НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
(повна назва інституту/факультету)

Кафедра інформатики та програмної інженерії
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2023 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення інформаційних систем»

спеціальності «121 Інженерія програмного забезпечення»

на тему: Архітектурне рішення веб-застосунку для цифрової дистрибуції

Виконав студент IV курсу, групи ІП-з91
(шифр групи)
Бубряк Микита Сергійович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник доцент, к.т.н., доц., Ліщук К. І. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант з графічної документації доцент, к.т.н., доц., Ліщук К. І. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент доцент кафедри ІСТ, к.т.н., доц., Писаренко А. В. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2023

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 Інженерія програмного забезпечення

Освітньо-професійна програма – Інженерія програмного забезпечення інформаційних систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2023 р.

ЗАВДАННЯ
на дипломний проєкт студенту

Бубряку Микиті Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема проєкту Архітектурне рішення веб-застосунку для цифрової дистрибуції

керівник проєкту Ліщук Катерина Ігорівна, доцент, к.т.н., доц.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «_31_» травня 2023 р. №_2102-с_

2. Термін подання студентом проєкту « 17 » червня 2023 року

3. Вихідні дані до проєкту: технічне завдання

4. Зміст пояснювальної записки
1) Аналіз вимог до програмного забезпечення: загальні положення, змістовний опис і аналіз предметної області, аналіз існуючих технологій та успішних IT-проєктів, розроблення функціональних та нефункціональних вимог, постановка задачі.
2) Моделювання та конструювання програмного забезпечення: моделювання та аналіз програмного забезпечення, архітектура програмного забезпечення, конструювання програмного забезпечення, аналіз безпеки даних.
3) Аналіз якості та тестування програмного забезпечення: аналіз якості програмного забезпечення, опис процесів тестування, опис контрольного прикладу.
4) Впровадження та супровід програмного забезпечення: розгортання програмного забезпечення, підтримка програмного забезпечення.

5) Технологічний розділ: керівництво користувача, методика випробувань програмного продукту.

5. Перелік графічного матеріалу

1) Схема структурна варіантів використань
2) Схема структурна компонентів програмного забезпечення
3) Креслення вигляду екранних форм

6. Консультанти розділів проєкту

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

7. Дата видачі завдання «10» березня 2023 року

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вивчення рекомендованої літератури	10.03.2023	
2	Аналіз існуючих методів розв’язання задачі	13.03.2023	
3	Постановка та формалізація задачі	19.03.2023	
4	Розробка інформаційного забезпечення	30.03.2023	
5	Алгоритмізація задачі	05.04.2023	
6	Обґрунтування вибору використаних технічних засобів	10.04.2023	
7	Розробка програмного забезпечення	24.04.2023	
8	Налагодження програми	29.04.2023	
9	Виконання графічних документів	07.05.2023	
10	Оформлення пояснювальної записки	23.05.2023	
11	Подання ДП на попередній захист	02.06.2023	
12	Подання ДП рецензенту	12.06.2023	
13	Подання ДП на основний захист	17.06.2023	

Студент _____ Микита БУБРЯК
(підпис) (ініціали, прізвище)

Керівник _____ Катерина ЛІЩУК
(підпис) (ініціали, прізвище)

АНОТАЦІЯ

Пояснювальна записка дипломного проєкту складається з чотирьох розділів, містить 49 таблиць, 39 рисунків та 28 джерел – загалом 105 сторінок.

Дипломний проєкт присвячений розробці архітектурного рішення веб-застосунку для цифрової дистрибуції.

Мета: забезпечення оптимізованого керування станом додатку шляхом реалізації архітектурного рішення, що спрощує написання логіки, налаштування та централізації стану, підтримує асинхронні запити та нормалізацію даних.

Об'єкт дослідження: Веб-розробка.

Предмет дослідження: Процеси розроблення веб-застосунку для цифрової дистрибуції із забезпеченням оптимізованого керування станом додатку.

У першому розділі викладено загальні положення, змістовно описано та проаналізовано предметну область, проаналізовано існуючі технології та успішні ІТ-проєкти, проведено аналіз вимог до програмного забезпечення, викладено постановку задачі.

У другому розділі проведено моделювання та аналіз програмного забезпечення, описано архітектуру програмного забезпечення, виконано конструювання програмного забезпечення, проведено аналіз безпеки даних.

У третьому розділі проведено аналіз якості програмного забезпечення, описано процеси тестування та контрольний приклад.

У четвертому розділі описано розгортання програмного забезпечення та забезпечено підтримку програмного забезпечення.

КЛЮЧОВІ СЛОВА: ВЕБ-ЗАСТОСУНОК, АРХІТЕКТУРНЕ РІШЕННЯ, ЦИФРОВА ДИСТРИБУЦІЯ, ОПТИМІЗАЦІЯ, КЕРУВАННЯ СТАНОМ ДОДАТКУ

ABSTRACT

The explanatory note of the diploma project consists of four sections, contains 49 tables, 39 figures and 28 sources – in total 105 pages.

The purpose of the diploma project is to develop a solution architecture for a digital distribution web app.

The goal is to provide optimized application state management by implementing an architectural solution that simplifies writing logic, configuration and centralization of an application state, supports asynchronous requests and data normalization.

Research object: Web development.

Research subject: Digital distribution web app development processes with optimized application state management.

In the first section, general principles were outlined, the subject area was meaningfully described and analyzed, existing technologies and successful IT projects were analyzed, software requirements analysis was conducted, and the problem statement was outlined.

In the second section, software modeling and analysis were performed, software architecture was described, software design was implemented, and data security analysis was conducted.

In the third section, software quality analysis was conducted, testing processes and control were described.

In the fourth section, software deployment and integration were described.

KEYWORDS: WEB APPLICATION, SOLUTION ARCHITECTURE, DIGITAL DISTRIBUTION, OPTIMIZATION, APPLICATION STATE MANAGEMENT

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний	2	
			проєкт		
2	А4	КПІ.ІП-з9101.045440.01.91	Технічне завдання	24	
3	А4	КПІ.ІП-з9101.045440.02.81	Пояснювальна записка	105	
4	А4	КПІ.ІП-з9101.045440.03.13	Текст програми	12	
5	А4	КПІ.ІП-з9101.045440.04.51	Програма та методика	6	
			тестування		
6	А4	КПІ.ІП-з9101.045440.05.34	Керівництво користувача	16	
7	А3	КПІ.ІП-з9101.045440.06.99.ССВ	Схема структурна варіантів	1	
			використань		
8	А3	КПІ.ІП-з9101.045440.06.99.ССМ	Схема структурна	1	
			компонентів		
			програмного забезпечення		
9	А3	КПІ.ІП-з9101.045440.06.99.КЕ	Креслення вигляду	1	
			екранних форм		

Змін.	Арк.	№ докум.	Підп.	Дата	КПІ.ІП-з9101.045440.00.90		
Розроб.		Бубряк М.С.			Відомість дипломного проєкту	Літ.	Аркуш 1 / Аркушів
Перевір.		Ліщук К.І.					
Н.контр.		Ліщук К.І.				КПІ ім. Ігоря Сікорського ФІОТ каф. ІПІ гр. ІП-з91	
Затв.		Жаріков Е.В.					

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

АРХІТЕКТУРНЕ РІШЕННЯ ВЕБ-ЗАСТОСУНКУ ДЛЯ ЦИФРОВОЇ ДИСТРИБУЦІЇ

Технічне завдання

КПІ.ІП-з9101.045440.01.91

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Катерина ЛІЩУК

Нормоконтроль:
_____ Катерина ЛІЩУК

Виконавець:
_____ Микита БУБРЯК

Київ – 2023

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Архітектурне рішення веб-застосунку для цифрової дистрибуції.

Галузь застосування:

Наведене технічне завдання поширюється на розробку архітектурного рішення веб-застосунку для цифрової дистрибуції «DD Web App» [КПІ.ІП-з9101.045440], котра використовується для оптимізованого керування станом додатку, та призначена для широкого кола користувачів, що бажають зручно та швидко обрати та придбати цифрові товари.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки архітектурного рішення веб-застосунку для цифрової дистрибуції є завдання на дипломне проєктування, затверджене кафедрою інформатики та програмної інженерії Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для спрощення та підвищення продуктивності керування станом веб-застосунків, в яких великі частини стану додатку потрібні у багатьох компонентах, з частим оновленням стану або із комплексною логікою оновлення стану.

Метою розробки є забезпечення оптимізованого керування станом додатку шляхом реалізації архітектурного рішення, що спрощує написання логіки, налаштування та централізації стану, підтримує асинхронні запити та нормалізацію даних.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

Програмне забезпечення повинно забезпечувати виконання наступних основних функцій:

4.1.1 Користувацького інтерфейсу

– інтерфейс реєстрації користувача (рисунок 4.1);

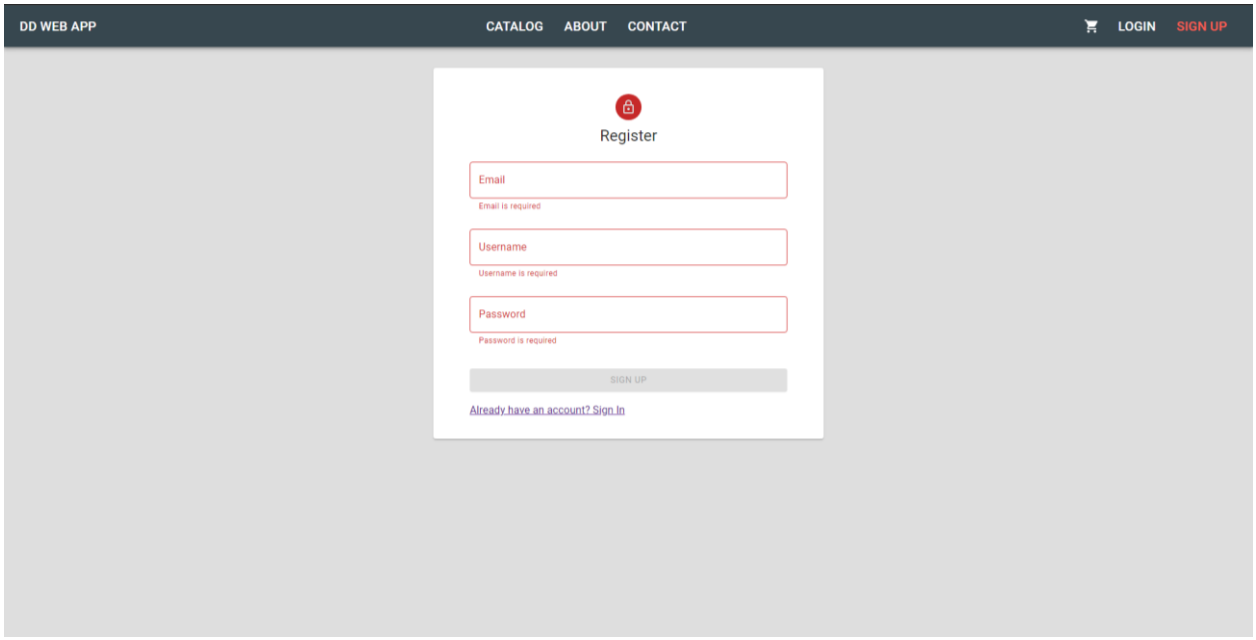


Рисунок 4.1 – Екранна форма інтерфейсу реєстрації користувача

– інтерфейс авторизації користувача (рисунок 4.2);

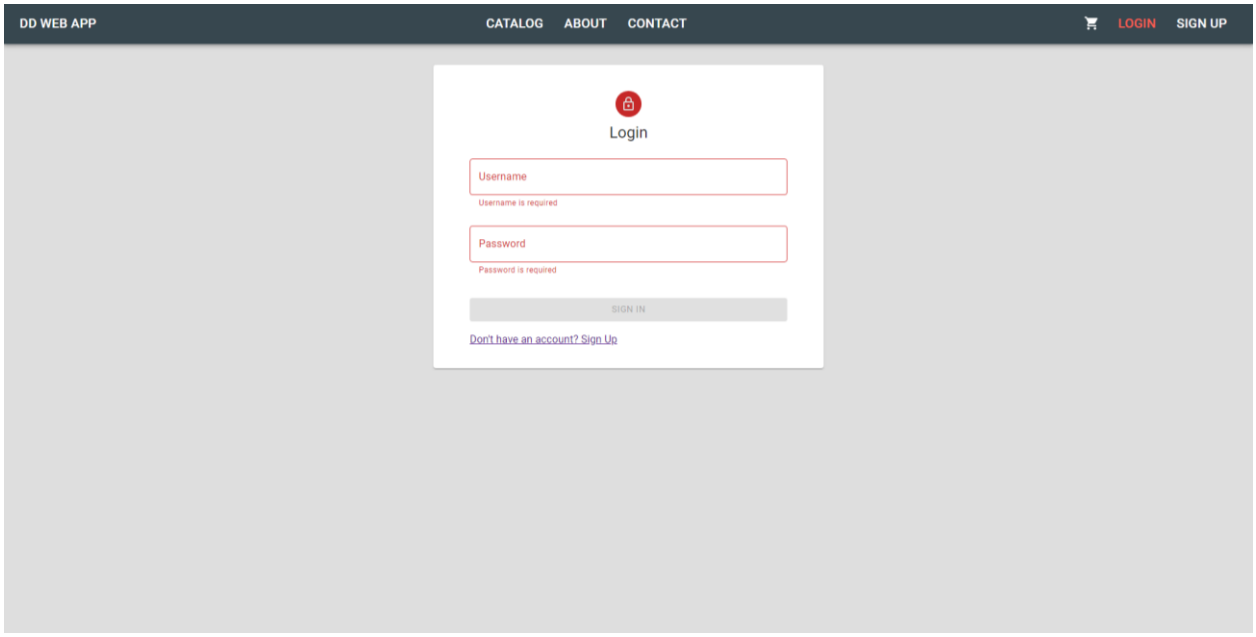


Рисунок 4.2 – Екранна форма інтерфейсу авторизації користувача

– інтерфейс каталогу продуктів (рисунок 4.3);

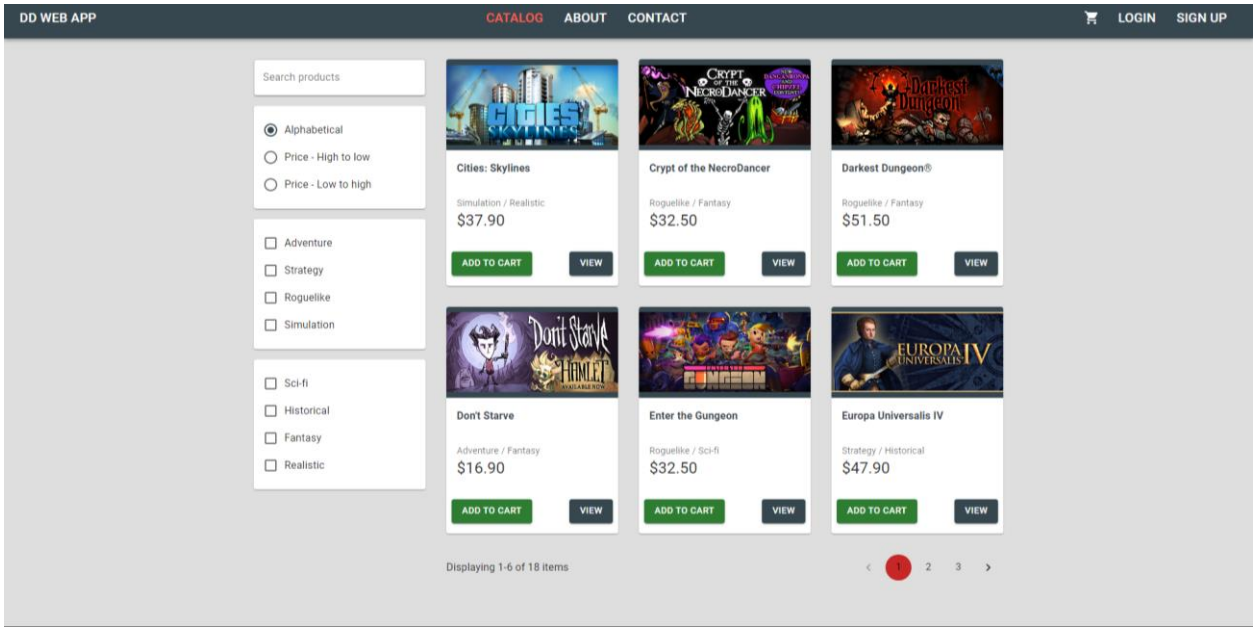


Рисунок 4.3 – Екранна форма інтерфейсу каталогу продуктів

– взаємодія з елементами посторінкової навігації у каталозі (рисунок 4.4);

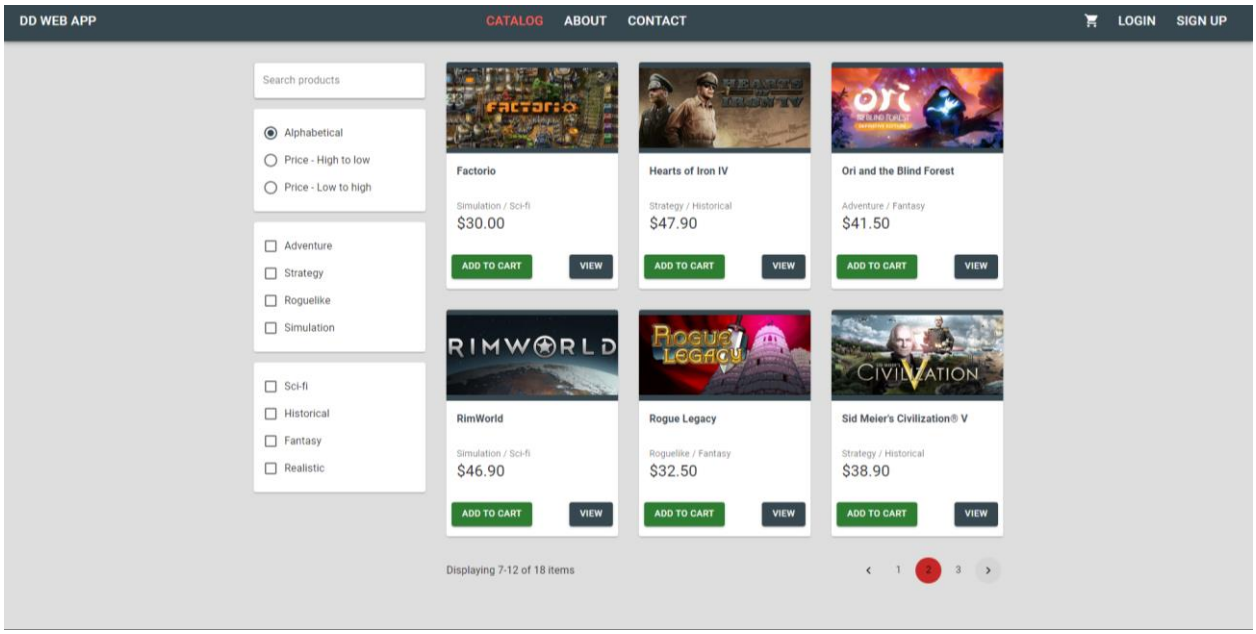


Рисунок 4.4 – Екранна форма взаємодії з елементами посторінкової навігації

– взаємодія з елементами сортування продуктів у каталозі (рисунок 4.5);

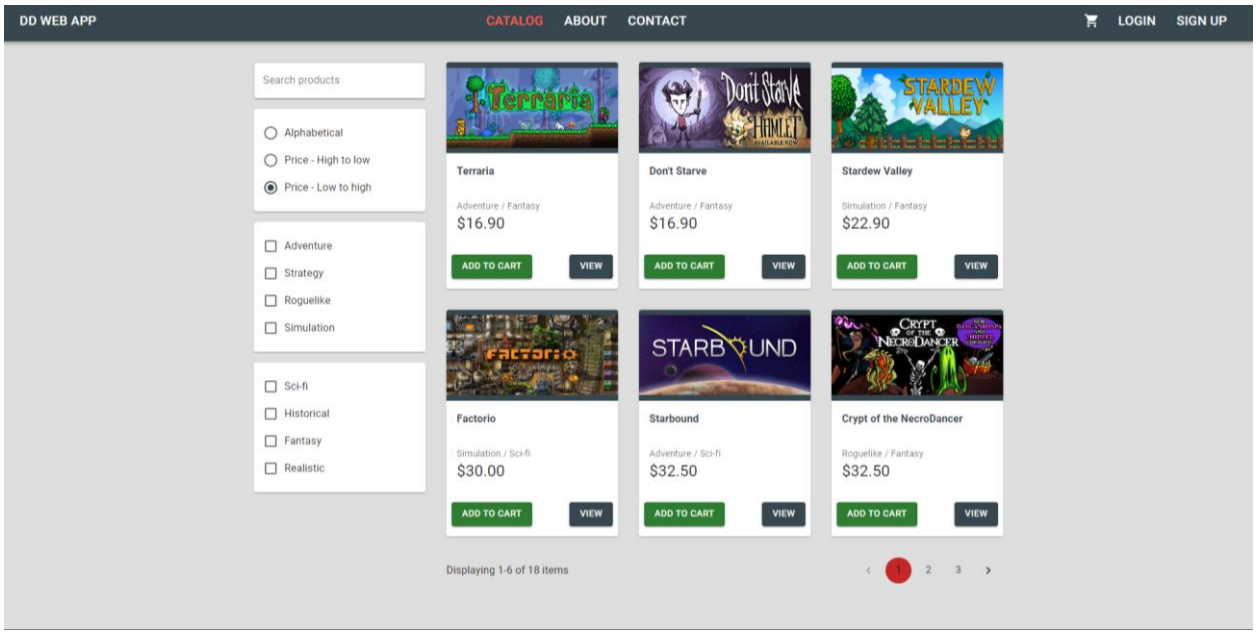


Рисунок 4.5 – Екранна форма взаємодії з елементами сортування продуктів

– взаємодія з елементами фільтрування продуктів у каталозі (рисунок 4.6);

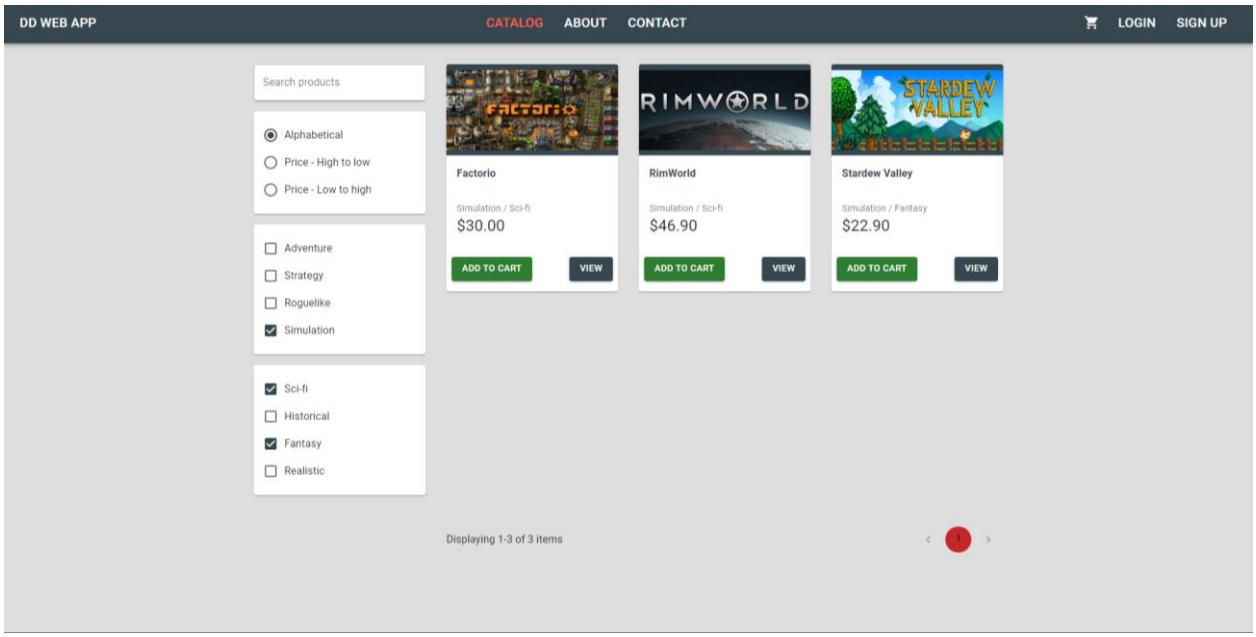


Рисунок 4.6 – Екранна форма взаємодії з елементами фільтрування продуктів

– взаємодія з елементом пошуку продуктів у каталозі (рисунок 4.7);

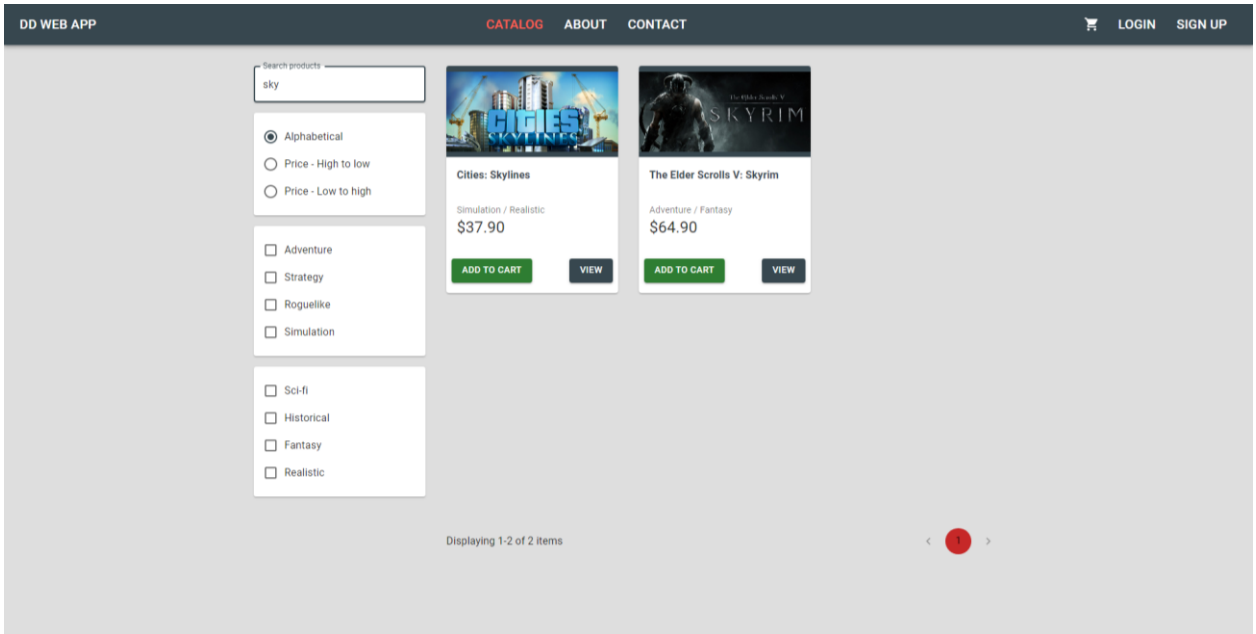


Рисунок 4.7 – Екранна форма взаємодії з елементом пошуку продуктів

– інтерфейс деталей продукту (рисунок 4.8);

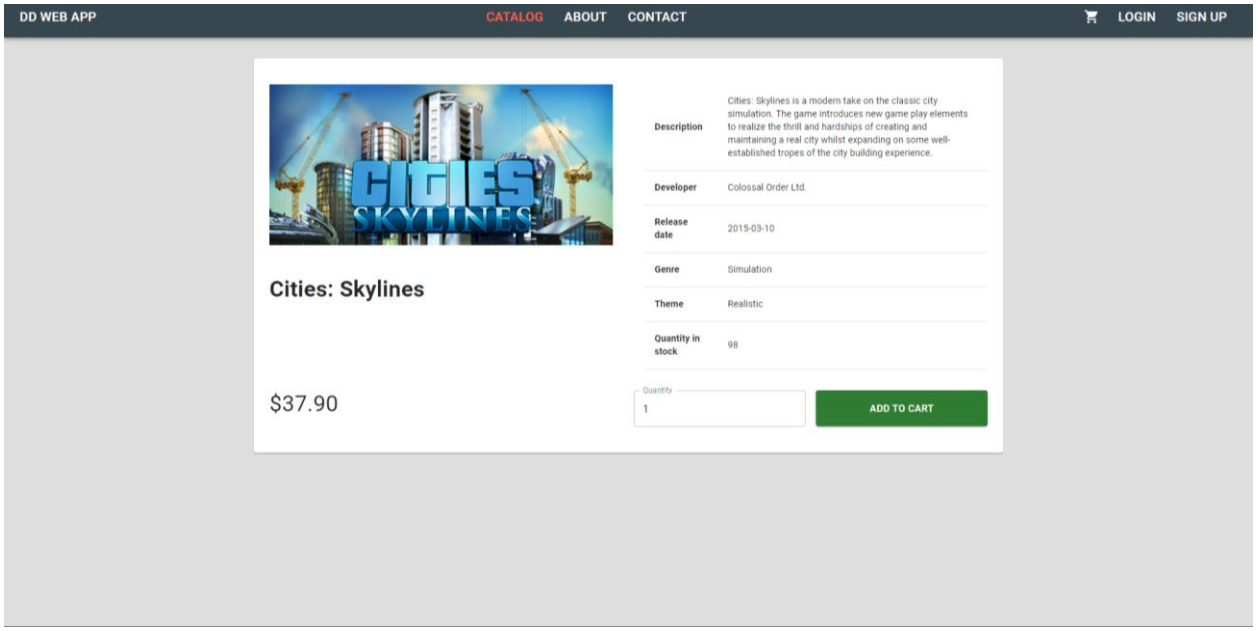


Рисунок 4.8 – Екранна форма інтерфейсу деталей продукту

– взаємодія з елементом додавання продукту до кошику (рисунок 4.9);

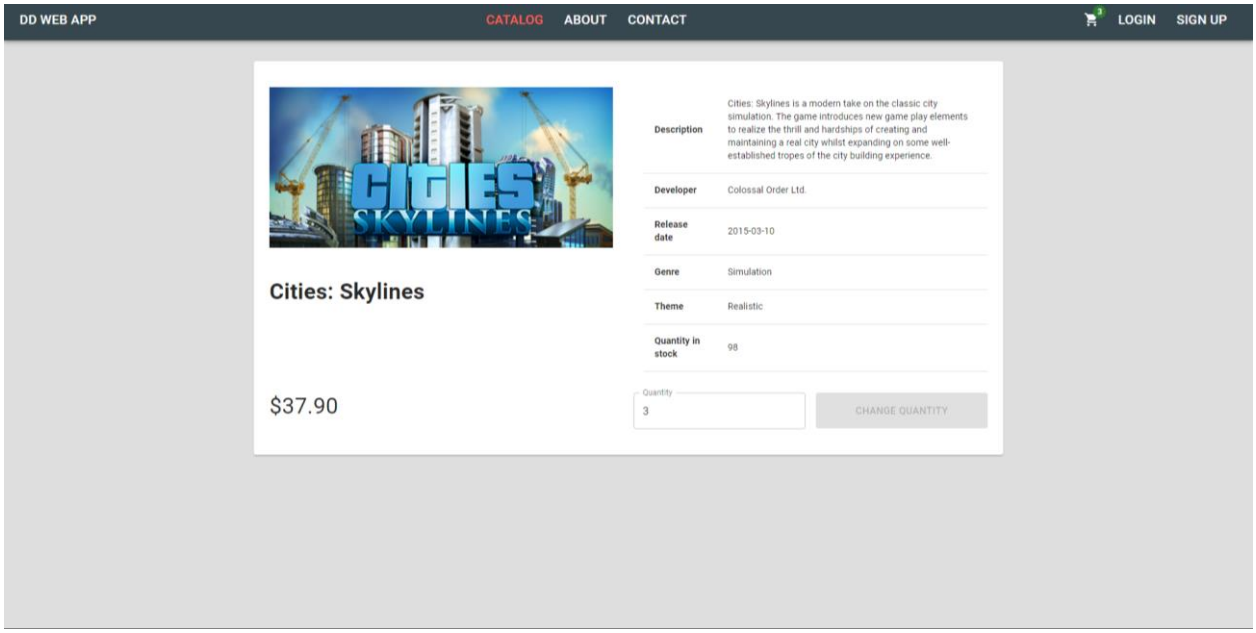


Рисунок 4.9 – Екранна форма взаємодії з елементом додавання продукту до кошику

– інтерфейс кошику (рисунок 4.10);

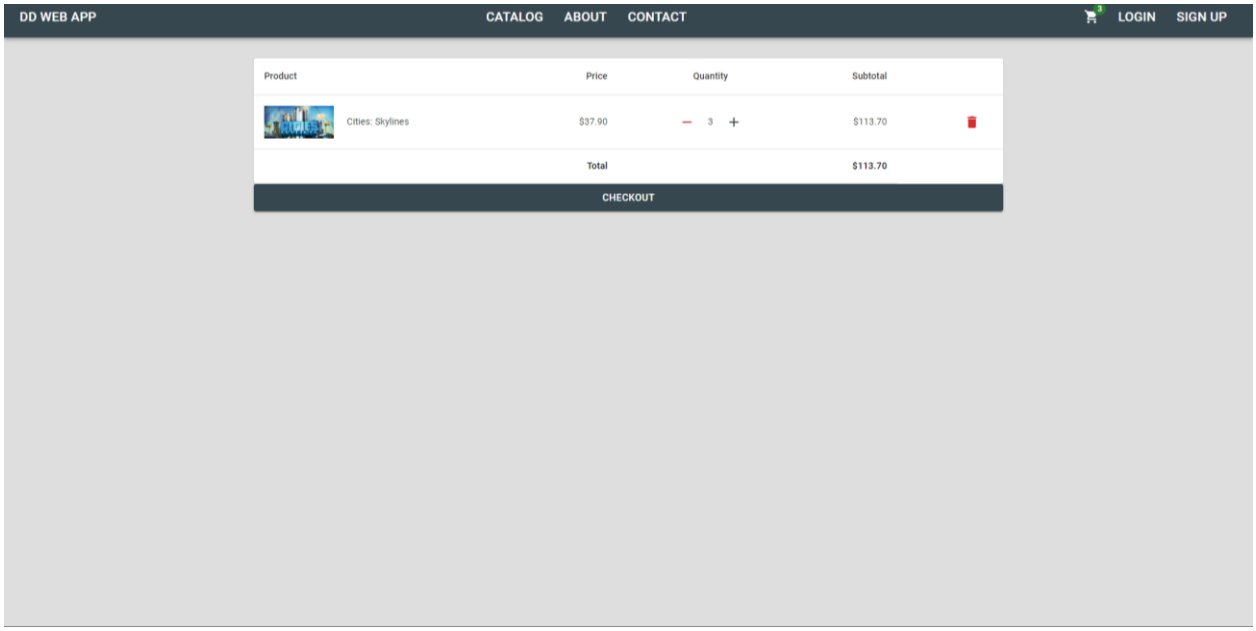


Рисунок 4.10 – Екранна форма інтерфейсу кошику

– інтерфейс чекауту (рисунок 4.11);

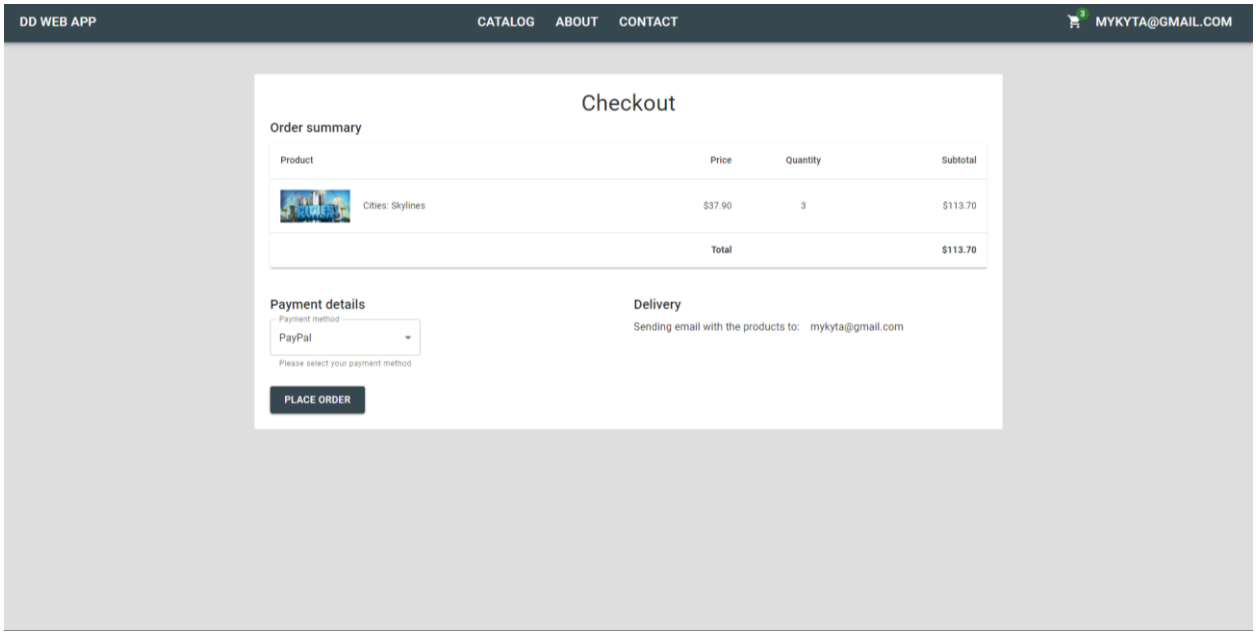


Рисунок 4.11 – Екранна форма інтерфейсу чекауту

– інтерфейс створення замовлення (рисунок 4.12);

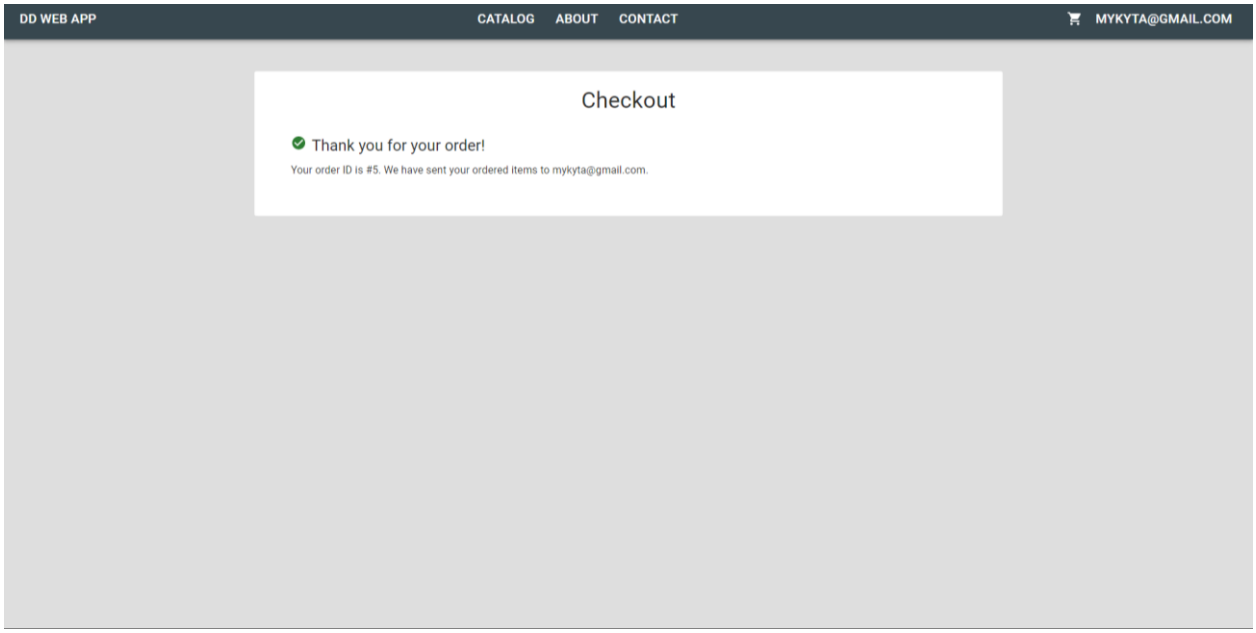


Рисунок 4.12 – Екранна форма інтерфейсу створення замовлення

– інтерфейс списку замовлень (рисунок 4.13);

