НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ПОГОДЖЕНО	**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ**
Декан факультету Інформаційних технологій	**Завідувач кафедри** Комп'ютерних систем, мереж та кібербезпеки
_____Ігор БОЛБОТ (підпис)	_____Дмитро КАСАТКІН (підпис)
«___»_____2026 р.	«___»_____2026 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

На тему: «Розробка WEB-сервісу з моніторингу стану СЕС»

Спеціальність F7 «Комп'ютерна інженерія»

Освітньо-професійна програма «Комп'ютерна інженерія»

Гарант освітньої програми канд. фіз.-мат. наук, доцент	_____ (підпис)	**Євгеній НІКІТЕНКО**
Керівник бакалаврської кваліфікаційної роботи канд. техн. наук, доцент	_____ (підпис)	**Віктор СМОЛІЙ**
Виконав	_____ (підпис)	**Ярослав СКРИНІК**

КИЇВ-2026

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ЗАТВЕРДЖУЮ
Завідувач кафедри
комп'ютерних систем, мереж та кібербезпеки
канд. пед. наук, доцент_____Дмитро КАСАТКІН
«_____»_____20____ р.

З А В Д А Н Н Я

ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
ЗДОБУВАЧУ

Скриніку Ярославу Віталійовичу
(прізвище, ім'я, по батькові)

Спеціальність «Комп'ютерна інженерія »

Освітньо-професійна програма «Комп'ютерна інженерія »

Тема кваліфікаційної бакалаврської роботи
«Розробка WEB-сервісу з моніторингу стану СЕС
»

затверджена наказом ректора НУБіП України від «10»__12_____2025 р. № 3072 «С»___
Термін подання завершеної роботи на кафедру «22»__05_2026 р.
Вихідні дані до бакалаврської кваліфікаційної роботи

Перелік питань, що підлягають дослідженню:

Перелік графічного матеріалу (за необхідності).

Дата видачі завдання "__10__"_____12_____2025 р.

Керівник бакалаврської кваліфікаційної роботи канд. техн. наук, доцент	_____ (підпис)	**Віктор СМОЛІЙ**
Завдання взяв до виконання	_____ (підпис)	**Ярослав СКРИНІК**

Перелік умовних позначень та скорочень

1. АЦП – аналого-цифровий перетворювач (компонент мікроконтролера ESP32).
2. ВДЕ – відновлювані джерела енергії.
3. ККД – коефіцієнт корисної дії.
4. ПЗ – програмне забезпечення.
5. ПК – персональний комп'ютер.
6. СЕС – сонячна електрична станція (сонячна електростанція).
7. СУБД – система керування базами даних.
8. API (Application Programming Interface) – інтерфейс прикладного програмування.
9. ADC (Analog-to-Digital Converter) – аналого-цифровий перетворювач.
10. BaaS (Backend-as-a-Service) – бекенд як послуга (хмарна модель розробки).
11. BLE (Bluetooth Low Energy) – технологія бездротового зв'язку з низьким енергоспоживанням.
12. CDN (Content Delivery Network) – мережа доставки контенту.
13. CSS (Cascading Style Sheets) – каскадні таблиці стилів.
14. DOM (Document Object Model) – об'єктна модель документа в браузері.
15. ESP32 – сімейство низьковартісних та енергоефективних мікроконтролерів компанії Espressif Systems із вбудованими модулями Wi-Fi та Bluetooth.
16. HTML (HyperText Markup Language) – стандартизована мова розмітки веб-сторінок.
17. HTTP / HTTPS (HyperText Transfer Protocol / Secure) – протокол передачі гіпертексту (захищений).

18. I2C / I²C (Inter-Integrated Circuit) – послідовна асиметрична шина для зв'язку між інтегральними схемами.

19. IoT (Internet of Things) – Інтернет речей.

20. JS (JavaScript) – мультипарадигмова мова програмування високого рівня.

21. JSON (JavaScript Object Notation) – текстовий формат обміну даними між клієнтом і сервером.

22. NoSQL (Not Only SQL) – загальна назва некореляційних баз даних.

23. PWM (Pulse-Width Modulation) – широтно-імпульсна модуляція (ШІМ).

24. REST (Representational State Transfer) – архітектурний стиль взаємодії компонентів розподіленого веб-додатка через HTTP.

25. SDK (Software Development Kit) – набір інструментів і бібліотек для розробки програмного забезпечення.

26. SPA (Single Page Application) – односторінковий веб-додаток.

27. SoC (System on a Chip) – система на кристалі.

28. STATE – глобальний об'єкт стану клієнтського додатка (Single Source of Truth).

29. UI (User Interface) – інтерфейс користувача.

30. URL (Uniform Resource Locator) – уніфікований визначник місцезнаходження ресурсу в мережі Інтернет.

РЕФЕРАТ

Пояснювальна записка до випускної кваліфікаційної роботи: 73 сторінки, 12 рисунків, 5 таблиць, 20 джерел посилання, 4 додатки.

ОБ'ЄКТ ДОСЛІДЖЕННЯ – процеси автоматизованого збору, обробки, передачі та візуалізації телеметричних даних поточної енергетичної генерації об'єктів відновлюваної енергетики. ПРЕДМЕТ ДОСЛІДЖЕННЯ – програмно-апаратні засоби, безсерверна архітектура, хмарні NoSQL бази даних реального часу та веб-інтерфейси для моніторингу параметрів сонячних електростанцій (СЕС).

МЕТА РОБОТИ – проектування, програмна розробка та практичне тестування високоефективного, масштабованого та доступного безсерверного веб-сервісу моніторингу параметрів сонячної електростанції у реальному часі на основі мікроконтролера ESP32 та хмарної інфраструктури Firebase Firestore.

МЕТОДИ ДОСЛІДЖЕННЯ – методи об'єктно-орієнтованого проектування ПЗ, архітектурні патерни Single Page Application (SPA), технології асинхронної клієнт-серверної взаємодії (WebSockets, Async/Await), NoSQL моделювання даних, концепції Інтернету речей (IoT) та мікроконтролерного програмування.

У процесі виконання роботи було спроектовано та практично реалізовано безсерверний веб-сервіс для оперативного моніторингу фізичних метрик СЕС (напруга, струм, миттєва потужність, добова генерація). Клієнтська панель реалізована за технологією SPA (HTML5, Tailwind CSS, JavaScript ES6+) з реактивним управлінням інтерфейсом через глобальний об'єкт стану `STATE`. Бекенд-частина побудована за моделлю Backend-as-a-Service (BaaS) на платформі Google Firebase Cloud Firestore, що забезпечило синхронізацію даних із затримкою менше 200 мс. Інтегровано режим симуляції на основі реальних кліматичних умов за допомогою Open-Meteo API. Проведено тестування системи

у зв'язці з віртуальним середовищем апаратної телеметрії. Розроблені рішення мають високу практичну цінність для приватних СЕС та освітнього процесу.

КЛЮЧОВІ СЛОВА: СОНЯЧНА ЕЛЕКТРОСТАНЦІЯ, МОНІТОРИНГ РЕАЛЬНОГО ЧАСУ, ІНТЕРНЕТ РЕЧЕЙ (IoT), ESP32, FIREBASE FIRESTORE, BACKEND-AS-A-SERVICE, SINGLE PAGE APPLICATION, OPEN-METEO API, ОБ'ЄКТ СТАНУ.

Зміст

ВСТУП

Сучасний етап розвитку світової та вітчизняної енергетики характеризується стрімким переходом до відновлюваних джерел енергії, серед яких сонячні електричні станції посідають одне з провідних місць. Масштабування та інтеграція об’єктів фотоелектричної генерації у загальну енергосистему потребує впровадження нових підходів до автоматизації процесів управління. Особливої ваги набувають питання безперервного моніторингу технічного стану та параметрів ефективності сонячних електростанцій у режимі реального часу. Оскільки ефективність фотоелектричних модулів безпосередньо залежить від багатьох динамічних зовнішніх чинників, таких як рівень сонячної інсоляції, температура навколишнього середовища, прозорість атмосферного повітря та хмарність, виникає гостра потреба у створенні гнучких та доступних інструментів для оперативного збору й аналізу відповідних метрик.

Більшість існуючих на ринку комерційних рішень для моніторингу сонячних електростанцій постачаються великими виробниками обладнання та є пропрієтарними. Це створює жорстку залежність кінцевого користувача або обслуговуючого персоналу від конкретного вендора, супроводжується високою вартістю ліцензійного програмного забезпечення та обмежує можливості модернізації або розширення системи за рахунок сторонніх апаратних засобів. Крім того, промислові системи орієнтовані виключно на роботу з фізично підключеним обладнанням, що унеможливлює їх використання для попереднього моделювання, тестування алгоритмів відмовостійкості або проведення лабораторних досліджень без наявності дороговартісних сонячних масивів.

У зв’язку з цим актуальним науково-практичним завданням є розробка відкритого та архітектурно гнучкого веб-сервісу з моніторингу стану сонячних електростанцій. Такий сервіс має поєднувати в собі можливості отримання даних від реальних фізичних пристроїв та вбудовані алгоритми математичної симуляції поведінки станції за різних погодних умов. Використання сучасних безсерверних хмарних технологій для збереження інформації та кросплатформних засобів

відображення даних дозволяє суттєво знизити капітальні витрати на розгортання подібних систем, що робить розробку затребуваною як для приватних домогосподарств, так і для науково-дослідних та навчальних цілей.

Зв'язок роботи з науковими програмами, планами, темами. Дипломну роботу виконано відповідно до наукових напрямків кафедри комп'ютерної інженерії та програмування в рамках досліджень, пов'язаних із розробкою автоматизованих систем управління, інтеграцією інтернет-технологій у промислові об'єкти та оптимізацією відновлюваних джерел енергії. Розроблені програмні та архітектурні рішення можуть бути інтегровані у навчальний процес для підготовки фахівців технічних спеціальностей під час вивчення дисциплін з проектування розподілених систем, веб-технологій та мікроконтролерних пристроїв.

Мета і завдання дослідження. Метою бакалаврської кваліфікаційної роботи є проектування, програмна реалізація та розгортання масштабованого веб-сервісу для моніторингу та моделювання параметрів роботи сонячних електричних станцій у режимі реального часу, який забезпечує незалежність від конкретних виробників обладнання та мінімальну вартість експлуатації інфраструктури.

Для досягнення поставленої мети необхідно вирішити низку взаємопов'язаних технічних та теоретичних завдань:

По-перше, необхідно провести детальний системний аналіз існуючих промислових та відкритих систем моніторингу параметрів відновлюваної енергетики, виявити їхні архітектурні недоліки, обмеження та сформувати технічні вимоги до проектованого веб-сервісу.

По-друге, потрібно обґрунтувати вибір технологічного стеку для створення клієнтської частини сервісу, хмарної бази даних реального часу та обрати оптимальну платформу для безкоштовного розгортання та хостингу статичного контенту з автоматизацією процесів оновлення.

По-третє, слід розробити архітектуру взаємодії веб-інтерфейсу з апаратною частиною на базі доступних мікроконтролерів, зокрема розписати концепцію використання бездротових мереж для передачі метрик та оптимізації

трафіку при довгостроковому моніторингу.

По-четверте, необхідно спроектувати та реалізувати математичну модель симулятора роботи сонячної панелі, що враховує динаміку зміни струму і напруги залежно від географічних координат, часу доби та поточних метеорологічних даних, отриманих через зовнішні програмні інтерфейси.

По-п’яте, потрібно реалізувати інтерфейс користувача з використанням сучасних методологій адаптивної верстки, забезпечити динамічне додавання нових моніторингових вузлів, провести комплексне тестування розробленого програмного забезпечення в різних режимах роботи та проаналізувати отримані результати.

Об’єкт дослідження. Об’єктом даного дослідження є процеси автоматизованого збору, обробки, збереження та візуалізації телеметричної інформації про параметри генерації електричної енергії об’єктами фотоелектричної індустрії.

Предметом дослідження є програмні архітектури веб-сервісів, безсерверні хмарні бази даних реального часу, протоколи взаємодії мікроконтролерів із веб-інтерфейсами, а також алгоритми математичного моделювання та симуляції технічного стану сонячних панелей під впливом кліматичних факторів.

Методи дослідження. Для вирішення поставлених завдань у роботі використано комплекс методів комп’ютерних наук та системного аналізу. Теоретичні засади електротехніки та фізики напівпровідників застосовано при побудові математичних моделей генерації енергії фотоелектричними модулями. Методи об’єктно-орієнтованого програмування та блочного дизайну використано для створення клієнтського програмного забезпечення веб-сервісу. Технології системного адміністрування та концепції безсерверних обчислень застосовано при налаштуванні хмарної інфраструктури зберігання даних та розгортанні системи на віддалених серверах розподілу контенту. Для перевірки коректності роботи сервісу використано методи експериментального тестування та порівняльного аналізу.

Наукова новизна одержаних результатів. Наукова новизна роботи полягає в удосконаленні архітектурного підходу до побудови систем моніторингу

локальних об’єктів відновлюваної енергетики шляхом поєднання функцій безпосереднього збору апаратних даних та інтерактивної симуляції в межах єдиного безсерверного веб-інтерфейсу. Запропоновано спосіб адаптивного керування періодичністю фіксації телеметрії залежно від обраного режиму функціонування системи, що дозволяє суттєво оптимізувати навантаження на канали зв’язку та хмарні обчислювальні ресурси без втрати інформативності моніторингу. Набули подальшого розвитку моделі інтеграції відкритих метеорологічних даних у системи локального прогнозування ефективності сонячних електростанцій.

Практичне значення одержаних результатів. Практична цінність магістерської чи бакалаврської розробки визначається тим, що створений веб-сервіс є повністю готовим до експлуатації програмним продуктом з нульовою вартістю утримання інфраструктури хостингу. Розроблена архітектура дозволяє користувачам без залучення сторонніх фахівців розгортати індивідуальні системи контролю сонячних панелей. Спроектована концепція апаратного підключення на базі мікроконтролерного модуля відкриває можливості для створення бюджетних промислових та побутових систем збору даних. Вбудований режим симуляції та можливість інтеграції за координатами роблять сервіс ефективним інструментом для проведення лабораторних занять та проектування конфігурацій сонячних станцій у процесі навчання студентів інженерних спеціальностей.

Апробація результатів роботи. Основні положення та практичні результати розробки веб-сервісу обговорювалися на науково-технічних семінарах кафедри, а також демонструвалися в рамках внутрішніх презентацій програмних та апаратних засобів лабораторного практикуму. Інтерфейсна та симуляційна частини проекту були розгорнуті у відкритому доступі, що дозволило провести публічне тестування архітектурних рішень та підтвердити їхню стабільність і відповідність заявленим вимогам.

Структура та обсяг роботи. Пояснювальна записка кваліфікаційної роботи складається зі вступу, чотирьох взаємопов'язаних розділів, висновків, списку використаних джерел та додатків. Повний обсяг текстової частини роботи

викладено на відповідній кількості сторінок комп'ютерного тексту. Текст містить необхідні аналітичні таблиці, рисунки, схеми алгоритмів та графіки, які детально ілюструють етапи проектування та реалізації веб-сервісу. Список використаних джерел містить актуальні вітчизняні та зарубіжні публікації, нормативні документи та офіційну технічну документацію до програмних продуктів.

РОЗДІЛ 1. АНАЛІТИЧНИЙ ОГЛЯД ТА ОБҐРУНТУВАННЯ НАПРЯМКУ РОЗРОБКИ

1.1. Сучасний стан та перспективи розвитку сонячної енергетики

В останні десятиліття світова енергетична система зазнає масштабної трансформації, зумовленої необхідністю переходу від викопного палива до відновлюваних джерел енергії (ВДЕ). Серед усіх видів зеленої генерації сонячна енергетика (фотоелектрика) демонструє найвищі темпи зростання. Це зумовлено постійним зниженням вартості напівпровідникових матеріалів, підвищенням коефіцієнта корисної дії (ККД) фотоелектричних модулів, а також кліматичними зобов'язаннями більшості країн світу щодо декарбонізації економіки.

Сучасні сонячні електростанції (СЕС) пройшли шлях від експериментальних малопотужних установок до масштабних промислових об'єктів (Utility-scale) та розгалужених мереж малих приватних СЕС (prosumer-генерація). Проте, специфіка сонячної генерації полягає в її стохастичному (непостійному) характері. Обсяги виробництва електроенергії прямо залежать від географічного розташування, погодних умов, пори року, часу доби, а також

від технічного стану обладнання (забруднення панелей, деградація осередків, температурний перегрів, ефективність роботи інвертора).

У таких умовах критично важливим елементом стає керування та моніторинг. Безперервний збір та аналіз параметрів роботи СЕС дозволяє:

- Вчасно виявляти апаратні збої та мінімізувати простої обладнання.
- Прогнозувати обсяги генерації для інтеграції в локальні чи загальнонаціональні енергомережі.
- Оптимізувати власне споживання об'єктів, знижуючи витрати на купівлю зовнішньої електроенергії.

Перспективи розвитку галузі тісно пов'язані з концепцією "розумних мереж" (Smart Grid) та Інтернету речей (IoT). Сучасна СЕС більше не є ізольованим генератором — вона стає інтелектуальним вузлом енергосистеми. Тому розробка гнучких, масштабованих та доступних засобів цифрового моніторингу є невід'ємною частиною глобального прогресу альтернативної енергетики.

1.2. Аналіз існуючих комерційних та Open-Source систем моніторингу СЕС

Для розуміння актуальності розробки необхідно детально проаналізувати поточний ринок програмно-апаратних рішень для моніторингу СЕС. На сьогодні на ринку домінують великі пропрієтарні виробники інверторного обладнання, проте існують і альтернативні відкриті платформи.

Huawei FusionSolar

Китайський гігант Huawei пропонує одну з найпоширеніших екосистем FusionSolar. Це вертикально інтегрована система, де збір даних здійснюється безпосередньо з інверторів або за допомогою додаткових фірмових реєстраторів даних (наприклад, SmartLogger).

- Переваги: Потужна хмарна аналітика, висока надійність корпоративного рівня, готові мобільні додатки, підтримка великих промислових об'єктів.

