МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА ІНТЕРНЕТ-МАГАЗИНУ РОЗДРІБНОЇ ТОРГІВЛІ ЗАСОБАМИ JAVA SPRING»**

Виконав: студент 4 курсу, групи 6.1210-1пі
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
В.П. Прохович
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, доцент, к.ф.-м.н. Горбенко В.І.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент доцент кафедри комп'ютерних наук, доцент, к.т.н. Борю С.Ю.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

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

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Проховичу Владиславу Павловичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка Інтернет-магазину роздрібної торгівлі засобами Java Spring

керівник роботи Горбенко Віталій Іванович, к.ф.-м.н., доцент
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

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

2. Строк подання студентом роботи 03.06.2024 р.

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

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

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

6. Консультанти розділів роботи

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

7. Дата видачі завдання 25.12.2023 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	09.01.2024	
2.	Збір вихідних даних.	19.01.2024	
3.	Обробка методичних та теоретичних	12.02.2024	
	джерел.		
4.	Розробка першого та другого розділу.	18.03.2024	
5.	Розробка третього розділу.	20.05.2024	
6.	Оформлення та нормоконтроль	27.05.2024	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	17.06.2024	

Студент _____ В.П. Прохович
(підпис) (ініціали та прізвище)

Керівник роботи _____ В.І. Горбенко
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

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

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка Інтернет-магазину роздрібної торгівлі засобами Java Spring»: 54 с., 25 рис., 3 табл., 14 джерел, 3 додатки.

ВЕБЗАСТОСУНОК, ЕЛЕКТРОННА КОМЕРЦІЯ, ІНФОРМАЦІЙНА СИСТЕМА, КЛІЄНТ-СЕРВЕРНА АРХІТЕКТУРА, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ.

Об’єкт дослідження – використання фреймворку Java Spring до реалізації інформаційної системи інтернет-магазину.

Мета роботи: виконати аналіз функціональності та архітектури інтернет-магазину для роздрібної торгівлі, та реалізувати його з використанням технологій Java Spring.

Метод дослідження – аналіз функціональності інформаційних систем інтернет магазинів та експериментальне дослідження реалізацій функціональності засобами фреймворку Java Spring.

Розробка Інтернет-магазину є актуальною проблемою сучасного електронного бізнесу, що дозволяє розширити можливості ведення бізнесу. У кваліфікаційній роботі досліджується процес розробки та реалізація функціональності Інтернет-магазину роздрібної торгівлі засобами Java Spring.

Архітектура інформаційної системи базується на клієнт-серверній архітектурі, де клієнтська частина взаємодіє із сервером для отримання та обробки інформації. Клієнтська частина представлена вебінтерфейсом для перегляду товарів, покупок та оплати. Серверна частина обробляє запити користувачів, керує базою даних з товарами та замовленнями, а також надає інформацію клієнтам.

Зроблено висновок про придатність використання Java Spring для реалізації Інтернет-магазину роздрібної торгівлі та розроблено рекомендації щодо вдосконалення системи.

SUMMARY

Bachelor's qualifying paper "Development of an Online Retail Store Using Java Spring": 54 pages, 25 figures, 3 tables, 14 references, 3 supplements.

WEB APPLICATION, ELECTRONIC COMMERCE, INFORMATION SYSTEM, CLIENT-SERVER ARCHITECTURE, SOFTWARE.

The object of the study is the use of the Java Spring framework for the implementation of an online retail store information system.

The aim of the study is to analyze the functionality and architecture of an online retail store and implement it using Java Spring technologies.

The methods of research are the analysis of the functionality of online retail store information systems and experimental research on the implementation of functionality using the Java Spring framework.

Developing an online store is a relevant issue in the context of modern e-business. Solving this problem allows businesses to expand their operations and increase their customer base. This qualifying paper examines the process of developing and implementing the functionality of a retail online store using Java Spring.

The architecture of the information system is based on the principles of client-server architecture, where the client side interacts with the server to receive and process information. The client side is represented by a web interface through which users can view products, make purchases, and process payments. The server side handles user requests, manages the database of products and orders, and provides the necessary information to clients.

A conclusion is made regarding the suitability of using Java Spring for implementing a retail online store, and recommendations are developed for further improvement of the system.

ЗМІСТ

Завдання на кваліфікаційну роботу……2
Реферат 4
Summary 5
Вступ..... 8
1 Аналіз сучасного стану електронної комерції та вимог до інтернет-магазинів 10
1.1 Визначення електронної комерції 10
1.2 Розвиток електронної комерції..... 11
1.3 Переваги електронної комерції 12
1.4 Опис інструментів ведення електронної комерції..... 12
1.5 Висновки до розділу 1 13
2 Розробка моделі інформаційної системи електронної комерції продажу товарів..... 14
2.1 Загальні вимоги до інформаційної системи 14
2.2 Діаграма використання (use-case diagram) 16
2.3 ER-діаграма (entity-relationship) 18
2.4 Діаграма класів (class diagram) 20
2.4.1 Система авторизації на сайті……22
2.5 Діаграма послідовності (sequence diagram)..... 23
2.6 Зовнішні системи програми та діаграма розгортання..... 25
2.7 Обґрунтування вибору та аналіз інструментів та мов програмування використаних під час розробки інформаційної системи. 27
2.7.1 Вибір інструментарію розробки серверу..... 27
2.7.2 Вибір СКБД для взаємодії з сервером 30
2.7.3 Порівняння та вибір існуючих інструментів розробки клієнтської частини інформаційної системи..... 31
2.8 Висновки до розділу 2 32

3 Етапи реалізації програмного забезпечення..... 34
3.1 Розробка ключових модулів серверної частини інформаційної системи…….34
3.2 Розробка користувацького інтерфейсу та функціоналу клієнтської частини……..……...37
3.3 Тестування серверної частини системи..... 40
3.4 Тестування клієнтської частини системи..... 43
3.5 Висновки до розділу 3 45
Висновки 47
Перелік посилань..... 48
Додаток А Клас для обробки запитів ProductController..... 50
Додаток Б Компонент для забезпечення функціоналу кошику магазину..... 51
Додаток В Клієнтський компонент інтерфейсу для авторизації AUTH PAGE 53

ВСТУП

Сучасне ведення підприємницької діяльності вимагає великої кількості різних способів продажу та розповсюдження різної інформації щодо компанії та її виду діяльності. На сьогоднішній день конкуренція в області продажу товарів або послуг вимагає не лише якісної продукції та сервісу, а також інструменти для приваблення та утримання клієнтів. Також розвиток електронної комерції та її вплив на сучасний ринок є фактором, який не можна ігнорувати в сучасному бізнес-середовищі. Згідно з дослідженнями, проведеними Digital Commerce, обсяги світового ринку електронної комерції у 2023 збільшилися на 14,3%, порівнюючи з 2022 році.Цей тренд стає ще більш очевидним з кожним роком [1].

Наприклад, в Україні за останні декілька років спостерігається значне збільшення обсягів онлайн-продажів на 72% в порівнянні з традиційними каналами розповсюдження товарів та послуг. Цьому зростанню сприяли декілька факторів:

- зростання доступності Інтернету: сьогодні Інтернет доступний більшості людей в Україні, що робить онлайн-покупки більш зручними та доступними;
- збільшення використання мобільних пристроїв: все більше людей використовують мобільні пристрої для онлайн-покупок, що робить процес ще більш зручним;
- зміна поведінки споживачів: споживачі стають все більш досвідченими в онлайн-покупках і цінують зручність, широкий вибір та порівняно низькі ціни, які пропонують онлайн-магазини.

У кваліфікаційній роботі досліджується застосування Java Spring до реалізації функціональності інформаційної системи для продажу товарів через інтернет, для оптимізації внутрішніх процесів продажів та покращення взаємодії з клієнтами.

Для створення серверної частини, яка взаємодіє з базою даних, надає та перевіряє ролі кожного користувача в системі та обробляє запити користувача було обрано мову програмування Java з фреймворком Spring Boot який містить в собі набір інструментів для реалізації веб застосунків.

Клієнтська частина платформи, за допомогою якої користувач має змогу взаємодіяти з системою, а саме переглядати товари, робити замовлення або написати листа до служби підтримки компанії, була розроблена за допомогою фреймворку Vue.js, який представляє кожен функціональний елемент користувацького інтерфейсу як окремий компонент, а сама клієнтська програма являє собою взаємодію між усіма компонентами системи.

Зробивши огляд основних тенденцій у сфері електронної комерції та важливості її розвитку, стає очевидним, що дослідження та розробка інформаційних систем для оптимізації продажів через Інтернет є актуальними та важливими завданнями. Проєкт, який описано у даній роботі, спрямований на вирішення цих задач та покращення ефективності взаємодії з клієнтами.

У кваліфікаційній роботі вирішуються наступні задачі:

– проведення аналізу предметної області та визначення сучасних вимог до Інтернет-магазину;
– виконання розробки моделі інформаційної системи електронної коме-рції продажу товарів;
– реалізація функціаналу інтернет-магазину та тестування правильності його роботи у різних умовах.

1 АНАЛІЗ СУЧАСНОГО СТАНУ ЕЛЕКТРОННОЇ КОМЕРЦІЇ ТА ВИМОГ ДО ІНТЕРНЕТ-МАГАЗИНІВ

У сфері продажу та взаємодії різних бізнес-процесів питання електронна комерція почалося в 1970х роках, але поширення на весь світ розглядається з кінця 1990х -років, після затвердження відповідної Декларації міністрів, м. Женева 18-20 травня 1998р, документ World Trade Organisation MIN(98)/DEC/2 [2].

1.1 Визначення електронної комерції

Електронна комерція – це процес пов’язаний з покупкою або продажею товарів та послуг через інтернет або інші електронні засоби зв’язку через будь які електронні пристрої. Іншими словами електронну комерцію можна назвати будь яку транзакцію здійснену з електронного пристрою, підключеного до мережі. До електронної комерції можна віднести не лише торгівлю, а й банкінг, онлайн-страхування, онлайн-маркетинг та обмін інформацією.

Внаслідок розвитку інтернету та значного збільшення користувачів, а з ним бізнесу в цьому середовищі сформувалося поняття електронної комерції як “E-Commerce (електронна комерція)”, яке стало ключовим для учасників ведення онлайн-бізнесу. Існує 5 типів електронної комерції [3]. Розглянемо їх.

C2C (Consumer to Consumer). Схема “споживач до споживача”. Під час реалізації даної схеми послуги надає та отримує людина яка не є комерційною або підприємцем. У приклад такої схеми можна назвати платформу eBay або український OLX.

B2B (Business to Business). Схема “бізнес до бізнесу” забезпечує продаж оптових партій товарів від виробника дилеру.

B2C (Business to Consumer), “бізнес до споживача”, будь яка роздрібна угода між юридичними та фізичними особами, наприклад покупка навчальних

курсів або розрахунок між інтернет-магазином та клієнтом.

E2E (Exchange to exchange), ”біржа до біржі” – використовується для позначення обміну інформацією або угод між вебсайтами, наприклад транзакція між електронним гаманцем та банківською карткою.

G2C (Government to citizen), схема “уряд до громадян”. Характерна будь яка оплата державних послуг такі як комунальні платежі, сплата податків, тощо [3].

1.2 Розвиток електронної комерції

Комунікація з аудиторією є основою електронної комерції, яка найкраще забезпечується інтернет-середовищем. Перша електронна комерція з’явилася у США в 1979 році, потім тренд поширився на країни Європи з 1981 років, а згодом естафету прийняли Китай та інші країни у 1990-ті роки. З того моменту показники та обсяги e-commerce тільки зростають. Наприклад на даний момент населення планети складає приблизно 8.1 млрд осіб, коли до глобальної мережі згідно розрахункам компанії Statista підключено 10.43 млрд пристроїв. Значний відсоток цих пристроїв застосовується в електронній комерції, який вирахувати дуже складно, але згідно цієї ж компанії 62.5% населення планети робило покупки в Інтернет в 2023 році [4].

Об’єми електронної комерції мали великий приріст під час епідемії COVID-19 [5]. Через постійні обмеження на пересування та перебування в людних місцях багато людей було змушено шукати альтернативні способи купівля товарів, і багато виявили що електронна комерція є зручним та безпечним альтернативним варіантом. За 2020 рік обсяг ринку e-commerce перевищив 26 трлн доларів, за даними Discover DHL [6].

Електронна комерція стала необхідною складовою сучасного способу життя, забезпечуючи зручність та доступність покупок для мільярдів користувачів по всьому світу, і цей тренд продовжує активно зростати в зв’язку з поширенням технологій та зміною покупних звичок.