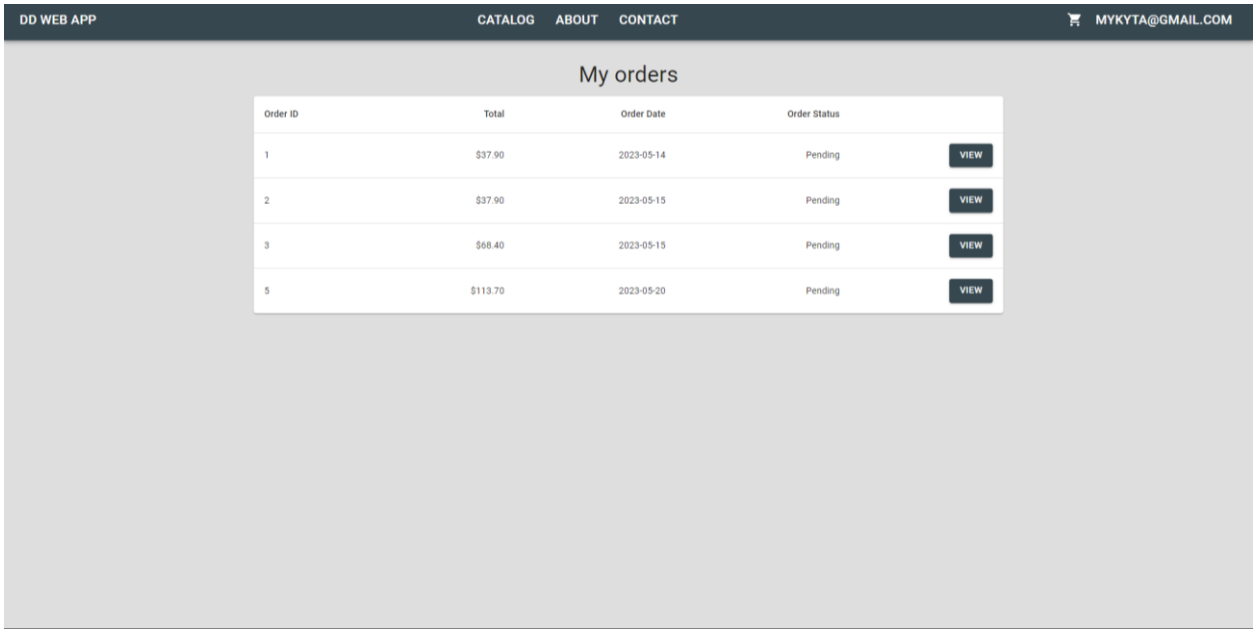


Рисунок 4.13 – Екранна форма інтерфейсу списку замовлень

– інтерфейс деталей замовлення (рисунок 4.14);

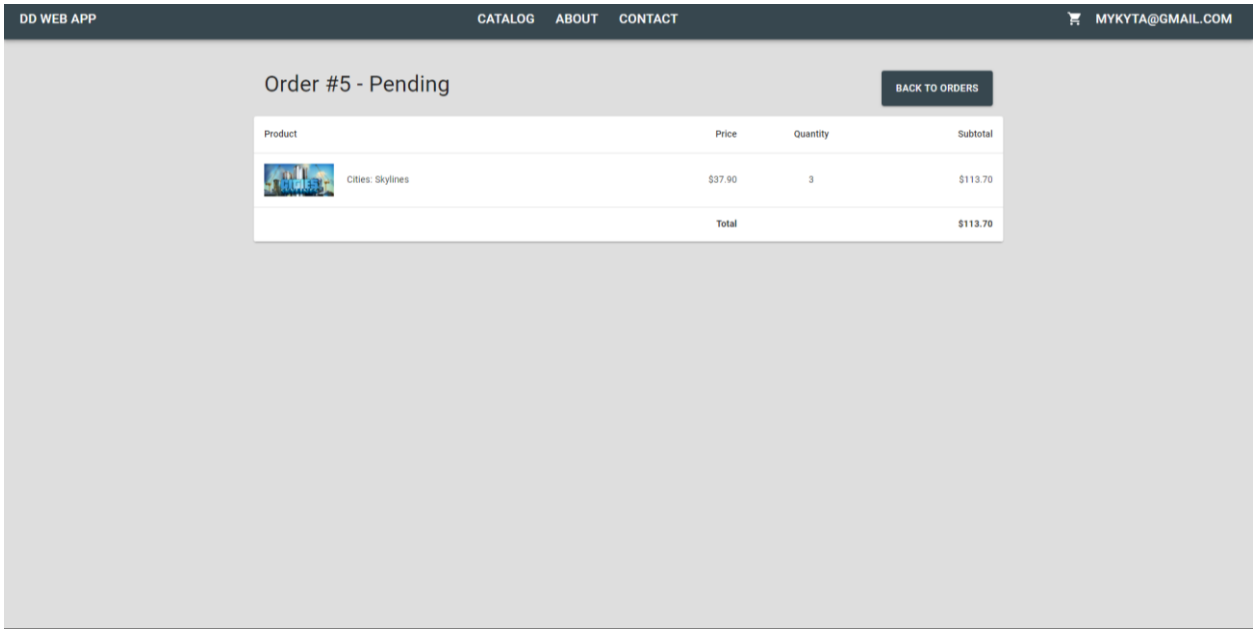


Рисунок 4.14 – Екранна форма інтерфейсу деталей замовлення

– повідомлення помилки 400 Bad Request (рисунок 4.15);

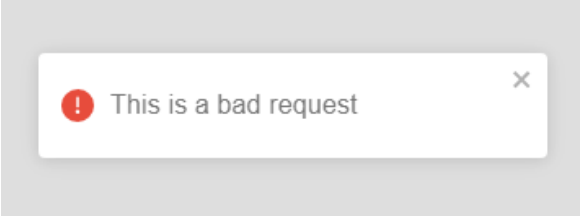


Рисунок 4.15 – Екранна форма повідомлення помилки 400 Bad Request

– повідомлення помилки 401 Unauthorized (рисунок 4.16);

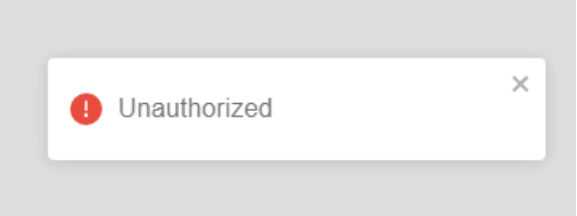


Рисунок 4.16 – Екранна форма повідомлення помилки 401 Unauthorized

– інтерфейс помилки 404 Not Found (рисунок 4.17);

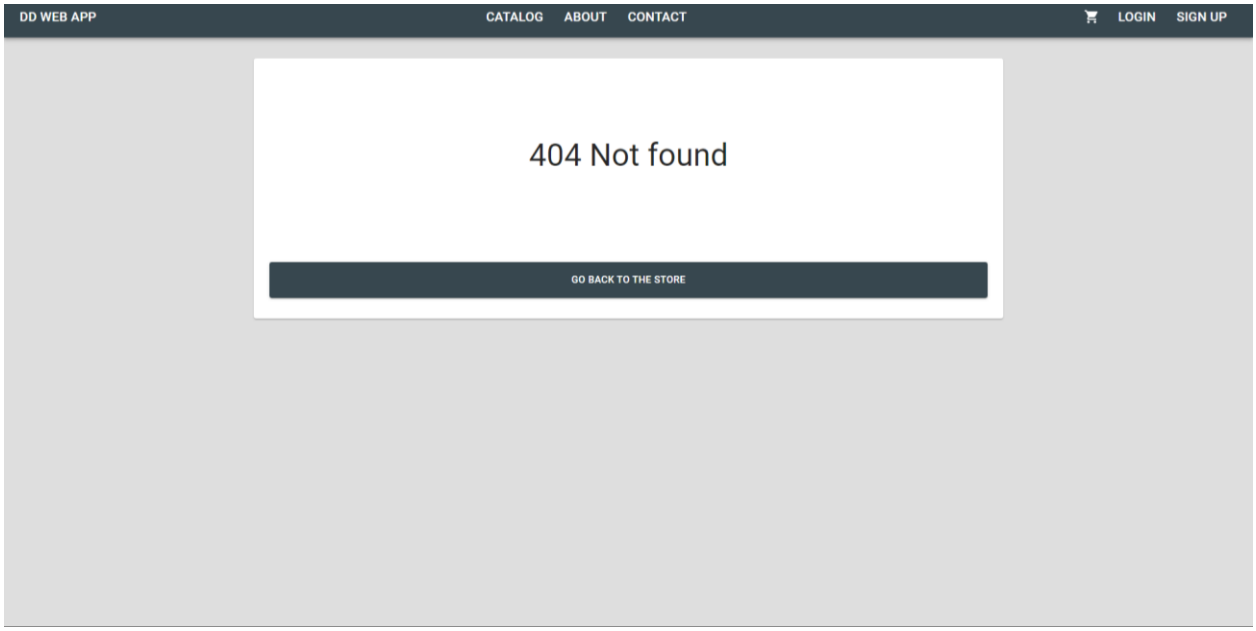


Рисунок 4.17 – Екранна форма інтерфейсу помилки 404 Not Found

– інтерфейс помилки 500 Internal Server Error (рисунок 4.18).

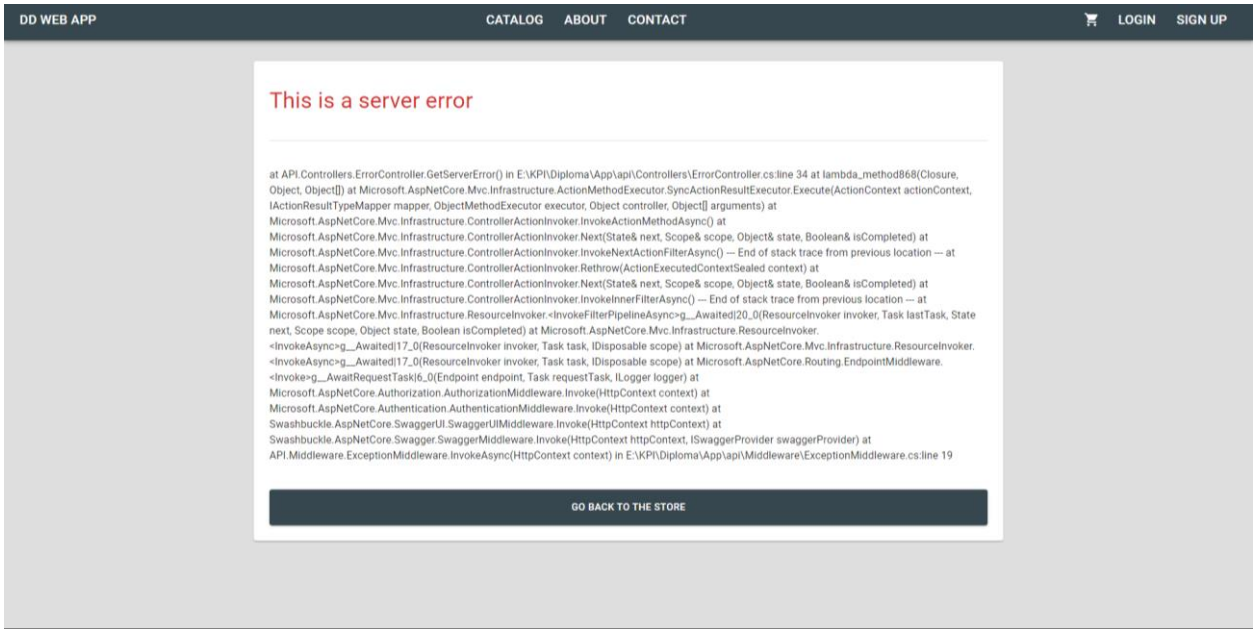


Рисунок 4.18 – Екранна форма інтерфейсу помилки 500 Internal Server Error

4.1.2 Для гостя (незареєстрованого користувача):

- реєстрація користувача;
  - введення електронної пошти;
    - валідація електронної пошти на необхідність введення, на відповідність шаблону регулярного виразу, на унікальність;
    - підсвічення поля червоним надписом при некоректно введених даних;
  - введення імені користувача;
    - валідація імені користувача на необхідність введення, на унікальність;
    - підсвічення поля червоним надписом при некоректно введених даних;
  - введення пароля;
    - валідація пароля на необхідність введення, на довжину більше шести символів, на наявність символу верхнього регістру, символу нижнього регістру, цифри та спеціального символу.
    - підсвічення поля червоним надписом при некоректно введених даних;
  - активація кнопки реєстрації при успішній валідації на стороні клієнта;
- каталог;
  - перегляд каталогу продуктів;
    - відображення компонента з індикатором завантаження при завантаженні продуктів;
  - посторінкова навігація у каталозі;

- вибір сторінки натисканням на номер сторінки або на кнопки навігації вперед або назад;
- деактивація кнопок навігації вперед або назад якщо відсутня попередня та/або наступна сторінки відносно поточної;

- сортування продуктів у каталозі;
  - вибір способу сортування натисканням на радіокнопки зі способами сортування;
- фільтрування продуктів у каталозі;
  - вибір способів фільтрування натисканням прапорців зі способами фільтрування;
- пошук продуктів у каталозі;
  - введення терміну пошуку у текстове поле;
- перегляд деталей продукту;
  - відображення помилки 404 not found при переході на сторінку продукту з неіснуючим id;

- кошик;
  - перегляд кошику;
  - додавання продукту до кошику;
    - введення зміни кількості продуктів для додавання в кошик;
    - деактивація кнопки додавання продукту при введенні нульової кількості;
  - віднімання продукту з кошику;
  - видалення продукту з кошику.

4.1.3 Для зареєстрованого користувача:

- авторизація користувача;
  - введення імені користувача;
    - валідація імені користувача на необхідність введення;
    - підсвічення поля червоним надписом при некоректно введених даних;
  - введення пароля;
    - валідація пароля на необхідність введення.
    - підсвічення поля червоним надписом при некоректно введених даних;
  - активація кнопки авторизації при успішній валідації на стороні клієнта;
  - виведення повідомлення помилки unauthorized при авторизації з неіснуючими обліковими даними;
- каталог;
  - перегляд каталогу продуктів;
    - відображення компонента з індикатором завантаження при завантаженні продуктів;
  - посторінкова навігація у каталозі;
    - вибір сторінки натисканням на номер сторінки або на кнопки навігації вперед або назад;
    - деактивація кнопок навігації вперед або назад якщо відсутня попередня та/або наступна сторінки відносно поточної;
  - сортування продуктів у каталозі;
    - вибір способу сортування натисканням на радіокнопки зі способами сортування;
  - фільтрування продуктів у каталозі;

- вибір способів фільтрування натисканням прапорців зі способами фільтрування;
- пошук продуктів у каталозі;
  - введення терміну пошуку у текстове поле;
- перегляд деталей продукту;
  - відображення помилки 404 not found при переході на сторінку продукту з неіснуючим id;

- кошик;
  - перегляд кошику;
  - додавання продукту до кошику;
    - введення зміни кількості продуктів для додавання в кошик;
    - деактивація кнопки додавання продукту при введенні нульової кількості;
  - віднімання продукту з кошику;
  - видалення продукту з кошику;
- замовлення;
  - чекаут;
    - вибір методу оплати;
    - виведення повідомлення помилки при відсутньому методі оплати;
  - створення замовлення;
  - перегляд списку замовлень;
    - відображення компонента з індикатором завантаження при завантаженні замовлень;
  - перегляд деталей замовлення.
    - відображення помилки 404 not found при переході на сторінку замовлення з неіснуючим id.

4.1.4 Додаткові вимоги:

– забезпечення підтримки англійської локалізації інтерфейса.

4.2 Вимоги до надійності

Передбачити валідацію введення інформації та захист від некоректних дій користувача. Забезпечити цілісність інформації в базі даних.

4.3 Умови експлуатації

Умови експлуатації згідно ДСанПіН 3.3.2.007 – 98.

4.3.1 Вид обслуговування

Вимоги до виду обслуговування не висуваються.

4.3.2 Обслуговуючий персонал

Вимоги до обслуговуючого персоналу не висуваються.

4.4 Вимоги до складу і параметрів технічних засобів

Програмне забезпечення повинно функціонувати на IBM-сумісних персональних комп‘ютерах.

Мінімальна конфігурація технічних засобів:

– тип процесору: Intel Core i5;
– об‘єм ОЗП: 4 Гб;
– підключення до мережі Інтернет зі швидкістю від 20 мегабіт.

Рекомендована конфігурація технічних засобів:

– тип процесору: Intel Core i7;
– об‘єм ОЗП: 8 Гб;
– підключення до мережі Інтернет зі швидкістю від 100 мегабіт.

4.5 Вимоги до інформаційної та програмної сумісності

Програмне забезпечення повинно працювати під управлінням операційних систем сімейства Windows та підтримувати браузер Google Chrome.

4.5.1 Вимоги до вхідних даних

Особливі вимоги до вхідних даних не висуваються.

4.5.2 Вимоги до вихідних даних

Особливі вимоги до вихідних даних не висуваються.

4.5.3 Вимоги до мови розробки

Розробку виконати на мові програмування C# для серверної частини та Code First підходу для доступу до бази даних, а також мовою програмування TypeScript для клієнтської частини.

4.5.4 Вимоги до середовища розробки

Розробку виконати на платформі .NET з використанням фреймворків ASP.NET Web API та Entity Framework Core для серверної частини та доступу до реляційної бази даних PostgreSQL, а також з використанням бібліотеки React для клієнтської частини.

4.5.5 Вимоги до представленню вихідних кодів

Вихідний код програми має бути представлений у вигляді текстових файлів.

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не висуваються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не висуваються.

4.8 Спеціальні вимоги

Веб-застосунок розгорнути на платформі з безкоштовним планом для розгортання full stack додатків і високодоступних кластерів баз даних.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1 Попередній склад програмної документації

У склад супроводжувальної документації повинні входити наступні документи на аркушах формату А4:

– пояснювальна записка;
– технічне завдання;
– текст програми;
– програма та методика тестування;
– керівництво користувача.

Графічна частина повинна бути виконана на аркушах формату А3 та містити наступні документи:

– схема структурна варіантів використання;
– схема структурна компонентів програмного забезпечення;
– креслення вигляду екранних форм.

5.2 Спеціальні вимоги до програмної документації

Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні містити всі необхідні коментарі.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№	Назва етапу	Строк	Звітність
1.	Вивчення рекомендованої літератури	10.03.2023	Пояснювальна записка
2.	Аналіз існуючих методів розв’язання задачі	13.03.2023	Технічне завдання
3.	Постановка та формалізація задачі	19.03.2023	Специфікації програмного забезпечення
4.	Розробка інформаційного забезпечення	30.03.2023	Схема структурна компонентів програмного забезпечення
5.	Алгоритмізація задачі	05.04.2023	Тексти програмного забезпечення
6.	Обґрунтування вибору використаних технічних засобів	10.04.2023	Пояснювальна записка
7.	Розробка програмного забезпечення	24.04.2023	Пояснювальна записка
8.	Налагодження програми	29.04.2023	Пояснювальна записка
9.	Виконання графічних документів	07.05.2023	Технічна документація
10.	Оформлення пояснювальної записки	23.05.2023	Пояснювальна записка

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Пояснювальна записка
до дипломного проєкту

на тему: Архітектурне рішення веб-застосунку для цифрової дистрибуції

КПІ.ІП-з9101.045440.02.81

Київ – 2023

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

IDE	–	Integrated Development Environment.
API	–	Application Programming Interface.
IT	–	Інформаційні Технології.
BPMN	–	Business Process Model and Notation.
ER	–	Entity-Relation diagram.
ОС	–	Операційна Система.
БД	–	База Даних.
SQL	–	Structured Query Language.
LINQ	–	Language Integrated Query.
ORM	–	Object-Relational Mapping.
HTTP	–	Hypertext Transfer Protocol.
REST	–	Representational State Transfer.
URI	–	Uniform Resource Identifier.
DOM	–	Document Object Model.
SPA	–	Single Page Application.
GUID	–	Globally Unique Identifier.
JWT	–	JSON Web Token.
JSON	–	JavaScript Object Notation.
CORS	–	Cross-Origin Resource Sharing.
CRUD	–	Create Read Update Delete.

ВСТУП

Оскільки вимоги до односторінкових веб-застосунків дедалі ускладнюються, наш код повинен керувати більшим станом додатку, ніж будь-коли раніше. Цей стан може включати локально створені дані, які ще не збережено на сервері, а також відповіді сервера та кешовані дані. Стан інтерфейсу користувача також ускладнюється, оскільки нам потрібно керувати статусами завантаження, заповненням компонентів продуктами, фільтрами запитів тощо. Керувати цим станом, що розсіяний по багатьом компонентам та постійно змінюється, важко. У якийсь момент більше не зрозуміло, що відбувається у додатку, оскільки розробник втрачає контроль над його станом. Коли система непрозора та недетермінована, важко відтворювати помилки або додавати нові функції. З цією складністю важко впоратися, оскільки ми змішуємо дві концепції, які дуже важко зрозуміти людському розуму: мутація та асинхронність.

Redux намагається зробити мутації стану передбачуваними, накладаючи певні обмеження на те, як і коли можуть відбуватися оновлення.

Можливими сферами застосування розробки є веб-застосунки, в яких великі частини стану додатку потрібні у багатьох місцях, з частим оновленням стану або із комплексною логікою оновлення стану.

Метою даної розробки є забезпечення оптимізованого керування станом додатку шляхом реалізації архітектурного рішення, що спрощує написання логіки, налаштування та централізації стану, підтримує асинхронні запити та нормалізацію даних.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Загальні положення

Цифрова дистрибуція – це один із сучасних методів швидкого поширення легального цифрового медіа контенту як аудіо, фільми, відеоігри тощо за допомогою Інтернету без використання фізичних носіїв (папір, компакт-диски, касети). Перевага цифрового розповсюдження полягає в легкому і швидкому пошуку та придбанні копій необхідного програмного забезпечення або іншого вмісту, яке має можливість так розповсюджуватись через мережу Інтернет та відповідне програмне забезпечення на пристрої.

Цифрова дистрибуція також є більш екологічною, ніж фізична. Оптичні диски виготовлені з полікарбонатного пластику та алюмінію. Для їх створення потрібно використовувати багато природного газу, нафти і води. Захисні футляри для оптичних дисків виготовлені з полівінілхлориду, відомого канцерогену.

У сфері розваг та відпочинку відеоігри створили значний ринок, викликаючи необхідність у можливості постачання та придбання все більшої і більшої кількості відеоігор найрізноманітніших жанрів та тематик, що найзручнішим чином виконується електронно через Інтернет, на заміну йдучих у минуле компакт-дисків.

Основні переваги цифрової дистрибуції в порівнянні з раніше домінуючим роздрібним розповсюдженням відеоігор включають значне зниження витрат на виробництво, розгортання та зберігання. Ігри, придбані в цифровому вигляді, є законними ліцензіями, а не продаються, тобто споживачі не мають законного права власності та не можуть перепродавати свої ігри.

Каталоги роздрібного магазину і цифрового магазину можуть мати схожий вигляд. Найновіші та найпривабливіші назви відображаються спереду, щоб привернути увагу клієнтів. Однак під час подальшого огляду стало зрозуміло, що фізичний магазин не може встигати за обсягами

постачання та вибору за цифровим магазином. У той час як типовий роздрібний магазин повинен планувати співвідношення між попитом і пропозицією, інтернет-магазин може фактично накопичувати нескінченну пропозицію, не турбуючись про попит, оскільки на віртуальних полицях завжди є місце.

Веб-застосунки для цифрової дистрибуції вирізняються не лише кількістю, але й асортиментом і різноманітністю продуктів. В Інтернеті можна знайти не тільки ті самі ігри, що є у будь-якому місцевому ігровому магазині, але й багато інших. Велика частина ігор, доступних онлайн, продається виключно в цифровій формі, оскільки це надає можливість розробникам публікувати свої ігри за нижчою ціною як для себе, так і для покупця. Публікація гри в цифровій формі також є способом для розробника випробувати свій продукт на ринку. Прикладами цього є успішні інді-ігри, такі як Limbo, Super Meat Boy і Magicka, які випустили роздрібні версії для додаткового розширення продажів після того, як отримали хороший прийом на цифровому ринку [1].

Розробник і власник прав інтелектуальної власності Steam, співзасновник компанії Valve Гейб Ньюелл зазначає, що «Найгіршими днями для розробки ігор були дні картриджів для NES. Це був величезний ризик – усі ці гроші лежали десь у силіконі на складі, тож ви були б консервативними у рішеннях, які, на вашу думку, могли б прийняти, дуже консервативними у інтелектуальних власностях, які ви підписували, ваш творчий напрямок не змінювався б, і так далі. Зараз це протилежна крайність: ми можемо розмістити щось у Steam, доставити це людям у всьому світі, внести зміни. Ми можемо піти на більш цікаві ризики. Роздрібна торгівля не знає, як поводитися з цими іграми. У цифровій дистрибуції Steam немає обмежень щодо місця на полицях» [2].

Новіша платформа продажів не обов’язково означає лише новий контент. Завдяки цифровій дистрибуції багато ігор, які протягом багатьох років зникали з полиць магазинів, знайшли нове життя. Це дозволяє

досвідченим гравцям заново відчути ігри зі свого минулого, а новачкам відкрити для себе стару «класику». Одним із сервісів, які спеціалізуються на відродженні та розповсюдженні цих класичних відеоігор, є сервіс дистрибуції GOG.com [3].

Цифрова дистрибуція також пропонує нові структурні можливості для всієї індустрії відеоігор, яка до появи цифрових медіа як відповідного засобу розповсюдження зазвичай будувалася навколо відносин між розробником відеоігор, який створив гру, та видавцем відеоігор, який фінансував і організовував дистрибуцію та продаж. Підвищені витрати на виробництво на початку 2000-х років змусили багатьох видавців відеоігор уникнути ризиків і призвели до відмови від багатьох невеликих проєктів розробки ігор [4].

Зростаюча поширеність цифрової дистрибуції дозволила незалежним розробникам відеоігор продавати та поширювати свої ігри без необхідності укладати угоди з видавцями. Не маючи потреби покладатися на звичайні фізичні роздрібні продажі, незалежні розробники досягли успіху завдяки продажу ігор, які зазвичай не приймаються видавцями для дистрибуції [5].

Оскільки електронна комерція розтягує взаємодії в просторі та часі, а тому вимагає більшої довіри, ніж традиційні покупки, проблема перетворення відвідувачів на споживачів стала більш важливою. Для того, щоб перетворити відвідувачів на споживачів, відвідувач повинен знати, що його гроші в надійних руках і що його продукт або послугу буде надано згідно з обіцянками. Тому очікується, що підприємства електронної комерції завоюють довіру споживачів.

Працювати над створенням надійного веб-застосунку цифрової дистрибуції актуально з кількох причин. Дослідники широко вивчали концепцію довіри в багатьох областях. Крістоф Стаутхейзен вважає, що «довіра – це бажання однієї сторони бути вразливою до дій іншої сторони на основі очікування, що інша сторона виконає певну дію щодо довіреної особи, незалежно від здатності іншої сторони стежити або контролювати її» [6].

Для власників електронної комерції життєво важливо шукати способи підвищити довіру клієнтів, оскільки зацікавлені відвідувачі сайту, які можуть стати клієнтами, шукатимуть знаки від продавців, які також можуть допомогти підвищити їх довіру до придбання продукту чи послуги онлайн [7].

Веб-застосунок для цифрової дистрибуції відеоігор являє собою спеціалізований інтернет-магазин, який належить фірмі-товаровиробнику, торговій фірмі тощо та призначений для дистрибуції цифрових товарів на ринку, збільшення обсягів продажу, залучення нових покупців.

1.2 Змістовний опис і аналіз предметної області

Характерними рисами веб-застосунків для цифрової дистрибуції є те, що вони можуть пропонувати значно більшу кількість товарів та послуг, ніж реальні магазини і забезпечувати споживачів значно більшим обсягом інформації, необхідної для прийняття рішення про покупку.

Також завдяки використанню Інтернет-технологій є можливою персоналізація підходу до споживачів з врахуванням попередніх відвідувань каталогу, зроблених покупок та використання інтернет-магазину як ефективного способу маркетингових досліджень (анкетування, конференції покупців і т.д.).

Дизайн веб-застосунку може застаріти і навіть стати відображенням пам'яті поколінь маркетологів компанії.

Тренди оновлюються щорічно, і веб-застосунок повинен йти в ногу з часом. І це не обов’язково мають бути химерні елементи чи градієнти по всьому сайту – це набагато простіше. Вимоги до дизайну сторінок змінюються, користувачі звикають до певного розташування елементів і навігації.

Не завжди легко пояснити, чому сайт виглядає архаїчно, але більшість користувачів швидко схоплюють застарілий дизайн. Довгі періоди

очікування, перевантажена структура, відволікаючий фон, нечитабельні шрифти, багато тексту, використання дрібних зображень і неприємних оку кольорів – все це відволікає і призводить до єдиного бажання швидко закрити вкладку в браузері.

Більшість відвідувачів сайту спочатку оцінять довіру до бізнесу виключно на основі його дизайну та змісту. Тренди веб-дизайну змінюються кожні кілька років, тому в середньому, сайт, який прослужив власнику більше 3 років, однозначно вартий оновлення.

Також, очікування людей щодо швидшого та кращого цифрового досвіду зростають, і користувачі стають дедалі мінливішими. У веб-застосунку є лише кілька секунд, щоб налаштувати потенційного користувача на те, щоб він став клієнтом. Якщо цього не зроблено, бізнес втратить довічну цінність цього потенційного клієнта.

Застарілі веб-сайти створюють негативний досвід взаємодії з брендом, що призводить людей мати на 62% менше шансів знову купувати на цій платформі [8].

Якщо веб-застосунок завантажується надто повільно, користувачі залишать його заради конкурента. Шляхом покращення ситуації є оптимізація швидкості сторінки та оптимізація конверсій, що допоможуть залучити більше потенційних клієнтів і збільшити продажі онлайн.

В рамках дипломного проєкту обрано пошвидшення взаємодії та покращення досвіду користувача з веб-застосунком шляхом оптимізації керування станом додатку.

1.3 Аналіз існуючих технологій та успішних ІТ-проєктів

Проаналізуємо відоме на сьогодні алгоритмічне забезпечення у даній області та технічні рішення, що допоможуть у реалізації оптимізованого керування станом веб-застосунку. Далі будуть розглянуті допоміжні програмні засоби, засоби розробки та готові програмні рішення.

1.3.1 Аналіз відомих алгоритмічних та технічних рішень

Стан додатку поєднує дані з будь-якого типу серверної або локальної зміни та представлення цих даних за допомогою елементів інтерфейсу користувача. Стан може синхронізувати дані різних компонентів, оскільки кожне оновлення стану повторно відтворює всі відповідні компоненти. Стан також може бути засобом комунікації між різними компонентами.

Стан інтерфейсу користувача означає те, що бачить користувач у будь-який момент часу. Стан змінюється, коли користувач взаємодіє з веб-застосунком, переміщаючись ним та надаючи дані. Добре керований стан допомагає створити більш плавний, узгоджений, продуктивний і приємний досвід користувача.

На сьогоднішній день актуальним є впровадження алгоритмів керування станом додатку, які, швидше за все, підтримуються набором інструментів розробки, зібраних у бібліотеці з керування станом додатку.

Такі бібліотеки забезпечують спільну роботу імплементацій керування станом на front-end та back-end, щоб успішно отримати та забезпечити розуміння та контроль застосунку. Бібліотеки керування станом пришвидшують та полегшують підтримку коду керування станом завдяки застосуванню загальних практик. Існує два типи бібліотек управління станом: атомарні та глобальні.

Атомарну бібліотеку для React можна уявити як 3-й вимір дерева компонентів, який є перпендикулярним до існуючих вузлів. За допомогою селекторів, до цих атомів можна отримати прямий доступ через окремі вузли без необхідності проходження дерева React, що робить цей спосіб швидшим. Вони використовують методи, схожі на React Hooks.

Популярні бібліотеки атомарного стану – Jotai [9] та Recoil [10]. Обидві підходять для простих рішень та легкої інтеграції, але мають проблеми з недостатньою документацією, збиранням сміття та витоками пам’яті.

Глобальні Stores дотримуються підходу архітектури FLUX: Action → Dispatcher → Store → View. Ці бібліотеки мають повністю незалежні файли керування Stores у проєкті, які працюють разом з рештою додатку, щоб забезпечити найкраще рішення для керування станом. Коли користувач взаємодіє з React View, View надсилає Action (зазвичай представлену як об’єкт JavaScript із деякими полями) через Dispatcher, який сповіщає різні Stores, які зберігають дані додатку та бізнес-логіку. Коли Store змінює стан, він сповіщує View про те, що щось оновлено. Це особливо добре працює з декларативною моделлю React, яка дозволяє Store надсилати оновлення, не вказуючи, як перемикати Views між станами.

Популярні бібліотеки з глобальним Store станом – Redux [11] та Zustand [12]. Обидві бібліотеки мають гнучкий функціонал. Бібліотека Zustand є простішою та стислішою за Redux, але має недостатню документацію. Redux використовується багатьма розробниками, тому має багато документації, ресурсів та інструментів стосовно різноманітних проблем і рішень.

Хоча модель Redux існує вже деякий час, її найбільшою проблемою є велика кількість шаблонного коду. Написання Actions для опису всіх можливих змін стану, а потім кількох Reducers для обробки цих дій може призвести до великої кількості коду, який стає важко підтримувати. Ось чому було створено Redux Toolkit.

Сьогодні Redux Toolkit є найпопулярнішим способом використання Redux. Він спрощує налаштування Store, зменшує необхідну кількість шаблонного коду та дотримується найкращих загальних практик за замовчуванням. Крім того, він поставляється разом з усіма можливими

частинами, необхідними для повноцінного використання, наприклад, Immer для легкої зміни стану та Redux-Thunk для роботи з асинхронною логікою.

Таким чином, для оптимізації керування станом додатку обрано Redux та Redux Toolkit відповідно.

Архітектура клієнт-сервер є одним із архітектурних шаблонів програмного забезпечення, що розподіляє завдання або робочі навантаження між постачальниками ресурсів або сервісів, які називаються серверами, і запитувачами сервісів, які називаються клієнтами [13]. Часто клієнти та сервери спілкуються через комп'ютерну мережу на окремому обладнанні, але і клієнт, і сервер можуть знаходитися в одній системі. Хост сервера запускає одну або кілька серверних програм, які спільно використовують свої ресурси з клієнтами. Клієнт зазвичай не ділиться жодними своїми ресурсами, але запитує вміст або послугу на сервері. Тому клієнти ініціюють сеанси зв'язку з серверами, які очікують вхідних запитів.

Загалом, сервіс є абстракцією ресурсів комп'ютера, і клієнту не потрібно турбуватися про те, як працює сервер під час виконання запиту та доставки відповіді. Клієнт має лише зрозуміти відповідь на основі загальновідомого протоколу програми, тобто зміст і форматування даних для запиту.

Сервер може отримувати запити від багатьох різних клієнтів за короткий проміжок часу. Комп'ютер може виконувати лише обмежену кількість завдань у будь-який момент і покладається на систему планування, щоб визначити пріоритетність вхідних запитів від клієнтів для їх виконання.

Трирівнева клієнт-серверна архітектура передбачає відділення прикладного рівня від управління даними. Інтерфейс користувача (рівень представлення), логіка функціональних процесів (рівень бізнес-логіки) та комп'ютерне зберігання даних (рівень доступу до даних) розробляються та підтримуються як незалежні модулі, найчастіше на окремих платформах [14].

У порівнянні з клієнт-серверною або файл-серверною архітектурою, трирівнева архітектура додатку забезпечує модель, за допомогою якої розробники можуть створювати гнучкі веб-застосунки, які можна легко масштабувати та багаторазово використовувати. Завдяки ізоляції рівнів поділом, розробники отримують можливість конфігурувати або додавати лише певний рівень замість того, щоб переробляти всю програму. Таким чином, для розроблюваного веб-застосунку обрано трирівневу архітектуру.

1.3.2 Аналіз допоміжних програмних засобів та засобів розробки

В ході аналізу засобів розробки обрано такі інструменти:

Visual Studio Code обрано як лідера у списку найкращих IDE для веб-розробки [15]. Легкий, але потужний редактор працює на Windows, Mac і Linux. Він підтримує широкий спектр мов, серед яких є JavaScript, TypeScript. Крім того, Visual Studio Code має багату екосистему розширень для підтримки розробки на платформі .NET та мовою програмування C#.

.NET, оскільки це перспективна безкоштовна платформа розробки програмного забезпечення з відкритим вихідним кодом для операційних систем Windows, Linux і macOS [16]. Це кросплатформний наступник .NET Framework. Відповідно платформі обраною мовою програмування серверної частини є C#.

ASP.NET Web API обрано як фреймворк, який дозволяє легко розробляти HTTP-сервіси для доступу до клієнтських об’єктів, таких як браузери, пристрої або планшети [17]. Web API надає доступ до повного функціоналу HTTP, наприклад заголовки запитів/відповідей, форматування вмісту, кешування, URI тощо, і, порівняно з рештою служб WCF (які вимагають визначення додаткових параметрів конфігурації для різних

пристроїв), набагато зручніше розробляти веб-додатки ASP.NET за допомогою RESTful веб-служб використовуючи Web API.

Entity Framework Core обрано як ORM фреймворк, який забезпечує швидшу розробку, прості та швидкі операції з базою даних, може бути дуже корисним і може допомогти у багатьох завданнях, але він може бути не найкращим варіантом для великих операцій з базами даних, які слід залишити для самого сервера бази даних [18].

React – це бібліотека JavaScript, розроблена Facebook, яка, серед іншого, використовувалася для створення Instagram.com. Цю бібліотеку обрано завдяки тому, що вона має на меті дозволити розробникам легко створювати швидкі користувацькі інтерфейси для веб-застосунків [19].

Основною концепцією React.js є віртуальний DOM. Це дерево на основі компонентів JavaScript, створених за допомогою React, яке імітує дерево DOM. Воно виконує найменшу кількість маніпуляцій з DOM, щоб підтримувати компоненти React в актуальному стані.

Будучи частиною мови JavaScript, використання React дає багато переваг. Продукти, створені за допомогою React, легко масштабувати, є шаблони робочого процесу для зручної командної роботи, код інтерфейсу користувача легко читається та підтримується тощо. Провідні світові компанії використовують React та технології надбудованою над JavaScript мовою TypeScript у багатьох продуктах, що визначають ринок.

PostgreSQL обрано як одну з найпопулярніших і добре оцінених у світі реляційних баз даних з відкритим кодом [20]. Цінуючи стабільність, функціональність і відповідність стандартам, PostgreSQL підходить для багатьох проєктів. Подібним чином, якщо веб-застосунку потрібна гнучкість у представленні даних або треба використовувати різні інструменти та мови, PostgreSQL також буде хорошим вибором.

1.3.3 Аналіз відомих програмних продуктів

Відомими аналогами даного дипломного проєкту є веб-застосунки інтернет-магазинів, супроводжуючі такі системи сервісів цифрової дистрибуції ігор як Steam [21], Epic Games Store (EGS) [22] та GOG [3].

Для порівняння особливостей та функціоналу розроблюваного веб-застосунку з аналогами можна скористатись таблицею 1.1.

Таблиця 1.1 – Порівняння з аналогами

Особливості/ Функціонал	**Дипломний проєкт (DD Web App)**	**Steam**	**EGS**	**GOG**
Односторінковий застосунок	+	-	+	-
Оптимізоване керування станом додатку	+	-	-	-
Велика кількість продуктів	-	+	-	+
Відгуки користувачів про продукти	-	+	-	+
Сучасний технологічний стек	+	-	+	-
Реєстрація та авторизація	+	+	+	+
Зручний перегляд каталогу	+	+	-	+
Додавання продуктів до списку бажань	-	+	+	+
Додавання продуктів до кошику	+	+	+	+

1.4 Аналіз вимог до програмного забезпечення

Головною функцією програмного забезпечення є перегляд каталогу продуктів, більше функцій можна побачити на рисунку 1.1.

