

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

магістр

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ
ВЕБЗАСТОСУНКУ ЩОДО ЗБОРУ, АНАЛІЗУ ТА ДИСПЕТЧЕРИЗАЦІЇ
ДАНИХ СПОЖИТОЇ ЕЛЕКТРОЕНЕРГІЇ
RESEARCH AND SOFTWARE IMPLEMENTATION OF A WEB
APPLICATION FOR COLLECTING, ANALYZING, AND MANAGING
CONSUMED ELECTRICITY DATA

Виконав: студент 2 курсу, групи КНТ-124м
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

ЩЕРБАКОВ К.Ю.

(ПРІЗВИЩЕ та ініціали)

Керівник КОЦУР М.І.

(ПРІЗВИЩЕ та ініціали)

Рецензент ПОЛЯКОВ М.О.

(ПРІЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти магістр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
“ ” 2025 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ЩЕРБАКОВА Кирила Юрійовича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження та програмна реалізація вебзастосунку
щодо збору, аналізу та диспетчеризації даних спожитої електроенергії
Research and Software Implementation of a Web Application for Collecting,
Analyzing, and Managing Consumed Electricity Data
керівник проєкту (роботи) к.т.н., доцент, КОЦУР Михайло Ігорович,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)
затверджені наказом закладу вищої освіти від “30” вересня 2025 року
№ 447

2. Строк подання студентом проєкту (роботи) 08 грудня 2025 року
3. Вихідні дані до проєкту (роботи) рекомендована література, технічне
завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Проєктування програмного
забезпечення 3. Конструювання програмного забезпечення. 4. Керівництво
користувача та приклад роботи. 5. Тестування та експериментальне дослідження
програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень,
кількість слайдів, плакатів)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	КОЦУР М.І., доцент		
Нормоконтроль	КАЛІНІНА М.В., асистент		

7. Дата видачі завдання « 30 » вересня 2025 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	2 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	3 тиждень	Розділ 2
4	Розробка архітектури програми.	4 тиждень	Розділ 3
5	Розробка програми.	5-7 тиждень	Розділ 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	8 тиждень	Розділ 5
7	Оформлення пояснювальної записки та документів до неї.	9-10 тиждень	Додатки
8	Нормоконтроль та рецензування.	11 тиждень	
9	Захист роботи.	12 тиждень	

Студент(ка)

Підпис Кирило ЩЕРБАКОВ
(підпис) (Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

(підпис)

Михайло КОЦУР
(Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:
112 с., 13 табл., 33 рис., 3 дод., 32 джерела.

СИСТЕМА ЗБОРУ ТА ВІЗУАЛІЗАЦІЇ ДАНИХ
ЕЛЕКТРОЛІЧИЛЬНИКА, ETHERNET, DLMS/COSEM, NODE.JS, VUE.JS.

Об'єкт дослідження – процес розроблення програмного забезпечення для збору та обробки даних у системах енергомоніторингу.

Предмет дослідження – алгоритми і програмні засоби автоматизованого зчитування даних електролічильників через Ethernet і передавання їх на сервер та у вебінтерфейс.

Метою роботи є підвищення швидкості опрацювання отриманих даних показів електролічильників за протоколом DLMS/COSEM шляхом розроблення програмного забезпечення, яке дозволить надавати користувачу доступ до актуальних та архівних значень.

Матеріали та методи дослідження включають об'єктно-орієнтоване програмування, модульний підхід, використання протоколу DLMS/COSEM, а також застосування середовища розробки Visual Studio Code, Node.js, Vue.js.

У роботі розроблено комплексну систему, що забезпечує автоматизоване зчитування параметрів електроспоживання, їх передачу через Ethernet та обробку на серверній частині, реалізованій засобами C# та Node.js. Дані зберігаються в базі Microsoft SQL Server і відображаються у клієнтському застосунку, створеному на Vue.js.

Переваги – висока точність збору інформації, зменшення ручних операцій, можливість масштабування та інтеграції з іншими приладами.

Рішення може застосовуватись у системах обліку та моніторингу енергоспоживання в житловій, промисловій і комунальній сферах.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 112 pages, 13 tables, 33 figures, 3 appendixes, 32 sources.

SYSTEM FOR ELECTRICITY METER DATA ACQUISITION AND VISUALIZATION, ETHERNET, DLMS/COSEM, NODE.JS, VUE.JS.

Object of the study is the process of software development for data collection and processing in energy monitoring systems.

Subject of the study is algorithms and software tools for automated reading of electricity meter data via Ethernet and transmitting them to a server and a web interface.

The aim of the work is to increase the speed of processing the obtained electricity meter readings using the DLMS/COSEM protocol by developing software that provides users with access to current and archived values.

Materials and research methods include object-oriented programming, a modular development approach, the DLMS/COSEM protocol, as well as the use of the Visual Studio Code development environment, Node.js, and Vue.js.

The work presents a comprehensive system that provides automated reading of electricity consumption parameters, their transfer over Ethernet, and processing on the server side implemented using C# and Node.js. The data is stored in a Microsoft SQL Server database and displayed in a client application developed with Vue.js.

Advantages of the system include high accuracy of data acquisition, reduction of manual operations, scalability, and the ability to integrate with other devices.

The solution can be applied in energy accounting and monitoring systems in residential, industrial, and public utility sectors.

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	8
Вступ.....	9
1 Аналіз предметної області.....	11
1.1 Види систем збору та аналізу даних споживання електроенергії.....	11
1.2 Підходи до створення систем збору та аналізу даних споживання електроенергії.....	12
1.3 Огляд програм	13
1.4 Стандарти передачі даних для збору показників лічильників електроенергії	16
1.5 Протокол обміну даними з лічильниками DLMS	24
1.6 Постановка задач дипломної роботи	25
2 Проектування програмного забезпечення	27
2.1 Аналіз вимог	27
2.2 Архітектура системи.....	27
2.3 Обґрунтування вибору програмних засобів	28
2.4 Обґрунтування вибору системи керування базами даних	31
2.5 Висновки за розділом 2	32
3 Конструювання програмного забезпечення	34
3.1 Схема алгоритму функціонування серверного додатку опитування лічильника.....	34
3.2 Особливості програмної реалізації додатку для опиту лічильника.....	35
3.3 Структура вебзастосунку	38
3.3 Опис бази даних	45
3.4 Висновки за розділом 3	48
4 Керівництво користувача та приклад роботи.....	50
4.1 Налаштування параметрів та запуск програми опитування.....	50
4.2 Розгортання вебзастосунку	51
4.3 Приклад роботи	55

4.4 Висновки за розділом 4	62
5 Тестування та еспериментальне дослідження програми	63
5.1 Тестування системи	63
5.2 Unit-тестування серверної частини	64
5.3 Аналіз впливу кроку дискретизації на точність розрахунку активної енергії	68
5.4 Висновки за розділом 5	71
Висновки	72
Перелік джерел посилань	74
Додаток А Технічне завдання	78
Додаток Б Текст програми	82
Додаток В Слайди презентації	106

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API	– Application Programming Interface;
COSEM	– Companion Specification for Energy Metering;
DLMS	– Device Language Message Specification;
DMS	– Distribution Management System;
EMS	– Energy Management System;
GSM	– Global System for Mobile Communications;
HTTP	– HyperText Transfer Protocol;
ISO	– International Organization for Standardization;
OBIS	– Object Identification System;
PLC	– Power Line Communication;
SCADA	– Supervisory Control and Data Acquisition;
SQL	– Structured Query Language;
TCP	– Transmission Control Protocol;
БД	– база даних;
СКБД	– система керування базами даних.

ВСТУП

Сучасні системи обліку електроенергії відіграють важливу роль у забезпеченні ефективного використання енергоресурсів, контролі споживання та підвищенні точності розрахунків між постачальниками та споживачами. Традиційні методи зчитування показників лічильників, що базуються на ручному зборі даних, мають суттєві недоліки — вони є трудомісткими, повільними та схильними до людських помилок. Тому актуальним напрямом розвитку є автоматизація процесу збору, обробки та збереження даних з енергетичних лічильників за допомогою сучасних програмно-апаратних рішень.

Такі системи дозволяють дистанційно опитувати лічильники через дротові або бездротові мережі, збирати показники у реальному часі, передавати їх на сервер для подальшої обробки та зберігання у базі даних. Використання таких технологій підвищує точність обліку, зменшує витрати на експлуатацію та забезпечує можливість моніторингу енергоспоживання у зручному інтерфейсі.

При створенні подібних систем розробник має вирішити низку ключових завдань, серед яких:

- забезпечення надійного з'єднання з лічильником і стабільної передачі даних;
- організація безпечного підключення до бази даних і збереження інформації;
- створення інтуїтивно зрозумілого інтерфейсу для відображення отриманих показників;
- забезпечення масштабованості та розширюваності системи.

Сучасні протоколи зв'язку, такі як DLMS/COSEM, у поєднанні з технологіями передачі даних дозволяють створювати гнучкі рішення для автоматизованого збору даних з лічильників, що можуть бути інтегровані у більші енергетичні системи або інформаційні платформи.

Використання таких рішень дозволяє не лише знизити витрати на ручне обслуговування, а й забезпечити постійний контроль параметрів споживання, виявлення несправностей і формування аналітичних звітів.

У даній дипломній роботі розглядається питання розробки програмно-апаратного комплексу для опитування електролічильника через Ethernet-з'єднання з передачею отриманих даних на сервер, де вони зберігаються у базі даних та відображаються через вебзастосунок.

Метою роботи є створення автоматизованої системи збору та моніторингу даних електролічильника, яка забезпечує стабільне з'єднання, зручне відображення інформації для користувача та можливість подальшого аналізу споживання електроенергії.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Види систем збору та аналізу даних споживання електроенергії

У сучасних умовах активного розвитку енергетики та цифрових технологій зростає потреба у системах, які забезпечують автоматизований облік, аналіз та диспетчеризацію даних о спожитій електроенергії. Такі системи дозволяють не лише точно вимірювати обсяги використаної енергії, а й оптимізувати процеси управління енергоспоживанням, виявляти відхилення та знижувати енергетичні втрати. На рис. 1.1 наведено основні види систем збору та аналізу даних електроенергії [1]–[3].



Рисунок 1.1 – Види систем безпеки [1]–[2]

Автоматизовані системи обліку електроенергії (АСКОЕ) – призначені для автоматичного зняття показників з електролічильників і передачі даних у єдину базу. Такі системи широко застосовуються енергопостачальними компаніями та промисловими підприємствами для комерційного обліку, контролю балансу енергоспоживання та виявлення несанкціонованого використання електроенергії [4]–[5].

Системи аналізу та моніторингу споживання – програмно-апаратні комплекси, які забезпечують збір великих масивів даних, їх обробку та подання у вигляді зручної аналітики. Завдяки таким системам можна відстежувати пікові навантаження, прогнозувати споживання та оптимізувати

роботу енергетичної інфраструктури. Сучасні рішення застосовують методи Big Data та штучного інтелекту для виявлення аномалій та формування рекомендацій [4]–[5].

Диспетчерські системи (SCADA, DMS) – забезпечують централізоване управління енергетичними об'єктами, дозволяють оперативно реагувати на аварійні ситуації та здійснювати керування потоками електроенергії. Такі системи є критично важливими для стабільної роботи розподільчих та магістральних мереж [5].

1.2 Підходи до створення систем збору та аналізу даних споживання електроенергії

Міжнародний стандарт ISO 50001 є основоположним документом у сфері енергоменеджменту, що встановлює вимоги до створення, впровадження, підтримки та вдосконалення системи енергетичного менеджменту. Його основна мета полягає у допомозі організаціям систематично підвищувати енергоефективність, зменшувати витрати на енергоресурси та скорочувати вплив на навколишнє середовище [6].

Застосування ISO 50001 ґрунтується на циклі Plan–Do–Check–Act, що дозволяє організаціям інтегрувати енергоменеджмент у загальну систему управління. Це означає, що процес енергоменеджменту не обмежується лише впровадженням окремих технічних рішень, а передбачає постійне планування, контроль і вдосконалення заходів у сфері енергоспоживання [6].

Першим кроком впровадження стандарту є визначення енергетичної політики та документування цілей і завдань у сфері енергоменеджменту. Важливою складовою є ідентифікація енергетично значущих об'єктів та процесів, що дозволяє сфокусувати ресурси на найважливіших напрямках підвищення ефективності [7].

Особливу увагу ISO 50001 приділяє моніторингу та аналізу даних. Організації мають налагодити процес регулярного вимірювання та збору

інформації про споживання електроенергії, визначення базових показників та оцінку ефективності впроваджених заходів. Це забезпечує об'єктивність прийняття управлінських рішень і дозволяє відслідковувати прогрес у досягненні цілей [7]–[8].

Ключовим аспектом стандарту є залучення персоналу. Працівники всіх рівнів повинні бути ознайомлені з енергетичною політикою підприємства та мати необхідну компетентність для її реалізації. Навчання та підвищення обізнаності сприяє формуванню культури енергозбереження в організації [7]–[8].

ISO 50001 також враховує питання відповідності законодавчим і нормативним вимогам. Організація зобов'язана відслідковувати чинні регуляторні документи та інтегрувати їх у власну систему енергетичного менеджменту, що знижує ризики штрафів та правових наслідків [7]–[8].

1.3 Огляд програм

1.3.1 АСКОЕ НЕК «Укренерго»

Автоматизована система комерційного обліку електроенергії (АСКОЕ), яка експлуатується НЕК «Укренерго», є центральним елементом ринку електроенергії України. Вона забезпечує збір, збереження, обробку та передачу даних про спожиту й передану електричну енергію з метою здійснення комерційних розрахунків. Розглянемо її можливості:

- централізований збір даних з електролічильників учасників ринку через канали зв'язку у єдину базу даних;
- передача даних у систему MMS (Market Management System) для забезпечення коректності розрахунків на ринку електроенергії;
- підтримка множинних точок обліку для генерації, споживання та перетоків між суміжними операторами системи розподілу;
- автоматичне формування балансів споживання і виробництва електроенергії та виявлення небалансів;

- забезпечення прозорості та уніфікації даних для учасників ринку: трейдерів, постачальників, операторів розподілу;
- можливість інтеграції з системами диспетчеризації (SCADA/EMS) для оперативного управління режимами енергосистеми;
- багаторівневий захист інформації та контроль доступу, що знижує ризики викривлення даних [9].

Попри ключову роль АСКОЕ у функціонуванні ринку електроенергії України, система має й певні недоліки:

- значна частина лічильників та комунікаційного обладнання потребує модернізації для підвищення точності та надійності;
- інтеграція даних з різних систем обліку учасників ринку іноді ускладнена через різні стандарти та технічні рішення;
- висока залежність від надійності каналів зв'язку, що у випадку аварій або кібератак може вплинути на розрахунки;
- складність експлуатації та необхідність постійної підтримки й оновлення програмного забезпечення;
- орієнтація на масштаб національної енергосистеми, тому локальні підприємства не можуть напряду користуватися функціоналом АСКОЕ без адаптації [9].

1.3.2 Siemens Spectrum Power

Spectrum Power від компанії Siemens – це сучасна платформа для управління енергосистемами, яка включає функціонал SCADA, EMS та DMS. Вона використовується для диспетчеризації, прогнозування і моніторингу великих електромереж, у тому числі інтегрованих з відновлюваними джерелами енергії.

Можливості:

- централізований моніторинг і управління режимами роботи енергосистеми у реальному часі;

- інтеграція SCADA , EMS та DMS в єдину платформу;
- моделювання енергосистеми та прогнозування навантажень із використанням математичних моделей;
- підтримка Smart Grid-функцій (інтеграція відновлюваних джерел, робота з мікромережами);
- розвинуті можливості кіберзахисту та контролю доступу;
- масштабованість для управління як локальними, так і національними енергосистемами [10].

Недоліки:

- система досить дорога у впровадженні та обслуговуванні;
- складний процес інтеграції із застарілою інфраструктурою;
- потребує висококваліфікованого персоналу для адміністрування та налаштування;
- орієнтована переважно на великих операторів систем передачі та розподілу, а не на малий бізнес чи окремі підприємства [10].

1.3.3 EcoStruxure Power Monitoring Expert

EcoStruxure Power Monitoring Expert (PME) від Schneider Electric – це програмне забезпечення для збору, аналізу та диспетчеризації даних енергоспоживання, орієнтоване на підприємства, промислові об’єкти та операторів розподілу.

Можливості:

- збір і збереження даних з електролічильників, сенсорів та обладнання Schneider та інших виробників;
- надання детальних звітів і візуалізацій для аналізу енергоспоживання та оптимізації витрат;
- підтримка енергетичного менеджменту відповідно до стандарту ISO 50001;

- інструменти для виявлення аномалій у споживанні електроенергії, що можуть свідчити про втрати або аварійні режими;
- можливість інтеграції з системами BMS (Building Management Systems) і промисловими IoT-платформами;
- хмарний і локальний варіанти розгортання з віддаленим доступом [11].

Недоліки:

- комерційне ПЗ, вартість залежить від кількості підключених точок обліку;
- найкраще працює з обладнанням Schneider, інтеграція з приладами інших виробників може вимагати додаткових налаштувань;
- у випадку хмарного варіанту потрібне стабільне інтернет-з'єднання;
- інтерфейс із великою кількістю функцій може бути складним для нових користувачів [11].

Порівняльну характеристику програмного забезпечення для енергоменеджменту наведено у табл. 1.1.

1.4 Стандарти передачі даних для збору показників лічильників електроенергії

У сучасних системах обліку енергоресурсів важливу роль відіграють інтерфейси обміну даними, які забезпечують надійний та стабільний зв'язок між лічильниками електроенергії та пристроями збору й обробки інформації. Такі інтерфейси дозволяють автоматизувати процес отримання показників, що значно зменшує ризик помилок під час ручного зняття даних і створює можливість для побудови інтегрованих систем моніторингу.

До найбільш поширених методів дротової та бездротової комунікації у цій сфері належать RS-232, RS-485, Ethernet, GSM-зв'язок, а також технологія PLC. Вони забезпечують передачу даних між лічильниками та верхнім рівнем

системи (сервером або контролером) і відрізняються особливостями застосування та технічними характеристиками [12]–[17].

Таблиця 1.1 – Порівняння програмного забезпечення для енергоменеджменту [9]–[11]

Критерій порівняння	Назва програми		
	АСКОЕ НЕК «Укренерго»	Siemens Spectrum Power	Schneider EcoStruxure PME
Призначення	Комерційний облік, балансування	Диспетчеризація	Аналіз споживання
Сумісність з обладнанням	+ - (лічильники стандартизованих типів)	+ (різні виробники, Smart Grid)	+ - (краще з обладнанням Schneider)
Інтеграція	+ (MMS, SCADA)	+ (Smart Grid, мікромережі)	+ (BMS, IoT, прилади різних виробників)
Вимоги до системи	Високі (мережа, сервери)	Дуже високі (потужні сервери, резервування)	Середні (може працювати локально чи в хмарі)
Вартість впровадження	Висока	Дуже висока	Середня
Віддалений доступ	+ - (через диспетчерські центри)	+ (повний віддалений моніторинг)	+ (хмара, мобільний доступ)
Гнучкість налаштувань	Низька	Висока	Висока

RS-232 – це один із найстаріших і водночас найпоширеніших стандартів послідовної передачі даних. Він застосовується для підключення

лічильників безпосередньо до комп'ютера або модему на невеликій відстані (до 15 метрів). Стандарт характеризується простотою реалізації та невисокою вартістю обладнання. Проте основним недоліком є обмежена дальність передачі сигналу та невисока завадостійкість [12].

RS-485 – це вдосконалений стандарт, який вирішує проблеми обмеженої відстані та кількості підключених пристроїв. На відміну від RS-232, він дозволяє організувати багатоточкові мережі (до 32 пристроїв на одній шині) та забезпечує передачу даних на відстань до 1200 метрів. Завдяки використанню диференціальної передачі сигналу RS-485 має високу стійкість до електромагнітних завад, що робить його оптимальним рішенням для промислових систем збору даних із лічильників електроенергії [13].

Ethernet – це один із найпоширеніших і найефективніших дротових інтерфейсів передачі даних, який активно використовується у сучасних системах обліку енергоресурсів. Його застосування дозволяє забезпечити високу швидкість та надійність обміну інформацією між лічильниками, контролерами та серверними системами. Завдяки підтримці стандартних мережевих протоколів (TCP/IP) Ethernet легко інтегрується у корпоративні та промислові локальні мережі, що спрощує централізований збір, обробку та передачу даних у реальному часі. Перевагами Ethernet є велика пропускна здатність, можливість підключення значної кількості пристроїв до однієї мережі, висока завадостійкість і гнучкість масштабування системи. Основним недоліком цього типу зв'язку є потреба у прокладанні кабельної інфраструктури та залежність від наявності мережевого обладнання, що може підвищувати вартість впровадження системи [14].

GSM-зв'язок застосовується у випадках, коли необхідно організувати дистанційний доступ до лічильників, розташованих на значній відстані від серверів збору даних. Використання мобільних мереж дозволяє передавати показники лічильників практично з будь-якої точки, де є покриття оператора мобільного зв'язку. Це робить GSM особливо корисним для віддалених об'єктів, де прокладання кабельних ліній економічно недоцільне. Основні

переваги GSM – глобальне покриття та можливість інтеграції із сучасними системами диспетчеризації та «розумними» мережами [15]–[16].

PLC технологія передачі даних по існуючих електричних мережах. Вона дозволяє організовувати зв'язок між лічильниками та центром збору даних без додаткових кабелів, використовуючи наявну силову інфраструктуру. Перевагою PLC є простота впровадження у багатоповерхових будинках та розгалужених мережах, де прокладання додаткових комунікацій ускладнене або небажане. Проте якість сигналу залежить від стану електромережі – високі навантаження, імпульсні перешкоди та старі кабелі можуть суттєво погіршувати стабільність передачі. Через це PLC частіше застосовується у побутових AMR/AMI-системах, ніж у промислових [17].

