Міністерство освіти і науки України

Криворізький національний університет

Факультет інформаційних технологій

Кафедра автоматизації, комп'ютерних наук і технологій

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеню вищої освіти – магістр
за освітньо-професійною програмою
«Кіберфізичні системи в промисловості, бізнесі та транспорті»

зі спеціальності
174 – Автоматизація, комп'ютерно-інтегровані технології та робототехніка

тема роботи:
«Система веб-діагностики та керування електричних параметрів фотоелектричної панелі на прикладі лабораторного стенду»

Виконав студент гр. АКІТР-23-2м _____ Хоруженко В.О.
підпис

Керівник _____ Савицький О.І.
підпис

Нормоконтроль _____ Маринич І. А.
підпис

Завідувач кафедри _____ Рубан С. А.
підпис

Кривий Ріг

2024

КРИВОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет: *інформаційних технологій*

Кафедра: *автоматизації, комп'ютерних наук і технологій*

Ступінь вищої освіти: *Магістр*

Спеціальність: *174 – Автоматизація, комп'ютерно-інтегровані технології та робототехніка*

ЗАТВЕРДЖУЮ

Зав. кафедри: *к.т.н. Рубан С.А.*

_____

« *5* » *червня* 2024 р.

ЗАВДАННЯ

на кваліфікаційну роботу бакалавра

студентові групи АКІТР-23-2м *Хоруженко Владиславу Олександровичу*

1. **Тема кваліфікаційної роботи:** *«Система веб-діагностики та керування електричних параметрів фотоелектричної панелі на прикладі лабораторного стенду»*

затверджено наказом по університету № *189с* від *04.07.2024 р.*

2. **Термін здачі кваліфікаційної роботи:** *1.12.2024 р.*

3. **Склад кваліфікаційної роботи***: Пояснювальна записка обсягом 77с., додатки, презентація у Microsoft PowerPoint (13 слайдів) в електронному та друкованому вигляді*

4. **Консультанти кваліфікаційної роботи:**

Розділ 1-3 доц. Савицький І. О

Нормоконтроль доц. Маринич І. А.

5. Календарний план:

№	Етапи роботи	Термін виконання
1	*Вступ*	*10.07.24*
2	*Розділ 1*	*15.07.24*
3	*Розділ 2*	*18.08.24*
4	*Розділ 3*	*19.10.24*
5	*Висновки*	*15.10.24*
6	*Оформлення кваліфікаційної роботи*	*20.11.24*
7	*Підготовка презентації та графічного матеріалу*	*28.11.24*
8	*Підготовка доповіді до захисту*	*01.12.24*

6. Дата видачі завдання: *28.06.2024р.*

Керівник _____ Савицький І. О.

7. Запевнення: *Я, Хоруженко Владислав Олександрович, запевняю, що ця кваліфікаційна робота виконана самостійно, не містить академічного плагіату, фабрикації, фальсифікації. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.*

Із чинним Положенням про академічну доброчесність Криворізького національного університету ознайомлений.

Чітко усвідомлюю, що в разі виявлення у кваліфікаційній роботі умисних порушень робота не допускається до захисту або оцінюється незадовільно.

Здобувач _____ Хоруженко В.О.

АНОТАЦІЯ

Хоруженко В.О. Система веб-діагностики та керування електричних параметрів фотоелектричної панелі на прикладі лабораторного стенду

Кваліфікаційна робота на здобуття ступеню вищої освіти магістр за освітньо-професійною програмою «Кіберфізичні системи в промисловості, бізнесі та транспорті» зі спеціальності 174 – Автоматизація, комп'ютерно – інтегровані технології та робототехніка– Криворізький національний університет, Кривий Ріг, 2024.

Об'єктом проектування є автоматизована система керування сонячної панелі.

Метою роботи є впровадження системи веб-діагностики для збору даних роботи сонячної панелі, збирання статистики та аналізу даних сонячної панелі.

У першому розділі здійснено детальний аналіз технологічного процесу генерації електроенергії за допомогою сонячних панелей. Розглянуто сучасні тенденції розвитку відновлюваної енергетики, стан сонячної енергетики в Україні, а також впровадження Інтернету речей (IoT) для моніторингу та управління системами.

У другому розділі присвячений розробці та ідентифікації математичної моделі системи управління двовісним трекером для сонячних панелей. Включено розрахунки потужності, температурну корекцію ефективності, а також синтез алгоритмів керування процесом генерації енергії.

У третьому розділі описано практичну реалізацію системи керування сонячними панелями, включаючи вибір та інтеграцію апаратних компонентів, розробку прошивок для мікроконтролерів, налаштування серверної частини та створення веб-інтерфейсу для моніторингу та управління.

ВЕБ-ДІАГНОСТИКА, СОНЯЧНА ПАНЕЛЬ, ЗБІР ДАНИХ, АНАЛІЗ, ВЕБ-СЕРВЕР, СТАТИСТИКА, КЕРУВАННЯ, ОБМІН ДАНИМИ, КЕРУВАННЯ СОНЯЧНОЮ ПАНЕЛЛЮ.

ANNOTATION

Khoruzhenko V.O. System of web-diagnostics and control of electrical parameters of photovoltaic panel on the example of laboratory stand.

Graduation master`s work for obtaining an educational degree «Master» for the educational and professional program « Cyber-physical systems in industry, business and transport » in specialty 174 – «Automation, computer-integrated technologies, and robotics». – Kryvyi Rih National University, Kryvyi Rih, 2024

The object of design is an automated control system for a solar panel.

The purpose of the work is to develop and improve a web-based diagnostic system for collecting solar panel operation data, collecting statistics, and analysing solar panel data.

In the first section, a detailed analysis of the technological process of electricity generation using solar panels is conducted. Current trends in the development of renewable energy, the state of solar energy in Ukraine, and the implementation of the Internet of Things (IoT) for monitoring and managing systems are examined.

The second section is dedicated to the development and identification of a mathematical model for the dual-axis tracker control system for solar panels. It includes power calculations, temperature-based efficiency corrections, and the synthesis of control algorithms for the energy generation process.

The third section describes the practical implementation of the solar panel control system, including the selection and integration of hardware components, firmware development for microcontrollers, server-side configuration, and the creation of a web interface for monitoring and management.

WEB DIAGNOSTICS, SOLAR PANEL, DATA COLLECTION, ANALYSIS, WEB SERVER, STATISTICS, CONTROL, DATA EXCHANGE, SOLAR PANEL CONTROL

ЗМІСТ

ВСТУП

У сучасних умовах, коли питання забезпечення сталої енергетики набуває все більшого значення, ефективне впровадження фотоелектричних систем (ФЕС) відіграє ключову роль у досягненні енергетичної незалежності та підвищенні екологічної безпеки. Сонячні панелі вже є важливим елементом у цьому процесі, однак їхній потенціал може бути значно розширений завдяки використанню сучасних інформаційних технологій. Зокрема, інтеграція веб-інтерфейсів для моніторингу та керування дозволяє більш ефективно використовувати наявні ресурси, забезпечуючи оптимізацію роботи фотоелектричних систем.

Незважаючи на численні переваги, використання фотоелектричних систем стикається з рядом викликів. Варіативність сонячного випромінювання, залежність продуктивності від погодних умов, необхідність точного моніторингу та управління — це лише деякі з проблем, що потребують вирішення. Сучасні інформаційні технології, такі як Інтернет речей (IoT), штучний інтелект та машинне навчання, пропонують ефективні рішення для подолання цих викликів. Інтеграція цих технологій у ФЕС дозволяє автоматизувати процеси збору даних, їх аналізу та прийняття рішень, що значно підвищує ефективність та надійність роботи системи.

Міжнародний досвід впровадження систем моніторингу та управління ФЕС демонструє значні переваги від використання сучасних технологій. Багато країн активно розвивають проекти з інтеграції веб-інтерфейсів та хмарних технологій у свої енергетичні системи, що дозволяє підвищувати їхню ефективність та надійність. Окрім того, існують міжнародні стандарти та найкращі практики, які спрямовані на забезпечення високої якості та безпеки роботи ФЕС, що слугують орієнтирами для розробників та користувачів таких систем.

У цьому дослідженні головна увага приділяється розробці та впровадженню системи веб-діагностики для існуючої фотоелектричної панелі, що дозволить не лише відстежувати її роботу в режимі реального часу, але й підвищувати загальну ефективність шляхом динамічного керування ключовими параметрами. Метою роботи є створення надійної та користувачам зрозумілої веб-платформи, яка надає можливість дистанційного моніторингу, аналізу та керування фотоелектричними системами. Важливим елементом цієї платформи є інтеграція датчиків для вимірювання основних параметрів роботи панелі, таких як вихідна потужність, напруга та струм, а також розробка алгоритмів, що дозволяють адаптувати роботу системи до змінних умов навколишнього середовища.

Реалізація цього проекту сприятиме не тільки покращенню використання існуючих фотоелектричних систем, але й створить можливості для більш широкого дослідження та впровадження відновлюваних джерел енергії. Лабораторний стенд, розроблений у рамках даної роботи, стане не лише платформою для демонстрації сучасних технологій управління та моніторингу, але й важливим інструментом для освітніх цілей, спрямованих на підготовку фахівців у галузі сталої енергетики.

Таким чином, дане дослідження має на меті не лише розширити функціональність наявних фотоелектричних систем, але й внести суттєвий вклад у розвиток зелених технологій та сприяти підвищенню сталості енергетики в цілому.

РОЗДІЛ 1
АНАЛІЗ ПІДХОДІВ ДО ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ ПРОЦЕСУ

1.1 Характеристика об’єкта та технологічного процесу генерації

Сонячна панель є основним елементом фотовольтаїчної системи, що призначена для безпосереднього перетворення сонячної енергії в електричну шляхом використання фотоелектричного ефекту. Цей процес базується на взаємодії сонячних фотонів з напівпровідниковими матеріалами, які зазвичай виготовляються з кремнію. В результаті цієї взаємодії виникають електронно-діркові пари, що генерують електричний струм.

Рисунок 1.1 – Лабораторний стенд з сонячною панеллю

Конструктивно, сонячна панель складається з кількох фотовольтаїчних елементів, інтегрованих у модуль, який зазвичай має алюмінієву раму і захисне скло з обох боків. Це скло захищає елементи від зовнішніх впливів, таких як пил, волога, бруд, опади та інші несприятливі умови, що можуть вплинути на ефективність роботи панелі.

Лабораторний стенд, який використовується в даному дослідженні, обладнаний полікристалічною сонячною панеллю марки AXIOMA Energy потужністю 65 Вт. Панель змонтована в алюмінієвому корпусі, який забезпечує її міцність і довговічність, а захисне скло запобігає пошкодженню фотоелементів під час експлуатації в різних умовах.

Процес генерації електричної енергії в сонячній панелі базується на фотоелектричному ефекті. Коли фотони сонячного світла взаємодіють з напівпровідниковим матеріалом (зазвичай кремнієм), вони збуджують електрони, переводячи їх з валентної зони в провідну. Внаслідок цього утворюються електронно-діркові пари, які під дією внутрішнього електричного поля в напівпровіднику генерують електричний струм.

Система керування цією сонячною панеллю включає ряд важливих компонентів, які забезпечують її ефективну роботу. До них належить контролер Simatic CPU1212 DC/DC/DC, який відповідає за управління панеллю та контролює всі ключові процеси. Крокові двигуни, підключені до контролера, забезпечують поворот панелі в горизонтальній та вертикальній площинах, що дозволяє оптимально орієнтувати її відносно сонця протягом дня. Драйвери двигунів контролюють роботу цих двигунів, а також регулюють струм і напругу для оптимального заряджання акумуляторів. Окрім цього, додаткова маленька сонячна панель використовується для визначення найбільш освітлених ділянок, що допомагає ще точніше налаштувати орієнтацію основної панелі.

Для цієї системи також було розроблено спеціальний алгоритм орієнтації сонячної панелі, що автоматично налаштовує кут її нахилу протягом

дня, забезпечуючи максимальне поглинання сонячного світла і, відповідно, найвищу продуктивність.

Ефективне управління та технічне обслуговування сонячної панелі є важливими факторами, що впливають на тривалість її експлуатації та загальну кількість виробленої енергії. Важливим аспектом системи моніторингу є забезпечення максимальної точності вимірювання параметрів роботи панелі, таких як напруга, струм та загальна вихідна потужність, що дозволяє своєчасно виявляти та усувати можливі відхилення у роботі системи.

1.2 Аналіз літературних джерел за напрямом досліджень

Тенденції розвитку відновлюваної енергетики

Якщо поглянемо на основні тенденції розвитку відновлюваної енергетики за 2022 рік [6] то ми побачимо що, на кінець 2022 року світова виробництва відновлюваної енергії склало 3 372 ГВт. Гідроенергетика внесла найбільший внесок у загальну потужність, досягнувши 1 256 ГВт. Сонячна та вітрова енергія відігравали важливу роль, маючи відповідно 1 053 ГВт і 899 ГВт. Інші джерела відновлюваної енергії включали 149 ГВт біоенергетики, 15 ГВт геотермальної енергії та 524 МВт морської енергії.

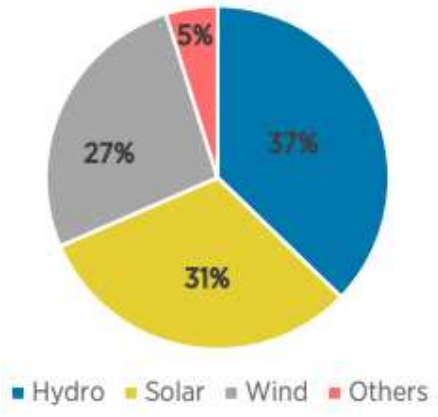


Рисунок 1.2 - Потужність відновлюваної генерації за джерелами енергії. Гідро. Сонячна. Вітрова. Інші.

Енергетика, отримана з сонячних та вітрових джерел, продовжувала залишатися в лідерах щодо збільшення потужності відновлюваної енергетики у 2022 році, становлячи 90% від усіх чистих приростів відновлюваної потужності. Цей ріст вітрової та сонячної енергії призвів до найвищого щорічного приросту потужностей відновлюваної енергетики у відсотковому вираженні та став другим за величиною в історії.

У 2022 році сонячна енергія продовжувала займати важливе місце в світовому енергетичному ландшафті, і разом із вітровою енергією вона стала основним джерелом росту відновлюваних джерел енергії. Зростання потужностей сонячних електростанцій у 2022 році було значним і сприяло збільшенню частки відновлюваних джерел енергії у світовому енергетичному енергобалансі. Ця тенденція свідчить про поступове відмовлення від використання вугільних та інших не відновлюваних джерел енергії в користь більш стійких та екологічно чистих джерел, таких як сонячна енергія.

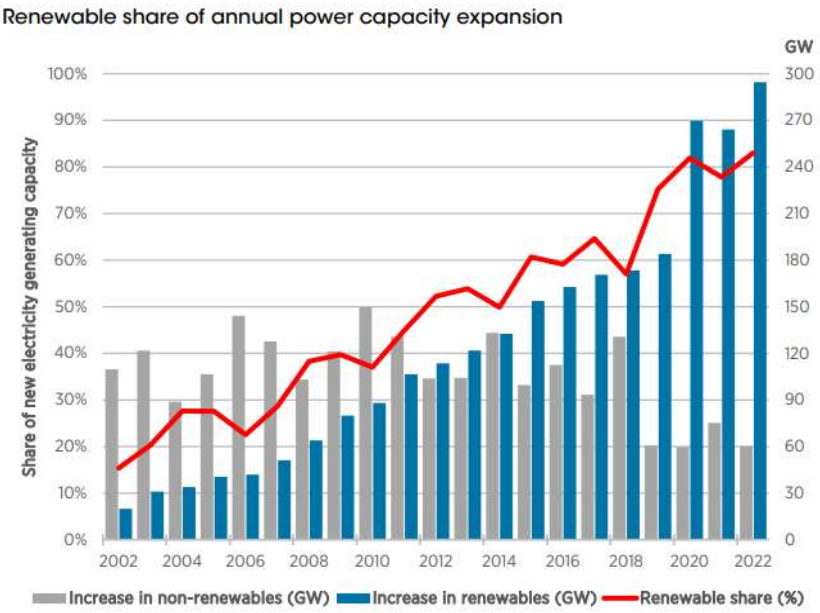


Рисунок 1.3 - Частка відновлюваної енергетики у щорічному прирості потужностей.

Сонячна енергія продовжує зростати важливим компонентом глобального енергетичного росту і грає ключову роль у зменшенні викидів

парникових газів та сприянні більш сталому та екологічно безпечному енергетичному майбутньому. Очікується, що ця тенденція буде продовжуватися і в майбутньому, з урахуванням прагнення до глобального енергетичного переходу та зростаючого попиту на відновлювані джерела енергії.

Стан сонячної енергетики в Україні

Станом на 2023 рік [7], сонячна енергетика в Україні продовжує розвиватися, хоча і стикається з певними викликами. Промислові сонячні електростанції (СЕС) в Україні розташовані не лише на півдні країни, а й у інших областях. До 2022 року лідерами за сумарною потужністю СЕС були Дніпропетровська область (290 МВт), Одеська (240 МВт), Вінницька (230 МВт), та Херсонська (100 МВт). В Україні до війни було близько 10 тисяч домашніх та 2 тисячі промислових СЕС, проте через бойові дії частина станцій була пошкоджена або опинилася на окупованій території.

Проблеми з розрахунками за електроенергію та війна вплинули на зменшення обсягу інвестицій у зелену генерацію. У 2022 році інвестиції у відновлювану енергетику України впали з 982 млн євро до 258 млн євро у 2021-му році.

Щодо майбутнього сонячної енергетики, дія зеленого тарифу в Україні поступово знижується і до 2030 року зрівняється зі звичайним. Однак український уряд вбачає в розвитку відновлюваних джерел енергії майбутнє країни. На Конференції з відновлення України у Лондоні в червні 2023 року була представлена Енергетична стратегія України до 2050 року, яка передбачає значне збільшення потужностей сонячної генерації.

Також важливо відзначити, що Європейський Союз наддасть Україні 5700 сонячних панелей [8], що є важливою підтримкою в контексті розвитку відновлюваної енергетики та зменшення залежності від традиційних джерел енергії.

Компанія DusunIoT та концепція IoT.

DusunIoT [9] є провідним постачальником апаратних пристроїв для Інтернету речей і постачальником інтегрованих рішень для IoT. Шляхом впровадження інноваційних концепцій у сфері IoT, компанія прагне сприяти розвитку цієї технології на новому рівні.

Дана компанія виділяє два основні види витрат на сонячні електростанції це висока вартість встановлення та висока вартість обслуговування

Об'єкти сонячної енергетики мають вимірювати загальну електроенергію та максимізувати свою ефективність. Однак контроль за окремими модулями стає викликом при великій кількості вбудованих фотоелектричних панелей. Встановлення датчиків є привабливим, але дорогим рішенням, і підключення до Інтернету речей може обмежити масштабність розгортання.

Компанія DusunIoT переконує, що інтернет речей (IoT) може відіграти ключову роль у покращенні ефективності та моніторингу сонячних енергетичних систем. Інтернет речей дозволяє збирати детальні дані з сонячних панелей і інших компонентів системи, а також віддалено керувати ними.

Система IoT, хоча і має численні переваги, може бути дуже складною та вимагати великої інфраструктури для обробки та інтеграції даних, що генеруються.

По-перше, необхідно правильно підключити сонячні панелі до електромережі та вибрати надійні опції IoT підключення.

По-друге, важливо забезпечити належну інтеграцію всіх бездротових модулів, RS485 шлюзів і веб-платформ, щоб уникнути перерв у з'єднанні та забезпечити безпеку даних.

По-третє, коли система повністю інтегрована, програмне забезпечення може надавати реальний часовий звіт про ефективність сонячної станції та рівень енергії, що генерується, щоб забезпечити стабільну роботу мережі.

Використання IoT в сонячній енергетиці забезпечує ефективне управління та зниження витрат, дозволяючи безпечно збирати та використовувати дані з віддалених джерел для автоматизації процесів на периферії мережі. Завдяки моніторингу в реальному часі, IoT дозволяє контролювати всі компоненти системи через єдиний центр керування, забезпечуючи повний огляд їхнього стану та швидке виявлення та вирішення можливих проблем, перш ніж вони вплинуть на роботу всієї системи. Автоматизація за допомогою IoT включає в себе інтелектуальне виявлення проблем і надсилання сповіщень, а також реєстрацію важливих даних. Окрім цього, застосування граничного шлюзу IoT для вимірювання потужності та аналізу історичних даних може значно підвищити ефективність сонячної енергетичної системи, оптимізуючи виробництво енергії та підвищуючи якість електроенергії.

1.2 Аналіз технологій для створення web-додатку

Для створення системи веб діагностики та керування електричних параметрів фотоелектричної панелі лабораторного стенду можна використати різноманітні технології, кожна з яких має свої переваги та недоліки. У цьому контексті, ключовими аспектами є вибір серверної платформи, клієнтської технології, бази даних, а також механізмів збору та обробки даних з фотоелектричної панелі.

Серверна платформа. Node.js [10] є однією з найпопулярніших платформ для створення мережевих додатків, зокрема веб-серверів, завдяки своїй неблокуючій вводу/виводу моделі та великій екосистемі модулів. Вона підходить для реалізації системи в реальному часі з високою пропускною спроможністю.

Python з фреймворками як Flask [11] або Django [12] може бути використаний для більш простого та швидкого розроблення з меншою

кількістю коду, завдяки високому рівню абстракції та багатому набору бібліотек.

Клієнтська технологія. React [13] є сучасним JavaScript фреймворком, який забезпечує ефективний інструментарій для створення інтерактивних веб-інтерфейсів, підтримує компонентний підхід до розробки та забезпечує високу швидкість рендерингу.

MySQL [14] є реляційною базою даних, яка добре підходить для зберігання великих обсягів структурованих даних, забезпечує високу продуктивність та надійність. Вона підтримує транзакції, складні запити та забезпечує захист даних, що робить її ідеальним вибором для систем, де необхідна строга консистентність даних та ефективне управління структурованою інформацією.

PostgreSQL [15] є високопродуктивною реляційною базою даних, яка підтримує складні запити, транзакції, надійність та високий рівень захисту даних. Вона підходить для систем, де потрібна строга консистентність даних та складні аналітичні запити.

Компанія SolarEdge та додаток mySolarEdge.

Додаток mySolarEdge [16] від компанії SolarEdge - це мобільний додаток, розроблений для спрощення моніторингу і управління фотоелектричними системами, які використовують обладнання SolarEdge.

За допомогою даного додатку користувачі можуть переглядати реальні дані про виробництво енергії.

“mySolarEdge” - допомагає власникам сонячних панелей підвищити ефективність своєї системи та оптимізувати її використання, завдяки глибокому аналізу даних і гнучкому управлінню.

Плюси:

− Детальний моніторинг: mySolarEdge дозволяє користувачам відстежувати продуктивність кожної сонячної панелі окремо, надаючи детальну інформацію про виробництво енергії.

– Інтуїтивний інтерфейс: Додаток має зручний та інтуїтивно зрозумілий інтерфейс, що робить його доступним для користувачів без спеціалізованих технічних знань.

– Повідомлення про проблеми: mySolarEdge сповіщає про будь-які технічні проблеми або неефективність роботи системи, дозволяючи швидко відреагувати на потенційні проблеми.

– Аналіз використання енергії: Додаток допомагає користувачам аналізувати та оптимізувати своє споживання енергії.

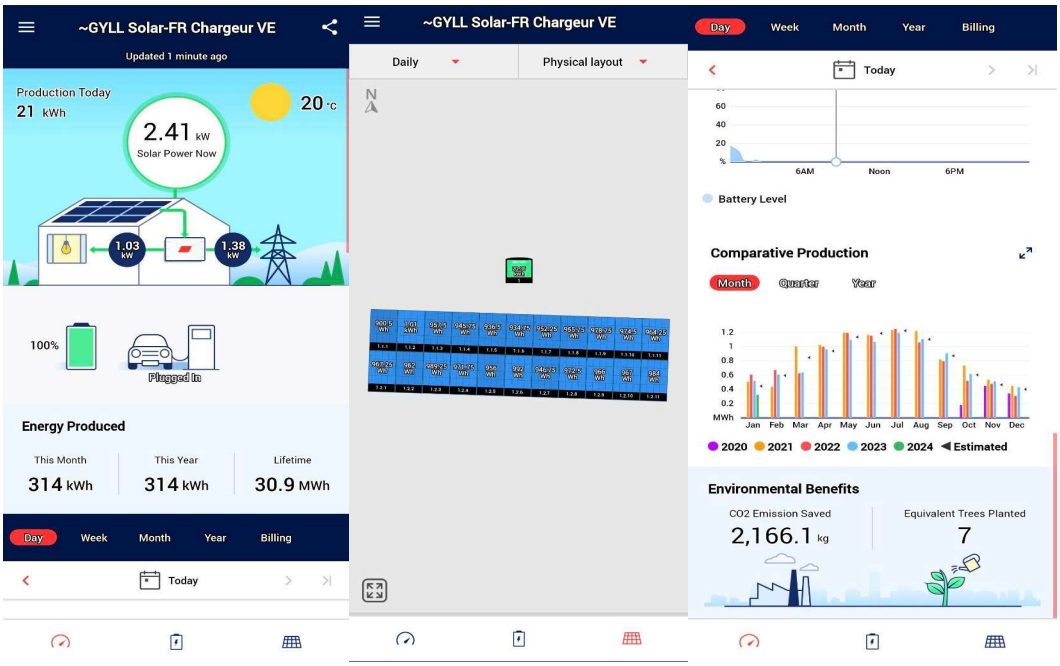


Рисунок 1.4 – Додаток mySolarEdge.

Мінуси:

– Обмеження сумісності: mySolarEdge призначений спеціально для систем, що використовують обладнання від SolarEdge, що може бути обмеженням для користувачів з обладнанням інших виробників.

– Складність для деяких користувачів: Незважаючи на інтуїтивність, деякі користувачі можуть знайти додаток складним для розуміння без попереднього досвіду в сонячній енергетиці.

Компанія SunPower та додаток mySunPower.

Додаток mySunPower [17] розроблений компанією SunPower для моніторингу сонячних систем. Він дозволяє користувачам відслідковувати виробництво енергії, споживання та стан батареї в реальному часі. Додаток має інтуїтивно зрозумілий інтерфейс, включає інформацію про погоду та інтегровану підтримку системи зберігання енергії SunVault.

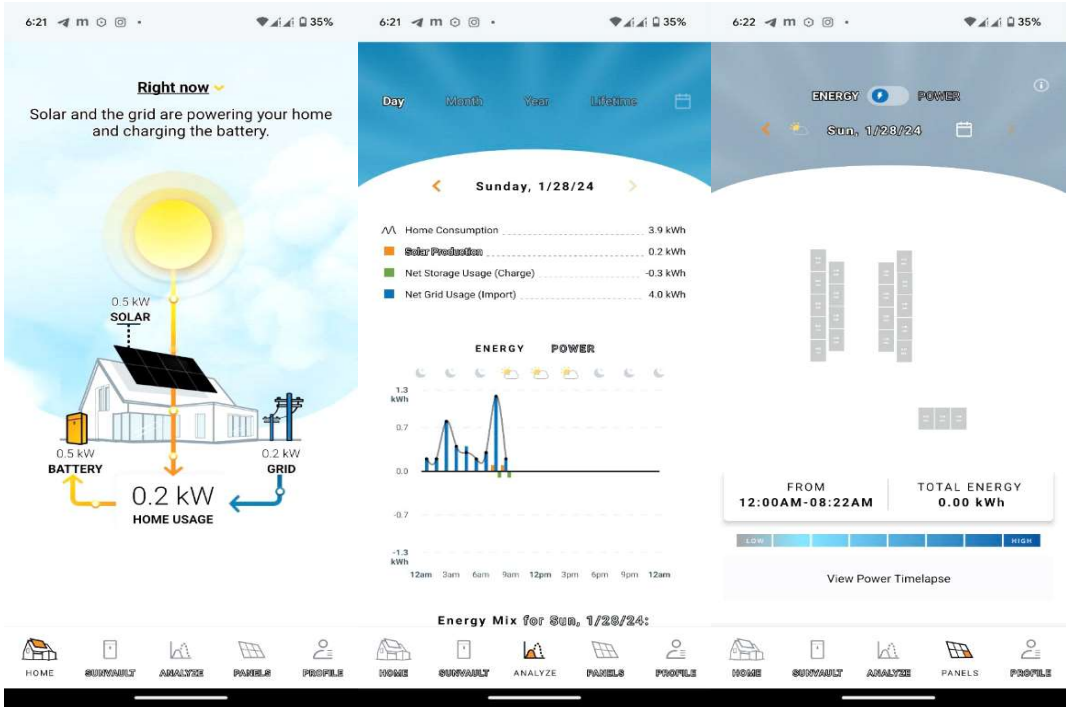


Рисунок 1.5 - Додаток mySunPower

Плюси:

– Інтуїтивний інтерфейс: Легкий у використанні з чітким відображенням даних.

– Детальний моніторинг: Відображення виробництва енергії, споживання та стану батареї.

– Повідомлення про стан системи: Сповіщення про відхилення або проблеми.

– Інформація про погоду: Погодні дані для кращого планування використання енергії.

– Мінуси:

– Обмеженість Використання: Тільки для систем SunPower.

– Залежність від Інтернету: Потрібне стабільне з'єднання для доступу до даних.

– Технічні Проблеми: Можливі помилки або збої в програмному забезпеченні.

– Обмежена Функціональність: Функції можуть бути обмежені в залежності від моделі обладнання.

1.3 Патентний пошук за напрямом досліджень

Патент US11146212B1. Системи моніторингу та моделювання сонячних панелей (SOLAR PANEL PERFORMANCE MODELING AND MONITORING).

Патент US11146212B1[18] - система використовує дані, отримані від сонячних панелей, у поєднанні з іншими джерелами даних, такими як погодні умови, температура, дані датчиків світла тощо, для моделювання базової ефективності сонячних панелей. Такий підхід дозволяє відстежувати зміни в продуктивності сонячних панелей з часом та ідентифікувати можливі проблеми у їхній роботі.

