МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ЮРИДИЧНА АКАДЕМІЯ»
ФАКУЛЬТЕТ КІБЕРБЕЗПЕКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«ВЕБСАЙТ ЕЛЕКТРОННОЇ КОМЕРЦІЇ»

Рівень «бакалавр»

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Виконав здобувач 4 курсу

Старенченко Іван Федорович

Керівник: к.пед.н., доцент

Логінова Наталья Іванівна

Рецензент к.т.н., доцент

Трофименко Олена Григорівна

Одеса – 2024

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ЮРИДИЧНА АКАДЕМІЯ»
Факультет кібербезпеки та інформаційних технологій
Кафедра інформаційних технологій
Галузь знань 12 Інформаційні технології
Спеціальність 121 «Інженерія програмного забезпечення»
Назва освітньої програми «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри інформаційних технологій
доцент Наталія Логінова _____
«____»_____ 20___ року

ЗАВДАННЯ
на кваліфікаційну (бакалаврську) роботу
здобувачу Старенченку Івану Федоровичу

1. Тема «Вебсайт електронної комерції»
   Керівник роботи: к.пед.н., доцент Логінова Наталія Іванівна
   затверджені наказом НУ «ОЮА» № 809-06 від «02» квітня 2024 року
2. Дата видачі завдання «15» вересня 2023 року.
3. Строк подання здобувачем роботи «20» травня 2024 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Формування функціональних вимог	10.10.2023	
2	Аналіз та вибір засобів розробки	30.10.2023	
3	Розробка архітектури вебзастосунку	20.11.2023	
4	Проєктування бази даних	25.12.2023	
5	Розробка серверної частини	12.02.2024	
6	Розробка клієнтської частини	19.03.2024	
7	Тестування функціональності сайту	09.04.2024	
8	Оформлення звітної частини	19.04.2024	
9	Подача електронного варіанту роботи для перевірки на плагіат	20.05.2024	
10	Допуск роботи до захисту завідуючим кафедри		
11	Захист роботи		

Здобувач _____
(підпис) (прізвище та ініціали)

Керівник роботи _____
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Старенченко І. Ф. Вебсайт електронної комерції – Кваліфікаційна робота бакалавра за спеціальністю 121 «Інженерія програмного забезпечення», освітньою програмою «Інженерія програмного забезпечення».

Кваліфікаційна робота викладена на 76 сторінках основного тексту, містить 3 розділи, 63 рисунки, 1 таблицю, 1 додаток, 23 використаних джерела.

Мета роботи – проаналізувати та визначити основні аспекти розробки вебсайту електронної комерції.

Об’єкт дослідження – вебсайт електронної комерції.

Предмет дослідження – засоби розробки вебсайту електронної комерції.

У кваліфікаційній роботі запроєктований та розроблений вебсайт електронної комерції з використанням передових технологій, таких як React JS, MSSQL, Dot.net та Microsoft.EntityFrameworkCore. В першому розділі проаналізована предметна області та вибрані засоби розробки. Визначені маркетингові інструменти створення та просування вебсайту. Другий розділ присвячений проєктуванню інтернет-магазину. В третьому розділі роботи описано розробка вебсайту електронної комерції. Здійснено тестування та виконана оцінка якості створеного вебсайту електронної комерції.

Ключові слова: вебсайт, електронна комерція, інтернет-магазін, React JS, MSSQL, Dot.net, Microsoft.EntityFrameworkCor, Redux.

ABSTRACT

Starenchenko I. F. E-commerce website – Bachelor's qualification work on specialty 121 "Software engineering", educational program "Software engineering".

The qualification work is laid out on 76 pages of the main text, contains 3 chapters, 63 figures, 1 table, 1 appendix, 23 used sources.

The purpose of the work is to analyze and determine the main aspects of the development of an e-commerce website.

The object of the study is an e-commerce website.

The subject of the study is the means of developing an e-commerce website.

In the qualifying work, an e-commerce website was designed and developed using advanced technologies such as React JS, MSSQL, Dot.net and Microsoft.EntityFrameworkCore. The first section analyzes the subject area and selected development tools. Defined marketing tools for website creation and promotion. The second section is devoted to the design of the online store. The third section of the work describes the development of an e-commerce website. Testing and quality assessment of the created e-commerce website was carried out.

Keywords: website, e-commerce, online store, React JS, MSSQL, Dot.net, Microsoft.EntityFrameworkCor, Redux.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

AJAX – Asynchronous Javascript and XML

API – Application Programming Interface

CRUD – Create Read Update Delete

CSR – Client–Side Rendering

CSS – Cascading Style Sheets

GIT – Global Information Tracker

HOC – High Order Component

HTML – HyperText Markup Language

HTTP – HyperText Transfer Protocol

IDE – Integrated Development Environment

JSON – JavaScript Object Notation

JWT – JSON Web Token

MSSQL – Microsoft SQL Server

REST – Representational State Transfer

SEO – Search Engine Optimization

SMM – Social Media Marketing

SPA – Single Page Application

SSMS – Server Management Studio

SWOT – Strengths, Weaknesses, Opportunities, Threats

UI – user interface

UML – Unified Modeling Language

URL – Uniform Resource Locator

VS – Visual Studio

ВСТУП

Інтернет-магазин – це одна з найпоширеніших форм електронної комерції, яка дозволяє користувачам легко здійснювати покупки онлайн. Завдяки швидкому розвитку технологій та поширенню Інтернету, онлайн-торгівля стала необхідністю для бізнесу будь-якого масштабу. Відповідно збільшується попит на розробку надійних й ефективних інтернет-магазинів. Розробка вебсайту електронної комерції може значно полегшити доступ споживачів до товарів та послуг, забезпечуючи зручний та швидкий спосіб здійснення покупок. Отже, дослідження цієї теми допоможе вдосконалити навички розробки та просунути електронну комерцію на новий рівень.

Проєкт присвячений розробці інтернет-магазину з використанням таких технологій як:

1. React JS використовується для побудови зручного та інтерактивного користувацького інтерфейсу.

2. MSSQL виступає в ролі бази даних, що забезпечує зберігання та керування даними.

3. Dot.net використовується для розробки серверної частини, яка відповідає за бізнес-логіку, обробку запитів та взаємодію з базою даних.

4. Microsoft.EntityFrameworkCore використовується як інструмент для доступу до бази даних та її моделювання.

Даний стек технологій дозволяє створити надійний, масштабований та зручний у використанні інтернет-магазин.

Мета дослідження – проаналізувати та визначити основні аспекти розробки вебсайту електронної комерції.

Для досягнення поставленої мети необхідно розв’язати такі *завдання*:

1. Розробити клієнтську частину інтернет-магазину.

2. Розробити серверну частину для взаємодії з базою даних.

3. Впровадити функціонал для виведення товарів з бази даних, аутентифікації та реєстрації користувача, можливість коментувати товар, пошук

по назві товару, керування кошиком та бажаними товарами, особистими даними.

4. Забезпечити зручність використання та інтуїтивно зрозумілий інтерфейс для користувачів.

Об'єктом дослідження є вебсайт електронної комерції.

Предмет дослідження – засоби розробки вебсайту електронної комерції.

В кваліфікаційній роботі використовувалися загальнонаукові та спеціальні *методи дослідження* таки, як аналіз, моделювання, порівняння, класифікації тощо.

Практична значущість результатів дослідження полягає в можливості залучення більшої аудиторії клієнтів, забезпеченні зручного та швидкого доступу до товарів і послуг, підвищенні продажів та покращенні взаємодії з клієнтами. Вебсайт електронної комерції дозволяє розширити географію продажів, підвищити конкурентоспроможність та оптимізувати процеси керування товаром і обліку клієнтської бази.

Структура кваліфікаційної роботи складається зі вступу, основної частини з трьох розділів, висновків, списку використаних джерел та одного додатку. Робота викладена на 83 сторінках, містить 63 рисунки, 1 таблицю. Перелік посилань має 23 джерела.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИБІР ЗАСОБІВ РОЗРОБКИ

1.1 Специфіка розробки інтернет-магазину

Специфіка розробки інтернет-магазину полягає у створенні онлайн-платформи, яка дозволяє клієнтам придбати товари або послуги в інтернет-середовищі. Основними етапами розробки такого сайту є:

1. Проєктування – важливо створити зручний та інтуїтивно зрозумілий інтерфейс, який сприятиме швидкому пошуку та оформленню замовлень.

2. Функціональність – вебсайт повинен мати усі необхідні функції для зручної навігації та оформлення замовлень, включаючи кошик покупок, систему оплати, фільтри для пошуку тощо.

3. Безпека – важливо забезпечити захист особистих даних клієнтів та безпеку платежів на сайті, щоб уникнути можливих кібератак.

4. Сумісність – вебсайт має бути оптимізований для роботи на різних пристроях.

Проєктування вебсайту потрібно розпочати з аналізу технічних аспектів розробки:

1. FronA застосунків, що значно підвищує зручність використання і привабливість для користувачів.

Для того щоб зробити застосунок інтерактивним, тобто додати можливість відстежувати зміни в компонентах та керувати ними було використано Redux бібліотеку, яка забезпечує передбачуване та зручне керування глобальним станом (state manegment) застосунку. Вона надає змогу легко відслідковувати та керувати змінами в React компонентах [1].

Redux-form було використано для додання валідації на форми, та збирання вписаної у неї інформації. Redux-form – це невелика, але потужна бібліотека, яка дає набір інструментів, що дозволяє працювати з валідацією, обробкою та відправкою даних. Це чудовий спосіб керування формами, створений на базі Redux. Це компонент вищого порядку (HOC), котрий використовує React-Redux,

щоб гарантувати, що HTML-формы в React використотвують Redux для храніння всього свого стану [2].

За допомогою Axios бібліотеки, була реалізована взаємодія з базою даних. Бібліотека Axios здебільшого використовується для надсилання асинхронних HTTP-запитів до кінцевих точок REST. Ця бібліотека дуже корисна для виконання операцій CRUD [3].

2. Backend на основі Dot.net і Microsoft.EntityFrameworkCore: для створення серверної частини інтернет-магазину було використана платформа Dot.net із застосуванням Microsoft.EntityFrameworkCore для роботи з базою даних. Це надійне і потужне рішення, що забезпечує швидке та ефективне опрацювання запитів користувачів і зберігання даних.

3. База даних MSSQL: ця база даних була використана для зберігання інформації про товари і користувачів. Вона забезпечує надійне та ефективне зберігання даних, а також можливість виконувати різні операції для їх обробки.

Функціональні можливості:

– *Реєстрація та авторизація користувачів*: користувачі мають змогу створювати нові облікові записи та входити до системи задля використання всіх недоступних функцій для неавторизованих користувачів.

– *Перегляд і пошук товарів*: користувачі можуть переглядати асортимент товарів які розбиті по категоріям, а також виконувати пошук за назвою товару.

– *Додавання товарів до кошика та до бажаних товарів*: клієнти переглядаючи каталог усіх товарів мають змогу додавати їх до своїх бажаних , а перейшовши до сторінки самого товару можуть. покласти товари до кошика, переглянути їхній опис і здійснити покупку. Також на сторінці товару можна змінювати його параметри, це також буде враховуватися при додаванні до кошика. На сторінці кошику можна змінювати кількість вибраного товару. Від кількості товару буде змінюватись кінцева ціна за продукт.

– *Додавання та редагування власної інформації на сторінці клієнта:* авторизований клієнт має право зайти до свого власного кабінету та додати власну інформацію та подалі буде мати можливість її редагувати.

– *Додавання та видалення коментарів до товару*: авторизований клієнт має право залишити свій відгук до будь-якого товару та подалі має можливість їх видаляти. Клієнт може видаляти лише ті коментарі, які залишив зі свого аккаунта.

– *Можливість запускати застосунок на всіх типах пристроїв*: адаптивність застосунку дає можливість без зайвих незручностей переглядати та користуватися інтернет-магазином як з персонального комп’ютера так і з всіх інших пристроїв.

1.2 Аналіз наявних аналогів

У мережі інтернет існує велика кількість вебсайтів інтернет-магазинів різних спеціалізацій. Проаналізуємо найбільш поширені.

Amazon – це один із провідних інтернет-магазинів у світі, що пропонує широкий асортимент товарів у різних категоріях. Вважається однією з «Великої п'ятірки» американських технологічних компаній [4].

Переваги:

1. Великий вибір товарів і гнучка система фільтрації.
2. Швидка доставка та висока якість обслуговування клієнтів.
3. Продуктивна система рекомендацій і персоналізації.

Недоліки:

1. Іноді складно орієнтуватися у великому обсязі товарів.
2. Можливість отримання неякісних або неправильних товарів.

eBay – це американський онлайн-аукціон, де користувачі можуть купувати і продавати різноманітні товари. Продажі відбуваються або через онлайн-аукціони, або через миттєві продажі «купи зараз» [5].

Переваги:

1. Можливість придбати товари за зниженими цінами на аукціоні;
2. Широкий асортимент товарів у різних категоріях.

Недоліки:

1. Ризики придбання товарів низької якості або підробок;
2. Визначення реальної вартості товару через аукціонну систему становить певні труднощі.

AliExpress – це інтернет-платформа, що пропонує продукцію міжнародним онлайн-покупцям від китайських виробників за доступними цінами [6].

Переваги:

1. Низькі ціни на товари і безкоштовна або недорога доставка;

2. Широкий вибір товарів у різних категоріях.

Недоліки:

1. Довгий час доставки товарів, особливо в країни за межами Китаю;

2. Можливість отримати товар низької якості або зіткнутися з проблемами під час доставки.

Для прийняття рішення щодо створення вебсайту інтернат-магазину виконуємо SWOT-аналіз, розглянутих інтернет-магазинів (табл. 1.1).

Таблиця 1.1 – SWOT-аналіз конкурентів

Аспекти	**Amazon**	**eBay**	**AliExpress**
Сильні сторони	Великий вибір товарів та швидка доставка	Можливість отримати товари за зниженими цінами, гнучка система аукціонів	Низькі ціни та широкий вибір асортименту
Слабкі сторони	Можливість отримати неякісний товар, складно зорієнтуватися в інтерфейсі застосунку через занадто великий асортимент	Ризик купівлі неякісних товарів, через аукціонну систему, складно визначити реальну вартість товару	Довгий час доставки, ризик отримання низькоякісних товарів
Можливості	Розвиток системи рекомендацій та поліпшення інтуїтивності інтерфейсу	Розширення географії доставки, покращення системи аукціонів	Збільшення асортименту товарів, покращення якості товарів
Загрози	Зростання конкуренції	Зростання конкуренції	Зростання конкуренції

Узагальнюючи результати порівняння, можна дійти висновку, що розробка нового програмного продукту має потенціал для успіху, оскільки може пропонувати унікальні рішення та розв'язувати проблеми, які присутні в наявних аналогів.

1.3 Розробка функціональних вимог та виявлення варіантів використання

Розробка інтернет-магазину, передбачає створення платформи, спрямованої на забезпечення ефективної взаємодії між продавцями та покупцями. Цей магазин стане місцем, де продавці матимуть можливість презентувати свою продукцію та залучати більше клієнтів, а покупці – шукати та замовляти потрібні їм товари, забезпечуючи собі зручність та доступність покупок.

Завдяки його функціоналу, клієнти зможуть швидко знайти потрібний товар, порівняти ціни та характеристики, а також здійснити покупку в кілька кліків.

Основними користувачами цього інтернет-магазину будуть продавці та покупці. Продавці матимуть можливість реєструватися на платформі та додавати свої товари до каталогу, представляючи їх широкому колу клієнтів. Покупці ж зможуть знаходити потрібні товари, додавати їх до кошика та здійснювати покупки відомих брендів та виробників.

Крім основних ролей продавців та покупців, важливо також передбачити роль адміністратора. Адміністратор буде відповідальним за адміністрування та підтримку роботи магазину, включаючи керування товарами та користувачами, а також вирішення будь-яких технічних проблем та конфліктів.

З огляду на важливість кожної з цих ролей у роботі інтернет-магазину, їхнє належне функціонування та взаємодія між собою є вирішальним для успішної реалізації цього проєкту.

Функціональні вимоги:

– *Реєстрація нового користувача:* користувач може створити новий обліковий запис записавши свій пароль та електронну адресу.

– *Авторизація клієнту:* після створення облікового запису клієнт матиме змогу у будь-який час заходити в свій аккаунт.

– *Запис персональних даних та зміна їх:* у своєму персональному аккаунті клієнт має змогу передивлятися свої дані та має змогу в будь-який час змінити їх.

– *Переглядати товари за категоріями*: клієнт має змогу переглядати товари які знаходяться за різними категоріями

– *Переглядати кожен товар окремо:* авторизований клієнт має змогу відкрити сторінку будь-якого товару

– *Вибір параметрів товару:* зайшовши на сторінку товару, клієнт має змогу не тільки передивлятися характеристики товару, а й вибирати окремі параметри, як-от: колір, розмір тощо.

– *Додавання товару до кошику:* на сторінці товару клієнт має змогу додавати вподобаний товар у кошик.

– *Додавання товару до списку бажаного:* на сторінці каталогу будь-який клієнт може додавати товар до списку бажаних, але до кошику зможе додати тільки авторизований клієнт.

– *Можливість залишати та видаляти коментарі:* перейшовши на сторінку товару, клієнт має змогу залишити свій коментар та за потреби видалити його.

– *Можливість переглядати застосунок на будь-якому пристрої без зайвих незручностей:* адаптивність застосунку дозволяє клієнту користуватись ним на будь-якому пристрої.

– *Можливість пошуку по назві товару:* на сторінці каталогу клієнт може здійснити пошук по ключовому слову в спеціальній для цього формі, та знайти товар з таким ім’ям.

– *Надати можливість адміністратору не виходячи з застосунку додавати товари та видаляти за потребою:*

1) *Додавання нового товару :* адміністратор увійшовши до адмін панелі має можливість додавати нові товари, вказуючи їхні назви, категорії, ціни, описи та інші характеристики. Після заповнення всіх необхідних полів, товар додається до каталогу магазину

2) *Редагування інформації про товар :* адміністратор може вносити зміни до інформації про вже існуючі товари, такі як ціна, опис, категорія та інші характеристики. Це дозволяє вчасно актуалізувати інформацію про товари в магазині та коригувати їхні параметри відповідно до зміни умов або попиту покупців.

3) *Видалення товару з каталогу :* у разі потреби адміністратор може видаляти товари з каталогу магазину. Це може бути необхідно в разі виходу товару з асортименту, припинення його продажу або інших обставин. Після видалення товару він стає недоступним для покупки користувачами.

Варіанти використання:

1) *Реєстрація нового користувача:* перейшовши в застосунок, клієнт створює новий обліковий запис, введучи свої особисті дані, такі як пароль та електронна адреса.

2) *Авторизація клієнта:* вже зареєстрований клієнт переходить до сторінки авторизації та вводить ті дані, які його просили вписати при реєстрації.

3) *Запис персональних даних та зміна їх:* авторизований клієнт переходить до особистого кабінету та на вкладці з його інформацією, він може додати інформації про себе, таку як ім’я, номер телефону і так далі. Також по бажанню він в змозі цю інформацію редагувати.

4) *Переглядати товари за категоріями:* клієнт переходить за посиланням до сторінки каталогу чоловічого одягу або жіночого.

5) *Переглядати кожен товар окремо:* авторизований клієнт через каталог переходить до обраного їм товару.

6) *Вибір параметрів товару:* на сторінці товару клієнт вибирає параметри кольору та розміру одягу які йому до вподоби.

7) *Додавання товару до кошику:* клієнт натиском спеціальної кнопки додає вподобаний товар з вибраними параметрами до кошику.

8) *Додавання товару до списку бажаного:* на сторінці каталогу клієнт додає вподобаний товар до списку бажаного, але без можливості додати його до кошику без попередньої авторизації.

9) *Залишати та видаляти відгуки на товар:* клієнт на сторінці товару залишає свій відгук, та по бажанню видаляє його.

10) *Перегляд застосунку на будь-якому пристрої без зайвих незручностей:* клієнт здійснює пошук товару на будь-якому наявному пристрої.

11) *Пошук по назві товару:* клієнт вписує назву товару до форми пошуку.

12) *В адмін. меню застосунку, додавати товари, оновлювати та видаляти за потребою:*

1. *Адмінітратор* додає товар;
2. *Адмінітратор* оновлює дані про товар;
3. *Адмінітратор* видаляє товар.

1.4 Вибір засобів розробки

Для розробки *Backend* частини інтернет-магазину було вибрано мову програмування *C#*, візуальне середовище *Visual Studio 2022* та середовище для баз даних *SQL Server Management Studio 2019*.

Visual Studio 2022 надає розробникам потужний інструмент для створення, налагодження та вдосконалення проєктів.

Для роботи з базою даних використовується *SQL Server Management Studio 2019*, а також фреймворк *Entity Framework* для зручної роботи з даними. Для *Frontend* частини було використано *Visual Studio Code.*

1.4.1 Засоби розробки backend частини

Середовище розробки Visual Studio 2022

Visual Studio 2022 – це інтегроване середовище розробки (IDE) для розробки програмного забезпечення на платформі *.Net*. Вибір *Visual Studio 2022* для розробки backend-частини створюваного застосунку.

Підтримка *.Net:* Visual Studio надає повну підтримку для розробки *.Net*-застосунків, зокрема *C#, ASP.Net* та інші технології, що відмінно забезпечує високу продуктивність і надійність розробки.

Широкі можливості: Visual Studio має розширений набір інструментів для розробки, тестування та налагодження backend застосунків. Він надає можливості для створення вебслужб, Api, консольних застосунків та інших компонентів backend.

Інтеграція з *Git: Visual Studio* має вбудовану підтримку *Git*, що дає можливість з легкістю керувати версіями коду та спільно працювати з іншими розробниками.

