КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет інформаційних технологій
Кафедра інтелектуальних технологій

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

НА ТЕМУ

«Масштабований високонавантажений торговий інформаційний портал»

Галузь знань **12 «Інформаційні технології»**

Спеціальність **122 «Комп'ютерні науки»**

Освітня програма «**Комп'ютерні науки»**

Освітній рівень: **бакалавр**

Виконав:
студент 4 курсу групи КН-42
Садовнік А.М.

Керівник:
асистент кафедри к.т.н., с.д.
Андрійчук Олег Валентинович

Випускна кваліфікаційна робота бакалавра допущена до захисту
рішенням кафедри *інтелектуальних технологій*
Протокол № 13 від 05.06.2023 р.
Зав. кафедри _____ доц. Іларіонов О.Є.

Київ – 2023

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет інформаційних технологій

Кафедра інтелектуальних технологій

Спеціальність 122 «Комп'ютерні науки»

ЗАТВЕРДЖУЮ
Завідувач кафедри
інтелектуальних технологій
Іларіонов О.Є.

_____

"___"_____ 2023 р.

ЗАВДАННЯ
НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Садовніку Антону Михайловичу

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи): «Масштабований високонавантажений торговий інформаційний портал»

затверджена протоколом засідання кафедри від «11» листопада 2022 р. № 4

2. Термін здачі студентом закінченого проекту (роботи): 15 червня 2023 року
3. Вихідні дані до проекту (роботи): Заздалегідь додані товари до системи.
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити): Аналіз предметної області, розробка структури додатку, розробка інформаційного та програмного забезпечення системи.
5. Перелік презентаційного матеріалу (з точним зазначенням обов'язкових презентацій):
    1. Мета дипломної роботи, об'єкт дослідження та предмет дослідження.
    2. Платформи-аналоги.
    3. Функціональні та нефункціональні вимоги.
    4. Функціональний аналіз системи.
    5. Проектування архітектури торгового інформаційного порталу.
    6. Програмна реалізація торгового інформаційного порталу.
    7. Тестування.

6. Консультанти з випускної кваліфікаційної роботи із зазначенням розділів випускної кваліфікаційної роботи, що їх стосуються

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 15 лютого 2023 року

Керівник _____ / Андрійчук О. В. /
(підпис) (ПІБ)

Завдання прийняв до виконання _____ / Садовнік А. М. /
(підпис) (ПІБ)

КАЛЕНДАРНИЙ ПЛАН

Пор. №	Назва етапів випускної кваліфікаційної роботи	Термін виконання етапів випускної кваліфікаційної роботи	Примітка
1	Опрацювання літератури	15.02 – 27.02	
2	Робота над розділом 1. АНАЛІЗ ОСОБЛИВОСТЕЙ ФУНКЦІОНУВАННЯ ІСНУЮЧИХ ТОРГОВИХ ІНФОРМАЦІЙНИХ ПОРТАЛІВ	27.02 – 08.03	
3	Робота над розділом 2. РОЗРОБКА АРХІТЕКТУРИ ТОРГОВОГО ІНФОРМАЦІЙНОГО ПОРТАЛУ	08.03 – 08.04	
4	Робота над розділом 3. ПОГРАМНА РЕАЛІЗАЦІЯ ТОРГОВОГО ІНФОРМАЦІЙНОГО ПОРТАЛУ	08.04 – 13.05	
5	Робота над оформленням пояснювальної записки	13.05 – 25.05	
6	Робота над презентацією	25.05 – 29.05	

Студент-дипломник _____ / Садовнік А. М. /
(підпис) (ПІБ)

Керівник випускної кваліфікаційної роботи _____ / Андрійчук О. В. /

(підпис) (ПІБ)

Анотація

Садовнік Антон Михайлович виконав кваліфікаційну роботу на тему «Масштабований високонавантажений торговий інформаційний портал» за спеціальністю 122 – «Комп'ютерні науки».

У цій кваліфікаційній роботі розроблений web-додаток для перегляду, вибору та замовлення товарів.

Об'єктом дослідження даної роботи є процес розробки сучасного високонавантаженого торгового інформаційного порталу.

Предметом дослідження роботи є технології розробки та тестування програмних продуктів, які застосовуються для високонавантажених торгових інформаційних порталів.

Метою дипломної роботи є розробка сучасного високонавантаженого торгового інформаційного порталу та проведення його тестування на відповідність сучасним вимогам.

У першому розділі проведено аналіз предметної області торгових порталів, проаналізовано існуючі рішення на ринку та розкрито питання впливу швидкодії на досвід користування порталом. У другому розділі спроектовано функціональну структуру системи, структуру бази даних та загальну архітектуру системи. У третьому розділі були розглянуті основні модульні категорії системи, побудований користувацький інтерфейс, проведене тестування для виявлення недоліків та подальшому удосконаленню системи.

Ключові слова: портал, високонавантажений торговий інформаційний портал, товар, тест.

Annotation

Anton Sadovnik performed a qualification work on the topic "Scalable high-load trade information portal" in the speciality 122 - "Computer Science".

In this qualification work, a web application for viewing, selecting and ordering goods has been developed.

The object of study of this work is the process of developing a modern high-load trading information portal.

The subject of the research is the technologies of development and testing of software products used for high-load trading information portals.

The purpose of the thesis is to develop a modern high-load trading information portal and test it for compliance with modern requirements.

The first chapter analyses the subject area of trading portals, analyses existing solutions on the market and reveals the impact of performance on the portal user experience. The second section describes the system's functional structure, database structure and overall architecture. In the third section, the main modular categories of the system were considered, the user interface was built, and testing was carried out to identify shortcomings and further improve the system.

Keywords: portal, high-load trade information portal, product, test.

ЗМІСТ

ВСТУП

З появи інтернету в 1969 році вже пройшло багато часу, але технологічний прогрес ні на мить не зупинявся. Сфера інформаційних технологій не стала винятком і навіть зараз продовжує розвиватись з неймовірною швидкістю. З’являються нові технології, формується все більше підходів до розробки ПЗ та їх тестуванню. Сфера веб-технологій не стала винятком і на даний момент займає одну із лідируючих позицій по кількості залучених робітників та представників бізнесу. Зараз, фактично, якщо бізнес ніяк неінтегрований з інтернет-системами, то його в принципі не існує фізично – цю тенденцію підтверджують такі корпорації як Google, Amazon, Microsoft, Apple та інші. Сфера торгівлі не стала винятком – з ростом кількості користувачів вона потребує все більше цифровізації, яка буде одночасно безпечною та не мати проблем зі швидкодією в досить складних умовах великої конкуренції проектних рішень.

Метою дипломної роботи є розробка масштабованого високонавантаженого торгового інформаційного порталу та проведення його тестування на відповідність сучасним вимогам.

Об’єктом дослідження даної роботи є процес розробки масштабованого високонавантаженого торгового інформаційного порталу.

Предметом дослідження роботи є технології розробки та тестування програмних продуктів, які застосовуються для масштабованих високонавантажених торгових інформаційних порталів.

РОЗДІЛ 1. АНАЛІЗ ОСОБЛИВОСТЕЙ ФУНКЦІОНУВАННЯ ІСНУЮЧИХ ТОРГОВИХ ІНФОРМАЦІЙНИХ ПОРТАЛІВ

1.1. Аналіз наявних торгових інформаційних порталів

На даний момент існують такі альтернативні торгові платформи: “Ninja Sushi”, “Binance”, “Amazon” та інші. Всі ці платформи мають на меті проведення інформаційно-торгових операцій в умовах високої завантаженості, оскільки одночасно користуватись додатком можуть безліч людей з усього світу.

Після аналізу торгових платформ, можна виділити їх 2 головні функції: інформаційна та торгова.

Для інформаційної функції важливо мати:

- Приємний і інтуїтивно зрозумілий інтерфейс.
- Гнучку систему пошуку товарів.
- Коректну інформацію про товар (назва, опис, характеристики тощо).
- Правильно вказані контактні дані для зв’язку при виникненні технічних проблем.

Для торгової функції важливо мати:

- Коректну ціну для кожного товару.
- Реалізований функціонал кошику.
- При необхідності, поля для вказання адреси доставки.
- Реалізований функціонал оплати.

Метою аналізу альтернативних продуктових реалізацій є перевірка актуальності розробки високонавантаженої торгової платформи.

В результаті проведеного аналізу існуючих торгових платформ можна виділити наступні елементи (табл. 1.1):

Таблиця 1.1 – Порівняння існуючих музичних платформ

	“Ninja Sushi”	“Binance”	“Amazon”
Швидкість завантаження головної сторінки	0.219c	0.451c	0.392c
Наявність гнучкого пошуку	-	-	+
Збереження інформації при перезавантаженні сторінки	+	-	+
Наявність можливості онлайн оплати	+	+	+
Персоналізація товарів відповідно вподобанням користувача	-	+	+
Зручний інтерфейс	+	-	-
Кроссбраузерний застосунок	+	+	+
Наявність мобільної версії	+	+	+

1.2. Вплив швидкодії на досвід користування торговими порталами

Ефективність завантаження сайту є вирішальним аспектом взаємодії з додатком. У сучасному швидкому цифровому світі користувачі очікують, що веб-сайти завантажуватимуться швидко та плавно. Повільне завантаження може розчарувати користувачів, змушуючи їх покинути платформу і шукати альтернативи. Тому важливо перевірити та оптимізувати продуктивність завантаження сайту, щоб забезпечити позитивний досвід користувачів.

Існують різні інструменти для тестування продуктивності завантаження сторінок. Одним із популярних інструментів є Google

PageSpeed Insights, який аналізує ефективність веб-сайту як на мобільних, так і на настільних пристроях [1]. Інструмент надає оцінку зі 100, а також дієві пропозиції щодо покращення швидкодії. Інші інструменти включають GTmetrix, Pingdom і WebPageTest, кожен зі своїм унікальним набором функцій.

Ефективність завантаження сайту визначається різними факторами, включаючи час відповіді сервера, розмір сторінки, оптимізацію зображень і оптимізацію коду. Оптимізуючи ці фактори, власники веб-додатків можуть підвищити швидкість застосунку, що призведе до покращення взаємодії з користувачем. Наприклад, стиснення зображень, зменшення кількості HTTP-запитів і мінімізація коду можуть сприяти швидшому завантаженню платформи [2].

Покращення продуктивності завантаження застосунку може мати значний вплив на взаємодію з користувачем. Веб-сайт, який швидко завантажується, не тільки забезпечує кращий досвід роботи з користувачем, але також сприяє більшій залученості та збільшенню конверсій. З іншого боку, сайт із повільним завантаженням може негативно вплинути на залучення користувачів, що призведе до вищого показника відмов, нижчої кількості конверсій і зменшення доходу [3].

Підсумовуючи, продуктивність завантаження торгової платформи є критично важливим аспектом взаємодії з користувачем. Тестуючи та оптимізуючи швидкодію, власники веб-сайтів можуть покращити залученість користувачів, збільшити конверсії та, зрештою, розвивати свій бізнес. Тому дуже важливо визначити пріоритетність завантаження сайту як частину будь-якої стратегії розробки або обслуговування веб-сайту.

Щоб оптимізувати роботу додатку є безліч підходів та методів, ось одні з найбільш популярних і дієвих:

- Використання мережі доставки вмісту (англ. CDN): CDN — це мережа серверів, розподілених по всьому світу, яка зберігає та доставляє вміст сайту користувачам із найближчого до них сервера. Використання CDN може зменшити час відповіді сервера та покращити швидкість завантаження сайту [6].
- Оптимізація зображень: великі зображення можуть значно сповільнити швидкість завантаження сайту. Оптимізація зображень шляхом їх стиснення, зменшення розміру та використання правильного формату файлу може допомогти підвищити ефективність сайту.
- Зменшення коду: скорочення коду передбачає видалення непотрібних символів і пробілів із файлів HTML, CSS і JavaScript сайту. Це може зменшити розмір файлу та покращити швидкість завантаження сайту.
- Зменшення кількості HTTP-запитів: кожен HTTP-запит, зроблений браузером для завантаження сторінки, потребує зворотного зв’язку між браузером і сервером. Зменшення кількості HTTP-запитів шляхом об’єднання файлів, використання спрайтів CSS і вбудованих зображень може покращити швидкість завантаження сайту [7].
- Використовувати кешування веб-переглядача: кешування веб-переглядача дозволяє браузеру зберігати деякі елементи сайту на комп’ютері користувача, зменшуючи потребу браузера повторно запитувати ті самі ресурси з сервера [7].
- Увімкнути стиснення Gzip: стиснення Gzip — це метод стиснення файлів, які надсилаються із сервера до браузера,

зменшуючи розмір файлу та покращуючи швидкість завантаження сайту [8].

- Використовуйте швидшого хостинг-провайдера: швидкість сервера може впливати на швидкість завантаження сайту. Вибір швидкого та надійного хостинг-провайдера може покращити продуктивність сайту.
- Видаліть непотрібні плагіни: використання занадто великої кількості плагінів може сповільнити швидкість завантаження сайту. Видалення непотрібних плагінів може покращити продуктивність сайту.

Підсумовуючи, оптимізація продуктивності сайту вимагає поєднання різних стратегій, включаючи використання CDN, оптимізацію зображень, скорочення коду, зменшення запитів HTTP, увімкнення кешування браузера, використання стиснення Gzip, вибір швидшого хостинг-провайдера та видалення непотрібних плагінів. Впроваджуючи ці способи, власники веб-сайтів можуть покращити швидкість завантаження сайту, забезпечуючи кращий досвід користувача та збільшуючи залучення та конверсії.

1.3. Постановка задачі на розробку високонавантаженого торгового порталу

Задачею роботи є отримання повністю працюючого клієнт-серверного додатку з функціоналом торгової платформи. Можна виділити наступні кроки розробки:

1. Розробити дизайн і функціональну наповненість торгової платформи.
2. Спроектувати та наповнити базу даних, яка буде зберігати в собі дані про клієнтів, товари тощо.

3. Спроектувати та реалізувати фронтенд додатку.
4. Спроектувати та реалізувати бекенд додатку.
5. Провести тестування високонавантаженої платформи з метою рефакторинга кода та оптимізації швидкодії додатка.

1.4. Аналіз основних процесів предметного середовища при розробці торгового порталу

1.4.1. Аналіз предметної області

Створення додатку таких масштабів – тяжкий та відповідальний процес який потребує детального аналізу функціонування та постановки задачі яку необхідно вирішувати.

Після того як користувач перейшов за посиланням на головну сторінку додатка є 2 основних сценарію його взаємодії з платформою:

1. Якщо користувач не пройшов реєстрацію/авторизацію, то йому буде закритий доступ до персонального кабінету з історією його покупок, під час оплати доведеться вказувати всі дані самостійно, система рекомендацій не буде пропонувати більш релевантні товари тощо.
2. Якщо користувач пройшов реєстрацію/авторизацію, то йому буде доступний персональний кабінет з інформацією про нього, історією покупок, платіжною інформацією, адресою доставки, система рекомендацій буде пропонувати більш релевантні товари тощо.

Основними цілями системи є:

- Донесення коректної інформації до користувача про асортимент тих чи інших товарів.
- Проведення реєстрації/авторизації.

- Проведення доставки та оплати товарів.

Складовими елементами є:

- База даних користувачів.
- База даних товарів.
- Підсистема авторизації.
- Підсистема доставки та оплати товарів.

1.5. Функціональні та нефункціональні вимоги до розробки торгового порталу

1.5.1. Функціональні вимоги

Функціональні вимоги — це вимоги до програмного забезпечення, які описують внутрішню роботу системи, її поведінку: обчислення даних, маніпулювання даними, обробка даних та інші специфічні функції, які має виконувати система.

До функціональних вимог можна віднести:

1. Наявність механізму авторизації.
2. Можливість переглядати та взаємодіяти з товарами.
3. Всі зображення з товарами повинні бути клікабельними і переносити користувача на сторінку відповідного товару.
4. Товари повинні сортуватись та фільтруватись за обраними користувачем критеріями.
5. Кошик має приймати в себе товари, показувати кількість товарів, мати можливість змінити їх кількість, підраховувати загальну ціну.
6. Повинна бути можливість використовувати навігацію по додатку.

7. Додаток мусить приводити коректну інформацію для кожного з товарів.
8. Мусить бути можливість використати пошук товарів.
9. При повільному з’єднанні потрібно відображати інформацію про хід завантаження товарів.
10. Наявність для авторизованих користувачів можливості залишати відгуки.

1.5.2. Нефункціональні вимоги

Нефункціональні вимоги — це вимоги до програмного забезпечення, які задають критерії для оцінки якості його роботи. На відміну від функціональних вимог, які визначають що система повинна робити, нефункціональні вимоги визначають якою система повинна бути.

До нефункціональних вимог можна віднести:

1. Швидкодія.
2. Цілісність.
3. Масштабованість.
4. Стійкість до збоїв.
5. Дотримуватись принципів «чистого коду» [4] для подальших модифікацій.
6. Система повинна мати інтуїтивно зрозумілий інтерфейс.
7. Система повинна забезпечувати одночасне підключення порядку 100 осіб.

1.6. Висновки до першого розділу

Отже, в результаті аналізу предметної області було оцінено сучасний стан та актуальність процесу розробки високонавантаженої торгової платформи.

Також було проаналізовано основні процеси предметного середовища. Заключним етапом було сформовано задачу на розробку, а саме – визначення всіх функціональних і нефункціональних вимог та їх опис.

РОЗДІЛ 2. РОЗРОБКА АРХІТЕКТУРИ ТОРГОВОГО ІНФОРМАЦІЙНОГО ПОРТАЛУ

2.1. Функціональний аналіз системи

Після того як користувач переходить за посиланням він опиняється на головній сторінці застосунку, звідси він може потрапити як до необхідної йому категорії, так і переглянути акційні пропозиції, взаємодіяти зі слайдерами, переглядати інформацію про магазин тощо. В будь-якому з цих варіантів користувач опиняється або на сторінці товару або одразу взаємодіє з кошиком товарів після чого відправляється до форми введення контактної інформації. Окрім користувача, для системи необхідно передбачити поведінку адміністратора, який повинен як і користувач мати увесь функціонал порталу, але окрім цього ще додаткові функціональні можливості для взаємодії з системою.

Отже, в даній ситуації дійовою особою виступає користувач та адміністратор, які так чи інакше взаємодіють з системою.

Для кращого розуміння функціональних вимог до системи була побудована діаграма варіантів використання програми (рис. 2.1).

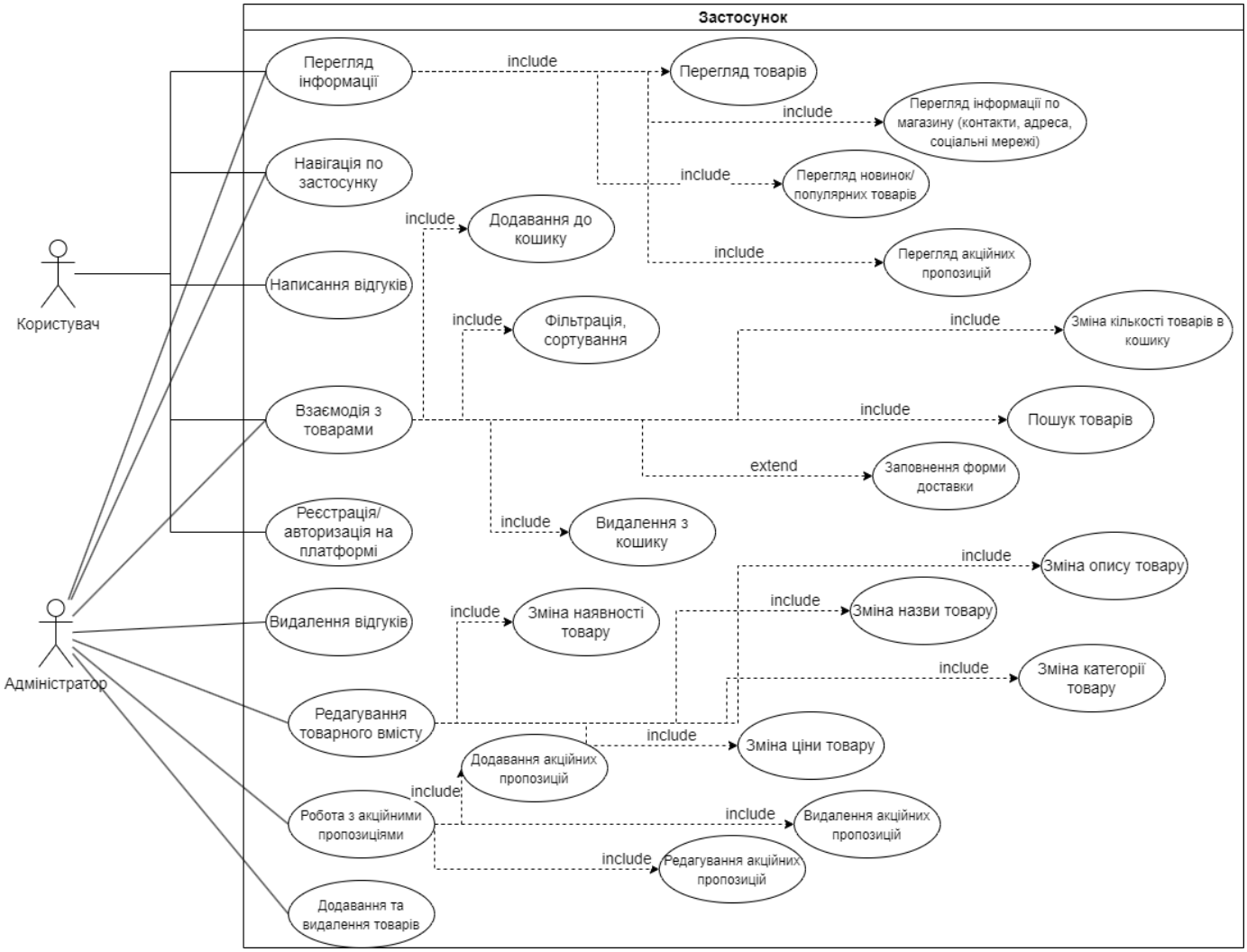

Рисунок 2.1 – Діаграма варіантів використання системи

Варіанти використання:

- Перегляд інформації.
- Навігація по застосунку.
- Написання відгуків.
- Взаємодія з товарами.
- Реєстрація/авторизація на платформі.
- Видалення відгуків.
- Редагування товарного вмісту.

