Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка інформаційної системи спостереження та сповіщення про стан енергосистеми розумної квартири

Виконав: студент IV курсу, групи СНс-41
спеціальності 122 Комп’ютерні науки
(шифр і назва спеціальності)

_____ Кібіткін Д.С.
(підпис) (прізвище та ініціали)

Керівник _____ Шимчук Г.В.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Липак Г.І
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Боднарчук І.О.
(підпис) (прізвище та ініціали)

Рецензент _____ Козак Р.О.
(підпис) (прізвище та ініціали)

Тернопіль
2026

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
«__» _червня_ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)

за спеціальністю 122 Комп’ютерні науки
(шифр і назва спеціальності)

Студенту Кібіткіну Денису Сергійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Розробка інформаційної системи спостереження та сповіщення про стан енергосистеми розумної квартири.

Керівник роботи Шимчук Григорій Валерійович, старший викладач кафедри комп’ютерних наук
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «*14*» *травня* 2026 року № *4/9-237*

2. Термін подання студентом завершеної роботи 21 червня 2026 р.

3. Вихідні дані до роботи Технічна документація та телеметричні дані гібридного Інвертора Must PV18-24/48V; параметри роботи побутової енергосистеми розумної квартири; Дані, отримані через інтерфейс USB-RS485; програмні засоби Python.

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. 1. Аналіз предметної області та постановка завдання. 1.1 Особливості моніторингу Енергосистем розумної квартири. 1.2 Аналіз існуючих системи моніторингу. 1.3 Обґрунтування вибору технологій та середовища розробки. 1.4 Висновки до першого розділу. 2. Проектування архітектури системи. 2.1 Архітектура системи моніторингу. 2.2 Логічна схема обміну даними. 2.3 Проектування інтерфейсу веб-панелі. 2.4 Висновки до другого розділу. 3. Реалізація та розгортання системи. 3.1 Реалізація серверної частини (скрипт для збору даних інвертора). 3.2 Реалізація веб-панелі (Flask-додаток). 3.3 Реалізація системи візуалізації даних (графіки та історичні показники). 3.4 Реалізація підсистеми сповіщень за допомогою Telegram-бота. 3.5 Розгортання системи на базі Proxmox VE. 3.6 Тестування та перевірка працездатності системи. 3.7 Висновки до третього розділу. 4. Безпека життєдіяльності, основи охорони праці. Висновки. Перелік джерел. Додатки.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень, слайдів)
1. Титульна сторінка. 2. Актуальність роботи. 3. Мета і завдання роботи. 4. Практичне значення одержаних результатів. 5. Інформаційна система моніторингу. 6. Порівняльний аналіз існуючих рішень. 7. Вимоги до системи моніторингу енергопроцесів інвертора. 8. Архітектура системи моніторингу інвертора. 9. Розробка бази даних моніторингу інвертора. 10. Сторінки: головна та графіки. 11. Сторінки: головна та графіки з телефону. 12. Telegram-бот: приклад сповіщень. 13. Telegram-бот: статус системи. 14. Розгортання. 15. Робочий прототип. 16. Висновки

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	Мариненко Сергій Юрійович	02.06.2026	10.06.2026

7. Дата видачі завдання 26 січня 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	26.01.2026	
2.	Підбір та опрацювання літературних джерел по темі кваліфікаційної роботи	27.01.2026-16.02.2026	
3.	Виконання дослідження щодо особливостей моніторингу енергосистем розумної квартири. Розроблення системи спостереження за енергопроцесами розумного домогосподарства.	17.02.2026-10.05.2026	
4.	Оформлення розділу «Аналіз предметної області та	11.05.2026-17.05.2026	
	постановка завдання»		
5.	Оформлення розділу «Проектування архітектури	18.05.2026-24.05.2026	
	системи»		
6.	Оформлення розділу «Реалізація та розгортання	25.05.2026-31.05.2026	
	системи»		
7.	Виконання завдання до підрозділу «Безпека життєдіяльності»	01.06.2026-08.06.2026	
8.	Виконання завдання до підрозділу «Основи охорони	01.06.2026-08.06.2026	
	праці»		
9.	Оформлення кваліфікаційної роботи	09.06.2026-11.06.2026	
10.	Нормоконтроль	12.06.2026-15.06.2026	
11.	Перевірка на плагіат	16.06.2026	
12.	Попередній захист кваліфікаційної роботи	18.06.2026	
13.	Захист кваліфікаційної роботи	26.06.2026	

Студент _____ Кібіткін Д.С.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Шимчук Г.В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Розробка інформаційної системи спостереження та сповіщення про стан енергосистеми розумної квартири // Кваліфікаційна робота освітнього рівня «Бакалавр» // Кібіткін Денис Сергійович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних наук, група СНс-41 // Тернопіль, 2026 // С. 75, рис. – 54, табл. – 3, кресл. – 16, додат. – 3, бібліогр. – 47.

Ключові слова: інвертор Must, веб-панель, Flask, Python, MySQL, Chart.js, розумне домогосподарство, розумна квартира.

Об'єктом дослідження є веб-система моніторингу для побутових сонячних енергетичних установок у середовищі розумного домогосподарства, включно з інфраструктурою розумної квартири.

Мета роботи – створити веб-панель для відображення поточних та історичних параметрів гібридного інвертора Must у зручному інтерфейсі з можливістю інтеграції в системи автоматизації розумного домогосподарства.

У рамках проєкту реалізовано: збір даних з інвертора через USB-інтерфейс, збереження інформації у базі MySQL, розробку веб-інтерфейсу на Flask із використанням Bootstrap та Chart.js, а також побудову API для отримання часових рядів та реалізацію підсистеми сповіщень на основі Telegram-бота. Система працює автономно у локальній мережі та розгорнута у контейнері Proxmox, що робить її сумісною з концепціями децентралізованого моніторингу в розумних квартирах.

Результатом є функціональна веб-панель, що відображає ключові технічні параметри роботи інвертора, надає графіки за вибрані періоди та забезпечує стабільне оновлення даних. Розроблене рішення придатне до використання власниками домашніх сонячних систем у рамках розумного домогосподарства та може бути розширене у майбутньому.

ANNOTATION

Development of a software system for monitoring and alerting on energy consumption status of a smart apartment // Qualification work of the educational level «Bachelor» // Kibitkin Denys Serhiiovych // Ternopil Ivan Pulyu National Technical University, Computer and Information Systems and Software Engineering Faculty, Computer Sciences Department, group SNs-41 // Ternopil, 2026 // P. 75, fig. – 54, tabl. – 3, chair. – 16, annexes. – 3, references – 47.

Keywords: inverter Must, web panel, Flask, Python, MySQL, Chart.js, smart household, smart apartment.

The object of the research is a web-based monitoring system for household solar energy installations within the environment of a smart household, including the infrastructure of a smart apartment.

The goal of the work is to develop a web panel for displaying current and historical parameters of a Must hybrid inverter in a user-friendly interface with the possibility of integration into smart home automation systems.

Within the project, the following were implemented: data collection from the inverter via USB interface, storage of information in a MySQL database, development of a web interface using Flask with Bootstrap and Chart.js, as well as the creation of an API for retrieving time series data and notifications using Telegram bot. The system operates autonomously within a local network and is deployed in Proxmox containers, which makes it compatible with the concept of decentralized monitoring in smart apartments.

The result is a functional web panel that displays key technical parameters of the inverter, provides charts for selected time periods, and ensures stable data updates. The developed solution is suitable for use by owners of home solar systems within a smart household and can be further expanded in the future.

ЗМІСТ

ВСТУП 6
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ 8
1.1 Особливості моніторингу енергосистем розумної квартири 8
1.2 Аналіз існуючих рішень та систем моніторингу 11
1.3 Обґрунтування вибору технологій та середовища розробки 12
1.4 Висновки до першого розділу 14
РОЗДІЛ 2. ПРОЕКТУВАННЯ АРХІТЕКТУРИ СИСЕМИ 16
2.1 Архітектура системи моніторингу 16
2.2 Логічна схема обміну даними 19
2.3 Проєктування інтерфейсу веб-панелі 21
2.4 Висновок до другого розділу 23
РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА РОЗГОРТАННЯ СИСТЕМИ 25
3.1 Реалізація серверної частини (скрипт для збору даних інвертора) 25
3.2 Реалізація веб-панелі (Flask-додаток) 30
3.3 Реалізація системи візуалізації даних (графіки та історичні показники) 37
3.4 Реалізація підсистеми сповіщень за допомогою Telegram-бота 42
3.5 Розгортання системи на базі Proxmox VE 48
3.6 Тестування та перевірка працездатності системи 54
3.7 Висновок до третього розділу 61
РОЗДІЛ 4. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ 63
4.1 Безпека при роботі з постійним та змінним струмом 63
4.2 Організація безпечного робочого місця користувача персонального комп’ютера під час розробки та експлуатації системи моніторингу 65
4.3 Висновок до четвертого розділу 68
ВИСНОВКИ 70
ПЕРЕЛІК ДЖЕРЕЛ 72
ДОДАТКИ

ВСТУП

У сучасних умовах розвитку відновлюваної енергетики та впровадження концепцій розумного домогосподарства зростає потреба у зручних інструментах для моніторингу роботи малих чи середніх сонячних електростанцій власного використання [14, 18, 33, 36]. Побутові гібридні інвертори, такі як Must PV18-24/48V [10, 12], активно використовуються в приватних будинках та розумних квартирах, де вони забезпечують автономне та резервне живлення. Проте штатні програмні рішення виробника часто обмежені за функціональністю [7], не мають веб-інтерфейсу, доступні тільки локально чи з певної операційної системи.

Власники невеликих домашніх енергосистем у середовищі розумного домогосподарства потребують інструментів, які дозволяють:

– Переглядати поточні параметри роботи інвертора в режимі реального часу.

– Аналізувати історичні дані у вигляді графіків.

– Контролювати генерацію, споживання та роботу мережі.

– Працювати локально, без сторонніх серверів;

– Гарантувати збереження та резервування даних (при тому, щоб збереження було доступним, і, до прикладу, не потрібно було чекати 15хв на завантаження даних, що часто буває з методами збережння від виробників).

Для вирішення цих задач у рамках кваліфікаційної роботи було створено веб-панель моніторингу інвертора Must, що працює у локальній домашній інфраструктурі розумної квартири, розгорнутій на базі Proxmox. Система складається з двох компонентів:

– Python-скрипт, який щохвилини зчитує дані через USB та записує їх у MySQL базу даних.

– Веб-панель на Flask, що відображає поточний стан, будує графіки та забезпечує зручний інтерфейс, використовуючи інформацію, збережену в базі даних.

Розроблена система відповідає ключовим вимогам веб-технологій:

– Доступ через браузер на будь-якому пристрої.
– Сучасні інструменти фронтенду (Bootstrap, Chart.js).
– Використання шаблонів HTML (Jinja2).
– Динамічне оновлення даних через AJAX.
– Адаптивність інтерфейсу.

Практичне значення роботи полягає у створенні автономної системи моніторингу для розумного домогосподарства, яка не залежить від зовнішніх серверів і працює з високою швидкістю. Система розгорнута в Linux контейнері Proxmox VE на Ubuntu 24.04, підтримує автозапуск і резервне .

Метою кваліфікаційної роботи є розроблення та реалізація веб-панелі моніторингу інвертора Must, яка може бути складовою частиною системи автоматизації розумної квартири.

Основні завдання:

– Проаналізувати особливості інверторів у побутових енергосистемах.
– Обґрунтувати вибір технологій.
– Спроєктувати архітектуру рішення.
– Реалізувати веб-інтерфейс для відображення параметрів.
– Забезпечити розгортання в середовищі Proxmox.

Результатом є завершений веб-додаток, що дозволяє контролювати роботу інвертора та інтегрується в інфраструктуру розумного домогосподарства [19, 29].

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 Особливості моніторингу енергосистем розумної квартири

Гібридні інвертори серії Must PV18-24/48V активно застосовуються у побутових сонячних енергетичних системах для забезпечення автономного живлення, заряджання акумуляторів та стабілізації роботи електромережі [10, 11, 36]. Інвертор виконує одразу кілька ключових функцій: перетворення енергії з сонячних панелей, керування зарядом акумуляторних батарей, перемикання між джерелами живлення та аналіз енергетичних потоків у реальному часі. Зовнішній вигляд інвертора під час встановлення зображено на рисунку 1.1.

Рисунок 1.1 – Гібридний інвертор Must PV18-24/48V, встановлений у домашній енергетичній системі

У процесі роботи інвертор формує значну кількість телеметричних даних [26, 33], зокрема:

– Напругу та струм з сонячних панелей.
– Потужність PV-генерації.
– Напругу та стан батареї.
– Споживання навантаження.
– Роботу мережі (grid).
– Внутрішні температурні показники.
– Режим енергетичного використання (SBU, SUB, UTI, SOL).

Ці дані є критично важливими для оцінки ефективності роботи системи, виявлення аномалій, розуміння поведінки акумулятора та планування енергоспоживання. Саме тому зручний та наочний інструмент моніторингу є обов’язковою складовою будь-якої домашньої сонячної станції. Проте, стандартні рішення часто не дають потрібної “свободи” даних чи їх доступності (детальніше дивіться розділ 1.3 – Аналіз існуючих рішень).

У рамках даної кваліфікаційної роботи інвертор було під’єднано до домашньої серверної інфраструктури через USB–RS485 [7], підключення зображено на рисунку 1.2.

Рисунок 1.2 – Підключення інвертора Must через USB

Далі USB-адаптер було “прокинуто” на віртуальну машину Proxmox, що працює на моєму домашньому сервері, як зображено на рисунку 1.3. Це дає змогу зчитувати телеметрію напряму та отримувати “сирі” дані, які згодом можуть бути збережені в базі даних [8, 24].

Віртуальна машина працює на базі Windows 10, тому що на даному етапі це дає змогу роботи в “тандемі” – де ми можемо відкривати як програму PowerSolarMonitor від Must, так і запускати наш скрипт.

Рисунок 1.3 – Підключення інвертора Must до віртуальної машини

Дані з інвертора збираються щохвилини і зберігаються в локальній базі даних для подальшого відображення у веб-панелі [21, 24].

Таким чином, моніторинг гібридного інвертора Must потребує спеціалізованого веб-інструменту, який дозволить:

– Отримувати всі ключові параметри роботи інвертора.
– Забезпечувати локальний та незалежний доступ до інформації.
– Переглядати історію роботи у вигляді графіків.
– Працювати стабільно навіть без доступу до інтернету.

Саме ці вимоги стали основою створення веб-панелі моніторингу, розробленої у межах даної кваліфікаційної роботи.

1.2 Аналіз існуючих рішень та систем моніторингу

Зазвичай, виробники інверторів надають в комплекті програмне забезпечення для перегляду даних чи керування інвертором, проте такі рішення мають свою недоліки, серед яких:

– неможливість зробити витяг даних ззовні. Тобто програма не “віддає” дані сторонньому користувачу;

– довге очікування при експорті. Через зберігання даних не в СУБД, а в текстовому форматі, наприклад в CSV чи JSON, де після накопичення даних – для завершення експорту доводиться чекати 10-15 хвилин, що значно ускладнює роботу з телеметрією розумної квартири;

– відсутність контролю, де зберігаються дані;

– відсутність “RAW” (чистих) даних. Тобто відсутність даних, які можна в подальшому використовувати відповідно до потреб, як, наприклад, зберігання даних в MySQL базі даних [29, 31].

Наприклад, виробником інвертора Must пропонується фірмовий застосунок PowerSolarMonitor, який дозволяє переглядати параметри системи у реальному часі, інтерфейс застосунка PowerSolarMonitor зображено на рисунку 1.4.

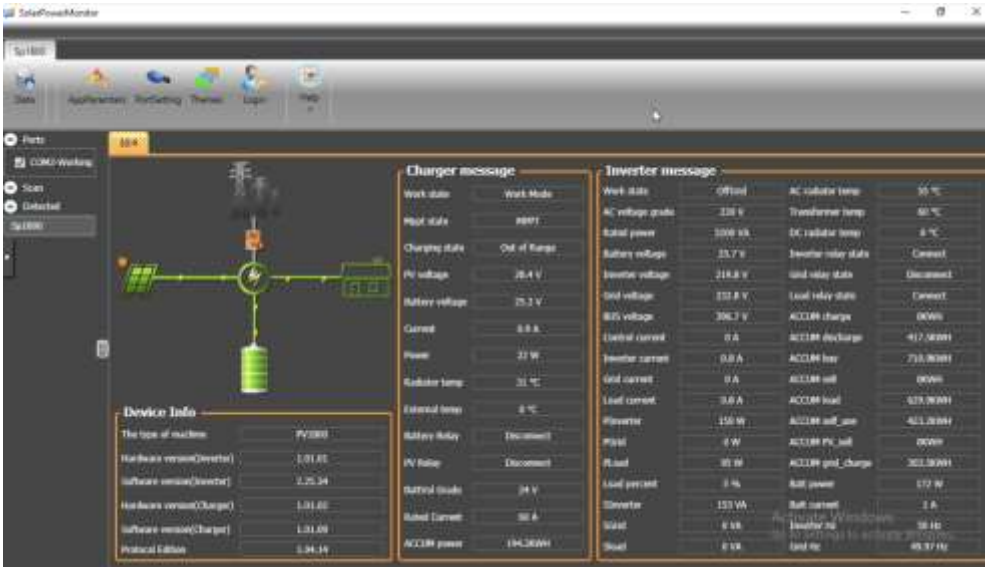

Рисунок 1.4 – Штатний застосунок PowerSolarMonitor від виробника Must

Проте, інструмент PowerSolarMonitor має низку обмежень, серед яких:

– Можливість перегляду даних тільки на ОС Windows.
– Неможливість віддаленого доступу до даних.
– Обмежена кількість параметрів для перегляду через інтерфейс.
– Відсутність веб-версії та адаптивного інтерфейсу.
– Неможливість переглядати дані з мобільного телефону.

Через ці недоліки власники домашніх енергетичних систем часто не мають можливості надійно стежити за роботою інвертора у випадках, коли доступ до мережі обмежений або з’єднання з сервером виробника стає недоступним.

Для забезпечення стабільної роботи та постійного доступу до даних є доцільним використання локальної системи моніторингу, яка працює у власній мережі та не залежить від зовнішніх сервісів.

А також, використання MySQL СУБД, яка дає можливість зберігати дані в базі даних та надавати швидкий доступ до них, або, за потреби – в майбутньому експортувати дані в любий інший формат (JSON, CSV та інші) не втрачаючи вже актуальні дані і запобігаючи помилками при експорті, оскільки MySQL досить ефективно зберігає великий об’єм даних [31, 34].

1.3 Обґрунтування вибору технологій та середовища розробки

Розроблення веб-панелі моніторингу інвертора Must потребує вибору такого технологічного стеку, який забезпечує стабільне зчитування даних, швидке відображення показників та доступність із різних пристроїв. Система повинна працювати автономно, у локальному середовищі, без залежності від серверів виробника.

Першим елементом проєктної інфраструктури стало підключення інвертора Must через USB–RS485 адаптер до серверної частини системи. Такий спосіб доступу до даних дозволяє напряму зчитувати телеметрію — напругу, струм, температуру, рівень заряду батареї, вироблену потужність та інші

параметри. На відміну від заводського програмного забезпечення, яке працює лише локально та зберігає дані у важких файлових архівах, USB-підключення дає змогу безперервно та швидко передавати вимірювання до веб-панелі.

Для роботи з USB-портом було обрано середовище Windows 10, розгорнуте у віртуальній машині Proxmox. Такий вибір є практичним, оскільки на Windows доступна заводська програма PowerSolarMonitor, яку зручно використовувати паралельно зі скриптом збору даних. Це дає можливість у режимі реального часу порівнювати значення з інвертора та перевіряти правильність роботи власного програмного коду. Крім того, Windows надає просту роботу з COM-портами, що спрощує процес налагодження.

Основною мовою створення скрипта збору даних обрано Python [2, 3]. Мова має велику кількість бібліотек для роботи з послідовними портами та зручні засоби для обробки байтових протоколів, якими інвертор передає телеметрію. Python дозволяє швидко реалізувати логіку зчитування даних, парсинг отриманих пакетів, об’єднання результатів та підготовку інформації до запису в базу даних.

Для збереження телеметрії використано MySQL [3, 4, 34]. База даних дозволяє ефективно опрацьовувати велику кількість записів — при частоті оновлення один запис на хвилину формується до півтори тисячі рядків на добу. MySQL забезпечує швидкі вибірки даних для графіків, можливість перегляду історії за будь-який період і значно перевершує файлову модель зберігання, яку використовує заводське програмне забезпечення. Доступ до бази даних здійснюється через ORM-бібліотеку SQLAlchemy, що спрощує роботу з таблицями.

Для створення веб-панелі було обрано Flask [1] — легкий та гнучкий веб-фреймворк на Python. Flask дозволяє створювати сторінки на основі шаблонів Jinja2, формувати REST-API і підтримує мінімалістичну структуру застосунку [1, 8, 13]. Такий підхід ідеально підходить для локальних проєктів, які мають працювати швидко, стабільно та без складної конфігурації.

Фронтенд реалізовано із використанням HTML, CSS, Bootstrap та JavaScript [5, 7, 16]. Bootstrap забезпечує адаптивний дизайн, завдяки чому веб-панель коректно відображається як на комп’ютері, так і на мобільних пристроях. Для візуалізації даних застосовано бібліотеку Chart.js [4, 30], яка надає інструменти побудови часових графіків і має зручні механізми анімації та масштабування.

Система розгорнута у контейнері Proxmox LXC [6, 35], що забезпечує стабільну роботу веб-панелі цілодобово, ізоляцію від інших сервісів та можливість швидкого резервного копіювання. Поєднання Proxmox, Windows-віртуальної машини та LXC-контейнера дозволило створити компактну та гнучку інфраструктуру, у якій всі елементи працюють автономно та не залежать від зовнішніх ресурсів.

У підсумку, вибраний стек технологій — Python, Flask, MySQL, Bootstrap і Chart.js — забезпечує надійну основу для побудови веб-панелі моніторингу, зручний доступ до даних користувачем і можливість подальшого розширення системи в межах дипломного проєкту.

1.4 Висновки до першого розділу

У першому розділі було проаналізовано особливості роботи гібридних інверторів Must та окреслено потребу в створенні зручної веб-панелі для їх моніторингу. Встановлено, що наявне заводське програмне забезпечення PowerSolarMonitor має суттєві обмеження: воно працює лише на Windows, не підтримує доступ із мобільних пристроїв, використовує важкі локальні файли для зберігання історичних даних та не забезпечує оперативного отримання інформації. У зв’язку з цим виникає потреба у створенні власного інструмента, який дозволяє переглядати дані інвертора швидко, з будь-якого пристрою та у локальній мережі.

Обґрунтовано вибір технологій, що були використані у роботі. Використання Python дає змогу ефективно працювати з послідовними портами

та обробляти телеметрію інвертора. Flask забезпечує гнучку та легку реалізацію веб-панелі, тоді як MySQL дозволяє надійно зберігати великі обсяги історичних даних і виконувати швидкі вибірки для побудови графіків. Використання Bootstrap і Chart.js на фронтенді забезпечує адаптивність та зручність інтерфейсу.

Також було обґрунтовано вибір інфраструктурного рішення на базі Proxmox, що дозволяє створити стабільне середовище для роботи скрипта збору даних та веб-панелі, а також забезпечує можливість паралельного використання заводської програми і власного програмного забезпечення в рамках однієї віртуальної машини.

Таким чином, перший розділ заклав основу для подальшого проєктування системи моніторингу та визначив технологічні рішення, які найкраще відповідають вимогам до функціональності, автономності та зручності користування [19, 20].

РОЗДІЛ 2. ПРОЕКТУВАННЯ АРХІТЕКТУРИ СИСЕМИ

2.1 Архітектура системи моніторингу

Архітектура веб-панелі моніторингу інвертора Must побудована на поєднанні кількох незалежних, але взаємопов’язаних компонентів, які працюють у межах локальної серверної інфраструктури Proxmox [20, 23, 35]. Такий підхід забезпечує стабільний збір даних, їх довготривале зберігання, швидке формування веб-інтерфейсу та доступ користувача до результатів роботи з будь-якого пристрою у локальній мережі.

Загальна структура системи складається з чотирьох основних блоків:

– Інвертор Must, підключений через USB–RS485 адаптер.

– Windows 10 віртуальна машина, де працює скрипт збору даних та заводська програма PowerSolarMonitor.

– Серверна база даних MySQL, розгорнута в окремому контейнері Proxmox.

– Веб-панель на Flask, що працює у LXC-контейнері та надає інтерфейс для перегляду показників.

Логічну архітектуру розробленої системи зображено на рисунку 2.1.

Рисунок 2.1 – Загальна архітектура системи моніторингу інвертора Must

У таблиці 2.1 наведені основні компоненти системи (схематично) та їх призначення.

Таблиця 2.1 — Основні компоненти системи та їх призначення

Компонент системи	**Призначення**	**Основні функції**
Інвертор Must	Джерело телеметрії	Генерація даних про PV, батарею, навантаження, мережу та робочі стани.
Windows 10 VM	Центр збору даних	Виконання Python-скрипта для отримання і запису даних, запуск PowerSolarMonitor, доступ до COM-порту.
MySQL LXC контейнер	Зберігання історії	Структуроване збереження телеметрії, можливість швидких SQL-вибірок.
Flask LXC контейнер	Відображення даних	Формування інтерфейсу, REST-API, побудова карток та графіків.
Proxmox VE	Інфраструктурна платформа	Віртуалізація, ізоляція сервісів, резервне копіювання, стабільність.

Інвертор Must є джерелом телеметричних даних, таких як значення напруги, струму, температури та режиму роботи. Через USB–RS485 адаптер пристрій підключений до фізичного хоста Proxmox, після чого USB-приймач прокидається всередину віртуальної машини Windows 10. Це дозволяє бачити інвертор як звичайний COM-порт і забезпечує коректну роботу як заводського програмного забезпечення, так і власного скрипта збору даних [8].

Windows 10 віртуальна машина виконує роль центра збору телеметрії. У ній працює Python-скрипт, який кожні 60 секунд опитує інвертор і отримує масив телеметричних даних. Паралельно може бути запущений додаток PowerSolarMonitor, який дозволяє візуально контролювати правильність зчитування параметрів та виконувати зміни налаштувань інвертора. Такий підхід створює зручне середовище для тестування, порівняння результатів та розробки.

Зібрані дані передаються на сервер бази даних MySQL, який розгорнуто у контейнері Proxmox. База MySQL зберігає історичні дані у структурованому

вигляді, забезпечує швидкий доступ до великої кількості записів і дозволяє фронтенду отримувати вибірки даних у будь-якому часовому проміжку. Завдяки цьому веб-панель може формувати графіки та відображати актуальні значення без затримок.

Веб-панель реалізована як окремий Flask-додаток, розміщений у LXC-контейнері. Вона отримує дані з бази, обробляє їх і надає користувачу інтерфейс для перегляду стану інвертора. Користувач може переглядати картки поточних параметрів, схему потоків енергії та деталізовані графіки за вибраний період часу. Панель адаптована для роботи як на ПК, так і на смартфонах.

Уся система функціонує всередині домашньої серверної інфраструктури Proxmox [27], що забезпечує такі переваги:

– Ізольованість кожного компоненту (VM, MySQL контейнер, Flask контейнер).

– Високу стабільність роботи.

– Можливість створювати резервні копії;

– Доступність сервісу в локальній мережі 24/7.

– Гнучкість у розширенні або перенесенні окремих частин системи.

Таким чином, архітектура системи є модульною та легко масштабованою. Дані з інвертора без затримок надходять до центральної бази даних, після чого миттєво передаються у веб-панель, що гарантує користувачу актуальну та достовірну інформацію у будь-який момент часу.