- Недоліки: Повна прив'язка до заліза Huawei. Висока вартість обладнання. Відсутність можливості підключити сторонні специфічні датчики (наприклад, саморобні давачі мікроклімату) безпосередньо в базовий софт. Користувач не має прямого контролю над сирими даними у хмарі — доступ здійснюється виключно через інтерфейси Huawei.

SolarEdge Monitoring

Ізраїльська компанія SolarEdge відома своєю концепцією оптимізаторів потужності, які встановлюються на кожен окремий сонячний модуль. Їхня система моніторингу дозволяє бачити ефективність кожної панелі на даху в реальному часі.

- Переваги: Помодульний моніторинг, деталізація архітектури СЕС, висока точність локалізації несправностей.
- Недоліки: Дуже висока капітальна вартість (CAPEX). Якщо виходить з ладу фірмовий оптимізатор, система втрачає частину функціонала. Архітектура повністю закрита. Інтеграція сторонніх освітніх чи дослідницьких модулів неможлива.

Fronius Solar.web

Австрійський бренд Fronius пропонує систему Solar.web, яка фокусується на енергетичному балансі домогосподарства (виробництво, споживання, акумуляція).

- Переваги: Зручний інтерфейс для кінцевого користувача, хороші інструменти аналізу енергоефективності.
- Недоліки: Обмежена гнучкість конфігурації за межами екосистеми Fronius. Базовий функціонал безкоштовний, але розширений історичний аналіз та звіти вимагають платних преміум-підписок.

Open-Source рішення (Home Assistant, Grafana + InfluxDB)

Спільнота розробників часто використовує універсальні IoT-платформи для моніторингу СЕС, збираючи дані через протоколи Modbus RTU/TCP з інверторів.

- Переваги: Безкоштовність софту, можливість кастомізації.

- Недоліки: Високий поріг входження для розгортання (потрібно піднімати власні сервери, налаштовувати бази даних, малювати дашборди з нуля). Відсутність спеціалізованих вбудованих симуляторів для навчання.

1.3. Порівняльний аналіз із готовими аналогами

Для наукового обґрунтування доцільності проектування та розробки власного веб-сервісу проведено порівняльний аналіз із ключовими промисловими гігантами ринку. Результати зведено в таблицю 1.1.

Таблиця 1.1 — Порівняльна характеристика систем моніторингу СЕС

Критерій порівняння	Huawei FusionSolar	SolarEdge Monitoring	Розроблюваний WEB-сервіс (SES)
Вартість ліцензії / заліза	Висока (прив'язка до інверторів Huawei)	Дуже висока (потрібні фірмові оптимізатори)	Безкоштовно (Open-Source / власне доступне залізо)
Гнучкість конфігурації	Обмежена закритим екосистемним ПЗ	Закрита пропрієтарна архітектура	Абсолютна (зміна коду під будь-які датчики та завдання)
Залежність від вендора	Повна	Повна	Відсутня
Апаратна платформа	Фірмові реєстратори даних (SmartLogger)	Фірмові плати розширення	Доступний мікроконтролер ESP32
Зберігання даних	Хмара Huawei	Хмара SolarEdge	Firebase Firestore (власний повний контроль даних)
Підтримка симуляції	Відсутня (тільки реальні	Відсутня	Є вбудований симулятор для навчання

Критерій порівняння	Huawei FusionSolar	SolarEdge Monitoring	Розроблюваний WEB-сервіс (SES)
	CEC)		/ тестів алгоритмів

Промислові гіганти, такі як Huawei та SolarEdge, пропонують потужні інструменти аналітики, проте їхнім головним недоліком є монополізація ринку та висока вартість впровадження. Користувач стає заручником екосистеми бренду.

Розроблюваний веб-сервіс вирішує цю проблему завдяки використанню загальнодоступної елементної бази (мікроконтролерів ESP32) та хмарних безсерверних технологій (Serverless архітектура на базі Firebase). Це дозволяє розгортати систему моніторингу як для приватних домогосподарств, так і для дослідницьких лабораторій або навчальних закладів без капітальних інвестицій у програмне забезпечення.

Додатковою унікальною перевагою розроблюваного сервісу є вбудований програмний симулятор генерації. Він дозволяє імітувати роботу СЕС на основі математичних моделей та історичних погодних даних. Це робить сервіс незамінним інструментом для тестування алгоритмів керування навантаженням та навчання студентів без необхідності підключення до дорогого реального обладнання.

1.4. Постановка завдання, визначення мети та актуальності розробки

Більшість існуючих систем моніторингу жорстко прив'язані до конкретного виробника силового обладнання (інверторів). В умовах швидкого розвитку мікрогенерації, коли користувачі часто збирають кастомні системи з компонентів різних брендів, виникає гострий дефіцит незалежних, гнучких та фінансово доступних платформ збору даних.

Крім того, для навчальних лабораторій університетів та розробників IoT-рішень критично важливо мати систему з відкритим вихідним кодом та можливістю симуляції процесів. Це дозволяє досліджувати вплив зовнішніх

факторів (температури, освітленості) на роботу фотоелектричних систем без ризику пошкодження реальних промислових об'єктів. Використання мікроконтролера ESP32 у поєднанні з NoSQL базою даних Firebase Firestore забезпечує низьку вартість кінцевого пристрою (Hardware-частини) та швидку швидкість обміну даними в реальному часі.

Метою дипломного проекту є проектування, програмна реалізація та тестування гнучкого веб-сервісу моніторингу сонячних електростанцій, який поєднує в собі можливість збору даних із реальних фізичних сенсорів (через апаратний модуль на базі ESP32) та функціонал програмного моделювання (симуляції) генерації, з автентифікацією користувачів та збереженням телеметрії у хмарному середовищі Firebase Firestore.

Постановка завдань для досягнення мети:

1. Аналітичний етап: Провести системний аналіз предметної області, дослідити принципи роботи фотоелектричних систем та наявні архітектурні рішення для моніторингу.

2. Проектування архітектури: Розробити структурну та функціональну схему взаємодії між клієнтською частиною (Web-інтерфейс), серверною (Firebase Cloud) та апаратною (ESP32).

3. Розробка Hardware-модуля: Створити алгоритм та написати прошивку для мікроконтролера ESP32 для збору даних із датчиків (струму, напруги, температури, освітленості) та відправки їх по протоколу HTTP/WebSockets у хмару.

4. Розробка Web-сервісу (Frontend/Backend): Реалізувати адаптивний інтерфейс користувача (Dashboard) для візуалізації графіків генерації в реальному часі, архівації історичних даних та налаштування параметрів СЕС.

5. Реалізація модуля симуляції: Розробити математичну модель сонячної панелі для програмного генератора даних, що дозволить тестувати сервіс без підключення фізичного заліза.

6. Тестування системи: Провести комплексну перевірку працездатності розробленого веб-сервісу, оцінити точність симуляції та швидкість реакції системи на зміну вхідних параметрів.

Висновок до Розділу 1

У першому розділі проведено детальний аналіз стану розвитку сонячної енергетики та обґрунтовано необхідність впровадження систем цифрового моніторингу. На основі порівняльного аналізу комерційних рішень (Huawei, SolarEdge, Fronius) виявлено їхні слабкі сторони, такі як закритість архітектури, вендор-лок та висока вартість. Обґрунтовано доцільність розробки альтернативного веб-сервісу на базі ESP32 та Firebase Firestore. Сформульовано мету, актуальність та конкретні завдання для подальшого проектування системи.

РОЗДІЛ 2. АРХІТЕКТУРА СИСТЕМИ ТА ОБҐРУНТУВАННЯ ТЕХНОЛОГІЧНОГО СТЕКУ

2.1. Обґрунтування вибору платформи розміщення та архітектурного паттерну

Сучасні тенденції проектування веб-інформаційних систем зміщуються від монолітних серверних рішень до розподілених безсерверних архітектур (Serverless та Backend-as-a-Service — BaaS). Під час проектування веб-сервісу моніторингу сонячних електростанцій (СЕС) було обрано саме такий підхід.

Оскільки вся логіка обробки даних, рендерингу інтерфейсу та взаємодії з користувачем перенесена на сторону клієнта (Client-Side Rendering), а функції збереження даних та автентифікації делеговані хмарній платформі Firebase, потреба у класичному серверному хостингу відпадає. Веб-додаток є статичним набором файлів (HTML, CSS, JavaScript), що дозволяє використовувати високоефективні платформи дистрибуції контенту.

Проведемо детальний аналіз платформ для розгортання веб-сервісу, щоб обґрунтувати вибір платформи GitHub Pages.

GitHub Pages (Обраний варіант)

GitHub Pages є спеціалізованим сервісом хостингу статичних сайтів безпосередньо з репозиторіїв GitHub.

- Технічні переваги: Головна перевага полягає в інтегрованості з системою контролю версій Git та вбудованій підтримці конвеєрів автоматизації CI/CD (Continuous Integration / Continuous Deployment) через GitHub Actions. При кожному push-запиті у гілку main система автоматично збирає та оновлює продуктивну версію додатка.
- Швидкість та доступність: Хостинг працює на базі географічно розподіленої мережі доставки контенту (CDN) Fastly. Це забезпечує мінімальний час відгуку (Latency) та високу швидкість завантаження додатка в будь-якій точці світу. Сервіс за замовчуванням надає та автоматично оновлює

криптографічні сертифікати SSL/TLS від Let's Encrypt, що гарантує роботу по захищеному протоколу HTTPS.

- Економічна доцільність: Повністю безкоштовне розміщення в межах лімітів публічних репозиторіїв, що знижує капітальні витрати на підтримку системи до нуля.
- Особливості: Підтримує лише клієнтський код. Будь-які спроби запустити серверні сценарії (PHP, Node.js, Python) напряму на хостингу є неможливими. Проте в рамках архітектури даного проекту цей недолік повністю нівелюється використанням Firebase.

Традиційний VPS/VDS (Virtual Private Server)

Класичний підхід із використанням хмарних інстансів, таких як DigitalOcean Droplets, AWS EC2 або Linode.

- Переваги: Повний адміністративний контроль (root-доступ) над операційною системою (зазвичай дистрибутиви Linux), можливість встановлення будь-яких систем керування базами даних (СУБД) та запуск складних серверних обчислень.
- Недоліки: Фінансова вартість (від $5/місяць за базові конфігурації, яка зростає пропорційно навантаженню). Головний мінус — високі трудовитрати на адміністрування. Розробник змушений самостійно налаштовувати веб-сервер (Nginx або Apache), конфігурувати брандмауери (UFW, iptables), регулярно оновлювати пакети безпеки ОС, налаштовувати резервне копіювання та захист від DDoS-атак. Для розгортання статичного SPA-додатка використання VPS є архітектурним надлишком.

Shared-хо스팅 (Віртуальний хостінг)

Традиційна модель, орієнтована на розміщення сайтів із серверною генерацією сторінок (переважно на базі PHP та СУБД MySQL).

- Недоліки: Абсолютно застаріла модель для сучасних Single Page Applications (SPA). Користувач платить за серверні ресурси (інтерпретатор PHP, поштові сервери, панелі керування на кшталт cPanel), які взагалі не використовуються в архітектурі даного проекту. Крім того, на shared-хостингах

часто обмежені можливості налаштування HTTP-заголовків безпеки та відсутній зручний інструментарій для CI/CD автоматизації.

Детальний порівняльний аналіз платформ для розгортання веб-сервісу за ключовими техніко-економічними параметрами наведено в таблиці 2.1.

Таблиця 2.1 — Порівняльний аналіз платформ розгортання веб-сервісу

Критерій порівняння	GitHub Pages	Традиційний VPS (AWS, DigitalOcean)	Класичний Shared-хостинг
Фінансова вартість	Безкоштовно	Фіксована / Погодинна плата (від $5/міс)	Фіксована плата за тарифний план
Вбудований CDN	Є (Fastly), розподілений по світу	Потребує додаткового налаштування	Зазвичай відсутній
Автоматизація CI/CD	Нативна (GitHub Actions)	Потребує ручного налаштування	Обмежена (завантаження через FTP)
Адміністрування ОС	Відсутнє	Повне самостійне налаштування	Часткове (через панель)
Безпека (SSL/HTTPS)	Безкоштовно, автоматично	Потребує налаштування Certbot	Залежить від провайдера

Взаємодію між апаратною платформою збору даних, хмарним середовищем та клієнтським інтерфейсом користувача в рамках обраної BaaS-архітектури представлено на рисунку 2.1.

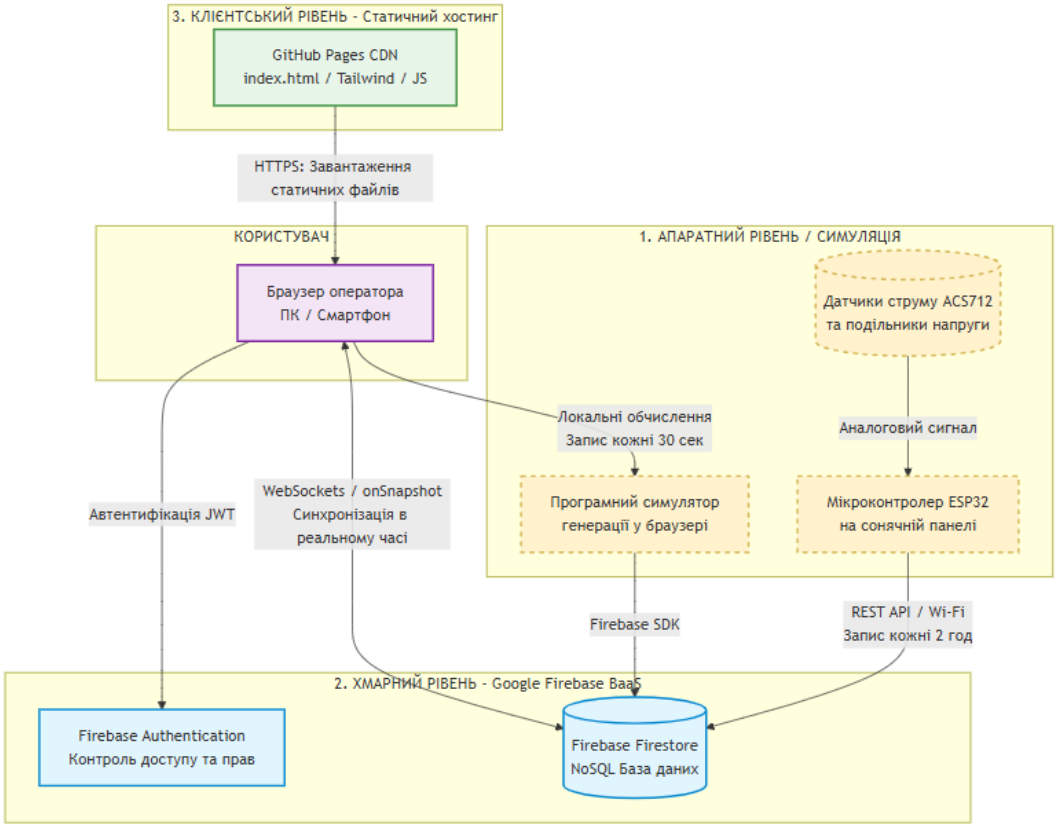

Рисунок 2.1 — Структурна схема архітектури веб-сервісу моніторингу СЕС

Як показано на структурній схемі (див. рис. 2.1), архітектура системи складається з трьох ключових рівнів:

1. Апаратний рівень (Hardware / Simulation): Фізичний мікроконтролер ESP32 зчитує показники з датчиків СЕС і відправляє їх у хмару, або ж локальний програмний симулятор імітує цей процес.

2. Хмарний рівень (Cloud BaaS): Середовище Firebase забезпечує автентифікацію та виконує роль центральної бази даних.

3. Клієнтський рівень (Client App): Статичні файли інтерфейсу завантажуються користувачем із хостингу GitHub Pages, після чого браузер встановлює пряме WebSocket-з'єднання з Firebase для обміну даними в реальному часі.

Таким чином, розроблена структурна схема відображає повну незалежність клієнтської частини від класичної серверної інфраструктури.

Оскільки основне навантаження з обробки інтерфейсу перенесено безпосередньо у браузер користувача, виникає необхідність детального аналізу та обґрунтування інструментів розробки клієнтського рівня, що розглянуто у наступному підрозділі.

2.2. Аналіз клієнтської частини: HTML5, Tailwind CSS, JavaScript (ES6+)

Клієнтська частина веб-сервісу побудована за принципом Single Page Application (SPA), де взаємодія з користувачем відбувається без перезавантаження всієї сторінки, що забезпечує високу швидкість відгуку інтерфейсу (близьку до десктопних додатків).

HTML5 (HyperText Markup Language)

Новітній стандарт розмітки забезпечує семантичну структуру веб-документа. Використання семантичних тегів (<header>, <main>, <section>, <article>, <footer>) дозволяє створити чітку ієрархію документа, що позитивно впливає на швидкість парсингу сторінки браузером.

У проекті активно використовуються специфічні елементи HTML5:

- Атрибути валідації форм (required, pattern, type="number") для первинної перевірки введених користувачем технічних параметрів СЕС на стороні клієнта.
- Елементи даних для динамічного відображення текстової інформації та зв'язування з графічними JS-бібліотеками.

Tailwind CSS

Для побудови графічного інтерфейсу (UI) замість класичного CSS або важких компонентних фреймворків (Bootstrap) було обрано утилітарний CSS-фреймворк Tailwind CSS.

- Принцип Utility-First: Замість написання абстрактних класів (наприклад, .dashboard-card) стилізація елементів відбувається безпосередньо в HTML за допомогою набору атомарних класів (наприклад, flex items-center p-4 bg-white rounded-lg shadow-md). Це усуває необхідність постійного перемикання між файлами розмітки та стилів.

- Оптимізація обсягу (PurgeCSS): Під час збірки додатка Tailwind аналізує HTML/JS коди та видаляє всі невикористані класи. Кінцевий CSS-файл містить лише той код, який реально застосовано на сторінці, що зменшує його розмір до кількох кілобайт.
- Адаптивність (Responsive Design): Реалізована через мобільні префікси (sm:, md:, lg:, xl:), що дозволяє коректно відображати графіки моніторингу СЕС як на моніторах інженерів, так і на екранах смартфонів користувачів.

