

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЧЕРНІВЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ЮРІЯ ФЕДЬКОВИЧА**

**Факультет математики та інформатики
кафедра математичного моделювання**

**РОЗРОБКА ВЕБДОДАТКА ДЛЯ КЛАСТЕРНОГО АНАЛІЗУ
СПОЖИВАННЯ ЕЛЕКТРОЕНЕРГІЇ**

Кваліфікаційна робота

Рівень вищої освіти – другий (магістерський)

Виконав:

студент 2 курсу, 607 групи

Крештанович

Микита

Андрійович

Керівник:

доцент Дорошенко І.В

До захисту допущено

на засіданні кафедри

протокол № 5 від 26 листопада 2024 р.

Зав. кафедрою _____ проф. Черевко І.М.

Чернівці – 2024

Анотація

У роботі проведено огляд сервісу для аналізу споживання електроенергії, детально описано використані технології та методи обробки даних. Розглянуто й проаналізовано підходи до розробки додатків, їхні особливості та переваги. У практичній частині розроблено вебдодаток із використанням мов програмування Python, TypeScript та JavaScript, що забезпечує отримання, обробку та візуалізацію даних. Для створення додатка застосовано архітектурний підхід MVC та REST API, що сприяє модульності, масштабованості та зручності використання.

Ключові слова: Python, TypeScript, JavaScript, кластерний аналіз, DBSCAN, k-means, прогнозування.

Abstract

The study provides an overview of a service for analyzing electricity consumption, detailing the technologies and methods used for data processing. It examines and evaluates various approaches to application development, highlighting their features and advantages. As part of the practical implementation, a web application was developed using Python, TypeScript, and JavaScript to facilitate data collection, processing, and visualization. The application employs the MVC architectural pattern and REST API, ensuring modularity, scalability, and ease of use.

Keywords: Python, TypeScript, JavaScript, cluster analysis, DBSCAN, k-means, prognostication.

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів наукових досліджень інших авторів мають посилання на відповідне джерело.

_____ М.А. Крештанович

Зміст

ВСТУП.....	4
РОЗДІЛ 1 ОПИС ВИКОРИСТАНИХ ТЕХНОЛОГІЙ.....	6
1.1 WebStorm.....	6
1.3 Vue.js.....	7
1.4 Nest.js.....	9
1.4.1 Серверна частина на Nest.js.....	10
1.5 Python та FastApi.....	13
1.6 Системи баз даних.....	14
1.6.1 Вибір бази даних для зберігання даних електроенергії.....	15
1.6.2 Реляційні та нереляційні бази даних.....	16
1.7. ORM для роботи з БД (SQLAlchemy, TypeORM).....	18
1.7.1 SQLAlchemy (Python).....	18
1.7.2 TypeORM (Nest.js).....	19
РОЗДІЛ 2. ФОРМУЛЮВАННЯ ЗАДАЧІ ТА СТВОРЕННЯ ПРОГРАМНОГО ПРОДУКТУ.....	21
2.1. Постановка задачі.....	21
2.1.1 Опис проблеми аналізу споживання електроенергії.....	21
2.1.2 Опис необхідних функцій вебдодатку.....	24
2.2 Проектування архітектури вебдодатку.....	26
2.2.1 Вибір архітектури (MVC, REST).....	26
2.2.2 Опис основних компонентів.....	28
2.3 Створення програмного продукту.....	31
2.3.1 Розробка фронтенду (Vue.js).....	31
2.3.2 Розробка бекенду (Nest.js).....	36
2.3.3 Розробка сервісу аналітики даних (FastApi).....	38
РОЗДІЛ 3. ПРОГРАМНИЙ ПРОДУКТ.....	43
3.1. Приклад роботи програмного продукту.....	43
3.2 Застосування програми.....	56
ВИСНОВКИ.....	58
ДОДАТКИ.....	60

ВСТУП

Ефективне використання енергоресурсів є однією з ключових проблем сучасного світу. У зв'язку з постійним зростанням споживання електроенергії, особливо в міських агломераціях, постає необхідність у створенні інструментів для моніторингу та аналізу використання енергоресурсів. Це дозволяє не лише оцінити поточний стан споживання, а й сприяти прийняттю рішень щодо оптимізації енергетичних витрат, зниження перевантаження мереж і впровадження енергоефективних рішень.

Розроблений вебдодаток спрямований на вирішення цих завдань. Основною його метою є створення зручного інструменту для постачальників електроенергії, який дозволяє завантажувати щомісячні звіти про споживання електроенергії в різних містах та здійснювати детальний аналіз даних. Зокрема, програма надає можливість оцінювати сумарне споживання електроенергії по будинках за кожен місяць, що є важливим для визначення загальних тенденцій, виявлення аномалій і розробки стратегій оптимізації енергоресурсів.

Додаток базується на сучасних технологіях, таких як Vue.js для створення інтерактивного інтерфейсу користувача, Nest.js для серверної логіки та Python FastAPI для виконання обчислень і роботи з великими обсягами даних. Така комбінація технологій забезпечує високу продуктивність, гнучкість і масштабованість системи, що особливо важливо в умовах зростання кількості даних і збільшення потреб користувачів.

Розробка цього додатка спрямована на вирішення таких проблем, як недостатність інструментів для зручної обробки великих обсягів даних про споживання електроенергії, відсутність доступних інструментів для виконання точного аналізу споживання та складність у визначенні ефективності використання енергоресурсів. Завдяки впровадженню вебдодатка постачальники отримають можливість краще контролювати процеси споживання, що, в свою чергу, сприятиме підвищенню енергетичної ефективності.

Робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. У першому розділі описуються технології, що були застосовані для розробки програми. Другий розділ є ключовим і містить опис алгоритму створення додатка та його користувацького інтерфейсу.

РОЗДІЛ.1 ОПИС ВИКОРИСТАНИХ ТЕХНОЛОГІЙ

Для розробки вебдодатка, який аналізує використання електроенергії, було використано кілька сучасних технологій та інструментів, що забезпечують зручність і продуктивність у створенні програмного забезпечення.

Інтегровані середовища розробки (IDE)

У процесі роботи над проєктом використовувалися два інтегровані середовища розробки: WebStorm для роботи з фронтендом та бекендом на JavaScript/TypeScript й PyCharm для розробки обчислювальних модулів на Python.

1.1 WebStorm



Рис.1.1. Логотип WebStorm

WebStorm – середовище розробки, оптимізоване для роботи з веб-технологіями. Воно використовувалося для написання коду фронтенд-додатка на Vue.js, а також для серверної частини на Nest.js. WebStorm забезпечує потужну підтримку синтаксису, автодоповнення коду, перевірку помилок у реальному часі та інтеграцію з системами контролю версій, такими як Git. Завдяки цим функціям розробка стала ефективною та швидшою.

1.2 Pycharm



Рис.1.2. Логотип Pycharm

PyCharm – інтегроване середовище розробки для мови програмування Python, яке було використано для реалізації модулів обчислення. PyCharm надає широкі можливості для роботи з базами даних, автоматизації процесів тестування та інтеграції з іншими інструментами розробки. Завдяки підтримці синтаксису Python та фреймворку FastAPI, PyCharm спростив розробку серверних модулів, відповідальних за обробку великих обсягів даних і виконання складних обчислень.

Обидві програми є комерційними, однак для роботи над проектом вони були доступні безкоштовно завдяки студентській підписці. Це дозволило отримати доступ до всіх функцій середовищ розробки без фінансових витрат, що особливо актуально для молодих фахівців і студентів.

1.3 Vue.js

Для реалізації інтерфейсу користувача вебдодатка було обрано Vue.js — прогресивний JavaScript-фреймворк, призначений для створення інтерактивних веб-застосунків. Vue.js поєднує простоту використання з потужними можливостями, що робить його одним із найпопулярніших рішень для фронтенд-розробки.



Рис.1.3. Логотип Vue.js

Основні переваги Vue.js, які були використані в проекті:

1. Реактивність та прив'язка даних

Vue.js забезпечує автоматичне оновлення інтерфейсу користувача у разі змін даних. Це дозволило створити динамічний інтерфейс, який відображає актуальні дані щодо споживання електроенергії без

необхідності вручну оновлювати сторінки. Наприклад, щомісячні звіти про споживання енергії миттєво відображаються на графіках після завантаження даних постачальниками.

2. Компонентна архітектура

Інтерфейс додатка було розроблено у вигляді окремих багаторазових компонентів, таких як таблиці для відображення звітів, інтерактивні графіки споживання та форми для завантаження даних. Це не лише спростило структуру коду, а й зробило його зручним для подальшої підтримки та розширення функціональності.

3. Інтеграція з бібліотеками та інструментами

У проєкті використовувалися додаткові бібліотеки для побудови графіків і візуалізації даних, наприклад, Chart.js. Vue.js легко інтегрується з такими інструментами, що дозволило реалізувати функціонал зручного візуального аналізу даних. Зокрема, графіки відображають сумарне споживання електроенергії за місяць у різних містах або окремих будинках[11].

4. Швидкість та ефективність

Завдяки невеликому розміру ядра та ефективному механізму оновлення DOM (Virtual DOM), Vue.js забезпечує високу продуктивність інтерфейсу навіть при обробці великих обсягів даних. Це особливо важливо для роботи із щомісячними звітами, що містять значні обсяги інформації.

5. Зручний синтаксис

Vue.js поєднує HTML, CSS і JavaScript у рамках одного файлу компоненту, що значно спрощує розробку. Завдяки цьому процес створення та налагодження інтерфейсу був інтуїтивним і швидким[14].

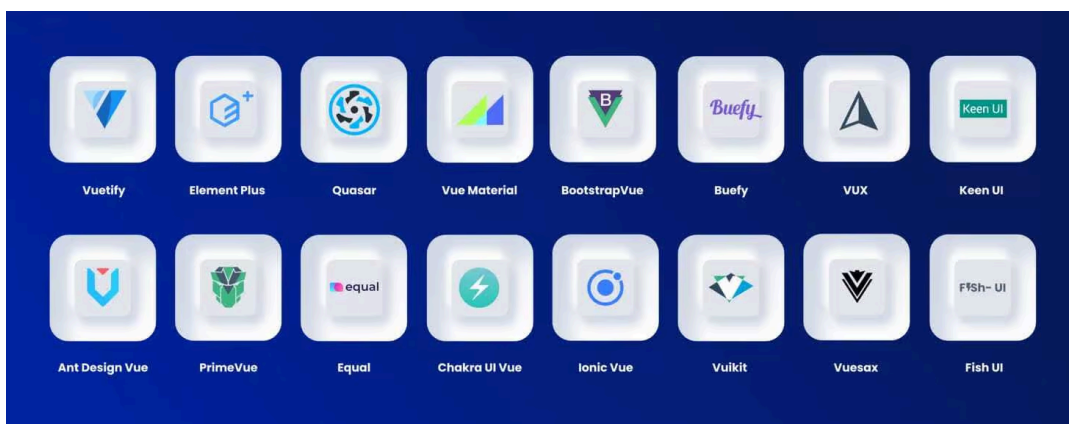


Рис.1.4. Бібліотеки та Фреймворки на Vue.js

1.4 Nest.js



Рис.1.5. Логотип Nest.js

Для створення серверної частини додатка було використано Nest.js — сучасний TypeScript-фреймворк, розроблений для створення масштабованих і добре структурованих серверних застосунків. Nest.js забезпечує гнучкість і високу продуктивність завдяки модульній архітектурі, яка дозволяє легко організувати код навіть у великих проєктах[3].

Основні переваги Nest.js, які були використані в проєкті:

1. Підтримка TypeScript

Завдяки використанню TypeScript, Nest.js забезпечує статичну типізацію, що підвищує надійність коду та допомагає уникнути багатьох помилок ще на етапі розробки. Це особливо важливо в

складних застосунках, які обробляють великі обсяги даних. У цьому проєкті типізація допомогла забезпечити точність передачі даних між фронтендом та сервером.

2. Інтеграція з базами даних

У цьому проєкті для зберігання даних про споживання електроенергії використовувалася реляційна база даних PostgreSQL. Nest.js надає можливість легко інтегрувати ORM-бібліотеки, такі як TypeORM або Prisma, для роботи з базами даних. У даному випадку це дозволило ефективно організувати збереження та обробку даних про щомісячне споживання електроенергії[9].

3. Масштабованість

Завдяки використанню Nest.js додаток легко масштабувати для роботи з великими обсягами даних або зростаючою кількістю користувачів. Модульна структура дозволяє додавати нові функції без значного впливу на вже існуючий функціонал[3].

4. Реалізація REST API

У цьому проєкті серверна частина була реалізована у вигляді REST API. Nest.js дозволяє легко створювати контролери для обробки HTTP-запитів, що зробило взаємодію між фронтендом і сервером швидкою та ефективною. Наприклад, постачальники електроенергії можуть завантажувати звіти через API, а також отримувати дані про споживання для подальшого аналізу.

5. Мідлвари та обробка помилок

Nest.js забезпечує зручну реалізацію мідлварів для попередньої обробки запитів, наприклад, перевірки автентифікації користувачів чи валідації даних. Це допомогло захистити сервер від помилкових чи шкідливих запитів.

1.4.1 Серверна частина на Nest.js

Авторизація користувачів

Для авторизації користувачів використовується механізм JWT

токенів. Коли користувач реєструється або входить в систему, він отримує JWT токен, який зберігається в браузері (у вигляді cookie або в локальному сховищі) і передається разом із кожним запитом на сервер через заголовок *Authorization*.

Кожен запит, що вимагає авторизації, перевіряється на наявність цього токена. Якщо токен валідний, то користувач отримує доступ до захищених ендпоінтів. Для цього в Nest.js використовуються бібліотеки Passport і JWT, які забезпечують безпеку, валідність токенів і дозволяють реалізувати механізм аутентифікації.

Реалізація виглядає так:

- Реєстрація: Користувач вводить свої дані (ім'я, електронну пошту, пароль), сервер хешує пароль і зберігає в базі даних.
- Аутентифікація: Після введення правильних даних для входу генерується JWT токен, який повертається користувачеві.
- Валідація токена: Кожен наступний запит до серверу перевіряється на наявність валідного токена. Якщо токен не валідний, доступ до ендпоінту заборонений.

Таким чином, кожен запит, що вимагає доступу до приватних даних, отримує від користувача токен для перевірки його автентичності.

Робота з TypeORM

Для роботи з базою даних PostgreSQL використовується TypeORM, який інтегровано в проект. TypeORM дозволяє створювати, зберігати та вибирати дані з бази даних за допомогою об'єктно-реляційного відображення[10].

Наприклад, для отримання статистики про споживання електроенергії для конкретного міста за певний період часу (рік або місяць) я використовую TypeORM для вибірки даних з таблиці, що зберігає інформацію про споживання. Для цього створено відповідні запити до бази даних, які фільтрують дані за вказаними параметрами, такими як рік і місяць, та автоматично їх підсумовують.

Отримання статистичних даних

Всі обчислення, такі як кластерний аналіз і пошук аномалій у даних, виконуються на стороні Python, де для цього використовується відповідна логіка для обробки даних (зокрема за допомогою бібліотек, таких як Scikit-learn або Pandas) [1], [4]. Після обробки, Python зберігає результати в базі даних, і Nest.js витягує ці дані через ендпоінти для відображення на фронтенді.

