КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
імені ТАРАСА ШЕВЧЕНКА
Факультет інформаційних технологій
Кафедра прикладних інформаційних систем

122 «Комп'ютерні науки»
(шифр і назва спеціальності)

«Прикладне програмування»
(назва освітньої програми)

Кваліфікаційна робота бакалавра

на тему: «Веб-застосунок системи моніторингу рівня балансу споживання активних споживачів в Smart Grid мережах»

Виконав _____
(Підпис)
Афанасьєв Антон Антонович
(прізвище, ім'я, по батькові)

Керівник _Зайцев Євген Олександрович
(прізвище, ім'я, по батькові)
_____ д.т.н.,с.н.с_____
(науковий ступінь, звання)

_____
(Резолюція «До захисту»)

Унікальність тексту – 93%

Попередній захист:

_____
(Висновок: "До захисту в екзаменаційній комісії ")
Завідувач кафедри _____ Плескач В.Л.
(Підпис) (Прізвище, ініціали) (Дата)

Київ – 2024

КАЛЕНДАРНИЙ ПЛАН ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ БАКАЛАВРА

№з/п	Назва етапів кваліфікаційної роботи бакалавра	Термін виконання етапів кваліфікаційної роботи бакалавра	Відмітка про виконання
1.	Вибір теми та наукового керівника кваліфікаційної роботи бакалавра	01.11.2023	Виконано
2.	Видача завдання кваліфікаційної роботи бакалавра	15.11.2023	Виконано
3.	Настановча групова співбесіда з питань кваліфікаційної роботи бакалавра	27.11.2023	Виконано
4.	Затвердження плану кваліфікаційної роботи бакалавра	01.12.2023	Виконано
5.	Підбір та вивчення літературних та інших джерел з теми дослідження	08.12.2023	Виконано
6.	Підготовка і подання науковому керівнику першого варіанту I розділу роботи	31.12.2023	Виконано
7.	Підготовка і подання науковому керівнику першого варіанту II розділу роботи	31.01.2024	Виконано
8.	Підготовка і подання науковому керівнику першого варіанту III розділу роботи	29.03.2024	Виконано
9.	Подання роботи у першому варіанті	**29.04.2024**	Виконано
10.	Оформлення пояснювальної записки кваліфікаційної роботи бакалавра	02.05.2024	Виконано
11.	Подання кваліфікаційної роботи бакалавра на попередній захист	**15.05.2024**	Виконано
12.	Врахування зауважень керівника і подання роботи в остаточному варіанті (з відповідним висновком про допуск) на кафедру	27.05.2024	Виконано
13.	Затвердження роботи в цілому (підготовка письмового відгуку керівника, письмова рецензія на бакалаврської роботу)	13.06.2024	Виконано
14.	Захист кваліфікаційної роботи бакалавра	17.06.2024	

Здобувач вищої освіти _____
(підпис)

Керівник _____
(підпис)

ВІДОМІСТЬ КВАЛІФІКАЦІЙНОЇ РОБОТИ БАКАЛАВРА

Складові частини дипломної роботи	Обсяг, арк.
Титульний аркуш	1
Завдання до дипломної роботи	1
Календарний план дипломної роботи	1
Відомість дипломної роботи	1
Анотація	2
Анотація (іноземною мовою - англійською)	1
Зміст	1
Перелік скорочень, умовних позначень, термінів	0
Вступ	2
1	17
2	23
3	10
Висновки	1
Перелік використаних джерел	4
Додатки	3

<table>
<tr><td></td><td></td><td></td><td></td><td colspan="3">ДП ХХХХ 00.000.00</td></tr>
<tr><td></td><td>ПІБ</td><td>Підп.</td><td>Дата</td><td colspan="3"></td></tr>
<tr><td>Розробн.</td><td>Афанасьєв А.А.</td><td></td><td></td><td rowspan="4">Відомість дипломної роботи</td><td>Лист</td><td>Листів</td></tr>
<tr><td>Керівн.</td><td>Зайцев Є.О.</td><td></td><td></td><td></td><td>67</td></tr>
<tr><td>Н/контр.</td><td></td><td></td><td></td><td colspan="2" rowspan="2"></td></tr>
<tr><td>Зав. каф.</td><td>Плескач В.Л.</td><td></td><td></td></tr>
</table>

АНОТАЦІЯ(РЕФЕРАТ)

Дипломна робота: 67 с., 20 рис., 4 табл., 53 джерела, 3 дод.

Метою дослідження є ефективний моніторинг рівня балансу споживання активних споживачів у Smart Grid мережах на основі розробленого веб-застосунку.

Для досягнення мети роботи потрібно вирішити такі **завдання**:

- Дослідити особливості Smart Grid мереж.
- Провести аналітичний огляд теоретичних основ побудови систем моніторингу рівня балансу споживання активних споживачів.
- Проаналізувати програмно-архітектурні рішення задля розробки веб-застосунків.
- Розробити та впровадити веб-застосунок системи моніторингу рівня балансу споживання активних споживачів в Smart Grid мережах.

Об'єкт дослідження - процес виробництва та споживання електричної енергії.

Предмет дослідження - засоби створення веб-застосунку системи моніторингу.

Методи дослідження:

- Метод аналізу, який використано задля дослідження ринку на предмет наявності схожих систем;
- Описовий метод, який використано задля формування алгоритму розробки та моделювання проекту ;
- Метод порівняння, який використано задля визначення переваг та недоліків існуючих альтернатив;
- Метод наукового моделювання, який використано задля формування діаграм та моделей.

Практичне значення отриманих результатів полягає в тому, що розроблений веб-застосунок допоможе активним споживачам зручно відстежувати та аналізувати ключові показники свого господарства. Що в

результаті надасть їм змогу вносити відповідні зміни для підвищення ефективності виробництва та споживання.

Ключові слова: веб-застосунок, Smart Grid , активний споживач, система моніторингу.

ANOTATION (ABSTRACT)

Thesis: 67 pp., 20 fig., 4 tables, 53 sources, 3 appendices.

The aim of the study is effective monitoring of the level of consumption balance of active consumers in Smart Grid networks based on the developed web application.

To achieve the goal of the work, you need to solve the following **tasks**:

-To study the features of Smart Grid networks.

-To conduct an analytical review of the theoretical foundations of building systems for monitoring the level of consumption balance of active consumers.

-Analyze software and architectural solutions for developing web applications.

-To develop and implement a web application for monitoring the level of consumption balance of active consumers in Smart Grid networks.

Object of study is the process of electricity production and consumption.

Subject of study - means of creating a web application of the monitoring system.

Research methods:

- The analysis method used to study the market for the presence of similar systems;

- Descriptive method, which was used to form an algorithm for the development and modeling of the project;

- Comparison method used to determine the advantages and disadvantages of existing alternatives;

- The method of scientific modeling, which was used to create diagrams and models.

The practical significance of the results obtained is that the developed web application will help active consumers to conveniently monitor and analyze key indicators of their farm. As a result, they will be able to make appropriate changes to improve the efficiency of production and consumption.

Keywords: web application, Smart Grid, active consumer, monitoring system.

ЗМІСТ

ВСТУП 8
РОЗДІЛ 1 ТЕОРЕТИЧНІ ОСНОВИ ПОБУДОВИ СИСТЕМ МОНІТОРИНГУ РІВНЯ БАЛАНСУ СПОЖИВАННЯ АКТИВНИХ СПОЖИВАЧІВ В SMART GRID МЕРЕЖАХ 10
1.1 Дослідження особливостей Smart Grid мереж 10
1.2 Аналіз архітектури системи 15
1.3 Огляд основних потреб активних споживачів 18
1.4 Аналіз існуючих рішень для моніторингу рівня балансу споживання активних споживачів 20
1.5 Постановка задачі 25
1.6 Висновки до розділу 26
РОЗДІЛ 2 АНАЛІЗ ПРОГРАМНО-АРХІТЕКТУРНИХ РІШЕНЬ ЗАДЛЯ РЕАЛІЗАЦІЇ ВЕБ-ЗАСТОСУНКІВ 27
2.1 Огляд базової структури веб-застосунку 27
2.2 Огляд інструментів для створення клієнтської частини 28
2.3 Огляд інструментів для створення серверної частини 35
2.4 Висновки до розділу 48
РОЗДІЛ 3 ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБ-ЗАСТОСУНКУ СИСТЕМИ МОНІТОРИНГУ РІВНЯ БАЛАНСУ СПОЖИВАННЯ АКТИВНИХ СПОЖИВАЧІВ В SMART GRID МЕРЕЖАХ 49
3.1 Проектування веб-застосунку 49
3.2 Структура серверної частини веб-застосунку 49
3.3 Структура клієнтської частини веб-застосунку 52
3.4 Результат роботи 54
3.5 Висновки до розділу 58
ВИСНОВКИ 59
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 60
ДОДАТКИ 65

ВСТУП

В останні десятиліття споживачі електроенергії все більше починають не лише споживати а й виробляти електроенергію. В першу чергу це відбувається за допомогою домашніх сонячних електростанцій, а також інших відновлювальних джерел енергії. Проте така генерація не є стабільною, як і споживання саме тому активним споживачам необхідно мати ефективний інструмент для моніторингу.

Актуальність цієї теми полягає у постійному розвитку інтелектуальних мереж Smart Grid та необхідності ефективного моніторингу рівня балансу споживання електроенергії активними споживачами. З високою динамікою змін в енергетичній системі та зростаючою роллю розумних технологій, веб-застосунок стає ключовим елементом забезпечення ефективного використання електроенергії та забезпечення стабільності Smart Grid мережі.

Метою дослідження є ефективний моніторинг рівня балансу споживання активних споживачів у Smart Grid мережах на основі розробленого веб-застосунку.

Для досягнення мети роботи потрібно вирішити такі **завдання**:

- Дослідити особливості Smart Grid мереж.
- Провести аналітичний огляд теоретичних основ побудови систем моніторингу рівня балансу споживання активних споживачів.
- Проаналізувати програмно-архітектурні рішення задля розробки веб-застосунків.
- Розробити та впровадити веб-застосунок системи моніторингу рівня балансу споживання активних споживачів в Smart Grid мережах.

Об’єкт дослідження - процес виробництва та споживання електричної енергії.

Предмет дослідження - засоби створення веб-застосунку системи моніторингу.

Методи дослідження:

- Метод аналізу, який використано задля дослідження ринку на предмет наявності схожих систем;
- Описовий метод, який використано задля формування алгоритму розробки та моделювання проекту ;
- Метод порівняння, який використано задля визначення переваг та недоліків існуючих альтернатив;
- Метод наукового моделювання, який використано задля формування діаграм та моделей..

Практичне значення отриманих результатів полягає в тому , що розроблений веб-застосунок допоможе активним споживачам зручно відстежувати та аналізувати ключові показники свого господарства. Що в результаті надасть їм змогу вносити відповідні зміни для підвищення ефективності виробництва та споживання.

Структура роботи:

Кваліфікаційна робота бакалавра складається зі вступу, трьох розділів, розподілених на підрозділи та висновку.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ОСНОВИ ПОБУДОВИ СИСТЕМ МОНІТОРИНГУ РІВНЯ БАЛАНСУ СПОЖИВАННЯ АКТИВНИХ СПОЖИВАЧІВ В SMART GRID МЕРЕЖАХ

1.1 Дослідження особливостей Smart Grid мереж

Створення електричних мереж змінного струму стало справжньою революцією другої половини 19-го століття.Тоді у таких мереж була одна мета: зв’язати електростанції із кінцевими споживачами, так щоб майже у кожної людини був доступ до електроенергії. І цього вдалося досягти, завдяки перетворенням електроенергії з використанням трансформаторів стало можливим забезпечити передачу електроенергії на дуже велику відстань від місця генерації, що врешті дозволило забезпечити електроенергією 89.6% населення Землі.

Схема передачі електроенергії з використанням звичайної електричної мережі змінного струму наведена на рис. 1.1.

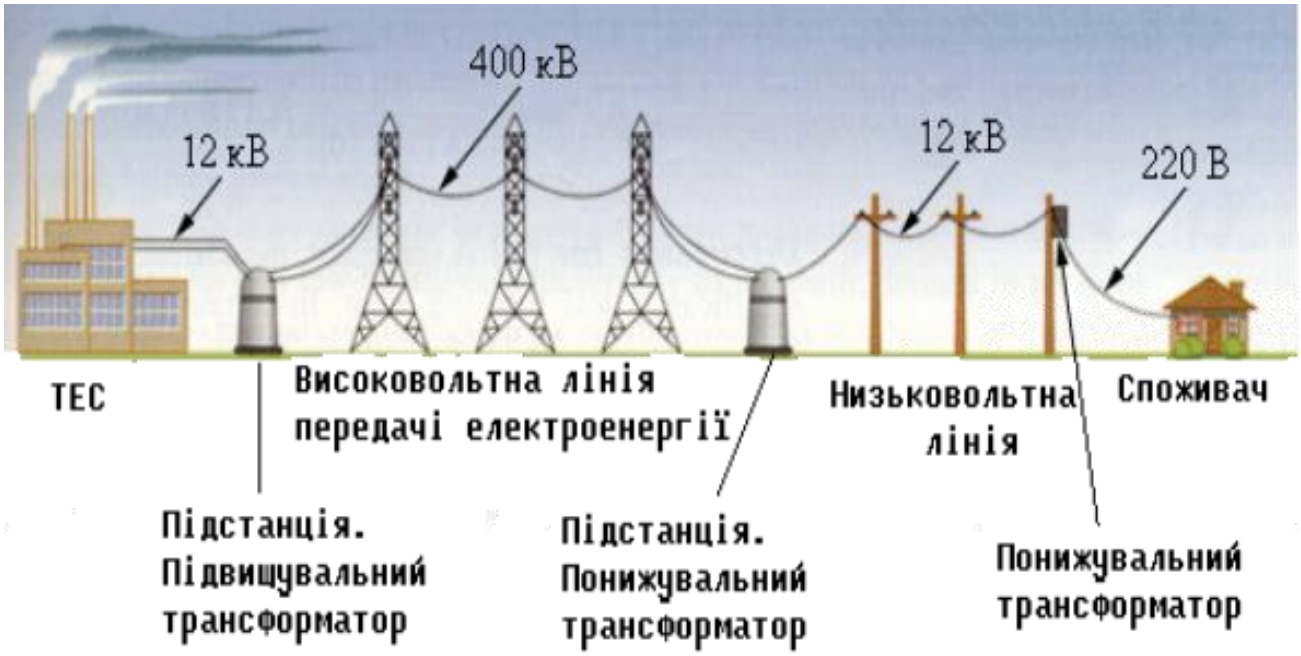

Рисунок 1.1 - Схема передачі електроенергії з використанням звичайної електричної мережі змінного струму

Проте незважаючи на практичність електричних мереж змінного струму, вони мали чотири основні проблеми:

- Великі втрати енергії.
- Обмежена ефективність використання ресурсів.
- Нестабільність мережі.
- Складне управління.

Довгий час енергетикам планомірно та ефективно вдавалося вирішувати проблему великої втрати енергії, оскільки постійно з'являлися нові технології, що дозволяли зменшувати опір дротів та проводити трансформацію більш ефективно [1, с. 20-25].

Для вирішення проблеми обмеженої ефективності використання ресурсів відбувалося впровадження енергоефективних технологій, таких як використання більш ефективного обладнання і систем енергозбереження. Також проводилися інформаційні кампанії щодо раціонального споживання енергії.

Для вирішення проблеми нестабільності мережі використовувалися різні методи регулювання, такі як встановлення компенсаційних пристроїв для збалансування напруги і частоти у мережі, а також підтримка за допомогою ручного керування з боку операторів мережі.

Для вирішення проблеми складного управління здійснювалися спроби автоматизації процесів, які втім довго не мали значних успіхів.

Таким чином рішення трьох вищенаведених проблем довгий час не було ефективним через обмеженість технологій у сфері комунікації та автоматизації.

Тож ці проблеми фактично не вирішувалися до другої половини 20-го століття, доки не почався розвиток інформаційних технологій, які нарешті змогли допомогти з їхнім вирішенням.

Отже основні технології розумних мереж виникли в результаті ранніх спроб використання інформаційних технологій для керування, вимірювання та моніторингу. Вагомих результатів вдалося досягти у 1980-х роках коли була розвинута технологія автоматичного зчитування показників лічильників [2]. Тоді ж вона вперше була використана для моніторингу споживання енергії великих

клієнтів. Цей успіх став основою для іншого значного досягнення 1990-х років , а саме створення інтелектуального лічильника, який зберігає інформацію про те, як електроенергія використовувалась протягом різних часів доби [3]. Прогресивність технології досягалась перш за все завдяки забезпеченню моніторингу в режимі реального часу, через що його можна використовувати як інтерфейс для пристроїв швидкого реагування на попит.

