МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет прикладної математики та інформатики
(повне найменування назва факультету)

Кафедра інформаційних систем
(повна назва кафедри)

ДИПЛОМНА РОБОТА

РОЗРОБКА ВЕБСАЙТУ ДЛЯ БАРБЕРШОПУ

Виконав(ла): студент(ка) групи ПМІ-42
спеціальності 122 – комп'ютерні науки
(шифр і назва спеціальності)

Заграй Д. М.
(підпис) (прізвище та ініціали)

Керівник доц. Горлач В. М.
(підпис) (прізвище та ініціали)

Рецензент
(підпис) (прізвище та ініціали)

2023

ЗМІСТ

ВСТУП

Сьогодні важко уявити наше життя без вебсайтів. Ще десять років тому інтернет був чимось новим, якимсь «ринком майбутнього», а зараз люди користуються ним чи не щодня. Багато різних систем працюють за допомогою інтернету, для прикладу системи електронного навчання, які є у більшості сучасних вищих навчальних закладів. Якщо спробувати перерахувати всі сфери нашого життя, де зараз застосовується всесвітня мережа, можливо, не вистачило б і десятка годин. Станом на початок 2021 року кількість українських інтернет-користувачів становила майже 30 мільйонів, тобто близько 67% населення. Інтернет вже обійшов телебачення, як основне джерело інформації, і продовжує набирати оберти. Не стоїть на місці і доступ до мережі з мобільних пристроїв – Україна піднялася в глобальному рейтингу швидкості мобільного інтернету на 15 позицій, повідомляє Ookla Speedtest Global Index, - з 77 місця в січні 2021 на 62 позицію в січні 2022 року [1]. Підсумовуючи, можна сміливо сказати, що всесвітня мережа стала невід'ємною частиною нашого життя, а тому і розробка в цьому напрямку є досить актуальною.

Темою дипломної роботи є «Розробка вебсайту для барбершопу», тому звернемо увагу саме на те, чи потрібен бізнесу свій сайт? Існує багато противників власного інтернет-ресурсу: одні вважають, що це зайва витрата часу і коштів, інші, що існують вже безкоштовні соціальні мережі, де можна просувати свою справу, а ще інші просто відмовляються, тому що мають достатню кількість клієнтів. Однак є багато причин для бізнесу володіти власним вебсайтом, ось деякі з них [2]:

- візитка компанії (люди часто оцінюють компанію дивлячись на її вебсторінку, вона надає основну інформацію про бізнес, його товари і послуги);
- додаткове залучення клієнтів (одна з найкращих причин для чого потрібен інтернет-ресурс, який до всього ще й буде працювати цілодобово без вихідних);
- вихід на нові ринки (про бізнес може дізнатись увесь світ);

- комерційна перспектива (завдяки сайту можна продавати різні товари і послуги онлайн).

Отже, якщо проаналізувати наведені вище причини, то зараз майже кожному бізнесу потрібен власний вебсайт. Відштовхуючись від теми, барбершопу інтернет-ресурс дозволить продемонструвати власні роботи, колектив, інтер'єр закладу тощо. Також це додатковий засіб залучення клієнтів, адже дуже часто такий сайт дозволяє людині сформувати візит до певного майстра у вказаний час, відповідно кошти, які були витрачені на розробку – можуть дуже швидко окупитися, а інтернет-ресурс почне приносити дохід. Такий підхід діє не тільки для барбершопів, а й для всіх підприємств, які займаються наданням певних послуг людям (салони краси, приватні клініки, СТО і тому подібне).

Основна мета роботи – це отримання базових навичок розробки вебсайтів, використовуючи сучасні технології. Зараз існує багато новітніх інструментів, які дозволяють розробляти сайти на зовсім іншому рівні, аніж кілька років тому. Також метою роботи є створення вебпроекту за домогою цих технологій. Це дозволить виділити певні етапи розробки, краще зрозуміти як побудовані вебсайти, які нас оточують і узагальнити знання про розробку вебсайтів.

Отже, на сьогоднішній день очевидним є те, що інтернет вже став невід'ємною частиною нашого життя, тому і сучасному бізнесу потрібно бути в тренді і використовувати всі можливості для підвищення конкурентоспроможності. Майже кожному підприємству потрібен власний вебсайт і у цьому вступі були наведені аргументи чому, зважаючи на це, тема розробки інтернет-ресурсу є актуальною.

1. ЕТАП ФОРМУВАННЯ ІДЕЇ ВЕБСАЙТУ, СТВОРЕННЯ ДІАГРАМ

1.1. Ідея

Робота розпочалась з етапу формування ідеї вебсайту. Дуже велику роль відіграє саме цей етап, тому що, важко виконувати якусь роботу із зав'язаними очима, потрібно чітко розуміти, що потрібно від інтернет-ресурсу і які функції він зможе виконувати. Щоб дізнатись більше про сферу цієї роботи, для початку можна відвідати деякі вебсайти існуючих барбершопів по всьому світу. Всі вони між собою сильно відрізняються, але можна виділити одну спільну рису – дуже часто такий сайт спочатку демонструє відомості про колектив, інтер'єр, ціни, інколи відгуки тощо, а вже опісля, такий ресурс надає можливість сформувати візит.

Формування візиту часто відбувається за допомогою зовсім іншої веб-аплікації. На основі спостережень були виписані деякі особливості в окремий текстовий файл. Після цього визначено основні завдання проекту:

- демонстрація основної інформації про барбершоп відвідувачу;
- можливість сформувати візит на основі вибору його деталей (барбер, послуга, дата й час), а також, щоб такий візит формувався автоматично;
- можливість адміністратору бачити записи і якось з ними взаємодіяти.

1.2. Діаграма варіантів використання

Щоб узагальнити усі ці ідеї була створена діаграма варіантів використання (англ. *Use case diagram*) на якій зображені відношення між дійовими особами і можливостями цих осіб в системі (див. Рисунок 1.1) [3]. Зліва на цій діаграмі зображено дійову особу «Користувач», яка може: переглянути основну інформацію (про барбершоп) і сформувати візит. Формування візиту включає в себе такі кроки: вибір барбера, вибір послуги, вибір дати і часу і ввід контактних даних. Справа на діаграмі зображено дійову особу «Адмін», яка може: переглядати записи, переглядати клієнтів, переглядати барберів і міняти статус записів (скасовувати їх чи підтверджувати). Всі ці дії адміністратор зможе виконувати лише після

авторизації. Отже, така діаграма дозволяє чітко визначити, хто буде користуватись системою і яким чином.

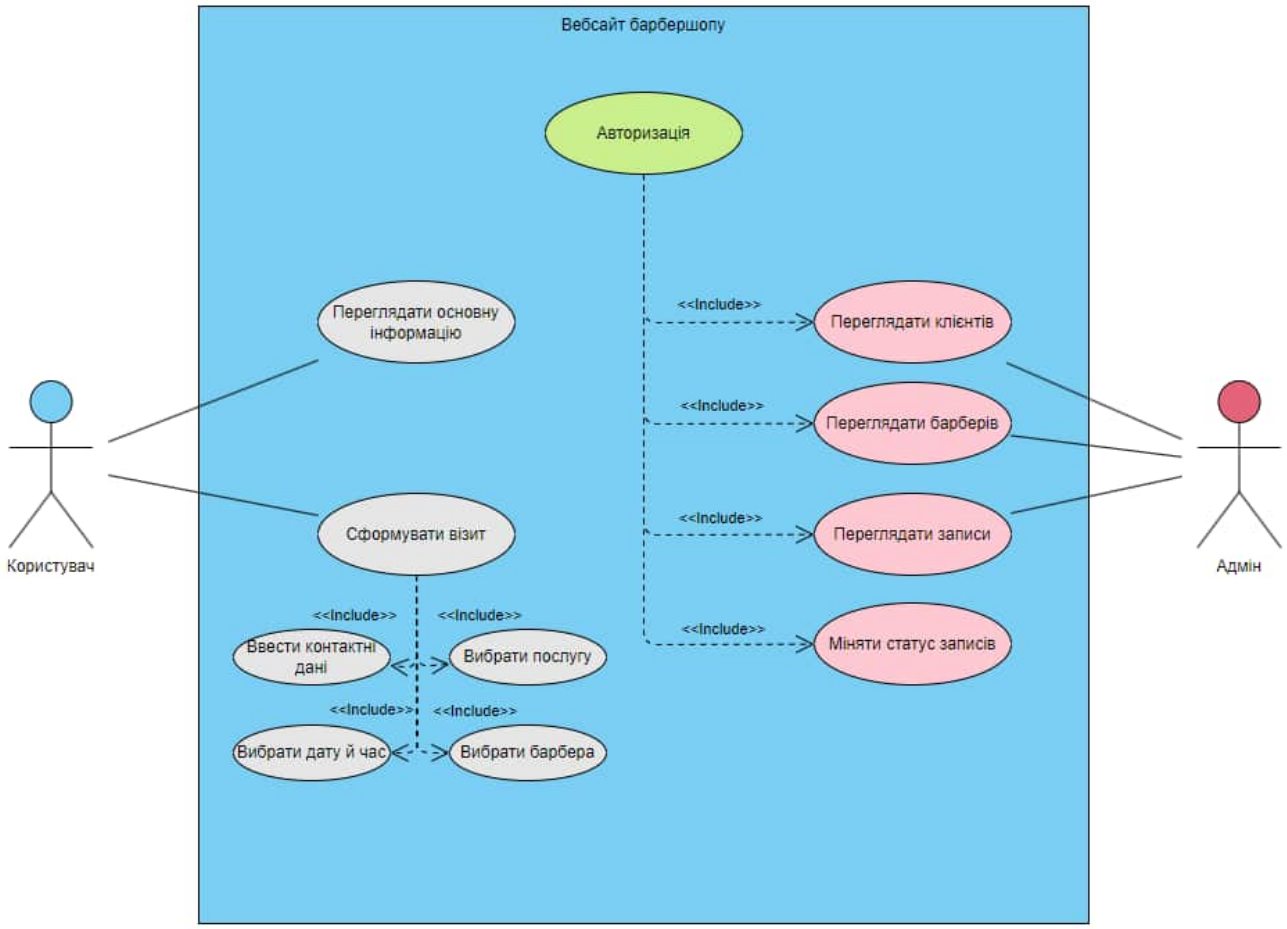

Рисунок 1.1

1.3.Діаграма «сутність – зв’язок»

Звісно, що такий проект потребує певної бази даних, де буде зберігатись різна інформація, наприклад: про барберів, про послуги, про записи і так далі. Для того, щоб визначити усі ці сутності і зв'язки між ними, існує спеціальна модель «сутність – зв'язок» (англ. *ER-model*) [4]. За допомогою спеціальних блоків сформована концепція бази даних вебсайту (див. Рисунок 1.2). Для цього, як і для діаграми варіантів використання, можна скористатись допомогою безкоштовних інтернет-ресурсів, які дозволяють створювати різні діаграми з їх ключових елементів.

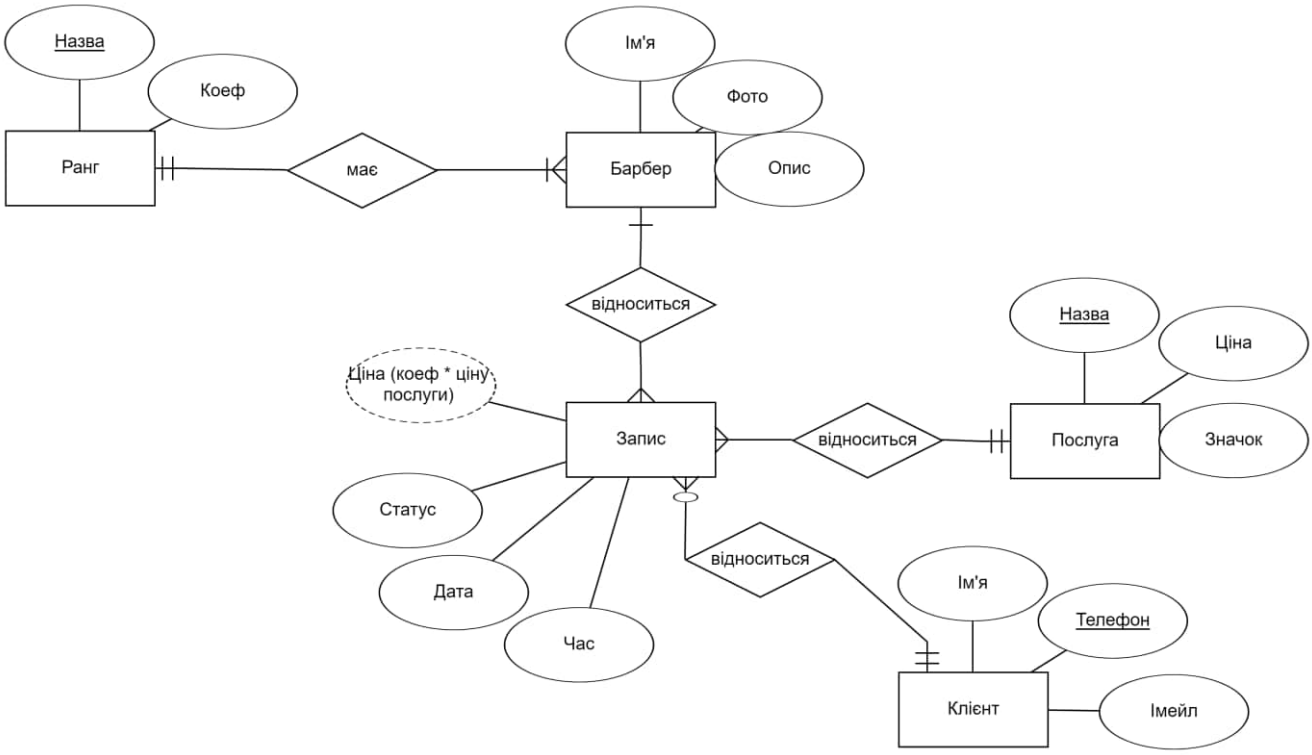

Рисунок 1.2

Визначено п’ять сутностей, які будуть у базі даних: запис, клієнт, послуга, барбер і ранг. Сутність «Барбер» має атрибути «ім’я», «фото», «опис» і за допомогою зовнішнього ключа буде зв’язуватись з іншою сутністю «Ранг», їх зв’язок чітко визначено як один до багатьох, тобто умовний барбер «А» і барбер «В» можуть мати ранг «А», але барбер «А» – не може мати зв’язок з більш, ніж одним рангом. Сутності «Послуга» і «Клієнт» мають лише певні атрибути і пов’язані з сутністю «Запис». Вона буде головною сутністю системи. «Запис» пов’язує в собі всі інші таблиці. Варто розповісти про атрибут «Ціна». Ціна за запис буде формуватись на основі атрибутів з інших таблиць. Для прикладу: клієнт формує запис і вибирає певного барбера з рангом «Gold», цей ранг в своїй таблиці має атрибут «Коефіцієнт», який в подальшому ще знадобиться, дальше відвідувач обирає послугу, яка в свою чергу має атрибут «Ціна», і на основі вибраних пропозицій ми формуємо ціну за послугу, помноживши значення з атрибуту «Коефіцієнт» на значення атрибуту «Ціна». Як вже згадувалось раніше, ця діаграма допоможе будувати базу даних і писати серверну частину проекту, також розуміючи, що можна отримати від сервера, набагато легше описувати інтерфейс користувача.