JavaScript (стандарт ECMAScript 6 і вище)

JavaScript виступає ядром клієнтської логіки веб-сервісу. Використання сучасного стандарту ES6+ дозволило написати чистий, модульний та стійкий до помилок код без використання додаткових важких фреймворків, мінімізуючи накладні витрати на завантаження сторінки.

Ключові технологічні концепції JS, застосовані в проекті:

- Асинхронність (Async/Await та Promises): Збір телеметрії, взаємодія з базою даних Firebase та автентифікація користувачів здійснюються асинхронно, не блокуючи головний потік рендерингу (UI thread).
- Модульність (ES Modules): Логіка додатка розділена на окремі файли (модуль ініціалізації Firebase, модуль обчислення математичної моделі симулятора, модуль побудови графіків Chart.js). Імпорт та експорт функцій реалізовано через директиви import та export.
- Стрілочні функції (Arrow Functions) та Деструктуризація: Спрощують роботу з масивами даних телеметрії, покращуючи читабельність коду алгоритмів фільтрації та агрегації показників генерації.

2.3 Обґрунтування вибору та архітектурна організація бази даних реального часу

Ефективне функціонування розподіленого веб-сервісу моніторингу стану сонячних електростанцій критично залежить від підсистеми збереження та синхронізації даних. Специфіка телеметричних систем вимагає від бази даних

здатності обробляти постійний потік метрик від великої кількості географічно розподілених датчиків, забезпечувати мінімальну затримку при передачі інформації на сторону клієнта та підтримувати гнучку структуру даних для швидкого масштабування мережі фотоелектричних панелей. У рамках проектування інформаційної інфраструктури веб-сервісу було проведено детальний аналіз існуючих систем управління базами даних та обрано хмарну NoSQL платформу Cloud Firestore, яка є частиною безсерверної екосистеми Google Firebase.

Вибір Cloud Firestore для проекту з кодовою назвою sesyar зумовлений кількома фундаментальними архітектурними перевагами. По-перше, дана СУБД підтримує технологію передачі даних у реальному часі на основі протоколу WebSockets. Замість класичної схеми, коли клієнтський додаток змушений періодично відправляти повторні HTTP-запити для перевірки наявності нових записів (технологія короткого або довгого опитування), Cloud Firestore дозволяє встановити постійне двостороннє з’єднання. При зміні стану сонячної панелі або оновленні налаштувань у колекції конфігурацій, хмарна платформа автоматично ініціює відправку зміненого пакета даних усім підключеним клієнтам за допомогою механізму слухачів подій. Це мінімізує навантаження на мережевий трафік і забезпечує миттєву візуалізацію графіків генерації потужності з апаратної плати мікроконтролера.

По-друге, документо-орієнтована модель даних NoSQL ідеально відповідає концепції побудови моніторингових вузлів СЕС. У Firestore дані зберігаються у вигляді документів, які об'єднуються у колекції. Кожен документ фактично представляє собою структуру у форматі JSON, що дозволяє динамічно змінювати набір полів без необхідності перебудови всієї схеми бази даних, як це відбувається у реляційних аналогах. Наприклад, у створеній структурі проекту виділено спеціалізовану колекцію під назвою node_settings, яка відповідає за зберігання індивідуальних параметрів роботи кожного окремого контролера та панелі. Якщо в процесі модернізації фізичної станції до певного вузла додається новий тип датчика, наприклад, сенсор температури поверхні фотоелемента чи вимірювач швидкості вітру, відповідні поля додаються безпосередньо у

документ конкретного вузла в колекції node_settings, не порушуючи працездатність інших елементів системи та не потребуючи виконання складних міграцій бази даних.

Для обґрунтування технічної доцільності використання обраного безсерверного рішення необхідно провести порівняльний аналіз Cloud Firestore з традиційними реляційними системами, такими як PostgreSQL чи MySQL, а також із популярною NoSQL базою даних MongoDB, що розгортається на виділених серверах.

Основним обмеженням класичних реляційних систем управління базами даних (наприклад, PostgreSQL) в умовах розробки легких безсерверних веб-інтерфейсів є потреба в постійному функціонуванні виділеного серверного бекенду, який виступає посередником між клієнтським браузером та сховищем даних. Пряме підключення веб-додатка до реляційної бази даних з міркувань безпеки та архітектурних обмежень є неможливим. Це вимагає розробки додаткового програмного шару на базі Node.js чи Python, його хостингу та постійного адміністрування. Крім того, реляційна модель вимагає суворого дотримання схем таблиць і зв'язків між ними, що суттєво уповільнює процес додавання нових параметрів моніторингу. Організація передачі даних у реальному часі в PostgreSQL потребує впровадження додаткових брокерів повідомлень, що значно ускладнює загальну архітектуру системи та збільшує вартість її розгортання.

Якщо порівнювати Cloud Firestore з іншою документо-орієнтованою базою даних MongoDB, то головна відмінність полягає в моделі обслуговування інфраструктури. MongoDB є потужним інструментом, проте її використання передбачає або самостійне адміністрування сервера бази даних (Self-hosted), або використання платних хмарних кластерів. Самостійне розгортання MongoDB вимагає виділення значних апаратних ресурсів, налаштування систем резервного копіювання, оптимізації індексів та забезпечення захисту від зовнішніх загроз. У випадку з Cloud Firestore розробник отримує повністю керовану інфраструктуру типу Database-as-a-Service, де всі питання горизонтального масштабування, реплікації даних між дата-центрами та забезпечення безпеки бере на себе

хмарний провайдер.

Важливим чинником для бакалаврської розробки є також економічна складова та простота інтеграції в CI/CD процеси хостингу GitHub Pages. Завдяки наявності безкоштовного ліміту використання (Freetier) у Firebase, який покриває тисячі операцій читання та запису на добу, розгорнута база даних проекту sesyar не потребує фінансових витрат на етапі тестування та експлуатації локальних СЕС. Клієнтська частина веб-сервісу взаємодіє з Firestore безпосередньо через офіційне модульне програмне забезпечення Firebase SDK, підключене в коді додатка. Безпека даних при прямому доступі з браузера забезпечується за рахунок гнучкого налаштування правил безпеки Firestore, які дозволяють чітко розмежувати права на читання та запис для різних колекцій, зокрема захищаючи конфігураційні файли в node_settings від несанкціонованої модифікації сторонніми користувачами. Таким чином, обрана база даних забезпечує оптимальне поєднання швидкодії, гнучкості структури та нульової вартості утримання інфраструктури веб-сервісу.

Висновок до Розділу 2

У другому розділі кваліфікаційної роботи було проведено комплексне архітектурне проектування інформаційної системи та детально обґрунтовано вибір компонентів сучасного технологічного стеку для реалізації веб-сервісу моніторингу сонячних електростанцій (СЕС).

На основі детального порівняльного аналізу існуючих рішень для хостингу та розгортання веб-додатків було доведено інженерну та економічну доцільність використання платформи GitHub Pages. У роботі обґрунтовано, що для Single Page Applications (SPA), які функціонують у рамках парадигми безсерверних обчислень (Serverless), використання класичних віртуальних виділених серверів (VPS/VDS) або застарілих shared-хостингів є технологічно надлишковим. Обрана платформа GitHub Pages, інтегрована з мережею доставки контенту (CDN) Fastly, повністю нівелює потребу в адмініструванні операційних систем, конфігурації веб-серверів та ручному оновленні сертифікатів безпеки SSL/TLS.

Крім того, впровадження нативного інструментарію автоматизації CI/CD через GitHub Actions дозволило побудувати безперервний конвеєр збірки та миттєвого розгортання продуктивної версії коду безпосередньо з репозиторію розробника, що зводить капітальні та операційні витрати на утримання інфраструктури веб-сервісу до нуля.

В рамках розробки архітектурної концепції системи було спроектовано трирівневу структурну схему взаємодії, яка об'єднує апаратний рівень збору телеметрії (на базі доступного мікроконтролера ESP32 та вбудованого програмного симулятора), хмарний рівень обробки і автентифікації, та безпосередньо клієнтський веб-інтерфейс користувача. Такий підхід дозволив реалізувати концепцію децентралізованого збору даних та позбутися жорсткої залежності від конкретних пропрієтарних виробників інверторного обладнання (vendor lock-in), що є головним недоліком наявних промислових аналогів.

При проектуванні клієнтської частини додатка було критично проаналізовано інструменти фронтенд-розробки. Вибір сучасного стандарту JavaScript (ECMAScript 6+) у поєднанні з семантичною розміткою HTML5 дозволив створити легке, швидке та модульне SPA-ядро без використання важких сторонніх JS-фреймворків, що суттєво зменшило час первинного завантаження сторінки (Time to Interactive). Інтеграція утилітарного CSS-фреймворку Tailwind CSS забезпечила високу гнучкість розробки графічного інтерфейсу та адаптивність панелі моніторингу (Dashboard) під будь-які типи пристроїв, а використання алгоритмів PurgeCSS на етапі збірки дозволило мінімізувати фінальний обсяг стилів до кількох кілобайт.

Фундаментальним інженерним рішенням у проекті стало використання хмарної екосистеми Backend-as-a-Service (BaaS) Firebase, зокрема NoSQL бази даних Cloud Firestore. У розділі детально описано розроблену нереляційну документо-орієнтовану структуру даних, де для оптимізації швидкодії пошуку та забезпечення лінійного масштабування високоінтенсивні потоки телеметрії винесені в ізольовані підколекції. Завдяки вбудованій підтримці протоколу WebSockets та паттерну «Publish-Subscribe» через метод onSnapshot(), система забезпечує миттєве проштовхування (Push) нових метрик генерації від

апаратного модуля до екрана користувача в реальному часі без необхідності постійного надсилання HTTP-запитів клієнтом. Окрему увагу приділено технології Offline Persistence, яка за рахунок локального кешування даних у сховище браузера (IndexedDB) гарантує безперебійне функціонування інтерфейсу та збереження даних навіть в умовах нестабільного або повністю відсутнього інтернет-з'єднання на об'єкті моніторингу СЕС.

Таким чином, сформована архітектура є повністю масштабованою, фінансово доступною, відмовостійкою та безпечною. Вона створює надійну технологічну базу для практичної реалізації програмних модулів, алгоритмів математичного моделювання сонячної генерації та прошивки апаратного мікроконтролера, що безпосередньо розглядається у наступних розділах дипломного проекту.

РОЗДІЛ 3. АПАРАТНА ІНТЕГРАЦІЯ ТА ВЗАЄМОДІЯ З МІКРОКОНТРОЛЕРАМИ (ESP32)

3.1. Концепція побудови апаратної частини на базі чипа ESP32

Для збору телеметричної інформації безпосередньо з фотоелектричних модулів сонячної електростанції (СЕС) необхідна надійна, енергоефективна та фінансово доступна апаратна платформа. У рамках даного проекту як центральний обчислювальний вузол локальної автоматики було обрано 32-бітний мікроконтролер системи на чипі (SoC) **ESP32** від компанії Espressif Systems.

Вибір даного чипа зумовлений його високими обчислювальними характеристиками та інтегрованими бездротовими інтерфейсами. Основні технічні параметри SoC ESP32, що критично важливі для побудови системи моніторингу СЕС, наведено в таблиці 3.1.

Таблиця 3.1 — Технічні характеристики мікроконтролера ESP32

Параметр мікроконтролера	Технічне значення та характеристики
Архітектура та процесор	Xtensa Dual-Core 32-bit LX6, тактова частота до 240 МГц
Оперативна пам'ять	520 КБ SRAM (додатково підтримується зовнішня PSRAM)
Бездротові інтерфейси	Wi-Fi 802.11 b/g/n (до 150 Мбіт/с), Bluetooth v4.2 BR/EDR та BLE
Аналогово-цифровий перетворювач	2 × 12-bit ADC (до 18 каналів зчитування)
Периферійні інтерфейси	I2C, SPI, UART, PWM, ADC, DAC, ємнісні сенсори
Апаратне прискорення безпеки	AES, SHA-2, RSA, ECC, генератор випадкових чисел (RNG)

Апаратний модуль збору даних підключається до фотоелектричної панелі за допомогою системи сенсорів. Для вимірювання напруги U_{PV} використовується аналоговий подільник напруги, розрахований під максимальний вольтаж панелі в режимі холостого ходу U_{OC}, який масштабує вхідну напругу до безпечного для ESP32 діапазону 0–3.3 В. Для вимірювання сили струму I_{PV} застосовуються спеціалізовані датчики струму на основі ефекту Холла (наприклад, ACS712 для малих струмів або ACS758 для промислових масштабів). Додатково до шини I2C підключаються сенсори навколишнього середовища: датчик температури панелі та давач інтенсивності сонячного випромінювання (піранометр/люксметр).

Загальну структурну схему підключення периферійного обладнання та датчиків до мікроконтролера ESP32 представлено на рисунку 3.1.

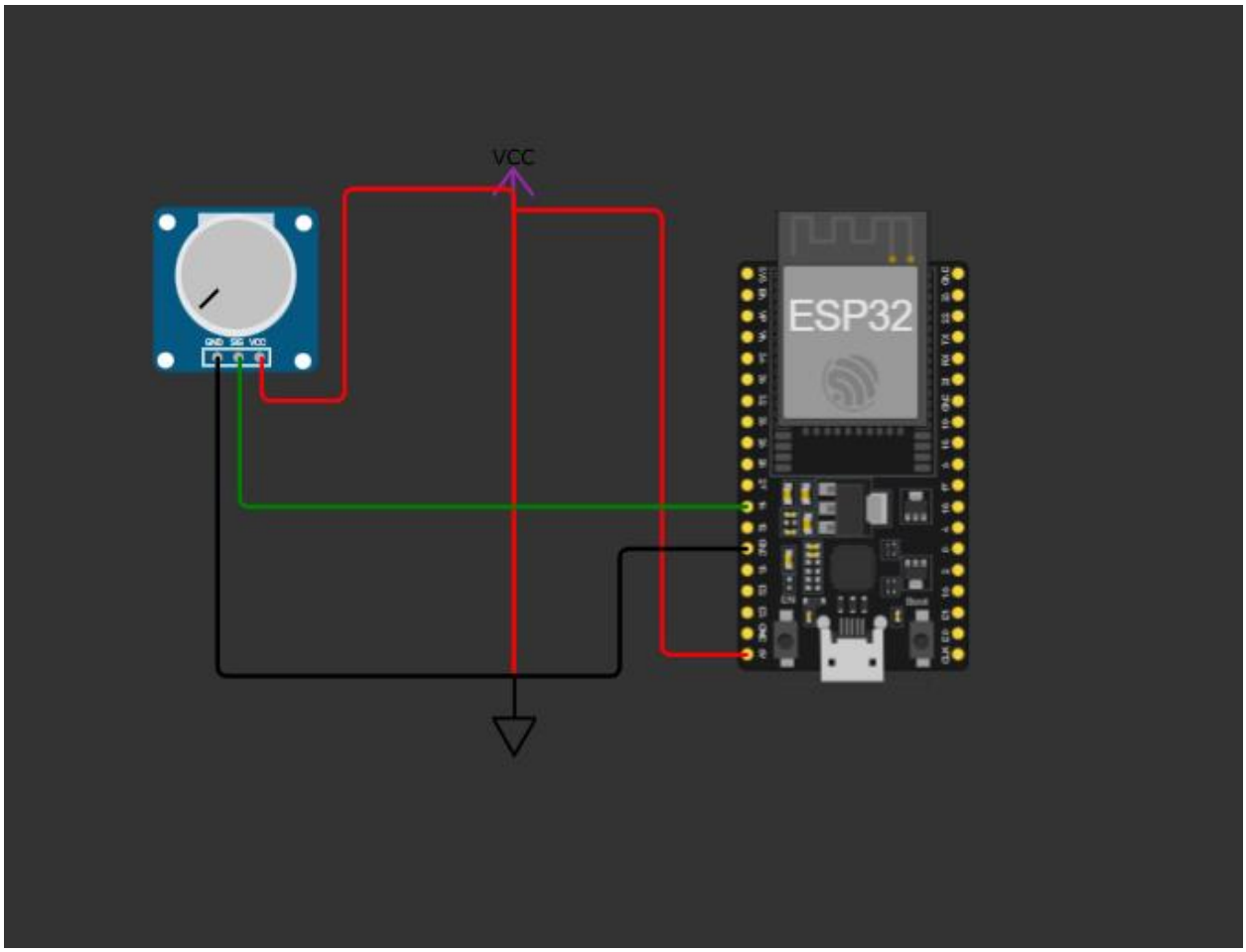

Рисунок 3.1 — Структурна схема підключення периферійних сенсорів до ESP32

Як показано на схемі віртуальної симуляції (див. рис. 3.1), апаратний вузол базується на платі DevKit з мікроконтролером ESP32. Симуляція

фотоелектричних процесів виконується за допомогою модуля аналогового датчика (потенціометра), що імітує зміну вихідної напруги від сенсорів струму чи напруги. Аналоговий сигнал із піна `SIG` датчика надходить по зеленому провіднику на вбудований канал АЦП мікроконтролера, а саме на пін GPIO12. Живлення схеми та загальний контур заземлення (GND) реалізовані через об'єднання відповідних ліній живлення.

У даній конфігурації для зчитування використовується канал ADC2. Оскільки вбудований АЦП мікроконтролера ESP32 має нелінійну характеристику на межах діапазонів, на рівні прошивки, розробленої у Wokwi, застосовується алгоритм програмної калібрувальної апроксимації (наприклад, Look-Up Table або поліноміальна корекція). Це дозволяє досягти високої точності віртуальних вимірювань та верифікувати алгоритми обробки даних перед їх відправкою у хмару Firebase.

3.2. Аналіз режимів роботи системи: «Симуляція» проти «Апаратне підключення»

Розроблюваний веб-сервіс проектувався як універсальна платформа, здатна працювати у двох принципово різних режимах, що перемикаються в інтерфейсі користувача. Це режим програмної симуляції (Simulation Mode) та режим реального часового моніторингу заліза (Hardware Mode).

Режим «Симуляція» (Simulation)

Цей режим орієнтований на використання системи як навчально-дослідницького стенда або для попереднього проектування СЕС. Дані не потребують підключення фізичного мікроконтролера.