- Робота з акційними пропозиціями.
- Додавання та видалення товарів.
- Зміна наявності товару.
- Зміна назви товару.
- Зміна опису товару.
- Зміна категорії товару.
- Зміна ціни товару.
- Додавання акційних пропозицій.
- Видалення акційних пропозицій.
- Редагування акційних пропозицій.
- Перегляд товарів.
- Перегляд інформації про магазин (контакти, адреса, соціальні мережі).
- Перегляд новинок/популярних товарів.
- Перегляд акційних пропозицій.
- Додавання товарів до кошику.
- Фільтрація/сортування товарів.
- Зміна кількості товарів у кошику.
- Пошук товарів.
- Видалення з кошику товарів.
- Заповнення форми доставки.

Сценарії:

Основний сценарій для користувача:

1. Користувач заходить на головну сторінку.
2. Користувач реєструється на платформі.
3. Користувач авторизується на платформі.
4. Користувач переглядає акційні пропозиції, популярні товари та новинки.
5. Користувач переходить до категорії товарів, що його цікавлять.
6. Користувач обирає товари.
7. Користувач переходить до кошику та якщо необхідно змінює кількість обраних товарів.
8. Користувач в кошику переходить на сторінку оформлення замовлення та вказує контактну інформацію.
9. Користувач обирає спосіб оплати та натискає на кнопку оформлення замовлення.

Альтернативні сценарії для користувача:

1. Якщо користувач не авторизувався то вся інформація по замовленню зберігається на фронтенді та він не має доступу до написання відгуку.
2. Якщо користувач випадково перезавантажив сторінку, то всі його товари, виставлені фільтри та сортування зберігаються та йому не доведеться знову їх налаштовувати.
3. Якщо користувач неавторизований додав товари до кошику, а потім авторизувався то його товари об’єднаються, при наявності, з товарами з бази даних.

Основний сценарій для адміністратора:

1. Адміністратор авторизується на сайті.
2. Адміністратор переходить на панель адміністратора.

3. Адміністратор може видаляти відгуки, редагувати вміст наявних товарів, додавати нові товари та видаляти наявні, працювати з акційними пропозиціями.

2.2. Проектування архітектури високонавантаженого торгового інформаційного порталу

Для отримання повної картини архітектури додатка, необхідно розділити його на 2 частини: фронтенд та бекенд (рис. 2.2).

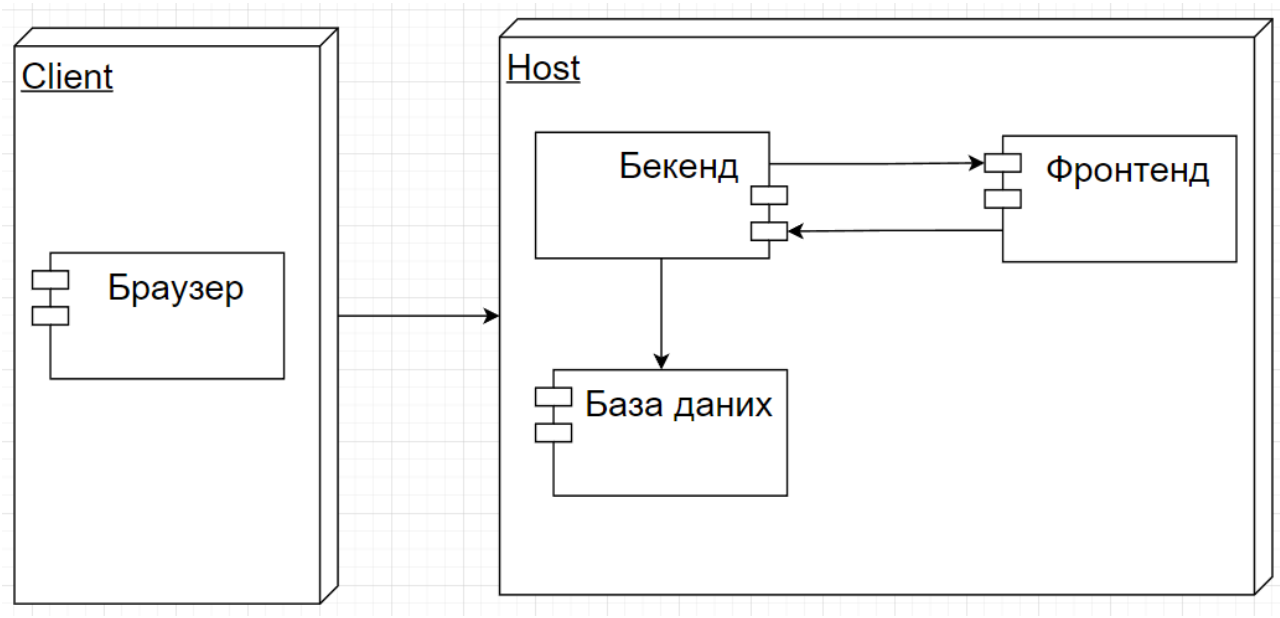

Рисунок 2.2 – Діаграма представлення загальної структури застосунку

Загальна структура застосунку являє собою частину клієнта та хоста, частина хоста – це Docker-контейнер, в якому знаходиться бекенд та фронтенд торгового порталу. Зроблено це з метою ізолювати код та пришвидшити його роботу за допомогою вбудованих оптимізаційних рішень всередині Docker.

2.2.1. Фронтенд високонавантаженого торгового порталу:

Фронтенд високонавантаженого торгового порталу складається з модулю генерації сторінок, який в свою чергу виконує 3 основні функції (рис. 2.3):

- Робота з компонентами інтерфейсу: їх відображення, модифікація, видалення.
- Відправка запитів на сервер.
- Маршрутизація, тобто можливість переходити з однієї сторінки на іншу

2.2.2. Бекенд високонавантаженого торгового порталу:

Бекенд торгової платформи складається з трьох модулів (рис. 2.3):

- Модуль бази даних – виконує налаштування бази даних та контролює зміни в ній.
- Модуль роботи з товарами – виконує функціонал сортування, фільтрації товарів, отримання за запитом товарів з бази даних.
- Модуль оформлення замовлення – виконує функціонал валідації введених даних, проведення оплати.

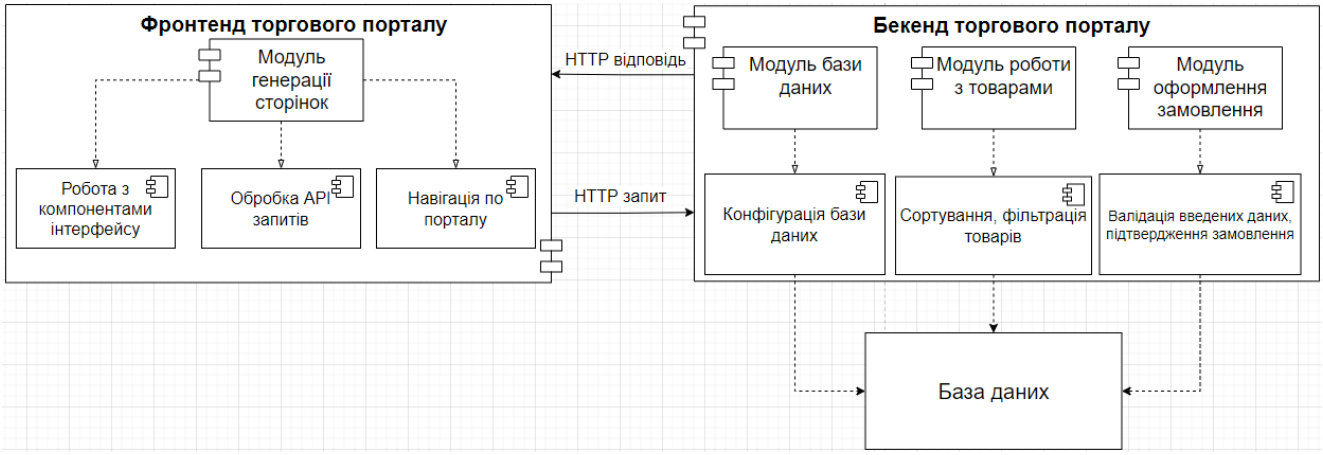

Рисунок 2.3 – Діаграма компонентів торгового порталу

Тепер необхідно описати один із основних сценаріїв функціонування торгового порталу (рис. 2.4). Його початком є перехід користувача за посиланням на головну сторінку. Торговий портал завантажує акційні пропозиції, категорії і товари. Далі користувач авторизується або реєструється в системі. З бази даних підягується інформація про кошик користувача. Коли клієнт закінчує вибір товарів, він переходить до кошику, а звідти на сторінку доставки де заповнює необхідні поля і обирає тип оплати.

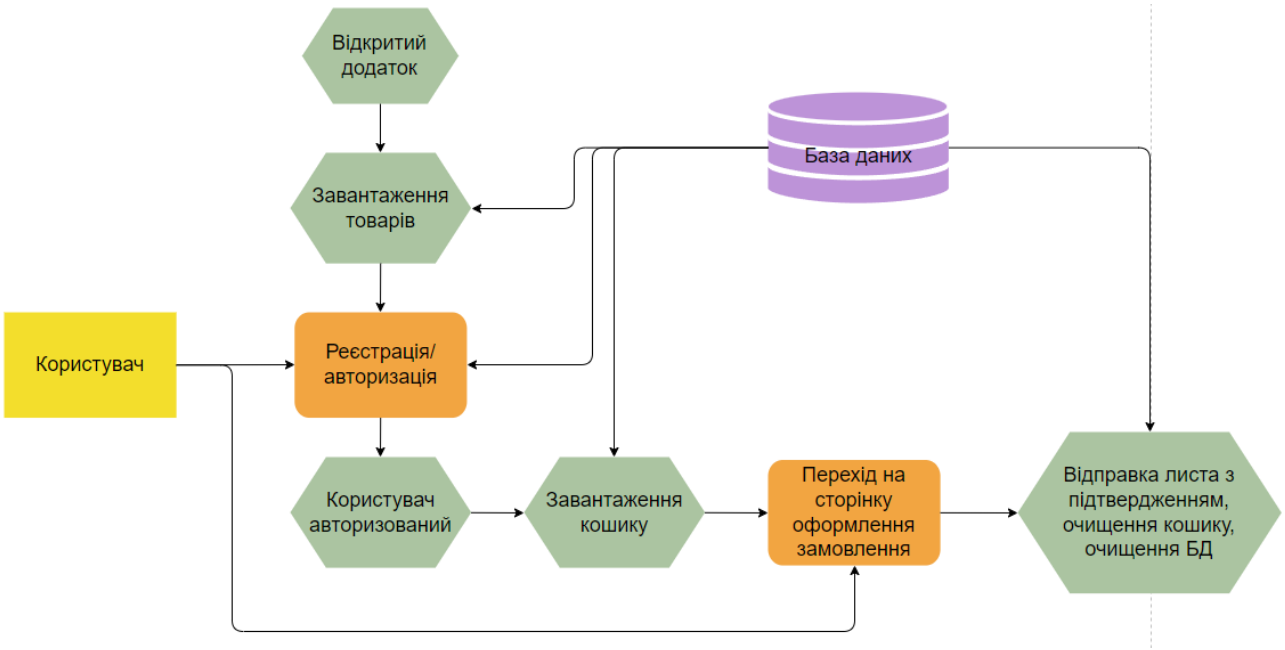

Рисунок 2.4 – EPC діаграма функціонування додатку

2.2.3. Опис процесу оформлення замовлення

Створення додатку таких масштабів – тяжкий та відповідальний процес який потребує детального аналізу функціонування та постановки задачі яку необхідно вирішувати.

Перший етап - це створення контекстної діаграми “ЯК Є” (рис. 2.5), який відображає процес оформлення замовлення користувачем. Після того, як користувач обрав товари, що його зацікавили, йому необхідно вказати адресу доставки. На вході система отримує дані по замовленню та перевіряє їх валідність, а на виході відправляє до користувача інформацію про успішну або неуспішну обробку замовлення. Серед елементів керування для даного процесу зазначено Закон України «Про захист персональних даних», який необхідний при обробці персональних даних користувача, таких як: ім’я, прізвище, телефонний номер, електронна адреса тощо. Механізмами у даному випадку є користувач системи який вводить інформацію та дає свою згоду на обробку персональних даних, і алгоритм, який перевірить коректність введеної інформації.

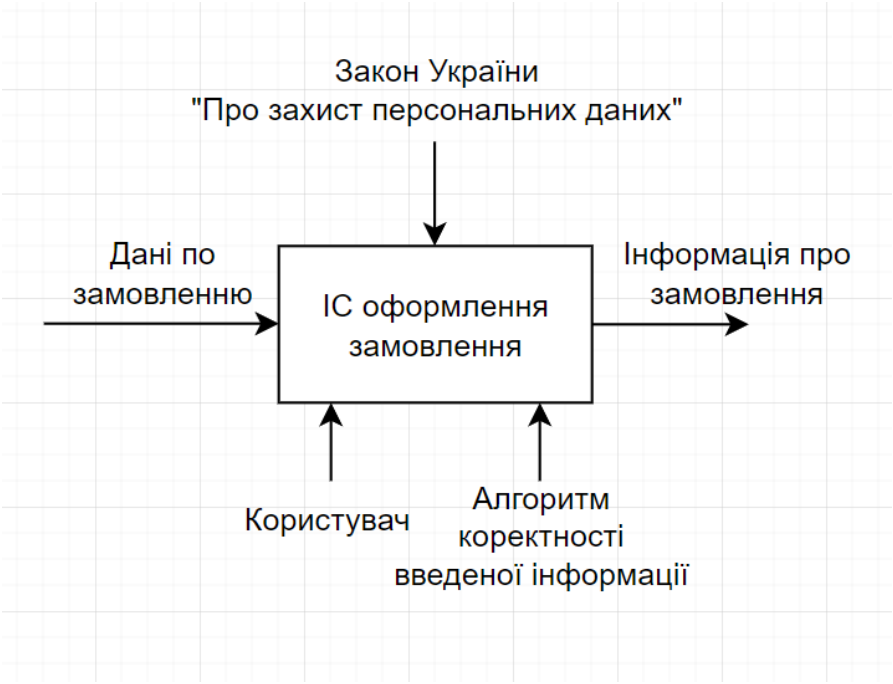

Рисунок 2.5 – Контекстна діаграма «ЯК Є»

Більш детально розглянути процес, який описаний раніше можливо завдяки декомпозиції основних функцій діаграми «ЯК Є» (рис. 2.6). Після отримання даних від користувача, система аналізує їх та перевіряє чи всі поля заповнені коректно. Після перевірки, система збирає всі дані та відправляє їх до конкретного магазину, де вже відбувається реалізація конкретного замовлення.

Рисунок 2.6 – Декомпозиція основних функцій діаграми «ЯК Є».

2.3. Архітектура бази даних торгового порталу

В результаті проведеного аналізу предметної області та поставленої задачі було прийнято рішення, що дана платформа не зможе працювати без великого сховища інформації, тому було прийнято рішення спроектувати базу даних торгового порталу.

Для того, щоб вдало спроектувати базу даних, необхідно розробити її концептуальну (рис. 2.7), логічну (рис. 2.8) та фізичну модель (рис. 2.9), які будуть чітко показувати як дані зберігаються і більш детальну інформацію про атрибути сховища інформації.

Концептуальна модель даних – це організоване уявлення про концепції бази даних та їхні зв'язки. Метою створення концептуальної моделі даних є визначення сутностей, їхніх атрибутів і зв'язків. На цьому рівні моделювання даних навряд чи можна отримати детальну інформацію про фактичну структуру бази даних.

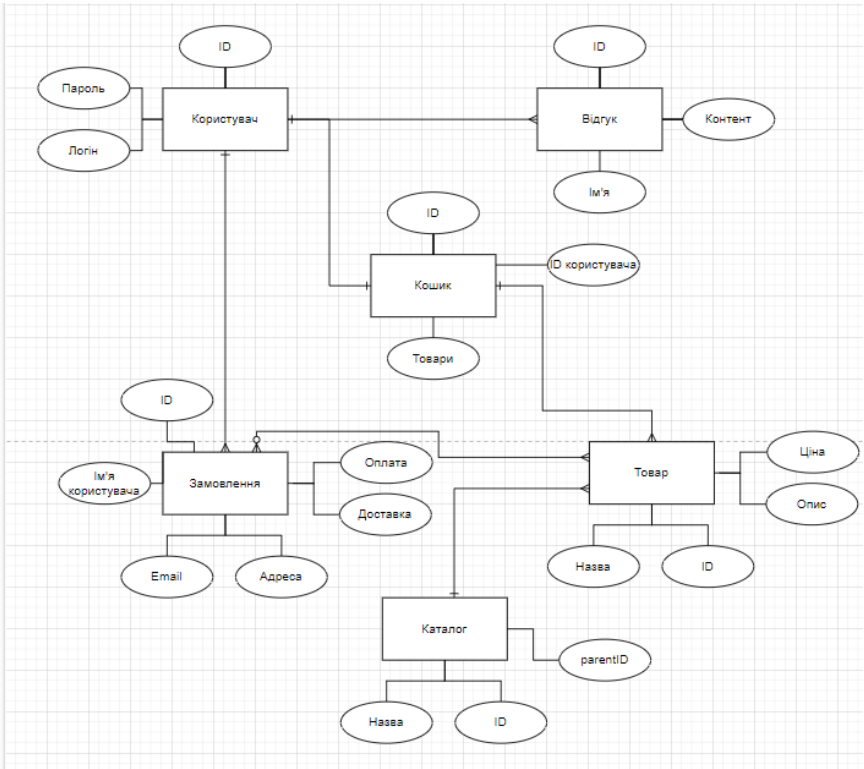

Рисунок 2.7 – Концептуальна модель бази даних

Отже, як видно з Рисунку 2.7, база даних має 6 сутностей, а саме:

- Користувач.
- Відгук.
- Кошик.
- Замовлення.
- Товар.
- Каталог

Логічна модель даних – це тип моделі даних, яка детально описує елементи даних і використовується для розробки візуального розуміння сутностей, атрибутів, ключів і зв'язків даних.

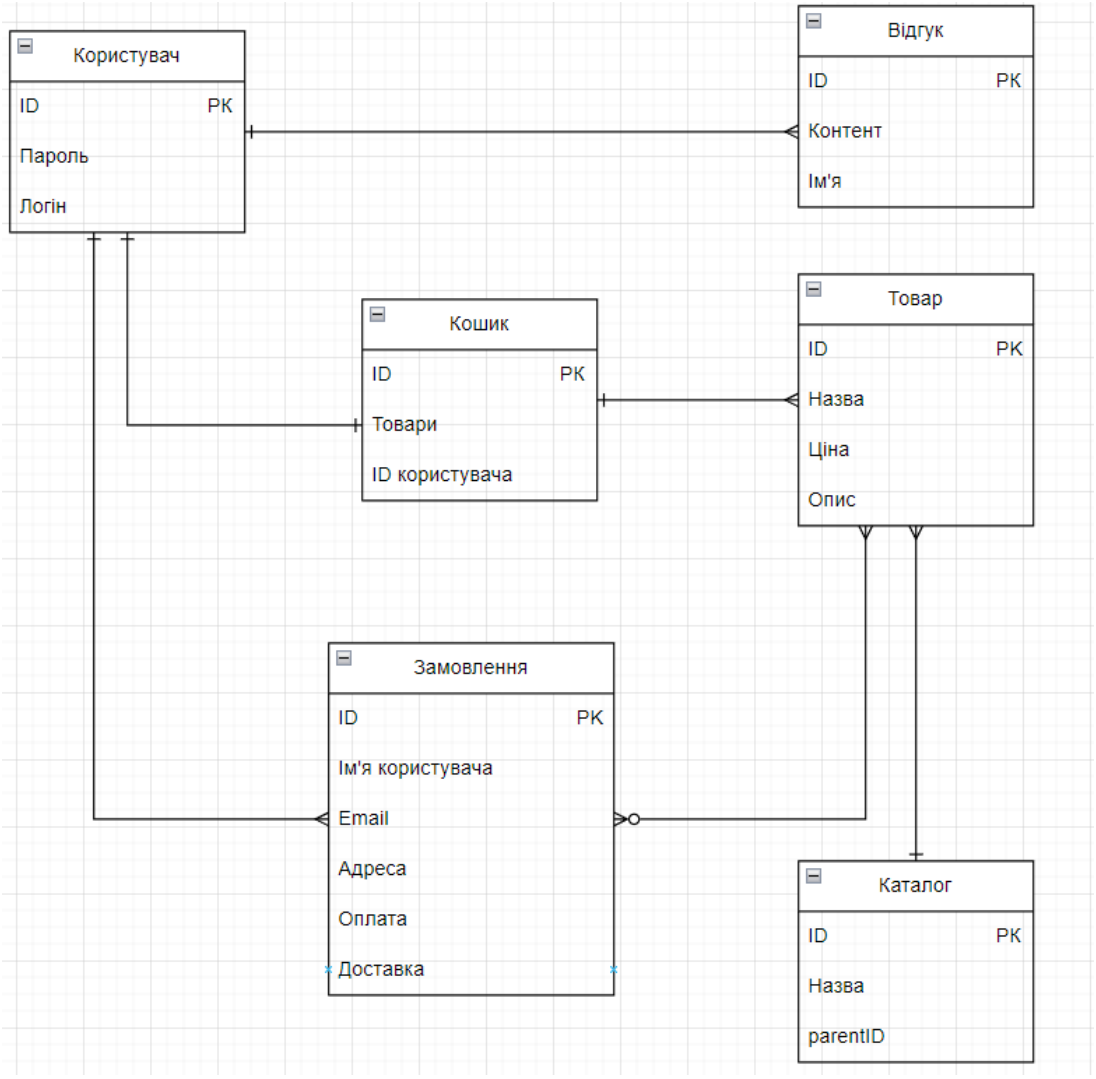

Рисунок 2.8 – Логічна модель бази даних

Отже, як видно з Рисунку 2.8 логічна модель бази даних дає змогу більш детально ознайомитись з сутностями, їх атрибутами та зв’язками між ними.

