МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет

Факультет електроніки та інформаційних технологій

Кафедра комп'ютерних наук

«До захисту допущено»

В.о. завідувача кафедри

_____ Ігор ШЕЛЕХОВ

(підпис)

09 червня 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавр

зі спеціальності 122 – Комп'ютерних наук,

освітньо-професійної програми «Інформатика»

на тему: «Інформаційна система управління взаємовідносинами з клієнтами підприємства електропостачання»

здобувача групи ІН-92 Карася Олександра Івановича

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

_____ Олександр КАРАСЬ

(підпис)

Керівник
старший викладач,
кандидат фізико-математичних наук Оксана ШОВКОПЛЯС _____

(підпис)

Суми – 2023

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавра

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувача групи ІН-92 Карася Олександра Івановича

1. Тема роботи: «Інформаційна система управління взаємовідносинами з клієнтами підприємства електропостачання»
затверджую наказом по СумДУ від *«01» червня 2023 р. № 0475-VI*
2. Термін здачі здобувачем кваліфікаційної роботи *до 09 червня 2023 року*
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
1) Аналіз проблеми та актуальності розробки систем управління взаємовідносинами з клієнтами, конкурентів та цільової аудиторії, постановка й формування завдань дослідження. 2) Огляд та вибір програмних засобів для інформаційних систем. 3) Розроблення інформаційної системи для управління взаємовідносинами з клієнтами для підприємства електропостачання. 4) Аналіз отриманих результатів
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____
6. Консультанти до проєкту (роботи), із значенням розділів проєкту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «____» _____ 20 ___ р.
Завдання прийняв до виконання _____ Керівник _____
(підпис) (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз проблеми та актуальності розробки систем управління взаємовідносинами з клієнтами, конкурентів та цільової аудиторії, постановка й формування завдань дослідження*		
2	*Огляд та вибір програмних засобів для інформаційних систем.*		
3	*Розроблення інформаційної системи для управління взаємовідносинами з клієнтами для підприємства електропостачання*		
4	*Аналіз отриманих результатів*		
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*		

Здобувач вищої освіти _____ Керівник _____
(підпис) (підпис)

АНОТАЦІЯ

Записка: 75 стр., 37 рис., 4 додатки, 30 використаних джерел.

Обґрунтування актуальності теми роботи – Тема кваліфікаційної роботи є актуальною, оскільки присвячена системі управління взаємовідносинами з клієнтами в межах організації шляхом розробки відповідних компонентів, модулів, моделей та інформаційних технологій.

Об'єкт дослідження – процес управління взаємодії з користувачами.

Предмет дослідження – методи і моделі автоматизованої інформаційної системи.

Мета роботи – розробка інформаційної системи управління взаємодії з користувачами для підприємства електропостачання за допомогою мов програмування та їх фреймворків.

Методи дослідження – методи передачі, обробки й зберігання інформації та алгоритми розроблення відповідних технологій.

Результати – розроблено інформаційну систему, яка надає змогу користувачам знаходити відповіді на запитання в інтерактивному боті, присилати свої показники електроенергії, розраховувати приблизну вартість переданих показників, відслідковувати спожиту електроенергію та замовляти послуги компанії, а менеджерам контролювати процеси і взаємодіяти з користувачами. Проведено тестування розробки на тестових даних клієнтів.

ІНФОРМАЦІЙНА СИСТЕМА, ЕЛЕКТРОПОСТАЧАННЯ,
REACTJS, NODEJS, MONGODB, CRM

ЗМІСТ

ВСТУП

Актуальність. Тема кваліфікаційної роботи є актуальною, оскільки присвячена системі управління взаємовідносинами з клієнтами в межах організації та реалізована шляхом розробки відповідних компонентів, модулів, моделей та інформаційних технологій.

Об’єкт дослідження – Процес управління взаємодії з користувачами.

Предмет дослідження – Методи і моделі автоматизованої інформаційної системи управління взаємодії з користувачами.

Гіпотеза. Автоматизацію бізнес процесів взаємодії з користувачами організацій можна досягнути шляхом застосування інформаційної технології, що реалізує сервіс для покращення задоволеності клієнтів та зниженню втрат клієнтів.

Новизна. Описане у даній роботі програмне рішення дозволить досягти більшої ефективності в побудові інформаційної системи підприємства, дозволяючи автоматизувати внутрішні процеси та спростити аналіз та візуалізації статистичних даних.

Апробація матеріалів роботи. Основні результати роботи оприлюднені та обговорені на міжнародній науково-технічній конференції студентів та молодих вчених «Інформатика, математика, автоматика» (IMA – 2023).

Структура. Дана робота складається зі вступу, аналітичного огляду, постановки задачі, вибір засобів та мов програмування для реалізації поставленої задачі, опису програмного забезпечення інформаційної системи, висновків, списку використаних джерел та додатків.

Зв'язок роботи з науковою темою. Кваліфікаційна робота виконана на кафедрі комп’ютерних наук та пов’язана з виконанням науково-дослідної роботи № 0118U006971 «Методи, математичні моделі та інформаційні технології аналізу і синтезу інфокомунікаційних систем» (2018-2023).

1 АНАЛІТИЧНИЙ ОГЛЯД

1.1 Аналіз предметної області

На сьогодні інформація виступає провідним засобом розвитку людства. В історії бізнесу інформація є безцінним і життєво важливим ресурсом і навіть має власну цінність. Підприємства активно фінансують інформаційні технології, оскільки вони приносять економічну цінність і прибутковість. Цінність відображається в значному підвищенні конкурентоспроможності, покращенні продуктивності та збільшенні прибутку. Інформаційні системи розглядаються як засіб підвищення і підвищення продуктивності праці, професійного зростання в різних компаніях і бізнесах [1]. Постійний потік інформації необхідний для успішного функціонування будь-якої системи, наприклад соціально-економічної та організаційної виробництва.

Інформаційна система – це системи, яка здійснює або в якій відбуваються інформаційні процеси: пошук, збирання, зберігання, передавання й опрацювання інформації [2]. Фундаментом діяльності кожної системи є процес, а фундаментом інформаційної системи – створення інформації для виконання вимог підприємства та гарантування успішного управління його роботою.

Для кращої побудови та автоматизації бізнес процесів та покращення взаємовідносин з клієнтами через веб ресурс введено таку технологію як CRM.

В загальному розумінні це технологія для керування всіма діловими відносинами з наявними та потенційними клієнтами всередині вашої компанії. Вона розроблена для того, щоб оптимізувати ваші бізнес-процеси [3,4]. Тож CRM-система дозволяє постійно підтримувати зв’язок з вашими лідами, прискорювати рішення робочих завдань та збільшувати прибуток, основні переваги можливо переглянути на рисунку 1.1:

Рисунок 1.1 – Основні переваги CRM системи

Щоб краще зрозуміти сутність такої системи розглянемо на прикладі ТЦ для бізнесу. Це велика онлайн платформа, яка автоматично збирає інформацію про клієнтів, відстежує ваше з ними спілкування на всіх етапах та дозволяє ділитися даними зі своєю командою, щоб делегувати завдання та прискорювати їх вирішення.

Саме таку систему ми будемо використовувати при створенні нашого сайту для підприємства електропостачання «СумЕнерго». Основними перевагами ведення онлайн бізнесу є гнучкість, охоплення клієнтів зі всього міста, економія кошт, автоматизоване управління даними. Для залучення нових та збереження вже існуючих клієнтів дуже важливо прийняти комплексну онлайн-стратегію та йти в ногу з новими тенденціями:

– Можливість використовувати сайт на мобільному пристрої. В наш час люди витрачають майже 90% свого часу на смартфон та різні мобільні додатки, тому розумним рішенням є впровадження системи на комп’ютерах та мобільних пристроях.

- Наявність підтримки користувачів у вигляді онлайн консультанта. Не дивлячись на велике використання ресурсу як системи для передачі показників та налагодження електричної мережі в будівлі, споживачам все одно потрібна підтримка та взаємодія консультанта. Такими консультантами можуть бути автоматизовані боти телеграм, які за спеціальними ключовими словами користувача будуть видавати корисні відповіді або посилання. Такий спосіб більш привабливий, так як з часу створення телеграм послуг та магазинів, люди почали користуватися більше часу.
- Активне залучення клієнтів до системи. Зрозуміло, що більшість літніх людей не можуть відвідувати такі системи і працюють з підприємствами в телефонному режимі. Проте за допомогою активної розсилки, участь в соціальних мережах, розсилки з новинами по електронній пошті, влаштовування регулярних оновлень системи, дарування подарунків на свята можливо залучити все більше клієнтів до своєї системи.

Отже, інформаційні технології відіграють важливу роль в сучасному світі, а інформаційні системи – в сучасному бізнесі. Їх використання створ ює нові можливості для розвитку та оптимізації компанії

1.2 Актуальність розробки CRM

В наш час кожна людина заклопотана своїми справами та й інколи забуває про передачу показників через велику кількість мобільних номерів або пошт. Інколи люди не можуть зрозуміти чому так багато спожили енергетичних ресурсів та не можуть проаналізувати свої витрати. Крім того й оплата є значною проблемою в наш час, постійна зміна тарифів та розрахункових рахунків. Аби вирішити кожну з таких проблем створюються CRM інформаційні системи. Для клієнтів це вирішення будь-яких проблем, так як можливо онлайн переглянути актуальну інформацію, надіслати показники прямо з телефону не шукаючи

пошти та номери компаній, оплатити квитанції, проаналізувати використання енергії тощо. Для підприємства це нові клієнти та автоматизування системи, що дає змогу не відволікатися на дзвінки для ручного занесення показників в бази даних клієнтів [5].

Сьогодні наявність таких систем забезпечує чималі можливості для онлайн-транзакцій та передачі показників безпечним та зручним шляхом. CRM системи пропонують підприємствам та користувачам однаково вигідні умови та можливості. З детальними можливостями можливо ознайомитись в таблиці 1.1.

Таблиця 1.1 – Можливості використання CRM систем.

Можливості для підприємства	**Можливості для покупця**
Широке охоплення клієнтів по всьому місту	Широкий вибір послуг
Приріст конкурентоспроможності	Цілодобова доступність
Покращення стосунків з клієнтами	Вигідні тарифи та пропозиції
Збільшення надання послуг	Прості перекази показників та їх аналіз
Економічна ефективність	Економія часу

1.3 Аналіз конкурентів-аналогів

Одним із завдань роботи є створення розроблення бізнес логіки та створення зрозумілого інтерфейсу для підприємства електропостачання «СумЕнерго», яке орієнтоване на забезпечення клієнтів послугами встановлення електролічильників, передачі показників. Беручи це до уваги, було проведено моніторинг різних сайтів такого типу. Наприклад, сайт «Енера Суми» https://sm.enera.ua/ (рис. 1.2), сайт «ХарківОблЕнерго» https://oblenergo.kharkov.ua/ (рис. 1.3), сайт «МиколаївЕнерго» https://www.energy.mk.ua/ (рис 1.4).

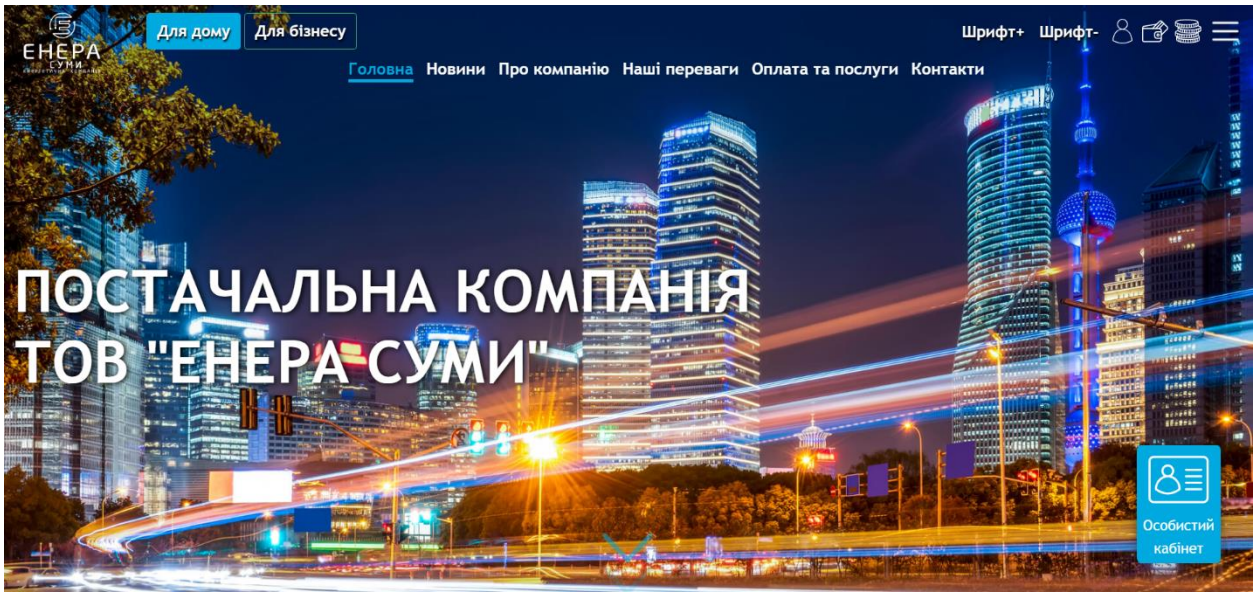

Рисунок 1.2 – Інтерфейс сайту «Енера Суми»

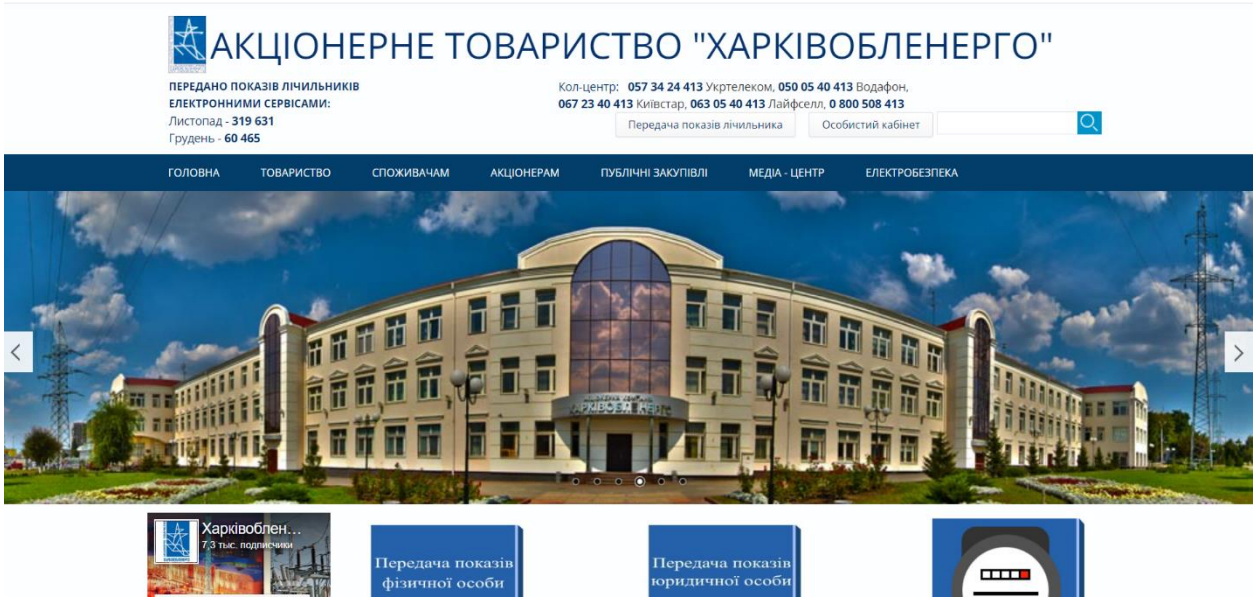

Рисунок 1.3 – Інтерфейс сайту «ХарківОблЕнерго»

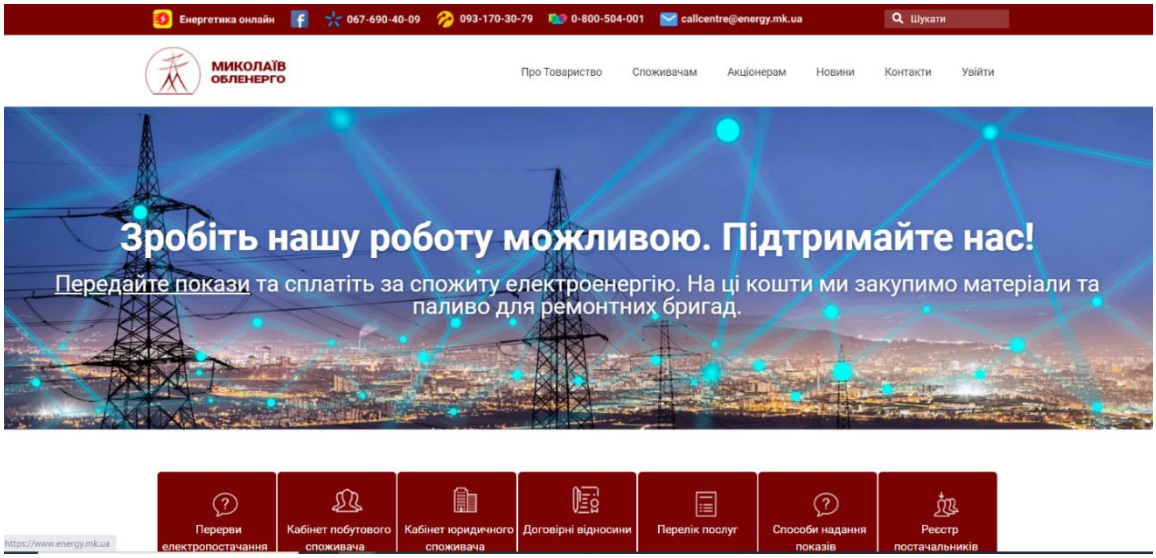

Рисунок 1.4 – Інтерфейс сайту «МиколаївОблЕнерго»

На основі вищезазначених інформаційних систем була створена порівняльна таблиця з аналізом аналогів (таблиця 1.2).