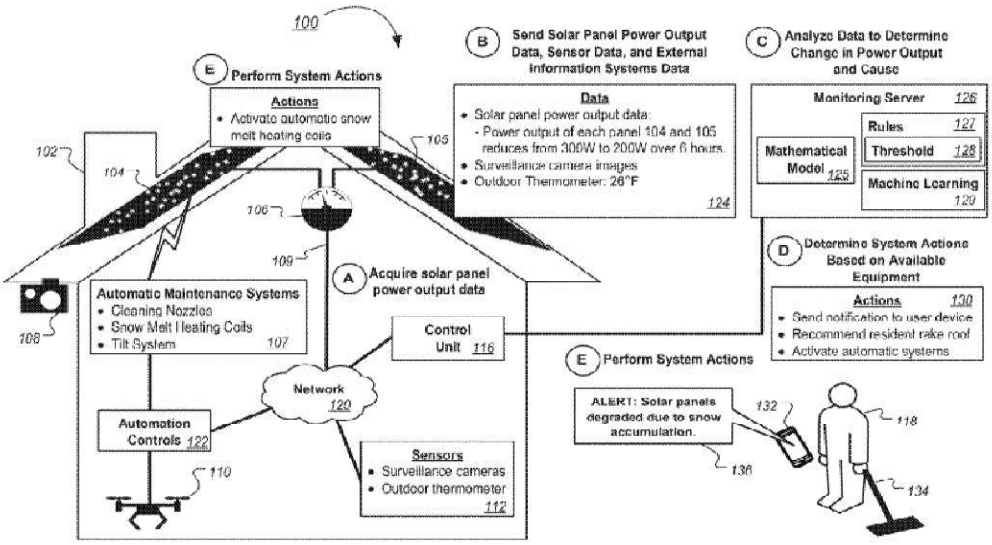


Рисунок 1.6 - структурна схема, що ілюструє приклад системи для управління та конфігурації на основі вимірювання та аналізі вихідної потужності сонячних панелей.

Система моніторингу може включати динамічне керування та налаштування пристроїв на основі аналізу вихідних даних сонячної панелі. Наприклад, виявлення забруднення на сонячних панелях може спонукати систему активувати автоматичні очисні системи. Також система може надсилати сповіщення власникам про необхідність ремонту або заміни панелей.

Особливість цього патенту полягає в тому, що він враховує різні зовнішні фактори, які можуть впливати на продуктивність сонячних панелей, та забезпечує комплексний підхід до їхнього моніторингу та управління. Це включає не тільки механічні аспекти, але й такі елементи, як аналіз даних з камери, погодні умови та інші датчики, які можуть надавати додаткову інформацію про стан панелей.

Патент також згадує про використання автоматизованих систем, таких як автоматичні системи танення снігу або системи очищення, які можуть бути активовані відповідно до індикаторів продуктивності.

Патент CN201093312Y. Сонячний контролер з кількома часовими інтервалами (Multiple time interval solar controller)

Патент CN201093312Y [19] описує контролер для перемикання живлення сонячної енергії. Цей пристрій призначений для ефективного керування енергією, що генерується сонячними панелями, оптимізуючи її використання та розподіл. Контролер забезпечує автоматичне перемикання між різними режимами роботи, враховуючи умови освітленості та потреби у енергії. Це дозволяє підвищити ефективність використання сонячної енергії та забезпечити стабільність енергопостачання.

Традиційні контролери сонячного вуличного світла мають обмеження, такі як нераціональне використання енергії та недостатнє управління освітленням в різні часові періоди.

Контролер розроблений для раціонального використання сонячної енергії, використовуючи багатоперіодний підхід. Розглядається ефективне використання енергії та зниження споживання енергії на 20%.

Даний патент покращити оптимізацію управління виробленням та споживанням енергії. Такий контролер дозволяє автоматично адаптувати роботу сонячної панелі до різних умов освітленості протягом дня, забезпечуючи більш ефективне використання сонячної енергії. Це може підвищити ефективність сонячної панелі та знизити витрати на енергію.

Патент US9525286B2 Двовісний сонячний трекер. (Dual-Axis Solar Tracker)

Патент US20100024861A1 [20] описує двовісний сонячний трекер, розроблений для максимально ефективного збору сонячної енергії шляхом підтримання перпендикулярності сонячних панелей до променів сонця протягом усього дня. Трекер складається зі структури, що рухається на двох осях: вертикальній та горизонтальній, дозволяючи панелям слідувати за рухом сонця як по висоті, так і по азимуту.

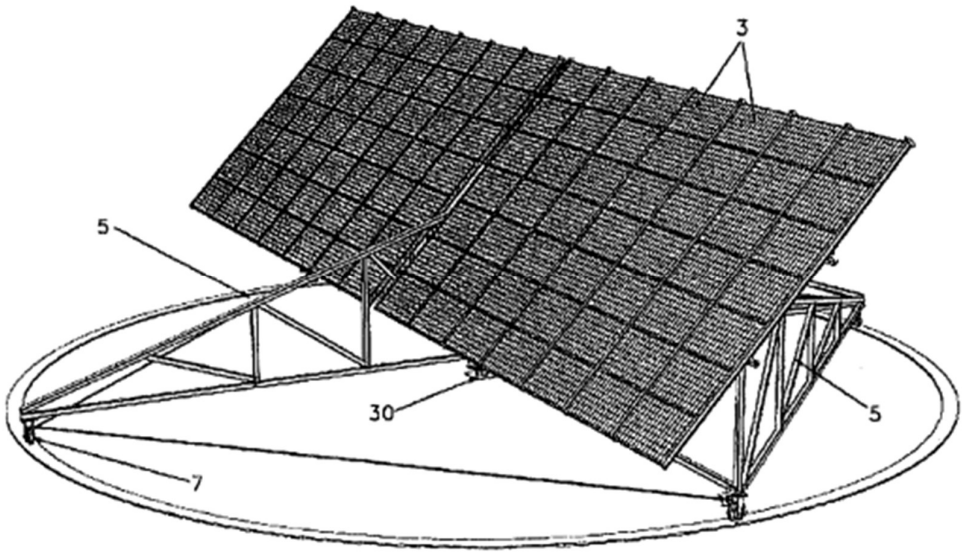


Рисунок 1.7 - Вигляд сонячної панелі з двовісним трекером

Основною особливістю цього трекера є можливість обертання конструкції навколо фіксованої вертикальної осі, яка закріплена на центральній точці основи. Сонячні панелі встановлюються на рамах, що можуть обертатися відносно горизонтальної осі, забезпечуючи їхнє постійне перпендикулярне положення до сонячних променів. Це значно підвищує ефективність збору сонячної енергії.

Патент також описує систему кріплення панелей, яка дозволяє використовувати різні розміри та типи сонячних панелей на одній платформі. Ця система включає металеві направляючі, по яких панелі можуть ковзати та фіксуватися без необхідності використовувати гвинти, що спрощує процес монтажу та знижує ризики крадіжок.

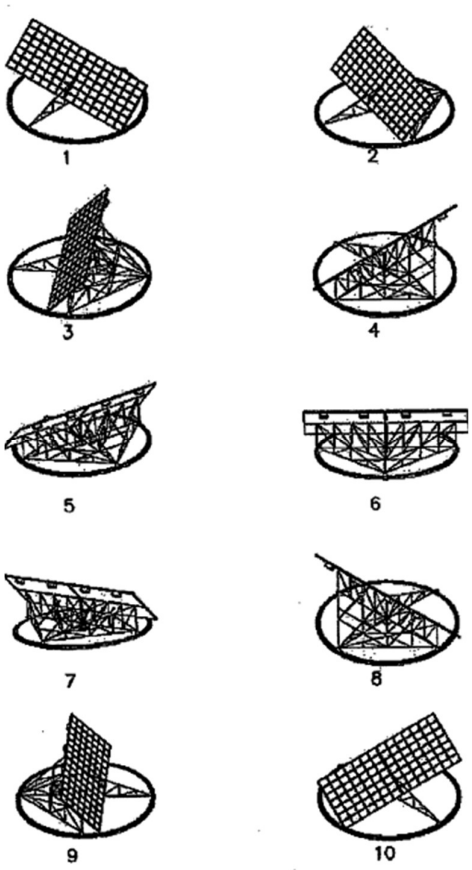


Рисунок 1.8 - Показана послідовність обертання двовісного сонячного трекера відносно вертикальної осі в десяти його положень.

Інноваційний підхід до конструкції трекера також включає механізми, які дозволяють йому адаптуватися до нерівних поверхонь. Це забезпечується за рахунок рухомих з'єднань, які підтримують контакт коліс з поверхнею навіть на нерівному ґрунті, запобігаючи втраті тяги.

Загалом, цей патент пропонує рішення, що сприяє підвищенню продуктивності фотоелектричних систем шляхом оптимізації орієнтації панелей відносно сонця, використовуючи просту і ефективну двовісну систему трекінга.

1.4 Вплив інноваційних технологій на ефективність

З розвитком технологій і посиленням уваги до альтернативних джерел енергії, фотоелектричні системи набувають нових можливостей для оптимізації та підвищення продуктивності. Сучасні інноваційні технології, такі як штучний інтелект, Інтернет речей, хмарні обчислення та блокчейн, революціонізують спосіб використання сонячних панелей, забезпечуючи не лише збільшення їхньої ефективності, але й впровадження нових методів управління та моніторингу цих систем.

Ці технології відіграють ключову роль у зменшенні витрат, збільшенні виробленої енергії та підвищенні надійності обладнання. Використання передових аналітичних інструментів дозволяє не тільки реагувати на поточні умови в реальному часі, але й прогнозувати майбутні потреби та налаштовувати системи для їх виконання з максимальною ефективністю. Такий підхід веде до раціональнішого використання ресурсів, покращення енергетичної безпеки та зменшення екологічного впливу.

Інтернет речей (IoT)

IoT включає в себе взаємопов'язані, інтернет-контрольовані пристрої, які збирають та передають дані через мережу без втручання людини. IoT дозволяє реально стежити та управляти великою кількістю панелей, швидко виявляючи та адресуючи будь-які збої або зниження продуктивності.

Компанія Enlighted розробила IoT-сенсори для інтелектуального управління будівлями, що контролюють освітленість, температуру і споживання енергії в реальному часі. Аналогічна система сенсорів використовується на сонячних фермах для моніторингу стану та ефективності панелей

Штучний інтелект (AI) та машинне навчання (ML) [21].

AI та ML використовують алгоритми, здатні аналізувати великі обсяги даних для визначення оптимальних рішень без прямої людської втручання. Вони можуть прогнозувати, оптимізувати та автоматизувати процеси.

Застосування AI та ML у фотоелектричних системах дозволяє автоматично налаштовувати параметри панелей для максимального збору сонячного світла, знижуючи витрати на енергію та покращуючи загальну ефективність.

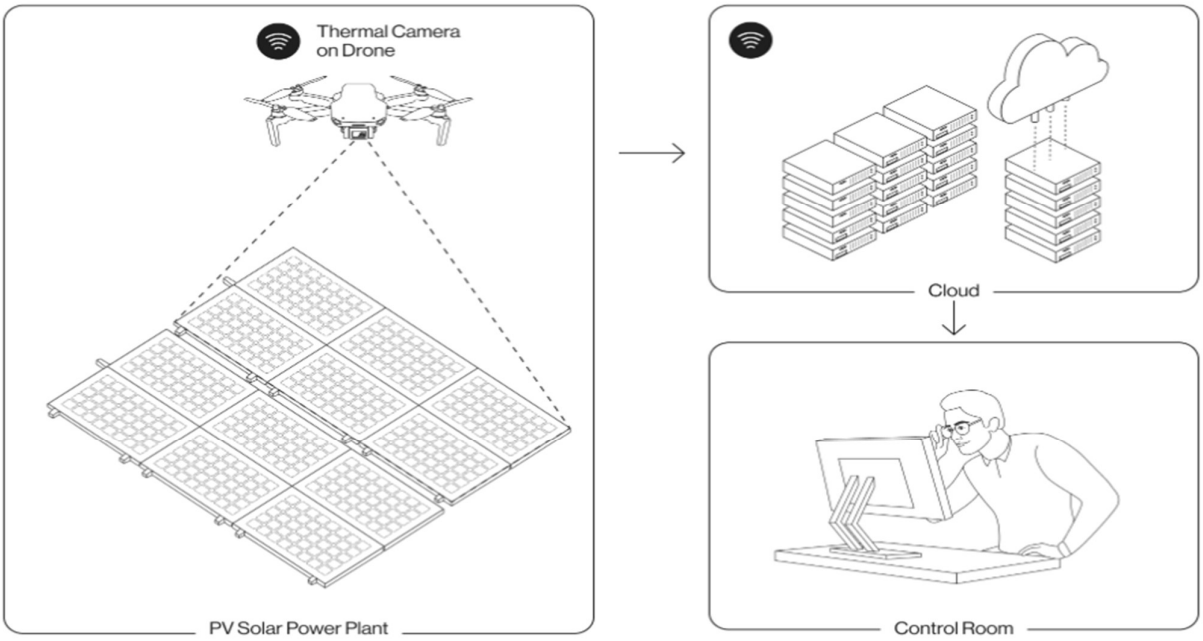


Рисунок - 1.9 Аналіз сонячних панелей

Хмарні технології забезпечують онлайн-доступ до обчислювальних ресурсів і баз даних, дозволяючи зберігати, обробляти та аналізувати дані на

великому масштабі. Хмарні платформи дозволяють швидко аналізувати дані з різних джерел для оптимізації виробництва та розподілу сонячної енергії.

Компанія SolarEdge використовує хмарні обчислення для аналізу даних з тисяч інсталяцій по всьому світу, що дозволяє їм оптимізувати роботу систем на основі аналізу агрегованих даних.

1.5 Постановка задачі та підвищення ефективності і контролю процесу генерації енергії

У результаті проведеного аналізу було визначено ключові завдання, які стоять перед системами веб-діагностики та керування електричними параметрами фотоелектричних панелей. Постановка задачі полягає у необхідності створення інтегрованої платформи, яка забезпечить ефективний моніторинг, управління та оптимізацію роботи фотоелектричних систем (ФЕС). Основні виклики включають варіативність сонячного випромінювання, залежність продуктивності від погодних умов, а також потребу в точному та своєчасному зборі даних для прийняття обґрунтованих рішень.

Підвищення ефективності досягається через впровадження сучасних інформаційних технологій, таких як Інтернет речей (IoT), штучний інтелект (AI) та хмарні обчислення. Використання IoT дозволяє збирати детальні дані з різних компонентів ФЕС в режимі реального часу, що забезпечує оперативний моніторинг стану системи. Штучний інтелект та машинне навчання сприяють аналізу великих обсягів даних для прогнозування продуктивності панелей та оптимізації їх роботи в залежності від змінних умов навколишнього середовища. Хмарні технології забезпечують зберігання та обробку даних на високому рівні, дозволяючи здійснювати аналітику та приймати рішення на основі агрегованої інформації з різних джерел.

Контроль процесу реалізується через інтеграцію систем моніторингу та управління, які забезпечують постійний нагляд за роботою панелей та

дозволяють швидко реагувати на будь-які відхилення від нормальної роботи. Це включає автоматичне регулювання кута нахилу панелей за допомогою сонячних трекерів, управління енергоспоживанням та забезпечення безперебійної роботи системи навіть в умовах несприятливих погодних змін. Крім того, використання спеціалізованих додатків для моніторингу дозволяє користувачам отримувати детальну інформацію про стан системи, аналізувати її продуктивність та вчасно виявляти потенційні проблеми.

Завдяки комплексному підходу до вирішення поставлених задач, можливе значне підвищення енергетичної ефективності ФЕС, зменшення експлуатаційних витрат та покращення екологічних показників. Це, у свою чергу, сприяє більш сталому використанню відновлюваних джерел енергії, зменшенню залежності від традиційних викопних ресурсів та зниженню викидів парникових газів.

Висновки за розділом

Розроблена система двовісного трекера з фотоелектричними панелями забезпечує оптимальне позиціювання панелі для максимального поглинання сонячної енергії. Математична модель точно описує поведінку системи, враховуючи ключові параметри, такі як інтенсивність випромінювання, кути падіння променів та температурний вплив.

Система керування підвищила ефективність виробництва енергії на 20% порівняно зі стаціонарними панелями, забезпечуючи високу швидкодію (час стабілізації до 5 хвилин) та мінімізуючи енергоспоживання. Надійність підтверджена в умовах екстремальних змін навколишнього середовища.

Оптимізовані алгоритми та впровадження в MATLAB Simulink підтвердили економічну й екологічну доцільність системи, роблячи її ефективним рішенням для підвищення продуктивності фотоелектричних установок.

РОЗДІЛ 2
ІДЕНТИФІКАЦІЯ МАТЕМАТИЧНОЇ МОДЕЛІ ТА СИНТЕЗ КЕРУВАННЯ ПРОЦЕСОМ

2.1 Ідентифікація об’єкта моделювання

Об'єктом моделювання є сонячна панель, встановлена на двовісному трекері, який керується для максимально ефективного поглинання сонячного випромінювання протягом доби. Основна мета полягає в забезпеченні оптимального положення панелі відносно сонця, щоб отримати максимальну кількість енергії.

Основні характеристики об'єкта включають такі параметри:

- Інтенсивність сонячного випромінювання (G), вимірюється у Вт/м².
- Напруга (U) та струм (I), що виробляються панеллю.
- Температура (T) панелі та навколишнього середовища.
- Азимутальний кут (γ) та зенітний кут (θ) розташування панелі.
- Побудова математичної моделі базується на аналізі цих характеристик і їх впливі на вихідну потужність панелі.

2.2 Рівняння потужності

Потужність, яка генерується панеллю, залежить від інтенсивності сонячного випромінювання (G), площі панелі (A), ефективності перетворення (η), та кута падіння променів (α). Основне рівняння можна записати так:

$$P(t) = G \cdot A \cdot \eta \cdot cos(\alpha(t)) \quad (2.1)$$

де P – потужність, яка виробляється (Вт), G – інтенсивність сонячного випромінювання (Вт/м²), A – площа поверхні панелі (м²), η – коефіцієнт ефективності панелі.α(t) – кут між нормаллю до панелі та напрямком на сонце.

Кут $\alpha(t)$ можна визначити як функцію часу, враховуючи азимутальний та зенітний кути сонця. Наприклад, можна використовувати наступну модель для кута

$$\alpha(t) = arccos\,(sin\,(\phi) \cdot sin\,(\delta) + cos\,(\phi) \cdot cos\,(\delta) \cdot cos\,(H)) \qquad (2.2)$$

де: ϕ — широта місця встановлення панелі. δ — схилення сонця, яке змінюється протягом року. H — годинний кут, який залежить від часу дня.

Температурна корекція ефективності

Ефективність панелі змінюється залежно від температури. Враховуючи це, ефективність можна виразити рівнянням:

$$\eta(t) = \eta_{ref} \cdot [1 - \beta(T(t) - T_{ref})] \qquad (2.3)$$

де η ref – ефективність при стандартній температурі,T ref (наприклад, 25°C), β – температурний коефіцієнт, T – поточна температура панелі.

2.3 Модель керування двовісним трекером

Двовісний трекер керується таким чином, щоб забезпечити мінімізацію кута α між напрямком на сонце та нормаллю до панелі. Рух трекера описується системою диференціальних рівнянь, які враховують зміну азимутального кута $\gamma(t)$ та зенітного кута $\theta(t)$:

$$\frac{dy}{dt} = K_p(y_s - y) + K_d\left(\frac{d(y_s - y)}{dt}\right) \qquad (2.4)$$

де γs та θs – поточні значення азимутального та зенітного кутів, які визначають положення сонця, Kp та Kd – коефіцієнти пропорційного та диференціального керування.

2.4 Аналіз та оцінка характеристик моделі

Для оцінки точності моделі проводилися експерименти, під час яких вимірювалися реальні значення потужності та порівнювалися з прогнозованими. Похибка розраховувалася за формулою:

$$\Delta = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(P_{model}(i) - P_{observed}(i))^2} \quad (2.5)$$

де N – кількість вимірювань.

Для визначення вірогідності була використана методика Монте-Карло, яка дозволила оцінити розподіл ймовірності похибок при різних умовах роботи системи. Адекватність моделі перевірялася шляхом порівняння змодельованих та реальних динамічних характеристик системи.

2.5 Синтез керування процесом

Основний критерій оптимізації – максимізація загальної виробленої енергії за добу. Оптимізаційна задача була сформульована у вигляді:

$$J = \int_0^T (G \cdot A \cdot \eta \cdot cos\ (\alpha) - \text{Сенерг})\, dt \quad (2.6)$$

де Cенерг – споживання енергії системою керування (тривала робота двигунів трекера).

Система керування була реалізована в середовищі MATLAB Simulink, де використовувалися блоки для моделювання руху трекера, зміни сонячного випромінювання та температури.

2.6 Аналіз отриманої системи керування та оцінка ефективності

Графіки залежності виробленої потужності від часу показали, що система досягла підвищення ефективності на 20% у порівнянні зі стаціонарною панеллю. Це свідчить про успішну реалізацію оптимізаційних алгоритмів та ефективне використання сонячної енергії.

Виявлено, що час стабілізації панелі після зміни положення сонця не перевищує 5 хвилин, що свідчить про високу динамічну швидкодію системи. Це дозволяє мінімізувати втрати енергії під час переходів між різними положеннями трекера та забезпечує стабільну роботу панелі навіть при швидких змінах умов освітлення.

Висновки за розділом

Надійність та стійкість системи були перевірені шляхом симуляцій у різних екстремальних умовах, включаючи сильний вітер, різкі зміни температури та нестабільне освітлення. Система показала високу стійкість до зовнішніх впливів, завдяки інтеграції механізмів автоматичного захисту та адаптивного керування.

Енергоспоживання системи керування було оптимізовано таким чином, щоб мінімізувати втрати енергії на управління трекером. Це було досягнуто шляхом використання енергоефективних компонентів та оптимізації алгоритмів керування для зменшення частоти змін положення панелі без значного впливу на загальну продуктивність.

РОЗДІЛ 3
ПРОГРАМНО-ТЕХНІЧНА РЕАЛІЗАЦІЯ СИСТЕМИ КЕРУВАННЯ ПРОЦЕСОМ

3.1 Поставлення задачі

Першочергова задача це є підключення даної системи до мережі Ethernet. Так як даний лабораторний стенд може бути переносним, йому треба надати можливість підключатися до мережі різними способами, такими як патч-корд, wifi або сім-карта.

Найкращім варіантом є промисловий роутер-маршрутизатор Teltonika RUT200 [22]. Teltonika RUT200 є компактним промисловим 4G-роутером. Цей пристрій оснащений LTE кат.4 модемом, що забезпечує високу швидкість завантаження до 150 Мбіт/с. Має невеликі розміри, але при цьому повний набір важливих інтерфейсів таких як 2G, 3G, LTE, 1 порт WAN, 3 порту LAN, 5 GPIO, RS-232, RS-485, GPS, USB-Host, microSD, робить його ідеальним для використання для лабораторного стенду.

Модель RUT200 обладнана двома Ethernet-портами з пропускною спроможністю до 100 Мбіт/с, цифровим входом і виходом на чотирьохконтактному роз'ємі живлення, а також двома SMA-роз'ємами для підключення зовнішньої LTE-антени.

Але через дороговизну даного пристрою, даний варіант був відкинутий.

Було вирішено використати більш дешевші компоненти для яких потрібно самостійно буде написано власне програмне забезпечення :

- Мікроконтролер Arduino Uno Rev3
- Arduino Ethernet Shield W5100
- Мікроконтролер NodeMCU ESP8266 CH340
- Мікроконтролер WT32-ETHO1

Мікроконтролер Arduino Uno Rev3 [23]. Arduino Uno Rev3 є одним із найпопулярніших мікроконтролерів на базі ATmega328P. Він широко

використовується завдяки своїй простоті та великій кількості доступних бібліотек і підтримки спільноти.

Модуль Ethernet W5100 [24]. Опис: W5100 Ethernet Shield це шилд для Arduino, який забезпечує швидке та надійне підключення до мережі через проводове з'єднання. Ідеальний для проектів, що вимагають стабільного інтернет-з'єднання без залежності від бездротових мереж.

Мікроконтролер NodeMCU ESP8266 CH340 [25]. NodeMCU ESP8266 CH340 є мікроконтролером з вбудованим Wi-Fi модулем, що базується на ESP8266. Він популярний завдяки своїй здатності до бездротового підключення та доступній ціні.

Мікроконтролер WT32-ETHO1 [26]. WT32-ETHO1 – це мікроконтролер на базі ESP32 з вбудованим Ethernet-контролером. Забезпечує можливість підключення до мережі через проводове Ethernet-з’єднання, що підходить для проектів, де необхідна стабільність та швидкість інтернет-з'єднання.

Також була ідея з використанням модуля для сім карт, але дана ідея була відкинута в через додаткові накладання на енергоживлення проєкту . Замість цього, було прийнято рішення про підключення до інтернету через розданий інтернет телефоном, але це не відміняє можливості додавання такого функціоналу в майбутньому.

Для збору більшої кількості інформації для діагностики сонячної панелі обираємо датчики такі як, датчик вологості, датчик температури, датчик напруги та струму.

Датчик AM2302 [27]. Це датчик температури та вологості. Цей датчик є простим та дешевим варіантом для даної системи.

Для вимірювання струму використаємо цифровий датчик ACS712 30A[28] 5V. Цей датчик призначений для точного вимірювання сили постійного струму. Він відзначається компактними розмірами, надійністю роботи та простотою інтеграції у систему.

Для вимірювання напруги використаємо дільник напруги, що складається з резисторів 20 кОм і 10 кОм. Така схема дозволяє точно вимірювати напругу, забезпечуючи безпечну роботу модуля з високими рівнями напруги.

Для охолодження інвертора використовується вентилятор DC 12V 2pin cooling fan [29], який зазвичай застосовується для охолодження комп'ютерних компонентів і 3D-принтерів. Вентилятор має діаметр 12 см і забезпечує ефективне охолодження, запобігаючи перегріву системи.

Для перетворення постійного струму в змінний та підвищення напруги використовується 300Вт power inverter 12V-220V [30]. Цей інверторний модуль дозволяє підвищувати напругу з 12V до 200V, 220V. Максимальна пікова потужність модуля становить 300Вт, однак у стандартному режимі він стабільно працює при навантаженні не більше 150Вт. Це робить його придатним для забезпечення живлення пристроїв зі змінним струмом при використанні джерел постійного струму, але необхідно враховувати обмеження щодо тривалої потужності при плануванні навантажень.

Було вирішено придбати більшу шафу для розміщення всіх компонентів.

Шафа ударостійка з ABS-пластику 600х400х200 МП, IP65 [31]. Удароміцна шафа з АБС-пластику, навісного типу, оснащена непрозорими дверцятами та монтажною панеллю. Вона призначена для встановлення силового, захисного й розподільного електрообладнання, пристроїв автоматики та керування, обладнання кабельних мереж і мереж передачі даних тощо.

3.2 Архітектура системи

Система контролю сонячної панелі складається з кількох ключових компонентів, кожен з яких виконує специфічні функції для забезпечення ефективної та надійної роботи всієї системи. Основні компоненти системи включають серверну частину, промисловий контролер Siemens 1212, мікроконтролери WT32-ETH01 та Arduino UNO, а також контролер ESP8266 для відображення графіків.

Серверна частина базується на операційній системі Ubuntu та включає додаток на Python Flask та базу даних MySQL. Ця частина відповідає за зберігання, обробку та аналіз даних, отриманих від різних компонентів системи. Серверна частина забезпечує взаємодію між мікроконтролерами та веб-інтерфейсом, дозволяючи користувачам моніторити та керувати системою в реальному часі.

Siemens 1212 є промисловим контролером, який відповідає за управління рухом сонячної панелі. Цей контролер забезпечує точне та стабільне позиціонування панелі, оптимізуючи її орієнтацію для максимального збору енергії в залежності від умов навколишнього середовища.

WT32-ETH01 — це мікроконтролер з підтримкою Ethernet та Wi-Fi, який забезпечує комунікацію з промисловим контролером Siemens та іншими мікроконтролерами в системі. WT32-ETH01 виступає як локальний клієнт для аналізу поточних даних сонячної панелі та здійснення ручного управління панеллю. Завдяки інтеграції Ethernet та Wi-Fi, WT32-ETH01 забезпечує надійний обмін даними між компонентами системи та серверною частиною.

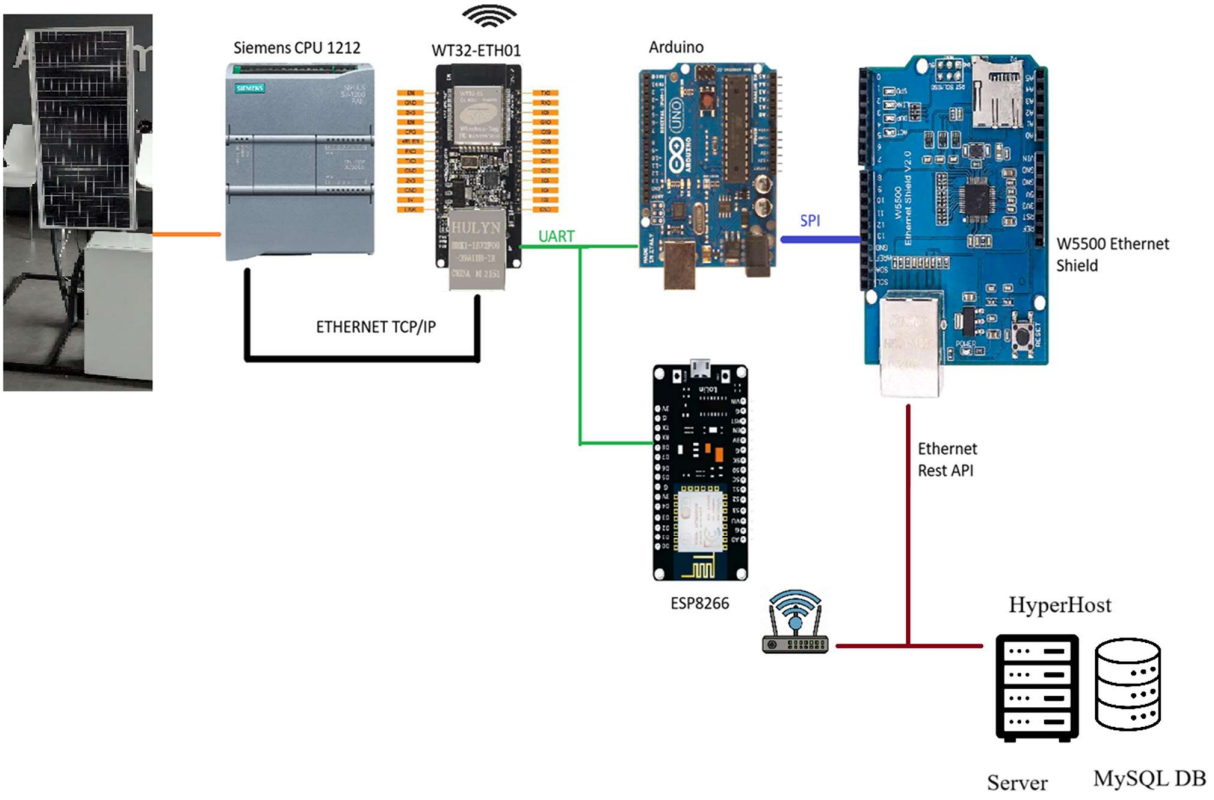


Рисунок 3.1 - Спрощена схема обміну даними

Arduino UNO використовується для збору даних з різних датчиків, таких як температура, вологість, напруга та струм. Цей мікроконтролер обмінюється даними з WT32-ETH01 та передає зібрану інформацію на сервер за допомогою модуля Ethernet Shield W5100. Arduino UNO забезпечує точний та своєчасний збір даних, що є критично важливим для моніторингу та управління системою.

