Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“____” _____ 2025 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за першим (бакалаврським) рівнем вищої освіти
на тему
“Програмне забезпечення системи автоматизації сонячно-вітрової електростанції”

Виконав здобувач вищої освіти
IV курсу, групи КІ-21-2
ОПП «Комп’ютерна інженерія»
спеціальності 123 «Комп’ютерна інженерія»
_____ Брагінець С.О.
«____» _____ 2025 р.

Керівник проекту
кандидат технічних наук, доцент
_____ Дрєєв О.М.
«____» _____ 2025 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет
Факультет *Механіко-технологічний*
Кафедра *Кібербезпеки та програмного забезпечення*
Освітній ступінь *бакалавр*
Галузь знань ._12 _____*"*Інформаційні технології*"*_____
Спеціальність *123 "*Комп'ютерна інженерія*"*
Освітньо-професійна (освітньо-наукова) програма _*"*Комп'ютерна інженерія*"*_

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____ Олексій СМІРНОВ
« 17 » січня 2025 року

ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ПЕРШИМ (БАКАЛАВРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Брагінцю Станіславу Олександровичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Програмне забезпечення системи автоматизації сонячно-вітрової електростанції*

2. Керівник роботи *Дрєєв Олександр Миколайович. кандидат технічних наук, доцент*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 47-02 від 17.01.2025 року

3. Строк подання студентом роботи до захисту *23.05.2025 р.*

4. Мета та завдання випускної кваліфікаційної роботи*: Метою роботи є розробка програмного забезпечення системи автоматизації сонячно-вітрової електростанції*

5. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Призначення та область використання.
2.Перегляд аналогічних існуючих систем.
3. Опис і обґрунтування проектних рішень.
4. Етапи програмування системи.
5. Впровадження системи в промислову експлуатацію.
6. Висновки

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

Структурна схема системи	*1 аркуш*
Функціональна схема системи	*1 аркуш*
Діаграма процесів	*1 аркуш*
Блок-схема алгоритму роботи додатку	*2 аркуша*

7. Дата видачі завдання « 17 » січня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем	10.03.2025 р.	
2.	Постановка задачі, оформлення ТЗ	15.03.2025 р.	
3.	Розробка моделі компонента	20.03.2025 р.	
4.	Розробка структур даних	25.03.2025 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.03.2025 р.	
6.	Програмування алгоритмів	10.04.2025 р.	
7.	Оформлення ПЗ	17.04.2025 р.	
8.	Попередній захист роботи	23.05.2025 р.	

Дата видачі завдання
« 17 » січня 2025 р.

Підпис керівника
_____ Дрєєв О.М.
(прізвище та ініціали)

Завдання прийнято до виконання
« 17 » січня 2025 р.

Підпис здобувача
_____ Брагінець С.О.
(прізвище та ініціали)

АНОТАЦІЯ

Брагінець С.О. Програмне забезпечення системи автоматизації сонячно-вітрової електростанції. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2025.

В даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи автоматизації сонячно-вітрової електростанції.

Метою розробки є програмне забезпечення системи автоматизації сонячно-вітрової електростанції.

Результат роботи – програмна реалізація системи автоматизації сонячно-вітрової електростанції.

В процесі роботи над програмною моделлю виконано аналіз існуючих апаратних та програмних засобів. В повній мірі описані всі компоненти розробленого програмного забезпечення.

Розроблено зручний інтерфейс користувача. Наведені інструкції по роботі з програмними засобами.

Програма може використовуватися на ПЕОМ з ОС Windows 10/11.

Програму розроблено в середовищі Arduino IDE C++.

Ключові слова: комп'ютерна інженерія, IoT, сонячно-вітрова електростанція

ABSTRACT

Brahinets S.O. Software for solar and wind power plant automation system. 123 Computer Engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2025.

In this final qualification work for the first (bachelor's) level of higher education, software has been developed that is intended for solar and wind power plant automation system.

The purpose of the development is the software for solar and wind power plant automation system.

The result of the work is the software implementation of the solar and wind power plant automation system

In the process of working on the software model, an analysis of existing hardware and software was performed. All components of the developed software are fully described.

A convenient user interface has been developed. Instructions for working with software are provided.

The program can be used on a PC with OS Windows 10/11.

The program was developed in the Arduino IDE C++.

Keywords: computer engineering, IoT, smart home, solar and wind power plant

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

ESP	–	Espressif
АСУ	–	автоматизована система управління
HTML	–	HyperText Markup Language
CSS	–	Cascading Style Sheets
JS	–	JavaScript
SPIFFS	–	Serial Peripheral Interface Flash File System
HTTP	–	HyperText Transfer Protocol
IDE	–	Integrated Development Environment
MQTT	–	Message Queuing Telemetry Transport
SDK	–	Software Development Kit
RTOS	–	Real-Time Operating System
ККД	–	Коефіцієнт корисної дії
RAM	–	Random Access Memory
ROM	–	Read-Only Memory
UART	–	Universal Asynchronous Receiver/Transmitter
I2C	–	Inter-Integrated Circuit
USB	–	Universal Serial Bus
IoT	–	Internet of Things
QoS	–	Quality of Service
EMQX	–	Erlang MQTT Broker X
SSL	–	Secure Sockets Layer
TLS	–	Transport Layer Security
OLED	–	Organic Light Emitting Dislay

ВСТУП

Актуальність теми. У сучасних умовах зростаючої потреби в автономних, екологічно чистих джерелах енергії, мобільні сонячно-вітрові електростанції набувають особливої актуальності. Сонячні панелі існують вже майже 80 років, а вітрогенератори вже більше 130 років. Найперша проблема, яку вирішують, або намагаються зменшити, ці види вироблення електроенергії – це зменшити викиди CO_2. За даними Міжнародного агентства з відновлюваної енергії (IRENA), у 2024 році глобальна потужність відновлюваних джерел енергії зросла на рекордні 585 гігават, що становить 92,5% від загального приросту енергетичних потужностей у світі [11].

Із збільшенням поширеності мобільного обладнання, яке потребує джерело енергії, стали більш затребувані мобільні види вироблення електроенергії. Мобільні електростанції, що поєднують сонячну та вітрову енергію, дозволяють забезпечити стабільне енергопостачання в різних умовах. Особливо важливою є автоматизація таких систем, яка забезпечує ефективне управління генерацією, накопиченням та споживанням енергії, адаптацією до змін навколишнього середовища та мінімізацію людського втручання.

Такі системи поєднують переваги відновлюваної енергії з мобільністю, забезпечуючи можливість швидкого розгортання й енергозабезпечення там, де традиційна інфраструктура відсутня або недоступна

Мета й завдання дослідження. Метою роботи є програмне забезпечення системи автоматизації сонячно-вітрової електростанції.

Для досягнення поставленої мети визначена програма дослідження, що складається з наступних завдань:

– Огляд існуючих системи автоматизації сонячно-вітрової електростанції.

– Дослідження системи автоматизації сонячно-вітрової електростанції.

– Програмна реалізація системи автоматизації сонячно-вітрової електростанції.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі автоматизації сонячно-вітрової електростанції.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи автоматизації сонячно-вітрової електростанції, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

У сучасних умовах спостерігається стрімке зростання інтересу до використання відновлюваних джерел енергії, зокрема сонячної та вітрової. Ефективність їхнього застосування значною мірою визначається тим, наскільки раціонально організовані процеси генерації, накопичення та розподілу електроенергії. Впровадження автоматизованих рішень у ці процеси сприяє підвищенню енергетичної ефективності, зменшенню втрат і забезпеченню стабільної роботи систем, що водночас підвищує економічну доцільність та знижує вплив на довкілля.

Світовий ринок мобільних електростанцій демонструє впевнене зростання. У 2024 році його вартість становила приблизно 603 мільйони доларів США, а до 2032 року очікується збільшення до понад 1,09 мільярда доларів. Це відповідає середньорічному темпу зростання (CAGR) на рівні 7,53% [12]. Особливо динамічно розвивається сектор портативних вітрових турбін — його обсяг у 2024 році досяг 640 мільйонів доларів, а до 2034 року прогнозується зростання до 4,5 мільярда доларів зі щорічним приростом у 21,44% [13].

Одним із важливих елементів підвищення ефективності використання сонячної енергії є система слідкування за Сонцем (solar tracker), яка автоматично змінює кут нахилу панелей протягом доби. Завдяки такому механізму можна збільшити кількість виробленої електроенергії на 20–40% у порівнянні зі стаціонарними установками. У поєднанні з вітровими турбінами та інтелектуальними системами керування енергоспоживанням та накопиченням електроенергії формується цілісна й ефективна модель роботи гібридної енергосистеми на базі відновлюваних джерел.

Ці технології вироблення та передачі електроенергії не нові, вони вже

давно існують. Особливий інтерес викликають пересувні енергетичні комплекси, які можна використовувати в польових умовах, для забезпечення живлення тимчасових об'єктів, експедицій, військових підрозділів або аварійно-рятувальних служб. Така система повинна мати високу енергоефективність, автономність і надійність у різних кліматичних умовах.

Головною метою цієї роботи є створення мобільної сонячно-вітрової електростанції, обладнаної системою автоматичного слідкування за Сонцем (solar tracker), яка поєднувала б у собі простоту конструкції, зручність у використанні, ефективність роботи, компактність, мобільність і доступність з точки зору вартості виготовлення.

Ключові завдання, що вирішуються в рамках роботи:

– автоматизація управління сонячними панелями з використанням відстеження Сонця для максимальної генерації електроенергії;

– розробка ефективної системи накопичення та розподілу енергії, що дозволяє зменшити втрати при передачі;

– інтеграція сонячних та вітрових джерел енергії в єдину гібридну систему з оптимальним розподілом навантажень;

– впровадження алгоритмів управління, що забезпечують стабільну та економічну роботу системи за різних умов;

– аналіз конструктивних варіантів побудови станції з метою визначення найбільш ефективного та практичного рішення.

Актуальність цієї теми зумовлена глобальною потребою у підвищенні ефективності використання відновлюваних джерел енергії, зниженні залежності від традиційних видів палива, а також прагненням до сталого розвитку й екологічно чистих технологій.

Розроблена система автоматизації забезпечує контроль за всіма ключовими процесами, пов'язаними з виробництвом, накопиченням і споживанням електроенергії, отриманої від сонячних панелей і вітрових турбін. Її основне завдання – досягнення максимальної ефективності генерації з

одночасним зменшенням втрат та забезпеченням стабільної роботи системи.

Серед основних функцій автоматизованої системи:

– сонячне стеження (solar tracking) – динамічна зміна кута нахилу сонячних панелей відповідно до положення Сонця, що дозволяє підвищити вироблення електроенергії до 40% у порівнянні зі стаціонарними установками;

– об’єднання з вітровими генераторами – гнучке керування обома джерелами енергії для максимальної продуктивності в умовах змінної погоди;

– акумуляція надлишкової енергії – збереження електроенергії у батареях або інших накопичувачах для подальшого використання;

– автоматизоване управління енергоспоживанням – розподіл енергії між споживачами, балансування навантаження і підтримання стабільного живлення;

– моніторинг і дистанційна діагностика – контроль технічного стану системи, збирання даних та можливість керування через віддалені інтерфейси.

1.2 Область застосування

Розроблена система автоматизації мобільної сонячно-вітрової електростанції призначена для застосування в умовах, де відсутнє або нестабільне централізоване енергопостачання, а також у ситуаціях, коли необхідне швидке розгортання автономного енергетичного комплексу.

Сфери де може застосовуватися або вже застосовується ця система автоматичної передачі електроенергії:

а) туризм та кемпінг:

– забезпечення електроенергією туристичних кемпінгів, наметових містечок, караванів, кемперів та автодомів;

– живлення освітлення, кухонного обладнання, холодильників, заряджання гаджетів і ноутбуків;

– організація комфортних умов для довготривалих подорожей без необхідності використовувати бензинові або дизельні генератори;

б) екстремальні експедиції та наукові дослідження:

– використання в полярних експедиціях, пустельних регіонах, джунглях, гірських районах та інших важкодоступних місцях;

– забезпечення живлення для метеорологічних станцій, геологічних лабораторій, обладнання для вимірювання рівня радіації, аналізаторів повітря, води та ґрунту;

– автономні дослідницькі бази в Арктиці, Антарктиці, океанах і безлюдних регіонах;

в) військові операції та польові штаби:

– постачання електроенергії для мобільних військових баз, зв’язкових центрів, польових госпіталів;

– живлення для радіолокаційних станцій, дронів, систем спостереження та безпеки;

– автономні системи підтримки життєдіяльності в місцях ведення бойових дій;

г) надзвичайні операції та рятувальні операції:

– забезпечення електроенергією тимчасових таборів для постраждалих у випадках землетрусів, повеней, ураганів, пожеж;

– живлення для медичних приладів, обладнання швидкої допомоги, систем зв’язку в умовах ліквідації наслідків катастроф;

– автономні системи енергозабезпечення для гуманітарних місій у зонах природних і техногенних катастроф;

ґ) сільське господарство та фермерство:

– постачання енергії для насосних станцій, зрошувальних систем, систем автоматизації теплиць та тваринницьких комплексів;

– автономне енергозабезпечення ферм у віддалених районах, де немає доступу до електромережі;

д) морські та річкові судна:

– живлення автономних яхт, рибальських суден, річкових барж та

плавучих платформ;

– альтернативне джерело енергії для навігаційних систем, освітлення, радіозв’язку;

– використання в морських дослідницьких місіях для живлення океанографічного обладнання;

е) побутове використання в умовах нестабільного електропостачання:

– забезпечення енергією приватних будинків у віддалених районах;

– використання як резервного джерела живлення під час відключень електроенергії;

– живлення побутової техніки та електроніки без необхідності використання дизельних генераторів;

є) індустріальні та комерційні об’єкти:

– використання на будівельних майданчиках для живлення електроінструментів та освітлення;

– постачання енергії для мобільних майстерень та пересувних офісів;

– забезпечення електропостачання для рекламних білбордів, автономних банкоматів, комерційних кіосків.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи автоматизації мобільної сонячно-вітрової електростанції, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти

Мобільні енергетичні системи, що поєднують сонячні панелі, вітрові турбіни та систему відстеження Сонця, активно розвиваються завдяки зростаючому попиту на автономні та екологічні джерела електроенергії. Нижче розглянемо існуючі технології, архітектури та програмні рішення, які застосовуються в таких системах.

Основним джерелом енергії у мобільних станціях є фотовольтаїчні (PV) панелі. Для вітрогенераторів зазвичай використовують середньошвидкісні або малошвидкісні турбіни з вертикальною або горизонтальною віссю. Вертикальні вітрогенератори – працюють навіть за слабкого вітру, компактні, безшумні. Горизонтальні вітрогенератори – мають вищу ефективність, але потребують сильного вітру.

Системи автоматичного стеження за сонцем дозволяє збільшити ефективність сонячних панелей на 20-40%. Є одновісні (Single-Axis) – змінюють нахил панелі від сходу до заходу, а також двовісні (Dual-Axis) – стежать за положенням сонця протягом дня та змінюють кут нахилу залежно від сезону.

Є всього дві архітектури – центарлізована та децентралізована. В централізованій архітектурі сонячні панелі та вітрогенератори підключені до єдиного гібридного інвертора, вся енергія подається у загальний акумуляторний блок та контролюється централізованим мікропроцесором. В децентралізованій – кожен елемент має власний контролер та акумулятор. Децентралізована має вищу стійкість до відмов та легше масштабується, але і складніше налаштовується.

Мобільні гібридні системи оснащуються програмним забезпеченням, що

забезпечує: моніторинг у реальному часі (напруга, заряд батарей, продуктивність), прогнозування погоди для оптимального використання енергії (також прогнозування можливих загроз), автоматичне керування solar tracker для підвищення ефективності. А також більшість сучасних систем підтримують дистанційний моніторинг через мобільний додаток або веб-інтерфейс.

Як уже згадувалося, технології вироблення енергії з сонячних та вітрових джерел — не нові. Також не є новинкою і застосування систем, що відстежують рух Сонця (solar tracker). Проте кожне рішення має свої особливості, тому варто коротко зупинитися на прикладах відомих компаній, розглянувши переваги та недоліки їхніх продуктів:

а) Smartflower: одна з найбільш нестандартних розробок — це система, що зовні нагадує квітку. Вона розкривається зранку та автоматично обертається за Сонцем, що дає змогу максимально ефективно збирати сонячну енергію. Така конструкція доволі компактна, її можна швидко встановити в різних умовах.

Плюси: слідкування за сонцем відбувається автоматично, що дозволяє отримувати більше енергії; пристрій не вимагає складного монтажу — достатньо встановити й підключити; система самостійно очищає панелі від пилу, що теж покращує ефективність; оригінальний дизайн.

Мінуси: у системі відсутній вітрогенератор, тож її ефективність залежить від наявності сонячного світла; ціна значно вища за звичайні панелі; має обмежену потужність – до 2,5 кВт, що не підійде для великого споживання; транспортування може бути складним через розміри та механіку [14].

Рисунок 2.1 – Розробка компанії Sunflower

б) Uprise Energy: американський стартап, що спеціалізується на розробці портативних вітрових турбін нового покоління. Їхній головний продукт – мобільна турбіна, що автоматично розгортається з контейнера і не потребує фундаменту.

Плюси: повністю автономна, розміщується в контейнері. Працює при низькій швидкості вітру. Автоматичне розгортання. Не потребує фундаменту або складного монтажу.

Мінуси: не має інтегрованого solar tracker. Основний акцент на вітрову генерацію. Не забезпечує баланс між вітром і Сонцем без зовнішніх доповнень [15].

Рисунок 2.2 – Розробка мобільного вітрогенератора

в) Atlas Copco ZSC: відомий шведський виробник індустріального обладнання (повітряні компресори, генератори). Їхній напрямок ZenergiZe Solar Container – це рішення для автономної енергетики, зокрема мобільні сонячні станції з батарейним блоком.

Плюси: компактна мобільна конструкція, зручна для транспортування. Можливість поєднання з іншими джерелами (дизель, батареї).

Мінуси: немає вбудованого вітрогенератора, відсутній solar tracker – кут нахилу фіксований. Дорожчий за звичайні фіксовані системи [16].

Рисунок 2.3 – «ZenergiZe Solar Container»

г) BIGLUX: китайська компанія, що спеціалізується на мобільних джерелах живлення для телекомунікацій, камер відеоспостереження, тимчасових баз. Пропонують гібридні установки на основі сонячних панелей і вітряків.

Плюси: поєднує сонячні панелі і малі вітрогенератори, підходять для відеоспостереження, телекомунікацій, кемпінгів.

Мінуси: solar tracker та портативність потрібно замовляти та виготовляти окремо. Обмежена потужність. Якість комплектуючих – середня.