Таблиця 1.2 Аналіз конкурентів

Критерії	**Сайти**		
	«Енера Суми»	**Сайт «Харків ОблЕнерго»**	**Сайт «Миколаїв ОблЕнерго»**
Зручність використання сайту	+	+	+
Дизайн сайту	–	+	+
Інформація про послуги	–	–	+
Швидкий доступ до особового кабінету	+	+	–
Наявність онлайн підтримки	–	–	+

Згідно з результатами проведеного аналізу аналогів можна виокремити деякі переваги та недоліки, посилаючись на які буде створено інформаційну систему CRM для підприємства «СумЕнерго».

Переваги веб-сайтів аналогів:

– зовнішній вигляд та зручний функціонал сайту;

– детальна інформація про послуги;

– можливість перейти до особистого кабінету;
– детальна інформація про підприємство;
– можливість переглянути новини компанії та міста.

Недоліки веб-сайтів аналогів:

– поєднання кольорів в дизайні;
– відсутнє онлайн консультування;
– відсутність інформації про актуальні послуги.

1.4 Аналіз цільової аудиторії

Для автоматизації системи і проектування майбутнього додатку потрібно знати потенційну цільову аудиторію, яка буде користуватися електронним веб-сервісом. Було проведено дослідження серед представників обох статей, чоловіків та жінок (діаграма 1.1), які майже однаково представлені (44% - жінки та 56% чоловіки).

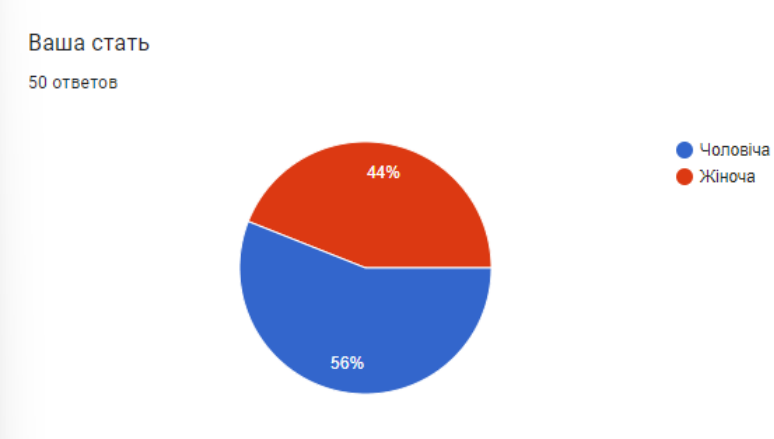

Рисунок 1.5 – Стать респондентів

Серед учасників опитування переважають повнолітні та літні респонденти (діаграма 1.2): до 23-30 років та 45-60 належать 32%, до 35-45 належать 30%, інші респонденти віком старші за 18 років або молодше 23.

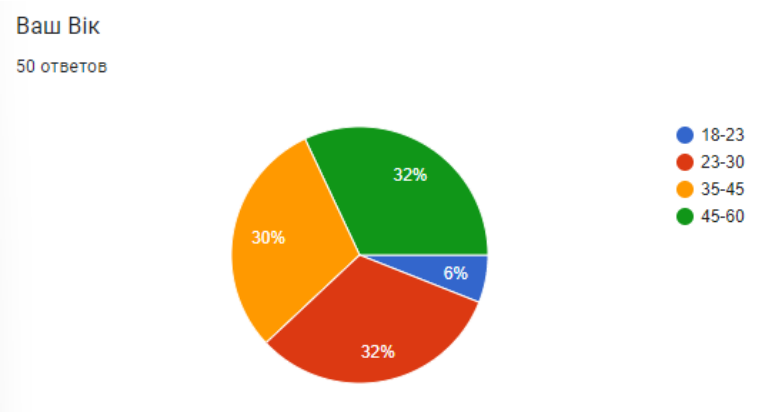

Рисунок 1.6 – Вік респондентів

Під час дослідження було з’ясовано на які ключові критерії звертають користувачі при перегляді сайтів підприємств (діаграма 1.3). До них входять: актуальність інформації (відображення послуг, тарифів), інформаційні довідки споживачам, зрозумілий інтерфейс та швидкодія застосунку.

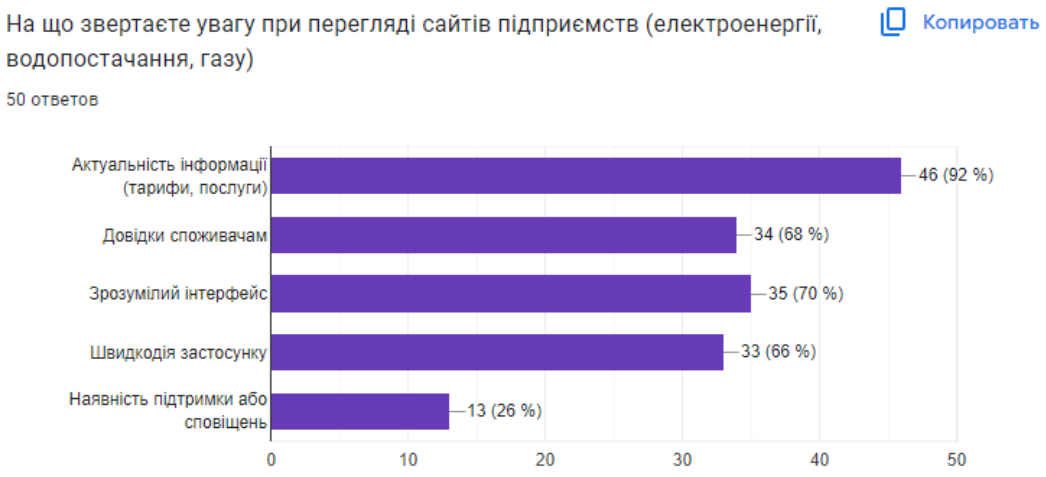

Рисунок 1.7 – Потреби клієнтів

На питання чи користуються респонденти електронними сайтами або додатками для передачі показників 64% згодились, інші 30% та 6% відповіли, що передають виключно в телефонному режимі або зовсім не користуються даними системами (діаграма 1.4).

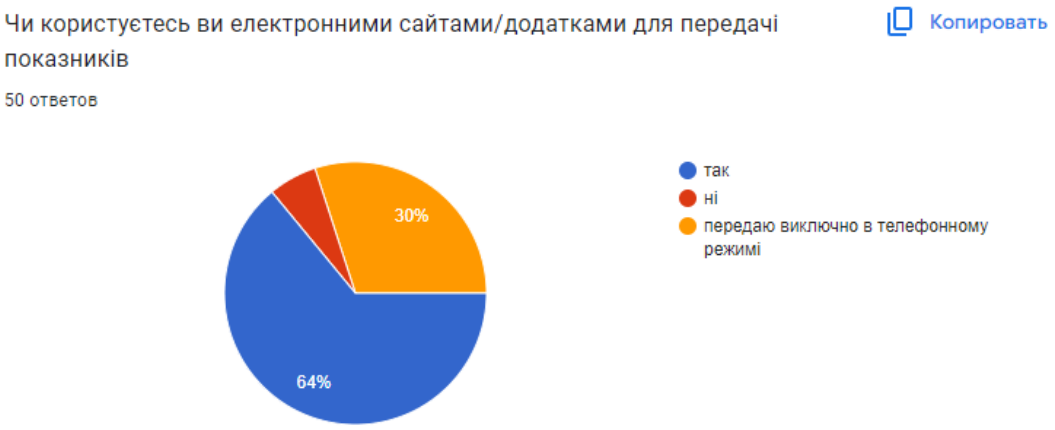

Рисунок 1.8 – Зацікавленість використання веб-ресурсів

Останнім питанням було досліджено чи зрозуміло людям як передавати показники в онлайн режимі та основні проблеми при передачі таким чином (діаграма 1.5). Більшість з респондентів, а саме 48%, відповіли, що при передачі показників не виникає проблем. Інші 34% та 18% відповіли, що мають проблеми та хотіли б дещо змінити. Переглянувши основні заперечення було зроблено висновок, що більшість незадоволена інтерфейсом додатків та сайтів. Через великий потік інформації опублікованої на сторінках, респонденти втрачають важливі повідомлення.

Рисунок 1.8 – Вивчення проблематики

1.5 Постановка задачі

Метою роботи є розроблення інформаційної системи CRM для підприємства електропостачання «СумЕнерго» використовуючи сучасні мови програмування. Ця система має виконувати функції додаткової платформи для забезпечення якісної і успішної діяльності на просторах Всесвітньої мережі Інтернет.

Реалізація поставленої мети визначає наступні задачі:

- Пошук та дослідження необхідних інструментів;
- Створення карти сайту;
- Розробка бази даних для збереження інформації, необхідної для роботи веб-системи;
- Розробка зрозумілого інтерфейсу веб-системи;
- Проектування архітектури веб-системи;
- Розробка логіки серверної частини веб-системи;
- Тестування з метою перевірки його функціональності та надійності.

Данна платформа буде інформаційною системою, в якій будуть описані всі можливі послуги підприємства, інформація для користувачів, доступ до власного кабінету для передачі показників, контролю витраченої енергії. Однією з найважливіших задач є розробка зручного та якісного інтерфейсу, який відіграє важливу роль у сучасному онлайн маркетингу.

2 ВИБІР ПРОГРАМНИХ ЗАСОБІВ

2.1 Вибір засобів для реалізації мети

На сьогоднішній день не виникає питань щодо переваг створення веб-додатків як для користувача, так і для бізнесу. Якісна система взаємовідносин з користувачами гарантує відмінну функціональність, ефективну систему підтримки користувачів та унікальний дизайн, орієнтований на користувачів, за рахунок використання новітніх технологій. Існує багато різних способів розробки та створення інформаційних систем CRM: створення за допомогою фреймворків, створення за допомогою сайтів конструкторів тощо. Одним з найпопулярніших є використання фреймворків, вони надають кращі практики веб-розробки, які дозволяють підтримувати проект протягом тривалого часу. Фреймворк – це стандартизований набір концепцій, практик, критеріїв для вирішення різних задач. Основною метою фреймворків є надання загальної структури, щоб не витрачати час на створення її з нуля, а використовувати наданий код повторно [6].

Враховуючі зазначені вище задачі, було обрано стек програмування MERN. MERN – це набір технологій, здебільшого фреймворків, які дозволяють пришвидшити розробку додатків. Основна мета та перевага використання даного стеку – розробка додатків лише за допомогою мови JavaScript. Це пов'язано з тим, що всі 4 мови, застосовані в стеку, базуються на JavaScript [7]. У стеку використовуються наступні технології: MongoDB, ExpressJS, ReactJS, NodeJS.

ReactJS – це фреймворк та бібліотека JavaScript з відкритим вихідним кодом, яка відноситься до клієнтської частини проекту та легшої взаємодії з серверною частиною. Фреймворк використовується для швидкого та ефективного створення інтерактивних користувацьких інтерфейсів і веб-додатків зі значно меншою кількістю коду. За допомогою ReactJS інтерфейс користувача не розглядається як єдине ціле, а розділяється на окремі багаторазові компоненти, які формують будівельні блоки всього інтерфейсу [8]. В фреймворку також

використовуються елементи стандартизованої мови розмітки документів HTML. За допомогою конструкцій HTML, зображення та інші об'єкти, такі як інтерактивні форми, можуть бути вбудовані у сторінку. Стилізація компонентів відбувається з використанням каскадних таблиць стилів, більш відомих як CSS. Такий процес розробки, який використовується для того, щоб зробити веб-сторінку більш представницькою. Це допомагає в розробці якісного та креативного веб-сайту, який вражає аудиторію та привертає увагу. Таким чином, на сьогоднішній день це невід'ємна частина створення веб-сайтів, якою не слід нехтувати [9].

MongoDB – це NOSQL система керування базами даних, в якій дані зберігаються у вигляді документів, що мають пари ключ-значення, подібні до об'єктів JSON. Частіше організації використовують для спеціальних запитів, індексування, балансування навантаження, виконання JavaScript на стороні сервера [10].

Використання MongoDB у парі з JavaScript дозволяє використовувати API – програмний інтерфейс застосунку, який дає змогу програмним компонентам взаємодіяти один з одним, використовуючи набір визначень і протоколів [11,12]. Загальну роботу API можливо розглянути на рисунку 2.1:

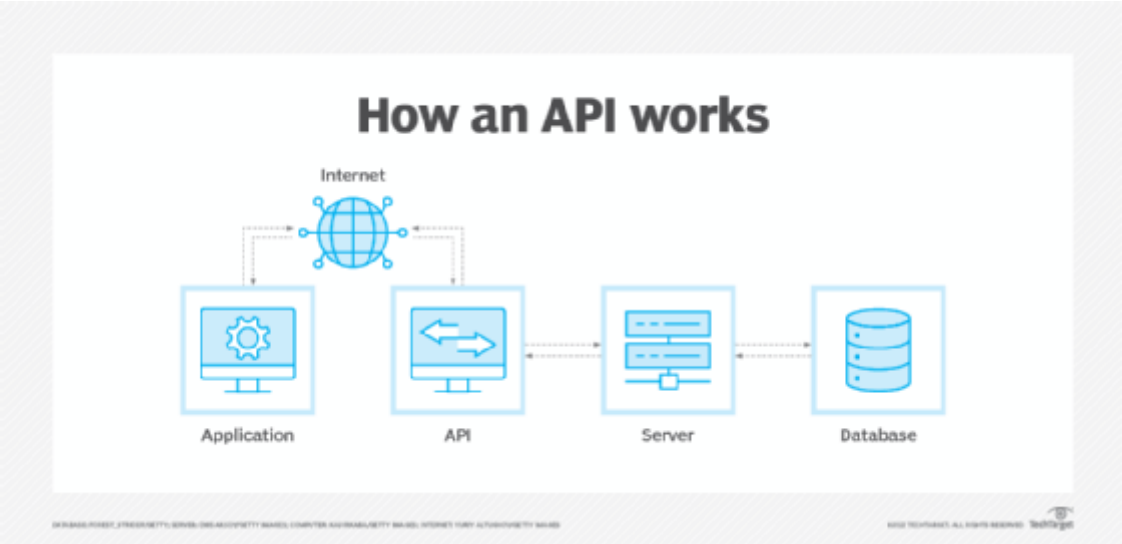

Рисунок 2.1 – Використання API

В проекті використовується принцип REST API, який працює за наступним алгоритмом:

Клієнт надсилає запити на сервер у вигляді даних

Сервер використовує вхідні дані клієнта для запуску внутрішніх функцій

Сервер повертає вихідні дані назад клієнту.

REST визначає набір функцій, таких як GET, PUT, DELETE та здійснює обмін даними за допомогою HTTP. Головною особливістю REST API є безстатусність, що означає, що сервери не зберігають клієнтські дані між запитами. Клієнтські запити до сервера схожі на URL адреса, які ми можемо вводити в браузері, щоб відвідати будь-який веб-сайт. В той час як відповідь від сервера – це звичайні дані, без типового графічного відображення веб-сторінки [13].

Наступним компонентом стеку MERN є середовище виконання JavaScript – NodeJS. Технологія є крос-платформною і працює не тільки на Windows, а й Linux, Unix та macOS та дозволяє користувачам запускати код на сервері. Середовище поставляється з менеджером пакетів вузлів або скорочено npm, що дозволяє користувачам вибирати з широкого спектру вузлових модулів або пакетів, такі бібліотеки також використано в проекті [14]. NodeJS має архітектуру, керовану подіями, здатну до асинхронного вводу-виводу. Ці дизайнерські рішення спрямовані на оптимізацію пропускної здатності та масштабованості у веб-додатках з великою кількістю операцій вводу/виводу. В нашому випадку саме це підходить для нас аби розробити серверну частину веб ресурсу [15].

Останнім використаним компонентом стеку є фреймворк NodeJS – ExpressJS. Він спрощує написання бекенд-коду, позбавляє необхідності створювати кілька модулів Node. Для забезпечення точності коду ExpressJS пропонує ряд проміжного програмного забезпечення, яке також було використано в програмній реалізації [16].

2.2 Проектування бази даних

База даних є невід'ємною складовою будь-якого проекту. Вона може зберігати не тільки дані про клієнтів, а й дані про послуги, продукцію тощо. Існують різні системи керування базами даних, такі як: MySQL, PostgreSQL, Oracle, Microsoft SQL Server, SQLite. Але всі вони відрізняються своїм функціоналом та предметною областю застосування. З попередніх досліджень ми виявили, що будемо використовувати нереляційну базу даних – MongoDB.

Перед початком роботи потрібно спроектувати моделі, які потім будемо застосовувати у своєму проекті (рис. 2.2).

