МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: «**РОЗРОБКА ФРОНТЕНДУ ІНТЕРНЕТ-МАГАЗИНУ СПОРТИВНОГО ХАРЧУВАННЯ З ВИКОРИСТАННЯМ ФРЕЙМВОРКУ REACT**»

Виконав: студент 4 курсу, групи 6.1210-2пі
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
В.А. Ніценко
(ініціали та прізвище)

Керівник професор кафедри програмної інженерії, професор, д.т.н. Чопоров С.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент завідувач кафедри комп'ютерних наук, доцент, д.т.н. Шило Г.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Ніценку Вадиму Андрійовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка фронтенду інтернет-магазину спортивного харчування з використанням фреймворку React

керівник роботи Чопоров Сергій Вікторович, д.т.н., професор
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 21 » грудня 2023 року № 2180-с

2. Строк подання студентом роботи 03.06.2024 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Розробка вебзастосунку.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою докладу

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 25.12.2023 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	09.01.2024	
2.	Збір вихідних даних.	05.02.2024	
3.	Обробка методичних та теоретичних	04.03.2024	
	джерел.		
4.	Розробка першого та другого розділу.	29.04.2024	
5.	Розробка третього розділу.	20.05.2024	
6.	Оформлення та нормоконтроль	27.05.2024	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	20.06.2024	

Студент _____ В.А. Ніценко
(підпис) (ініціали та прізвище)

Керівник роботи _____ С.В. Чопоров
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка фронтенду інтернет-магазину спортивного харчування з використанням фреймворку React»: 43 с., 25 рис., 1 табл., 8 джерел.

ЕЛЕКТРОННА КОМЕРЦІЯ, ІНТЕРФЕЙС КОРИСТУВАЧА, КОМПОНЕНТИ, РОЗРОБКА, СПОРТИВНЕ ХАРЧУВАННЯ, ФРОНТЕНД, REACT.

Об’єкт дослідження – вебсайти магазинів спортивного харчування.

Мета роботи: розробка фронтенд частини сучасного вебсайту спортивного харчування та спортивних товарів.

Метод дослідження – методи ручного кодування вебсайтів.

У дипломному проєкті було досліджено вебсайти спортивного харчування та зроблені висновки та на основі цих висновків було розроблено фронтенд частину вебсайта за допомогою React, для того щоб сайт виглядав привабливо було застосовано TailwindCSS, для роутінгу було використано axios.

Серверна частина була взята с api яке було розроблене на Django та з’єднане с фронтенд частиною.

SUMMARY

Bachelor’s qualifying paper «Development of a Frontend of an Online Sports Nutrition Store Using the React Framework»: 43 pages, 25 figures, 1 table, 8 references.

E-COMMERCE, USER INTERFACE, COMPONENTS, DEVELOPMENT, SPORTS NUTRITION, FRONTEND, REACT.

Research Object: websites of sports nutrition stores.

Objective Development of the frontend part of a modern website for sports nutrition and sports goods.

Research Method: manual coding methods for websites.

In this thesis project, websites of sports nutrition stores were examined, conclusions were drawn, and based on these conclusions, the frontend part of a website was developed using React. To make the site look attractive, Tailwind CSS was applied, and Axios was used for routing.

The backend was developed with an API using Django and connected to the frontend part.

ЗМІСТ

ВСТУП

Чим далі рухається прогрес, тим менше можна зробити без вебдодатків. Таке вже всюди у багатьох сферах бізнесу, в тому числі в магазинах. На даний момент жоден магазин не може продуктивно існувати без свого вебсайту. У вебсайту занадто багато переваг, щоб не створити його, і занадто багато недоліків, якщо намагатися будувати будь-який клієнтоорієнтований бізнес без онлайн-додатка.

Сайт – це водночас комунікація з клієнтами, оскільки на сайті можна зробити зворотний зв'язок і коментарі, збирати статистику та таким чином враховувати побажання клієнтів. Сайт – це продуктивна робота 24/7, адже йому не потрібні вихідні, відпустки, і він не може захворіти, а якщо й захворіє, то команда розробників швидко його вилікує. Якщо говорити двома словами, сайт – це зручно та ефективно як для бізнесу, так і для клієнтів. Створення свого вебдодатка чудово масштабує бізнес і таким чином збільшує прибуток. Це насамперед зручність для клієнта, адже з зручним вебсайтом користувачі з більшою ймовірністю будуть повертатися на нього і купувати товари.

Метою моєї кваліфікаційної роботи було вивчити сайти спортивного харчування в Україні і створити своє конкурентоспроможне вебдодаток. Для цього були використані актуальні технології, такі як React. React на даний момент найпопулярніший фреймворк, на ньому створюється величезна кількість вебдодатків. За допомогою нього була створена більша частина логіки додатка. Мова програмування використовувалася TypeScript, він чудово взаємодіє з React і є більш зручним, ніж ванільний JavaScript, адже в ньому сувора типізація, і легше виявляти помилки. Для вигляду сайту використовувався TailwindCSS, дуже зручна і швидка технологія. З Tailwind не потрібно створювати новий файл, писати в ньому щось паралельно і вигадувати класи. Можна замість семантичного класу прямо прописати назву стилів.

Для досягнення мети були визначені наступні завдання:

– зручне та зрозуміле оформлення сайту;
– швидка робота вебдодатка;
– реалізація додатка за допомогою актуальних технологій;
– тестування додатка;
– проєктування додатка та його компонентів;
– читабельний і зрозумілий код.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд літературних джерел

В електронній комерції з кожним роком проводять все більше й більше досліджень, спрямованих на з’ясування приросту продажів та активних користувачів. Чим далі йде потяг розвитку, тим популярнішими стають онлайн-магазини. Це стало дуже популярною галуззю як для бізнесу, так і для клієнтів.

Для бізнесу це чудовий спосіб масштабуватися і вийти за рамки локального продажу та почати надавати свою продукцію за межами свого міста, країни і навіть континенту. Також для бізнесу це відмінний спосіб швидше розвинути свою справу і закріпитися на ринку, адже чим раніше почнеш, тим більше шансів. З кожним роком з’являється все більше й більше ecommerce-сайтів, і, як наслідок, зростає конкуренція.

Клієнти, у свою чергу, у виграші, коли конкуренти починають змагатися і боротися за клієнтську базу, оскільки зазвичай, щоб завоювати і відібрати клієнтів, підприємства можуть робити знижки на своїх сайтах, щоб залучити більше людей.

Для клієнтів ecommerce сайти – це чудова можливість закуповувати продукти, які, можливо, відсутні в магазинах поблизу, і набагато зручніше буде замовити онлайн, або, можливо, клієнт хоче спробувати якісь незвичайні продукти, які відсутні в місці релокації людини.

Огляд літературних джерел показує, що ця тема стала центральною у багатьох дослідженнях.Частка електронної комерції на ринку України значно розширилась протягом останніх років. Згідно з відомостями Інтернет асоціації України, обсяг електронної комерції в Україні досяг вражаючих $5,2 млрд у 2020 році, що порівняно з $3,8 млрд у 2019 році. Згідно з звітом, кількість онлайн-покупців в Україні збільшилася на 16% у 2020 році, а середня вартість

замовлення підвищилася на 10%. Розвиток електронної комерції в Україні підтримували кілька ключових факторів, таких як зростання доступу до Інтернету, поширення мобільних пристроїв та зміна споживчої поведінки. За даними Міністерства цифрової трансформації України, рівень використання Інтернету в Україні збільшився до 56,4% у 2020 році, що є вражаючим зростанням порівняно з 44,1% у 2015 році. За останні п'ять років електронна комерція в Україні показує стабільний ріст на рівні 20-25% щорічно. Цей позитивний тренд визначається збільшенням користування смартфонами та Інтернетом, а також зручністю онлайн-шопінгу. Не останню роль у цьому відіграла і пандемія COVID-19, яка прискорила перехід споживачів до онлайн-покупок для уникнення ризиків для здоров'я в фізичних магазинах [1].

Крім того, дослідження «Персоналізація як стратегія лояльності в електронній комерції» підкреслює важливість персоналізованих рекомендацій та пропозицій для утримання клієнтів Важливим є розгляд сучасних підходів та наукових праць, що досліджують вплив персоналізації на задоволеність та лояльність споживачів. Одним із важливих аспектів є гіперперсоналізація, яка може значно підвищити лояльність та задоволеність клієнтів у системах управління взаєминами з клієнтами (CRM). Дослідження, проведене Rane, Choudhary та Rane (2023), вказує на те, що застосування гіпер-персоналізації може бути ефективним стратегічним інструментом для підвищення лояльності та задоволеності клієнтів. Ologunebi (2023) у своїй праці дає аналіз стратегій утримання клієнтів у сегменті електронної комерції, що може допомогти в розумінні та вдосконаленні стратегій лояльності, зокрема через персоналізацію. Робота Xuan Nguyen, Nguyen, Suseno та Bui Quang (2020) зосереджена на розгляді взаємодії сприйняття цінності та цифрових технологій, що може бути важливим для розуміння і вдосконалення персоналізованих стратегій у B2B-сегменті. Mushtaq, Ahmad, Rextin та Malik (2020) у своєму дослідженні вказують на важливість покращення користувацького досвіду через зрозумілі та легкі у прийнятті рішень інтерфейси, що є суттєвим аспектом у контексті електронної комерції й

персоналізації [2]. Приклад зростання продажів у США можна побачити нижче на діаграмі (див. рис. 1.1).