На початку пристрої швидкого реагування на попит пасивно визначали навантаження на енергосистему, розраховуючи зміни частоти джерела живлення. Це відкривало великі можливості, наприклад отримана інформація дозволяла операторам коригувати робочий цикл таким пристроям, як промислові та побутові кондиціонери, холодильники та обігрівач, щоб уникнути запуску під час пікового навантаження мережі.

Крім того, приблизно у той самий час на початку 1990-х років, відбувся інший ривок у розвитку моніторингу та синхронізації глобальних мереж, який зробило американське агентство Bonneville Power Administration, яке розширило дослідження розумних мереж сенсорами, здатними проводити дуже швидкий аналіз аномалій якості електроенергії на дуже великих географічних масштабах. Кульмінацією цієї роботи стала перша система вимірювань на широких площах у 2000 році. Багато країн прийняли цю технологію, зокрема найбільшого поширення ця технологія набула на теренах мереж Китаю [4].

Таким чином вже на початку 2000-х років були створені та впроваджені перші розумні електричні мережі, проте сам термін Smart Grid (розумна мережа) ще не був популярним та загальноприйнятим.

Тож термін **Smart Grid** став популярним з 2003 року, коли він з'явився у статті "Попит на надійність керуватиме інвестиціями в автоматизацію" Майкла Бура. У цій роботі наведено декілька функціональних і технологічних визначень розумної мережі, а також перелічені деякі переваги. Одним із загальних елементів більшості визначень є використання цифрової обробки даних та зв'язку для електричної мережі, що робить потік даних і управління інформацією ключовими технологіями у розумних мережах. Різноманітні можливості широкої

інтеграції цифрових технологій, а також інтеграція нової мережі інформаційних потоків для контролю над процесами та системами, є ключовими технологіями у розробці розумних мереж.

На сьогоднішній день, електроенергетика перетворюється у трьох класах: покращення інфраструктури, додавання цифрового шару, який є сутністю розумної мережі, та перетворення бізнес-процесів, що робить розумні мережі прибутковими. Більша частина зусиль спрямована на модернізацію електричних мереж: особливо це стосується розподілу та автоматизації підстанцій, які тепер будуть включені в загальну концепцію розумних мереж, проте також розвиваються і інші додаткові можливості.

Таким чином створення Smart Grid мереж ефективно допомагає вирішити три наведені вище основні проблеми електричних мереж, з огляду на те що ефективність використання ресурсів значно зросла, завдяки наявності даних щодо реального споживання та можливості реагувати на них динамічно. Також така система може повідомляти споживачам про бажані проміжки часу для використання електроенергії. Крім того Smart Grid може динамічно встановлювати тариф , щоб стимулювати споживачів використовувати електроенергію у проміжки низького споживання [5]. Регулювання споживання Smart Grid мережею відповідно до періодів низького споживання на прикладі зарядки електромобіля наведено на рис. 1.2.

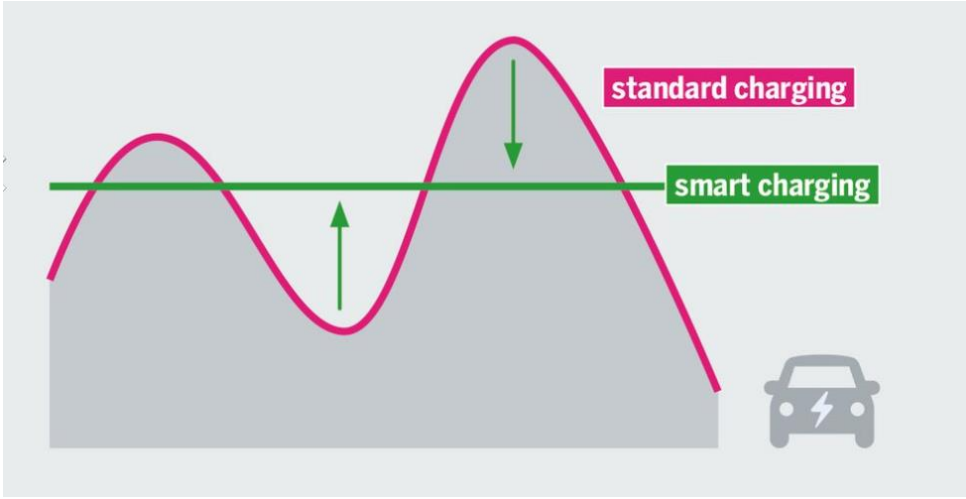

Рисунок 1.2 - Регулювання споживання Smart Grid мережею відповідно до періодів низького споживання на прикладі зарядки електромобіля

Для збільшення надійності мережі Smart Grid дозволяє автоматично регулювати напругу та частоту у мережі, щоб уникнути перепадів, які можуть призвести до відключення енергії. Наприклад в разі одномоментного великого збільшення навантаження на мережу smart grid може попередити всіх індивідуальних споживачів, щоб тимчасово зменшити навантаження для надання часу на запуск більшого генератора щоб задовольнити збільшений попит. Також Smart Grid може виявити споживачів які внесли дисбаланс в систему та автоматично вимкнути їх, або миттєво передати інформацію оператору, щоб той ефективно відреагував.

Загалом же Smart Grid мережі значно спростили управління мережею оскільки вони дозволяють збирати детальні дані про споживання енергії, що дозволяє операторам мережі керувати постачанням електроенергії більш ефективно.

Новий етап розвитку Smart grid розпочався з розповсюдженням електростанцій, що виробляють електроенергію за допомогою відновлюваних джерел енергії. До них відносяться такі види електростанцій:

- Вітрові;
- Сонячні;
- Геотермальні.

Особливістю вітрових та сонячних електростанцій є саме те , що вони можуть бути використані децентралізовано. Тобто такі електростанції можуть бути встановлені не лише великими компаніями для забезпечення електроенергією великої кількості споживачів, а й звичайним домогосподарством на території приватної ділянки [6, с.135-138]. Таким чином це повністю змінює систему подачі електроенергії, оскільки замість одного централізованого постачальника з’являється безліч дрібних, на що звичайні мережі не були пристосовані. Тож Smart Grid мережі стали єдиним інструментом , яким можна було б ефективно регулювати та синхронізувати виробництво, для того щоб відновлювальні джерела енергії були якнайкраще інтегровані існуючі мережі електропостачання [7, с. 223-226].

1.2 Аналіз архітектури системи

На сучасному етапі розвитку Smart Grid технологій існують різноманітні рішення для моніторингу рівня балансу споживання активних споживачів.

Загалом їх можна розділити на декілька типів.

Перш за все, можна виділити системи, які використовують розумні лічильники та сенсори для збору даних про споживання електроенергії в режимі реального часу. Після чого дані передаються до централізованої системи моніторингу, яка аналізує їх та виводить інформацію про стан балансу споживання.

Багато рішень базуються на використанні інтернету речей та забезпечують підключення різних пристроїв і обладнання до мережі. Інтернет речей загалом вніс великий вклад у розвиток smart grid мереж, оскільки ця технологія дозволяє здійснювати дистанційне вимірювання та керування, а також збирати дані для подальшого аналізу.

Крім того останнім часом в розробку систем моніторингу вносять технології штучного інтелекту та аналітики даних. Вони дозволяють автоматизувати процес обробки великих обсягів інформації, виявляти патерни та тренди, що сприяє швидкому реагуванню на зміни у споживанні електроенергії.

Система моніторингу рівня балансу споживання активних споживачів в Smart Grid мережах має свої особливості через специфічні потреби активних споживачів.

Активні споживачі електроенергії – це споживачі, які одночасно споживають та виробляють енергію [8]. Через що сама архітектура системи моніторингу значно відрізняється від звичайної. Оскільки в традиційній системі моніторингу є лише виробники та споживачі, а отже виробники мають лише моніторити виробництво, щоб воно задовольняло попит, а споживачі власне використання електроенергії. Тепер же ж з’явилася велика кількість дрібних виробників які є одночасно і споживачами , тому для збереження балансу та ефективності електропостачання у таких системах стали використовувати Smart Grid мережі [9].

Відмінності між звичайними електричними мережами та Smart Grid наведені на рис. 1.3.

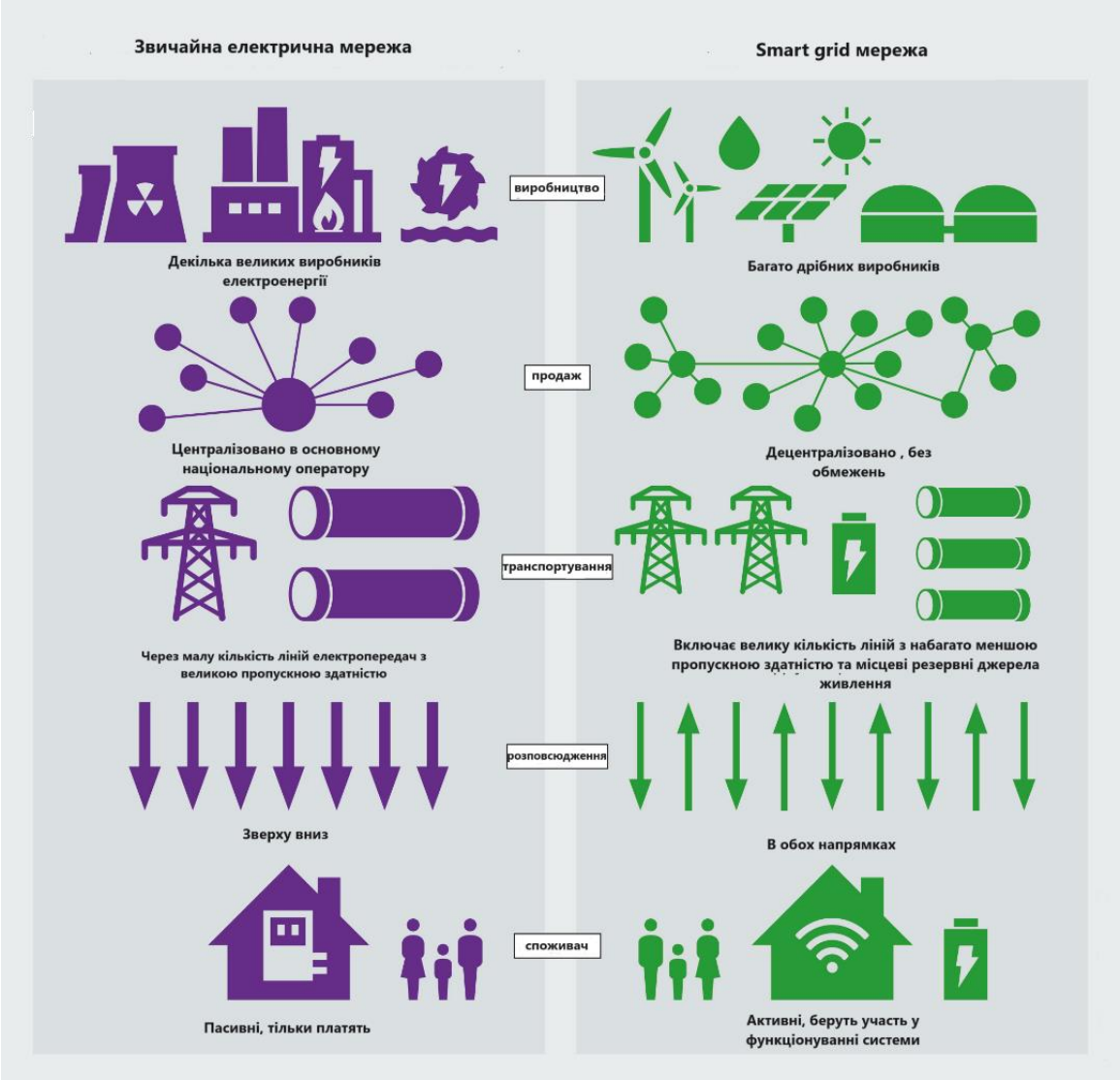

Рисунок 1.3 - Відмінності між звичайними електричними мережами та Smart Grid

Отже архітектура системи моніторингу рівня балансу споживання активних споживачів в Smart Grid мережах є складною та інтегрованою структурою, що включає в себе різноманітні компоненти та технології для забезпечення ефективного контролю та управління енергоспоживанням. Загалом система складається із чотирьох рівнів, які наведені на рис. 1.4 .

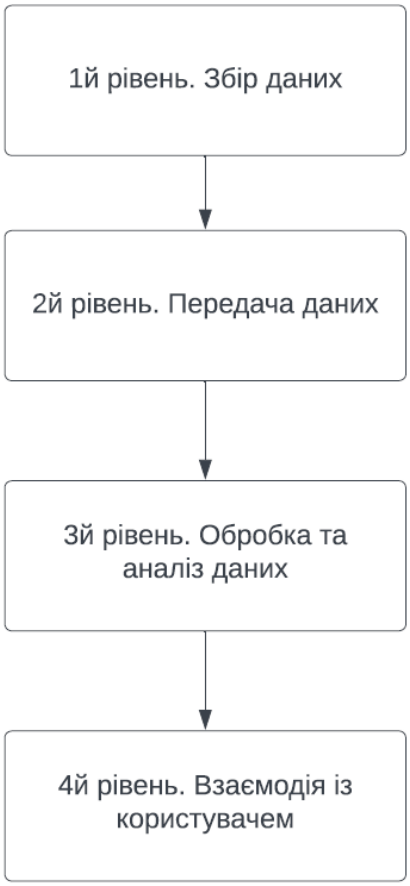

Рисунок 1.4 - Архітектурні рівні системи моніторингу рівня балансу споживання активних споживачів в Smart Grid мережах

На першому рівні архітектури розташовані сенсори та вимірювальні пристрої, які забезпечують збір реальних даних про енергоспоживання активних споживачів. Ці дані передаються на наступний рівень, де розташовані апаратне та програмне забезпечення для їх обробки та аналізу.

Другий рівень архітектури включає в себе системи збору та передачі даних, такі як мережі інтернету речей, що дозволяють забезпечити зв'язок між датчиками та централізованою системою моніторингу. Важливою частиною цього рівня є технології забезпечення безпеки, оскільки дані про енергоспоживання є критичними та конфіденційними.

Третій рівень включає централізовану систему моніторингу, яка отримує, обробляє та аналізує дані в реальному часі. Вона використовує аналітичні

алгоритми, враховуючи різні параметри, такі як попит, виробництво, та прогнози споживання електроенергії.

На четвертому рівні розташовані інструменти взаємодії з користувачем, такі як як застосунок із відповідним графічним інтерфейсом, що дозволяє операторам ефективно моніторити та управляти рівнем балансу споживання. Графічний інтерфейс надає інтуїтивний доступ до важливої інформації та дозволяє приймати швидкі та обґрунтовані рішення.

Архітектура системи моніторингу в Smart Grid мережах є розгалуженою та складною, оскільки вона повинна враховувати різні вимоги, стандарти та специфікації, а також забезпечувати швидку реакцію на зміни в енергетичній системі. Здійснюючи постійний моніторинг та оптимізацію цієї архітектури, можна забезпечити ефективну та надійну роботу Smart Grid мереж. Крім того можна виділити особливо роль інструментів взаємодії таких як застосунок та графічний інтерфейс, оскільки вони знаходяться на останньому четвертому архітектурному рівні і відповідають за безпосередню взаємодію з користувачем. Таким чином надзвичайно важливо створити веб-застосунок із якісним графічним інтерфейсом, оскільки саме від нього залежить безпосереднє враження користувача і без нього усі інші компоненти втрачатимуть сенс [10].

1.3 Огляд основних потреб активних споживачів

Активні споживачі мають досить специфічні потреби порівняно із звичайними споживачами, які не виробляють електроенергію. Це пояснюється тим, що вони є ще й виробниками електроенергії, а отже й інструменти у них мають бути відповідні: як для споживача так і для виробника.

Тож активні споживачі мають власні електростанції, а значить вони мають знати поточні дані по генерації, які збираються з відповідних датчиків. Проте це далеко не все що потрібно, оскільки генерація з використанням відновлювальних джерел енергії не є сталою. Вона залежить перш за все від погоди. У випадку сонячної електростанції генерація залежить від того наскільки ясний день. У випадку вітряної - наскільки вітряний.

Отже активному споживачу необхідно мати докладну статистику по продуктивності виробництва електроенергії в залежності від погоди, щоб можна було планувати використання електроенергії так , щоб не закуповувати електроенергію із загальної мережі.

