Хмельницький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерної інженерії та інформаційних систем

КВАЛІФІКАЦІЙНА РОБОТА

бакалавр
Освітній рівень

Блок керування і обробки цифрових даних системи сонячних панелей
Назва теми

КвРКІ. 101064.21.01.14 ПЗ
Шифр

Галузь знань *12 «Інформаційні технології»*
Шифр, назва

Спеціальність *123 «Комп'ютерна інженерія»*
Шифр, назва

Освітня програма *«Комп'ютерна інженерія та програмування»*
Назва

Виконав: студент IV курсу, група КІ2с-21-1 _____ Є. В. Цибульський
Підпис Ініціали, прізвище

Керівник _____ П. Г. Регіда
Підпис, дата Ініціали, прізвище

Нормоконтролер _____ І.О. Засорнова
Підпис, дата Ініціали, прізвище

До захисту допускаю:
Зав. кафедри комп'ютерної інженерії та інформаційних систем _____ Т.О. Говорущенко
Підпис Ініціали, прізвище

«31» травня 2024 р.

Хмельницький 2024

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій

Кафедра Комп'ютерної інженерії та інформаційних систем

Освітній рівень бакалавр

Галузь знань 12 Інформаційні технології

Спеціальність 123 Комп'ютерна інженерія

Освітня програма «Комп'ютерна інженерія та програмування»

ЗАТВЕРДЖУЮ

Зав. кафедри Т.О. Говорущенко

" 10 " 01 2024 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА

Цибульському Єгору Володимировичу

Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Блок керування і обробки цифрових даних системи сонячних панелей

Керівник проекту (роботи) Регіда П.Г., старший викладач.

Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 15.02.2024 р. № 8

2. Строк подання студентом проекту (роботи) на кафедру 01.06.2024 р.

3. Вихідні дані до проекту (роботи) Завдання на кваліфікаційну роботу

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)

Функціонування фотоелектричних систем

Планування та реалізація поставленої задачі

Демонстрація роботи проєкту

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)

Принципова схема фотоелектричної системи з інвертором

Структурна схема фотоелектричної системи з кіберфізичним пристроєм

Загальна діаграма збору та трансформації даних зібраних з сонячних панелей

6. Консультанти розділів дипломного проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Засорнова І. О., доцент кафедри КІІС		
Антиплагіат	Нічепорук А.О., доцент кафедри КІІС		

7. Дата видачі завдання «_10_»_____01_____2024 р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів (розділів) дипломного проекту (роботи)	Термін виконання етапів проекту (роботи)	Примітка
1	Вибір напряму дослідження та узгодження тематики кваліфікаційної роботи з керівником	10.01.2024	виконано
2	Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження	01.02.2024	виконано
3	Робота над розділом 1 – Функціонування фотоелектричних систем	01.03.2024	виконано
4	Робота над розділом 2 – Планування та реалізація поставленої задачі	01.04.2024	виконано
5	Робота над розділом 3 – Демонстрація роботи проєкту	29.04.2024	виконано
6	Оформлення пояснювальної записки згідно вимог	25.05.2024	виконано
7	Попередній захист ВКР	26.05.2024	виконано
8	Захист ВКР на засіданні ЕК	Червень 2024 року	

Студент _____ Є. В. Цибульський
Підпис Ініціали, прізвище

Керівник роботи _____ П. Г. Регіда
Підпис Ініціали, прізвище

№ рядка	формат	Позначення	Найменування	Кіл. листів	№ екз	Примітка
			Текстові документи			
1		КвРКІ. 101064.21.01.14 ПЗ	Пояснювальна записка	72		
			Графічні документи			
2		КвРКІ. 101064.21.01.14 Е8	Принципова схема фотоелектричної системи з інвертором	1		
3		КвРКІ. 101064.21.01.14 Е8	Структурна схема фотоелектричної системи з кіберфізичним пристроєм	1		
4		КвРКІ. 101064.21.01.14 Е8	Загальна діаграма збору та трансформації даних зібраних з сонячних панелей	1		

Зм	Арк	№ докум	Підпис	Дата	КвРКІ. 101064.21.01.14 ПЗ		
Розробив		Цибульський		31.05	Блок керування і обробки цифрових даних системи сонячних панелей	Літера	Аркуш	Аркушів
Перевір.		Регіда		31.05		У	1	72
						ХНУ, КІ2с-21-1		
Н. контр.		Засорнова		31.05				
Затв.		Говорущенко		31.05				

АНОТАЦІЯ

Тема кваліфікаційної роботи: «Блок керування і обробки цифрових даних системи сонячних панелей».

Автор роботи: Цибульський Єгор Володимирович.

Керівник роботи: Регіда Павло Геннадійович.

Пояснювальна записка: 72 с., 22 рис., 1 табл., 4 дод., 62 джерела.

Графічна частина: 3 креслення.

ФОТОЕЛЕКТРИЧНА СИСТЕМА, МОНІТОРИНГ ТА АНАЛІЗ МЕРЕЖЕВОГО ТРАФІКУ, КІБЕРФІЗИЧНА СИСТЕМА.

Метою роботи було створення програмно-апаратного комплексу, що забезпечуватиме користувацький інтерфейс для отримання, зберігання та обробки даних з блоку керування фотоелектричними системами.

Об’єктом роботи є функціонування кіберфізичної системи альтернативних методів генерації електроенергії із фотоелектричними елементами.

Предметом роботи є програмно-апаратний комплекс для роботи з кіберфізичною системою із фотоелектричними елементами.

Під час виконання даної роботи було використано аналіз мережевого трафіку, що збирається з кіберфізичних пристроїв та застосовано сучасні методи подання інформації для користувача.

Підпис студента

30.05.2024

Дата

ЗМІСТ

ВСТУП

Кіберфізичні системи стають все більш життєво важливими в сучасному житті, оскільки вони долають розрив між цифровим і фізичним світами, забезпечуючи автоматизацію, ефективність та інновації в різних секторах, таких як виробництво, транспорт, охорона здоров'я, енергетика тощо. Ці системи інтегрують обчислювальні алгоритми та фізичні компоненти для моніторингу та керування фізичними процесами. Вони покладаються на датчики, виконавчі механізми та комунікаційні мережі для збору даних із фізичного світу, обробки їх за допомогою обчислювальних алгоритмів та ініціювання дій для впливу на фізичні процеси.

Неможливо переоцінити значення кіберфізичних систем у сучасному суспільстві, житті та промисловості, оскільки вони відіграють вирішальну роль у різних аспектах сучасного життя. Кіберфізичні системи — це інтелектуальні системи, які об'єднують обчислювальні алгоритми та фізичні компоненти для взаємодії з фізичним світом.

Кіберфізичні системи дозволяють автоматизувати завдання та процеси, що сприяє підвищенню ефективності, продуктивності та економічності. У виробничому секторі кіберфізичні системи дають початок розумним фабрикам, де машини спілкуються одна з одною та коригують операції в режимі реального часу для оптимізації виробництва.

Кіберфізичні системи збирають величезну кількість даних із фізичного світу, які можна проаналізувати, щоб отримати інформацію та інформувати процеси прийняття рішень. Цей підхід, що керується даними, забезпечує кращий розподіл ресурсів та вдосконалення операційних стратегій.

Кіберфізичні системи відіграють вирішальну роль у підвищенні безпеки та захисту в різних сферах. В охороні здоров'я кіберфізичні системи використовується в медичних пристроях для моніторингу здоров'я пацієнтів і забезпечення своєчасного втручання, покращуючи результати пацієнтів і знижуючи витрати на охорону здоров'я [1].

Підсумовуючи, кіберфізичні системи мають важливе значення в сучасному житті через свою роль в автоматизації, ефективності, безпеці та інноваціях. Стандартизація даних, отриманих із цих систем, має вирішальне значення для забезпечення сумісності, масштабованості, якості даних, безпеки, конфіденційності та відповідності нормативним вимогам, що зрештою максимізує переваги та мінімізує ризики, пов’язані з розгортанням кіберфізичних систем [2].

На сьогоднішній день актуальність фотоелектричних систем є одним з найбільших пріоритетів. Наразі проходять незчислені кампанії щодо боротьби з неекологічними методами генерації електроенергії, а з щоденними обстрілами усієї енергетичної інфраструктури питання розвитку та підтримки альтернативних методів генерації електроенергії є ключовим.

Мета роботи було розробити програмно-апаратний комплекс, який надаватиме користувацький інтерфейс для отримання, збереження та опрацювання інформації з блоку керування фотоелектричних систем.

Об’єктом роботи є процес опрацювання даних з інвертору в кіберфізичній системі альтернативних методів генерації електроенергії із фотоелектричними елементами.

Предметом роботи є програмно-апаратний комплекс, що забезпечує опрацювання даних з інвертору в кіберфізичній системі альтернативних методів генерації електроенергії із фотоелектричними елементами.

Для досягнення поставленої мети було проведено аналіз мережевого трафіку, програмування кіберфізичних пристроїв на базі Raspberry Pi, високорівневе програмування та робота з базами даних.

Практичне значення має спроєктовано програмно-апаратний комплекс для опрацювання інформації фотоелектричних елементів в кіберфізичній системі генерації електроенергії, що збирає та зберігає отримані дані з сонячних панелей, формує звітність у зручному форматі для користувача, відображає поточний стан інвертору із діодною індикацією.

1 ФУНКЦІОНУВАННЯ ФОТОЕЛЕКТРИЧНИХ СИСТЕМ

1.1 Особливості функціонування сонячних станцій

Світовий ринок інверторів для керування сонячними панелями характеризується великою різноманітністю та відсутністю єдиного універсального стандарту. Це ускладнює розробку програмного забезпечення для моніторингу та керування сонячними електростанціями, оскільки кожен інвертор може мати свій власний протокол обміну даними. Тому одним із головних завдань проєкту з розробки системи моніторингу та керування сонячними електростанціями є уніфікація інверторів. Це передбачає, що потрібно знайти спосіб привести різні інвертори до спільного знаменника, щоб їх можна було легко використовувати в рамках одного програмного інтерфейсу.

Наприклад, одним із таких інверторів є Deye SUN-10K-SG04LP1-EU [3], який використовується як цільовий пристрій у проекті, та який можна побачити на рисунку 1.1.

Рисунок 1.1 – Інвертор Deye SUN-10K-SG04LP1-EU

Світ інверторів вражає своєю різноманітністю. Різні виробники, моделі та технології роблять вибір оптимального рішення складним завданням. До того ж, набори сенсорів та інші особливості інверторів можуть істотно відрізнятися, що ускладнює їх порівняння та інтеграцію. План завершеної системи з використанням інверторів для керування, та обробки інформації з сонячних панелей можна побачити на рисунку 1.2. Також на рисунку 1.2 зображено усі типи підключення, необхідні для роботи системи, з виводами для розширення можливостей системи та масштабування.

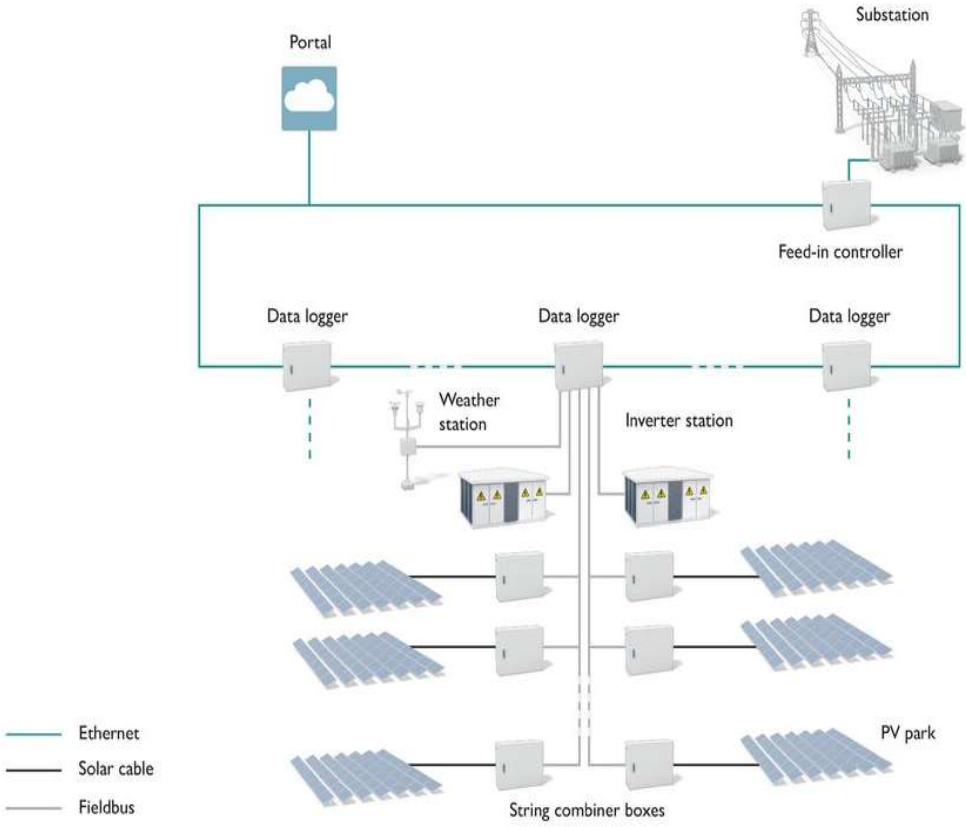


Рисунок 1.2 – Завершений план системи для використання інверторів обробки даних з сонячних панелей

Зображена система є узагальненою та універсальною, без чіткої прив’язки до певної моделі, або бренду інверторів. Саме тому виникла потреба в уніфікації. Універсальні стандарти та протоколи дозволять зробити ринок та системи використання інверторів більш прозорим та зручним для користувачів.

Для вирішення задачі уніфікації було обрано підхід, що ґрунтується на конфігураційних файлах, які можна знайти в Інтернеті. Через обмеженість ресурсів та доступу до різних інверторів, було прийнято рішення обмежитися переліком вже існуючих конфігураційних файлів [4]. Ці файли містять всю необхідну інформацію про протокол обміну даними з інвертором, а також про параметри, які він може надавати.

Було сплановано розробити гнучкий алгоритм для роботи з наявними файлами конфігурації. Цей алгоритм має за мету читати файли конфігурації та створювати об'єкт, що містить всі значення, надані інвертором. Однак, варто відзначити, що у деяких випадках деякі значення можуть бути недоступними для отримання з інвертора через особливості моделі (наприклад, у моделі Deye SUN-12K-SG01LP1-EU). У таких випадках ці значення будуть мати порожні значення (null).

Цей алгоритм повинен бути гнучким та адаптованим до різних форматів файлів конфігурації, забезпечуючи правильне зчитування та обробку інформації незалежно від її доступності. Для цього необхідно провести детальний аналіз структури файлів конфігурації та врахувати можливі варіанти відсутності деяких параметрів у деяких моделях інверторів.

Будучи ключовою складовою проєкту, правильно розроблений алгоритм забезпечить надійну та ефективну роботу інверторів у різних умовах експлуатації. Це важливо для забезпечення стабільності та продуктивності сонячних енергетичних систем у будь-яких умовах [5].

Уніфікація інверторів дає ряд переваг для користувачів та розробників системи моніторингу та керування сонячними електростанціями. По-перше, це забезпечує зручність використання, оскільки користувачі можуть однаково легко

використовувати будь-який інвертор, який підтримується системою, без потреби встановлювати додаткове програмне забезпечення або налаштовувати параметри. По-друге, це спрощує розробку, оскільки розробникам не потрібно писати окремий код для кожного інвертора, а лише використовувати загальний алгоритм, що працює з файлами конфігурації.

По-третє, це надає можливість масштабування, оскільки систему можна легко розширити, щоб підтримувати нові інвертори, які з’являються на ринку, без необхідності змінювати основний код. Таким чином, уніфікація інверторів підвищує ефективність та гнучкість системи [6].

Уніфікація інверторів є важливою складовою розробки систем моніторингу та керування сонячними електростанціями. Це дозволяє зробити систему більш зручною для користувачів, простою для розробників та масштабованою.

Через нестандартну природу конфігураційних файлів та потребу у спеціальному підході для кожного з них, алгоритм роботи з конфігураційними файлами потребує окремої бібліотеки для спрощення розбору у об'єкти зі значеннями. Для цього було обрано бібліотеку YamlDotNet [7], яка надає зручні засоби для роботи з YAML-файлами у середовищі .NET.

YamlDotNet забезпечує легкий та інтуїтивно зрозумілий механізм для перетворення конфігураційних даних з YAML-файлів у об'єкти, що значно спрощує процес розробки та підтримки програмного забезпечення. Використання цієї бібліотеки дозволяє зменшити обсяг коду, необхідного для обробки конфігураційних файлів, та підвищує загальну надійність системи за рахунок використання перевіреного і широко розповсюдженого інструменту.

Завдяки уніфікації інверторів і використанню існуючих рішень для обробки інформації у вигляді файлів конфігурації, вдалося значно спростити майбутню розробку програмного забезпечення для моніторингу та керування інвертором. Уніфікація дозволяє створювати більш гнучкі та масштабовані системи, що легко адаптуються до змін у вимогах або умовах експлуатації.

1.2 Аналіз існуючих рішень

Аналіз проводився в декілька етапів:

- визначення ключових слів та критеріїв пошуку. На цьому етапі було сформульовано чітке розуміння завдання та визначено ключові слова, які використовувалися для пошуку релевантних рішень;
- пошук і збір інформації. За допомогою онлайн-ресурсів, таких як Google Scholar, GitHub, та тематичні форуми, було зібрано інформацію про різні підходи до вирішення подібних завдань;
- відбір та аналіз релевантних рішень. Зібрані дані були ретельно проаналізовані, і на основі визначених критеріїв (функціональність, надійність, простота використання, тощо) було відібрано декілька найбільш багатообіцяючих рішень;
- детальний аналіз та оцінка. Для кожного з відібраних рішень було проведено детальний аналіз, який включав вивчення його характеристик, переваг та недоліків.

Детальний опис трьох проаналізованих проектів наведено нижче.

Deye-logger-at-cmd [8]. Проект deye-logger-at-cmd вивчає внутрішню роботу мікроінверторів Deye, головним чином націлюючись на їхні протоколи зв’язку та заходи безпеки. Використовуючи AT-команди через послідовне з’єднання, він обходить традиційну автентифікацію та витягує облікові дані та ключі Wi-Fi безпосередньо з пристроїв.

AT-команди [9] – це спеціальні команди, які надсилаються по послідовному інтерфейсу, використовуючи спеціальний однойменний протокол обміну. Сама частка «AT» походить від англійського слова «Attention», що означає «Увага», та є доречному, адже кожна команда є запитом уваги від пристрою.

Хоча проект може похвалитися простотою використання та доступністю, реалізація захисних методів потребує вдосконалення. Це збільшує шанс обходу процесу автентифікації що створює небезпеку у вигляді несанкціонованого доступу та потенційної вразливості системи в цілому. Розробник Deye зазначає, що

проєкт потребує вдосконалення та рекомендує не використовувати його на цьому етапі розробки.

Незважаючи на проблеми безпеки, проект приваблює людей, які прагнуть глибше зрозуміти свої інвертори Deye. Деякі використовують цю інформацію для особистих цілей моніторингу, тоді як інші інтегрують її зі стороннім програмним забезпеченням для візуалізації та аналізу даних.

Проект також розкриває деякі цікаві особливості та обмеження інверторів Deye. Наприклад, він показує, що пристрої мають вбудований веб-сервер, до якого можна отримати доступ через локальну IP-адресу. Однак це також показує, що пристрої мають закодований пароль, який користувач не може змінити, що становить серйозну загрозу безпеці.

Сам проект відносно невеликий, складається з одного виконуваного файлу для різних операційних систем. Однак його можливості та функціональний список можливостей, які він надає є видатними для його розміру. Це викликає дискусії про прозорість, право власності користувачів на дані пристрою та тонку межу між дослідженням і використанням у технологічному просторі.

Дуже важливо визнати, що використання цього проєкту несе потенційні ризики. Deye може вважати це порушенням гарантії або умов обслуговування, що призведе до несправності пристрою або втрати гарантії. Також варто зазначити, що несанкціонований доступ до будь-якого пристрою викликає проблеми на рахунок безпеки та потенційні правові наслідки.

Та через те, що проект може мати непередбачені наслідки для інверторів Deye та електромережі. Наприклад, це може перешкоджати оновленню мікропрограми або протоколам зв’язку пристроїв, впливаючи на їх продуктивність і стабільність. Це також може порушити баланс і координацію електромережі, викликаючи коливання або збої [10].

На завершення проект deye-logger-at-cmd пропонує вікно у внутрішню роботу мікроінверторів Deye, минаючи традиційну автентифікацію для прямого доступу до інформації про пристрій. Хоча його технічні аспекти є потенційно

привабливими, важливо зважити етичні міркування та потенційні ризики, перш ніж використовувати цей інструмент. Побачити розглянутий проєкт у дії модна на рисунку 1.3.

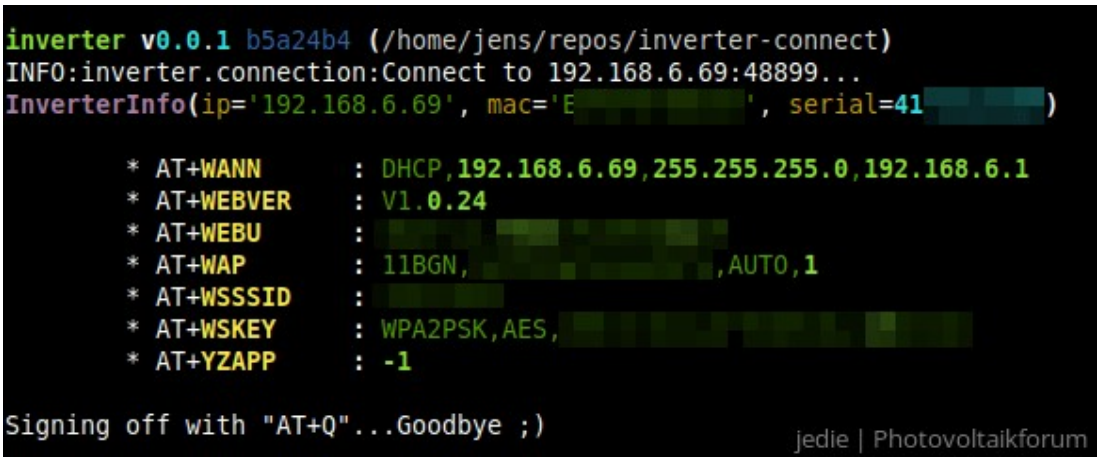


Рисунок 1.3 - deye-logger-at-cmd в дії

Solarman Datalogger [11]. Реєстратори даних Solarman, запропоновані однойменною компанією, діють як центральна програмна складова фотоелектричних систем. Ці пристрої збирають і передають важливі дані про продуктивність системи, включаючи роботу інвертора, виробництво електроенергії та фактори навколишнього середовища. Розуміння їх функцій відкриває цінну інформацію для власників і операторів систем.

На відміну від проєкту deye-logger-at-cmd, який зосереджений на отриманні даних за допомогою нетрадиційних методів, Solarman Dataloggers пропонує офіційний та авторизований доступ. Вони випускаються в різних моделях, кожна з яких відповідає конкретним потребам і розмірам системи. Залежно від розміру та складності системи користувачі можуть вибрати Stick Logger, DIN-Rail Logger або Pro Logger. Кожна з цих опцій має свої переваги та особливості, як-от функція plug-and-play, підключення кількох пристроїв або надійна обробка даних і протоколи зв’язку.

Ці реєстратори даних підключаються до інверторів та інших компонентів системи за допомогою різних методів зв’язку, таких як Wi-Fi, Ethernet або

стільникові мережі. Потім зібрані дані передаються на центральну платформу Solarman Portal, де користувачі можуть отримати до них віддалений доступ через веб-браузер або мобільний додаток.

Solarman Portal надає величезну кількість інформації, такої як продуктивність системи в реальному часі, аналіз історичних даних, попередження та сповіщення, а також віддалене налаштування та керування. Користувачі можуть контролювати вироблення енергії, стан інвертора та умови навколишнього середовища, відстежувати тенденції, виявляти потенційні проблеми та оптимізувати продуктивність системи, отримувати своєчасні попередження про системні збої або потенційні проблеми, а також дистанційно керувати системними налаштуваннями та конфігураціями.

