МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Острозька академія»
Економічний факультет
Кафедра економіко-математичного моделювання та інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття освітнього ступеня бакалавра

на тему: «**Проектування та розробка MVP версії продукту інтернет-магазину на базі Typescript/.NET»**

Виконав: студент 4 курсу, групи КН-41
першого (бакалаврського) рівня вищої освіти
спеціальності 122 Комп'ютерні науки
освітньо-професійної програми «Комп'ютерні науки»
Марчук Микола Олександрович

Керівник: Клебан Ю.В.
старший викладач кафедри ЕММІТ

Рецензент: *кандидат технічних наук, доцент, доцент кафедри прикладної математики та кібербезпеки Донецького національного університету імені Василя Стуса* ***Загоруйко Любов Василівна***

РОБОТА ДОПУЩЕНА ДО ЗАХИСТУ

Завідувач кафедри економіко-математичного моделювання та інформаційних технологій _____ (проф., д.е.н. Кривицька О.Р.)

Протокол № 11 від «30» травня 2024 р.

Острог, 2024

Міністерство освіти і науки України
Національний університет «Острозька академія»

Факультет: *економічний*

Кафедра: *економіко-математичного моделювання та інформаційних технологій*

Спеціальність: 122 Комп'ютерні науки
Освітньо-професійна програма: Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Ольга КРИВИЦЬКА
«____» _____20__ р.

ЗАВДАННЯ
на кваліфікаційну роботу студента
Марчука Миколи Олександровича

1. Тема роботи: "Проектування та розробка MVP версії продукту інтернет-магазину на базі Typescript/.NET"

Керівник роботи: Клебан Юрій Вікторович, старший викладач кафедри ЕММІТ

Затверджено наказом ректора НаУОА від 03.11.2023 р., № 98.

2. Термін здачі студентом закінченої роботи: 31 травня 2024 року.

3. Вихідні дані до роботи: Visual Studio Code, Swagger, PostgreSql, GitHub, TypeScript, .NET, C#, React, Stripe, Docker, Fly.io.

4. Перелік завдань, які належить виконати: створити інтернет магазин з клієнтської частиною на базі React та серверною на базі ASP.NET.

5. Перелік графічного матеріалу: рисунки.

6. Консультанти розділів роботи:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Клебан Ю. В.	01.12.2023	01.12.2023
2	Клебан Ю. В.	01.12.2023	01.12.2023
3	Клебан Ю. В.	01.12.2023	01.12.2023

7. Дата видачі завдання: 01.12.2023

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів	Примітка
1	Затвердження теми проєкту	до 31.10.2023 р.	
2	Постановка технічного завдання.	до 01.12.2023 р.	
3	Ознайомлення з документацією	до 10.12.2023 р.	
4	Написання розділу 1	до 01.02.2024 р.	
5	Написання розділу 2	до 01.03.2024 р.	
6	Написання розділу 3	до 01.04.2024 р.	
7	Тестування застосунку	до 20.04.2024 р.	
8	Виправлення помилок	до 01.05.2024 р.	
9	Попередній захист та перевірка на рівень унікальності кваліфікаційної роботи/проєкту	до 31.05.2024 р.	
10	Здача кваліфікаційної роботи на кафедру	31.05.2024 р.	

Студент: _____ Микола МАРЧУК

Керівник кваліфікаційної роботи: _____ Юрій КЛЕБАН

АНОТАЦІЯ
кваліфікаційної роботи
на здобуття освітнього ступеня бакалавра

Тема: *Проектування та розробка MVP версії продукту інтернет-магазину на базі Typescript/.NET*

Автор: *Марчук Микола Олександрович*

Науковий керівник: *Клебан Ю.В., старший викладач кафедри ЕММІТ*

Захищена «……»…… 2024 року.

Пояснювальна записка до кваліфікаційної роботи: *57 с., 29 рис.,18 джерел..*

Ключові слова: *клієнтська частина, додаток, електронна комерція.*

Короткий зміст праці:

Завданням кваліфікаційної роботи було проектування та розробка MVP (Minimum Viable Product) версії продукту інтернет-магазину на базі технологій TypeScript і .NET. Головними користувачами онлайн-сервісу будуть люди, які бажають здійснювати покупки через інтернет, користуючись зручним і безпечним інтерфейсом. Фронтенд частина була створена за допомогою React та бібліотеки компонентів Material-UI (MUI), написаної на TypeScript. Бекенд частина проекту була розроблена на ASP.NET Core 7.0 з використанням Entity Framework для роботи з базою даних та Identity Framework для управління користувачами та автентифікацією. Для реалізації платіжної системи було інтегровано Stripe. Деплоймент додатку був здійснений на платформі Fly.io, що забезпечило надійну та масштабовану інфраструктуру для роботи вебзастосунку.

ANNOTATION
of qualification paper
for bachelor's degree

Theme: *Design and development of the MVP version of the online store product based on Typescript/.NET.*

Author: *Mykola Marchuk*

Scientific supervisor: *Kleban Y., senior lecturer at DEMMIT*

Defensed «……»…… of 2024.

Explanatory note to the qualification work: *57 p., 29 pic., 18 sources.*

Keywords: *client side, application, e-commerce.*

Summary of the paper:

The objective of the qualification project was the design and development of an MVP (Minimum Viable Product) version of an e-commerce product based on TypeScript and .NET technologies. The primary users of the online service are people who wish to make purchases via the internet, utilizing a convenient and secure interface. The frontend part was created using React and the Material-UI (MUI) component library, written in TypeScript. The backend part of the project was developed on ASP.NET Core 7.0, utilizing Entity Framework for database operations and Identity Framework for user management and authentication. Stripe was integrated to implement the payment system. The deployment of the application was carried out on the Fly.io platform, ensuring a reliable and scalable infrastructure for the web application.

ЗМІСТ

ВСТУП

Комерція - важливий економічний процес взаємовідносин споживання в сучасному світі. Цей термін охоплює не лише процедури купівлі та продажу, але й створення ефективної системи обміну, що диктує економічні відносини та сприяє розвитку суспільства.

У цьому контексті електронна комерція виступає як ключовий каталізатор економічного росту. Дедалі більше люди переходять від звичних покупок в магазині до замовлень в інтернеті і цей тренд буде рости ще протягом багатьох років.

З ростом потреби в електронній комерції росте потреба в модернізації процесів розробки веб-систем, впровадженні та реалізації нових бізнес ідей, аналізу даних, тощо. Отже, досліджуючи застосування нових технологій для покращення процесів електронної комерції, ми навчаємося не лише адаптуватися до змін, але і активно формувати їх, розробляючи стратегії, які відповідають сучасним викликам бізнесу та визначають майбутнє економічного прогресу. Розвиваючи цей напрямок, ми створюємо фундамент для нових можливостей, трансформуючи та змінюючи обличчя сучасного бізнесу.

Актуальність теми полягає в необхідності застосування сучасних технологій для покращення таких факторів електронної комерції:

1. Зростання популярності через створення зручного, доступного, привабливого UX/UI веб-додатку.
2. Глобалізація бізнесу - можливість бізнесу розширювати ринок на інші країни без потреби фізичного перебування в тій чи іншій країні.
3. Аналіз та оптимізація бізнес-процесів - веб-системи надають багато даних поведінки користувача, які можна оптимізувати за рахунок удосконалення маркетингової стратегії.

Метою даного дослідження є проектування та розробка веб-системи, яка буде надавати інструменти зручної покупки товару для користувача та зручної, гнучкої системи з можливістю постійних змін у додатку для власника магазину.

Для досягнення мети потрібно виконати наступні задачі:

1. Проаналізувати конкурентів на ринку електронної комерції.

2. Спроектувати систему:
    a. Архітектура бази даних.
    b. На основі аналізу конкурентів вибрати основний функціонал додатку.
    c. Вибір сучасних популярних технологій для швидкого розгортання проекту.
3. Розробити систему за попереднім планом.
4. За результатами дослідження підготувати гіпотези покращення та макет продукту, що міститиме покращені рішення для розробки інтерфейсу.

Об’єктом дослідження є клієнтська та серверна частина інтернет-магазину, а **предметом** – методи та інструменти створення інтернет-магазинів.

Для того, щоб створити гарний приклад системи електронної комерції, для початку нам потрібно проаналізувати ринок, його специфіку, тренди та виявити основних конкурентів на ринку.

РОЗДІЛ 1
АНАЛІЗ РИНКУ ТА КОНКУРЕНТІВ

1.1. Постановка проблеми

У сучасному світі електронна комерція стає все більш важливою складовою глобальної економіки. Багато підприємств переходять на цифрові платформи, щоб розширити свою аудиторію та підвищити ефективність продажів. Незважаючи на це, багато малих та середніх підприємств (МСП) стикаються з труднощами при розробці ефективних і конкурентоспроможних онлайн-магазинів. Основні проблеми включають високу вартість розробки, складність інтеграції сучасних технологій та потребу в швидкому виході на ринок.

E-commerce (електронна комерція) — це використання технологій для проведення комерційних операцій в Інтернеті. Цей термін охоплює всі форми бізнесу, що здійснюються через інтернет, включаючи онлайн-магазини, платформи електронної торгівлі, цифрові маркетплейси та інші. Основною метою електронної комерції є покращення та спрощення процесу купівлі-продажу товарів і послуг, забезпечуючи ефективну та доступну платформу для бізнесу та споживачів.

В Україні електронна комерція також активно розвивається. З'являються нові українські стартапи та платформи, які пропонують інноваційні рішення для потреб онлайн-торгівлі. Одним із випадків, що збільшили актуальність електронної комерції, стала пандемія у 2019 році. Багато фізичних магазинів були змушені закритися, що призвело до різкого зростання попиту на онлайн-шопінг. Споживачі почали активніше використовувати інтернет для купівлі товарів, а підприємства — для забезпечення дистанційних продажів.

Сфера електронної комерції продовжує стрімко зростати, тому створення продукту в ній є дуже перспективним. Використання сучасних технологій, таких як Typescript та .NET, дозволяє створювати масштабовані та гнучкі платформи для електронної комерції, які можуть швидко адаптуватися до змінних потреб ринку та забезпечувати високу продуктивність і безпеку.

Щороку загальний об’єм продаж за рахунок електронної комерції зростає і на це є багато факторів, як пандемія COVID-19, покращення маркетингових стратегій та персональних пропозицій за рахунок штучного інтелекту, зростання довіри до електронних покупок за рахунок впровадження заходів безпеки та захисту конфіденційної інформації юзера, інтуїтивний та легкий UX/UI, швидкість, ефективність та надійність доставки товарів, збільшення довіри до брендів за роки їх існування.

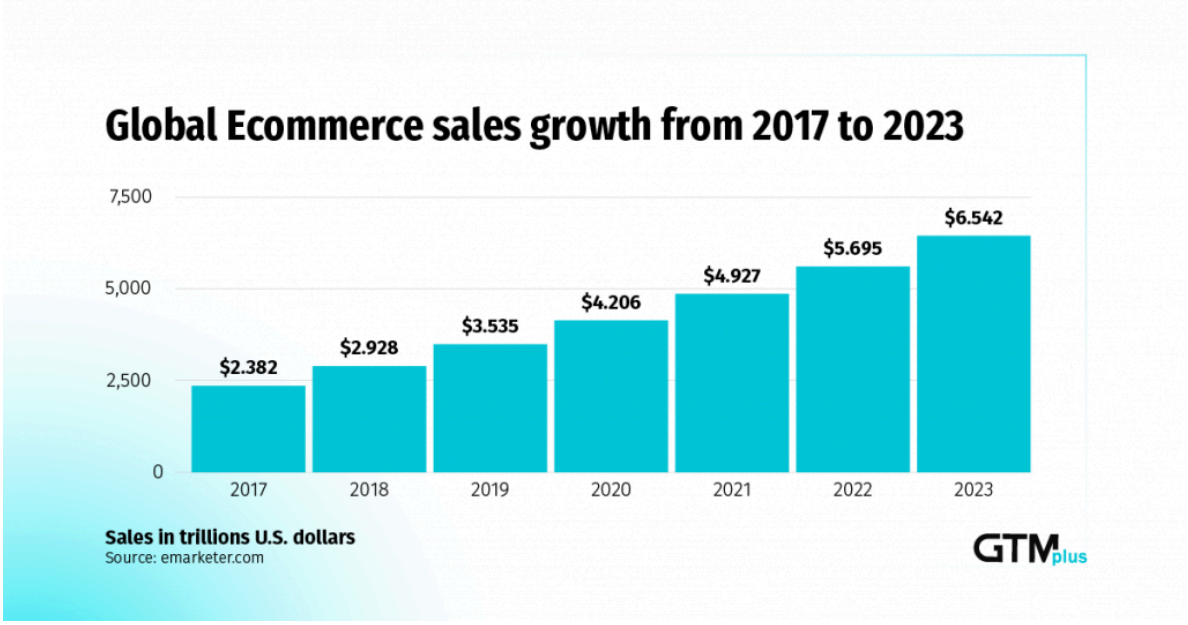

Рис. 1.1. Обсяг продаж з 2017-2023 рік

Джерело: gtm-plus.com

1.2. Аналіз конкурентів на українському ринку

На ринку електронної комерції існує багато успішних бізнесів з десятками тисяч транзакцій та найменувань товарів, основними з них є: Rozetka, Comfy, Allo, Epicenter, Auto Ria.

Rozetka (Рис. 1.2.) — це одна з найбільших і найпопулярніших платформ електронної комерції в Україні, відома своїм широким асортиментом, конкурентоспроможними цінами та високим рівнем обслуговування клієнтів. Вона пропонує зручну систему доставки та різноманітні способи оплати, що робить

процес покупок легким і зручним для користувачів. Крім того, компанія постійно впроваджує нові технологічні рішення та інновації, щоб покращити користувацький досвід і залишатися лідером на ринку електронної комерції України.

