МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра інформаційних технологій

«До захисту допущено»
В.о. завідувача кафедри
_____Світлана ВАЩЕНКО
____ _____ 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття освітнього ступеня бакалавр

зі спеціальності 122 «Комп'ютерні науки»,

освітньо-професійної програми «Інформаційні технології проектування»
на тему: «Вебдодаток підтримки діяльності сервісного центру комп'ютерної та мобільної техніки»

Здобувача групи ІТ-02 Бордюгов Олексій Миколайович
(шифр групи) (прізвище, ім'я, по батькові)

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

_____ Олексій Бордюгов
(підпис) (Ім'я та ПРІЗВИЩЕ здобувача)

Керівник доц., к.т.н., доц. Світлана ВАЩЕНКО _____
(посада, науковий ступінь, вчене звання, Ім'я та ПРІЗВИЩЕ) (підпис)

Суми – 2024

Сумський державний університет
Факультете електроніки та інформаційних технологій
Кафедра інформаційних технологій
Спеціальність 122 «Комп'ютерні науки»
Освітньо-професійна програма «Інформаційні технології проектування»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри ІТ
_____ Світлана ВАЩЕНКО
«_____» _____ 2024 р.

ЗАВДАННЯ

на кваліфікаційну роботу бакалавра студентові

Бордюгов Олексій Миколайович

1 Тема кваліфікаційної роботи «Вебдодаток підтримки діяльності сервісного центру комп'ютерної та мобільної техніки»
затверджена наказом по університету від «07» травня 2024р. № 0482-VI

2 Термін здачі студентом кваліфікаційної роботи «26» травня 2024 р.

3 Вхідні дані до кваліфікаційної роботи дані про каталог послуг, дані про запроси користувачів, дані про запчастини

4 Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) аналіз предметної області, огляд останніх досліджень і публікацій, аналіз програмних продуктів-аналогів, мета та задачі дослідження, моделювання, функціональне моделювання в IDEF0, моделювання даних, практична реалізація, програмна реалізація, використання розробки продукту.

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових слайдів презентації) тема, актуальність, постановка задачі, огляд програмних продуктів-аналогів, таблиця порівняння аналогів, функціональні вимоги до вебдодатку, інструменти реалізації, структурно-функціональне моделювання, моделювання бази даних, моделювання варіантів використання, практична реалізація, демонстрація роботи, висновок

6 Консультанти випускної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7.Дата видачі завдання 17.10.2023

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Дослідження предметної області	17.10.23-13.11.23	
2	Вибір технологій для створення вебдодатку	14.11.23-25.12.23	
3	Розробка дизайну вебдодатку	26.12.23-14.02.24	
4	Розробка функціональності вебдодатку	15.02.24-08.05.24	
5	Наповнення контентом вебдодатку	09.05.24-05.06.24	
6	Тестування та підготовка до релізу	06.06.24-08.07.24	
7	Підготовка письмового звіту	09.07.24-09.07.24	
8	Реліз вебдодатку	10.07.24-10.07.24	

Студент _____ Олексій БОРДЮГОВ

Керівник роботи _____ к.т.н., доц. Світлана ВАЩЕНКО

АНОТАЦІЯ

Тема кваліфікаційної роботи магістра «Вебдодаток підтримки діяльності сервісного центру комп’ютерної та мобільної техніки».

Пояснювальна записка складається зі вступу, 3 розділів, висновків, списку використаних джерел із 17 найменувань, додатків. Загальний обсяг роботи − 73 сторінок, у тому числі 47 сторінок основного тексту, 2 сторінки списку икористаних джерел, 24 сторінок додатків.

Актуальність роботи полягає в розробці вебдодатку, який оптимізує та майже автоматизує роботу сервісного центру, що дає можливість клієнту та майстру не задумуватись про інше окрім, як ремонт техніки.

Мета роботи: створення вебдодатку на основі технології React, що автоматизує роботу сервісного центру.

В дипломній роботі показано процес розробки вебдодатку для сервісного центру. Проведено наліз та порівняння програмних додатків-аналогів, які допоможуть в створенні ідеї. Виконано планування розробки додатку, аналіз і моделювання роботи майбутнього, на основі чого створюється додаток. За допомогою обраних технологій реалізовано базу даних та виконано програмну реалізацію. Проведено тестування роботи додатку.

Ключові слова: вебдодаток, JavaScript, React, HTML, CSS, PosgreSQL, Node.JS.

ЗМІСТ

ВСТУП

У сучасному інформаційному суспільстві розвиток технологій та поширення комп'ютерної та мобільної техніки суттєво змінили підходи до обслуговування користувачів. Зростаюча кількість технічних пристроїв вимагає надійної та ефективної системи підтримки для забезпечення їх безперебійної роботи. У цьому контексті виникає актуальна проблема оптимізації роботи сервісних центрів комп'ютерної та мобільної техніки для забезпечення якісного обслуговування та задоволення потреб клієнтів.

Об'єктом дослідження є діяльність сервісних центрів, спрямована на обслуговування комп'ютерної та мобільної техніки.

Предметом дослідження є програмні засоби підтримки діяльності сервісного центру комп'ютерної та мобільної техніки.

Метою дипломної роботи є розробка та впровадження вебдодатку, який дозволить оптимізувати процеси обслуговування техніки, поліпшити якість послуг та забезпечити зручний інтерфейс для взаємодії з клієнтами.

Задачі дослідження:

1. Провести аналіз потреб та вимог за іншими сервісними центрами.

2. Розробити архітектуру вебдодатку та його функціонал.

3. Виконати моделювання структури та роботи додатку підтримки діяльності сервісного центру.

4. Реалізувати та протестувати розроблений вебдодаток.

Розроблений вебдодаток матиме практичне значення для сервісних центрів, забезпечуючи їм зручний та ефективний інструмент для управління процесами обслуговування техніки. Впровадження цього рішення сприятиме підвищенню ефективності роботи сервісних центрів, зменшенню часу обробки заявок та покращенню якості обслуговування клієнтів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд предметної області

Дані статистичних досліджень, проведених Rau.ua [1] ,показують стійке зростання обсягів продажів та експлуатації техніки протягом останніх років. Прогнозується, що цей тренд буде продовжуватися і набувати ще більшого розмаху в наступні роки. Це означає, що попит на послуги сервісних центрів буде лише зростати.

На основі даних з ресурсу Rau.ua можна помітити тенденцію стабільного зростання обсягів продажів. Українці повертаються до звичної сезонності покупок, і продажі наближаються до рівня, притаманного періоду перед війною. Крім того, за підсумками 2023 року ринок електроніки продемонстрував значне зростання – на 29% у гривневому еквіваленті порівняно з 2022 роком, при цьому продажі побутової техніки, ігрових моніторів, приставок та аксесуарів, ноутбуків, жорстких дисків, телевізорів, навушників і комп'ютерної техніки збільшились на третину, а в категорії "Мала побутова техніка" найбільше зростання показали мультипечі, стайлери для волосся, пилососи, кавоварки та електрогрилі, що загалом свідчить про поступове відновлення та нормалізацію українського роздрібного ринку.

Цей тренд свідчить про постійне зростання популярності цифрових пристроїв серед споживачів.

Основні завдання сервісних центрів включають:

1. Діагностика та ремонт: Прийом та обробка заявок на ремонт, виконання діагностики несправностей, проведення ремонтних робіт та тестування техніки перед поверненням клієнту.
2. Технічна підтримка: Надання консультацій та технічної підтримки клієнтам щодо налаштування, використання та усунення неполадок у роботі їхньої техніки.

3. Запасні частини та аксесуари: Управління запасними частинами та аксесуарами, включаючи замовлення, отримання, зберігання та використання.

4. Документація та звітність: Ведення документації про проведені роботи, створення звітів про продуктивність та ефективність діяльності сервісного центру.

У сучасних умовах ведення документації ручним способом в сервісних центрах є недоцільним та неефективним. Це призводить до великої кількості проблем, таких як втрата даних, помилки в обробці інформації та ускладнення управління процесами. Тому розробка вебдодатку для автоматизації та оптимізації роботи сервісних центрів є надзвичайно актуальною та потребує негайного впровадження.

На основі наведених даних стає очевидним, що розробка вебдодатку для сервісних центрів є нагальною потребою. Застосування вебдодатку дозволить автоматизувати процеси обробки заявок, ведення документації, а також забезпечить швидкий та зручний доступ до інформації для співробітників та клієнтів. Покращення ефективності роботи сервісного центру за допомогою вебдодатку дозволить збільшити задоволеність клієнтів, знизити час обробки заявок та збільшити продуктивність персоналу. Таким чином, розробка вебдодатку є кроком вперед у сучасній організації роботи сервісних центрів, що відповідає вимогам сьогодення та забезпечує конкурентні переваги на ринку.

1.2 Аналіз програмних продуктів - аналогів

Аналіз існуючих програмних аналогів є ключовим етапом у розробці вебдодатку, оскільки він дозволяє отримати цінну інформацію про популярні функції серед користувачів, виявити можливості для покращення, а також визначити, чим ваш продукт може вирізнятися на ринку.

У цьому розділі будуть розглянуті два програмних продукти-аналоги: ServiceDesk+ [2] та Freshdesk. [3] Для кожного з них буде надано загальний опис, перелік основних можливостей та огляд їх популярності, а також сильні та слабкі сторони.. Завершується розділ висновком, в якому робиться огляд здобутих з аналізу висновків та надаються рекомендації щодо покращення вашого вебдодатку на основі отриманих даних.

ServiceDesk+ - це інтегрована платформа для управління сервісними заявками, яка надає можливості для реєстрації та відслідковування замовлень на обслуговування, ведення бази даних клієнтів, планування робочого часу та аналізу продуктивності.

ServiceDesk+ пропонує широкий функціонал, включаючи автоматизацію процесів, зручний користувацький інтерфейс та розширені засоби аналітики. Завдяки цьому функціоналу, продукт набув популярності у бізнес-середовищі та серед компаній, які шукають ефективне рішення для управління своєю сервісною діяльністю.

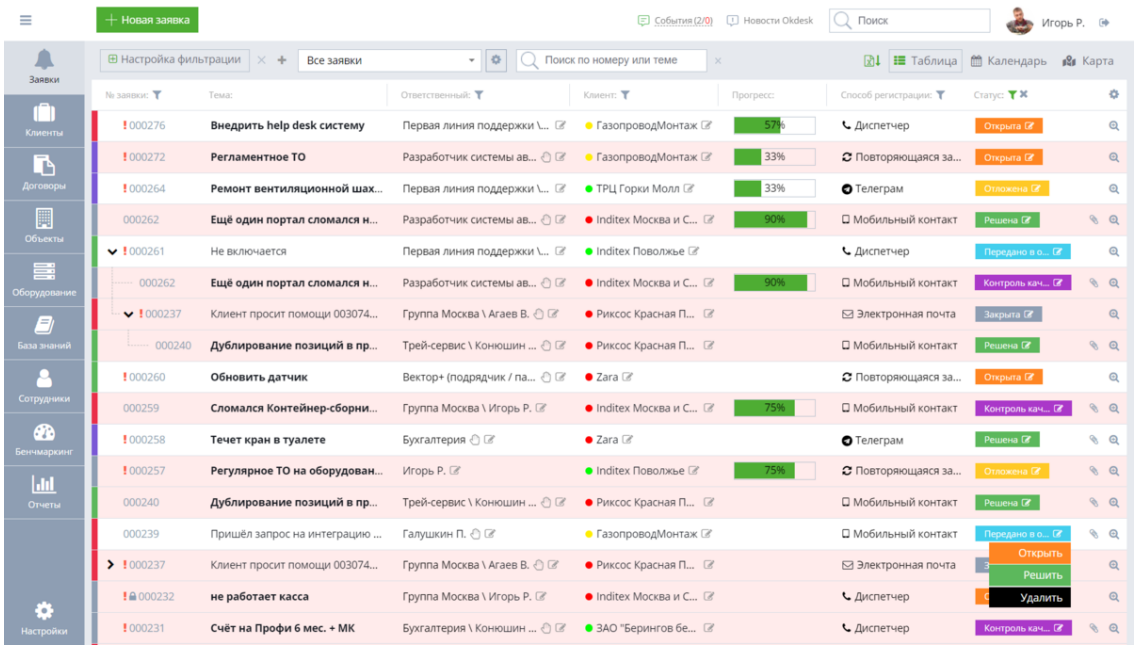


Рисунок 1.1 – Рисунок додатку ServiceDesk+

Переваги ресурсу:

- розширений функціонал для управління сервісними заявками тобто додаткові інструменти, модулі, інтеграції з іншими сервісами та покращену аналітику;
- зручний та інтуїтивно зрозумілий інтерфейс користувача;
- потужні засоби аналітики та звітності наприклад: звіти про продуктивність, використання ресурсів, клієнтське обслуговування, наявних запасних частин, та фінансові звіти.

Слабкі сторони:

- високі витрати на впровадження та підтримку;

можливість перенавантаження інтерфейсу через велику кількість функцій.

Freshdesk - це інтерактивна платформа, яка призначена для управління взаємодією з клієнтами та сервісною підтримкою. Вона включає в себе систему квитків, базу знань, чат для спілкування з клієнтами та інші інструменти для покращення клієнтського досвіду.

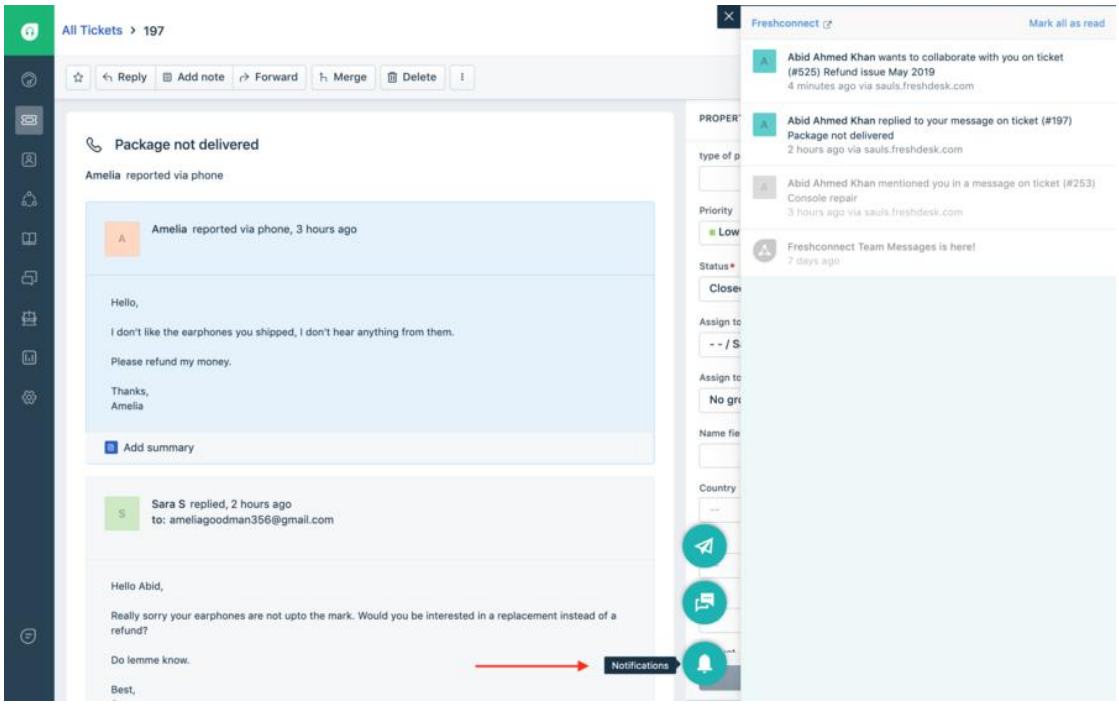


Рисунок 1.2 – Рисунок додатку Freshdesk

Freshdesk є дуже популярним інструментом [4] у сфері обслуговування клієнтів та сервісної підтримки. Він пропонує широкий спектр можливостей, таких як автоматизація рутинних завдань, інтеграція з різноманітними іншими інструментами та зручний інтерфейс.

Переваги:

- широкий спектр функцій для управління взаємодією з клієнтами.
- інтеграція з популярними додатками та сервісами.
- легка настройка та використання.

Слабкі сторони:

- потребує підписки на платну версію для доступу до ряду розширених функцій;
- деяка складність для новачків у налаштуванні деяких аспектів.

Таблиця 1.1 – Результати порівняння продуктів аналогів

Характеристика	**ServiceDesk+**	**Freshdesk**
Модуль управління заявками	Має розширений модуль управління заявками з можливістю автоматизації процесів та індивідуальними налаштуваннями	Також має потужний модуль управління заявками з можливістю автоматизації, проте обмежений у функціоналісті за замовчуванням
Аналітика	Потужні засоби аналітики та звітності	Пропонує засоби аналітики та звітності
Модуль чату	Вбудований чат для спілкування з клієнтами	Надає інтерактивний чат для спілкування з клієнтами
Інтеграція	Інтеграція з різноманітними іншими інструментами	Забезпечує інтеграцію з популярними сервісами та додатками
Вартість	Вища вартість в порівнянні з Freshdesk	Пропонує різні пакети цін залежно від функціоналу та потреб користувача

Аналіз аналогів показав перлік тих функцій та можливостей додатку, які потрібно використати при створенні власного вебдодатку, зосереджуючись на

спрощенні користувальницького інтерфейсу, вдосконаленні функціональності для аналізу даних та розширенні можливостей для автоматизації процесів.

1.3 Мета та задачі дослідження

Головною метою даної дипломної роботи є розробка та впровадження вебдодатку підтримки діяльності сервісного центру комп'ютерної та мобільної техніки на основі React. Цей вебдодаток спрямований на автоматизацію процесів управління та обслуговування техніки, що дозволить підвищити ефективність роботи сервісного центру та покращити задоволення клієнтів.

Задачі дослідження:

1. Проектування та розробка вебдодатку:

Розробити архітектуру вебдодатку, вибрати відповідні технології та засоби для реалізації функціоналу, а також розробити інтерфейс користувача з урахуванням сучасних UX/UI тенденцій.

- Створення детальної специфікації вимог до системи на основі результатів аналізу потреб користувачів.
- Розробка архітектури вебдодатку, включаючи складові бази даних, бізнес-логіку та інтерфейс користувача.

2. Тестування та вдосконалення:

Провести тестування вебдодатку на відповідність вимогам та виявлення можливих помилок та недоліків. Внести необхідні зміни та вдосконалення для забезпечення найвищої якості продукту.

- Проведення модульних, інтеграційних та системних тестів для перевірки роботи окремих компонентів та взаємодії між ними.
- Виявлення та виправлення помилок, а також оптимізація продукту для покращення його продуктивності та надійності.

Вимоги до вебдодатку:

1. Авторизація та управління користувачами:

- Можливість реєстрації нових користувачів та авторизації вже існуючих.
- Різні рівні доступу для адміністраторів та звичайних користувачів.

- Можливість управління профілями користувачів та їхніми правами.

2. Управління заявками та замовленнями:

- Можливість подання заявок на обслуговування або ремонт техніки.
- Відстеження статусу замовлення та отримання сповіщень про його зміни.
- Адміністративний інтерфейс для управління заявками, призначення спеціалістів, термінів виконання тощо.

3. Чат для зв'язку з клієнтами:

- Вбудований чат для спілкування клієнтів зі спеціалістами сервісного центру.
- Можливість обміну повідомленнями, відправлення файлів та зображень.

4. Керування інвентарем:

- Можливість ведення бази даних обладнання, запасних частин та інших матеріалів.
- Автоматичне оновлення інформації про наявність товарів та їхню кількість.

5. Аналітика та звітність:

- Генерація звітів щодо продуктивності та ефективності роботи сервісного центру.
- Аналіз часу виконання заявок, розподілу ресурсів та інших показників.

6. Інтеграція з платіжними системами:

- Можливість оплати послуг прямо через вебдодаток.
- Інтеграція з різними платіжними системами для зручності користувачів.

Цей функціонал і можливості дозволять забезпечити повноцінне та ефективне функціонування сервісного центру комп'ютерної та мобільної техніки та забезпечать задоволення потреб користувачів.

Повне технічне завдання на розробку міститься в додатку А.

2 МОДЕЛЮВАННЯ

2.1 Функціональне моделювання в IDEF0

Основна мета цього етапу виконання кваліфікаційної роботи - це детальний опис функціональності вебдодатку для сервісного обслуговування комп'ютерів. Використання IDEF0 дозволяє структурувати та візуалізувати процеси, які відбуваються в системі.

Контекстна діаграма моделі в нотації IDEF0 представлене на рисунку 2.1.

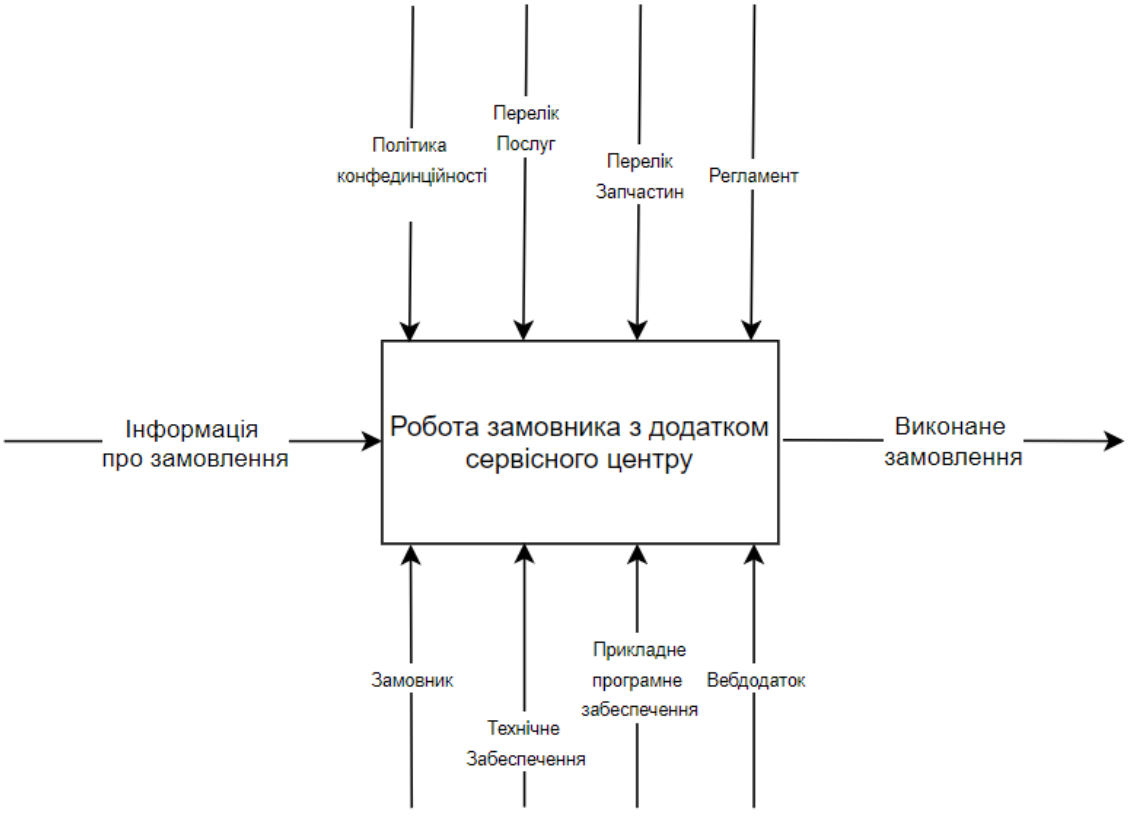


Рисунок 2.1 – IDEF0: контекстна діаграма

Далі виконується декомпозиція попередньої діаграми на основні етапи, що забезпечують реалізацію даного блоку.

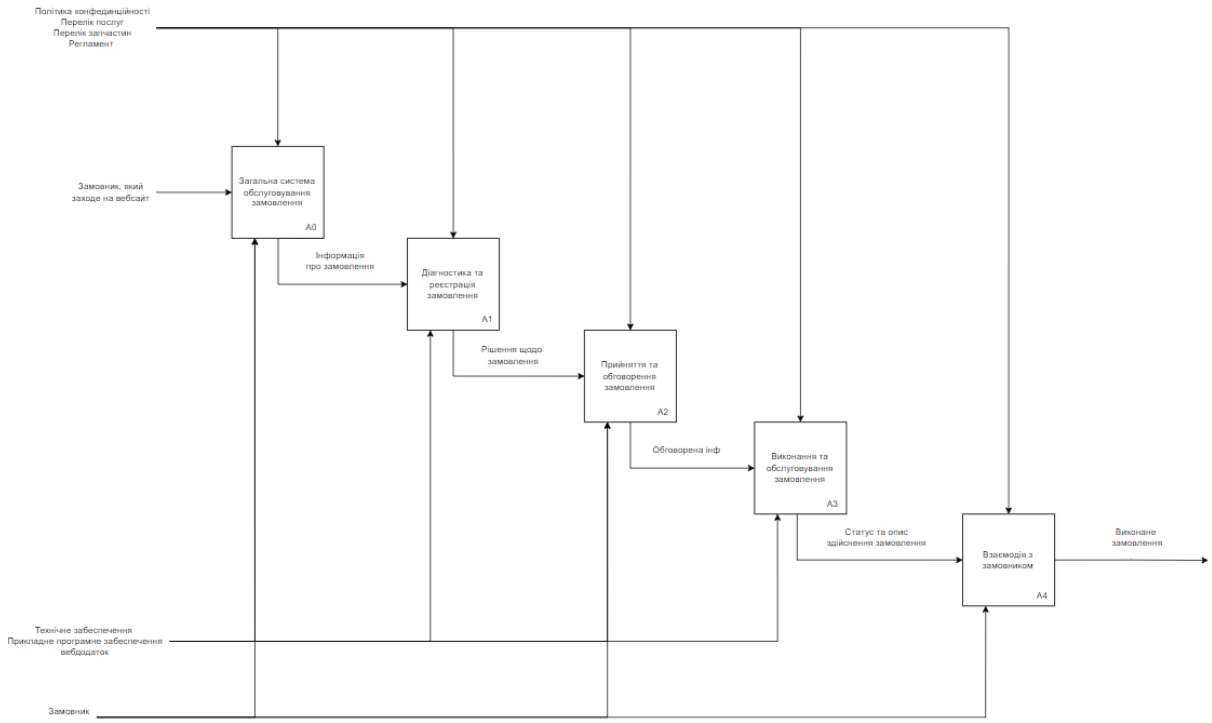


Рисунок 2.2 – Декомпозиція контекстної діаграми

На рівні вищої абстракції, функціональне моделювання IDEF0 для розробки Web-додатку може включати наступні ключові етапи:

A0. Створення замовлення:

- Функція: Створення замовлення.
- Входи: Замовник, який заходе на вебсайт.
- Виходи: Інформація про замовлення.

A1. Діагностика заявок:

- Функція: Діагностика та реєстрація замовлення.
- Входи: Інформація про пристрій та контактні дані.
- Виходи: рішення щодо замовлення.