- Наприклад, після того, як Python виконує кластеризацію даних споживання електроенергії, результат (класи споживання для різних міст) зберігається в базі даних.
- Nest.js запитує ці дані з бази, а також генерує відповідні ендпоінти для того, щоб передати їх на фронтенд.

У цьому випадку Nest.js передає дані як для простого відображення на фронті, наприклад, кількість міст і їх назви, так і для побудови більш складних графіків, де відображаються результати кластеризації, що були попередньо оброблені на Python. Дані кластеризації, як правило, включають групи міст, де спостерігаються подібні патерни споживання, що дає змогу візуалізувати різні варіанти споживання та виявляти аномалії.

Таким чином, Nest.js відіграє роль шлюзу між базою даних, Python-сервісом і фронтендом, обробляючи запити, виконуючи аутентифікацію, отримуючи необхідні дані з бази та передаючи їх на фронтенд для подальшої візуалізації та аналізу

1.5 Python та FastAPI



Рис.1.6. Логотипи Python та FastAPI

Для реалізації модулів обробки даних у додатку було використано Python та його веб-фреймворк FastAPI. Python відомий своєю універсальністю та багатою екосистемою бібліотек для роботи з даними, що дозволило ефективно реалізувати складну обробку інформації про електроспоживання. FastAPI було обрано завдяки його високій продуктивності, простоті в роботі та інтеграції з асинхронними операціями.

Основні завдання, реалізовані за допомогою Python та FastAPI:

1. Обробка великих обсягів даних

Python забезпечив зручний і ефективний інструментарій для роботи з великими наборами даних, завантаженими постачальниками електроенергії у вигляді CSV-файлів. З використанням бібліотек **pandas** та **NumPy** були реалізовані операції з очищення, агрегації та підготовки даних для подальшого аналізу[1,6,13].

2. Кластерний аналіз

Для виявлення закономірностей у даних було використано кластерний аналіз. Застосування алгоритмів із бібліотеки **scikit-learn** дозволило групувати об'єкти (наприклад, будинки чи міста) за схожістю у споживанні електроенергії. Це дало змогу ідентифікувати основні сегменти користувачів із подібними характеристиками споживання[4].

3. Виявлення аномалій

З метою виявлення аномалій у споживанні електроенергії були реалізовані алгоритми пошуку відхилень. Наприклад, із використанням методів **DBSCAN** можна було ідентифікувати будинки з нетиповими показниками споживання. Це дозволяє оперативно виявляти проблеми, такі як некоректні дані чи технічні несправності.

4. API для обчислень

FastAPI забезпечив зручний інтерфейс для взаємодії між фронтом (Vue.js) та модулем обробки даних. Через створені API-ендпоінти на сервер передаються дані для обробки, а результати, такі як кластеризація чи виявлення аномалій, поверталися у вигляді зручного формату JSON.

5. Асинхронна обробка

Завдяки асинхронній природі FastAPI вдалося реалізувати паралельну обробку великих наборів даних. Це значно скоротило час обробки інформації, особливо для великих звітів із даними за кілька місяців.

1.6 Системи баз даних

У розробці вебдодатків для зберігання, обробки та аналізу великих обсягів даних важливо правильно вибрати тип бази даних, щоб забезпечити ефективність, надійність і масштабованість. Зберігання аналітичних даних про споживання електроенергії потребує вибору відповідної системи баз даних, яка здатна обробляти великі обсяги даних, забезпечувати швидкий доступ до інформації та підтримувати складні запити та аналітику.

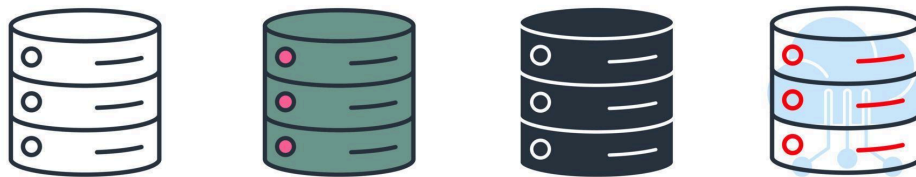


Рис.1.7. Різні типи баз даних

1.6.1 Вибір бази даних для зберігання даних електроенергії

Для зберігання даних електроенергії, зокрема інформації про споживання на рівні міст, будинків, місяців і років, і для забезпечення швидкої аналітики, найкращим вибором є реляційна база даних. Однією з найбільш популярних та ефективних реляційних баз даних є PostgreSQL. Вона дозволяє здійснювати складні запити для вибірки, агрегації, фільтрації даних і підсумовування, що є необхідним для аналітичних задач, таких як обчислення сум споживання по містах або виявлення аномалій в даних.

Переваги PostgreSQL для зберігання аналітичних даних:

1. **Скалованість:** PostgreSQL підтримує великі обсяги даних, зокрема, надає ефективні індекси для швидкого доступу до даних.
2. **Потужні можливості для обробки запитів:** PostgreSQL дозволяє використовувати складні SQL-запити для агрегації, фільтрації та обчислень, що робить її ідеальним інструментом для аналітики.
3. **Реляційні можливості:** Реляційний підхід дозволяє зберігати дані в добре структурованих таблицях, що полегшує їх подальшу обробку та управління.
4. **Можливість роботи з великими обсягами даних:** PostgreSQL ефективно працює з великими обсягами даних, що робить її чудовим вибором для аналітичних задач, де потрібна обробка та аналіз великої кількості записів, таких як споживання електроенергії.

Причини для вибору PostgreSQL:

- Надійність: PostgreSQL є стабільною та надійною базою даних, що дозволяє зберігати важливі аналітичні дані з гарантією їх цілісності.
- Підтримка транзакцій: PostgreSQL підтримує транзакції, що забезпечує цілісність даних при паралельній обробці великої кількості запитів.
- Гнучкість запитів: Завдяки підтримці складних SQL-запитів, PostgreSQL дозволяє ефективно працювати з даними для подальшого їх аналізу[9].



Рис.1.8. Логотип PostgreSQL

1.6.2 Реляційні та нереляційні бази даних

У контексті вибору між реляційними та нереляційними базами даних важливо розуміти ключові відмінності між ними та вибирати залежно від характеру даних і необхідних операцій.

1. Реляційні бази даних

Реляційні бази даних, такі як PostgreSQL, MySQL, Oracle, використовують структуру таблиць з чітко визначеними взаємозв'язками між даними. Вони підходять для зберігання структурованих даних, де є чітка схема і взаємозв'язки між таблицями. Реляційні бази даних підтримують SQL (Structured Query Language), що дозволяє виконувати складні запити для фільтрації, агрегації та обчислення даних.

Переваги реляційних баз даних:

- Чітка структура даних і підтримка складних зв'язків.
- Потужні можливості для обробки запитів і аналітики.

- Надійність і консистентність даних (особливо за допомогою транзакцій).
- 2. Для зберігання даних про споживання електроенергії реляційні бази даних є найкращим вибором, оскільки такі дані мають чітку структуру, зокрема, інформація про міста, роки та місяці потребує зберігання в таблицях з чіткими зв'язками між ними.
- 3. Нереляційні бази даних
Нереляційні бази даних, такі як MongoDB, Cassandra, Redis, використовуються для зберігання неструктурованих або слабо структурованих даних. Вони можуть бути корисними, коли дані не мають чіткої схеми або структура даних може змінюватися, що не є типовим для задач з електроенергією.
Переваги нереляційних баз даних:
 - Гнучкість структури даних.
 - Висока швидкість запису і масштабованість.
 - Підходять для обробки великих обсягів неструктурованих даних.
- 4. Однак для завдань з аналітики споживання електроенергії, де потрібно підтримувати чітку структуру даних, зберігати статистику за певні періоди часу та виконувати складні агрегації, нереляційні бази даних можуть не забезпечити ту ефективність і зручність, як реляційні.

Таким чином, реляційні бази даних (зокрема, PostgreSQL) є оптимальним вибором для зберігання аналітичних даних про споживання електроенергії, оскільки вони дозволяють ефективно працювати з структурованими даними, підтримують складні запити та надають надійність і цілісність даних.

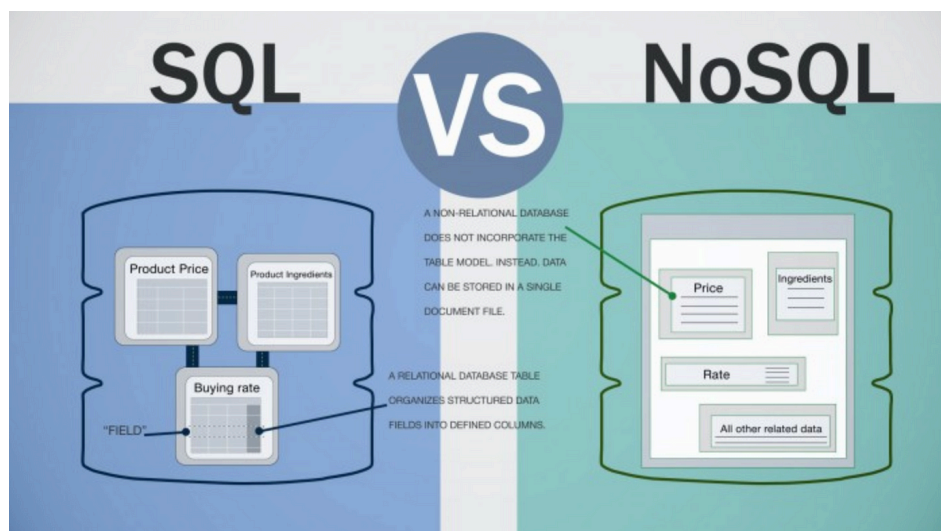


Рис.1.9. Різниця між реляційною та нереляційною базою даних

1.7. ORM для роботи з БД (SQLAlchemy, TypeORM)

У сучасних вебдодатках для зручного та ефективного взаємодії з базами даних часто використовуються ORM (Object-Relational Mapping) бібліотеки. ORM дозволяють розробникам працювати з базою даних, як з об'єктами програмного коду, не писати SQL-запити вручну, а замість цього використовувати методи та властивості об'єктів для виконання операцій із даними. Це значно спрощує процес розробки та дозволяє краще управляти взаємодією з базою даних, покращує читабельність коду і зменшує кількість помилок.

1.7.1 SQLAlchemy (Python)



Рис.1.10. Логотип SQLAlchemy

SQLAlchemy — це потужна ORM-бібліотека для мови програмування Python, яка дозволяє ефективно працювати з реляційними базами даних. Вона підтримує різні СУБД, зокрема PostgreSQL, MySQL, SQLite, Oracle та інші. SQLAlchemy дає можливість працювати з даними як з об'єктами Python, що спрощує інтеграцію бази даних з додатком.

Переваги SQLAlchemy:

1. Висока гнучкість: SQLAlchemy дозволяє комбінувати SQL-запити та ORM для виконання складних операцій із базою даних, що дає можливість оптимізувати продуктивність, коли це необхідно.
2. Підтримка різних СУБД: SQLAlchemy підтримує широкий спектр реляційних баз даних, включаючи PostgreSQL, MySQL, SQLite, Oracle та інші.
3. Автоматичне відображення об'єктів у таблиці: SQLAlchemy дозволяє легко визначати таблиці та їхні поля як класи та властивості Python-об'єктів. Це дозволяє розробникам працювати з даними без необхідності писати SQL-запити.
4. Можливість кастомізації запитів: За допомогою SQLAlchemy можна створювати складні запити, що включають обчислення, агрегацію, фільтрацію та інші SQL-операції.

1.7.2 TypeORM (Nest.js)

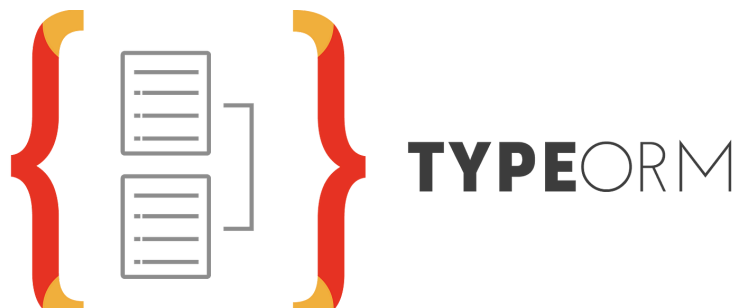


Рис.1.11. Логотип TypeORM

TypeORM — це ORM для TypeScript і JavaScript, який підтримує різні реляційні бази даних, включаючи PostgreSQL, MySQL, SQLite, MariaDB та інші. TypeORM інтегрується з Node.js-додатками, такими як Nest.js, і дозволяє працювати з базами даних за допомогою об'єктно-орієнтованого підходу. Це зручний інструмент для розробників, які працюють з TypeScript, оскільки він повністю підтримує статичну типізацію.

Переваги TypeORM:

1. Інтеграція з Nest.js: TypeORM легко інтегрується з Nest.js, що дозволяє створювати масштабовані серверні додатки з мінімальними зусиллями.
2. Підтримка TypeScript: TypeORM підтримує TypeScript, що дає можливість використовувати переваги типізації для кращої безпеки коду.
3. Автоматичне створення таблиць: TypeORM дозволяє автоматично генерувати таблиці на основі моделей, що значно спрощує створення і підтримку структури бази даних.
4. Підтримка складних запитів: TypeORM надає можливість створювати складні запити, обчислення та агрегації за допомогою ланцюгових методів, що робить код більш зручним для читання[10].

РОЗДІЛ 2. ФОРМУЛЮВАННЯ ЗАДАЧІ ТА СТВОРЕННЯ ПРОГРАМНОГО ПРОДУКТУ

2.1. Постановка задачі

2.1.1 Опис проблеми аналізу споживання електроенергії

Сучасне суспільство дедалі більше залежить від електроенергії, що вимагає оптимізації її споживання та управління. Постачальники електроенергії стикаються з низкою проблем, пов'язаних із аналізом великих обсягів даних про споживання. Основними викликами є складність обробки цих даних, відсутність зручних інструментів для їх візуалізації, а також неможливість оперативно виявляти відхилення чи формувати прогнози на основі наявної інформації.

Актуальність розробки подібного додатка зумовлена також вимогами державної політики України щодо енергозбереження та ефективного використання енергоресурсів. Відповідно до Закону України «Про місцеве самоврядування в Україні» [15] органи місцевого самоврядування відповідають за управління житлово-комунальним господарством, закладами охорони здоров'я, освіти та іншими об'єктами, що належать територіальним громадам. Одним із їхніх обов'язків є встановлення нормативів споживання комунальних послуг (стаття 30). Це передбачає необхідність збору, аналізу та моніторингу даних про енергоспоживання, що є важливим для прийняття ефективних управлінських рішень.

Крім того, Закон України «Про енергетичну ефективність будівель» зобов'язує впроваджувати автоматизовані системи моніторингу та управління інженерними системами з метою підвищення рівня енергетичної ефективності будівель. На підтримку цього законодавства було затверджено ДСТУ ISO 50001:2014 «Енергозбереження. Системи енергетичного менеджменту. Вимоги та настанова щодо використання», який визначає вимоги до систем енергоменеджменту для зниження енергоспоживання та покращення енергоефективності [16].