Таблиця 1.2 – Стандарти передачі даних для збору показників лічильників електроенергії [12]–[17]

Характеристика	RS-232	RS-485	Ethernet	PLC	GPRS
1	2	3	4	5	6
Дальність передачі	До 15 м	До 1200 м	до 100 м (без повторювачів)	До кількох км у мережі живлення	Обмежена зоною покриття
Тип з'єднання	Точка-точка	Багатоточкове (до 32 пристроїв)	Дротове, мережеве	Через електромережу	Через мобільну мережу
Швидкість передачі	До 115,2 Кбіт/с	До 10 Мбіт/с	до 1 Гбіт/с і більше	Від кількох до сотень Кбіт/с	До 85,6 Кбіт/с

Продовження таблиці 1.2

1	2	3	4	5	6
Завадостійкість	Низька	Висока (диференціальна передача)	Висока	Середня/ залежить від стану мережі	Середня, залежить від мережі
Основні переваги	Простота, низька вартість	Велика відстань, надійність	Висока швидкість, інтеграція з TCP/IP	Використання існуючої мережі	Дистанційність, глобальне покриття

Порівняння різних інтерфейсів передачі даних показує, що кожен із них має свої переваги та сфери оптимального застосування. RS-232 підходить для простих локальних підключень, RS-485 – для промислових багатоточкових мереж, GSM – для віддалених об’єктів, а PLC забезпечує економічну передачу даних по існуючих електромережах [12]–[17].

Однак найбільш ефективним і перспективним рішенням для сучасних систем обліку енергоресурсів є Ethernet, оскільки він поєднує високу швидкість, надійність, масштабованість, сумісність із мережевими протоколами та підтримку інтегрованих рішень у реальному часі. Саме тому Ethernet сьогодні вважається оптимальним вибором для побудови сучасних комплексних систем збору та моніторингу енергоспоживання [12]–[17].

1.4.1 Дані, що передаються лічильниками електроенергії

У системах автоматизованого обліку електроенергії лічильники виконують не лише функцію реєстрації показників, а й забезпечують передачу великого обсягу інформації, необхідної для аналізу та контролю стану електричних мереж. Сучасні прилади здатні збирати та передавати дані різних типів від базових облікових параметрів до розширених характеристик якості електроенергії та журналів подій. Обсяг і склад цих даних визначають

ефективність роботи всієї системи моніторингу та можливість інтеграції з диспетчерськими платформами [18].

Основою інформації, що передається лічильниками, є облікові дані. До них належать величини активної та реактивної енергії, зареєстровані приладом за певні періоди, які можуть враховуватися як у загальному вигляді, так і окремо за тарифними зонами. Такі показники є ключовими для формування рахунків, визначення фактичного споживання та оперативного контролю витрат електроенергії [18].

Окрім облікових значень, лічильники передають поточні вимірювальні параметри електричної мережі, зокрема значення напруги та струму на кожній фазі, активної, реактивної та повної потужності, коефіцієнта потужності та частоти. Ця інформація дозволяє оцінювати режим роботи споживача в реальному часі, виявляти перевантаження, дисбаланс фаз чи нестандартні режими, що є важливими для підтримання стабільності та надійності електропостачання [19].

Важливу роль у сучасних системах моніторингу відіграють так звані профілі навантаження дані, які лічильник накопичує з певним інтервалом упродовж доби. Така інформація містить деталізовану історію змін споживання, що дозволяє аналізувати пікові періоди, здійснювати прогнозування, будувати графіки навантажень та визначати нехарактерну поведінку споживачів. Профілі навантаження значно підвищують аналітичні можливості системи порівняно з простим зчитуванням показників [19]–[20].

Додатково передаються сервісні та технічні параметри: дата та час пристрою, серійний номер, версія програмного забезпечення, рівень сигналу (для GSM чи PLC), стан батареї та інші допоміжні характеристики. Вони забезпечують коректну ідентифікацію лічильника та підтримку його працездатності [19]–[20].

Таким чином, лічильники електроенергії передають комплексну інформацію, яка охоплює облікові показники, оперативні параметри режиму, профілі навантаження та сервісні відомості. Завдяки цьому системи обліку

трансформуються із засобів звичайного зчитування показників у повноцінні інформаційно-аналітичні комплекси, здатні забезпечувати глибоку діагностику стану електромереж та ефективний контроль енергоспоживання [20].

1.4.2 Аналіз методів обробки інформації, які застосовуються у програмних застосунках

У сучасних системах енергоменеджменту — таких як АСКОВ НЕК «Укренерго», Siemens Spectrum Power та EcoStruxure Power Monitoring Expert використовується широкий спектр дискретних і статистичних методів обробки інформації. Їхня мета полягає у точному зборі, очищенні, узгодженні та подальшому використанні даних для розрахунків, моніторингу й оптимізації роботи енергосистем. Усі ці системи працюють з дискретними даними, отриманими у визначені моменти часу, тому перша стадія обробки це перетворення первинних вимірювань у структуровані набори значень, що можуть бути збережені в архівах і використані для подальших операцій [21]–[22].

Зібрані дані проходять етапи фільтрації та контролю коректності, поширено використовуються методи виявлення пропусків вимірювань, некоректних або аномальних значень, що можуть виникати внаслідок помилок вимірювального обладнання чи проблем зі зв'язком. У таких випадках застосовуються алгоритми дискретної інтерполяції або реконструкції даних, що дозволяють частково відновити відсутні показники. У системах комерційного обліку, таких як АСКОВ, використовуються стандартизовані процедури валідації, редагування та заміщення даних що гарантує точність розрахунків [21]–[22].

Після очищення дані піддаються агрегуванню, часові ряди об'єднуються у більші інтервали від хвилин до годин чи доби, залежно від потреб користувача та регламентів. Це дає змогу зменшити обсяг інформації і

підвищити ефективність обчислень, зберігаючи при цьому характер змін навантаження або виробітку. У складних системах прогнозування застосовуються статистичні моделі та методи дискретної апроксимації, що дозволяє будувати коротко та середньострокові прогнози навантаження. Застосовуються алгоритми регресії, кореляційного аналізу, а також методи частотного аналізу, що дає можливість виявляти циклічні та періодичні компоненти навантаження [22].

На етапі аналізу споживання енергії програмні засоби виконують метрики енергетичного менеджменту, розрахунок коефіцієнтів потужності, визначення втрат, формування енергетичних балансів. Для швидкого розуміння стану системи широко застосовується дискретна фільтрація, зокрема ковзне середнє або експоненціальне згладжування, які пом'якшують різкі коливання та роблять графіки зручнішими для сприйняття. Алгоритми виявлення аномалій дозволяють визначати відхилення у споживанні, що можуть бути наслідком аварійних режимів, помилок обладнання або несанкціонованого підключення [22]–[23].

Фінальним етапом обробки є представлення результатів у вигляді звітів, діаграм і дашбордів. Усі три розглянуті системи мають розвинуті механізми візуалізації, де часові ряди даних перетворюються у графіки, гістограми та теплові карти. Оброблені значення експортуються до SCADA/EMS систем (як у Siemens Spectrum Power) або передаються до ринкових платформ, як MMS у випадку АСКОЕ. Таким чином, дискретні методи обробки даних забезпечують можливість від оперативного моніторингу до стратегічного планування та комерційних розрахунків [22]–[23].

Загалом, у сучасних програмних комплексах для енергетики методи обробки інформації відіграють ключову роль, оскільки саме від якості та точності цифрових даних залежить коректність ринкових операцій, стабільність режимів роботи енергосистеми та ефективність управління ресурсами [22]–[23].

1.5 Протокол обміну даними з лічильниками DLMS

Одним із найпоширеніших міжнародних стандартів обміну даними між приладами обліку енергоресурсів є DLMS/COSEM (Device Language Message Specification / Companion Specification for Energy Metering). Цей стандарт визначає структуру даних, способи адресації, механізми безпеки та протоколи комунікації, що забезпечують уніфіковану взаємодію між лічильниками різних виробників і системами збору даних [24]–[25].

Стандарт DLMS/COSEM розроблено та підтримується організацією DLMS User Association і затверджено Міжнародною електротехнічною комісією (IEC) як серія стандартів IEC 62056. Основною метою його створення є забезпечення сумісності між різними типами приладів обліку електричними, водяними, газовими або тепловими та програмно-апаратними системами збору, обробки й передавання показників [24]–[25].

Архітектура DLMS/COSEM базується на двох основних складових:

COSEM (Companion Specification for Energy Metering) — описує модель даних і об'єктну структуру лічильника. Вона визначає, яким чином організовані дані про споживання енергоресурсів, параметри стану пристрою, журнал подій та інші [24]–[25].

DLMS (Device Language Message Specification) — описує протокол обміну повідомленнями між клієнтом (наприклад, системою диспетчеризації) та сервером. Він визначає послідовність запитів і відповідей, а також рівні автентифікації та шифрування даних [24]–[25].

DLMS/COSEM підтримує кілька типів транспортних рівнів комунікації: HDLC (High-Level Data Link Control) для послідовних інтерфейсів (RS-232, RS-485), TCP/IP для Ethernet-з'єднань, а також бездротові протоколи, такі як GPRS. Завдяки цьому стандарт може бути реалізований у системах різних масштабів від локального зчитування даних до централізованих енергетичних платформ [24]–[25].

Важливою особливістю DLMS/COSEM є підтримка кількох рівнів безпеки, що забезпечують захист переданих даних від несанкціонованого доступу. Залежно від вимог системи можуть застосовуватися режими автентифікації None, Low, High, або High GMAC (із криптографічним захистом) [24]–[25].

Стандарт також передбачає механізм OBIS-кодів (Object Identification System) унікальних ідентифікаторів, які описують усі доступні параметри лічильника. OBIS-коди забезпечують універсальність у трактуванні даних, що дозволяє системам збору інформації правильно інтерпретувати показники незалежно від виробника пристрою [24]–[25].

Завдяки своїй гнучкості, масштабованості та відкритій структурі, DLMS/COSEM став базовим стандартом у галузі інтелектуального обліку енергоресурсів. Його використання дає змогу підвищити сумісність між пристроями, спростити інтеграцію в інформаційно-аналітичні системи та забезпечити достовірність і безпечність передавання вимірювальних даних [24]–[25].

1.6 Постановка задач дипломної роботи

За результатами проведеного аналізу можна зробити висновок, що в сучасних умовах існує велика кількість рішень для автоматизації систем обліку енергоресурсів. Проте більшість із них є дорогавартісними, потребують спеціалізованого обладнання або не забезпечують гнучкості інтеграції з іншими системами.

Тому розробка програмного комплексу для збору та передачі даних з лічильників електроенергії до бази даних з подальшою диспетчеризацією є актуальною задачею, оскільки вона дозволяє підвищити ефективність моніторингу, зменшити людський фактор і забезпечити своєчасний аналіз споживання енергії.

Метою роботи є підвищення швидкості опрацювання отриманих даних показів електrolічильників за протоколом DLMS/COSEM шляхом розроблення програмного забезпечення, яке дозволить надавати користувачу доступ до актуальних та архівних значень.

Для досягнення поставленої мети в роботі необхідно виконати такі задачі:

- провести аналіз існуючих програмно-апаратних засобів для автоматизації збору даних з лічильників електроенергії;
- спроектувати архітектуру програмно-апаратного комплексу;
- розробити програмне забезпечення для прийому, обробки та передачі даних;
- здійснити тестування розробленого програмного забезпечення та перевірити його ефективності.

2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Аналіз вимог

За проведеним аналізом предметної області було встановлено яким вимогам має відповідати програмно-апаратне рішення.

Функціональні вимоги:

- автоматичне опитування лічильника та запис отриманої відповіді у БД;
- можливість передивлятися миттєві значення з лічильника;
- можливість встановлювати порогові значення споживання;
- попередження при виходу споживання за порогові значення;
- можливість передивлятися архівні значення;
- можливість встановлювати план споживання на день;
- можливість розподілити план споживання погодинно;
- попередження при виході споживання за планове значення.

2.2 Архітектура системи

Архітектура системи була створена на основі аналізу вимог та зображена на рис. 2.1.

Вона включає три основні компоненти:

- лічильник;
- сервер;
- комп'ютер диспетчера.

Лічильник повинен знаходитися у одній локальній мережі що і сервер, використовується для отримання значень споживаної електроенергії.

Основною частиною є сервер на якому буде знаходитися ПЗ для опитування лічильника, база даних та вебзастосунок, ПЗ буде відправляти запити до лічильника, отриману відповідь записувати до БД, після чого через API по запиту отриманні дані будуть відображатись у вебзастосунку, з якого

також можна буде міняти порогові значення та налаштовувати план споживання.

Комп'ютер на якому буде відкрита вебсторінка з якою буде працювати диспетчер.

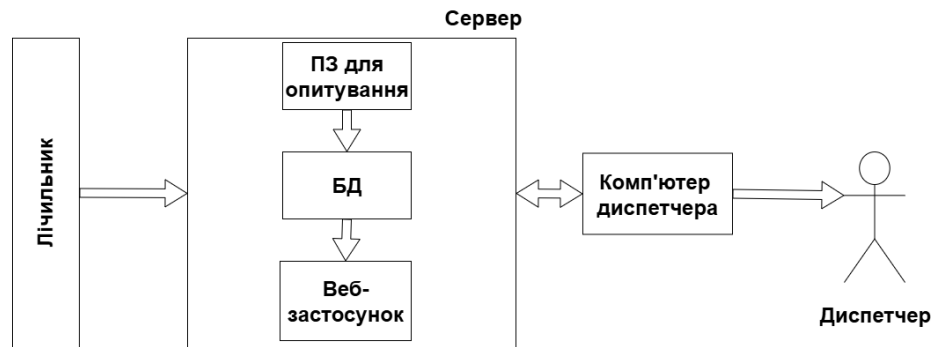


Рисунок 2.1 – Узагальнена архітектура системи

2.3 Обґрунтування вибору програмних засобів

2.3.1 Обґрунтування вибору мови програмування

Для розробки програмної частини системи збору даних з лічильників було обрано мову програмування C#.

Мова C# є сучасною, потужною та універсальною мовою, що поєднує продуктивність із простотою розробки. Вона забезпечує високий рівень надійності, зручність у створенні масштабованих рішень і широкі можливості об'єктно-орієнтованого програмування. Саме ці характеристики роблять її доцільною для створення прикладного програмного забезпечення, яке взаємодіє з обладнанням – у даному випадку, з лічильниками енергоресурсів [26]–[27].

C# має розвинену екосистему бібліотек і фреймворків для роботи з послідовними інтерфейсами (RS-232, RS-485) та мережевими з'єднаннями (Ethernet, TCP/IP). Завдяки бібліотеці System.IO.Ports і можливостям .NET Networking API можна реалізувати швидкий та стабільний обмін даними між програмою та лічильниками [26]–[27].

Важливою перевагою є підтримка багатопотоковості та асинхронного програмування (async/await), що дозволяє одночасно опитувати кілька пристроїв, обробляти отримані показники та оновлювати інтерфейс користувача без зниження швидкодії [26]–[27].

Мова C# також підтримує створення графічного інтерфейсу користувача за допомогою технологій Windows Presentation Foundation (WPF) або Windows Forms, що дозволяє створити зручний і сучасний інтерфейс для моніторингу даних. Обґрунтування вибору мови програмування наведено у табл. 2.1.

Таблиця 2.1 – Обґрунтування вибору мови програмування [26]–[27]

Критерій порівняння мов програмування	Назва мови		
	Python	C#	C++
Швидкодія	+-	+-	+
Доступ до апаратних ресурсів (послідовні порти, мережа)	+-	+	+
Кросплатформність	+	+-	+
Наявність бібліотек для взаємодії з лічильниками	+-	+	+
Оптимальність для промислових застосувань	+-	+	+
Зручність розробки та підтримки коду	+-	+	-

Таким чином, мова програмування C# була обрана завдяки поєднанню високої продуктивності, розвинених інструментів для роботи з апаратним забезпеченням, підтримці асинхронної та багатопотокової обробки. Це робить C# оптимальним вибором для розробки надійного, швидкодіючого та функціонального програмного забезпечення збору й обробки показників з лічильників енергоресурсів.

2.3.2 Обґрунтування вибору мови програмування для вебзастосунка

Для реалізації вебзастосунку, що здійснює диспетчеризацію та візуалізацію даних, отриманих із лічильників, було обрано стек технологій Node.js (з фреймворком Express.js) для серверної частини та Vue.js для клієнтської частини [28]–[29].

Таке поєднання технологій забезпечує високу швидкодію, масштабованість та зручність інтеграції з базами даних і зовнішніми пристроями збору інформації [28]–[29].

Node.js — це середовище виконання JavaScript на стороні сервера, яке дозволяє створювати високопродуктивні асинхронні застосунки [28]–[29].

Для розробки серверної частини застосунку використано фреймворк Express.js, який є мінімалістичним, але потужним інструментом для побудови REST API [28]–[29].

Express.js забезпечує зручну маршрутизацію запитів та легку інтеграцію з базами даних, що робить його оптимальним вибором для серверів збору та диспетчеризації даних у реальному часі [28]–[29].

Vue.js — це прогресивний фреймворк для створення реактивних користувацьких інтерфейсів. Його головними перевагами є висока швидкість розробки, простота структури коду.

Це дозволяє реалізувати інтерфейс, у якому дані з лічильників оновлюються в реальному часі без перезавантаження сторінки, що є критично важливим для систем моніторингу та диспетчеризації [30].

Комбінація Node.js та Vue.js утворює повноцінну архітектуру типу «клієнт–сервер», у якій:

- Node.js відповідають за логіку збору, обробки та зберігання даних;
- Vue.js за візуалізацію інформації, побудову графіків і відображення поточного стану системи.

Така архітектура легко масштабується, підтримує обмін даними через REST API, а також дозволяє без складнощів додавати нові функціональні модулі, наприклад, аналітику або систему сповіщень [30].

Обґрунтування вибору технологій для розробки вебзастосунку наведено у табл. 2.2 [28]–[30].

Таблиця 2.2 – Обґрунтування вибору технологій для розробки вебзастосунку [28]–[30]

Критерій порівняння мов програмування	Назва мови		
	Node.js + Vue.js	Django + React	ASP.NET Core + Blazor
Швидкодія	+	+-	+
Масштабованість	+	+	+
Реактивний інтерфейс	+	+-	+-
Швидкість розробки	+	+-	+-
Можливості розширення	+	+	+-

2.4 Обґрунтування вибору системи керування базами даних

Для зберігання, обробки та аналітики даних, отриманих із лічильників, у проєкті було обрано систему керування базами даних Microsoft SQL Server.

Вибір саме цієї СКБД обумовлений її високою продуктивністю, стабільністю роботи у промислових умовах та зручною інтеграцією із серверними застосунками на Node.js [31]–[32].

MSSQL є реляційною системою керування базами даних, розробленою компанією Microsoft, яка забезпечує надійне зберігання великих обсягів даних і підтримку складних запитів мовою SQL [31]–[32].

Завдяки оптимізованому механізму обробки транзакцій MSSQL дозволяє ефективно працювати з даними у реальному часі, що є критично важливим для систем диспетчеризації та моніторингу [31]–[32].

Однією з ключових переваг MSSQL є розвинена система безпеки, що включає аутентифікацію користувачів, шифрування даних і контроль доступу на рівні об'єктів бази [31]–[32].

Це дозволяє гарантувати конфіденційність даних, отриманих від лічильників, та відповідність вимогам сучасних стандартів захисту інформації [31]–[32].

Бібліотеки, такі як mssql для Node.js, забезпечують простий обмін даними між сервером і базою, підтримку асинхронних запитів, транзакцій та збережених процедур, що полегшує реалізацію логіки на стороні сервера [31]–[32].

Обґрунтування вибору бази даних 2.3 [31]–[32].

Таблиця 2.3 – Обґрунтування вибору СКБД [17]

Критерій порівняння мов програмування	Назва СКБД		
	MSSQL	MySQL	PostgreSQL
Швидкодія	+	+-	+
Безпека та контроль доступу	+	+-	+
Можливості аналітики	+	-	+-
Підтримка корпоративного рівня	+	+-	+-
Інтеграція з Node.js	+	+	+

2.5 Висновки за розділом 2

Для реалізації серверної частини системи збору та диспетчеризації даних з лічильників було обрано мову програмування C#, яка забезпечує високу продуктивність, надійність та зручність роботи з мережевими протоколами. Завдяки підтримці асинхронного обміну даними та широким можливостям для інтеграції з апаратними пристроями, C# є оптимальним вибором для створення модулів опитування лічильників і передачі отриманої інформації на сервер.

Для зберігання та обробки даних використано систему керування базами даних Microsoft SQL Server, що відзначається стабільністю, високим рівнем безпеки. MSSQL забезпечує надійне зберігання великого обсягу даних, підтримує транзакції, аналітичні запити та легко масштабується при розширенні систем.

Клієнтська частина буде реалізована із використанням Vue.js прогресивного фреймворку для створення динамічних вебінтерфейсів, що дозволяє відображати інформацію в реальному часі, та Node.js, який забезпечує обробку запитів і взаємодію між клієнтом та сервером. Таке поєднання технологій утворює сучасну архітектуру типу «клієнт–сервер», що гарантує стабільну роботу системи, гнучкість у розширенні функціональності та зручність користувацької взаємодії.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Схема алгоритму функціонування серверного додатку опитування лічильника

На рис. 3.1 наведено узагальнену схему алгоритму додатку опитування лічильника.

