Date, intervalSeconds: number) {
  const deviceId = "esp32";
  const measurement = "sensor-data";

  // Заголовки файлу
  let csv = `#group,false,false,true,true,false,false,true,true,true\n`;
  csv += `#datatype,string,long,dateTime:RFC3339,dateTime:RFC3339,dateTime:RFC3339,double,string,string,string\n`;
  csv += `#default,mean,,,,,,,,,\n`;
  csv += `,result,table,_start,_stop,_time,_value,_field,_measurement,deviceId\n`;

  const sIso = startTime.toISOString();
  const eIso = endTime.toISOString();
  let curr = new Date(startTime);
  let battery = 85.0;

  const rows: string[] = [];

  while (curr <= endTime) {
    const t = curr.toISOString();
    const solarP = getSolarPower(curr);

    // Логіка заряду/розряду
    battery = solarP > 100 ? Math.min(100, battery + 0.01) : Math.max(10, battery - 0.01);

    // Список полів для CSV
    const fields = [
      { id: 0, f: "battery_charge", v: battery },
      { id: 1, f: "charge_i", v: solarP > 0 ? 5.0 + Math.random() : 0 },
      { id: 2, f: "charge_p", v: solarP * 0.9 },
      { id: 3, f: "charge_v", v: 27.5 + Math.random() * 0.5 },
    ];
  }
}

```

```

    { id: 4, f: "dev1_i",      v: 0.7 + Math.random() * 0.2 },
    { id: 5, f: "dev1_p",      v: 160 + Math.random() * 20 },
    { id: 6, f: "dev1_v",      v: 230 },
    { id: 7, f: "dev2_i",      v: 1.5 + Math.random() * 0.3 },
    { id: 8, f: "dev2_p",      v: 350 + Math.random() * 40 },
    { id: 9, f: "dev2_v",      v: 230 },
    { id: 10, f: "dev3_i",     v: 1.8 + Math.random() * 0.4 },
    { id: 11, f: "dev3_p",     v: 10 + Math.random() * 2 },
    { id: 12, f: "dev3_v",     v: 5 },
    { id: 13, f: "discharge_i", v: 10.5 + Math.random() },
    { id: 14, f: "discharge_p", v: 280 + Math.random() * 30 },
    { id: 15, f: "discharge_v", v: 26.8 + Math.random() },
    { id: 16, f: "solar_i",    v: solarP / 30 },
    { id: 17, f: "solar_p",    v: solarP },
    { id: 18, f: "solar_v",    v: solarP > 0 ? 35 + Math.random() * 5 : 0 }
  ];

  fields.forEach(field => {
    rows.push(`,${field.id},${sIso},${eIso},${t},${field.v.toFixed(14)},${field.f},${measurement},${deviceId}`);
  });

  curr = new Date(curr.getTime() + intervalSeconds * 1000);
}

csv += rows.join('\n');
fs.writeFileSync('./data_simulated/generated_query_03_2026.csv', csv);
console.log(`Створено файл з ${rows.length} записами.`);
}