Така архітектура органічно вписується у типову домашню інфраструктуру розумного домогосподарства, де різні сенсори, сервери та мережеві пристрої взаємодіють між собою у локальній мережі [36]. Завдяки цьому веб-панель може виконувати роль окремого модуля моніторингу в структурі розумної квартири, забезпечуючи постійний доступ до енергетичних показників без залучення зовнішніх сервісів.

2.2 Логічна схема обміну даними

Робота системи моніторингу інвертора Must ґрунтується на послідовному обміні даними між окремими компонентами, кожен з яких виконує чітко визначену функцію [11, 28]. На відміну від загальної архітектури, що описує структуру системи в цілому, логічна схема пояснює шлях даних — від моменту їх появи на інверторі до відображення на екрані користувача.

Загальну послідовність обробки даних наведено на рисунку 2.2.

Рисунок 2.2 – Логічна схема обміну даними в системі моніторингу інвертора Must

Отримання даних починається на стороні інвертора. Пристрій формує телеметричні значення кожні кілька секунд, і ці параметри доступні через USB-інтерфейс. Windows-віртуальна машина, у яку прокидається USB-порт, бачить інвертор як стандартний COM-порт. Такий спосіб підключення дозволяє Python-скрипту для збору телеметрії працювати з обладнанням безпосередньо та отримувати «свіжі» дані незалежно від наявності інтернету.

Python-скрипт для збору даних виконується циклічно з інтервалом у 60 секунд [25, 33]. На кожній ітерації він посилає інвертору запит, отримує відповідь у вигляді масиву байтів та перетворює його у структурований набір значень. Далі скрипт формує JSON-об’єкт і записує дані до таблиці бази даних MySQL, що зберігається в окремому контейнері Proxmox. Це зберігає всю історію вимірювань, що дозволяє будувати графіки за будь-який проміжок часу [34].

Фронтенд-частина системи працює в окремому Flask-контейнері. Коли користувач відкриває веб-панель, Flask виконує запити до бази даних та отримує останній запис з телеметрією. Ці дані показуються у вигляді карток, схеми потоків енергії та агрегованих показників. Усі розрахунки для відображення (наприклад, відсоток заряду батареї чи потужність за формулою $P = U \times I$) виконуються безпосередньо у Flask-додатку.

Для побудови графіків використовується окремий REST-endpoint /api/timeseries. Коли користувач натискає кнопку «Оновити» або змінює параметри, браузер надсилає запит JavaScript-функцією fetch. Flask формує вибірку з MySQL, повертає дані у форматі JSON, і графік оновлюється без перезавантаження сторінки. Завдяки цьому веб-панель працює швидко та має сучасний інтерактивний інтерфейс.

Уся система працює повністю автономно у межах домашньої мережі. Навіть якщо інтернет відсутній, інвертор продовжує передавати дані у Python-скрипт для запису даних, а веб-панель працює без обмежень. Це робить рішення зручним та надійним для повсякденного використання.

Таким чином, логічна схема обміну даними демонструє чіткий поділ ролей між компонентами системи та забезпечує безперервний, стабільний і надійний спосіб отримання і відображення даних користувачу.

2.3 Проєктування інтерфейсу веб-панелі

Першим етапом проєктування інтерфейсу стало створення загального макета головної сторінки (дашборду), який демонструє логічне розташування ключових елементів системи [15-16]. На рисунку 2.3 подано структуру інтерфейсу у десктопному вигляді: у верхній частині розташована навігаційна панель, ліворуч — блок "Статус підключення до Must", праворуч — секція поточних значень. Нижче обох панелей розміщено блок із відображенням потоків енергії. Така структура забезпечує швидкий доступ до основних показників та дає змогу користувачу одразу оцінити стан системи.

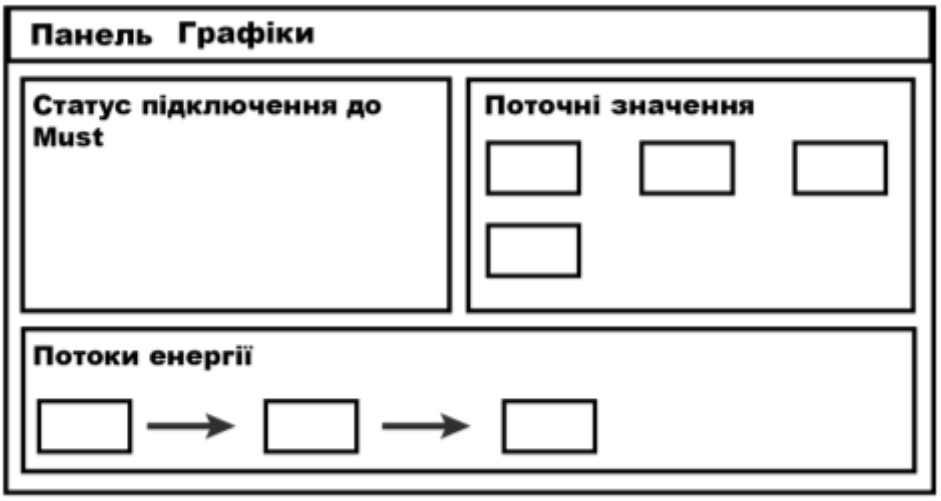

Рисунок 2.3 – Макет інтерфейсу веб-панелі (дашборд, десктопна версія)

Для інтерфейсу сторінки графіків було створено окремий макет, наведений на рисунку 2.4. На ньому показано мінімалістичну структуру сторінки із засобами вибору параметра, часової глибини та кнопкою оновлення. Центральну частину займає область побудови графіка. Така композиція

дозволяє зручно взаємодіяти з історичними даними й не перевантажує користувача зайвими елементами.

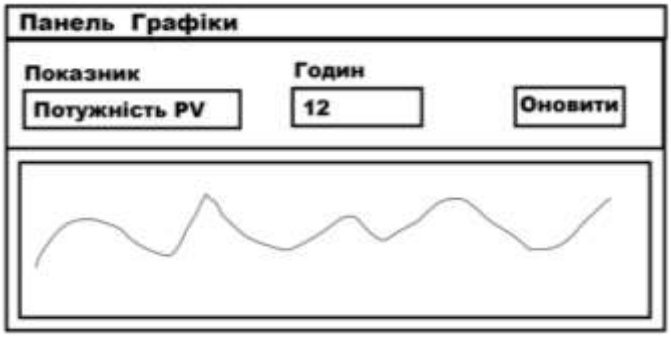

Рисунок 2.4 – Макет сторінки графіків

Оскільки веб-панель повинна коректно працювати на мобільних пристроях, було спроєктовано також адаптивний mobile-first макет [7, 30]. На рисунку 2.5 представлено мобільну версію дашборду, де елементи інтерфейсу розташовані вертикально, щоб забезпечити зручність перегляду з невеликих екранів. Основні блоки — статус підключення, поточні значення та потоки енергії — збережені, однак їх композиція оптимізована під портретну орієнтацію смартфона.

Рисунок 2.5 – Мобільна версія дашборду

Окремо було підготовлено мобільний макет сторінки графіків, поданий на рисунку 2.6. Мобільна версія повторює функціональність десктопної, зберігаючи вибір показника, поле для введення кількості годин та кнопку оновлення. Графік розташований у нижній частині екрана й займає всю доступну ширину, що забезпечує зручність аналізу даних на мобільному пристрої.

Рисунок 2.6 – Макет мобільно версії графіків

Окрему увагу приділено адаптивності інтерфейсу, оскільки в умовах розумної квартири користувач часто взаємодіє із системою не лише з комп’ютера, але й зі смартфонів, планшетів чи домашніх панелей управління. Мінімалістичний та мобільний дизайн дозволяє без проблем використовувати панель у повсякденному побуті як частину екосистеми розумного домогосподарства [17].

2.4 Висновок до другого розділу

У другому розділі було детально проаналізовано архітектуру розробленої веб-системи моніторингу інвертора Must та розглянуто логіку взаємодії між її основними компонентами. Визначено, що система має модульну структуру, яка

включає інвертор як джерело телеметрії, Windows-віртуальну машину зі скриптом збору даних, сервер бази MySQL та веб-панель, реалізовану у вигляді Flask-додатку. Така організація забезпечує ізоляцію окремих частин, гнучкість у масштабуванні та можливість незалежного оновлення або заміни компонентів [19, 27].

Побудована логічна схема обміну даними демонструє послідовний рух інформації: від отримання телеметрії через USB-порт до збереження у базі даних та подальшого відображення в інтерфейсі користувача. Це дозволяє ясно уявити внутрішні процеси системи, спрощує налагодження та забезпечує надійну роботу в автономному режимі, без залучення зовнішніх сервісів.

Окрему увагу в розділі було приділено проєктуванню інтерфейсу веб-панелі. Створені мінімалістичні wireframe-макети (десктопні та мобільні) відображають логічну структуру дашборду та сторінки графіків, визначають розташування ключових елементів і демонструють адаптивний підхід до побудови UI. Додавання макетів дозволило сформувати цілісне бачення майбутнього інтерфейсу ще до етапу реалізації та забезпечило узгодженість між архітектурою системи та її візуальною частиною.

У сукупності проведений аналіз підтверджує, що обраний підхід дає змогу створити стабільну, автономну та зручну у використанні систему моніторингу інвертора. Другий розділ сформував цілісну концептуальну основу, на якій базується подальша реалізація серверної, клієнтської та візуальної частин веб-застосунку, що буде розглянуто у наступному розділі.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА РОЗГОРТАННЯ СИСТЕМИ

3.1 Реалізація серверної частини (скрипт для збору даних інвертора)

Серверна частина системи моніторингу відповідає за безперервний збір телеметричних даних з інвертора Must та передачу їх до центральної бази даних MySQL [13, 33]. Цей компонент є необхідною основою роботи всієї веб-панелі, оскільки забезпечує формування історії вимірювань та доступ до актуальних значень параметрів у реальному часі. У даній роботі збір даних реалізовано у вигляді Python-скрипта, який працює всередині віртуальної машини Windows 10.

Однією з причин вибору Windows 10 як середовища виконання є наявність заводської програми PowerSolarMonitor, що встановлюється лише на операційні системи сімейства Windows. Використання цієї програми під час розробки дозволило паралельно перевіряти правильність зчитування значень та порівнювати результати скрипта з офіційним програмним забезпеченням. Крім того, Windows надає зручний доступ до COM-порту інвертора, що спрощує налагодження процесу зв’язку. Підключення інвертора всередину віртуальної машини здійснюється через прокидання USB-пристрою в Proxmox, як зображено на рисунку 3.1.

Рисунок 3.1 – Прокидання USB-адаптера інвертора Must у Windows 10 VM

Усередині Windows 10 інвертор визначається як COM-порт, що дозволяє скрипту працювати з ним так само, як із звичайним послідовним пристроєм. Приклад відображення COM-порту у диспетчері пристроїв наведено на рисунку 3.2.

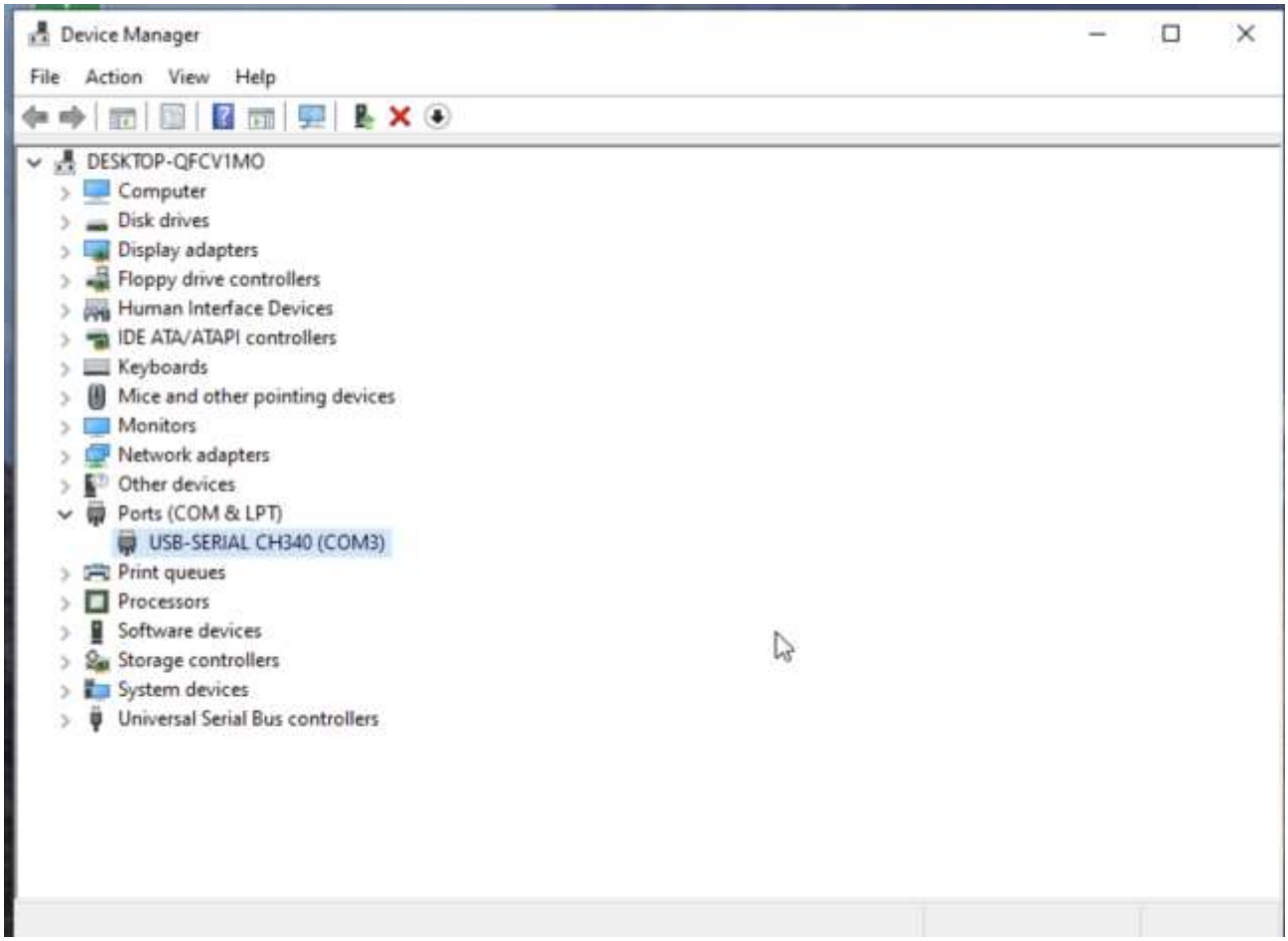

Рисунок 3.2 – COM-порт інвертора Must у Windows VM

Уся логіка зчитування даних реалізована у класі MustInverterDataHandler, який відповідає за формування команд, надсилання їх до інвертора та обробку отриманих відповідей [11-12]. При створенні екземпляра класу задається послідовність команд для отримання різних груп параметрів: напруг, струмів, станів, температур, накопичених енергетичних показників тощо. Для зручності розробки проєкт відкривався у середовищі PyCharm, як зображено на рисунку 3.3.

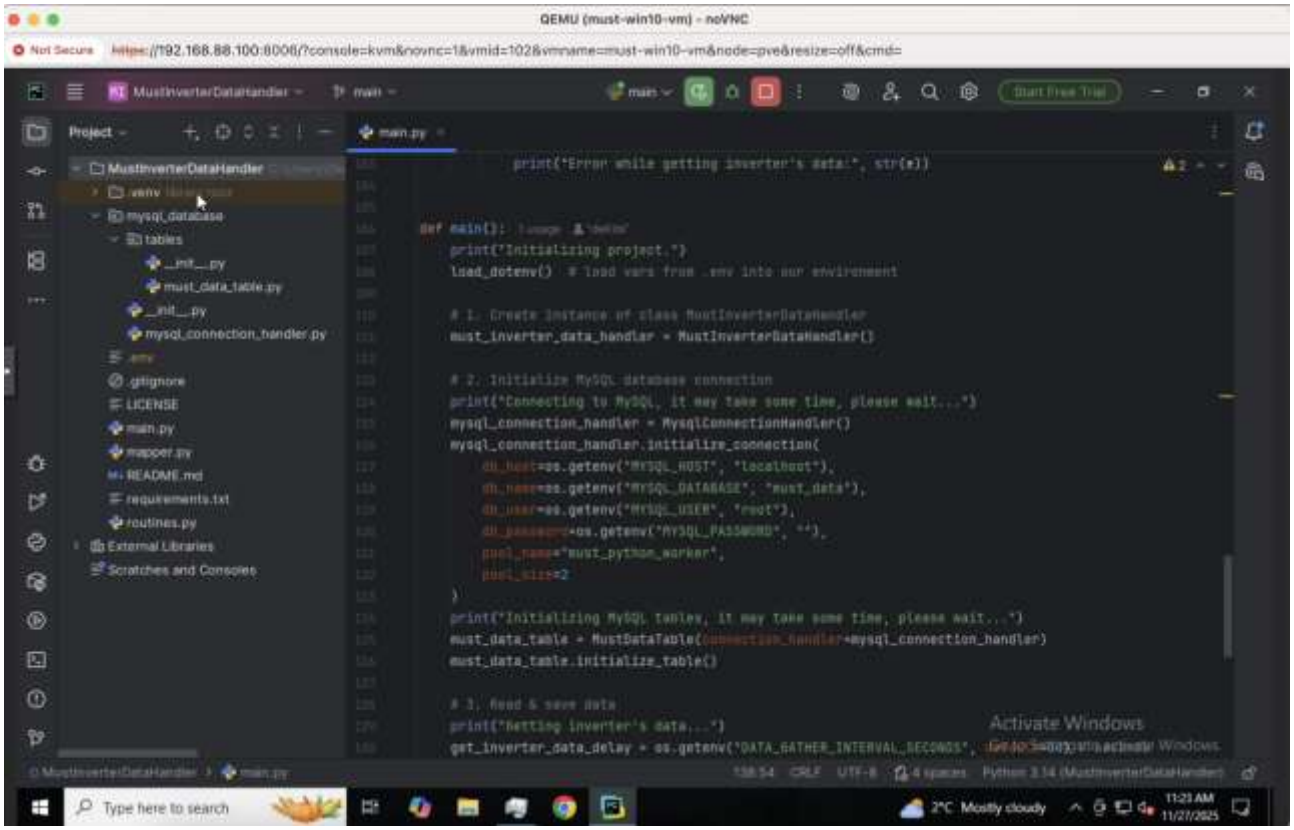

Рисунок 3.3 – Робоча директорія проєкту MustInverterDataHandler у середовищі PyCharm

Основний цикл роботи скрипта є нескінченним: він зчитує дані, записує їх у базу даних та переходить у режим очікування. Така схема дозволяє збирати дані цілодобово, без втручання користувача [24, 33]. Фрагмент коду роботи циклу наведено у лістингу 3.1.

Лістинг 3.1 – Основна частина циклу збору даних

```
while True:
    must_data = must_inverter_data_handler.get_data()
    print("Inverter's data received:", must_data)

    if must_data and len(must_data) > 2:
        must_data_table.insert_data(data=must_data)
        print("Data inserted into the database.")
    else:
        print("Unable to get data from the inverter.")

    time.sleep(get_inverter_data_delay)
```

Функція get_data() виконує основну роботу зі зчитування та обробки інформації. На кожній ітерації формується набір команд, які надсилаються до інвертора через відкритий COM-порт. Отримані відповіді представлені у вигляді масивів байтів, які надалі перетворюються у структурований словник Python. Частину процесу ініціалізації запиту можна побачити у наступному невеликому фрагменті – дивіться лістинг 3.2.

Лістинг 3.2 – Формування команд для отримання даних

```
command_bytes_3 = generate_crc(self._command_string_3)
response_3 = get_part_arr(ser, command_bytes_3, 21, 20)
responses.append(convert_partArr3_to_json(response_3))
```

Після зчитування та парсингу дані записуються у таблицю must_data. Таблиця містить понад 60 полів [24], що дозволяє зберігати практично повну телеметрію інвертора: напруги, струми, потужності, температури, акумуляторні параметри, режими роботи, попередження, накопичену енергію тощо. Для перегляду таблиці під час тестування використовувалася утиліта phpMyAdmin, зображена на рисунку 3.4.

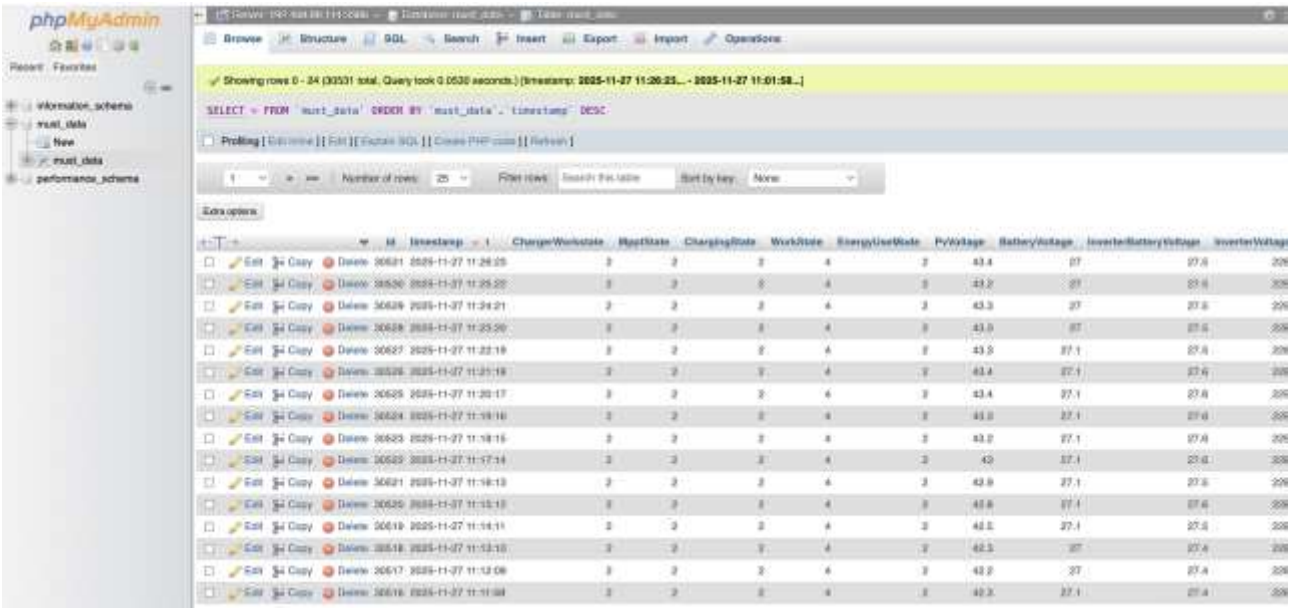

Рисунок 3.4 – Таблиця must_data у phpMyAdmin (перегляд останніх записів)

У процесі роботи скрипт виводить у консоль службові повідомлення, що дозволяє контролювати стабільність підключення та результати виконання команд. Зокрема, при успішному отриманні даних виводиться інформація про структуру телеметрії, а при успішному записі — відповідне підтвердження, як наведено на рисунку 3.5.

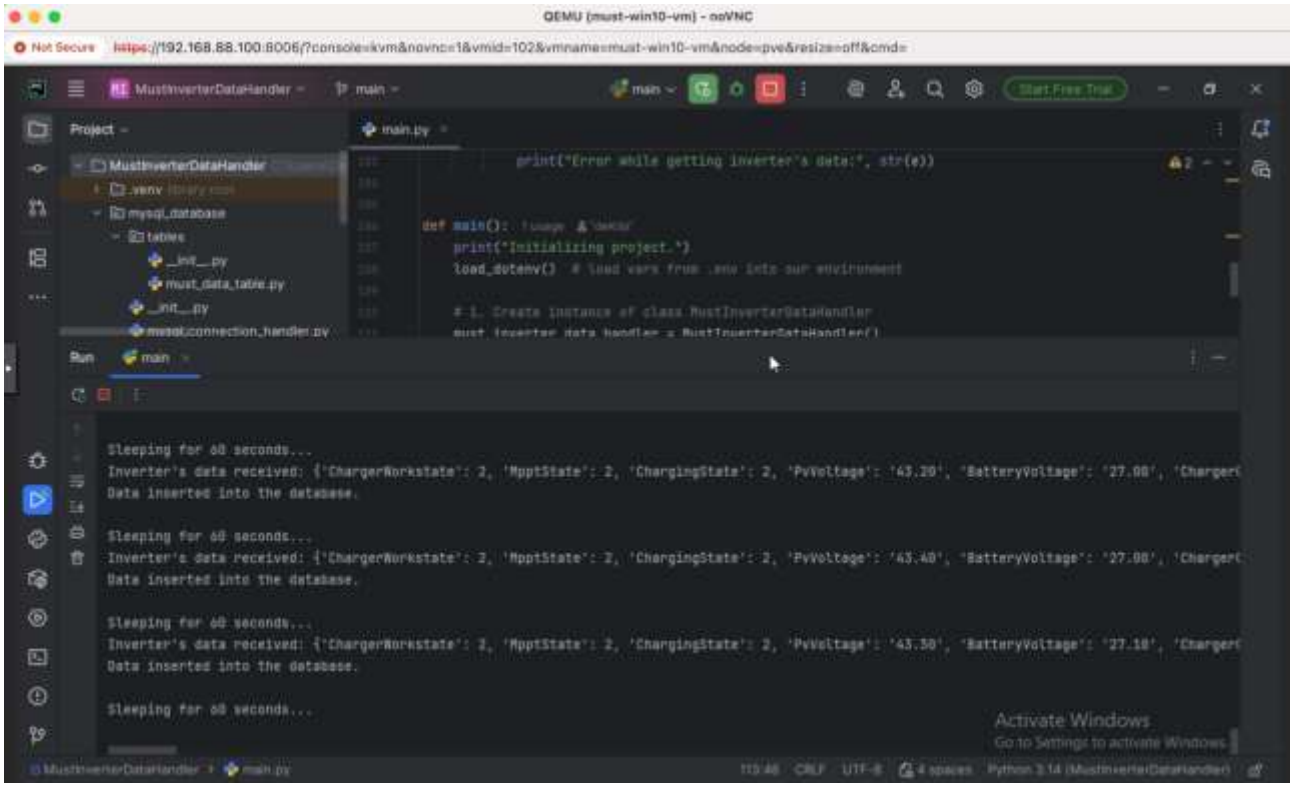

Рисунок 3.5 – Лог роботи скрипта збору даних у Windows 10

Таким чином, серверна частина забезпечує безперервний та стабільний збір даних з інвертора Must, перетворення їх у формат, придатний для зберігання, та передачу у центральну базу даних. Саме ця база даних згодом використовується веб-панеллю для формування поточних показників та історичних графіків, що робить роботу скрипта ключовим елементом у загальній системі моніторингу.

Реалізована модель збору даних є типовою для систем розумного домогосподарства, де сенсорні пристрої повинні передавати інформацію безперервно та незалежно від зовнішнього інтернет-з’єднання [29, 33]. Завдяки

цьому інвертор інтегрується в інфраструктуру розумної квартири як окремий енергетичний модуль.

3.2 Реалізація веб-панелі (Flask-додаток)

Веб-панель є центральним компонентом розробленої системи моніторингу інвертора Must [1, 7]. Саме вона забезпечує доступ користувача до актуальних і історичних даних, дозволяє оцінювати роботу інвертора в реальному часі та переглядати статистику за будь-який період. Веб-панель реалізована у вигляді Flask-додатку, який працює всередині LXC-контейнера на серверній платформі Proxmox та отримує дані безпосередньо з бази даних MySQL.