Використовуючи Solarman Dataloggers та Solarman Portal, користувачі отримують цінну інформацію про свої фотоелектричні системи, дозволяючи їм максимізувати виробництво енергії, зменшити експлуатаційні витрати, збільшити термін служби системи та отримати спокій. Ці переваги можуть підвищити ефективність і прибутковість фотоелектричних систем, а також задоволеність і впевненість користувачів.

Однак важливо пам’ятати, що ці реєстратори даних є пропрієтарними пристроями та потребують підписки на Solarman Portal для повної функціональності. Крім того, можуть виникнути занепокоєння щодо конфіденційності та безпеки даних, оскільки дані передаються та зберігаються на серверах компанії. Користувачі повинні знати про умови надання послуги, а також про потенційні ризики та відповідальність.

На закінчення Solarman Dataloggers пропонує потужний інструмент для моніторингу та керування фотоелектричними системами. Незважаючи на те, що вони відрізняються від проєкту deye-logger-at-cmd щодо офіційного доступу та функцій, розуміння їхніх можливостей і обмежень має вирішальне значення для прийняття обґрунтованих рішень під час оптимізації продуктивності вашої сонячної системи. Побачити проєкт у дії можна на рисунку 1.4

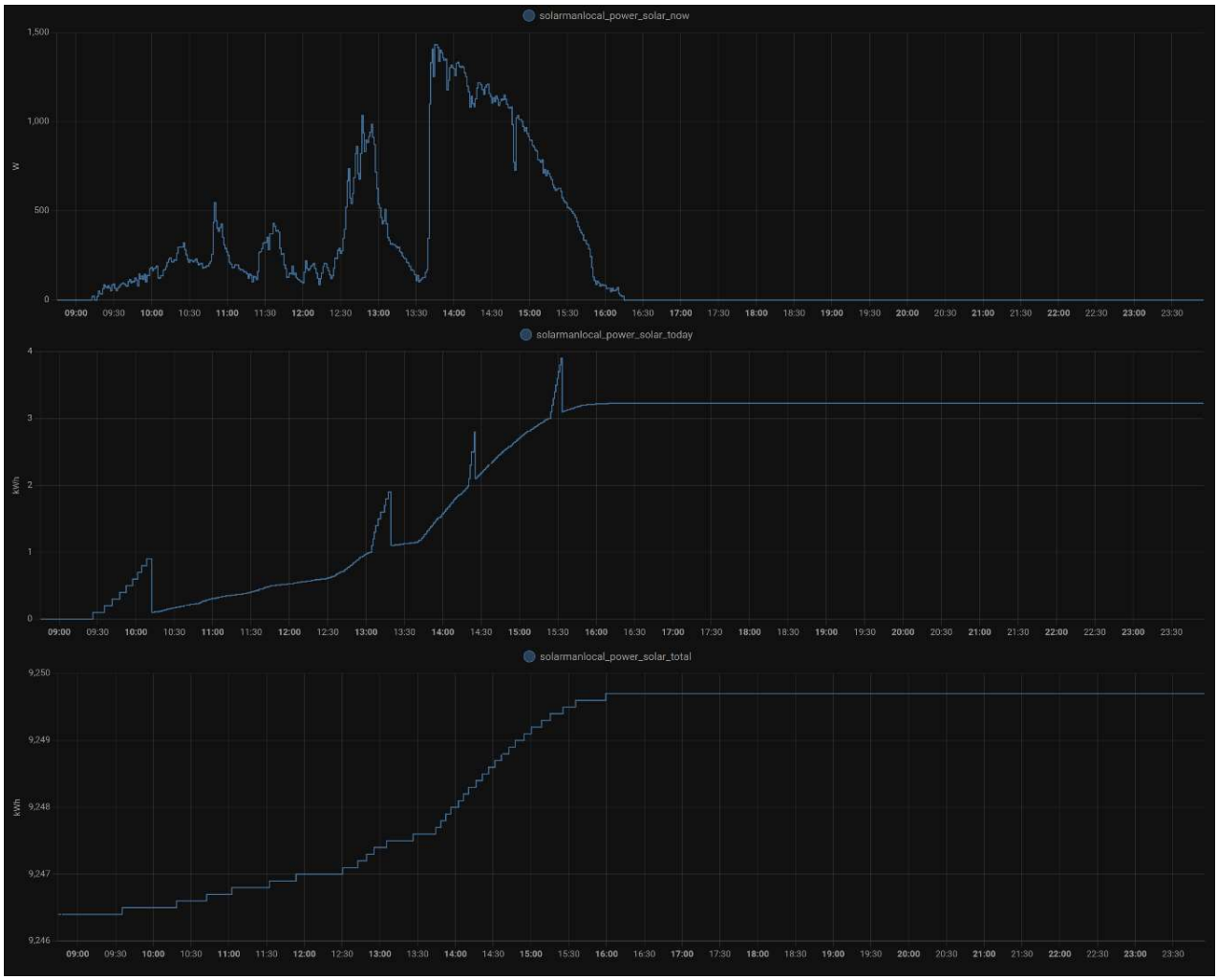


Рисунок 1.4 - Solarman Datalogger у дії

DeyeInverter [12]. DeyeInverter, компанія, що спеціалізується на рішеннях для сонячної енергетики, пропонує різноманітний портфель продуктів, від мікроінверторів для окремих панелей до мережевих інверторів для великих систем. Незважаючи на те, що їхні продукти пропонують ефективне перетворення енергії, розуміння їхніх функцій і потенціалу можна покращити шляхом глибшого дослідження.

На відміну від суперечливого проєкту deye-logger-at-cmd, який зосереджений на обході автентифікації, дослідження DeyeInverter включає авторизовані методи та офіційні ресурси.

Deye пропонує різноманітні інструменти та платформи для отримання цінної інформації, як-от додаток Deye Monitoring, центр обробки даних Deye та API Deye. Ці інструменти дозволяють користувачам відстежувати, аналізувати та контролювати продуктивність, дані та налаштування своєї системи.

Вивчення цих інструментів дає змогу користувачам оптимізувати продуктивність системи, визначаючи періоди пікового генерування, потенційні проблеми із затіненням і регулюючи параметри для максимальної продуктивності. Вони також можуть виявляти несправності та діагностувати проблеми, використовуючи дані в реальному часі та історичні тенденції, щоб визначити проблеми на ранніх стадіях, запобігаючи простою та дорогому ремонту.

Крім того, вони можуть контролювати споживання та витрати енергії, отримуючи уявлення про моделі споживання енергії та оптимізуючи стратегії управління енергією. Також вони можуть інтегруватися з системами «розумного дому», використовуючи дані для автоматизації та контролю в ширшій екосистемі «розумного будинку» [13].

Однак важливо пам’ятати, що DeyeInverter є комерційною організацією, і для деяких розширених функцій може знадобитися придбання додаткового апаратного чи програмного забезпечення.

Варто не забувати про питання безпеки та конфіденційності даних, які завжди актуальні, і розуміння практики обробки даних Deye має вирішальне значення для прийняття обґрунтованих рішень.

Підсумовуючи, DeyeInverter пропонує ряд інструментів і ресурсів, окрім своїх основних продуктів, та детальну документацію до них. Досліджуючи авторизовані методи, користувачі можуть отримати цінну інформацію, оптимізувати продуктивність системи та отримати глибше розуміння генерації сонячної енергії.

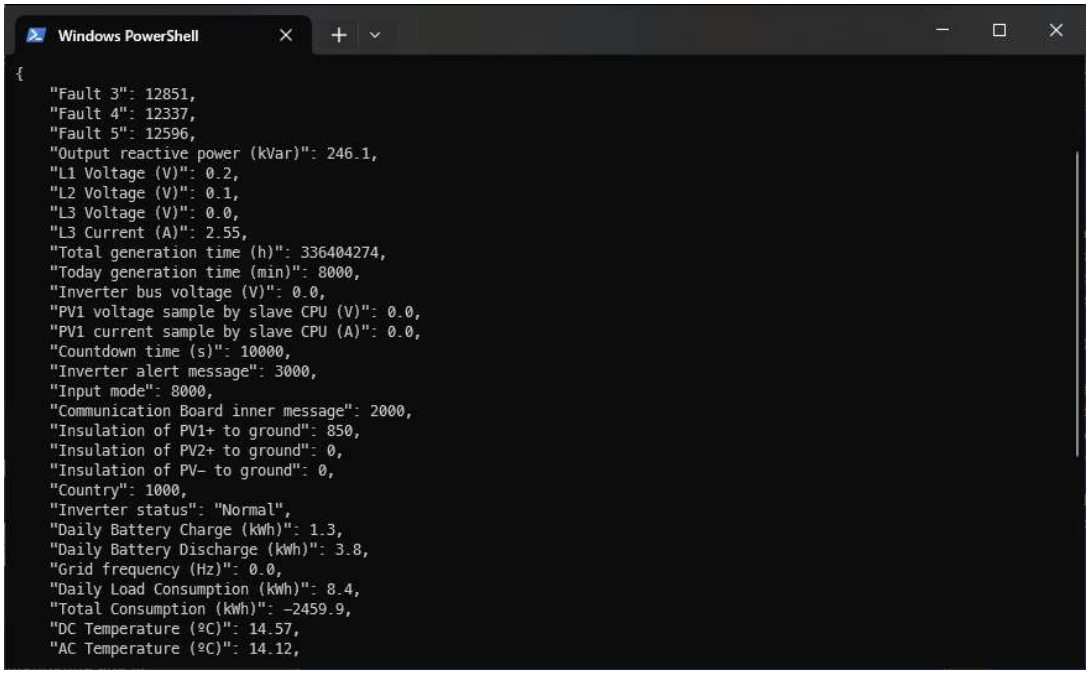


Рисунок 1.5 – DeyeInverter у дії

На основі проведеного аналізу можна зробити висновок, що жодна з представлених опцій не є ідеальною. Кожна з них має свої переваги та недоліки, які необхідно враховувати при виборі оптимального рішення для конкретного проєкту.

Мова програмування: 2/3 проєктів реалізовані на Python [14], що підкреслює його популярність завдяки швидкості розробки та простоті взаємодії з різними компонентами. 1/3 проєктів використовує інші мови програмування, такі як C++ [15] та Java [16], що може бути обумовлено специфічними вимогами до продуктивності або функціональності.

Протокол MODBUSCRC [17]: усі проєкти без винятку використовують протокол MODBUSCRC для забезпечення надійної передачі даних між пристроями. Це свідчить про його загальновживаність та високу надійність у сфері промислової автоматизації.

Графічний інтерфейс: жоден з проаналізованих проєктів не має графічного інтерфейсу, окрім одного, де це було реалізовано веб-інтерфейс. Це може бути пояснено їх орієнтацією на використання: у вбудованих системах, де графічний

інтерфейс не є необхідним; на серверних платформах, де доступ до даних здійснюється через API.

Програмний інтерфейс: 2/3 проєктів не мають програмного інтерфейсу для використання як компоненту. Це може ускладнювати їх інтеграцію з іншими програмами та системами. 1/3 проєктів мають відкритий API, що дозволяє використовувати їх як бібліотеки або модулі в інших проектах.

Крос-платформенність: всі розглянуті проєкти є крос-платформенними, що дає можливість використовувати їх на різних операційних системах без суттєвих змін та проблем. Це робить їх універсальними та зручними для розробки мульти-платформних рішень.

Механізм взаємодії з апаратною частиною: кожен з проєктів використовує власну реалізацію механізму взаємодії з апаратною частиною. Це свідчить про різноманітність підходів до вирішення цієї задачі та підкреслює важливість вибору оптимального рішення з урахуванням специфіки конкретного проєкту, та його задач. Після перегляду усіх цих деталей та визначення плану розробки можливостей програмного забезпечення, можна провести більш детальний аналіз та було складено розширену таблицю порівняння, таблиця 1.1.

Таблиця 1.1 – Порівняння розглянутих проєктів

Критерій	InterServer	deye-logger-at-cmd	Solarman Datalogger	DeyeInverter
1	2	3	4	5
Графічний інтерфейс	✅	❌	✅	❌
Програмний інтерфейс	✅	❌	❌	❌
Підтримка декількох пристроїв	✅	❌	✅	❌

Кіцень таблиці 1.1

1	2	3	4	5
Збереження даних	✅	❌	✅	❌
Якість та гнучкість коду	✅	✅	✅	✅
Моніторинг в реальному часі	✅	✅	❌	✅
Візуалізація даних	✅	✅	❌	✅
Сповіщення про помилки	❌	✅	❌	✅
Автоматичне оновлення прошивки	❌	❌	❌	✅
Підтримка різних типів інверторів	✅	❌	✅	✅
Доступність API	✅	❌	❌	✅
Локалізація інтерфейсу	❌	❌	❌	✅
Відкритий код	✅	✅	✅	✅

Перепланування проєкту вимагало додаткового часу та ресурсів на початковому етапі, але в довгостроковій перспективі це дозволить покращити якість, надійність, та спроможність подальшого розвитку та підтримки проєкту, а також забезпечити його відповідність сучасним стандартам розробки. Зокрема, переоцінка стратегії розробки та вибір оптимальних інструментів та технологій

може значно позитивно позначитися на результативності та ефективності проєкту в майбутньому. Ці критерії були застосовані до чотирьох різних систем: InterServer, deye-logger-at-cmd, Solarman Datalogger та DeyeInverter. Результати порівняння допомогли визначити сильні та слабкі сторони кожної системи, що дозволяє переконатись в необхідності проєкту. Наприклад, виявлення потенційних проблем або обмежень у вже існуючих системах може виявитися критичним для прийняття правильного рішення щодо подальших кроків у розвитку проєкту [18]. Аналіз існуючих рішень виявив, що деякі системи мають обмеження у функціональності або проблеми з інтеграцією, що може призвести до неефективного використання ресурсів або недоліків у забезпеченні безпеки.

Додатково, перепланування дозволило врахувати нові вимоги та тенденції в галузі, що з'явилися під час розробки або в процесі дослідження подібних рішень. Це дозволило включити вдосконалення функціоналу, адаптуватись до змін у вимогах користувачів, та вдосконалити заходи безпеки та захисту даних, які стають все більш важливими в умовах постійно зростаючих загроз в інтернеті та підлеглого обладнання, яке може бути підключено до інтернету. Крім того, нові вимоги включають підвищену потребу у масштабованості системи, що дозволить ефективно обробляти збільшену кількість даних та підключених пристроїв без втрати продуктивності.

Перепланування також враховувало необхідність інтеграції з іншими системами та сервісами, що дозволить створити більш комплексне та гнучке рішення, здатне відповідати потребам різних користувачів. В результаті, проведений аналіз та перепланування стали основою для створення більш досконалої системи, яка не лише відповідає сучасним вимогам, але й готова до майбутніх викликів. Цей підхід дозволив створити фундамент для подальшого розвитку проєкту, забезпечуючи його актуальність та конкурентоспроможність на ринку. Завдяки ретельному аналізу та врахуванню новітніх тенденцій, проект отримав чітке бачення своїх сильних та слабких сторін, що стало ключем до успішного впровадження та подальшого вдосконалення.

1.3 Висновки. Постановка задачі

Перш ніж розпочати роботу над проектом, було ретельно досліджено існуючі рішення для подібних завдань. Це дослідження дало змогу не лише отримати цінні ідеї та зрозуміти складність завдання, але й детально вивчити доступні інструменти та бібліотеки, які могли б бути корисними під час реалізації проєкту.

Аналіз аналогів: Було вивчено широкий спектр існуючих проектів, подібних до нашого. Це дозволило зрозуміти, які підходи до вирішення задачі вже використовуються, які з них є найефективнішими, а які мають недоліки.

Вивчення доступних інструментів: Було досліджено широкий спектр інструментів та бібліотек, які могли б бути корисними під час реалізації проєкту. Це включало в себе як загальні інструменти розробки, так і спеціалізовані бібліотеки, призначені для вирішення конкретних завдань.

Оцінка складності завдання: Дослідження дозволило чітко зрозуміти складність завдання, яке стоїть перед нами. Це включало в себе визначення основних проблем, з якими ми можемо зіткнутися, а також оцінку необхідних ресурсів і часу для реалізації проєкту.

Окрім вищеописаного, дослідження також дало зазначити наступні переваги:

Підвищення впевненості в успіху проєкту: Дослідження дало чітке розуміння складності завдання, доступних інструментів, ресурсів та ризиків. Це значно підвищило нашу впевненість у тому, що проєкт можливо успішно реалізувати.

Підвищення якості проєкту: Дослідження допомогло нам обрати оптимальний шлях для вирішення задачі, враховуючи всі фактори та ризики. Це, в свою чергу, призвело до значного підвищення якості проект

Таким чином ретельне дослідження існуючих рішень для подібних завдань стало важливим фактором успішного завершення проєкту. Це дослідження дало чітке розуміння поставлених задач, доступних інструментів, ресурсів та ризиків, а також допомогло нам обрати оптимальний шлях для їх вирішення.

Наступним кроком у роботі стало проведення детального аналізу та вибору програмних технологій, на основі яких буде реалізовано проєкт. Цей етап включав

ретельне вивчення доступних технологічних рішень, враховуючи їхні можливості, переваги та недоліки. Зокрема заплановано розглянуто мови програмування, фреймворки, типи баз даних та інші інструменти, необхідні для розробки та впровадження проєкту. Після цього буде детально описані всі етапи плану реалізації проєкту, включаючи оцінку можливих недоліків та потенційних проблем, що можуть виникнути під час впровадження.

Після проведення ретельного аналізу, наступним кроком для проєкту стало планування формування самої структури проєкту, яка б надавала загальний опис усіх його складових та їх взаємодії. Цей крок є надзвичайно важливим, оскільки він дозволятиме чітко зрозуміти, з яких компонентів та складових повинна складатися система, щоб досягти поставлених цілей. Було заплановано визначити основний функціонал проєкту, який включатиме у себе ключові компоненти та складові, необхідні для досягнення цих цілей. Крім того, також необхідно буде приділити особливу увагу опису взаємодії між цими компонентами, щоб забезпечити цілісність та узгодженість роботи системи. Цей етап планування дозволить створити чіткий та детальний план дій, який буде керувати всіма наступними кроками розробки.

Після цього було заплановано створено макет підлеглого кіберфізичного пристрою, призначеного для взаємодії з проєктом через раніше заплановане використання мережі та розробку системи кінцевих точок програмного інтерфейсу проєкту. Було заплановано функціонал пристрою для відображення поточного стану блоку керування фотоелектричними пристроями, використовуючи дані отримані через мережу та кінцецеві точки проєкту у реальному часі, що має дозволити ретельно перевірити та оптимізувати роботу системи в умовах, наближених до реальних. Такий підхід має значно підвищити надійність та ефективність проєкту, забезпечивши його відповідність сучасним вимогам та стандартам.

2 ПЛАНУВАННЯ ТА РЕАЛІЗАЦІЯ ПОСТАВЛЕНОЇ ЗАДАЧІ

2.1 Аналіз та вибір програмних технологій

Після проведення початкової оцінки та аналізу, даний проєкт пропонує ряд суттєвих переваг порівняно з іншими розглянутими альтернативами. Покращена реалізація коду сприяє полегшенню впровадження нових функцій без необхідності коригування інших компонентів системи. Це підвищує загальну ефективність та адаптивність. Відстеження в режимі реального часу є ключовим аспектом для прийняття оперативних та обґрунтованих операційних рішень [19]. Це дозволяє користувачам спостерігати за роботою сонячної електростанції у реальному часі. Візуалізація даних перетворює робочі дані електростанції на графіки та діаграми, що полегшує їх аналіз та розуміння.

Сповіщення про помилки дозволяють системі надсилати повідомлення про помилки та несправності, що забезпечує швидке реагування на потенційні проблеми та мінімізує час простою, підвищуючи продуктивність. Автоматичне оновлення мікропрограми гарантує актуальність програмного забезпечення, надаючи користувачам найновіші функції та оновлення безпеки. Підтримка різноманітних типів інверторів розширює можливості застосування системи, роблячи її більш універсальною та адаптованою до різноманітних потреб користувачів. Доступність API дозволяє інтегрувати систему з іншими пакетами програмного забезпечення, що розширює її функціональні можливості та застосування. Відкритий вихідний код створює можливість спільноті розробників внести внесок у вдосконалення системи, сприяючи постійному розвитку та інноваціям.

Враховуючи ці фактори, було прийнято рішення реалізувати проект з використанням C#, мови програмування високого рівня, розробленої для комп’ютерних і системних додатків. До цього розглядалися такі мови системного програмування, як JavaScript [20] і його низькорівнева реалізація Node.JS [21], Python, що використовує ті самі бібліотеки, що й ті, що використовувалися в

розглянутих раніше аналогах, і PHP [22]. Однак жоден із цих варіантів не забезпечив необхідного рівня зручності, функціональних властивостей і документації, необхідної для вибору мови програмування для реалізації проєкту.

Отож, було визначено основні переваги обраної мови програмування – C#:

- комплексний аналіз вищезазначених атрибутів існуючих рішень у поєднанні з унікальними вимогами проєкту привів до рішення використовувати мову програмування C#;
- C# є кросплатформною мовою, що робить її зручною для розробки програми для будь-якої сучасної системи та підтримуваної операційної системи [23];
- C# відомий своєю високою швидкістю обробки[8], що робить його ідеальним вибором для завдань, які вимагають високої продуктивності та швидкої обробки інформації;
- C# характеризується зручним синтаксисом та широким набором бібліотек, компонентів і стандартних функцій [24], що значно спрощує процес розробки;
- C# пропонує високий ступінь гнучкості та масштабованості, що робить його ідеальним вибором для проектів, які потребують майбутнього розширення та розвитку.

На додаток до вищезазначених переваг, подальше дослідження показує безліч переваг, які зміцнюють позицію проєкту як піонерського рішення в царині управління сонячною енергією:

Масштабованість і гнучкість. Архітектура проєкту дозволяє легко масштабувати роботу, враховуючи зростання потужності або технологічний прогрес з мінімальними збоями [25]. Його гнучкий дизайн гарантує, що він може адаптуватися до мінливих вимог, не вимагаючи значних зусиль з реконструкції.

Аналітика та прогноз: використовуючи розширені алгоритми, система може прогнозувати потенційні проблеми з продуктивністю або вимоги до обслуговування на основі історичних даних і поточних умов експлуатації. Цей

проактивний підхід дозволяє вживати попереджувальні заходи для оптимізації продуктивності та зменшення простоїв.

Віддалена діагностика: за допомогою можливостей віддаленого доступу технічні спеціалісти можуть діагностувати та усувати проблеми без необхідності фізично бути присутнім на місці. Ця можливість не тільки скорочує час відгуку, але й мінімізує ризик виникнення збоїв в роботі та пов’язані з цим витрати.

Оптимізація енергії. Аналізуючи дані від різних датчиків і компонентів, система оптимізує моделі виробництва та споживання енергії в режимі реального часу. Ця оптимізація забезпечує максимальне використання наявних ресурсів при мінімізації відходів, що в кінцевому підсумку покращує загальну ефективність і економічну ефективність сонячної електростанції.

Відповідність нормативним вимогам, через які відбувається розширене використання вбудованих функції, які сприяють дотриманню нормативних стандартів і вимог до звітності, спрощуючи адміністративні аспекти управління сонячною електростанцією. Від моніторингу викидів до управління документацією, система спрощує дотримання галузевих норм, зменшуючи ризик отримання штрафів за їх недотримання.

Покращені заходи безпеки, де було використано надійні протоколи шифрування та засоби контролю доступу захищають конфіденційні дані та критичні компоненти інфраструктури від несанкціонованого доступу чи кіберзагроз [26]. Цей підвищений рівень безпеки вселяє довіру серед зацікавлених сторін і захищає цілісність усієї енергетичної екосистеми.

Повна інтеграція з пристроями Інтернету речей, яка забезпечує здатність проєкту легко інтегруватись з широким спектром пристроїв Інтернету речей (IoT), що дозволяє здійснювати комплексний моніторинг і контроль допоміжного обладнання, такого як датчики погоди, інвертори та системи зберігання акумуляторів. Ця сумісність покращує загальну видимість і управління усією екосистемою сонячної енергії.