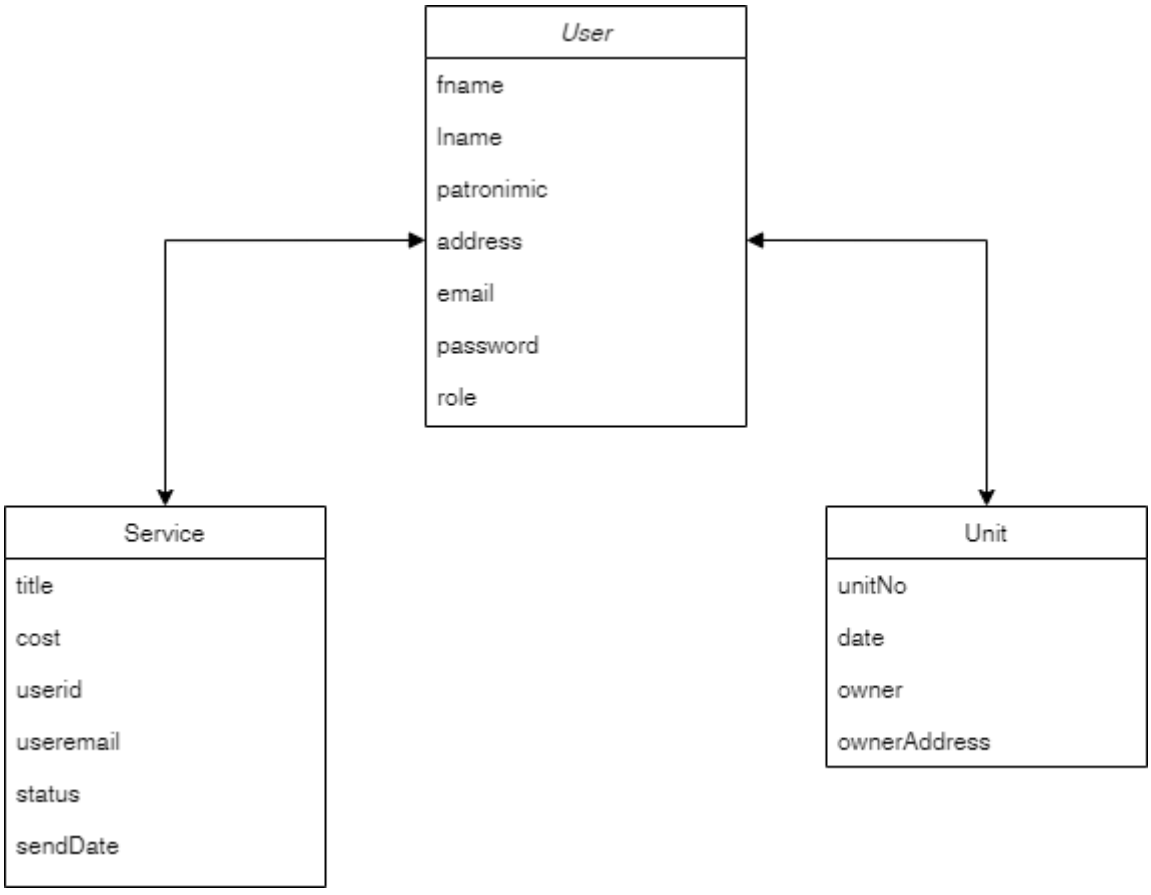

Рисунок 2.2 – Спроектована схема бази даних

Як бачимо з рисунку, ми маємо 3 моделі, з якими будемо працювати у серверній та клієнтській частині.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ

3.1 Інформаційна модель та структура веб-сайту

Веб-сайт – це комплекс веб-сторінок, зображень та анімацій, електронних файлів та різних документів, баз даних, які розміщені у Всесвітній мережі Інтернет. Це структурована кореляційна інформаційна платформа, що має деяке оформлення та власний інтерфейс функціонування з даними та інформацією.

Створення інформаційної моделі веб-сайту це перше, що необхідно виконати при написанні веб-сайту будь-якого типу. Інформаційна модель складається з основних елементів веб-сторінки. Розміщення та порядок розділів веб-сторінки відображає структура веб-сайту [17].

Розрізняють два типи структури веб-сайту: внутрішня і зовнішня. До внутрішньої структури входить розроблення бізнес логіки веб сайту, категорій, послуг тощо. Веб сайт для підприємства «СумЕнерго» має ієрархічну структуру, так як існує багато розділів, підрозділів. Такий тип структури є найоптимальнішим для CRM систем, так як користувач завжди має вибір та можливість перейти в будь-який розділ, переглянути інформацію як з головної сторінки так і з будь-якої іншої [18]. Така внутрішня структура показана на рисунку 3.1.

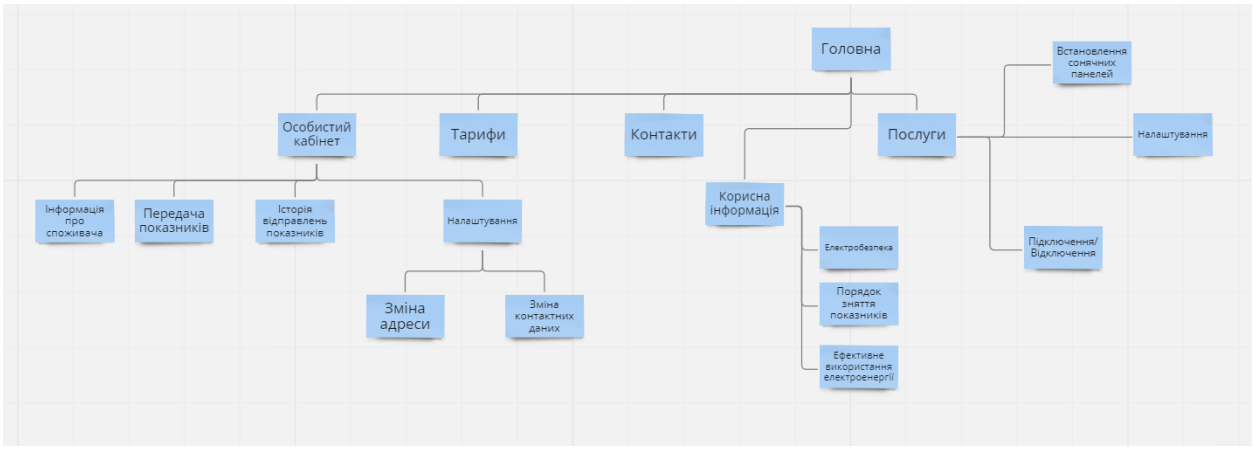

Рисунок 3.1 – Внутрішня структура сайту

Вибір правильних кольорів для дизайну сайту є також вирішальним значенням для успіху в Інтернеті. Кольори можуть бути найпотужнішим інструментом для отримання реакції від цільової аудиторії. Ви можете використовувати кольори, щоб викликати у відвідувачів емоції або відповісти на заклик до дії на вашому сайті. Колір допомагає нам обробляти і зберігати зображення більш ефективно, ніж безбарвні (чорно-білі) зображення [19]. Так як наша цільова аудиторія переважно повнолітні та люди старшого віку, потрібно використати кольори, які б означали надійність. Щодо палітри кольорів використаних в дизайні, в проекті використано наступні кольори (рис. 3.2):

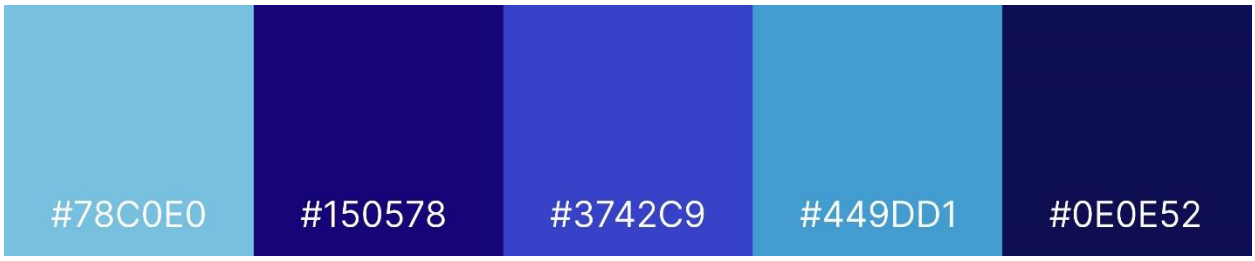

Рисунок 3.2 – Використані кольори на сайті

За психологією кольорів саме синій колір та його відтінки передають надійність. Вважається, що саме цей колір викликає у людей довіру до постачальників послуг та вони можуть бути більш схильні до покупки або замовлення будь-яких послуг, оскільки вони сприймають сайт, як надійний та професійний. Також синій колір асоціюється з оптимізмом. Коли люди бачать певні відтінки синього кольору, вони з більшою ймовірностю відчувають піднесений настрій. Такий спосіб також може допомогти людям відчути оптимізм щодо продукції або послуг, які сайт постачає і підвищує ймовірність замовлення або реєстрації [20].

Зовнішня структура веб сайту передбачає модель сайту та відповідає його дизайну. Вона визначає розміщення даних на веб сторінці при певному виборі пункту меню сайту. Зовнішня структура створюється для полегшення користування веб-сайтом користувачами та повторює навігацію сайтом. Також розроблення такої структури дає змогу веб програмісту краще орієнтуватися при

створенні веб ресурсу. Таким чином, цей вид структурування сайту відповідає за розташування елементів на веб-сторінці.

Основними елементами зовнішньої структури є:

- Шапка сайту. Складається з логотипу або назви сайту. Також там може розміщуватися додаткові посилання такі як, інформація про компанію, меню, контактні дані та інше;
- Меню. Засіб для переходу до каталогу товарі або інформації;
- Центральна частина або контент. Містить основні елементи сторінки, які доступні користувачу, такі як послуги, різна інформація у вигляді тексту, онлайн підтримка, зображення або відео;
- Футер сайту. Розміщується у нижній частині сайту, та наповнений контактними даними, посиланнями на деякі пункти та авторські права.

Важливими чинниками для успішної структури сайту є зручність користування, інтуїтивно зрозуміле розташування елементів на веб-сторінці, яскравість дизайну та інше. Саме такої тенденції дотримано в ході розробки зовнішньої структури сайту. Інформаційна система розділена на наступні блоки:

- Головна сторінка, якою переважно можуть користуватися гості, які тільки зайшли на сайт (рис. 3.3). Вона включає в себе:
  1) Сторінку з актуальною інформацією про послуги, які надає компанія електропостачання (рис 3.4);
  2) Сторінку з інтерактивним ботом для надання відповідей на запити користувачів (рис. 3.5).
- Особистий кабінет клієнта. В кабінеті представлена загальна спожита електроенергія і останні оформлені замовлення (рис. 3.6). До кабінету також входять наступні сторінки:
  1) Сторінка для можливості передачі показників та приблизного розрахунку вартості спожитої електроенергії (рис. 3.7);
  2) Сторінка з актуальними послугами та можливість створити замовлення на одну з представлених послуг (рис. 3.8).

– Особистий кабінет менеджера. В кабінеті також представлена основна інформація про менеджера та список всіх актуальних користувачів, які вже зареєстровані в системі (рис. 3.9). Також особистий кабінет менеджера включає в собі наступні сторінки:
  1) Сторінка з актуальними переданими користувачами показниками електроенергії (рис. 3.10);
  2) Сторінка з актуальними замовленими послугами та можливістю керувати ними: видаляти, змінювати статус (рис. 3.11).