Файлова структура проєкту складається з кількох основних частин: файлу app.py, каталогу шаблонів HTML (templates), директорії зі стилями та статичними файлами (static), а також конфігураційних файлів середовища. Загальний вигляд структури веб-проєкту наведено на рисунку 3.6.

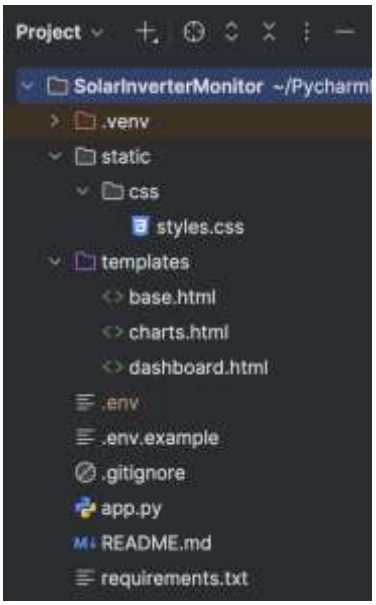

Рисунок 3.6 – Структура Flask-додатку в PyCharm

Основним виконуваним файлом є app.py [1, 9, 13]. У ньому визначено конфігурацію застосунку, параметри підключення до бази даних, маршрути (routes) для сторінок та API, а також функції обробки даних. Також саме тут здійснюється автоматичне відображення таблиці бази даних у вигляді Python-класу через механізм SQLAlchemy automap. Частина коду, що відповідає за ініціалізацію Flask і підключення до MySQL, наведена у лістингу 3.3.

Лістинг 3.3 – Ініціалізація Flask-додатку та підключення до MySQL

```
app = Flask(__name__)
app.config["SQLALCHEMY_DATABASE_URI"] = DB_URL
app.config["SQLALCHEMY_TRACK_MODIFICATIONS"] = False
db = SQLAlchemy(app)

engine = create_engine(DB_URL)
metadata = MetaData()
metadata.reflect(engine, only=[TABLE_NAME])
Base = automap_base(metadata=metadata)
Base.prepare()
Telemetry = Base.classes[TABLE_NAME]
```

Центральною сторінкою веб-панелі є інформаційна панель (/), де відображаються поточні значення всіх ключових параметрів інвертора. Серед них: напруга сонячної панелі, рівень заряду батареї, навантаження, температура радіатора, потужності та стан режиму роботи інвертора. Сторінка інтерактивна й оновлюється під час кожного переходу на неї. Інтерфейс головної панелі зображено на рисунку 3.7.

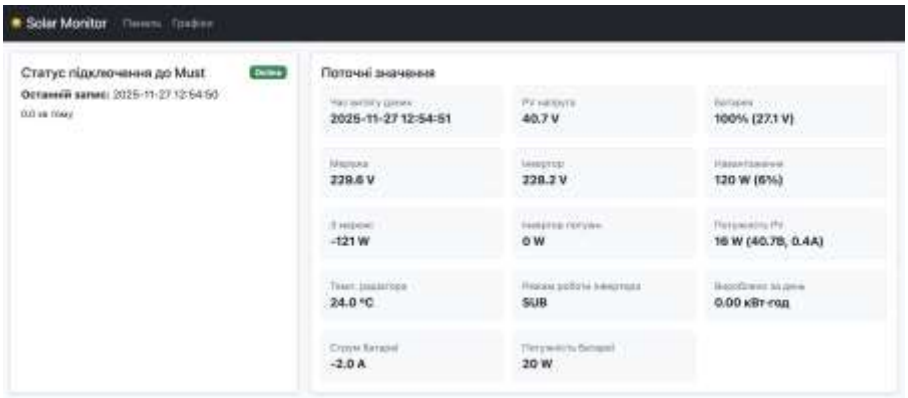

Рисунок 3.7 – Головна сторінка веб-панелі з поточними значеннями інвертора

Важливим елементом панелі є відображення рівня заряду батареї у вигляді відсотків, що обраховується прямо у Flask-додатку на основі напруги. Важливо зазначити, що % заряду батареї переводяться на основі кривої, оскільки напруга батареї і її перевід в % не є лінійним [25]. Значення кривої використані для нашої системи (2 батареї по 100А кожна з'єднані послідовно), функція для обрахунку – battery_voltage_to_percent_curve, фрагмент якої подано нижче – дивіться лістинг 3.4.

Лістинг 3.4 – Функція перетворення напруги на відсоток заряду

```
def battery_voltage_to_percent_curve(v: float) -> float:
    """
    Estimate battery SOC for 24V AGM/GEL based on voltage + load
compensation.
    """

    # Clamp to realistic range
    v = max(23.4, min(25.6, v))

    # Voltage -> % curve (плоский верх)
    curve = {
        25.6: 100,
        25.4: 98,  # було 90 → лагідний спад
        25.2: 95,  # було 80 → реалістичніше для AGM після зарядки
        25.0: 90,  # плавний перехід, не падає різко
        24.9: 70,  # нижче — оригінальні значення
        24.6: 60,
        24.4: 50,
        24.2: 40,
        24.0: 30,
        23.8: 20,
        23.6: 10,
        23.4: 0,
    }

    volts = sorted(curve.keys(), reverse=True)

    for i in range(len(volts) - 1):
        v1, v2 = volts[i], volts[i + 1]
        if v1 >= v >= v2:
            p1, p2 = curve[v1], curve[v2]
            # linear interpolation
            return p1 + (v - v1) * (p2 - p1) / (v2 - v1)

    return curve[volts[-1]]
```

Окремим блоком на головній сторінці реалізовано схему потоків енергії, яка візуально показує напрям роботи системи: від сонячної панелі до акумулятора, інвертора, навантаження та мережі. Вона побудована на основі Flex-компонентів Bootstrap і дозволяє користувачу інтуїтивно зрозуміти, які саме потоки енергії активні зараз. Приклад роботи схеми наведено на рисунку 3.8.

Рисунок 3.8 – Схема потоків енергії у веб-панелі

Другий ключовий розділ веб-панелі — сторінка графіків /charts. Вона дозволяє переглядати історичні дані за будь-який параметр (наприклад, навантаження, напругу батареї, потужність інвертора або сонячної панелі) у вигляді інтерактивного графіка [34]. Користувач може вибрати показник та часовий інтервал, після чого дані автоматично завантажуються через AJAX-запит до API.

Фрагмент коду ендпоінту /api/timeseries [22], який повертає дані для графіка у форматі JSON, наведено у лістингу 3.5.

Лістинг 3.5 – API-ендпоінт для отримання даних графіків

```
@app.route("/api/timeseries")
def api_timeseries():
    metric = request.args.get("metric", "PLoad")
    hours = float(request.args.get("hours", 24))
    if metric not in METRICS:
        return jsonify({"ok": False, "error": "unknown metric"}),
400

    cutoff = datetime.now() - timedelta(hours=hours)
    field = getattr(Telemetry, METRICS[metric])
```

```
    ts_col = getattr(Telemetry, "timestamp")

    # Simple query: last N hours, ordered by time
    with db.session() as s:
        rows = (
            s.query(ts_col, field)
            .filter(ts_col >= cutoff)
            .order_by(ts_col.asc())
            .all()
        )

    # Format for Chart.js time scale
    # UPD: in Unix format
    points = [
        {"x": int(ts.timestamp() * 1000), "y": float(value) if
value is not None else None}
        for ts, value in rows
    ]

    return jsonify({"ok": True, "metric": metric, "points":
points})
```

На стороні клієнта графіки побудовані за допомогою JavaScript-бібліотеки Chart.js, яка дозволяє відображати дані у вигляді плавної лінії, автоматично масштабувати часові ряди та оновлювати графік без перезавантаження сторінки [4, 30]. Веб-інтерфейс сторінки графіків показано на рисунку 3.9.

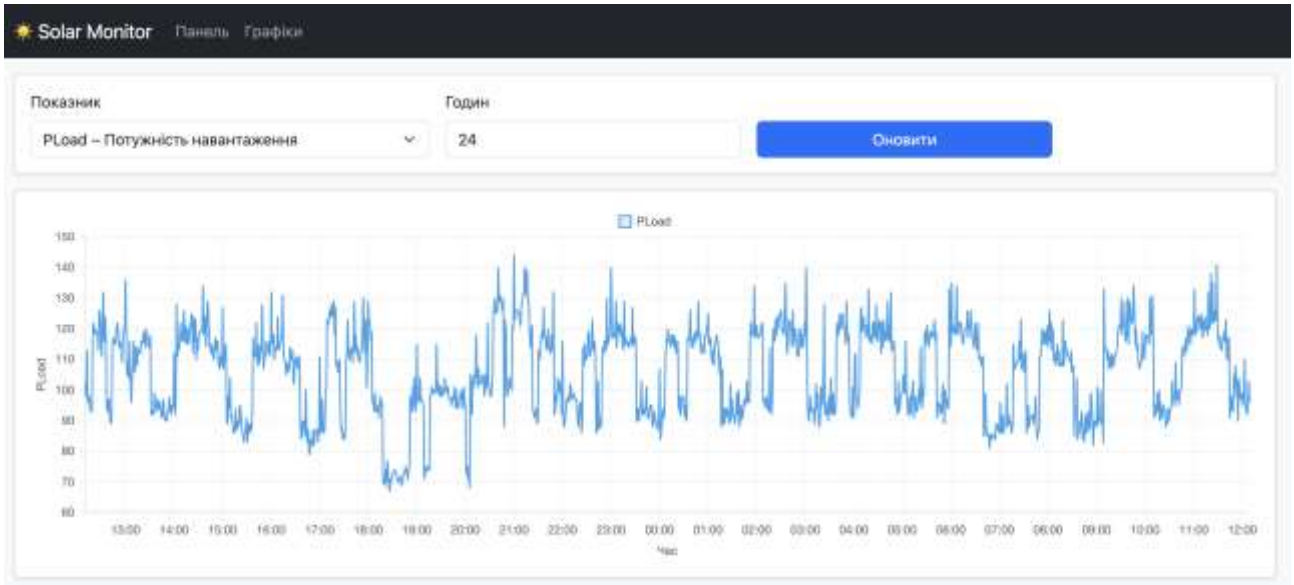

Рисунок 3.9 – Сторінка графіків у веб-панелі

Використання Bootstrap забезпечує адаптивність інтерфейсу, завдяки чому панель автоматично підлаштовується під ширину екрана та коректно відображається як на комп’ютері, так і на смартфоні [5, 17]. Приклад вигляду панелі на мобільному пристрої наведено на рисунку 3.10.

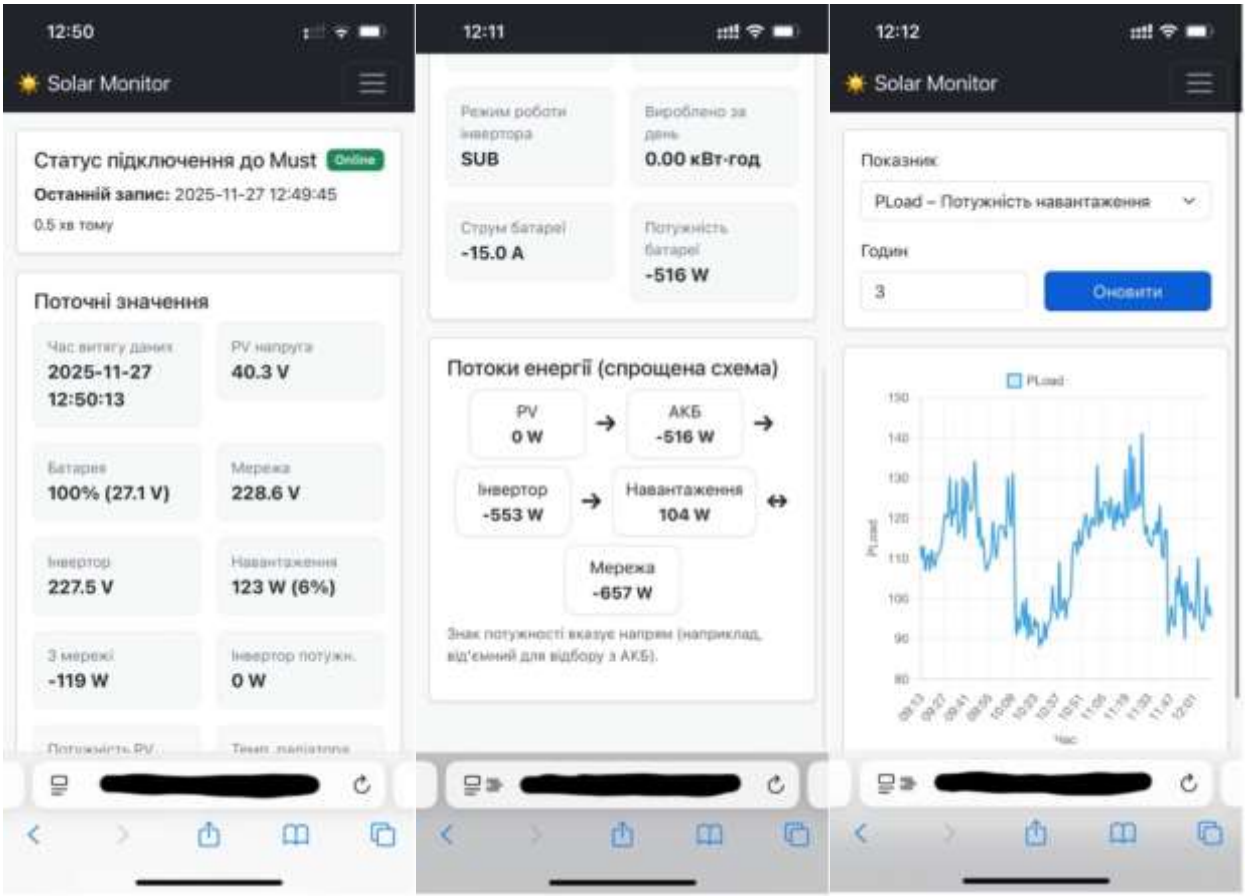

Рисунок 3.10 – Відображення веб-панелі на смартфоні

Усі сторінки побудовані на базі шаблону base.html, що містить основну структуру HTML-документа, загальні стилі, підключення Bootstrap та місце для динамічного контенту. Завдяки використанню Jinja2 кожна сторінка підключає шаблон і вставляє власний вміст лише у відповідні блоки. Невеликий приклад структури шаблону наведено у лістингу 3.6.

Лістинг 3.6 – Фрагмент шаблону base.html

```
<!DOCTYPE html>
<html lang="uk">
<head>
```

```
    <meta charset="UTF-8">
    <title>{% block title %}{% endblock %}</title>
    <link href="/static/css/bootstrap.min.css" rel="stylesheet">
</head>
<body>
<nav class="navbar navbar-light bg-light">
  <a class="navbar-brand" href="/">Solar Monitor</a>
</nav>

<div class="container mt-3">
  {% block content %}{% endblock %}
</div>
</body>
</html>
```

Для розгортання веб-панелі використовується LXC-контейнер у Proxmox, де встановлено Python, Flask, потрібні бібліотеки та веб-сервер. Завдяки контейнеризації веб-панель працює ізольовано від інших сервісів та може бути перезапущена незалежно від бази даних або скрипта для запису [6, 35]. Приклад вікна управління контейнером у Proxmox наведено на рисунку 3.11.

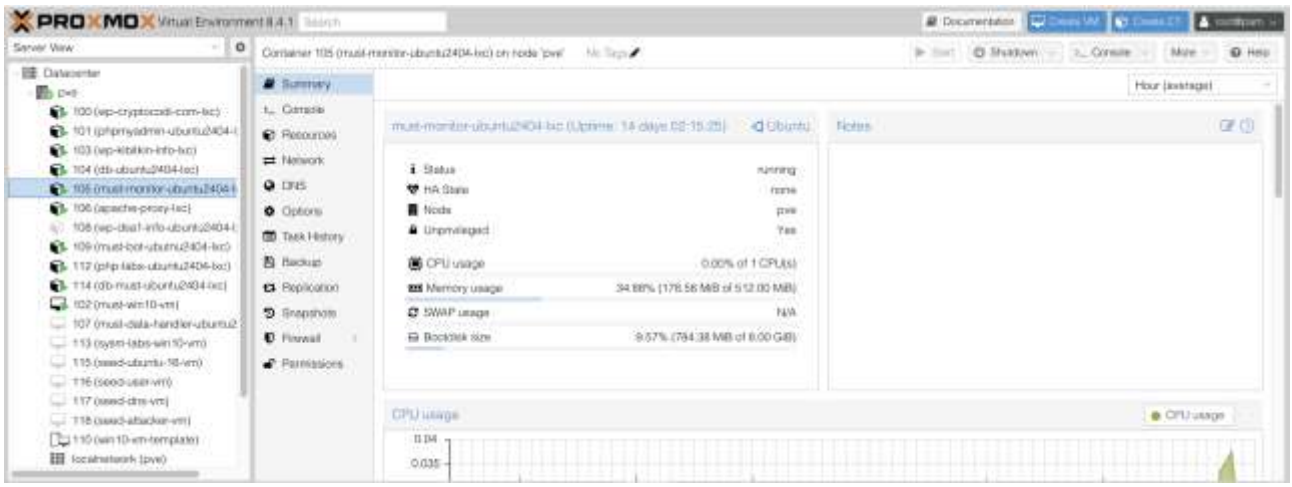

Рисунок 3.11 – Контейнер Proxmox, у якому розгорнуто Flask-додаток

У підсумку веб-панель реалізує повний цикл взаємодії з даними інвертора: отримує їх із бази, обробляє, відображає поточні значення, генерує інформативні графіки та формує зручний інтерфейс для користувача. Завдяки Flask-фреймворку, Chart.js і адаптивному дизайну, система забезпечує швидкий

доступ до інформації та комфортну роботу з будь-якого пристрою. Це робить веб-панель ключовим елементом у всій системі моніторингу.

3.3 Реалізація системи візуалізації даних (графіки та історичні показники)

Однією з ключових можливостей веб-панелі є візуалізація історичних показників роботи інвертора у вигляді інтерактивних графіків [16, 30]. На відміну від статичних фабричних звітів, графіки у веб-панелі формуються динамічно, у реальному часі, та дозволяють користувачу аналізувати будь-який вибраний параметр за довільний часовий проміжок. Завдяки цьому система стає значно інформативнішою та зручнішою для повсякденного моніторингу.

Основний інтерфейс сторінки графіків містить три компоненти:

1. Вибір показника.
2. Вибір часової глибини.
3. Інтерактивне полотно з графіком.

Загальний вигляд інтерфейсу зображено на рисунку 3.12.

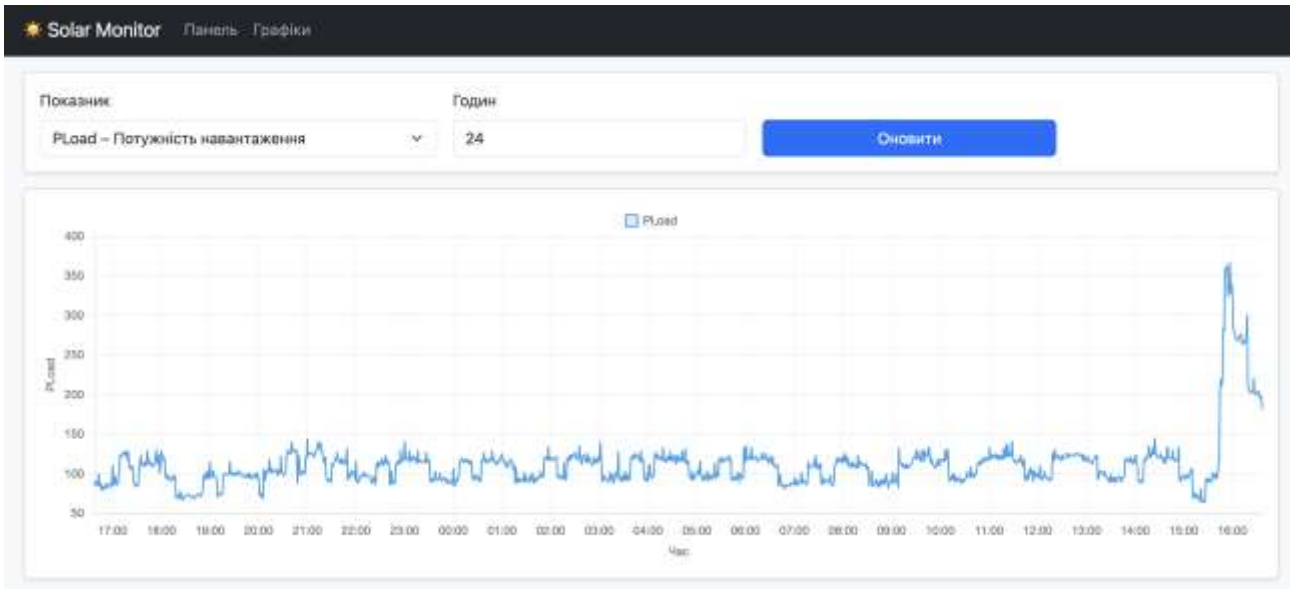

Рисунок 3.12 – Загальний вигляд інтерфейсу графіків

Якщо сторінку відкривати з мобільного пристрою, інтерфейс автоматично переходить у компактний режим та складає меню, що забезпечує зручну навігацію на невеликих екранах, як зображено на рисунку 3.13.

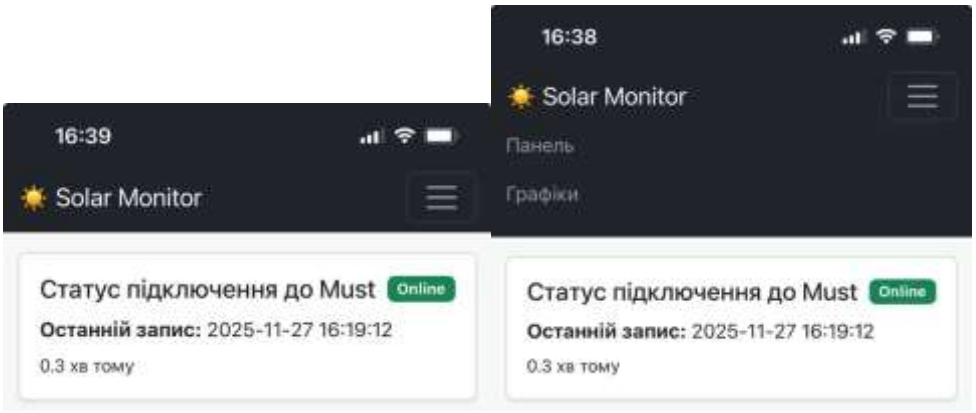

Рисунок 3.13 – Інтерфейс меню для переходу в сторінку графіків на мобільному пристрої

Обробка даних для графіка здійснюється через AJAX-запит до API-ендпоінту /api/timeseries. При кожній зміні параметра або часової глибини браузер надсилає запит до сервера, отримує масив координат формату x–y та будує графік без перезавантаження сторінки. Завдяки такому підходу користувач може швидко перемикатись між різними показниками та бачити зміни майже моментально.

Графік будується за допомогою бібліотеки Chart.js [4], яка забезпечує плавність ліній, автоматичне масштабування, коректне відображення часової осі та адаптацію під будь-який розмір екрана. На рисунку 3.14 наведено приклад графіка навантаження інвертора за останні 24 години.

Подібним чином можуть бути проаналізовані інші показники. Наприклад, графік напруги батареї дозволяє оцінити, як змінюється рівень заряду протягом доби [25, 30].

Рисунок 3.14 – Графік зміни навантаження (PLoad) за останні 24 години

Завдяки цьому користувач може зрозуміти, у які періоди батарея заряджається від панелі, а коли переходить на режим віддачі енергії. Приклад такого графіка наведено на рисунку 3.15.

Рисунок 3.15 – Графік зміни напруги батареї протягом доби

Окремий інтерес становлять графіки, пов’язані з роботою сонячної панелі. Показники PV-напроги та потужності дозволяють чітко відслідковувати продуктивність панелі упродовж дня. Продуктивність панелі упродовж дня. Зокрема, графік потужності PV демонструє розподіл сонячного виробітку та дозволяє оцінити ефективність панелі в ранкові, денні та вечірні години, як зображено на рисунку 3.16.

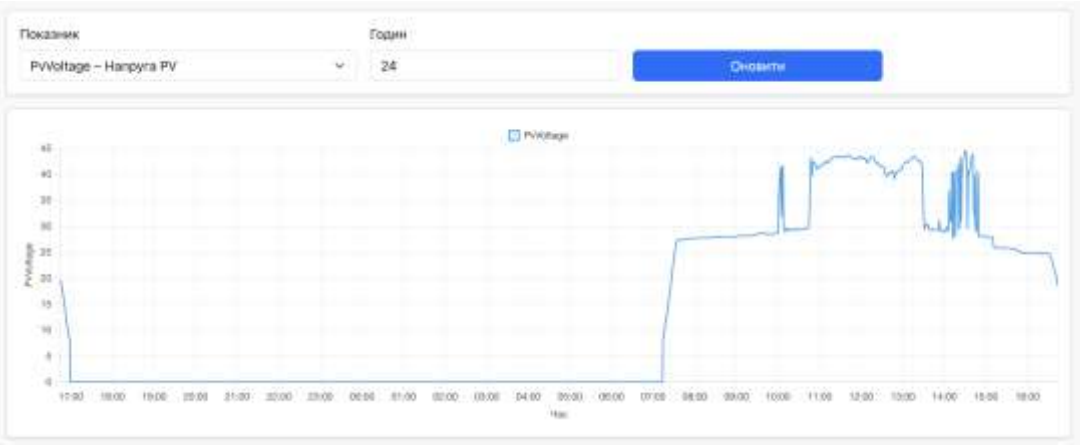

Рисунок 3.16 – Графік потужності сонячної панелі за світловий день

Користувачу також доступний перегляд температурних показників. Наприклад, графік температури радіатора дозволяє визначити, як інвертор поводиться під різним навантаженням та чи не перегрівається він в моменти пікової роботи [21, 26]. Інверсія температури ввечері та їхнє поступове зниження можуть свідчити про перехід інвертора в менш енергоємні режими. Приклад наведено на рисунку 3.17.

Рисунок 3.17 – Температура радіатора інвертора за 12 годин роботи

Механізм побудови графіка реалізовано у браузері через JavaScript-функцію, яка звертається до серверної частини. Код, відповідаючий за оновлення графіка, працює наступним чином:

1. Отримує значення параметрів.

2. Виконує запит до API.
3. Передає масив точок у Chart.js.
4. Знищує попередній графік і створює новий.

Невеликий фрагмент логіки наведено у лістингу 3.7.

Лістинг 3.7 – Фрагмент логіки побудови графіка на клієнтській стороні

```
const res = await fetch(`/api/timeseries?metric=${metric}&hours=${hours}`);
const json = await res.json();
chartInstance = new Chart(ctx, {
  type: 'line',
  data: { datasets: [{ data: json.points }] },
  options: { responsive: true, parsing: true }
});
```

Завдяки адаптивній верстці Bootstrap і можливостям Chart.js графіки коректно відображаються на мобільних пристроях [4, 5, 30]. Це дозволяє користувачу переглядати стан інвертора та аналізувати його роботу навіть зі смартфона. Приклад такого відображення наведено на рисунку 3.18.