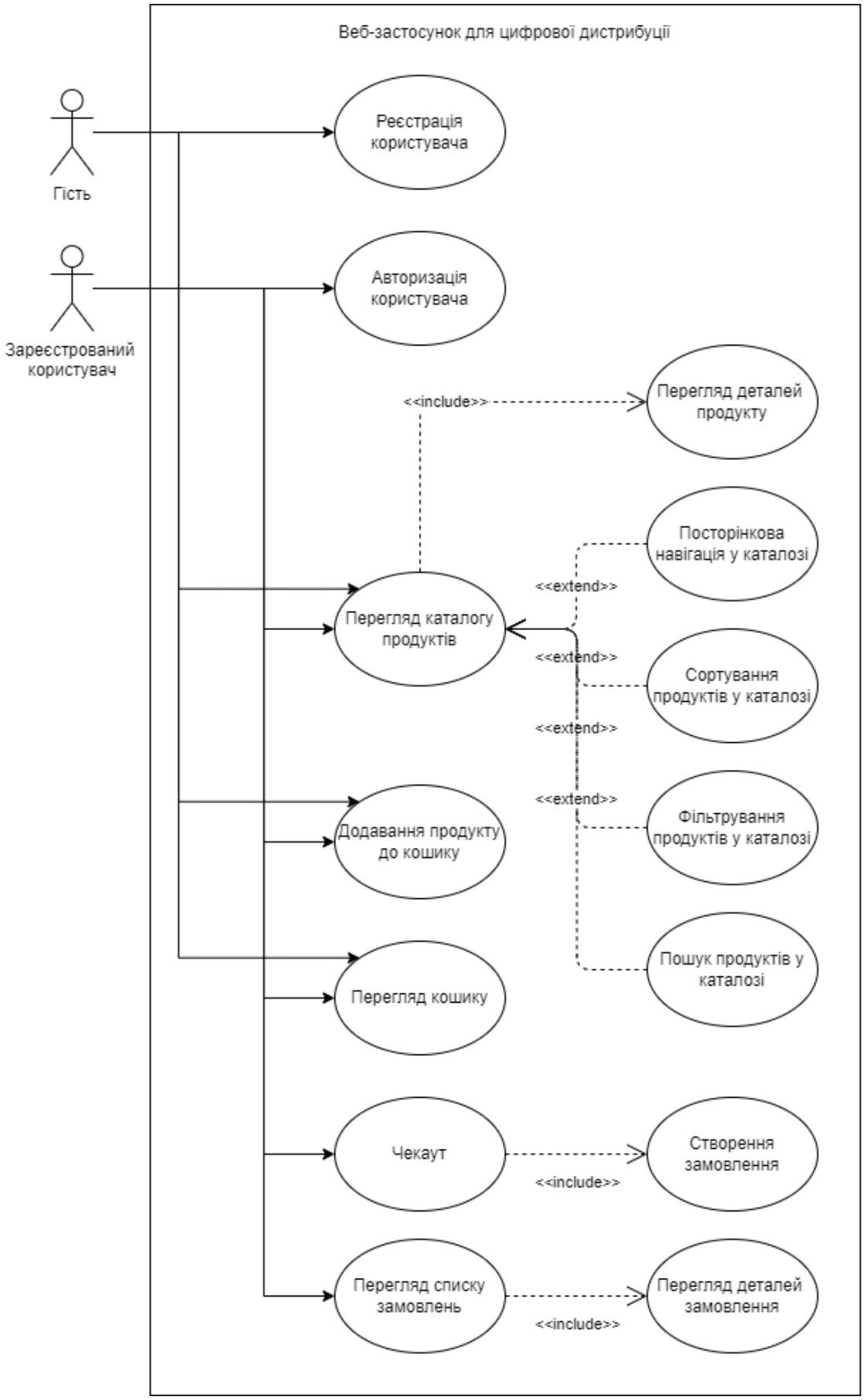


Рисунок 1.1 – Діаграма варіантів використання

В таблицях 1.2 – 1.15 наведені основні варіанти використання програмного забезпечення.

Таблиця 1.2 - Варіант використання UC-1

Use case name	Реєстрація користувача
Use case ID	UC-1
Goals	Реєстрація нового користувача в системі
Actors	Гість (незареєстрований користувач)
Trigger	Користувач бажає зареєструватися
Pre-conditions	-
Flow of Events	Користувач переходить на сторінку реєстрації. В поля для реєстрації вводяться відповідні дані: пошта користувача, ім’я користувача та пароль в системі. Після заповнення даних користувач натискає кнопку реєстрації. Після цього з’являється повідомлення про успішну реєстрацію, і користувач перенаправляється на сторінку входу.
Extension	У випадку введення некоректних даних, кнопка реєстрації залишається неактивною. Якщо якесь конкретне поле введено некоректно, то воно підсвічується червоним надписом.
Post-Condition	Створення облікового запису користувача, перехід на сторінку входу

Таблиця 1.3 - Варіант використання UC-2

Use case name	Авторизація користувача
Use case ID	UC-2
Goals	Авторизація користувача в системі
Actors	Зареєстрований користувач
Trigger	Користувач бажає авторизуватися
Pre-conditions	Користувач має дані для входу в обліковий запис
Flow of Events	Користувач переходить на сторінку авторизації. В поля для авторизації вводяться відповідні дані: ім’я користувача та пароль в системі. Після заповнення даних користувач натискає кнопку авторизації. Після цього користувач перенаправляється на сторінку каталогу.
Extension	У випадку введення некоректних даних користувач отримує повідомлення помилки Unauthorized.
Post-Condition	Користувача авторизовано, перехід на сторінку каталогу

Таблиця 1.4 - Варіант використання UC-3

Use case name	Перегляд каталогу продуктів
Use case ID	UC-3
Goals	Користувач може переглянути каталог продуктів
Actors	Гість, Зареєстрований користувач

Продовження таблиці 1.4

Trigger	Користувач бажає переглянути каталог продуктів
Pre-conditions	-
Flow of Events	Користувач переходить на сторінку каталогу. Веб-застосунок відображає каталог.
Extension	При завантаженні продуктів відображається компонент з індикатором завантаження.
Post-Condition	Завантажено та відображено продукти в сітці каталогу

Таблиця 1.5 - Варіант використання UC-4

Use case name	Перегляд деталей продукту
Use case ID	UC-4
Goals	Користувач може переглянути деталі продукту
Actors	Гість, Зареєстрований користувач
Trigger	Користувач бажає переглянути деталі продукту
Pre-conditions	Користувач знаходиться на сторінці каталогу або вже має посилання на сторінку продукту
Flow of Events	Користувач переходить на сторінку деталей продукту. Веб-застосунок відображає деталі продукту.

Продовження таблиці 1.5

Extension	У випадку переходу на сторінку неіснуючого продукту відображається помилка 404. У випадку вибору нуля продуктів для додавання в кошик кнопка додавання в кошик стає неактивною.
Post-Condition	Завантажено та відображено деталі продукту

Таблиця 1.6 - Варіант використання UC-5

Use case name	Посторінкова навігація у каталозі
Use case ID	UC-5
Goals	Користувач може використати посторінкову навігацію у каталозі
Actors	Гість, Зареєстрований користувач
Trigger	Користувач бажає використати посторінкову навігацію у каталозі
Pre-conditions	Користувач знаходиться на сторінці каталогу
Flow of Events	Користувач обирає сторінку, натискаючи на номер сторінки або на кнопки навігації вперед або назад. Після цього користувачу відображається обрана сторінка каталогу.
Extension	Кнопки навігації вперед або назад стають неактивними якщо відсутня попередня та/або наступна сторінки відносно поточної.
Post-Condition	Відображено обрану сторінку каталогу

Таблиця 1.7 - Варіант використання UC-6

Use case name	Сортування продуктів у каталозі
Use case ID	UC-6
Goals	Користувач може використати сортування продуктів у каталозі
Actors	Гість, Зареєстрований користувач
Trigger	Користувач бажає використати сортування продуктів у каталозі
Pre-conditions	Користувач знаходиться на сторінці каталогу
Flow of Events	Користувач обирає спосіб сортування, натискаючи на радіокнопки зі способами сортування. Після цього користувачу відображається відсортована сторінка каталогу.
Extension	-
Post-Condition	Відображено відсортовану сторінку каталогу

Таблиця 1.8 - Варіант використання UC-7

Use case name	Фільтрування продуктів у каталозі
Use case ID	UC-7
Goals	Користувач може використати фільтрування продуктів у каталозі

Продовження таблиці 1.8

Actors	Гість, Зареєстрований користувач
Trigger	Користувач бажає використати фільтрування продуктів у каталозі
Pre-conditions	Користувач знаходиться на сторінці каталогу
Flow of Events	Користувач обирає спосіб фільтрування, натискаючи на прапорці зі способами фільтрування. Після цього користувачу відображається відфільтрована сторінка каталогу.
Extension	-
Post-Condition	Відображено відфільтровану сторінку каталогу

Таблиця 1.9 - Варіант використання UC-8

Use case name	Пошук продуктів у каталозі
Use case ID	UC-8
Goals	Користувач може використати пошук продуктів у каталозі
Actors	Гість, Зареєстрований користувач
Trigger	Користувач бажає використати пошук продуктів у каталозі
Pre-conditions	Користувач знаходиться на сторінці каталогу

Продовження таблиці 1.9

Flow of Events	Користувач обирає термін пошуку, вводячи слова у текстове поле. Після цього користувачу відображається сторінка каталогу з результатами пошуку.
Extension	-
Post-Condition	Відображено сторінку каталогу з результатами пошуку

Таблиця 1.10 - Варіант використання UC-9

Use case name	Додавання продукту до кошику
Use case ID	UC-9
Goals	Користувач може додати продукт до свого кошику
Actors	Гість, Зареєстрований користувач
Trigger	Користувач бажає додати продукт до свого кошику
Pre-conditions	Користувач знаходиться на сторінці каталогу або деталей продукту
Flow of Events	Користувач натискає на кнопку додавання в кошик обраного продукту. Після цього значок кошику відображає загальну кількість продуктів у кошику.
Extension	Знаходячись на сторінці деталей продукту, користувач має змогу використовувати числове меню введення для зміни кількості продукту в кошику.
Post-Condition	Продукт додано до кошику

Таблиця 1.11 - Варіант використання UC-10

Use case name	Перегляд кошику
Use case ID	UC-10
Goals	Користувач може переглянути вміст свого кошику
Actors	Гість, Зареєстрований користувач
Trigger	Користувач бажає переглянути вміст свого кошику
Pre-conditions	-
Flow of Events	Користувач переходить на сторінку кошику. Веб-застосунок відображає кошик.
Extension	Якщо користувач ще не додавав продукти до кошику, відображається інформація про відсутність продуктів в кошику. Користувач має змогу використовувати кнопки додавання, віднімання та видалення продуктів з кошику для відповідних дій.
Post-Condition	Відображено вміст кошику користувача

Таблиця 1.12 - Варіант використання UC-11

Use case name	Чекаут
Use case ID	UC-11
Goals	Користувач може виконати чекаут

Продовження таблиці 1.12

Actors	Зареєстрований користувач
Trigger	Користувач бажає виконати чекаут
Pre-conditions	Користувач авторизувався та знаходиться на сторінці кошику з продуктами
Flow of Events	Користувач впевнюється у вмісті та сумі кошику, після чого переходить на сторінку чекауту.
Extension	-
Post-Condition	Відображено сторінку чекауту

Таблиця 1.13 - Варіант використання UC-12

Use case name	Створення замовлення
Use case ID	UC-12
Goals	Користувач може створити замовлення
Actors	Зареєстрований користувач
Trigger	Користувач бажає створити замовлення
Pre-conditions	Користувач авторизувався та знаходиться на сторінці чекауту

Продовження таблиці 1.13

Flow of Events	Користувачу відображається звіт замовлення з інформацією про обрані продукти, їх ціну та спосіб доставки. Користувач обирає метод оплати з випадаючого меню та натискає кнопку створення замовлення.
Extension	В випадку відсутності обраного методу оплати користувачеві відображається повідомлення про відсутній метод оплати.
Post-Condition	Створено замовлення та відображено його номер

Таблиця 1.14 - Варіант використання UC-13

Use case name	Перегляд списку замовлень
Use case ID	UC-13
Goals	Користувач може переглянути свій список замовлень
Actors	Зареєстрований користувач
Trigger	Користувач бажає переглянути свій список замовлень
Pre-conditions	Користувач авторизувався
Flow of Events	Користувач переходить на сторінку списку замовлень. Веб-застосунок відображає список замовлень.
Extension	При завантаженні замовлень відображається компонент з індикатором завантаження.
Post-Condition	Завантажено та відображено замовлення в списку замовлень

Таблиця 1.15 - Варіант використання UC-14

Use case name	Перегляд деталей замовлення
Use case ID	UC-14
Goals	Користувач може переглянути деталі свого замовлення
Actors	Зареєстрований користувач
Trigger	Користувач бажає переглянути деталі свого замовлення
Pre-conditions	Користувач авторизувався та знаходиться на сторінці списку замовлень
Flow of Events	Користувач переходить на сторінку деталей замовлення. Веб-застосунок відображає деталі замовлення.
Extension	У випадку переходу на сторінку неіснуючого замовлення відображається помилка 404.
Post-Condition	Завантажено та відображено деталі замовлення

1.4.1 Розроблення функціональних вимог

Веб-застосунок розділений на модулі. Кожен модуль має свій певний набір функцій. У таблиці 1.16 наведено загальну модель вимог, а в таблицях 1.17 – 1.30 наведений опис функціональних вимог до програмного забезпечення. Матрицю трасування вимог можна побачити на рисунку 1.2.

Таблиця 1.16 – Модель вимог у загальному вигляді

№	Назва	Код	Пріоритет	Ризик
1	Реєстрація користувача	FR-1	Високий	Високий
2	Авторизація користувача	FR-2	Високий	Високий
3	Перегляд каталогу продуктів	FR-3	Високий	Низький
4	Перегляд деталей продукту	FR-4	Середній	Середній
5	Посторінкова навігація у каталозі	FR-5	Середній	Низький
6	Сортування продуктів у каталозі	FR-6	Середній	Низький
7	Фільтрування продуктів у каталозі	FR-7	Середній	Низький

Продовження таблиці 1.16

8	Пошук продуктів у каталозі	FR-8	Середній	Низький
9	Додавання продукту до кошику	FR-9	Високий	Середній
10	Перегляд кошику	FR-10	Високий	Середній
11	Чекаут	FR-11	Високий	Високий
12	Створення замовлення	FR-12	Високий	Високий
13	Перегляд списку замовлень	FR-13	Низький	Середній
14	Перегляд деталей замовлення	FR-14	Низький	Середній

Таблиця 1.17 – Функціональна вимога FR-1

Назва	Реєстрація користувача
Опис	Веб-застосунок повинен надавати можливість реєстрації користувача шляхом введення пошти користувача, імені користувача та паролю в системі, з використанням централізованої логіки керування станом облікового запису.

Таблиця 1.18 – Функціональна вимога FR-2

Назва	Авторизація користувача
Опис	Веб-застосунок повинен надавати можливість авторизації користувача шляхом введення імені користувача та паролю в системі, з використанням централізованої логіки керування станом облікового запису.

Таблиця 1.19 – Функціональна вимога FR-3

Назва	Перегляд каталогу продуктів
Опис	Веб-застосунок повинен надавати можливість переглянути каталог продуктів, використовуючи централізовану логіку керування станом каталогу.

Таблиця 1.20 – Функціональна вимога FR-4

Назва	Перегляд деталей продукту
Опис	Веб-застосунок повинен надавати можливість переглянути деталі продукту, використовуючи централізовану логіку керування станом каталогу.

Таблиця 1.21 – Функціональна вимога FR-5

Назва	Посторінкова навігація у каталозі
Опис	Веб-застосунок повинен надавати можливість використати посторінкову навігацію у каталозі, використовуючи централізовану логіку керування станом каталогу.

Таблиця 1.22 – Функціональна вимога FR-6

Назва	Сортування продуктів у каталозі
Опис	Веб-застосунок повинен надавати можливість використати сортування продуктів у каталозі, використовуючи централізовану логіку керування станом каталогу.

Таблиця 1.23 – Функціональна вимога FR-7

Назва	Фільтрування продуктів у каталозі
Опис	Веб-застосунок повинен надавати можливість використати фільтрування продуктів у каталозі, використовуючи централізовану логіку керування станом каталогу.

Таблиця 1.24 – Функціональна вимога FR-8

Назва	Пошук продуктів у каталозі
Опис	Веб-застосунок повинен надавати можливість використати пошук продуктів у каталозі, використовуючи централізовану логіку керування станом каталогу.

Таблиця 1.25 – Функціональна вимога FR-9

Назва	Додавання продукту до кошику
Опис	Веб-застосунок повинен надавати можливість додати продукт до кошику, використовуючи централізовану логіку керування станом кошику.

Таблиця 1.26 – Функціональна вимога FR-10

Назва	Перегляд кошику
Опис	Веб-застосунок повинен надавати можливість переглянути кошик, використовуючи централізовану логіку керування станом кошику.

Таблиця 1.27 – Функціональна вимога FR-11

Назва	Чекаут
Опис	Веб-застосунок повинен надавати можливість виконати чекаут, використовуючи централізовану логіку керування станом замовлень.

Таблиця 1.28 – Функціональна вимога FR-12

Назва	Створення замовлення
Опис	Веб-застосунок повинен надавати можливість створити замовлення, використовуючи централізовану логіку керування станом замовлень.

Таблиця 1.29 – Функціональна вимога FR-13

Назва	Перегляд списку замовлень
Опис	Веб-застосунок повинен надавати можливість переглянути свій список замовлень, використовуючи централізовану логіку керування станом замовлень.

Таблиця 1.30 – Функціональна вимога FR-14

Назва	Перегляд деталей замовлення
Опис	Веб-застосунок повинен надавати можливість переглянути деталі свого замовлення, використовуючи централізовану логіку керування станом замовлень.

	FR-1	FR-2	FR-3	FR-4	FR-5	FR-6	FR-7	FR-8	FR-9	FR-10	FR-11	FR-12	FR-13	FR-14
UC-1	+													
UC-2	+	+												
UC-3			+											
UC-4			+	+										
UC-5			+		+									
UC-6			+			+								
UC-7			+				+							
UC-8			+					+						
UC-9			+	+					+					
UC-10										+				
UC-11		+								+	+			
UC-12		+									+	+		
UC-13		+											+	
UC-14		+										+	+	+

Рисунок 1.2 – Матриця трасування вимог

1.4.2 Розроблення нефункціональних вимог

Опис нефункціональних вимог:

- передбачити валідацію введення інформації та захист від некоректних дій користувача;
- забезпечити цілісність інформації в базі даних;
- програмне забезпечення повинно функціонувати на IBM-сумісних персональних комп‘ютерах;
- мінімальна конфігурація технічних засобів:
  - тип процесору: Intel Core i5;
  - об‘єм ОЗП: 4 Гб;
  - підключення до мережі Інтернет зі швидкістю від 20 мегабіт;
- рекомендована конфігурація технічних засобів:
  - тип процесору: Intel Core i7;
  - об‘єм ОЗП: 8 Гб;
  - підключення до мережі Інтернет зі швидкістю від 100 мегабіт;
- програмне забезпечення повинно працювати під управлінням операційних систем сімейства Windows та підтримувати браузер Google Chrome;
- забезпечити підтримку англійської локалізації інтерфейса.

1.5 Постановка задачі

Планується основна задача – забезпечити оптимізоване керування станом React компонентів веб-застосунку, що є результатом розробки програмного забезпечення на дипломний проєкт, шляхом реалізації архітектурного рішення, що спрощує написання логіки Redux, налаштування та централізації стану, підтримує асинхронні запити та нормалізацію даних.

Також для реалізації поставлені такі задачі:

– реєстрація та авторизація користувачів;
– перегляд каталогу та деталей продуктів;
– посторінкова навігація, сортування, фільтрування та пошук продуктів у каталозі;
– додавання продуктів до кошику та перегляд кошику;
– чекаут та створення замовлення;
– перегляд списку та деталей замовлень.

Висновки до розділу

Отже, у даному розділі виконано аналіз вимог до програмного забезпечення:

- викладено загальні положення та наведено загальну теорію по предметній області;
- змістовно описано та проаналізовано предметну область, наведено процес використання знань предметної області у програмному забезпеченні на поточний момент розвитку IT-технологій, недоліки поточного стану речей з предметною областю, вказано та обрано в рамках дипломного проєкту можливі шляхи покращення ситуації з розробками у сфері IT;
- проаналізовано відомі алгоритмічні та технічні рішення, розглянуто та порівняно відомі архітектурні патерни та їх реалізації, серед них обрано доречне технічне рішення для використання у розробці;
- проаналізовано та обрано допоміжні засоби розробки;
- проаналізовано відомі програмні продукти у вигляді порівняльної таблиці;
- проаналізовано вимоги до програмного забезпечення, а саме: надано діаграму та опис варіантів використання, модель функціональних вимог, їх опис та матрицю трасування, опис нефункціональних вимог;
- викладено постановку задачі.

2 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Моделювання та аналіз програмного забезпечення

Для опису основного бізнес процесу завантаження сторінки каталогу веб-застосунку використовується модель BPMN 2.0 (рисунок 2.1).

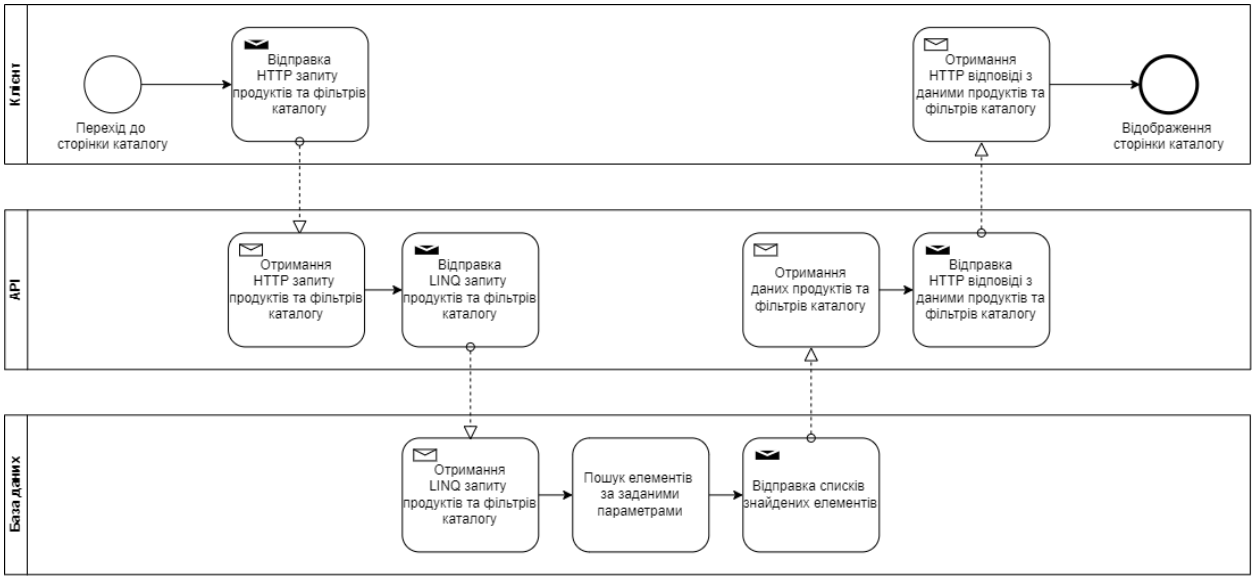


Рисунок 2.1 – Схема бізнес процесу завантаження сторінки каталогу

Опис послідовності завантаження сторінки каталогу:

- користувач переходить до сторінки каталогу;
- клієнт відправляє HTTP запит на сервер щоб отримати дані для заповнення сторінки каталогу;
- сервер отримує HTTP запит клієнта;
- сервер формує та відправляє LINQ запит до структури об'єктно-реляційного відображення з параметрами, заданими клієнтом;
- база даних отримує SQL запит, створений за формою LINQ запиту на сервері;
- база даних шукає необхідні дані та відправляє їх на сервер;

– сервер отримує дані з бази даних та формує з ними HTTP відповідь;
– сервер відправляє клієнту HTTP відповідь з даними;
– клієнт отримує дані з HTTP відповіді сервера;
– клієнт заповнює даними та відображає сторінку каталогу користувачеві.

2.2 Архітектура програмного забезпечення

Для створення веб-застосунку втілено трирівневу архітектуру, що складається з трьох компонентів: рівню представлення, бізнес-логіки та доступу до даних (рисунок 2.2).

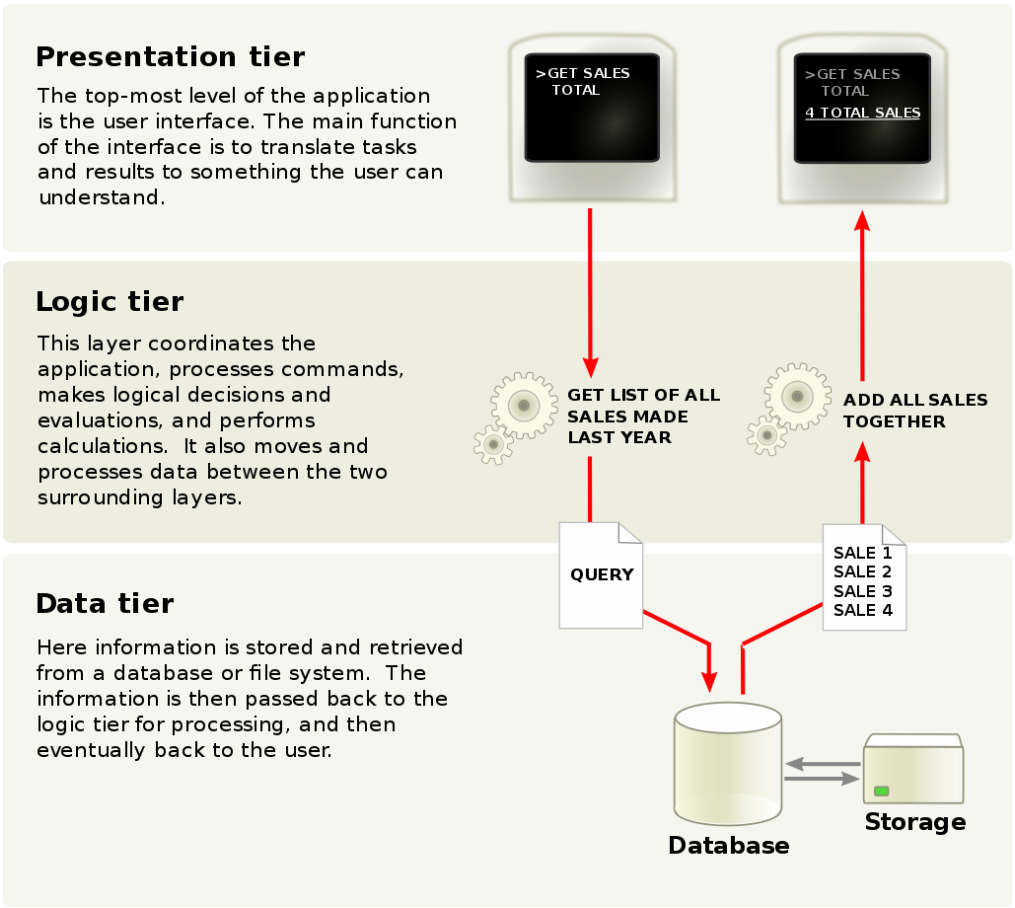


Рисунок 2.2 – Схема трирівневої архітектури [14]

Клієнт – це компонент інтерфейсу, який представляє перший рівень, власне застосунок для кінцевого користувача. На рівні представлення розроблено односторінковий застосунок (SPA), написаний з використанням відкритої JavaScript бібліотеки React [19].

Сервер – це компонент, який розташовується на другому рівні та зосереджує в собі більшу частину бізнес-логіки. На рівні сервера з бізнес-логікою розроблено REST API, написаний з використанням ASP.NET Web API [17].

База даних – це компонент, що забезпечує зберігання даних і виноситься на третій рівень. На рівні доступу до даних розроблено модель даних сутностей з Code First підходом, написану з використанням структури об'єктно-реляційного відображення (ORM) Entity Framework Core [18] та бази даних PostgreSQL [20].

ER діаграма описує взаємопов’язані речі, що представляють інтерес у певній галузі знань. Базова модель ER складається з типів сутностей (які класифікують предмети інтересу) і визначає зв’язки, які можуть існувати між сутностями (екземплярами цих типів сутностей).

У розробці програмного забезпечення модель ER зазвичай створюється для представлення того, що бізнес повинен пам’ятати, щоб виконувати бізнес-процеси. Отже, модель ER стає абстрактною моделлю даних, яка визначає структуру даних або інформації, яка може бути реалізована, як правило, в реляційній базі даних. (рисунок 2.3).

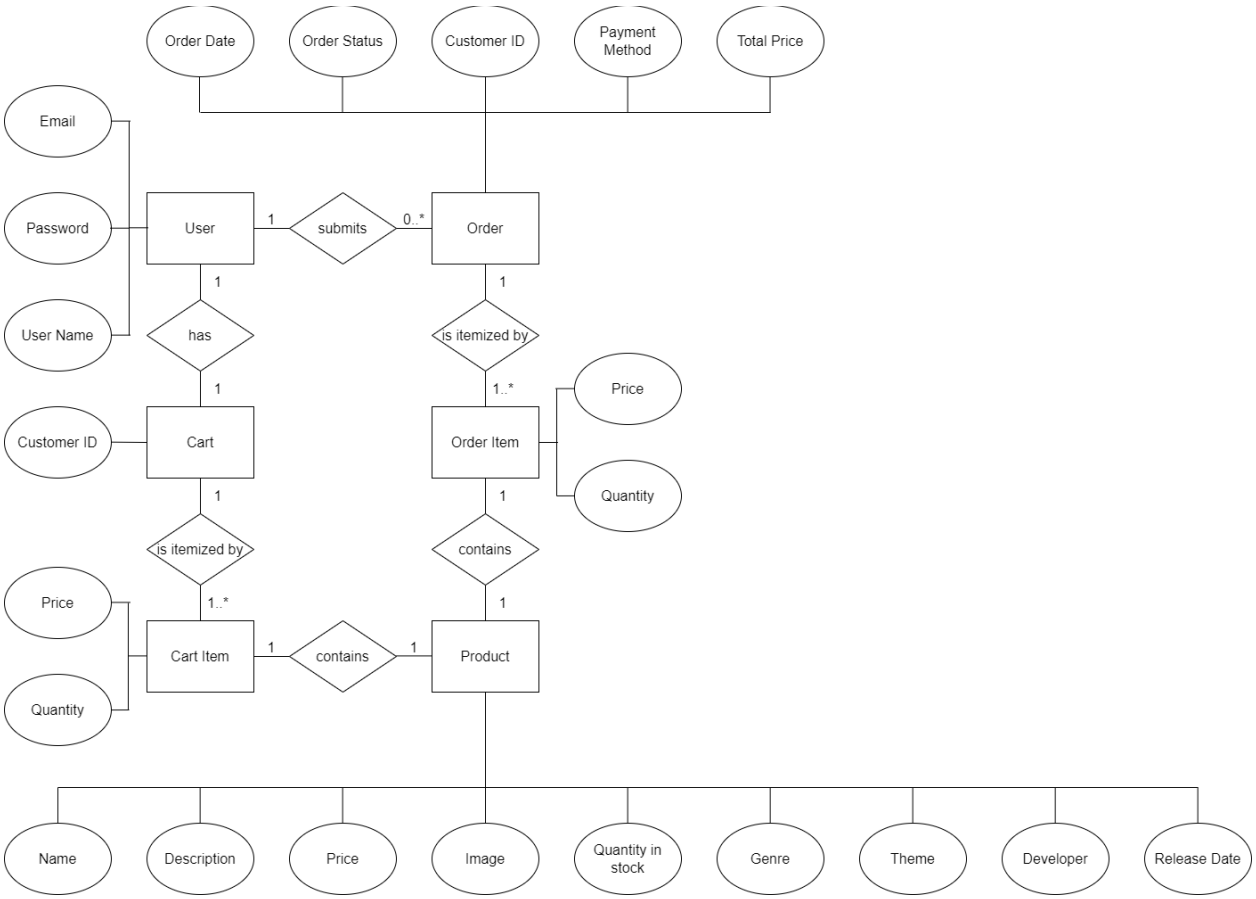


Рисунок 2.3 – ER діаграма сутностей User, Order, Order Item, Cart, Cart Item, Product

В ході аналізу предметної області та створення ER моделі було виділено такі сутності та їх атрибути:

- user;
  - email;
  - password;
  - user name;
- order;
  - order date;
  - order status;
  - customer ID;
  - payment method;
  - total price;
- order item;
  - price;
  - quantity;
- cart;
  - customer ID;
- cart item;
  - price;
  - quantity;
- product;
  - name;
  - description;
  - price;
  - image;
  - quantity in stock;
  - genre;
  - theme;
  - developer;
  - release date.

2.3 Конструювання програмного забезпечення

Опис утиліт, фреймворків, бібліотек, технологій, сервісів та іншого стороннього програмного забезпечення, що використовується у розробці наведено в таблиці 2.1.

Таблиця 2.1 – Опис інструментів

№ п/п	**Назва інструменту**	**Опис застосування**
1	Visual Studio Code	Головне середовище розробки веб-застосунку.
2	PostgreSQL	Безкоштовна відкрита об’єктно-реляційна база даних.
3	DBeaver	Програмне забезпечення яке надає легкий графічний інтерфейс для доступу до бази даних.
4	.NET	Безкоштовна кросплатформна відкрита платформа для створення різноманітного програмного забезпечення мовою програмування C# та іншими.
5	Entity Framework Core	Структура об'єктно-реляційного відображення (ORM) що дозволяє працювати з базою даних за допомогою об’єктів .NET.

Продовження таблиці 2.1

6	ASP.NET Core	Кросплатформний відкритий фреймворк для розробки сучасних веб-застосунків.
7	NuGet	Пакетний менеджер, який в основному використовується для пакування та розповсюдження програмного забезпечення, написаного за допомогою платформи .NET.
8	Swagger	Набір інструментів для розробників API, що допомагає розробникам створювати, документувати, тестувати та використовувати RESTful веб-сервіси. Використовувався для тестування API інтерфейсів та клієнтських запитів.
9	React	Безкоштовна відкрита front-end бібліотека JavaScript для створення інтерфейсу користувача на основі компонентів.
10	npm	Пакетний менеджер, який в основному використовується для пакування та розповсюдження програмного забезпечення, написаного на JavaScript.
11	Material UI	Комплексна бібліотека компонентів інтерфейсу користувача, яка характеризує реалізацію системи Material Design від Google.

Продовження таблиці 1.2

12	Axios	HTTP-клієнт на основі promises для браузера, що дозволяє легко надсилати асинхронні HTTP запити до кінцевих точок REST.
13	Redux	Відкрита JavaScript бібліотека для керування та централізації стану додатку.
14	Redux Toolkit	Офіційний стандартний підхід для написання логіки Redux, що охоплює ядро Redux і містить необхідні йому пакети та функції, використовує передові практики, спрощує більшість завдань Redux, запобігає поширеним помилкам і полегшує написання додатків з Redux.
15	React Router	Легка повнофункціональна бібліотека що дозволяє створити маршрутизацію на стороні клієнта для бібліотеки React.
16	React Hook Form	Бібліотека, яка допомагає перевіряти та керувати складними формами в React.
17	React Toastify	Бібліотека, що дозволяє легко додавати сповіщення до React інтерфейсу веб-застосунку.
18	Redux DevTools	Розширення браузеру для налагодження змін стану програми та покращення процесу розробки з використанням Redux.

Продовження таблиці 1.2

19	Docker	Набір платформ як сервісних продуктів, які використовують віртуалізацію на рівні ОС для доставки програмного забезпечення в пакетах, які називаються контейнерами.
20	Fly.io	Використовується як платформа для розгортання додатку.

База даних PostgreSQL призначена для зберігання даних про користувачів, продукти, кошиків та замовлень. Діаграму структури бази даних створено використовуючи DBeaver та наведено на рисунку 2.4.

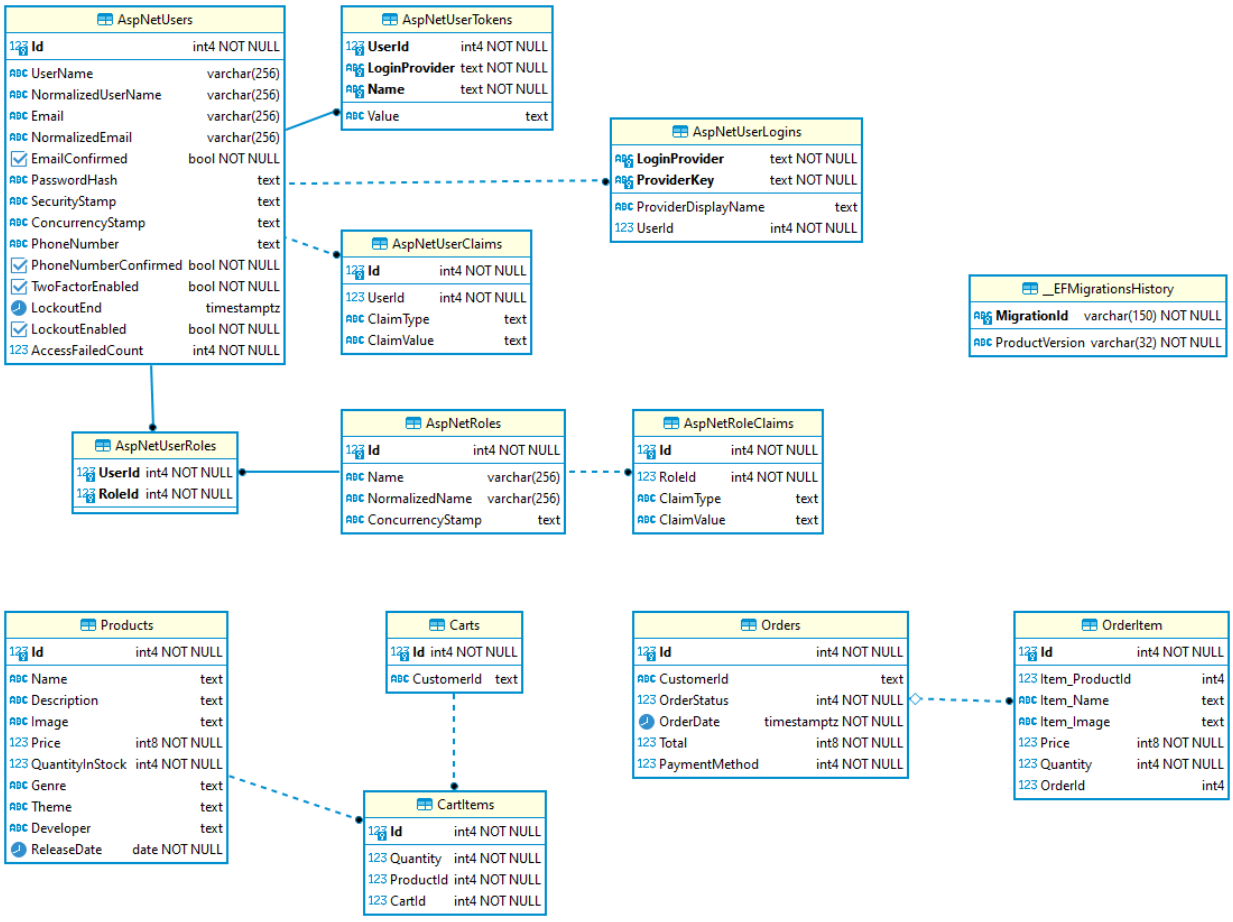


Рисунок 2.4 – Діаграма структури бази даних

Entity Framework Core відстежує, які міграції було застосовано до бази даних, записуючи їх у таблицю під назвою __EFMigrationsHistory.

Оскільки при розробці веб-застосунку використано ASP.NET Core Identity [24], Entity Framework Core генерує у базі даних таблиці для зберігання імен користувачів, паролів, даних профілю, ролей, токенів та ін.

Для роботи з базою даних ASP.NET Identity використовує контекст даних, який успадковується від класу IdentityDbContext із простору імен Microsoft.AspNetCore.Identity.EntityFrameworkCore.

Короткий опис таблиць бази даних наведено у таблиці 2.2.

Таблиця 2.2 – Опис таблиць бази даних

Назва таблиці	**Опис таблиці**
Products	Дані про продукти
Carts	Дані про кошики користувачів
CartItems	Дані про предмети в кошиках користувачів
Orders	Дані про замовлення
OrderItems	Дані про предмети в замовленнях

Продовження таблиці 2.2

AspNetUsers	Набір об'єктів IdentityUser відповідає таблиці користувачів
AspNetUserTokens	Набір об'єктів IdentityUserToken відповідає таблиці токенів аутентифікації користувачів
AspNetUserLogins	Набір об'єктів IdentityUserLogin відповідає таблиці зв'язку користувачів зі своїми логінами зовнішніх сервісів
AspNetUserClaims	Набір об'єктів IdentityUserClaim відповідає таблиці зв'язку користувачів та об'єктів claims користувачів
AspNetUserRoles	Набір об'єктів IdentityUserRole відповідає таблиці, яка зіставляє користувачів та їх ролі
AspNetRoles	Набір об'єктів IdentityRole відповідає таблиці ролей
AspNetRoleClaims	Набір об'єктів IdentityRoleClaim відповідає таблиці зв'язку ролей та об'єктів claims
__EFMigrationsHistory	Історія міграцій Entity Framework Core, що були застосовані до бази даних

Опис полів у таблицях бази даних наведено у таблицях 2.3 - 2.7.