Багатофункціональність: IDE має широкий набір інструментів для автоматизації рутинних завдань, зокрема підказки коду, автоматичне доповнення, налагодження та інше.

1.4.2 Засоби розробки бази даних застосунку

Server Management Studio (SSMS) – це інтегроване середовище для керування будь-якою інфраструктурою SQL [7]. Вибір *SSMS* для розробки та адміністрування баз даних *SQL Server* вдалий по багатьом причинам.

Одна з найголовніших причин, це надійність і продуктивність. SSMS – це офіційний інструмент від Microsoft для роботи з *SQL Server*, що відмінно забезпечує оптимізацію та надійніть роботи з базою даних.

Багатофункціональність: SSMS надає широкий спектр функцій, зокрема можливості редагування та виконання SQL-запитів, розробку процедур і функцій, налаштування безпеки, перегляд і дослідження плану виконання запитів і багато іншого.

SSMS має інтеграцію з Microsoft Azure, тому він з легкістю інтегрується з хмарними роботами Microsoft Azure, що надає можливість з легкістю працювати з базами даних, розгорнутими в них.

Інтуїтивно зрозумілий інтерфейс користувача: Користувацький інтерфейс SSMS є простим у застосуванні, що полегшує роботу з базою даних навіть для початківців.

Безкоштовне програмне забезпечення: SSMS є безкоштовним ПЗ, що відмінно забезпечує доступність для всіх користувачів без додаткових витрат.

1.4.3 Засоби розробки frontend частини

Середовище розробки Visual Studio Code:

Visual Studio Code (VS Code) – це оптимізований редактор коду з підтримкою таких операцій розробки, як налагодження, виконання завдань і контроль версій, що розроблений компанією Microsoft. Він надає розширені можливості для більш комфортної розробки програмного забезпечення на різних мовах програмування.[8]

Швидкість і продуктивність: Навіть на повільних комп'ютерах Vs Code працює досить швидко і забезпечує високий рівень продуктивності через швидкий запуск та інтерфейс.

Розширення: VS Code має величезну бібліотеку розширень, за допомогою яких можна розширити його функціональність для роботи з різними мовами програмування, технологіями та фреймворками.

Інтеграція з Git: За рахунок такої інтеграції , VS Code має можливість без труднощів керувати версіями коду і спільно працювати з іншими розробниками.

Багатофункціональність: VS Code має безліч корисних функцій, наприклад:

1. Підказки в коді;
2. Автоматичне доповнення коду;
3. Вбудовані інструменти для розроблення вебсторінок.

Безкоштовність і відкритий код: VS Code - це повністю безкоштовне програмне забезпечення, який має відкритий вихідний код, що дає можливість будь-якому розробнику застосовувати та модифікувати його без обмежень.

1.5 Маркетингові інструменти в створенні та просуванні вебсайту

Мета даного дослідження полягала у визначенні видів маркетингових інструментів, що можуть працювати для програмного продукту та можуть допомогти оптимізувати витрати для досягнення необхідних показників.

Сучасний стан ринку програмних продуктів розвивається, що спонукає до пошуку та придбання різноманітних маркетингових інструментів, які допомагають продавати програмне забезпечення. На сьогоднішній день більшість програмних продуктів або мають такі інструменти, або створюють їх. Також не обійтися без знання поведінки клієнтів і потенційних продажів для майбутнього

програмного забезпечення, а також поточних продажів на ринку. Головною метою кожного програмного забезпечення завжди є залучення якомога більше клієнтів, і маркетинг є основним інструментом в таких ситуаціях.

Далі будуть описані необхідні маркетингові інструменти, що будуть працювати для програмного продукту, та просувати його.

1.5.1 Карта клієнта

Демографічні дані:

Вік споживачів дозволяє розділити їх на різні категорії та адаптувати пропозиції інтернет-магазину до конкретних вікових груп.

Стать – розрізняючи клієнтів за статтю, можна створювати персоналізовану рекламу та пропозиції, які відповідають їхнім інтересам.

Місце проживання – знання місця проживання допомагає у розрахунку вартості доставки та адаптації асортименту до місцевих особливостей та попиту.

Історія покупок:

Список попередніх покупок – дозволяє аналізувати звички клієнта та надавати рекомендації щодо подібних товарів.

Витрати – дозволяє оцінити середній чек та здійснювати персоналізовані пропозиції на основі історії витрат.

Частота покупок – визначає активність клієнта та дозволяє створювати програми лояльності та знижки.

Переваги та інтереси:

Вподобання – дозволяє персоналізувати рекламу та пропозиції, відповідно до вподобань клієнта,

Інтереси – допомагає адаптувати асортимент та рекламу до інтересів клієнта.

Потреби споживачів – аналізує потреби та побажання клієнтів для покращення сервісу та асортименту товарів.

1.5.2.Аналіз ринку

Дослідження конкурентів:

Аналіз продуктів та послуг конкурентів – допомагає розробити унікальну пропозицію товарів та послуг.

Їхні ціни – визначає конкурентоспроможність цін.

Маркетингові стратегії – дозволяє визначити ефективні методи маркетингу та реклами.

Визначення цільової аудиторії:

Ідентифікація ключових сегментів ринку та їхніх потреб – дозволяє адаптувати асортимент та маркетингові стратегії до потреб цільової аудиторії.

1.5.3 Аналіз споживачів

Вивчення поведінки споживачів:

Звички покупців – допомагає прогнозувати та адаптувати асортимент до потреб клієнтів.

Способи прийняття рішень – визначає ефективні методи маркетингу та реклами.

Взаємодія з інтернет-магазином – дозволяє покращувати користувацький досвід та сервіс.

Збір фідбеку:

Оцінка задоволеності клієнтів – дозволяє виявляти слабкі місця та покращувати сервіс.

Виявлення слабких місць та можливостей для покращення – стимулює постійне вдосконалення сервісу та пропозицій.

1.5.4 SEO (Пошукова оптимізація):

Вибір ключових слів:

Підбір та оптимізація ключових слів для підвищення видимості застосунку в пошукових системах.

Оптимізація контенту:

Створення унікального та цікавого контенту оптимізованого під ключові запити.

1.5.5 SMM (Соціальні медіа маркетинг):

Створення контенту:

Розробка публікацій – створення цікавого та залучаючого контенту для соціальних медіа з використанням фотографій, графіки, текстів та відео.

Відео та інший контент для соціальних медіа – створення відеороликів, сторіз, анімацій та іншого візуального контенту для привертання уваги аудиторії.

Взаємодія з аудиторією:

Розробка публікацій – створення цікавого контенту для соціальних медіа з використанням фотографій, графіки, текстів та відео.

Відео та інший контент для соціальних медіа – створення відеороликів, сторіз, анімацій та іншого візуального контенту для привертання уваги аудиторії.

1.5.6 Email-маркетинг

Розсилка розсилок:

Надсилання інформаційних листів – регулярна розсилка новин, акцій, спеціальних пропозицій та інших повідомлень на адреси електронної пошти клієнтів;

Рекламні акції та спецпропозиції – проведення спеціальних промо-кампаній та розсилка персоналізованих пропозицій для підвищення продажів.

Автоматизація:

Налаштування автоматичних листів – створення автоматизованих листів для привітання нових підписників, нагадування про незавершені покупки, подяки за замовлення тощо.

1.5.7 Аналіз вебаналітики:

Моніторинг трафіку:

Вимірювання кількості відвідувань – аналіз кількості відвідувачів сайту за різними періодами часу та джерелами трафіку.

Конверсії та інші метрики – визначення конверсійних цілей та вимірювання їхньої ефективності, а також аналіз інших метрик, таких як середній час сесії, відмови та інші.

Вивчення поведінки користувачів:

Аналіз сторінок – вивчення популярних сторінок сайту, сторінок виходу та глибини перегляду для виявлення основних шляхів навігації користувачів;

Час проведений на сайті – визначення середнього часу, який користувачі проводять на сайті, для оцінки їхньої зацікавленості та взаємодії з контентом.

1.6 Висновки до розділу

В розділі здійснено аналіз предметної області та обґрунтовано вибір засобів розробки для створення програмного продукту – інтернет-магазину.

Описано специфіку розробки програмного продукту. Розглянуто технічні аспекти розробки та доцільність вибраних засобів. Проаналізовано функціональні можливості продукту, а саме можливість пошуку товарів, додавання їх у кошик, додавання їх до бажаних та інше.

Проведено аналіз наявних аналогів інтернет-магазинів. Оцінено їх об'єктивні переваги та недоліки, а також складено SWOT-аналіз конкурентів.

Описано функціональні вимоги до програмного продукту та виявлено варіанти використання цього продукту. Даний аналіз допоміг чітко визначити функціональні можливості застосунку та його інтерфейс для користувачів. Обґрунтовано вибір засобів розробки для різних частин програмного продукту: бази даних, backend і frontend.

Розглянуто маркетингові інструменти в створенні та просуванні програмного продукту, а саме: карта клієнта, аналіз ринку, аналіз споживачів, *SEO, SMM* тощо. Використання даних інструментів допоможе залучити аудиторію та забезпечити вдале позиціонування продукту на ринку.

В результаті проведеного аналізу і вибору засобів розробки вдалося чітко сформулювати вимоги до програмного продукту. Розробка інтернет-магазину засобами React JS, MSSQL, Dot.net та Microsoft.EntityFrameworkCore є доцільною та перспективною задачею, що відповідає сучасним вимогам інтернет-бізнесу.

2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ ІНТЕРНЕТ-МАГАЗИНУ

2.1 Інформаційна модель інтернет-магазину

Розробка та функціонування онлайн-магазину в більшій степені залежить від *інформаційної моделі продукту*.

Ця модель дозволяє організувати та описати всю інформацію в магазині, включаючи дані про користувачів, товари та інші дані.

Інформаційна модель інтернет-магазину складається з основних об'єктів, а також їх взаємозв’язків:

Сутності (об'єкти):

Товари: інформація про товари включає назву, опис, ціну, категорію, наявність на складі та його параметри. Кожен товар має унікальний ідентифікатор.

Користувачі: клієнти є центральною складовою інтернет-магазину. Інформація про користувачів включає їх особисті дані, контактну інформацію, адреси доставки, відгуки та бажані товари, та адміністратори, які керують магазином. Кожен користувач має свій обліковий запис з унікальним ідентифікатором, електронною поштою, паролем тощо.

Відгуки на товар: коментарі що залишають клієнти про який-небудь товар. Кожен товар має свій ідентифікатор, що дає змогу клієнтам видаляти коментарі.

Адміністратори: відповідають за керування магазином, та можуть додавати, видаляти та редагувати товари.

Замовлення: в замовленні міститься така інформація, як дата та час, загальна сума, статус та інше. Вони пов'язані з користувачем, який здійснив замовлення.

Відносини між сутностями:

Користувачі можуть взаємодіяти з будь-яким продуктом і залишати відгуки на нього.

Продукти належать до певних категорій.

Замовлення пов'язані з конкретним користувачем та включають в себе продукти, які були замовлені.

Сценарії взаємодії користувачів з системою:

Реєстрація та вхід у систему: користувачі можуть легко створювати облікові записи та використовувати ці дані для входу до свого акаунту.

Пошук товарів: користувачі можуть шукати товари за ключовим словом, а саме за назвою товару.

Додавання товарів до кошику та списку бажань: користувачі можуть додавати товари до кошика, вибравши бажані характеристики, а також мають можливість додати до списку бажань.

Адміністрування магазину: адміністратори мають право додавати, видаляти та редагувати продукти.

Правила та обмеження:

Користувачі можуть переглянути характеристики товару на сторінці товару, та додавати його до кошика, лише після реєстрації та входу до свого облікового запису.

Адміністратори мають певні особливі права на доступ до магазину. Адміністратори виконують особливу роль, яка дозволяє їм використовувати меню адміністратора.

2.2 Архітектура програмного застосунку

Клієнтська частина (Frontend):

Інтерфейс користувача (UI): це засіб взаємодії між користувачем і комп'ютерною програмою або пристроєм. Це охоплює все, що користувач бачить на екрані – від тексту і зображень до кнопок та інших елементів керування [9]. В цьому застосунку це: сторінки каталогу, товарів, кошика, оформлення замовлення та інші. Застосунок інтернет-магазину являється *Single Page Application* (SPA), тобто оновлення сторінки ніколи не відбувається. Натомість увесь необхідний код HTML, JavaScript і CSS або витягується браузером під час одного завантаження сторінки, або відповідні ресурси динамічно завантажуються та додаються на сторінку в міру необхідності, зазвичай у відповідь на дії користувача [10]. Тобто замість того, щоб щоразу завантажувати нову сторінку з сервера, SPA

використовує техніку AJAX (асинхронні запити до сервера) для отримання даних і оновлення тільки необхідної її частини. Наприклад, перехід з головної сторінки у каталог призведе до того, що головна частина застосунку буде стерта та замінена на інший контент, а “шапка” та “підвал” залишаться не торкнутими, позаяк на обох сторінках вони виглядають однаково.

Логіка взаємодії з користувачем (Client-Side Logic) – це спосіб відображення всього контенту вебсайтів у браузері користувача. Браузер завантажує незначний HTML-код і джерела JavaScript, а потім керує відображенням вмісту сторінки [11].

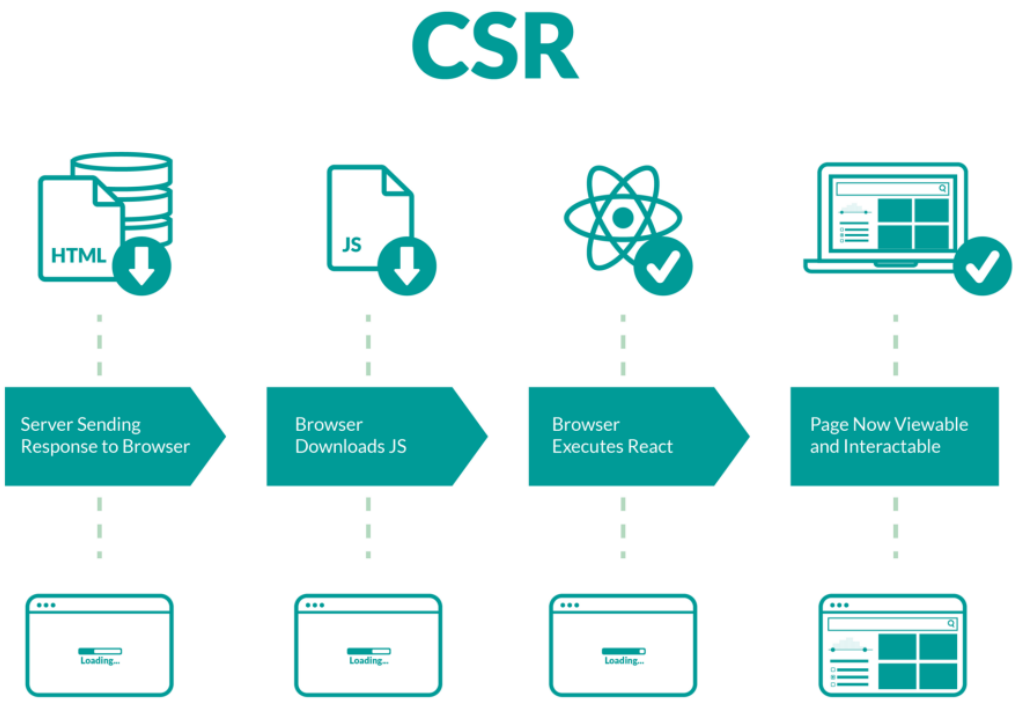

Рисунок 2.1 – Принцип роботи Client-Side Rendering

Керування станом (State Management) використовується для зберігання та оновлення даних на клієнтському боці, наприклад, за допомогою бібліотеки Redux або контексту React.

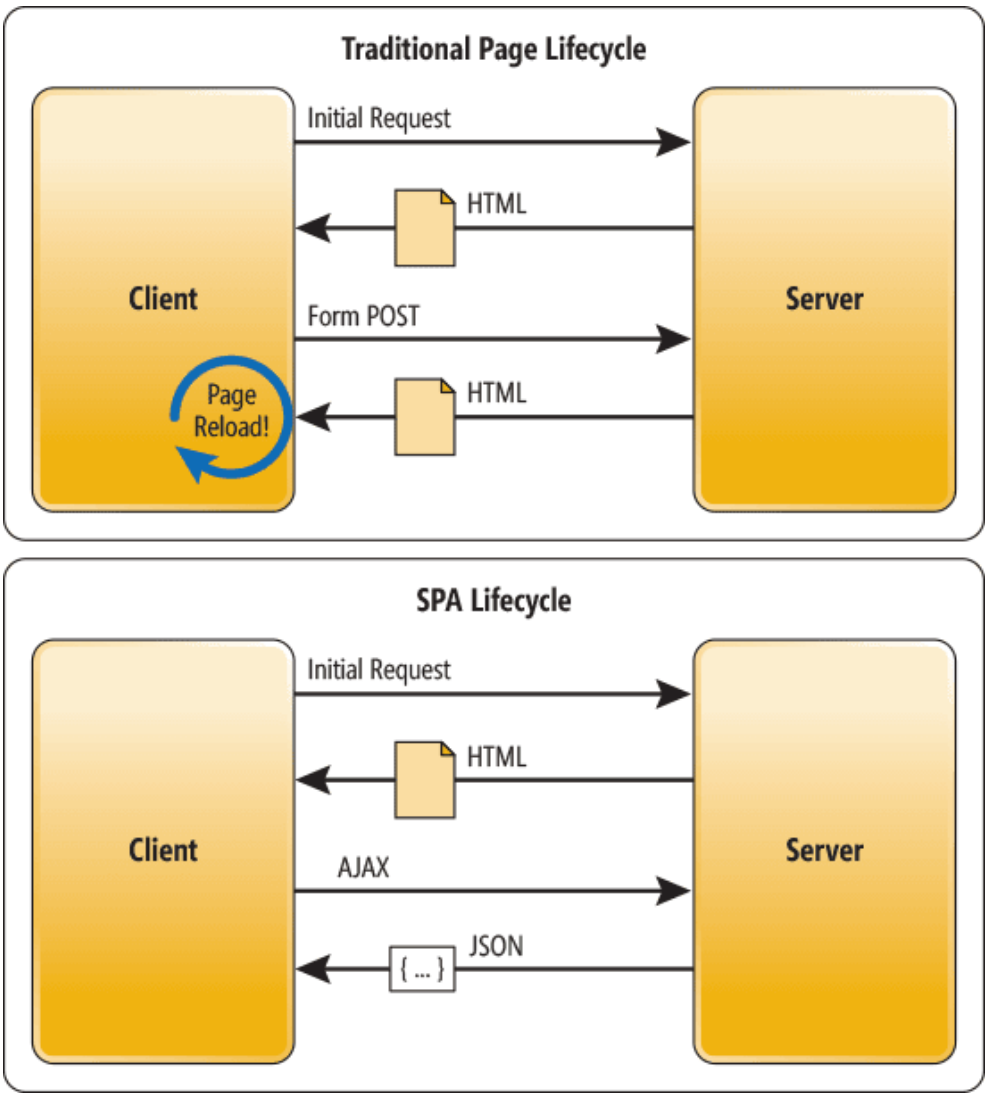

Рисунок 2.2 – Приклад життєвих циклів застосунку

Серверна частина (Backend):

API (Application Programming Interface): це механізми, які дають змогу двом програмним компонентам взаємодіяти один з одним, використовуючи набір визначень і протоколів [12]. В даному програмному продукті використовується REST API.

Rest Api застосунку написаний на мові програвання C# із фреймворком .Net. Це зручний інструмент для розробки вебзастосунків. Також було обрано такі Nuget пакети при розробці:

1. *AutoMapper* – для того щоб швидко та зручно зробити маппінг даних.

2. *BouncyCastle* – бібліотека яка містить в собі заготовлені методи для того щоб робити хешування даних.

1. *FluentValidation* – бібліотека допомагає вивести базу валідацію даних у окремий файл, а не робити її у контролері.

2. *Microsoft.AspNetCore.Authentication.Jwt* – бібліотека для роботи із JWT токіном.

3. *Microsoft.EntityFrameworkCore* – бібліотека для роботи із базою даних.

4. *Swashbuckle.AspNetCore* – бібліотека для підключення Swagger для того щоб протести end-points.

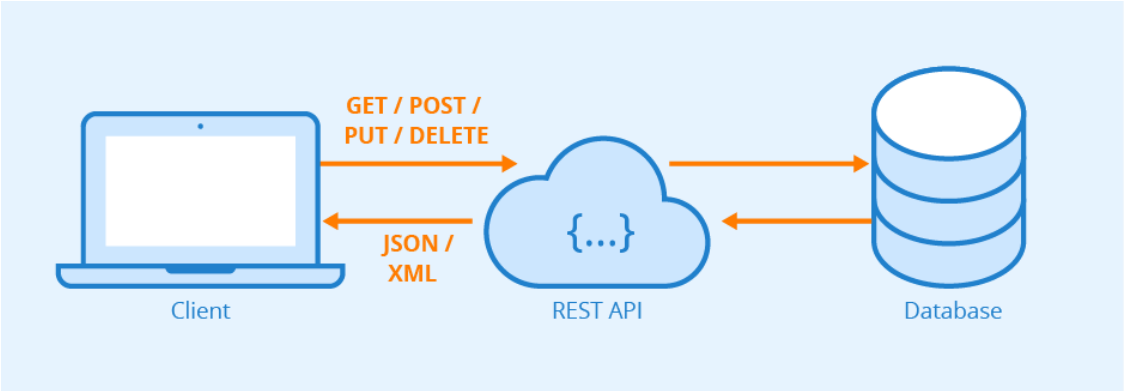

Рисунок 2.3 – Приклад роботи REST API

Бізнес-логіка (Business Logic): виконує обробку запитів від клієнта, валідацію даних, операції з базою даних та інші операції, пов'язані з бізнес-логікою застосунку. Для цього в застосунку використовується Redux бібліотека. Redux використовується для керування станом застосунку засобами створення Reducer, Thunk та ActionCreator.

