

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ
кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

“До захисту допущено”
Завідувач кафедри ЦТЕ
_____ Наталія АУШЕВА

“ ” _____ 2024 р.

Дипломна робота
на здобуття ступеня бакалавра
за освітньо-професійною програмою
“Цифрові технології в енергетиці”
зі спеціальності 122 “Комп’ютерні науки”

на тему: Веб-застосунок особистого кабінету споживача
електроенергії АТ “НАЕК “Енергоатом”

Виконав (-ла):
студент (-ка) IV курсу, групи ТР-03

ТОПАЛОВА Ольга Євгенівна
(прізвище, ім’я, по батькові) _____ (підпис)

Керівник: доцент каф. цифрових технологій в енергетиці
доцент, к.т.н., Ігор КУЗЬМЕНКО
(посада, вчене звання ,науковий ступінь, ім’я, ПРІЗВИЩЕ) _____ (підпис)

Рецензент: Директор філії “ВП НТЦ”,
Олег ЗЕЛЕНИЙ
(посада, вчене звання, науковий ступінь, ім’я, ПРІЗВИЩЕ) _____ (підпис)

Н.контроль: асистент Микита ГОЛОВАКІН
(посада, ім’я, ПРІЗВИЩЕ) _____ (підпис)

Засвідчую, що у цій дипломній роботі немає
запозичень з праць інших авторів без
відповідних посилань.
Студент (-ка) _____ (підпис)

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”**

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ

Кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

Рівень вищої освіти – перший (бакалаврський)

спеціальність 122 “Комп’ютерні науки”

Освітньо-професійна програма “Цифрові технології в енергетиці”

ЗАТВЕРДЖУЮ

Завідувач кафедри ЦТЕ

Наталія АУШЕВА

«__» _____ 2024 р.

**ЗАВДАННЯ
на дипломну роботу студенту**

ТОПАЛОВІЙ Ользі Євгенівні

(прізвище, ім’я, по батькові)

1. Тема роботи **Веб-застосунок особистого кабінету споживача
електроенергії АТ “НАЕК “Енергоатом”**

керівник роботи **Кузьменко Ігор Миколайович, к.т.н., доцент**

(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «__» _____ 202__ р.

№ _____

2. Термін подання роботи студентом **07.06.2024 року**

3. Вихідні дані до роботи мова програмування JavaScript, фреймворки Spring та React, СКБД MySQL, середовище розробки Visual Studio Code

4. Перелік питань, які потрібно розробити _____

1) провести аналіз серед існуючих рішень особистих кабінетів

2) аргументувати обрані засоби програмної реалізації, технології.

3) розробити та впровадити клієнтську та серверні частини веб-застосунку

4) побудувати архітектури програмного забезпечення, клієнтської та серверної частин, баз даних

5) створити програмний веб-інтерфейс

6) представити процес застосування веб-інтерфейсу

5. Орієнтований перелік ілюстративного матеріалу діаграми, що демонструють роботу алгоритмів веб-інтерфейсу, архітектуру системи, бази даних, взаємодію користувачів із системою, класи програмного забезпечення; приклади роботи з програмним продуктом.

6. Дата видачі завдання «15» вересня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Затвердження теми роботи		
2.	Вивчення та аналіз задачі	17-20.04.24	
3.	Розробка архітектури та загальної структури системи	21-25.04.24	
4.	Розробка частин окремих підсистем	25.04-02.05.24	
5.	Програмна реалізація системи	03-07.05.24	
6.	Оформлення пояснювальної записки	08-12.05.24	
7.	Захист програмного продукту	13-17.05.24	
8.	Передзахист	03-06.06.24	
9.	Подання готової роботи на кафедру	07.06.2024	
10	Захист	17-21.06.2024	
.			

Студент

(підпис)

Ольга ТОПАЛОВА

(ім'я, ПРІЗВИЩЕ)

Керівник роботи

(підпис)

Ігор КУЗЬМЕНКО

(ім'я, ПРІЗВИЩЕ)

АНОТАЦІЯ

Дипломна робота складається з 67 сторінки, на яких представлено 20 рисунків, 1 таблиця, 27 джерел та 2 додатки. Тема роботи: “Веб-застосунок особистого кабінету споживача електроенергії АТ “НАЕК “Енергоатом”. Метою роботи є дослідження та розробка веб-застосунку, який дозволяє користувачу здійснювати моніторинг та аналіз власного енергоспоживання за різні проміжки часу, а також отримувати звітність у зручному форматі.

В даній роботі було розглянуто та проаналізовано існуючі особисті кабінету споживачів електроенергії, а також описано розробку бази даних, інтерфейсу, клієнтської та серверної частин веб-застосунку.

Перелік ключових слів: особистий кабінет, сервер, база даних.

ABSTRACT

The thesis consists of 67 pages, on which 20 figures, 27 sources and 2 appendices are presented. The topic of the work: "Web application of the personal account of the electricity consumer of JSC "NAEK "Energoatom". The purpose of the work is to research and develop a web application that allows the user to monitor and analyze their own energy consumption at different time intervals, as well as receive reports in a convenient format.

In this work, the existing personal accounts of electricity consumers were considered and analyzed, as well as the development of the database, interface, client and server parts of the web application were described.

List of keywords: personal account, server, database.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ.....	8
ВСТУП.....	9
1 РОЗРОБКА ВЕБ-ЗАСТОСУНКУ ОСОБИСТОГО КАБІНЕТУ СПОЖИВАЧА ЕЛЕКТРОЕНЕРГІЇ АТ “НАЕК “ЕНЕРГОАТОМ”.....	11
1.1 Потенційні користувачі.....	11
1.2 Задачі програмного забезпечення.....	12
1.3 Методи вирішення поставлених задач.....	13
2 ПРОБЛЕМАТИКА ТА ОГЛЯД ІСНУЮЧИХ РІШЕНЬ.....	17
2.1 Проблема впровадження веб-застосунків особистих кабінетів для споживачів енергоресурсів в Україні.....	17
2.2 Огляд існуючих рішень.....	19
2.3.1 Особистий кабінет для приватних клієнтів Yasno.....	20
2.3.2 Особистий кабінет побутового споживача ПрАТ «ДТЕК Київські регіональні електромережі».....	23
2.3.3 Особистий кабінет споживача енергоресурсів iEnergо.....	24
3 ЗАСОБИ РОЗРОБКИ ПРОГРАМНОГО ЗАСТОСУНКУ.....	26
3.1 Архітектура програмного забезпечення.....	26
3.2 Розробка бази даних.....	30
3.3 Інтегроване середовище розробки (IDE).....	32
3.4 Мова програмування.....	33
3.4.1 Реалізація frontend.....	34
3.4.2 Реалізація backend.....	35
3.5 Бібліотеки та розширення.....	36
4 ПРОГРАМНА РЕАЛІЗАЦІЯ.....	37

4.1 Загальна структура проекту.....	37
4.1.1 Основний каталог для серверної логіки.....	37
4.1.2 Ресурси для серверної частини.....	39
4.1.3 Файли клієнтської частини.....	40
4.2 Налагодження серверної частини.....	41
4.2.1 Сервер бази даних.....	41
4.2.2 Налагодження веб-серверу.....	43
4.3 Взаємодія клієнтської частини з сервером.....	46
4.4 Алгоритм роботи веб-застосунку.....	48
5 РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ.....	50
5.1 Системні вимоги.....	40
5.2 Приклад інтерфейсу.....	50
ВИСНОВКИ.....	57
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	58
ДОДАТОК А.....	61
ДОДАТОК Б.....	66

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

API (Application Programming Interface) — прикладний програмний інтерфейс.

БД — база даних.

СУБД — системи управління базами даних.

HTTP (Hyper Text Transfer Protocol) — протокол прикладного рівня передачі гіпертекстової інформації.

Фреймворк — структура програмних рішень, що полегшує розробку складних систем.

ORM (Object Relation Mapping) — технологія, що перетворює реляційну модель в класи мови об'єктно—орієнтованої мови програмування.

JPA (Java Persistence API) — стандартизований інтерфейс для Java ORM фреймворків.

DAO (Data Access Object) — об'єкт що надає абстрактний інтерфейс до деяких видів баз даних чи механізмів персистентності реалізуючи певні операції без розкриття деталей бази даних.

MVC (Model-View-Controller) — архітектурний шаблон, який використовується під час проєктування та розробки програмного забезпечення.

UI (User Interface) — інтерфейс користувача

ВСТУП

Питання доступності електроенергії в Україні протягом останніх років в умовах воєнного стану стало хвилюючим, а тема вироблення та споживання звернули на себе увагу громадян, які бажають поглибити свої знання у галузі та дізнатись більше про енергію, що змушує працювати майже усе обладнання у їх оселях.

Особисті кабінети є важливим інструментом у створенні двостороннього контакту між користувачем та стороною, що надає послуги. Зростання попиту на енергію, необхідність зменшення витрат та економного використання електроенергії в умовах дефіциту спонукають людей до пошуку інноваційних рішень. Одним з таких рішень є впровадження веб-застосунків для управління та моніторингу споживанням енергоресурсів. Розробка особистого кабінету споживача зокрема електроенергії стає важливим напрямком досліджень і розробок, оскільки дозволяє забезпечити прозорість і ефективність управління енергоспоживанням.

Метою даної дипломної роботи є створення зручного веб-застосунку для споживачів електроенергії в Україні зі зрозумілим інтерфейсом, візуалізацією витраченої електроенергії та коштів у вигляді графіків, можливістю бачити динаміку власного споживання та отримувати просту зрозумілу аналітику. Веб-застосунок, розроблений у даній дипломній роботі, створено з використанням передових технологій та мов програмування, таких як HTML, CSS, JavaScript, Node.js та бази даних SQL, що забезпечують високу функціональність, адаптивність, надійність і зручність у користуванні.

Завданнями дипломної роботи будуть:

1. Огляд і аналіз існуючих веб-застосунків, що зосереджують увагу на контактуванні із енергоспоживачами у вигляді особистого кабінету та відповідним інструментарієм, що є поширеним серед відповідників.

2. Розробка та реалізація власного веб-застосунку із використанням можливостей баз даних, його клієнтської та серверної частини.

3. Порівняння розробленого продукту з існуючими програмними рішеннями.

4. Аналіз результатів, інструментів, використаних при роботі, та висновки стосовно можливих напрямів подальших досліджень у даній області.

Очікується, що результати дипломної роботи матимуть практичне застосування у взаємодії користувачів із власними енерго- та грошовими витратами, що залучить клієнтів у процес свідомого використання побутових електроприладів та аналізу ведення власних витрат та заощаджень.

1 РОЗРОБКА ВЕБ-ЗАСТОСУНКУ ОСОБИСТОГО КАБІНЕТУ СПОЖИВАЧА ЕЛЕКТРОЕНЕРГІЇ АТ “НАЕК “ЕНЕРГОАТОМ”

Веб-застосунок, розроблений за темою дипломної роботи, призначений для полегшення процесу обліку електроенергії та її вартості споживачами, покращення отримання досвіду моніторингу щоденних, щомісячних та витрат протягом року та можливість визначити орієнтовні обсяги витрат електроенергії та грошей користувача за той проміжок часу, що користувач вкаже у вікні калькулятора.

1.1 Потенційні користувачі

Розвиток цифрових технологій та інновацій у сфері енергетики відкриває нові можливості для покращення взаємодії між постачальниками електроенергії та споживачами. Впровадження особистого кабінету споживача електроенергії – це сучасний підхід та інструмент, що дозволяє користувачам не лише ефективно керувати своїм енергоспоживанням, а й отримувати широкий спектр додаткових послуг, спрямованих на підвищення рівня комфорту та економії коштів.

Опишемо потенційних користувачів веб-застосунку, якими можуть бути наступні групи людей:

Приватні особи: люди, що мають приватну власність, квартира або будинок з ділянкою, планують свідоме споживання електроенергії в межах своєї власності та бажають періодично відслідковувати та отримувати звіти у текстовому форматі або на електронну пошту.

Малі бізнеси: невеликі підприємства, наприклад кафе, кав'ярні, офіси, сімейні бізнеси та домогосподарства, що бажають підвищити ефективність споживання власних ресурсів, бачити динаміку власних витрат та згідно із цими даними розраховувати подальші потреби.

Орендарі: орендарі житлових та комерційних приміщень, які мають на меті контроль витрат та виплат за електроенергію та бажають отримати статистичні дані власного споживання у вигляді звітів, наприклад для орендарів.

1.2 Задачі програмного забезпечення

Для реалізації програмного забезпечення відповідно сформуємо та поставимо конкретні задачі, які забезпечать чітке розуміння вимог та плану виконання роботи. Опишемо задачі, які розв'язуватимуться програмним забезпеченням:

1. Моніторинг енергоспоживання. Користувачі зможуть переглядати дані про споживання ними електроенергії в різні проміжки часу, вважається що візуальне подання інформації, наприклад у вигляді графіків, може допомогти полегшити сприйняття інформації а також ефективніше аналізувати дані, що дозволить їм бачити та виявляти неефективне використання енергоресурсів.

2. Аналіз споживання та статистика. Надати користувачам інструмент для аналізу даних про споживання енергії допомагають користувачам розуміти тенденції у використанні енергії та планувати свої витрати на майбутнє, а також допоможе користувачам звернути увагу на аномальні показники у їх даних.

3. Ефективне зберігання даних. Бази даних SQL дозволять організувати дані у структурованій формі, що забезпечить швидкий доступ та ефективне зберігання великих обсягів інформації, таких як дані про споживання енергії, транзакції та користувацькі профілі, що згодом будуть тільки накопичуватись накопичуватись.

4. Безпека даних. Бази даних забезпечать високий рівень безпеки, включаючи контроль доступу та шифрування, що важливо для захисту конфіденційної інформації користувачів.

5. Масштабованість. Сучасні SQL-бази даних можуть масштабуватись відповідно до зростання кількості користувачів та обсягів даних, що забезпечить стабільну роботу системи навіть при збільшенні навантаження та обсягу клієнтів.

6. Звітність. Користувачі матимуть змогу отримувати звіти про поточний період у зручному форматі у зручний для них час на пошту або у вигляді завантаженого текстового файлу.