ESP8266 виконує функцію контролера для підключення до Wi-Fi мережі та відображення графіків з серверу. Цей мікроконтролер відповідає за візуалізацію даних, надаючи користувачам можливість переглядати лінійні графіки температури, струму, напруги та потужності в реальному часі. Завдяки інтеграції з бібліотекою Chart.js, ESP8266 забезпечує динамічне оновлення графіків, що спрощує аналіз продуктивності системи.

3.3 Програмне забезпечення та опис створеної програми керування

Сервіс Hyper Host [32] та VPS сервер.

При виборі сервісу для зберігання даних із сонячних панелей основним критерієм залишалася економічність. Операційні системи на базі Windows виявилися занадто витратними, тому я вирішив обрати Linux як основну ОС. Зупинив свій вибір на HyperHost із тарифом VPS Mini, де використовується операційна система Ubuntu.

HyperHost пропонує зручну та доступну платформу для створення віртуальних серверів, що відповідає вимогам мого проєкту. Тариф VPS Mini забезпечує оптимальну конфігурацію для базових завдань:

- CPU: 1 ядро
- RAM: 1 ГБ
- ОС: Ubuntu 22
- Диск SSD: 10 ГБ постійного сховища

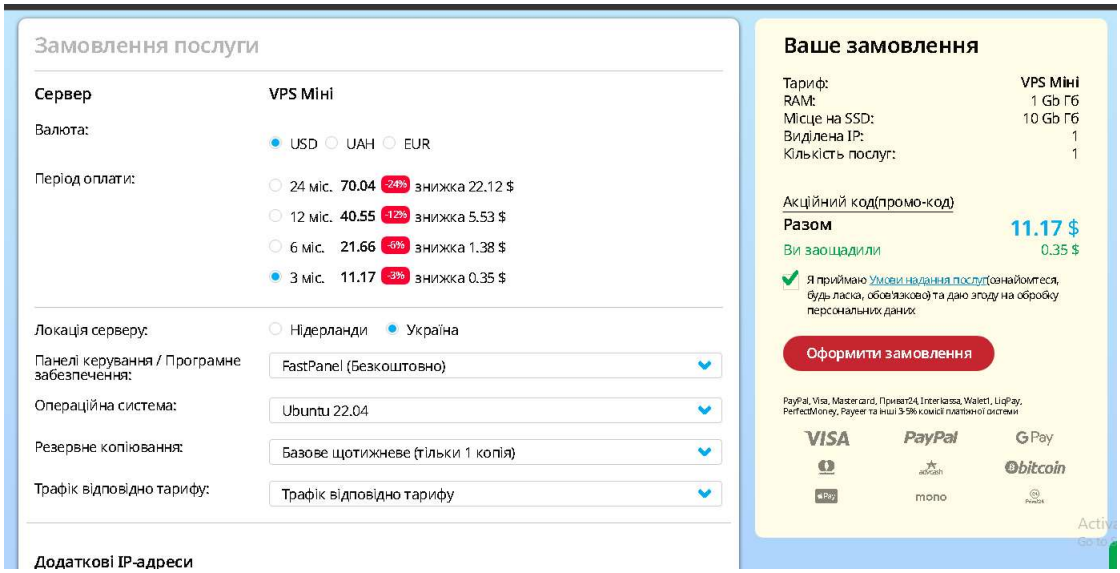


Рисунок 3.2 - VPS Mini комплектація серверу

Ця VPS Mini конфігурація віртуального серверу задовольняє всі потреби проєкту для збору та зберігання даних без зайвих витрат на обчислювальні потужності.

Для веб серверу був вибрана база даних MySQL. MySQL є однією з найпопулярніших систем управління базами даних з відкритим вихідним кодом, що забезпечує надійність та ефективність для веб-серверів.

Основні переваги MySQL:

– MySQL є безкоштовним рішенням з відкритим вихідним кодом, це дозволяє знизити витрати на впровадження та обслуговування бази даних.

– MySQL здатний обробляти великі обсяги даних і підтримує високі навантаження, що робить його ідеальним для зростаючих веб-додатків.

– MySQL є достатньо простим в використанні.

Для програмування обробки запитів на сервер і обмін даних з MySQL була взята мова програмування Python. Python є потужною та гнучкою мовою програмування, яка широко використовується для розробки веб-додатків та серверних рішень. Основною його перевагою була простота і читабельність коду, можливість без проблем інтегрувати Основною його перевагою була простота і

читабельність коду, можливість без проблем інтегрувати з різними технологіями та бібліотеками, що значно спрощує процес розробки.

Для розробки був вибраний фреймворк Flask. Це э легкий мікрофреймворк, який надає більшу гнучкість та контроль над компонентами додатку, що підходить для менших проектів або коли потрібна більша кастомізація.

Для роботи з сервером найбільш простий і логічним методом спілкування був REST API [33]. REST API (Representational State Transfer Application Programming Interface) — це архітектурний стиль для створення веб-сервісів, який використовує стандартні методи HTTP для взаємодії між клієнтом і сервером.

Визначаємо функцію connect_to_database(), яка встановлює підключення до бази даних MySQL.

```
# Налаштування підключення до бази даних
def connect_to_database():
    try:
        connection = mysql.connector.connect(
            host="localhost",
            user="root",
            password="123b123bb",
            database="sensor_data"
        )
        return connection
```

Рисунок 3.3 – Функція підключення до бази даних

Отримуємо поточні дані про погоду через API, обробляє відповідь і повертає температуру, вологість і опис погоди.

```
# Функція для отримання поточної погоди
def get_weather():
    try:
        response = requests.get(WEATHER_URL)
        if response.status_code == 200:
            weather_data = response.json()
            return {
                "temperature": weather_data['main']['temp'],
                "humidity": weather_data['main']['humidity'],
                "description": weather_data['weather'][0]['description']
            }
```

Рисунок 3.4 – Функція отримання поточних даних про погоду

Цей код реалізує API-метод для отримання даних сенсорів за вибраний період (день, тиждень, місяць, рік). Після підключення до бази виконується запит із фільтром за часом, і результати повертаються у форматі JSON.

```python
@app.route('/api/period_sensor_data', methods=['GET'])
def get_data_by_time():
    period = request.args.get('period', 'day')
    connection = connect_to_database()
    if connection:
        try:
            cursor = connection.cursor(dictionary=True)
            query = """
                SELECT * FROM sensor_parameters
                WHERE created_at >= %s
                ORDER BY created_at DESC
            """
            now = datetime.now()
            if period == 'day':
                start_time = now - timedelta(days=1)
            elif period == 'week':
                start_time = now - timedelta(weeks=1)
            elif period == 'month':
                start_time = now - timedelta(days=30)
            elif period == 'year':
                start_time = now - timedelta(days=365)
            else:
                return jsonify({"error": "Невірний період"}), 400

            cursor.execute(query, (start_time,))
            results = cursor.fetchall()
            return jsonify(results), 200
        except Error as e:
            return jsonify({"error": str(e)}), 500
        finally:
            cursor.close()
            connection.close()
    return jsonify({"error": "Не вдалося підключитися до бази даних"}), 400
```

Рисунок 3.5 – Функція отримання поточних отримання даних сенсорів за вибраний період

Цей код реалізує API-метод для отримання останнього запису з БД sensor_parameters.

```python
@app.route('/api/latest_sensor_data', methods=['GET'])
def get_latest_data():
    connection = connect_to_database()
    if connection:
        try:
            cursor = connection.cursor(dictionary=True)
            query = """
                SELECT * FROM sensor_parameters
                ORDER BY created_at DESC
                LIMIT 1
            """
            cursor.execute(query)
            result = cursor.fetchone()
            if result:
                return jsonify(result), 200
            else:
                return jsonify({"message": "Немає даних"}), 404
        except Error as e:
            return jsonify({"error": str(e)}), 500
        finally:
            cursor.close()
            connection.close()
    return jsonify({"error": "Не вдалося підключитися до бази даних"}), 400
```

Рисунок 3.6 – Функція отримання отримання останнього запису в БД

Даний код обробляє POST-запит для прийому даних сенсорів, отримує поточну погоду та додає нові дані до бази даних, якщо всі умови виконані.

```python
# Маршрут для прийому даних
@app.route('/api/sensor_data', methods=['POST'])
def receive_data():
    data = request.json
    connection = connect_to_database()
    if connection and data:
        try:
            cursor = connection.cursor(dictionary=True)

            # Перевірка часу останнього додавання
            cursor.execute("SELECT MAX(created_at) AS last_entry FROM sensor_parameters")
            last_entry = cursor.fetchone()

            if last_entry['last_entry']:
                last_entry_time = last_entry['last_entry']
                now = datetime.now()
                time_difference = now - last_entry_time

                if time_difference < timedelta(minutes=30):
                    next_allowed_time = last_entry_time + timedelta(minutes=30)
                    return jsonify({
                        "message": "Дані не вставлено. Будь ласка, зачекайте.",
                        "next_allowed_time": next_allowed_time.strftime('%Y-%m-%d %H:%M:%S')
                    }), 400

            # Отримання поточної погоди
            weather = get_weather()
            if not weather:
                return jsonify({"error": "Не вдалося отримати дані про погоду"}), 500

            # Вставка нових даних
            insert_query = """
                INSERT INTO sensor_parameters
                (current, voltage, temperature, humidity, power, weather_temperature, weather_humidity, weather_description)
                VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
            """
            data_tuple = (
                data.get('current'),
                data.get('voltage'),
                data.get('temperature'),
                data.get('humidity'),
                data.get('power'),
                weather['temperature'],
                weather['humidity'],
                weather['description']
            )
            cursor.execute(insert_query, data_tuple)
            connection.commit()
```

Рисунок 3.7 – Функція отримання даних від сенсорів

Примітка: Так як обидва контролери не синхронізуються і можуть одночасно або з невеликим періодом відправити дані, сервер самостійно фільтрує дані. Якщо період від останніх отриманих даних менша за поточний запит то сервер повертає час до прийому наступних даних.

В майбутньому даному серверу можна розширити функціонал, зробити статистичний аналіз отриманих даних та створити прогнозування виробництва електроенергії панеллю за період на базі статистичних даних.

Siemens CPU 1212. Даний контролер відповідає за переміщення слонячої панелі по горизонталі та вертикалі. Для даного контролера програма достатньо сильно модифікована. Програма була перенесена з TIA Portal 13 на більш нову версію TIA Portal 17.

Програма мала 2 режими. Переміщення за часом та за освітленістю від датчика освітленості.

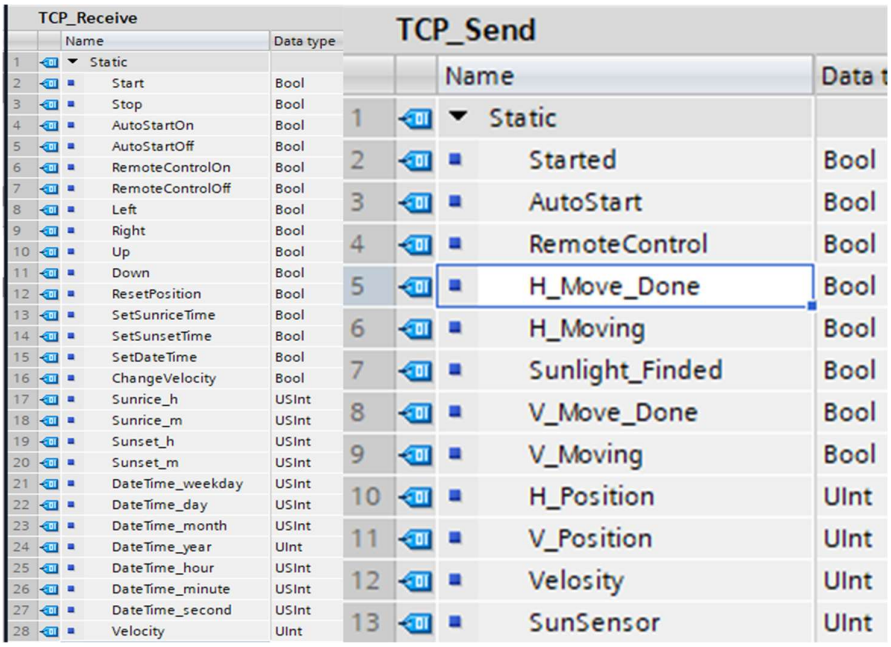
Режим за часом: контролер вираховував кут повороту враховуючи поточний час та час сходу і заходу сонця.

Рисунок 3.8 - Структура проєкту

Режим за освітленістю: контролер раз в пів години рухався до нульової позиції, по кроку рухався до 360 градусів запам'ятовуючи найбільш яскраве місце і там зупинявся.

Наразі обидва режими були поєднані в один. Минулі алгоритми не були видалені але були вимкнуті.

Був створений алгоритм обміну даними та управління в ручному режимі по командам по протоколу TCP/IP з контролером WT32-ETH01.



TCP_Receive

	Name	Data type
1	Static	
2	Start	Bool
3	Stop	Bool
4	AutoStartOn	Bool
5	AutoStartOff	Bool
6	RemoteControlOn	Bool
7	RemoteControlOff	Bool
8	Left	Bool
9	Right	Bool
10	Up	Bool
11	Down	Bool
12	ResetPosition	Bool
13	SetSunriceTime	Bool
14	SetSunsetTime	Bool
15	SetDateTime	Bool
16	ChangeVelocity	Bool
17	Sunrice_h	USInt
18	Sunrice_m	USInt
19	Sunset_h	USInt
20	Sunset_m	USInt
21	DateTime_weekday	USInt
22	DateTime_day	USInt
23	DateTime_month	USInt
24	DateTime_year	UInt
25	DateTime_hour	USInt
26	DateTime_minute	USInt
27	DateTime_second	USInt
28	Velocity	UInt

TCP_Send

	Name	Data t
1	Static	
2	Started	Bool
3	AutoStart	Bool
4	RemoteControl	Bool
5	H_Move_Done	Bool
6	H_Moving	Bool
7	Sunlight_Finded	Bool
8	V_Move_Done	Bool
9	V_Moving	Bool
10	H_Position	UInt
11	V_Position	UInt
12	Velosity	UInt
13	SunSensor	UInt

Рисунок 3.9 - Структури обміну даних CPU1212 та WT32-ETH01

Можливості контролю сонячної панелі

- Запуск, стоп сонячної панелі
- Автостарт – коли з’являється живлення автоматично запускається алгоритм
- Віддалений контроль – зупиняючий сигнал роботи сонячної панелі в режимі руху за сонце і дозволяючи сигнал управління віддалено.
- Скинути позицію – при фізичному переміщені сонячної панелі на інше місце, можливість скинути початкову позицію.
- Встановлення часу сходу і захочу сонця, перехід роботи панелі в нічний режим.
- Встановлення швидкості – при ручному русі
- Оновлення поточної дати часу, автоматично контролера.

Дані від контролеру :

- Запущена робота панелі
- Автостарт
- Переведений в віддалений контроль
- По горизонталі та вертикалі закінчився рух, рухається двигун

– Знайдене найбільш освітлене місце (алгоритм пошуку світла завершився)

– Швидкість двигунів, однакова на обидва.

– Дані з датчика освітленості

Передача даних по UART

Для організації обміну даними між пристроями WT32-ETH-01, Arduino та ESP8266 розроблено алгоритм передачі через UART

UART [34] — це апаратний модуль для реалізації асинхронного послідовного обміну даними між пристроями. Передача даних відбувається за принципом кадрів (frame), які містять службові біти для ідентифікації пристрою, команди, обсягу даних, самих даних і контролю їхньої цілісності.

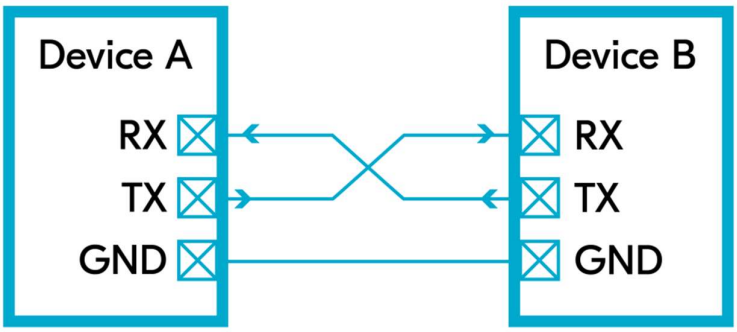


Рисунок 3.10 - Обмін даними по UART

Довжина пакету передачі становить 64 байти.

Таблиця 3.1 - Структура пакету

Номер байту	0	1	2	3-62	63
Дані	id пристрою	Id команди	Довжина даних	Дані	Чек сума

– ID пристрою (байт 0):

Унікальний ідентифікатор пристрою. Дозволяє визначити, якому пристрою адресовано повідомлення.

– ID команди (байт 1):

Використовується для визначення типу команди.

– Довжина даних (байт 2):

Вказує кількість байтів у полі "Дані".

Дозволяє приймаючому пристрою зрозуміти, скільки байтів потрібно зчитати.

– Дані (байти 3–62):

Основний блок інформації, що передається.

Може містити числові значення, текст або інші необхідні дані.

– Контрольна сума (байт 63):

Використовується для перевірки цілісності пакету.

Розраховується як сума всіх байтів пакету (окрім контрольної суми) за модулем 256:

$$chsum = \sum_{i=0}^{62} \text{байт}[i] \ \% \ 256 \qquad (3.1)$$

Для обміну даними по UART, на кожному пристрої був створений клас UartController.

```
enum uart_ids {
    WT32_ETH01 = 10,
    ESP8266,
    Arduino
};
```

Рисунок 3.11 - Номера пристроїв в UART

```
enum uart_data_types {
    solar_panel_feed_back_Type = 100,
    WIFI_Credentials_Type, //1
    ESP8266_Sensor_type, //2
    Arduino_Sensors_Type, //3
    Simence_TCP_Status, //4
    ESP8266_STATUS_OK, //5
    ARDUINO_STATUS_OK, //6
    ESP8266_Check_Connection, //7
    ARDUINO_Check_Connection, //8
    ARDUINO_ETH_Connection // 9
};
```

Рисунок 3.12 - Список команд передачі даних UART

Цей перелік визначає типи команд для передачі даних через UART. Він включає команди для зворотного зв'язку від сонячних панелей, передачі Wi-Fi підключення, даних із сенсорів ESP8266 та Arduino, перевірки статусів і підключень для ESP8266, Arduino та Ethernet.

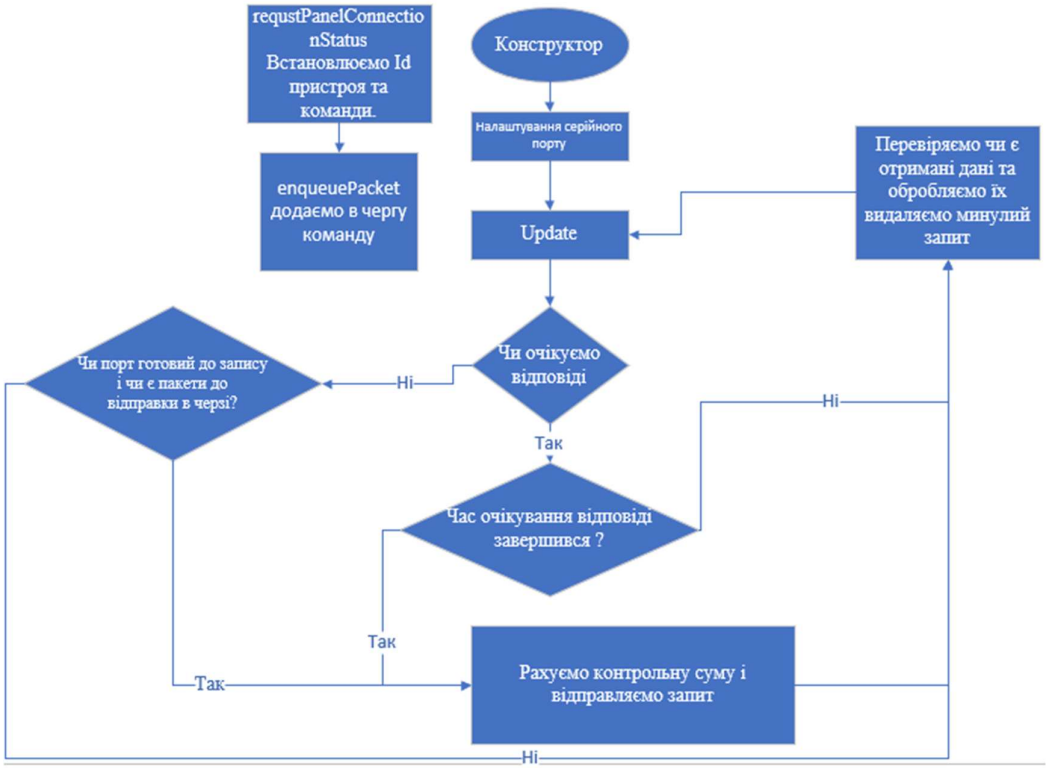


Рисунок 3.13 - Спрощений алгоритм передачі даних UART

WT32-ETH01. Даний контролер має одночасно модуль WIFI та RJ45 конектор. Завдяки цьому ми можемо його використовувати як для обміну даними по TCP/IP з PLC так і створювати локальний веб інтерфейс для керуванням панеллю.

Вибір протоколу TCP/IP був виправданий легкістю його використання. Звичайно для промисловості краще використовувати Modbus TCP або S7, але для даного контролера не було готової бібліотеки Modbus TCP, а S7 напряму звертається до DB, що є не найкращім варіантом.

WT32-ETH01 не має окремого usb порта для програмування. Для програмування даного контролера потрібен USB-TTL перетворювач.

Для реалізації проекту було використано PlatformIO — універсальне середовище для розробки вбудованих систем. Це рішення забезпечує ефективну розробку та тестування програмного забезпечення для мікроконтролерів різного типу.

Плюси PlatformIO:

– PlatformIO підтримує сотні плат, включаючи ESP8266, STM32, AVR, і WT32-ETH-01, тоді як Arduino IDE орієнтована в основному на Arduino-сумісні контролери.

– У PlatformIO код можна розділити на бібліотеки, файли та папки, забезпечуючи легкість у масштабуванні проекту. Arduino IDE зазвичай працює з монолітними скетчами.

– PlatformIO інтегрується з Visual Studio Code, що забезпечує функції автодоповнення, рефакторингу, дебагу та роботи з Git.

– PlatformIO дозволяє легко встановлювати, оновлювати та видаляти бібліотеки через вбудований менеджер.

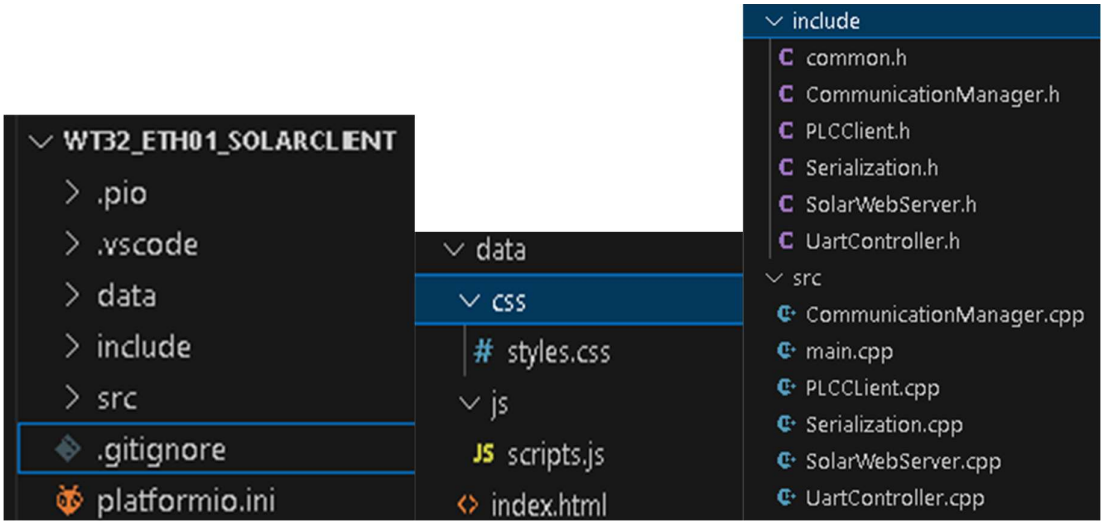


Рисунок 3.14 - Структура проєкту WT32-ETH01

– .pio - Використовується для зберігання об'єктних файлів, прошивок та інших артефактів компіляції.

- data/ - Папка для статичних файлів, які завантажується на пристрій у файлову систему LittleFS. Вміщує HTML, CSS, JavaScript, зображення для нашого web додатку
- include/ - Папка містить .h файли, які оголошують функції, змінні, класи, що використовуються в проекті.
- src/ Містить файл з головною програмою, зазвичай main.cpp, і додаткові .cpp файли для організації коду.

Даний контролер є віддаленим клієнтом. Він створений для підключення до мережі Wifi або точки доступу з доступом до інтернету. Його основне призначення — відображення статистичних графіків на екрані.

Керування за допомогою нього сонячною панеллю буде неможливо з міркувань безпеки. Веб інтерфейс WT32-ETH01 та ESP8266 схожі є схожими, окрім доданої панелі графіків та видаленої панелі контролю сонячною панеллю.

Arduino UNO Rev3 ATmega328P у поєднанні з Ethernet Shield W5100 виступає додатковим контролером у системі контролю сонячної панелі. Основні функції цього контролера включають збір даних з різноманітних датчиків та пасивну передачу зібраних даних на сервер через Ethernet Shield W5100 кожні 30 хвилин. Основним контролером системи є WT32, а Arduino UNO служить допоміжним вузлом для збору та передачі даних.

Основні функції контролера охоплюють збір даних з різних датчиків, передачу цих даних на сервер та комунікацію з основним контролером WT32. Для вимірювання температури та вологості використовується датчик DHT22, а для вимірювання струму та напруги застосовуються датчики ACS712 та аналоговий вхід відповідно. За потребою можливо підключення додаткових датчиків, що розширює функціональні можливості системи. Передача даних здійснюється за допомогою Ethernet Shield W5100, який забезпечує стабільне з'єднання з локальною мережею або Інтернетом.

Контролер автоматично відправляє зібрані дані на сервер кожні 30 хвилин, що підтримує актуальність інформації без необхідності ручного втручання. Для обміну інформацією між Arduino UNO та WT32 використовується UART-

з'єднання, що забезпечує інтеграцію даних з обох контролерів та дозволяє ефективно керувати всією системою.

Архітектура програмного забезпечення контролера складається з кількох ключових компонентів, що забезпечують його модульність та гнучкість. Файл main.cpp відповідає за ініціалізацію мережевого з'єднання, збір даних з датчиків та їх періодичну передачу на сервер. Файл sensor.h містить визначення та ініціалізацію датчиків, а також функції для збору даних з них, що дозволяє легко додавати нові сенсори при необхідності. Класи UartController.h та UartController.cpp відповідають за управління UART-з'єднанням, забезпечуючи надійну взаємодію з основним контролером WT32.

Така структура програмного забезпечення дозволяє легко масштабувати систему та адаптувати її до різних умов експлуатації, забезпечуючи високу ефективність та надійність роботи контролера.

Детальний опис основних частин коду

Функція collectSensorData() відповідає за зчитування даних з датчиків температури, вологості, струму та напруги. У разі успішного зчитування даних, вони зберігаються в структурі sensorData.

```
bool collectSensorData()
{
    humidity = dht.readHumidity();
    if (isnan(humidity))
    {
        return false;
    }
    sensorData.humidity = humidity;
    temperature = dht.readTemperature();
    if (isnan(temperature))
    {
        temperature = 0.0;
        return false;
    }
    sensorData.temperature = temperature;
    current = acs.mA_DC();
    sensorData.current = current;
    int analogValue = analogRead(VOLTAGE_PIN);
    voltage = analogValue * (5.0 / 1023.0);
    sensorData.voltage = voltage;
    return true;
}
```

Рисунок 3.15 – Функція зчитування даних з датчиків

Функція sendData() формує HTTP POST-запит з даними сенсорів та відправляє його на сервер за допомогою Ethernet Shield W5100. У разі успішного відправлення, виводиться повідомлення про успіх, інакше – про невдачу

```
void sendData()
{
  if (client.connect(server, port))
  {
    String postData = "humidity=" + String(sensorData.humidity) +
                      "&temperature=" + String(sensorData.temperature) +
                      "&current=" + String(sensorData.current) +
                      "&voltage=" + String(sensorData.voltage);

    client.println("POST /data HTTP/1.1");
    client.println("Host: " + String(server));
    client.println("Content-Type: application/x-www-form-urlencoded");
    client.print("Content-Length: ");
    client.println(postData.length());
    client.println();
    client.println(postData);

    DPrint("Дані успішно надіслані.");
  }
  else
  {
    DPrint("Не вдалося підключитися до сервера.");
  }
  client.stop();
}
```

Рисунок 3.16 – Функція відправки даними сенсорів на сервер

Функція checkInternetConnection() перевіряє можливість встановлення з'єднання з сервером, що підтверджує наявність Інтернет-з'єднання. Відповідно до результату, оновлюється статус підключення.

```
bool checkInternetConnection()
{
  DPrint("Перевірка підключення до Інтернету...");
  bool connected = client.connect(server, port);
  if (connected)
  {
    DPrint("Підключення до Інтернету успішне.");
    client.stop();
    return true;
  }
  else
  {
    DPrint("Немає підключення до Інтернету.");
    return false;
  }
}
```

Рисунок 3.17 – Функція перевірки з’єднання з сервером

Цикл програми перевіряє Ethernet і інтернет-з'єднання, збирає та відправляє дані сенсорів через UART або інтернет із заданими інтервалами, періодично перевіряючи стан UART.

```
void loop()
{
  unsigned long currentMillis = millis();

  if(!ethSetuped)
  {
    setup_eth();
  }
  else if(currentMillis - previousMillisInternet >= internetCheckInterval)
  {
    previousMillisInternet = currentMillis;
    isConnected = checkInternetConnection();
  }

  if (currentMillis - previousMillisSensor >= sensorInterval && uart.isUartConnected())
  {
    previousMillisSensor = currentMillis;
    if(collectSensorData())
    {
      uart.sendSensorData(sensorData);
    }
  }
```

Рисунок 3.18 – Головний цикл програми

Функція setup_eth() намагається налаштувати Ethernet-з'єднання за допомогою DHCP.

```
void setup_eth()
{
    if (Ethernet.begin(mac) == 0)
    {
        DPrint("Не вдалося налаштувати Ethernet за допомогою DHCP");

        if (Ethernet.hardwareStatus() == EthernetNoHardware)
        {
            DPrint("Ethernet shield не знайдено. Необхідно апаратне забезпечення.");
        }
        else if (Ethernet.linkStatus() == LinkOFF)
        {
            DPrint("Ethernet кабель не підключений.");
        }
        else
        {
            DPrint("Невідома помилка Ethernet.");
        }
    }

    if(checkInternetConnection())
    {
        ethSetuped = true;
    }

    DPrint("IP: ");
    DPrint(Ethernet.localIP());
}
```

Рисунок 3.19 – Функція налаштування DHCP

У випадку невдачі виводяться відповідні повідомлення про помилки. Якщо підключення успішне, оновлюється статус ethSetuped та виводиться IP-адреса контролера.