*Reducer***:** це функції, які приймають поточний стан і дію як аргументи і повертають результат нового стану [13].

Thunk: функція, яка використовується в Redux для організації асинхронних операцій. Дозволяє викликати AC, які повертають функцію замість об'єкта дії. Ця функція отримує метод обробки магазину, який потім використовується для обробки регулярних синхронних дій всередині тіла функції після виконання асинхронних операцій [14]. Також вона дозволяє робити запити на сервер і обробляти результати цих запитів. Після написання логіки взаємодії з API через відповідний ендпоінт, thunk-функція організовує виконання цієї логіки.

ActionCreator: приймає дані з метою створення дій (actions), які будуть використані редуктором для зміни стану застосунка.

База даних (Database) зберігає інформацію про товари, користувачів, відгуки на товари та інші дані. Для зберігання даних в цьому проєкті використовується *SQL Server Management Studio 2019 (SSMS)*. Для роботи із самою базою даних було вибрано фреймворк *Entity framework*, який забезпечує зручну роботу із базою даних. Для роботи із базою даних було розроблено два файли *MarcetDbContext*.cs, в якому зберігаються конфігурація контексту бази даних, включаючи визначення таблиць та їх зв'язків, та файл *Repositor.cs*, в якому реалізовані дженерік-методи для стандартних дій із базою даних: додавання, видалення, пошук.

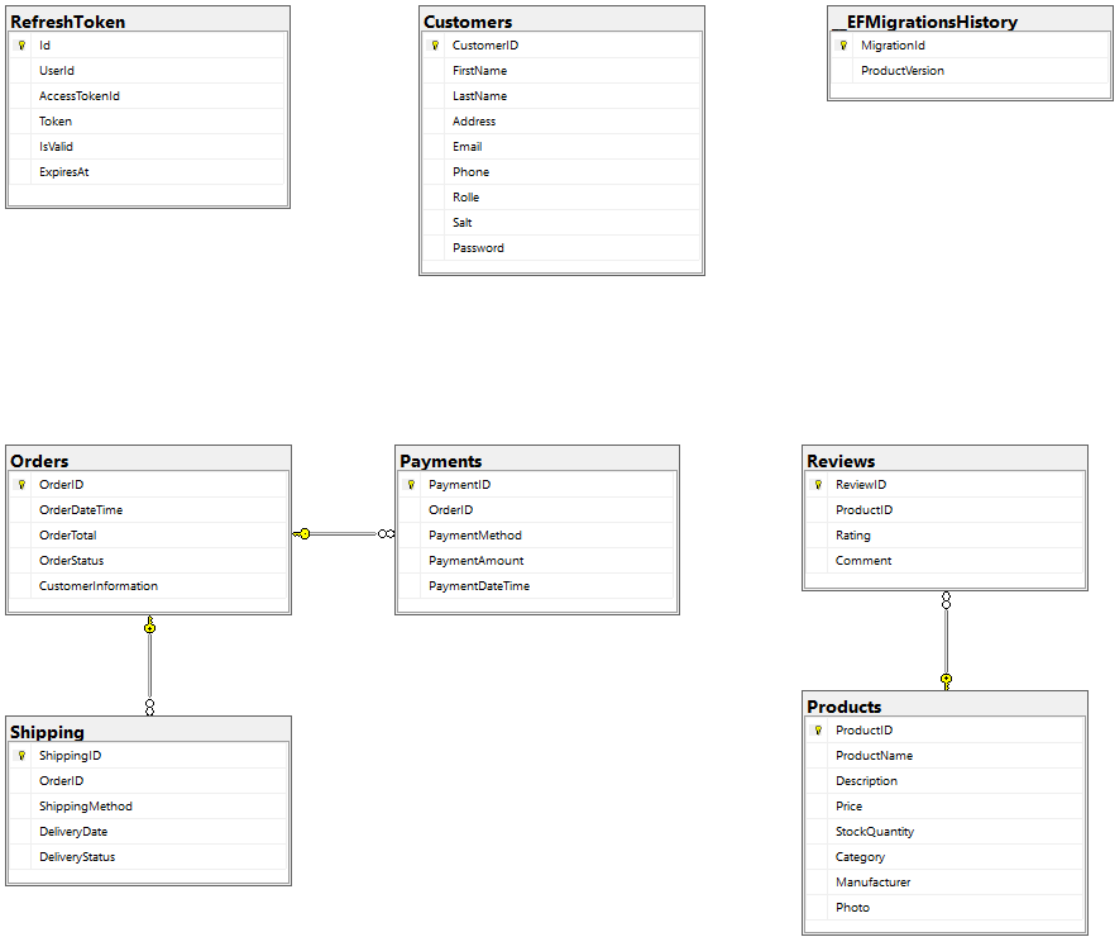

Рисунок 2.4 – Схема бази даних

На рис. 2.4 рефреш токени ні з чим не пов'язані з метою безпеки, міграції. Ця таблиця створена за допомогою Entity framework, вона технічна і не пов'язана ні з чим більше, користувач налаштований так, що жодні зв'язки йому не потрібні.

Розглянемо кожну таблицю:

RefreshToken: містить JWT Refresh Token, який виділяється для кожного користувача та зберігається протягом 7 днів. Поле "id" має тип даних "uniqueidentifier" у MSSQL та "Guid" у C#. Поле "UserId" також має тип даних "uniqueidentifier" та зберігає ідентифікатор користувача, якому був виданий JWT Token. У полі "AccessTokenId" зберігається ідентифікатор Access токена, який дійсний протягом однієї години. Поле "Token" містить сам токен, "IsValid" вказує на те, чи є токен дійсним (True або False), а у полі "ExpiresAt" зберігається дата закінчення терміну дії токена.

Customers: зберігає дані про користувачів. Поле "CustomerID" містить ідентифікатор користувача. Поля "FirstName" та "LastName" зберігають ім'я та прізвище користувача. Поле "Address" містить адресу проживання користувача. У полі "Email" зберігається електронна адреса користувача. Поле "Phone" містить мобільний номер користувача. "Role" визначає роль користувача (admin або user). У полі "Salt" зберігається сіль, яку додають до паролю, а "Password" містить хешований пароль, до якого спочатку додається сіль перед хешуванням.

Products: містить інформацію про товари, що продаються. Ідентифікація продукта міститься в полі «ProductID». Назву продукту містить поле «ProductName». Опис товару зберігається в розділі "Description". "Price" вказує на ціну товару, а "StockQuantity" вказує на кількість товару на складі. У полі "Category" вказується категорія товару, а у полі "Manufacturer" міститься назва виробника. В полі "Photo" можна знайти фотографії товару.

Reviews: містить відгуки про продукти. Ідентифікація відгуку міститься в полі «ReviewID». "ProductID" вказує на товар, до якого залишено відгук. Поля "Rating" та "Comment" містять оцінку та текст відгуку.

Payments: зберігає інформацію про оплату. Ідентифікації платежу та замовлення містяться в полях «PaymentID» і «OrderID». Інформація про спосіб

оплати міститься в розділі «PaymentMethod». Сума платежу та дата платежу містяться в розділах «PaymentAmount» і «PaymentDateTime».

Shipping: містить інформацію про доставку товарів. Ідентифікація доставки та замовлення містяться в полях "ShippingID" та "OrderID". "ShippingMethod" визначає спосіб доставки, а "DeliveryDate" та "DeliveryStatus" містять інформацію про дату доставки та її статус.

2.3 Проєктування бази даних (ER-модель)

Сутність "Товар" (Product):

1. *ProductID*: Унікальний ідентифікатор продукту;
2. *Назва (Name)*: Назва товару;
3. *Ціна (Price)*: Ціна товару;
4. *Виробник (Manufacturer)*: Компанія, що виробляє товар;
5. *Опис (Description)*: Опис товару для інформаційних цілей;
6. *Категорія (Category)*: Категорія, до якої належить товар;
7. *Фото (Photo):* Зображення товару;
8. *Кількість на складі (StockQuantity):* Кількість одиниць товару, які доступні на складі.

Сутність "Клієнт" (Customer):

1. *CustomerID*: Унікальний ідентифікатор клієнта;
2. *Ім'я (FirstName)*: Ім'я клієнта;
3. *Прізвище (LastName)*: Прізвище клієнта;
4. *Адреса (Address)*: Адреса клієнта;
5. *Електронна пошта (Email)*: Електронна адреса клієнта;
6. *Телефон (Phone)*: Номер телефону клієнта;
7. *Роль (Rolle)*: Роль авторизованого клієнта;
8. *Пароль (Password)*: Пароль клієнта для авторизації.

Сутність "Замовлення" (Order):

1. *OrderID*: Унікальний ідентифікатор замовлення;

2. *Дата та час*, коли було здійснено замовлення;

3. *Загальна вартість замовлення* (OrderTotal): Сума всіх товарів у замовленні;

4. *Статус замовлення (OrderStatus)*: Поточний статус замовлення (наприклад, "в обробці", "відправлено", "доставлено" тощо).

Сутність "Відгук" (Review):

1. *ReviewID*: Унікальний ідентифікатор відгуку;

2. *ProductID*: Ідентифікатор товару, до якого відноситься відгук;

3. *Оцінка (Rating)*: Оцінка товару від 1 до 5 або інша шкала;

4. *Коментар (Comment)*: Текстовий відгук клієнта про товар.

Сутність "Платіж" (Payment):

1. *PaymentID*: Унікальний ідентифікатор платежу;

2. *OrderID*: Ідентифікатор замовлення, за який здійснено платіж;

3. *Метод* (*PaymentMethod*), яким було здійснено оплату (наприклад, кредитна карта, PayPal тощо);

4. *Сума (PaymentAmount)*, сплачена клієнтом;

5. *Дата та час платежу (PaymentDateTime)*: Дата та час, коли було здійснено платіж.

Сутність "Доставка" (Shipping):

1. *ShippingID*: Унікальний ідентифікатор доставки;

2. *OrderID*: Ідентифікатор замовлення, для якого здійснюється доставка;

3. *Метод* (*ShippingMehod*), яким здійснюється доставка (наприклад, кур'єрська служба, поштова служба тощо);

4. Планована дата доставки (*ShippingData*);

5. Поточний статус доставки (*ShippingStatus*).

Взаємозв'язки:

Клієнт – замовлення (одно-до-багатьох): кожен клієнт може мати багато замовлень, але кожне замовлення належить тільки одному клієнту. Це означає, що

хоча в одного клієнта може бути кілька замовлень, кожне з них належить одному клієнту.

Замовлення – товар (багато-до-багатьох): кожне замовлення може містити багато товарів, і кожен товар може бути включений в багато замовлень. Це означає, що у замовлення може бути декілька товарів, і один товар може бути частиною різних замовлень.

*Товар – Відгук (одно-до-багато):*багато відгуків можуть бути про один товар, але кожен відгук стосується лише одного товару. Це означає, що, хоча багато відгуків на один продукт можуть бути надані, кожен відгук стосується лише одного продукту.

Замовлення – Платіж (одно-на-багато): багато платежів можуть бути включені до кожного замовлення, але кожен платіж стосується лише одного замовлення. Це означає, що ви можете сплачувати кілька замовлень одночасно, але кожен платіж стосується лише одного замовлення.

Замовлення – Доставка (одно-до-одного або Багато-до-багато): кожне замовлення може мати одну або багато доставок. Це означає, що доставка може бути організована для одного або кількох замовлень.

Діаграма сутностей-зв'язків для інтернет-магазину наведена на рис. 2.5.

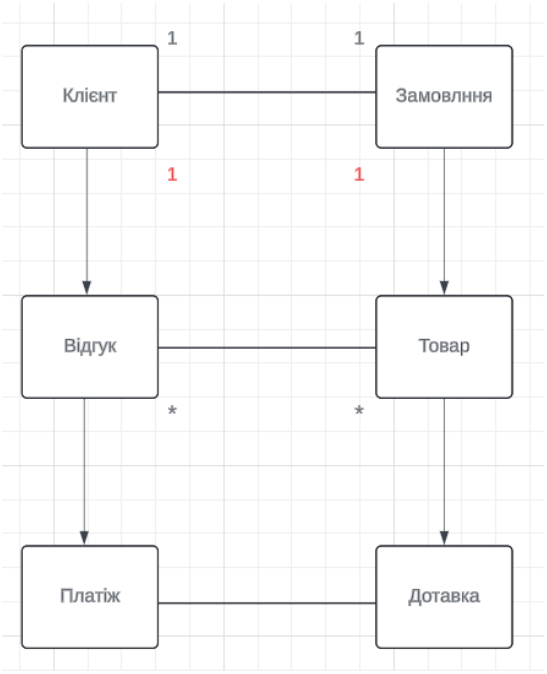

Рисунок 2.5 – Діаграма сутностей-зв'язків

2.4 Моделювання програмної системи

2.4.1 Use Cases діаграми

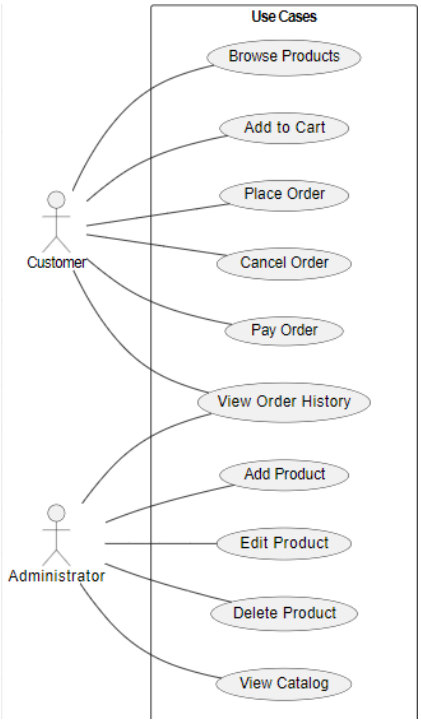

Рисунок 2.6 – Use Cases діаграма покупця та адміністратора

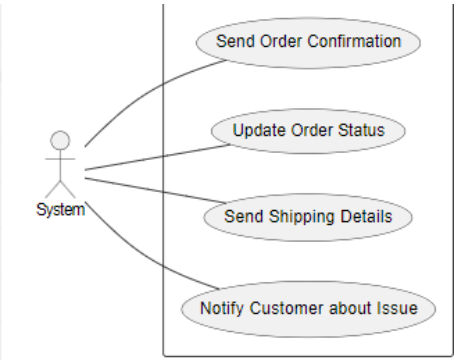

Рисунок 2.7 – Системна Use Cases діаграма

Ці діаграми моделюють взаємодію між різними аспектами онлайн-магазину, як-от:

1) Користувач (Customer):

Перегляд товарів (Browse Products): користувач може переглядати товари, що доступні в магазині.

Додавання в кошик (Add to Cart): користувач може додати вподобані товари *у кошик для подальшого оформлення замовлення.*

Розміщення замовлення (Place Order): після того, як користувач вибрав товари, він в змозі розмістити замовлення для подальшого придбання цих товарів.

Скасування замовлення (Cancel Order): користувач може скасувати раніше розміщене ним замовлення до того, як воно буде оброблене.

Оплата замовлення (Pay Order): користувач може оплатити замовлення після його розміщення, щоб завершити покупку.

Перегляд історії замовлень (View Order History): користувач може переглядати історію замовлення.

2) Адміністратор (Administrator):

Додавання товару (Add Product): адміністратор має право додавати нові товари до каталогу магазину.

Редагування товару (Edit Product): адміністратор має право змінювати властивості вже існуючих товарів.

Видалення товару (Delete Product): адміністратор має право видаляти товари з каталогу, які більше не продаються або не актуальні.

Перегляд каталогу (View Catalog): адміністратор, як і звичайний клієнт, може переглядати всі товари в каталозі магазину.

Перегляд історії замовлень (View Order History): адміністратор також може переглядати історію замовлень для відстеження стану та обробки замовлень.

3) Система (System):

Надсилання підтвердження замовлення (Send Order Confirmation): після успішного розміщення замовлення система автоматично надсилає клієнту підтвердження замовлення.

Оновлення статусу замовлення (Update Order Status): система автоматично оновлює статуси замовлень відповідно до їхнього поточного стану (наприклад, чекає на оплату, у процесі обробки, відправлено тощо).

Надсилання деталей доставки (Send Shipping Details): після відправлення товарів система надсилає клієнту деталі доставки та інструкції щодо відстеження замовлення.

Сповіщення клієнта про проблеми (Notify Customer about Issues): система автоматично сповіщає клієнта про проблеми, пов’язані з обробкою замовлення.

2.4.2 Діаграма класів

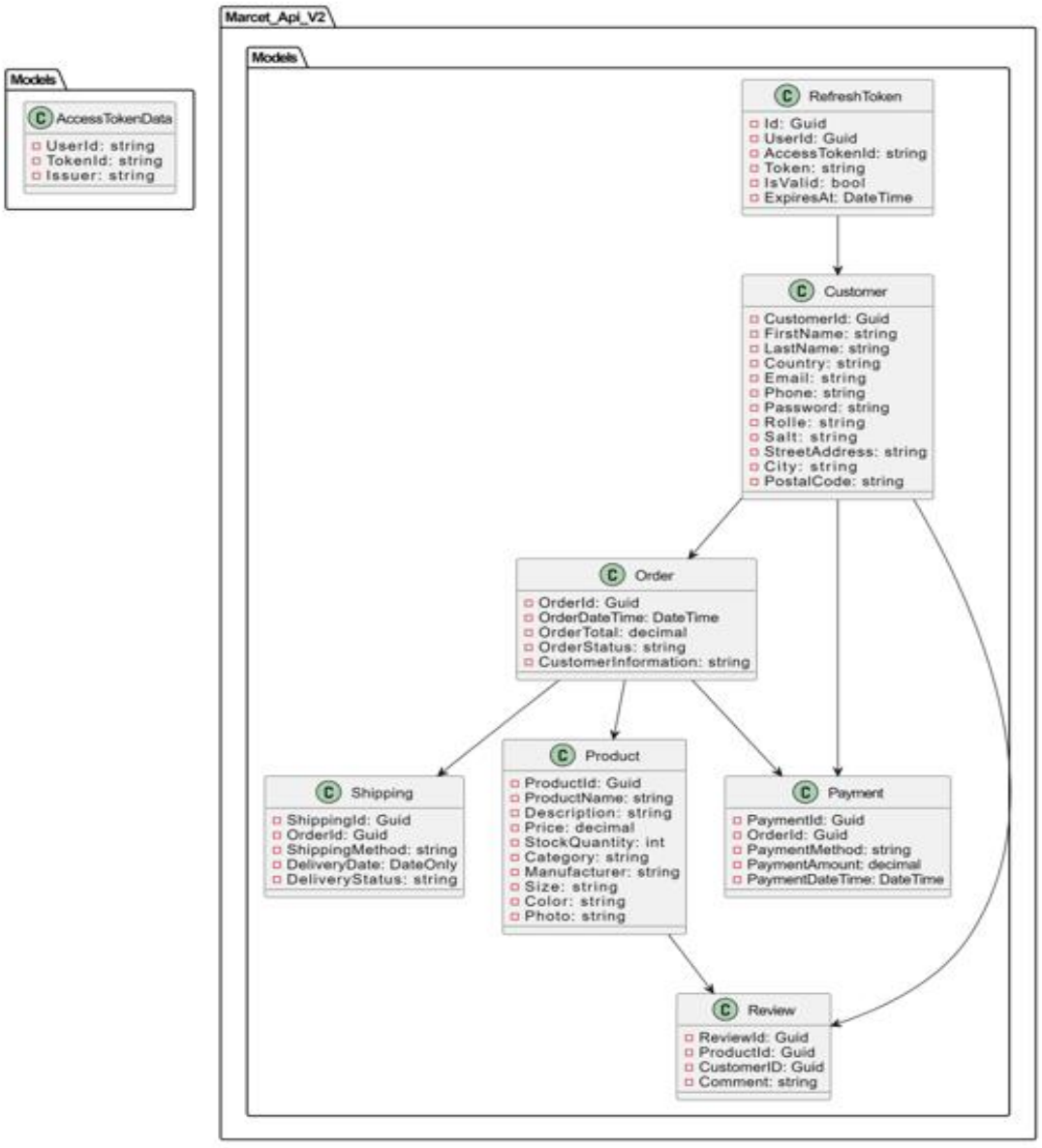

Рисунок 2.8 – Діаграма класів

Подана діаграма класів моделей описує структуру об'єктів та їхні зв'язки в системі керування магазином:

1) *AccessTokenData*: подає дані токену доступу, який може використовуватися для автентифікації користувача. Він містить поля UserId (токен користувача), TokenId (ідентифікатор токену) і Issuer (видавець токену).

2) *Customer:* подає дані про клієнта магазину. Містить особисті дані клієнта: ім'я, прізвище, адреса, контактні дані, роль тощо.

3) *Order:* формує замовлення в системі. Містить інформацію про дату та час замовлення, загальну суму, статус, інформацію про клієнта та оплату.

4) *Payment:* подає інформацію про оплату замовлення. Містить дані про спосіб оплати, суму оплати, дату та час оплати.

5) *Product:* описує товари, доступні в магазині. Включає атрибути: назва, опис, ціна, категорія, виробник, розмір, колір тощо.

6) *RefreshToken:* подає інформацію про токени оновлення, які можуть використовуватися для продовження терміну дії токенів доступу. Містить дані про користувача, ідентифікатор токену, термін дії тощо.