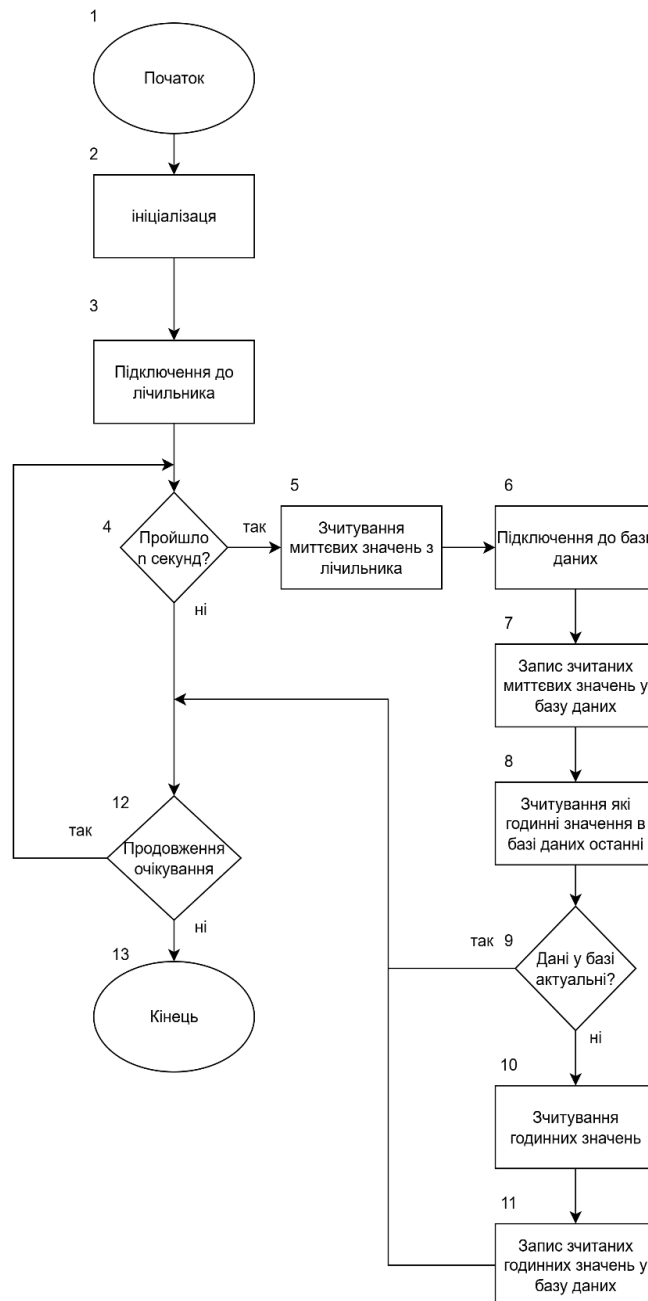


Рисунок 3.1 – Узагальнений алгоритм функціонування додатку опитування лічильника

Блок 1. Початок.

Блок 2. Встановлення значення змінних підключення бібліотек.

Блок 3. Підключення до лічильника енергоспоживання по TCP за протоколом DLMS.

Блок 4. Чи пройшов інтервал часу для наступного опитування лічильника, де n кількість секунд, у випадку якщо пройшов перехід до блоку 5 інакше до 12.

Блок 5. Зчитування миттєвих значень з лічильника за протоколом DLMS запис отриманих даних до змінних.

Блок 6. Підключення до MSSQL БД.

Блок 7. Записи даних до БД SQL запитом update з підставкою змінних з отриманими даними з лічильника.

Блок 8. SQL запит до БД для отримання останнього запису годинних значень які є у БД.

Блок 9. Перевірка чи актуальні значення у БД у випадку розбіжності з поточним часом переходу до блоку 10, інакше до 12.

Блок 10. Зчитування годинних значень з лічильника за протоколом DLMS запис отриманих даних до змінних.

Блок 11. Записи даних до БД SQL запитом insert з підставкою змінних з отриманими даними з лічильника.

Блок 12. Продовжити очікування інтервалу для наступного опитування, у випадку продовження перехід до блоку 4, інакше до 13.

Блок 13. Кінець.

3.2 Особливості програмної реалізації додатку для опиту лічильника

Текст програми наведено в Додатку Б. У програмі оголошено декілька змінних, які відповідають за параметри підключення: ipAddress, port, clientAddress, serverAddress, authentication, password, які використовуються для

встановлення з'єднання з лічильником електроенергії за протоколом DLMS/COSEM через TCP-інтерфейс.

Також змінні `connectionString`, `tableName` відповідають за підключення до бази даних Microsoft SQL Server та збереження отриманих результатів у таблиці з показниками споживання.

Основною функцією програми є `ReadMeterData()`. Вона викликається після настання інтервалу опитування, визначеного таймером, та виконує основні дії з опитування лічильника, отримання поточних і архівних даних та їх подальший запис до БД.

З'єднання з лічильником виконується за допомогою бібліотеки `Gurux.DLMS`, яка реалізує стандартний протокол DLMS/COSEM.

Для цього створюється клієнт `GXDLMSClient`, який встановлює зв'язок із лічильником за IP-адресою та портом. Після встановлення сеансу автентифікації проводиться обмін DLMS-кадрами для отримання даних.

Отримання значень відбувається через зчитування об'єктів типу `GXDLMSObject`, за допомогою методу `Read()`, який повертає поточне значення з лічильника.

Дані, отримані з лічильника, зберігаються у проміжні змінні, після чого програма підключається до бази даних MSSQL за допомогою бібліотеки `System.Data.SqlClient`.

Для оновлення поточних значень у таблиці виконується SQL-запит `UPDATE`, у якому замінюються відповідні поля на свіжі значення з лічильника.

Для збереження погодинних або архівних даних застосовується запит `INSERT`, який додає новий рядок у таблицю з відміткою часу та параметрами споживання.

У програмі реалізовано перевірку актуальності даних. Для цього виконується SQL-запит, який отримує останній запис з таблиці годинних значень. Якщо час останнього запису відрізняється від поточного на більше

ніж одну годину, викликається функція ReadHourlyData(), яка читає архівні значення з лічильника і додає їх у базу даних.

Таким чином, програма забезпечує автоматичний обмін даними з лічильником електроенергії, оновлення поточних показників і збереження архівних значень для подальшої аналітики.

Опис використаних бібліотек, що застосовуються у проєкті, наведено в табл. 3.1, а опис основних функцій програми – у табл. 3.2.

Таблиця 3.1 – Опис використаних бібліотек додатку опитування

Бібліотека	Опис бібліотеки
Gurux.DLMS.Enums.h	Містить перелік типів даних, методів аутентифікації, типів інтерфейсів (HDLC, TCP/IP) та інших параметрів, необхідних для взаємодії з пристроями за протоколом DLMS/COSEM.
Gurux.DLMS.Objects.h	Надає класи для представлення об'єктів DLMS (наприклад, Register, Data, Clock), що дозволяють зчитувати та змінювати параметри лічильника.
Gurux.DLMS.Reader.h	Забезпечує функції для встановлення зв'язку, ініціалізації сесії та читання/запису даних між клієнтом і пристроєм.
Gurux.Net.h	Надає класи для роботи з TCP/IP-з'єднаннями, зокрема клас GXNet, який використовується для створення клієнта TCP.
Gurux.DLMS.Secure.h	Містить реалізацію безпечного клієнта GXDLMSSecureClient, який підтримує різні рівні аутентифікації та шифрування.
System.h	Базова бібліотека .NET, що забезпечує доступ до стандартних типів даних і функцій

Таблиця 3.2 – Опис використаних базових функцій додатку опитування

Функція	Опис функції
media.Open()	Відкриває мережеве з'єднання (TCP) між клієнтом і лічильником.
reader.InitializeConnection()	Ініціалізує DLMS-сесію, виконує обмін кадрами асоціації між клієнтом і пристроєм.
GXDLMSClient.CreateObject(ObjectType.Register)	Створює DLMS-об'єкт типу «Register» для зчитування числових значень із лічильника.
reader.Read(obj, attributeIndex)	Зчитує вказаний атрибут об'єкта (наприклад, значення лічильника) за його логічною адресою.
reader.Close()	Завершує DLMS-сесію, виконує процедуру від'єднання від пристрою.
media.Close()	Закриває TCP-з'єднання після завершення роботи програми.

3.3 Структура вебзастосунку

3.3.1 Функціонал вебзастосунку

На рис. 3.2 наведено діаграму прецедентів, де відображено основний функціонал застосунку для перегляду, аналізу та обробки даних електролічильників.

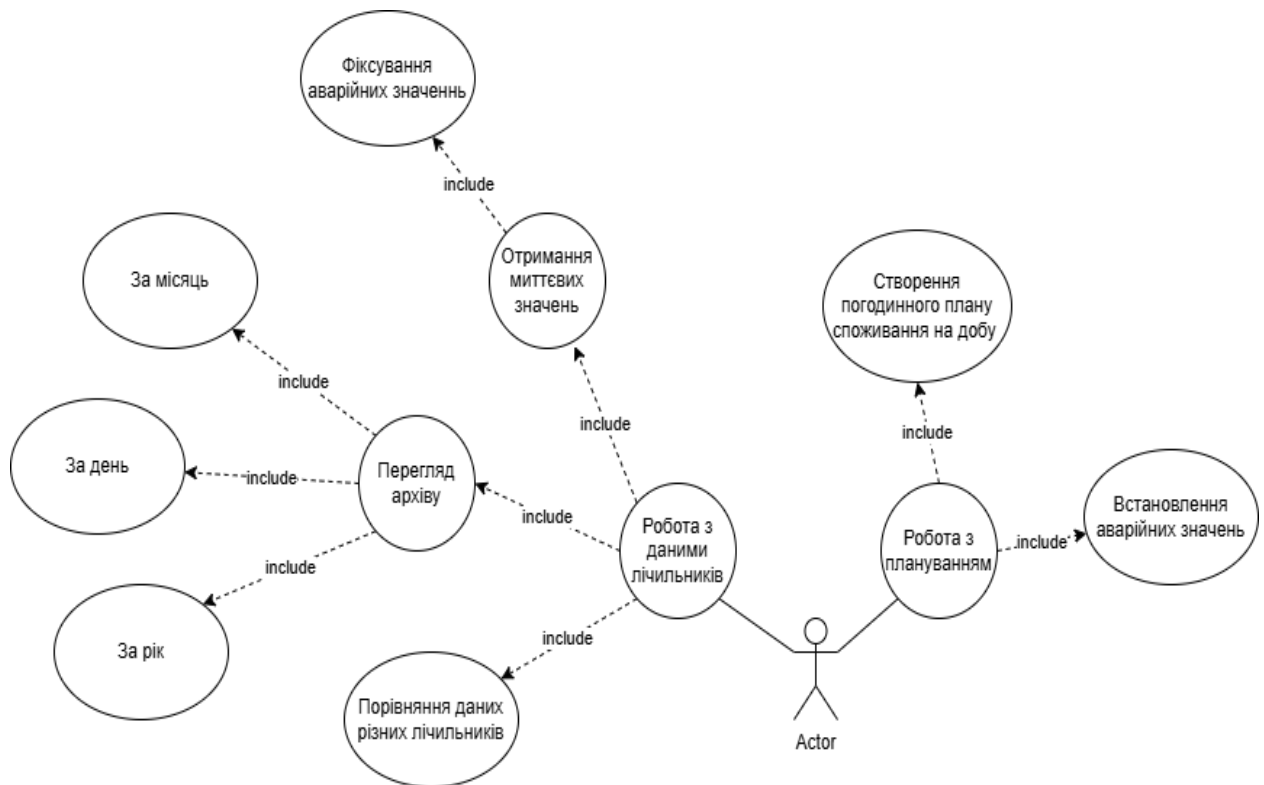


Рисунок 3.2 – Діаграма основних можливостей застосунку

Основним виконавцем (актором) системи є користувач, який взаємодіє з даними електролічильників, аварійними значеннями, архівами та плануванням добового навантаження.

Усі ці прецеденти формують основні можливості застосунку, забезпечуючи користувачу повний цикл взаємодії з даними лічильників від отримання поточних значень до перегляду архівів, фіксування аварій та формування планів споживання.

У наступних розділах буде детально описано реалізацію кожного з представлених функціональних компонентів

3.3.2 Серверна частина вебзастосунку

Серверна частина вебзастосунку виконує роль центральної ланки системи, що забезпечує зв'язок між клієнтським інтерфейсом користувача та базою даних. Вона відповідає за прийом запитів від користувача, їх обробку, звернення до сховища даних, виконання необхідних обчислень і повернення результатів у зручному для клієнта форматі.

Розробка серверної частини здійснювалася на основі середовища Node.js із використанням фреймворку Express.js. Node.js ґрунтується на подієвій моделі з неблокуючим введенням-виведенням, що робить його ідеальним рішенням для вебдодатків, де важливою є швидка реакція на дії користувача.

3.3.3 Архітектура серверної частини вебзастосунку

Архітектура вебзастосунку побудована за принципом «клієнт–сервер», у межах якого сервер виконує роль посередника між користувачем та базою даних. Процес взаємодії відбувається послідовно: користувач, працюючи з клієнтським інтерфейсом вебсторінки, надсилає запит до сервера. Серверна частина приймає цей запит, перевіряє його коректність, після чого формує відповідний SQL-запит до бази даних. Отримані з бази результати проходять попередню обробку та структурування, після чого у вигляді об'єкта формату JSON передаються назад на клієнт. Клієнтська частина, у свою чергу, відображає отримані дані у зручній для користувача формі.

Такий підхід забезпечує чітке розділення логіки застосунку на три незалежні рівні: рівень представлення, який відповідає за інтерфейс користувача; рівень логіки, реалізований на серверній стороні; та рівень даних, що включає базу даних і всі операції з нею. Подібна структурна організація спрощує подальший супровід програмного продукту, полегшує внесення змін

до окремих компонентів і дає можливість масштабувати систему без суттєвих змін її загальної архітектури.

3.3.4 Принцип роботи та структура серверної частини

Серверна частина вебзастосунку базується на використанні фреймворку Express.js, що забезпечує швидку та зручну розробку REST API для взаємодії з клієнтським інтерфейсом. На початковому етапі здійснюється ініціалізація та налаштування сервера. Підключаються необхідні бібліотеки, створюється екземпляр додатка Express і задається порт, на якому сервер очікує вхідні з'єднання. Для забезпечення взаємодії з клієнтами, що можуть працювати з різних доменів, налаштовано політику CORS, яка дозволяє обмін даними між різними джерелами. Використання методу `express.json()` забезпечує можливість прийому та обробки запитів, тіло яких містить дані у форматі JSON, що є стандартом у сучасних вебзастосунках.

Для роботи з базою даних Microsoft SQL Server створено об'єкт конфігурації `config`, який містить усі необхідні параметри з'єднання: логін і пароль користувача, назву бази даних, ім'я сервера, а також налаштування безпеки з урахуванням довірених з'єднань. З'єднання з базою реалізовано за допомогою бібліотеки `mssql`, яка підтримує асинхронні виклики (`async/await`). Такий підхід дозволяє виконувати SQL-запити без блокування основного потоку виконання, що суттєво підвищує ефективність і стабільність роботи серверного застосунку.

Серверна частина реалізує кілька маршрутів API, які відповідають за обмін інформацією з клієнтським інтерфейсом:

- `/getdat/:param` використовується для отримання комплексних даних за певну дату. Під час його виконання сервер звертається одночасно до трьох таблиць бази даних — `instans`, `plan` та `hours`. Отримані результати об'єднуються в один структурований об'єкт і повертаються клієнту у форматі

JSON. Такий підхід дозволяє зменшити кількість окремих запитів до сервера та підвищує швидкодію системи.

- `/warning` реалізує механізм оновлення граничних параметрів попередження, що дає змогу користувачеві задавати допустимі межі для потужності або напруги. Ці параметри надалі можуть бути використані для автоматичного сповіщення про відхилення від нормальних значень, підвищуючи інформативність і надійність роботи системи моніторингу.

- `/plan` відповідає за додавання або оновлення планових показників у таблиці `plan`. Під час обробки запиту сервер перевіряє, чи існує запис для вказаної дати. Якщо запис відсутній, система автоматично створює новий рядок, що гарантує цілісність бази даних і запобігає дублюванню інформації. Такий підхід забезпечує збереження актуальності даних і спрощує керування плановими параметрами.

Особливістю серверної частини є організація асинхронної роботи, що базується на використанні конструкції `async/await`. Завдяки цьому всі запити до бази даних виконуються у неблокуючому режимі, що дозволяє серверу обслуговувати одночасно кілька користувачів без затримок. Такий механізм сприяє підвищенню продуктивності та швидкодії системи, особливо за умов великої кількості клієнтських звернень або значних обсягів даних.

У результаті така структура серверної частини забезпечує ефективну взаємодію між користувачем, сервером і базою даних, підтримує стабільність роботи та дозволяє легко масштабувати систему в майбутньому.

3.3.5 Клієнтська частина вебзастосунку

Клієнтська частина вебзастосунку призначена для забезпечення взаємодії користувача із системою моніторингу та керування енергоспоживанням. Вона реалізує графічний інтерфейс, за допомогою якого користувач може переглядати поточні параметри, встановлювати планові

показники, задавати граничні значення напруги та струму, а також аналізувати зібрані дані у вигляді графіків.

Розробка клієнтської частини здійснювалася з використанням фреймворку Vue.js, який дозволяє створювати реактивні вебінтерфейси та динамічно оновлювати відображення даних без перезавантаження сторінки. Vue.js забезпечує компонентну структуру, що сприяє логічному поділу інтерфейсу на окремі елементи — панелі, таблиці, поля введення, графіки тощо. Кожен компонент відповідає за власний функціонал і може бути повторно використаний у різних частинах системи.

Основним завданням клієнтської частини є обробка введених користувачем даних, формування запитів до серверної частини та відображення отриманих результатів. Для обміну інформацією використовується протокол HTTP, а формат переданих даних — JSON, що забезпечує простоту серіалізації та сумісність між різними мовами програмування.

Інтерфейс користувача містить елементи керування для вибору дати, за яку необхідно отримати дані, та кнопки для надсилання відповідних запитів. Результати відображаються у вигляді таблиць і графіків, що дозволяє зручно аналізувати зміну параметрів у часі. Для побудови графічних елементів застосовано бібліотеку Chart.js, інтегровану через Vue-компонент. Це забезпечує візуальне представлення фактичних і планових показників у єдиній координатній системі.

Крім перегляду даних, клієнтська частина забезпечує можливість внесення змін у параметри системи. Користувач може редагувати граничні значення або оновлювати планові дані, після чого ці зміни передаються на сервер через POST-запити. Таким чином реалізовано повноцінний двосторонній зв'язок між користувачем і сервером, що дає змогу оперативно впливати на процеси керування енергоспоживанням.

3.3.6 Архітектура клієнтської частини вебзастосунок

Архітектура клієнтської частини побудована за принципом компонентно-орієнтованої структури, що є типовим підходом для сучасних односторінкових вебзастосунків. Кожен компонент інтерфейсу відповідає за окрему частину функціоналу: відображення графіків, роботу з таблицями даних, обробку форм введення або виконання запитів до сервера. Це забезпечує модульність і гнучкість системи, а також спрощує її підтримку та розширення.

Взаємодія між клієнтською та серверною частинами відбувається через API-запити, які надсилаються з Vue-компонентів за допомогою бібліотеки Axios. Отримані відповіді проходять попередню обробку, після чого результати зберігаються у змінних стану компонентів і відображаються у вигляді таблиць або графіків.

Клієнтська частина також реалізує функції локальної перевірки даних (валідації), що запобігає введенню некоректних значень — наприклад, від’ємних чисел або порожніх полів. Завдяки цьому система підвищує рівень надійності та захисту від помилкових дій користувача.

Для підвищення зручності роботи інтерфейс передбачає автоматичне оновлення даних через певний часовий інтервал, що дозволяє користувачу спостерігати актуальні показники у майже реальному часі. Такий підхід підвищує ефективність моніторингу та робить застосунок більш інтерактивним.

Загалом архітектура клієнтської частини забезпечує зручність, стабільність та масштабованість системи, дозволяючи легко інтегрувати нові модулі або розширювати функціональність без суттєвого втручання в існуючу структуру коду.

3.3.7 Принцип роботи та структура клієнтської частини

Клієнтська частина складається з кількох основних логічних модулів:

- компонент відображення даних відповідає за візуалізацію таблиці з поточними показниками та графіків;
- компонент взаємодії з користувачем забезпечує обробку подій, введення даних, вибір дати, підтвердження змін і повідомлення про помилки;
- модуль запитів до сервера виконує асинхронні запити (через `axios`) до API для отримання або оновлення інформації;
- модуль обробки стану контролює збереження актуальних даних у пам'яті застосунку, щоб уникнути повторних запитів і прискорити роботу інтерфейсу.

Усі модулі взаємодіють між собою через систему подій при зміні будь-якого параметра (наприклад, вибору дати або редагуванні плану) компонент генерує подію, яку обробляють інші частини інтерфейсу, оновлюючи відповідні елементи.

При зміні будь-якого параметра дані не просто змінюються локально вони автоматично проходять перевірку, надсилаються на сервер, і після підтвердження оновлюються всі пов'язані елементи інтерфейсу. Це створює відчуття роботи в реальному часі, коли система миттєво реагує на дії користувача

Після завантаження сторінки клієнтська частина автоматично ініціалізує запит до сервера для отримання даних за поточну дату. Отримана інформація зберігається у змінних стану та миттєво відображається в таблиці. Користувач може змінити дату при цьому виконується новий запит до сервера з параметром дати, і таблиця та графік оновлюються в реальному часі.

Інтерфейс реалізує двостороннє зв'язування даних: усі зміни, які користувач вносить у форму або таблицю, одразу відображаються у внутрішньому стані застосунку. Після натискання кнопки “Зберегти” модуль обробки запиту формує об'єкт даних і надсилає його на сервер через API `/plan`

або /warning. Якщо операція успішна, користувач отримує повідомлення про успішне збереження; у разі помилки – система виводить відповідне попередження.