Таким чином, після дослідження подібних вебсайтів були створені необхідні діаграми, які в подальшому допоможуть при розробці проекту. Основною метою цього етапу було перенесення ідеї сайту з голови у якийсь матеріальний і визначений вигляд.

2. ВИБІР ЗАСОБІВ І ТЕХНОЛОГІЙ

Для початку потрібно зрозуміти, яка саме робота буде виконуватись, щоб правильно обрати технології. Багато навчальних ресурсів поділяють етап розробки на три фази: дизайн вебсторінки, «front end» розробка та «back end» розробка. І якщо з етапом дизайну все зрозуміло, то «Front end» - це клієнтська частина програми (інтерфейс користувача), яка дозволяє комунікувати з серверною частиною, а «back end» - це саме ця серверна частина, яка обробляє запити від користувача і, можливо, надає відповідь [5].

Дизайн проекту буде створюватись за допомогою Adobe Photoshop. Існує велика кількість різних спеціальних програм для цього, але дуже часто вони не розповсюджуються безкоштовно. Adobe Photoshop надає пробний тижневий період, якого буде досить для створення дизайну проекту. Ця програма має величезну кількість різних інструментів, які стануть у пригоді [6].

Для розробки клієнтської частини проекту буде використовуватись HTML, SCSS, JavaScript та React. HTML – це мова розмітки вебсторінок. SCSS – це своєрідний «діалект» мови SASS. Код SCSS інтерпретується автоматично в мову каскадних таблиць стилів. Вона призначена для підвищення рівня абстракції коду і спрощення CSS файлів. SASS – це оригінальна метамова, яка має власний синтаксис, а SCSS – це ця ж мова, тільки з синтаксисом більш подібним до звичайного CSS [7-8]. Одна з вимог роботи – використання сучасних технологій, тому був обраний саме такий підхід до написання стилів.

JavaScript – це мова для створення сценаріїв вебсторінки, яка надає багато різних можливостей: налаштування взаємодії користувача з ресурсом, обмін даними з сервером, керування браузером, зміна структурних елементів вебсайту і тому подібне [9].

React – це відкрита бібліотека для створення інтерфейсів користувача, яка покликана вирішувати проблеми часткового оновлення вмісту вебсторінки. Також

вона містить в собі велику кількість шаблонів, які допоможуть при розробці проекту [10].

Для розробки серверної частини вебзастосунку буде використовуватись .NET 7, Entity Framework, ASP.NET Core та система керування базами даних PostgreSQL. .NET – це платформа від Microsoft, яка дозволяє створювати програмні застосунки. Вона використовує мову програмування C# (хоча й не тільки), яка є об’єктно-орієнтованою мовою програмування, а також ця платформа є досить популярною і постійно оновлюється. З її допомогою можна створювати проекти різної складності [11].

Entity Framework – це спеціальний набір технологій для роботи з базами даних, який використовується на .NET. Він дозволяє працювати напряму із сутностями, а не з таблицями, і не турбуватись за підключення, підготування SQL і параметрів, відправку запитів і транзакцій. Цей інструмент дозволить писати код програми набагато швидше.

ASP.NET Core – це спеціальна платформа для розробки саме вебзастосунків від компанії Microsoft. Цей фреймворк працює на .NET і використовує мову програмування C#.

Остання технологія в списку – система керування базами даних PostgreSQL. Ця технологія містить безліч функцій, які роблять її одним із хороших варіантів вибору з усіх баз даних [12].

3. СТВОРЕННЯ ДИЗАЙНУ ЗА ДОПОМОГОЮ ADOBE PHOTOSHOP

Щоб створити дизайн вебпроекту потрібно повернутись до діаграми варіантів використання (див. Рисунок 1.1). На ній зображено, що «Користувач» може переглядати основну інформацію і формувати візити. Для того, щоб подати інформацію користувачу буде створюватись дизайн в стилі лендинг (сайт-вітрина). Такий підхід дозволить зображувати на одній сторінці всю інформацію, якою може цікавитись відвідувач вебсайту, а також лендинг повинен реалізовувати можливість переходу на окрему сторінку, яка буде використовуватись для формування візиту.

3.1. Підготовка проєкту

Першим кроком був створений проект в Adobe Photoshop і задана ширина полотна на якій буде відбуватись робота – 1920 пікселів, а висота – 1080 пікселів. Це стандартний формат Full HD, який використовують майже завжди у вебдизайні. Такий підхід дозволить створювати секції, з яких буде складатись проект, одна за одною, змінюючи висоту полотна.

3.2. Розробка дизайну секцій вебсторінки

Першою секцією лендингу буде «Промо» (див. Рисунок 3.1). Це стартова частина вебсайту, яку побачить відвідувач, який на нього потрапив. Зверху цієї секції розміщений логотип і меню навігації по сторінці. Додано чорний фон, щоб ці елементи добре виділялись. В центрі секції знаходиться великий текст із слоганом барбершопу. Нижче знаходиться спеціальна кнопка, яка буде перенаправляти користувача на сторінку формування візиту, також вона буде мати певні стилі відповідно до положення мишки на ній. Уся секція на задньому фоні буде мати фото.

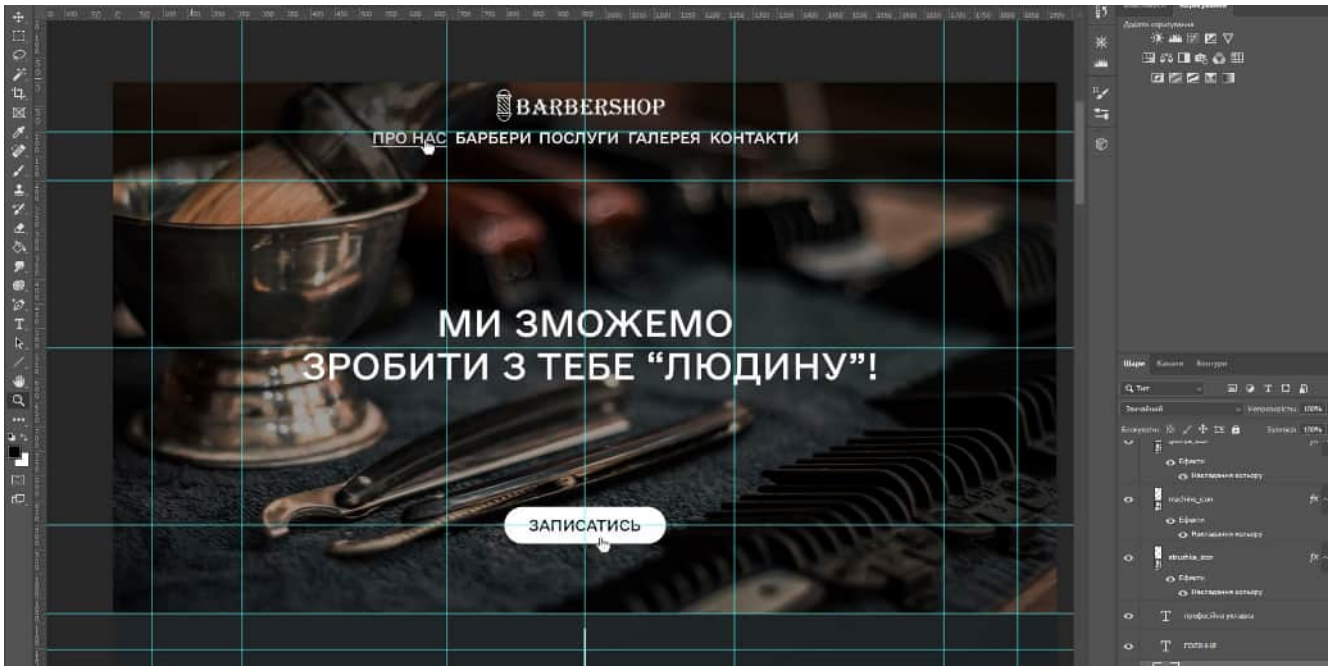

Рисунок 3.1

Також потрібно зауважити, що фон для всіх секцій зроблений єдиним градієнтом. Отже, після закінчення меж фото заднього фону на першій секції, користувач буде бачити плавний перехід кольору до кінця сторінки.

Наступною секцією в дизайні буде «Про нас» (див. Рисунок 3.2). Ця секція повинна розповідати користувачу коротку історію барбершопу.

Рисунок 3.2

В її центрі присутнє фото закладу, а нижче – текстова частина. Зверху повинно з’являтись динамічне меню навігації по сторінках, яке дозволить повертатись на секцію назад або перейти на секцію нижче. Також кнопка «Записатись», яка була в «Промо», повинна змінювати своє положення і слідувати за відвідувачем сайту. Можна побачити стилі кнопок у навігаційній панелі, якщо на них навести курсором миші.

Інша секція – «Наші барбери» (див. Рисунок 3.3). Її основне завдання продемонструвати фото барберів у спеціальних блоках. Спочатку до фото буде застосовуватись чорно-білий фільтр, але коли користувач наведе курсором на блок, то він плавно зникне і з’явиться ім’я барбера. Знизу можна додати кілька слів опису майстрів.

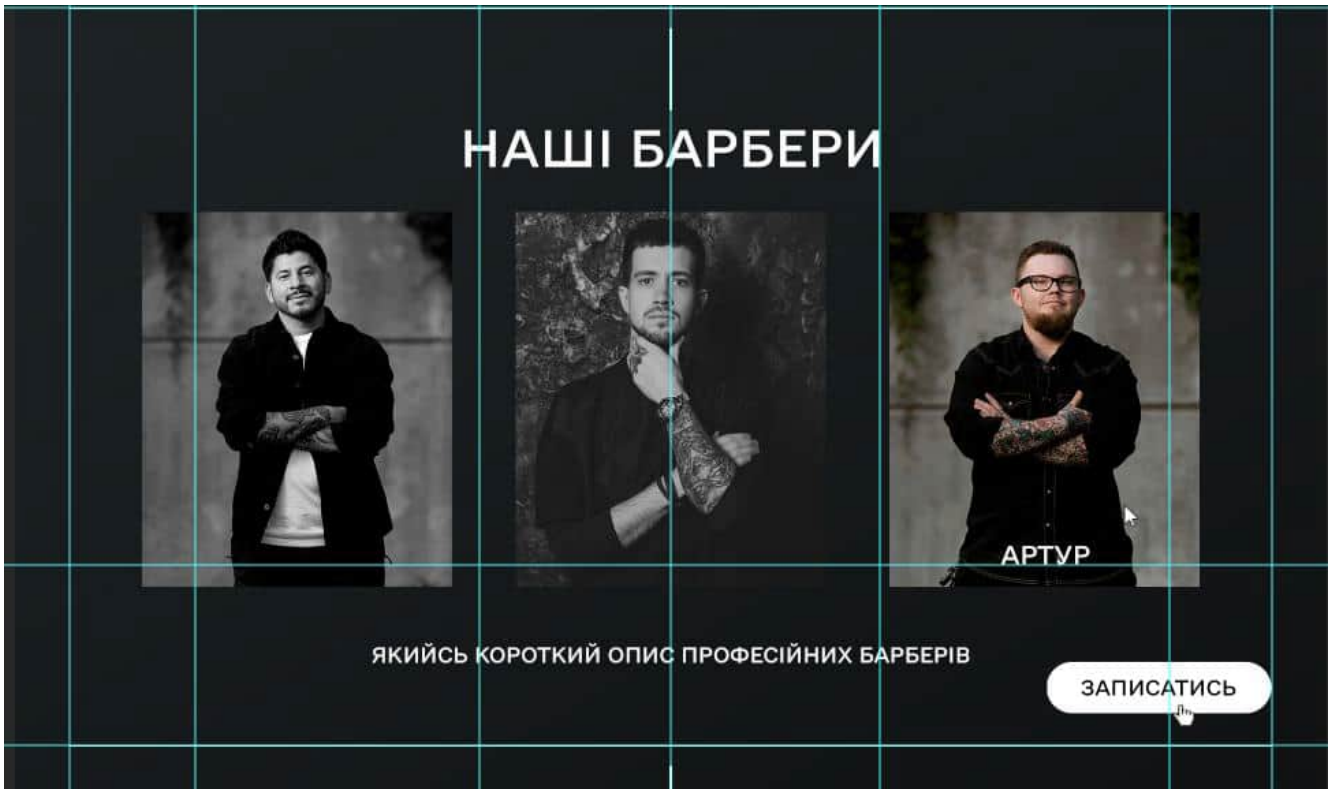

Рисунок 3.3

Наступна секція – «Послуги» (див. Рисунок 3.4). Уже з назви стає зрозуміло для чого вона потрібна. На цій секції користувач зможе отримати основну інформацію про послуги, які надає барбершоп. Вона буде відображатись у вигляді карток, які складаються зі значка, назви і найнижчої ціни за послугу. Також знизу присутнє повідомлення для відвідувача.