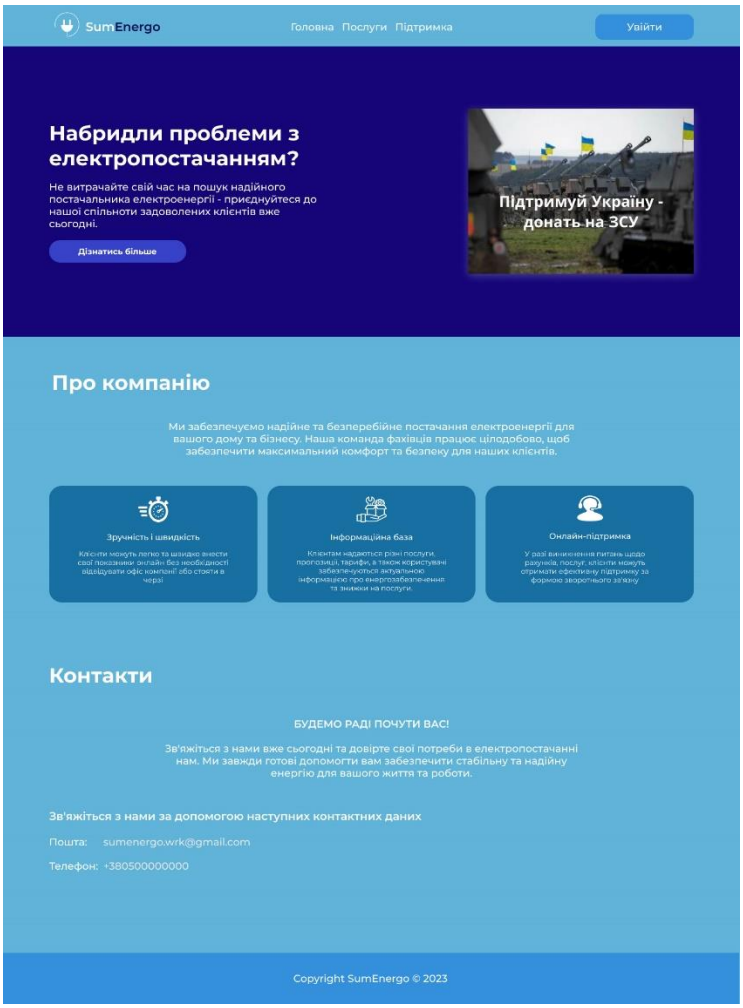

Рисунок 3.3 – Головна сторінка сайту

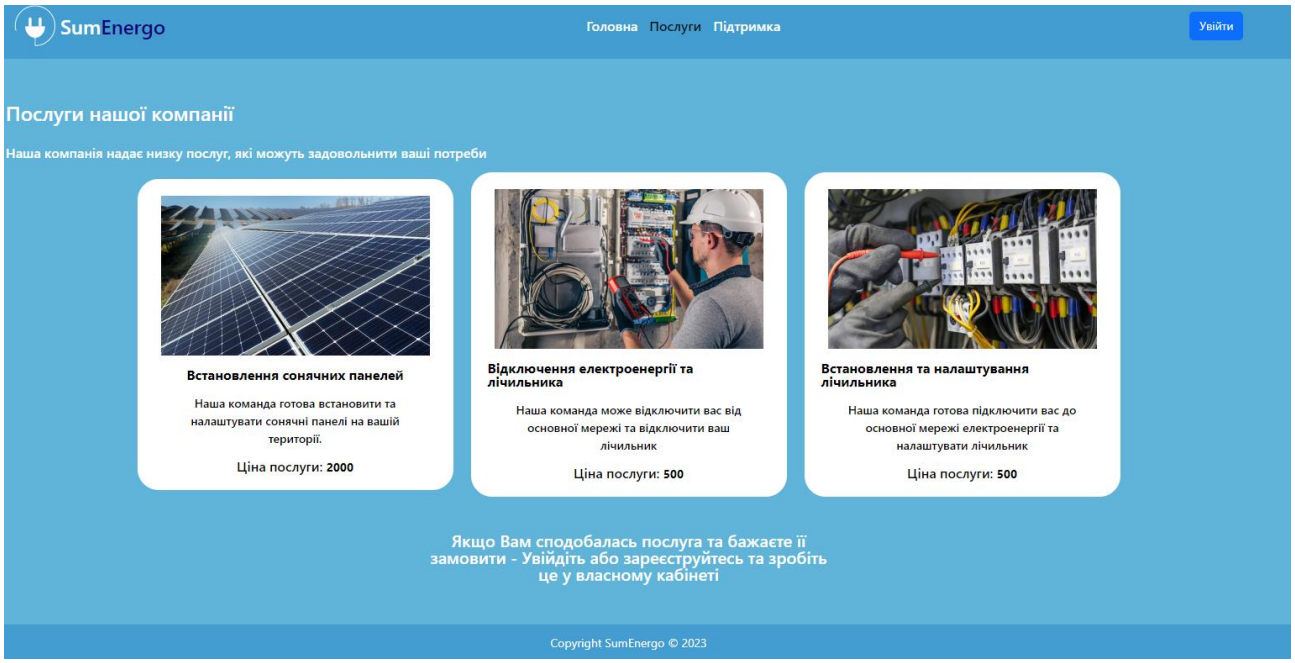

Рисунок 3.4 – Гостьова сторінка послуг

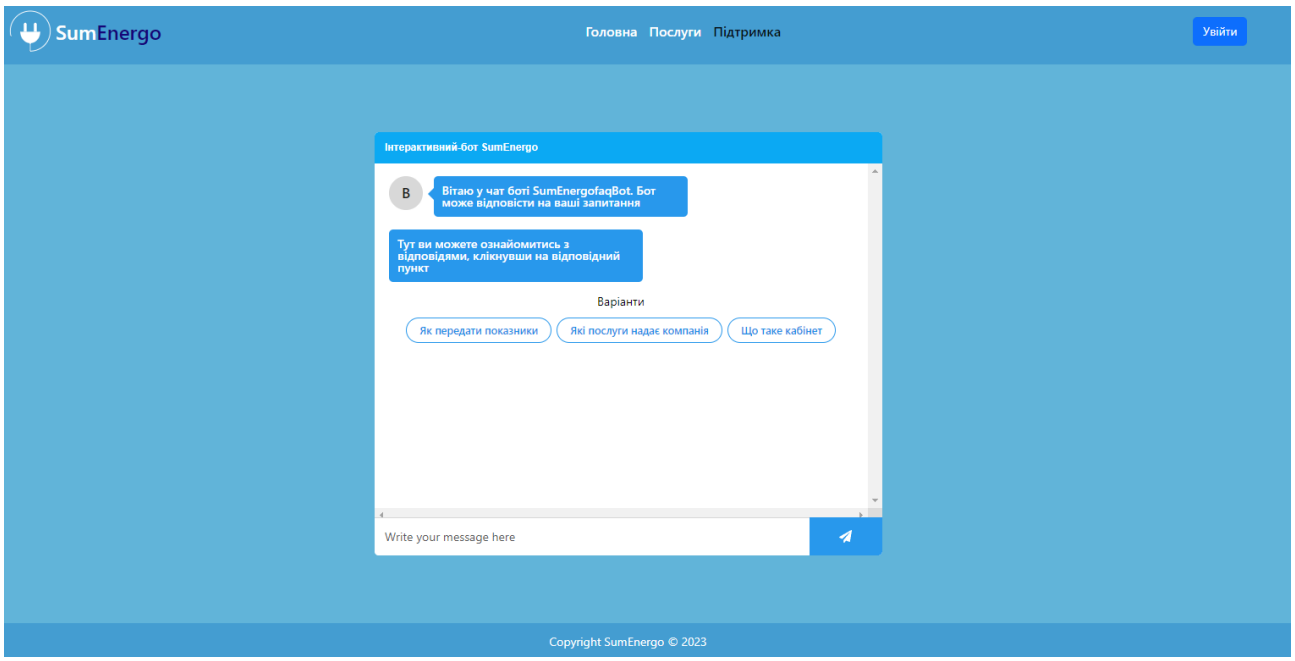

Рисунок 3.5 – Сторінка інтерактивного боту

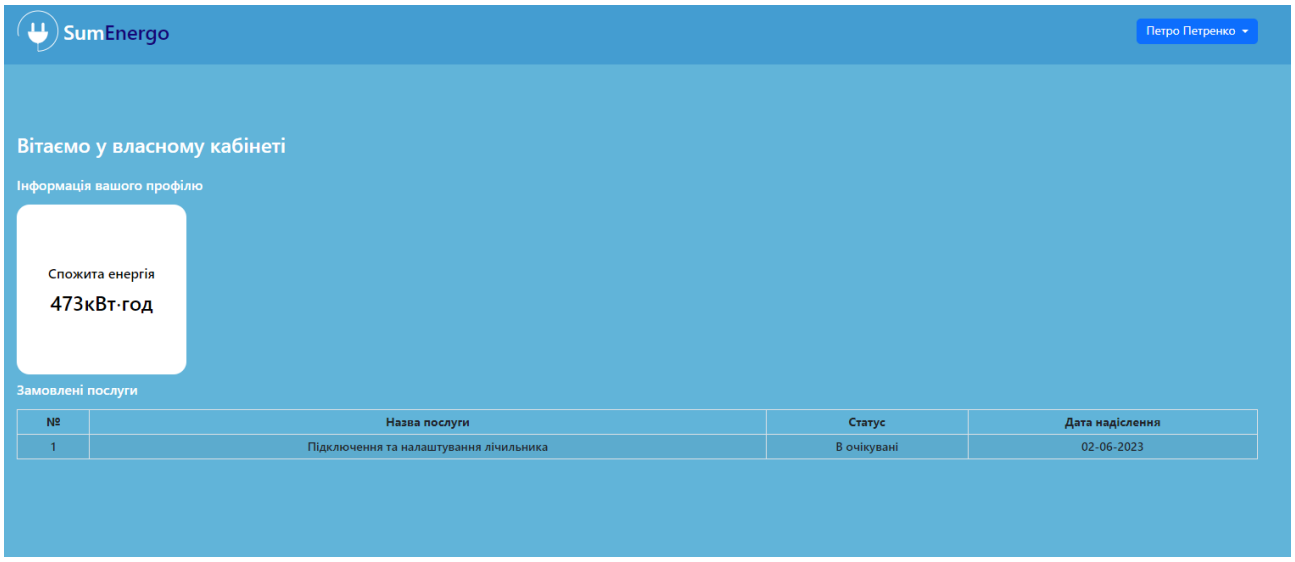

Рисунок 3.6 – Головна сторінка особистого кабінету

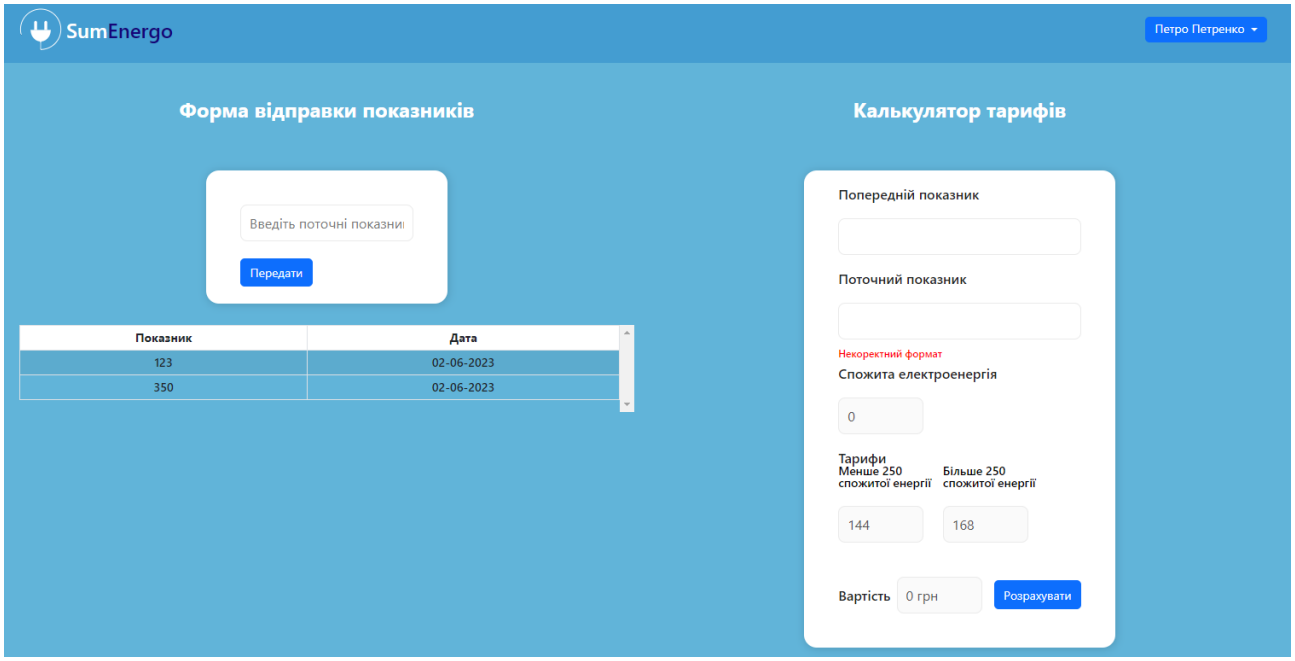

Рисунок 3.7 – Сторінка передачі показників та розрахунку вартості

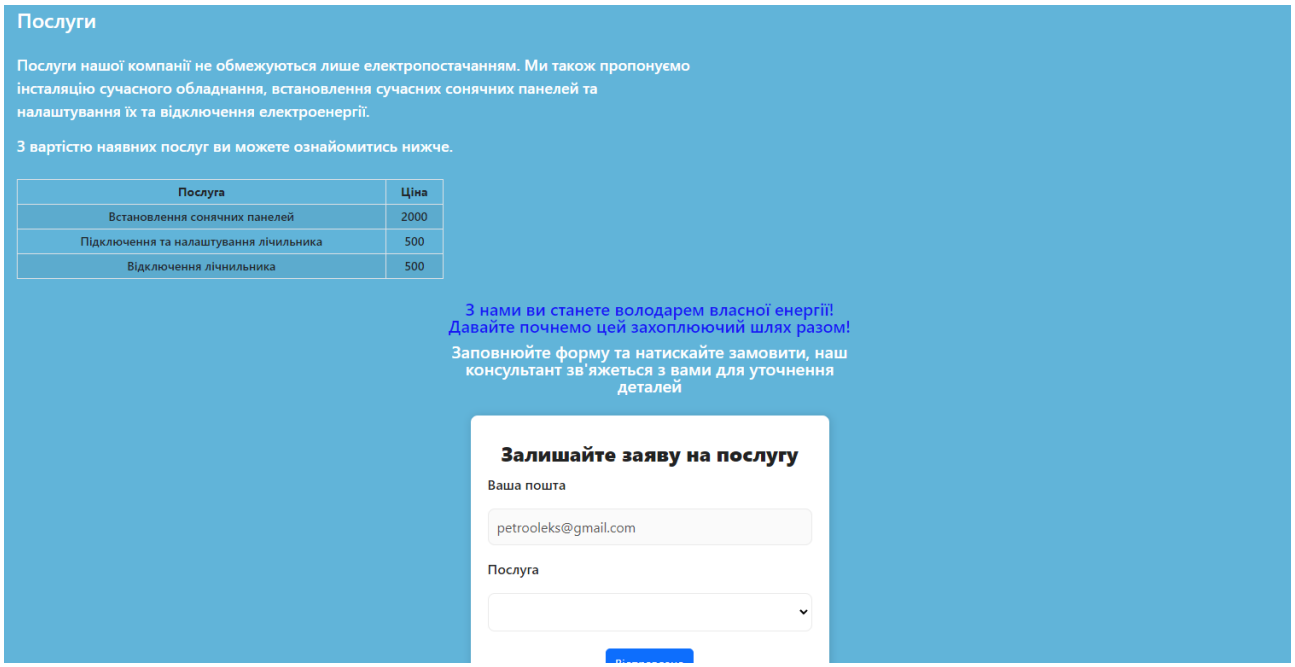

Рисунок 3.8 – Сторінка послугів користувача та форма для створення замовлення на послугу

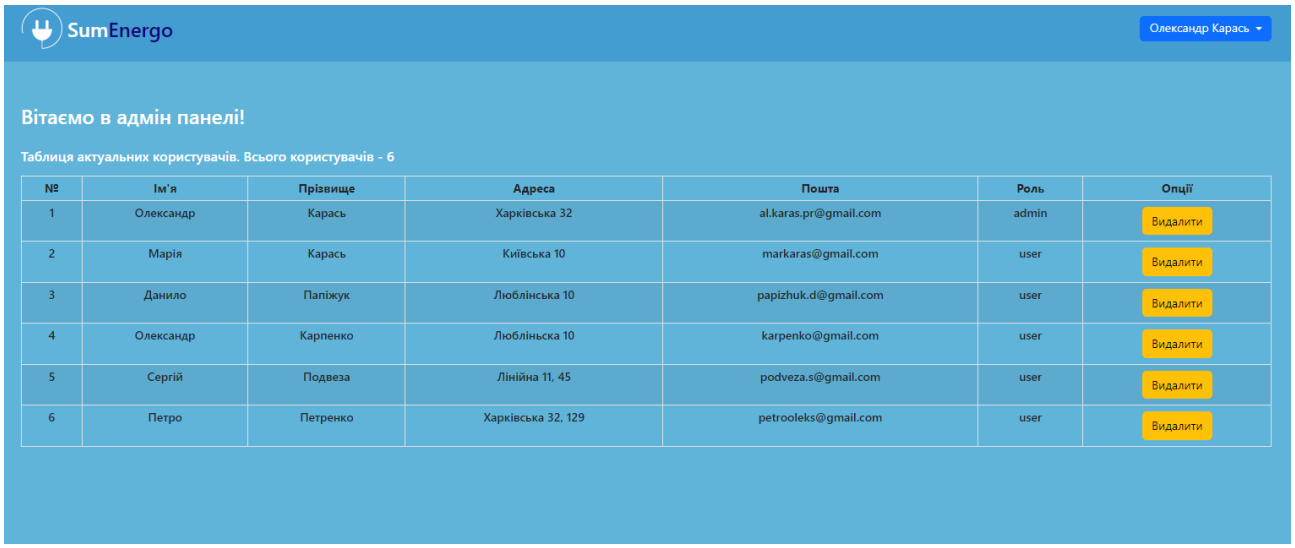

Рисунок 3.9 – Головна сторінка менеджера

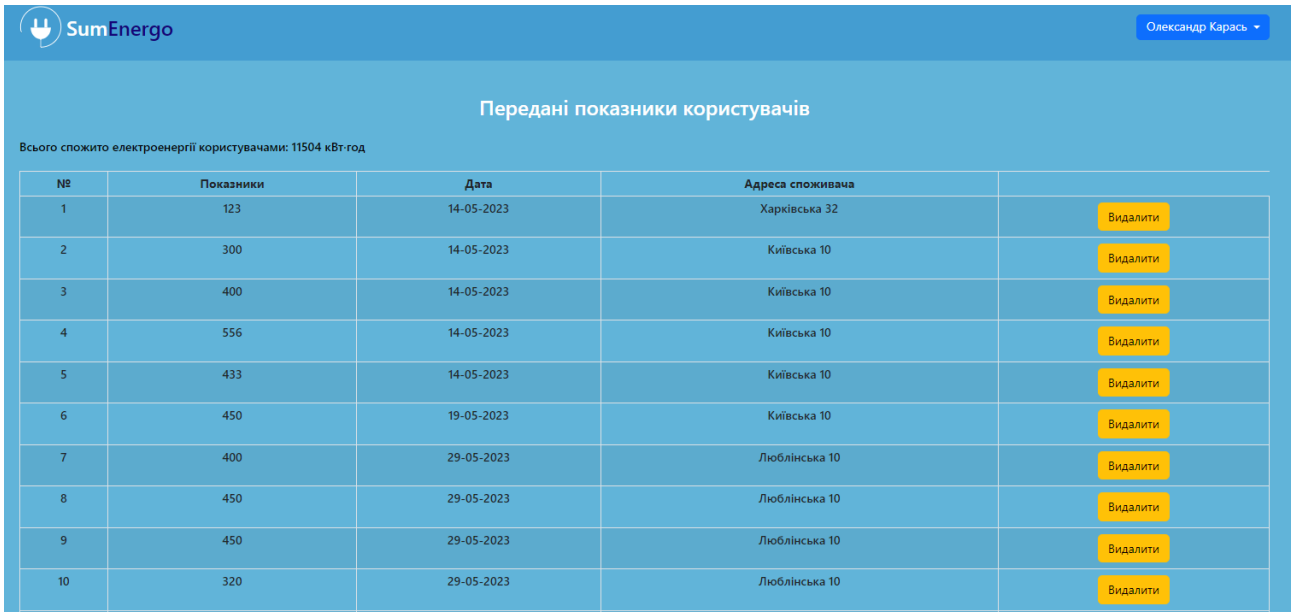

Рисунок 3.10 – Сторінка переданих показників всіма користувачами системи

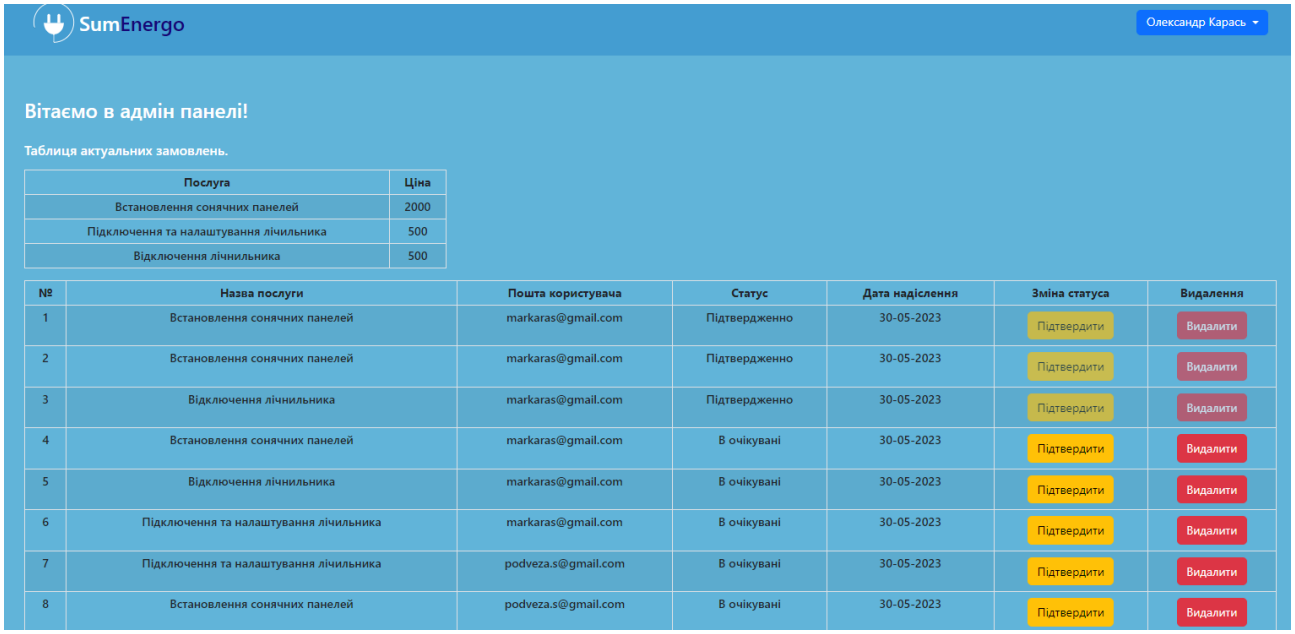

Рисунок 3.11 – Сторінка з замовленими користувачами послугами

Для користувачів така структура веб-сайту та надана інформація та навігація сайтом буде абсолютно зрозумілою на візуальному та інтуїтивному рівні. Ефективність якісно структурованої CRM системи пояснюється запорукою успішної оптимізації, полегшенням просування веб-сайту, високою відвідуваністю користувачами та стабільною роботою.

3.2 Налаштування бібліотек та модулів

Для досягнення швидкодії та надійності роботи системи доцільно використовувати необхідні програмні бібліотеки та модулі, які можливо знайти у реєстрі програмного забезпечення NPM [21]. До переваг використання таких пакетів входять:

- Прискорення роботи при розробці програмного забезпечення;
- Керування декількома версіями коду та залежностями;
- Оновлення додатку, так як оновлюється базовий код;
- Прискорення швидкодії застосунку.

Для підключення пакетів та бібліотек використовують консольну команду:

```
npm install <назва бібліотеку/модулю/пакету>
```

В проекті використано важливі бібліотеки, які надають додаткові функції та можливості для реалізації роботи з базою даних, валідації даних, взаємодії клієнта з серверною частиною, роботи зі стилями, розробки клієнтської та серверної частини, роботи з датами, надання безпеки та налаштування автентифікації. До таких бібліотек відносяться: Express, BcryptJS, JsonWebToken, Cors, Mongoose, Axios, React (React-dom, react-router-dom, React-hook-form), React-bootstrap та react-router-bootstrap, React-redux, Sass. Зберегти та налаштувати дані пакети можливо у конфігураційному файлі package.json (Додаток А – серверна частина, Б – клієнтська частина), який зберігає основну інформацію про додаток, а саме опис того, як взаємодіяти з додатком та запускати його [22]. Для застосування пакетів в програмному коді використовується команда: `import <назва компоненту> from «<назва пакету>»`

3.3 Визначення архітектури та структури додатку

Гарно спроектована архітектура папок впливає на ефективність, обслуговування та швидкодію продукту. Такий спосіб забезпечує зрозумілу і логічну організацію файлів та компонентів системи, полегшує розробку сумісно з розробниками та забезпечує навігацію та управління продуктом. Правильна структура модулів та каталогів також допомагає забезпечити чистоту коду, дотримання стандартів програмування, що впливає на якість та підтримку продукту протягом його життєвого циклу.

Так в нашому проекті використовуються різні структури папок для серверної (рис. 3.12) та клієнтської (рис. 3.13) частини.