3.4 Опис бази даних

У розробленому застосунку структура бази даних спроектована для ефективного збереження, обробки та аналізу даних, отриманих від електровимірювальних пристроїв. Основна мета цієї бази даних забезпечення централізованого доступу до інформації про споживання електроенергії, миттєві показники напруги та потужності, а також планові значення енергоспоживання для подальшого порівняння.

База даних складається з кількох взаємопов'язаних сутностей:

- `Pribori` зберігає інформацію про пристрої, що здійснюють вимірювання. Кожен прилад має унікальний ідентифікатор та назву, що дозволяє однозначно ідентифікувати його в системі;
- `instans` містить миттєві показники активної потужності, фазної напруги, а також поточний час зняття показників. Ця таблиця використовується для оперативного моніторингу стану електромережі в реальному часі;
- `hours` накопичує погодинні дані про спожиту енергію за кожен прилад, що дозволяє будувати аналітику за вибраний період;
- `plan` зберігає планові значення енергоспоживання по годинах доби, що дозволяє здійснювати порівняння між фактичними та запланованими показниками, а також розраховувати відхилення від заданих норм.

Сутності представлені у табл. 3.3-3.6.

Така структура бази даних забезпечує модульність та логічну зв'язність між сутностями, дозволяючи системі швидко отримувати необхідну інформацію для візуалізації, статистичного аналізу та контролю енергоспоживання. Використання реляційних зв'язків між таблицями

спрощує виконання складних SQL-запитів, підвищує швидкодію доступу до даних і дає змогу легко розширювати функціонал застосунку без необхідності суттєвої зміни структури БД.

Таблиця 3.3 – Сутність pribori

Назва поля	Тип змінної	Призначення
id	int	Унікальне числове значення
Pribor_Name	string	Зберігає назву лічильника

Таблиця 3.4 – Сутність instans

Назва поля	Тип змінної	Призначення
id	int	Унікальне числове значення
Id_pribor	int	Id лічильника з якого зчитані значення
Inst_activ	float	миттєве значення сили току
Phas_voltag	float	миттєве значення напруги
w_activ	float	зберігає крайню межу сили струму, задану оператором
w_voltag	float	зберігає крайню межу напруги, задану оператором
date	datetime	дата та час коли були записані значення

У табл. 3.5-3.6 представленні сутності hours та plan.

Таблиця 3.5 – Сутність hours

Назва поля	Тип змінної	Призначення
id	int	Унікальне числове значення
energy	float	зберігає накопичувальне значення використаної електроенергії
data	datetime	дата та час коли були записані значення
Id_pribor	int	Id лічильника з якого зчитані значення

Таблиця 3.6 – Сутність plan

Назва поля	Тип змінної	Призначення
id	int	Унікальне числове значення
[00-01]...[23-00]	float	зберігає план по кількості споживаної енергії за кожну годину
date	datetime	дата на яку повинен використовуватись план
Id_pribor	int	Id лічильника з якого зчитані значення
Sum_pot	float	Планове значення використаної енергії за весь день

На рис. 3.3 зображено діаграму структури бази даних.

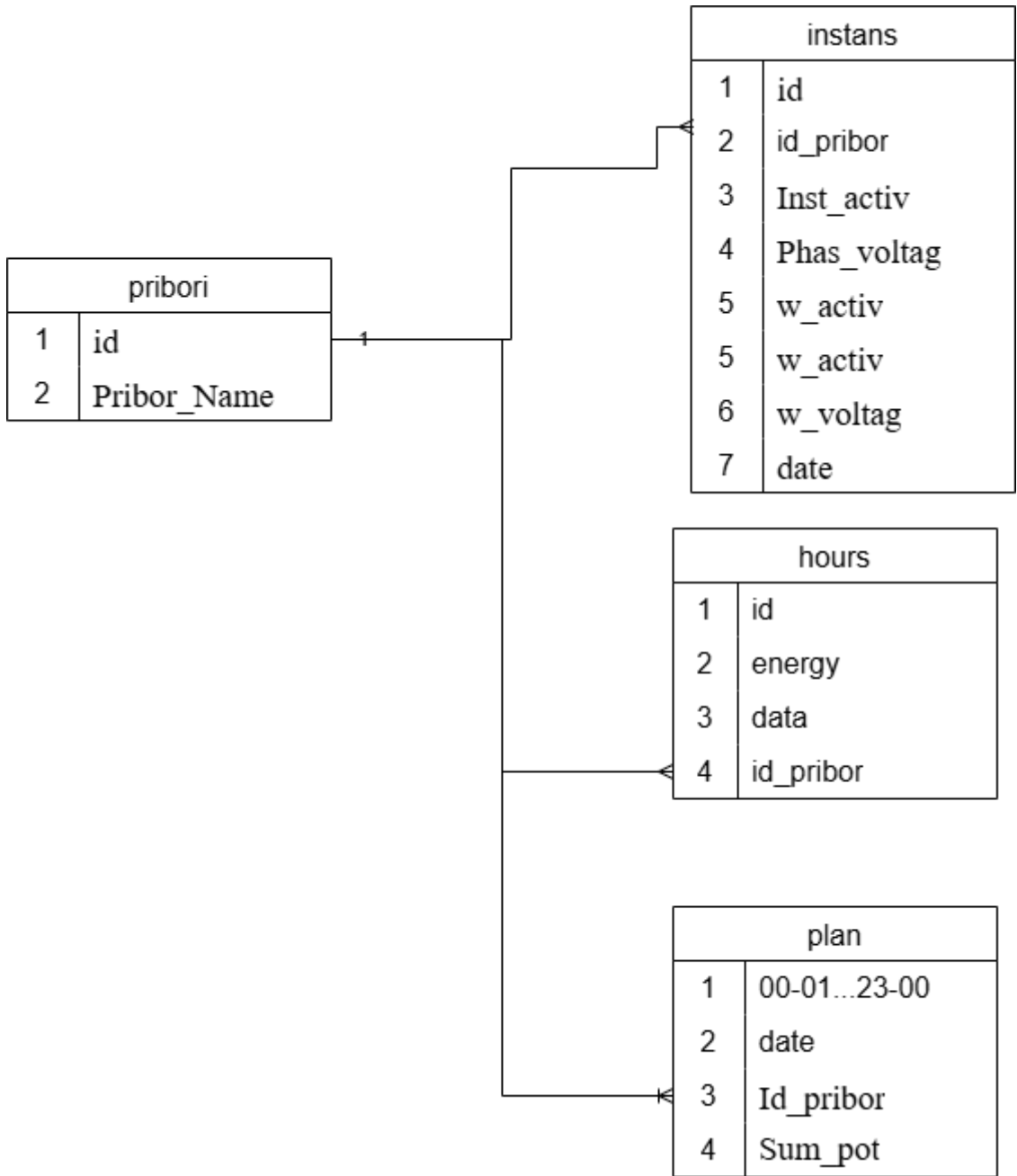


Рисунок 3.3 – Діаграма бази даних

3.4 Висновки за розділом 3

У цьому розділі було детально розглянуто архітектуру, алгоритми та програмну реалізацію системи опитування електролічильників. Зокрема, описано схему алгоритму функціонування серверного додатку, що відповідає за періодичний збір даних із пристроїв вимірювання, їх попередню обробку та передачу до бази даних.

Було проаналізовано особливості програмної реалізації серверної частини, у якій застосовано сучасні технології та бібліотеки для роботи з протоколом DLMS/COSEM, забезпечення стабільного з'єднання з приладами та ефективного обміну даними. Наведено опис основних функцій і змінних, які реалізують процес ініціалізації з'єднання, зчитування показників і передачу результатів на клієнтську частину.

Розглянуто структуру вебзастосунку, що включає взаємодію між клієнтським та серверним компонентами. Клієнтська частина забезпечує відображення даних у зручному інтерфейсі, реалізованому за допомогою сучасного фреймворку, тоді як серверна частина виконує роль проміжної ланки між користувачем та приладами.

Також розроблено та представлено структуру бази даних, яка забезпечує надійне збереження отриманих показників, їх впорядкування та можливість подальшого аналізу. Описано основні таблиці, їх взаємозв'язки та призначення, що дозволяє ефективно працювати з даними різного типу — миттєвими, погодинними та плановими.

Загалом у розділі було сформовано повне уявлення про логіку роботи, структуру та принципи функціонування програмного комплексу, що реалізує процес автоматизованого збору, збереження та візуалізації даних електролічильників.

4 КЕРІВНИЦТВО КОРИСТУВАЧА ТА ПРИКЛАД РОБОТИ

4.1 Налаштування параметрів та запуск програми опитування лічильника

Перед запуском готової програми необхідно виконати початкове налаштування параметрів підключення до бази даних і лічильника, а також задати перелік OBIS-кодів, які мають зчитуватись. Після внесення змін можна здійснити компіляцію та запуск виконавчого файлу.

Крок 1. Для підключення до серверної бази даних використовується єдина змінна рядка підключення, у якій зазначаються всі параметри: адреса сервера, назва бази даних, логін і пароль користувача. Перед запуском програми необхідно перевірити коректність адреси сервера та назви бази даних

Приклад:

```
string connectionString =  
"Server=DESKTOPNM9BPD2\\SQLEXPRESS;  
Database=electro_opros;  
User Id=test;  
Password=werystrong;  
Trusted_Connection=True;"
```

Крок 2. Програма виконує з'єднання з лічильником через Ethernet. Для цього потрібно вказати IP-адресу пристрою та порт зв'язку, зазначений у налаштуваннях лічильника а також адресу сервера лічильника та клієнта.

Приклад:

```
string host = "127.0.0.1";  
int port = 4060;  
int clientAddress = 16;  
int serverAddress = 1;
```

Крок 3. Вказання OBIS-кодів для зчитування

Перелік кодів, за якими буде виконуватись опитування, береться з технічної документації лічильника.

Приклад:

Vol = "1-0:1.8.0.255"

Nap = "1-0:1.6.0.255"

Hours = "1-0:1.4.0.255"

Крок 4. Компіляція програми

Крок 5. Запуск програми

Для запуску програми достатньо відкрити каталог з виконуваним файлом і запустити його подвійним кліком або через консоль командою.

Крок 6. Перевірка результату, після запуску програми у консольному вікні з'явиться повідомлення про встановлення з'єднань та виконання опитування (рис. 4.1).

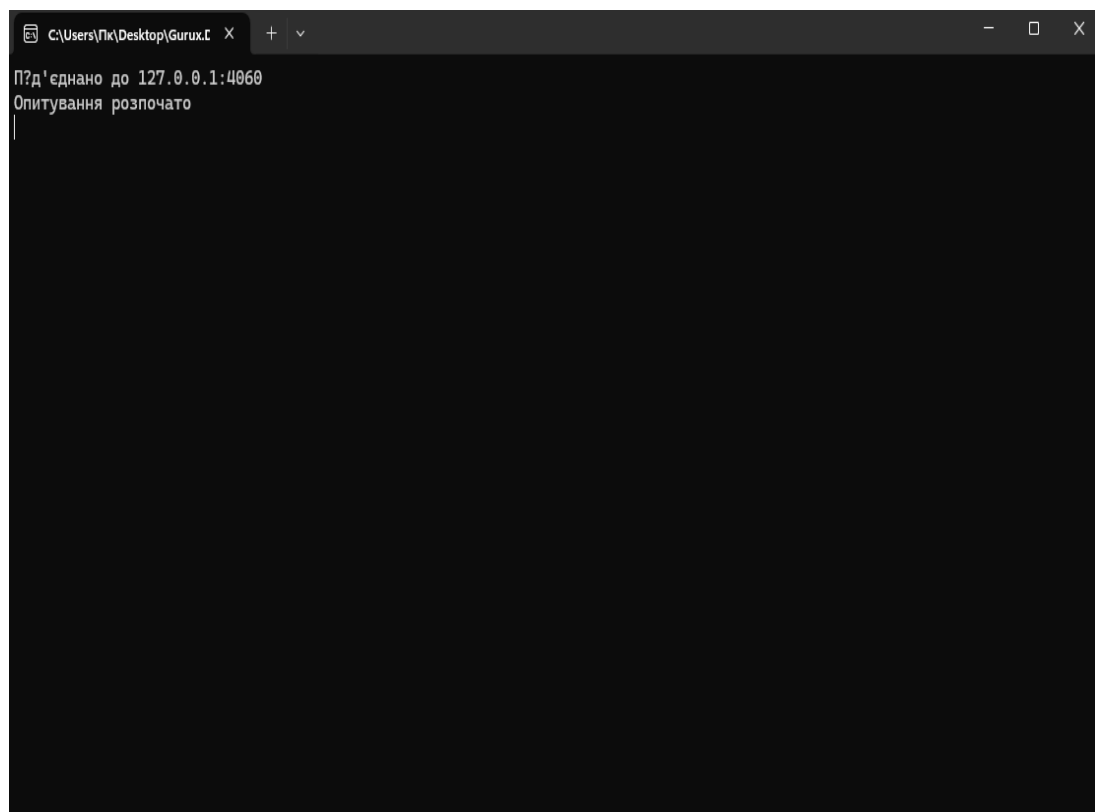


Рисунок 4.1 – Успішний запуск програми опитування

4.2 Розгортання вебзастосунок

Розгортання вебзастосунок є одним із завершальних етапів життєвого циклу створення програмного забезпечення, що визначає його готовність до практичного використання. На цьому етапі здійснюється перенесення розробленого коду на сервер, налаштування середовища виконання, встановлення залежностей та запуск системи у стабільному виробничому режимі. У даному проєкті реалізовано вебзастосунок, серверна частина якого побудована з використанням Node.js (Express), а клієнтська – на фреймворку Vue.js.

Перед початком розгортання необхідно підготувати середовище, у якому функціонуватиме застосунок. Сервер може бути розташований як на окремому фізичному комп'ютері, так і на віртуальній машині, що працює під управлінням операційної системи Windows або Linux. Важливо забезпечити стабільне підключення до локальної мережі або Інтернету, оскільки вебзастосунок обмінюється даними з клієнтськими пристроями та базою даних.

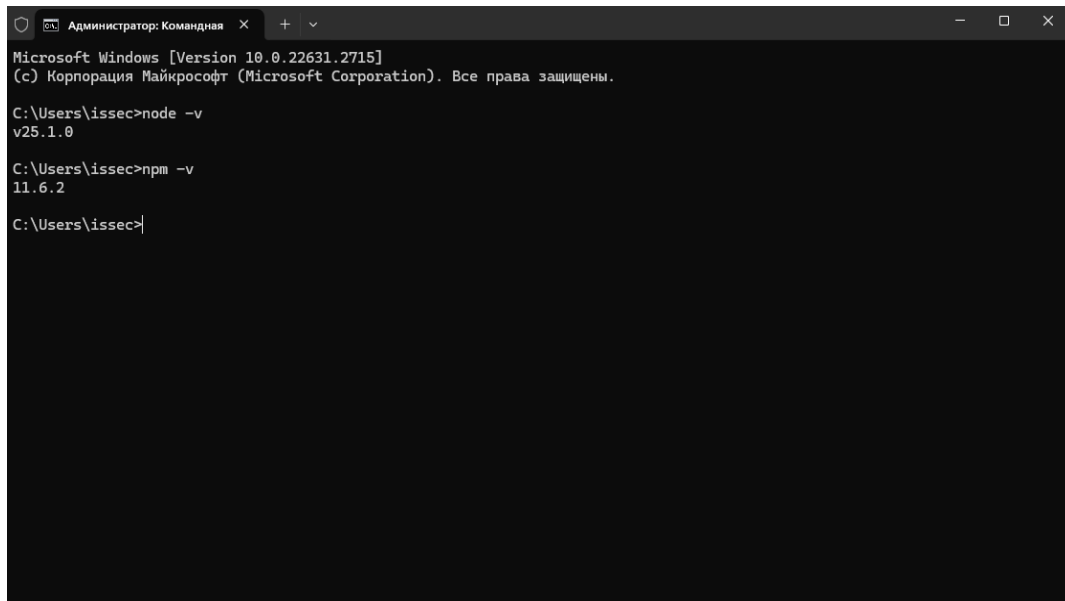
На сервер встановлюється Node.js – середовище виконання JavaScript, яке забезпечує роботу серверної логіки, обробку API-запитів та керування залежностями. Рекомендується використовувати актуальну стабільну версію Node.js. Перевірка встановлення виконується командами `node -v`, `npm -v` де `npm` використовується для встановлення необхідних модулів і бібліотек, приклад виводу команд при успішній установці наведено на рис. 4.2.

У межах системи використовується база даних Microsoft SQL Server, що зберігає зчитані показники лічильників, параметри з'єднання, журнали подій та інші дані. На сервері необхідно встановити SQL Server, створити окрему базу даних та налаштувати користувача з відповідними правами доступу.

Підключення до бази даних здійснюється за допомогою рядка з'єднання, який зберігається у конфігураційному файлі серверного застосунок.

У цьому рядку зазначаються ім'я сервера, назва бази даних, логін і пароль користувача.

Такий підхід дозволяє централізовано змінювати параметри доступу без внесення змін у вихідний код програми, що є зручним і безпечним під час супроводу системи.



```
Администратор: Командная строка
Microsoft Windows [Version 10.0.22631.2715]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

C:\Users\issec>node -v
v25.1.0

C:\Users\issec>npm -v
11.6.2

C:\Users\issec>
```

Рисунок 4.2 – Вивід команд node -v ,npm -v

Після підготовки середовища здійснюється копіювання проєкту на сервер. Це можна виконати звичайним копіюванням через файловий менеджер, або за допомогою протоколу SCP що забезпечує безпечну передачу файлів через мережу.

Після перенесення проєкту у кореневу директорію, наприклад /var/www/app, необхідно встановити всі залежності, зазначені у файлі package.json, командою npm install.

У результаті будуть завантажені всі бібліотеки, необхідні для роботи застосунку: модулі для обробки HTTP-запитів, підключення до бази даних, логування, маршрутизації тощо, результат виводу команди наведено на рис. 4.3.

```

Администратор: Командная
112 packages are looking for funding
  run `npm fund` for details

22 vulnerabilities (4 low, 12 moderate, 5 high, 1 critical)

To address issues that do not require attention, run:
  npm audit fix

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.

C:\Users\issec\Desktop\diplom>cd dipsq
C:\Users\issec\Desktop\diplom\dipsq>npm install

up to date, audited 183 packages in 946ms

24 packages are looking for funding
  run `npm fund` for details

3 vulnerabilities (2 low, 1 high)

To address all issues, run:
  npm audit fix

Run `npm audit` for details.

C:\Users\issec\Desktop\diplom\dipsq>S|

```

Рисунок 4.3 – Вивід команди npm install

Після цього серверна частина запускається командою `node index.js`. У результаті буде запущено додаток серверу який реалізує API для взаємодії клієнта з базою даних, приклад успішного запуску на рис. 4.4.

```

Администратор: Командная
C:\Users\issec\Desktop\diplom\dipsq>
C:\Users\issec\Desktop\diplom\dipsq>
C:\Users\issec\Desktop\diplom\dipsq>
C:\Users\issec\Desktop\diplom\dipsq>
C:\Users\issec\Desktop\diplom\dipsq>node index.js
C:\Users\issec\Desktop\diplom\dipsq\node_modules\buffer-equal-constant-time\index.js:37
var origSlowBufEqual = SlowBuffer.prototype.equal;
                                     ^
TypeError: Cannot read properties of undefined (reading 'prototype')
    at Object.<anonymous> (C:\Users\issec\Desktop\diplom\dipsq\node_modules\buffer-equal-constant-time\index.js:37:35)
    at Module._compile (node:internal/modules/cjs/loader:1760:14)
    at Object.<js> (node:internal/modules/cjs/loader:1892:10)
    at Module.load (node:internal/modules/cjs/loader:1480:32)
    at Module._load (node:internal/modules/cjs/loader:1299:12)
    at TracingChannel.traceSync (node:diagnostics_channel:328:14)
    at wrapModuleLoad (node:internal/modules/cjs/loader:245:24)
    at Module.require (node:internal/modules/cjs/loader:1503:12)
    at require (node:internal/modules/helpers:152:16)
    at Object.<anonymous> (C:\Users\issec\Desktop\diplom\dipsq\node_modules\jsonwebtoken\node_modules\jwa\index.js:1:19)

Node.js v25.1.0

C:\Users\issec\Desktop\diplom\dipsq>node index.js
сервер запущено прослуховується порт 3001

```

Рисунок 4.4 – Успішно запущений сервер

Клієнтська частина, створена за допомогою Vue.js, потребує компіляції у виробничу версію. Для цього у кореневому каталозі клієнтського застосунку виконується команда `npm run build`.

У результаті формується оптимізований пакет статичних файлів (HTML, CSS, JavaScript), який зберігається у директорії `dist`, результати виконання команди наведено на рис. 4.5.

```

warning
entrypoint size limit: The following entrypoint(s) combined asset size exceeds the recommended limit (244 KiB). This can
impact web performance.
Entrypoints:
  app (784 KiB)
    js/chunk-vendors.6a23428f.js
    css/app.e5a5f33b.css
    js/app.865bb1b5.js

warning
webpack performance recommendations:
You can limit the size of your bundles by using import() or require.ensure to lazy load some parts of your application.
For more info visit https://webpack.js.org/guides/code-splitting/

File                                Size                                Gzipped
dist\js\chunk-vendors.6a23428f.js    768.69 KiB                         221.31 KiB
dist\js\app.865bb1b5.js             15.12 KiB                          5.53 KiB
dist\css\app.e5a5f33b.css           0.47 KiB                          0.28 KiB

Images and other types of assets omitted.
Build at: 2025-11-07T05:08:29.731Z - Hash: ecf2093d744098ce - Time: 9927ms

DONE Build complete. The dist directory is ready to be deployed.
INFO Check out deployment instructions at https://cli.vuejs.org/guide/deployment.html

PS C:\Users\issec\Desktop\diplom>

```

Рисунок 4.5 – Результат компіляції клієнта

Цю директорію необхідно розмістити у вебкаталозі сервера (наприклад, `/var/www/html`), який обслуговується вебсервером Nginx або Apache.