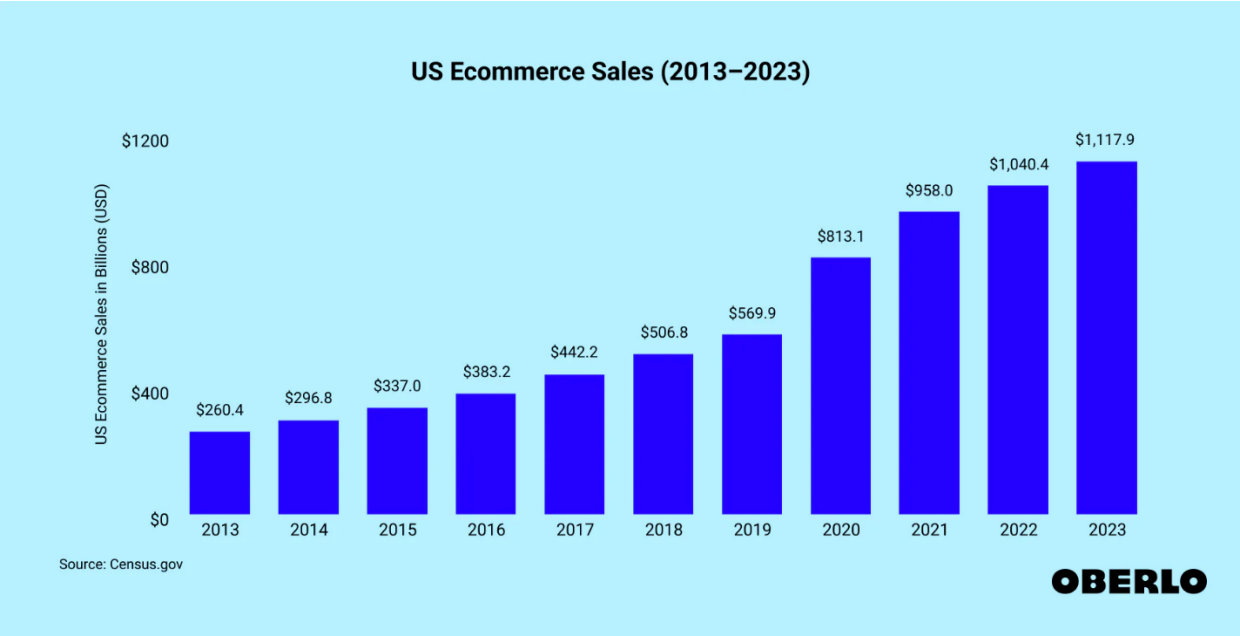

Рисунок 1.1 – Зростання продажів екомерс магазинів у США [3]

Виходячи з усіх вищезазначених фактів, можна зробити висновок, що електронна комерція – це не лише зручність для клієнта та можливість бізнесу масштабуватися, а й складна екосистема, яка з кожним роком підлаштовується та еволюціонує під впливом технологій, вимог клієнтів та глобальних викликів.

1.2 Аналіз програмних продуктів-аналогів

Сайти спортивного харчування – це вузькоспрямовані сайти для клієнтів, які пов’язали своє життя зі спортом, або людей, які хочуть підтримувати здорове тіло. Ось кілька прикладів досить популярних сайтів.

GymBeam – один із найвідоміших сайтів для спорттоварів. Що є у цього сайту такого, чого зазвичай немає в інших: у нього досить великий функціонал сайту та самих товарів, там є не лише спортивні продукти, а й спортивний одяг.

Плюси:

– різноманітність товарів;
– є можливість безкоштовної доставки за дотримання певних умов;
– зрозуміла навігація по сайту;
– є блог і допоміжні статті;
– можливість створення власного комплексу харчування.

Ось як виглядає головна сторінка сайту, наведеного вище (див. рис. 1.2).

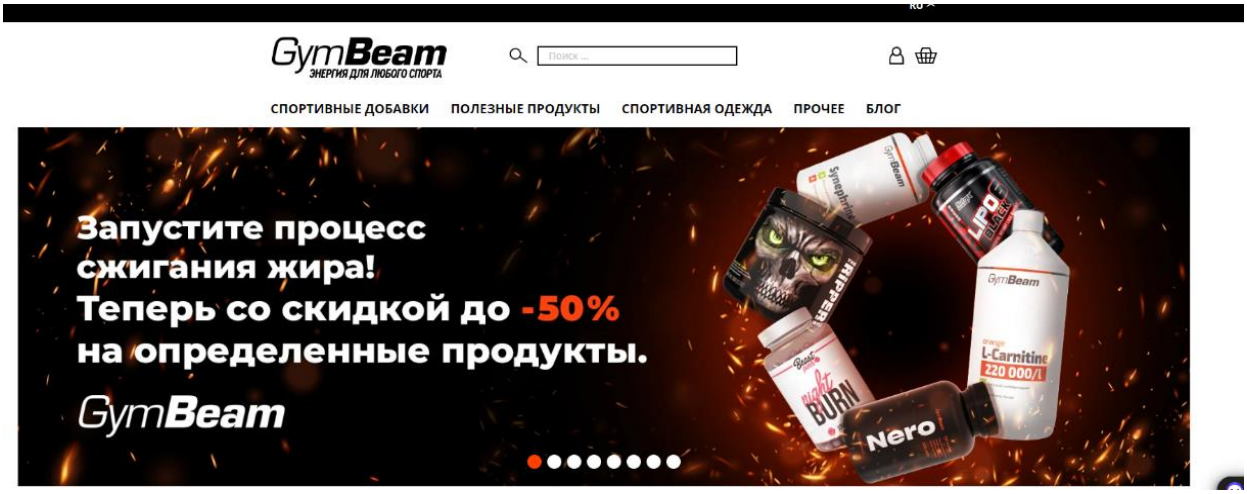

Рисунок 1.2 – Вигляд головної сторінки сайту GymBeam

Bcaa.ua також досить популярний сайт, він уже більш вузькоспеціалізований на спортивному харчуванні і немає якихось додаткових товарів, що покривають ще якісь вимоги клієнтів.

Ось як виглядає головна сторінка сайту, наведеного вище (див. рис. 1.3).

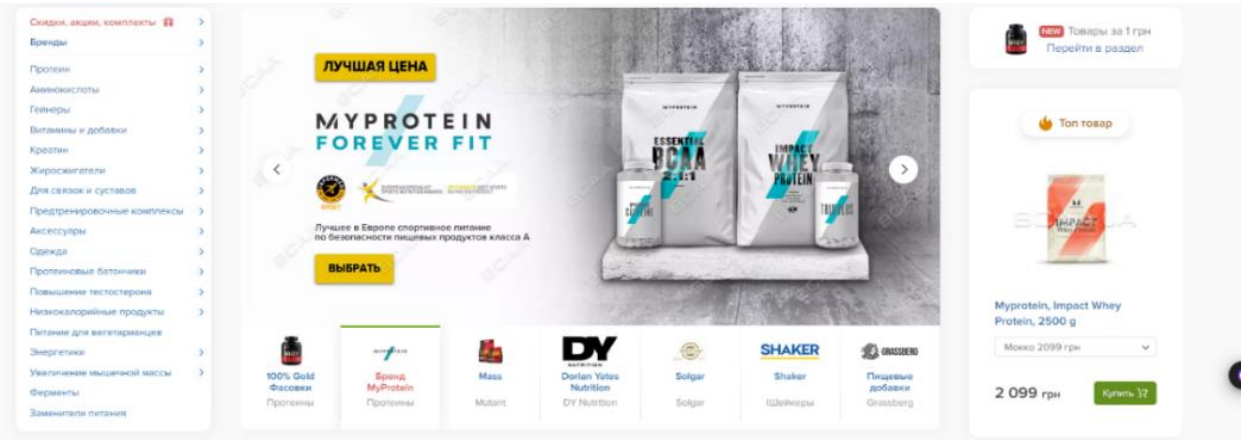

Рисунок 1.3 – Вигляд головної сторінки сайту Bcaa.ua

Плюси:

– можливість відразу зателефонувати менеджеру;
– доступні ціни;
– сайт дуже привабливо виглядає.

Ще один сайт, який не відстає за популярністю від попередніх двох – waysport. У нього не настільки привабливе оформлення, але для деяких це не найважливіший пункт. У сайту є плюси, що перекривають недолік оформлення.

Плюси:

– часті знижки на товари;
– дуже велике різноманіття продукції;
– чуйний сервіс підтримки.

Ось як виглядає головна сторінка сайту, наведеного вище (див. рис. 1.4).

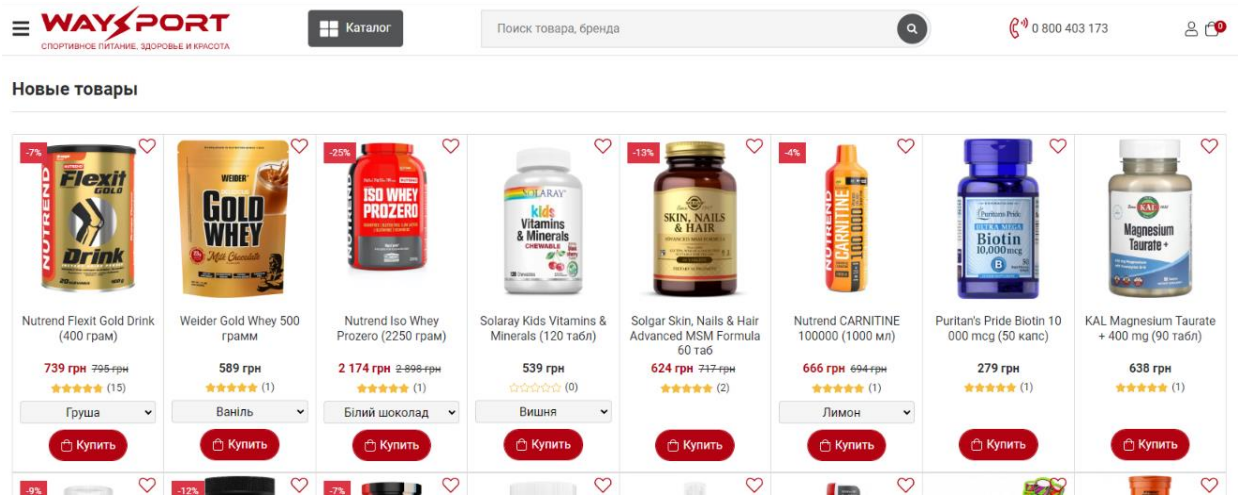

Рисунок 1.4 – Вигляд головної сторінки сайту Waysport

Також є сайти найпопулярніших виробників спортивної продукції, але зазвичай на таких сайтах цінник може бути досить високим, і не у кожної людини буде можливість напряму співпрацювати з брендом. Наприклад, такий бренд як Ultimate Nutrition: у нього відмінне оформлення сайту, швидка доставка і багато інших плюсів, але цінник точно не входить у ці плюси. Інтерфейс сайту Ultimate Nutrition наведено на рисунку (див. рис. 1.5).