Дизайн, орієнтований на користувача: віддаючи перевагу взаємодії з користувачем, проект має інтуїтивно зрозумілий інтерфейс, який дає змогу операторам і адміністраторам ефективно орієнтуватися в складних наборах даних і функціях. Від налаштованих інформаційних панелей до персоналізованих сповіщень, кожен аспект інтерфейсу користувача адаптовано для оптимізації продуктивності та прийняття рішень.

Об'єднуючи ці різноманітні переваги, проект не тільки революціонізує управління сонячною енергією, але й встановлює новий стандарт для технологічних інновацій та сталого розвитку в енергетичному секторі. Беручи до уваги ці фактори, було прийнято рішення реалізувати проект з використанням C#, мови програмування високого рівня, призначеної для комп'ютерних і системних додатків. До цього розглядалися такі мови системного програмування, як JavaScript, та його низькорівнева реалізація - Node.JS, Python, що використовує ті самі бібліотеки, що й ті, що використовувалися в розглянутих раніше аналогах, і PHP . Однак жодна з цих опцій не забезпечувала необхідного рівня зручності, функціональних властивостей і документації, необхідної для вибору мови програмування для реалізації проєкту.

Мова програмування C# була обрана через її здатність легко виконувати завдання проєкту, але не є обмеженою у цьому функціоналі [27]. Головні цілі та складні етапи розробки проєкту, які було заплановано вирішити за допомогою обраної мови програмування, через її переваги над розглянутими аналогами:

Інтеграція Entity Framework [28] у проект забезпечує безперебійне включення інформації. Ця структура спрощує керування та організацію даних, роблячи процес більш ефективним і оптимізованим. Також це спрощує підтримку різних типів баз даних, що є необхідно у проєктах, ціллю яких є підтримка великої кількості різноманітного обладнання [29].

Обмін даними між проєктом та інвертором заплановано реалізувати через протокол обміну інформацією TCP, що забезпечить надійність і швидкість передачі даних. Важливим аспектом цього рішення є те, що більшість необхідних

компонентів для реалізації комунікації пристроїв вбудовані в саму мову програмування, яку було обрано для розробки проєкту – C#. Цей підхід усуває потребу в додатковому програмному забезпеченні, бібліотеках чи інструментах, що значно спрощує процес розробки та підтримки проєкту в майбутньому.

Завдяки вбудованим засобам для роботи з мережевими протоколами в C#, процес налаштування і підтримки мережевих з'єднань стає більш інтуїтивним і зручним для розробників. Це дозволяє зосередитися на реалізації основної функціональності системи, не витрачаючи значні ресурси на вирішення проблем з мережевою комунікацією.

Для обміну інформацією по протоколу HTTP, на якому буде побудована реалізація програмного інтерфейсу, та функцій, пов'язаних з веб-технологіями, є фреймворк ASP.NET, який пропонує повний набір інструментів і функцій для розробки веб-додатків. ASP.NET є ідеальним вибором для даного проєкту, де потреба в надійної веб-функціональності є однією з головних задач.

Наявність перевіреної документації до раніше вказаних компонентів також є суттєвою перевагою. Це надає можливість отримати усі необхідні відомості про компонент без необхідності у пошуку джерел інформації про вказаний компонент, з сумнівних та застарілих джерел. Це є особливо важливо при роботі з сучасними версіями програмних продуктів, адже їх оновлення зазвичай вимагають суттєвих змін у кодовій базі програм. Але через технологічного гіганта Microsoft, який є відповідальним за більшість зазначених компонентів це не є проблемою для мови програмування C#, та є проблемою для більшості інших розглянутих опцій.

Тому враховуючи усі перераховані переваги, мова програмування C# була обрана, тому що вона пропонує потужний і універсальний набір інструментів для вирішення широкого кола завдань, від консолідації даних до обміну інформацією за допомогою різних протоколів. Вбудовані компоненти мови програмування C# та фреймворки роблять процес розробки більш ефективним і спрощеним, а підтримка веб-технологій робить її ідеальним вибором для проектів, які потребують надійної веб-функціональності [30].

2.2 Опис реалізації етапів проєкту

З самого початку проєкту було визначено, що консольна програма, побудована на C#, слугуватиме веб-сервером, забезпечуючи інтерфейс прикладного програмування (API) для доступу до даних із зовнішніх пристроїв, не пов’язаних із проектом, таких як база даних, яка накопичує інформацію і дані безпосередньо з інвертора. Такий підхід полегшить зручний доступ до даних проєкту та функціональності з будь-якого пристрою, підключеного до мережі [31].

Для реалізації графічної складової проєкту планувалося використовувати простий HTML-файл, що містить лише мінімально необхідну інформацію [32]. Більшість вмісту файлу буде заповнено після запиту IP-адреси проєкту через браузер і надіслано у вигляді веб-сторінки. Такий підхід забезпечить швидкий і легкий доступ до основних функцій без непотрібних затримок або перевантажень, які можуть виникнути під час завантаження важких графічних елементів. Однак цей підхід також накладає значні обмеження на вбудований графічний інтерфейс проєкту, обмежуючи його однією сторінкою з інформацією. На тому етапі планування та розробки було невідомо, чи можливо буде включити допоміжні компоненти сучасних веб-сайтів, такі як JavaScript і CSS [33], із такою реалізацією веб-інтерфейсу.

Щоб досягти цього, консольна програма C# буде використовувати структуру ASP.NET Core [34] для створення легкого та масштабованого веб-сервера. Ця структура надає ряд інструментів і функцій для створення веб-додатків, включаючи підтримку протоколів HTTP, маршрутизацію та проміжне програмне забезпечення. Використовуючи ці можливості, консольна програма може відкрити API, який дозволяє зовнішнім пристроям взаємодіяти з даними та функціональними можливостями проєкту.

Крім того, для заповнення файлу HTML динамічним вмістом консольна програма може використовувати механізм шаблонів Razor [35], який також є частиною фреймворку ASP.NET Core. Цей механізм дозволяє генерувати HTML-

сторінки на льоту на основі шаблонів і моделей даних, що дає змогу створювати адаптивний і керований даними графічний інтерфейс.

Незважаючи на обмеження, пов’язані з односторінковим інтерфейсом, все ще можна використовувати сучасні веб-технології, такі як JavaScript і CSS, для покращення взаємодії з користувачем. Наприклад, консольна програма може обслуговувати статичні файли, що містять код JavaScript і CSS, які можуть бути завантажені сторінкою HTML для надання додаткових функцій і стилів. Цей підхід дозволить створити складніший графічний інтерфейс, зберігаючи при цьому переваги легкого та ефективного веб-сервера [36].

Реалізація програмного інтерфейсу для можливості взаємодії теж була невирішеною проблемою. На той момент не існувало механізму для коректної та масштабованої побудови програмного інтерфейсу. Тому було сплановано втілити це шляхом зазначення абсолютного мінімуму кінцевих точок, де тип відповіді визначався би через параметр у запиті до кінцевої точки.

Після ретельного розгляду було визначено наступні відмінні характеристики проєкту:

- модульність: проект буде розроблено з акцентом на модульність, щоб полегшити додавання нових функцій і можливостей у майбутньому. Такий підхід дозволить легко розширити проект і адаптувати його до мінливих вимог;
- документація: проектна документація не буде інтегрована в сам проект. Натомість усі посилання на проектну документацію будуть надані як посилання на зовнішні ресурси. Цей підхід вимагатиме додаткових ресурсів для підтримки двох окремих компонентів одного проєкту;
- безпека: через унікальну природу більшості компонентів проєкту безпеку не можна гарантувати. Забезпечення належної безпеки призведе до додаткових складнощів, витрат часу та ресурсів.

Після кількох днів активної розробки та ретельного тестування практичних можливостей проєкту стало зрозуміло, що обрана архітектура не виправдала

очікувань. Він не тільки не зміг гарантувати необхідний рівень простоти використання, але й створив додаткові ускладнення в процесі розробки.

Щоб вирішити ці проблеми, архітектуру проєкту було перероблено з увагою на вбудовані можливості C#, та пов'язані з ним інструменти та фреймворки. Модульну конструкцію було збережено, але проект було перебудовано для використання більш надійної та масштабованої архітектури на основі бібліотек і компонентів C#. Цей підхід дозволив використати потужність і гнучкість C#, зберігаючи при цьому переваги модульної конструкції [37].

Для задоволення вимог до документації було створено окреме сховище документації за допомогою популярного інструменту для створення документації. Цей репозиторій містить усю проектну документацію, включаючи посилання на API, посібники користувача та технічні характеристики. Документація створюється автоматично з коментарів у коді та інших файлах документації, що гарантує, що вона завжди актуальна та точна.

Щоб вирішити проблеми з безпекою, було реалізовано комплексну систему безпеки з використанням стандартних протоколів шифрування та механізмів автентифікації. Ця структура забезпечує надійні функції безпеки, такі як безпечна автентифікація користувачів, шифрування даних і контроль доступу, щоб забезпечити захист проєкту від несанкціонованого доступу та порушень даних [38].

Таким чином, завдяки використанню потужності та гнучкості C# та пов'язаних із ним інструментів і фреймворків архітектуру проєкту було перероблено для вирішення проблем, виявлених під час розробки. Та в процесі планування було збережено модульний дизайн, а вимоги до документації та безпеки вирішувалися за допомогою окремих компонентів, які інтегровані в основний проект.

Цей підхід дозволив проєкту відповідати необхідному рівню простоти використання, а також забезпечував надійні функції безпеки та актуальну документацію, без потреби на розділення на окремі складові проєкту, при яких би документація була би реалізована окремим, та незалежним компонентом.

Основні недоліки початкової архітектури:

- обмежені можливості API**:** Функціонал API не був достатнім для реалізації всіх необхідних завдань, що ускладнювало роботу з даними та інтеграцію з іншими системами [39];

- негнучкість: Архітектура не передбачала можливості гнучкого налаштування та адаптації до мінливих потреб проєкту;

- невідповідність консольного додатку: Використання консольного додатку як початковий шаблон не надавало суттєвих переваг, змушуючи втілювати усі складові компоненти, які вже було втілені у різних фреймворках раніше, що ускладнювало подальше існування та розробку проєкту.

Враховуючи виявлені недоліки та потреби проєкту, було прийнято рішення про кардинальну зміну архітектури та підходу до розробки. Найбільшим фактором зміни архітектури проєкту стало вивчення фреймворку ASP.NET. До цього моменту цілями проєкту було створення компонентів, функціонал яких вже реалізовано у фреймворку ASP.NET, та було вирішено, що це є марною тратою часу та ресурсів. Таким чином було вирішено припинити розробку вже існуючого консольного додатку, та почати новий додаток з використання шаблону, в якому заздалегідь було підключено, та налаштовано фреймворк ASP.NET [40].

В подальшій розробці було прийнято рішення також змінити архітектуру схожою до мікросервісної, через її суттєві переваги у даному проєкті. Компоненти з попереднього консольного проєкту було збережені, для можливості пере використання їх у вигляді компонентів, або сервісів проєкту, ти самим зменшуючи потребу у написанні нового коду.

Одним із основних елементів переглянутого підходу є прийняття мікросервісної архітектури [41]. Цей підхід передбачає декомпозицію системи на окремі автономні служби, кожна з яких відповідає за певну функціональність.

Переваги мікросервісної архітектури включають у себе багато особливостей, та найголовніші з них, які стануть у нагоді розробки проєкту висвітлено нижче.

Спрощена розробка та обслуговування: розділивши систему на служби, процес розробки та обслуговування стає більш простим і ефективним. Кожну службу можна розробити, протестувати та розгорнути незалежно, зменшуючи складність і накладні витрати, пов’язані з монолітними архітектурами [42].

Масштабованість: кожну службу можна масштабувати незалежно, що забезпечує більш гнучку реакцію на зміну вимог до навантаження. Такий підхід дозволяє більш ефективно використовувати ресурси та покращувати продуктивність.

Гнучкість: мікросервіси можна легко інтегрувати з іншими системами та пристроями, забезпечуючи більшу взаємодію та гнучкість. Цей підхід дозволяє створювати більш модульні та адаптовані системи, які можна легко розширити та налаштувати.

Прийнявши архітектуру мікросервісів у C#, надасть можливість створити більш масштабовані, гнучкі та зручні в обслуговуванні системи, які можна легко інтегрувати з іншими системами та пристроями. Цей підхід дозволяє нам розробляти більш модульні та адаптовані системи, які можна легко розширити та налаштувати відповідно до мінливих вимог.

Перехід до мікросервісної архітектури та впровадження інших сучасних технологій дозволять значно покращити гнучкість, зручність використання, продуктивність та масштабованість проєкту.

Та було прийнято рішення, що переосмислення архітектури та впровадження нових технологій дозволять проєкту досягти поставлених цілей та стати ефективним інструментом для вирішення поставлених завдань.

Відмова від початкової архітектури обумовлена наступними критеріями:

1) неефективна реалізація веб-інтерфейсу:
    - обмежена динамічність та складність оновлення контенту;
    - незручність для користувачів, відсутність інтерактивності;
    - неможливість використовувати сучасні веб-технології та бібліотеки;

- жорстка прив'язка логіки та представлення, що ускладнює розробку та тестування;

2) складність реалізації API:
   - обмежена можливість інтеграції з іншими системами та пристроями;
   - необхідність розробки та підтримки власних протоколів обміну даними;
   - відсутність масштабованості та гнучкості;
3) відсутність готових рішень:
   - необхідність самостійної розробки базових механізмів, таких як аутентифікація, авторизація, кешування;
   - високі витрати часу та ресурсів на розробку та підтримку;
   - відсутність гарантії стійкості та безпеки.

Після проведення поглибленого аналізу виявлених проблем та участі в широких дискусіях було прийнято рішення переглянути значну частину проєкту. Цей крок було визнано необхідним для забезпечення довгострокової життєздатності проєкту, оптимізації використання ресурсів і кращого узгодження його з сучасними стандартами розвитку.

Обґрунтування перепланування включає перехід до більш широко використовуваних бібліотек і компонентів, що розширить коло потенційних розробників, які могли б зробити внесок у проект, і спростить пошук допомоги та інформації. Крім того, це підвищить сумісність з іншими системами та платформами. Іншою причиною є підвищення ефективності та простоти розробки, які пропонують оновлені бібліотеки та компоненти, забезпечуючи кращу продуктивність, гнучкість та інструменти, що робить процес розробки більш динамічним та ефективним. Нарешті, перепланування допоможе зменшити технічний борг, оскільки застарілі технології можуть призвести до його накопичення, негативно вплинувши на швидкість, надійність і масштабованість проєкту [43].

Новий стек технологій включає .NET версії 7.0 [44], яка надає доступ до нових функцій і можливостей, таких як покращена підтримка хмарних середовищ, DevOps і контейнеризація. ASP.NET Core, сучасна структура для розробки веб-додатків, також включена і характеризується високою продуктивністю, масштабованістю та модульністю. Entity Framework Core, ORM-фреймворк, є частиною нового стеку і спрощує роботу з базами даних, роблячи код більш чітким і зрозумілим. Нарешті, Docker [45], платформа для контейнеризації, включена для спрощення розгортання та масштабування проєкту [46].

Новий стек технологій пропонує кілька переваг, зокрема більшу гнучкість і швидкість розробки завдяки використанню сучасних інструментів і бібліотек. Це також покращує якість і надійність програмного забезпечення, оскільки нові технології гарантують кращу продуктивність, безпеку та стійкість до пошкоджень. Крім того, новий стек зменшує витрати на розробку та обслуговування завдяки кращій екосистемі та документації, а також зменшенню технічної заборгованості. Нарешті, він пропонує значно кращу масштабованість та здатність проєкту розширюваність, що дозволяє легко масштабувати проект і додавати нові функції в майбутньому.

Також новий стек технологій суттєво полегшить задачу реалізації логіки роботи проєкту. Адже завдяки використанню ASP.NET буде набагато легше реалізувати обмін інформацією через протокол HTTP [47]. Це є особливо важливо через те, що комунікація проєкту з інверторами відбуватиметься через протокол TCP, інструменти роботи з яким вбудовані у загальні бібліотеки для виконання програм написаних на C#. Та новий стек технології значно спростить комутацію трафіку з одного протоколу в інший, при перетворенні інформації на льоту.

Переосмислення архітектури проєкту вимагало значних додаткових зусиль, часу та ресурсів на початковому етапі. Незважаючи на ці витрати, цей процес був необхідним для забезпечення високої якості, надійності та стійкості кінцевого продукту. Оновлена архітектура враховує новітні технологічні тенденції та

стандарти, що дозволить проекту залишатися актуальним і конкурентоспроможним упродовж тривалого часу.

Отже, вкладення в переосмислення архітектури на початку розвитку проекту є обґрунтованим і стратегічно важливим кроком, який забезпечить успішність та довговічність проекту в сучасних умовах.

2.3 Структура проєкту та його основний функціонал

Після того, як було проведено початкову реалізацію, та внесено ряд змін, було почато повторну реалізацію проєкту з корективами, які було визначено раніше. Та першим кроком розробки стала реалізація механізму комунікації з інвертором. При реалізації механізму комунікації однією з найважливіших частин було створення правильного TCP запиту на інвертор[18].

Побудова запиту для отримання інформації з інвертора відбувається шляхом читання файлу конфігурації, який був сформований раніше. Файл конфігурації містить у собі усі необхідні регістри, які використовуються у ролі вказівника на ділянку інформації у двійковому просторі. Також файли конфігурації містять у собі регістри початку та кінця обробки інформації з отриманого двійкового пакету інформації.

При побудові некоректного запиту на інвертор у відповідь буде відправлено пакет інформації, який містить у собі службове повідомлення про неправильний запит, та порожню ділянку, яка має містити у собі інформацію. Збір запиту для інвертора відбувається шляхом операції з серійним номером у двійковому просторі [48].

Після побудови коректного запиту для інвертора та надсилання його шляхом відправлення TCP запиту приходить відповідь у вигляді пакету з інформацією, який містить у собі службове повідомлення про успішну операцію, та значення показників.

Деякі значення показників можуть містити у собі інформації більше ніж 1 байт, тому усі показники неможливо передати одним TCP пакетом, та по цій

причині отримання інформації відбувається у декілька запитів. Щоб отримати усі показники з даної моделі інвертора необхідно зробити 3 запити.

При тому на експериментальній фазі було визначено, що останній, 3 запит інформації не є важливим через насичення технічною інформацією з інвертора, на яку немає належної документації, та неможливо розібрати у правильні значення. Та через це було вирішено обмежитись двома запитами інформації.

Після успішного впровадження механізму отримання та обробки інформації від інвертора, було зроблено зусилля для вдосконалення існуючого рішення обробки інформації.

Під час пошуку варіантів покращення були виявлені конфігураційні файли для розбору інформації з інверторів. Ці файли пропонували вищу якість опису та підтримували більший діапазон пристроїв порівняно з файлами конфігурації, які використовувалися раніше.

Однак проста заміна старих конфігураційних файлів новими була неможливою через відмінності у їхньому форматі. Попередні файли конфігурації були у форматі JSON, тоді як нові файли були записані у форматі YAML. Ця невідповідність зумовила необхідність використання зовнішньої бібліотеки YamlDotNet для роботи з новими конфігураційними файлами.

Хоча додавання залежності від зовнішньої бібліотеки збільшило складність проєкту, воно спростило процес роботи з файлами конфігурації та забезпечило більшу гнучкість. Використання конфігураційних файлів YAML дозволило створити більш просту та зрозумілу людині конфігурацію, полегшивши додавання нових пристроїв і зміну існуючих конфігурацій [49].

Крім того, бібліотека YamlDotNet надала додаткові функції, такі як підтримка складних типів даних, прив’язок і псевдонімів, що забезпечило більш ефективну та гнучку конфігурацію. Це зробило проект більш адаптованим до мінливих вимог і дозволило підтримувати більш широкий спектр пристроїв і варіантів використання.

Для забезпечення сумісності з новими конфігураційними було необхідно внести зміни до алгоритму роботи з ними. Це було втілено за допомогою

використання бібліотеки YamlDotNet для аналізу та обробки файлів конфігурації YAML. Оновлений алгоритм було перевірено у окремому проєкті, щоб переконатися у його роботоспроможності, та інформація з конфігураційні файлів сумісна з попередньою реалізацією.

Загалом, прийняття конфігураційних файлів YAML і бібліотеки YamlDotNet забезпечило більш гнучке та ефективне рішення для роботи з конфігураційними файлами. Це дозволило проєкту підтримувати більш широкий діапазон пристроїв і варіантів використання, одночасно спрощуючи процес налаштування.

Хоча додавання зовнішньої залежності збільшило складність проєкту, переваги використання конфігураційних файлів YAML і бібліотеки YamlDotNet переважили недоліки, що зробило це цінним доповненням до проєкту [50].

До цього моменту була проведена робота лише з частиною, яка не взаємодіє з користувачем напряму, та більшість значень для роботи з інвертором, такі для IP-адреса та порт інвертора було задані константами у коді програми.

Тому було направлено зусилля для побудови веб-інтерфейсу, через який би можна було би налаштувати проєкт, та отримати як поточну інформацію з інвертора, так і інформацію за минуле використання програмно-апаратного комплексу, наприклад за тиждень використання.

Зразок веб-інтерфейсу для відображення поточної інформації з інвертора можна побачити на рисунку 2.1. До моменту відображення інформації через веб-інтерфейс було використано API проєкту, а саме кінцеві точки /get-data та /test, при тому що існувало всього 3 кінцеві точки API:

- /get-data – повертає поточну інформацію з інвертора;
- /get-cache – повертає кешовану інформацію з інвертора, яку було отримано раніше;
- /test – кінцева точка для перевірки компонентів, та експериментів.

Інформаційну таблицю на головній сторінці підключено до кінцевої точки API /get-data, щоб забезпечити візуальний огляд поточного стану пристрою. Цю кінцеву точку було обрано, оскільки на той час це була єдина робоча кінцева

точка API, яка не змінювала свою функціональність залежно від змін у проекті. Кінцева точка /get-data надає дані в реальному часі з пристрою, включаючи поточну вихідну потужність, напругу та інші відповідні показники. Ця інформація відображається у вигляді таблиці на головній сторінці, що дозволяє легко контролювати продуктивність пристрою з першого погляду [51].

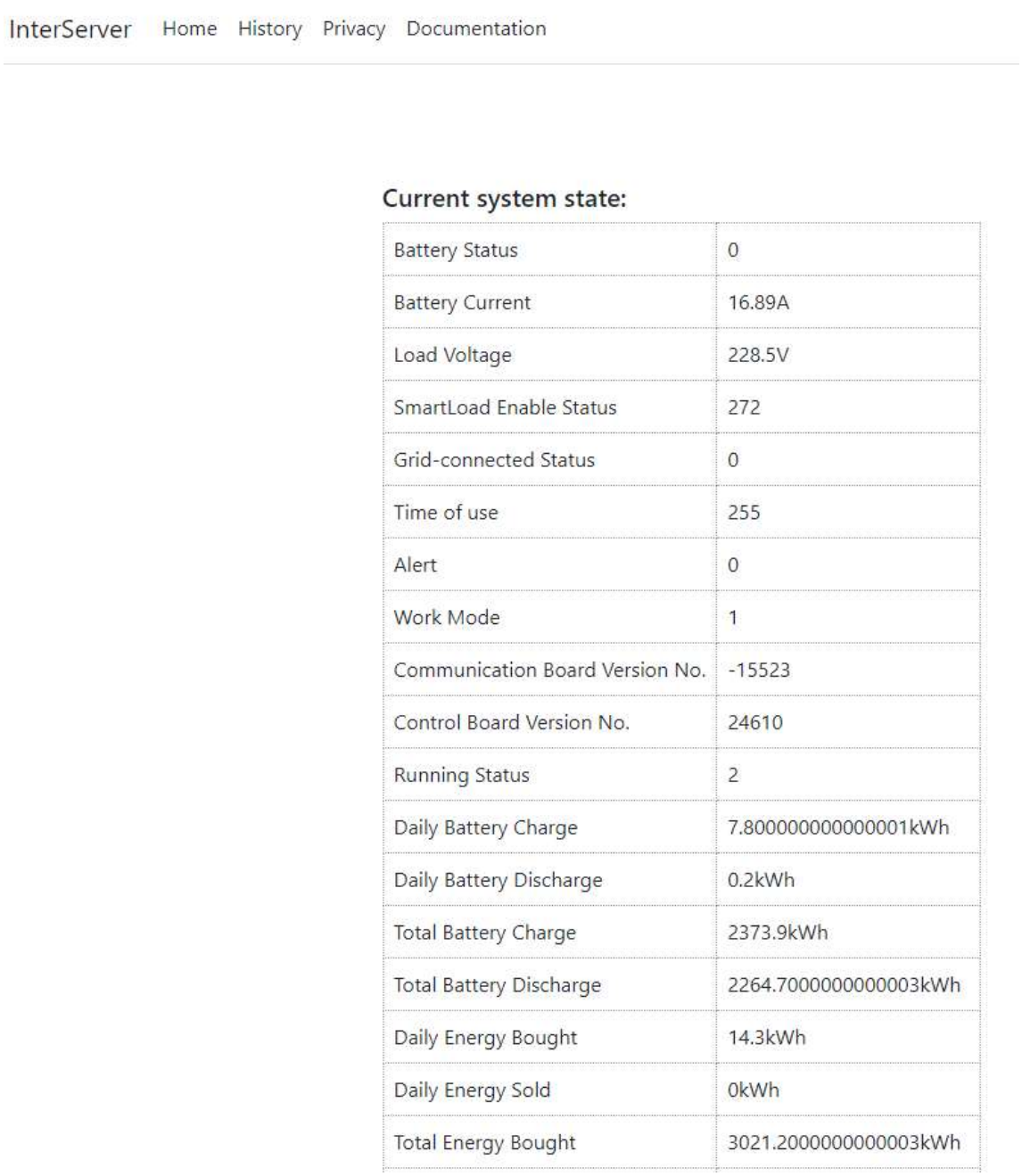


Рисунок 2.1 – Головна сторінка веб-інтерфейсу

Хоча кінцева точка API /test на той час працювала, її не було обрано для інформаційної таблиці через її нестабільну природу. Кінцева точка /test була розроблена для тестування і не призначена для використання у виробництві. Її функціональність може бути змінена без попередження, та дана кінцева точка не є стабільним джерелом інформації, щоб використовувати її як джерело інформації для головної сторінки.