ґ) ua KNESS: один з лідерів українського ринку відновлюваної енергетики. Компанія проєктує та виготовляє рішення для сонячної генерації, включаючи мобільні комплекси. Має власні інженерні та виробничі потужності у Вінниці [17].

Плюси: власне виробництво, адаптоване до українських умов. Є приклади автономного живлення польових лікарень. Можлива інтеграція з вітром у проєктних рішеннях.

Мінуси: немає стандартної моделі із solar tracker. Рішення не серійні, а

індивідуально зібрані. Доступність обмежена виробничими обсягами.

Як видно більшість виробників пропонують окремі компоненти, а не готові мобільні енергосистеми, або навість інтегровані рішення мають громіздку конструкцію або складне транспортування. Або немає дуальності в генеративній системі: сонячні панелі малоефективні в похмуру погоду, а вітрогенератори залежать від сили вітру, якщо поєднати, тоді ймовірність вироблення енергії при різних погодних умовах збільшується. А також, більшість рішень не є «plug-and-play» (не можна підключити та одразу використовувати без складного налаштування) і вимагають професійного встановлення та технічного обслуговування. Немає універсальної мобільної сонячно-вітрової станції, яка була б водночас ефективною, недорогою, легко транспортувалася та мінімально залежала від погодних умов.

Отже, з вищенаведених переліків переваг та недоліків існуючих впроваджених систем генерації сонячно-вітрової енергії, можна побачити доцільність проектування своєї подібної системи, яка робить акцент на поєднання одночасного використання Сонця та вітру, мобільність, компактність, зручність та ефективність використання.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Вибір мови програмування, що найбільш підходить для ESP32

C++ – це мова низького рівня, що дають програмісту максимальний контроль над апаратними ресурсами ESP32. Це дозволяє створювати ефективний код з мінімальними витратами пам'яті. Можна використовувати спеціальні функції для управління пінами, використання переривань, реалізації алгоритмів відстеження та керування енергією.

Якщо порівнювати з Python, C++ має більш складний синтаксис і потребує детальнішого розуміння апаратних засобів;

MicroPython – це високоабстрагована мова, яка має простий синтаксис і добре підходить для швидкої розробки та тестування прототипів. MicroPython це спеціальна версія Python, оптимізована для мікроконтролерів, включаючи ESP32. Вона має більшу частину стандартної бібліотеки Python і підтримує основні функції ESP32, такі як Wi-Fi, Bluetooth, робота з GPIO пинами, таймерами та ін. Python дозволяє швидко розробляти код, тестувати та адаптувати його. Це корисно на етапі прототипування системи або для управління простими завданнями (наприклад, моніторинг стану батарей чи керування датчиками). MicroPython має велику спільноту, яка активно розвиває бібліотеки для різних датчиків та периферії.

Отже, MicroPython є хорошим вибором для швидкого прототипування або для простих проектів, де важлива зручність кодування і не так важлива висока продуктивність, але C++ є найкращим вибором для розробки на ESP32, оскільки ця мова забезпечує високий рівень контролю над апаратними ресурсами, дозволяє оптимізувати програму за швидкістю та пам'яттю, а також є стандартною для роботи з мікроконтролерами через Arduino IDE.

Так як проект передбачає **створення веб-інтерфейсу** для моніторингу або управління, HTML, CSS та JavaScript – це три основні технології, які використовуються разом для створення веб-інтерфейсу на ESP32. HTML визначає структуру веб-сторінки – це її “скелет”. У HTML описуються елементи, такі як заголовки, абзаци, кнопки, форми, таблиці, зображення, а також блоки контенту, які можна оновлювати чи стилізувати. У контексті ESP32 HTML часто зберігається як текст у пам’яті пристрою або в його файловій системі (SPIFFS або LittleFS), і надсилається браузеру при HTTP-запиті, наприклад, коли користувач заходить на сторінку IP-адреси ESP32.

CSS забезпечує оформлення HTML-елементів, тобто задає їхній колір, розмір, шрифти, відступи, вирівнювання, розташування в просторі та анімацію. Це дозволяє зробити інтерфейс привабливим і зручним для користувача. CSS може бути вбудованим прямо в HTML-файл у вигляді <style> блоку, або

винесеним у окремий файл, який також можна зберігати в файловій системі ESP32 і завантажувати окремо.

JavaScript – це мова програмування, яка виконується безпосередньо в браузері користувача й дозволяє додавати інтерактивність. З його допомогою можна реагувати на натискання кнопок, змінювати вміст сторінки без її перезавантаження, надсилати та отримувати дані з ESP32 через HTTP-запити, щоб відправити дані про напругу з ESP32 або включити реле. JavaScript може виконувати опитування сервера для оновлення даних, наприклад, для графіка потужності чи моніторингу стану.

Вибір середовища розробки

Arduino IDE – це популярне середовище розробки, яке часто використовують для програмування ESP32 завдяки його простоті, доступності й великій спільноті користувачів. Воно базується на C/C++ і дає змогу писати код у спрощеному вигляді, приховуючи багато складних деталей. Для ESP32 Arduino IDE підтримує підключення плати, встановлення потрібних бібліотек і прошивку мікроконтролера напряму через USB. Це робить старт надзвичайно швидким навіть для новачків.

Основною перевагою Arduino IDE для ESP32 є саме простота. Сотні бібліотек вже адаптовані для ESP32, зокрема для Wi-Fi, SPIFFS, датчиків, сервоприводів, дисплеїв, MQTT й багатьох інших. Більшість прикладів коду легко знайти, скопіювати та змінити під свої потреби. Крім того, Arduino IDE дозволяє працювати з асинхронним вебсервером, створювати Wi-Fi точки доступу, працювати з клієнтами, запускати вебінтерфейс тощо. Завдяки цьому можна створити повноцінний проєкт із вебінтерфейсом прямо в Arduino, не вдаючись до складної конфігурації або командного рядка.

Недоліки полягають у меншій гнучкості та обмеженому контролі над низькорівневими аспектами ESP32, особливо порівняно з офіційним SDK – ESP-IDF. Наприклад, налаштування енергоефективності, точна робота з FreeRTOS, роздільне керування ядрами, профілювання пам’яті чи використання апаратного

криптомодуля можуть бути недоступними або ускладненими. До того ж, Arduino IDE не має повноцінного автодоповнення коду, інтеграції з відлагодженням чи розширеної підтримки CMake-проєктів.

Visual Studio Code (VS Code) – це потужний і безкоштовний редактор коду, який дуже часто використовують для розробки вебінтерфейсів, зокрема для створення HTML, CSS та JavaScript-файлів, які потім використовуються у вебсервері ESP32. Його головна перевага — це висока швидкість роботи, багатофункціональність і величезна кількість розширень, які роблять розробку комфортною та ефективною.

Для HTML, CSS і JavaScript VS Code надає інтелектуальне автодоповнення, підсвічування синтаксису, перевірку помилок у режимі реального часу, підтримку Emmet (швидке написання HTML/CSS через скорочення) та зручну структурування коду. Завдяки інтеграції з live-server або Live Preview можна миттєво переглядати вебсторінку в браузері без збереження вручну чи перезавантаження. Це дуже зручно для перевірки вигляду та поведінки HTML/CSS/JS перед завантаженням в ESP32.

Ще одна важлива перевага — розширення (extensions): для CSS і JS можна встановити додаткові лінтери (наприклад, ESLint або Stylelint), підтримку препроцесорів (Sass, Less), автогенерацію коду, перевірку сумісності зі старими браузерами тощо. Якщо потрібно, можна легко додати інтеграцію з Git, файловими системами ESP32 (через esp32fs плагіни), або платформами на кшталт PlatformIO.

Початківцю інтерфейс VS Code може видатися перевантаженим: тут багато панелей, команд і розширень, які спочатку відволікають. Також для повноцінного використання JavaScript функцій VS Code не дає всіх можливостей IDE рівня WebStorm (від JetBrains), наприклад, складного рефакторингу чи автоматичної перевірки типів (якщо не використовувати TypeScript). Крім того, VS Code не є повноцінним браузером, тому потрібно використовувати сторонні засоби попереднього перегляду або запускати локальний сервер.

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи автоматизації сонячно-вітрової електростанції.

В процесі розробки випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти необхідно виконати наступний обсяг роботи:

а) провести аналіз існуючих систем-аналогів для виявлення їх позитивних і негативних якостей. Результати аналізу врахувати в подальших розробках;

б) вибрати та обґрунтувати методику побудови системи контролю роботи технологічного обладнання на виробництві в автоматизованому режимі. Розробити функціональну та структурну схеми системи;

в) розробити програмне забезпечення системи, що дозволить реалізувати поставлену технічним завданням задачу. Побудувати блок-схеми алгоритмів програми та підпрограми;

г) організувати інтерфейс користувача з метою формування та виводу на екран ЕОМ повідомлень про некоректні дії користувача та нестандартні ситуації в роботі технологічного обладнання;

д) розробити рекомендації по організаційних та методичних заходах, які забезпечать впровадження системи в промислову експлуатацію та її подальшу успішну експлуатацію;

е) провести розрахунки по визначенню економічної ефективності розробленої системи;

ж) розробити заходи по охороні праці при впровадженні та експлуатації системи, а також розробити заходи з цивільного захисту;

з) сформувати висновки про виконаний обсяг робіт.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Система проектування має на меті зберегти первинні функції електрогенераторів за допомогою Сонця та вітру, а також показати дешевший варіант розробки з акцентом на компактності та мобільності, тому внутрішня будова буде простою і економною. Звісно це має свої наслідки, зменшення: потужності, ефективності та втрати в швидкості зарядки. Далі ми розглянемо основні засоби, які є основою для побудови системі.

Апаратні засоби:

а) сонячні панелі:

– монокристалічні: найвища ефективність серед мікропанелей (до 20-22%), краща робота в умовах слабкого освітлення, дорожчі порівняно з іншим;

– полікристалічні: дешевші, але менш ефективні (до 16-18%), менш чутливі до температури (менше нагріваються);

– аморфні (тонкоплівкові): гнучкі, легкі, можуть бути вбудовані у тканину або пластик, але ефективність 8-12%, але і добре працюють при розсіяному світлі;

б) вітрогенератор;

Таблиця 3.1 – Порівняння горизонтальних та вертикальних вітрогенераторів

Параметр	Горизонтальний	Вертикальний
Ефективність	Вища (ККД)	Нижча ефективність
Напрямок вітру	Потрібен стабільний напрямок вітру	Може працювати при будь-якому напрямку вітру
Механізм орієнтації	Так	Ні

Продовження таблиці 3.1

Стабільність роботи	Вища при сильних вітрах	Стабільно працює при слабких вітрах
Потужність	Вища, для великих систем	Мала та середня потужність
Вартість	Дорожчі	Дешевші
Шум	Гучніші	Тихіший
Вибір лопатей	Довгі горизонтальні профілі	Лопаті можуть бути круглими, арковими або спіральними
Нагрівання	Сильніше нагрівається	Менше нагрівається

в) аккумулятор;

Таблиця 3.2 – Порівняння видів акумуляторів

Характеристики	**Свинцево-кислотний**	**Літій-залізо-фосфатний**	**Літій-іонний**
ККД (%)	70-85	95	90-95
Глибина розряду (%)	50	90-100	80-90
Кількість циклів заряду-розряду	300-1000	2000-5000	500-1500
Саморозряд (в місяць, %)	3-5	<1	1-2
Температурний діапазон (°C)	-20…+50	-20…+60	0…+45
Чутливість до глибокого розряду	Висока	Низька	Висока
Маса (відносно ємності)	Важкі	Легші за свинцеві	Найлегші
Вартість	Дешеві	Дуже дорогі	Дорогі

Продовження таблиці 3.2

Обслуговування	Так	Ні	Ні

г) контролер заряду;

Таблиця 3.3 – Порівняння котролерів заряду

Характеристика	**PWM-контролер**	**MPPT-контролер**	**Простий контролер**
Принцип роботи	Широтно-імпульсна модуляція (просто знижує напругу)	Максимальне відстеження потужності (відбирає оптимальну напругу)	Лінійний заряд без оптимізації
ККД зарядки (%)	70-80	95-99	60-80
Сумісність з вітрогенератором	Обмежений	Так	Так (але для малопотужних)
Сумісність з сонячними панелями	Так для малопотужних	Так для будь-якої потужності	Тільки для малих панелей (1-5 Вт)
Напруга акумуляторів (В)	12, 24	12, 24, 48	3,7, 5
Максимальний струм (А)	10-30	10-60	2
Вартість	Дешевий	Дорогий	Дуже дешевий
Захист акумуляторів	Базовий	Інтелектуальний	Мінімальний
Контроль заряду для Li-ion	Без BMS	Потрібен BMS	Є, але тільки для окремих елементів

ґ) тип системи підключення до мережі:

– автономна система: не підключається до загальної електромережі, а працює повністю незалежно. Вона особливо корисна для мобільного застосування. Повна незалежність – не залежить від електромережі. Мобільність – можна транспортувати та розгортати в будь-якому місці. Робота навіть при аваріях в мережі – зберігає електропостачання у разі відключення зовнішньої електроенергії;

– гібридна система: ця система працює як від сонячно-вітрових джерел, так і від зовнішньої електромережі. Вона може віддавати надлишкову електроенергію в мережу або ж використовувати її як резервне джерело. Гнучкість у використанні – можна чергувати живлення від мережі та акумуляторів. Зменшення витрат на електроенергію – у сонячні/вітряні дні можна мінімізувати споживання з мережі. Можливість продажу електроенергії – у деяких країнах надлишкову енергію можна повертати в мережу (Net Metering). Резервне живлення при аваріях – якщо мережа зникне, система продовжить працювати;

д) датчики освітленості;

Таблиця 3.4 – Порівняння видів датчиків освітленості

Тип датчика	**Принцип роботи**	**Діапазон чутливості (lx)**	**Точність**	**Вихідний сигнал**
Фотодіод	Генерує струм	0,1-100000	Висока	Аналоговий (мА, В)
Фоторезистор (LDR)	Опір змінюється	1-100000	Середня	Аналоговий (Ом)
Фототранзистор	Посилений фотодіод	0,1-50000	Висока	Цифровий або аналоговий
TSL2561	Цифровий	0,01-88000	Висока	I^2C

Продовження таблиці 3.4

BH1750	Цифровий	1-65535	Висока	I^2C
VEML7700	Високочутливий цифровий	0,003-120000	Дуже висока	I^2C

е) привід для solar tracker;

Таблиця 3.5 – Порівняння приводів

Тип приводу	**Споживання енергії (W)**	**Швидкість переміщення (градуси/с)**	**Точність**	**Вартість**
Мотор-редуктор	10	Швидка (2-5)	Середня	Середня
Серводвигун	3-5	Низька (0,1-1)	Висока	Середня
П’єзоелектричний	0,2	Дуже висока	Дуже висока	Дуже дорога

є) мультиплікатор для вітрогенератора;

Таблиця 3.6 – Порівняння видів мультиплікатора

Тип	**ККД (%)**	**Передаточне число**	**Швидкість обертання (об/хв)**	**Навантаження (Нм)**
Циліндричний	95-98	3:1 – 10:1	Висока	Середнє
Конічний	96-98	5:1 – 15:1	Висока	Середнє
Планетарний	90-96	3:1 – 100:1	Середня	Високе
Черв’ячний	50-90	10:1 – 100:1	Низька	Високе
Ремінний	80-97	2:1 – 10:1	Висока	Низьке

ж) мікроконтролер.

Таблиця 3.7 – Порівняння підходящих МК

МК	**Тактова частота**	**RAM**	**Wi-Fi/ Bluetooth**	**Аналогові входи**	**Споживання**
Arduino Mega 2560	16 МГц	8 Кб	Немає	Є (16)	Низьке
ESP32	240 МГц (2 ядра)	520 Кб	Вбудований	Є (18)	Дуже низьке
Raspberry Pi 4	1,5 ГГц (4 ядра)	2-8 Гб	Вбудований	Немає	Високе
STM32F103	72 МГц	20 Кб	Немає	Є (10)	Низьке

Характеристики обраних варіантів:

а) мікрокотролер ESP32 WROOM-32 [18]:

– процесор: 32-бітний Xtensa LX6, 2 ядра;

– тактова частота: 240 МГц;

– RAM: 520 кБ;

– ROM: 448 кБ;

– Flash: 16 мБ;

– Wi-Fi: 802.11 b/g/n (2,4 ГГц);

– GPIO: 36 пінів;

– ADC: 12-біт, 18-каналів, 3,3 В;

– DAC: 8-біт, 2 канали;

– PWM: 16 каналів;

– периферія: UART, I2C, SPI;

– живлення: 3,3 В;

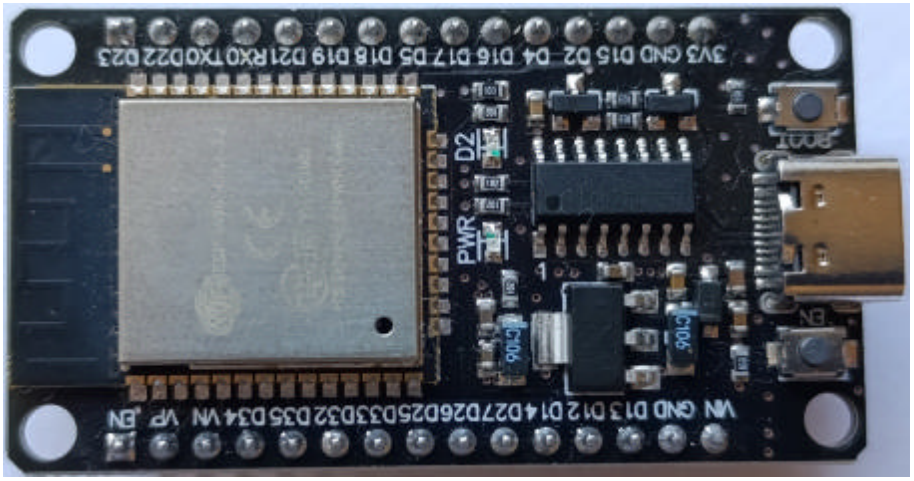

Рисунок 3.1 – ESP32

б) акумулятор 18650 (4 шт.) з холдером [19]:

– літій-іонний, 2500 мАг, 3,6 В, розмір: 18650;

– стандартна зарядка: 1,25 А (0,5 С), 4,2 В, 0,1 А cut-off;

– максимальна зарядка: 6 А (2,4 С), 4,2 В, 0,1 А cut-off;

– стандартна розрядка: 0,5 А (0,2 С), 2,5 В cut-off;

– максимальна розрядка: 20 А (12 С), 2,5 В cut-off;

Рисунок 3.2 – Акумулятори 18650, 4 штуки, в холдері

в) контролер заряду TP4056 [20]:

– максимальний струм зарядку 1,2 А;

– діапазон живлення 4-8 В;

– споживання: 55-500 мкА;

– якщо виявлено глибокий розряд <2,9 В, тоді струм заряджання буде 1/10 від основного;