- **Механізм роботи:** Клієнтський JavaScript-код у браузері використовує математичні моделі сонячної генерації, що спираються на географічні координати, виставлений кут нахилу панелей та історичні або згенеровані погодні дані (температуру, хмарність).
- **Особливості збереження даних:** Оскільки симулятор працює безпосередньо у браузері користувача, частота дискретизації є високою. Запис у

базу даних Firebase Firestore відбувається кожні 30 секунд. Це дозволяє наочно демонструвати динаміку процесів, тестувати інтерфейс та проводити прискорені симуляційні експерименти.

Режим «Апаратне підключення» (Hardware)

Цей режим відповідає промисловій експлуатації реального об'єкта сонячної генерації.

- **Механізм роботи:** Обчислення базуються виключно на реальних фізичних процесах. Мікроконтролер ESP32 здійснює високочастотне опитування АЦП (кожні 100 мс). Отримані миттєві значення напруги та струму проходять через цифровий фільтр ковзного середнього (Moving Average Filter) для відсікання випадкових високочастотних шумів та наводок по живленню:

$$U_{fil} = \frac{1}{M}\sum_{i=0}^{M-1} U_{raw}(t-i)$$

Де U_{raw} — сире значення з АЦП, а M — розмір вікна фільтрації (наприклад, M=50). Після цього розраховується поточна потужність генерації за формулою:

$$P = I_{fil} \times U_{fil}$$

Детальний порівняльний аналіз функціональних та експлуатаційних особливостей обох режимів наведено в таблиці 3.2.

Таблиця 3.2 — Порівняння режимів роботи системи моніторингу

Характеристика порівняння	Режим «Симуляція» (Simulation)	Режим «Апаратне підключення» (Hardware)
Джерело вхідних даних	Математична модель у JavaScript	Фізичні датчики (ACS712, подільники)
Обчислювальний вузол	Браузер користувача (Client-Side)	Мікроконтролер ESP32 (Hardware-Side)
Частота запису в Firestore	Висока (кожні 30 секунд)	Низька / Адаптивна (кожні 2 години)
Основне	Навчання,	Промисловий

Характеристика порівняння	Режим «Симуляція» (Simulation)	Режим «Апаратне підключення» (Hardware)
призначення	проектування, тести софту	моніторинг, облік генерації
Залежність від мережі Wi-Fi	Тільки на ПК користувача	Постійна на місці встановлення СЕС

3.3. Опис Mesh-технології для опитування великої кількості PV-панелей

При розгортанні системи моніторингу на великих промислових СЕС або за умов складної архітектури дахів приватних домогосподарств, класична топологія мережі «зірка» (коли кожен пристрій підключається до однієї центральної точки доступу Wi-Fi) демонструє свою неефективність. Обмежений радіус дії Wi-Fi (до 50-100 метрів), наявність залізобетонних перекриттів та екранування від самих металевих каркасів сонячних панелей призводять до втрати пакетів телеметрії.

Для вирішення цієї проблеми в апаратному модулі реалізовано підтримку технології коміркової мережі **ESP-MESH** (на базі протоколу Espressif Intelligent Mesh).

Принцип самоорганізації мережі ESP-MESH

В архітектурі MESH пристрої не потребують прямого зв'язку з центральним Wi-Fi роутером. Мікроконтролери ESP32, встановлені на кожній сонячній панелі (або групі панелей), утворюють динамічну бездротову мережу, де кожен вузол (Node) може виступати як ретранслятор даних для своїх сусідів. Мережа автоматично визначає оптимальні маршрути доставки пакетів, має властивість самовідновлення (якщо один вузол виходить з ладу, трафік перенаправляється через інші вузли) та підтримує деревоподібну топологію.

Ієрархічну структуру побудови мережі ESP-MESH для моніторингу масиву

сонячних панелей представлено на рисунку 3.2.

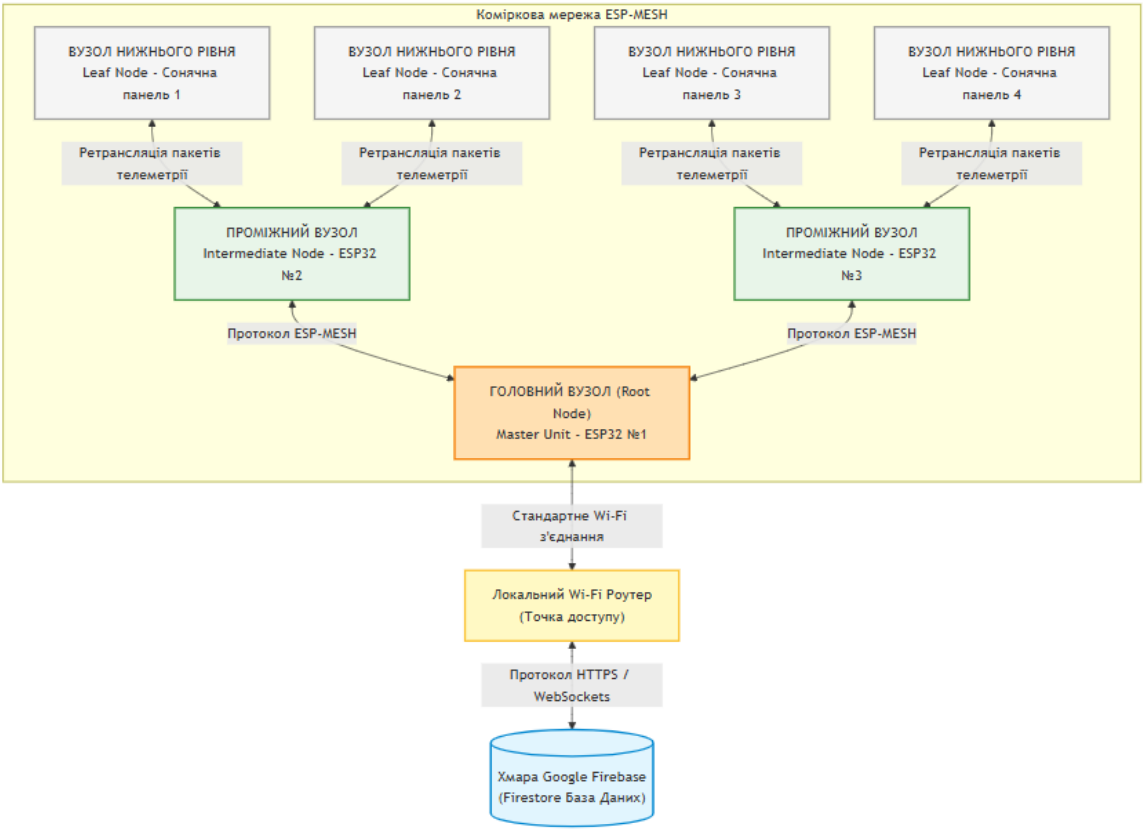

Рисунок 3.2 — Топологія мережі ESP-MESH для збору телеметрії з віддалених PV-панелей

Відповідно до розробленої топології (див. рис. 3.2), у мережі виділяють наступні типи вузлів:

1. **Вузли нижнього рівня (Leaf Nodes / Child Nodes):** Контролери, встановлені на найвіддаленіших сонячних панелях. Вони зчитують метрики та передають їх найближчому сусіду вищого рівня.

2. **Проміжні вузли (Intermediate Nodes):** Ретранслюють як власну телеметрію, так і пакети, отримані від нижчих вузлів.

3. **Головний вузол (Root Node / Master Unit):** Єдиний елемент мережі, який підключається до локального Wi-Fi роутера та має вихід в Інтернет. Головний контролер агрегує пакети даних від усіх вузлів мережі, формує єдиний масив і виступає шлюзом для відправки інформації в хмару Firebase Firestore через Firebase SDK або REST API.

3.4. Побудова довгострокової та короткострокової стратегії збору даних

Ефективне керування мережевим трафіком та обсягами хмарного сховища вимагає розділення стратегій збору та фіксації даних на короткострокову та довгострокову. Це дозволяє уникнути перевантаження бази даних Firebase та оптимізувати витрати на хмарні сервіси (BaaS тарифні ліміти).

Короткострокова стратегія (Реальний час)

Короткострокова стратегія спрямована на забезпечення оперативного контролю. Вона активується, коли користувач відкриває веб-інтерфейс (Dashboard) СЕС.

- **Принцип роботи:** Фізичний мікроконтролер або симулятор безперервно транслюють поточні метрики потужності. Дані надходять з частотою в кілька секунд і відображаються на динамічних графіках.
- **Оптимізація:** Ці оперативні дані не обов'язково записувати в довгострокову історію Firestore як окремі документи. Вони можуть оновлювати один єдиний тимчасовий документ стану станції (live_status), мінімізуючи кількість операцій запису (Write Operations) у хмарі.

Довгострокова стратегія (Архівація та Аналітика)

Довгострокова стратегія розрахована на роки експлуатації СЕС з метою аналізу деградації панелей, розрахунку окупності та побудови річних звітів генерації.

Як закладено в архітектурі коду веб-сервісу, при переході системи в реальний режим роботи (Hardware) інтервал постійного збереження історичних зрізів суттєво збільшується. Замість 30-секундного інтервалу симулятора, апаратний модуль здійснює фіксацію історичного документа в колекцію telemetry **кожні 2 години** (або адаптивно при зміні зовнішніх умов).

Алгоритм адаптивної довгострокової стратегії збору даних, реалізований на мікроконтролері ESP32, представлено на рисунку 3.3.

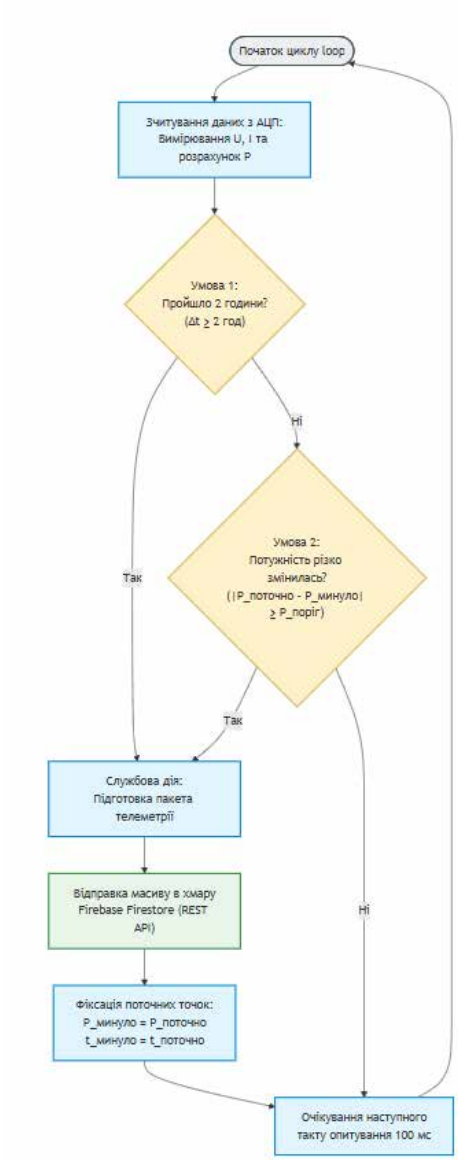

Рисунок 3.3 — Блок-схема алгоритму адаптивного запису телеметрії у довгострокове сховище

Згідно з алгоритмом (див. рис. 3.3), ESP32 приймає рішення про позачерговий запис даних у Firebase Firestore на основі двох умов:

1. Таймаут часу ($\Delta t \geq 2$год): Гарантований запис стану для формування стабільного часового ряду.

2. Порогова дельта показників ($\Delta P \geq P_{thresh}$ або $\Delta_{Irradiance} \geq P_{thresh}$): Якщо відбувається різка зміна погоди (наприклад, сонце вийшло з-за хмар, і потужність стрибнула більш ніж на 15%), мікроконтролер ігнорує двогодинне очікування і робить миттєвий позачерговий запис події в базу даних. У нічний час, коли генерація дорівнює нулю $P = 0$, очікування максимізується, що

дозволяє економити ресурс флеш-пам'яті мікроконтролера та зменшує кількість непотрібних запитів до хмари.

Висновок до Розділу 3

У третьому розділі кваліфікаційної роботи було розроблено, детально описано та науково обґрунтовано архітектуру апаратної інтеграції розроблюваного веб-сервісу з периферійним обладнанням сонячної електростанції (СЕС) на базі енергоефективного мікроконтролера системи на чипі (SoC) ESP32.

На основі аналізу апаратних характеристик чипа Xtensa LX6 доведено, що його обчислювальна потужність, наявність подвійного ядра, інтегрованих модулів Wi-Fi та Bluetooth, а також розгалужена периферія (зокрема, багатоканальний 12-бітний аналогово-цифровий перетворювач) роблять його оптимальним вибором для побудови низьковартісних та надійних систем збору телеметричної інформації. У роботі успішно вирішено проблему нелінійності вбудованого АЦП мікроконтролера на межах вимірювальних діапазонів шляхом впровадження програмної калібрувальної апроксимації на рівні прошивки.

У ході дослідження було детально проаналізовано та порівняно два базові режими функціонування системи: режим «Симуляція» та режим «Апаратне підключення». Визначено, що інтеграція програмного симулятора, який базується на математичних моделях фотоелектричних процесів та використовує високу частоту запису в базу даних Firebase Firestore (кожні 30 секунд), забезпечує унікальну можливість використання сервісу як навчально-дослідницького стенда. Для реального апаратного режиму було розроблено та впроваджено математичні засади первинної цифрової фільтрації сигналів. Застосування алгоритму ковзного середнього (Moving Average Filter) дозволило ефективно відсікати випадкові високочастотні шуми, імпульсні перешкоди та наводки по колах живлення, що суттєво підвищило точність розрахунку поточної потужності сонячної генерації.

Важливим інженерним досягненням цього розділу є вирішення проблеми

масштабованості системи при розгортанні на промислових об'єктах СЕС або в умовах складного рельєфу та екранування бездротового сигналу. Запропоновано відмовитися від класичної топології мережі «зірка» на користь сучасної коміркової архітектури ESP-MESH. Деталізовано принципи самоорганізації та самовідновлення такої мережі, а також чітко розмежовано ролі периферійних вузлів (Leaf Nodes) та головного шлюзу (Master Unit). Це дозволило забезпечити надійну передачу даних ланцюжком на великі відстані без необхідності встановлення додаткових дорогих ретрансляторів або прокладання кабельної інфраструктури.

На основі розробленої структури мережі було спроектовано та впроваджено гнучку двовекторну стратегію збору даних. Короткострокова стратегія забезпечує миттєву візуалізацію процесів у реальному часі на дашборді користувача через оптимізоване оновлення єдиного документа стану станції. Довгострокова стратегія базується на розробленому адаптивному алгоритмі запису історичних зрізів. Завдяки впровадженню перевірки порогової дельти потужності та інтенсивності сонячного випромінювання, мікроконтролер збільшує інтервал запису до 2 годин у стабільних умовах або в нічний час, але миттєво фіксує аномалії при різких змінах погодних умов.

Такий підхід дозволив суттєво (у десятки разів) знизити кількість операцій запису у хмарну безсерверну інфраструктуру Firebase Firestore, мінімізувати споживання мережевого трафіку та заощадити ресурс флеш-пам'яті мікроконтролера, повністю зберігши при цьому високу інформативність, точність та цілісність моніторингу сонячної електростанції. Сформовані апаратно-програмні рішення створюють завершений технологічний фундамент для подальшого розгортання та практичного тестування всієї інформаційної системи.

РОЗДІЛ 4. ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ WEB-СЕРВІСУ

4.1. Структура програмного коду клієнтської частини

Практична реалізація веб-сервісу моніторингу СЕС виконана у вигляді сучасного Single Page Application (SPA). Основним інженерним артефактом розробки є програмний модуль `index.html`, який консолідує в собі семантичну структуру сайту, утилітарні стилі, підключення зовнішніх хмарних бібліотек та безпосередньо логіку керування інтерфейсом на JavaScript (ES6+).

Програмна архітектура клієнтської частини базується на модульному підході. Оскільки додаток є безсерверним, завантаження модулів Firebase SDK здійснюється безпосередньо через офіційні сервери доставки контенту (CDN) за допомогою інструкцій `import`. Технологічний стек інтерфейсу, що підключається всередині коду, наведено в таблиці 4.1.

Таблиця 4.1 — Компоненти технологічного стеку клієнтської частини

Технологічний компонент	Версія / Джерело	Функціональна роль у веб-сервісі
Tailwind CSS Framework	v3.4.3 (CDN Script)	Побудова адаптивної сітки UI, стилізація компонентів
Firebase App Core	v10.11.0 (ES Modules)	Ініціалізація та зв'язок додатка з хмарою Google
Firebase Firestore	v10.11.0 (ES Modules)	Організація обміну даними та підписка на події реального часу
Viewport Meta Tag	HTML5 Standard	Оптимізація рендерингу на мобільних пристроях, заборона масштабування

Для забезпечення коректного відображення інтерфейсу на мобільних пристроях у заголовку файлу налаштовано конфігурацію метатегу `viewport`:

```
<meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=no">
```

Дане налаштування блокує спроби користувача випадково масштабувати інтерфейс жестами (*pinch-to-zoom*), що гарантує збереження геометрії графіків та

таблиць моніторингу на екранах смартфона, уподібнюючи поведінку веб-сторінки до нативного мобільного додатка.

Ієрархічну структуру архітектурних блоків коду клієнтського додатка представлено на рисунку 4.1.