З іншого боку, кінцева точка API /get-cache була створена на дуже ранній стадії розробки й не отримала свого впровадження. Протягом усього часу існування вона повертав службову інформацію про те, що ця кінцева точка не була завершена. У результаті цю кінцеву точку було вирішено не придатним варіантом для інформаційної таблиці.

Щоб забезпечити надійність і точність інформації, що відображається на головній сторінці, було вирішено зосередитися на кінцевій точці API /get-data. Ця кінцева точка була ретельно протестована та підтверджена, щоб переконатися, що вона надає точну та актуальну інформацію про продуктивність пристрою. Також було запроваджено обробку помилок і перевірку інформації, що надсилається, щоб переконатися, що інформація, яка відображається на головній сторінці, завжди точна та надійна [52].

Загалом підключення інформаційної таблиці на головній сторінці до кінцевої точки API /get-data забезпечило зручний і надійний спосіб моніторингу продуктивності пристрою. Завдяки відображенню даних у реальному часі в табличному форматі була отримана можливість швидко виявити будь-які проблеми чи аномалії та вжити відповідних заходів для їх усунення. Це підвищило загальну надійність проєкту та полегшило керування та обслуговування [53].

Наступним етапом у реалізації проєкту стало повторне планування архітектури кінцевих точок API, через їх обмежену кількість та обмежену функціональність. Тому для досягнення мети проєкту було змінено структуру API в загальному. Початкова ідея була розробити усі точки на рівні /api/v1/*кінцева-точка*, але відносно швидко при розробці стало ясно, що така структура не є

оптимальною, та відрізняється від загальноприйнятих стандартів побудови API. Тому було винести усі кінцеві точки API у своєрідну ієрархію кінцевих точок, групуючи їх, та прибираючи потребу у використанні декількох слів у назві кінцевої точки. Нову ієрархію кінцевих точок можна побачити на рисунку 2.2.

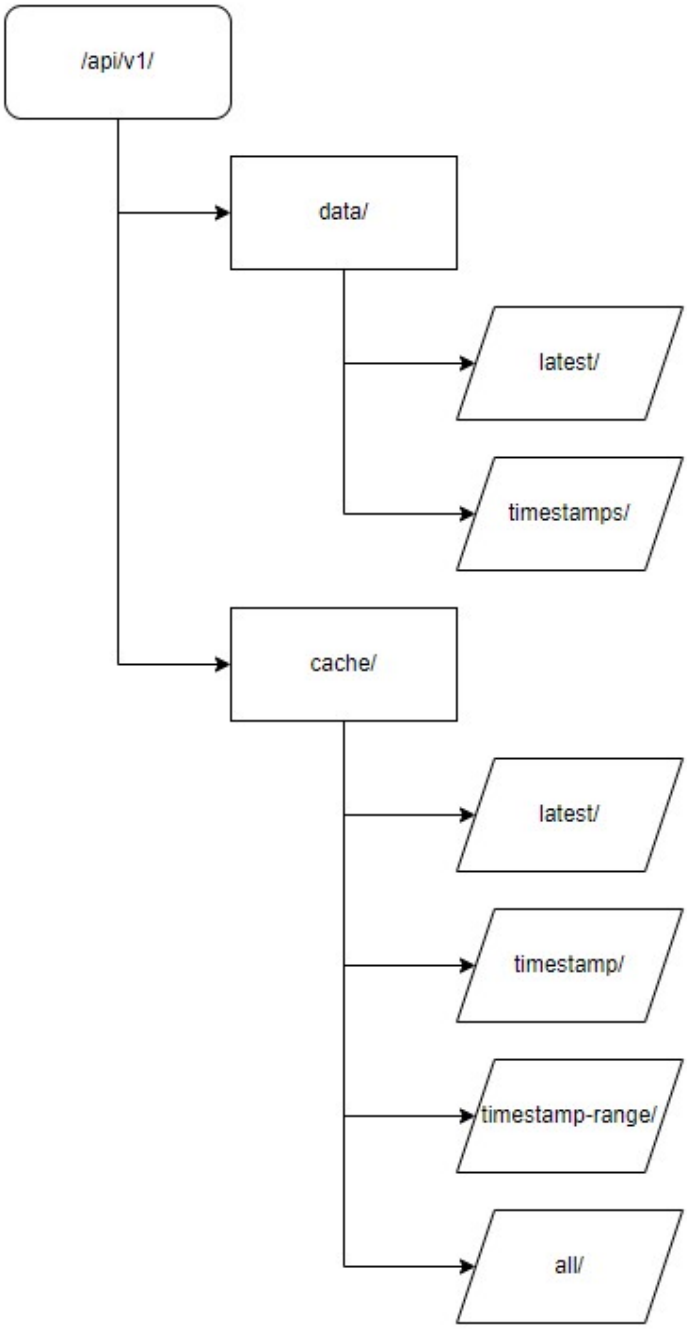


Рисунок 2.2 – Оновлена структура API

Наступним етапом після перепланування стала реалізація взаємодії з базами даних, а точніше уся її реалізація. Для досягнення мети даного етапу розробки було розгорнуто базу даних MariaDB на персональному сервері, та підключено фреймворк Entity Framework для взаємодії з новоствореною базою даних [54]. У процесі розробки проєкту було викрито критичну проблему роботи з базами даних через фреймворк Entity Framework, а саме її неможливість працювати з базами даних типу MariaDB версії 14. Точну причину проблеми встановити не вийшло, але було знайдено ще одну бібліотеку, яка підтримує роботу з базами даних такого типу - Pomelo.EntityFrameworkCore.MySq. Данна бібліотека стала ще однією критичною залежністю проєкту, яка суттєво збільшила список баз даних, з якими додаток може працювати [55].

Було створено окремий клас для взаємодії з базами даних, який реалізував усі кінцеві точки API у їх базовому вигляді, дозволяючи отримувати інформацію з бази даних, яка оброблятиметься далі для відправки. Інформація з бази даних або повертається у тому вигляді, в якому вона була записана, або повертається значення null, що сигналізує про нестачу інформації. У випадку помилки повертається помилка, яка також розпізнається програмою, та відповідно формує відповідь для звернення через кінцеву точку API.

Щоб забезпечити масштабованість, усі компоненти проєкту були розраховані для розміщення та підтримки багатьох підключених пристроїв одночасно. Хоча до цього моменту проект розроблявся та тестувався лише з одним підключеним пристроєм, програмний код був написаний для підтримки одночасного використання кількох інверторів.

Це було досягнуто шляхом проведення ретельного аналізу сервісного поля у відгуку інвертора. Кожна відповідь інвертора містить на початку 16-байтний блок інформації, який відповідає за ідентифікацію пристрою та надання мінімальної інформації про стан відповіді. Ця інформація не є критичною для роботи проєкту в цілому, і у випадках, коли використовується лише один інвертор, вона повністю ігнорується.

Однак, аналізуючи цей 16-байтовий блок інформації, проект може ідентифікувати та розрізняти кілька підключених інверторів. Це дозволяє горизонтально масштабувати проект, забезпечуючи підтримку все більшої кількості пристроїв без шкоди для продуктивності чи надійності. Зразок завершеної масштабованої системи можна побачити на рисунку 2.3.

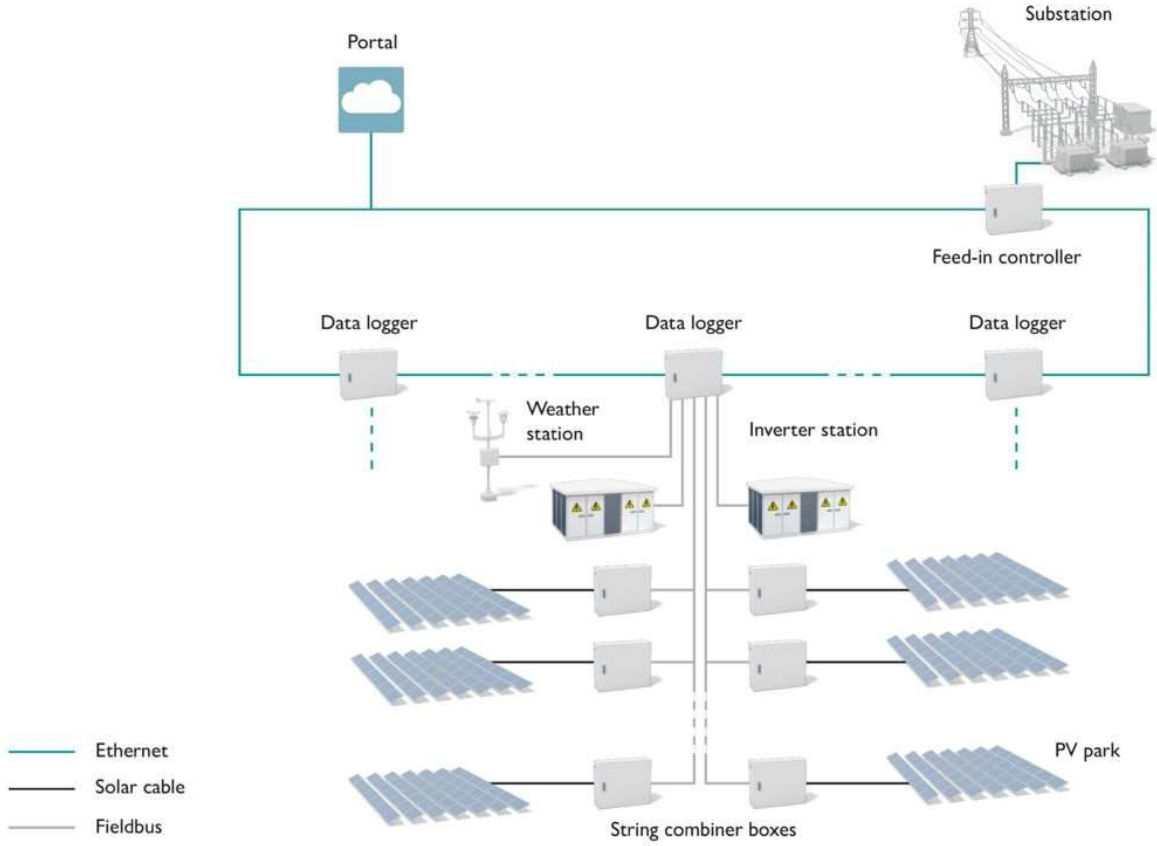


Рисунок 2.3 – Промислова масштабована електростанція для використання проєкту

Крім того, модульна конструкція проєкту дозволяє легко додавати нові інвертори до системи. Кожен інвертор розглядається як окремий компонент, а архітектура проєкту гарантує, що додавання або видалення інверторів не впливає на функціональність інших компонентів. Це робить проект дуже адаптивним до мінливих вимог і дає змогу підтримувати широкий спектр пристроїв і випадків використання.

На додаток до підтримки кількох інверторів, масштабована конструкція проєкту також дозволяє обробляти зростаючі обсяги даних і трафіку. Архітектура даного проєкту дозволяє розподіляти робоче навантаження між кількома підлеглими сервісами, гарантуючи, що кожна служба відповідає за конкретне завдання та може самостійно масштабуватися при потребі. Це дозволяє проєкту обробляти зростаючі обсяги даних і трафіку без шкоди для продуктивності та надійності [56].

Масштабування решти компонентів електростанції не матиме суттєвого ефекту на даному проєкті, через відсутність прямої залежності. Відмінності можливі на показниках з інверторів, з якими проєкт пов’язаний напряму, та це матиме відображення у веб-інтерфейсі. Решта факторів не матиме суттєвого впливу, та може бути опущена при плануванні топології, яка матиме залежність на даний проєкт.

Загалом, масштабований дизайн проєкту гарантує, що він може вмістити широкий спектр пристроїв і варіантів використання, що робить його дуже адаптивним і перспективним. Розробляючи всі компоненти з урахуванням масштабованості, проект може підтримувати все більшу кількість пристроїв і обробляти зростаючі обсяги даних і трафіку, що робить його надійним і надійним рішенням для широкого спектру програм.

2.4 Кіберфізичний пристрій для виведення поточних параметрів інвертору

Однією з головних задач проєкту – надання програмного інтерфейсу для взаємодії з іншими пристроями, шляхом надання інформації з інвертору за допомогою використання протоколу обміну інформацією HTTP. Тому було вирішено реалізувати пристрій у віртуальному середовищі, де можна з легкістю симулювати цілий спектр задач та умов. Для створення віртуального пристрою для отримання інформації з проєкту було обрано платформу Raspberry Pi Pico через її технічні характеристики - гнучкість та потужність.

Для реалізації логіки роботи одноплатного комп'ютеру для отримання інформації у віртуальному середовищі було обрано програмний стек та мову програмування MicroPython [57,58]. Це рішення було обумовлено швидкістю розгортання програмних продуктів на цільовій платформі, швидкістю розробки, та можливістю працювати на малопотужних пристроях. При тому, обраний стек технологій та інструментів для розробки пропонує можливості та потужності з стеку мови програмування Python, що дозволятиме пришвидшити розробку ще сильніше.

Для відображення інформації, отриманої з пристрою, було використано семисегментний індикатор на чотири позиції. Цей семисегментний індикатор дозволяє відображати числові значення, отримані з інвертора, де більшість даних є числовими. Використання семисегментного індикатора забезпечує чітке та зрозуміле відображення інформації, що важливо для оперативного моніторингу та аналізу.

Для швидкого і наочного відображення статусу роботи інвертора були застосовані кольорові світлодіоди – червоний, жовтий та зелений. Червоний світлодіод сигналізує про критичні помилки або збої, жовтий вказує на попереджувальні повідомлення або менш критичні проблеми, а зелений свідчить про нормальну роботу системи. Це дозволяє кінцевому користувачу швидко орієнтуватись у стані інвертора без необхідності додаткових роздумів або аналізу [59].

Таким чином, було створено макет пристрою для отримання та відображення інформації, який зображено на рисунку 2.4. У віртуальному середовищі, на даний момент, використовується пряме підключення компонентів. Цей метод підключення є ефективним для невеликої кількості індикаторів, проте має обмежені можливості для масштабування. Це означає, що при збільшенні кількості індикаторів або при необхідності розширення функціональності можуть виникнути труднощі з управлінням та синхронізацією компонентів.

Додатково, макет пристрою був оптимізований для зручності використання та подальшого розвитку. У майбутньому планується вдосконалити метод підключення для підтримки більшої кількості індикаторів та розширених функцій. Це може включати використання мультиплексування або інших технологій для ефективного керування великою кількістю індикаторів та забезпечення надійної роботи системи.

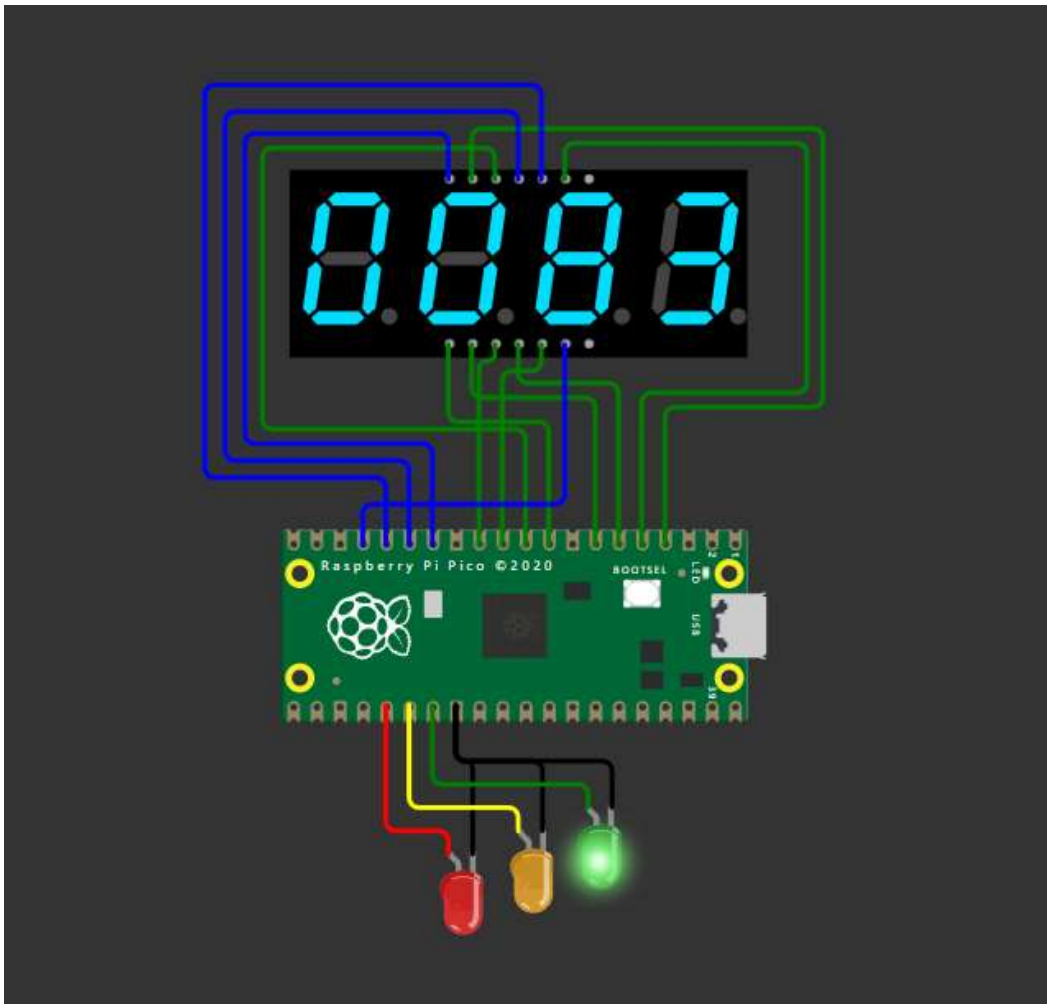


Рисунок 2.4 – Пристрій для роботи з отриманою інформацією

Складений пристрій являвся лише прототипом для реалізації взаємодії, та доказом можливості роботи проєкту через мережу, а саме передачі інформації через мережу інтернет. Прототип містить у собі семи сегментний індикатор на 4 позиції для відображення значень, та 3 світлодіоди для відображення наглядного та

простого для сприйняття стану пристрою. У даному випадку було обрано значення підключеної батареї до пристрою, де було виведено відсоткове значення заряду батареї, який був 83 відсотки у момент запису.

Три світлодіоди відображали статус батареї, та запалювалися в залежності від значення, яке було виведено на чотирьох-позиційний семи сегментний індикатор, де значення від 0 до 10 відсотків запалюватимуть червоний світлодіод, значення від 10 до 30 запалюватимуть жовтий світлодіод, та решта значень від 30 відсотків до 100 відсотків запалюватимуть зелений світлодіод. При цьому може бути запалений лише один світлодіод одночасно, в залежності від переданого значення у логіку роботи запалювання світлодіодів. Через це ефективні межі значень є наступні:

- червоний світлодіод – від 0 до 9;
- жовтий світлодіод – від 10 до 29;
- зелений світлодіод – від 30 до 100.

Подальший етап передбачав реалізацію компонування пристрою, що полегшує додавання індикаторів для відображення інформації. Щоб досягти цього, традиційно використовують протокол зв’язку низького рівня між компонентами I2C. Цей підхід забезпечує масштабованість для задоволення промислових вимог [60,61].

Однак це рішення також має певні недоліки. Наприклад, складність і розмір програмного коду значно збільшуються, що може створити проблему в майбутньому. Крім того, використання цього протоколу зв’язку призводить до збільшення енергоспоживання.

Щоб запобігти наслідки невизначеного стан сигналу, необхідно використовувати резистори. Ці резистори зводять невизначений стан до логічної одиниці. Однак це не є серйозною проблемою під час використання Raspberry Pi Pico, оскільки ця функція вбудована в нього з самого початку.

Варто відзначити, що протокол I2C є широко використовуваним стандартом для зв'язку між інтегральними схемами. Це двопровідна послідовна шина, яка

дозволяє декільком пристроям спілкуватися один з одним, використовуючи лише два дроти - один для даних (SDA) і один для годинника (SCL). Це спрощує підключення та зменшує кількість контактів, необхідних для зв’язку.

Крім того, протокол I2C підтримує зв'язок з декількома головними пристроями, що дозволяє декільком пристроям діяти як головні та ініціювати зв'язок з іншими пристроями на шині. Ця функція особливо корисна в складних системах, де кілька пристроїв повинні спілкуватися один з одним.

Таким чином, реалізація схеми пристрою за допомогою протоколу зв’язку I2C забезпечує масштабованість і спрощує підключення. Однак він також має певні недоліки, такі як підвищена складність коду та енергоспоживання. Використання резисторів необхідне для пом’якшення невизначеного стану сигналу, але це не є серйозною проблемою при використанні Raspberry Pi [62] Pico. Протокол I2C є широко використовуваним стандартом, який підтримує зв’язок із кількома майстрами, що робить його придатним вибором для складних систем.

Протокол I2C працює в конфігурації головний-підлеглий, де один пристрій діє як головний і ініціює зв’язок з одним або кількома підлеглими пристроями. Головний пристрій генерує тактовий сигнал і керує зв'язком на шині. Кожен підлеглий пристрій має унікальну адресу, яку провідний пристрій використовує для зв’язку з ним.

Протокол I2C підтримує різні режими передачі даних, такі як стандартний режим, швидкий режим і високошвидкісний режим. Стандартний режим має максимальну швидкість передачі даних 100 Кбіт/с, тоді як швидкий режим підтримує до 400 Кбіт/с. Високошвидкісний режим підтримує швидкість передачі даних до 3,4 Мбіт/с.

Протокол I2C також підтримує різні функції, такі як розтягування тактової частоти, арбітраж і зв’язок із кількома майстрами. Розтягування тактової частоти дозволяє підпорядкованому пристрою утримувати лінію синхронізації на низькому рівні, щоб уповільнити зв’язок, якщо він не готовий приймати або передавати дані. Арбітраж дозволяє кільком майстрам спілкуватися в шині, не конфліктуючи один

з одним [63]. Зв'язок із кількома головними пристроями надає змогу декільком пристроям діяти як головні та ініціювати зв'язок з іншими пристроями на шині, тим самим підвищуючи стійкість до помилок усієї системи.

Оновлений макет пристрою можна побачити на рисунку 2.5, де було додано динамік для звукового сповіщення при отриманні сигналу про критичну помилку з інвертору, та серво привід, за допомогою якого можна виконувати прості механічні дії. Також дисплей було замінено на LCD-дисплей, на якому значно простіше виводити спеціальні символи, та виводити інформацію загалом.