Вебсервер виконує роль посередника між користувачем і серверною частиною: статичні файли віддаються напряму клієнту, а API-запити перенаправляються до Node.js-сервера. Це дозволяє забезпечити коректну маршрутизацію та спрощує доступ до серверних ресурсів.

4.3 Приклад роботи

Після завершення розгортання серверної частини програми опитування та вебзастосунку система готова до використання. Доступ до вебінтерфейсу

здійснюється з комп'ютера, який знаходиться в одній локальній мережі з сервером.

Приклад роботи, для початку роботи треба відкрити веббраузер та вести посилання на сторінку (у даному випадку <http://localhost:8080/oinc/>), після чого можна буде побачити сторінку вебзастосунку, на якій автоматично буде зроблено запити до сервера на поточну дату та миттєві значення які одразу відобразяться (рис 4.6).

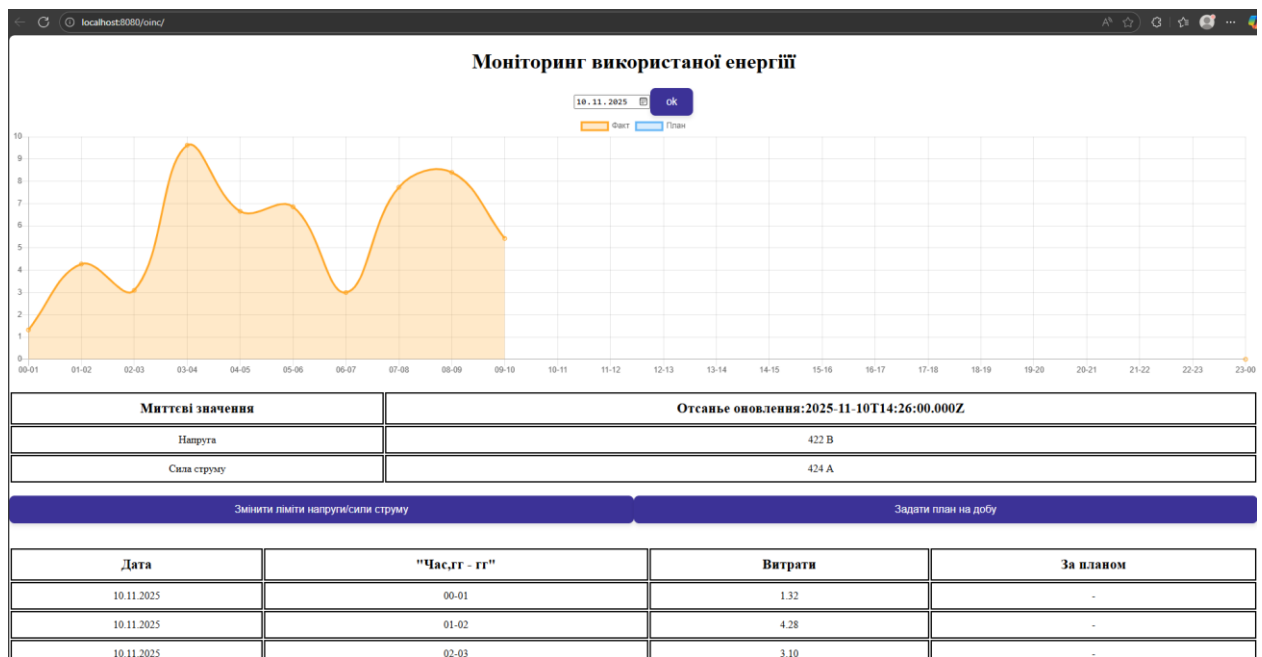


Рисунок 4.6 – Сторінка застосунку після відкриття.

Для перегляду архівних даних у верхній частині вебзастосунку передбачено елемент керування вибором дати (рис. 4.7). Користувач має можливість вручну ввести необхідну дату у відповідне поле або, натиснувши на елемент вибору, відкрити інтерактивний календар та обрати дату зі списку.

Після вибору дати необхідно підтвердити дію натисканням кнопки «ОК», після чого система надсилає запит до серверної частини програми з переданим параметром дати. Сервер виконує обробку запиту та повертає дані, що відповідають зазначеному часовому періоду.

Отримана інформація відображається у вигляді таблиці, яка містить значення фактичного споживання електроенергії та планові показники (якщо

вони були задані для обраної дати). Приклад відображення даних за дату 30.10.2025.

Під вибором дати можна бачити графік на якому відображаються погодинні витрати та план витрат по цим годинам.

Під графіком знаходяться миттєві значення, для редагування їх порогових значень потрібно натиснути відповідну кнопку.

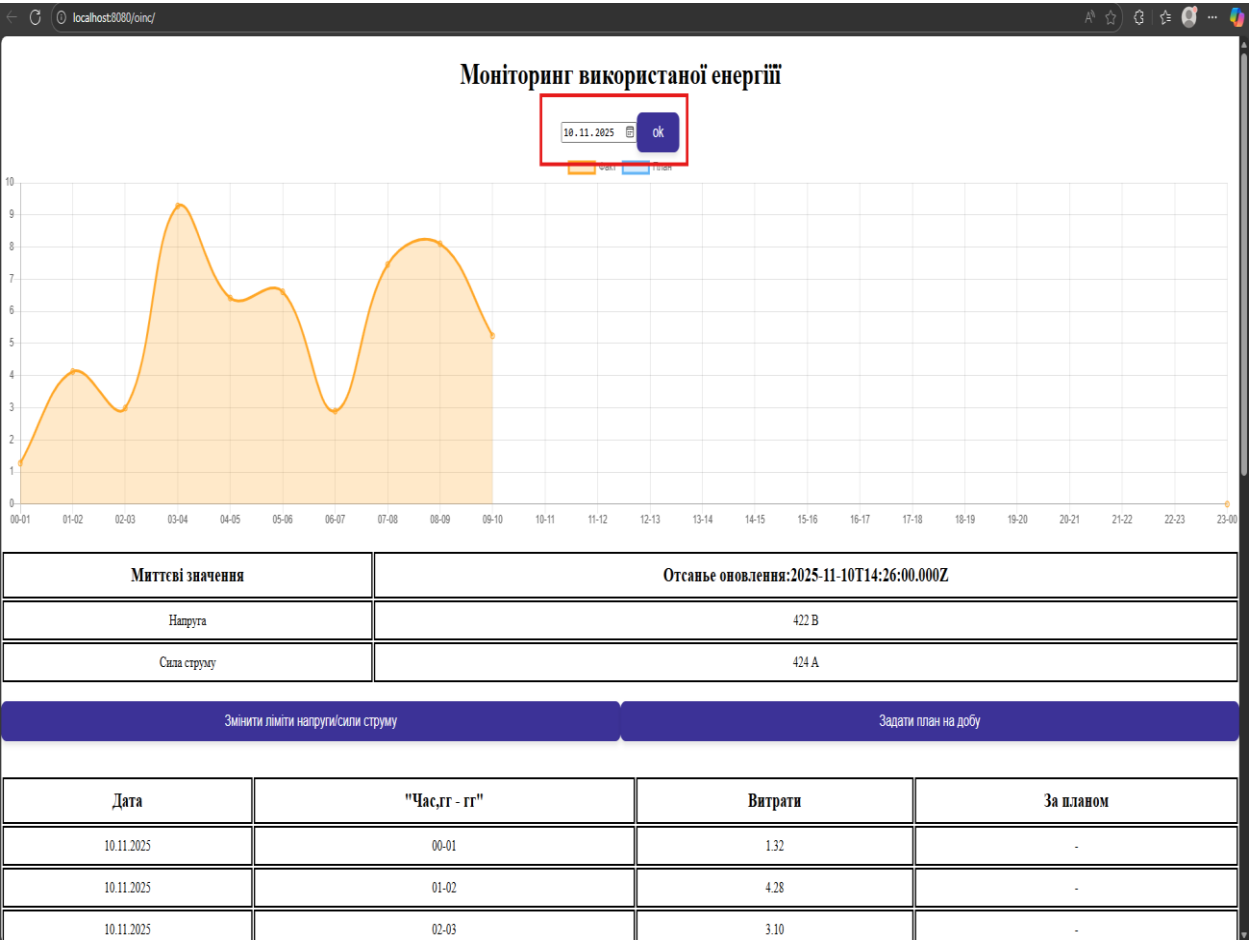


Рисунок 4.7 – Введення дати

Після чого відкриється вікно редагування де можна їх змінити після потрібних змін треба натиснути «Зберегти».

Біля кнопки редагування планових значень знаходиться кнопка щоб задати план на добу, вона буде активна лише у поточну та майбутні дати, створювати та редагувати план за минулі числа не можливо.

На рис. 4.8-4.9 зображено приклад обрання дати для перегляду архіву.

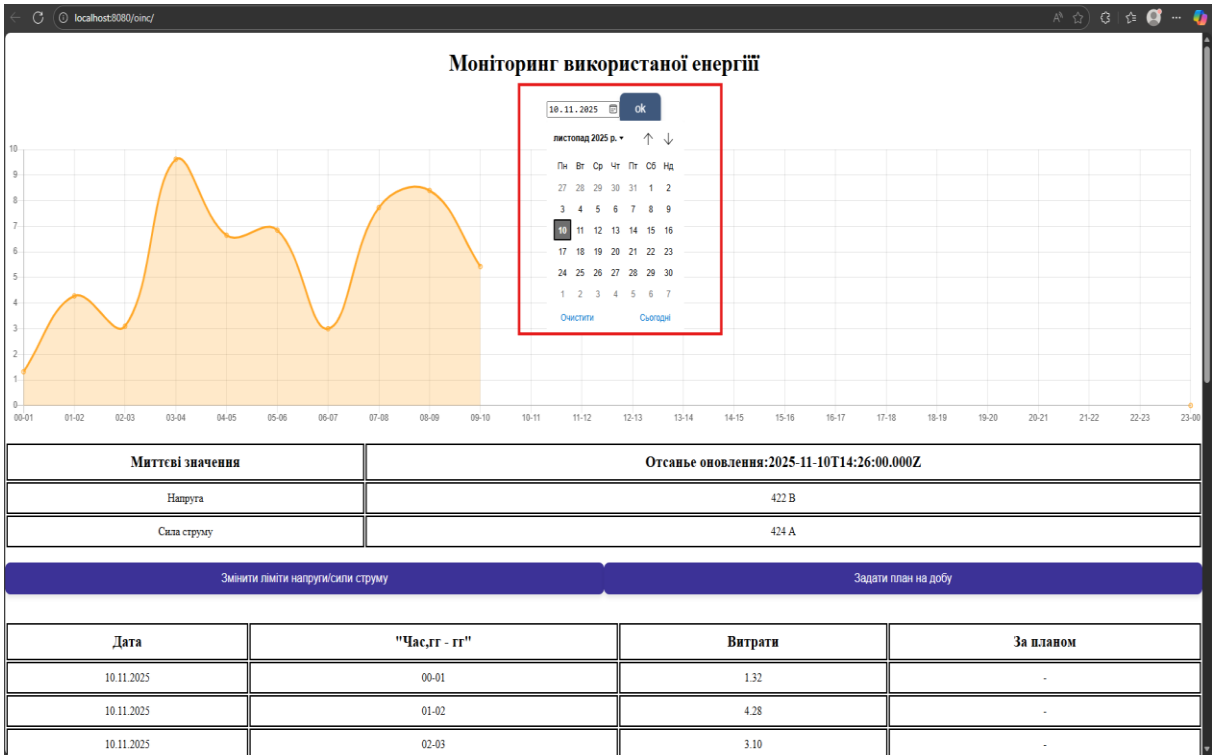


Рисунок 4.8 – Відкритий календар вибору дати

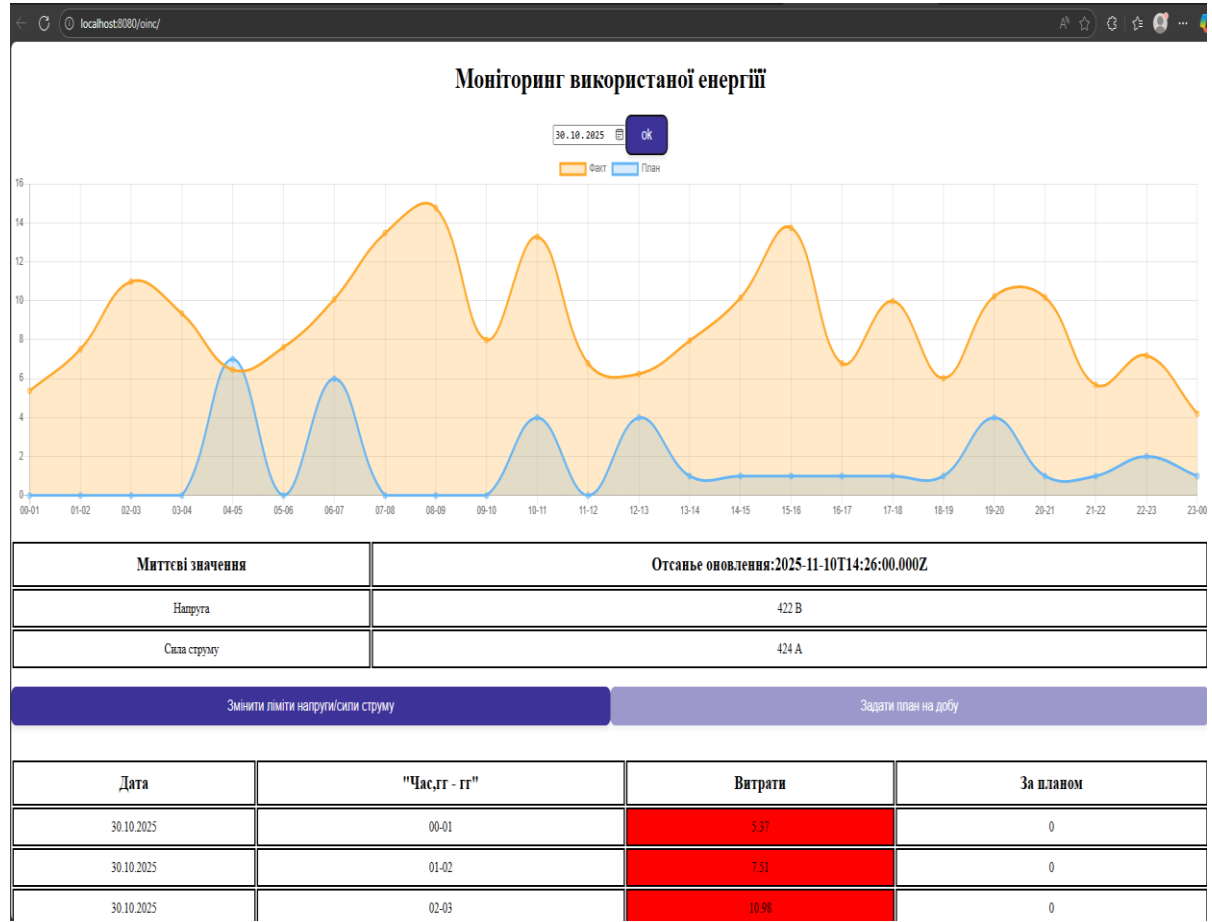


Рисунок 4.9 – Приклад перегляду архіву за 30.10.2025

На рис. 4.10 зображено знаходження кнопки редагування в інтерфейсі.

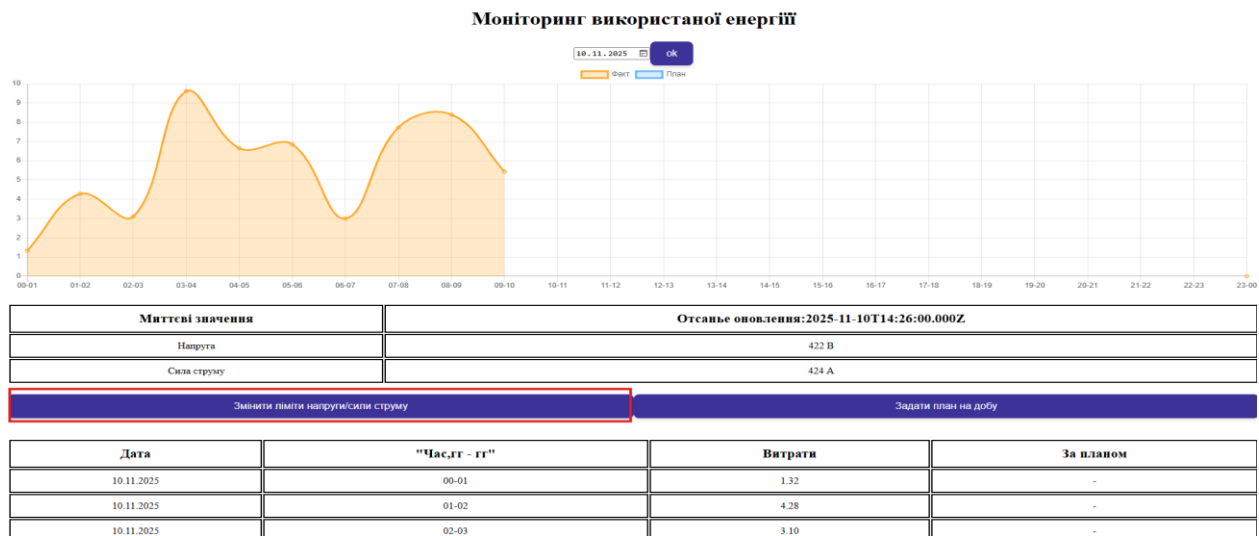


Рисунок 4.10 – Кнопка для початку редагування порогових значень

Після натискання в таблиці у комірках плану з'являться поля для його задання (рис. 4.13), де треба вести потрібні значення не менше 0, після чого для збереження потрібно натиснути «Зберегти», а випадку якщо це не потрібно, натиснути «Відмінити».

На рис. 4.11-4.12 зображено інтерфейс редагування порогових значень та кнопка редагування плану на добу.

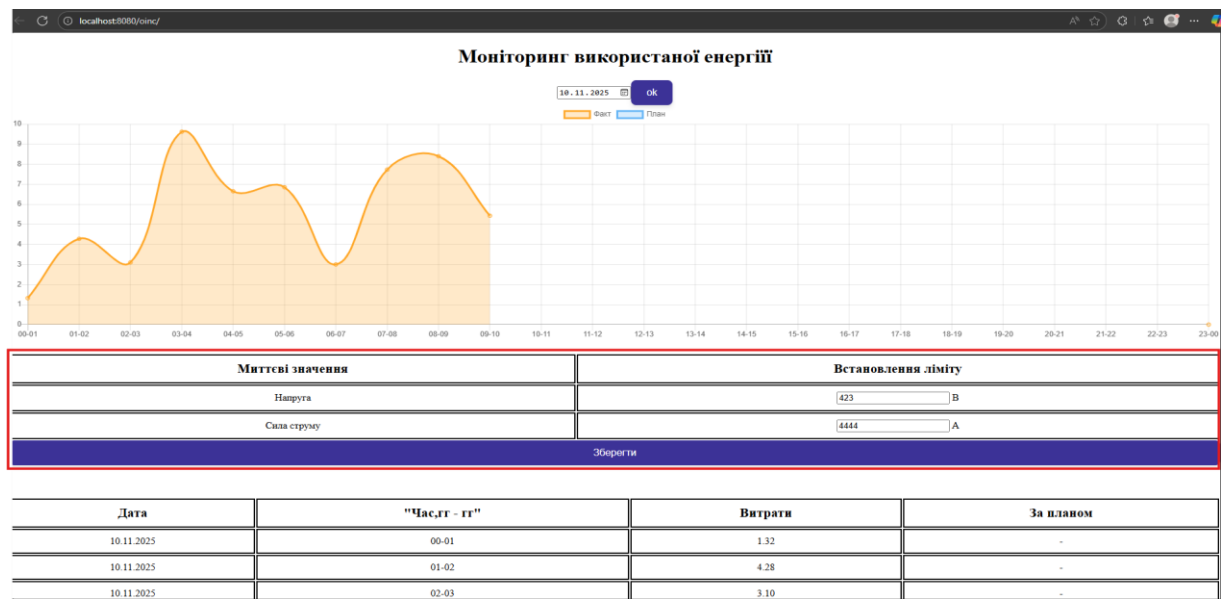


Рисунок 4.11 – Інтерфейс редагування порогових значень

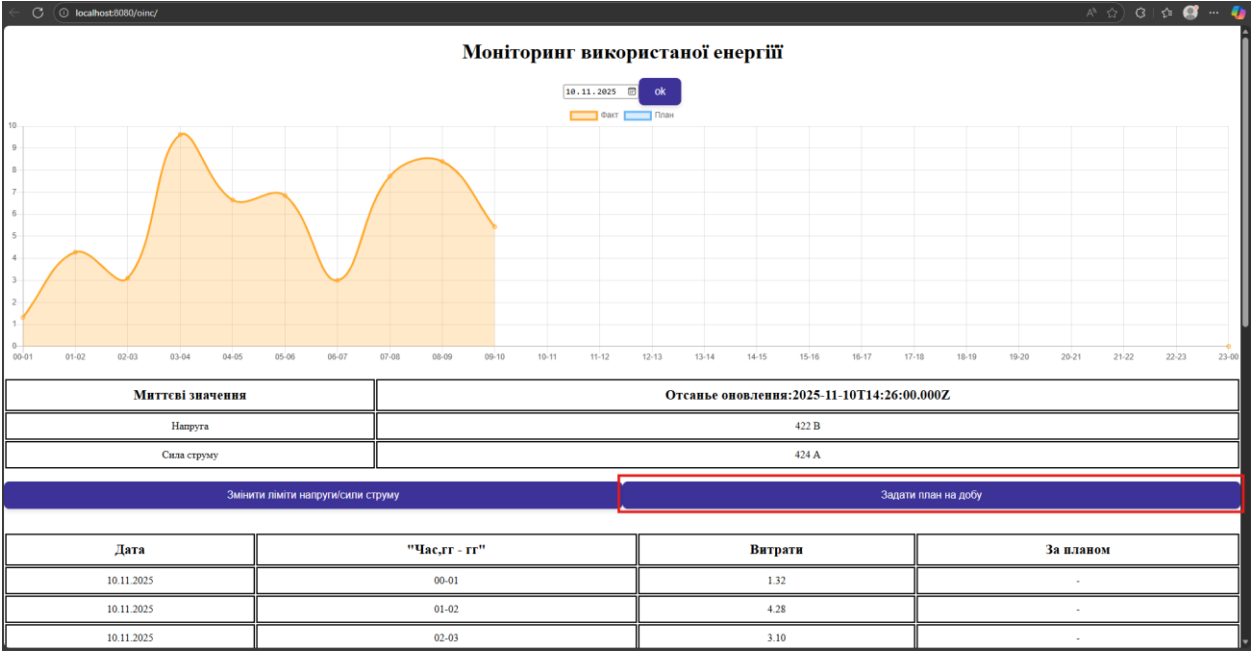


Рисунок 4.12 – Кнопка редагування плану на добу

Далі буде наведенні приклади що відбуваються коли значення виходять за задані межі:

У випадку коли миттєві значення доходять до заданих граничних значень, вони виділяються червоною кольором (рис. 4.14) та з'являється вікно з попередженням який параметр дійшов до граничного значення (рис. 4.15).