// Генерація за останню добу з кроком 10 секунд
const end = new Date();
const start = new Date(end.getTime() - 24 * 60 * 60 * 1000);
generateFullInfluxCSV(start, end, 10);

```



Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці

СИСТЕМА МОНІТОРИНГУ ГЕНЕРАЦІЇ ТА СПОЖИВАННЯ ЕЛЕКТРОЕНЕРГІЇ В SMARTENERGY LAB

виконав: студент групи ТВ-з21 Монастирний Віктор Ігорович

керівник: д.т.н., професор кафедри ІПЗЕ Федорова Наталія Володимирівна

2026

Мета та постановка задачі

- **Мета:**
Розробити сучасну кіберфізичну систему моніторингу генерації і споживання електроенергії
- **Завдання:**
 - Проаналізувати існуючі подібні smart-системи
 - Визначити компоненти системи
 - Створити електричну схему системи
 - Створити програмне забезпечення
 - Провести тестування



Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Аналіз існуючих смарт-систем



Рис. 1- Loxone (монолітний розумний будинок)



Рис. 2 - Shelly (пристрої і ПЗ до них)



Рис. 3 - Victron Energy (промислові пристрої)

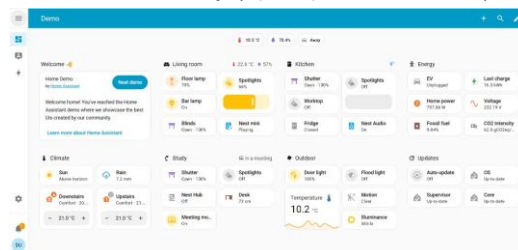


Рис. 4 - Home Assistant (проект з відкритим кодом)



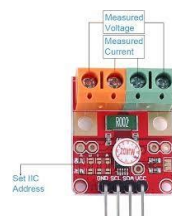
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

3

Компоненти системи



PZEM-017



INA-226



TTL to RS485 Модуль



LM2596 понижуючий модуль

Критерії вибірки:

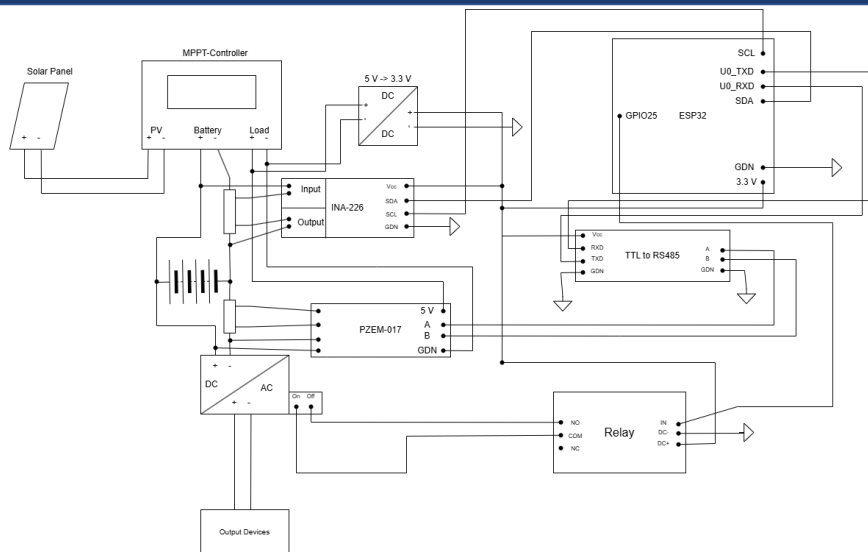
- Макс. вимірювальний струм
- Макс. вимірювана напруга
- Ціна
- Доступність



Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

4

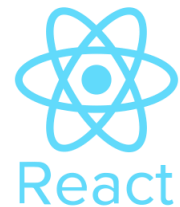
Функціональна схема електричної системи



Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

5

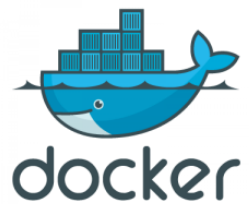
Використані інструменти



Tailwind CSS



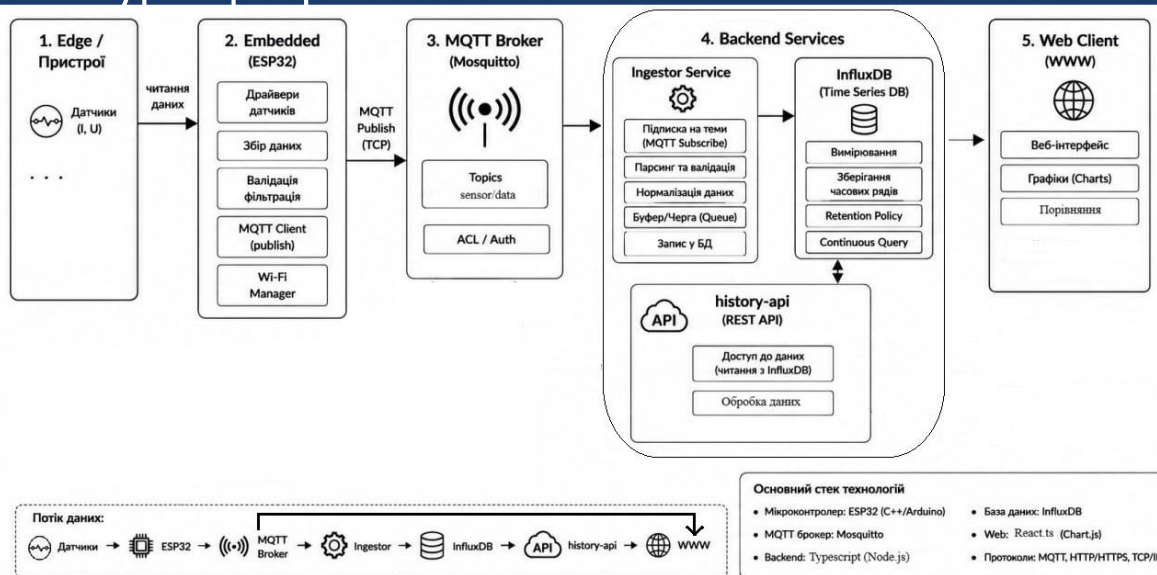
influxdb



Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

6

Архітектура програмного забезпечення



Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

7

Апаратна частина