1.3 Переваги електронної комерції

Зростання обсягів онлайн торгівлі обумовлено великою кількістю факторів, однією з головних переваг є відсутність географічних обмежень. Співпраця з лідерами логістичних компаній допоможе охопити аудиторію на весь світ.

Також неможливо проігнорувати перевагу людей до інтернет магазинів через нижчі ціни пов’язані з зменшенням ланцюжку посередників.

Комерційна електронна діяльність охоплює 2 сторони, сторону яка надає послугу чи товар та сторона отримувач, для них можна виділити значну кількість переваг.

Переваги для сторони бізнесу:

– обробка бізнес процесів стає швидше, ефективніше та зручніше;
– значне зменшення роботи з паперами;
– зростає аудиторія та якість обслуговування клієнтів.

Переваги для сторони отримувача:

– завжди доступна докладна інформація про товар або послугу;
– підприємець змушений через високу конкуренцію в електронній комерції робити знижки, проводити акції;
– можливість вибору і порівняння товарів в різних магазинах.

Однак, незважаючи на велику кількість переваг у ведені такої діяльності, важливо враховувати фактори які не є зручними для ведення бізнесу онлайн.

1.4 Опис інструментів ведення електронної комерції

Як було вже сказано в передніх пунктах, електронна комерція є досить об’ємним поняттям. Для якісного ведення та просування онлайн бізнесу важливим етапом роботи є не лише продаж товарів та послуг, а й маркетинг бізнесу, ведення статистики та документообіг, оповіщення співробітників та аналіз даних. Для кожного з цих аспектів роботи існує велика кількість

інструментів.

В Україні великі компанії надають свої інструменти для маркетингу. Компанія Київстар надає платформу ADVISOR, яка дозволяє самостійно налаштовувати розсилки у різних форматах даних і надсилати пропозиції своїй та потенційній аудиторії.

Look-alike – інструмент від компанії Київстар, який за наявними даними в системі проводить підбір подібних користувачів до певної аудиторії задля подальшої комунікації та інформування їх щодо товарів чи послуг через SMS.

Для функції документообігу, створення та обробки таблиць найчастішим рішенням є використання набору інструментів Microsoft Office ,такі як Word та Excel. Вони мають багатий набір функцій для зручного збереження великої кількості даних, а також візуалізація та обробка.

У випадку великої компанії зі значною кількістю працівників є потреба у відправленні листів працівникам. Інструментом автоматизації відправлення електронних листів, або, іншими словами, розсилка, є SendinBlue. Платформа забезпечує необхідну функціональність для оповіщення працівників та клієнтів у визначений час.

Існує незлічена кількість інструментів для якісного ведення бізнесу, що відповідає кажній її потребі.

1.5 Висновки до розділу 1

Електронна комерція стала важливою частиною товарообігу та економіки будь-якої сучасної країни та займає такі сектори як: грошові перекази, банкінг, освітні платформи та онлайн торгівля, обсяги якої зростають з кожним роком завдяки постійній появі необхідних інструментів.

Зі зростанням популярності такого роду діяльності, будь яка компанія стає конкурентоспроможною тільки при наявності свого цифрового ресурсу для просунення власного іміджу та способів надання послуг, продажу товарів, приваблення та утримання клієнтів.

2 РОЗРОБКА МОДЕЛІ ІНФОРМАЦІЙНОЇ СИСТЕМИ ЕЛЕКТРОННОЇ КОМЕРЦІЇ ПРОДАЖУ ТОВАРІВ

2.1 Загальні вимоги до інформаційної системи

На сьогоднішній момент галузь продажу товарів серед усіх галузей електронної комерції займає провідні місця. Покупці – потенціальні користувачі розроблювальної системи повинні мати змогу безперешкодно переглядати інформацію щодо товарів та послуг, наданих компанією, мати інструментарій пошуку та орієнтування на платформі, робити онлайн замовлення, залишати відгуки та завжди мати змогу звернутися до представників компанії або технічної підтримки для вирішення своїх питань щодо користування наданими послугами.

Для реалізації подібних сервісів, кожна система має оперувати певним набором сутностей для забезпечення зручної та ефективної взаємодії з користувачем. Для майбутньої системи виділимо наступні головні сутності:

а) користувач системи, що має різні рівні доступу та привілеї використання, такі як:
   1) гість – неавторизований клієнт, який має змогу лише переглядати товар та робити замовлення;
   2) користувач – клієнт який має власний обліковий запис в системі та має всі права гостя а також переглядати історію замовлень та написати листа до служби підтримки користувачів;
   3) модератор – користувач системи який має доступ до різного інструментарію системи для обробки та відстеження замовлень користувачів;
   4) адміністратор – користувач який має право змінювати контент сторінок, додавати, видаляти, та оновлювати товари також надавати права користувачам або блокувати їх облікові записи та переглядати статистику продажів товарів.

б) товар – сутність яка описує послугу або конкретний предмет який може замовити користувач системи (товари відносяться та розрізняються за категоріями які існують в системі);
в) категорія – це ознака, за якою товари групуються та класифікуються у системі для полегшення навігації та пошуку;
г) підкатегорія – це додаткова розділова одиниця, яка дозволяє детальніше класифікувати товари в межах основної категорії;
д) кошик – сутність яка містить в собі товари які обрав користувач для подальшого замовлення;
е) замовлення – запис в системі з усією необхідною інформацією щодо замовлених товарів, даних замовника та адреси доставки для надання послуги користувачу.

Даний набір сутностей є основним для забезпечення якісного сервісу аудиторії. Слід зазначити, що типи сутності користувача будуть розташовані в системі за ієрархічною структурою, під час якої кожен наступний тип буде мати привілеї та можливості попереднього. Такі привілеї надаються для забезпечення рівня безпеки даних та забезпечення виконання обов’язків працівників, наприклад, модератор – працівник компанії, який повинен перевіряти надходження нових замовлень, обробляти їх та інформувати клієнтів щодо відправлення або наявності обраних клієнтом товарів, це забезпечить клієнта оновленою інформацією щодо бажаного товару, а також дозволяє компанії виявляти та уникати шахрайства, включаючи замовлення, які можуть бути зроблені автоматизованими системами або ботами. Додатковою функцією модератора може бути аналіз та виявлення ненормальних шаблонів або недоречних дій, що можуть вказувати на можливі спроби шахрайства або порушення безпеки даних.

Визначивши усі необхідні сутності, система повинна забезпечувати своїми функціональними можливостями наступні пункти:

– мати простий та зрозумілий інтерфейс;
– прозора та ненадмірна інформацію про товари;
– швидка та надійна система оплати;

– мінімальна кількість дій для створення замовлення.

Крім того, в майбутню програму було запропоновано додати ряд додаткових корисних функцій та деяких обмежень:

– фільтрація товарів за ціною, кольором або брендом;
– сортування цін товарів за спадання або зростанням;
– пошук товару за назвою або ідентифікатором в системі у відповідному полі яке знаходиться у шапці сайту;
– обмеження для відгуку, користувач може залишити тільки один відгук і тільки у випадку якщо він купив цей товар.

Представлений набір сутностей відіграє ключову роль у забезпеченні якісного обслуговування аудиторії. Зазначено, що розташування користувачів у системі за ієрархічною структурою гарантує ефективну взаємодію з платформою, забезпечуючи кожному типу користувача відповідні привілеї та можливості.

Визначені сутності, такі як користувачі, товари, категорії, підкатегорії, кошик та замовлення, є фундаментальними для оптимального функціонування системи електронної комерції. Крім того, виокремлено ключові функціональні можливості, які повинна забезпечувати система для зручного користування.

Додатково, у майбутню програму було запропоновано додати ряд корисних функцій та обмежень.

У цілому, цей набір сутностей та функціональних можливостей визначає стратегію розробки та забезпечує якісне обслуговування користувачів системи електронної комерції.

2.2 Діаграма використання (use-case diagram)

Для опису сценаріїв використання системи користувачем було розроблено діаграму використання [7]. Така діаграма показує роботу системи а також актуальні функції для різних типів користувачів без уточнення будь якої реалізації самих функцій. Даний тип діаграми зображено на рис. 2.1.

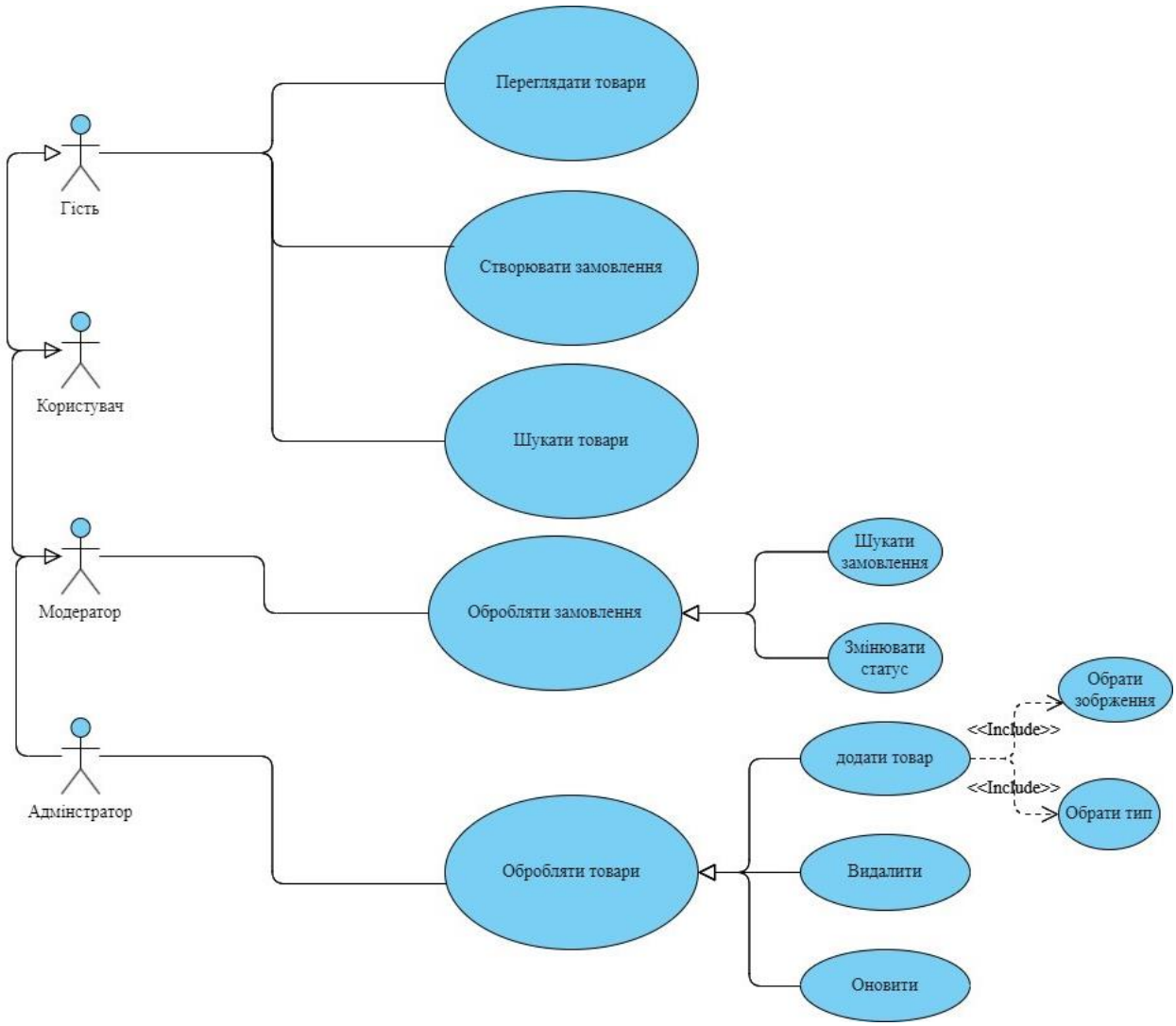


Рисунок 2.1 – Діаграма використання системи взаємодії з товарами

На діаграмі зображено акторів, прецедент та зв’язки між ними.

Актор – це абстракте поняття деякого об’єкта який має змогу взаємодіяти з системою, на рисунку можна побачити 4 актори, розташовані в ієрархічному вигляді, де кожен рівень з цієї ієрархії має змогу взаємодіяти з різними функціями. Така ієрархічна система створена за допомогою відношення узагальнення який наявний в мові моделювання UML та показує що кожен з акторів може користуватися усіма варіантами використання з яким з’єднаний інший актор.

Прецедент – визначає поведінку системи у вигляді функцій які може використати актор.