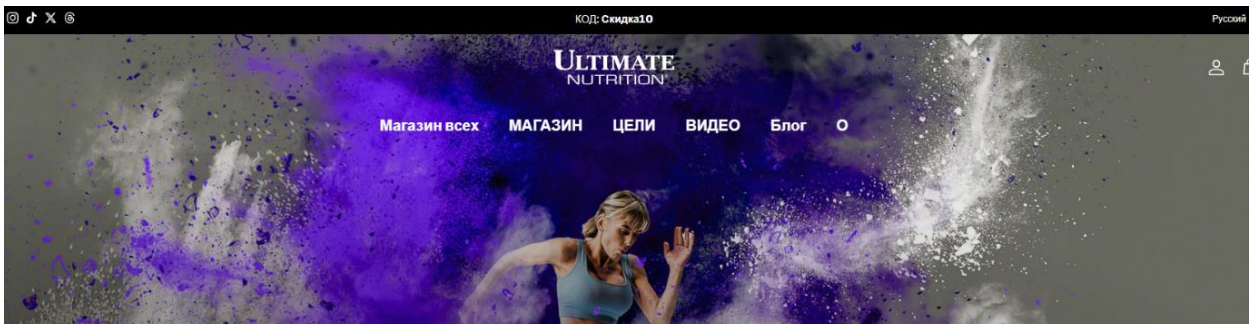

Рисунок 1.5 – Вигляд головної сторінки сайту Ultimate Nutrition

1.3 Постановка завдання

Мета кваліфікаційної роботи – створити фронтенд частину вебдодатку інтернет-магазину спортивного харчування з використанням фреймворку React. Для досягнення цієї мети було необхідно поставити перед собою завдання і в процесі розробки орієнтуватися на ці завдання:

- проаналізувати актуальні та ефективні технології, за допомогою яких створюються додатки типу онлайн-магазинів більш результативно (визначити переваги використання React та TypeScript у порівнянні з іншими фреймворками та мовами програмування);
- проаналізувати необхідний функціонал сайту та сайтів конкурентів, щоб виділити найважливіші аспекти, на яких варто зробити акцент (провести аналіз користувацького досвіду (UX) та інтерфейсу користувача (UI) для створення зручного та привабливого дизайну);
- написати моделі даних, які приймаються з API бекенду для різних задач, таких як реєстрація, авторизація та завантаження продуктів (забезпечити надійну валідацію та обробку даних на фронтенді);
- використати переваги мови TypeScript для зменшення кількості помилок за допомогою статичної типізації (застосувати інтерфейси та типи для покращення читабельності та підтримуваності коду);
- використати можливості фреймворку React, а саме хуки (useState, useEffect, useContext, тощо) для управління станом компонентів та

життєвим циклом додатку;

– написати інтерфейси для компонентів, щоб забезпечити їхню повторне використання та гнучкість;
– впровадити систему маршрутизації (наприклад, react-router) для навігації між різними сторінками додатку без перезавантаження.

Використання React дозволяє досить швидко розробляти вебдодатки. На ньому часто створюють онлайн-магазини завдяки його гнучкості, модульності та потужній екосистемі. React надає розробникам можливість створювати повторно використовувані компоненти, що значно прискорює процес розробки та полегшує підтримку коду. Крім того, віртуальний DOM (Document Object Model) React забезпечує високу продуктивність, оптимізуючи оновлення інтерфейсу користувача.

У поєднанні з TypeScript, React стає ще потужнішим інструментом. TypeScript додає статичну типізацію, яка допомагає виявляти помилки на етапі компіляції, що особливо важливо у великих проєктах, таких як інтернет-магазини. Це зменшує кількість помилок у продакшені та полегшує рефакторинг коду.

1.4 Опис інструментарію

У процесі розробки фронтенду інтернет-магазину спортивного харчування було використано сучасний та ефективний інструментарій, що дозволив створити високоякісний, масштабований та продуктивний вебдодаток. Обрані технології та бібліотеки є лідерами у своїх категоріях та широко застосовуються у професійній розробці вебдодатків, опис яких наведено нижче.

React – бібліотека JavaScript для побудови користувацьких інтерфейсів, розроблена компанією Facebook. React використовує компонентний підхід, що дозволяє створювати повторно використовувані UI компоненти. Віртуальний

DOM React оптимізує оновлення інтерфейсу, забезпечуючи високу продуктивність. React також надає потужні хуки для управління станом та життєвим циклом компонентів. У таблиці 1.1 наведено порівняння самих популярних фреймворків.

TypeScript – надбудова над JavaScript, розроблена Microsoft, яка додає статичну типізацію. TypeScript дозволяє виявляти потенційні помилки на етапі компіляції, що значно підвищує якість коду та полегшує його підтримку. Використання TypeScript особливо корисне в масштабних проєктах, таких як інтернет-магазини, де надійність та безпека є критичними.

TailwindCSS – утилітарний CSS фреймворк, який надає низькорівневі класи для швидкої та гнучкої побудови дизайну. На відміну від традиційних CSS фреймворків, Tailwind дозволяє створювати унікальні дизайни без необхідності перевизначати стилі. Це прискорює розробку, зменшує об’єм CSS і забезпечує адаптивність дизайну [4].

Axios – популярна JavaScript бібліотека для виконання HTTP-запитів. Axios спрощує взаємодію з серверним API, надаючи простий та інтуїтивний інтерфейс. Він підтримує перехоплення запитів та відповідей, трансформацію даних та обробку помилок, що робить його незамінним для інтеграції фронтенду з бекендом.

React Router – стандартна бібліотека маршрутизації для React. React Router дозволяє створювати односторінкові додатки (SPA) з динамічною навігацією без перезавантаження сторінки. Це покращує користувацький досвід, прискорює взаємодію з сайтом та зменшує навантаження на сервер.

Електронна комерція стрімко розвивається, демонструючи стабільний ріст продажів та кількості активних користувачів, що зумовлено зростанням доступу до Інтернету, поширенням мобільних пристроїв та зміною споживчої поведінки. Для створення успішного інтернет-магазину важливо використовувати сучасні технології, такі як React та TypeScript, які дозволяють розробити гнучкий, масштабований та продуктивний вебдодаток з урахуванням потреб користувачів та тенденцій ринку.

Таблиця 1.1 – Порівняння фреймворків

Критерій	**React**	**Angular**	**Vue.js**
Переваги	Гнучкість та модульність компонентів віртуальний DOM для високої продуктивності велика екосистема та спільнота; підтримка Facebook; легкий для вивчення; відмінна інтеграція з TypeScript.	Повний фреймворк (MVC); двостороннє зв’язування даних; Dependency injection; TypeScript за замовчуванням; підтримка Google; хороша для великих корпоративних проєктів.	Прогресивний фреймворк (можна додавати функціонал поступово); простий синтаксис, легкий для вивчення; реактивна система оновлення даних; Менший розмір бібліотеки; добре документований.
Недоліки	Тільки View шар (потрібні додаткові бібліотеки для повного MVC) часті оновлення можуть ускладнити підтримку JSX може бути незвичним спочатку.	Крива навчання великий розмір фреймворку; може бути повільним для простих додатків; складна міграція між версіями.	Менша екосистема порівняно з React і Angular; ризик надмірного використання директив; менша спільнота та ресурси.

Продовження таблиці 1.1

Критерій	**React**	**Angular**	**Vue.js**
Продуктивність	Висока (завдяки віртуальному DOM)	Середня (покращилась у останніх версіях)	Висока (реактивна система та віртуальний DOM)
Масштабованість	Висока	Висока	Висока для малих та середніх проєктів
Спільнота та екосистема	Дуже велика	Велика	Зростаюча
Типова архітектура	Компонентна	MVC	Компонентна

2 ДОСЛІДЖЕННЯ ПРОГРАМНИХ ЗАСОБІВ РЕАЛІЗАЦІЇ

2.1 Проєктування системи вебдодатку

Для демонстрації основних можливостей вебдодатку інтернет-магазину спортивного харчування та показу різноманітних сценаріїв взаємодії користувача з системою, були розроблені наступні прецеденти (use cases) які можно побачити на рисунку 2.1.