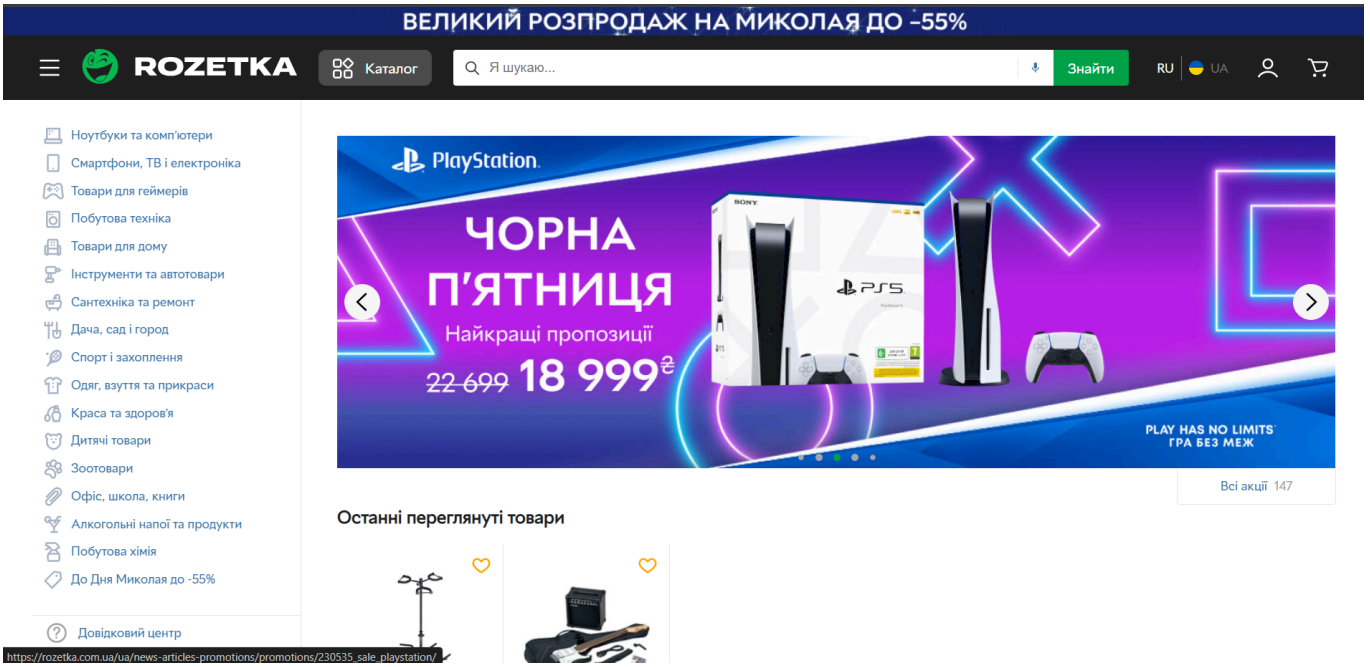

Рис. 1.2. Графічний інтерфейс програми “Rozetka”

Джерело: rozetka.com.ua

Comfy (Рис. 1.3.) — це одна з провідних мереж онлайн-магазинів, спеціалізується на продажу різноманітної побутової техніки, електроніки, комп'ютерної техніки, мобільних пристроїв та аксесуарів. Компанія також інвестує в технологічні рішення, щоб забезпечити зручність онлайн-шопінгу, включаючи зручну навігацію по сайту, різноманітні способи оплати та швидку доставку. Comfy продовжує залишатися одним із лідерів на українському ринку побутової техніки та електроніки, завдяки постійному вдосконаленню своїх сервісів і пропозицій для клієнтів.

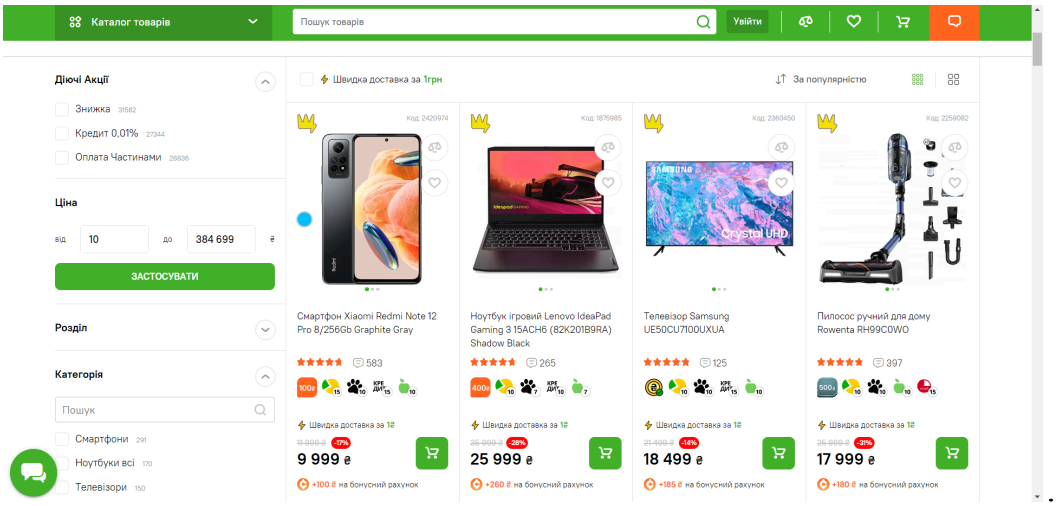

Рис. 1.3. Графічний інтерфейс програми “Comfy”

Джерело: comfy.ua

Алло (Рис. 1.4.) — це одна з найуспішніших українських компаній у сфері роздрібної торгівлі електронікою та побутовою технікою. Заснована в 1998 році, компанія виросла до масштабної мережі магазинів по всій Україні та успішно розвиває свій онлайн-магазин, пропонуючи широкий асортимент товарів від провідних світових брендів. Компанія відома своїм високим рівнем обслуговування, консультаціями фахівців, зручністю замовлень та різноманітністю способів оплати і доставки.

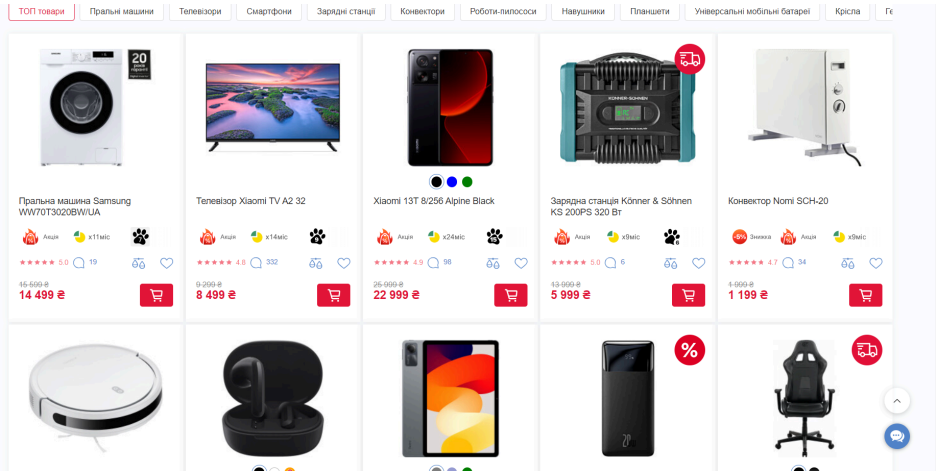

Рис. 1.4. Графічний інтерфейс програми “Allo”

Джерело: allo.ua

Епіцентр (Рис 1.5. та Рис. 1.6.) — це одна з найбільших торговельних мереж в Україні, що спеціалізується на продажу товарів для будівництва, ремонту, облаштування дому та саду. Заснована у 2003 році, компанія швидко зросла і на сьогоднішній день має широкомасштабну мережу гіпермаркетів по всій країні. Компанія відома своїм широким вибором товарів, конкурентоспроможними цінами та високим рівнем обслуговування клієнтів. Епіцентр також активно розвиває та підтримує свій онлайн-магазин, пропонуючи зручну платформу для покупок через Інтернет з різноманітними способами оплати і доставки.

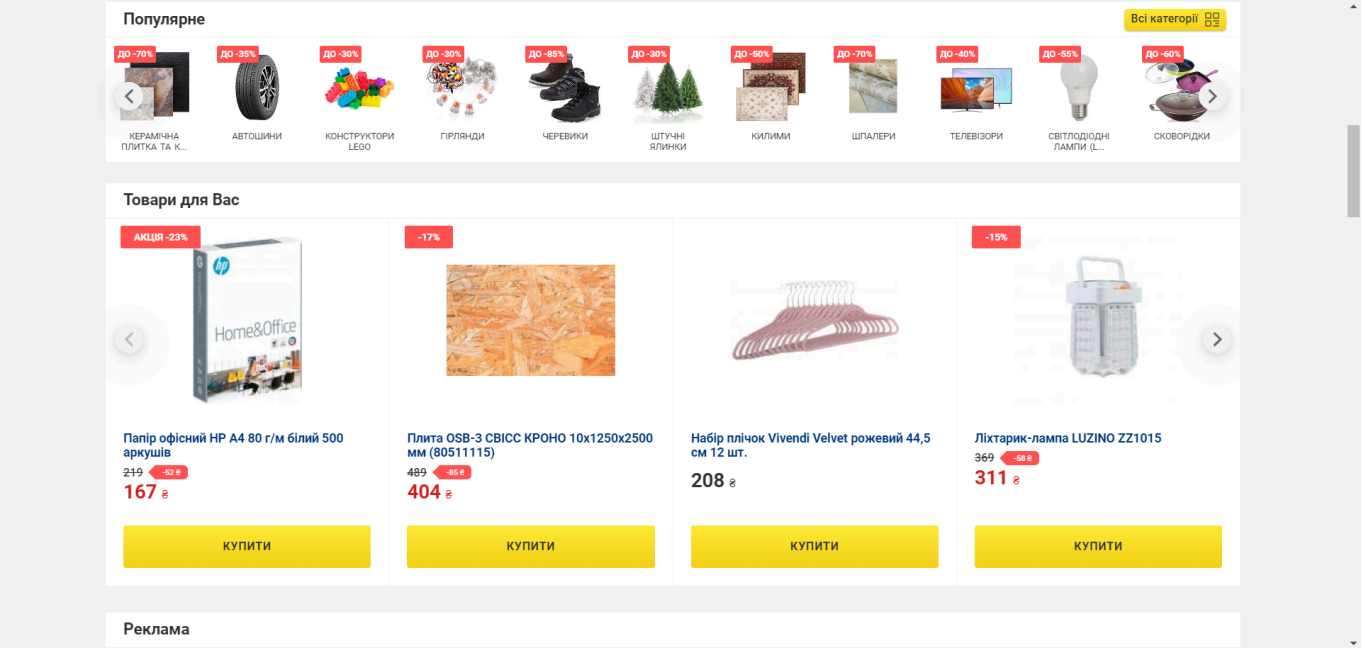

Рис. 1.5. Графічний інтерфейс програми “Epicenter”

Джерело: epicentrk.ua

Незважаючи на успішність бізнесу, користувацький досвід показує, що онлайн-магазину є куди розвиватись, враховуючи велику кількість незадоволених покупців.

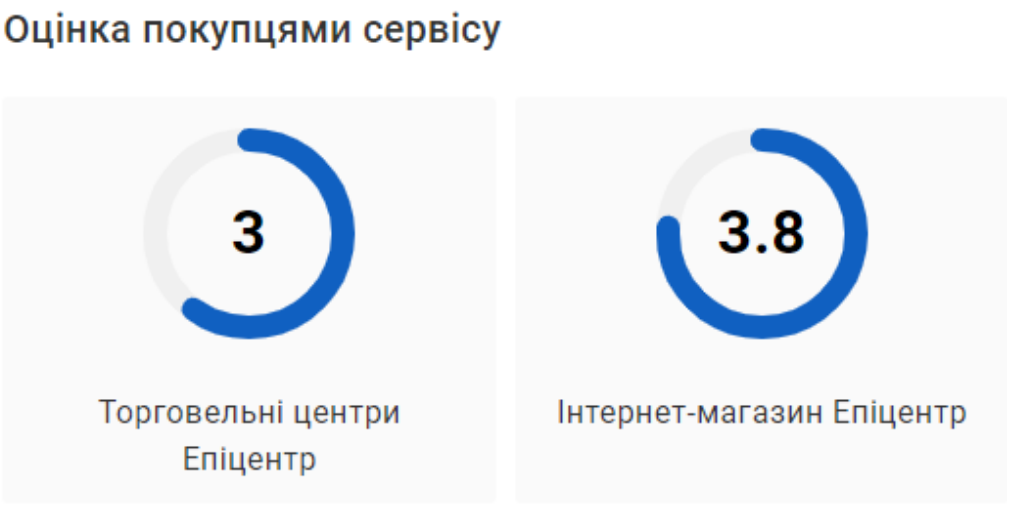

Рис. 1.6. відгуки покупців “Epicenter” на сервіс

Джерело: epicentrk.ua/ua/reviews

AutoRia (Рис. 1.7.) — це провідний український онлайн-ресурс для купівлі та продажу автомобілів. Заснований у 2002 році, сайт став одним з найбільших майданчиків для торгівлі транспортними засобами в Україні, пропонуючи користувачам широкий вибір нових і вживаних автомобілів, мотоциклів, вантажівок та інших транспортних засобів. AutoRia надає зручний інтерфейс для розміщення оголошень, а також розширені функції пошуку та фільтрації, що дозволяють швидко знайти потрібний транспортний засіб за різними критеріями. Платформа також пропонує сервіси перевірки історії автомобілів, оцінки ринкової вартості та різні фінансові послуги, такі як кредитування та страхування.

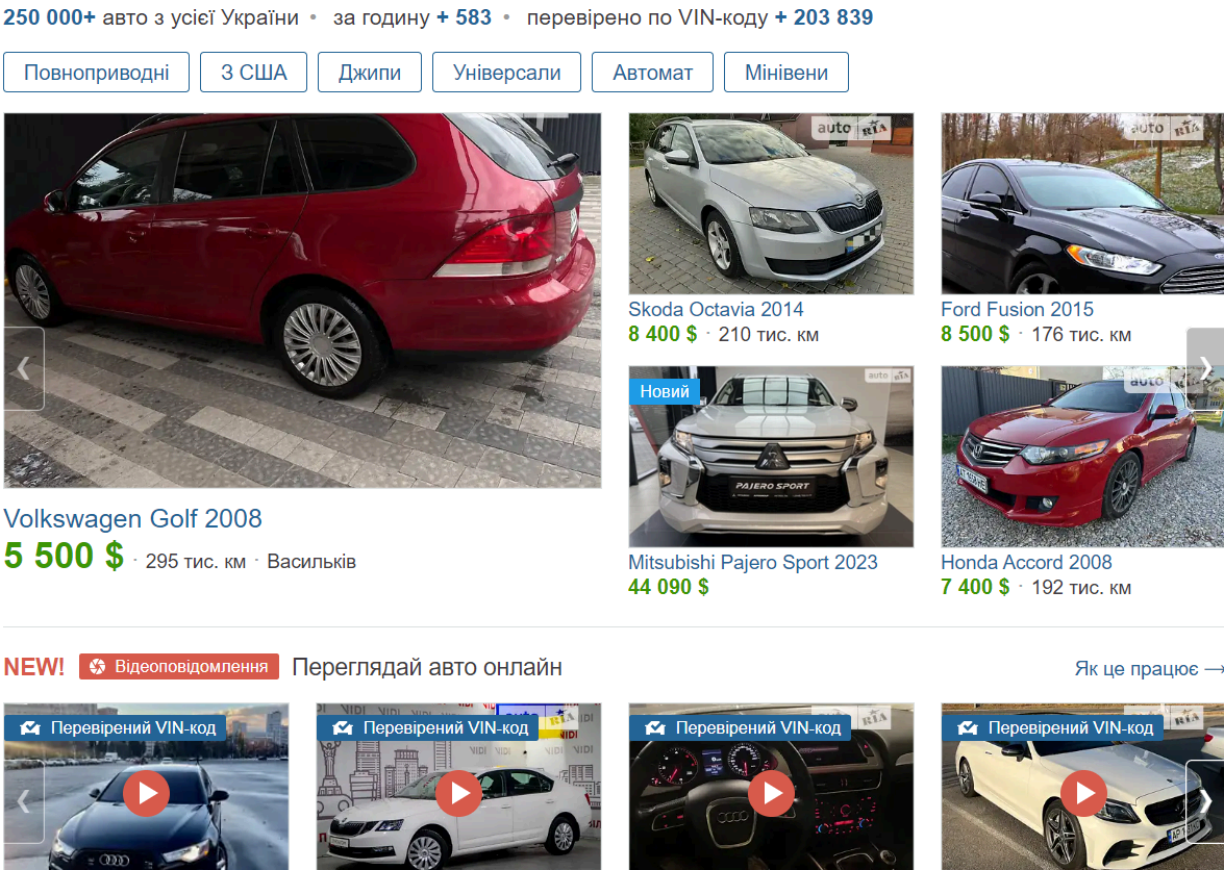

Рис. 1.7 графічний інтерфейс програми “AutoRia”

Джерело: auto.ria.com/uk/

Аналіз конкурентів показав, що на ринку присутня велика конкуренція і кожен учасник ринку має доволі потужний функціонал та пропрацьований шлях користувача, який складається з кількох пунктів:

1. перша покупка користувача
2. авторизація для збереження історії замовлень
3. заохочення користувача купляти товар саме у них різними емейл розсилками про знижки та повідомленнями в браузері/додатку
4. постійні знижки
5. підтримка користувача по різних питаннях/проблемах при покупці/доставці товару.