На ринку вже існують програми для аналізу споживання енергоресурсів, такі як uMuni та Енергоплан. Програма uMuni орієнтована на облік та аналітику витрат енергії в масштабах муніципалітетів, а Енергоплан надає користувачам звіти про споживання та дозволяє

планувати енергетичні витрати. Проте ці рішення часто є дорогавартісними і значною мірою покладаються на аналітичну допомогу фахівців, що знижує рівень автоматизації процесів. Для багатьох місцевих громад та невеликих організацій їхня вартість є значною перешкодою, а необхідність залучення людського ресурсу для роботи з цими системами обмежує їхню ефективність.

Таким чином, розробка вебдодатка, який поєднує можливості автоматизованого аналізу споживання електроенергії, є нагальною потребою. Він дозволить вирішити актуальні завдання енергоменеджменту, знизити витрати на обслуговування та надасть доступ до важливої інформації навіть для невеликих користувачів.

Етапи обробки та аналізу споживання електроенергії:

1. Збір даних з Excel файлів

Користувач надає Excel файл, що містить інформацію про споживання електроенергії по будинках за місяць. Цей файл може містити дані, такі як:

- Назва або ідентифікатор будинку.
- Дата або період споживання (наприклад, день, тиждень, місяць).
- Обсяг спожитої електроенергії (в кВт·год).
- Інші показники, такі як тип споживання (основне, резервне), тарифні категорії і т. д.

2. Після завантаження, файл проходить етап попередньої обробки, під час якого перевіряється його формат, коректність даних, та відсутність помилок.

3. Обробка даних

Після зчитування даних з Excel файлів, система виконує такі кроки:

- Перевірка та очищення даних: усуваються помилки, що можуть бути викликані неправильним форматом або пропущеними значеннями.
- Агрегування даних: для зручності аналізу, дані за певні періоди (наприклад, тиждень або місяць) агрегуються та нормалізуються, щоб порівняння між різними будинками було коректним.

- Обчислення додаткових метрик: таких як середнє споживання за період, загальні витрати за всіма будинками, чи навіть порівняння споживання з попередніми періодами для виявлення трендів.

4. Аналіз даних

Після обробки даних система застосовує різноманітні методи для аналізу та виявлення аномалій:

- Виявлення аномалій: за допомогою статистичних методів, таких як аналіз відхилень від середнього значення або пошук незвичних піків споживання, система може виявити потенційно надмірне або неефективне використання електроенергії.
- Прогнозування споживання: на основі історичних даних, система може прогнозувати подальші потреби в енергетичних ресурсах для кожного будинку, допомагаючи постачальникам точніше планувати споживання енергії на наступний період.
- Кластеризація споживачів: будинки можуть бути згруповані за схожими патернами споживання, що дозволяє постачальникам енергії краще розуміти різні категорії споживачів та їхні потреби.

5. Візуалізація результатів

Для зручності користувачів, усі результати аналізу представлені у вигляді інтерактивних графіків, діаграм та таблиць, що дозволяють:

- Легко побачити тренди споживання.
- Оцінити ефективність використання електроенергії по кожному будинку.
- Виявити аномалії або піки споживання.

6. Наприклад, користувач може побачити графік, що показує споживання кожного будинку в порівнянні з іншими.

Проблеми, що виникають при обробці та аналізі даних з Excel:

1. Нестабільність або помилки в даних

Excel файли можуть містити помилки, як-от неповні або некоректні записи (наприклад, відсутність значень у деяких клітинках), що ускладнює автоматизовану обробку даних. Ці помилки потрібно виявляти і виправляти перед запуском аналітичних процесів.

2. Великі обсяги даних

Якщо користувач надає файли з великими обсягами даних, наприклад, для багатьох будинків або для кількох місяців, система повинна мати можливість швидко обробляти такі файли, що вимагає оптимізації алгоритмів і використання ефективних методів зберігання даних.

3. Невизначеність даних

Деякі дані можуть бути неповними або неточними через людський фактор або технічні проблеми при зборі. Такі дані необхідно коректно обробляти, щоб вони не впливали на точність аналізу.

4. Аномалії в даних

Збирання даних вручну або використання неавтоматизованих систем може призводити до появи аномальних значень або виходу за межі звичних трендів, що може вказувати на помилки у зборі даних або на технічні проблеми з обладнанням.

2.1.2 Опис необхідних функцій вебдодатку

Вебдодаток для аналізу споживання електроенергії, розроблений для постачальників електричної енергії, має на меті забезпечити ефективний збір, обробку, аналіз та візуалізацію даних щодо споживання електроенергії за місяць. Ось перелік основних функцій, які має підтримувати вебдодаток:

1. Завантаження Excel файлів

- Користувач повинен мати можливість завантажити Excel файл, що містить дані про споживання електроенергії по будинках.
- Підтримка різних форматів Excel (XLS, XLSX).
- Перевірка файлів на коректність і відповідність вимогам (наприклад, наявність необхідних стовпців і правильний формат даних).

2. Обробка даних

- Автоматичне зчитування даних з Excel файлів та їх валідація.
- Очищення і нормалізація даних для подальшого аналізу.

- Обчислення основних показників, таких як середнє споживання, максимальні/мінімальні значення, агреговані дані за певний період.
 - Ідентифікація та виправлення помилок, таких як відсутність або некоректні значення [12].
3. Аналіз споживання
- Застосування методів для виявлення аномалій у споживанні енергії.
 - Кластеризація будинків за схожими патернами споживання.
 - Прогнозування споживання на основі історичних даних для кожного будинку.
 - Розрахунок витрат енергії на одиницю площі чи кількість мешканців (за потреби).
4. Візуалізація результатів
- Побудова графіків, діаграм і таблиць для зручного представлення результатів аналізу.
 - Візуалізація споживання енергії кожним будинком або групою будинків.
 - Побудова інтерактивних панелей управління для швидкого доступу до звітів.
 - Інтерактивні графіки для порівняння показників споживання по будинках за різні періоди.
5. Звітність
- Можливість генерації та експорту звітів у форматах PDF або Excel, що містять аналіз споживання енергії та рекомендації щодо оптимізації.
 - Детальні звіти для кожного будинку, включаючи зведені та порівняльні показники.
 - Налаштування параметрів звітів (наприклад, вибір періоду аналізу).
6. Безпека
- Реалізація аутентифікації та авторизації для доступу до додатка.
 - Захист від несанкціонованого доступу через використання сесій та токенів.

2.2 Проектування архітектури вебдодатку

Архітектура вебдодатку повинна бути спроектована так, щоб забезпечити зручність у використанні, масштабованість і високу ефективність при обробці великих обсягів даних, що надходять з Excel файлів.

2.2.1 Вибір архітектури (MVC, REST)

Для цього вебдодатку найкраще підійде архітектура REST з використанням Model-View-Controller (MVC) для організації коду. Такий підхід дозволяє чітко розподілити функціональність додатку і забезпечити легкість в обслуговуванні та масштабуванні.

Детально:

1. Model (Модель)

Модель відповідає за представлення даних і бізнес-логіку. Вона здійснює:

- Взаємодію з базою даних або файлами (Excel).
- Виконання обчислень, таких як аналіз споживання, виявлення аномалій і прогнози.
- Зберігання та управління станом даних.

2. View (Подання)

Подання є інтерфейсом користувача, який відповідає за відображення даних, отриманих від моделі, у зручному для користувача форматі:

- Графічний інтерфейс користувача (GUI) для завантаження файлів, перегляду результатів аналізу, побудови графіків і звітів.
- Візуалізація результатів аналізу (графіки, діаграми).
- Використання таких технологій, як HTML, CSS, JavaScript (наприклад, React або Vue.js), для забезпечення динамічного інтерфейсу.

3. Controller (Контролер)

Контролер обробляє запити від користувача та взаємодіє з моделями та поданнями:

- Обробка запитів для завантаження та обробки Excel файлів.
- Виконання запитів для отримання та аналізу даних.
- Керування логікою обробки даних (наприклад, очищення файлів, агрегування даних, запуск методів аналізу).

4. RESTful архітектура

Архітектура REST (Representational State Transfer) дозволяє додатку бути незалежним від платформи та забезпечує простоту комунікації через HTTP. Ось основні компоненти:

- API для завантаження файлів — користувач може завантажити Excel файл через HTTP POST запит.
- API для обробки та аналізу даних — після завантаження файл обробляється на сервері через запити до REST API.
- API для візуалізації результатів — забезпечує доступ до отриманих даних у форматі JSON або інших форматах для відображення в інтерфейсі.

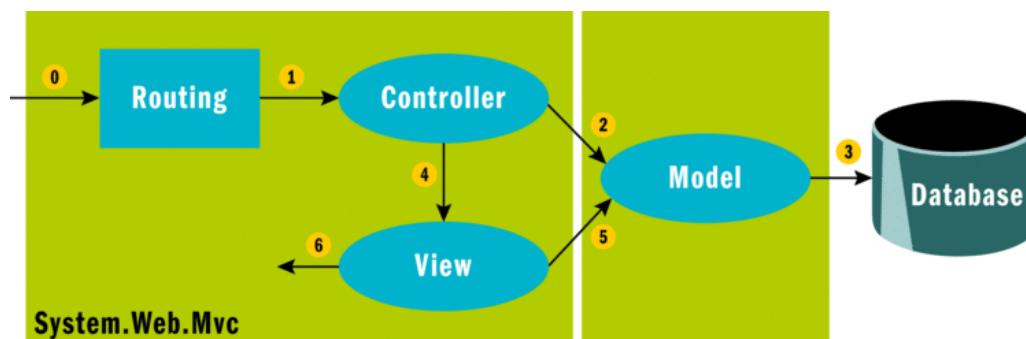


Рис.2.1. Приклад роботи MVC

Переваги цієї архітектури:

- Масштабованість: REST API дозволяє додатку легко інтегруватися з іншими системами та бути адаптованим до нових вимог.
- Модульність: Розділення на модель, подання та контролер дозволяє легко змінювати або масштабувати окремі частини додатку.
- Легкість в обслуговуванні: Логіка додатку чітко розділена, що полегшує підтримку і тестування коду.

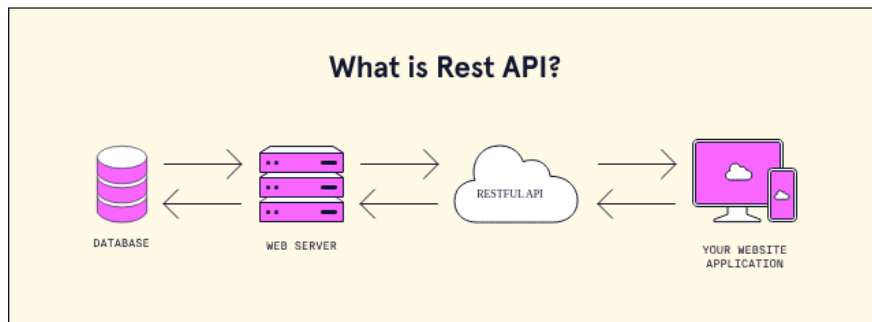


Рис.2.2. Приклад роботи Rest API

Таким чином, використання архітектур MVC і REST забезпечить зручну та ефективну організацію коду, що відповідає потребам аналізу споживання електроенергії та дозволяє користувачам легко взаємодіяти з системою через інтерфейс.

2.2.2 Опис основних компонентів

Для розробки вебдодатку, який здійснює аналіз споживання електроенергії, використовуються три основні компоненти: фронтенд, бекенд та обробка даних, кожен з яких має свою функціональність та взаємодіє з іншими частинами системи. Далі описані основні компоненти та технології, що використовуються в кожному з них.

1. Фронтенд (Vue.js)

Фронтенд є частиною вебдодатку, з якою користувач взаємодіє безпосередньо. Основною задачею фронтенду є надання зручного та інтуїтивно зрозумілого інтерфейсу для користувача. Для цієї мети використовується Vue.js — прогресивний JavaScript фреймворк, який дозволяє створювати швидкі та динамічні інтерфейси користувача.

- Основні функції фронтенду:
 - Завантаження файлів: Користувач може завантажити Excel файл через інтуїтивно зрозумілу форму завантаження.
 - Візуалізація даних: Після обробки даних на сервері, фронтенд відповідає за побудову графіків, діаграм та таблиць для представлення результатів аналізу споживання.

- Інтерактивні елементи: Забезпечує інтерактивність інтерфейсу, такі як фільтри для вибору періоду, сортування та порівняння даних.
- Комунікація з бекендом: Фронтенд здійснює HTTP-запити до серверу для отримання та відправлення даних, таких як завантаження файлів, отримання результатів аналізу тощо.
- Переваги Vue.js:
 - Легкість інтеграції: Vue.js інтегрується з іншими бібліотеками та фреймворками, що дозволяє використовувати його в різних контекстах.
 - Швидка розробка: Vue.js дозволяє швидко створювати динамічні та реактивні інтерфейси з мінімумом коду.
 - Компонентна архітектура: Завдяки компонентній модульній структурі, Vue.js дозволяє розділяти інтерфейс на невеликі частини, що полегшує супровід та масштабування [14].

2. Бекенд (Nest.js)

Бекенд — це серверна частина додатку, яка відповідає за обробку запитів від користувачів, управління даними, авторизацію, а також інтеграцію з іншими системами. Для реалізації бекенду використовується Nest.js — фреймворк для Node.js, який базується на TypeScript і використовує архітектурні патерни, такі як MVC і Dependency Injection.

- Основні функції бекенду:
 - Авторизація та аутентифікація: Бекенд відповідає за перевірку користувачів та надання доступу до різних частин додатку, з використанням токенів (JWT) для безпечної авторизації.
 - API для взаємодії з фронтендом: Бекенд обробляє запити від фронтенду, наприклад, на завантаження файлів, отримання результатів аналізу, генерацію звітів.
 - Інтеграція з обробкою даних: Бекенд може отримувати дані від FastAPI (Python) для подальшого оброблення та відправлення результатів назад на фронтенд.
 - База даних: Бекенд також відповідає за взаємодію з базою даних для збереження результатів аналізу, користувацьких налаштувань та історії запитів.
- Переваги Nest.js:

- Модульність: Nest.js має зручну модульну структуру, що дозволяє легко організувати код і поділити додаток на логічні частини.
- Підтримка TypeScript: Використання TypeScript забезпечує сильну типізацію та автоматичні перевірки під час розробки, що знижує ймовірність помилок.
- Гнучкість: Nest.js підтримує багато популярних бібліотек для роботи з базами даних, системами аутентифікації та іншими компонентами, що дозволяє легко інтегрувати додаткові функціональності.

3. Обробка даних (Python, FastAPI)

Обробка даних з Excel файлів є критично важливою частиною цього вебдодатку, оскільки користувачі завантажують дані саме у цьому форматі. Для цього використовується FastAPI (Python), який дозволяє ефективно обробляти запити та виконувати складні операції з даними.