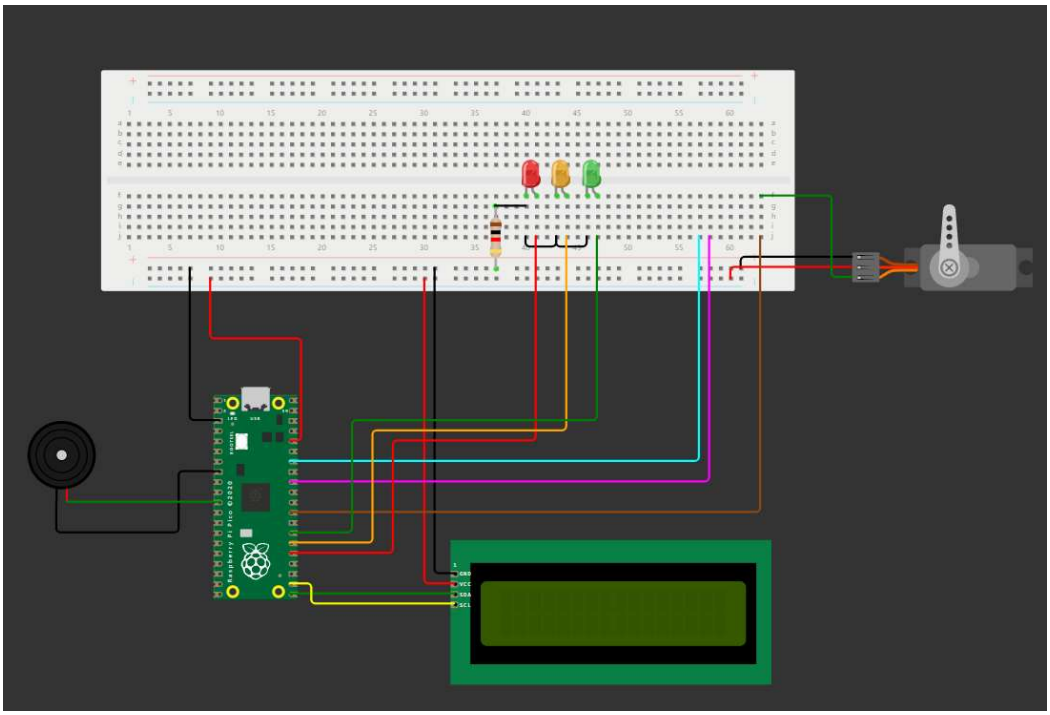


Рисунок 2.5 – Пристрій для роботи з отриманою інформацією, з використанням протоколу I2C

У даному оновленому зразку віртуального пристрою для роботи з отриманою інформацією було значно розширено функціонал шляхом збільшення кількості компонентів, що вдалося досягти завдяки зменшенню використаних точок підключення. Це дозволило зробити пристрій набагато гнучкішим і функціональнішим.

Для забезпечення можливості розширення функціоналу та виконання більшої кількості дій, а також для відображення більшої кількості інформації, можна додавати необхідні компоненти, які підтримують роботу через протокол I2C. Цей протокол є надзвичайно зручними для підключення великої кількості пристроїв, оскільки дозволяє використовувати лише дві лінії для передачі даних, що значно спрощує монтаж і підключення.

Пристрої, які не підтримують протокол I2C також можуть бути під'єднані до додадкової частини пристрою, яка конвертуватиме сигнал, отриманий по протоколу I2C, у підтримуваний сигнал для таких пристроїв. Це дозволяє використовувати широкий спектр компонентів без значних змін у загальній архітектурі системи.

Програмний код до пристрою може потребувати мінімальних змін при підключенні нових пристроїв, які підтримують протокол I2C. Це обумовлено стандартністю протоколу і наявністю великої кількості готових бібліотек та рішень для роботи з ним. Однак, підключення пристроїв, які не підтримують I2C, вимагатиме більше змін у програмному коді, оскільки для них необхідно буде окремо реалізовувати функціонал для взаємодії.

Ці зміни значно покращують гнучкість та функціональність віртуального пристрою, роблячи його більш придатним для використання в різних умовах та з різними типами обладнання. Такий підхід дозволяє не лише зменшити складність апаратної частини, але й значно спростити процес модернізації та розширення функціоналу пристрою в майбутньому.

Крім того, використання протоколу I2C забезпечує високу швидкість передачі даних та надійність, що є критично важливим для багатьох застосувань. Завдяки цьому система може ефективно працювати навіть у складних умовах, забезпечуючи точну та своєчасну обробку і відображення отримуємих даних з віддаленої серверної частини проєкту.

Отже, оновлений макет віртуального пристрою демонструє значний прогрес у порівнянні з попередньою версією макету пристрою, пропонуючи більш широкий

функціонал, гнучкість і надійність, що робить його ще більш придатним для використання в даному проєкті та різноманітних умовах використання даного віртуального макету пристрою.

2.5 Висновки

У цьому розділі детально розглянуто технологічний стек, на якому ґрунтувався проект у своїй завершеній формі. Починаючи з початкової реалізації, з усіма її недоліками, його поступовий розвиток веде до формування повноцінного та фінального технологічного стеку.

Одним із ключових обґрунтованих виборів було використання мови програмування C# для втілення концепції проєкту, а також допоміжних мов програмування для реалізації підтримуючих компонентів. Мови, такі як JavaScript, Python, а також SQL, використовувалися у розробці веб-інтерфейсу, логіки бізнес-логіки та взаємодії з базами даних відповідно.

Паралельно з цим, був створений пристрій для взаємодії з програмним комплексом проєкту у віртуальному середовищі. Використання віртуального середовища було обумовлено двома ключовими чинниками: неможливістю завдати шкоди обладнанню та гнучкістю реалізації пристрою.

Важливо відзначити, що випробування пристроїв у віртуальному середовищі здійснюється швидше та з більшою зручністю, порівняно з тестуванням у реальних умовах.

Крім того, розгляд можливостей розширення функціоналу та оптимізації використання інших мов програмування та технологій може відкрити нові перспективи для проєкту. Наприклад, використання мікросервісної архітектури може полегшити розвиток та супроводження системи, забезпечуючи більшу гнучкість та масштабованість.

3 ДЕМОНСТРАЦІЯ РОБОТИ ПРОЄКТУ

3.1 Тестування кіберфізичного засобу із використанням API

Після того, як було втілено одно з найголовніших складових проєкту – система кінцевих точок програмного інтерфейсу, виникла потреба у перевірці та відлагоджені системи. Через те, що система кінцевих точок програмного інтерфейсу побудована з використанням протоколу обміну інформації HTTP, також виникла потреба у програмному комплексі, яке зможе надати можливість взаємодіяти з будь-якими кінцевими точками програмного інтерфейсу, які використовують протокол обміну інформації HTTP.

Тому було використано наступний програмний комплекс для перевірки кінцевих точок, з використанням протоколу HTTP – Postman. Postman — це популярний інструмент тестування API, який пропонує зручний інтерфейс, автоматичне тестування, програми для збирання даних, імітаційні сервери та інтеграцію з конвеєрами CI/CD. Однак його функції звітування обмежені, а безкоштовна версія має деякі обмеження, наприклад обмеження на кількість запитів API, які можна зробити.

Програмний комплекс «Postman» є багатофункціональним, та через велику кількість інструментів програмний комплекс дозволяє налаштувати цілі сценарії запитів з безліччю додаткових параметрів. Це й стало вирішальним фактором у перевірці програмно-апаратної складової проєкту, адже для задач, які необхідно вирішити у даному проєкті потребується велика кількість запитів, де кожен запит матиме унікальні параметри.

На рисунку 3.1 можна побачити інтерфейс обраного програмного комплексу, який було обрано для перевірки програмної та апаратної складової проєкту. Завдяки інтеграції з різноманітними сервісами та платформами, Postman надає широкі можливості для автоматизації та тестування API, що дозволяє ефективно перевіряти інформацію та роботу всіх компонентів системи.

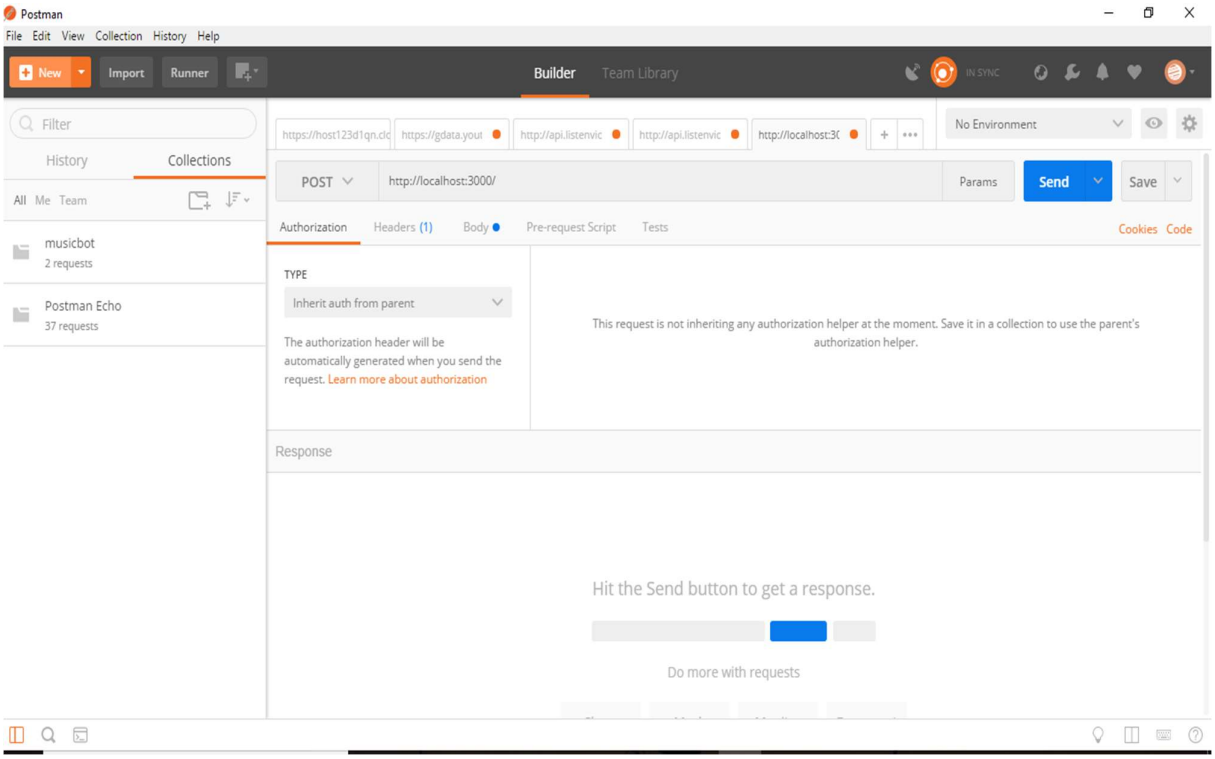


Рисунок 3.1 – Програмний комплекс «Postman»

Таким чином, було розпочато перевірку кінцевих точок проєкту за допомогою обраного програмного комплексу. Перевірку було проведено лише для кінцевих точок програмного інтерфейсу, які працюють виключно по методу «POST», адже кінцеві точки, які працюють з методом «GET» можуть бути перевірені у самому браузері, без необхідності у встановлені додаткового програмного забезпечення.

Кінцеві точки програмного інтерфейсу, що працюють виключно по методу «POST» у проєкті наступні: “/api/v1/cache/timestamp”, “/api/v1/cache/timestamp-range”, та “/api/v1/cache/all”. Усі ці кінцеві точки використовують метод «POST», через необхідність передачі додаткових параметрів у проєкт для формування відповіді згідно запиту. Процес перевірки кінцевих точок проєкту можна побачити на рисунку 3.2, де зображено програмний комплекс «Postman» у взаємодії з проєктом.

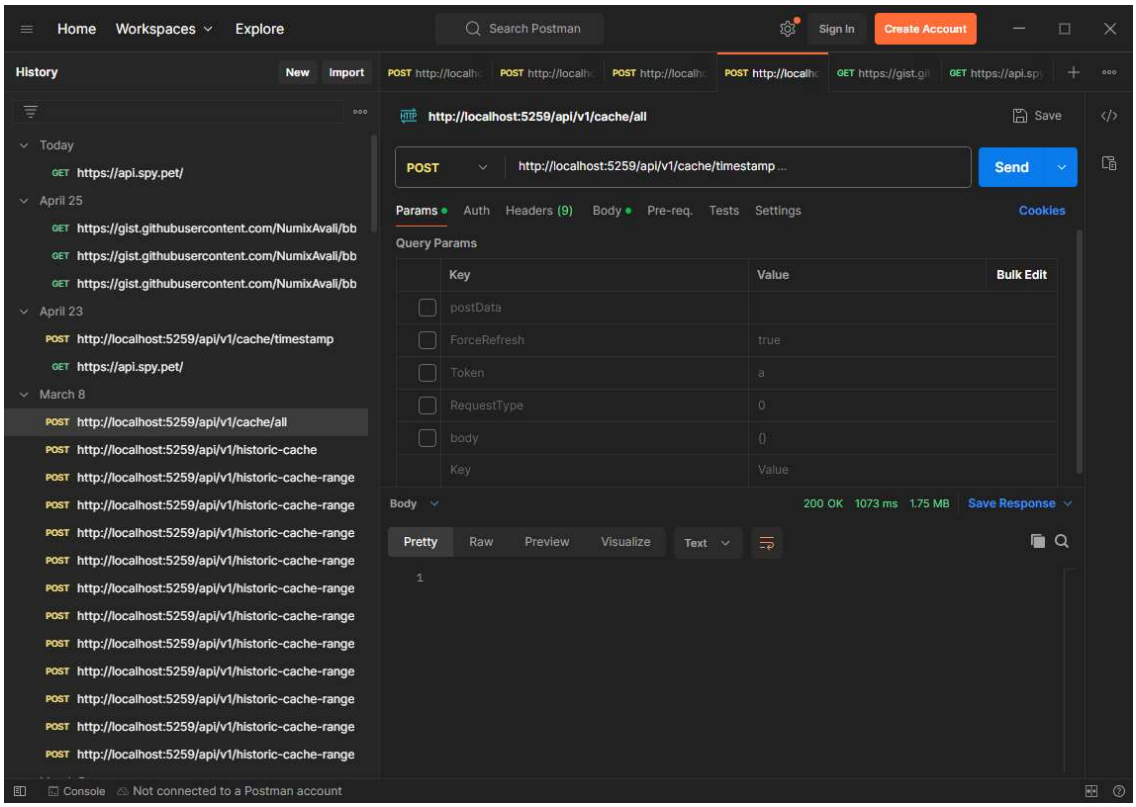


Рисунок 3.2 – Програмний комплекс «Postman» у дії

Далі було розпочато ретельну перевірку усіх даних, які були надіслані, та отримані у пристрою, який було реалізовано у віртуальному середовищі. Увесь процес розпочався з перевірки початкового запиту, та було ретельно перевірено усі поля самого запиту на присутність значень, адже поля з відсутніми значеннями можуть завдати великої шкоди самому пристрою.

Перевірку на правильність значень, які надходять з програмної частини проєкту було виконано через надсилання запиту на одну з кінцевих точок програмного інтерфейсу. Для цього було виконано запит типу «GET» на кінцеву точку програмного інтерфейсу “/api/v1/cache/latest”, де було отримано відповідь у вигляді JSON-файлу у його текстовому вигляді. Данна відповідь є непридатною для читання людиною, адже вона була позбавлена усіх перенесень на нову строку, тим самим зменшуючи кількість інформації для передачі. Це було виправлено за допомого використання онлайн-сервісу для форматування JSON-файлів, повертаючи назад усі переноси строк, та навіть дозволяючи відобразити вміст

файлу у вигляді дерева. Описаний результат переформатування відповіді отриманої з проєкту можна побачити на рисунку 3.3.

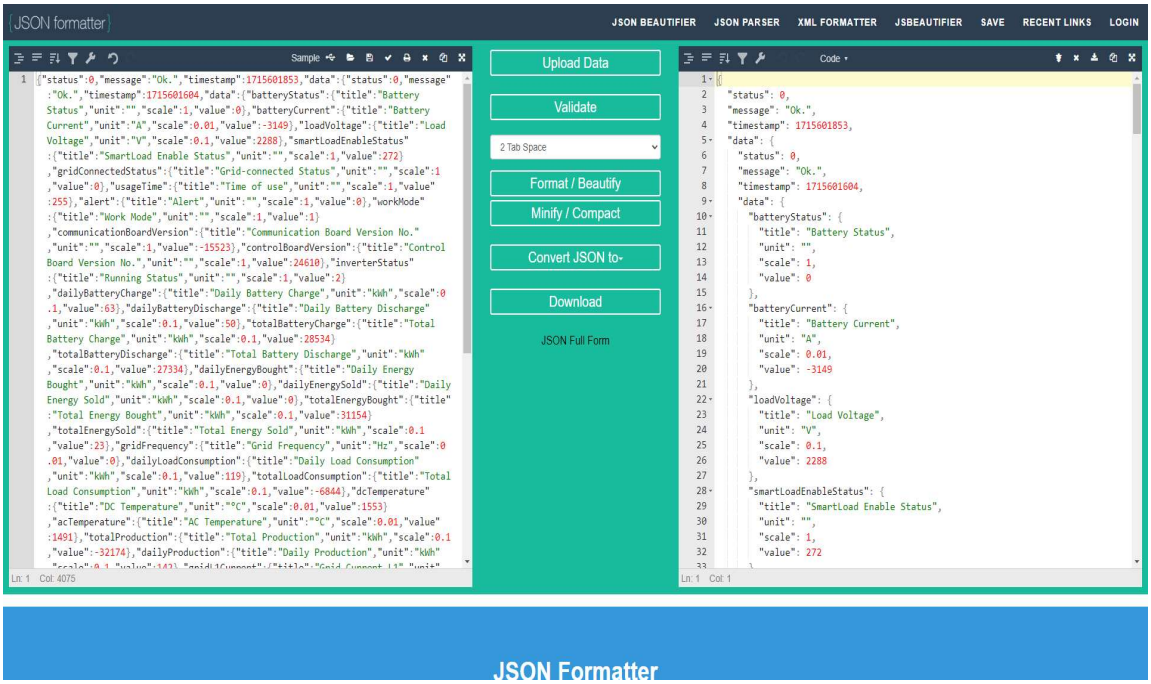


Рисунок 3.3 – Переформатована відповідь з кінцевої точки проєкту

Далі було проведено перевірку отриманої відповіді, використовуючи вбудовану документацію по можливостям проєкту, яку було створено під час реалізації алгоритму побудови відповіді, тим самим підтримуючи її в актуальному стані. Вбудована документація оновлюється разом з оновленням компонентів проєкту, тим самим завжди забезпечуючи її актуальний статус.

Та після проведення детального аналізу отриманої відповіді, було встановлено, що відповідь є придатною для подальшого використання у перевірці комунікації з підлеглою апаратною частиною проєкту. Цей зразок відповіді було збережено для подальшого використання у якості запиту для підлеглого віртуального пристрою.

Наступним кроком для перевірки підлеглого пристрою було налаштовано раніше розглянутий програмний комплекс «Postman» для роботи з IP адресою підлеглого віртуального пристрою для подальшої перевірки. Після цього було

надіслано HTTP-запит на підлеглий віртуальний пристрій, та було встановлено, що пристрій отримав відповідь, ті відобразив її у вікні для емуляції серійної консолі.

Після отримання впевненості що до роботи підлеглого віртуального пристрою, було виконано розбір отриманої відповіді назад у програмний об’єкт, з використанням якого можна буде з легкістю отримати необхідні значення, які надійшли у HTTP-запиті. Та після використання динамічного алгоритму для розбору JSON-об’єкту, який надійшов у якості запиту було отримано програмний об’єкт, який також було перевірено на відповідність, використовуючи попереднє збережену відповідь, як зразок.

Після проведення аналізу, та виправлення алгоритму для більш коректного розбору JSON-об’єкту, було задіяно механізм для надсилання HTTP-запитів, тим самим наближуючи віртуальний пристрій до завершеного стану. Одразу після цього було запущено віртуальний пристрій, який надіслав запит на програмний комплекс проєкту, та отримав правильну відповідь. Відповідь була розібрана на потрібну інформацію, та пристрій відобразив отриману інформацію.

Таким чином була проведена перевірка кінцевих точок програмного інтерфейсу проєкту за допомогою програмного комплексу «Postman». Під час перевірки було ретельно проаналізовано відповідь кожної точки на вхідні та вихідні дані, які були надіслані використовуючи протокол HTTP, а також було встановлено, що вони виконують свої функції згідно з встановленими вимогами.

Після завершення перевірки було підтверджено, що кінцеві точки програмного інтерфейсу працюють правильно, обробляючи дані згідно з призначенням та відповідаючи встановленим стандартам безпеки та ефективності. Також було ретельно перевірено взаємодію між компонентами системи, включаючи їхню відповідність специфікаціям і здатність до справжнього часу. Разом з цим також було підтверджено, що підлеглий віртуальний пристрій працює правильно, та виконує запрограмовані дії, в залежності від отриманої інформації. Додатково, враховуючи результати перевірки, були запропоновані можливі шляхи покращення функціональності та надійності системи.

3.2 Демонстрація графіків роботи фотоелектричної системи

При реалізації сторінки «History» у веб-інтерфейсі виникла потреба у візуалізації інформації для надання можливості кінцевому користувачу переглядати стан інвертора за будь-який проміжок часу.

Таким чином було прийнято рішення візуалізувати інформацію у вигляді графіків, де інформація надходить з програмного інтерфейсу самого програмно-апаратного комплексу.

Було обрано бібліотеку “Charts.js” для побудови графіків, щоб уникнути потребу у реалізації механізму побудови та відображення інформації своїми силами. “Charts.js” — це проста та зручна у використанні бібліотека, яка є відмінним варіантом для створення базових діаграм.

Згідно з опитуванням State of JS, це найпопулярніша бібліотека, що робить її гарним вибором, якщо необхідна бібліотека, яка широко використовується та добре підтримується

Далі ця бібліотека стала частиною проєкту у директорії, у якій знаходяться решта бібліотек та компонентів, з використанням яких було побудовано веб-інтерфейс, та підключено у заголовку веб-інтерфейсу. Далі на сторінці було оголошено порожні полотна, на яких буде розташовано та відображено графіки.

Після цього було створено файл charting.js, у якому було розпочато реалізацію логіки, за допомогою якої буде відображено графіки з інформацією, отриманою з проєкту.

Побудування починається з отримання трьох елементів із HTML-документа за допомогою “document.getElementById()” та отримання їх двовимірного контексту для малювання на них діаграм. Елементи мають такі ідентифікатори, як “historicChartDaily”, “historicChartDailyTotals” та “historicChartTotal”.

Далі визначається функція “timeConverter”. Данна функція приймає мітку часу UNIX (у секундах) як аргумент і перетворює її на зрозумілий людині рядок дати й часу. Функція також приймає два необов’язкові булеві параметри, “useYear”

і “useMonth”, які визначають, чи повинні рік і місяць бути включені у вихідний рядок відповідно.

Потім код обирає всі елементи за допомогою класу “btn-check” за допомогою “document.querySelectorAll()” і зберігає їх у змінній “radioButtons”. Прослуховувач подій змін додається до кожної кнопки. Коли кнопка натискається користувачем, функція оновлює змінну “timePeriodStr” текстовим вмістом наступного однорідного елемента кнопки та викликає функцію “updateCharts()”.

Далі оголошена функція “strTimeToInt()”. Ця функція приймає рядок, що представляє період часу (наприклад, «день», «тиждень», «місяць», «рік» або «за весь час») і повертає кількість секунд у цьому періоді часу. Ця функція використовується пізніше в коді для визначення діапазону часу для отримання даних.

Функція “getData()” оголошена як асинхронна функція. Вона ініціалізує порожній об’єкт даних за допомогою функції “initializeData()”, а потім надсилає HTTP запит за допомогою бібліотеки AJAX для отримання даних із програмного інтерфейсу проєкту. Потім отримані дані обробляються за допомогою функції “populateData()”, і повертається оновлений об’єкт даних.

