МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет

Факультет електроніки та інформаційних технологій

Кафедра комп'ютерних наук

«До захисту допущено»

В.о. завідувача кафедри

_____ Ігор ШЕЛЕХОВ

(підпис)

28 травня 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавр

зі спеціальності 122 – Комп'ютерних наук,

освітньо-професійної програми «Інформатика»

на тему: «Інтернет-магазин спортивного одягу»

здобувача групи ІН-06-2 Пінчука Андрія Вікторовича

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Андрій ПІНЧУК

(підпис)

Керівник,

асистент кафедри комп'ютерних наук

кандидат фізико-математичних наук Олександр ВЛАСЕНКО _____

(підпис)

Суми – 2024

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавра

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувача групи ІН-06-2 Пінчука Андрія Вікторовича

1. Тема роботи: «Інтернет-магазин спортивного одягу» затверджую наказом по СумДУ від 22 квітня 2024 р. № 0414-VI
2. Термін здачі здобувачем кваліфікаційної роботи 28 травня 2024 р.
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Аналіз проблеми предметної області, постановка й формування завдань дослідження. 2) Огляд технологій, що використовуються для розробки інтернет-магазинів. 3) Розробка інтернет-магазину спортивного одягу. 4) Тестування і аналіз результатів.*
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____
6. Консультанти до проєкту (роботи), із значенням розділів проєкту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «08» квітня 2024 р.

Завдання прийняв до виконання _____ Керівник _____
(підпис) (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз проблеми предметної області, постановка й формування завдань дослідження*		
2	*Огляд технологій, що використовуються для розробки інтернет-магазинів*		
3	*Розробка інтернет-магазину спортивного одягу*		
4	*Тестування і аналіз отриманих результатів*		
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*		

Здобувач вищої освіти _____ Керівник _____
(підпис) (підпис)

АНОТАЦІЯ

Записка: 64 стр., 42 рис., 0 додатків, 19 використаних джерел.

Обґрунтування актуальності теми роботи – Тема кваліфікаційної роботи є актуальною, оскільки присвячена розв’язанню важливої практичної задачі спрощення бізнес-процесів магазинів шляхом розробки відповідних методів, моделей та інформаційної технології.

Об’єкт дослідження — продаж спортивного одягу в мережі інтернет.

Мета роботи — розробка інтернет-магазину спортивного одягу з можливістю адміністрування.

Методи дослідження — аналіз існуючих аналогів розроблюваної системи, порівняння та вивчення технологій та методів розробки інтернет-магазинів.

Результати — розроблено інтернет-магазин спортивного одягу для пошуку та замовлення товарів, адміністративну панель для адміністрування магазину, серверну частину для ізоляції бізнес-процесів та забезпечення безпеки системи та користувачів. Проведено тестування серверної частини з використанням різних методів автоматизованого тестування.

ІНФОРМАЦІЙНА СИСТЕМА, ЕЛЕКТРОННА КОМЕРЦІЯ, ІНТЕРНЕТ-МАГАЗИН, RESTFUL API, ASTRO, REACT, NEXTJS, JAVA, SPRING FRAMEWORK, POSTGRESQL.

ЗМІСТ

ВСТУП

У сучасному інформаційному суспільстві використання лише комп'ютерних технологій означає просто відтворення даних, не забезпечуючи ефективної взаємодії з інформацією. Це питання вирішується шляхом використання передових інформаційних технологій. Використання цих технологій у таких галузях, як продаж спортивного одягу, спрощує процес управління та дозволяє зосередити зусилля на зміцненні слабких ланок компанії.

Інформаційні технології надають бізнесу можливості краще зрозуміти потреби клієнтів і швидко реагувати на зміни. Це важливо, оскільки традиційні методи дослідження ринку можуть бути повільними та неточними. Крім того, використання інформаційних технологій може пришвидшити вихід нових продуктів на ринок.

Актуальність. Інтернет-магазини можуть допомогти бізнесу отримати конкурентні переваги за рахунок залучення нових клієнтів та аналізу статистики продажів. Власники бізнесу зацікавлені у створенні власних інтернет-магазинів, так як це дозволяє частково автоматизувати бізнес-процеси, допомагає залучати нових клієнтів і утримувати існуючих, а також підвищує ефективність роботи, зменшує витрати та прискорює виробничі процеси. У цьому контексті, розробка інтернет-магазину спортивного одягу є необхідним етапом розширення бізнесу та оптимізації ресурсів, що важливо для збереження конкурентоспроможності та підтримці діяльності бізнесу.

Об’єкт дослідження. Процес розробки веб-застосунку, що являє собою інтернет магазин та адміністративну панель.

Предмет дослідження. Методи та засоби для розробки та підтримки інтернет-магазинів спортивного одягу.

Гіпотеза. Забезпечення конкурентоспроможності бізнесу можливе завдяки використанню передових технологій для автоматизації бізнес-процесів.

Наукова новизна. Запропонована система надає можливість автоматизувати бізнес-процеси, враховуючи специфічні вимоги бізнесу з продажу спортивного одягу.

Структура. Дана робота складається зі вступу, аналітичного огляду, постановки задачі, вибір методу розв’язання поставленої задачі, опису програмного забезпечення інформаційної системи, висновків, списку використаних джерел та додатків.

1 АНАЛІТИЧНИЙ ОГЛЯД

1.1 Аналіз актуальності

Інформаційні системи відіграють важливу роль у розвитку різноманітних галузей економіки, в тому числі вони є невід'ємною частиною сучасного бізнесу. У сучасному цифровому світі, де електронна комерція набуває все більшого значення, створення та ефективне управління інформаційними системами інтернет-магазинів стає особливо актуальним завданням [1].

Стрімке поширення електронної комерції призвело до значних змін у способах ведення бізнесу та взаємодії з клієнтами. Оскільки все більше споживачів обирають покупки в Інтернеті, підприємства визнають необхідність впровадження надійних інформаційних систем для ефективного управління та оптимізації своїх операцій.

Поява інтернет-магазинів відкрила нові можливості для підприємців та вже існуючих компаній. Однак інформаційна система інтернет-магазину є складним комплексом програмних, апаратних та людських ресурсів, яка забезпечує автоматизацію бізнес-процесів, таких як управління товарами, замовленнями, складським обліком, обслуговуванням клієнтів тощо [2].

Інтернет-магазини дозволяють забезпечити зручний та швидкий доступ клієнтів до інформації про товари, послуги та умови придбання. Клієнти можуть зручно шукати, порівнювати та замовляти товари без необхідності відвідувати фізичний магазин [3]. Інформаційні системи дозволяють магазинам підтримувати актуальну інформацію про товари, надавати зручні способи оплати та доставки, а також надають можливість автоматизовано формувати необхідні звіти для отримання актуальних даних для покращення якості обслуговування та збільшення прибутків.

Оскільки глобальний доступ до Інтернету та його впровадження стрімко зростають, кількість людей, які здійснюють покупки в Інтернеті, постійно зростає. У 2022 році роздрібні продажі електронної комерції в усьому світі

перевищили 5,7 трильйона доларів США, і очікується, що ця цифра досягне нових висот у найближчі роки [4].

У сфері послуг конкуренція змушує компанії постійно підвищувати якість обслуговування і скорочувати час обробки замовлень. Одним із прикладів цього є торговельно-посередницька діяльність у галузі спортивного одягу. Хоча тут велика конкуренція і спостерігається певна сезонність, попит на цей бізнес залишається високим. Інформаційні технології допомагають компаніям ефективніше вести облік та аналізувати популярність певних товарів, що дозволяє обрати правильну маркетингову стратегію та розширити клієнтську базу, що, у свою чергу, підвищує прибутковість.

Привабливий та зручний у використанні сайт може привернути увагу потенційних клієнтів. На відміну від традиційних магазинів, сайт дозволяє бізнесу охопити всю країну без значних фінансових витрат. До того ж, візуальний дизайн сайту можна оновити, не витрачаючи багато коштів.

Після створення сайту стає можливим онлайн-просування з використанням ефективних маркетингових інструментів. Це часто виявляється більш результативним, ніж звичайна реклама на вулицях. Маркетингові інструменти надають можливість просувати товари на власному ресурсі для цільових клієнтів, кому це дійсно буде цікаво. Така взаємодія з клієнтами є ключовим елементом для розвитку та розширення бізнесу.

Для клієнтів наявність сайту робить вибір і замовлення товарів набагато зручнішим. Вони можуть легко замовити бажаний продукт з будь-якого місця в країні.

Таким чином, інтернет-магазини є незамінним інструментом для підтримки ефективної діяльності та розвитку бізнесу в онлайн-середовищі. Їх застосування дозволяє магазинам підвищити конкурентоспроможність, забезпечити зручний та персоналізований сервіс для клієнтів, а також отримати цінні дані для покращення маркетингових стратегій.

1.2 Розвиток електронної комерції в світі та в Україні

Світова пандемія COVID-19 значно вплинула на розвиток міжнародної електронної комерції, стимулюючи її швидкий ріст. Очікується, що цей розвиток буде продовжуватись у наступні роки. Глобальний ринок електронної комерції виявляє потенціал для подальшого зростання, зокрема завдяки посиленому попиту на онлайн-покупки під впливом пандемії [5].

Україна також не залишається осторонь цього глобального тренду. Ринок електронної комерції в країні демонстрував стабільне зростання до початку конфлікту, проте, війна призвела до зменшення обсягів: у 2021 році обсяги склали 3506,98 млн доларів США. Проте вже у 2022 році на тлі війни обсяги зменшились майже у 12 разів і склали всього 295,85 млн доларів США. Але прогнози на майбутнє вказують на позитивні тенденції, передбачаючи відновлення та подальший розвиток ринку електронної комерції в Україні [6].

У контексті російсько-української війни, підприємства збуту товарів української електронної комерції виявили потребу у розширенні своїх каналів збуту. Електронна комерція в цьому випадку є більш вигідним та швидким способом замовлення та отримання товарів для споживачів. Важливим аспектом є також перспектива адаптації української електронної комерції до стандартів європейського ринку, що може відкрити нові можливості для підприємств [7].