Таблиця 2.3 – Опис таблиці Products

Назва поля	Тип даних	Опис
Id	int4	Ідентифікаційний номер продукту
Name	text	Назва продукту
Description	text	Опис продукту
Image	text	Адреса зображення продукта
Price	int8	Ціна продукту
QuanityInStock	int4	Кількість продуктів в наявності
Genre	text	Жанр продукту
Theme	text	Тематика продукту
Developer	text	Розробник продукту
ReleaseDate	date	Дата випуску продукту

Таблиця 2.4 – Опис таблиці Carts

Назва поля	Тип даних	Опис
Id	int4	Ідентифікаційний номер кошику
CustomerId	text	Унікальний ідентифікатор покупця (GUID отриманий з cookie для гостя, або UserName зареєстрованого користувача)

Таблиця 2.5 – Опис таблиці CartItems

Назва поля	Тип даних	Опис
Id	int4	Ідентифікаційний номер предмета кошику
Quantity	int4	Кількість продуктів у предметі кошику
ProductId	int4	Ідентифікаційний номер продукта
CartId	int4	Ідентифікаційний номер кошику

Таблиця 2.6 – Опис таблиці Orders

Назва поля	Тип даних	Опис
Id	int4	Ідентифікаційний номер замовлення
CustomerId	text	Унікальний ідентифікатор покупця (GUID отриманий з cookie для гостя, або UserName зареєстрованого користувача)
OrderStatus	int4	Статус замовлення
OrderDate	timestamptz	Час створення замовлення з урахуванням часового поясу
Total	int8	Сума замовлення
PaymentMethod	int4	Обраний користувачем спосіб оплати

Таблиця 2.7 – Опис таблиці OrderItems

Назва поля	**Тип даних**	**Опис**
Id	int4	Ідентифікаційний номер предмета замовлення
Item_ProductId	int4	Знімок ідентифікаційного номеру продукта
Item_Name	text	Знімок назви продукта
Item_Image	text	Знімок адреси зображення продукта
Price	int8	Ціна предмета замовлення
Quanity	int4	Кількість продуктів у предметі замовлення
OrderId	int4	Ідентифікаційний номер замовлення

Опис перелічуваних типів даних наведено у таблицях 2.8 - 2.9.

Таблиця 2.8 – Опис перелічуваного типу даних OrderStatus

Значення	**Назва енумератору**	**Опис**
0	Undefined	Статус замовлення не визначено
1	Pending	Статус замовлення в очікуванні
2	Completed	Статус виконаного замовлення
3	Failed	Статус невдалого замовлення

Таблиця 2.9 – Опис перелічуваного типу даних PaymentMethod

Значення	**Назва енумератору**	**Опис**
0	Undefined	Спосіб оплати не визначено
1	PayPal	Користувачем обрано спосіб оплати PayPal
2	Visa	Користувачем обрано спосіб оплати Visa
3	MasterCard	Користувачем обрано спосіб оплати MasterCard

На рисунку 2.5 наведено загальну структуру backend проєкту ASP.NET Web API веб-застосунку.

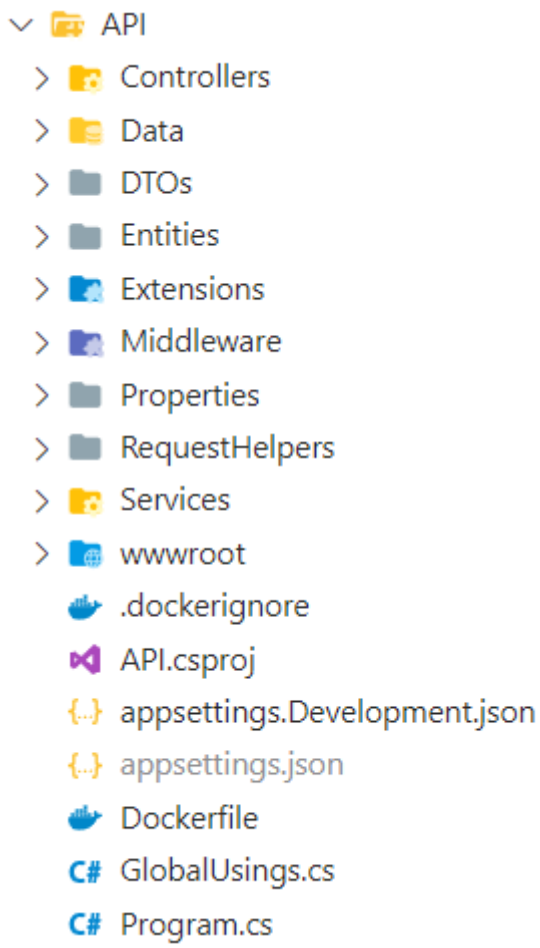


Рисунок 2.5 – Загальна програмна структура серверної частини

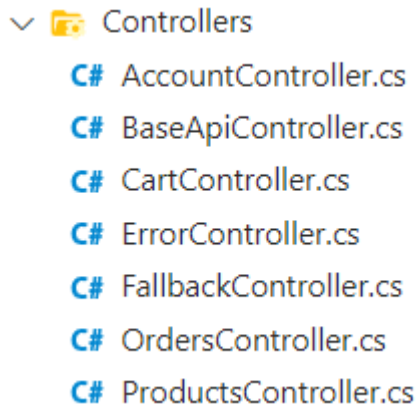


Рисунок 2.6 – Класи Controllers

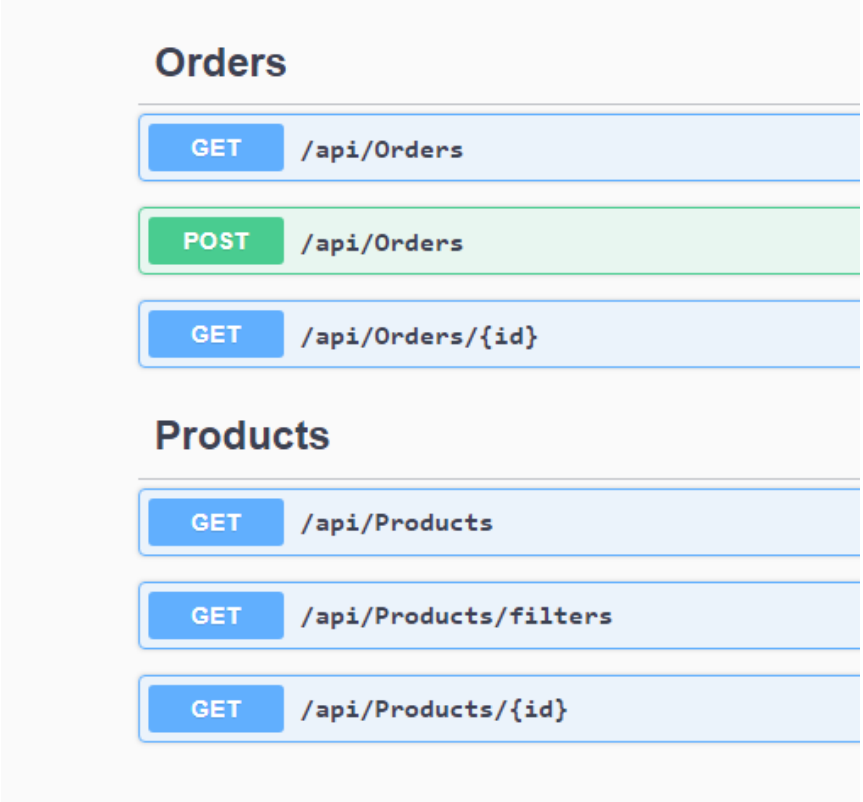


Рисунок 2.7 – API виклики для Orders, Products контролерів

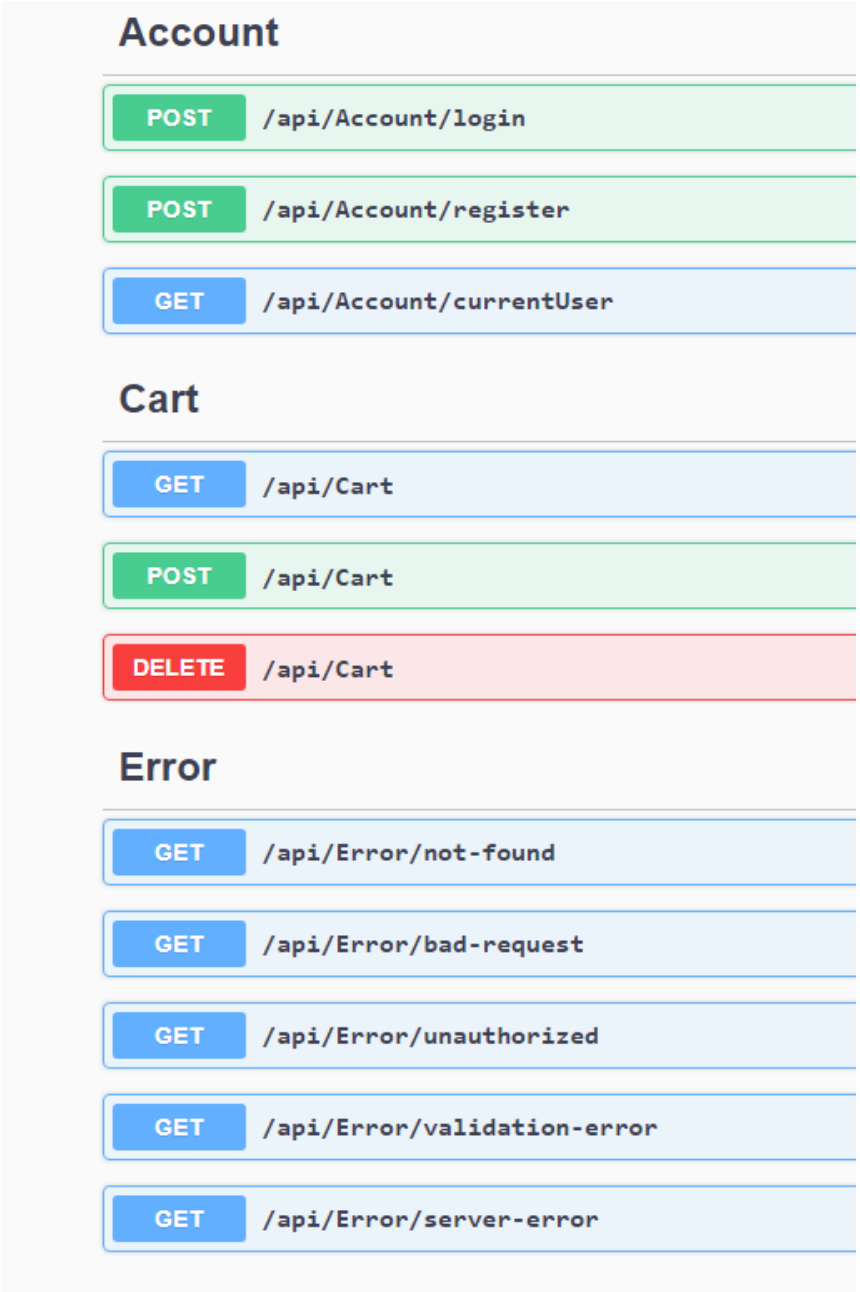


Рисунок 2.8 – API виклики для Account, Cart, Error контролерів

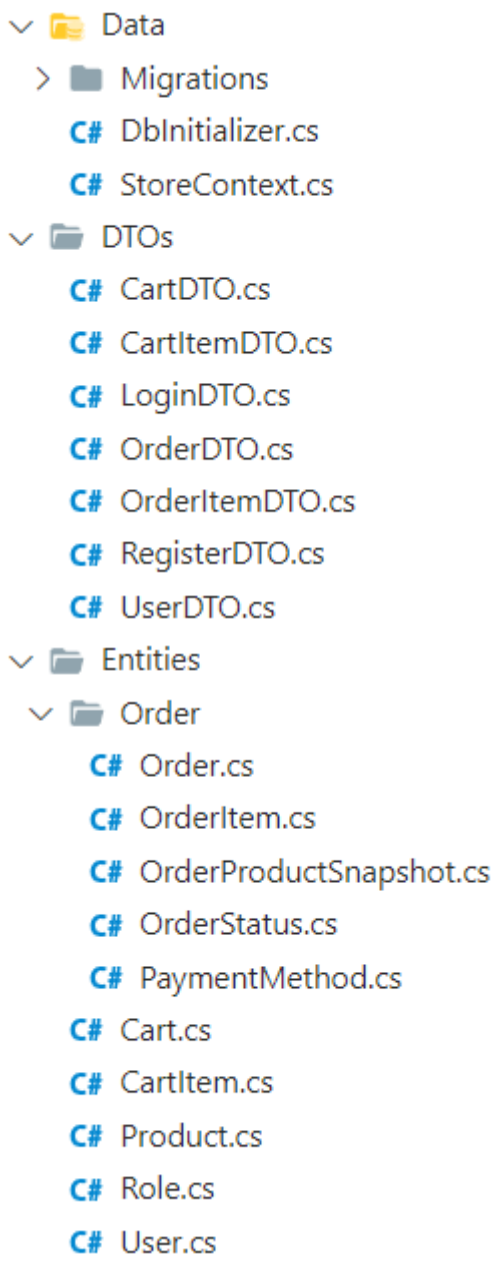


Рисунок 2.9 – Класи Data, DTOs, Entities

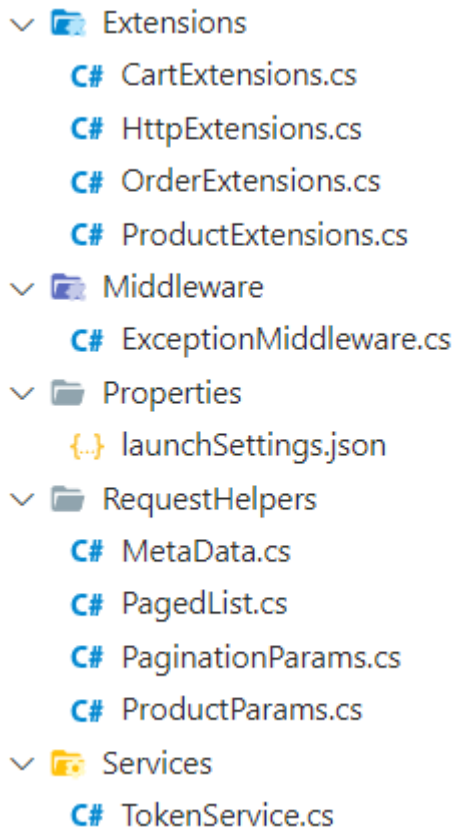


Рисунок 2.10 – Класи Extensions, Middleware, RequestHelpers, Services

На рисунку 2.11 наведено загальну структуру frontend проєкту React веб-застосунку.

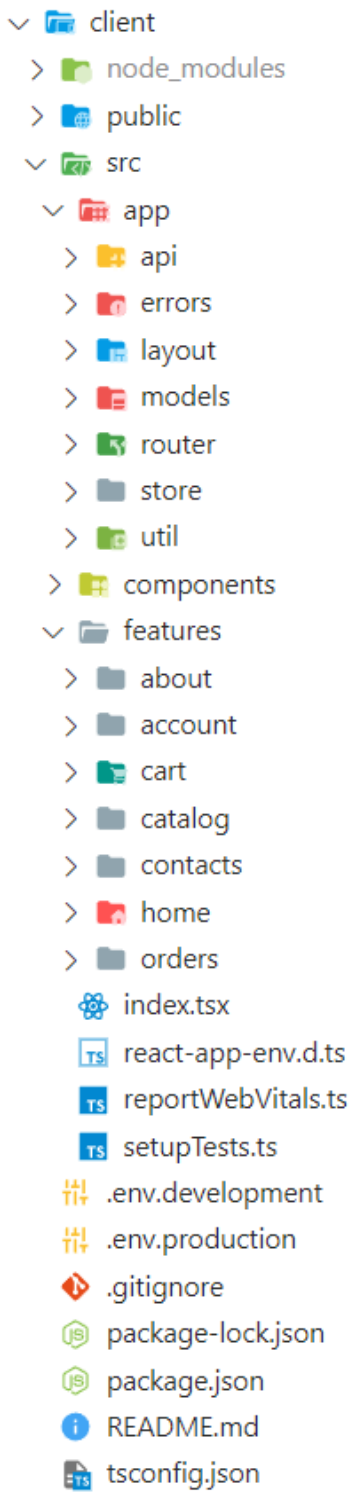


Рисунок 2.11 – Загальна програмна структура клієнтської частини

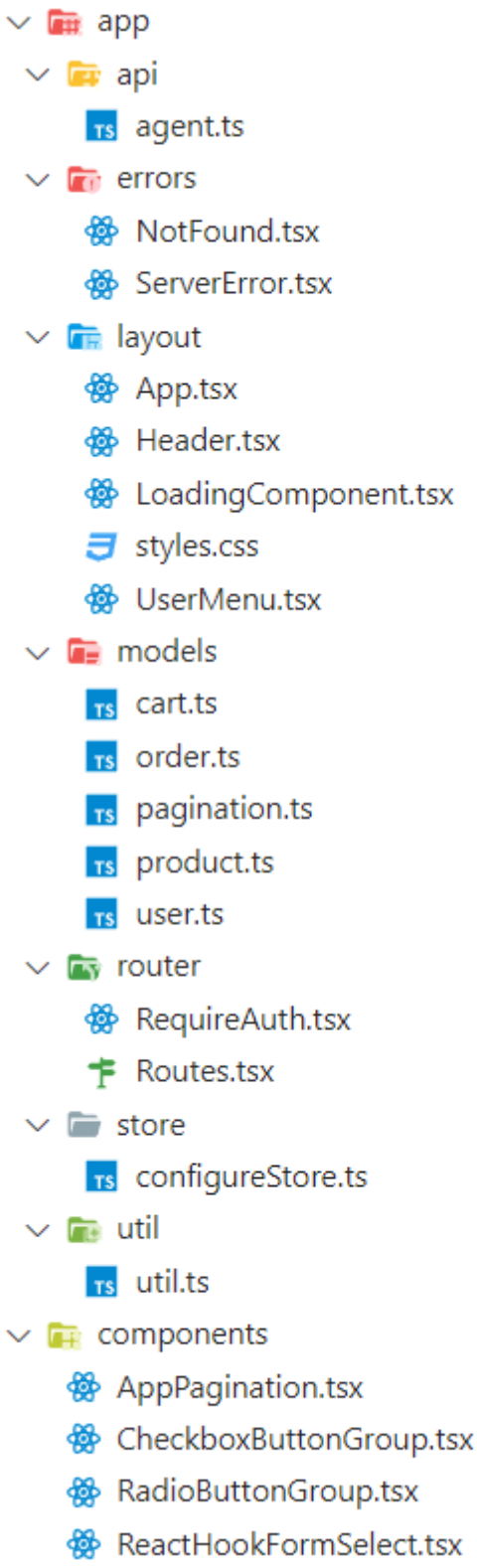


Рисунок 2.12 – Файли app: api, errors, layout, models, router, store, util, components

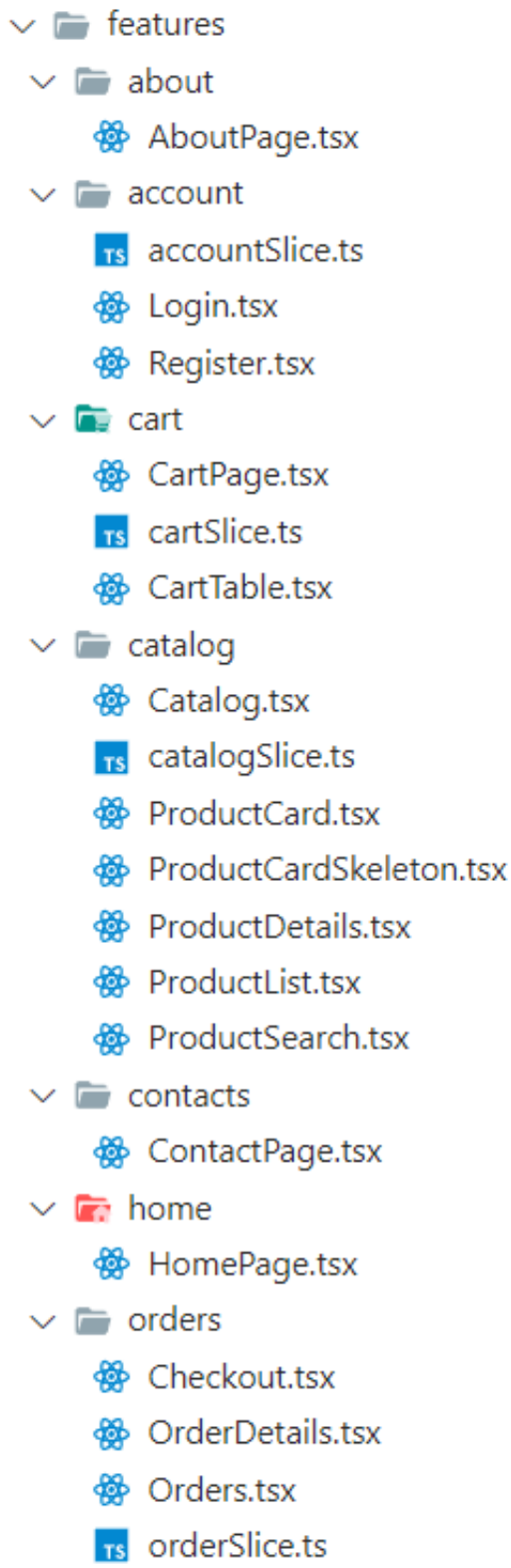


Рисунок 2.13 – Файли features: about, account, cart, catalog, contacts, home, orders

Опис логіки та застосування React, Redux та Redux Toolkit для оптимізації керування станом веб-застосунку:

Redux – це бібліотека для керування глобальним станом програми. Redux зазвичай використовується з бібліотекою React-Redux для взаємоінтеграції Redux і React. Redux Toolkit – рекомендований спосіб написання логіки Redux. Redux має декілька основних термінів:

- actions – це звичайні об’єкти з полем задаючим конкретний тип action, які описують «що сталося» в додатку;
- action creators – це функції, які створюють та повертають об’єкт action;
- reducers – це функції, які обчислюють нове значення стану на основі попереднього стану та action;
- store – це об’єкт, вміщуючий в себе стан додатку та запускаючий кореневий reducer кожного разу, коли виконується action dispatch;
- dispatch – це метод store, що викликається для оновлення стану передаючи об’єкт action;
- selectors – це функції, які приймають стан Redux як аргумент і повертають деякі дані.

З Redux використано такий структурний підхід як «односторонній потік даних» (рисунок 2.14). Інтерфейс користувача відображається на основі стану додатку в певний момент часу. Опишемо алгоритм оптимізації керування станом додатку. Коли щось відбувається у додатку:

- інтерфейс користувача виконує action dispatch;
- store запускає reducers, і стан оновлюється на основі того, що сталося;
- store сповіщає інтерфейс користувача про те, що стан змінився;
- інтерфейс користувача повторно відображається на основі нового стану.

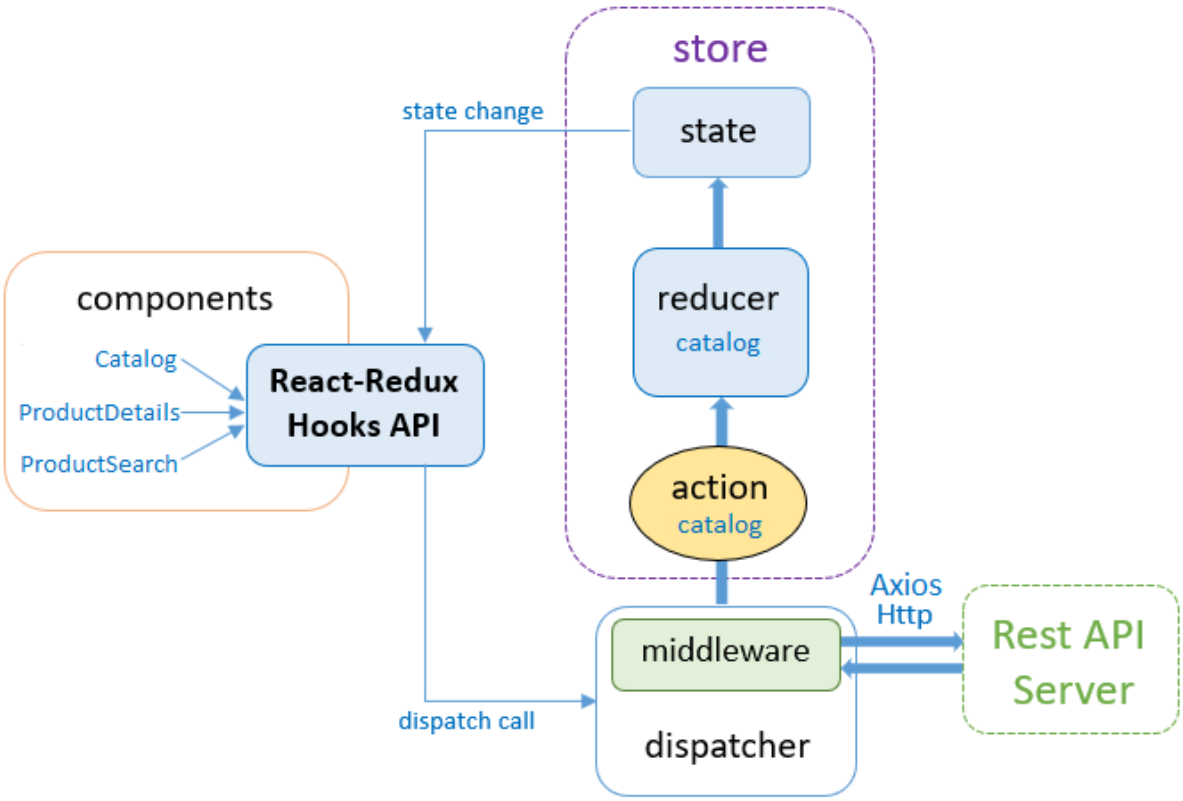


Рисунок 2.14 – Діаграма взаємодії елементів Redux

Redux Toolkit спрощує процес написання логіки Redux і налаштування store. Таким чином, Redux Toolkit дозволяє писати коротший код, що легше читати, і при цьому слідувати тій самій поведінці і потоку даних Redux.

Redux store створено за допомогою configureStore API з Redux Toolkit. configureStore приймає reducer функцію як іменований аргумент та автоматично налаштовує store із хорошими параметрами за замовчуванням.

React-Redux дозволяє компонентам React взаємодіяти з Redux store. Завдяки обгортці додатку у <Provider store={store}> всі компоненти мають змогу використовувати store. Глобальний стан має бути в Redux store, а локальний стан має залишатися в компонентах React.

Дані з Redux store можуть використовуватися компонентами React за потреби:

- будь-який компонент може читати будь-які дані, які є в store;
- кілька компонентів можуть одночасно читати одні й ті самі дані;
- компоненти повинні виймати найменшу кількість даних, які їм потрібні для відображення;
- компоненти можуть поєднувати значення з props та стану React та Redux store, щоб визначити, які елементи інтерфейсу користувача їм потрібно відобразити. Вони можуть зчитувати кілька фрагментів даних зі store та змінювати форму даних за потреби для відображення;
- будь-який компонент може виконувати action dispatch для оновлення стану.

Компоненти React зчитують дані зі store за допомогою хука useSelector. Selectors отримують увесь об’єкт стану та мають повертати значення. Selectors запускатимуться повторно щоразу, коли буде оновлено store, і якщо дані, які вони повертають, змінилися, компонент буде повторно відтворено. Багаторазові selector функції можуть бути створені, щоб інкапсулювати читання значень зі стану Redux.

Компоненти React викликають action dispatch для оновлення store за допомогою хука useDispatch та передачі action creator до dispatch. Після цього запустяться reducers, перевірять, чи актуальний цей action, і повернуть новий стан, якщо потрібно. Тимчасові дані, такі як значення введені у форми, слід зберігати як стан компонента React. Коли користувач завершить роботу з формою, тоді можна виконати action dispatch для оновлення store.

Логіка Redux організована у файли, які називаються slices. Slice містить reducer логіку та actions, пов’язані з певним функціоналом/частиною стану Redux. createSlice API з Redux Toolkit генерує action creators і action типи для кожної reducer функції наданої до slice.

Стан Redux оновлюється reducer функціями. Reducers завжди обчислюють новий стан незмінно, тобто копіюючи існуючі значення стану та замінюючи вміст копії новими даними. createSlice API з Redux Toolkit генерує slice reducer функції і дозволяє писати «мутуючий» код, який перетворюється на безпечні незмінні оновлення стану. Slice reducers додаються до поля reducer в configureStore, і це визначає назви полів даних і стану в store. createSlice API з Redux Toolkit використовує інструменти бібліотеки Immer, щоб дозволити «мутувати» незмінні оновлення.

Загалом, Redux reducers повинні дотримуватися певних правил:

– можуть обчислювати нові значення стану лише на основі аргументів state та action;
– можуть робити незмінні оновлення лише шляхом копіювання існуючого стану;
– не можуть містити жодної асинхронної логіки.

З Redux використано проміжне програмне забезпечення, щоб підтримувати асинхронну логіку. Стандартне асинхронне проміжне програмне забезпечення називається redux-thunk, яке вже входить до Redux Toolkit. Спеціальні функції thunk отримують dispatch і getState як аргументи та можуть використовувати їх як частину асинхронної логіки.

Виконання додаткових action dispatch допомагає відстежити стан завантаження виклику API. Типовим шаблоном є відправка action зі статусом “pending” перед викликом, потім “fulfilled” action, що містить дані, або “rejected” action, що містить помилку.

Redux Toolkit має createAsyncThunk API для виконання асинхронних action dispatch. createAsyncThunk приймає Callback з "корисним навантаженням", який має повернути Promise, і автоматично генерує типи action для pending / fulfilled / rejected статусів. Згенеровані action creators виконують action dispatch на основі Promise, що повертається. Ці action типи можна прослуховувати у createSlice за допомогою поля extraReducers, і оновлювати стан через reducers на основі цих actions. Action creators використовуються для автоматичного заповнення об’єкта extraReducers, щоб slice знав, які actions слід прослуховувати.

Крім того, для оптимізації продуктивності використано нормалізовану структуру стану, що є рекомендованим підходом для зберігання множин елементів. Нормалізація означає відсутність дублювання даних і збереження елементів, які зберігаються в таблиці пошуку за ідентифікатором елемента.

createEntityAdapter API з Redux Toolkit допомагає керувати нормалізованими даними у slice.

Об'єкт створеного адаптера включає в себе:

– getInitialState, що може приймати додаткові поля стану, наприклад стан завантаження;
– готові reducers для типових випадків, наприклад setAll, upsertOne, addMany і removeMany;
– getSelectors, що генерує selectors, такі як selectAll і selectById.

2.4 Аналіз безпеки даних

Загальний регламент про захист даних [23] застосовується з 25 травня 2018 року для гармонізації законів про конфіденційність даних у всій Європі:

- цей регламент встановлює правила, що стосуються захисту фізичних осіб щодо обробки персональних даних, а також правила, що стосуються вільного руху персональних даних;
- цей регламент захищає основні права та свободи фізичних осіб і, зокрема, їх право на захист персональних даних;
- вільне переміщення персональних даних у межах Союзу не повинно бути ані обмежено, ані заборонено з причин, пов'язаних із захистом фізичних осіб щодо обробки персональних даних.

При розробці даного веб-застосунку використано ASP.NET Core Identity, що представляє вбудовану в ASP.NET систему автентифікації та авторизації [24]. Ця система дозволяє користувачам безпечно створювати облікові записи, автентифікуватися, керувати обліковими записами або використовувати для входу на сайт облікові записи зовнішніх провайдерів.

ASP.NET Core Identity дозволяє додавати налаштування функцій входу/виходу та налаштування функцій профілю, які полегшують керуванням даних про користувача, який увійшов у систему. Сьогодні існує набагато ширший спектр варіантів зберігання даних (які дуже швидко зростають) для веб-додатків, і більшість розробників хочуть використовувати ASP.NET Core Identity щоб дозволити своїм веб-сайтам використовувати постачальників соціальних ідентифікаторів для автентифікації та авторизації. Терміни автентифікація та авторизація різняться між собою.

Автентифікація являє собою процес розпізнавання користувачів. Під час автентифікації користувач повинен підтвердити свою особу на веб-сервері, увійшовши в систему за допомогою паролю, електронної пошти, імені користувача, токенів автентифікації, cookies або за допомогою різних зовнішніх сервісів соціальних провайдерів.

Веб-застосунок з ASP.NET Core Identity використовує спосіб автентифікації «ім'я користувача та пароль» за замовчуванням, як і переважна більшість додатків. Коли користувач реєструється в додатку, він надає електронну пошту, ім’я користувача та пароль. Додаток з ASP.NET Core Identity створить хеш пароля та збереже його в базі даних разом із даними користувача.

Хеш є односторонньою функцією, тому за допомогою пароля можна розрахувати хеш, але за допомогою хешу вже не можна повернути вихідний пароль. З міркувань безпеки характеристики хеш-функції важливі; зокрема, хеш-функція має бути відносно дорогою для обчислення, оскільки якщо база даних з хешами паролів буде скомпрометована, для їх злому знадобиться занадто багато часу.

Коли справа доходить до входу в акаунт, користувачі надсилають своє ім’я користувача та пароль HTTP-методом POST. Сервер візьме ідентифікатори і спробує знайти наявний обліковий запис у своїй базі даних. Якщо він знаходить обліковий запис, він отримує збережений хеш пароля, пов’язаний з обліковим записом. Потім сервер хешує надісланий пароль і порівнює два хеші. Якщо хеші збігаються, то пароль правильний, і користувача можна автентифікувати. Якщо хеші не збігаються, користувач ввів неправильний пароль і його потрібно відхилити.

В конвеєрі обробки запиту для виконання автентифікації відповідає спеціальний middleware компонент – AuthenticationMiddleware. Для вбудовування цього middleware в конвеєр застосовується метод розширення UseAuthentication(). Слід зазначити, що метод UseAuthentication() повинен вбудовуватися в конвеєр до будь-яких компонентів middleware, які використовують автентифікацію користувачів.

Для застосування автентифікації цей компонент використовує сервіси автентифікації, зокрема, IAuthenticationService, які реєструються в додатку за допомогою методу AddAuthentication().

При розробці даного веб-застосунку для автентифікації встановлено схему автентифікації на основі JWT-токенів з налаштованим терміном приданості в 30 днів, яка зберігається у константі JwtBearerDefaults.AuthenticationScheme.

JWT є веб-стандартом, який визначає спосіб передачі даних про користувача у форматі JSON у зашифрованому вигляді.

JWT-токен складається з трьох частин:

– header – об'єкт JSON, який містить інформацію про тип токена та алгоритм його шифрування;
– payload – об'єкт JSON, який містить дані, потрібні для авторизації користувача;
– signature – рядок, який створюється за допомогою секретного коду, Header та Payload. Цей рядок служить для верифікації токена.

Схема автентифікації дозволяє вибирати певний обробник автентифікації. Обробник автентифікації власне виконує безпосередню автентифікацію користувачів на основі даних запитів і виходячи зі схеми автентифікації.

ASP.NET Core не обмежується вбудованими схемами автентифікації і надає можливість створювати власні схеми та обробники автентифікації. Як обробник автентифікації буде застосовуватися клас Microsoft.AspNetCore.Authentication.JwtBearer.JwtBearerHandler.

Для автентифікації за допомогою JWT-токенів до проєкту додано пакет Microsoft.AspNetCore.Authentication.JwtBearer використовуючи пакетний менеджер Nuget.

Крім застосування схеми автентифікації, необхідно підключити автентифікацію відповідного типу. Для цього використано метод AddJwtBearer(), що підключає та конфігурує автентифікацію за допомогою JWT-токенів і реалізований як метод розширення типу AuthenticationBuilder, який повертається методом AddAuthentication().

Авторизація – це процес, за допомогою якого сервер визначає, чи має клієнт дозвіл та умови на використання ресурсу або доступ до файлу після успішної автентифікації.

Для застосування авторизації необхідно зареєструвати сервіси авторизації за допомогою методу AddAuthorization().

Для виконання авторизації в конвеєрі обробки запиту відповідає спеціальний middleware компонент – AuthorizationMiddleware. Для вбудовування цього middleware в конвеєр застосовується метод розширення UseAuthorization().

Ключовим елементом механізму авторизації в ASP.NET Core є атрибут AuthorizeAttribute із простору імен Microsoft.AspNetCore.Authorization, який дозволяє обмежити доступ до ресурсів програми. В даному веб-застосунку такий атрибут використано для отримання даних поточного користувача.

За замовчуванням веб-браузери з метою безпеки обмежують ajax запити між різними доменами. Однак нерідко виникає ситуація, коли необхідно виконувати запити із додатка з однієї адреси (або домену) до додатка, який розміщено за іншою адресою. Для цього необхідно використовувати CORS.

Для підключення сервісів CORS у програмі викликається метод builder.Services.AddCors(). Щоб використовувати CORS для обробки запиту викликається метод app.UseCors(). Для конфігурації параметрів CORS цей метод використовує делегат, до якого передається об'єкт CorsPolicyBuilder. І за допомогою цього об'єкта можна виконати налаштування CORS.

Для розробки веб-застосунку взаємодію з колом адрес обмежено за допомогою методу WithOrigins().

Висновки до розділу

Отже, у даному розділі виконано моделювання та конструювання програмного забезпечення:

- проведено моделювання та аналіз програмного забезпечення, описано та наочно представлено за допомогою засобу моделювання BPMN бізнес процеси розробки;
- втілено архітектурний патерн розроблюваного веб-застосунку та деталізовано його компоненти, описано зв’язки між сутностями за допомогою ER діаграми;
- при конструюванні програмного забезпечення описано утиліти, бібліотеки та інше стороннє програмне забезпечення, що використовується у розробці. Представлено опис бази даних, її таблиць та полів, а також діаграму структури бази даних. Наведено програмні структури серверної та клієнтської частин розроблюваного веб-застосунку. Описано логіку та застосування алгоритмічних та технічних рішень для оптимізації керування станом веб-застосунку;
- проведено аналіз безпеки даних, виклавши питання та рішення пов’язані з вразливостями програмного забезпечення та безпекою даних.

3 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз якості програмного забезпечення

Сканування коду з GitHub Actions [25] використовується для аналізу коду в GitHub репозиторії, щоб знайти в ньому уразливості та помилки. Будь-які проблеми, виявлені під час аналізу, відображаються у інтерфейсі GitHub.

Сканування коду дає можливість знаходити, сортувати та визначати пріоритетність виправлень для існуючих проблем у коді. Сканування коду також запобігає створенню нових проблем розробниками. Сканування можна запланувати на певну дату та час або запускати сканування, коли в репозиторії відбувається певна подія, наприклад push.

CodeQL Action запускає провідний у галузі механізм семантичного аналізу коду GitHub, CodeQL, проти вихідного коду репозиторію, щоб знайти уразливості в безпеці та якості коду [26]. Потім результати автоматично завантажуються на GitHub, щоб їх можна було відобразити на вкладці безпеки репозиторію.

CodeQL виконує розширюваний набір запитів, розроблених спільнотою та лабораторією безпеки GitHub для пошуку поширених уразливостей у вашому коді.

Для аналізу якості веб-застосунку використано набір запитів CodeQL, що називається security-and-quality. Цей набір включає в себе запити з набору за замовчуванням, а також запити з меншою серйозністю та точністю, запити щодо підтримуваності та надійності.

У файл конфігурації CodeQL Action workflow додано запис with: queries: +security-and-quality у розділ uses: github/codeql-action/init@v2 (рисунок 3.1).

dd-web-app / .github / workflows / **codeql.yml**

Code | Blame | 77 lines (65 loc) · 2.99 KB

```yaml
on:
  push:
    branches: [ "main" ]
  pull_request:
    # The branches below must be a subset of the branches above
    branches: [ "main" ]
  schedule:
    - cron: '35 2 * * 0'

jobs:
  analyze:
    name: Analyze
    runs-on: ${{ (matrix.language == 'swift' && 'macos-latest') || 'ubuntu-latest' }}
    permissions:
      actions: read
      contents: read
      security-events: write

    strategy:
      fail-fast: false
      matrix:
        language: [ 'csharp', 'javascript' ]
        # CodeQL supports [ 'cpp', 'csharp', 'go', 'java', 'javascript', 'python', 'ruby' ]
        # Use only 'java' to analyze code written in Java, Kotlin or both
        # Use only 'javascript' to analyze code written in JavaScript, TypeScript or both
        # Learn more about CodeQL language support at https://aka.ms/codeql-docs/language-support

    steps:
    - name: Checkout repository
      uses: actions/checkout@v3

    # Initializes the CodeQL tools for scanning.
    - name: Initialize CodeQL
      uses: github/codeql-action/init@v2
      with:
        languages: ${{ matrix.language }}
        queries: +security-and-quality
        # If you wish to specify custom queries, you can do so here or in a config file.
        # By default, queries listed here will override any specified in a config file.
        # Prefix the list here with "+" to use these queries and those in the config file.

```

Рисунок 3.1 – Конфігурація CodeQL Action workflow

Діагностична інформація відображається в Action workflow логах і складається із підсумкових показників і діагностики екстрактора.

Підсумкові показники включають:

– рядки коду в кодовій базі (використовуються як базова лінія), перед створенням і вилученням бази даних CodeQL;
– рядки коду в базі даних CodeQL, отримані з коду, включаючи зовнішні бібліотеки та автоматично створені файли;
– рядки коду в базі даних CodeQL, за винятком автоматично створених файлів і зовнішніх бібліотек.

Діагностика екстрактора охоплює лише файли, які було показано під час аналізу, метрики включають:

– кількість успішно проаналізованих файлів;
– кількість файлів, які викликали помилки екстрактора під час створення бази даних;
– кількість файлів, які згенерували попередження екстрактора під час створення бази даних.

Перегляд лог-файлів сканування коду наведено на рисунках 3.2 – 3.3.