A2. Прийняття та обговорення:

- Функція: Прийняття та обговорення замовлення.
- Входи: Прийняте рішення майстра щодо пристою, уточнюючі питання.
- Виходи: Обговорена інформація.

А3. Виконання та обслуговування:

- Функція: Виконання та обслуговування замовлення.
- Входи: Уточнююча інформація від замовника та дані з бази.
- Виходи: статус та опис здійснення замовлення.

А4. Взаємодія з замовником:

- Функція: Взаємодія з замовником.
- Входи: статус та документ про зроблену роботу.
- Виходи: Виконане замовлення.

2.2 Діаграма варіантів використання

Для досягнення цілей функціонування спочатку будується модель у формі діаграми варіантів використання (use-case diagram), яка описує функціональне призначення системи. Діаграма варіантів використання є вихідною концептуальною моделлю системи в процесі її проектування та розробки.

Краще після функціональних діаграм та доводити, що наведені варіанти використання реалізують необхідні функції

1. **Тех. Персонал авторизується в системі:**

Аргументація: Не кожен має мати доступ до адмін панелі та інших інструментів роботи з додатком.

2. **Замовник отримує повідомлення про виконання:**

Аргументація: Коли майстер виконав роботу він змінює дані в додатку про це замовлення на виконане, та телефонує йому за бажанням, але на почту прийде повідомлення.

3. **Замовник створює замовлення :**

Аргументація: Етап на якому замовник робить вибір між послугами чи замовленням запчастини.

4. **Тех. Персонал передає статус замовлення в БД:**

Аргументація: Після закінчення роботи майстер ставить статус в додатку на виконано, де БД отримує ці дані і відправляє повідомлення замовнику.

5. **Тех. Персонал отримує дані замовлення та передає в БД:**

Аргументація: Так коригується процес замовлення щоб дані були коректні для майстра.

6. **Тех. Персонал отримує замовлення**

Аргументація: Коли отримують замовлення передають його на майстра де він знайде всю інформацію в додатку.

Діаграма варіантів використання враховує основні функції системи та визначають способи їх реалізації, що в свою чергу сприяє ефективності та успішному функціонуванню інформаційної системи.

Діаграма варіантів використання в нотації UML представлена на рисунку 2.3.

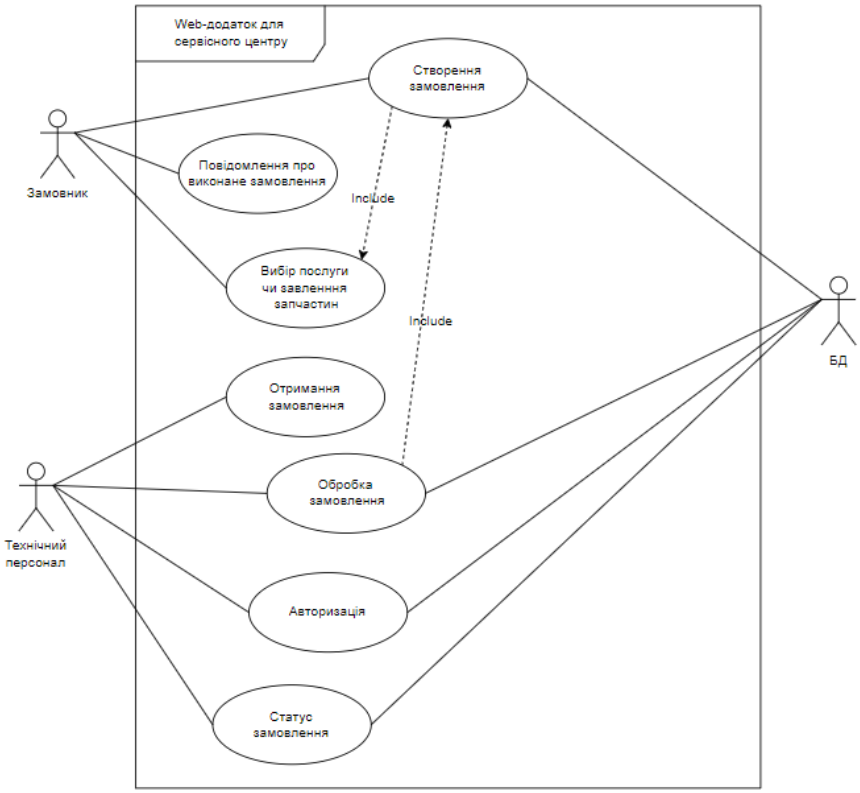


Рисунок 2.3 – Діаграма варіантів використання

2.3 Модель даних

Логічна модель даних для інформаційної системи з обслуговування комп'ютерів базується на потребах системи у збереженні та обробці інформації про клієнтів, технічний персонал, обладнання та замовлення. Модель передбачає використання реляційної бази даних для ефективного управління інформацією.

Також вона покликана забезпечити ефективне управління інформацією, зручний доступ до даних та забезпечити нормалізацію бази даних для уникнення зберігання дубльованої інформації. Вона враховує основні сутності та їх взаємозв'язки для оптимального функціонування інформаційної системи.

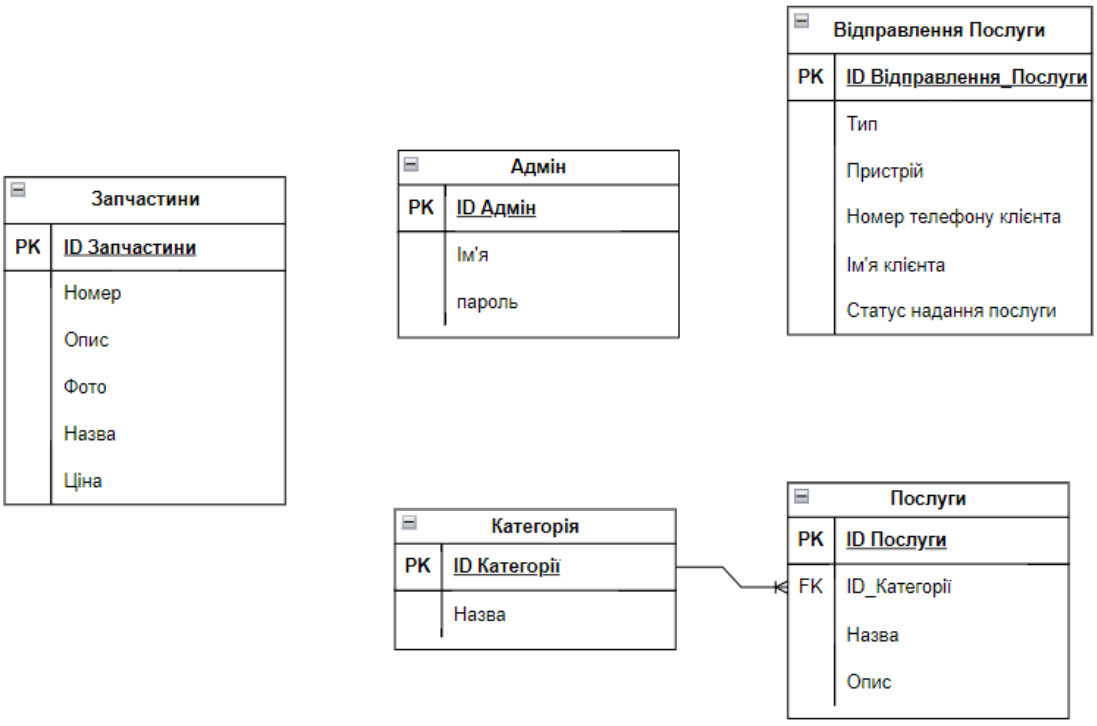


Рисунок 3.5 – Логічна модель даних

Основні елементи:

1. **Таблиця "Адмін":**

- Поля: ID адміна, ім'я, пароль.

2. **Таблиця "Категорія":**

- Поля: ID категорії, назва.

3. **Таблиця "Запчастини":**

- Поля: ID запчастин, номер, опис, фото, назва, ціна.

4. **Таблиця "Послуги":**

- Поля: ID послуги, ID категорії, назва, опис.
- Посилання: Категорія

5. **Таблиця "Відправлення послуги":**

- Поля: ID відправлення послуги, тип, пристрій, номер телефону клієнта, Ім’я клієнта, статус надання послуги.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ

3.1 Програмна реалізація

Програмна реалізація вебдодатку для підтримки діяльності сервісного центру комп'ютерної та мобільної техніки охоплює всі етапи розробки від проектування архітектури до впровадження функціональних модулів та інтеграції з базою даних. Основними цілями програмної реалізації були забезпечення зручного та інтуїтивного інтерфейсу користувача, стабільної роботи системи, висока продуктивність та безпека даних.

3.1.1 Архітектура системи

Вебдодаток побудований за трирівневою архітектурою, яка включає наступні компоненти:

Клієнтська частина (Frontend): відповідає за інтерфейс користувача та взаємодію з сервером через HTTP-запити. Реалізована за допомогою HTML, CSS, JavaScript та фреймворку React.

Серверна частина (Backend): обробляє запити від клієнтської частини, виконує бізнес-логіку та взаємодіє з базою даних. Реалізована за допомогою Node.js та фреймворку Express.

База даних (Database): зберігає всі дані додатку, включаючи інформацію про заявки, користувачів, запчастини та статуси ремонтів. Використовується реляційна база даних PostgreSQL.

3.1.2 Реалізація бази даних

Для зберігання даних використовується реляційна база даних PostgreSQL. Взаємодія з базою даних здійснюється за допомогою ORM-

бібліотеки Sequelize, що забезпечує зручну роботу з даними через об'єктно-реляційне відображення (ORM). Основні сутності бази даних включають:

Майстри (Admins): інформація про користувачів системи, їх ролі, які дають право на адмін панель.

Відправлення послуги (ServicesSend): Процес послуги, його тип та пристрій з даними клієнта що замовив та статус його обробки

Послуги (Services): Інформація про послугу та категорія послуги.

Категорії (Categories): Інформація про категорію.

Запчастини (SpareParts): Дані про запчастини з своїм номером описом, фото та назвою, з ціною.

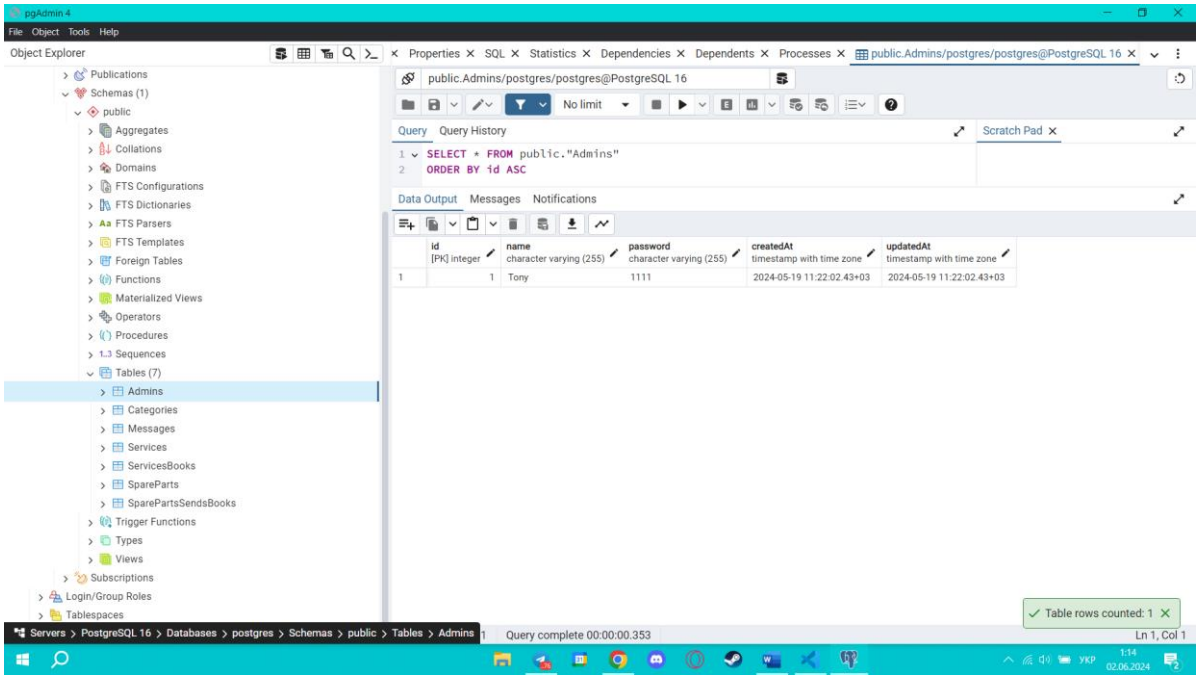


Рисунок 3.1 – Адмін таблиця

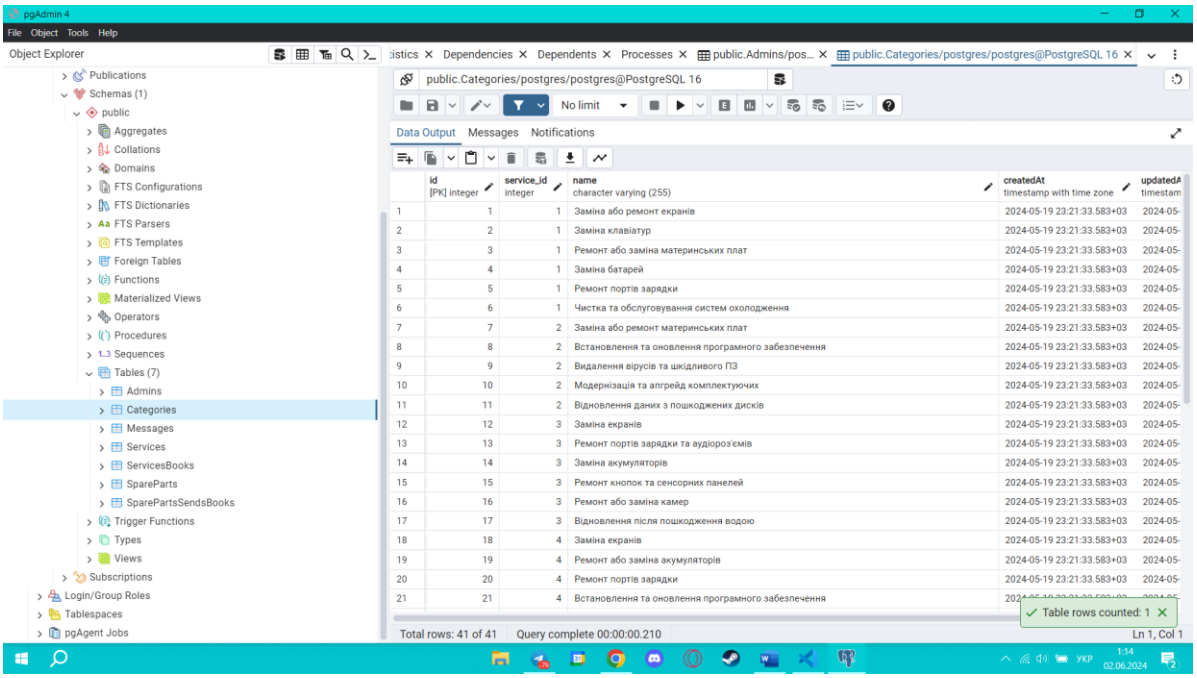


Рисунок 3.2 – Таблиця Категорій

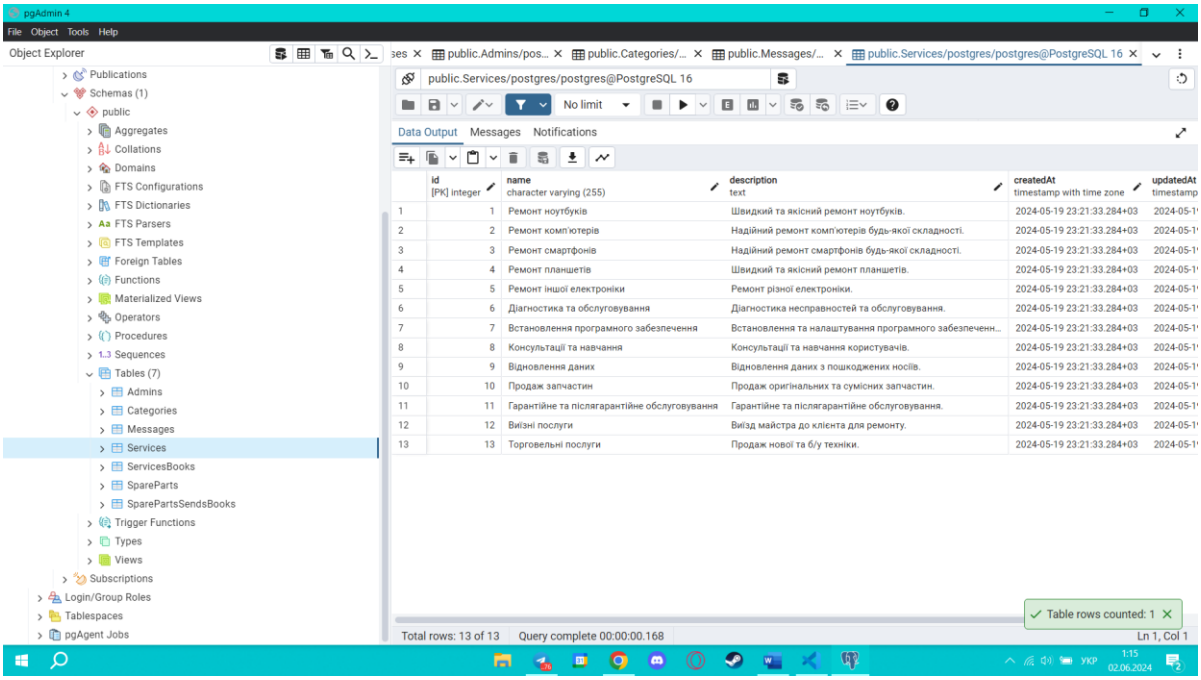


Рисунок 3.3 – Таблиця послуг

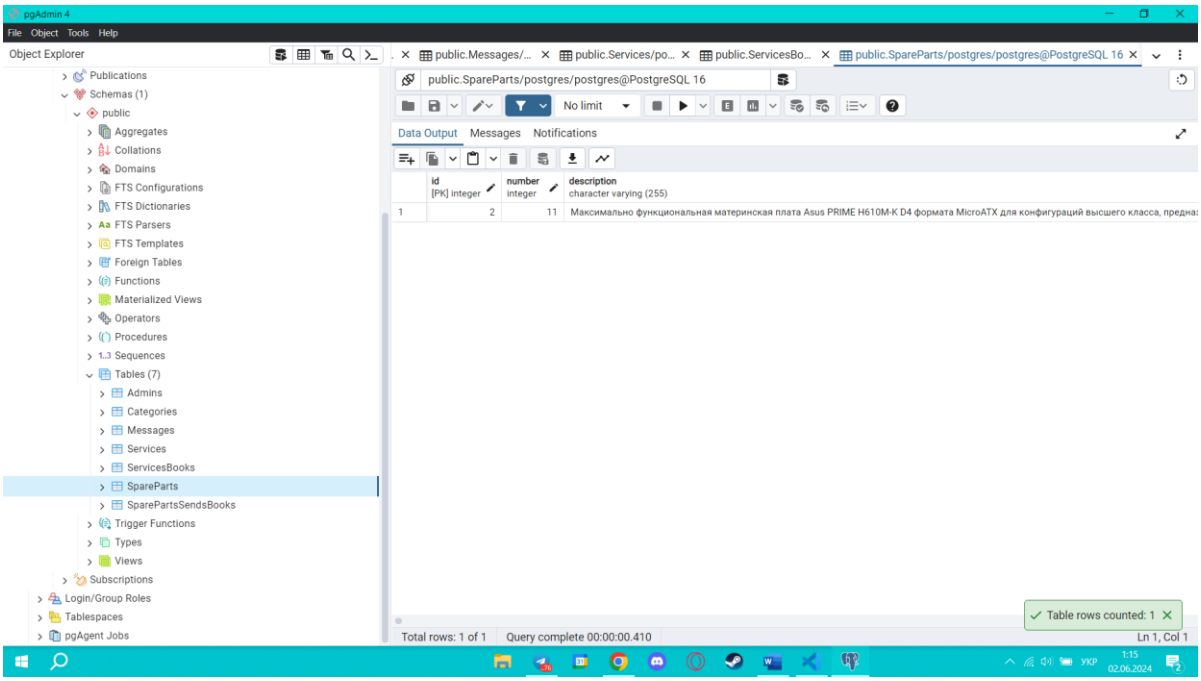


Рисунок 3.4 – Таблиця запчастин

3.1.2 Розробка серверної частини

Серверна частина вебдодатку була реалізована за допомогою платформи Node.js та фреймворку Express, які забезпечують високу продуктивність, масштабованість та зручність у розробці. Основні компоненти серверної частини включають:

- **Роутери (Routers)**: обробляють HTTP-запити від клієнтської частини, викликають необхідні контролери та повертають відповіді у форматі JSON.
- **Контролери (Controllers)**: містять бізнес-логіку додатку, включаючи обробку заявок, управління користувачами та замовленнями на запчастини.
- **Моделі (Models)**: відповідають за взаємодію з базою даних через ORM (Object-Relational Mapping) за допомогою бібліотеки Sequelize.

3.1.3 Розробка клієнтської частини

Клієнтська частина вебдодатку реалізована за допомогою React, що забезпечує динамічний та інтерактивний інтерфейс користувача. Основні компоненти клієнтської частини включають:

- **Компоненти (Components)**: відокремлені частини інтерфейсу, що відповідають за відображення та взаємодію з користувачем, такі як форми створення заявок, списки заявок, детальні перегляди тощо.
- **Сторінки (Pages)**: основні сторінки додатку, які об'єднують компоненти та забезпечують навігацію між ними.
- **Сервіси API (API Services)**: функції для виконання HTTP-запитів до серверної частини та обробки отриманих даних.

3.2 Використання розробленого продукту

Розроблений вебдодаток забезпечує такі основні функціональні можливості.

- Управління замовленнями на ремонт: клієнти можуть створювати, а майстри переглядати та редагувати замовлення на ремонт техніки.
- Управління замовленнями запчастин: користувачі можуть оформляти замовлення на необхідні запчастини, відстежувати їх статус та оновлювати інформацію.
- Технічна підтримка: система дозволяє зберігати та переглядати запити користувачів, необхідну для виконання ремонтних робіт чи відповідей на питання.
- Звітування та аналітика: майстри мають можливість генерувати звіти про виконані роботи, використані запчастини та інші важливі показники.

Інтерфейс користувача розроблений таким чином, щоб забезпечити максимальну зручність та ефективність при виконанні щоденних завдань. На головній сторінці користувач бачить кнопки з доступом до всіх основних функцій системи.

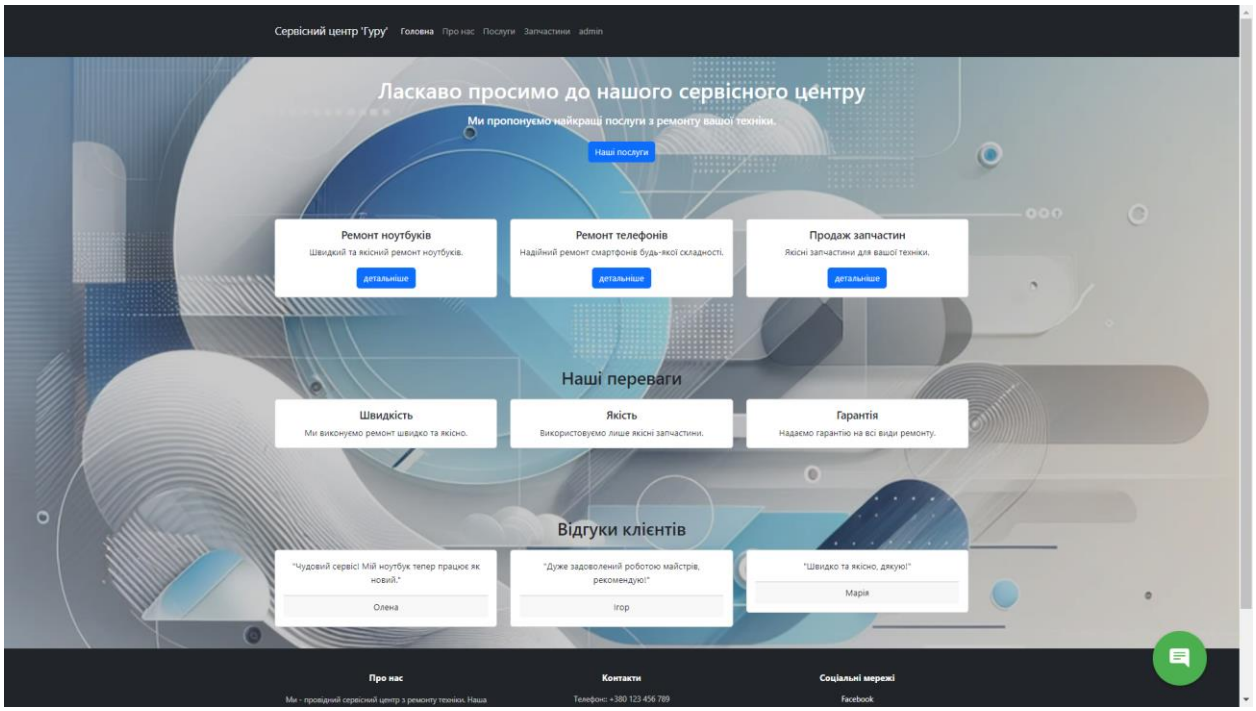


Рисунок 3.5 – Головна сторінка

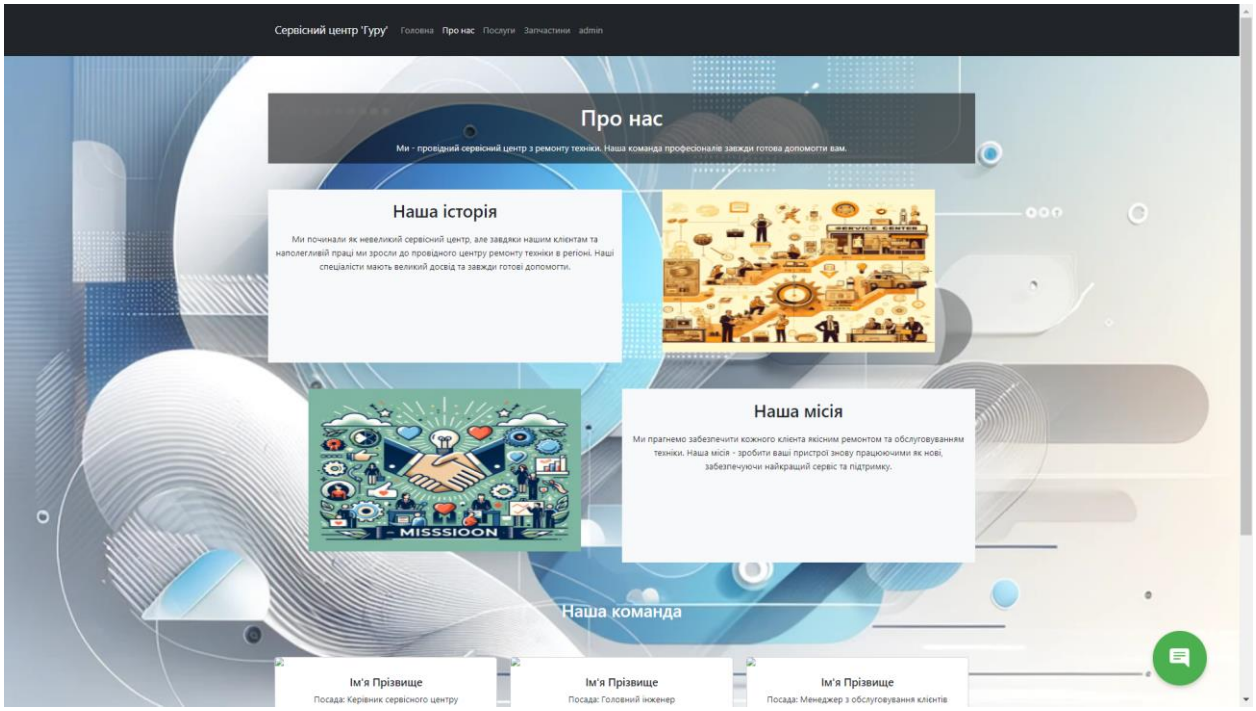


Рисунок 3.6 – Сторінка «Про нас»

Процес управління замовленнями на ремонт

Користувач може створити нове замовлення на ремонт через відповідну форму, заповнюючи необхідні поля, такі як ПІБ, опис проблеми(повідомлення), контактні дані клієнта.