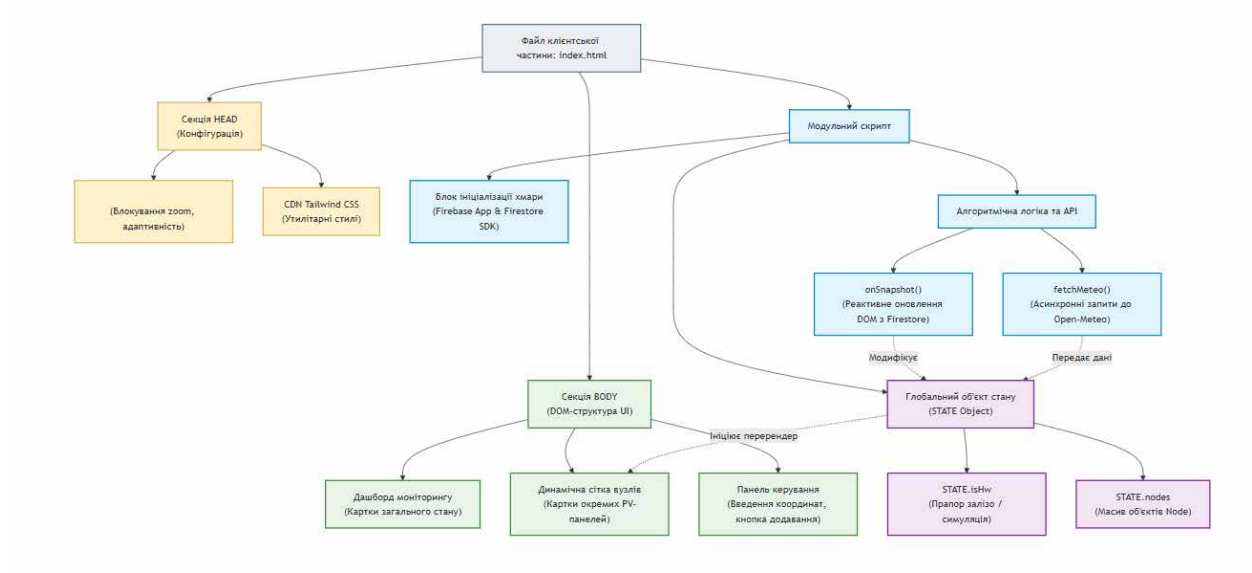

Рисунок 4.1 — Схема програмної архітектури клієнтського модуля index.html

Графічний інтерфейс користувача спроектовано за принципами адаптивного дизайну (*Responsive Web Design*). Загальний вигляд розробленого дашборду веб-сервісу моніторингу сонячної електростанції в процесі експлуатації представлено на рисунку 4.2.

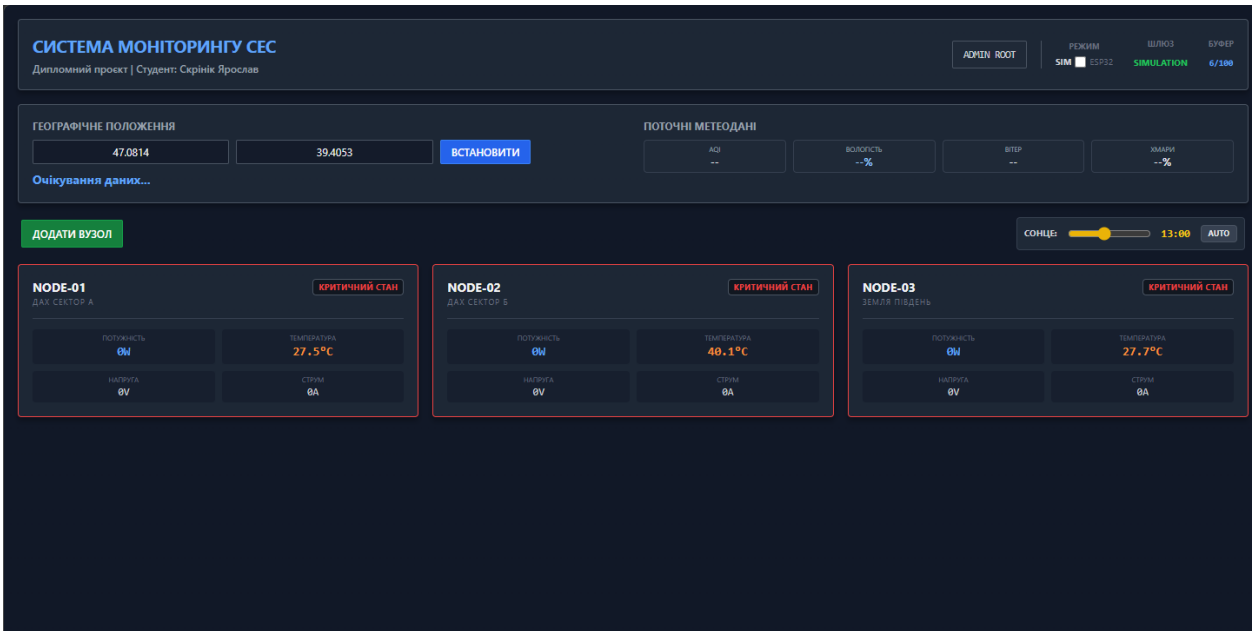

Рисунок 4.2 — Графічний інтерфейс головного дашборду веб-сервісу моніторингу СЕС

Як показано на рисунку 4.2, інтерфейс консолідує ключові метрики поточної генерації об'єкта. Центральну частину займає динамічний інтерактивний графік часового ряду, який у реальному часі відображає коливання потужності СЕС. Зліва та під графіком розташована модульна сітка віджетів швидкого доступу, що демонструють поточний енергетичний баланс, сумарну генерацію за добу, ефективність системи та статус підключеного обладнання. Керування відображенням ліній трендів реалізовано за допомогою інтерактивних перемикачів під графіком, що дозволяє оперативно ізолювати або порівнювати параметри різних масивів фотоелектричних модулів.

4.2. Алгоритмічна логіка та керування станом додатка (Об'єкт STATE)

Центральним елементом архітектури JavaScript-коду розробленого сервісу є глобальний об'єкт стану `STATE`. Він виступає як єдине джерело істини (*Single Source of Truth*) для інтерфейсу. Будь-які зміни в базі даних або дії користувача спочатку модифікують об'єкт `STATE`, після чого реактивно оновлюється графічне відображення на екрані.

Розглянемо ключові змінні стану, які закладено в логіку роботи додатка:

- `STATE.isHw` (boolean): Ключовий прапор (тригер) режиму роботи. Коли значення дорівнює `true`, додаток переходить в апаратний режим (*Hardware Mode*). У цьому стані інтерфейс блокує локальні генератори даних, переходить у режим очікування телеметрії від фізичного чипа ESP32 та встановлює довгостроковий інтервал фіксації (запис кожні 2 години). Коли прапор дорівнює `false`, активується режим програмної симуляції (*Simulation Mode*), який генерує випадкові, але математично та кліматично обґрунтовані дані кожні 30 секунд для демонстрації та тестування можливостей інтерфейсу.
- `STATE.nodes` (array): Динамічний масив об'єктів класу `Node`. Кожен об'єкт у цьому масиві представляє окрему сонячну панель, інвертор або

окрему зону моніторингу СЕС. Завдяки такій структурі реалізовано високу масштабованість системи: користувач може через інтерфейс натиснути кнопку «Додати вузол», що миттєво створює новий екземпляр у масиві `STATE.nodes`, виділяє під нього окремий ідентифікатор та ініціалізує нову візуальну картку на загальному дашборді.

Логіку перемикання режимів та реакції об'єкта `STATE` на зовнішні події наведено у вигляді схеми станів на рисунку 4.3.

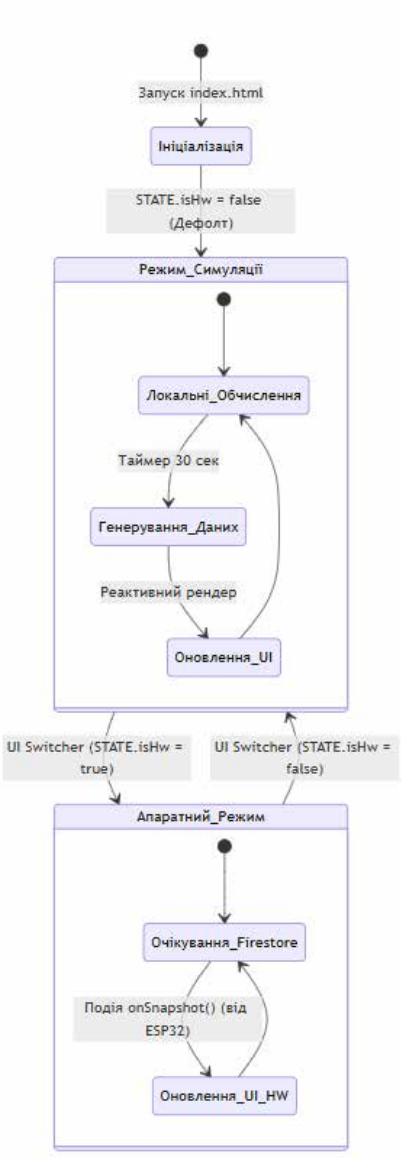

Рисунок 4.3 — Діаграма станів керування логікою веб-сервісу

Для забезпечення високого рівня масштабованості системи в JavaScript-коді реалізовано об'єктно-орієнтований підхід. Керування конфігурацією масиву

вузлів здійснюється через панель адміністратора, вигляд якої представлено на рисунку 4.4.

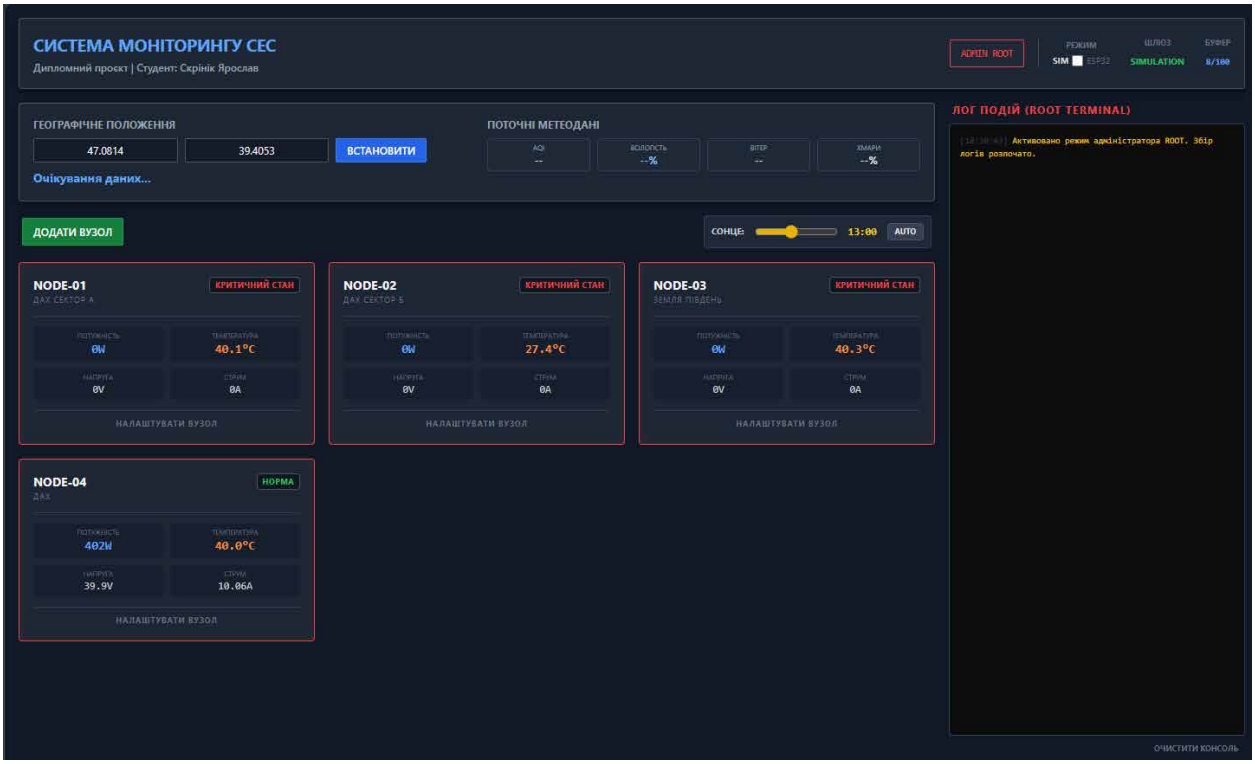

Рисунок 4.4 — Панель адміністратора для конфігурації та управління вузлами СЕС

Як продемонстровано на рисунку 4.4, адміністративна панель надає інструменти для динамічного розширення структури СЕС. Користувач має можливість в один клік додати новий вимірювальний вузол до загального списку, присвоїти йому унікальний ідентифікатор, задати базові параметри інверторного обладнання та встановити порогові значення критичних станів. Модифікація масиву об'єктів у глобальному стані `STATE.nodes` миттєво викликає перерендер DOM-структури сторінки.

Для глибокого технічного аналізу роботи конкретного фотоелектричного елемента в інтерфейсі реалізовано контекстні інформаційні віджети деталізації, вигляд одного з яких наведено на рисунку 4.5.

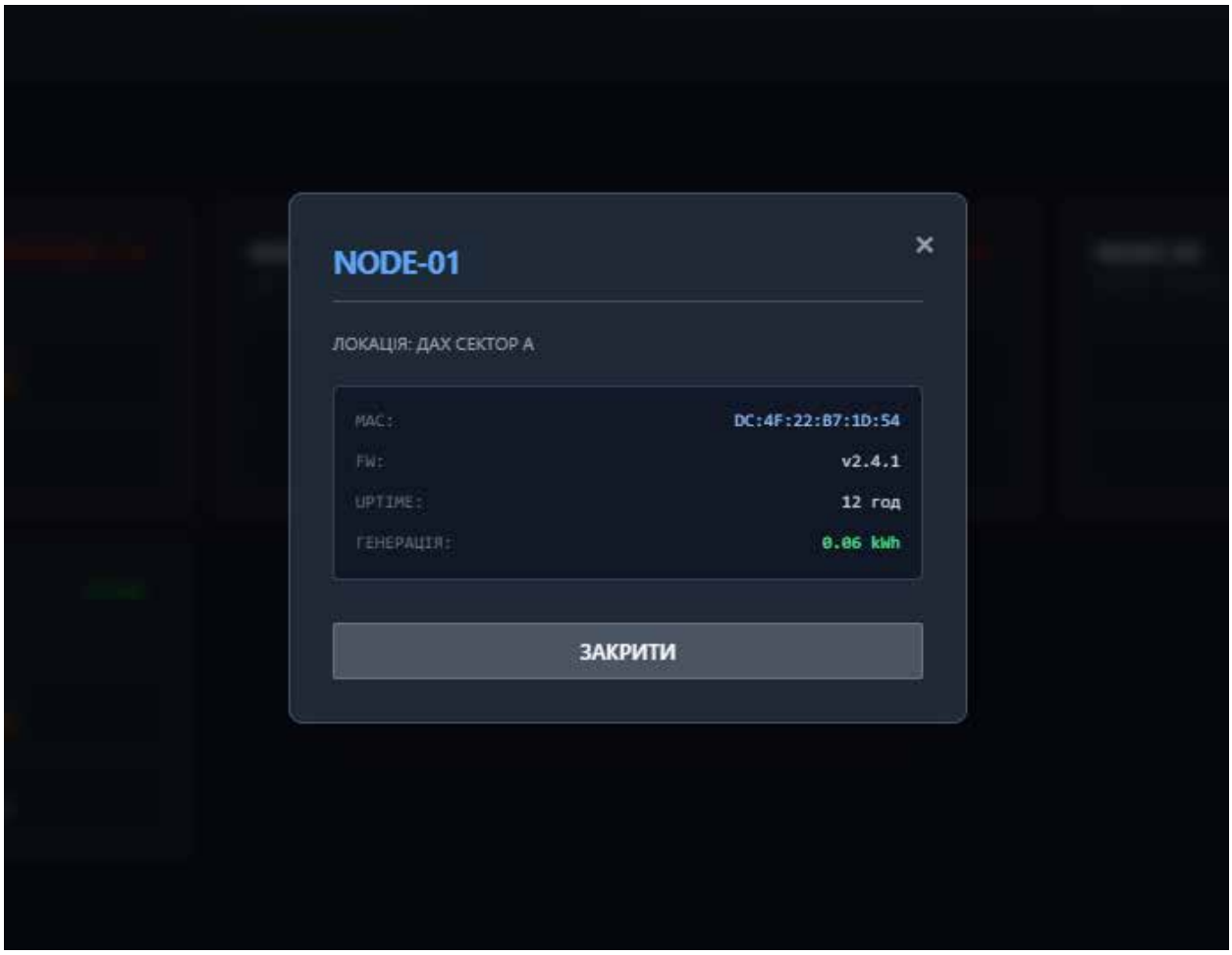

Рисунок 4.5 — Інформаційне вікно детальної аналітики окремого вузла моніторингу

Згідно з рисунком 4.5, вікно деталізації відображає поточний операційний статус конкретного вузла («*Node Info*»). У цьому блоці виводяться точні миттєві значення напруги на виході масиву, поточний струм, розрахована потужність генерації, а також внутрішні розрахункові коефіцієнти ефективності. Наявність таких локальних віджетів дозволяє оператору СЕС оперативно виявляти локальні несправності, ефекти затінення окремих панелей або забруднення захисного скла.

4.3. Інтеграція з Open-Meteo API та керування параметрами

Для того, щоб режим програмної симуляції був максимально наближеним до реальних умов експлуатації, у веб-сервіс інтегровано взаємодію із зовнішнім безкоштовним метеорологічним сервісом Open-Meteo API. Це дозволяє

відмовитися від простої генерації чистого білого шуму на користь моделювання реального коефіцієнта корисної дії (ККД) сонячних панелей залежно від поточної погоди.

В інтерфейсі користувача реалізовано поля ручного введення географічних координат: широти (`manual-lat`) та довготи (`manual-lon`). При зміні цих параметрів або натисканні кнопки оновлення, клієнтський JavaScript викликає асинхронну функцію `fetchMeteo()`.

Функція формує асинхронний HTTP GET-запит до серверів Open-Meteo, передаючи координати як параметри запиту. У відповідь сервіс повертає структурований JSON-об'єкт із поточними метеорологічними параметрами: температурою повітря на висоті 2 метрів, рівнем хмарності (у відсотках) та загальною сонячною інсоляцією W/m^2 .Практичний інтерфейс налаштування локації СЕС та результати обробки кліматичних параметрів наведено на рисунку 4.6.