Висока конкуренція дозволила нам сформувати SWOT аналіз додатку для того, щоб усвідомити слабкі та сильні сторони нашого продукту, які будуть враховані при розробці.

Таблиця 1.1

SWOT аналіз майбутнього додатку

Сильні сторони	Слабкі сторони
1. Простий інтерфейс 2. Висока гнучкість, яка дозволить швидко змінити цільову аудиторію за потреби 3. Велика кількість апдейтів для проведення A/B тестування на початкових стадіях 4. Високий рівень захисту даних користувача	1. Відсутність підтримки користувача при оплаті/доставці в MVP версії 2. Відсутність бренду 3. Висока конкуренція 4. Відсутність експертизи на ринку
Можливості	**Загрози**
1. Залучення нових спеціалістів у команду для пришвидшення процесу роботи 2. Підключення третьо-партійних сервісів аналітики такі як Amplitude, Google Analytics, Facebook CAPI для збору інформації користування додатком 3. Реклама та проведення A/B тестів для знаходження прибуткової бізнес-стратегії 4. Підключення різних платіжних провайдерів для полегшення процесу онлайн-платежів та диверсифікації ризиків бану провайдера	1. Нові конкуренти 2. Зміна законодавства IT сфери та/або введення нових податків 3. Проблеми з production середовищем 4. Потреба в розширенні серверної інфраструктури через великий приплив користувачів

Джерело: створено автором

На основі аналізу конкурентів та SWOT-аналізу, ми з’ясували, що користувачі люблять простий, інтуїтивно-зрозумілий дизайн інтернет-магазинів, де чітко вказано, що ти купуєш, за яку ціну з зрозумілою та швидкою доставки. Саме ця інформація допоможе нам закласти фундамент нашої майбутньої програми власного інтернет-магазину.

Висновки до розділу 1

У цьому розділі ми описали проблему та розказали про стрімкий розвиток сфери електронної комерції. Було проведено аналіз найбільш успішних гравців на українському ринку. На основі цієї інформації було створено модель успішного додатку в даній сфері та описано SWOT аналіз майбутнього додатку.

РОЗДІЛ 2

ПРОЕКТУВАННЯ СИСТЕМИ ВЕБ-ДОДАТКУ

2.1. Вибір стеку технологій

Для максимально швидкої та гнучкої розробки потрібно вибирати інструменти для розробки, які мають наступні характеристики:

1. Легкість написання коду.
2. Готові компоненти, які можуть використовуватись в проекті велику кількість разі.
3. Популярність, що означає:
    a. Інструмент, на якому буде розроблятись продукт має сумісність з багатьма іншими інструментами.
    b. Велике комьюніті, яке має велику експертизу і допоможе знайти допомогу на таких сайтах як stackoverflow.com, якщо виникають питання.
    c. Хороша документація від розробників інструменту.
    d. Велика кількість допоміжних інструментів, які можна буде завантажити через менеджери пакетів.

Дані правила для підбору інструменту допоможуть нам вибрати найкращі бібліотеки та фреймворки на сьогодні для розробки продукту.

2.1.1. Фронтенд частина

Для того, щоб створити швидку, інтерактивну UX/UI систему, нам потрібно вибрати найбільш популярні та перевірені часом бібліотеки та фреймворки, які будуть легкими у розробці та підтриманні чистоти коду. На основі цих даних створено наступний список інструментів, потрібних у створенні сучасного веб-сайту.

Vite - Популярний компілятор на сьогоднішній день, який має на меті замінити застарілі CRA(Create React App) та інші компілятори js коду. З його допомогою на виході ми отримуємо оптимізовану релізну збірку та багато інших особливостей для пришвидшення швидкості розробки клієнтської частини, такі як:

1. **швидкість розробки**: основний акцент в Vite.js зроблено на швидкості розробки. Vite використовує ESM (ECMAScript Modules) під час розробки, що дозволяє швидше перевантаження коду та більше миттєвих змін без необхідності повторної компіляції.
2. **гаряче перезавантаження** (Hot Module Replacement - HMR): Vite надає можливість гарячого перезавантаження, що дозволяє бачити зміни в реальному часі без повторної завантаження сторінки.
3. **миттєва компіляція**: в режимі розробки Vite компілює тільки ті частини коду, які фактично змінились. це робить процес компіляції значно швидше.
4. **модульна архітектура**: Vite побудований навколо модульної архітектури, що сприяє розділенню коду на логічні частини.
5. **підтримка SSR** (Server-Side Rendering): Vite може працювати як інструмент для рендерингу на стороні сервера, що полегшує створення універсальних додатків (додатків, які можуть працювати як на клієнті, так і на сервері).

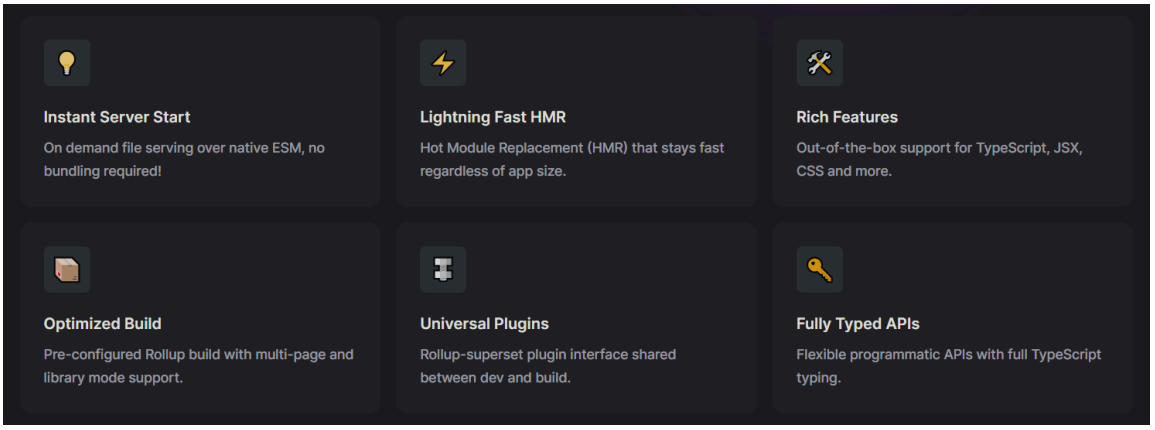

Рис. 2.1. Переваги vite.js

Джерело: vitejs.dev

React - Найкращий друг стартапу та найпопулярніша UI бібліотека для розробки клієнтської частини через надзвичайну гнучкість та легкість розробки. Основні переваги інструменту:

1. **компонентний підхід**: React базується на компонентній архітектурі, що дозволяє розбити інтерфейс на невеликі, компоненти, які можна використовувати багато разів. Це полегшує розробку, тестування і утримання коду.
2. **віртуальний DOM**: React використовує віртуальний DOM для ефективного оновлення і відображення змін в інтерфейсі. React відображає лише зміни у компонентах і не рендерить весь додаток постійно, що збільшує швидкодію веб-програми у користувача
3. **велика екосистема різних бібліотек та фреймворків**: React може легко інтегруватися з іншими бібліотеками та фреймворками, такими як Redux для керування станом, або React Router для навігації.
4. **декларативний синтаксис**: React використовує декларативний синтаксис, що дозволяє описувати, як повинен виглядати інтерфейс у певний момент, і React самостійно керує оновленням відповідно до змін у стані.
5. **підтримка інструментів розробки**: розробка на React полегшується завдяки різноманітним інструментам, таким як React DevTools, які допомагають відстежувати та аналізувати структуру компонентів.

Typescript - це мова програмування, яка є розширенням JavaScript. Потужний інструмент для розробки веб-додатків які вимагають великої ступені надійності та легкої утримуваності. Основні перевагу інструменту:

1. **статична типізація**: однією з ключових переваг є можливість визначення типів для змінних, параметрів функцій та інших елементів коду. Це дозволяє виявляти помилки під час компіляції, що полегшує відладку та поліпшує надійність коду.

2. **підтримка ООП**: TypeScript підтримує об'єктно-орієнтоване програмування, включаючи класи, інтерфейси, наслідування та поліморфізм, що дозволяє розробникам створювати більш структуровані та легко утримувані програми.
3. **екосистема JavaScript**: TypeScript побудований на основі JavaScript і повністю сумісний з існуючим JavaScript-кодом. Це дозволяє поступово впроваджувати TypeScript в проекти і забезпечує велику сумісність із засобами і бібліотеками JavaScript.
4. **кодова безпека та підтримка інструментів рефакторингу**: статична типізація допомагає уникати типових помилок, а інструменти рефакторингу та автодоповнення забезпечують продуктивну розробку.
5. **типізація об'єктів**: TypeScript дозволяє визначати типи для складних структур даних, таких як об'єкти та масиви, що полегшує роботу з ними та запобігає потенційним помилкам.

Redux - бібліотека для керування станом додатків у JavaScript. Основні переваги бібліотеки:

1. **централізований стан**: Redux зберігає весь стан додатка в одному місці - відомому як "store". Це полегшує відстеження та управління станом вашого додатка.
2. **прогнозований стан**: стан у Redux є незмінним об'єктом, і зміни в стані відбуваються за допомогою "екшенів" (actions). Це робить стан додатка прогнозованим та відтворюваним.
3. **спрощене управління станом**: Redux використовує чисті функції, відомі як "reducers", для управління змінами в стані. Це спрощує управління та розуміння того, як змінюється стан у різних частинах додатка.
4. **легко інтегрована взаємодія з React**: Redux часто використовується разом із бібліотекою React, що полегшує інтеграцію та спільне використання стану між компонентами React.

5. **легко локалізує помилки**: Оскільки стан є централізованим, виявлення та виправлення помилок у стані стає простіше, оскільки ви знаєте, де знаходиться весь стан.
6. **розширюємість**: Redux спроектований з урахуванням легкої розширюваності. Це дозволяє додавати додаткові функціональності, такі як middleware, для розширення можливостей бібліотеки.

Також у клієнтському додатку будуть використовуватись різні бібліотеки, які додаються через node-js пакет менеджер NPM, наприклад:

1. **MUI** - популярна бібліотека для розробки інтерфейсів користувача веб-додатків, яка має багато готових реакт компонентів з високим рівнем кастомізації

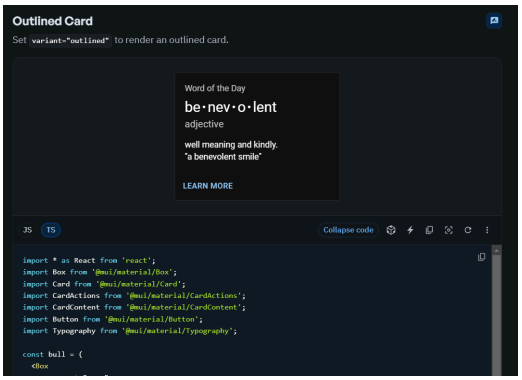

Рис. 2.2. Приклад готового компоненту “Card” на сайті MUI

Джерело: mui.com/material-ui/react-card

2. **axios** - популярна бібліотека для створення та виконання запитів на бекенд частину
3. **react-hook-form** - бібліотека для легкої та швидкої обробки форм

Рис. 2.3. Переваги react-hook-form

Джерело: react-hook-form.com

4. **yup** - валідаційна схема форм, працює у зв’язці з react-hook-form.
5. **react-router-dom** - налаштування url сторінок, їх компонентів та переходів між ними

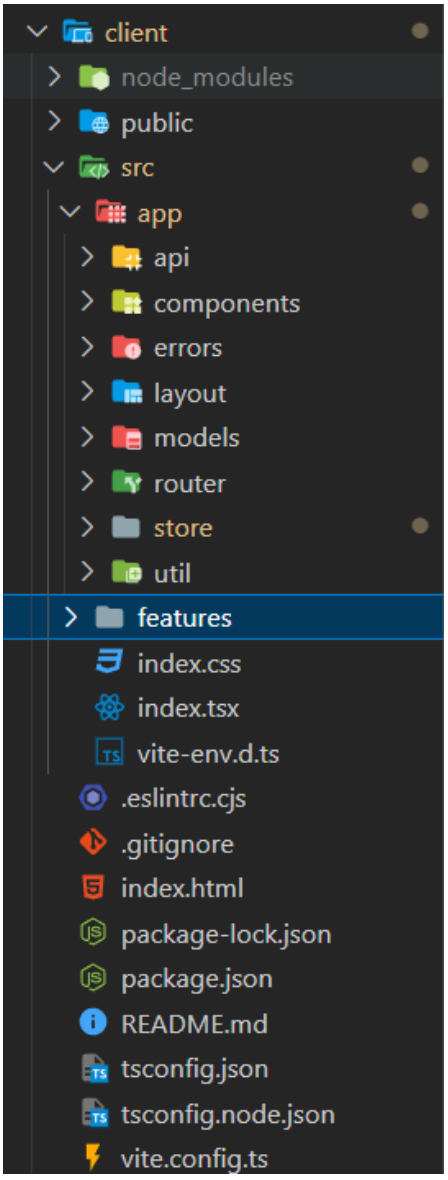

Рис. 2.4. Структура клієнтської частини

Джерело: створено автором

Підбиваючи підсумки, я вважаю що саме такий стек технологій на клієнтській частині дозволить нам створити гнучкий, легко маштабований функціонал веб-додатку.

2.1.2. Бекенд частина

Для прогресивної серверної частини було обрано фреймворк у відкритому доступі ASP.NET, який призначений для створення web UI та web API систем, який має гарну документацію, підтримку Microsoft та є легким у розширенні функціоналу.

1. ASP.NET Core 7 - фреймворк для розробки веб-додатків, мультиплатформенний, з високою оптимізацією для високої продуктивності та ефективності в роботі з пам'яттю та веб-запитами, підтримує легку інтеграцію з Docker та фронтенд фреймворками.
2. ASP.NET Core Web API - фреймворк для розробки веб-серверів та API (інтерфейсів програмування застосунків) за допомогою ASP.NET Core.
3. EntityFrameworkCore - технологія доступу до даних для .NET. Продукт для роботи з базами даних у .NET-додатках.
4. Jwt bearer - захищена токенізована авторизація.
5. EntityFrameworkCore Identity - це система аутентифікації та авторизації для додатків, побудованих на фреймворку ASP.NET Core. Вона надає готовий набір інструментів для роботи з користувачами, ролями, підтримує зберігання та валідацію облікових записів, інтегрується з базами даних та надає можливості для налаштування політик безпеки.
6. Swagger - фреймворк для документації та тестування RESTful API.
7. PostgreSql - потужна, безкоштовна СУБД (система управління базами даних) для зберігання та обробки даних.