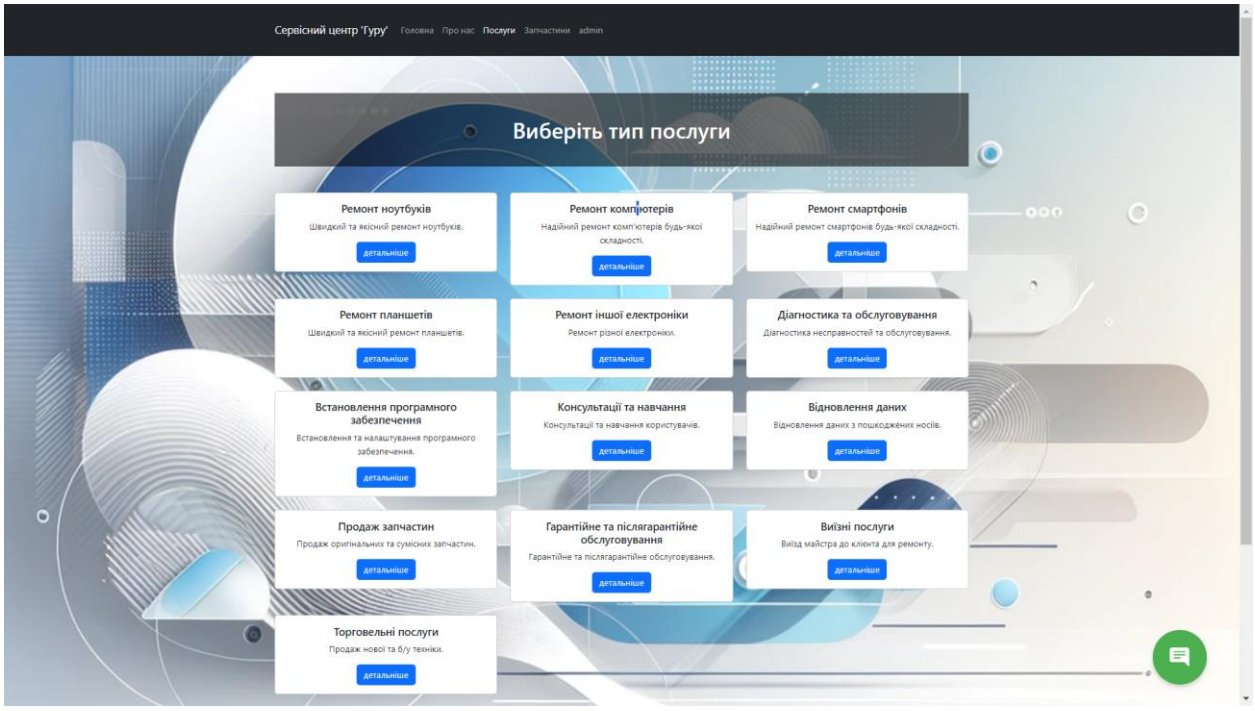


Рисунок 3.7 – Сторінка послуг

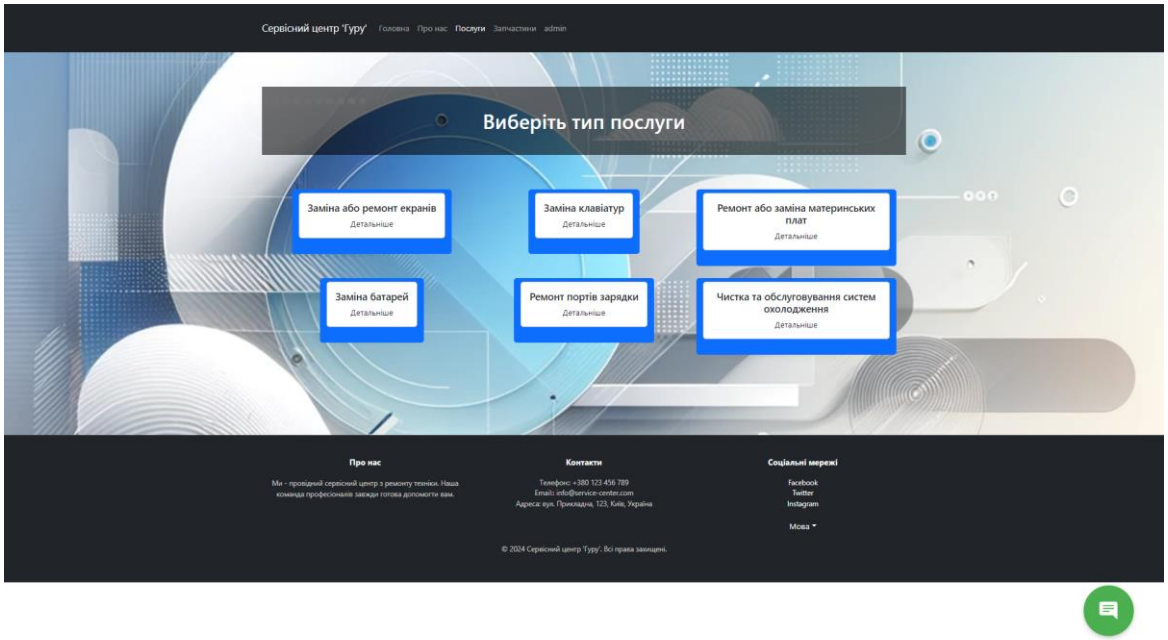


Рисунок 3.8 – Сторінка типу послуг

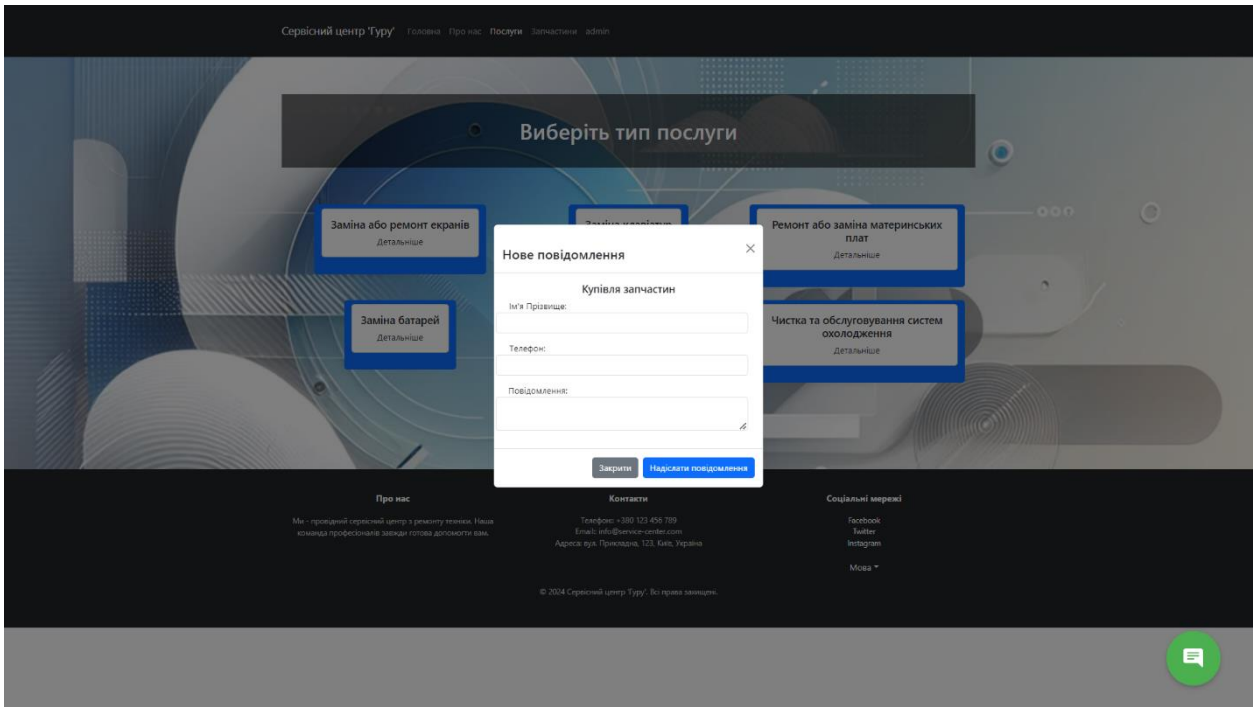


Рисунок 3.9 – Вікно замовлення послуги

Процес управління замовленнями запчастин

Система дозволяє користувачам оформляти замовлення на запчастини через окрему форму, де вони можуть зазначити необхідні деталі, кількість та бажану дату доставки.

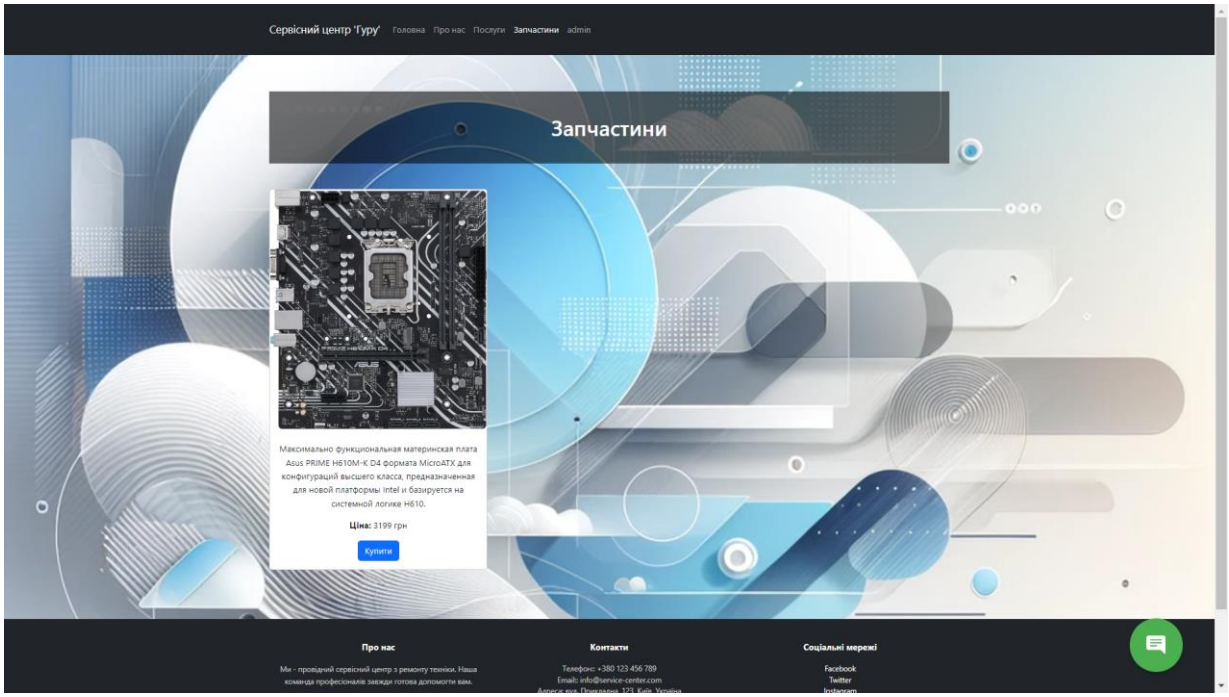


Рисунок 3.10 – Сторінка каталогу запчастин

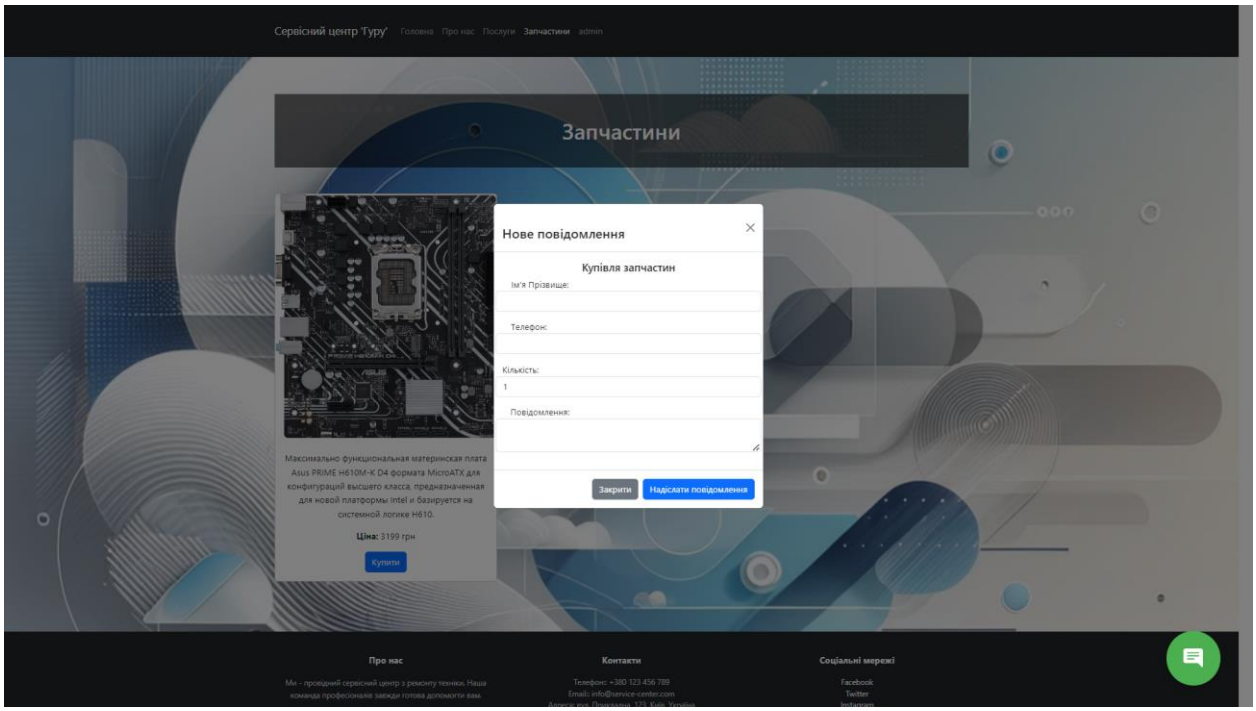


Рисунок 3.11 – Вікно замовлення запчастин

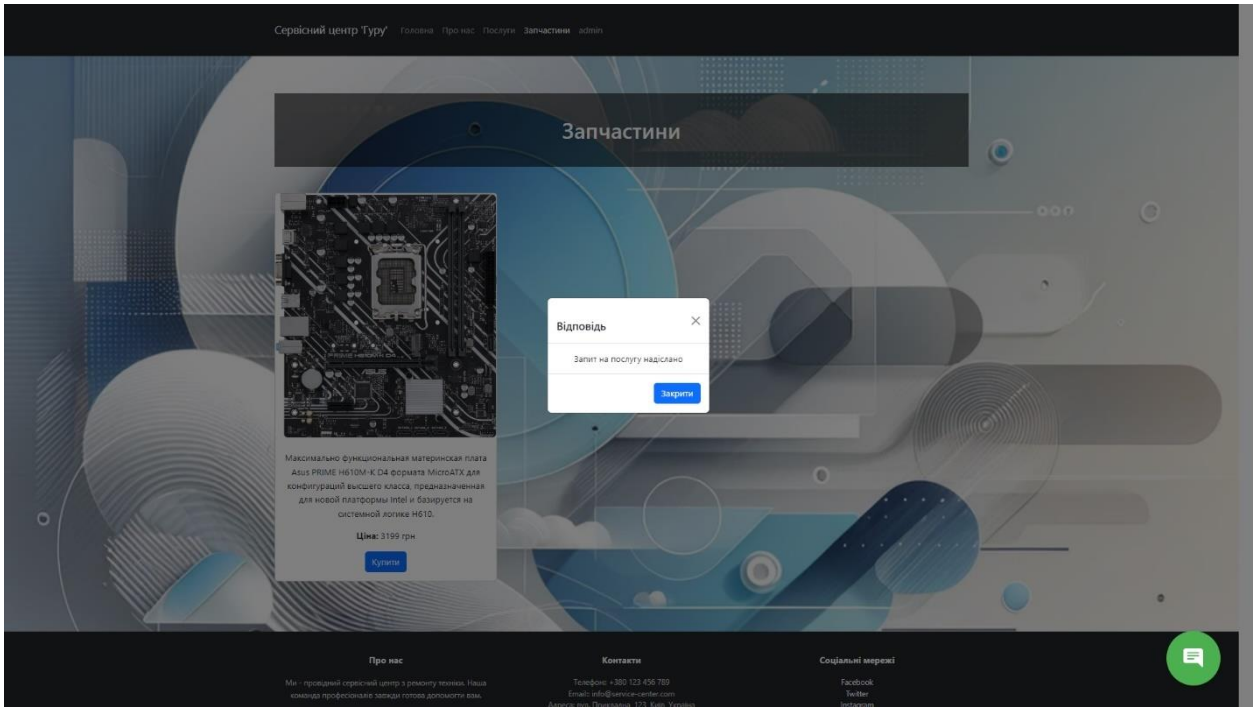


Рисунок 3.12 – Вікно відповіді на замовлення

Технічна підтримка

Для зручності користувачів, технічна підтримка зберігається в системі у вигляді логів, які можна переглядати, завантажувати та видаляти. Це значно спрощує процес доступу клієнта з сервісним центром.

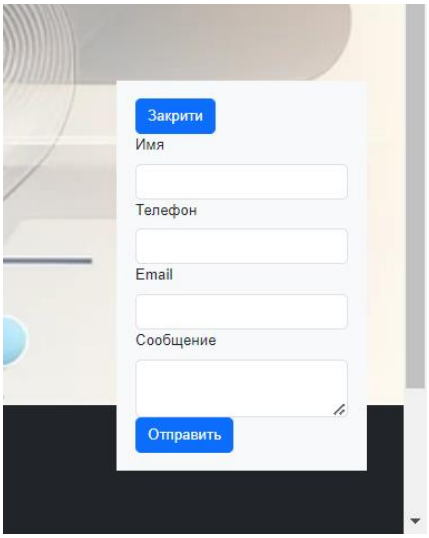


Рисунок 3.13 – Вікно технічної підтримки

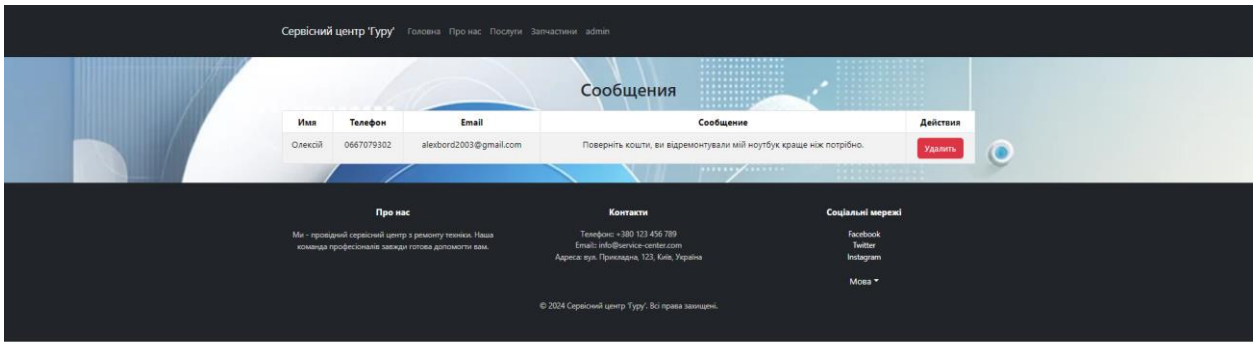


Рисунок 3.14 – Сторінка логів повідомлень

Звітування та аналітика

Майстри мають можливість генерувати різні звіти для аналізу діяльності сервісного центру. Це можуть бути звіти про кількість виконаних замовлень, використані запчастини, час виконання робіт та інші важливі показники.