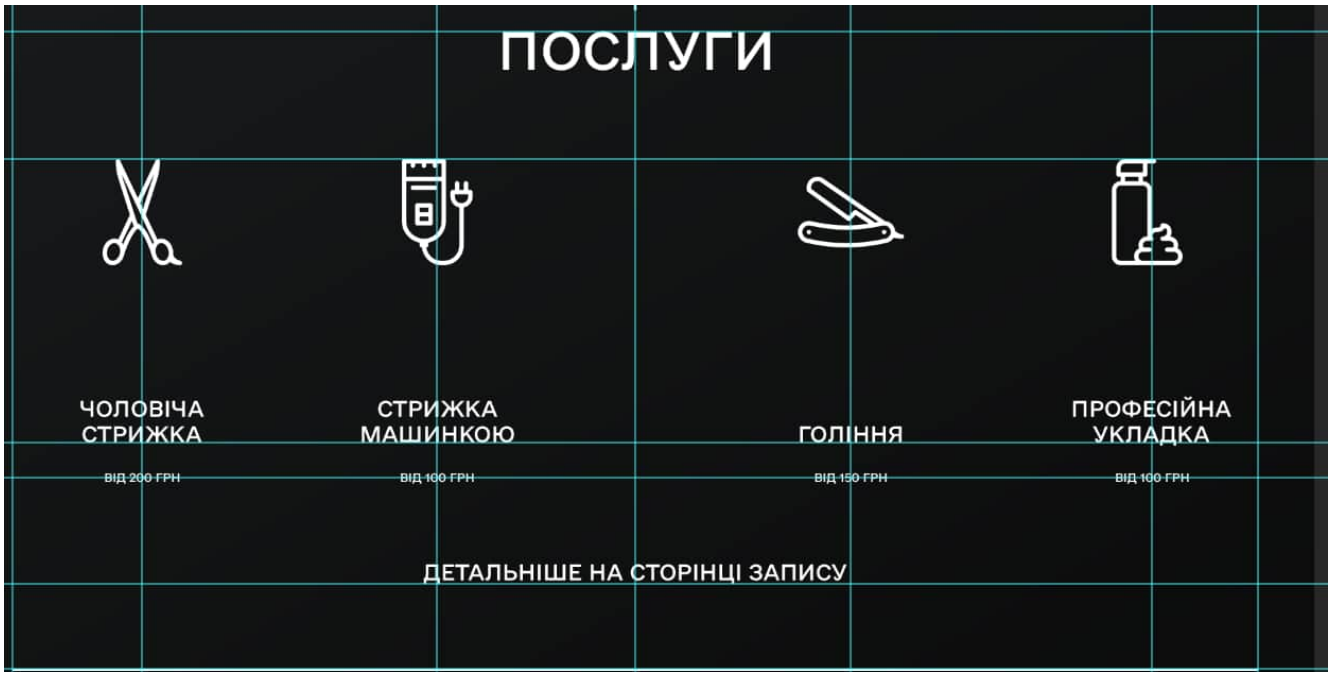

Рисунок 3.4

Секція «Галерея» (див. Рисунок 3.5). У цій частині вебсайту користувач зможе переглядати фото барбершопу, колективу і робочого процесу. Все це повинно бути реалізоване у вигляді якогось динамічного об’єкту з елементами управління, тобто користувач зможе переключатись між фото.

Рисунок 3.5

Остання секція дизайну – «Контакти» (див. Рисунок 3.6). Ця секція буде містити в собі інформацію про соціальні мережі, номери телефонів, адресу, а також карту Google Maps, яка буде відображати геолокацію барбершопу.

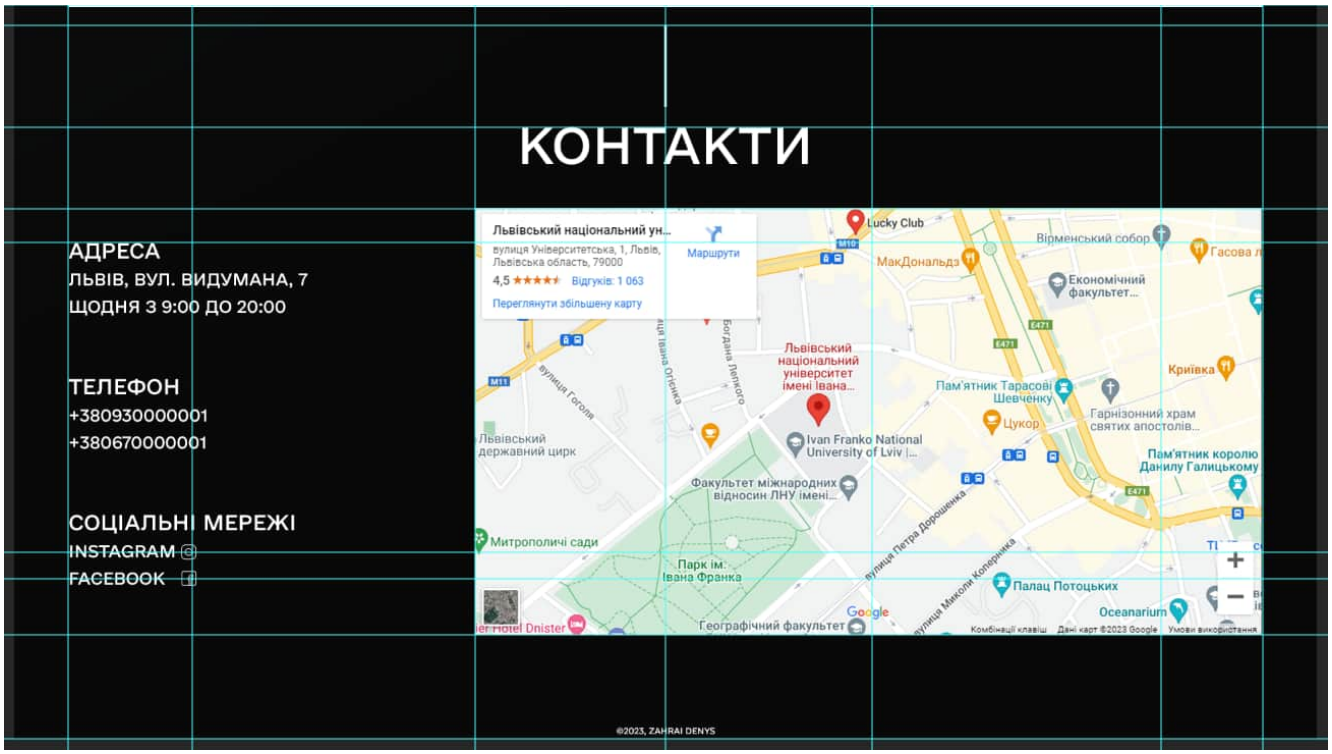

Рисунок 3.6

На цьому робота над дизайном завершилась і у висновку отримано проект, який містить в собі вигляд усіх секцій лендингу. Також Adobe Photoshop дозволяє копіювати стилі певних елементів у вигляді CSS коду – це дозволить пришвидшити роботу у наступній фазі розробки. Варто зазначити, що отриманий макет не буде демонструвати кінцевий вигляд вебсторінки.

4. НАПИСАННЯ «FRONT END» ЧАСТИНИ

4.1. Підготовка проекту

Маючи створений макет дизайну – можна переходити до наступної фази розробки, а саме написання клієнтської частини. Під час роботи використовувався засіб для редагування вихідного коду Visual Studio Code. Створення пустого проекту і його налаштування відбувається за лічені хвилини. React дозволяє розробнику зосереджуватись на написанні коду, тому створення проекту – це одна команда в командному рядку Windows, детальніше про це та інші аспекти можна прочитати в документації [10].

Першим ділом була створена папка public (див. Рисунок 4.1) в якій зберігаються публічні файли: «index.html» та тека assets з різними зображенням і значками, які використовуються в макеті. Також налаштований html-файл з доданими до нього шрифтами, які будуть використовуватись на вебсайті (див. Рисунок 4.2).

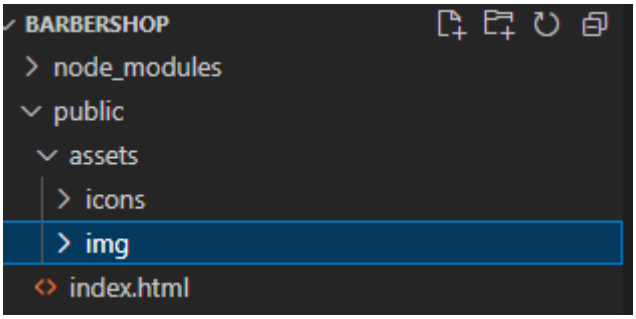

Рисунок 4.1

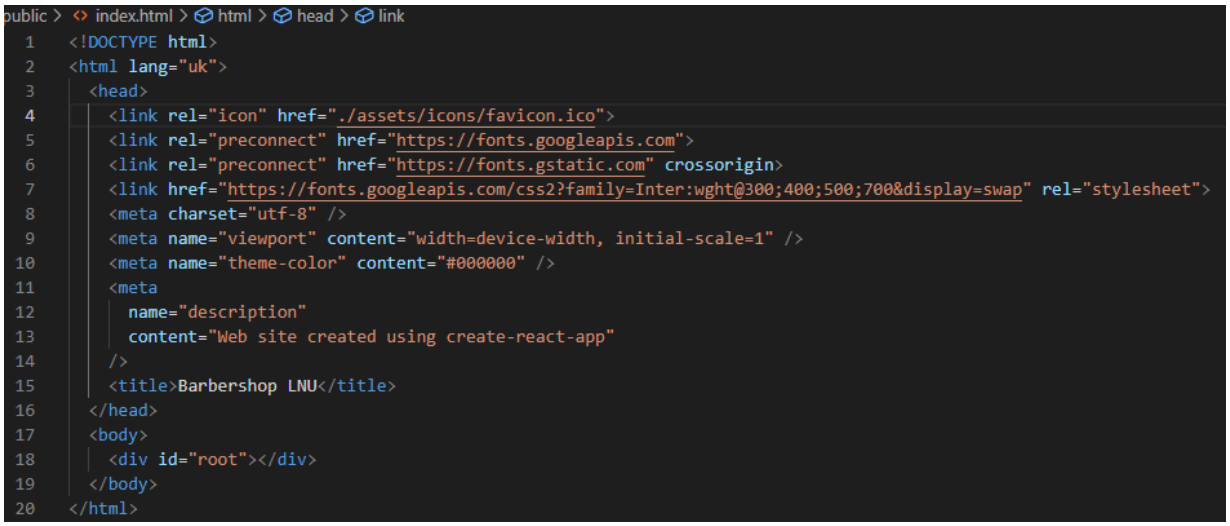

Рисунок 4.2

Далі створена наступна папка src, де буде зберігатись основний JavaScript код проекту, файли стилів SCSS та деякі інші зображення (див. Рисунок 4.3). В файлі «style.scss» будуть записуватись загальні стилі вебпроекту, які використовуватимуться у всіх компонентах, щоб не повторювати їх щоразу. Також у ньому можна визначити спеціальні стилі, щоб знімати стандартну стилізацію певних тегів, наприклад кнопки чи поля для вводу [8].

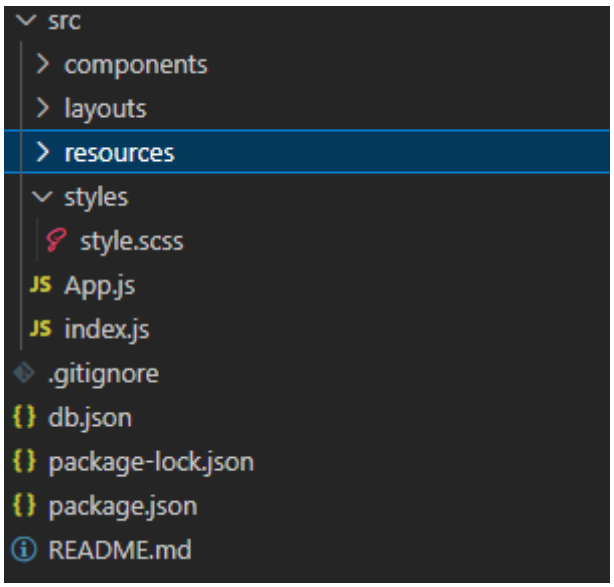

Рисунок 4.3

В інших файлах або зберігаються налаштування проекту і підключені бібліотеки, або вони пригодяться пізніше.

Файл «index.js» буде створювати на основі нашого коду HTML документ, який буде відображатись користувачу. Для цього до нього підключено інший файл – «App.js» (див. Рисунок 4.4).

```
src > JS index.js > ...
1   import React from 'react';
2   import ReactDOM from 'react-dom/client';
3   import App from './App';
4
5   const root = ReactDOM.createRoot(document.getElementById('root'));
6   root.render(
7     <React.StrictMode>
8       <App />
9     </React.StrictMode>
10  );
```

Рисунок 4.4

У файлі «App.js» потрібно підключити «react-router-dom» - це спеціальна бібліотека, яка дозволить переходити між сторінками і створювати різні шляхи на вебсайті, для прикладу основний лендинг буде за шляхом http://localhost:3000/, а сторінка формування візиту вже буде на http://localhost:3000/booking , щоб це працювало потрібно завантажити бібліотеку за допомогою терміналу. Для цього використовується спеціальна команда «npm і react-router-dom», яка автоматично встановлює вказану бібліотеку. Таким самим методом будуть встановлюватись всі подальші необхідні для роботи пакети. Команди для встановлення цих бібліотек можна знайти на спеціальному ресурсі в інтернеті [13].