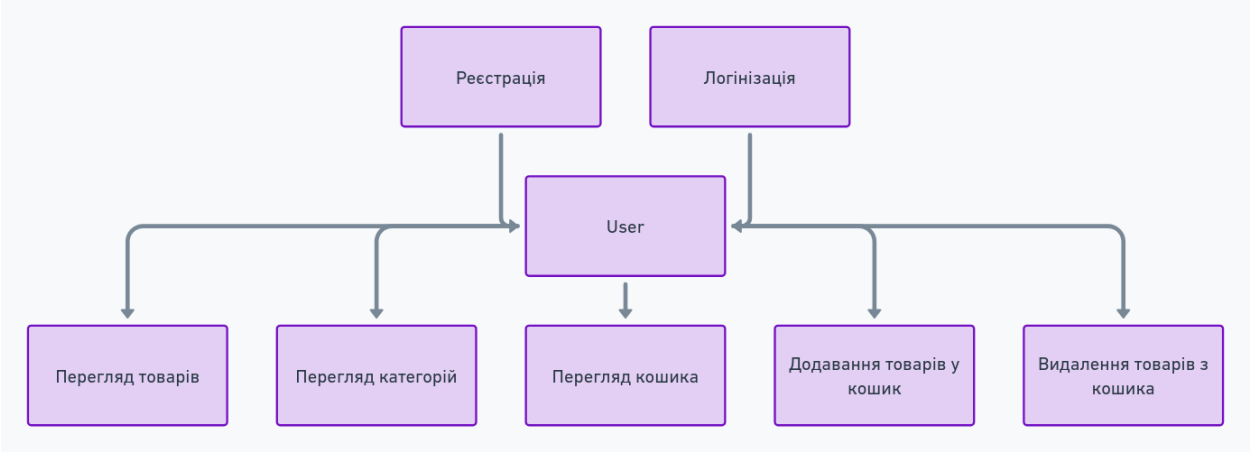

Рисунок 2.1 – Діаграма різноманітних сценаріїв(use cases)

Діаграма ілюструє такі прецеденти як:

– реєстрація: дозволяє новому користувачеві зареєструватися та створити обліковий запис у системі (це включає введення особистих даних, таких як ім’я, електронна пошта та пароль; після успішної реєстрації користувач отримує доступ до повного функціоналу системи);
– логін: прецедент для авторизації зареєстрованих користувачів шляхом введення облікових даних (електронна пошта та пароль) (після успішного входу користувач може отримати доступ до персональних налаштувань та історії замовлень);
– перегляд товарів: дозволяє користувачеві переглядати каталог

доступних товарів та їх опис (користувач може сортувати товари за різними критеріями, такими як ціна, рейтинг та новизна);

– перегляд категорій товарів: користувач може вибирати та переглядати товари за певними категоріями (це допомагає швидко знаходити потрібні товари в залежності від їх типу чи призначення);
– додавання товарів у кошик: прецедент, який дозволяє користувачеві додавати вибрані товари до кошика (користувач може бачити кількість та загальну вартість товарів у кошику);
– видалення товарів із кошика: користувач може видаляти товари з кошика, якщо він передумав купувати їх або хоче змінити кількість замовлених одиниць;
– перегляд кошика: дозволяє користувачеві переглядати всі товари, додані до кошика.

Таким чином, діаграма прецедентів (use case diagram) охоплює найосновніші сценарії використання вебдодатку інтернет-магазину спортивного харчування для різних типів користувачів.

Сутність Category:

– id: унікальний ідентифікатор категорії, що зберігається у числовому полі;
– name: назва категорії, що зберігається у рядковому полі;
– slug: унікальна частина URL-адреси, що ідентифікує категорію, зберігається у рядковому полі.

Сутність Product:

– id: унікальний ідентифікатор продукту, що зберігається у числовому полі;
– name: назва продукту, що зберігається у рядковому полі;
– description: детальний опис продукту, що зберігається у рядковому полі;
– image: шлях до зображення продукту, що зберігається у рядковому полі;

- price: ціна продукту, що зберігається у числовому полі;
- category: об’єкт типу Category, що містить інформацію про категорію, до якої належить продукт;
- short_description: короткий опис продукту, що зберігається у рядковому полі;
- imageUrl?: необов’язковий рядок, що може містити альтернативний URL-адрес зображення продукту.

Ці інтерфейси визначають структуру даних про категорії та продукти, що надходять з API бекенду. Використання TypeScript забезпечує типізацію цих даних у фронтенд-частині додатку, допомагаючи виявляти потенційні помилки та підвищуючи надійність обробки інформації.

Сутність CartItem:

- id: унікальний ідентифікатор елемента кошика, що зберігається у числовому полі;
- product: об’єкт, що містить інформацію про продукт у кошику;
- id: унікальний ідентифікатор продукту, що зберігається у числовому полі;
- name: назва продукту, що зберігається у рядковому полі;
- price: ціна одиниці продукту, що зберігається у числовому полі;
- image: шлях до зображення продукту, що зберігається у рядковому полі;
- quantity: кількість одиниць даного продукту у кошику, що зберігається у числовому полі;
- price_sum: загальна вартість даної позиції у кошику (ціна * кількість), що зберігається у числовому полі.

Цей інтерфейс CartItem визначає структуру даних про елемент кошика покупок у інтернет-магазині спортивного харчування. Він містить інформацію про сам продукт (назва, ціна, зображення), кількість одиниць продукту у кошику та підсумкову вартість цієї позиції.

Діаграма ілюструє прецеденти використання для адміністратора вебдодатку магазину спортивного харчування (див. рис. 2.2).

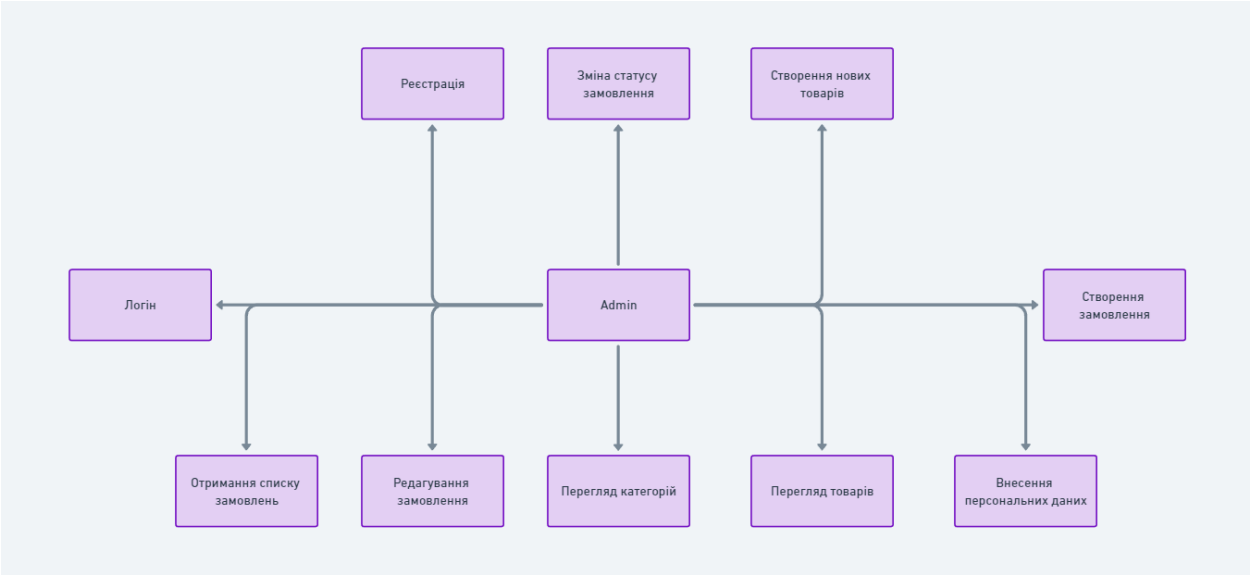

Рисунок 2.2 – Діаграма різноманітних сценаріїв для адміністратратора

Ось опис цих прецедентів:

– ***зміна статусу замовлення***: адміністратор має можливість змінювати поточний статус замовлення на новий;
– ***створення нових товарів***: адміністратор може додавати нові товари до бази даних магазину;
– ***створення замовлення***: адміністратор може створювати нове замовлення в системі;
– ***внесення персональних даних***: адміністратор має доступ до внесення та редагування персональних даних користувачів;
– ***перегляд товарів***: адміністратор може переглядати список доступних товарів у магазині;
– ***редагування замовлення***: адміністратор може вносити зміни до існуючого замовлення;
– ***отримання списку замовлень***: адміністратор має доступ до перегляду всіх замовлень у системі;
– ***логін***: адміністратор може увійти в систему зі своїми обліковими даними;
– ***реєстрація***: адміністратор може зареєструвати новий обліковий запис у системі;

– ***перегляд категорій***: адміністратор має можливість переглядати список категорій товарів.

Візуальне представлення структури бази даних усіх полів і таблицьз API написаного з допомогою Django які використовуються для збереження інформації можна побачити на рисунку (див. рис. 2.3).