Завдяки розвитку електронної комерції, українські підприємства можуть досліджувати нові ринки і залучати клієнтів з усього світу. Проте, важливо пам'ятати, що точні прогнози стосовно подальшого розвитку електронної комерції в Україні в контексті російсько-української війни складно зробити, оскільки ситуація є динамічною та залежить від багатьох факторів.

У будь-якому випадку, електронна комерція в Україні має потенціал для подальшого росту та розширення, а враховуючи світові тенденції, вона може стати важливим драйвером економічного розвитку країни.

1.3 Аналіз аналогічних проєктів

На сьогоднішній день багато магазинів речей мають власні сайти. Такі сайти допомагають привернути увагу потенційних клієнтів та надають можливості для придбання товару для вже існуючих клієнтів.

Такі сайти використовуються для презентації товарів і послуг, і їх структура зазвичай включає такі елементи, як головна сторінка з популярними товарами, каталог, сторінка клієнта, віртуальний кошик та сторінка з інформацією про продукт.

Інтернет-магазин Puma [12] – офіційний сайт американського підрозділу бренду «PUMA». На головній сторінці (рис.1.1) можна знайти навігаційне меню, інформацію про знижки, нові колекції та інші спеціальні пропозиції.

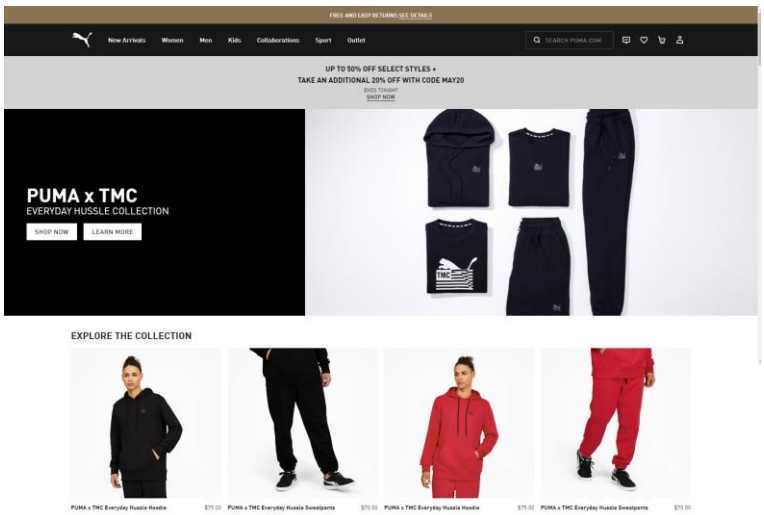

Рисунок 1.1 – Головна сторінка інтернет-магазина Puma

На сторінках каталогу (рис. 1.2) наявний пошук за допомогою фільтрів по ціні, розмірам, стилю, кольору та іншим властивостям товарів, а також по віковій групі та статі клієнта.

Також на сайті наявні такі функції як: зміна мови, список бажаних речей, кошик, чат підтримки клієнтів та власний кабінет з інформацією про замовлення.

Переваги сайту:

– зрозумілий та простий до сприйняття дизайн;

– зрозуміле навігаційне меню;

– сайт швидко реагує на дії користувача;

– є вся необхідна інформація та функції.

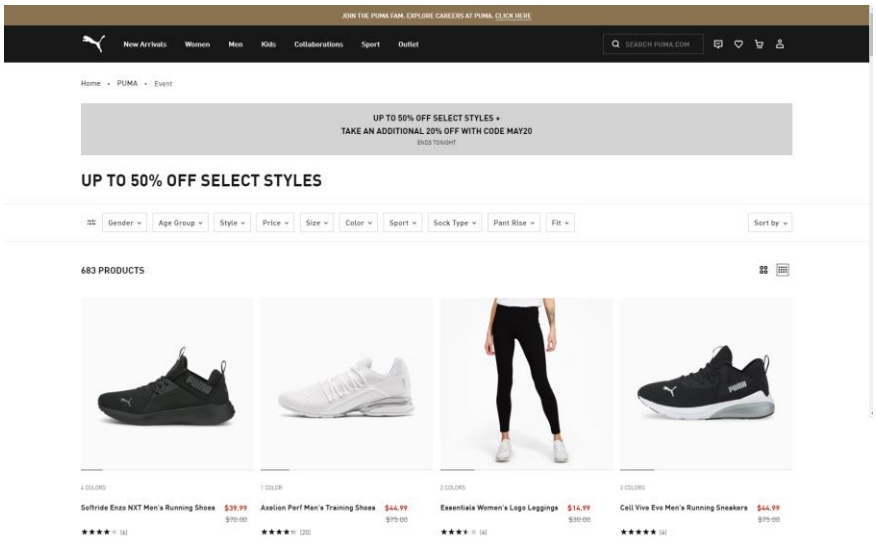

Рисунок 1.2 – Сторінка каталогу інтернет-магазина Puma

Інтернет-магазин Sportcenter [13] – сайт посередника з продажу спортивних речей популярних брендів спортивного одягу. На головній сторінці (рис. 1.3), так само, як і на офіційному сайті бренду «PUMA» можна знайти навігаційне меню, інформацію про знижки та нові колекції.

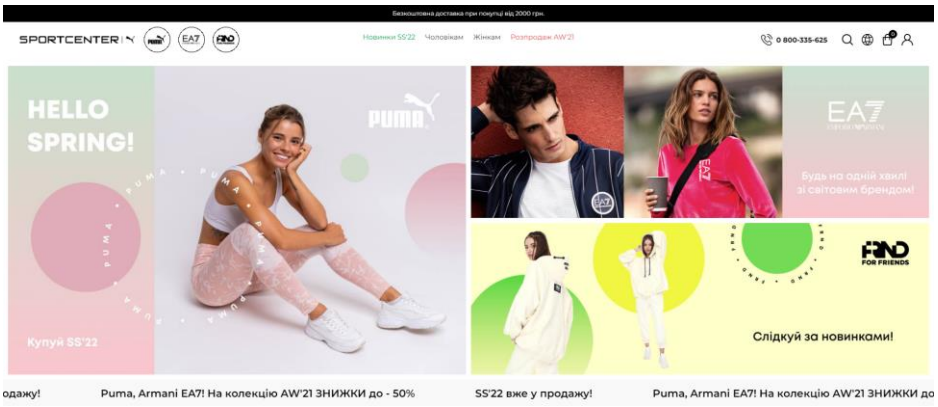

Рисунок 1.3 – Головна сторінка інтернет-магазина Sportcenter

На сторінках каталогу (рис. 1.4) наявний пошук за допомогою фільтрів по ціні, розмірам, бренду, кольору товарів та по статі клієнта.

Також на сайті наявні такі функції як: зміна мови, кошик та власний кабінет з інформацією про замовлення.

Переваги сайту:

– зрозумілий та простий до сприйняття дизайн;

– зрозуміле навігаційне меню.

Недоліки сайту:

– сайт повільно реагує на дії користувача;

– відсутня сторінка «Про нас».

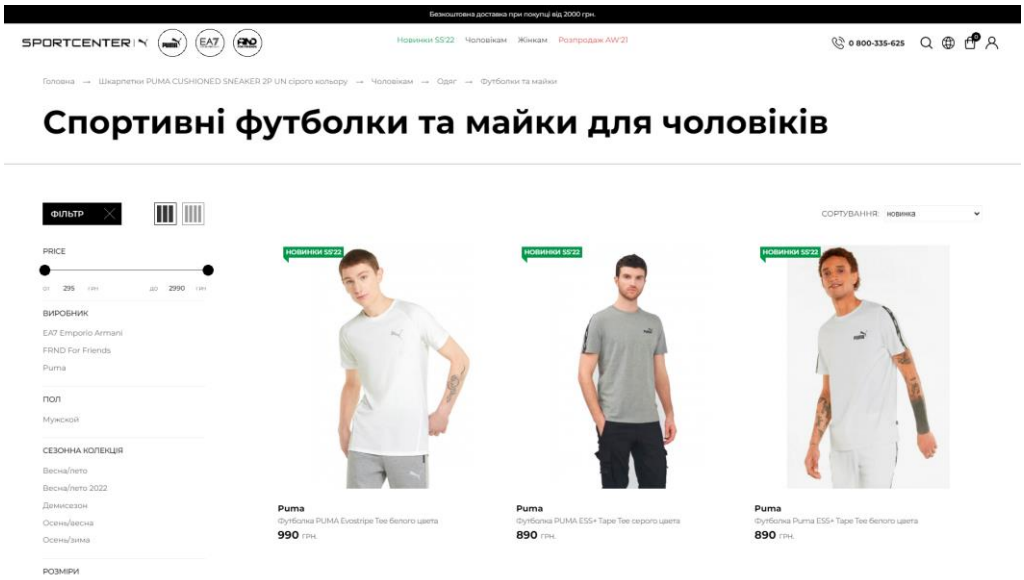

Рисунок 1.4 – Сторінка каталогу інтернет-магазина Sportcenter

Інтернет-магазин Asos [14] – сайт посередника з продажу речей багатьох всесвітньовідомих брендів. На головній сторінці (рис. 1.5) можна знайти навігаційне меню, інформацію про компанію та посилання на соціальні мережі. На сторінках каталогу (рис. 1.6) наявна інформація про розділ товарів, та пошук за допомогою фільтрів по ціні, розмірам, бренду, кольору товарів та по статі клієнта.

Також на сайті наявні такі функції як: зміна місця розташування та мови сайту, кошик та власний кабінет з інформацією про замовлення.

Переваги сайту:

– зрозумілий та простий до сприйняття дизайн;

– зрозуміле навігаційне меню;

– сайт швидко реагує на дії користувача;

– наявні всі необхідні функції та інформація.