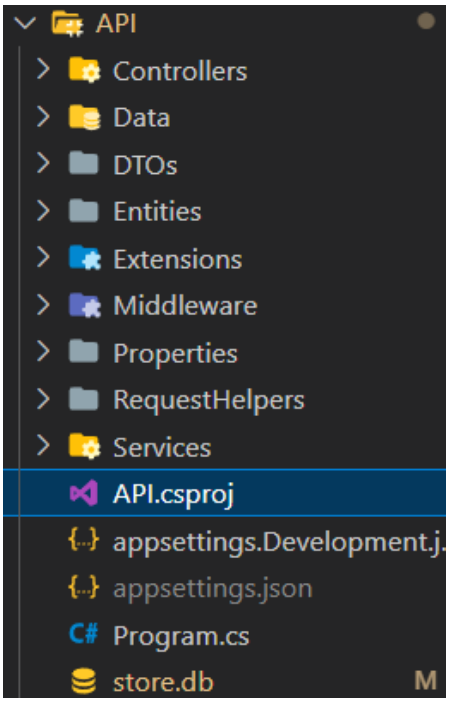

Рис. 2.5. Структура бекенд частини

Джерело: створено автором

Даний стек технологій на серверній частині дозволить нам швидко розгорнути та розширити функціонал веб-додатку.

2.1.3. Інфраструктурна частина

Для спрощення процесу роботи та створення організації контролю проектом було вирішено використовувати такі інструменти, як Git, Github та Docker.

Для створення повноцінної версії електронної комерції було вирішено використати сервіс для зберігання медіафайлів Cloudinary, сервіс для проведення транзакцій Stripe, а також сервіс для хостингу проекту Fly.io.

Git - система керування версіями, яка використовується для відстеження змін у вихідному коді під час розробки програмного забезпечення. Вона дозволяє багатьом розробникам працювати над одним проектом одночасно, зберігаючи при цьому історію всіх внесених змін.

Github - веб-сервіс для хостингу та спільної роботи над проектами, які використовують систему керування версіями Git. Заснований у 2008 році, GitHub став однією з найпопулярніших платформ для розробників, надаючи інструменти для спільної роботи, версійного контролю, та управління проектами. Сервіс також зручний тим, що ми зможемо за допомогою інтеграції з Docker та Fly.io завантажувати нову версію веб сайту в production за допомогою Github Actions.

Github Actions - інструмент для автоматизації робочих процесів (workflows). В даному випадку нас цікавить саме налаштування безперервної інтеграції та безперервного розгортання (CI/CD).

Docker - платформа з відкритим вихідним кодом, яка дозволяє автоматизувати розгортання, масштабування та управління додатками за допомогою контейнеризації. Контейнери забезпечують ізольоване середовище для запуску програмного забезпечення, включаючи всі необхідні залежності, бібліотеки та конфігураційні файли, що дозволяє забезпечити стабільність і портативність додатків. Завдяки цьому сервісу ми будемо мати можливість зберегти наший додаток в окремий контейнер та відобразити для світу через Fly.io.

Cloudinary - хмарна платформа для управління медіа-контентом, що надає інструменти для завантаження, зберігання, трансформації та доставки зображень і відео. Платформа дозволяє автоматизувати багато аспектів роботи з медіа-контентом і забезпечує його оптимальне відображення на різних пристроях і платформах. Платформа дозволить нам проводити інвентаризацію продуктів, в тому числі додавати нові картинки для продуктів та відображати їх для користувачів.

Stripe - платіжна платформа, яка дозволяє бізнесам обробляти онлайн-платежі. З початку заснування Stripe швидко стала однією з провідних компаній у сфері фінансових технологій (FinTech), пропонуючи комплексні рішення для інтеграції платіжних систем у вебсайти та мобільні додатки.

Fly.io - платформа для розгортання та управління додатками в хмарі, яка фокусується на забезпеченні глобальної доступності та високої продуктивності додатків. Вона дозволяє розробникам легко розгортати свої додатки ближче до

кінцевих користувачів, зменшуючи затримки і покращуючи загальний користувацький досвід.

Висновки до розділу 2

У цьому розділі було описано критерії вибору технологій для майбутнього додатку електронної комерції і на основі цієї інформації було складено технологічний стек проекту для серверної та клієнтської частини. Ключовими елементами клієнтської частини є React, Typescript та графічна бібліотека компонентів MUI. Ключовими елементами серверної частини є платформа ASP.NET Core та її бібліотеки. Також виділили третьо-партійні сервіси, які будуть інтегровані в додаток під час розробки програми. Серед них Stripe, Fly.io та Cloudinary.

РОЗДІЛ 3

РОЗРОБКА СИСТЕМИ ВЕБ-ДОДАТКУ

3.1. Реалізація Бекенд частини

Головний функціонал додатку складається з авторизації через JWT токенізацію, CRUD операції над продуктами, кошиком та створення/обробка замовлень. З таблицями користувача, його логінами, ролями допомагає Identity бібліотека.

CRUD (Create, Read, Update, Delete) - означає, що ми створюємо систему, в якій користувач буде мати можливість переглядати товари, додавати їх в корзину або створювати корзину і оплачувати товари. Адміністратор в свою чергу буде мати можливість змінювати зміст товару, його ціну, наявність на складі тощо, додавати нові або видаляти старі товари.

JWT (JSON Web Token) - це стандарт для створення токенів доступу, який використовується для передачі інформації між сторонами у компактному і безпечному форматі. JWT-токени широко застосовуються в авторизації та аутентифікації для підтвердження особистості користувача і надання доступу до ресурсів.

Після успішного входу користувача в систему (наприклад, через введення логіна і пароля), сервер створює JWT-токен, який містить інформацію про користувача і його права доступу. Токен підписується секретним ключем і відправляється клієнту. Клієнт зберігає створений ключ у локальному сховищі і при здійсненні кожного запиту на сервер передає в заголовку запиту токен, щоб підтвердити особу. Сервер перевіряє інформацію, зашифровану в токені і надає доступ до результату запиту, якщо перевірка пройшла успішно.

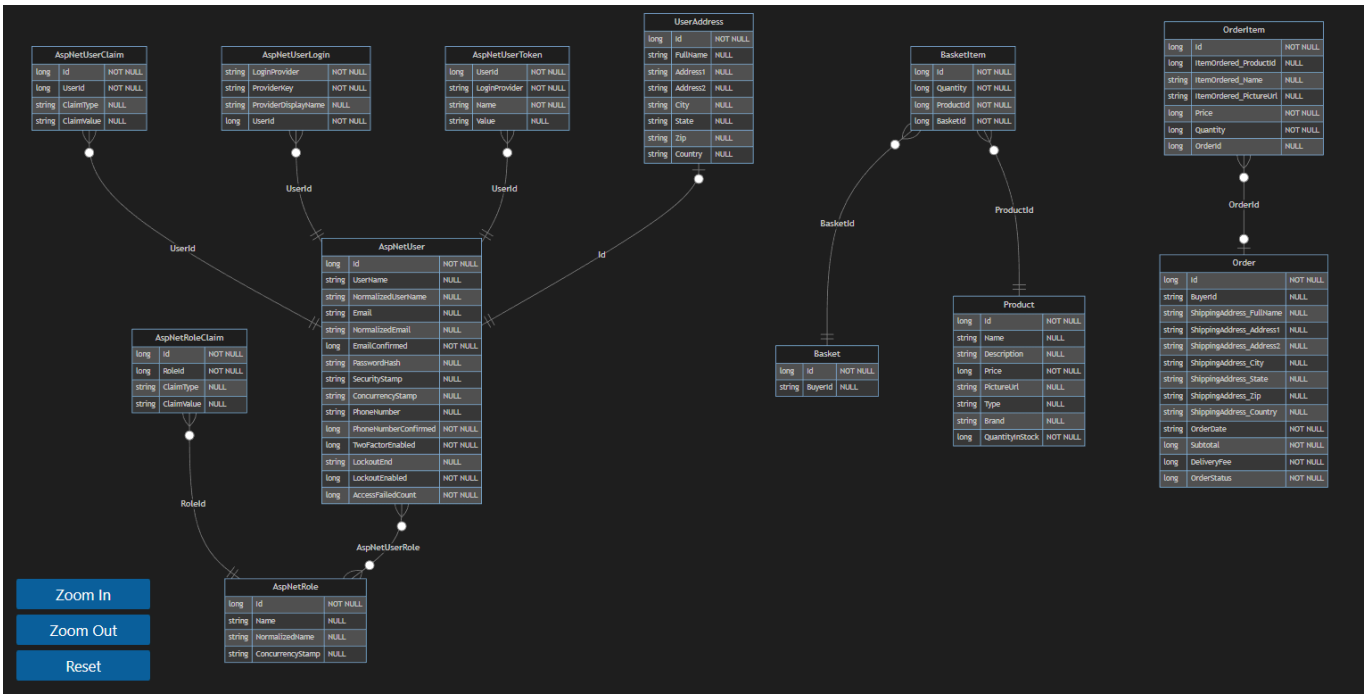

Рис. 3.1. Діаграма бази даних

Джерело: створено автором

Для реалізації бекенд частини були створені наступні endpoints для клієнтської частини:

1. [POST] api/account/login - вхід юзера в систему.
    a. Має поле імені та паролю.
2. [POST] api/account/register - реєстрування юзера на сайті.
    a. Має поле імені, емейлу, та паролю.
3. [GET] api/account/currentUser - збір інформації про конкретного юзера, пошук здійснюється по jwt токену.
    a. Запит не має тіла, потрібен лише токен в заголовку.
4. [GET] api/account/savedAddress - збір інформації про збережені дані адреси для швидкої оплати користувачем.
    a. аналогічна логіка роботи, як у api/account/currentUser.

Лістинг 3.1. Endpoint для логіну користувача

```
[HttpPost("login")]
public async Task < ActionResult < UserDto >> Login(LoginDto loginDto) {
```

```
  var user = await _userManager.FindByNameAsync(loginDto.Username);
  if (user == null || !await _userManager.CheckPasswordAsync(user,
loginDto.Password))
    return Unauthorized();

  var userBasket = await RetrieveBasket(loginDto.Username);
  var anonBasket = await RetrieveBasket(Request.Cookies["buyerId"]);

  if (anonBasket != null) {
    if (userBasket != null) _context.Baskets.Remove(userBasket);
    anonBasket.BuyerId = user.UserName;
    Response.Cookies.Delete("buyerId");
    await _context.SaveChangesAsync();
  }

  return new UserDto {
    Email = user.Email,
      Token = await _tokenService.GenerateToken(user),
      Basket = anonBasket != null ? anonBasket.MapBasketToDto() :
userBasket?.MapBasketToDto()
  };
}
```

Сутність Basket - кошик, де зберігаються продукти перед покупкою. Якщо юзер не має акаунту, створюється Cookie buyerId, яка передається на клієнтську частину і Cookie активна протягом 30 днів. Це дозволяє нам зберігати інформацію кошику у тимчасового користувача і полегшує процес покупки продуктів.

1. [GET] api/Basket - збір інформації по кошику.
2. [POST] api/Basket - створення кошика, коли додаються нові продукти. Якщо клієнт не зареєстрований, у нього може бути кошик, який створюється за допомогою куки buyerId формату UUID. При реєстрації користувача buyerId замінюється на ім’я користувача.
3. [DELETE] api/Basket - видалення продуктів з кошику.

Лістинг 3.2. Endpoint для отримання кошика користувачем

```
private async Task < Basket > RetrieveBasket(string buyerId) {
```

```
  if (string.IsNullOrEmpty(buyerId)) {
    Response.Cookies.Delete("buyerId");
    return null;
  }

  return await _context.Baskets
    .Include(i => i.Items)
    .ThenInclude(p => p.Product)
    .FirstOrDefaultAsync(basket => basket.BuyerId == buyerId);
}
```

Сутність Order - замовлення користувача. Після успішної покупки сутність кошику переходить у замовлення.

1. [GET] api/Orders - збір інформації по всіх замовленнях авторизованого користувача.
2. [POST] api/Orders - створення замовлення. При створенні видаляється basket entity, прикріплене за користувачем.
3. [GET] api/Orders/{id} - збір інформації по конкретному замовленні.

Лістинг 3.3. Endpoint для отримання всіх замовлень за прикріпленим користувачем

```
[HttpGet]
public async Task < ActionResult < List < OrderDto >>> GetOrders() {
  var orders = await _context.Orders
    .ProjectOrderToOrderDto()
    .Where(x => x.BuyerId == User.Identity.Name)
    .ToListAsync();

  return orders;
}
```

Сутність Product - одиниця товару, яка відображається на клієнтській частині, яку можна вибрати і додати в кошик, а також сформувати замовлення. Ця сутність є основною в додатку, тому що саме цю сутність будуть купувати користувачі, ціну з цією сутністю ми будемо передавати в сервіс Stripe, оперувати кошиком, замовленнями тощо.

1. [GET] api/Products - збір інформації по продуктам за фільтрами та пагінацією.
2. [GET] api/Products/{id} - збір інформації по конкретному продукту.
3. [GET] api/Products/filters - збір інформації по доступних брендах та категоріях для фільтрації.
4. [POST] api/Products - створення нового продукту.
5. [PUT] api/Products - зміна існуючого продукту.
6. [DELETE] api/Products/{id} - видалення існуючого продукту.

Лістинг 3.4. Endpoints для створення продукту

```
[Authorize(Roles = "Admin")]
[HttpPost]
public async Task < ActionResult < Product >> CreateProduct([FromForm]
CreateProductDto productDto) {
  var product = _mapper.Map < Product > (productDto);

  if (productDto.File != null) {
    var imageResult = await _imageService.AddImageAsync(productDto.File);

    if (imageResult.Error != null) return BadRequest(new ProblemDetails {
      Title = imageResult.Error.Message
    });

    product.PictureUrl = imageResult.SecureUrl.ToString();
    product.PublicId = imageResult.PublicId;
  }

  _context.Products.Add(product);

  var result = await _context.SaveChangesAsync() > 0;

  if (result) return CreatedAtRoute("GetProduct", new {
    Id = product.Id
  }, product);

  return BadRequest(new ProblemDetails {
    Title = "Problem creating new product"
  });
}
```

Варто зазначити, що для створення продуктів я створюю картинку за допомогою imageService. За допомогою цієї обгортки файл, який мені приходить в тіло запиту обробляється на стороні третьо-партійного сервісу Cloudinary і як результат я отримую посилання на картинку в хмарі.