1.3 Методи вирішення поставлених задач

Впровадження особистого кабінету споживача електроенергії потребує використання різноманітних методів та підходів для забезпечення його ефективності, зручності та особливо безпеки. У сучасному світі наявність у надавача послуг особистого кабінету для клієнтів є невід'ємною частиною взаємодії та розвитку. Розглянемо основні методи, що використовуються для вирішення поставлених задач, пов'язаних із створенням та функціонуванням такого кабінету.

З метою визначення потреб і очікувань різних категорій споживачів часто проводяться опитування та інтерв'ю, іноді мобільні додатки та веб-застосунки самостійно запитують у користувачів зворотній зв'язок стосовно якості наданих послуг. Це дозволяє зібрати цінну інформацію про функції та сервіси, які користувачі хотіли б бачити в особистому кабінеті та допомагає актуалізувати та модернізувати продукт.

Дослідження вже існуючих рішень на ринку або у галузі дасть можливість визначити сильні та слабкі сторони конкурентних продуктів, що допоможе уникнути типових помилок та запозичити кращі практики.

Важливою частиною процесу розробки є створення такої архітектури, яка дозволить легко розширювати та модернізувати систему. Це забезпечить гнучкість у додаванні нових функцій та сервісів.

У перспективі майже будь-який особистий кабінет потребує інтеграції з іншими системами постачальника електроенергії та системами оплати полуг онлайн.

Для інтерфейсу користувача (UI) потрібно створити інтуїтивно зрозумілий та візуально привабливий інтерфейс, що забезпечить зручний доступ до всіх функцій особистого кабінету. Використання сучасних фреймворків та бібліотек для розробки адаптивного дизайну є необхідністю, особливо в умовах високої конкуренції сучасного простору.

Розробка основних функціональних модулів, таких як перегляд та аналіз даних про споживання, онлайн-оплата рахунків, налаштування повідомлень та індивідуальних тарифів.

Зберігання та управління даними за допомогою використання реляційної бази даних MySQL забезпечить ефективне зберігання та управління великими обсягами даних про споживання електроенергії, рахунки, користувачів та інші сутності системи, навіть якщо згодом база даних потребуватиме розширення.

Завдяки потужним можливостям SQL, MySQL дозволить виконувати складні запити для отримання аналітичної інформації, що є важливим для генерації звітів, аналізу трендів споживання та інших бізнес-аналітичних задач, з якими можуть стикнутися надавачі послуг.

Також важливим є те, що MySQL пропонує вбудовані механізми безпеки, включаючи шифрування даних, контроль доступу та управління правами користувачів, що забезпечує надійний захист конфіденційної інформації споживачів.

При створенні особистого кабінету важливим кроком є побудова серверної та клієнтської частини веб-застосунку. Серверна частина відповідатиме за обробку запитів користувачів, управління даними та логікою додатка. Для цього використовуються сучасні фреймворки та технології, такі як Node.js з Express, Java з Spring Boot тощо.

Клієнтська частина веб-застосунку забезпечує інтерфейс користувача і взаємодію з сервером через API. Для її реалізації використовуються фреймворки

та бібліотеки, такі як React, що забезпечать створення динамічних, швидких та зручних у використанні інтерфейсів.

Аутентифікація та авторизація користувачів використовуються для захисту облікових записів користувачів. Слід реалізувати гнучку систему управління правами доступу та застосувати сучасні методи шифрування для захисту даних під час їх передачі та зберігання.

Використання автоматизованих та ручних методів тестування для забезпечення високої якості продукту, це включає в себе функціональне тестування, тестування безпеки та навантажувальне тестування.

Важливим є також аналіз продуктивності системи та оптимізація кодової бази для забезпечення швидкої та стабільної роботи особистого кабінету навіть при високих навантаженнях.

При створенні програмної системи варто думати також і про її подальший розвиток, тому впровадження механізмів збору зворотного зв'язку від користувачів для постійного покращення системи та регулярні оновлення на основі відгуків та нових вимог ринку буде на користь, як і розробка та впровадження нових функцій і сервісів, що відповідатимуть сучасним тенденціям та потребам користувачів.

На етапі планування роботи варто звернути особливу прискіпливу увагу на вибір програмних засобів для реалізації веб-застосунку, щоб він базувався на сучасних технологіях, що забезпечать високу продуктивність, безпеку та зручність використання. Сумісність і адаптивність до платформ, браузерів та пристроїв є ключовим питанням в охопленні аудиторії, тому це включає в себе вибір фреймворків для фронтенду та бекенду, бази даних, за потреби системи контейнеризації та оркестрації тощо.

Обрані засоби повинні легко інтегруватися з вже існуючими системами та забезпечувати можливість масштабування, щоб відповідати зростаючим потребам та вибагливості користувачів та новим викликам ринку.

Розгляд та використання усіх цих методів дозволить створити ефективний, надійний та зручний веб-застосунок особистого кабінету споживача

електроенергії, який матиме на меті відповідати високим стандартам якості та прагнучиме задовольнити потреби різних категорій користувачів.

2 ПРОБЛЕМАТИКА ТА ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

Сучасні технології постійно відкривають нові можливості для підвищення ефективності та зручності у сфері постачання та споживання електроенергії. Одним із таких інноваційних рішень є особистий кабінет споживача електроенергії, який дозволяє користувачам отримувати доступ до важливої інформації, керувати своїм споживанням та оптимізувати витрати. Однак, впровадження таких систем супроводжується рядом проблем і викликів, які потребують детального аналізу та вирішення. Серед основних проблем, з якими стикаються розробники та користувачі особистих кабінетів для споживачів електроенергії, можна виділити питання безпеки та конфіденційності даних, забезпечення високої продуктивності та надійності системи, інтеграції з існуючими інформаційними системами постачальників електроенергії, а також необхідність забезпечення зручного та інтуїтивно зрозумілого інтерфейсу для різних категорій користувачів. Зважаючи на зазначені обставини важливо проаналізувати вже існуючі рішення, що є на ринку, та які надають аналогічні або схожі за характером послуги. Огляд існуючих рішень дозволить виявити їх сильні та слабкі сторони, врахувати найкращі практики та уникнути типових помилок при розробці власного продукту.

2.1 Проблематика впровадження веб-застосунків особистих кабінетів для споживачів енергоресурсів в Україні

Впровадження веб-застосунків особистих кабінетів для споживачів електроенергії є важливим кроком на шляху до цифровізації енергетичного сектору України. Ці системи дозволяють споживачам мати зручний доступ до інформації про своє енергоспоживання, оплачувати рахунки онлайн, отримувати рекомендації щодо енергозбереження та, у перспективі, багато інших зручних

функцій. Однак, процес впровадження таких кабінетів в Україні стикається з низкою складностей, які потребують комплексного підходу для їх вирішення.

На заваді може стати низький рівень цифрової грамотності серед певної частини населення. Багато споживачів, особливо літніх людей або людей, що живуть у віддалених чи сільських районах, та мають обмежений доступ до інтернету або недостатньо знань для ефективного використання онлайн-сервісів. Це може ускладнити масове впровадження та використання особистих кабінетів на постійній основі, а тому вимагає додаткових заходів для навчання та підтримки користувачів.

При створенні та впровадженні особистих кабінетів для клієнтів у сфері надання послуг слід звертати увагу на розв'язання таких важливих задач:

Безпека даних в особистих кабінетах завжди збирають та зберігають велику кількість персональних даних клієнтів, зокрема це адреси проживання, електронна пошта, номер телефону, ПІБ користувача та інші дані відповідно до потреб, тому, щоб уникнути витоку даних та шахрайства, особисті кабінети повинні мати високий рівень захисту не тільки з боку користувача, у вигляді складного пароля та двофакторної автентифікації, а і впровадженням захисту серверів від сторонніх кібератак.

Інтероперабельність у застосунків, що надають широкий спектр комплексних послуг, може виникнути проблема сумісності з деякими платформами або версіями приладів чи браузерів; якщо до роботи залучено інші сервіси із надання додаткових послуг, можуть виникнути конфлікти, яких можна уникнути в процесі розробки, забезпечуючи адаптивність та сумісність застосунку.

Зручність користування веб-застосунки не завжди мають зрозумілий та інтуїтивний інтерфейс, що може стати на заваді в користуванні деяких груп громадян, наприклад людей похилого віку, людей із порушенням зору або молодих людей, що є користувачами-початківцями чи мають низький рівень навичок користування цифрових технологій, і тому потребують простого інструментарію. Натомість користувачі отримують скорочення кількості фізичних

візитів до центру надання послуг, віддалений доступ до контролю власних ресурсів та повне або часткове ведення документації онлайн.

Технічні можливості можуть стати проблемою в роботі можуть стати якість та надійність техніки, що використовується з боку розробників для надання послуг, причинами чого можуть бути недостатнє фінансування, застарілі методи роботи або технічні рішення або несумісність розробки із технічними можливостями великої кількості користувачів.

Прозорість та контроль у роботі з клієнтами важливо забезпечити довіру та впевненість користувачів у збереженні їх особистої інформації та уникненні помилок, при обробці даних; надання можливості зворотнього зв'язку між клієнтами та надавачами послуг, щоб клієнт мав змогу отримати відповідь на свій запит у короткий проміжок часу у зручному для нього форматі, наприклад за номером телефону, у чат месенджеру або на електронну пошту.

Впровадження інноваційних рішень в умовах високої конкуренції, швидкому розвитку ринку надання послуг та активного розвитку технічних можливостей, зокрема в сфері штучного інтелекту та сентимент-аналізу, важливо впроваджувати перевірені інноваційні рішення у продукт, щоб підтримувати власну конкурентноспроможність та підвищувати ефективність роботи продукту.

2.2 Огляд існуючих рішень

В Україні вже деякий час широко використовується та стрімко розвивається культура впровадження цифрових технологій, зокрема веб-застосунків та мобільних додатків різними компаніями для надання громадянам послуг та скорочення бюрократії, яка завжди супроводжувала процес роботи із послугами з обох сторін. Автоматизація процесів обміну документацією під час платіжних періодів електроспоживання значно спростила та пришвидшила процес вирішення важливих питань.

2.2.1 Особистий кабінет для приватних клієнтів Yasno

Yasno це велика українська компанія-постачальник електроенергії і газу, чию головну сторінку можна побачити на рисунку 2.1. Він пропонує приватним і юридичним клієнтам рішення з енергоефективності, підтримку для електромобілістів та інші послуги [1]. Також Yasno надають у своєму веб-застосунку достатньо широкий функціонал рішень питань з електропостачань, що є особливо актуально в умовах появи дефіциту електроенергії в Україні.

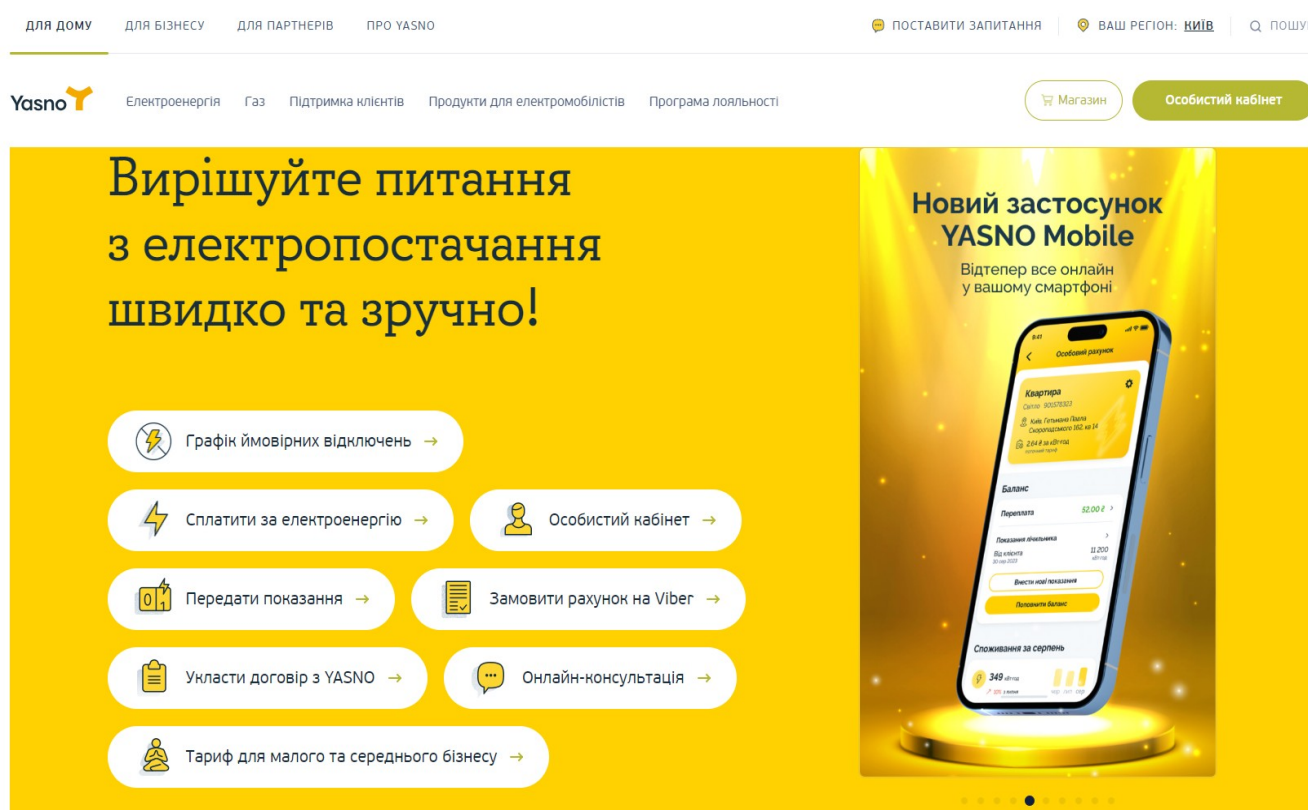
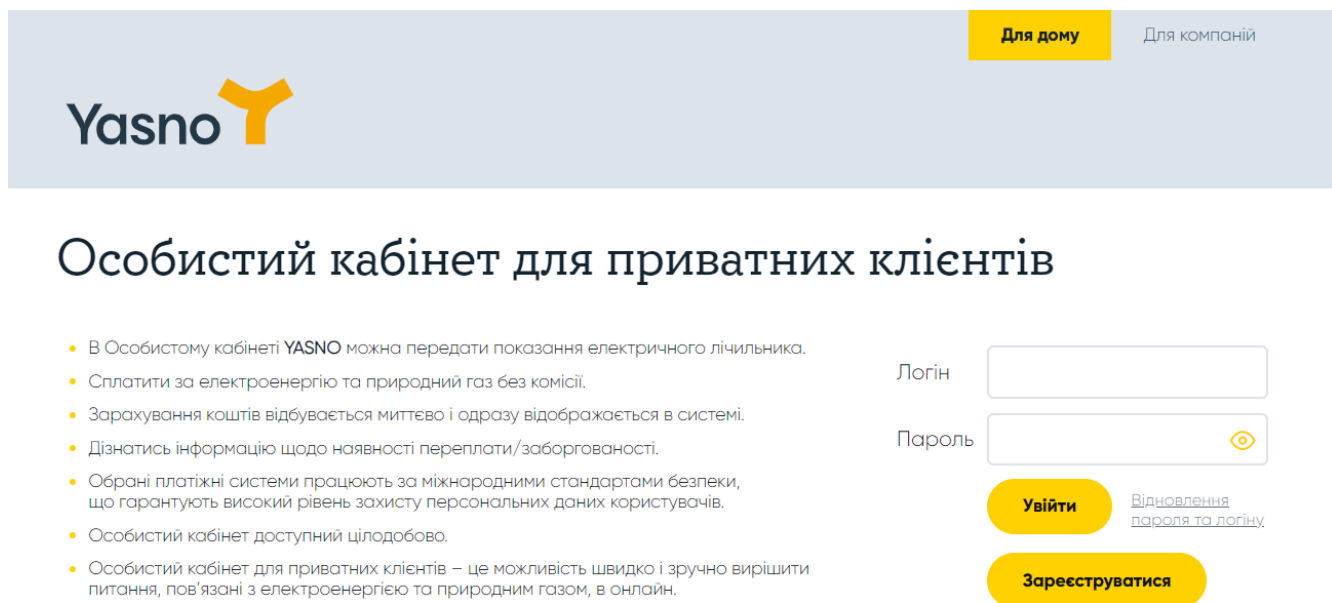