Також цілком можливо , що активний споживач буде продавати надлишки енергії в загальну мережу. У багатьох країнах світу існують програми що стимулюють споживачів до встановлення домашніх електростанцій. В Україні така програма називається “Зелений тариф”. Ця програма якраз надає споживачам можливість продавати надлишки електроенергії, виробленої з використанням відновлюваних джерел енергії до державного оператора електромережі за вищими тарифами, які гарантують вигоду для виробників зеленої енергії. Хоч така програма і стимулює поширення виробництва електроенергії за допомогою відновлювальних джерел енергії , однак така модель взаємодії також не є ідеальною, з огляду на те що з’являється дисбаланс між генерацією та споживанням у мережі. Така ситуація складається через те, що надлишки зазвичай виробляються в непікові години, коли енергосистема не потребує додаткової генерації, в той час як в пікові години коли система максимально навантажена, сам виробник зеленої енергії споживає більше енергії.Тому надлишків стає набагато менше і система не отримує додаткову генерацію. Таким чином під час збоїв в роботі в пікові години система матиме значно менше засобів для резервної генерації [11]. А в таких екстрених ситуаціях як блекаут цей дисбаланс буде досягати критичних значень, які призведуть до збоїв в роботі обладнання енергетичної мережі [12].

Тому активному споживачу надзвичайно важливо мати докладну статистику по виробництву та споживанню електроенергії щоб використовувати електроенергію у непікові години, а продавати у час із найбільшим попитом.

Для стимулювання до такого відповідального балансу споживання уряди країн запроваджують спеціальні тарифи , наприклад в Україні діють денні та нічні тарифи, така опція стала можливою завдяки впровадженню Smart Grid мереж та інтеграції в них двохфазних лічильників. Такі лічильники автоматично

розділяють енергію, що була спожита в ночі та в день . Тарифи в день вищі за стандартний тариф через підвищений попит, а вночі нижчий через менше загальне споживання.

Крім того у США та деяких країнах західної Європи існують програми "динамічного ціноутворення" що застосовуються до енергії , що продають активні споживачі. Таким чином для збільшення власних доходів вони зацікавлені продавати якомога більше енергії в пікові години коли вартість більша [13].

Таким чином засоби моніторингу, що надають докладну статистику є необхідними для усіх активних споживачів, як для тих хто виробляє електроенергію для власного використання , так і для тих хто продає енергію в мережу . Також ця інформація є необхідною при виникненні екстренних ситуацій таких як блекаут.

1.4 Аналіз існуючих рішень для моніторингу рівня балансу споживання активних споживачів

На ринку існує ряд програмних продуктів які дозволяють моніторити стан Smart Grid мережі та обробляти відповідні дані.

Найбільш розповсюдженими серед них є EcoStruxure™ Electric Power Monitoring Expert, Zabbix та SolarWinds Network Performance Monitor.

EcoStruxure™ Electric Power Monitoring Expert це інтегрована система моніторингу енергоспоживанням, розроблена компанією Schneider Electric. Power Monitoring Expert включає в себе широкий спектр аналітичних інструментів, таких як графіки та діаграми.Ці інструменти спрямовані на ефективний контроль та оптимізацію енергетичних ресурсів у режимі реального часу. Система Power Monitoring Expert призначена для застосування в різних галузях, включаючи комерційні, промислові та громадські будівлі [14]. Офіційна веб-сторінка EcoStruxure™ Electric Power Monitoring Expert наведена на рис. 1.5 .

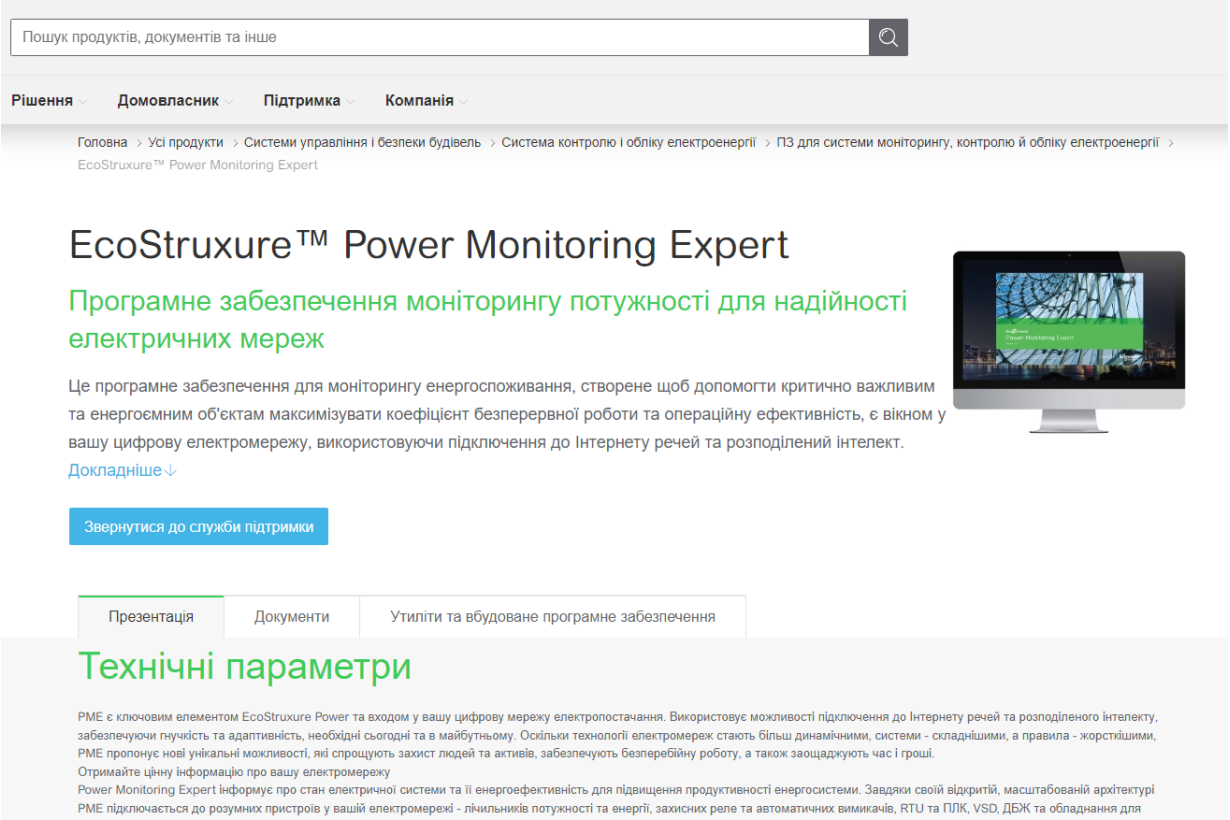

Рисунок 1.5 - Офіційна веб-сторінка EcoStruxure™ Electric Power Monitoring Expert

Перевагами цієї системи є:

- Профільність, ця система призначена саме для smart grid мереж.
- Широкий спектр аналітичних інструментів.
- Універсальність, цю систему може використовувати не лише дрібний виробник , а й промисловий.
- Безпека.

Недоліками цієї системи є:

- Складність для пересічних користувачів.
- Необхідність використання спеціального обладнання.
- Відсутність дистанційного доступу до інформації.
- Застарівший, недостатньо інтуїтивний дизайн інтерфейсу.
- Висока вартість

Zabbix - це безкоштовна та відкрита система моніторингу, яка використовується для відстеження та візуалізації стану та продуктивності серверів, мережевих пристроїв та датчиків. Вона дозволяє централізовано моніторити показники , що надають датчики Smart Grid мережі [15]. Офіційна веб-сторінка Zabbix наведена на рис. 1.6 .

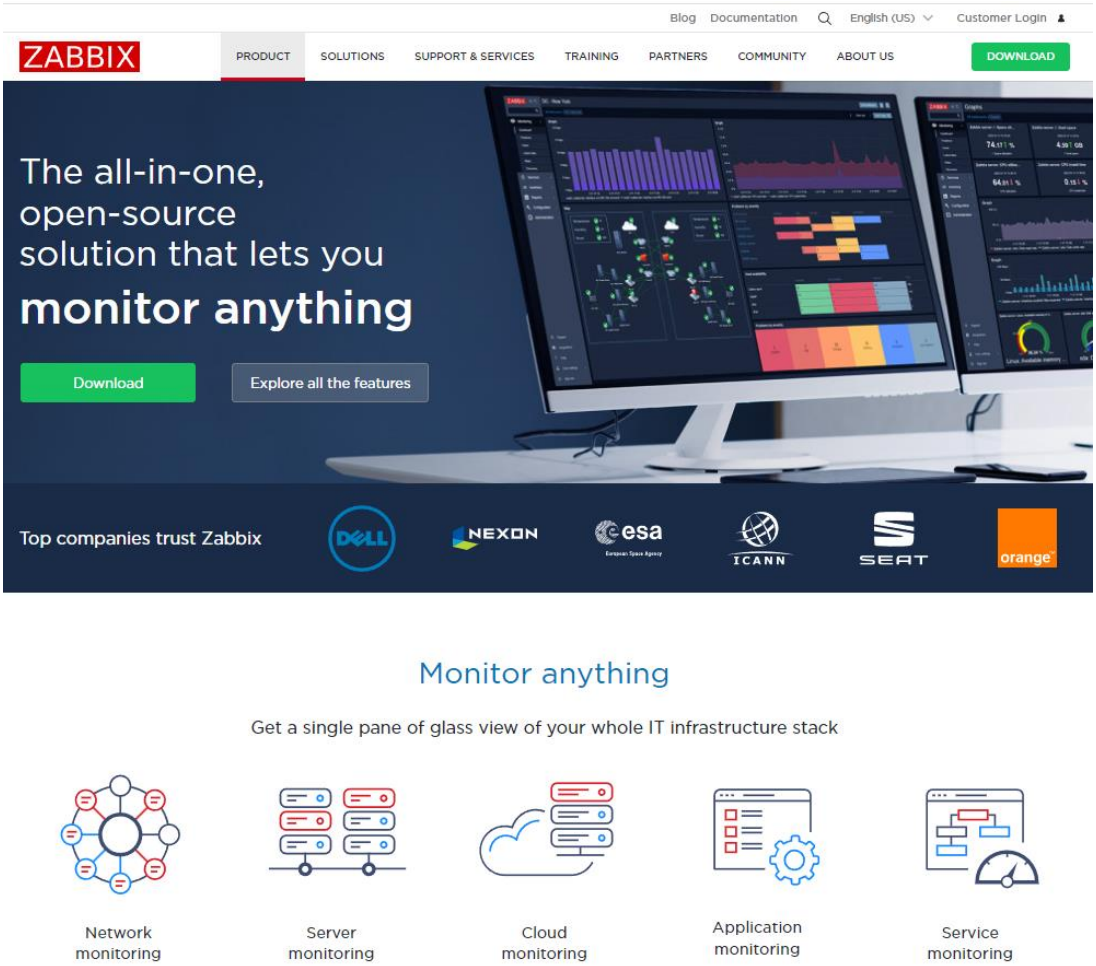

Рисунок 1.6 - Офіційна веб-сторінка Zabbix

Перевагами цієї системи є:

- Широкий спектр аналітичних інструментів.
- Можливість дистанційного доступу з комп’ютера під управлінням Windows.
- Відносна простота використання.
- Безкоштовність.

Недоліками цієї системи є:

- Складність налаштування, оскільки система не є профільною саме для моніторингу smart grid мереж, то необхідно її налаштовувати для чого знадобляться певні профільні навички.
- Відсутність доступу зі смартфонів.
- Застарівший, недостатньо інтуїтивний дизайн інтерфейсу.

SolarWinds Network Performance Monitor - це потужне програмне забезпечення для моніторингу мережі, яке надає комплексний набір функцій для відстеження та візуалізації стану та продуктивності мережевих пристроїв та датчиків.

Це програмне забезпечення пропонує широкий спектр аналітичних звітів та візуалізацій, які допоможуть активному споживачу зрозуміти стан та продуктивність мережі. Ці звіти можна налаштувати відповідно до потреб споживача, щоб можна було отримати інформацію, яка є найбільш необхідною [16]. Офіційна веб-сторінка SolarWinds Network Performance Monitor наведена на рис. 1.7 .

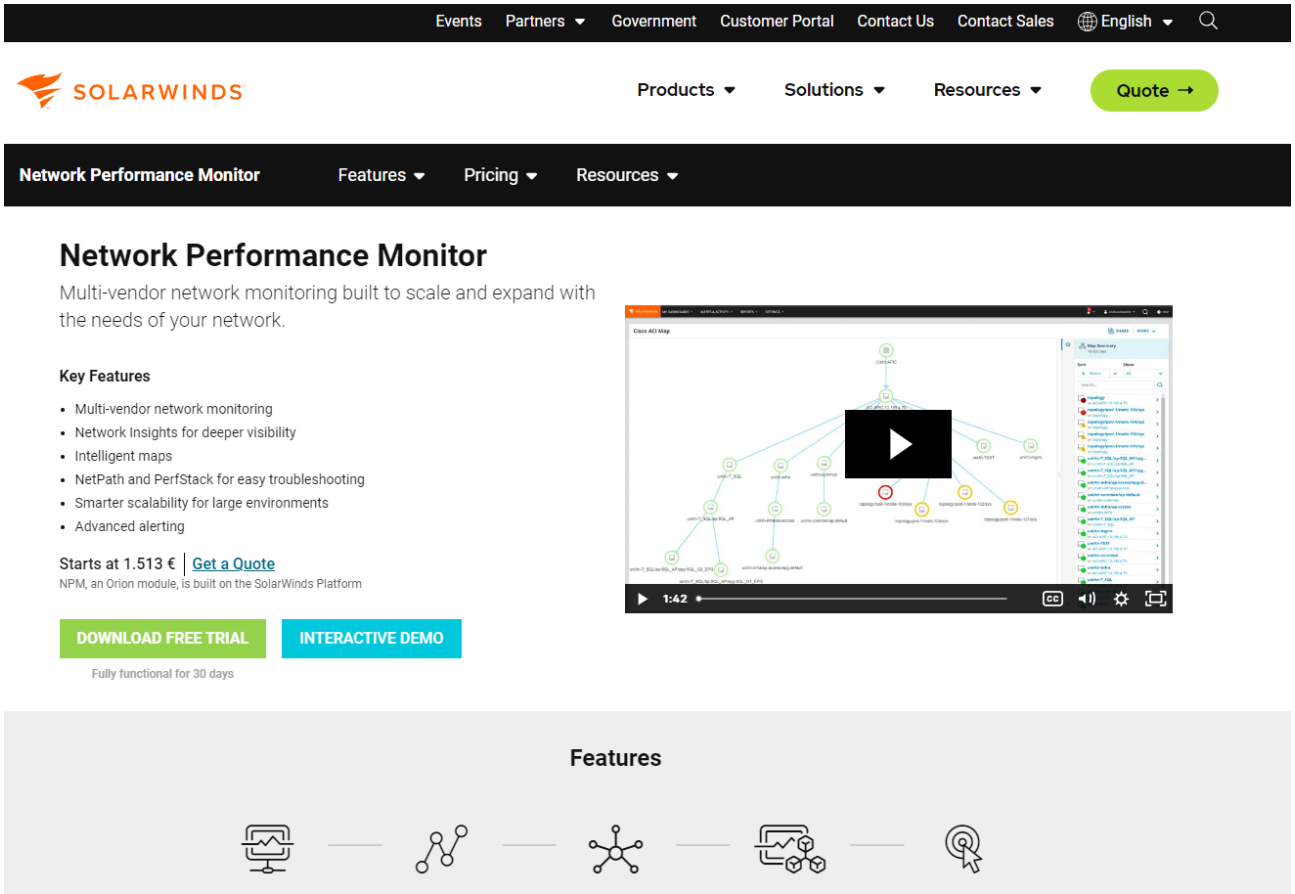

Рисунок 1.7 - Офіційна веб-сторінка SolarWinds Network Performance Monitor

Перевагами цієї системи є:

- Широкий спектр аналітичних інструментів.
- Можливість дистанційного доступу з комп’ютера під управлінням Windows.

Недоліками цієї системи є:

- Складність налаштування. Оскільки система не є профільною саме для моніторингу Smart Grid мереж, то необхідно її налаштовувати для чого знадобляться певні профільні навички.
- Відсутність доступу зі смартфонів.
- Висока вартість.
- Складність використання.
- Застарівший, недостатньо інтуїтивний дизайн інтерфейсу.
- Проблеми з безпекою, оскільки у компанії SolarWinds стався серйозний кібер-витік даних у 2020 році,

Загалом існуючі рішення дозволяють ефективно відстежувати виробництво та споживання енергії активними споживачами.

Проте, існуючі рішення також виявляють певні недоліки. Деякі з них можуть бути недостатньо гнучкими та неадаптованими для використання активними споживачами. Інші можуть бути не досить безпечними, оскільки обробка великої кількості конфіденційної інформації вимагає високого рівня захисту. Також більшість з них є занадто складними для пересічних користувачів, оскільки навіть якщо складність експлуатації невисока, то виникають складнощі із налаштуванням.