Фізична модель даних – це специфічна для бази даних модель, яка представляє реляційні об'єкти даних (наприклад, таблиці, стовпці, первинні та зовнішні ключі) та їхні зв'язки.

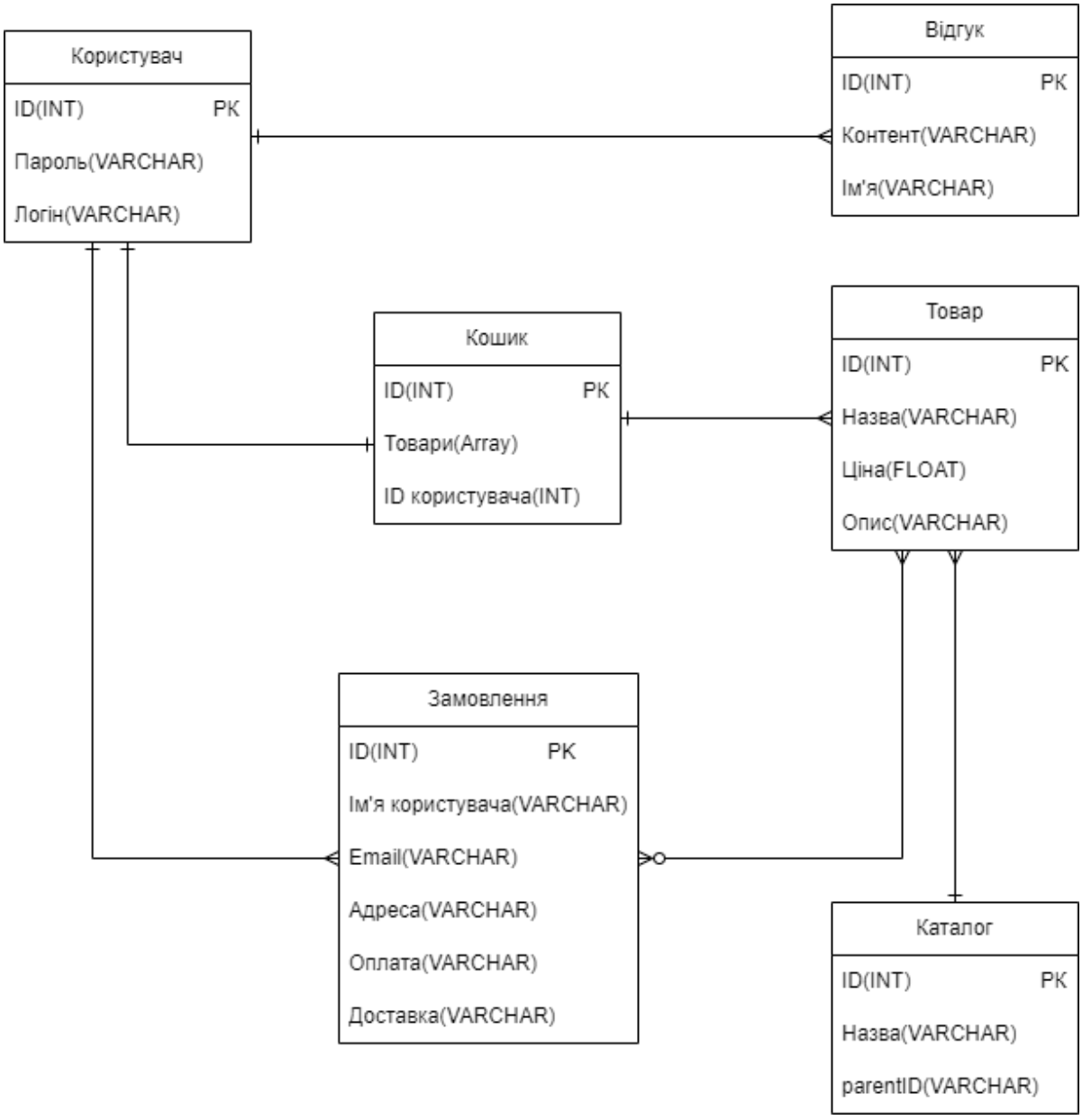

Рисунок 2.9 – Фізична модель бази даних

Отже, після проектування фізичної моделі бази даних, необхідно описати атрибути кожної з сутностей (табл. 2.1 – табл. 2.6).

Таблиця 2.1 – Склад та характеристика атрибутів сутності «Користувач»

№ п/п	Назва елемента даних	Тип елемента даних	Обов’язкове значення	Обмеження	Ключ
1	ID	Ціле число	так		ПК
2	Пароль	Символьний	так		
3	Логін	Символьний	так		

Таблиця 2.2 – Склад та характеристика атрибутів сутності «Товар»

№ п/п	Назва елемента даних	Тип елемента даних	Обов’язкове значення	Обмеження	Ключ
1	ID	Ціле число	так		ПК
2	Назва	Символьний	так		
3	Ціна	Числовий	так		
4	Опис	Символьний	так		

Таблиця 2.3 – Склад та характеристика атрибутів сутності «Відгук»

№ п/п	Назва елемента даних	Тип елемента даних	Обов’язкове значення	Обмеження	Ключ
1	ID	Ціле число	так		ПК
2	Контент	Символьний	так		
3	Ім’я	Символьний	так		

Таблиця 2.4 – Склад та характеристика атрибутів сутності «Кошик»

№ п/п	Назва елемента даних	Тип елемента даних	Обов’язкове значення	Обмеження	Ключ
1	ID	Ціле число	так		ПК
2	Товари	Масив	так		
3	ID користувача	Ціле число	так		

Таблиця 2.5 – Склад та характеристика атрибутів сутності «Замовлення»

№ п/п	Назва елемента даних	Тип елемента даних	Обов’язкове значення	Обмеження	Ключ
1	ID	Ціле число	так		ПК
2	Email	Символьний	так		
3	Ім’я користувача	Символьний	так		
4	Адреса	Символьний	так		
5	Оплата	Символьний	так		
6	Доставка	Символьний	так		

Таблиця 2.6 – Склад та характеристика атрибутів сутності «Каталог»

№ п/п	Назва елемента даних	Тип елемента даних	Обов’язкове значення	Обмеження	Ключ
1	ID	Ціле число	так		ПК
2	Назва	Символьний	так		
3	parentID	Символьний	так		

2.4. Висновки до другого розділу

Отже, в результаті процесу проектування високонавантаженої торгової платформи було сформовано загальне уявлення проекту.

Також було сформовано діаграми представлення структури платформи, побудована EPC діаграма функціонування додатку, розроблена діаграма «ЯК Є» для системи оформлення замовлення. Заключним етапом було сформовано архітектуру бази даних торгової платформи у вигляді фізичної моделі.

РОЗДІЛ 3. ПОГРАМНА РЕАЛІЗАЦІЯ ТОРГОВОГО ІНФОРМАЦІЙНОГО ПОРТАЛУ

3.1. Розробка користувацького інтерфейсу торгового інформаційного порталу

Продуманий інтерфейс є не менш важливим елементом всього високонавантаженого торгового порталу, ніж його логіка чи інформація, яку користувачу необхідно показати [8].

Для того, щоб реалізувати красивий та інтуїтивно зрозумілий інтерфейс, необхідно використовувати спеціальні графічні дизайнерські застосунки. Один з них називається «Figma» і саме в ній була виконана розробка дизайну торгової інформаційної платформи.

Перше що бачить кожен хто заходить до порталу є головна сторінка (рис. 3.1). На ній необхідно відобразити найважливішу і найцікавішу інформацію, яка зможе зацікавити клієнта – це можуть бути акційні пропозиції, найбільш популярні категорії, сезонні меню тощо. Було прийнято рішення розділити сторінку на 4 інформаційні блоки:

1. Блок зі слайдером має на своїй меті відображати найбільш вигідні акційні пропозиції з можливістю додавання їх у кошик, ключовими елементами, які відповідають за інформаційну наповненість є: зображення товару, його назва, вага, стандартна ціна, акційна ціна та кнопка додавання до кошику.
2. Блок з категоріями є одним з елементів навігації по найбільш популярним категоріям додатку, а також якщо користувач переглядає платформу з телефону, то виконує роль повноцінного меню навігації.
3. Блок із слайдером, який відображає нові та найбільш популярні товари, вони змінюються в залежності від обраної категорії.

4. Інформативний блок з описом магазину, який можна розгорнути і дізнатись більше про компанію, а також текст опису змінюється відповідно до обраного міста.

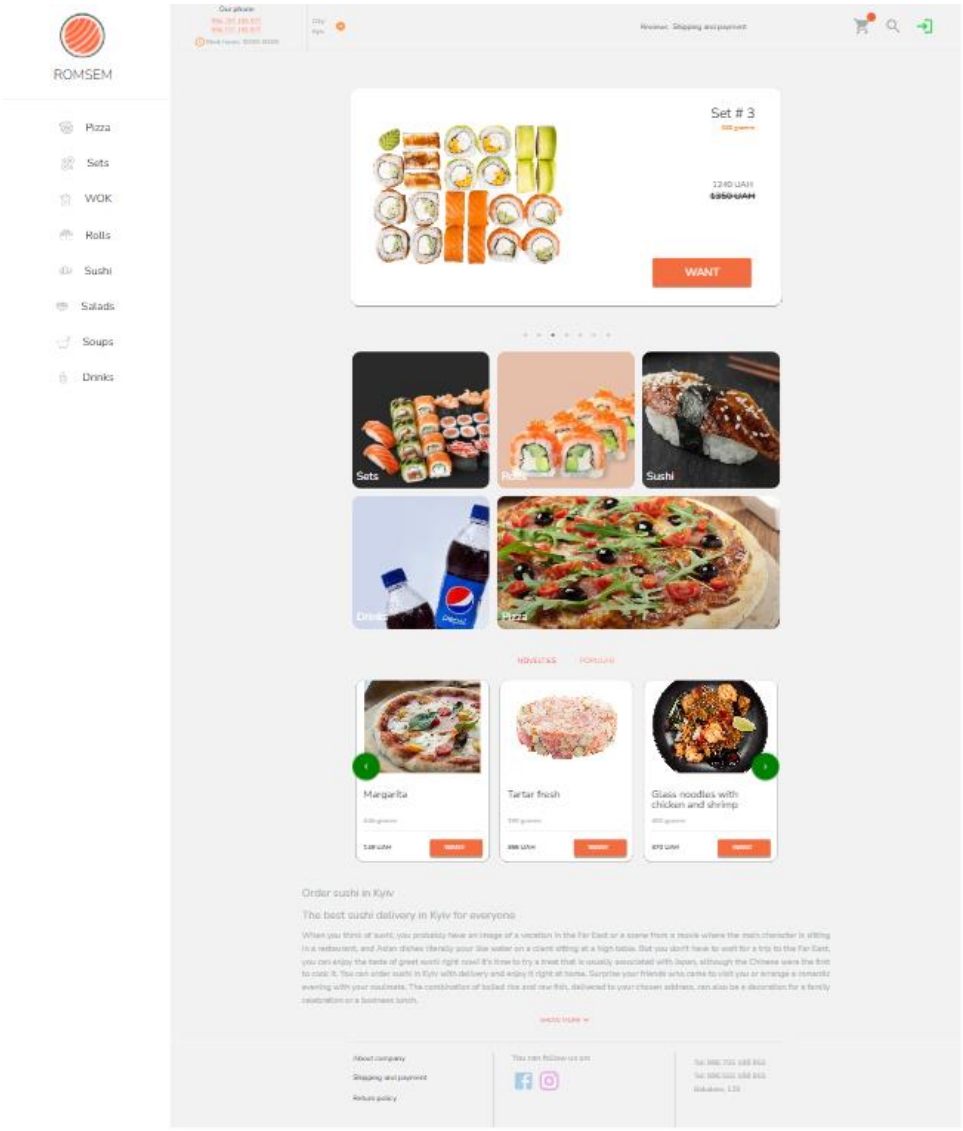

Рисунок 3.1 – Інтерфейс головної сторінки.

Наступним етапом була розробка інтерфейсу сторінки каталогу товарів (рис. 3.2). В цьому випадку важливо було врахувати як і де будуть відображатись товари: їх назва, ціна, кнопка замовлення, необхідно було відображувати назву категорії, а також елементи керування такі як сортування та фільтрація продукції. Було прийнято рішення розділити сторінку на 3 інформаційні блоки:

1. Блок з назвою категорії, функцією фільтрації товарів та їх сортування. При виборі опцій фільтрації та сортування важливо було врахувати момент, щоб випадаючий список не порушував геометрії сторінки та правильно відображався.
2. Блок з товарами має в собі 6 карточок з назвою товару, його вагою, зображенням, ціною та кнопкою замовлення. Також окремо можна виділити блок пагінації, який дозволяє легко змінювати сторінки з товарами.
3. Інформативний блок з описом магазину, який можна розгорнути і дізнатись більше про компанію, а також текст опису змінюється відповідно до обраного міста.

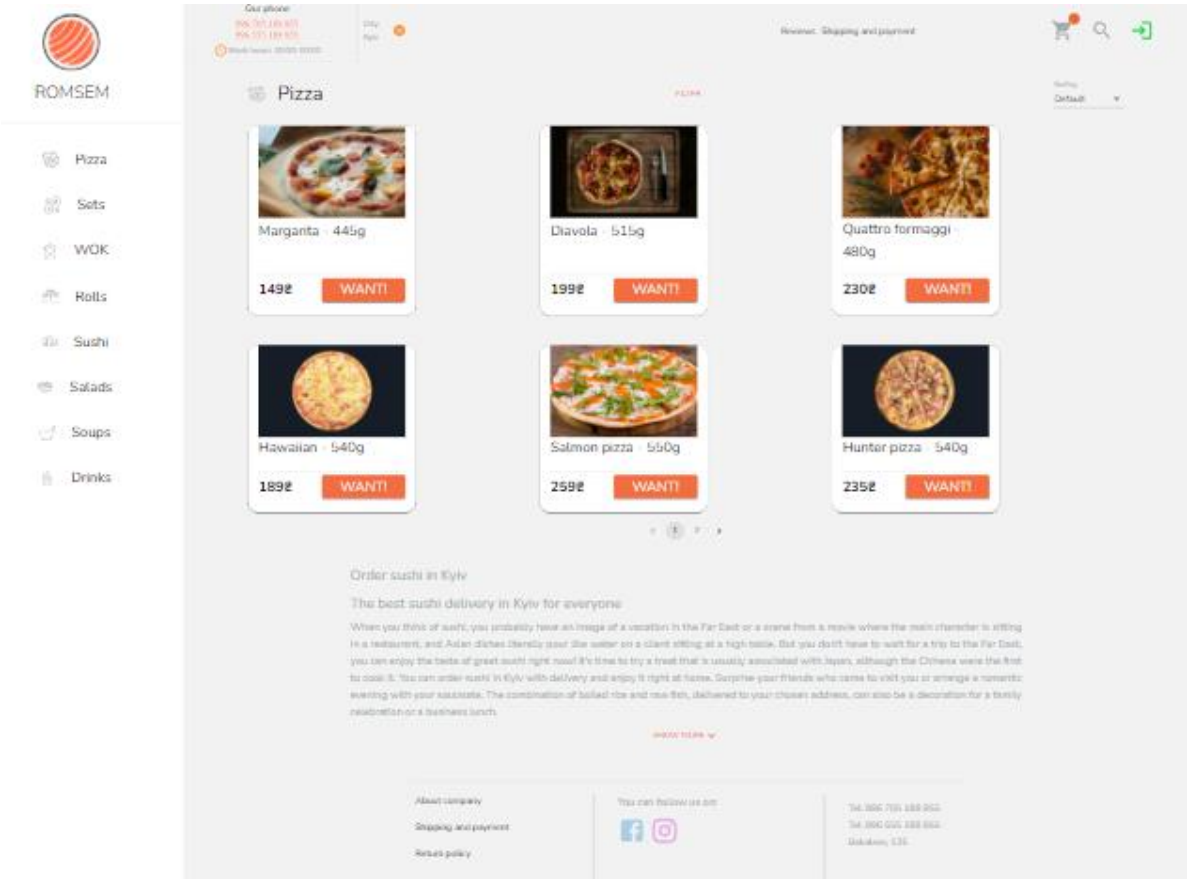

Рисунок 3.2 – Інтерфейс сторінки каталогу товарів.

Наступним етапом була розробка інтерфейсу сторінки одного товару (рис. 3.3). В цьому випадку важливо було врахувати взаємне розміщення

елементів керування у вигляді кнопок «Вперед» та «Назад», рекомендаційного слайдера та елемента з інформацією про товар до якого належали: зображення товару, його назва, вага, ціна, склад, керуючі елементи з вибором кількості та кнопкою замовлення.

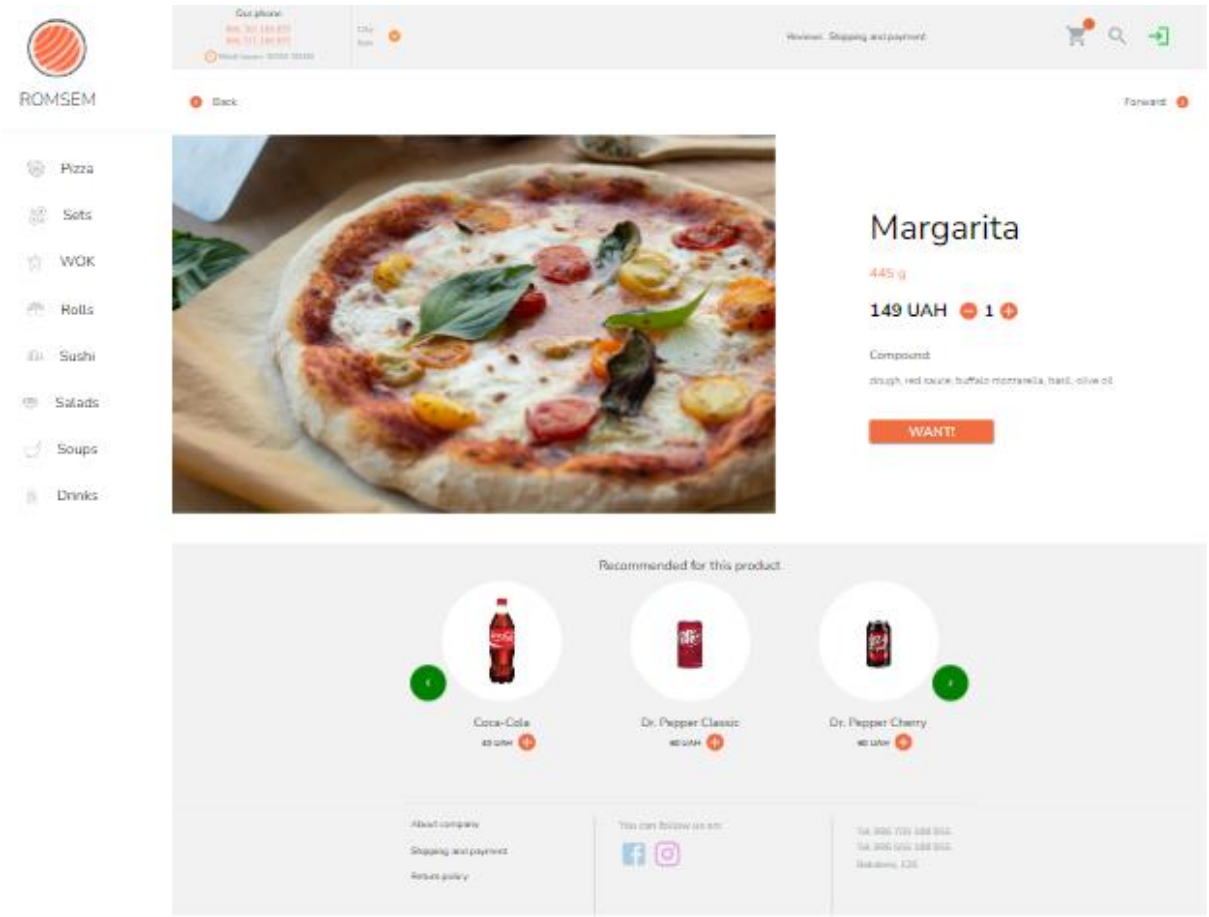

Рисунок 3.3 – Інтерфейс товарної сторінки.

Також необхідно відмітити, що деякі елементи інтерфейсу залишаються статичними і ніяк не змінюються при переході між сторінками, такі як список категорій в лівій частині дизайну, блок з керуючими елементами в верхній частині дизайну та блок з посиланнями в нижній частині дизайну. Блок з керуючими елементами ділиться на 4 частини:

1. Блок з контактною інформацією де наявні номера телефону, а також години відкриття.