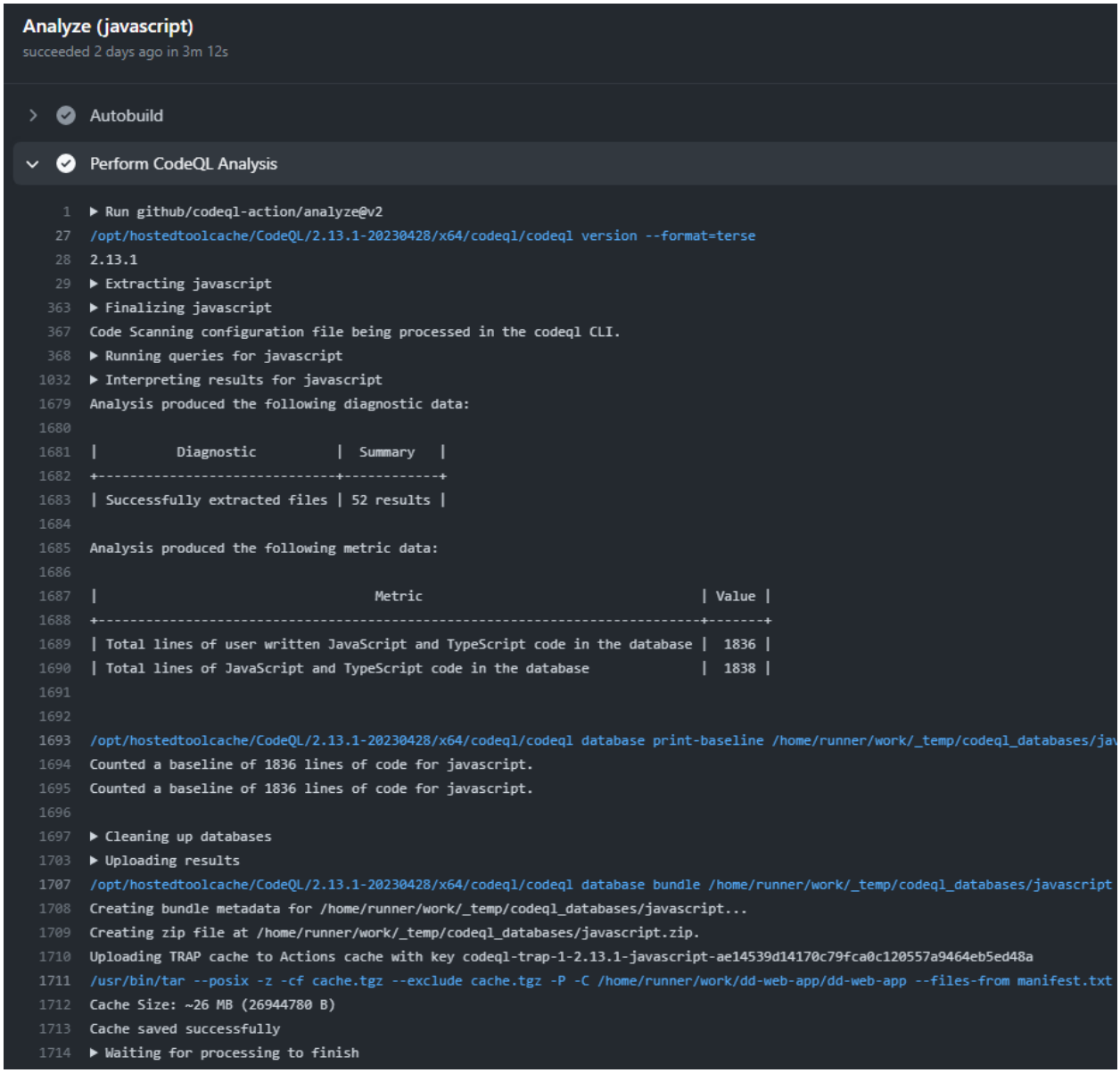


Рисунок 3.2 – Лог-файл CodeQL аналізу Javascript коду

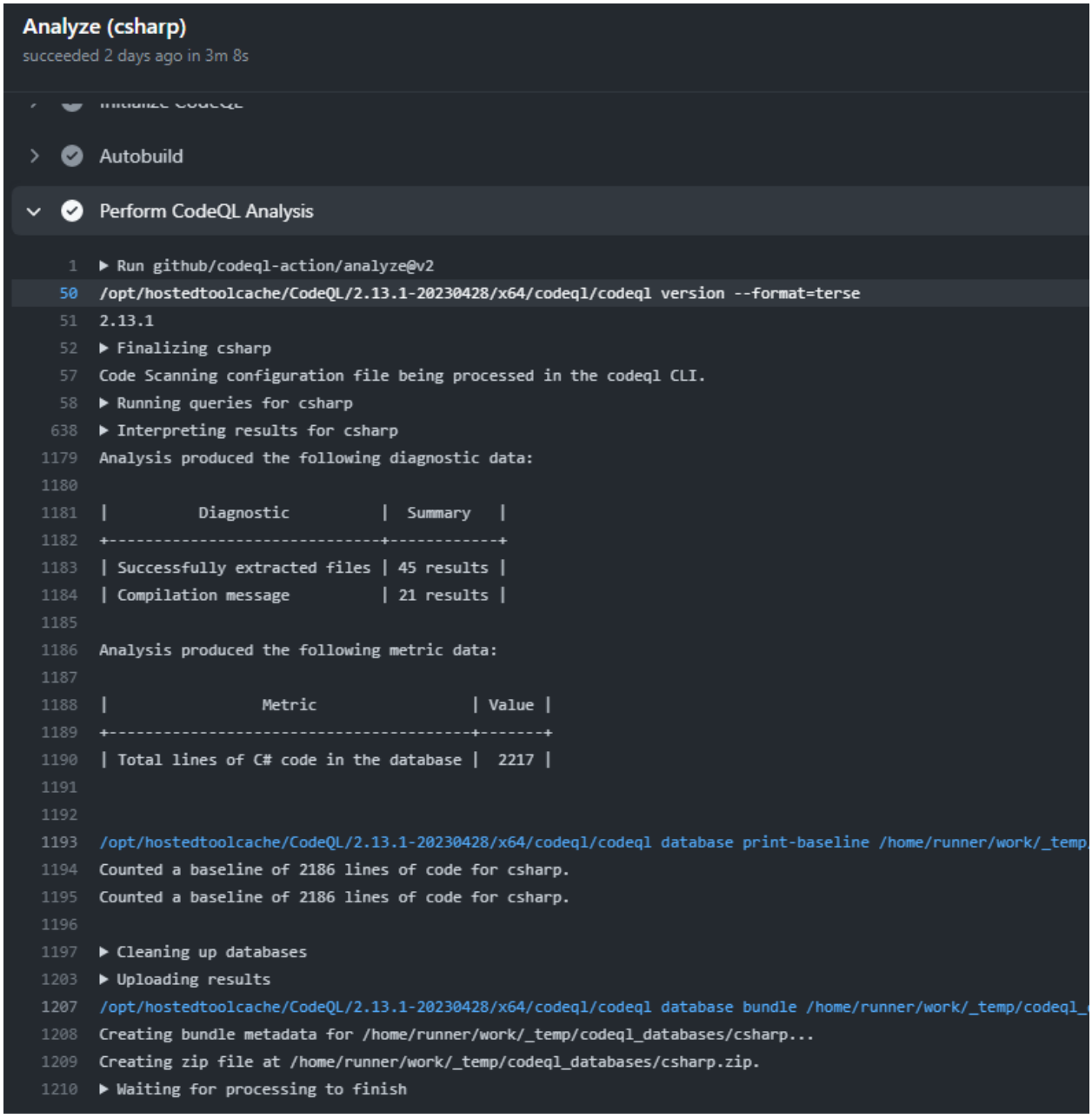


Рисунок 3.3 – Лог-файл CodeQL аналізу C# коду

На рисунку 3.4 показано результати аналізу якості веб-застосунку.

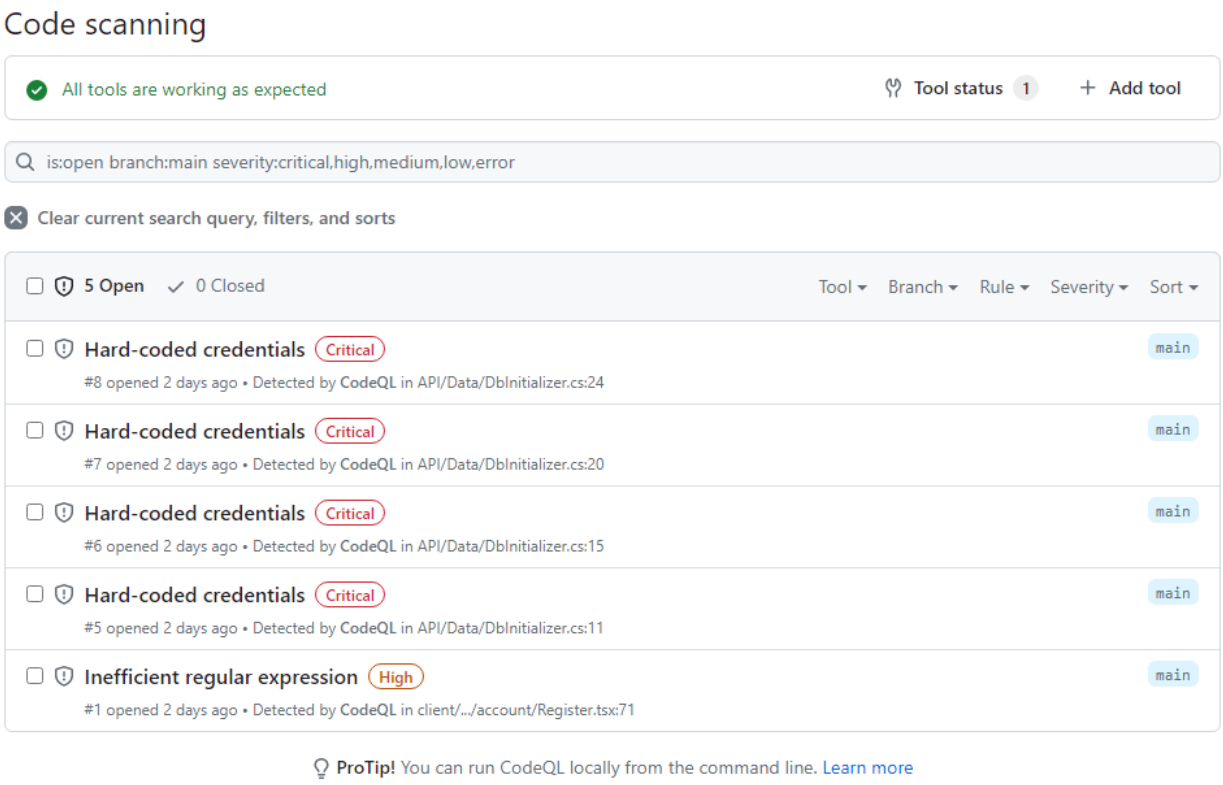


Рисунок 3.4 – Результати аналізу якості веб-застосунку

Проведено інтерпретацію результатів:

Чотири уразливості «Hard-coded credentials» можуть бути небезпечними в принципі, оскільки облікові дані буде легко виявити, але в даному випадку – це не критично. Причиною є те, що клас DbInitializer створює облікові записи неіснуючих користувачів у процесі database seeding лише для прикладу.

Уразливість «Inefficient regular expression» не є критичною, оскільки варіантів та шаблонів регулярних виразів для відповідності формату електронної пошти у інтернеті безліч, і всі вони мають власні особливості застосування та недоліки.

Після цього проведено аналіз виконання нефункціональних вимог.

При розробці веб-застосунку вдалося досягти виконання таких нефункціональних вимог:

- передбачено валідацію введення інформації та захист від некоректних дій користувача;
- забезпечено цілісність інформації в базі даних;
- програмне забезпечення функціонує на IBM-сумісних персональних комп‘ютерах;
- підтримується мінімальна конфігурація технічних засобів:
  - тип процесору: Intel Core i5;
  - об‘єм ОЗП: 4 Гб;
  - підключення до мережі Інтернет зі швидкістю від 20 мегабіт;
- підтримується рекомендована конфігурація технічних засобів:
  - тип процесору: Intel Core i7;
  - об‘єм ОЗП: 8 Гб;
  - підключення до мережі Інтернет зі швидкістю від 100 мегабіт;
- програмне забезпечення працює під управлінням операційних систем сімейства Windows та підтримувати браузер Google Chrome;
- забезпечено підтримку англійської локалізації інтерфейса.

3.2 Опис процесів тестування

Мануальне тестування є одним із фундаментальних процесів тестування, тому що воно може виявити як видимі, так і приховані дефекти програмного забезпечення. Різниця між очікуваним результатом і фактичним результатом від програмного забезпечення визначається як дефект. Розробник усуває дефекти і тестувальник проводить повторне тестування.

Основним процесом тестування в процесі аналізу необхідності тестів обрано мануальне тестування, оскільки воно є обов’язковим для кожного нещодавно розробленого програмного забезпечення ще до планування автоматизованих тестів.

Тестування методом чорної скриньки – це використаний метод мануального тестування програмного забезпечення, за якого функціональні можливості програмного забезпечення перевіряються без знання внутрішньої структури коду, деталей реалізації та внутрішніх шляхів. Тестування методом чорної скриньки в основному зосереджується на вхідних та вихідних даних програмного забезпечення і повністю базується на вимогах і специфікаціях програмного забезпечення.

Було проведено мануальне тестування програмного забезпечення, опис прикладів відповідних тестових кейсів наведено у таблицях 3.1 – 3.10.

Таблиця 3.1 – Тест 1.1

Тест	Реєстрація користувача з коректними вхідними даними
Модуль	Обліковий запис
Номер тесту	1.1
Початковий стан системи	Користувач знаходиться на сторінці реєстрації
Вхідні дані	Коректні електронна пошта, ім’я користувача, пароль
Опис проведення тесту	У відповідні поля вводяться: коректна електронна пошта, яка до цього не була зареєстрована в системі та ім’я користувача та пароль, що має містити символ верхнього регістру, символ нижнього регістру, цифру та спеціальний символ. Довжина паролю має бути не менше шести символів. Після цього натискається кнопка підтвердження реєстрації.
Очікуваний результат	Реєстрація проходить успішно, користувач додається у систему і перенаправляється на сторінку авторизації.
Фактичний результат	Реєстрація проходить успішно, користувач додається у систему і перенаправляється на сторінку авторизації.

Таблиця 3.2 – Тест 1.2

Тест	Реєстрація користувача з некоректними вхідними даними
Модуль	Обліковий запис
Номер тесту	1.2
Початковий стан системи	Користувач знаходиться на сторінці реєстрації
Вхідні дані	Некоректні електронна пошта, ім’я користувача, пароль

Продовження таблиці 3.2

Опис проведення тесту	У відповідні поля вводяться: некоректна електронна пошта, яка до цього не була зареєстрована в системі та ім’я користувача та пароль, що не містить символ верхнього регістру, символ нижнього регістру, цифру або спеціальний символ. Довжина паролю менше шести символів.
Очікуваний результат	Кнопка реєстрації залишається неактивною. Якщо якесь конкретне поле введено некоректно, то воно підсвічується червоним надписом.
Фактичний результат	Кнопка реєстрації залишається неактивною. Якщо якесь конкретне поле введено некоректно, то воно підсвічується червоним надписом.

Таблиця 3.3 – Тест 1.3

Тест	Авторизація користувача з коректними вхідними даними
Модуль	Обліковий запис
Номер тесту	1.3
Початковий стан системи	Користувач знаходиться на сторінці авторизації
Вхідні дані	Коректні ім’я користувача, пароль
Опис проведення тесту	У відповідні поля вводяться: коректні, існуючі та відповідні одне одному ім’я користувача та пароль, що має містити символ верхнього регістру, символ нижнього регістру, цифру та спеціальний символ. Довжина паролю має бути не менше шести символів. Після цього натискається кнопка авторизації.

Продовження таблиці 3.3

Очікуваний результат	Авторизація проходить успішно, користувач перенаправляється на сторінку каталогу.
Фактичний результат	Авторизація проходить успішно, користувач перенаправляється на сторінку каталогу.

Таблиця 3.4 – Тест 1.4

Тест	Авторизація користувача з некоректними вхідними даними
Модуль	Обліковий запис
Номер тесту	1.4
Початковий стан системи	Користувач знаходиться на сторінці авторизації
Вхідні дані	Некоректні ім’я користувача, пароль
Опис проведення тесту	У відповідні поля вводяться: некоректні або неіснуючі ім’я користувача та пароль, що не містить символ верхнього регістру, символ нижнього регістру, цифру або спеціальний символ. Довжина паролю менше шести символів.
Очікуваний результат	При відсутності вхідних даних кнопка авторизації залишається неактивною. Якщо якесь конкретне поле введено некоректно, то воно підсвічується червоним надписом. При авторизації з неіснуючиими обліковими даними користувач отримує повідомлення помилки Unauthorized.

Продовження таблиці 3.4

Фактичний результат	При відсутності вхідних даних кнопка авторизації залишається неактивною. Якщо якесь конкретне поле введено некоректно, то воно підсвічується червоним надписом. При авторизації з неіснуючиими обліковими даними користувач отримує повідомлення помилки Unauthorized.

Таблиця 3.5 – Тест 2.1

Тест	Перегляд деталей продукту
Модуль	Каталог продуктів
Номер тесту	2.1
Початковий стан системи	Користувач знаходиться на сторінці каталогу
Вхідні дані	-
Опис проведення тесту	Натискається кнопка «View», що відповідна обраному продукту у каталозі.
Очікуваний результат	Веб-застосунок успішно відображає деталі продукту.
Фактичний результат	Веб-застосунок успішно відображає деталі продукту.

Таблиця 3.6 – Тест 2.2

Тест	Перегляд деталей неіснуючого продукту
Модуль	Каталог продуктів
Номер тесту	2.2
Початковий стан системи	Користувач знаходиться у браузері
Вхідні дані	Посилання на сторінку продукту з неіснуючим ID
Опис проведення тесту	В адресний рядок вводиться посилання на сторінку продукту з неіснуючим ID та виконується перехід на введену адресу.
Очікуваний результат	Веб-застосунок відображає помилку 404 Not Found.
Фактичний результат	Веб-застосунок відображає помилку 404 Not Found.

Таблиця 3.7 – Тест 3.1

Тест	Додавання продуктів до кошику
Модуль	Кошик
Номер тесту	3.1
Початковий стан системи	Користувач знаходиться на сторінці деталей обраного продукту
Вхідні дані	Додатна кількість продуктів для додавання в кошик
Опис проведення тесту	В поле «Quantity» вводиться додатна кількість продуктів для додавання в кошик. Після цього натискається кнопка «Add to cart».
Очікуваний результат	В кошик успішно додано продукт в заданій кількості.
Фактичний результат	В кошик успішно додано продукт в заданій кількості.

Таблиця 3.8 – Тест 3.2

Тест	Додавання нуля продуктів до кошику
Модуль	Кошик
Номер тесту	3.2
Початковий стан системи	Користувач знаходиться на сторінці деталей обраного продукту
Вхідні дані	Нульова кількість продуктів для додавання в кошик
Опис проведення тесту	В поле «Quantity» вводиться нульова кількість продуктів для додавання в кошик.
Очікуваний результат	Кнопка «Add to cart» залишається неактивною.
Фактичний результат	Кнопка «Add to cart» залишається неактивною.

Таблиця 3.9 – Тест 4.1

Тест	Створення замовлення без обраного методу оплати
Модуль	Замовлення
Номер тесту	4.1
Початковий стан системи	Користувач авторизувався та знаходиться на сторінці чекауту
Вхідні дані	Продукти, додані до кошику
Опис проведення тесту	Поле методу оплати залишається порожнім. Після цього натискається кнопка «Place order».
Очікуваний результат	Створення замовлення неуспішне, відображається повідомлення про відсутній метод оплати.
Фактичний результат	Створення замовлення неуспішне, відображається повідомлення про відсутній метод оплати.

Таблиця 3.10 – Тест 4.2

Тест	Перегляд деталей неіснуючого замовлення
Модуль	Замовлення
Номер тесту	4.2
Початковий стан системи	Користувач авторизувався та знаходиться у браузері
Вхідні дані	Посилання на сторінку замовлення з неіснуючим ID
Опис проведення тесту	В адресний рядок вводиться посилання на сторінку замовлення з неіснуючим ID та виконується перехід на введену адресу.
Очікуваний результат	Веб-застосунок відображає помилку 404 Not Found.
Фактичний результат	Веб-застосунок відображає помилку 404 Not Found.

3.3 Опис контрольного прикладу

На рисунках 3.5 – 3.18 наведено проходження по основному функціоналу веб-застосунку.