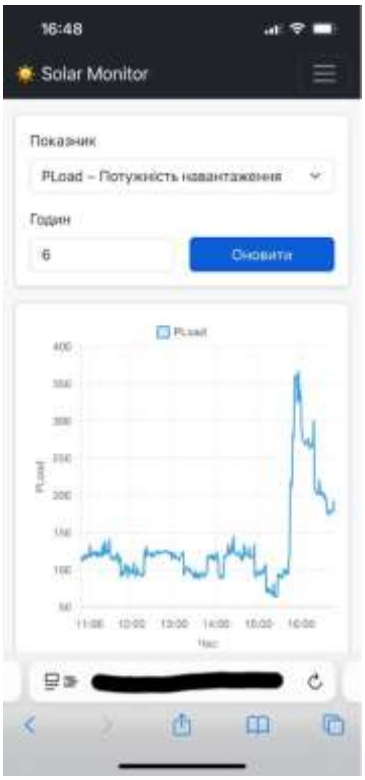

Рисунок 3.18 – Графік навантаження, відкритий на смартфоні

Таким чином, підсистема візуалізації історичних даних дозволяє отримувати важливу інформацію про роботу інвертора Must у зручному та наочному вигляді. Побудова графіків у реальному часі, гнучкий вибір параметрів та можливість переглядати дані з різних пристроїв роблять веб-панель корисним інструментом як для користувача, так і для подальшого аналізу енергетичної системи.

3.4 Реалізація підсистеми сповіщень за допомогою Telegram-бота

У рамках розроблюваної системи моніторингу енергосистеми розумної квартири було реалізовано додаткову підсистему сповіщень на основі Telegram-бота [32] – дивіться рисунок 3.19.

Рисунок 3.19 – Створений Telegram-бот для підсистеми сповіщень

Дана підсистема не є частиною веб-панелі, а функціонує як окремий програмний модуль, що використовує вже наявні дані, які зберігаються у базі даних системи.

Оскільки в розробленого Telegram-бота є доступ на перегляд бази даних, він може отримувати дані з бази даних і наводити їх для довідки, дублюючи частковий функціонал панелі і виступаючи в ролі резервного варіанту, а також

демонструючи користувачеві поточні індивідуальні налаштування бота – дивіться рисунок 3.20.

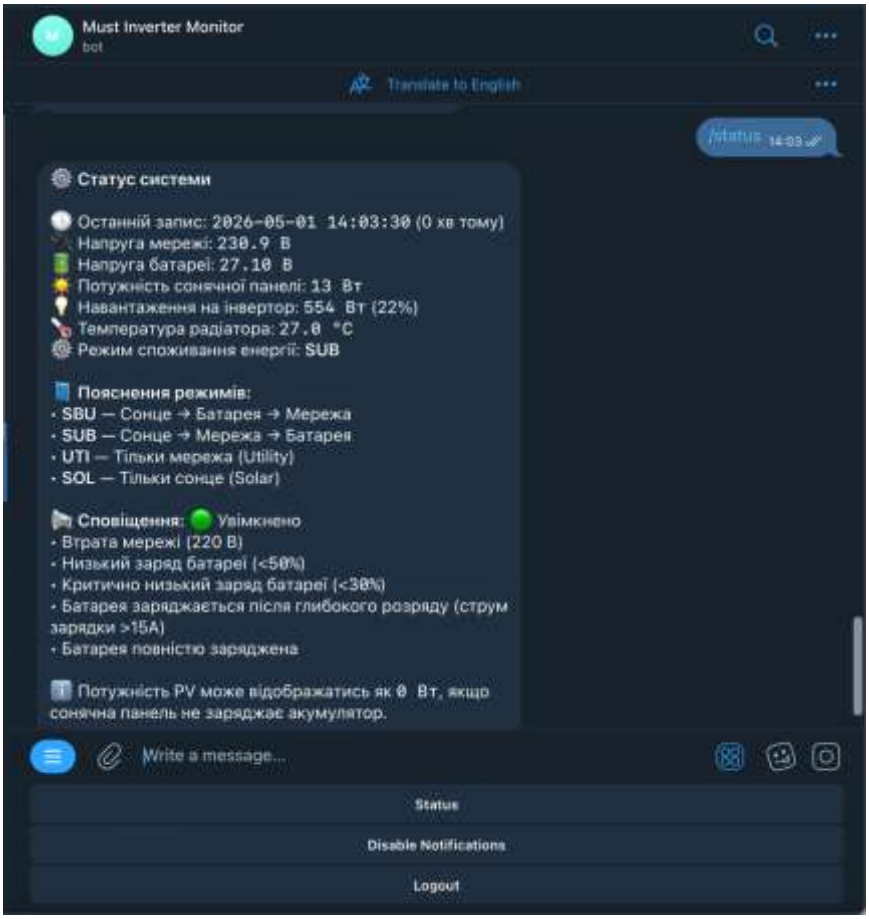

Рисунок 3.20 – Статус системи та поточні налаштування

Основною метою створення Telegram-бота є забезпечення оперативного інформування користувача про критичні події, пов’язані зі станом енергосистеми, зокрема [32, 36]:

– відсутність електропостачання;
– відновлення електропостачання;
– втрату зв’язку з інвертором;
– низький або критичний рівень заряду батареї;
– початок та завершення процесу заряджання акумулятора.

Таким чином, на відміну від класичних систем моніторингу, що передбачають лише перегляд інформації у веб-інтерфейсі, запропоноване рішення дозволяє реалізувати проактивну модель взаємодії з користувачем [32].

Підсистема сповіщень реалізована як незалежний сервіс, який періодично звертається до бази даних, отримує актуальні показники інвертора та аналізує їх на предмет виникнення подій.

Важливою особливістю є те, що Telegram-бот:

– не здійснює збір даних самостійно;
– не залежить від веб-панелі;
– використовує вже існуючу базу даних як джерело інформації.

Це дозволяє використовувати підсистему як окремий компонент, а також спрощує масштабування системи в майбутньому.

Для реалізації Telegram-бота було використано бібліотеку python-telegram-bot, яка дозволяє організувати обробку команд та повідомлень у асинхронному режимі.

Бот підтримує базові команди:

– /start – авторизація користувача;
– /status – отримання поточного стану системи;
– /enable / disable – ввімкнути / вимкнути сповіщення;
– /logout – вихід із системи.

Рисунок 3.21 – Список доступних команд при вводі “/”

Команди, також підтримують “tab-complete” – тобто авто доповнення команд, коли користувач починає вводити відповідну команду – дивіться рисунок 3.21.

Крім того, реалізовано простий графічний інтерфейс у вигляді клавіатури з кнопками [17, 32] – дивіться рисунок 3.22.

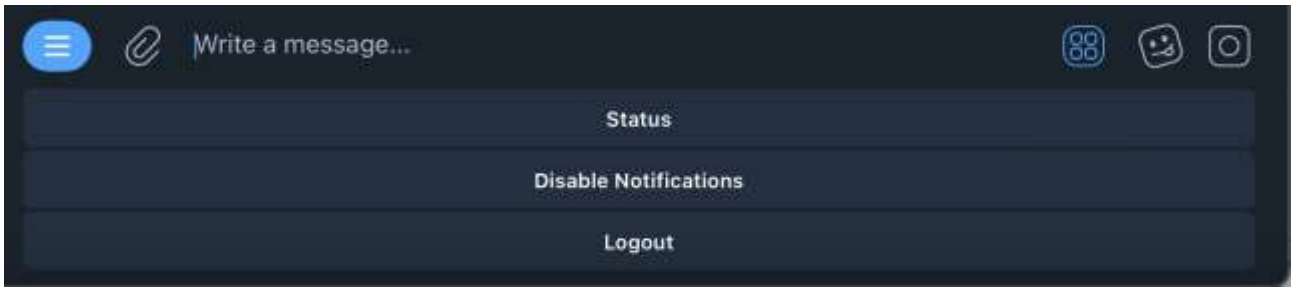

Рисунок 3.22 – Реалізація меню з кнопками в Telegram-боті

Приклад формування меню користувача наведено у лістингу 3.1.

Лістинг 3.8 – Формування меню користувача

```
keyboard = [["Status"]]
if notif_enabled:
    keyboard.append(["Disable Notifications"])
else:
    keyboard.append(["Enable Notifications"])
keyboard.append(["Logout"])

reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
```

Такий підхід дозволяє спростити взаємодію користувача з системою та зробити її інтуїтивно зрозумілою.

Бот отримує дані безпосередньо з MySQL бази даних, використовуючи окремі функції для виконання запитів [3, 32]. Наприклад, отримання останнього запису реалізовано у вигляді запиту до таблиці даних інвертора:

cur.execute(f"SELECT * FROM {DB_DATA_TABLE} ORDER BY id DESC LIMIT 1;")

Це дозволяє отримувати актуальну інформацію без дублювання логіки збору даних.

Ключовою частиною підсистеми є асинхронний цикл моніторингу, який виконується з певним інтервалом та аналізує стан системи. Сповіщення надсилаються всім авторизованим користувачам за допомогою окремої функції, приклад якої наведено у лістингу 3.9.

Лістинг 3.9 – Відправка сповіщень користувачам

```
async def notify_all(text):
    for uid in recipients:
        await bot.send_message(chat_id=uid, text=text)
```

Цей підхід дозволяє централізовано керувати розсилкою повідомлень.

Однією з ключових функцій підсистеми є автоматичне визначення факту відсутності електропостачання та розрахунок тривалості відключення.

Визначення відсутності мережі здійснюється шляхом перевірки значення напруги:

grid_voltage = float(row.get("GridVoltage", 0))

if grid_voltage == 0:

Після цього система фіксує момент події та, при відновленні електропостачання, обчислює тривалість відключення та надсилає відповідне повідомлення користувачу.

Рисунок 3.23 – Механізм авторизації в Telegram-боті

Такий підхід дозволяє не лише інформувати про факт події, але й надавати додаткову аналітичну інформацію.

Для обмеження доступу до функціоналу бота реалізовано механізм авторизації користувачів за допомогою пароля – дивіться рисунок 3.23.

Дані авторизованих користувачів зберігаються у окремій таблиці бази даних.

Користувачі можуть:

– отримати доступ до системи після введення пароля;

– вмикати або вимикати сповіщення (для себе);

– виходити з системи.

Це забезпечує базовий рівень захисту та контроль доступу до інформації [27].

У процесі тестування було перевірено коректність роботи підсистеми у різних сценаріях. Наприклад, повідомлення про відсутність електроенергії – дивіться рисунок 3.24.

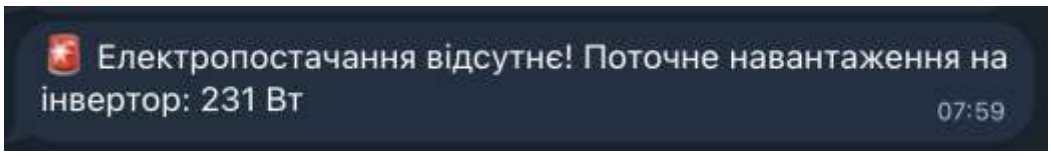

Рисунок 3.24 – Повідомлення в Telegram-боті про відсутність електромережі

Також, варто зазначити, що в сповіщення було додане поточне навантаження на інвертор, що допомагає контролювати навантаження під час вимкнення електромережі і знизити його тільки до базових споживачів, за потреби.

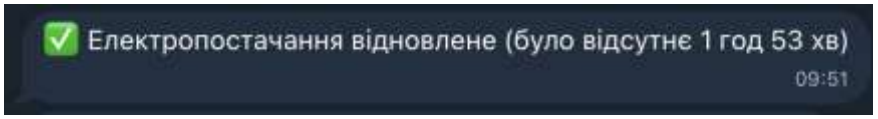

Рисунок 3.25 – Повідомлення про відновлення електропостачання та тривалість вимкнення

Коли електропостачання відновлене – бот також надсилає сповіщення, додаючи інформацію про тривалість відключення – дивіться рисунок 3.25.

Система коректно визначає зміну стану та надсилає повідомлення користувачу у режимі реального часу [32, 36].

3.5 Розгортання системи на базі Proxmox VE

Для забезпечення стабільної роботи системи моніторингу енергосистеми розумної квартири було виконано розгортання програмного забезпечення на локальному сервері з використанням платформи віртуалізації Proxmox Virtual Environment (Proxmox VE) версії 8.4 – дивіться рисунок 3.26.

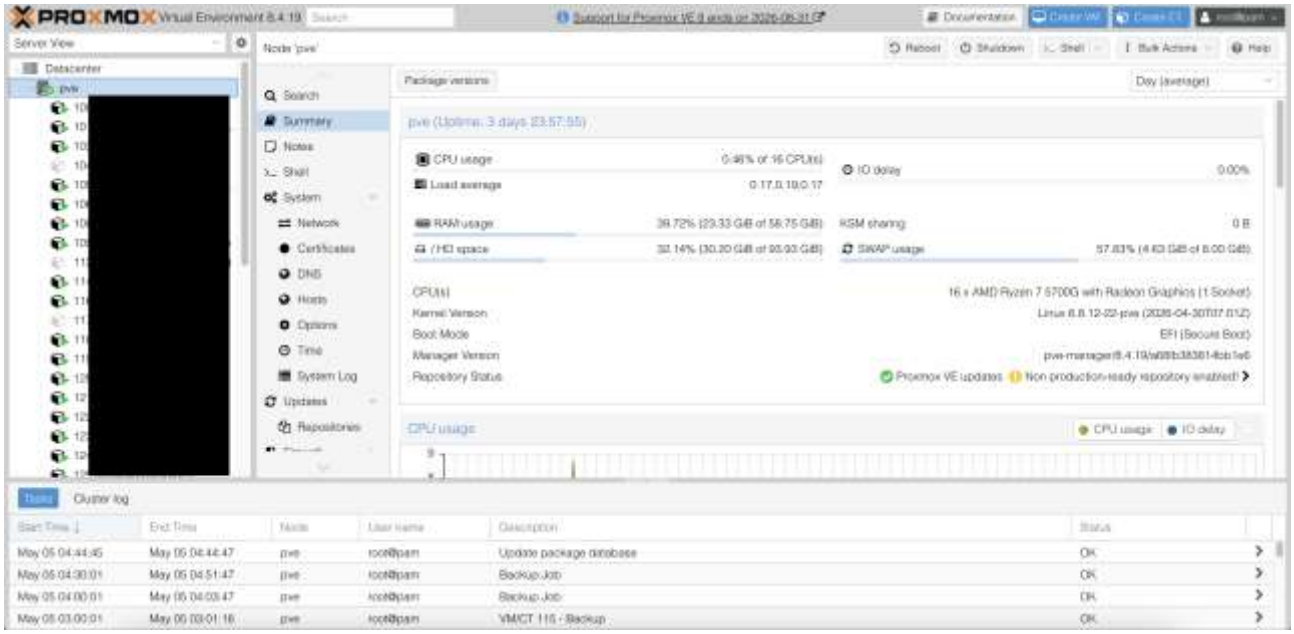

Рисунок 3.26 – Веб-панель керування гіпервізором 1-го типу Proxmox VE 8.4

Використання Proxmox VE дозволило реалізувати ізольоване середовище для кожного компонента системи, підвищити відмовостійкість, а також забезпечити зручність адміністрування та масштабування [35].

Система розгорнута на локальному сервері та складається з декількох ізольованих середовищ (віртуальних машин та контейнерів), кожне з яких виконує окрему функцію – дивіться рисунок 3.27.

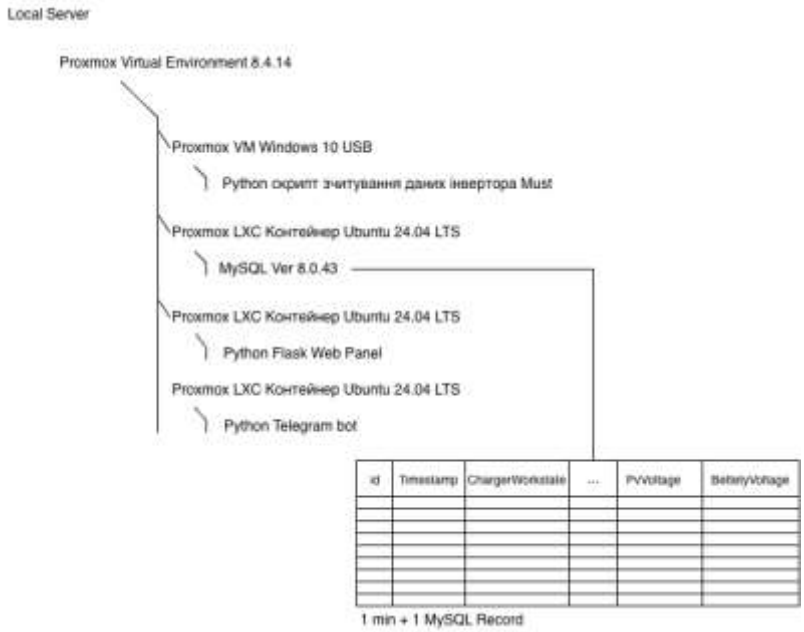

Рисунок 3.27 – Архітектура розгортання системи моніторингу

Як видно з рисунку 3.27, система складається з наступних компонентів:

– must-win10-vm (VM) – віртуальна машина з Windows 10, яка виконує Python-скрипт збору даних з інвертора через USB-підключення – дивіться рисунок 3.28;

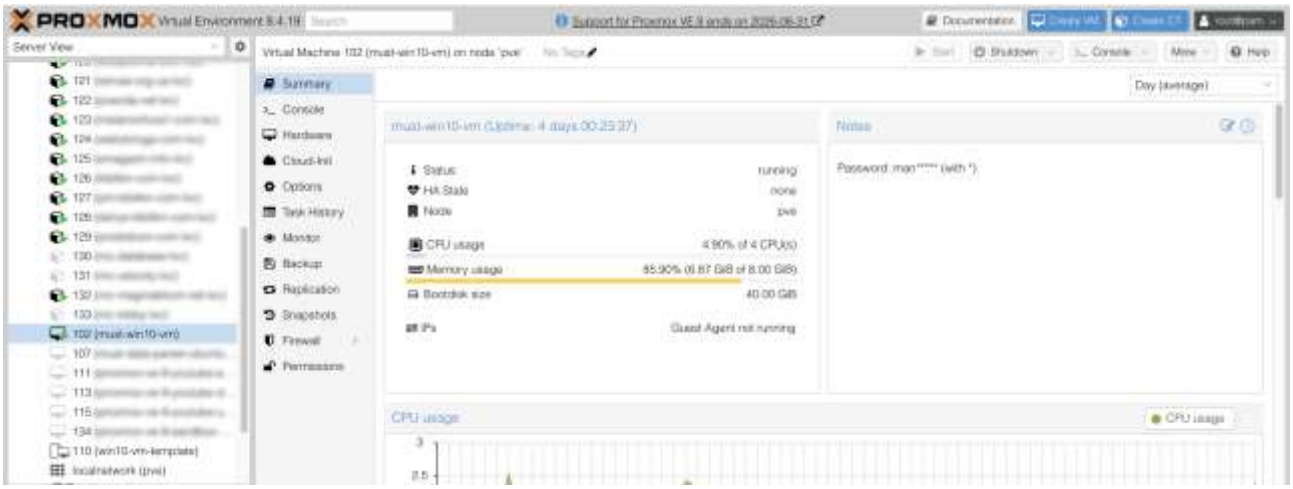

Рисунок 3.28 – Віртуальна машина Windows 10

– must-db-ubuntu2404-lxc – контейнер з СУБД MySQL, що використовується для зберігання даних – дивіться рисунок 3.29;

Рисунок 3.29 – Контейнер з СУБД MySQL

– must-monitor-ubuntu2404-lxc – контейнер з Flask-додатком веб-панелі [6, 35] – дивіться рисунок 3.30;

Рисунок 3.30 – Контейнер для веб-панелі Flask

– must-bot-ubuntu2404-lxc – контейнер з Telegram-ботом для сповіщень [32] – дивіться рисунок 3.31.

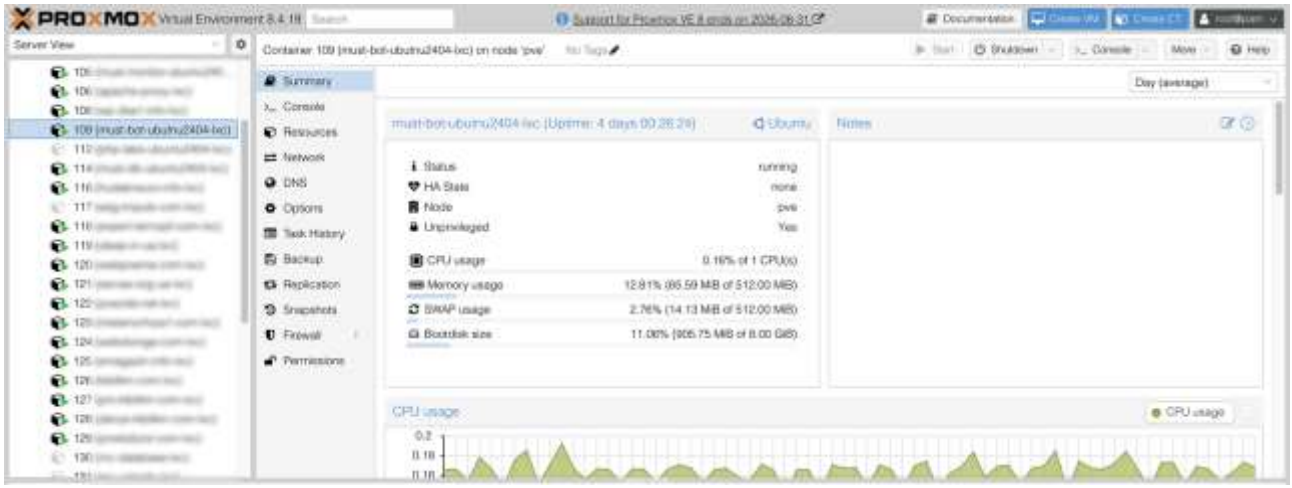

Рисунок 3.31 – Контейнер з Telegram-ботом

Такий підхід дозволяє розділити систему на незалежні модулі, що значно спрощує її підтримку та модернізацію.

Використання контейнерів LXC для більшості компонентів обумовлено їх низьким споживанням ресурсів та швидким запуском, що є важливим для постійно працюючої системи моніторингу [35].

Віртуальна машина з Windows використовується окремо, оскільки:

– взаємодія з інвертором реалізована через USB;

– драйвери та програмне забезпечення інвертора орієнтовані на середовище Windows.

Таким чином, комбіноване використання VM та LXC дозволило отримати оптимальний баланс між продуктивністю та сумісністю [35].

Для забезпечення стабільного доступу до компонентів системи було використано механізм статичного призначення IP-адрес (DHCP reservation) на серверному маршрутизаторі TP-Link ER605 – дивіться рисунок 3.32.

Кожному контейнеру та віртуальній машині призначено фіксовану IP-адресу на основі MAC-адреси, що дозволяє:

– уникнути зміни адрес після перезавантаження;

– забезпечити стабільну роботу сервісів;

– спростити налаштування доступу та інтеграцію між компонентами.

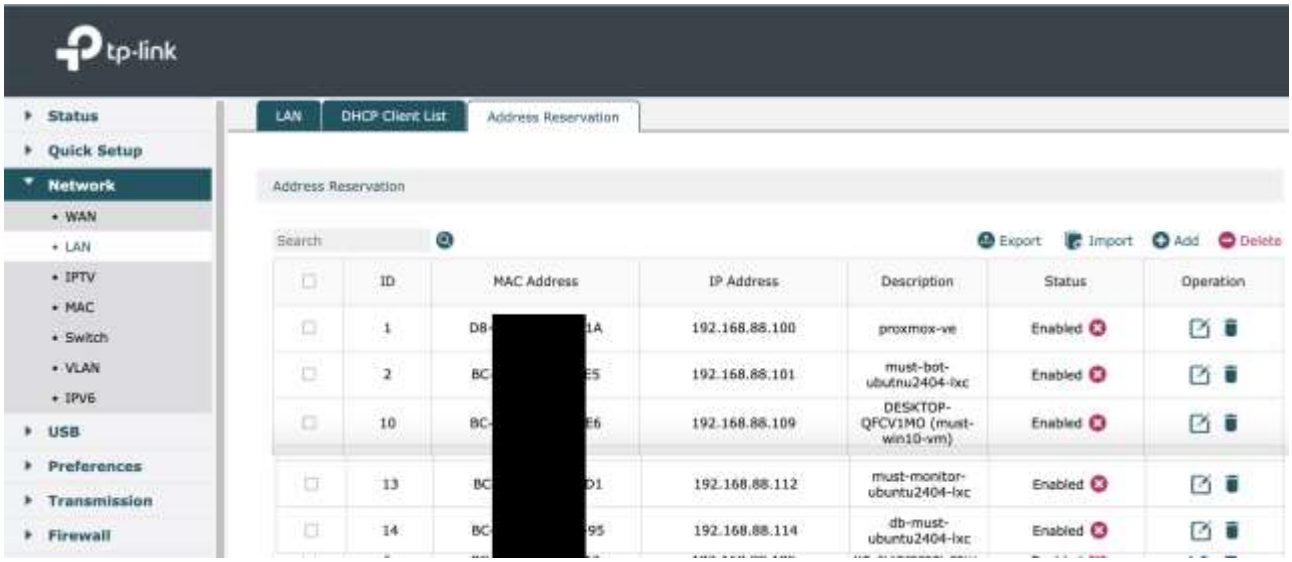

Рисунок 3.32 – Закріплення IP-адрес за компонентами системи

Для доступу до веб-панелі моніторингу було налаштовано прокидання портів (NAT) на маршрутизаторі – дивіться рисунок 3.33.

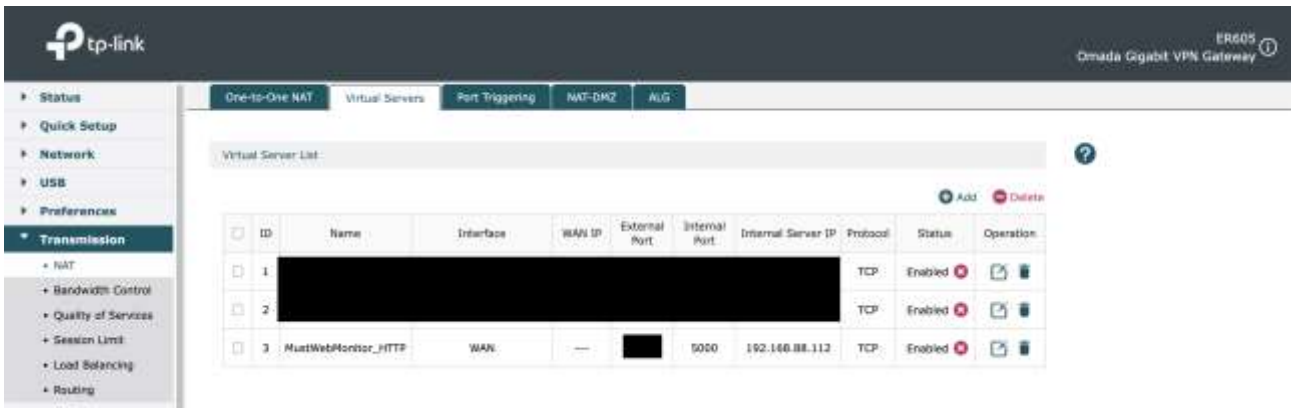

Рисунок 3.33 – Налаштування доступу до веб-панелі через NAT

Зовнішній порт маршрутизатора перенаправляється на внутрішній порт контейнера з веб-панеллю, що дозволяє отримати доступ до системи з локальної мережі або, за потреби, ззовні.

Важливо зазначити, що у даному проєкті використовується прямий доступ без застосування зворотного проксі-сервера, оскільки система не є публічним веб-сервісом, а використовується у межах приватної інфраструктури.

Для підвищення надійності системи було налаштовано регулярне резервне копіювання віртуальних машин та контейнерів засобами Proxmox VE [6] – дивіться рисунок 3.34.