2. Блок з вибором міста, при виборі опцій важливо було врахувати момент, щоб випадаючий список не порушував геометрії сторінки та правильно відображався.
3. Навігаційний блок з переходом на інформаційні сторінки.
4. Блок з керуючими елементами, який відкриває доступ до функціоналу кошика, пошуку товарів та механізму авторизації.

3.2. Розробка фронтенду торгового інформаційного порталу

Файлова структура фронтенду (рис. 3.4) представляє собою головну папку «client», всередині якої знаходяться папки «.husky», «node_modules», «public», «src» та конфігураційні файли «.eslintrc.json», «.gitignore», «.prettierrc.json», «package-lock.json», «package.json», «README.md», які являють собою основні налаштування для більш правильної та простої роботи з програмним кодом. В папці «node_modules» знаходяться всі допоміжні пакети, які потрібні при побудові користувацького інтерфейсу. В папці «public» знаходяться статичні файли, такі як: зображення, HTML-файли, документи тощо.

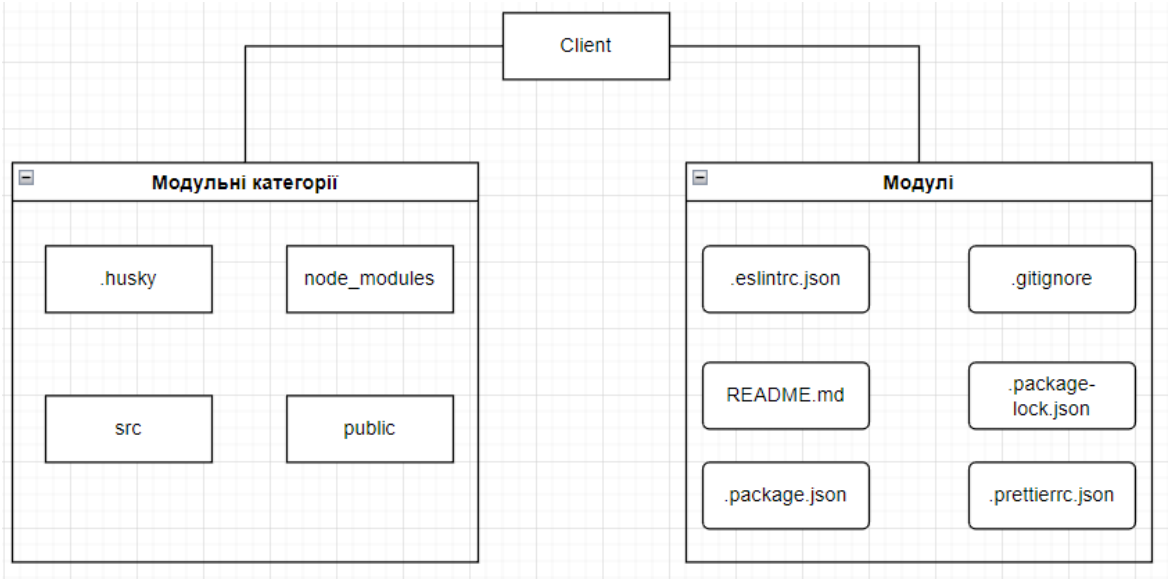

Рисунок 3.4 – Діаграма представлення структури фронтенду.

Архітектура модульної категорії «src» (рис. 3.5) являє собою набір модульних колекцій меншого розміру і головних модулів фронтенду.

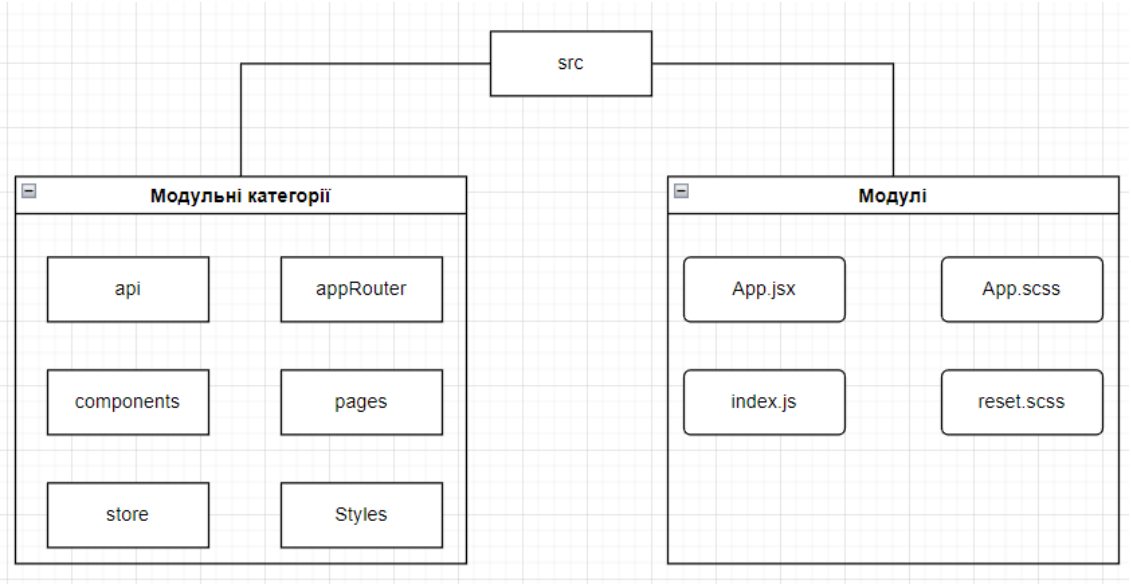

Рисунок 3.5 – Діаграма представлення структури модульної категорії «src».

Модульна категорія «api» (рис. 3.6) містить в собі модулі написані мовою Javascript, що виконують функціонал комунікації фронтенду з бекендом.

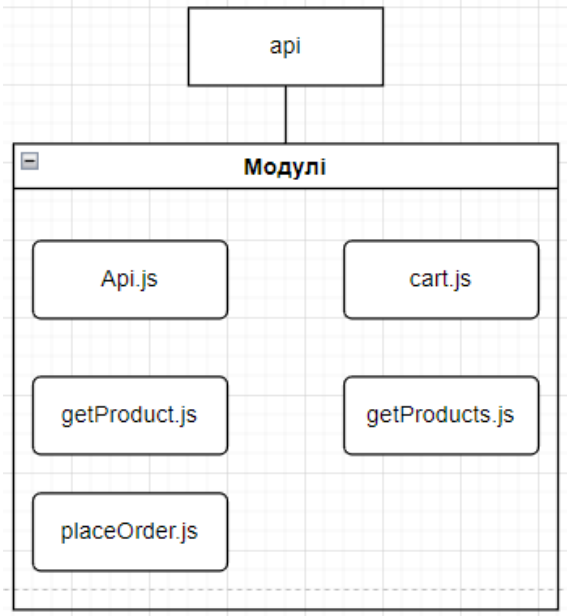

Рисунок 3.6 – Діаграма представлення структури модульної категорії «api».

Сам по собі код програмного модулю має вигляд функцій (рис. 3.7), які можуть приймати в себе аргументи і за допомогою «Axios» звертатись до серверу з метою отримати, відправити чи видалити дані. Для того, щоб не було конфліктів імен при виклику деяких функцій довелось створювати нові файли.

```
export const getNoveltieProduct = () =>
  axios.get('/api/products/filter?promo=new');

export const getPopularProduct = () =>
  axios.get('/api/products/filter?promo=popular');

export const getAllProducts = () => axios.get('/api/products');

export const registrateCustomer = (newCustomer) =>
  axios.post('api/customers', newCustomer);
```

Рисунок 3.7 – Фрагмент коду програмного модулю «Api.js».

Програмний модуль «appRouter» (рис. 3.8) містить в собі функціонал, що забезпечує можливість клієнту використовувати навігацію по додатку без необхідності очікувати на завантаження сторінки.

```
function AppRouter() {
  return (
    <Routes>
      <Route path="/" element={<Main />} />
      <Route path="/reviews" element={<Reviews />} />
      <Route path="/shipping" element={<Shipping />} />
      <Route path="/ordering" element={<Ordering />} />
      <Route path="/about" element={<About />} />
      <Route path="/products" element={<Products />} />
      <Route path="/products/:id" element={<Product />} />
      <Route path="/cart" element={<CartMob />} />
      <Route path="/return" element={<Return />} />
      <Route path="/backError" element={<ErrorPage backendError={1} />} />
      <Route path="*" element={<ErrorPage backendError={0} />} />
    </Routes>
  );
}
```

Рисунок 3.8 – Фрагмент коду програмного модулю «AppRouter.jsx»

Модульна категорія «components» (рис. 3.9) містить в собі основні елементи користувацького інтерфейсу, в залежності від складності того чи іншого елемента він також може бути розділений на окремі частини.

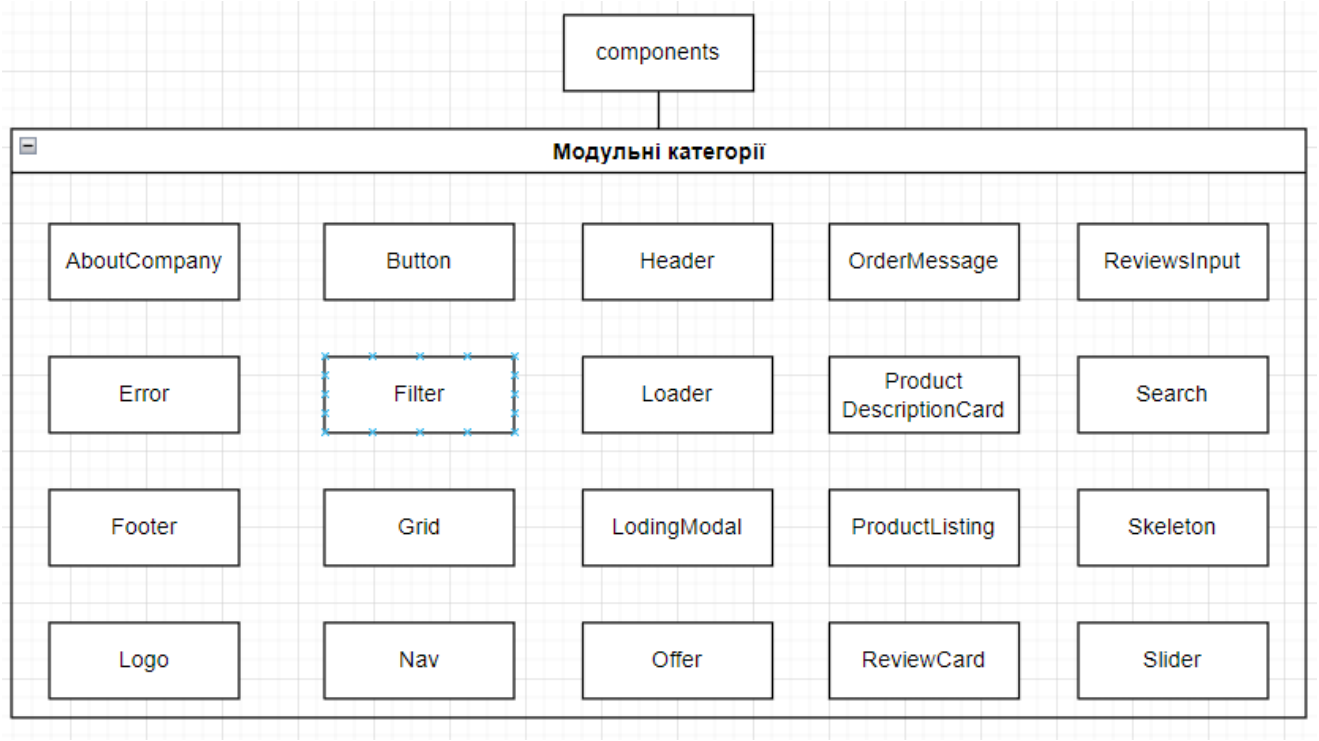

Рисунок 3.9 – Діаграма представлення структури модульної категорії «components».

Головна відмінність подібних елементів в тому, що вони реалізують відображення окремих елементів користувацького інтерфейсу та окрім простого відображення можуть мати в собі логічні конструкції та функціональну наповненість (рис. 3.10).

```
export default function Cart(props) {
  const {open, setOpen, scroll} = props;
  const handleClose = () => {
    setOpen(false);
  };

  const descriptionElementRef = React.useRef(null);
  React.useEffect(() => {
    if (open) {
      const {current: descriptionElement} = descriptionElementRef;
      if (descriptionElement !== null) {
        descriptionElement.focus();
      }
    }
  }, [open]);
```

Рисунок 3.10 – Фрагмент коду програмного модулю «Cart.jsx».

Модульна категорія «pages» (рис. 3.11) являє собою місце, в якому поєднуються різні локальні елементи з категорії «components» та на цьому рівні формується розмітка конкретної сторінки.

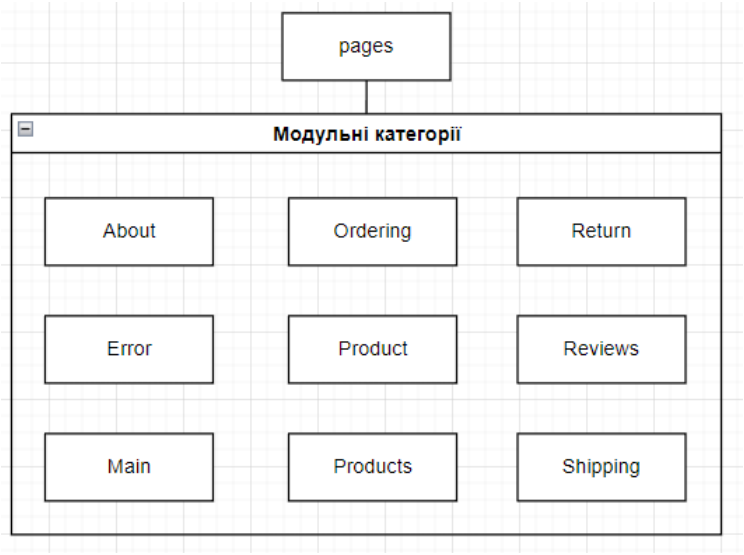

Рисунок 3.11 – Діаграма представлення структури модульної категорії «pages».

Як правило, на сторінках може використовуватись технологія умовного рендерингу (рис. 3.12), щоб надати більшої гнучкості при роботі з даними, які змінюються та використовувати ресурси клієнта більш ефективно. Також, необхідно зазначити, що на цьому рівні організації коду

небажано реалізовувати ніяку логіку, але іноді через специфіку застосування деяких технологічних особливостей платформи, правило можна порушити.

```
{components.length ? (
  <Grid>
    <ProductsPagination
      productsQuantity={productsQuantity}
      perPageProducts={perPageProducts}
    />
  </Grid>
) : null}
```

Рисунок 3.12 – Фрагмент коду програмного модулю «Products.jsx» з технологією умовного рендерингу.

Модульна категорія «store» (рис. 3.13) являє собою окрему технологію під назвою «Redux». Redux – це сховище для зберігання стану змінних у додатку. Redux створює процес і процедури для взаємодії зі сховищем, щоб компоненти не просто оновлювали або читали сховище випадковим чином.

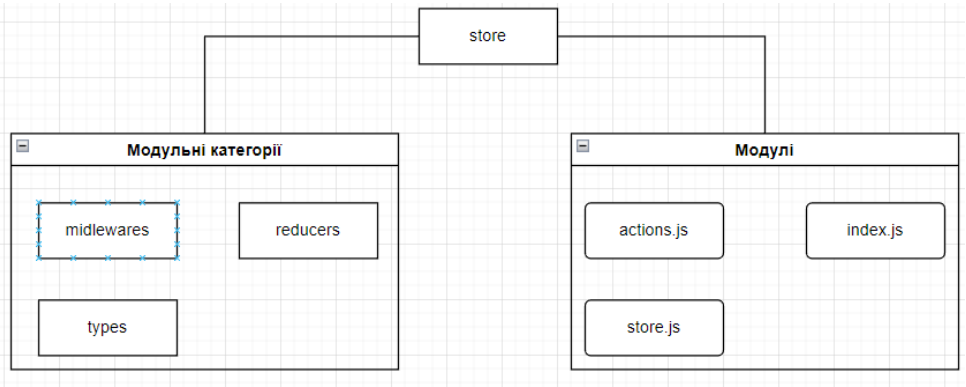

Рисунок 3.13 – Діаграма представлення структури модульної категорії «store».

Якщо коротко, то Redux – це спосіб управління "станом", або можна сказати, що це кеш або сховище, до якого всі компоненти можуть мати структурований доступ. Основними його елементами є: state, action, reducer.

State – це глобальне сховище, в якому зберігається інформація (рис. 3.14).

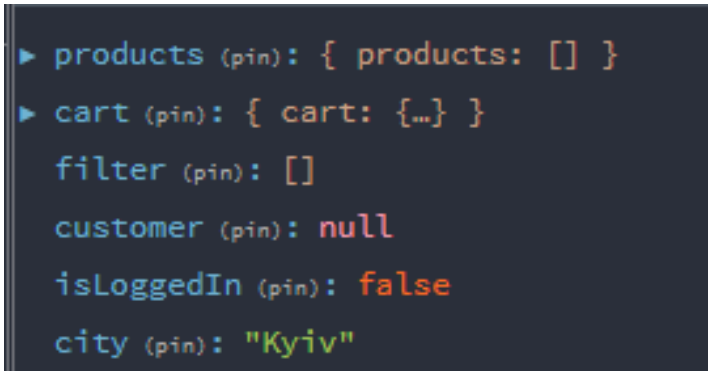

Рисунок 3.14 – Redux State.

Action – це функція, яка приймає в себе 2 атрибути: тип інформації і самі дані, які передаються до глобального сховища (рис. 3.15).

```
export const addToCart = (data) => async (dispatch) => {
  dispatch({ type: ADD_TO_CART, payload: data });
};
```

Рисунок 3.15 – Приклад Redux Action.

Reducer – це функція, яка приймає в себе 2 атрибути: актуальний стан сховища та action (рис. 3.16). Сам по собі reducer виконує роль контролера, який на вході перевіряє тип даних, що надходять до нього і виконує відносно цього типу певні маніпуляції зі сховищем даних, наприклад, видаляє дані зі сховища, модифікує їх, додає нові тощо.

```
const cityReducer = (state = '', action) => {
  switch (action.type) {
    case SET_CITY: {
      return action.payload.city;
    }
    default: {
      return state;
    }
  }
};
```

Рисунок 3.16 – Приклад Redux Reducer.

3.3. Розробка бекенду торгового інформаційного порталу

Файлова структура бекенду (рис. 3.17) представляє собою головну папку «server», всередині якої знаходяться папки «commonHelpers», «node_modules», «config», «controllers», «models», «routes», «static», «validation» та конфігураційні файли «.env», «.gitignore», «package-lock.json», «package.json», які являють собою основні налаштування для більш правильної та простої роботи з кодом. В папці «node_modules» знаходяться всі допоміжні пакети, які потрібні при побудові користувацького інтерфейсу. В папці «static» знаходяться статичні файли, такі як: зображення, HTML-файли, документи тощо. Також на цьому рівні файлової організації коду знаходиться файл «server.js», який є головним виконавчим бекендовим сценарієм для правильної роботи системи.

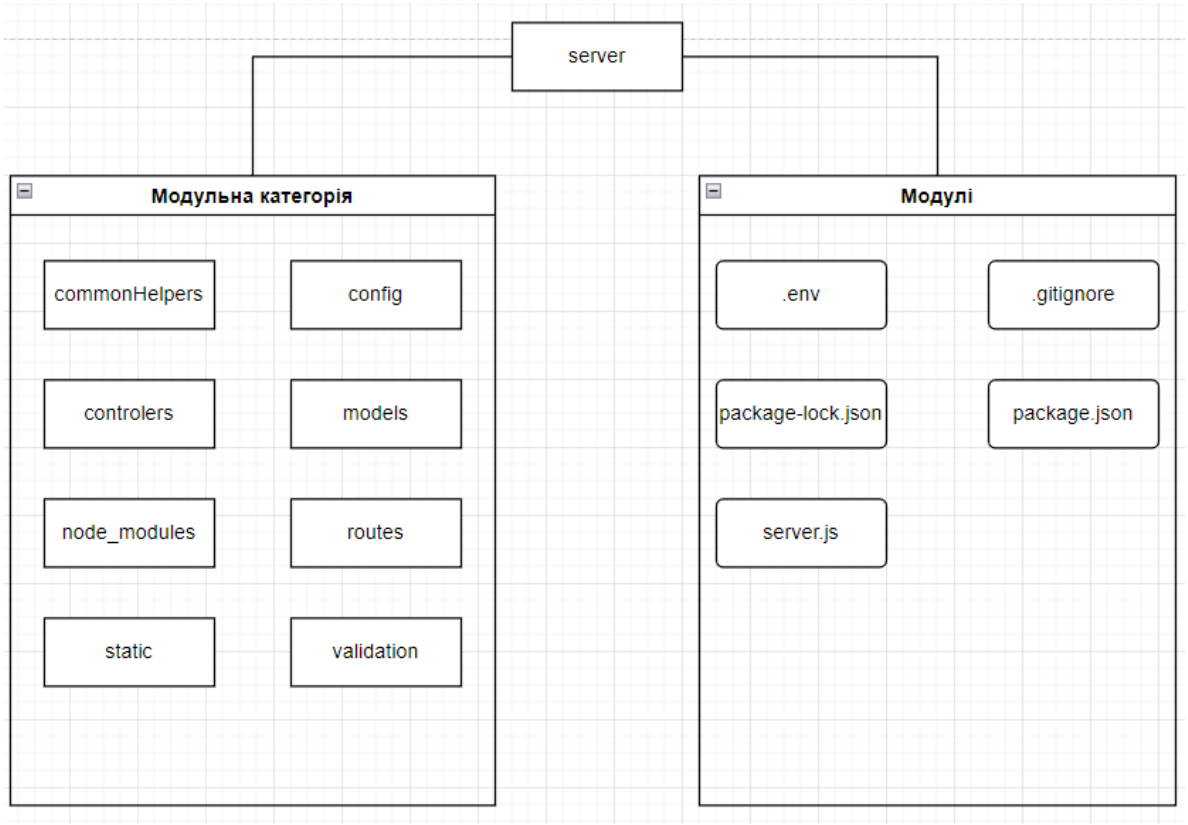

Рисунок 3.17 – Діаграма представлення структури бекенду.