- Основні функції обробки даних:
 - Завантаження та обробка Excel файлів: Обробка файлів Excel за допомогою бібліотек, таких як pandas, openpyxl або xlrd, для зчитування даних про споживання електроенергії по будинках[1,2].
 - Очищення та нормалізація даних: Виявлення помилок в даних (відсутні значення, некоректний формат) та їх виправлення.
 - Аналіз даних: Виконання обчислень, таких як визначення середнього споживання, максимальних та мінімальних значень, а також виявлення аномалій за допомогою статистичних методів.
 - Інтерфейс для бекенду: FastAPI надає REST API для бекенду (Nest.js), щоб надсилати результат обробки даних назад на сервер або безпосередньо в базу даних [3].
- Переваги FastAPI:
 - Висока продуктивність: FastAPI є дуже швидким завдяки асинхронній обробці запитів, що особливо важливо при обробці великих Excel файлів.

- Легкість у роботі з даними: Завдяки бібліотекам Python для обробки даних, FastAPI ефективно працює з великими обсягами даних з Excel файлів.

4. База даних

Для зберігання історичних даних, результатів аналізу та користувацьких налаштувань, використовується реляційна база даних, така як PostgreSQL.

- Основні функції бази даних:
 - Зберігання результатів аналізу: База даних зберігає зібрані та оброблені дані про споживання електроенергії для подальшого аналізу.
 - Запити до бази даних: Бекенд взаємодіє з базою даних для отримання необхідних даних або збереження нових результатів.
 - Зберігання користувацьких даних: База даних зберігає інформацію про користувачів, їх налаштування, історію завантажених файлів тощо.
- Переваги PostgreSQL:
 - Підтримка складних запитів: PostgreSQL підтримує складні SQL запити для аналізу великих обсягів даних, що є важливим для даного додатку.
 - Масштабованість: PostgreSQL може ефективно працювати з великими обсягами даних, що дозволяє масштабувати систему в разі необхідності.

2.3 Створення програмного продукту

2.3.1 Розробка фронтенду (Vue.js)

Для реалізації користувацького інтерфейсу вебдодатку використано Vue.js, що дозволяє створювати динамічні та інтуїтивно зрозумілі компоненти з використанням готових модулів. Ось як виглядає розробка фронтенду по етапах:

1. Головна сторінка

- Карта України: На головній сторінці відображається інтерактивна карта, на якій позначені міста, для яких проводилася аналітика. Це дозволяє користувачам швидко орієнтуватися і знаходити потрібну інформацію.

```
initializeMap() {  
  let map = am4core.create(this.$refs.map, am4maps.MapChart);  
  map.geodata = am4geodata_ukraineHigh;  
  map.projection = new am4maps.projections.Mercator();  
  
  map.zoomControl = new am4maps.ZoomControl();  
  map.homeZoomLevel = 0.9;  
  
  let polygonSeries = map.series.push(new am4maps.MapPolygonSeries());  
  polygonSeries.useGeodata = true;  
  
  let polygonTemplate = polygonSeries.mapPolygons.template;  
  polygonTemplate.tooltipText = "{name}";  
  polygonTemplate.fill = am4core.color( value: "#474D84");  
  
  this.map = map;  
}
```

Рис.2.3. Приклад виклику генерації мапи

- Віджети:
 - Віджет для відображення експорту електроенергії.
 - Віджет для відображення заробітку з експорту електроенергії.
 - Графік для перегляду експорту електроенергії в різні міста за певний період (наприклад, за рік).
 - Використовуються компоненти для інтерактивного відображення даних, такі як Chart.js або ECharts[11].

```

<Widget title="<h6> Експорт <span class='fw-semi-bold'>по містам</span></h6>" close customHeader> You, 19.11.
<div class="widget-body">
  <h3>{{ (consumptions.totalconsumptionsum / 1000000).toFixed(3) }} млн грн</h3>
  <p class="fs-mini text-muted mb mt-sm">
    <span class="text-align-right">Експорт по містам</span>
  </p>
</div>
<div class="widget-bottom-overflow">
  <table class="table table-striped table-sm market-stats">
    <thead class="no-bd">
      <tr>
        <th></th>
        <th>Micro</th>
        <th class="text-align-right">Експорт всього</th>
        <th class="text-align-right">Експорт в середньому</th>
      </tr>
    </thead>
    <tbody>
      <tr v-for="(city, index) in cityConsumptions" :key="index">
        <td>
        </td>
        <td>{{ city.data_location }}</td>
        <td class="text-align-right fw-semi-bold">{{ (city.totalconsumptionsum / 1000000).toFixed(2) }} MW</td>
        <td class="text-align-right fw-semi-bold">{{ (city.totalconsumptionavg / 1000000).toFixed(2) }} MW</td>
      </tr>
    </tbody>
  </table>
</div>
</Widget>

```

Рис.2.4. Приклад використання віджетів

2. Сторінка з усіма проведеними аналізами

- Ця сторінка містить перелік всіх завантажених Excel файлів з результатами аналізу. Для кожного файлу вказуються основні метадані, такі як дата завантаження, кількість оброблених даних, статус обробки.

```

<tr v-for="row in tableFiles" :key="row.id" @click="goToDetails(row.id)" class="clickable-row">
  <td>{{ row.id }}</td>
  <td>{{ row.name }}</td>
  <td>{{ row.location }}</td>
  <td>{{ row.description }}</td>
  <td>{{ parseDate(row.investigationMonth) }}</td>
  <td>{{ parseDate(row.uploadDate) }}</td>
</tr>

```

Рис.2.5. Приклад реалізації таблиці

3. Сторінка створення нового аналізу

- На цій сторінці користувач може завантажити новий Excel файл, який містить дані про споживання електроенергії. Також є можливість додавання додаткових даних, таких як міста, регіони, тип споживачів, ціна за одиницю товару.

Для цього використовуються Vue компоненти для вибору файлів та додаткових даних, а також інтерактивні форми для їх заповнення.

```
cityOptions: [
  { value: 'Chernivtsi', text: 'Чернівці' },
  { value: 'Kyiv', text: 'Київ' },
  { value: 'Odesa', text: 'Одеса' },
  { value: 'Khmelnytskyi', text: 'Хмельницький' }
],
categoryOptions: [
  { value: 'data', text: 'Дані' },
  { value: 'report', text: 'Звіт' },
  { value: 'analytics', text: 'Аналіз' },
  { value: 'other', text: 'Інше' }
],
processingOptions: [
  { value: 'peoples', text: 'Побутові споживачі' },
  { value: 'communal', text: 'Комунальне підприємство' },
],
```

Рис.2.6. Приклад вибору додаткових параметрів

4. Сторінка конкретного аналізу файлу

- Це детальна сторінка для перегляду інформації про конкретний файл аналізу.
- Основні функції:
 - Відображення детальної інформації про споживання електроенергії в різних місяцях.
 - Створення графіка споживання для всіх вулиць, що дозволяє аналізувати дані по кожній вулиці.
 - Показ топу вулиць за рівнем споживання електроенергії.
 - Відображення даних про кількість організацій, що беруть участь в аналізі.
 - Виведення аномальних даних, наприклад, високе або низьке споживання, порівняно з середнім рівнем.

```
<Widget
  title="<h5>Energy Consumption <span class='fw-semi-bold'>Statistics</span></h5>"
  close collapse customHeader
>
<p class="widget-text">Максимальне споживання: {{ responseData.maxConsumption }} кВт·год.</p>
<p class="widget-text">Середнє споживання: {{ responseData.meanConsumption }} кВт·год.</p>
<p class="widget-text">Мінімальне споживання: {{ responseData.minConsumption }} кВт·год.</p>
<p class="widget-text">Медіана споживання: {{ responseData.medianConsumption }} кВт·год.</p>
<p class="widget-text">Загальне споживання: {{ responseData.totalConsumption }} кВт·год.</p>
</Widget>
```

Рис.2.7. Приклад виводу інформації про аналіз

- Для відображення складної інформації використовуються таблиці, графіки та діаграми.

```
<Widget
  title="<h5>Consumption <span class='fw-semi-bold'>Beeswarm Chart</span></h5>"
  close collapse customHeader
>
<!-- Контейнер для Beeswarm графіка --> | You, 19.11.24, 18:15 • Uncommitted changes
<div id="beeswarmChart" style="..."></div>
</Widget>
```

Рис.2.8. Приклад відображення графіку

```
renderBeeswarmChart(data) | You, 19.11.24, 18:15 • Uncommitted changes
this.chart = am4core.create( htmlElement: "beeswarmChart", am4charts.XYChart);
this.chart.data = data;

// Налаштування осі X (sum_value)
const valueAxisX = this.chart.xAxes.push(new am4charts.ValueAxis());
valueAxisX.title.text = "Consumption (kW·h)";
valueAxisX.min = 0; // Мінімум осі X
valueAxisX.max = 200000; // Максимум осі X
valueAxisX.strictMinMax = true;
valueAxisX.renderer.baseGrid.disabled = true;

valueAxisX.renderer.labels.template.adapter.add( key: "text", callback: function(text, target) {
  const value = target.dataItem.value;
  if (value === 15000) {
    return "15,000"; // Для центру відображаємо 15,000
  }
  return text;
})
```

Рис.2.9. Приклад створення графіка

5. Реєстрація та авторизація

- Створення форм для реєстрації та авторизації користувачів, що дозволяє захистити доступ до даних.
- Для цього використовуються стандартні формуляри з валідацією введених даних, перевірка email та паролю.
- Реалізація JWT авторизації через бекенд на Nest.js, що дозволяє користувачам мати безпечний доступ до особистих даних та аналітики.

```

try {
  const response = await axios.post('http://localhost:3003/login', {
    email,
    password,
  });

  if (response.data && response.data.access_token) {
    console.log(response)
    // Збереження JWT в локальне сховище
    localStorage.setItem('jwt', response.data.access_token);
    this.$router.push('/app/dashboard');
  } else {

```

Рис.2.10. Приклад аутентифікації

6. Взаємодія з бекендом (Nest.js)

- API-запити: Всі дані, що відображаються на фронтенді, отримуються через API-ендпоінти, розроблені в бекенд частині на Nest.js.
 - Використання Axios на фронтенді для отримання та відправлення запитів до серверу.
 - Для отримання даних про аналізи, користувачів та іншу інформацію, відправляються запити до Nest.js API.
- Авторизація та захист даних: Всі запити на сторінки, що вимагають авторизації, перевіряються через JWT токени, що зберігаються в браузері користувача.

2.3.2 Розробка бекенду (Nest.js)

1. Ініціалізація проекту

- Налаштував структуру проекту, створив модулі, контролери та сервіси для управління різними функціональними блоками.
- Інтегрував базову конфігурацію Nest.js для підтримки масштабованості та розширюваності.

2. Підключення до бази даних

- Використав ORM TypeORM (або інший вибраний інструмент) для взаємодії з базою даних.
- Налаштував з'єднання з базою даних, визначив і створив сутності (entities), що відповідають структурі таблиць у БД.

- Реалізував механізми міграцій для контролю змін у структурі бази даних.

```
TypeOrmModule.forRoot({
  type: 'postgres',
  host: 'localhost',
  port: 5432,
  username: 'postgres',
  password: 'qwerty',
  database: 'wattscope',
  entities: [User, AnalyzedDataEntity, PredictedEntity],
  synchronize: true,
}),
```

Рис.2.11. Підключення до бази даних та контроль міграцій

3. Реалізація JWT-авторизації

- Налаштував модуль для роботи з JWT (JSON Web Token) для забезпечення безпечної аутентифікації та авторизації користувачів.
- Створив ендпоінти для реєстрації та входу користувачів. При реєстрації нові користувачі зберігаються у БД, а при вході генерується токен.
- Реалізував middleware та guard для перевірки валідності токенів на кожному захищеному ендпоінті.

```
canActivate( no usages new *
  context: ExecutionContext,
): boolean | Promise<boolean> | Observable<boolean> {
  const request :any = context.switchToHttp().getRequest();
  const token :any = request.headers.authorization?.split(' ')[1];

  if (!token) {
    throw new UnauthorizedException('Token is missing');| You, 18.11.
  }

  try {
    const decoded :any = this.jwtService.verify(token);
    request.user = decoded;
    return true;
  } catch (error) {
    throw new UnauthorizedException('Invalid token');
  }
}
```

Рис.2.12. Реалізація JWT токенів

4. Розробка API

- Створив необхідні ендпоінти для взаємодії з фронтом.
- Реалізував функції для отримання даних з бази, їх обробки (форматування, фільтрації, обчислень) та повернення у відповідному форматі.
- Додатково забезпечив валідацію даних на рівні DTO (Data Transfer Object), щоб гарантувати коректність вхідних даних.

```
@UseGuards(JwtAuthGuard) no usages
@Get('users/all-orders-by-user-id')
Codeium: Refactor | Explain | Docstring | x
async getAllOrdersByUserId(@Request() req :any ): Promise<any> {
  const userEmail :any = req.user.email;
  const user :User = await this.userService.findOneByEmail(userEmail);
  const result :any = await this.appService.getAllOrdersByUserId(user.id);
  return result;
}
```

Рис.2.13. Приклад роутера захищеного перевіркою jwt

5. Захист API та управління помилками

- Забезпечив обробку винятків та створив глобальний фільтр для управління помилками, який повертає стандартизовані відповіді при різних типах помилок.

6. Тестування та оптимізація

- Провів модульне тестування ключових модулів та сервісів для забезпечення стабільності роботи системи.
- Виконав інтеграційне тестування ендпоінтів для перевірки коректності їх роботи з фронтом.
- Оптимізував запити до бази даних для підвищення продуктивності.

2.3.3 Розробка сервісу аналітики даних (FastAPI)

Сервіс обробки даних на Python був створений для аналітики даних електроспоживання. Основні його завдання: обробка отриманих з фронтенду даних, їхній аналіз, кластеризація та прогнозування майбутніх показників споживання. Сервіс інтегрований із бекендом і виконує складну аналітику, що включає статистичну обробку, кластерний аналіз і прогнозування.

Етапи роботи сервісу

1. Отримання даних

- Джерело даних: Сервіс отримує дані у форматі Excel від користувачів через фронтед.
- Структура даних: Таблиця містить наступні поля:
 - Ідентифікатор запису
 - Дата та час
 - Споживання
 - Організація
 - Ідентифікаційний номер організації
 - Постачальник
 - Ідентифікаційний номер постачальника
 - Ідентифікаційний номер лічильника
 - Тип надання послуг
 - Кількість
 - Одиниця вимірювання
 - Вулиця
 - Номер будинку

A	B	C	D	E	F	G	H	I	J	K	L	M
id	date	organization	organization	providerName	providerID	meterNumber	type	quantity	unitName	addressStreet	addressLocatorNumber	
1	2024-08-29	ЕнергоПотік	45897896	ПАТ «Черн	1453785	ЛБ-432234/9	Енергосист	481	kWh	10-річчя Нез	5	
2	2024-08-30	СвітлоФорс	15678942	ПАТ «Черн	1453785	ЛБ-100100/2	Енергосист	328	kWh	10-річчя Нез	9	

Рис.2.14. Структура даних ексель файлу

Дані передаються до сервісу у вигляді Excel-файлу, який зберігається у тимчасовій файловій системі для подальшої обробки.

2. Обробка та очищення даних

- Очищення даних:
 - Видалення дублікатів.
 - Заповнення пропущених значень:
 - Пропущені значення споживання заповнюються за допомогою методів лінійної інтерполяції.
 - Видалення спостережень із некоректними даними (наприклад, негативне споживання).
- Форматування дат:
 - Конвертація стовпців Дата та Час у стандартний формат для подальшого аналізу[12].