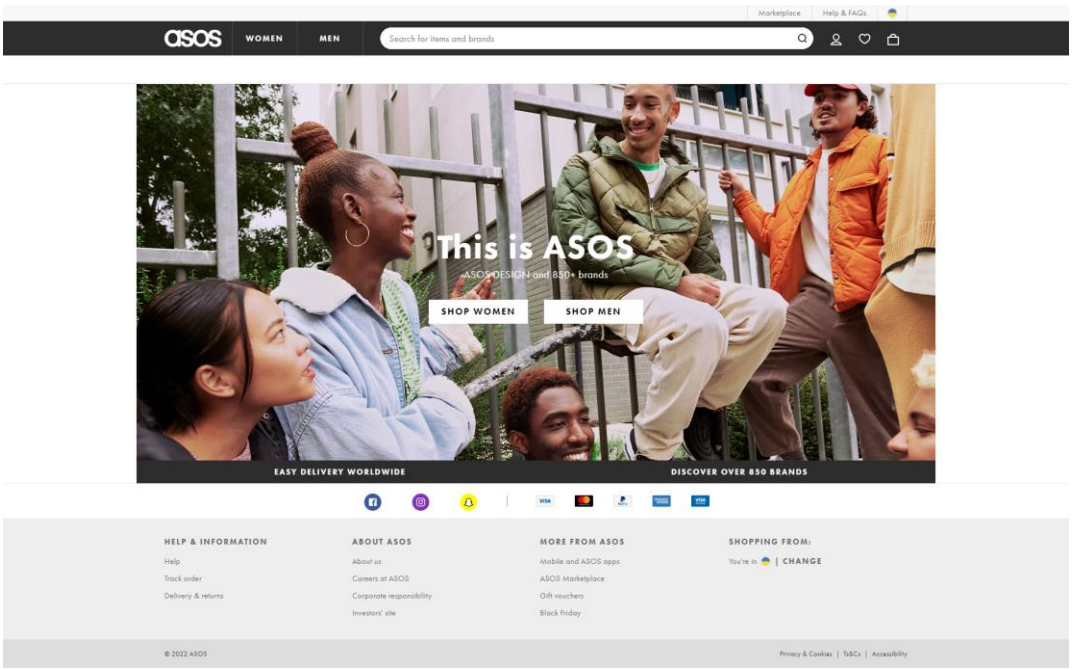

Рисунок 1.5 – Головна сторінка інтернет-магазина Asos

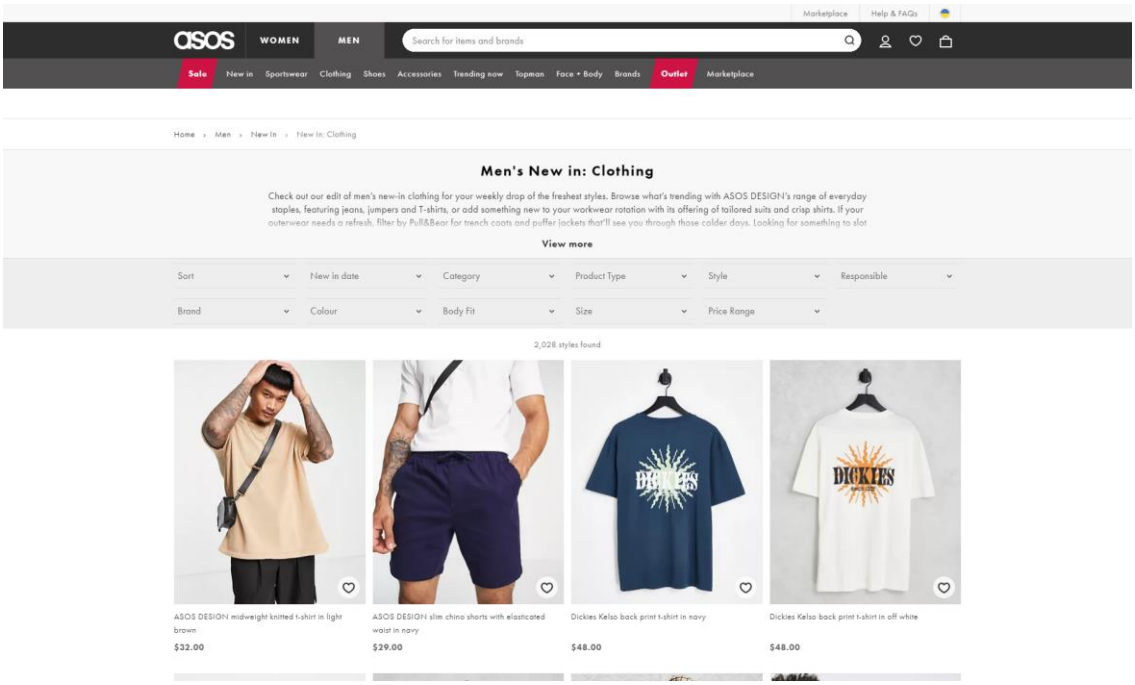

Рисунок 1.6 – Сторінка каталогу інтернет-магазина Asos

Answear [15] – інтернет-магазин одягу багатьох брендів. На головній сторінці (рис. 1.7) наявний перелік категорій, інформація про компанію та умови доставки та повернення.

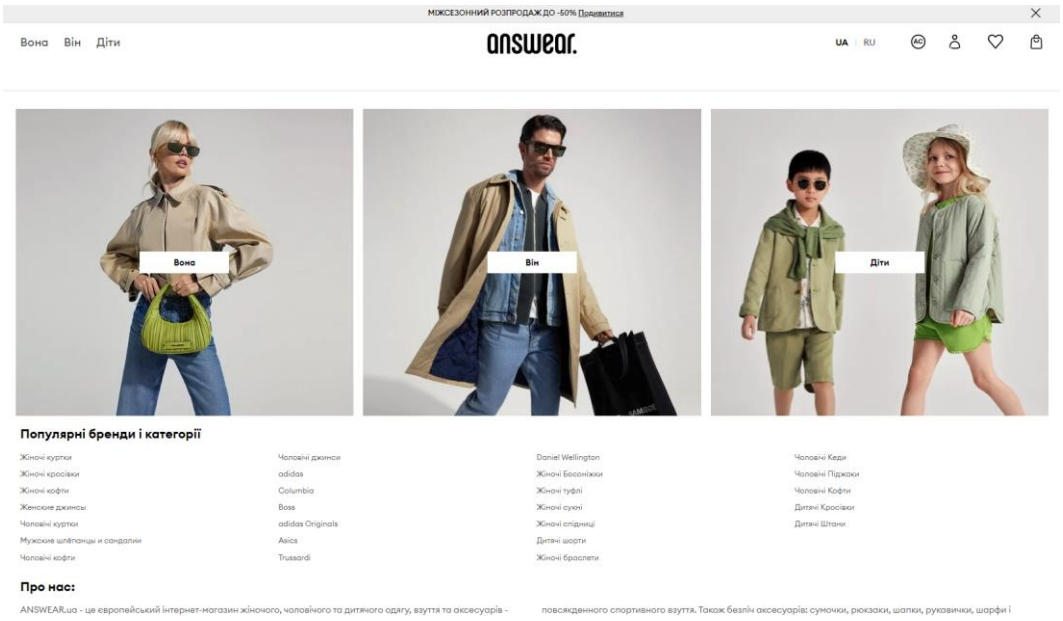

Рисунок 1.7 – Головна сторінка інтернет-магазина Answear

На сторінках каталогу (рис. 1.8) є елементи для здійснення пошуку по товарам за допомогою назви або фільтрів. Також є дерево категорій, в якому виділена поточна категорія.

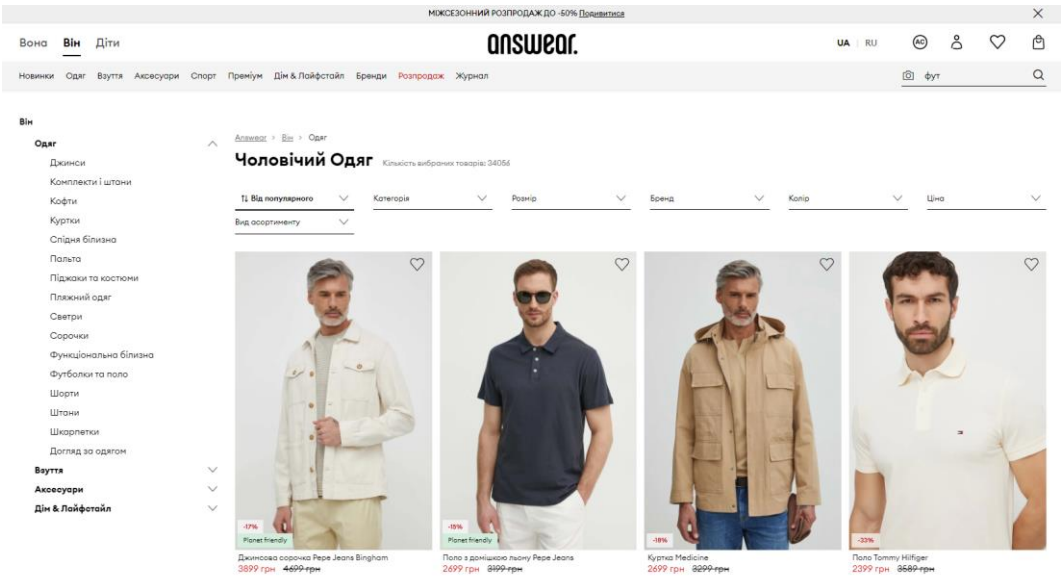

Рисунок 1.8 – Сторінка каталогу інтернет-магазина Answear

Також на сайті наявні такі функції як: зміна мови сайту, кошик, список бажаного та власний кабінет з інформацією про замовлення.

Переваги сайту:

– зрозуміла навігація по категоріям;

– сайт швидко реагує на дії користувача;

– наявні всі необхідні функції та інформація.

Недоліки:

– Необхідність реєстрації для виконання замовлення;

– Перенасиченість інтерфейсу.

Zara [15] – магазин одягу власного бренда. На головній сторінці (рис 1.9) наявне навігаційне меню, зображення товарів, посилання на соціальні мережі та посилання для отримання додаткової інформації.