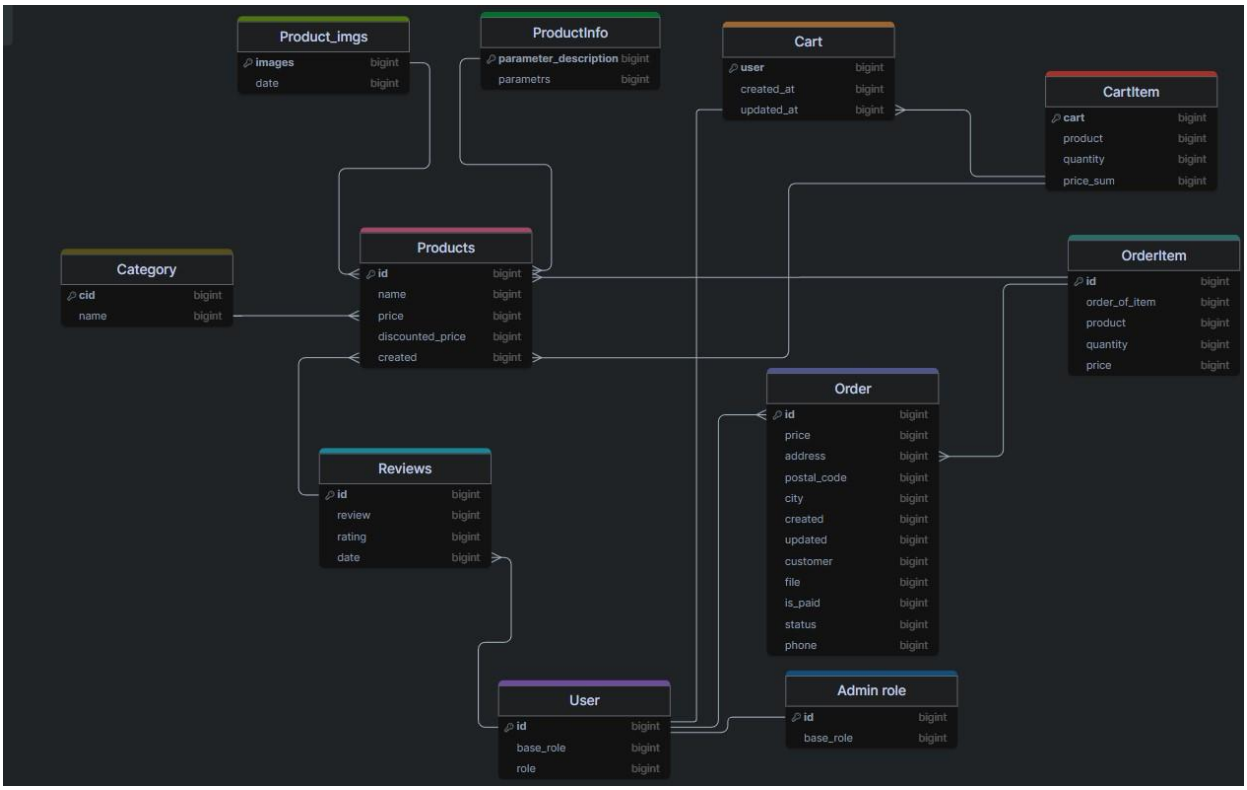

Рисунок 2.3 – Візуальне представлення схеми бази даних

Кожна таблиця має унікальний первинний ключ, який ідентифікує кожен запис у таблиці. Основні атрибути, які використовуються на сайті з бази даних, можна описати наступним чином.

Сутність Product:

– category: зовнішній ключ (ForeignKey) на модель Category, що вказує на категорію, до якої належить продукт;

– name: назва продукту, що зберігається у текстовому полі CharField з максимальною довжиною 150 символів;

– image: зображення продукту, що зберігається у полі ImageField з шляхом завантаження 'products/';

– price: ціна продукту, що зберігається у полі DecimalField з максимальною кількістю цифр 10 та кількістю десяткових знаків 2;
– short_description: короткий опис продукту, що зберігається у текстовому полі TextField з максимальною довжиною 200 символів;
– description: більш детальний опис продукту, що зберігається у полі RichTextField;
– old_price: стара ціна продукту, що зберігається у полі DecimalField з максимальною кількістю цифр 10 та кількістю десяткових знаків 2;
– created: дата та час створення продукту, що зберігається у полі DateTimeField;

Сутність ProductImages:

– images: зображення продукту, що зберігається у полі ImageField з шляхом завантаження 'product-images';
– product: зовнішній ключ (ForeignKey) на модель Product, що пов’язує зображення з певним продуктом;
– date: дата та час завантаження зображення, що зберігається у полі DateTimeField.

Сутність Cart:

– user: зовнішній ключ (ForeignKey) на модель User, що пов’язує кошик з певним користувачем;
– created_at: дата та час створення кошика, що зберігається у полі DateTimeField;
– updated_at: дата та час останнього оновлення кошика, що зберігається у полі DateTimeField.

Сутність CartItem:

– cart: зовнішній ключ (ForeignKey) на модель Cart, що пов’язує елемент кошика з певним кошиком;
– product: зовнішній ключ (ForeignKey) на модель Product, що вказує на продукт, який додано до кошика;
– quantity: кількість одиниць продукту в кошику, що зберігається у

полі PositiveIntegerField;

– price_sum: загальна вартість одиниць продукту в кошику, що зберігається у полі PositiveIntegerField.

2.2 Постачальник модулів Webpack

При розробці додатків на TypeScript та Angular, додаток складається з численних модулів. Веббраузери не здатні безпосередньо інтерпретувати ці модулі. Для вирішення цієї проблеми необхідно або об’єднати весь JavaScript-код в один файл і імпортувати його, або вручну підключити всі файли до сторінки за допомогою тегів script.

Для вирішення цього питання був обраний бандлер модулів Webpack. Він поєднує різні модулі та їхні залежності в правильному порядку в один файл. Крім того, Webpack перетворює і комбінує код у формат, який можуть зрозуміти веббраузери (див. рис. 2.4).

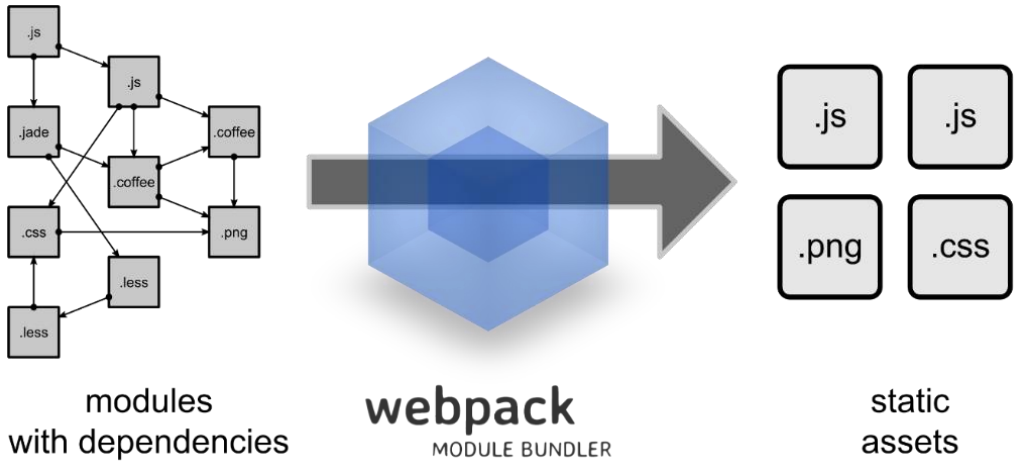

Рисунок 2.4 – Робота Webpack [5]

Таким чином, замість численних роз’єднаних модулів, Webpack генерує один або декілька файлів, готових для завантаження та виконання у браузері. Це спрощує процес розробки та розгортання додатків, забезпечуючи сумісність з різними браузерами та оптимізацію продуктивності.

Webpack також здатний виконувати численні інші операції:

- допомагає об’єднати всі ваші ресурси (CSS, JavaScript, зображення тощо) в єдиний пакет;
- відстежує зміни у файлах проєкту та автоматично перезбирає пакет при потребі;
- може виконувати транспіляцію JavaScript нових версій до старішого стандарту (ES5) за допомогою Babel, що дозволяє використовувати новітні функції JavaScript без турбот про сумісність з браузерами;
- може конвертувати вбудовані зображення у формат data:URI для зменшення кількості HTTP-запитів;
- дозволяє імпортувати CSS-файли за допомогою require() та об’єднувати їх з JavaScript-кодом;
- може запускати webpack-dev-server (вбудований локальний сервер з функцією живого оновлення браузера);
- підтримує Hot Module Replacement (гарячу заміну модулів без перезавантаження сторінки);
- може розділити вихідний файл на декілька окремих файлів (code splitting), щоб уникнути повільного завантаження сторінки через великий розмір JavaScript;
- може виконувати Tree Shaking (видалення невикористаного коду) для оптимізації розміру пакету;
- підтримує різноманітні плагіни та завантажувачі (loaders) для розширення функціональності та обробки додаткових типів файлів (TypeScript, SCSS, PostCSS та ін.) [6].

Крім того, Webpack забезпечує гнучку конфігурацію та можливість створення користувацьких плагінів, що дозволяє налаштувати процес збірки під конкретні потреби проєкту. Це робить його потужним інструментом для створення сучасних вебдодатків з великою кількістю залежностей та ресурсів різних типів.

Використання Webpack значно спрощує процес розробки, оптимізації та

розгортання вебдодатків, забезпечуючи ефективне керування модулями, ресурсами та продуктивністю. Завдяки своїй гнучкості та розширюваності Webpack став одним з найпопулярніших та найважливіших інструментів у сучасній веброзробці.

У процесі проєктування вебдодатку інтернет-магазину спортивного харчування були розроблені діаграми прецедентів для користувачів та адміністраторів, що відображають основні сценарії взаємодії з системою. Структура бази даних та її візуальне представлення демонструють зв’язки між сутностями та їх атрибути.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ПРАКТИЧНОГО ЗАВДАННЯ

3.1 Структурна реалізація фронтенд частини

Структура проєкту відіграє важливу роль у забезпеченні зрозумілості, підтримуваності та масштабованості коду. Логічна організація файлів та компонентів допомагає полегшити розробку, тестування та подальше супроводження додатку. Нижче наведено зображення зрозумілу, ефективну та популярну структуру папок, яка використовувалася для даного проєкту (див. рис. 3.1).