3. Обчислення статистичних показників

Сервіс виконує розрахунки базових статистичних метрик для кожного міста, вулиці або будинку, наприклад:

- Максимальне споживання.
- Мінімальне споживання.
- Середнє споживання.
- Медіанне споживання.

```
# 1. Обчислюємо статистики для споживання електроенергії (quantity)
max_consumption = df_clean['quantity'].max()
mean_consumption = df_clean['quantity'].mean()
min_consumption = df_clean['quantity'].min()
median_consumption = df_clean['quantity'].median()
total_consumption = df_clean['quantity'].sum()
```

Рис.2.15. Приклад знаходження статистичних метрик

4. Кластерний аналіз

Для аналізу споживання було реалізовано кілька підходів кластеризації:

- К-Means кластеризація[5]:
 - Використовується для розбиття даних на групи споживачів залежно від рівня споживання.
 - Наприклад, кластеризація споживачів у три групи:
 - Мале споживання.
 - Середнє споживання.

■ Велике споживання.

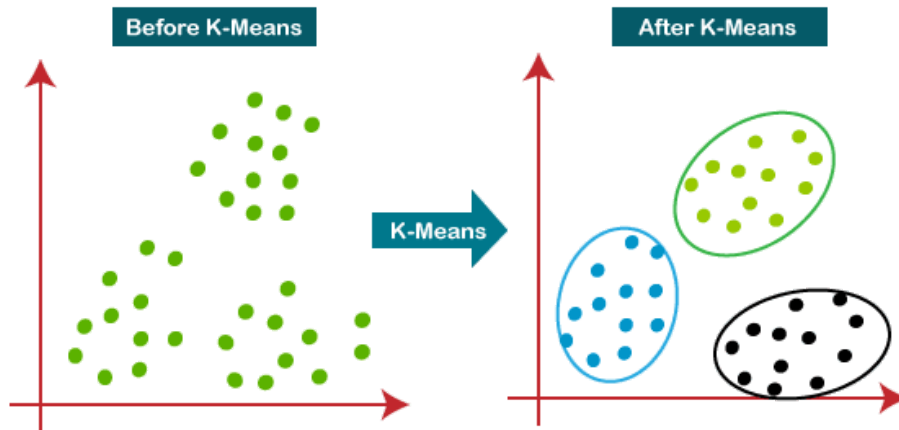


Рис. 2.16. Результат роботи алгоритму k-means

- DBSCAN (Density-Based Spatial Clustering of Applications with Noise):
 - Використовується для виявлення аномалій у даних (наприклад, різкі піки або провали у споживанні)[5].
 - Аномальні споживачі визначаються як віддалені точки за межами основних кластерів.

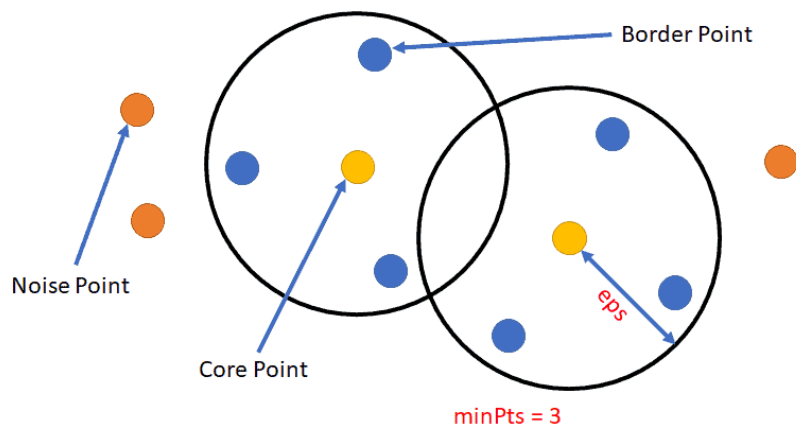


Рис.2.17. Принцип роботи алгоритму DBSCAN

- Розбиття за географічними критеріями:
 - Додаткова кластеризація проводиться за вулицями. Це дозволяє визначати рівні споживання по окремих районах або вулицях.

5. Прогнозування споживання (модель Prophet)

Для прогнозування електроспоживання використовується модель Prophet[7]:

- Вхідні дані:
 - Дані споживання за останні 6 місяців по конкретному місту.
- Вихідні дані:
 - Прогноз споживання на наступний місяць , для всього міста та конкретних вулиць.
- Особливості моделі:
 - Prophet автоматично враховує сезонність, тренди та свята, що можуть впливати на споживання електроенергії[8].
 - Вихідні результати включають межі довіри для кожного прогнозу, що дозволяє оцінити ризик відхилення.

```
# Ініціалізація моделі Prophet
model = Prophet()
model.fit(df)

# Прогноз на наступний місяць
future = model.make_future_dataframe(periods=1, freq="M")
forecast = model.predict(future)
```

Рис.2.18. Приклад ініціалізації моделі Prophet

7. Результати та інтеграція з фронтом

- Сформовані результати аналізу та прогнозів передаються у базу даних, де вони доступні для отримання або подальшої обробки.
- Фронтенд отримує готові візуалізації для зручності користувачів за допомогою Nest.js.

РОЗДІЛ 3. ПРОГРАМНИЙ ПРОДУКТ

3.1. Приклад роботи програмного продукту

Розроблений програмний продукт забезпечує інтуїтивно зрозумілий і зручний процес реєстрації та завантаження даних. Цей етап є першим кроком до використання всіх аналітичних можливостей системи, тому його функціонал максимально спрощений для кінцевого користувача, при цьому зберігаючи високий рівень надійності та безпеки.

Реєстрація користувача

Щоб отримати доступ до системи, користувач має пройти процес реєстрації. Це необхідний крок для створення персоналізованого облікового запису, де будуть зберігатися всі проекти, результати аналізу та інші дані, доступні лише цьому користувачу.

1. Форма реєстрації:

- **Ім'я користувача:** дозволяє створити унікальний обліковий запис.
 - **Електронна пошта:** використовується для зв'язку з користувачем, а також для відновлення доступу до облікового запису.
 - **Пароль:** система вимагає надійний пароль, що включає щонайменше одну велику літеру, одну цифру та спеціальний символ.
 - **Підтвердження пароля:** запобігає помилкам при введенні пароля.
2. Після заповнення форми користувач отримує листа на вказану електронну пошту для підтвердження реєстрації. Цей крок забезпечує додатковий рівень безпеки, унеможливорюючи створення фейкових облікових записів.
 3. **Авторизація:** Після реєстрації користувач може увійти до системи, ввівши свої облікові дані. Система зберігає сесію, щоб уникнути необхідності повторного введення пароля при кожному вході.
 4. **Відновлення доступу:** У разі втрати пароля користувач може скористатися функцією відновлення доступу, де після введення своєї електронної пошти отримує інструкції для його скидання.

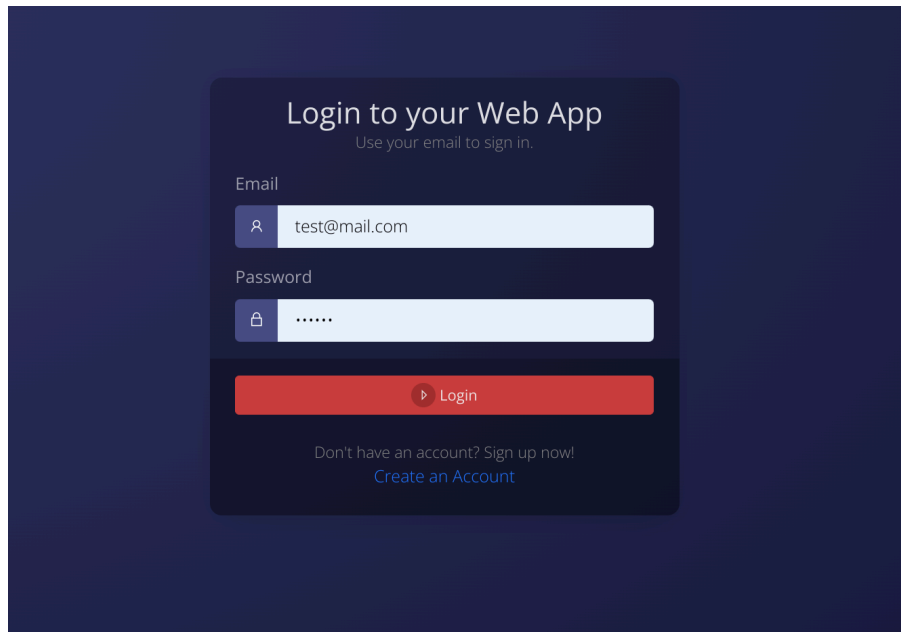


Рис.3.1. Вікно авторизації користувача

Завантаження даних

Завантаження даних є основним етапом для початку аналітичної роботи. Процес розроблений таким чином, щоб користувач міг легко додавати свої файли, навіть не маючи технічних навичок.

1. **Форма завантаження:** Після входу в систему користувач потрапляє на сторінку завантаження даних. Форма містить кілька обов'язкових полів:
 - **Назва файлу:** дає змогу ідентифікувати конкретний набір даних. Це важливо для подальшої роботи, коли користувачеві потрібно знайти результати аналізу саме цього файлу серед інших.
 - **Ціна за кВт·год:** користувач вводить актуальну вартість електроенергії. Це поле важливе для фінансових розрахунків, таких як визначення загальної вартості спожитої електроенергії для кожного споживача.
 - **Місяць збору даних:** дозволяє прив'язати набір даних до конкретного періоду. Це корисно для аналізу трендів та порівняння даних між різними місяцями.
 - **Місто:** користувач обирає місто з випадального списку. Це забезпечує можливість сегментувати дані за географічною ознакою.

- **Коментар:** текстове поле для внесення будь-яких додаткових зауважень або опису специфіки даних.
- **Тип експорту:** користувач обирає між побутовими споживачами та комунальними підприємствами. Це дозволяє системі адаптувати розрахунки та аналітику відповідно до обраного сегмента.
- **Завантаження Excel-файлу:** користувач завантажує файл із даними, який відповідає визначеній структурі описаній на сайті. Структура включає такі колонки, як "Дата", "Споживання" та "Лічильник" та інші.

- Перевірка та обробка файлу:** Після завантаження система автоматично перевіряє файл на відповідність структурі. У разі виявлення помилок (наприклад, відсутніх обов'язкових полів або некоректного формату даних) користувач отримує відповідне сповіщення.
- Збереження даних:** Дані успішно завантаженого файлу зберігаються у базі даних для подальшої обробки. Кожен файл отримує унікальний ідентифікатор, який дозволяє швидко знаходити його в системі.

Рис.3.2. Завантаження файлу з даними

Обробка даних

Обробка даних є центральною частиною роботи системи, яка забезпечує їх підготовку до аналітичного аналізу. На цьому етапі завантажені користувачем дані проходять серію трансформацій, валідацій та розрахунків, щоб стати придатними для подальшого використання в аналітичних інструментах.

Попередня обробка даних

Після завантаження даних у систему починається їх автоматична обробка. Цей процес включає такі основні етапи:

1. Перевірка даних на помилки:

- **Відсутні значення:** система ідентифікує порожні поля та автоматично заповнює їх відповідними значеннями (наприклад, нулями для числових даних або спеціальними маркерами для текстових полів).
- **Аномальні значення:** наприклад, якщо вказано негативне споживання електроенергії, система помічає такі рядки як помилкові.

2. Нормалізація даних: Усі числові значення конвертуються до стандартного формату. Наприклад, якщо дані про споживання електроенергії введені в різних одиницях, система автоматично конвертує їх до кВт·год.

3. Видалення дублікатів: Система перевіряє наявність однакових записів (наприклад, дані одного лічильника за той самий період) і видаляє дублікати, залишаючи лише унікальні записи.

Сегментація даних

Для покращення аналітичних розрахунків система сегментує дані за такими параметрами:

- **Географічна сегментація:** поділ за містами або регіонами.
- **Сегментація за типом споживачів:** поділ на побутових і комунальних споживачів.
- **Сегментація за періодами:** поділ на місяці або роки.

Ця сегментація дозволяє проводити детальний аналіз, зокрема порівнювати показники між різними групами споживачів або регіонами.

Агрегація даних

Для аналітики необхідно не лише аналізувати окремі записи, але й узагальнювати їх. Система автоматично виконує такі операції:

- **Сума споживання:** обчислення загальної кількості спожитої електроенергії за певний період.
- **Середнє значення:** визначення середнього споживання електроенергії за день, тиждень або місяць.
- **Мінімальні та максимальні значення:** визначення періодів із найнижчим та найвищим рівнем споживання.

Обробка аномалій

Аномальні значення, такі як різкі скачки або падіння у споживанні електроенергії, виділяються для подальшого аналізу. Це може свідчити про технічні несправності лічильників або несанкціоноване споживання електроенергії.

Система пропонує два способи обробки аномалій:

1. **Автоматична корекція:** система може автоматично коригувати такі значення на основі попередніх трендів.
2. **Сповіщення користувача:** у разі серйозних аномалій користувач отримує сповіщення для ручної перевірки.

Аналітичні розрахунки

1. Розрахунок загального споживання електроенергії

Система автоматично підсумовує дані про спожиту електроенергію для кожного лічильника за вказаний період. Це дозволяє користувачу оцінити загальне навантаження на енергетичну систему.

2. Визначення середньої вартості споживання

На основі ціни за кВт·год, введеної користувачем, система обчислює середню вартість споживання для кожного споживача або групи споживачів.

3. Аналіз пікових навантажень

Система визначає періоди максимального споживання електроенергії, що дозволяє виявити пікові місяці або квартали. Це корисно для оптимізації енергетичних ресурсів.

Користувацький інтерфейс

Головна сторінка

При запуску програми користувача зустрічає інтерактивна мапа України, на якій відображено всі міста, для яких здійснено постачання електроенергії та проведено аналітичний аналіз. Кожне місто позначене точкою, що вказує на географічне розташування, а також дозволяє отримати швидкий доступ до даних конкретного міста.

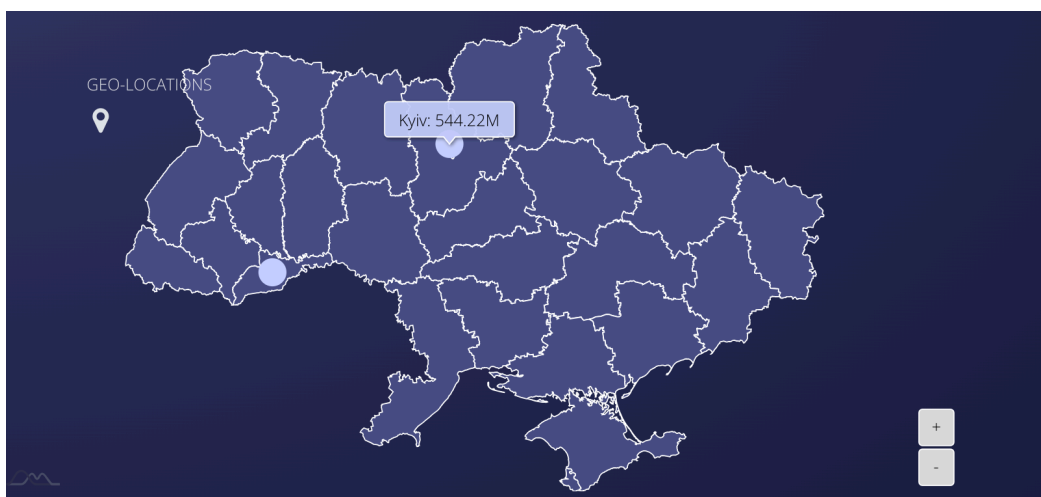


Рис.3.3. Інтерактивна мапа

З правого боку інтерфейсу розміщено інформаційну панель із загальними показниками експорту електроенергії. Вона містить наступні дані:

- Загальні обсяги експорту електроенергії:
 - За рік.
 - За весь час.
 - За місяць.
- Середні показники споживання для всіх міст.