– можливість налаштування струму заряджання батареї;

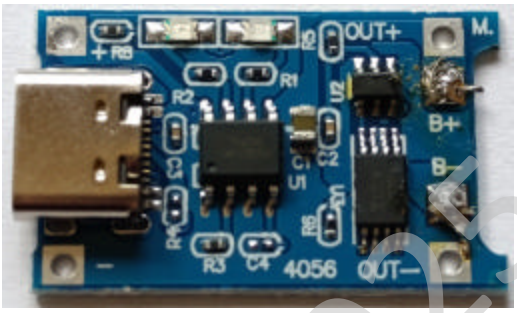

Рисунок 3.3 – TP4056

г) BMS 1S [21]:

– напруга від перезаряду: 4.25 В;

– напруга при якій знову дозволяється заряджати: 4.23 В;

– напруга від перерозряду: 2.54 В;

– максимальний струм заряджання/розряджання: 2 А;

– захист від струму: 3А;

Рисунок 3.4 – BMS 1S

г) XL6009E1 DC-DC step-up (boost) [22]:

– потужність: 20 Вт;

– вхідна напруга: 3,5 – 32 В;

– вихідна напруга: 5 – 35 В;

– максимальний струм на виході: 2 А;

– струм спокою: 5 мА;

– ефективність: 92%;

– розміри: 43x20,6 мм;

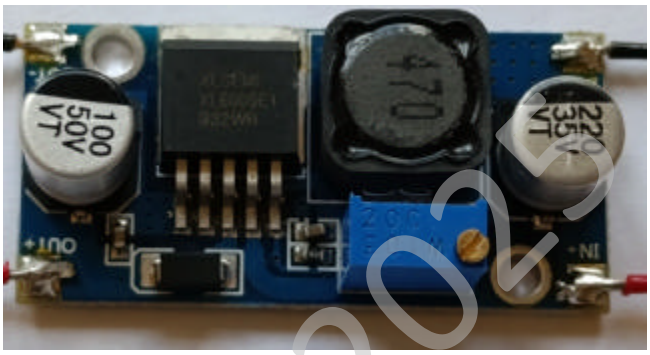

Рисунок 3.5 – XL6009E1

д) TPS63020 DC-DC step-down/up (buck-boost) [23]:

– потужність: 10 Вт;

– вхідна напруга: 1,8 – 5 В;

– вихідна напруга: 5 В;

– максимальний струм на виході: 3 А;

– струм спокою: 6 мА;

– ефективність: 96%;

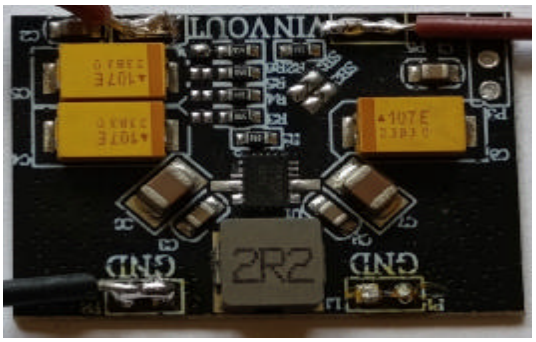

Рисунок 3.6 – TPS63020

е) Mini-560 DC-DC step-down (buck) [24]:

– вхідна напруга: 5 – 20 В;

– вихідна напруга: 5 В;

– максимальний струм на виході: 4 А;

– струм спокою: 6 мА;

– ефективність: 96%;

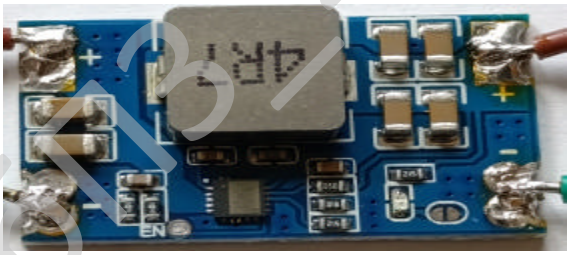

Рисунок 3.7 – Mini-560

є) JX-887Y Powerbank модуль [25]:

– вбудована система BMS 1S;

– вхідна напруга USB-A, 2 шт.: 5В/1А, 5В/2А;

– вихідна напруга micro-USB: 5 Вт;

– 4-ьох рівневий світлодіодний індикатор для відображення рівня заряду акумуляторів;

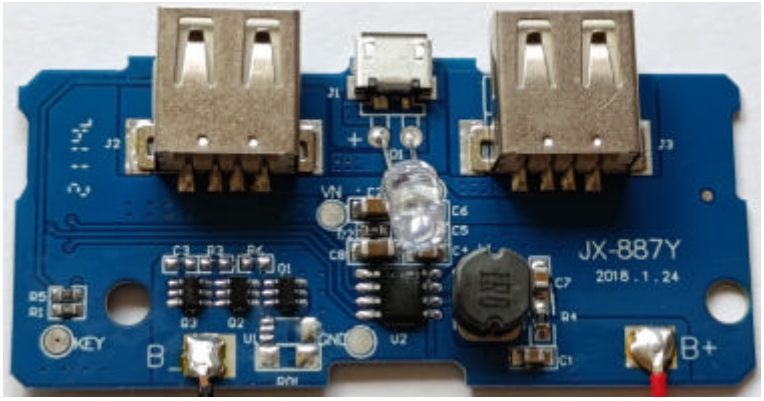

Рисунок 3.8 – Powerbank модуль

ж) сонячна панель (4 шт.): полікристалічна, 0,75 Вт, 6 В, $I_{кз}$=135 мА, $U_{хх}$=7,2 В, розмір: 110х55 мм [26];

Рисунок 3.9 – Сонячна панель

и) вертикальний вітрогенератор;

і) гібридна система підключення;

ї) датчик освітленості – фоторезистор (LDR)

й) привід для системи слідування за Сонцем (два серводвигуна для двох осей), Tower Pro SG90 (горизонтальна вісь) [27]:

– крутий момент: 1,2-1.4 кг/см;

– живлення: 4,8-7,2 В;

– пусковий струм: 650-800 мА;

– швидкість: 0,1с/60°;

– вага: 9 гр;

– розмір: 22,2х11,8х31 мм;

Рисунок 3.10 – Сервомотор

к) циліндричний мультиплікатор з DC-моторчиком [28]:

– 2000 об/хв;

– передаточне число: 1:10;

– вихідна напруга: 3 – 9 В;

Рисунок 3.11 – Моторчик з мультиплікатором

л) ASC712 модуль вимірювання напруги [29]:

– напруга живлення: 4.5 – 5 В;

– споживання струму: 13 мА;

– напруга навантаження: 5 А;

– чутливість до зміни струму: 190 мВ/А;

– нульова точка (точка порівняння): 2.5 В;

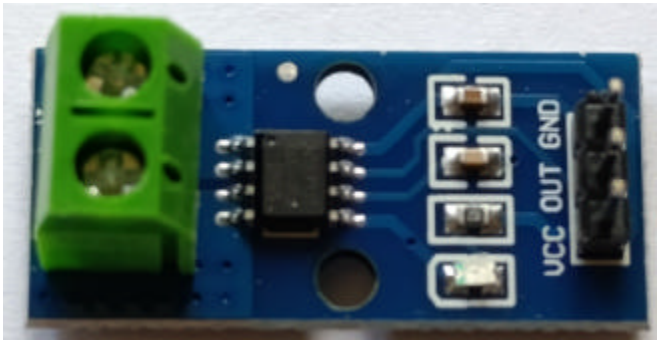

Рисунок 3.12 – ASC712 5A

м) OLED-дисплей [30]:

– розмір: 128x32 пк.;

– максимальна напруга живлення: 3,5 В;

Рисунок 3.13 – OLED-дисплей

н) транзистор 2N7000 (польовий):

– максимальний струм стоку (струм навантаження): 350 мА;

– напруга затвор-витік: 3 В;

о) випрямний міст GBJ2510, максимальний струм 3,5 А;

п) електролітичний конденсатор:

– ємність: 1000 мкФ;

– напруга: 16 В;

р) діод Шотткі BAT43:

– максимальна зворотня напруга: 30 В;

– максимальний зворотній струм 500 мА;

– падіння напруги: 0,3 В;

с) кнопка тумблер;

т) резистор 2,7 кОм 2 шт., 10 кОм 2 шт.

Мережеві засоби контролю

MQTT (Message Queuing Telemetry Transport) – це протокол обміну повідомленнями, який широко використовується для передачі даних між пристроями в Інтернеті речей (IoT). Це легкий, простий і ефективний протокол, який особливо підходить для мобільних, віддалених або ресурсозбережних пристроїв.

MQTT споживає мінімум ресурсів, що робить його ідеальним для малопотужних пристроїв, таких як сенсори, мікроконтролери чи інші IoT-устаткування. Його маленький заголовок (всього 2 байти) дозволяє зменшити навантаження на мережу та зберігати енергію пристроїв.

MQTT працює за принципом публікації та підписки. Publish (публікація) – пристрій відправляє повідомлення (дані) на певний топік (ім'я каналу або теми). Топіки в MQTT мають ієрархічну структуру, яка виглядає як рядок, поділений на рівні за допомогою слешів (/). Топіки можна організовувати по-різному, в залежності від того, яку інформацію ви хочете передавати або отримувати. Вони можуть бути простими (наприклад, temperature) або складними, з кількома рівнями (наприклад, home/room1/temperature). Subscribe (підписка) – інші пристрої підписуються на певний топік і отримують ці повідомлення, коли вони надходять. Це дозволяє створювати ефективну і масштабовану архітектуру для обміну даними між пристроями.

MQTT призначений для швидкої та надійної доставки повідомлень навіть при обмеженій пропускній здатності мережі та високих затримках.

MQTT підтримує три рівні якості обслуговування (QoS):

а) QoS 0 – повідомлення надсилається один раз, без підтвердження;

б) QoS 1 – повідомлення надсилається принаймні один раз, з підтвердженням;

в) QoS 2 – гарантована доставка повідомлення тільки один раз, з додатковими механізмами для уникнення дублювання.

MQTT дозволяє використовувати "збереження останнього повідомлення" на сервері брокера, щоб новий підписник отримав останню відправлену інформацію навіть після того, як підключиться до системи.

MQTT брокер – це сервер, який здійснює розподіл повідомлень між пристроями, перевіряє підключення та підтримує топіки.

Брокер може бути публічним або приватним (для безпеки і конфіденційності).

MQTT є ідеальним протоколом для обміну даними в мобільних сонячно-вітрових енергетичних системах завдяки його легкості, ефективності та здатності підтримувати реальний час з мінімальними ресурсами. Вибір MQTT дозволяє забезпечити надійне та масштабоване рішення для контролю, моніторингу і керування такими системами на основі відновлюваних джерел енергії.

EMQX – це високопродуктивний MQTT-брокер з відкритим кодом, написаний на мові Erlang/OTP. Він реалізує протокол MQTT (Message Queuing Telemetry Transport) версій 3.1, 3.1.1 та 5.0 і призначений для обробки великої кількості одночасних підключень клієнтів (до мільйонів), а також передачі повідомлень з низькою затримкою. EMQX побудовано з урахуванням масштабованості, відмовостійкості та підтримки різних механізмів автентифікації, авторизації та інтеграцій з іншими системами (наприклад, базами даних, HTTP, LDAP, JWT, Redis, PostgreSQL, MongoDB тощо). Основна функція EMQX – це посередництво між видавцями (publishers) і підписниками

(subscribers), що спілкуються через топіки (тематики) за принципом “publish/subscribe”. Повідомлення можуть бути доставлені з різними рівнями якості (QoS 0, 1, 2) залежно від вимог до надійності доставки.

EMQX має сучасну архітектуру, що дозволяє запускати брокер у кластерному режимі – як на одному сервері, так і розподілено між кількома вузлами. Завдяки використанню Erlang воно забезпечує гарну конкуренцію, балансування навантаження та автоматичне відновлення у разі збоїв. Також EMQX підтримує з'єднання через TCP, SSL/TLS, WebSocket і WebSocket Secure (wss), а в платній версії – також QUIC і MQTT over QUIC. Окрім базового MQTT, EMQX підтримує розширення за допомогою плагінів та вебхуків (Webhooks), а також має RESTful API для адміністрування, збору статистики, керування підключеннями й перегляду журналів. EMQX забезпечує безпеку завдяки TLS-шифруванню, автентифікації на основі сертифікатів, ACL (access control lists), інтеграції з OAuth 2.0, LDAP тощо.

Крім open-source версії, існує EMQX Enterprise Edition з додатковими можливостями для корпоративного використання — такими як моніторинг у режимі реального часу, зберігання повідомлень у базах даних, кластеризація через WAN, підтримка гібридних розгортань у хмарах, трасування повідомлень, інтеграції з Apache Kafka, ClickHouse, InfluxDB, Prometheus та іншими системами обробки великих даних. Для спрощення розгортання EMQX підтримує Docker і Kubernetes, а також надає офіційний Helm Chart. Платформа активно розвивається, має документацію, відкритий код на GitHub, спільноту та комерційну підтримку. Вона ідеально підходить для застосунків в IoT, розумних містах, енергетиці, автомобільній промисловості та будь-яких системах, що вимагають надійного обміну повідомленнями між великою кількістю пристроїв.

EMQX Cloud пропонує безкоштовний **тарифний план Serverless**, який ідеально підходить для розробників, невеликих проєктів та тестових середовищ. Цей план дозволяє користуватися MQTT-брокером безкоштовно, за умови дотримання встановлених лімітів. Щомісяця безкоштовна квота включає:

– 1 мільйон хвилин сесій: це еквівалентно приблизно 23 пристроям, які залишаються підключеними протягом усього місяця;

– 1 ГБ трафіку: включає як вхідний, так і вихідний трафік MQTT-повідомлень;

– 1 мільйон дій правил (rule actions): використовується для обробки даних, таких як зберігання, перетворення або пересилання повідомлень.

Обмеження та технічні характеристики тарифного плану:

– користувач може створити до двох окремих розгортань;

– максимальна кількість одночасних з’єднань (пристроїв): 1000;

– максимальний розмір повідомлення: 1 МБ;

– максимальна кількість підписок на клієнта: 10;

– швидкість публікації для одного клієнта: 10 повідомлень за секунду;

– підтримка TLS 1.2 та 1.3.

MQTTX – це сучасний кросплатформенний клієнт для протоколу MQTT (Message Queuing Telemetry Transport), який розроблено компанією EMQ (провідним провайдером MQTT-рішень). Його основна мета — забезпечити зручний інтерфейс для взаємодії з MQTT-брокерами: підключення, підписка на топіки, публікація повідомлень, а також перегляд і аналіз отриманих даних. MQTTX підтримує роботу з останніми версіями MQTT (включаючи MQTT 5.0), і завдяки сучасному дизайну та інтуїтивно зрозумілому інтерфейсу він став популярним інструментом серед розробників IoT-систем, особливо під час тестування і налагодження MQTT-з'єднань.

Однією з головних переваг MQTTX є те, що він дозволяє створювати одночасно кілька клієнтів (тобто симулювати декілька пристроїв), кожен з яких може мати свої параметри з'єднання, окремі підписки й повідомлення. Це дуже зручно, якщо потрібно протестувати взаємодію пристроїв або різних частин системи. Крім того, MQTTX має вбудовану підтримку TLS/SSL, що дозволяє підключатися до захищених брокерів (наприклад, через порт 8883) і використовувати сертифікати безпеки, зокрема Root CA, клієнтські сертифікати і

приватні ключі.

Інтерфейс MQTTX дає змогу легко налаштувати з'єднання: вказати адресу брокера (IP або домен), порт, клієнтський ідентифікатор (Client ID), ім’я користувача й пароль (за потреби), а також QoS (рівень якості доставки повідомлень). Після підключення можна створювати підписки на довільні топіки (включаючи з використанням шаблонів з символами «+» і «#»), і в реальному часі спостерігати за вхідними повідомленнями, які автоматично відображаються в окремому вікні.

Ще однією важливою особливістю MQTTX є можливість публікації повідомлень у форматі JSON, текстовому або бінарному форматі, з вибором QoS, утриманням повідомлення (retained), а також таймерами для автоматичної публікації через певні інтервали. Це особливо корисно при тестуванні сценаріїв періодичної передачі даних, наприклад, від датчиків.

MQTTX також підтримує імпорт і експорт конфігурацій, що дозволяє швидко ділитися налаштуваннями з колегами або переносити їх між машинами. Крім настільної версії, доступний MQTTX CLI – командний інтерфейс для роботи з MQTT, який ідеально підходить для автоматизації, написання скриптів та CI/CD-процесів.

Підсумовуючи, MQTTX – це потужний інструмент, який значно спрощує роботу з MQTT-брокерами завдяки простому GUI, підтримці сучасних стандартів безпеки, мультиклієнтності, а також можливостям публікації, підписки й аналізу повідомлень у режимі реального часу. Він особливо корисний у розробці, тестуванні та налагодженні IoT-рішень, де MQTT відіграє ключову роль у передачі телеметричних даних.

Конструкторське проектування

Так як одне із основних завдань, це створення мобільної, компактної сонячно-вітрової електростанції, то відповідно, треба щоб його було зручно переносити, монтувати та демонтувати. Одним із засобів, що реалізовують цей функціонал особистого користування – це snap-fit (засувне з’єднання).

Це тип конструкторського рішення, що забезпечує з'єднання двох або більше компонентів за допомогою пружної деформації матеріалу без потреби у додаткових кріпильних елементах, таких як гвинти, клей чи зварювання. Основний принцип полягає у використанні виступу (фіксатора), який заходить у відповідний паз або отвір, фіксуючись завдяки зусиллю, що виникає під час монтажу. Після фіксації з'єднання залишається стабільним за рахунок пружних властивостей матеріалу. Snap-fit-з'єднання широко використовуються у виробах з термопластів, зокрема в побутовій електроніці, автомобільній промисловості, корпусах пристроїв та дитячих іграшках, оскільки забезпечують швидке складання, зниження ваги виробу та зменшення витрат на виробництво. Серед переваг snap-fit слід відзначити простоту монтажу й демонтажу, можливість автоматизації складання, а також зменшення кількості деталей. Водночас такі з'єднання потребують точного розрахунку геометрії фіксуючих елементів і врахування механічних характеристик матеріалів, оскільки надмірне навантаження чи втома матеріалу можуть призвести до поломки. Основні типи snap-fit включають консольні (beam-type), кільцеві (annular) та уловлюючі (cantilever hook) з'єднання. У проєктуванні snap-fit важливу роль відіграють такі фактори, як товщина, довжина та форма фіксатора, допустимий кут згину, модуль пружності матеріалу, а також температура експлуатації. Розробка ефективного snap-fit-рішення потребує моделювання навантажень і ретельного тестування, особливо в умовах багаторазового використання або дії агресивного середовища.