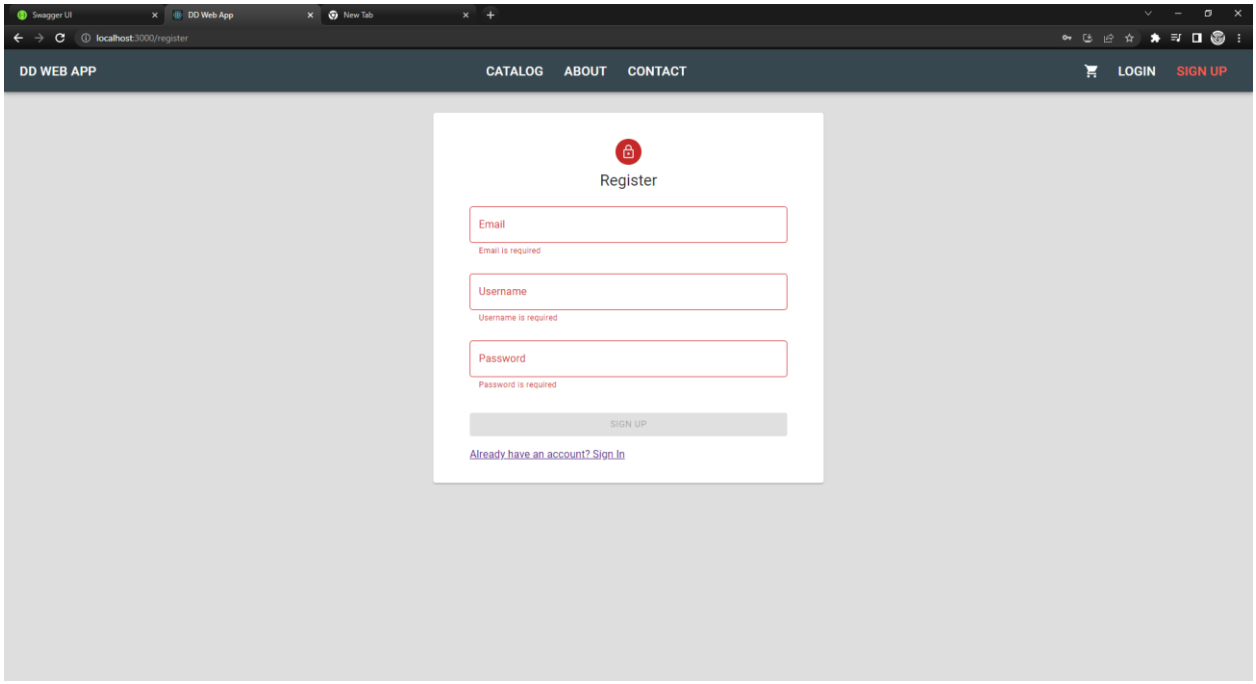


Рисунок 3.5 – Реєстрація користувача

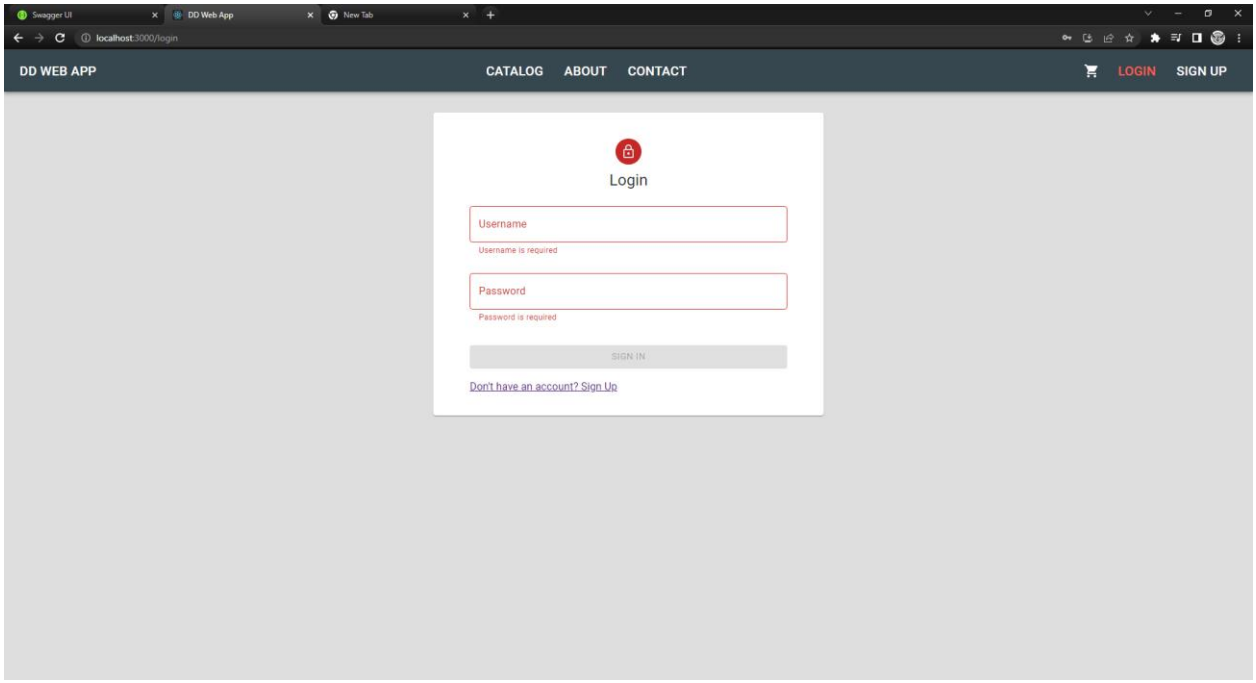


Рисунок 3.6 – Авторизація користувача

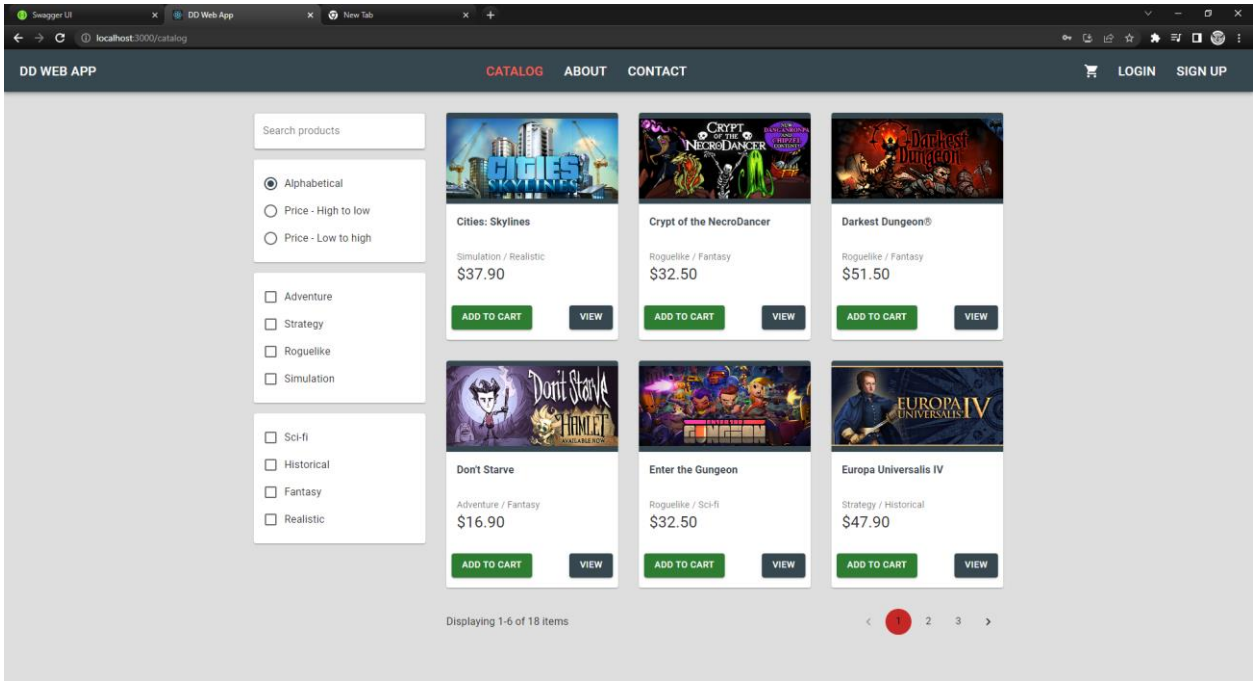


Рисунок 3.7 – Перегляд каталогу продуктів

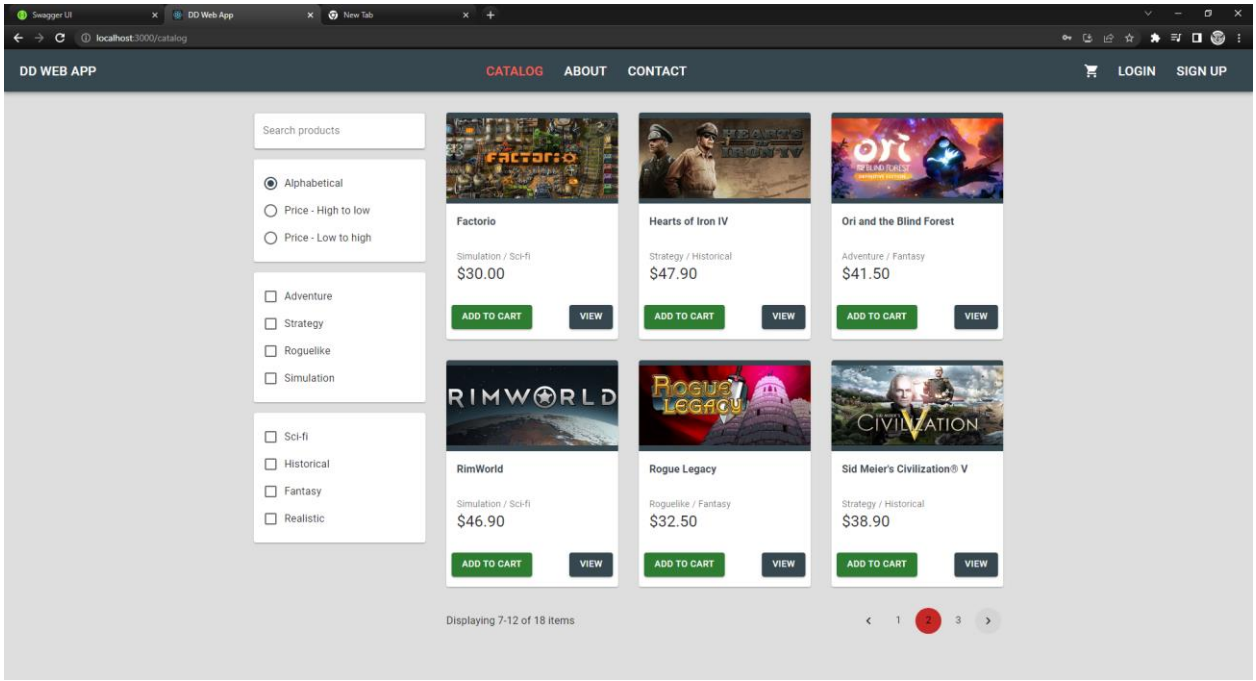


Рисунок 3.8 – Посторінкова навігація у каталозі

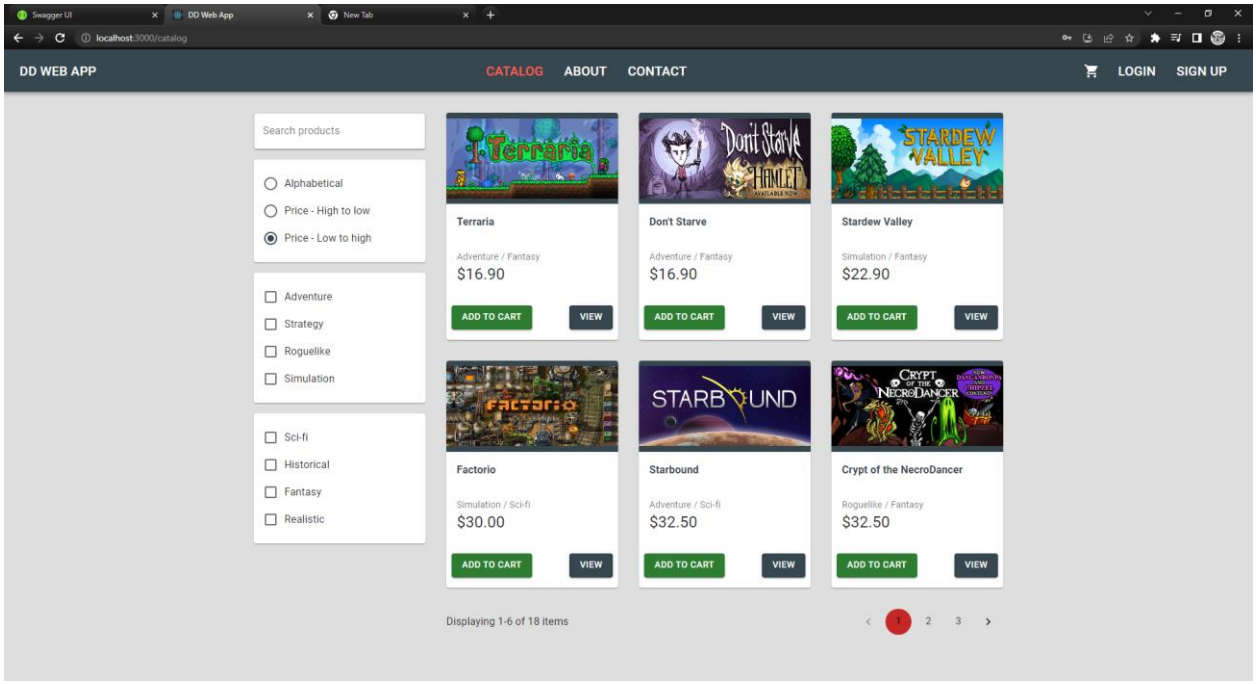


Рисунок 3.9 – Сортування продуктів у каталозі

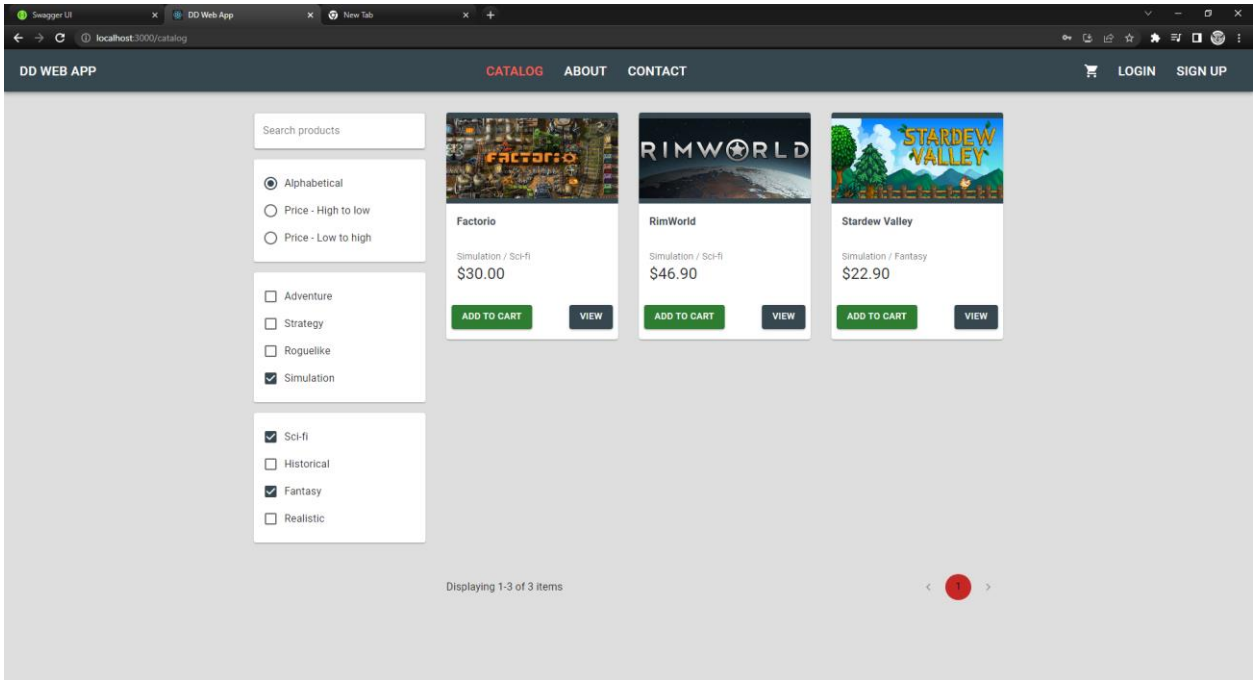


Рисунок 3.10 – Фільтрування продуктів у каталозі

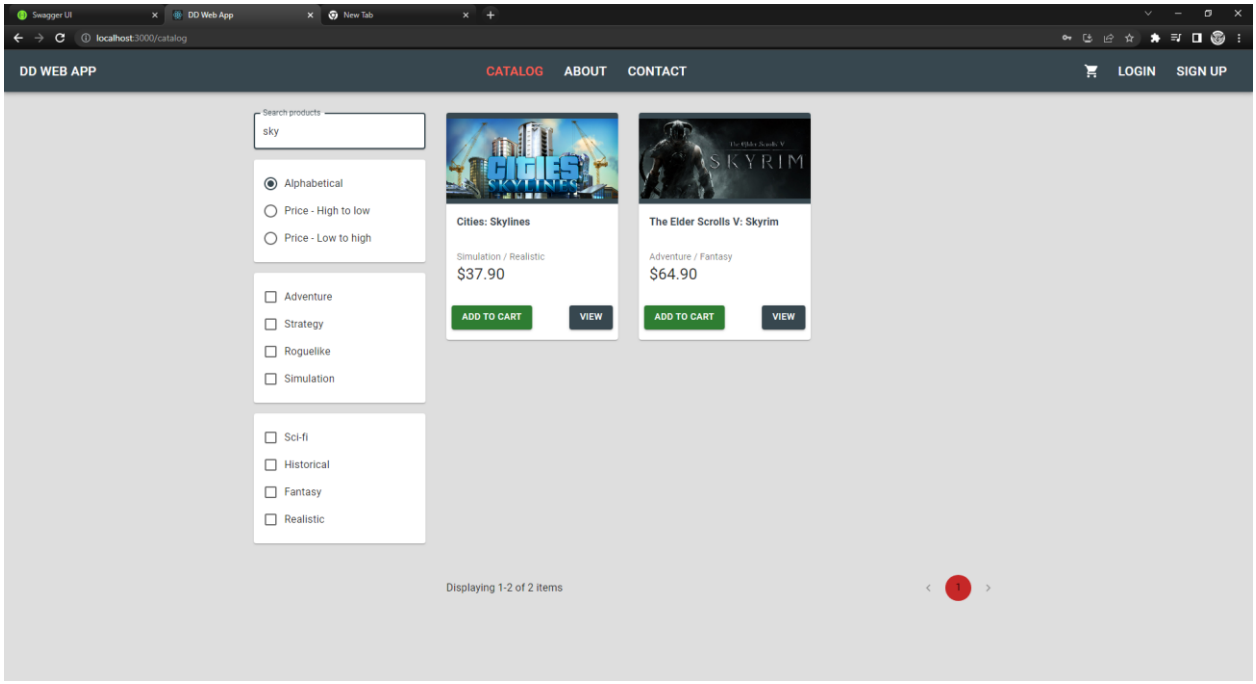


Рисунок 3.11 – Пошук продуктів у каталозі

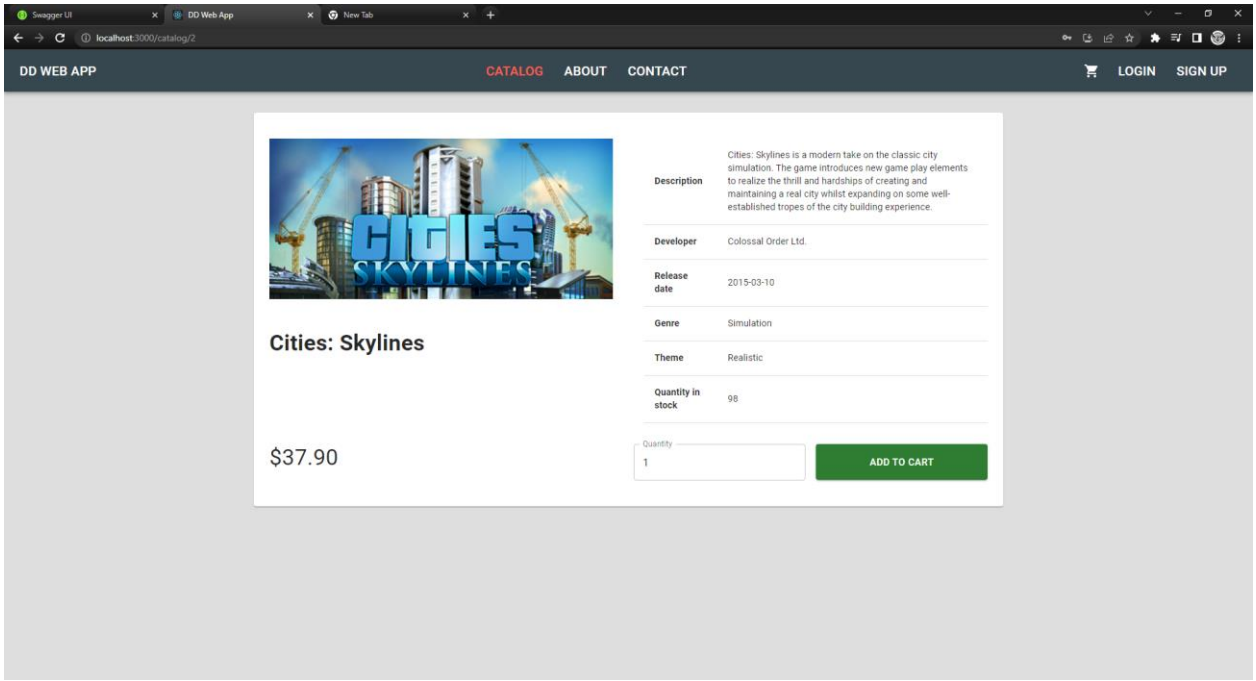


Рисунок 3.12 – Перегляд деталей продукту

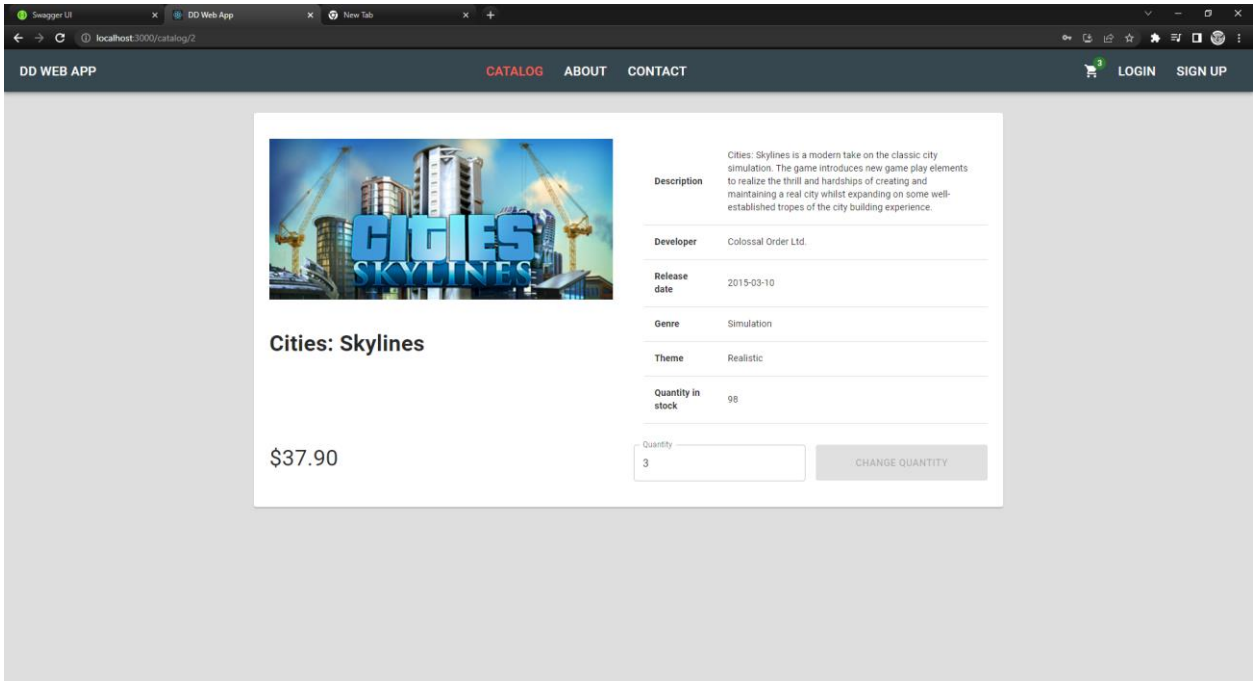


Рисунок 3.13 – Додавання продукту до кошику

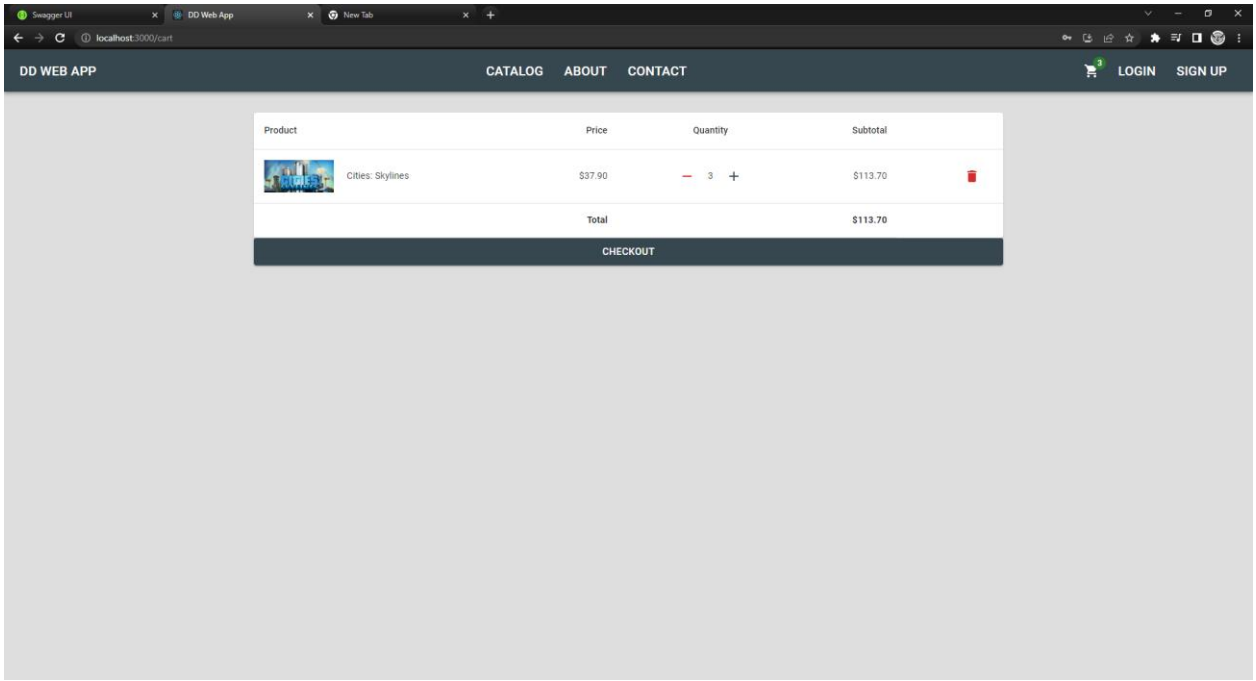


Рисунок 3.14 – Перегляд кошику

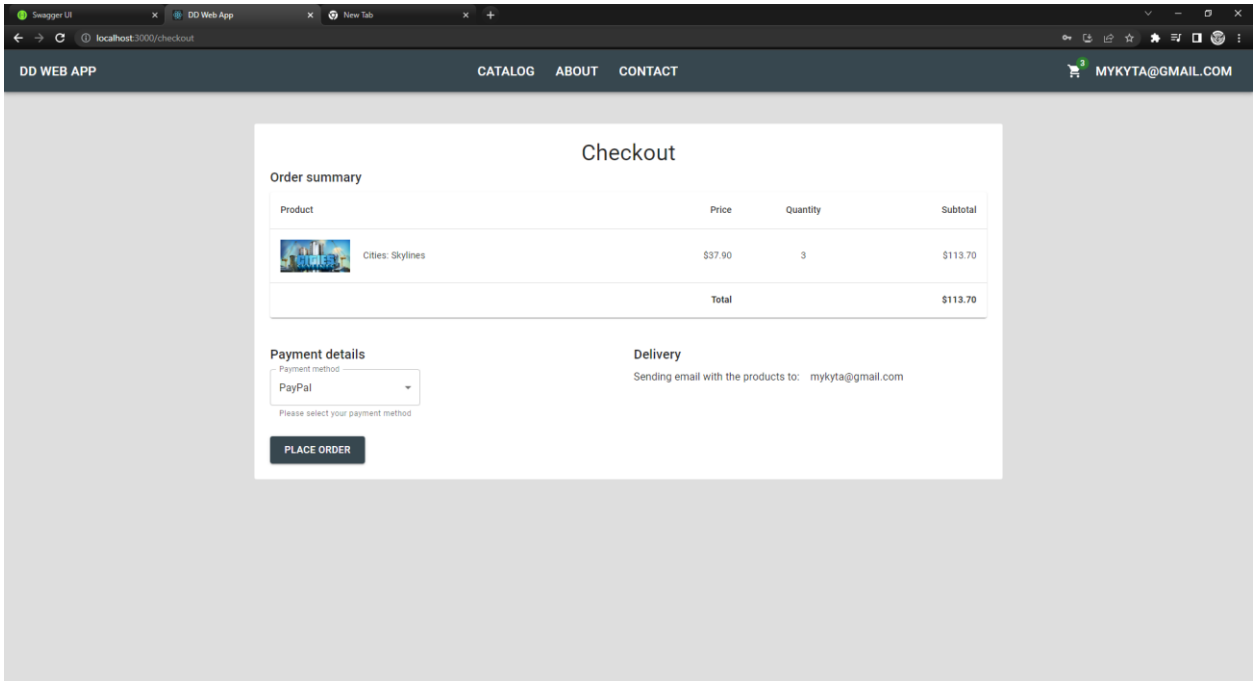


Рисунок 3.15 – Чекаут

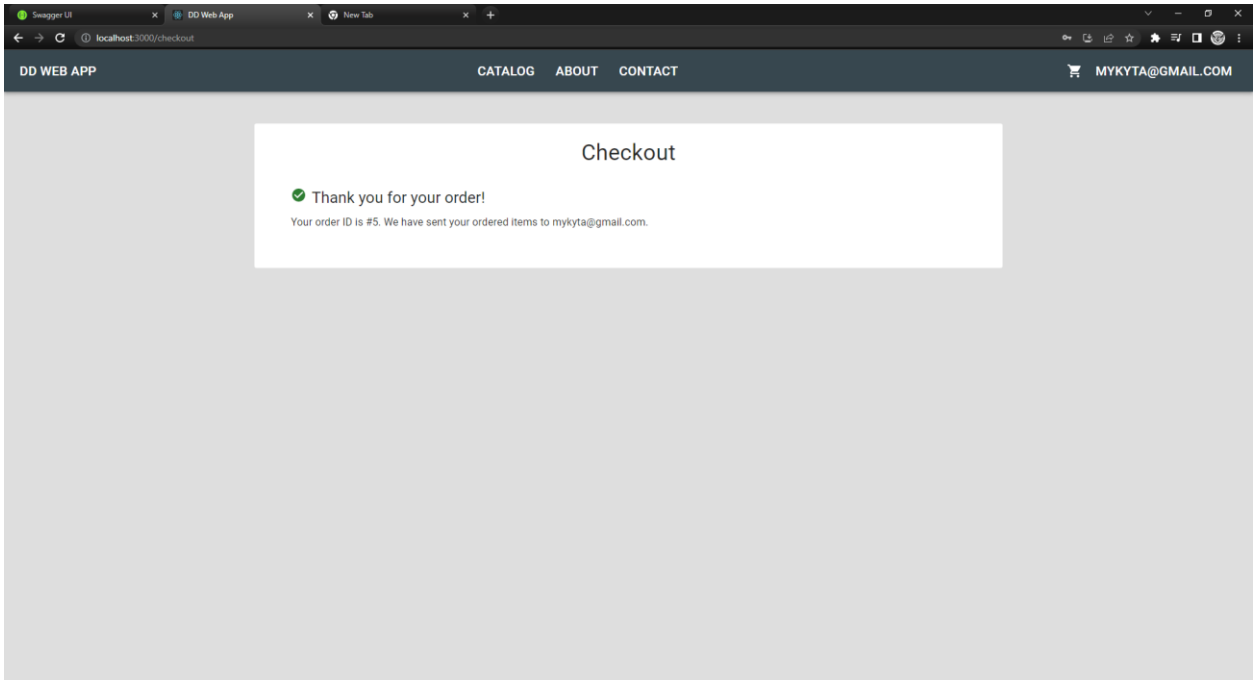


Рисунок 3.16 – Створення замовлення

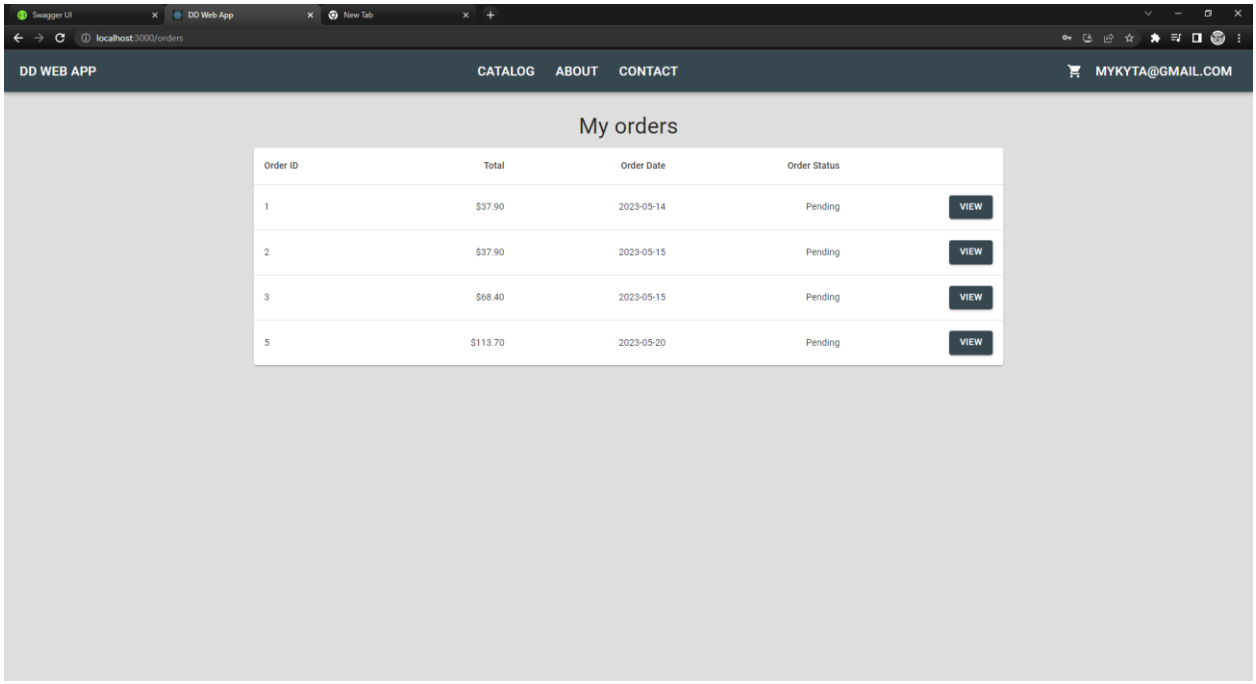


Рисунок 3.17 – Перегляд списку замовлень

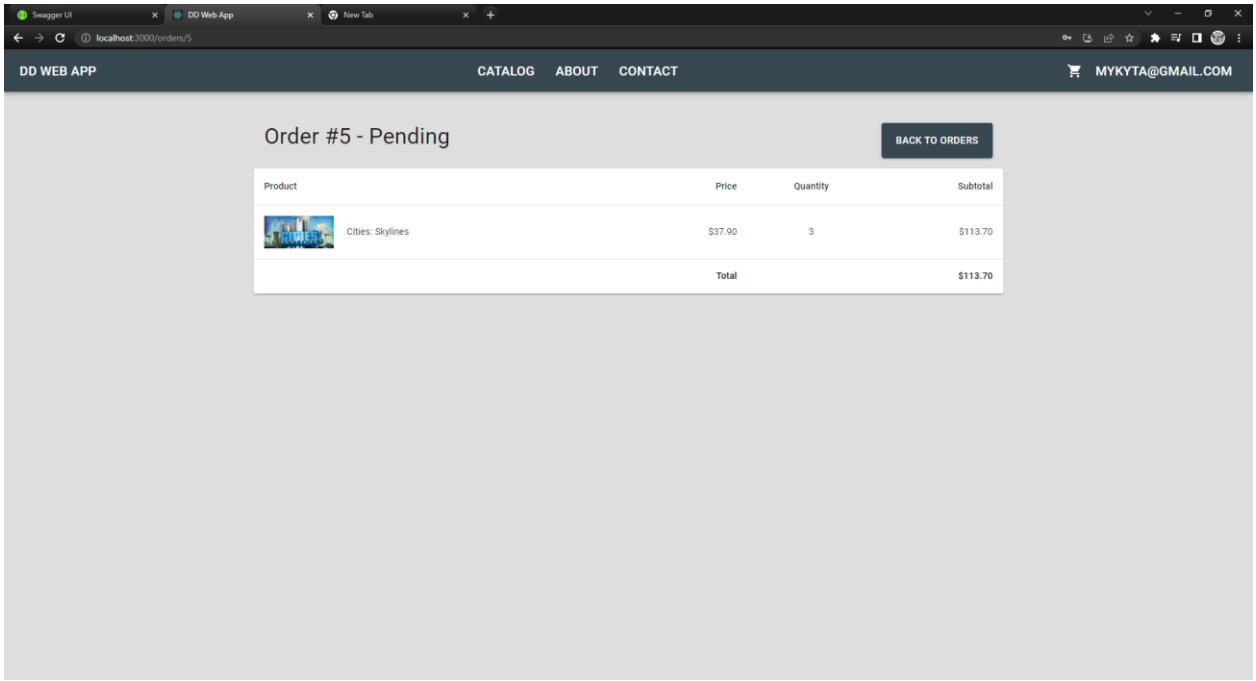


Рисунок 3.18 – Перегляд деталей замовлення

Висновки до розділу

Отже, у даному розділі виконано аналіз якості та тестування програмного забезпечення:

– проведено аналіз якості програмного забезпечення та аналіз виконання нефункціональних вимог. Виконано сканування коду за допомогою GitHub CodeQL Action;
– описано процеси тестування та тестові кейси, проведено мануальне тестування програмного забезпечення;
– описано контрольний приклад, навівши ілюстроване проходження по основному функціоналу веб-застосунку.

4 ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Розгортання програмного забезпечення

Fly.io обрано як платформу з безкоштовним планом для розгортання full stack додатків і високодоступних кластерів баз даних PostgreSQL, які працюють на фізичних серверах у містах, розташованих поблизу користувачів [27].

Docker – це набір платформ як сервісних продуктів, які використовують віртуалізацію на рівні ОС для доставки програмного забезпечення в пакетах, які називаються контейнерами [28].

Клієнтський додаток React збирається командою npm run build та статично розміщується на API сервері.

Docker image з API сервером, що збирається командою docker build, та postgres cluster з сервером бази даних було вирішено розгорнути на платформі fly.io з використанням інтерфейсу командного рядка flyctl.

API сервер розгорнуто командами fly launch і fly deploy, а сервер бази даних командою fly postgres create.

Розгортання починається коли новий код застосунку доставляється з push у репозиторій у гілку main та викликає action за workflow конфігурацією у docker-push.yml. Тоді у середовищі GitHub Actions створюється Docker image за допомогою Dockerfile, що знаходиться у проєкті. Цей image розгортається з flyctl deploy на платформі fly.io.

Інформацію про розгорнутий проєкт наведено на рисунках 4.1 – 4.3.

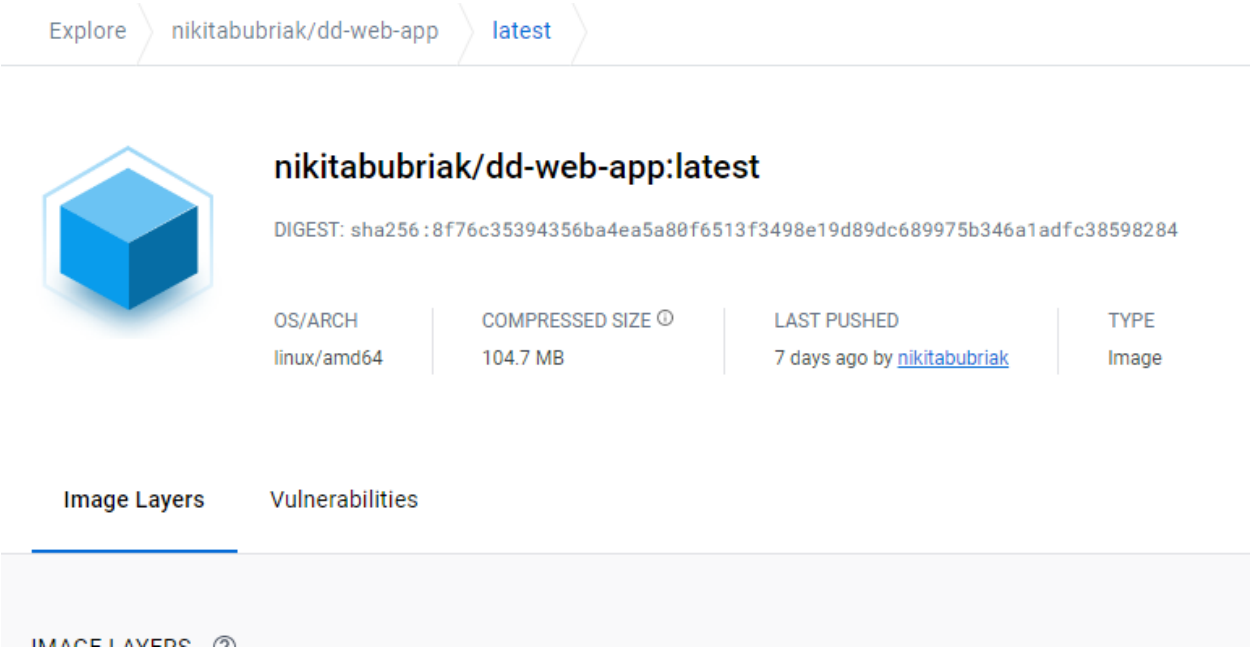


Рисунок 4.1 – Docker image додатку

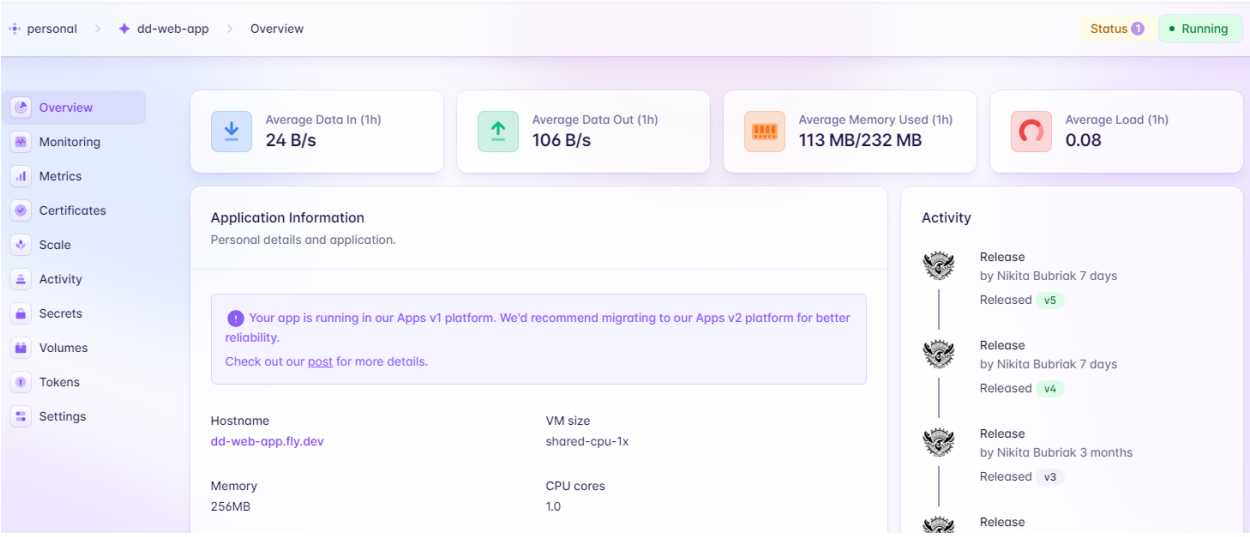


Рисунок 4.2 – Панель з інформацією про розгорнутий додаток

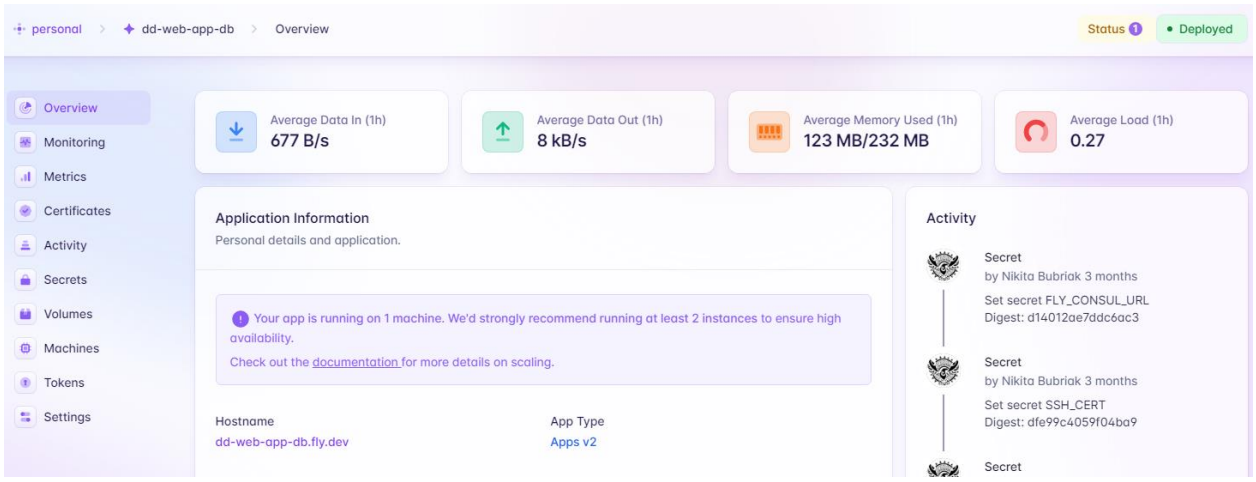


Рисунок 4.3 – Панель з інформацією про розгорнутий кластер бази даних

4.2 Підтримка програмного забезпечення

Користувачі повинні мати можливість отримати нову версію веб-застосунку після того як розробник виконає push нової версії до репозиторію проєкту. Для автоматизації цього процесу був використаний сервіс GitHub Actions, який надає можливості для постійної інтеграції і розгортання (рисунки 4.4 – 4.5).

GitHub Actions дає можливість створювати workflows, які збирають і тестують кожний pull request або push до репозиторію [25].

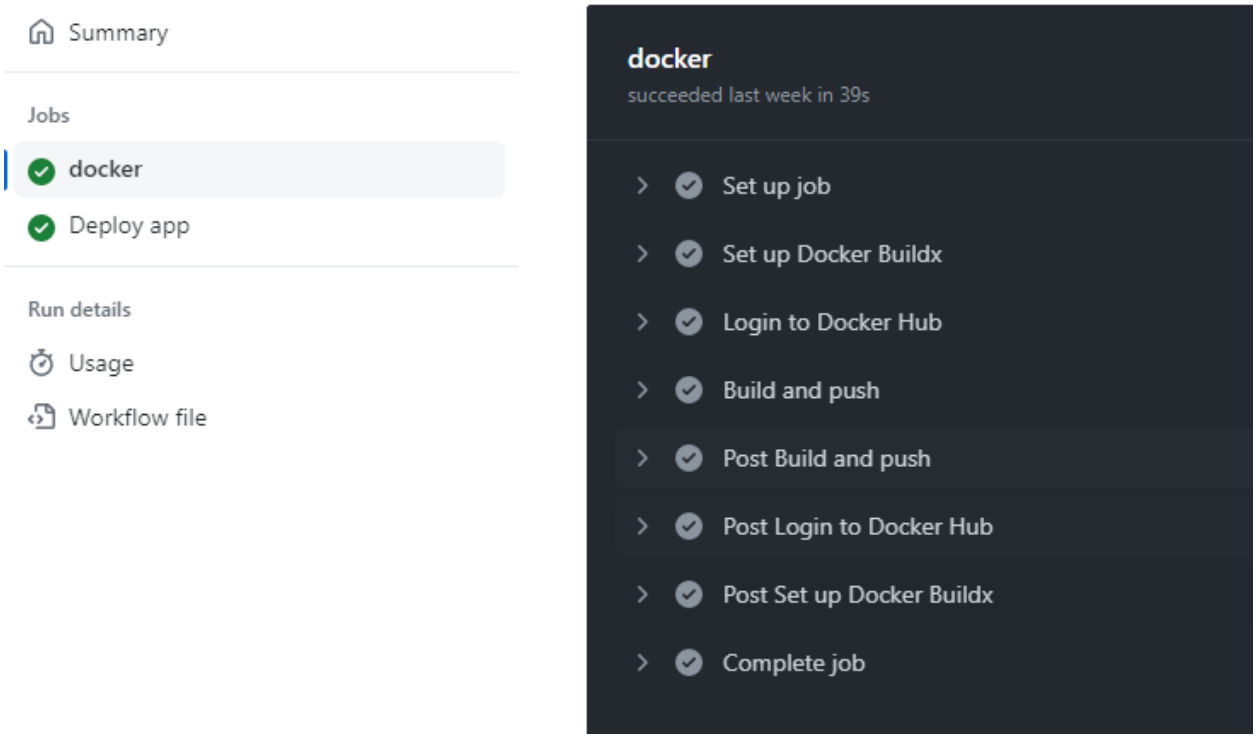


Рисунок 4.4 – Постійна інтеграція і розгортання Docker з GitHub Actions

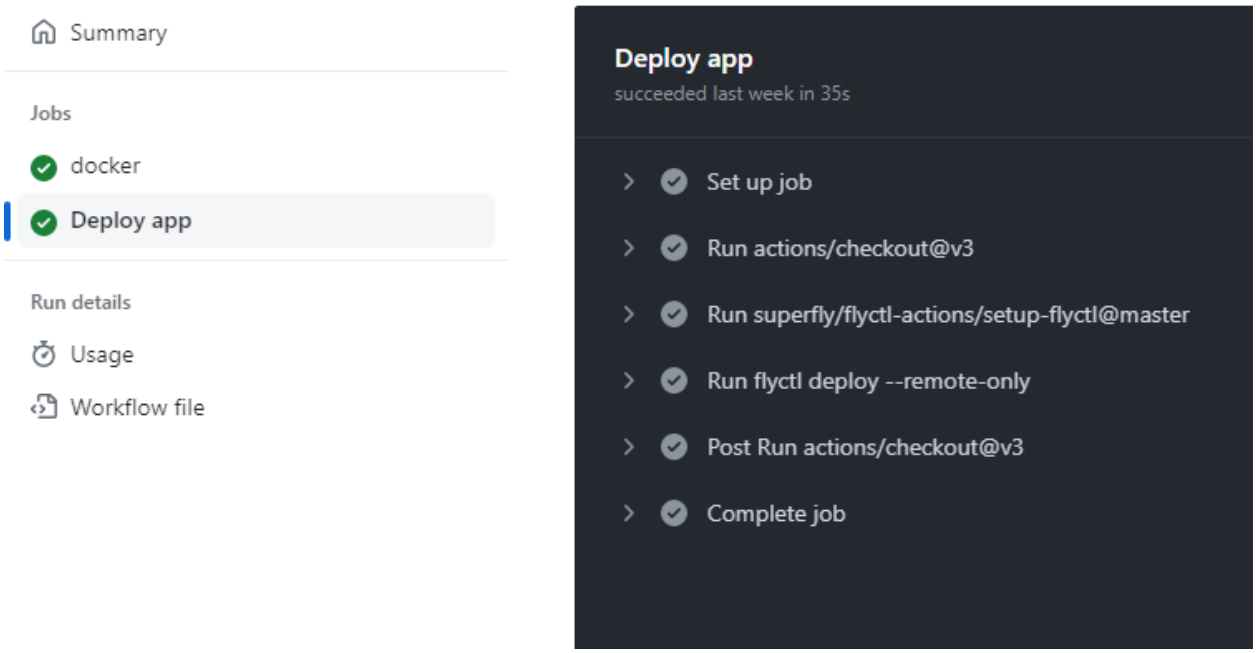


Рисунок 4.5 – Постійна інтеграція і розгортання Fly.io з GitHub Actions

Висновки до розділу

Отже, у даному розділі виконано впровадження та супровід програмного забезпечення:

– розгорнуто веб-застосунок використовуючи Fly.io, Docker та GitHub Actions;
– описано та проілюстровано методи постійної інтеграції та розгортання програмного забезпечення;
– забезпечено підтримку програмного забезпечення.

ВИСНОВКИ

В результаті виконання дипломного проєкту було спроєктовано архітектурне рішення веб-застосунку для цифрової дистрибуції, що спрощує написання логіки, налаштування та централізації стану додатку, підтримує асинхронні запити та нормалізацію даних. Таким чином, досягнуто оптимізоване керування станом додатку.

Також, розроблений веб-застосунок реалізує усі задачі, поставлені в дипломному проєктуванні, дозволяючи користувачу зручно і швидко реєструватися та авторизуватися, переглядати каталог та деталі продуктів, проводити посторінкову навігацію, сортування, фільтрування та пошук продуктів у каталозі, додавати продукти до кошику та переглядати кошик, проводити чекаут та створювати замовлення, переглядати список та деталі замовлень.

Спочатку було проведено аналіз вимог до програмного забезпечення, далі змодельовано та сконструйовано програмне забезпечення, після цього виконано аналіз якості та тестування програмного забезпечення, і в кінці описано впровадження та супровід програмного забезпечення.

На мою думку, досліджувана тема є актуальною, оскільки вимоги до односторінкових веб-застосунків стають складнішими з кожним роком, потребуючи все більшого керування станом відповідей сервера та інтерфейсу користувача, ніж будь-коли раніше.

Дане архітектурне рішення практично спрощує розробку та підвищує продуктивність керування станом веб-застосунків, в яких великі частини стану додатку потрібні у багатьох компонентах, з частим оновленням стану або із комплексною логікою оновлення стану.

Веб-застосунок можна удосконалити при подальшій розробці, додавши роль адміністратора та реалізувавши CRUD операції для продуктів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Höglund N. Digital distribution of video games for PC: A SWOT analysis [Електронний ресурс] / Niklas Höglund // Arcada – Nylands svenska yrkeshögskola. – 2014. – Режим доступу до ресурсу: https://www.theseus.fi/bitstream/handle/10024/76150/Hoglund_Niklas.pdf.

2) Walker J. RPS Exclusive: Gabe Newell Interview [Електронний ресурс] / John Walker // Rock, Paper, Shotgun. – 2007. – Режим доступу до ресурсу: https://www.rockpapershotgun.com/rps-exclusive-gabe-newell-interview.

3) GOG.com [Електронний ресурс]. – 2008. – Режим доступу до ресурсу: https://www.gog.com/.

4) Prince M. Videogame Publishers Place Big Bets on Big-Budget Games [Електронний ресурс] / M. Prince, P. Roth // The Wall Street Journal Online. – 2004. – Режим доступу до ресурсу: https://www.wsj.com/articles/SB110243451698593254.

5) Digital distribution of video games [Електронний ресурс] // Wikipedia. – 2023. – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Digital_distribution_of_video_games.

6) Initial trust and intentions to buy: The effect of vendor-specific guarantees, customer reviews and the role of online shopping experience [Електронний ресурс] / K. Stouthuysen, I. Teunis, E. Reusen, H. Slabbinck // Electronic Commerce Research and Applications. – 2018. – Режим доступу до ресурсу: https://www.sciencedirect.com/science/article/abs/pii/S1567422317300911#!.

7) Sagandira C. User Interface design in e-commerce and its impact on consumer trust [Електронний ресурс] / C. Sagandira, K. Berg // Jönköping University. – 2020. – Режим доступу до ресурсу: https://www.researchgate.net/publication/344516713_User_Interface_design_in_e-commerce_and_it_impacts_on_consumer_trust.

8) Diorio J. A few tips to speed up your mobile site and tools to test it [Електронний ресурс] / Jon Diorio // Think with Google. – 2018. – Режим доступу до ресурсу: https://www.thinkwithgoogle.com/marketing-strategies/app-and-mobile/mobile-site-speed-tools/.

9) Primitive and flexible state management for React [Електронний ресурс] // Jotai – Режим доступу до ресурсу: https://jotai.org/.

10) A state management library for React [Електронний ресурс] // Recoil – Режим доступу до ресурсу: https://recoiljs.org/.

11) A Predictable State Container for JS Apps [Електронний ресурс] // Redux – Режим доступу до ресурсу: https://redux.js.org/.

12) A small, fast and scalable bearbones state-management solution using simplified flux principles [Електронний ресурс] // Zustand – Режим доступу до ресурсу: https://www.npmjs.com/package/zustand.

13) Client–server model [Електронний ресурс] // Wikipedia – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Client%E2%80%93server_model.

14) Multitier architecture [Електронний ресурс] // Wikipedia – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Multitier_architecture.

15) Visual Studio Code [Електронний ресурс] – Режим доступу до ресурсу: https://code.visualstudio.com/.

16) .NET [Електронний ресурс] – Режим доступу до ресурсу: https://dotnet.microsoft.com/en-us/.

17) ASP.NET Core Web API [Електронний ресурс] – Режим доступу до ресурсу: https://dotnet.microsoft.com/en-us/apps/aspnet/apis.

18) Entity Framework Core [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/ef/core/.

19) React [Електронний ресурс] – Режим доступу до ресурсу: https://react.dev/.

20) PostgreSQL [Електронний ресурс] – Режим доступу до ресурсу: https://www.postgresql.org/.

21) The ultimate destination for playing, discussing, and creating games [Електронний ресурс] // Steam – Режим доступу до ресурсу: https://store.steampowered.com/.

22) Epic Games Store [Електронний ресурс] // EGS – Режим доступу до ресурсу: https://store.epicgames.com/en-US/.

23) General Data Protection Regulation [Електронний ресурс] // Official Journal of the European Union. – 2016. – Режим доступу до ресурсу: https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32016R0679.

24) Introduction to Identity on ASP.NET Core [Електронний ресурс] // Microsoft. – 2022. – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-7.0&tabs=visual-studio.

25) GitHub Actions [Електронний ресурс] // GitHub – Режим доступу до ресурсу: https://github.com/features/actions.

26) About code scanning with CodeQL [Електронний ресурс] // GitHub – Режим доступу до ресурсу: https://docs.github.com/en/code-security/code-scanning/automatically-scanning-your-code-for-vulnerabilities-and-errors/about-code-scanning-with-codeql.

27) Platform for running full-stack applications and databases close to the users [Електронний ресурс] // Fly – Режим доступу до ресурсу: https://fly.io/.

28) Set of platform as a service products that use OS-level virtualization to deliver software in packages called containers [Електронний ресурс] // Docker – Режим доступу до ресурсу: https://www.docker.com/.