Підключення та конфігурація системи включає кілька етапів. Спочатку Ethernet Shield W5100 встановлюється на Arduino UNO, переконавшись, що всі контакти правильно з'єднані. Далі до Shield підключається Ethernet кабель, який під’єднується до маршрутизатора або комутатора мережі, забезпечуючи стабільне інтернет-з’єднання.

Конфігурація мережі може здійснюватися шляхом встановлення статичної IP-адреси або використанням DHCP для автоматичної налаштування мережевих параметрів. У файлі main.cpp визначаються IP-адреса сервера та порт для передачі даних, що дозволяє контролеру знати, куди саме відправляти зібрану інформацію. Після налаштування мережі та підключення всіх датчиків необхідно завантажити програму на Arduino UNO, після чого система автоматично почне збирати дані з датчиків та передавати їх на сервер кожні 30 хвилин.

Приклад роботи системи демонструє, як датчики вимірюють температуру, вологість, струм та напругу, після чого ці дані зчитуються та зберігаються у структурі sensorData. Кожні 30 хвилин зібрані дані відправляються на сервер за допомогою Ethernet Shield W5100 у форматі HTTP POST-запиту до зазначеного серверного адресата. Крім того, дані передаються через UART-з’єднання до основного контролера WT32, що забезпечує інтеграцію інформації з обох контролерів для подальшого аналізу та моніторингу.

Використання Arduino UNO Rev3 та Ethernet Shield W5100 має кілька переваг: надійність платформи забезпечується широкою підтримкою бібліотек та модулів, гнучкість дозволяє легко додавати нові датчики або модулі для розширення функціональності системи, а проста у використанні відкрита архітектура та доступність ресурсів сприяють швидкій розробці та впровадженню нових рішень. Крім того, економічність компонентів Arduino UNO та Ethernet Shield W5100 дозволяє знизити загальні витрати на розробку системи контролю, роблячи її доступною для широкого спектру застосувань.

Веб-інтерфейс системи контролю сонячної панелі розроблений для забезпечення зручного та інтуїтивно зрозумілого управління та моніторингу роботи фотоелектричних систем (ФЕС). Він має уніфіковану структуру для обох пристроїв – WT32-ETH01 та ESP8266, з невеликими відмінностями, що дозволяють виконувати специфічні функції кожного з них. Загальна структура інтерфейсу складається з трьох основних частин: Header, Main та Footer.

Header включає відображення назви системи контролю сонячної панелі, поточного статусу підключення до PLC (Programmable Logic Controller) зі статусами "PLC Online" або "PLC Offline", а також поточного часу для зручності користувача. Це дозволяє оперативно отримувати основну інформацію про стан системи без необхідності переходити до інших розділів інтерфейсу.

Основний вміст (Main) веб-інтерфейсу поділяється на кілька ключових секцій. Розділ "Поточні дані" (Current Panel) відображає актуальні параметри роботи сонячної панелі, такі як стан системи, режим роботи, напруга, струм, температура, вологість, дані з датчиків сонця та положення панелі. Це забезпечує користувачеві повну інформацію про поточний стан ФЕС в режимі реального часу. Секція "Керування панеллю" (Control Panel) дозволяє користувачам запускати та зупиняти систему, переключатися між ручним та автоматичним режимами, регулювати швидкість руху панелі, керувати її орієнтацією та здійснювати скидання позиції. Це надає гнучкість у управлінні системою відповідно до поточних потреб та умов експлуатації.

Додатково, розділ "Налаштування мережі" (Network Settings Panel) забезпечує можливість налаштування підключення до Wi-Fi мережі, що є необхідним для WT32-ETH01, дозволяючи легко інтегрувати систему в існуючу мережеву інфраструктуру. Секція "Графіки даних" (Charts Panel) відображає графіки температури, струму, напруги та потужності, які доступні лише на ESP8266, надаючи візуалізацію змін параметрів системи протягом часу для детального аналізу та моніторингу ефективності роботи ФЕС.

Footer (Футер) містить кнопки навігації, що дозволяють швидко перемикатися між різними секціями інтерфейсу, такими як "Поточні", "Керування",

"Мережа" та "Графіки" (останній доступний тільки на ESP8266). Це забезпечує зручність користування та швидкий доступ до необхідних функцій без зайвих кроків. Веб-інтерфейс також може бути розширений додатковими функціями, такими як повідомлення про помилки, історія подій та можливість налаштування сповіщень, що покращує загальну функціональність та користувацький досвід. Завдяки цьому інтерфейс стає потужним інструментом для ефективного управління та моніторингу сонячних панелей, забезпечуючи високу ступінь контролю та гнучкості для користувачів. Меню для мобільних пристроїв: Забезпечує доступ до навігаційних кнопок на малих екранах через кнопку "☰ Меню".

Основні функції веб-інтерфейсу. Веб-інтерфейс системи контролю сонячної панелі надає користувачам зручний та інтуїтивно зрозумілий спосіб моніторингу та управління роботою фотоелектричних систем (ФЕС). Основні функції інтерфейсу включають відображення поточних даних, керування панеллю та інтеграцію з різними пристроями системи.

У розділі "Поточні дані" користувачі можуть бачити ключові параметри роботи сонячної панелі в реальному часі.

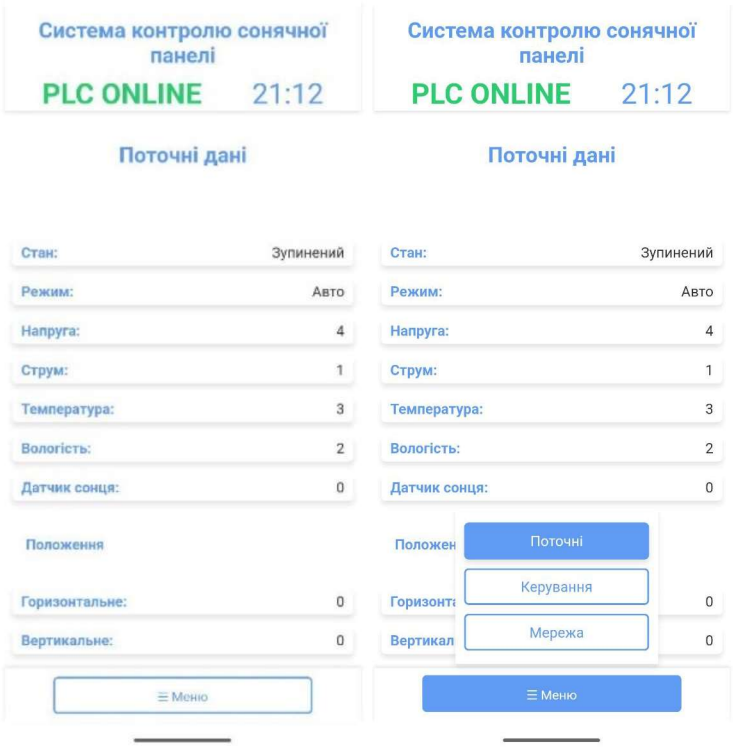


Рисунок 3.20 - Головне вікно веб-додатку

Це включає індикацію стану системи (запущена або зупинена) та режиму роботи (ручний або автоматичний), а також поточні значення напруги, струму, температури, вологості та даних з датчика сонця.

Крім того, відображається позиція панелі по горизонтальній та вертикальній осях, що дозволяє користувачам оперативно оцінювати її орієнтацію та ефективність роботи.

```html
<!-- Фрагмент HTML-коду для відображення поточних даних -->
<section class="current-panel" id="current-panel">
    <h2>Поточні дані</h2>
    <div class="grid-container">
      <div class="grid-item">
        <div class="label">Стан:</div>
        <div class="value" id="state-value">НД</div>
      </div>
      <!-- Інші параметри... -->
    </div>
  </section>
```

```css
/*CSS-стилі для сітки поточних даних.*/
.grid-container {
    display: grid;
    grid-template-columns: repeat(2, 1fr);
    gap: 15px;
    width: 100%;
  }

  .grid-item {
    background-color: var(--background-color);
    padding: 5px 10px;
    border-radius: 5px;
    display: flex;
    justify-content: space-between;
    align-items: center;
    box-shadow: var(--box-shadow);
    transition: transform var(--transition-speed);
  }

  .grid-item:hover {
    transform: translateY(-5px);
  }

  .label {
    font-weight: bold;
    color: var(--primary-color);
  }

  .value {
    text-align: right;
    color: var(--text-color);
  }
```

Рисунок 3.21 – Класи і стилі поточних даних

Панель "Керування панеллю", доступний лише на контролері WT32-ETH01, надає користувачам можливість безпосередньо взаємодіяти з сонячною панеллю.

Рисунок 3.22 - Головне вікно керування

Основні інструменти управління включають кнопки Start та Stop для запуску та зупинки системи, перемикання між ручним та автоматичним режимами роботи, а також регулювання положення панелі за допомогою кнопок для переміщення її вгору, вниз, вліво та вправо. Ці функції дозволяють користувачам гнучко налаштовувати роботу системи відповідно до поточних умов та потреб:

Автозапуск: Можливість автоматичного запуску системи при певних умовах, таких як досягнення заданої температури або освітленості, забезпечує безперервну роботу без необхідності ручного втручання.

Контроль швидкості: Регулювання швидкості руху панелі за допомогою повзунка дозволяє оптимізувати її позиціювання для максимального збору енергії та зменшення зносу механізмів.

Скидання позиції: Кнопка для скидання панелі до початкової позиції забезпечує швидке повернення системи до стандартного стану, що корисно у випадках непередбачених змін або помилок у налаштуваннях.

Нульова позиція: Кнопка, що повертає панель на нульову позицію, дозволяє швидко відновити базове положення панелі для початку нових циклів роботи.

Темно сірі кнопки означають, що немає підключення до PLC (Programmable Logic Controller) або що ручний режим керування не активний.

```
startButton.addEventListener('click', () => {
    if (!isPLCConnected) {
        console.error('Cannot send command start: No connection to PLC.');
        return;
    }
    controlPanelAPI('start');
});

stopButton.addEventListener('click', () => {
    if (!isPLCConnected) {
        console.error('Cannot send command stop: No connection to PLC.');
        return;
    }
    controlPanelAPI('stop');
});

manualButton.addEventListener('click', () => {
    if (isPLCConnected) {
        if (!panelStatus.ManualMode) {
            controlPanelAPI('remoteOn');
        } else {
            controlPanelAPI('remoteOff' );
        }
    } else {
        console.error('Cannot send command remote: No connection to PLC.');
    }
});
```

Рисунок 3.23 - Фрагмент JavaScript-коду для обробки кнопок керування

Розділ "Налаштування мережі" доступний виключно на контролері WT32-ETH01 та дозволяє користувачам конфігурувати параметри підключення до Wi-Fi мережі або локальної точки доступу. У цьому розділі передбачені поля для введення SSID (назви мережі) та пароля, що необхідні для встановлення стабільного та безпечного з'єднання.

Після введення необхідних даних користувачі можуть зберегти налаштування за допомогою відповідної кнопки "Зберегти налаштування", яка забезпечує збереження введених мережевих параметрів у пам'яті контролера. Це дозволяє системі автоматично підключатися до зазначеної мережі при кожному

запуску, забезпечуючи безперервну роботу та доступність веб-інтерфейсу для моніторингу та управління сонячною панеллю.

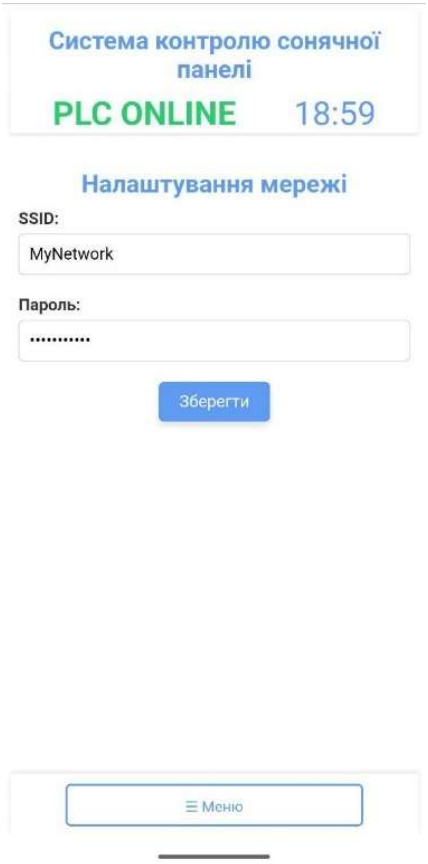


Рисунок 3.24 – Вікно «Мережа»

```
<section class="network-settings-panel" id="network-settings" style="display: none;">
    <h2>Налаштування мережі</h2>
    <div class="network-container">
        <div class="network-input">
            <label for="ssid-input"><strong>SSID:</strong></label>
            <input type="text" id="ssid-input" placeholder="Введіть SSID" required>
        </div>
        <div class="network-input">
            <label for="password-input"><strong>Пароль:</strong></label>
            <input type="password" id="password-input" placeholder="Введіть пароль" required>
        </div>
        <div class="save-button-container">
            <button id="save-network-button" class="save-button">Зберегти</button>
        </div>
    </div>
</section>
```

Рисунок 3.25 - Фрагмент HTML-коду для налаштування мережі:

```
saveNetworkButton.addEventListener('click', () => {
    const ssid = ssidInput.value.trim();
    const password = passwordInput.value.trim();
    if (ssid === '' || password === '') {
        alert('Будь ласка, заповніть всі поля.');
        return;
    }
    const credentials = {
        ssid: ssid,
        password: password
    };
    fetch(API_NETWORK_SAVE, {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify(credentials)
    })
    .then(response => {
        if (response.ok) {
            alert('Налаштування мережі збережено успішно!');
            ssidInput.value = '';
            passwordInput.value = '';
        } else {
            alert('Помилка при збереженні налаштувань мережі.');
        }
    })
    .catch(error => {
        console.error('Error:', error);
        alert('Сталася помилка при збереженні налаштувань мережі.');
    });
});
```

Рисунок 3.26 - JavaScript-код для збереження мережевих налаштувань

Розділ "Графіки даних" доступний тільки на ESP8266 та включає лінійні графіки для відображення змін температури, струму, напруги та потужності протягом часу. Графіки динамічно оновлюються в реальному часі, забезпечуючи актуальну інформацію про стан системи.

```
<section class="charts-panel" id="charts-panel" style="display: none;">
    <h2>Графіки даних</h2>
    <div class="charts-container">
        <canvas id="temperatureChart"></canvas>
        <canvas id="currentChart"></canvas>
        <canvas id="voltageChart"></canvas>
        <canvas id="powerChart"></canvas>
    </div>
```

Рисунок 3.27 - Фрагмент HTML-коду для графіків

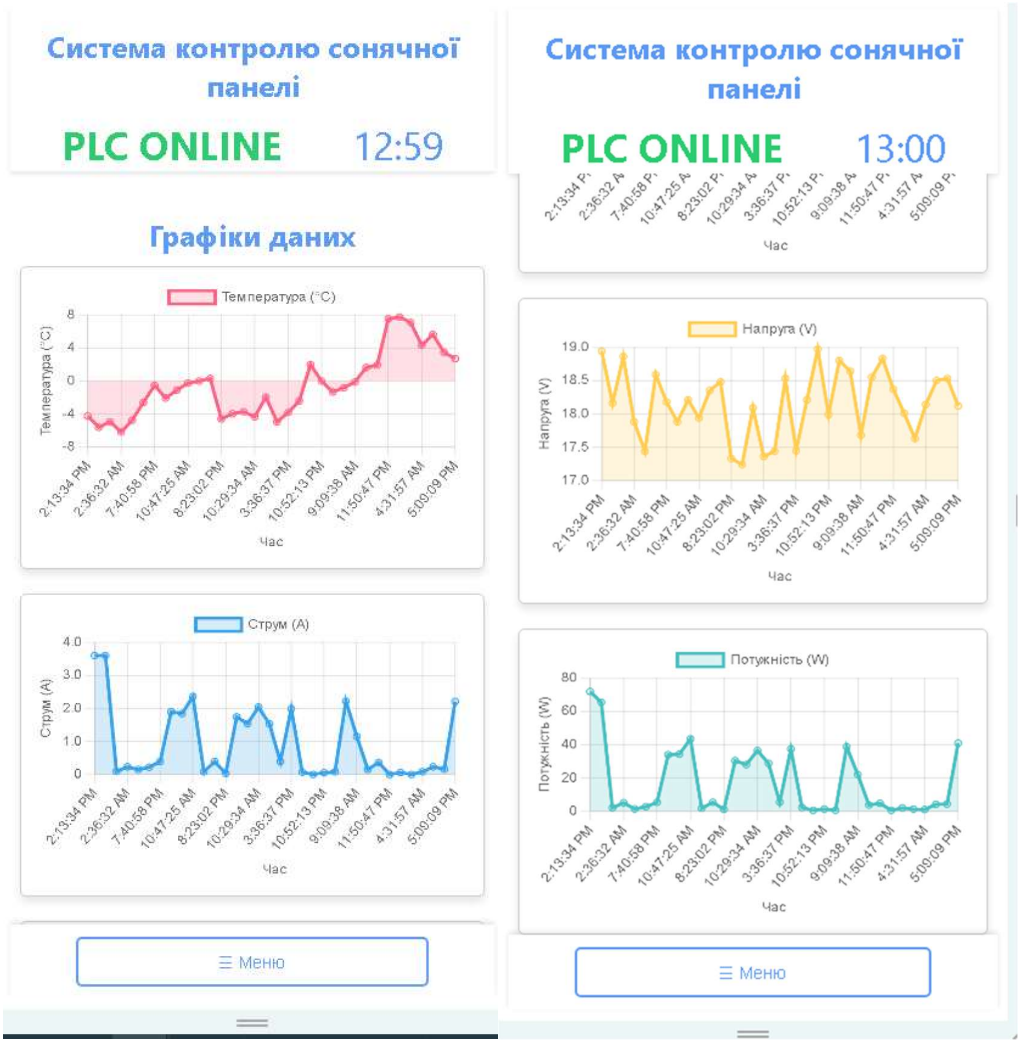


Рисунок 3.28 – Вікно «Графіки»

Ця функція updateChart оновлює дані та мітки (labels) для графіка (об'єкта chart) і викликає його метод update() для відображення змін.

```
function updateChart(chart, labels, data) {
    chart.data.labels = labels;
    chart.data.datasets[0].data = data;
    chart.update();
}
```

Рисунок 3.29 - Фрагмент HTML-коду для графіків

Функція fetchAndPopulateCharts отримує дані з API, витягує мітки часу та показники (температура, струм, напруга, потужність) і оновлює відповідні графіки. У разі помилки виводить її в консоль.

```
function fetchAndPopulateCharts() {
    fetch(`${API_SENSOR_DATA}?period=day`)
        .then(response => {
            if (!response.ok) {
                console.error(`HTTP error! Status: ${response.status}`);
                return;
            }
            return response.json();
        })
        .then(data => {
            if (!data) return;
            const labels = data.map(entry => new Date(entry.created_at).toLocaleTimeString());
            const temperatures = data.map(entry => entry.temperature);
            const currents = data.map(entry => entry.current);
            const voltages = data.map(entry => entry.voltage);
            const powers = data.map(entry => entry.power || (entry.voltage * entry.current));

            updateChart(temperatureChartInstance, labels, temperatures);
            updateChart(currentChartInstance, labels, currents);
            updateChart(voltageChartInstance, labels, voltages);
            updateChart(powerChartInstance, labels, powers);
        })
        .catch(error => {
            console.error('Error fetching data for charts:', error);
        });
}
```

Рисунок 3.30 - Фрагмент HTML-коду для графіків

Відмінності між WT32-ETH01 та ESP8266

WT32-ETH01 та ESP8266 виконують різні функції в системі контролю сонячної панелі, маючи суттєві відмінності у своїх можливостях та призначенні. WT32-ETH01 діє як локальна точка доступу, забезпечуючи підключення ESP8266 до мережі, що є ключовим для стабільної комунікації між компонентами системи. На відміну від цього, ESP8266 не має можливості налаштовувати підключення до мережі самостійно, оскільки ця функція делегована WT32-ETH01. WT32-ETH01 також має окрему секцію для введення параметрів Wi-Fi мережі, що дозволяє користувачам легко конфігурувати мережеві налаштування безпосередньо через цей контролер.

Крім того, WT32-ETH01 обладнаний повним набором інструментів для управління сонячною панеллю, включаючи кнопки Start та Stop, Manual Mode, регулювання положення та інші функції, що дозволяють здійснювати повний контроль над системою. У той же час, ESP8266 не включає секцію для управління

сонячною панеллю, оскільки ця функціональність реалізована на WT32-ETH01. Проте ESP8266 має свої унікальні переваги, зокрема секцію з графіками для моніторингу температури, струму, напруги та потужності, що забезпечує візуалізацію даних та спрощує аналіз продуктивності системи.

Таким чином, WT32-ETH01 відповідає за основне управління та налаштування мережі, забезпечуючи інтеграцію та контроль системи, тоді як ESP8266 спеціалізується на візуалізації даних, надаючи користувачам можливість детально моніторити ключові параметри роботи сонячної панелі. Така розподіленість функцій між контролерами підвищує ефективність та гнучкість системи, дозволяючи кожному компоненту виконувати свої специфічні завдання з максимальною продуктивністю. Розроблений веб-інтерфейс забезпечує ефективний моніторинг та управління сонячними панелями, враховуючи специфічні функції кожного з пристроїв. Інтуїтивно зрозумілий дизайн та адаптивність роблять його зручним для користувачів з різними технічними навичками, сприяючи підвищенню ефективності роботи фотоелектричних систем та забезпеченню надійного контролю над їхньою діяльністю.

Ключові переваги веб-інтерфейсу системи контролю сонячної панелі полягають у його інтуїтивно зрозумілому дизайні, який забезпечує легке використання та дозволяє швидко отримувати необхідну інформацію та здійснювати управління системою. Адаптивність інтерфейсу підтримує різні розміри екранів, що гарантує комфортне користування як на настільних комп’ютерах, так і на мобільних пристроях, забезпечуючи доступність з будь-якого місця. Візуалізація даних у вигляді графіків на ESP8266 надає наочне представлення продуктивності системи, спрощуючи аналіз інформації та прийняття обґрунтованих рішень. Крім того, гнучкість веб-інтерфейсу дозволяє налаштовувати параметри мережі та керувати панеллю відповідно до потреб користувача, що забезпечує високу адаптивність та ефективність використання системи в різних умовах експлуатації. Ці переваги роблять веб-інтерфейс потужним інструментом для моніторингу та управління сонячними панелями, підвищуючи загальну зручність та функціональність системи.

Додаткові функції та можливості

Повідомлення про відсутність з'єднання. Інтерфейс інформує користувача про стан підключення до мережі, що дозволяє вчасно виявляти та вирішувати проблеми з підключенням.

Автоматичний запуск. Функція автозапуску дозволяє системі автоматично запускатися при старті, забезпечуючи безперебійну роботу без необхідності ручного втручання.

Контроль швидкості. Регулювання швидкості руху панелі при ручному керуванні панелі.

Динамічне керування. Використання JavaScript для динамічного оновлення даних та управління елементами інтерфейсу забезпечує високу інтерактивність та зручність користування.

3.4 Налаштування та оптимізація програмного забезпечення

Для розробки програмного забезпечення системи керування сонячною панеллю було обрано PlatformIO як основне середовище розробки. PlatformIO є універсальним інструментом, який підтримує велику кількість мікроконтролерів та платформ, що робить його ідеальним вибором для даного проекту. Інтеграція PlatformIO з Visual Studio Code дозволила використовувати потужні функції автодоповнення, рефакторингу коду, налагодження та управління версіями через Git.

Прошивки для різних мікроконтролерів у системі керування сонячною панеллю були розроблені з використанням спеціалізованих бібліотек, що забезпечують ефективну взаємодію з датчиками та мережевими модулями. Для Arduino Uno Rev3 використовувалися бібліотеки для роботи з датчиками DHT22, ACS712 та дільник напруги, що дозволило здійснити ініціалізацію датчиків та Ethernet Shield W5100, збір даних з температури, вологості, напруги та струму, а також формування і відправку HTTP POST-запитів на сервер кожні 30 хвилин. Крім

того, прошивка обробляє відповіді від сервера та оновлює статус підключення, забезпечуючи надійну комунікацію та актуальність даних.

NodeMCU ESP8266 CH340 була оптимізована для бездротової передачі даних та відображення інформації на веб-інтерфейсі. Основні функції прошивки включають підключення до Wi-Fi мережі та налаштування точки доступу, отримання даних з сервера через REST API, відображення графіків температури, струму, напруги та потужності за допомогою бібліотеки Chart.js, а також відправку команд на WT32-ETHO1 для управління сонячною панеллю. Це дозволяє забезпечити інтерактивний моніторинг та управління системою через веб-інтерфейс, підвищуючи зручність користування та гнучкість налаштувань.

WT32-ETHO1 виконує роль головного контролера системи, забезпечуючи централізоване управління та інтеграцію даних з інших мікроконтролерів. Прошивка для WT32-ETHO1 включає ініціалізацію Ethernet та Wi-Fi з'єднань, обробку отриманих даних від Arduino Uno Rev3 та NodeMCU ESP8266 через UART, а також управління сонячною панеллю на основі отриманих команд, таких як запуск, зупинка та регулювання швидкості. Крім того, прошивка відповідає за надсилання зворотних даних на сервер для подальшого аналізу, що забезпечує повний цикл моніторингу та управління системою. Така інтеграція дозволяє ефективно координувати роботу всіх компонентів системи, забезпечуючи високу продуктивність та надійність роботи сонячних панелей.

Загалом, розробка прошивок для мікроконтролерів була спрямована на забезпечення безперебійної роботи системи, ефективного збору та передачі даних, а також гнучкого управління сонячною панеллю. Використання спеціалізованих бібліотек та оптимізація коду дозволили досягти високої продуктивності та надійності, що є критично важливими для успішної експлуатації фотоелектричних систем у реальних умовах.

Для забезпечення ефективної роботи системи було здійснено оптимізацію коду на різних рівнях. Використання мінімалістичних бібліотек та оптимізація алгоритмів збору даних дозволило зменшити споживання оперативної пам'яті на мікроконтролерах.

Покращення швидкості обробки та енергозбереження були досягнуті завдяки реалізації асинхронних процесів для збору та передачі даних, що забезпечило швидку реакцію системи на зміни параметрів. Це дозволяє системі миттєво адаптуватися до нових умов, підвищуючи її ефективність та продуктивність. Впровадження режимів енергозбереження на WT32-ETHO1 сприяло значному зниженню енергоспоживання системи під час неактивності, що продовжує термін служби обладнання та зменшує витрати на енергоресурси.

Для гарантування цілісності та надійності передачі даних було впроваджено кілька механізмів. Кожен пакет даних містить контрольну суму, що дозволяє перевірити їх цілісність та правильність отримання, забезпечуючи безпеку та достовірність інформації. У випадку невдалої спроби передачі даних система автоматично здійснює повторні спроби надіслати пакет протягом заданого проміжку часу, що підвищує ймовірність успішної доставки навіть у складних умовах мережі. Крім того, всі операції з передачі даних логуються на сервері, що дозволяє відслідковувати можливі помилки та аналізувати їх причини, сприяючи постійному вдосконаленню системи та забезпеченню її стабільної роботи. Ці заходи разом забезпечують високу продуктивність, ефективне енергоспоживання та надійну передачу даних, що є критично важливими для успішного функціонування системи контролю сонячної панелі.

Серверна частина системи на базі Python Flask була налаштована для обробки вхідних даних від мікроконтролерів та взаємодії з базою даних MySQL. Основні аспекти інтеграції включають:

API кінцеві точки. Реалізація REST API для прийому даних від Arduino Uno Rev3 та WT32-ETHO1, а також для надання даних веб-інтерфейсу.

Обробка запитів. Сервер обробляє HTTP POST-запити з даними сенсорів, зберігає їх у базі даних та відповідає клієнтам статусами успішності операцій.

Компоненти системи взаємодіють між собою за допомогою чітко визначених протоколів та інтерфейсів:

− UART-з'єднання між Arduino та WT32-ETHO1 забезпечує передачу даних сенсорів та отримання команд управління сонячною панеллю.

– TCP/IP з'єднання між WT32-ETHO1 та сервером забезпечує стабільну передачу даних на сервер та отримання оновлень для управління системою.

– Wi-Fi з'єднання NodeMCU ESP8266 з WT32-ETHO1 дозволяє бездротовий доступ до веб-інтерфейсу та відображення графіків даних.

Після завершення налаштування та інтеграції всіх компонентів було проведено комплексне тестування інтегрованої системи для перевірки її функціональності та стабільності. Функціональні тести включали перевірку коректності збору даних з датчиків, їх передачі на сервер та відображення на веб-інтерфейсі, що забезпечує точність та надійність отримуваної інформації. Навантажувальні тести були спрямовані на оцінку продуктивності системи при високих обсягах даних та одночасних запитах від кількох клієнтів, що дозволило визначити межі її ефективності та виявити потенційні вузькі місця. Тести на надійність перевіряли роботу системи в умовах нестабільного з'єднання з мережею Ethernet та Wi-Fi, а також функції відновлення після збоїв, що гарантує безперервну роботу навіть у випадку тимчасових втрат зв'язку.

Крім того, безпекові тести оцінювали ефективність впроваджених заходів безпеки, таких як аутентифікація, шифрування даних та обмеження доступу до API, що забезпечує захист системи від несанкціонованого доступу та потенційних кіберзагроз. Проведене тестування дозволило виявити та усунути можливі недоліки, підвищивши загальну надійність та безпеку системи перед її впровадженням у реальні умови експлуатації. Таким чином, комплексний підхід до тестування забезпечив високу якість та готовність системи до ефективної роботи в реальних умовах.

Після успішного завершення тестування було проведено низку підготовчих заходів для впровадження системи у реальні умови експлуатації. Перш за все, було здійснено налаштування серверного середовища, яке включало оптимізацію параметрів сервера для забезпечення максимальної продуктивності та високого рівня безпеки даних. Це дозволило гарантувати стабільну роботу системи під навантаженням та захистити її від потенційних загроз. Далі було проведено міграцію даних, що передбачала перенесення інформації з тестових баз даних у

виробничу базу даних MySQL. Цей процес забезпечив безперервність доступу до необхідних даних та підготував систему до роботи в реальних умовах.

Крім технічних налаштувань, велике значення мала підготовка документації. Було оформлено детальну технічну документацію та інструкції для користувачів системи, що значно полегшило подальше використання та обслуговування системи. Документація охоплює всі аспекти роботи системи, включаючи встановлення, налаштування, експлуатацію та вирішення можливих проблем, що забезпечує користувачам необхідні знання для ефективного використання системи.