Напруга		422 В	
Сила струму		424 А	
Відмінити		Зберегти	
Дата	"Час,гг - гг"	Витрати	За планом
10.11.2025	00-01	1.32	
10.11.2025	01-02	4.28	
10.11.2025	02-03	3.10	
10.11.2025	03-04	9.62	
10.11.2025	04-05	6.65	
10.11.2025	05-06	6.85	
10.11.2025	06-07	3.00	
10.11.2025	07-08	7.73	
10.11.2025	08-09	8.40	
10.11.2025	09-10	5.43	
10.11.2025	10-11	-	
10.11.2025	11-12	-	
10.11.2025	12-13	-	
10.11.2025	13-14	-	
10.11.2025	14-15	-	

Рисунок 4.13 – Комірки редагування плану

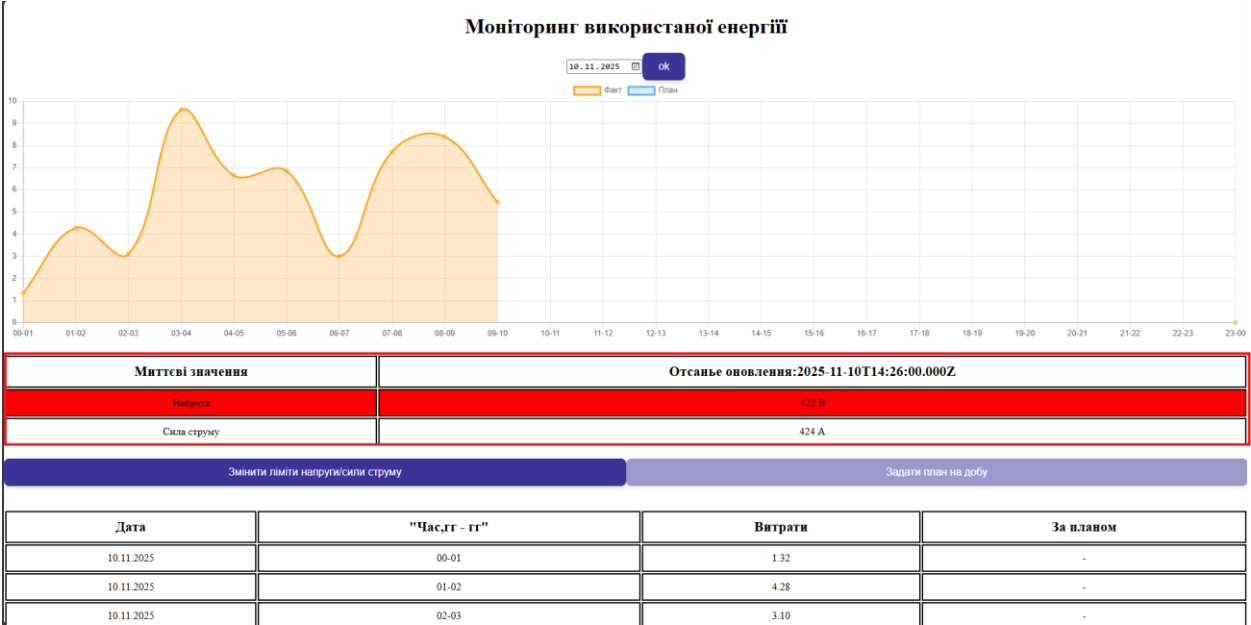


Рисунок 4.14 – Миттєве значення вийшло за задані межі

Так само у випадку коли погодинні значення виходять за межі плану у таблиці можливо побачити (виділені червоним кольором (рис. 4.16)) у яку годину яке значення вишло за межі плану.

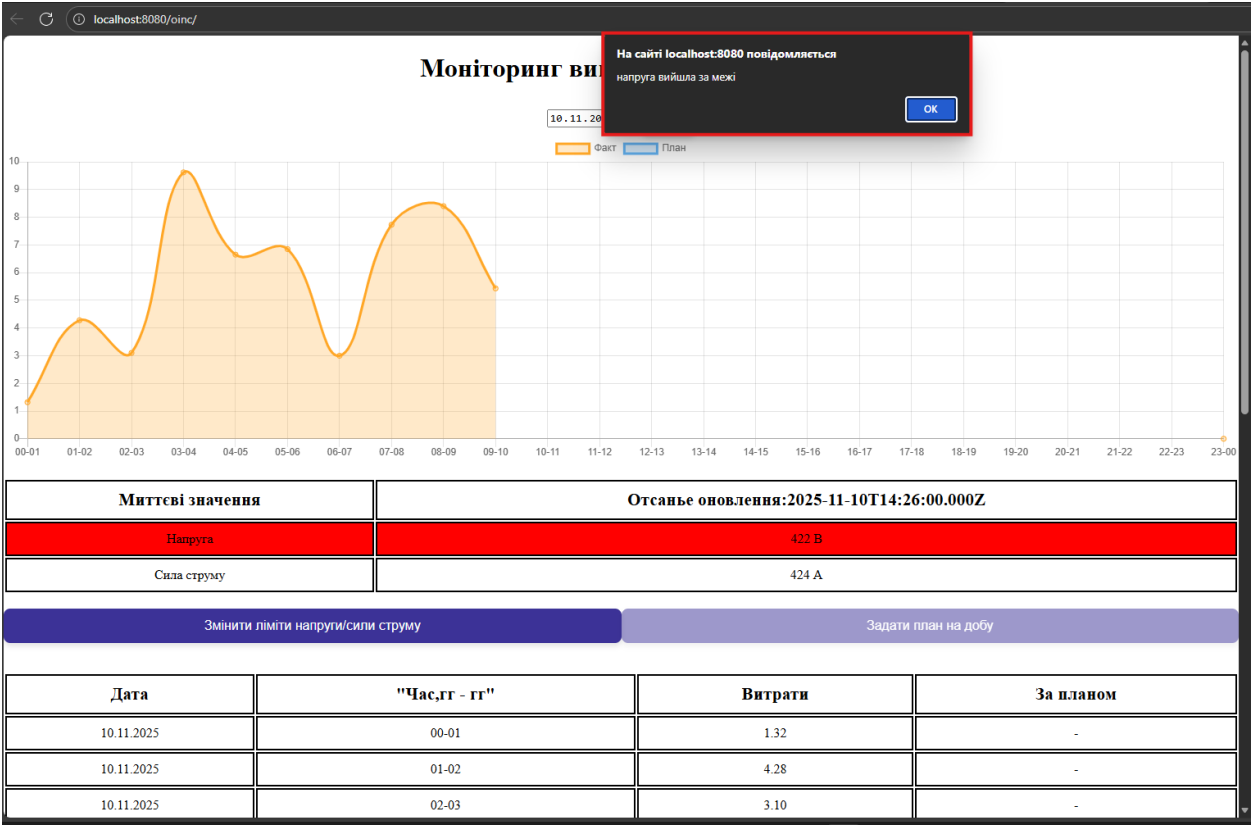


Рисунок 4.15 – Повідомлення про вихід миттєвого значення за задані межі

Миттєві значення		Отримане оновлення: 2025-11-10T14:26:00.000Z	
Напруга		423 В	
Сила струму		424 А	
Змінити ліміти напруги/сили струму		Задати план на добу	
Дата	"Час,гг - гг"	Витрати	За планом
30.10.2025	00-01	5.37	0
30.10.2025	01-02	7.51	0
30.10.2025	02-03	10.98	0
30.10.2025	03-04	9.34	0
30.10.2025	04-05	6.45	7
30.10.2025	05-06	7.61	0
30.10.2025	06-07	10.07	6
30.10.2025	07-08	11.47	0
30.10.2025	08-09	14.71	0
30.10.2025	09-10	7.99	0
30.10.2025	10-11	13.28	4
30.10.2025	11-12	6.78	0

Рисунок 4.16 – Червоним підсвічуються годинні значення які вийшли за план

4.5 Висновки за розділом 4

У даному розділі було детально розглянуто процес підготовки та налаштування всіх компонентів програмно комплексу, починаючи з встановлення параметрів підключення до електролічильника, та завершуючи прикладами роботи програми. Наведено приклади налаштування змінних з даними для з'єднання, вибору OBIS-кодів згідно з технічною документацією.

5 ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

5.1 Тестування системи

У процесі дослідження було виконано тестування працездатності розробленої системи збору та візуалізації даних електролічного.

Тестування проводилося на персональному комп'ютері з такими технічними характеристиками:

- центральний процесор – Pentium g4560;
- оперативна пам'ять – 4 ГБ;
- відеокарта – Intel HD Graphics 610.

Програмне середовище включало такі компоненти:

- операційна система – Windows 10;
- платформа серверної частини – Node.js v20;
- система керування базами даних – Microsoft SQL Server 2022;
- середовище розробки – Visual Studio Code.

Для перевірки працездатності серверної частини використовувалося юніт-тестування. Цей підхід дав змогу ізолювати окремі функціональні частини програми та перевірити їхню коректність незалежно від зовнішніх компонентів.

Зокрема, юніт-тестування проводилося для:

- перевірки коректності запитів до бази даних SQL Server;
- перевірки роботи ендпоінтів `/getdat/:param`, `/plan` та `/warning`;
- контролю відповідності структури об'єкта відповіді очікуваному формату JSON;
- перевірки коректності обробки виключних ситуацій (наприклад, відсутності даних або помилкового формату дати).

Юніт-тестування є одним із ключових підходів забезпечення надійності програмного забезпечення, особливо у вебсистемах із багаторівневою архітектурою, таких як Node.js + Vue.js.

Одним із важливих прикладів його застосування є тестування модулів серверної частини, що відповідають за зчитування даних з бази, їхню попередню обробку та передачу клієнтському застосунку для подальшої візуалізації.

Для створення тестів використовувалася бібліотека Jest, яка дозволяє автоматизувати процес перевірки функцій Node.js застосунку.

Окремо було проведено тестування програми опитування електролічного лічильника, розробленої на мові C#. Перевірка включала тестові запуски у середовищі Windows 10 з реальним підключенням до лічильника через Ethernet-інтерфейс. Було підтверджено стабільність обміну даними, коректність зчитування OBIS-кодів та передачу отриманих показників на серверний застосунок. Додатково перевірено реакцію програми на втрату зв'язку та відсутність відповіді від пристрою — система коректно фіксує помилки та повторює запит після короткої затримки.

5.2 Unit-тестування серверної частини

Модульне тестування проводилося для ізольованої перевірки роботи серверних маршрутів, реалізованих у середовищі Node.js за допомогою фреймворку Express.

Основна увага приділялася правильності виконання SQL-запитів, обробці отриманих результатів і поверненню коректної відповіді клієнту.

Для реалізації unit-тестів використано такі інструменти:

- Jest фреймворк для написання та виконання автоматизованих тестів у середовищі Node.js;
- Supertest бібліотека для перевірки HTTP-запитів до Express-додатків без запуску сервера на реальному порту;

Для початку тестування треба підключити додаток і налаштування Supertest:

```
const request = require('supertest');
const app = require('../server');
```

Ця конфігурація дозволяє емулювати HTTP-запити до серверу без запуску реального сервера. Завдяки цьому тести виконуються швидко, ізолювано та без взаємодії зі сторонніми сервісами. Ініціалізація забезпечує:

- контрольоване середовище тестування;
- можливість перевірки відповідей API;
- стабільну взаємодію із базами даних;

Розглянемо наступний код тестування :

Тестування маршруту `/getdat/:param`

```
describe('GET /getdat/:param', () => {
  it('повертає дані для вибраної дати', async () => {
    const res = await request(app).get('/getdat/2025-11-10');
    expect(res.statusCode).toBe(200);
    expect(res.body).toHaveProperty('instans');
    expect(res.body).toHaveProperty('plan');
    expect(res.body).toHaveProperty('fact');
  });
});
```

Маршрут `/getdat/:param` відповідає за отримання даних для вибраної дати. Юніт-тест перевіряє:

- статус відповіді - 200;
- наявність ключових об'єктів у відповіді (`instans`, `plan`, `fact`);
- відсутність помилок при виконанні SQL-запитів;

Тестування включає:

- ініціалізацію запиту через `Supertest`;
- перевірку статусу відповіді, що гарантує успішну обробку запиту сервером;

– перевірку структури відповіді, що підтверджує коректність формування SQL-запитів і обробки даних.

Тестування маршруту `/plan`

```
describe('POST /plan', () => {
  it('створює новий запис при відсутності плану', async () => {
    const mockBody = {
      date: '2025-11-10',
      val: Array(24).fill({ plan: 10 }),
      sum: 240
    };
    const res = await request(app).post('/plan').send(mockBody);
    expect(res.statusCode).toBe(200);
  });
});
```

Маршрут `/plan` відповідає за створення або оновлення планових значень.

Тест перевіряє:

- чи приймає сервер дані з тіла запиту;
- чи формує SQL-запит коректно;
- чи повертає статус 200 при успішному виконанні.

Тестування включає:

- моделювання даних створюється масив з 24 елементів для моделювання планових значень;
- верифікація відповіді сервера, щоб упевнитися, що дані були успішно прийняті;
- перевірка логіки створення запису, яка гарантує відсутність дублювання або помилок при відсутності плану.

Тестування маршруту `/warning`

```
describe('POST /warning', () => {
  it('оновлює граничні параметри', async () => {
```

```

const res = await request(app)
  .post('/warning')
  .send({ w_activ: 4000, w_voltag: 240 });
expect(res.statusCode).toBe(200);
});
});

```

Маршрут /warning дозволяє оновлювати граничні параметри.

Тест перевіряє:

- чи приймає сервер нові значення параметрів;
- чи успішно оновлює таблицю у базі даних;
- відсутність помилок при виконанні SQL-запиту.

Тестування включає:

- відправку даних на сервер для оновлення параметрів;
- перевірку статусу відповіді як ознаку успішного виконання;
- перевірку логіки оновлення, що гарантує коректну роботу бізнес-логіки при зміні граничних значень.

У табл. 5.1 наведенні тести вхідні данні та результати

Таблиця 5.1 – Тестування та результати

№	Назва тесту	Вхідні дані	Очікуваний результат	Статус
1	2	3	4	5
1	Перевірка підключення до бази даних	Дані з config: user, password, database, server	Успішне підключення без помилок	Пройдено
2	Отримання даних за маршрутом /getdat/:param	param = '2025-10-10'	Повертається JSON із трьома масивами	Пройдено
3	Некоректна дата у /getdat/:param	param = 'не_дата'	Сервер повертає помилку 400 або обробляє виняток без зупинки роботи	Пройдено

Продовження таблиці 5.1

1	2	3	4	5
4	Оновлення попереджень /warning	w_activ = 50, w_voltag = 220	Дані в таблиці instans оновлено, запит не викликає помилки	Пройдено
5	Надсилання плану на день /plan (існуючий запис)	Масив 24 значень + дата	Оновлюється рядок у таблиці plan для вказаної дати	Пройдено
6	Надсилання плану на день /plan (новий запис)	Масив 24 значень + нова дата	Створюється новий запис у таблиці plan	Пройдено
7	Відсутній параметр у тілі POST-запиту /plan	Порожній req.body	Сервер повертає помилку	Пройдено
8	Крос-доменний запит (CORS)	Запит з іншого домену	Сервер дозволяє виконання запиту (CORS працює)	Пройдено
9	Некоректний SQL-сервер	Змінене поле server у config	Виводиться помилка з'єднання, сервер не завершує роботу	Пройдено
10	Завантаження сторінки при запуску сервера	node server.js	Сервер запускається на порту 3001, доступний для запитів	Пройдено

5.3 Аналіз впливу кроку дискретизації на точність розрахунку активної енергії

Для оцінки впливу частоти опитування лічильника на точність визначення добового споживання було використано набір реальних даних з інтервалом у одну годину, вхідні данні наведені в табл. 5.2. Еталонне споживання визначалося як різниця між кінцевим та початковим показами накопиченої активної енергії за добу.

Таблиця 5.2 – Вхідні данні отриманні з лічильника

Спожита енергія кВт·год.	Час споживання
88,0774477754778	2025-10-30 00:00:00.000
93,4477252406468	2025-10-30 01:00:00.000
100,957576280457	2025-10-30 02:00:00.000
111,94152558886	2025-10-30 03:00:00.000
121,285264621723	2025-10-30 04:00:00.000
127,731385344242	2025-10-30 05:00:00.000
135,344989926614	2025-10-30 06:00:00.000
145,414020746403	2025-10-30 07:00:00.000
158,88208213192	2025-10-30 08:00:00.000
173,648388050562	2025-10-30 09:00:00.000
181,633838603686	2025-10-30 10:00:00.000
194,909680359668	2025-10-30 11:00:00.000
201,687489302528	2025-10-30 12:00:00.000
207,928048575282	2025-10-30 13:00:00.000
215,863075627835	2025-10-30 14:00:00.000
226,002525910078	2025-10-30 15:00:00.000
239,738994552734	2025-10-30 16:00:00.000
246,518121154092	2025-10-30 17:00:00.000
256,500032611983	2025-10-30 18:00:00.000
262,524727282896	2025-10-30 19:00:00.000
272,742886473942	2025-10-30 20:00:00.000
282,92051359274	2025-10-30 21:00:00.000
288,604546549828	2025-10-30 22:00:00.000
295,798864757013	2025-10-30 23:00:00.000

$$E_{\text{сТ}} = 295,798864757013 - 88,0774477754778 \approx 207,72 \text{ кВт·год.}$$

Це значення приймалося як базове для подальших порівнянь.

Щоб змодельовати інші інтервали дискретизації, з початкового ряду були вибрані значення з кроком 2, 3 та 6 годин. Для кожного випадку добова енергія визначалася як сума приростів між сусідніми точками. Наприклад, для кроку 2 години перші два інтервали розраховувалися так:

$$\Delta E_{0-2} = 100,957576280457 - 88,0774477754778 = 12,88 \text{ кВт}\cdot\text{год}$$

$$\Delta E_{2-4} = 121,285264621723 - 100,957576280457 = 20,33 \text{ кВт}\cdot\text{год}$$

Сумування всіх подібних приростів за добу дає:

$$E_{2h} \approx 203,7 \text{ кВт}\cdot\text{год},$$

що формує відносну похибку:

$$\delta_{2h} = |207,72 - 203,7| / 207,72 \approx 1,9 \, \%.$$

При збільшенні кроку до 3 годин перші інтервали стають ширшими, що приводить до значного згладжування динаміки навантаження. Для прикладу:

$$\Delta E_{0-3} = 111,94152558886 - 88,0774477754778 = 23,86 \text{ кВт}\cdot\text{год}$$

$$\Delta E_{3-6} = 135,344989926614 - 111,94152558886 = 23,40 \text{ кВт}\cdot\text{год}$$

Після обчислення всіх приростів було отримано:

$$E_{3h} \approx 198,47 \text{ кВт}\cdot\text{год},$$

похибка при цьому становить:

$$\delta_{3h} \approx 4,45 \, \%.$$

Для інтервалу 6 годин обчислення проводилися аналогічно, наприклад:

$$\Delta E_{0-6} = 135,344989926614 - 88,0774477754778 = 47,27 \text{ кВт}\cdot\text{год}$$

$$\Delta E_{6-12} = 201,687489302528 - 135,344989926614 = 66,34 \text{ кВт}\cdot\text{год}$$

Узагальнення результатів розрахунків показує, що годинний інтервал забезпечує найвищу точність оцінки спожитої енергії, а двогодинний може розглядатися як прийнятний компроміс між точністю та обсягом даних. Інтервали у три години та більше призводять до спотворення профілю навантаження та перевищення допустимих значень похибки для задач аналізу й комерційного обліку. З урахуванням проведених розрахунків рекомендовано застосовувати крок не більше двох годин для забезпечення достовірності обчислення активної енергії.

5.3 Висновки за розділом 5

Аналіз впливу кроку дискретизації показав, що точність визначення добового споживання активної енергії значно знижується при збільшенні інтервалу між вимірами. Оптимальним виявився крок не більше двох годин, що забезпечує допустиму похибку та зберігає динаміку навантаження. Модульне тестування серверної частини вебзастосунку підтвердило її стабільну та коректну роботу: маршрути API обробляють запити та повертають очікувані результати, взаємодія з базою даних відбувається без збоїв, а логіка створення планів і отримання даних працює відповідно до очікувань. Результати робіт гарантують надійність системи та її готовність до використання.