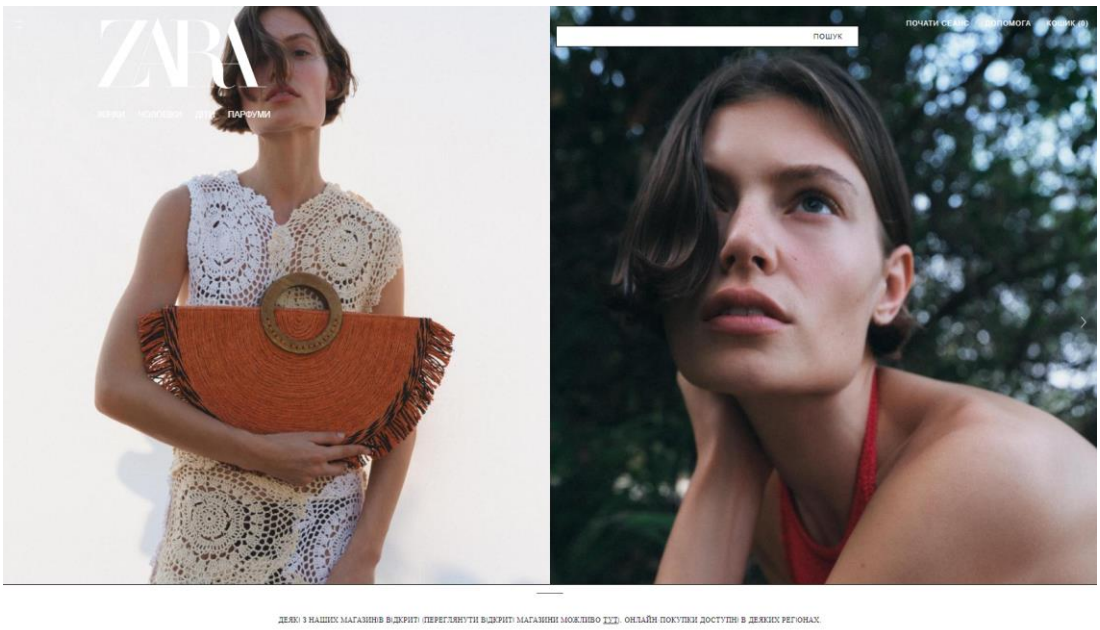

Рисунок 1.9 – Головна сторінка інтернет-магазина Answear

На сторінках каталогу (рис. 1.10) є елементи для здійснення пошуку по товарам за допомогою назви або фільтрів та інформація про категорію. Також є дерево категорій, в якому виділена поточна категорія.

Також на сайті наявні такі функції як: зміна мови сайту, кошик, список бажаного та власний кабінет з інформацією про замовлення.

Переваги сайту:

– зрозуміла навігація по сайту;

– сайт швидко реагує на дії користувача;

– наявні всі необхідні функції та інформація.

Недоліки:

– Необхідність реєстрації для виконання замовлення;

– Непродуманий інтерфейс: в деяких випадках текст зливається з фоном, елементи інтерфейсу мають різну поведінку та оформлення.

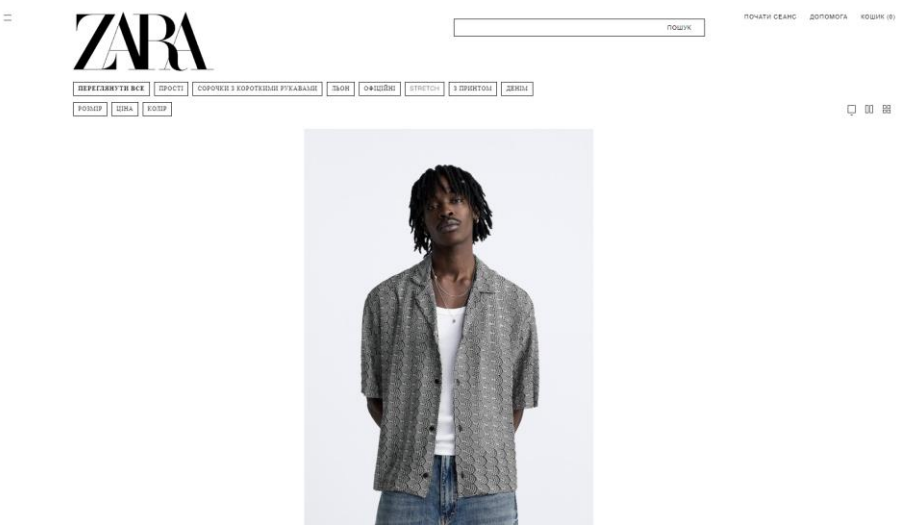

Рисунок 1.10 – Сторінка каталогу інтернет-магазина Answear

Як результат проведеного аналізу було визначено основні властивості та функції, які властиві більшості або усім аналогічним програмним продуктам. Основними властивостями є швидке завантаження сторінок та швидкий відгук сайту на дії користувача. Хоча реалізація та подання функцій різниться, серед основних функцій подібних програмних продуктів можна виокремити можливість текстового пошуку, навігацію по категоріям, можливість додавання товарів до кошику та можливість виконання замовлення товарів, що були додані до кошику.

1.4 Постановка задачі

Метою роботи є розробка інтернет-магазину з адміністративною панеллю для управління товарами.

Розроблений програмний продукт має задовольняти наступні вимоги:

- Пошуку товарів за допомогою текстового пошуку;
- Пошук товарів за категоріями;
- Забезпечення можливості вкладеності категорій;
- Можливість товарів мати варіації;

- Додавання товарів до кошику;
- Замовлення товарів;
- Перегляд історії замовлень;
- Виведення статистики замовлень та інших показників для адміністраторів;
- Можливість редагувати та додавати товари та категорії адміністраторами.

Для досягнення поставленої мети необхідно вирішити наступні задачі:

1) Розробити інформаційну модель;

2) Обрати засоби для програмної реалізації:

a) Обрати систему керування базами даних;

b) Обрати інструменти розробки серверної частини;

c) Обрати інструменти розробки клієнтської частини;

3) Програмно реалізувати інтернет-магазин спортивного одягу:

a) Спроєктувати та нормалізувати базу даних;

b) Розробити серверну частину;

c) Розробити основний сайт;

d) Розробити адміністративну панель;

4) Провести тестування розробленого програмного продукту.

Практична значимість розроблюваного програмного застосунку полягає у спрощенні взаємодії між клієнтами магазину та самим магазином, а також у спрощенні ведення статистики та аналізу продажів для адміністраторів магазину.

2 ВИБІР МЕТОДІВ РОЗВ'ЯЗАННЯ ЗАДАЧІ

2.1 Підходи до розробки серверної частини

З огляду на проведений аналіз аналогічних продуктів було розроблено інформаційну модель у вигляді діаграми варіантів використання (рис. 2.1).

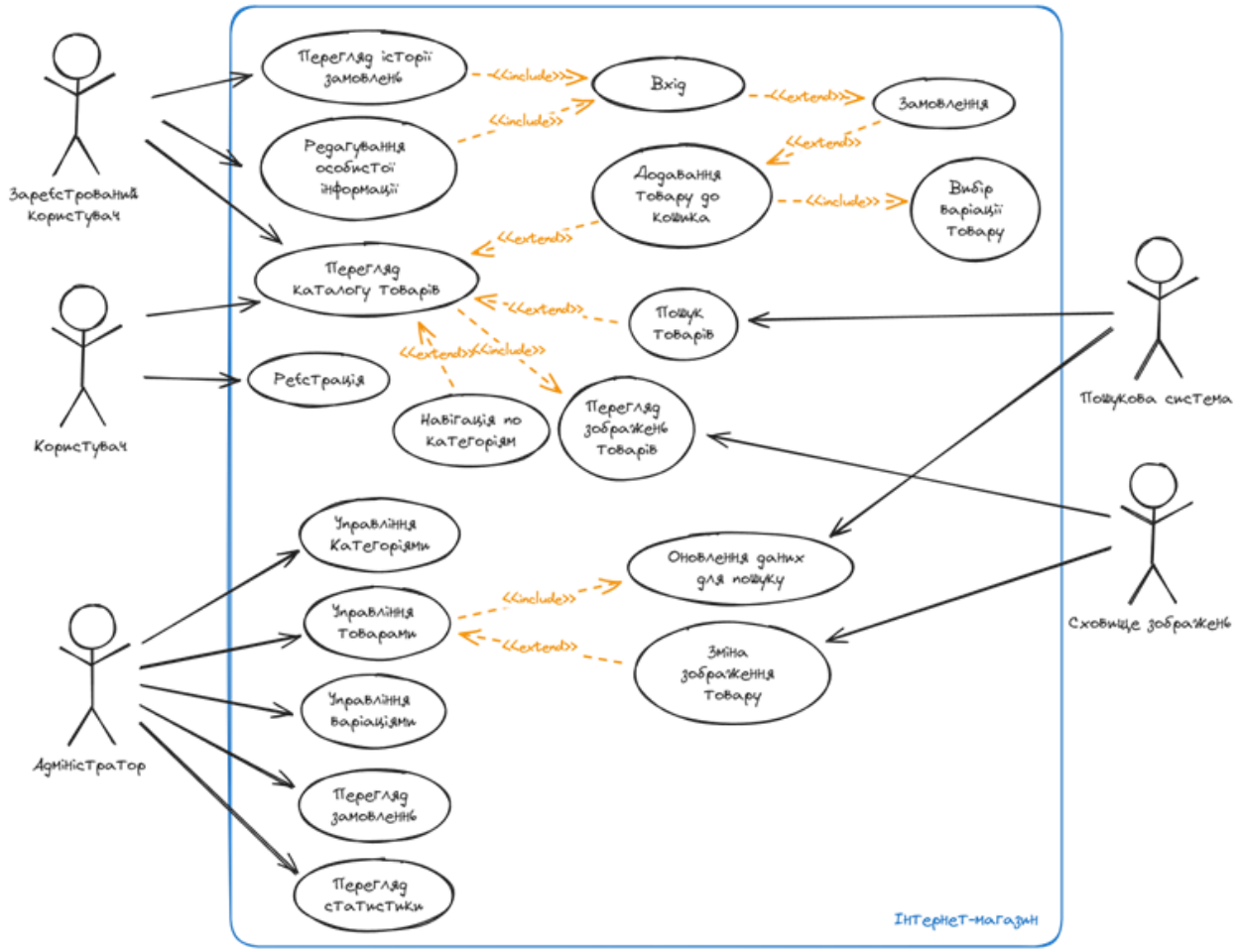

Рисунок 2.1 – Діаграма варіантів використання

Існує багато підходів для реалізації подібних систем, кожен із яких має свої унікальні особливості та переваги. Було розглянуто три основні підходи, які часто використовуються в створенні вебзастосунків та сервісів: RESTful API, SOAP та використання шаблонізаторів.