Сформовано основний скелет інтернет-ресурсу (див. Рисунок 4.5). Головною сторінкою вебсайту буде лендинг. Вона позначається атрибутом index і в атрибут element передається тег JS компоненти «Main.js», яка буде відповідати за формування цієї сторінки. Простими словами, якщо користувач зайде на сайт, то він побачить те, що йому поверне компонента «Main.js». Також створені шляхи для формування візиту (booking) та для окремого функціоналу адміністратора барбершопу (admin, admin/bookings, …). Існує також маршрут з атрибутом path=”*” – це спеціальний маршрут, який буде показувати користувачу, що він ввів неправильну адресу і попав на сторінку, яка не існує в даному вебпроекті.

```
43  function App() {
44    return <>
45      <Router>
46        <Routes>
47          <Route index element={<Main />} />
48          <Route path="booking" element={<Booking />} />
49          <Route path="admin">
50            <Route index element={<AdminLogin />} />
51            <Route path="bookings" element={<AdminBookings />} />
52            <Route path="barbers" element={<AdminBarbers />} />
53            <Route path="clients" element={<AdminClients />} />
54            <Route path="*" element={<div>not found</div>} />
55          </Route>
56        </Routes>
57      </Router>
58    </>
59  }
60
61  export default App;
```

Рисунок 4.5

4.2. Створення компонент за допомогою React, JavaScript та SCSS

4.2.1. Компонента «Main»

Перша компонента – «Main». Вона має містити в собі код лендинг сторінки. Для початку створюються окремі папки, щоб в подальшому не заплутатись між різними файлами. В layouts будуть зберігатись теки, які відповідають за складові частини вебсайту (див. Рисунок 4.6).

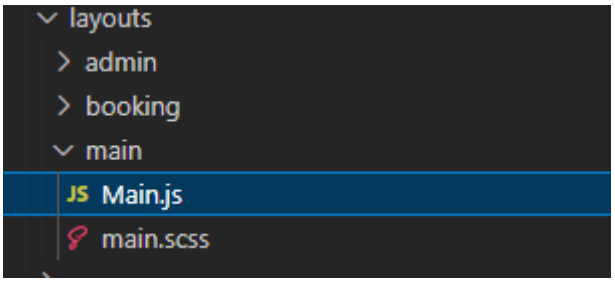

Рисунок 4.6

Створено два файли: «Main.js» та «main.scss». В «Main.js» за допомогою JavaScript описуний сценарій сторінки, а «main.scss» – це файл, який містить в собі стилі для «Main.js».

Написання лендингу розділене окремо по кожній секції (див. Рисунок 4.7).

src > layouts > main > JS Main.js > default

```
1  import Promo from "../../components/main/Promo/Promo";
2  import AboutUs from "../../components/main/AboutUs/AboutUs";
3  import Barbers from "../../components/main/Barbers/Barbers";
4  import Services from "../../components/main/Services/Services";
5  import Gallery from "../../components/main/Gallery/Gallery";
6  import Contacts from "../../components/main/Contacts/Contacts";
7  import Navbar from "../../components/main/Navbar/Navbar";
8  import MainButton from "../../components/main/MainButton/MainButton";
9  import "./main.scss";
10 import { useNavigate } from "react-router-dom";
11
12 function Main() {
13   const navigate = useNavigate()
14   return (
15     <div className="main">
16       <Navbar />
17       <MainButton value="Записатись" onClick={() => navigate('/booking')}/>
18       <Promo />
19       <AboutUs />
20       <Barbers />
21       <Services />
22       <Gallery />
23       <Contacts />
24       <footer>©2023, Zahrai Denys</footer>
25     </div>
26   );
27 }
28
29 export default Main;
```

Рисунок 4.7

Компонента «Main» буде повертати HTML-код, який складається з інших компонент. Щоб це реалізувати з перших рядків файлу потрібно імпортувати ці компоненти, вказати їх назву і шлях. Таким же способом імпортується файл стилів «main.scss».

Була створена папка components, в якій зберігаються окремі файли потрібні для тих чи інших сторінок, всі вони між собою розділені, щоб не заплутатись (див. Рисунок 4.8).

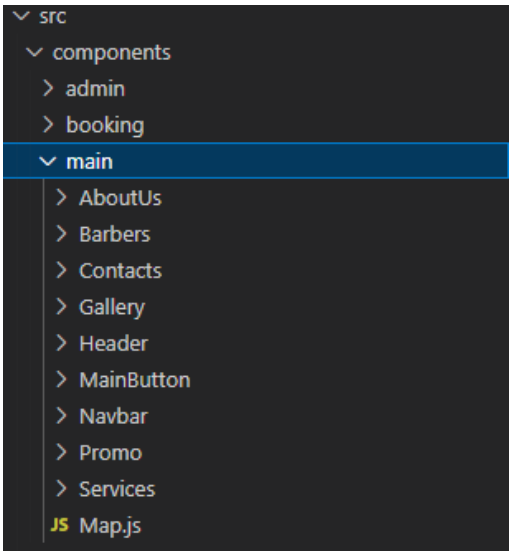

Рисунок 4.8

Компонента «Promo» відповідає за початкову секцію. У ній присутня окрема компонента «Header», яка відповідає за навігаційне меню зверху екрана і заголовок, який відвідувач бачить у центрі. В компоненті «Header» записаний код спеціальних елементів <AnchorLink>, які дозволять користувачу переходити між різними секціями. Щоб використовувати ці елементи потрібно додатково встановити їхню бібліотеку способом, який вже згадувався раніше (див. підрозділ 4.1). Також у компонентів «Header» використовується тег <img>, щоб зобразити лого барбершопу. Використовуючи команду «npm start», можна запустити проект на локальному сервері і побачити чи правильно описана дана секція (див. Рисунок 4.9).

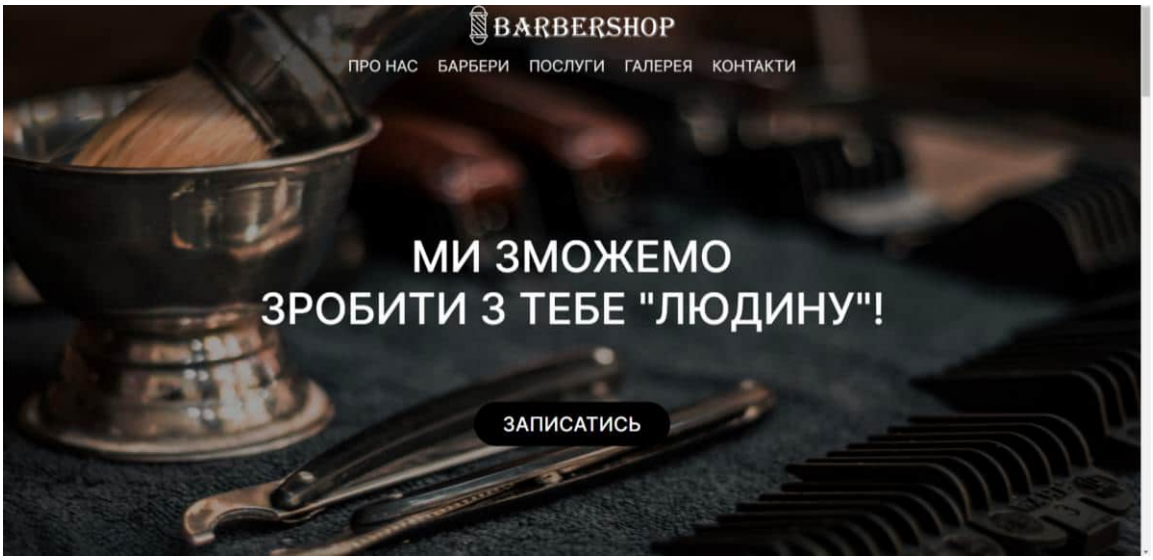

Рисунок 4.9

Кнопка «Записатись» знаходиться в іншій компоненті і вона викликається загально для всієї сторінки. Додатково визначені її стилі, щоб вона виглядала згідно створеному макету і при наведенні курсора змінювалась з чорної на білу.

Наступна секція – «AboutUs». Детальний опис цієї частини не наводиться, тому що код повторюється, але з деякими відмінностями. Тут з’являється спеціальний блок , який містить в собі білу вертикальну лінію, яка буде розділювати наші секції, кнопка «Записатись» буде змінювати положення і переходити в нижню праву частину екрану, а також з’являється спеціальна навігаційна панель.

Варто окремо згадати про цю панель, адже в ній використовуються важливі хуки React (хуки – механізми, які дозволяють працювати повністю без класів в React і дозволяють не повторювати написання коду для розв’язання загальних задач), які ще стануть у пригоді. Для початку ця панель описана в окремій компоненті «Navbar», також для неї визначені спеціальні стилі. Потрібно демонструвати цю панель тільки в тому випадку, якщо користувач догортав до секції «Про нас». Щоб це стало можливим використовується хук useEffect, який постійно перевіряє позицію сторінки, якщо вона змінюється. Коли вона відповідає певній умові, то до панелі присвоюється ім’я класу, яке містить стилі для її відображення. Також в цій компоненті використовується useState, він дозволяє

присвоювати значення в змінні за допомогою функції. Ці хуки будуть використовуватись ще не один раз.

Після написання коду секцію обов’язково потрібно перевірити її працездатність, запустивши проект (див. Рисунок 4.10).

Рисунок 4.10

Наступна секція – «Barbers». Її особливістю є те, що вона потребує створення спеціальних контейнерів в яких будуть картки барберів. Ці картки повинні розташовуватись на одній відстані один від одного і бути по центру екрана. Також потрібно застосовувати чорно-білий фільтр для зображень, якщо користувач не навів курсором на картку. У коді цієї секції часто використовується тег з різними іменами класів, щоб правильно налаштувати відображення (див. Рисунок 4.11). Також варто згадати про властивість display: flex в файлі стилів, яка дозволила налаштувати положення елементів так, як потрібно. Щоб накласти фільтр на зображення використовується filter: grayscale(100%), який при наведенні на фото курсором буде плавно змінюватись на 0%.

```
<section className='barbers' id='barbers'>
  <div className='hr'></div>
  <div className='vr'></div>
  <h2>Наші барбери</h2>
  <div className='barbers_images'>
    <div className='item_container'>
      <div className='image1'/>
      <div className='overlay'>
        <div className='barber_name'>Максим</div>
      </div>
    </div>
    <div className='item_container'>
      <div className='image2'/>
      <div className='overlay'>
        <div className='barber_name'>Денис</div>
      </div>
    </div>
    <div className='item_container'>
      <div className='image3'/>
      <div className='overlay'>
        <div className='barber_name'>Артур</div>
      </div>
    </div>
  </div>
  <div className='barbers_text'>
    Якийсь опис професійних барберів.
```

Рисунок 4.11

Отже, стилі вірно застосувались і при наведенні картинка змінюється так, як потрібно (див. Рисунок 4.12).

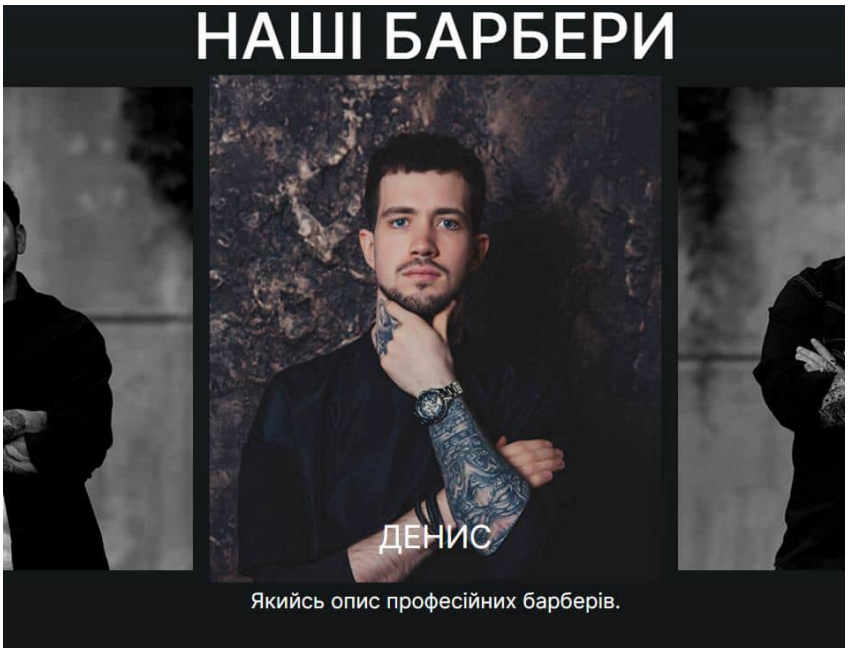

Рисунок 4.12

Секція «Services». Код цієї секції дуже схожий до попередньої, але тут не використовуються ніякі додаткові стилі при наведенні мишею. Виводиться загальна інформація про послуги в окремих картках і вирівнюється по центру.

Компонента і стилі для цієї секції були повністю описані і перевірена їхня працездатність.

Секція «Галерея». Тут використовується окрема компонента «ImageSlider», яка реалізовує функціонал вікна галереї. В React можна знайти готові бібліотеки для таких завдань, але цікаво написати це власноруч. Для початку був створений масив посилань на фото, які будуть використовуватись. Далі цей масив передається в компоненту «ImageSlider». Були написані спеціальні функції, щоб можна було переключатись між фото вперед і назад, також існує спеціальний індекс, при зміні якого і відбувається зміна зображення. Створюється масив слайдів і масив індикаторів відповідно до кількості зображень (див. Рисунок 4.13). Також при виході за межі масиву – користувач буде повертатись на позицію першого елемента.

```
return (
    <div className="slider">
        <div className="left_arrow" onClick={goToPrevious}>←</div>
        <div className="right_arrow" onClick={goToNext}>→</div>
        <div className="container_overflow">
            <div className="photo_container" style={getSlidesContainerStylesWithWidth()}>
                {slides.map((_, slideIndex) => (
                    <div key={slideIndex} style={slideImage(slideIndex)}></div>
                ))}
            </div>
        </div>
        <div className="dot_container">
            {slides.map((_, slideIndex) => (
                <div key={slideIndex} className="dot" onClick={() => goToSlide(slideIndex)}>o</div>
            ))}
        </div>
    </div>
);
```

Рисунок 4.13

Отже, вдалося написати компоненту, яка буде реалізовувати просту фотогалерею з елементами управління (стрілочками) (див. Рисунок 4.14).