Крім того, їх усіх об’єднує ще один недолік, а саме них відсутність інтуїтивного та зручного інтерфейсу з сучасним дизайном, яким можна було б користуватися людям, що не мають профільних знань у сфері інформаційних систем. Також ці системи потребують спеціальних пристроїв для відстеження показників, або як мінімум спеціального програмного забезпечення, що ускладнює доступ для користувачів.

Усі ці аспекти підкреслюють необхідність подальших досліджень та розробки у цьому напрямку. Розуміння сильних та слабких сторін існуючих рішень дозволить створити новий застосунок для системи моніторингу, який буде відповідати потребам активних споживачів та забезпечить ефективний та загальнодоступний моніторинг та аналіз виробництва та споживання електроенергії.

1.5 Постановка задачі

Проведений огляд основних потреб активних споживачів та аналіз існуючих рішень дозволили виділити їхні сильні та слабкі сторони, що в свою чергу надало можливість створити вимоги до нового проекту.

Тож для того щоб веб-застосунок системи моніторингу рівня балансу споживання активних споживачів в Smart Grid мережах був якісним, типові проблеми подібних програмних систем були вирішені, а потреби активних споживачів були задоволені він повинен бути:

- Простим та зрозумілим для пересічного користувача.
- Готовим до використання без складних додаткових налаштувань.
- Доступним з будь-якого пристрою через мережу Інтернет.
- Приємним у використанні завдяки сучасному та інтуїтивному інтерфейсу.
- Інформативним та надавати вичерпні інструменти для відображення деталізованої статистики.
- Практичним та розраховувати додаткову інформацію таку як підрахунок заробітків та витрат активного споживача.

Таким чином найбільш доцільним вважаю створення саме веб-застосунку, оскільки такі застосунки не потребують спеціального , обладнання або програмного забезпечення до них на відміну від наявних програмних систем. Також жодна система не надає можливості переглядати дані через смартфон, що обмежує їхнє дистанційне використання. У свою чергу веб-застосунок не потребує жодного додаткового програмного забезпечення та дозволяє

отримувати доступ до інформаційної системи дистанційно з будь-якого пристрою.

1.6 Висновки до розділу

В результаті досліджень проведених у цьому розділі було:

- Досліджено особливості Smart Grid мереж.
- Проведено аналітичний огляд теоретичних основ побудови систем моніторингу рівня балансу споживання активних споживачів.
- Виокремлено основні типові проблеми подібних програмних систем та визначено вимоги до програмного продукту.

РОЗДІЛ 2
АНАЛІЗ ПРОГРАМНО-АРХІТЕКТУРНИХ РІШЕНЬ ЗАДЛЯ РЕАЛІЗАЦІЇ ВЕБ-ЗАСТОСУНКІВ

2.1 Огляд базової структури веб-застосунку

Веб-застосунки є дуже комплексними програмними системами, що включають в себе велику кількість різних частин які відповідають за окремі частини застосунку. Двома основними складовими є:

- Клієнтська частина.
- Серверна частина.

Клієнтська частина використовується для взаємодії з користувачем. В той час як серверна частина відповідає за підтримку бази даних та обробку запитів до неї та зв’язок із клієнтською частиною.

Приклад структури веб-застосунку наведений на рис. 2.1 .

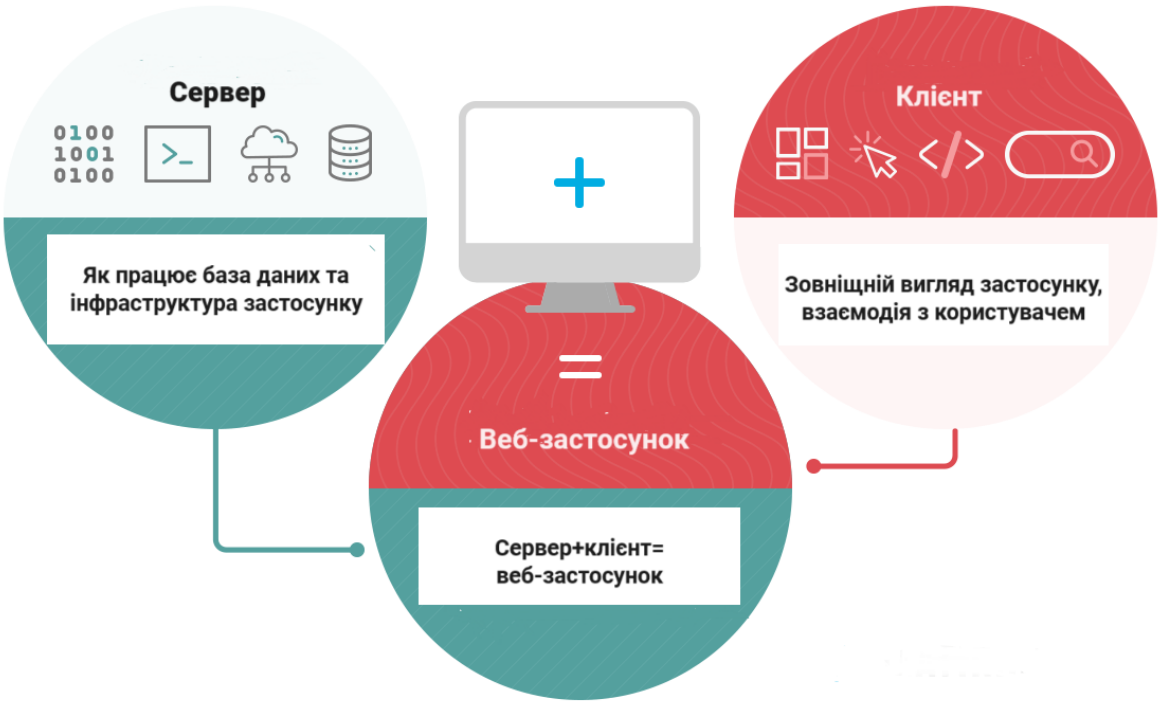

Рисунок 2.1 - Приклад структури веб-застосунку

2.2 Огляд інструментів для створення клієнтської частини

Клієнтська частина в базових веб-застосунках складається з трьох основних компонентів:

- Розмітки.
- Стилізації.
- Програмної логіки.

Розмітка реалізується за допомогою HTML [17, с.85-90].

HTML - це стандартна мова розмітки документів, призначених для відображення у веб-браузерах. Вона визначає зміст і структуру веб-контенту.

В ході запуску веб-застосунку веб-браузер отримує HTML-документ з веб-сервера або локального сховища і перетворює його на мультимедійну веб-сторінку. Таким чином основна роль HTML полягає саме в наданні семантичного опису структури веб-сторінки.

HTML-елементи є основою будь-якої HTML-сторінки, оскільки саме вони є її будівельними блоками. Завдяки HTML-елементам на сторінку можна додати такі елементи як поля введення, абзаци, зображення та навіть цілі форми [18].

HTML-елемент складається з трьох основних частин:

- Тегу.
- Атрибуту.
- Змісту.

Теги HTML є невід'ємною частиною HTML-елементу через те що саме вони визначають тип контенту (текст, зображення, відео), положення елементів на сторінці, формат тексту (заголовки, абзаци, списки тощо), мультимедійні елементи, такі як відео та аудіо, створюють посилання на інші сторінки та ресурси, включаючи форми користувача (реєстраційні форми, форми замовлень тощо), і використовуються для збору даних з форм користувача (наприклад, реєстраційних форм, форм замовлень тощо).

Атрибути HTML-елементу також є дуже важливими тому що саме через них налаштовуються властивості елементів, такі як колір, розмір і вирівнювання,

що є необхідним для створення привабливого і функціонального веб-контенту [19].

Стилізація реалізується за допомогою CSS.

CSS - це мова, яка визначає зовнішній вигляд і стиль веб-документів, написаних мовами розмітки, такими як HTML. Вона використовується для розділення та відображення вмісту, включаючи макет, кольори та шрифти. Таке розділення допомагає поліпшити доступність, підвищує гнучкість і контроль презентації, дозволяє використовувати спільний стиль для кількох сторінок і зменшує повторення структурованого контенту.

Крім того CSS також відокремлює розмітку від зовнішнього вигляду, дозволяючи відформатувати одну сторінку належним чином для різних пристроїв і способів перегляду, включаючи екран, друк, аудіо та тактильні пристрої на основі шрифту Брайля. CSS також має правила для адаптації до альтернативних форматів, коли доступ до контенту здійснюється з мобільних пристроїв. Назва "каскадний" означає порядок пріоритету, який визначає, який стиль застосувати, коли для елемента визначено кілька стилів. Такий каскадний підхід є передбачуваним і дозволяє ефективно керувати зовнішнім виглядом веб-сторінки [20].

Програмна логіка реалізується за допомогою JavaScript.

Отже **JavaScript** є основною мовою програмування, що використовується для стоврення веб-застосунків. Веб-браузери мають спеціальні платформи JavaScript, які виконують код на стороні клієнта. Ці платформи також використовуються деякими серверами та різними застосунками.

Загалом же JavaScript - це мова високого рівня, яка відповідає стандарту ECMAScript і зазвичай компілюється на льоту. Вона відрізняється динамічною типізацією, прототипним об'єктно-орієнтованим програмуванням і першокласними функціями. Вона підтримує мультипарадигмальний підхід, включаючи подієво-орієнтований, функціональний та імперативний стилі програмування. Ця мова програмування має інтерфейс прикладного

програмування (API) для роботи з текстом, датами, регулярними виразами, стандартними структурами даних і об'єктною моделлю документа (DOM).

Стандарт ECMAScript не включає функції вводу/виводу (I/O), такі як робота в мережі, зберігання даних і графічні функції. На практиці веб-браузери та інші системи виконання надають JavaScript API для вводу/виводу [21].

Таким чином використання таких інструментів як HTML, CSS та JavaScript забезпечує повноцінну розробку клієнтської частини веб-застосунку. Проте з плином часу веб-застосунку ставали все складнішими , а розробка з використанням стандартних інструментів стала занадто довгою та складною через що довелося шукати нові шляхи для створення сучасних веб-застосунків. Через що й були створенні фреймворки, які доповнюють базове програмування на JavaScript та пропонують якісно новий підхід до створення веб-застосунків.

Використання фреймворків у програмуванні включає кілька суттєвих переваг порівняно з базовим JavaScript.

По-перше, новий підхід значно прискорює процес розробки завдяки наявності готових структурного підходу та готових компонентів. Таким чином програмісти можуть витрачати набагато менше часу , завдяки зменшенню кількості однотипних, рутинних завдань.

По-друге, такий підхід значно покращує організацію коду. Фреймворки надають стандартні практики та конвенції, які сприяють легшому розумінню та спільній роботі між командами розробників. Таким чином це надає величезну перевагу особливо при створенні та обслуговуванні великих проектів.

Іншою важливою перевагою є забезпечення високого рівня абстракції. Фреймворки надають абстракції для багатьох завдань, що дозволяє розробникам уникнути деталей реалізації та швидше концентруватися на функціональності. Це робить код більш модульним та легким для розширення.

Крім того серед переваг фреймворків можна виділити активну спільноту та багату екосистему готових рішень, плагінів та інструментів, що полегшує інтеграцію та підтримку проектів.

Загалом, використання фреймворків дозволяє збільшити продуктивність розробників, поліпшити організацію коду, підвищити рівень абстракції та використовувати широку спільноту для обміну знаннями та ресурсами.

Тому враховуючи вищезазначене вважаю доцільним використати фреймворк для реалізації застосунку.

Фреймворків існує декілька. Три основних це:

- Angular.
- React.
- Vue.js.

React це фреймворк розроблений компанією Meta Platforms. Цей фреймворк використовує декларативний підхід у програмуванні. Це означає, що для створення бажаного інтерфейсу користувача треба описати його бажаний стан , а сам React відповідає за оновлення DOM для відображення цього стану, що спрощує розробку та робить код більш зрозумілим. React також використовує компонентний підхід, де структура веб-застосунку поділяється на дрібні, повторно використовувані компоненти, що робить код модульним та організованим. З використанням віртуального DOM React ефективно оновлює інтерфейс, мінімізуючи кількість змін у реальній об'єктній моделі документа [22].

Також цей фреймворк використовує інтеграцію JSX, що дозволяє вставляти HTML-код у JavaScript, робить код більш зрозумілим та зручним для розробників. Ще однією перевагою є однонаправлений потік даних, що спрощує відстеження стану.

Крім того важливою перевагою React є те що він надає широкі можливості для створення графіків. Основною бібліотекою для створення графіків у веб-застосунках є Chart.js. Проте на відміну від інших фреймворків для React було створено адаптовану версію з розширеними можливостями 'react-chartjs-2'. Вона надає можливості для відображення різноманітних типів графіків, таких як лінійні, кругові, стовпчасті тощо, і дозволяє легко налаштовувати їх вигляд, масштабування та взаємодію з користувачем [23].

Переваги React:

- Простота.
- Гнучкість.
- Продуктивність.
- Велика та активна спільнота.
- Широкі можливості для відображення графіків [24; 25].

Angular - це фреймворк, розроблений Google для створення веб-застосунків. Основною особливістю Angular є його структурований підхід до створення великомасштабних веб-застосунків з акцентом на модульність, залежності та чітку архітектуру. Для цього він використовує модель Model-View-Controller (MVC) [26].

MVC – це архітектурний підхід до програмування, який розділяє програму на три основні компоненти: модель, представлення та контролер. Кожен з цих компонентів відповідає за власну функціональність і забезпечує високий ступінь модульності, розширюваності та простоти розробки програмного забезпечення. Таким чином застосунок розділяється на три основні компоненти: модель, що представляє дані, представлення (розмітка, стилізація), яке відповідає за візуальне представлення даних, логіку застосунку та контролер, які відповідають за взаємодію між моделлю та логікою застосунку, моделлю та представленням. Це спрощує обслуговування та покращує розподіл завдань у коді [27].

Структура MVC наведена на рис. 2.2 .

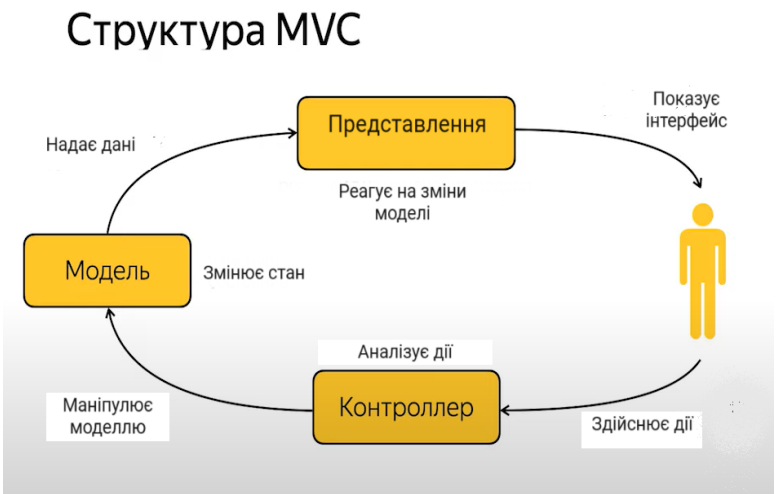

Рисунок 2.2 – Структура MVC

Angular також використовує систему та ін'єкцію залежностей, що спрощує тестування та підвищує модульність, чітко визначаючи сервіси та інші компоненти, які повинні надавати компоненти. TypeScript, що використовується в Angular, додає статичну типізацію, зменшує час компіляції та полегшує написання коду.

Крім того Angular має багатий набір директив, які можуть додавати різні функції до елементів HTML, такі як умовне відображення та ітерації. Він також пропонує різноманітні сервіси - незалежні класи, які містять бізнес-логіку та надають функціональність компонентам.

Переваги Angular

- Структурований підхід.
- Типізація.
- Багата екосистема.
- Велика та активна спільнота.

Vue.js - це наймолодший фреймворк JavaScript. При його створенні розробники прагнули забезпечити баланс між простотою використання та гнучкістю функціональності.

Vue.js ґрунтується на компонентному підході, схожому до React та Angular, де компоненти виконують роль блоків коду з HTML, CSS та JavaScript для керування логікою та станом, що повторно використовуються. Серед особливостей Vue можна виділити шаблонні розширення, що нагадують HTML, що разом з директивами надає ряд корисних функцій, таких як v-for для ітерацій, v-if для умовного виведення або v-model для двостороннього з'єднання даних. Іншою особливістю Vue.js є його легкість і гнучкість, а також можливість застосування як для малих веб-компонентів так і для масштабування до складних веб-застосунків [28].

Переваги Vue.js.

- Простота навчання.
- Гнучкість.
- Ефективність.

Порівняння фреймворків наведено в таблиці 2.1.

Таблиця 2.1 - Порівняння фреймворків.