Модульна категорія «commonHelpers» (рис. 3.18) являє собою набір допоміжного функціоналу, який був розроблений для більш гнучкого і простого функціонування системи, а також для реалізації невеликих сценаріїв, наприклад, відправка повідомлень на електронну пошту, обробка пошукового рядка, перевірка наявності тої чи іншої продукції, коректна валідація JSON формату тощо.

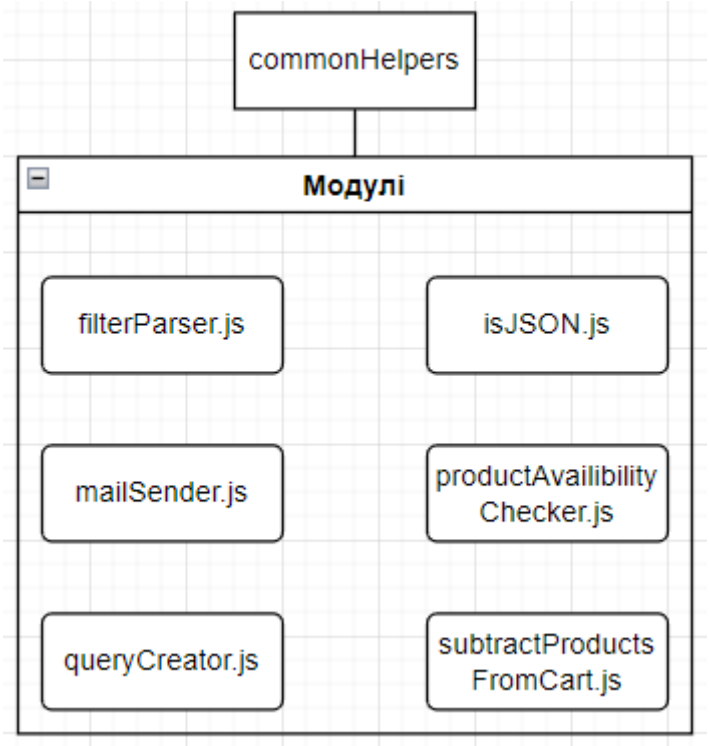

Рисунок 3.18 – Діаграма представлення структури модульної категорії «commonHelpers».

Модульна категорія «config» (рис. 3.19) являє собою набір модулів з налаштуваннями доступу різних користувачів до вмісту додатка, тобто тут визначається чи у користувача є права на те, щоб редагувати вміст платформи в якості адміністратора чи він є звичайним клієнтом, який може лише переглядати вміст та створювати замовлення, також тут знаходиться інформація по підключенню до бази даних.

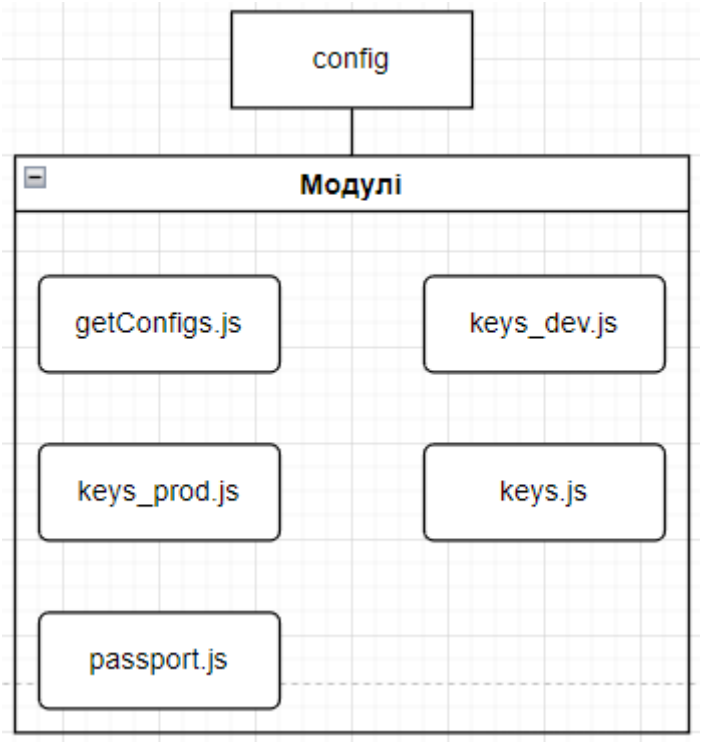

Рисунок 3.19 – Діаграма представлення структури модульної категорії «config».

Модульна категорія «controllers» (рис. 3.20) являє собою набір основного функціоналу бекенду. Саме тут реалізований збір та обробка даних. Основною частиною функціоналу є, відповідно до запиту, який надходить з клієнта, виконати збір та обробку інформації з бази даних.

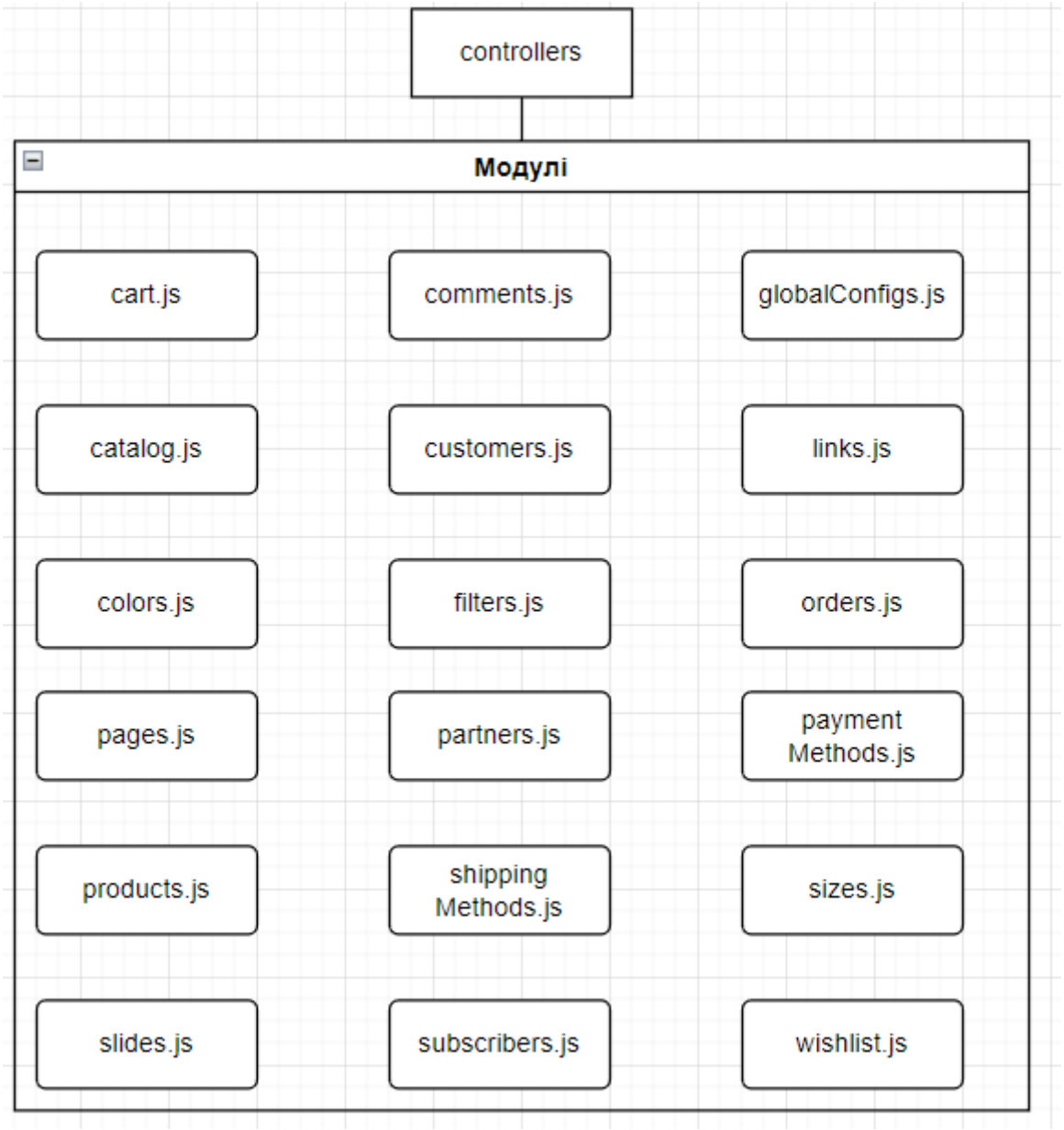

Рисунок 3.20 – Діаграма представлення структури модульної категорії «controllers».

Для прикладу розглянемо механізм отримання і обробки інформації з бази даних для товарів (рис. 3.21). Тут можна бачити, що сам код знаходиться в функції getProducts, яка приймає на вхід 3 атрибути: req (вхідний запит), res (відповідь від сервера) та next (перехід до наступного запиту). Напочатку також ініціалізуються 3 змінні: perPage (кількість товарів яка буде відправлена за один раз), startPage (початкова сторінка

товару), sort (метод сортування, який отримується за допомогою серверної службової функції з рядка запиту на фронтенді). Після цього відбувається пошук товарів, які задовольняють формулу (3.1):

$$K = (S * P) - P \tag{3.1}$$

де, К – кількість елементів які необхідно пропустити, S – початкова сторінка, P – кількість товарів на одній сторінці.

Для правильної відповіді від серверу необхідно, щоб товарів було не більше, ніж вказано в змінній perPage, після цього до зібраної інформації використовується один з типів сортування, після чого зібрана інформація відправляється користувачу.

```
exports.getProducts = (req, res, next) => {
  const perPage = Number(req.query.perPage);
  const startPage = Number(req.query.startPage);
  const sort = req.query.sort;

  Product.find()
    .skip(startPage * perPage - perPage)
    .limit(perPage)
    .sort(sort)
    .then(products => res.send(products))
    .catch(err =>
      res.status(400).json({
        message: `Error happened on server: "${err}" `
      })
    );
};
```

Рисунок 3.21 – Фрагмент кодової бази «products.js».

Модульна категорія «models» (рис. 3.22) являє собою набір схем, які необхідні при отриманні інформації з бази даних. Саме тут визначаються які саме поля та їх тип очікує отримати сервер.

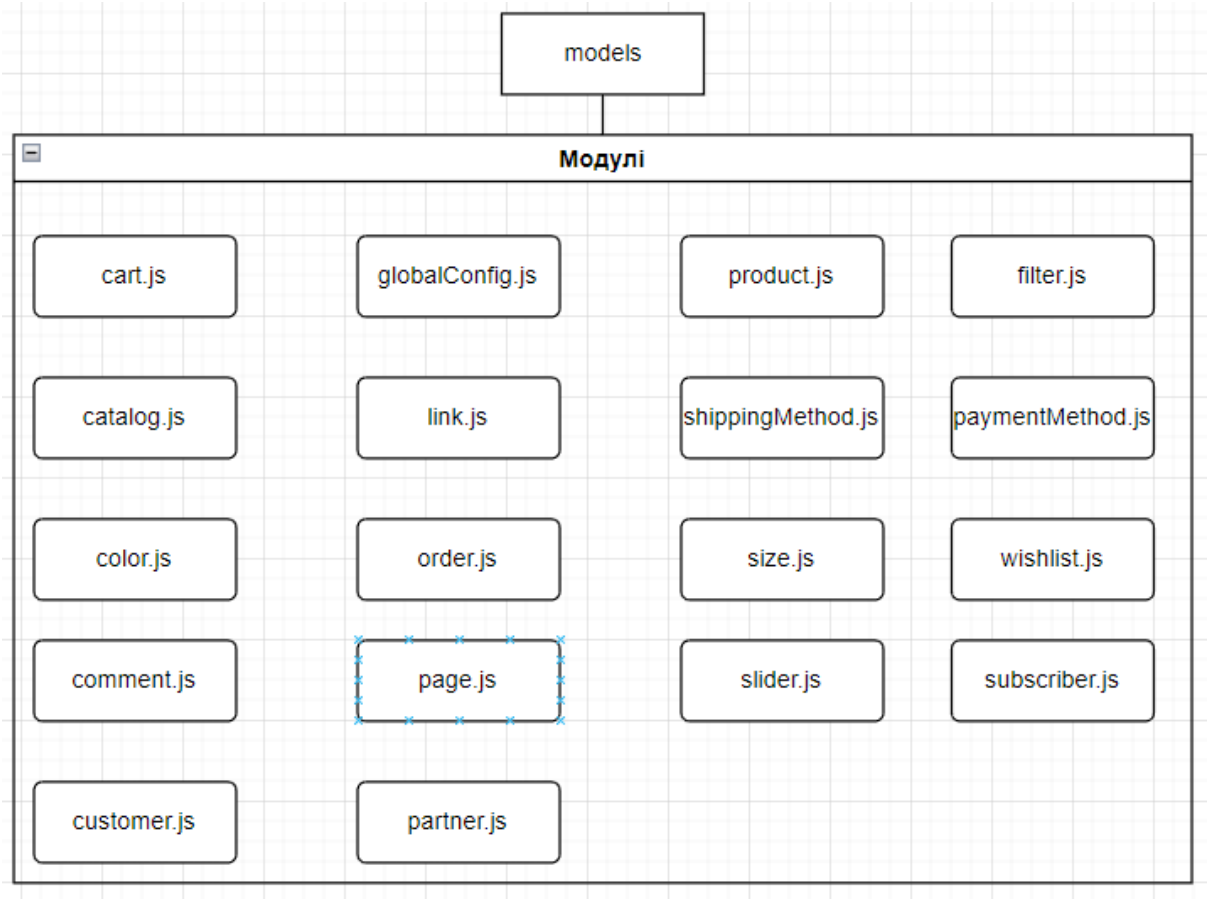

Рисунок 3.22. – Діаграма представлення структури модульної категорії «models».

Обов’язковим для всіх схем є поле type, яке має тип Schema.Types.ObjectId (рис. 3.23). Це поле необхідно для індексації кожного об’єкту з даними в MongoDB і без нього інформація просто не зможе потрапити до користувача. Інші поля вже є опціональними і залежать від функціональних вимог проекту де застосовуються.

```
const CartSchema = new Schema(
  {
    customerId: {
      type: Schema.Types.ObjectId,
      ref: "customers",
      required: true
    },
```

Рисунок 3.23. – Фрагмент кодової бази «Cart.js».

Модульна категорія «routes» (рис. 3.24) являє собою набір ендпоінтів, які потрібні для того, щоб була можлива комунікація між сервером та клієнтом за допомогою REST API. Саме тут відбувається прийом запитів від фронтенду та після обробки переводить потік виконання до сценаріїв, реалізованих в модульній категорії «controllers».

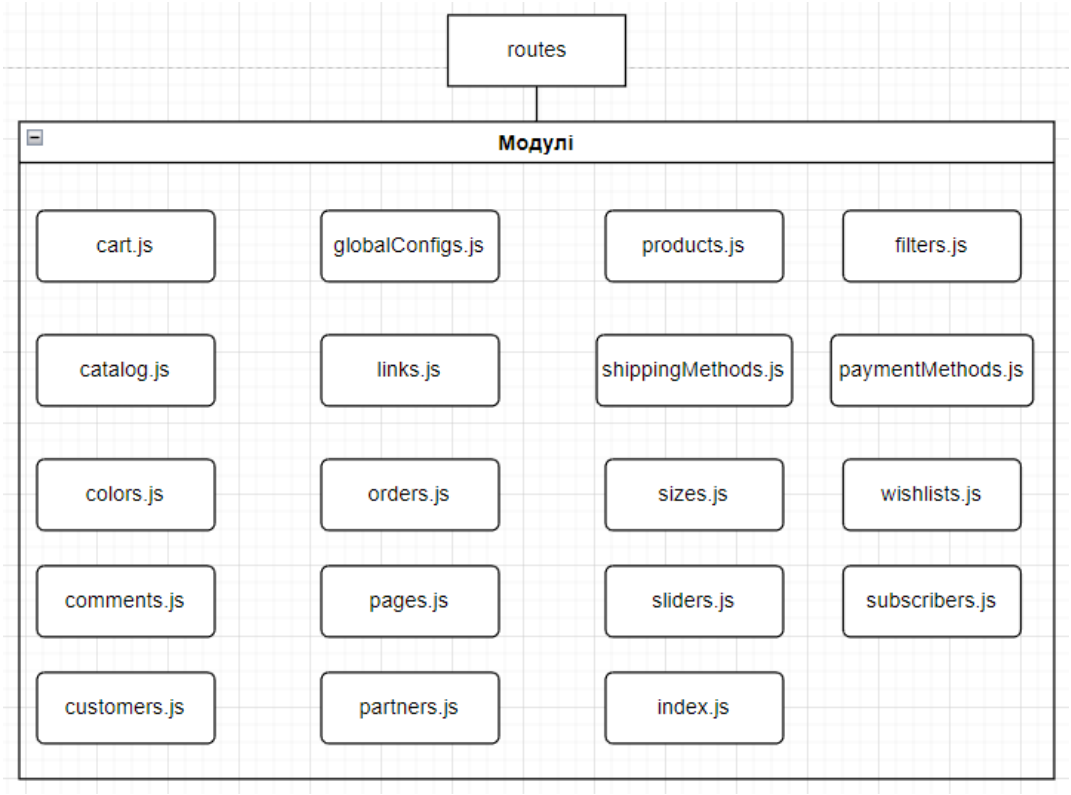

Рисунок 3.24. – Діаграма представлення структури модульної категорії «routes».

Модульна категорія «validation» (рис. 3.25) являє собою механізм серверної валідації отриманих даних. Це необхідно, оскільки зібрані дані на фронтенді не завжди одразу відправляються на сервер, іноді вони якось модифікуються і в цьому випадку, щоб уникнути збоїв при отриманні таких даних на бекенді, використовується такий набір інструкцій для того, щоб уникнути збоїв і неправильно отриманих даних від користувачів.

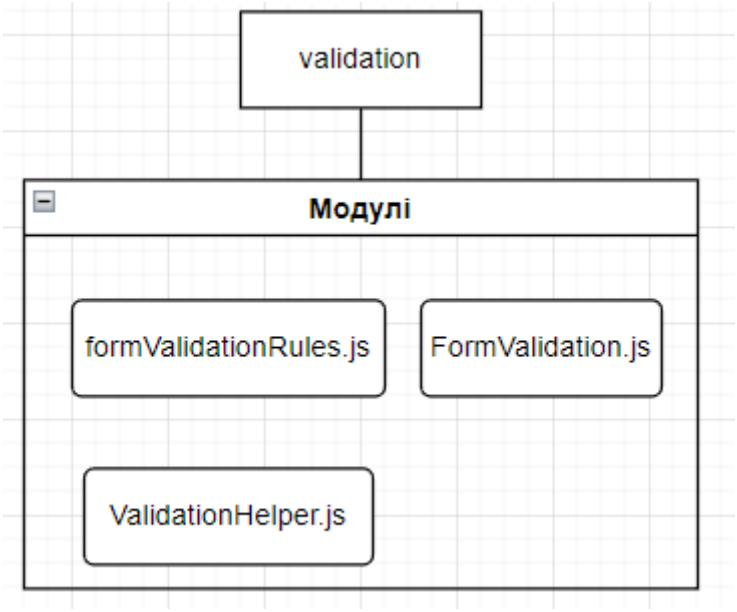

Рисунок 3.25 – Діаграма представлення структури модульної категорії «validation».

3.4. Тестування торгового інформаційного порталу

Тестування системи таких масштабів має в собі декілька складностей, оскільки необхідно перевірити не тільки роботу загального функціоналу та того як він відображається кінцевому користувачу, а ще і того як ця система працює загалом при нестандартних та екстремальних умовах високої завантаженості, коли ним користується не один клієнт одночасно, а десять чи навіть 20. Це необхідно для оцінки того наскільки всі компоненти платформи ефективно взаємодіють один з одним і у випадку невідповідності очікуванням прийти до тих чи інших вжитих заходів по покращенню результатів роботи платформи.

3.4.1. Автоматизоване тестування

Автоматизоване тестування – це застосування програмних засобів для автоматизації ручного процесу перевірки та валідації програмного

продукту, який виконується людиною-тестувальником. Більшість сучасних гнучких і DevOps програмних проектів зараз включають автоматизоване тестування з самого початку [5].

У випадку розробки високонавантаженої торгової інформаційної платформи необхідно перевірити швидкість завантаження головної сторінки на стороні клієнта, швидкість взаємодії фронтенду та бекенду частини при обробці запитів на отримання великої кількості товарів для одного, п’яти та десяти користувачів, які одночасно користуються додатком, провести стрес-тести по завантаженості центрального процесора та оперативної пам’яті бекенду, під кінець буде проведений краш-тест застосунку, направлений на доведення додатка до непрацюючого стану за допомогою інтенсивної симуляції різних сценаріїв взаємодії великої кількості користувачів.

3.4.1.1. Тестові сценарії для одного користувача:

Сценарій «Головна сторінка»:

При написанні тестового сценарію для одного користувача необхідно зімітувати наступні дії: запуск браузера за замовчуванням, створення нової сторінки, перехід за посиланням на головну сторінку, фіксація часу початку завантаження головної сторінки, фіксація часу закінчення завантаження головної сторінки, підрахунок витраченого часу, порівняння з очікуваним результатом. Підрахунок витраченого часу відбувається за наступною формулою (3.2):

$$D = E - S \tag{3.2}$$

де, D – витрачений час одного користувача, E – час кінця підрахунку, S – час початку підрахунку.