ДОДАТОК А ЗВІТ ПОДІБНОСТІ

UNICHECK

Ім'я користувача:
Лісовиченко Олег Іванович

ID перевірки:
1015394124

Дата перевірки:
02.06.2023 14:29:14 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
02.06.2023 14:43:40 EEST

ID користувача:
76913

Назва документа: **ІП-з91_Бубряк_ПЗ**

Кількість сторінок: **99** Кількість слів: **11329** Кількість символів: **85622** Розмір файлу: **5.19 MB** ID файлу: **1015058508**

14.8%
Схожість

Найбільша схожість: **3.92%** з джерелом з Бібліотеки (ID файлу: **1015042084**)

5.98% Джерела з Інтернету	172	Сторінка 101
14.3% Джерела з Бібліотеки	415	Сторінка 103

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

АРХІТЕКТУРНЕ РІШЕННЯ ВЕБ-ЗАСТОСУНКУ ДЛЯ ЦИФРОВОЇ ДИСТРИБУЦІЇ

Текст програми

КПІ.ІП-з9101.045440.03.13

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Катерина ЛІЩУК

Нормоконтроль:
_____ Катерина ЛІЩУК

Виконавець:
_____ Микита БУБРЯК

Київ – 2023

Файл Product.cs

```
namespace API.Entities
{

    // Тип сутності продукту
    public class Product
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Description { get; set; }
        public string Image { get; set; }
        public long Price { get; set; }
        public int QuantityInStock { get; set; }
        public string Genre { get; set; }
        public string Theme { get; set; }
        public string Developer { get; set; }
        public DateOnly ReleaseDate { get; set; }
    }
}
```

Файл orderSlice.ts

```
// Стан додатку для функціональної задачі замовлення
interface OrderState {
    ordersLoaded: boolean;
    status: string
}

// Адаптер сутностей для нормалізації даних множини замовлень у стані
export const ordersAdapter = createEntityAdapter<Order>();

// Додавання до стану списку замовлень, асинхронно отриманих від сервера
export const fetchOrdersAsync = createAsyncThunk<Order[], void>(
    'orders/fetchOrdersAsync',
    async (_, thunkAPI) => {
        try {
            return await agent.Orders.list();
        } catch (error: any) {
            return thunkAPI.rejectWithValue({ error: error.data });
        }
    }
);
```

```
// Додавання до стану одного замовлення, асинхронно отриманого від сервера
export const fetchOrderAsync = createAsyncThunk<Order, number>(
    'orders/fetchOrderAsync',
    async (orderId, thunkAPI) => {
        try {
            return await agent.Orders.details(orderId);
        } catch (error: any) {
            return thunkAPI.rejectWithValue({ error: error.data });
        }
    }
)

// Створення Slice, що містить reducer логіку та actions, пов’язані
// з замовленнями
export const orderSlice = createSlice({
    name: 'orders',
    initialState: ordersAdapter.getInitialState<OrderState>({
        ordersLoaded: false,
        status: 'idle'
    }),
    reducers: {
        // Для контролю індикатора завантаження замовлень
        unloadOrders: (state) => {
            state.ordersLoaded = false;
        }
    },

    // Асинхронна логіка для відстеження стану завантаження виклику API
    extraReducers: (builder => {
        builder.addCase(fetchOrdersAsync.pending, (state, action) => {
            state.status = 'pendingFetchOrders';
        });
        builder.addCase(fetchOrdersAsync.fulfilled, (state, action) => {
            ordersAdapter.setAll(state, action.payload);
            state.status = 'idle';
            state.ordersLoaded = true;
        });
        builder.addCase(fetchOrdersAsync.rejected, (state, action) => {
            console.log(action.payload);
            state.status = 'idle';
        });

        builder.addCase(fetchOrderAsync.pending, (state, action) => {
            state.status = 'pendingFetchOrder';
        });
```

```
        builder.addCase(fetchOrderAsync.fulfilled, (state, action) => {
            ordersAdapter.upsertOne(state, action.payload);
            state.status = 'idle';
        });
        builder.addCase(fetchOrderAsync.rejected, (state, action) => {
            console.log(action.payload);
            state.status = 'idle';
        });
    })
});

// Для використання selectAll та selectById
export const orderSelectors = ordersAdapter.getSelectors((state: RootState) =>
state.orders);

// Для контролю індикатора завантаження замовлень
export const { unloadOrders } = orderSlice.actions;
```

Файл cartSlice.ts

```
// Стан додатку для функціональної задачі кошика
interface CartState {
    cart: Cart | null;
    status: string
}

// Початковий стан
const initialState: CartState = {
    cart: null,
    status: 'idle'
}

// Додавання до стану кошику користувача (якщо такий є), асинхронно
// отриманого від сервера
export const fetchCartAsync = createAsyncThunk<Cart>(
    'cart/fetchCartAsync',
    async (_, thunkAPI) => {
        try {
            return await agent.Cart.get();
        } catch (error: any) {
            return thunkAPI.rejectWithValue({ error: error.data });
        }
    },
    {
```

```
        condition: () => {
            if (!getCookie('customerId')) return false;
        }
    }
)

// Додавання до серверних даних кошику продуктів,
// асинхронно відправлених на сервер
export const addItemToCartAsync = createAsyncThunk<Cart, { productId:
number, quantity?: number }>(
    'cart/addItemToCartAsync',
    async ({ productId, quantity = 1 }, thunkAPI) => {
        try {
            return await agent.Cart.addItem(productId, quantity);
        } catch (error: any) {
            return thunkAPI.rejectWithValue({ error: error.data });
        }
    }
)

// Віднімання з серверних даних кошику продуктів,
// асинхронно відправлених на сервер
export const removeItemFromCartAsync = createAsyncThunk<void, { productId:
number, quantity?: number, name?: string }>(
    'cart/removeItemFromCartAsync',
    async ({ productId, quantity = 1 }, thunkAPI) => {
        try {
            await agent.Cart.removeItem(productId, quantity);
        } catch (error: any) {
            return thunkAPI.rejectWithValue({ error: error.data });
        }
    }
)

// Створення Slice, що містить reducer логіку та actions, пов’язані
// з кошиком
export const cartSlice = createSlice({
    name: 'cart',
    initialState,
    reducers: {

        // Для завантаження кошика при авторизації користувача
        setCart: (state, action) => {
            state.cart = action.payload;
        },
```

```
        // Для розвантаження кошика при створенні замовлення
        // або виході користувача
        resetCart: (state) => {
            state.cart = null;
        }
    },

// Асинхронна логіка для відстеження стану завантаження виклику API
    extraReducers: (builder => {
        builder.addCase(addItemToCartAsync.pending, (state, action) => {
            state.status = 'pendingAddItem' + action.meta.arg.productId;
        });

        builder.addCase(removeItemFromCartAsync.pending, (state, action) => {
            state.status = 'pendingRemoveItem' + action.meta.arg.productId +
action.meta.arg.name;
        });

        // Для відповідності між станом кошику на клієнті
        // та актуальними даними на сервері
        builder.addCase(removeItemFromCartAsync.fulfilled, (state, action) => {
            const { productId, quantity = 1 } = action.meta.arg;
            const itemIndex = state.cart?.items.findIndex(i => i.productId ===
productId);
            if (itemIndex === undefined || itemIndex === -1) return;

            state.cart!.items[itemIndex].quantity -= quantity!;
            if (state.cart!.items[itemIndex].quantity === 0) {
                state.cart?.items.splice(itemIndex, 1);
            }
            state.status = 'idle';
        })
        builder.addCase(removeItemFromCartAsync.rejected, (state, action) => {
            console.log(action.payload);
            state.status = 'idle';
        });

        builder.addMatcher(isAnyOf(addItemToCartAsync.fulfilled,
fetchCartAsync.fulfilled), (state, action) => {
            state.cart = action.payload;
            state.status = 'idle';
        });
        builder.addMatcher(isAnyOf(addItemToCartAsync.rejected,
fetchCartAsync.rejected), (state, action) => {
```

```
            console.log(action.payload);
            state.status = 'idle';
        });
    })
})

// Для завантаження кошика при авторизації користувача, та розвантаження
// кошика при створенні замовлення або виході користувача
export const { setCart, resetCart } = cartSlice.actions;
```

Файл CartController.cs

```
namespace API.Controllers
{
    public class CartController : BaseApiController
    {
        private readonly StoreContext _context;

        // Ін’єкція контексту бази даних
        public CartController(StoreContext context)
        {
            _context = context;
        }

        // Кінцева точка для отримання даних кошику користувача
        [HttpGet(Name = "GetCart")]
        public async Task<ActionResult<CartDTO>> GetCart()
        {
            var cart = await LoadCart(GetCustomerId());

            if (cart == null) return NotFound();
            return cart.MapCartToDTO();
        }

        // Кінцева точка для створення чи додавання
        // предмета кошику до кошику користувача
        [HttpPost]
        public async Task<ActionResult<CartDTO>> AddItemToCart(int productId,
int quantity)
        {
            var cart = await LoadCart(GetCustomerId());
            if (cart == null) cart = CreateCart();

            var product = await _context.Products.FindAsync(productId);
```

```
            if (product == null) return BadRequest(new ProblemDetails { Title =
"Couldn't find this item to add to cart." });

            cart.AddItem(product, quantity);

            var result = await _context.SaveChangesAsync();
            if (result > 0) return CreatedAtRoute("GetCart", cart.MapCartToDTO());

            return BadRequest(new ProblemDetails { Title = "Couldn't add item to
cart" });
        }

        // Кінцева точка для видалення чи віднімання
        // предмета кошику з кошику користувача
        [HttpDelete]
        public async Task<ActionResult> RemoveItemFromCart(int productId, int
quantity)
        {
            var cart = await LoadCart(GetCustomerId());
            if (cart == null) return NotFound();

            cart.RemoveItem(productId, quantity);

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

            return BadRequest(new ProblemDetails { Title = "Couldn't remove item
from cart" });
        }

        // Створення кошику користувача
        private Cart CreateCart()
        {
            var customerId = User.Identity?.Name;
            if (string.IsNullOrEmpty(customerId))
            {
                customerId = Guid.NewGuid().ToString();
                var cookieOptions = new CookieOptions { IsEssential = true, Expires =
DateTime.Now.AddDays(30) };
                Response.Cookies.Append("customerId", customerId, cookieOptions);
            }

            var cart = new Cart { CustomerId = customerId };
            _context.Carts.Add(cart);
```

```
            return cart;
        }

        // Завантаження кошику користувача
        private async Task<Cart> LoadCart(string customerId)
        {
            if (string.IsNullOrEmpty(customerId))
            {
                Response.Cookies.Delete("customerId");
                return null;
            }
            return await _context.Carts.LoadCart(customerId);
        }

        // Отримання унікального ідентифікатора покупця
        // (GUID отриманий з cookie для гостя,
        // або UserName зареєстрованого користувача)
        private string GetCustomerId()
        {
            return User.Identity?.Name ?? Request.Cookies["customerId"];
        }
    }
}
```

Файл ProductDetails.tsx

```
// Компонент деталей продукту
export default function ProductDetails() {

    // Отримання Id продукту з URL
    const { id } = useParams<{ id: string }>();

    // Компоненти React викликають action dispatch для оновлення store
    // за допомогою хука useDispatch та передачі action creator до dispatch
    const dispatch = useAppDispatch();

    // Компоненти React зчитують дані стану додатку зі store
    // за допомогою хука useSelector
    const product = useAppSelector(state => productSelectors.selectById(state,
id!));
    const { cart, status: cartStatus } = useAppSelector(state => state.cart);
    const { status: productStatus } = useAppSelector(state => state.catalog);

    // Локальний стан інтерфейсу компоненту
    const [quantity, setQuantity] = useState(1);
```

```
    const item = cart?.items.find(i => i.productId === product?.id);

    // Ініціалізація даних для інтерфейсу кількості продуктів
    // та окреме завантаження продукту у випадку
    // відсутності множини продуктів
    useEffect(() => {
        if (item) setQuantity(item.quantity);
        if (!product) dispatch(fetchProductAsync(parseInt(id!)));
    }, [dispatch, product, id, item])

   // Для введення кількості продуктів
    function changeQuantityInput(event: any) {
        if (event.target.value >= 0) setQuantity(parseInt(event.target.value));
    }

    // Оновлення кошику натисканням кнопки зміни кількості
    // продуктів у кошику
    function updateCartInput() {
        if (!item || quantity > item.quantity) {
            const newQuantity = item ? quantity - item.quantity : quantity;
            dispatch(addItemToCartAsync({ productId: product?.id!, quantity:
newQuantity }))
        } else {
            const newQuantity = item.quantity - quantity;
            dispatch(removeItemFromCartAsync({ productId: product?.id!, quantity:
newQuantity }))
        }
    }

    // Відображення компонента з індикатором завантаження
    // при завантаженні продукту у випадку відсутності множини продуктів
    if (productStatus.includes('pending')) return <LoadingComponent
message='Loading product...' />

    // Відображення помилки 404 Not Found при переході
    // на сторінку продукту з неіснуючим ID
    if (!product) return <NotFound />

    // Повернення tsx елементу з деталями продукту
    return (
        <Container component={Paper} sx={{ pt: 5, pb: 5 }}>
            <Grid container spacing={6}>
                <Grid item xs={6}>
                    <img src={product.image} alt={product.name} style={{ width:
'100%' }} />
```

```
            <Typography variant="h4" sx={{ fontWeight: 'bold', pt: 5, pb: 5
}}>{product.name}</Typography>
          </Grid>

          <Grid item xs={6}>
            <TableContainer>
              <Table>
                <TableBody>
                  <TableRow>
                    <TableCell sx={{ fontWeight: 'bold'
}}>Description</TableCell>
                    <TableCell>{product.description}</TableCell>
                  </TableRow>
                  <TableRow>
                    <TableCell sx={{ fontWeight: 'bold'
}}>Developer</TableCell>
                    <TableCell>{product.developer}</TableCell>
                  </TableRow>
                  <TableRow>
                    <TableCell sx={{ fontWeight: 'bold' }}>Release
date</TableCell>
                    <TableCell>{product.releaseDate.toString()}</TableCell>
                  </TableRow>
                  <TableRow>
                    <TableCell sx={{ fontWeight: 'bold' }}>Genre</TableCell>
                    <TableCell>{product.genre}</TableCell>
                  </TableRow>
                  <TableRow>
                    <TableCell sx={{ fontWeight: 'bold'
}}>Theme</TableCell>
                    <TableCell>{product.theme}</TableCell>
                  </TableRow>
                  <TableRow>
                    <TableCell sx={{ fontWeight: 'bold' }}>Quantity in
stock</TableCell>
                    <TableCell>{product.quantityInStock}</TableCell>
                  </TableRow>
                </TableBody>
              </Table>
            </TableContainer>
          </Grid>
        </Grid>

        <Grid container spacing={2} sx={{ pt: 4 }}>
          <Grid item xs={6}>
```

```
                <Typography variant="h4"
color='success'>{currencyFormat(product.price)}</Typography>
            </Grid>
            <Grid item xs={3}>
              <TextField onChange={changeQuantityInput}
                fullWidth variant="outlined" type="number" label="Quantity"
value={quantity} />
            </Grid>
            <Grid item xs={3}>
              <LoadingButton onClick={updateCartInput}
                loading={cartStatus.includes('pending')}
                disabled={item?.quantity === quantity || (!item && quantity ===
0)}
                fullWidth variant="contained" size='large' sx={{ height: '55px' }}
color='success' >
                {item ? 'Change quantity' : 'Add to cart'}
              </LoadingButton>
            </Grid>
          </Grid>
        </Container>
    )
}
```

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

АРХІТЕКТУРНЕ РІШЕННЯ ВЕБ-ЗАСТОСУНКУ ДЛЯ ЦИФРОВОЇ ДИСТРИБУЦІЇ

Програма та методика тестування

КПІ.ІП-з9101.045440.04.51

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Катерина ЛІЩУК

Нормоконтроль:
_____ Катерина ЛІЩУК

Виконавець:
_____ Микита БУБРЯК

Київ – 2023

ЗМІСТ

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Об’єктом випробування є веб-застосунок для цифрової дистрибуції, що втілює трирівневу архітектуру та складається з односторінкового застосунку (SPA) на рівні представлення, REST API на рівні сервера з бізнес-логікою, та моделі даних сутностей з Code First підходом, створеної з використанням структури об'єктно-реляційного відображення (ORM) та бази даних на рівні доступу до даних.

2 МЕТА ТЕСТУВАННЯ

Метою тестування є наступне:

- перевірка правильності роботи програмного забезпечення відповідно до функціональних вимог:
  - реєстрації користувача;
  - авторизації користувача;
  - перегляду каталогу продуктів;
  - перегляду деталей продукту;
  - посторінкової навігації у каталозі;
  - сортування продуктів у каталозі;
  - фільтрування продуктів у каталозі;
  - пошуку продуктів у каталозі;
  - додавання продукту до кошику;
  - перегляду кошику;
  - чекауту;
  - створення замовлення;
  - перегляду списку замовлень;
  - перегляду деталей замовлення;
- перевірка збереження даних;
- перевірка виведення повідомлень про помилки;
- знаходження проблем, помилок і недоліків з метою їх усунення;
- перевірка зручності графічного інтерфейсу.

3 МЕТОДИ ТЕСТУВАННЯ

Для тестування програмного забезпечення використовуються такі методи:

- статичне тестування – перевіряється програма разом з усією документацією, яка аналізується на предмет дотримання стандартів програмування;
- динамічне тестування – застосовується в процесі виконання програми. Коректність програмного засобу перевіряється на певній кількості тестів. При прогоні кожного з них збираються та аналізуються дані про проблеми та помилки в роботі програми;
- функціональне тестування – полягає у перевірці відповідності реальної поведінки програмного забезпечення очікуваній;
- системне тестування – перевіряється усе програмне забезпечення в цілому;
- мануальне тестування – тестування без використання автоматизації, тест-кейси пише особа, що тестує програмне забезпечення;
- тестування «чорної скриньки» – об’єктом тестування тут є функції присутні у програмі. Перевіряється коректність вихідних даних при заданих вхідних;
- тестування «білої скриньки» – об’єктом тестування тут є внутрішня поведінка програми. Перевіряється коректність побудови всіх елементів програми та правильність їхньої взаємодії один з одним;
- тестування «сірої скриньки» – об’єктом тестування тут є деякі особливості внутрішньої поведінки програми. Перевіряється коректність вихідних даних при заданих вхідних, застосовується для тестування окремих алгоритмів (функцій).

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування виконується мануально з використанням пакету Swashbuckle Swagger та розширення браузеру Redux DevTools, з метою знаходження помилок та недоліків як у функціональній частині програмного забезпечення так і в зручності користування.

Swagger застосовано як набір інструментів для розробки API, що допомагає створювати, документувати, тестувати та використовувати RESTful веб-сервіси. Використовувався для тестування API інтерфейсів та клієнтських запитів.

Redux DevTools застосовано як засіб для налагодження змін стану програми у браузері та покращення процесу розробки з використанням Redux.

Для того, щоб перевірити працездатність та відмовостійкість застосунку, необхідно провести наступні тестування:

- тестування на відповідність функціональним вимогам;
- тестування на відповідність нефункціональним вимогам;
- тестування на виведення повідомлень про помилку валідації введених даних;
- тестування на виведення повідомлень про помилку з кодами стану HTTP 400 Bad Request, 401 Unauthorized, 404 Not Found і 500 Internal Server Error;
- тестування інтерфейсу користувача;
- тестування зручності використання.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

АРХІТЕКТУРНЕ РІШЕННЯ ВЕБ-ЗАСТОСУНКУ ДЛЯ ЦИФРОВОЇ ДИСТРИБУЦІЇ

Керівництво користувача

КПІ.ІП-з9101.045440.05.34

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Катерина ЛІЩУК

Нормоконтроль:
_____ Катерина ЛІЩУК

Виконавець:
_____ Микита БУБРЯК

Київ – 2023

ЗМІСТ

1 ПРИЗНАЧЕННЯ ПРОГРАМИ

«DD Web App» – це веб-застосунок для цифрової дистрибуції, розроблений з оптимізованим керуванням стану додатку та призначений для широкого кола користувачів, що бажають зручно та швидко обрати та придбати цифрові товари.

Користувач має можливість реєструватися та авторизуватися, переглядати каталог та деталі продуктів, проводити посторінкову навігацію, сортування, фільтрування та пошук продуктів у каталозі, додавати продукти до кошику та переглядати кошик, проводити чекаут та створювати замовлення, переглядати список та деталі замовлень.

2 ПІДГОТОВКА ДО РОБОТИ З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ

2.1 Системні вимоги для коректної роботи

Для успішної роботи даного застосунку необхідне виконання наступних вимог:

- програмне забезпечення повинно функціонувати на IBM-сумісних персональних комп‘ютерах;
- мінімальна конфігурація технічних засобів:
  - тип процесору: Intel Core i5;
  - об‘єм ОЗП: 4 Гб;
  - підключення до мережі Інтернет зі швидкістю від 20 мегабіт;
- рекомендована конфігурація технічних засобів:
  - тип процесору: Intel Core i7;
  - об‘єм ОЗП: 8 Гб;
  - підключення до мережі Інтернет зі швидкістю від 100 мегабіт;
- програмне забезпечення повинно працювати під управлінням операційних систем сімейства Windows та підтримувати браузер Google Chrome.

2.2 Завантаження застосунку

Для завантаження застосунку необхідно відкрити браузер, ввести URL адресу веб-застосунку в адресний рядок та натиснути Enter. Також можна перейти за посиланням, що веде до даного веб-застосунку.

2.3 Перевірка коректної роботи

Після переходу до веб-застосунку за допомогою адресного рядка браузера або за посиланням повинна завантажитись та відобразитись початкова сторінка веб-застосунку.

У разі, якщо сторінка веб-застосунку некоректно відображається, користувач може оновити сторінку, перевіривши введену адресу, використане посилання або підключення до мережі Інтернет.

3 ВИКОНАННЯ ПРОГРАМИ

Після завантаження веб-застосунку користувачу буде відображено початкову сторінку (рисунок 3.1).

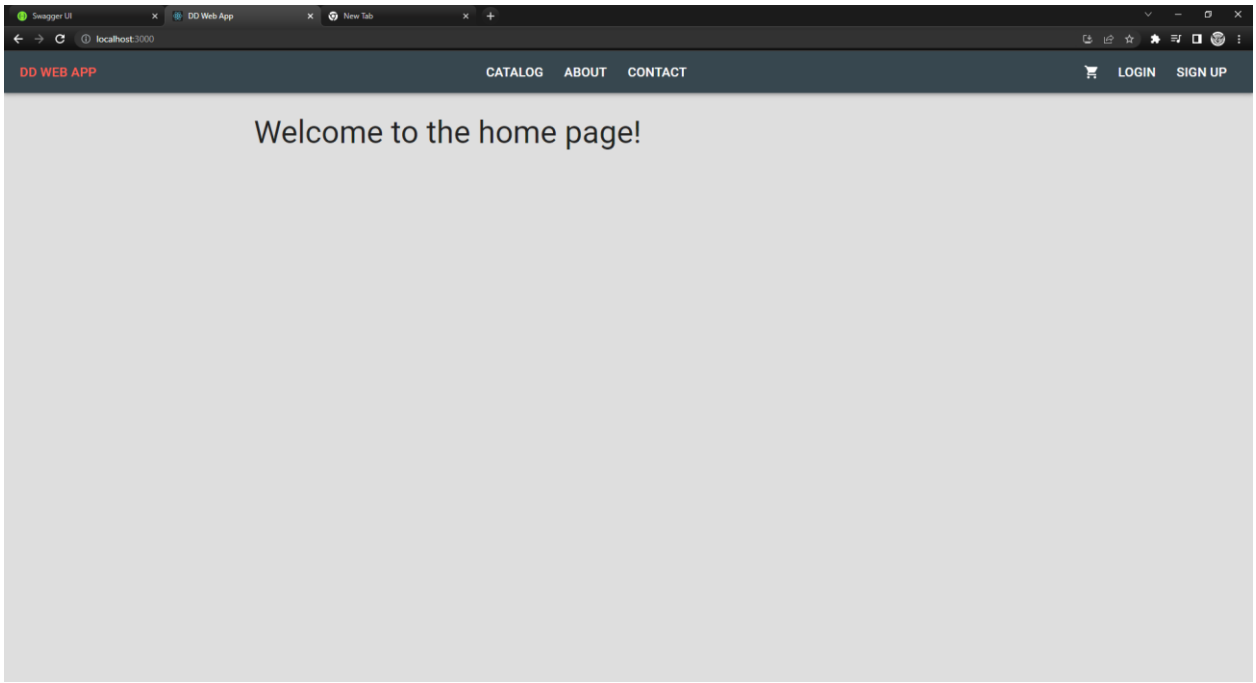


Рисунок 3.1 – Початкова сторінка додатку

Користувач може зареєструватися, перейшовши на сторінку реєстрації (рисунок 3.2). В поля для реєстрації вводяться відповідні дані: пошта користувача, ім’я користувача та пароль в системі. У випадку введення некоректних даних, кнопка реєстрації залишається неактивною. Якщо якесь конкретне поле введено некоректно, то воно підсвічується червоним надписом.

Після заповнення даних користувач натискає кнопку реєстрації. Після цього з’являється повідомлення про успішну реєстрацію, і користувач перенаправляється на сторінку входу.

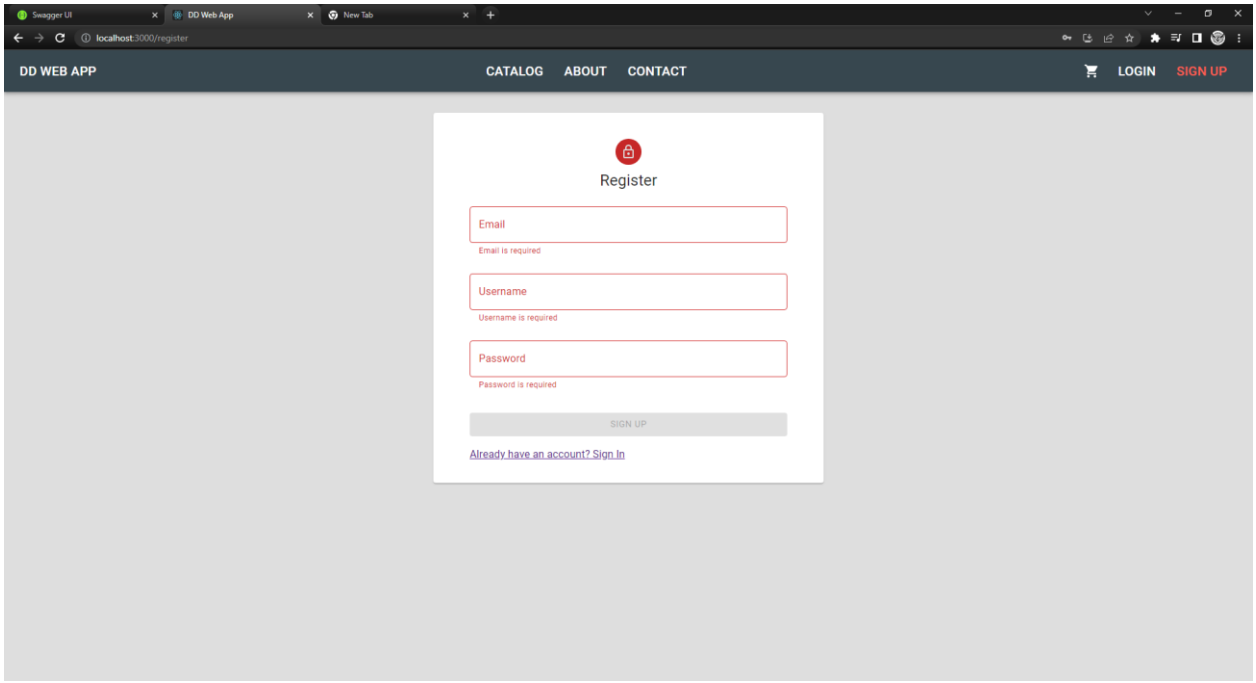


Рисунок 3.2 – Реєстрація користувача

Користувач може авторизуватися, перейшовши на сторінку авторизації (рисунок 3.3). В поля для авторизації вводяться відповідні дані: ім’я користувача та пароль в системі. У випадку введення некоректних даних користувач отримає помилку Unauthorized. Після заповнення даних користувач натискає кнопку авторизації. Після цього користувач перенаправляється на сторінку каталогу.

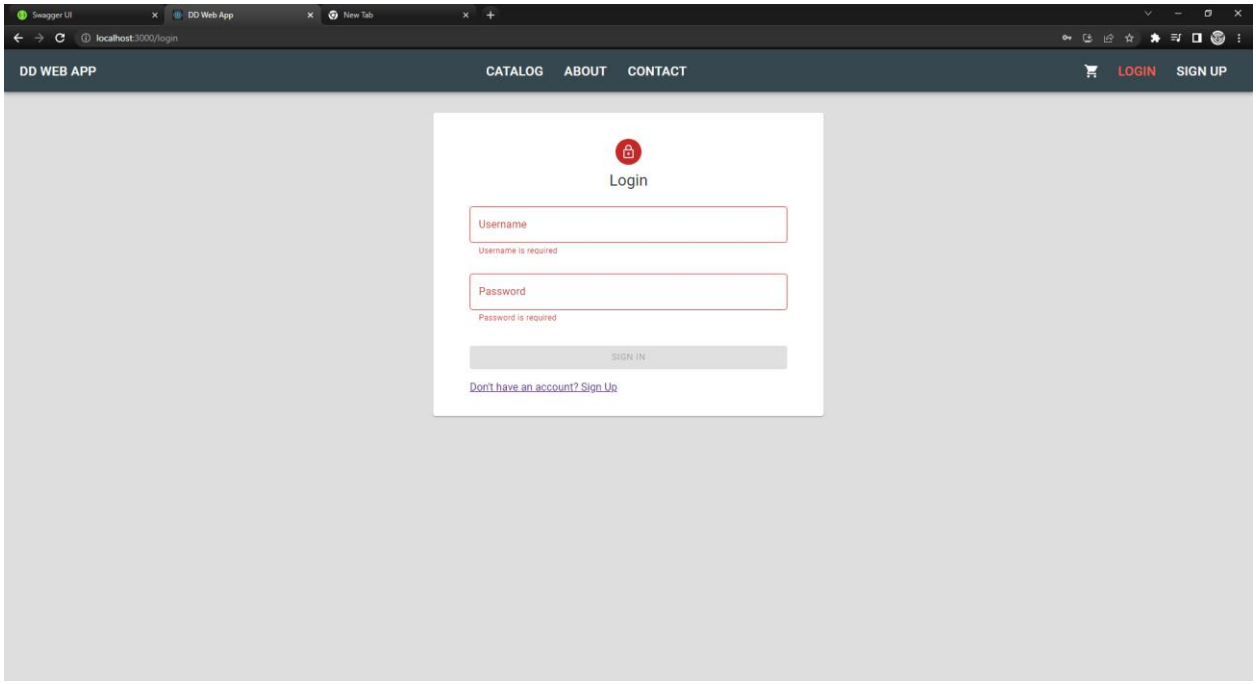


Рисунок 3.3 – Авторизація користувача

Користувач може переглянути каталог продуктів, перейшовши на сторінку каталогу (рисунок 3.4). При завантаженні продуктів відображатиметься компонент з індикатором завантаження.

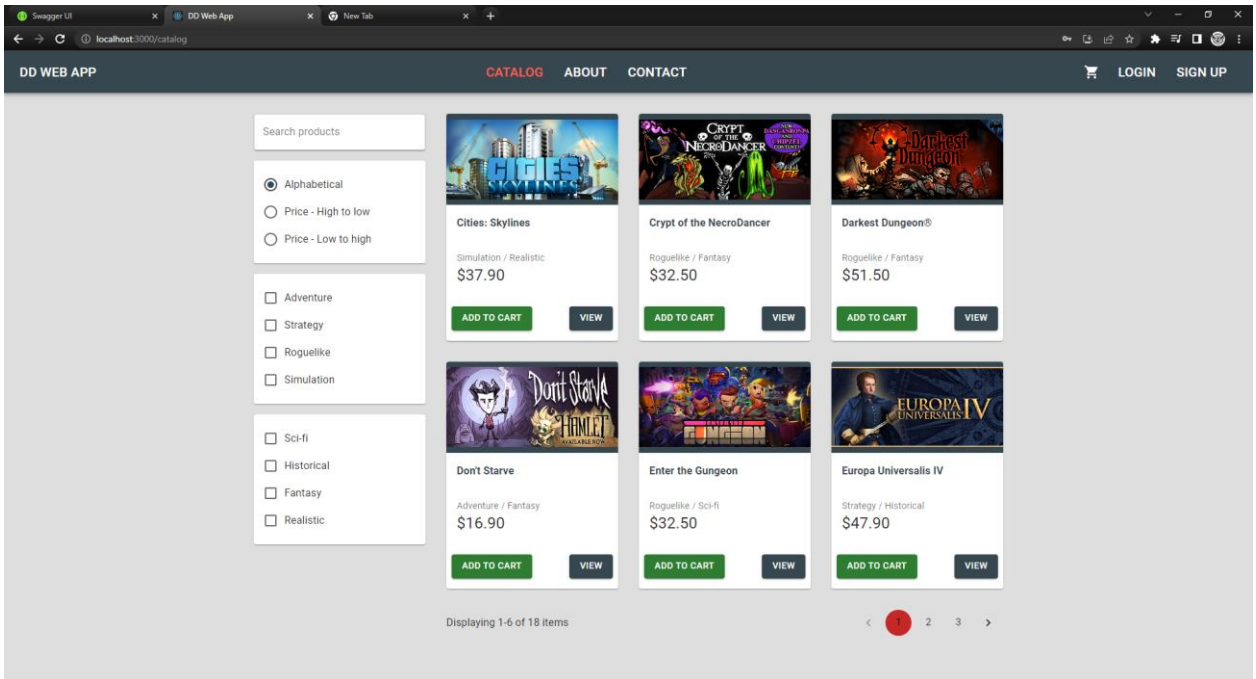


Рисунок 3.4 – Перегляд каталогу продуктів

Користувач може використати посторінкову навігацію у каталозі, обираючи сторінку, натискаючи на номер сторінки або на кнопки навігації вперед або назад. Після цього користувачу відображається обрана сторінка каталогу (рисунок 3.5).

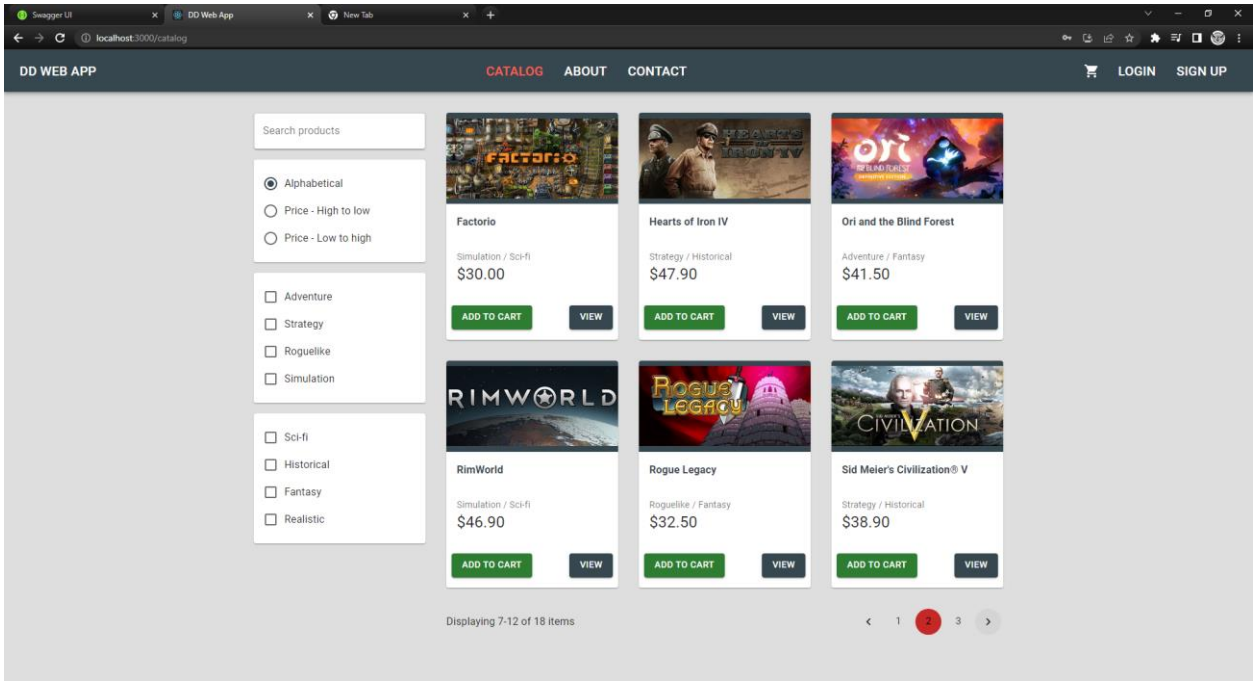


Рисунок 3.5 – Посторінкова навігація у каталозі

Користувач може використати сортування продуктів у каталозі, обравши спосіб сортування, натискаючи на радіокнопки зі способами сортування. Після цього користувачу відображається відсортована сторінка каталогу (рисунок 3.6).

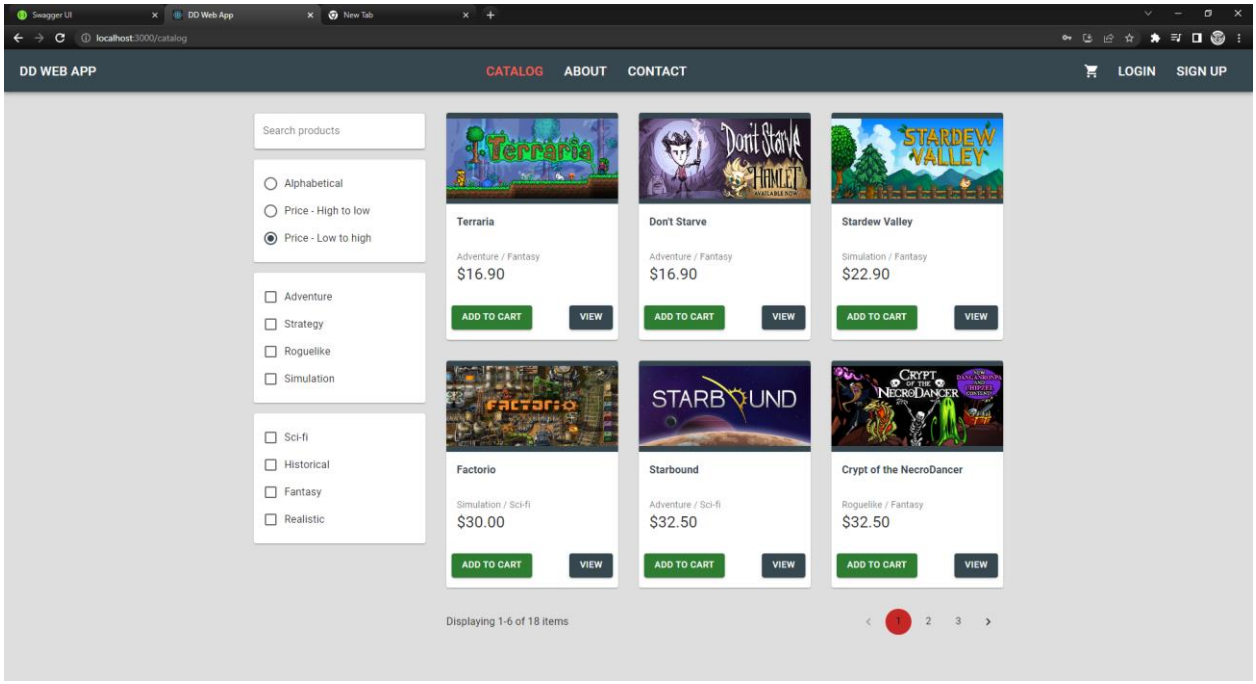


Рисунок 3.6 – Сортування продуктів у каталозі

Користувач може використати фільтрування продуктів у каталозі, обравши спосіб фільтрування, натискаючи на прапорці зі способами фільтрування. Після цього користувачу відображається відфільтрована сторінка каталогу (рисунок 3.7).

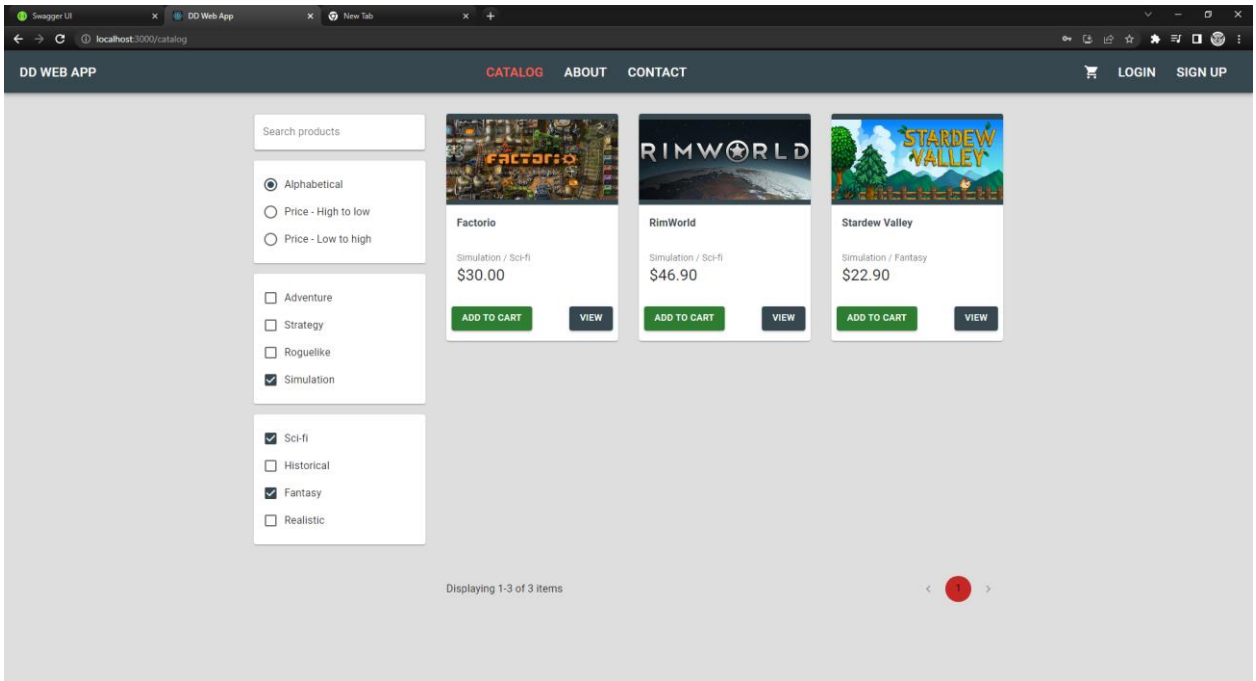


Рисунок 3.7 – Фільтрування продуктів у каталозі

Користувач може використати пошук продуктів у каталозі, обравши термін пошуку, вводячи слова у текстове поле. Після цього користувачу відображається сторінка каталогу з результатами пошуку (рисунок 3.8).