Рисунок 2.1 – Головна сторінка офіційного сайту Yasno [1]

Серед багатьох функцій, що надає сервіс Yasno розглянемо саме особистий кабінет, що є розмежованим для приватних клієнтів та юридичних осіб, оскільки вони потребують різного підходу.

На рисунку 2.2 можна побачити сторінку входу в особистий кабінет для приватних клієнтів:



Yasno

Для дому Для компаній

Особистий кабінет для приватних клієнтів

- В Особистому кабінеті **YASNO** можна передати показання електричного лічильника.
- Сплатити за електроенергію та природний газ без комісії.
- Зарахування коштів відбувається миттєво і одразу відображається в системі.
- Дізнатись інформацію щодо наявності переплати/заборгованості.
- Обрані платіжні системи працюють за міжнародними стандартами безпеки, що гарантують високий рівень захисту персональних даних користувачів.
- Особистий кабінет доступний цілодобово.
- Особистий кабінет для приватних клієнтів – це можливість швидко і зручно вирішити питання, пов'язані з електроенергією та природним газом, в онлайн.

Логін

Пароль

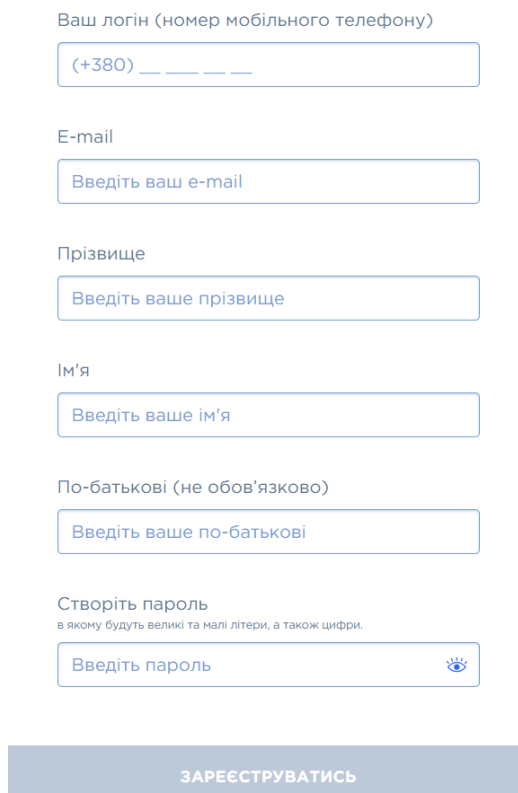
[Відновлення пароля та логіну](#)

Увійти

Зареєструватися

Рисунок 2.2 – Особистий кабінет для приватних клієнтів Yasno [1]

Розглянемо форму реєстрації на сайті Yasno, яку зображено на рисунку 2.3. У клієнта при реєстрації запитують номер мобільного телефону та адресу електронної пошти, також передбачено, що поле по-батькові не є обов'язковим, і також можна бачити які умови є до складності пароля.



Ваш логін (номер мобільного телефону)

(+380) _ _ _ _ _

E-mail

Введіть ваш e-mail

Прізвище

Введіть ваше прізвище

Ім'я

Введіть ваше ім'я

По-батькові (не обов'язково)

Введіть ваше по-батькові

Створіть пароль
в якому будуть великі та малі літери, а також цифри.

Введіть пароль 

ЗАРЕЄСТРУВАТИСЬ

Рисунок 2.3 – Форма реєстрації на сайті Yasno [1]

Розглянувши існуюче рішення у вигляді особистого кабінету Yasno, можна виділити такі характерні особливості та переваги сервісу, як:

- Завдяки офіційному статусу Yasno має можливість подання та отримання усіх важливих документів, використовуючи для цього електронний підпис.
- Залучення у співпрацю великої кількості платіжних систем, а отже і можливість швидко і зручно оплачувати послуги електроспоживання, що гарантовано відображатимуться у системі.
- Можливість побачити у системі інформацію щодо переплат або заборгованостей, включаючи показники лічильників для прозорості процесу.
- Наявність консультантів та техпідтримки, по чій допомозі можна звернутися у повідомленнях онлайн або за номеру телефону гарячої лінії, завдяки

чому клієнти не залишаються без підтримки кваліфікованих осіб та отримують відповідь на свої питання в індивідуальному порядку.

2.2.2 Особистий кабінет побутового споживача ПрАТ «ДТЕК Київські регіональні електромережі»

ПрАТ «ДТЕК Київські регіональні електромережі» є компанією, що обслуговує понад 1 мільйон споживачів електроенергії [2]. Веб-застосунок особистого кабінету “ДТЕК Київські регіональні електромережі” має особисті кабінети для побутових клієнтів, юридичних клієнтів та постачальників послуг. У ході перегляду наявних існуючих рішень, розглянемо оформлення особистого кабінету для побутових клієнтів. Переглянути сторінку з особистими кабінетами можна на рисунку 2.2.

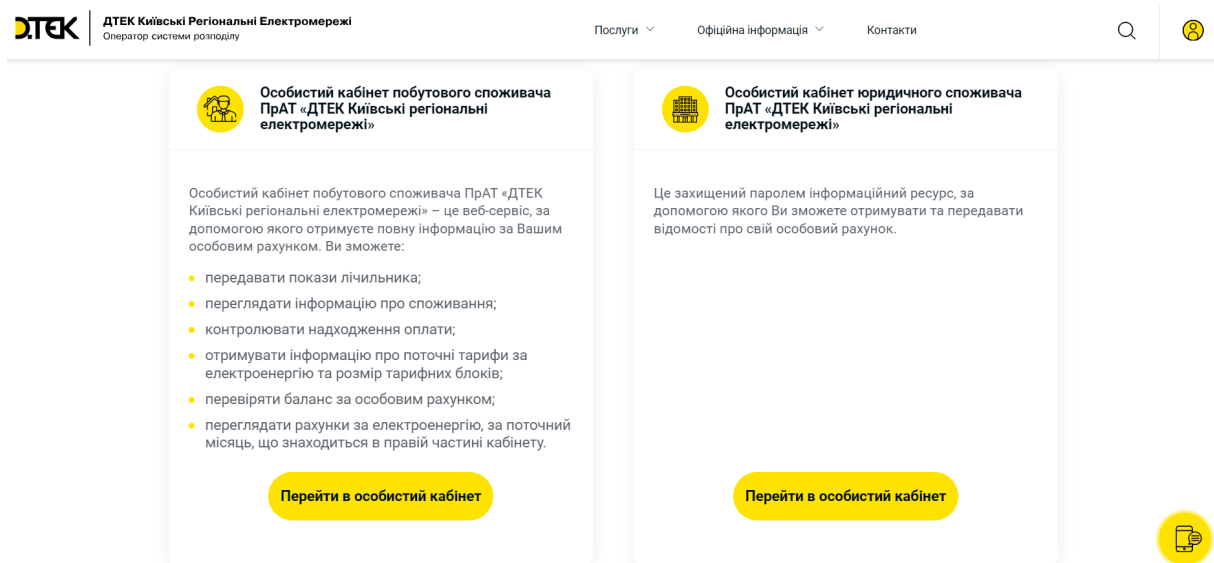


Рисунок 2.4 – Особистий кабінет побутового та юридичного споживачів ПрАТ “ДТЕК Київські регіональні електромережі” [2]

Перевагами веб-застосунку є:

- Зрозумілий інтерфейс, яким зможе скористуватись як новачок у сфері послуг, так і досвідчені користувачі, яким буде важливо слідкувати за межами власного електроспоживання.

- Чітка розмежованість клієнтів із різними потребами, а саме побутових клієнтів, юридичних та постачальників послуг.

- Структурованість широкого інструментарію, який розрахований на користувачів з різними потребами та запитами.

- Наявність перевіреної довіреної платіжної системи у застосунку.

- Можливість отримання звіту за поточний платіжний період та внесення даних з лічильників.

2.2.3. Особистий кабінет споживача енергоресурсів iEnergо

Серед веб-застосунків, що надають схожий пакет послуг, є особистий кабінет споживача енергоресурсів iEnergо, що базується у Хмельницькій області, та вікно входу в кабінет споживача можна побачити на рисунку 2.5. «Особистий кабінет споживача електроенергії» є точкою доступу до персоніфікованої інформації і призначений для надання сервісу з віддаленої роботи із оперативними даними про Особовий рахунок Споживача, в тому числі про платежі, що надійшли; показники лічильників; виставлення рахунків та оплати спожитої електричної енергії [3].

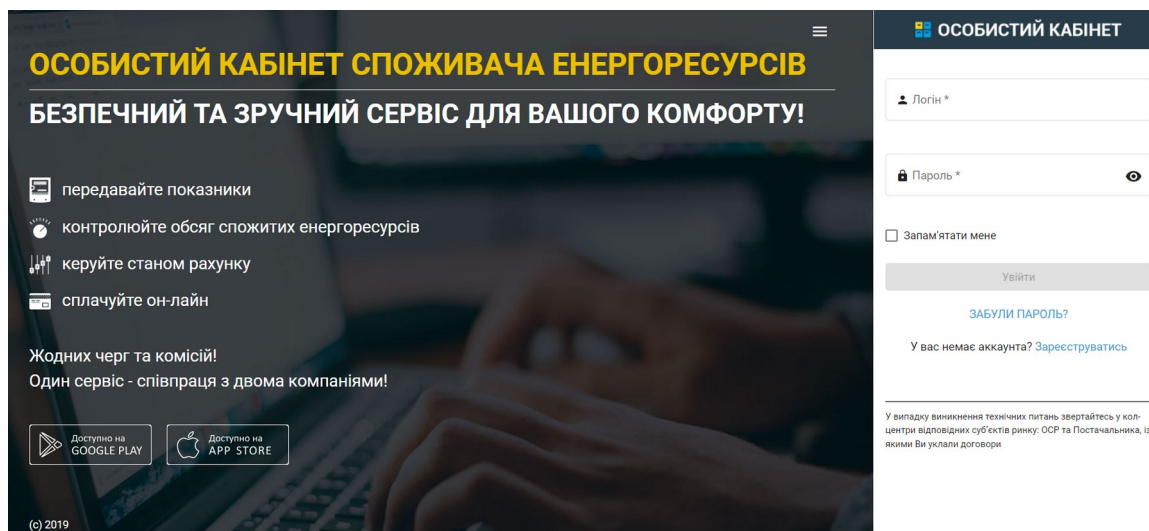


Рисунок 2.5 – Особистий кабінет споживача енергоресурсів iEnergo [3]

Серед переваг веб-застосунку iEnergo можна виділити такі:

- Розробник одразу надає псилання на мобільні додатки у Google Play та App Store, що свідчить про широке коло користувачів, які зможуть користуватись послугами незалежно від доступної їм платформи.
- Можливість передавати показники з лічильників одразу в особистому кабінеті веб-застосунку.
- iEnergo також підтримує оплату послуг онлайн без комісій.

3 ЗАСОБИ РОЗРОБКИ ПРОГРАМНОГО ЗАСТОСУНКУ

Розробка програмного забезпечення для особистих кабінетів споживачів електроенергії є комплексним процесом, що потребує використання сучасних технологій та інструментів. Успішність значною мірою залежить від правильного вибору засобів розробки, які забезпечують високу продуктивність, масштабованість, безпеку та зручність використання кінцевими користувачами. Сучасна розробка веб-застосунків включає в себе кілька важливих аспектів: побудову серверної та клієнтської частин, управління даними, забезпечення безпеки та продуктивності системи, а також інструменти для тестування та підтримки. Кожен з цих аспектів вимагає використання спеціалізованих технологій та засобів, які найкраще відповідають вимогам проекту.

3.1 Архітектура програмного забезпечення

Для виконання поставленої задачі було обрано клієнт-серверну архітектуру. Вона дозволяє розподілити логіку між клієнтом та сервером, спрощуючи процес розробки і тестування. Така архітектура забезпечує усі потреби веб-застосунку і дозволяє впровадити його у широке використання.