Функціональність	React	Angular	Vue.js
Підхід до розробки	Декларативний	MVC (Модель-Вид-Контролер)	Компонентний
Складність навчання	Середня	Складна	Низька
Структура	Гнучка	Жорстка	Гнучка
Типізація	Немає	TypeScript за замовчуванням	Немає
Маршрутизація	React Router	Angular Router	Vue Router
Продуктивність	Висока	Висока	Висока
Екосистема	Велика та активна	Велика та активна	Зростаюча
Підходить для	SPA, мобільні застосунки,	Масштабні SPA, корпоративні веб-застосунки	SPA, веб-компоненти,
Широкі можливості для відображення графіків	Так	Ні	Ні
Переваги	Простота, гнучкість, віртуальний DOM, Широкі можливості для відображення графіків	Структура, масштабованість, TypeScript, швидка компіляція	Простота, гнучкість, двостороння прив'язка даних

Продовження таблиці 2.1.

Функціональність	React	Angular	Vue.js
Недоліки	Недостатньо швидка компіляція	Складність	Менш розвинена екосистема, Недостатньо швидка компіляція

Усі фреймворки свої переваги та недоліки, проте мій вибір припав саме на React. Він був обраний через те, що він містить дуже функціональну та просту у використанні бібліотеку графіків 'react-chartjs-2'. Це є необхідним для розробки застосунку з великою кількістю статистичних даних, оскільки графіки дозволяють найбільш наочно та зрозуміло продемонструвати статистику.

2.3 Огляд інструментів для створення серверної частини

Серверна частина в базових веб-застосунках складається з двох основних компонентів:

- Бази даних.
- Інструменту для зв’язування серверної та клієнтської частин.

База даних є невід’ємною частиною серверної частини веб-застосунку, оскільки зберігає всі необхідні дані. Без бази даних усі дані веб-застосунку втрачаються після завершення сесії [29, с. 23].

Бази даних управляються системами управління базами даних.

Система управління базами даних - це комплекс програмних засобів, що дають змогу створювати бази даних і керувати даними. Іншими словами, СУБД це набір програм, що дає змогу організовувати, контролювати й адмініструвати бази даних [30, с. 112-113].

Основні функції СУБД:

- Управління даними в зовнішній пам'яті (на дисках);
- Управління даними в оперативній пам'яті з використанням дискового кешу;
- Журналізація змін (збереження історії), резервне копіювання та відновлення бази даних після збоїв;
- Підтримка мов баз даних (мова визначення даних, мова маніпулювання даними).

Схема роботи системи управління базами даних наведена на рис. 2.3 .

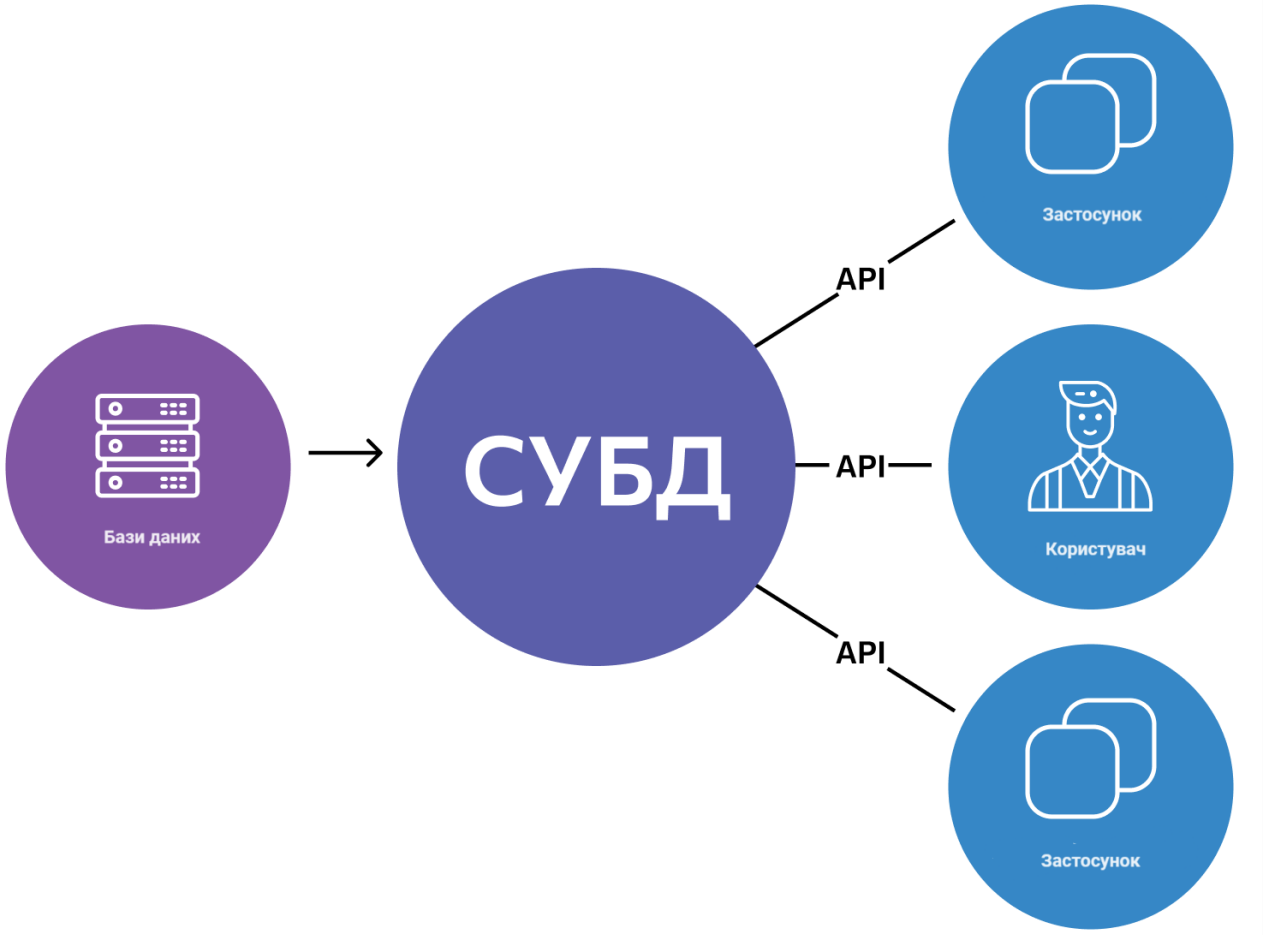

Рисунок 2.3 – Схема роботи системи управління базами даних

Кожна СУБД ґрунтується на якійсь моделі даних, це є однією з ознак класифікації. За моделлю даних СУБД бувають:

- Нереляційні.
- Реляційні.

Ці типи СУБД представляють два різних підходи до зберігання інформації.

Нереляційні СУБД не використовують реляційну модель даних і можуть зберігати дані в різних форматах, включаючи JSON, графи та документи. Через що вони відомі своєю гнучкістю, масштабованістю та швидкістю, особливо для певних типів запитів. Проте через відсутність єдиної структури вони можуть бути менш надійними, можуть не мати підтримки транзакцій, а їх мови запитів можуть бути складнішими для вивчення та використання, ніж SQL [31].

Типи нереляційних СУБД.

- Колоночні.
- Графові.
- Ключ-значення.
- Документо-орієнтовані.

Колоночна СУБД це система управління базами даних, яка використовується у великих системах. Така СУБД працює відповідно до свого унікального підходу до організації баз даних, де дані зберігаються відокремлено по колонках чи стовпцях, що значно відрізняється від реляційних СУБД, де інформація зберігається у вигляді рядків цілих записів. Завдяки такій структурі значно підвищується оптимізація операцій з даними. Так відбувається через те що колоночні СУБД можуть ефективно виконувати запити, які вимагають роботи тільки з певними колонками.

Колоночна СУБД може стати в нагоді коли користувач працює з великим обсягом даних в якому часто потрібно виконувати аналітичні операції. Такими операціями можуть бути підрахунок агрегатів або фільтрація за певними атрибутами.

До того ж такий тип СУБД доцільно використовувати коли необхідно швидко обробляти великі потоки даних. Така потреба часто виникає у фінансових установах для аналізу транзакцій або в телекомунікаційних компаніях для обробки великої кількості клієнтських даних.

Також важливою перевагою колоночних баз даних є їх масштабованість. Вони можуть легко масштабуватися горизонтально, додаванням нових серверів для розподілення навантаження, що робить їх ефективними для великих і зростаючих систем .

Незважаючи на значну кількість переваг колоночні СУБД не є універсальними та мають свої недоліки. Основним серед них є менша ефективність для операцій, які потребують роботи з цілими рядками даних або вимагають частого оновлення багатьох полів [32].

Прикладами колоночних СУБД є :

- Apache HBase [33].
- Apache Cassandra [34].
- Google Bigtable [35].

Графова СУБД це система управління базами даних, у якій дані представлені у вигляді графу, де вузли представляють сутності, а ребра відображають взаємозв'язки між цими сутностями. Завдяки такому унікальному підходу користувачі отримують можливість ефективно моделювати та виконувати операції зі складними мережами залежностей та взаємозв'язків.

Такі особливості роблять графові СУБД незамінними для використання в соціальних мережах для зберігання та аналізу зв'язків між користувачами, друзями та іншими сутностями. Також вони корисні в транспортних системах для моделювання маршрутів та зв'язків між різними точками, у наукових дослідженнях для аналізу мережі взаємозв'язків між об'єктами та в багатьох інших областях.

Головною перевагою графових СУБД є їх здатність ефективно виконувати операції, які вимагають знаходження шляхів, аналізу графа, визначення кратних зв'язків та взаємодій між різними сутностями, через що вони забезпечують швидку відповідь на складні запити та аналіз великих графів даних.

Іншою сферою застосування такого типу СУБД є розподілені системи. Графові СУБД є гарним вибором для таких систем через що така система

забезпечує ефективну обробку зв'язків між об'єктами, які розташовані на різних серверах чи вузлах.

Однак незважаючи на вищенаведені переваги такі системи не є універсальними та мають значні недоліки. Основним серед яких є низька ефективність для операцій, які вимагають швидкої і частої зміни даних або коли потрібно працювати з великим обсягом неструктурованих даних [36].

Прикладами графових СУБД є :

- Neo4j [37].
- OrientDB [38].
- ArangoDB [39].

СУБД типу ключ-значення це тип нереляційних СУБД, в яких дані зберігаються у вигляді пар ключ-значення. Завдяки такому унікальному підходу забезпечується швидке знаходження значення за ключем без необхідності складних операцій пошуку великих обсягів даних.

Такі особливості роблять СУБД типу ключ-значення незамінними для кешування, сесійного зберігання, управління конфігурацією та реалізації простих структур даних.

Тож головною перевагою СУБД типу ключ-значення є її швидкодія. Ця властивість особливо проявляється при роботі з великою кількістю простих операцій зберігання та отримання даних.

Іншою сферою застосування такого типу СУБД є розподілені системи, де важлива швидкість доступу до даних та масштабованість.

Однак незважаючи на вищенаведені переваги такі системи не є універсальними та мають значні недоліки. Основним серед яких є низька ефективність для операцій, які потребують складного аналізу даних, фільтрації або пошуку за різними критеріями. До того ж СУБД типу ключ-значення недоцільно використовувати у складних структурах даних, де потрібно зберігати багато взаємозв'язаних даних [40].

Прикладами СУБД типу ключ-значення є :

- Redis [41].

- DynamoDB [42].
- Riak [43].

Документо-орієнтована СУБД це тип нереляційних СУБД, де дані зберігаються у вигляді документів, зазвичай у форматі JSON. Відповідно до цього підходу кожен документ містить ключі-значення, які представляють дані та їх властивості, завдяки чому такий тип СУБД дозволяє зберігати структуровані та неструктуровані дані в одному документі, що робить їх гнучкими та динамічними.

Такі особливості роблять документо-орієнтовані СУБД незамінними для управління контентом, систем керування даними та інших сценаріїв, де важливо мати швидкий доступ до структурованих та гнучких даних. Наприклад, їх можна використовувати для зберігання користувацьких профілів, документів, блогів, конфігурацій та багатьох інших типів даних.

Головною перевагою документо-орієнтованих СУБД є їх гнучкість у роботі зі структурованими та неструктурованими даними, завдяки чому користувач отримує можливість додавати, видаляти та змінювати поля в документах без необхідності зміни структури бази даних.

Іншою значною перевагою документо-орієнтованих СУБД є підтримка різних типів запитів, включаючи пошук за ключами, фільтрацію, сортування та агрегацію даних. Такий функціонал дозволяє швидко та ефективно отримувати необхідну інформацію з бази даних.

Однак незважаючи на вищенаведені переваги такі системи не є універсальними та мають значні недоліки. Основним серед яких є низька ефектиність для операцій, які вимагають складного аналізу даних або великої кількості зв'язків між документами. До того ж документо-орієнтовані СУБД недоцільно використовувати для сценаріїв, де потрібно виконувати складні операції з великим обсягом даних [44].

Прикладами документо-орієнтованих СУБД є :

- MongoDB [45].
- CouchDB [46].

- Elasticsearch [47].

Порівняння типів нереляційних СУБД наведено в таблиці 2.2.

Таблиця 2.2 - Порівняння основних нереляційних СУБД.

Тип СУБД	Опис	Переваги	Недоліки	Приклади
Колоночна	Дані у вигляді стовпців, оптимізована для агрегування та аналітики.	Швидке агрегування, масштабованість, економія пам'яті.	Менш ефективна для оновлення рядків, не підходить для складних запитів до рядків.	Apache HBase, Apache Cassandra, Google Bigtable.
Графова	Дані у вигляді графу, де вузли - це сутності, а ребра - зв'язки між ними.	Ефективна для аналізу зв'язків, пошуку шляхів, моделювання складних мереж.	Менш ефективна для операцій над окремими рядками, не підходить для великих обсягів неструктурованих даних.	Neo4j, OrientDB, ArangoDB.
Ключ-значення	Дані у вигляді пар ключ-значення, швидкий доступ за ключем.	Дуже швидка для простих операцій, масштабована, розподілена.	Не підходить для складних запитів, аналізу зв'язків, неструктурованих даних.	Redis, DynamoDB (Amazon), Riak.
Документо-орієнтована	Дані у вигляді документів (JSON), гнучка структура.	Гнучкість, динамічність, підтримка структурованих та неструктурованих даних.	Менш ефективна для аналізу зв'язків між документами, складних операцій з великими обсягами даних.	MongoDB, CouchDB, Elasticsearch.

Приклад структури різних типів нереляційних СУБД наведено на рис. 2.5 .

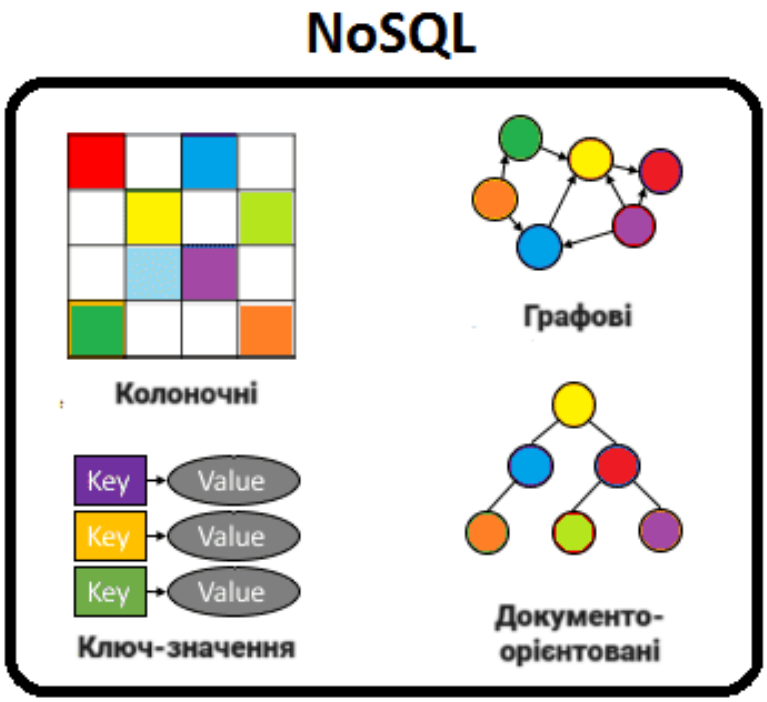

Рисунок 2.5 – Приклад структури різних типів нереляційних СУБД

На основі вищезгаданих типів нереляційних СУБД, можна зробити висновок, що жоден тип нереляційних систем управління базами даних не є універсальним . Тому найкращий тип буде залежати від конкретних потреб та характеристик проекту. Колоночні СУБД підходять для великих обсягів даних та аналізу даних з певною структурою. Графові СУБД використовуються для роботи з великими графами даних та складних взаємозв'язків. Ключ-значення СУБД підходять для швидкого доступу до даних з великою кількістю одночасних записів та читань. Документо-орієнтовані СУБД добре підходять для розробки застосунків зі змінюваною структурою даних та гнучкими схемами.