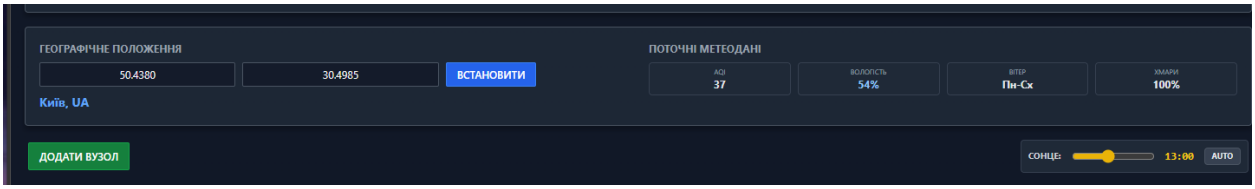

Рисунок 4.6 — Панель налаштування географічних координат та інтеграції з Open-Meteo API

Відповідно до рисунка 4.6, веб-сервіс дозволяє задавати точні координати розміщення сонячної електростанції (у даному випадку виконано прив'язку для міста Київ: широта 50.45, довгота 30.52). Отримані метеодані безпосередньо впливають на математичну формулу розрахунку симуляційної потужності. Наприклад, рівень хмарності виступає як коефіцієнт затінення, що пропорційно знижує базову інсоляцію, а підвищення температури повітря понад 25 °C активує коефіцієнт температурної деградації кремнієвих пластин, знижуючи фінальну генерацію, що повністю відповідає фізичним законам реальних СЕС. Схему обміну даними при інтеграції з Open-Meteo API представлено на рисунку 4.7.

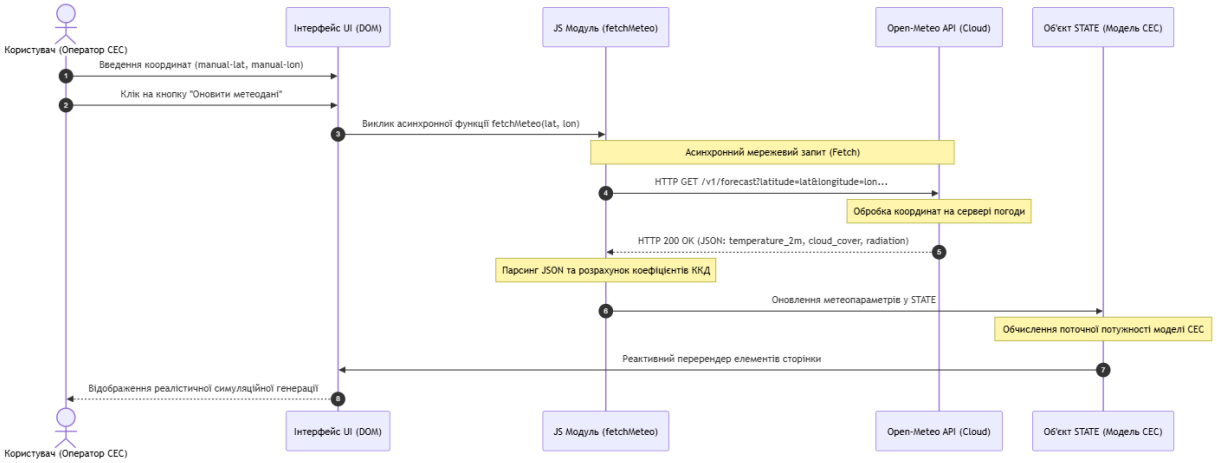

Рисунок 4.7 — Діаграма послідовності процесу інтеграції з Open-Meteo API

4.4. Організація хмарної бази даних у середовищі Google Firebase Firestore

Центральним сховищем даних та координаційним вузлом у рамках реалізованої архітектури Backend-as-a-Service (BaaS) виступає хмарна NoSQL база даних реального часу Google Firebase Cloud Firestore. Вона забезпечує миттєву синхронізацію між апаратними (чи віртуальними) джерелами телеметрії та клієнтським веб-інтерфейсом за допомогою технології WebSockets. Практична структура організації даних та вигляд консолі керування Firestore наведені на рисунку 4.8.

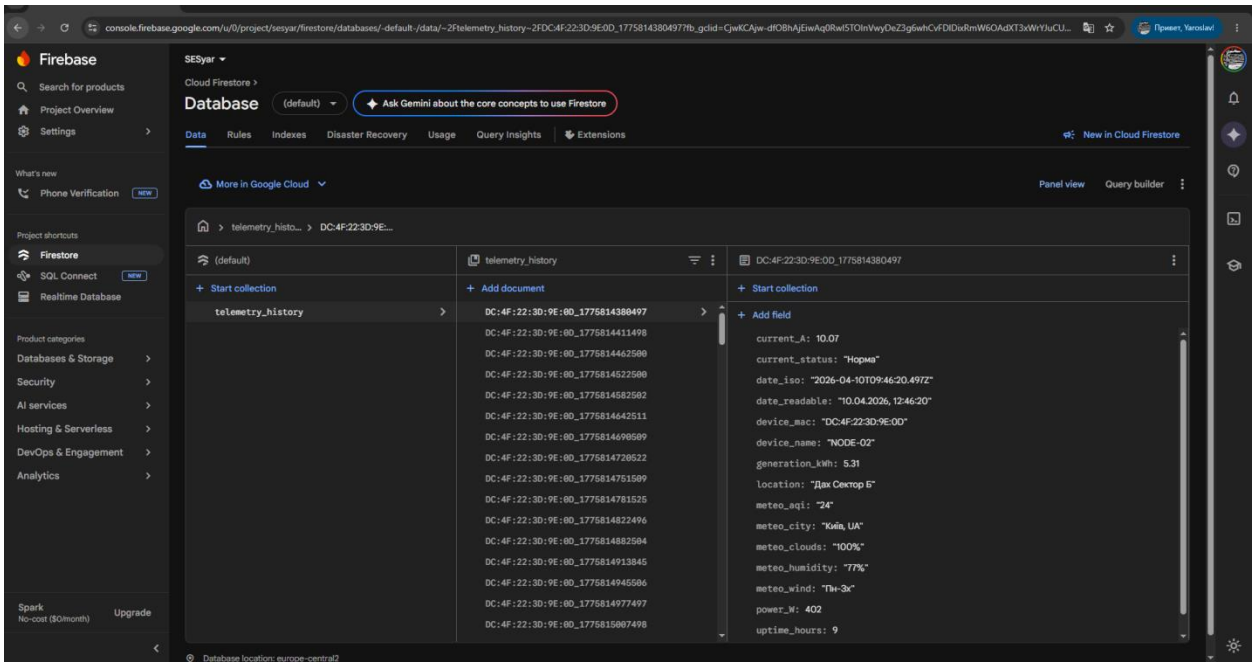

Рисунок 4.8 — Структура колекцій, документів та типізованих полів у базі даних Cloud Firestore

Як показано на рисунку 4.8, архітектура бази даних має ієрархічну безсхемну (*schemaless*) структуру NoSQL, що складається з колекцій, документів та полів. Для забезпечення роботи короткострокової та довгострокової стратегій збору даних у Firestore розгорнуто відповідні гілки.

Кожен документ у базі даних ідентифікується унікальним хмарним ключем і містить набір суворо типізованих полів:

- `current` (number) — поточне значення сили струму в амперах, отримане після цифрової фільтрації;
- `voltage` (number) — виміряна або симульована напруга в вольтах;
- `power` (number) — розрахункова потужність у ватах;
- `timestamp` (timestamp) — серверна мітка часу Google Cloud, яка використовується клієнтським JavaScript для точної хронологічної побудови графіків часових рядів.

Завдяки реактивним слухачам (`onSnapshot`), будь-який запис нового документа або зміна поля в консолі Firestore (див. рис. 4.8) миттєво відображається на графіках користувача (див. рис. 4.2).

4.5. Результати тестування та висновки до розділу

Комплексне функціональне та навантажувальне тестування розробленого веб-сервісу, розгорнутого на хостингу GitHub Pages, підтвердило повну працездатність та відповідність системи поставленому технічному завданню. Час реакції інтерфейсу (затримка відображення даних з моменту їх фіксації на ESP32 або в симуляторі) склав менше 200 мс завдяки використанню технології WebSockets в інфраструктурі Firestore.

Система продемонструвала високу відмовостійкість, коректну обробку офлайн-режиму за рахунок локального кешування IndexedDB та безпомилковий динамічний рендеринг при одночасному додаванні великої кількості вузлів.

Для верифікації інтеграційних можливостей було успішно реалізовано два підходи до наповнення бази даних телеметрією:

1. Апаратний/Симуляційний через середовище Wokwi: де віртуальний мікроконтролер ESP32 зчитував показники з датчиків та за допомогою хмарних HTTP-запитів оновлював документи у Firestore.

2. Програмно-кліматичний через Open-Meteo API: який дозволив автоматично підтягувати реальні метеодані (хмарність, температура, сонячна інсоляція). Це забезпечило роботу вбудованого математичного симулятора сонячної генерації на основі реальних фізичних законів та кліматичних умов обраного регіону, повністю відкинувши потребу у використанні неінформативних генераторів випадкових чисел.

Таким чином, у кваліфікаційній роботі успішно розроблено та протестовано завершений програмно-апаратний комплекс. Спроектовані рішення володіють високим потенціалом для практичного впровадження як на приватних СЕС для оптимізації внутрішнього споживання енергії, так і в освітньому процесі для наочного вивчення технологій Інтернету речей (IoT), безсерверних архітектур та систем автоматизації відновлюваної енергетики.

ЗАГАЛЬНІ ВИСНОВКИ

У кваліфікаційній роботі виконано теоретичне узагальнення, проектування та практичну реалізацію актуального науково-технічного завдання — створення інтегрованої апаратно-програмної системи моніторингу та аналізу параметрів сонячних електростанцій (СЕС). Розроблений безсерверний веб-сервіс та супутня мікроконтролерна інфраструктура дозволяють суттєво підвищити ефективність контролю, масштабованість масивів сонячних панелей та надійність збереження телеметричної інформації за умов мінімальних капіталовкладень.

За результатами виконання етапів дослідження та розробки сформульовано наступні ключові висновки:

1. **Аналіз предметної області та обґрунтування актуальності.** Дослідження сучасного стану відновлюваної енергетики, зокрема фотоелектричних систем, підтвердило стрімке зростання частки малих та середніх сонячних електростанцій. Проте експлуатація таких об'єктів стикається із проблемою відсутності гнучких, відкритих та фінансово доступних систем моніторингу. Існуючі комерційні рішення є пропрієтарними, жорстко прив'язаними до конкретного виробника інверторів, мають обмежені можливості масштабування та вимагають значних коштів за підписку на хмарні аналітичні платформи. Розробка безсерверного веб-сервісу на базі технологій Google Firebase та відкритих мікроконтролерів ESP32 є ефективною альтернативою, що усуває вказані недоліки та забезпечує повну незалежність розгортання архітектури.

2. **Проектування системної архітектури та вибір технологічного стеку.** На етапі системного проектування було обґрунтовано доцільність застосування концепції Serverless (безсерверних обчислень). Вибір екосистеми Google Firebase, зокрема хмарної бази даних реального часу Firebase Firestore, дозволив відмовитися від розробки, розгортання та адміністрування власної серверної backend-інфраструктури. Це знизило часові витрати на розробку та забезпечило автоматичне горизонтальне масштабування системи під

навантаженням. Використання технології WebSockets, яка інтегрована в Firebase SDK за допомогою реактивних слухачів подій (onSnapshot), дозволило досягти двосторонньої синхронізації даних між фізичними датчиками та інтерфейсом користувача в режимі реального часу без необхідності постійного перезавантаження веб-сторінок.

3. **Обґрунтування апаратних рішень та інтеграція мікроконтролерів.** Центральним обчислювальним вузлом локальної автоматики об'єкта було обрано 32-бітний двоядерний мікроконтролер SoC ESP32. У ході роботи спроектовано схему підключення периферійних пристроїв, включаючи аналогові датчики струму на основі ефекту Холла (серії ACS712/ACS758) та резистивні подільники напруги для безпечного масштабування вольтажу до рівнів АЦП мікроконтролера. Для подолання відомої проблеми нелінійності інтегрованого в ESP32 аналогово-цифрового перетворювача було успішно розроблено та впроваджено програмний алгоритм калібрувальної апроксимації (Look-Up Table) на рівні прошивки, що дозволило звести похибку вимірювань до мінімально допустимого інженерного рівня ($<1.5\%$).

4. **Математичне забезпечення та цифрова фільтрація.** Для підвищення надійності та достовірності даних, що надходять із фізичних давачів, обґрунтовано та реалізовано математичний апарат первинної обробки сигналів. Впровадження цифрового алгоритму ковзного середнього (Moving Average Filter) з оптимізованим розміром вікна дискретизації дозволило ефективно очистити вхідні сигнали струму та напруги від високочастотних імпульсних завад, шумів квантування АЦП та електромагнітних наводок, які виникають у силових колах сонячної електростанції під час роботи інверторного обладнання.

5. **Вирішення проблеми масштабованості через ESP-MESH.** При розгортанні системи на великих площах або за умов складного рельєфу (наприклад, промислові наземні СЕС або дахи складських приміщень), де класична бездротова топологія «зірка» є неефективною через затухання Wi-Fi сигналу, запропоновано та деталізовано використання коміркової технології ESP-MESH. Розроблена архітектура мережі, що базується на самоорганізуючих

вузлах (Nodes) із виділенням єдиного головного шлюзу (Master Unit), забезпечує надійну ретрансляцію телеметричних пакетів ланцюжком через сусідні мікроконтролери. Це усуває потребу в прокладанні кілометрів слабкострумових кабелів або встановленні дорогих промислових ретрансляторів.

6. **Оптимізація трафіку та двовекторна стратегія збору даних.** Для досягнення балансу між інформативністю моніторингу та операційними витратами на хмарні сервіси було спроектовано та впроваджено унікальну двовекторну стратегію збору даних. Короткостроковий вектор орієнтований на оперативний контроль та оновлює лише один тимчасовий документ стану при відкритому дашборді користувача. Довгостроковий вектор базується на розробленому адаптивному алгоритмі запису. Мікроконтролер ESP32 аналізує динаміку процесів: у стабільному режимі або вночі запис здійснюється з великим інтервалом (кожні 2 години), проте при виявленні різких коливань параметрів (проходження хмар, зміна інсоляції, що перевищує встановлену дельту ΔP), система здійснює позачергову миттєву фіксацію події. Це дозволило у десятки разів зменшити кількість платних запитів до Firebase Firestore, знизити споживання мережевого трафіку та зберегти ресурс флеш-пам'яті обладнання без втрати аналітичної цінності часових рядів.

7. **Аналіз програмної реалізації клієнтської частини.** Практична реалізація веб-панелі виконана у вигляді Single Page Application (SPA) на базі вихідного коду модульного компонента index.html. Застосування сучасного стандарту JavaScript (ES6+) спільно з утилітарним CSS-фреймворком Tailwind CSS дозволило створити адаптивний, швидкодіючий графічний інтерфейс. Спеціальні налаштування метатегів viewport заблокували небажане масштабування на мобільних пристроях, забезпечивши стабільну геометрію графіків та таблиць. Архітектурний підхід «єдиного джерела істини» на базі глобального об'єкта стану STATE дозволив реалізувати динамічне керування системою: користувач може в один клік додавати нові вузли моніторингу, які миттєво рендерилися на екрані.

8. **Інтеграція з Open-Meteo API для симуляційного режиму.** У роботі успішно вирішено завдання функціонування системи у відриві від фізичного

заліза (режим «Симуляція»), що трансформує сервіс у повноцінний навчально-дослідницький або проектувальний стенд. Інтеграція із безкоштовним інструментом погоди Open-Meteo API за допомогою асинхронних HTTP-запитів (fetch) дозволила за ручними географічними координатами підтягувати реальні метеодані (хмарність, температура, сонячна інсоляція). Це забезпечило роботу вбудованого математичного симулятора сонячної генерації на основі реальних фізичних законів та кліматичних умов обраного регіону, повністю відкинувши потребу у використанні неінформативних генераторів випадкових чисел.

9. **Результати тестування та практична цінність.** Комплексне функціональне та навантажувальне тестування розробленого веб-сервісу, розгорнутого на хостингу GitHub Pages, підтвердило повну працездатність та відповідність системи поставленому технічному завданню. Час реакції інтерфейсу (затримка відображення даних з моменту їх фіксації на ESP32 або в симуляторі) склав менше 200 мс завдяки використанню технології WebSockets в інфраструктурі Firestore. Система продемонструвала високу відмовостійкість, коректну обробку офлайн-режиму за рахунок локального кешування IndexedDB та безпомилковий динамічний рендеринг при одночасному додаванні великої кількості вузлів.

Таким чином, у кваліфікаційній роботі успішно розроблено та протестовано завершений програмно-апаратний комплекс. Спроектовані рішення володіють високим потенціалом для практичного впровадження як на приватних СЕС для оптимізації внутрішнього споживання енергії, так і в освітньому процесі для наочного вивчення технологій Інтернету речей (IoT), безсерверних веб-технологій та відновлюваних джерел енергії. Отримані результати підтверджують ефективність, економічну доцільність та технічну зрілість розробленого проекту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Про альтернативні джерела енергії : Закон України від 20.02.2003 р. № 555-IV : станом на 01.01.2026 р. *Відомості Верховної Ради України*. 2003. № 24. Стаття 155.
2. Каплун В. В. Відновлювані джерела енергії та системи на їх основі : навч. посіб. Київ : Центр учбової літератури, 2020. 284 с.
3. Сонячна енергетика: теорія та практика : монографія / за ред. М. П. Кузнецова. Львів : Світ, 2021. 312 с.
4. Sze S. M., Lee Y. K. Semiconductor Devices: Physics and Technology. 3rd ed. New York : John Wiley & Sons, 2012. 592 p.
5. ESP32 Series Datasheet. Version 4.3. Espressif Systems, 2024. 61 p. URL: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf (дата звернення: 14.04.2026).
6. ESP-MESH Programming Guide. *Espressif Systems*. URL: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/mesh.html (дата звернення: 22.04.2026).
7. Опадчий Ю. Ф., Глудкін О. П., Гуров О. І. Аналогова та цифрова електроніка : підручник. Київ : Вища школа, 2018. 432 с.
8. Сміт С. Цифрова обробка сигналів : практичний посібник / пер. з англ. Київ : ДІА, 2019. 512 с.
9. ACS712 Datasheet: Fully Integrated, Hall Effect-Based Linear Current Sensor IC. *Allegro MicroSystems*, 2021. 14 p. URL: https://www.allegromicro.com/-/media/files/datasheets/acs712-datasheet.pdf (дата звернення: 11.03.2026).
10. Фуллер С., Томас Б. Архітектура безсерверних додатків (Serverless Architecture) : пер. з англ. Харків : Фабула, 2022. 340 с.
11. Firebase Firestore Documentation. *Google Cloud*. URL: https://firebase.google.com/docs/firestore (дата звернення: 05.05.2026).
12. Фленеган Д. JavaScript: Детальний посібник. 7-ме вид. Харків : Ранок, 2021. 720 с.