Тестовий сценарій вважається успішно пройденим коли задовольняється умова (3.3):

$$D < K \tag{3.3}$$

де, D – витрачений час одного користувача, K – константа, яка дорівнює 3 секундам.

Відповідно тестовий сценарій вважається пройденим не успішно коли задовольняється умова (3.4):

$$D > K \tag{3.4}$$

де, D – витрачений час одного користувача, K – константа, яка дорівнює 3 секундам.

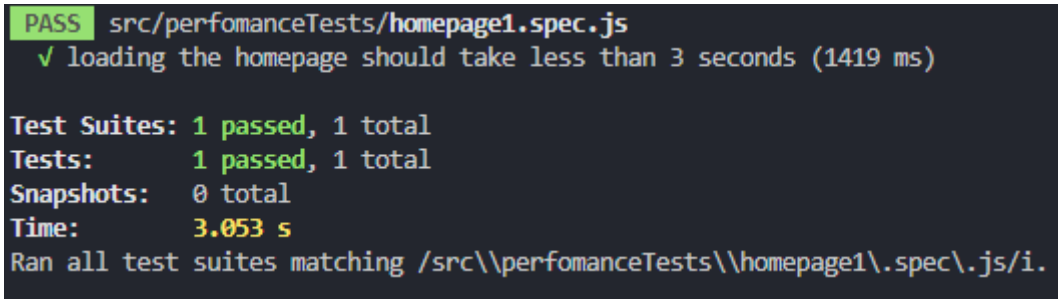

Рисунок 3.35 – Результат виконання тестового сценарію «homepage1»

Сценарій «Сторінка товарів»:

При написанні тестового сценарію для одного користувача необхідно зімітувати наступні дії: запуск браузера за замовчуванням, створення нової сторінки, перехід за посиланням на головну сторінку, перехід за посиланням на сторінку товарів, фіксація часу початку завантаження сторінки товарів, фіксація часу закінчення завантаження сторінки товарів, підрахунок витраченого часу, порівняння з очікуваним результатом. Підрахунок витраченого часу відбувається за наступною формулою (3.5):

$$D = E - S \tag{3.5}$$

де, D – витрачений час одного користувача, E – час кінця підрахунку, S – час початку підрахунку.

Тестовий сценарій вважається успішно пройденим коли задовольняється умова (3.6):

$$D < K \tag{3.6}$$

де, D – витрачений час одного користувача, K – константа, яка дорівнює 3 секундам.

Відповідно тестовий сценарій вважається пройденим не успішно коли задовольняється умова (3.7):

$$D > K \tag{3.7}$$

де, D – витрачений час одного користувача, K – константа, яка дорівнює 3 секундам.

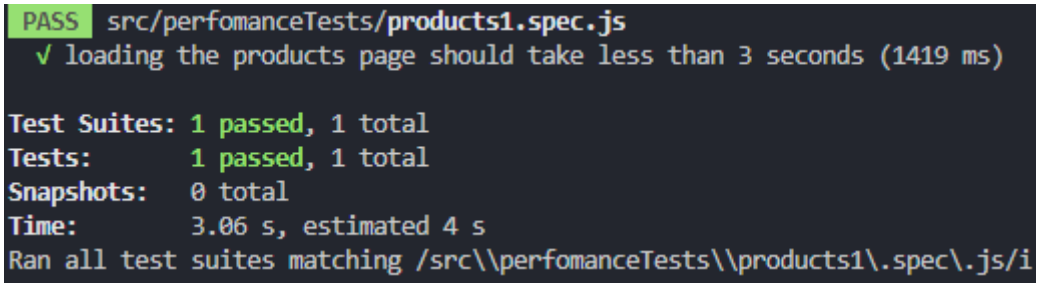

Рисунок 3.36 – Результат виконання тестового сценарію «products1»

3.4.1.2. Тестові сценарії для п’яти та десяти користувачів:

Сценарій «Головна сторінка»:

При написанні тестового сценарію для п’яти та десяти користувачів необхідно зімітувати наступні дії: запуск браузера за замовчуванням, створити змінні кількості користувачів та пустий список куди будуть надходити результати підрахунку витраченого часу, для кожного користувача створюється нова сторінка, створюється функція, яка для кожного користувача виконує перехід на головну сторінку та фіксує час початку та кінця її завантаження, після цього вираховується загальна кількість витраченого часу та порівнюється з очікуваннями. Підрахунок витраченого часу відбувається за наступною формулою (3.8, 3.9):

$$D = E - S \tag{3.8}$$

де, D – витрачений час одного користувача, E – час кінця підрахунку, S – час початку підрахунку.

$$T = D_1 + D_n \quad (3.9)$$

де, T – загальний витрачений час, D_1 – витрачений час одного користувача, n – порядковий номер користувача.

Тестовий сценарій вважається успішно пройденим коли задовольняється умова (3.10):

$$T < K * N \quad (3.10)$$

де, T – витрачений час, K – константа, яка дорівнює 3 секундам, N – кількість користувачів.

Відповідно тестовий сценарій вважається пройденим не успішно коли задовольняється умова (3.11):

$$T > K * N \quad (3.11)$$

де, T – витрачений час, K – константа, яка дорівнює 3 секундам, N – кількість користувачів.

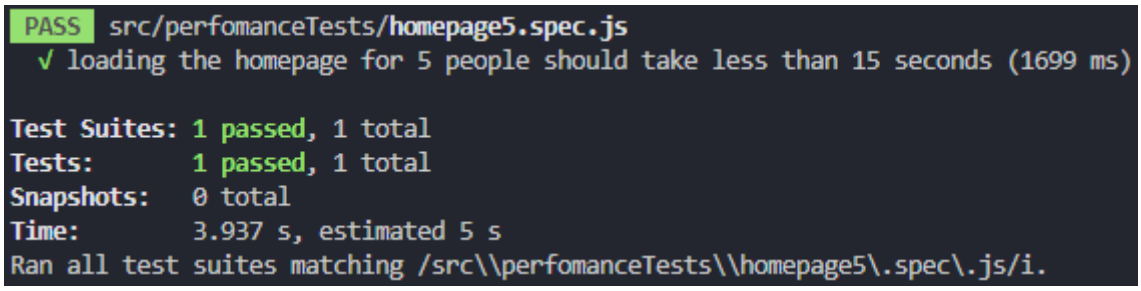

Рисунок 3.37 – Результат виконання тестового сценарію «homepage5»

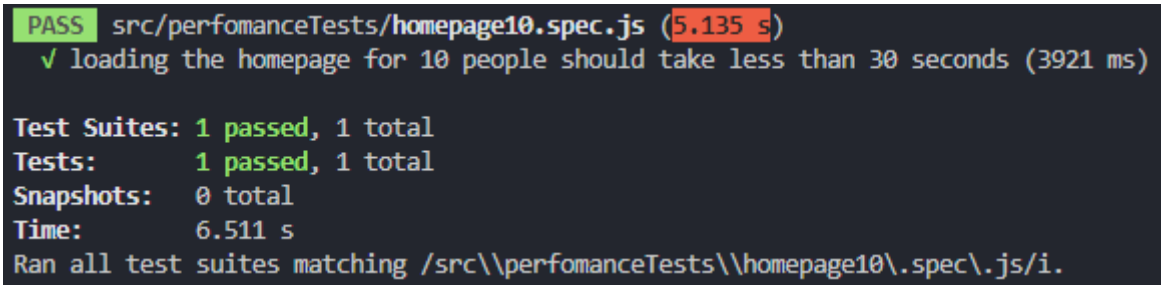

Рисунок 3.38 – Результат виконання тестового сценарію «homepage10»

Сценарій «Сторінка товарів»:

При написанні тестового сценарію для п’яти та десяти користувачів необхідно зімітувати наступні дії: запуск браузера за замовчуванням, створити змінні кількості користувачів та пустий список куди будуть надходити результати підрахунку витраченого часу, для кожного користувача створюється нова сторінка, створюється функція, яка для кожного користувача виконує перехід на головну сторінку, а потім на сторінку з товарами та фіксує час початку та кінця її завантаження, після цього вираховується загальна кількість витраченого часу та порівнюється з очікуваннями. Підрахунок витраченого часу відбувається за наступною формулою (3.12, 3.13):

$$D = E - S \tag{3.12}$$

де, D – витрачений час одного користувача, E – час кінця підрахунку, S – час початку підрахунку.

$$T = D_1 + D_n \tag{3.13}$$

де, T – загальний витрачений час, D_1 – витрачений час одного користувача, n – порядковий номер користувача.

Тестовий сценарій вважається успішно пройденим коли задовольняється умова (3.14):

$$T < K * N \tag{3.14}$$

де, T – витрачений час, K – константа, яка дорівнює 3 секундам, N – кількість користувачів.

Відповідно тестовий сценарій вважається пройденим не успішно коли задовольняється умова (3.15):

$$T > K * N \tag{3.15}$$

де, T – витрачений час, K – константа, яка дорівнює 3 секундам, N – кількість користувачів.

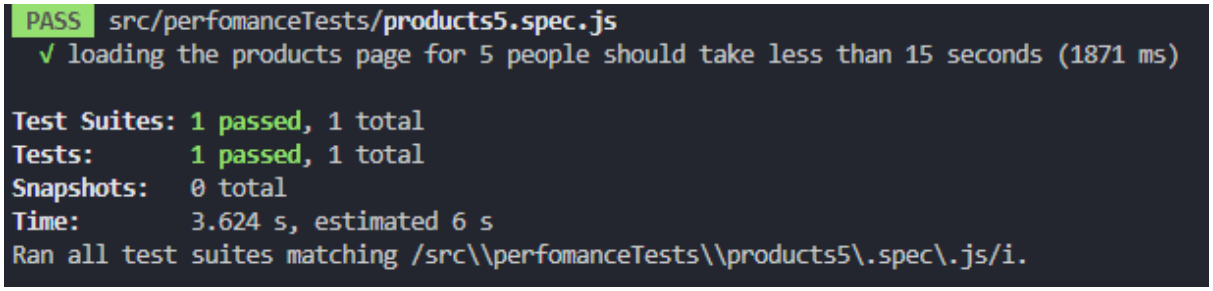

Рисунок 3.39 – Результат виконання тестового сценарію «products5»

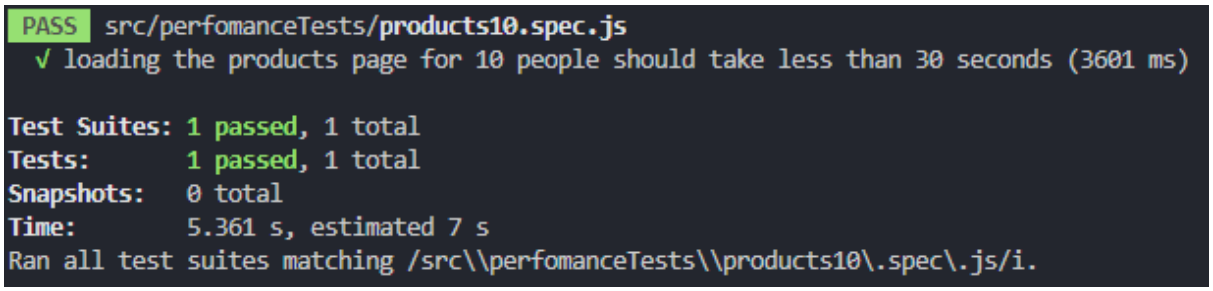

Рисунок 3.40 – Результат виконання тестового сценарію «products10»

3.4.1.3. Стрес-тест завантаженості центрального процесора:

При написанні тестового сценарію для стрес-тесту завантаженості центрального процесора необхідно виконати наступні дії: створити змінні тривалості тесту та відсоткового критичного порогу при якому тест переривається, створити функцію моніторингу за станом комплектуючих з механізмом припинення тестування, створити цикл, в якому проводяться операції, які навантажують центральний процесор. Підрахунок кінця тестування визначається за формулою (3.16):

$$D = N - S \tag{3.16}$$

де, D – час, протягом якого тестування триває, N – актуальний час, S – час початку тестування.

Тестовий сценарій вважається успішно пройденим коли задовольняється умова (3.17):

$$D > T \tag{3.17}$$

де, T – визначений час тестування, D – час, протягом якого тестування триває.

Відповідно тестовий сценарій вважається пройденим не успішно коли навантаження на процесор перевищує встановлений поріг.

Під час тестування були обрані наступні параметри (3.18):

$$T = 10000, K = 70 \quad (3.18)$$

де, T – визначений час тестування, K – відсотковий критичний поріг.

В результаті, тест був пройдений успішно (рис. 3.41).

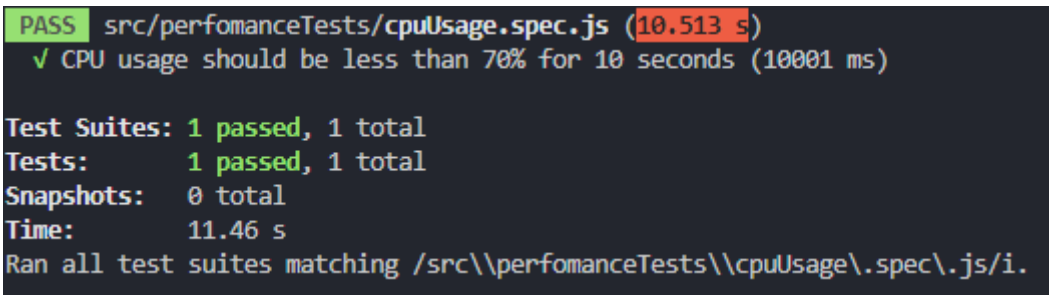

Рисунок 3.41 – Результат виконання тестового сценарію «cpuUsage»

3.4.1.4. Стрес-тест завантаженості оперативної пам’яті:

При написанні тестового сценарію для стрес-тесту завантаженості оперативної пам’яті необхідно виконати наступні дії: створити змінні тривалості тесту та критичного порогу при якому тест переривається, створити функцію моніторингу за станом комплектуючих з механізмом припинення тестування, створити цикл, в якому проводяться операції, які заповнюють оперативну пам’ять, після закінчення тестування необхідно звільнити пам’ять. Підрахунок кінця тестування визначається за формулою (3.19):

$$D = N - S \quad (3.19)$$

де, D – час, протягом якого тестування триває, N – актуальний час, S – час початку тестування.

Тестовий сценарій вважається успішно пройденим коли задовольняється умова (3.20):

$$D > T \quad (3.20)$$

де, T – визначений час тестування, D – час, протягом якого тестування триває.

Відповідно тестовий сценарій вважається пройденим не успішно коли навантаження на процесор перевищує встановлений поріг. Встановлений поріг визначається за формулою (3.21):

$$M = L * 1024 * 1024 \quad (3.21)$$

де, M – критичний поріг в мегабайтах, L – параметр, який необхідно привести до мегабайту.

Під час тестування були обрані наступні параметри (3.22):

$$T = 10000, K = 50 \quad (3.22)$$

де, T – визначений час тестування, K –критичний поріг в мегабайтах.

В результаті, тест був пройдений успішно (рис. 3.42).

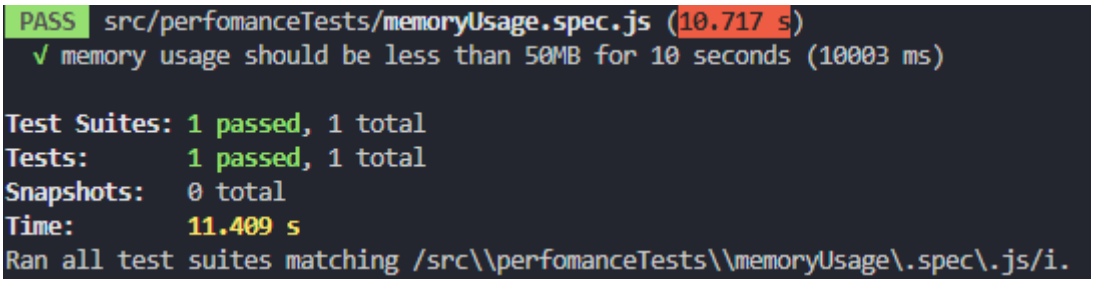

Рисунок 3.42 – Результат виконання тестового сценарію «memoryUsage»

3.4.1.5. Краш-тест торгового інформаційного порталу:

При написанні тестового сценарію для краш-тесту торгового інформаційного порталу необхідно виконати наступні дії: створити змінні тривалості тесту та порогу користувачів при якому тест переривається, створити функцію моніторингу за станом системи та виведення кінцевих результатів, створити цикл, в якому симулюються випадкові дії користувачів. Підрахунок середнього часу відповіді фронтенду та бекенду відбувається за формулою (3.23):

$$R = \frac{U_1 + U_n}{N} \quad (3.23)$$

де, R – середній час відповіді, N – кількість користувачів, U_1 – тривалість відповіді на запит першого користувача, U_n – порядковий номер користувача.

Даний сценарій направлений на те, щоб виявити максимально можливе навантаження на систему, тому тут немає визначення коли тест пройдений успішно, а коли ні. Також важливо зазначити, що особливість цього тесту полягає в тому, що користувачі не покидають порталу і постійно взаємодіють з ним відправляючи нові і нові запити. Після його завершення відбувається підрахунок і вивід інформації про результати тестування (рис. 3.43). Отже, одночасно користуватись порталом можуть 86 користувачів.

```
All users: 100
Users Before Crush: 86
Average Frontend Perfomance: 4.9s
Average Backend Perfomance: 4.3s
Time consumed: 22 min
```

Рисунок 3.43 – Результат виконання тестового сценарію «crushTest»

В результаті тестування були отримані наступні результати (табл. 3.1):

Таблиця 3.1 – Результати тестування без застосування віртуальної машини.

№	Назва тесту	Результат
1	homepage1.spec.js	1419мс
2	products1.spec.js	1419мс
3	homepage5.spec.js homepage10.spec.js	1699мс 3921мс
4	products5.spec.js products10.spec.js	1871мс 3601мс
5	cpuUsage.spec.js	Успіх. Завантаженість менше 70% від заданих ресурсів ЦП

6	memoryUsage.spec.js	Успіх. Завантаженість менше 70% від заданих ресурсів ОЗУ
7	crushTest.spec.js	Користувачів: 100 Користувачів до поломки: 86 Середній час відповіді фронтенду: 4.9с Середній час відповіді бекенду: 4.3с

3.5. Тестування масштабованості високонавантаженого торгового інформаційного порталу

Для того, щоб провести тестування масштабованості торгового інформаційного порталу, необхідно використати механізм віртуалізації на початковій машині. Для реалізації механізму віртуалізації була обрана програма «Vmware Workstation 17 Player» зі встановленою на неї Windows 10. Для того, щоб система не витрачала ресурси на зайві процеси, на неї були встановлені лише необхідні програми та утиліти для запуску торгового порталу, а саме:

- Microsoft Visual Studio Code.
- Git
- Node.js

Щоб розуміти які ресурси будуть виділятись при тестуванні масштабованості необхідно описати параметри початкової машини:

- Процесор – intel core i5-9600k, 3,7 ГГц, 4 ядра.
- 16 ГБ оперативної пам’яті DDR4, 4000 МГц.

Враховуючи характеристики початкової машини, було прийнято рішення проводити тестування масштабованості у наступному відсотковому відношенні:

1. 100% від початкових параметрів.
2. 80% від початкових параметрів.
3. 60% від початкових параметрів.
4. 40% від початкових параметрів.
5. 20% від початкових параметрів.

3.5.1. Сто відсотків від початкових параметрів

Для того, щоб провести тестування масштабованості торгового інформаційного порталу необхідно виставити параметри в налаштуваннях віртуальної машини (рис. 3.44).

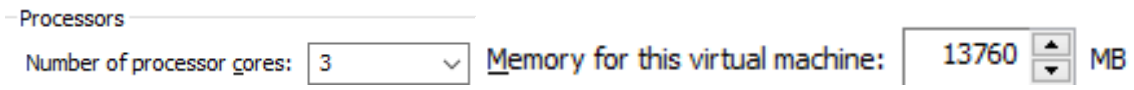

Рисунок 3.44 – Параметри віртуальної машини

В результаті тестування були отримані наступні результати (табл. 3.2):

Таблиця 3.2 – Результати тестування масштабованості із залученням 100% ресурсів.

№	Назва тесту	Результат
1	homepage1.spec.js	1632мс
2	products1.spec.js	1526мс
3	homepage5.spec.js homepage10.spec.js	2026мс 4038мс
4	products5.spec.js products10.spec.js	2561мс 3308мс

5	cpuUsage.spec.js	Успіх. Завантаженість менше 70% від заданих ресурсів ЦП
6	memoryUsage.spec.js	Успіх. Завантаженість менше 70% від заданих ресурсів ОЗУ
7	crushTest.spec.js	Користувачів: 100 Користувачів до поломки: 82 Середній час відповіді фронтенду: 5.2с Середній час відповіді бекенду: 4.1с

Після проведеного тестування масштабованості були отримані, в цілому задовільні результати роботи торгового інформаційного порталу.

3.5.2. Вісімдесят відсотків від початкових параметрів

Для того, щоб провести тестування масштабованості торгового інформаційного порталу необхідно виставити параметри в налаштуваннях віртуальної машини (рис. 3.45).

Рисунок 3.45 – Параметри віртуальної машини

В результаті тестування були отримані наступні результати (табл. 3.3):

Таблиця 3.3 – Результати тестування масштабованості із залученням 80% ресурсів.