Рисунок 4.14

Остання секція лендингу – «Contacts». В ній багато роботи повторювалось з минулих секцій, але також були і новинки. Наприклад, потрібно було багато використовувати тег <a>, щоб правильно оформлювати посилання і номери телефонів. Також ця секція має додаткову компоненту «Map», вона дозволяє зобразити на вебсайті Google Мапу з вказаною геолокацією. Для цього потрібно перейти на сайт Google Maps і вибрати локацію, далі в вікні «Поділитись» можна скопіювати посилання на це місце, потім це посилання вставляється в спеціальний тег <iframe> і розміщується на секції в окремому блоці, таким чином відвідувачі вебсайту зможуть бачити локацію барбершопу (див. Рисунок 4.15). У висновку отримано готовий лендинг, який надає основну інформацію користувачу, який потрапив на вебсайт.

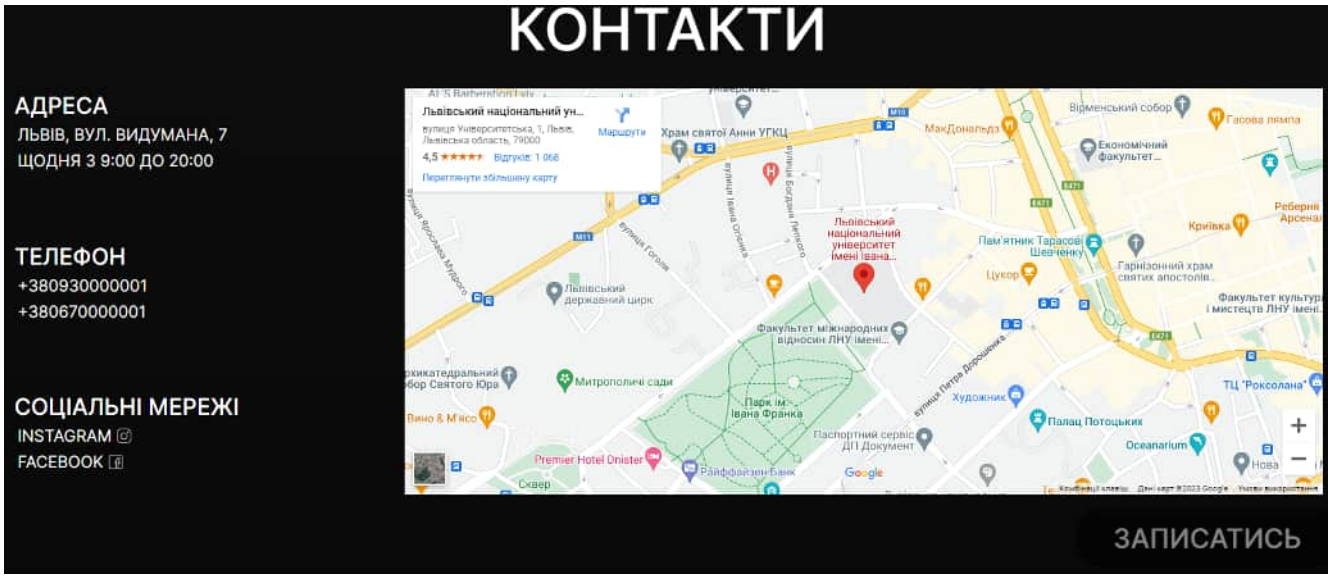

Рисунок 4.15

4.2.2. Компонента «Booking»

Наступним кроком буде написання іншої компоненти «Booking», яка вже не має жодного відношення до «Main», але на яку можна потрапити за допомогою кнопки «Записатись» (детальніше про шляхи можна дізнатись з рисунка 4.5). Ця компонента буде відповідати за сторінку формування візиту.

Користувач на сторінці формування візиту буде мати можливість обрати барбера, обрати послугу, яку він бажає, а також обрати дату й час, після цього він вказує свої контактні дані і відправляє форму на сервер. Якщо все пройшло успішно, він повинен отримати повідомлення. Щоб реалізувати ці можливості потрібно використати бібліотеку «formik», яка допоможе створити форму для заповнення і налаштувати її. Також у цій компоненті використовується бібліотека «yup», яка дозволить створити спеціальну схему для валідації даних, які ввів користувач. Форма буде заповнюватись покроково. Відповідно до кожного кроку створюється окрема форма, яка буде відповідати за вибір того чи іншого параметру, а потім це все буде формуватись в один об’єкт і передаватись на сервер (див. Рисунок 4.16). Ці форми будуть створюватись в різних компонентах в папці components. Кожна компонента також буде мати свій файл зі стилями.

```
const FormStep = (props) => {
  switch (formStep) {
    case 0:

      return <BarbersForm  {...props} />;
    case 1:

      return <ServicesForm {...props} />;
    case 2:

      return <DateTimeForm {...props} />;
    case 3:

      return <FinishForm {...props} />;

    default: return <></>
  }
}
```

Рисунок 4.16

Отже, перша форма, яка зустрічає нашого користувача – це форма вибору барбера. Вона повинна відображати картки барберів на які можна натиснути, коли користувач натискає на одну з них, то вона підсвічується, а ідентифікатор вибраного барбера записується в спеціальну змінну, яка в подальшому відправиться на сервер. Щоб імітувати дані отримані від сервера – був створений спеціальний масив з вказаними вручну елементами (див. Рисунок 4.17).

```
const barbersData = [
  { id: '1', name: 'Максим', bio: '"Працюю барбером вже 3 роки"', rank: "silver", photo: "./assets/img/first_barber.jpg" },
  { id: '2', name: 'Денис', bio: '"Зі мною можна поговорити"', rank: "gold", photo: "./assets/img/second_barber.jpg" },
  { id: '3', name: 'Артур', bio: '"Доповню ваш образ гарною зачіскою"', rank: "platinum", photo: "./assets/img/third_barber.jpg" },
];
```

Рисунок 4.17

В подальшому прийдеться лише змінити дані на ті, які отримуються від сервера по спеціальному запиту, а зараз можна працювати з цими і бачити готовий результат і вигляд. Отже, маючи масив даних, витягується кожен елемент, щоб сформувати картки для вибору барберів. Також необхідно написати функцію onClick, яка буде викликатись при виборі картки. Ці функції були описані в батьківській компоненті і передані у форму, таким чином значення буде присвоюватись в основній компоненті «Booking» без зайвих змінних. Налаштувавши стилі, створено сторінку для вибору барбера (див. Рисунок 4.18).

Рисунок 4.18

Обраний барбер підсвічується і знизу з’являється повідомлення з іменем вибраного майстра. Також додано кнопку «Продовжити», яка дозволить переходити на наступний крок і завантажувати компоненту з іншою формою. Ця кнопка буде недоступна поки користувач не зробить свій вибір. Зверху додано спеціальну панель на якій зображений логотип барбершопу, заголовок сторінки і посилання на лендинг («Повернутись»).

Наступним кроком буде вибір послуги. Знову ж таки ця форма описана в окремому файлі «ServicesForm». Важливо є те, що тут буде показуватись ціна за послугу, і для кожного барбера вона буде відрізнятись. Тому прийдеться окремо вираховувати ціну відповідно до вибраного користувачем барбера. З цим не буде проблеми, адже код проєкту зберігає його в попередній формі. До поки не використовується «back end» частина – прийдеться виконувати обчислення прямо в компоненті. Далі можна буде створити спеціальний метод і лише повертати потрібні значення.

Написано спеціальну функцію, яка множить базову ціну за послугу на коефіцієнт рангу барбера (див. Рисунок 4.19).

```
const getCoef = (price) => {
  const barberRank = values.barber.rank;
  return price * coefs[barberRank];
}
```

Рисунок 4.19

Далі ця функція використовується, щоб демонструвати актуальну ціну за вибрану послугу (див. Рисунок 4.20).

```
return (
  <div className="services_form">
    <h1>Виберіть послугу</h1>
    <div className="services_cards_container">
      {servicesData.map((service) => (
        <div key={service.id} className={getSelectedServiceClass(service.id)} onClick={() => handleSelectService(service)}>
          <img src={service.icon} alt="Service" draggable="false" className="service_icon" />
          <div>
            <h3>{service.name}</h3>
            <h4>Ціна: {getCoef(service.price)} UAH</h4>
          </div>
        </div>
      ))}
    </div>
    {values?.service?.id && (
      <div className="selected_service_container">
        <h3>Вибрана послуга:</h3>
        <p>Назва послуги: {values.service.name}. Ціна: {values.total} UAH</p>
      </div>
```

Рисунок 4.20

У висновку отримано сторінку форми, яка дозволяє вибрати послугу і дізнатись ціну за неї. Також додано кнопку «Назад», щоб користувач міг змінити вибраного барбера – в такому разі і ціни будуть динамічно змінюватись (див. Рисунок 4.21).

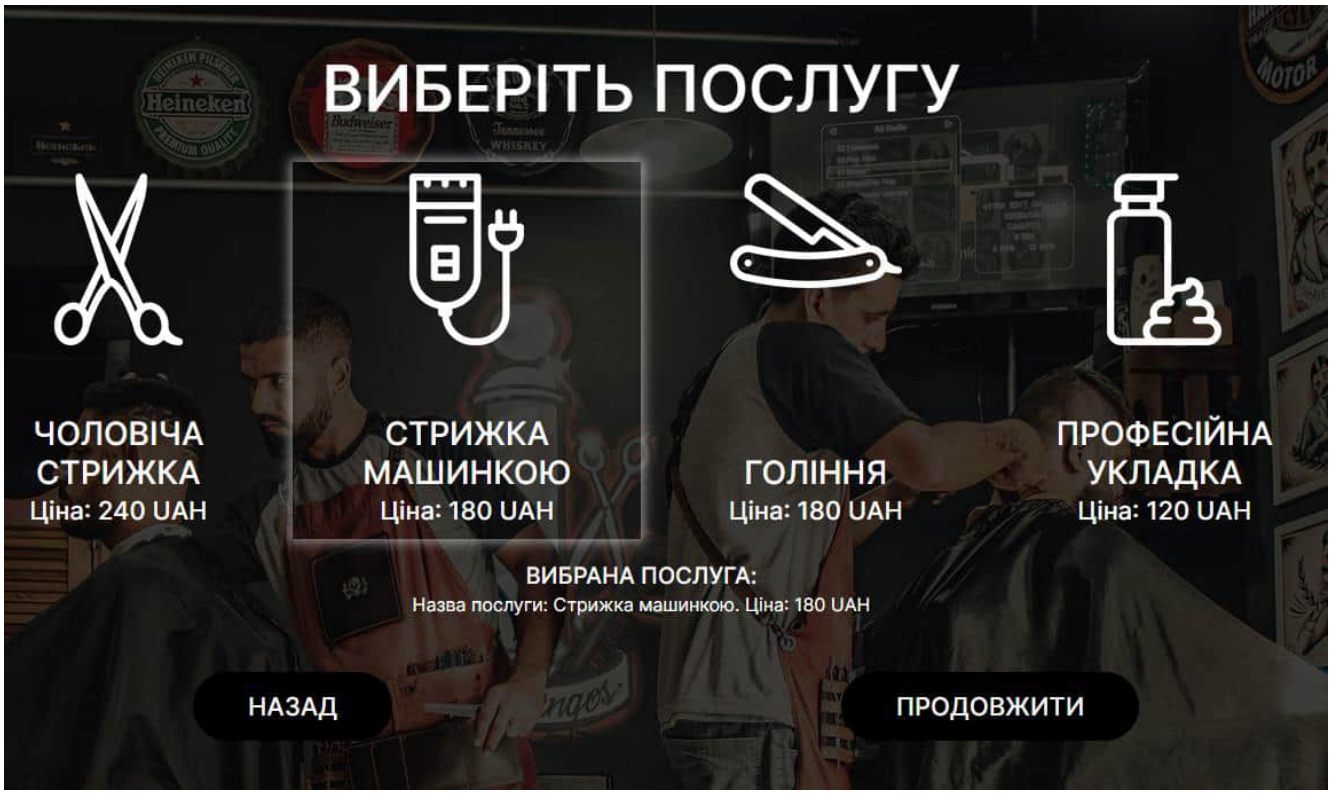

Рисунок 4.21

Одна із найважчих частин – форма вибору дати й часу. Коли користувач обрав барбера і послугу, на яку бажає записатись, то йому потрібно обрати дату і час. Для цього використовується спеціальна бібліотека, яка дозволяє викликати календар і налаштувати його під свої потреби. Назва цієї бібліотеки – «react-datepicker». Були створені окремі компоненти для вибору дати і часу, тому що, коли користувач обирає дату – на основі його вибору потрібно дізнатись, які вільні години є для цієї дати. Відповідно прийдеться формувати спеціальний запит, відправляти його на сервер, а він буде вертати години, які зайняті для цього барбера у цей день. Дальше з масиву усіх можливих годин потрібно виключати ті, на які уже є запис, і надавати користувачу лише доступні години. Також написано спеціальну функцію, яка перевіряє чи людина обрала сьогоднішню дату, щоб виключати години, які вже пройшли. Для прикладу, користувач не зможе записатись на 12:00, якщо на годиннику уже 12:40 і тому подібні ситуації. На рисунку 4.22 зображено, як виглядає функція, яка обраховує можливі години для запису.

```
const availableHours = hours.filter(hour => {
    if (isDateToday(values?.date)){
    const hourTime = new Date();
    const [hourValue, minuteValue] = hour.value.split(':');
    hourTime.setHours(hourValue);
    hourTime.setMinutes(minuteValue);
    return !bookedHours.includes(hour.value) && hourTime > thresholdTime;
    } else {
        return !bookedHours.includes(hour.value);
    }
  });
```

Рисунок 4.22

Щоб правильно зберігати дати – використовується бібліотека «moment», яка надає можливість форматувати записи з датою. Для створення випадаючого списку використовується бібліотека «react-select». Також написані файли стилів, які налаштовують об’єкти вибору дати і часу відповідно до дизайну. Якщо користувач обере час, а потім змінить дату, то йому прийдеться знову обирати час, щоб запобігти можливості вибрати зайнятий час на цю дату. Також, якщо користувач повернеться до форми вибору барбера і змінить його, то вибір дати зіб’ється, це зроблено для того, щоб запобігти вибору вже зайнятої дати й часу. Отже, на рисунку 4.23 зображений фінальний вигляд цієї форми.