SOAP (Simple Object Access Protocol) – це протокол для обміну структурованими повідомленнями у форматі XML. Він частіше використовується у корпоративних середовищах, де потрібні суворі стандарти та безпека.

Переваги:

– Стандартизація: SOAP дотримується чітких стандартів, що робить його надійним вибором для корпоративних застосунків.

– Підтримка WS-стандартів: SOAP підтримує різноманітні WS-стандарти, такі як WS-Security, WS-AtomicTransaction, що забезпечує додаткову функціональність і безпеку.

Недоліки:

– Складність: SOAP складніший у реалізації та підтримці, ніж REST, через його суворі стандарти та використання XML.

– Продуктивність: Висока складність призводить до збільшення часу відгуку та використання ресурсів.

RESTful API (Representational State Transfer) — це архітектурний стиль для розробки веб-сервісів, який базується на використанні HTTP-протоколу та базових HTTP-методів (GET, POST, PUT, DELETE, PATCH). Він є одним із найпопулярніших підходів для створення веб-інтерфейсів через свою простоту та гнучкість.

Переваги:

– Легкість використання: RESTful API легко інтегрується з існуючими веб-технологіями, оскільки використовує стандартні HTTP-запити та відповіді.

– Гнучкість: API може бути використане різними клієнтами, включаючи веб-додатки, мобільні додатки та інші сервіси.

– Масштабованість: Архітектура REST дозволяє створювати масштабовані та розподілені системи.

Недоліки:

– Відсутність суворих стандартів: REST більше орієнтований на практичність, що може призвести до непослідовності в реалізаціях.

– Безпека: Оскільки REST використовує відкриті стандарти, безпека може бути проблемою, якщо не впроваджувати додаткові заходи (такі як OAuth та JWT).

Шаблонізатори – це інструменти, які дозволяють розробникам створювати динамічний контент, поєднуючи дані з шаблонами. Вони використовуються в різних фреймворках та мовах програмування для створення сторінок з динамічним контентом.

Переваги:

– Спрощене створення контенту: шаблонізатори полегшують процес створення динамічних сторінок, дозволяючи зосередитися на даних, а не на деталях HTML.

– Зменшення дублювання коду: Вони дозволяють використовувати компоненти повторно, що підвищує ефективність коду.

Недоліки:

– Обмежена гнучкість: шаблонізатори можуть обмежувати гнучкість розробника, оскільки працюють за певними правилами та синтаксисом.

– Складність в розробці складних компонентів: шаблонізатори можуть бути не такими зручними для створення складних динамічних компонентів, як інші інструменти, такі як React чи Vue.

RESTful API, SOAP і шаблонізатори пропонують різні підходи до розробки веб-сервісів. REST надає гнучкість і простоту, SOAP орієнтований на стандарти та безпеку, а шаблонізатори полегшують створення динамічного контенту. Вибір між ними залежить від конкретних вимог вашого проекту, таких як безпека, продуктивність та гнучкість. З огляду на вимоги проєкту до гнучкості та можливості розробки мобільного додатку в майбутньому було обрано варіант використання RESTFul API.

2.2 Архітектура серверної частини

Як правило, для проєктування та впровадження інформаційної системи електронної комерції широко використовується 3-рівнева архітектура. Вона складається з користувацької вебсторінки, сервера додатків та бази даних [8]. Вебсторінка зазвичай створюється за допомогою набору інструментів користувацького інтерфейсу, таких як HTML, JavaScript та CSS. Сервер

додатків містить основну логіку дизайну та може зберігатися на фізичному сервері або на сервері, розташованому в хмарі.

Хмарні сервіси доступні в режимі «pay-as-you-go», тобто користувачі сплачують послуги на основі кількості використаних ресурсів та тривалості часу, використаного сервісом. Користувачі хмарних сервісів можуть запитувати IT-ресурси в будь-який час, коли вони потрібні, та звільняти ресурси після завершення обчислювальних завдань [9].

Основна перевага використання хмарних сервісів над традиційними способами на полягає в тому, що попит на сервіс не потрібно знати заздалегідь. Наприклад, сервер потрібно буде налаштувати, щоб задовольнити найвищий попит протягом місяця, і він буде невикористаний в інші часи місяця. Хмарні сервіси пом'якшують цю проблему, динамічно виділяючи більше ресурсів для обробки високої завантаженості та звільняючи ресурси, коли надходить менше запитів [10].

У традиційній 3-рівневій інформаційній системі веб-сервер повинен працювати постійно, без зупинок і без перерв, коли обслуговуються запити клієнтів. Для невеликих бізнесів, система може мати лише кілька сотень замовлень на день, наприклад, ресторан, протягом двох-трьох вихідних годин на день. Решту часу сервер буде неактивним. За кожну неактивну годину сервер коштує компанії гроші без отримання будь-якої фінансової вигоди. Щоб пом'якшити цю проблему, можна використовувати безсерверну архітектуру замість традиційної 3-рівневої архітектури. Безсерверна архітектура дозволяє бізнесу платити лише під час використання хмарних ресурсів, що значно знизить витрати на обслуговування для компаній [11].

Важливою відмінністю між 3-рівневою та хмарною архітектурою є різниця у складності підтримки вже готового продукту. Традиційна трирівнева архітектура дозволяє розробникам доволі швидко запустити продукт і при цьому не потребує для цього додаткових зусиль. Підтримка такого продукту є доволі простою і все зводиться лише до обслуговування сервера та коду програмного продукту. Проте, у випадку, коли сервер перестає справлятись із

навантаженням та потребує масштабування, то залишається лише варіант вертикального масштабування – тобто додавання ресурсів до сервера. З іншого боку, хмарні сервіси вирішують дане питання шляхом розподілу ресурсів між декількома серверами. Проте, підтримка таких систем є важчою і потребує більшої команди IT-спеціалістів для правильного налаштування та підтримки бази даних та серверної частини.

Отже, вибір архітектури залежить від багатьох факторів, включаючи розмір команди та тип застосунку. З огляду на це було обрано традиційну серверну архітектуру.

Інформаційна система складається із 3 основних компонентів: клієнтської частини, серверної частини, та бази даних. Для забезпечення швидкодії та повноти реалізації функцій було додано додаткові компоненти: сховище зображень та пошукову систему. Також, клієнтську частину було розділено на інтернет-магазин та адміністративну панель. На рисунку 2.2 схематично зображено архітектуру майбутньої інформаційної системи.

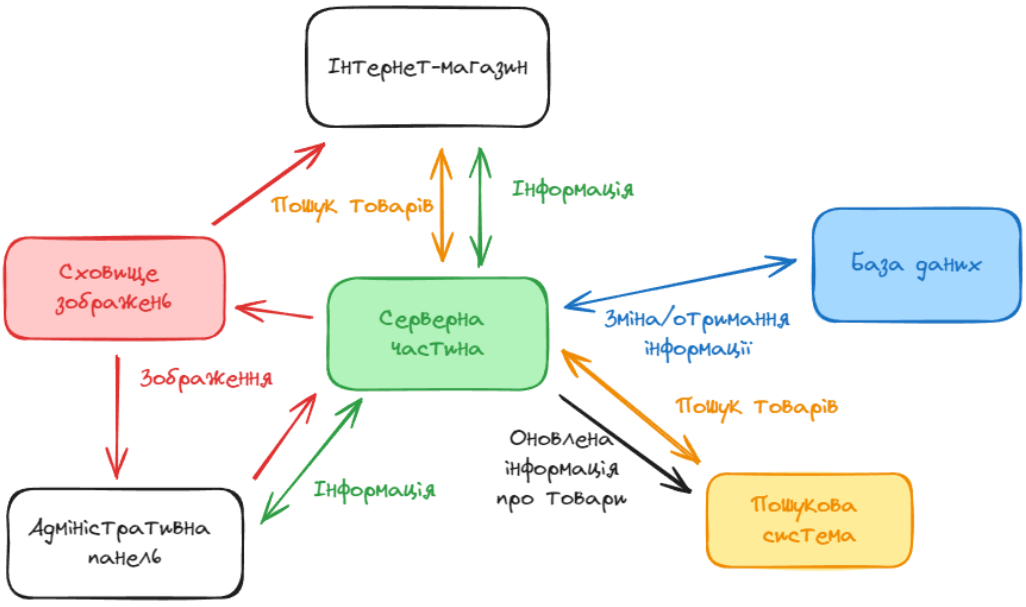

Рисунок 2.2 – Схематичне зображення архітектури ІС

2.3 Архітектура клієнтської частини

У веб-розробці "клієнтська сторона" стосується всього в вебзастосунку, що відображається або відбувається на клієнтському пристрої (пристрої

кінцевого користувача). Це включає те, що користувач бачить, наприклад, текст, зображення та інші елементи інтерфейсу, а також будь-які дії, які застосунок виконує всередині браузера користувача.