№	Назва тесту	Результат
1	homepage1.spec.js	1789мс
2	products1.spec.js	1693мс
3	homepage5.spec.js homepage10.spec.js	2313мс 4211мс

4	products5.spec.js products10.spec.js	2312мс 2997мс
5	cpuUsage.spec.js	Успіх. Завантаженість менше 70% від заданих ресурсів ЦП
6	memoryUsage.spec.js	Успіх. Завантаженість менше 70% від заданих ресурсів ОЗУ
7	crushTest.spec.js	Користувачів: 100 Користувачів до поломки: 79 Середній час відповіді фронтенду: 4.7с Середній час відповіді бекенду: 4.9с

Після проведеного тестування масштабованості були отримані, в цілому задовільні результати роботи торгового інформаційного порталу, зменшення ресурсів на 20% не сильно змінило ситуацію.

3.5.3. Шістдесят відсотків від початкових параметрів

Для того, щоб провести тестування масштабованості торгового інформаційного порталу необхідно виставити параметри в налаштуваннях віртуальної машини (рис. 3.46).

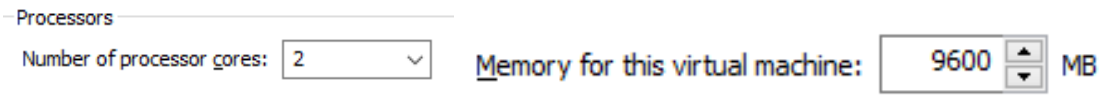

Рисунок 3.46 – Параметри віртуальної машини

В результаті тестування були отримані наступні результати (табл. 3.4):

Таблиця 3.4 – Результати тестування масштабованості із залученням 60% ресурсів.

№	Назва тесту	Результат

1	homepage1.spec.js	2829мс
2	products1.spec.js	3629мс
3	homepage5.spec.js homepage10.spec.js	3821мс 5112мс
4	products5.spec.js products10.spec.js	4381мс 5006мс
5	cpuUsage.spec.js	Успіх. Завантаженість менше 70% від заданих ресурсів ЦП
6	memoryUsage.spec.js	Успіх. Завантаженість менше 70% від заданих ресурсів ОЗУ
7	crushTest.spec.js	Користувачів: 100 Користувачів до поломки: 51 Середній час відповіді фронтенду: 6.1с Середній час відповіді бекенду: 7.6с

Після проведеного тестування масштабованості результати погіршились, система все ще демонструє непогану стійкість до збоїв, стійкість до навантаження процесора та оперативної пам’яті, але в цілому час обробки запитів виріс як на фронтенді так і на бекенді.

3.5.4. Сорок відсотків від початкових параметрів

Для того, щоб провести тестування масштабованості торгового інформаційного порталу необхідно виставити параметри в налаштуваннях віртуальної машини (рис. 3.47).

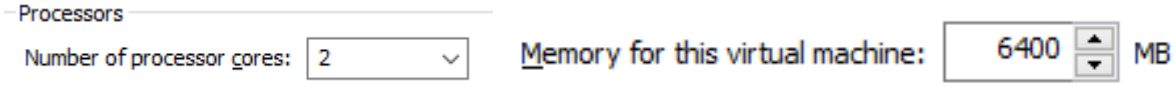

Рисунок 3.47 – Параметри віртуальної машини

В результаті тестування були отримані наступні результати (табл. 3.5):

Таблиця 3.5 – Результати тестування масштабованості із залученням 40% ресурсів.

№	Назва тесту	Результат
1	homepage1.spec.js	3112мс
2	products1.spec.js	3412мс
3	homepage5.spec.js homepage10.spec.js	3911мс 5021мс
4	products5.spec.js products10.spec.js	4491мс 4812мс
5	cpuUsage.spec.js	Успіх. Завантаженість менше 70% від заданих ресурсів ЦП
6	memoryUsage.spec.js	Успіх. Завантаженість менше 70% від заданих ресурсів ОЗУ
7	crushTest.spec.js	Користувачів: 100 Користувачів до поломки: 31 Середній час відповіді фронтенду: 6.8с Середній час відповіді бекенду: 8.1с

Після проведеного тестування масштабованості відбулося погіршення результатів, але воно було не настільки значущим як при тестуванні 60% від початкових параметрів. В основному результати погіршились в краш-тесті, де кількість користувачів до поломки впала до 31 та середній час відповіді бекенду ще збільшився відносно фронтенду.

3.5.5. Двадцять відсотків від початкових параметрів

Для того, щоб провести тестування масштабованості торгового інформаційного порталу необхідно виставити параметри в налаштуваннях віртуальної машини (рис. 3.48).

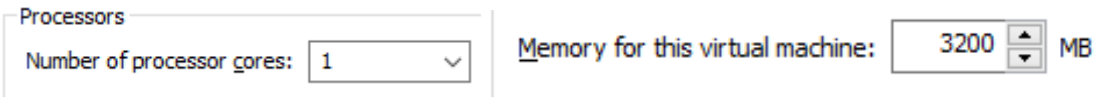

Рисунок 3.47 – Параметри віртуальної машини

В результаті тестування були отримані наступні результати (табл. 3.6):

Таблиця 3.6 – Результати тестування масштабованості із залученням 20% ресурсів.

№	Назва тесту	Результат
1	homepage1.spec.js	6117мс
2	products1.spec.js	7564мс
3	homepage5.spec.js homepage10.spec.js	7268мс 8621мс
4	products5.spec.js products10.spec.js	7218мс 9617мс
5	cpuUsage.spec.js	Провал. Завантаженість більше 70% від заданих ресурсів ЦП
6	memoryUsage.spec.js	Провал. Завантаженість більше 70% від заданих ресурсів ОЗУ
7	crushTest.spec.js	Користувачів: 100 Користувачів до поломки: 12 Середній час відповіді фронтенду: 12.8с Середній час відповіді бекенду: 19.9с

Після проведеного тестування масштабованості відбулося значне погіршення результатів. Виходячи з цього, можна зробити висновок, що торгова платформа неспроможна на роботу в таких умовах.

За цими результатами були побудовані графіки результатів тестування (рис. 3.44, рис. 3.45).

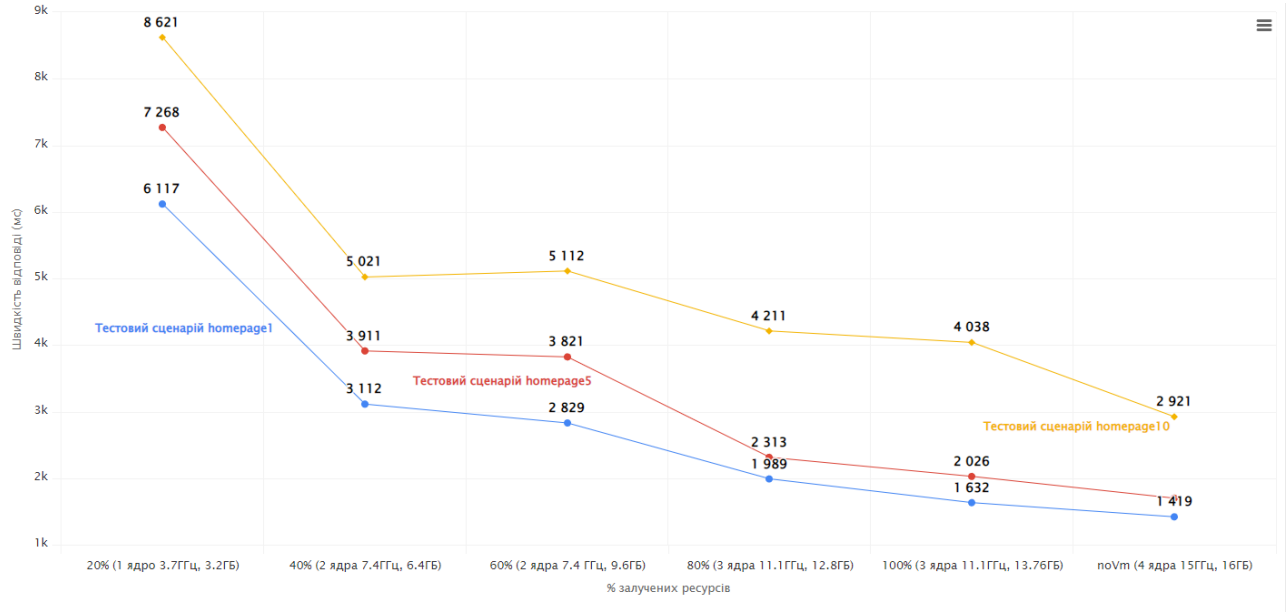

Рисунок 3.44 – Графік результатів тестування сценарію для головної сторінки

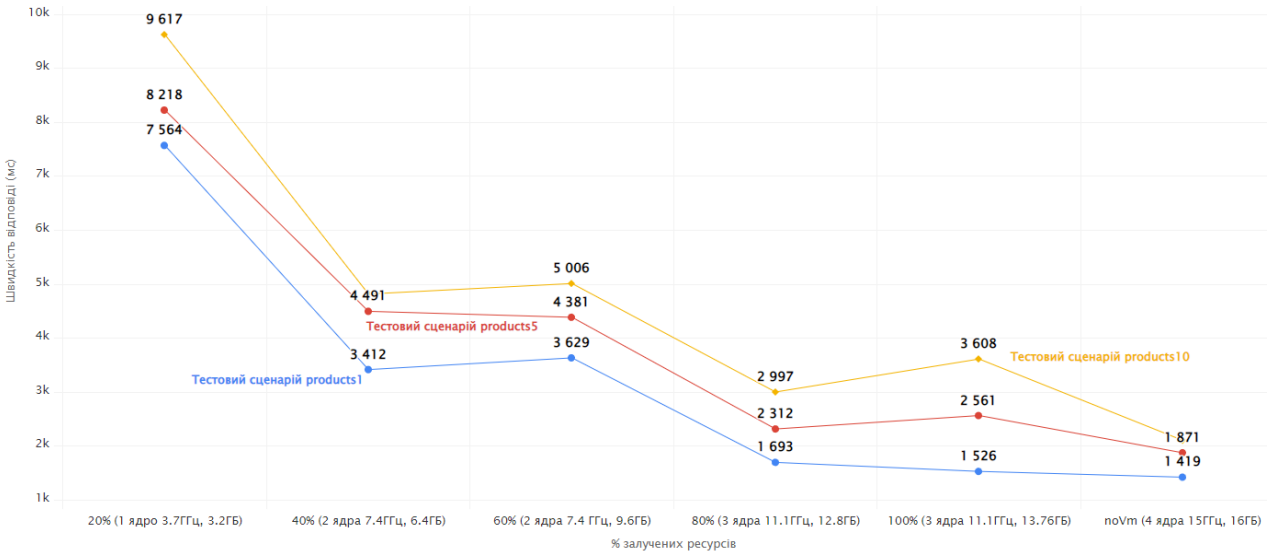

Рисунок 3.45 – Графік результатів тестування сценарію для продуктової сторінки

Також окремо була побудована гістограма результатів виконання сценарію crushTest.spec.js за користувачами до поломки (рис. 3.46).

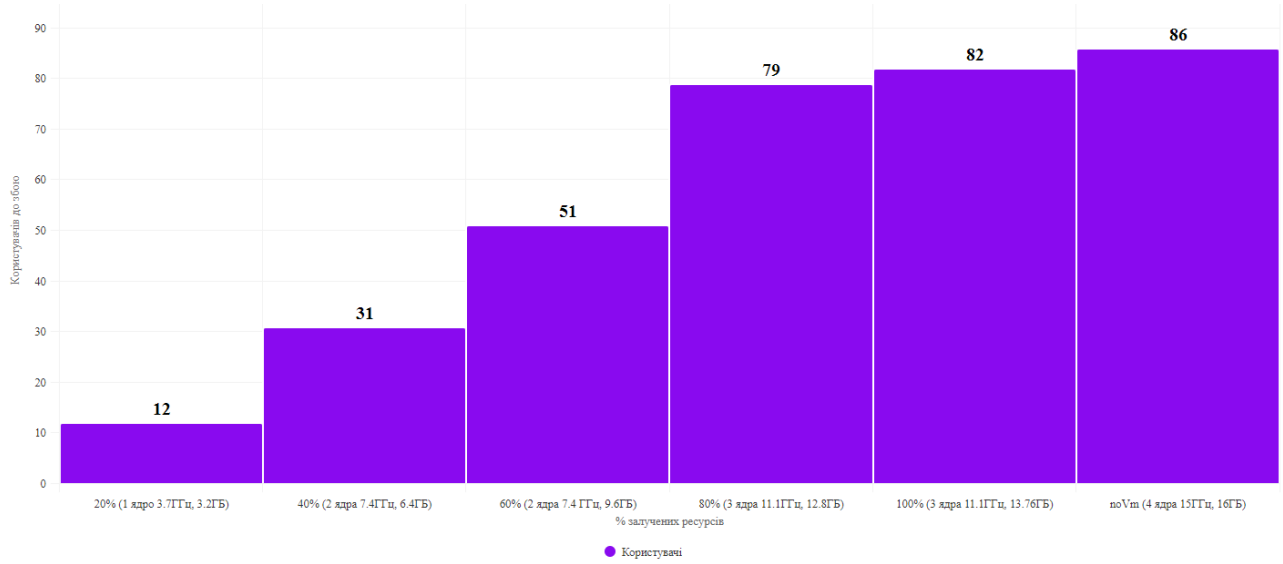

Рисунок 3.46 – Гістограма результатів виконання сценарію crushTest.spec.js за користувачами до поломки

Також окремо була побудована гістограма результатів виконання сценарію crushTest.spec.js за середнім часом відповіді фронтенду та бекенду (рис. 3.47).

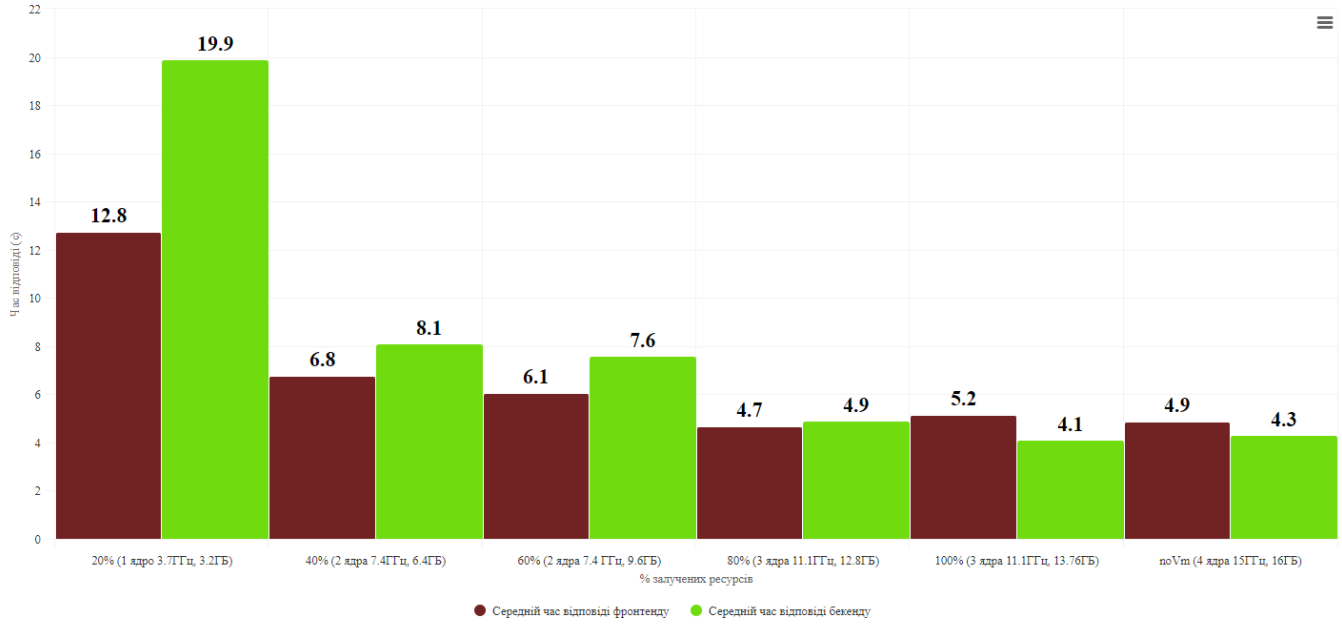

Рисунок 3.47 – Гістограма результатів виконання сценарію crushTest.spec.js за середнім часом відповіді фронтенду та бекенду

3.6. Інструкційний матеріал з розгортання високонавантаженого торгового інформаційного порталу

Користувацький інтерфейс платформи був побудований таким чином, щоб не було необхідності користуватись додатковими інструкційними вказівками. Єдине, що потребує інструктажу – це запуск програмного продукту. Оскільки застосунок складається з двох частин: фронтенду і бекенду, їх необхідно запускати окремо.

Обов'язково першим треба запустити бекенд застосунку командою «npm run server» (рис. 3.48).

```
$ npm run server

> fe-final-backend@1.0.0 server
> nodemon server.js

[nodemon] 1.19.4
[nodemon] to restart at any time, enter `rs`
[nodemon] watching dir(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node server.js`
Server running on port 5000
(node:25516) [MONGODB DRIVER] Warning: Current Server Discovery and Monitoring engine is deprecated
{ useUnifiedTopology: true } to the MongoClient constructor.
(Use `node --trace-warnings ...` to show where the warning was created)
(node:25516) DeprecationWarning: collection.ensureIndex is deprecated. Use createIndexes instead.
MongoDB Connected
```

Рисунок 3.48 – Запуск бекенду платформи

Після отримання повідомлення про підключення до бази даних можна запускати фронтенд командою «npm start» (рис. 3.49).

```
Admin@DESKTOP-3QF4010 MINGW64 ~/Desktop/final-project-shop/client (main)
$ npm start
```

Рисунок 3.49 – Запуск фронтенду платформи

В результаті буде відкрито вікно браузера за замовчуванням з інтерфейсом торгової інформаційної платформи.

3.7. Висновки до третього розділу

В результаті виконаної роботи була розроблений сучасний масштабований високонавантажений торговий інформаційний портал. Був спроектований та побудований деталізований та інтуїтивно зрозумілий користувацький інтерфейс. На основі інтерфейсу відбулась побудова фронтенду та бекенду порталу, була детально розглянута їх архітектура, описаний функціонал кожної з модульних категорій. Також було проведене ручне та автоматизоване тестування для виявлення недоліків та подальшому удосконаленню системи.

ВИСНОВКИ

В результаті виконаної дипломної роботи було виконано глибокий аналіз предметної області з метою розробки високонавантаженого торгового порталу. Під час аналізу процесу, були проаналізовані наявні торгово-інформаційні портали, висвітлено питання впливу швидкодії застосунку на користувацький досвід, також була поставлена задача на розробку додатка і формування до нього функціональних та нефункціональних вимог.

Під час проектування архітектури порталу було висвітлено питання взаємодії основних елементів порталу між собою, спроектована та описана база даних, а також сформовано базове розуміння функціонування додатку.

Під час практичної реалізації було спроектовано та розроблено інтуїтивно зрозумілий користувацький інтерфейс. Детально описаний функціонал модульних категорій фронтенду та бекенду, який дає розуміння масштабності всієї системи. Також було проведено масштабне ручне та автоматизоване тестування додатка з метою демонстрації роботи системи в умовах високої завантаженості.

В результаті роботи був отриманий працюючий масштабований клієнт-серверний застосунок зі стійкістю до збоїв, з інтуїтивно зрозумілим інтерфейсом та спроможністю працювати в екстремальних умовах високої завантаженості системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Keith Yorkston. Performance Testing: An ISTQB Certified Tester Foundation Level Specialist Certification Review / Apress, 2021. — 394 c. — ISBN 1484272544, 9781484272541.

2. Mark Myers. A Smarter Way to Learn Javascript. / ASmarterWayToLearn.com, 2014. — 293 c. — ISBN 1497408180, 9781497408180.

3. Marijn Haverbeke. Eloquent JavaScript. / No Starch Press, 2018. — 472 c. — ISBN 1593279507, 9781593279509.

4. Robert C. Martin. Clean Code: A Handbook of Agile Software Craftsmanship. / Pearson Education, 2008. — 464 c. — ISBN 0136083250, 9780136083252.

5. Boris Beizer. Software Testing Techniques. / International Thomson Computer Press, 1990. — 550 c. — ISBN 1850328803, 9781850328803.

6. Alex Petrov. Database Internals: A Deep Dive into How Distributed Data Systems Work. / O'Reilly Media, Inc., 2019 – 376 c. – ISBN 1492040304, 9781492040309.

7. Rod Stephens. Beginning Database Design Solutions. / Wiley, 2009 – 400 c. – ISBN 0470440511, 9780470440513.

8. Laine Campbell, Charity Majors. Database Reliability Engineering: Designing and Operating Resilient Database Systems. / O'Reilly Media, Inc., 2017 – 294 c. – ISBN 1491926228, 9781491926222.

9. Amit Phaltankar, Juned Ahsan, Michael Harrison, Liviu Nedov. MongoDB Fundamentals: A hands-on guide to using MongoDB and Atlas in the real world. / Packt Publishing Ltd, 2020 – 748 c – ISBN 1839213043, 9781839213045.

10. Martin Kleppmann. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. / O'Reilly Media, Inc., 2017 – 616 с – ISBN 1491903112, 9781491903117.

11. Adrian Mouat. Using Docker: Developing and Deploying Software with Containers. / O'Reilly Media, Inc., 2015 – 354с – ISBN 1491915919, 9781491915912.

12. Brandon Shaw. Docker Step-by-Step: The Ultimate Guide From Beginner to Expert. Learn & Master The Platform and Containerize, Create, Deploy and Run Your Application Like a Professional. / Independently Published, 2019 – 122 с – ISBN 1672361796, 9781672361798.