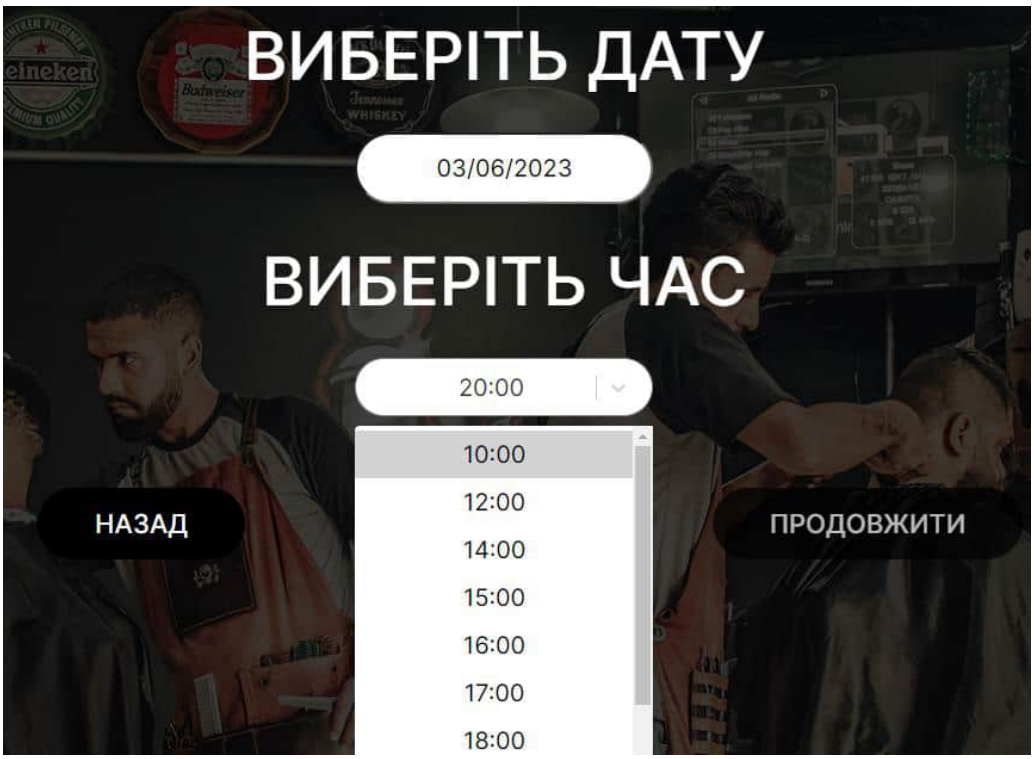

Рисунок 4.23

Останньою формою, яка була написана, є форма «FinishForm». Основне її завдання – це отримання імені користувача, номеру телефону і імейлу. Також ця форма буде демонструвати вибрані позиції клієнту. До полів цієї форми буде застосовуватись валідації. Для цього було написано спеціальну схему (див. Рисунок 4.24).

```
const phoneRegExp = /^((\\+[1-9]{1,4}[ \\-]*)|(\\([0-9]{2,3}\\)[ \\-]*)|([0-9]{2,4})[ \\-]*)*

const validationSchema = Yup.object().shape({
  name: Yup.string()
    .min(2, 'Ім\'я занадто коротке!')
    .max(50, 'Ім\'я занадто довге!')
    .required('*Обов\'язкове поле'),
  phone: Yup.string().matches(phoneRegExp, 'Введіть коректний номер телефону (без +38)')
    .min(10, 'Введіть коректний номер телефону (без +38)')
    .max(10, 'Введіть коректний номер телефону (без +38)')
    .required('*Обов\'язкове поле'),
  email: Yup.string().email('Введіть коректний email').required('*Обов\'язкове поле'),
});
```

Рисунок 4.24

Ця схема дозволить сказати користувачу, що всі поля обов’язкові до заповнення, а також, якщо він введе дані з помилками – вона постарається виявити їх і повідомити про це (див. Рисунок 4.25). Кнопка «Сформувати» буде відправляти зібрані дані на сервер і повертати якусь відповідь. Якщо запис користувача буде сформовано, то з’явиться спеціальне повідомлення.

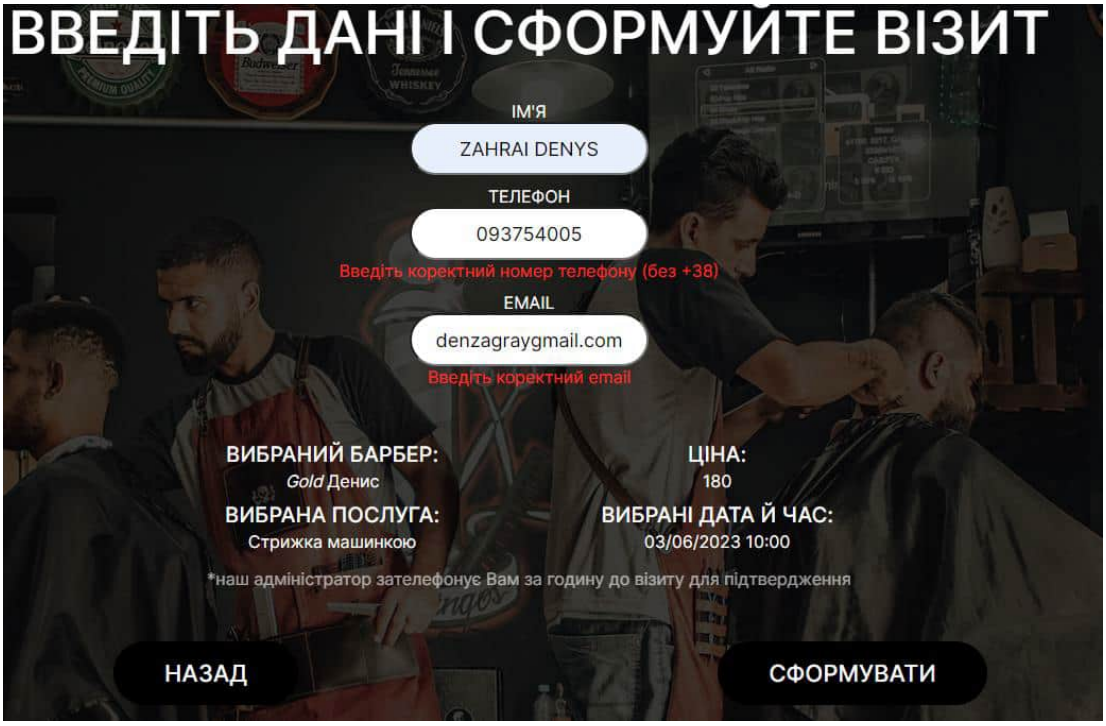

Рисунок 4.25

Дані приводяться в той вигляд, який потребує сервер (див. Рисунок 4.26), і тільки після цього відправляються.

```
const handleSubmit = async (values) => {
  const BookingData = {
    barberId: values.barber.id,
    serviceId: values.service.id,
    date: values.date,
    time: values.time,
    totalPrice: values.total,
    client: {
      name: values.name,
      phone: values.phone,
      email: values.email
    },
    status: "unconfirmed"
  }
```

Рисунок 4.26

Для реалізації відповіді від сервера використовується бібліотека «react-toastify». Якщо щось піде не так – з'явиться повідомлення з помилкою (див. Рисунок 4.27). В іншому випадку – користувач отримає позитивну відповідь і його перенаправить на головну сторінку через короткий проміжок часу.

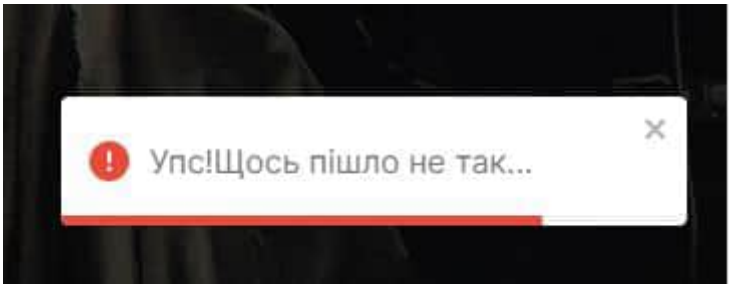

Рисунок 4.27

На цьому розробка сторінки формування запису закінчується і можна переходити до останньої частини «front end'у».

4.2.3. Компоненти «Admin»

Потрібно створити робочу зону для адміністратора, таку, в яку можна потрапити лише після авторизації. Ідея в тому, що адміністратор повинен знаходитись в барбершопі за своїм робочим місцем, він буде здатний бачити усі записи за вказану дату, до якого з барберів цей запис і дані клієнта, який записаний. Також він буде мати змогу скасовувати чи підтверджувати нові записи. Маючи номер телефона клієнта – це можна зробити за допомогою дзвінка.

Знову ж таки створюються нові компоненти в папці layouts – «AdminLogin», «AdminBarbers», «AdminClients» та «AdminBookings». «AdminLogin» – буде відповідати за авторизацію, а інші за відображення таблиць з якими зможе працювати адміністратор. На рисунку 4.28 зображений вигляд сторінки для авторизації адміністратора.

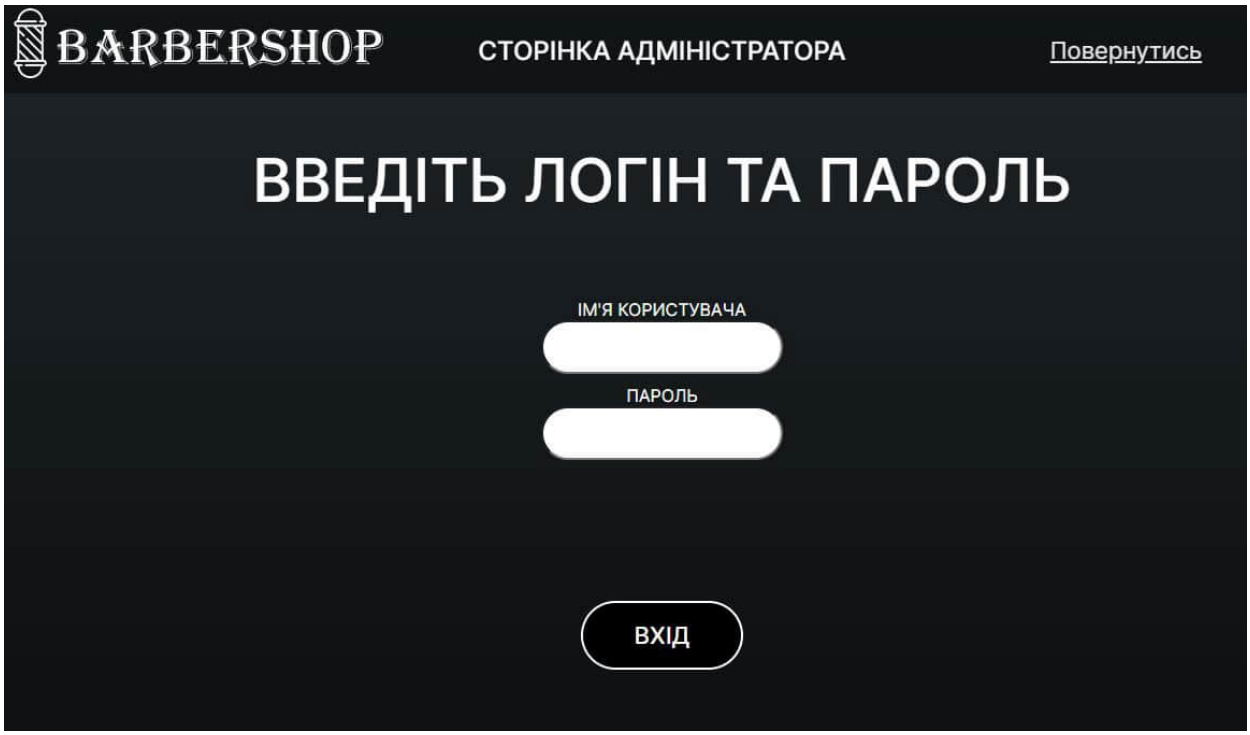

Рисунок 4.28

Тут не використовувалось нічого нового. Ця форма реалізована таким самим чином, як і форми з компоненти «Booking» (див. пункт 4.2.2). Якщо адміністратор вірно ввів свої дані, то сервер створить для нього спеціальний токен, який буде ключем до його робочої зони.