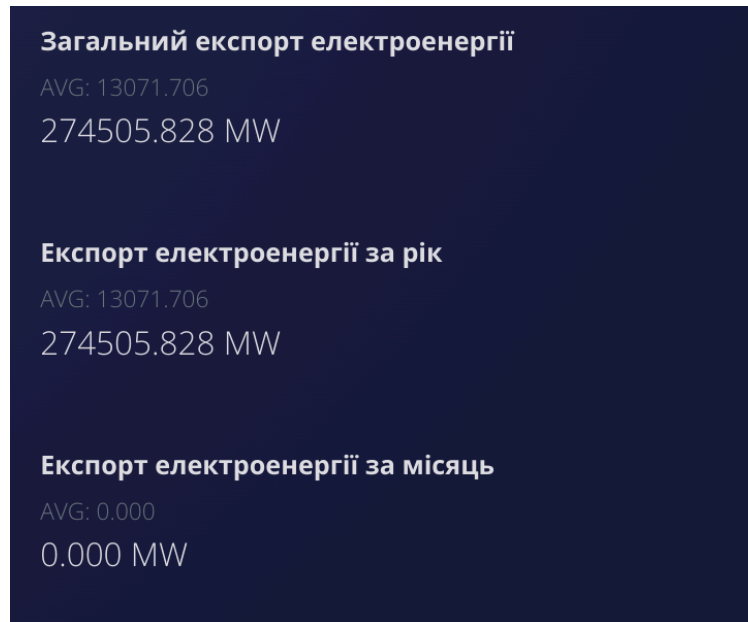


Рис.3.4. Загальні обсяги постачання

У нижній частині сторінки розташовані три ключові елементи:

1. **Таблиця загального прибутку по всіх містах** (без урахування податків та собівартості електроенергії).
2. **Таблиця експорту електроенергії по конкретних містах:**
 - Дані доступні як за весь час, так і середні значення.
3. **Інтерактивний графік споживання електроенергії по містах за рік.**
 - Користувач може обрати, які міста відображати на графіку, або сховати непотрібні.

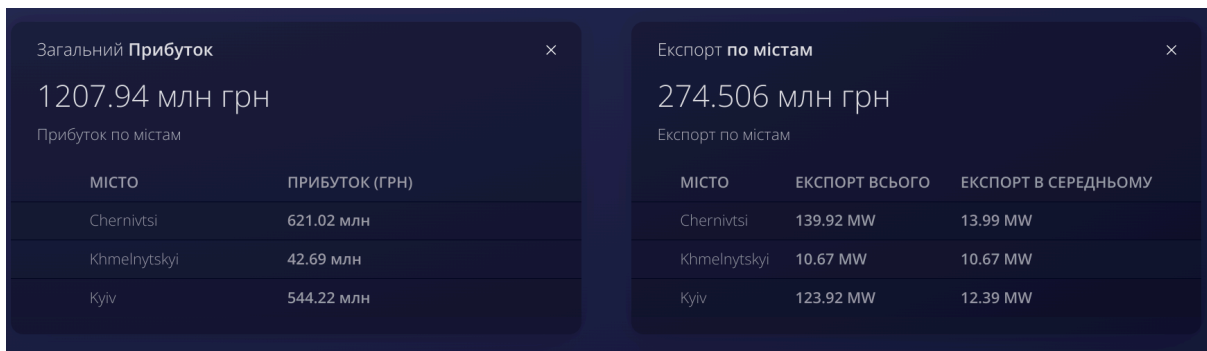


Рис.3.5. Таблиці з даними по містах

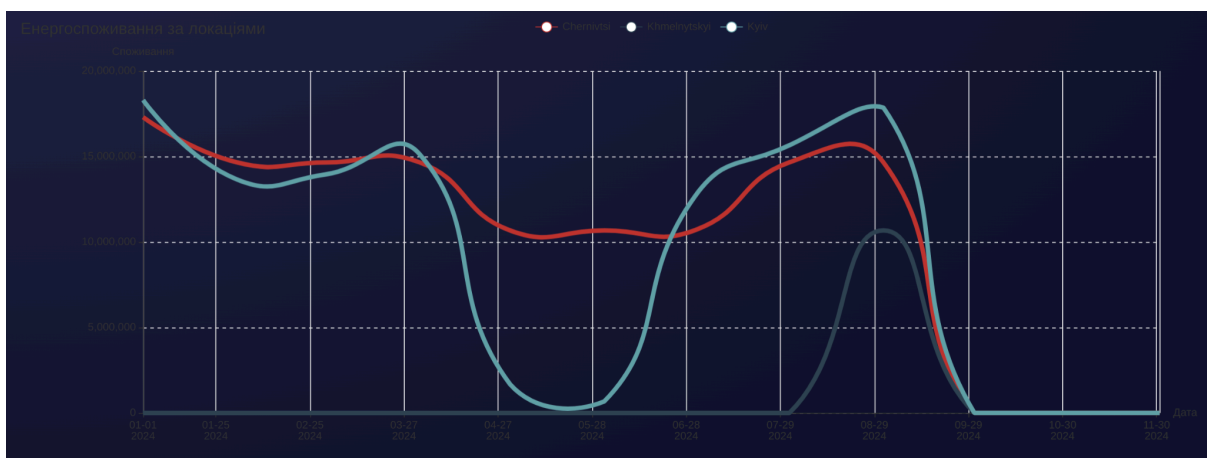


Рис.3.6. Інтерактивний графік експорту енергії за рік

Розділ "Всі таблиці"

У вкладці "**Всі таблиці**" зібрані результати всіх аналітичних обчислень. Кожен запис містить наступну інформацію:

- Назву аналізу.
- Дату та час його створення.
- Коментар, якщо він був доданий користувачем.

Користувач може обрати конкретний аналіз зі списку для детального перегляду.

12	Побутові споживачі	Chernivtsi	Побутові споживачі Липень	01.07.2024	19.11.2024
13	Побутові споживачі	Chernivtsi	Побутові споживачі Серпень	01.08.2024	19.11.2024
14	Побутові споживачі	Chernivtsi	Побутові споживачі Вересень	01.09.2024	19.11.2024
15	Побутові споживачі	Chernivtsi	Побутові споживачі Жовтень	01.10.2024	19.11.2024
16	Побутові споживачі	Kyiv	Побутові споживачі	19.01.2024	19.11.2024
17	Побутові споживачі	Kyiv	Побутові споживачі Лютий	01.02.2024	19.11.2024
18	Побутові споживачі	Kyiv	Побутові споживачі Березень Збільшення ціни	01.03.2024	19.11.2024
19	Побутові споживачі	Kyiv	Побутові споживачі Квітень	01.04.2024	19.11.2024
20	Побутові споживачі	Kyiv	Побутові споживачі Травень	01.05.2024	19.11.2024
21	Побутові споживачі	Kyiv	Побутові споживачі Червень зміна ціни	01.06.2024	19.11.2024
22	Побутові споживачі	Kyiv	Побутові споживачі Липень	01.07.2024	19.11.2024
23	Побутові споживачі	Kyiv	Побутові споживачі Серпень	01.08.2024	19.11.2024

Рис.3.7. Розділ “Всі таблиці”

Детальний аналіз

На сторінці детального аналізу відображається вся доступна інформація щодо обраного аналізу. Користувач може побачити:

- **Дата завантаження даних.**
- **Кількість рядків і стовпців у файлі.**
- **Загальні статистичні показники, такі як:**
 - Максимальне, середнє, мінімальне та медіанне споживання.
 - Загальний обсяг споживання.



Рис.3.8. Віджети з інформацією про обраний аналіз

Під цими даними розміщено:

- **Стовпчиковий графік споживання електроенергії по всіх вулицях міста:**
 - Графік динамічний: можна обирати лише певні вулиці для відображення.

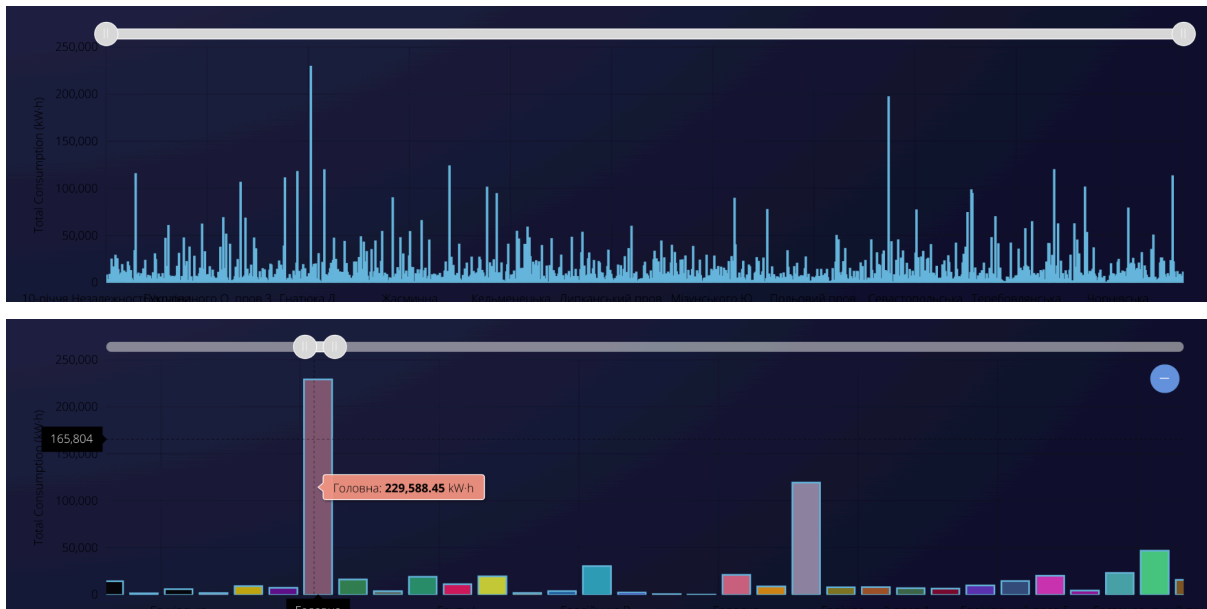


Рис.3.9. Інтерактивний графік споживання

Далі представлено:

- **Розподіл на кластери:**
 - Кількість кластерів визначається автоматично, а для кожного з них показано середнє споживання.
 - Графік кластерів відображає, як алгоритм розподілив дані.

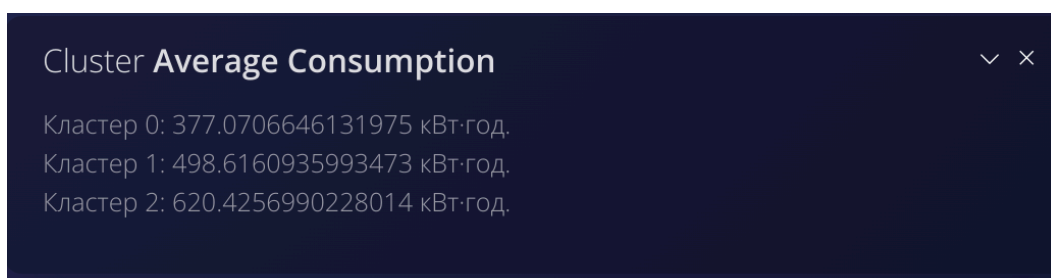


Рис.3.10. Розподіл кластерів

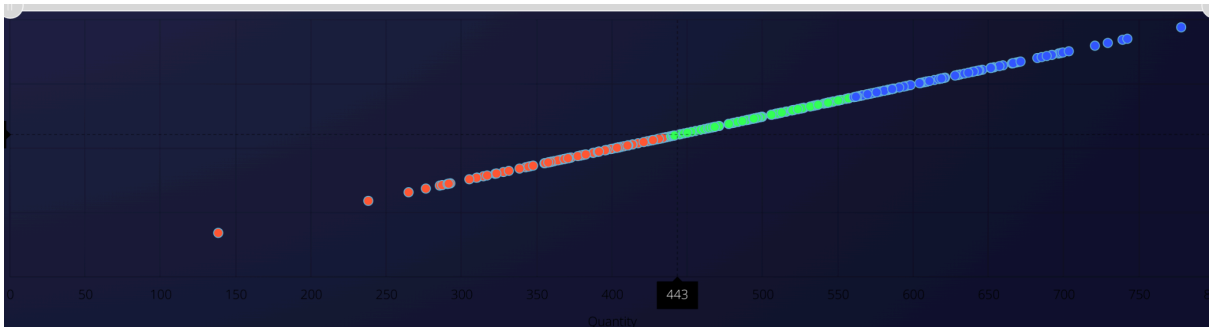


Рис.3.11. Графік розподілу кластерів

- **Аномальні дані:**
 - У випадку наявності аномальних значень (наприклад, дуже високого споживання), вони виводяться у вигляді окремої таблиці.
- **Графік точкового споживання по вулицях:**
 - Він дозволяє користувачеві вибрати діапазон споживання для відображення відповідних вулиць.
 - Аномальні показники виділяються візуально.