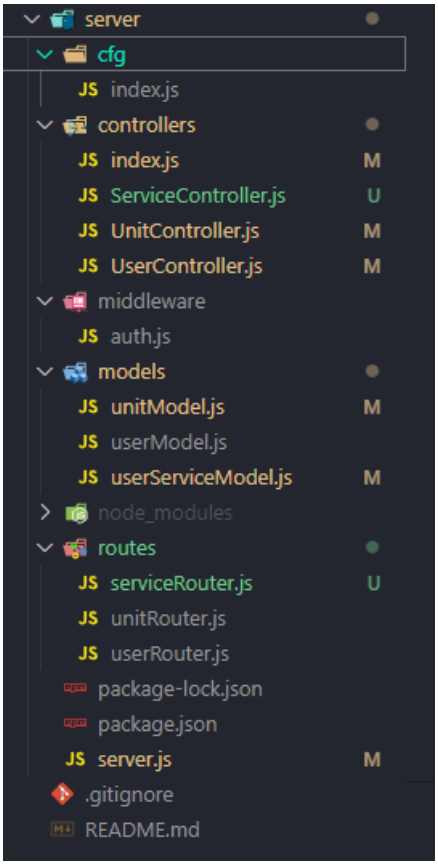

Рисунок 3.12 – Структура папок серверної частини

Назви папок відповідають за свої обов’язки та функції в програмній реалзіації:

– Cfg – містить в собі конфігураційний файл з назвами хосту,

налаштування роботи з БД та JWT токеном;

– Controllers – містить файли роботи з базою даних та моделями;
– Middleware – містить файли для роботи з автентифікацією користувача на стороні серверу;
– Models – містить моделі бази даних mongoDB;
– Routes – містить файли для роботи з напрямками виконання API запитів.

В основному файлі server.js відбувається підключення модулів та використання файлів з перечислених папок для належної роботи з сервером.

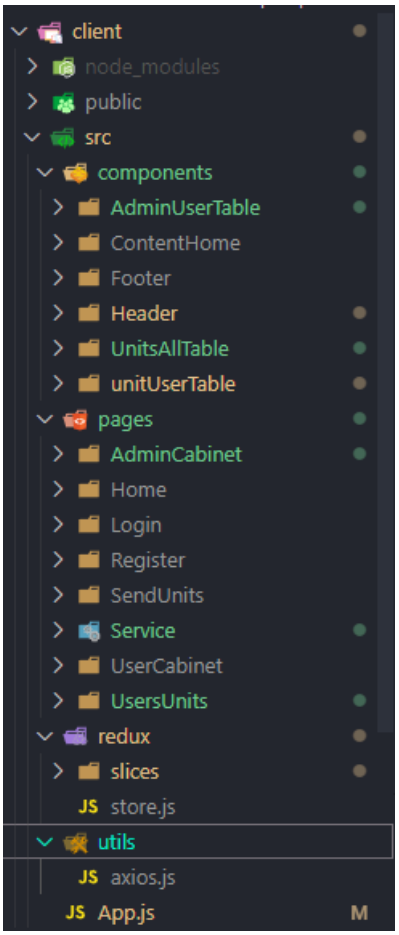

Рисунок 3.13 – Структура папок клієнтської частини

В клієнтській частині структура папок включає в собі папки компонентів, сторінок, роботу з отриманням даних з серверу та конфігураційний файл для перевірки автентифікації користувача та сумісної роботи серверу та клієнту.

3.4 Налаштування взаємодії з базою даних та API

Для забезпечення коректної роботи з базою даних, необхідно налаштувати конфігураційний файл з назвою бази даних, портом та хостом до якого потрібно підключитись. Ці конфігураційні дані зберігаються у папці cfg і мають наступний вигляд (рис. 3.14).

```
module.exports = {
    "host": {
        PORT: 3001,
    },
    "database": {
        HOST: '127.0.0.1',
        PORT: '27017',
        DBNAME: 'sumendb',
    },
```

Рисунок 3.14 – Конфігураційний файл з даними для підключення до БД

З'єднання з базою даних відбувається в основному файлі серверу і виглядає наступним чином (рис. 3.15).

```
/* CONNECT TO MONGODB */
const mongoose = require('mongoose');
const MONGOURL = `mongodb://${config.database.HOST}:${config.database.PORT}/${config.database.DBNAME}`;
mongoose
    .connect(MONGOURL, {
        useNewUrlParser: true,
    })
    .then(() => console.log('Connected to DB successfully'))
    .catch(() => console.log('Cannot connect to DB'));
```

Рисунок 3.14 – Підключення до БД

Після підключення до БД також потрібно створити моделі за концептуальною моделлю бази даних, контроллери для роботи з цими моделями та роути (напрямки) (Додаток Б) за якими будуть відбуватися взаємодія з даними на стороні серверу та клієнту.

Мій проект містит 4 моделі (Додаток А):

– Модель користувача – Додаток А (UserModel.js). Містить інформацію про споживача: П.І.Пб., адресу, адреса електронної пошти, телефон, пароль та роль;

- Модель показників – Додаток А UnitModel.js. Містить в собі номер показника, дату надсилання, адреса споживача, який передав показник та ідентифікаційний номер користувача;
- Модель послуги та взаємодії користувача з послугою – Додаток А userServiceModel.js. Містить наступну інформацію: ідентифікаційний номер користувача, тип послуги, електронна адреса користувача, статус замовлення, дата відправки;
- Модель цінників для послуг – Додаток А pricesModel.js. Містить інформацію про назву послуги та її вартість.

Після створення моделі потрібно налаштувати контроллери та напрямки для застосування API (Додаток Б).

Також потрібно забезпечити автентифікацію користувача з використанням JWT Tokena. JWT – відкритий стандарт, який визначає компактний і автономний спосіб безпечної передачі інформації між сторонами у вигляді об’єкта JSON [23]. Токен використовується при авторизації призначаючи користувача секретний токен, який має строк придатності 30 діб. Для перевірки чи автентифікований користувач чи ні – використовується наступний модуль:

```
const jwt = require('jsonwebtoken');
const config = require('../cfg');
const User = require('../models/userModel');

module.exports = async (req, res, next) => {
    try {
        const token = req.headers.authorization.split(' ')[1];
        if (!token) {
            res.status(401).json({ message: "Please auth using a
valid token" });
        }
        const decodedToken = jwt.verify(token, config.jwt.TOKEN);
        const user = await User.findById(decodedToken.usid);
        req.user = user;
        next();
    } catch (error) {
        res.json({ message: "Автентифікація не пройдена. Немає
доступу" });
    }
}
```

Таким чином відбувається налаштування роботи з базою даних.

Щоб налаштувати правильне отримання даних з серверної частини на клієнтській потрібно налаштувати axios - HTTP-бібліотека на основі promise, яка дозволяє розробникам надсилати запити до власного або стороннього сервера для отримання даних [24]. Відповідні функції описуються наступним кодом:

```
import axios from 'axios';

const instance = axios.create({
    baseURL: `http://localhost:3001/api`
});

instance.interceptors.request.use((config) => {
    config.headers.Authorization =
window.localStorage.getItem("token");
    return config;
})

export default instance;
```

Однієї http бібліотеки для роботи з даними не вистачить, тому використовуються компоненти Redux – фрагменти, частини коду Redux, які відносяться до певного набору даних і дій в межах стану сховища (Додаток В).

Для відображення даних на сторінках також використовуються хуки UseSelector, useEffect та useDispatch [25,26]. За допомогою хуків ми можемо отримати дані про користувача, його роль, дані про показники, послуги тощо, шляхом використання функцій та станів фрагментів Redux. На приклад ми можемо перевірити чи авторизований користувач адмін та направити його на потрібну сторінку (використані функції та стани фрагменту loginSlice.js Додатку Г):

```
const isLogged = useSelector(selectIsLogged);
const isAdmin = useSelector(selectIsAdmin);
const navigate = useNavigate();
const dispatch = useDispatch();
useEffect(() => {
        if (isLogged && isAdmin) {
            navigate('/adminCabinet')
        } else if (isLogged && !isAdmin) {
            navigate('/userCabinet')
        }
    }, [isLogged, isAdmin, navigate])
```

Таким чином дані з серверу використовуються в клієнтській частині.

3.5 Створення та налаштування інтерактивного бота

Щоб забезпечити потреби користувачів веб-ресурс має мати будь-яку підтримку. Це може бути онлайн консультування з реальним менеджером, бот в соціальних мережах, або консультація по телефону. Так як наш продукт розробляється для користувачів, які користуються більше мобільним пристроєм було спроектовано інтерактивний чат-бот для відповідей на запитання користувачів.

За основу було взято компонент react-chatbot-kit та методи, які наявні в компоненті [27]. Розробка передбачає створення штучного інтелекту, який буде відповідати на питання користувачів в форматі «питання» – «відповідь». Реалізація передбачає налаштування конфігураційного файлу, віджетів для швидкого доступу до питань та інших функцій (Додаток Г).

3.6 Механізми авторизації та реєстрації

Важливим компонентом в розробці інформаційних систем є авторизація та реєстрація. В момент реєстрації пароль шифрується за допомогою алгоритму Bcrypt та зберігається в зашифрованому вигляді в таблиці бази даних (Додаток Б – userController.js, register). В стані авторизації пароль розшифровується за допомогою функцій дефишровки того ж алгоритму та звіряється з введеним користувачем (Додаток Б – userController.js, login). Користувачу також присвоюється токен авторизації з терміном зберігання 30 діб. Такий спосіб допоможе користувачу залишатись авторизованим в системі. В такий спосіб забезпечується безпека та конфіденційність даних користувача в момент використання системи.

3.7 Кабінет користувача

В рамках розробки нашої системи кабінет користувача є невід’ємною складовою. Кабінет є основним компонентом системи і надає користувачам широкий спектр функцій та можливостей.

В особистому кабінеті користувач може переглянути загальну спожиту електроенергію (Додаток Б – unitController.js – getSumOfUnitsById) та статус замовлень на послуги (Додаток Б – serviceController.js – getServById). Користувач також може ознайомитись зі своїми попередніми показниками та передати новий показник (Додаток Б – unitController – createUnit) за допомогою форми (рис. 3.15).

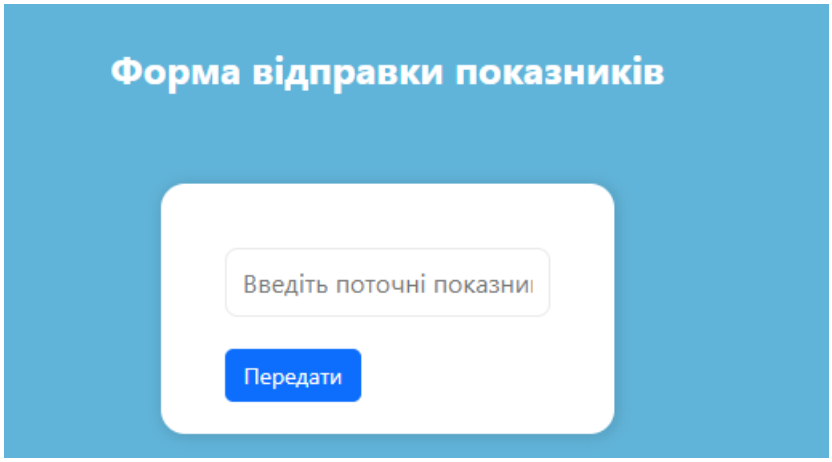

Рисунок 3.15 – Форма передачі показників

До того ж на сторінці передачі показників користувач може дізнатись приблизну вартість, яку має сплатити за спожиту електроенергію. Функція вирахування приблизної вартості має наступний вид:

```
const [prevnum, setPrevNum] = useState("");
const [nextnum, setNextnum] = useState("");
const [result, setResult] = useState("0");
const usedEnergy = nextnum - prevnum;
const CalculatePrices = () => {
     if (usedEnergy > 250) {
          let res = usedEnergy * 1.68;
          setResult(res);
        } else {
          let res = usedEnergy * 1.44;
          setResult(res);
        }
    }
```

Також користувач має можливість переглядати вартість послуги (Додаток Б – PricesController.js – getPriceList) та замовляти послугу за потребою (Додаток Б – ServiceController.js – createService).

3.8 Кабінет менеджера

Кабінет менеджера є також невід'ємною складовою розробки інформаційної системи для підприємства. Менеджер може не виходячи з дому ефективно контролювати користувачів та їх передані показники та замовлені послуги.

В кабінеті менеджер може переглядати актуальних користувачів системи (Додаток Б – userController.js – getUsers) та їх кількість (Додаток Б – userController.js – getAmountUsers) на головній сторінці. На «вкладці замовлені послуги» менеджер може переглядати замовлення користувачів (Додаток Б – serviceController.js – getService) та управляти ними: видаляти (Додаток Б – serviceController.js – deleteService) та змінювати статус на «підтверджений» (Додаток Б – serviceController.js – updateStatusServ). Після підтвердження або видалення замовлення на послугу таблиця автоматично оновлюється і кнопка підтвердження або видалення стає недоступною.

Також менеджер може контролювати передані показники (Додаток Б – unitController.js - getAllUnits) та бачити загальну спожиту енергію всіх користувачів (Додаток Б – unitController.js – getSumofAllUnits). Всі дані представлено в таблиці з полями: номер запису, номер показника, дата передачі, адреса споживача. За потребою менеджер може також видаляти переданий показник користувача (Додаток Б – unitController.js - deleteUnitById).

3.9 Тестування інформаційної системи

Перед релізом будь-якої інформаційної системи або додатку потрібно провести відповідно тестування системи аби передати замовнику вже готовий продукт і зберегти рейтинг компанії в майбутньому.

Тестування буде проводитись шляхом White-box тестування. Специфікацією такого тестування є те, що його проводять розробники, яким відома внутрішня структура та реалізація системи. Дані для тестування вибираються, ґрунтуючись на знанні коду, який будемо тестувати. Також ми знаємо яким повинен бути результат тестування [28].

В нашій системі ми будемо тестувати наступні компоненти: авторизація, реєстрація, передача показників, розрахунок приблизної вартості спожитої електроенергії, інтерактивний-бот, замовлення послуги.

Першим етапом є перевірка інтерактивного боту. За замовченням він надсилає заготовлений текст та кнопки для швидкого доступу до відповідей на запитання. Для виведення відповідей повідомлення має містити одну з команд «послуги», «показник»/«показники», «кабінет», «допомогти» / «питання». В разі відправки пустого повідомлення або повідомлення, яке не буде включати згадані команди бот буде виводити привітальне повідомлення з кнопками (рис. 3.16).

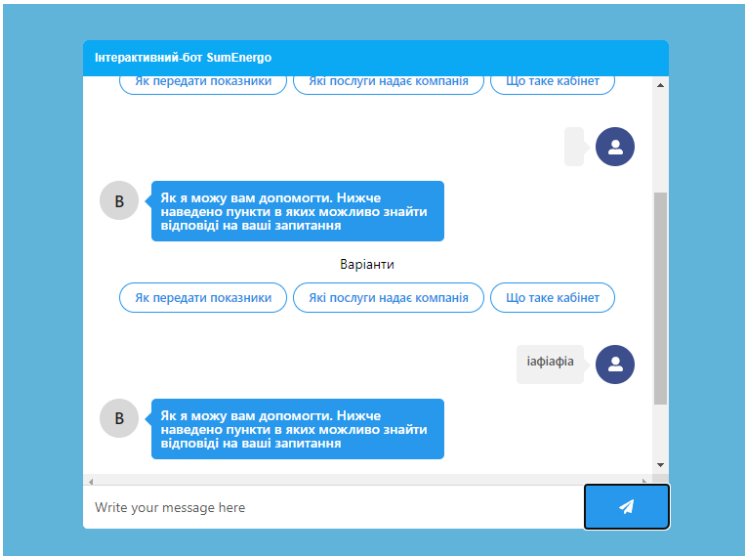

Рисунок 3.16 – Реакція бота на недопустимі дані

Перевірка на валідацію даних є важливим аспектом при розробці будь-якого ресурсу. В нашому програмному коді перевірка на валідацію на всіх формах відбувається за допомогою хуку useForm [29]. Частина хуку має набір даних за замовченням (наприклад значення для форми реєстрації):

```
const {
        register,
        handleSubmit,
        formState: {
            errors,
        }
    } = useForm({
        defaultValues: {
            fname: "",
            lname: "",
            patronimic: "",
            address: "",
            email: "",
            password: "",
        },
        mode: "onChange",
    });
```

Поля форми реєструються за допомогою функції register та в разі помилки зберігають повідомлення в змінній errors (рис. 3.17).

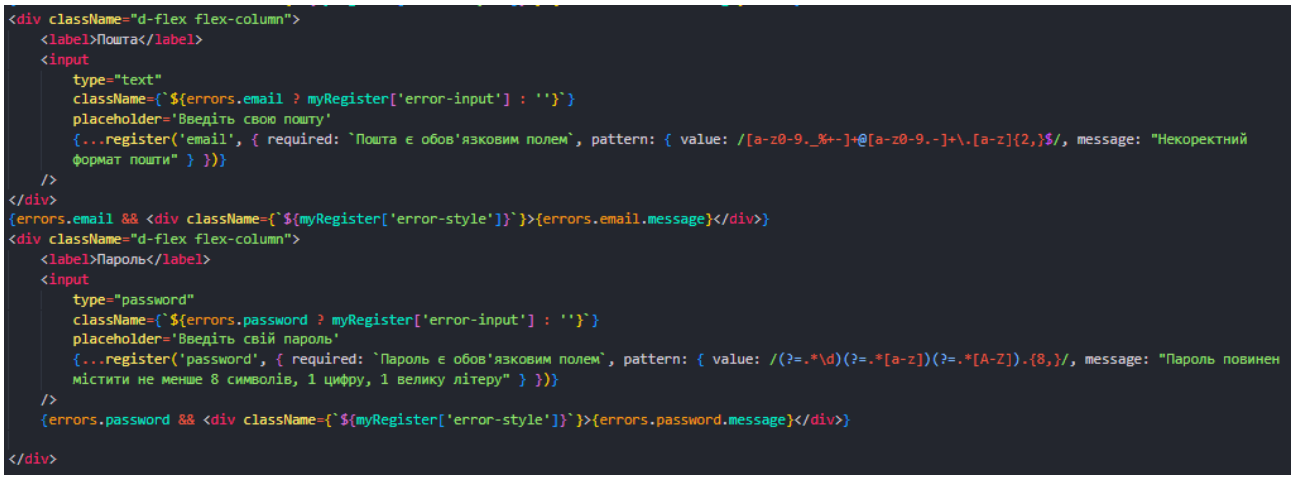

Рисунок 3.17 – Перевірка введенних даних на валідацію

Перевірка відбувається після відправки форми, якщо ж дані проходять валідацію, то форма відправляється, якщо ні – користувач отримує повідомлення.