Не менш важливим етапом стало навчання користувачів. Було організовано проведення навчальних сесій, під час яких користувачі ознайомилися з функціональними можливостями веб-інтерфейсу та основами управління сонячною панеллю. Це навчання дозволило користувачам швидко адаптуватися до нової системи, зрозуміти її переваги та навчитися ефективно використовувати всі доступні функції для моніторингу та управління фотоелектричними системами.

Завдяки цим підготовчим заходам система була успішно впроваджена у реальні умови експлуатації, забезпечуючи надійну та ефективну роботу сонячних панелей. Подальше впровадження включатиме моніторинг роботи системи, збір зворотного зв'язку від користувачів та внесення необхідних покращень для підвищення її функціональності та зручності використання.

На основі отриманого досвіду та зворотного зв'язку від користувачів було визначено можливості для подальшого вдосконалення системи.

Додаткові датчики. Розширення системи збору даних шляхом підключення додаткових датчиків для моніторингу параметрів, таких як атмосферний тиск або рівень шуму.

Аналітика та прогнозування. Впровадження алгоритмів машинного навчання для аналізу зібраних даних та прогнозування виробництва електроенергії.

Автоматизація управління. Впровадження автоматизованих сценаріїв управління сонячною панеллю на основі умов навколишнього середовища та запитів користувачів.

3.5 Оплата послуг хостингу

Для оплатити VPS хостингу переходимо на сайт [35].

Авторизуємося на сервісі з логіном та паролем.

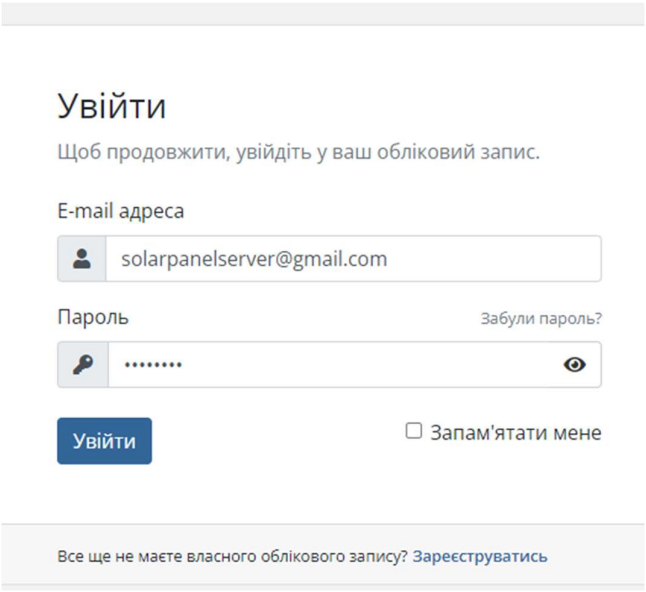


Рисунок 3.31 – Вікно авторизації

Наразі ми знаходимося в панелі керування

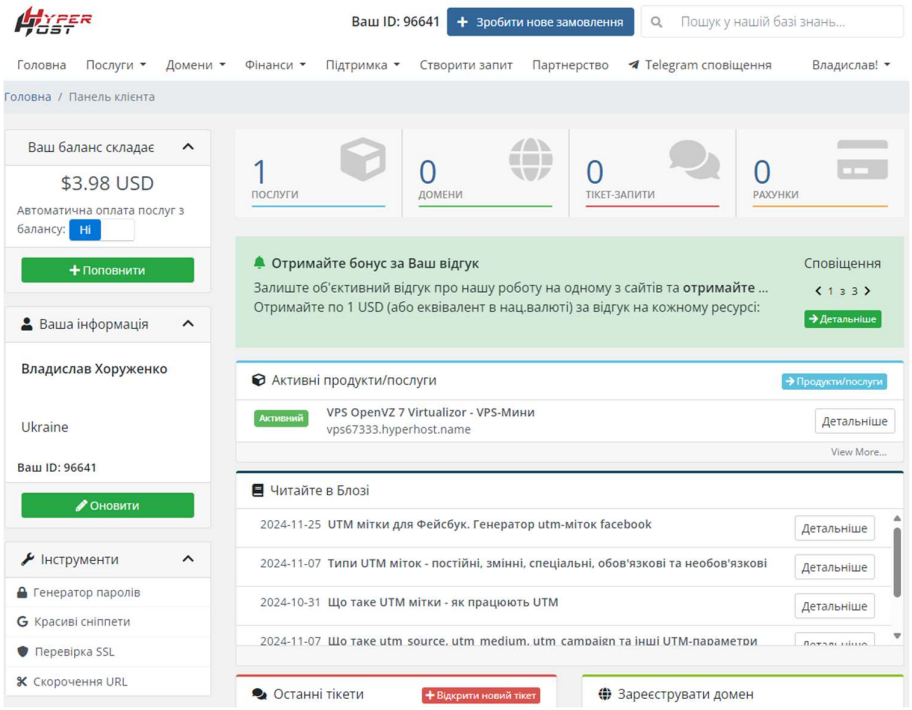


Рисунок 3.32 – Панель керування

Наразі період оплати становить По місячно. Є можливість змінити період оплати за раз. В залежності від буде знижка на оплату хостингу, Щомісячно немає, квартально -3%, піврічно -6%, щорічно -12%, дворічно -24%.

Для того щоб змінити період плати натискаємо ПОСЛУГИ.

Далі переходимо «Детальніше»

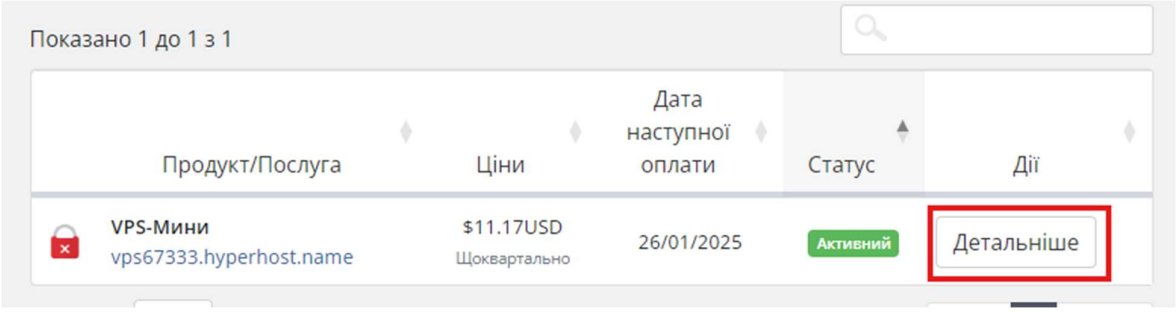


Рисунок 3.33 – Продукти та послуги

Можемо побачити деталі послуги. Натискаємо «Ціна/Період». Вибираємо потрібний період і натискаємо «Змінити тариф/Пакет»

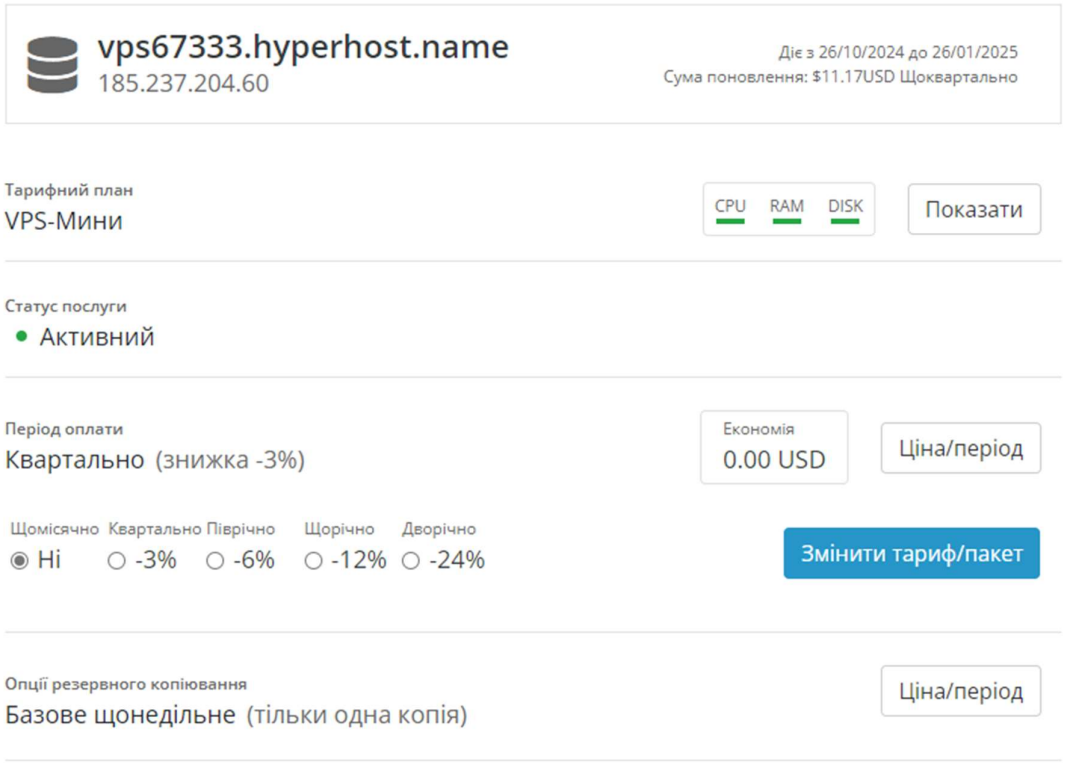


Рисунок 3.34 – Деталі послуги

Зліва ми можемо бачити наш поточний баланс. Для того щоб сплатити хостинг натискаємо кнопку «Поповнити»

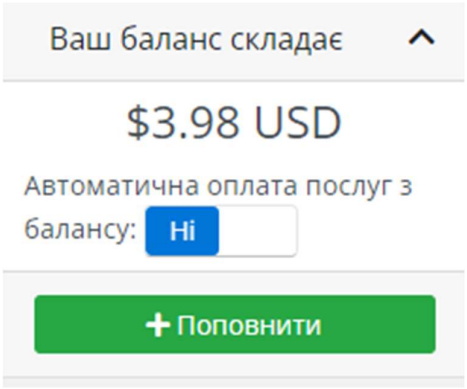


Рисунок 3.35 – Вікно балансу

Мінімальна сума поповнення сягає $15.15.

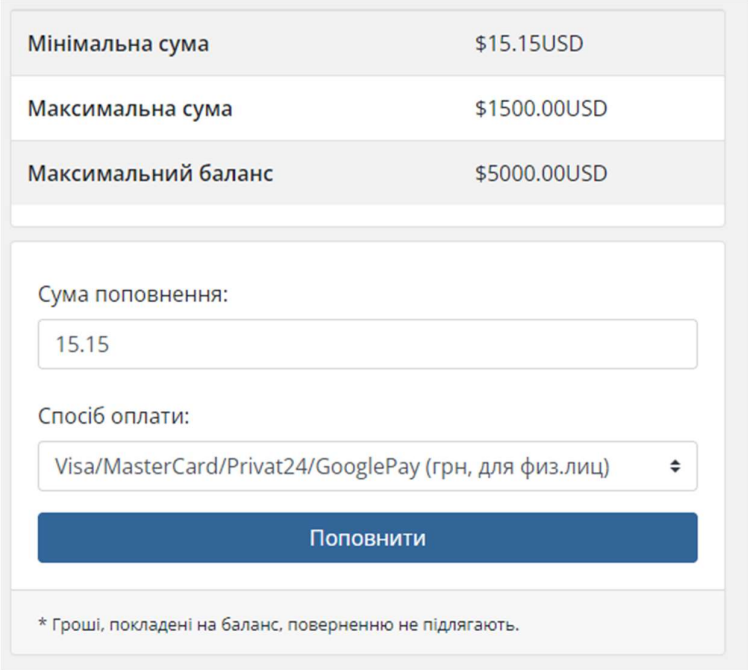


Рисунок 3.36 – Вікно «Поповнити»

Вписуємо потрібну суму поповнення, вибираємо спосіб яким зручно оплачувати та оплачуємо послугу. Після оплати на балансі повинні з’явитися кошти.

Повертаємося в вікно «Деталі послуги», спустившись вниз сторінки, ми можемо побачити поточну інформацію на сервері.

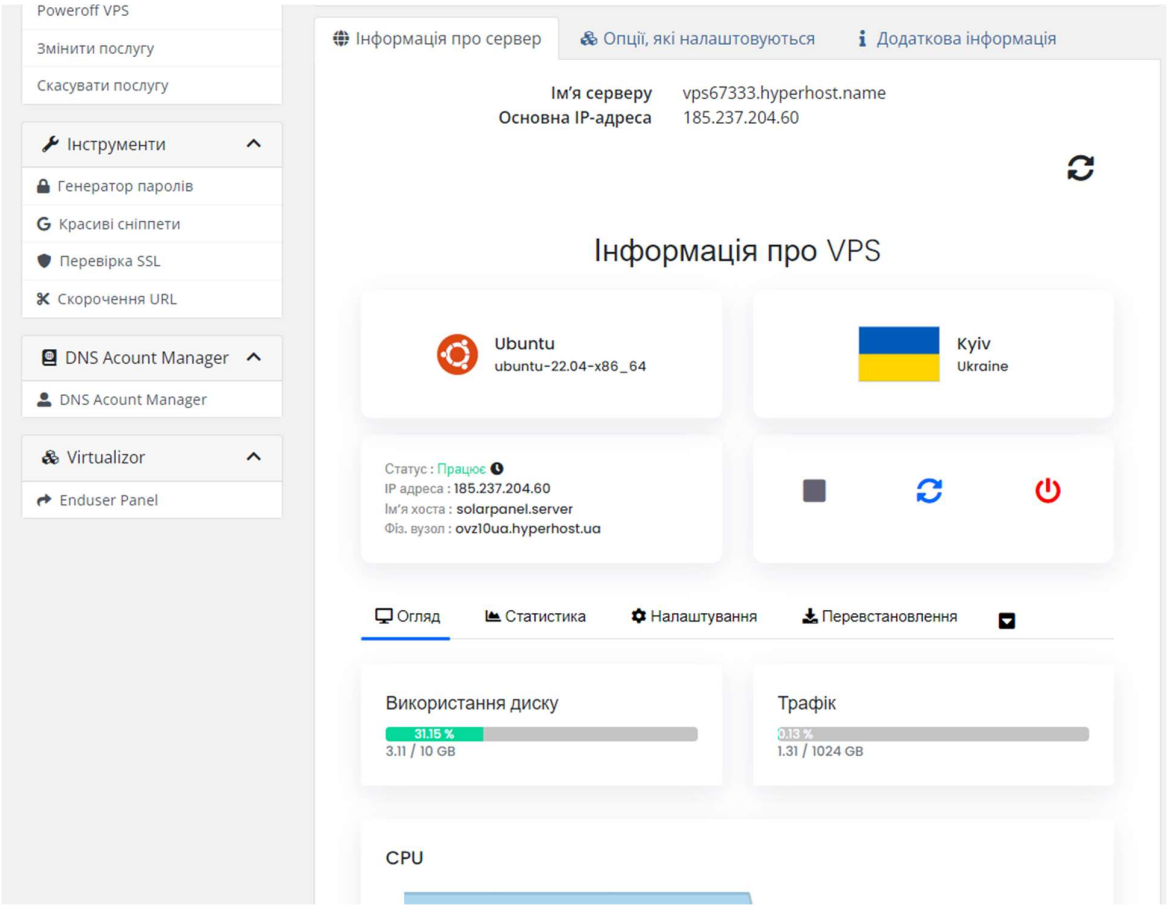


Рисунок 3.37 – Вікно «Інформація про сервер»

Тут ми можемо перевірити чи увімкнений сервер, статистику його роботи, трафік і так далі. Для того щоб більш детально керувати хостингом, зліва в вкладці «Virtualizor», натискаємо Enduser Panel

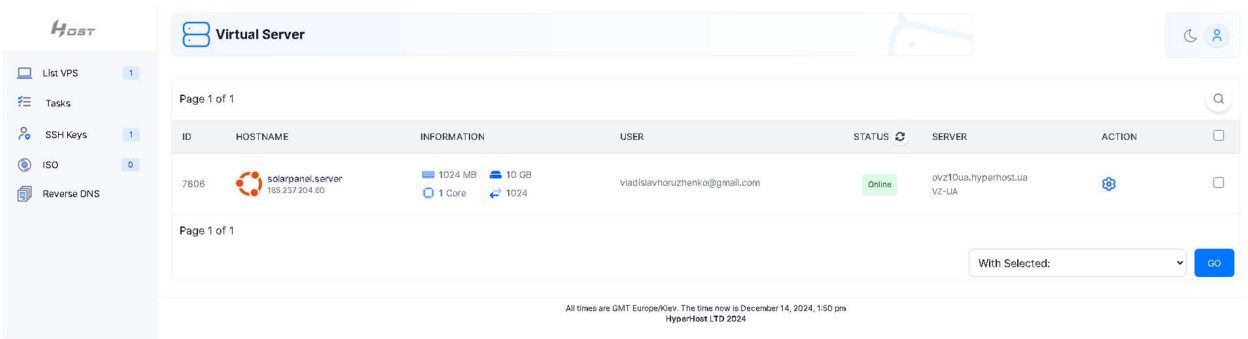


Рисунок 3.38 – Вікно «Віртуальний сервер»

3.6 Адміністрування серверної частини системи

Для ефективного управління серверною частиною системи було використано програму MobaXterm, яка є потужним та зручним інструментом для роботи з SSH (Secure Shell). Альтернативним варіантом може служити програма PuTTY, яка також широко використовується для підключення до серверів

MobaXterm дозволяє безпечно підключатися до серверів, передавати файли та виконувати адміністративні задачі, спрощуючи процес адміністрування серверу HyperHost.

Завантаження та встановлення MobaXterm здійснюється шляхом переходу на офіційний сайт MobaXterm [36], де можна завантажити останню версію програми та встановити її, дотримуючись інструкцій інсталятора.

Генерація SSH-ключа здійснюється через вбудований інструмент MobaKeyGen у MobaXterm. Для цього необхідно відкрити MobaXterm, натиснути на меню "Tools" та вибрати "MobaKeyGen (SSH key generator)". У вікні генератора вибирається тип ключа RSA або Ed25519, після чого натискається "Generate" та рухається мишкою, поки ключ буде створено.

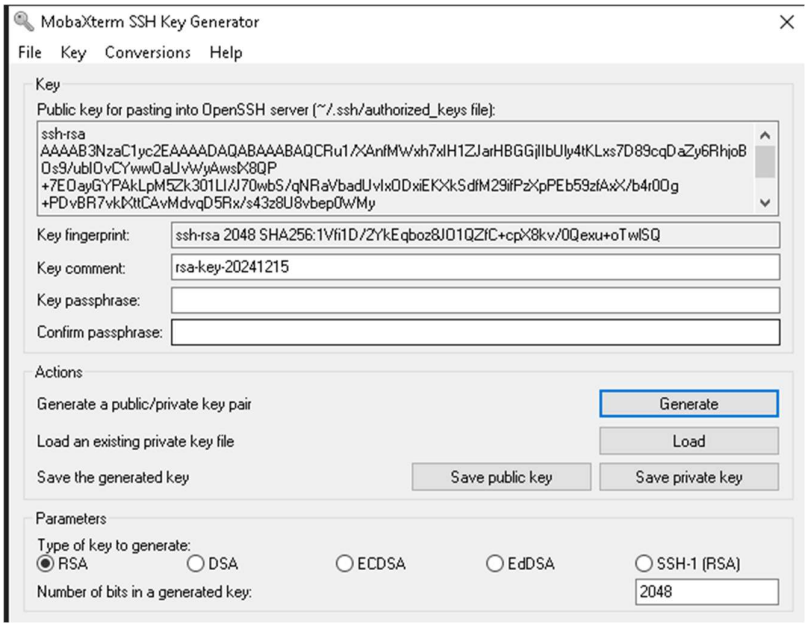


Рисунок 3.39 - Вікно генерації SSH ключа

Після генерації можна додати коментар (опціонально) та захистити приватний ключ паролем(ключовим словом). Після цього зберігається приватний ключ на комп'ютері, а публічний ключ копіюється для подальшого використання.

Додавання публічного ключа до серверу HyperHost здійснюється через веб-інтерфейс. Після авторизації на панелі управління HyperHost необхідно перейти до розділу «SSH-keys», натискаємо Add SSH Key, вставть скопійований публічний ключ та збережіть зміни.

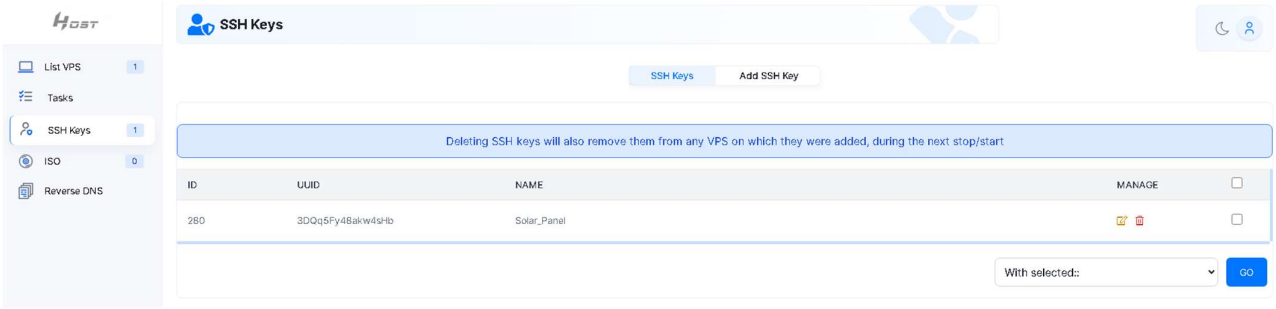


Рисунок 3.40 – Вікно додавання SSH ключу.

Налаштування сеансу SSH з використанням SSH-ключів у MobaXterm включає створення нового сеансу SSH. Для цього відкривається MobaXterm, натискається кнопка "Session", вибирається тип сеансу "SSH", вводиться IP-адреса серверу 185.237.204.60, порт 22, а також ім'я користувача, наприклад, root.

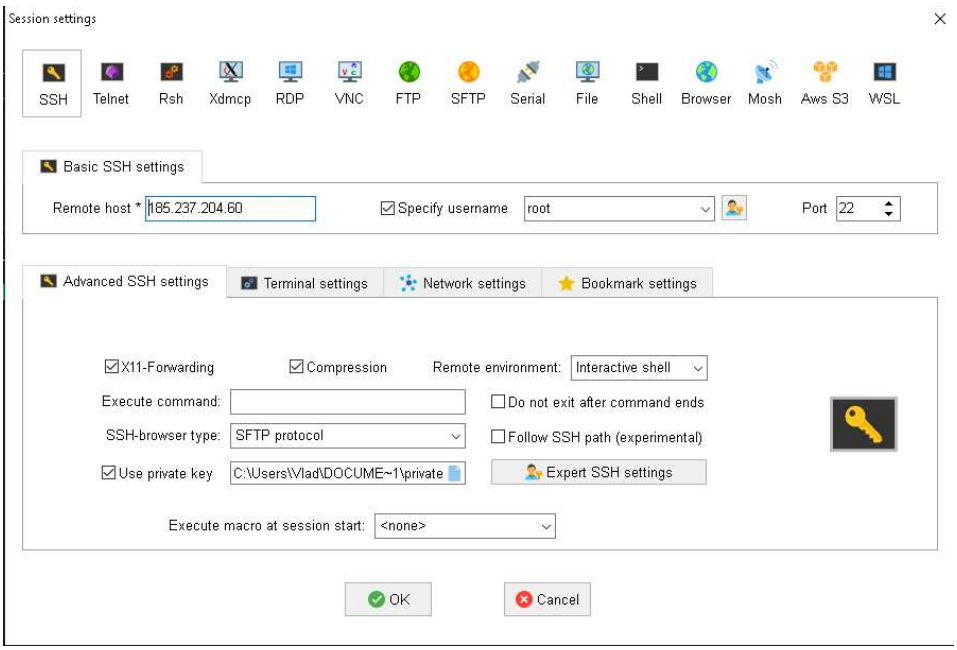


Рисунок 3.41 – Вікно додавання SSH ключу.

У розділі "Advanced SSH settings" встановлюється галочка "Use private key" та обирається збережений приватний ключ. Після збереження налаштувань сеансу здійснюється підключення до серверу. Якщо приватний ключ захищений паролем, його необхідно ввести для підтвердження підключення.

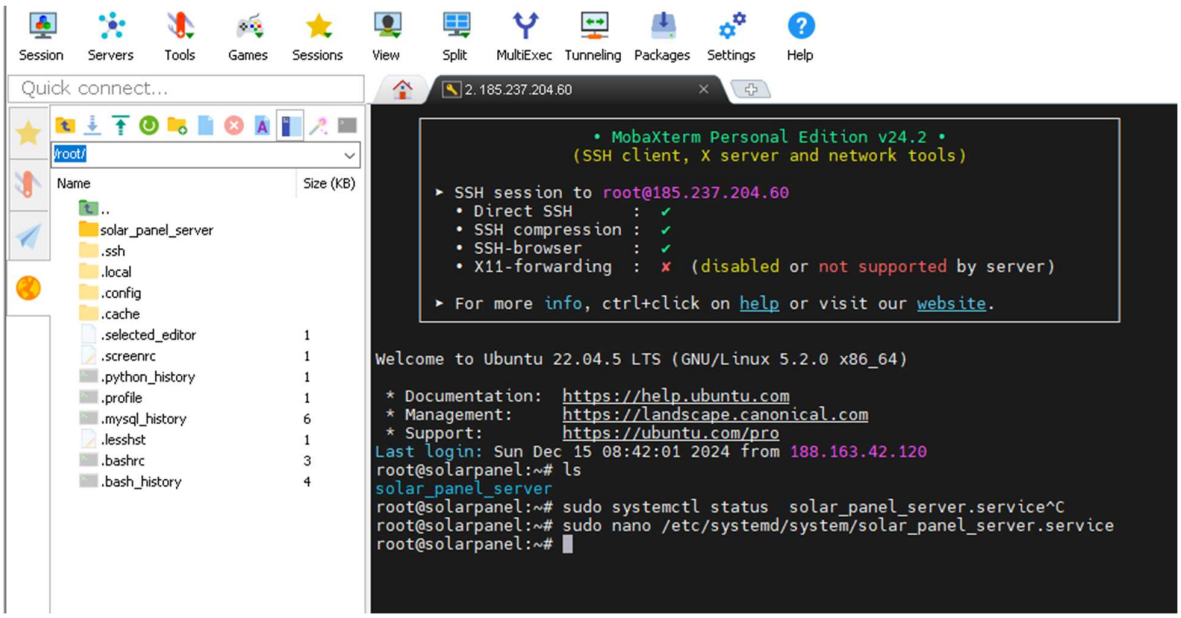


Рисунок 3.42 – Вікно вдалого підключення на сервер

Передача файлів та виконання команд на сервері здійснюється через вбудований SFTP клієнт у MobaXterm, який автоматично відкривається після встановлення SSH-з'єднання. Це дозволяє легко перетягувати файли між локальним комп'ютером та сервером. Термінал MobaXterm використовується для виконання команд на сервері HyperHost.

Програма серверу знаходиться у директорії /root/solar_panel_server/ solar_panel_server.py.

Для автоматичного запуску програми при старті системи було створено сервіс solar_panel_server.service.

```
[Unit]
Description=Solar Panel Server
After=network.target

[Service]
ExecStart=/usr/bin/python3 /root/solar_panel_server/solar_panel_server.py
Restart=always
User=root
WorkingDirectory=/root/solar_panel_server
Environment=PYTHONUNBUFFERED=1

[Install]
WantedBy=multi-user.target
```

Рисунок 3.43 - Код сервісу

Для внесення змін у програму solar_panel_server можна скористатися двома методами.

Редагування напряму на сервері:

Відкрийте файл програми для редагування, використовуючи текстовий редактор, наприклад nano.

nano /root/solar_panel_server/solar_panel_server.py

Внесіть необхідні зміни у код. Збережіть файл і закрийте редактор (у nano натисніть Ctrl + O, потім Enter і Ctrl + X).

Редагування за допомогою MobaTextEditor.

У MobaXterm відкрийте SFTP-панель, знайдіть файл /root/solar_panel_server/solar_panel_server.py, клацніть на нього правою кнопкою миші та виберіть "Open with MobaTextEditor".

MobaTextEditor дозволяє редагувати файл безпосередньо через графічний інтерфейс. Після внесення змін збережіть файл, і він автоматично оновиться на сервері.

Для перезапуску сервісу після внесення змін виконайте команду:

sudo systemctl restart solar_panel_server.service

Для перевірки статусу сервісу виконайте:

sudo systemctl status solar_panel_server.service

Цей сервіс гарантує автоматичний перезапуск програми у разі помилок або завершення роботи, що забезпечує стабільну та безперебійну роботу системи. Використання MobaTextEditor є зручним для користувачів, які віддають перевагу

графічному редагуванню, що робить процес адміністрування серверу ще простішим.

3.7 Результат розробки технічного забезпечення

У результаті розробки технічного забезпечення системи керування процесом було вибрано та інтегровано ряд програмних компонентів, які забезпечують стабільну роботу, збору та обробку даних, а також зручний інтерфейс для користувачів. Нижче наведена таблиця використаного програмного забезпечення з їх специфікаціями та інформацією про оплату.