Функція “initializeData()” створює об’єкт із різними властивостями, що представляють різні категорії даних. Кожна категорія ініціалізується за допомогою функції “initializeCategory()”, яка встановлює одиницю вимірювання категорії, мітки, значення та кількість цільових фрагментів.

Функція “sendRequest()” надсилає запит AJAX до програмного інтерфейсу проєкту для отримання даних. Запит надсилається за URL-адресою “${baseUrl}/api/v1/cache/timestamp-range” за допомогою методу POST. Дані запиту містять мітки часу початку та кінця для даних, які потрібно отримати, а також маркер для автентифікації.

Змінна “baseUrl” є переданою змінною з частини проєкту, яка реалізована на мові програмування Blazor, та визначає фрагмент адреси до сторінки проєкту. Значення цієї змінної вираховується в залежності від налаштувань проєкту,

хостингу, де розміщено сам проєкт, та в залежності від домену, на якому розташовано проєкт. Цю змінно неможливо обрахувати у середовищі, де доступна лише мова програмування JavaScript.

Функція “populateData()” обробляє дані, які повертає програмний інтерфейс. Загальна суть цієї функції полягає у приведені отриманої інформації у вид, з яким зможе працювати решта функцій для побудови та відображення діаграм. Далі оголошена функція “updateCharts()”. Вона викликається кожного разу, коли користувач вибирає новий період часу за допомогою перемикачів. Ця функція отримує дані за вибраний період часу за допомогою функції “getData()”, а потім оновлює діаграми новими даними.

В самому кінці роботи скрипту викликається функцію “updateCharts()”, щоб ініціалізувати діаграми даними періоду часу за замовчуванням. Та таким чином запустити усю подальшу логіку, яка дозволяє користувачу взаємодіяти з діаграмами. На зображенні представлено систему блок-схем реалізації логіки, яка складається з трьох основних компонентів: "Інтерфейс користувача", "Обробка інформації" та "Оновлення діаграм". Кожен з цих блоків відповідає за конкретний аспект в процесі розрахунку та відображення графіку, як зображено на рисунку 3.4.

"Інтерфейс користувача" відповідає за взаємодію з користувачем, забезпечуючи зручний та інтуїтивно зрозумілий інтерфейс для введення даних та отримання результатів. Цей блок включає в себе різноманітні елементи інтерфейсу, такі як кнопки, поля для введення тексту, вікна відображення результатів і т. д. "Обробка інформації" виконує обчислення та аналіз введених даних, враховуючи необхідні параметри та умови. Цей блок забезпечує точність та ефективність обчислень, використовуючи різноманітні алгоритми та методи для опрацювання вхідних даних і генерації висновків. "Оновлення діаграм" відповідає за оновлення графічного відображення результатів, забезпечуючи користувачу актуальну інформацію у зручному для сприйняття вигляді. Цей блок включає в себе механізми автоматичного оновлення графіків, діаграм і інших візуальних елементів, що відображають результати обробки даних.

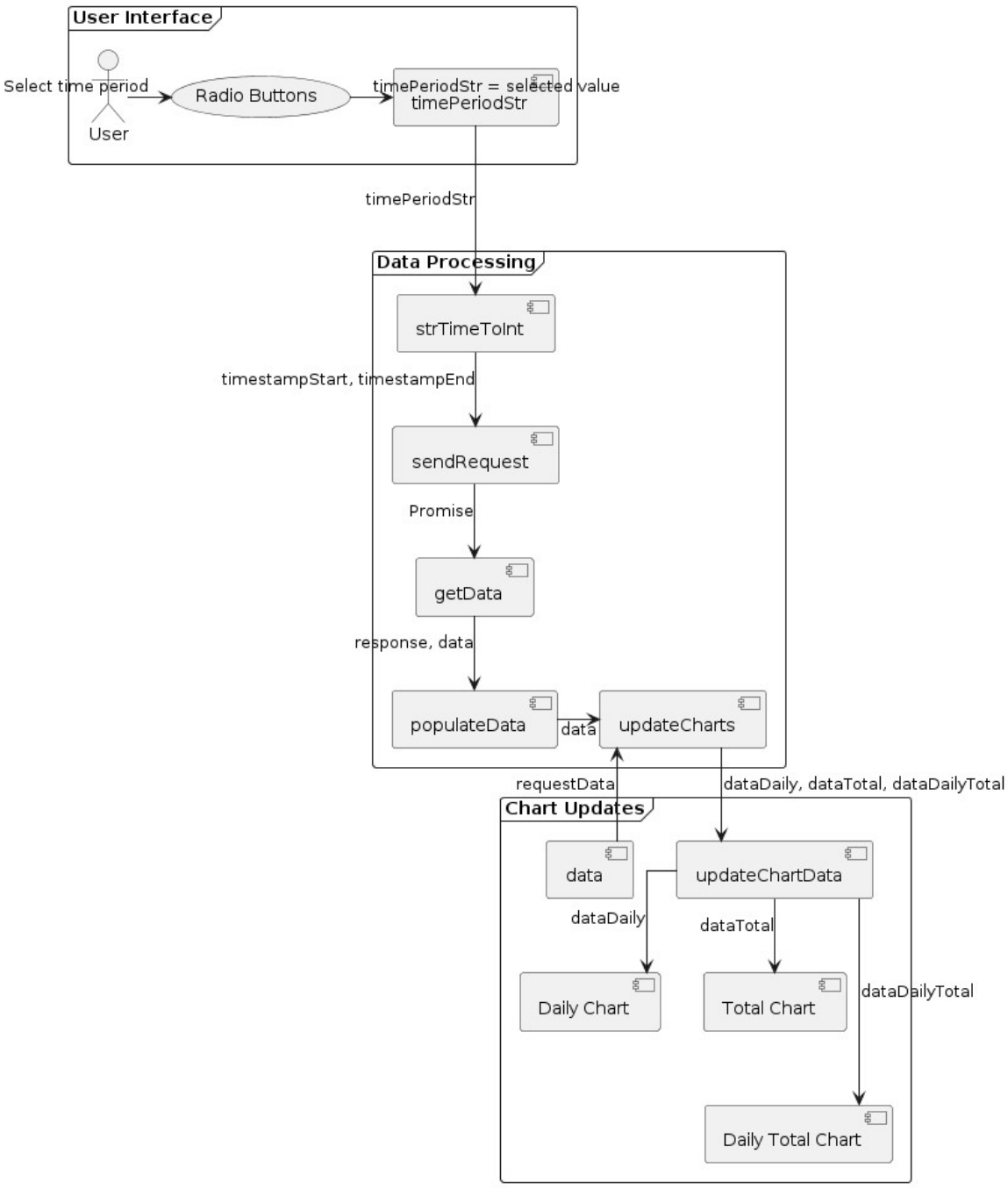


Рисунок 3.4 – UML-діаграма логіки побудування графіків

З особливостей роботи алгоритму побудови графіків варто відмітити факт того, що через велику кількість отриманої інформації з кінцевої точки програмного інтерфейсу, алгоритм було спеціально модифіковано для роботи з великою кількістю даних. Однією з найбільших проблем при отриманні інформації є її кількість, адже кінцева точка програмного інтерфейсу повертає усі показники, які

були вказані у відправлених точках часу при запиті інформації. По замовчуванню проєкт у автоматичному режимі збирає інформацію кожні 10 хвилин, та таким чином за один день проєкт збирає близько 144 знімків стану інвертора. Але при виборі проміжку часу у тиждень, місяць, або більше – кількість знімків стану збільшується до неймовірно великих значень, де за тиждень буде зібрано та відображено 1008 знімків, та приблизно 4464 за місяць.

Жодна з раніше розглянутих бібліотек не здатна працювати з таким об’ємом інформації напряму без сповільнень, та у всіх розглянутих бібліотеках були помічені проблеми з швидкістю роботи. У деяких бібліотеках це спостерігалось зависанням браузера, у деяких просто сповільненням завантаження сторінки. Але жодна з бібліотек не надавала потрібної швидкодії, та через це було прийнято рішення що до реалізації агрегації інформації перед передачею інформації для відображення у діаграмах.

Агрегація інформації відбувається у секції «Data Processing», а саме у функції «populateData», де отримана інформація з кінцевої точки програмного інтерфейсу розбивається на підкатегорії, та перерозподілюється у змінну для відібраних значень. Відібрані значення групуються у секції таким чином, щоб загальна кількість секцій не перебільшувала 200 секцій. Наступним кроком по усім 200 секціям виконується операція обчислення середнього загального значення, тим самим обмежуючи загальну кількість знімків стану інвертора у 200 значень. Після проведення процесу агрегації значень, та тим самим обмежуючи її кількість, усі розглянуті бібліотеки витримували навантаження.

Таким чином на сторінці «History» виводяться графіки з інформації, яку проєкт отримує з інвертору на постійній основі, які у свою чергу зчитуються з бази даних, після чого усі значення проходять процес агрегації для спрощення роботи обраної бібліотеки для побудови діаграм, і одразу після цього з них будується діаграми. На рисунку 3.5 можна побачити результат роботи алгоритму роботи з інформації для побудови першої діаграми, на якій виведено інформацію використання батареї, генерацію електроенергії, та споживання.

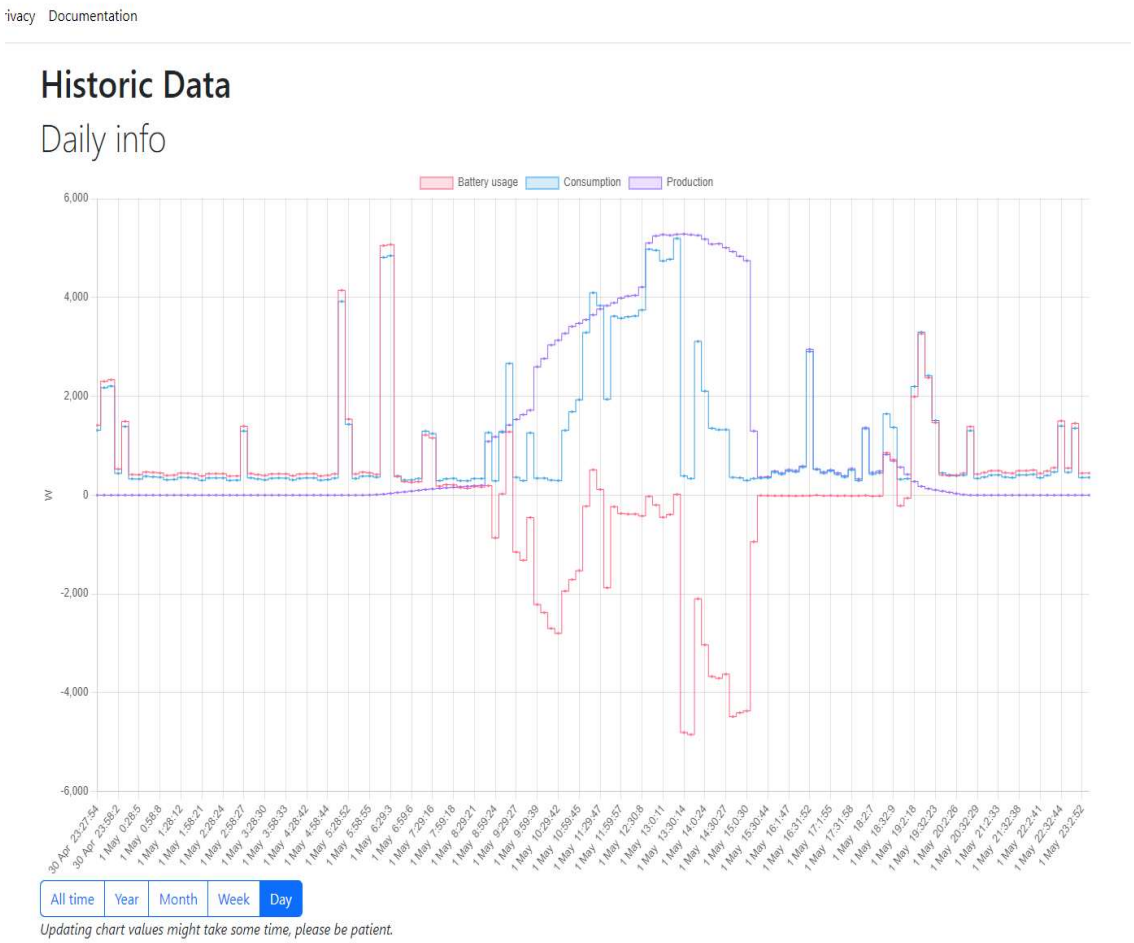


Рисунок 3.5 – Перша діаграма, яка відображає значення генерації, споживання та використання батареї

На другій діаграмі, яку також зображено на сторінці «History» виводиться схожа інформація до попередньої діаграми, але при тому вона виводиться у іншому вигляді. Інформація на другій діаграмі виводиться на діаграмі, яка складається зі стовпців, сумарне значення яких відповідає за загальне значення отриманої енергії у систему, та використаної енергії з системи. Другу діаграму у дії можна побачити на рисунку 3.6, з проміжком часу значень за день, на якій також можна побачити, як щоденні показники скидаються кожні 24 години до нуля, тим самим формуючи хвильо-подібну діаграму з відображеної інформації.

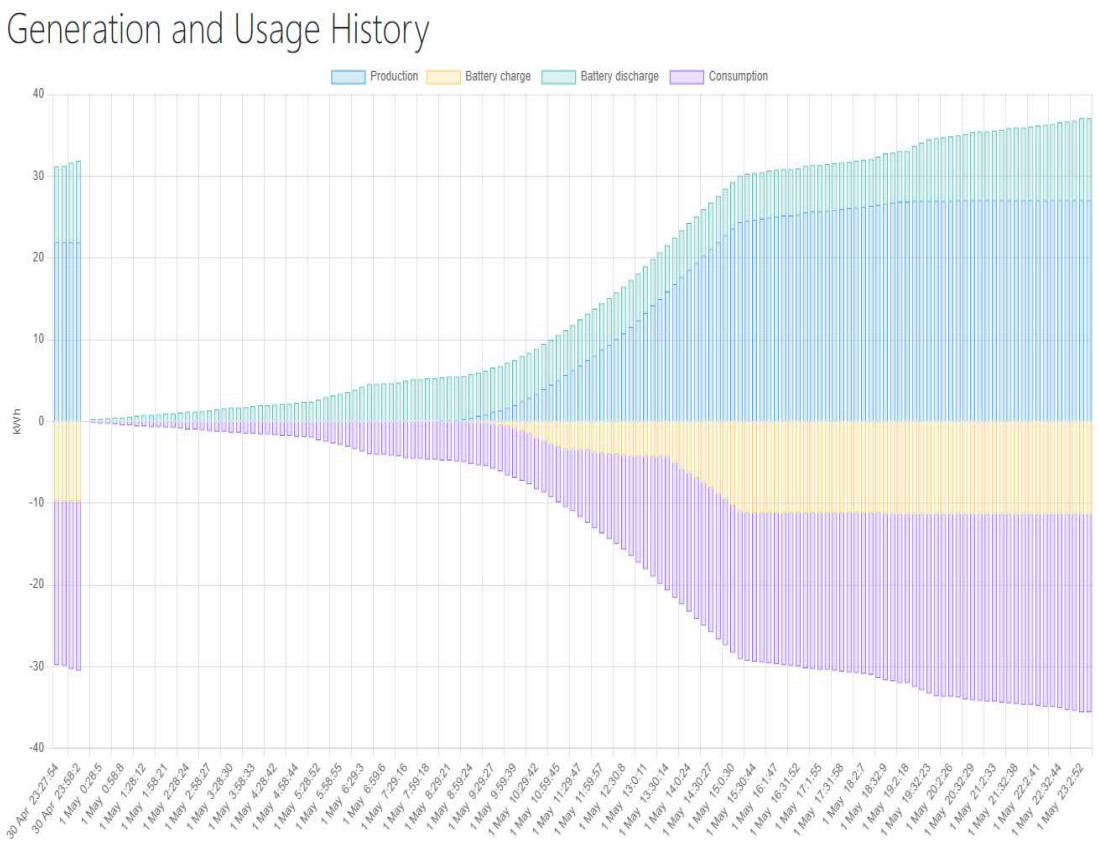


Рисунок 3.6 – Друга діаграма, яка відображає загальні значення генерації, споживання та використання за день

Третьою, та останньою діаграмою є графік загальних значень, який є статистикою використання інвертора за весь час роботи інвертора. Ця діаграма розташована у спеціальному підменю, яке відкривається та показує діаграму по натисканню на стрілку розгорнення меню.

Це було зроблено через відносно низьку інформативність діаграми, та через низьку потребу у відслідковуванні загальних значень у реальному часі. Значення з третьої діаграми є виключеннями з алгоритму розрахунку значень, та рахуються по найбільшому значенню отриманому з кінцевої точки. Саму діаграму для відображення загальних значень, у відкритому меню можна побачити на рисунку 3.7. Також на цій діаграмі було встановлено ліміт мінімальних значень, через те, що значення менші за 10% неможливо було побачити на самій діаграмі.

Рисунок 3.7 – Третя діаграма, яка відображає загальні значення генерації, купування енергії, продаж енергії, та використання батареї

Таким чином було сформовано динамічний алгоритм для отримання інформації, з використанням та агрегації якої відбувається побудова, та відображення інформаційних діаграм.

3.3 Передача та перетворення даних сонячних станцій

Проєкт містить у собі унікальну реалізацію обробки та передачі інформації, через яку проєкт отримав унікальну та гнучку топологію передачі інформації. Проєкт використовує одну з кращих практик побудови проєктів на мові

програмування C#, а саме використання «конвеєрів» для обробки інформації. Загальний зразок використання конвеєрної архітектури зображено на рисунку 3.8, де зображено архітектуру, яку використано та застосовано у хмарному середовищі «Azure» від корпорації Microsoft.

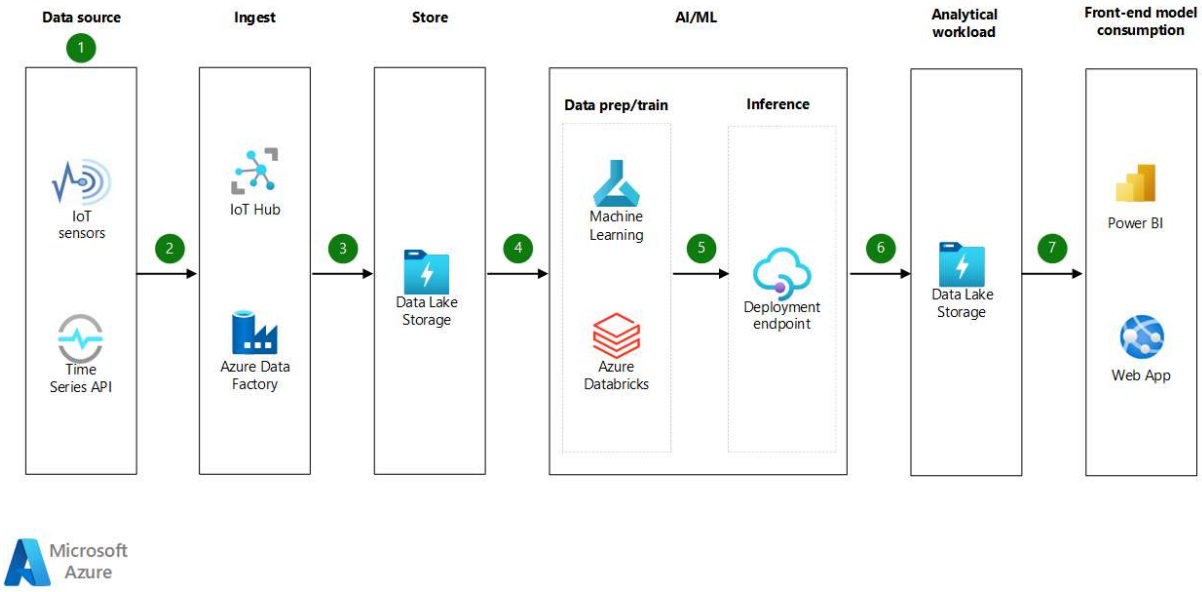


Рисунок 3.8 – Конвеєрна архітектура у хмарному середовищі Microsoft Azure Cloud

Отож, у даній конвеєрній архітектурі інформація починає свій шлях у класі «RequestHandler», який відповідає за усю мережеву складову проєкту, та запуск виконання послідуючих етапів конвеєрної архітектури. Цей клас викликається при отриманні дійсного запиту на одну з кінцевих точок програмного інтерфейсу, де додаткові параметри визначаються за допомогою вмісту самого запиту, або типом кінцевої точки програмного інтерфейсу.

Після створення об’єкту у пам’яті з певними переданими параметрами відбувається розгалуження логіки для визначення шляху обробки інформації, виходячи з якого виконується один з методів для отримання інформації. Це може бути отримання збереженої інформації, або отримання миттєве інформації, прямо з інвертору.

При проходженні конвеєру, у якому необхідно отримати миттєвий знімок стану інвертора, з усіма показниками, виконується метод, у якому запускається вбудований клієнт для виконання мережевих операцій з використанням мережевого протоколу «TCP». У мережевий клієнт передаються такі необхідні параметри, як IP-адреса та порт інвертору, для можливості ініціювати TCP-запит на інвертор.

Але перед виконанням самого TCP-запиту виконується метод для обчислення самого запиту. Адже порожній запит на інвертор завжди повертатиме помилку, яка не містить у собі інформації. Це відбувається шляхом проведення двійкових операцій з серійним номером інвертору, який у свою чергу зчитується з класу, який виконується для отримання налаштувань проєкту. Після проведення математичних операцій, та отримання двійкового значення коректного запиту, відбувається запуск TCP-клієнту, який повертає декілька отриманих відповідей у масив, який далі буде передано у клас для обробки інформації – «DataProcessor».

У класі для обробки інформації «DataProcessor» відбувається поетапний розбір двійкових пакетів інформації, де кожен з пакетів розбирається на значення шляхом переведення певних позицій двійкового пакету у цільночисельні значення, згідно регістрів. Відповідь з інвертора у її вихідному двійковому вигляді, який зображено у системі Base64 для покращеного сприйняття людиною можна побачити на рисунку 3.9. Регістри для визначення позицій, згідно значень яких відбувається розбір значень отримуються з окремих файлів конфігурації, які були створені для цілого ряду інверторів. Ці файли конфігурації можна вважати окремими модулями, за допомогою яких можна збільшувати кількість підтримуваних пристроїв.

Також використання системи відображення Base64 дозволяє перенесення інформації з однієї мови програмування у іншу мову програмування, через подібність до звичайного тексту. Це було використано для перевірки вхідних та вихідних даних з іншими проєктами, де були використані такі самі запити для інверторів.

```
Connected to the server.
Message #0 was sent to the inverter
Reply from the inverter, length: 252 bytes
pe8AEBUAJjKt3qMCAbCUbAE+GwAA7JbVZAED3DIzMDExNDYyNTgAAIEAAAAXJAAAYCLDXQACOIAAAQIDAAAA/wAKGAUNEgA5AAEfQAUUAAAAAAAAAAAnEAu4
H0AH0ANSAAAAAAPoAAAAAAAAAAEAAAABAAAAAAAAAAAAAAAAAAAjKAAAAAAAAAABAAEAAgCPAAAAAGvAAAALUQmBCYE7PwAAAGwAN2+jAABqywAAAAAAAHmy
AAAAAAAXAAAAtACP5V
wAAGovAAAAAAXEBa4AAAAAB9AAAIKTAAAz9wAACgEAAAAAAAAAAAAAAAACWADPCZUAFAgnABPsv2oVAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA==
Message #1 was sent to the inverter
Reply from the inverter, length: 230 bytes
pdkAEBUAJzKt3qMCAbCUbAE/GwAA7JbVZAEDxgA2AAAANgjoCPwAAAj8COgAAAAAAAAAAAAAAAADwAAAAAAAAAAAAAAAAAAAAAIcQAACHEIcQAACHEIcQAA
ACwEthN2AFcIcQHnAYcAAAAABTcKeROIE4gAAAEQAAAAAAAAAAAAARaAFoAVkAJYEZQAAABaAAAAAABkAIwAAAABAAEACAPeAAcAFAAPEfgUUBKOAPAAABMk
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA==
```

Рисунок 3.9 – Інформація отримана з інвертору

Після того, як було проведено розбір інформації на її цільночисельні значення, з файлів конфігурації інвертора також отримуються такі значення, як назва, одиниці виміру, та модифікатор значення. Модифікатор значення – це значення, на яке буде помножене значення, яке раніше було отримано з інвертору. Це є частиною отримання кінцевого об’єкту зі значеннями, адже інвертор надає високоточні значення без плаваючої коми, тим самим вимагаючи конвертацію цілих чисел у числа з плаваючою комою шляхом множення їх на модифікатор.