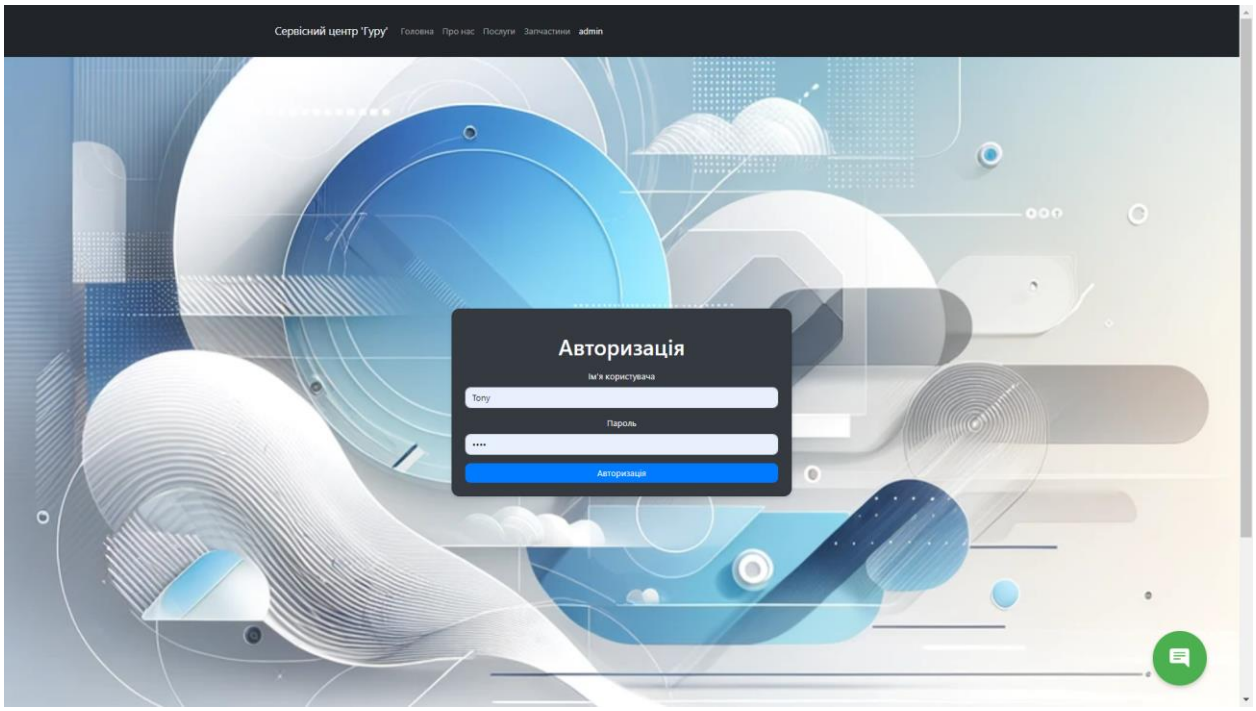


Рисунок 3.15 – Сторінка авторизації

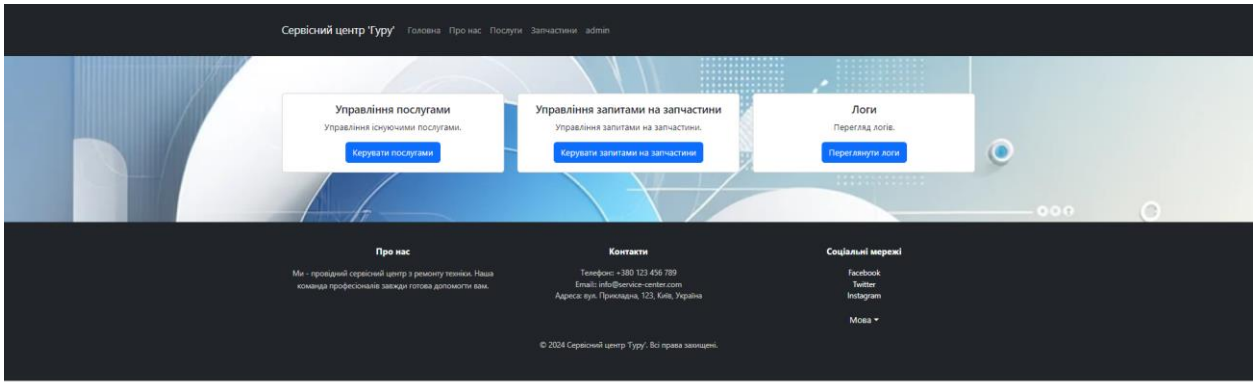


Рисунок 3.16 – Сторінка адмін панелі

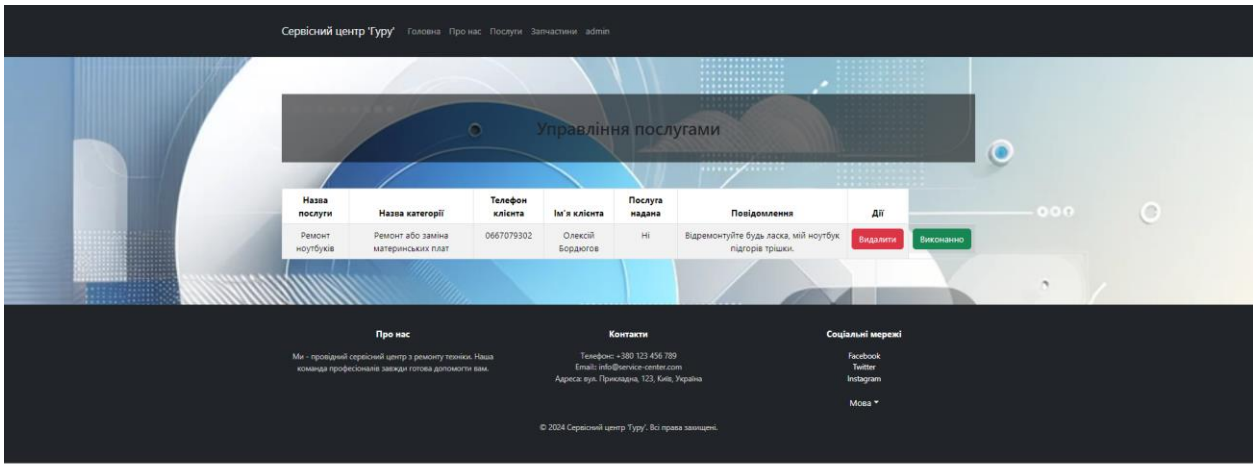


Рисунок 3.17 – Сторінка управління послугами

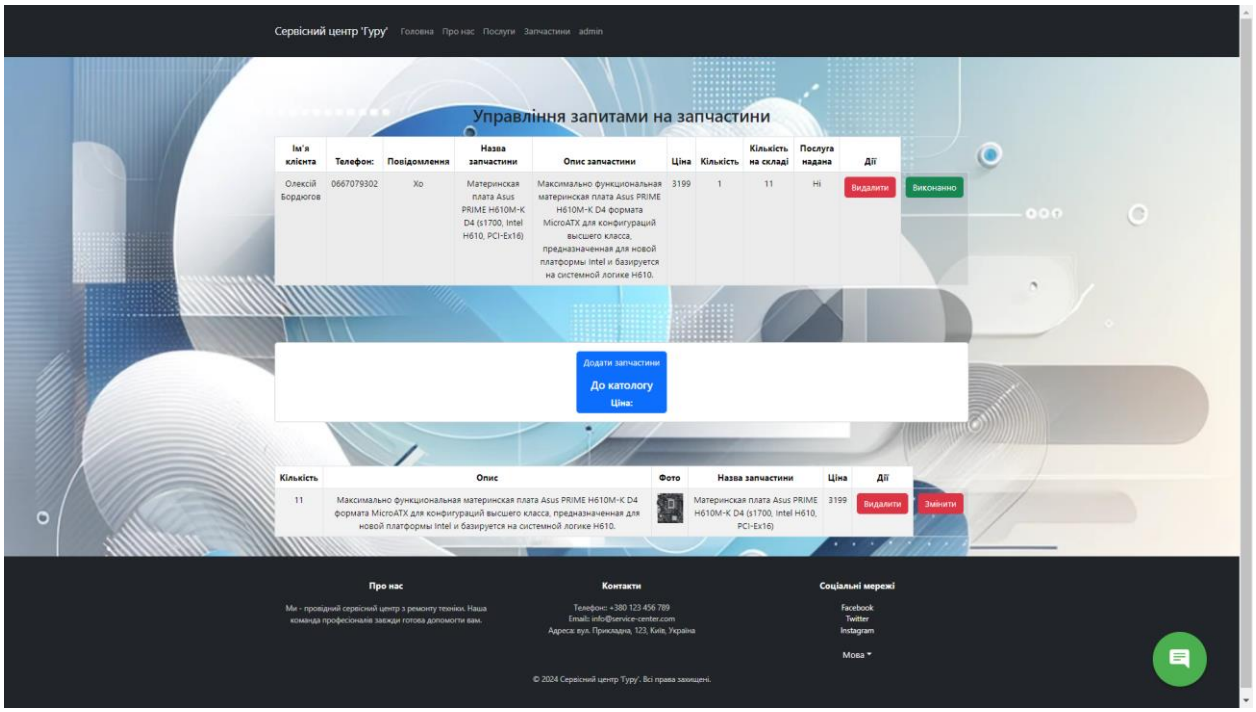


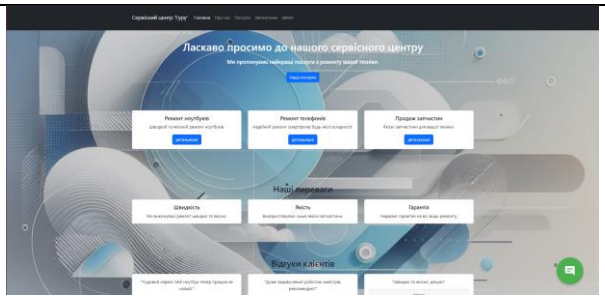
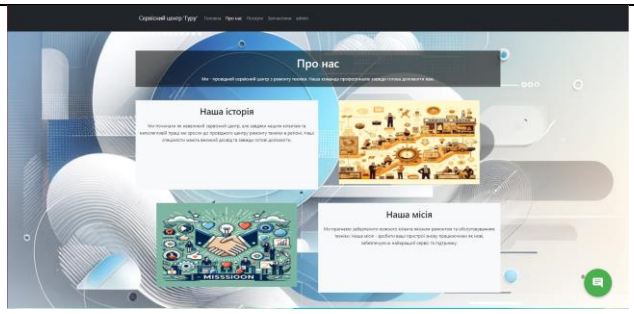
Рисунок 3.18 – сторінка управління запитами та запчастинами

Використання розробленого вебдодатку дозволяє значно підвищити ефективність роботи сервісного центру комп'ютерної та мобільної техніки. Завдяки зручному інтерфейсу, широкому спектру функцій та можливості генерування детальних звітів, система допомагає оптимізувати робочі процеси та покращити якість обслуговування клієнтів.

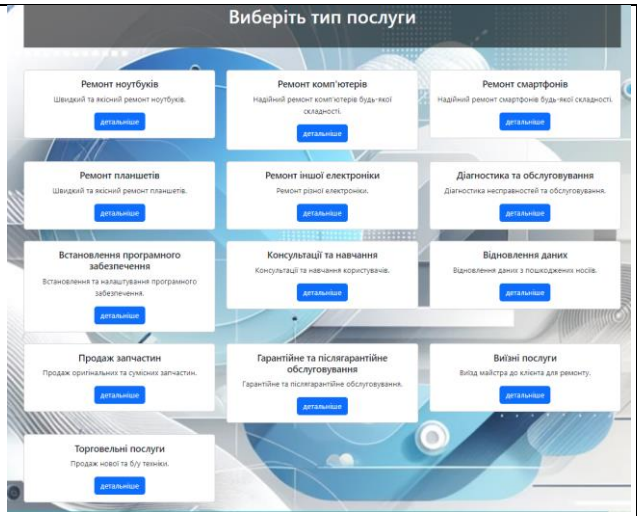
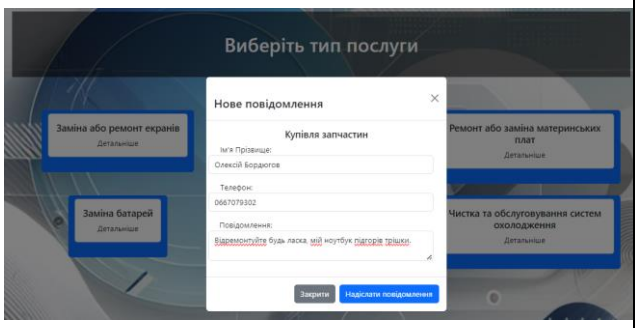
3.3 Тестування розробленого вебдодатку

Після завершення основної розробки було мануальне тестування функціоналу вебдодатку. Також було налаштовано середовище розгортання та забезпечено автоматичне оновлення системи.

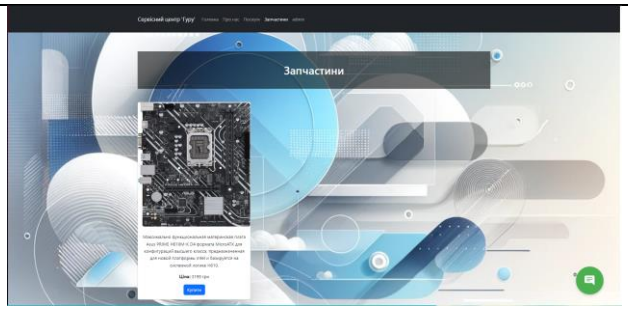
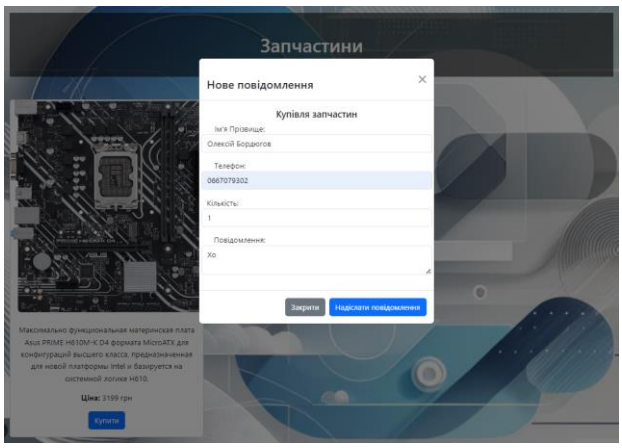
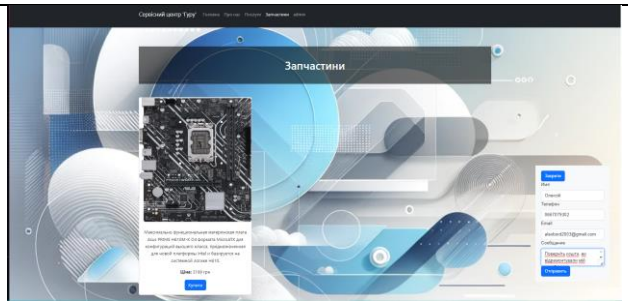
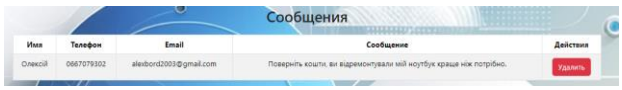
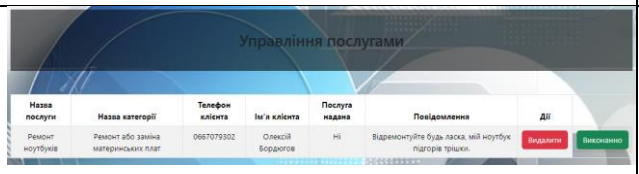
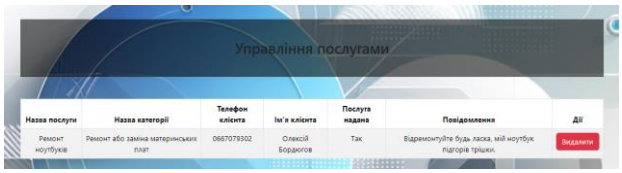
Таблиця 3.1 – Тестування вебдодатку

Назва	**Очікуваний результат**	**Фактичний результат**	**0/1**
Перевірка головної сторінки	Сторінка завантажується правильно, відображає інформацію про послуги, продаж, переваги та відгуки, навігаційне меню		1
Перевірка сторінки про нас	Сторінка завантажується правильно, відображає інформацію про історію, місію,		1

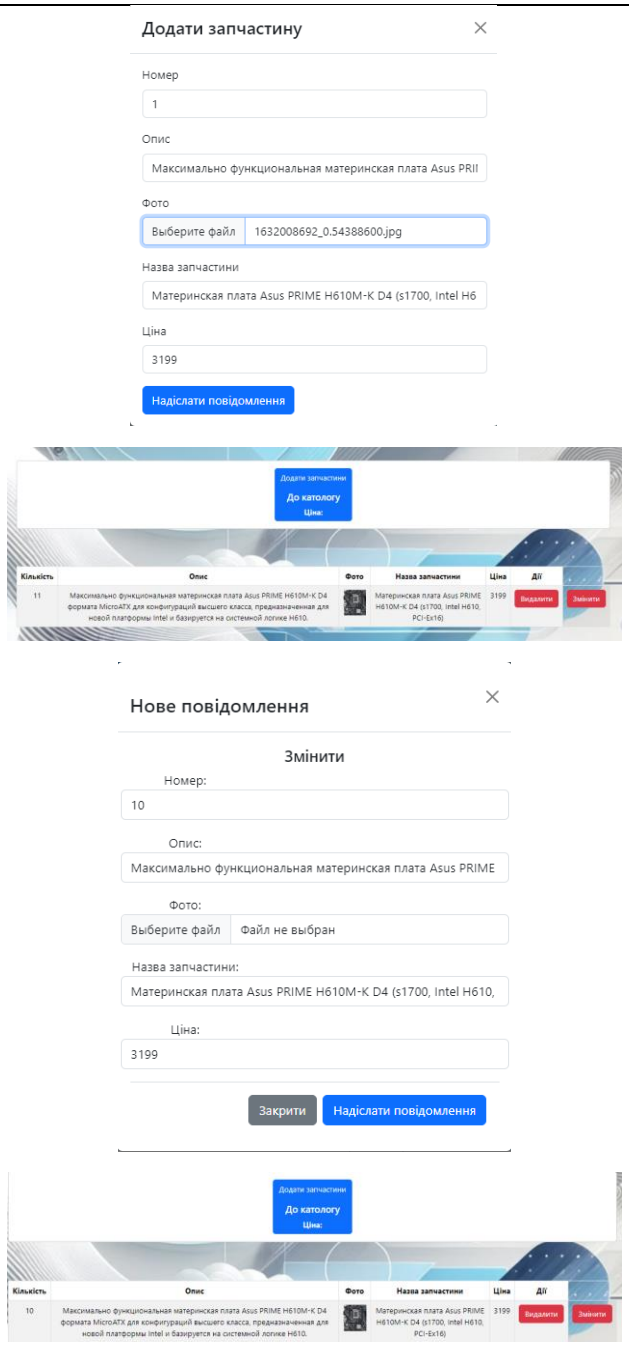
Продовження табл. 3.1

Перевірка роботи замовлення послуг	На сторінках всі послуги та категріх з БД вигружаються та замовлення на БД відправляється	   	1

Продовження табл. 3.1

Перевірка роботи замовлення запчастин	Всі запчастини загружені дані відображаються коректно, замовлення проходить, як треба і зв'зок з БД є	  	1
Перевірка роботи зв'язка з тех. персоналом	Повідомлення відправилось в БД, і відобразилось в адмін панелі	 	1
Перевірка виконаного завдання	Повідомлення на БД є	 	1

Продовження табл. 3.1

Перевірка роботи адміна на панелі для запчастин	Додається запчастина без проблем, як і змінюється		1

ВИСНОВКИ

Кваліфікаційна робота на тему "Вебдодаток підтримки діяльності сервісного центру комп'ютерної та мобільної техніки" спрямована на розробку інтегрованого рішення для оптимізації процесів обслуговування клієнтів та управління ремонтними замовленнями в сервісному центрі. Підсумовуючи результати виконаної роботи, можна зробити наступні висновки:

В сучасних умовах зростаючого використання комп'ютерної та мобільної техніки, кількість ремонтних заявок постійно збільшується. Це вимагає впровадження ефективних інструментів для автоматизації та оптимізації роботи сервісних центрів.

Ручне ведення документації стає все менш доцільним, оскільки це призводить до затримок, помилок та низької продуктивності. Використання сучасних вебтехнологій дозволяє подолати ці проблеми і значно підвищити ефективність роботи.

Метою роботи було створення вебдодатку, який дозволить автоматизувати процеси прийому, обробки та управління заявками на ремонт, замовлення запчастин та забезпечить зручний інтерфейс для користувачів та адміністраторів сервісного центру.

Створено інтуїтивно зрозумілий користувацький інтерфейс, що включає головну сторінку, форми для створення заявок, списки заявок, детальні перегляди заявок, адміністративну панель, сторінку користувача та розділ звітів та аналітики. Здійснено інтеграцію додатку з базою даних PostgreSQL, що забезпечує надійне зберігання та обробку даних. Реалізовано адміністративну панель з інструментами для обробки замовлень на ремонт, замовлень запчастин, редагування даних і зв'язку з технічною підтримкою. Проведено тестування системи, що підтвердило її стабільність, надійність та високу продуктивність.

Використання розробленого вебдодатку в сервісний центр дозволяє значно підвищити ефективність роботи, зменшити час обробки заявок, покращити управління ресурсами та підвищити задоволеність клієнтів.

Використання сучасних вебтехнологій забезпечує зручність та доступність системи з будь-якого пристрою, що має доступ до інтернету.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Rau. В Алло підбили підсумки: за 2023 рік ринок електроніки виріс на третину. URL: https://rau.ua/novyni/novini-partneriv/allo-pidsumki-za-2023-rik-rinok/ (дата звернення: 18.05.2024).

2. Документація ServiceDesk+. URL: https://www.manageengine.com/products/service-desk/ (дата звернення: 18.05.2024).

3. Документація Freshdesk. URL: https://freshdesk.com/support (дата звернення: 18.05.2024).

4. GetApp. Freshdesk Reviews - Pros & Cons, Ratings & more. URL: GetApp (дата звернення: 18.05.2024).

5. Гроховський І.Ф., Павленко І.П., Захарченко В.О. Основи програмування на JavaScript. Навчальний посібник. – Київ: Видавництво "Освіта", 2020. – 256 с.

6. Карпов А.В. Веброзробка з використанням React. – Санкт-Петербург: Пітер, 2021. – 320 с.

7. Казаков А.Ю. Архітектура вебдодатків: Підручник. – Харків: ХНУРЕ, 2019. – 350 с.

8. Хомяков П.М., Левченко В.В. Проектування баз даних: Навчальний посібник. – Львів: Вид-во ЛНУ ім. Івана Франка, 2018. – 288 с.

9. Сапрунов М.В., Поляков Д.С. Тестування програмного забезпечення: Підручник для вишів. – Київ: Видавництво "Освіта", 2017. – 312 с.

10. Левін О.М. Управління ІТ-проектами: Навчальний посібник. – Київ: КНЕУ, 2016. – 280 с.

11. Єфімов С.М. Безпека інформаційних систем: Практичний посібник. – Одеса: Видавництво "Чорномор'я", 2018. – 230 с.

12. Документація React. URL: https://reactjs.org/docs/getting-started.html (дата звернення: 18.05.2024).

13.Документація Node.js. URL: https://nodejs.org/en/docs/ (дата звернення: 18.05.2024).

14. Документація PostgreSQL. URL: https://www.postgresql.org/docs/ (дата звернення: 18.05.2024).

ДОДАТОК А

ТЕХНІЧНЕ ЗАВДАННЯ
на розробку інформаційної системи
«Вебдодаток підтримки діяльності сервісного центру комп'ютерної та мобільної техніки»

ПОГОДЖЕНО:

Доцент кафедри інформаційних технологій

_____ Ващенко С.М.

Студент групи ІТ-02

_____ Бордюгов О.М.

Суми 2024

1. Призначення й мета створення вебдодатку

1.1 Призначення вебдодатку

Вебдодаток призначений для організації діяльності сервісного центру комп’ютерної та мобільної техніки.

1.2 Мета створення вебдодатку

Головна мета проекту – це створення вебдодатку для підтримки діяльності сервісного центру, використання якого забезпечить належну організацію роботи даної компанії за рахунок автоматизації частини бізнеспроцесів.

1.3 Цільова аудиторія

Цільовою аудиторією даного проекту є замовник та клієнти сервісного центру, які зацікавлені послугами ремонту та/або покупкою запчастин чи інших товарів.

2 Вимоги до вебдодатку

2.1 Вимоги до вебдодатку в цілому

2.1.1 Вимоги до структури й функціонування

Вебдодаток організації діяльності сервісного центру повинен бути реалізований за допомогою вебінструментів та забезпечувати визначений набір функціональних можливостей.

Кінцевий продукт даного проекту має бути представлений вебдодатком, який містить якісне інформаційне наповнення та графічні матеріали.

2.1.2 Вимоги до персоналу

Персонал сервісу, як і користувачі, не повинен мати особливих технічних навичок для роботи з вебдодатком і його підтримкою. Єдиною вимогою є наявність навичок користування персональним комп’ютером та веббраузером.

2.1.3 Вимоги до збереження інформації

Уся інформація надана у вебдодатку повинна зберігатися у базі даних реалізованій засобами системи управління базами даних PostgreSQL.

2.1.4 Вимоги до розмежування доступу

Розроблюваний вебдодаток має бути загальнодоступним у мережі Інтернет. Права доступу до інформації розмежовані за групами користувачів: адміністратор, відвідувач та клієнт. Адміністратор має необмежений доступ до даних з правами перегляду, додавання, редагування та видалення товарів та послуг. Доступ до адміністративної панелі надається за спеціальним логіном та паролем.

Відвідувач вебдодатку може тільки переглядати інформацію на вебсторінках та надсилати повідомлення за допомогою форми зворотного зв’язку. У клієнта вебдодатку спектр доступу до інформації ширший за відвідувача, але менший за адміністратора. До переліку його можливостей входять ті, які визначені в групі користувачів «Відвідувач» та можливість з переглядом історії замовлень та вподобань.

2.2 Структура вебдодатку

2.2.1 Загальна інформація про структуру вебдодатку

До структури вебдодатку входять усі його вебсторінки, які є загальнодоступними, та адміністративна панель для користування персоналом сервісного центру.

Перелік сторінок вебдодатку наступний:

− «Головна» сторінка містить навігаційне меню, яке закріплене на кожній сторінці вебдодатку, карусель із зображеннями акційних пропозицій та оголошень, форму зв’язку з адміністрацією закладу, кнопку переходу до особистого кабінету та кнопку оформлення замовлення;

− на сторінці «Запчастини» розміщені категорії асортименту запчастин закладу, при виборі певного виду запчастин – повинен здійснюватися перехід на сторінку з усіма запчастинами цієї категорії;

− сторінка «Створити замовлення» призначена для створення Відвідувачем вебдодатку персонального замовлення за його вподобаннями: підібрати начинку ком’ютера чи ноутбука та інших запчастин;

− сторінка «Доставка і оплата» містить інформацію про способи доставки й оплати замовлень;

− призначенням сторінки «Контакти» є демонстрація способів зв’язку з адміністрацією та прикріплена Google-карта з міткою місця розташування закладу;

− для переходу на сторінку «Кабінет» користувач може клікнути по закріпленій кнопці «Увійти» із будь-якої сторінки вебдодатку. Ця сторінка містить історію замовлень та уподобань клієнта.

2.2.2 Навігація

Для зручної навігації повинно бути створене меню, що забезпечить швидке переміщення користувача по всім доступним сторінкам вебдодатку. Меню має бути закріплене і розташовуватися зверху (у шапці) на кожній сторінці.

2.2.3 Управління контентом

Управління контентом вебдодатку має здійснюватися за допомогою адміністративної панелі. Усе інформаційне наповнення вебдодатку має міститися у базі даних. Графічні матеріали та інформацію для наповнення надає Замовник.

2.2.4 Дизайн вебдодатку

Дизайн вебдодатку має бути виконаний у мінімалістичному та сучасному стилі. Корпоративними кольорами кондитерського закладу є білий, чорний та палітра відтінків синього кольору. Тому під час розробки вебдодатку треба використовувати саме ці кольори.

Види і розміри шрифтів повинні бути комфортними для перегляду. Інформаційні блоки, графічні матеріали та інші елементи вебсторінок повинні мати зручне і логічне розташування. Шаблон сторінки майбутнього програмного продукту зображено на рисунку А.1

Рисунок А.1 – Схема головної сторінки

2.2.5 Система навігації (карта вебдодатку)

Карта вебдодатку зображена на рисунку А.2.

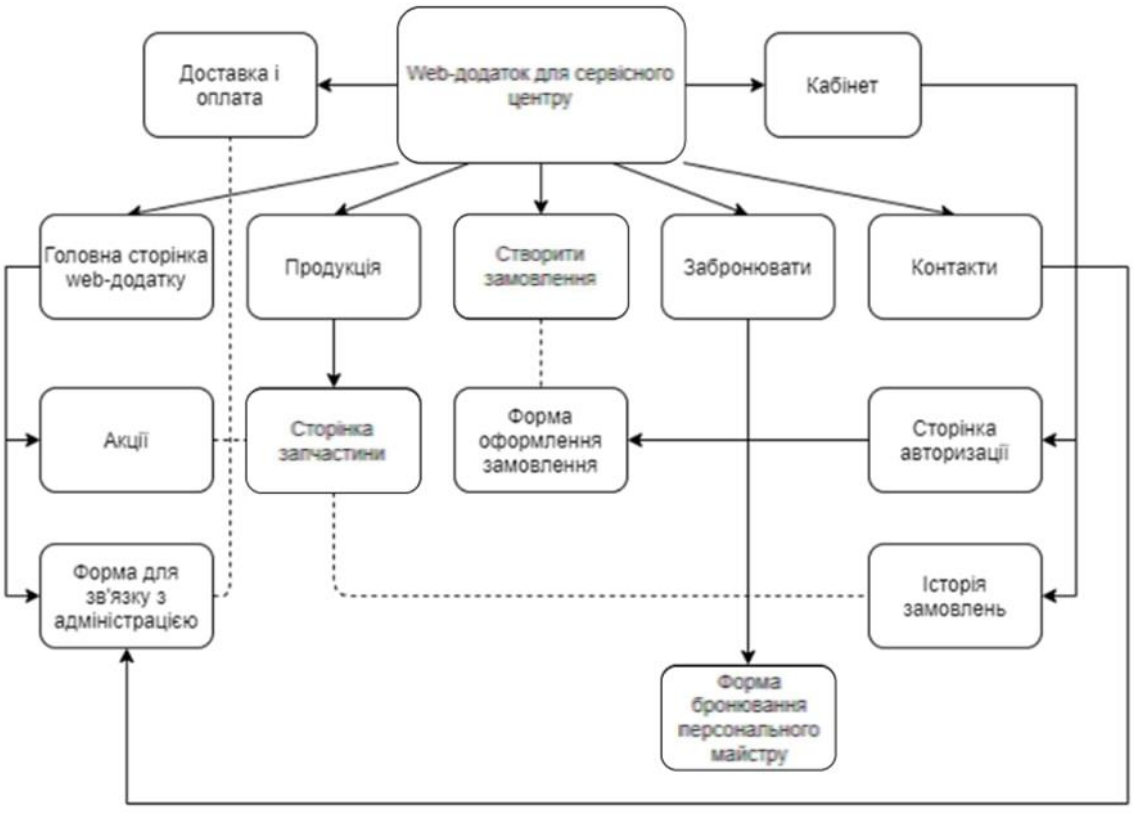


Рисунок А.2 – Система навігації

2.3 Вимоги до функціонування системи

2.3.1 Потреби користувача

Потреби користувача, визначені на основі рішення замовника, представлені у таблиці А.1.