```

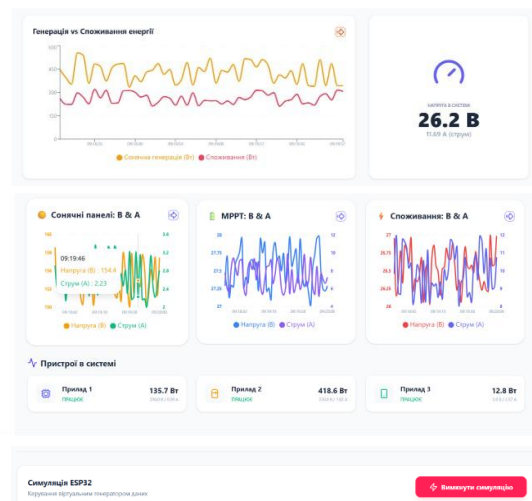
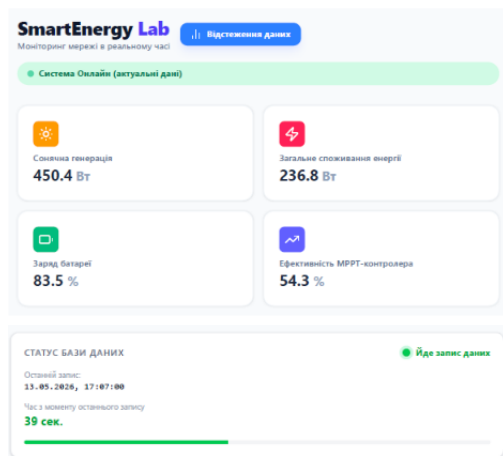
esp32_data_simulator.ino
44 void reconnect() {
45   while (!client.connected()) {
46     Serial.println("Attempting MQTT connection...");
47
48     // Attempt to connect using last will and topic (if set)
49     String lastWillTopic = "ESP32Data";
50     if (client.connect(lastWillTopic, "sensor/data", 1, true, lastWillTopic, 1)) {
51       Serial.println("connected");
52       // client.publish("sensor/data", "ESP32Data", true);
53     } else {
54       Serial.println("failed, rc=");
55       Serial.println(client.state());
56       Serial.println("try again in 5 seconds");
57       delay(5000);
58     }
59   }
60 }
61
62
63 void setup() {
64   Serial.begin(115200);
65   setup_wifi();
66   client.setServer(MQTT_BROKER, 1883);
67   client.setClientId("ESP32Data");
68 }
69
70 void loop() {
71   if (!client.connected()) {
72     reconnect();
73   }
74   client.loop();
75 }
    