7) *Review* – клас для відгуків користувачів про товари. Містить коментарі, оцінки, інформацію про користувача та товар.

8) *Shipping:* подає інформацію про доставку замовлення. Містить дані про метод доставки, дату доставки, статус доставки тощо.

2.4.3 Діаграма активностей

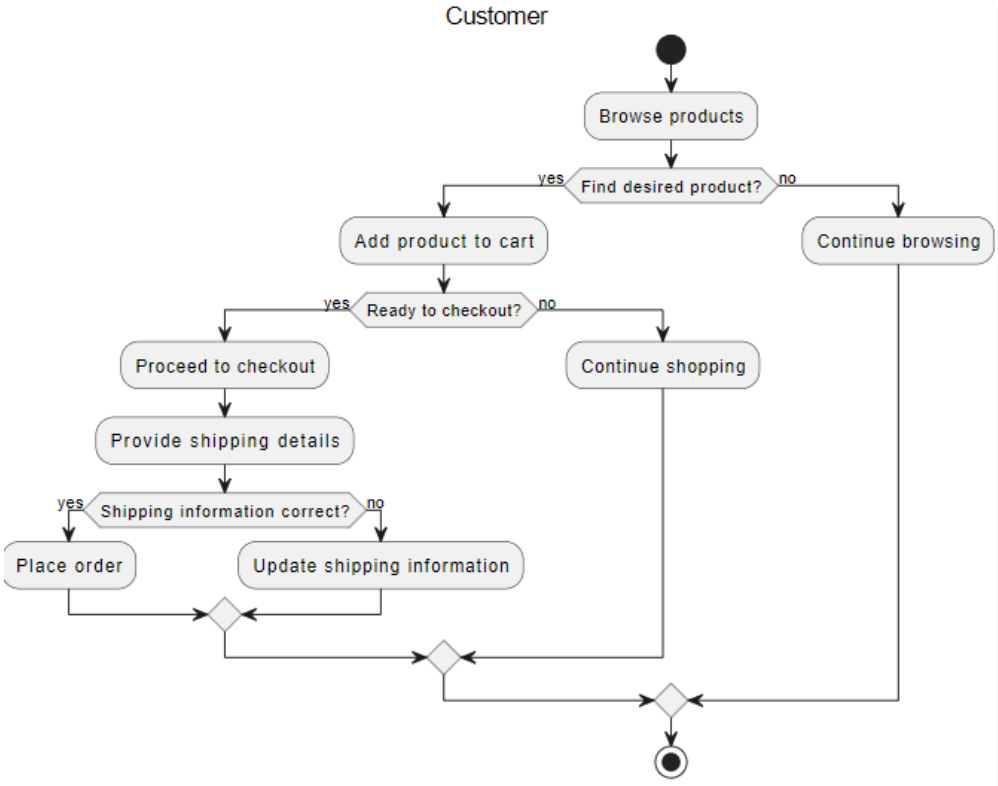

Рисунок 2.9 – Activity діаграма користувача

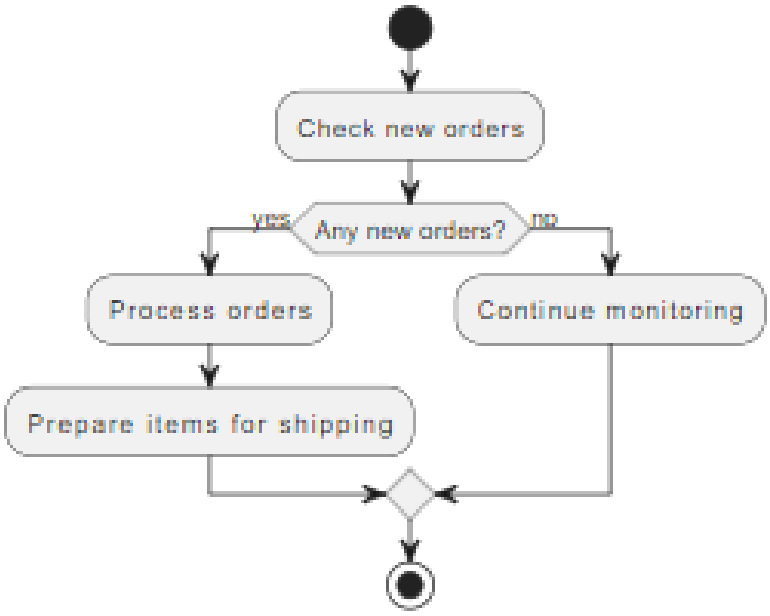

Рисунок 2.10 – Activity діаграма адміністратора

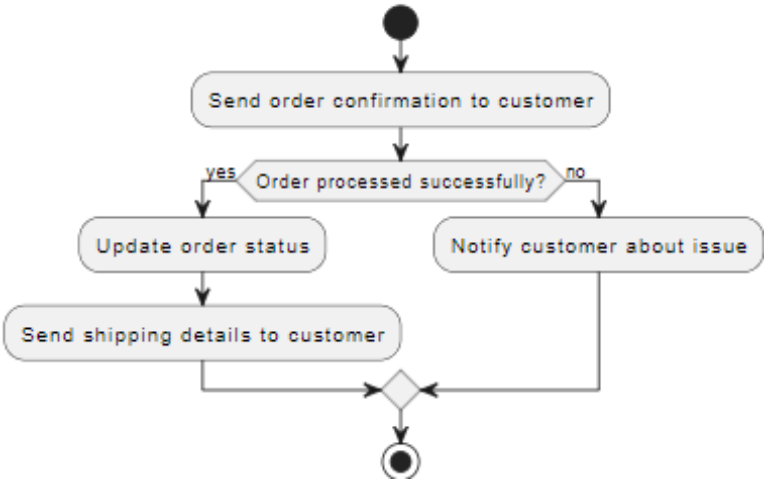

Рисунок 2.11 – Системна activity діаграма

Представлені діаграми активностей описують можливі сценарії дій акторів.

Детальний опис дій кожного з акторів:

Дії користувача (*Customer*) (рис. 2.9):

Починається з початкового стану *start*.

Користувач переглядає продукти, позначені лінією *Browse products*.

Він перевіряє, чи знайшов він бажаний товар, позначений умовним блоком *Find desired product*? Якщо так, він додає товар у кошик та переходить до стану *Add product to cart*. Потім користувач перевіряє, чи готовий він здійснити покупку, умовний блок *Ready to checkout*? Якщо так, він переходить до стану Proceed to checkout, де надає інформацію для доставки. Система перевіряє коректність наданої інформації про доставку, визначено умовним блоком *Shipping information correct*? Якщо так, замовлення розміщується та переходить до стану *Place order*. У випадку некоректної інформації про доставку, користувач може оновити інформацію, *позначену Update shipping information*. Після завершення замовлення користувач закінчує сесію, позначену *stop*.

Дії адміністратора (Administrator) (рис 2.10):

Адміністратор починає з перевірки наявних нових замовлень. Якщо є нові замовлення, він обробляє їх та підготовлює товари для відправки. У випадку відсутності нових замовлень адміністратор продовжує відстежувати стан системи.

Дії системи (рис. 2.11):

Система надсилає підтвердження замовлення користувачеві. Вона перевіряє, чи було замовлення оброблено успішно. Якщо так, вона оновлює статус замовлення та надсилає користувачеві інформацію про доставку. У випадку невдалого оброблення замовлення система сповіщає користувача про виниклу проблему.

2.5 Висновки до розділу

Мета інформаційної моделі інтернет-магазину полягала в тому, щоб дослідити сутності та те, як вони взаємодіють один з одним. Модель пояснює, як працюють різні компоненти системи, що полегшує розуміння функціональних можливостей і вимог користувачів.

Під час розробки архітектури програмного застосунку була розроблена детальна структура, яка охоплює частини та взаємодію кожної частини системи. Це гарантує, що різні модулі програми добре працюють один з одним.

База даних була розроблена так, щоб враховувати вимоги системи, а також методи зберігання та обробки даних. Модель бази даних демонструє структуру та взаємозв’язки між таблицями, що полегшує зберігання та керування різними частинами системи.

Моделювання програмної системи проводилося за допомогою діаграм UML, таких як Use Cases, Classes та Activity. Такі діаграми допомогли краще зрозуміти функціональні можливості системи.

3 РОЗРОБКА ТА ТЕСТУВАННЯ ІНТЕРНЕТ-МАГАЗИНУ

3.1 Розробка запитів до бази даних відповідно до функціоналу застосунку

1) Створення таблиць бази даних

У наведеному коді (рис. 3.1) визначається міграція для створення бази даних.

```
namespace Marcet_Api_V2.Migrations
{
    /// <inheritdoc />
    Ссылок: 1
    public partial class Initial : Migration
    {
        /// <inheritdoc />
        Ссылок: 0
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.CreateTable(
                name: "Customers",
                columns: table => new
                {
                    CustomerID = table.Column<Guid>(type: "uniqueidentifier", nullable: false),
                    FirstName = table.Column<string>(type: "varchar(255)", unicode: false, maxLength: 255, nullable: true),
                    LastName = table.Column<string>(type: "varchar(255)", unicode: false, maxLength: 255, nullable: true),
                    Address = table.Column<string>(type: "varchar(255)", unicode: false, maxLength: 255, nullable: true),
                    Email = table.Column<string>(type: "varchar(255)", unicode: false, maxLength: 255, nullable: true),
                    Phone = table.Column<string>(type: "varchar(20)", unicode: false, maxLength: 20, nullable: true),
                    Password = table.Column<string>(type: "varchar(20)", unicode: false, maxLength: 20, nullable: true),
                    Rolle = table.Column<string>(type: "varchar(10)", unicode: false, maxLength: 10, nullable: true),
                    Salt = table.Column<string>(type: "nvarchar(max)", nullable: true)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK__Customer__A4AE64B85F9253AB", x => x.CustomerID);
                });
```

Рисунок 3.1 – Код міграції бази даних

EF Core Migrations – це функція EF Core, яка дає змогу із часом розвивати схему бази даних з урахуванням версій у міру розвитку програми.

Завдяки міграції можна легко застосовувати або скасовувати зміни бази даних по мірі розвитку застосунку, не втрачаючи при цьому наявні дані [15].

У методі *Up* описується, як створити таблиці бази даних і встановити зв'язки між ними. Для кожної таблиці визначаються колонки з їх типами даних та обмеженнями. Також описується первинний ключ та зовнішні ключі для зв'язків між таблицями.

Наприклад, для таблиці "*Customers*" описані наступні колонки:

CustomerID (Guid) – унікальний ідентифікатор клієнта;

FirstName (varchar(255)) – ім'я клієнта;

LastName (varchar(255)) – прізвище клієнта;

Address (varchar(255)) – адреса клієнта;

Email (varchar(255)) – електронна пошта клієнта;

Phone (varchar(20)) – телефонний номер клієнта;

Password (varchar(20)) – пароль клієнта;

Rolle (varchar(10)) – роль користувача;

Salt (nvarchar(max)) – сіль для пароля.

Далі для кожної таблиці описані відповідні обмеження первинного ключа та зовнішніх ключів для зв'язків з іншими таблицями. Наприклад, таблиця "*Payments*" має зовнішній ключ "OrderID", який посилається на поле "OrderID" таблиці "Orders".

Код на рис. 3.2 визначатиме структуру бази даних та зв'язки між таблицями, які будуть створені під час міграції.

```
protected override void Down(MigrationBuilder migrationBuilder)
{
    migrationBuilder.DropTable(
        name: "Customers");

    migrationBuilder.DropTable(
        name: "Payments");

    migrationBuilder.DropTable(
        name: "RefreshToken");

    migrationBuilder.DropTable(
        name: "Reviews");

    migrationBuilder.DropTable(
        name: "Shipping");

    migrationBuilder.DropTable(
        name: "Products");

    migrationBuilder.DropTable(
        name: "Orders");
}
```

Рисунок 3.2 – Код видалення структури бази даних

У методі Down (*рис. 3.2*) описується, як відкатати міграцію, тобто як видалити структуру бази даних. В цьому випадку всі таблиці видаляються.

2) Структура таблиць бази даних:

Для наведення прикладу структури знов наведу таблицю *Customer*

```
modelBuilder.Entity("Marcet_Api_V2.Models.Customer", b =>
    {
        b.Property<Guid>("CustomerId")
            .HasColumnType("uniqueidentifier")
            .HasColumnName("CustomerID");

        b.Property<string>("Address")
            .HasMaxLength(255)
            .IsUnicode(false)
            .HasColumnType("varchar(255)");

        b.Property<string>("Email")
            .HasMaxLength(255)
            .IsUnicode(false)
            .HasColumnType("varchar(255)");

        b.Property<string>("FirstName")
            .HasMaxLength(255)
            .IsUnicode(false)
            .HasColumnType("varchar(255)");

        b.Property<string>("LastName")
            .HasMaxLength(255)
            .IsUnicode(false)
            .HasColumnType("varchar(255)");

        b.Property<string>("Password")
            .HasMaxLength(20)
            .IsUnicode(false)
            .HasColumnType("varchar(20)");

        b.Property<string>("Phone")
            .HasMaxLength(20)
            .IsUnicode(false)
            .HasColumnType("varchar(20)");

        b.Property<string>("Rolle")
            .HasMaxLength(10)
            .IsUnicode(false)
            .HasColumnType("varchar(10)");

        b.Property<string>("Salt")
            .HasColumnType("nvarchar(max)");

        b.HasKey("CustomerId")
            .HasName("PK__Customer__A4AE64B85F4253AB");

        b.ToTable("Customers");
    });
```

Рисунок 3.3 – Код властивостей

В коді, наведеному на рис. 3.3, описано властивості (колонки) з їх типами даних та обмеженнями. Для кожної властивості вказано тип даних (*HasColumnType)*, максимальну довжину (*HasMaxLength*), чи використовується *Unicode* (*IsUnicode*), а також назву колонки у базі даних (*HasColumnName*). Також описано первинний ключ (*HasKey*) та назву обмеження (*HasName*).

Далі для кожної таблиці описано зовнішні ключі (якщо такі є) та їхні обмеження (*HasForeignKey*, *HasConstraintName*).

3) Пояснення MarcetDbContext:

Код, представлений у застосунку, має клас *MarcetDbContext*, який є похідним від *DbContext* у контексті *EntityFrameworkCore*. Цей клас відповідає за взаємодію з базою даних, забезпечуючи доступ до різних сутностей та їх взаємозв'язків.

4) Основні дії, які виконує MarcetDbContext:

Визначає конструктори для класу MarcetDbContext, один з яких приймає параметр DbContextOptions<MarcetDbContext>, що дозволяє налаштовувати підключення до бази даних.

Оголошує властивості DbSet для кожної сутності (таблиці) в базі даних, такі як Customers, Orders, Payments і т.д. Ці властивості представляють собою колекції об'єктів, які можуть бути отримані, додані, видалені або змінені в контексті бази даних.

Визначає метод OnConfiguring, який встановлює параметри підключення до бази даних. У цьому випадку використовується підключення до SQL Server, яке міститься прямо у коді (що може бути потенційним загрозою безпеці), але рекомендується використовувати безпечніші методи зберігання підключень, такі як конфігураційні файли.

Визначає метод OnModelCreating, який налаштовує модель бази даних за допомогою Fluent API. Цей метод встановлює ключі, типи даних та відносини між сутностями.

5) Пояснення що таке dbSet:

Клас DbSet<TEntity> є частиною Entity Framework Core і подає набір об'єктів (entities) даного типу в контексті бази даних. Він дозволяє звертатися до колекції об'єктів, що зберігаються в базі даних, та виконувати з ними різноманітні операції, такі як додавання, оновлення, видалення тощо;

Основна функція класу DbSet<TEntity> полягає в тому, щоб він реалізовував можливість взаємодії з об'єктами бази даних через об'єкти типу TEntity, який подає сутність (entity) в контексті бази даних. Він надає різноманітні

методи та властивості, які дозволяють отримувати, додавати, оновлювати та видаляти дані з бази даних;

Клас DbSet<TEntity> є однією з основних складових частин Entity Framework Core та дозволяє легко взаємодіяти з об'єктами бази даних в об'єктно-орієнтованому стилі, що спрощує розробку програм, що використовують дані з бази даних.

6) Пояснення класу Repository:

У наведеному нижче коді (рис. 3.4) було створено загальний репозиторій Repository<T>, який реалізує інтерфейс IRepository<T>. Цей репозиторій призначений для взаємодії з базою даних для будь-якої сутності T.

```
public class Repository<T> : IRepository<T> where T : class
{
    private readonly MarcetDbContext _context;
```

Рисунок 3.4 – Створення загального репозиторія

Основні функції, які виконує цей репозиторій:

1. *Конструктор*: При створенні об'єкта репозиторію, передається об'єкт контексту бази даних MarcetDbContext. Також встановлюється зв'язок з набором даних _dbSet, який подає собою таблицю бази даних, що відповідає сутності T;

```
private readonly DbSet<T> _dbSet;
Ссылок: 0
public Repository(MarcetDbContext context, ILogger<Repository<T>> logger)
{
    _context = context;
    _dbSet = _context.Set<T>();
}
```

Рисунок 3.5 – Конструктор

2. *Метод CreateAsync*: додає новий об'єкт сутності до таблиці бази даних та зберігає зміни асинхронно;

```
public async Task CreateAsync(T entity)
{
    try
    {
        await _dbSet.AddAsync(entity);
        await SaveAsync();
    }
    catch (Exception ex)
    {
        throw;
    }
}
```

Рисунок 3.6 – 2. Метод CreateAsync

3. *Метод RemoveAsync*: видаляє об'єкт сутності з таблиці бази даних та зберігає зміни асинхронно;

```
public async Task RemoveAsync(T entity)
{
    try
    {
        _dbSet.Remove(entity);
        await SaveAsync();
    }
    catch (Exception ex)
    {
        throw;
    }
}
```

Рисунок 3.7 – Метод RemoveAsync

4. *Метод GetAsync*: отримує один об'єкт сутності з таблиці бази даних відповідно до вказаного фільтра або за умови, що вказана;

```
public async Task<T> GetAsync(Expression<Func<T, bool>> filter = null, bool tracked = true)
{
    try
    {
        IQueryable<T> query = _dbSet;

        if (!tracked)
        {
            query = query.AsNoTracking();
        }

        if (filter != null)
        {
            query = query.Where(filter);
        }

        return await query.FirstOrDefaultAsync();
    }
    catch (Exception ex)
    {
        throw;
    }
}
```

Рисунок 3.8 – 4. Метод GetAsync

5. *Метод GetAllAsync*: отримує всі об'єкти сутності з таблиці бази даних відповідно до вказаного фільтра або усі об'єкти, якщо фільтр не вказано;

```
public async Task<List<T>> GetAllAsync(Expression<Func<T, bool>>? filter = null)
{
    try
    {
        IQueryable<T> query = _dbSet;

        if (filter != null)
        {
            query = query.Where(filter);
        }

        return await query.ToListAsync();
    }
    catch (Exception ex)
    {
        throw;
    }
}
```

Рисунок 3.9 – Метод GetAllAsync

6. *Метод UpdateAsync*: оновлює об'єкт сутності в таблиці бази даних та зберігає зміни асинхронно;

```
public async Task UpdateAsync(T entity)
{
    try
    {
        _dbSet.Update(entity);
        await SaveAsync();
    }
    catch (Exception ex)
    {
        throw;
    }
}
```

Рисунок 3.10 – Метод UpdateAsync

7. *Метод DeleteAsync*: видаляє об'єкти сутності з таблиці бази даних, що задовольняють вказаний фільтр, та зберігає зміни асинхронно;

```
public async Task DeleteAsync(Expression<Func<T, bool>> filter)
{
    try
    {
        var entitiesToDelete = _dbSet.Where(filter);
        _dbSet.RemoveRange(entitiesToDelete);
        await SaveAsync();
    }
    catch (Exception ex)
    {
        throw;
    }
}
```

Рисунок 3.11 – Метод DeleteAsync

8. *Метод SaveAsync*: зберігає всі зміни в контексті бази даних асинхронно.

```
public async Task SaveAsync()
{
    try
    {
        await _context.SaveChangesAsync();
    }
    catch (Exception ex)
    {
        throw;
    }
}
```

Рисунок 3.12 – Метод SaveAsync

Додаткові методи для отримання даних: в репозиторії є декілька додаткових методів для отримання даних, таких як отримання продуктів за розміром чи категорією, а також отримання відгуків за ідентифікатором.

Цей репозиторій реалізує різноманітні функції для роботи з базою даних і може бути використаний для взаємодії з будь-якою сутністю в контексті Entity Framework Core.

Клас Repository<T> універсальний для різних сутностей. Він є шаблонним класом, де T є типом сутності, з якою буде працювати репозиторій. Це дозволяє використовувати один і той же клас репозиторію для будь-якої сутності в контексті Entity Framework Core.

Ключовою перевагою в використанні універсального репозиторію є те, що можно використовувати один і той же код для створення, зчитування, оновлення та видалення об'єктів будь-якої сутності. Це спрощує роботу з базою даних, оскільки не потрібно створювати окремий репозиторій для кожної окремої сутності.

```
public virtual DbSet<TEntity> Set<[DynamicallyAccessedMembers(IEntityType.DynamicallyAccessedMemberTypes)] TEntity>()
    where TEntity : class
    => (DbSet<TEntity>)((IDbSetCache)this).GetOrAddSet(DbContextDependencies.SetSource, typeof(TEntity));
```

Рисунок 3.13 – метод Set<TEntity>

У представленому коді (*рис. 3.13*), визначено метод Set<TEntity>(), який є частиною класу контексту бази даних (наприклад, класу, який успадковує

DbContext у Entity Framework Core). Цей метод повертає об'єкт DbSet<TEntity>, який подає набір сутностей заданого типу TEntity.