ВИСНОВКИ

У процесі розробки програмного забезпечення було досягнуто поставлених цілей, які полягали у створенні програмно комплексу для автоматизованого опитування електролічильника з передачею зібраних даних на сервер і подальшою їх візуалізацією у вебінтерфейсі. Під час виконання роботи було реалізовано повний цикл створення системи – від аналізу предметної області та вибору технічних засобів до розробки, налагодження й тестування функціональних модулів.

У ході проєкту створено серверний застосунок на базі Node.js і Express.js, який забезпечує прийом, обробку та збереження даних у базі даних Microsoft SQL Server. Сервер реалізує REST API для взаємодії з клієнтською частиною та контролером, а також виконує перевірку, структурування та передавання результатів у форматі JSON.

На клієнтській стороні розроблено вебзастосунок із використанням фреймворку Vue.js, що забезпечує відображення отриманих показників у зручному для користувача вигляді. Інтерфейс передбачає перегляд історії зчитувань, планових значень, графіки енергоспоживання та можливість оновлення параметрів попереджень. Застосунок реалізовано як SPA, що забезпечує високу швидкість роботи та комфортну взаємодію користувача із системою.

Програмна частина опитування реалізована мовою C#, що забезпечує стабільний обмін даними з лічильником через Ethernet-з'єднання. Програма виконує періодичний опит лічильника за протоколом DLMS/COSEM, обробляє отримані дані та надсилає їх на сервер.

Було реалізовано базу даних, у якій зберігаються миттєві показники, планові значення та історія споживання. Структура бази розроблена таким чином, щоб забезпечити узгодженість даних, швидкий доступ і можливість масштабування в майбутньому.

Проведене тестування підтвердило стабільність роботи системи під час безперервного опитування лічильника, правильність передавання даних до сервера та коректне їх відображення у вебінтерфейсі. Усі основні функції виконуються відповідно до поставлених вимог, що підтверджує доцільність обраних рішень і технологій.

Використання сучасних інструментів Node.js, Vue.js та Microsoft SQL Server забезпечило високу продуктивність, гнучкість і масштабованість системи. Застосування асинхронної архітектури дозволило мінімізувати затримки при передачі даних і забезпечити стабільну роботу в режимі реального часу.

Розроблений застосунок може бути основою для подальшого вдосконалення, зокрема розширення аналітичних можливостей або інтеграції з іншими приладами обліку. Реалізація таких функцій дозволить створити повноцінну платформу моніторингу енергоспоживання для побутових і промислових користувачів.

Таким чином, виконана робота повністю відповідає поставленим завданням і демонструє практичну цінність створеного програмного продукту, який може бути використаний як основа для побудови комплексних систем автоматизованого енергомоніторингу.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Коцар, О. В. SMART-системи вимірювання, обліку та управління енерговикористанням / О. В. Коцар // Енергетика: економіка, технології, екологія. — 2018. — № 2. — С. 20–30.
2. Шпак, О. Л., Луців, П. Д., Калінчик, В. П., Шиянов, О. О. Автоматизована система комерційного обліку електроенергії ПАТ «Хмельницькобленерго» / О. Л. Шпак, П. Д. Луців, В. П. Калінчик, О. О. Шиянов // Енергетика: економіка, технології, екологія. — 2014. — Т. 2. — С. 45–54.
3. Гапон, Д. А., Гриб, О. Г., Карпалюк, І. Т., Рудевіч, Н. В. Автоматизовані системи обліку і якості електроенергії у системах електропостачання / Д. А. Гапон, О. Г. Гриб, І. Т. Карпалюк, Н. В. Рудевіч // Вісник Національного технічного університету «ХПІ». Серія: Енергетика: надійність та енергоефективність. — 2021. — № 2 (3). — С. 12–22.
4. U.S. Department of Energy. Advanced Metering Infrastructure and Customer Systems: SGIG Program Report / U.S. Department of Energy. — 2016. — [Electronic resource]. — Access mode: https://www.energy.gov/sites/prod/files/2016/12/f34/AMI%20Summary%20Report_09-26-16.pdf.
5. Kermani, M., et al. Intelligent Energy Management based on SCADA System in a Real Microgrid / M. Kermani, et al. // Renewable Energy. — 2021. — Vol. 170. — P. 1234–1247. — [Electronic resource]. — Access mode: <https://www.sciencedirect.com/science/article/pii/S0960148121003566>.
6. ISO 50001:2018 — Energy management systems — Requirements with guidance for use [Electronic resource]. — Access mode: <https://www.iso.org/standard/69426.html>.
7. ДСТУ ISO 50001:2014 — Системи енергетичного менеджменту. Вимоги та керівництво із застосування / Національний орган стандартизації України. — Київ: ДП «УкрНДНЦ», 2014.

8. Fuchs, H., Therkelsen, P., Miller, W. C., Siciliano, G., Sheaffer, P. ISO 50001-Based Energy Management Systems as a Practical Path for Decarbonization: Initial Findings from a Survey of Technical Assistance Cohort Participants / H. Fuchs, P. Therkelsen, W. Miller, G. Siciliano, P. Sheaffer // *Energies*. — 2023. — Vol. 16, № 14.
9. НЕК «Укренерго». Впровадження АСКОВЕ [Електрон. ресурс]. — Режим доступу: <https://www.ukrenergogoexport.com/uk/node/71>.
10. Siemens. Spectrum Power Advanced Distribution Management System (ADMS) [Electronic resource]. — Access mode: <https://www.siemens.com/us/en/products/energy/grid-software/operation/grid-control/advanced-distribution-management.html>.
11. Schneider Electric. EcoStruxure™ Power Monitoring Expert (PME) [Electronic resource]. — Access mode: <https://www.se.com/us/en/product-range/65404-ecostruxure-power-monitoring-expert/>.
12. Axelson J. Serial Port Complete: COM Ports, USB Virtual COM Ports, and Ports for Embedded Systems / J. Axelson. — 2nd ed. — Lakeview Research, 2007. — 400 p.
13. Texas Instruments. RS-485 Basics Series / Texas Instruments. — 2025 [Electronic resource]. — Access mode: <https://www.ti.com/lit/pdf/slla545>.
14. Rinaldi J. S., Marshall P. S. Industrial Ethernet, Third Edition / J. S. Rinaldi, P. S. Marshall. — ISA, 2019. — 512 p.
15. Heine G. GPRS: Gateway to Third Generation Mobile Networks / G. Heine. — Artech House, 2002. — 300 p.
16. Hoffman J. GPRS Demystified / J. Hoffman. — McGraw-Hill, 2002. — 280 p.
17. Sendin, A., Peña, I., Angueira, P. Strategies for Power Line Communications Smart Metering Network Deployment / A. Sendin, I. Peña, P. Angueira // *Energies*. — 2014. — Vol. 7, № 4, С. 2377–2420.
18. Коцар, О.В., Довгий, С. Підвищення продуктивності систем енергетичного менеджменту МСП через управління даними

енергомоніторингу // Енергетика: економіка, технології, екологія. — 2025. — № 2.

19. Lee, S., Jin, H., Nengroo, S. H., Doh, Y., Lee, C., Heo, T., Har, D. Smart metering system capable of anomaly detection by bi-directional LSTM autoencoder // — 2021.

20. O. Smith et al. Empowering smart city through smart grid communication and measurement technology // International Journal of Low-Carbon Technologies. — (Oxford Academic). — 2025 — 470 p.

21. Dahunsi, F. M., Olawumi, A. E., Ale, D. T., Sarumi, O. A. «A systematic review of data pre-processing methods and unsupervised mining methods used in profiling smart meter data» // AIMS Electronics and Electrical Engineering. — 2021. — T. 5, № 4, C. 284–314.

22. Gupta, R., Chaturvedi, K. T. «Adaptive Energy Management of Big Data Analytics in Smart Grids» // Energies. — 2023. — Vol. 16, № 16, article 6016.

23. Chicco, G., Domingo, M., Nieto, M., et al. «Data Consistency for Data-Driven Smart Energy Assessment» // Entropy. — 2021.

24. Schaub T. DLMS/COSEM Overview [Electronic resource]. – Access mode: https://www.ofgem.gov.uk/sites/default/files/docs/2011/03/smdg-dcg---meeting-1---dlms-overview-slides_0.pdf.

25. Ju, S. Hwan, Seo, H. Suk. Design Key Management System for DLMS/COSEM Standard-based Smart Metering / S. H. Ju, H. S. Seo // International Journal of Engineering and Technology. — 2018. — Vol. 7, № 3.34, C. 554–557.

26. Halvorsen T. Introduction to Data Communication with C# Programming [Electronic resource] Access mode: <https://www.halvorsen.blog/documents/teaching/courses/csharp/resources/OBD/DataCommunicationCsharpTutorial.pdf>.

27. Kimura, S., Imai, K. Fluent Session Programming in C# / S. Kimura, K. Imai // Electronic Proceedings in Theoretical Computer Science. — 2020. — Vol. 314, P. 61–75.

28. Kaimer, F., et al. Towards a Node.js-Based Software Architecture for Online Service Network Applications / F. Kaimer, // Procedia Computer Science. — 2018. — Vol. 138. — C. 583–590.
29. The Good and the Bad of Express.js Web Framework / N. E. Atori // AltexSoft. — 2024. — [Electronic resource]. — Access mode: <https://www.altexsoft.com/blog/expressjs-pros-and-cons/>.
30. Vue.js for Business: More Clients with a Fast Website / Asabix. — 2025. — [Electronic resource]. — Access mode: <https://asabix.com/blog/how-vue-js-helps/>.
31. Madyatmadja, E. D., Hakim, A. N., Sembiring, D. J. Performance testing on Transparent Data Encryption for SQL Server's reliability and efficiency / E. D. Madyatmadja, A. N. Hakim, D. J. Sembiring // Journal of Big Data. — 2021. — Vol. 8. — Article 134.
32. Li, J., König, A. C., Narasayya, V., Chaudhuri, S. Robust Estimation of Resource Consumption for SQL Queries Using Statistical Techniques / J. Li, A. C. König, V. Narasayya, S. Chaudhuri //— 2012.

ДОДАТОК А
Технічне завдання

Вступ

З ростом потреб у точному контролі та аналізі електроспоживання з боку промислових, комунальних та побутових об'єктів актуальним стає створення ефективних застосунків, що забезпечують автоматизований збір, обробку та диспетчеризацію енергетичних даних. У межах цього проєкту розглядається розробка вебзастосунку для збору, аналізу та керування даними спожитої електроенергії з використанням сучасних клієнтських і серверних технологій, орієнтованих на точність, масштабованість та зручність користування.

A.1 Підстава для розробки

Підставою для розробки є завдання на дипломну кваліфікаційну роботу на тему: «Дослідження та програмна реалізація вебзастосунку щодо збору, аналізу та диспетчеризації даних спожитої електроенергії. Research and Software Implementation of a Web Application for Collecting, Analyzing, and Managing Consumed Electricity Data», затверджене наказом Національного університету «Запорізька політехніка» № 447 від 30 вересня 2025 р.

A.2 Призначення розробки

Розробка спрямована на створення вебзастосунку, що забезпечує автоматизований збір даних електроспоживання з електрولیчильників, їх подальше структурування, аналіз та відображення у вигляді зручних графіків, таблиць і показників. Система повинна надавати можливість оперативного моніторингу стану енергоспоживання, формування архівів, виявлення аномалій та виконання диспетчерських функцій. Важливими є масштабованість, точність і надійність роботи, а також інтуїтивний інтерфейс для операторів і користувачів.

A.3 Основні вимоги до програми, що розробляється

A.3.1 Вимоги до функціональних характеристик

Програма повинна забезпечувати:

- автоматичний збір даних електролічильників через мережеві інтерфейси;
- обробку отриманих значень та їх запис у базу даних;
- побудову графіків, таблиць та звітів за вибраний період;
- формування історичних архівів;
- можливість перегляду поточних показників у реальному часі;
- диспетчеризацію сповіщення про відхилення.

A.3.2 Вимоги до інтерфейсу програми

Інтерфейс повинен бути інтуїтивно зрозумілим, з акцентом на швидку навігацію та наочність даних. Вебзастосунок має відповідати сучасним UX/UI стандартам та дозволяти зручно працювати зі складними наборами інформації.

A.3.3 Вимоги до надійності

Система повинна стабільно працювати при великих об'ємах даних та значному навантаженні. Надійність забезпечується використанням серверної архітектури з контролем помилок, резервною обробкою даних, а також тестуванням критично важливих функцій.

A.3.4 Умови експлуатації

Для роботи вебзастосунку необхідно стабільне інтернет-з'єднання та сучасний браузер. Серверна частина повинна функціонувати на обладнанні,

здатному обробляти запити від множини підключених лічильників та користувачів.

A.3.5 Вимоги до складу та параметрів технічний засобів

Для серверної частини:

- процесор від 1 ГГц, 4 ядра або більше;
- ОЗП від 2 ГБ;
- встановлене середовище Node.js та СУБД (наприклад, Microsoft SQL Server або аналогічна).

Для клієнтської частини (користувача):

- процесор від 1 ГГц, 2 ядра або більше;
- ОЗП: від 4 ГБ;
- дисплей: від 720p;
- сучасний браузер (Chrome, Edge, Firefox, Opera, Safari).

A.3.6 Вимоги до маркування і пакування

Маркування програми має включати її назву та версію. Пакування програмного забезпечення передбачає оптимізовані пакети для клієнтської та серверної частини. Також необхідно забезпечити наявність документації користувача та короткого опису функціональних можливостей для ознайомлення перед встановленням чи використанням.

ДОДАТОК Б
Текст програми

Б.1 Програма опитування лічильника

```
using Gurux.DLMS.Enums;
using Gurux.DLMS.Objects;
using Gurux.DLMS.Reader;
using Gurux.Net;
using System;
using System.Data.SqlClient;
using System.Text;
using System.Threading;

namespace DLMSPollingApp
{
    class Program
    {
        static string ipAddress = "127.0.0.1";
        static int port = 4060;
        static int clientAddress = 16;
        static int serverAddress = 1;
        static Authentication authentication = Authentication.None;
        static string password = "";

        static string connectionString =
            "Server=localhost;Database=EnergyDB;User
Id=sa;Password=12345;";
        static string tableName = "EnergyData";

        static int pollingInterval = 60000;
        static Timer pollingTimer;
```

```

static void Main(string[] args)
{
    pollingTimer = new Timer(_ => ReadMeterData(), null, 1000,
pollingInterval);
    Console.ReadLine();
}

static void ReadMeterData()
{
    try
    {
        var client = new Gurux.DLMS.Secure.GXDLMSSecureClient(
            true,
            clientAddress,
            serverAddress,
            authentication,
            password,
            InterfaceType.HDLC
        );

        using (GXNet media = new GXNet(NetworkType.Tcp, ipAddress,
port))
        {
            media.Open();
            GXDLMSReader reader = new GXDLMSReader(client, media,
System.Diagnostics.TraceLevel.Off, null, 1000);
            reader.InitializeConnection();

            GXDLMSObject obj =
GXDLMSClient.CreateObject(ObjectType.Data);

```

```

obj.LogicalName = "1.0.1.8.0.255";
object activeEnergy = reader.Read(obj, 2);

obj.LogicalName = "0.0.96.1.0.255";
object nameValue = reader.Read(obj, 2);
string meterName = nameValue is byte[] arr
    ? Encoding.ASCII.GetString(arr)
    : nameValue.ToString();

SaveCurrentValues(activeEnergy, meterName);

if (IsHourlyDataOutdated())
    ReadHourlyData(reader);

    reader.Close();
}
}
catch { }
}

static void SaveCurrentValues(object activeEnergy, string meterName)
{
    using (SqlConnection conn = new SqlConnection(connectionString))
    {
        conn.Open();

        string sql = $"@
        UPDATE {tableName}
        SET ActiveEnergy = @Energy,
            MeterName = @Name,

```

```

        UpdateTime = @Time
    WHERE Id = 1";

```

```

        SqlCommand cmd = new SqlCommand(sql, conn);
        cmd.Parameters.AddWithValue("@Energy", activeEnergy);
        cmd.Parameters.AddWithValue("@Name", meterName);
        cmd.Parameters.AddWithValue("@Time", DateTime.Now);

        cmd.ExecuteNonQuery();
    }
}

static bool IsHourlyDataOutdated()
{
    using (SqlConnection conn = new SqlConnection(connectionString))
    {
        conn.Open();

        string sql = @"
            SELECT TOP 1 Timestamp
            FROM HourlyArchive
            ORDER BY Timestamp DESC";

        SqlCommand cmd = new SqlCommand(sql, conn);
        object result = cmd.ExecuteScalar();

        if (result == null)
            return true;

        DateTime last = Convert.ToDateTime(result);
    }
}

```

```

        return (DateTime.Now - last).TotalHours >= 1;
    }
}

static void ReadHourlyData(GXDLMMSReader reader)
{
    GXDLMSObject profile =
GXDLMSClient.CreateObject(ObjectType.ProfileGeneric);
    profile.LogicalName = "1.0.99.1.0.255";

    object row = reader.Read(profile, 2);

    if (row is object[] rows)
    {
        using (SqlConnection conn = new
SqlConnection(connectionString))
        {
            conn.Open();

            foreach (object row in rows)
            {
                DateTime ts = DateTime.Now.AddHours(-1);
                double val = 0;

                string sql = @"
                    INSERT INTO HourlyArchive (Timestamp,
EnergyValue)
                    VALUES (@ts, @val)";

                SqlCommand cmd = new SqlCommand(sql, conn);

```

```

        cmd.Parameters.AddWithValue("@ts", ts);
        cmd.Parameters.AddWithValue("@val", val);

        cmd.ExecuteNonQuery();
    }
}
}
}
}
}
}
}

```

Б.2 Серверна частина вебзастосунку

```

const express = require("express");
const app = express();
const cors = require('cors');

```

```

const port = 3001;

```

```

const sql = require('mssql')

```

```

var config = {
  user: "test",
  password: "werystrong",
  database: 'electro_opros',
  server: 'DESKTOP-NM9BPD2\\SQLEXPRESS',
  driver: 'msnodesqlv8',
  options: {
    trustedConnection: true,

```

```

    encrypt: false,
    trustServerCertificate: true,
    enableArithAbort: true
  }
};

```

```

app.use(cors({
  origin: '*',
  methods: ['GET', 'POST', 'PUT', 'DELETE', 'OPTIONS'],
  allowedHeaders: ['Origin', 'X-Requested-With', 'Content-Type', 'Accept']
}));

```

```

app.use(express.json());

```

```

app.get("/getdat/:param", function (req, res) {
  (async () => {
    try {

```

```

      await sql.connect(config)

```

```

      const inst = `SELECT * from instans where id =${id}`

```

```

      const result_inst = await sql.query(inst)

```

```

      const plan = `SELECT * from [plan] where date
=${req.params['param']}`

```

```

      const result_plan = await sql.query(plan)

```

```

    const fact = `SELECT * FROM hours WHERE [data] >=
'${req.params.param} 00:00:00' AND [data] <= DATEADD(HOUR, 1,
'${req.params.param} 23:00:00')`;

    const result_fact = await sql.query(fact)
    const result = {
      instans: result_inst.recordset,
      plan: result_plan.recordset,
      fact: result_fact.recordset

    };

    res.send(result);

  } catch (err) {
    console.dir(err)
  }
}) ()

});

app.post('/warning', (req, res) => {

  (async () => {
    try {

      await sql.connect(config)

      const warn = `update instans set w_activ =
${req.body.w_activ},w_voltag = ${req.body.w_voltag} where id =1`

      await sql.query(warn)

```

```

    } catch (err) {
      console.dir(err)
    }
  })()

});

app.post('/plan', (req, res) => {

  (async () => {
    try {
      await sql.connect(config)
      var plan = `
UPDATE [plan] SET
  [00-01] = ${req.body.val[0].plan},
  [01-02] = ${req.body.val[1].plan},
  [02-03] = ${req.body.val[2].plan},
  [03-04] = ${req.body.val[3].plan},
  [04-05] = ${req.body.val[4].plan},
  [05-06] = ${req.body.val[5].plan},
  [06-07] = ${req.body.val[6].plan},
  [07-08] = ${req.body.val[7].plan},
  [08-09] = ${req.body.val[8].plan},
  [09-10] = ${req.body.val[9].plan},
  [10-11] = ${req.body.val[10].plan},
  [11-12] = ${req.body.val[11].plan},
  [12-13] = ${req.body.val[12].plan},
  [13-14] = ${req.body.val[13].plan},

```

```

[14-15] = ${req.body.val[14].plan},
[15-16] = ${req.body.val[15].plan},
[16-17] = ${req.body.val[16].plan},
[17-18] = ${req.body.val[17].plan},
[18-19] = ${req.body.val[18].plan},
[19-20] = ${req.body.val[19].plan},
[20-21] = ${req.body.val[20].plan},
[21-22] = ${req.body.val[21].plan},
[22-23] = ${req.body.val[22].plan},
[23-00] = ${req.body.val[23].plan},
[sum_pot] = ${req.body.sum}
WHERE [date] = '${req.body.date}' AND [id_pribor] = 1;
`;

```

```
const check = await sql.query(plan);
```

```

if(check.rowsAffected[0] === 0){
  plan = `insert into [plan]
([00-01]
,[01-02]
,[02-03]
,[03-04]
,[04-05]
,[05-06]
,[06-07]
,[07-08]
,[08-09]
,[09-10]
,[10-11]
,[11-12]