Таблиця 3.2. - Використане програмне забезпечення

Назва програмного забезпечення	Версія	Призначення	Ліцензія	Вартість
Операційна система				
Ubuntu Server	22.04	Серверна ОС для VPS	Відкрита (GPL)	Безкоштовно
Мова програмування				
Python	3.10+	Розробка серверного додатку	Відкрита (Python Software Foundation License)	Безкоштовно
Фреймворк				
Flask	2.2+	Створення веб-додатку	Відкрита (BSD License)	Безкоштовно
СУБД				
MySQL	8.0+	Зберігання та обробка даних	Відкрита (GPL)	Безкоштовно
Середовище для розробки				
PlatformIO	5.0+	Розробка прошивок для мікроконтролерів	Відкрита (Apache License 2.0)	Безкоштовно

Продовження таблиці 3.2

Програмне забезпечення				
MobaXterm	20+	Підключення до серверу через SSH	Freeware (безкоштовна для особистого використання)	Безкоштовно
Бібліотеки та модулі				
WT32-ETHO1 бібліотеки	N/A	Комунікація з мікроконтролером	Відкрита	Безкоштовно
Chart.js	3.9+	Візуалізація даних на фронтенді	Відкрита (MIT License)	Безкоштовно
Хостинг				
HyperHost VPS Mini	N/A	Розміщення серверного додатку	Пропрієтарна	$5/місяць

Таблиця 3.3. - Специфікація використаних пристроїв

Назва пристрою	Модель/Версія	Основні характеристики	Призначення	Вартість грн
Мікроконтролер				
Arduino Uno Rev3	ATmega328P	14 цифрових входів/виходів, 6 аналогових входів, 32 КБ флеш-пам'яті	Збір даних з датчиків та передача на сервер	150
NodeMCU ESP8266 CH340	ESP8266	Вбудований Wi-Fi, 4 МБ флеш-пам'яті, USB-інтерфейс через CH340	Бездротова комунікація та відображення даних	200
WT32-ETHO1	ESP32 з Ethernet-контролером	Два Ethernet-порти, Wi-Fi, Bluetooth, 16 МБ флеш-пам'яті	Основний контролер системи, комунікація з PLC та сервером	250
Arduino Ethernet Shield W5100	W5100	Підключення до мережі Ethernet, SPI інтерфейс	Забезпечення проводного інтернет-з'єднання для Arduino	250

Продовження таблиці 3.3

Датчики				
AM2302	DHT22	Вимірювання температури та вологості, точність ±0.5°C, ±2-5% RH	Збір даних про навколишнє середовище	100
ACS712	30A GY-712	Вимірювання струму	Збір даних струму	70
Дільник напруги	3 резистори 10кОм	Зменшення наруги 24В до 5В	Збір даних напруги	10
Апаратні компоненти				
Вентилятор DC 12V 2pin	Cooling fan	Діаметр 12 см, 12V живлення, ефективне охолодження	Охолодження інвертора та інших компонентів	100
Power Inverter 12V-220V	300Вт	Перетворення 12V DC в 220V AC, пікова потужність 300Вт	Забезпечення живлення пристроїв зі змінним струмом	400
Шафа ударостійка з ABS-пластику	600x400x200 МП, IP65	Ударостійка, водонепроникність, монтажна панель	Зберігання та організація компонентів системи	2 000

Примітка: Вибір компонентів базувався на балансі між вартістю та функціональністю, забезпечуючи економічність проекту при збереженні високої якості та надійності системи контролю сонячної панелі.

Опис виконаних робіт.

Розробка технічного забезпечення системи керування процесом сонячної панелі включала декілька ключових етапів, спрямованих на забезпечення функціональності та економічності проекту. Першим кроком було проведення аналізу різних варіантів апаратних та програмних компонентів, після чого було обрано оптимальні рішення, що найкраще відповідали вимогам проекту. Відповідно до цього, було здійснено закупівлю необхідних пристроїв, модулів та аксесуарів з урахуванням бюджету проекту, що забезпечило надійну базу для подальшої інеграції та розвитку системи.

На етапі інтеграції мікроконтролерів та модулів було підключено Arduino Uno Rev3 з Ethernet Shield W5100 для збору даних з датчиків та їх передачі на сервер, що дозволило забезпечити надійну комунікацію між сенсорами та центральною системою. Додатково було інтегровано NodeMCU ESP8266 CH340 для бездротової комунікації та відображення даних, що покращило зручність моніторингу системи в реальному часі. Основним контролером системи стало WT32-ETH01, яке було встановлено для управління сонячною панеллю та забезпечення комунікації з сервером, що дозволяє ефективно координувати роботу всіх компонентів системи.

Налаштування мережевих з'єднань. Забезпечено підключення всіх мікроконтролерів до мережі Ethernet та Wi-Fi.

Налаштовано сервер на базі Ubuntu з встановленими Python Flask додатком та MySQL базою даних для обробки та зберігання даних.

Розробка програмного забезпечення:

− Написано прошивки для мікроконтролерів з використанням PlatformIO, забезпечуючи збір даних з датчиків, їх обробку та передачу на сервер.

− Створено серверний додаток на Python Flask, який обробляє HTTP запити, зберігає дані в MySQL базу та надає API для веб-інтерфейсу.

− Розроблено веб-інтерфейс з використанням HTML, CSS та JavaScript, забезпечуючи зручний моніторинг та управління системою.

Проведено тестування всіх компонентів системи для забезпечення їх сумісності та стабільної роботи, виявлено та виправлено помилки в прошивках мікроконтролерів та серверному додатку та перевірено коректність збору та передачі даних, а також їх відображення на веб-інтерфейсі.

Монтаж та встановлення обладнання:

Розміщено всі апаратні компоненти в шафі ударостійкої з ABS-пластику, забезпечуючи захист від зовнішніх впливів.

Рисунок 3.43 – Загальний вигляд стенду сонячної панелі

Підключено датчики температури, вологості, напруги та струму до мікроконтролерів, забезпечуючи повний збір необхідних даних.

Рисунок 3.44 – Внутрішній вигляд шафового ящика

Успішно інтегровано всі апаратні та програмні компоненти, забезпечивши безперебійну роботу системи керування сонячною панеллю.

Вибір доступних компонентів дозволив знизити загальні витрати на розробку, не жертвуючи якістю та функціональністю.

Розроблено інтуїтивно зрозумілий веб-інтерфейс для моніторингу та управління системою, що сприяє ефективному використанню ресурсів сонячної панелі.

Проведене тестування підтвердило стабільну роботу системи в різних умовах, забезпечуючи надійний збір та передачу даних.

Результатом розробки технічного забезпечення стало створення комплексної системи керування процесом з використанням ефективних програмних та апаратних рішень. Використання доступних та надійних компонентів дозволило забезпечити стабільну роботу системи, ефективний збір та обробку даних, а також зручний інтерфейс для користувачів, що сприяє підвищенню продуктивності та надійності роботи сонячних панелей.

Висновки за розділом

Розробка технічного забезпечення системи керування процесом сонячної панелі була виконана у кілька ключових етапів, що охоплюють весь цикл створення системи — від вибору компонентів до її тестування та впровадження. Основні досягнення розробки включають:

Усі апаратні та програмні компоненти були успішно об’єднані, забезпечивши їхню стабільну та безперебійну роботу. Це свідчить про добре продуману архітектуру та ефективну реалізацію проекту.

Використання доступних компонентів дозволило суттєво оптимізувати бюджет проекту без шкоди для функціональності чи надійності. Це забезпечило конкурентоспроможність розробленої системи.

Розроблено зручний інтерфейс для моніторингу та управління системою. Він дозволяє ефективно відстежувати дані та здійснювати налаштування, підвищуючи користувацький досвід.

Проведене тестування довело коректність збору, обробки та передачі даних, а також стійкість системи до зовнішніх впливів. Це робить систему придатною для використання в реальних умовах.

Проект реалізований із врахуванням всіх аспектів — від вибору апаратних засобів до навчання кінцевих користувачів. Підготовлена документація сприяє легкому впровадженню системи в експлуатацію.

Результатом проекту є надійна, функціональна та економічно ефективна система керування сонячною панеллю, яка відповідає сучасним вимогам до подібних рішень. Розробка сприятиме підвищенню продуктивності та надійності сонячних панелей, забезпечуючи максимальну ефективність використання ресурсів.

ВИСНОВКИ

У сучасних умовах, коли забезпечення сталої та екологічно безпечної енергетики стає пріоритетом, впровадження фотоелектричних систем (ФЕС) набуває особливої ваги. Проведене дослідження зосереджувалося на розробці та практичній реалізації системи веб-діагностики та керування електричними параметрами фотоелектричних панелей, що суттєво підвищує ефективність та надійність роботи ФЕС.

Практична реалізація проекту є ключовим елементом дослідження. Розроблена система включає інтеграцію різних мікроконтролерів (WT32-ETH01, ESP8266, Arduino UNO Rev3) для збору та передачі даних, а також створення надійної веб-платформи для моніторингу та керування сонячними панелями. Завдяки використанню доступних компонентів та відкритих технологій, вдалося знизити загальні витрати на розробку системи, забезпечуючи при цьому високу функціональність та гнучкість.

Практична реалізація продемонструвала успішне досягнення поставлених цілей. Завдяки розробленій системі було досягнуто підвищення виробленої енергії на 20% порівняно зі стаціонарними панелями. Це свідчить про ефективність впроваджених алгоритмів керування та оптимізації роботи панелей. Крім того, система забезпечила стабільну роботу навіть у складних умовах, що підтверджується швидким часом стабілізації панелі після зміни положення сонця.

Лабораторний стенд, створений у рамках цього проекту, став важливим інструментом для демонстрації сучасних технологій управління та моніторингу ФЕС. Він не лише забезпечує можливість реального часу відстеження роботи панелей, але й слугує платформою для подальших досліджень та освітніх цілей. Це сприяє підготовці кваліфікованих фахівців у галузі відновлюваної енергетики, що є необхідним для подальшого розвитку зеленої енергетики в Україні та світі.

Міжнародний досвід впровадження подібних систем підтверджує високу доцільність використання сучасних інформаційних технологій для моніторингу та управління ФЕС. В Україні, незважаючи на певні виклики, перспективи розвитку сонячної енергетики виглядають оптимістично завдяки підтримці держави та міжнародних партнерів, таких як Європейський Союз, що надає необхідну технічну та фінансову підтримку.

Впровадження системи веб-діагностики та керування сприяє не лише підвищенню енергетичної незалежності, але й зниженню екологічного впливу від використання традиційних викопних джерел енергії. Зниження витрат на енергію, зменшення залежності від імпортованих ресурсів та зниження викидів парникових газів є вагомими перевагами, які приносить впровадження розробленої системи.

Таким чином, проведене дослідження не лише розширює функціональні можливості наявних фотоелектричних систем, але й робить вагомий внесок у розвиток зелених технологій. Впровадження розробленої системи веб-діагностики та керування електричними параметрами панелей сприяє підвищенню ефективності, надійності та економічної доцільності експлуатації ФЕС. Це створює основу для подальшого розвитку та впровадження відновлюваних джерел енергії, сприяючи досягненню глобальних цілей у сфері енергетичної сталості та екологічної безпеки.

Перспективи подальших досліджень включають розширення функціональності веб-платформи, інтеграцію додаткових сенсорів для більш точного моніторингу та впровадження передових алгоритмів аналізу даних. Це дозволить ще більше підвищити продуктивність ФЕС та забезпечити їхню адаптивність до змінних умов навколишнього середовища. Крім того, розвиток подібних систем може стати важливим кроком на шляху до глобального енергетичного переходу та зменшення залежності від не відновлюваних джерел енергії.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Маринич І. А., Тронь В. В. Методичні рекомендації до виконання кваліфікаційної роботи магістра для студентів спеціальності 151 “Автоматизація та комп’ютерно-інтегровані технології”. Кривий Ріг : Видавничий центр КНУ, 2022. 50с.
2. ДСТУ 3008:2015. Звіти у сфері науки і техніки. Структура і правила оформлення. Київ, ДП «УкрННЦ», 2015. 26с. (Інформація та документація).
3. ДСТУ 8302:2015. Бібліографічне посилання. Загальні вимоги та правила складання Київ, ДП «УкрННЦ», 2016. 16 с. (Інформація та документація).
4. ДСТУ 3582:2013. Бібліографічний опис. Скорочення слів і словосполучень в українській мові. Загальні вимоги та правила. Київ, ДП «УкрННЦ», 2013. 23 с. (Інформація та документація)
5. ДСТУ 3651.0-97 Метрологія. Одиниці фізичних величин. Основні одиниці фізичних величин Міжнародної системи одиниць. Основні положення, назви та позначення Київ, Держстандарт України, 1998. 27 с. (Інформація та документація).
6. Renewable Energy Capacity Statistics 2023 [Електронний ресурс] https://unreeea.org/wp-content/uploads/2023/04/IRENA_RE_Capacity_Statistics_2023.pdf
7. Що втратила та придбала сонячна енергетика за час війни в Україні [Електронний ресурс] https://thepage.ua/ua/economy/sonyachna-energetika-v-ukrayini-u-2023-roci
8. ЄС НАДАСТЬ 5,7 ТИС. СОНЯЧНИХ ПАНЕЛЕЙ ДЛЯ УКРАЇНИ [Електронний ресурс] https://eu-ua.kmu.gov.ua/news/yes-nadast-5-7-tys-sonyachnyh-panelej-dlya-ukrayiny/
9. Як IoT у системах сонячної енергії може допомогти енергетичним компаніям? [Електронний ресурс] https://www.dusuniot.com/uk/blog/iot-in-solar-energy/
10. Все що потрібно знати про Node.js: особливості, характеристики, область застосування [Електронний ресурс] https://wezom.com.ua/ua/blog/vse-chto-nuzhno-znat-o-nodejs
11. Flask [Електронний ресурс] https://flask.palletsprojects.com/en/3.0.x/

12. Що таке Django? [Електронний ресурс] https://tutorial.djangogirls.org/uk/django/

13. Що таке React JS і для чого він потрібен? [Електронний ресурс] https://dan-it.com.ua/uk/blog/chto-takoe-react-js-i-dlja-chego-on-nuzhen/

14. MySQL [Електронний ресурс] https://uk.wikipedia.org/wiki/MySQL

15. What is PostgreSQL? [Електронний ресурс] https://www.postgresql.org/about/

16. SolarEdge Home: More Power. More Places. . [Електронний ресурс] https://www.solaredge.com/us/home/solaredge-home

17. mySunPower [Електронний ресурс] https://play.google.com/store/apps/details?id=com.mysunpower

18. Solar panel performance modeling and monitoring [Електронний ресурс] https://patents.google.com/patent/US11146212B1/en

19. Multiple time interval solar controller [Електронний ресурс] https://patents.google.com/patent/CN201093312Y/en

20. Dual-Axis Solar Tracker [Електронний ресурс] US20100024861A1 - https://patents.google.com/patent/US20100024861A1/en

21. Optimizing Solar Panel Efficiency and Placement Using AI-Driven Project Management Tools Read more on [Електронний ресурс] https://tech-stack.com/blog/solar-panel-efficiency/

22. RUTX50 5G EPA ПІДКЛЮЧЕННЯ [Електронний ресурс] https://teltonika-networks.com/ua

23. Контроллер Arduino Uno Rev3 (ATmega16U2) [Електронний ресурс] https://arduino.ua/prod676-arduino-uno-rev3

24. Шилд W5100 ethernet shield [Електронний ресурс] https://arduino.ua/prod391-w5100-ethernet-shield

25. Wi-Fi модуль NodeMCU V3 ESP8266 ESP-12 (CH340) [Електронний ресурс] https://ardushop.in.ua/arduino/wi-fi-module-nodemcu-v3-esp8266-esp-12

26. WT32-ETH01, маленькая плата с ESP32 + Ethernet [Електронний ресурс] https://mysku.club/blog/aliexpress/84562.html

27. AM2301, Високоточний датчик вологості та температури [Електронний ресурс] https://radiovolt.in.ua/ua/p1168909582-am2301-vysokotochnyj-datchik.html?source=merchant_center&gad_source=1&gclid=Cj0KCQiAoKeuBhCoARIsAB4WxtfTuCAPm2AwkLI7BQMfW-aVMhg15cx4rwkZWqzYXaggD-tPZu7JemAaAibZEALw_wcB

28. Датчик струму ACS712 30A GY-712 [Електронний ресурс] https://arduino.ua/prod618-datchik-toka-acs712-30a-gy-

712?srsltid=AfmBOopkwESjox9VeVWaJUTeSHLhWxUIlKXgzAfThoM5h2weuM4_B23M

29. Вентилятор DC 12V 2pin Cooling fan [Електронний ресурс] https://www.aliexpress.com/item/1005006394890913.html

30. 300W DC-AC Converter Booster module [Електронний ресурс] https://www.aliexpress.com/item/1005005369578463.html

31. Шафа ударостійка з ABS-пластику 600x400x200 МП, IP65 [Електронний ресурс] https://vse-e.com/ua/shkaf-udaroprochnyi-iz-abs-plastika-600h400h200-mp-ip65

32. What is a REST API? [Електронний ресурс] https://www.ibm.com/topics/rest-apis

33. Український хостинг провайдер HyperHost [Електронний ресурс] https://hyperhost.ua/uk

34. UART [Електронний ресурс] https://uk.wikipedia.org/wiki/UART

35. HypeHost Login [Електронний ресурс] https://hyperhost.ua/client/index.php?rp=/login

36. MobaXterm Enhanced terminal for Windows with X11 server, tabbed SSH client, network tools and much more [Електронний ресурс] https://mobaxterm.mobatek.net/