Мовами розмітки, такими як HTML і CSS, керує браузер на клієнтській стороні. Крім того, багато сучасних розробників додають клієнтські процеси в архітектуру своїх додатків, відходячи від виконання всього на сервері; бізнес-логіка для динамічних вебсторінок, наприклад, зазвичай працює на клієнтській стороні в сучасному вебзастосунку. Процеси на клієнтській стороні майже завжди написані мовою JavaScript.

HTML, CSS і JavaScript, які визначають, як вебсайт виглядає для користувача, обробляються браузером на клієнтській стороні. Сторінка також може реагувати на "події": наприклад, якщо користувач натискає на кнопку, обробник виконує описані розробником дії. Це приклад клієнтського процесу; код всередині самої веб-сторінки реагує на дії користувача і виконує процеси без взаємодії з сервером.

Клієнтська сторона також відома як "фронтенд", хоча ці два терміни не зовсім тотожні. "Клієнтська сторона" відноситься виключно до місця, де виконуються процеси, а "фронтенд" описує типи процесів, які працюють на клієнтській стороні.

У сучасній веб-розробці існують різні підходи до створення вебзастосунків: SPA (Single Page Applications), SSG (Static Site Generators) і SSR (Server-Side Rendered Apps).

SPA (односторінкові додатки) – це програми, що працюють у браузері та завантажують вміст через JavaScript. Вони використовують одну HTML-сторінку, яка завантажує дані на вимогу, без попереднього завантаження вмісту. Вони добре підходять для динамічних додатків, але можуть мати проблеми з SEO і швидкістю початкового завантаження.

Переваги:

– Динамічний вміст без додаткових завантажень між різними частинами додатка;

– Гнучкість у використанні сучасних веб-фреймворків;

– Можливість працювати в поєднанні з іншими технологіями.

Недоліки:

– Повільне початкове завантаження при складних додатках;

– Проблеми з SEO через брак початкового вмісту;

– Можуть виникати труднощі в підтримці великих файлів і складних додатків.

SSG (генератори статичних сайтів) створюють сторінки під час компіляції, а не на вимогу користувача. Це підходить для контенту, що не потребує персоналізації, і має переваги в швидкості завантаження та SEO.

Переваги:

– Швидкий час завантаження сторінок завдяки попередньому рендерингу;

– Переваги для SEO;

– Легка масштабованість і можливість створення децентралізованих архітектур.

Недоліки:

– Відсутність персоналізації та динамічного вмісту без додаткових зусиль;

– Для відображення змін у контенті необхідно змінювати вихідний код сторінки.

SSR (Додатки з рендерингом на стороні сервера) дозволяють створювати динамічний і персоналізований вміст, який відображається на стороні сервера до передачі в браузер. Це добре підходить для вмісту, який потребує миттєвих змін і персоналізації.

Переваги:

– Динамічний вміст без складних обходів, які потребують багато ресурсів;

– Миттєве відображення змін у контенті;

– Підходить для SEO та забезпечує гнучкість у персоналізації.

Недоліки:

– Вимагає більше API-запитів до сервера;

– SSR часто повільніше, ніж SPA і SSG, через необхідність збирати вміст на стороні сервера та передавання великого обсягу даних на клієнт.

Кожен із підходів має свої переваги та недоліки, і вибір залежить від конкретних потреб проекту. SPA добре підходять для динамічних додатків, але можуть мати проблеми з SEO та швидкістю початкового завантаження. SSG оптимальні для статичних сайтів, але обмежують можливість персоналізації. SSR забезпечують гнучкість і персоналізацію, але можуть бути повільнішими та вимагати більшої інфраструктури.

Наразі існує багато технології, таких як Next.js, які дозволяють поєднувати переваги різних підходів, надаючи можливість вибирати між SSR та SSG залежно від потреб проекту.

2.4 Огляд технологій клієнтської частини

Клієнтська частина інформаційної частини складається з двох частин: адміністративної панелі та сторінок самого інтернет-магазину. Для розробки цих частин було використано різні інструменти, так як вони виконують різні функції. Наприклад, інтернет-магазин має завантажуватися та працювати швидко, а для адміністративної панелі важливішим є можливість використання більшої кількості інструментів.

Проте, для використання CSS-стилів в обох проєктах було використано бібліотеку під назвою «TailwindCSS», що дозволяє використовувати стилі шляхом додавання визначених бібліотекою назв класів замість написання власних. Такий підхід здатний пришвидшити та спростити розробку, так як відсутня необхідність слідкувати за глобальними стилями, що зменшує кількість помилок під час розробки. Також даний інструмент дозволяє пришвидшити завантаження сторінок шляхом зменшення кількості даних, що відправляються з сервера до клієнта, так як на клієнт відправляються лише ті стилі, які було використано в проєкті.

Вибір основних інструментів для розробки клієнтської частини є важливим етапом розробки майбутнього веб-застосунка. Використання готових бібліотек здатне спростити розробку таких застосунків і сфокусуватися на розв’язанні проблем бізнеса.

Довгий час найпопулярнішим рішенням, яке задовольняло такі вимоги була бібліотека jQuery, проте на сьогоднішній день стандартом стала бібліотека React. jQuery та React створили власні ніші у сфері веб-розробки.

jQuery, багатофункціональна бібліотека JavaScript, є спрощеним, але потужним інструментом для виконання завдань JavaScript на веб-сайтах. Це відносно старий, проте потужний інструмент, який існує з 2006 року, полегшуючи життя розробників, обернувши багато рядків коду JavaScript у методи, які можна викликати з одного рядка. Це значно спрощує взаємодію скриптів з HTML-документами та спрощує маніпуляції з DOM сторінки, дозволяючи розробникам легко створювати веб-програми.

Переваги використання jQuery:

– Підтримує маніпуляції HTML/DOM та CSS;

– Спрощує використання методів подій HTML;

– Спрощує використовувати ефектів та анімацій;

– Надає зручний інтерфейс для виконання запитів до сервера;

– Надає можливість використання різних утиліт, включаючи плагіни для майже будь-яких завдань;

– Має гарну сумісність з усіма основними браузерами.

React – бібліотека JavaScript, яка призначена для створення інтерфейсів користувача, особливо для односторінкових вебзастосунків. На відміну від jQuery, React використовує компонентний підхід, зосереджуючись на повторно використовуваних компонентах і ефективному управлінні станом. Створений Facebook у 2011 році, React революціонізував спосіб створення інтерфейсів користувача.

Переваги використання React:

– Використання React надає розробнику доступ до фрагментів коду та компонентів React, таким чином вони можуть створювати певні частини інтерфейсу користувача, які можна використовувати на різних сторінках застосунка;

– Використання JSX, надає можливість зручно маніпулювати DOM використовуючи віртуальний DOM для покращення продуктивності веб-сайту;

– Існує безліч бібліотек для будь-якої конкретної функції інтерфейсу користувача, яку потребує розробник.

З огляду на більшу популярність та підтримку більшої кількості бібліотек основним інструментом для розробки компонентів для обох частин було обрано бібліотеку ReactJS. Це дозволило сфокусуватися саме на розробці клієнтської частини, а не на вивченні багатьох різних бібліотек та фреймворків.

Для покращення досвіду роботи клієнтів з сайтом було обрано фреймворк Astro. Даний фреймворк дозволяє використовувати обидва підходи для відображення сторінок: генерація статичних сторінок (SSG) та збір сторінок на сервері під час запиту (SSR). Використання SSG дозволяє генерувати більшість сторінок завчасно без додаткових обчислень під час запиту. Проте для деяких сторінок, таких як сторінка продукту потрібно використовувати SSG, так як неможливо завчасно знати, чи є необхідний ресурс за необхідним посиланням. Тобто, SSR надає змогу використання динамічних посилань і генерувати відповідні сторінки.

Іншою перевагою використання фреймворку Astro є можливість використання «острівців» – компонентів, які можуть містити в собі код JavaScript і підвантажуються динамічно в залежності від вказаних умов [12]. Використання таких компонентів дозволяє значно пришвидшити завантаження сторінок із важливим вмістом, тоді як менш важливі компоненти завантажуються пізніше.

Так як адміністративна панель має більше компонентів і є складнішою в реалізації, було обрано фреймворк NextJS. Next.js базується на React.js і призначений для розробників, які створюють високопродуктивні веб-додатки та надшвидкісні статичні сайти.

Цей фреймворк розширив рамки звичайного поділу між статичними та динамічними веб-сайтами. Завдяки автоматичній статичній оптимізації та новій функції часткового попереднього рендерингу Next.js дозволяє створювати гібридні додатки, які поєднують сторінки, що збираються на стороні сервера, зі статично згенерованими. Такий підхід підвищує продуктивність і зручність для користувачів, забезпечуючи швидкий початковий відгук від статичних сторінок та можливість динамічного завантаження контенту в разі потреби.

Цей фреймворк дозволяє використовувати набагато більше інструментів, заточених саме під використання на клієнтській частині застосунку, наприклад, Redux, що дозволяє описувати глобальний стан додатку, який поширюватиметься на всі сторінки та компоненти та інструмент Redux Toolkit Query, що надає змогу простого описання запитів до серверу та кешування результатів запитів, яке дозволяє зменшити кількість однакових запитів до серверу від одного клієнта. Ці інструменти вже стали стандартом для розробки високопродуктивних складних вебзастосунків.

2.5 Огляд технологій серверної частини

Для створення серверної частини застосунку було розглянуто різні мови та фреймворки, такі як:

– Rust – серверна мова програмування, яка дозволяє створювати швидкі застосунки, проте потребує багато часу для вивчення;

– Ruby on Rails – веб-додаток на стороні сервера, написаний мовою Ruby. Використання цього фреймворку значно полегшує створення серверної частини коду, проте такі застосунки мають порівняно невисоку швидкість та не є гнучкими;

– NodeJS з фреймворком Express – комбінація, що значно спрощує розробку серверної частини коду. Фреймворк Express полегшує роботу з HTTP-запитами та має функціональні можливості для простого створення маршрутизації. NodeJS не підходить для виконання «важких» запитів, що потребують складної обробки процесором.