Така архітектура дозволяє розділити завдання між клієнтською частиною (front-end) і серверною частиною (back-end), що забезпечує масштабованість, гнучкість і покращену продуктивність застосунку [4].

Перевагами, які забезпечує клієнт-серверна архітектура, можна вважати:

Можливість масштабувати окремо клієнтську і серверну частини. Це означає, що якщо збільшиться кількість користувачів, то можна збільшити й потужність сервера, не змінюючи при цьому клієнтську частину. Розмежовані процеси обслуговування клієнтів і серверу значною мірою полегшують обслуговування системи, а також, за необхідності, стане в нагоді у ситуації, коли

система потребуватиме оновлень, впровадження нових функцій або проведення ремонтних робіт.

Розділення на front-end і back-end забезпечує для розробників можливість гнучко й незалежно вносити оновлення або змінювати частину коду, не впливаючи на іншу [5].

Серверна частина відповідає за виконання основних функцій програми, обробку запитів та виконання таких операцій, як аутентифікація та авторизація користувачів, зберігання та обробка даних, читання та запис у базах даних. Завдяки тому що клієнтська частина не має прямого доступу до даних це гарантує певний рівень безпеки.

Сервер бере на себе процеси виконання важливих та важкоємних завдань, звільняючи клієнтський пристрій від деякої частини навантаження. Це дозволяє не скорочувати кількість можливих користувачів із менш потужними пристроями, адже важливо охопити якнайбільшу кількість публіки та забезпечити комфортні умови користування веб-застосунком.

У процесі створення та побудови сервера важливо було обрати універсальні інструменти за засоби, тому до вибору технологій було здійснено відповідальний підхід. Для реалізації клієнтської частини основною мірою було використано JavaScript та React, для серверної частини Spring Boot, для взаємодії між клієнтом та сервером API, HTTP протоколи, для зберігання даних реляційну базу даних MySQL. Для створення серверної частини було налаштовано середовища розробки: встановлено Java Development Kit (JDK) [6] та Spring Boot [7].

Налаштування підключення до бази даних, а саме вказання параметрів підключення до MySQL у файлі конфігурації application.properties можна побачити на рисунку 3.1.

```
spring.datasource.url=jdbc:mysql://localhost:3306/electricity_consumers
spring.datasource.username=root
spring.datasource.password=password
spring.jpa.hibernate.ddl-auto=update
```

Рисунок 3.1 – Вказання параметрів підключення до MySQL у файлі конфігурації application.properties [7]

Створення моделей даних для користувачів та їхніх даних у вигляді Java класів з анотаціями JPA [8], приклад можна побачити на рисунку 3.2.

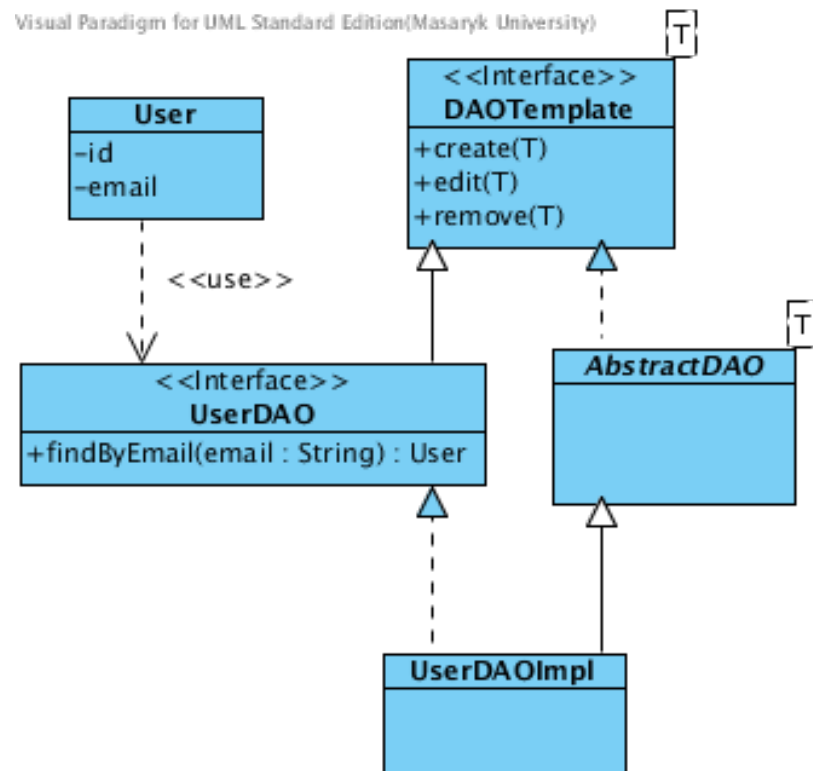


Рисунок 3.2 – Створення моделей даних у вигляді Java класів з анотаціями JPA DAO [26]

Створення REST API [9] із визначенням контролерів для обробки HTTP-запитів від клієнта. Першим кроком є створення контролерів. `UserController.java`

відповідає за обробку запитів, пов'язаних з користувачами. У цьому контролері визначаються методи для різних типів запитів: GET, POST, PUT та DELETE [9]. Ці методи дозволяють створювати нових користувачів, отримувати інформацію про них, оновлювати їхні дані та видаляти користувачів.

Створення клієнтської частини складається з ініціалізації проекту, налаштування структури проекту та взаємодії з API. На етапі ініціалізації проекту створюємо новий React проект, використовуючи Create React App [10]. Для налаштування структури проекту творення компонентів та налаштування маршрутизації для навігації між різними сторінками застосунку. Для взаємодії з API використовується бібліотека Axios [11] для надсилання HTTP-запитів до серверного API.

Архітектура системи організована таким чином, як зображено на рисунку 3.4.

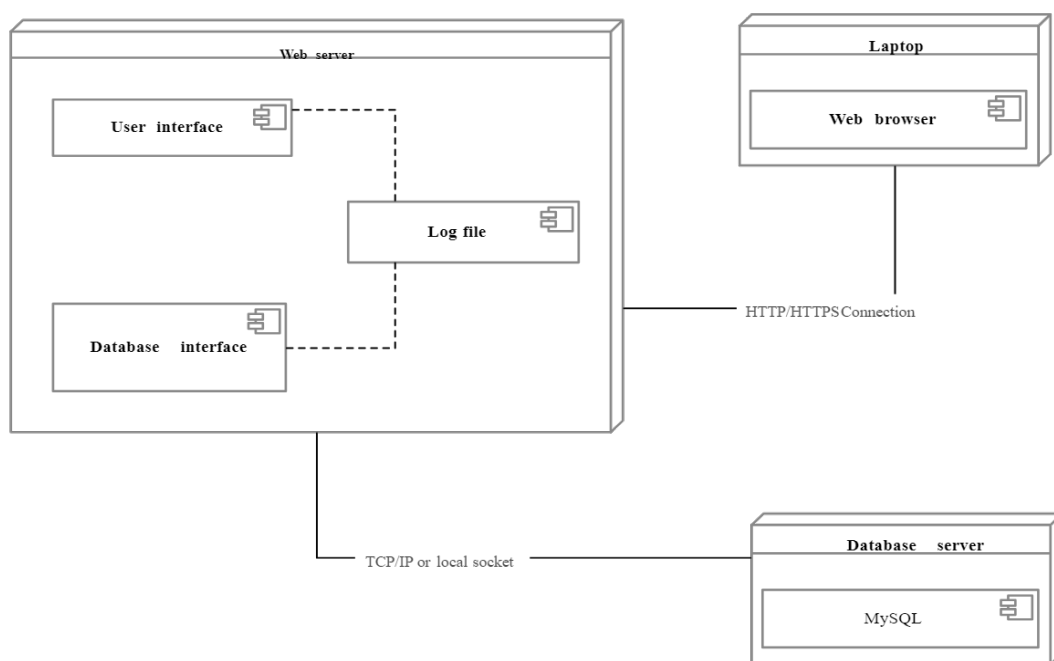


Рисунок 3.4 – Клієнт-серверна архітектура

Використання клієнт-серверної архітектури разом з такими технологіями, як JavaScript, React, Spring Boot, та MySQL, дозволяє створити ефективний,

масштабований і безпечний веб-застосунок для споживачів електроенергії. Цей підхід забезпечує гнучкість у розробці та підтримці застосунку, дозволяючи легко адаптувати його до змінних вимог та збільшення кількості користувачів.

3.2 Розробка бази даних

Для розробки особистого кабінету споживача електроенергії було обрано реляційну базу даних. Реляційні бази даних забезпечують високу цілісність даних завдяки використанню ключів, зв'язків і обмежень, що дозволяє уникнути дублювання та непослідовності даних. Вони також підтримують ACID-властивості (атомарність, узгодженість, ізоляція, довговічність) [12], що гарантує надійність та безпеку транзакцій. Завдяки реляційній моделі, дані організовуються в таблиці, що дозволяє створювати складні та ефективні запити за допомогою мови SQL, забезпечуючи високу гнучкість в отриманні та обробці інформації.

MySQL було обрано як систему керування базами даних завдяки її численним перевагам. По-перше, MySQL є відкритим програмним забезпеченням, що дозволяє знизити витрати на розробку та підтримку. Вона має широку підтримку та використовується у багатьох проектах, що забезпечує доступ до великої кількості ресурсів та спільнот [13]. MySQL відома своєю високою продуктивністю та здатністю обробляти великі обсяги даних, що є важливим для систем, які потребують обробки великої кількості транзакцій і даних. Крім того, MySQL пропонує надійні механізми безпеки, включаючи аутентифікацію користувачів, шифрування та контроль доступу, що критично важливо для захисту конфіденційних даних споживачів.

В особистому кабінеті споживача електроенергії використовується кілька таблиць, які взаємодіють між собою за певною логікою. Таблиця Users зберігає інформацію про користувачів, включаючи їх імена, прізвища, номери телефонів, електронні адреси, паролі та номери клієнтів. Кожен користувач може мати одну

або кілька адрес, які зберігаються в таблиці `Addresses`. Кожна адреса пов'язана з користувачем за допомогою зовнішнього ключа `UserID`.

Лічильники електроенергії зберігаються в таблиці `Meters`, кожен з яких прив'язаний до конкретної адреси за допомогою зовнішнього ключа `AddressID`. Це дозволяє зберігати інформацію про встановлення лічильників та їх типи. Платіжні періоди для кожного користувача зберігаються в таблиці `PayPeriods`, де кожен платіжний період пов'язаний з користувачем за допомогою зовнішнього ключа `UserID`.

Щоденне споживання електроенергії записується в таблиці `DailyUsage`, де кожен запис прив'язаний до конкретного платіжного періоду за допомогою зовнішнього ключа `PeriodID`. Це дозволяє зберігати дані про щоденне споживання електроенергії для кожного платіжного періоду. Нарешті, щомісячні підсумки споживання електроенергії за рік зберігаються в таблиці `YearlySummary`, де кожен запис пов'язаний з користувачем за допомогою зовнішнього ключа `UserID`.

Типи даних для кожного поля були обрані з урахуванням типу інформації, яка зберігається. Наприклад, імена та прізвища зберігаються у форматі `VARCHAR`, оскільки вони можуть містити текстові дані різної довжини. Дані про дати та час зберігаються у форматі `DATE`, а числові значення, такі як споживання електроенергії, зберігаються у форматі `DECIMAL`, що дозволяє точно зберігати числа з плаваючою точкою.

ER-діаграму створеної бази даних під назвою `ElectricityBilling` можна побачити на рисунку 3.5.

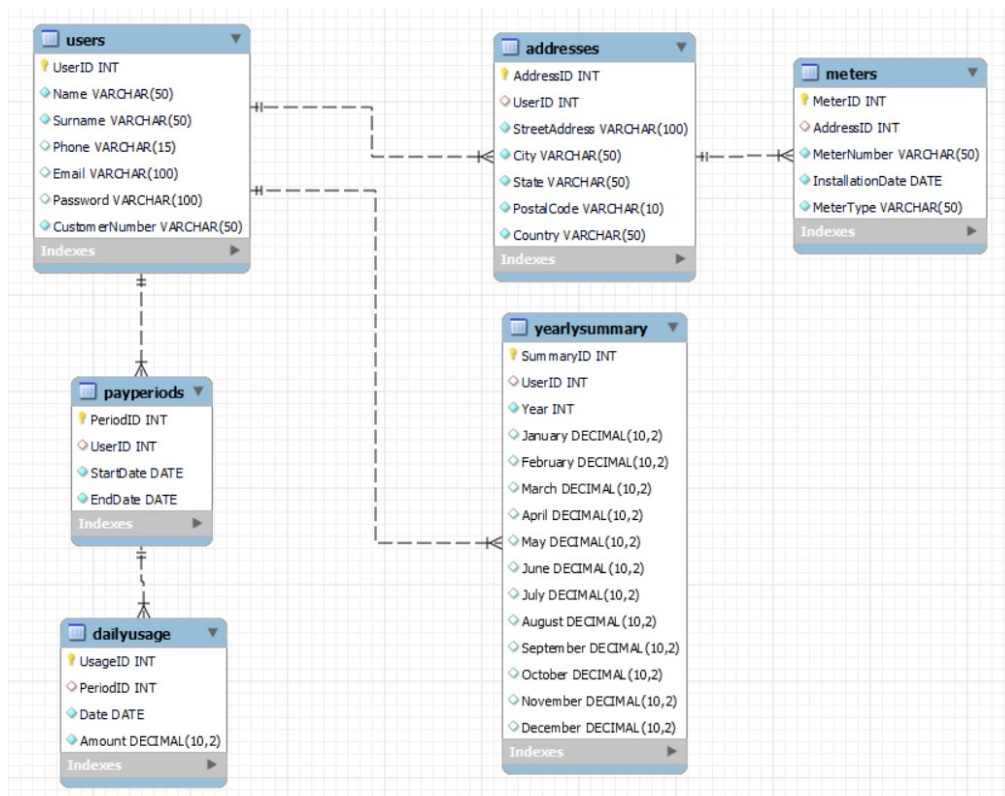


Рисунок 3.5 – База даних ElectricityBilling

Загальна структура бази даних забезпечує логічну цілісність даних, де кожен лічильник прив'язаний до конкретної адреси, а кожна адреса - до користувача. Це дозволяє легко організовувати та запитувати дані, забезпечуючи ефективну роботу особистого кабінету споживача електроенергії.