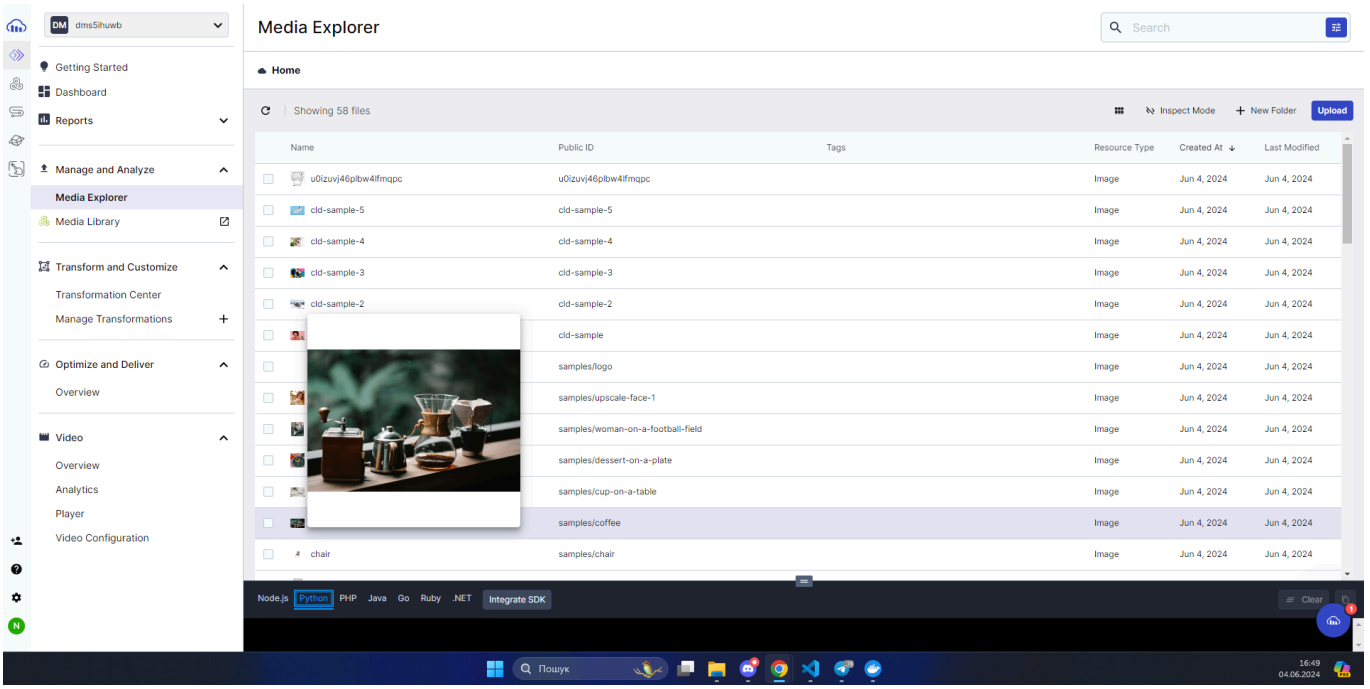

Рис. 3.2. Адмін-панель сервісу Cloudinary

Джерело: console.cloudinary.com

Cloudinary використовується багатьма відомими брендами, як Spotify, Uber, Nike, Samsung та інші. Сервіс є чудовим вибором для компаній та розробників, які шукають потужний, простий у використанні та масштабований інструмент для роботи з зображеннями. Також є плюсом те, що він абсолютно безплатний, якщо ти зберігаєш менше 25 гігабайтів контенту.

Лістинг 3.5. ImageService

```
public class ImageService {
  private readonly Cloudinary _cloudinary;

  public ImageService(IConfiguration config) {
    var acc = new Account(
```

```
      config["Cloudinary:CloudName"],
      config["Cloudinary:ApiKey"],
      config["Cloudinary:ApiSecret"]
    );

    _cloudinary = new Cloudinary(acc);
  }

  public async Task < ImageUploadResult > AddImageAsync(IFormFile file) {
    var uploadResult = new ImageUploadResult();

    if (file.Length > 0) {
      using
      var stream = file.OpenReadStream();
      var uploadParams = new ImageUploadParams {
        File = new FileDescription(file.FileName, stream)
      };
      uploadResult = await _cloudinary.UploadAsync(uploadParams);
    }

    return uploadResult;
  }

  public async Task < DeletionResult > DeleteImageAsync(string publicId) {
    var deleteParams = new DeletionParams(publicId);

    var result = await _cloudinary.DestroyAsync(deleteParams);

    return result;
  }
}
```

У сервісу Cloudinary є своя бібліотека CloudinaryDotNet, яку легко інтегрувати в проект та використовувати. При створенні акаунта в сервісі тобі надаються три ключі: назва хмари, апі ключ та секретний ключ. Назву хмари та апі можна показувати, це є публічна інформація. Секретний ключ має бути доступний лише для тебе. Витік такого ключа може понести серйозні збитки для бізнесу, тому потрібно знайти таке рішення, де при запущеному проекті серверна частина буде знати, куди звертатись, але цих даних не буде в коді. Для цього використовуються secrets або

environmental variables. В даному випадку можна використати dotnet secrets, або в сервісі для хостингу Fly.io також є fly secrets.

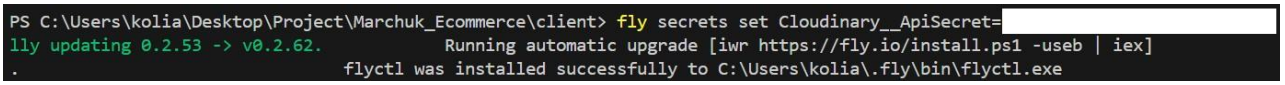

Рис. 3.3. Додавання секретного ключа сервісу Cloudinary в терміналі

Джерело: створено автором

3.2. Реалізація клієнтської частини

Клієнтська частина також має свої основні моделі, завдяки яким відбувається валідування даних, які приходять з запитів серверу та обробка внутрішньої логіки на клієнтській частині. Основні серед них:

Лістинг 3.6. Basket Interface

```
export interface BasketItem {
  productId: number;
  name: string;
  price: number;
  pictureUrl: string;
  brand: string;
  type: string;
  quantity: number;
}

export interface Basket {
  id: number;
  buyerId: string;
  items: BasketItem[];
  paymentIntentId?: string;
  clientSecret?: string;
}
```

Важливо зазначити, що в інтерфейсі використовується поле paymentIntendId та clientSecret, яке нам потрібне для оброблення оплат з допомогою сервісу Stripe. Про це згодом.

Лістинг 3.7. User Interface

```
import { Basket } from "./basket";

export interface User {
    email: string;
    token: string;
    basket?: Basket
    roles?: string[];
}
```

На клієнті за допомогою бібліотеки Redux в будь якому місці в коді проекту ми знаємо, що наш користувач має емейл, якщо він залогінений, токен, кошик, якщо він вже обрав товари і додав їх у кошик, а також ролі. Наразі в додатку є лише роль Member - звичайний користувач та Admin - адміністратор який має доступ до адмін-панелі. Це нам потрібно, щоб не давати звичайному користувачу доступу до сторінки інвентаризації продуктів.

Фронтенд проект складається з таких основних сторінок:

1. Логін
2. Реєстрація
3. Каталог
4. Чекаут
5. Історія замовлень
6. Кошик
7. Інвентаризація для адмін користувачів

Сторінка каталогу дозволяє нам бачити список доступних продуктів, де можна швидко додати продукт в кошук або переглянути більше інформації по продукту.

Панель фільтрації дозволяє сортувати список продуктів за різними параметрами. Кожен раз, коли ми робимо зміну в панелі, ми шлемо новий запит на сервер, де сервер в свою чергу сортує всі записи за заданими параметрами і як результат віддає новий список клієнту, де ми рендеримо знову список продуктів.

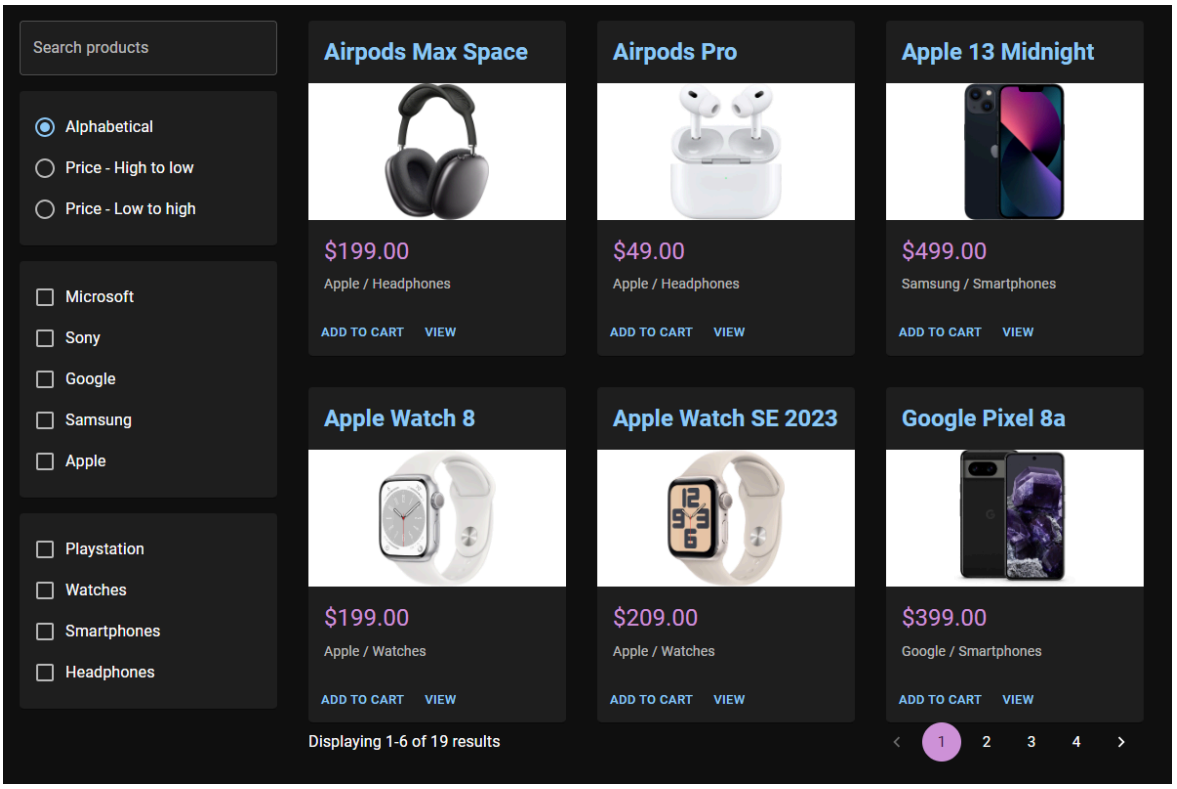

Рис. 3.4. Сторінка каталогу

Джерело: створено автором

Компонент створений за допомогою використання Grid - компонент React в Material UI, який використовується для створення сіток макета. Він дозволяє легко розбивати інтерфейс на рядки та стовпці, а також розміщувати елементи в них.

Лістинг 3.8. Панель фільтрації

```
<Grid item xs={3}>
  <Paper sx={{ mb: 2 }}>
    <ProductSearch />
  </Paper>
  <Paper sx={{ p: 2, mb: 2 }}>
    <RadioButtonGroup
      selectedValue={productParams.orderBy}
      options={sortOptions}
      onChange={(e) => dispatch(setProductParams({ orderBy: e.target.value
}))}
    />
  </Paper>
  <Paper sx={{ p: 2, mb: 2 }}>
```

```
    <CheckboxButtons
      items={brands}
      checked={productParams.brands}
      onChange={(items: string[]) =>
        dispatch(setProductParams({ brands: items }))
      }
    />
  </Paper>
  <Paper sx={{ p: 2 }}>
    <CheckboxButtons
      items={types}
      checked={productParams.types}
      onChange={(items: string[]) =>
        dispatch(setProductParams({ types: items }))
      }
    />
  </Paper>
</Grid>
```

Вибравши продукти в сторінці каталогу, ми можемо перейти в сторінку кошику, де ми зможемо побачити своє замовлення, обрані продукти та їх кількість. У нас також є можливість перед оплатою ще додати кількість обраних продуктів або видалити їх.

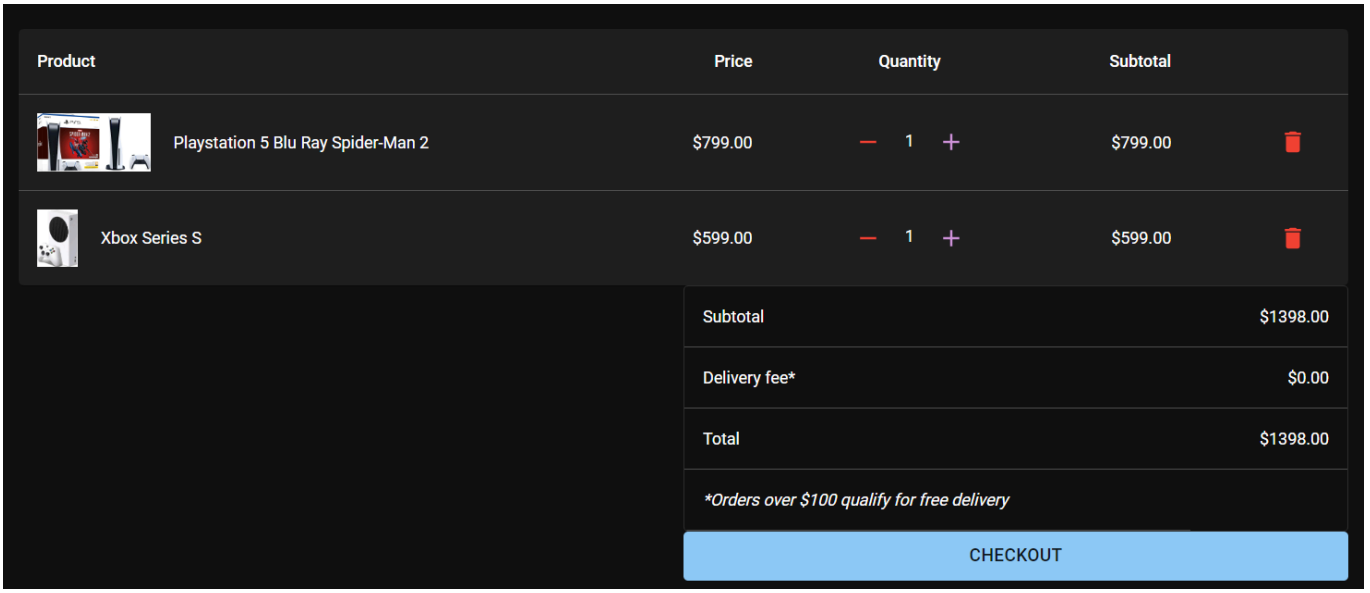

Рис. 3.5. UI вигляд сторінки логіну

Джерело: створено автором

Сторінка логіну - авторизація через ім’я користувача та пароль. Якщо юзер не користувався нашим продуктом раніше, перед покупкою йому потрібно буде пройти процес авторизації, зареєструвавшись в нашому додатку або залогінитись. Логіка обробки форми працює через бібліотеку useForm.