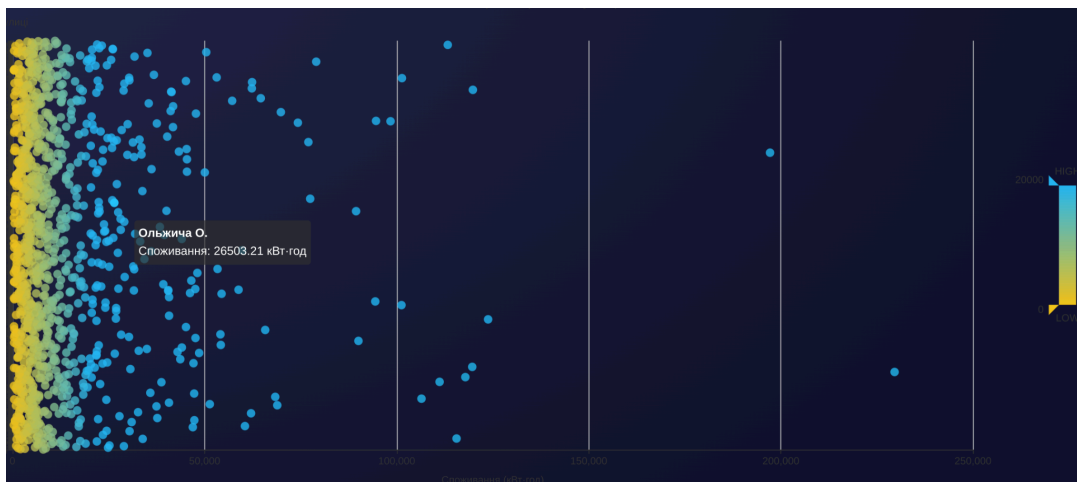


Рис.3.12. Інтерактивний точковий графік споживання

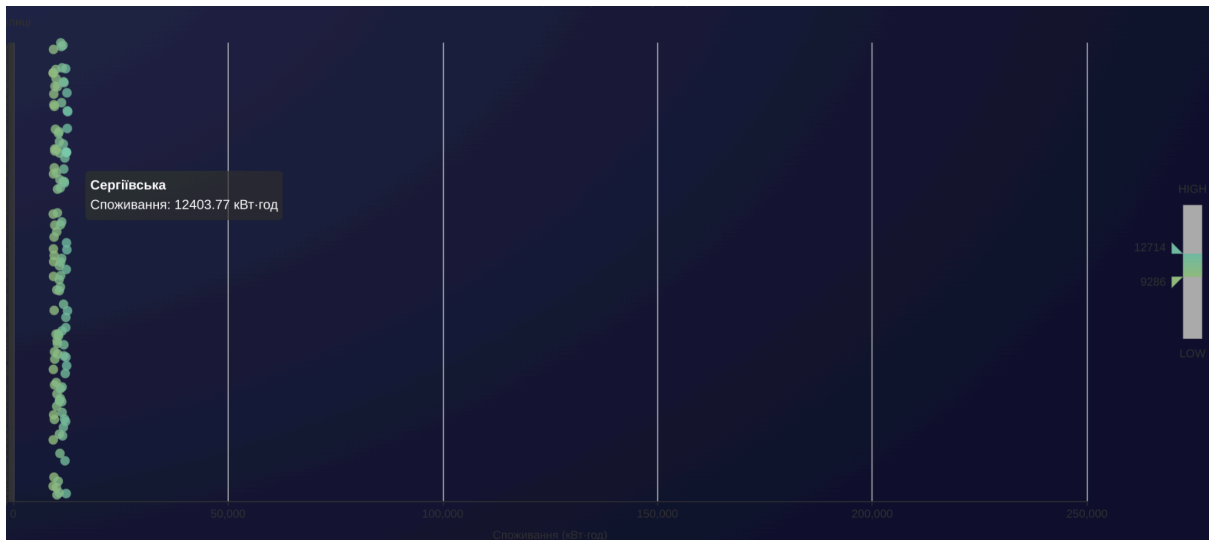


Рис.3.13. Зміна діапазону на точковому графіку

Розділ "Міста"

Таблиця Файлів CSV

#	МІСТО	ЗАГАЛЬНИЙ ПРИБУТОК	ЗАГАЛЬНИЙ ЕКСПОРТ
#	Chernivtsi	621.02млн	139.92MW
#	Khmelnitskyi	42.69млн	10.67MW
#	Kyiv	544.22млн	123.92MW

Базова таблиця з файлів CSV

Відправити в... Очистити

Рис.3.15. Розділ міста

Цей розділ містить аналіз споживання електроенергії для кожного міста за рік. Вибравши місто, користувач отримує доступ до таких даних:

- **Річне споживання.**
- **Річний дохід**, розбитий по місяцях.
- Інші аналітичні показники, включаючи середнє споживання, пікове навантаження тощо.

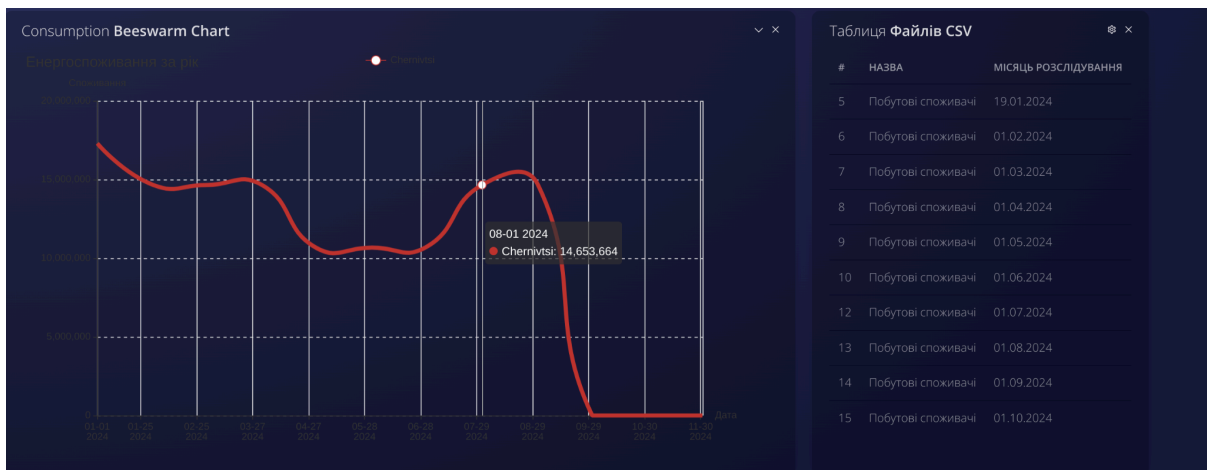


Рис.3.16. Річний графік споживання та доступні аналітичні дані



Рис.3.17. Річний графік прибутку та дані про споживання

Особливо цінним є блок прогнозування:

- **Передбачуване споживання на наступний місяць.**
 - Прогноз будується на основі аналізу даних за останні шість місяців.
 - Графік прогнозу дозволяє візуалізувати очікувані показники споживання, що допомагає у плануванні.

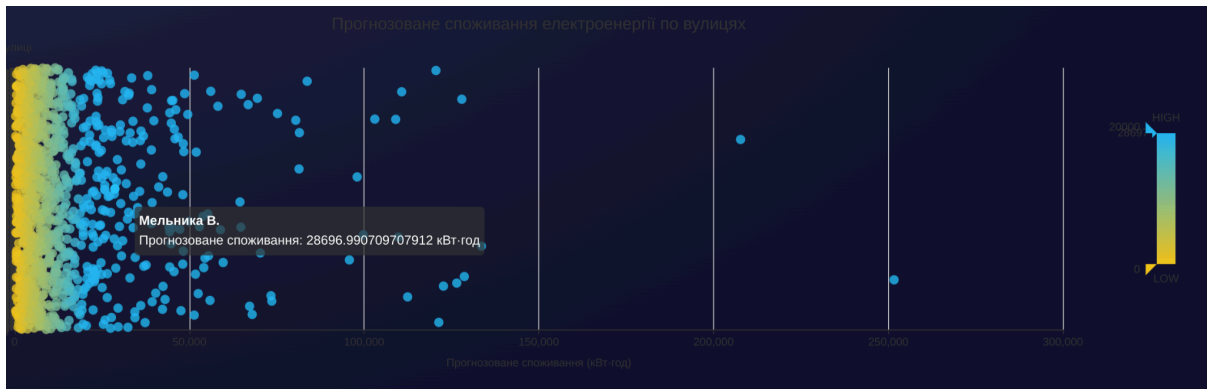


Рис.3.18. Візуалізація прогнозованого споживання

3.2 Застосування програми

Програма призначена для компаній, які займаються експортом електроенергії, зокрема для оптимізації їхньої діяльності та підвищення ефективності прийняття рішень. Уявімо, що компанія *ЕнергоЕкспорт* використовує цей додаток для аналізу постачання електроенергії до кількох країн-сусідів.

Приклад:

Компанія постачає електроенергію до 10 міст. Їхнє завдання — проаналізувати прибутковість експорту, виявити потенційні точки зростання, знизити втрати та оцінити споживання електроенергії в кожному місті для планування майбутніх поставок.

1. Аналіз загальних показників:

За допомогою головної сторінки керівництво компанії бачить загальні обсяги експорту електроенергії за рік, а також середні показники споживання для всіх міст. Це дозволяє швидко оцінити загальний стан справ.

2. Детальний аналіз по містах:

У вкладці "Міста" вони обирають конкретне місто та аналізують його споживання за останній рік. Програма показує, що, наприклад, місто N споживає найбільше електроенергії, але дохід із цього міста нижчий.

3. Виявлення аномалій:

Програма виявляє місто, в якому зафіксовано аномально високе споживання електроенергії в окремі періоди. Завдяки цьому компанія може перевірити контракти або обговорити з місцевими представниками причини підвищеного споживання.

4. Прогнозування:

На основі прогнозу споживання на наступний місяць компанія планує збільшення поставок для міст, які потребуватимуть більше енергії, та зменшує експорт до менш споживаючих міст, знижуючи витрати на доставку.

5. Кластеризація:

Використовуючи дані кластеризації, компанія визначає групи міст із подібними рівнями споживання. Це дозволяє впроваджувати спеціальні тарифні плани для кожного кластера, що підвищує прибутковість.

ВИСНОВКИ

Розроблена програма має велике значення для компаній-експортерів електроенергії, оскільки забезпечує наступні переваги:

1. Оптимізація процесу аналізу даних:

Інтерактивні графіки, таблиці та аналітичні панелі дозволяють швидко отримати ключову інформацію про експорт електроенергії та приймати обґрунтовані рішення.

2. Виявлення проблемних зон:

Завдяки функціям виявлення аномалій компанія може оперативно реагувати на відхилення в споживанні, знижуючи фінансові ризики.

3. Прогнозування та планування:

Програма допомагає передбачити майбутнє споживання електроенергії, що дозволяє більш ефективно планувати обсяги постачання, зменшувати витрати та підвищувати дохід.

4. Гнучкість у роботі з даними:

Користувачі можуть аналізувати дані на різних рівнях деталізації: від загального огляду до детальних показників по кожному місту чи навіть вулиці.

5. Зручність використання:

Зрозумілий інтерфейс із інтерактивними елементами робить програму доступною навіть для користувачів без глибоких технічних знань.

Загалом, додаток значно спрощує управління експортом електроенергії, підвищуючи його ефективність і прибутковість.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Офіційна документація pandas [Електронний ресурс] : <https://pandas.pydata.org/docs/>
2. Офіційна документація openpyxl [Електронний ресурс] : <https://openpyxl.readthedocs.io/en/stable/>
3. Офіційна документація [Електронний ресурс] : <https://docs.nestjs.com/>
4. Nest.js: A progressive Node.js framework [Електронний ресурс] : <https://github.com/nestjs/nest>
5. Офіційна документація scikit-learn (K-Means, DBSCAN) [Електронний ресурс] : <https://scikit-learn.org/stable/modules/clustering.html>
6. Hastie, T., Tibshirani, R., Friedman, J. "The Elements of Statistical Learning" [Електронний ресурс] : <https://web.stanford.edu/~hastie/ElemStatLearn/>
7. Офіційна документація Prophet [Електронний ресурс] : <https://facebook.github.io/prophet/>
8. Taylor, S.J., Letham, B. "Prophet: Forecasting at Scale", 2018.
9. PostgreSQL офіційна документація [Електронний ресурс] : <https://www.postgresql.org/docs/>
10. TypeORM офіційна документація [Електронний ресурс] : <https://typeorm.io/>
11. Chart.js для побудови графіків [Електронний ресурс] : <https://www.chartjs.org/docs/>
12. Python Data Science Handbook by Jake VanderPlas [Електронний ресурс] : <https://jakevdp.github.io/PythonDataScienceHandbook/>
13. Офіційна документація NumPy [Електронний ресурс] : <https://numpy.org/doc/stable/>
14. Офіційна документація Vue.js [Електронний ресурс] : <https://vuejs.org/guide/introduction.html>
15. Закон України «Про місцеве самоврядування в Україні» [Електронний ресурс] : <https://zakon.rada.gov.ua/laws/show/280/97-вр>
16. Державне підприємство «Український науково-дослідний і навчальний центр проблем стандартизації, сертифікації та якості». ДСТУ ISO 50001:2014 «Енергозбереження. Системи енергетичного

менеджменту. Вимоги та настанова щодо використання»
[Електронний ресурс] : <https://online.budstandart.com>

ДОДАТКИ

- **Модуль додатку на Nest.js**

```
@Module({
  imports: [
    JwtModule.register({
      secret: 'secretKey',
      signOptions: { expiresIn: '30d' },
    }),
    TypeOrmModule.forRoot({
      type: 'postgres',
      host: 'localhost',
      port: 5432,
      username: 'postgres',
      password: 'qwerty',
      database: 'wattscope',
      entities: [User, AnalyzedDataEntity, PredictedEntity],
      synchronize: true,
    }),
    TypeOrmModule.forFeature([User, AnalyzedDataEntity, PredictedEntity])
  ],
  controllers: [AppController],
  providers: [AppService, UsersService, AuthService, AnalyzedDataRepository],
})
export class AppModule {}
```

- **Контролери Nest.js з перевіркою JWT токена**

```
@Get('stats/data-by-id/:id')
async getDataById(@Param('id') id: number): Promise<any> {
  return await this.appService.getDataById(id);
}

@UseGuards(JwtAuthGuard)
@Get('global-stats')
async getGlobalStatsByUserId(@Request() req): Promise<any> {
  const userEmail = req.user.email;
  const user = await this.usersService.findOneByEmail(userEmail);

  const result = await this.appService.getGlobalStatsByUserId(user.id);
  return result;
}

@Post('register')
async register(
  @Body('email') email: string,
  @Body('password') password: string,
  @Body('username') username: string,
) {
  return this.usersService.create(email, password, username);
}

@Post('login')
```

```

async login(
  @Body('email') email: string,
  @Body('password') password: string,
) {
  const user = await this.authService.validateUser(email, password);
  if (!user) {
    throw new Error('Invalid credentials');
  }
  return this.authService.login(user);
}

@UseGuards(JwtAuthGuard)
@Get('users/all-orders-by-user-id')
async getAllOrdersByUserId(@Request() req): Promise<any> {
  const userEmail = req.user.email;
  const user = await this.userService.findOneByEmail(userEmail);
  const result = await this.appService.getAllOrdersByUserId(user.id);
  return result;
}

@UseGuards(JwtAuthGuard)
@Get('users/year-cons-stats')
async getYearConsStats(@Request() req): Promise<any> {
  const userEmail = req.user.email;
  const user = await this.userService.findOneByEmail(userEmail);
  const result = await this.appService.getTotalConsYearStats(user.id);
  return result;
}

@UseGuards(JwtAuthGuard)
@Get('users/get-user-by-jwt')
async getUserByJwt(@Request() req): Promise<any> {
  const userEmail = req.user.email;
  return await this.userService.findOneByEmail(userEmail);
}

@UseGuards(JwtAuthGuard)
@Get('cities/get-city-stats')
async getCityStats(@Request() req): Promise<any> {
  const userEmail = req.user.email;
  const city = req.query.city;
  const user = await this.userService.findOneByEmail(userEmail);
  return await this.appService.getCityStats(user.id, city);
}

@UseGuards(JwtAuthGuard)
@Get('users/all-city-orders-by-user-id/')
async getAllCityOrdersByUserId(@Request() req): Promise<any> {
  const userEmail = req.user.email;
  const city = req.query.city;
  const user = await this.userService.findOneByEmail(userEmail);
  const result = await this.appService.getAllCityOrdersByUserId(user.id, city);
}