```



Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

8

Головна сторінка програмного продукту



Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

9

Функціонал розширеного перегляду і порівняння



Рис. 5 – Меню перегляду збережених даних

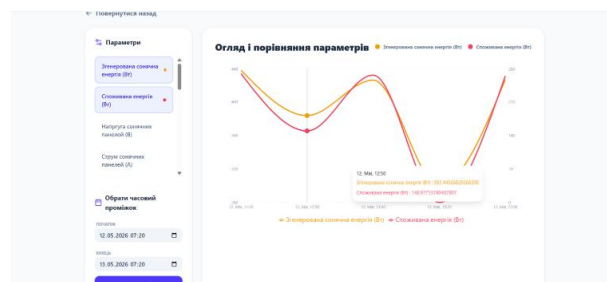


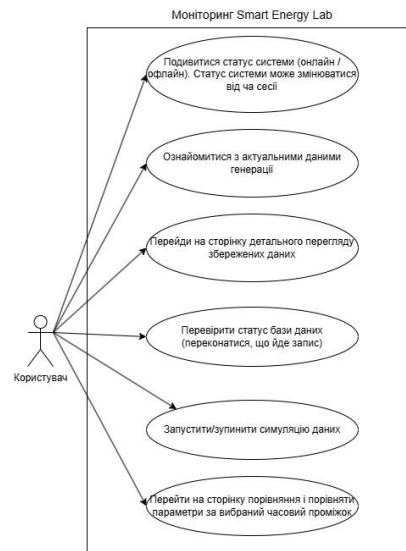
Рис. 6 – Меню відстеження і порівняння даних в проміжку часу



Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

10

Можливості користувача в системі



Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

11

Тестування системи

1. Тестування моніторингу: скрипт esp32.ts, який симулює роботу фізичних пристроїв і передає дані в систему моніторингу.

esp32.ts

2. Тестування бази даних: спецскрипт, який формує файл з необхідними даними, які потім імпортуються в БД і переглядаються в системі

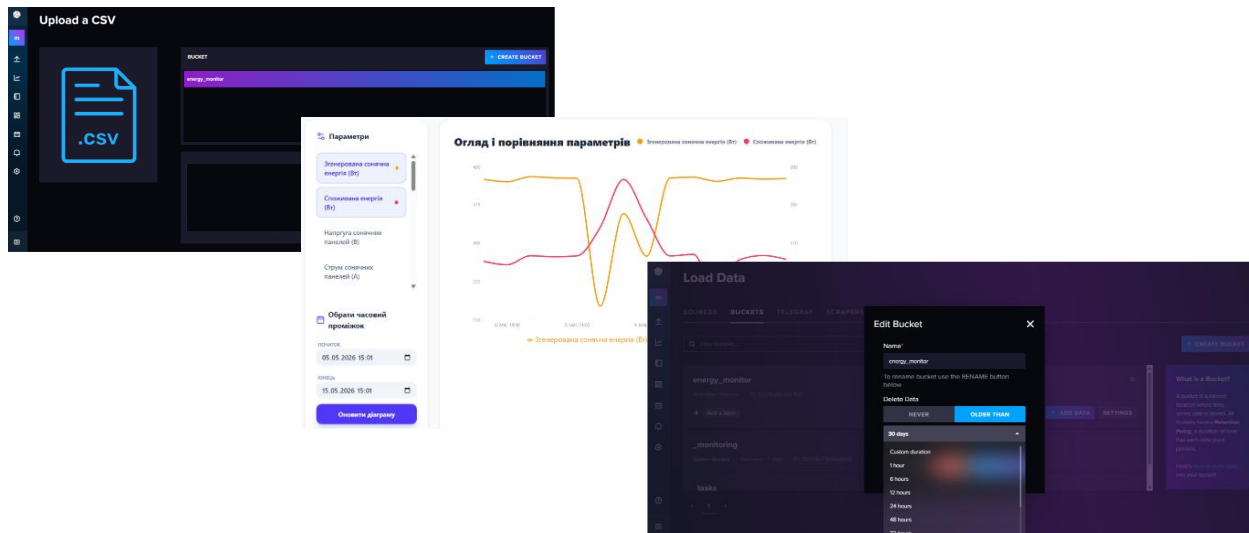
```
1 #group,false,false,true,true,false,false,true,true,true
2 #datatype,string,long,dateTime:RFC3339,dateTime:RFC3339,double,string,string,string
3 #defval,tuple,*****
4 #result,table,*,start_time,value,*,field,measurement,deviceid
5 ,,0,2026-05-07T20:58:05.627Z,2026-05-08T20:58:05.627Z,2026-05-07T20:58:05.627Z,84.98999999999999,battery_charge,sensor-data,esp32
6 ,,1,2026-05-07T20:58:05.627Z,2026-05-08T20:58:05.627Z,2026-05-07T20:58:05.627Z,0.0000000000000000,charge_i,sensor-data,esp32
7 ,,2,2026-05-07T20:58:05.627Z,2026-05-08T20:58:05.627Z,2026-05-07T20:58:05.627Z,0.0000000000000000,charge_p,sensor-data,esp32
8 ,,3,2026-05-07T20:58:05.627Z,2026-05-08T20:58:05.627Z,2026-05-07T20:58:05.627Z,27.51868025183266,charge_v,sensor-data,esp32
9 ,,4,2026-05-07T20:58:05.627Z,2026-05-08T20:58:05.627Z,2026-05-07T20:58:05.627Z,0.88253581235203,dev1_i,sensor-data,esp32
10 ,,5,2026-05-07T20:58:05.627Z,2026-05-08T20:58:05.627Z,2026-05-07T20:58:05.627Z,318.20700000000000,dev1_p,sensor-data,esp32
11 ,,6,2026-05-07T20:58:05.627Z,2026-05-08T20:58:05.627Z,2026-05-07T20:58:05.627Z,230.00000000000000,dev1_v,sensor-data,esp32
12 ,,7,2026-05-07T20:58:05.627Z,2026-05-08T20:58:05.627Z,2026-05-07T20:58:05.627Z,1.62625451189716,dev2_i,sensor-data,esp32
13 ,,8,2026-05-07T20:58:05.627Z,2026-05-08T20:58:05.627Z,2026-05-07T20:58:05.627Z,200.32651838368773,dev2_p,sensor-data,esp32
14 ,,9,2026-05-07T20:58:05.627Z,2026-05-08T20:58:05.627Z,2026-05-07T20:58:05.627Z,230.00000000000000,dev2_v,sensor-data,esp32