Параметр TEntity в цьому методі вказує на тип сутності, з якою буде працювати набір. Вираз where TEntity : class обмежує тип TEntity класами (тобто не можна використовувати цей метод з іншими типами, такими як структури або делегати).

3.2 Розробка серверної частини з використанням .net

Розглянемо сервіс *CustomerService*, який відповідає за взаємодію з моделлю користувача (Customer):

Конструктор (рис 3.14): у конструкторі відбувається ініціалізація репозиторію користувачів (IRepository<Customer>) та контексту бази даних (MarcetDbContext).

```
Ссылок: 0
public CustomerService(IRepository<Customer> dbCustomer, MarcetDbContext dbContext)
{
    _customerRepository = dbCustomer;
    _dbContext = dbContext;
}
```

Рисунок 3.14 – Ініціалізація репозиторію користувачів

Репозиторій користувачів передається через залежність (dbCustomer).

Метод *GetCustomerByIdAsync* (рис 3.15): даний метод приймає ідентифікатор користувача та повертає відповідного користувача з бази даних за його ідентифікатором.

```
public async Task<Customer> GetCustomerByIdAsync(Guid id)
{
    try
    {
        return await _customerRepository.GetAsync(u => u.CustomerId == id);
    }
    catch (Exception ex)
    {
        throw;
    }
}
```

Рисунок 3.15 – Метод GetCustomerByIdAsync

Використовується асинхронний запит до репозиторію, щоб отримати користувача за його ідентифікатором.

Метод *EditCustomerInfo* (рис 3.16): цей метод використовується для редагування інформації про користувача. Він знаходить існуючого користувача за його ідентифікатором, оновлює його дані за допомогою даних з об'єкта CustomerDTO, які були передані в метод, та зберігає зміни у базі даних.

```
public async Task EditCustomerInfo(Guid id, CustomerDTO customerDTO)
{
    try
    {
        var existingCustomer = await _dbContext.Customers.FindAsync(id);
        if (existingCustomer != null)
        {
            existingCustomer.FirstName = customerDTO.FirstName;
            existingCustomer.LastName = customerDTO.LastName;
            existingCustomer.StreetAddress = customerDTO.StreetAddress;
            existingCustomer.Email = customerDTO.Email;
            existingCustomer.Phone = customerDTO.Phone;
            existingCustomer.Country = customerDTO.Country;
            existingCustomer.City = customerDTO.City;
            existingCustomer.PostalCode = customerDTO.PostalCode;

            await _dbContext.SaveChangesAsync();
        }
    }
    catch (ArgumentException ex)
    {
        throw;
    }
}
```

Рисунок 3.16 – Метод EditCustomerInfo

Метод *DeleteCustomerAsync* (рис 3.17): цей метод видаляє користувача з бази даних за його ідентифікатором. Використовується асинхронний запит до репозиторію для видалення користувача.

```
public async Task DeleteCustomerAsync(Guid id)
{
    try
    {
        await _customerRepository.DeleteAsync(u => u.CustomerId == id);
    }
    catch (Exception ex)
    {
        throw;
    }
}
```

Рисунок 3.17 – Метод DeleteCustomerAsync

Метод *CreateAsync* (рис 3.18): цей метод використовується для створення нового користувача у базі даних. Метод використовує асинхронні запити до репозиторію.

```
public async Task CreateAsync(Customer customer)
{
    try
    {
        await _customerRepository.CreateAsync(customer);
    }
    catch (Exception ex)
    {
        throw;
    }
}
}
```

Рисунок 3.18 – Метод CreateAsync

3.3 Розробка клієнтської частини засобами React

3.3.1 Робота з API

Для роботи з API використовується бібліотека Axios.

Axios – це бібліотека, яка використовується для надсилання запитів до API, повернення даних з API, а потім виконання дій з цими даними в нашому застосунку React [16].

Перед надсиланням запитів на сервер, створюється “*instance*”.

Instance – це екземпляр *Axios*, що містить базові налаштування(ключі-значення), які будуть використовуватися для всіх HTTP запитів.

```
const instance = axios.create(
    {
        withCredentials:true,
        baseURL: `https://localhost:7189/api/`,
        headers: {
          'Content-Type': 'application/json',
          Authorization: `Bearer ${accessToken}`,
        }
    }
);
```

Рисунок 3.19 – Базові налаштування

В коді на рис. 3.19 описано декілька базових налаштувань, а саме:

1) *withCredentials* зі значенням true: даний запис забезпечує запис токенів авторизації та оновлення до cash'у сторінки;

2) *baseURL*: містить стартовий URL всіх запитів до якої будуть подалі додаватися відносні шляхи у запитах;

3) Об'єкт *headers* містить два ключи, *Content-Type* та *Authorization*.

Content-Type із значенням application/json говорить, що формат даних, які відправляються в тілі запиту, є JSON. Це допомагає серверу зрозуміти, як обробляти тіло запиту.

Authorization із значенням Bearer ${accessToken} використовується для передачі токену аутентифікації.

```
instance.interceptors.request.use(
    (config) => {
        const currentToken = localStorage.getItem('accessToken');
        if (currentToken) {
            config.headers.Authorization = `Bearer ${currentToken}`;
        }
        return config;
    },
    (error) => {
        return Promise.reject(error);
    }
);
```

Рисунок 3.20 – Інтерцептор Axios

У наведеному коді (рис. 3.20) використовується інтерцептор (interceptor) Axios для обробки кожного запиту перед його відправкою.

Інтерцептори в Axios дозволяють перехоплювати запити або відповіді перед тим, як вони будуть оброблені. В даному випадку, ми перехоплюємо кожен запит перед його відправою щоб автоматично додати поточний токен доступу до кожного запиту. Крім того, коли токен оновлюється (наприклад, після логіну), інтерцептор автоматично використовуватиме новий токен із localStorage. Тобто замість того, щоб вручну додавати токен аутентифікації до кожного запиту, Axios Interceptor може автоматизувати цей процес [17].

Розглянемо приклад запиту програми до сервера (рис. 3.21).

```
export const productsAPI = {
  getProducts(category){
    return instance.get(`Products/getByCategory/${category}`);
  },

  getProductById(productId){
      return instance.get(`Products/${productId}`)
    },
  getProductByName(productName){
      return instance.post(`Products/search?productName=${productName}`  )
    },

}
```

Рисунок 3.21 – Метод products API

На рис. 3.21 наведено три запити, згруповані в одному єдиному методі productsAPI:

1) getProducts – це endpoint з методом get, який відповідає за отримання товарів з бази даних. Він приймає обов'язковий параметр category, для того, щоб дістати товари з певною категорією

2) getProductById – це endpoint з методом get, який відповідає за отримання певного товару по його Id. Відповідно приймає обов'язковий параметр productId;

3) getProductByName – це endpoint з методом post, який відповідає за пошук товару по ключовому слову, а саме його назві. На відміну від попередніх запитів в своєму URL він має параметр запиту, що починається з познаки "?", в який треба передати назву товару для пошуку.

Взагалі існують декілька методів для роботи з API [18]:

1. Метод GET використовується для отримання даних з сервера;

2. Метод POST використовується для відправи даних на сервер;

3. Метод DELETE використовується для видалення чогось з серверу:

4. Метод PUT використовується для оновлення чогось що вже є на сервері.

3.3.2 Робота з бібліотекою Redux

Redux – це JS-бібліотека для передбачуваного та зручного глобального керування станом (state manegment). Використання цієї бібліотеки надають можливості для редагування коду в реальному часі [19].

```
import {applyMiddleware, combineReducers, compose, legacy_createStore as createStore} from "redux";
import catalogReducer from './catalogReducer'
import authReducer from "./auth-reducer";
import profileReducer from "./profileReducer";
import productsReducer from "./productsReducer";
import cartReducer from "./cartReducer";
import favoriteReducer from "./favoriteReducer";
import  thunkMiddleware from "redux-thunk";
import { reducer as formReducer } from "redux-form";
import personalInfoReducer from "./infoReducer";

let reducers = combineReducers({
  catalog: catalogReducer,
  personalInfo: personalInfoReducer,
  products: productsReducer,
  form: formReducer,
  auth: authReducer,
  profile: profileReducer,
  cart: cartReducer,
  favorite: favoriteReducer,
});
const composeEnhancers = window.__REDUX_DEVTOOLS_EXTENSION_COMPOSE__ || compose;
const store = createStore(reducers, /* preloadedState, */ composeEnhancers(
    applyMiddleware(thunkMiddleware)
  ));

export default store;
```

Рисунок 3.22 – Створення Redux store

У коді на рис. 3.22 створюється *Redux store* для керування станом застосунку в середовищі React.

Store (сховище) є центральним компонентом *Redux*. Це об’єкт, який містить глобальний стан застосунку. Роль Store полягає в тому, щоб зберігати і надавати доступ до даних, які необхідні в застосунку. Store являє собою деревоподібну структуру даних, де кожна частина стану має свій шлях (схожий на шлях до файлу у файловій системі). Це означає, що доступ до даних здійснюється через ключі, і кожен ключ вказує на конкретну частину стану [19].

Redux імпорти:

1) *applyMiddleware*: використовується для застосування middleware до Redux store.

2) *combineReducers*: об'єднує декілька ред'юсерів в один.

3) *compose*: використовується для комбінування декількох функцій enhancers.

4) *createStore*: створює Redux store.

Ред'юсери: Імпортовані ред'юсери, які будуть використовуватися для обробки різних частин стану застосунку:

1) catalogReducer, authReducer, profileReducer, productsReducer, cartReducer, favoriteReducer, personalInfoReducer;

2) *formReducer*: ред'юсер для обробки форм, наданий бібліотекою *redux-form*.

Middleware:

thunkMiddleware: middleware, який дозволяє писати асинхронні логіки, що можуть виконуватися з Redux actions.

Принцип того, як працює redux, ґрунтується на таких концепціях [20]:

1) *Actions*: це об’єкти, які представляють зміни стану застосунку.

2) *Reducers*: це функції, які визначають, як стан програми змінюється у відповідь на дії.

В коді на рис. 3.23 можна побачити цілісну картину catalogReducer. Даний Reducer відповідає за декілька основних функцій:

1) Отримує товари з бази даних та записує їх до масиву products;

2) Отримує товар з визначеним Id та записує його до об’єкту product;

3) Отримує коментарі до кожного товару, та надає можливість їх залишати та видаляти.

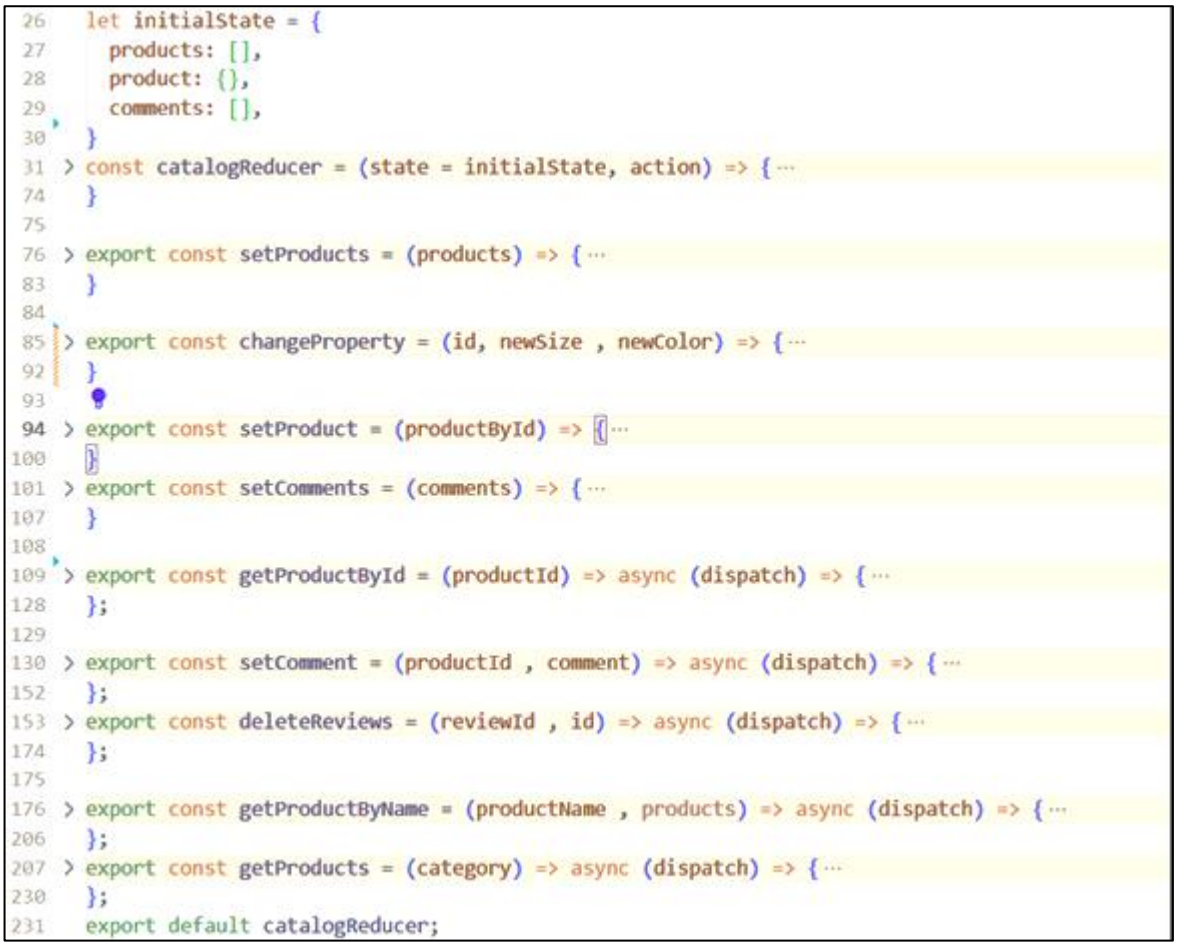

Рисунок 3.23 – catalogReducer

Роздивимося детально на прикладі отримання товарів за категорією:

```
export const getProducts = (category) => async (dispatch) => {
  try {
      let response = await productsAPI.getProducts(category);
      if (response && response.data) {
          dispatch(setProducts(response.data));
          console.log('Men products:', response.data);
      } else {
          console.error('Unexpected response structure:', response.data);
      }
  } catch (error) {
      if (error.response) {
          console.error('HTTP Error:', error.response.status);
          console.error('Response data:', error.response.data);
          console.error('Response headers:', error.response.headers);
      } else if (error.request) {
          console.error('No response received for the request:', error.request);
      } else {
          console.error('Error setting up request:', error.message);
      }
  }
};
```

Рисунок 3.24 – Асинхронний запит до endpoint getProducts

В коді який приведений на рис. 3.24, можна побачити асинхронний запит до endpoint getProducts, що визивається з об’єкту ProductsAPI.

Конструкція try { ... }: виконує спробу отримати дані з API і обробити відповідь.

Конструкція “*if (response && response.data) { ... }*”: перевіряє, чи є відповідь і чи містить вона дані.

Конструкція catch (error) { ... }: обробляє будь-які помилки, що виникають під час запиту.

Якщо *if (response && response.data) { ... }* відповідає і повертається з даними, то передаємо дані до ActionCreator під назвою setProducts. Ця функція забезпечує надійний спосіб взаємодії з API та керування станом продуктів у застосунку.

ActionCreator – це просто функція, яка повертає об’єкт дії. В ActionCreator під назвою setProducts ми передали весь масив даних, отриманий із сервера.

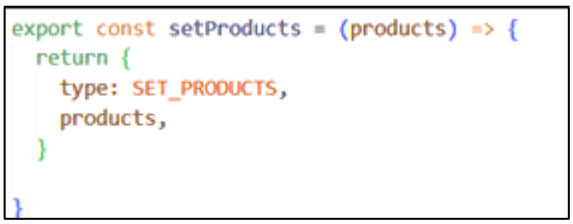

Рисунок 3.25 – Функція ActionCreator

type: тип дії, який визначає, яку саме дію необхідно виконати;.

products: параметр, що був переданий з асинхронної функції getProducts.

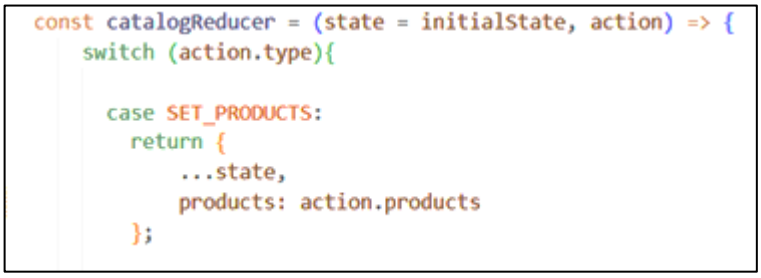

Рисунок 3.26 – Функція catalogReducer

В коді на рис. 3.26 оголошується функція catalogReducer, яка є редуктором для стану каталогу продуктів. Вона приймає два аргументи: поточний стан state

(зі значенням за замовчуванням initialState) і дію action що міститься в ActionCreator.

Далі використовується конструкція switch-case, щоб обробляти різні типи дій, які можуть бути виконані у редукторі та залежно від *type* виконувати цю дію.

В *return*: повертається новий об'єкт *state*, який містить оновлену інформацію про продукти.

```
        default:
            return state
    }

}
```

Рисунок 3.27 – Return

В коді на рис. 3.27 продемонстровано конструкцію, яка виповниться, якщо тип дії не відповідає жодному випадку. В такому разі повертається поточний стан без змін.

3.3.3 Робота з компонентами React

Компоненти дають змогу розбити інтерфейс на незалежні частини, про які легко думати окремо. Їх можна складати разом і використовувати кілька разів [21].

Пишучи код на React, програма становиться SPA, тобто звичайні посилання на сторінки потрібно замінити на *NavLink*.

```
const Header = (props) => {
    const headerLinkItems = [
        {
            links: ['Shop', 'Men', 'Women', 'Combos', 'Joggers'],
            path: ['/Main', '/Man', '/Women', '/Women', '/Man']
        }
    ]
```

Рисунок 3.28 – Об'єкт headerLinkItems

```
<nav>
    <ul className="menu">
        {headerLinkItems.map((item, index) => (
            <li key={index} onClick={() => { setActiveLink(true) }} className='menu__item'>
                {item.links.map((link, linkIndex) => (
                    <NavLink key={linkIndex} className={activeLink && 'menu__item-link active' || 'menu__item-link'} to={item.path[linkIndex]}>
                        {link}
                    </NavLink>
                ))}
            </li>
        ))}
    </ul>
</nav>
```

Рисунок 3.29 – Клас menu

У фрагментах коду наведених на рисунках 3.28 та 3.29 створюється навігаційне меню в шапці застосунку, яке складається з різних пунктів меню, кожен з яких містить список посилань.

Об'єкт *headerLinkItems* містить масив об'єктів, кожний з яких подає окремий розділ меню.

Спочатку метод *map* використовується для обходу масиву *headerLinkItems* та рендерингу кожного пункту меню, потім для обходу масиву посилань в *item.links*.

Коли переходимо за посиланням, в URL адресу додається відповідний *path*.

```
render() {
  return (
    <div className='app-wrapper'>
      <Header />
      <main className='main'>
        <Suspense fallback={<div>wait</div>}>
          <Routes>
            <Route path="/" element={<Navigate to="/Main" />} />
            <Route path="/Main" element={<Hero />} />
            <Route path="/Women" element={<WomansProducts />} />
            <Route path="/Man" element={<MansProducts />} />
            {/* ////////////////////////////////////////////////// */}
            <Route path="/Description/:id" element={<DetailsContainer />} />
            <Route path='/Cart' element={<Cart />} />
            <Route path='/Details' element={<Details />} />
            {/* ////////////////////////////////////////////////// */}
            <Route path='/CheckOut' element={<CheckOut />} />
            <Route path='/SignIn' element={<SignIn />} />
            <Route path='/SignUp' element={<SignUp />} />
            <Route path='/Check' element={<Check />} />
            <Route path='/Reset' element={<Reset />} />
            {/* ////////////////////////////////////////////// */}
            <Route path='/Profile/Active' element={<Active />} />
            <Route path='/Profile/Cancelled' element={<Cancelled />} />
            <Route path='/Profile/Completed' element={<Completed />} />
            <Route path='/Profile/MyInfo' element={<MyInfo />} />
            <Route path='/Profile/Wishlist' element={<Wishlist />} />
            <Route path="*" element={<div>404 NOT FOUND</div>} />
          </Routes>
        </Suspense>
        <Footer />
      </main>
    </div>
  );
}
}
```

Рисунок 3.30 – Код написано автором у редакторі кода Visual Studio Code

Якщо *URL-адреса* збігається з якоюсь з поданих, то до цього шляху підвантажиться відповідна компонента. Якщо щось сталося і URL не буде знайдений, то спрацює останній *Route,* який покаже, що сторінку не знайдено.

Після того як клієнт перейде до пункту меню *Men* або *Women*, спрацює UseEffect.

```
const MansProducts = (props) => {
    const [isSwitched , setSwitch] = useState(false)
    const [isNewActive , setActiveNew] = useState(true)
    const [isRecActive , setActiveRec] = useState(false)


    useEffect(() => {
        props.getProducts('man')
    }, {})
```

Рисунок 3.31 – Функція *thunkMiddleware*

Хук useEffect дозволяє виконувати side effect в функціональних компонентах, що означає що він виконається раніше, ніж зарендериться сторінка.[22] На рис. 3.31 можна побачити, що в *useEffect* визивається *thunkMiddleware* функція, в котру передаємо назву категорії і завдяки коду описаному раніше, на сторінці відобразяться чоловічі товари. На сторінці с жіночими товарами все теж само, але передається вже відповідна категорія.