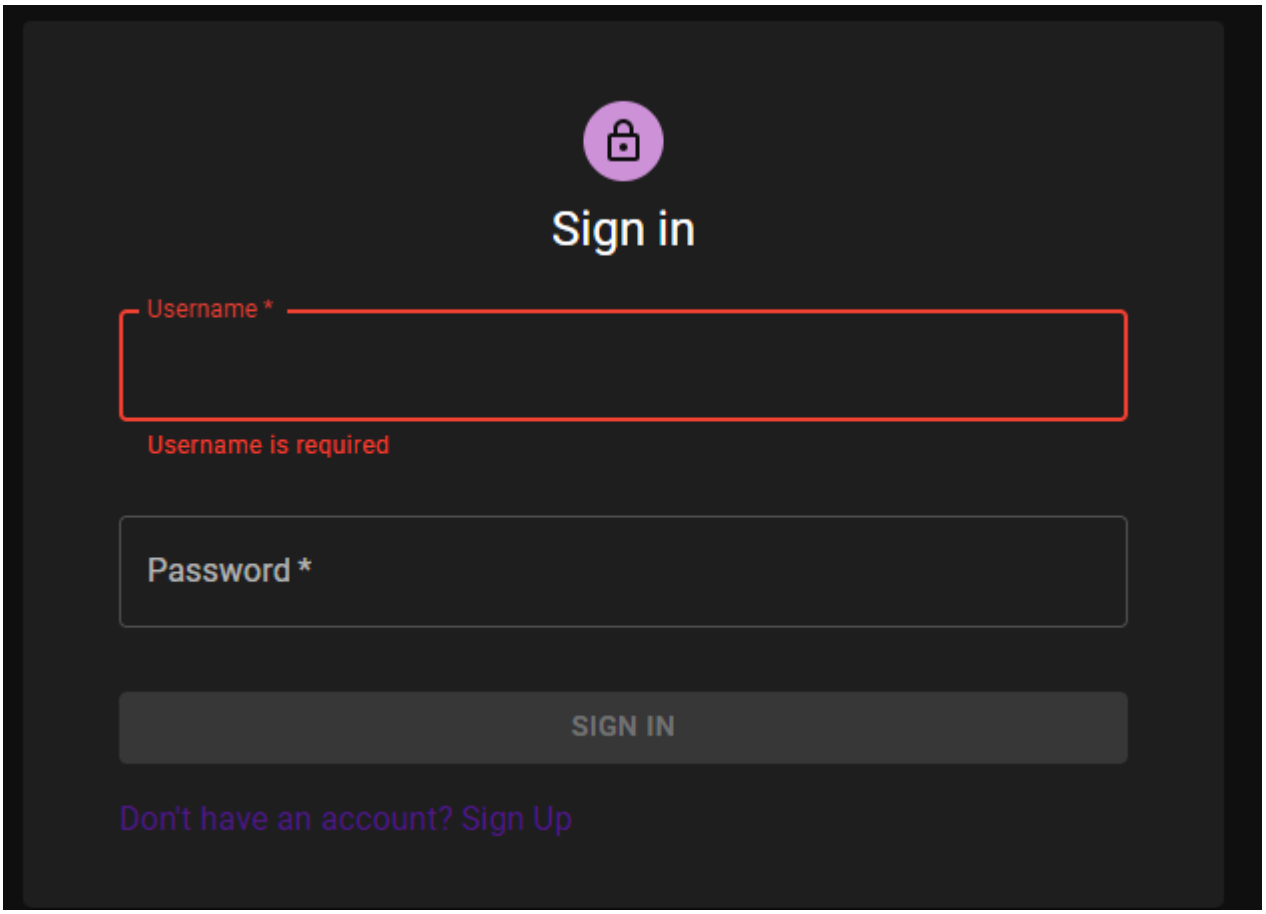

Рис. 3.6. UI вигляд сторінки логіну

Джерело: створено автором

Пройшовши валідацію інпутів, ми дозволяємо натиснути на кнопку Sign In і виконати запит на сервер, де ми зможемо перевірити, чи існує такий юзер і зробити редірект користувача на сторінку каталогу, якщо перевірка пройшла успішно

Лістинг 3.8. Створення форми та обробки кнопки Sign In

```
const {
  register,
```

```
  handleSubmit,
  formState: {
    isSubmitting,
    errors,
    isValid
  }
} = useForm({
  mode: 'onTouched'
});

async function submitForm(data: FieldValues) {
  try {
    await dispatch(signInUser(data));
    navigate(location.state?.from || '/catalog');
  } catch (error) {
    console.log(error);
  }
}
```

В коді знизу було використано createAsyncThunk - функція в Redux Toolkit, яка дозволяє обробляти асинхронні операції, в даному випадку отримання даних від серверного API.

Лістинг 3.9. Надсилання запиту на сервер для спроби користувача залогінитись

```
export const signInUser = createAsyncThunk<User, FieldValues>(
    'account/signInUser',
    async (data, thunkAPI) => {
        try {
            const userDto = await agent.Account.login(data);
            const {basket, ...user} = userDto;
            if (basket) thunkAPI.dispatch(setBasket(basket));
            localStorage.setItem('user', JSON.stringify(user));
            return user;
        } catch (error: any) {
            return thunkAPI.rejectWithValue({error: error.data});
        }
    }
);
```

Оплата здійснюється в 3 етапа:

1. Форма адреси доставки - може бути збережена за бажанням користувача.
2. Перегляд замовлення.
3. Оформлення та оплата.

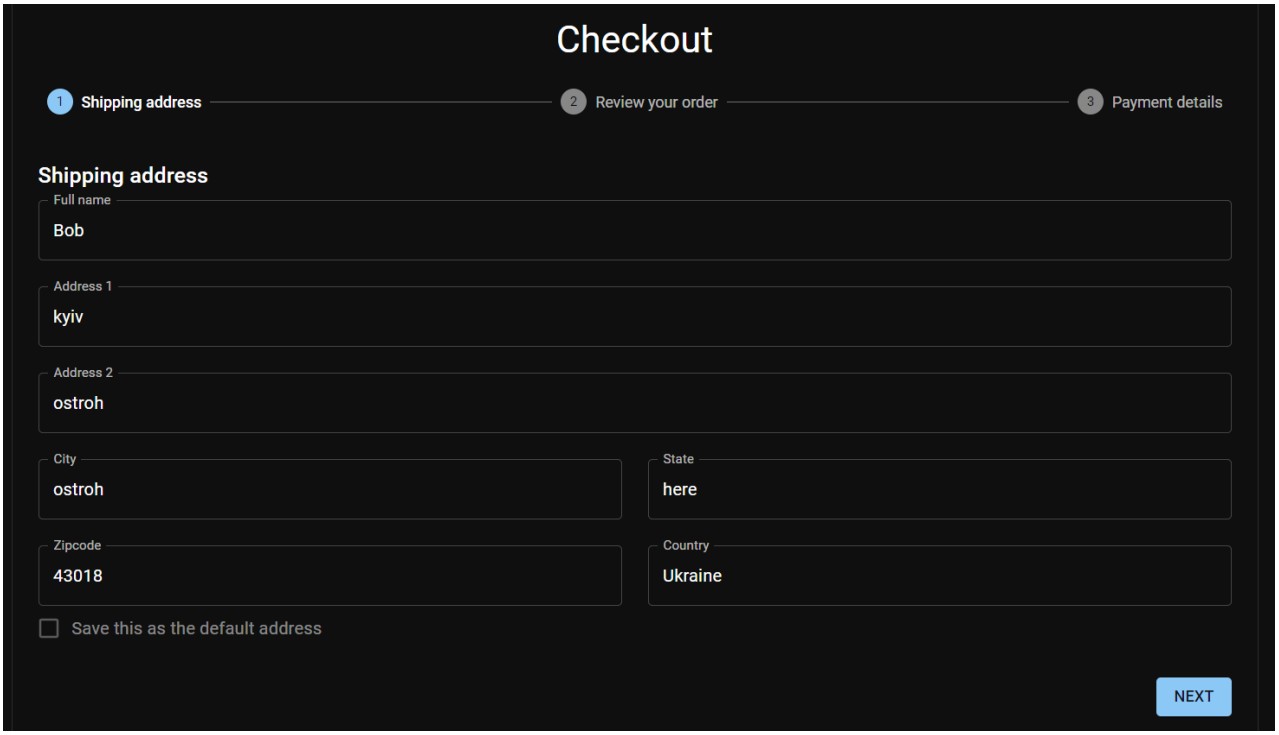

Рис. 3.7. UI вигляд сторінки чекаута на етапі доставки

Джерело: створено автором

Після того як ми ввели свою адресу, більше ми не можемо редагувати наше замовлення, даємо користувачу ще раз переглянути своє замовлення перед його оплатою.

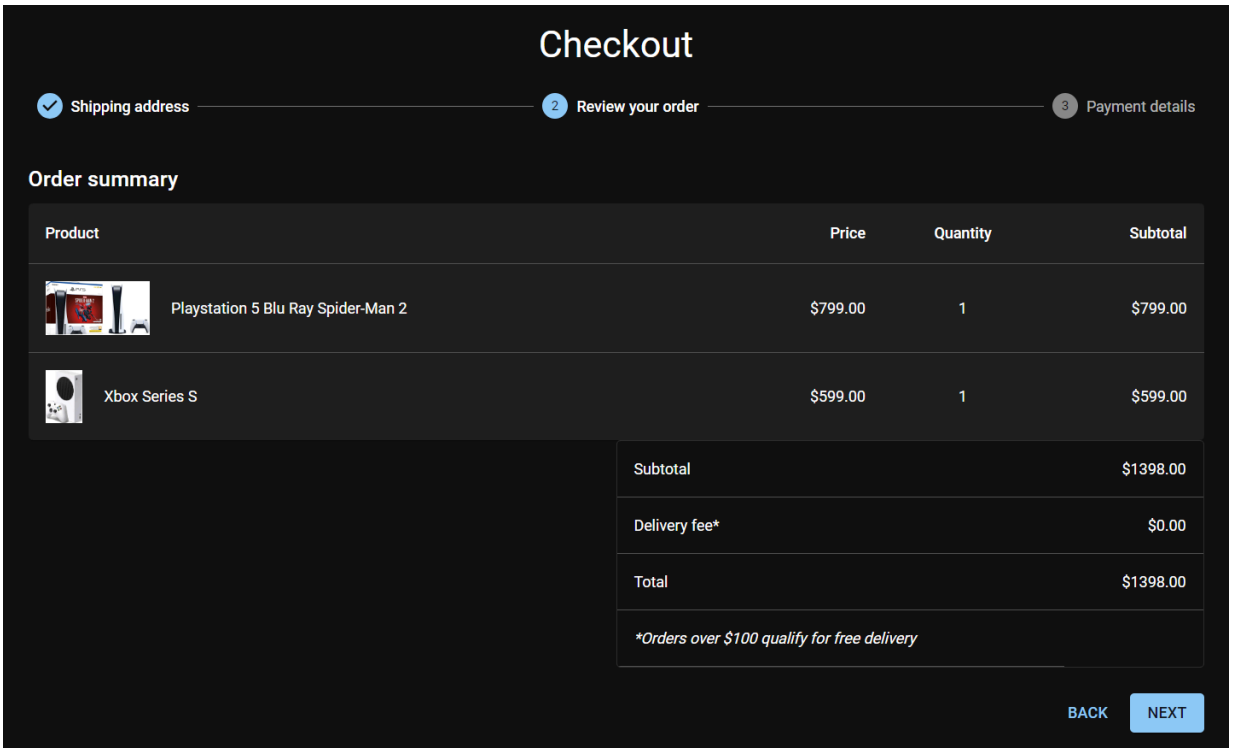

Рис. 3.8. UI вигляд сторінки чекаута на етапі перегляду замовлення

Джерело: створено автором

Всі інпути у формі чекаута створені за допомогою Stripe Elements. Ми використовуємо свій стиль, лише підключаємо логіку обробку кожного елементу бібліотекою нашого платіжного провайдера.

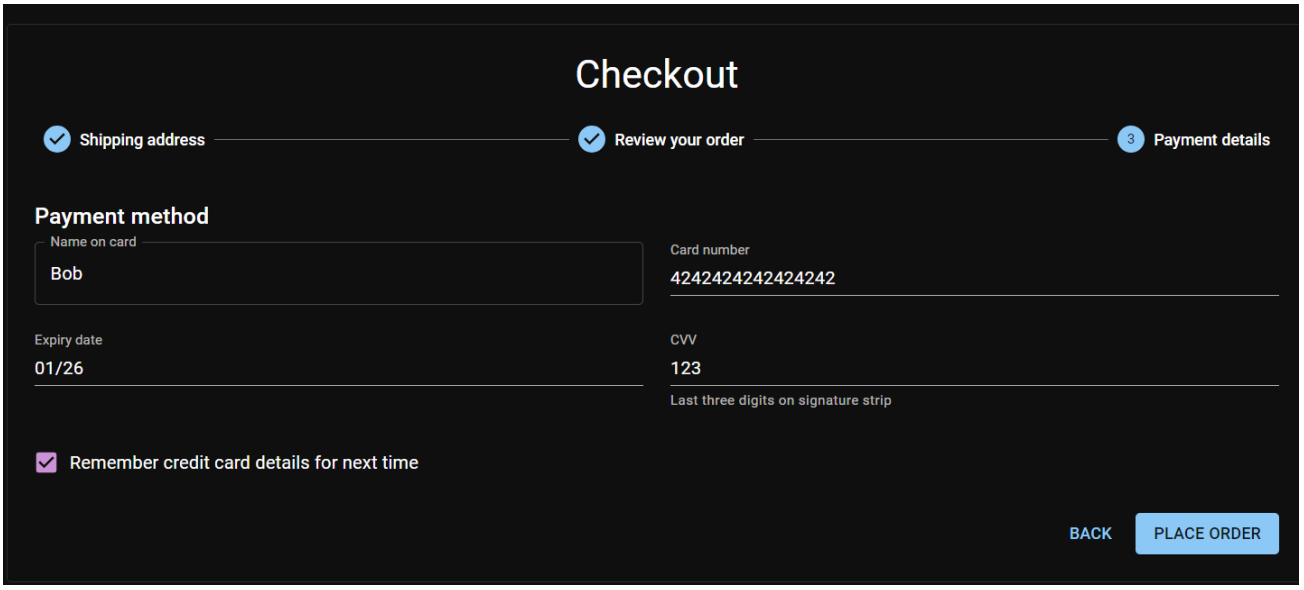

Рис. 3.9. UI вигляд сторінки чекаута на етапі оплати

Джерело: створено автором

Після успішного платежу показуємо користувачу номер замовлення та його статус.

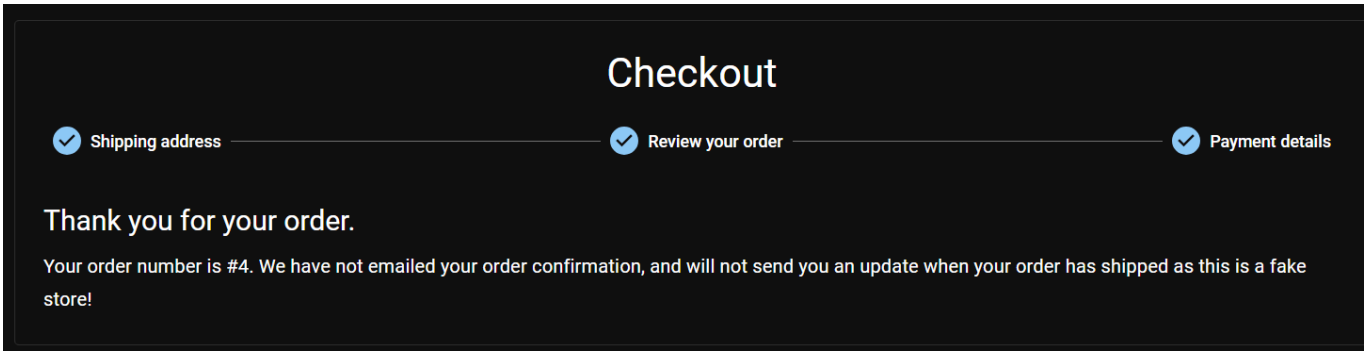

Рис. 3.10. UI вигляд сторінки чекаута на етапі успішного оформлення замовлення

Джерело: створено автором

Після успішної оплати користувач може перейти на сторінку своїх замовлень, та переглянути кожне з них.