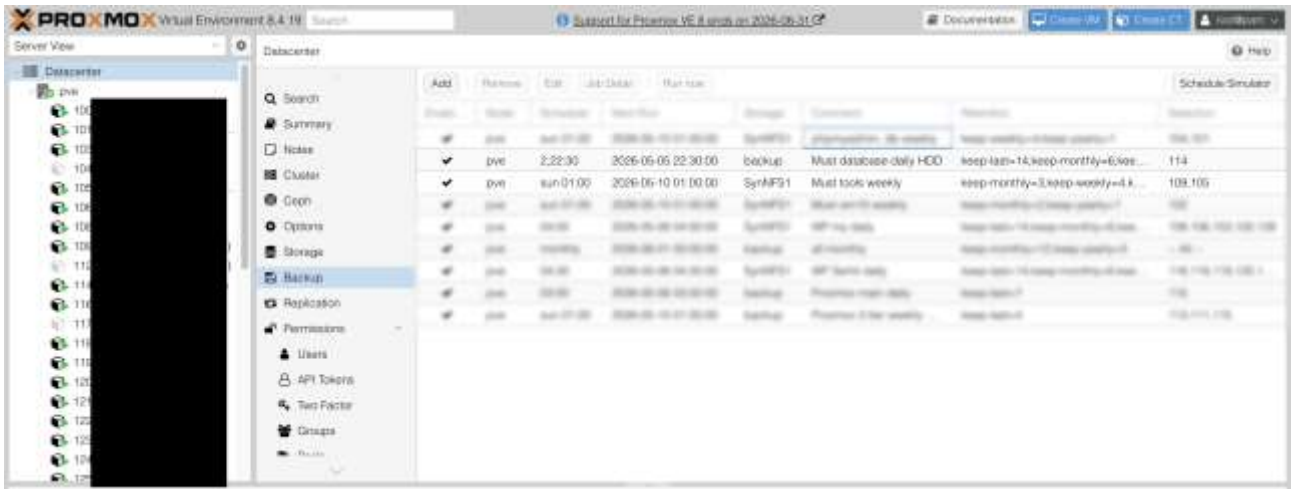

Рисунок 3.34 – Налаштування резервного копіювання в Proxmox VE

Резервні копії виконуються за розкладом та зберігаються на окремих носіях:

– локальний диск сервера;

– мережеве сховище (Synology NAS).

Такий підхід дозволяє:

– швидко відновити систему у разі збою;

– забезпечити збереження історичних даних;

– мінімізувати ризик втрати інформації.

Реалізована інфраструктура має наступні переваги:

– модульність – кожен компонент працює незалежно;

– відмовостійкість – збій одного сервісу не впливає на інші;

– масштабованість – можливість додавання нових компонентів;

– зручність адміністрування через Proxmox VE;

– можливість централізованого резервного копіювання.

У результаті було реалізовано розгортання системи моніторингу на базі Proxmox VE із використанням контейнеризації та віртуалізації.

Запропонований підхід забезпечує стабільну роботу системи, гнучкість у розширенні функціоналу та зручність обслуговування, що є важливими характеристиками для систем моніторингу, які працюють у режимі 24/7 [6, 35].

3.6 Тестування та перевірка працездатності системи

Після реалізації системи моніторингу було проведено тестування її працездатності з метою перевірки коректності роботи веб-панелі та підсистеми сповіщень у різних середовищах.

З метою перевірки кросбраузерності та кросплатформеності веб-панелі було виконано тестування на різних пристроях та у різних браузерах.

Тестування на персональному комп’ютері у браузері Google Chrome показало коректне відображення інтерфейсу, графіків та елементів керування – дивіться рисунок 3.35.

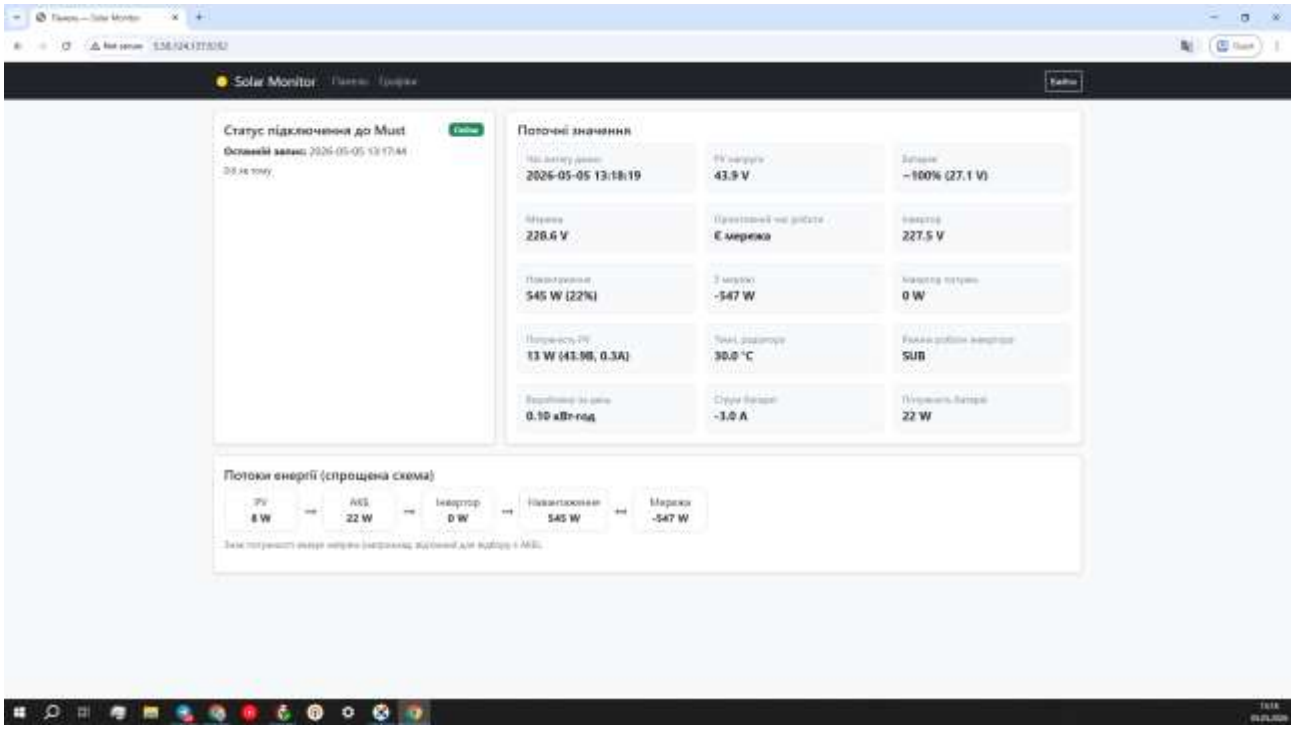

Рисунок 3.35 – Веб-панель у Chrome на Windows

Тестування у браузері Mozilla Firefox також підтвердило стабільну роботу системи без спотворення інтерфейсу [17] – дивіться рисунок 3.36.

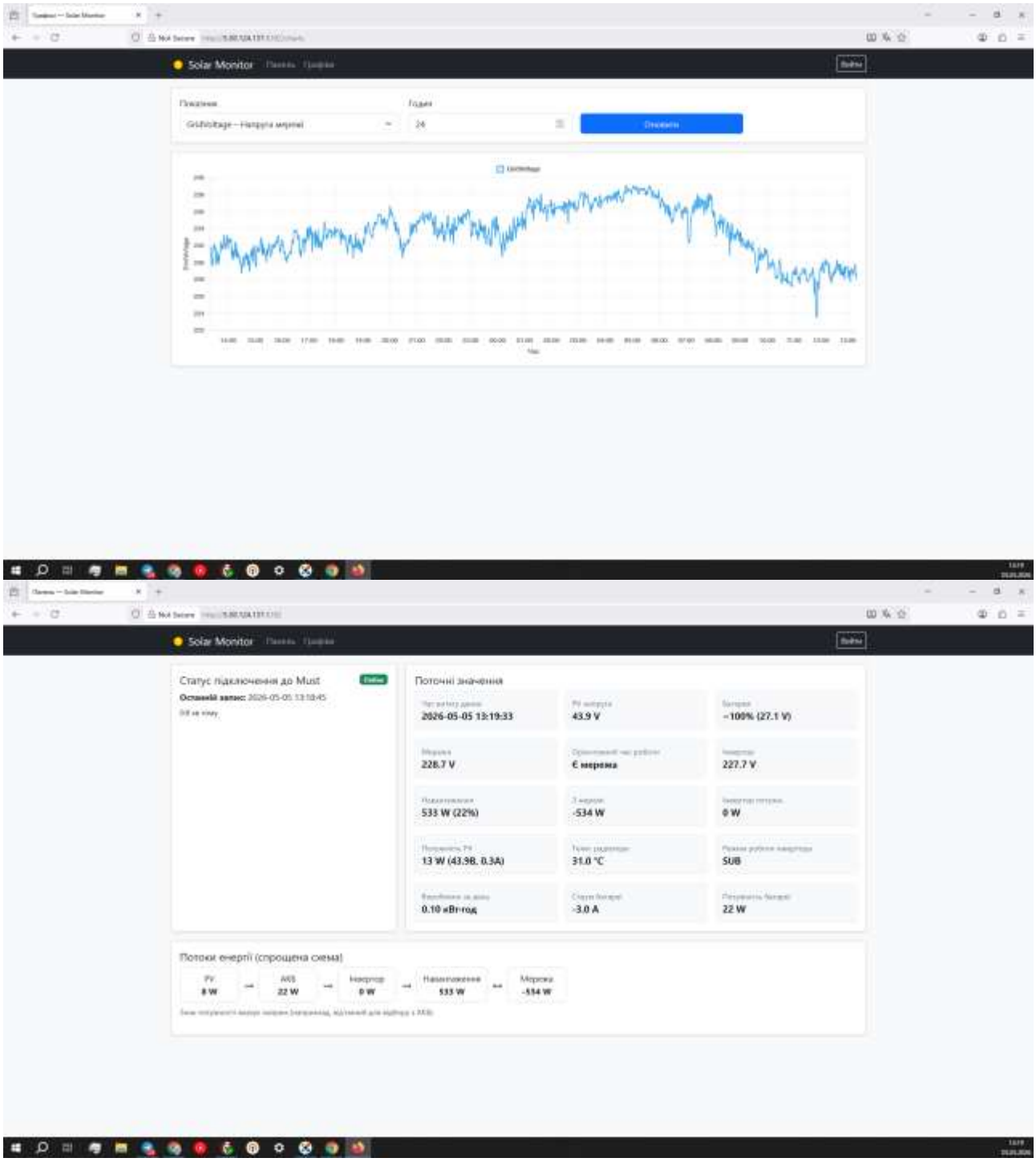

Рисунок 3.36 – Веб-панель у Firefox на Windows

Також, панель було протестовано в браузері за замовчуванням – Internet Explorer, який попередньо встановлений на Windows [17] – дивіться рисунок 3.37.

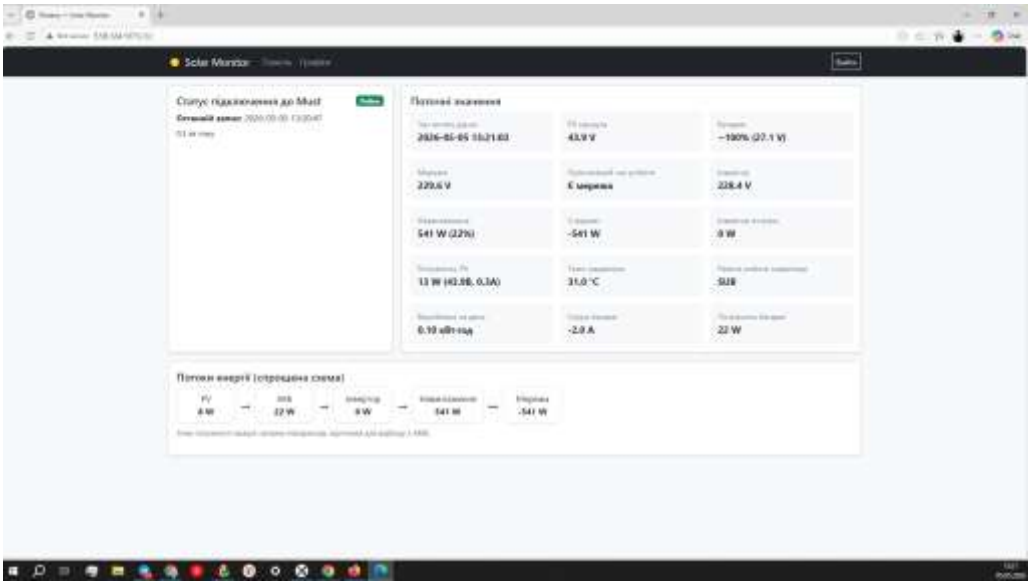

Рисунок 3.37 – Веб-панель у Internet Explorer на Windows

Додатково було змінено час для графіка, щоб переконатись що не тільки відображення працює коректно, а також оновлення даних та відображення нових [4, 30]. Тестування не виявило ніяких відхилень – всі елементи відображається коректно – дивіться рисунок 3.38.

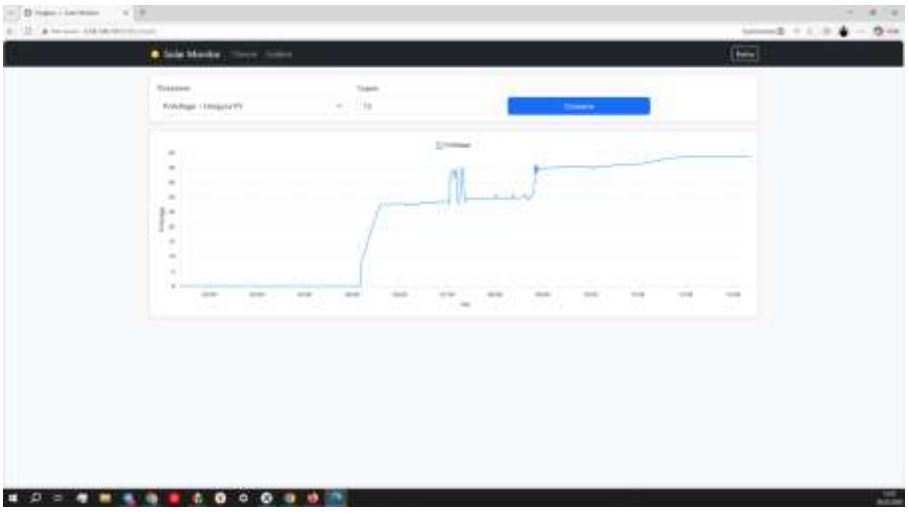

Рисунок 3.38 – Відображення графіка зі зміненим часовим показником в Internet Explorer на Windows

Тестування на ноутбуці під управлінням macOS у браузері Safari показало коректне відображення графіків та адаптацію інтерфейсу [4, 5] – дивіться рисунок 3.39.

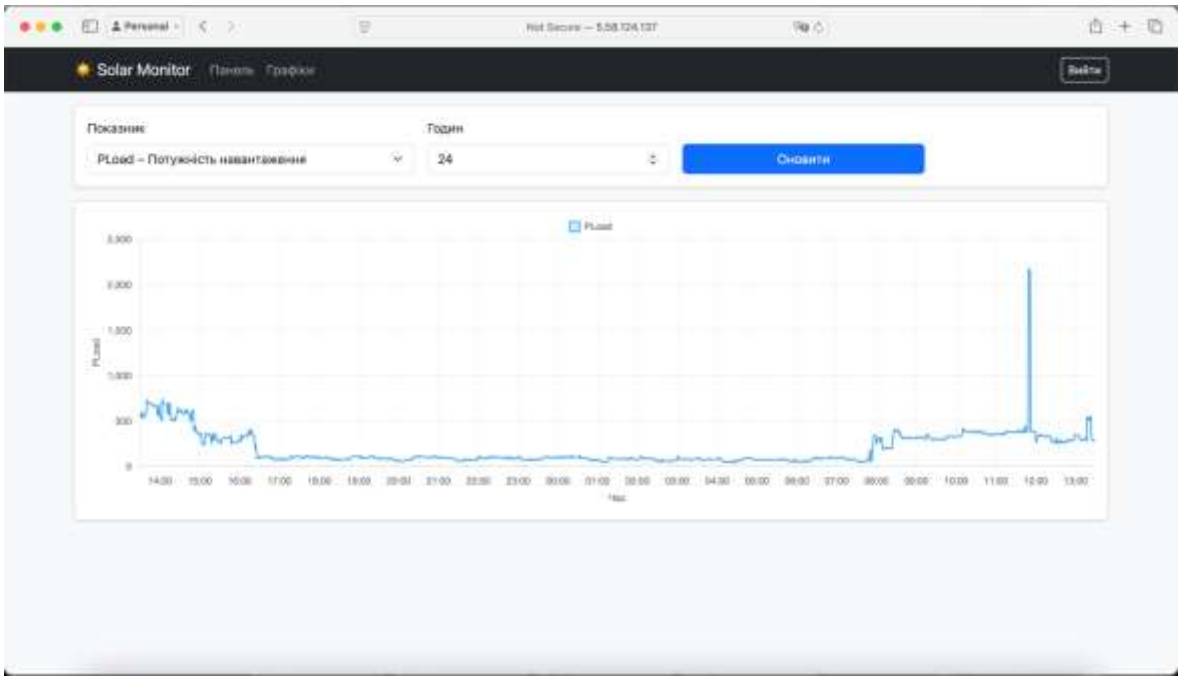

Рисунок 3.39 – Веб-панель у Safari на MacBook

Також, панель була протестована на мобільних пристроях – iPhone та Android. Тестування на мобільному пристрої iPhone у браузері Safari підтвердило адаптивність інтерфейсу та зручність використання [5, 17] – дивіться рисунок 3.40.

На мобільних пристроях бокове меню ховається (дивіться другий скріншот на рисунку 3.40) і, за потреби, відкривається при натисненні на нього. При цьому сам інтерфейс не “їде” та елементи коректно відображаються.

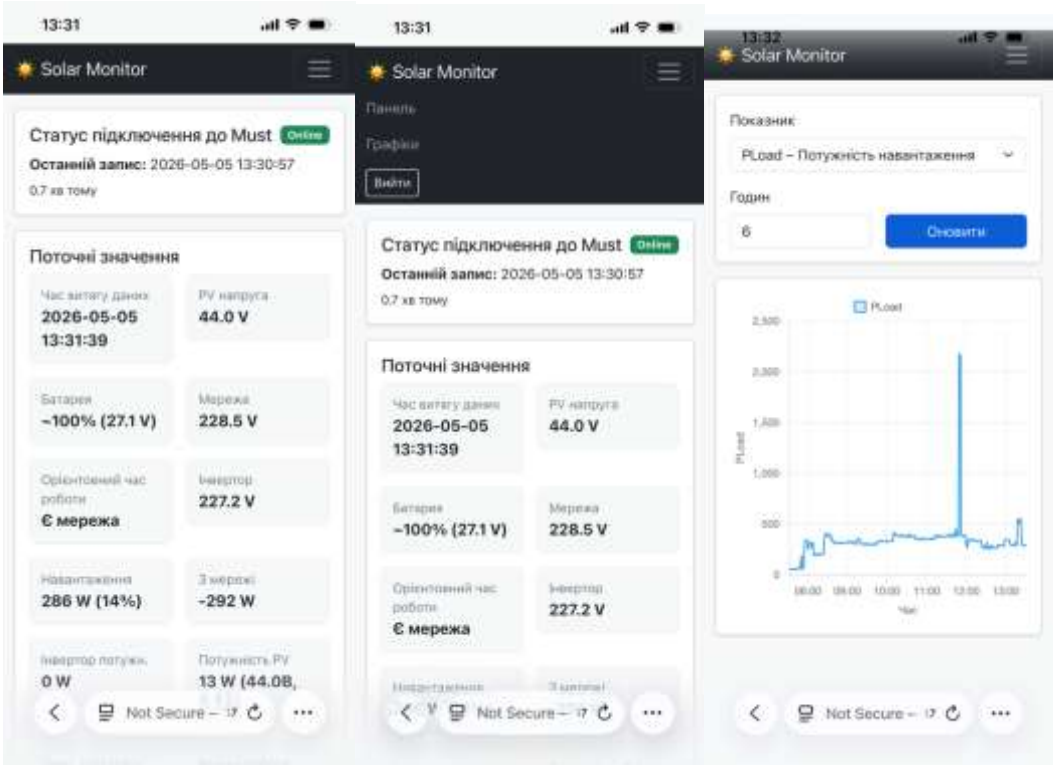

Рисунок 3.40 – Веб-панель на iPhone (Safari)

Тестування на пристрої з операційною системою Android у браузері Google Chrome також показало коректну роботу веб-панелі та відображення даних – дивіться рисунок 3.41.

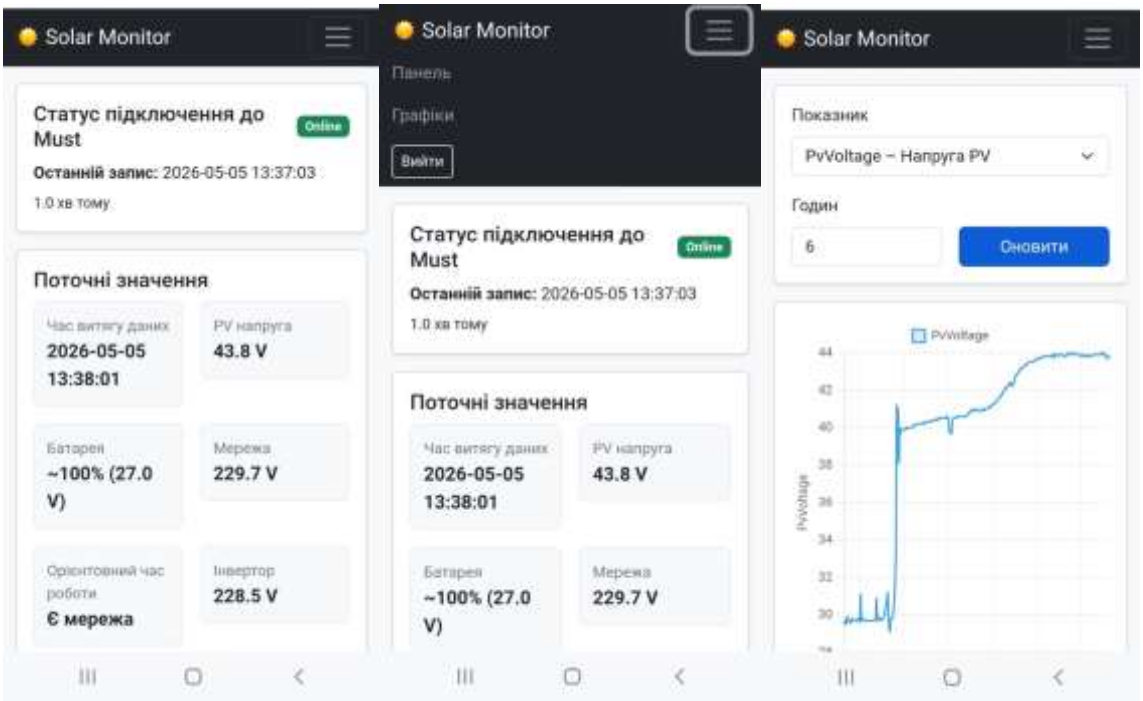

Рисунок 3.41 – Веб-панель на Android

У результаті тестування встановлено, що веб-панель коректно працює у різних браузерах та на різних типах пристроїв.

Тестування Telegram-бота проводилось з метою перевірки коректності роботи команд та механізму сповіщень [32].

Було перевірено виконання основних команд, зокрема /start та /status, а також коректність відображення поточного стану системи – дивіться рисунок 3.42.

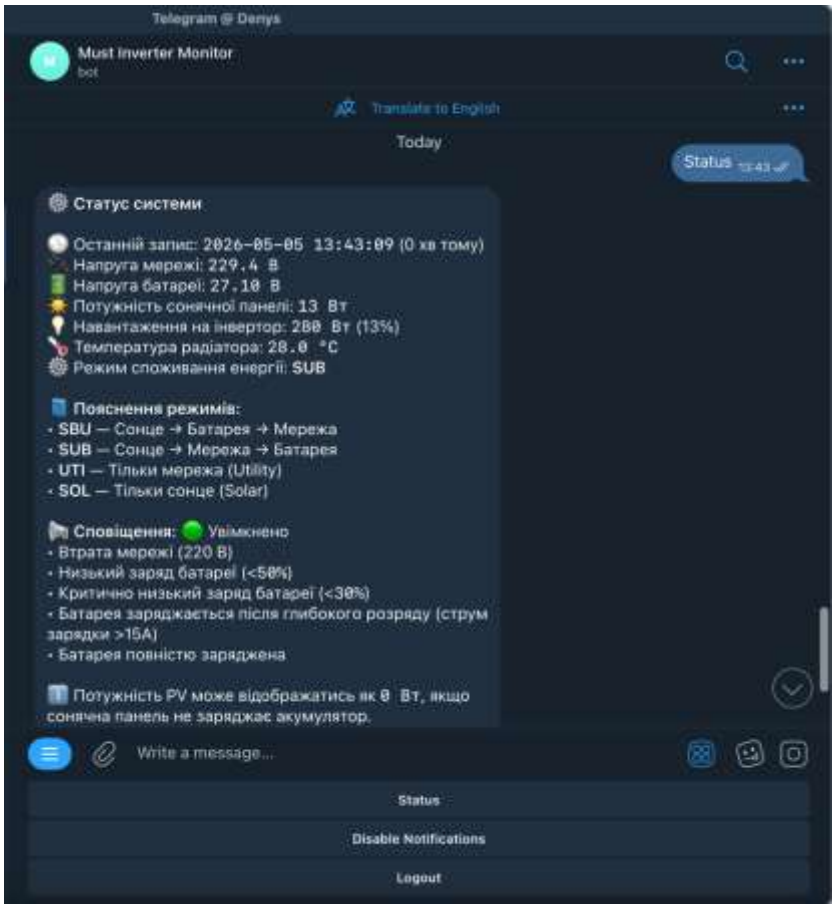

Рисунок 3.42 – Інтерфейс Telegram-бота на macOS та команда Status

Тестуємо функцію ввімкнення та вимкнення сповіщень [32]. Очікувана поведінка: коли сповіщення ввімкненні – кнопка має показувати “Disable” (з англ. Вимкнути) та навпаки. Отримуємо очікувану поведінку – дивіться рисунок 3.43.

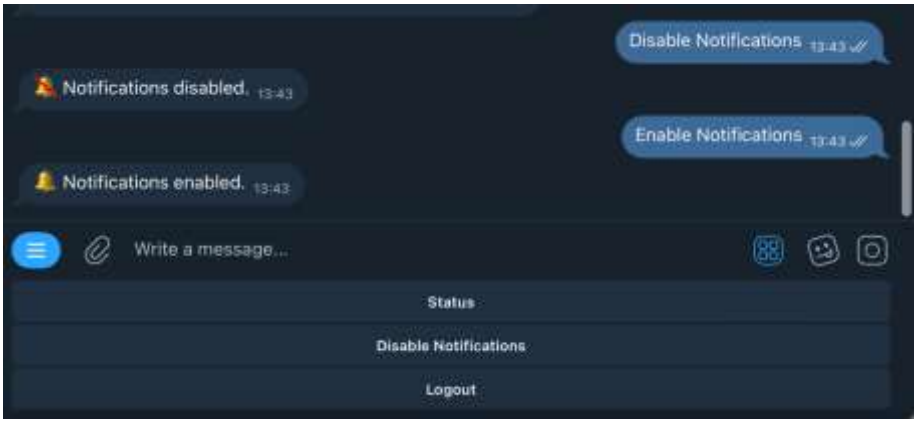

Рисунок 3.43 – Тестування ввімкнення та вимкнення сповіщень

Крім того, протестовано надсилання сповіщень про зміну стану енергосистеми, зокрема відсутність та відновлення електропостачання з відображенням тривалості відключення [32, 36] – дивіться рисунок 3.44.