3. 4 Функціональне тестування

Функціональне тестування – це вид тестування програмного забезпечення, спрямований на перевірку того, як функціональність програми відповідає її вимогам. Основна мета функціонального тестування - переконатися, що програмне забезпечення працює так, як очікувалося, і виконує свої функції коректно [23].

До такого тестування входить:

1) Тестування реєстрації, входу в систему та керування обліковим записом;
2) Перевірка роботи сортування товарів та огляд певних товарів;

3) Перевірка основних функцій інтернет-магазину, таких як додавання товарів у кошик, пошук товару за назвою та ін.

3.4.1 Тестування реєстрації, входу в систему та керування обліковим записом

Реєстрація нового користувача

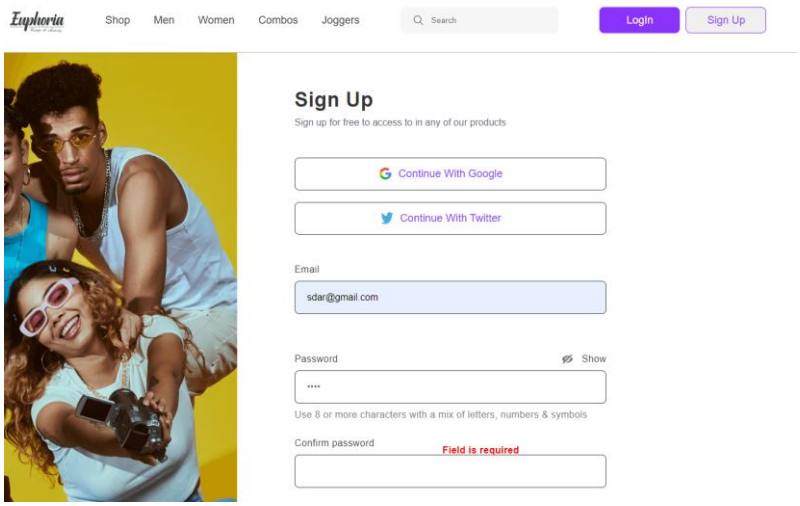

Рисунок 3.32 – Сторінка реєстрації клієнта

На рис. 3.32 є необхідні поля для заповнення даних. Якщо залишити якесь поле пустим, то побачимо валідацію. Коли всі умови будуть виконані, то аккаунт буде успішно створено.

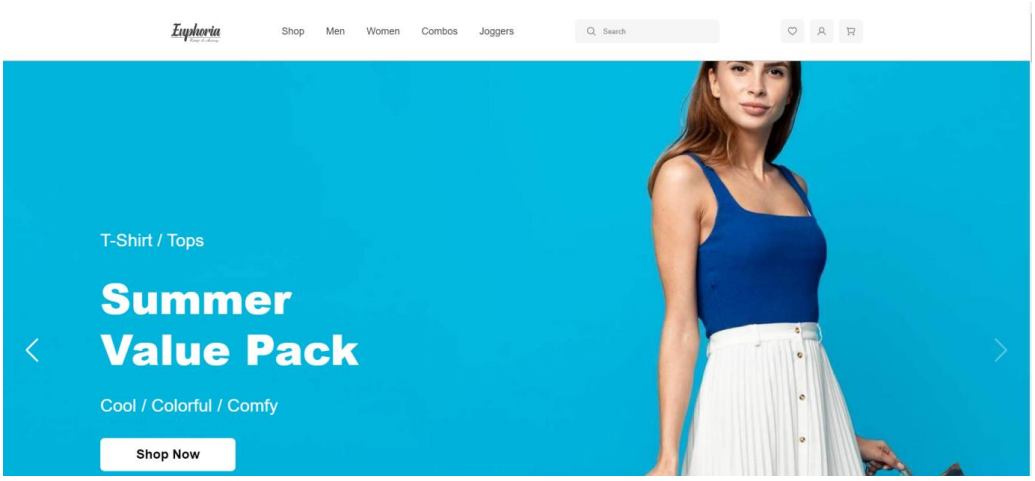

Рисунок 3.33 – Сторінка реєстрації клієнта

Зрозуміти це можна буде по тому, що зі сторінки “*Sign Up*”, клієнта переадресовує на головну сторінку, та в “шапці” застосунку, замість двух кнопок “*LogIn*” та “*Sign Up*” з’являються три іконки, що ведуть до різних сторінок застосунку.

Перша іконка переадресовує користувача до списку бажаних товарів, друга до особистого кабінету, а третя до кошику товарів.

Огляд особистих даних та зміна їх.

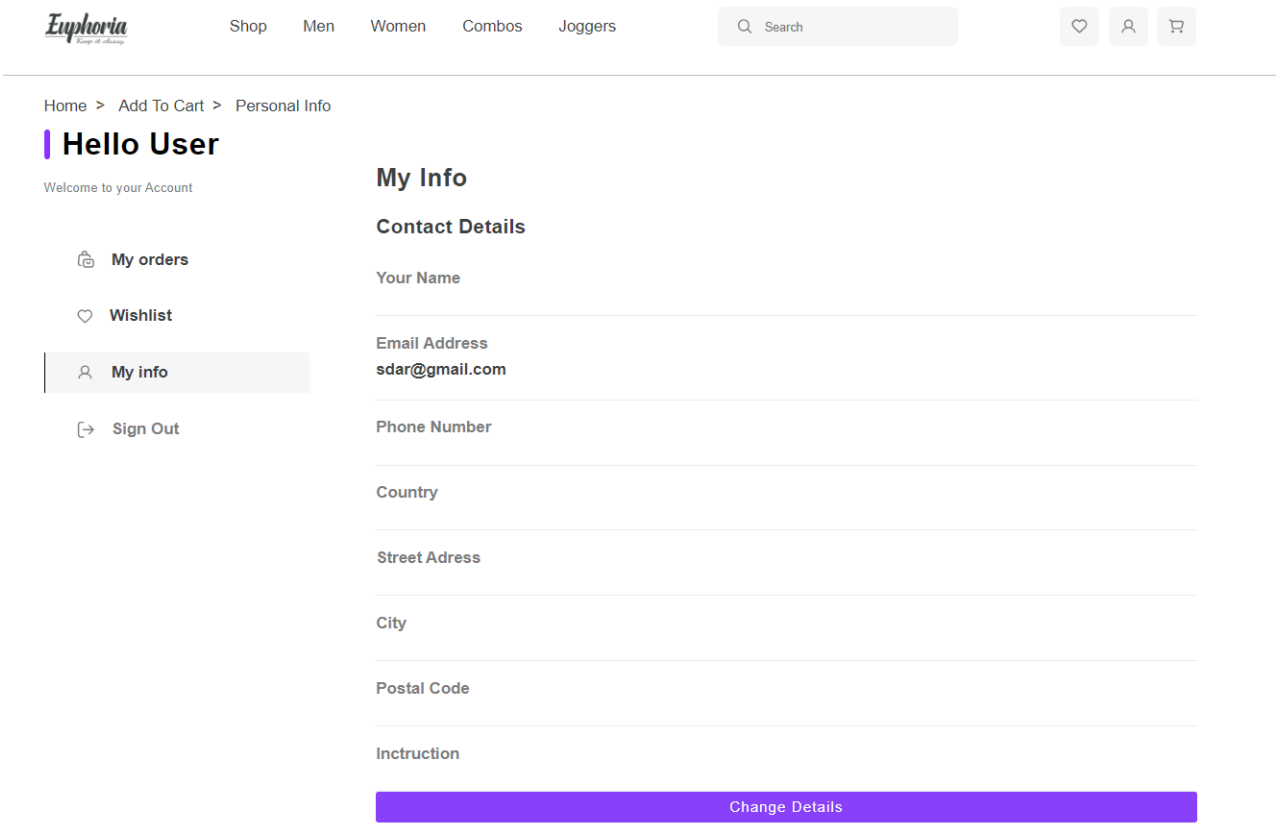

Рисунок 3.34 – Сторінка особистих даних

Якщо в особистому кабінеті обрати в меню зліва розділ “*My info*”, то побачимо інформацію, які надав клієнт. Але оскільки, при реєструванні програма просить лише пошту та пароль, відповідно і бачимо на рис. 3.34 тільки введену адресу.

Змінити це можна натиснувши кнопку “*Change Detaise*”.

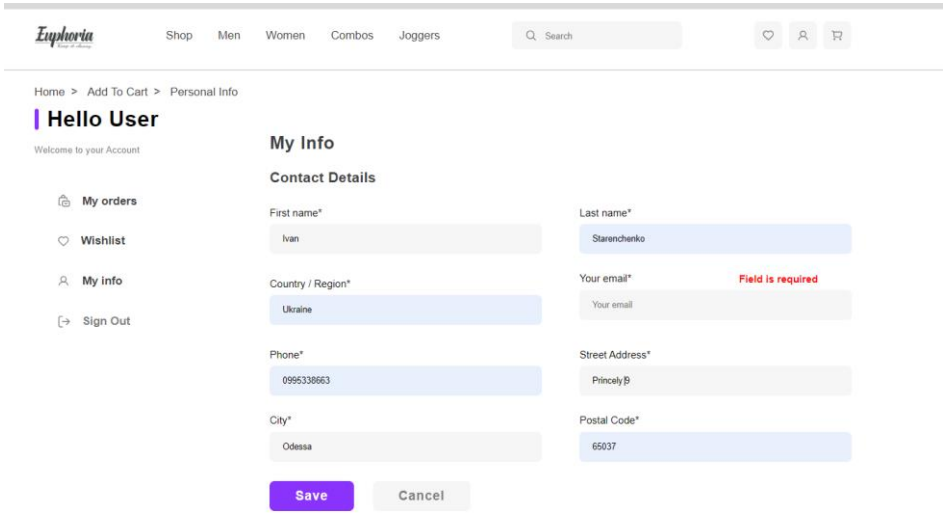

Рисунок 3.35 – Поля для додавання та оновлення даних користувача

Рис. 3.35 показує, що тут також працює валідація, а також можна змінити пошту, після чого вхід до акаунта буде відбуватися саме за новою поштою.

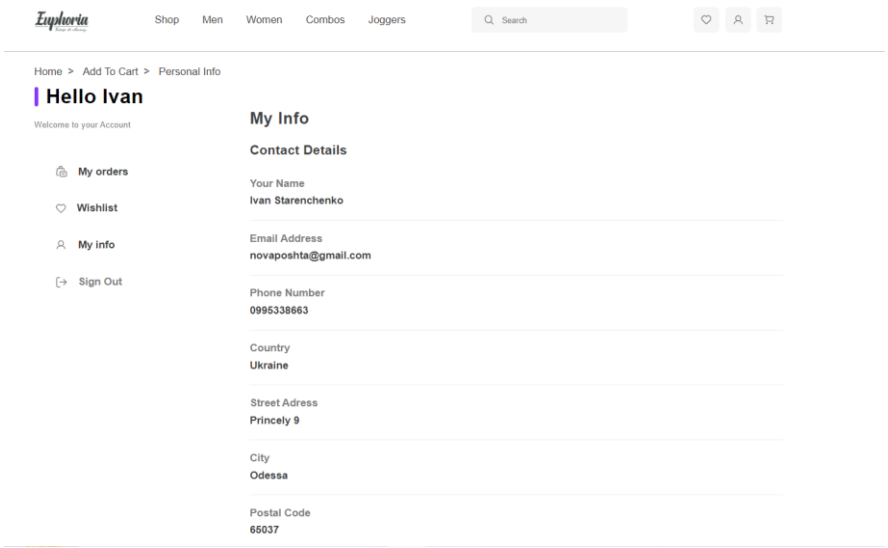

Рисунок 3.36 – Оновлені дані користувача

Після успішного збереження даних, на рис. 3.36 бачимо, що пусті поля заповнились введеними даними, з’явилися зміни в вітальному написі над меню.

У меню зліва також можна побачити кнопку “*Sign Out*”, її натискання характерно призведе до негайного виходу з акаунту.

Після цього в акаунт можна зайти, попередньо перейшовши на сторінку “*LogIn*”, та вписавши свої дані (рис. 3.37).

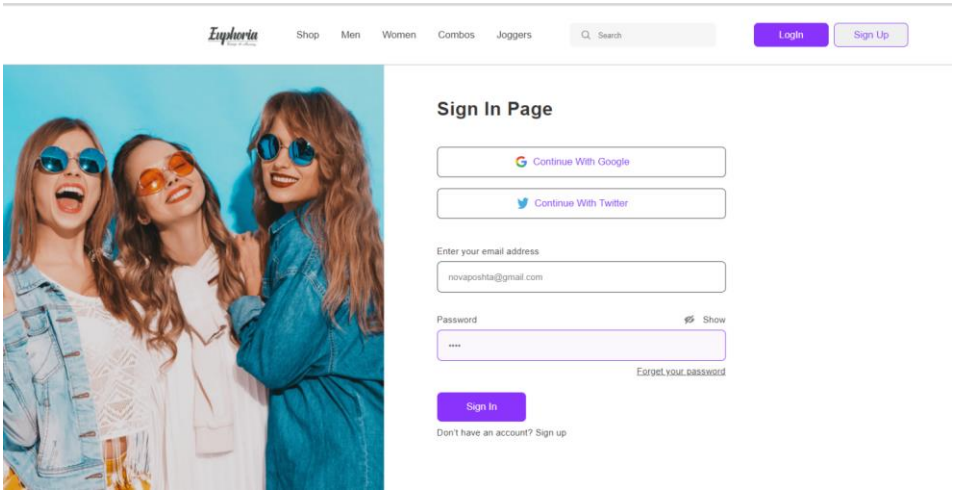

Рисунок 3.37 – Сторінка для входу до акаунту

3.4.2 Перевірка роботи сортування товарів та огляд певних товарів

Огляд товарів за категоріями

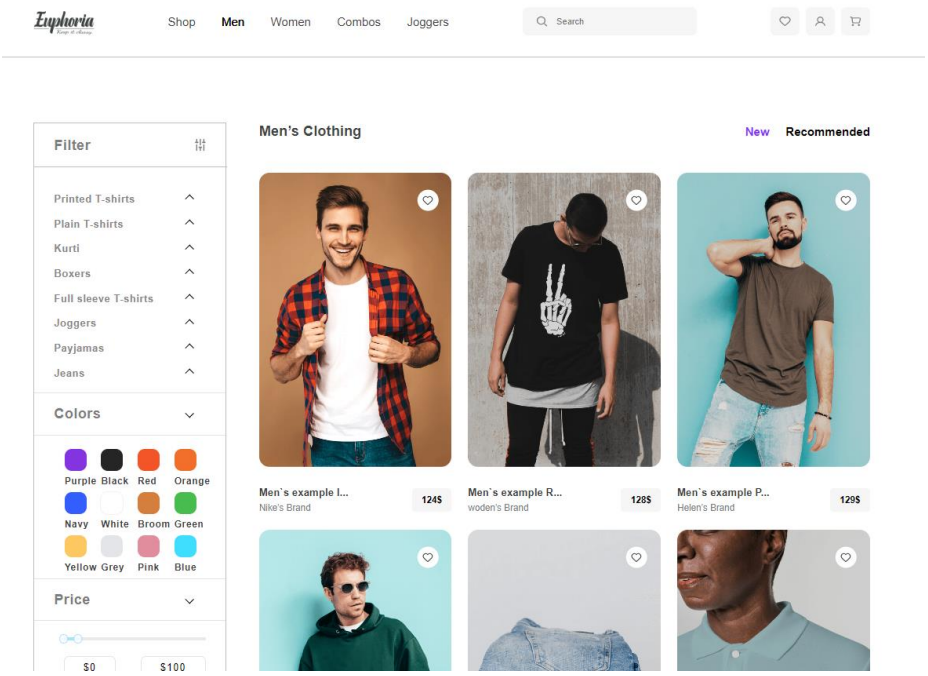

Рисунок 3.38 – Товари з чоловічим одягом

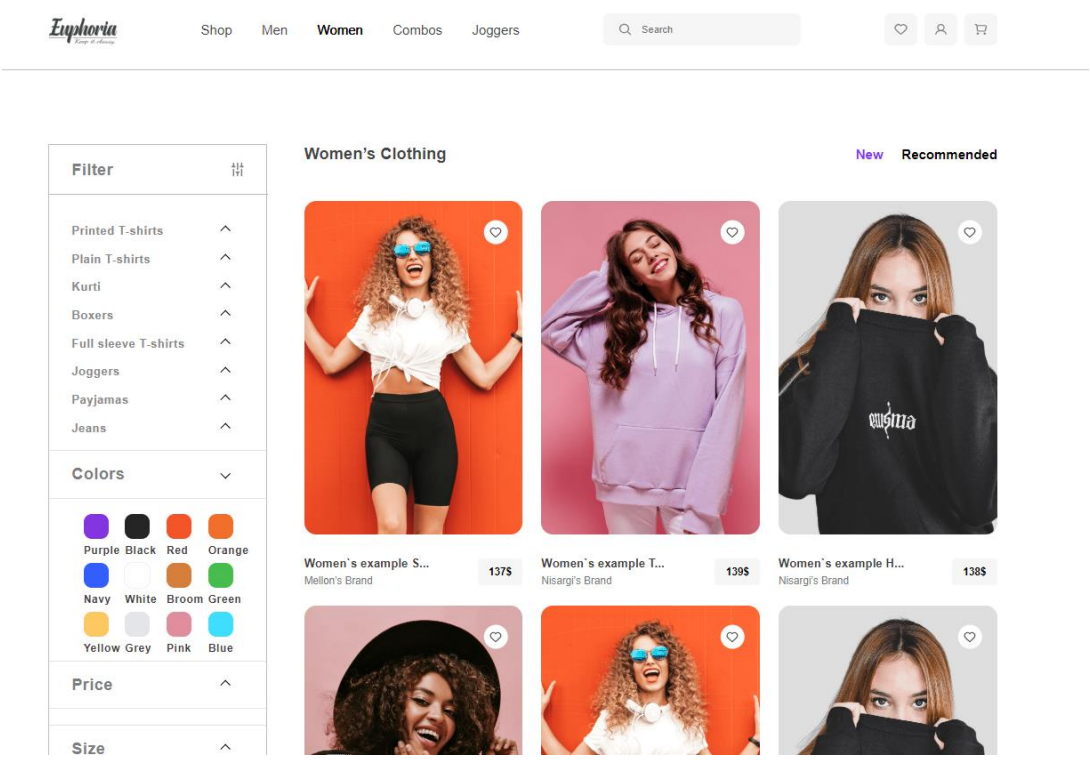

Рисунок 3.39 – Товари з жіночим одягом

На рис. 3.38 та 3.39 можна побачити сортування по категоріях жіночого та чоловічого одягу.

Огляд певного товару

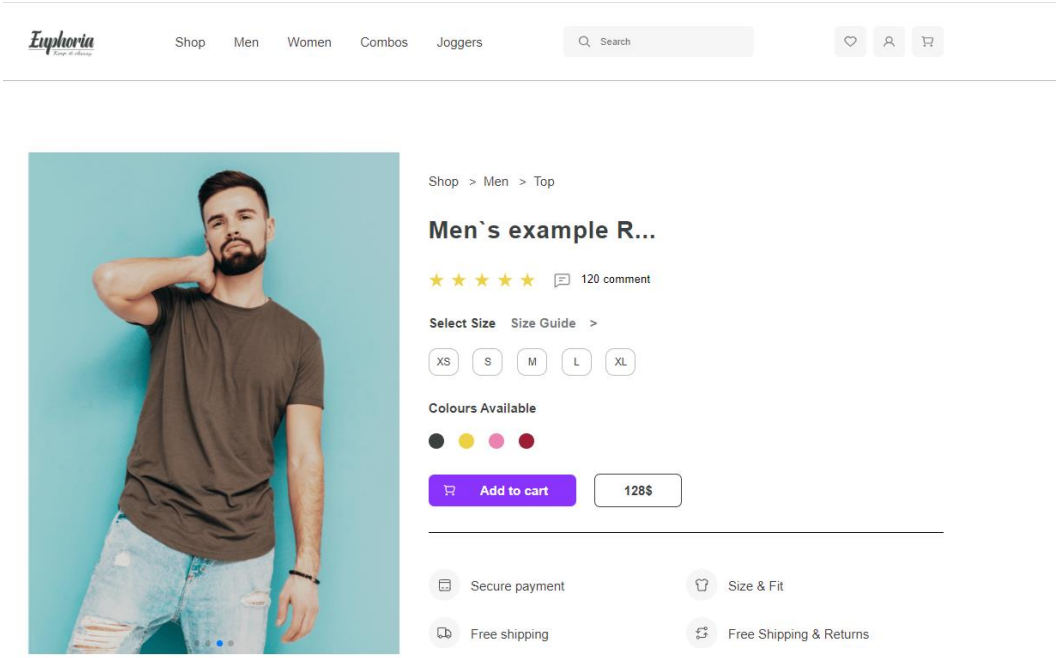

Рисунок 3.40 – Сторінка з товаром

Можливість переходити на сторінку товару надається тільки авторизованим користувачам, інших система переадресовує до сторінки логіну.