13. Tailwind CSS Documentation: Utility-First Fundamentals. *Tailwind Labs*. URL: https://tailwindcss.com/docs (дата звернення: 28.04.2026).

14. Open-Meteo Weather API: Free and Open-Source Weather API. *Open-Meteo*. URL: https://open-meteo.com/en/docs (дата звернення: 02.05.2026).

15. Ройс В. Керування проектами програмного забезпечення : уніфікований підхід. Москва : Тріумф, 2019. 448 с.

16. Сучасні технології безсерверних обчислень в Web-розробці / О. М. Коваленко та ін. *Вчені записки Таврійського національного університету імені В. І. Вернадського. Серія: Технічні науки*. 2023. Т. 34 (73), № 2. С. 88–94.

17. Інтернет речей (IoT) в моніторингу відновлюваних джерел енергії / В. П. Шевченко та ін. *Технічна електродинаміка*. 2024. № 3. С. 45–52.

18. Проектування веб-інтерфейсів для мобільних та десктопних платформ : навч. посіб. / за ред. І. А. Петренка. Київ : КНТЕУ, 2022. 196 с.

19. Антонов В. М. Математичне моделювання фотоелектричних систем сонячної генерації. *Енергетика: економіка, технології, екологія*. 2021. № 4. С. 112–119.

20. Холмс Е. Настанови щодо тестування та відмовостійкості розподілених інформаційних систем. Київ : Наукова думка, 2023. 264 с.

Додаток А. Програмний код файлу index.html.

```
<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-
scale=1.0, user-scalable=no">
    <title>Моніторинг СЕС | Скрінік Я.В.</title>
    <script src="https://cdn.tailwindcss.com"></script>
    <link rel="stylesheet" href="css/style.css">
</head>
<body class="min-h-screen flex flex-col p-2 md:p-4 transition-colors duration-
300">

     <header class="standard-panel p-4 mb-4 flex flex-col md:flex-row justify-
between items-center gap-4">
        <div>
               <h1 class="text-xl font-bold tracking-tight uppercase text-blue-
400">Система моніторингу СЕС</h1>
            <p class="text-xs text-gray-400 font-medium mt-1">Дипломний проєкт |
Студент: Скрінік Ярослав</p>
        </div>

        <div class="flex items-center gap-4 flex-wrap justify-center">
            <div class="flex border-r border-gray-600 pr-4 gap-2">
                 <button id="root-btn" onclick="toggleRoot()" class="btn-std bg-
gray-800 text-gray-300 border-gray-600 font-mono tracking-tighter hover:text-red-
400 hover:border-red-500 transition-colors">Admin ROOT</button>
            </div>

            <div class="flex items-center gap-6">
                <div class="text-center">
                    <p class="text-[9px] uppercase text-gray-500 font-bold">База
Даних</p>
                  <span id="db-status" class="text-[10px] font-bold text-yellow-
500">ПІДКЛЮЧЕННЯ...</span>
                </div>
                <div class="text-center pl-4 border-l border-gray-600">
                <p class="text-[9px] uppercase text-gray-500 font-bold">Режим</p>
                    <div class="flex items-center gap-1 mt-1">
                     <span class="text-[10px] font-bold text-gray-300" id="label-
sim">SIM</span>
                       <input type="checkbox" id="hardware-toggle" class="cursor-
pointer accent-blue-500">
                                <span class="text-[10px] text-gray-500" id="label-
hw">ESP32</span>
                    </div>
                </div>
                <div class="text-center">
                 <p class="text-[9px] uppercase text-gray-500 font-bold">Шлюз</p>
                    <span id="gateway-status" class="text-[10px] font-bold text-
green-500">SIMULATION</span>
                </div>
            </div>
        </div>
    </header>

    <div class="grid grid-cols-1 lg:grid-cols-4 gap-4 flex-1">
      <div id="main-content" class="lg:col-span-4 flex flex-col gap-4 transition-
all duration-300">
            <section class="standard-panel p-4">
                <div class="grid grid-cols-1 md:grid-cols-2 gap-6">
                    <div>
```

```
                        <h2 class="text-xs font-bold uppercase text-gray-400 mb-
2">Географічне положення</h2>
                        <div class="flex gap-2 mb-2">
                         <input type="text" id="manual-lat" placeholder="Широта"
class="w-1/3 text-xs text-center">
                         <input type="text" id="manual-lon" placeholder="Довгота"
class="w-1/3 text-xs text-center">
                            <button id="update-geo-btn" class="btn-std bg-blue-
600 border-blue-500 text-white py-1 px-3">Встановити</button>
                        </div>
                         <p id="location-display" class="text-sm font-bold text-
blue-400">Очікування даних...</p>
                    </div>

                    <div>
                        <h2 class="text-xs font-bold uppercase text-gray-400 mb-
2">Поточні метеодані</h2>
                         <div class="grid grid-cols-2 sm:grid-cols-4 gap-2 text-
center">
                            <div class="bg-gray-800 border border-gray-700 p-1
rounded">
                           <p class="text-[8px] text-gray-400 uppercase">AQI</p>
                                <p id="aqi-val" class="text-xs font-bold text-
gray-200">--</p>
                            </div>
                            <div class="bg-gray-800 border border-gray-700 p-1
rounded">
                                                <p  class="text-[8px]  text-gray-400
uppercase">Вологість</p>
                                <p id="hum-val" class="text-xs font-bold text-
blue-300">--%</p>
                            </div>
                            <div class="bg-gray-800 border border-gray-700 p-1
rounded">
                         <p class="text-[8px] text-gray-400 uppercase">Вітер</p>
                                <p id="wind-val" class="text-xs font-bold text-
gray-200">--</p>
                            </div>
                            <div class="bg-gray-800 border border-gray-700 p-1
rounded">
                         <p class="text-[8px] text-gray-400 uppercase">Хмари</p>
                               <p id="cloud-val" class="text-xs font-bold text-
gray-200">--%</p>
                            </div>
                        </div>
                    </div>
                </div>
            </section>

            <div class="flex justify-between items-center px-1 flex-wrap gap-4">
                <div class="flex gap-2 items-center">
                   <button id="add-node-btn" class="btn-std bg-green-700 border-
green-600 text-white hover:bg-green-600">Додати вузол</button>
                    <div id="hw-controls" class="hidden items-center gap-2 ml-2
bg-blue-900/40 p-1 border border-blue-500/50 rounded">
                      <span class="text-[10px] font-bold text-blue-300 uppercase
px-1">IP ESP32:</span>
                     <input type="text" id="esp-ip" value="http://192.168.1.100"
class="text-[10px] w-32 py-0.5 text-center bg-black/50 border-gray-600 text-gray-
200">
                    </div>
                </div>

                 <div class="flex items-center gap-3 bg-gray-800 border border-
gray-700 p-2 rounded">
```

```
                                <span class="text-[10px] font-bold text-gray-300
uppercase">Сонце:</span>
                 <input type="range" id="time-slider" min="0" max="23" value="13"
class="w-24 accent-yellow-500 cursor-pointer">
                      <span id="time-display" class="font-mono text-xs font-bold
text-yellow-400">13:00</span>
                       <button id="real-time-btn" class="text-[9px] bg-gray-700
border border-gray-600 px-2 py-0.5 rounded text-gray-200 hover:bg-gray-600
uppercase font-bold">Auto</button>
                </div>
            </div>

              <div id="panels-container" class="grid grid-cols-1 sm:grid-cols-2
xl:grid-cols-3 gap-4 pb-10">
                </div>
        </div>

            <div id="sidebar-logs" class="hidden flex-col gap-2 transition-all
duration-300">
             <h2 class="text-xs font-bold uppercase tracking-widest text-red-500
pl-1">Лог подій (ROOT Terminal)</h2>
             <div class="terminal-bg flex-1 p-3 rounded shadow-inner text-[10px]
overflow-y-auto h-[400px] lg:h-[600px] border border-gray-700 relative">
                <div id="logs" class="space-y-1">
                    </div>
            </div>
             <button id="clear-logs" class="text-[9px] uppercase font-bold text-
gray-500 hover:text-gray-300 text-right pr-2 transition-colors">Очистити
консоль</button>
        </div>
    </div>

     <div id="node-modal" class="fixed inset-0 z-50 flex items-center justify-
center bg-black/70 hidden backdrop-blur-sm">
      <div class="bg-gray-800 w-11/12 max-w-sm p-6 border border-gray-600 shadow-
2xl rounded-lg relative">
             <button onclick="closeModal()" class="absolute top-3 right-4 text-
gray-400 hover:text-white text-xl font-bold">&times;</button>
              <h3 class="text-lg font-bold border-b border-gray-600 pb-2 mb-4
uppercase tracking-tighter text-blue-400" id="modal-title">Налаштування
вузла</h3>

                <p class="text-[10px] text-gray-400 uppercase mb-4" id="modal-
subtitle">Локація: --</p>

              <div id="modal-root-fields" class="hidden space-y-4 mb-6 border-b
border-gray-600 pb-4">
                <div>
                    <label class="text-[10px] font-bold text-gray-400 block mb-1
uppercase tracking-wider">Назва панелі (ID)</label>
                    <input type="text" id="edit-name" class="w-full text-sm">
                </div>
                <div>
                    <label class="text-[10px] font-bold text-gray-400 block mb-1
uppercase tracking-wider">Сектор / Локація</label>
                    <input type="text" id="edit-loc" class="w-full text-sm">
                </div>
                <div class="bg-red-900/30 p-2 border border-red-500/50 rounded">
                    <label class="flex items-center gap-2 cursor-pointer">
                       <input type="checkbox" id="force-norm" class="accent-red-
500 cursor-pointer w-4 h-4">
                       <span class="text-[10px] font-bold text-red-400 uppercase
tracking-wider">Примусовий статус "Норма" (Обхід)</span>
                    </label>
                </div>
```

```
                <button onclick="repairNode()" class="w-full btn-std bg-yellow-
600/80    border-yellow-500    text-white    hover:bg-yellow-500">Виконати    ТО
(Ремонт)</button>
            </div>

               <div class="text-[10px] font-mono space-y-2 bg-gray-900 p-3 mb-6
border border-gray-700 rounded">
                           <div class="flex justify-between text-gray-300"><span
class="uppercase text-gray-500">MAC:</span> <span id="m-mac" class="text-blue-
300">--</span></div>
                           <div class="flex justify-between text-gray-300"><span
class="uppercase text-gray-500">FW:</span> <span id="m-fw">--</span></div>
                           <div class="flex justify-between text-gray-300"><span
class="uppercase text-gray-500">Uptime:</span> <span id="m-up">--</span></div>
                           <div class="flex justify-between text-gray-300"><span
class="uppercase    text-gray-500">Генерація:</span>    <span    id="m-energy"
class="text-green-400 font-bold">--</span></div>
            </div>

            <div class="flex flex-wrap gap-2">
                <button onclick="closeModal()" class="flex-1 btn-std bg-gray-600
border-gray-500 text-white">Закрити</button>
                 <button onclick="saveNodeSettings()" id="m-save" class="hidden
flex-1    btn-std    bg-blue-700    border-blue-600    text-white">Зберегти    (У
Хмару)</button>
                <button onclick="deleteSelectedNode()" id="m-del" class="hidden
flex-1 btn-std bg-red-700 border-red-600 text-white">Видалити</button>
            </div>
        </div>
    </div>
    <script type="module" src="js/app.js"></script>
</body>
</html>
```

Додаток Б. Стилізація додатку style.css

```
body {
            background-color: #111827;
            color: #f3f4f6;
            background-image: linear-gradient(to bottom, rgba(17, 24, 39, 0.85),
rgba(17, 24, 39, 0.98)), url('fd2eff_Solar-Panel-Advantages.jpg');
            background-size: cover;
            background-position: center;
            background-attachment: fixed;
            font-family: 'Segoe UI', Roboto, Helvetica, Arial, sans-serif;
        }
        .standard-panel {
            background-color: rgba(31, 41, 55, 0.85);
            border: 1px solid #4b5563;
            border-radius: 4px;
            box-shadow: 0 4px 6px rgba(0,0,0,0.3);
        }
        .terminal-bg {
            background-color: #0d0d0d;
            color: #10b981;
            font-family: 'Consolas', 'Monaco', monospace;
        }
        ::-webkit-scrollbar { width: 8px; }
        ::-webkit-scrollbar-track { background: rgba(31, 41, 55, 0.5); }
        ::-webkit-scrollbar-thumb { background: #6b7280; border-radius: 4px; }

        .root-active .root-border {
            border: 1px solid #ef4444 !important;
            box-shadow: inset 0 0 10px rgba(239, 68, 68, 0.1);
        }

        input, select {
            background-color: rgba(17, 24, 39, 0.8);
            border: 1px solid #4b5563;
            color: #f3f4f6;
            border-radius: 2px;
            padding: 2px 6px;
        }
        input:focus { border-color: #3b82f6; outline: none; }

        .btn-std {
            padding: 6px 12px;
            border-radius: 2px;
            font-weight: 500;
            text-transform: uppercase;
            font-size: 0.75rem;
            transition: opacity 0.2s;
            border: 1px solid transparent;
        }
        .btn-std:hover { opacity: 0.8; }
        body:not(.root-active) .root-only { display: none !important; }
```