Таблиця А.1 – Потреби користувача

ID	**Потреби користувача**	**Джерело**
UN-01	Перегляд продукції чи послуг сервісного центру	Клієнт, Відвідувач
UN-02	Замовлення продукції онлайн	Клієнт, Відвідувач
UN-03	Перегляд історії своїх замовлень	Клієнт
UN-04	Перегляд опису запчастин	Клієнт, Відвідувач
UN-05	Перегляд акцій та оголошень	Клієнт, Відвідувач
UN-06	Перегляд інформації про сервісний центр	Клієнт, Відвідувач
UN-07	Бронювання персонального майстра онлайн	Клієнт, Відвідувач
UN-08	Можливість створення замовлення послуг чи запчастин	Клієнт, Відвідувач
UN-09	Редагування даних	Адміністратор
UN-10	Автоматизована побудова графіку роботи працівників	Адміністратор

2.3.2 Системні вимоги

Серверна частина (Backend)

Процесор (CPU): Двоядерний процесор з частотою 2.4 GHz або вище.

Оперативна пам'ять (RAM): 4 GB або більше.

Дисковий простір (HDD/SSD): 20 GB вільного місця для зберігання додатку та даних.

Операційна система (OS): Linux (Ubuntu, CentOS) або Windows Server.

Мережеве підключення: Інтернет-з'єднання зі швидкістю не менше 10 Mbps.

Клієнтська частина (Frontend)

Браузер: Останні версії Chrome, Firefox, Safari або Edge.

Процесор (CPU): Двоядерний процесор з частотою 1.8 GHz або вище.

Оперативна пам'ять (RAM): 2 GB або більше.

Дисковий простір (HDD/SSD): 500 MB для тимчасових файлів і кешу браузера.

Роздільна здатність екрану: Мінімум 1280x720.

База даних

Процесор (CPU): Двоядерний процесор з частотою 2.4 GHz або вище. **Оперативна пам'ять (RAM)**: 4 GB або більше (може збільшуватись залежно від обсягу даних).

Дисковий простір (HDD/SSD): 20 GB або більше (залежно від обсягу даних).

Операційна система (OS): Linux (рекомендовано) або Windows Server. **Система керування базами даних (DBMS)**: MySQL, PostgreSQL або MongoDB (в залежності від архітектури додатку).

Додаткові вимоги

Розробницькі інструменти:

- Node.js (LTS версія) для запуску та розробки React-додатків.

- npm або Yarn для керування залежностями.

Середовище виконання:

- Web сервер, такий як Nginx або Apache, для розгортання додатку.
- Контейнеризація (опціонально) за допомогою Docker для легшого розгортання та масштабування.

Безпека:

- SSL сертифікат для забезпечення захищеного з'єднання.
- Регулярні резервні копії бази даних.

Рекомендації

1. **Масштабування**: Для великих обсягів даних або високої відвідуваності можна розглянути використання кластеризації або балансування навантаження.
2. **Моніторинг**: Використання інструментів моніторингу (наприклад, Prometheus, Grafana) для відстеження стану серверів та додатку.
3. **Резервування**: Налаштування резервного копіювання для збереження даних та швидкого відновлення у разі збою.

3 Склад і зміст робіт зі створення вебдодатку сервісного центру

Детальний опис етапів створення вебдодатку наведено в таблиці А.2.

Таблиця А.2 – Етапи створення вебдодатку

№	Склад і зміст робіт	Строк розробки
1	Розробка шаблону вебдодатку	2 дні
2	Задання верстки сторінок вебдодатку	14 днів
3	Розробка модулю резервування майстрів	7 днів
4	Розробка модулю оформлення замовлень	7 днів
5	Розробка модулю графіку роботи працівників	7 днів

Продовження табл. А.2

6	Розробка бази даних	8 днів
7	Наповнення контентом вебдодатку	14 днів
8	Beta-тестування	8 днів
9	Alpha-тестування	5 днів
10	Розміщення на хостингу	1 день
11	Перевірка працездатності	2 дні
12	Написання супровідної документації	2 дні
13	Реліз вебдодатку	1 день
	Загальна тривалість робіт	78 днів

4 Вимоги до складу й змісту робіт із введення вебдодатку в експлуатацію

Вебдодаток має бути затверджено та розміщено на вебхостингу.

ДОДАТОК Б
ПЛАНУВАННЯ РОБІТ

У сучасному світі зростає значення електронних сервісів, які відіграють важливу роль у зручності та доступності надання послуг. В рамках дипломного проекту розглядається розробка вебдодатку сервісного центру, спрямованого на забезпечення зручного та ефективного замовлення та отримання послуг з ремонту та обслуговування техніки.

Цінність розробки виражається у наступних аспектах:

- Покращення досвіду клієнтів:
  - Зручний пошук необхідних послуг за допомогою фільтрів та сортування
  - Детальна інформація про послуги, включаючи вартість, терміни та умови надання
  - Можливість запису на прийом в один клік
- Збільшення продажів:
  - Розширення аудиторії потенційних клієнтів
  - Покращення лояльності клієнтів
  - Збільшення обсягів продажів та прибутку

Застосування вебдодатку сервісного центру має велику актуальність в умовах розвитку технологій та змін споживацьких практик.

Особливості сучасного споживача:

- Звик до швидкого та зручного сервісу
- Оцінює якість інформації та можливість порівняння послуг
- Віддає перевагу онлайн-замовленням, оскільки це зручніше та дешевше

Упровадження вебдодатку сервісного центру дозволить:

- Задовольнити потреби сучасного споживача
- Підвищити конкурентоспроможність компанії на ринку

- Розширити бізнес та збільшити прибуток.

Додаткові можливості вебдодатку:

- Онлайн-консультація: клієнти можуть отримати допомогу від консультанта в режимі реального часу
- Оплата онлайн: клієнти можуть оплатити послуги онлайн
- Моніторинг статусу замовлення: клієнти можуть відстежувати статус свого замовлення в будь-який час
- Зворотний зв'язок: клієнти можуть залишити відгук про якість послуг

Деталізація мети проєкту методом SMART

Для успішності та конкурентоспроможності проекту треба на концептуальному етапі правильно визначити його мету за допомогою SMART-методу. Вона має ширше формулювання. А саме: «Розробка вебдодатку для сервісного центру на основі затвердженого технічного завдання для збільшення прибутковості для сервісних центрів завдяки створенню зручного вебдодатку, який надає клієнтам змогу замовляти ремонт та деталі онлайн до 1 червня 2024 року».

Результати деталізації мети методом SMART розміщені у таблиці Б.1.

Таблиця Б.1 – Деталізація мети проєкту методом SMART

Specific(конкретна)	Розробити та впровадити вебдодаток сервісного центру для ремонту та обслуговування техніки під назвою «IT-сервіс».
Measurable (вимірювана)	Вебдодаток повинен мати такі функціональні можливості: • Зручний каталог послуг з ремонту та обслуговування комп'ютерної техніки.

	• Онлайн-форму для запису на прийом. • Онлайн-консультацію з технічними фахівцями. • Оплату послуг онлайн. • Моніторинг статусу замовлення. • Відгуки клієнтів.
Achievable (досяжна, узгоджена)	Мета досяжна, є затверджене технічне завдання від замовника та клієнтська база сервісного центру
Relevant (реалістична)	Вебдодаток повинен відповідати потребам клієнтів та бізнесу.
Time-framed (обмежена в часі)	Вебдодаток повинен бути розроблений та впроваджений до 1 червня 2024 року.

Планування змісту робіт. WBS (Work Breakdown Structure – ієрархічна структура робіт) – це графічний вигляд елементів проекту, які згруповані ієрархією у єдине ціле з продуктом проекту. Структура декомпозиції робіт орієнтована на досконале виконання робіт по частинам і сама є ключовою частиною проекту, яка спрямована на організацію командної роботи. Елементами декомпозиції можуть бути продукти, дані та послуги. Більше того, WBS забезпечує необхідним каркасом для ретельної оцінки термінів та контролю та графіків роботи.

На найвищому (першому) рівні розміщений продукт проекту. Основні дії та заходи, що забезпечують досягнення мети проекту, зафіксовані на другому рівні декомпозиції. Декомпозиція робіт виконується до тих пір, поки вони не стануть елементарними (простими).

Елементарні роботи – це дії, які мають однозначний чіткий результат, на які призначена відповідальному одна конкретна особа, для якої можна

обчислити витрати праці і тривалість виконання. На рисунку Б.1 представлено WBS з розробки вебдодатку.

Планування структури виконавців. Наступним етапом після декомпозиції процесів є розробка організаційної структури виконавців або OBS, яка визначається як графічна структура відображення учасників або відповідальних осіб, які беруть участь у реалізації проєкту.

У ролі відповідальних осіб виступають співробітники, що відповідають за організацію і виконання елементарної роботи, що зазначена у WBS. Кожну елементарну роботи можна розглядати як окремий проект.

На рисунку Б.2 представлено організаційну структуру планування проєкту. Список виконавців, що функціонують в проекті описано в таблиці Б.2.

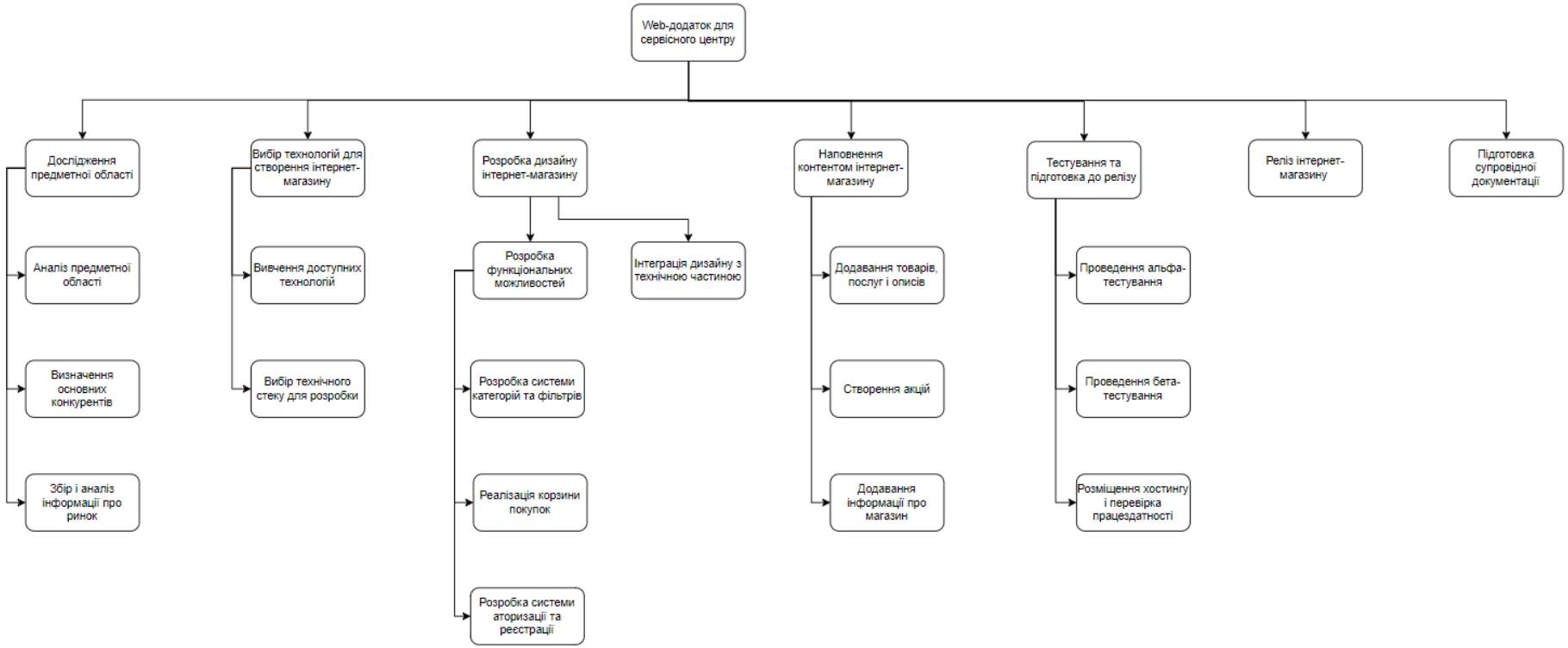


Рисунок Б.1 – WBS-структура робіт проекту

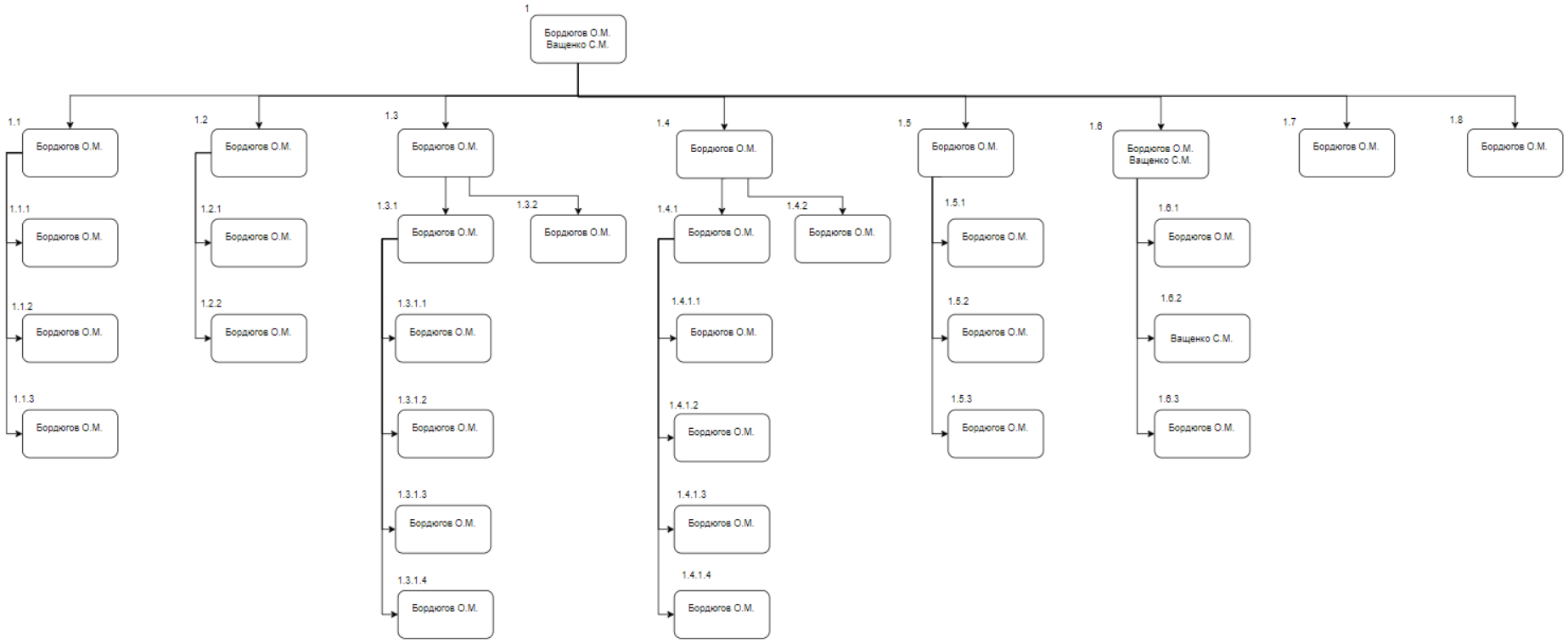


Рисунок Б.2 – OBS-структура робіт проекту

Таблиця Б.2 – Виконавці проєкту

Роль	ПІБ	Проектна роль
Розробник	Бордюгов О.М.	Виконує front-end та back-end розробку.
Проєктувальник	Бордюгов О.М.	Виконує проєктування бази даних та розробляє структуру вебдодатку.
Керівник проекту	Ващенко С.М.	Формує завдання на розробку проєкту.
Менеджер проєкту	Бордюгов О.М.	Відповідає за виконання термінів, розподіл ресурсів та завдань між учасниками. Виконує збір та аналіз даних.
Тестувальник	Бордюгов О.М.	Відповідає за тестування функціоналу та дизайну вебдодатку

Діаграма Ганта. Побудова календарного графіку (діаграми Ганта) є одним з важливих етапів планування проєкту, що виглядає як розклад виконання робіт з реальним розподілом дат. Завдяки йому можна отримати достовірне уявлення про тривалість процесів з обмеженнями у ресурсах, урахуванням вихідних днів та свят. Календарний графік проєкту представлено на рисунках Б.3-Б.4.

WBS	Name	Duration	Start	Finish	Predecessors	Resource Names
1	**Web-додаток для сервісного центру обслуговування техніки**	**192 days**	**Tue 17.10.23**	**Wed 10.07.24**		**Developer;QA**
1.1	**Дослідження предм**	**20 days**	**Tue 17.10.23**	**Mon 13.11.23**		
1.1.1	Аналіз предметної с	10 days	Tue 17.10.23	Mon 30.10.23		Developer
1.1.2	Збір і аналіз інформ	5 days	Tue 31.10.23	Mon 06.11.23	3	Developer
1.1.3	Визначення основни	5 days	Tue 07.11.23	Mon 13.11.23	4	Developer
1.2	**Вибір технологій для створення інтернет-магазину**	**30 days**	**Tue 14.11.23**	**Mon 25.12.23**	**2**	
1.2.1	Вивчення доступни	15 days	Tue 14.11.23	Mon 04.12.23	5	Developer
1.2.2	Вибір технічного сте	15 days	Tue 05.12.23	Mon 25.12.23	7	Developer
1.3	**Розробка дизайну інтернет-магазину**	**37 days**	**Tue 26.12.23**	**Wed 14.02.24**	**6**	
1.3.1	**Розробка графічного**	**27 days**	**Tue 26.12.23**	**Wed 31.01.24**	**8**	
1.3.1.1	Створення дизайн	5 days	Tue 26.12.23	Mon 01.01.24	8	Developer
1.3.1.2	Розробка дизайну	5 days	Thu 11.01.24	Wed 17.01.24	11	Developer
1.3.1.3	Створення дизайн	5 days	Thu 18.01.24	Wed 24.01.24	12	Developer
1.3.1.4	Розробка дизайну	5 days	Thu 25.01.24	Wed 31.01.24	13	Developer
1.3.2	Інтеграція дизайну з	10 days	Thu 01.02.24	Wed 14.02.24	10	Developer
1.4	**Розробка функціональності інтернет-магазину**	**60 days**	**Thu 15.02.24**	**Wed 08.05.24**	**9**	
1.4.1	**Розробка функціональних можливостей**	**45 days**	**Thu 15.02.24**	**Wed 17.04.24**	**15**	
1.4.1.1	Розробка системи	15 days	Thu 15.02.24	Wed 06.03.24	15	Developer
1.4.1.2	Реалізація корзин	10 days	Thu 07.03.24	Wed 20.03.24	18	Developer
1.4.1.3	Розробка системи	10 days	Thu 21.03.24	Wed 03.04.24	19	Developer

Рисунок Б.3 – Календарний графік проекту

1.3.1.3	Створення дизайн	5 days	Thu 18.01.24	Wed 24.01.24	12	Developer
1.3.1.4	Розробка дизайну	5 days	Thu 25.01.24	Wed 31.01.24	13	Developer
1.3.2	Інтеграція дизайну з	10 days	Thu 01.02.24	Wed 14.02.24	10	Developer
1.4	**Розробка функціональності інтернет-магазину**	**60 days**	**Thu 15.02.24**	**Wed 08.05.24**	**9**	
1.4.1	**Розробка функціональних можливостей**	**45 days**	**Thu 15.02.24**	**Wed 17.04.24**	**15**	
1.4.1.1	Розробка системи	15 days	Thu 15.02.24	Wed 06.03.24	15	Developer
1.4.1.2	Реалізація корзин	10 days	Thu 07.03.24	Wed 20.03.24	18	Developer
1.4.1.3	Розробка системи	10 days	Thu 21.03.24	Wed 03.04.24	19	Developer
1.4.1.4	Реалізація перегл	10 days	Thu 04.04.24	Wed 17.04.24	20	Developer
1.4.2	Тестування та випр	15 days	Thu 18.04.24	Wed 08.05.24	17	QA
1.5	**Наповнення контентом інтернет-магазину**	**20 days**	**Thu 09.05.24**	**Wed 05.06.24**	**16**	
1.5.1	Додавання товарів і	10 days	Thu 09.05.24	Wed 22.05.24	21	Developer
1.5.2	Створення акцій та	5 days	Thu 23.05.24	Wed 29.05.24	24	Developer
1.5.3	Додавання інформа	5 days	Thu 30.05.24	Wed 05.06.24	25	Developer
1.6	**Тестування та підготовка до**	**23 days**	**Thu 06.06.24**	**Mon 08.07.24**	**23**	
1.6.1	Проведення бета-те	8 days	Thu 06.06.24	Mon 17.06.24	26	QA
1.6.2	Проведення альфа-	5 days	Fri 28.06.24	Thu 04.07.24	28	QA
1.6.3	Розміщення на хост	2 days	Fri 05.07.24	Mon 08.07.24	29	Developer
1.7	Підготовка супровідної	1 day	Tue 09.07.24	Tue 09.07.24	27	Developer
1.8	Реліз інтернет-магазину	1 day	Wed 10.07.24	Wed 10.07.24	31	Developer

Рисунок Б.4 – Продовження календарного графіку проєкту

Управління ризиками проєкту. Під час виконання якісної оцінки ризиків треба визначити ризики, які мають бути усунені якнайшвидше. В залежності від ступеня важливості ризику – реагування буде відповідне. Наступним етапом є виконання кількісного оцінювання ризиків. Кількісне та якісне оцінювання можуть виконувати одночасно або окремо, що залежить від ступеня забезпечення проєкту. У таблиці Б.3 надано перелік ризиків даного проєкту. Результати оцінки ризиків надано у таблиці Б.4.

Таблиця Б.3 – Ризики проекту

№ ризику	Назва (опис) ризику
1	Недостатня кваліфікація технічного персоналу
2	Зміна вимог клієнтів або бізнесу
3	Нестача фінансування
4	Нестача ресурсів (час, обладнання, матеріали)
5	Непередбачені обставини (природні катаклізми, політичні заворушення)
6	Технологічний прогрес
7	Конкурентне середовище
8	Помилки в проектуванні
9	Помилки в кодуванні
10	Неякісне тестування

Таблиця Б.4 – Результати визначення ймовірності, впливу та рангу ризиків проекту

№ ризику	Назва (опис) ризику	Ймовірність (0,1-0,9)	Вплив (0,05-0,8)	Ранг
1	Недостатня кваліфікація технічного персоналу	0,6	0,6	0,12
2	Зміна вимог клієнтів або бізнесу	0,7	0,7	0,045
3	Нестача фінансування	0,5	0,5	0,12
4	Нестача ресурсів (час, обладнання, матеріали)	0,6	0,6	0,10
5	Непередбачені обставини (природні катаклізми, політичні заворушення)	0,2	0,8	0,10
6	Технологічний прогрес	0,5	0,4	0,05
7	Конкурентне середовище	0,4	0,5	0,06
8	Помилки в проектуванні	0,5	0,6	0,035
9	Помилки в кодуванні	0,6	0,5	0,03
10	Неякісне тестування	0,5	0,4	0,14

Для того, щоб знизити негативний вплив ризиків на проект треба виконати планування реагування на них. До нього входить оцінка наслідків впливу на проект і розробка відповідних заходів. Аналіз виконується за показниками, які описані в таблиці Б.4. У результаті планування заходів реагування на ризики проєкту було отримано матрицю ймовірності виникнення та впливу ризиків (таблиця Б.5. Зеленим кольором на матриці позначають прийнятні ризики, жовтим – виправдані, а червоним – недопустимі.

Таблиця Б.5 – Матриця ймовірності та впливу згідно проекту

Ймовірність ризику (Й)	Вплив загрози (ризику)				
	Дуже малий	Малий	Середній	Великий	Дуже великий
	0.05	0.1	0.2	0.4	0.8
0.9	0,045 R2	0,09	0,18	0,36 R4	0,72
0.7	0,035 R8	0,07	0,14 R10	0,28	0,56 R3
0.5	0,025	0,05 R6	0,10R5,4	0,20	0,40
0.3	0,015	0,03 R9	0,06 R7	0,12R1,3	0.24
0.1	0,005	0,01	0,02	0,04	0,08

Класифікація ризиків проєкту за рівнем, відповідно до отриманого значення індексу, представлена у таблиці Б.6. У таблиці Б.7 описано ризики та стратегії реагування на кожен із них.