Отже, вибір найкращого типу нереляційної СУБД залежить від конкретних потреб проекту, його масштабу, типу даних та вимог до швидкості та ефективності операцій з даними.

Таким чином враховуючі недоліки нереляційних СУБД та універсальність реляційних вважаю доцільним обрати для розробки веб-застосунку реляційну СУБД.

Реляційні СУБД, використовують таблиці з рядками та стовпцями для зберігання даних, а також мову SQL для доступу до цих даних. Вони відомі своєю структурованістю, надійністю, підтримкою транзакцій та потужними можливостями SQL. Однак вони можуть бути складними в налаштуванні та обслуговуванні, не такими гнучкими для неструктурованих даних, а також можуть мати проблеми з масштабуванням при обробці великих обсягів даних [48].

Приклад структури реляційної СУБД наведено на рис. 2.4 .

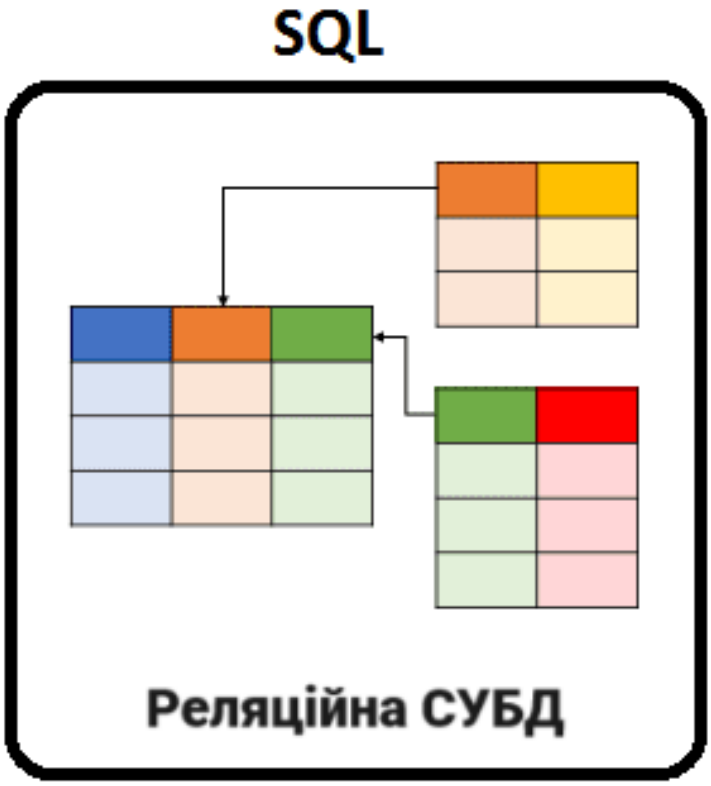

Рисунок 2.4 – Приклад структури реляційної СУБД

Основними реляційними СУБД є:

- MySQL.
- PostgreSQL.
- Microsoft SQL Server.

MySQL це реляційна система управління базами даних розроблена компанією Oracle. Її важливою особливістю є те, що цю СУБД можна використовувати, як при розробці малих веб-сайтів так і великих корпоративних

застосунків. Крім того ця СУБД відома своєю надійністю, швидкістю та гнучкістю, що робить її популярним вибором серед розробників та архітекторів баз даних усього світу.

Як і в інших реляційних СУБД в MySQL використовується мова запитів SQL, через що користувачу надаються широкі можливості. Серед них можна виділити легке створення, зчитування, оновлення та видалення даних з бази.

Серед головних особливостей MySQL особливе місце займає підтримка транзакцій. Саме вона забезпечує цілісність даних та узгодженість в роботі з базою даних, що особливо важливо в великих проектах.

Іншою особливістю MySQL є підтримка індексації даних. Саме ця технологія дозволяє користувачу швидко знаходити та вибирати дані за допомогою індексів. Це в свою чергу дуже підвищує продуктивність та швидкість роботи з базою даних.

Також дуже важливою рисою, особливо для невеличких проектів є те що MySQL має відкритий код та її використання є безкоштовним..

Крім того можна відмітити відносну легкість опанування MySQL через її загальну простоту та відсутність використання розширених функцій [49].

PostgreSQL це інша реляційна система управління базами даних. Серед її основним переваг виділяють її надійність, гнучкість та високу продуктивність. PostgreSQL має чудову швидкість обробки даних та високий рівень безпеки через що широко використовується у великих корпоративних проектах.

Головною особливістю вважається її розширена підтримка SQL. Тож на відміну від інших реляційних СУБД PostgreSQL підтримує складні запити, підзапити, відкладені транзакції та інші розширені функції. Через що ця СУБД є відмінним вибором для проектів з великою кількістю даних та складними вимогами до операцій з ними.

Іншими унікальними функціями PostgreSQL є підтримка геоданих та розширені можливостями роботи з JSON-даними. Тож підтримка геоданих надає користувачу можливість зберігати та опрацьовувати географічні дані. В той час

як розширені можливості роботи з JSON-даними надають гнучкість для роботи зі структурованими та неструктурованими даними в форматі JSON.

Також важливою характерною рисою PostgreSQL є його висока розширюваність та підтримка реплікації. Завдяки цьому користувач може створювати розподілені системи зберігання даних . Крім того це підвищує доступність та надійність системи.

Відносно до інших реляційних СУБД, PostgreSQL має велику активну спільноту користувачів та розробників. В свою чергу це надає велику кількість переваг, таких як підтримка нових користувачів, або розробка розширень [50].

Microsoft SQL Server це ще одна реляційна система управління базами даних , розроблена корпорацією Microsoft. Серед її основним переваг виділяють високу надійність, продуктивність та різноманітні функції.

Серед видатних особливостей Microsoft SQL Server в першу чергу можна виділити розширені можливості управління даними, включаючи підтримку транзакцій, оптимізацію запитів, індексацію та високий рівень безпеки даних. Це робить його відмінним вибором для великих підприємств та організацій з великим обсягом даних та складними вимогами до управління ними.

Головною ж особливістю Microsoft SQL Server є його інтеграція з екосистемою Microsoft, завдяки чому розробники та адміністратори отримують можливість легко інтегрувати бази даних SQL Server у свої застосунки та середовища.

Крім того Microsoft SQL Server має різноманітні можливості управління даними, включаючи аналітичні функції, засоби реплікації, агенти планування завдань та багато іншого. Такий функціонал дозволяє користувачам ефективно використовувати дані для аналізу, звітності та прийняття управлінських рішень.

Іншою значною перевагою Microsoft SQL Server є також широка підтримка користувачів та доступність різноманітних ресурсів. Через що користувач не матиме проблем з пошуком документації чи навчальних матеріалів. Або в будь-якому разі зможе звернутися на форуми підтримки. Таким чином значно

пришвидшується розв'язання проблеми та підтримується оптимальна роботу з базою даних.

Серед недоліків можна виділити те що на відміну від MySQL та PostgreSQL Microsoft SQL Server немає відкритого коду та є платним [51].

Порівняння основних реляційних СУБД наведено в таблиці 2.3.

Таблиця 2.3 - Порівняння основних реляційних СУБД.

Функція	MySQL	PostgreSQL	Microsoft SQL Server
Ліцензія	Відкритий код	Відкритий код	Комерційна
Платформа	Windows, Linux, macOS	Windows, Linux, macOS	Windows, Linux
Простота використання	Проста	Середня	Складна
Масштабованість	Висока	Висока	Дуже висока
Безпека	Добра	Дуже висока	Висока
Функціональність SQL	Базова	Розширена	Дуже розширена
Підтримка JSON	Обмежена	Розширена	Добра
Підтримка геоданих	Обмежена	Розширена	Добра
Інтеграція з іншими продуктами	Широка	Обмежена	Дуже широка (Microsoft)
Вартість	Безкоштовна	Безкоштовна	Платна
Підтримка спільноти	Велика	Велика	Дуже велика

Тож, MySQL, PostgreSQL та Microsoft SQL Server мають власні переваги та особливості. Однак мій вибір пав саме на PostgreSQL через те, що використання цієї СУБД є безкоштовним як і MySQL , проте вона має ширший функціонал.

Інструмент для зв’язування серверної та клієнтської частин виконує такі основні функції:

- Маршрутизація API.
- Обробка запитів.
- Міжсерверна взаємодія.
- Аутентифікація та авторизація.

Основним фреймворком , що для цього застосовується при розробці веб-застосунків є Express.js.

Express.js це веб-фреймворк для Node.js, який дозволяє створювати веб-застосунки та API швидко та ефективно. Головною ж його перевагою є широкий набір інструментів та можливостей для побудови різноманітних веб-застосунків. До того ж цей веб-фреймворк є одним з найкращих завдяки легкості використання та гнучкості.

Express використовує три основні концепції:

- Маршрутизацію.
- Обробник запитів.
- Шаблонізацію.

Маршрутизація є невід'ємною частиною Express , що дозволяє легко визначати шляхи для обробки різних типів запитів та виконувати необхідні дії відповідно до цих запитів.

Особливу роль в роботі Express займає обробник запитів, тобто функції, які обробляють запит перед тим, як він потрапить у фактичний обробник запиту. Завдяки такій особливості користувачі отримують можливість реалізувати різноманітні функції. Найпоширенішими такими функціями є логування, аутентифікація, перевірка прав доступу тощо, перед тим як обробляти основний запит.

Також дуже важливою для функціонування express є шаблонізація. Саме вона дозволяє розробникам створювати статичні та динамічні HTML-сторінки [52].

Загалом, Express.js є невід'ємною частиною веб-застосунку оскільки є легким та гнучким фреймворком що забезпечує стабільний зв'язок між серверною та клієнтською частиною.

Візуалізація ролі Express.js у довільному веб-застосунку наведена на рис. 2.6.

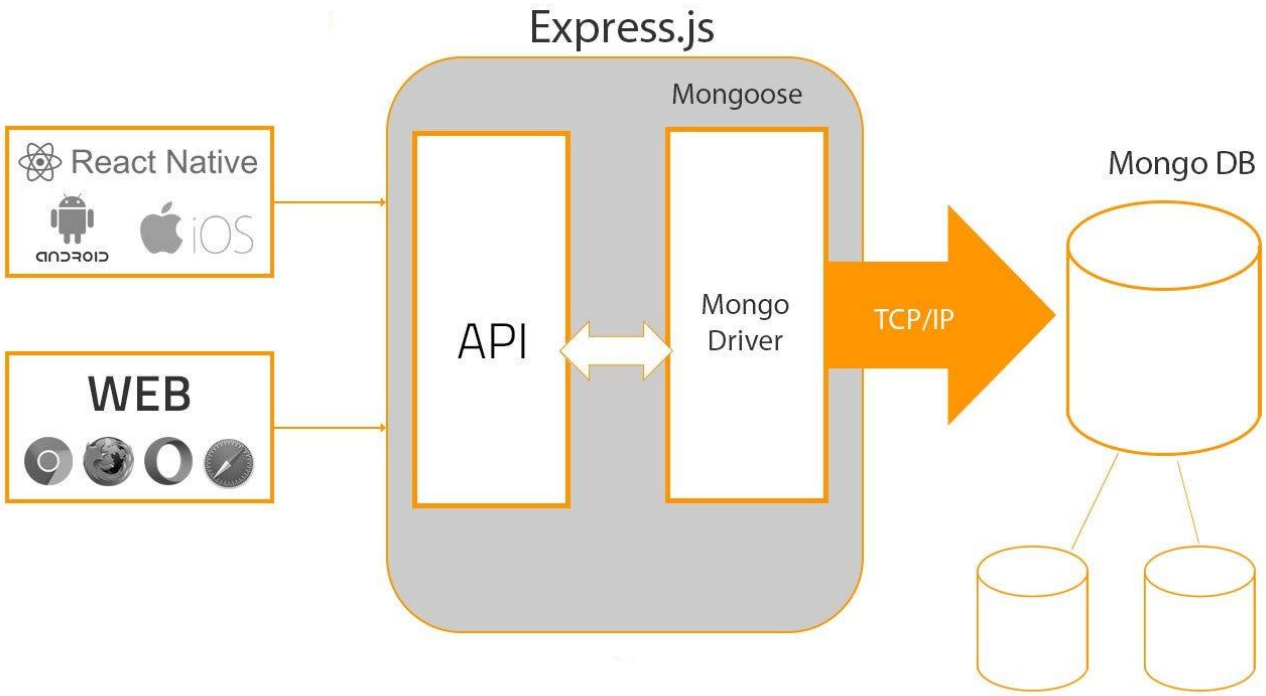

Рисунок 2.6 – Візуалізація ролі express.js у довільному веб-застосунку

2.4 Висновки до розділу

В результаті досліджень проведених у цьому розділі було проаналізовано програмно-архітектурні рішення задля розробки веб-застосунків та обрано найбільш підходящі для веб-застосунку системи моніторингу рівня балансу споживання активних споживачів в Smart Grid мережах.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБ-ЗАСТОСУНКУ СИСТЕМИ МОНІТОРИНГУ РІВНЯ БАЛАНСУ СПОЖИВАННЯ АКТИВНИХ СПОЖИВАЧІВ В SMART GRID МЕРЕЖАХ

3.1 Проектування веб-застосунку

Для виконання поставлених вимог ,на мою думку, необхідним є створити такі структурні елементи:

- Головну сторінку.
- Вікно авторизації.
- Вікно реєстрації.
- Відображення поточної генерації споживання та продажу електроенергії.
- Графіки, що відображають докладну статистику.
- Інструмент для прогнозування основних показників на декілька наступних днів.
- Базу даних, де будуть зберігатися дані користувачів.
- Проміжне програмне забезпечення для зв’язку клієнтської та серверної частини застосунку.
- Видалення користувача для захисту персональних даних.

3.2 Структура серверної частини веб-застосунку

Веб-застосунок складається з двох основних частин: клієнтської та серверної.

Серверна частина використовує СУБД PostgreSQL для управління базою даних.

Сама ж база даних використовується для збереження даних користувачів та відповідно їхньої статистики по споживанню та виробництву електроенергії.

База даних складається із таблиці users, яка містить колонки id, name, email, password, network_id. А також таблиць data, data_mon та data_year які зберігають статистичну інформацію про основні показники активних споживачів.

Структура бази даних наведена на рис. 3.1.

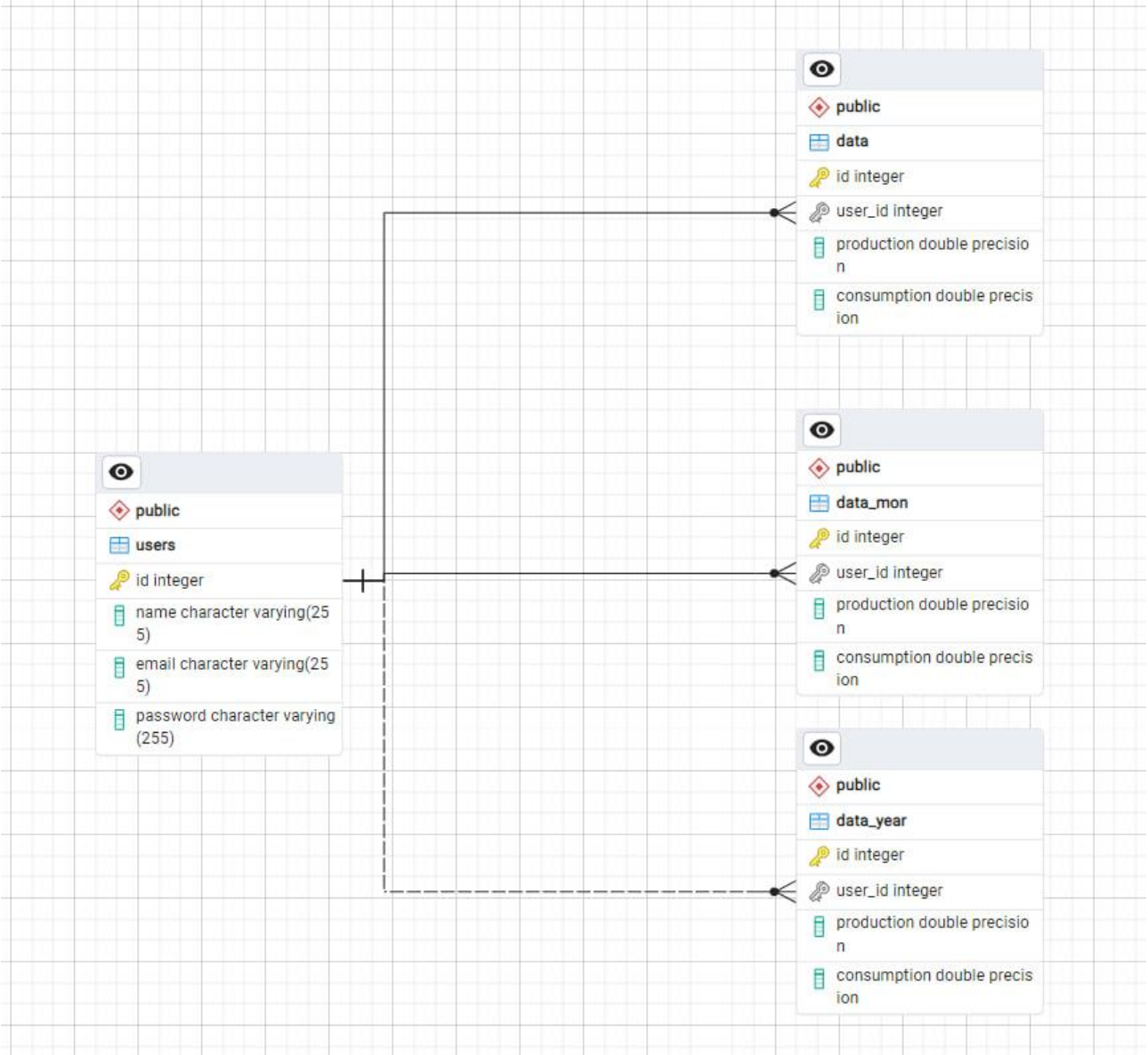

Рисунок 3.1 – Структура бази даних

Вся необхідна логіка для зв’язку з базою даних прописана в файлі server.js. Основні дії що виконуються в цьому файлі це:

- Запуск серверу.
- Підключення до СУБД.
- Створення запитів.