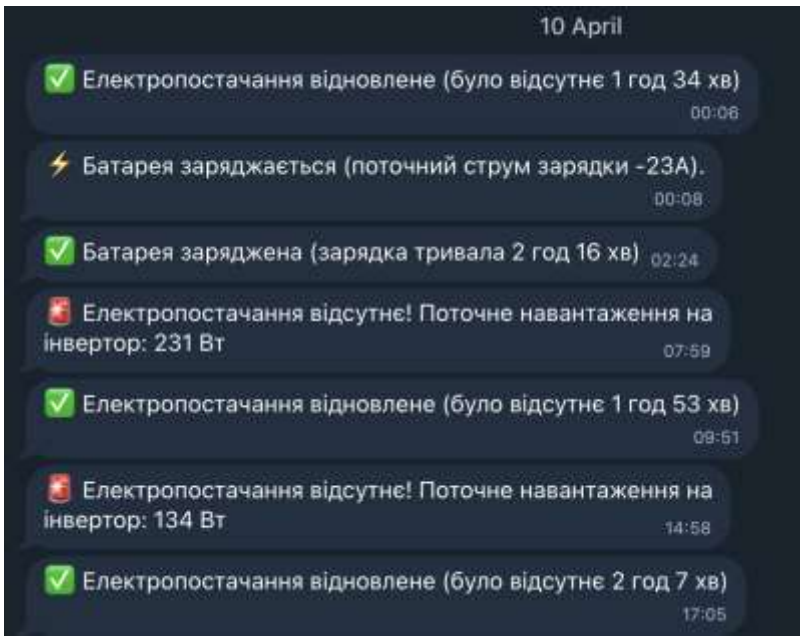

Рисунок 3.44 – Сповіщення “немає світла” та “відновлено (було X хв)”

Тестування сповіщень відбувалось в реальних ситуаціях (коли справді були вимкнення електроенергії). В результаті тестування встановлено, що Telegram-бот коректно реагує на події та забезпечує своєчасне інформування користувача (з затримкою до 1 хвилини на очікування актуальних даних з інвертора).

Проведене тестування підтвердило стабільну роботу системи моніторингу як у веб-інтерфейсі, так і у підсистемі сповіщень.

3.7 Висновок до третього розділу

У третьому розділі було представлено повну практичну реалізацію системи моніторингу інвертора Must, що включає серверну частину для збору даних, веб-панель для їх обробки й відображення, а також підсистему сповіщень на основі Telegram-бота. Описано роботу скрипта збору даних, який забезпечує регулярне отримання телеметрії через COM-порт віртуальної машини Windows 10 та передає її до централізованої бази даних MySQL. Такий підхід гарантує стабільність роботи системи, її незалежність від інтерфейсу користувача та можливість накопичення великого обсягу історичних даних.

Основну увагу було приділено розробленій веб-панелі на Flask, яка виступає головним інструментом взаємодії користувача з даними інвертора. Реалізована система підтримує адаптивний інтерфейс, відображення поточних параметрів, візуальну схему потоків енергії та інтерактивні графіки історичних показників. Завдяки інтеграції Bootstrap, Chart.js та шаблонів Jinja2, веб-панель забезпечує зручне й інтуїтивно зрозуміле керування переглядом інформації з різних пристроїв.

Розширена підсистема візуалізації дозволяє аналізувати поведінку інвертора у різні часові проміжки та робить веб-застосунок практичним інструментом для щоденного моніторингу роботи сонячної енергетичної системи. Гнучкий вибір параметрів і часових діапазонів значно підвищує інформативність системи та дає змогу користувачу оперативно оцінювати стан енергетичного обладнання.

Додатково було реалізовано підсистему сповіщень за допомогою Telegram-бота, яка функціонує як незалежний модуль та використовує дані з існуючої бази даних [32]. Основною перевагою даного підходу є можливість оперативного інформування користувача про критичні події, зокрема

відсутність електропостачання, відновлення мережі та тривалість відключень. Така реалізація дозволяє перейти від пасивного моніторингу до проактивної моделі взаємодії з користувачем.

У межах розділу також було виконано розгортання системи на базі платформи Proxmox VE із використанням віртуальних машин та контейнерів LXC. Це забезпечило ізоляцію компонентів системи, підвищення її відмовостійкості, а також зручність адміністрування і масштабування. Додатково було налаштовано мережеву взаємодію компонентів, доступ до веб-панелі та систему резервного копіювання, що підвищує надійність експлуатації системи у довготривалій перспективі.

Проведене тестування підтвердило коректну роботу системи у різних умовах експлуатації. Зокрема, веб-панель була перевірена на кросбраузерність (Google Chrome, Mozilla Firefox, Opera, Safari) та кросплатформеність (Windows, macOS, Android, iOS), що підтвердило її адаптивність та стабільність роботи. Telegram-бот також продемонстрував коректне виконання команд та своєчасне надсилання сповіщень у відповідь на зміну стану енергосистеми.

Таким чином, реалізована система є завершеним програмним рішенням для моніторингу енергетичних показників інвертора, яке поєднує збір даних, їх обробку, візуалізацію, сповіщення та інфраструктурне розгортання. Отримані результати можуть бути використані як основа для подальшого розвитку систем розумного домогосподарства та інтеграції з іншими компонентами автоматизації [36].

РОЗДІЛ 4. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Безпека при роботі з постійним та змінним струмом

Одним із важливих аспектів експлуатації сучасних енергетичних систем є забезпечення безпеки життєдіяльності користувачів під час роботи з електротехнічним обладнанням. Особливої актуальності це питання набуває для систем резервного та автономного електроживлення, до складу яких входять гібридні інвертори, акумуляторні батареї та фотоелектричні панелі. Такі системи одночасно використовують кола постійного та змінного струму, що створює потенційну небезпеку для життя людини у випадку порушення правил експлуатації.

У межах даної кваліфікаційної роботи розглядається система моніторингу енергосистеми розумної квартири на базі гібридного інвертора Must. Незважаючи на те, що програмна частина проєкту призначена для збору та відображення інформації про стан обладнання, її експлуатація пов’язана з реальними джерелами електричної енергії, які потребують дотримання вимог електробезпеки.

Основною небезпекою під час роботи з інверторними системами є можливість ураження людини електричним струмом. Згідно з положеннями безпеки життєдіяльності, електричний струм є одним із найбільш небезпечних виробничих та побутових факторів, оскільки його вплив може призводити до порушення роботи серцево-судинної системи, опіків, втрати свідомості та інших тяжких наслідків для здоров’я людини [37].

Особливістю сонячних енергетичних систем є використання джерел постійного струму. До них належать акумуляторні батареї та сонячні панелі. Постійний струм характеризується сталою полярністю та здатністю підтримувати електричну дугу під час розмикання електричного кола. У разі неправильного підключення обладнання або короткого замикання можливий

перегрів провідників, пошкодження обладнання та виникнення пожежонебезпечних ситуацій.

Акумуляторні батареї, що використовуються в системах резервного живлення, здатні накопичувати значний запас енергії. Навіть за відсутності зовнішнього електропостачання вони залишаються джерелом підвищеної небезпеки. Під час підключення акумуляторів необхідно суворо дотримуватися полярності з’єднань та рекомендацій виробника обладнання. Неправильне підключення може призвести до виходу інвертора з ладу або пошкодження акумуляторної батареї.

Окрему увагу слід приділяти сонячним панелям. Навіть після вимкнення інвертора фотоелектричні модулі продовжують виробляти електричну енергію за наявності сонячного випромінювання. Саме тому роботи з їх обслуговування повинні виконуватися лише після відключення відповідних ланцюгів та із застосуванням необхідних засобів захисту.

Крім постійного струму, у складі енергосистеми присутня мережа змінного струму напругою 220 В. Саме вона використовується для живлення побутових споживачів та забезпечення взаємодії інвертора із зовнішньою електромережею. Дотик до відкритих струмоведучих частин або пошкоджених елементів електропроводки може призвести до небезпечного ураження електричним струмом. З цієї причини будь-які роботи з електричними щитами, автоматичними вимикачами та силовими кабелями повинні виконуватися лише після повного знеструмлення відповідної ділянки мережі.

Для зменшення ризику виникнення аварійних ситуацій в енергосистемах використовуються автоматичні вимикачі, запобіжники, пристрої захисного відключення та системи заземлення. Такі засоби дозволяють оперативно відключити пошкоджену ділянку мережі та мінімізувати наслідки можливих несправностей.

Важливу роль у забезпеченні безпеки життєдіяльності відіграють сучасні системи моніторингу. Оперативне отримання інформації про параметри роботи інвертора, стан акумуляторних батарей та наявність зовнішнього

електроживлення дозволяє своєчасно виявляти відхилення від нормального режиму роботи. У розробленій системі користувач має можливість контролювати стан обладнання через веб-панель та отримувати повідомлення про критичні події за допомогою Telegram-бота. Це дозволяє швидко реагувати на відключення електроенергії, аварійні режими роботи та інші події, що можуть впливати на безпечну експлуатацію обладнання.

Таким чином, дотримання правил безпеки під час роботи з колами постійного та змінного струму є необхідною умовою експлуатації сучасних енергосистем розумної квартири. Використання засобів захисту, виконання вимог виробника обладнання та застосування систем моніторингу сприяють підвищенню рівня безпеки та зменшенню ризику виникнення небезпечних ситуацій для людини [37].

4.2 Організація безпечного робочого місця користувача персонального комп’ютера під час розробки та експлуатації системи моніторингу

Одним із важливих аспектів розробки та подальшої експлуатації інформаційних систем є забезпечення належних умов праці користувачів персональних комп’ютерів. У межах даної кваліфікаційної роботи основна частина програмного забезпечення створювалася та тестувалася із використанням комп’ютерної техніки. Крім того, взаємодія з розробленою системою моніторингу передбачає перегляд даних через веб-панель на персональному комп’ютері, ноутбуці або мобільному пристрої. Тому актуальним питанням охорони праці є організація безпечного робочого місця користувача персонального комп’ютера.

Охорона праці охоплює комплекс правових, організаційних, санітарно-гігієнічних та профілактичних заходів, спрямованих на збереження здоров’я і працездатності людини під час виконання професійних обов’язків [38]. Під час тривалої роботи за комп’ютером працівник може зазнавати впливу низки

факторів, які не становлять безпосередньої загрози життю, однак здатні негативно впливати на стан здоров’я. До них належать тривале перебування у статичній позі, підвищене навантаження на органи зору, недостатня рухова активність, нервово-емоційне напруження та втома.

Робоче місце користувача персонального комп’ютера повинно бути організоване таким чином, щоб забезпечувати комфортне положення тіла та мінімізувати надмірне навантаження на опорно-руховий апарат. Робочий стіл повинен мати достатню площу для розміщення монітора, клавіатури, маніпулятора типу «миша» та інших необхідних пристроїв. При цьому обладнання необхідно розташовувати так, щоб користувач мав змогу виконувати робочі операції без зайвого напруження рук, плечового поясу та шийного відділу хребта.

Важливим елементом організації робочого місця є правильний вибір крісла. Воно повинно забезпечувати підтримку спини, мати стійку конструкцію та дозволяти користувачу займати зручне положення під час тривалої роботи. Бажаною є можливість регулювання висоти сидіння та положення спинки. Ступні користувача повинні мати опору, а руки під час роботи з клавіатурою мають розташовуватися без надмірного напруження м’язів.

Монітор слід розміщувати безпосередньо перед користувачем, щоб уникнути постійних поворотів голови та тулуба. Екран повинен знаходитися на комфортній для зорового сприйняття відстані, а його верхня межа — приблизно на рівні очей або трохи нижче. Таке розташування сприяє зменшенню навантаження на шийний відділ хребта та допомагає підтримувати природне положення голови під час роботи.

Під час організації робочого місця користувача персонального комп’ютера необхідно враховувати ергономічні, санітарно-гігієнічні та організаційні вимоги. Раціональне розташування обладнання, достатнє освітлення, комфортне положення тіла та регулярне чергування праці й відпочинку сприяють зменшенню негативного впливу тривалої роботи з екранними пристроями на здоров’я працівника [39].

Окрему увагу необхідно приділяти освітленню робочого місця. Недостатня освітленість ускладнює сприйняття інформації та може призводити до швидкої втоми очей. Водночас надмірно яскраве світло або поява відблисків на поверхні екрана також негативно впливають на працездатність користувача. Доцільним є використання поєднання природного та штучного освітлення. Джерела світла слід розміщувати таким чином, щоб вони не створювали засліплення та не відбивалися на поверхні монітора.

Під час розробки програмного забезпечення значну частину часу працівник проводить у сидячому положенні. Тривале перебування в одній позі може спричиняти дискомфорт у спині та шиї, погіршувати кровообіг і знижувати загальну працездатність. Для зменшення негативного впливу статичного навантаження доцільно періодично змінювати положення тіла, виконувати короткі фізичні вправи та чергувати різні види діяльності.

Не менш важливим є дотримання раціонального режиму праці та відпочинку. Під час тривалої роботи з екранними пристроями рекомендується робити регулярні короткі перерви, під час яких доцільно змінити положення тіла, відвести погляд від монітора та виконати нескладні вправи для очей. Це дозволяє зменшити зорове навантаження, запобігти накопиченню втоми та підтримувати належний рівень концентрації уваги [39].

Суттєвий вплив на працездатність користувача мають також параметри мікроклімату приміщення. Для комфортної роботи необхідно забезпечити належний рівень температури, вологості та повітрообміну. Надмірно висока температура, недостатня вентиляція або занадто сухе повітря можуть спричиняти втому, зниження концентрації уваги та погіршення загального самопочуття.

Під час роботи з програмними засобами моніторингу важливе значення має не лише фізична організація робочого місця, але й зручність користувацького інтерфейсу. Перевантажений або недостатньо зрозумілий інтерфейс може збільшувати нервово-емоційне напруження та ускладнювати сприйняття інформації. У розробленій веб-панелі основні параметри інвертора

подано у структурованому вигляді, використано адаптивне компонування елементів та передбачено окрему сторінку для перегляду історичних графіків. Це дозволяє користувачу швидко знаходити необхідні показники без надмірного когнітивного навантаження.

Розроблена система також передбачає використання Telegram-бота для надсилання сповіщень про зміну стану енергосистеми. Такий підхід зменшує необхідність постійно контролювати веб-панель вручну та дозволяє отримувати важливу інформацію лише у випадку виникнення відповідної події. Це додатково знижує навантаження на користувача та робить експлуатацію системи зручнішою.

Таким чином, правильна організація робочого місця користувача персонального комп’ютера є важливою складовою охорони праці під час розробки та експлуатації інформаційних систем. Дотримання ергономічних вимог, забезпечення належного освітлення і мікроклімату, а також раціональне чергування праці та відпочинку сприяють збереженню здоров’я людини, зменшенню втоми та підвищенню ефективності роботи [38].

4.3 Висновок до четвертого розділу

У четвертому розділі було розглянуто питання безпеки життєдіяльності та охорони праці, пов’язані з розробкою та експлуатацією системи моніторингу енергосистеми розумної квартири. Особливу увагу приділено факторам, які можуть становити небезпеку для життя та здоров’я людини під час використання електротехнічного обладнання та комп’ютерних засобів.

У межах підрозділу з безпеки життєдіяльності було проаналізовано ризики, пов’язані з роботою енергосистем, що використовують постійний та змінний струм. Розглянуто особливості експлуатації акумуляторних батарей, сонячних панелей, інвертора та побутової електромережі напругою 220 В. Встановлено, що дотримання вимог електробезпеки, використання захисних

пристроїв та виконання рекомендацій виробника обладнання є необхідними умовами безпечної експлуатації таких систем.

У підрозділі з охорони праці було розглянуто вимоги до організації робочого місця користувача персонального комп’ютера під час розробки та експлуатації програмного забезпечення. Проаналізовано вплив факторів, пов’язаних із тривалою роботою за комп’ютером, зокрема навантаження на органи зору, опорно-руховий апарат та загальну працездатність людини. Визначено основні заходи щодо забезпечення комфортних і безпечних умов праці, серед яких правильна організація робочого місця, належне освітлення та дотримання режимів праці та відпочинку.

Проведений аналіз показав, що безпечна експлуатація енергосистем розумної квартири повинна поєднувати як технічні заходи захисту від небезпечних факторів електричного струму, так і створення належних умов праці під час роботи з програмними засобами моніторингу. Реалізована в роботі система моніторингу сприяє підвищенню рівня безпеки завдяки своєчасному інформуванню користувача про стан обладнання та можливі відхилення в його роботі.

Таким чином, розглянуті питання безпеки життєдіяльності та охорони праці підтверджують важливість комплексного підходу до експлуатації сучасних енергетичних систем і програмних засобів моніторингу, що забезпечує як безпеку людини, так і надійність функціонування обладнання.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було розроблено повноцінну веб-систему для моніторингу гібридного інвертора Must, яка забезпечує збір, збереження, обробку та візуалізацію телеметричних даних у реальному часі. Поставлена мета була повністю досягнута, а всі задачі, визначені на етапі проєктування, виконано у повному обсязі.

У ході роботи було проаналізовано принципи роботи гібридних інверторів та особливості наявного програмного забезпечення виробника. Проведений аналіз показав, що штатний застосунок має низку обмежень: він працює лише в операційній системі Windows, не передбачає перегляд даних із телефонів чи інших пристроїв, не має гнучкої системи побудови графіків та використовує громіздкі файли для збереження історії. Це визначило доцільність створення власного веб-рішення, яке було б мобільним, адаптивним і доступним у будь-який момент часу.

На основі проведеного аналізу було спроєктовано архітектуру системи, що включає кілька взаємопов’язаних компонентів: інвертор Must, робочу віртуальну машину Windows 10, сервер бази даних MySQL, веб-панель на Flask та інфраструктурне середовище Proxmox. Така модульна структура дозволила досягти ізольованості компонентів, високої стабільності роботи та простоти масштабування системи.

Реалізація серверної частини передбачала створення Python-скрипта, який зчитує параметри інвертора через COM-порт, перетворює їх у структурований формат та записує в MySQL з інтервалом у одну хвилину. Це забезпечило безперервне накопичення телеметрії та створило основу для подальшого аналізу. Окремою перевагою такого підходу є можливість паралельного використання заводської програми PowerSolarMonitor, що значно спрощувало налагодження та тестування системи на етапі розробки.

Головною складовою проєкту стала веб-панель, реалізована за допомогою фреймворку Flask [1]. Вона забезпечує зручний доступ до даних інвертора,

відображає поточні параметри системи, візуалізує потоки енергії та дозволяє переглядати історичні дані у вигляді інтерактивних графіків. Використання шаблонів Jinja2, адаптивної верстки Bootstrap та бібліотеки Chart.js дало змогу створити інтерфейс, який однаково добре працює на комп'ютерах, ноутбуках і мобільних пристроях.

Особливої уваги заслуговує реалізована система візуалізації. Користувач може вибрати будь-який показник (навантаження, напругу, температуру, потужність сонячної панелі тощо) та переглядати його зміну у будь-якому часовому діапазоні. Це дозволяє оцінювати продуктивність сонячної панелі, аналізувати поведінку акумулятора та відстежувати стан інвертора упродовж доби чи тижня. Інтерактивність графіків та їх автоматичне оновлення без перезавантаження сторінки роблять систему зручною та ефективною у використанні.

Загалом розроблена система повністю відповідає сучасним вимогам до веб-технологій: вона є адаптивною, інтерактивною, швидкодіючою та автономною. Вибраний технологічний стек (Python, Flask, MySQL, HTML/CSS, Bootstrap, Chart.js) продемонстрував свою ефективність, а побудована архітектура забезпечує легкість у подальшому розширенні. Система може бути вдосконалена шляхом додавання Telegram-сповіщень, інтелектуального аналізу даних, прогнозування виробітку та створення мобільного додатка — це відкриває перспективи для розширення проєкту у межах дипломної роботи [19-20].

Таким чином, розроблена веб-панель моніторингу інвертора Must є повністю працездатною, практично корисною та готовою до використання у реальних умовах. Вона забезпечує зручний і швидкий доступ до телеметрії енергетичної системи та створює основу для подальших досліджень і вдосконалень у сфері веб-технологій та енергетичного моніторингу.

ПЕРЕЛІК ДЖЕРЕЛ

1. Grinberg, Miguel. Flask Web Development. O'Reilly Media, 2018.
2. Python Software Foundation. "Python 3.12 Documentation." python.org, 2024.
3. Oracle. "MySQL 8.0 Reference Manual." Oracle, 2024.
4. The Chart.js Team. "Chart.js Documentation." chartjs.org, 2024.
5. Bootstrap Team. "Bootstrap 5 Documentation." getbootstrap.com, 2024.
6. Proxmox Server Solutions. "Proxmox VE Administration Guide." proxmox.com, 2024.
7. Must Power. "PV18-24/48 Series Hybrid Inverter User Manual." MustPower.net, 2023.
8. Microsoft. "Serial Port Communication in Windows." learn.microsoft.com, 2024.
9. SQLAlchemy Developers. "SQLAlchemy 2.0 Documentation." sqlalchemy.org, 2024.
10. Nizetic, Sandro. "A software tool for solar inverter monitoring." Renewable Energy, vol. 132, 2019, pp. 349–361.
11. Zhao, Jie, et al. "Monitoring of solar PV inverters using IoT." Energies, vol. 14, no. 13, 2021, p. 3886.
12. Rahman, M. M., et al. "A review of solar PV inverter technologies for residential applications." Solar Energy, vol. 219, 2021, pp. 50–66.
13. Islam, Zia Ul, et al. "Hybrid renewable energy systems…". Results in Engineering, vol. 27, 2025.
14. Jovic, Sanja, et al. "Smart home systems based on IoT." Sensors, vol. 20, no. 19, 2020, p. 5630.
15. Yigitbasioglu, O. M., and O. Velcu. "Business intelligence dashboards." Int. Journal of Accounting Information Systems, vol. 13, 2012, pp. 41–59.
16. Few, Stephen. Information Dashboard Design. Analytics Press, 2013.
17. Nielsen, Jakob. Usability Engineering. Morgan Kaufmann, 1994.

18. Juntunen, M., and S. Hyysalo. “Solar energy adoption in households.” Energy Policy, vol. 108, 2017, pp. 139–150.

19. Kumar, Neeraj, et al. “Energy monitoring systems in smart homes.” IEEE Access, vol. 8, 2020, pp. 17559–17578.

20. Al-Ali, A. R., et al. “Smart monitoring of solar energy systems.” Energy, vol. 170, 2019, pp. 1292–1301.

21. Luna, Á., et al. “Real-time monitoring of hybrid solar inverters.” Applied Energy, vol. 236, 2019, pp. 1188–1203.

22. Natsheh, E., et al. “PV system performance evaluation…”. Sustainability, vol. 11, 2019, p. 1508.

23. Li, Z., et al. “Internet-based monitoring of distributed PV systems.” Renewable Energy, vol. 150, 2020, pp. 314–328.

24. Jiang, Yi, et al. “Data acquisition and monitoring for PV systems.” Solar Energy, vol. 195, 2020, pp. 297–308.

25. Cabral, R., et al. “Battery SoC estimation using voltage curves.” Journal of Energy Storage, vol. 32, 2020.

26. Yan, Ruqi, et al. “Telemetric data processing for energy systems.” Energy Reports, vol. 6, 2020.

27. Shetty, Sachin, et al. “Secure architecture for smart home systems.” IEEE IoT Journal, vol. 7, no. 4, 2020.

28. Wang, Z., et al. “Real-time monitoring of residential energy systems.” Energy and Buildings, vol. 199, 2019.

29. Patel, Ajay. “Cloud-independent local monitoring for IoT energy devices.” Future Internet, vol. 13, 2021.

30. Rodrigues, E., et al. “Data visualization in IoT monitoring systems.” Sensors, vol. 21, no. 5, 2021.

31. Nurdien, Agief Prakasa, et al. "Comparative Performance Analysis of CSV and JSON Formats in Data Processing System." 2025 IEEE International Conference on Artificial Intelligence for Learning and Optimization (ICoAILO). IEEE, 2025.

32. Kanwer, Budesh, and Udit Mamodiya. "Development of an IoTbased Intelligent Energy Management System with Motion Notification on Telegram in a Smart Home." Proceedings of the 5th International Conference on Information Management & Machine Intelligence. 2023.

33. De Lima, João Pedro Magalhaes, et al. "Data Acquisition and Monitoring for Photovoltaic Plants Based on Low-Cost IoT Technologies." 2024 IEEE Biennial Congress of Argentina (ARGENCON). IEEE, 2024.

34. Wang, Chen, et al. "Apache iotdb: A time series database for iot applications." Proceedings of the ACM on Management of Data 1.2 (2023): 1-27.

35. Sisinni, Emiliano, et al. "Assessment of time performance of lightweight virtualization for edge computing applications." 2023 IEEE 19th International Conference on Factory Communication Systems (WFCS). IEEE, 2023.

36. Abbasi, Obaid Ur Rehman, et al. "Energy management strategy based on renewables and battery energy storage system with IoT enabled energy monitoring." Electrical Engineering 106.3 (2024): 3031-3043.

37. Желібо Є. П., Заверуха Н. М., Зацарний В. В. Безпека життєдіяльності : навч. посіб. / за ред. Є. П. Желібо. 6-те вид. Київ : Каравела, 2023. 344 с. ISBN 966-95596-4-2.

38. Андрейчук Н.І. Охорона праці : навч. посіб. / Н.І. Андрейчук, Ю.В. Кіт, С.В. Шибанов, О.В. Шерстньова. Львів : Видавництво Львівської політехніки, 2021. 276 с.

39. Бедрій Я.І. Основи охорони праці : навч. посіб. 4-е вид., перероб. і доп. Тернопіль : Навчальна книга – Богдан, 2018. 240 с.

40. Шимчук, Г. В., Назаревич, О. Б., Литвиненко, Я. В., Готович, В. А., Никитюк, В. В., & Боднарчук, І. О. (2025). Грід-системи та технології хмарних обчислень. Навчальний посібник для здобувачів освітнього рівня «магістр» спеціальностей: F3 «Комп’ютерні науки», F6 «Інформаційні системи та технології».

41. Leshchyshyn, Y., Scherbak, L., Nazarevych, O., Gotovych, V., Tymkiv, P., & Shymchuk, G. (2019, May). Multicomponent Model of the Heart Rate

Variability Change-point. In 2019 IEEE XVth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH) (pp. 110-113). IEEE.

42. Lytvynenko, I., Lupenko, S., Nazarevych, O., Shymchuk, G., & Hotovych, V. (2021, September). Mathematical model of gas consumption process in the form of cyclic random process. In 2021 IEEE 16th International Conference on Computer Sciences and Information Technologies (CSIT) (Vol. 1, pp. 232-235). IEEE.

43. Kozlovskyi, V., Balanyuk, Y., Martyniuk, H., Nazarevych, O., Scherbak, L., & Shymchuk, G. (2022, April). Information Technology for Estimating City Gas Consumption During the Year. In 2022 International Conference on Smart Information Systems and Technologies (SIST), Nur-Sultan, Kazakhstan (pp. 1-4).

44. Lytvynenko, I., Lupenko, S., Kunanets, N., Nazarevych, O., Shymchuk, G., & Hotovych, V. (2021). Simulation of gas consumption process based on the mathematical model in the form of cyclic random process considering the scale factors. In 1st International Workshop on Information Technologies: Theoretical and Applied Problems, ITTAP (Vol. 2021).