На самій сторінці знаходиться декілька фотографій товару, які можна скролити за допомогою бібліотеки слайдеру “*swiper/react*” та його модулів. Побачити наявність кількох зображень можна подивившись на пагінацію внизу зображення. Саме виведенне на екран, це третє з чотирьох наявних.

```
import { Pagination, Scrollbar, Navigation, Thumbs, FreeMode } from 'swiper/modules';

import { Swiper, SwiperSlide } from 'swiper/react';
import 'swiper/swiper-bundle.css';
```

Рисунок 3.41 – Підключення бібліотеки “swiper/react”

```
return (
    <Swiper
        modules={[Pagination, Scrollbar, Navigation, Thumbs, FreeMode]}
        thumbs={{ swiper: thumbsSwiper }}
        spaceBetween={30}
        slidesPerView={1}

        pagination={{ clickable: true }}
    >
        {images.map((slide, index) => (
            <SwiperSlide key={index} className="swiper-slide">
                <img src={slide} />
            </SwiperSlide>
        ))}
    </Swiper>
)
```

Рисунок 3.42 – Використання бібліотеки в коді

Окрім зображень можна побачити усю наявну інформацію про товар, таку як ціна, наявні розміри та кольори.

Якщо переглянути сторінку нижче, буде видно індивідуальний опис товару та коментарі до нього.

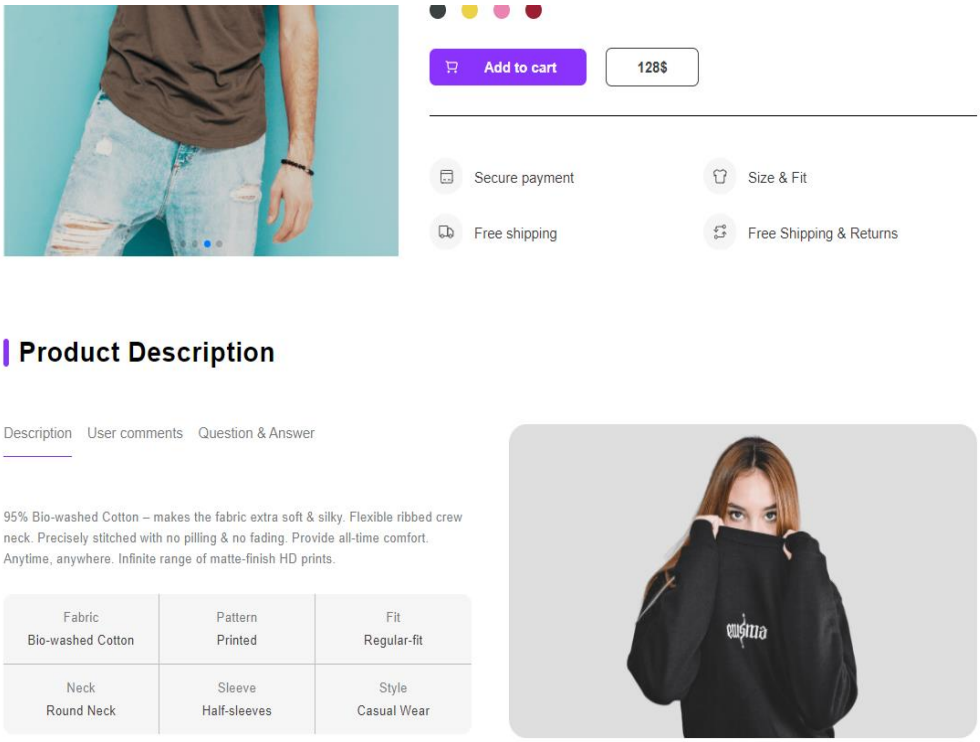

Рисунок 3.43 – Огляд опису товару

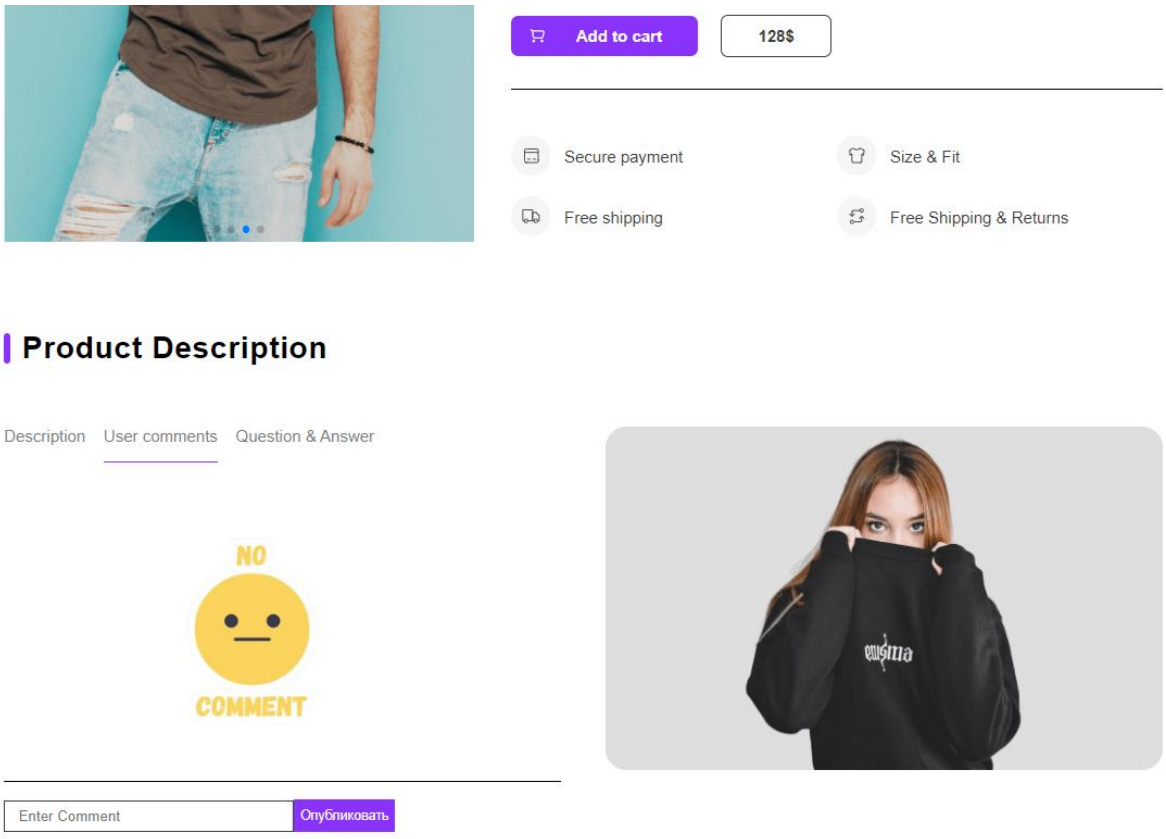

Рисунок 3.44 – Огляд коментарів товару

3.4.3 Перевірка основних функцій інтернет-магазину

Додавання та видалення коментарів

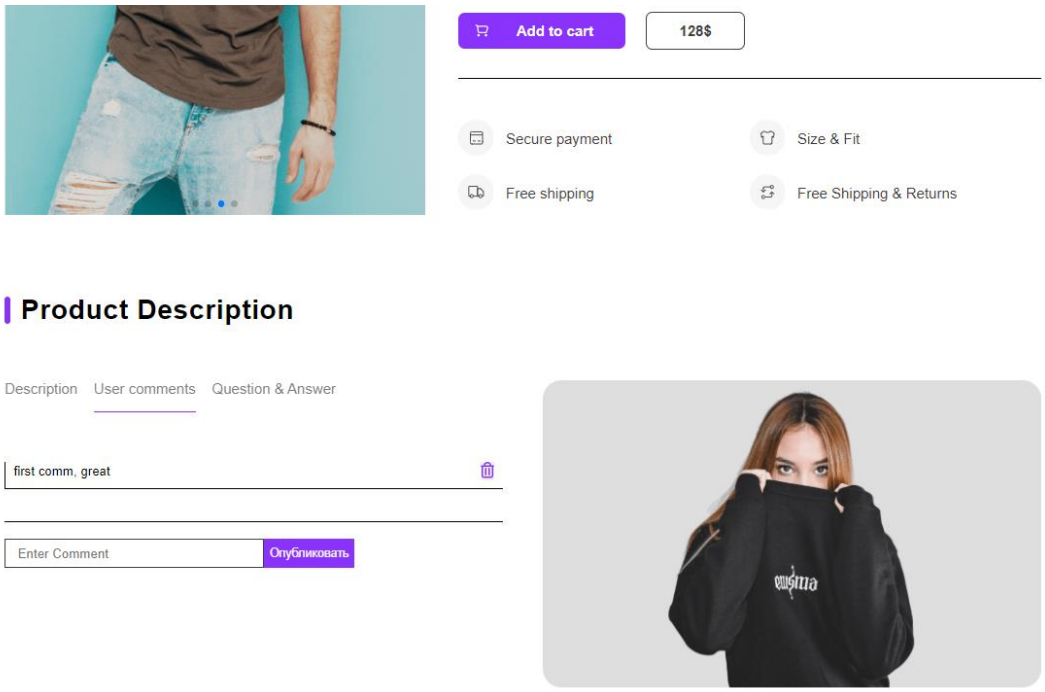

Рисунок 3.45 – Приклад коментаря

На рис. 3.45 був доданий коментар, який можна видалити просто натиснувши на іконку кошику. Якщо це був один єдиний коментар, клієнт побачить пусте поле зі “смайликом” як на рис. 3.44, в інакшому випадку, він просто зітреться. Користувач має право видаляти виключно власні коментарі.

Додавання товарів до кошику та до бажаних товарів

На рис. 3.44 можна побачити кнопку “*Add to cart*”, натиснувши котру можна додати товар до кошику, попередньо обравши характеристики бажаного товару.

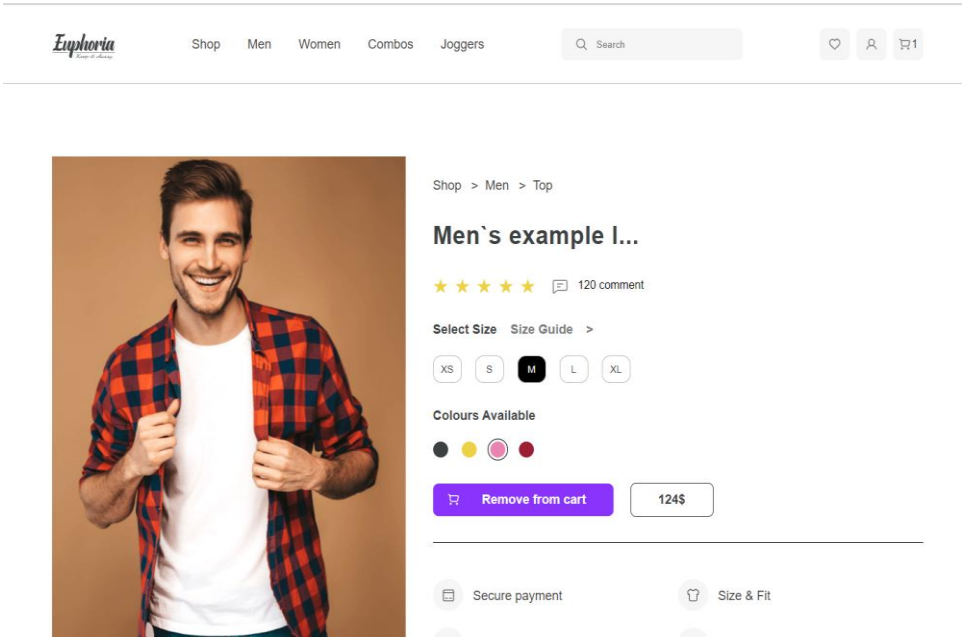

Рисунок 3.46 – Приклад додавання до кошику

На рис. 3.46 був доданий товар розміру “*M*” та рожевим кольором. Переконатися що товар був дійсно доданий до кошику, не переходячи до самої сторінки кошику можна побачити по тому як змінилась кнопка “*Add to cart*”, та по тому, що з’явився лічильник біля іконки кошику. Можна додавати один й той же товар, але з різними характеристиками, вони будуть йти як різні товари, тому якщо наприклад натиснути на інший колір або розмір, то ми знов побачимо “*Add to cart*”.

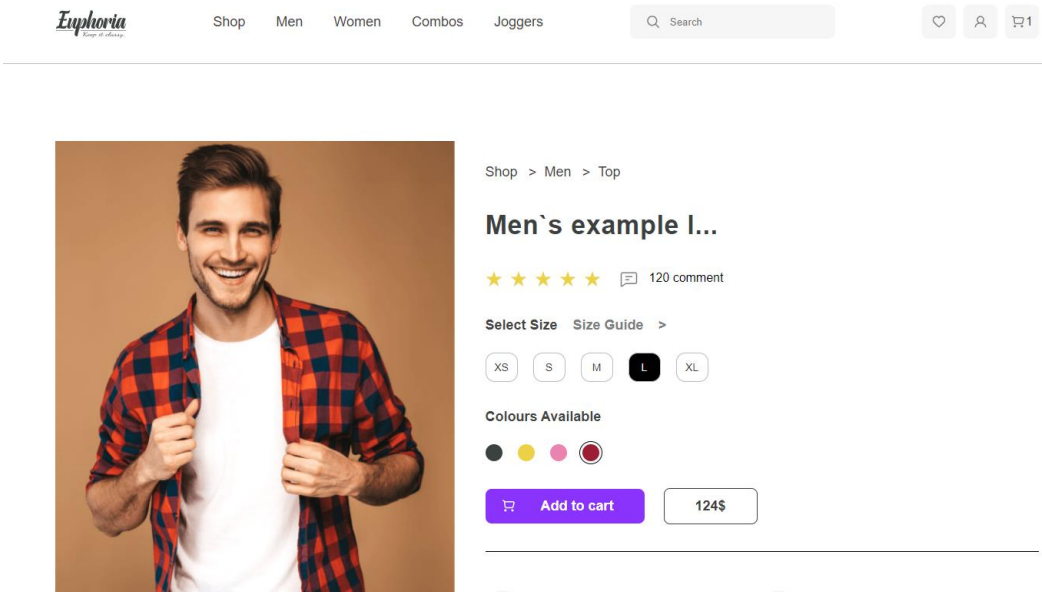

Рисунок 3.47 – Приклад зміни характеристик одного товару

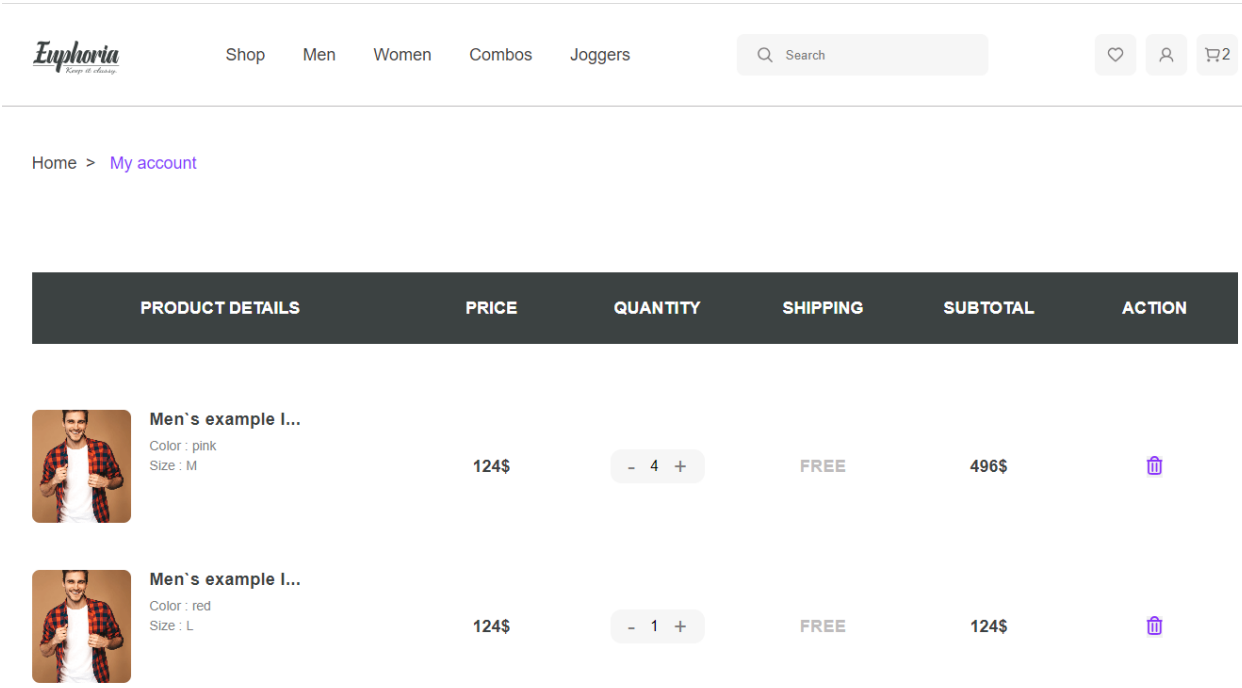

Рисунок 3.48 – Огляд кошику

На рис. 3.48 можна побачити сторінку кошику з вже доданими товарами та їх характеристиками з рис. 3.46 та 3.47, відповідно. У першого товару була змінена кількість товарів, тому і фінальна ціна також більша.

Видалити товари можна або зі сторінки товару, натиснувши на кнопку “*Remove from cart*”, яка змінює *“Add to cart*” (рис. 3.46), або напряму зі сторінки кошику, натиснувши іконку кошику.

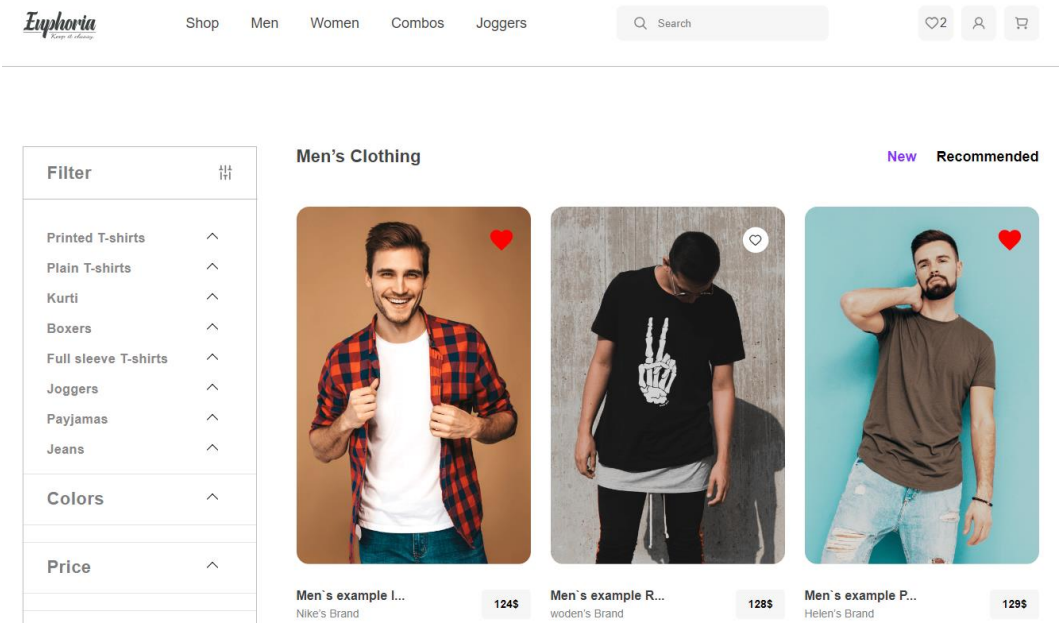

Рисунок 3.49 – Додавання товару до бажаних товарів

Додавати до “*Wishlist*” та видаляти звідти можна з каталогу товарів просто натиснувши на іконку серця.

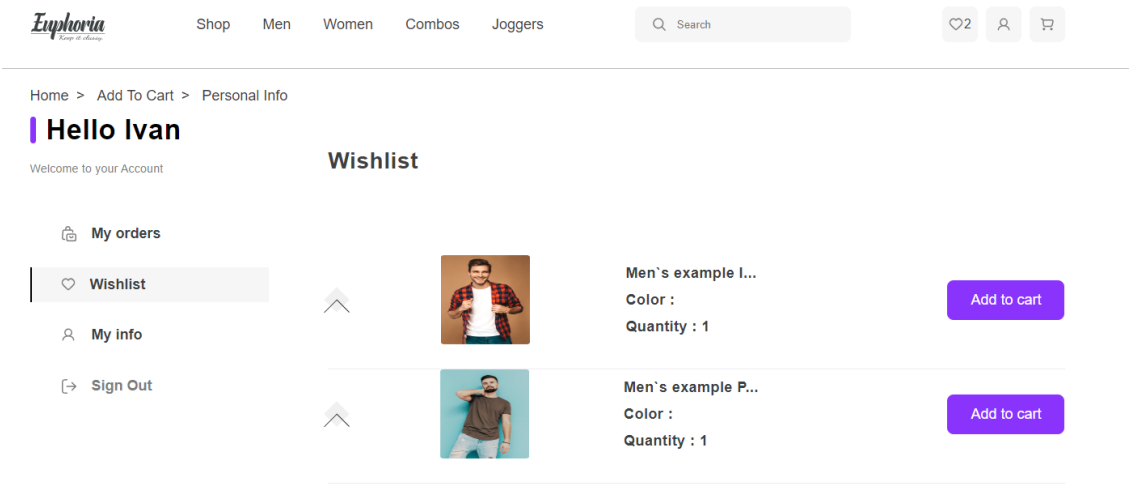

Рисунок 3.50 – Огляд бажаних товарів

Зі сторінки “*Wishlist*” також можна видаляти товари та додавати до кошику.