Така діаграма застосовується для аналізу та проєктування систем на

основі описаних замовником вимог. Діаграма акторів та прецедентів є важливим інструментом для розуміння взаємодії між різними користувачами та системою в цілому. Вона допомагає уточнити ролі різних учасників системи та їхні взаємодії з нею для досягнення поставлених цілей.

2.3 ER-діаграма (entity-relationship)

Діаграма відношень сутностей відображає відношення набору об’єктів які зберігаються в базі даних.

Сутність – це об’єкт реального світу, який знаходиться в навколишньому середовищі. Наприклад товар є окремою сутністю з якою взаємодіє користувач. Для розуміння відношень між такими сутностями було створено діаграму рисунок 2.2.

Наведена діаграма описує сутності та їх відношення у інформаційній системі, які будуть зберігатися у вигляді таблиць під час створення схеми бази [8] даних майбутнього інтернет-магазину [9].

На рисунку зображено головні сутності такі як: користувач та його роль в системі, товари та їх види, категорія до якої належить товар.

Також зображено сутності які будуть зберігати дотакову інформацію щодо оцінки товару, його зображенн та коментарів які зробив користувач.

Аналізуючи дану діаграму, слід сказати що основним видом відношення є “один до багатьох” це означає що один елемент в одній з сутностей взаємодіє з кількома елементами в іншій сутності, іншими словами один об’єкт або екземпляр сутності має зв’язок з багатьма іншими об’єктами або екземплярами іншої сутності.

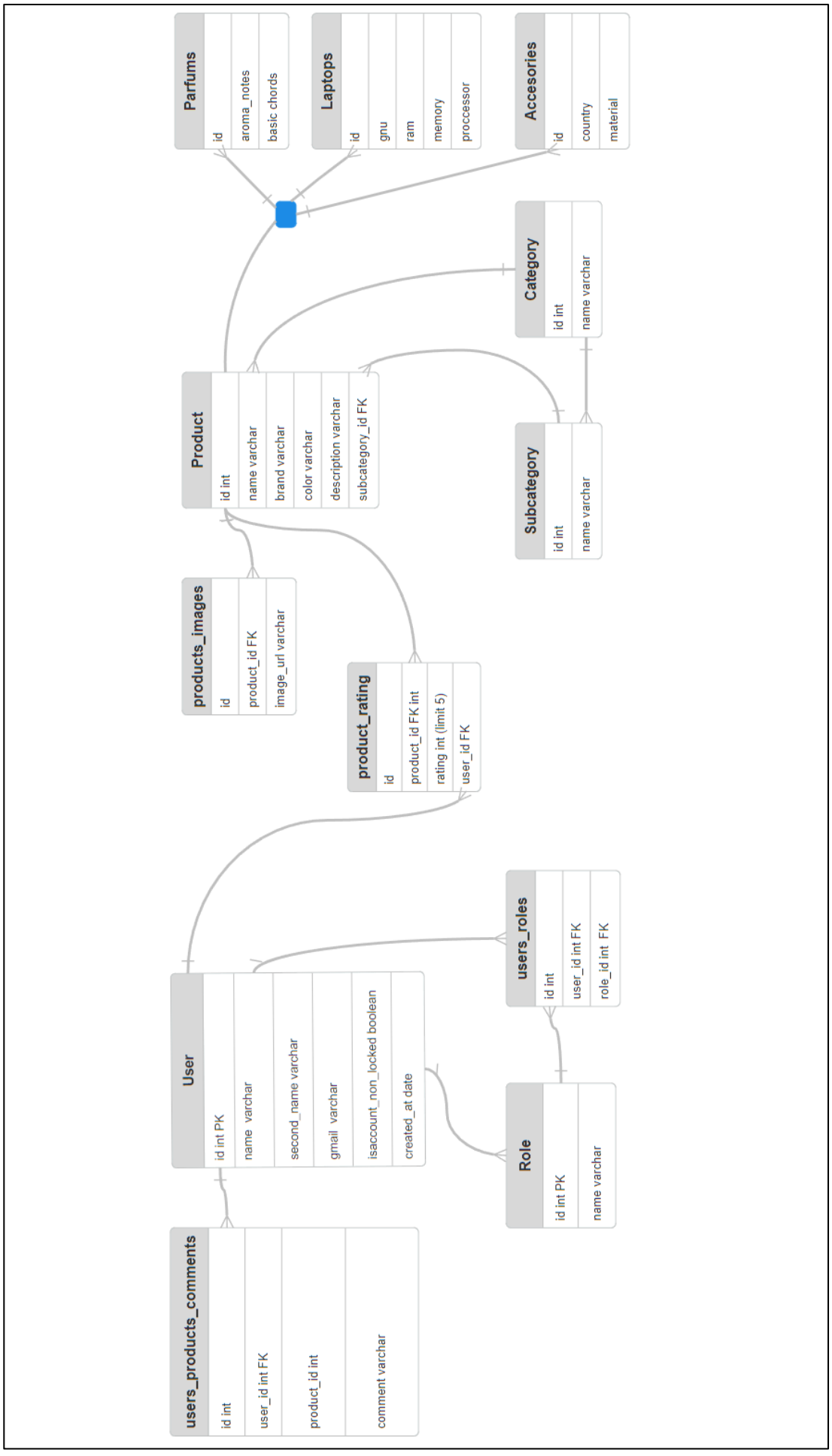


Рисунок 2.2 – ER-діаграма інформаційної системи

2.4 Діаграма класів (class diagram)

Наступна діаграма є однією з найбільш важливих та корисних діаграм яка детально візуалізує систему шляхом моделювання її класів, операцій, властивостей та зв’язків між екземплярами класів. Слід зазначити що дана нотація використовується для побудови та зображення об’єктно-орієнтованих систем.

За допомогою даної діаграми можна якісно відобразити існуючі класи у системі, властивості та зв’язки між об’єктами та наявні операції між кожним класом системи. Надання чіткого розуміння щодо архітектури майбутнього ПО, допомагає при аналізі та розробці коду.

Такий інструмент використовується для комунікації між розробниками , оскільки він надає однозначне уявлення щодо системи. Він допомагає уникнути непорозумінь між членами команди та сприяє ефективній співпраці. Додатково може бути використаний для порівняння різних версій програми або для аналізу впливу змін на її структуру.

Для майбутньої системи було розроблено діаграми класів для різних модулів програми [10]. Наприклад рисунок 2.3 – Діаграма класів для моделі взаємодії товарів в системі.

Діаграма показує структуру модуля системи для взаємодії з товарами та іншими класами, для забезпечення функціональності інтернет-магазину з можливістю створити замовлення. На діаграмі зображено класи конкретних товарів які наявні в системі, користувач з роллю адмінстратора може додати їх до продажу, видалити або редагувати. Репозиторії які являють собою інтерфейси для взаємодії з базою даних та забезпечують основні функції для взаємодії з товарами, а саме: додавання, пошук, оновлення та видалення. Сервіс який проводить перевірку та обробляє дані знайдені репозиторієм. Клас для створення категорій та підкатегорій до якої належить товар, для фільтрації та зручного пошуку товарів у системі. Зображено клас який описує сутність замовлення, яке буде створюватися користувачами після замовлення товарів через клієнтський інтерфейс та оброблятися модераторами магазину.

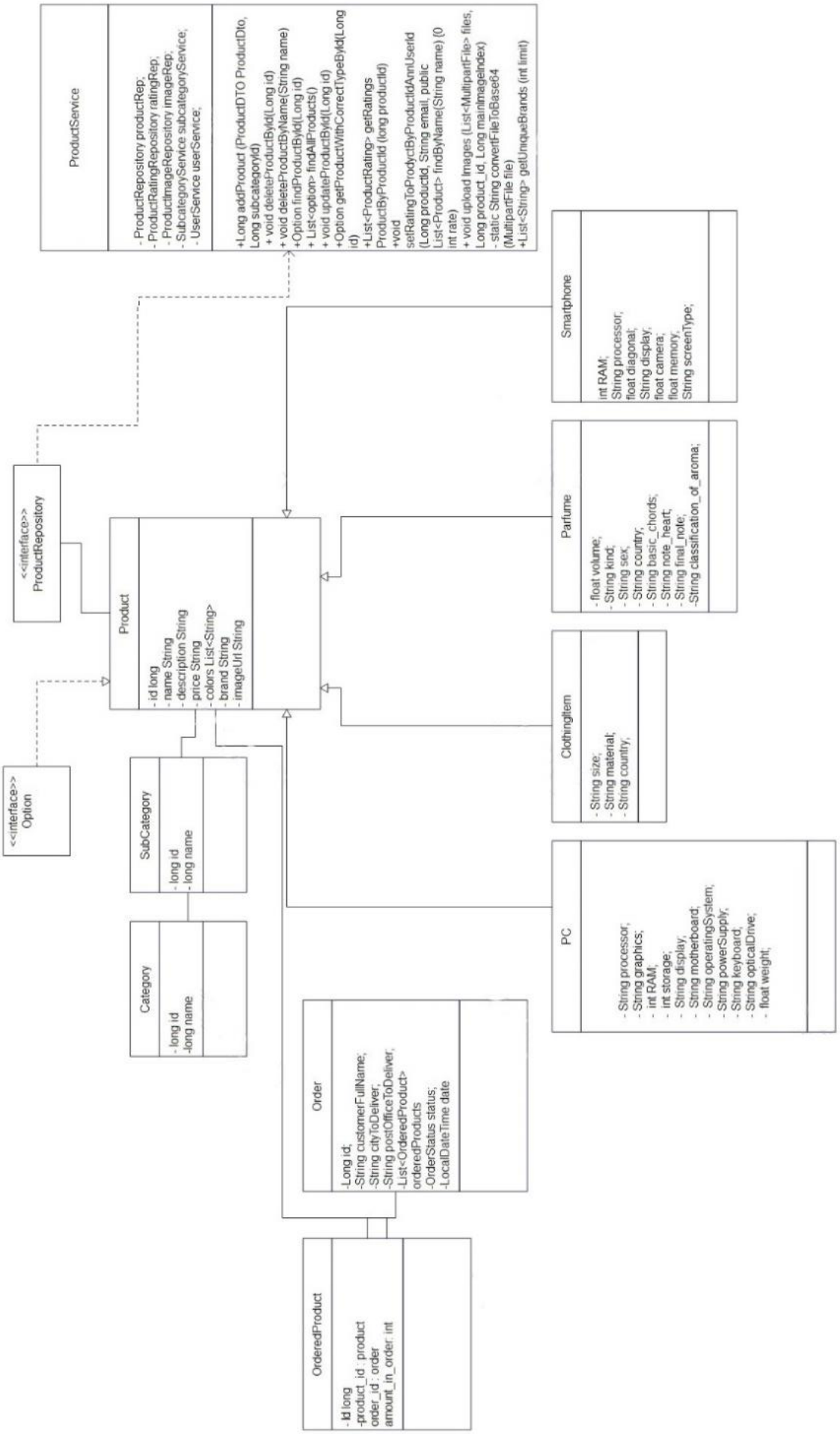


Рисунок 2.3 – Діаграма класів для моделі взаємодії товарів в системі

Слід зазначити що для оптимізації навантаження на сервер, сутність кошику замовлень було реалізовано на стороні клієнту та буде розглянуто у додатку Б.

2.4.1 Система авторизації на сайті

Для створення системи авторизації у застосунку, було обрано 2 способи. Перший спосіб – використання зовнішнього інструменту авторизації OAuth 2.0 який надає змогу увійти на вебсайтах або в додатки за допомогою облікового запису на інших ресурсах. Такий спосіб входу до системи надає змогу створити або увійти до програми за допомогою облікового запису Google або соціальній мережі Facebook або X (Twitter в минулому). Другий спосіб, створення та надання тимчасового токену який містить в собі всю інформацію щодо користувача системи, таку як логін, пароль та його ролі. Токен отримується шляхом шифрування всіх даних користувача з секретним словом та надає можливість користувачу отримувати приховані дані згідно його ролей. Для створення такого функціонального модулю, було розроблено ряд класів які використовують сутність користувача та основі його даних забезпечують весь функціонал авторизації в системі. Діаграму класів вказаної функціональності наведено на рисунку 2.4.

Діаграма описує детальний процес авторизації користувача в системі, який використовує JSON Web Tokens (JWT) для захисту ресурсів. Використовуються наступні алгоритми дій для роботи з даними класами.

Реєстрація:

– користувач надсилає запит на реєстрацію (SignUpRequest) з клієнту;
– AuthenticationController створює екземпляр класу User та зберігає його в базі даних;
– AuthenticationService генерує JWT та надсилає його користувачеві.