3.2 Розробка структурної схеми

Структурна схема розглядає загальну архітектуру системи автоматизації сонячно-вітрової електростанції. Структурна схема дозволяє узагальнено представити взаємодію основних апаратних та програмних компонентів, необхідних для ефективного функціонування гібридної електроенергетичної установки. Основними елементами системи є сонячні панелі, вітрогенератор,

контролер заряду, акумуляторний блок, а також програмне забезпечення, що забезпечує моніторинг, збір даних та дистанційне керування. Структура побудована таким чином, щоб забезпечити максимальну автономність роботи станції з можливістю ручного або автоматизованого втручання. Центральним вузлом є мікроконтролер, який зчитує дані з фоторезисторів, рівень акумулятора, тощо та передає їх на MQTT-сервер та Web-сервер через бездротове з'єднання. Програмна частина виконує обробку отриманої інформації, генерує попередження у разі виходу параметрів за допустимі межі та надає користувачеві інтерфейс для перегляду в реальному часі. Такий підхід дозволяє ефективно інтегрувати відновлювані джерела енергії в єдину керовану систему, підвищити її надійність та оптимізувати енергоспоживання.

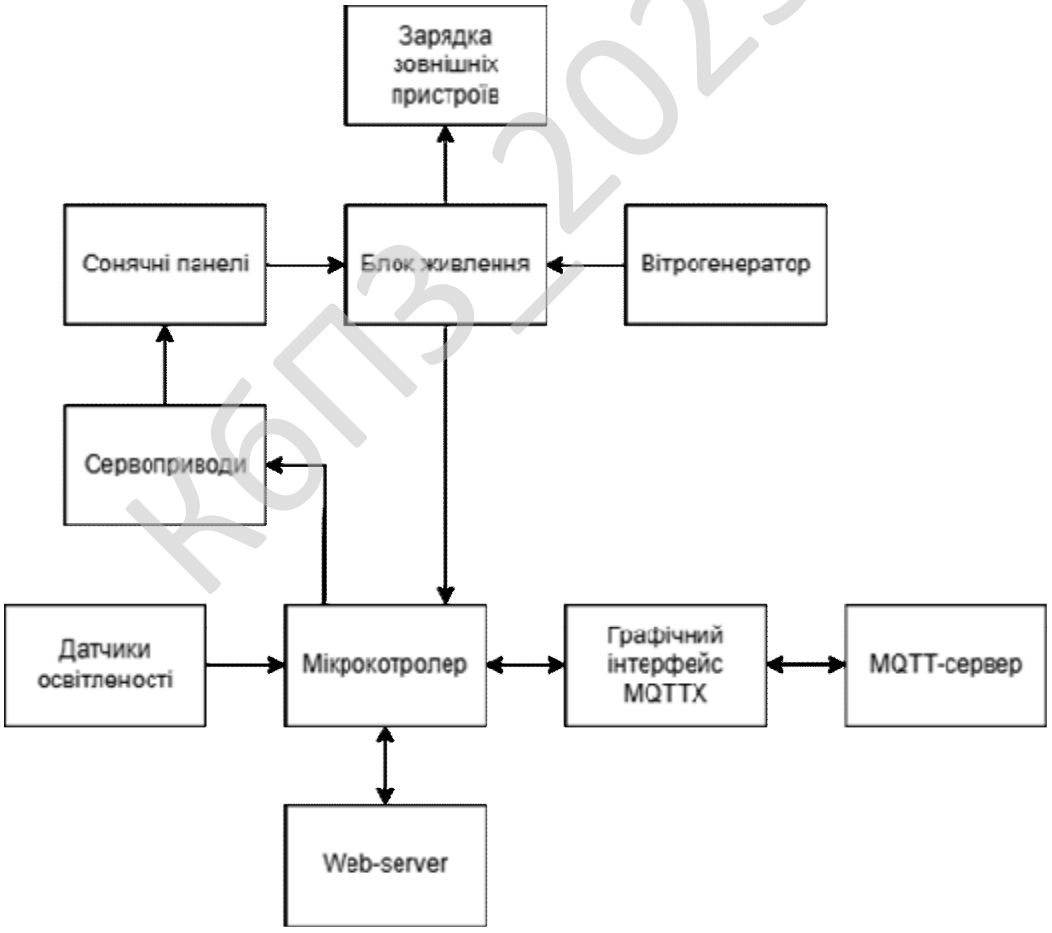

Рисунок 3.14 – Структурна схема системи

3.3 Розробка функціональної схеми

Функціональна схема описує логічну структуру взаємодії компонентів програмного забезпечення та апаратної частини системи автоматизації сонячно-вітрової електростанції. Функціональна схема відображає послідовність обробки даних, основні процеси керування та контролю, а також інформаційні потоки між підсистемами. На вхід системи надходять дані від датчиків — зокрема, інформація про рівень напруги на фоторезисторах, стан заряду акумуляторів та споживання енергії. Ці дані обробляються в реальному часі мікроконтролером, який виконує функції аналізу та прийняття рішень згідно з вбудованою логікою керування. На основі заданих алгоритмів мікроконтролер формує відповідні сигнали для перемикання між джерелами енергії, активації або деактивації зарядного пристрою, захисту акумуляторів від глибокого розряду або перенапруги. Одночасно з цим дані передаються до мережевих засобів контролю, де відбувається їхня візуалізація та надсилання сповіщень користувачу. Функціональна схема передбачає також можливість ручного керування системою через інтерфейс користувача, що дозволяє оперативно реагувати на зміну умов або аварійні ситуації.

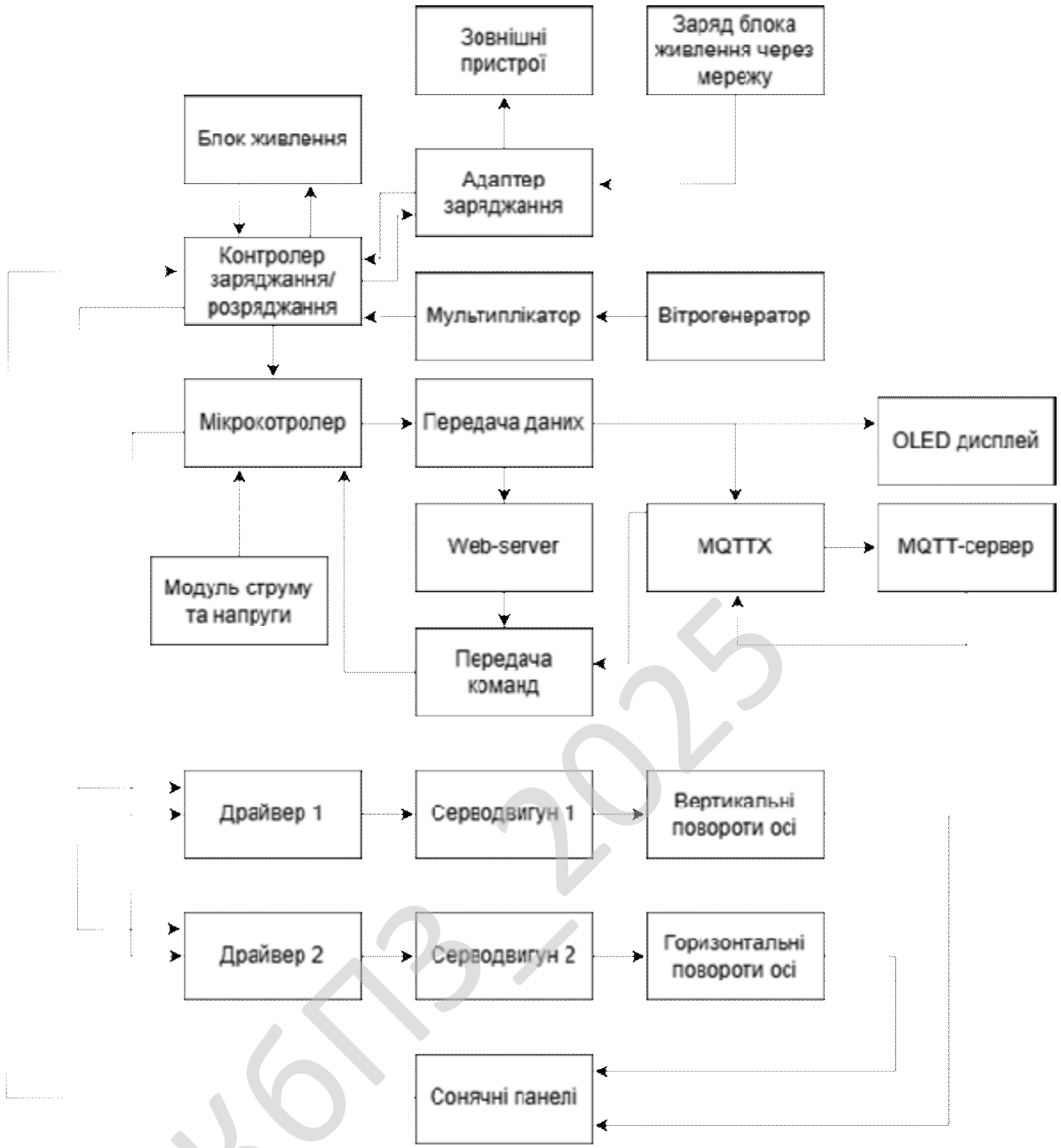

Рисунок 3.15 – Функціональна схема системи

3.4 Розробка діаграми процесів

Діаграма процесів – візуальне представлення графу процесів. Граф процесів є різновидом графу станів скінченного автомату, вершинами якого є певні дії, а переходи відбуваються по завершенню дій. Процес (дія) є фундаментальною одиницею визначення поведінки системи. Процес отримує множину вхідних

сигналів та перетворює їх на множину вихідних сигналів. Одна із цих множин, або обидві водночас, можуть бути порожніми. Кожен процес може виконуватись один, два, або більше разів під час одного запуску системи. Деякі процеси можуть вимагати певної послідовності.

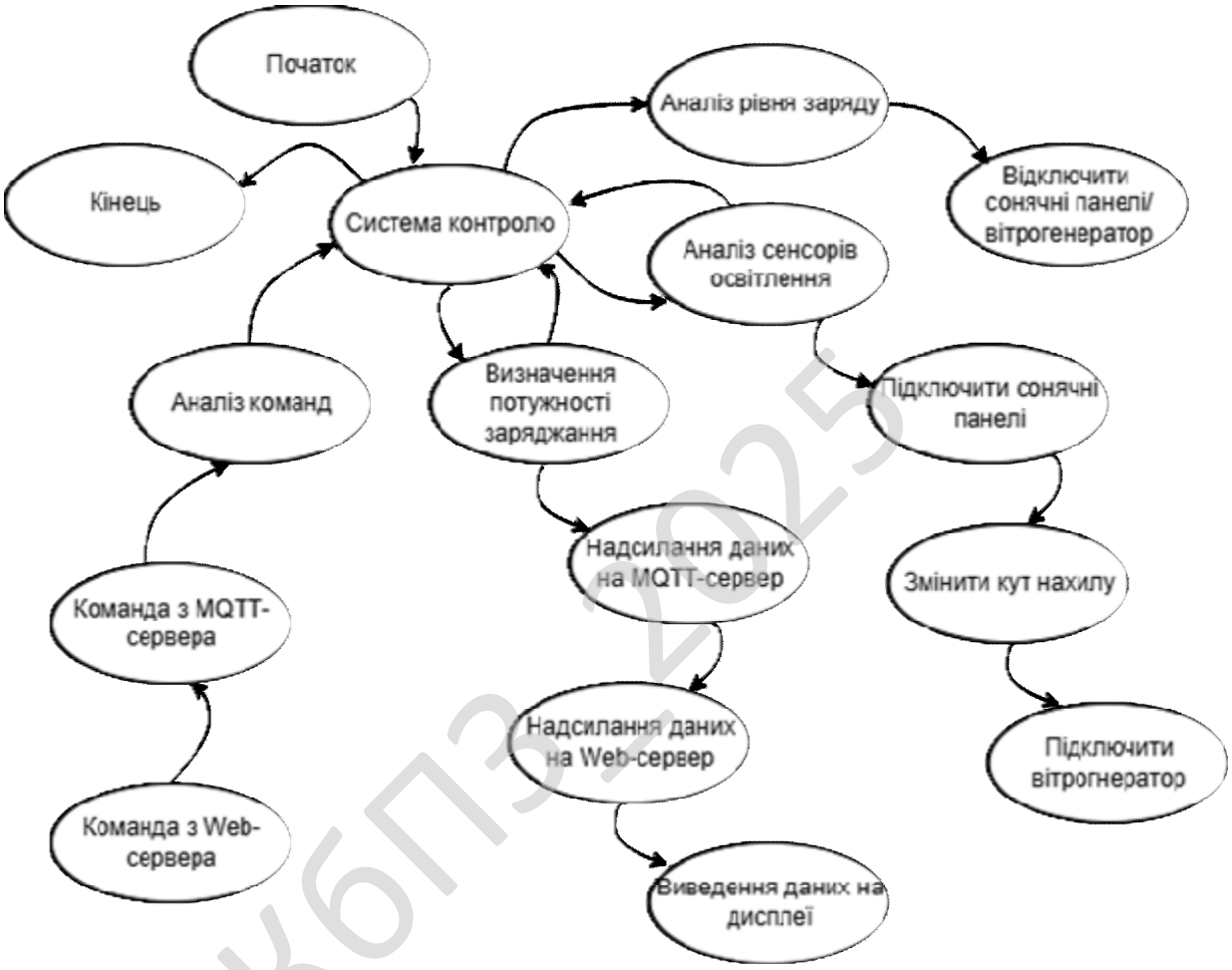

Рисунок 3.16 – Діаграма процесів

4 РЕАЛІЗАЦІЯ ПРОЕКТУ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ПРАВИЛЬНІСТЬ ПРОЕКТНИХ РІШЕНЬ

4.1 Блок-схеми та опис алгоритмів функціонування системи

На рисунку 4.1 зображена основна блок-схема програми, на рисунку 4.2 зображено роботу підпрограми.

З яких видно що робота основної програми складається з початкових етапів налаштування основних модулів функціонування системи: сервомотори, кнопки, дисплей, вимірювання струму, Інтернет з'єднання, Web-сервер та MQTT.

Нижче наведемо програмний код. Який відображає підключення основних функцій у програмі.

```
void setup() {
  Serial.begin(115200);

  // Налаштувати піни для сервомоторів
  configServoPins();

  // Налаштувати кнопку для Oled
  set_up_button();
  // Налаштувати дисплей
  set_up_display();

  // Налаштування пінів для ACS712
  configPins();

  // Одна Спроба Підключення до вайфай
  wifiConnectInit();

  // Локальна IP-адреса, якщо є підключення до Інтернету
  Serial.println(WiFi.localIP());

  // Налаштування MQTT
  configMQTT();
```

```
    // Сервер прослуїовує запити
    listtenTo();

    // Під'єднання клієнта до івентів
    eventConnect();

    // Web-сервер включити
    server.begin();
}
```

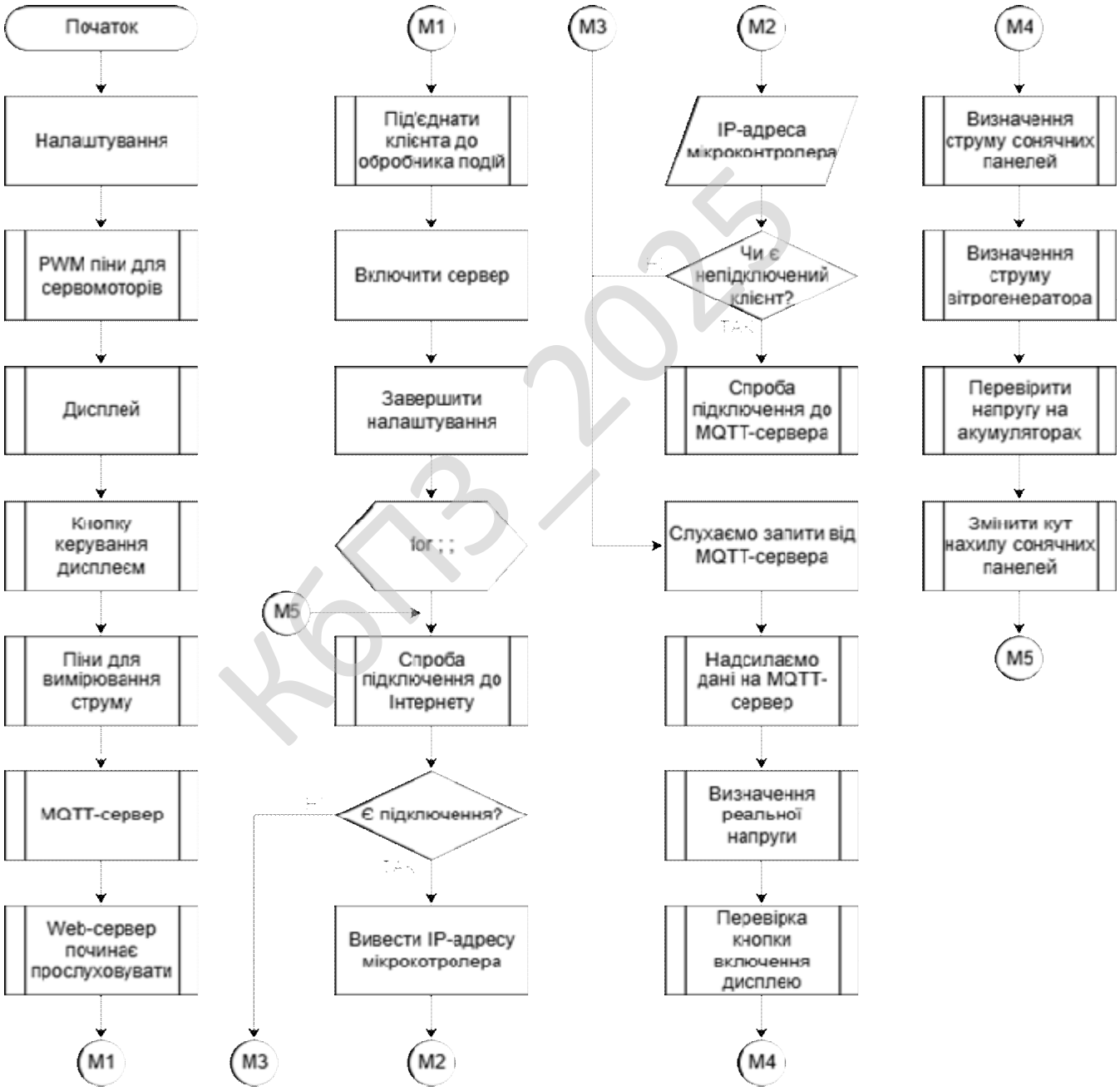

Рисунок 4.1 – Блок-схема основної програми

При роботі підпрограми виконується основний функціонал системи з циклічними послідовностями, перевіркою поточного стану напруги акумулятора.

```
// Автоматично має роздільну здатність 12-біт
adc_value = analogRead(res_div_Pin);
// Переведення АЦП в вольти
volt_value = adc_value * 3.3 / 4095;
Serial.println("Volt: " + String(volt_value, 2));

// Визначення реальної напруги на акумуляторах
real_volt = mapFloatVolt(volt_value, 2.35, 3.3, 3.0, 4.2);

// Переводимо вольти у відсотки
real_volt_percent = mapFloatPercent(volt_value, 2.36, 3.3, 0, 100);
Serial.println("Volt_Percent: " + String(real_volt_percent, 2));
```