45. Kunanets, N., Pasichnyk, V., Bodnarchuk, I., Martsenko, S., Matsiuk, O., Matsiuk, A., ... & Shymchuk, H. (2019). Information system for visual analyzer disease diagnostics. In CEUR Workshop Proceedings (pp. 43-56).

46. Шимчук Г., Голотенко О., Небесний Р., Готович В. Застосування мови Scala у системах паралельних і хмарних обчислень. Наука і техніка сьогодні. 2026. № 4(58). С. 4794–4807. DOI: 10.52058/2786-6025-2026-4(58)-4794-4807.

47. Шевченко Н., Шимчук Г., Готович В., Голотенко О., Литвиненко С., Петрошук М. Математична модель для прогнозування змін у бездротових сенсорних мережах. Наука і техніка сьогодні. 2026. № 4(58). С. 4767–4782. DOI: 10.52058/2786-6025-2026-4(58)-4767-4782.

ДОДАТКИ

Додаток А – Код зборгу даних з інвертора

```
# main.py

# Standard Libraries
from typing import Optional
import os
from routines import *
from mapper import *

# Third-party Libraries
from dotenv import load_dotenv

# Custom Modules
from mysql_database import MysqlConnectionHandler
from mysql_database.tables import MustDataTable



class MustInverterDataHandler:
    def __init__(self):
        # Define the serial port (change the port name as needed)
        # (Optional) Get the port from .env
        load_dotenv()  # Load vars from .env into our environment
        # serial_port = "/dev/ttyUSB0"
        self._serial_port = os.getenv("MUST_PORT", "/dev/ttyUSB0")  # default, if not found

        # Define the configuration for command strings # Set to True or False as needed
        # This configuration can slow down the execution
```

```
        self._command_config = {
            "command_1_enabled": False,  # not really needed/implemented
            "command_2_enabled": False,
            "command_3_enabled": True,  # good data
            "command_4_enabled": False,
            "command_5_enabled": True,
            "command_6_enabled": True  # good data
        }

        # Construct the command strings based on the configuration
        charger_id = '04'
        inverter_id = '04'

        self._command_string_1 = f"{charger_id} 03 27 11 00 0A"
        self._command_string_2 = f"{charger_id} 03 27 75 00 18"
        self._command_string_3 = f"{charger_id} 03 3B 61 00 15"
        self._command_string_4 = f"{inverter_id} 03 4E 21 00 10"
        self._command_string_5 = f"{inverter_id} 03 4E 85 00 2C"
        self._command_string_6 = f"{inverter_id} 03 62 71 00 4F"

    def get_data(self) -> Optional[dict]:
        try:
            # Calculate the CRC value for the command
            command_bytes_1 = generate_crc(self._command_string_1)
            command_bytes_2 = generate_crc(self._command_string_2)
            command_bytes_3 = generate_crc(self._command_string_3)
            command_bytes_4 = generate_crc(self._command_string_4)
            command_bytes_5 = generate_crc(self._command_string_5)
            command_bytes_6 = generate_crc(self._command_string_6)
```

```
# Perform other operations with the serial connection
ser = serial.Serial(self._serial_port, baudrate=19200, timeout=1)
ser.setRTS(True)
responses = []

if self._command_config["command_1_enabled"]:
    response_1 = get_part_arr(ser, command_bytes_1, 10, 20)
    time.sleep(0.03)
    responses.append(convert_partArr1_to_json(response_1))

if self._command_config["command_2_enabled"]:
    response_2 = get_part_arr(ser, command_bytes_2, 24, 20)
    time.sleep(0.03)
    responses.append(convert_partArr2_to_json(response_2))

if self._command_config["command_3_enabled"]:
    response_3 = get_part_arr(ser, command_bytes_3, 21, 20)
    time.sleep(0.03)
    responses.append(convert_partArr3_to_json(response_3))

if self._command_config["command_4_enabled"]:
    response_4 = get_part_arr(ser, command_bytes_4, 16, 20)
    time.sleep(0.03)
    responses.append(convert_partArr4_to_json(response_4))

if self._command_config["command_5_enabled"]:
    # === Getting EnergyUseMode  ===
    response_5 = get_part_arr(ser, command_bytes_5, 44, 20)
    time.sleep(0.03)
    responses.append(convert_partArr5_to_json(response_5))
```

```
            if self._command_config["command_6_enabled"]:
                response_6 = get_part_arr(ser, command_bytes_6, 79, 40)
                responses.append(convert_partArr6_to_json(response_6))

            # Close the serial connection when done
            ser.close()

            # Gather all data and convert it to dict
            merged_result_str: str = merge_json(responses)
            merged_result: dict = json.loads(merged_result_str)

            return merged_result

        except Exception as e:
            print("Error while getting inverter's data:", str(e))



def main():
    print("Initializing project.")
    load_dotenv()  # load vars from .env into our environment

    # 1. Create instance of class MustInverterDataHandler
    must_inverter_data_handler = MustInverterDataHandler()

    # 2. Initialize MySQL database connection
    print("Connecting to MySQL, it may take some time, please wait...")
    mysql_connection_handler = MysqlConnectionHandler()
    mysql_connection_handler.initialize_connection(
        db_host=os.getenv("MYSQL_HOST", "localhost"),
```

```
        db_name=os.getenv("MYSQL_DATABASE", "must_data"),
        db_user=os.getenv("MYSQL_USER", "root"),
        db_password=os.getenv("MYSQL_PASSWORD", ""),
        pool_name="must_python_worker",
        pool_size=2
    )
    print("Initializing MySQL tables, it may take some time, please wait...")
    must_data_table = MustDataTable(connection_handler=mysql_connection_handler)
    must_data_table.initialize_table()

    # 3. Read & save data
    print("Getting inverter's data...")
    get_inverter_data_delay = os.getenv("DATA_GATHER_INTERVAL_SECONDS", default=60) # seconds
    if get_inverter_data_delay < 10:
        get_inverter_data_delay = 10  # min interval 10 seconds
    elif get_inverter_data_delay > 3600:
        get_inverter_data_delay = 3600  # max internal 1 hour
    while True:
        # 1. Get data
        must_data = must_inverter_data_handler.get_data()
        print("Inverter's data received:", must_data)

        # 2. Check & insert data
        if must_data and len(must_data) > 2:
            must_data_table.insert_data(data=must_data)
            print("Data inserted into the database.")
        else:
            print("Unable to get data from the inverter.")
```

```
        print(f"\nSleeping for {get_inverter_data_delay} seconds...")
        time.sleep(get_inverter_data_delay)

if __name__ == '__main__':
    main()
```

Додаток Б – Код веб панелі моніторингу

```
# app.py

# Standard Libraries
from datetime import datetime, timedelta
import os
from typing import Optional
from functools import wraps

# Third-party Libraries
from flask import Flask, render_template, jsonify, request
from flask import session as flask_session
from flask import flash, redirect, url_for
from flask_limiter import Limiter
from flask_limiter.util import get_remote_address
from flask_sqlalchemy import SQLAlchemy
from sqlalchemy.ext.automap import automap_base
from sqlalchemy import create_engine, MetaData
from dotenv import load_dotenv

load_dotenv()

# Flask
APP_SECRET_KEY = os.getenv("FLASK_SECRET", "dev")
# Project Data
APP_USERNAME = os.getenv("APP_USERNAME", "admin")
APP_PASSWORD = os.getenv("APP_PASSWORD", "admin")
DB_URL = os.getenv("DATABASE_URL")
```

```
OFFLINE_MINUTES = int(os.getenv("WORKER_OFFLINE_MINUTES", "2"))
TABLE_NAME = os.getenv("INVERTER_TABLE", "inverter_telemetry") # set to your real table name
# Inverter Data
BATTERY_MIN_VOLTAGE = float(os.getenv("BATTERY_MIN_VOLTAGE", 22.0))
BATTERY_MAX_VOLTAGE = float(os.getenv("BATTERY_MAX_VOLTAGE", 27.0))
BATTERY_CAPACITY_WH = float(os.getenv("BATTERY_CAPACITY_W", 1400))

app = Flask(__name__)

limiter = Limiter(
    get_remote_address,
    app=app,
    default_limits=["50 per hour"]  # глобальний ліміт на всі запити
)

app.config["SQLALCHEMY_DATABASE_URI"] = DB_URL
app.config["SQLALCHEMY_TRACK_MODIFICATIONS"] = False
app.secret_key = APP_SECRET_KEY
app.permanent_session_lifetime = timedelta(days=7)

db = SQLAlchemy(app)
engine = create_engine(DB_URL)
metadata = MetaData()
metadata.reflect(engine, only=[TABLE_NAME])
Base = automap_base(metadata=metadata)
```

```
Base.prepare()
Telemetry = Base.classes[TABLE_NAME]

# ----- Whitelisted metrics ------

# Whitelist of metrics we expose easily (add more as needed)
METRICS = {
    "BatteryVoltage": "BatteryVoltage",
    "PvVoltage": "PvVoltage",
    "GridVoltage": "GridVoltage",
    "InverterVoltage": "InverterVoltage",
    "PLoad": "PLoad",
    "PGrid": "PGrid",
    "PInverter": "PInverter",
    "ChargerPower": "ChargerPower",
    "LoadPercent": "LoadPercent",
    "RadiatorTemperature": "RadiatorTemperature",
    "ExternalTemperature": "ExternalTemperature",
    "BattPower": "BattPower",
    "BattCurrent": "BattCurrent",
    "PvPower": "ChargerPower",
}
# === Available only in charts ===
METRICS_CHARTS = {
    "PLoad": "Потужність навантаження",
    "BatteryVoltage": "Напруга батареї",
    "GridVoltage": "Напруга мережі",
    "InverterVoltage": "Напруга інвертора",
    "PInverter": "Потужність інвертора",
    "LoadPercent": "Завантаженість інвертора (%)",
```

```
    "PGrid": "Потужність мережі",
    "PvVoltage": "Напруга PV",
    "ChargerPower": "Потужність PV",
    "RadiatorTemperature": "Температура радіатора",
    "BattCurrent": "Струм батареї (A)",
    "BattPower": "Потужність батареї (W)"
}

# --------- Helpers ---------

def get_latest_row(db_session):
    return db_session.query(Telemetry).order_by(getattr(Telemetry,
"timestamp").desc()).first()



def worker_status(db_session):
    latest = get_latest_row(db_session)
    if not latest:
        return {"status": "offline", "last_seen": None, "minutes_ago": None}
    ts = getattr(latest, "timestamp")

    # Assume DB timestamps are in server local time. Adjust here if needed.
    minutes = (datetime.now() - ts).total_seconds() / 60.0
    minutes_rounded = round(minutes, 1)

    if minutes >= 60:
        minutes_ago_human =
format_minutes_human_ua(minutes=minutes_rounded)
        offline_minutes_ago_display = minutes_ago_human
    else:
```

```
        offline_minutes_ago_display = f"{minutes_rounded} хв"

    return {
        "status": "online" if minutes <= OFFLINE_MINUTES else "offline",
        "last_seen": ts.isoformat(sep=" ", timespec="seconds"),
        "minutes_ago": offline_minutes_ago_display,
    }


def format_minutes_human_ua(minutes: float | int) -> str:
    """
    Convert minutes into human-readable string:
    - 5m -> "5 хв"
    - 70m -> "1 год 10 хв"
    - 1500m -> "1 д 1 год"
    - 6708m -> "4 д 15 год 48 хв"
    """
    if minutes is None:
        return "—"

    try:
        minutes = int(round(minutes))
    except (TypeError, ValueError):
        return "—"

    if minutes < 0:
        minutes = 0

    days = minutes // 1440          # 24 * 60
    hours = (minutes % 1440) // 60
```

```
    mins = minutes % 60

    parts = []

    if days > 0:
        parts.append(f"{days} д")
    if hours > 0:
        parts.append(f"{hours} год")
    if mins > 0 or not parts:
        parts.append(f"{mins} хв")

    return " ".join(parts)


# ------- Dashboard Utilities --------

def battery_voltage_to_percent_curve(v: float) -> float:
    """
    Estimate battery SOC for 24V AGM/GEL based on voltage + load compensation.
    """

    # Clamp to realistic range
    v = max(22.7, min(25.7, v))

    # Voltage -> % curve (плоский верх)
    curve = {
        25.7: 100,
        25.4: 90,
        25.1: 80,
```

```
        24.8: 70,
        24.5: 60,
        24.2: 50,
        23.9: 40,
        23.6: 30,
        23.3: 20,
        23.0: 10,
        22.7: 0,
    }

    volts = sorted(curve.keys(), reverse=True)

    for i in range(len(volts) - 1):
        v1, v2 = volts[i], volts[i + 1]
        if v1 >= v >= v2:
            p1, p2 = curve[v1], curve[v2]
            # linear interpolation
            return p1 + (v - v1) * (p2 - p1) / (v2 - v1)

    return curve[volts[-1]]


def get_avg_battery_voltage(db_session, limit=5):
    rows = (
        db_session.query(Telemetry.BatteryVoltage)
        .order_by(Telemetry.timestamp.desc())
        .limit(limit)
        .all()
    )
    values = [r[0] for r in rows if r[0] is not None]
```

```
    return sum(values) / len(values) if values else 0.0


def get_avg_load(db_session, limit=15):
    rows = (
        db_session.query(Telemetry.PLoad)
        .order_by(Telemetry.timestamp.desc())
        .limit(limit)
        .all()
    )
    values = [r[0] for r in rows if r[0] is not None]
    return sum(values) / len(values) if values else 0.0


def calculate_inverter_runtime_minutes(
    battery_percent: float,
    safe_capacity_wh: float,
    avg_load_w: float,
    loss_factor: float = 1.05
) -> Optional[int]:
    if avg_load_w <= 0:
        return None

    # Доступна енергія у ват-годинах
    available_wh = safe_capacity_wh * (battery_percent / 100)

    # Реальне навантаження з урахуванням втрат
    effective_load = avg_load_w * loss_factor

    hours = available_wh / effective_load
```

```
    minutes = int(hours * 60)

    return max(minutes, 1)  # захист від нуля


# --------- helper functions ---------
def _safe_float(v, default=0.0):
    try:
        return float(v)
    except (TypeError, ValueError):
        return default


# --------- routes ---------
def login_required(f):
    @wraps(f)
    def wrapper(*args, **kwargs):
        # Доступ дозволено тільки якщо є session["logged_in"] = True
        if flask_session.get("logged_in"):
            return f(*args, **kwargs)
        return redirect(url_for("login", next=request.path))
    return wrapper


@app.route("/login", methods=["GET", "POST"])
@limiter.limit("5 per hour", methods=["POST"])
def login():
    if request.method == "POST":
        username = request.form.get("username", "")
        password = request.form.get("password", "")
```

```
        if username == APP_USERNAME and password == APP_PASSWORD:
            flask_session.permanent = True
            flask_session["logged_in"] = True

            # Повернути на сторінку, з якої користувач зайшов
            next_page = request.args.get("next", "/")
            return redirect(next_page)

        flash("Невірний логін або пароль", "danger")

    return render_template("login.html")


@app.route("/logout")
def logout():
    flask_session.clear()
    return redirect(url_for("login"))


@app.route("/")
@login_required
def dashboard():
    with db.session() as s:
        latest = get_latest_row(s)
        status = worker_status(s)

        # --- Battery voltage & percent ---
        batt_v = _safe_float(getattr(latest, "BatteryVoltage", 0.0)) if latest else 0.0
```

```
batt_v_avg = get_avg_battery_voltage(db_session=s, limit=15)  # беремо середнє з останніх записів
batt_pct = battery_voltage_to_percent_curve(v=batt_v_avg)

grid_v = _safe_float(getattr(latest, "GridVoltage", 0.0)) if latest else 0.0
batt_current = _safe_float(getattr(latest, "BattCurrent", 0.0)) if latest else 0.0

# Base display (we don't change anything in the logic)
batt_display = f"~{int(round(batt_pct))}% ({batt_v:.1f} V)"

# === UI behavior override ===
if grid_v > 0:
    # Є мережа -> дивимось на струм
    if batt_current <= -4:
        batt_display = "Заряджається ⚡"

# --- PV Power (prefer U*I, fallback to charger value) ---
if latest:
    pv_voltage = _safe_float(getattr(latest, "PvVoltage", 0.0))
    pv_current = _safe_float(getattr(latest, "ChargerCurrent", 0.0))

    calc_power = pv_voltage * pv_current
    calc_power = round(calc_power)

    charger_power = getattr(latest, "ChargerPower", None)
    charger_power = round(float(charger_power)) if charger_power is not None else None

    # Основний метод: U×I (тільки якщо струм > 0)
```

```
            if pv_current > 0:
                pv_power = calc_power
                pv_info_show_ui = True
            else:
                # fallback: ChargerPower
                pv_power = charger_power if charger_power is not None else calc_power
                pv_info_show_ui = False  # не показуємо U, I бо там 0A
        else:
            pv_power = 0
            pv_info_show_ui = False

        latest.PvEffectivePower = pv_power
        latest._pv_voltage = round(pv_voltage, 1) if latest else 0
        latest._pv_current = round(pv_current, 1) if latest else 0
        latest._pv_show_ui = pv_info_show_ui

        # Форматований текст для шаблону: "160 W (47.3В, 1.5А)"
        pv_voltage_fmt = f"{pv_voltage:.1f}В"
        pv_current_fmt = f"{pv_current:.1f}А"
        pv_extra_display = f"{pv_voltage_fmt}, {pv_current_fmt}"

        # --- Daily PV production (AccumulatedPower) ---
        pv_today = "0 кВт·год"

        if latest:
            today = datetime.now().date()

            # Перше значення AccumulatedPower після 00:00 сьогодні
            first_row = (
```

```
            s.query(Telemetry)
            .filter(Telemetry.timestamp >= datetime(today.year, today.month, today.day))
            .order_by(Telemetry.timestamp.asc())
            .first()
        )

        first_val = _safe_float(getattr(first_row, "AccumulatedPower", None)) if first_row else None
        last_val = _safe_float(getattr(latest, "AccumulatedPower", None))

        if first_val is not None and last_val is not None:
            diff = last_val - first_val
            if diff < 0:
                diff = 0
            pv_today = f"{diff:.2f} кВт·год"

    # --- Inverter energy mode (EnergyUseMode) ---
    energy_mode_map = {
        1: "SBU",
        2: "SUB",
        3: "UTI",
        4: "SOL",
    }

    raw_mode = getattr(latest, "EnergyUseMode", None)
    inverter_status = energy_mode_map.get(raw_mode, "Невідомий")

    # === RUNTIME ESTIMATION ===
    if latest and latest.GridVoltage == 0:
```

```
            # 1. Рахуємо середню напругу батареї і навантаження
            avg_batt_v = get_avg_battery_voltage(db_session=s, limit=5)
            avg_batt_pct = battery_voltage_to_percent_curve(v=avg_batt_v)

            # 2. Рахуємо середнє навантаження на інвертор
            # беремо 15 останні навантаження
            avg_load = get_avg_load(db_session=s, limit=15)

            # 3. Рахуємо орієнтовний час роботи
            minutes_left = calculate_inverter_runtime_minutes(
                battery_percent=avg_batt_pct,
                safe_capacity_wh=BATTERY_CAPACITY_WH,
                avg_load_w=avg_load,
                loss_factor=1.05
            )

            # Переводимо час роботи в формат для користувача
            runtime_display = format_minutes_human_ua(minutes=minutes_left)
        else:
            runtime_display = "Є мережа"

        # --- Round key power metrics ---
        for attr in ["PLoad", "PGrid", "PInverter", "BattPower", "ChargerPower",
"PvEffectivePower"]:
            if hasattr(latest, attr):
                try:
                    setattr(latest, attr, round(float(getattr(latest, attr) or 0)))
                except (ValueError, TypeError):
                    setattr(latest, attr, 0)
```

```
        # --- Time of data retrieval ---
        page_refresh_time = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

        return render_template(
            "dashboard.html",
            latest=latest,
            status=status,
            batt_display=batt_display,
            batt_pct=batt_pct,
            inverter_status=inverter_status,
            page_refresh_time=page_refresh_time,
            pv_today = pv_today,
            pv_extra_display=pv_extra_display,
            runtime_display=runtime_display
        )


@app.route("/charts")
@login_required
def charts():
    return render_template(
        "charts.html",
        metrics=METRICS_CHARTS,
        default_metric="PLoad"
    )


@app.route("/api/timeseries")
@login_required
def api_timeseries():
```

```
    metric = request.args.get("metric", "PLoad")
    hours = float(request.args.get("hours", 24))
    if metric not in METRICS:
        return jsonify({"ok": False, "error": "unknown metric"}), 400

    cutoff = datetime.now() - timedelta(hours=hours)
    field = getattr(Telemetry, METRICS[metric])
    ts_col = getattr(Telemetry, "timestamp")

    # Simple query: last N hours, ordered by time
    with db.session() as s:
        rows = (
            s.query(ts_col, field)
            .filter(ts_col >= cutoff)
            .order_by(ts_col.asc())
            .all()
        )

    # Format for Chart.js time scale
    # UPD: in Unix format
    points = [
        {"x": int(ts.timestamp() * 1000), "y": float(value) if value is not None else
None}
        for ts, value in rows
    ]

    return jsonify({"ok": True, "metric": metric, "points": points})


if __name__ == "__main__":
```

```
app.run(host="0.0.0.0", port=5000, debug=True)
```