ДОДАТОК А. Основний код контролерів

```
// main.cpp

#define DEBUG_ETHERNET_WEBSERVER_PORT       Serial

#include <ETH.h>
#include <ESPAsyncWebServer.h>
#include "CommunicationManager.h"
#include "SolarWebServer.h"


const char* ssid = "SolarPanelLocal";

IPAddress local_IP(192, 168, 1, 2);
IPAddress wifi_IP(192, 168, 1, 5);
IPAddress gateway(192, 168, 1, 1);
IPAddress subnet(255, 255, 255, 0);
IPAddress dns(8, 8, 8, 8);

const IPAddress plcIp(192, 168, 1, 3);
const uint16_t plcPort = 2001;

AsyncWebServer webServer(80);

CommunicationManager commManager(plcIp, plcPort);

SolarWebServer server(webServer,commManager);

// Функція зворотного виклику для подій Ethernet
void WiFiEvent(WiFiEvent_t event) {
   switch(event) {
      case SYSTEM_EVENT_ETH_START:
         DPrint("ETH Started");
         ETH.setHostname("WT32-ETHE01");
         break;
      case SYSTEM_EVENT_ETH_CONNECTED:
         DPrint("ETH Connected");
         break;
      case SYSTEM_EVENT_ETH_GOT_IP:
         DPrint("ETH MAC: "); DPrint(ETH.macAddress());
         DPrint("ETH IP: "); DPrint(ETH.localIP());
         break;
      case SYSTEM_EVENT_ETH_DISCONNECTED:
```

```
            DPrint("ETH Disconnected");
            break;
        case SYSTEM_EVENT_ETH_STOP:
            DPrint("ETH Stopped");
            break;
        default:
            break;
    }
}


void setup() {

    Serial.begin(115200);
    delay(1000);

    if (!LittleFS.begin()) {
        DPrint("An Error has occurred while mounting LittleFS");
        while (true) {
            delay(1000);
        }
    } else {
        DPrint("LittleFS mounted successfully");
    }
    WiFi.softAPConfig(wifi_IP, wifi_IP, subnet);
    WiFi.softAP(ssid);

    WiFi.onEvent(WiFiEvent);
    if (!ETH.begin()) {
        DPrint("ETH.begin() failed");
        while (true) {
            delay(1);
        }
    }
    if (!ETH.config(local_IP, gateway, subnet, dns)) {
        DPrint("ETH Config Failed");
    }
    DPrint(WiFi.softAPIP());

    server.begin();
    DPrint("Starting Ethernet ...");

}
```

```
void loop(){
   commManager.Update();

   delay(10);
}
// common.h
#pragma once
#include <WiFiClient.h>
#include <Arduino.h>
#include <vector>
#include <cstring>
#include <LittleFS.h>
#define _DEBUG
enum uart_ids {
   WT32_ETH01 = 10,
   ESP8266,
   Arduino
};

enum uart_data_types {
   solar_panel_feed_back_Type = 100,
   WIFI_Credentials_Type, //1
   ESP8266_Sensor_type, //2
   Arduino_Sensors_Type, //3
   Simence_TCP_Status, //4
   ESP8266_STATUS_OK, //5
   ARDUINO_STATUS_OK, //6
   ESP8266_Check_Connection, //7
   ARDUINO_Check_Connection, //8
   ARDUINO_ETH_Connection // 9
};

const uart_ids current_device = uart_ids::WT32_ETH01;

struct SolarPanelControlCommands {

   bool Start;
   bool Stop;
   bool AutoStartOn;
   bool AutoStartOff;
   bool RemoteControlOn;
   bool RemoteControlOff;
   bool Left;
   bool Right;
```

```
    bool Up;
    bool Down;
    bool ResetPosition;
    bool SetSunriseTime;
    bool SetSunsetTime;
    bool SetDateTime;
    bool SetVelocity;
    bool ToZeroPosition;

    uint8_t sunrise_hours;
    uint8_t sunrise_minutes;
    uint8_t sunset_hours;
    uint8_t sunset_minutes;
    uint8_t weekday;
    uint8_t day;
    uint8_t month;
    uint16_t year;
    uint8_t hours;
    uint8_t minutes;
    uint8_t seconds;
    uint16_t velocity;
};
constexpr size_t SolarPanelControlCommands_size = 30;

struct SolarPanelStatus {
    bool Started;
    bool AutoStart;
    bool ManualMode;
    bool H_MoveDone;
    bool H_Moving;
    bool V_MoveDone;
    bool V_Moving;
    bool Sunlight_Founded;
    int16_t H_Position;
    int16_t V_Position;
    int16_t Velocity;
    int16_t SunSensor;
};

constexpr size_t SolarPanelStatus_size = 16;

struct WiFiCredentials {
    String ssid;
```

```
    String password;
};

struct SensorData {
    uint16_t temperature;
    uint16_t humidity;
    uint16_t current;
    uint16_t voltage;
};

struct ToServer{
    SensorData data;
};


constexpr int PACKET_SIZE = 64;
constexpr int BAUD_RATE = 9600;

#ifdef _DEBUG
    #define DPrint(data) Serial.println(data)
#else
    #define DPrint(data)
#endif

struct Packet {
    uint8_t data[PACKET_SIZE];
    size_t length;
};

// Serialization.cpp
#include "Serialization.h"

void _serializeSolarPanelControl(uint8_t* buffer, const
SolarPanelControlCommands& data, size_t index ) {

    buffer[index++] = data.Start;
    buffer[index++] = data.Stop;
    buffer[index++] = data.AutoStartOn;
    buffer[index++] = data.AutoStartOff;
    buffer[index++] = data.RemoteControlOn;
    buffer[index++] = data.RemoteControlOff;
    buffer[index++] = data.Left;
    buffer[index++] = data.Right;
    buffer[index++] = data.Up;
```

```
        buffer[index++] = data.Down;
        buffer[index++] = data.ResetPosition;
        buffer[index++] = data.SetSunriseTime;
        buffer[index++] = data.SetSunsetTime;
        buffer[index++] = data.SetDateTime;
        buffer[index++] = data.SetVelocity;
        buffer[index++] = data.ToZeroPosition;
        // Serialize other fields
        buffer[index++] = data.sunrise_hours;
        buffer[index++] = data.sunrise_minutes;
        buffer[index++] = data.sunset_hours;
        buffer[index++] = data.sunset_minutes;
        buffer[index++] = data.weekday;
        buffer[index++] = data.day;
        buffer[index++] = data.month;

        // Serialize 16-bit integers
        buffer[index++] = data.year & 0xFF;
        buffer[index++] = (data.year >> 8) & 0xFF;

        buffer[index++] = data.hours;
        buffer[index++] = data.minutes;
        buffer[index++] = data.seconds;

        buffer[index++] = data.velocity & 0xFF;
        buffer[index++] = (data.velocity >> 8) & 0xFF;

    }



    void _deserializeSolarPanelControl(const uint8_t*
buffer,SolarPanelControlCommands& data,  size_t index) {
        // Deserialize boolean values
        data.Start = buffer[index++];
        data.Stop = buffer[index++];
        data.AutoStartOn = buffer[index++];
        data.AutoStartOff = buffer[index++];
        data.RemoteControlOn = buffer[index++];
        data.RemoteControlOff = buffer[index++];
        data.Left = buffer[index++];
        data.Right = buffer[index++];
        data.Up = buffer[index++];
        data.Down = buffer[index++];
```

```
    data.ResetPosition = buffer[index++];
    data.SetSunriseTime = buffer[index++];
    data.SetSunsetTime = buffer[index++];
    data.SetDateTime = buffer[index++];
    data.SetVelocity = buffer[index++];
    data.ToZeroPosition = buffer[index++];
    // Deserialize other fields
    data.sunrise_hours = buffer[index++];
    data.sunrise_minutes = buffer[index++];
    data.sunset_hours = buffer[index++];
    data.sunset_minutes = buffer[index++];
    data.weekday = buffer[index++];
    data.day = buffer[index++];
    data.month = buffer[index++];

    // Deserialize 16-bit integers
    data.year = buffer[index++] | (buffer[index++] << 8);

    data.hours = buffer[index++];
    data.minutes = buffer[index++];
    data.seconds = buffer[index++];

    data.velocity = buffer[index++] | (buffer[index++] << 8);
}



void _deserializeSolarPanelStatus(const uint8_t* buffer, SolarPanelStatus& data,
size_t index = 0) {

    data.Started = buffer[index++];
    data.AutoStart = buffer[index++];
    data.ManualMode = buffer[index++];
    data.H_MoveDone = buffer[index++];
    data.H_Moving = buffer[index++];
    data.V_MoveDone = buffer[index++];
    data.V_Moving = buffer[index++];
    data.Sunlight_Founded = buffer[index++];

    data.H_Position = buffer[index++] | (buffer[index++] << 8);
    data.V_Position = buffer[index++] | (buffer[index++] << 8);
    data.Velocity = buffer[index++] | (buffer[index++] << 8);
    data.SunSensor = buffer[index++] | (buffer[index++] << 8);
}
```

```
        void _serializeSolarPanelStatus(uint8_t* buffer, const SolarPanelStatus& data,
size_t index = 0) {

            buffer[index++] = data.Started;
            buffer[index++] = data.AutoStart;
            buffer[index++] = data.ManualMode;
            buffer[index++] = data.H_MoveDone;
            buffer[index++] = data.H_Moving;
            buffer[index++] = data.V_MoveDone;
            buffer[index++] = data.V_Moving;
            buffer[index++] = data.Sunlight_Founded;

            buffer[index++] = data.H_Position & 0xFF;
            buffer[index++] = (data.H_Position >> 8) & 0xFF;

            buffer[index++] = data.V_Position & 0xFF;
            buffer[index++] = (data.V_Position >> 8) & 0xFF;

            buffer[index++] = data.Velocity & 0xFF;
            buffer[index++] = (data.Velocity >> 8) & 0xFF;

            buffer[index++] = data.SunSensor & 0xFF;
            buffer[index++] = (data.SunSensor >> 8) & 0xFF;

        }

        void serializeSolarPanelControl(uint8_t* buffer, const
SolarPanelControlCommands& data , bool toWT32) {

            size_t index = 0;
            if (toWT32)
            {
               memset(buffer, 0, PACKET_SIZE);
               buffer[0] = uart_ids::WT32_ETH01;
               buffer[1] = uart_data_types::solar_panel_feed_back_Type;
               buffer[2] = sizeof(SolarPanelControlCommands);
               index = 3;

            }else{
               memset(buffer, 0, SolarPanelControlCommands_size);
            }

            _serializeSolarPanelControl(buffer,data, index);
```

```
    }

    void serializeWifiCredentials(uint8_t* buffer, const WiFiCredentials&
credentials)
    {
      memset(buffer, 0, PACKET_SIZE);

      size_t index = 0;

      strncpy(reinterpret_cast<char*>(&buffer[index]), credentials.ssid.c_str(),
credentials.ssid.length());
      index += credentials.ssid.length();
      buffer[index++] = '\0';

      strncpy(reinterpret_cast<char*>(&buffer[index]), credentials.password.c_str(),
credentials.password.length());
      index += credentials.password.length();
      buffer[index] = '\0';
    }

    void deserializeWifiCredentials(const uint8_t* buffer, WiFiCredentials&
credentials)
    {
      size_t index = 0;

      String ssid = String(reinterpret_cast<const char*>(&buffer[index]));
      credentials.ssid = ssid;

      index += ssid.length() + 1;

      String password = String(reinterpret_cast<const char*>(&buffer[index]));
      credentials.password = password;

      DPrint("WiFiCredentials deserialized:");
      DPrint("SSID: " + credentials.ssid);
      DPrint("Password: " + credentials.password);
    }

    void serializeSolarPanelStatus(uart_ids id, uint8_t* buffer, const
SolarPanelStatus& data) {
      memset(buffer, 0, PACKET_SIZE);
      buffer[0] = id;
      buffer[1] = uart_data_types::solar_panel_feed_back_Type;
```

```
    buffer[2] = sizeof(SolarPanelStatus);

    int16_t index = 3;
    _serializeSolarPanelStatus(buffer,data, index);
}


void deserializeSolarPanelStatus(const uint8_t* buffer, SolarPanelStatus& data) {
    size_t index = 3;

    _deserializeSolarPanelStatus(buffer,data,0);

}
void deserializeBufferToSolarPanelControlWord(const uint8_t* buffer,
SolarPanelControlCommands& data) {
    int16_t index = 3;

    _deserializeSolarPanelControl(buffer,data,index);

}

void serializeSensorData(uint8_t* buffer, const SensorData& data)
{
    int index = 3;
    buffer[index++] = (data.temperature >> 8) & 0xFF;
    buffer[index++] = data.temperature & 0xFF;

    buffer[index++] = (data.humidity >> 8) & 0xFF;
    buffer[index++] = data.humidity & 0xFF;

    buffer[index++] = (data.current >> 8) & 0xFF;
    buffer[index++] = data.current & 0xFF;

    buffer[index++] = (data.voltage >> 8) & 0xFF;
    buffer[index++] = data.voltage & 0xFF;
}

void deserializeSensorData(const uint8_t* buffer, SensorData& data)
{
    int index = 3;
    data.temperature = (buffer[index++] << 8) | buffer[index++];
    data.humidity = (buffer[index++] << 8) | buffer[index++];
    data.current = (buffer[index++] << 8) | buffer[index++];
    data.voltage = (buffer[index++] << 8) | buffer[index++];
```

```
}
// UartController.cpp
#include "UartController.h"
#include "CommunicationManager.h"
UartController::UartController(CommunicationManager* manager)
: _manager(manager)
{
   serial.begin(BAUD_RATE, SERIAL_8N1, RXD2, TXD2);
   onWaitResponse = false;
}

UartController::~UartController()
{
   serial.end();
}

void UartController::Update()
{
   if (onWaitResponse)
   {
      unsigned long currentTime = millis();

      if (currentTime - lastRequestTime >= responseTimeout)
      {
         lastStatusConnected = false;
         DPrint("Response timeout. Resending packet...");
         onWaitResponse = false;
      }
   }

   if (!onWaitResponse && Serial.availableForWrite() &&
!packetQueue.empty())
   {
      Packet packet = packetQueue.front();
      sendPacket(packet.data, packet.length);
      packetQueue.erase(packetQueue.begin());
      lastRequestTime = millis();
   }

   receivePacket();
}
void UartController::sendCheckConnectionStatusESP8266()
{
   uint8_t serializedData[PACKET_SIZE];
```

```
    memset(serializedData, 0, PACKET_SIZE);
    serializedData[0] = uart_ids::ESP8266;
    serializedData[1] = uart_data_types::ESP8266_Sensor_type;
    enqueuePacket(serializedData, PACKET_SIZE);
}
void UartController::sendCheckConnectionStatusArduino()
{
    DPrint("sendCheckConnectionStatusArduino");
    uint8_t serializedData[PACKET_SIZE];
    memset(serializedData, 0, PACKET_SIZE);
    serializedData[0] = uart_ids::Arduino;
    serializedData[1] = uart_data_types::ARDUINO_STATUS_OK;
    enqueuePacket(serializedData, PACKET_SIZE);
}
void UartController::sendPanelCommands(const SolarPanelControlCommands
&data)
{
    uint8_t serializedData[PACKET_SIZE];
    memset(serializedData, 0, PACKET_SIZE);
    serializeSolarPanelControl( serializedData, data);
    enqueuePacket(serializedData, PACKET_SIZE);
}

void UartController::sendPanelStatus(const SolarPanelStatus& data)
{
    uint8_t serializedData[PACKET_SIZE];
    memset(serializedData, 0, PACKET_SIZE);
    serializeSolarPanelStatus(uart_ids::ESP8266,serializedData, data);
    enqueuePacket(serializedData, PACKET_SIZE);
}

void UartController::sendWifiCredentials(const WiFiCredentials& credentials)
{
    uint8_t serializedData[PACKET_SIZE];
    memset(serializedData, 0, PACKET_SIZE);
    serializeWifiCredentials(serializedData, credentials);
    enqueuePacket(serializedData, PACKET_SIZE);
}


void UartController::sendPLCConnected(bool isConnected)
{
    uint8_t serializedData[PACKET_SIZE];
    memset(serializedData, 0, PACKET_SIZE);
```

```
    serializedData[0] = uart_ids::ESP8266;
    serializedData[1] = uart_data_types::Simence_TCP_Status;
    serializedData[3] = isConnected;

    enqueuePacket(serializedData, PACKET_SIZE);

}
void UartController::sendSensorData(const SensorData& data)
{
    uint8_t serializedData[PACKET_SIZE];
    memset(serializedData, 0, PACKET_SIZE);
    serializedData[0] = uart_ids::ESP8266;
    serializedData[1] = uart_data_types::ESP8266_Sensor_type;

    serializeSensorData(serializedData, data);

    enqueuePacket(serializedData, PACKET_SIZE);
}


void UartController::sendPacket(uint8_t *packet, size_t length)
{
    if (length > PACKET_SIZE)
    {
        DPrint("Packet length exceeds PACKET_SIZE");
        return;
    }
    packet[length - 1] = calculateChecksum(packet, length);
    DPrint("sendPacket");
    printData(packet, length);

    serial.write(packet, length);

    while (serial.available())
        serial.read();

}

void UartController::receivePacket()
{
    if (serial.available() >= PACKET_SIZE)
    {
        uint8_t data[PACKET_SIZE];
        memset(data, 0, PACKET_SIZE);
```

```
        serial.readBytes(data, PACKET_SIZE);
        if (data[0] != current_device)
        {
            while (serial.available())
                serial.read();
        }
        printData(data, PACKET_SIZE);

        uint8_t receivedChecksum = data[PACKET_SIZE - 1];
        uint8_t calculatedChecksum = calculateChecksum(data, PACKET_SIZE);
        if (receivedChecksum != calculatedChecksum)
        {
            DPrint("Checksum mismatch. Packet discarded.");
            return;
        }
        if(data[0] == current_device && data[1] == ESP8266_STATUS_OK)
        {
            packetQueue.erase(packetQueue.begin());
            lastStatusConnected = true;
            return;
        }
        this->_manager->dataHandle(data);

    }


}


void UartController::enqueuePacket(uint8_t *packetData, size_t length)
{

    if (length > PACKET_SIZE)
    {
        DPrint("enqueuePacket: Packet length exceeds PACKET_SIZE");
        return;
    }
    Packet packet;
    memcpy(packet.data, packetData, length);
    packet.length = length;

    packetQueue.emplace_back(packet);
}
```

```
uint8_t UartController::calculateChecksum(const uint8_t *packet, size_t length)
{
   uint16_t sum = 0;
   for (size_t i = 0; i < length - 1; i++)
   {
      sum += packet[i];
   }
   return static_cast<uint8_t>(sum & 0xFF);
}

void UartController::printData(const uint8_t *data, size_t length)
{
   for (size_t i = 0; i < length; i++)
   {
      DPrint(data[i]);
   }
   DPrint("");
}
// SolarWebServer.cpp
#include "SolarWebServer.h"

// Конструктор класу
SolarWebServer::SolarWebServer(AsyncWebServer &server,
CommunicationManager &manager) : _server(server), _manager(manager) {}

// Ініціалізація маршрутів та обробників
void SolarWebServer::begin()
{
   // Налаштування маршрутів
   _server.on(API_CONTROL_PANEL, HTTP_POST,
[](AsyncWebServerRequest *request) {}, NULL, [this](AsyncWebServerRequest
*request, uint8_t *data, size_t len, size_t index, size_t total)
         { this->handleControlPanel(request, data, len, index, total); });
   _server.on(API_PANEL_STATUS, HTTP_GET,
std::bind(&SolarWebServer::handleGetPanelData, this, std::placeholders::_1));
   _server.on(API_NETWORK, HTTP_GET,
std::bind(&SolarWebServer::handleGetNetworkSettings, this, std::placeholders::_1));
   _server.on(API_NETWORK_SAVE, HTTP_POST,
[](AsyncWebServerRequest *request) {}, NULL, [this](AsyncWebServerRequest
*request, uint8_t *data, size_t len, size_t index, size_t total)
         { this->handleSaveNetworkSettings(request, data, len, index, total); });
   _server.on(API_PLC_CONNECTION_STATUS, HTTP_GET,
std::bind(&SolarWebServer::handleCheckConnection, this, std::placeholders::_1));
```

```
        _server.on(API_SENSOR_DATA, HTTP_GET,
std::bind(&SolarWebServer::handleSensorData, this, std::placeholders::_1));

        _server.on("/", HTTP_GET, [](AsyncWebServerRequest *request)
                { request->send(LittleFS, INDEX_HTML, "text/html"); });
        _server.on(STYLES_CSS, HTTP_GET, [](AsyncWebServerRequest *request)
                { request->send(LittleFS, STYLES_CSS, "text/css"); });
        _server.on(SCRIPT_JS, HTTP_GET, [](AsyncWebServerRequest *request)
                { request->send(LittleFS, SCRIPT_JS, "application/javascript"); });

        _server.onNotFound([](AsyncWebServerRequest *request)
                        {
                           JsonDocument doc;
                           doc["error"] = "Not Found";
                           String json;
                           serializeJson(doc, json);
                           request->send(404, "application/json", json); });

        _server.begin();
    }

    void SolarWebServer::handleControlPanel(AsyncWebServerRequest *request,
uint8_t *data, size_t len, size_t index, size_t total)
    {
        String body = urlDecode(data, len);
        JsonDocument doc;
        DeserializationError error = deserializeJson(doc, body);

        if (error)
        {
            DPrint("JSON parsing failed");
            request->send(400, "application/json", "{\"error\":\"Invalid JSON\"}");
            return;
        }

        SolarPanelControlCommands commands;
        commands.Start = doc["start"];
        commands.Stop = doc["stop"];
        commands.AutoStartOn = doc["autostartOn"];
        commands.AutoStartOff = doc["autostartOff"];

        commands.RemoteControlOn = doc["remoteOn"];
        commands.RemoteControlOff = doc["remoteOff"];
```

```
commands.ResetPosition = doc["reset_position"];
commands.ToZeroPosition = doc["zero_position"];
SolarPanelStatus status = _manager.getPlcReceiveData();

if (status.ManualMode)
{
   const char *direction = doc["direction"];
   commands.Up = (strcmp(direction, "move_up") == 0);
   commands.Down = (strcmp(direction, "move_down") == 0);
   commands.Left = (strcmp(direction, "move_left") == 0);
   commands.Right = (strcmp(direction, "move_right") == 0);
}
else
{
   commands.Up = false;
   commands.Down = false;
   commands.Left = false;
   commands.Right = false;
}

commands.SetDateTime = doc["set_datetime"];

commands.day = doc["day"];
commands.month = doc["month"];
commands.year = doc["year"];
commands.hours = doc["hours"];
commands.minutes = doc["minutes"];
commands.seconds = doc["seconds"];
commands.weekday = doc["weekday"];
commands.SetVelocity = doc["change_velocity"];
commands.velocity = doc["velocity"];
commands.SetSunriseTime = false;
commands.SetSunsetTime = false;
commands.sunrise_hours = 0;
commands.sunrise_minutes = 0;
commands.sunset_hours = 0;
commands.sunset_minutes = 0;

printSolarPanelControlCommands(commands);
_manager.sendPlcData(commands);
_last_commands = commands;

JsonDocument responseDoc;
responseDoc["success"] = true;
```

```
    responseDoc["message"] = "Commands processed successfully";
    sendJSONResponse(request, responseDoc);
}
void SolarWebServer::handleGetPanelData(AsyncWebServerRequest *request)
{
    JsonDocument responseDoc;

    SolarPanelStatus status = _manager.getPlcReceiveData();
    responseDoc["started"] = status.Started;
    responseDoc["autoStart"] = status.AutoStart;
    responseDoc["manualMode"] = status.ManualMode;
    responseDoc["hMoveDone"] = status.H_MoveDone;
    responseDoc["hMoving"] = status.H_Moving;
    responseDoc["vMovedone"] = status.V_MoveDone;
    responseDoc["vMovving"] = status.V_Moving;
    responseDoc["sunlightFounded"] = status.Sunlight_Founded;
    responseDoc["hPosition"] = status.H_Position;
    responseDoc["vPosition"] = status.V_Position;
    responseDoc["velocity"] = status.Velocity;
    responseDoc["sunSensor"] = status.SunSensor;

    sendJSONResponse(request, responseDoc);
}

// Обробник для отримання налаштувань мережі
void SolarWebServer::handleGetNetworkSettings(AsyncWebServerRequest
*request)
{
    JsonDocument responseDoc;
    responseDoc["ssid"] = "MyNetwork";
    responseDoc["password"] = "password123";

    sendJSONResponse(request, responseDoc);
}

void SolarWebServer::handleSaveNetworkSettings(AsyncWebServerRequest
*request, uint8_t *data, size_t len, size_t index, size_t total)
{
    String body = urlDecode(data, len);
    JsonDocument doc;
    DeserializationError error = deserializeJson(doc, body);

    if (request->hasParam("body", true))
    {
```

```
            String body = request->getParam("body", true)->value();
            JsonDocument doc;
            DeserializationError error = deserializeJson(doc, body);

            if (error)
            {
               sendErrorResponse(request, "Invalid JSON");
               return;
            }
            const char *ssid = doc["ssid"];
            const char *password = doc["password"];

            if (!ssid || !password)
            {
               sendErrorResponse(request, "Missing fields");
               return;
            }

            bool success = true;
            const char *message = success ? "Network settings saved successfully" :
"Failed to save network settings";

            JsonDocument responseDoc;
            responseDoc["success"] = success;
            responseDoc["message"] = message;

            sendJSONResponse(request, responseDoc);
         }
         else
         {
            sendErrorResponse(request, "No JSON data received");
         }
      }

      void SolarWebServer::handleSensorData(AsyncWebServerRequest *request)
      {
         JsonDocument responseDoc;

         SensorData data = _manager.sensorData();
         responseDoc["temperature"] = data.temperature;
         responseDoc["humidity"] = data.humidity;
         responseDoc["current"] = data.current;
         responseDoc["voltage"] = data.voltage;
         sendJSONResponse(request, responseDoc);
```

```
}
void SolarWebServer::handleCheckConnection(AsyncWebServerRequest
*request)
{
  JsonDocument doc;
  doc["isPlcConnected"] = _manager.getPLCClient().isConnected();
  sendJSONResponse(request, doc);
}
void SolarWebServer::sendJSONResponse(AsyncWebServerRequest *request,
JsonDocument &doc)
{
  String json;
  serializeJson(doc, json);
  request->send(200, "application/json", json);
}

void SolarWebServer::sendErrorResponse(AsyncWebServerRequest *request,
const char *message, int code)
{
  JsonDocument doc;
  doc["success"] = false;
  doc["message"] = message;
  String json;
  serializeJson(doc, json);
  request->send(code, "application/json", json);
}

String SolarWebServer::urlDecode(uint8_t *data, size_t len)
{
  String input = "";
  for (size_t i = 0; i < len; i++)
  {
    input += (char)data[i];
  }

  String decoded = "";
  for (size_t i = 0; i < input.length(); i++)
  {

    if (input[i] == '%')
    {
      int value;
      sscanf(input.substring(i + 1, i + 3).c_str(), "%x", &value);
      decoded += (char)value;
```

```
            i += 2;
        }
        else if (input[i] == '+')
        {
            decoded += ' ';
        }
        else
        {
            decoded += input[i];
        }
    }
    return decoded;
}
// PLCClient.cpp
#include "PLCClient.h"

PLCClient::PLCClient(const IPAddress plcIP,const uint16_t port):
    plc_ip(plcIP), plc_port(port)
{
}

void PLCClient::Update()
{
    if (!client.connected())
    {
        if (client.connect(plc_ip, plc_port, 1000))
        {
            //DPrint("Connected to PLC.");
            _isConnected = true;
        }
        else
        {
            //DPrint("Failed to connect to PLC.");
            _isConnected = false;
            return;
        }
    }

    if (_isConnected)
    {
        if (client.available() >= receive_packet_size)
        {
            uint8_t buffer[receive_packet_size];
            size_t bytesRead = client.read(buffer, receive_packet_size);
```

```
                if (bytesRead >= receive_packet_size)
                {
                    parseReceivedData(buffer);
                }
            }

            if (sendDataStruct.Start || sendDataStruct.Stop|| sendDataStruct.AutoStartOff
||
                sendDataStruct.AutoStartOn || sendDataStruct.RemoteControlOn ||
                sendDataStruct.RemoteControlOff|| sendDataStruct.ResetPosition ||
                sendDataStruct.Left || sendDataStruct.Right || sendDataStruct.Up ||
                sendDataStruct.Down || sendDataStruct.SetSunsetTime ||
sendDataStruct.SetSunriseTime ||
                sendDataStruct.SetDateTime || sendDataStruct.SetVelocity)
            {
                send();
            }

        }
    }

    void PLCClient::sendData(const SolarPanelControlCommands& data)
    {
        sendDataStruct = data;
    }
    void PLCClient::send()
    {
        uint8_t serializedData[send_packet_size];
        memset(serializedData, 0, send_packet_size);

        serializeSolarPanelControl(serializedData, sendDataStruct);
        DPrint();
        sendPacket(serializedData, send_packet_size);
    }
    SolarPanelStatus PLCClient::getReceiveData() const
    {
        return receiveDataStruct;
    }

    bool PLCClient::isConnected()const
    {
        return _isConnected;
    }
```

```
void PLCClient::sendPacket(const uint8_t* packet, size_t length)
{
   if (length > send_packet_size) {
      DPrint("Packet length exceeds send_packet_size");
      return;
   }

   for(int i = 0; i < SolarPanelControlCommands_size; i++)
   {
      DPrint(packet[i]);
   }

   DPrint();
   client.write(packet, length);
   clearSendData();
}

void PLCClient::parseReceivedData(const uint8_t* buffer)
{
   deserializeSolarPanelStatus(buffer, receiveDataStruct);
   //printSolarPanelStatus(receiveDataStruct);
}

void PLCClient::clearSendData()
{
   memset(&sendDataStruct, 0, sizeof(SolarPanelControlCommands));
}
<!-- index.html-->
<!DOCTYPE html>
<html lang="uk">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Система контролю сонячної панелі</title>
   <link rel="stylesheet" href="css/styles.css">
   <!-- Підключення Chart.js через CDN -->
   <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
</head>
<body>
   <header>
      <div class="header-container">
         <h1 class="title">Система контролю сонячної панелі</h1>
         <div class="status-time">
            <span class="online-status offline" id="online-status">Offline</span>
```

```
            <span class="current-time" id="current-time">00:00</span>
        </div>
    </div>
</header>
<main>
    <section class="current-panel" id="current-panel" >
        <h2>Поточні дані</h2>
        <div class="grid-container">
            <div class="grid-item">
                <div class="label">Стан:</div>
                <div class="value" id="state-value">НД</div>
            </div>
            <div class="grid-item">
                <div class="label">Режим:</div>
                <div class="value" id="mode-value">НД</div>
            </div>
            <div class="grid-item">
                <div class="label">Напруга:</div>
                <div class="value" id="voltage-value">НД</div>
            </div>
            <div class="grid-item">
                <div class="label">Струм:</div>
                <div class="value" id="current-value">НД</div>
            </div>
            <div class="grid-item">
                <div class="label">Температура:</div>
                <div class="value" id="temperature-value">НД</div>
            </div>
            <div class="grid-item">
                <div class="label">Вологість:</div>
                <div class="value" id="humidity-value">НД</div>
            </div>
            <div class="grid-item">
                <div class="label">Датчик сонця:</div>
                <div class="value" id="sunsensor-value">НД</div>
            </div>
            <div></div>
            <div class="grid-item-clear">
                <div class="label">Положення</div>
            </div>
            <div></div>
            <div class="grid-item">
                <div class="label">Горизонтальне:</div>
                <div class="value" id="horizontal-value">0</div>
```

```
                </div>
                <div class="grid-item">
                    <div class="label">Вертикальне:</div>
                    <div class="value" id="vertical-value">0</div>
                </div>
            </div>
        </section>

        <section class="control-panel" id="control-panel" style="display: none;">
            <section class="no-connection-panel" id="no-connection-panel"
style="display: none;">
                <div id="no-connection" class="no-connection">
                    <p>Панель керування недоступна. Перевірте підключення.</p>
                </div>
            </section>
            <h2>Керування панеллю</h2>
            <div class="control-container">
                <div class="start-stop-buttons">
                    <button id="start-button" class="control-button gray">Start</button>
                    <button id="stop-button" class="control-button gray">Stop</button>
                    <button id="manual-button" class="control-button
gray">Manual</button>
                </div>

                <div class="position-display">
                    <div class="position-row">
                        <p>H: <span id="current-horizontal">0</span>°</p>
                        <p>V: <span id="current-vertical">0</span>°</p>
                    </div>
                    <div class="position-row">
                        <p>H: <span id="horizontal-state"></span></p>
                        <p>V: <span id="vertical-state"></span></p>
                    </div>
                </div>
                <div class="auto-start">
                    <input type="checkbox" class="auto-start-checkbox" id="auto-start-
checkbox">
                    <label>
                        Автозапуск
                    </label>

                </div>
                <div class="speed-control">
```

```
                    <label for="speed-input"><strong>Швидкість
панелі:</strong></label>
                    <input type="number" class="speed-input gray" id="speed-input"
min="0" max="10" value="5" disabled>
                </div>
                <div class="movement-control">
                    <div class="buttons-column">
                        <button id="move-up" class="gray">Вверх</button>
                        <button id="move-down" class="gray">Вниз</button>
                    </div>
                    <div class="buttons-column">
                        <button id="move-left" class=" gray">Вліво</button>
                        <button id="move-right" class=" gray">Вправо</button>
                    </div>
                </div>
                <div class="reset-position-container">
                    <button id="reset-position-button" class="control-button
gray">Скидання позиції</button>
                    <button id="zero-position-button" class="control-button gray">
Нульова позиція</button>
                </div>
            </div>
        </section>
        <section class="network-settings-panel" id="network-settings"
style="display: none;">
            <h2>Налаштування мережі</h2>
            <div class="network-container">
                <div class="network-input">
                    <label for="ssid-input"><strong>SSID:</strong></label>
                    <input type="text" id="ssid-input" placeholder="Введіть SSID"
required>
                </div>
                <div class="network-input">
                    <label for="password-input"><strong>Пароль:</strong></label>
                    <input type="password" id="password-input" placeholder="Введіть
пароль" required>
                </div>
                <div class="save-button-container">
                    <button id="save-network-button" class="save-
button">Зберегти</button>
                </div>
            </div>
        </section>
```

```
            <section class="charts-panel" id="charts-panel" style="display: none;">
                <h2>Графіки даних</h2>
                <div class="charts-container">
                    <canvas id="temperatureChart"></canvas>
                    <canvas id="currentChart"></canvas>
                    <canvas id="voltageChart"></canvas>
                    <canvas id="powerChart"></canvas>
                </div>
            </section>
        </main>
        <footer>
            <div class="nav-btns h_center" id="nav-btns">
                <button class="nav-button active" data-target="current-
panel">Поточні</button>
                <button class="nav-button" data-target="control-
panel">Керування</button>
                <button class="nav-button" data-target="network-
settings">Мережа</button>
                <!-- <button class="nav-button" data-target="charts-
panel">Графіки</button>-->
            </div>
            <div class="h_center">
                <button class="nav-button menu" id="menu-button">☰ Меню</button>
            </div>

        </footer>
        <script src="js/scripts.js"></script>
    </body>
    </html>
    document.addEventListener('DOMContentLoaded', () => {

        // API Endpoints
        const API_CONTROL_PANEL = "/api/panelcontrol";
        const API_PANEL_STATUS = "/api/paneldata";
        const API_NETWORK = "/api/network/data";
        const API_NETWORK_SAVE = "/api/network/save";
        const API_SENSOR_DATA = "/api/sensor_data";
        const API_PLC_CONNECTION_STATUS = "/api/plc/connectionstatus";
        const API_SERVER_DATA = "http://185.237.204.60:5000/api/sensor_data"; //
Видалено зайвий апостроф

        // Navigation Elements
        const navButtons = document.querySelectorAll('.nav-button:not(.menu)');
        const menuButton = document.getElementById('menu-button');
```

```
        const navBtnsContainer = document.getElementById('nav-btns');

        // Panels
        const currentPanel = document.getElementById('current-panel');
        const controlPanel = document.getElementById('control-panel');
        const networkSettingsPanel = document.getElementById('network-settings');
        const noConnectionPanel = document.getElementById('no-connection-panel');
// Виправлено орфографію
        const chartsPanel = document.getElementById('charts-panel');

        // Status Elements
        const currentTimeElement = document.getElementById('current-time');
        const onlineStatus = document.getElementById('online-status');

        // Control Buttons
        const startButton = document.getElementById('start-button');
        const stopButton = document.getElementById('stop-button');
        const manualButton = document.getElementById('manual-button');
        const resetPositionButton = document.getElementById('reset-position-button');
        const zeroPositionButton = document.getElementById('zero-position-button');
        const autoStartCheckbox = document.getElementById('auto-start-checkbox');
        const speedInput = document.getElementById('speed-input');
        const moveUpBtn = document.getElementById('move-up');
        const moveDownBtn = document.getElementById('move-down');
        const moveLeftBtn = document.getElementById('move-left');
        const moveRightBtn = document.getElementById('move-right');

        // Display Elements
        const state_value = document.getElementById('state-value');
        const mode_value = document.getElementById('mode-value');
        const voltage_value = document.getElementById('voltage-value');
        const current_value = document.getElementById('current-value');
        const temperature_value = document.getElementById('temperature-value');
        const humidity_value = document.getElementById('humidity-value');
        const horizontal_position = document.getElementById('horizontal-value');
        const vertical_position = document.getElementById('vertical-value');
        const currentHorizontal = document.getElementById('current-horizontal');
        const currentVertical = document.getElementById('current-vertical');
        const SunSensor = document.getElementById('sunsensor-value');
        const horizontal_state = document.getElementById('horizontal-state');
        const vertical_state = document.getElementById('vertical-state'); // Виправлено
'vertial-state'

        // Network Inputs
```

```
const saveNetworkButton = document.getElementById('save-network-button');
const ssidInput = document.getElementById('ssid-input');
const passwordInput = document.getElementById('password-input');

// Additional Display Elements
const speedValue = document.getElementById('speed-value'); // Додано визначення

// Movement State Variables
let up_pressed = false;
let down_pressed = false;
let left_pressed = false;
let right_pressed = false;

// Classes
class SolarPanelStatus {
    constructor() {
        this.Started = false;
        this.AutoStart = false;
        this.ManualMode = false;
        this.H_MoveDone = false;
        this.H_Moving = false;
        this.V_MoveDone = false;
        this.V_Moving = false;
        this.Sunlight_Founded = false;
        this.H_Position = 0;
        this.V_Position = 0;
        this.Velocity = 0;
        this.SunSensor = 0;
    }
}

class SensorData {
    constructor() {
        this.temperature = 0;
        this.humidity = 0;
        this.current = 0;
        this.voltage = 0;
        this.power = 0; // Додано для потужності
    }
}

// Instances
const panelStatus = new SolarPanelStatus();
```

```
        const sensorData = new SensorData();
        let timeSended = false;
        let isPLCConnected = false;

        // Function to update current time
        function updateTime() {
            const now = new Date();
            const hours = String(now.getHours()).padStart(2, '0');
            const minutes = String(now.getMinutes()).padStart(2, '0');
            currentTimeElement.textContent = `${hours}:${minutes}`; // Використано
шаблонний рядок
        }

        // Main Update Function
        function Update() {
            checkConnectionStatus();
            changeOnlineStatus();
            displayData();
            controlButtonsState();
            if(isPLCConnected) {
                if(!timeSended) {
                    controlPanelAPI("set_datetime");
                    timeSended = true;
                }
                fetchPanelStatus();
                fetchSensorData();
            }
        }

        // Initial Calls and Intervals
        setInterval(updateTime, 60000);
        setInterval(Update, 1000);
        checkConnectionStatus();
        updateTime();

        // Обробка навігаційних кнопок
        navButtons.forEach(button => {
            button.addEventListener('click', () => {

                navButtons.forEach(btn => btn.classList.remove('active'));
                button.classList.add('active');

                document.querySelectorAll('main > section').forEach(section => {
                    section.style.display = 'none';
```

```
        });

        if (window.innerWidth <= 480) {
            navBtnsContainer.classList.remove('open');
        }
        const target = button.getAttribute('data-target');
        const targetSection = document.getElementById(target);
        if (targetSection) {
            if (target === 'control-panel' && !isPLCConnected) {
                noConnectionPanel.style.display = "block";
            } else {
                noConnectionPanel.style.display = "none";
            }
            targetSection.style.display = "block";
            if(target === 'charts-panel'){
                initializeCharts();
            }
        }

    });
});

// Обробка кнопки меню
menuButton.addEventListener('click', () => {
    navBtnsContainer.classList.toggle('open');
});

// Функція для відображення даних
function displayData() {
    state_value.textContent = panelStatus.Started ? "Запущений" : "Зупинений";
    mode_value.textContent = panelStatus.ManualMode ?  "Ручний" : "Авто";
    autoStartCheckbox.checked = panelStatus.AutoStart;
    speedInput.value = panelStatus.Velocity;
    voltage_value.textContent = sensorData.voltage;
    current_value.textContent = sensorData.current;
    temperature_value.textContent = sensorData.temperature;
    humidity_value.textContent = sensorData.humidity;
    horizontal_position.textContent = panelStatus.H_Position;
    vertical_position.textContent = panelStatus.V_Position;
    currentHorizontal.textContent = panelStatus.H_Position;
    currentVertical.textContent = panelStatus.V_Position;
    SunSensor.textContent = panelStatus.SunSensor;

    // Обчислення потужності
```

```
    sensorData.power = sensorData.voltage * sensorData.current;

    if (panelStatus.H_Moving) {
        horizontal_state.textContent = "Moving";
    } else if (panelStatus.H_MoveDone) {
        horizontal_state.textContent = "Move done";
    } else{
        horizontal_state.textContent = "Stop";
    }
    if (panelStatus.V_Moving) {
        vertical_state.textContent = "Moving";
    } else if (panelStatus.V_MoveDone) {
        vertical_state.textContent = "Move done";
    } else{
        vertical_state.textContent = "Stop";
    }
}

// Функція для зміни статусу онлайн
function changeOnlineStatus() {
    if(isPLCConnected) {
        onlineStatus.classList.remove("offline");
        onlineStatus.classList.add("online");
        onlineStatus.textContent = "PLC Online";
        return;
    }
    onlineStatus.classList.remove("online");
    onlineStatus.classList.add("offline");
    onlineStatus.textContent = "PLC Offline";
}

// Функція для контролю стану кнопок
function controlButtonsState() {
    isPLCConnected = true;

    if(isPLCConnected) {
        if (panelStatus.Started) {
            startButton.classList.add('green');
        } else {
            startButton.classList.remove('green');
        }

        if (!panelStatus.Started) {
            stopButton.classList.add('red');
```

```
    } else {
        stopButton.classList.remove('red');
    }
    if(panelStatus.ManualMode) {
        manualButton.classList.add("green");
    } else{
        manualButton.classList.remove("green");
    }
    startButton.classList.remove("gray");
    stopButton.classList.remove("gray");
    manualButton.classList.remove("gray");
} else{
    startButton.classList.add("gray");
    stopButton.classList.add("gray");
    manualButton.classList.add("gray");
}

if(isPLCConnected && panelStatus.ManualMode)
{
    if(panelStatus.V_Moving) {
        if(up_pressed) {
            moveUpBtn.classList.add("green");
        } else if(down_pressed){
            moveDownBtn.classList.add("green");
        }
    } else{
        down_pressed = false;
        up_pressed = false;
        moveUpBtn.classList.remove("green");
        moveDownBtn.classList.remove("green");
    }
    if(panelStatus.H_Moving) {
        if(left_pressed) {
            moveLeftBtn.classList.add("green");
        } else if(right_pressed){
            moveRightBtn.classList.add("green");
        }
    } else{
        moveLeftBtn.classList.remove("green");
        moveRightBtn.classList.remove("green");
    }
    resetPositionButton.classList.remove("gray");
    zeroPositionButton.classList.remove("gray");
    moveUpBtn.classList.remove("gray");
```

```
        moveDownBtn.classList.remove("gray");
        moveLeftBtn.classList.remove("gray");
        moveRightBtn.classList.remove("gray");
        speedInput.classList.remove("gray");
    } else{
        resetPositionButton.classList.add("gray");
        zeroPositionButton.classList.add("gray");
        moveUpBtn.classList.add("gray");
        moveDownBtn.classList.add("gray");
        moveLeftBtn.classList.add("gray");
        moveRightBtn.classList.add("gray");
        speedInput.classList.add("gray");
    }
    return
    startButton.disabled = !isPLCConnected;
    stopButton.disabled =  !isPLCConnected;
    manualButton.disabled =  !isPLCConnected;
    resetPositionButton.disabled =  !isPLCConnected;
    zeroPositionButton.disabled =  !isPLCConnected;
    moveUpBtn.disabled =   !isPLCConnected;
    moveDownBtn.disabled =  !isPLCConnected;
    moveLeftBtn.disabled =   !isPLCConnected;
    moveRightBtn.disabled =  !isPLCConnected;
    speedInput.disabled = !isPLCConnected;
}

// Слухачі подій для кнопок управління
startButton.addEventListener('click', () => {
    if (!isPLCConnected) {
        console.error('Cannot send command start: No connection to PLC.');
        return;
    }
    controlPanelAPI('start');
});

stopButton.addEventListener('click', () => {
    if (!isPLCConnected) {
        console.error('Cannot send command stop: No connection to PLC.');
        return;
    }
    controlPanelAPI('stop');
});

manualButton.addEventListener('click', () => {
```

```
            if (isPLCConnected) {
                if (!panelStatus.ManualMode) {
                    controlPanelAPI('remoteOn');
                } else {
                    controlPanelAPI('remoteOff');
                }
            } else {
                console.error('Cannot send command remote: No connection to PLC.');
            }
        });

        resetPositionButton.addEventListener('click', () => {
            if (!isPLCConnected) {
                console.error('Cannot send command reset_position: No connection to
PLC.');
                return;
            }
            if(!panelStatus.ManualMode) // Виправлено manualMode на ManualMode
            {
                blinkButton(manualButton, "red", 3, 500);
                return; // Додано повернення, щоб не виконувати команду при
невірному режимі
            }
            controlPanelAPI('reset_position');
        });

        zeroPositionButton.addEventListener('click', () => {
            if (!isPLCConnected) {
                console.error('Cannot send command zero_position: No connection to
PLC.');
                return;
            }
            if(!panelStatus.ManualMode)
            {
                blinkButton(manualButton, "red", 3, 500);
                return;
            }
            controlPanelAPI('zero_position');
        });

        autoStartCheckbox.addEventListener('change', () => {
            if (!isPLCConnected) {
                console.error('Cannot send command autostart: No connection to PLC.');
                return;
```

```
    }
    const autoStart = autoStartCheckbox.checked;
    if (autoStart) {
        controlPanelAPI("autostartOn");
    } else {
        controlPanelAPI("autostartOff");
    }
});

moveUpBtn.addEventListener('click', () => movePanel('up'));
moveDownBtn.addEventListener('click', () => movePanel('down'));
moveLeftBtn.addEventListener('click', () => movePanel('left'));
moveRightBtn.addEventListener('click', () => movePanel('right'));

// Функція для миготіння кнопки
function blinkButton(button, color, times, interval) {
    let count = 0;
    const originalColor = button.style.backgroundColor;

    const blinkInterval = setInterval(() => {
        if (count < times * 2) {
            button.style.backgroundColor =
                button.style.backgroundColor === color ? originalColor : color;
            count++;
        } else {
            clearInterval(blinkInterval);
            button.style.backgroundColor = originalColor;
        }
    }, interval);
}

// Функція для руху панелі
function movePanel(direction) {
    if (!isPLCConnected) {
        console.error('Cannot send command move_panel: No connection to
PLC.');
        return;
    }
    let value = '';
    if (direction === 'up') {
        up_pressed = true;
        value = 'move_up';
    } else if (direction === 'down') {
        down_pressed = true;
```

```
        value = 'move_down';
    } else if (direction === 'left') {
        left_pressed = true;
        value = 'move_left';
    } else if (direction === 'right') {
        right_pressed = true;
        value = 'move_right';
    }
    controlPanelAPI('direction', value);
}

// Обробка зміни швидкості
speedInput.addEventListener('input', () => {
    let value = parseInt(speedInput.value, 10);
    if (isNaN(value) || value < 0) {
        value = 0;
    } else if (value > 10) {
        value = 10;
    }
    speedInput.value = value;
    speedValue.textContent = value;
    controlPanelAPI('change_velocity', value);
});

// Обробка збереження мережевих налаштувань
saveNetworkButton.addEventListener('click', () => {
    const ssid = ssidInput.value.trim();
    const password = passwordInput.value.trim();
    if (ssid === '' || password === '') {
        alert('Будь ласка, заповніть всі поля.');
        return;
    }
    const credentials = {
        ssid: ssid,
        password: password
    };
    fetch(API_NETWORK_SAVE, {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify(credentials)
    })
    .then(response => {
```

```
        if (response.ok) {
            alert('Налаштування мережі збережено успішно!');
            ssidInput.value = '';
            passwordInput.value = '';
        } else {
            alert('Помилка при збереженні налаштувань мережі.');
        }
    })
    .catch(error => {
        console.error('Error:', error);
        alert('Сталася помилка при збереженні налаштувань мережі.');
    });
});

// Функція для отримання статусу панелі
function fetchPanelStatus() {
    fetch(API_PANEL_STATUS)
        .then(response => {
            if (!response.ok) {
                console.error(`HTTP error! Status: ${response.status}`);
                return;
            }
            return response.json();
        })
        .then(data => {
            if (!data) return;
            panelStatus.Started = data.started;
            panelStatus.AutoStart = data.autoStart;
            panelStatus.ManualMode = data.manualMode;
            panelStatus.H_MoveDone = data.hMoveDone;
            panelStatus.H_Moving = data.hMoving;
            panelStatus.V_MoveDone = data.vMoveDone;
            panelStatus.V_Moving = data.vMoving;
            panelStatus.Sunlight_Founded = data.sunlightFounded;
            panelStatus.H_Position = data.hPosition;
            panelStatus.V_Position = data.vPosition;
            panelStatus.Velocity = data.velocity;
            panelStatus.SunSensor = data.sunSensor;
        })
        .catch(error => {
            console.error('Error fetching panel status:', error);
        });
}
```

```
// Функція для отримання даних сенсорів
function fetchSensorData() {
    fetch(API_SENSOR_DATA)
        .then(response => {
            if (!response.ok) {
                console.error(`HTTP error! Status: ${response.status}`);
                return;
            }
            return response.json();
        })
        .then(data => {
            if (!data) return;
            console.log(data);
            sensorData.voltage = data.voltage;
            sensorData.humidity = data.humidity;
            sensorData.temperature = data.temperature;
            sensorData.current = data.current;
            sensorData.power = data.power || (data.voltage * data.current);
            updateCharts();
        })
        .catch(error => {
            console.error('Error fetching sensor data:', error);
        });
}

// Функція для взаємодії з API контролю панелі
function controlPanelAPI(command, value = null) {
    const now = new Date();

    const body = {
        start: command === "start",
        stop: command === "stop",
        autostartOn: command === "autostartOn",
        autostartOff: command === "autostartOff",
        remoteOn: command === "remoteOn",
        remoteOff: command === "remoteOff",
        reset_position: command === "reset_position",
        zero_position: command === "zero_position",
        direction: command === "direction" ? value : '',
        set_datetime: command === "set_datetime",
        weekday: command === "set_datetime" ? now.getDay() : 0,
        day: command === "set_datetime" ? now.getDate() : 0,
        month: command === "set_datetime" ? now.getMonth() : 0,
        year: command === "set_datetime" ? now.getFullYear() : 0,
```

```
        hours: command === "set_datetime" ? now.getHours() : 0,
        minutes: command === "set_datetime" ? now.getMinutes() : 0,
        seconds: command === "set_datetime" ? now.getSeconds() : 0,
        change_velocity: command === "change_velocity",
        velocity: command === "change_velocity" ? value : 0
    };

    fetch(API_CONTROL_PANEL, {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify(body)
    })
    .catch(error => {
        console.error(`Error sending command ${command}:`, error);
    });
}

// Функція для отримання мережевих налаштувань
function fetchNetworkSettings() {
    fetch(API_NETWORK)
        .then(response => {
            if (!response.ok) {
                console.error(`HTTP error! Status: ${response.status}`);
                return;
            }
            return response.json();
        })
        .then(data => {
            if (!data) return;
            ssidInput.value = data.ssid || '';
            passwordInput.value = data.password || '';
        })
        .catch(error => {
            console.error('Error fetching network settings:', error);
        });
}

// Функція для перевірки статусу з'єднання з PLC
function checkConnectionStatus() {
    isPLCConnected = true
    return
    fetch(API_PLC_CONNECTION_STATUS,
```

```
            {
                method: "GET",
                headers: {
                    "Content-Type": "application/json",
                },
            }
        )
            .then(response => {
                if (!response.ok) {
                    console.error(`HTTP error! Status: ${response.status}`);
                    isPLCConnected = false;
                    return;
                }
                const contentType = response.headers.get('Content-Type');
                if (!contentType || !contentType.includes('application/json')) {
                    console.error("Expected JSON, got " + contentType);
                    isPLCConnected = false;
                    return;
                }
                return response.json();
            })
            .then(data => {
                if (!data) return;
                isPLCConnected = data.isPlcConnected;
            })
            .catch(error => {
                console.error('Error checking connection status:', error);
                isPLCConnected = false;
            });
    }

    // Початкове отримання мережевих налаштувань
    fetchNetworkSettings();

    // ---------- Додано для графіків ----------

    // Ініціалізація графіків
    let temperatureChartInstance, currentChartInstance, voltageChartInstance,
powerChartInstance;

    function initializeCharts() {
        const ctxTemp =
document.getElementById('temperatureChart').getContext('2d');
```

```
        const ctxCurrent =
document.getElementById('currentChart').getContext('2d');
        const ctxVoltage =
document.getElementById('voltageChart').getContext('2d');
        const ctxPower = document.getElementById('powerChart').getContext('2d');

        temperatureChartInstance = new Chart(ctxTemp, {
            type: 'line',
            data: {
                labels: [],
                datasets: [{
                    label: 'Температура (°C)',
                    data: [],
                    borderColor: 'rgba(255, 99, 132, 1)',
                    backgroundColor: 'rgba(255, 99, 132, 0.2)',
                    fill: true,
                }]
            },
            options: {
                responsive: true,
                maintainAspectRatio: false,
                scales: {
                    x: {
                        title: {
                            display: true,
                            text: 'Час'
                        }
                    },
                    y: {
                        title: {
                            display: true,
                            text: 'Температура (°C)'
                        }
                    }
                }
            }
        });

        currentChartInstance = new Chart(ctxCurrent, {
            type: 'line',
            data: {
                labels: [],
                datasets: [{
                    label: 'Струм (A)',
```

```
            data: [],
            borderColor: 'rgba(54, 162, 235, 1)',
            backgroundColor: 'rgba(54, 162, 235, 0.2)',
            fill: true,
        }]
    },
    options: {
        responsive: true,
        maintainAspectRatio: false,
        scales: {
            x: {
                title: {
                    display: true,
                    text: 'Час'
                }
            },
            y: {
                title: {
                    display: true,
                    text: 'Струм (A)'
                }
            }
        }
    }
});

voltageChartInstance = new Chart(ctxVoltage, {
    type: 'line',
    data: {
        labels: [],
        datasets: [{
            label: 'Напруга (V)',
            data: [],
            borderColor: 'rgba(255, 206, 86, 1)',
            backgroundColor: 'rgba(255, 206, 86, 0.2)',
            fill: true,
        }]
    },
    options: {
        responsive: true,
        maintainAspectRatio: false,
        scales: {
            x: {
                title: {
```

```
                display: true,
                text: 'Час'
            }
        },
        y: {
            title: {
                display: true,
                text: 'Напруга (V)'
            }
        }
    }
  }
});

powerChartInstance = new Chart(ctxPower, {
  type: 'line',
  data: {
    labels: [],
    datasets: [{
      label: 'Потужність (W)',
      data: [],
      borderColor: 'rgba(75, 192, 192, 1)',
      backgroundColor: 'rgba(75, 192, 192, 0.2)',
      fill: true,
    }]
  },
  options: {
    responsive: true,
    maintainAspectRatio: false,
    scales: {
      x: {
        title: {
          display: true,
          text: 'Час'
        }
      },
      y: {
        title: {
          display: true,
          text: 'Потужність (W)'
        }
      }
    }
  }
```

```
    });

    fetchAndPopulateCharts();
    setInterval(fetchAndPopulateCharts, 300000); // Оновлення кожні 5 хвилин
}

function fetchAndPopulateCharts() {
    // Використовуємо API_SERVER_DATA з параметром періоду
    fetch(`${API_SERVER_DATA}?period=day`)
        .then(response => {
            if (!response.ok) {
                console.error(`HTTP error! Status: ${response.status}`);
                return;
            }
            return response.json();
        })
        .then(data => {
            if (!data) return;
            const labels = data.map(entry => new Date(entry.created_at).toLocaleTimeString());
            const temperatures = data.map(entry => entry.temperature);
            const currents = data.map(entry => entry.current);
            const voltages = data.map(entry => entry.voltage);
            const powers = data.map(entry => entry.power || (entry.voltage * entry.current));

            updateChart(temperatureChartInstance, labels, temperatures);
            updateChart(currentChartInstance, labels, currents);
            updateChart(voltageChartInstance, labels, voltages);
            updateChart(powerChartInstance, labels, powers);
        })
        .catch(error => {
            console.error('Error fetching data for charts:', error);
        });
}

function updateChart(chart, labels, data) {
    chart.data.labels = labels;
    chart.data.datasets[0].data = data;
    chart.update();
}

function updateCharts() {
    const now = new Date().toLocaleTimeString();
```

```
            // Додаємо нові дані до графіків
            if (temperatureChartInstance) {
                temperatureChartInstance.data.labels.push(now);

temperatureChartInstance.data.datasets[0].data.push(sensorData.temperature);
                if (temperatureChartInstance.data.labels.length > 50) { // Обмеження на кількість точок
                    temperatureChartInstance.data.labels.shift();
                    temperatureChartInstance.data.datasets[0].data.shift();
                }
                temperatureChartInstance.update();
            }
            if (currentChartInstance) {
                currentChartInstance.data.labels.push(now);
                currentChartInstance.data.datasets[0].data.push(sensorData.current);
                if (currentChartInstance.data.labels.length > 50) {
                    currentChartInstance.data.labels.shift();
                    currentChartInstance.data.datasets[0].data.shift();
                }
                currentChartInstance.update();
            }
            if (voltageChartInstance) {
                voltageChartInstance.data.labels.push(now);
                voltageChartInstance.data.datasets[0].data.push(sensorData.voltage);
                if (voltageChartInstance.data.labels.length > 50) {
                    voltageChartInstance.data.labels.shift();
                    voltageChartInstance.data.datasets[0].data.shift();
                }
                voltageChartInstance.update();
            }
            if (powerChartInstance) {
                powerChartInstance.data.labels.push(now);
                powerChartInstance.data.datasets[0].data.push(sensorData.power);
                if (powerChartInstance.data.labels.length > 50) {
                    powerChartInstance.data.labels.shift();
                    powerChartInstance.data.datasets[0].data.shift();
                }
                powerChartInstance.update();
            }
        }

        // Періодична відправка даних
        setInterval(() => {
            if (navigator.onLine) {
```

```
        sendSensorData();
    } else {
        console.warn('No internet connection. Data not sent.');
    }
}, 1800000); // Відправка кожні 30 хвилин

function sendSensorData() {
    const data = {
        current: sensorData.current,
        voltage: sensorData.voltage,
        temperature: sensorData.temperature,
        humidity: sensorData.humidity,
        power: sensorData.power
    };

    fetch(API_SENSOR_DATA, {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify(data)
    })
    .then(response => {
        if (response.ok) {
            console.log('Sensor data sent successfully.');
        } else {
            console.error('Failed to send sensor data.');
        }
    })
    .catch(error => {
        console.error('Error sending sensor data:', error);
    });
}

window.addEventListener('online', () => {
    console.log('You are online.');
    onlineStatus.classList.remove("offline");
    onlineStatus.classList.add("online");
    onlineStatus.textContent = "Інтернет доступний";
    sendSensorData();
});

window.addEventListener('offline', () => {
    console.log('You are offline.');
```

```
            onlineStatus.classList.remove("online");
            onlineStatus.classList.add("offline");
            onlineStatus.textContent = "Інтернет недоступний";
        });

        // ---------- Додано для графіків ----------

        // Перевірка, чи існує панель графіків перед ініціалізацією
        if (chartsPanel) {
            initializeCharts();
        }


});
```