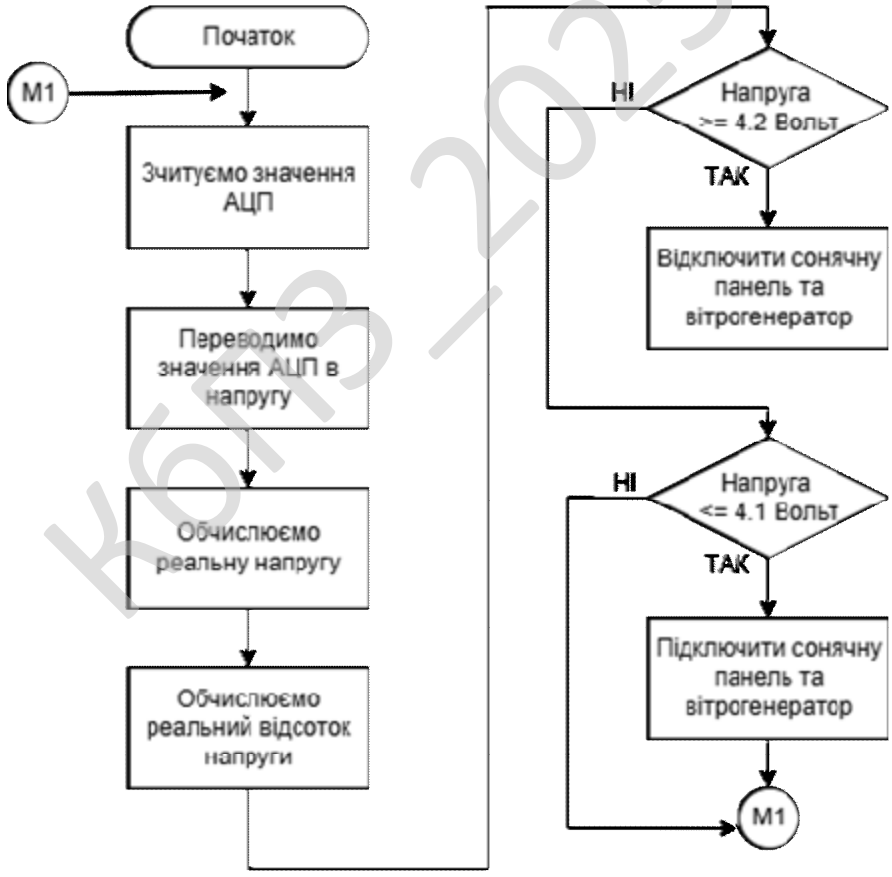

Рисунок 4.2 – Блок-схема роботи підпрограми

Далі буде розглянуто основні та важливі фізичні з’єднання, які описані на принципових схемах. У зв’язку з великим обсягом компонентів, принципову схему було поділено на три основні частини: система живлення, дисплей з системою електрогенерації та система стеження за Сонцем.

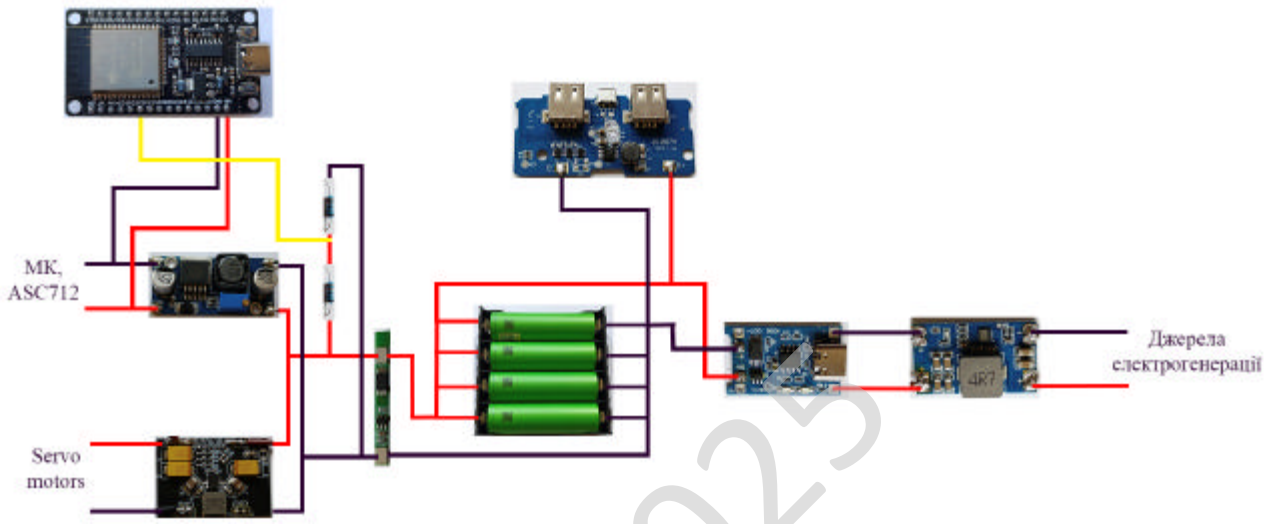

Рисунок 4.3 – Принципова схема системи живлення

Система живлення складається з двох основних частин: заряджання акумулятора та розряджання під дією навантаження. Кожна частина має свої перетворювачі напруги та контролери заряджання/розряджання.

Модуль портативного джерела живлення дає можливість заряджати зовнішні електроприлади: телефони, планшети, навушники, тощо, а також, заряджати акумулятори через зовнішню елетромережу. Він має вбудований захист від розряджання, тому додаткових компонентів не потребує.

Джерела електрогенарції проходять через понижувальний перетворювач та контролер заряду, який захищає акумулятори від перезаряджання.

Плата BMS 1S контролює розряджання акумуляторів, через неї йде живлення на підвищувальні перетворювачі напруги, які живлять модуль вимірювання струму, мікроконтролер та сервомотори.

ESP32 вимірює напругу на акумуляторах за допомогою пари резисторів,

так як є певні обмеження величини подачі напруги на вхід АЦП мікроконтролера.

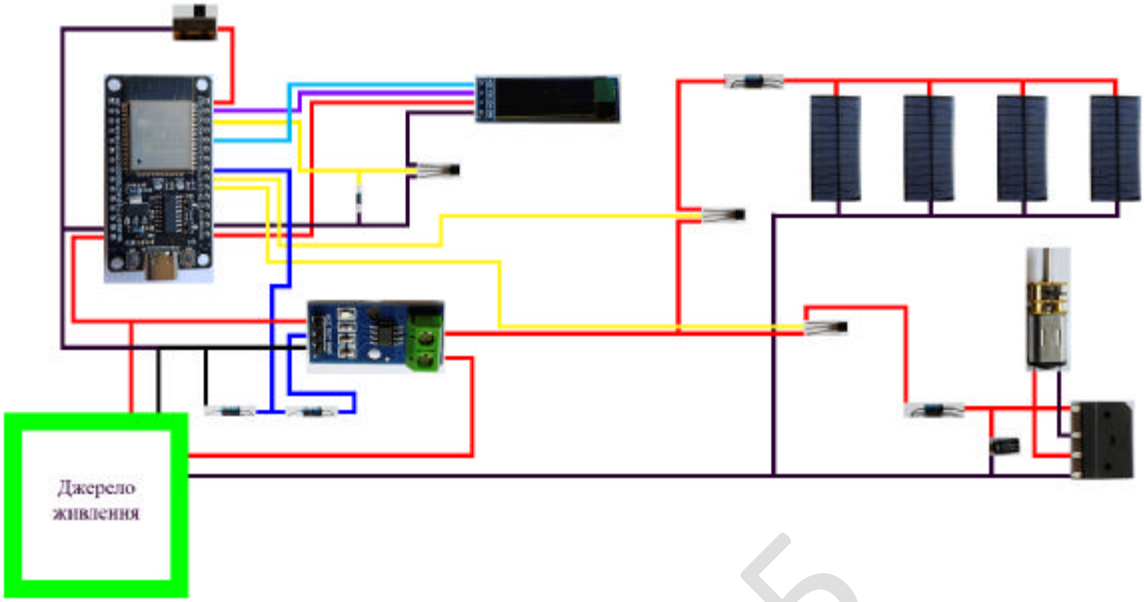

Рисунок 4.4 – Принципова схема електрогенерації та живлення дисплею

З вище наведеної схеми видно, що дисплей живиться від 3.3 вольт мікроконтролер, і що його можна контролювати за допомогою кнопки та транзистора. Мікроконтролер перевіряє потенціал на вході лапки, і якщо він низький, тоді на іншу лапку подається високий потенціал, що зумовлює відкриття затвора транзистора.

Інші транзистори, що під'єднані до сонячних панелей та вітрогенератора, відкриті постійно, так як через них йде струм на акумулятори. Вони по черзі закриваються, періодично, для того, щоб визначити струм, який вони генерують, за допомогою модуля ASC712.

Дані, у вигляді напруги, з ASC712 йдуть до мікроконтролера, який за допомогою певних обчислень перетворює їх в реальні значення струму.

Діоди Шотткі використовуються для запобігання протікання зворотного струму від сонячних панелей до вітрогенератора та навпаки.

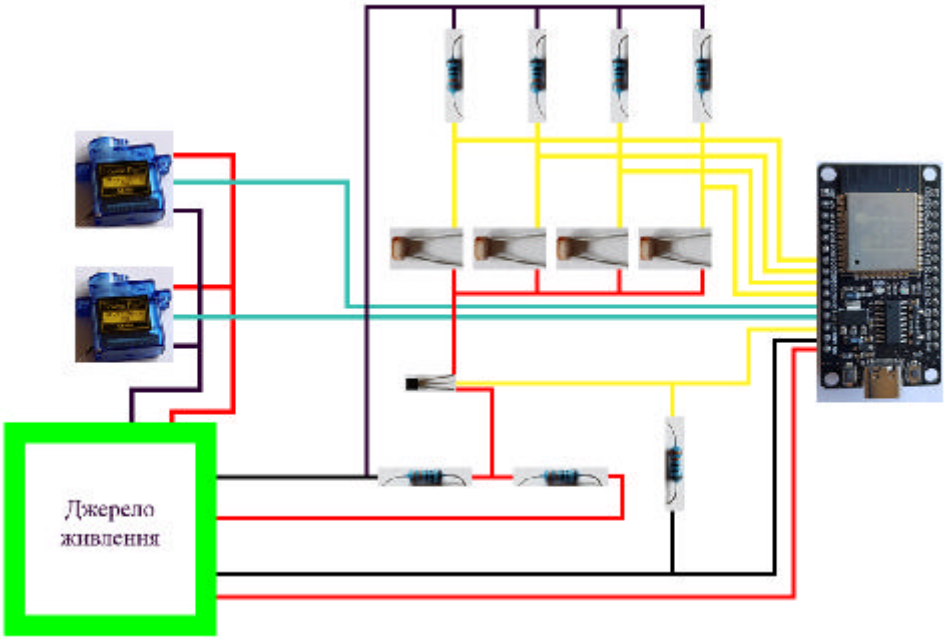

Рисунок 4.5 – Система стеження за Сонцем

Все доволі просто: транзистор, що періодично відкривається для визначення напруги на фоторезисторах. За допомогою певного алгоритму дій, визначаються деякі нерівності, що в свою чергу веде до подачі PWM до сервомоторів, які змінюють положення сонячних панелей відносно Сонця.

4.2 Захист розробленого програмного забезпечення

Захист розробленого коду на мікроконтролері від несанкціонованого доступу та плагіату є важливою складовою безпеки проєкту, особливо у випадку коли мікроконтролер використовується в комерційних або критично важливих системах.

Щоб забезпечити надійність, по-перше, використовуються апаратні можливості ESP32, такі як Flash Encryption і Secure Boot. Flash Encryption дозволяє зашифрувати прошивку у внутрішній флеш-пам’яті мікроконтролера, що унеможливлює прочитання бінарного коду ззовні. Secure Boot перевіряє цілісність підписаної прошивки перед її виконанням, запобігаючи запуску

неавторизованого або модифікованого коду. Ці функції потрібно вмикати ще на етапі прошивки через idf.py або espefuse.py, після чого пристрій працюватиме лише з підписаною і зашифрованою прошивкою.

По-друге, обмежуєтья доступ до UART та інших програмних інтерфейсів. У виробничих версіях прошивка вимикає вивід логів через UART або використовуються апаратні фільтри, щоб унеможливити підключення до дебаг-консолі. Також важливо контролювати доступ до кнопок "BOOT" і "EN", оскільки через них можна ініціювати перепрошивку.

По-третє, так як ESP32 використовується з протоколом MQTT-протоколом, то для захисту з’єднання використовується критично важливий компонент TLS/SSL, особливо через публічні мережі. TLS (Transport Layer Security) забезпечує шифрування всього трафіку між ESP32 та MQTT-брокером, що унеможливлює перехоплення, підробку чи модифікацію повідомлень третіми сторонами. Більш високий рівень захисту досягається при використанні взаємної автентифікації – коли ESP32 також має власний сертифікат клієнта (Client Certificate) і закритий ключ. У цьому випадку брокер приймає з'єднання лише від пристроїв з дійсним сертифікатом. Цей метод практично виключає можливість під'єднання неавторизованих пристроїв. Самі сертифікати (Root CA, Client Cert, Private Key) зберігатєть не у відкритому вигляді у коді, а в секціях захищеної пам’яті (наприклад, NVS або Partition Table), або у форматі PEM у прошивці, якщо використовується Flash Encryption. Що якраз підтримую EMQX-брокер.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Розглянемо як налаштувати та почати користуватися автоматизованою сонячно-вітровою електростанцією.

Почнемо з Web-сервера. Щоб в нього зайти потрібно дізнатися його IP-адресу. Для цього потрібно під’єднатися до однієї мережі. Як тільки цей крок зроблено, на OLED-дисплеї на 10 секунд висвітиться адреса, яку треба вставити у свій браузер. Вигляд сторінки, для користувача зображено на малюнку 5.1.

Рисунок 5.1 – Web-сторінка користувача

Видно, що є шість графіків, по три для сонячної панелі та для вітрогенератора, які відображають напругу, струм та потужність генерації. Також, в самому низу є кнопка регулювання дисплеєм та відображення напруги на акумуляторах в вольтах та відсотках. Якщо дисплей включився з сервера, тоді кнопка не зможе вимкнути його.

Далі розглянемо процес налаштування MQTT-серверу. Спочатку потрібно

зайти на їхній офіційний сайт, зареєструватися, обрати безкоштовний план та створити розгортання (сесію). На малюнку 5.2 буде представлено дані, які знадобляться в подальшому в розділі «MQTT Connection Information».

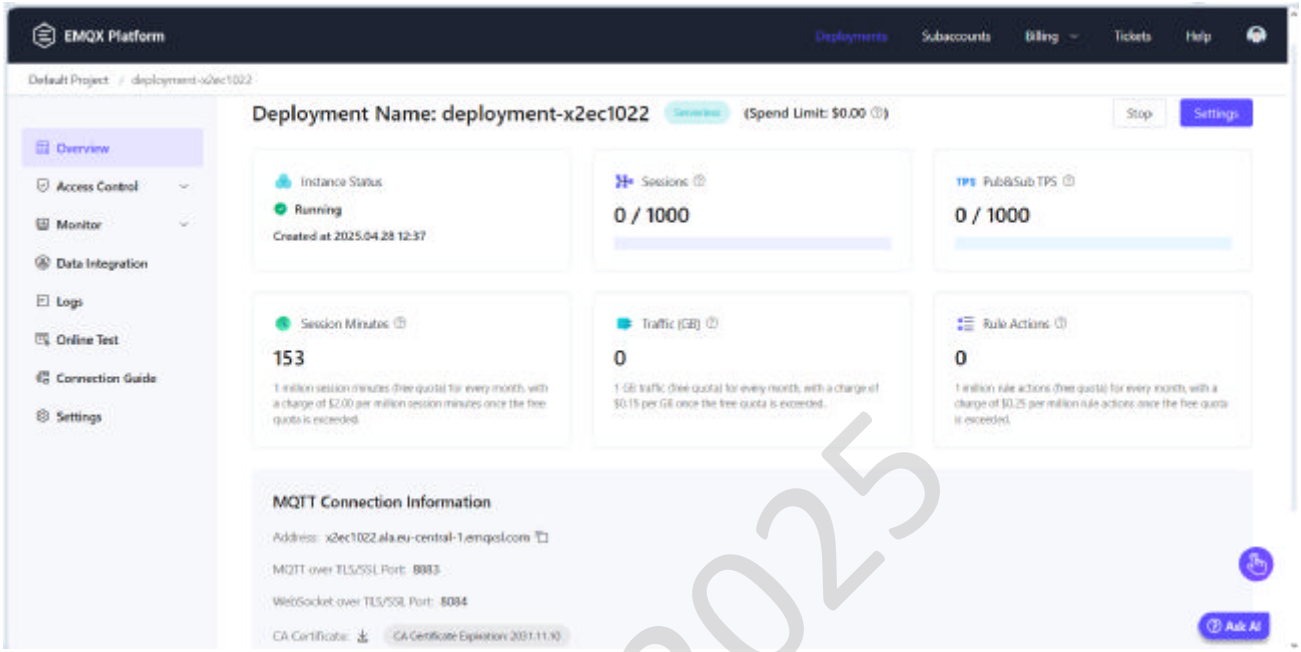

Рисунок 5.2 – Розгортання на платформі EMQX

Далі треба завантажити MQTTX, це програмне забезпечення допоможе комфортніше працювати: отримувати дані про стан компонентів та надсилати команди. Щоб створити з’єднання, потрібно вказати кілька обов’язкових параметрів: ім’я з’єднання (довільне, для ідентифікації), адресу брокера (це може бути IP-адреса або доменне ім’я, наприклад broker.hivemq.com), порт (зазвичай 1883 для незашифрованого з’єднання або 8883 для TLS), та унікальний ідентифікатор клієнта (Client ID), який відрізняє ваш MQTT-клієнт від інших. За потреби можна вказати ім’я користувача та пароль, якщо брокер вимагає авторизацію. Якщо з’єднання захищене, потрібно увімкнути TLS та, за потреби, додати сертифікати безпеки, наприклад Root CA, клієнтський сертифікат або приватний ключ.