3.3 Інтегроване середовище розробки (IDE)

На стадії планування веб-застосунку для реалізації було обрано Visual Studio Code (VSC) [14], як основне середовище для роботи. Вибір даного редактору базувався на переліку таких переваг:

VSC позиціонується як зручний інструмент з інтуїтивним інтерфейсом для розробників, за допомогою якого можна швидко працювати з проектами різної

складності, при цьому редактор автоматично підлаштується під розширення відкритого файлу, а також сам редактор не потребує від розробника високих характеристик обладнання.

VSC підтримує широку низку мов програмування, серед яких є всі основні найпоширеніші популярні мови, зокрема ті, що було обрано для реалізації веб-застосунку, а саме: HTML [15], CSS [15], JavaScript [19], Node.js [19] та бази даних SQL [20].

VSC встановлюється із вбудованим налагоджувачем для Node.js [19], що значно прискорює та полегшує процес редактури коду, а при бажанні може бути вимкненою, що не буде конфліктувати із налагоджувачами для інших середовищ програмування, що можуть бути встановлені розробником додатково пізніше.

У редакторі доступний великий перелік різних безкоштовних розширень, що можуть суттєво вплинути на процес роботи та досвід, зокрема це розширення для дебагінгу, редагування коду або додаткові можливості інтерфейсу, що дозволяють максимально налагодити кастомізацію робочої області під вимоги розробника.

VSC підтримує різні платформи, як і найрозповсюдженіші Windows, Linux и macOS. Універсальність на невибагливість робить процес роботи швидким та комфортним для розробника.

Тут надається можливість використовувати кількість команд, які роблять процес програмування комфортнішим та пришвидшує процес не лише на початку роботи, але і в кінці, на етапі налагодження, зокрема Go to Definition, Peek Definition, Find all References, що у поєднанні із візуальними виділенням та супроводженням значно полегшують редактуру.

3.4 Мова програмування

Вибір мов програмування та відповідних бібліотек є одним із ключових аспектів розробки програмного забезпечення, особливо коли йдеться про

створення складних і багатофункціональних систем, таких як особистий кабінет споживача електроенергії. Правильний вибір технологій дозволяє забезпечити високу продуктивність, надійність, безпеку та зручність використання застосунку.

Для розробки веб-застосунку особистого кабінету споживача електроенергії було виділено функціонал таких мов програмування як HTML[15], CSS [15], JavaScript [19], Node.js [16], MySQL [13], React [10] та Spring Boot [7]. Вони є популярними та розповсюдженими, використовуються у переважній більшості вже існуючих програмних рішень, а тому є перевіреним та надійним вибором. У сукупності дані мови програмування дозволяють створити гнучкий веб-застосунок, що задовольнить потребу сучасних користувачів та матиме потенційну можливість отримати розвиток зі збільшенням клієнтської бази у майбутньому.

3.4.1 Реалізація frontend

Розглянемо детальніше переваги обраних технологій:

HTML (HyperText Markup Language) [15] є основною мовою розмітки для створення веб-сторінок, що забезпечує стандартизовану сумісність з усіма браузерами, а також дозволяє структурувати контент таким чином, щоб він був зручним для читання і розуміння як користувачам, так і пошуковим системам.

CSS (Cascading Style Sheets) [15] забезпечує можливість гнучкого і ефективного оформлення веб-сторінок, надаючи їм привабливого вигляду, а також дозволяє розділити структуру документа (HTML) від його візуального оформлення, що спрощує підтримку, оновлення дизайну та введення та оформлення нових функцій.

JavaScript [15] дозволяє додати інтерактивні елементи на веб-сторінки, роблячи їх більш динамічними і зручними у використанні, також JavaScript підтримується всіма сучасними браузерами, що забезпечує його широке застосування для створення веб-додатків.

3.4.2 Реалізація backend

Обираючи технологічний стек для реалізації backend було розглянуто та обрано такі засоби:

Node.js [16], [19], що забезпечує високу серверну продуктивність завдяки асинхронній обробці запитів, що дозволяє обробляти велику кількість одночасних з'єднань. Єдина мова для клієнта і сервера, адже використовується як на стороні клієнта, так і на стороні сервера, що спрощує процес розробки і підтримки застосунку.

SQL (Structured Query Language) [13]: Надає ефективне управління даними, SQL є стандартною мовою для управління і маніпулювання даними в реляційних базах даних. Дозволяє створювати складні запити для вибірки, оновлення і видалення даних, забезпечуючи ефективне зберігання та доступ до інформації.

Бекенд-фреймворк Spring Boot [7] є одним з найпопулярніших та найпотужніших фреймворків для розробки веб-додатків. Він забезпечує велику кількість готових компонентів та бібліотек, що спрощує розробку та забезпечує високу продуктивність. Приклад архітектури веб-застосунку побудованого за допомогою цього фреймворку можна побачити на рисунку 3.6.

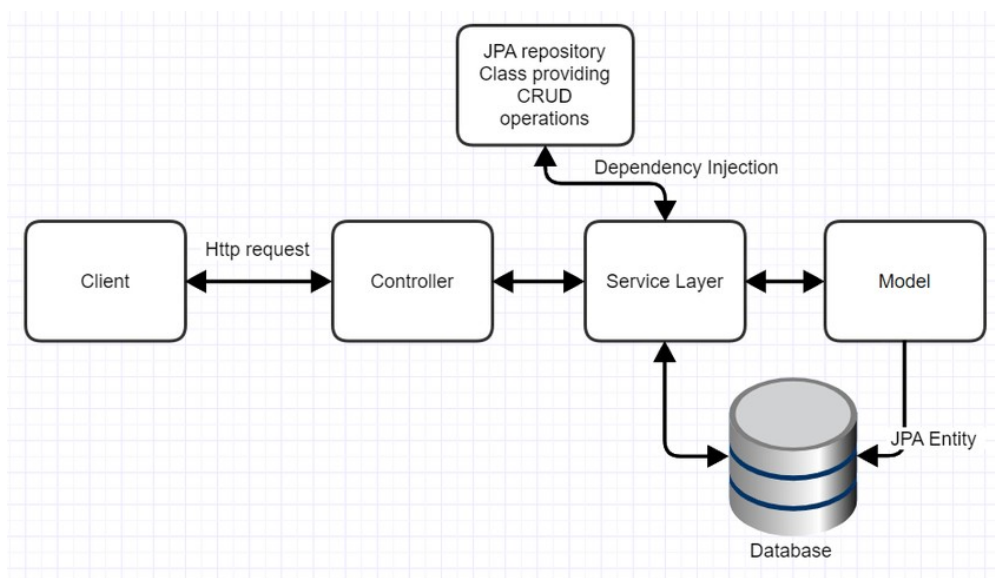


Рисунок 3.6 – Архітектура веб-застосунку побудованого за допомогою фреймворку Spring Boot [22]

Інструменти для роботи з базою даних, а саме Spring Data JPA [8], який є частиною Spring Framework та дозволяє просто взаємодіяти з базою даних за допомогою Java Persistence API (JPA) [8] і об'єктно-реляційного відображення (ORM) [23]. Цей інструмент спрощує взаємодію з базою даних, дозволяючи писати менше коду для виконання рутинних операцій з базою даних.

HTTP протокол для взаємодії сервера та клієнта, адже він є універсальним, працює на всіх пристроях, не потребує додаткових налаштувань, а також є достатньо швидким.

Для взаємодії з сервером використовується Axios [11], потужна та популярна бібліотека для виконання HTTP-запитів з JavaScript, яка забезпечує простий та зручний інтерфейс для взаємодії з сервером.

3.5 Бібліотеки та розширення

Для покращення ефективної роботи з HTML, CSS, JavaScript, Node.js та SQL у Visual Studio Code (VSC) [14] необхідно було встановити декілька корисних бібліотек і розширень. Серед них є:

- HTML CSS Support для додаткових функцій автозаповнення та підсвічування синтаксису для HTML та CSS.
- Live Server для запуску локального сервера для перегляду внесених змін у реальному часі під час розробки веб-застосунку.
- Node.js Extension Pack, що надає набір розширень, включаючи підтримку Node.js, npm та інших інструментів для розробки на Node.js.
- SQLTools, що є потужним інструментом для роботи з базами даних, який підтримує різні СУБД (системи управління базами даних), включаючи MySQL, PostgreSQL та інші.
- Vscode-database що є розширенням для підключення до баз даних та виконання SQL-запитів безпосередньо в середовищі VS Code.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ

Головною метою розробки особистого кабінету споживача електроенергії є створення такої програмної системи, яка за допомогою простого та зручного інтерфейсу надасть можливість користувачу скористатись повним функціоналом веб-застосунку. У цьому розділі буде розглянуто структуру проекту, включаючи ключові компоненти та технології, що використовуються для його реалізації. Кожен з модулів відіграє важливу роль у забезпеченні функціональності, надійності та безпеки системи. Буде детально розглянуто кожен з компонентів та їх взаємодію та внесок у загальну архітектуру проекту.

4.1 Загальна структура проекту

Структура проекту складається з основних каталогів `src`, `public`, `views`. У каталозі `src` містяться файли для серверної логіки, підключення до бази даних та моделей даних. У каталозі `public` розміщено статичні файли. У каталозі `views` розміщено шаблони для відображення сторінок на стороні клієнта. Важливо підтримувати чітку організацію структури проекту, адже це дозволяє розробникам легко орієнтуватися у коді не лише на етапі створення програмного забезпечення, а й підтримувати та розширювати функціональність застосунку, забезпечуючи високу продуктивність та надійність системи.

4.1.1 Основний каталог для серверної логіки

Для побудови серверної логіки великою мірою було використано фреймворк Spring Boot, який надає зручні інструменти для створення REST API [9], обробки HTTP-запитів та взаємодії з базою даних за допомогою Spring Data JPA [8] та Hibernate [25]. Каталог `src` є основним каталогом у якому прописано

все, що стосується серверної логіки. Тут містяться підкаталоги та файли для різних аспектів функціонування серверної частини веб-застосунку.

У підкаталозі `controller` описані контролери для обробки HTTP-запитів. Вони відповідають за обробку вхідних HTTP-запитів від клієнтів та відправку відповідей на них. Вони взаємодіють із сервісами для виконання бізнес-логіки а також відповідають за отримання або збереження даних у базі даних. Контролери використовують анотації Spring для мапування URL-адрес на методи [7]. Опис найважливіших використаних анотацій можемо побачити у таблиці 4.1.

Таблиця 4.1 — Опис використаних анотацій

Анотації	Опис
<code>@GetMapping</code>	Використовується для обробки HTTP GET-запитів. Застосовується для отримання даних з сервера.
<code>@PostMapping</code>	Використовується для обробки HTTP POST-запитів. Застосовується для створення нових записів на сервері.
<code>@PutMapping</code>	Використовується для обробки HTTP PUT-запитів. Застосовується для оновлення існуючих записів на сервері.
<code>@DeleteMapping</code>	Використовується для обробки HTTP DELETE-запитів. Застосовується для видалення записів з сервера.
<code>@PathVariable</code>	Використовується для отримання значень з URL-адреси, наприклад, ідентифікатора користувача з URL (<code>/users/{id}</code>).
<code>@RequestBody</code>	Використовується для запиту, що результат методу контролера повинен бути записаний у тіло HTTP-відповіді. Застосовується для повернення JSON або XML даних у відповідь.
<code>@RestController</code>	Сполучає <code>@Controller</code> та <code>@ResponseBody</code> . Вказує, що клас є контролером, а всі методи автоматично повертають дані у вигляді JSON або XML, які записуються у тіло відповіді.
<code>@Autowired</code>	Використовується для автоматичного впровадження залежностей (наприклад, сервісів або репозиторіїв) у клас.
<code>@Service</code>	Вказує, що клас є сервісом, який містить бізнес-логіку застосунку.

У підкаталозі `model` містяться моделі даних, що відповідають за відображення таблиць бази даних. Моделі даних представляють структуру таблиць бази даних у вигляді Java класів. Кожен клас відображає відповідну таблицю та містить поля, які відповідають колонкам таблиці. Моделі використовують анотації JPA [8], такі як `@Entity`, `@Table`, `@Id`, `@Column`, щоб визначити відповідність між класами та таблицями бази даних.

Підкаталог `repository` відповідає за інтерфейси для взаємодії з базою даних. Репозиторії є інтерфейсами, які надають методи для виконання CRUD-операцій (створення, читання, оновлення, видалення) [24] з базою даних. Вони успадковують від інтерфейсу `JpaRepository`, що надається Spring Data JPA [8], і автоматично отримують реалізацію основних методів для роботи з базою даних [21].

У підкаталозі `service` описано бізнес-логіку застосунку. Сервіси отримують дані з репозиторіїв, обробляють їх відповідно до вимог бізнес-логіки та повертають результати контролерам. Сервіси використовують анотацію `@Service` для позначення їх як сервісних компонентів Spring [7].

Файл `App.java` це головний клас Spring Boot [7] застосунку, який виконує функцію точки входу до програми. Він містить метод `main`, що запускає Spring Boot застосунок. Файл використовує анотацію `@SpringBootApplication`, яка є комбінацією трьох анотацій: `@EnableAutoConfiguration`, `@ComponentScan`, та `@Configuration`. Цей файл конфігурує та ініціалізує всі компоненти Spring.

4.1.2 Ресурси для серверної частини

Каталог `src` для ресурсів серверної частини застосунку. Він містить файли та підкаталоги, які використовуються для налаштування, рендерингу та зберігання статичних файлів.

Підкаталог `static` відповідає за статичні файли. Цей підкаталог містить статичні файли, які використовуються для створення користувацького інтерфейсу застосунку. До таких файлів належать файли CSS [15] для стилізації веб-сторінок,

JavaScript [19], [20] файли для додавання динамічної поведінки та інші статичні ресурси (наприклад, зображення, шрифти).

Підкаталог `templates` містить шаблони, які використовуються для серверного рендерингу веб-сторінок. Spring Boot [7] за замовчуванням використовує Thymeleaf як шаблонізатор. Шаблони дозволяють динамічно створювати HTML-сторінки, включаючи дані з серверної частини.