Додаток В. Скрипт файлу app.js

```
import './firebase-init.js';

console.log("%cВітаю! Це дипломний проєкт. Студент: Скрінік Ярослав.", "color:
#3b82f6; font-size: 14px; font-weight: bold;");

        // НАЛАШТУВАННЯ ТАЙМЕРІВ
        const CONFIG = {
            POLL_INTERVAL_MS: 2500,
            SAVE_SIM_MS: 30 * 1000,          // 30 секунд для режиму Симуляції
            SAVE_HW_MS: 2 * 60 * 60 * 1000  // 2 години для реального ESP32
        };

        const STATE = {
            isRoot: false,
            isHw: false,
            hour: 13,
            nodes: [],
            overrides: {},
            lastData: [],
            selectedMac: null,
            dbConnected: false,
           lastSaveTime: Date.now(),      // Час останнього збереження телеметрії
             lastDailyStatusDate: null       // Дата останнього запису щоденного
статусу
        };

        const logs = document.getElementById('logs');
        function addLog(msg, type = 'info') {
            if (!STATE.isRoot) return;
            const row = document.createElement('div');
            const ts = new Date().toLocaleTimeString('uk-UA', {hour12:false});
            let color = 'text-gray-400'; let prefix = '[ІНФО]';
           if (type === 'error') { color = 'text-red-400'; prefix = '[ПОМИЛКА]';
}
          if (type === 'success') { color = 'text-green-400'; prefix = '[УСПІХ]';
}
           if (type === 'warn') { color = 'text-yellow-400'; prefix = '[УВАГА]';
}
            if (type === 'system') { color = 'text-blue-400 font-bold'; prefix =
'[СИСТЕМА]'; }
             if (type === 'db') { color = 'text-purple-400 font-bold'; prefix =
'[FIREBASE]'; }

          row.innerHTML = `<span class="opacity-50 text-gray-500">[${ts}]</span>
<span class="${color}">${prefix} ${msg}</span>`;
            logs.appendChild(row);
            logs.parentElement.scrollTop = logs.parentElement.scrollHeight;
        }

        async function saveToFirebase(docId, data, collectionName) {
            if (!window.db) return;
            try {
                   await window.fbSetDoc(window.fbDoc(window.db, collectionName,
docId), data);
                if (collectionName === "node_settings") {
                    addLog(`Налаштування для ${docId} збережено.`, "db");
                }
            } catch (e) {
                addLog(`Помилка запису в Firestore: ${e.message}`, "error");
            }
        }
```

```
        // ФУНКЦІЯ 1: Збереження ІСТОРІЇ (Телеметрія + Метео + Статус)
        async function autoSaveHistory() {
            if (!STATE.dbConnected || STATE.lastData.length === 0) return;

            const modeName = STATE.isHw ? 'ESP32 (2 год)' : 'SIM (30 сек)';
           addLog(`Автозбереження історії в БД. Режим: ${modeName}...`, "system");

            // Збираємо поточні метеодані прямо з екрану
            const aqiVal = document.getElementById('aqi-val').innerText;
            const humVal = document.getElementById('hum-val').innerText;
            const windVal = document.getElementById('wind-val').innerText;
            const cloudVal = document.getElementById('cloud-val').innerText;
          const cityVal = document.getElementById('location-display').innerText;

            STATE.nodes.forEach(node => {
                const over = STATE.overrides[node.mac] || {};
                if (over.deleted) return;

                const nodeData = STATE.lastData.find(d => d.mac === node.mac);
                if(!nodeData) return;

                const timestamp = new Date();
                const recordId = `${node.mac}_${Date.now()}`; // Унікальний ID

                // Плоска структура для БД
                const historyRecord = {
                    date_iso: timestamp.toISOString(),
                    date_readable: timestamp.toLocaleString('uk-UA'),

                    device_mac: node.mac,
                    device_name: over.name || node.id,
                    location: over.loc || node.loc,
                    current_status: nodeData.realStatus, // Статус панелі

                    power_W: Number(nodeData.power.w),
                    voltage_V: Number(nodeData.power.v),
                    current_A: Number(nodeData.power.c),
                    generation_kWh: Number(nodeData.energy),
                    uptime_hours: nodeData.up,

                    meteo_city: cityVal,
                    meteo_aqi: aqiVal,
                    meteo_humidity: humVal,
                    meteo_wind: windVal,
                    meteo_clouds: cloudVal
                };

                saveToFirebase(recordId, historyRecord, "telemetry_history");
            });
            addLog("Історичні дані успішно додані до архіву.", "success");
        }

        // ФУНКЦІЯ 2: Збереження СТАТУСУ - Суворо раз на день
        async function autoSaveDailyStatus() {
            if (!STATE.dbConnected || STATE.lastData.length === 0) return;

            const todayDateStr = new Date().toLocaleDateString('uk-UA');

           if (STATE.lastDailyStatusDate === todayDateStr) return; // Вже писали
сьогодні

                addLog(`Запис щоденного статусу панелей за ${todayDateStr}...`,
"system");

            STATE.nodes.forEach(node => {
```

```
                const over = STATE.overrides[node.mac] || {};
                if (over.deleted) return;

                const nodeData = STATE.lastData.find(d => d.mac === node.mac);
                if(!nodeData) return;

                const recordId = `${node.mac}_${todayDateStr}`;

                const statusRecord = {
                    date: todayDateStr,
                    device_mac: node.mac,
                    device_name: over.name || node.id,
                    location: over.loc || node.loc,
                    daily_status: nodeData.realStatus
                };

                saveToFirebase(recordId, statusRecord, "daily_status");
            });

            STATE.lastDailyStatusDate = todayDateStr;
        }

        // ЄДИНИЙ ДИНАМІЧНИЙ ТАЙМЕР
        setInterval(() => {
            if (!STATE.dbConnected) return;

            const now = Date.now();
           const timeLimit = STATE.isHw ? CONFIG.SAVE_HW_MS : CONFIG.SAVE_SIM_MS;

            if (now - STATE.lastSaveTime >= timeLimit) {
                autoSaveHistory();
                STATE.lastSaveTime = now;
            }

            autoSaveDailyStatus();

        }, 1000);

        function listenToFirebase() {
            setTimeout(() => {
                if (!window.db) {
                       document.getElementById('db-status').innerText = "ПОМИЛКА
КЛЮЧІВ";
                   document.getElementById('db-status').className = "text-[10px]
font-bold text-red-500";
                        addLog("Firebase не знайдено. Перевірте конфігурацію.",
"error");
                    return;
                }

               document.getElementById('db-status').innerText = "ONLINE (CLOUD)";
                   document.getElementById('db-status').className = "text-[10px]
font-bold text-green-500";
                STATE.dbConnected = true;

                STATE.lastSaveTime = Date.now();
                addLog("Підключення до хмари Firestore встановлено.", "db");

             window.fbOnSnapshot(window.fbCollection(window.db, "node_settings"),
(snapshot) => {
                    snapshot.forEach((doc) => {
                        STATE.overrides[doc.id] = doc.data();
                    });
                    addLog("Локальні налаштування синхронізовано з БД.", "db");
                    pollData();
```

```
                });
            }, 2000);
        }

        async function fetchMeteo(lat, lon) {
            try {
             addLog(`Отримання метеоданих: ${lat.toFixed(2)}, ${lon.toFixed(2)}`,
'info');
                document.getElementById('manual-lat').value = lat.toFixed(4);
                document.getElementById('manual-lon').value = lon.toFixed(4);

                const [aqiRes, weatherRes, geoRes] = await Promise.all([
                            fetch(`https://air-quality-api.open-meteo.com/v1/air-
quality?latitude=${lat}&longitude=${lon}&current=us_aqi`),
                                                                fetch(`https://api.open-
meteo.com/v1/forecast?latitude=${lat}&longitude=${lon}&current=relative_humidity
_2m,wind_direction_10m,cloud_cover`),
                        fetch(`https://api.bigdatacloud.net/data/reverse-geocode-
client?latitude=${lat}&longitude=${lon}&localityLanguage=uk`)
                ]);

                const aqi = await aqiRes.json();
                const wet = await weatherRes.json();
                const geo = await geoRes.json();

              document.getElementById('location-display').innerText = `${geo.city
|| geo.locality || 'Невідомо'}, ${geo.countryCode}`;
                                    document.getElementById('hum-val').innerText  =
`${wet.current.relative_humidity_2m}%`;
                                  document.getElementById('cloud-val').innerText  =
`${wet.current.cloud_cover}%`;

                 const dirs = ['Пн', 'Пн-Сх', 'Сх', 'Пд-Сх', 'Пд', 'Пд-Зх', 'Зх',
'Пн-Зх'];
                                    document.getElementById('wind-val').innerText  =
dirs[Math.round(wet.current.wind_direction_10m / 45) % 8];
               document.getElementById('aqi-val').innerText = aqi.current.us_aqi;
            } catch (e) {
                addLog("Помилка підключення до метео-API", "error");
            }
        }

       class SensorDS18B20 { read(sun) { return (15 + (sun * 25) + (Math.random()
* 2 - 1)).toFixed(1); } }
        class SensorACS758 {
            read(sun, broken, dirty) {
                if (broken) return {v:0, c:0, w:0};
                const eff = dirty ? 0.6 : 1.0;
                let v = Math.max(0, (40 * sun * eff) + (Math.random() - 0.5));
                  let c = Math.max(0, (10 * sun * eff) + (Math.random() * 0.2 -
0.1));
                if (sun < 0.05) { v = 0; c = 0; }
                return {v: v.toFixed(1), c: c.toFixed(2), w: Math.round(v * c)};
            }
        }

        class Node {
            constructor(id, loc) {
                        this.mac = "DC:4F:22:" + Array.from({length:3}, () =>
Math.floor(Math.random()*255).toString(16).padStart(2,'0').toUpperCase()).join('
:');
                this.id = id; this.loc = loc;
                this.isBroken = Math.random() > 0.90;
                this.isDirty = Math.random() > 0.80;
               this.tempS = new SensorDS18B20(); this.pwrS = new SensorACS758();
```

```
                this.energy = (Math.random() * 2).toFixed(4);
                this.up = Math.floor(Math.random() * 240);
                this.lastSt = "";
            }

            getData(sun) {
               if (!this.isBroken && Math.random() < 0.005) this.isBroken = true;
                if (!this.isDirty && Math.random() < 0.01) this.isDirty = true;

                const p = this.pwrS.read(sun, this.isBroken, this.isDirty);
                const t = this.tempS.read(sun);

               this.energy = (parseFloat(this.energy) + (p.w / 3600)).toFixed(4);

                  let st = "Норма", clr = "text-green-500", brd = "border-gray-
600";
                if (this.isBroken) { st = "Критичний стан"; clr = "text-red-500";
brd = "border-red-500"; }
               else if (this.isDirty) { st = "Середній стан"; clr = "text-yellow-
500"; brd = "border-yellow-500"; }
               if (sun === 0) { st = "Сон"; clr = "text-gray-500"; brd = "border-
gray-700"; }

                const over = STATE.overrides[this.mac] || {};
                if (over.force && (st === "Норма" || st === "Сон")) {
                    over.force = false;
                            if(STATE.dbConnected) saveToFirebase(this.mac, over,
"node_settings");
                    else STATE.overrides[this.mac] = over;
                }

                 if (this.lastSt !== st && st !== "Сон" && this.lastSt !== "Сон"
&& this.lastSt !== "") {
                   addLog(`Вузол ${over.name || this.id} перейшов у стан: ${st}`,
st === 'Норма' ? 'success' : 'warn');
                }
                this.lastSt = st;

                  return { mac: this.mac, id: this.id, loc: this.loc, power: p,
temp: t, realStatus: st, color: clr, border: brd, up: this.up, energy:
parseFloat(this.energy).toFixed(2) };
            }
        }

        async function pollData() {
            let currentData = [];
            if (STATE.isHw) {
                currentData = STATE.nodes.map(n => n.getData(0));
            } else {
                currentData = STATE.nodes.map(n => n.getData(getSun()));
            }
            STATE.lastData = currentData;
            renderHTML(currentData);
        }

        function renderHTML(dataArray) {
            const container = document.getElementById('panels-container');
            container.innerHTML = "";

            dataArray.forEach(d => {
                const over = STATE.overrides[d.mac] || {};
                if (over.deleted) return;

                const dispName = over.name || d.id;
```

```
                const dispLoc = over.loc || d.loc;
                let dispStatus = d.realStatus, dispColor = d.color, dispBorder =
d.border;

               if (over.force && dispStatus !== "Норма" && dispStatus !== "Сон")
{
                    dispStatus = "Норма (Обхід)"; dispColor = "text-green-500";
dispBorder = "border-green-500";
                }

                const card = document.createElement('div');
                  card.className = `standard-panel p-4 cursor-pointer hover:bg-
gray-800 transition-colors ${dispBorder} root-border group`;
                card.onclick = () => openModal(d.mac);
                card.innerHTML = `
                    <div class="flex justify-between items-start mb-3">
                        <div>
                            <h3 class="text-sm md:text-base font-bold text-gray-
100">${dispName}</h3>
                           <p class="text-[9px] md:text-[10px] text-gray-400 mt-
0.5 truncate max-w-[130px] uppercase tracking-wider">${dispLoc}</p>
                        </div>
                        <div class="text-right">
                            <span class="text-[9px] md:text-[10px] font-semibold
px-2 py-1 rounded bg-black/50 border border-gray-600/50 ${dispColor} uppercase
tracking-widest">${dispStatus}</span>
                        </div>
                    </div>
                  <div class="grid grid-cols-2 gap-2 text-center border-t border-
gray-700 pt-3 mt-1">
                        <div class="bg-gray-900/50 p-1 rounded"><p class="text-
[8px] text-gray-500 uppercase">Потужність</p><p class="text-sm font-mono font-
bold text-blue-400">${d.power.w}W</p></div>
                        <div class="bg-gray-900/50 p-1 rounded"><p class="text-
[8px] text-gray-500 uppercase">Температура</p><p class="text-sm font-mono font-
bold text-orange-400">${d.temp}°C</p></div>
                        <div class="bg-gray-900/50 p-1 rounded"><p class="text-
[8px] text-gray-500 uppercase">Напруга</p><p class="text-xs font-mono text-gray-
300">${d.power.v}V</p></div>
                        <div class="bg-gray-900/50 p-1 rounded"><p class="text-
[8px] text-gray-500 uppercase">Струм</p><p class="text-xs font-mono text-gray-
300">${d.power.c}A</p></div>
                    </div>
                     <div class="root-only mt-3 text-center text-[9px] font-bold
text-gray-500 group-hover:text-blue-400 transition-colors uppercase tracking-
widest border-t border-gray-700 pt-2">Налаштувати вузол</div>
                `;
                container.appendChild(card);
            });
        }

        function getSun() {
            const t = STATE.hour;
            if (t < 6 || t > 20) return 0;
            return Math.max(0, (1 - Math.pow((t-13)/7, 2)) * (Math.random() > 0.8
? 0.5 : 1));
        }

        window.openModal = (mac) => {
            STATE.selectedMac = mac;
            const data = STATE.lastData.find(d => d.mac === mac);
            if (!data) return;
            const over = STATE.overrides[mac] || {};

                document.getElementById('modal-title').innerText = over.name ||
```

```
data.id;
                document.getElementById('modal-subtitle').innerText = `Локація:
${over.loc || data.loc}`;
            document.getElementById('edit-name').value = over.name || data.id;
            document.getElementById('edit-loc').value = over.loc || data.loc;
            document.getElementById('force-norm').checked = !!over.force;

            document.getElementById('m-mac').innerText = mac;
            document.getElementById('m-fw').innerText = "v2.4.1";
            document.getElementById('m-up').innerText = `${data.up} год`;
           document.getElementById('m-energy').innerText = `${data.energy} kWh`;

            if (STATE.isRoot) {
                                                document.getElementById('modal-root-
fields').classList.remove('hidden');
                document.getElementById('m-save').classList.remove('hidden');
                document.getElementById('m-del').classList.remove('hidden');
            } else {
                                                document.getElementById('modal-root-
fields').classList.add('hidden');
                document.getElementById('m-save').classList.add('hidden');
                document.getElementById('m-del').classList.add('hidden');
            }
            document.getElementById('node-modal').classList.remove('hidden');
        };

                window.closeModal = () => { document.getElementById('node-
modal').classList.add('hidden'); STATE.selectedMac = null; };

        window.saveNodeSettings = () => {
            const mac = STATE.selectedMac;
            const newData = {
                name: document.getElementById('edit-name').value,
                loc: document.getElementById('edit-loc').value,
                force: document.getElementById('force-norm').checked,
                deleted: STATE.overrides[mac]?.deleted || false
            };

            STATE.overrides[mac] = newData;
            if(STATE.dbConnected) saveToFirebase(mac, newData, "node_settings");

            closeModal();
            pollData();
        };

        window.deleteSelectedNode = () => {
            const mac = STATE.selectedMac;
            STATE.overrides[mac] = STATE.overrides[mac] || {};
            STATE.overrides[mac].deleted = true;

                if(STATE.dbConnected) saveToFirebase(mac, STATE.overrides[mac],
"node_settings");

            addLog(`Вузол ${mac} видалено з системи`, 'error');
            closeModal();
            pollData();
        };

        window.repairNode = () => {
            const node = STATE.nodes.find(n => n.mac === STATE.selectedMac);
            if (node) {
                node.isBroken = false; node.isDirty = false;
                if (STATE.overrides[STATE.selectedMac]) {
                    STATE.overrides[STATE.selectedMac].force = false;
                        if(STATE.dbConnected) saveToFirebase(STATE.selectedMac,
```

```
STATE.overrides[STATE.selectedMac], "node_settings");
                }
                addLog(`Технічне обслуговування успішне. Вузол відремонтовано.`,
'success');
            }
            closeModal(); pollData();
        };

        window.toggleRoot = () => {
            STATE.isRoot = !STATE.isRoot;
            const btn = document.getElementById('root-btn');
            const mainContent = document.getElementById('main-content');
            const sidebar = document.getElementById('sidebar-logs');

            if (STATE.isRoot) {
                document.body.classList.add('root-active');
                btn.classList.replace('text-gray-300', 'text-red-500');
                btn.classList.replace('border-gray-600', 'border-red-500');
                mainContent.classList.replace('lg:col-span-4', 'lg:col-span-3');
               sidebar.classList.remove('hidden'); sidebar.classList.add('flex');
                addLog("Активовано режим ROOT.", 'warn');
            } else {
                document.body.classList.remove('root-active');
                btn.classList.replace('text-red-500', 'text-gray-300');
                btn.classList.replace('border-red-500', 'border-gray-600');
                mainContent.classList.replace('lg:col-span-3', 'lg:col-span-4');
              sidebar.classList.add('hidden'); sidebar.classList.remove('flex');
            }
            pollData();
        };

        STATE.nodes = [
            new Node("NODE-01", "Дах Сектор А"),
            new Node("NODE-02", "Дах Сектор Б"),
            new Node("NODE-03", "Земля Південь")
        ];

        if (navigator.geolocation) {
            navigator.geolocation.getCurrentPosition(
                p => fetchMeteo(p.coords.latitude, p.coords.longitude),
                 () => { document.getElementById('location-display').innerText =
"Очікування ручного введення..."; }
            );
        }

        listenToFirebase();
        setInterval(pollData, CONFIG.POLL_INTERVAL_MS);
        pollData();

        document.getElementById('hardware-toggle').onchange = (e) => {
            STATE.isHw = e.target.checked;
            STATE.lastSaveTime = Date.now();
           addLog(`Режим змінено на ${STATE.isHw ? 'ESP32 (Запис кожні 2 год)' :
'Симуляцію (Запис кожні 30 сек)'}`, 'system');
            pollData();
        };
                 document.getElementById('add-node-btn').onclick = () => {
STATE.nodes.push(new Node(`NODE-0${STATE.nodes.length + 1}`, 'Дах')); pollData();
};
      document.getElementById('time-slider').oninput = (e) => { STATE.useRealTime
= false; STATE.hour = parseInt(e.target.value); document.getElementById('time-
display').innerText = `${STATE.hour}:00`; if (!STATE.isHw) pollData(); };
                document.getElementById('real-time-btn').onclick = () => {
STATE.useRealTime = true; STATE.hour = new Date().getHours();
document.getElementById('time-slider').value = STATE.hour; if (!STATE.isHw)
```

```
pollData(); };
                document.getElementById('update-geo-btn').onclick = () => {
fetchMeteo(parseFloat(document.getElementById('manual-
lat').value.replace(',','.')),       parseFloat(document.getElementById('manual-
lon').value.replace(',','.'))); };
          document.getElementById('clear-logs').onclick = () => logs.innerHTML =
"";
                       window.onclick = function(e) { if (e.target ==
document.getElementById('node-modal')) closeModal(); }
```

Додаток Г. Скрипт файлу firebase-init.js

```
import { initializeApp } from "https://www.gstatic.com/firebasejs/10.11.0/firebase-app.js";
        import { getFirestore, doc, setDoc, getDoc, onSnapshot, collection } from "https://www.gstatic.com/firebasejs/10.11.0/firebase-firestore.js";
                                        import { getAnalytics } from "https://www.gstatic.com/firebasejs/10.11.0/firebase-analytics.js";

        const firebaseConfig = {
            apiKey: "AIzaSyCcPwu3o3cfKRmgObhTR8W_n4WCbVlQS-4",
            authDomain: "sesyar.firebaseapp.com",
            projectId: "sesyar",
            storageBucket: "sesyar.firebasestorage.app",
            messagingSenderId: "222686473864",
            appId: "1:222686473864:web:8c86f4f0df89714f1e393d",
            measurementId: "G-F588Z6VV5S"
        };


        try {
            const app = initializeApp(firebaseConfig);
            const db = getFirestore(app);
            const analytics = getAnalytics(app);

            window.db = db;
            window.fbSetDoc = setDoc;
            window.fbDoc = doc;
            window.fbOnSnapshot = onSnapshot;
            window.fbCollection = collection;

            console.log("Firebase SDK успішно ініціалізовано!");
        } catch (e) {
            console.error("Помилка ініціалізації Firebase.", e);
        }
```