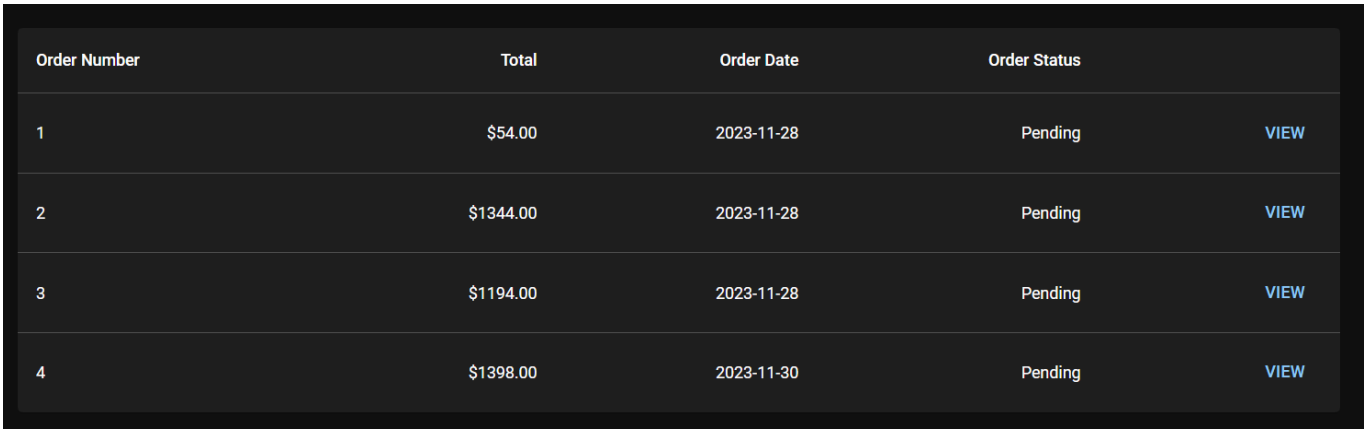

Рис. 3.11. UI вигляд сторінки оформлених замовлень

Джерело: створено автором

Кожне успішно виконане замовлення має свій статус та список куплених продуктів, а також загальну ціну замовлення.

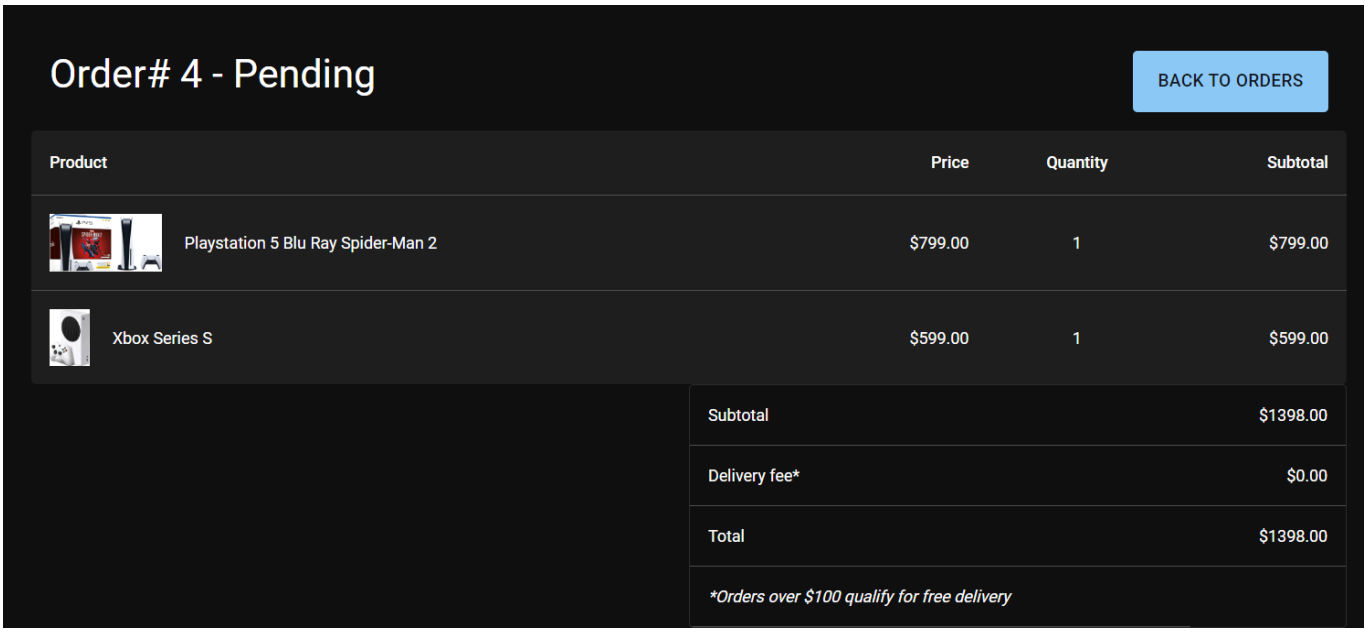

Рис. 3.12. UI вигляд сторінки конкретного оформленого замовлення

Джерело: створено автором

3.3. Реалізація загальної логіки

Оплати є ключовими в електронній комерції для функціонування бізнесу загалом. Створити можливість проводити оплати самому погана ідея, тому що тут потрібна дуже велика експертиза в сфері FinTech, а також потрібно витрати величезні ресурси на імплементацію свого пеймент провайдера. Саме тому є сервіси, які допомагають бізнесам проводити транзакції через своє API та роблять велику частину роботи за тебе. Саме тому було вирішено взяти за основу Stripe, як основного посередника між нашим бізнесом та гаманцем користувача. Основний принцип - серверна частина не повинна зберігати дані карти, тому що це суперечить правилам PCI DSS.

PCI DSS (Payment Card Industry Data Security Standard) - Відповідність стандарту безпеки даних платіжних карток (PCI DSS) вимагається від усіх організацій, які зберігають, обробляють або передають дані власників карток Visa, включаючи фінансові установи, продавців і постачальників послуг.

Для того, щоб пройти цю перевірку також потрібно додати багато зусиль. Платформа Stripe має доступ до зберігання карт. Працює це наступним чином:

1. При створенні кошика серверна частина створює payment intent та client secret для кошику, де додає список продуктів та їх ціну та за потреби оновлює і передає на клієнта
2. Клієнтська частина за допомогою бібліотеки Stripe для React через інпути, наданні бібліотекою вводить дані карти та оплачує з payment Intent та client Secret.
3. Stripe оброблює платіж на своїй стороні, за потреби перевіряє оплату за допомогою 3ds.
4. Сервер ловить вебхук про успішну оплату і міняє статус оплати.

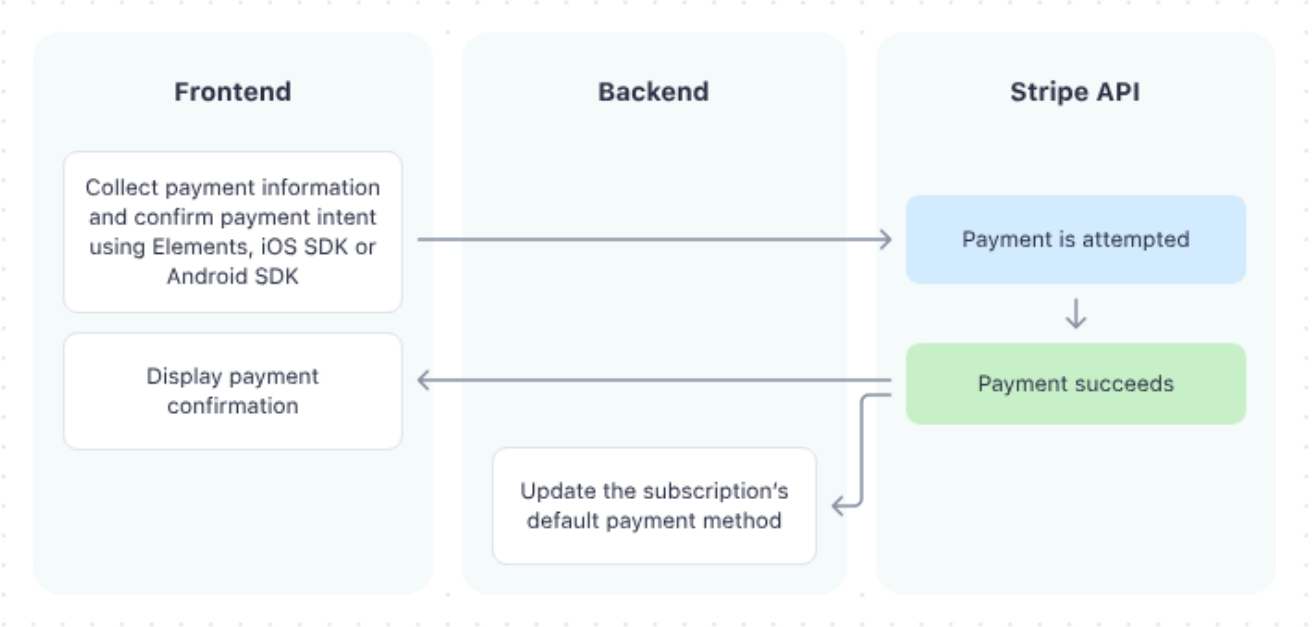

Рис. 3.13. Логіка роботи оплати через сервіс Stripe

Джерело: docs.stripe.com/billing/subscriptions/overview

Якщо у Stripe будуть проблеми зі створенням payment intent через внутрішні проблеми, ми повідомимо користувача про проблему, показавши йому помилку, яка висвітиться на екрані.

Лістинг 3.10. Створення або оновлення payment Intent

```
[Authorize]
[HttpPost]
public async Task < ActionResult < BasketDto >>
CreateOrUpdatePaymentIntent() {
  var basket = await _context.Baskets
    .RetrieveBasketWithItems(User.Identity.Name)
    .FirstOrDefaultAsync();

  if (basket == null) return NotFound();

  var intent = await _paymentService.CreateOrUpdatePaymentIntent(basket);

  if (intent == null) return BadRequest(new ProblemDetails {
    Title = "Problem creating payment intent"
  });

  basket.PaymentIntentId = basket.PaymentIntentId ?? intent.Id;
  basket.ClientSecret = basket.ClientSecret ?? intent.ClientSecret;

  _context.Update(basket);

  var result = await _context.SaveChangesAsync() > 0;

  if (!result) return BadRequest(new ProblemDetails {
    Title = "Problem updating basket with intent"
  });

  return basket.MapBasketToDto();
}
```

Stripe після успішної оплати посилає запит на наш ендпоїнт, щоб повідомити нас про статус транзакції. Для вебхуків також є свій ключ, спеціально виділений для вебхуків, які потрібно зберігати в захищеному місці.

Лістинг 3.11. Створення або оновлення payment Intent

```
[AllowAnonymous]
[HttpPost("webhook")]
public async Task < ActionResult > StripeWebhook() {
  var json = await new
```

```
StreamReader(HttpContext.Request.Body).ReadToEndAsync();

  var stripeEvent = EventUtility.ConstructEvent(json,
Request.Headers["Stripe-Signature"],
    _config["StripeSettings:WhSecret"]);

  var charge = (Charge) stripeEvent.Data.Object;

  var order = await _context.Orders.FirstOrDefaultAsync(x =>
    x.PaymentIntentId == charge.PaymentIntentId);

  if (charge.Status == "succeeded") order.OrderStatus =
OrderStatus.PaymentReceived;

  await _context.SaveChangesAsync();

  return new EmptyResult();
}
```

Для підключення Stripe на клієнтській частині ми використовуємо елемент-обгорту Elements, куди в пропси прокидуємо публічний ключ Stripe, що дозволяємо нам проводити оплату через бібліотеку.

Лістинг 3.12. Оплата за допомогою Stripe на клієнтській частині

```
async function submitOrder(data: FieldValues) {
  console.log(data);
  setLoading(true);
  const { nameOnCard, saveAddress, ...shippingAddress } = data;
  console.log(basket?.clientSecret, stripe, elements);
  if (!basket?.clientSecret || !stripe || !elements) return; // stripe is
not ready;
  try {
    const cardElement = elements.getElement(CardNumberElement);
    const paymentResult = await
stripe.confirmCardPayment(basket.clientSecret, {
      payment_method: {
        card: cardElement!,
        billing_details: {
          name: nameOnCard,
        },
      },
```

```
    });
    console.log(paymentResult);
    if (paymentResult.paymentIntent?.status === "succeeded") {
      const orderNumber = await agent.Orders.create({
        saveAddress,
        shippingAddress,
      });
      setOrderNumber(orderNumber);
      setPaymentSucceeded(true);
      setPaymentMessage("Thank you - we have received your payment");
      setActiveStep(activeStep + 1);
      dispatch(clearBasket());
      setLoading(false);
    } else {
      setPaymentMessage(paymentResult.error?.message || "Payment failed");
      setPaymentSucceeded(false);
      setLoading(false);
      setActiveStep(activeStep + 1);
    }
  } catch (error) {
    console.log(error);
    setLoading(false);
  }
}
```

Перед тим як сервер зможе приймати вебхуки зі сторони Stripe, вебхук потрібно спочатку налаштувати в адмін-панелі Stripe.

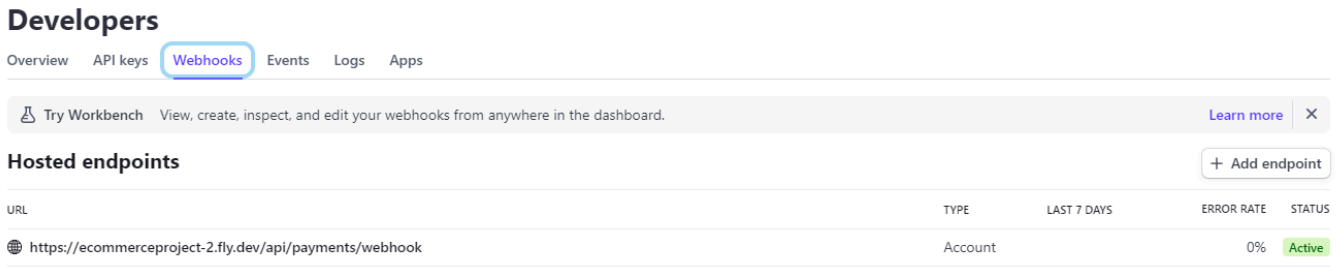

Рис. 3.14. Створення Endpoint для Stripe, щоб відсилати вебхуки про статуси оплати

Джерело: dashboard.stripe.com/test/webhooks

3.4. Реалізація деплойменту та хостингу

Для того щоб деплоїти наш контейнер у сервіс для хостингу Fly.io, для початку нам потрібно створити production версію контейнера, запушити його в docker Hub та в кінці оновлений контейнер задеплоїти на сервіс Fly.io. Для того щоб створити контейнер, нам потрібно зробити білд клієнтської частини та додати в папку API (Рис 3.18)