Файл `application.properties` [7] використовується для налаштування параметрів Spring Boot застосунку. У ньому можна вказувати налаштування для підключення до бази даних, параметри логування, налаштування серверу та інші конфігураційні параметри. Цей файл дозволяє централізовано керувати конфігурацією застосунку без необхідності змінювати код.

4.1.3 Файли клієнтської частини.

Каталог `public` містить статичні файли, які використовуються клієнтською частиною веб-застосунку. До таких файлів належать HTML [15], CSS [15], JavaScript [19], зображення та інші ресурси, які завантажуються безпосередньо з веб-сервера до браузера користувача.

Каталог `views` містить HTML-шаблони, які використовуються для рендерингу сторінок на клієнтській стороні.

Каталог `frontend` містить усі файли, необхідні для розробки клієнтської частини веб-застосунку за допомогою React [10]. Він поділяється на підкаталоги `src` та `public`, а також містить конфігураційний файл `package.json`.

Підкаталог `src` містить всі вихідні файли для розробки React-компонентів. Він включає основні компоненти, файли для керування станом, маршрути та інші необхідні модулі. `Components` - каталог з React-компонентами, `App.js` - головний компонент застосунку, `index.js` - вхідна точка для React-застосунку, яка монтує головний компонент `App` у `DOM`.

Підкаталог `public` містить статичні файли, які використовуються React-застосунком [10]. Зазвичай це файли, які не змінюються під час роботи застосунку, такі як HTML-шаблон, іконки та інші статичні ресурси.

Файл `package.json` це файл з залежностями та налаштуваннями проекту. Цей файл містить інформацію про проект, його залежності, скрипти для запуску, збірки та інші налаштування. Він дозволяє керувати залежностями проекту та налаштовувати різні аспекти роботи з React-застосунком.

4.2 Налагодження серверної частини

Процес налагодження серверної частини нашого веб-застосунку з клієнто-серверною архітектурою, який є особистим кабінетом споживача електроенергії. Основна мета серверної частини полягає у забезпеченні надійної взаємодії між клієнтською частиною застосунку, яка побудована на React [10], і базою даних, що зберігає всі необхідні дані.

Для реалізації серверної частини ми використовуємо Spring Boot, що дозволяє швидко та ефективно створювати REST API [9] для обробки HTTP-запитів. Ми розглянемо, як створювати контролери для виконання CRUD-операцій [24], як впроваджувати бізнес-логіку через сервіси, та як забезпечити взаємодію з базою даних за допомогою JPA та Hibernate [25].

Особливу увагу буде приділено налаштуванню MySQL Workbench [13], [20], [21] для створення та керування схемою бази даних, а також налаштуванню параметрів підключення у Spring Boot. Важливою складовою буде і розгляд процесу розробки клієнтської частини, включаючи створення React-компонентів, використання Redux для керування станом та виконання HTTP-запитів за допомогою бібліотеки axios.

4.2.1 Сервер бази даних

Першим кроком у розробці веб-застосунку було налаштування бази даних. Для цього використовувалася MySQL Workbench, яку спершу потрібно було завантажити та встановити. Після встановлення MySQL Workbench, відбулось

підключення до MySQL-сервера, де було створено нову схему бази даних під назвою ElectricityBilling.

Було створено таблиці, які будуть використовуватися для зберігання різних типів даних. Наприклад, таблиця users містить інформацію про користувачів, таблиця ElectricityBilling зберігає дані про споживання електроенергії. Для зв'язків між таблицями були встановлені зовнішні ключі, щоб забезпечити цілісність даних.

Після налаштування бази даних, потрібно перейти до конфігурації підключення до неї у Spring Boot. У файлі pom.xml були додані необхідні залежності для роботи з MySQL та Spring Data JPA. Це включає залежності для mysql-connector-java, spring-boot-starter-data-jpa та spring-boot-starter-web. Потім у файлі application.properties [7] було налаштовано параметри підключення до бази даних, включаючи URL, ім'я користувача та пароль. Також було налаштовано параметри Hibernate [25], щоб керувати автоматичним оновленням схеми бази даних.

Коли користувач відкриває веб-застосунок у своєму браузері, клієнтська частина, розроблена на React, завантажує головний HTML-файл (index.html) та основні JavaScript-файли, які містяться у каталозі public. React-компоненти, які знаходяться у каталозі src, взаємодіють між собою для створення користувацького інтерфейсу.

Для надсилання HTTP-запитів до серверної частини застосовуються бібліотеки, такі як axios. Ці запити обробляються серверною частиною, яка розроблена на Spring Boot. Контролери, розташовані у каталозі controller, отримують HTTP-запити від клієнтської частини, викликають відповідні сервіси з каталогу service для виконання бізнес-логіки, і потім повертають відповіді назад до клієнтської частини. Сервіси взаємодіють з репозиторіями з каталогу repository, які використовують JPA [8] для доступу до бази даних.

Дані, які зберігаються у базі даних MySQL [13], включають інформацію про користувачів, їх споживання електроенергії та рахунки. Ці дані передаються від клієнтської частини до серверної частини, де вони зберігаються у відповідних

таблицях бази даних. Потім серверна частина формує відповіді на основі цих даних і повертає їх клієнтській частині, яка оновлює інтерфейс користувача відповідно до отриманих даних.

Для забезпечення зручного інтерфейсу користувача та динамічної взаємодії між компонентами, клієнтська частина використовує бібліотеки, такі як React для створення компонентів, Redux для керування станом застосунку (якщо це необхідно), React Router [10] для маршрутизації між різними сторінками та axios для виконання HTTP-запитів. У свою чергу, серверна частина використовує Spring Boot [7] для розробки REST API [9], Spring Data JPA [8] для роботи з базою даних та MySQL Connector для підключення до MySQL.

Таким чином, взаємодія між компонентами забезпечується через чітко визначені інтерфейси та використання сучасних бібліотек та фреймворків, що дозволяє ефективно розробляти та підтримувати веб-застосунок.

4.2.2 Налаштування веб-серверу

Розробка веб-сервера починається зі створення REST API за допомогою Spring Boot. Це включає створення контролерів, які обробляють HTTP-запити, та реалізацію методів для виконання CRUD-операцій (створення, читання, оновлення, видалення) [24].

Першим кроком є створення контролерів. Наприклад, UserController.java відповідає за обробку запитів, пов'язаних з користувачами. У цьому контролері визначаються методи для різних типів запитів: GET, POST, PUT та DELETE. Ці методи дозволяють створювати нових користувачів, отримувати інформацію про них, оновлювати їхні дані та видаляти користувачів.

Ключовими особливостями патерну MVC є те, що він відокремлює рівень представлення від бізнес-рівня. Контролер виконує дію виклику Моделі та надсилання даних до Представлення. Модель може навіть не знати, що її використовує якийсь веб-додаток [27].

Представлення взаємодії між компонентами шаблону MVC можна побачити зображене на рисунку 4.1.

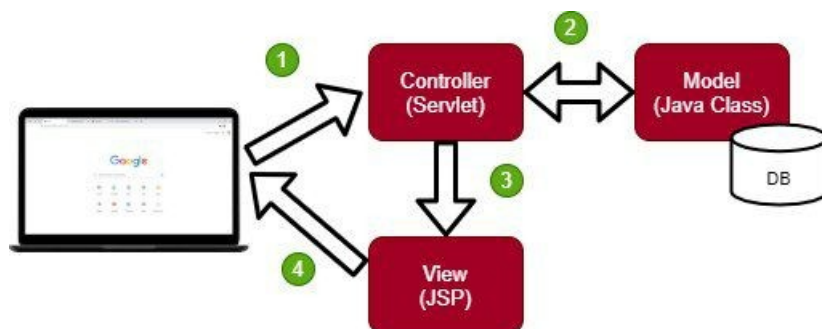


Рисунок 4.1 – Представлення взаємодії між компонентами шаблону MVC [27]

На рівні моделі (Model Layer) [27] міститься бізнес-логіка системи, а також відображається поточний стан програми. Контролер отримує дані з шару моделі та надсилає їх до шару представлення.

Рівень контролера діє як інтерфейс між представленням і моделлю. Він отримує запити від рівня представлення і обробляє їх, включаючи усі необхідні перевірки.

Рівень представлення (View Layer) [27] представляє вихідні дані програми, зазвичай це деяка форма користувацького інтерфейсу. Рівень представлення використовується для відображення даних моделі, отриманих контролером.

Наступним кроком є впровадження бізнес-логіки. Це реалізується через створення сервісів, які обробляють основну логіку застосунку. Наприклад, `UserService.java` містить методи для виконання операцій, що пов'язані з користувачами.

Веб-застосунок з бізнес-логікою дозволяє автоматизувати складні бізнес-процеси, обробляючи та аналізуючи великі обсяги даних для прийняття обґрунтованих рішень. Він забезпечує взаємодію між різними системами та користувачами, виконуючи операції за заздалегідь визначеними правилами та умовами

Сервіс використовує репозиторій для взаємодії з базою даних та включає методи для створення, отримання, оновлення та видалення користувачів.

Загальне представлення бізнес-логіки має вигляд як на рисунку 4.2.

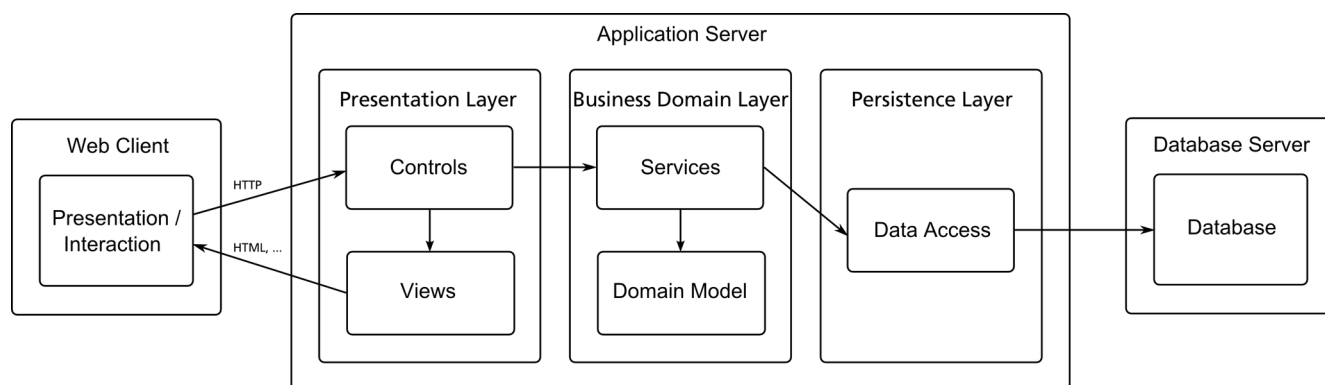


Рисунок 4.2 – Представлення шару бізнес-логіки

Для взаємодії з базою даних використовується JPA [8] та Hibernate [25]. Репозиторії, що знаходяться у каталозі repository, дозволяють виконувати операції з базою даних за допомогою методів, що надаються JPA [8].

Репозиторій UserRepository використовує інтерфейс JpaRepository, який надає базові CRUD-операції [24] без необхідності їхньої реалізації. Це значно спрощує процес взаємодії з базою даних.

Взаємодія між компонентами відбувається наступним чином:

Клієнтська частина на React відправляє HTTP-запити до серверної частини через REST API [9].

Контролери у серверній частині, наприклад, UserController.java, отримують ці запити та передають їх до відповідних сервісів.

Сервіси, такі як UserService.java, обробляють запити, виконуючи бізнес-логіку та взаємодіючи з репозиторіями для доступу до бази даних.

Репозиторії використовують JPA [8] та Hibernate [25] для виконання операцій з базою даних.

Результати операцій передаються назад через сервіси до контролерів, які формують відповідь і повертають її клієнтській частині.

Для реалізації всього цього використовуються різні бібліотеки та фреймворки. Spring Boot [7] забезпечує легке створення REST API [9] та інтеграцію з різними модулями. Spring Data JPA [8] спрощує взаємодію з базою

даних, надаючи абстракцію над базовими CRUD-операціями [24]. Hibernate [25] працює як ORM (Object-Relational Mapping) [23], дозволяючи використовувати об'єкти Java для взаємодії з реляційними базами даних. Взаємодію зображено на рисунку 4.3.

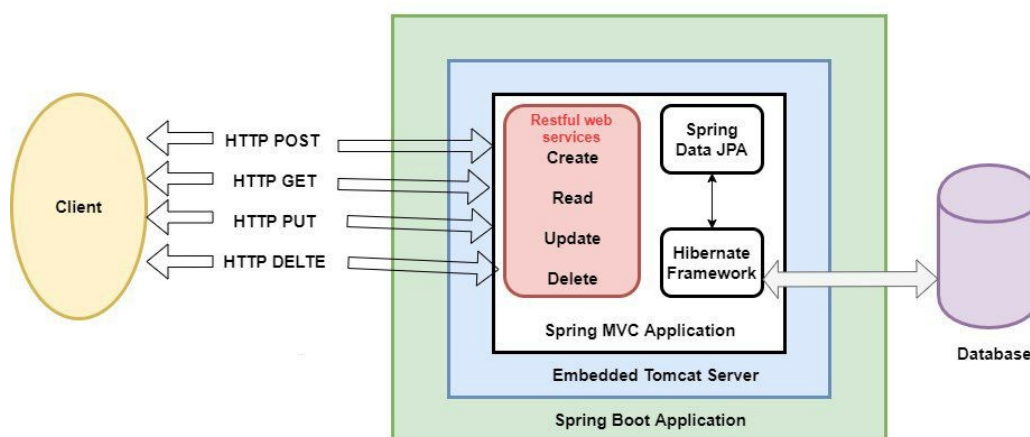


Рисунок 4.3 – Приклад операцій Spring Boot CRUD із JPA та Hibernate [25]

Таким чином, завдяки чіткій структурі проекту та використанню потужних інструментів, таких як Spring Boot [7], Spring Data JPA [8] та Hibernate [25], забезпечується ефективна розробка та інтеграція всіх компонентів веб-застосунку.

4.3 Взаємодія клієнтської частини з сервером

Розробка клієнтської частини починається зі створення основної структури застосунку на React [10]. Основне завдання фронтенду полягає у відображенні даних, отриманих з сервера, та забезпеченні інтерфейсу для користувача.