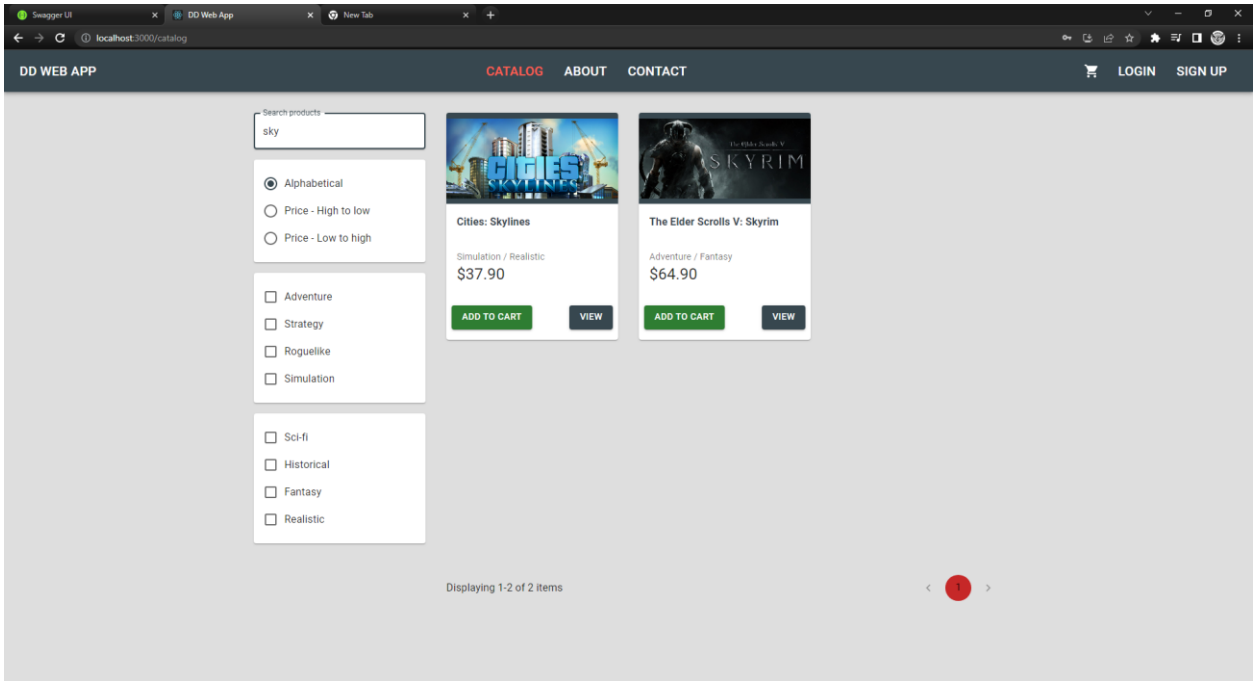


Рисунок 3.8 – Пошук продуктів у каталозі

Користувач може переглянути деталі продукту, перейшовши на сторінку деталей продукту (рисунок 3.9). Веб-застосунок відображає деталі продукту. У випадку переходу на сторінку неіснуючого продукту користувач отримає помилку 404 (рисунок 3.16). У випадку вибору нуля продуктів для додавання в кошик кнопка додавання в кошик стане неактивною.

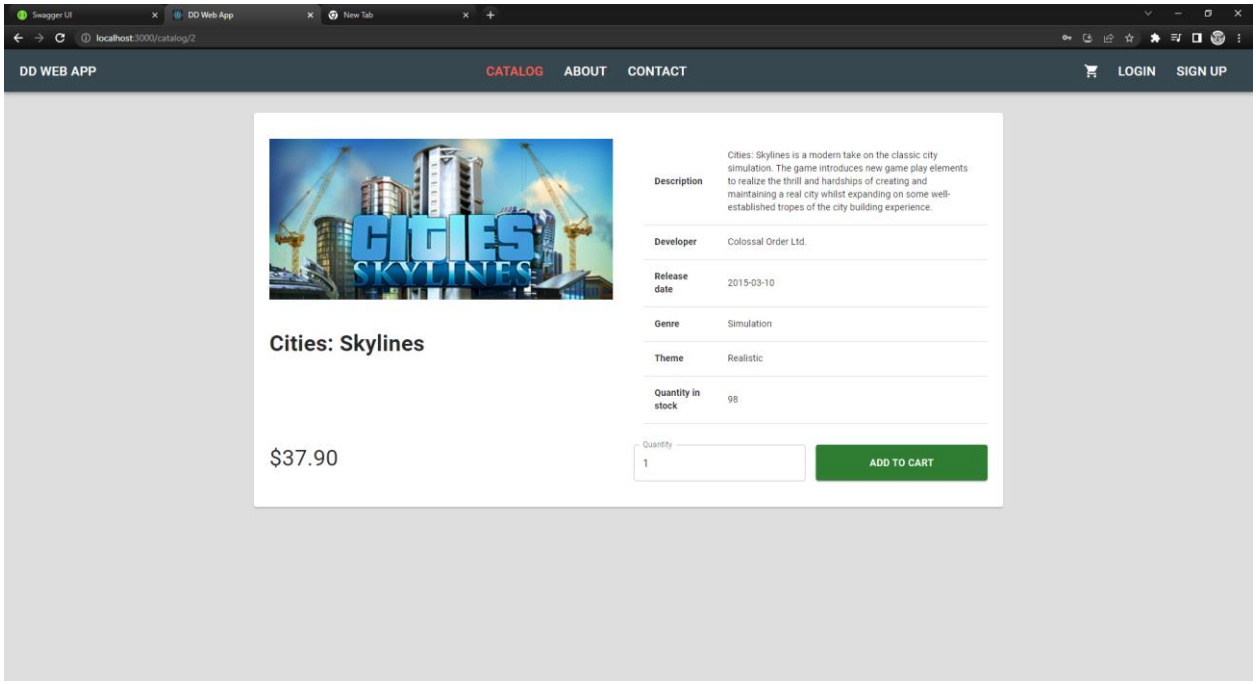


Рисунок 3.9 – Перегляд деталей продукту

Користувач може додати продукт до свого кошику, натиснувши на відповідну кнопку. Знаходячись на сторінці деталей продукту, користувач може змінити кількість продукту в кошику. Після цього значок кошику відображає загальну кількість продуктів у кошику (рисунок 3.10).

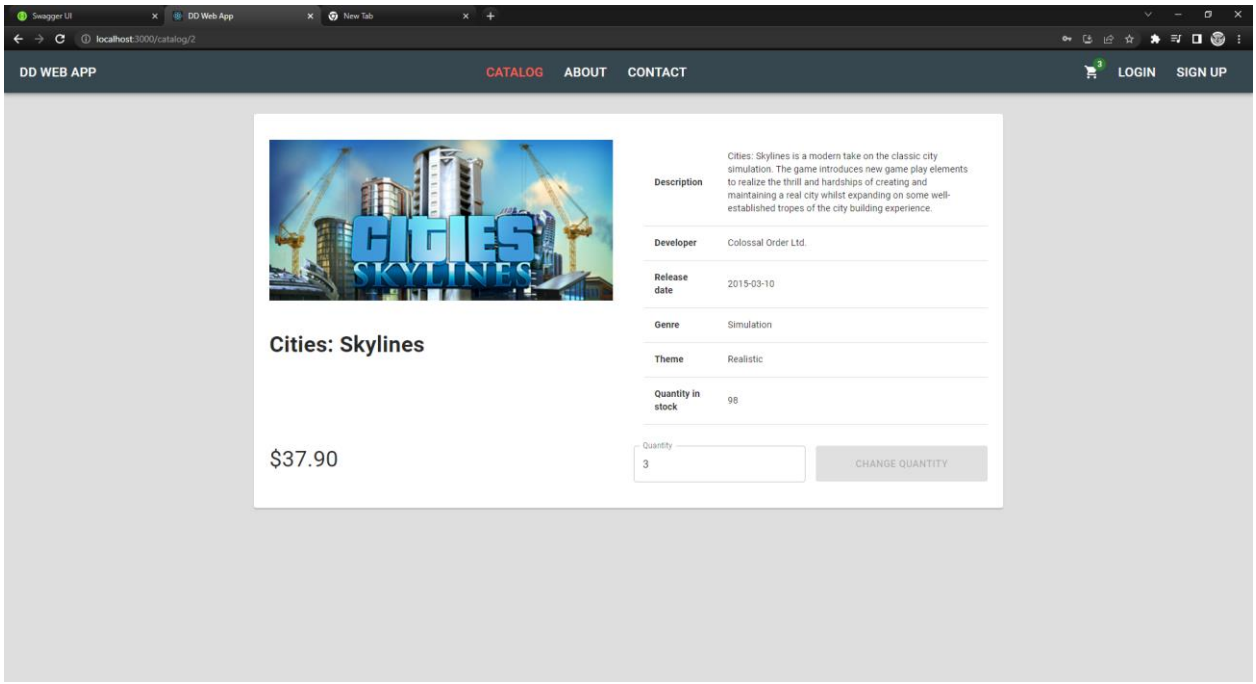


Рисунок 3.10 – Додавання продукту до кошику

Користувач може переглянути вміст свого кошику, перейшовши на сторінку кошику (рисунок 3.11). Якщо користувач ще не додавав продукти до кошику, відображатиметься інформація про відсутність продуктів в кошику. Користувач має змогу використовувати кнопки додавання, віднімання та видалення продуктів з кошику для відповідних дій.

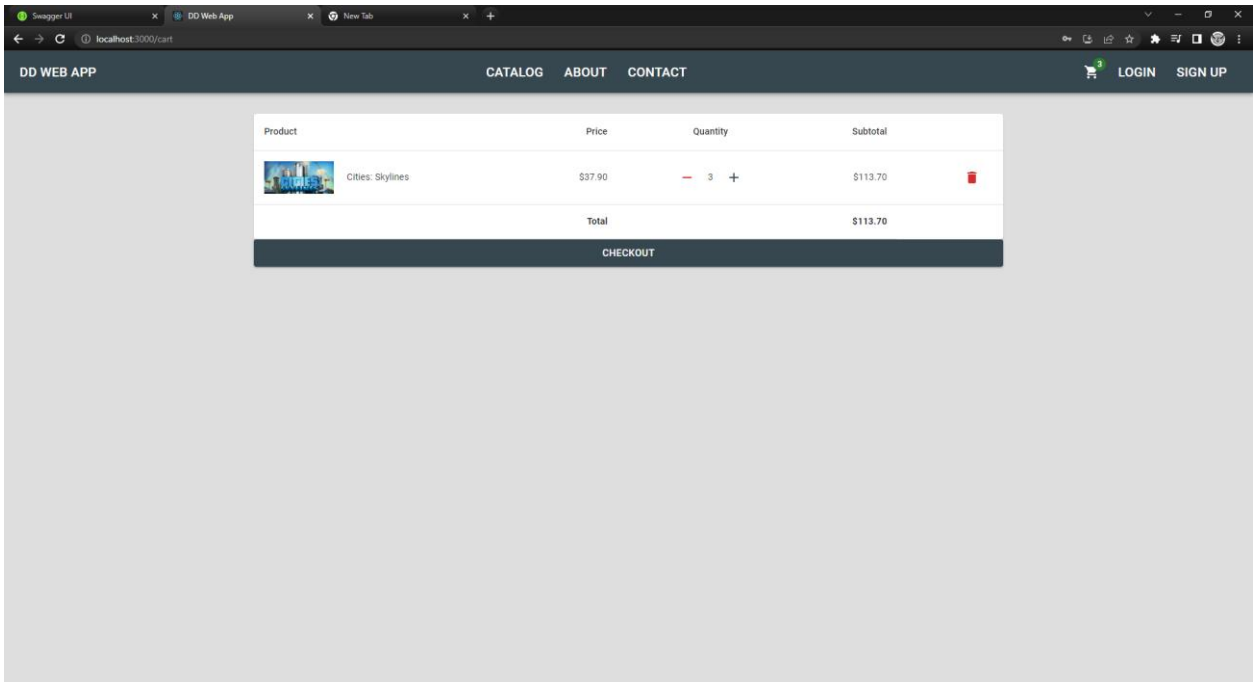


Рисунок 3.11 – Перегляд кошику

Користувач може виконати чекаут, авторизувавшись та впевнившись у вмісті та сумі кошику, після чого перейшовши на сторінку чекауту (рисунок 3.12).

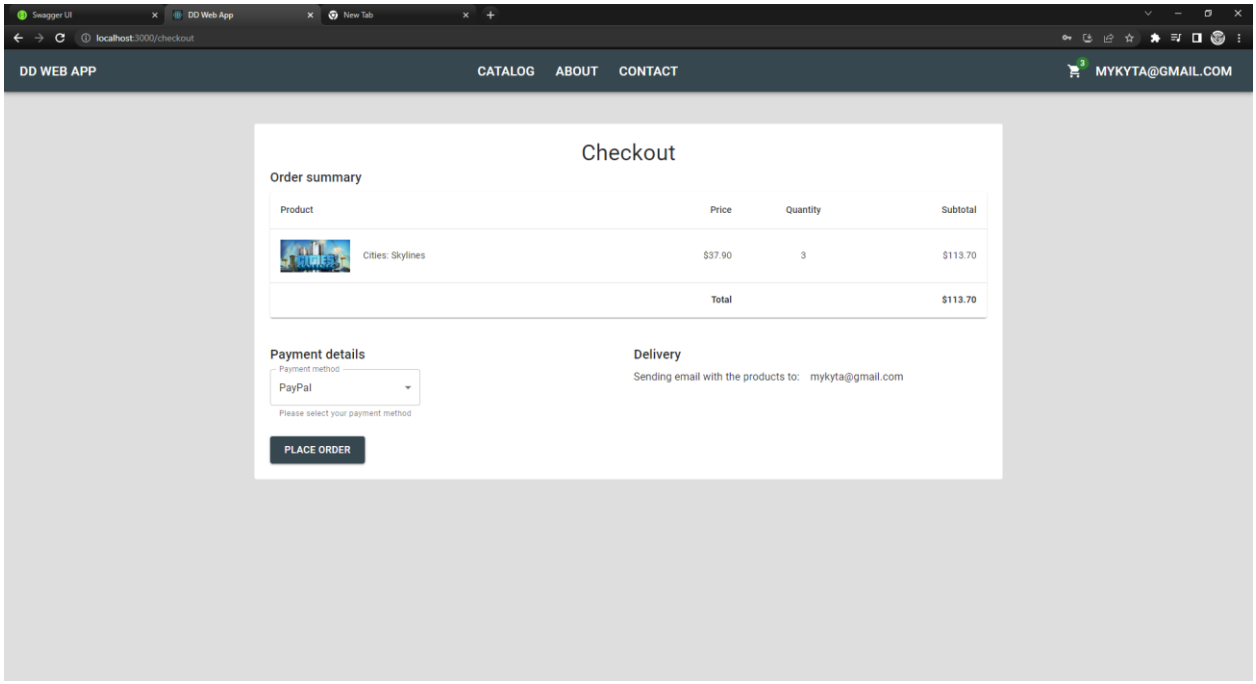


Рисунок 3.12 – Чекаут

Користувач може створити замовлення, авторизувавшись та переглянувши звіт замовлення з інформацією про обрані продукти, їх ціну та спосіб доставки. Користувач обирає метод оплати з випадаючого меню та натискає кнопку створення замовлення. В випадку відсутності обраного методу оплати користувачеві відображається повідомлення про відсутній метод оплати. При успішному створенні замовлення відображено його номер (рисунок 3.13).

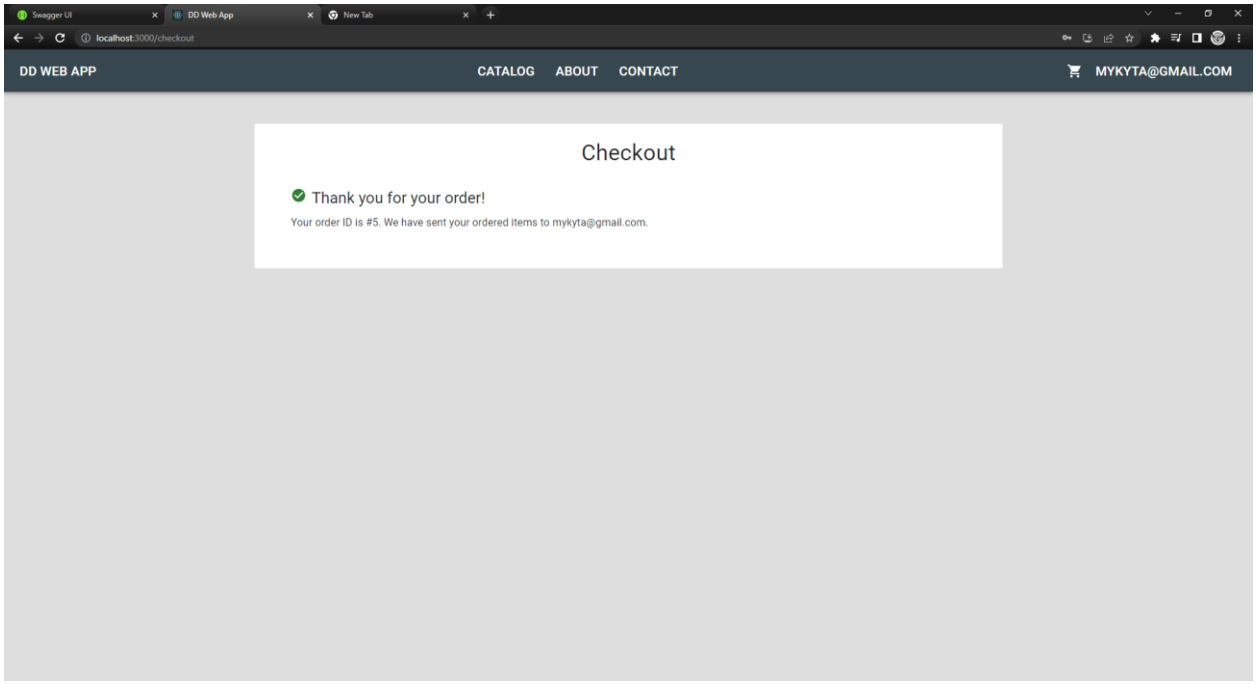


Рисунок 3.13 – Створення замовлення

Користувач може переглянути свій список замовлень, авторизувавшись та перейшовши на сторінку списку замовлень (рисунок 3.14). При завантаженні замовлень відображається компонент з індикатором завантаження.

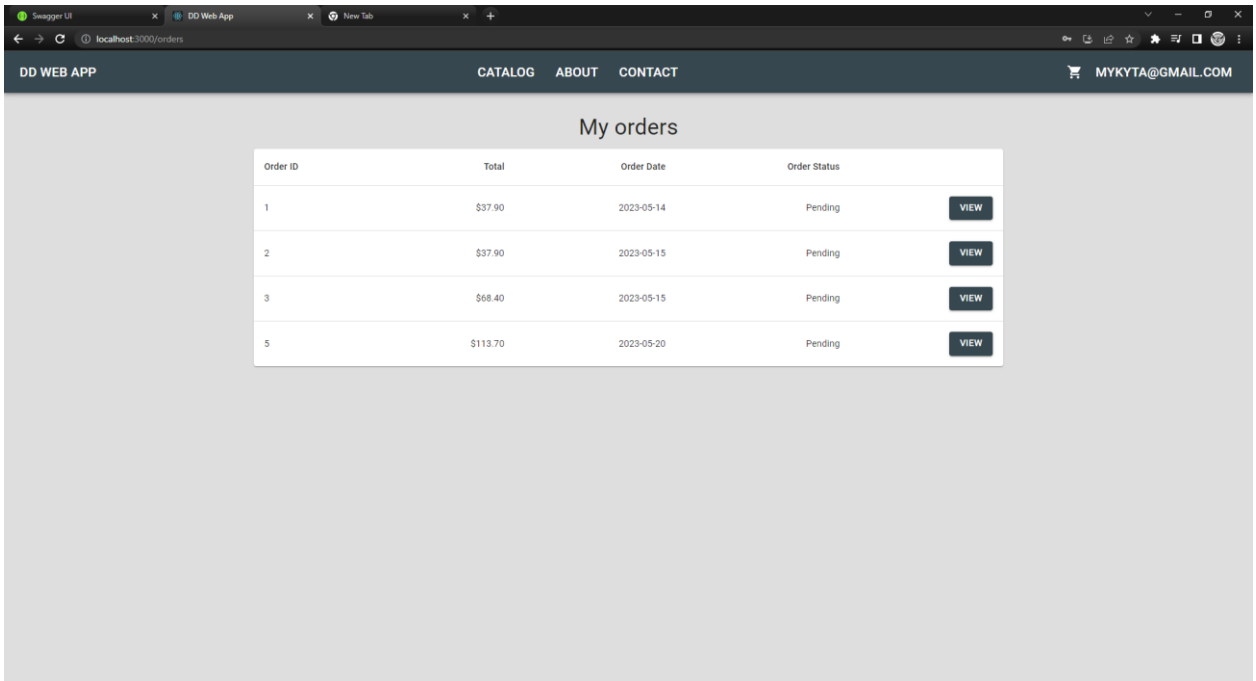


Рисунок 3.14 – Перегляд списку замовлень

Користувач може переглянути деталі свого замовлення, авторизувавшись та перейшовши на сторінку деталей замовлення (рисунок 3.15). У випадку переходу на сторінку неіснуючого замовлення користувач отримає помилку 404 (рисунок 3.16).

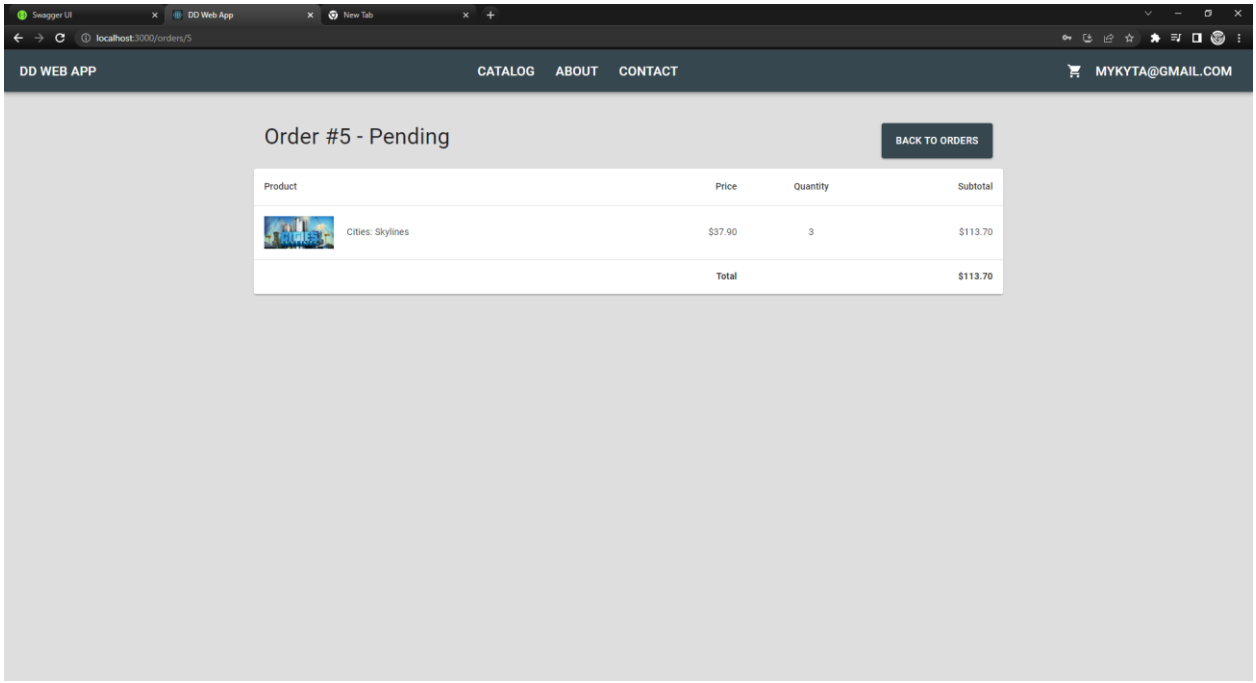


Рисунок 3.15 – Перегляд деталей замовлення

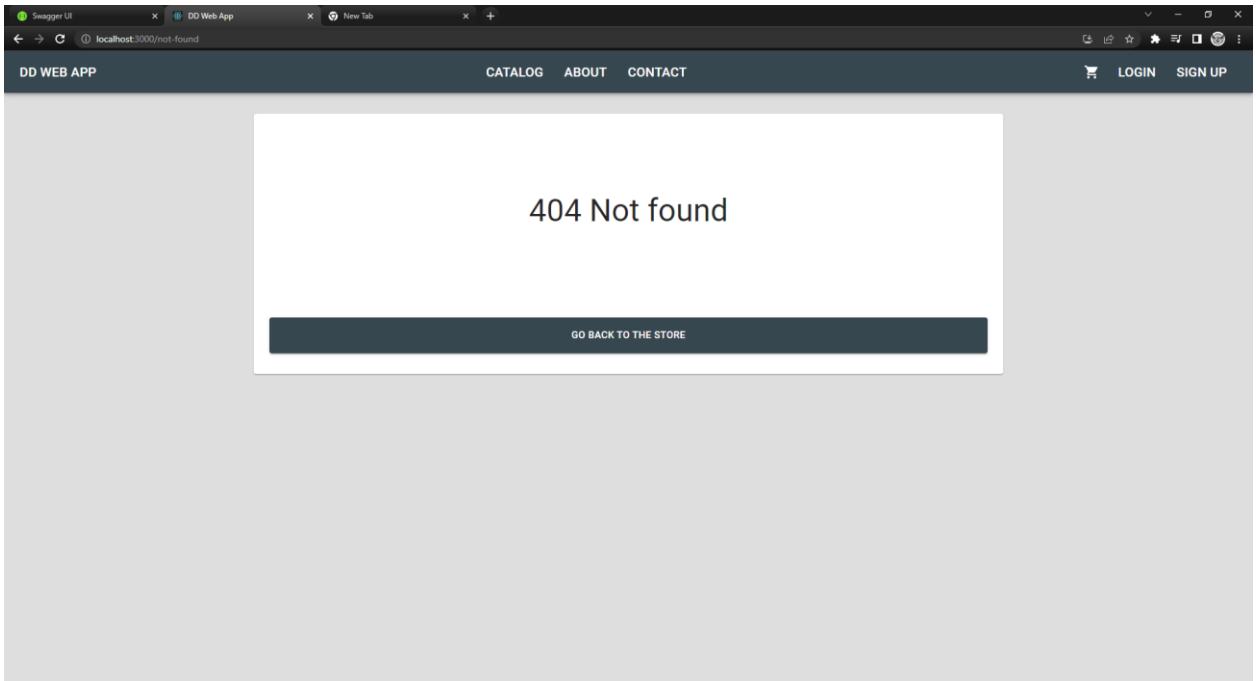


Рисунок 3.16 – Помилка 404 Not Found

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

АРХІТЕКТУРНЕ РІШЕННЯ ВЕБ-ЗАСТОСУНКУ ДЛЯ ЦИФРОВОЇ ДИСТРИБУЦІЇ

Графічний матеріал

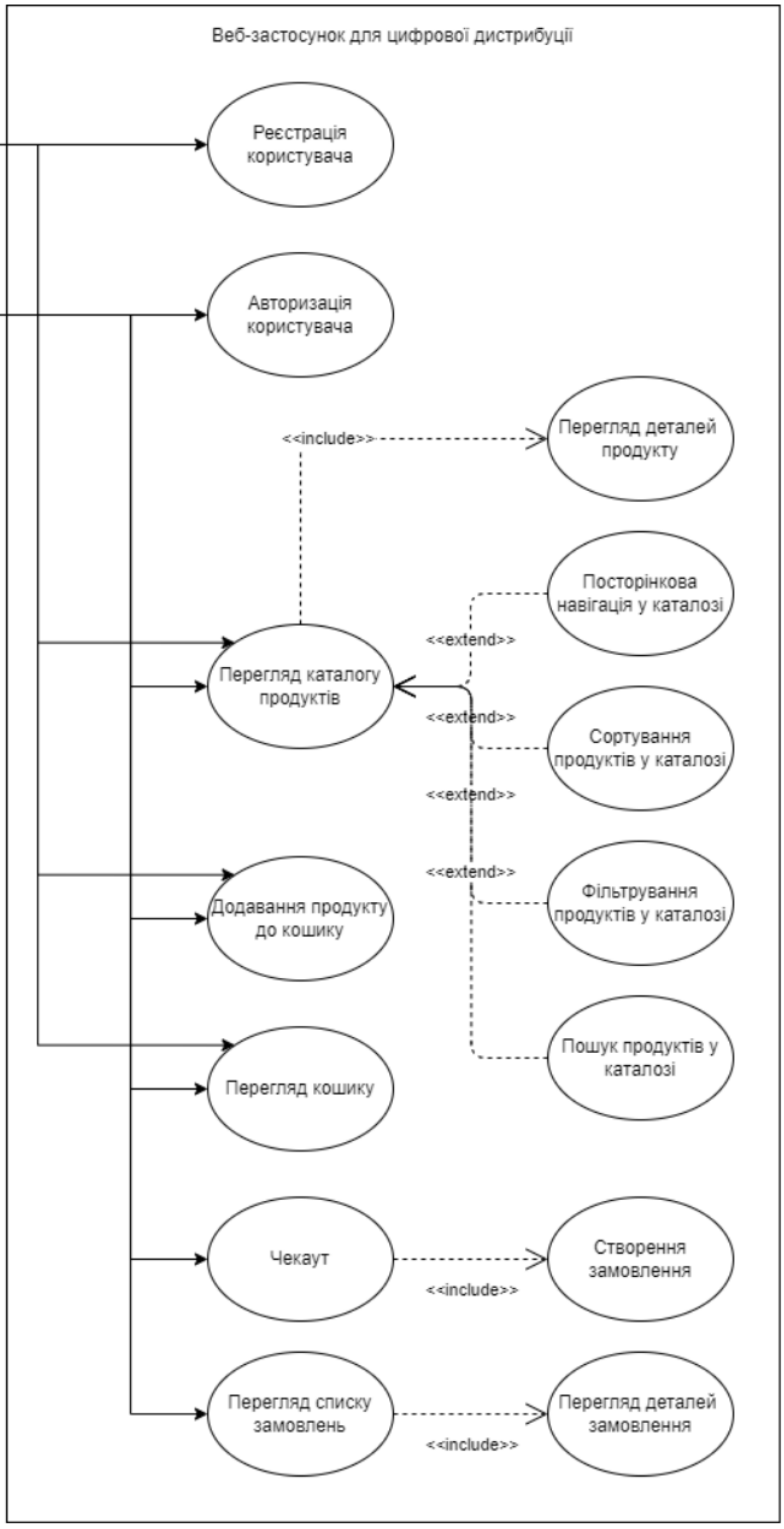
КПІ.ІП-з9101.045440.06.99

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Катерина ЛІЩУК

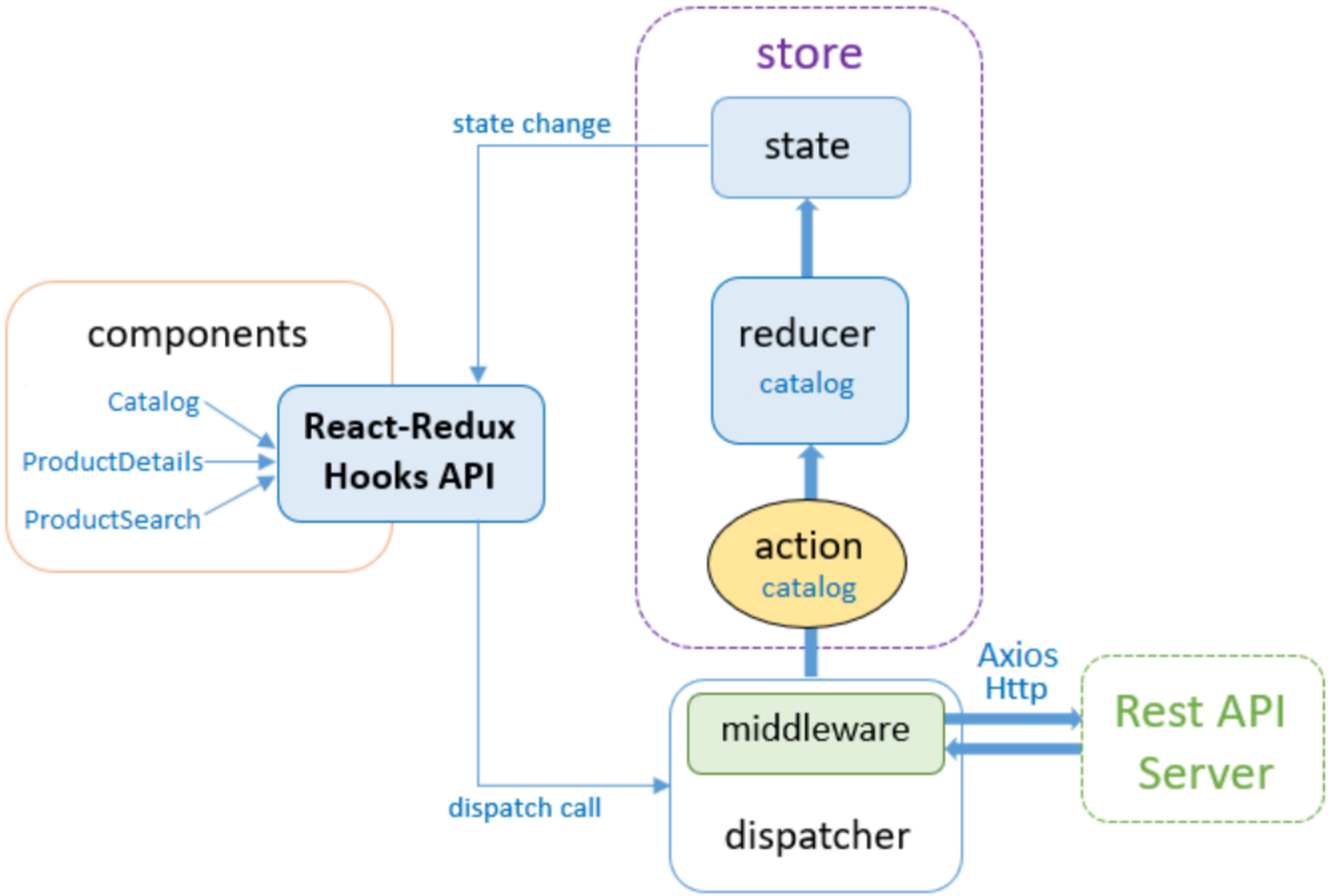
Нормоконтроль:
_____ Катерина ЛІЩУК

Виконавець:
_____ Микита БУБРЯК

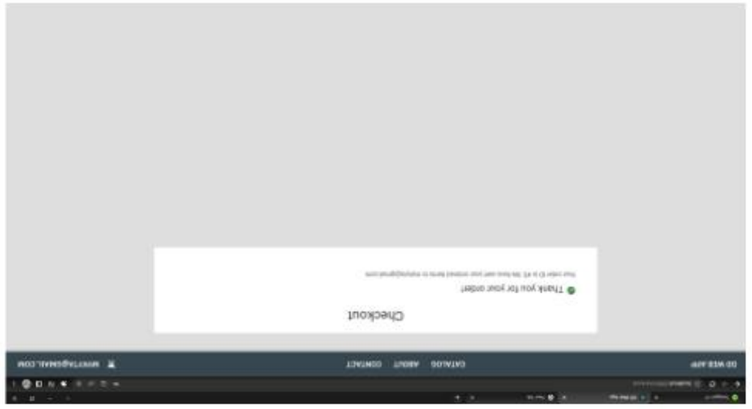
Київ – 2023

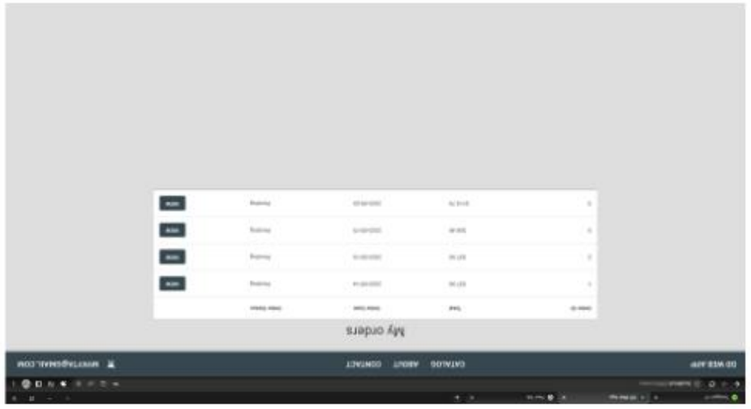


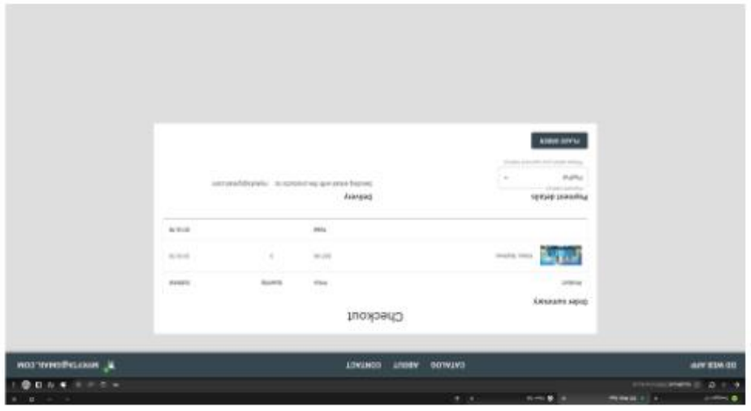
					КПІ.ІП-з9101.045440.06.99.ССВ	
					Схема структурна варіантів використань	Лит. / Арк. / Аркушів: 1
Зм.	Арк.	№ докум.	Підп.	Дата		Аркуш / Аркушів
Розроб.		Бубряк М.С.			Архітектурне рішення веб-застосунку для цифрової дистрибуції	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-з91
Перев.		Ліщук К.І.				
Т. Кон.						
Н. Кон.		Ліщук К.І.				
Затв.		Жаріков Е.В.				

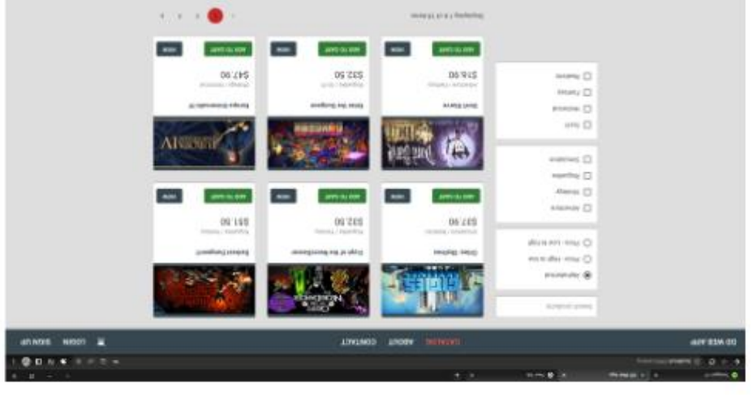


					КПІ.ІП-з9101.045440.06.99.ССМ	
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна компонентів програмного забезпечення	Літера / Маса / Масштаб
Розробив		Бубряк М.С.				
Перевірив		Ліщук К.І.				Аркуш / Аркушів
Т. кон.						
Н. кон.		Ліщук К.І.			Архітектурне рішення веб-застосунку для цифрової дистрибуції	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-з91
Затвердив		Жаріков Е.В.				









Зм.	Арк.	№ докумeнта	Підпис	Дата	КПІ.ІП-з9101.045440.06.99.КЕ						
Розробив		Бубрак М.С.									
Перевірив		Пашук К.І.			Кресленяя вигляду екранних форм				Літера	Маса	Масштаб
Т. кон.											
Н. кон.		Пашук К.І.									
Затвердив		Жарікова Е.В.							Аркуш	Аркушів	
					Архітектуре рішення веб-застосунку для цифрової дистрибуції				КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-391		