13. Deepak Vohra. Kubernetes Microservices with Docker. / Apress, 2016 – 432 с – ISBN 1484219074, 9781484219072.

14. Rajdeep Dua, Vaibhav Kohli, Santosh Kumar Konduri. Learning Docker Networking. / Packt Publishing, 2016 – 176 с – ISBN 1785280953, 9781785280955.

15. Martin Kleppmann. Designing Data-Intensive Applications. / O'Reilly Media, Inc., 2017 – 562 с – ISBN 9781449373320.

16. HubSpot. 19 Website Speed Optimization Strategies for 2022. [Електронний ресурс]. – Режим доступу : https://blog.hubspot.com/website/how-to-optimize-website-speed

17. KeyCdn. 18 Tips for Website Performance Optimization [Електронний ресурс]. – Режим доступу : https://www.keycdn.com/blog/website-performance-optimization

18. Architizer. An Architect’s Guide to Building Codes: 7 Steps to a Safer Design [Електронний ресурс]. – Режим доступу : https://architizer.com/blog/practice/details/young-architect-guide-building-codes/

19. Builtin. 17 High-Performance Computing Applications and Examples [Електронний ресурс]. – Режим доступу : https://builtin.com/hardware/high-performance-computing-applications

20. Geeksforgeeks. Types of Software Architecture Patterns [Електронний ресурс]. – Режим доступу : https://www.geeksforgeeks.org/types-of-software-architecture-patterns/

21. Turing. Software Architecture Patterns: What Are the Types and Which Is the Best One for Your Project [Електронний ресурс]. – Режим доступу : https://www.turing.com/blog/software-architecture-patterns-types/

22. Guru99. Performance Testing Tutorial – Types (Example) [Електронний ресурс]. – Режим доступу : https://www.guru99.com/performance-testing.html

23. Stackify. The Ultimate Guide to Performance Testing and Software Testing: Testing Types, Performance Testing Steps, Best Practices, and More [Електронний ресурс]. – Режим доступу : https://stackify.com/ultimate-guide-performance-testing-and-software-testing/

24. Stackify. Docker Tutorial: Get Going From Scratch [Електронний ресурс]. – Режим доступу : https://stackify.com/docker-tutorial/

25. BrowserStack. JavaScript Unit Testing Tutorial [Електронний ресурс]. – Режим доступу : https://www.browserstack.com/guide/unit-testing-in-javascript

ДОДАТКИ

Додаток А. Лістинги програмних модулів тестування системи

cpuPerfomanceUsage.spec.js

```
const { PerformanceObserver, performance } = require('perf_hooks');

test('CPU usage should be less than 70% for 10 seconds', async () => {
	// Define the duration of the test in milliseconds
	const duration = 10000;

	// Define the threshold for CPU usage as a percentage
	const threshold = 70;

	// Create a new PerformanceObserver to monitor CPU usage
	// eslint-disable-next-line no-shadow
	const observer = new PerformanceObserver((list, observer) => {
		const entry = list.getEntries()[0];
		const cpuUsage = entry?.name === 'cpu' ? entry?.value : null;

		// If the CPU usage exceeds the threshold, fail the test
		if (cpuUsage && cpuUsage > threshold) {
			observer.disconnect();
			throw new Error(
				`CPU usage exceeded threshold of ${threshold}%: ${cpuUsage}%`
			);
		}
	});

	// Start monitoring CPU usage
	observer.observe({ entryTypes: ['cpu'], buffered: true });

	// Perform some CPU-intensive operation for the duration of the test
	const start = performance.now();
	while (performance.now() - start < duration) {
		…
	}

	// Stop monitoring CPU usage
	observer.disconnect();
});
```

homepage1.spec.js

```
// eslint-disable-next-line import/no-extraneous-dependencies
const puppeteer = require('puppeteer');

test('loading the homepage should take less than 3 seconds', async () => {
	// Launch a new browser instance
	const browser = await puppeteer.launch();

	// Open a new page
	const page = await browser.newPage();

	// Measure the time it takes to load the page
	const start = Date.now();
	await page.goto('http://localhost:3000/');
	const end = Date.now();
	const duration = end - start;
```

```
	// Assert that the duration is less than 3 seconds
	expect(duration).toBeLessThan(3000);

	// Close the browser
	await browser.close();
}, 30000);
```

homepage5.spec.js

```
// eslint-disable-next-line import/no-extraneous-dependencies
const puppeteer = require('puppeteer');

test('loading the homepage for 5 people should take less than 15 seconds', async () => {
	// Launch a new browser instance
	const browser = await puppeteer.launch();

	// Define the number of users to simulate
	const numUsers = 5;

	// Define an array to store the page load durations
	const durations = [];

	// Create a new page for each user
	const pages = await Promise.all(
		Array.from({ length: numUsers }, () => browser.newPage())
	);

	// Define a function to measure the time it takes to load a page
	const measurePageLoadTime = async (page) => {
		const start = Date.now();
		await page.goto('http://localhost:3000/');
		const end = Date.now();
		const duration = end - start;
		durations.push(duration);
	};

	// Simulate concurrent page loads for each user
	await Promise.all(pages.map((page) => measurePageLoadTime(page)));

	// Assert that the total duration is less than 10 seconds
	const totalDuration = durations.reduce((acc, duration) => acc + duration, 0);
	expect(totalDuration).toBeLessThan(16000);

	// Close the browser
	await browser.close();
}, 20000);
```

homepage10.spec.js

```
// eslint-disable-next-line import/no-extraneous-dependencies
const puppeteer = require('puppeteer');

test('loading the homepage for 10 people should take less than 30 seconds', async () => {
	// Launch a new browser instance
	const browser = await puppeteer.launch();

	// Define the number of users to simulate
	const numUsers = 10;
```

```
	// Define an array to store the page load durations
	const durations = [];

	// Create a new page for each user
	const pages = await Promise.all(
		Array.from({ length: numUsers }, () => browser.newPage())
	);

	// Define a function to measure the time it takes to load a page
	const measurePageLoadTime = async (page) => {
		const start = Date.now();
		await page.goto('http://localhost:3000/');
		const end = Date.now();
		const duration = end - start;
		durations.push(duration);
	};

	// Simulate concurrent page loads for each user
	await Promise.all(pages.map((page) => measurePageLoadTime(page)));

	// Assert that the total duration is less than 10 seconds
	const totalDuration = durations.reduce((acc, duration) => acc + duration, 0);
	expect(totalDuration).toBeLessThan(37000);

	// Close the browser
	await browser.close();
}, 100000);
```

crushTest.spec.js

```
const puppeteer = require('puppeteer');
test('system crush test', async () => {
	let users = 0;
	const threshold = 100;
	while (users < threshold) {
		// eslint-disable-next-line no-await-in-loop
		const browser = await puppeteer.launch();
		// eslint-disable-next-line no-await-in-loop
		const page = await browser.newPage();
		const user = new User(browser, page, users);
		user.simulateActivity();
		users += 1;
	}
});
```

products1.spec.js

```
// eslint-disable-next-line import/no-extraneous-dependencies
const puppeteer = require('puppeteer');

test('loading the products page should take less than 3 seconds', async () => {
	// Launch a new browser instance
	const browser = await puppeteer.launch();

	// Open a new page
	const page = await browser.newPage();

	// Measure the time it takes to load the page
	const start = Date.now();
```

```
        await page.goto('http://localhost:3000/products?categories=rolls');
        const end = Date.now();
        const duration = end - start;

        // Assert that the duration is less than 3 seconds
        expect(duration).toBeLessThan(3000);

        // Close the browser
        await browser.close();
}, 30000);
```

products5.spec.js

```
// eslint-disable-next-line import/no-extraneous-dependencies
const puppeteer = require('puppeteer');

test('loading the products page for 5 people should take less than 15 seconds', async () => {
        // Launch a new browser instance
        const browser = await puppeteer.launch();

        // Define the number of users to simulate
        const numUsers = 5;

        // Define an array to store the page load durations
        const durations = [];

        // Create a new page for each user
        const pages = await Promise.all(
                Array.from({ length: numUsers }, () => browser.newPage())
        );

        // Define a function to measure the time it takes to load a page
        const measurePageLoadTime = async (page) => {
                const start = Date.now();
                await page.goto('http://localhost:3000/products?categories=rolls');
                const end = Date.now();
                const duration = end - start;
                durations.push(duration);
        };

        // Simulate concurrent page loads for each user
        await Promise.all(pages.map((page) => measurePageLoadTime(page)));

        // Assert that the total duration is less than 10 seconds
        const totalDuration = durations.reduce((acc, duration) => acc + duration, 0);
        expect(totalDuration).toBeLessThan(15000);

        // Close the browser
        await browser.close();
}, 20000);
```

products10.spec.js

```
// eslint-disable-next-line import/no-extraneous-dependencies
```

```
const puppeteer = require('puppeteer');

test('loading the products page for 10 people should take less than 30 seconds', async () => {
	// Launch a new browser instance
	const browser = await puppeteer.launch();

	// Define the number of users to simulate
	const numUsers = 10;

	// Define an array to store the page load durations
	const durations = [];

	// Create a new page for each user
	const pages = await Promise.all(
		Array.from({ length: numUsers }, () => browser.newPage())
	);

	// Define a function to measure the time it takes to load a page
	const measurePageLoadTime = async (page) => {
		const start = Date.now();
		await page.goto('http://localhost:3000/products?categories=rolls');
		const end = Date.now();
		const duration = end - start;
		durations.push(duration);
	};

	// Simulate concurrent page loads for each user
	await Promise.all(pages.map((page) => measurePageLoadTime(page)));

	// Assert that the total duration is less than 10 seconds
	const totalDuration = durations.reduce((acc, duration) => acc + duration, 0);
	expect(totalDuration).toBeLessThan(30000);

	// Close the browser
	await browser.close();
}, 100000);
```

memoryUsage.spec.js

```
const { PerformanceObserver } = require('perf_hooks');

test('memory usage should be less than 50MB for 10 seconds', async () => {
	// Define the duration of the test in milliseconds
	const duration = 10000;

	// Define the threshold for memory usage in bytes
	const threshold = 50 * 1024 * 1024; // 50MB

	// Create a new PerformanceObserver to monitor memory usage
	// eslint-disable-next-line no-shadow
	const observer = new PerformanceObserver((list, observer) => {
		const entry = list.getEntries()[0];
		const memoryUsage = entry?.name === 'gc' ? entry?.delta : null;

		// If the memory usage exceeds the threshold, fail the test
		if (memoryUsage && memoryUsage > threshold) {
			observer.disconnect();
```

```
                                throw new Error(
                                        `Memory usage exceeded threshold of ${threshold} bytes: ${memoryUsage} bytes`
                                );
                        }
                });

                // Start monitoring memory usage
                observer.observe({ entryTypes: ['gc'], buffered: true });

                // Allocate some memory for the duration of the test
                const buffers = Array.from({ length: duration / 1000 }, () =>
                        Buffer.alloc(1024 * 1024)
                ); // Allocate 1MB buffer for each second
                const start = performance.now();
                while (performance.now() - start < duration) {
                        // Do some CPU-intensive operation here
                }
                // Free the memory
                buffers.forEach((buffer) => buffer.fill(0));

                // Trigger garbage collection
                // global.gc();

                // Stop monitoring memory usage
                observer.disconnect();
        });
```

Додаток Б. Лістинг програмних модулів системи

Програмний код проекту викладений на публічний доступ в GitHub за посиланням: https://github.com/AntonSadovnik/diploma.git

Main.jsx

```
import React from 'react';
import Grid from '@mui/material/Grid';
import NoveltieSlider from '../../components/Slider/components/noveltieSlider/Slider';
import GridItem from '../../components/Grid/Grid';
import SimpleAccordion from '../../components/AboutCompany/AboutCompany';
import Socials from '../../components/Footer/components/socials/Socials';

import OfferSliderSkeleton from '../../components/Skeleton/Skeleton';

function Main() {
 return (
  <main>
   <Grid
    container
    justifyContent="space-around"
    flexDirection="column"
    backgroundColor={(theme) => theme.palette.lightGrayColor.main}
    sx={{ padding: '0 10px' }}
   >
    <Grid
     sx={{
      padding: { xs: '30px 5px 90px', sm: '30px 5px 30px' },
      width: '100%',
     }}
    >
     <Grid sx={{ display: { md: 'block', xs: 'none' } }}>
      <OfferSliderSkeleton />
     </Grid>
     <Grid container justifyContent="center">
      <GridItem />
     </Grid>
     <Grid
      container
      justifyContent="center"
      sx={{ display: { xs: 'none', sm: 'flex' }, marginTop: '50px' }}
     >
      <NoveltieSlider />
     </Grid>
     <Grid
      container
      justifyContent="center"
      sx={{ marginTop: { sm: '50px', xs: '20px' } }}
     >
      <Grid item sm={10}>
       <SimpleAccordion />
      </Grid>
     </Grid>
     <Grid
      container
      textAlign="center"
```

```
          sx={{ display: { xs: 'flex', sm: 'none' }, marginTop: '20px' }}
        >
          <Socials />
        </Grid>
      </Grid>
    </Grid>
  </main>
 );
}
export default Main;
```

About.jsx

```
import React from 'react';
import { Box, Typography } from '@mui/material';
import { useSelector } from 'react-redux';

function About() {
const city = useSelector((store) => store.city);
return (
        <Box
                sx={{
                        padding: { xs: '20px', sm: '50px' },
                        textAlign: 'justify',
                }}
        >
                <Typography
                        variant="h6"
                        component="h2"
                        fontSize="24px"
                        sx={{ marginBottom: '9px' }}
                >
                        The best sushi delivery in {city} for everyone
                </Typography>
                <Typography
                        color="text.secondary"
                        variant="h6"
                        component="h2"
                        fontSize="18px"
                        sx={{ marginBottom: '9px' }}
                >
                        When you think of sushi, you probably have an image of a vacation in the
                        Far East or a scene from a movie where the main character is sitting in
                        a restaurant, and Asian dishes literally pour like water on a client
                        sitting at a high table. But you don&amp;apos;t have to wait for a trip to
                        the Far East, you can enjoy the taste of great sushi right now!
                        It&amp;apos;s time to try a treat that is usually associated with Japan,
                        although the Chinese were the first to cook it. You can order sushi
                        in {city} with delivery and enjoy it right at home. Surprise your friends who
                        came to visit you or arrange a romantic evening with your soulmate. The
                        combination of boiled rice and raw fish, delivered to your chosen
                        address, can also be a decoration for a family celebration or a business
                        lunch.
                </Typography>
                <Typography
                        variant="h6"
                        component="h2"
                        fontSize="24px"
                        sx={{ marginBottom: '9px' }}
                >
```

```
                    Sushi delivery in {city}
                </Typography>
                <Typography
                    variant="h6"
                    component="h2"
                    color="text.secondary"
                    fontSize="18px"
                    sx={{ marginBottom: '9px' }}
                >
                    Sushi was introduced to the West in the early 1900s by Japanese
                    immigration after the Meiji Restoration. However, it was not popular
                    among the population outside of the upper class, and as Japanese
                    immigration declined in the late 1900s, it became much less common.
                    Sushi re-emerged a few years after the end of World War II, when Japan
                    re-opened to international trade, tourism and business.
                    <br />
                    To help Europeans get used to the idea of eating sushi, many restaurants
                    have begun experimenting with new flavor combinations and sushi rolls.
                    One of the most popular rolls was the now ubiquitous California roll,
                    which is an inside-out Makizushi roll with cucumber, crab meat (or
                    imitation crab meat), and avocado topped with white rice.
                    <br />
                    Despite the fact that sushi is considered one of the haute oriental
                    dishes, sushi lovers do not need to plan a trip to Japan, because in the
                    modern world you can simply order a portion of Japanese seafood right in
{city}.
                    Sushi in {city} is incredibly popular and is in first place in terms
                    of the number of orders for delivery. In {city} there is no lack of
opportunities,
                    but very often there is a lack of time. If you are hungry and urgently
                    want to treat yourself to something tasty, then sushi delivery in {city} is
                    exactly what you need. Delicious and hearty sushi will quickly
                    energize you. You can also order sushi not only home but also directly
                    to the office. This gourmet Asian dish is an interesting alternative to
                    typical catering,
                    <br />
                    &amp;quot;Sushi and Noodles&amp;quot; is the largest delivery network for the
                    most delicious food in Ukraine. We have already gained trust and
                    popularity among customers in the largest cities, namely Kharkiv,
                    Odesa, Dnipro and Kyiv. “Sushi and Noodles” offers not just sushi
                    delivery in {city}, we offer delicious and healthy dishes at a bargain
                    price.
                </Typography>
            </Box>
        );
        }
        export default About;
```

Ordering.jsx

```
        import React from 'react';
        import { useDispatch } from 'react-redux';
        import { Stack } from '@mui/material';
        import ClientDataForm from '../../components/ClientDataForm/ClientDataForm';
        import { OrderMessage } from '../../components/OrderMessage/OrderMessage';
        import { placeOrder, removeOrderedProducts } from '../../api/placeOrders';
        import { resetCart } from '../../store/actions';

        function Ordering() {
        const [openOrderModal, setOpenModal] = React.useState(false);
        const [orderProcessing, setOrderProcessing] = React.useState(true);
```

```
const [error, setError] = React.useState(false);
const [success, setSuccess] = React.useState(false);
const dispatch = useDispatch();

const handleClose = () => setOpenModal(false);

const handleOrder = (userData, cartProducts, resetForm, customerId) => {
        setOrderProcessing(true);
        setSuccess(false);
        setError(false);

        let newOrder = {};

        if (customerId) {
                newOrder = { customerId, ...userData };
        } else {
                newOrder = { products: cartProducts, ...userData };
        }

        placeOrder(newOrder)
                .then(() => {
                        setOrderProcessing(false);
                        setSuccess(true);
                        setError(false);
                        localStorage.removeItem('cart');
                        dispatch(resetCart());
                        resetForm();
                        if (customerId) {
                                removeOrderedProducts(localStorage.getItem('token'));
                        }
                })
                .catch(() => {
                        setOrderProcessing(false);
                        setError(true);
                        setSuccess(false);
                });
};

return (
        <main>
                <Stack alignItems="center">
                        <ClientDataForm handleOrder={handleOrder}
setOpenModal={setOpenModal} />
                        <OrderMessage
                                open={openOrderModal}
                                handleClose={handleClose}
                                orderProcessing={orderProcessing}
                                success={success}
                                error={error}
                        />
                </Stack>
        </main>
);
}
export default Ordering;
```

Products.jsx

```
import React, { useEffect, useState } from 'react';
import { Grid } from '@mui/material';
import { useSearchParams, useNavigate } from 'react-router-dom';
```

```
import { useDispatch, useSelector } from 'react-redux';
import { addToCart, getProductsAction } from '../../store/actions';
import ProductCard from '../../components/ProductListing/Card/Card';
import SimpleAccordion from '../../components/AboutCompany/AboutCompany';
import Socials from '../../components/Footer/components/socials/Socials';
import SortSelect from '../../components/ProductListing/SortSelect/SortSelect';
import Filter from '../../components/Filter/Filter';
import Title from '../../components/ProductListing/Title/Title';
import NoResults from '../../components/ProductListing/NoResults/NoResults';
import ProductsPagination from
'../../components/ProductListing/ProductsPagination/ProductsPagination';
import Loader from '../../components/Loader/Loader';

function Products() {
const getQuery = (s) => s.includes('?') && s.substr(s.lastIndexOf('?') + 1);
const navigate = useNavigate();
const perPageProducts = 6;
const dispatch = useDispatch();
const [searchParams] = useSearchParams({});
const [loader, setLoader] = useState(false);
const { products, productsQuantity } = useSelector((state) => state.products);

useEffect(() => {
    dispatch(
        getProductsAction(
            `perPage=${perPageProducts}&${getQuery(window.location.href)}`,
            navigate,
            setLoader
        )
    );
}, [searchParams]);

const components = products.map((product) => (
    <ProductCard
        key={product.itemNo}
        data={product}
        onClick={() => dispatch(addToCart(product))}
    />
));
return (
    <main>
        <Grid
            sx={{
                padding: {
                    xs: '15px 15px 90px',
                    sm: '15px 15px 30px',
                    lg: '30px 111px 60px',
                },
                backgroundColor: '#F2F2F2',
            }}
        >
            <Grid
                container
                marginBottom={3.75}
                justifyContent={{ xs: 'center', lg: 'space-between' }}
                flexDirection={{ xs: 'column', lg: 'row' }}
                alignItems="center"
            >
                <Grid
                    item
```

```
                                        container
                                        width="fit-content"
                                        alignItems="center"
                                        marginBottom={{ xs: 2, lg: 0 }}
                                    >
                                        <Title />
                                    </Grid>
                                    <Grid>
                                        <Filter />
                                    </Grid>
                                    <Grid width={{ xs: '100%', sm: '50%', lg: 'fit-content' }}>
                                        <SortSelect />
                                    </Grid>
                                </Grid>

                                {loader ? (
                                    <Loader />
                                ) : (
                                    <>
                                        <Grid container rowGap={{ xs: 1.25, sm: 6.25 }}>
                                            {components.length ? components : <NoResults
/>}
                                        </Grid>
                                        {components.length ? (
                                            <Grid>
                                                <ProductsPagination

        productsQuantity={productsQuantity}

        perPageProducts={perPageProducts}
                                                />
                                            </Grid>
                                        ) : null}
                                        <Grid margin={{ xs: '30px 0 20px', sm: '40px 0 0' }}>
                                            <SimpleAccordion />
                                        </Grid>
                                        <Grid display={{ xs: 'block', sm: 'none' }}
textAlign="center">
                                            <Socials />
                                        </Grid>
                                    </>
                                )}
                            </Grid>
                        </main>
        );
        }

        export default Products;
```