Після того, як адмін пройшов верифікацію і авторизувався, йому доступні три таблиці: записи, де він може міняти статус і переглядати записи за вказану дату, таблиця барберів, де можна додавати барбера і таблиця клієнтів, де можна переглянути список усіх клієнтів. При створенні цих компонент використовувалась бібліотека «react-table». Також можна натискати на рядки таблиці, щоб вибрати відповідний запис. Якщо натиснути за її межами, то вибір скасується. Після того, як адміністратор вибрав якийсь рядок таблиці – йому відкривається доступ до функцій підтвердження чи скасування. В таблиці барберів є кнопка «Додати», яка викликає спеціальне модальне вікно, де потрібно заповнити форму з даними нового барбера. Ось знімки екрана робочої зони адміністратора (див. Рисунок 4.29 – 4.31):

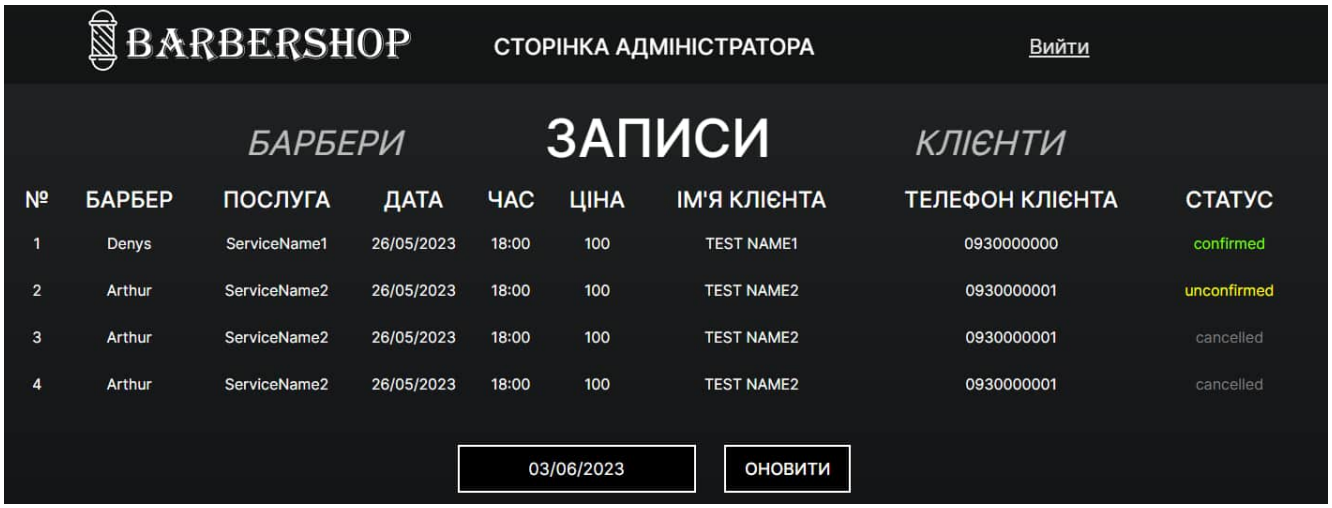

Рисунок 4.29

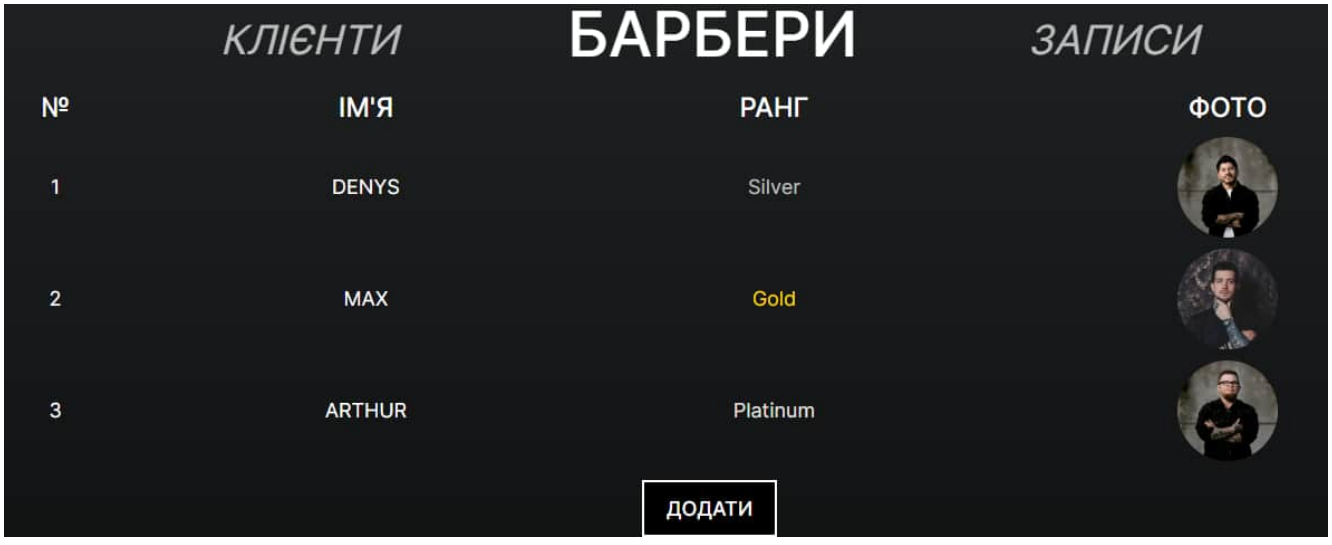

Рисунок 4.30

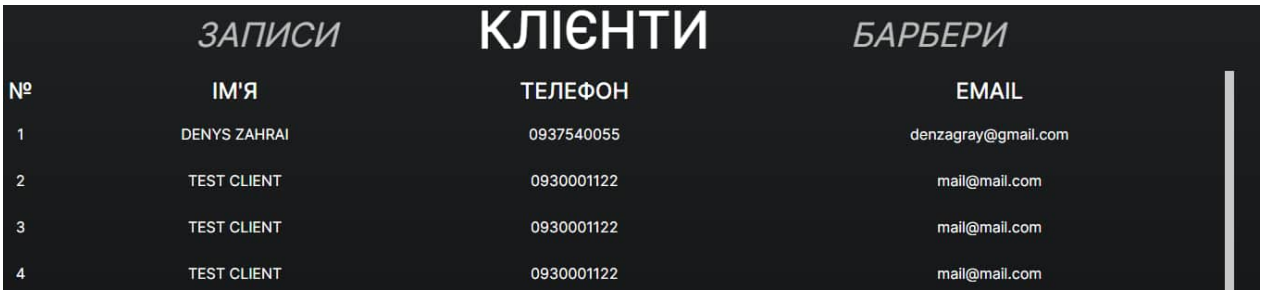

Рисунок 4.31

Підбиваючи висновки цього розділу, було розроблено «front end» частину проєкту, яку потрібно підключити запитами до сервера. Реалізовані компоненти «Main», «Booking» та «Admin». Після цього можна переходити до етапу написання коду «back end» частини.

5. НАПИСАННЯ «BACK END» ЧАСТИНИ

Для написання серверної частини використовувалось середовище розробки JetBrains Rider. Також були підготовані всі необхідні файли для коректної роботи.

5.1. Архітектура сервера

Розробляючи проект сервера вебсайту, було вирішено дотримуватись архітектури «Onion» («Цибулина»). Вона є одним із шаблонів архітектури програмного забезпечення для розробки додатків мовою C#. Така архітектура надає своєрідний підхід до розміщення і організації різних компонентів. Вона характеризується високою модульністю, зв’язуванням з низькою залежністю та легкою заміною окремих компонентів, якщо це потрібно [14].

Ця архітектура має наступні шари:

- Доменний шар (Domain Layer): Це сердце проекту, тут містяться моделі даних, правила бізнес-логіки та інші компоненти, які відповідають за реалізацію основного функціоналу програми. Цей шар повинен бути, наскільки це можливо, незалежним від інших платформ розробки чи бібліотек.
- Інфраструктурний шар (Infrastructure Layer): Цей шар займається реалізацією деталей, які використовуються для взаємодії додатку з зовнішніми системами. Він включає компоненти, такі як бази даних, репозиторії, робота з файлами і інші зовнішні служби. Цей шар може використовувати залежності від інших бібліотек та фреймворків.
- Інтерфейсний шар (Interface Layer): Цей шар виступає в ролі точки входу в додаток та забезпечує взаємодію між користувачем або іншими системами. Він може бути представлений у вигляді веб-інтерфейсу, API або інших інтерфейсів, які взаємодіють з користувачем. Цей шар використовує сервіси додатку з доменного шару для виконання запитів і відображення результатів.
- Шар логіки додатку (Application Layer): Цей шар використовується для координації взаємодії між інтерфейсним та домменим шаром. Він включає в себе сервіси застосунку, які обробляють запити, передають їх до доменного

шару для обробки та повертають результати до інтерфейсного шару. Цей шар також може включати мапери даних для перетворення об'єктів між доменним та інфраструктурними шарами.

Головною ідеєю архітектури «Цибулина» є забезпечення легкої зміни окремих компонент програми, якщо це потрібно. Вона дозволяє змінювати функціонал нашої програми, мінімізуючи вплив на інші частини системи.

5.2. Доменний рівень

Згідно з переліком шарів, який поданий в попередньому підрозділі, розпочалась робота над доменним рівнем на якому потрібно визначити бізнес-логіку. Першим ділом було описано усі сутності проекту в окремих файлах (див. Рисунок 5.1).

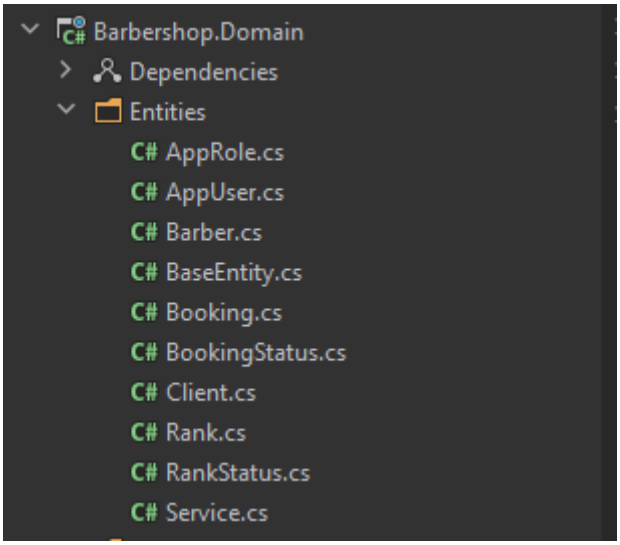

Рисунок 5.1

Для прикладу розглянемо файл «Barber.cs» (див. Рисунок 5.2). У цьому файлі створений клас «Барбер», який наслідує базовий клас, що містить в собі лише єдине поле – Id, відповідно всі класи, які наслідують базовий – будуть мати унікальний ідентифікатор за замовчуванням. Також у класі «Барбер» присутні поля рядкового типу для зберігання імені, адреси фото та опису. Поле «RankId» відповідає за зовнішній ключ і буде вказувати на рядок з таблиці рангів, який відповідає барберу.

```
namespace Barbershop.Domain.Entities;

public class Barber : BaseEntity
{
    public string FirstName { get; set; } = null!;

    public string PhotoUrl { get; set; } = null!;

    public string Description { get; set; } = null!;

    public Guid RankId { get; set; }

    //Navigation props

    public Rank? Rank { get; set; }
}
```

Рисунок 5.2

Також на цьому рівні створені спеціальні файли для обробки помилок, щоб в подальшому при підключенні клієнту до сервера було легше зрозуміти чим викликаний збій системи (див. Рисунок 5.3).

```
tects naming conventions in opened solutions and updates settings accordingly. Click 'Configure
using Barbershop.Domain.Interfaces;

namespace Barbershop.Domain.Exceptions;

public class ValidationException : Exception, IAppException
{
    public ValidationException(IEnumerable<string> errors)
    {
        Errors = errors;
    }

    public int StatusCode => 400;

    public IEnumerable<string> Errors { get; private set; }
}
```

Рисунок 5.3

5.3. Інфраструктурний рівень

На цьому рівні визначені зв’язки між сутностями в базі даних. Повернемось до описаного класу «Барбер». На доменному рівні ми вказали, що він повинен мати зовнішній ключ від таблиці рангів, але ми жодним чином не зазначили, який саме зв’язок між цими двома сутностями, тому потрібно зробити це тут.

На рисунку 5.4 бачимо, що створюватись барбери будуть в таблицю з назвою «Barbers» і далі вказані зв'язки. Барбер повинен мати один ранг, але ранги можуть мати багато барберів (один до багатьох), також вказано, що «RankId» в таблиці «Barbers» буде зовнішнім ключем. Далі ініціалізовані певні стартові дані для тестування.

```
namespace Barbershop.Infrastructure.Configurations;

public class BarberConfiguration : IEntityTypeConfiguration<Barber>
{
    public void Configure(EntityTypeBuilder<Barber> builder)
    {
        builder.ToTable("Barbers");

        builder.HasOne(navigationExpression: x :Barber => x.Rank) // ReferenceNavigationBuilder<Barber,Rank>
            .WithMany(navigationExpression: x :Rank => x.Barbers)
            .HasForeignKey(x :Barber => x.RankId);

        builder.HasData(new[]
        {
            new Barber()
            {
                Id = Guid.NewGuid(),
                FirstName = "Максим",
                PhotoUrl = "http://localhost:3000/assets/img/first_barber.jpg",
                Description = "\"Працюю барбером вже 3 роки\"",
                RankId = Guid.Parse("17ff6de9-52d0-4eae-94a1-00e18a51939b")
            },
            new Barber()
            {
                Id = Guid.NewGuid(),
                FirstName = "Денис",
                PhotoUrl = "http://localhost:3000/assets/img/second_barber.jpg",
                Description = "\"Зі мною можна поговорити\"",
                RankId = Guid.Parse("1c35ff11-df37-4d9c-9077-7ebf6a0d52fb")
            },
```

Рисунок 5.4

В інших файлах сутностей в загальному код повторюється. Варто згадати про файл «DatabaseContext.cs», який є важливою складовою інфраструктурного шару (див. Рисунок 5.5).

Рисунок 5.5

Цей клас використовується для взаємодії з базою даних, включаючи виконання запитів та збереження даних, а також для механізму аутентифікації та авторизації з використанням Identity Framework в контексті ASP.NET Core.

5.4. Інтерфейсний рівень

Цей рівень повинен реалізовувати точки входу в додаток. Він був реалізований у вигляді веб-інтерфейсу, який буде обробляти HTTP-запити. Для цього були написані спеціальні контролери (див. Рисунок 5.6).