Авторизація:

– користувач надсилає запит на авторизацію (SigninRequest);

– AuthenticationController перевіряє ім’я користувача та пароль, надані користувачем;
– якщо ім’я користувача та пароль вірні, AuthenticationService генерує JWT та надсилає його користувачеві.

Користувач може використовувати JWT для доступу до захищених ресурсів системи, які забезпечують його ролі.

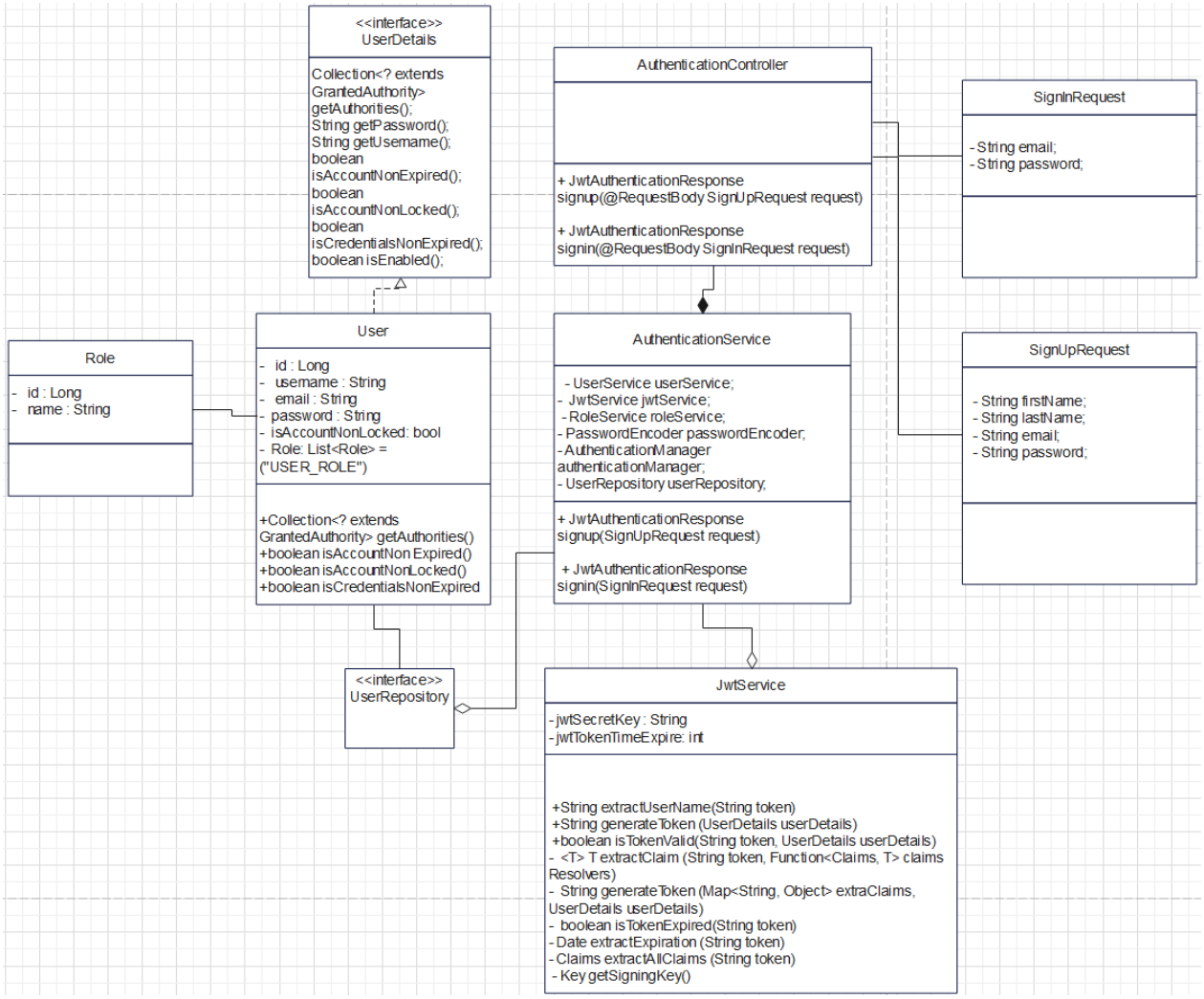


Рисунок 2.4 – Діаграма класів системи авторизації

2.5 Діаграма послідовності (sequence diagram)

Діаграма послідовності використовується для моделювання процесів виконання сценаріїв використання, показує як взаємодіють об’єкти під час

сценарію, не показуючи як вони пов’язані один з одним. Таке відображення виконання функціональності представлене у вигляді блоків, у верхній частині сторінки зліва на право. Кожен об’єкт стовпець який тягнеться зверху донизу сторінки, в той час коли повідомлення між об’єктами зображені у вигляді стрілок. Порядок виконання такої функції зчитується зверху вниз та зліва направо починаючи з точки входу у верхньому лівому куті. Приклад діаграми послідовності для інформаційної системи інтернет-магазину наведено на рисунку 2.5.

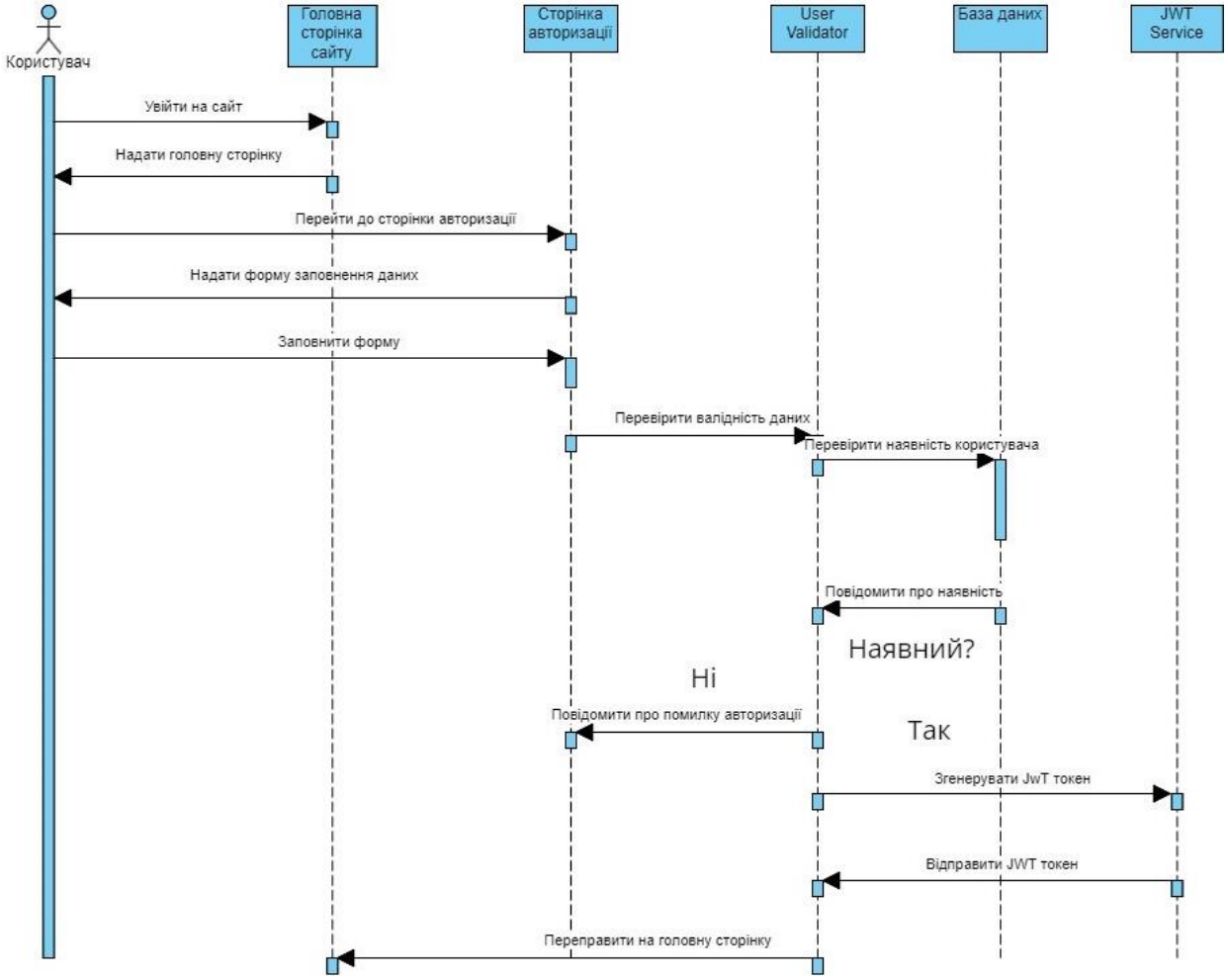


Рисунок 2.5 – Діаграма послідовності авторизації користувача

Рисунок демонструє взаємозв’язок між об’єктами які використовуються для авторизації користувача, також зображено що база даних виступає як окремий об’єкт та за відсутності користувача , отримуємо помилку

авторизації.

Використання даної діаграми надає значну кількість переваг, такі як:

– вона демонструє взаємодію між компонентами в хронологічному порядку, в якому відбувається взаємодія;
– візуалізує взаємодію між частинами системи, що дозволяє легко перевірити логіку замовнику та розробникам;
– надає можливість деталізувати варіанти використання в одну або декілька діаграм послідовності для забезпечення глибокого розуміння бізнес-процесу.

Використання такого типу візуалізації бізнес-процесів є дуже зручним та ефективним в багатьох випадках, але при моделюванні майбутньої інформаційної системи слід бути обачним через можливість витратити час та зусилля для створення повного набору діаграм такого типу для кожного використання системи, який може бути надлишковим та непотрібним.

2.6 Зовнішні системи програми та діаграма розгортання

На сьогоднішній день існує велика кількість інструментів та різних способів оптимізації розроблювальних систем. Одним з таких підходів є використання зовнішніх систем для уникнення дублювання функціональності яка вже існує та надається компаніями для використання. Використання таких систем дуже економить час створення проєкту, кошти на розробку та тестування а також поліпшує архітектуру майбутньої програми. Для взаємодії розроблювальної системи з іншими, компанії надають API(інтерфейси програмування застосунків) які дозволяють взаємодіяти з цими системами шляхом використання відомих методів комунікації. Цей підхід робить платформу більш гнучкою та розширювальною.

Для візуальної відображення такої системи використовують діаграму розгортання, яка відображає архітектурну модель програми та її компонентів

на апаратному забезпеченні. Вона допомагає визначити як різні частини програми такі як серверна частина, клієнт, база даних та зовнішні системи взаємодіють один між одним утворюючи повнофункціональне програмне забезпечення. Це допомагає управляти ресурсами, оптимізувати швидкодію та забезпечити стабільну роботу програми у виробничому середовищі. Приклад даної діаграми було розроблено під час розробки інформаційної системи інтернет-магазину та зображено на рисунку 2.6.