Таблиця Б.6 – Шкала оцінювання ризику за рівнем

№	Назва	Межі	Ризики, які входять (номера)
1	Прийнятні	$0,005 \leq R \leq 0,05$	2,6,8,9
2	Виправдані	$0,05 < R \leq 0,14$	5,1,7,10,4,3
3	Недопустимі	$0,14 < R \leq 0,72$	

Таблиця Б.7 – Ризики проекту та стратегії реагування

ID ризику	Статус ризику	Опис ризику	Ймовірність виникнення	Вплив ризику	Ранг ризику	Тип стратегії реагування	План А (заходи запобігання виникненню ризику)	План Б (заходи усунення наслідків ризику)
1	Виправданий	Недостатня кваліфікація технічного персоналу	0,6	0,6	0,12	ЗМЕНШЕННЯ	Навчання, сертифікація, наставництво	Перепідготовка, залучення кваліфікованих фахівців, зменшення функціональних можливостей
2	Виправданий	Зміна вимог клієнтів або бізнесу	0,7	0,7	0,045	ЗМЕНШЕННЯ	Регулярне спілкування, гнучке технічне завдання, тестування клієнтами	Переробка технічного завдання, перевиконання, зменшення функціональних можливостей
3	Виправданий	Нестача фінансування	0,5	0,5	0,12	ЗМЕНШЕННЯ	Детальний кошторис, додаткові джерела, зменшення масштабу	Звернення до інвесторів, завершення на меншому масштабі

Продовження табл. Б.7

ID ризику	Статус ризику	Опис ризику	Ймовірність виникнення	Вплив ризику	Ранг ризику	Тип стратегії реагування	План А (заходи запобігання виникненню ризику)	План Б (заходи усунення наслідків ризику)
4	Виправданий	Нестача ресурсів (час, обладнання, матеріали)	0,6	0,6	0,10	ЗМЕНШЕННЯ	Запланування, резервування, додаткові ресурси, зміна термінів або вимог	Збільшення термінів, зменшення функціональних можливостей, залучення постачальників
5	Виправданий	Непередбачені обставини (природні катаклізми, політичні заворушення)	0,2	0,8	0,10	ЗМЕНШЕННЯ	Страхування, план дій	Резервні копії, перенесення
6	Виправданий	Непередбачені обставини (природні катаклізми, політичні заворушення)	0,5	0,4	0,05	ЗМЕНШЕННЯ	Регулярне оновлення	Зменшення функціональних можливостей
7	Виправданий	Технологічний прогрес	0,4	0,5	0,06	ЗМЕНШЕННЯ	Регулярне дослідження, впровадження інновацій	Зменшення ціни
8	Виправданий	Конкурентне середовище	0,5	0,6	0,035	ЗМЕНШЕННЯ	Ретельне тестування на всіх етапах розробки	Переробка дизайну
9	Виправданий	Помилки в проектуванні	0,6	0,5	0,03	ЗМЕНШЕННЯ	Залучення кваліфікованих розробників	Навчання розробників
10	Виправданий	Помилки в кодуванні	0,5	0,4	0,14	ЗМЕНШЕННЯ	Залучення кваліфікованих тестувальників	Розробка більш ретельного плану тестування

ДОДАТОК В
Лістинги основних модулів

Лістинг клієнтської частини

Код з папки admin

Файл Logs.js

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';
import { Table, Button, Container } from 'react-bootstrap';

const MessageTable = () => {
  const [messages, setMessages] = useState([]);

  const fetchMessages = async () => {
    try {
      const response = await axios.get('http://localhost:3000/api/admin/messages');
      setMessages(response.data);
    } catch (error) {
      console.error('Ошибка получения сообщений', error);
    }
  };

  useEffect(() => {
    fetchMessages();
  }, []);

  const deleteMessage = async (id) => {
    try {
      await axios.delete(`http://localhost:3000/api/admin/messages/${id}`);
      fetchMessages(); // Refresh the list after deletion
    } catch (error) {
      console.error('Ошибка удаления сообщения', error);
    }
  };

  return (
    <Container>
      <h2>Сообщения</h2>
      <Table striped bordered hover>
        <thead>
          <tr>
            <th>Имя</th>
            <th>Телефон</th>
            <th>Email</th>
            <th>Сообщение</th>
            <th>Действия</th>
```

```
        </tr>
      </thead>
      <tbody>
        {messages.map((message) => (
          <tr key={message.id}>
            <td>{message.name}</td>
            <td>{message.numberPhone}</td>
            <td>{message.email}</td>
            <td>{message.message}</td>
            <td>
              <Button
                variant="danger"
                onClick={() => deleteMessage(message.id)}
              >
                Удалить
              </Button>
            </td>
          </tr>
        ))}
      </tbody>
    </Table>
  </Container>
 );
};

export default MessageTable;
```

Файл ServicesManagement.jsx

```
import React, { useState, useEffect } from "react";
import axios from "axios";
import { Container, Table, Button } from "react-bootstrap";
import { useTranslation } from "react-i18next";

function ServicesManagement() {
 const [servicesSend, setServicesSend] = useState([]);
 const { t } = useTranslation();

 console.log(servicesSend);

 useEffect(() => {
  fetchServicesSendData();
 }, []);

 const fetchServicesSendData = async () => {
  try {
   const response = await axios.get(
    "http://localhost:3000/api/admin/services"
   );
   setServicesSend(response.data);
```

```
  } catch (error) {
    console.error("Error fetching services send data:", error);
  }
};

const deleteServiceSend = async (id) => {
  try {
    await axios.delete(
      `http://localhost:3000/api/admin/servicesDelete/${id}`
    );
    fetchServicesSendData(); // Refresh the list
  } catch (error) {
    console.error("Error deleting service send:", error);
  }
};

const providedServiceSend = async (_id) => {
  try {
    const response = await axios.put(
      "http://localhost:3000/api/admin/services",
      {
        id: _id, // замініть на правильний id запиту на сервіс
      }
    );
    fetchServicesSendData(); // Refresh the list
  } catch (error) {
    console.error("Error deleting service send:", error);
  }
};

return (
  <Container className="my-5">
    <div style={{ background: "#00000090" }} className="mb-5 p-4">
      <h2>{t("services_management")}</h2>
    </div>
    <Table striped bordered hover>
      <thead>
        <tr>
          <th>{t("service_name")}</th>
          <th>{t("category_name")}</th>
          <th>{t("number_phone_client")}</th>
          <th>{t("name_client")}</th>
          <th>{t("provided_service")}</th>
          <th>{t("message")}</th>
          <th>{t("actions")}</th>
        </tr>
      </thead>
      <tbody>
        {servicesSend && servicesSend.length > 0 ? (
          servicesSend.map((serviceSend) => (
            <tr key={serviceSend.id}>
              <td>
```

```
                {serviceSend.Category && serviceSend.Category.Service
                  ? serviceSend.Category.Service.name
                  : t("no_service")}
                </td>
                <td>
                  {serviceSend.Category
                    ? serviceSend.Category.name
                    : t("no_category")}
                </td>
                <td>{serviceSend.numberPhoneClient}</td>
                <td>{serviceSend.nameClient}</td>
                <td>{serviceSend.providedService ? t("yes") : t("no")}</td>
                <td>{serviceSend.message}</td>
                <td>
                  <Button
                    variant="danger"
                    onClick={() => deleteServiceSend(serviceSend.id)}
                  >
                    {t("delete")}
                  </Button>
                </td>
                {serviceSend.providedService ? (
                  ""
                ) : (
                  <div style={{ backgroundColor: "#ffffff00" }}>
                    <Button
                      variant="success"
                      className="bg-green-500"
                      onClick={() => providedServiceSend(serviceSend.id)}
                    >
                      {t("Виконанно")}
                    </Button>
                  </div>
                )}
              </tr>
            ))
          ) : (
            <tr>
              <td colSpan="7">{t("no_services_sends")}</td>
            </tr>
          )}
        </tbody>
      </Table>
    </Container>
  );
}

export default ServicesManagement;
```

Файл SparePartsManagement.js

```
import React, { useState, useEffect } from "react";
import axios from "axios";
import {
  Container,
  Table,
  Button,
  Row,
  Col,
  Card,
  Modal,
  Form,
} from "react-bootstrap";
import { useTranslation } from "react-i18next";

function SparePartsManagement() {
  const [sparePartsRequests, setSparePartsRequests] = useState([]);
  const [showModal, setShowModal] = useState(false);
  const [showModalDelete, setShowModalDelete] = useState(false);
  const [showDeleteModal, setShowDeleteModal] = useState(false);
  const [selectedSparePartsId, setSelectedSparePartsId] = useState(null);
  const [ responseData, setResponseData ] = useState('')
  const [selectedRequestId, setSelectedRequestId] = useState(null);
  const [spareParts, setSpareParts] = useState([]);
  const [resultResponse, setResultResponse] = useState("");
  const [formData, setFormData] = useState({
    number: "",
    description: "",
    photo: "",
    nameSpareParts: "",
    price: "",
  });

  const [formDataChange, setFormDataChange] = useState({
    id:"",
    number: "",
    description: "",
    photo: "",
    nameSpareParts: "",
    price: "",
  });
  const { t } = useTranslation();

  const handleChange = (e) => {
    const { name, value } = e.target;
    setFormData({ ...formData, [name]: value });
  };

  const handleChangeSubmit = (e) => {
    const { name, value } = e.target;
    setFormDataChange({ ...formDataChange, [name]: value });
```

```
};

const handleFileChange = (e) => {
  const file = e.target.files[0];
  const reader = new FileReader();
  reader.onloadend = () => {
    setFormData({ ...formData, photo: reader.result });
  };
  reader.readAsDataURL(file);
};

const handleSubmit = async (e) => {
  e.preventDefault();
  try {
    const response = await axios.post(
      "http://localhost:3000/api/admin/createSpareParts",
      formData
    );
    console.log("Spare part created:", response.data);
    setResultResponse(response.data.message);
    fetchSparePartsRequests(); // Refresh the list after adding a new spare part
    setShowModal(false);
  } catch (error) {
    console.error("Error creating spare part:", error);
  }
};

const handleSubmitChange = async (e) => {
  e.preventDefault();
  try {
    const response = await axios.put(
      `http://localhost:3000/api/admin/updateSpareParts/${formDataChange.id}`,
      formDataChange
    );
    console.log("Spare part created:", response.data);
    setResponseData(response.data.message);
    console.log(responseData)
    fetchSparePartsRequestsAll(); // Refresh the list after adding a new spare part
    // setShowModal(false);
  } catch (error) {
    console.error("Error creating spare part:", error);
  }
};

const fetchSparePartsRequests = async () => {
  try {
    const response = await axios.get(
      "http://localhost:3000/api/admin/userSpareParts"
    );
    setSparePartsRequests(response.data);
  } catch (error) {
    console.error("Error fetching spare parts requests:", error);
```

```
  }
};

const fetchSparePartsRequestsAll = async () => {
  try {
    const response = await axios.get("http://localhost:3000/api/viewing");
    // setSparePartsRequests(response.data);
    setSpareParts(response.data);
    console.log(response.data);
  } catch (error) {
    console.error("Error fetching spare parts requests:", error);
  }
};

useEffect(() => {
  fetchSparePartsRequests();
  fetchSparePartsRequestsAll();
}, []);

const deleteSparePartRequest = async (id) => {
  try {
    await axios.delete(
      `http://localhost:3000/api/admin/userSpareParts/${id}`
    );
    fetchSparePartsRequests(); // Refresh the list
  } catch (error) {
    console.error("Error deleting spare part request:", error);
  }
};

const deleteSparePart = async (id) => {
  try {
    console.log();
    await axios.delete(`http://localhost:3000/api/admin/spareParts/${id}`);
    fetchSparePartsRequestsAll(); // обновление списка запчастей после удаления
  } catch (error) {
    console.error("Error deleting spare part:", error);
  }
};

const providedSpearePart = async (
  id,
  sparePartInStock,
  needDevice,
  spareParts_id
) => {
  try {
    const response = await axios.put(
      "http://localhost:3000/api/admin/sparePartsProvided",
      {
        id, // замініть на правильний id запиту на сервіс
        sparePartInStock,
```

```
        needDevice,
        spareParts_id,
      }
    );
    fetchSparePartsRequests(); // Refresh the list
  } catch (error) {
    console.error("Error updating spare part request:", error);
  }
};

return (
  <>
    <Container className="my-5">
      <h2>{t("spare_parts_requests_management")}</h2>
      <Table striped bordered hover>
        <thead>
          <tr>
            <th>{t("client_name")}</th>
            <th>{t("phone")}</th>
            <th>{t("message")}</th>
            <th>{t("spare_part_name")}</th>
            <th>{t("spare_part_description")}</th>
            <th>{t("price")}</th>
            <th>{t("quantity")}</th>
            <th>{t("quantity_left")}</th>
            <th>{t("provided_service")}</th>
            <th>{t("actions")}</th>
          </tr>
        </thead>
        <tbody>
          {spareRequests.map((request) => (
            <>
              <tr key={request.id}>
                <td>{request.nameClient}</td>
                <td>{request.numberPhoneClient}</td>
                <td>{request.message}</td>
                <td>{request.SparePart.nameSpareParts}</td>
                <td>{request.SparePart.description}</td>
                <td>{request.SparePart.price}</td>
                <td>{request.number}</td>
                <td>{request.SparePart.number}</td>
                <td>{request.providedService ? t("yes") : t("no")}</td>
                <td>
                  <Button
                    variant="danger"
                    type="button"
                    class="btn btn-primary mb-4 mt-4"
                    data-bs-toggle="modal"
                    data-bs-target="#DeleteModalUsersSparePats"
                    onClick={() => {
                      setSelectedRequestId(request.id);
                      // setShowDeleteModal(true);
```

```
      }}
    >
      {t("delete")}
    </Button>
  </td>
  {request.providedService ? (
    ""
  ) : (
    <td style={{ backgroundColor: "#00000000" }}>
      <Button
        variant="success"
        className="bg-green-500"
        // style={{backgroundColor: "#000000"}}
        onClick={() =>
          providedSpearePart(
            request.id,
            request.SparePart.number,
            request.number,
            request.SparePart.id
          )
        }
      >
        {t("Виконанно")}
      </Button>
    </td>
    ////////////////////////////////////
  )}
</tr>

<div
  class="modal fade"
  id="DeleteModalUsersSparePats"
  tabindex="-1"
  aria-labelledby="exampleModalLabel"
  aria-hidden="true"
>
  <div
    class="modal-dialog modal-dialog-centered modal-sm"
    style={{ background: "#ffffff00" }}
  >
    <div class="modal-content p-4">
      <div class="modal-header">
        <h1 class="modal-title fs-4" id="DeleteModalLabel">
          {t("Підтвердження")}
        </h1>
        <button
          type="button"
          class="btn-close"
          data-bs-dismiss="modal"
          aria-label="Close"
        ></button>
      </div>
```

```
            <br />
            <div>Ви підтверджуете видалення?</div>
            <button
              type="button"
              class="btn btn-primary mt-4"
              data-bs-dismiss="modal"
              onClick={() =>
                deleteSparePartRequest(selectedRequestId)
              }
            >
              Підтвердити
            </button>
          </div>
        </div>
      </div>
    </>
  ))}
 </tbody>
</Table>
</Container>

<Container>
 <Row>
  <Col>
   <Card className="mb-4">
    <Card.Body>
     <Button
      variant="primary"
      type="button"
      className="btn btn-primary"
      data-bs-toggle="modal"
      onClick={() => setShowModal(true)}
     >
      <Card.Text>Додати запчастини</Card.Text>
      <Card.Title>До катологу</Card.Title>
      <Card.Text>
       <strong>{t("price")}:</strong>
      </Card.Text>
     </Button>
    </Card.Body>
   </Card>
  </Col>
 </Row>

 <Modal show={showModal} onHide={() => setShowModal(false)} centered>
  <Modal.Header closeButton>
   <Modal.Title>{t("add_spare_part")}</Modal.Title>
  </Modal.Header>
  <Modal.Body>
   <Form onSubmit={handleSubmit}>
    <Form.Group className="mb-3">
     <Form.Label>{t("number")}</Form.Label>
```

```
        <Form.Control
          type="text"
          name="number"
          value={formData.number}
          onChange={handleChange}
        />
      </Form.Group>
      <Form.Group className="mb-3">
        <Form.Label>{t("description")}</Form.Label>
        <Form.Control
          type="text"
          name="description"
          value={formData.description}
          onChange={handleChange}
        />
      </Form.Group>
      <Form.Group className="mb-3">
        <Form.Label>{t("photo")}</Form.Label>
        <Form.Control
          type="file"
          name="photo"
          onChange={handleFileChange}
        />
      </Form.Group>
      <Form.Group className="mb-3">
        <Form.Label>{t("name_spare_parts")}</Form.Label>
        <Form.Control
          type="text"
          name="nameSpareParts"
          value={formData.nameSpareParts}
          onChange={handleChange}
        />
      </Form.Group>
      <Form.Group className="mb-3">
        <Form.Label>{t("price")}</Form.Label>
        <Form.Control
          type="text"
          name="price"
          value={formData.price}
          onChange={handleChange}
        />
      </Form.Group>
      <Button variant="primary" type="submit">
        {t("send_message")}
      </Button>
    </Form>
  </Modal.Body>
</Modal>

<Modal
  show={!!resultResponse}
  onHide={() => setResultResponse("")}
```

```
    centered
  >
    <Modal.Header closeButton>
      <Modal.Title>Відповідь</Modal.Title>
    </Modal.Header>
    <Modal.Body>{resultResponse}</Modal.Body>
    <Modal.Footer>
      <Button variant="primary" onClick={() => setResultResponse("")}>
        {t("close_modal")}
      </Button>
    </Modal.Footer>
  </Modal>
</Container>

<Container>
  <Table striped bordered hover className="mt-3">
    <thead>
      <tr>
        <th>{t("count")}</th>
        <th>{t("description")}</th>
        <th>{t("photo")}</th>
        <th>{t("name_spare_parts")}</th>
        <th>{t("price")}</th>
        <th>{t("actions")}</th>
      </tr>
    </thead>
    <tbody>
      {spareParts.map((part) => (
        <>
          <tr key={part.id}>
            <td>{part.number}</td>
            <td>{part.description}</td>
            <td>
              {part.photo && (
                <img
                  src={part.photo}
                  alt={part.nameSpareParts}
                  width="50"
                />
              )}
            </td>
            <td>{part.nameSpareParts}</td>
            <td>{part.price}</td>
            <td>
              <Button
                variant="danger"
                type="button"
                class="btn btn-primary mb-4 mt-4"
                data-bs-toggle="modal"
                data-bs-target="#DeleteModal"
                onClick={(e) => setSelectedSparePartsId(part.id)}
              >
```

```
      {t("delete")}
    </Button>
  </td>
  <td style={{ backgroundColor: "#00000000" }}>
    <Button
      variant="danger"
      type="button"
      class="btn btn-primary mb-4 mt-4"
      data-bs-toggle="modal"
      data-bs-target="#editModal"
      onClick={() => {
        setFormDataChange(part);
        // setShowDeleteModal(true);
      }}
    >
      {t("Змінити")}
    </Button>
  </td>
</tr>

<div
  class="modal fade"
  id="DeleteModal"
  tabindex="-1"
  aria-labelledby="exampleModalLabel"
  aria-hidden="true"
>
  <div
    class="modal-dialog modal-dialog-centered modal-sm"
    style={{ background: "#ffffff00" }}
  >
    <div class="modal-content p-4">
      <div class="modal-header">
        <h1 class="modal-title fs-4" id="DeleteModalLabel">
          {t("Підтвердження")}
        </h1>
        <button
          type="button"
          class="btn-close"
          data-bs-dismiss="modal"
          aria-label="Close"
        ></button>
      </div>
      <br />
      <div>Ви підтверджуете видалення?</div>
      <button
        type="button"
        class="btn btn-primary mt-4"
        data-bs-dismiss="modal"
        onClick={() => deleteSparePart(selectedSparePartsId)}
      >
        Підтвердити
```

```
            </button>
          </div>
        </div>
      </div>
    </>
  ))}
 </tbody>
</Table>
<div
 className="modal fade"
 id="editModal"
 tabIndex="-1"
 aria-labelledby="exampleModalLabel"
 aria-hidden="true"
>
 <div className="modal-dialog modal-dialog-centered">
  <div className="modal-content">
   <div className="modal-header">
    <h1 className="modal-title fs-4" id="exampleModalLabel">
     {t("new_mess")}
    </h1>
    <br />
    <button
     type="button"
     className="btn-close"
     data-bs-dismiss="modal"
     aria-label="Close"
    ></button>
   </div>
   <div className="modal-body row">
    <form onSubmit={handleSubmitChange}>
     <h4 className="fs-5">Змінити</h4>
     <div className="mb-3 row">
      <label htmlFor="number" className="col-md-4">
       {t("number")}:
      </label>
      <input
       type="text"
       name="number"
       value={formDataChange.number}
       onChange={handleChangeSubmit}
       className="form-control"
       id="number"
      />
     </div>
     <div className="mb-3 row">
      <label htmlFor="description" className="col-md-4">
       {t("description")}:
      </label>
      <input
       type="text"
       name="description"
```

```
    value={formDataChange.description}
    onChange={handleChangeSubmit}
    className="form-control"
    id="description"
   />
  </div>
  <div className="mb-3 row">
   <label htmlFor="photo" className="col-md-4">
    {t("photo")}:
   </label>
   <input
    type="file"
    name="photo"
    onChange={handleChangeSubmit}
    className="form-control"
    id="photo"
   />
  </div>
  <div className="mb-3 row">
   <label htmlFor="nameSpareParts" className="col-md-4">
    {t("name_spare_parts")}:
   </label>
   <input
    type="text"
    name="nameSpareParts"
    value={formDataChange.nameSpareParts}
    onChange={handleChangeSubmit}
    className="form-control"
    id="nameSpareParts"
   />
  </div>
  <div className="mb-3 row">
   <label htmlFor="price" className="col-md-4">
    {t("price")}:
   </label>
   <input
    type="text"
    name="price"
    value={formDataChange.price}
    onChange={handleChangeSubmit}
    className="form-control"
    id="price"
   />
  </div>
  <div className="modal-footer">
   <Button
    type="button"
    className="btn btn-secondary"
    data-bs-dismiss="modal"
   >
    {t("close_modal")}
   </Button>
```

```
                <button
                  type="button"
                  data-bs-target="#exampleModalToggle2"
                  className="btn btn-primary"
                  data-bs-toggle="modal"
                  onClick={handleSubmitChange}
                >
                  {t("send_message")}
                </button>
              </div>
            </form>
          </div>
        </div>
      </div>
      <div
      class="modal fade"
      id="exampleModalToggle2"
      aria-hidden="true"
      aria-labelledby="exampleModalToggleLabel2"
      tabindex="-1"
    >
      <div class="modal-dialog modal-dialog-centered modal-sm">
        <div class="modal-content">
          <div class="modal-header">
            <h1 class="modal-title fs-5" id="exampleModalToggleLabel2">
              Відповідь
            </h1>
            <button
              type="button"
              class="btn-close"
              data-bs-dismiss="modal"
              aria-label="Close"
            ></button>
          </div>
          <div class="modal-body">{responseData}</div>
          <div class="modal-footer">
            <button
              type="button"
              class="btn btn-primary"
              data-bs-dismiss="modal"
            >
              {t("close_modal")}
            </button>
          </div>
        </div>
      </div>
    </div>
    </Container>
  </>
 );
}
```

```
export default SparePartsManagement;
```

Файл admin.jsx

```
import React from "react";
import { Outlet } from "react-router-dom";
import "../../styles/admin.css";

function Login() {
  return <Outlet />;
}

export default Login;
```

Файл administrationPage.jsx

```
import React from "react";
import { Link } from "react-router-dom";
import { Container, Row, Col, Card } from "react-bootstrap";
import { useTranslation } from "react-i18next";

function AdminDashboard() {
  const { t } = useTranslation();

  return (
    <Container className="my-5">
      <Row className="justify-content-md-center">
        <Col md={4}>
          <Card className="mb-4">
            <Card.Body>
              <Card.Title>{t("services_management")}</Card.Title>
              <Card.Text>{t("manage_services_description")}</Card.Text>
              <Link to="/admin/services" className="btn btn-primary">
                {t("manage_services")}
              </Link>
            </Card.Body>
          </Card>
        </Col>
        <Col md={4}>
          <Card className="mb-4">
            <Card.Body>
              <Card.Title>{t("spare_parts_requests_management")}</Card.Title>
              <Card.Text>
                {t("manage_spare_parts_requests_description")}
              </Card.Text>
              <Link to="/admin/spareParts" className="btn btn-primary">
                {t("manage_spare_parts_requests")}
              </Link>
            </Card.Body>
          </Card>
```

```
        </Col>
        <Col md={4}>
          <Card className="mb-4">
            <Card.Body>
              <Card.Title>{t("logs")}</Card.Title>
              <Card.Text>{t("view_logs_description")}</Card.Text>
              <Link to="/admin/message" className="btn btn-primary">
                {t("view_logs")}
              </Link>
            </Card.Body>
          </Card>
        </Col>
      </Row>
    </Container>
  );
}

export default AdminDashboard;
```

Файл auth.jsx

```
import React, { useState } from "react";
import { useNavigate } from "react-router-dom";
import axios from "axios";
import {
  Container,
  Row,
  Col,
  Form,
  Button,
  Alert,
  Card,
} from "react-bootstrap";
import { useTranslation } from "react-i18next";
import "../../styles/admin.css";

function Login() {
  const { t } = useTranslation();
  const navigate = useNavigate();
  const [formData, setFormData] = useState({ login: "", password: "" });
  const [error, setError] = useState("");

  const handleChange = (e) => {
    const { name, value } = e.target;
    setFormData({ ...formData, [name]: value });
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      const response = await axios.post(
```

```
      "http://localhost:3000/api/admin/auth",
      formData
    );
    console.log(response.data);
    //   localStorage.setItem('token', response.data.token);
    navigate("/admin/administrationPage"); // Перенаправлення на сторінку після успішного входу
  } catch (error) {
    setError(t("invalid_credentials"));
  }
};

return (
  <div className="login-background">
    <Container className="my-5">
      <Row className="justify-content-md-center">
        <Col md={6}>
          <Card className="login-card p-4">
            <h1 className="text-center">{t("login")}</h1>
            {error && <Alert variant="danger">{error}</Alert>}
            <Form onSubmit={handleSubmit}>
              <Form.Group controlId="formUsername">
                <Form.Label style={{ color: "white" }}>
                  {t("username")}
                </Form.Label>
                <Form.Control
                  type="text"
                  name="login"
                  value={formData.login}
                  onChange={handleChange}
                  placeholder={t("enter_username")}
                  required
                />
              </Form.Group>
              <Form.Group controlId="formPassword" className="mt-3">
                <Form.Label style={{ color: "white" }}>
                  {t("password")}
                </Form.Label>
                <Form.Control
                  type="password"
                  name="password"
                  value={formData.password}
                  onChange={handleChange}
                  placeholder={t("enter_password")}
                  required
                />
              </Form.Group>
              <Button variant="primary" type="submit" className="mt-3 w-100">
                {t("login")}
              </Button>
            </Form>
          </Card>
        </Col>
```

```
      </Row>
    </Container>
  </div>
 );
}

export default Login;
```

Код з папки categories

Файл categories.js

```
import React, { useState, useEffect } from "react";
import { Container, Row, Col, Card, Button } from "react-bootstrap";
import { useParams } from "react-router";
import axios from "axios";
import { useTranslation } from "react-i18next";

function Categories() {
 const { serviceId } = useParams();
 const [categories, setCategories] = useState([]);
 const [responseData, setResponseData] = useState("");
 const [products, setProducts] = useState({
  typeServicesId: "",
  categoriesServicesId: "",
  numberPhoneClient: "",
  nameClient: "",
  message: "",
 });

 const { t } = useTranslation();

 const fetchCategories = async (serviceId) => {
  try {
   const response = await axios.get(
    `http://localhost:3000/api/services/categories/${serviceId}`
   );
   setCategories(response.data);
  } catch (error) {
   console.error("Error fetching categories", error);
  }
 };

 useEffect(() => {
  fetchCategories(serviceId);
 }, [serviceId]);

 const handleSubmit = async (e) => {
  e.preventDefault();
  try {
   const response = await axios.post(
    "http://localhost:3000/api/servicesCreate/Send",
```

```
      products
    );
    setResponseData(response.data);
  } catch (error) {
    console.error("Error submitting form", error);
  }
};

const handleGetProduct = async (service_id, categories_id) => {
  setProducts({
    ...products,
    typeServicesId: service_id,
    categoriesServicesId: categories_id,
  });
};

return (
  <Container className="my-5">
    <Row>
      {categories.map((categorie) => (
        <Col key={categorie.id} md={4} className="mb-4">
          <Button
            variant="primary"
            type="button"
            class="btn btn-primary mb-4 mt-4"
            data-bs-toggle="modal"
            data-bs-target="#exampleModal"
            onClick={() =>
              handleGetProduct(categorie.service_id, categorie.id)
            }
          >
            <Card className="mb-4">
              <Card.Body>
                <Card.Title>{categorie.name}</Card.Title>
                Детальніше
              </Card.Body>
            </Card>
          </Button>
        </Col>
      ))}
    </Row>

    <div
      class="modal fade"
      id="exampleModal"
      tabindex="-1"
      aria-labelledby="exampleModalLabel"
      aria-hidden="true"
    >
      <div class="modal-dialog modal-dialog-centered">
        <div class="modal-content">
          <div class="modal-header">
```

```
  <h1 class="modal-title fs-4" id="exampleModalLabel">
    {t("new_mess")}
  </h1>
  <br />