Після успішного підключення до брокера, можна створювати підписки на певні топіки (наприклад, esp32/volts) – це дозволить отримувати повідомлення, які публікують інші пристрої чи клієнти на ці топіки. Повідомлення з'являються в MQTTX у вигляді списку вхідних даних, де зручно переглядати вміст, час отримання та QoS. Далі можна почати надсилати свої власні повідомлення – для цього достатньо натиснути "Publish", вказати топік, QoS, та тіло повідомлення (у форматі JSON або текстовому). Також можна налаштувати таймери для автоматичної публікації через певні проміжки часу.На рисуноку 5.3 буде предаставлено вигляд інтерфейсу MQTTX.

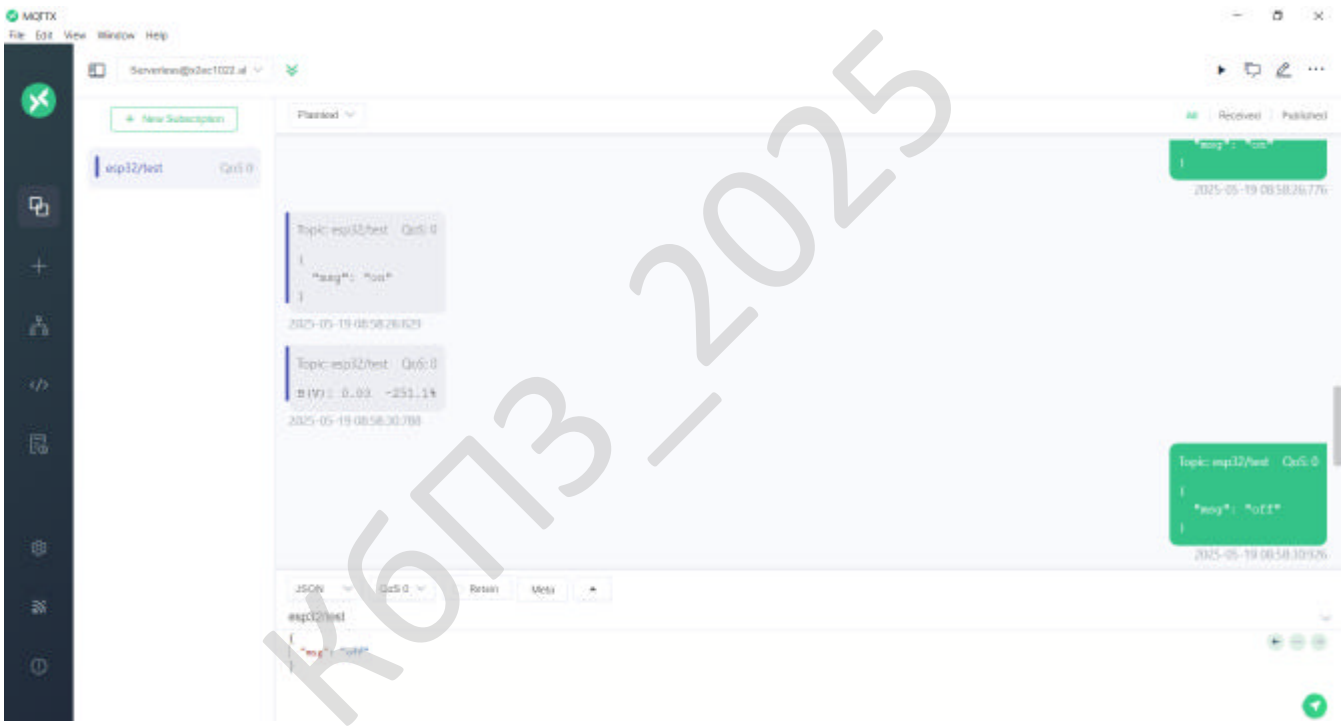

Рисунок 5.3 – Інтерфейс користувача MQTTX

В кінці, все що потрібно це отримувати регулярні дані про параметри системи, або надсилати команди включення або виключення дисплею.

Користувач нічого не платить, якщо не перевищує ліміти тарифного плану.

6 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти, призначено для системи автоматизації сонячно-вітрової електростанції.

В межах України в недостатній мірі представлені вітчизняні розробки в цій області.

Рішення завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем автоматизації сонячно-вітрової електростанції.

– Досліджена система автоматизації сонячно-вітрової електростанції

– На основі отриманих результатів досліджень створена програмна реалізація системи автоматизації сонячно-вітрової електростанції.

Розроблені під час виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання автоматизації сонячно-вітрової елктростанції.

Обґрунтовано вибір елементів системи: фотогальванічні панелі, вітровий генератор, акумуляторні батареї, контролер заряду, та мікроконтролер для керування та моніторингу. А також використання мережевих систем контролю.

Розроблене програмне забезпечення має простий, дружній та зручний інтерфейс користувача, що забезпечує легкість у освоєнні роботи програмного продукту, зручність у використанні, і не потребує особливих спеціальних знань.

При створені програмного забезпечення було використано модульний підхід, що відповідає сучасним тенденціям у галузі розробки комерційних програмних систем.

Програма реалізована на мові високого рівня C++. Дана мова програмування дозволяє найбільш ефективно обробляти дані призначені для системи автоматизації сонячно-вітрової елктростанції. Це дозволило мінімізувати

строк розробки програмного забезпечення, і, як слід, зменшити витрати на його розробку. Запропоноване програмне забезпечення ділиться на внутрішнє, яке реалізує всю логіку керування та зовнішньо-допоміжне, яке дозволяє відслідковувати основні показники системи.

Запропонована мобільна малопотужна сонячно-вітрова електростанція із відстеженням Сонця є перспективним та ефективним рішенням для автономного енергопостачання. Вона забезпечує зменшення залежності від традиційних джерел енергії, екологічність та мобільність, що робить її важливим кроком у розвитку технологій автономного енергозабезпечення.

В цілому створене програмне забезпечення підтверджує правильність використаних проектних рішень та повністю відповідає вимогам технічного завдання. Створене програмне забезпечення має потенційну можливість для подальшого вдосконалення і застосування у різних галузях.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Зелена Енергія [Електронний ресурс]. – Режим доступу: https://events.pstu.edu/konkurs-energy/wp-content/uploads/sites/2/2018/03/Зелена-Енергія.pdf

2. ЕНЕРГОЕФЕКИВНІСТЬ ТА ЕНЕРГОЗБЕРЖЕННЯ: ЕКОНОМІЧНИЙ, ТЕХНІКО-ТЕХНОЛОГІЧНИЙ ТА ЕКОЛОГІЧНИЙ АСПЕКТИ [Електронний ресурс]. – Режим доступу: https://essuir.sumdu.edu.ua/bitstream-download/123456789/87012/1/Melnyk_Zelena_energetika.pdf

3. ВІДНОВЛЮВАЛЬНІ ДЖЕРЕЛА ЕНЕРГІЇ [Електронний ресурс]. – Режим доступу: https://www.ive.org.ua/wp-content/uploads/Monografia_final_21.12.2020.pdf

4. Solar Power System: What You Should Know [Електронний ресурс]. – Режим доступу: https://solar.huawei.com/nl/blog/nl/2024/solar-power-system

5. Wind explained: Types of wind turbines [Електронний ресурс]. – Режим доступу: https://www.eia.gov/energyexplained/wind/types-of-wind-turbines.php

6. ЩО КРАЩЕ ВІТРОГЕНЕРАТОР ЧИ СОНЯЧНА БАТЕРЯ [Електронний ресурс]. – Режим доступу: https://www.solargarden.com.ua/shho-krashhe-vitrogenerator-chy-sonyachnaya-batareya/

7. СПІЛЬНЕ ВИКОРИСТАННЯ ВІТРОГЕНЕРАТОРІВ ТА ФОТОЕЛЕКТРИЧНИХ ЕЛЕМЕНТІВ [Електронний ресурс]. – Режим доступу: http://www.tsatu.edu.ua/tstt/wp-content/uploads/sites/6/barsukova_2-22.pdf

8. Сотник І. М. Енергоефективність та відновлювальна енергетика в Україні: проблеми управління.: Університетська книга, 2023. – 247с.

9. Віктор Малишев, Андрій Поліщук, Ангеліна Габ, Дмитро Шахнін. Альтернативна енергетика.: Університет «Україна», 2020. – 60 с.

10. VipMart О. Сонячна електроенергія для початківців. Частина 1. – VipMart 2023. – 25с.

11. Renewables in 2024: 5 Key Facts Behind a Record-Breaking Year [Електронний ресурс]. – https://www.irena.org/News/articles/2025/Apr/Renewables-in-2024-5-Key-Facts-Behind-a-Record-Breaking-Year

12. Portable Power Station Market [Електронний ресурс]. – https://www.fortunebusinessinsights.com/portable-power-station-market-105508

13. Global Portable Wind Turbine Market Overview [Електронний ресурс]. – https://www.fortunebusinessinsights.com/portable-power-station-market-105508

14. SmartFlower [Електронний ресурс]. – https://smartflower.com/products/

15. Mobile Power Station [Електронний ресурс]. – https://upriseenergy.com/?utm_sourc

16. Mobile solar container range [Електронний ресурс]. – https://www.atlascopco.com/en-ua/construction-equipment/products/mobile-solar-power/mobile-solar-container-range?utm_source

17. Renewables for Ukraine [Електронний ресурс]. – https://www.ekoenergy.org/renewables-for-ukraine/?utm_source

18. ESP32 Series [Електронний ресурс]. – https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

19. Lithium Ion Cell 18650 2500mAh Battery Datasheet [Електронний ресурс]. – https://www.mouser.com/datasheet/2/855/ASR00050_18650_2500mAh-3078640.pdf?srsltid=AfmBOoq6mFWpGc1CXtFuEo-BQamGeO61Hbnnskormjqj3RDlPuAKe6Zn

20. TP4056 1A Standalone Linear Battery Charger with Thermal Regulation in SOP-8 [Електронний ресурс]. – https://grobotronics.com/images/companies/1/datasheets/TP4056%20Datasheet.pdf?1530184261538=&srsltid=AfmBOopLkRHTguZrqbc8hD2L2VX7rUfaSQC85C

21. BMS 1S [Електронний ресурс]. – https://www.rcscomponents.kiev.ua/datasheets/BMS1S37V_3A.png

22. XL6009 [Електронний ресурс]. – https://mega-radiodetali.com.ua/uploads/files_predprenimatel/XL6009E1.pdf

23. TPS6302x High Efficiency Single Inductor Buck-boost Converter with 4-F Switches [Електронний ресурс]. – https://www.ti.com/lit/ds/symlink/tps63020.pdf

24. Mini-560 [Електронний ресурс]. – https://www.rcscomponents.kiev.ua/datasheets/JW5068A-datasheet.pdf?srsltid=AfmBOorQm3llXiZ2b9vPQk7sarzhzyCDG5iwkw9b7ZTptlW

25. JX-887Y Power Bank Battery Management System Module [Електронний ресурс]. – https://einstronic.com/product/jx-887y-power-bank-battery-management-system-module/

26. Сонячний модуль 0,75Вт/6В [Електронний ресурс]. – https://www.rcscomponents.kiev.ua/product/soniachnyi-modul-0-75vt-6v_87241.html

27. SG 90 9g Micro Servo [Електронний ресурс]. – https://www.friendlywire.com/projects/ne555-servo-safe/SG90-datasheet.pdf

28. Мінімотор із редуктором 6VDC [Електричний ресурс]. – https://www.rcscomponents.kiev.ua/datasheets/12GAN20-datasheet.pdf

29. ACS712 [Електронний ресурс]. – https://www.allegromicro.com/-/media/files/datasheets/acs712-datasheet.ashx

30. OLED [Електронний ресурс]. – https://www.vishay.com/docs/37902/oled128o064dbpp3n00000.pdf

31. Ковальчук С. П. Автоматизація систем альтернативної енергетики. – Львів: «Новий світ - 2000», 2021. – 152 с.

32. Мельник А. М. Основи проєктування гібридних енергетичних систем: навчальний посібник. – К.: КПІ ім. Ігоря Сікорського, 2020. – 168 с.

33. Савченко В. Г. Сонячна і вітрова енергетика: система управління та контролю. – Харків: УкрЕНЕРГО, 2022. – 134 с.

34. Ткаченко Ю. І. Вітроенергетичні установки з автономним керуванням. – Дніпро: ДНУ, 2019. – 112 с.

35. Яценко П. М. Інтелектуальні системи керування у відновлювальній енергетиці. – К.: Наукова думка, 2021. – 140 с.

36. Міщенко В. І. Альтернативна енергетика: навчальний посібник, - К.:

Центр учбової літератури, 2020. – 248 с.

37. Левченко С. В. Основи сонячної енергетики. – Харків: НТУ «ХПІ», 2021. – 156 с.

38. Данилюк І. В. Вітроенергетика: технології та обладнання. – Львів: Вид-во Львівської політехніки, 2022. – 132 с.

39. Турчак Л. І. Сонячні електростанції: проєктування та експлуатація. – Тернопіль: Навчальна книга – Богдан, 2020. – 144 с.

40. Клименко В. В. Відновлювана енергетика: сонце, вітер, біомаса. – К.: Ліра-К, 2019. – 208 с.

41. Швець С. І. Енергетика майбутнього: основи автоматизації альтернативних джерел. – Черкаси: ЧНУ, 2021. – 160 с.

42. Гаврилюк О. П. Розподілені енергосистеми на основі сонця і вітру. – К.: Техніка, 2022. – 120 с.

43. Ільїн В. М. Гібридні джерела енергії: принципи роботи та контролю. – Дніпро: Ліра, 2020. – 140 с.

44. Баран Ю. М. Мікроенергетика: автономні системи з ВДЕ. – Івано-Франківськ: Нова Ідея, 2021. – 128 с.

45. Зайченко О. Л. Моніторинг та диспетчеризація сонячних електростанцій. – Одеса: ОНПУ, 2020. – 116 с.

46. Назаренко Т. Ю. Практика автоматизації об'єктів ВДЕ. – Суми: СумДУ, 2023. – 150 с.

47. Романенко П. О. Вітряні електричні установки: конструкції та автоматика. – Харків: УкрЕНЕРГО, 2022. – 138 с.

48. Мороз А. С. Системи зберігання енергії в сонячно-вітрових установках. – К.: Енергоатом, 2021. – 130 с.

49. Плахотнюк Ю. С. Гібридні сонячно-вітрові системи: елементи керування. – Луцьк: СНУ імені Лесі Українки, 2023. – 124 с.

50. Олійник Р. В. Автоматизовані системи управління в сонячно-вітрових електростанціях. – Вінниця: ВНТУ, 2023. – 118 с.

Додаток А
(обов’язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку системи автоматизації сонячно-вітрової електростанції.

2 Підстава для розробки

Підставою для розробки служить завдання на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 47-02 від 17.01.2025 року).

3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є розробка програмного забезпечення системи автоматизації сонячно-вітрової електростанції.

4 Джерела розробки

Джерелом цієї випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є стосовна до теми література і існуючі аналоги.

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

– розробка програмної частин системи, а також розробка взаємодії системи з користувачем;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– системи автоматизації сонячно-вітрової електростанції;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

Розроблене програмне забезпечення не повинно мати обмежень на версію драйверів та операційної системи.

5.4 Вимоги до архітектури

Компонент, що розробляється повинен використовувати системні засоби та апаратні засоби, що на даному етапі розвитку обчислювальної техніки найбільше поширені.

5.5 Вимоги до надійності

Програмні модулі написані по всім правилам, які стосуються стандартних викликів процедур, функцій, методів, визначених технічною документацією на середовище розробки.

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

– температура повітря: 19-20 град. по Цельсію;

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

Програмне забезпечення повинно бути реалізоване на ПЕОМ, працювати в ОС Windows 10/11 і з сумісними з цією платформою пристроями і прикладним програмним забезпечення.

5.8 Вимоги до інформаційної і програмної сумісності

Переносність програмного забезпечення повинна бути забезпечена за рахунок його реалізації стандартного інтерфейсу взаємодії з ОС, що працюють під управлінням ОС Windows 10/11.

5.8.1 Обладнання

Комп’ютер Intel® Celeron/8 Mb/1.2 Gb/SVGA 14” 1Mb або сумісні з ним.

5.8.2 Мова програмування

Середовище Arduino IDE C++.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

Програмна продукція повинна бути представлена у виді опису структури даних, схем та опису алгоритму, а також текстів вихідних модулів програмного забезпечення згідно ЄСПД .

7 Перелік документів, що розробляються

- Структурна схема системи – 1 аркуш.
- Функціональна схема системи – 1 аркуш.
- Діаграма процесів – 1 аркуш.
- Блок-схема алгоритму роботи програми – 2 аркуша.
- Пояснювальна записка – 58 аркушів.

8 Етапи розробки

8.1 Збір і обробка інформації по темі випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти. Постановка задачі на виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти (складання ТЗ).

8.2 Проведення досліджень або експериментальних робіт для уточнення основних положень випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти.

8.3 Розробка функціональних схем, блок схем алгоритмів роботи програмного забезпечення.

8.4 Побудова схем взаємодії даних.

8.5 Створення прототипу ПЗ.

8.6 Віднаходження ПЗ, аналіз отриманих результатів.

8.7 Оформлення пояснювальної записки і виконання робіт по графічній частині.

9 Порядок контролю та приймання

9.1 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на попередній захист 23.05.2025 р.