Для розробки серверної частини застосунку було використано фреймворк під назвою Spring Framework. Даний фреймворк забезпечує комплексну модель програмування та конфігурації для сучасних корпоративних додатків на основі Java. Ключовим елементом Spring є підтримка інфраструктури на рівні додатків: Spring фокусується на «сантехніці» корпоративних додатків, щоб команди могли зосередитися на бізнес-логіці на рівні додатків, без зайвих прив'язок до конкретних середовищ розгортання. Також важливим є можливість переходу на мікросервісну або безсерверну архітектуру в майбутньому, у випадку необхідності масштабування серверу.

Spring Framework надає змогу використання багатьох інструментів для полегшення розробки функціоналу, наприклад:

- Spring Security – для захисту від поширених атак та забезпечення автентифікації та авторизації.
- Spring Data – для полегшення роботи з базою даних.
- Spring Boot – для розробки веб-орієнтованих застосунків.

Для роботи з базою даних також використовувався інструмент об’єктно-реляційного відображення під назвою Hibernate, основною функцією якого є трансформування записів з таблиць у об’єкти мови програмування, з якими можна працювати далі.

Внесення змін в структуру бази даних вручну має багато недоліків, наприклад необхідність виконання тих самих запитів для бази даних у середовищі розробки та робочому середовищі. Для вирішення даної проблеми було використано інструмент Liquibase – рішення для управління змінами схем баз даних, яке дозволяє переглядати і випускати зміни в базі даних

швидше і безпечніше, починаючи з розробки і закінчуючи виробничим середовищем. Він використовується для відстеження, керування та застосування змін схеми бази даних.

Для реалізації функції пошуку по товарам було розглянуто варіант використання Apache Lucene та Elasticsearch.

Apache Lucene - це високопродуктивна, повнофункціональна бібліотека текстової пошукової системи. Це технологія, що лежить в основі багатьох пошукових платформ, забезпечуючи базові можливості індексування та пошуку.

Ключові особливості Lucene:

– Розширені алгоритми пошуку: Lucene підтримує складні і точні пошукові запити, включаючи фразові запити, запити зі спеціальними символами і запити на основі близькості.

– Висока продуктивність: Система розроблена для ефективності і може працювати з великими наборами даних.

– Масштабованість: Хоча Lucene сама по собі не може автоматично обробляти розподілені обчислення, її можна масштабувати за допомогою додаткових архітектурних рівнів.

– Гнучкість: Можливо налаштовувати процес індексування, включаючи токенізацію, стеммінг і використання власних алгоритмів пошуку.

Elasticsearch, з іншого боку, є розподіленою, RESTful пошуковою та аналітичною системою, побудованою на основі Lucene. Він розширює можливості Lucene, надаючи масштабовану і просту у використанні платформу для управління і запитів великих обсягів даних в режимі реального часу. Elasticsearch абстрагує значну частину складнощів безпосереднього використання Lucene за допомогою більш доступного API та додаткових функцій, таких як розподілені обчислення та RESTful операції.

Було зроблено вибір на користь Apache Lucene, так як цей інструмент вирішує проблему пошуку і на відміну від Elasticsearch не потребує додаткового розгортання та підтримки.

2.6 Сховище зображень

Зображення грають важливу роль у покращенні користувацького досвіду веб-додатків. Для вибору методу збереження та доставки зображень було розглянуто різні способи зберігання зображень.

Першим варіантом є збереження зображень у файловій системі, що є одним з найпростіших підходів до зберігання зображень. Зберігання зображень безпосередньо у файловій системі сервера швидке і не вимагає додаткових сервісів. Однак, такий підхід може мати обмеження обсягу зберігання та бути вразливим до відмови сервера, так як обробка та доставка зображень вимагає використання більшої кількості ресурсів.

Хмарні сервіси зберігання, такі як Amazon S3 або Google Cloud Storage, надають масштабовані та надійні рішення для зберігання зображень. Вони забезпечують високу доступність та довговічність, але можуть бути дорожчими порівняно зі зберіганням у файловій системі сервера.

Кодування Base64 є зручним варіантом для невеликої кількості зображень, оскільки дозволяє уникнути потреби у додатковому зберіганні. Закодовані зображення можна зберігати у базі даних. Однак, воно може збільшити розмір бази даних та вплинути на продуктивність.

Використання мережі доставки контенту (CDN) дозволяє покращити швидкість та продуктивність веб-сайту. CDN зберігають зображення на кількох серверах по всьому світу, зменшуючи навантаження на основний сервер. Однак, послуги сервісів, що надають можливість використання CDN, часто потребують додаткових вкладень.

У якості сховища зображень було обрано мережу доставки контенту Cloudinary, так як окрім послуг CDN даний сервіс надає можливість редагування зображень перед та після збереження.

2.7 Система керування базами даних

Під час вибору системи керування базами даних було розглянуто два варіанти:

– реляційна база даних;

– документоорієнтована база даних.

Реляційні бази даних (RDBMS) організовуються у вигляді ієрархії з баз даних, таблиць і колонок. База даних містить різні таблиці та їхні налаштування. Таблиці та зв’язки між ними визначають структуру даних у базі. Вони створюються шляхом вказування імен та властивостей колонок, а також додаткових атрибутів, що застосовуються до всієї таблиці. Кожна колонка має тип даних, що визначає, які дані в ній можуть зберігатися. Можна також додавати обмеження для колонок і таблиць, щоб забезпечити дотримання певних правил.

Дані зберігаються у вигляді рядків у таблицях. Кожен рядок містить значення для кожної колонки. Також є можливість використання зовнішніх ключів для вказання первинного ключа для даних з іншої таблиці. Таким чином, реляційні системи керування базами даних об’єднують пов’язані таблиці та їхні налаштування в бази даних, де кожна таблиця задає власну структуру за допомогою колонок з різними типами даних.

Документно-орієнтовані системи керування базами даних структурують свої дані через ієрархію компонентів, але за іншими принципами. Вони зазвичай використовують концепцію баз даних, колекцій та документів.

Як і в реляційних базах даних, документні системи застосовують "базу даних" як контейнер для пов'язаних даних, забезпечуючи політику та простори імен на глобальному рівні. Тут також задаються загальні параметри, правила доступу та контекст для дій.

Колекції в документних базах даних виконують роль групування окремих документів. Вони менш суворо визначені, ніж таблиці в реляційних базах даних, і служать для зберігання документів та обмеження їхнього контексту.

На відміну від реляційних баз даних, колекції не визначають поля або властивості документів, які вони містять. Структура кожного документа залежить від полів і даних, що в ньому містяться. Тому документи в одній

колекції можуть мати різну структуру та властивості. Гарантування узгодженої структури є відповідальністю користувача, а не системи баз даних.

Хоч документні БД дозволяють швидше розробити застосунок, для розробки додатка було обрано варіант використання саме реляційної бази даних, так як підтримка такої бази даних є простішою завдяки чіткій завчасно визначеній структурі даних. Найпопулярнішими реляційними СКБД є SQLite, MySQL та PostgreSQL.

SQLite – це автономна, файлова і повністю відкрита система керування реляційними базами даних (RDBMS), відома своєю портативністю, надійністю та високою продуктивністю навіть у середовищах з обмеженою пам'яттю. Її транзакції відповідають вимогам ACID, навіть якщо система виходить з ладу або трапляються перебої в електропостачанні.

SQLite є "безсерверною" базою даних. Більшість реляційних систем баз даних реалізовані як серверні процеси, де програми спілкуються з сервером через міжпроцесну комунікацію. У випадку SQLite дані зберігаються у файловій системі і будь-який додаток може взаємодіяти з базою даних. Це спрощує процес налаштування, оскільки немає потреби конфігурувати сервер.

Переваги SQLite:

– Компактний розмір: СКБД SQLite дуже легка – вона може займати менше 600 КіБ простору. Крім того, вона не вимагає встановлення додаткових залежностей;

– Зручність у використанні: SQLite не працює як серверний процес, що означає, що її не потрібно зупиняти або перезапускати;

– Портативність: Вся база даних SQLite зберігається в одному файлі, який можна розмістити в будь-якому місці файлової системи.

Недоліки SQLite:

– Обмежена паралельність: Хоча кілька процесів можуть одночасно звертатися до бази даних SQLite, лише один процес може здійснювати запис у будь-який момент часу. Це робить SQLite менш придатним для високопаралельних систем;

– Відсутність управління користувачами: Оскільки SQLite записує в звичайний файл, єдині дозволи — це дозволи операційної системи. Це не підходить для застосунків, що потребують складної системи доступу;

– Безпека: Бази даних, що використовують сервер, можуть бути безпечнішими, оскільки клієнтські помилки не впливають на сервер. Також серверний підхід може забезпечити кращу контрольованість доступу.

MySQL є однією з найпопулярніших реляційних СКБД. Це багатофункціональний продукт, який використовується для роботи багатьох найбільших вебсайтів та додатків. Розпочати роботу з MySQL відносно просто завдяки вичерпній документації, великій спільноті розробників та великій кількості ресурсів, пов'язаних з MySQL, доступних онлайн.

На відміну від додатків, які використовують SQLite, додатки, що працюють з MySQL, взаємодіють з базою даних через окремий процес-демон. Завдяки цьому MySQL дозволяє краще контролювати доступ до бази даних.

MySQL надихнув створення великої кількості сторонніх інструментів, програм та бібліотек, які розширюють його функціональність та роблять його зручнішим у використанні. Деякі з найбільш популярних інструментів – це phpMyAdmin, DBeaver та HeidiSQL.

Переваги MySQL:

– Простота використання: MySQL - існує велика кількість документації як в друкованому, так і в онлайн-форматі. Доступні сторонні інструменти, як-от phpMyAdmin, які спрощують налаштування бази даних;

– Безпека: MySQL надає змогу налаштувати безпеку бази даних, налаштовуючи рівень безпеки паролів, задаючи пароль для root-користувача, видаляючи анонімні облікові записи та тестові бази даних. Також, на відміну від SQLite, MySQL підтримує управління користувачами та дозволяє задавати привілеї на індивідуальній основі;

– Швидкість: Завдяки відмові від реалізації деяких функцій SQL, MySQL зміг зосередитися на швидкості;

– Реплікація: MySQL підтримує кілька типів реплікації, що допомагає підвищити надійність, доступність та стійкість до відмов. Це корисно для налаштування резервного копіювання бази даних або для горизонтального масштабування.