Сервер необхідний для отримання запитів із клієнтської частини. Запуск серверу відбувається за допомогою бібліотеки express.js.

Підключення до СУБД відбувається за допомогою класу Client бібліотеки pg. Цей клас забезпечує авторизацію та зв’язок із необхідною базою даних.

Запити створені для того щоб клієнтська частина могла працювати із інформацією що знаходиться у базі даних. Це може бути отримання даних(GET), додавання даних(POST), або видалення даних(DELETE). Наприклад при реєстрації нового користувача відбувається POST-запит, а для видалення DELETE-запит.

Також у цьому ж файлі прописані усі необхідні запити до бази даних.

Схема обробки запитів у веб-застосунку наведена на рис 3.2.

Рисунок 3.2 – Схема обробки запитів

При формуванні запиту на збереження паролів при реєстрації вони хешуються за допомогою алгоритму bcrypt. Такий підхід унеможливлює перехоплення паролю в ході передачі запиту чи навіть при збереженні у базі даних у випадку її викриття. Так відбувається через те , що такі шифри неможливо розшифрувати напряму. Замість цього функція bcrypt.compare при авторизації порівнює надане значення із тим що було зашифровано на початку.

3.3 Структура клієнтської частини веб-застосунку

Для створення клієнтської частини застосунку було використано фреймворк React. Для цього фреймворку окремі структурні елементи веб-застосунку є окремими компонентами які імпортуються до головного компонента App.jsx.

У табл. 3.1 наведено основні компоненти веб-застосунку та їх призначення.

Таблиця 3.1 – основні компоненти веб-застосунку

Назва елемента	Призначення
App.jsx	Основний компонент до якого під’єднуються інші компоненти та організовується маршрутизація.
BarChart.jsx	Створює стовпчатий графік.
LineChart.jsx	Створює лінійний графік.
PieChart.jsx	Створює круговий графік.
Dat.jsx	Оброблює дані по поточній генерації та споживанню електроенергії, розраховує кількість проданої/закупленої енергії, виводить дані на екран.
Footer.jsx	Створює футер веб-сторінки.
About.jsx	Створює головну сторінку.
Login.jsx	Створює форму авторизації та оброблює відповідні дані.
Delete.jsx	Створює необхідні компоненти форми та надсилає запит на видалення користувача до проміжного програмного забезпечення.

Продовження таблиці 3.1.

Назва елемента	Призначення
Registration.jsx	Створює форму реєстрації та надсилає запит на додавання отриманих даних користувача до проміжного програмного забезпечення.
Statistics.jsx	Отримує дані по генерації та споживанню за різні періоди часу та виводить відповідні графіки. Також розраховує дані по заробітку за зеленим тарифом.
Weather.jsx	Відправляє запит на API OpenWeather та отримує дані про погоду на наступні декілька днів. Після чого їх виводить та надає рекомендації відповідно до отриманих даних
InfoCard.jsx	Створює картки для інформативного та лаконічного відображення даних

Із отриманих компонентів створюються веб-сторінки які врешті забезпечують необхідний функціонал.

Список усіх сторінок:

- Головна.
- Авторизація.
- Реєстрація.
- Прогноз.
- Налаштування.
- Поточні показники
- Статистика.

Схема зв’язків між сторінками наведена на рис. 3.3

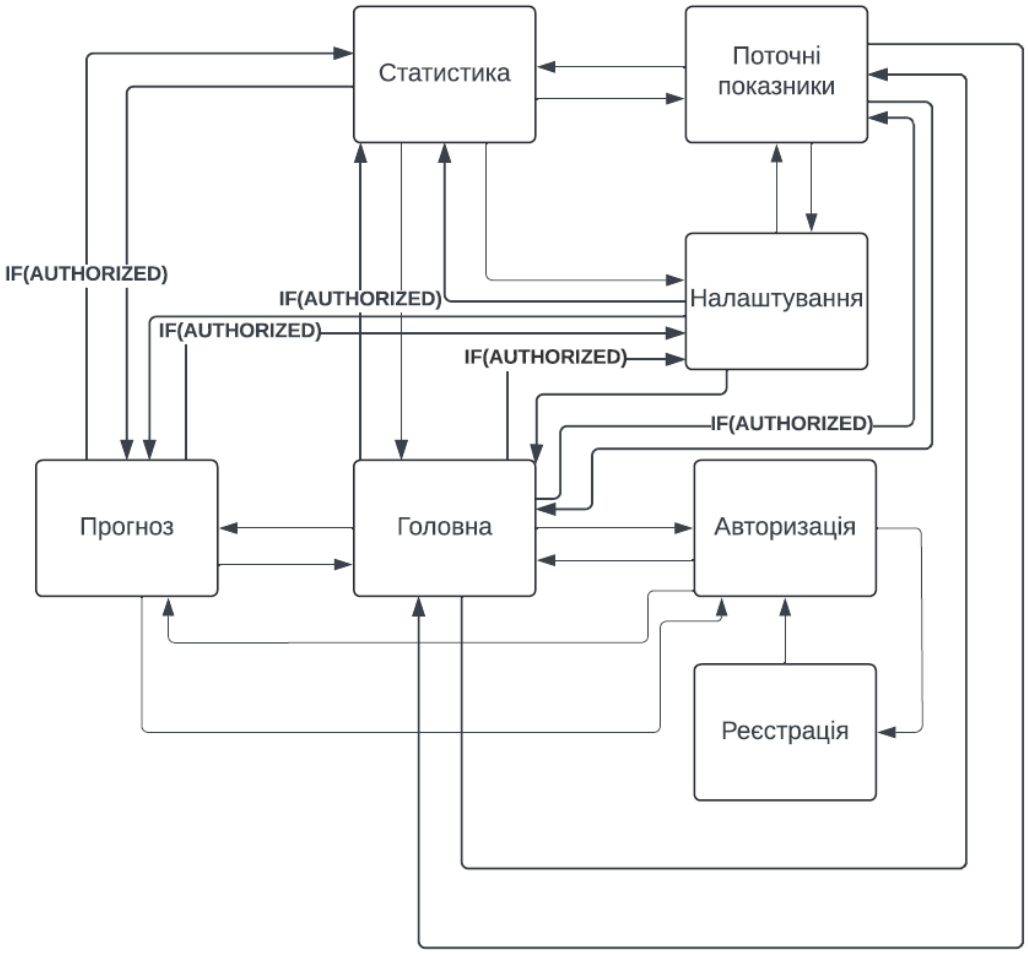

Рисунок 3.3 – Схема зв’язків між сторінками

3.4 Результат роботи

Після переходу до веб-застосунку користувач одразу потрапляє до головної сторінки. Ця сторінка носить інформаційний характер, на ній користувач може ознайомитись з основними характеристиками веб-застосунку. Після чого користувач може перейти до авторизації або переглянути прогноз. Прогноз є універсальним та не залежить від персональних даних через що й не потребує авторизації. Для усіх інших функцій необхідна авторизація.

На сторінці прогнозу можна обрати необхідне місто , після чого буде виведена інформація про хмарність та швидкість вітру та буде створено висновки

щодо ефективності генерації під час такої погоди. Також можна обрати дату серед найближчих п’яти днів.

При позитивних висновках колір тексту буде зелений, при негативних - червоний.

Приклад роботи сторінки “Прогноз” наведено на рис 3.4.

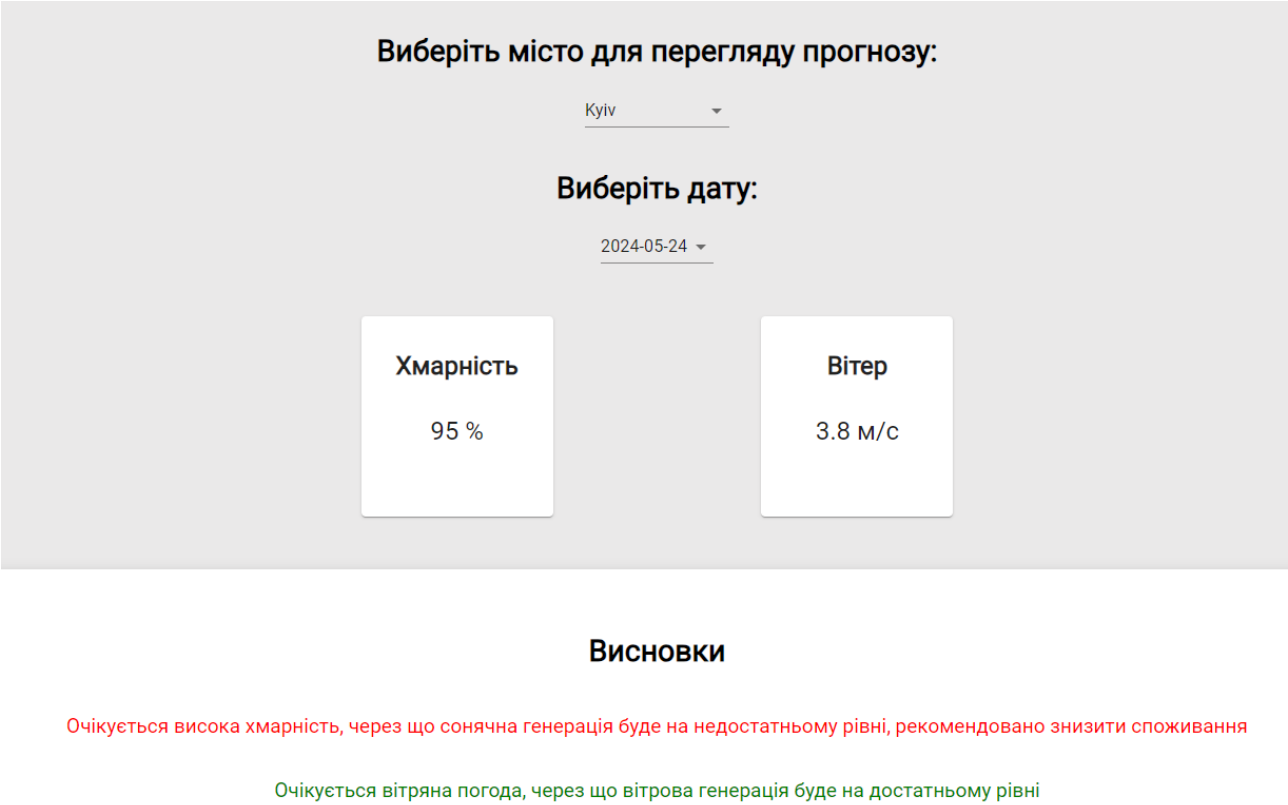

Рисунок 3.4 – Приклад роботи сторінки “Прогноз”

Тож наступним етапом є авторизація, якщо у користувача ще немає аккаунту то із сторінки авторизації можна перейти на сторінку реєстрації.

Після успішної авторизації користувач може перейти на сторінку “Поточні показники” та відслідковувати інформацію про поточну генерацію, споживання та продаж. Крім числових даних буде виведено ще й візуалізацію інформації у вигляді динамічного стовпчатого графіку який оновлюється кожні три секунди разом із числовими даними. Стовпчик генерації відображається зеленим, в той час як стовпчик споживання червоним кольором.

Стовпчик продажу на відміну від інших не має статичного кольору та змінює його в залежності від даних які надходять. При позитивних значеннях продажу стовпчик буде зеленого кольору, при від'ємних значеннях – червоного.

Приклад роботи сторінки "Поточні показники" наведено на рис 3.5.

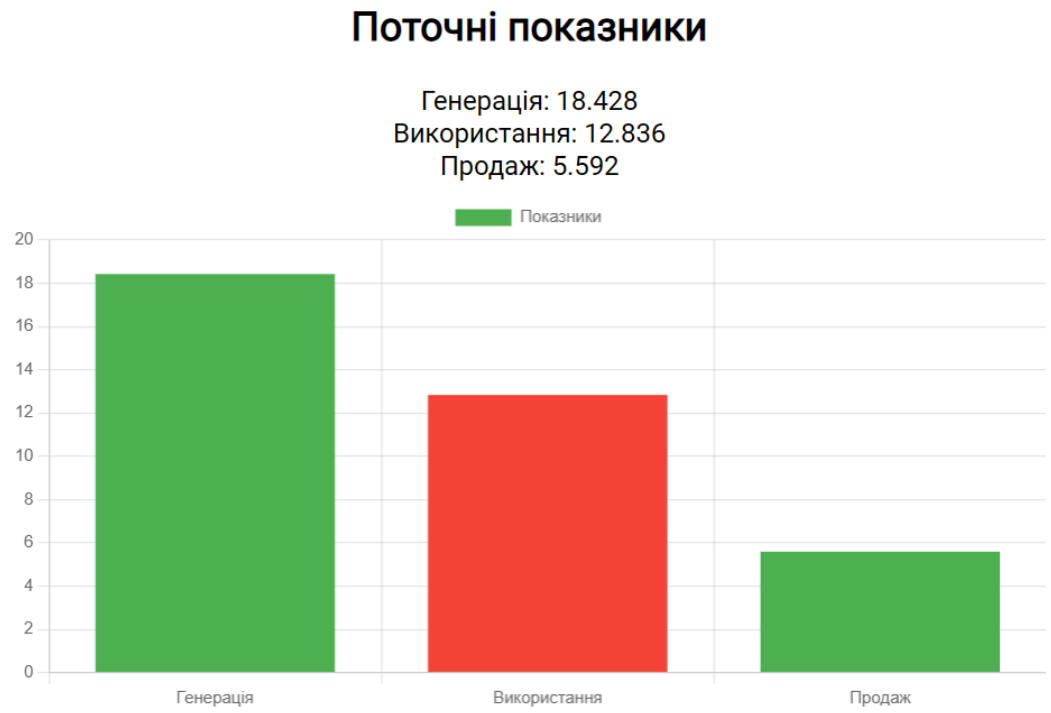

Рисунок 3.5 – Приклад роботи сторінки "Поточні показники"

Також авторизовані користувачі мають доступ до сторінки "Статистика". Це дуже корисний інструмент який стане в нагоді будь-якому активному споживачу оскільки він містить по 5 графіків за різні періоди часу, а саме за останню добу, місяць та рік. Ці графіки відображають вичерпні дані необхідні для оптимізації виробництва. Серед цих п'яти графіків:

- 3 стовпчаті.
- 1 лінійний.
- 1 круговий.

За допомогою стовпчатих графіків представлені базові дані по таким параметрам:

- Виробництво.
- Споживання.
- Продаж/покупка.

Дані для графіків отримуються із бази даних. Для створення даних по продажу виконуються обчислення власне під час виконання застосунку, як різниця виробництва та споживання.

За допомогою лінійного графіка виводяться дані по заробіткам/витратам електроенергії, саме такий вид графіка був обраний через те що на ньому найкраще видно відносну різницю по відповідному показнику за кожен день. Дані для цього графіку також обчислюються власне під час виконання програми як добуток продажу/покупки на зелений тариф.

За допомогою кругового графіка виводиться інформація про співвідношення споживання до продажу. Саме цей тип графіка було обрано через те що він найкраще підходить для візуалізації співвідношень.

Приклад графіків по статистичним даним наведено на рис. 3.6.

Рисунок 3.6 – Приклад графіків по статистичним даним

Також важливим функціоналом для будь-якого веб-застосунку є забезпечення права на забуття персональних даних.