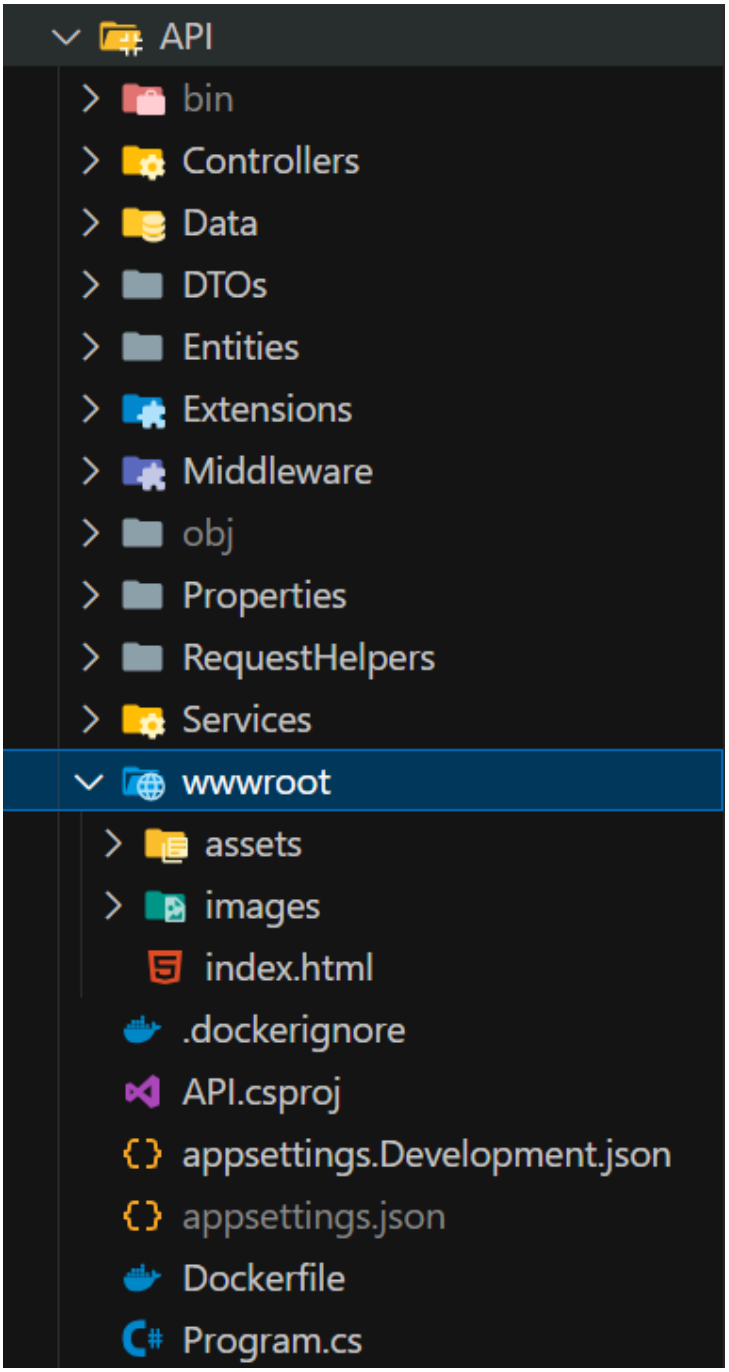

Рис. 3.15. Структура проекту з білдом клієнтської частини

Джерело: створено автором

Docker Hub дуже популярна бібліотека образів, яка використовується мільйонами користувачів по всьому світу, а також надає функціонал безпеки, де можна закрити образ від інших та сканування образів для виявлення помилок або злоякісних образів.

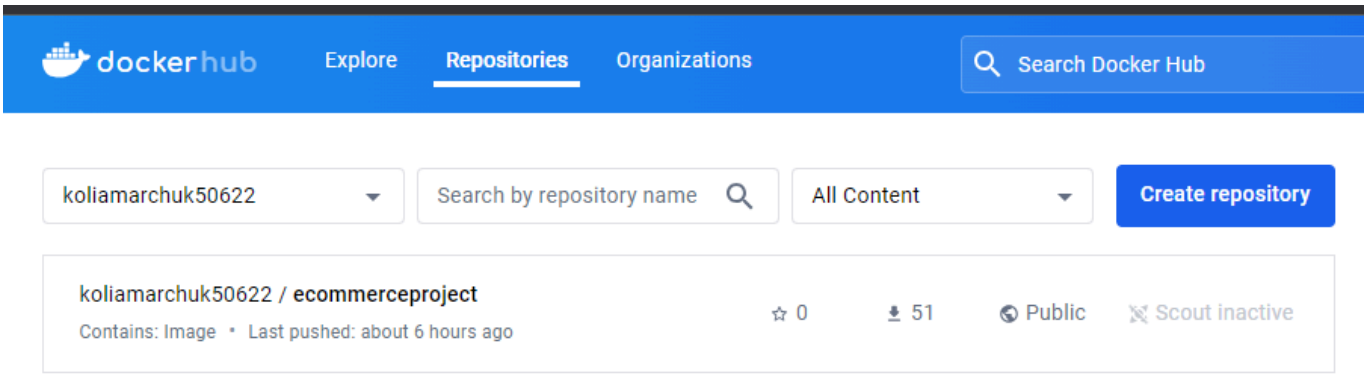

Рис. 3.16. Репозиторій в dockerHub

Джерело: hub.docker.com

Далі ми створюємо файл конфігурації контейнера docker.yaml(, який відповідає за налаштування контейнера.

Логіка роботи файлу:

1. Створюємо Docker-образ на основі офіційного образу Microsoft .NET SDK версії 7.0. Цьому образу присвоюється псевдонім *build-env*, який буде використовуватися всередині Dockerfile.
2. Встановлюємо робочий каталог у контейнері на /app. Саме сюди буде скопійовано та створено код програми.
3. відкриваємо порт 8080 контейнера. Це порт, на якому веб-сервер буде очікувати вхідних HTTP-запитів.
4. Копіюємо всі файли з розширенням .csproj (файли проекту C#) з поточного каталогу (де знаходиться Dockerfile) до каталогу /app всередині контейнера.
5. Виконуємо команду dotnet restore всередині контейнера. Ця команда відновлює пакети NuGet, на які посилаються файли проекту, роблячи їх доступними для створення програми.
6. Публікуємо production версію.

7. Створюємо новий образ на основі офіційного образу Microsoft .NET ASP.NET Core runtime версії 7.0. Цей образ містить мінімальне середовище, необхідне для запуску програми .NET.
8. Встановлюємо команду за замовчуванням та точку входу програми.

Лістинг 3.13. налаштування контейнеру в Docker

```
FROM mcr.microsoft.com/dotnet/sdk:7.0 as build-env
WORKDIR /app
EXPOSE 8080

COPY *.csproj ./
RUN dotnet restore

COPY . ./
RUN dotnet publish -c Release -o out

FROM mcr.microsoft.com/dotnet/aspnet:7.0
WORKDIR /app
COPY --from=build-env /app/out .
ENTRYPOINT [ "dotnet", "API.dll" ]
```

Далі залишилось створити Github Action для деплою контейнеру в Fly.io. Тут ми використовуємо секрети Github для токену репозиторію від DockerHub та акаунту Fly.io. У самому файлі ми логінимось у DockerHub та пушимо актуальний проект з Github. Після закінчення цього процесу ми деплоїмо контейнер в Fly.io

Лістинг 3.14. налаштування docker-push.yaml в Github workflows

```
name: docker-push

on:
  workflow_dispatch:
  push:
    branches:
      - 'main'

jobs:
  docker:
```

```
    runs-on: ubuntu-latest
    steps:
      -
        name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v2
      -
        name: Login to Docker Hub
        uses: docker/login-action@v2
        with:
          username: ${{ secrets.DOCKERHUB_USERNAME }}
          password: ${{ secrets.DOCKERHUB_TOKEN }}
      -
        name: Build and push
        uses: docker/build-push-action@v3
        with:
          context: "{{defaultContext}}:API"
          push: true
          tags: koliamarchuk50622/ecommerceproject
  deploy:
    needs: docker
    name: Deploy app
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - uses: superfly/flyctl-actions/setup-flyctl@master
      - run: flyctl deploy --remote-only
        env:
          FLY_API_TOKEN: ${{ secrets.FLY_API_TOKEN }}
```

Також нам потрібний файл конфігурації fly.toml для сервісу Fly.io. В цьому файлі є назва програма, назва контейнеру в Docker Hub який буде деплоїтись, публічні ключі, які додадуться в Environmental Variables при деплої контейнера, порт на якому будуть слухатись запити серверної частини, та налаштування машини, на якій буде лежати проект.

Лістинг 3.15. автоматично згенерований файл конфігурації fly.toml

```
# fly.toml app configuration file generated for ecommerceproject-2 on
2024-05-15T14:18:07+03:00
#
```

```
# See https://fly.io/docs/reference/configuration/ for information about
how to use this file.
#

app = 'ecommerceproject-2'
primary_region = 'waw'

[build]
  image = 'koliamarchuk50622/ecommerceproject'

[env]
  ASPNETCORE_URLS="http://+:8080"

StripeSettings__PublishableKey="pk_test_51PGAckB3oH7lkF0jGvi9ZJO0sYrqfhgGoU
uPXDYBFlMAHH4wZXCctCGirHAh43tB1OOI2Jciiu2I56dUKUXxJuPU00bXs9o8ea"
  Cloudinary__CloudName="dms5ihuwb"
  Cloudinary__ApiKey="185192147862928"

[http_service]
  internal_port = 8080
  force_https = true
  auto_stop_machines = true
  auto_start_machines = true
  min_machines_running = 0
  processes = ['app']

[[vm]]
  memory = '1gb'
  cpu_kind = 'shared'
  cpus = 1
```

Висновки до розділу 3

У цьому розділі було розроблено MVP версію інтернет-магазину з використанням React та Typescript на клієнтській частині, ASP.NET Core 7.0 та супутні бібліотеки на серверній частині. Також ми контенйнерували наш проект за допомогою DockerHub, інтегрували сервіси для проведення оплат Stripe, сервіс для хостингу Fly.io та сервіс збереження медіа-файлів Cloudinary.

ВИСНОВКИ

Під час виконання даної роботи ми розібрались, як створюються сучасні інтернет-магазини, проаналізували ситуацію на ринку, спроектували та розробили власну систему, яка складається з серверної та клієнтської частини веб-додатку.

Ми дослідили та вибрали стек технологій, які підходить для даної задачі

Клієнтська частина:

1. React.
2. Vite.
3. Redux.
4. Material UI та інші.

Серверна частина:

1. ASP.NET Core 7.
2. JWT Bearer.
3. EntityFramework Core.
4. EntityFramework Core Identity та інші.

Проект розміщений у відкритому доступі за допомогою сервісу контролю версій Github.

За допомогою даного стеку технологій ми розробили авторизацію, вибір товарів, фільтрацію, кошик, створення замовлень та їх оплату. Даний продукт має високу оптимізацію як клієнтської, так і серверної частини, високу безпеку, де користувачі можуть бути впевненими у захищеності їх даних, а також сучасний та зручний інтерфейс.

Даний проект має міцний фундамент та гнучкість у розширенні функціоналу. Для того, щоб даний проект можна було використовувати у реальному світі потрібно:

1. Зробити редизайн та ребрендинг проекту.
2. Підключити production версію Stripe для надання можливості проводити реальні транзакції з коштами користувачів.
3. Розширити можливості інвентаризації, зокрема додати можливість додавати та редагувати категорії продуктів.

4. Додати підтвердження акаунту через емейл.

Цей проект заклав міцний фундамент для створення успішного та конкурентоспроможного інтернет-магазину. Завдяки гнучкій архітектурі та чітко визначеним напрямкам розвитку, система має великий потенціал для розширення функціоналу та адаптації до мінливих потреб ринку.

Ми впевнені, що наш проект стане цінним внеском у розвиток сучасного електронного комерції, надаючи користувачам надійний та зручний інструмент для онлайн-шопінгу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. How to Implement JWT Token Authentication in .NET Core 6 URL: *https://shorturl.at/ngxgd* (дата звернення: 23.11.2023).
2. Speeding Up React Projects with Vite and SWC URL: *https://shorturl.at/LIu66* (дата звернення: 12.10.2023).
3. Introduction to Identity on ASP.NET Core URL: *https://shorturl.at/c43Gi* (дата звернення: 19.10.2023).
4. Документація react-hook-form URL: *https://www.react-hook-form.com/* (дата звернення: 13.10.2023).
5. Документація EntityFramework Core URL: *https://learn.microsoft.com/en-us/ef/core/* (дата звернення: 8.10.2023).
6. ASP.NET Core WepApi folder organization best practices URL: *https://shorturl.at/Qj431* (дата звернення: 7.10.2023).
7. Документація redux URL: *https://redux.js.org/usage/configuring-your-store* (дата звернення: 14.10.2023).
8. Документація Typescript URL: *https://shorturl.at/rQOts* (дата звернення: 6.10.2023).
9. Enable Cross-Origin Requests (CORS) in ASP.NET Core URL: *https://rb.gy/inkzym* (дата звернення: 21.10.2023).
10. React Js — 8 best practices + Folder Structure URL: *https://rb.gy/pqbsg1* (дата звернення: 7.10.2023).
11. Cloudinary Developer onboarding guide URL: *https://rb.gy/c7354l* (дата звернення: 8.10.2023).
12. Docker share the application URL: *https://rb.gy/v19h52* (дата звернення: 9.10.2023).
13. Fly.io deployment docs URL: *https://fly.io/docs/* (дата звернення: 10.10.2023).
14. Stripe документація сервер URL: *https://rb.gy/j8h0l4*(дата звернення: 11.10.2023).
15. Stripe документація клієнт URL: *https://t.ly/q1a-A* (дата звернення: 12.10.2023).

16. Create Data Transfer Objects (DTOs) URL: *https://t.ly/153G8* (дата звернення: 13.10.2023).
17. Automapper docs URL: *https://t.ly/HZ9Wv* (дата звернення: 13.10.2023).
18. How to: Read application settings URL: *https://t.ly/CgoPA* (дата звернення: 14.10.2023).

ДОДАТКИ

ДОДАТОК А

Посилання на репозиторій, розміщений у Github

https://github.com/devman-3000/Marchuk_Ecommerce

ДОДАТОК Б

Посилання на сайт, розміщений за допомогою Fly.io

https://ecommerceproject-2.fly.dev/catalog

ДОДАТОК В

Лістинг 1. Приклад лістингу коду обгортки товарів в каталозі:

```
import { Grid } from "@mui/material";
import { Product } from "../../app/models/product";
import { useAppSelector } from "../../app/store/configureStore";
import ProductCard from "./ProductCard";
import ProductCardSkeleton from "./ProductCardSkeleton";

interface Props {
  products: Product[];
}

export default function ProductList({ products }: Props) {
  const { productsLoaded } = useAppSelector((state) => state.catalog);
  return (
    <Grid container spacing={4}>
      {products.map((product) => (
        <Grid key={product.id} item xs={4}>
          {!productsLoaded ? (
            <ProductCardSkeleton />
          ) : (
            <ProductCard product={product} />
          )}
        </Grid>
      ))}
    </Grid>
  );
}
```

ДОДАТОК Г

Лістинг 2. Приклад коду сервісу токенізації:

```
namespace API.Services {
  public class TokenService {
    private readonly UserManager < User > _userManager;
    private readonly IConfiguration _config;

    public TokenService(UserManager < User > userManager, IConfiguration
config) {
      _config = config;
      _userManager = userManager;
    }

    public async Task < string > GenerateToken(User user) {
      //claims
      var claims = new List < Claim > {
        new Claim(ClaimTypes.Email, user.Email),
        new Claim(ClaimTypes.Name, user.UserName)
      };

      var roles = await _userManager.GetRolesAsync(user);
      foreach(var role in roles) {
        claims.Add(new Claim(ClaimTypes.Role, role));
      }
      var key = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(_config["JWTSettings:TokenKey"]
));
      var creds = new SigningCredentials(key,
SecurityAlgorithms.HmacSha512);
      var tokenOptions = new JwtSecurityToken(
        issuer: null,
        audience: null,
        claims: claims,
        expires: DateTime.Now.AddDays(7),
        signingCredentials: creds
      );
      return new JwtSecurityTokenHandler().WriteToken(tokenOptions);
    }
  }
}
```