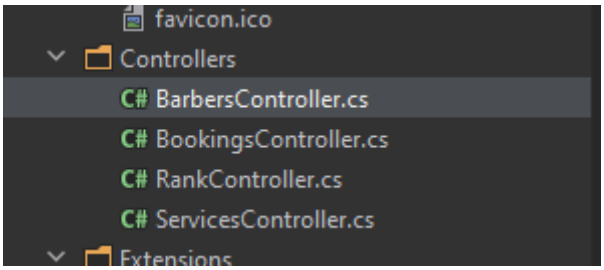

Рисунок 5.6

Ці контролери взаємодіять з клієнтом і обробляють різні HTTP-запити, такі як GET, POST, PATCH, DELETE. Кожен метод контролера може мати атрибути маршрутизації, які вказують на URL-шляхи за якими доступні певні дії. Наприклад, щоб отримати таблицю барберів, потрібно буде відправити запит на шлях, який зображено на рисунку 5.7:

```
[ApiController]
[Route( template: "api/v1/barbers")]
[Route( template: "api/barbers")]
public class BarbersController : ControllerBase
{
```

Рисунок 5.7

Також на цьому рівні реалізовані певні класи валідаторів, які перевіряють чи отримані моделі даних із запитів відповідають певним вимогам. На рисунку 5.8 зображені правила, які визначають, що модель даних «CreateBarberVm» (спеціальна модель для створення барбера) повина мати поле з іменем, яке не має бути пустим, такі ж вимоги до поля з описом, передана модель повинна мати файл, який є зображенням і також обов’язково повинен бути ідентифікатор рангу барбера.

```
public class CreateBarberValidator : AbstractValidator<CreateBarberVm>
{
    public CreateBarberValidator()
    {
        RuleFor( expression: x :CreateBarberVm => x.FirstName) // IRuleBuilderInitial<CreateBarberVm,string>
            .NotNull()
            .NotEmpty();

        RuleFor( expression: x :CreateBarberVm => x.Description) // IRuleBuilderInitial<CreateBarberVm,string>
            .NotNull()
            .NotEmpty();

        RuleFor( expression: x :CreateBarberVm => x.Photo) // IRuleBuilderInitial<CreateBarberVm,IFormFile>
            .NotNull()
            .NotEmpty()
            .Must(IsImageFile);

        RuleFor( expression: x :CreateBarberVm => x.RankId) // IRuleBuilderInitial<CreateBarberVm,Guid>
            .NotNull()
            .NotEmpty();
    }
```

Рисунок 5.8

На цьому рівні також присутній спеціальний файл з налаштуваннями сервера, а саме з його URL-шляхом і портом, який буде використовуватись (див. Рисунок 5.9).

Рисунок 5.9

5.5. Рівень логіки додатку

У цьому рівні визначені інтерфейси програми та деякі додаткові сервіси. Тут описані моделі даних, які ми можем отримати в запиті з клієнта і навпаки, моделі, які сервер може відправити клієнту. Розберемо код інтерфейсу «IBookingService» (див. Рисунок 5.10).

Рисунок 5.10

У цьому інтерфейсі присутні три методи, кожен з яких відповідає за своє завдання. GetAllAsync() – дозволяє клієнту отримувати усі записи з таблиці відповідно до певної моделі «BookingVm». GetBusyTimes() – спеціальний метод, який отримує дату і ідентифікатор барбера, і вже на основі цих даних повертає записи, які відфільтрувались. Також тут присутній метод CreateAsync(), який отримує певну модель даних і на основі цієї моделі створює новий запис у таблиці.

На рисунку 5.11 можемо бачити вигляд класу моделі даних, яка потрібна для створення запису. Отже, щоб створити запис, нам необхідно мати дату й час, ідентифікатор барбера, ідентифікатор послуги та об’єкт «Клієнт», який або створиться в базі даних, або на його місце присвоїться вже існуючий ідентифікатор.

Рисунок 5.11

Те, як працюють ті чи інші методи описано в окремій папці – Services (див. Рисунок 5.12).

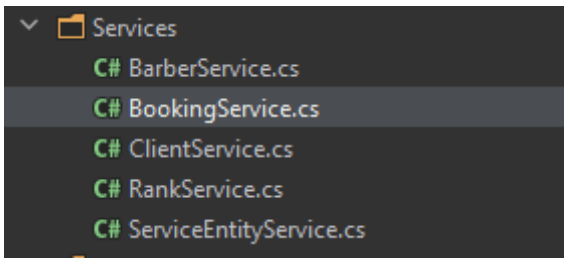

Рисунок 5.12

Розглянемо метод, який вже згадувався раніше, CreateAsync() для записів (див. Рисунок 5.13). У ньому в окремі змінні записуються ідентифікатори барбера та послуги. Створюється змінна сутності за допомогою мапера (метод, який перетворює модель в інший об’єкт). Далі перевіряється чи існує клієнт з такими даними в базі даних, якщо ні, то створюється і передається ідентифікатор клієнта. В іншому випадку – крок створення пропуститься і просто присвоїться ідентифікатор. Ціна для запису обчислюється за допомогою метода CalculateTotalPrice, який отримує коефіцієнт рангу і базову ціну за послугу. Далі

створюється новий об'єкт в репозиторії і зміни зберігаються в базі даних, а користувач отримує відповідь у вигляді статусу «OK» (код 200).

```
public async Task<AppResponse> CreateAsync(CreateBookingVm model)
{
    var service = await GetServiceAsync(model.ServiceId);
    var barber = await GetBarberAsync(model.BarberId);

    var entity = _mapper.Map<Booking>(model);

    await CheckForClientExist(entity);

    entity.TotalPrice = CalculateTotalPrice(barber.Rank!.Coefficient, service.Price);

    await _unitOfWork.BookingRepository.CreateAsync(entity);
    await _unitOfWork.SaveAsync();

    return new AppResponse(HttpStatusCode.OK, errors: null);
}
```

Рисунок 5.13

Подібно до цього інтерфейсу написані також і інші, наприклад для класу «Барбер», «Послуга», «Клієнт» і так далі.

5.6. База даних

У цій частині розглянемо, як зберігаються дані у базі даних. Почати варто з того, що сама по собі база даних формується після міграції. Інформація про те, які існують таблиці і які зв’язки між ними, вже міститься у коді наших сутностей на доменному рівні. За допомогою спеціальної команди «add migration» в терміналі база даних створюється автоматично, якщо її не існує. Розберемо детальніше таблицю записів (див. Рисунок 5.14)

Data Output | Explain | Messages | Notifications

	Id [PK] uuid	BarberId uuid	ServiceId uuid	ClientId uuid	Time timestamp with time zone	Status integer	TotalPrice numeric
1	262e7cc6-31...	960e8c7d-9a...	8991cc89-80...	658035ae-5...	2023-06-07 11:00:00+00	2	240.00
2	7b29ae69-eb...	9bab005d-e...	8991cc89-80...	c5fe13a4-2...	2023-06-07 15:00:00+00	2	200.0
3	7dcbdf96-a6f...	960e8c7d-9a...	8991cc89-80...	658035ae-5...	2023-06-08 09:00:00+00	2	240.00
4	c72807b2-1b...	960e8c7d-9a...	8991cc89-80...	8fb84f46-1...	2023-06-08 08:00:00+00	1	240.00

Рисунок 5.14

Записи у цій таблиці мають унікальний ідентифікатор (колонка Id). Також тут присутні зовнішні ключі барбера, послуги і клієнта, за допомогою них виконується пошук у інших таблицях і адміністратор має змогу бачити всю інформацію про запис. Колонка Time відповідає за дату й час запису. Статус може бути від 0 до 2, де «0» – непідтверджений, «1» – підтверджений, а «2» - скасований. TotalPrice – ціна, яку повинен сплатити клієнт. Вона вираховується окремою формулою CalculateTotalPrice() (див. Рисунок 5.13).

За таким або подібним принципом організовані усі таблиці в базі даних.

5.7. Приклад запитів клієнта

У цьому підрозділі наведемо приклади, як будуть надсилатись запити на сервер з клієнта і як організовується відповідь. Основною бібліотекою, яка цим буде займатись є «axios» [15]. На рисунку 5.15 наведений код функції, яка відсилає запит на вказаний URL-шлях і очікує отримати відповідь в вигляді масиву даних, який пізніше трансформується за допомогою мапування в таку модель, з якою працює клієнт, відповідно до цього, якщо все пройде успішно, то дані присвояться в масив «barbersData», якщо виникне якась помилка, то функція обробить її і виведе в консоль для кращого розуміння.

```
const [barbersData, setBarbersData] = useState([]);

useEffect(() => {
  const fetchBarbersData = async () => {
    try {
      const response = await axios.get('https://localhost:5001/api/v1/barbers');
      const transformedData = response.data.data.map(barber => ({
        id: barber.id.toString(),
        name: barber.firstName,
        bio: barber.description,
        rank: barber.rank.status,
        photo: barber.photoUrl,
      }));
      setBarbersData(transformedData);
    } catch (error) {
      console.error('Error fetching barbers:', error);
    }
  };

  fetchBarbersData();
}, []);
```

Рисунок 5.15

Це був запит GET, який отримує дані від сервера. Тепер розглянемо запит типу POST. Для прикладу візьмемо частину коду, де дані відправляються на сервер після того, як користувач додав нового барбера (див. Рисунок 5.16).

```
const handleSubmit = async (values, { setSubmitting }) => {
    if (!values.photo) {
        setSubmitting(false);
        return;
    }
    const form = new FormData();
    form.append('FirstName', values.name);
    form.append('Description', values.bio);
    form.append('Photo', values.photo, values.photo.name);
    form.append('RankId', values.rank);
    const notifyError = () => toast.error("Упс!Щось пішло не так...")
    const notifySuccess = () => {
        toast.success("Успішно!");
        closeWindow()
    }

    await axios.post(serverUrl, form)
        .then(() => notifySuccess())
        .catch(() => notifyError())
};
```

Рисунок 5.16

Допустимо, що користувач заповнив всі дані вірно і додав фото. Після цього він натискає кнопку «Зберегти» і з тих даних, які він ввів, формується новий об’єкт за допомогою бібліотеки «form-data». Цей об'єкт передається разом із URL-шляхом в метод axios.post() і відправляється на сервер. Після цього, якщо сервер поверне помилку, то з’явиться відповідне повідомлення, в іншому випадку вікно закриється і з’явиться повідомлення про успіх. Далі в таблиці барберів з’явиться новий барбер.

На цьому роботу над цим етапом можна вважати закінченою, було створено серверну частину в вигляді веб-інтерфейсу, яка дозволяє клієнту надсилати запити на відповідні шляхи і отримувати якусь інформацію, або надсилати дані на сервер.

ВИСНОВКИ

Після завершення роботи можна дійти деяких висновків. Вдалося перевести ідею з голови на папір, створити діаграми і макети проекту, написати клієнтську і серверну частини, поєднати це все і отримати в результаті вебсайт для барбершопу. Цей проєкт надає можливість користувачу ознайомитись з основною інформацією про барбершоп і сформувати візит, вибравши барбера, послугу, дату й час. Також на сайті присутня авторизація для адміністратора, який може бачити записи по даті, змінювати їх статус, додавати барберів і переглядати список клієнтів.

Також була виконана одна із головних цілей роботи, а саме - отримання базових навичок створення вебзастосунків за допомогою сучасних технологій. Можна зазначити те, що на сьогоднішній день вони призначені максимально полегшити роботу програмісту. Існує велика кількість готових бібліотек, які часто використовувались і у нашому проєкті, і така ж кількість різної документації, яку можна знайти у вільному доступі.

Величезним плюсом вебсайту можна виділити те, що він не має меж для доопрацювання, ця система може розвиватись стільки – скільки потрібно. Наприклад, можна додати інші таблиці і функції адміністратору, можливість надавати відпустку барберам і робити їх тимчасово неактивними тощо. Але і на даному етапі – це повністю робочий вебзастосунок, який працює автономно і не потребує додаткового обслуговування.

Було продемонстровано те, як виглядає структура сучасних вебсайтів і викладено цю інформацію у роботі, щоб кожен міг більше дізнатись про те, як працюють сучасні інтернет-ресурси і спробував себе у цій сфері, яка сьогодні, без сумніву, є актуальною.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Інтернет в Україні [Електронний ресурс] : Матеріал з Вікіпедії – Режим доступу: https://uk.wikipedia.org/wiki/Інтернет_в_Україні
2. Навіщо потрібен сайт і що він дає бізнесу [Електронний ресурс] : WebTune 2023 – Режим доступу: https://webtune.com.ua/statti/web-rozrobka/navishho-potriben-sajt/
3. Діаграма прецедентів [Електронний ресурс] : Матеріал з Вікіпедії – Режим доступу: https://uk.wikipedia.org/wiki/Діаграма_прецедентів
4. Модель «сутність — зв'язок» [Електронний ресурс] : Матеріал з Вікіпедії – Режим доступу: https://uk.wikipedia.org/wiki/Модель_«сутність_—_зв%27язок»
5. Front end та back end [Електронний ресурс] : Матеріал з Вікіпедії – Режим доступу: https://uk.wikipedia.org/wiki/Front_end_та_back_end
6. Посібник користувача Photoshop [Електронний ресурс] : 2023, Adobe – Режим доступу: https://helpx.adobe.com/ua/photoshop/user-guide.html
7. Sass [Електронний ресурс] : Матеріал з Вікіпедії – Режим доступу: https://uk.wikipedia.org/wiki/Sass
8. Sass. Documentation [Електронний ресурс] : Sass 2006–2023, the Sass team, and numerous contributors – Режим доступу: https://sass-lang.com/documentation/
9. JavaScript [Електронний ресурс] : 2023 MDN contributors – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/JavaScript
10. React Docs [Електронний ресурс] : 2023 Meta Platforms, Inc – Режим доступу: https://legacy.reactjs.org/docs/getting-started.html
11. .NET documentation / ASP.NET Core documentation [Електронний ресурс] : Microsoft 2023 – Режим доступу: https://learn.microsoft.com/uk-ua/dotnet/
12. PostgreSQL. Documentation [Електронний ресурс] : 1996-2023 The PostgreSQL Global Development Group – Режим доступу: https://www.postgresql.org/docs/
13. npm Docs [Електронний ресурс] : Режим доступу: https://docs.npmjs.com/

14. The Onion Architecture [Електронний ресурс] : Jeffrey Palermo, 2008 : Режим доступу: https://jeffreypalermo.com/2008/07/the-onion-architecture-part-1/
15. Axios Docs [Електронний ресурс] : Axios : Режим доступу: https://axios-http.com/docs/intro