Одним з наших етапів є перевірка компоненту реєстрації. Програмно перевірка відбувається за допомогою вже згаданого хуку та за допомогою regex

патернів – набір текстових значень, які описують набір рядків, що відповідають шаблону [30]. В нашому випадку перевіряються наступні стани:

– Пошта – має складатись з англійських літер та містити знак @ та точку в тексті після знаку. Пошта має бути нова та не співпадати з вже існуючою;
– Пароль – має містити не менше 8 символів, як мінімум 1 цифру та 1 велику літеру.

У разі пустих полів (рис. 3.18) та некоректного формату видається відповідне повідомлення (рис. 3.19).

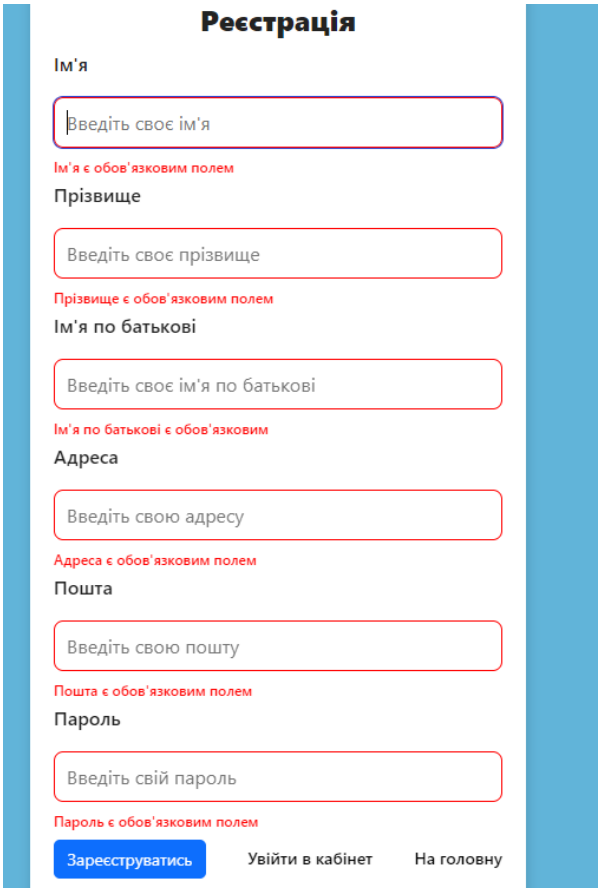

Рисунок 3.18 – Реєстрація з пустими полями

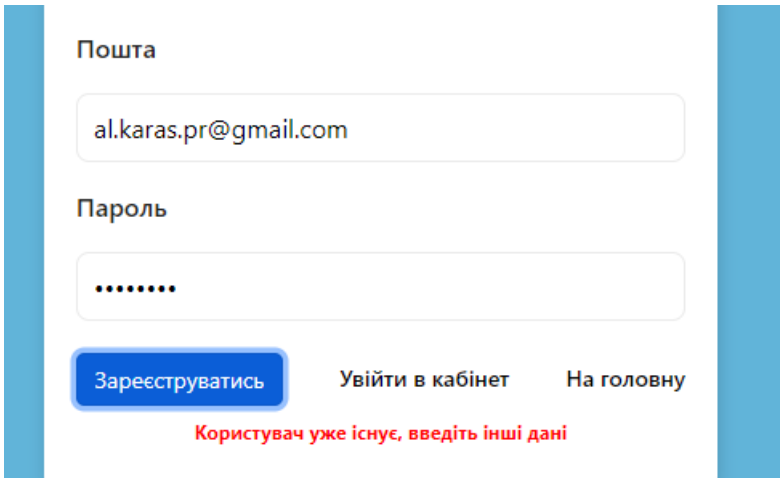

Рисунок 3.19 – Реєстрація з існуючою поштою

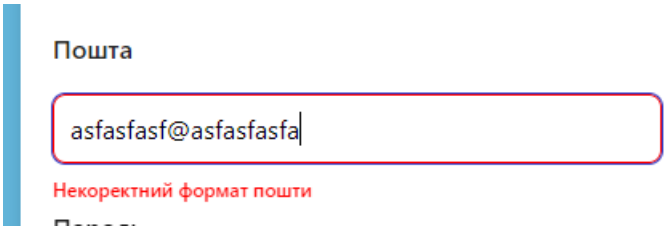

Рисунок 3.20 – Реєстрація з невалідною поштою

Наступним компонентом є авторизація. Вона має вже згадані регулярні вирази для перевірки пошти та паролю. При введенні неіснуючої пошти користувач отримує повідомлення «Користувач не існує» (рис. 3.21). В разі введення некоректного паролю або пошти виводиться повідомлення «Некоректний формат пошти», як це було на формі реєстрації.

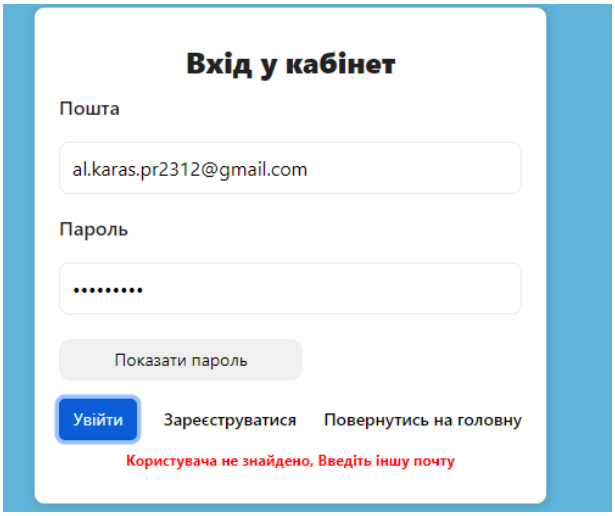

Рисунок 3.21 – Авторизація з неіснуючою поштою

В разі введення неправильного паролю користувач отримує повідомлення «Пароль не вірний». В разі введення невалідного паролю користувач повідомляється про це (рис. 3.22)

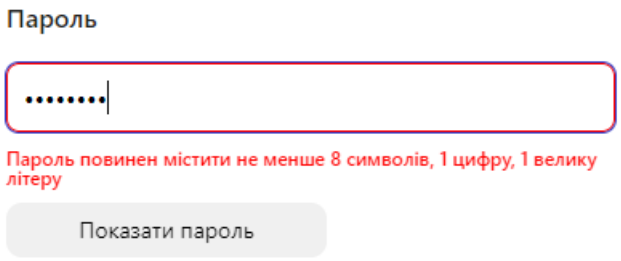

Рисунок 3.22 – Авторизація з невалідним паролем

Наступним компонентом реалізації є модуль передачі показників. За регулярним виразом користувач не може вводити значення, які менше або дорівнюють нулю (рис. 3.23).

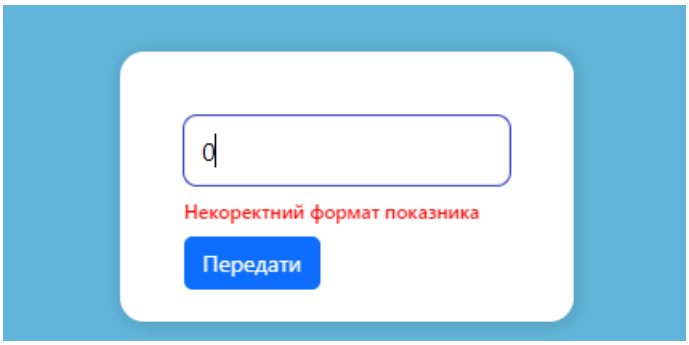

Рисунок 3.23 – Передача нульового значення

При розрахунку приблизної вартості перевіряються поля попереднього та поточного показника. Поля не можуть бути пустими та не можуть бути менше нуля. Також поточний показник не може бути менше ніж попередній, якщо поточний показник менше, то кнопка «розрахувати» стає недоступною (рис. 3.24). В разі правильно введених даних користувач отримує в полі приблизну вартість яку має сплатити (рис. 3.25).

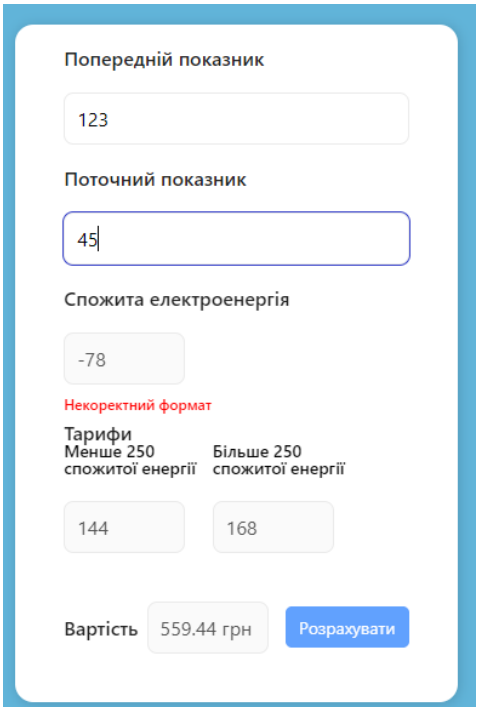

Рисунок 3.24 – Реакція на некоректне значення поточного показника

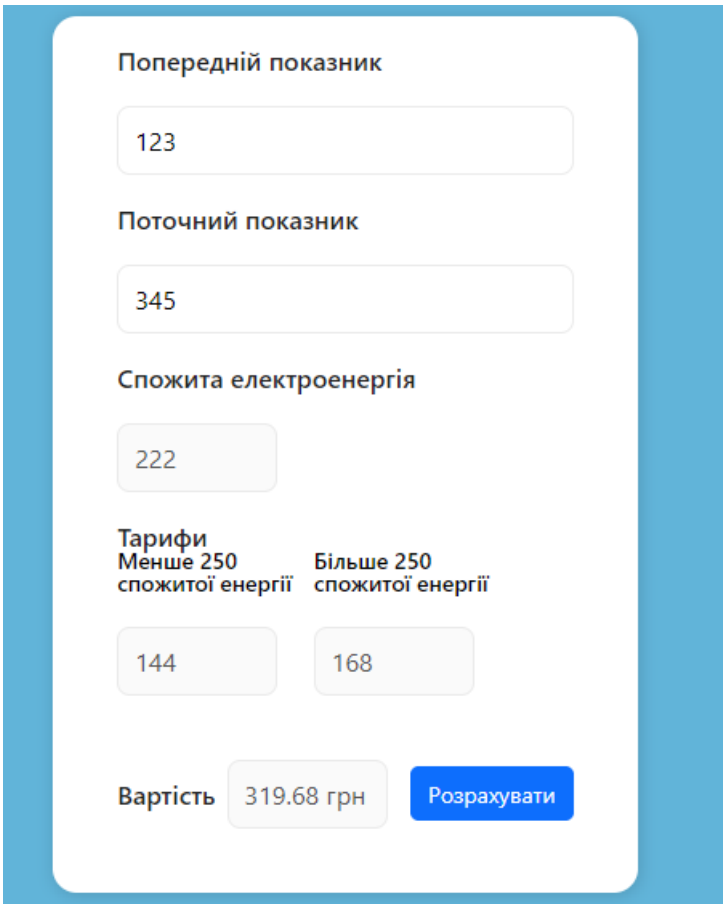

Рисунок 3.25 – Реакція на правильно введені дані

Наступним і останнім модулем для тестування є модуль подачі замовлення на послуги в кабінеті користувача. Поле пошта автоматично заповнюється значенням пошти користувача. Якщо користувач не вибирає одну з послуг з випадаючого списку, система попереджає, що операцію потрібно зробити обов'язково (рис. 3.26).

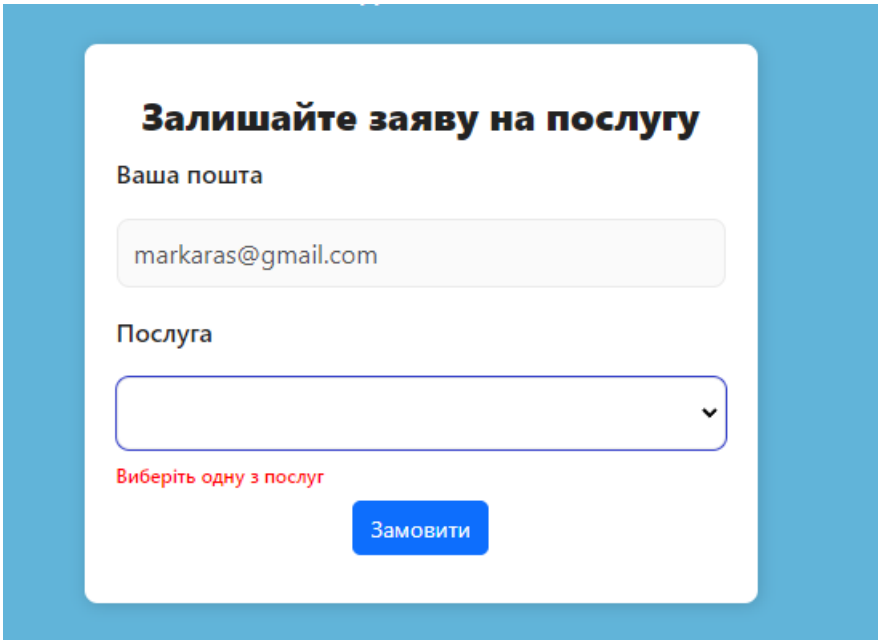

Рисунок 3.26 – Реакція на невибрану послугу користувачем

В разі, якщо користувач вибрав послугу і відправив форму, кнопка змінює текст на «Відправлено», що означає про успішне відправлення замовлення (рис. 3.27).

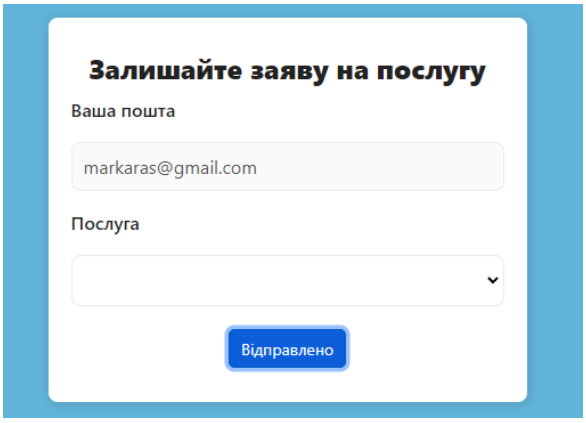

Рисунок 3.27 – Реакція на успішне відправлення форми

3.10 Алгоритм роботи системи

Гість

Клієнт, який ще не має облікового запису та хоче просто переглянути дані про компанію, має доступ до всього сайту, окрім кабінету та замовлення послуг. Він може переглядати головну сторінку, ознайомитись з послугами компанії та їх ціною, спілкуватися з інтерактивним чат-ботом. Для доступу до інших функцій та сторінок користувач має зареєструватися або увійти у свій обліковий запис.

Авторизований користувач

Користувач, який має обліковий запис може також переглядати усі сторінки, включаючи: головну, сторінку з послугами та підтримку. Для доступу до інших функцій та сторінок користувач має увійти у свій обліковий запис з валідними даними, якщо користувач введе неправильні данні система його попередить про це. Авторизований користувач може користуватися усіма сторінками особистого кабінету: передивлятися замовлені послуги, спожиту електроенергію, передавати показники, розраховувати приблизну вартість спожитої електроенергії, передивлятись та замовляти послуги.

Менеджер компанії

Менеджер як і всі користувачі має доступ до головних сторінок веб-системи. Для доступу до кабінету менеджера, користувач має увійти з своїми даними у систему. Як правило звичайний користувач не може зареєструватися як менеджер і мати доступ до всіх функцій, дані для входу до кабінету менеджеру знає лише власник компанії та менеджер.

Після успішного входу менеджер потрапляє до власного кабінету, де він може передивитися кількість актуальних користувачів та таблицю з користувачами. За потреби самого клієнта він може видалити сторінку користувача з системи, тоді користувач не зможе увійти повторно у свій акаунт. Також менеджер має доступ до сторінки замовлених послуг, де він може переглянути ціни на послуги та замовлені послуги користувачам. За потребою

користувача менеджер може видалити замовлення (наприклад, якщо це замовлення було випадкове). Перед підтвердженням статусу замовлення менеджер має уточнити у користувача чи погоджується він на дану послугу, шляхом написання користувачу на пошту вказану в замовленні. Якщо користувач підтверджує операцію – менеджер може змінити статус замовлення на «підтверджено». До того ж менеджер може перевіряти спожиту електроенергії усіх користувачів та за потреби видаляти її. Перед видаленням користувач має вказати адресу з якої було надіслані помилкові показники, після цього менеджер може видалити відповідний запис з показниками.

ВИСНОВКИ

В результаті виконання кваліфікаційної бакалаврської роботи розроблено інформаційну систему управління взаємодії з користувачами для підприємства електропостачання та виконано наступні завдання:

1) Проведено аналіз та літературний огляд сучасних джерел за тематикою розробки веб-додатків, з метою формування постановки задачі для подальшої реалізації;
2) Проведено моніторинг аналогів даної системи задля виокремлення необхідних факторів;
3) Розглянуто та обрано інструменти розробки задля досягнення поставленої мети;
4) Спроектовано дизайн сайту та створено карту системи;
5) Розроблено ефективну серверну сторону системи з урахуванням захисту даних;
6) Визначено бізнес логіку інформаційної системи та програмно налаштовано різні аспекти її;
7) Налаштований розподіл за ролями та доступ до різних функцій додатку;
8) Розроблено клієнт-орієнтований інтерфейс для зручного користування додатком;
9) Проведено аналіз та тестування різних компонентів додатку.

Розроблена інформаційна система не є ідеальною та має ряд подальших удосконалень. До таких відноситься підключення платіжного функціоналу за допомогою сторонніх сервісів для оплати рахунків, розташування системи на віддаленому сервері та хостингу для доступу в різних межах міста та країни, підключення графіків аналізу спожитої електроенергії та система сповіщень на пошту.