15 ,,10,2026-05-07T20:58:05.627Z,2026-05-08T20:58:05.627Z,2026-05-07T20:58:05.627Z,1.86720769454890,dev3_i,sensor-data,esp32
16 ,,11,2026-05-07T20:58:05.627Z,2026-05-08T20:58:05.627Z,2026-05-07T20:58:05.627Z,10.38429516174379,dev3_p,sensor-data,esp32
17 ,,12,2026-05-07T20:58:05.627Z,2026-05-08T20:58:05.627Z,2026-05-07T20:58:05.627Z,1.0000000000000000,dev3_v,sensor-data,esp32
18 ,,13,2026-05-07T20:58:05.627Z,2026-05-08T20:58:05.627Z,2026-05-07T20:58:05.627Z,10.76488856693237,discharge_i,sensor-data,esp32
19 ,,14,2026-05-07T20:58:05.627Z,2026-05-08T20:58:05.627Z,2026-05-07T20:58:05.627Z,303.88889447500970,discharge_p,sensor-data,esp32
20 ,,15,2026-05-07T20:58:05.627Z,2026-05-08T20:58:05.627Z,2026-05-07T20:58:05.627Z,27.08221504000145,discharge_v,sensor-data,esp32
21 ,,16,2026-05-07T20:58:05.627Z,2026-05-08T20:58:05.627Z,2026-05-07T20:58:05.627Z,0.0000000000000000,solar_i,sensor-data,esp32
22 ,,17,2026-05-07T20:58:05.627Z,2026-05-08T20:58:05.627Z,2026-05-07T20:58:05.627Z,0.0000000000000000,solar_p,sensor-data,esp32
23 ,,18,2026-05-07T20:58:05.627Z,2026-05-08T20:58:05.627Z,2026-05-07T20:58:05.627Z,0.0000000000000000,solar_v,sensor-data,esp32
24 ,,9,2026-05-07T20:58:05.627Z,2026-05-08T20:58:05.627Z,2026-05-07T20:58:15.627Z,84.97999999999999,battery_charge,sensor-data,esp32
25 ,,1,2026-05-07T20:58:05.627Z,2026-05-08T20:58:05.627Z,2026-05-07T20:58:15.627Z,0.0000000000000000,charge_i,sensor-data,esp32
26 ,,2,2026-05-07T20:58:05.627Z,2026-05-08T20:58:05.627Z,2026-05-07T20:58:15.627Z,0.0000000000000000,charge_p,sensor-data,esp32
27 ,,3,2026-05-07T20:58:05.627Z,2026-05-08T20:58:05.627Z,2026-05-07T20:58:15.627Z,27.69473171333921,charge_v,sensor-data,esp32
```



Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

12

Результати тестування



Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

13

Висновки і результати

Виконані завдання:

- Проаналізовано рішення конкурентів
- Визначено компоненти системи
- Створено електричну схему
- Створений програмний продукт
- Проведено тестування

Практична цінність:

- Готова система, може бути розширена іншими компонентами Smart Energy
- Програмний продукт можна підлаштувати під будь яку електричну систему

Перспективи подальших досліджень:

- Результати роботи будуть використані в контексті проєктів Smart Energy



Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

14