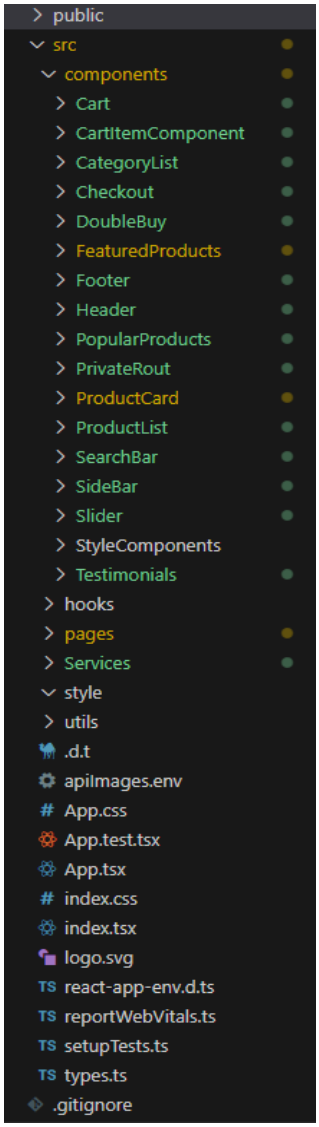

Рисунок 3.1 – Структура проєкту

Проєкт організовано за принципом розподілу компонентів, служб, утиліт та окремих сторінок на різні каталоги. Такий підхід дозволяє краще розділити відповідальності та забезпечити модульність та повторне використання коду.

Каталог components містить компоненти React, які використовуються в різних частинах додатку. Наприклад, ProductCard є компонентом, що відображає карточку продукту, а CartComponent – компонентом для відображення кошика покупок.

Каталог hooks призначений для зберігання хуків React, які інкапсулюють повторно використовувану логіку керування станом або взаємодії з API. Наприклад, useCart – це хук, що реалізує логіку роботи з кошиком покупок.

Каталог utils містить допоміжні функції та утиліти, які можуть використовуватися в різних частинах додатку. Наприклад, api.ts містить функції для взаємодії з серверною частиною через API, а cart.ts – утиліти для роботи з кошиком покупок.

Каталог pages містить компоненти React, які представляють окремі сторінки додатку, такі як HomePage, ProductListPage або CheckoutPage.

Каталог services у цьому каталозі знаходяться файли, відповідальні за обробку викликів API та інших сервісів. Сервіси відокремлюють деталі реалізації взаємодії з зовнішніми ресурсами, забезпечуючи розподіл обов’язків та підтримуваність коду.

Структура проєкту відображає організацію коду та файлів, і чиста, проста та пристойна структура проєкту демонструє чистий написаний код, що допомагає відлагоджувати код, а інший розробник може легко його прочитати. Крім того, під час розгортання коду на сервері він може легко розпізнавати файли, ось чому розробники впроваджують чисту, просту та пристойну структуру проєктів.

Кращою практикою є створення структури проєкту для React-додатку, розподіл файлів відповідно до їх роботи та розміщення їх у правильних каталогах. Наприклад, окремі компоненти, які можна використовувати в

декількох місцях, повинні знаходитися в каталозі компонентів, а також стандартизувати конвенції іменування додатку, щоб полегшити перевірку логіки та наявності будь-якого окремого файлу чи каталогу [7].

Створення добре організованої структури каталогів є важливим для підтримки проєкту React. Це підвищує читабельність, підтримуваність та масштабованість додатку. Дотримуючись належної структури, буде легше керувати кодом.

3.2 Приклади реалізації компонентів та сторінок

Однією з ключових особливостей розробленого вебдодатку інтернет-магазину спортивного харчування є відображення асортименту продуктів для перегляду та покупки користувачами. Ця функціональність реалізована за допомогою компонентів React, які забезпечують відображення інформації про продукт, а саме карточка продукта, компонент має назву ProductCard реалізациію даного компоненту можно подивитися на рисунку нижче (див. рис. 3.2).

```
  <div className="bg-white rounded-lg shadow-md overflow-hidden transition-transform duration-300 transform hover:-
translate-y-2 relative">
      <img
        src={imageURL}
        alt={product.name}
        className="w-full h-48 object-cover"
      />
      <div
        className="absolute top-2 right-2 bg-gray-600 text-white p-2 rounded-full cursor-pointer hover:bg-blue-700"
        onClick={openQuickView}
      >
        &#128065;
      </div>
      <div className="p-4">
        <h3 className="text-lg font-semibold mb-2">{product.name}</h3>
        <p className="text-gray-600 mb-4">${product.price}</p>
        <button
        className="bg-blue-500 hover:bg-blue-600 text-white py-2 px-4 rounded"
        onClick={handleAddToCart}
        >
          В корзину
        </button>
      </div>
    </div>
```

Рисунок 3.2 – Компонент ProductCard для картки товара

Дані про картки товарів отримуються з API, зокрема фотографія, назва, опис та ціна (див. рис. 3.3). Запит на API здійснюється за допомогою функції, яка має назву fetchProducts. Ця функція відповідає за отримання даних про продукти з серверної частини додатку.

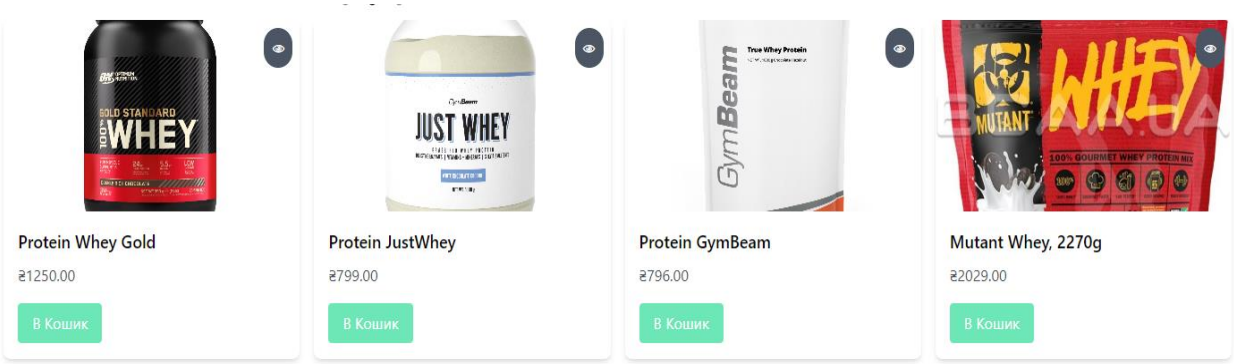

Рисунок 3.3 – Вигляд карточек товарів

Вебдодаток взаємодіє з API, написаним на Django, за допомогою файлу api.ts (див. рис. 3.4). Цей файл містить утиліти для взаємодії з серверною частиною через HTTP-запити, такі як отримання списку продуктів, створення замовлень тощо.

```
export const fetchProducts = async (): Promise<Product[]> => {
  const response = await api.get('api/products/');
  return response.data;
};
```

Рисунок 3.4 – HTTP запит на api

У цьому прикладі функція fetchProducts робить GET-запит до /api/products/ за допомогою axios та повертає масив об’єктів Product, які містять дані про продукти, отримані від API.

Компоненти, такі як ProductList або ProductCard, можуть імпортувати та використовувати функцію fetchProducts для отримання даних про продукти з API.

Компонент SearchBar використовує хук useEffect для завантаження даних про продукти з API за допомогою функції fetchProducts (див. рис. 3.5). Завантажені дані зберігаються в стані products, а також копіюються в стан filteredProducts для початкового відображення всіх продуктів.

```
useEffect(() => {
  const loadProducts = async () => {
    try {
      const products = await fetchProducts();
      setProducts(products);
    } catch (error) {
      console.error('Failed to load products:', error);
    }
  };
  loadProducts();
}, []);
```

Рисунок 3.5 – Релізація пошуку товарів

При введенні тексту в поле пошуку викликається функція handleSearch, яка оновлює стан searchQuery. Потім вона фільтрує масив products за введеним запитом та оновлює стан filteredProducts (див. рис. 3.6) відфільтрованим масивом.

У розмітці компонента відображається поле введення та список елементів, що відповідають відфільтрованим продуктам (filteredProducts).

```
export const filterProducts = (products: Product[], searchQuery: string): Product[] => {
 return products.filter(product =>
   product.name.toLowerCase().includes(searchQuery.toLowerCase())
 );
};
```

Рисунок 3.6 – Релізація фльтрації товарів

Таким чином, компонент SearchBar використовує функцію fetchProductsFromAPI для отримання даних про продукти з API, а потім застосовує фільтрацію на основі введеного користувачем пошукового запиту. Це дозволяє користувачеві легко знаходити потрібні продукти у списку.

Після того, як користувач шукає товар і вибирає потрібний, його переадресовують на сторінку SearchResultPage, яка відображає вибраний товар.

На цій сторінці використовується хук useSearchParams з бібліотеки react-router-dom для отримання параметрів пошуку з URL. Пошуковий запит зберігається у змінній searchQuery.

Компонент використовує хук useEffect (див. рис. 3.7) для завантаження всіх продуктів з API за допомогою функції fetchProducts та фільтрації цих продуктів відповідно до пошукового запиту за допомогою функції filterProducts. Відфільтровані продукти зберігаються у стані filteredProducts.

Якщо завантаження продуктів ще не завершено, відображається повідомлення «Завантаження...». Якщо під час завантаження сталася помилка, відображається повідомлення про помилку.

Для нових користувачів деякі функції недоступні, а саме функція додавання товарів до кошика. Щоб виправити це, потрібно перейти на сторінку реєстрації. Функціональність реєстрації нових користувачів реалізована в компоненті RegistrationPage.

```
useEffect(() => {
   const fetchData = async () => {
     try {
       setLoading(true);
       const data = await fetchProducts();
       setProducts(data);
       const filtered = filterProducts(data, searchQuery);
       setFilteredProducts(filtered);
       setError(null);
     } catch (err) {
       setError(err as Error);
     } finally {
       setLoading(false);
     }
   };
   fetchData();
  }, [searchQuery]);
```

Рисунок 3.7 – Завантажує та фільтрує дані продуктів

Цей компонент відповідає за відображення форми реєстрації та обробку введених даних користувача.