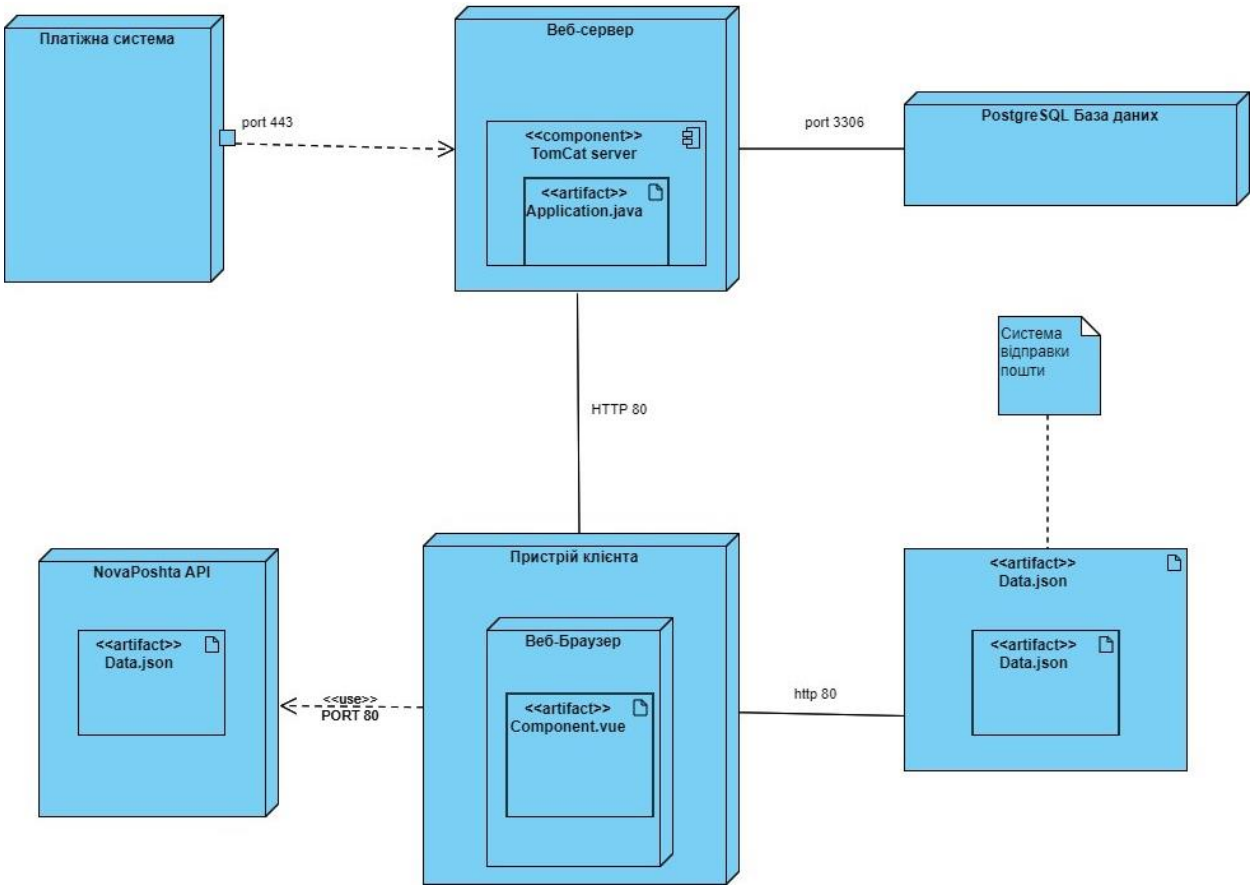


Рисунок 2.6 – Діаграма розгортання інтернет-магазину

Наведена діаграма демонструє загальну архітектуру системи в якій заначено такі компоненти як серверна частина яка складається з середовища виконання, тобто програмне забезпечення серверу Apache Tomcat та виконуваних частин написаних на мові програмування java. Видно що сервер взаємодіє з базою даних та платіжною системою на різних портах.

Клієнтська частина системи реалізована засобами Vue.js яка

запускається в браузері що також продемонстровано на діаграмі. Зовнішні системи з якими взаємодіє клієнтська частина – API компанії яка надає послуги з доставкою товарів та система відправки листів на пошту від клієнтів до служби підтримки. Клієнтська частина програми використовує зовнішню систему пошти, яка надсилає адреси усіх відділень пошти у форматі json для подальшого відображення їх у користувацькому інтерфейсі. Такий підхід забезпечує автоматизоване оновлення даних про адреси відділень пошти без необхідності ручного втручання, що сприяє підтримці актуальності інформації для користувачів системи доставки товарів.

Даний тип діаграм є дуже корисним інструментом для демонстрацій найбільш узагальненого виду архітектури, без уточнення деталей реалізації функцій, схем бази даних та інших технічних аспектів.

2.7 Обґрунтування вибору та аналіз інструментів та мов програмування використаних під час розробки інформаційної системи

2.7.1 Вибір інструментарію розробки серверу

Першим кроком в розробці майбутньої системи є вибір технологій та інструментів розробки програмного забезпечення. Наявність досить великої кількості таких засобів розробки дає змогу проаналізувати декілька варіантів та обрати найзручніший та оптимальний. При виборі технологій слід враховувати велику кількість факторів такі як:

- наявність достатньої кількості документації засобів;
- можливість забезпечити виконання вимог замовника за допомогою даної технології;
- технічні навички розробників;
- витрати на технологію;

– наявність готових компонентів системи для вирішення розповсюджених задач;
– актуальність інструменту.

Враховуючи вищезгадані фактори, коло вибору серед технологій скоротилося до вибору серед мов об’єктно орієнтовного програмування та їх фреймворків, таких як :

– Java з фреймворком Spring;
– C# з фреймворком .NET;
– Python з фреймворком Django.

Проведемо порівняння серед них в таблиці 2.1.

Таблиця 2.1 – Порівняння інструментів розробки

	Інструмент		
Критерій	Java Spring	C# .Net	Python Django
Швидкість розробки	Середня	Середня	Швидка
Легкість вивчення	Середня	Середня	Легка
Масштабованість	Висока	Висока	Висока
Продуктивність	Висока	Висока	Середня
Безпека	Висока	Висока	Середня
Спільнота та підтримка	Велика та активна	Велика та активна	Велика та активна
Типові сфери застосування	Вебзастосунки, корпоративні системи, мобільні застосунки	Вебзастосунки, десктопні застосунки, ігри	Вебзастосунки, машинне навчання, штучний інтелект

Для опису роботи з базами даних було складено таблицю 2.2.

Таблиця 2.2 – Інструменти взаємодії з базами даних

Технологія	Інструменти доступу до баз даних
Java Spring	Підтримує різні драйвери JDBC Hibernate для ORM (ORM – модель взаємодії об’єктів на основі даних з таблиць) Spring Data для спрощеного доступу до даних
C# .NET	ADO.NET забезпечує низькорівневу взаємодію з бд Entity Framework-ORM-фреймворк аналог Hibernate до Java.
Python Django	Django ORM забезпечує простий та інтуїтивно зрозумілий інтерфейс для роботи з БД.

Обираючи між даними технологіями слід зазначити наступні критерії:

– Django: вважається найбільш простим для початківців завдяки інтуїтивно зрозумілому ORM та Django admin для керування даними;
– ADO.NET та Entity Framework може бути складним для початківців, полегшує роботу, але має криву навчання;
– Java Spring: JDBC вимагає більше знань SQL, Hibernate та Spring Data спрощують роботу, але потребують вивчення та найефективніший для складних проєктів.

Аналізуючи складені таблиці та інші порівняння можна сказати що кожен інструмент має переваги в деякій області але враховуючи вимогу майбутнього інтернет-магазину, слід виділити найголовніші критерії такі як висока масштабованість, висока безпека та високий рівень продуктивності. Такі вимоги задовольняє мова Java яка за рахунок компіляції коду та швидкої й більш оптимізованої роботи з базами даних порівняно з іншими інструментами, забезпечить швидкодію одночасного використання усього функціоналу системи для великої кількості користувачів. На відміну від мови програмування python, яка є інтерпретованою та уступає в продуктивності

мові Java та C#. Використання Django є зручним для початківців, має велику швидкість в розробці, не вимагає великих навичок та використовується для досить ненавантажених проєктів, в той час коли .Net є збалансованим вибором, який надає оптимальний результат продуктивності та вимагає певний набір навичок у розробці.

2.7.2 Вибір СКБД для взаємодії з сервером

Система керування базою даних – є також важливою частиною роботи усієї системи. Від неї залежить спосіб збереження даних та способи взаємодії їх з системою. Вибір бази даних залежить також від багатьох факторів, такі як:

- вимоги до даних: різні бази даних мають різні можливості щодо типів даних, які вони можуть зберігати, і їхньої структури (наприклад, реляційні бази даних добре підходять для структурованих даних, тоді як NoSQL бази даних можуть краще впоратися з неструктурованими даними);
- масштабованість: вибір бази даних також залежить від очікуваної кількості даних та потреб системи у масштабованості;
- швидкодія: деякі бази даних оптимізовані для швидкого доступу до даних або виконання складних запитів;
- безпека даних;
- можливість міграції: важливо також враховувати можливість міграції даних з однієї бази даних до іншої, якщо у майбутньому знадобиться змінити систему управління базою даних.

Інформаційну систему інтернет-магазин цілком задовольняє вибір реляційної системи в якій всі записи зберігаються у вигляді таблиць. Протягом великої кількості часу реляційні бази даних займали провідні місця у сфері збереження даних. Незважаючи на значний попит у альтернативних, нереляційних базах даних, традиційні структурні СКБД вважаються

найкращим вибором на роль збереження даних загального призначення. Прикладом такої системи керування базами даних є PostgreSQL який пропонує широкий набір функцій та можливостей для зберігання та обробки даних у великих масштабах [11]. Важливою перевагою реляційних баз даних, таких як PostgreSQL, є їхня надійність, консистентність та можливість забезпечення цілісності даних завдяки використанню транзакцій.

По відношенню роботи з базами даних, попередньо обрана технологія розробки Java Spring Boot, має два варіанти роботи з ними JDBC та JPA. Spring підтримує усі варіанти, надаючи свої абстракції для реалізації функціоналу роботи зі сховищем даних, які будуть розглянуть у наступних розділах.

2.7.3 Порівняння та вибір існуючих інструментів розробки клієнтської частини інформаційної системи

Для якісної взаємодії користувача та системою, стає питання у створенні користувацького інтерфейсу з певним набором функціоналу. Наразі існує дуже велика кількість способів створення клієнтської частини але в даному розділі буде розглянуто та порівняно два з найпопулярніших інструменти для вирішення задач такого типу – Vue.js та React.

Vue – це фреймворк на базі мови програмування JavaScript, який використовує принцип розділення кожного функціонального елементу інтерфейсу на компоненти а сама програма – це взаємодія між такими компонентами [12].

React – не є фреймворком, являє собою бібліотекою мови програмування JavaScript, яка має аналогічний до vue.js функціонал та використовує систему взаємодій окремих компонентів. Компоненти в React можуть бути повністю самодостатніми та перевикористовуваними. Компоненти в React також можуть бути вкладеними один в одного, що дозволяє створювати складні інтерфейси з логічною структурою. Крім того, React пропонує можливості для

створення стану компонентів, який визначає, як компонент повинен реагувати на зміни та інтеракції користувача.

За допомогою React можна створювати як невеликі вебсайти, так і складні вебдодатки з великою кількістю взаємодії і асинхронних запитів до сервера.Зробимо порівняння та виберемо інструмент за допомогою таблиці 2.3.

Таблиця 2.3 – Порівняння інструментів розробки клієнтської програми

Критерій	Vue.js	React
Тип	JavaScript-фреймворк	Бібліотека JavaScript
Синтаксис шаблонів	HTML-шаблони або JSX	Тільки JSX
Прив’язка даних	Двостороннє оновлення DOM	Односторонній потік даних
Крива навчання	Легша для початківців	Складніший для вивчення
Спільнота та екосистема	Велика та активна спільнота, багато сторонніх бібліотек	Дуже велика та активна спільнота, більше офіційних інструментів

Порівнюючи такі засоби розробки, було обрано використовувати фреймворк Vue.js, який в змозі забезпечити усі вимоги програми та має більш легку криву навчання та деякі полегшені способи реалізації, наприклад використання HTML-шаблонів для створення інтерфейсів.

2.8 Висновки до розділу 2

В даному розділі було сформульовано загальні вимоги до інформаційної системи та розглянуто й розроблено частину існуючих діаграм з різним

ступенем деталізації архітектури та технічних аспектів розробки функціональності системи за допомогою інструменту моделювання UML. Було створено діаграму використання для визначення основних функцій майбутньої системи, доступних для кожного виду користувачів. Розроблено ER-діаграму для визначення основних атрибутів сутності та зв’язків між кожною сутністю для створення схеми бази даних. Проілюстровано діаграми класів для системи взаємодії з товарами та системи авторизації, на яких було зображено схему класів які будуть створені на стороні сервера. Для детальної покрокової демонстрації авторизації на сайті, показано діаграму послідовності яка відображає всі внутрішні процеси які відбуваються під час авторизації. На останок було візуалізовано діаграму розгортання системи, яка показує які інструменти та середовища використовуються для завантаження програми в глобальний простір.

Було проаналізовано та обґрунтовано вибір обраних інструментів розробки програмного забезпечення для реалізації інтернет-магазину.

3 ЕТАПИ РЕАЛІЗАЦІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Розробка ключових модулів серверної частини інформаційної системи

Однією з найважливіших сутностей в інформаційній системі інтернет-магазину є товар. Для створення різних операцій над товарами вони повинні мати власну модель на сервері. Код об’єкту товару наведений в лістингу на рисунку 3.1.

```
@Entity
@Table(name = "products")
@Inheritance(strategy = InheritanceType.JOINED)
@AllArgsConstructor
@JsonInclude(JsonInclude.Include.NON_NULL)
@JsonIgnoreProperties({"hibernateLazyInitializer", "handler"})
public class Product implements Option {
    @JsonIgnore
    @OneToMany(mappedBy = "product", cascade = CascadeType.ALL, orphanRemoval = true)
    private List<OrderedProduct> orderedProducts;
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private Long id;
    private String productName;
    private String description;
    private double price;
    private String brand = "none brand";
    private List<String> colors;

    @JsonIgnore
    @ManyToOne(cascade = CascadeType.PERSIST)
    @JoinColumn(name = "subcategory_id")
    private Subcategory subcategory;

    private String displayImageURL = "https://i.ibb.co/R0bywzW/istockphoto-990697446-612x612.jpg";
    @JsonIgnoreProperties("product")
    @OneToMany(mappedBy = "product", cascade = CascadeType.ALL, fetch = FetchType.LAZY)
    private List<ProductImage> images;
}
```

Рисунок 3.1 – Модель товару на сервері

Наведений код має набір полів що описують загальні характеристики товару такі як ціна, набір кольорів, опис та інше.