Додаток В – Код Telegram-бота

```
# main.py

# === Standard Libraries ===
import asyncio
import os
from datetime import datetime, timedelta

# === Third-party Libraries ===
import nest_asyncio
import mysql.connector
from dotenv import load_dotenv
from telegram import Bot, Update
from telegram.ext import Application, CommandHandler, MessageHandler, filters, ContextTypes
from telegram import ReplyKeyboardMarkup



# === Load environment variables ===
load_dotenv()

BOT_TOKEN = os.getenv("BOT_TOKEN")
ADMIN_USER_ID = int(os.getenv("ADMIN_USER_ID", 0))

# --- Inverter DB (read-only) ---
DB_DATA = {
    "host": os.getenv("DB_DATA_HOST"),
    "user": os.getenv("DB_DATA_USER"),
```

```
    "password": os.getenv("DB_DATA_PASSWORD"),
    "database": os.getenv("DB_DATA_NAME"),
}
DB_DATA_TABLE = os.getenv("DB_DATA_TABLE", "must_data")

# --- Bot DB (read/write) ---
DB_BOT = {
    "host": os.getenv("DB_BOT_HOST"),
    "user": os.getenv("DB_BOT_USER"),
    "password": os.getenv("DB_BOT_PASSWORD"),
    "database": os.getenv("DB_BOT_NAME"),
}
DB_BOT_TABLE = os.getenv("DB_BOT_TABLE", "authorized_users")

ACCESS_PASSWORD = os.getenv("ACCESS_PASSWORD")
CHECK_INTERVAL = int(os.getenv("CHECK_INTERVAL", 60))
MAX_DELAY_MINUTES = int(os.getenv("MAX_DELAY_MINUTES", 2))
BATTERY_WARNING_PERCENT =
float(os.getenv("BATTERY_WARNING_PERCENT", 50))
BATTERY_ALERT_PERCENT =
float(os.getenv("BATTERY_ALERT_PERCENT", 30))
WEB_PANEL_URL = os.getenv("WEB_PANEL_URL", "")
WORKER_REMIND_INTERVAL =
int(os.getenv("WORKER_REMIND_INTERVAL", 15))

bot = Bot(token=BOT_TOKEN)
allowed_users = set()



# === Inverter Utils ====
```

```
def get_avg_battery_voltage(limit: int = 30) -> float:
    """Return average BatteryVoltage over the last N records from DB.
    Falls back to 0.0 if no data or query fails.
    """
    try:
        conn = mysql.connector.connect(**DB_DATA, connection_timeout=5)
        cur = conn.cursor(dictionary=True)
        cur.execute(
            f"""
            SELECT BatteryVoltage
            FROM {DB_DATA_TABLE}
            ORDER BY id DESC
            LIMIT {limit};
            """
        )
        rows = cur.fetchall()
        conn.close()

        if rows:
            voltages = [float(r["BatteryVoltage"]) for r in rows if r["BatteryVoltage"] is not None]
            if voltages:
                return sum(voltages) / len(voltages)
    except Exception as e:
        print(f"[DB_DATA ERROR] Failed to read recent battery voltages: {e}")
    return 0.0  # fallback


def get_avg_battery_current(limit: int = 10) -> float:
    """Return average BattCurrent over the last N records from DB.
```

```
    Negative values означають заряд (як у Must).
    Повертає 0.0, якщо немає даних або запит не вдався.
    """
    try:
        conn = mysql.connector.connect(**DB_DATA, connection_timeout=5)
        cur = conn.cursor(dictionary=True)
        cur.execute(
            f"""
            SELECT BattCurrent
            FROM {DB_DATA_TABLE}
            ORDER BY id DESC
            LIMIT {limit};
            """
        )
        rows = cur.fetchall()
        conn.close()

        if rows:
            currents = [float(r["BattCurrent"]) for r in rows if r["BattCurrent"] is not
None]
            if currents:
                return sum(currents) / len(currents)
    except Exception as e:
        print(f"[DB_DATA ERROR] Failed to read recent battery currents: {e}")
    return 0.0


def get_avg_charging_current(limit: int = 15) -> float | None:
    """
    Return average BattCurrent for CHARGING only (negative values)
```

```
over the last N records.

- Negative values = battery charging (Must logic)
- Positive values (discharge) are ignored
- Returns None if:
   - no data
   - no charging samples in window
   - DB error
"""
try:
   conn = mysql.connector.connect(**DB_DATA, connection_timeout=5)
   cur = conn.cursor(dictionary=True)
   cur.execute(
      f"""
      SELECT BattCurrent
      FROM {DB_DATA_TABLE}
      ORDER BY id DESC
      LIMIT {limit};
      """
   )
   rows = cur.fetchall()
   conn.close()

   if not rows:
      return None

   # Беремо ТІЛЬКИ заряд (негативні значення)
   charging_currents = [
      float(r["BattCurrent"])
      for r in rows
```

```
            if r["BattCurrent"] is not None and float(r["BattCurrent"]) < 0
        ]

        if not charging_currents:
            return None

        return sum(charging_currents) / len(charging_currents)

    except Exception as e:
        print(f"[DB_DATA ERROR] Failed to read charging battery current: {e}")
        return None


# === Utility Helpers ===
def format_duration(minutes: int, console_log: bool = False) -> str:
    """
    Format duration in minutes into a user-friendly string.
    Example:
      80 -> '1 год 20 хв'
      45 -> '45 хв'
    If console_log=True → returns English format (e.g. '1 h 20 min').
    """
    if console_log:
        if minutes < 60:
            return f"{minutes} min"
        hours = minutes // 60
        rem = minutes % 60
        if rem == 0:
            return f"{hours} h"
        return f"{hours} h {rem} min"
```

```
    else:
        if minutes < 60:
            return f"{minutes} хв"
        hours = minutes // 60
        rem = minutes % 60
        if rem == 0:
            return f"{hours} год"
        return f"{hours} год {rem} хв"


def format_minutes_human_ua(minutes: float | int) -> str:
    """
    Convert minutes into human-readable string:
    - 70  -> "1 год 10 хв"
    - 1500 -> "1 д 1 год"
    - 6708 -> "4 д 15 год 48 хв"
    """
    if minutes is None:
        return "—"

    try:
        minutes = int(round(minutes))
    except (TypeError, ValueError):
        return "—"

    if minutes < 0:
        minutes = 0

    days = minutes // 1440          # 24 * 60
    hours = (minutes % 1440) // 60
```

```
    mins = minutes % 60

    parts = []

    if days > 0:
        parts.append(f"{days} д")
    if hours > 0:
        parts.append(f"{hours} год")
    if mins > 0 or not parts:
        parts.append(f"{mins} хв")

    return " ".join(parts)


def battery_voltage_to_percent_curve(v: float) -> float:
    """
    Estimate battery SOC for 24V AGM/GEL based on voltage + load compensation.
    """

    # Clamp to realistic range
    v = max(22.7, min(25.7, v))

    # Voltage -> % curve (оновлена крива)
    curve = {
        25.7: 100,
        25.4: 90,
        25.1: 80,
        24.8: 70,
        24.5: 60,
```

```
        24.2: 50,
        23.9: 40,
        23.6: 30,
        23.3: 20,
        23.0: 10,
        22.7: 0,
    }

    volts = sorted(curve.keys(), reverse=True)

    for i in range(len(volts) - 1):
        v1, v2 = volts[i], volts[i + 1]
        if v1 >= v >= v2:
            p1, p2 = curve[v1], curve[v2]
            # linear interpolation
            return p1 + (v - v1) * (p2 - p1) / (v2 - v1)

    return curve[volts[-1]]



# === Database helpers ===
def get_latest_row():
    """Get the latest inverter data row."""
    try:
        conn = mysql.connector.connect(**DB_DATA, connection_timeout=5)
        cur = conn.cursor(dictionary=True)
        cur.execute(f"SELECT * FROM {DB_DATA_TABLE} ORDER BY id
DESC LIMIT 1;")
        row = cur.fetchone()
        conn.close()
```

```
        return row
    except Exception as e:
        print(f"[DB_DATA ERROR] {e}")
        return None


def get_allowed_users():
    """Load authorized users from bot DB."""
    try:
        conn = mysql.connector.connect(**DB_BOT, connection_timeout=5)
        cur = conn.cursor()
        cur.execute(
            f"CREATE TABLE IF NOT EXISTS {DB_BOT_TABLE} ("
            "user_id BIGINT PRIMARY KEY, "
            "username VARCHAR(255), "
            "first_name VARCHAR(255), "
            "last_name VARCHAR(255), "
            "notifications_enabled BOOLEAN DEFAULT TRUE, "
            "added_at DATETIME DEFAULT CURRENT_TIMESTAMP)"
        )
        cur.execute(f"SELECT user_id FROM {DB_BOT_TABLE};")
        users = {row[0] for row in cur.fetchall()}
        conn.close()
        print(f"[INIT] Authorized users loaded: {len(users)}")
        return users
    except Exception as e:
        print(f"[DB_BOT ERROR] {e}")
        return set()
```

```
def add_allowed_user(user_id: int, username: str, first_name: str, last_name: str):
    """Add a new authorized user with profile info."""
    try:
        conn = mysql.connector.connect(**DB_BOT, connection_timeout=5)
        cur = conn.cursor()
        cur.execute(
            f"CREATE TABLE IF NOT EXISTS {DB_BOT_TABLE} ("
            "user_id BIGINT PRIMARY KEY, "
            "username VARCHAR(255), "
            "first_name VARCHAR(255), "
            "last_name VARCHAR(255), "
            "notifications_enabled BOOLEAN DEFAULT TRUE, "
            "added_at DATETIME DEFAULT CURRENT_TIMESTAMP)"
        )
        cur.execute(
            f"INSERT IGNORE INTO {DB_BOT_TABLE} "
            "(user_id, username, first_name, last_name, notifications_enabled) "
            "VALUES (%s, %s, %s, %s, TRUE)",
            (user_id, username, first_name, last_name),
        )
        conn.commit()
        conn.close()
        print(f"[AUTH] New user added: {first_name} ({username or 'no_username'}) [{user_id}]")
    except Exception as e:
        print(f"[DB_BOT ERROR] {e}")


def toggle_notifications(user_id: int, enabled: bool):
```

```
    """Enable or disable notifications for a user."""
    try:
        conn = mysql.connector.connect(**DB_BOT, connection_timeout=5)
        cur = conn.cursor()
        cur.execute(
            f"UPDATE {DB_BOT_TABLE} SET notifications_enabled = %s WHERE user_id = %s",
            (enabled, user_id),
        )
        conn.commit()
        conn.close()
        print(f"[DB] Notifications {'enabled' if enabled else 'disabled'} for user {user_id}")
    except Exception as e:
        print(f"[DB_BOT ERROR] toggle_notifications: {e}")


def remove_user(user_id: int):
    """Remove a user from authorization table."""
    try:
        conn = mysql.connector.connect(**DB_BOT, connection_timeout=5)
        cur = conn.cursor()
        cur.execute(f"DELETE FROM {DB_BOT_TABLE} WHERE user_id = %s", (user_id,))
        conn.commit()
        conn.close()
        print(f"[DB] User removed: {user_id}")
    except Exception as e:
        print(f"[DB_BOT ERROR] remove_user: {e}")
```

```
# === Telegram handlers ===
async def start(update: Update, context: ContextTypes.DEFAULT_TYPE):
    """Handle /start command."""
    user = update.effective_user
    uname = f"@{user.username}" if user.username else "no_username"
    print(f"[COMMAND] /start from {user.first_name} ({uname}) [{user.id}]")

    if user.id in allowed_users:
        # Fetch current notification status
        conn = mysql.connector.connect(**DB_BOT, connection_timeout=5)
        cur = conn.cursor(dictionary=True)
        cur.execute(f"SELECT notifications_enabled FROM {DB_BOT_TABLE}
WHERE user_id = %s", (user.id,))
        row = cur.fetchone()
        conn.close()
        notif_enabled = row and row["notifications_enabled"]

        # Build dynamic keyboard
        keyboard = [["Status"]]
        if notif_enabled:
            keyboard.append(["Disable Notifications"])
        else:
            keyboard.append(["Enable Notifications"])
        keyboard.append(["Logout"])

        reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
        await update.message.reply_text(
            "✅You are authorized. Use the buttons below:",
            reply_markup=reply_markup,
```

```
        )
    else:
        keyboard = [["Authorize"]]
        reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
        await update.message.reply_text(
            "☐ Please enter the access password or tap *Authorize*.",
            parse_mode="Markdown",
            reply_markup=reply_markup,
        )


async def message_handler(update: Update, context: ContextTypes.DEFAULT_TYPE):
    """Handle regular text messages."""
    user = update.effective_user
    uname = f"@{user.username}" if user.username else "no_username"
    text = update.message.text.strip()
    text_lower = text.lower()

    print(f"[MESSAGE] From {user.first_name} ({uname}) [{user.id}]: {text}")

    # === Handle reply keyboard buttons first ===
    # --- Logout ---
    if text_lower == "logout":
        if user.id in allowed_users:
            remove_user(user.id)
            allowed_users.remove(user.id)
            print(f"[LOGOUT] {user.first_name} ({uname}) [{user.id}]")
            await update.message.reply_text("☐ You have been logged out.")
            await start(update, context)
```

```
        else:
            await update.message.reply_text("☐ You are not logged in.")
        return

    # --- Authorize (pressed button, not password) ---
    if text_lower == "authorize":
        if user.id in allowed_users:
            await update.message.reply_text("✅You are already authorized.")
        else:
            await update.message.reply_text("☐ Please enter your access password:")
        return

    # --- Disable / Enable notifications ---
    if text_lower in ["disable notifications", "enable notifications"]:
        enable = text.lower() == "enable notifications"
        toggle_notifications(user.id, enable)

        # Build updated keyboard
        keyboard = [["Status"]]
        if enable:
            keyboard.append(["Disable Notifications"])
            msg = "☐ Notifications enabled."
        else:
            keyboard.append(["Enable Notifications"])
            msg = "☐ Notifications disabled."
        keyboard.append(["Logout"])

        reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
```

```
        # Send concise status message with updated keyboard
        await update.message.reply_text(msg, reply_markup=reply_markup)
        print(f"[NOTIFY TOGGLE] {user.first_name} ({uname}) [{user.id}] → {enable}")
        return

    # --- Status (from button or text) ---
    if text_lower == "status":
        await status(update, context)
        return

    # === Password check ===
    if user.id not in allowed_users and text == ACCESS_PASSWORD:
        add_allowed_user(user.id, user.username, user.first_name, user.last_name)
        allowed_users.add(user.id)
        print(f"[ACCESS GRANTED] {user.first_name} ({uname}) [{user.id}]")

        # Deleting right password
        try:
            await update.message.delete()
            print(f"[SECURITY] Deleted password message from {user.first_name} ({uname})")
        except Exception as e:
            print(f"[WARN] Could not delete password message: {e}")

        await update.message.reply_text("✅Access granted! You will now receive alerts.")
        await start(update, context)  # updating menu
        return
```

```
    # === Wrong password ===
    if user.id not in allowed_users:
        await update.message.reply_text("☐ Wrong password.")
        print(f"[ACCESS DENIED] {user.first_name} ({uname}) [{user.id}]")
        return

    # === /status typed manually ===
    if text == "/status":
        await status(update, context)


async def status(update: Update, context: ContextTypes.DEFAULT_TYPE):
    """Show current inverter status."""
    user = update.effective_user
    uname = f"@{user.username}" if user.username else "no_username"
    print(f"[COMMAND] /status from {user.first_name} ({uname}) [{user.id}]")

    if user.id not in allowed_users:
        await update.message.reply_text("☐ You are not authorized.")
        print(f"[DENIED] {user.first_name} ({uname}) [{user.id}] tried /status")
        return

    row = get_latest_row()
    if not row:
        await update.message.reply_text("☐ No data in the database.")
        print(f"[STATUS] No data for {user.first_name} ({uname})")
        return

    ts = row.get("timestamp")
```

```
    if not ts:
        await update.message.reply_text("☐ Invalid record format.")
        print(f"[STATUS] Invalid timestamp for {user.first_name} ({uname})")
        return

    delay = datetime.now() - ts

    # --- PV Power calculation (prefer U*I, fallback to ChargerPower) ---
    if row:
        pv_voltage = float(row.get("PvVoltage", 0))
        pv_current = float(row.get("ChargerCurrent", 0))

        calc_power = round(pv_voltage * pv_current)

        charger_power = row.get("ChargerPower")
        charger_power = round(float(charger_power)) if charger_power is not None else None

        # Основний метод: U×I (якщо є струм)
        if pv_current > 0:
            pv_power = calc_power
        else:
            # fallback: ChargerPower
            pv_power = charger_power if charger_power is not None else calc_power
    else:
        pv_power = 0

    # === Get energy use mode ===
    energy_mode_map = {
```

```
        1: "SBU",
        2: "SUB",
        3: "UTI",
        4: "SOL",
    }

    raw_mode = row.get("EnergyUseMode")
    inverter_status = energy_mode_map.get(raw_mode, "Невідомо")

    # --- Fetch notification status from DB ---
    try:
        conn = mysql.connector.connect(**DB_BOT, connection_timeout=5)
        cur = conn.cursor(dictionary=True)
        cur.execute(f"SELECT notifications_enabled FROM {DB_BOT_TABLE}
WHERE user_id = %s", (user.id,))
        user_row = cur.fetchone()
        conn.close()
        notif_enabled = user_row and user_row["notifications_enabled"]
    except Exception as e:
        print(f"[DB_BOT ERROR] fetch notif status: {e}")
        notif_enabled = True

    # === Get notifications status ===
    notif_status = "☐ Увімкнено" if notif_enabled else "☐ Вимкнено"

    # --- Compose message ---
    msg = (
        f"☐ *Статус системи*\n\n"
        f"☐ Останній запис: `{ts}` ({delay.seconds // 60} хв тому)\n"
```

```
        f"☐ Напруга мережі: `{row.get('GridVoltage', 0):.1f} В`\n"
        f"☐ Напруга батареї: `{row.get('BatteryVoltage', 0):.2f} В`\n"
        f"☀ Потужність сонячної панелі: `{pv_power:.0f} Вт`\n"
        f"☐ Навантаження на інвертор: `{row.get('PLoad', 0):.0f} Вт` "
        f"({row.get('LoadPercent', 0):.0f}%)\n"
        f"☐ Температура радіатора: `{row.get('RadiatorTemperature', 0):.1f} °C`\n"
        f"☐ Режим споживання енергії: *{inverter_status}*\n\n"
        f"☐ *Пояснення режимів:*\n"
        f"• *SBU* — Сонце → Батарея → Мережа\n"
        f"• *SUB* — Сонце → Мережа → Батарея\n"
        f"• *UTI* — Тільки мережа (Utility)\n"
        f"• *SOL* — Тільки сонце (Solar)\n\n"
        f"☐ *Сповіщення:* {notif_status}\n"
        f"• Втрата мережі (220 В)\n"
        f"• Низький заряд батареї (<`{round(BATTERY_WARNING_PERCENT)}%`)\n"
        f"• Критично низький заряд батареї (<`{round(BATTERY_ALERT_PERCENT)}%`)\n"
        f"• Батарея заряджається після глибокого розряду (струм зарядки >15A)\n"
        f"• Батарея повністю заряджена\n\n"
        f"ℹ Потужність PV може відображатись як `0 Вт`, "
        f"якщо сонячна панель не заряджає акумулятор.\n\n"
    )

    # --- Optional Web panel link ---
    if WEB_PANEL_URL:
        msg += f"☐ [Веб-панель]({WEB_PANEL_URL})"
```

```
    await update.message.reply_text(msg, parse_mode="Markdown", disable_web_page_preview=True)
    print(f"[STATUS] Sent to {user.first_name} ({uname}) [{user.id}] — notif: {notif_status}")


# === Notification helper ===
async def notify_all(text):
    """Send a message to all authorized users who have notifications enabled."""
    print(f"[NOTIFY] Sending alert: {text}")

    try:
        conn = mysql.connector.connect(**DB_BOT, connection_timeout=5)
        cur = conn.cursor()
        cur.execute(f"SELECT user_id FROM {DB_BOT_TABLE} WHERE notifications_enabled = TRUE")
        recipients = [row[0] for row in cur.fetchall()]
        conn.close()
    except Exception as e:
        print(f"[DB_BOT ERROR notify_all]: {e}")
        recipients = list(allowed_users)

    for uid in recipients:
        try:
            await bot.send_message(chat_id=uid, text=text)
        except Exception as e:
            print(f"[NOTIFY ERROR] {uid}: {e}")
```

```
# === Monitoring loop ===
async def monitor_task():
    """Background task that checks inverter data periodically."""
    print("Monitoring started.")

    last_alert_worker = None
    last_alert_grid = None
    last_alert_battery  = None
    last_alert_charge = None
    charge_start_time = None

    while True:
        try:
            row = get_latest_row()

            # Notify admin only
            if not row:
                msg = "☐ Database empty or unavailable."
                print(f"[MONITOR ALERT] {msg}")

                # Send only to admin!
                if ADMIN_USER_ID:
                    try:
                        await bot.send_message(chat_id=ADMIN_USER_ID, text=msg)
                        print(f"[MONITOR NOTICE] Sent DB alert to admin ({ADMIN_USER_ID})")
                    except Exception as e:
                        print(f"[NOTIFY ERROR] Cannot send DB alert to admin: {e}")

                await asyncio.sleep(CHECK_INTERVAL)
```

```
            continue

        ts = row["timestamp"]
        now = datetime.now()
        delay = now - ts

        # === Worker offline / back online detection (admin only, DB-based downtime fallback) ===
        if delay > timedelta(minutes=MAX_DELAY_MINUTES):
            minutes_offline = int(delay.total_seconds() // 60)

            # ігноруємо короткі лаги (менше 2 хв)
            if minutes_offline < MAX_DELAY_MINUTES:
                pass
            elif (
                last_alert_worker is None
                or (now - last_alert_worker) >= timedelta(minutes=WORKER_REMIND_INTERVAL)
            ):
                # minutes_offline_friendly = format_duration(minutes=minutes_offline)
                long_minutes_offline_human = format_minutes_human_ua(minutes=minutes_offline)
                msg = f"□ Втрачено зв'язок з Must! Останній запис {long_minutes_offline_human} тому."
                minutes_offline_fr_console = format_duration(minutes=minutes_offline, console_log=True)
                print(f"[MONITOR ALERT] Must worker offline! Last record {minutes_offline_fr_console}.")
                if ADMIN_USER_ID:
```

```
                try:
                    await bot.send_message(chat_id=ADMIN_USER_ID, text=msg)
                except Exception as e:
                    print(f"[NOTIFY ERROR] Cannot send worker alert to admin: {e}")
            last_alert_worker = now

        else:
            # Worker back online
            if last_alert_worker is not None:
                downtime_friendly = None
                downtime_minutes = 0

                try:
                    # Отримуємо останній запис з таблиці даних (момент, коли воркер востаннє писав)
                    conn = mysql.connector.connect(**DB_DATA, connection_timeout=5)
                    cur = conn.cursor(dictionary=True)
                    cur.execute(
                        f"""
                        SELECT timestamp
                        FROM {DB_DATA_TABLE}
                        ORDER BY id DESC
                        LIMIT 1 OFFSET 1;
                        """
                    )
                    last_record = cur.fetchone()
                    conn.close()
```

```
                if last_record and last_record["timestamp"]:
                    last_ts = last_record["timestamp"]
                    downtime_minutes = int((datetime.now() - last_ts).total_seconds() // 60)
                    downtime_friendly = format_duration(downtime_minutes)

            except Exception as e:
                print(f"[DB_DATA ERROR] Failed to calculate worker downtime: {e}")

            # --- fallback через локальну змінну ---
            if not downtime_friendly:
                downtime_minutes = int((datetime.now() - last_alert_worker).total_seconds() // 60)
                downtime_friendly = format_duration(downtime_minutes)

            msg = f"✅ Зв'язок з Must відновлено (був оффлайн {downtime_friendly})."
            print(f"[MONITOR INFO] Must worker online! Downtime {format_duration(downtime_minutes, console_log=True)} ({'DB' if downtime_friendly else 'local'} based).")
            if ADMIN_USER_ID:
                try:
                    await bot.send_message(chat_id=ADMIN_USER_ID, text=msg)
                except Exception as e:
                    print(f"[NOTIFY ERROR] Cannot send worker restore alert: {e}")
```

```
            last_alert_worker = None

        # === Grid power loss / restore detection (DB-based downtime with fallback) ===
        grid_voltage = float(row.get("GridVoltage", 0))

        # --- Grid lost ---
        if grid_voltage == 0:
            if last_alert_grid is None:  # trigger only once per runtime
                msg = f"☐ Електропостачання відсутнє! Поточне навантаження на інвертор: {row.get('PLoad', 0):.0f} Вт"
                print("[MONITOR ALERT] Grid lost")
                last_alert_charge = None  # reset battery charging notification
                await notify_all(msg)
                last_alert_grid = now  # фіксуємо момент втрати

        # --- Grid restored ---
        else:
            if last_alert_grid is not None:
                downtime_friendly = None
                downtime_minutes = 0

                try:
                    # Спроба отримати останній запис, коли мережа була присутня
                    conn = mysql.connector.connect(**DB_DATA, connection_timeout=5)
                    cur = conn.cursor(dictionary=True)
                    cur.execute(
                        f"""
```

```
                SELECT timestamp
                FROM {DB_DATA_TABLE}
                WHERE GridVoltage > 0
                ORDER BY id DESC
                LIMIT 1 OFFSET 1;
                """
            )
            last_online_row = cur.fetchone()
            conn.close()

            if last_online_row and last_online_row["timestamp"]:
                last_online = last_online_row["timestamp"]
                downtime_minutes = int((datetime.now() - last_online).total_seconds() // 60)
                downtime_friendly = format_duration(downtime_minutes)
        except Exception as e:
            print(f"[DB_DATA ERROR] Failed to calculate grid downtime: {e}")

        # --- fallback через локальну змінну ---
        if not downtime_friendly:
            downtime_minutes = int((datetime.now() - last_alert_grid).total_seconds() // 60)
            downtime_friendly = format_duration(downtime_minutes)

        msg = f"✅ Електропостачання відновлене (було відсутнє {downtime_friendly})"
        print(f"[MONITOR INFO] Grid restored after {format_duration(downtime_minutes, console_log=True)} ({'DB' if downtime_friendly else 'local'} based).")
```

```
            await notify_all(msg)

            last_alert_grid = None  # скидаємо стан

        # === Battery charge check (with 5-sample voltage smoothing) ===
        # battery_voltage = float(row.get("BatteryVoltage", 0))
        battery_avg_voltage = get_avg_battery_voltage(limit=30)

        batt_percent = battery_voltage_to_percent_curve(battery_avg_voltage)

        # Warning threshold (менше ~50%)
        if BATTERY_WARNING_PERCENT > batt_percent > BATTERY_ALERT_PERCENT and last_alert_battery is None:
            msg = (
              f"☐ Рівень заряду батареї нижчий за 50%.\n"
              f"☐ Напруга: {battery_avg_voltage:.2f} В → {batt_percent:.0f}%"
            )
            print(f"[MONITOR ALERT] Battery warning ({battery_avg_voltage:.2f} V, {batt_percent:.0f}%)")
            await notify_all(msg)
            last_alert_battery = "warning"

        # Critical threshold (менше ~30%)
        elif batt_percent <= BATTERY_ALERT_PERCENT and last_alert_battery != "critical":
            msg = (
              f"☐ Критично низький рівень заряду батареї!\n"
```

```
                f"□ Напруга: {battery_avg_voltage:.2f} В → {batt_percent:.0f}%"
            )
            print(f"[MONITOR ALERT] Battery critical ({battery_avg_voltage:.2f} V, {batt_percent:.0f}%)")
            await notify_all(msg)
            last_alert_battery = "critical"

        # --- Reset only when батарея дійсно заряджена ---
        elif batt_percent > 99:
            last_alert_battery = None

        # === Battery charging status detection (simple duration tracking) ===
        batt_current = float(row.get("BattCurrent", 0))

        avg_fast_batt_current_values = 3
        avg_batt_current_values = 30  # 30 min to avoid false detecting
        batt_avg_current_fast = get_avg_battery_current(limit=avg_fast_batt_current_values)
        batt_avg_charging_current = get_avg_battery_current(limit=avg_batt_current_values)
        batt_avg_charge_current_negative_only = get_avg_charging_current(limit=avg_batt_current_values)

        # TODO DEBUG
        print(f"Batt current: {batt_current} A")
        print(f"Batt charge trigger average current: {batt_avg_current_fast} A (avg last {avg_fast_batt_current_values} values)")
        print(f"Batt fully charged trigger average current: {batt_avg_charging_current} A (avg last {avg_batt_current_values} values) ")
```

```
        print(f"Batt avg current negative only: {batt_avg_charge_current_negative_only} A (last {avg_batt_current_values} values avg)")

        # --- Battery starts charging ---
        if (
            batt_avg_current_fast <= -10
            and last_alert_charge != "charging"
            and batt_avg_current_fast < 0
        ):
            msg = f"⚡ Батарея заряджається (поточний струм зарядки {round(batt_current)}A)."
            print(f"[MONITOR INFO] Battery charging started (current {round(batt_current)}A).")
            await notify_all(msg)
            last_alert_charge = "charging"
            charge_start_time = now  # фіксуємо час початку зарядки

        # --- Battery fully charged ---
        elif (
            grid_voltage != 0
            and batt_avg_charge_current_negative_only is not None
            and -1 >= batt_avg_charge_current_negative_only >= -5
            and last_alert_charge == "charging"
        ):
            if charge_start_time:
                charge_duration = now - charge_start_time
                charge_minutes = int(charge_duration.total_seconds() // 60)
                msg = f"✅ Батарея заряджена (зарядка тривала {format_duration(charge_minutes)})"
```

```
                print(
                    f"[MONITOR INFO] Battery fully charged. Charging lasts {charge_minutes} min. "
                    f"Battery current: {batt_current:.2f} A. "
                    f"Batt avg charging current: {batt_avg_charging_current:.2f} A. "
                    f"Batt avg current: {batt_avg_current_fast:.2f} A."
                )
            else:
                msg = f"✅Батарея повністю заряджена."
                print(f"[MONITOR INFO] Battery fully charged. Unable to determine charge time.")
            await notify_all(msg)
            last_alert_charge = "charged"

        except Exception as e:
            print(f"[MONITOR ERROR] {e}")
            await notify_all(f"❗Monitoring error: {e}")

        await asyncio.sleep(CHECK_INTERVAL)


# === Direct command handlers ===
async def enable_cmd(update: Update, context: ContextTypes.DEFAULT_TYPE):
    """Handle /enable command (turn on notifications)."""
    user = update.effective_user
    if user.id not in allowed_users:
        await update.message.reply_text("☐ Please authorize first using /start.")
        return
```

```
    toggle_notifications(user.id, True)

    # --- Updated menu after enabling notifications ---
    keyboard = [
        ["Status"],
        ["Disable Notifications"],
        ["Logout"]
    ]
    reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)

    await update.message.reply_text(
        text="☐ Notifications enabled.",
        reply_markup=reply_markup
    )

    print(f"[CMD] /enable by {user.first_name} ({user.username}) [{user.id}] — notifications enabled")


async def disable_cmd(update: Update, context: ContextTypes.DEFAULT_TYPE):
    """Handle /disable command (turn off notifications)."""
    user = update.effective_user
    if user.id not in allowed_users:
        await update.message.reply_text("☐ Please authorize first using /start.")
        return

    toggle_notifications(user.id, False)

    # --- Updated menu after disabling notifications ---
```

```
    keyboard = [
        ["Status"],
        ["Enable Notifications"],
        ["Logout"]
    ]
    reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)

    await update.message.reply_text(
        text="☐ Notifications disabled.",
        reply_markup=reply_markup
    )

    print(f"[CMD] /disable by {user.first_name} ({user.username}) [{user.id}]")


async def logout_cmd(update: Update, context: ContextTypes.DEFAULT_TYPE):
    """Handle /logout command."""
    user = update.effective_user
    if user.id not in allowed_users:
        await update.message.reply_text("☐ You are not logged in.")
        return
    remove_user(user.id)
    allowed_users.remove(user.id)
    reply_markup = ReplyKeyboardMarkup([["Authorize"]], resize_keyboard=True)
    await update.message.reply_text(
        text="☐ You have been logged out.",
        reply_markup=reply_markup
    )
```

```
    print(f"[CMD] /logout by {user.first_name} ({user.username}) [{user.id}]")


# === MAIN ===
async def main():
    """Initialize the bot and start polling."""
    global allowed_users
    allowed_users = get_allowed_users()
    print(f"Loaded {len(allowed_users)} authorized users from DB.")

    app = Application.builder().token(BOT_TOKEN).build()

    # Set bot commands for Telegram UI suggestions
    await bot.set_my_commands([
        ("start", "Start or reauthorize"),
        ("status", "Show latest inverter data"),
        ("logout", "Logout and revoke access"),
        ("enable", "Enable notifications"),
        ("disable", "Disable notifications")
    ])
    print("[INIT] Telegram command menu registered.")

    # Handlers
    app.add_handler(CommandHandler("start", start))
    app.add_handler(MessageHandler(filters.TEXT & ~filters.COMMAND,
message_handler))
    app.add_handler(CommandHandler("status", status))
    app.add_handler(CommandHandler("enable", enable_cmd))
    app.add_handler(CommandHandler("disable", disable_cmd))
    app.add_handler(CommandHandler("logout", logout_cmd))
```

```
    # Start background monitor
    asyncio.create_task(monitor_task())

    print("Bot started...")
    await app.run_polling()


# === ENTRY POINT ===
if __name__ == "__main__":
    print("Bot starting...")

    # allow nested event loop reuse
    nest_asyncio.apply()
    loop = asyncio.get_event_loop()

    try:
        loop.create_task(main())
        loop.run_forever()
    except KeyboardInterrupt:
        print("\nBot stopped by user.")
    finally:
        pending = asyncio.all_tasks(loop)
        for task in pending:
            task.cancel()
        try:
            loop.run_until_complete(asyncio.gather(*pending,
return_exceptions=True))
        except Exception:
            pass
```

```
loop.close()
```