9.2 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на захист 3.06.2025 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти

_____ Дрєєв О.М.

Програмне забезпечення системи автоматизації сонячно-вітрової електростанції

Лістинг програми

Код документу 12

Носій: CD/DVD-диск / USB-флеш-накопичувач

Загальна кількість аркушів: 48

Літера: РП

Кропивницький – 2025 року

Основна програма

Файл main.ino основної програми

```
#include <Arduino.h>

#include "myWifi.h"
#include "MQTTBroker.h"
#include "webServerEvents.h"
#include "serverOn.h"
#include "readVoltageBattery.h"
#include "oledDisplay.h"
#include "ACS712Current.h"
#include "solarTracker.h"

//Основні налаштування
void setup() {
  Serial.begin(115200);

  // Налаштування PWM пінів для серво
  configServoPins();

  // Налаштування фізичної кнопки контролю
  setUpButton();
  // Налаштувння дисплею
  setUpDisplay();

  // Налаштування пінів вимірювання струму
  configCurrentPins();

  // Спроба підключення до WiFi
  wifiConnectInit();
  // Виведення локальної IP-адреси
  Serial.println(WiFi.localIP());

  // Налаштування MQTT
  configMQTT();

  // Web-сервер починає слухати події
  listtenTo();
  eventConnect();
  server.begin();
}

// Нескінченний процес обробки
void loop() {
  // Повторні спроби підключення до WiFi
  wifiConnectLoop();

  if(isWiFiConnected())
  {
    // Вивести IP-адресу на дисплей
    displayIP();

    // Якщо клієнт не підключений
    if (!mqtt_client.connected())
    {
      tryConnectMQTT();
    }
  }

  // Запити від клієнта
  mqtt_client.loop();
  // Надсилання даних на MQTT-сервер
  sendToMQTT();
```

```
  // Вимірюємо напругу
  getVoltage();
  getFinishVoltage();
  getFinishVoltagePercent();

  // Перевірка стану фізичної кнопки
  onOffOled();

  // Визначаємо по черзі струм на панелях та вітрогенераторі
  getSolarWindCurrent();
  // Перевірити, чи акумулятор повнітю заряджений
  ifFullBattery();

  // Зміна положення сервомоторів
  changeServoPos();
}
```

Файл solarTracker.h - бібліотека для файлу solarTracker.cpp

```
#ifndef SOLARTRACKER_H
#define SOLARTRACKER_H

#include "Arduino.h"
#include <ESP32Servo.h>

// Налаштувати піни для серво
void configServoPins();
// Зміна положення
void changeServoPos();

#endif
```

Файл solarTracker.cpp – зміна положення сонячних панелей

```
#include "solarTracker.h"

// Піни для АЦП фоторезисторів
const int topLeft = 36;
const int topRight = 39;
const int botRight = 34;
const int botLeft = 35;

// Інтервал між вимірюваннями (в мс)
unsigned long lastLdrCheck = 0;
const unsigned long ldrInterval = 10000;

// Пін що регулює періодичне відкриття транзистора, для вимірювання
const int pinLdrVt = 13;

// Горизонтальний (вліво-вправо)
Servo servoH;
// Вертикальний (вгору-вниз)
Servo servoV;

// Уникнути дріботіння, і незрозумілих поворотів, поріг спрацювання
int thresholdValue = 30;

// Налаштувати піни для серво
void configServoPins()
{
  // PWM піни котролю повороту серво
  servoH.attach(14);
  servoV.attach(12);
  // Початковий стан
  servoH.write(90);
  servoV.write(90);

  // Налаштувати пін на вихід
  pinMode(pinLdrVt, OUTPUT);
  // Початковий стан закритий
  digitalWrite(pinLdrVt, LOW);

}

// Зміна положення
void changeServoPos()
{
  // ініціалізуємо великим значенням, щоб зайти у цикл
  // Відслідковують величину відмінності в значеннях світлодіодів
  // По горизонталі
  int diffHor = 1000;
  // По вертикалі
  int diffVer = 1000;

  // Визначаємо час точки входу
  unsigned long currentMillis = millis();

  // Чекаємо на час для вимірювання LDR
  // Якщо пройшло 10 секунд,
  if (currentMillis - lastLdrCheck >= ldrInterval)
  {
    // Запам'ятовуємо час, та оновлюємо змінну: останній час входу
    lastLdrCheck = currentMillis;

    // Відкриваємо затвор VT
    digitalWrite(pinLdrVt, HIGH);

    // Допоки відмінність в значеннях не досягне порогового значення точності
    while (abs(diffHor) >= thresholdValue || abs(diffVer) >= thresholdValue)
    {
```

```
Serial.println("Починаю зміну кута сонячних панелей відносно Сонця!");

// Зчитування значення з фотороезисторів
int topr = analogRead(topRight);
int topl = analogRead(topLeft);
int botr = analogRead(botRight);
int botl = analogRead(botLeft);

// Обчислення середніх значень
// Вершина горизонталь
int avgTop = (analogRead(topRight) + analogRead(topLeft)) / 2;
// Низ горизонталь
int avgBot = (analogRead(botRight) + analogRead(botLeft)) / 2;
// Ліво вертикаль
int avgLeft = (analogRead(topLeft) + analogRead(botLeft)) / 2;
// Право вертикаль
int avgRight = (analogRead(topRight) + analogRead(botRight)) / 2;

// Різниця по осях
// Різниця між верхом та низом, вертикаль
diffVer = avgTop - avgBot;
// Різниця між лівою та правою сторонами, горизонталь
diffHor = avgRight - avgLeft;

// Якщо різниця значень більше порогового
if (abs(diffVer) >= thresholdValue)
{
  // Від'ємне або додатнє значення показує куди трба поворот
  if  (diffVer > 0) {
    // Окреслює межі повороту
    if (servoV.read() < 180)
    {
      // Зміна положення береться від теперішнього положення серво
      // Вверх
      servoV.write((servoV.read()  + 2));
    }
  }
  if (diffVer <  0)
  {
    if (servoV.read()  > 0)
    {
      //Вниз, бо внизу більше Сонця за розрахунками
      servoV.write((servoV.read() - 2));
    }
  }
}

// Різниця в горизонталі більше порогового
if (abs(diffHor) >= thresholdValue)
{
  if (diffHor > 0)
   {
    if  (servoH.read() < 180)
    {
      // Вправо
      servoH.write((servoH.read()  + 2));
    }
  }
  if (diffHor <  0)
  {
    if (servoH.read()  > 0)
    {
      //Вліво
      servoH.write((servoH.read() - 2));
    }
  }
}
// трохи затримки для плавності руху
delay(50);
```

```
    }
    Serial.println("Обрано найоптимальніший кут відносно Сонця!");
    // Знову закриваємо затвор, щоб струм не протікав до фоторезисторів постійно
    digitalWrite(pinLdrVt, LOW);
  }
}
```

Файл ACS712Current.h - бібліотека для файлу ACS712Current.cpp

```
#ifndef ACS712CURRENT_H
#define ACS712CURRENT_H

#include "Arduino.h"
#include "readVoltageBattery.h"

// Кінцеві значення
extern float solarCurrent;
extern float windCurrent;

// Налаштування пінів
void configCurrentPins();
// Визначення струму
void getCurrent();

// Визначення окремо струму Сонячних панелей та вітрогенератора
void getSolarWindCurrent();

// Якщо напруга на акумах full, тоді можна вимкнути подачу струму
void ifFullBattery();

#endif
```

Файл ACS712Current.cpp – вимірювання струму

```
#include "ACS712Current.h"

// Пін, який контролює затвор тз для сонячних панелей
// що йдуть до ACS712
const int sonCurrentPin = 19;
// Пін, який контролює затвор тз для вітрогенератор
// що йде до ACS712
const int windCurrentPin = 18;
// Пін 5 не підходить, бо при роботі Wifi цей пін в режимі АЦП нестабільний,
карще взяти 33
const int currentPin = 33; // Пін, підключений до ACS712

// Кінцеві значення
float solarCurrent = 0;
float windCurrent = 0;


// Для визначення реальної напруги
float R1 = 5100.0;  // Ом
float R2 = 10000.0; // Ом


// Налаштування пінів
void configCurrentPins()
{
  // Прийом значень напруги від ACS712
  pinMode(currentPin, INPUT);

  pinMode(sonCurrentPin, OUTPUT);
  // За замовчуванням постійно відкритий, щоб струм постійно проходив до акумів
  digitalWrite(sonCurrentPin, HIGH);
  pinMode(windCurrentPin, OUTPUT);
  digitalWrite(windCurrentPin, HIGH);
}

// Визначення струму
void getCurrent()
{
  // Робота АЦП
  int adc = analogRead(currentPin);
  Serial.println("ADC: " + String(adc));

  // Напруга на АЦП
  float adc_voltage = adc * (3.3 / 4096.0);
  Serial.println("volt: " + String(adc_voltage));
  // Якщо є резистивний подільник напруги
  float current_voltage = (adc_voltage * (R1 + R2)) / R2;
  Serial.println("current_voltage: " + String(current_voltage));

  // Для ACS712: 2.5 В — нульовий струм, чутливість 185 мВ/А (для ACS712-5A)
  float current = (current_voltage - 2.5) / 0.185;

  // Якщо сонячний тз закрився, тоді вимірюємо струм на вітрогенераторі
  if (sonCurrentPin == LOW)
  {
    windCurrent = current;
  }
  // Якщо вітрогенраторний тз закрився, тоді вимірюємо струм на сонячних панелях
  else if (windCurrentPin == LOW)
  {
    solarCurrent = current;
  }

  Serial.print("Поточне значення: ");
  Serial.print(current);
  Serial.println(" A");
```

```
  }

// Визначення окремо струму Сонячних панелей та вітрогенератора
void getSolarWindCurrent()
{
  // Закриваємо транзистор, що йде до сонячних панелей
  digitalWrite(sonCurrentPin, LOW);
  getCurrent();
  // Закриваємо транзистор, що йде до вітрогенератор
  digitalWrite(sonCurrentPin, LOW);
  getCurrent();

  // Знову відновлюємо роботу
  digitalWrite(sonCurrentPin, HIGH);
  digitalWrite(windCurrentPin, HIGH);
}

// Якщо напруга на акумах full, тоді можна вимкнути подачу струму
void ifFullBattery()
{
  if (realVoltage == 4.2)
  {
    Serial.println("Напргуа на акумуляторах 100%!");
    Serial.println("Відключаю генеративні джерела!");
    digitalWrite(sonCurrentPin, LOW);
    digitalWrite(windCurrentPin, LOW);
  }
  else if (realVoltage <= 4.1)
  {
    Serial.println("Напргуа на акумуляторах впала нижче 4.1 В!");
    Serial.println("Підключаю генеративні джерела!");
    digitalWrite(sonCurrentPin, HIGH);
    digitalWrite(windCurrentPin, HIGH);
  }
}
```

Файл readVoltageBattery.h - бібліотека для файлу readVoltageBattery.cpp

```
#ifndef READVOLTAGEBATTERY_H
#define READVOLTAGEBATTERY_H

#include "Arduino.h"

// Кінцеве значення після mapFloatVolt()
extern float realVoltage;
// Кінцеве значення після mapFloatPercent()
extern float realVoltagePercent;

/*// Початкові значення напргуи та відсотків, для Гістерезису
extern float initial_volt;
extern float initial_percent;
// Поріг зміни: на 1% припадає 42 мВ, ми use 1,5%
extern float threshold_volt;
extern float threshold_percent;*/

void getVoltage();
void getFinishVoltage();
void getFinishVoltagePercent();

// Повертає значення з одним знаком після крапки
float mapFloatVolt(float x, float inMin, float inMax, float outMin, float
outMax);
// Повертає значення у відсотках з одним знаком після крапки
float mapFloatPercent(float x, float inMin, float inMax, int outMin, int
outMax);

#endif
```

Файл readVoltageBattery.cpp – визначення напруги

```
#include "readVoltageBattery.h"

// В цій точці дільник напруги
const int adcResDivBatt = 32;

// Змінна зберігає значення АЦП
int adcValue = 0;
// Перетворення АЦП в значення напруги
float adcToVoltage = 0;
// Кінцеве значення, реальне значення напруги після mapFloatVolt()
float realVoltage = 0;
// Кінцеве значення напруги в відсотках, після mapFloatPercent()
float realVoltagePercent = 0;

// Переведення значення АЦП в напругу
void getVoltage()
{
  adcValue = analogRead(adcResDivBatt);
  //adc_value = adc1_get_raw(ADC1_CHANNEL_4);
  Serial.println("Значення АЦП: " + String(adcValue));

  // Переведення АЦП в вольти
  adcToVoltage = adcValue * 3.3 / 4095;
  Serial.println("АЦП в вольти: " + String(adcToVoltage, 2));
}

// Визначення реальної напруги на акумах
void getFinishVoltage()
{
  realVoltage = mapFloatVolt(adcToVoltage, 2.35, 3.3, 3.0, 4.2);
  Serial.println("Реальна напруга на акумуляторах: " + String(realVoltage, 2));
}

// Визначення реальної напруги в відсотках
void getFinishVoltagePercent()
{
  // Переводимо вольти у відсотки
  realVoltagePercent = mapFloatPercent(realVoltage, 3.0, 4.2, 0, 100);
  Serial.println("Реальна напруга у відсотках: " + String(realVoltagePercent,
2));
}

//Функція map тільки для float, та повернення значення реальних вольт на акумах
float mapFloatVolt(float x, float inMin, float inMax, float outMin, float
outMax)
{
  return round(((x - inMin) * (outMax - outMin) / (inMax - inMin) + outMin) *
100.0) / 100.0;
}

// Фунція map для переведення значень напруги (float) в відсотки (int)
float mapFloatPercent(float x, float inMin, float inMax, int outMin, int outMax)
{
  return round(((x - inMin) * (outMax - outMin) / (inMax - inMin) + outMin) *
10.0) / 10.0;
}
```

Файл oledDisplay.h - бібліотека для файлу oledDisplay.cpp

```
#ifndef oledDisplay_H
#define oledDisplay_H

#include <Arduino.h>

#include "readVoltageBattery.h"
#include "myWifi.h"

#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 32

// Відслідковуємо стан кнопки для OLED
extern bool globalOledButton;

// Налаштувати кнопку
void setUpButton();

// Налаштувати дисплей
void setUpDisplay();

// Перевіряємо кнопку
void onOffOled();

// Виводить текст на дисплей
void setDisplayOutput();

void displayIP();

#endif
```

Файл oledDisplay.cpp – керування виведення на дисплей

```
#include "oledDisplay.h"

// Якщо контроль вкл/викл через Web або MQTT
bool globalOledButton = false;

// Для кнопки, що контролює затвор VT
const int buttonVTPin = 23;
// Вхід на затвор
const int gateVTPin = 19;

unsigned long lastDebounceTime = 0;
const unsigned long debounceDelay = 2000;

// Стан фізичної кнопки вкл/викл OlED
int phisButtonState = HIGH;

// Прапорець слідкує, щоб IP вивелося на екран лише один раз
bool ipDisplayed = false;

// Ініціалізація дисплею SSD1306, для підключення до I2C
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);

// Налаштувати кнопку
void setUpButton()
{
  pinMode(buttonVTPin, INPUT_PULLUP);  // Вхід з підтягуванням
  pinMode(gateVTPin, OUTPUT);
}

// Налаштувати дисплей
void setUpDisplay()
{
  if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
    //Не можнемо знайти пристрій з такою адресою, або взагалі пристрій
    Serial.println(F("SSD1306 allocation failed"));
    // Нескінченний цикл
    for(;;);
  }
  delay(200);
  display.clearDisplay();
}

// Перевіряємо кнопку
void onOffOled()
{
  //Oled begin
  //Треба перевірити, чи не контролюється OLED-дисплей за допомогою Web-server
  if (globalOledButton)
  {
    phisButtonState = LOW;
  }
  else
  {
    phisButtonState = digitalRead(buttonVTPin);
    Serial.println(phisButtonState);
  }


  if ((millis() - lastDebounceTime) > debounceDelay)
  {
    if (phisButtonState == LOW)
    {
      Serial.println("button Low");
      digitalWrite(gateVTPin, HIGH);
      setDisplayOutput();
    }
```

```
  else
  {
      display.clearDisplay();
      // Не встигає очиститись, а вже треба виключатись
      delay(20);
      display.display();
      digitalWrite(gateVTPin, LOW);   // Кнопка відпущена — вимкнути
      Serial.println("button High");
    }
    lastDebounceTime = millis();
  }
}

//Вивдення тексту на OLED
void setDisplayOutput()
{
  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(WHITE);
  // По центру
  display.setCursor(13, 11);
  display.println("V(B): " + String(realVoltage, 2) + "  " +
String(realVoltagePercent, 1) + "%");
  display.display();

}

void displayIP()
{
  if (!ipDisplayed)
  {
    display.clearDisplay();
    display.setTextSize(1);
    display.setTextColor(WHITE);

    String localIP = WiFi.localIP().toString();
    display.setCursor(1, 1);
    display.println(localIP);
    display.display();
    // Затримка, щоб користувач зміг побачити IP
    delay(10000);

    // Щоб більше сюди не заходити
    ipDisplayed = true;
  }
}
```

Файл webServerEvents.h - бібліотека для файлу webServerEvents.cpp

```
#ifndef WEBSERVEREVENTS_H
#define WEBSERVEREVENTS_H

#include <Arduino.h>
#include <AsyncTCP.h>
#include <ESPAsyncWebServer.h>

// Відслідковуємо події від клієнта
void eventConnect();

#endif
```

Файл webServerEvents.cpp – підключення веб-подій

```
#include "webServerEvents.h"

// Джерело подій
AsyncEventSource events("/events");

void eventConnect()
{
  // З'єднання
  events.onConnect([](AsyncEventSourceClient *client){
    if(client->lastId()){
      Serial.printf("Client reconnected! Last message ID that it got is: %u\n",
client->lastId());
    }
    client->send("hello!", NULL, millis(), 10000);
  });
}
```

Файл myWifi.h - бібліотека для файлу myWifi.cpp

```
#ifndef MYWIFI_H
#define MYWIFI_H

#include <Arduino.h>
#include <WiFi.h>

// Спроба з'єдання з Інтеренетом
void wifiConnectInit();

// Повторюваність спроб підключення, але без затримок
void wifiConnectLoop();
// Повернення статусу підключення
bool isWiFiConnected();

#endif
```

Файл myWifi.cpp – підключення до WiFi

```
#include "myWifi.h"

// Параметри підключення до WiFi
const char* ssid = "*****************";
const char* password = "*********";

// Насупна спроба підключення до WiFi буде через 5 секунд
unsigned long previousWifiAttempt = 0;
const unsigned long wifiRetryInterval = 5000;
// Прапорець підключення до WiFi
bool wifiConnected = false;

//Спроба під'єднатися до Інтернету
void wifiConnectInit()
{
  WiFi.begin(ssid, password);
  Serial.println("Підключення до WiFi...");
}

// Постійні спроби під'єднатися до WiFi
void wifiConnectLoop()
{
  // Якщо підлкючено
  if (WiFi.status() == WL_CONNECTED)
  {
    // Якщо прапорець false, то його треба зиінити і вивести Ip лише раз
    if (!wifiConnected)
    {
        wifiConnected = true;
        Serial.println("WiFi connected!");
        Serial.println(WiFi.localIP());
    }
  }
  // якщо не підключено
  else
  {
    // Щоб якщо відключитися від Wifi, потім знову мати змогу підключатися
    wifiConnected = false;

    // Якщо пройшло 5 секунд
    if (millis() - previousWifiAttempt >= wifiRetryInterval)
    {
        previousWifiAttempt = millis();

        Serial.println("Перепідключення WiFi...");
        WiFi.disconnect();
        WiFi.begin(ssid, password);
    }
  }
}

// Статус підключення до WiFi
bool isWiFiConnected()
{
  return wifiConnected;
}
```

Файл serverOn.h - бібліотека для файлу serverOn.cpp

```
#ifndef SERVERON_H
#define SERVERON_H

#include "Arduino.h"
#include <AsyncTCP.h>
#include <ESPAsyncWebServer.h>
#include <LittleFS.h>

#include "readVoltageBattery.h"
#include "oledDisplay.h"
#include "ACS712Current.h"

// Створюємо Асинхронних Веб-сервер з портом 80
extern AsyncWebServer server;

// Сервер прослуховує запити
void listtenTo();
// Файлова система для верстування сайту
void initLittleFS();

#endif
```