  <button
    type="button"
    class="btn-close"
    data-bs-dismiss="modal"
    aria-label="Close"
  ></button>
</div>
<div class="modal-body row">
  <h4 class="fs-5">{t("buy_spare_part")}</h4>
  <div class="mb-3 row">
    <label for="recipient-name" class="col-md-4">
      {t("team_member_name")}:
    </label>
    <input
      type="text"
      class="form-control"
      value={products.nameClient}
      name="nameClient"
      onChange={(e) =>
        setProducts({ ...products, nameClient: e.target.value })
      }
      id="recipient-name"
    />
  </div>
  <div class="mb-3 row">
    <label for="recipient-name" class="col-md-3">
      {t("phone1")}:
    </label>
    <input
      type="text"
      class="form-control"
      value={products.numberPhoneClient}
      name="numberPhoneClient"
      onChange={(e) =>
        setProducts({
          ...products,
          numberPhoneClient: e.target.value,
        })
      }
      id="recipient-name"
    />
  </div>
  <div class="mb-3 row">
    <label for="message-text" class="col-md-4">
      {t("mess")}:
    </label>
    <textarea
```

```
            class="form-control"
            value={products.message}
            name="message"
            onChange={(e) =>
              setProducts({ ...products, message: e.target.value })
            }
            id="message-text"
          ></textarea>
        </div>
      </div>
      <div class="modal-footer">
        <button
          type="button"
          class="btn btn-secondary"
          data-bs-dismiss="modal"
        >
          {t("close_modal")}
        </button>
        <button
          type="button"
          data-bs-target="#exampleModalToggle2"
          class="btn btn-primary"
          data-bs-toggle="modal"
          onClick={(e) => handleSubmit(e)}
        >
          {t("send_message")}
        </button>
      </div>
    </div>
  </div>
</div>
<div
  class="modal fade"
  id="exampleModalToggle2"
  aria-hidden="true"
  aria-labelledby="exampleModalToggleLabel2"
  tabindex="-1"
>
  <div class="modal-dialog modal-dialog-centered modal-sm">
    <div class="modal-content">
      <div class="modal-header">
        <h1 class="modal-title fs-5" id="exampleModalToggleLabel2">
          Відповідь
        </h1>
        <button
          type="button"
          class="btn-close"
          data-bs-dismiss="modal"
          aria-label="Close"
        ></button>
      </div>
      <div class="modal-body">{responseData.message}</div>
```

```
          <div class="modal-footer">
            <button
              type="button"
              class="btn btn-primary"
              data-bs-dismiss="modal"
            >
              {t("close_modal")}
            </button>
          </div>
        </div>
      </div>
    </div>
  </Container>
 );
}

export default Categories;
```

Код з папки connectUs

Файл connectUs.js

```
import { useState } from "react";
import {
 Container,
 Row,
 Col,
 Navbar,
 Nav,
 Button,
 Card,
 Form,
} from "react-bootstrap";
import axios from "axios";
import "./connectUs.css";

function ConnectUs() {
 const [isChatOpen, setIsChatOpen] = useState(false);

 const toggleChat = () => {
  setIsChatOpen(!isChatOpen);
 };

 const [formData, setFormData] = useState({
  name: '',
  numberPhone: '',
  email: '',
  message: '',
 });

 const handleChange = (e) => {
  const { name, value } = e.target;
```

```
  setFormData({ ...formData, [name]: value });
};

const handleSubmit = async (e) => {
  e.preventDefault();
  try {
    await axios.post('http://localhost:3000/api/admin/messages', formData);
    // alert('Сообщение отправлено!');
  } catch (error) {
    console.error('Ошибка отправки сообщения', error);
  }
};


return (
  <div className="connectUs">
    {isChatOpen ? (
      <Container className="my-5 bg-light">
        <Button onClick={(e) => toggleChat()}>Закрити</Button>
    <Form onSubmit={handleSubmit}>
      <Form.Group controlId="formName">
        <Form.Label>Имя</Form.Label>
        <Form.Control
          type="text"
          name="name"
          value={formData.name}
          onChange={handleChange}
        />
      </Form.Group>
      <Form.Group controlId="formNumberPhone">
        <Form.Label>Телефон</Form.Label>
        <Form.Control
          type="text"
          name="numberPhone"
          value={formData.numberPhone}
          onChange={handleChange}
        />
      </Form.Group>
      <Form.Group controlId="formEmail">
        <Form.Label>Email</Form.Label>
        <Form.Control
          type="email"
          name="email"
          value={formData.email}
          onChange={handleChange}
        />
      </Form.Group>
      <Form.Group controlId="formMessage">
        <Form.Label>Сообщение</Form.Label>
        <Form.Control
          as="textarea"
          name="message"
```

```
          value={formData.message}
          onChange={handleChange}
        />
      </Form.Group>
      <Button variant="primary" type="submit">
       Отправить
      </Button>
    </Form>
   </Container>
    ) : (
      <div className="container">
       <button className="round-button" onClick={toggleChat}>
        <svg
          xmlns="http://www.w3.org/2000/svg"
          width="36"
          height="36"
          fill="currentColor"
          className="bi bi-chat-right-text-fill"
          viewBox="0 0 16 16"
        >
          <path d="M16 2a2 2 0 0 0-2-2H2a2 2 0 0 0-2 2v8a2 2 0 0 0 2 2h9.586a1 1 0 0 1 .707.293l2.853
2.853a.5.5 0 0 0 .854-.353zM3.5 3h9a.5.5 0 0 1 0 1h-9a.5.5 0 0 1 0-1m0 2.5h9a.5.5 0 0 1 0 1h-9a.5.5 0 0
1 0-1m0 2.5h5a.5.5 0 0 1 0 1h-5a.5.5 0 0 1 0-1" />
        </svg>
       </button>
      </div>
    )}
   </div>
  );
}

export default ConnectUs;
```

Файл connectUs.css

```
.connectUs {
  display: flex;
  justify-content: center;
  align-items: center;
  /* height: 100vh; */
  margin: 0;
  font-family: Arial, sans-serif;
  /* background-color: #f0f0f0; */
  position: fixed;
  bottom: 20px;
  right: 40px;
  z-index: 9999;
}

.container {
  /* text-align: center; */
```

```
}

.round-button {
  width: 100px;
  height: 100px;
  border-radius: 50%;
  border: none;
  background-color: #4caf50;
  color: white;
  font-size: 16px;
  cursor: pointer;
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2);
  transition: background-color 0.3s ease;
  /* position: fixed;
        bottom: 20px;
        right: 50px;
        z-index: 9999; */
}

.round-button:hover {
  background-color: #45a049;
}

.round-button:active {
  background-color: #3e8e4117;
  box-shadow: 0 5px #666;
  transform: translateY(4px);
}

.chat-close {
  display: grid;
  grid-template-columns: 1fr auto;
}
```

Файл footer.jsx

```
import React from "react";
import { Container, Row, Col } from "react-bootstrap";
import { useTranslation } from "react-i18next";
import "./footer.css"; // Стилі для футера

function Footer() {
  const { t, i18n } = useTranslation();

  const handleChange = (event) => {
    i18n.changeLanguage(event.target.value);
  };

  return (
    <footer className="bg-dark text-white pt-4">
      <Container>
```

```
<Row>
  <Col md={4}>
    <h5>{t("about_us")}</h5>
    <p>{t("about_us_description")}</p>
  </Col>
  <Col md={4}>
    <h5>{t("contacts")}</h5>
    <ul className="list-unstyled">
      <li>{t("phone")}: +380 123 456 789</li>
      <li>{t("email")}: info@service-center.com</li>
      <li>{t("address")}</li>
    </ul>
  </Col>
  <Col md={4}>
    <h5>{t("social_media")}</h5>
    <ul className="list-unstyled">
      <li>
        <a href="#!" className="text-white">
          Facebook
        </a>
      </li>
      <li>
        <a href="#!" className="text-white">
          Twitter
        </a>
      </li>
      <li>
        <a href="#!" className="text-white">
          Instagram
        </a>
      </li>
    </ul>
    <div className="dropdown mt-3">
      <button
        className="btn btn-secondary dropdown-toggle bg-dark"
        style={{ border: "0px" }}
        type="button"
        id="dropdownMenuButton"
        data-bs-toggle="dropdown"
        aria-expanded="false"
      >
        {t("language")}
      </button>
      <ul
        className="dropdown-menu"
        aria-labelledby="dropdownMenuButton"
      >
        <li>
          <button
            className="dropdown-item"
            value="ua"
            onClick={handleChange}
```

```
              >
                {t("ukrainian")}
              </button>
            </li>
            <li>
              <button
                className="dropdown-item"
                value="en"
                onClick={handleChange}
              >
                {t("english")}
              </button>
            </li>
          </ul>
        </div>
      </Col>
    </Row>
    <Row className="mt-3">
      <Col className="text-center">
        <p>
          &copy; 2024 {t("service_center")}. {t("all_rights_reserved")}.
        </p>
      </Col>
    </Row>
  </Container>
 </footer>
 );
}

export default Footer;
```

Файл footer.css

```
footer {
  background-color: #a7acb1;
  padding-top: 20px;
  padding-bottom: 20px;
}

footer h5 {
  margin-bottom: 20px;
  font-size: 16px;
  font-weight: 700;
  color: #ffffff;
}

footer p,
footer li {
  font-size: 14px;
  color: #cccccc;
}
```

```
footer a {
  color: #ffffff;
  text-decoration: none;
}

footer a:hover {
  color: #dddddd;
  text-decoration: underline;
}

footer .list-unstyled {
  padding-left: 0;
  list-style: none;
}
```

Код з папки pages

Файл abouts.js

```
import React from "react";
import { Container, Row, Col, Card } from "react-bootstrap";
import { useTranslation } from "react-i18next";
import "../styles/about.css";
import history from "../image/history.jpg";
import mission from "../image/mission.jpg";

function About() {
  const { t } = useTranslation();

  return (
    <Container className="my-5">
      <Row>
        <Col style={{ background: "#00000090" }}>
          <h1 className="text-center">{t("about_us")}</h1>
          <p className="text-center">{t("about_us_description")}</p>
        </Col>
      </Row>
      <Row className="my-5">
        <Col md={6} className="bg-light">
          <h2>{t("our_history")}</h2>
          <p>{t("history_description")}</p>
        </Col>
        <Col md={6}>
          <img
            src={history}
            style={{ width: "80%", height: "300px" }}
            alt="Our History"
            className="img-fluid"
          />
        </Col>
      </Row>
```

```
      <Row className="my-5">
        <Col md={6}>
          <img
            src={mission}
            style={{ width: "80%", height: "300px" }}
            alt="Our Mission"
            className="img-fluid"
          />
        </Col>
        <Col md={6} className="bg-light">
          <h2>{t("our_mission")}</h2>
          <p>{t("mission_description")}</p>
        </Col>
      </Row>
      <Row className="my-5">
        <Col>
          <h2 className="text-center">{t("our_team")}</h2>
        </Col>
      </Row>
      <Row>
        <Col md={4}>
          <Card>
            <Card.Img variant="top" src="your-team-member-image-url.jpg" />
            <Card.Body>
              <Card.Title>{t("team_member_name")}</Card.Title>
              <Card.Text>{t("team_member_position_1")}</Card.Text>
            </Card.Body>
          </Card>
        </Col>
        <Col md={4}>
          <Card>
            <Card.Img variant="top" src="your-team-member-image-url.jpg" />
            <Card.Body>
              <Card.Title>{t("team_member_name")}</Card.Title>
              <Card.Text>{t("team_member_position_2")}</Card.Text>
            </Card.Body>
          </Card>
        </Col>
        <Col md={4}>
          <Card>
            <Card.Img variant="top" src="your-team-member-image-url.jpg" />
            <Card.Body>
              <Card.Title>{t("team_member_name")}</Card.Title>
              <Card.Text>{t("team_member_position_3")}</Card.Text>
            </Card.Body>
          </Card>
        </Col>
      </Row>
    </Container>
  );
}
```

```
export default About;
```

Файл home.js

```
export default About;
import React from "react";
import {
  Container,
  Row,
  Col,
  Navbar,
  Nav,
  Button,
  Card,
  Form,
} from "react-bootstrap";
import { useTranslation } from "react-i18next";

import { NavLink } from "react-router-dom";

function Home() {
  const { t } = useTranslation();
  return (
    <>
      <main style={{ background: "#2c2c2c40" }}>
        <Container fluid className="p-3  text-center">
          <Row>
            <Col>
              <h1>{t("welcome")}</h1>
              <h5>{t("best_services")}</h5>
              <Button
                style={{ margin: "20px" }}
                variant="primary"
                href="/services"
              >
                {t("servicesOur")}
              </Button>
            </Col>
          </Row>
        </Container>

        <Container className="my-5">
          <Row>
            <Col md={4}>
              <Card>
                <Card.Body>
                  <Card.Title>{t("repair_laptops")}</Card.Title>
                  <Card.Text>{t("repair_laptops_desc")}</Card.Text>
                  <Button variant="primary">
                    <NavLink className="nav-link" to="/services">
                      {t("details")}
```

```
            </NavLink>
          </Button>
        </Card.Body>
      </Card>
    </Col>
    <Col md={4}>
      <Card>
        <Card.Body>
          <Card.Title>{t("repair_phones")}</Card.Title>
          <Card.Text>{t("repair_phones_desc")}</Card.Text>
          <Button variant="primary">
            <NavLink className="nav-link" to="/services">
              {t("details")}
            </NavLink>
          </Button>
        </Card.Body>
      </Card>
    </Col>
    <Col md={4}>
      <Card>
        <Card.Body>
          <Card.Title>{t("sell_spare_parts")}</Card.Title>
          <Card.Text>{t("sell_spare_parts_desc")}</Card.Text>
          <Button variant="primary">
            <NavLink className="nav-link" to="/products">
              {t("details")}
            </NavLink>
          </Button>
        </Card.Body>
      </Card>
    </Col>
  </Row>
</Container>

<Container className="my-5">
  <Row>
    <Col>
      <h2>{t("our_advantages")}</h2>
      <Row>
        <Col md={4}>
          <Card>
            <Card.Body>
              <Card.Title>{t("speed")}</Card.Title>
              <Card.Text>{t("speed_desc")}</Card.Text>
            </Card.Body>
          </Card>
        </Col>
        <Col md={4}>
          <Card>
            <Card.Body>
              <Card.Title>{t("quality")}</Card.Title>
              <Card.Text>{t("quality_desc")}</Card.Text>
```

```
              </Card.Body>
            </Card>
          </Col>
          <Col md={4}>
            <Card>
              <Card.Body>
                <Card.Title>{t("warranty")}</Card.Title>
                <Card.Text>{t("warranty_desc")}</Card.Text>
              </Card.Body>
            </Card>
          </Col>
        </Row>
      </Col>
    </Row>
  </Container>

  <Container>
    <Row>
      <Col>
        <h2>{t("customer_reviews")}</h2>
        <Row>
          <Col md={4}>
            <Card>
              <Card.Body>
                <Card.Text>{t("review_1")}</Card.Text>
                <Card.Footer>{t("reviewer_1")}</Card.Footer>
              </Card.Body>
            </Card>
          </Col>
          <Col md={4}>
            <Card>
              <Card.Body>
                <Card.Text>{t("review_2")}</Card.Text>
                <Card.Footer>{t("reviewer_2")}</Card.Footer>
              </Card.Body>
            </Card>
          </Col>
          <Col md={4}>
            <Card>
              <Card.Body>
                <Card.Text>{t("review_3")}</Card.Text>
                <Card.Footer>{t("reviewer_3")}</Card.Footer>
              </Card.Body>
            </Card>
          </Col>
        </Row>
      </Col>
    </Row>
  </Container>
 </main>
</>
);
```

```
}

export default Home;
```

Файл products.js

```
import React, { useState, useEffect } from "react";
import { Container, Row, Col, Card, Button } from "react-bootstrap";
import axios from "axios";
import "../styles/products.css";
import { useTranslation } from "react-i18next";

function Products() {
  const [products, setProducts] = useState([]);
  const [responseData, setResponseData] = useState("");
  const [buySpareParts, setBuySpareParts] = useState({
    nameClient: "",
    numberPhoneClient: "",
    message: "",
    count: "1",
    deviceId: "",
  });
  const { t } = useTranslation();

  const handleGetProduct = async () => {
    try {
      const response = await axios.get("http://localhost:3000/api/viewing");
      setProducts(response.data);
    } catch (error) {
      console.error("Error fetching products", error);
    }
  };

  useEffect(() => {
    handleGetProduct();
  }, []);

  const handleSubmit = async (e) => {
    e.preventDefault();

    const { nameClient, numberPhoneClient, message, count, deviceId } =
      buySpareParts;
    try {
      const response = await axios.post(
        "http://localhost:3000/api/buySpareParts",
        {
          nameClient,
          numberPhoneClient,
          message,
          count,
          deviceId,
```

```
      }
    );
    setResponseData(response.data);
  } catch (error) {
    console.error("Error submitting form", error);
  }
};

const handleGetBuySpareParts = (id) => {
  setBuySpareParts((prevState) => ({
    ...prevState,
    deviceId: id,
  }));
};

const handleChange = (e) => {
  const { name, value } = e.target;
  setBuySpareParts((prevState) => ({
    ...prevState,
    [name]: value,
  }));
};

return (
  <Container className="my-5">
    <div style={{ background: "#00000090" }} className="mb-5 p-4">
      <h1 className="text-center">{t("spare_parts")}</h1>
    </div>

    <Row>
      {products.map((product) => (
        <Col key={product.id} md={4}>
          <Card className="mb-4">
            <Card.Img variant="top" src={product.photo} />
            <Card.Body>
              <Card.Title>{product.name}</Card.Title>
              <Card.Text>{product.description}</Card.Text>
              <Card.Text>
                <strong>{t("price")}:</strong> {product.price} грн
              </Card.Text>
              <Button
                variant="primary"
                type="button"
                className="btn btn-primary"
                data-bs-toggle="modal"
                onClick={() => handleGetBuySpareParts(product.id)}
                data-bs-target="#exampleModal"
              >
                {t("buy")}
              </Button>
            </Card.Body>
          </Card>
```

```
    </Col>
  ))}
</Row>

<div
  className="modal fade"
  id="exampleModal"
  tabIndex="-1"
  aria-labelledby="exampleModalLabel"
  aria-hidden="true"
>
  <div className="modal-dialog modal-dialog-centered">
    <div className="modal-content">
      <div className="modal-header">
        <h1 className="modal-title fs-4" id="exampleModalLabel">
          {t("new_mess")}
        </h1>
        <br />
        <button
          type="button"
          className="btn-close"
          data-bs-dismiss="modal"
          aria-label="Close"
        ></button>
      </div>
      <div className="modal-body row">
        <form>
          <h4 className="fs-5">{t("buy_spare_part")}</h4>
          <div className="mb-3 row">
            <label htmlFor="recipient-name" className="col-md-4">
              {t("team_member_name")}:
            </label>
            <input
              type="text"
              value={buySpareParts.nameClient}
              name="nameClient"
              onChange={handleChange}
              className="form-control"
              id="recipient-name"
            />
          </div>
          <div className="mb-3 row">
            <label htmlFor="recipient-name" className="col-md-3">
              {t("phone1")}:
            </label>
            <input
              type="text"
              value={buySpareParts.numberPhoneClient}
              name="numberPhoneClient"
              onChange={handleChange}
              className="form-control"
              id="recipient-name"
```

```
          />
        </div>
        <div className="mb-3 row">
          <label htmlFor="recipient-name" className="col-md-2">
            {t("count")}:
          </label>
          <input
            type="number"
            min="1"
            value={buySpareParts.count}
            name="count"
            onChange={handleChange}
            className="form-control"
            id="recipient-name"
          />
        </div>
        <div className="mb-3 row">
          <label htmlFor="message-text" className="col-md-4">
            {t("mess")}:
          </label>
          <textarea
            className="form-control"
            value={buySpareParts.message}
            name="message"
            onChange={handleChange}
            id="message-text"
          ></textarea>
        </div>
      </form>
    </div>
    <div className="modal-footer">
      <button
        type="button"
        className="btn btn-secondary"
        data-bs-dismiss="modal"
      >
        {t("close_modal")}
      </button>
      <button
        type="button"
        data-bs-target="#exampleModalToggle2"
        className="btn btn-primary"
        data-bs-toggle="modal"
        onClick={handleSubmit}
      >
        {t("send_message")}
      </button>
    </div>
  </div>
 </div>
</div>
<div
```

```
        className="modal fade"
        id="exampleModalToggle2"
        aria-hidden="true"
        aria-labelledby="exampleModalToggleLabel2"
        tabIndex="-1"
      >
        <div className="modal-dialog modal-dialog-centered modal-sm">
          <div className="modal-content">
            <div className="modal-header">
              <h1 className="modal-title fs-5" id="exampleModalToggleLabel2">
                Відповідь
              </h1>
              <button
                type="button"
                className="btn-close"
                data-bs-dismiss="modal"
                aria-label="Close"
              ></button>
            </div>
            <div className="modal-body">{responseData.message}</div>
            <div className="modal-footer">
              <button
                type="button"
                className="btn btn-primary"
                data-bs-dismiss="modal"
              >
                {t("close_modal")}
              </button>
            </div>
          </div>
        </div>
      </div>
    </Container>
  );
}

export default Products;
```

Файл services.js

```
import React, { useState, useEffect } from "react";
import { Container, Row, Col, Card, Button } from "react-bootstrap";
import axios from "axios";
import "../styles/services.css";
import { Outlet, useNavigate, useLocation } from "react-router";
import { useTranslation } from "react-i18next";

function Services() {
  const [services, setServices] = useState([]);
  const [isServiceCategories, setIsServiceCategories] = useState(false);
  const navigate = useNavigate();
```

```
const location = useLocation();
const { t, i18n } = useTranslation();

const handleGetProduct = async () => {
  try {
    const response = await axios.get("http://localhost:3000/api/services");
    const data = response.data;

    // Додавання динамічних перекладів
    data.forEach((service) => {
      i18n.addResource(
        "ua",
        "translation",
        `service_name_${service.id}`,
        service.name
      );
      i18n.addResource(
        "ua",
        "translation",
        `service_description_${service.id}`,
        service.description
      );
      i18n.addResource(
        "en",
        "translation",
        `service_name_${service.id}`,
        service.name
      ); // Додайте переклади для англійської мови
      i18n.addResource(
        "en",
        "translation",
        `service_description_${service.id}`,
        service.description
      ); // Додайте переклади для англійської мови
    });

    setServices(data);
  } catch (error) {
    console.error("Error fetching services", error);
  }
};

useEffect(() => {
  handleGetProduct();

  if (location.pathname === "/services") {
    setIsServiceCategories(true);
  }
}, [isServiceCategories, location.pathname]);

const getCategories = (serviceId) => {
  navigate(`/services/categories/${serviceId}`);
```

```
    setIsServiceCategories(false);
  };

  return (
    <Container className="my-5">
      <div style={{ background: "#00000090" }} className="mb-5 p-4">
        <h1 className="text-center">{t("select_service_type")}</h1>
      </div>
      {isServiceCategories ? (
        <Row>
          {services.map((service) => (
            <Col key={service.id} md={4}>
              <Card className="mb-4">
                <Card.Body>
                  <Card.Title>{t(`service_name_${service.id}`)}</Card.Title>
                  <Card.Text>
                    {t(`service_description_${service.id}`)}
                  </Card.Text>
                  <Button
                    onClick={() => getCategories(service.id)}
                    variant="primary"
                  >
                    {t("details")}
                  </Button>
                </Card.Body>
              </Card>
            </Col>
          ))}
        </Row>
      ) : (
        ""
      )}
      <Outlet />
    </Container>
  );
}

export default Services;
```

Код з папки styles

Файл about.css

```
.container {
  padding: 20px;
}

h1,
h2 {
  margin-top: 20px;
  margin-bottom: 20px;
}
```

```
p {
  font-size: 16px;
  line-height: 1.6;
}

.img-fluid {
  max-width: 100%;
  height: auto;
  margin-bottom: 20px;
}

.card {
  margin-bottom: 20px;
}

.text-center {
  text-align: center;
}
```

Файл admin.css

```
.login-background {
  /* background-color: #343a40; Темний фон */
  min-height: 100vh;
  display: flex;
  align-items: center;
  justify-content: center;
}

.login-card {
  background-color: #343a40; /* Білий фон для форми */
  border-radius: 15px; /* Закруглені кути */
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
}

.login-card .form-control {
  border-radius: 10px; /* Закруглені кути для інпутів */
}

.login-card .btn-primary {
  background-color: #007bff; /* Колір кнопки */
  border: none;
  border-radius: 10px;
}

.login-card .btn-primary:hover {
  background-color: #0056b3; /* Колір кнопки при наведенні */
}
```

```
.text-center {
  color: #ffffff; /* Білий колір тексту заголовка */
}
```

Файл products.css

```
.container {
  padding: 20px;
}

h1 {
  margin-top: 20px;
  margin-bottom: 20px;
  text-align: center;
}

.card {
  margin-bottom: 20px;
}
```

Файл services.css

```
.container {
  padding: 20px;
}

h1 {
  margin-top: 20px;
  margin-bottom: 20px;
  text-align: center;
}

.card {
  margin-bottom: 20px;
}
```

Файл App.js

```
import "./App.css";
import { NavLink, Outlet, useLocation } from "react-router-dom";
import { Container, Navbar, Nav } from "react-bootstrap";
import Footer from "./components/footer";
import ConnectUs from "./components/connectUs/connectUs";
import { useTranslation } from "react-i18next";

function App() {
  const { t } = useTranslation();
```

```
  return (
    <>
      <div className="App">
        <header>
          <Navbar bg="dark" variant="dark" expand="lg">
            <Container>
              <Navbar.Brand>{t("service_center")}</Navbar.Brand>
              <Navbar.Toggle aria-controls="basic-navbar-nav" />
              <Navbar.Collapse id="basic-navbar-nav">
                <Nav className="ml-auto">
                  <NavLink className="nav-link" to="/">
                    {t("home")}
                  </NavLink>
                  <NavLink className="nav-link" to="/about">
                    {t("about_us")}
                  </NavLink>
                  <NavLink className="nav-link" to="/services">
                    {t("services")}
                  </NavLink>
                  <NavLink className="nav-link" to="/products">
                    {t("spare_parts")}
                  </NavLink>
                  <NavLink className="nav-link" to="/admin/auth">
                    {t("admin")}
                  </NavLink>
                </Nav>
              </Navbar.Collapse>
            </Container>
          </Navbar>
        </header>

        <Outlet />

        <Footer />
      </div>
      <ConnectUs />
    </>
  );
}

export default App;
```

Файл App.css

```
.App {
  text-align: center;
  background-image: url("./image/backGround.jpg");
  background-attachment: fixed;
  background-repeat: no-repeat;
  background-size: cover;
  background-position: center;
```

```
  background-attachment: fixed;
  background-color: #282c34;
}

.App-logo {
  height: 40vmin;
  pointer-events: none;
}

@media (prefers-reduced-motion: no-preference) {
  .App-logo {
    animation: App-logo-spin infinite 20s linear;
  }
}