```

,[12-13]
 ,[13-14]
 ,[14-15]
 ,[15-16]
 ,[16-17]
 ,[17-18]
 ,[18-19]
 ,[19-20]
 ,[20-21]
 ,[21-22]
 ,[22-23]
 ,[23-00]
 ,[date]
 ,[id_pribor]
 ,[sum_pot]) values
 (\${req.body.val[0].plan}
 ,\${req.body.val[1].plan}
 ,\${req.body.val[2].plan}
 ,\${req.body.val[3].plan}
 ,\${req.body.val[4].plan}
 ,\${req.body.val[5].plan}
 ,\${req.body.val[6].plan}
 ,\${req.body.val[7].plan}
 ,\${req.body.val[8].plan}
 ,\${req.body.val[9].plan}
 ,\${req.body.val[10].plan}
 ,\${req.body.val[11].plan}
 ,\${req.body.val[12].plan}
 ,\${req.body.val[13].plan}
 ,\${req.body.val[14].plan}

```

    ,${req.body.val[15].plan}
    ,${req.body.val[16].plan}
    ,${req.body.val[17].plan}
    ,${req.body.val[18].plan}
    ,${req.body.val[19].plan}
    ,${req.body.val[20].plan}
    ,${req.body.val[21].plan}
    ,${req.body.val[22].plan}
    ,${req.body.val[23].plan}
    ,`${req.body.date}`
    ,1
    ,${req.body.sum}
  );`
  await sql.query(plan)
}

} catch (err) {
  console.dir(err)
}
})()

});

app.listen(port);

```

Б.3 Клієнтська частина вебзастосунку

```

<template>
  <div style="width: 100%;" id="app">

```

```

    <h1 style=" text-align: center;">Моніторинг використаної енергії
</h1>

<div style="display: flex;justify-content: center;">
    <div style="
        display: flex;
        align-items: center;
    ">
        <input          type="date"          v-model="data">    <button
@click="requestinst(data),p_form = false"> ok</button>
    </div>

</div>

<div style="width: 100%; height: 400px;">

    <canvas ref="canvas"></canvas>
</div>
<br>
<table v-if="!w_form" style="width: 100%;">

    <tr style="font-weight:bold;font-size: 16pt;">
        <td>Миттєві значення</td>
        <td>Отсанье оновлення: {{ this.instans[0].date }}</td>
    </tr>

    <tr
        :style="{backgroundColor:    (this.instans[0].Phas_voltag    >=
this.instans[0].w_voltag) ? 'red' : 'white' }">
        <td>Напруга</td>
        <td>{{ this.instans[0].Phas_voltag }} B</td>
    </tr>

```

```

        <tr
            :style="{backgroundColor: (this.instans[0].Inst_activ
this.instans[0].w_activ) ? 'red' : 'white' }" >
            <td>Сила струму</td>
            <td>{{ this.instans[0].Inst_activ }} A</td>
        </tr>

</table>

<form v-if="w_form" @submit.prevent="updatewarning(),w_form =
false">
    <table style="width: 100%;">

        <tr style="font-weight:bold;font-size: 16pt;">
            <td>Миттєві значення</td>
            <td>Встановлення ліміту</td>
        </tr>
        <tr >
            <td>Напруга</td>
            <td>
                <input type="number" required min="1" step="0.01" v-
model="instans[0].w_voltag" /> B</td>
        </tr>
        <tr >
            <td>Сила струму</td>
            <td>
                <input type="number" required min="1" step="0.01" v-
model="instans[0].w_activ" /> A</td>
        </tr>

    </table>

    <button style="width: 100%;" type="submit">Зберегти</button>
</form>

```

```

<br>
<div v-if="!w_form && !p_form">
  <button style="width: 50%;" @click="w_form = true" >Змінити ліміти
напруги/сили струму</button>
  <button style="width: 50%;" @click="p_form = true " :style="{
    pointerEvents: (data < new Date().toISOString().slice(0, 10)) ? 'none' :
'auto',
    opacity: (data < new Date().toISOString().slice(0, 10)) ? '0.5' : '1'
  }" >Задати план на добу</button>
</div>
<form @submit.prevent="setupdplan(),updateChart(),p_form = false">
  <div v-if="p_form">
    <button style="width: 50%;" @click="p_form = false"
>Відмінити</button>
    <button style="width: 50%;" >Зберегти</button>
  </div>
  <br>
  <br>
  <table style="width: 100%;">

    <tr style="font-weight:bold;font-size: 16pt;">
      <td >Дата</td>
      <td >"Час,ГГ - ГГ"</td>
      <td >Витрати</td>
      <td>За планом</td>
    </tr>
    <tr v-for="(v, index) in val " :key="index">
      <td>{{ formatted() }}</td>
      <td >{{ v.tm }}</td>

```

```

        <td :style="{ backgroundColor: (v.plan !== '-' && Number(v.fact) >
Number(v.plan)) ? 'red' : 'white' }"> {{v.fact}}</td>
        <td><input v-if="p_form" type="number" required min="0" v-
model="v.plan" step="0.01"> <span v-if="!p_form"> {{v.plan}}</span></td>

```

```

</tr>

```

```

<tr style="font-weight:bold;font-size: 16pt;">

```

```

    <td colspan="2">Заголовок</td>

```

```

    <td >{{ sumo2.toFixed(2) }}</td>

```

```

    <td >{{ sumdom.toFixed(2) }}</td>

```

```

</tr>

```

```

</table>

```

```

</form>

```

```

<br><br>

```

```

</div>

```

```

</template>

```

```

<script>

```

```

import axios from "axios";

```

```
import { Line } from 'vue-chartjs'
```

```
var curentdate= new Date();
```

```
export default {
```

```
  name: 'App',
```

```
  extends: Line,
```

```
  mounted() {
```

```
    this.requestinst(this.data)
```

```
    this.intervalId = setInterval(() => {
```

```
      this.request(this.data);
```

```
    }, 600000);
```

```
  },
```

```
  beforeDestroy() {
```

```
    if (this.intervalId) {
```

```
      clearInterval(this.intervalId);
```

```
    }
```

```
  },
```

```
  components: {
```

```
  },
```

```
  data() {
```

```
    return {
```

```
      instans:[{id: 0 ,date:" ,Inst_activ:" ,Phas_voltag:" ,w_activ:" ,w_voltag:"}],
```

```

val: [
  { dt: '01', tm: '00-01', fact: '-', plan: '-' },
  { dt: '02', tm: '01-02', fact: '-', plan: '-' },
  { dt: '03', tm: '02-03', fact: '-', plan: '-' },
  { dt: '04', tm: '03-04', fact: '-', plan: '-' },
  { dt: '05', tm: '04-05', fact: '-', plan: '-' },
  { dt: '06', tm: '05-06', fact: '-', plan: '-' },
  { dt: '07', tm: '06-07', fact: '-', plan: '-' },
  { dt: '08', tm: '07-08', fact: '-', plan: '-' },
  { dt: '09', tm: '08-09', fact: '-', plan: '-' },
  { dt: '10', tm: '09-10', fact: '-', plan: '-' },
  { dt: '11', tm: '10-11', fact: '-', plan: '-' },
  { dt: '12', tm: '11-12', fact: '-', plan: '-' },
  { dt: '13', tm: '12-13', fact: '-', plan: '-' },
  { dt: '14', tm: '13-14', fact: '-', plan: '-' },
  { dt: '15', tm: '14-15', fact: '-', plan: '-' },
  { dt: '16', tm: '15-16', fact: '-', plan: '-' },
  { dt: '17', tm: '16-17', fact: '-', plan: '-' },
  { dt: '18', tm: '17-18', fact: '-', plan: '-' },
  { dt: '19', tm: '18-19', fact: '-', plan: '-' },
  { dt: '20', tm: '19-20', fact: '-', plan: '-' },
  { dt: '21', tm: '20-21', fact: '-', plan: '-' },
  { dt: '22', tm: '21-22', fact: '-', plan: '-' },
  { dt: '23', tm: '22-23', fact: '-', plan: '-' },
  { dt: '00', tm: '23-00', fact: '-', plan: '-' },
],
w_form: false,
p_form: false,
sumdom:    0,
sumo2:     0,

```

```
sumdomc:    0,  
sumo2c:     0,  
intervalId: null,  
date:       new Date(),  
data:       curentdate.toISOString().split('T')[0],  
  
};  
  
},  
methods: {  
  
  updateChart () {  
    const labels = this.val.map(item => item.tm)  
    const plan = this.val.map(item => Number(item.plan))  
    const fact = this.val.map(item => Number(item.fact))  
  
    if (this.$data._chart) {  
      this.$data._chart.data.labels = labels  
      this.$data._chart.data.datasets[0].data = fact  
      this.$data._chart.data.datasets[1].data = plan  
      this.$data._chart.update()  
    } else {  
  
      this.renderChart(  
        {  
          labels,  
          datasets: [  
  
            {  
              label: 'Факт',  
              borderColor: 'rgb(255, 167, 38)',
```

```

        backgroundColor: 'rgba(255, 167, 38, 0.25)',
        data: fact
    },
    {
        label: 'План',
        borderColor: 'rgb(100, 181, 246)',
        backgroundColor: 'rgba(100, 181, 246, 0.25)',
        data: plan
    },
]
},
{
    responsive: true,
    maintainAspectRatio: false
}
)
}
},

```

```

formatted(){
    const parts = this.data.split('-');
    return `${parts[2]}.${parts[1]}.${parts[0]}`
},

```

```

zero(){
    this.instans = [{id: 0
, date: ", Inst_activ: ", Phas_voltag: ", w_activ: ", w_voltag: " }];

```

```

    this.val = [
      { dt: '01', tm: '00-01', fact: '-', plan: '-' },
      { dt: '02', tm: '01-02', fact: '-', plan: '-' },
      { dt: '03', tm: '02-03', fact: '-', plan: '-' },
      { dt: '04', tm: '03-04', fact: '-', plan: '-' },
      { dt: '05', tm: '04-05', fact: '-', plan: '-' },
      { dt: '06', tm: '05-06', fact: '-', plan: '-' },
      { dt: '07', tm: '06-07', fact: '-', plan: '-' },
      { dt: '08', tm: '07-08', fact: '-', plan: '-' },
      { dt: '09', tm: '08-09', fact: '-', plan: '-' },
      { dt: '10', tm: '09-10', fact: '-', plan: '-' },
      { dt: '11', tm: '10-11', fact: '-', plan: '-' },
      { dt: '12', tm: '11-12', fact: '-', plan: '-' },
      { dt: '13', tm: '12-13', fact: '-', plan: '-' },
      { dt: '14', tm: '13-14', fact: '-', plan: '-' },
      { dt: '15', tm: '14-15', fact: '-', plan: '-' },
      { dt: '16', tm: '15-16', fact: '-', plan: '-' },
      { dt: '17', tm: '16-17', fact: '-', plan: '-' },
      { dt: '18', tm: '17-18', fact: '-', plan: '-' },
      { dt: '19', tm: '18-19', fact: '-', plan: '-' },
      { dt: '20', tm: '19-20', fact: '-', plan: '-' },
      { dt: '21', tm: '20-21', fact: '-', plan: '-' },
      { dt: '22', tm: '21-22', fact: '-', plan: '-' },
      { dt: '23', tm: '22-23', fact: '-', plan: '-' },
      { dt: '00', tm: '23-00', fact: '-', plan: '-' },
    ];
  },

```

```

requestinst(data){
  this.zero();

```

```
axios.get(`http://localhost:3001/getdat/${data}`).then((Response) => {
```

```

    this.instans[0].id = Response.data.instans[0].id
    this.instans[0].Inst_activ = Response.data.instans[0].Inst_activ
    this.instans[0].Phas_voltag = Response.data.instans[0].Phas_voltag
    this.instans[0].w_activ = Response.data.instans[0].w_activ
    this.instans[0].w_voltag = Response.data.instans[0].w_voltag
    this.instans[0].date = Response.data.instans[0].date
    try {
```

```

let vitrat = Response.data.fact[0].energy;
Response.data.fact.forEach(element => {
    let curent = element.data;
    let hour = curent.split("T")[1].slice(0,2);
    const indx = this.val.findIndex(item => item.tm.split('-')[1] === hour);
    this.val[indx].fact = (element.energy - vitrat).toFixed(2);
    vitrat = element.energy;
});
```

```

} catch (error) {
    console.log("no data")
}
try {
    for (let index = 1; index <25; index++) {
```

```

        this.val[index-1].plan =
```

```
Object.values(Response.data.plan[0])[index]
```

```

    }
    } catch (error) {
```

```

        console.log("no data")
    }

    this.updateChart()
    });

},

updatewarning(){
    axios.post('http://localhost:3001/warning',{w_activ:
this.instans[0].w_activ,w_voltag: this.instans[0].w_voltag})
    },
    setupdplan(){
        axios.post('http://localhost:3001/plan',{val:  this.val,date:  this.data,sum:
this.sumdom})
    }

}

}

</script>

```

ДОДАТОК В
Слайди презентації

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
дипломна кваліфікаційна робота магістра
на тему:

ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБЗАСТОСУНКУ ЩОДО ЗБОРУ,
АНАЛІЗУ ТА ДИСПЕТЧЕРИЗАЦІЇ ДАНИХ СПОЖИТОЇ ЕЛЕКТРОЕНЕРГІЇ

Виконав студент групи
КНТ-124м

Кирило ЩЕРБАКОВ

Керівник
к.т.н доцент

Михайло КОЦУР

Рисунок В.1 – Слайд 1

Актуальність роботи



Рисунок В.2 – Слайд 2

МЕТА ТА ЗАВДАННЯ ДИПЛОМНОЇ РОБОТИ

Метою роботи є підвищення швидкості опрацювання отриманих даних показів електролічильників за протоколом DLMS/COSEM шляхом розроблення програмного забезпечення, яке дозволить надавати користувачу доступ до актуальних та архівних значень.

Завдання роботи:

- провести аналіз існуючих програмно-апаратних засобів для автоматизації збору даних з лічильників електроенергії;
 - спроектувати архітектуру програмно комплексу;
 - розробити програмне забезпечення для прийому, обробки та передачі даних;
- здійснити тестування розробленого програмного забезпечення та перевірити його ефективності.

3

Рисунок В.3 – Слайд 3

Аналіз програм аналогів

Критерій порівняння	Назва програми		
	АСКОЕ НЕК «Укренерго»	Siemens Spectrum Power	Schneider EcoStruxure PME
Призначення	Комерційний облік, балансування	Диспетчеризація	аналіз споживання
Сумісність з обладнанням	+/- (лічильники стандартизованих типів)	+ (різні виробники, Smart Grid)	+/- (краще з обладнанням Schneider)
Інтеграція	+ (MMS, SCADA)	+ (Smart Grid, мікромережі)	+ (BMS, IoT, прилади різних виробників)
Вимоги до системи	Високі (мережа, сервери)	Дуже високі (потужні сервери, резервування)	Середні (може працювати локально чи в хмарі)
Вартість впровадження	Висока	Дуже висока	Середня
Віддалений доступ	+/- (через диспетчерські центри)	+ (повний віддалений моніторинг)	+ (хмара, мобільний доступ)
Гнучкість налаштувань	Низька	Висока	Висока

4

Рисунок В.4 – Слайд 4

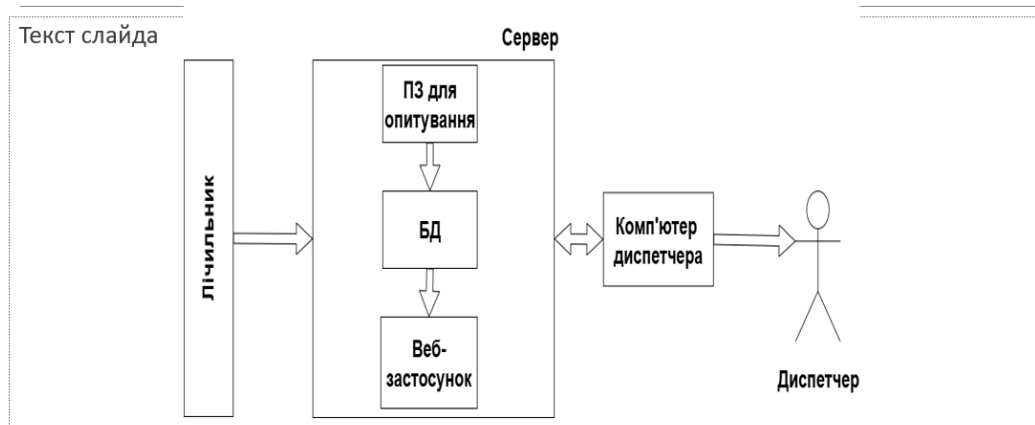
Аналіз стандартів передачі даних

Характеристика	RS-232	RS-485	Ethernet	PLC	GPRS
Дальність передачі	До 15 м	До 1200 м	до 100 м (без повторювачів)	До кількох км у мережі живлення	Обмежена зоною покриття
Тип з'єднання	Точка-точка	Багатоточкове (до 32 пристроїв)	Дротове, мережеве	Через електромережу	Через мобільну мережу
Швидкість передачі	До 115,2 Кбіт/с	До 10 Мбіт/с	до 1 Гбіт/с і більше	Від кількох до сотень Кбіт/с	До 85,6 Кбіт/с
Завдастійкість	Низька	Висока (диференціальна передача)	Висока	Середня/залежить від стану мережі	Середня, залежить від мережі
Основні переваги	Простота, низька вартість	Велика відстань, надійність	Висока швидкість, інтеграція з TCP/IP	Використання існуючої мережі	<u>Дистанційність</u> , глобальне покриття

5

Рисунок В.5 – Слайд 5

ПРОЄКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ



6

Рисунок В.6 – Слайд 6

Обґрунтування вибору мови програмування для системи опитування

Критерій порівняння мов програмування	Назва мови		
	<u>Python</u>	C#	C++
Швидкодія	+-	+-	+
Доступ до апаратних ресурсів (послідовні порти, мережа)	+-	+	+
Кросплатформність	+	+-	+
Наявність бібліотек для взаємодії з лічильниками	+-	+	+
Оптимальність для промислових застосувань	+-	+	+
Зручність розробки та підтримки коду	+-	+	-

7

Рисунок В.7 – Слайд 7

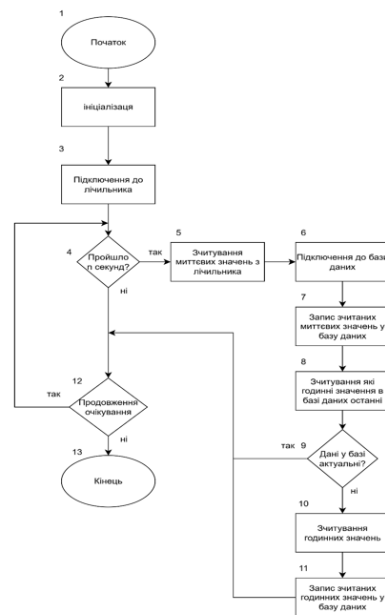
Обґрунтування вибору стеку для вебзастосунка

Критерій порівняння мов програмування	Назва мови		
	Node.js + Vue.js	<u>Django</u> + <u>React</u>	ASP.NET <u>Core</u> + <u>Blazor</u>
Швидкодія	+	+-	+
Масштабованість	+	+	+
Реактивний інтерфейс	+	+-	+-
Швидкість розробки	+	+-	+-
Можливості розширення	+	+	+-

8

Рисунок В.8 – Слайд 8

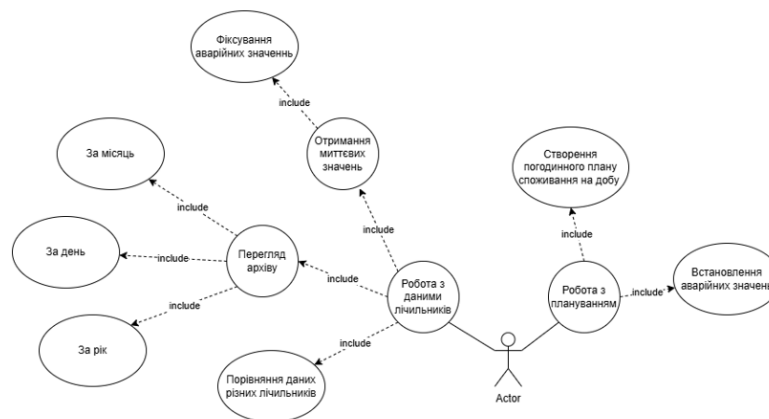
Схема алгоритму функціонування серверного додатку опитування лічильника



9

Рисунок В.9 – Слайд 9

Діаграма основних можливостей застосунку



10

Рисунок В.10 – Слайд 10

Демонстрація роботи

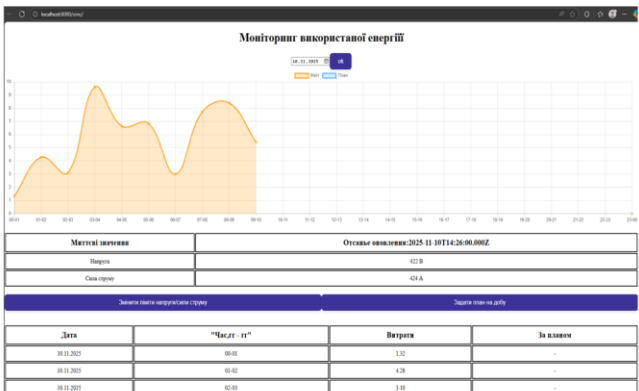


Рисунок В.11 – Слайд 11

Висновки

- Результатом виконання дипломної роботи стало створення програмного рішення для забезпечення систем обліку електроспоживання.
- Цінність розробки полягає у високій точності збору інформації, зменшення ручних операцій, можливість масштабування та інтеграції з іншими приладами. Система забезпечує стабільний обмін даними та підвищує ефективність контролю електроспоживання.
- У створеного рішення багато можливостей розширення функціоналу від додавання більшої кількості лічильників , збору додаткових показників , та до можливості інтеграції у системи автоматизації або збору інших енергоресурсів

Рисунок В.12 – Слайд 12