Spring Boot використовує велику кількість анотацій для задання певних

характеристик коду, наприклад анотація Entity вказує що даний клас є сутністю а також за допомогою її, середовище Spring може автоматично управляти залежностями в програмі ініціалізуючи поля інших класів даним, де це необхідно [13]. Подальший код інших модулів програми наведений в наступних лістингах.

Класи-нащадки мають додаткові поля, що описують конкретний товар, та метод ініціалізації.

Проведення операцій з товарами забезпечується методами класу ProductService, приклад якого зображено на рисунку 3.3.

```
@Entity
@Table(name = "laptop")
@Data
@EqualsAndHashCode(callSuper=false)
@AllArgsConstructor
public class Laptop extends Product implements Option{

   private String processor;
   private String GNU;
   private String RAM;
   private String memory;
   private String motherboard;

   public Laptop() {
      super();
   }
   public void initializeFromAdditionalFields(Map<String, Object> additionalFields) {
      this.setProcessor(additionalFields.get("processor").toString());
      this.setGNU(additionalFields.get("graphics").toString());
      this.setRAM(additionalFields.get("RAM").toString());
      this.setMemory(additionalFields.get("storage").toString());
      this.setMotherboard(additionalFields.get("motherboard").toString());
   }
}
```

Рисунок 3.2 – Дочірній клас головного класу Product

```
public Long addProduct(ProductDTO ProductDto,Long subcategoryId) {
      Map<String, Object> additionalFields = ProductDto.getAdditionalFields();
      Subcategory subcategoryForCurrentProduct = subcategoryService.findById(subcategoryId).orElse(null);
      Product commonProduct  = ProductDto.getProduct();
      commonProduct.setBrand(additionalFields.get("brand").toString());

      ProductFactory productFactory =
ProductFactoryProvider.getProductFactory(subcategoryForCurrentProduct);
      Product specificProduct = productFactory.createProduct();

      BeanUtils.copyProperties(commonProduct, specificProduct);

      specificProduct.setSubcategory(subcategoryForCurrentProduct);
      productFactory.initializeAdditionalFields(specificProduct, additionalFields);

      return this.productRep.save(specificProduct).getId();
}
```

Рисунок 3.3 – Метод класу ProductService для додавання товару в базу даних

У даному методі, параметром виступають дані товару та ідентифікатор підкатегорії. Для оптимізації коду та можливості майбутнього масштабування, метод використовує шаблон проектування “Фабричний метод”, який дозволяє створювати об’кти без явного уточнення конкретного класу цих об’єктів. Він використовується для розширення гнучкості коду і можливості майбутнього масштабування програми. Основна ідея полягає в тому, що визначається інтерфейс для створення об’єктів, а конкретизація реалізації залишається на підкласах, які можуть визначати, який саме об’єкт повертати. Код шаблону “Фабричний метод” наведений на рисунку 3.4. Код класу який надає “фабрики” за умовою, описаний в лістингу на рисунку 3.5.

```
public interface ProductFactory {
    Product createProduct();
    void initializeAdditionalFields(Product product, Map<String, Object> additionalFields);
}
public class LaptopFactory implements ProductFactory{
    @Override
    public Product createProduct() {
        return new Laptop();
    }
    @Override
    public void initializeAdditionalFields(Product product, Map<String, Object> additionalFields) {
        ((Laptop) product).initializeFromAdditionalFields(additionalFields);
    }
}
```

Рисунок 3.4 – Фабричний метод з додатковою ініціалізацією полів

```
public class ProductFactoryProvider {
    public static ProductFactory getProductFactory(Subcategory subcategory) {
        switch (subcategory.getName()) {
            case "Parfumes":
                return new ParfumeFactory();
            case "Smartphones":
                return new SmartphoneFactory();
            case "Laptops":
                return new LaptopFactory();
            case "PC":
                return new PcFactory();
            case "men clothes":
            case "women clothes":
            case "child clothes":
                return new ClothesItemFactory();
            case "accessories":
                return new AccessoryFactory();
            default:
                return new DefaultFactory();
        }
    }
}
```

Рисунок 3.5 – Код класу ProductFactoryProvider

Було продемонстровано код основних компонентів пов’язаних з товарами. Клас контролеру через його обсяг наведений в додатку А.

3.2 Розробка користувацького інтерфейсу та функціоналу клієнтської частини

Як було вже зазначено раніше, інтерфейс користувача повинен бути зручним, інтуїтивно зрозумілим та привабливим щоб користувач мав бажання користуватися та взаємодіяти з системою. За допомогою обраного інструменту розроблено ряд візуальних компонентів інтерфейсу та продемонстровано в наступних лістингах (див. рис. 3.6).

```
<template>
  <div id="HeaderContainer">
    <header>
      <div id="autorizatinBtnPannel">
        <div v-if="!isAuthorised">
          <router-link to="/login">
          <i class="btn bi bi-box-arrow-in-right"></i>
        </router-link>
       </div>
        <div v-else>
       <div @click="handleLogout">
          <i class="btn bi bi-door-closed-fill"></i>
       </div>
        </div>
      </div>
       <div class="language-container">
        <select id="languages" class="language-dropdown" ref="languages" @change="changeLanguage">
           <option value="en">Eng</option>
           <option value="uk">Укр</option>
           <option value="ru">Рус</option>
        </select>
       </div>

      <div class="header-content">
       <router-link  :to="isHomeRoute ? '' : '/home'" @click="refreshOrRedirect">
       <div class="logo">
        <img src="../assets/3.png" alt="Logo" id="logoItem">
       </div>
      </router-link>

        <form class="d-flex flex-grow-1" role="search" @submit.prevent="searchData" id="formSearch">
         <input id="fieldSearch" class="form-control form-control-lg me-2" type="search"  placeholder="...." aria-label="Search" v-model="inputedProductName">
         <button class="btn btn-outline-dark btn-lg" id = "btnSearch" type="submit">{{$t(('search'))}}</button>
        </form>
        <div class="header-buttons" >
```

```
        <router-link to="/home">
              <i class="btn bi bi-house"></i>
        </router-link>
        <router-link to="/contactUs">
           <i class="btn bi bi-chat-right-dots"></i>
        </router-link>
        <router-link to="/about">
             <i class="btn bi bi-book"></i>
        </router-link>
        <router-link to="/cart" class="cart-icon-container">
           <i class="btn bi bi-cart3"></i>
           <span class="cart-count">{{ countItemsCart }}</span>
        </router-link>
        <router-link to="/admin" v-if="userRoles && userRoles.includes('ROLE_ADMIN')">
          <i class="btn bi bi-dpad-fill"></i>
        </router-link>
        <router-link to="/moderator" v-if="userRoles && userRoles.includes('ROLE_MODERATOR')">
          <i class="btn bi bi-person-fill"></i>
        </router-link>
      </div>
    </div>
  </header>
 </div>
</template>
```

Рисунок 3.6 – Шаблон головної навігаційної панелі сайті

Компонент має деякий функціонал та код розрахування стану деяких елементів який наведено на рисунку 3.7.

```
export default {
 data() {
  return {
   isAuthorised:null,
   userRoles:null,
   inputedProductName:"",
  };
 },
 mounted() {
  this.$nextTick(() => {
   this.$refs.languages.value = this.$i18n.locale;
  });
  this.checkAuthorization();
  this.loadCartFromStorage();
 },
 computed:{
  isHomeRoute() {
   return this.$route.path === '/home';
  },
  countItemsCart(){
   return this.$store.getters.cartItemsCount;
  }
 },
 methods: {
  changeLanguage(event) {
   const selectedLocale = event.target.value;
   this.$i18n.locale = selectedLocale;
  },
```

```
  checkAuthorization(){
    const token = localStorage.getItem('jwtToken');
      if (token) {
        this.isAuthorised = true;
        this.userRoles = localStorage.getItem('userRoles');
      } else {
        this.isAuthorised = false;
      }
  },
  handleLogout() {
    localStorage.removeItem('jwtToken');
    localStorage.removeItem('emailUser');
    this.checkAuthorization();
    this.$router.go();
  },
  async searchData(){
     await this.$store.dispatch('GET_SEARCHEDPRODUCTS', {
        name: this.inputedProductName.toLowerCase(),
    });
  },

  refreshOrRedirect() {
    if (this.isHomeRoute) {
      window.location.reload();
    } else {
      this.$router.push('/home');
    }
  },

  loadCartFromStorage() {
    this.$store.commit("LOAD_CART_FROM_STORAGE");
  }

 },

};
</script>
```

Рисунок 3.7 – Програмний код компонента HeaderPage.vue

Візуальний компонент зображено на рисунку 3.8.

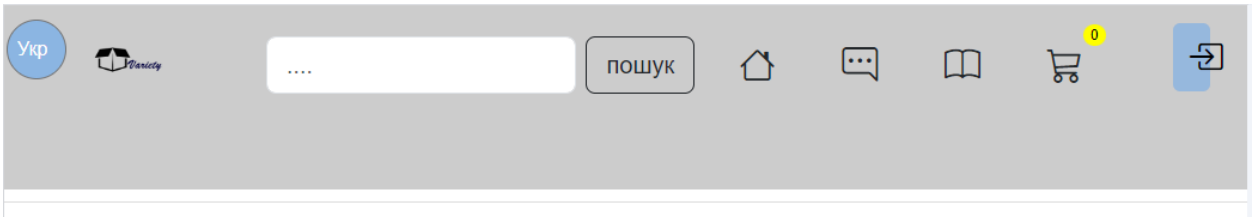


Рисунок 3.8 – Шапка сайту

Даний компонент містить усі необхідні елементи навігації, такі як кнопки-посилання на розділи, поле пошуку товарів, перемикання мов сторінки та кнопка авторизації.

Через великі обсяги CSS-таблиць стилів, код наводитися не буде, а програмна реалізація інших компонентів таких як кошик товарів буде наведено в додатку Б та компонент авторизації додаток В.

3.3 Тестування серверної частини системи

Тестування інформаційної системи є досить широким поняття яке охоплює значну кількість видів, способів та метрик якості тестування системи яке виконується на різних стадіях програмної реалізації. В даному розділі буде наведено два види тестування серверної частини програми, які були виділені з усіх інших заходів тестування системи. Один з таких видів є модульне тестування, суть якого полягає в написанні тестів які виконують код існуючих класів та перевіряють фактичний результат з очікуваним. Тест вважається успішним у випадку коли збігається очікуваний та фактичний результат виконання функції тестування. Якість системи виміряється у відсотковому відношенні покритого коду тестами до загальної кількості коду. Приклад чистини модульного тестування наведено на рисунку 3.9.

```
@RunWith(MockitoJUnitRunner.class)
public class ProductControllerTest {

    @InjectMocks
    private ProductController productController;

    @Mock
    private ProductService productService;

    @Test
    public void testAddProduct() {
        ProductDTO productDTO = new ProductDTO();
        Long subcategoryId = 1L;
        Long productId = 1L;
        when(productService.addProduct(productDTO, subcategoryId)).thenReturn(productId);
        ResponseEntity<Long> response = productController.addProduct(productDTO, subcategoryId);
        assertEquals(HttpStatus.OK, response.getStatusCode());
        assertEquals(productId, response.getBody());
    }
}
```

Рисунок 3.9 – Клас тестування контролеру взаємодії з товарами

Даний клас має метод для створення товару потім додає його в системи та перевіряє стан відповіді системи з очікуваним позитивним результатом за допомогою бібліотеки тестування Mockito [14].

Другим видом тестування системи було вирішено обрати навантажувальне тестування, за допомогою якого перевіряється здатність програми стабільно працювати при великої кількості користувачів. Інструментом такого виду тестування було обрано Apache JMeter, він надає можливість конфігурувати навантаження та виводити результат у різних видах. Параметри навантаження майбутнього тестування наведено на рисунку 3.10.