Компонент RegistrationPage використовує хук useState для керування станом форми реєстрації. Стан formData зберігає введені користувачем дані, такі як ім’я користувача, електронну пошту та паролі (див. рис. 3.8).

Функція handleChange оновлює стан formData при зміні значень полів форми. Функція handleSubmit викликається при відправленні форми реєстрації та виконує наступні дії:

– перевіряє, чи збігаються введені паролі;
– відправляє дані реєстрації на сервер за допомогою функції apiRequest з Services/api;
– якщо реєстрація пройшла успішно, зберігає отримані токени доступу та оновлення в localStorage і перенаправляє користувача на сторінку входу;
– у разі помилки реєстрації, відображає повідомлення про помилку.

```
try {
    const data = await apiRequest('POST', '/api/register/', {
      username: formData.username,
      email: formData.email,
      password1: formData.password1,
      password2: formData.password2,
    });

    localStorage.setItem('accessToken', data.access);
    localStorage.setItem('refreshToken', data.refresh);



    setSuccess('Регистрация успешна!');
    setError('');
    console.log('Успешная регистрация:', data);

    window.location.href = '/login';
  } catch (err: any) {
  setError(err.response?.data?.error || 'Ошибка регистрации');
  console.error('Ошибка:', err.response?.data || err.message);
 }
};
```

Рисунок 3.8 – Форма реєстрації з API

Після успішної реєстрації користувач перенаправляється на сторінку входу (LoginPage), де він може увійти в свій новостворений обліковий запис. Компонент LoginPage відповідає за відображення форми входу та обробку даних, введених користувачем. Компонент також використовує хуки useNavigate та useSearchParams для перенаправлення користувача після успішного входу та отримання параметрів URL відповідно

Функція handleLogin (див. рис. 3.9) викликається при відправленні форми входу та виконує наступні дії:

– відправляє дані входу (ім’я користувача та пароль) на сервер за допомогою функції apiRequest з Services/api;
– якщо вхід пройшов успішно, зберігає отримані токени доступу та оновлення в localStorage;
– оновлює стан аутентифікації, викликаючи функцію onLogin, передану як проп;
– перенаправляє користувача на захищену сторінку (наприклад, головну сторінку) за допомогою хука useNavigate та параметра redirect, отриманого з URL;
– у разі помилки входу, відображає повідомлення про помилку.

```
const handleLogin = async (e: React.FormEvent<HTMLFormElement>) => {
   e.preventDefault();
   try {
     const data = await apiRequest('POST', '/api/login/', {
       username: formData.username,
       password: formData.password,
     });
     localStorage.setItem('accessToken', data.access);
     localStorage.setItem('refreshToken', data.refresh);
     onLogin();
     navigate(redirectTo, { replace: true });
   } catch (err: any) {
     setError(err.response?.data?.error || 'Ошибка входа');
     console.error('Ошибка:', err.response?.data || err.message);
   }
  };
```

Рисунок 3.9 – Логінізація юзера

Після успішного входу користувач перенаправляється на головну сторінку додатку за допомогою хука useNavigate та параметра redirect, отриманого з URL.

Після успішної реєстрації та створення облікового запису користувач може додавати потрібні товари до кошика для оформлення замовлення. Процес додавання товарів до кошика реалізовано за допомогою компонентів Cart, CartItemComponent та сторінки CartPage (див. рис. 3.10). Компонент Cart відповідає за відображення списку товарів у кошику. Він приймає масив об’єктів CartItem, загальну вартість товарів у кошику та функції для оновлення кількості товарів, видалення товарів та очищення кошика.

```
<Cart
 items={cartItems}
 totalPrice={totalPrice}
 clearCart={clearCart}
 handleQuantityChange={handleQuantityChange}
 removeCartItem={removeCartItem}
/>
```

Рисунок 3.10 – Відображення списку товарів у кошику

Компонент CartItemComponent відображає окремий товар у кошику. Він містить інформацію про товар, такі як назва, ціна та зображення, а також елементи керування кількістю товару та кнопку для його видалення з кошика.

Сторінка CartPage є контейнерним компонентом, який обробляє стан кошика та взаємодіє з API для отримання даних про товари в кошику, оновлення їх кількості, видалення та очищення кошика. При завантаженні сторінки CartPage викликається функція fetchCartData, яка відправляє GET-запит на /api/cart/ для отримання поточного стану кошика з сервера. Отримані дані використовуються для ініціалізації стану cartItems та totalPrice.

Функція handleQuantityChange (див. рис. 3.11) викликається при зміні кількості товару в кошику. Вона відправляє PUT-запит на /api/cart/update-cart-item/${cartItemId}/${newQuantity}/ для оновлення кількості товару на сервері

та оновлює локальний стан cartItems та totalPrice відповідно до отриманих даних.

```
  const handleQuantityChange = useCallback(
   async (cartItemId: number, newQuantity: number) => {
    try {
     const response = await apiRequest(
      'put',
      `/api/cart/update-cart-item/${cartItemId}/${newQuantity}/`
     );

     if (response) {
      const updatedCartItems = cartItems.map((item) =>
       item.id === cartItemId ? { ...item, quantity: newQuantity } : item
      );

      const updatedTotalPrice = updatedCartItems.reduce((total, item) => total + item.product.price *
item.quantity, 0);

      setCartItems(updatedCartItems);
      setTotalPrice(updatedTotalPrice);
     }
    } catch (error) {
     console.error('Ошибка при обновлении количества товара:', error);
    }
   },
   [cartItems]
  );
```

Рисунок 3.11 – Функція зміни кількості товарів

Функція removeCartItem (див. рис. 3.12) викликається при натисканні на кнопку «Видалити» для певного товару в кошику. Вона відправляє DELETE-запит на /api/cart/remove-from-cart/${cartItemId}/ для видалення товару з кошика на сервері та оновлює локальний стан cartItems та totalPrice відповідно до оновлених даних.

```
  const removeCartItem = async (cartItemId: number) => {
    try {
      await apiRequest('delete', `/api/cart/remove-from-cart/${cartItemId}/`);
      const updatedCartItems = cartItems.filter((item) => item.id !== cartItemId);
      const updatedTotalPrice = updatedCartItems.reduce((total, item) => total + item.product.price *
item.quantity, 0);

      setCartItems(updatedCartItems);
      setTotalPrice(updatedTotalPrice);
    } catch (error) {
      console.error('Ошибка при удалении товара из корзины:', error);
    }
  };
```

Рисунок 3.12 – Видалення товару з кошика

3.3 Маршрутизація проєкту

У React є вбудована система маршрутизації, яка дозволяє зіставляти запити до застосування з певними компонентами. Ключовою ланкою в роботі маршрутизації є модуль react-router, який містить основний функціонал роботи з маршрутизацією. Маршрутизація в React є важливим аспектом для створення динамічних односторінкових додатків (SPA – Single Page Applications). Вона дозволяє користувачам переходити між різними розділами або сторінками вебдодатку без необхідності перезавантаження всієї сторінки, що покращує користувацький досвід та продуктивність [8].

Основним компонентом для маршрутизації є BrowserRouter Цей компонент оточує весь додаток і забезпечує синхронізацію URL-адреси браузера з структурою маршрутів додатка. Всередині BrowserRouter використовуються інші компоненти, такі як Routes та Route, для визначення маршрутів та відповідних компонентів, які слід відображати (див. рис. 3.13).

Компонент Routes є контейнером для групи Route. Кожен компонент Route визначає шлях та компонент, який повинен бути відображений, коли URL-адреса збігається з цим шляхом.

Важливим аспектом маршрутизації в React є захист певних маршрутів або розділів додатку від несанкціонованого доступу.

```
return (
   <Router>
    <div className="flex flex-col min-h-screen">
      <Header isAuthenticated={isAuthenticated} onLogout={handleLogout} />
      <main className="flex-grow">
        <Routes>
          <Route path="/" element={<HomePage />} />
          <Route path="/catalog" element={<ProductListPage />} />
          <Route path="/category/:slug" element={<ProductListPage />} />
          <Route path="/product/:id" element={<ProductPage />} />
          <Route path="/cart" element={<CartPage />} />
          <Route path="/search" Component={SearchResultsPage} />
          <Route path="/checkout" element={<CheckoutPage />} />
          <Route path="/register" element={<RegistrationPage />} />
          <Route path="/login" element={<LoginPage onLogin={handleLogin} />} />
          <Route path="/products" element={<PrivateRoute path="/products"
component={ProductPage} />} />
        </Routes>
      </main>
      <Footer />
    </div>
   </Router>
  );
};
```

Рисунок 3.13 – Маршрутизація проєкту

3.3 Приклади роботи застосунку

Після заходу на сайт користувачеві потрібно створити обліковий запис (див. рис. 3.14). Він може натиснути на кнопку реєстрації або, якщо обліковий запис вже існує, на кнопку «Увійти». Після цього користувача перенаправляє

на головну сторінку. На головній сторінці користувач може переглядати різні категорії товарів, наприклад, «Вітаміни», «Білки», «Креатин» тощо. Після вибору потрібної категорії відкривається сторінка з відповідними товарами.