Детальну інформацію про програмну реалізацію можливо переглянути в моєму GitHub репозитарії (клієнтська та серверна частини).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Прикладні інформаційні системи | Факультет інформаційних технологій [Electronic resource]. URL: http://fit.univ.kiev.ua/archives/3181 (accessed: 19.05.2023).
2. What is Information System? Definition, Examples, & Facts | Emeritus [Electronic resource]. URL: https://emeritus.org/in/learn/information-system/ (accessed: 19.05.2023).
3. What is Customer Relationship Management (CRM): Definition, types, and examples of CRM companies | Snov.io [Electronic resource]. URL: https://snov.io/glossary/customer-relationship-management-crm/ (accessed: 08.04.2023).
4. Guide on How to Build Your Own CRM System for Your Business [Electronic resource]. URL: https://www.cleveroad.com/blog/how-to-build-your-own-custom-crm-system/ (accessed: 08.04.2023).
5. Карась О. І., Шовкопляс О. А. Використання CRM в інформаційних системах постачання послуг. Суми, 2023.
6. Що таке фреймворк (framework)? Пояснюємо простими словами [Electronic resource]. URL: https://highload.today/uk/frejmvorki-u-veb-rozrobtsi-shho-tse-yaki-isnuyut-i-dlya-chogo-potribni/ (accessed: 03.06.2023).
7. What is the MERN Stack? Introduction & Examples | Simplilearn [Electronic resource]. URL: https://www.simplilearn.com/tutorials/mongodb-tutorial/what-is-mern-stack-introduction-and-examples (accessed: 19.05.2023).
8. What is React.js? (Uses, Examples, & More) [Electronic resource]. URL: https://blog.hubspot.com/website/react-js (accessed: 20.05.2023).
9. What is CSS? - Learn web development | MDN [Electronic resource]. URL: https://developer.mozilla.org/en-US/docs/Learn/CSS/First_steps/What_is_CSS (accessed: 20.05.2023).

10. What is MongoDB? Features and how it works – TechTarget Definition [Electronic resource]. URL: https://www.techtarget.com/searchdatamanagement/definition/MongoDB (accessed: 20.05.2023).
11. What is an API? (Application Programming Interface) | MuleSoft [Electronic resource]. URL: https://www.mulesoft.com/resources/api/what-is-an-api (accessed: 20.05.2023).
12. What is an API? - Application Programming Interfaces Explained - AWS [Electronic resource]. URL: https://aws.amazon.com/what-is/api/?nc1=h_ls (accessed: 20.05.2023).
13. What is a REST API? | IBM [Electronic resource]. URL: https://www.ibm.com/topics/rest-apis (accessed: 20.05.2023).
14. What is Node.js: A Comprehensive Guide [Electronic resource]. URL: https://www.simplilearn.com/tutorials/nodejs-tutorial/what-is-nodejs (accessed: 20.05.2023).
15. Що таке Node JS простими словами - застосування Node JS у програмуванні | DAN-IT [Electronic resource]. URL: https://dan-it.com.ua/uk/blog/chto-jeto-takoe-node-js-prostymi-slovami/ (accessed: 20.05.2023).
16. Express JS Tutorial: What is Express in Node JS? [Electronic resource]. URL: https://www.simplilearn.com/tutorials/nodejs-tutorial/what-is-express-js (accessed: 20.05.2023).
17. Веб-дизайн. Інформаційна структура сайту [Electronic resource]. URL: https://webstudio2u.net/ua/design-web/443-information-structure.html (accessed: 20.05.2023).
18. Структура сайту: що це таке, види та приклади - як створити структуру для інтернет-магазину [Electronic resource]. URL:

https://elit-web.ua/ua/blog/kak-vyvesti-tekuschuju-strukturu-sajta (accessed: 20.05.2023).

19. How Important is Color in Website Design? | Studio 1 Design [Electronic resource]. URL: https://studio1design.com/how-important-is-color-in-website-design/ (accessed: 20.05.2023).
20. Advantages of using blue in your web design - Pumpkin Web Design Manchester [Electronic resource]. URL: https://www.pumpkinwebdesign.com/web-design-manchester/advantages-of-using-blue-in-your-web-design/ (accessed: 20.05.2023).
21. About npm | npm Docs [Electronic resource]. URL: https://docs.npmjs.com/about-npm (accessed: 20.05.2023).
22. What Is package.json? | heynode.com [Electronic resource]. URL: https://heynode.com/tutorial/what-packagejson/ (accessed: 20.05.2023).
23. JSON Web Tokens - jwt.io [Electronic resource]. URL: https://jwt.io/ (accessed: 02.06.2023).
24. Making HTTP requests with Axios | CircleCI [Electronic resource]. URL: https://circleci.com/blog/making-http-requests-with-axios/ (accessed: 03.06.2023).
25. UseSelector and UseDispatch: A Guide to React-Redux Hooks | Built In [Electronic resource]. URL: https://builtin.com/software-engineering-perspectives/useselector-usedispatch-react-redux (accessed: 03.06.2023).
26. A Simple Explanation of React.useEffect() [Electronic resource]. URL: https://dmitripavlutin.com/react-useeffect-explanation/ (accessed: 03.06.2023).
27. Hello from React-chatbot-kit | React-chatbot-kit [Electronic resource]. URL: https://fredrikoseberg.github.io/react-chatbot-kit-docs/ (accessed: 03.06.2023).

28. White/black/grey box-тестування - QALight [Electronic resource]. URL: https://qalight.ua/baza-znaniy/white-black-grey-box-testuvannya/ (accessed: 03.06.2023).
29. useForm | React Hook Form - Simple React forms validation [Electronic resource]. URL: https://www.react-hook-form.com/api/useform/ (accessed: 03.06.2023).
30. Regular Expression (Regex) Tutorial [Electronic resource]. URL: https://www3.ntu.edu.sg/home/ehchua/programming/howto/Regexe.html (accessed: 03.06.2023).

ДОДАТОК А
МОДЕЛІ БАЗИ ДАНИХ

A1 userModel.js

```
const mongoose = require('mongoose');
const UserModelSchema = new mongoose.Schema(
    {
        fname: {
            type: String,
            require: true,
        },
        lname: {
            type: String,
            require: true,
        },
        patronimic: {
            type: String,
            require: true,
        },
        address: {
            type: String,
            require: true,
        },
        email: {
            type: String,
            require: true,
            unique: true,
        },
        password: {
            type: String,
            require: true,
        },
        role: {
            type: String,
            require: true,
            default: "user",
        },
    },
    {
        collection: "UserInfo",
    }
);
const User = mongoose.model("UserInfo", UserModelSchema);
module.exports = User;
```

A2 unitModel.js

```
const mongoose = require('mongoose');
const unitModelSchema = new mongoose.Schema(
```

```
    {
        unitNo: {
            type: Number,
            require: true,
        },
        date: {
            type: Date,
            require: true,
            default: Date.now(),
        },
        owner: {
            type: mongoose.Types.ObjectId,
            require: true,
            ref: "UserInfo",
        },
        ownerAddress: {
            type: String,
            require: true,
            ref: "UserInfo",
        }

    },
    {
        collection: "UnitInfo",

    }
)
const Unit = mongoose.model("UnitInfo", unitModelSchema);
module.exports = Unit;
```

A3 userServiceModel.js

```
const mongoose = require('mongoose');
const userServiceSchema = new mongoose.Schema(
    {
        userid: {
            type: mongoose.Types.ObjectId,
            ref: "UserInfo",
        },
        type: {
            type: String,
            enum: ["sunpanels", "powercut", "powerconnect"],
            require: true,
        },
        useremail: {
            type: String,
            ref: "UserInfo",
        },
        status: {
            type: Boolean,
            default: false,
        },
```

```
        sendDate: {
            type: Date,
            default: Date.now(),
        },
    }
);
const UserService = mongoose.model("userService",
userServiceSchema);
module.exports = UserService;
```

A4 pricesModel.js

```
const mongoose = require('mongoose');
const pricesModelSchema = new mongoose.Schema(
    {
        sunpanels: {
            type: String,
            default: "Встановлення сонячних панелей"
        },
        suncost: {
            type: Number,
            require: true,
            default: 2000,
        },
        powerconnect: {
            type: String,
            default: "Підключення та налаштування лічильника",
        },
        powerconcost: {
            type: Number,
            require: true,
            default: 500,
        },
        powercut: {
            type: String,
            default: "Відключення лічнильника",
        },
        powercutcost: {
            type: Number,
            require: true,
            default: 500,
        }
    },
    {
        collection: "PriceList",

    }
)
const PriceList = mongoose.model("PriceList", pricesModelSchema);
module.exports = PriceList;
```

ДОДАТОК Б
КОНТРОЛЛЕРИ ТА НАПРЯМКИ

Б1 userController.js

```
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');
const config = require('../cfg');
const User = require('../models/userModel');
const register = async (req, res) => {
    const { fname, lname, patronimic, address, email, password } =
req.body;
    const encryptedPassword = await bcrypt.hash(password, 10);
    try {
        const oldUser = await User.findOne({ email });
        if (oldUser) {
            return res.status(401).json({ message: "Користувач уже
існує, введіть інші дані" });
        } else {
            const newUser = await User.create({
                fname,
                lname,
                patronimic,
                address,
                email,
                password: encryptedPassword
            });
            const user = await newUser.save();
            return res.status(200).json({
                user: user,
            })
        }
    } catch (error) {
        return res.status(401).json({ message: "Не вдалось
створити акаунт" })
    }
}
const login = async (req, res) => {
    const { email, password } = req.body;
    const user = await User.findOne({ email });
    if (!user) {
        return res.status(401).json({ message: "Користувача не
знайдено, Введіть іншу почту" });
    }
    if (await bcrypt.compare(password, user.password)) {
        const token = jwt.sign({ usid: user._id, email:
user.email, role: user.role }, config.jwt.TOKEN, {
            expiresIn: config.jwt.EXPIRESIN,
        })
        if (res.status(200)) {
            return res.json({
```

```
                user: user,
                userRole: user.role,
                token: `Bearer ${token}`
            });
        }
    } else {
        return res.status(401).json({ message: "Пароль не вірний"
})
    }
}
const getUsers = async (req, res) => {
    const users = await User.find({});
    return res.status(200).json({ users: users });
}
const getAmountUsers = async (req, res) => {
    const amountUsers = await User.countDocuments({});
    return res.status(200).json({ amount: amountUsers })
}
const getUser = async (req, res) => {
    try {
        const user = await User.findById(req.user._id);
        if (!user) {
            res.status(404).json({
                message: "Користувача не знайдено"
            })
        }
        const token = jwt.sign({ usid: user._id, email:
user.email, role: user.role }, config.jwt.TOKEN, {
            expiresIn: config.jwt.EXPIRESIN,
        })
        return res.status(200).json({
            user: user,
            token: `Bearer ${token}`
        })
    } catch (error) {
        res.status(404).json({ message: "Не вдалось знайти
користувача" })
    }
}
const deleteUser = async (req, res) => {
    try {
        const userId = req.params.id.toString();
        console.log(userId);
        const user = await User.findByIdAndDelete(userId);
        if (!user) {
            return res.status(404).json({ message: `Не знайдено
користувача за ID: ${userId} ` });
        }
        return res.status(200).json({ deleteduser: user })
    } catch (error) {
        console.log(error);
```

```
        return res.status(500).json({ message: "Користувача не
видалено" });

    }
}
module.exports = {
    register,
    login,
    getUsers,
    getUser,
    deleteUser,
    getAmountUsers,
}
```

Б2 userRouter.js

```
const express = require('express');
const router = express.Router();
const { UserController } = require('../controllers');
const auth = require('../middleware/auth')
router.post('/login', UserController.login);
router.post('/register', UserController.register);
router.get('/getUsers', UserController.getUsers);
router.get('/getUser', auth, UserController.getUser);
router.delete('/deleteuser/:id', UserController.deleteUser);
router.get('/getamount', UserController.getAmountUsers);
module.exports = router;
```

Б3 unitController.js

```
const Unit = require('../models/unitModel')

const createUnit = async (req, res) => {
    try {
        const { unitNo } = req.body;
        const userId = req.user._id;
        const userAddress = req.user.address;
        const unit = await Unit.create({
            unitNo,
            owner: userId,
            ownerAddress: userAddress,
        })
        if (res.status(200)) {
            return res.json({ unit: unit });
        }
    } catch (error) {
        return res.send({ message: "Не вдалось відправити
показники" });
    }
}
const getUnits = async (req, res) => {
    try {
```

```
        const userId = req.user._id;
        const unitsById = await Unit.find({ owner: userId });
        return res.status(200).json({ units: unitsById });

    } catch (error) {
        return res.status(500).json({ message: "Щось пішло не так"
});
    }
}
const getAllUnits = async (req, res) => {
    const unitsAll = await Unit.find({});
    return res.status(200).json({ units: unitsAll });
}
const deleteUnitById = async (req, res) => {
    try {
        const unitID = req.params.unitid.toString();
        const delUnitbyID = await Unit.findByIdAndDelete(unitID);
        return res.status(200).json({ deletedunit: delUnitbyID });
    } catch (error) {
        return res.status(500).json({ message: "Не вдалось
видалити запис" });
    }
}
const getSumofUnitsById = async (req, res) => {
    try {
        const userId = req.user._id;
        const sum = await Unit.aggregate([
            { $match: { owner: userId } },
            { $group: { _id: null, total: { $sum: '$unitNo' } } },
        ]);
        res.status(200).json({ sumunits: sum[0].total });
    } catch (error) {
        return res.status(500).json({ message: "Не вдалося зробити
операцію" })
    }
}
const getSumofAllUnits = async (req, res) => {
    try {
        const sumall = await Unit.aggregate([
            { $group: { _id: null, total: { $sum: '$unitNo' } } },
        ]);
        res.status(200).json({ sumalunits: sumall[0].total });
    } catch (error) {
        return res.status(500).json({ message: "Не вдалось зробити
операцію" });
    }
}
const getAmountUnits = async(req, res) => {
    const amountUnits = await Unit.countDocuments({});
    return res.status(200).json({ amountunits: amountUnits })
}
```

```
module.exports = {
    createUnit,
    getUnits,
    getAllUnits,
    deleteUnitById,
    getSumofUnitsById,
    getSumofAllUnits,
    getAmountUnits,
}
```

Б4 unitRouter.js

```
const express = require('express');
const router = express.Router();
const { UnitController } = require('../controllers');
const auth = require('../middleware/auth');

router.post('/createUnit', auth, UnitController.createUnit);
router.get('/getUnits', auth, UnitController.getUnits);
router.get('/getAllUnits',  UnitController.getAllUnits);
router.delete('/deleteunit/:unitid',
UnitController.deleteUnitById);
router.get('/getSumUserUnits', auth,
UnitController.getSumofUnitsById);
router.get('/getsumallunits', UnitController.getSumofAllUnits);
router.get('/getamount', UnitController.getAmountUnits);
module.exports = router;
```

Б5 ServiceController.js

```
const Service = require('../models/userServiceModel');
const createService = async (req, res) => {
    try {
        const uid = req.user._id;
        const uemail = req.user.email;
        const { type } = req.body;
        const newService = await Service.create({
            type,
            userid: uid,
            useremail: uemail,
        });

        const service = await newService.save();

        return res.status(200).json({
            service: service,
            message: "Внесено",
        })
    } catch (error) {
        res.status(505).json({ message: "Щось пішло не так" });
    }
}
```

```
const getServices = async (req, res) => {
    const services = await Service.find({});
    return res.status(200).json({ servs: services });
}
const updateStatusServ = async (req, res) => {
    try {
        const servID = req.params.servid.toString();
        const updatedService = await
Service.findByIdAndUpdate(servID, { status: true });
        return res.status(200).json({ updated: updatedService });
    } catch (error) {
        return res.status(500).json({ message: "Не вдалось
виконати редагування" })
    }
}
const deleteService = async (req, res) => {
    try {
        const servID = req.params.servid.toString();
        const deletedService = await
Service.findByIdAndDelete(servID);
        return res.status(200).json({ deleted: deletedService });
    } catch (error) {
        return res.status(500).json({ message: "Не вдалось
виконати операцію" })
    }
}
const getServById = async (req, res) => {
    try {
        const uid = req.user._id;
        const servById = await Service.find({ userid: uid });

        return res.status(200).json({ services: servById });
    } catch (error) {
        return res.status(500).json({ message: "Щось пішло не так"
});
    }
}
module.exports = {
    createService,
    getServById,
    getServices,
    updateStatusServ,
    deleteService,
}
```

Б6 ServiceRouter.js

```
const express = require('express');
const router = express.Router();
const { ServiceController } = require('../controllers');
const auth = require('../middleware/auth');
```

```
router.post('/createService', auth,
ServiceController.createService);
router.get('/getUserService', auth,
ServiceController.getServById);
router.get('/getServices', ServiceController.getServices);
router.put('/updateStatus/:servid',
ServiceController.updateStatusServ);
router.delete('/deleteService/:servid',
ServiceController.deleteService);
module.exports = router;
```

Б7 PricesController.js

```
const PriceList = require('../models/pricesModel');
const createPriceList = async (req, res) => {
    try {
        const { suncost, powerconcost, powercutcost } = req.body;
        const priceList = await PriceList.create({
            suncost: suncost,
            powerconcost: powerconcost,
            powercutcost: powercutcost,
        });
        return res.status(200).json({ prices: priceList });
    } catch (error) {
        return res.status(500).json({ message: "Не вдалось
створити цінники" });
    }
}
const getPriceList = async (req, res) => {
    const prices = await PriceList.find({});
    return res.status(200).json({ dataprices: prices });
}
module.exports = {
    createPriceList,
    getPriceList,
}
```

Б8 priceslistRouter.js

```
const express = require('express');
const router = express.Router();
const { PricesController } = require('../controllers');
router.post('/createlist', PricesController.createPriceList);
router.get('/getpricelist', PricesController.getPriceList);
module.exports = router;
```