Тож для забезпечення права на забуття персональних даних реалізовано функціонал видалення даних про поточного користувача із бази даних в один клік. Спочатку видаляються статистичні дані , після чого видаляються й особисті дані користувача. Для цього користувач має перейти в налаштування та натиснути кнопку “Видалити користувача”.

Форма для видалення користувача наведена рис. 3.7.

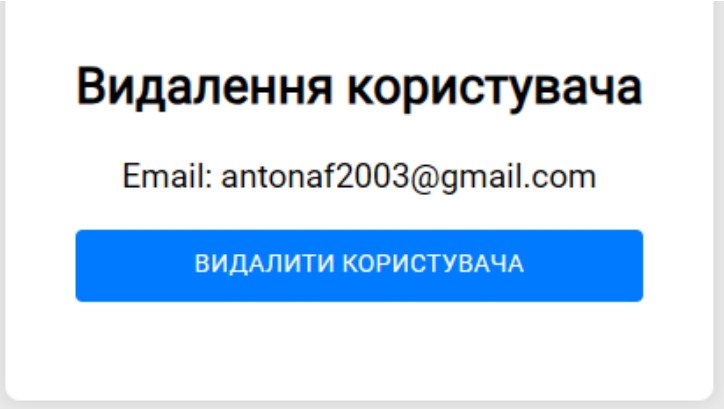

Рисунок 3.7 – Форма для видалення користувача

3.5 Висновки до розділу

У цьому розділі було проведено розробку та впровадження веб-застосунку системи моніторингу рівня балансу споживання активних споживачів в Smart Grid мережах.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи бакалавра було виконано такі завдання:

- Досліджено особливості Smart Grid мереж.
- Проведено аналітичний огляд теоретичних основ побудови систем моніторингу рівня балансу споживання активних споживачів.
- Проаналізовано програмно-архітектурні рішення задля розробки веб-застосунків.
- Розроблено та впроваджено веб-застосунок системи моніторингу рівня балансу споживання активних споживачів в Smart Grid мережах.

З дипломної роботи можна зробити висновок що створення веб-застосунку системи моніторингу рівня балансу споживання активних споживачів в Smart Grid мережах є актуальним як ніколи, оскільки Smart Grid мережі представляють собою революційне рішення споконвічних проблем енергетики таких як обмежена ефективність ресурсів та нестабільність мережі. Крім того було виявлено значне зростання кількості дрібних виробників електроенергії, яким просто необхідний простий , зручний та доступний у мережі інструмент для відстеження основних показників виробництва. Також дослідження показали що оптимальним варіантом для забезпечення потреб активних споживачів є саме веб-застосунок.

Серед іншого було виявлено, що розробку веб-застосунків доцільно вести не класичним методом, а за допомогою прогресивних інструментів – фреймворків. Для візуалізації ж даних серед них найбільш оптимальним виявився React.

Загалом досягнуто поставлених вимог щодо реалізації функціоналу у веб-застосунку. Таким чином цей веб-застосунок має простий та зручний інтерфейс, а інформація яку надає веб-застосунок є корисною та зрозумілою, що робить цей веб-застосунок чудовим інструментом для активних споживачів, які хочуть оптимізувати виробництво та споживання електроенергії.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Телекомунікаційні системи та мережі / В. В. Поповський, О. В. Левешко, М. Д. Плотніков та ін. ; Львів : СМІТ, 2018. 134 с. (дата звернення: 08.12.2023)

2. Smart meters. *José Manuel Carou Álvarez* : веб-сайт. URL: https://www.sciencedirect.com/science/article/abs/pii/B9780128212042000672 (дата звернення: 08.12.2023)

3. What is Advanced Metering Infrastructure (AMI)? *Cristina Tuser* : веб-сайт. URL: https://www.wwdmag.com/what-is-articles/article/10940067/what-is-advanced-metering-infrastructure-ami (дата звернення: 08.12.2023)

4. Distribution and outage management systems. *Bas Kruimer* : веб-сайт. URL: https://www.dnv.com/services/distribution-and-outage-management-systems-5311 (дата звернення: 11.12.2023)

5. Demand Response Management, a critical component of the future smart grid. *Lao Ren* : веб-сайт. URL: Режим доступу до ресурсу: https://www.smart-energy.com/smart-grid/demand-response-management-a-critical-component-of-the-future-smart-grid/ (дата звернення: 11.12.2023)

6. Sioshansi F. P. Smart Grid: Integrating Renewable, Distributed and Efficient Energy. Cambridge: Academic Press, 2011. 585 с. (дата звернення: 11.12.2023)

7. Lawrence E. Renewable Energy Integration.Amsterdam Inc., 2017. 570 с. (дата звернення: 12.12.2023)

8. Prosumer. *Enerdynamics*: веб-сайт. URL: https://energyknowledgebase.com/topics/prosumer.asp (дата звернення: 12.12.2023)

9. Consumer vs Prosumer: What's the Difference? *Sarah Harman* : веб-сайт. URL: https://www.energy.gov/eere/articles/consumer-vs-prosumer-whats-difference (дата звернення: 12.12.2023)

10. Smart Grids: An Optimised Electric Power System. *Jerry Jackson* : веб-сайт. URL: https://www.sciencedirect.com/science/article/abs/pii/B9780080994246000284 (дата звернення: 12.12.2023)

11. Афанасьєв А.А., Зайцев Є.О. Інформаційна система моніторингу рівня балансу споживання проактивних споживачів в smart grid мережах. Прикладні системи та технології в інформаційному суспільстві: зб. тез доповідей і наук. повідомл. учасників VII Міжнародної науково-практичної конференції, Київ, 29 вересня 2023 р., К.: Київський нац. ун-т ім. Тараса Шевченка, 2023. С. 13-18. (дата звернення: 16.12.2023)

12. Зайцев Є., Березниченко В., Закусило С., Антоненко А. SMART засоби визначення аварійних станів в розподільних електричних мережах міст. Таврійський науковий вісник. Серія: Технічні науки. №5, С. 3-12. DOI: https://doi.org/10.32851/tnv-tech.2022.5.1. (дата звернення: 16.12.2023)

13. Morstyn T. Using peer-to-peer energy-trading platforms to incentivize prosumers to form federated power plants. Oxford: Oxford university, 2018 102 с. (дата звернення: 17.12.2023)

14. Power Monitoring Expert. *Schneider electric* : веб-сайт. URL: https://www.se.com/ua/uk/product-range/65404-ecostruxure-power-monitoring-expert/#overview. (дата звернення: 21.12.2023)

15. Zabbix Documentation. *Zabbix ZIA* : веб-сайт. URL: https://www.zabbix.com/manuals (дата звернення: 21.12.2023)

16. Network Performance Monitor web-site. *SolarWinds Inc* : веб-сайт. URL: https://www.solarwinds.com/network-performance-monitor (дата звернення: 21.12.2023)

17. Тіттел Е., Ноубл Д. HTML, XHTML і CSS для чайників, 7-е видання HTML, XHTML & CSS For Dummies, 7th Edition.: «Діалектика», 2011. 400 с. (дата звернення: 06.01.2024)

18. What is HTML. *Javatpoint*: веб-сайт. URL: https://www.javatpoint.com/what-is-html (дата звернення: 08.01.2024)

19. Документація HTML, CSS. *w3schools* : веб-сайт. URL: https://www.w3schools.com/default.asp (дата звернення: 08.01.2024)

20. What is CSS. *Javatpoint*: веб-сайт. URL: https://www.javatpoint.com/whatis-css (дата звернення: 08.01.2024)

21. What is JavaScript. *Javatpoint*: веб-сайт. URL: https://www.javatpoint.com/javascript-tutorial (дата звернення: 08.01.2024)

22. Документація React. *Meta* : веб-сайт. URL: https://react.dev/blog/2023/03/16/introducing-react-dev (дата звернення: 13.01.2024)

23. Документація ChartJS. *Nick Downie* : веб-сайт. URL: https://www.chartjs.org/docs/latest/ (дата звернення: 13.01.2024)

24. React JS: Advantages and Disadvantages? *Wojciech Baranowski*: веб-сайт. URL: https://massivepixel.io/blog/react-advantages-disadvantages/ (дата звернення: 13.01.2024)

25. Pros and Cons of ReactJS. *Javatpoint* : веб-сайт. URL: https://www.javatpoint.com/pros-and-cons-of-react (дата звернення: 13.01.2024)

26. Angular docs. Google : веб-сайт. URL: https://angular.io/docs (дата звернення: 14.01.2024)

27. MVC Pattern. *Tutorialspoint*: веб-сайт. URL: https://www.tutorialspoint.com/design_pattern/mvc_pattern.htm (дата звернення: 14.01.2024)

28. Vue.js documentation. *devdocs* : веб-сайт. URL: https://devdocs.io/vue~3/ (дата звернення: 15.01.2024)

29. Date C. J. An Introduction to Database Systems. London : Pearson, 2003. 520 с. (дата звернення: 15.01.2024)

30. Глушаков С. В. Бази даних / С. В. Глушаков, Д. В. Ломотько: Фоліо, 2017. 504 с. (дата звернення: 20.01.2024)

31. What is a NoSQL database? *IBM* : веб-сайт. URL: https://www.ibm.com/topics/nosql-databases (дата звернення: 20.01.2024)

32. What is a Columnar Database? Examples, Benefits, Differences & More! *Atlan* : веб-сайт. URL: https://atlan.com/what-is/columnar-database/ (дата звернення: 20.01.2024)

33. HBase Reference Guide. *Apache* : веб-сайт. URL: https://hbase.apache.org/book.html (дата звернення: 20.01.2024)

34. What is Apache CassandraRegistered? *Apache* : веб-сайт. URL: https://cassandra.apache.org/_/index.html (дата звернення: 20.01.2024)

35. Bigtable documentation. *Google* : веб-сайт. URL: https://cloud.google.com/bigtable/docs (дата звернення: 20.01.2024)

36. What is a graph database? *AWS* : веб-сайт. URL: https://aws.amazon.com/nosql/graph/# (дата звернення: 21.01.2024)

37. Documentation. *Neo4j* : веб-сайт. URL: https://neo4j.com/docs/ (дата звернення: 21.01.2024)

38. OrientDB Manual - version 3.2.24. *SAP* : веб-сайт. URL: https://orientdb.com/docs/last/index.html (дата звернення: 21.01.2024)

39. What is ArangoDB? *ArangoDB GmbH* : веб-сайт. URL: https://docs.arangodb.com/3.11/about-arangodb/ (дата звернення: 21.01.2024)

40. What is a key-value database? *AWS* : веб-сайт. URL: https://aws.amazon.com/nosql/key-value/ (дата звернення: 25.01.2024)

41. Redis docs. *VMware* : веб-сайт. URL: https://redis.io/docs/latest/ (дата звернення: 25.01.2024)

42. Amazon DynamoDB Documentation. *AWS* : веб-сайт. URL: https://docs.aws.amazon.com/dynamodb/ (дата звернення: 25.01.2024)

43. RIAK DOCS. *Basho Technologies* : веб-сайт. URL: https://docs.riak.com/ (дата звернення: 25.01.2024)

44. What is a document database? *AWS* : веб-сайт. URL: https://aws.amazon.com/nosql/document/ (дата звернення: 27.01.2024)

45. MongoDB Documentation. *MongoDB Inc.* : веб-сайт. URL: https://www.mongodb.com/docs/ (дата звернення: 27.01.2024)

46. CouchDB 3.3.3 Documentation. *Apache* : веб-сайт. URL: https://docs.couchdb.org/en/stable/ (дата звернення: 27.01.2024)

47. Elasticsearch Guide. *Shay Banon* : веб-сайт. URL: https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html (дата звернення: 27.01.2024)

48. What is a Relational Database (RDBMS)? *Oracle* : веб-сайт. URL: https://www.oracle.com/database/what-is-a-relational-database/ (дата звернення: 30.01.2024)

49. SQL Server technical documentation. *Microsoft* : веб-сайт. URL: https://learn.microsoft.com/en-us/sql/sql-server/?view=sql-server-ver16 (дата звернення: 30.01.2024)

50. Documentation. *PostgreSQL* : веб-сайт. URL: https://www.postgresql.org/docs/ (дата звернення: 30.01.2024)

51. MySQL documentation. *Oracle* : веб-сайт. URL: https://dev.mysql.com/doc/ (дата звернення: 30.01.2024)

52. Express.js documentation. *TJ Holowaychuk* : веб-сайт. URL: https://expressjs.com/ (дата звернення: 31.01.2024)

ДОДАТКИ

ДОДАТОК А

```
app.use(express.json());
app.use(cors());

const client = new Client({
  user: 'postgres',
  host: 'localhost',
  database: 'my_database',
  password: '0000',
});

client.connect();

app.listen(port, () => {
  console.log(`Server listening on port ${port}`);
});
```

Рисунок А1. Реалізація з’єднання з базою даних

```
useEffect(() => {
  Codeium: Refactor | Explain | Generate JSDoc | X
  const fetchWeatherForecast = async () => {
    try {
      const response = await axios.get(
        `https://api.openweathermap.org/data/2.5/forecast?q=${selectedCity}&appid=${API_KEY}&units=metric`
      );
      const tomorrowWeather = response.data.list.find(item => item.dt_txt.includes(tomorrow));
      setWeatherData(tomorrowWeather);
    } catch (error) {
      console.error('Error fetching weather forecast:', error);
    }
  };

  fetchWeatherForecast();
}, [selectedCity, API_KEY, tomorrow]);
```

Рисунок А2. Реалізація з’єднання та отримання даних з API

ДОДАТОК Б

```
app.post('/register', async (req, res) => {
  console.log(req.body);
  const { name, email, password, confirmpassword, network_id } = req.body;
  if (!name || !email || !password || !confirmpassword || !network_id) {
    res.status(400).json({ error: 'Будь ласка, заповніть всі поля' });
    console.log('Будь ласка, заповніть всі поля');
    return;
  }
  console.log(req.body);
  if (password !== confirmpassword) {
    res.status(400).json({ error: 'Паролі не співпадають' });
    return;
  }
  const existingUser = await client.query('SELECT * FROM users WHERE email = $1', [email]);
    if (existingUser.rows.length > 0) {
      res.status(400).json({ error: 'Користувач з таким email вже існує' });
      return;
    }
  try {
    const hashedPassword = await bcrypt.hash(password, 10);
    const result = await client.query('INSERT INTO users (name, email, password, network_id) VALUES ($1, $2, $3, $4) RETURNING *',
    [name, email, hashedPassword, network_id]);
    const newUser = result.rows[0];
    res.status(201).json(newUser);
  } catch (error) {
    console.error('Помилка реєстрації:', error);
    res.status(500).json({ error: 'Помилка сервера' });
  }
});
```

Рисунок Б1. Реалізація функціоналу запита на реєстрацію

```
app.post('/login', async (req, res) => {
  const { email, password } = req.body;
  if (!email || !password) {
    res.status(400).send('Необхідно ввести електронну пошту та пароль');

    return;
  }

  try {
    const result = await client.query('SELECT * FROM users WHERE email = $1', [email]);
    const user = result.rows[0];
    if (!user) {
      res.status(401).send('Невірна електронна пошта або пароль');
    } else {

      const isMatch = await bcrypt.compare(password, user.password);

      if (isMatch) {
        res.status(200).send('Успішна авторизація');
      } else {
        res.status(401).send('Невірна електронна пошта або пароль');

      }
    }
  } catch (error) {
    console.error('Помилка авторизації:', error);
    res.status(500).send('Помилка сервера');
  }
});
```

Рисунок Б2. Реалізація функціоналу запита на авторизацію

ДОДАТОК В

```
<div>
  <h2>Видалення користувача</h2>
  <p>Email: {useremail}</p>
  <Button onClick={handleDeleteUser}>Видалити користувача</Button>
  {error && <p style={{ color: 'red' }}>{error}</p>}
</div>
);
```

Рисунок В1. Представлення HTML форми видалення користувача

```
app.delete('/users/:email', async (req, res) => {
  const useremail = req.params.email;
  try {

    await client.query('DELETE FROM data WHERE user_id = (SELECT id FROM users WHERE email = $1)', [useremail]);
    await client.query('DELETE FROM data_mon WHERE user_id = (SELECT id FROM users WHERE email = $1)', [useremail]);
    await client.query('DELETE FROM data_year WHERE user_id = (SELECT id FROM users WHERE email = $1)', [useremail]);

    const result = await client.query('DELETE FROM users WHERE email = $1 RETURNING *', [useremail]);
    const deletedUser = result.rows[0];

    if (!deletedUser) {
      res.status(404).send('Користувач не знайдений');
    } else {
      res.status(200).json(deletedUser);
    }
  } catch (error) {
    console.error('Помилка видалення користувача:', error);
    res.status(500).send('Помилка сервера');
  }
});
```

Рисунок В2. Реалізація функціоналу запита на видалення користувача