.App-header {
  background-color: #282c34;
  min-height: 100vh;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  font-size: calc(10px + 2vmin);
  color: white;
}

.App-link {
  color: #61dafb;
}

@keyframes App-logo-spin {
  from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
}
```

Лістинг серверної частини

Код з папки config

Файл database.js

```
const { Sequelize } = require("sequelize");

const sequelize = new Sequelize(
  "postgres://postgres:2002@localhost:5432"
);

async function connectToPostgreSQL() {
  try {
    await sequelize.authenticate();
    console.log("Successfully connected to PostgreSQL!");
  } catch (error) {
    console.error("Failed to connect to PostgreSQL:", error);
  }
}

module.exports = {
  sequelize,
  connectToPostgreSQL,
};
```

Код з папки controllers

Файл Admin.js

```
const { Model } = require("sequelize");
const {
  Admin,
  ServicesSend,
  SparePartsSend,
  SpareParts,
  Message,
} = require("../models/models");

class AdminController {
  async singIn(req, res) {
    try {
      const { login, password } = req.body;
      const admin = await Admin.findOne({
        where: { name: login, password: password },
      });

      if (!admin) {
        return res.status(400).json({ message: "Invalid login or password" });
      }
      res.json(admin);
```

```
  } catch (error) {
    console.error(error);
    res.status(500).json({ error: "Internal server error" });
  }
}

async deleteService(req, res) {
  try {
    const id = req.params.id;
    console.log(id);
    await ServicesSend.destroy({ where: { id } });
    res.status(200).json({ message: "Service request deleted successfully" });
  } catch (error) {
    console.error("Error deleting service request:", error);
    res.status(500).json({ message: "Error deleting service request" });
  }
}

// Оновити providedService на true
async putServiceProvided(req, res) {
  try {
    const { id } = req.body;
    const serviceSend = await ServicesSend.findByPk(id);

    if (!serviceSend) {
      return res.status(404).json({ message: "Service request not found" });
    }

    serviceSend.providedService = true;
    await serviceSend.save();

    res
      .status(200)
      .json({ message: "Service request updated successfully", serviceSend });
  } catch (error) {
    console.error("Error updating service request:", error);
    res.status(500).json({ message: "Error updating service request" });
  }
}

async putSparePartsProvided(req, res) {
  try {
    const { id, sparePartInStock, needDevice, spareParts_id } = req.body;
    const sparePartsSend = await SparePartsSend.findByPk(id);

    if (!sparePartsSend) {
      return res.status(404).json({ message: "Service request not found" });
    }

    sparePartsSend.providedService = true;
    await sparePartsSend.save();
```

```
    const spareParts = await SpareParts.findByPk(spareParts_id);

    if (!spareParts) {
      return res.status(404).json({ message: "Service request not found" });
    }

    spareParts.number = sparePartInStock - needDevice;
    await spareParts.save();

    res.status(200).json({
      message: "Service request updated successfully",
      sparePartsSend,
    });
  } catch (error) {
    console.error("Error updating service request:", error);
    res.status(500).json({ message: "Error updating service request" });
  }
}

async CreateSpareParts(req, res) {
  try {
    const { number, description, photo, nameSpareParts, price } = req.body;

    const newSparePart = await SpareParts.create({
      number,
      description,
      photo,
      nameSpareParts,
      price,
    });
    res.status(201).json({ message: "Запчастина додана до бази" });
  } catch (error) {
    console.error("Error creating spare part:", error);
    res.status(500).json({ error: "Failed to create spare part" });
  }
}

async GetUsersSpareParts(req, res) {
  try {
    const sparePartsRequests = await SparePartsSend.findAll({
      include: [
        {
          model: SpareParts,
          attributes: [
            "nameSpareParts",
            "description",
            "price",
            "id",
            "number",
          ],
        },
      ],
```

```
    });
    res.status(200).json(spareRequests);
  } catch (error) {
    console.error("Error fetching spare parts requests:", error);
    res.status(500).json({ message: "Error fetching spare parts requests" });
  }
}

async UpdataUsersSpareParts(req, res) {
  const { id } = req.params;
  const { number, description, photo, nameSpareParts, price } = req.body;

  try {
    const sparePart = await SpareParts.findByPk(id);

    if (!sparePart) {
      return res.status(404).json({ message: "Запчастина не знайдена" });
    }

    sparePart.number = number || sparePart.number;
    sparePart.description = description || sparePart.description;
    sparePart.photo = photo || sparePart.photo;
    sparePart.nameSpareParts = nameSpareParts || sparePart.nameSpareParts;
    sparePart.price = price || sparePart.price;

    await sparePart.save();

    res
      .status(200)
      .json({ message: "Запчастина оновлена успішно", sparePart });
  } catch (error) {
    console.error("Error updating spare part:", error);
    res.status(500).json({ message: "Помилка при оновленні запчастини" });
  }
}

// Оновити запит на запчастини
async PutUsersSpareParts(req, res) {
  try {
    const {
      id,
      providedService,
      number,
      nameClient,
      message,
      device_id,
      numberPhoneClient,
    } = req.body;
    const sparePartsRequest = await SparePartsSend.findByPk(id);

    if (!sparePartsRequest) {
      return res
```

```
        .status(404)
        .json({ message: "Spare parts request not found" });
    }

    sparePartsRequest.providedService = providedService;
    sparePartsRequest.number = number;
    sparePartsRequest.nameClient = nameClient;
    sparePartsRequest.message = message;
    sparePartsRequest.device_id = device_id;
    sparePartsRequest.numberPhoneClient = numberPhoneClient;

    await sparePartsRequest.save();

    res.status(200).json({
      message: "Spare parts request updated successfully",
      sparePartsRequest,
    });
  } catch (error) {
    console.error("Error updating spare parts request:", error);
    res.status(500).json({ message: "Error updating spare parts request" });
  }
}

// Видалити запит на запчастини
async DeleteUsersSpareParts(req, res) {
  try {
    const id = req.params.id;
    await SparePartsSend.destroy({ where: { id } });
    res
      .status(200)
      .json({ message: "Spare parts request deleted successfully" });
  } catch (error) {
    console.error("Error deleting spare parts request:", error);
    res.status(500).json({ message: "Error deleting spare parts request" });
  }
}

async DeleteSpareParts(req, res) {
  try {
    const id = req.params.id;
    await SpareParts.destroy({ where: { id } });
    res
      .status(200)
      .json({ message: "Spare parts request deleted successfully" });
  } catch (error) {
    console.error("Error deleting spare parts request:", error);
    res.status(500).json({ message: "Error deleting spare parts request" });
  }
}


async getAllMessages  (req, res)   {
```

```
    try {
      const messages = await Message.findAll();
      res.status(200).json(messages);
    } catch (error) {
      res.status(500).json({ error: 'Ошибка получения сообщений' });
    }
  };

  // Создать сообщение
  async createMessage  (req, res)   {
    try {
      const { name, numberPhone, email, message } = req.body;
      const newMessage = await Message.create({ name, numberPhone, email, message });
      res.status(201).json(newMessage);
    } catch (error) {
      res.status(500).json({ error: 'Ошибка создания сообщения' });
    }
  };

  // Удалить сообщение
  async deleteMessage  (req, res)   {
    try {
      const { id } = req.params;
      await Message.destroy({ where: { id } });
      res.status(204).end();
    } catch (error) {
      res.status(500).json({ error: 'Ошибка удаления сообщения' });
    }
  };
}

module.exports = new AdminController();
```

Файл services.js

```
const { ServicesSend, Categories, Services } = require("../models/models");

const fs = require("fs");
const path = require("path");
const directoryPath = path.join(__dirname, "../uploads");

class ServicesController {
  async requestService(req, res) {   ////////////users book services
    try {



      const { typeServicesId, categoriesServicesId, numberPhoneClient, nameClient, message,
providedService } = req.body;

    console.log(req.body);
```

```
  // Перевіряємо наявність сервісу та категорії
  const serviceExists = await Services.findByPk(typeServicesId);
  const categoryExists = await Categories.findByPk(categoriesServicesId);

  if (!serviceExists) {
    return res.status(400).json({ message: `Service with id ${typeServicesId} does not exist.` });
  }

  if (!categoryExists) {
    return res.status(400).json({ message: `Category with id ${categoriesServicesId} does not exist.` });
  }


    const servicesSend = await ServicesSend.create({
      typeServices: typeServicesId,
      categoriesServices: categoriesServicesId,
      numberPhoneClient: numberPhoneClient,
      nameClient: nameClient,
      providedService: providedService,
      message: message,
    });

    console.log("Запит на послугу надіслано");
    res.status(201).json({
      message: "Запит на послугу надіслано",
      newServiceSend: servicesSend,
    });
  } catch (error) {
    console.error(error);
    res.status(500).json({ error: "Помилка створення запиту" });
  }
}

async getAllnewSpareParts(req, res) { /////////////////////////// admin viewing users book services
  try {
    const servicesSendData = await ServicesSend.findAll({
      include: [
        {
          model: Categories,
          include: [
            {
              model: Services,
            }
          ]
        }
      ]
    });
    res.json(servicesSendData);
  } catch (error) {
    console.error(error);
    res.status(500).json({ error: "Помилка отримання запчастин" });
  }
```

```
}

async GetServicesAll(req, res) { ////////////////////////////////////users viewing services
  try {
    const services = await Services.findAll({
      include: [{
        model: Categories,
        as: 'Categories'
      }]
    });
    res.json(services);
  } catch (error) {
    console.error("Error fetching services with categories", error);
    res.status(500).send("Internal Server Error");
  }
};

async GetServicesCategories( req, res ) {


  try {

    const serviceId = req.params.serviceId;
    console.log(serviceId);
    const categories = await Categories.findAll({
      where: { service_id: serviceId}
    });
    res.json(categories);
  } catch (error) {
    console.error("Error fetching services with categories", error);
    res.status(500).send("Internal Server Error");
  }

}


// async reques(req, res) {   ////////////users book services
//   const { services, categories } = req.body;

// try {
//   // Створюємо послуги
//   const createdServices = await Services.bulkCreate(services);

//   // Отримуємо ідентифікатори створених послуг
//   const serviceMap = createdServices.reduce((acc, service) => {
//     acc[service.name] = service.id;
//     return acc;
//   }, {});

//   console.log(serviceMap.id);

//   // Оновлюємо service_id у категоріях
```

```
  //   const categoriesToCreate = categories.map(category => ({
  //     ...category,
  //     service_id: category.service_id

  //   }));
  //   console.log(categoriesToCreate);

  //   // Створюємо категорії
  //   await Categories.bulkCreate(categoriesToCreate);

  //   res.status(201).json({
  //     message: 'Services and categories created successfully',
  //     services: createdServices,
  //     categories: categoriesToCreate
  //   });
  // } catch (error) {
  //   console.error('Error creating services and categories:', error);
  //   res.status(500).json({ error: 'Internal Server Error' });
  // }
  // }

//   async getUserById(req, res) {
//     try {
//       const userId = req.params.id;

//       const user = await User.findByPk(userId);

//       if (!user) {
//         return res.status(404).json({ error: "Користувача не знайдено" });
//       }

//       res.json(user);
//     } catch (error) {
//       console.error(error);
//       res.status(500).json({ error: "Помилка отримання користувача" });
//     }
//   }

//   async googleLoginUser(req, res) {
//     try {
//       const token = req.body.token;
//       const clientId = req.body.clientId;
//       console.log("Token:", token);

//       const client = new OAuth2Client(clientId);

//       const ticket = await client.verifyIdToken({
//         idToken: token,
//         audience: clientId,
//       });

//       const payload = ticket.getPayload();
```

```
//     const userId = payload["sub"];

//     console.log("User ID:", userId);
//     console.log("Email:", payload.email);
//     console.log("Name:", payload.name);

//     const user = await User.findOne({ where: { email: payload.email } });

//     if (user) {
//       res.status(200).json(user);
//     } else {
//       const newUser = await User.create({
//         username: payload.name,
//         email: payload.email,
//         role: "student",
//       });
//       res.status(200).json(newUser);
//     }
//   } catch (error) {
//     console.error(error);
//     res.status(500).send("Internal Server Error");
//   }
// }

// async loginUser(req, res) {
//   try {
//     const { email, password } = req.body;
//     const user = await User.findOne({ where: { email: email } });

//     if (user) {
//       const isMatch = bcrypt.compareSync(password, user.password);
//       if (isMatch) {
//         const token = jwt.sign({ email: user.email, id: user.id }, keys.jwt, {
//           expiresIn: 3600,
//         });

//         res.status(200).json({
//           token: `Bearer ${token}`,
//           user: {
//             id: user.id,
//             username: user.username,
//             name: user.name,
//             email: user.email,
//             role: user.role,
//           },
//         });
//       } else {
//         res.status(401).json({
//           message: "Неправильний пароль",
//         });
//       }
//     } else {
```

```
//       res.status(404).json({ error: "Користувача не знайдено" });
//     }
//   } catch (error) {
//     console.error(error);
//     res.status(500).json({ error: "Помилка авторизації" });
//   }
// }

// async updateUser(req, res) {
//   try {
//     const userId = req.params.id;
//     const updatedUserData = req.file;

//     const filePath = path.join(directoryPath, updatedUserData.filename);

//     if (fs.existsSync(filePath)) {
//       res.sendFile(filePath);
//     } else {
//       console.error("Файл не найден:", filePath);
//       res.status(404).json({ error: "Файл не найден" });
//     }
//   } catch (error) {
//     console.error(error);
//     res.status(500).json({ error: "Помилка редагування користувача" });
//   }
// }

// async deleteSpareParts(req, res) {
//   try {
//     const userId = req.params.id;

//     const rowsDeleted = await User.destroy({ where: { id: userId } });

//     if (rowsDeleted === 0) {
//       return res.status(404).json({ error: "Користувач не знайдено" });
//     }

//     res.json({ message: "Користувач видалено успішно" });
//   } catch (error) {
//     console.error(error);
//     res.status(500).json({ error: "Помилка видалення користувача" });
//   }
// }
}

module.exports = new ServicesController();
```

Файл spareParts.js

```
// const { body, param, validationResult } = require("express-validator");
const { SpareParts, SparePartsSend } = require("../models/models");
// const bcrypt = require("bcryptjs");
```

```
// const jwt = require("jsonwebtoken");
// const keys = require("../config/jwt");
// const { OAuth2Client } = require("google-auth-library");
// const fs = require("fs");
// const path = require("path");
// const directoryPath = path.join(__dirname, "../uploads");

class SparePartsController {
  async add(req, res) {
    try {


      const { number, description, photo, nameSpareParts, price } = req.body;

      const newSpareParts = await SpareParts.create({
        number: number,
        description: description,
        photo: photo,
        nameSpareParts: nameSpareParts,
        price: price,
      });

      console.log("Запчастина успішно доданно в базу даних");
      res.status(201).json(newSpareParts);
    } catch (error) {
      console.error(error);
      res.status(500).json({ error: "Помилка створення запчастин" });
    }
  }

  async getAllnewSpareParts(req, res) {
    try {
      const spareParts = await SpareParts.findAll();
      res.json(spareParts);
    } catch (error) {
      console.error(error);
      res.status(500).json({ error: "Помилка отримання запчастин" });
    }
  }

  async buySpareParts(req, res) {
    try {
      const { nameClient, numberPhoneClient, message, count, deviceId } = req.body;

      console.log(req.body);

      const servicesSend = await SparePartsSend.create({
        number: count,
        nameClient: nameClient,
        message: message,
        device_id: deviceId,
        numberPhoneClient: numberPhoneClient,
```

```
      });

      console.log("Запит на послугу надіслано");
      res.status(201).json({
        message: "Запит на послугу надіслано",
      });
    } catch (error) {
      console.error(error);
      res.status(500).json({ error: "Помилка створення запиту" });
  }
}

//   async getUserById(req, res) {
//     try {
//       const userId = req.params.id;

//       const user = await User.findByPk(userId);

//       if (!user) {
//         return res.status(404).json({ error: "Користувача не знайдено" });
//       }

//       res.json(user);
//     } catch (error) {
//       console.error(error);
//       res.status(500).json({ error: "Помилка отримання користувача" });
//     }
//   }

//   async googleLoginUser(req, res) {
//     try {
//       const token = req.body.token;
//       const clientId = req.body.clientId;
//       console.log("Token:", token);

//       const client = new OAuth2Client(clientId);

//       const ticket = await client.verifyIdToken({
//         idToken: token,
//         audience: clientId,
//       });

//       const payload = ticket.getPayload();
//       const userId = payload["sub"];

//       console.log("User ID:", userId);
//       console.log("Email:", payload.email);
//       console.log("Name:", payload.name);

//       const user = await User.findOne({ where: { email: payload.email } });

//       if (user) {
```

```
//       res.status(200).json(user);
//     } else {
//       const newUser = await User.create({
//         username: payload.name,
//         email: payload.email,
//         role: "student",
//       });
//       res.status(200).json(newUser);
//     }
//   } catch (error) {
//     console.error(error);
//     res.status(500).send("Internal Server Error");
//   }
// }

// async loginUser(req, res) {
//   try {
//     const { email, password } = req.body;
//     const user = await User.findOne({ where: { email: email } });

//     if (user) {
//       const isMatch = bcrypt.compareSync(password, user.password);
//       if (isMatch) {
//         const token = jwt.sign({ email: user.email, id: user.id }, keys.jwt, {
//           expiresIn: 3600,
//         });

//         res.status(200).json({
//           token: `Bearer ${token}`,
//           user: {
//             id: user.id,
//             username: user.username,
//             name: user.name,
//             email: user.email,
//             role: user.role,
//           },
//         });
//       } else {
//         res.status(401).json({
//           message: "Неправильний пароль",
//         });
//       }
//     } else {
//       res.status(404).json({ error: "Користувача не знайдено" });
//     }
//   } catch (error) {
//     console.error(error);
//     res.status(500).json({ error: "Помилка авторизації" });
//   }
// }

// async updateUser(req, res) {
```

```
//     try {
//       const userId = req.params.id;
//       const updatedUserData = req.file;

//       const filePath = path.join(directoryPath, updatedUserData.filename);

//       if (fs.existsSync(filePath)) {
//         res.sendFile(filePath);
//       } else {
//         console.error("Файл не найден:", filePath);
//         res.status(404).json({ error: "Файл не найден" });
//       }
//     } catch (error) {
//       console.error(error);
//       res.status(500).json({ error: "Помилка редагування користувача" });
//     }
//   }

  async deleteSpareParts(req, res) {
    try {
      const id = req.params.id;

      const rowsDeleted = await SpareParts.destroy({ where: { id: id } });

      if (rowsDeleted === 0) {
        return res.status(404).json({ error: "Запчастину не знайдено" });
      }

      res.json({ message: "Запчастину видалено успішно" });
    } catch (error) {
      console.error(error);
      res.status(500).json({ error: "Помилка видалення запчастини" });
    }
  }
}

module.exports = new SparePartsController();
```

Код з папки models

Файл models.js

```
const { Sequelize, DataTypes } = require("sequelize");
const sequelize = new Sequelize(
  "postgres://postgres:2002@localhost:5432"
);

const SpareParts = sequelize.define('SpareParts', {
  number: {
    type: DataTypes.INTEGER,
```

```
    allowNull: true,
  },
  description: {
    type: DataTypes.STRING,
    allowNull: true,
  },
  photo: {
    type: DataTypes.TEXT,
    allowNull: true,
  },
  nameSpareParts: {
    type: DataTypes.STRING,
    allowNull: true,
  },
  price: {
    type: DataTypes.STRING,
    allowNull: true,
  },
  createdAt: {
    type: DataTypes.DATE,
    allowNull: false,
    defaultValue: Sequelize.NOW,
  },
  updatedAt: {
    type: DataTypes.DATE,
    allowNull: false,
    defaultValue: Sequelize.NOW,
  },
});

// Модель SparePartsSend
const SparePartsSend = sequelize.define('SparePartsSendsBook', {
  number: DataTypes.INTEGER,
  nameClient: DataTypes.STRING,
  message: DataTypes.TEXT,
  device_id: DataTypes.INTEGER,
  numberPhoneClient: DataTypes.STRING,
  providedService: DataTypes.BOOLEAN,
});


const ServicesSend = sequelize.define("ServicesBook", {
  typeServices: DataTypes.INTEGER,
  categoriesServices: DataTypes.INTEGER,
  numberPhoneClient: DataTypes.STRING,
  nameClient: DataTypes.STRING,
  providedService: DataTypes.BOOLEAN,
  message: DataTypes.TEXT,
});

const Admin = sequelize.define("Admin", {
  name: DataTypes.STRING,
```

```
  password: DataTypes.STRING,
});

const Services = sequelize.define("Services", {
  name: {
    type: DataTypes.STRING,
    allowNull: false,
  },
  description: {
    type: DataTypes.TEXT,
    allowNull: true,
  },
});

// Модель для таблиці categories
const Categories = sequelize.define("Categories", {
  service_id: DataTypes.INTEGER,
  name: {
    type: DataTypes.STRING,
    allowNull: false,
  },
});

const Message = sequelize.define('Message', {
  name: {
    type: DataTypes.STRING,
    allowNull: false,
  },
  numberPhone: {
    type: DataTypes.STRING,
    allowNull: false,
  },
  email: {
    type: DataTypes.STRING,
    allowNull: false,
  },
  message: {
    type: DataTypes.TEXT,
    allowNull: false,
  },
}, {
  timestamps: true,
});

// Визначення зв'язків
Services.hasMany(Categories, { foreignKey: 'service_id' });
Categories.belongsTo(Services, { foreignKey: 'service_id' });

Services.hasMany(ServicesSend, { foreignKey: 'typeServices' });
ServicesSend.belongsTo(Services, { foreignKey: 'typeServices' });

Categories.hasMany(ServicesSend, { foreignKey: 'categoriesServices' });
```

```
ServicesSend.belongsTo(Categories, { foreignKey: 'categoriesServices' });


SparePartsSend.belongsTo(SpareParts, { foreignKey: 'device_id' });
SpareParts.hasMany(SparePartsSend, { foreignKey: 'device_id' });


sequelize
  .sync({ alter: true })
  .then(() => {
    console.log("Models synchronized successfully");
  })
  .catch((error) => {
    console.error("Error synchronizing models");
  });

module.exports = { SpareParts, Services, Categories, ServicesSend, Admin, SparePartsSend, Message };
```

Код з папки routers

Файл admin.js

```
const express = require('express')
const router = express.Router();
const admin = require('../controllers/Admin.js')

router.post('/admin/auth',  admin.singIn )

// http://localhost:3000/api/admin/services/
// router.get('/admin/services', admin.GetServicesAll )

// http://localhost:3000/api/admin/servicesDelete
router.delete('/admin/servicesDelete/:id', admin.deleteService)
router.put('/admin/services', admin.putServiceProvided )

router.post('/admin/createSpareParts', admin.CreateSpareParts)
router.get('/admin/userSpareParts', admin.GetUsersSpareParts )
router.put('/admin/updateSpareParts/:id', admin.UpdataUsersSpareParts )

router.put('/admin/userSpareParts', admin.PutUsersSpareParts )
router.put('/admin/sparePartsProvided', admin.putSparePartsProvided )
router.delete('/admin/userSpareParts/:id', admin.DeleteUsersSpareParts )
router.delete('/admin/spareParts/:id', admin.DeleteSpareParts )

router.get('/admin/messages', admin.getAllMessages);
router.post('/admin/messages', admin.createMessage);
router.delete('/admin/messages/:id', admin.deleteMessage);

module.exports = router
```

Файл services.js

```
const express = require('express')
const router = express.Router();
const services = require('../controllers/services.js')

// http://localhost:5000/api/servicesUsersBook
router.get('/admin/services', services.getAllnewSpareParts )
// http://localhost:3000/api/servicesCreate/Send
router.post('/servicesCreate/Send',  services.requestService )
// http://localhost:3000/api/services/categories/:serviceId
router.get('/services/categories/:serviceId', services.GetServicesCategories )

// http://localhost:3000/api/admin/services/
router.get('/services', services.GetServicesAll )
// http://localhost:3000/api/admin/services/
// router.get('/admin/servicesDelete', services.deleteService)

// router.post('/services', services.reques )

module.exports = router
```

Файл spare-parts.js

```
const express = require('express')
const routerChange = express.Router();
const spareParts = require('../controllers/spareParts.js')

// http://localhost:3000/api/viewing

routerChange.get('/viewing', spareParts.getAllnewSpareParts )
// http://localhost:3000/api/changeSpareParts
// routerChange.put('/changeSpareParts',  spareParts.change )
// http://localhost:3000/api/addSpareParts
routerChange.post('/addSpareParts',  spareParts.add )
routerChange.post('/buySpareParts',  spareParts.buySpareParts )
// http://localhost:3000/api/deleteSpareParts/:id
routerChange.delete('/deleteSpareParts/:id',  spareParts.deleteSpareParts )

module.exports = routerChange
```

Файл app.js

```
const app = require('./server');
const port = 3000;

app.listen(port, () => {
  console.log(`app listening at http://localhost:${port}`);
});
```

Файл server.js

```
const express = require('express');
const morgan = require('morgan');
const body = require('body-parser')
const app = express();
const axios = require('axios');
const cheerio = require('cheerio');

const { connectToPostgreSQL } = require("./config/database.js");

connectToPostgreSQL();



const spareParts = require('./routers/spare-parts')
const services = require('./routers/services')
const admin = require('./routers/admin.js')

app.use(morgan('dev'));
app.use(require('cors')())
app.use(express.json({ limit: '50mb' })); // Устанавливаем лимит для JSON запросов
app.use(express.urlencoded({ limit: '50mb', extended: true }));
app.use(body.json());

app.use('/api', spareParts);
app.use('/api', services);
app.use('/api', admin);

// const fetchData = async (url) => {
//     try {
//       const response = await axios.get(url);
//     //   console.log(response.data);
//       return response.data;
//     } catch (error) {
//       console.error(`Error fetching data: ${error}`);
//       return null;
//     }
//   };

//   // Функція для парсингу HTML-коду
//   const parseData = (html) => {
//     const $ = cheerio.load(html);
//     const parsedData = [];
//     $('.img200').each((index, element) => {
//         console.log(element);
//       const title = $(element).find('#img_200_1887588').attr('alt');
//       const content = $(element).find('#img_200_1887588').attr('src');
//       parsedData.push({ title, content });
//     });
//     return parsedData;
//   };
```

```
//   // Маршрут для отримання парсингу даних
//   app.get('/api/posts', async (req, res) => {
//     const url = 'https://ek.ua/ua/APPLE-SILICONE-CASE-WITH-MAGSAFE-FOR-IPHONE-12-12-PRO.htm'; // Замість цього URL використовуйте потрібний вам
//     const html = await fetchData(url);
//     if (html) {
//       const data = parseData(html);
//       res.json(data);
//     } else {
//       res.status(500).send('Error fetching data');
//     }
//   });

module.exports = app;
```