Для цього було створено окремі компоненти. На прикладі компонента UserComponent.js реалізовано як компонент може відповідати за відображення інформації про користувача. Важливо організувати компоненти таким чином, щоб

кожен з них виконував конкретну функцію та був максимально незалежним від інших.

Для керування станом застосунку я використовую бібліотеку Redux. Redux дозволяє централізовано зберігати стан застосунку та забезпечує предбачуваність його змін. Створюються actions (дії) та reducers (редуктори), які визначають, як стан змінюється у відповідь на певні дії.

Для виконання HTTP-запитів до серверу було використано бібліотеку Axios [11], яка дозволяє легко надсилати запити до сервера і обробляти відповіді. Дані, отримані від сервера, відображаються у компонентах React [10]. Наприклад, у `UserComponent.js` список користувачів зберігається у стані компонента і відображається у вигляді HTML-елементів.

Процес виглядає наступним чином:

Користувач відкриває веб-застосунок, який завантажує основні компоненти.

React-компонент `UserComponent.js` під час монтування виконує HTTP-запит до серверу за допомогою `axios`. Серверна частина, побудована на Spring Boot, обробляє цей запит і повертає відповідні дані у форматі JSON. Компонент отримує дані і оновлює свій стан, що призводить до повторного рендерингу з відображенням отриманих даних.

Для більш складного керування станом та взаємодії між компонентами використовується Redux. У Redux визначаються дії для виконання запитів до сервера, редуктори для оновлення стану на основі отриманих відповідей та селектори для отримання даних з Redux-стору.

Таким чином, клієнтська частина застосунку розробляється з використанням React для побудови компонентів інтерфейсу, Redux для керування станом та `axios` для виконання HTTP-запитів до серверної частини.

Такий алгоритм забезпечує зручний, динамічний та інтерактивний користувацький досвід, що дозволяє ефективно відображати та обробляти дані, отримані від сервера.

4.4 Алгоритм роботи веб-застосунку

Особистий кабінет споживача електроенергії дозволяє користувачам керувати своїм споживанням енергії, переглядати історію та статистику споживання та отримувати аналітику. Нижче наведено алгоритм роботи:

Користувач відкриває веб-додаток у браузері і переходить на сторінку входу, заповнює форму входу, вводячи свою електронну пошту та пароль.

На фронтенді JavaScript збирає введені дані та відправляє їх на сервер через API запит до відповідного маршруту. Сервер отримує запит і звертається до бази даних для перевірки облікових даних користувача.

Якщо облікові дані правильні, сервер створює сесію для користувача та відправляє відповідь з підтвердженням входу. Браузер отримує відповідь від сервера, зберігає токен авторизації і перенаправляє користувача на головну сторінку особистого кабінету. Користувач переглядає різні сторінки особистого кабінету за власним бажанням, як наприклад графіки споживання.

Фронтенд надсилає запити до API сервера для отримання інформації, такої як поточне споживання енергії, налаштування користувача тощо. Сервер обробляє ці запити, звертається до бази даних для отримання необхідної інформації та повертає дані у форматі json. Фронтенд отримує дані від сервера та динамічно оновлює інтерфейс користувача, відображаючи актуальну інформацію.

Користувач переходить на сторінку звітів, натискає на кнопку завантаження звіту. Браузер відправляє запит на сервер для отримання файлу звіту. Сервер отримує запит, обробляє його та виконує SQL-запити до бази даних для отримання необхідних даних. Браузер отримує файл і ініціює його завантаження на комп'ютер користувача.

Користувач натискає на кнопку виходу з облікового запису, фронтенд відправляє запит на сервер для завершення сесії. Сервер отримує запит на вихід, видаляє сесію користувача. Сервер відправляє відповідь з підтвердженням виходу. Браузер видаляє локально збережені дані сесії. Користувач перенаправляється на сторінку входу.

Цей алгоритм описує основні етапи роботи веб-застосунку для особистого кабінету споживача електроенергії. Він охоплює процеси входу в обліковий запис, навігації по сторінках, отримання та завантаження звітів, а також виходу з облікового запису, забезпечуючи зручність і ефективність взаємодії користувача з системою.

5 РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

Для того щоб зробити процес користування веб-застосунком особистого кабінету споживача електроенергії більш комфортним для користувача було зроблено простий інтерфейс а також дуже простий і невибагливий процес інсталяції, що дозволяє користуватися додатком людям будь-якого віку.

5.1 Системні вимоги

Для коректної роботи веб-застосунку користувачу достатньо сучасної операційної системи, наприклад Windows 7+, macOS 10.12+ або Linux. Браузер останньої актуальної версії, зазначеної розробником Google Chrome, Mozilla Firefox, Microsoft Edge, Safari з увімкненою підтримкою JavaScript та cookies. Веб-застосунок не потребує від користувача підключення ніяких сторонніх плагінів.

5.2 Приклад інтерфейсу

При відкритті сторінки веб-застосунку, користувач побачить вікно входу, зображене на рисунку 5.1, де потрібно здійснити введення даних: ввести адресу електронної пошти та пароль. Здійсниться перевірка чи наявний користувач у базі даних вже зареєстрованих споживачів електроенергії. Якщо у якомусь з полів введено дані у некоректному форматі або користувача не буде знайдено у базі даних, користувач побачить відповідне вікно з повідомленням про помилку. У верхньому правому куті екрану можна побачити кнопку для переходу на сторінку з реєстрацією для користувачів, у яких нема особистого кабінету.

web-Енérго

Ваша енергія — наша турбота!

Реєстрація

Вітаємо!

Пошта:

Email

Пароль:

Пароль

Вхід

© 2024 web-Енérго. Права захищено.

Рисунок 5.1 – Вікно входу в особистий кабінет споживача електроенергії

Після натискання на кнопку “Реєстрація” здійснюється перехід на сторінку, де можна побачити форму, зображену на рисунку 5.2, що потребує заповнення. Користувачу потрібно ввести свій ідентифікаційний номер за яким йому надходять квитанції, ім’я, прізвище, номер телефону, адресу електронної пошти та придумати пароль. Усі ці поля є обов’язковими для заповнення.

web-Енѣрго

Ваша енергія — наша турбота!

Повернутися

Вітаємо!

ID:

Ім'я:

Прізвище:

Номер телефону:

Пошта:

Email

Пароль:

Пароль

Зареєструватися

© 2024 web-Енѣрго. Права захищено.

Рисунок 5.2 – Вікно реєстрації

Якщо ж вхід до особистого кабінету здійснено успішно, то користувач переходить на сторінку, яку можна побачити на рисунку 5.3, із щоденною статистикою витрат електроенергії протягом останнього платіжного періоду. Статистику подано у вигляді стовпчикової діаграми, де по лівій стороні легенди відображено споживання електроенергії, виміряної у кВт/год, а під діаграмою можна побачити дати у форматі рр-мм-дд. На сайдбарі користувач може побачити сумовану кількість спожитих енергоресурсів за поданий на діаграмі період, що вимірюється та загальну вартість електроенергії, обрахованої за тарифом 2.64 грн.

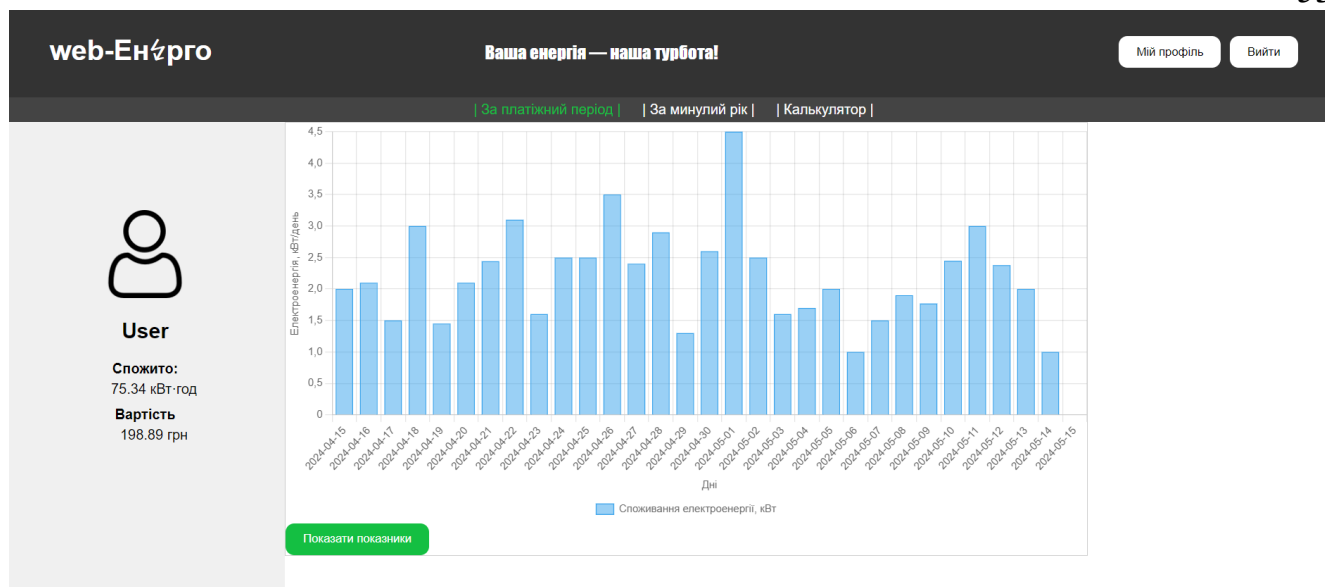


Рисунок 5.3 – Сторінка особистого кабінету споживача із щоденною статистикою витрат електроенергії протягом останнього платіжного періоду

Під графіком є кнопка, при натисканні на яку буде обраховано середнє значення спожитої електроенергії за період, дні із мінімальним та максимальним електроспоживанням, а також буде зазначено, якщо буде виявлено день із аномальним показником електро споживання, у програмі передбачено що це або від'ємні значення, або значення, що є у 5 разів більші за середнє. Проведені після натискання кнопки розрахунки можна побачити на рисунку 5.4.

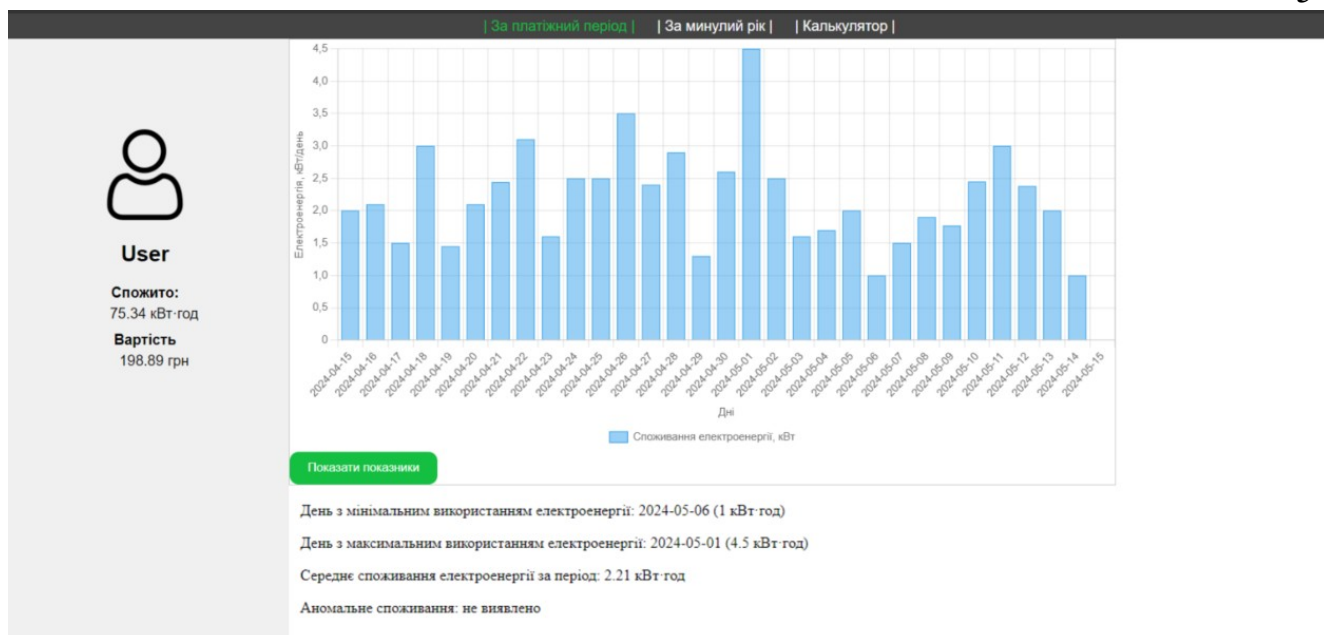


Рисунок 5.4 – Обрахунки середнього значення спожитої електроенергії за період, дні із мінімальним та максимальним електроспоживанням та виявлення аномального значення

Користувач має змогу обрати переглянути статистику власного енергоспоживання за минулий рік, де також буде відображено сумовану кількість спожитої електроенергії та її вартість. Також користувач може порівняти свої витрати із середнім значенням витрат користувачів у базі даних. Це зображено на рисунку 5.5.