Після проведення додаткових математичних операцій над значеннями, які було отримано з інвертору, усі параметри записуються у вкладений об’єкт, який містить у собі такі поля, як назва, значення, одиниця виміру, та модифікатор значення. Усі вкладені об’єкти записуються у один загальних об’єкт, у якому усі значення повертаються з класу «DataProcessor». Далі, коли інформація повернулась у клас «RequestHandler», вона обгортається у об’єкт з інформаційними полями, такими як мітка часу, коли відповідь була створена, машинне повідомлення про відповідь, та повідомлення про відповідь у форматі, який зрозумілий людині. Та

після усіх операцій по обгорненню інформації, готова відповідь повертається на кінцеву точку, на яку поступив запит з самого початку. Діаграму подорожі інформації, яку було описано можна побачити на рисунку 3.10

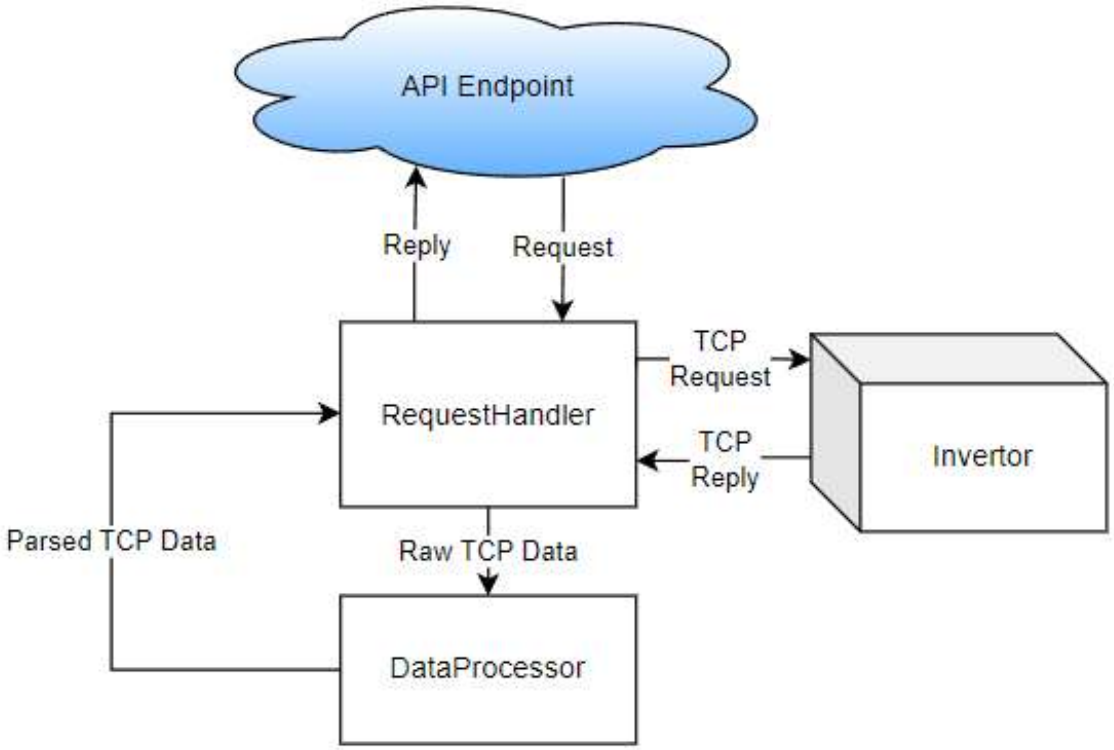


Рисунок 3.10 – Діаграма подорожі інформації по проєкту при запиті теперішньої інформації

Але на цьому логіка роботи розгалуження не завершена, та існує ще один, значно коротший конвеєр, який використовується при запиті історичної інформації. У цьому другому скороченому варіанті подорожі інформації не відбувається взаємодії з інвертором напряму, та тим самим значно спрощуючи логіку роботи. Замість взаємодії з інвертором відбувається взаємодія з класом для роботи з базами даних, який працює значно швидше через відсутність потреби у виконанні багаточисленних математичних операцій з потоком двійкової інформації.

У цій гілці розгалуження логіки роботи подорож інформації починається знову з кінцевої точки програмного інтерфейсу, з якого цього разу ще передаються додаткові параметри. Ці параметри можуть бути декількох видів – одна мітка часу, або дві мітки часу, що вказують на початок і кінець певного періоду. Окрім UNIX-міток часу також використовується поле для токену, який виконує роль ключа для подальшого використання програмного інтерфейсу. Ці параметри дозволяють здійснювати більш точний і деталізований запит до системи, враховуючи конкретний часовий ідентифікатор та інші умови.

Усі ці параметри передаються у клас “RequestHandler”, який якраз і запускає логіку розгалуження. У даному випадку була обрана гілка розгалуження для взаємодії з базами даних, де більшість інформації передається без змін у наступний клас – “DbHandler”. У класі для взаємодії з базами баних “DbHandler” передані параметри використовуються для визначення проміжку часу, за який необхідно отримувати інформацію.

Далі клас для взаємодії з базами даних отримує усю необхідну інформацію для підключення до бази даних з файлу налаштувань, використовуючи спеціально відведений клас для роботи з файлом налаштувань проєкту. Це дозволяє реалізувати плавну та ефективну передачу параметрів інтерфейсу користувача до бази даних та забезпечує надійність та стабільність роботи програми навіть у складних сценаріях взаємодії з даними.

Отримавши інформацію для підключення до бази даних, клас комбінує її з рештою параметрів отриманих раніше у один запит до бази даних, та надсилає запит до бази даних. Уся взаємодія з базами даних була реалізована з використанням фреймворку «Entity Framework», та його розширення у вигляді бібліотеки «Pomelo.EntityFrameworkCore.MySql». Також варто зазначити, що час отримання інформації з бази даних пропорційна кількості інформації, що запитується. Це є важливо у випадках, коли отримується інформація за великий проміжок часу, наприклад за місяць або рік. Діаграму для описаної топології подорожі інформації можна побачити на рисунку 3.11.

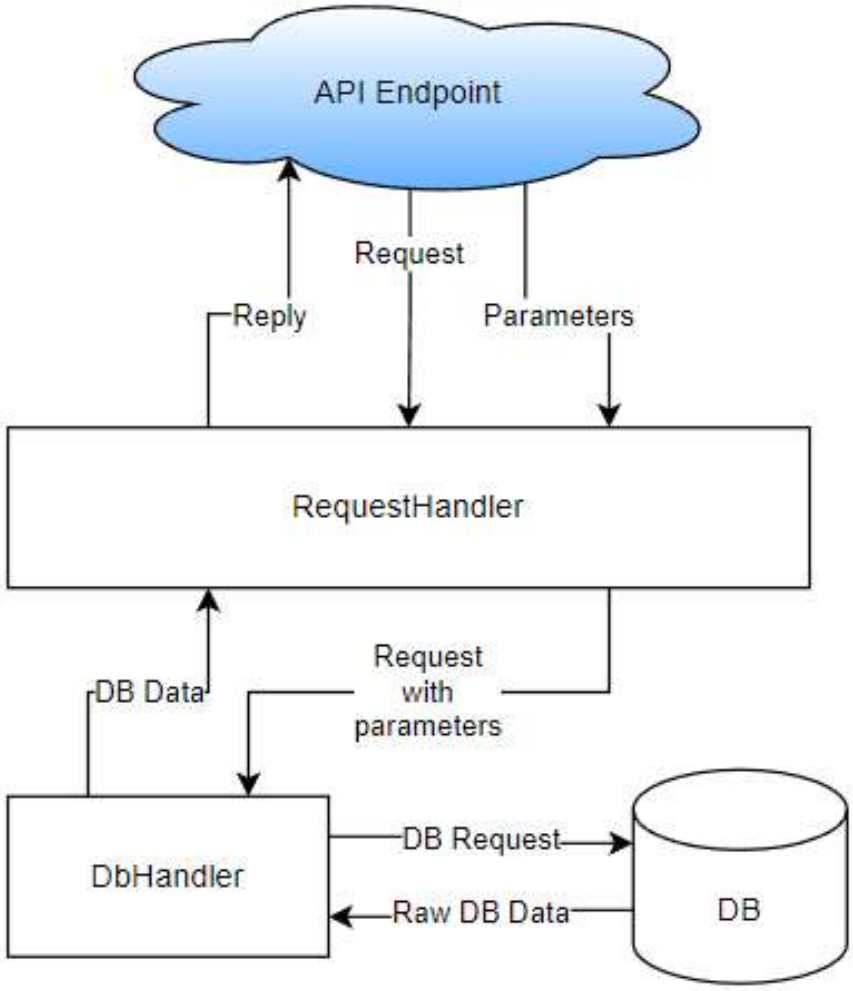


Рисунок 3.11 – Діаграма подорожі інформації по проєкту при запиті історичної інформації

Проте механізм автоматичного збору інформації у базу даних поєднує у собі обидва методи. У цьому випадку виконується більша частину методу для отримання інформації з інвертора, але без участі користувача, або інших зовнішніх факторів, які змушують проєкт виконувати дії. Процес змінюється, коли процес отримання та обробки інформації завершує отримання та обробки інформації з інвертора, та замість обернення отриманої інформації у заголовки та відправки на кінцеву точку, відбувається запуск збереження інформації у базу даних. Далі починається звичайний процес отримання параметрів необхідних для відкриття

підключення до бази даних, з послідуючим запитом до бази даних на запис інформації.

Це відбувається через необхідність запису у базу даних інформації з інвертора, яка далі буде використовуватись для історичних цілей, таких, як побудування порівняльних характеристик інформації з інвертора за декілька років неперервного використання. Також ця інформація використовується на сторінці проєкта «History», з якої будуються діаграми використання інвертора. Цей процес забезпечує постійний та надійний потік даних, необхідних для аналізу роботи інвертора, що дозволяє вчасно виявляти потенційні проблеми та покращувати його функціонал.

Запис інформації у базу даних відбувається у текстовому вигляді об’єкту JSON. Це надає базі даних більш стійку архітектуру, порівняно з архітектурою, де у базі даних зберігаються значення у їх вихідному вигляді. Також це спрощує взаємодію з базою даних, через відсутність потреби у зборі інформації у JSON об’єкт з вихідних значень отриманих з бази даних. Та через найсучасніші технології роботи у базі даних було встановлено, що робота з текстовим виглядом JSON об’єктів не є повільнішою, порівняно з іншими типами даних для збереження у базі даних.

Узагальнюючи логіку подорожі інформації по проєкту, можна зробити висновок, що уся топологія побудована навколо класу, який містить у собі більшу частину розподільчої логіки. Та у випадках, де потребується велика кількість вузькоспеціалізованої логіки для виконання певної дії, викликаються допоміжні класи.

Загальна діаграма для зображення подорожі інформації по проєкту, де зображені обидва раніше описані шляхи зображена на рисунку 3.12, де помітно, що клас “RequestHandler” є найбільшим компонентом, з найбільшою кількістю логічних взаємодій з рештою компонентів проєкту.

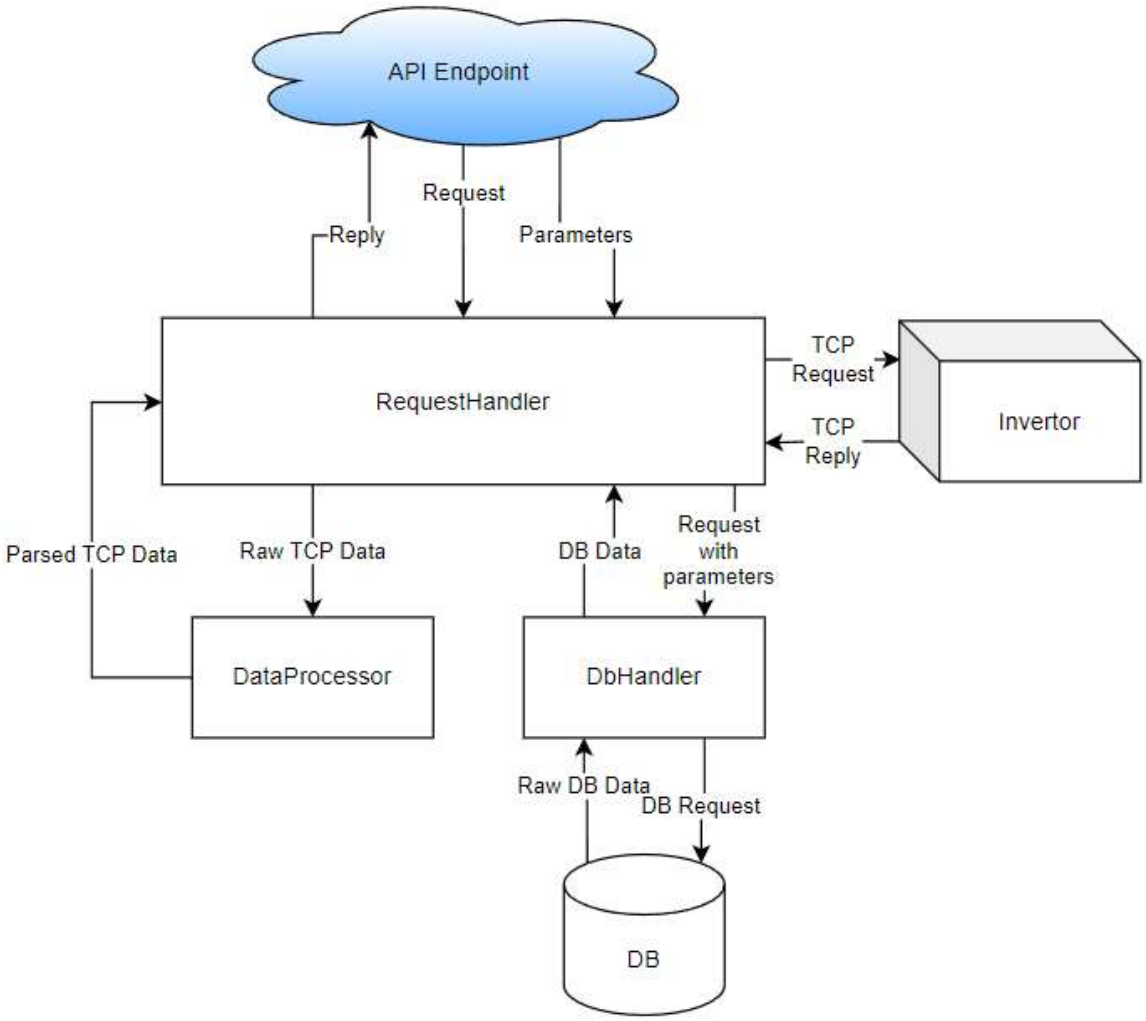


Рисунок 3.12 – Загальна діаграма подорожі інформації по проєкту

Отож, діаграма подорожі даних по проєкту ілюструє, як дані переміщуються проектом від початкового введення до кінцевого виходу. Процес починається, коли надходить запит на одну з кінцевих точок програмного інтерфейсу, який потім перевіряється та обробляються одним з раніше описаних конвеєрів обробки інформації. Звідти дані передаються назад до початку процесу, де вони перетворюються та обробляються відповідно до конкретних бізнес-правил у відповідь, яка надсилається назад, згідно запиту.

3.4 Висновки

У цьому завершальному розділі було розглянуто тестування та перевірку кінцевої системи проєкту за допомогою програмного забезпечення Postman, а також процес побудови графіків на основі отриманої інформації.

Кінцева система проєкту, яка використовує протокол HTTP, була протестована за допомогою Postman. Це програмне забезпечення дозволяє створювати сценарії з кількома параметрами для тестування. Тестування проводилося лише на кінцевих точках, які використовують метод "POST", оскільки кінцеві точки методу "GET" можна перевірити безпосередньо у веб-браузері. Було перевірено три конкретні кінцеві точки: «/api/v1/cache/timestamp», «/api/v1/cache/timestamp-range» і «/api/v1/cache/all».

Далі була проведена ретельна перевірка всіх даних, які надіслано та отримано на віртуальному пристрої. Це включало перевірку початкового запиту та всіх його полів на наявність значень, оскільки відсутність значень може призвести до пошкодження пристрою. Перевірка значень, отриманих від програмної частини проєкту, була виконана шляхом надсилання запиту «GET» до кінцевої точки «/api/v1/cache/latest», яка повертала файл JSON у текстовому форматі. Ця відповідь не була придатною для читання людиною, оскільки в ній були відсутні розриви рядків і містився великий обсяг інформації. Це було виправлено за допомогою онлайн-сервісу для форматування файлів JSON, який відновив розриви рядків і дозволив відображати вміст файлу у вигляді дерева.

Проект також включає підлеглий віртуальний пристрій, який було перевірено шляхом налаштування програмного забезпечення Postman для роботи з IP-адресою пристрою. Запит HTTP було надіслано на пристрій, і було підтверджено, що пристрій отримав відповідь і відобразив його у вікні серійної консолі. Відповідь потім було проаналізовано назад у програмний об’єкт за допомогою динамічного алгоритму, а отриманий об’єкт було перевірено на правильність, використовуючи попередньо збережену відповідь як зразок.

Після аналізу та коригування алгоритму розробки об’єкта JSON було реалізовано механізм надсилання HTTP-запитів, що наблизило віртуальний пристрій до завершення. Після цього пристрій був запущений і відправив запит до програмного комплексу проєкту, отримавши правильну відповідь. Відповідь була проаналізована на наявність необхідної інформації та відображена пристроєм.

На завершення тестування та верифікація кінцевої системи проєкту була виконана за допомогою програмного забезпечення Postman. Під час тестування було ретельно проаналізовано реакцію кожної точки на вхідні та вихідні дані, надіслані за протоколом HTTP, і підтверджено, що кінцеві точки виконують свої функції відповідно до встановлених вимог. Також було підтверджено, що підлеглий віртуальний пристрій працює коректно та виконує запрограмовані дії на основі отриманої інформації.

У тексті також описується процес побудови графіків з отриманої інформації для сторінки «Історія» веб-інтерфейсу. Для побудови графіків була обрана бібліотека Charts.js, реалізована в каталозі проєкту та підключена в заголовку веб-інтерфейсу. Для відображення графіків на сторінці було оголошено порожні полотна. Логіка відображення графіків з інформацією, отриманою з проєкту, реалізована у файлі "charting.js". Процес побудови графіків починається з отримання трьох елементів із документа HTML за допомогою «document.getElementById()» та отримання їхнього двовимірного контексту для малювання на них діаграм. Елементи мають ідентифікатори «historicChartDaily», «historicChartDailyTotals» і «historicChartTotal».

Додатково, для кращого управління та адаптації до різних розмірів екрану та пристроїв, було реалізовано використання адаптивних параметрів та налаштувань бібліотеки Charts.js. Це дозволило оптимізувати відображення графіків на різних пристроях та забезпечило зручний доступ до них. Крім того, були додані можливості змінювати типи та стилі графіків в залежності від вимог конкретної сторінки або відповідно до побажань користувачів, що дозволяє більш гнучко налаштовувати відображення даних у веб-інтерфейсі.

ВИСНОВКИ

У першому розділі було досліджено існуючі рішення для завдання проєкту. Це дослідження дало змогу не лише отримати цінні ідеї та зрозуміти складність завдання, але й детально вивчити доступні інструменти та бібліотеки, які могли б бути корисними під час реалізації проєкту. Аналіз аналогів дозволив визначити найефективніші підходи та потенційні недоліки. Також це дослідження дало визначити потребу у даному проєкті.

У другому розділі детально описано технологічний стек, на якому ґрунтувався проект у своїй завершеній формі. Ключовим обґрунтованим вибором було використання мови програмування C# для реалізації концепції проєкту, а також допоміжних мов програмування для реалізації підтримуючих компонентів. Описано також створення пристрою для взаємодії з програмним комплексом проєкту у віртуальному середовищі, що значно прискорило тестування.

Значний час був приділений аналізу альтернативних варіантів технологічного стеку для забезпечення оптимальної продуктивності та масштабованості проєкту, враховуючи його потреби та специфіку функціональності.

У третьому розділі детально описано процес тестування та перевірки кінцевої системи за допомогою програмного забезпечення Postman, а також побудову графіків на основі отриманої інформації. Кінцева система пройшла комплексне тестування через Postman, що дозволило аналізувати реакцію кожної кінцевої точки на вхідні та вихідні дані.

Окрім цього, розглянуто процес побудови графіків на основі зібраної інформації за допомогою бібліотеки Charts.js. У рамках тестування було враховано різноманітні сценарії взаємодії з системою, включаючи випадки нормального функціонування, обробку помилок та навантаження на систему. Детально проаналізовано результати тестів з метою виявлення потенційних проблем та вдосконалення якості програмного забезпечення.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Cyber-Physical Systems in Everyday Life: A Case Study Collection, Springer. 2019. 286 p.
2. Cyber-Physical Systems: From Theory to Practice, Elsevier. 2017. 342 p.
3. SUN-10K-SG04LP1-EU User Manual, Oracle, 2005. URL: https://www.oracle.com/a/ocom/docs/sun-10k-sg04lp1-eu-user-manual.pdf (дата звернення: 8.04.2024).
4. Конфігураційні файли для інверторів, StephanJoubert. 2023. URL: https://github.com/StephanJoubert/home_assistant_solarman/tree/main/custom_components/solarman/inverter_definitions (дата звернення: 10.04.2024).
5. Dynamic Algorithms for Network Problems, Springer. 2016. 486 p.
6. Li J., Wang Q., Wang C., Cao N., Ren K. and Lou W., Fuzzy Keyword Search over Encrypted Data in Cloud Computing, *2010 Proceedings IEEE INFOCOM*, San Diego, CA, USA, 2010. p. 1-5. DOI: 10.1109/INFCOM.2010.5462196. URL: https://ieeexplore.ieee.org/document/5462196 (дата звернення: 11.04.2024).
7. YamlDotNet User Guide, YamlDotNet. 2021. URL: https://www.yaml.info/learn/user-guide.html (дата звернення: 11.04.2024).
8. DEYE Logger AT Command Manual, DEYE Technology. 2017. URL: https://www.deye.com/download/DEYE-Logger-AT-Command-Manual.pdf (дата звернення: 20.05.2024).
9. Introduction to AT Commands, GSM-Modem.de. 2019,.URL: https://www.gsm-modem.de/M2M/m2m-university/m2m-tutorials/at-commands/ (дата звернення: 13.04.2024).
10. Designing Fail-Proof Algorithms: Techniques for Building Reliable and Robust Software Systems, O'Reilly Media. 2017. 244 p.
11. SolarMan Datalogger User Manual, SolarMan. 2021. URL: https://www.solarmanguide.com/datalogger-user-manual/ (дата звернення: 13.04.2024).

12. DEYE Inverter User Manual, DEYE Technology. 2021. URL: https://www.deye.com/download/DEYE-Inverter-User-Manual.pdf (дата звернення: 14.04.2024).
13. Smart Home IoT Protocols and Devices, Packt Publishing. 2018. 318 p.
14. Python Documentation, Python Software Foundation. 2021. URL: https://docs.python.org/3/ (дата звернення: 15.04.2024).
15. C++ Reference, cppreference.com. 2021. URL: https://en.cPpreference.com/w/ (дата звернення: 15.04.2024).
16. Java Documentation, Oracle. 2021. URL: https://docs.oracle.com/en/java/ (дата звернення: 15.04.2024).
17. MODBUS CRC Calculation and Usage, Control Engineering. 2017. URL: https://www.controleng.com/articles/modbus-crc-calculation-and-usage/ (дата звернення: 16.04.2024).
18. Building Evolutionary Architectures: Support Constant Change, O'Reilly Media. 2017. URL: https://www.oreilly.com/library/view/building-evolutionary-architectures/9781491986383/ (дата звернення: 17.04.2024).
19. Pro C# Project Management, Apress. 2014. 352 p.
20. JavaScript Documentation, Mozilla Developer Network. 2021. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (дата звернення: 19.04.2024).
21. Node.js Documentation, Node.js. 2021. URL: https://nodejs.org/en/docs/ (дата звернення: 19.04.2024).
22. PHP Documentation, PHP: Hypertext Preprocessor. 2021. URL: https://www.php.net/docs.php (дата звернення: 19.04.2024).
23. Cross-Platform Development with .NET Core, Microsoft. 2021. URL: https://docs.microsoft.com/en-us/dotnet/core/tutorials/cross-platform-app (дата звернення: 23.04.2024).