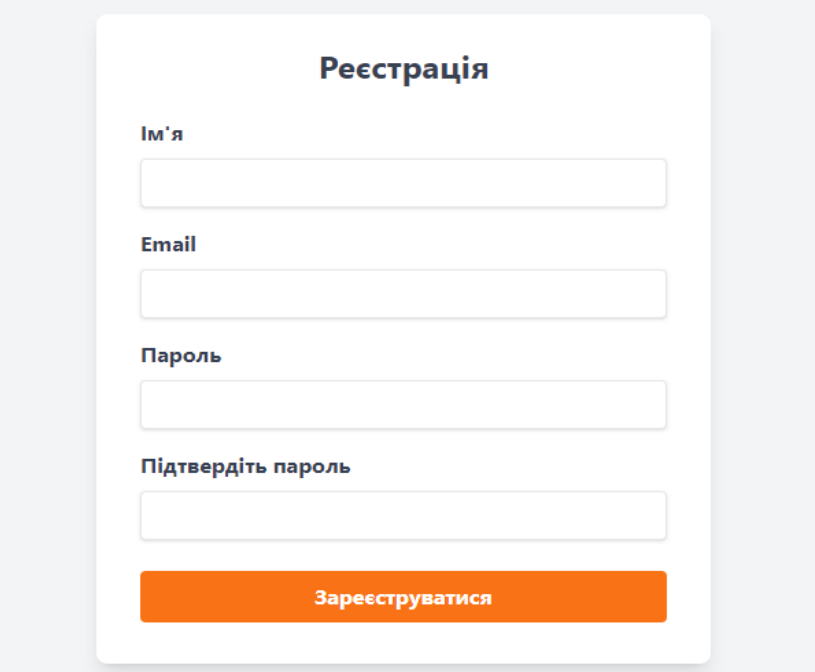

Рисунок 3.14 – Форма реєстрації

Наприклад, виберемо категорію «Білки», і відкриється сторінка з товарами (див. рис. 3.15). Тут користувач може ознайомитися з детальною інформацією про кожен продукт: складом, калорійністю, ціною.

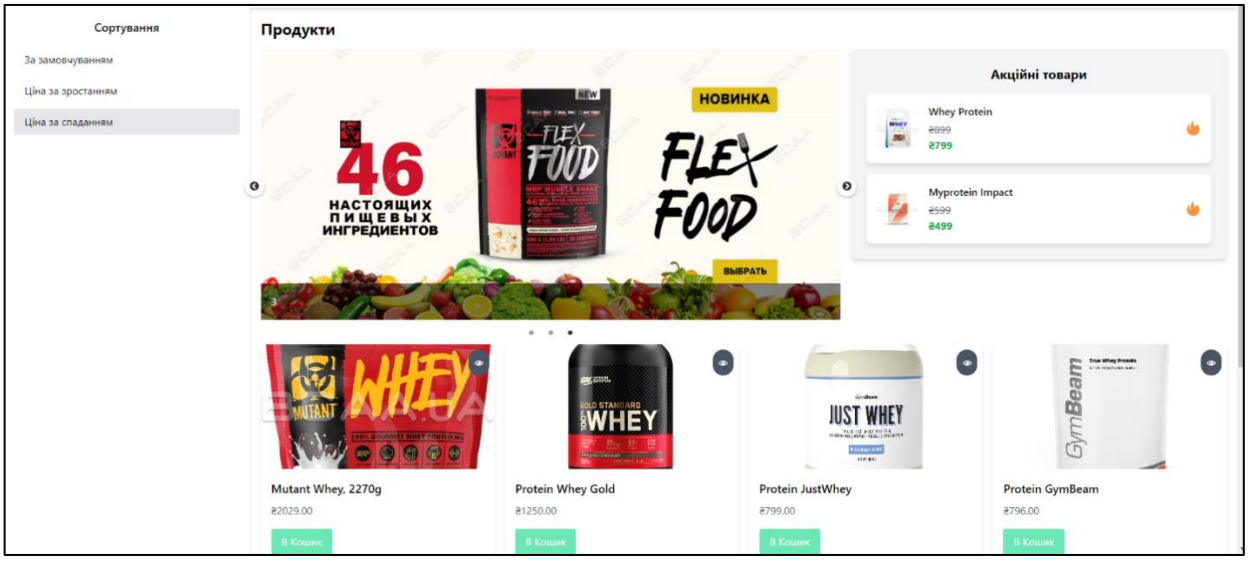

Рисунок 3.15 – Сторінка товарів

Якщо певний товар зацікавив користувача, він може додати його в кошик для покупок, натиснувши відповідну кнопку. Після цього можна продовжити вибір інших товарів або перейти до кошика (див. рис. 3.16), де відображаються всі вибрані продукти. На сторінці кошика користувач може переглянути загальну суму замовлення, змінити кількість одиниць товару або видалити непотрібні позиції.

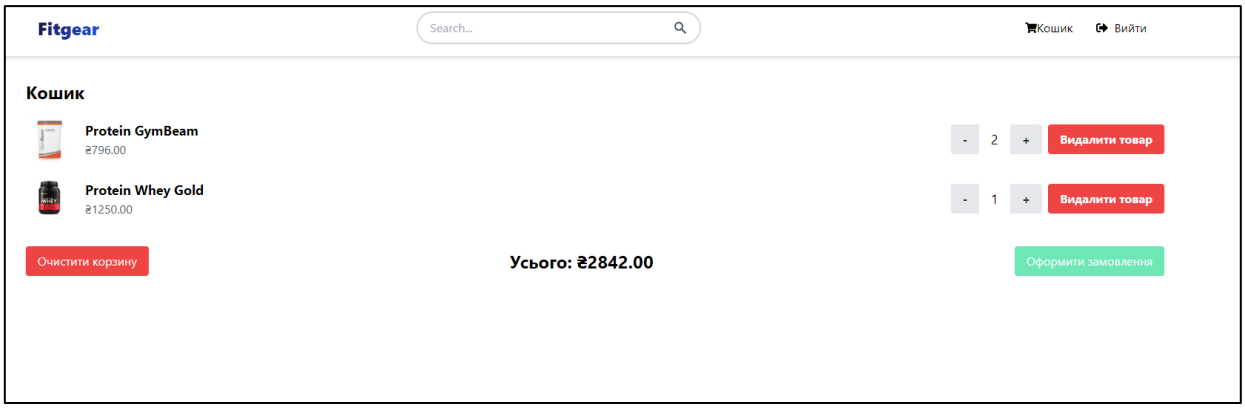

Рисунок 3.16 – Сторінка кошику

У розділі детально описано програмну реалізацію фронтенд частини вебдодатку інтернет-магазину спортивного харчування. Представлено структуру проєкту, яка забезпечує зрозумілість, підтримуваність та масштабованість коду. Наведено приклади реалізації ключових компонентів, таких як відображення товарів, реєстрація користувачів, авторизація, робота з кошиком покупок. Окрему увагу приділено маршрутизації проєкту з використанням react-router для створення динамічного односторінкового додатку. Продемонстровано роботу застосунку на прикладах взаємодії користувача з інтерфейсом, від реєстрації до оформлення замовлення.

ВИСНОВКИ

Сайти інтернет-магазинів відіграють важливу роль в сучасний час для користувачів та бізнесу. Метою кваліфікаційної роботи було вивчити та створити робочу фронтенд-частину, поєднану з бекенд-частиною. Під час виконання кваліфікаційної роботи було ретельно вивчено потреби та вимоги користувачів до сайтів спортивного харчування. Це дозволило створити функціональний та зручний у використанні додаток, який відповідає очікуванням цільової аудиторії. Для ефективної розробки фронтенд-частини було обрано React – потужну бібліотеку JavaScript, яка забезпечує високу продуктивність, гнучкість та можливість масштабування. Завдяки компонентному підходу та концепції віртуального DOM, React дозволив швидко та якісно реалізувати користувацький інтерфейс з плавною роботою та інтерактивністю.

У першому розділі було досліджено сферу сайтів спортивного харчування за функціоналом, наповненням тощо. У цьому розділі я провів аналіз існуючих рішень, вивчив особливості та вимоги до подібних сайтів, а також визначив необхідний функціонал для майбутнього додатку.

У другому розділі було виконано проєктування та використано допоміжні засоби для ефективної роботи сайту та взаємодії з даними. Тут я розробив архітектуру додатку, визначив необхідні компоненти та їх взаємодію, а також обрав відповідні бібліотеки та інструменти для розробки.

У третій частині було практично реалізовано велику частину важливого функціоналу та показано роботу додатку сайту інтернет-магазину спортивного харчування з можливостями реєстрації, авторизації, перегляду товарів, додавання до кошика, оформлення замовлень.

Таким чином, виконуючи всі поставлені завдання, було досягнуто мети даної кваліфікаційної роботи.

ПЕРЕЛІК ПОСИЛАНЬ

1. Всеукраїнська студентська наукова конференція розвиток сучасної науки. URL:https://archive.liga.science/index.php/conference-proceedings/issue/view/ukr-17.11.2023/52 (дата звернення: 30.03.2024).
2. Персоналізація як стратегія лояльності в електронній комерції2024 | The international scientific and practical journal. URL:https://www.researchgate.net/publication/378931042_Personalizacia_ak_strategia_loalnosti_v_elektronnij_komercii. (дата звернення: 27.04.2024).
3. US ecommerse sales 2013–2023. URL:https://www.oberlo.com/statistics/us-ecommerce-sales (дата звернення: 27.04.2024).
4. TailwindCSS documentation. URL: https://tailwindcss.com/ (дата звернення: 30.04.2024).
5. Webpack documentation. URL:https://github.com/webpack/docs/wiki/what-is-webpack. (дата звернення: 02.05.2024).
6. Введение в Webpack. URL:http://surl.li/uldjy. (дата звернення: 27.04.2024).
7. Folder Structure for a React JS Project. URL:https://www.geeksforgeeks.org/folder-structure-for-a-react-js-project/. (дата звернення: 05.05.2024).
8. Маршрутизація. Визначення маршрутів. URL:https://krypton.com.ua/rozdil-5-marshrutyzacziya-vyznachennya-marshrutiv/ (дата звернення: 03.05.2024).