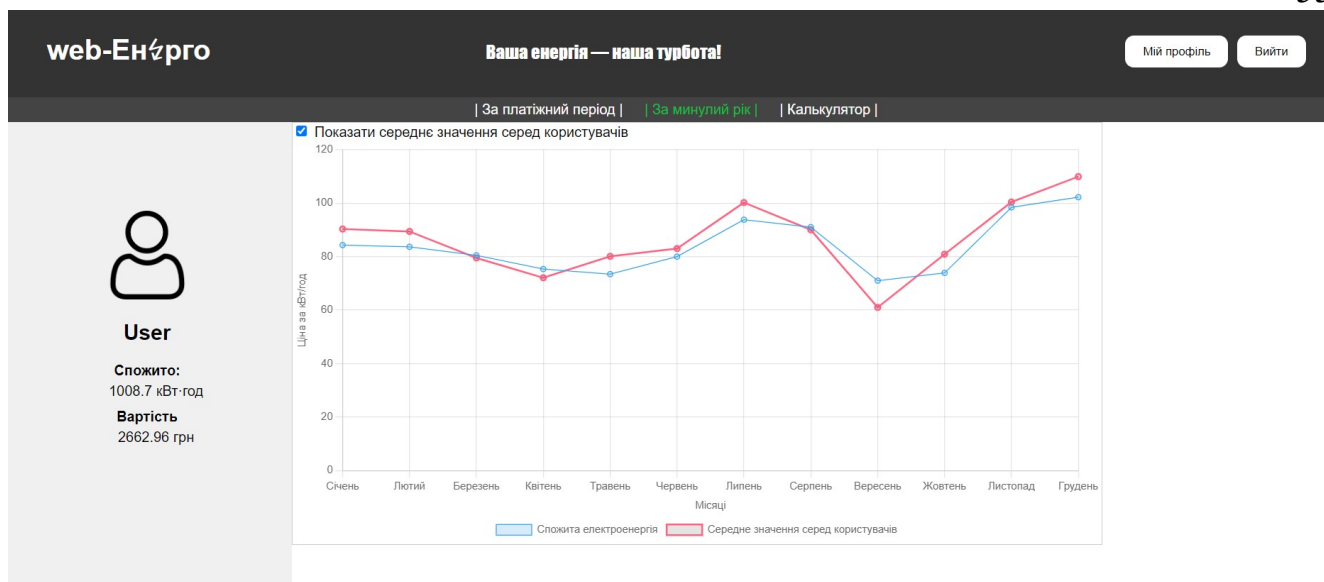


Рисунок 5.5 – Статистику енергоспоживання користувача за минулий рік

На рисунку 5.6 можна побачити сторінку з профілем користувача та звітом, що є доступним для надсилання на пошту та завантаження у текстовому форматі:

The screenshot shows the 'web-Енєрго' interface with the header 'Ваша енергія — наша турбота!'. The main content area is divided into two sections. On the left is the 'Мій профіль' (My Profile) section, which includes a user icon, the name 'User', and contact information: 'Пошта: topalova13062013@gmail.com', 'ID: 880216', 'Пільги: none', and 'Борг: 0'. On the right is the 'Звіт про споживання електроенергії' (Report on electricity consumption) section, which displays the following information: 'Місяць: Квітень 2024', 'Спожито: 75.34 кВт·год', 'Вартість: 198.89 грн', and 'Дата наступного платежу: 15.05.2024'. Below the report information are two buttons: 'Завантажити звіт' (Download report) and 'Надіслати на пошту' (Send to email).

Рисунок 5.6 – Сторінка з профілем користувача та звітом

На рисунку 5.7 можна побачити лист, який користувач отримує на вказану в особистому кабінеті, а отже і в базі даних, електронну пошту у випадку, якщо обере відповідну функцію:

Документ



Olha <topalova13062013@gmail.com>

07:02

Кому: topalova13062013@gmail.com

Звіт про споживання електроенергії

Місяць: Квітень 2024

Спожито: 350 кВт·год

Вартість: 700 грн

Дата наступного платежу: 15.05.2024

Рисунок 5.7 – Лист на пошті користувача

Таким чином користувач взаємодіє з веб-застосунком через зручний та інтуїтивний інтерфейс, побудований за допомогою React. Інтерфейс забезпечує швидкий доступ до основних функцій, таких як перегляд поточного споживання електроенергії, історії рахунків та особистих даних. Всі візуальні елементи, включаючи кнопки, форми та таблиці, ретельно розроблені для забезпечення максимальної зручності, привабливості та ефективності.

ВИСНОВКИ

В ході виконання дипломної роботи та розробки веб-застосунку особистого кабінету споживача електроенергії АТ “НАЕК “Енергоатом” дозволяє ефективно вирішувати ряд важливих задач, пов'язаних з моніторингом та плануванням енергоспоживання. У процесі виконання даної роботи були досягнуті наступні результати:

1. На основі аналізу підходів, методів та алгоритмів розв’язання поставленої задачі обґрунтовано використання таких мов програмування як HTML, CSS, Javascript, Node.js, SQL, Spring Boot та React. За допомогою перелічених мов програмування було створено веб-застосунок з використанням останніх версій інструментів та надійні перевірені механізми реалізації.

2. Розроблено програмну систему веб-застосунку особистого кабінету споживача електроенергії АТ “НАЕК “Енергоатом”, перевагами якого є зручний та інтуїтивно зрозумілий інтерфейс для користувачів, що дозволяє легко управляти своїм споживанням енергії та отримувати аналітичні звіти. На серверній стороні використання Node.js забезпечило ефективну обробку запитів та взаємодію з базою даних. SQL бази даних дозволили організувати зберігання великих обсягів даних та швидкий доступ до них.

3. На основі тестування доведено що веб-застосунок відповідає на поставлені задачі та виконує заплановані функції, такі як автоматизація процесів відстеження споживання енергії, генерації звітів та управління користувацькими даними, що значно спрощує взаємодію користувачів із системою.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Особистий кабінет для приватних клієнтів URL: <https://my.kyiv.yasno.com.ua/> (дата звернення 17.05.2024)
2. Особистий кабінет ПрАТ «ДТЕК Київські регіональні електромережі» URL: <https://www.dtek-krem.com.ua/ua/cabinet> (дата звернення 17.05.2024)
3. ОСОБИСТИЙ КАБІНЕТ СПОЖИВАЧА ЕНЕРГОРЕСУРСІВ URL: <https://ienergo.com.ua/> (дата звернення 17.05.2024)
4. The Benefits of Building a Backend First (and Vice Versa) URL: <https://www.linkedin.com/pulse/benefits-building-backend-first-vice-versa-austin-stewart#:~:text=Backend%20developers%20can%20work%20independently,team%20members%20with%20different%20expertise.&text=With%20the%20backend%20in%20place,and%20validation%20of%20core%20functionalities> (дата звернення 02.06.2024)
5. Frontend, Backend чи Full Stack: в чому відмінності, та що краще обрати для старту кар'єри в IT-сфері URL: <https://dan-it.com.ua/uk/blog/frontend-backend-ili-full-stack-v-chem-razlichija-i-hto-luchshe-vybrat-dlja-starta-karery-v-it-sfere/> (дата звернення 02.06.2024)
6. Java Downloads — Oracle URL: <https://www.oracle.com/java/technologies/downloads/> (дата звернення 02.06.2024)
7. Spring Boot — Application Properties URL: <https://www.geeksforgeeks.org/spring-boot-application-properties/> (дата звернення 02.06.2024)
8. Defining JPA Entities URL: <https://www.baeldung.com/jpa-entities> (дата звернення 02.06.2024)
9. Building REST services with Spring URL: <https://spring.io/guides/tutorials/rest> (дата звернення 02.06.2024)

10. Start a New React Project URL: <https://react.dev/learn/start-a-new-react-project>
(дата звернення 02.06.2024)
11. Getting Started | Axios Docs URL: <https://axios-http.com/docs/intro> (дата звернення 02.06.2024)
12. Курс Модуль 4. Робота з БД - Лекція: Концепція ACID URL: <https://javarush.com/ua/quests/lectures/ua.questhibernate.level18.lecture02> (дата звернення 02.06.2024)
13. Основні переваги субд MySQL URL: <https://studfile.net/preview/5607354/page:3/> (дата звернення 02.06.2024)
14. Why did we build Visual Studio Code? URL: <https://code.visualstudio.com/docs/editor/whyvscode> (дата звернення 17.05.2024)
15. HTML: HyperText Markup Language URL: <https://developer.mozilla.org/en-US/docs/Web/HTML> (дата звернення 17.05.2024)
16. Node.js debugging in VS Code URL: <https://code.visualstudio.com/docs/nodejs/nodejs-debugging> (дата звернення 17.05.2024)
17. User Interface URL: <https://code.visualstudio.com/docs/getstarted/userinterface>
(дата звернення 17.05.2024)
18. Extension Guides URL: <https://code.visualstudio.com/api/extension-guides/overview> (дата звернення 17.05.2024)
19. Node.js and JavaScript in Visual Studio Code URL: <https://code.visualstudio.com/learn/educators/nodejs> (дата звернення 17.05.2024)
20. Access SQL Server in Node.js URL: <https://www.tutorialsteacher.com/nodejs/access-sql-server-in-nodejs> (дата звернення 17.05.2024)
21. Connect and query SQL Server URL: <https://learn.microsoft.com/en-us/sql/ssms/quickstarts/ssms-connect-query-sql-server?view=sql-server-ver16> (дата звернення 17.05.2024)

22. Architecture flow of spring boot Applications URL:
https://www.researchgate.net/figure/Fig-2-Architecture-flow-of-spring-boot-Applications-Spring-boot-uses-all-the-features_fig2_341151097 (дата звернення 02.06.2024)
23. Top 11 Node.js ORMs, query builders & database libraries in 2022 URL: <https://www.prisma.io/dataguide/database-tools/top-nodejs-orms-query-builders-and-database-libraries> (дата звернення 02.06.2024)
24. CRUD-ПЕРЕВІРКА БАЗИ ДАНИХ URL:
<https://training.qatestlab.com/blog/technical-articles/crud-database-validation/#:~:text=CRUD> (дата звернення 02.06.2024)
25. Spring Boot CRUD Operations Example with JPA and Hibernate URL:
<https://www.javaguides.net/2019/10/spring-boot-crud-operations-example-with-jpa-hibernate.html> (дата звернення 02.06.2024)
26. The DAO Pattern in Java URL: <https://www.baeldung.com/java-dao-pattern> (дата звернення 02.06.2024)
27. Model View Controller (MVC) Design Pattern in Java URL:
<https://www.javaguides.net/2019/08/model-view-controller-mvc-design-in-java.html> (дата звернення 02.06.2024)

ДОДАТОК А

РЕАЛІЗАЦІЯ СЕРВЕРНОЇ ЧАСТИНИ ДЛЯ ОТРИМАННЯ
ДАНИХ ПРО ЩОДЕННЕ СПОЖИВАННЯ ЕЛЕКТРОЕНЕРГІЇ
ЗА ПЛАТІЖНИЙ ПЕРІОД ДЛЯ ВІДОБРАЖЕННЯ ЦИХ
ДАНИХ У ВИГЛЯДІ ГРАФІКА.

“Веб-застосунок особистого кабінету споживача електроенергії
АТ “НАЕК “Енергоатом”

УКР.НТУУ”КПІ ім. Ігоря Сікорського”_ІАТЕ_ЦТЕ_ТР-03

Аркушів 4

Київ - 2024

```
npm install express mysql2 cors chart.js
npm install axios chart.js react-chartjs-2
const express = require('express');
const mysql = require('mysql2');
const cors = require('cors');

const app = express();
app.use(cors());
app.use(express.json());

const db = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  password: 'password',
  database: 'ElectricityBilling'
});

db.connect(err => {
  if (err) {
    console.error('Error connecting to the database:', err);
    return;
  }
  console.log('Connected to the MySQL database.');
```

```
});

app.get('/api/consumption/:periodId', (req, res) => {
  const periodId = req.params.periodId;
  const query = `
```

```

SELECT d.Date, d.Consumption
FROM DailyConsumption d
INNER JOIN PayPeriods p ON d.PeriodID = p.PeriodID
WHERE p.PeriodID = ?
`;
db.query(query, [periodId], (err, results) => {
  if (err) {
    console.error('Error fetching data from database:', err);
    res.status(500).send('Internal Server Error');
    return;
  }
  res.json(results);
});
});

```

```

const PORT = process.env.PORT || 5000;
app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}`);
});

```

```
// src/components/ConsumptionChart.js
```

```

import React, { useEffect, useState } from 'react';
import axios from 'axios';
import { Line } from 'react-chartjs-2';

const ConsumptionChart = ({ periodId }) => {
  const [data, setData] = useState({ labels: [], datasets: [] });

  useEffect(() => {

```

```

    axios.get(`http://localhost:5000/api/consumption/${periodId}`)
      .then(response => {
        const labels = response.data.map(item => item.Date);
        const consumptionData = response.data.map(item => item.Consumption);

        setData({
          labels,
          datasets: [
            {
              label: 'Daily Consumption',
              data: consumptionData,
              borderColor: 'rgba(75,192,192,1)',
              backgroundColor: 'rgba(75,192,192,0.2)',
            },
          ],
        });
      })
      .catch(error => {
        console.error('Error fetching consumption data:', error);
      });
  }, [periodId]);

  return (
    <div>
      <h2>Consumption Chart</h2>
      <Line data={data} />
    </div>
  );
};

```

```
export default ConsumptionChart;
```

```
// src/App.js
```

```
import React from 'react';
```

```
import ConsumptionChart from '../components/ConsumptionChart';
```

```
const App = () => {
```

```
  const periodId = 1;
```

```
  return (
```

```
    <div>
```

```
      <h1>Electricity Billing Dashboard</h1>
```

```
      <ConsumptionChart periodId={periodId} />
```

```
    </div>
```

```
  );
```

```
};
```

```
export default App;
```

ДОДАТОК Б

ДОВІДКА ПРО ВПРОВАДЖЕННЯ РЕЗУЛЬТАТІВ ДИПЛОМНОГО ДОСЛІДЖЕННЯ

“Веб-застосунок особистого кабінету споживача електроенергії
АТ “НАЕК “Енергоатом”

УКР.НТУУ”КПІ ім. Ігоря Сікорського”_ІАТЕ_ЦТЕ_ТР-03

Аркушів 1

Київ - 2024



АКЦІОНЕРНЕ ТОВАРИСТВО
«НАЦІОНАЛЬНА АТОМНА
ЕНЕРГОГЕНЕРУЮЧА
КОМПАНІЯ «ЕНЕРГОАТОМ»

Філія «Відокремлений підрозділ
«НАУКОВО-ТЕХНІЧНИЙ ЦЕНТР»

вул. Гоголівська, 22-24, м. Київ,
01054, Україна

телефон/факс +38 (044) 206-97-21
e-mail: office@ntc.atom.gov.ua

Директору
Навчально-наукового інституту
атомної та теплової енергетики
НТУУ «КПІ ім. Ігоря Сікорського»

Євгенію ПИСЬМЕННОМУ

ДОВІДКА

про впровадження результатів дипломного дослідження

Топалової Ольги Євгенівни