ДОДАТОК В
REDUX ФРАГМЕНТИ

В1 loginSlice.js

```
import { createSlice, createAsyncThunk } from "@reduxjs/toolkit";
import axios from '../../utils/axios.js';
const initialState = {
    user: null,
    deleteduser: null,
    datausers: [],
    token: null,
    adminrole: false,
    isLoading: 'loading',
    error: null,
    isRegistered: false,
    amountusers: null,
}
export const loginUser = createAsyncThunk('login/loginUser', async
(params, { _, rejectWithValue }) => {
    try {
        const { data } = await axios.post('/user/login', params);
        if (data.token) {
            window.localStorage.setItem('token', data.token);
        }
        return data;
    } catch (error) {
        return rejectWithValue(error.response.data)
    }
})
export const registerUser = createAsyncThunk('login/registerUser',
async (params, { rejectWithValue }) => {
    try {
        const { data } = await axios.post('/user/register',
params);
        return data;
    } catch (error) {
        return rejectWithValue(error.response.data);
    }
})
export const getUser = createAsyncThunk('login/getUser', async (_,
{ rejectWithValue }) => {
    try {
        const { data } = await axios.get('/user/getUser');
        return data;
    } catch (error) {
        return rejectWithValue(error.response.data)
    }
})
export const getUsers = createAsyncThunk('login/getUsers', async
() => {
```

```
    const { data } = await axios.get('/user/getUsers');
    return data;
})
export const deleteUser = createAsyncThunk('login/deleteUser',
async (param, { rejectWithValue }) => {
    try {
        const { data } = await
axios.delete(`/user/deleteuser/${param}`);
        return data;
    } catch (error) {
        return rejectWithValue(error.response.data);
    }
})
export const getAmountUsers = createAsyncThunk('login/getAmount',
async () => {
    const { data } = await axios.get('/user/getamount');
    return data;
})
const loginSlice = createSlice({
    name: 'logreg',
    initialState,
    reducers: {
        logout: (state) => {
            state.user = null
            state.adminrole = null
            state.error = null
            state.token = null
        }
    },
    extraReducers: (builder) => {
        builder
            .addCase(registerUser.pending, (state) => {
                state.isLoading = 'loading'
                state.error = null
            })
            .addCase(registerUser.fulfilled, (state, action) => {
                state.isLoading = 'loaded'
                state.user = action.payload.user
                state.error = null
                state.isRegistered = true
            })
            .addCase(registerUser.rejected, (state, action) => {
                state.isLoading = 'error'
                state.error = action.payload.message
            })
            .addCase(loginUser.pending, (state) => {
                state.isLoading = 'loading'
                state.error = null
            })

            .addCase(loginUser.fulfilled, (state, action) => {
```

```
                state.isLoading = 'loaded'
                state.user = action.payload.user
                state.adminrole = Boolean(action.payload.userRole
=== "admin")
                state.token = action.payload.token
                state.error = null
            })
            .addCase(loginUser.rejected, (state, action) => {
                state.isLoading = 'error'
                state.error = action.payload.message
            })
            .addCase(getUser.pending, (state) => {
                state.isLoading = 'loading'
                state.error = null
            })
            .addCase(getUser.fulfilled, (state, action) => {
                state.isLoading = 'loaded'
                state.adminrole = action.payload?.role
                state.user = action.payload?.user
                state.token = action.payload?.token
                state.error = null
            })
            .addCase(getUser.rejected, (state, action) => {
                state.isLoading = 'error'
                state.error = action.error.message
            })
            .addCase(getUsers.pending, (state) => {
                state.isLoading = 'loading'
            })
            .addCase(getUsers.fulfilled, (state, action) => {
                state.isLoading = "loaded"
                state.datausers = action.payload.users
            })
            .addCase(getUsers.rejected, (state) => {
                state.isLoading = 'error'
            })
            .addCase(deleteUser.pending, (state) => {
                state.isLoading = 'loading'
                state.error = null
            })
            .addCase(deleteUser.fulfilled, (state, action) => {
                state.isLoading = 'loaded'
                state.deleteduser = action.payload.deleteduser
                state.error = null
            })
            .addCase(deleteUser.rejected, (state, action) => {
                state.isLoading = 'error'
                state.error = action.payload.message
            })
            .addCase(getAmountUsers.pending, (state) =>{
                state.isLoading = 'loading'
```

```
                state.error = null
            })
            .addCase(getAmountUsers.fulfilled, (state, action) =>{
                state.isLoading = 'loaded'
                state.amountusers = action.payload.amount
                state.error = null
            })
            .addCase(getAmountUsers.rejected, (state, action) =>{
                state.isLoading = 'error'
                state.error = action.payload.message
            })
    }
})
export const selectIsRegged = (state) =>
(state.logreg.isRegistered);
export const selectIsLogged = (state) =>
Boolean(state.logreg.token)
export const selectIsAdmin = (state) =>
Boolean(state.logreg.adminrole)
export const selectDeletedInfo = (state) =>
(state.logreg.deleteduser);
export const infoAboutUser = (state) => (state.logreg.user);
export const fetchUsers = (state) => (state.logreg.datausers);
export const selectAmount = (state) => (state.logreg.amountusers);
export const { logout } = loginSlice.actions;
export const loginReducer = loginSlice.reducer;
```

B2 unitSlice.js

```
import { createAsyncThunk, createSlice } from "@reduxjs/toolkit";
import axios from '../../utils/axios.js';
const initialState = {

    items: [],
    deletedunit: null,
    itemsall: [],
    isLoading: "loading",
    error: null,
    isSent: false,
    sumUnits: null,
    sumallun: null,
    statusloadsumall: false,

};
export const createUnit = createAsyncThunk('units/create', async
(params, { rejectWithValues }) => {
    try {
        const { data } = await axios.post('/unit/createUnit',
params);

        return data;
```

```
    } catch (error) {
        return rejectWithValues(error.response.data)
    }
})
export const getUnitById = createAsyncThunk('units/getById', async
(_, { rejectWithValue }) => {
    try {
        const { data } = await axios.get('/unit/getUnits');
        return data;
    } catch (error) {
        rejectWithValue(error.response.data);
    }
})
export const getUnits = createAsyncThunk('units/getUnits', async
() => {
    const { data } = await axios.get('/unit/getAllUnits');
    return data;
})
export const deleteUnitById = createAsyncThunk('units/deleteUnit',
async (param, { rejectWithValue }) => {
    try {
        const { data } = await
axios.delete(`/unit/deleteunit/${param}`);
        return data;
    } catch (error) {
        return rejectWithValue(error.response.data);
    }
})
export const getSumUserUnits =
createAsyncThunk('units/getsumusunits', async () => {
    const { data } = await axios.get('unit/getSumUserUnits');
    return data;
})
export const getSumAllUnits =
createAsyncThunk('units/getsumallunits', async () => {
    const { data } = await axios.get('unit/getsumallunits');
    return data;
})
const unitsSlice = createSlice({
    name: 'units',
    initialState,
    reducers: {
    },
    extraReducers: (builder) => {
        builder
            .addCase(createUnit.pending, (state) => {
                state.isLoading = "loading"
                state.error = null
            })
            .addCase(createUnit.fulfilled, (state, action) => {
                state.isLoading = "loaded"
```

```
            state.isSent = true
        })
        .addCase(createUnit.rejected, (state, action) => {
            state.isLoading = "error"
            state.error = action.error.message
        })
        .addCase(getUnitById.pending, (state) => {
            state.isLoading = "loading"
            state.items = []
            state.error = null
        })
        .addCase(getUnitById.fulfilled, (state, action) => {
            state.isLoading = "loaded"
            state.items = action.payload.units
            state.isSent = false
        })
        .addCase(getUnitById.rejected, (state, action) => {
            state.isLoading = "error"
            state.error = action.error.message
        })
        .addCase(getUnits.pending, (state) => {
            state.isLoading = "loading"
            state.error = null
        })
        .addCase(getUnits.fulfilled, (state, action) => {
            state.isLoading = "loaded"
            state.itemsall = action.payload.units
        })
        .addCase(getUnits.rejected, (state, action) => {
            state.isLoading = "error"
            state.error = action.error.message
        })
        .addCase(deleteUnitById.pending, (state) => {
            state.isLoading = "loading"
            state.error = null
        })
        .addCase(deleteUnitById.fulfilled, (state, action) =>
{
            state.isLoading = "loaded"
            state.deletedunit = action.payload.deletedunit
            state.error = null
        })
        .addCase(deleteUnitById.rejected, (state, action) => {
            state.isLoading = "error"
            state.error = action.payload.message
        })
        .addCase(getSumUserUnits.pending, (state) => {
            state.isLoading = "loading"
            state.error = null
        })
        .addCase(getSumUserUnits.fulfilled, (state, action) =>
```

```
{
                state.isLoading = "loaded"
                state.sumUnits = action.payload.sumunits
                state.error = null
            })
            .addCase(getSumUserUnits.rejected, (state, action) =>
{
                state.isLoading = "error"
                // state.error = action.payload.message
            })
            .addCase(getSumAllUnits.pending, (state) => {
                state.isLoading = "loading"
                state.error = null
            })
            .addCase(getSumAllUnits.fulfilled, (state, action) =>
{
                state.isLoading = "loaded"
                state.sumallun = action.payload.sumalunits
                state.error = null
            })
            .addCase(getSumAllUnits.rejected, (state, action) => {
                state.isLoading = "error"
                state.error = action.payload.message
            })
    }
})
export const unitsReducer = unitsSlice.reducer;
export const infoAboutUnits = (state) => (state.units.items);
export const infoAboutAllUnits = (state) =>
(state.units.itemsall);
export const selectDeletedUnit = (state) =>
(state.units.deletedunit);
export const isUnitSent = (state) => (state.units.isSent);
export const selectSumOfUsUnits = (state) =>
(state.units.sumUnits);
export const selectSumOfAllUnits = (state) =>
(state.units.sumallun);
```

B3 serviceSlice.js

```
import { createAsyncThunk, createSlice } from "@reduxjs/toolkit";
import axios from '../../utils/axios.js';
const initialState = {

    servid: [],
    servall: [],
    isLoading: "loading",
    error: null,
    statussend: false,
    loadsumserv: false,
    updated: false,
```

```
    deletedserv: null,
};
export const createReqService =
createAsyncThunk('service/createReqServ', async (params, {
rejectWithValue }) => {
    try {
        const { data } = await
axios.post('/service/createService', params);
        return data;
    } catch (error) {
        return rejectWithValue(error.response.data);
    }
})
export const getAllServices =
createAsyncThunk('service/getallServices', async () => {
    const { data } = await axios.get('/service/getServices');
    return data;
})
export const getUserServices =
createAsyncThunk('service/getUserServices', async (_, {
rejectWithValue }) => {
    try {
        const { data } = await
axios.get('/service/getUserService');
        return data;
    } catch (error) {
        return rejectWithValue(error.response.data);
    }
})
export const updateStatusService =
createAsyncThunk('service/updateStatus', async (param, {
rejectWithValue }) => {
    try {
        const { data } = await
axios.put(`/service/updateStatus/${param}`);
        return data;
    } catch (error) {
        return rejectWithValue(error.response.data);
    }
})
export const deleteService =
createAsyncThunk('service/deleteService', async (param, {
rejectWithValue }) => {
    try {
        const { data } = await
axios.delete(`/service/deleteService/${param}`);
        return data;
    } catch (error) {
        return rejectWithValue(error.response.data);
    }
})
```

```
const serviceSlice = createSlice({
    name: "services",
    initialState,
    reducers: {

    },
    extraReducers: (builder) => {
        builder
            .addCase(createReqService.pending, (state) => {
                state.error = null
                state.isLoading = 'loading'
            })
            .addCase(createReqService.fulfilled, (state) => {
                state.error = null
                state.statussend = true
                state.isLoading = 'loaded'
            })
            .addCase(createReqService.rejected, (state, action) =>
{
                state.error = action.payload.message
                state.isLoading = 'error'
            })
            .addCase(getAllServices.pending, (state) => {
                state.error = null
                state.isLoading = 'loading'
            })
            .addCase(getAllServices.fulfilled, (state, action) =>
{
                state.servall = action.payload.servs
                state.isLoading = 'loaded'
                state.error = null
            })
            .addCase(getAllServices.rejected, (state, action) => {
                state.error = action.payload.message
                state.isLoading = 'error'
            })
            .addCase(getUserServices.pending, (state) => {
                state.error = null
                state.isLoading = 'loading'
            })
            .addCase(getUserServices.fulfilled, (state, action) =>
{
                state.error = null
                state.servid = action.payload.services
                state.isLoading = 'loaded'
            })
            .addCase(getUserServices.rejected, (state, action) =>
{
                state.error = action.payload.message
                state.isLoading = 'error'
            })
```

```
            .addCase(updateStatusService.pending, (state) => {
                state.error = null
                state.isLoading = 'loading'
            })
            .addCase(updateStatusService.fulfilled, (state,
action) => {
                state.error = null
                state.updated = true
            })
            .addCase(updateStatusService.rejected, (state, action)
=> {
                state.error = action.payload.message
                state.isLoading = 'error'
            })
            .addCase(deleteService.pending, (state) => {
                state.error = null
                state.isLoading = 'loading'
            })
            .addCase(deleteService.fulfilled, (state, action) => {
                state.error = null
                state.deletedserv = action.payload.deleted
            })
            .addCase(deleteService.rejected, (state, action) => {
                state.error = action.payload.message
                state.isLoading = 'error'
            })
    }
})

export const UserServicesInfo = (state) =>
(state.services.servid);
export const ServicesInfo = (state) => (state.services.servall);
export const isUpdated = (state) => (state.services.updated);
export const isDeleted = (state) => (state.services.deletedserv);
export const ServiceReducer = serviceSlice.reducer;
```

B4 priceSlice.js

```
import { createSlice, createAsyncThunk } from "@reduxjs/toolkit";
import axios from '../../utils/axios.js';
const initialState = {
    pricelist: [],
    isLoading: 'loading',
    error: null,
}
export const fetchPrices = createAsyncThunk('prices/fetchPrices',
async () => {
    const { data } = await axios.get('/prices/getpricelist');
    return data;
})
const PriceSlice = createSlice({
```

```
    name: "prices",
    initialState,
    reducers: {
    },
    extraReducers: (builder) => {
        builder
        .addCase(fetchPrices.pending, (state) => {
            state.error = null
            state.isLoading = 'loading'
        })
        .addCase(fetchPrices.fulfilled, (state, action) => {
            state.error = null
            state.isLoading = 'loaded'
            state.pricelist = action.payload.dataprices
        })
        .addCase(fetchPrices.rejected, (state, action) => {
            state.isLoading = 'error'
            state.error = action.payload.message
        })
    }
})
export const PriceReducer = PriceSlice.reducer;
export const PricesData = (state) => (state.prices.pricelist);
```

ДОДАТОК Г
ЛІСТИНГ ЧАТ-БОТУ

Г1 config.js

```
import { createChatBotMessage } from "react-chatbot-kit";
import Overview from "./widgets/Overview";
const name = "SumEnergofaqBot"
const config = {
    lang: "no",
    botName: name,
    initialMessages: [
        createChatBotMessage(`Вітаю у чат боті ${name}. Бот може
відповісти на ваші запитання `),
        createChatBotMessage(`Тут ви можете ознайомитись з
відповідями, клікнувши на відповідний пункт`, {
            delay: 400,
            widget: "overview"
        }),

    ],
    state: {},
    widgets: [
        {
            widgetName: "overview",
            widgetFunc: (props) => <Overview {...props} />,
            mapStateToProps: ["message"]
        }
    ]
}
export default config;
```

Г2 ActionProvider.js

```
class ActionProvider {
    constructor(createChatBotMessage, setStateFunc,
createClientMessage) {
        this.createChatBotMessage = createChatBotMessage;
        this.setState = setStateFunc;
        this.createClientMessage = createClientMessage;
    }
    handleOptions = (options) => {
        const message = this.createChatBotMessage(
            "Як я можу вам допомогти. Нижче наведено пункти в яких
можливо знайти відповіді на ваші запитання",
            {
                widget: "overview",
                loading: true,
                terminateLoading: true,
                ...options
```

```
            }
        );

        this.addMessageToState(message);
    };

    handleUnits = () => {
        const message = this.createChatBotMessage(
            "Щоб передати показники вам потрібно \n 1.
Зареєструватися в системі/Увійти у кабінет\n 2. Перейти на
сторінку 'Передати показники'\n 3. У формі передати показники"
        );
        this.addMessageToState(message);
    }

    handleServices = () => {
        const message = this.createChatBotMessage(
            "Наша компанія надає наступні послуги: \n
1.Встановлення та налаштування сонячних панелей \n 2. Підключення
та налаштування лічильників \n 3. Відключення лічильника"
        );
        this.addMessageToState(message);
    }

    handleCabinet = () => {
        const message = this.createChatBotMessage(
            "Кабінет - це місце, де ви можете передавати та
відслідковувати свої показники, замовляти послуги та переглядати
замовлені послуги, розраховувати очікувану вартість за спожиту
електроенергію"
        );
        this.addMessageToState(message);
    }
    addMessageToState = (message) => {
        this.setState((state) => ({
            ...state,
            messages: [...state.messages, message]
        }));
    };

}

export default ActionProvider;
```

Г3 MessageParser.js

```
class MessageParser {
    constructor(actionProvider, state) {
        this.actionProvider = actionProvider;
        this.state = state;
    }
```

```
    parse(message) {
        message = message.toLowerCase();
        if (
            message.includes("допомогти") ||
            message.includes("питання")
        ) {
            return this.actionProvider.handleOptions();
        }
        if (
            message.includes("показник") ||
            message.includes("показники")
        ) {
            return this.actionProvider.handleUnits();
        }
        if (
            message.includes("послуги")
        ) {
            return this.actionProvider.handleServices();
        }
        if (message.includes("кабінет")){
            return this.actionProvider.handleCabinet();
        }
        return this.actionProvider.handleOptions();
    }
}

export default MessageParser;
```

Г4 Options.jsx

```
import React from "react"
import style from './Chatbot.module.scss';
const Options = (props) => {
    return (
        <div className={`${style.options}`}>
            <h1 className={`${style['options-
header']}`}>{props.title}</h1>
            <div className={`${style['options-container']}`}>
                {props.options.map((option) => {
                    return (
                        <div
                            className={`${style['option-item']}`}
                            onClick={option.handler}
                            key={option.id}
                        >
                            {option.name}
                        </div>
                    );
                })}
            </div>
        </div>
```

```
    )
}
export default Options;
```

Г5 Overview.jsx

```
import Options from './Options'

export default function GeneralOptions(props) {
    const options = [
        {
            name: "Як передати показники",
            handler: props.actionProvider.handleUnits,
            id: 1,
        },
        {
            name: "Які послуги надає компанія",
            handler: props.actionProvider.handleServices,
            id: 2,
        },
        {
            name: "Що таке кабінет",
            handler: props.actionProvider.handleCabinet,
            id: 3,
        }
    ];
    return (<Options options={options} title="Варіанти" {...props}
/>)
}
```