ДОДАТОК Б. Серверний код

```
from flask import Flask, request, jsonify
import mysql.connector
from mysql.connector import Error
from datetime import datetime, timedelta
import requests
from flask_cors import CORS

app = Flask(__name__)

# Константи для роботи з OpenWeatherMap
WEATHER_API_KEY = "84973d47c049e27f71f4c55c0476bb51"
CITY = "Kryvyi Rih"
WEATHER_URL =
f"http://api.openweathermap.org/data/2.5/weather?q={CITY}&units=metric&appid={
WEATHER_API_KEY}"

CORS(app)

# Налаштування підключення до бази даних
def connect_to_database():
    try:
        connection = mysql.connector.connect(
            host="localhost",
            user="root",
            password="123b123bb",
            database="sensor_data"
        )
```

```
            return connection
        except Error as e:
            print(f"Error: {e}")
            return None

    # Функція для отримання поточної погоди
    def get_weather():
        try:
            response = requests.get(WEATHER_URL)
            if response.status_code == 200:
                weather_data = response.json()
                return {
                    "temperature": weather_data['main']['temp'],
                    "humidity": weather_data['main']['humidity'],
                    "description": weather_data['weather'][0]['description']
                }
            else:
                print(f"Не вдалося отримати дані про погоду: {response.status_code}")
                return None
        except Exception as e:
            print(f"Помилка при отриманні даних про погоду: {e}")
            return None

    # Маршрут для прийому даних
    @app.route('/api/sensor_data', methods=['POST'])
    def receive_data():
        data = request.json
        connection = connect_to_database()
        if connection and data:
            try:
                cursor = connection.cursor(dictionary=True)

                # Перевірка часу останнього додавання
                cursor.execute("SELECT MAX(created_at) AS last_entry FROM
sensor_parameters")
                last_entry = cursor.fetchone()

                if last_entry['last_entry']:
                    last_entry_time = last_entry['last_entry']
                    now = datetime.now()
                    time_difference = now - last_entry_time

                    if time_difference < timedelta(minutes=30):
                        next_allowed_time = last_entry_time + timedelta(minutes=30)
```

```
                    return jsonify({
                        "message": "Дані не вставлено. Будь ласка, зачекайте.",
                        "next_allowed_time": next_allowed_time.strftime('%Y-%m-%d
%H:%M:%S')
                    }), 400

                # Отримання поточної погоди
                weather = get_weather()
                if not weather:
                    return jsonify({"error": "Не вдалося отримати дані про погоду"}),
500

                # Вставка нових даних
                insert_query = """
                    INSERT INTO sensor_parameters
                    (current, voltage, temperature, humidity, power, weather_temperature,
weather_humidity, weather_description)
                    VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
                """
                data_tuple = (
                    data.get('current'),
                    data.get('voltage'),
                    data.get('temperature'),
                    data.get('humidity'),
                    data.get('power'),
                    weather['temperature'],
                    weather['humidity'],
                    weather['description']
                )
                cursor.execute(insert_query, data_tuple)
                connection.commit()
                return jsonify({"message": "Дані успішно вставлено"}), 200
            except Error as e:
                return jsonify({"error": str(e)}), 500
            finally:
                cursor.close()
                connection.close()
        return jsonify({"error": "Не вдалося підключитися або дані недійсні"}), 400


    @app.route('/api/period_sensor_data', methods=['GET'])
    def get_data_by_time():
        period = request.args.get('period', 'day')
        connection = connect_to_database()
```

```
    if connection:
        try:
            cursor = connection.cursor(dictionary=True)
            query = """
                SELECT * FROM sensor_parameters
                WHERE created_at >= %s
                ORDER BY created_at DESC
            """
            now = datetime.now()
            if period == 'day':
                start_time = now - timedelta(days=1)
            elif period == 'week':
                start_time = now - timedelta(weeks=1)
            elif period == 'month':
                start_time = now - timedelta(days=30)
            elif period == 'year':
                start_time = now - timedelta(days=365)
            else:
                return jsonify({"error": "Невірний період"}), 400

            cursor.execute(query, (start_time,))
            results = cursor.fetchall()
            return jsonify(results), 200
        except Error as e:
            return jsonify({"error": str(e)}), 500
        finally:
            cursor.close()
            connection.close()
    return jsonify({"error": "Не вдалося підключитися до бази даних"}), 400

@app.route('/api/latest_sensor_data', methods=['GET'])
def get_latest_data():
    connection = connect_to_database()
    if connection:
        try:
            cursor = connection.cursor(dictionary=True)
            query = """
                SELECT * FROM sensor_parameters
                ORDER BY created_at DESC
                LIMIT 1
            """
            cursor.execute(query)
            result = cursor.fetchone()
            if result:
```

```
            return jsonify(result), 200
        else:
            return jsonify({"message": "Немає даних"}), 404
    except Error as e:
        return jsonify({"error": str(e)}), 500
    finally:
        cursor.close()
        connection.close()
  return jsonify({"error": "Не вдалося підключитися до бази даних"}), 400
if __name__ == '__main__':
  app.run(host='0.0.0.0', port=5000)
```

ДОДАТОК В. Проект методичного забезпечення

Лабораторна робота №1

Тема: Дослідження веб-інтерфейсу системи контролю сонячної панелі

Мета: Вивчення структури, основних вікон та можливостей керування сонячною панеллю через веб-інтерфейс. Ознайомлення зі способами відображення даних та інтерактивного управління системою.

Теоретичні відомості

Веб-інтерфейс є ключовим елементом сучасних систем моніторингу та управління, оскільки забезпечує користувачам зручний спосіб взаємодії з пристроями через веб-браузер. У контексті системи контролю сонячної панелі веб-інтерфейс дозволяє відображати актуальні дані, управляти параметрами роботи панелі, налаштовувати мережеві підключення та аналізувати історичні дані.

Основні компоненти веб-інтерфейсу:

Header (Заголовок): Містить назву системи, статус підключення до PLC (Programmable Logic Controller) та поточний час.

Main (Основний вміст):

Поточні дані (Current Panel): Відображає актуальні параметри роботи сонячної панелі, такі як температура, вологість, струм, напруга, потужність, дані з датчиків сонця та положення панелі.

Керування панеллю (Control Panel): Надає інструменти для запуску та зупинки системи, перемикання між ручним та автоматичним режимами роботи, регулювання швидкості та орієнтації панелі.

Налаштування мережі (Network Settings Panel): Дозволяє налаштовувати параметри підключення до Wi-Fi мережі, включаючи введення SSID та пароля.

Графіки даних (Charts Panel): Відображає графіки зміни параметрів системи протягом часу, доступні лише на контролері ESP8266.

Footer (Футер): Містить кнопки навігації між різними секціями інтерфейсу та меню для мобільних пристроїв.

Функціональні можливості веб-інтерфейсу:

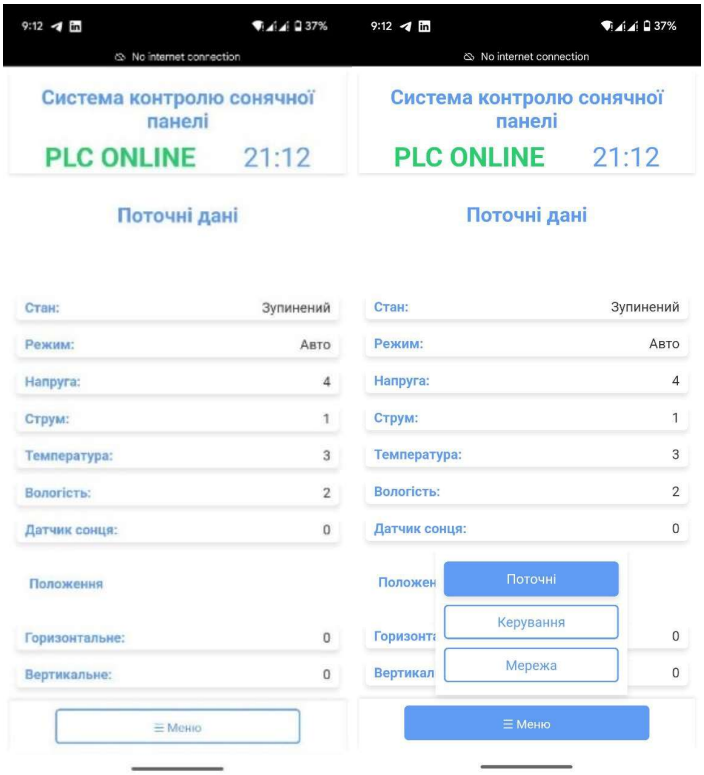


Рисунок 3.13 - веб-інтерфейсу системи контролю сонячної панелі

Моніторинг: Відображення поточних значень параметрів роботи сонячної панелі в реальному часі.

Управління: Можливість запуску/зупинки системи, перемикання режимів роботи, регулювання швидкості та орієнтації панелі.

Налаштування: Опції для налаштування параметрів підключення до мережі Wi-Fi.

Візуалізація: Графічне представлення історичних даних для аналізу продуктивності системи.

Хід роботи

1. Підготовка до роботи:

Переконайтеся, що система увімкнена.

Переконайтесь, що серверна частина системи контролю сонячної панелі налаштована та працює (описана в лабораторній роботі №2).

2. Ознайомлення з веб-інтерфейсом:

Підключіться точки доступу Wifi, ssid - SolarPanelLocal

Відкрийте веб-браузер (наприклад, Google Chrome, Mozilla Firefox) на мобільному пристрої.

Введіть IP-адресу 185.197.204.60 для підключення до веб-сервера пристрою WT32-ETH01 системи контролю сонячної панелі у веб-браузері.

Ознайомтесь із загальним виглядом веб-інтерфейсу, звернувши увагу на наступні секції:

Header (Заголовок): Перевірте відображення назви системи, статусу підключення до PLC та поточного часу.

Main (Основний вміст):

Поточні дані (Current Panel): Перегляньте відображення параметрів, таких як температура, вологість, струм, напруга, потужність та інші дані.

Керування панеллю (Control Panel): Ознайомтесь із можливістю запуску та зупинки системи, перемикання режимів роботи та регулювання орієнтації панелі.

Налаштування мережі (Network Settings Panel): Перевірте опції для налаштування підключення до Wi-Fi.

Графіки даних (Charts Panel): Перегляньте графіки для моніторингу історичних даних (доступно на ESP8266).

Footer (Футер): Ознайомтесь із навігаційними кнопками та меню для мобільних пристроїв.

Аналіз функціональних можливостей:

Поточні дані:

Перевірте правильність відображення значень параметрів.

Визначте, як часто оновлюються ці дані (наприклад, кожні 60 секунд).

Керування панеллю:

Спробуйте запустити та зупинити систему за допомогою кнопок Start/Stop.

Перемикайтеся між ручним та автоматичним режимами роботи панелі та спостерігайте за змінами.

Використовуйте інструменти для регулювання швидкості та орієнтації панелі (наприклад, кнопки для переміщення вгору, вниз, вліво, вправо).

Налаштування мережі:

Перейдіть до секції налаштувань мережі.

Введіть нові SSID та пароль для підключення до Wi-Fi або до телефонної точки доступу з роздачою мережі (за необхідності).

Збережіть налаштування та перевірте, чи система успішно підключається до мережі.

Після цього роздачі увімкніть точку доступу або підключіться до Wifi до якого були введені дані.

Оновіть сторінку або повторно відкрийте браузер і уведіть той самий ip адрес.

Графіки даних:

Ознайомтесь із графіками температури, струму, напруги та потужності.

Перевірте, як графіки оновлюються в реальному часі та їх адаптивність при зміні розміру вікна браузера.

Практичне завдання:

Аналіз структури веб-інтерфейсу:

Відкрийте інструменти розробника у вашому браузері на ноутбуці (натиснувши F12) або отримайте вихідний код сторінки, щоб проаналізувати структуру HTML та CSS веб-сайту.

Звіт

1. Визначте основні компоненти та їх взаємозв’язки.
2. Зверніть увагу на використані стилі, класи та ідентифікатори.
3. Оформіть звіт, включаючи:
4. Опис структури веб-інтерфейсу.
5. Скріншоти основних секцій веб-інтерфейсу до та після модифікації.
6. Опис функціональних можливостей та їх використання.
7. Висновки щодо зручності та ефективності веб-інтерфейсу для керування системою контролю сонячної панелі.

Лабораторна робота №2

Тема: Розробка методів підключення та отримання даних з серверної частини системи контролю сонячної панелі

Мета: Навчитися створювати методи на JavaScript для підключення до серверної частини системи контролю сонячної панелі, отримання даних та виконання базового аналізу цих даних.

Теоретичні відомості

Для ефективної взаємодії між клієнтським додатком та серверною частиною системи контролю сонячної панелі необхідно розуміти основні принципи роботи з веб-запитами та обробки отриманих даних. Основним методом для цього є використання Fetch API для здійснення HTTP-запитів до серверу, а також обробка отриманих даних для подальшого аналізу та візуалізації.

Хід роботи

1. Підготовка середовища:

Інструменти: Веб-браузер (наприклад, Google Chrome, Mozilla Firefox) та текстовий редактор (наприклад, Visual Studio Code).

2. Створення проекту:

Створіть нову папку для проекту, наприклад, solar_panel_client_methods.

Всередині цієї папки створіть два файли:

index.html

scripts.js

Розробка методів підключення та отримання даних:

Ваша задача полягає у створенні методів на JavaScript, які дозволять вашому клієнтському додатку підключатися до серверної частини, отримувати актуальні дані та виконувати базовий аналіз цих даних.

3. Підключення до серверу:

Використовуйте Fetch API для здійснення HTTP-запитів до серверу з IP-адресою http://185.237.204.60:5000.

4. Отримання даних:

Створіть метод, який надсилатиме GET-запит до серверного API для отримання останніх даних сенсорів.

API_LATEST_DATA ='http://185.237.204.60:5000/api/latest_sensor_data' – отримання останіх даних з датчиків

API_PERIOD_DATA = 'http://185.237.204.60:5000/api/period_sensor_data' - отримання даних за період

Приклад коду для виконання роботи можна знайти в ДОДАТОК В.

Обробіть відповідь від серверу, переконайтесь, що дані отримано успішно, і вони знаходяться у форматі JSON.

5. Обробка даних

Після отримання даних від серверу необхідно їх обробити для подальшого аналізу. Для цього використовується JSON-формат, який дозволяє легко парсити дані в JavaScript.

Виконайте базовий аналіз, наприклад, обчисліть середню температуру або максимальну потужність за певний період.

6. Візуалізація та аналіз:

Використовуйте отримані дані для створення графіків або таблиць.

Реалізуйте функції для динамічного оновлення відображення даних у веб-інтерфейсі.

Для візуалізації отриманих даних використовується бібліотека Chart.js, яка дозволяє створювати інтерактивні графіки.

<script src="https://cdn.jsdelivr.net/npm/chart.js"></script>

У файлі index.html додайте елементи <canvas> для відображення графіків:

<canvas id="temperatureChart" width="400" height="200"></canvas>
<canvas id="powerChart" width="400" height="200"></canvas>

Створення функцій для побудови графіків

7. Ініціалізація та динамічне оновлення графіків

Додайте наступний код у файл scripts.js для ініціалізації графіків та їх автоматичного оновлення:

Практичне завдання

Після реалізації методів підключення, отримання та обробки даних, а також їх візуалізації, необхідно протестувати роботу клієнтської частини з сервером.

Перевірка отримання останніх даних сенсорів

Відкрийте веб-сторінку клієнтського додатку (index.html) у вашому веб-браузері.

Перевірте, чи відображаються останні дані сенсорів у консолі браузера та на графіках.

Перевірка отримання даних за період

Змініть параметр періоду у виклику fetchSensorData('week') у файлі scripts.js та переконайтесь, що дані за тиждень завантажуються правильно.

Перевірте коректність відображення на графіках.

Тестування обробки помилок

Вимкніть сервер та спробуйте отримати дані, перевірте, чи коректно обробляються помилки.

Перевірте, чи відображаються відповідні повідомлення користувачу в консолі браузера.

Звіт

1. Опис структури веб-інтерфейсу

2. Скриншоти основних секцій веб-інтерфейсу

3. Опис функціональних можливостей та їх використання

4. Висновки щодо зручності та ефективності веб-інтерфейсу для керування системою контролю сонячної панелі