Пошук товару за його назвою

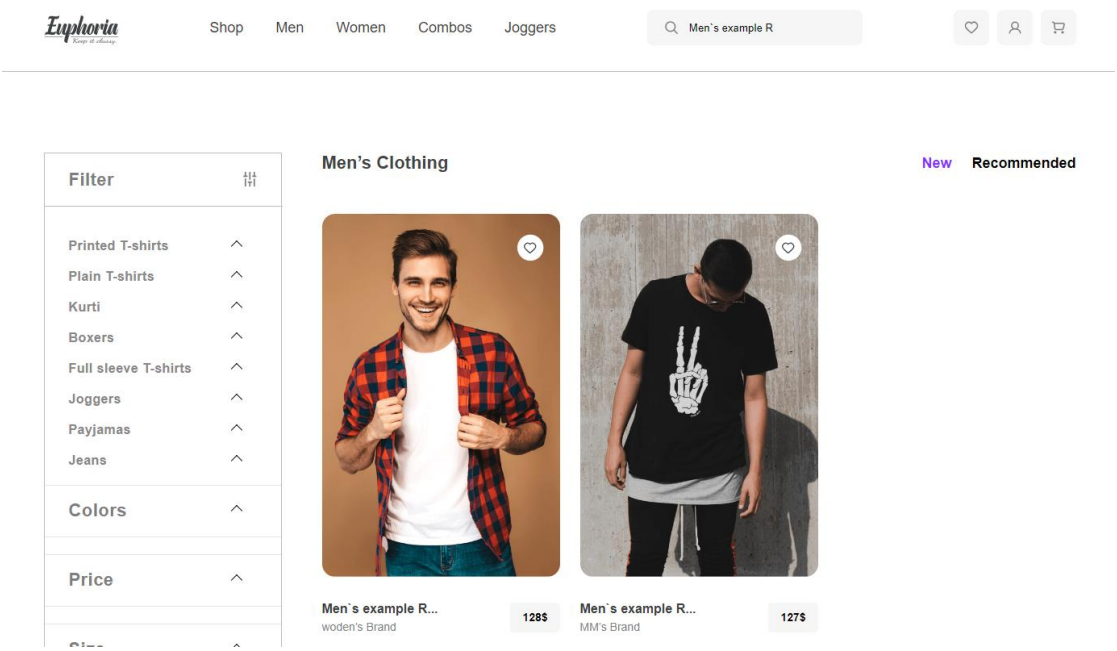

Рисунок 3.51 – Здійснення пошуку товару

Пошук здійснюється в спеціальній пошуковій формі в “шапці” застосунку. Його можна здійснити лише з каталогів.

3.5 Висновки до розділу

У розділі розглянуто процес розробки інтернет-магазину, що відповідає сучасним вимогам електронної комерції. Було описано структуру системи, яка включає базу даних, серверну та клієнтську частини.

Було створено та структуровано таблиці бази даних, необхідні для функціонування інтернет-магазину. Розглянуто детальну структуру таблиць, що включають інформацію про користувачів, товари, замовлення та інші важливі аспекти системи.

Серверна частина була розроблена з використанням Dot.net. Використання Microsoft.EntityFrameworkCore дозволило створити ефективну та гнучку ORM для взаємодії з базою даних MSSQL. Це забезпечує надійність та масштабованість серверної частини.

Клієнтська частина інтернет-магазину була розроблена за допомогою React, що забезпечує створення зручного та інтерактивного інтерфейсу користувача. Використання Redux для керування станом застосунку, Redux-form для обробки форм та Axios для взаємодії з API забезпечили ефективність та зручність роботи з застосунком.

Описано результати функціонального тестування створеного вебсайту. Проведено тестування реєстрації користувачів та входу до системи, керування обліковим записом. Перевірена робота сортування товарів та основних функцій інтернет-магазину.

ВИСНОВКИ

У кваліфікаційній роботі розроблено інтернет-магазин засобами технологій React JS, MSSQL, Dot.net та Microsoft.EntityFrameworkCore. Використання такого стека технологій дозволяє створити надійний, масштабований та зручний у використанні інтернет-магазин.

Для розробки клієнтської частини використано React JS. Для керування станом застосунку та керування змінами використовувався Redux. Для роботи з формами та їх валідацією використовувалась бібліотека Redux-form, а для взаємодії з API – Axios.

Серверна частина магазину побудована засобами Dot.net, що забезпечує реалізацію бізнес-логіки та оброблення запитів. Microsoft.EntityFrameworkCore використовувався для взаємодії з базою даних MSSQL, забезпечуючи надійну роботу з даними.

Під час розробки архітектури програмного застосунку була розроблена детальна структура, яка охоплює частини та взаємодію кожної частини системи. Це гарантує, що різні модулі програми добре працюють один з одним.

База даних розроблена так, щоб враховувати вимоги системи, а також методи зберігання та обробки даних. Модель бази даних демонструє структуру та взаємозв’язки між таблицями, що полегшує зберігання та керування різними частинами системи.

Моделювання програмної системи проводилося за допомогою діаграм UML: Use Cases, Classes та Activity. Такі діаграми допомогли краще зрозуміти функціональні можливості системи.

Проведено функціональне тестування створеного вебсайту електронної комерції.

У результаті виконання дослідження був розроблений інтернет-магазин, який відповідає сучасним вимогам електронної комерції. Користувач може створювати аккаунт та в подальших сесіях авторизуватись по вписаним при створюванні акаунту даним, шукати товари по категоріях та їх назві, додавати їх у кошик та у бажані товари, а також керувати особистими даними.

Розроблений інтернет-магазин надає користувачам добре злагоджений функціонал та дозволяє користуватись ним з будь-якого пристрою. Його актуальність полягає в тому, що онлайн-покупки стали однією з найпопулярніших форм придбання товарів, тому попит на якісні та зручні інтернет-магазини продовжує зростати. Використання передових технологій та інструментів розробки забезпечили якісний результат та гарантували високу ефективність магазину. Такий проєкт має великий потенціал у сфері електронної комерції та є актуальним у сучасному світі онлайн-торгівлі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Getting Started with Redux | Redux. URL: https://redux.js.org/introduction/getting-started

2. Managing Form State in React With Redux Form | DigitalOcean URL:https://www.digitalocean.com/community/tutorials/managing-form-state-in-react-with-redux-form

3. Axios in React: A Guide for Beginners - GeeksforGeeks URL: https://www.geeksforgeeks.org/axios-in-react-a-guide-for-beginners/

4. Amazon (company). URL: https://en.wikipedia.org/wiki/Amazon_(company)

5. eBay URL: https://en.wikipedia.org/wiki/EBay

6. AliExpress URL: https://eu.wikipedia.org/wiki/AliExpress

7. SQL Server Management Studio (SSMS) - SQL Server Management Studio (SSMS) URL: https://learn.microsoft.com/en-us/sql/ssms/sql-server-management-studio-ssms?view=sql-server-ver16

8. Visual Studio Code Frequently Asked Questions URL: https://code.visualstudio.com/docs/supporting/FAQ#:~:text=Visual%20Studio%20Code%20is%20a,such%20as%20Visual%20Studio%20IDE.

9. User interface это про обеспечение пользователям удобства URL: https://foxminded.ua/ru/user-interface-eto

10. Single-page application. URL: https://en.wikipedia.org/wiki/Single-page_application

11. What do client side and server side mean? | Client side vs. Serverside | Cloudflare. URL:https://www.cloudflare.com/learning/serverless/glossary/client-side-vs-server-side/

12. Що таке API? – Опис інтерфейсів прикладного програмування – AWS URL: https://aws.amazon.com/ru/what-is/api/

13. Redux Fundamentals, Part 3: State, Actions, and Reducers | Redux. URL: https://redux.js.org/tutorials/fundamentals/part-3-state-actions-reducers

14. Understanding Asynchronous Redux Actions with Redux Thunk | DigitalOcean. URL: https://www.digitalocean.com/community/tutorials/redux-redux-thunk

15. Entity Framework Core Migrations. URL: https://www.learnentityframeworkcore.com/migrations#:~:text=EF%20Core%20Migrations%20is%20a,evolves%2C%20without%20losing%20existing%20data.

16. How to Use Axios with React: The Ultimate Guide [2024] URL:https://www.knowledgehut.com/blog/web-development/axios-in-react

17. Axios Interceptors in React. URL: https://dev.to/sundarbadagala081/axios-interceptors-34b2

18. REST API for Oracle Enterprise Performance Management Cloud. URL:https://docs.oracle.com/en/cloud/saas/enterprise-performance-management-common/prest/rest_api_methods.html

19. Що таке redux і в яких випадках варто його використовувати URL: https://foxminded.ua/shcho-take-redux

20. Redux Fundamentals, Part 3: State, Actions, and Reducers | Redux. URL:https://redux.js.org/tutorials/fundamentals/part-3-state-actions-reducers

21. Components and Props – React. URL: https://legacy.reactjs.org/docs/components-and-props.html

22. ReactuseEffect URL: https://www.w3schools.com/react/react_useeffect.asp#:~:text=The%20useEffect%20Hook%20allows%20you,The%20second%20argument%20is%20optional.

23. Foundations of Software Testing" by Dorothy Graham, Erik van Veenendaal, Isabel Evans, and Rex Black. URL: https://www.utcluj.ro/media/page_document/78/Foundations%20of%20software%20testing%20-%20ISTQB%20Certification.pdf

ДОДАТКИ

Додаток А. Фрагменти програмного коду

Backend

MappingConfig.cs:

```
using AutoMapper;
using Marcet Api V2.Model.DTO.Product;
using Marcet_Api_V2.Models;
using Models.Dto.Auth;
using Models.Dto.UserAddInformation;
using TestRest.Models.Dto;

namespace TestRest
{
    public class MappingConfig : Profile
    {
        public MappingConfig()
        {
            CreateMap<AuthRequestWithGoogleDTO, Customer>();
            CreateMap<Customer, AuthUserResponseWithGoogleDTO>();
            CreateMap<CustomerDTO, Customer>();
            CreateMap<Product, ProductDTO>().ReverseMap();
        }
    }
}
```

Program.cs:

```
namespace Marcet Api V2

{
    public class Program
    {
        public static void Main(string[] args)
        {
            var config = new ConfigurationBuilder()
                .AddJsonFile("appsettings.json", optional: true, reloadOnChange: true)
                .Build();

            CreateHostBuilder(args, config).Build().Run();
        }

        public static IHostBuilder CreateHostBuilder(string[] args, IConfiguration config)
=>
           Host.CreateDefaultBuilder(args)
               .ConfigureWebHostDefaults(webBuilder =>
               {
                   webBuilder.UseStartup<Startup>();
               })
               .ConfigureAppConfiguration((hostingContext, configuration) =>
               {
                   configuration.AddConfiguration(config.GetSection("ConnectionStrings"));
               });
    }
}
```

Startup.cs:

```
using FluentValidation;
using Marcet Api V2.Models;
using Marcet Api V2.Repository;
using Microsoft.EntityFrameworkCore;
using Marcet Api.Repository;
using Marcet Api.Services;
using Marcet Api.Services.IServices;
using Marcet Api.Services.IServices.ITokens;
using Marcet Api.Validators;
using System.Text.Json.Serialization;
using Repository.IRepository;
using Services.IServices;
using Models.Dto.Token;
using Microsoft.OpenApi.Models;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.IdentityModel.Tokens;
using System.Security.Cryptography;
using Marcet Api V2.Services.IServices;
using Marcet Api V2.Services;

public class Startup
{
    public IConfiguration Configuration { get; }

    public Startup(IConfiguration configuration)
    {
        Configuration = configuration;
    }

    public void ConfigureServices(IServiceCollection services)
    {
        services.AddDbContext<MarcetDbContext>(options =>
        {

options.UseSqlServer(Configuration.GetConnectionString("DefaultSqlConnection"));
        });

        services.AddScoped<IRepository<Customer>, Repository<Customer>>();
        services.AddScoped<IRepository<Product>, Repository<Product>>();

        services.AddScoped<ICustomerService, CustomerService>();
        services.AddScoped<IAccessTokenService, TokenService>();
        services.AddScoped<IRefreshTokenService, TokenService>();
        services.AddScoped<IHashService, HashService>();
        services.AddTransient<IAuthService, AuthService>();
        services.AddScoped<IProductService, ProductService>();
        services.AddScoped<IReviewsService, ReviewsService>();
        services.AddScoped(typeof(IRepository<>), typeof(Repository<>));
        services.AddHttpContextAccessor();


        services.AddCors();

        services.AddScoped<IRepository<RefreshToken>, Repository<RefreshToken>>();

        services.AddDistributedMemoryCache();

        services.AddScoped<IValidator<TokensDTO>, TokensDTOValidator>();

        services.AddAutoMapper(typeof(Startup));

        services.AddControllers().AddJsonOptions(x =>
        {
            x.JsonSerializerOptions.ReferenceHandler = ReferenceHandler.IgnoreCycles;
        })
         .AddNewtonsoftJson();

        services.AddSwaggerGen(c =>
        {

c.SwaggerDoc("v1", new OpenApiInfo { Title = "Your API", Version = "v1" });

            c.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme
            {
                Description = "JWT Authorization header using the Bearer scheme",
                Name = "Authorization",
```

```
                In = ParameterLocation.Header,
                Type = SecuritySchemeType.ApiKey,
                Scheme = "Bearer"
            });

            c.AddSecurityRequirement(new OpenApiSecurityRequirement
            {
                {
                    new OpenApiSecurityScheme
                    {
                        Reference = new OpenApiReference
                        {
                            Type = ReferenceType.SecurityScheme,
                            Id = "Bearer"
                        }
                    },
                    new string[] { }
                }
            });
        });

        services.AddAuthorization(options =>
        {
            options.AddPolicy("Admin", policy => policy.RequireRole("Admin"));
            options.AddPolicy("User", policy => policy.RequireRole("User"));
        });

        var publicKey = Configuration.GetValue<string>("JWT:PublicKey");
        var issuer = Configuration.GetValue<string>("JWT:Issuer");

        services.AddAuthentication(x =>
        {
            x.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
            x.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;

        }).AddJwtBearer(options =>
        {
            RSA rsa = RSA.Create();
            rsa.FromXmlString(publicKey);

            options.TokenValidationParameters = new TokenValidationParameters()
            {
                IssuerSigningKey = new RsaSecurityKey(rsa),
                ValidateIssuerSigningKey = true,
                ValidateIssuer = true,
                ValidateAudience = false,
                ValidIssuer = issuer,
                ValidateLifetime = true,
                ValidAlgorithms = new[] { SecurityAlgorithms.RsaSha256 },
            };

            options.Events = new JwtBearerEvents
            {
                OnMessageReceived = context =>
                {
                    var accessToken = context.Request.Query["access token"];

                    var path = context.HttpContext.Request.Path;
                    if (!string.IsNullOrEmpty(accessToken) &&
path.StartsWithSegments("/chatHub"))
                    {
                        context.Token = accessToken;
                    }
                    return Task.CompletedTask;
                }
            };
        });

    }

    public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

    {

            app.UseDeveloperExceptionPage();

            app.UseSwagger();
```

```
            app.UseSwaggerUI(c =>
            {
                c.SwaggerEndpoint("/swagger/v1/swagger.json", "Your API V1");

                c.OAuthClientId("swagger-ui");
                c.OAuthClientSecret("swagger-ui-secret");
                c.OAuthUseBasicAuthenticationWithAccessCodeGrant();
            });

            app.UseExceptionHandler("/Home/Error");
            app.UseHsts();

        app.UseCors(builder =>
        {
            builder.WithOrigins("http://localhost:3000")
                   .AllowAnyMethod()
                   .AllowAnyHeader()
                   .AllowCredentials();
        });

        app.UseHttpsRedirection();
        app.UseStaticFiles();

        app.UseRouting();

        app.UseAuthentication();
        app.UseAuthorization();

        app.UseEndpoints(endpoints =>
        {
            endpoints.MapControllers();
        });
    }
}
```

CustomerController.cs:

```
using FluentValidation;
  using FluentValidation;
  using Microsoft.AspNetCore.Mvc;
  using Marcet Api.Models.Dto.Auth;
  using Marcet Api.Services.IServices;
  using Marcet Api.Services.IServices.ITokens;
  using Marcet Api.Validators.Auth;
  using Models.Dto.Token;
  using Models.Dto.Auth;

  namespace Marcet Api.Controllers
  {
      [Route("api/auth")]
      [ApiController]
      public class AuthController : ControllerBase
      {
          private readonly IValidator<TokensDTO>  tokensDTOValidator;
          private readonly IAuthService  googleAuthService;
          private readonly IRefreshTokenService  refreshTokenService;


          public AuthController(IAuthService authService,
              IRefreshTokenService refreshTokenService,
              IValidator<TokensDTO> tokensDTOValidator)
          {
               googleAuthService = authService;
               refreshTokenService = refreshTokenService;
               tokensDTOValidator = tokensDTOValidator;
          }

          [HttpPost("refresh")]
          public async Task<ActionResult<TokensDTO>> GetNewTokenFromRefreshToken([FromBody]
  TokensDTO tokensDTO)
          {
              try
              {
```

```
                var isValidGoogleData = await
 tokensDTOValidator.ValidateAsync(tokensDTO);

                if (!isValidGoogleData.IsValid)
                {
                    return BadRequest(isValidGoogleData.Errors);
                }

                var tokenDTOResponse = await
 refreshTokenService.RefreshAccessToken(tokensDTO);

                if (tokenDTOResponse == null)
                {
                    return BadRequest("Invalid token");
                }

                return Ok(tokenDTOResponse);
            }
            catch (Exception ex)
            {
                return StatusCode(500, "Internal server error");
            }
        }

        [HttpPost("register")]
        public async Task<ActionResult<TokensDTO>> Register([FromBody] RegistrationDTO
registerRequest)
        {
            try
            {
                var validator = new RegistrationDTOValidator();
                var validationResult = await validator.ValidateAsync(registerRequest);

                if (!validationResult.IsValid)
                {
                    return BadRequest(validationResult.Errors.Select(e =>
e.ErrorMessage));
                }

                var result = await  googleAuthService.RegisterAsync(registerRequest);

                if (result == null)
                {
                    return Conflict("User with this email already exists");
                }

                return Ok(result);
            }
            catch (Exception ex)
            {
                return StatusCode(500, "Internal server error");
            }
        }

        [HttpPost("authenticate")]
        public async Task<ActionResult<TokensDTO>> AuthenticateAsync([FromBody] LoginDTO
loginRequest)
        {
            try
            {
                var validator = new LoginDTOValidator();
                var validationResult = await validator.ValidateAsync(loginRequest);

                if (!validationResult.IsValid)
                {
                    return BadRequest(validationResult.Errors.Select(e =>
e.ErrorMessage));
                }

                var tokensDTO = await  googleAuthService.AuthenticateAsync(loginRequest);
```

```
                if (tokensDTO == null)
                {
                    return Unauthorized("Invalid email or password");
                }

                return Ok(tokensDTO);
            }
            catch (Exception ex)
            {
                return StatusCode(500, "Internal server error");
            }
        }
    }
}
```

ProductsController.cs:

```
using FluentValidation;
using Microsoft.AspNetCore.Mvc;
using Marcet Api.Models.Dto.Auth;
using Marcet Api.Services.IServices;
using Marcet Api.Services.IServices.ITokens;
using Marcet Api.Validators.Auth;
using Models.Dto.Token;
using Models.Dto.Auth;

namespace Marcet Api.Controllers
{
    [Route("api/auth")]
    [ApiController]
    public class AuthController : ControllerBase
    {
        private readonly IValidator<TokensDTO>  tokensDTOValidator;
        private readonly IAuthService  googleAuthService;
        private readonly IRefreshTokenService  refreshTokenService;


        public AuthController(IAuthService authService,
            IRefreshTokenService refreshTokenService,
            IValidator<TokensDTO> tokensDTOValidator)
        {
             googleAuthService = authService;
             refreshTokenService = refreshTokenService;
             tokensDTOValidator = tokensDTOValidator;
        }

        [HttpPost("refresh")]
        public async Task<ActionResult<TokensDTO>> GetNewTokenFromRefreshToken([FromBody]
TokensDTO tokensDTO)
        {
            try
            {
                var isValidGoogleData = await
 tokensDTOValidator.ValidateAsync(tokensDTO);

                if (!isValidGoogleData.IsValid)
                {
                    return BadRequest(isValidGoogleData.Errors);
                }

                var tokenDTOResponse = await
 refreshTokenService.RefreshAccessToken(tokensDTO);

                if (tokenDTOResponse == null)
                {
                    return BadRequest("Invalid token");
                }

                return Ok(tokenDTOResponse);
            }
            catch (Exception ex)

{
                 return StatusCode(500, "Internal server error");
             }
        }
```

```
        [HttpPost("register")]
        public async Task<ActionResult<TokensDTO>> Register([FromBody] RegistrationDTO
registerRequest)
        {
            try
            {
                var validator = new RegistrationDTOValidator();
                var validationResult = await validator.ValidateAsync(registerRequest);

                if (!validationResult.IsValid)
                {
                    return BadRequest(validationResult.Errors.Select(e =>
e.ErrorMessage));
                }

                var result = await  googleAuthService.RegisterAsync(registerRequest);

                if (result == null)
                {
                    return Conflict("User with this email already exists");
                }

                return Ok(result);
            }
            catch (Exception ex)
            {
                return StatusCode(500, "Internal server error");
            }
        }

        [HttpPost("authenticate")]
        public async Task<ActionResult<TokensDTO>> AuthenticateAsync([FromBody] LoginDTO
loginRequest)
        {
            try
            {
                var validator = new LoginDTOValidator();
                var validationResult = await validator.ValidateAsync(loginRequest);

                if (!validationResult.IsValid)
                {
                    return BadRequest(validationResult.Errors.Select(e =>
e.ErrorMessage));
                }

                var tokensDTO = await  googleAuthService.AuthenticateAsync(loginRequest);

                if (tokensDTO == null)
                {
                    return Unauthorized("Invalid email or password");
                }

                return Ok(tokensDTO);
            }
            catch (Exception ex)
            {
                return StatusCode(500, "Internal server error");
            }
        }
    }
}
```