Файл serverOn.cpp – керування подіями веб-сервера

```
#include "serverOn.h"

// Створюємо асинхронний Web-сервер на порту 80
AsyncWebServer server(80);

// Чи є в GET-запиті параметр value
const char* PARAM_INPUT = "value";

// Надсилає актуальні дані на веб сторінку, при запиті SENSORVALUE, SENSORVALUE
String processor(const String& var){
  Serial.println(var);
  if (var == "VOLTS"){
    String realVoltageStr = String(realVoltage, 2);
    return realVoltageStr;
  }
  if (var == "PERCENT")
  {
    String realVoltagePercentStr = String(realVoltagePercent, 1);
    return realVoltagePercentStr;
  }
  return String();
}

void initLittleFS()
{
  /* LittleFS.begin() - це запуск файлової системи ESP32, без якої
  клієнт не зможе викликати сторінку з серверу.*/
  if(!LittleFS.begin()){
    Serial.println("Помилка в запуску файлової системи LittleFS");
    return;
  }
}

void listtenTo()
{
  // Ініціалізація файлової ситеми
  initLittleFS();

  // Шлях до корінної сторінки
  server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send(LittleFS, "/index.html", String(), false, processor);
  });

  // Шлях до файлу style.css
  server.on("/style.css", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send(LittleFS, "/style.css", "text/css");
  });

  // Шлях до файлу script.js
  server.on("/script.js", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send(LittleFS, "/script.js", "text/javascript");
  });

  /*Параметри сонячних панелей*/
  server.on("/son_V", HTTP_GET, [] (AsyncWebServerRequest *request) {
    String sonVoltage = String(5);
    request->send_P(200, "text/plain", sonVoltage.c_str());
  });

  server.on("/son_C", HTTP_GET, [] (AsyncWebServerRequest *request) {
    String sonCurrent = String(solarCurrent);
    request->send_P(200, "text/plain", sonCurrent.c_str());
  });

  server.on("/son_P", HTTP_GET, [] (AsyncWebServerRequest *request) {
    String sonPower = String(solarCurrent * 5, 2);
```

```
    request->send_P(200, "text/plain", sonPower.c_str());
  });

  // Параметри вітрогенератора
  server.on("/wind_V", HTTP_GET, [] (AsyncWebServerRequest *request) {
    String windVoltage = String(5);
    request->send_P(200, "text/plain", windVoltage.c_str());
  });

  server.on("/wind_C", HTTP_GET, [] (AsyncWebServerRequest *request) {
    String windCurrent = String(windCurrent);
    request->send_P(200, "text/plain", windCurrent.c_str());
  });

  /*Значення напруги та потужності взяті з повітря. тай ACS712 не працює*/
  server.on("/wind_P", HTTP_GET, [] (AsyncWebServerRequest *request) {
    String windPower = String(windCurrent * 5, 2);
    request->send_P(200, "text/plain", windPower.c_str());
  });

  // Відслідковуємо зміни датчика, та надсилаємо їх до клієнта в рядок
"теперішнього значення датчика", кожні 5 секунд
  server.on("/getVoltsValue", HTTP_GET, [] (AsyncWebServerRequest *request) {
    String realVoltageStr = String(realVoltage);
    request->send_P(200, "text/plain", realVoltageStr.c_str());
  });

  // Значення вологості кожні 30 секунд для графіку
  server.on("/getPercentValue", HTTP_GET, [](AsyncWebServerRequest *request){
    String realVoltagePercentStr = String(realVoltagePercent);
    request->send_P(200, "text/plain", realVoltagePercentStr.c_str());
  });

  // Відслідковуємо дії з кнопкою
  server.on("/turn", HTTP_GET, [] (AsyncWebServerRequest *request) {
    String inputMessage;
    if (request->hasParam(PARAM_INPUT))
    {
      inputMessage = request->getParam(PARAM_INPUT)->value();
      // Це влючити
      if (inputMessage == "OFF")
      {
        // Вказує на те, що зараз вкл/викл контролюється Web або MQTT
        globalOledButton = true;
        // Вивести значення на екран
        setDisplayOutput();
      }
      else
      {
        globalOledButton = false;
        Serial.println("Запит від Web-сервера: викл OLED!");
      }
    }
    request->send(200, "text/plain", "OK");
  });
}
```

Файл MQTTBroker.h - бібліотека для файлу MQTTBroker.cpp

```
#ifndef MQTTBROKER_H
#define MQTTBROKER_H

#include "Arduino.h"
#include <ArduinoJson.h>

#include <WiFi.h>
#include <PubSubClient.h>
#include <WiFiClientSecure.h>

#include "oledDisplay.h"
#include "readVoltageBattery.h"


extern PubSubClient mqtt_client;

void tryConnectMQTT();
// Функція для отримання команд
void mqttCallback(char *topic, byte *payload, unsigned int length);

//Дані для періодичного надсилання
void sendToMQTT();

void configMQTT();

#endif
```

Файл MQTTBroker.cpp – керування MQTT-серером

```
#include "MQTTBroker.h"

// Налаштування MQTT Broker
const char *mqtt_broker = "x2ec1022.ala.eu-central-1.emqxsl.com";
const char *mqtt_topic = "esp32/test";
const char *mqtt_username = "esp32";
const char *mqtt_password = "esp32begin";
const int mqtt_port = 8883;  // (TLS)

// Ініціалізація WiFi та MQTT-клієнт
WiFiClientSecure esp_client;
PubSubClient mqtt_client(esp_client);

// Інтервал періодичного надсилання даних на сервер
unsigned long lastPublishTime = 0;
const unsigned long publishInterval = 10000; // 10 секунд

// Інтервал підключення до MQTT
unsigned long mqttLastAttempt = 0;
const unsigned long mqttRetryInterval = 5000;

// Якщо тариф Serverless, тоді обираємо саме цей сертифікат
const char* ca_cert = R"EOF(
-----BEGIN CERTIFICATE-----
MIIDrzCCApegAwIBAgIQCDvgVpBCRrGhdWrJWZHHSjANBgkqhkiG9w0BAQUFADBh
MQswCQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMRkwFwYDVQQLExB3
d3cuZGlnaWNlcnQuY29tMSAwHgYDVQQDExdEaWdpQ2VydCBHbG9iYWwgUm9vdCBD
QTAeFw0wNjExMTAwMDAwMDBaFw0zMTExMTAwMDAwMDBaMGExCzAJBgNVBAYTAlVT
MRUwEwYDVQQKEwxEaWdpQ2VydCBJbmMxGTAXBgNVBAsTEHd3dy5kaWdpY2VydC5j
b20xIDAeBgNVBAMTF0RpZ2lDZXJ0IEdsb2JhbCBSb290IENBMIIBIjANBgkqhkiG
9w0BAQEFAAOCAQ8AMIIBCgKCAQEA4jvhEXLeqKTTo1eqUKKPC3eQyaKl7hLOllsB
CSDMAZOnTjC3U/dDxGkAV53ijSLdhwZAAIEJzs4bg7/fzTtxRuLWZscFs3YnFo97
nh6Vfe63SKMI2tavegw5BmV/Sl0fvBf4q77uKNd0f3p4mVmFaG5cIzJLv07A6Fpt
43C/dxC//AH2hdmoRBBYMql1GNXRor5H4idq9Joz+EkIYIvUX7Q6hL+hqkpMfT7P
T19sdl6gSzeRntwi5m3OFBqOasv+zbMUZBfHWymeMr/y7vrTC0LUq7dBMtoM1O/4
gdW7jVg/tRvoSSiicNoxBN33shbyTApOB6jtSj1etX+jkMOvJwIDAQABo2MwYTAO
BgNVHQ8BAf8EBAMCAYYwDwYDVR0TAQH/BAUwAwEB/zAdBgNVHQ4EFgQUA95QNVbR
TLtm8KPiGxvDl7I90VUwHwYDVR0jBBgwFoAUA95QNVbRTLtm8KPiGxvDl7I90VUw
DQYJKoZIhvcNAQEFBQADggEBAMucN6pIExIK+t1EnE9SsPTfrgT1eXkIoyQY/Esr
hMAtudXH/vTBH1jLuG2cenTnmCmrEbXjcKChzUyImZOMkXDiqw8cvpOp/2PV5Adg
06O/nVsJ8dWO41P0jmP6P6fbtGbfYmbW0W5BjfIttep3Sp+dWOIrWcBAI+0tKIJF
PnlUkiaY4IBIqDfv8NZ5YBberOgOzW6sRBc4L0na4UU+Krk2U886UAb3LujEV0ls
YSEY1QSteDwsOoBrp+uvFRTp2InBuThs4pFsiv9kuXclVzDAGySj4dzp30d8tbQk
CAUw7C29C79Fv1C5qfPrmAESrciIxpg0X40KPMbp1ZWVbd4=
-----END CERTIFICATE-----
)EOF";

// Спроба бідключення до MQTT
void tryConnectMQTT() {
    if (!mqtt_client.connected() &&  millis() - mqttLastAttempt >
mqttRetryInterval) {

        mqttLastAttempt = millis();

        String client_id = "esp32-client-" + String(WiFi.macAddress());
        Serial.printf("Connecting to MQTT Broker as %s...\n",
client_id.c_str());

        if (mqtt_client.connect(client_id.c_str(), mqtt_username,
mqtt_password)) {
            Serial.println("Connected to MQTT broker");
            mqtt_client.subscribe(mqtt_topic);
            mqtt_client.publish(mqtt_topic, "Hi EMQX I'm ESP32 ^^");  // Publish
message upon connection
        } else {
            Serial.print("Failed to connect to MQTT broker, rc=");
```

```
            Serial.print(mqtt_client.state());
            Serial.println(" Retrying in 5 seconds.");
        }
    }
}

// Прийом даних від MQTT-брокера
void mqttCallback(char *topic, byte *payload, unsigned int length) {
  StaticJsonDocument<200> doc;

  DeserializationError error = deserializeJson(doc, payload, length);
  if (error) {
    Serial.print("JSON parsing failed: ");
    Serial.println(error.c_str());
    return;
  }

  String msg = doc["msg"];

  Serial.print("Message received: ");
  Serial.println(msg);

  if (msg.equalsIgnoreCase("on")) {
    Serial.println("Turning ON OLED...");
    globalOledButton = true;
    setDisplayOutput();
  } else if (msg.equalsIgnoreCase("off")) {
    Serial.println("Turning OFF OLED...");
    globalOledButton = false;
    Serial.println("Запит від MQTT-сервера: викл OLED!");
  } else {
    Serial.println("Unknown command in JSON.");
  }
}

//Надсилання даних
void sendToMQTT()
{
  unsigned long currentMillis = millis();

  if (currentMillis - lastPublishTime >= publishInterval) {
    lastPublishTime = currentMillis;

    // Дані для надсилання
    String payload = "B(V): " + String(realVoltage, 2) + "  " +
String(realVoltagePercent, 1) + "%";
    mqtt_client.publish(mqtt_topic, payload.c_str());

    Serial.println("Published: " + payload);
  }
}

// Налаштування MQTT-брокера
void configMQTT()
{
  // Встановлюємо сертифікат довіри (Root CA) для захищеного TLS-з'єднання
  esp_client.setCACert(ca_cert);

  // Вказуємо адресу MQTT-брокера
  mqtt_client.setServer(mqtt_broker, mqtt_port);
  // MQTT-клієнт посилає періодичні пакети, щоб брокер знав, що клієнт ще на
зв'зяку
  mqtt_client.setKeepAlive(60);
  // Встановлюємо фукнцію зворотного зв'язку, яка викликається при отриманні
MQTT-повідомлення
  mqtt_client.setCallback(mqttCallback);
}
```

Файл index.hmtl – веб-сторінка

```
<!DOCTYPE html>
<html lang="ua">
<head>
      <meta charset="UTF-8">
      <meta http-equiv="X-UA-Compatible" content="IE=edge">
      <meta name="viewport" content="width=device-width, initial-scale=1.0">
      <title>Контроль поливу</title>
      <link rel="stylesheet" href="style.css">
      <script src="https://code.highcharts.com/highcharts.js"></script>
</head>
<body>
      <div class="group">
            <h3>Сонячна панель</h3>
            <div class="wrapper">
              <div id="son-voltage" class="container">sdfsdfsd</div>
              <div id="son-current" class="container">sdfsdfsd</div>
              <div id="son-power" class="container">sdfsdfsd</div>
            </div>
        </div>

        <div class="group">
            <h3>Вітрогенератор</h3>
            <div class="wrapper">
              <div id="wind-voltage" class="container">sdfsdfsd</div>
              <div id="wind-current" class="container">sdfsdfsd</div>
              <div id="wind-power" class="container">sdfsdfsd</div>
            </div>
        </div>

      <div class="control-row">
            <div class="switch-block">
            <label for="toggle" class="switch-label">Вкл/викл OLED-
дисплей</label>
            <button id="toggle" onclick="buttonProcess(this)">ON</button>
            </div>

            <div class="battery-status">
            <span class="volts">B(V): <span id="volts">%VOLTS%</span></span>
                  <span id="volts-percent" class="volts-
percent">%PERCENT%%</span>
            </div>
      </div>

      <script src="script.js"></script>
</body>
</html>
```

Файл style.css – стиль веб-сторінки

```
*
{
  margin: 0;
}

/*Це для графіку*/
.container
{
  width: 450px;
  height: 225px;
  border: 1px solid #aaa;
  display: flex;
  align-items: center;
  justify-content: center;
}

.wrapper, .wrapper1
{
  display: flex;
  justify-content: center;
  align-items: center;
  gap: 20px;
  margin-top: 30px;
}

.group
{
  text-align: center;
  margin-bottom: 15px;
}

.group h3
{
  margin-bottom: 10px;
  font-size: 1.8em;
  color: #333;
}

.control-row
{
  display: flex;
  align-items: center;
  gap: 30px;
  margin-top: 20px;
}

.switch-block
{
  display: flex;
  align-items: center;
  gap: 10px;
  margin-left: 70px;
}

.switch-label
{
  font-weight: 700;
  font-size: 18px;
  color: #333;
}

.switch-block button
{
  padding: 8px 16px;
  font-size: 16px;
  background-color: green;
```

```
  color: white;
  border: none;
  border-radius: 6px;
  cursor: pointer;
  min-width: 70px;
  box-sizing: border-box;
  transition: background-color 0.2s ease;

}

.switch-block button:active
{
  background-color: darkgreen;
  transform: translateY(1px); /* опціонально, щоб не зміщувалась */
}

.switch-block button:hover
{
  background-color: darkgreen;
}

.battery-status
{
  font-size: 20px;
  font-weight: 800;
  color: #444;
  margin-left: 250px;
}

.volts-percent
{
  margin-left: 7px;
}
```

Файл script.js – поведінка веб-сторінки

```
//Функція для оновлення значення для клієнта
function fetchAndUpdateValue(url, elementId, description) {
    const xhr = new XMLHttpRequest();
    xhr.open("GET", url, true);
    xhr.onreadystatechange = function () {
        if (xhr.readyState === 4 && xhr.status === 200) {
            const newValue = xhr.responseText;
            document.getElementById(elementId).innerHTML = newValue;
            console.log(`Нове значення ${description}: ${newValue}`);
        }
    };
    xhr.send();
}

// Виклики з інтервалами
// Всі запити одночасно відбуваються
setInterval(() => fetchAndUpdateValue("/getVoltsValue", "volts", "напруги
акумів"), 5000);
setInterval(() => fetchAndUpdateValue("/getPercentValue", "volts-percent",
"акума у відсотках"), 5000);

/*Вкл/Викл Ручний режим*/
function buttonProcess(bt)
{
  var mode = "";
  /* Включаємо,*/
  if (bt.innerHTML == "ON")
  {
    bt.style.backgroundColor = "#555555";
    mode = "OFF";
    bt.innerHTML = mode;
  }
  else if (bt.innerHTML == "OFF")
  {
    bt.style.backgroundColor = "#4CAF50";
    mode = "ON";
    bt.innerHTML = mode;
  }

  var xhr = new XMLHttpRequest();
  xhr.open("GET", "/turn?value=" + mode, true);
  console.log("/turn?value=" + mode);
  xhr.send();
}


// Встановлюємо локальний час замість UTC
Highcharts.setOptions({
  time: {
    useUTC: false
  }
});


/*Для графіку*/
function createChart(renderToId, titleText, yAxisLabel, dataUrl) {
  const chart = new Highcharts.Chart({
    chart: { renderTo: renderToId },
    title: { text: titleText },
    series: [{
      showInLegend: false,
      data: []
    }],
    plotOptions: {
      line: {
        animation: false,
```

```
        dataLabels: { enabled: true }
      },
      series: { color: '#059e8a' }
    },
    xAxis: {
      type: 'datetime',
      dateTimeLabelFormats: { second: '%H:%M:%S' }
    },
    yAxis: {
      title: { text: yAxisLabel }
    },
    credits: { enabled: false }
  });

  setInterval(() => {
    const xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function () {
      if (this.readyState == 4 && this.status == 200) {
        const x = (new Date()).getTime();
        const y = parseFloat(this.responseText);
        const shift = chart.series[0].data.length > 40;
        chart.series[0].addPoint([x, y], true, shift, true);
      }
    };
    xhttp.open("GET", dataUrl, true);
    xhttp.send();
  }, 30000); // 30 секунд
}

// Виклики для кожного графіка
createChart('son-voltage', 'Напруга', 'U(мВ)', '/son_V');
createChart('son-current', 'Струм', 'I(мА)', '/son_C');
createChart('son-power', 'Потужність', 'P(мВт)', '/son_P');

createChart('wind-voltage', 'Напруга', 'U(мВ)', '/wind_V');
createChart('wind-current', 'Струм', 'I(мА)', '/wind_C');
createChart('wind-power', 'Потужність', 'P(мВт)', '/wind_P');

/*Клієнт слухає надходження івентів від серверу.*/
if (!!window.EventSource) {
      var source = new EventSource('/events');
      source.addEventListener('open', function(e) {
            console.log("Events Connected");
      }, false);
      source.addEventListener('error', function(e) {
            if (e.target.readyState != EventSource.OPEN) {
            console.log("Events Disconnected");
            }
      }, false);

      source.addEventListener('message', function(e) {
            console.log("message", e.data);
      }, false);

      source.addEventListener('on', function(e) {
            console.log("on");
            document.getElementById("on_off").innerHTML = "OFF";
            document.getElementById("on_off").style.backgroundColor = "#555555";
      }, false);

      source.addEventListener('off', function(e) {
            console.log("on");
            document.getElementById("on_off").innerHTML = "ON";
            document.getElementById("on_off").style.backgroundColor = "#4CAF50";
      }, false);
}
```