24. C# Coding Conventions, Microsoft. 2021. URL: https://docs.microsoft.com/en-us/dotnet/csharp/fundamentals/coding-style/coding-conventions (дата звернення: 25.04.2024).
25. Writing High-Quality Code in C#, Pluralsight. 2019. URL: https://www.pluralsight.com/guides/writing-high-quality-code-in-csharp (дата звернення: 26.04.2024).
26. Secure Coding Guidelines for C#, OWASP Foundation. 2021. URL: https://owasp.org/www-project-secure-coding-practices/v3.1/dotnet.html (дата звернення: 28.04.2024).
27. C# in Depth, Manning Publications. 2019. 592 p.
28. Entity Framework Core Documentation, Microsoft. 2021. URL: https://docs.microsoft.com/en-us/ef/core/ (дата звернення: 30.04.2024).
29. C# Hardware Access, Microsoft. 2021. URL: https://docs.microsoft.com/en-us/dotnet/standard/microservices-architecture/microservice-ddd-cqrs-patterns/infrastructure-persistence-layer-implements-entity-framework-core (дата звернення: 01.05.2024).
30. ASP.NET Core Web API, Microsoft. 2021. URL: https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-web-api (дата звернення: 02.05.2024).
31. Building RESTful APIs with ASP.NET Core, Microsoft. 2021. URL: https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-web-api&tabs=visual-studio (дата звернення: 02.05.2024).
32. HTML Documentation, Mozilla Developer Network. 2021. URL: https://developer.mozilla.org/en-US/docs/Web/HTML (дата звернення: 05.05.2024).
33. CSS Documentation, Mozilla Developer Network. 2021. URL: https://developer.mozilla.org/en-US/docs/Web/CSS (дата звернення: 06.05.2024).

34. ASP.NET Core vs ASP.NET: Comparing Benefits, CodeProject. 2020. URL: https://www.codeproject.com/Articles/1225315/ASP-NET-Core-vs-ASP-NET-Comparing-Benefits (дата звернення: 07.05.2024).
35. Razor Pages in ASP.NET Core, Microsoft. 2021. URL: https://docs.microsoft.com/en-us/aspnet/core/razor-pages/ (дата звернення: 07.05.2024).
36. ASP.NET Core Web Server Implementations, Microsoft. 2021. URL: https://docs.microsoft.com/en-us/aspnet/core/fundamentals/servers (дата звернення: 07.05.2024).
37. ASP.NET Core Web Application Planning and Design, Microsoft. 2021. URL: https://docs.microsoft.com/en-us/aspnet/core/fundamentals/index (дата звернення: 08.05.2024).
38. Securing ASP.NET Core Applications, Microsoft. 2021. URL: https://docs.microsoft.com/en-us/aspnet/core/security/ (дата звернення: 09.05.2024).
39. Building an API with ASP.NET Core without Visual Studio, Code Maze. 2021. URL: https://code-maze.com/building-api-aspnet-core-without-visual-studio/ (дата звернення: 10.05.2024).
40. ASP.NET Core Tutorial for Beginners, freeCodeCamp. 2020. URL: https://www.freecodecamp.org/news/asp-net-core-tutorial-for-beginners/ (дата звернення: 10.05.2024).
41. Microservices Architecture: Aligning Principles, Practices, and Culture, Microsoft. 2019. URL: https://docs.microsoft.com/en-us/dotnet/architecture/microservices/ (дата звернення: 12.05.2024).
42. ASP.NET Core Performance Best Practices, Microsoft. 2021. URL: https://docs.microsoft.com/en-us/aspnet/core/performance/performance-best-practices (дата звернення: 14.05.2024).

43. .NET Core vs .NET Framework: Which One to Choose in 2021, ScienceSoft. 2021. URL: https://www.scnsoft.com/blog/net-core-vs-net-framework (дата звернення: 15.05.2024).
44. Docker Documentation, Docker. 2021. URL: https://docs.docker.com/ (дата звернення: 16.05.2024).
45. Hosting ASP.NET Core on Linux with Docker, Microsoft. 2021. URL: https://docs.microsoft.com/en-us/aspnet/core/host-and-deploy/docker/building-net-docker-images (дата звернення: 16.05.2024).
46. HTTP Documentation, Mozilla Developer Network. 2021. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP (дата звернення: 16.05.2024).
47. C# Bitwise and Shift Operators, Microsoft. 2021. URL: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/bitwise-and-shift-operators (дата звернення: 17.05.2024).
48. Configuration in ASP.NET Core, Microsoft. 2021. URL: https://docs.microsoft.com/en-us/aspnet/core/fundamentals/configuration/ (дата звернення: 17.05.2024).
49. Using YAML Configuration in ASP.NET Core, Exception Not Found. 2020. URL: https://www.exceptionnotfound.net/using-yaml-configuration-in-asp-net-core/ (дата звернення: 18.05.2024).
50. Front-end Web Development with ASP.NET Core, Microsoft. 2021. URL: https://docs.microsoft.com/en-us/aspnet/core/client-side/ (дата звернення: 18.05.2024).
51. Using AJAX in ASP.NET Core, Microsoft. 2021. URL: https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/working-with-forms#call-a-web-api-from-javascript (дата звернення: 18.05.2024).
52. Building a Blazor WebAssembly App with ASP.NET Core, Microsoft. 2021. URL: https://docs.microsoft.com/en-us/aspnet/core/blazor/host-and-deploy/webassembly (дата звернення: 19.05.2024).

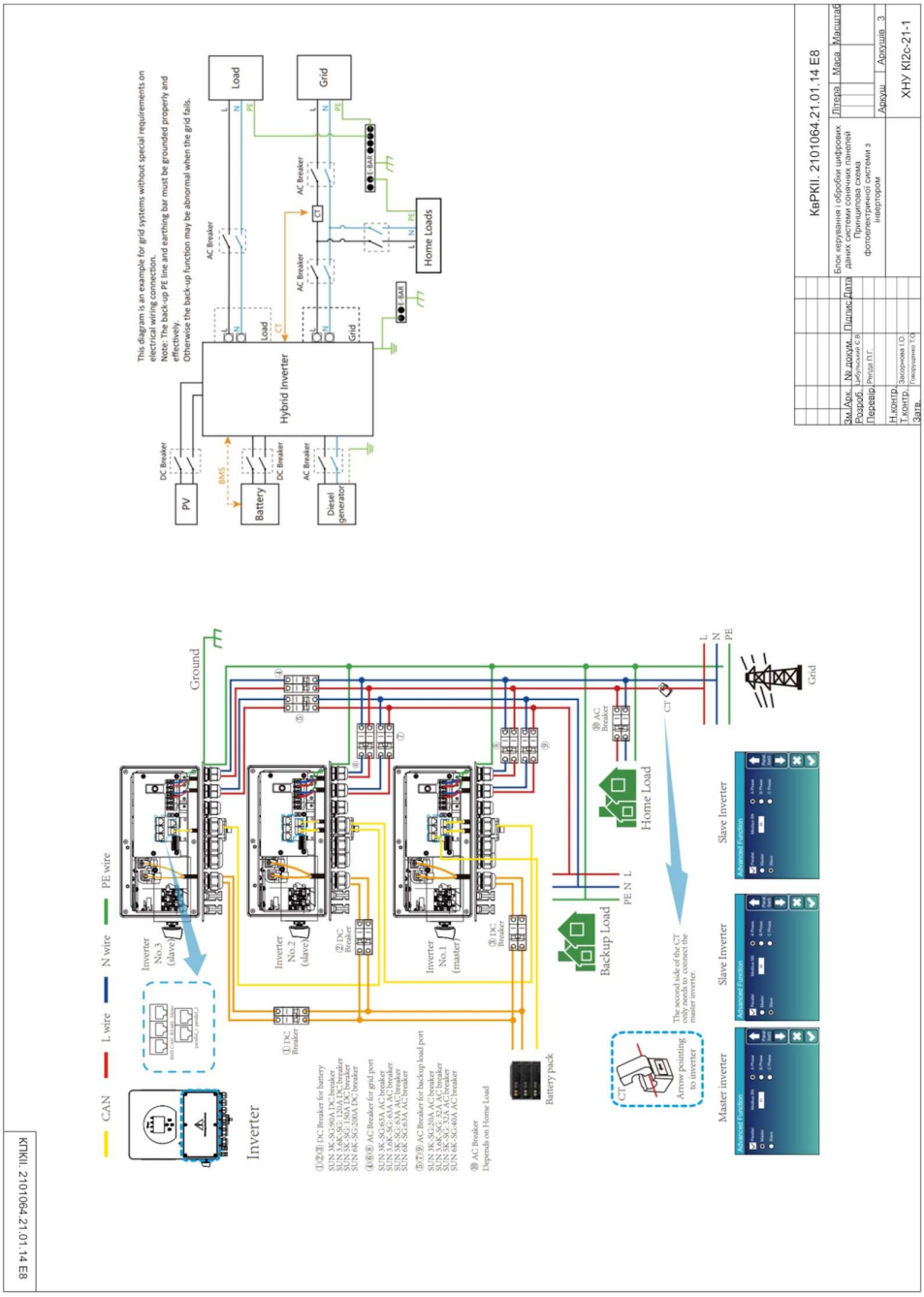
53. Using Entity Framework Core with MariaDB, MariaDB. 2021. URL: https://mariadb.com/kb/en/entity-framework-core/ (дата звернення: 19.05.2024).
54. Using Pomelo.EntityFrameworkCore.MySql with ASP.NET Core, Pomelo Foundation. 2021. URL: https://github.com/PomeloFoundation/Pomelo.EntityFrameworkCore.MySql/blob/main/src/Pomelo.EntityFrameworkCore.MySql.IntegrationTests/README.md (дата звернення: 20.05.2024).
55. Asynchronous Programming with Async and Await, Microsoft. 2021. URL: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/async/ (дата звернення: 20.05.2024).
56. MicroPython Documentation, MicroPython. 2021. URL: https://docs.micropython.org/en/latest/ (дата звернення: 20.05.2024).
57. Getting Started with MicroPython on the ESP32, Random Nerd Tutorials. 2021. URL: https://randomnerdtutorials.com/micropython-esp32-development-board/ (дата звернення: 20.05.2024).
58. Driving 7-Segment and 16x2 Displays with MicroPython, Adafruit Learning System. 2021. URL: https://learn.adafruit.com/driving-7-segment-and-16x2-displays-with-micropython (дата звернення: 21.05.2024).
59. I2C Communication with MicroPython, Random Nerd Tutorials. 2021. URL: https://randomnerdtutorials.com/micropython-i2c-communication/ (дата звернення: 22.05.2024).
60. I2C Documentation, MicroPython. 2021. URL: https://docs.micropython.org/en/latest/library/machine.I2C.html (дата звернення: 22.05.2024).
61. Getting Started with the Raspberry Pi Pico, Raspberry Pi. 2021. URL: https://www.raspberrypi.org/documentation/microcontrollers/raspberry-pi-pico.html (дата звернення: 23.05.2024).

62. I2C Communication with Raspberry Pi, CircuitBasics. 2021. URL: https://www.circuitbasics.com/raspberry-pi-i2c-communication-with-picluster-hat/ (дата звернення: 23.05.2024).

ДОДАТОК А

(обов'язковий)

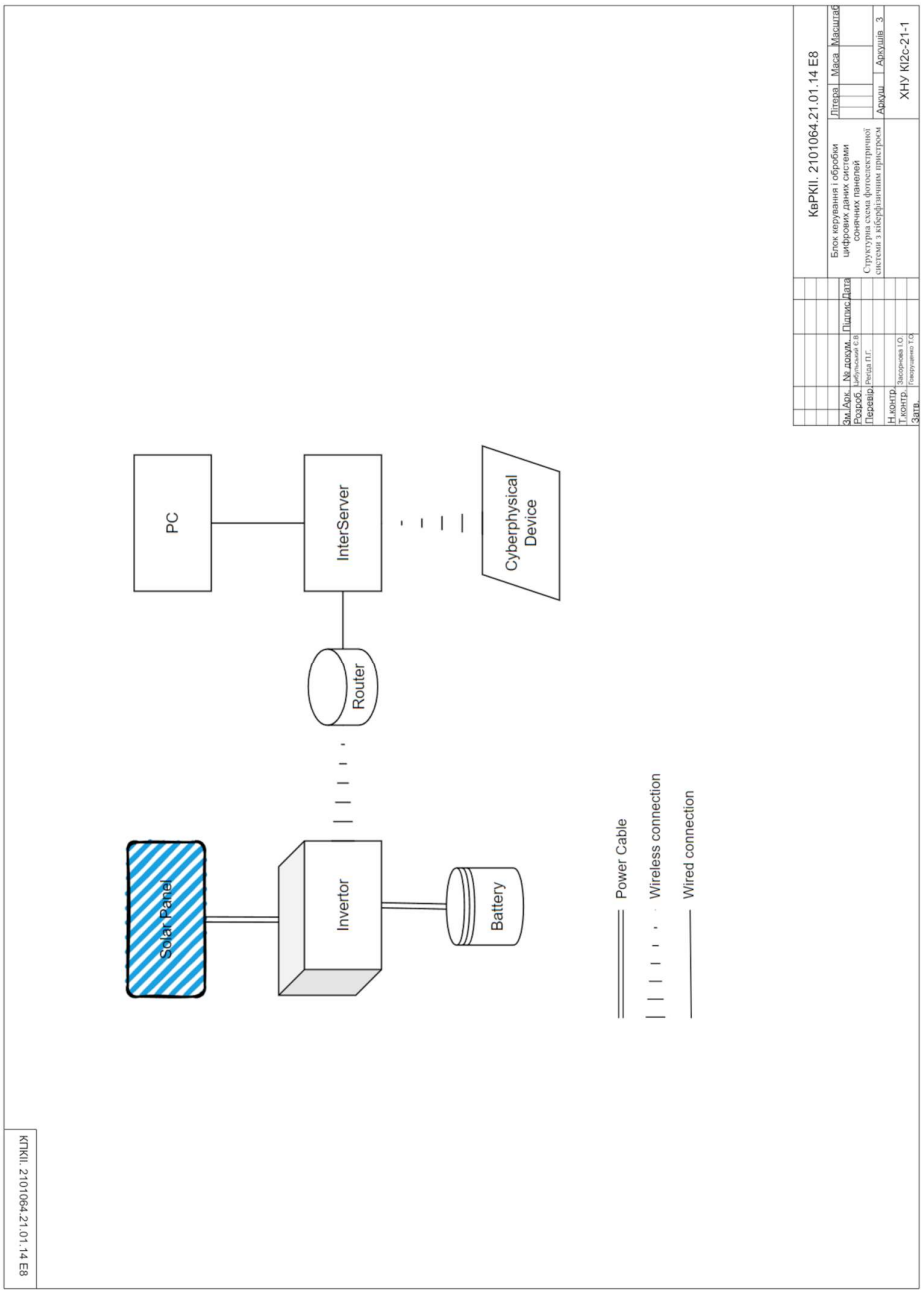
Копія креслення «Принципова схема фотоелектричної системи з інвертором»



ДОДАТОК Б

(обов’язковий)

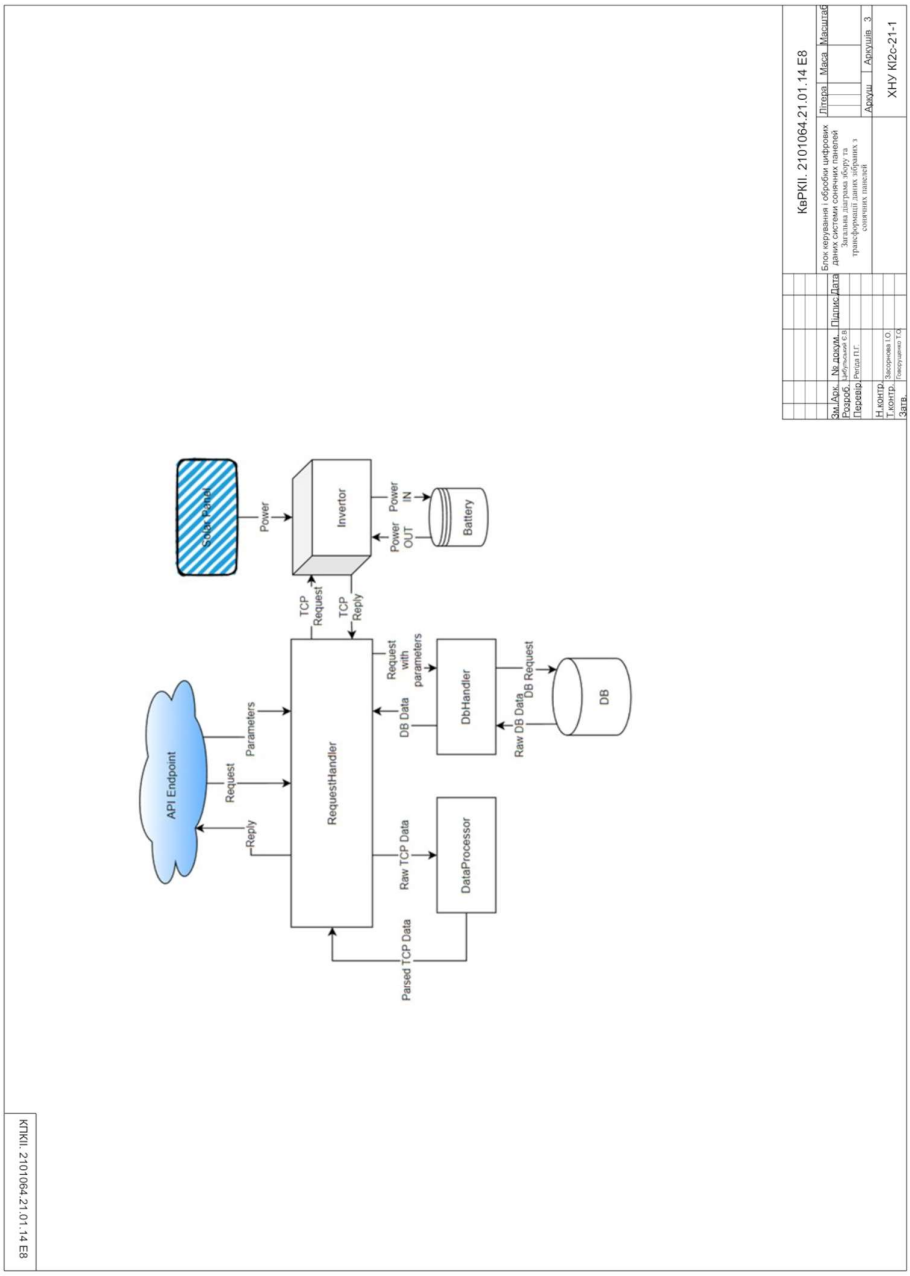
Копія креслення «Структурна схема фотоелектричної системи з кіберфізичним пристроєм»



ДОДАТОК В

(обов’язковий)

Копія креслення «Загальна діаграма збору та трансформації даних зібраних з сонячних панелей»



UNICHECK
by Turnitin

Ім'я користувача:
Кафедра КІ

ID перевірки:
1016276215

Дата перевірки:
23.05.2024 13:32:51 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
23.05.2024 15:28:20 EEST

ID користувача:
100005591

Назва документа: Цибульський_Блок керування і обробки цифрових даних системи сонячних панелей

Кількість сторінок: 96 **Кількість слів:** 15443 **Кількість символів:** 122067 **Розмір файлу:** 1.65 MB **ID файлу:** 1016067528

4.07%
Схожість

Найбільша схожість: 1.09% з Інтернет-джерелом (https://www.utn.uu.se/sts/student/wp-content/uploads/2021/12/2202_.

3.92% Джерела з Інтернету	580	Сторінка 98
1.66% Джерела з Бібліотеки	153	Сторінка 102

0.01% Цитат

Цитати	1	Сторінка 103

Не знайдено жодних посилань

0%
Вилучень

Немає вилучених джерел

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 0.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 10%**

	Документ		Сумарний збіг по Базі Даних	
ID: 127014 Назва: БКР Блок керування і обробки цифрових даних системи сонячних панелей Додано в БД: 2024-05-23 Автора: Є. В. Цибульський Керівники: П. Г. Регіда Консультанти: Опоненти:	Символи	Лексеми	Символи	Лексеми
	113205	833	868 (1%)	9 (1%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Дипломник: Цибульський Єгор Володимирович

Тема: Блок керування і обробки цифрових даних системи сонячних панелей

Спеціальність: 123 «Комп'ютерна інженерія»

Обсяг кваліфікаційної роботи:

Кількість листів креслень 3 Кількість сторінок записки 72

1. Короткий зміст роботи та прийнятих рішень: Метою кваліфікаційної роботи є створення програмно-апаратного комплексу, для зберігання та обробки даних з блоку керування фотоелектричними системами

2. Висновок про відповідність роботи дипломному завданню: Робота повністю відповідає поставленому завданню.

3. Характеристика виконання кожного розділу, ступінь використання останніх досягнень науки і техніки і передових методів роботи: У першому розділі було досліджено існуючі рішення для завдання проєкту. Аналіз аналогів дозволив визначити найефективніші підходи. Також це дослідження дало визначити основні етапи розробки даного проєкту. У другому розділі детально описано програмний та технологічний стек, а також допоміжних мов програмування для реалізації підтримуючих компонентів. Значний час був приділений для забезпечення оптимальної продуктивності та масштабованості проєкту. У третьому розділі детально описано процес тестування та перевірки кінцевої системи за допомогою програмного забезпечення Postman, а також побудову графіків на основі отриманої інформації.

4. Позитивні сторони роботи: висока практична цінність роботи з точки зору альтернативних джерел генерації електроенергії.

5. Негативні сторони роботи: інформація що збирається з інвертора використовується в не повному обсязі, варто додати більше діаграм, що дозволить більш точно оцінювати поточний стан системи.

6. Оцінка графічного оформлення та пояснювальної записки роботи: пояснювальна записка оформлена коректно, згідно діючих стандартів оформлення документації.

7. Відгук про роботу в цілому: робота виконана на належному науково-технічному рівні.

8. Інші зауваження: _____

9. Оцінка дипломної роботи: Розглянувши позитивні та негативні сторони представленої дипломної роботи вважаю, що робота заслуговує оцінки «відмінно», 5.0 (А).

Рецензент (прізвище, ім'я, по батькові, посада, місце роботи) _____
Мартинюк Валерій Володимирович,
зав. каф. АКІТтаР

"5" 06 2024 р. _____ (підпис)

Завідувачу кафедри КІІС
д-р.техн.наук, проф. Говорущенко Т. О.

Цибульського Єгора Володимировича
ПІБ здобувача вищої освіти

ФІТ, 3 курсу, групи КІ2с-21-1

ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності у Хмельницькому національному університеті» від 01.07.2022, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений (а). Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений(а) та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

3 червня 2024 року

РІШЕННЯ ЕКСПЕРТНОЇ КОМІСІЇ

КАФЕДРИ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: Блок керування і обробки цифрових даних системи сонячних панелей

Автор: Цибульський Єгор Володимирович

Спеціальність: 123– Комп'ютерна інженерія

Освітня програма: освітньо-професійна

Науковий керівник: Регіда Павло Геннадійович, старший викладач

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) запозичення розміщені в розділах аналізу існуючих аналогів та прототипів, які не описують безпосередньо авторське дослідження і не стосуються результатів роботи;
2) усі запозичення фрагментарні, або мають належним чином оформленні посилання;
3) більшість запозичень яких є джерела з інтернету, кількість запозичень з яких становить 3.92%;
4) запозичення з друкованих ресурсів, та ресурсів з бібліотеки становить 1.66%
5) решта зафіксованих системою ознак модифікацій тексту відносяться до перекладу англомовних ресурсів на українську мову.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості, складає 4.07% і адресується до 580 першоджерела, що, з урахуванням наведених обґрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

Керівник роботи _____ П. Г. Регіда

Гарант ОП _____ С.М. Лисенко

Завідувач кафедри КІІС _____ Т. О. Говорущенко