```

```

return result;
}

```

```

@UseGuards(JwtAuthGuard)
@Get('users/city-year-cons-stats/')
async getCityYearConsStats(@Request() req): Promise<any> {
  const userEmail = req.user.email;
  const city = req.query.city;
  const user = await this.userService.findOneByEmail(userEmail);
  const result = await this.appService.getCityTotalConsYearStats(user.id, city);
  return result;
}

```

```

@UseGuards(JwtAuthGuard)
@Get('city/get-predicted')
async getPredicted(@Request() req): Promise<any> {
  const userEmail = req.user.email;
  const city = req.query.city;
  const user = await this.userService.findOneByEmail(userEmail);
  const result = await this.appService.getPredicted(user.id);
  return result;
}

```

- **Приклад реалізації сервісних методів Nest.js**

```

async getGlobalStatsByUserId(userId: number): Promise<any> {
  const consumptions = await this.analyzedDataRepository.getConsumptionStatsByUserId(userId);
  const consumptionsByRegion = await
this.analyzedDataRepository.getConsumptionStatsByRegion(userId);
  const consumptionPrice = await this.analyzedDataRepository.getEarningsByLocation(userId);
  return {
    cityPrices: consumptionPrice,
    consumptions: consumptions,
    cityConsumptions: consumptionsByRegion
  };
}

```

```

async getAllOrdersByUserId(userId: number): Promise<any> {
  const orders = await this.analyzedDataRepository.getAllOrdersByUserId(userId);

  const filteredOrders = orders.map(order => ({
    id: order.id,
    userId: order.userId,
    name: order.name,
    location: order.location,
    description: order.description,
    investigationMonth: order.investigationMonth,
    uploadDate: order.uploadDate,
  }));

  return filteredOrders;
}

```

```

async getTotalConsYearStats(userId: number): Promise<any> {
  const results = await this.analyzedDataRepository.getYearlyConsumptionDetails(userId);
  return results
}

```

- **Приклад репозиторія для взаємодії з базою даних**

```

async getAllOrdersByUserId(userId: number) {
  return await this.analyzedDataRepository.find({ where: { userId } });
}

```

```

async getAllCityOrdersByUserId(userId: number, city: string) {
  return await this.analyzedDataRepository.find({where: {userId: userId, location: city}})
}

```

```

async getYearlyConsumptionDetails(userId: number) {
  const results = await this.analyzedDataRepository
    .createQueryBuilder('data')
    .select([
      'data.location AS location', // Місто
      'data.totalConsumption AS totalConsumption', // Споживання
      'data.investigationMonth AS investigationMonth' // Місяць
    ])
    .where('data.userId = :userId', { userId }) // Фільтр по користувачу
    .andWhere('EXTRACT(YEAR FROM data.investigationMonth) = EXTRACT(YEAR FROM CURRENT_DATE)') // Тільки дані за поточний рік
    .orderBy('data.location', 'ASC') // Сортуювання за містом
    .addOrderBy('data.investigationMonth', 'ASC') // Сортуювання за місяцем
    .getRawMany();
  return results;
}

```

- **Приклад сутності бази даних**

```

@Entity()
export class AnalyzedDataEntity {
  @PrimaryGeneratedColumn()
  id: number;
  @Column()
  userId: number;
  @Column()
  unitPrice:number;
  @Column()
  processing: string;
  @Column()
  name: string;
  @Column()
  location: string;
  @Column()
  description: string;
  @Column()

```

```

investigationMonth: Date;
@Column()
uploadDate: Date;
@Column()
status: string;
@Column()
totalRows: number;
@Column()
totalColumns: number;
@Column()
totalNullValues: number;
@Column()
maxConsumption: number;
@Column()
meanConsumption: number;
@Column()
minConsumption: number;
@Column()
medianConsumption: number;
@Column()
totalConsumption: number;
@Column()
sumMaxStreetConsumption: string;
@Column()
sumMinStreetConsumption: string;
@Column('json')
maxStreetConsumption: { street: string, consumption: number };
@Column('json')
minStreetConsumption: { street: string, consumption: number };
@Column('jsonb')
organizationMaintenanceData: { organization: string; consumption: number }[];
@Column('jsonb')
clustersAverageConsumption: { cluster: string; consumption: number }[];
@Column('jsonb')
anomalyResults: { id: number; quantity: string; anomaly: number }[];
@Column('jsonb')
allStreetsConsumption: { street: string; consumption: number }[];
@Column('jsonb')
avgDataPerCluster: { clusterId: string; quantity: number, street: string }[];
}

```

- **Приклад контролера на FastAPI**

```

@router.post("/send/data-file")
async def calculate_first_data(service: ServiceDataBaseConnection, file: UploadFile = File(...), data:
str = Form(...)):
    file_contents = await file.read()
    other_data = json.loads(data)
    await service.calculate(file_contents, other_data)
    return True

@router.get('/predict_for_city')

```

```

async def predict_for_city(service: ServiceDataBaseConnection):
    city = 'Chernivtsi'
    return await service.prediction(city)

```

- **Приклад моделі бази даних на FastAPI**

```

class DataStatistics(Base):
    __tablename__ = 'analyzed_data_entity'

    id = Column(Integer, primary_key=True)
    name = Column(String)
    userId = Column(Integer)
    unitPrice = Column(Integer)
    processing = Column(String)
    location = Column(String)
    description = Column(String)
    investigationMonth = Column(Date)
    uploadDate = Column(Date)
    status = Column(String)
    totalRows = Column(Integer)
    totalColumns = Column(Integer)
    totalNullValues = Column(Integer)
    maxConsumption = Column(Integer)
    meanConsumption = Column(Integer)
    minConsumption = Column(Integer)
    medianConsumption = Column(Integer)
    totalConsumption = Column(Integer)
    sumMaxStreetConsumption = Column(String)
    sumMinStreetConsumption = Column(String)
    maxStreetConsumption = Column(JSONB)
    minStreetConsumption = Column(JSONB)
    organizationMaintenanceData = Column(JSONB)
    clustersAverageConsumption = Column(JSONB)
    anomalyResults = Column(JSONB)
    allStreetsConsumption = Column(JSONB)
    avgDataPerCluster = Column(JSONB)

```

Знаходження глобальних метрик на Python

```

results['max_consumption'] = df_clean['quantity'].max()
results['mean_consumption'] = df_clean['quantity'].mean()
results['min_consumption'] = df_clean['quantity'].min()
results['median_consumption'] = df_clean['quantity'].median()
results['total_consumption'] = df_clean['quantity'].sum()

```

- **Приклад роботи з кластеризованими методом к-середніх даними**

```

df_scaled = df_clean.copy()
scaler = StandardScaler()
df_scaled.loc[:, 'quantity'] = scaler.fit_transform(df_scaled[['quantity']])
df_scaled['quantity_scaled'] = scaler.fit_transform(df_scaled[['quantity']])

```

Метод 1: Аналіз кластерів за обсягом споживання енергії

```

df_dbscan = df_scaled.copy()

```

```

kmeans = KMeans(n_clusters=3, random_state=12)
df_dbscan['kmeans_cluster'] = kmeans.fit_predict(df_dbscan[['quantity_scaled']])

# Знаходження середнього quantity для кожного кластера (використовуємо оригінальні дані в df_clean)
df_clean['kmeans_cluster'] = df_dbscan['kmeans_cluster']
mean_quantity_per_cluster = df_clean.groupby('kmeans_cluster')['quantity'].mean()

clusters_avg = [
    {"cluster": clusters, "consumption": consumption}
    for clusters, consumption in mean_quantity_per_cluster.items()
]

```

- **Виявлення аномалій за допомогою DBSCAN**

```

df_anomalies = df_scaled.copy()
dbscan = DBSCAN(eps=0.5, min_samples=5)
df_anomalies['anomaly'] = dbscan.fit_predict(df_anomalies[['quantity']])
anomalies = df_anomalies[df_anomalies['anomaly'] == -1]
anomalys = [
    {"id": idx + 1, "quantity": row['quantity'], "anomaly": row['anomaly']}
    for idx, row in anomalies.iterrows()
]

```

```

# Додаємо до результатів
results['anomaly_results'] = anomalys

```

- **Передбачення споживання за допомогою моделі Prophet**

```

# 3. Прогноз для кожної вулиці
forecasts = []
for street, data in street_data.items():
    df = pd.DataFrame(data)
    df["month"] = pd.to_datetime(df["month"])
    df = df.sort_values("month")
    df = df.rename(columns={"month": "ds", "consumption": "y"})

    # Ініціалізація моделі Prophet
    model = Prophet()
    model.fit(df)

    # Прогноз на наступний місяць
    future = model.make_future_dataframe(periods=1, freq="M")
    forecast = model.predict(future)

    # Збереження прогнозу
    next_month_prediction = forecast.iloc[-1]["yhat"]

    # Знаходження статистики
    max_value = df["y"].max()
    min_value = df["y"].min()
    mean_value = df["y"].mean()

```

```

    forecasts.append({
        "street": street,
        "next_month_prediction": next_month_prediction,
        "max_consumption": max_value,
        "min_consumption": min_value,
        "mean_consumption": mean_value
    })

# Сортуюмо результати за назвою вулиці для кращої читабельності
forecasts = sorted(forecasts, key=lambda x: x["street"])

# Виводимо перші 10 результатів
print("Перші 10 прогнозів:")
for result in forecasts[:10]:
    print(json.dumps(result, indent=2, ensure_ascii=False))

# Виводимо статистику по всіх даних
all_consumptions = [data["consumption"] for street in street_data for data in street_data[street]]
total_max = max(all_consumptions)
total_min = min(all_consumptions)
total_mean = sum(all_consumptions) / len(all_consumptions)

```

- **Приклад роботи віджету для відображення прибутку в кожному місті на Vue.js**

```

<Widget title="<h6> Загальний <span class='fw-semi-bold'>Прибуток</span></h6>" close
customHeader>
<div class="widget-body">
  <h3>{{ totalEarnings.toFixed(2) }} млн грн</h3>
  <p class="fs-mini text-muted mb mt-sm">
    <span class="text-align-right">Прибуток по містах</span>
  </p>
</div>
<div class="widget-bottom-overflow">
  <table class="table table-striped table-sm market-stats">
    <thead class="no-bd">
      <tr>
        <th>&nbsp;</th>
        <th>Місто</th>
        <th class="text-align-right">Прибуток (грн)</th>
      </tr>
    </thead>
    <tbody>
      <tr v-for="(city, index) in cityPrices" :key="index">
        <td>
          </td>
        <td>{{ city.data_location }}</td>
        <td class="text-align-right fw-semi-bold">{{ (city.totalearnings / 1000000).toFixed(2) }}
млн</td>
      </tr>
    </tbody>
  </table>
</div>

```

</Widget>

- **Приклад перевірки JWT токена та отримання даних з бекенду**

```
try {
  const token = localStorage.getItem('jwt'); // Токен з локального сховища

  const response = await axios.get('http://localhost:3003/global-stats/', {
    headers: {
      Authorization: `Bearer ${token}`
    }
  });
  console.log(response.data)
  this.consumptions = response.data.consumptions;
  this.cityPrices = response.data.cityPrices;
  this.cityConsumptions = response.data.cityConsumptions;

  const responseGlobal = await axios.get('http://localhost:3003/users/year-cons-stats', {
    headers: {
      Authorization: `Bearer ${token}`
    }
  });

  this.chartData = responseGlobal.data;

  console.log("chartData", responseGlobal.data);

  this.initChart();
```

- **Приклад створення графіку типу BessWarm**

```
renderBeeswarmChart(data) {
  this.chart = am4core.create("beeswarmChart", am4charts.XYChart);
  this.chart.data = data;

  // Налаштування осі X (sum_value)
  const valueAxisX = this.chart.xAxes.push(new am4charts.ValueAxis());
  valueAxisX.title.text = "Consumption (kW·h)";
  valueAxisX.min = 0; // Мінімум осі X
  valueAxisX.max = 200000; // Максимум осі X
  valueAxisX.strictMinMax = true;
  valueAxisX.renderer.baseGrid.disabled = true;

  valueAxisX.renderer.labels.template.adapter.add("text", function(text, target) {
    const value = target.dataItem.value;
    if (value === 15000) {
      return "15,000"; // Для центру відображаємо 15,000
    }
    return text;
  });
}
```

```

// Налаштування осі Y (-1 до 1)
const valueAxisY = this.chart.yAxes.push(new am4charts.ValueAxis());
valueAxisY.title.text = "Position";
valueAxisY.min = -1; // Мінімум осі Y
valueAxisY.max = 1; // Максимум осі Y
valueAxisY.renderer.baseGrid.disabled = true;
valueAxisY.renderer.labels.template.disabled = true; // Приховуємо підписи на осі Y

// Створення серії для кульок
const series = this.chart.series.push(new am4charts.XYSeries());
series.dataFields.valueX = "sum_value"; // Споживання (для осі X)
series.dataFields.valueY = "position"; // Позиція (для осі Y)
series.name = "Consumption";

// Генерація випадкового значення для осі Y в діапазоні від -0.5 до 0.5
data.forEach(item => {
    item.position = Math.random() * 1 - 0.5; // Випадкове значення для Y між -0.5 та 0.5
});

// Створення кульок
series.bullets.push(new am4charts.CircleBullet()); // Використовуємо кульки
series.bullets.push(new am4charts.LabelBullet()); // Додаємо підпис до кульки

// Підписи до кульок
// series.bullets.getIndex(1).label.text = "{street_name}"; // Підписи з назвами вулиць
series.tooltipText = "{categoryY}: [bold]{valueX}[/] kW·h"; // Підказка при наведенні на кульку

// Кульки будуть відображатися тільки при наведенні
series.tooltip.getFillFromObject = false;
series.tooltip.background.fill = am4core.color("#fff");
series.tooltip.background.strokeWidth = 1;
series.tooltip.background.stroke = am4core.color("#000");

// Колір кульок (випадкові кольори)
series.bullets.getIndex(0).circle.fill = am4core.color("#ff5733"); // Задаємо колір кульок (для прикладу)
series.bullets.getIndex(0).circle.stroke = am4core.color("#000000");
series.bullets.getIndex(0).circle.strokeWidth = 2;

// Розподіл кульок по осі X від центру (де 15,000 в центрі)
// Розподіл кульок по осі X від центру (де 15,000 в центрі)
valueAxisX.renderer.labels.template.adapter.add("text", function(text, target) {
    const value = target.dataItem.value;
    if (value === 15000) {
        return "15,000"; // Для центру відображаємо 15,000
    }
    return text;
});

// Додаємо курсор для графіка
this.chart.cursor = new am4charts.XYCursor();
this.chart.cursor.behavior = "zoomY";

```

```
this.chart.scrollbarX = new am4core.Scrollbar();

// Додаємо випадкові кольори кулькам
series.bullets.getIndex(0).circle.adapter.add("fill", (fill, target) => {
    return am4core.color("#" + Math.floor(Math.random() * 16777215).toString(16)); // Випадкові
    кольори
})
},
```