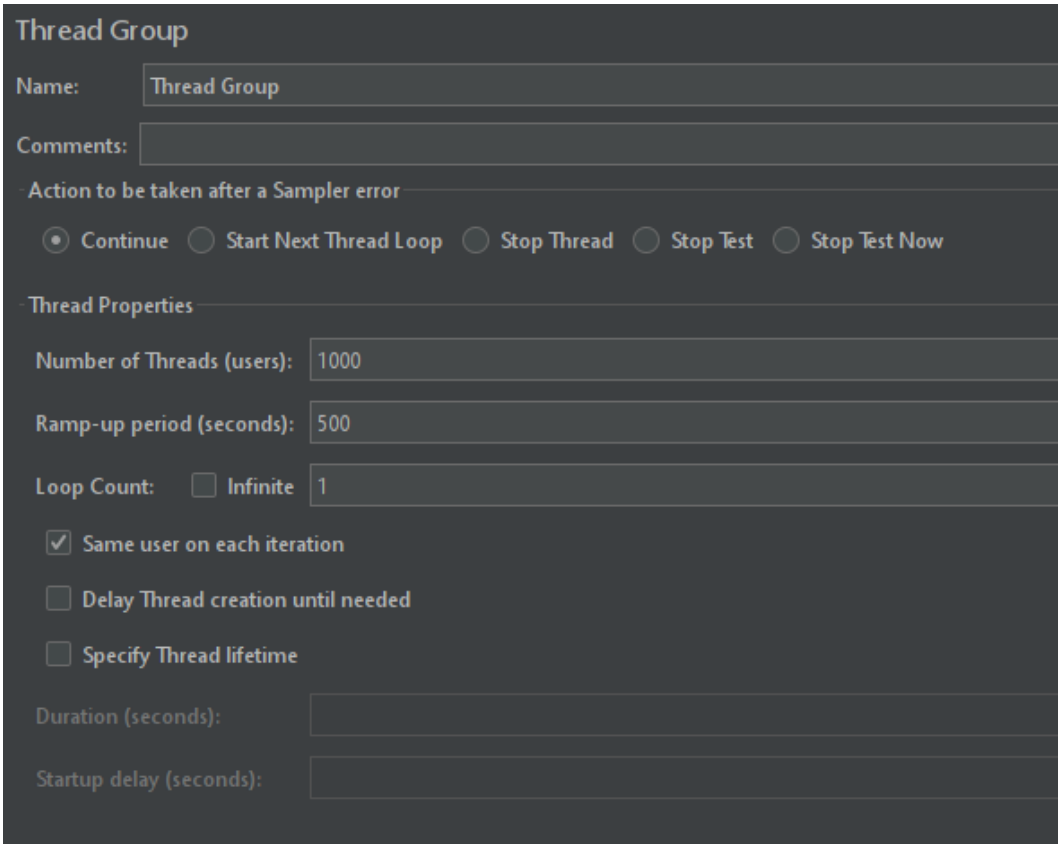


Рисунок 3.10 – Параметри навантаження

Наведений рисунок показує що програма зробить модель ситуації при якій 1000 користувачів будуть робити запити протягом 500 секунд.

Додаткова конфігурація наведена на рисунку 3.11.



Рисунок 3.11 – Конфігурація шляху, протоколу та порту

Результат виконання навантажувального тестування наведено на рисунку 3.12.

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received K...	Sent KB...
HTTP Request	1858	14	9	79	3.55	0.00%	2.5/sec	5.00	0.30
TOTAL	1858	14	9	79	3.55	0.00%	2.5/sec	5.00	0.30

Рисунок 3.12 – Результат навантаження на систему згідно конфігурації

Середній час виконання запиту (Average): середній час виконання одного запиту складає 14 мілісекунд. Це показник ефективності системи при даному навантаженні.

Мінімальний та максимальний час виконання запиту Min та Max. Мінімальний час виконання запиту складає 9 мілісекунд, а максимальний – 79 мілісекунди. Ці значення показують розкид у часі виконання запитів, що може вказувати на різноманітність відповідей від сервера.

Стандартне відхилення Std.Dev. Стандартне відхилення часу виконання запиту складає 3.55 мілісекунди. Це міра розкиду значень відносно середнього часу виконання. Велике стандартне відхилення може вказувати на нестабільність продуктивності.

Відсоток помилок Error – відсоток помилок дорівнює 0%, що вказує на відсутність помилок під час тестування.

Пропускна здатність Throughput – пропускна здатність становить 2.5 запитів в секунду. Це кількість запитів, які система може обробляти за одиницю часу. Більша пропускна здатність означає більшу продуктивність.

Середня кількість даних (Received KB/sec та Sent KB/sec). У даному

випадку система отримує приблизно 5 кілобайтів даних на секунду та відправляє приблизно 0.3 кілобайтів даних на секунду.

Як зазначалося раніше, дана програма надає можливість представляти результат у різних форматах, наприклад у вигляді графіку, який було зображено на рисунку 3.13.

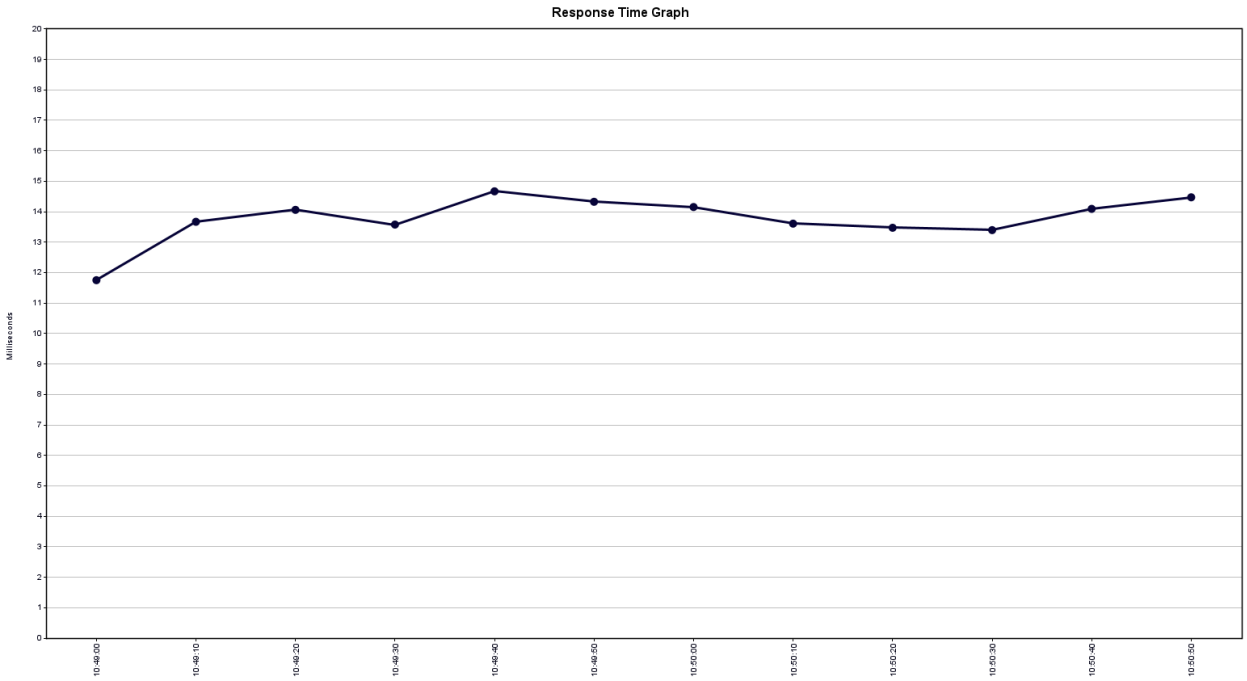


Рисунок 3.13 – Графік часу відповіді серверу

Графік візуалізує час отримання відповіді на запити, та показує загалом стабільну роботу серверу.

3.4 Тестування клієнтської частини системи

Аналогічно до тестування серверної частини для клієнтської частини існує різноманітна кількість інструментів та видів тестування для забезпечення якісної та безперешкодної роботи розроблюваної платформи. Деякі з існуючих видів тестування - тестування якості відображення елементів, наявність стандартів розробки інтерфейсу користувача, тестування локалізації

при якому перевіряються забезпечення надання функціоналу згідно місця використання програми, кросбраузерне тестування яке перевіряє ідентичну роботу застосунку в різних браузерах або тестування безпеки під час якого перевіряється наявність доступу різних видів користувачів до службової інформації.

Для проведення тестування та знаходження дефектів дуже часто використовують чек-листи, які зазвичай представляють собою таблиці, де кожен рядок відповідає певному аспекту або функціоналу програми, а кожен стовпчик відображає результат тестування для певного тест-кейсу або критерію. Такі критерій відносяться до категорій функціональності системи, сумісності з різними пристроями, безпекою, документацією та різними іншими. Під час тестування кожен пункт чек-листа перевіряється, а результат, наприклад “passed”, “failed” або “require to check” фіксується. Це допомагає тестувальникам не пропустити важливі аспекти програми та ефективно виявляти та виправляти дефекти.

Прикладом такого чек листа для інформаційної системи інтернет-магазину наведено на рисунку 3.14.

	Сайт http://localhost:8086	*Chrome 124.0.63*	*FireFox 125.03*	*Micrsoft 124.04*	*Примітки/Посилання на баг-репорт*
1	**Наявність елементів сторінок**				
	Наявний футер	passed	passed	passed	
	Наявна шапка сайту	passed	passed	passed	
	Наявна відображення breadcrumb trails	failed	failed	failed	
2.	**Функціональність сайту**				
	Кожен елемент є клікабельним	passed	passed	passed	
	Кожна кнопка виконує очікуваний функціонал	passed	passed	passed	
	Коректно додаються товари до кошику	passed	passed	passed	
	Перевірити валідацію попів	passed	passed	passed	
	Перевірити роботу пошуку за назвами товарів	passed	passed	passed	
	Перевірити перехід за підказками у випадаючому списку поля пошуку після введення ключового слова	failed	failed	failed	
	Перевірити роботу пошуку по різноманітним параметрам	passed	passed	passed	
	Перевірити сортування товарів за назвою	passed	passed	passed	
	Перевірити фільтрування за категоріями	passed	passed	passed	

Рисунок 3.14 – Чек-лист клієнтської програми

Чек-лист зображує стан системи згідно різним аспектам тестування у різних браузерах. Можна зазначити що не всі пункти мають статус “passed”, що означає про наявність дефектів та відсутності деяких функцій, що потребує подальшого виправлення.

3.5 Висновки до розділу 3

У даному розділі було розглянуто процес реалізації та тестування інформаційної системи інтернет-магазину, як з боку серверної, так і з боку клієнтської частини.

Було детально описано створення основного функціоналу серверної частини, включаючи визначення моделі товару, реалізацію класів для роботи з товарами, використання шаблону проєктування "Фабричний метод" для гнучкості та можливості масштабування коду. Наведено приклади коду для моделей, сервісів та фабричних методів, що забезпечують створення різних видів товарів. Було проведено модульне тестування з використанням бібліотеки Mockito. Це дозволило перевірити коректність роботи методів контролеру взаємодії з товарами, підтверджуючи, що фактичні результати відповідають очікуваним. Проведено навантажувальне тестування за допомогою Apache JMeter, яке продемонструвало здатність системи стабільно працювати під значним навантаженням. Проаналізовано результати тестування, такі як середній час виконання запитів, мінімальний та максимальний час, стандартне відхилення, відсоток помилок та пропускна здатність.

Описано розробку користувацького інтерфейсу та функціоналу клієнтської частини. Наведено приклади шаблонів та програмного коду для реалізації інтерфейсу. Розглянуто важливість зручного, інтуїтивно зрозумілого та привабливого інтерфейсу для забезпечення позитивного користувацького досвіду. Розглянуто різні види тестування клієнтської частини, такі як тестування якості відображення елементів, стандартів розробки інтерфейсу, локалізації, кросбраузерне тестування та тестування безпеки. Наведено приклад чек-листа для тестування клієнтської частини, який допомагає систематично перевіряти різні аспекти програми та виявляти дефекти.

Загалом, реалізація та тестування інформаційної системи інтернет-

магазину охоплюють різні аспекти, що забезпечують її стабільну та ефективну роботу. Використання сучасних підходів та інструментів дозволило створити масштабовану, гнучку та надійну платформу, яка відповідає вимогам сучасного ринку.

ВИСНОВКИ

У кваліфікаційній роботі було розглянуто розвиток електронної комерції, її інструменти та обсяги які зростають з кожним роком. Було проведено аналіз предметної області, розглянуто деякі статистичні дані та визначено ряд функціональниї вимог які повинна задовольняти майбутня система. Проаналізовано та обґрунтовано вибір одних з найпопулярніших технологій розробки, які довели свою актуальність та зручніть у створені потужних інформаційних систем для ведення електронної комерції через інтернет, шляхом розробки інтернет-магазину.

На основі визначених вимог до інформаційної системи було розроблено модель системи за допомогою діаграм, кожна з якою демонструє різний аспект роботи програми, показує вимоги або відображає структуру майбутнього застосунку.

Останнім кроком було реалізовано вебзастосунок на основі інструментарію, проаналізованих на попередніх етапах кваліфікаційної роботи, та проведено навантажувальне тестування за допомогою інструменту JMeter.

Результати розробки показали сьогоднішню актуальність архітектури клієнт-сервер, яка забезпечує стабільну роботу при досить великому навантаженні.

Отже, з врахуванням стрімкого росту електронної комерції та постійного попиту на ефективні інструменти, розробка інтернет-магазинів залишається актуальною та перспективною галуззю. Результати даної кваліфікаційної роботи свідчать про важливість вибору правильних технологій та архітектурних рішень для забезпечення надійності та швидкодії додатків у сучасному цифровому середовищі.

ПЕРЕЛІК ПОСИЛАНЬ

1. Показники зростання обсягів електронної комерції. Digital Commerce. URL: https://www.digitalcommerce360.com/2023/01/24/consumers-in-control-of-conversion/ (дата звернення: 15.01.2024).
2. Міністерство економіки України. Визначення електронної комерції та прийняті декларації. URL: https://www.me.gov.ua/Documents/Detail?lang=uk-UA&i=6d3d0ba1-219b-470a-a9b9-a88f7dbf02ec&title=ElektronnaKomertsiia&isSpecial=true (дата звернення: 16.01.2024).
3. Затвердженні види електронної комерції. URL: https://voll.com.ua/uk/blog/e-commerce-yak-pracyuye-elektronna-komerciya (дата звернення: 16.01.2024).
4. Статистичні дані показників електронної комерції. URL: https://www.statista.com/markets/413/e-commerce/ (дата звернення: 14.02.2024).
5. Бергер А., Галета А. Світові тенденції розвитку електронної комерції з урахуванням кризових умов пандемії COVID-19. *Економіка та суспільство*. 2021. № 26. URL: https://doi.org/10.32782/2524-0072/2021-26-18 (дата звернення: 20.02.2024).
6. Discover DHL. E-commerce during COVID-19. URL: https://www.dhl.com/discover/en-global/logistics-advice/logistics-insights/what-impact-has-covid-19-had-on-consumer-returns (дата звернення: 25.02.2024).
7. UML Use Case Diagram Tutorial. URL: https://www.lucidchart.com/pages/uml-use-case-diagram (дата звернення: 20.02.2024).
8. Date C. J. An Introduction to Database Systems. New York, London, 2005. 1336 p.

9. ER Diagrams in DBMS: Entity Relationship Diagram Model. URL: https://www.simplilearn.com/tutorials/sql-tutorial/er-diagram-in-dbms#:~:text=An%20Entity (дата звернення: 23.03.2024).
10. UML Class Diagram Tutorial. URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/ (дата звернення: 23.03.2024).
11. PostgreSQL: a Practical Guide-Features and Advantages. URL: https://www.linkedin.com/pulse/postgresql-practical-guidefeatures-advantages-brainerhub-solutions/ (дата звернення: 23.03.2024).
12. Введення в Vue.js. Документація та гайди. URL: https://vuejs.org/guide/introduction.html (дата звернення: 25.03.2024).
13. Craig S. W. Spring in Action Sixth Edition. Castle Rock, CO, 2022. 546 p.
14. Документація використання бібліотеки Mockito. URL: https://javadoc.io/doc/org.mockito/mockito-core/latest/org/mockito/Mockito.html (дата звернення: 26.03.2024).

ДОДАТОК А

Клас для обробки запитів ProductController

Через великий обсяг коду в класі-контролері для обробки запитів пов’язаних з товарами, для підвищення читаємості коду було прийнято рішення відформатувати його та зобразити рисунком як показано на рисунку А.1.



```
@RestController
@CrossOrigin(origins = "http://localhost:8080", methods = {RequestMethod.POST, RequestMethod.GET,
RequestMethod.OPTIONS})

@RequestMapping("/products")
public class ProductController {

    @Autowired
    private ProductService productService;

    @RequestMapping(value = "/add", method = RequestMethod.POST)
    public ResponseEntity<Long> addProduct(@RequestBody ProductDTO ProductDTO,
@RequestParam("subcategoryId") Long subcategoryId) {
        Long product_id = productService.addProduct(ProductDTO,subcategoryId);
        return ResponseEntity.ok().body(product_id);
    }
    @RequestMapping(value = "/delete/{id}",method = RequestMethod.POST)
    public void deleteProduct(@PathVariable("id") Long id) {
        this.productService.deleteProductById(id);
    }
    @RequestMapping(value = "/id/{id}",method = RequestMethod.GET)
    public Option findProductById(@PathVariable("id")Long id) {
        return this.productService.getProductWithCorrectTypeById(id);
    }
    @RequestMapping(value = "/getRatings/{productId}",method = RequestMethod.GET)
    public List<ProductRating> getAllRatingProductById(@PathVariable("productId")Long productId) {
        return productService.getRatingsProductByProductId(productId);
    }
    @RequestMapping(value = "/setRatings/{productId}/{userEmail}/{rating}",method = RequestMethod.POST)
    public void getAllRatingProductById(@PathVariable("productId")Long
productId,@PathVariable("userEmail")String userEmail,@PathVariable("rating")Integer rating) {
        productService.setRatingToProdyctByProductIdAnnUserId(productId, userEmail,rating);
    }
    @RequestMapping(value = "/{name}",method = RequestMethod.GET)
    public List<Product> findProductByName(@PathVariable("name")String name) {
        return productService.findByName(name);
    }
    @RequestMapping(value = "/uploadImages",method = RequestMethod.POST)
    public void handleFileUpload(@RequestParam("photos") List<MultipartFile>
photos,@RequestParam("product_id") Long product_id,@RequestParam("mainImageIndex") Long mainImageIndex)
throws IOException {
        productService.uploadImages(photos,product_id,mainImageIndex);
    }
    @RequestMapping(value = "/brands/unique/{limit}",method = RequestMethod.GET)
    public List<String> getUniqueBrands(@PathVariable("limit") Integer limit){
        return  productService.getUniqueBrands(limit);
    }
}
```

Рисунок А.1 – Код класу ProductController

ДОДАТОК Б

Компонент для забезпечення функціоналу кошику магазину

Функціонал корзини товарів був розроблений на стороні клієнтської частини програми, тому через використання фреймворку Vue який використовує в компоненті різні секцій для створення різних аспектів компонента, код був розділений на візуальний шаблон та функкціональні скрипти та наведений на рисунках Б.1 та Б.2 відповідно.

```
<template>
  <div id="appCart">
    <page-header></page-header>
    <div id="content">
        <div>
            <h2>Корзина</h2>
            <h3 v-if = "checkCartIsEmpty">
            {{ cartIsEmptyMessage }}
            </h3>
            <div id="productsPanel">
              <div class="productPanelItem" v-for="(item, index) in cartItems" :key="index">
                  <img :src="item.displayImageURL" alt="Фото товара" class="productImage" />
                  <div class="productInfo">
                      <span class="productPanelItemName">{{ item.productName}}</span>
                      <span class="productPanelItemPrice">{{ item.price * item.amountInOrder }}  ₴</span>
                  </div>
                  <input type="number" class = "amountOfCurrentProducts"  step="1" min="1" max="20" value
                      = "1" v-model ="cartItems[index].amountInOrder">
                  <button @click="removeFromCart(index)" class="removeButton">Видалити</button>
              </div>
          </div>
          <div id = "additionalPanel" v-if = "!checkCartIsEmpty">
          <div id = "TotalPrice">
            <h3>Сума: {{ total }} ₴</h3>
          </div>
          <div id = "createOrderPanel">
            <router-link :to="{ path: '/order', query: { orderedProducts:JSON.stringify(this.cartItems)
                }}">
            <button id="createOrderBtn">Оформити замовлення</button>
            </router-link>
          </div>
          </div>
      </div>
    </div>
    <page-footer></page-footer>
  </div>
</template>
<script>
```

Рисунок Б.1 – HTML-шаблон кошику товарів

Шаблон демонструє сторінку корзини з повним функіоналом видалення з корзини, відображенням загальної суми та можливістю перейти до сторінки замовлення.



```
<script>
import PageHeader from '../../PageHeader.vue';
import PageFooter from '../../PageFooter.vue';

export default {
  name: 'ProductCart',
  components: {
    PageHeader,
    PageFooter,
  },
    data() {
        return {
        cartItems: [],
        cartIsEmptyMessage:"Корзина порожня",
        };
    },
    computed: {
        total() {
            if (Array.isArray(this.cartItems)) {
                return this.cartItems.reduce((acc, item) => acc + item.price*item.amountInOrder, 0);
            }
            return 0;
        },
        checkCartIsEmpty(){
          return this.cartItems.length === 0;
        }
    },
    mounted() {
        this.loadCartItems();
    },
    methods: {
        removeFromCart(index) {
          this.$store.dispatch('REMOVE_FROM_CART',index)
        },
        async loadCartItems() {
        this.$store.dispatch('loadCartFromStorage').then(() => {
              this.cartItems = this.$store.getters.cartItems;
              if (this.cartItems.length>0) {
                this.cartItems.forEach(function(item){
                  item.amountInOrder = 1;
            });
          }
        }).catch((error) => {
          console.error('Error loading cart:', error);
        });
        },
    },
};

</script>
```

Рисунок Б.2 – Секція скриптів компонента кошику товарів

Дана секція містить в собі усі методи для забезпечення шаблону наповненням даними та коректної взаємодії сторінки кошику з користувачем.

ДОДАТОК В

Клієнтський компонент інтерфейсу для авторизації AUTH PAGE

За аналогією до додатка Б код компонента для авторизації у системі буде розділено на 2 частини шаблон та секція скриптів які наведені на рисунках В.1 та В.2 відповідно.



```
<template>
    <div class="signup">
  <div class="signup-connect">
    <h1>Sign in</h1>
    <a href="#" class="btn btn-social btn-facebook"> <i class="fab fa-facebook"></i> Sign in with
Facebook</a>
    <a href="#" class="btn btn-social btn-twitter"> <i class="fab fa-twitter"></i> Sign in with
Twitter</a>
    <a href="#" class="btn btn-social btn-google"> <i class="fab fa-google"></i> Sign in with Google</a>
  </div>
  <div class="signup-classic">
    <h1 id = "LoginField">Login</h1>
    <form class="form" @submit.prevent="login">

      <fieldset class="email">
        <input type="text" placeholder="email"  v-model="email" autocomplete="email"/>
      </fieldset>
      <div v-if="email_error" class="error-message">{{ email_error}}</div>
      <fieldset class="password">
        <input type="password" placeholder="password" v-model="password"  autocomplete="current-
password"/>
      </fieldset>

      <div v-if="auth_error" class="error-message">{{ auth_error}}</div>
      <div v-if="input_error" class="error-message">{{ input_error}}</div>
      <button type = "submit" class="btn">log in</button>
       <a href="/registration">Registration</a>
    </form>
  </div>
</div>
</template>
```

Рисунок В.1 – HTML-шаблон компонента авторизації

Даний шаблон зображує головну форму авторизації та декілька посилань на реєстрацію або авторизацію через сторонні сервіси такі як Google або Facebook.



```
<script>
export default {
  data() {
    return {
      email: '',
      password: '',
      auth_error:'',
      input_error:'',
      email_error:'',
    };
  },
  methods: {
    async login() {
      this.email_error = '';
      this.input_error = '';
      this.auth_error = '';

    if (!this.email || !this.email.trim() || !this.password || !this.password.trim()) {
        this.input_error = 'Field can not be empty';
        return;
    }
    const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
      if (!emailRegex.test(this.email)) {
          this.email_error = 'Follow @mail format';
          return;
      }
      try {
          await this.$store.dispatch('authenticate', { email: this.email, password: this.password });
      } catch (error) {
          this.auth_error = 'Invalid email or password';
          console.error('Authentication error:', error);
          return;
      }
      if(this.$store.getters.authError){
        this.auth_error = this.$store.getters.authError;
        return;
      }
      this.$router.push('/home');
    }
}
}
</script>
```

Рисунок В.2 – Секція методів для компонента авторизації

Даний набір методів дозволяє зробити валідацію даних уведених користувачем та повідомити його про пимилки або авторизувати у системі.