Недоліки MySQL:

– Відомі обмеження: Через те, що СКБД MySQL орієнтована на швидкість і простоту використання, вона має певні функціональні обмеження. Наприклад, відсутня підтримка FULL JOIN;

– Паралельність і великі обсяги даних: хоча MySQL загалом добре працює з важкими операціями читання, проте одночасне читання та запис може бути проблематичним.

PostgreSQL, також відомий як Postgres, називають "найбільш просунутою відкритою реляційною базою даних у світі". Його створили з метою бути розширюваним і відповідним стандартам. PostgreSQL є об'єктно-реляційною базою даних, що означає, що, хоч він і є переважно реляційним, він також має функції, які частіше асоціюються з об'єктними базами даних, такі як успадкування таблиць та перевантаження функцій.

Postgres здатен ефективно обробляти багато завдань одночасно завдяки своїй реалізації мультиверсійного контролю паралельності (MVCC), що забезпечує атомарність, узгодженість, ізоляцію та довговічність транзакцій, відомі як відповідність ACID.

Переваги PostgreSQL:

– Відповідність SQL: PostgreSQL більше, ніж SQLite чи MySQL, намагається відповідати стандартам SQL. Згідно з офіційною документацією PostgreSQL, він підтримує 160 з 179 необхідних функцій для повної відповідності SQL:2011, а також довгий список додаткових опцій;

– Відкрите програмне забезпечення та спільнота: PostgreSQL є повністю відкритим проектом, код якого розроблений великою та відданою спільнотою. Спільнота PostgreSQL також підтримує безліч ресурсів, які пояснюють, як

працювати з цією базою даних, включаючи офіційну документацію, вікі PostgreSQL та різноманітні онлайн-форуми;

– Розширюваність: Користувачі можуть розширювати PostgreSQL програмно та в режимі реального часу через каталого-залежну операцію та використання динамічного завантаження. Наприклад, можна вказати файл з об'єктним кодом, такий як спільна бібліотека, і PostgreSQL завантажить його за потребою.

Недоліки PostgreSQL:

– Використання пам'яті: Для кожного нового клієнтського з'єднання PostgreSQL створює новий процес. Кожному процесу виділяється близько 10 МБ пам'яті, що може швидко накопичуватися, якщо є багато з'єднань. Через це для простих операцій, які потребують лише читання, PostgreSQL зазвичай поступається в продуктивності іншим RDBMS, таким як MySQL;

– Швидкість: PostgreSQL розроблений з акцентом на розширюваність та сумісність, іноді на шкоду швидкості. Якщо ваш проєкт вимагає максимально швидких операцій читання, PostgreSQL може бути не найкращим вибором;

– Налаштування: Через широкий набір функцій і сувору відповідність стандартам SQL, PostgreSQL може бути занадто складним для простих конфігурацій бази даних;

– Складна реплікація: Хоча PostgreSQL підтримує реплікацію, ця функція відносно нова. Деякі конфігурації можливі лише за допомогою розширень.

Реляційною СКБД, яка б задовольняла усі вимоги до безпеки та розширюваності стала PostgreSQL. Важливою перевагою стала можливість використання інструменту Ltree, який дозволяє легко працювати з ієрархіями об’єктів.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 База даних

У базі даних мають зберігатися дані про товари, клієнтів, категорії та замовлення. При цьому категорії являють собою ієрархію, тобто у категорії може бути батьківська категорія та декілька дочірніх. Іншою вимогою було забезпечення можливості збереження варіацій товарів, наприклад розмір або колір. Останньою вимогою була можливість використання латинських слів, які далі будуть фрагментами адреси сторінки (slug), для ідентифікації товарів та категорій.

Для забезпечення збереження ієрархії категорій було використано підхід з використанням «матеріалізованого шляху», що зберігає повний шлях до об’єкта, починаючи від кореню. При цьому для полегшення роботи було використано розширення Ltree, що дозволяє легко та швидко шукати батьківські та дочірні категорії. Шлях складається із латинських слів, розділених крапками.

Для забезпечення можливості створення варіацій було створено відповідні таблиці: варіації, опції та конфігурація товару. При цьому основний товар не дублюється і використовується для зберігання основних даних про товар, таких як назва чи опис.

Схему бази даних, що відповідає зазначеним вимогам, наведено на рисунку 3.1.

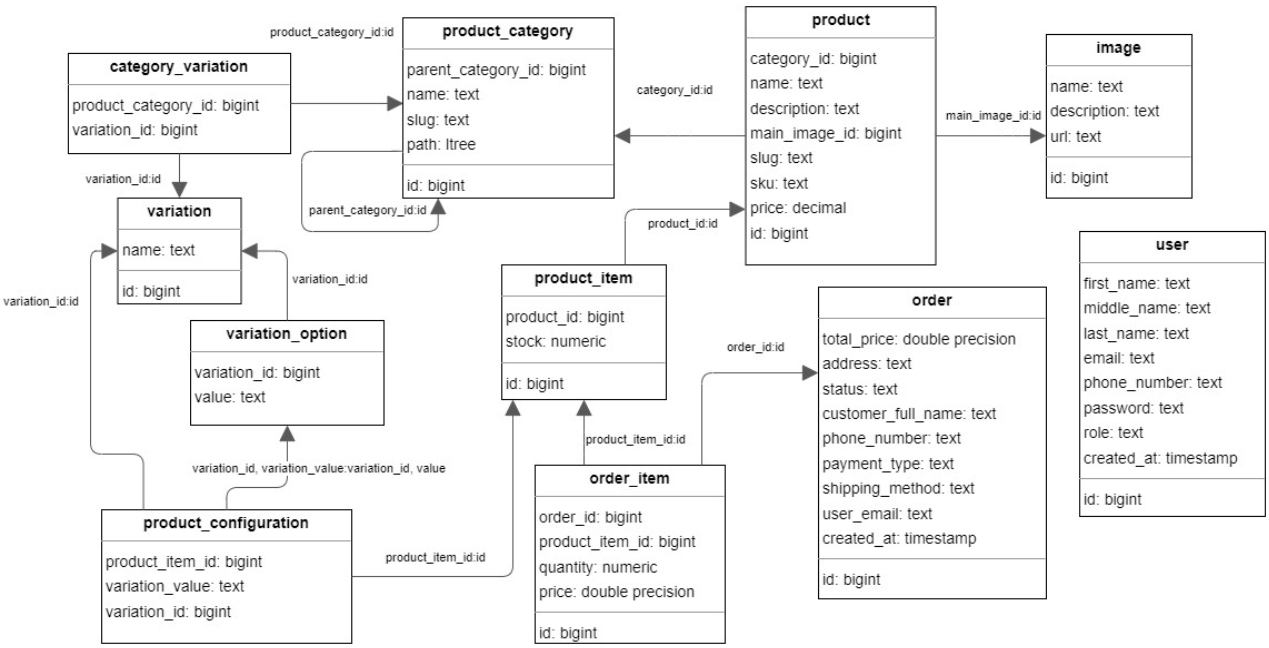

Рисунок 3.1 – Схема бази даних

Структура таблиць бази даних створюється зі сценаріїв, написаних для бібліотеки Liquibase у форматі «yaml», хоча можливе використання сценаріїв, написаних мовою запитів SQL. Приклад сценарію, в якому відбувається реєстрація розширення Ltree, додаються стовпчики до таблиці та створюється тригер наведено на рисунку 3.2.

```yaml
databaseChangeLog:
  - changeSet:
      id: 028-use-materialized-path-for-categories
      author: andrii
      changes:
        - sql:
            dbms: postgresql
            sql: CREATE EXTENSION ltree;
        - addColumn:
            schemaName: main
            tableName: product_category
            columns:
              - column:
                  name: slug
                  type: tex
                  constraints:
                    nullable: false
              - column:
                  name: path
                  type: ltree
                  constraints:
                    nullable: false
                    unique: true
        - sql:
            dbms: postgresql
            endDelimiter: "@@"
            sql: |
              CREATE function main.create_category_path()
                  RETURNS trigger AS
              $$
              BEGIN
                  IF NEW.parent_category_id IS NOT NULL THEN
                      NEW.path =
                              (SELECT path || NEW.slug
                               FROM main.product_category
                               WHERE id = NEW.parent_category_id);
                  ELSE
                      NEW.path = NEW.slug;
                  END IF;
```

Рисунок 3.2 – Приклад сценарію Liquibase

3.2 Серверна частина

Головною функцією цієї частини інформаційної системи є робота з базою даних. Для представлення об’єктів бази даних було розроблено відповідні класи, що містять інформацію про структуру цих об’єктів. Приклад такого класу наведено на рисунку 3.3.

```java
@Entity
@Table(name = "product", schema = "main")
@Getter
@Setter
public class Product {
  @OneToMany(mappedBy = "product", cascade = CascadeType.ALL, orphanRemoval = true)
  private final Set<ProductItem> productItems = new HashSet<>();

  @Id
  @GeneratedValue(generator = "product_seq")
  @SequenceGenerator(
      name = "product_seq",
      sequenceName = "product_seq",
      schema = "main",
      allocationSize = 1)
  @Column(unique = true, nullable = false, updatable = false)
  private Long id;

  @Column(nullable = false)
  private String name;

  @Column private String description;

  @Column private String slug;

  @ManyToOne(cascade = CascadeType.MERGE)
  private Image mainImage;

  @ManyToOne private Category category;
}
```

Рисунок 3.3 – Клас, що відповідає об’єкту товару

Для роботи з базою даних було створено окремі класи, що називаються репозиторіями. Хоча бібліотека Spring Data автоматично створює основні запити для збереження, пошук, оновлення та видалення об’єктів. Іноді потрібно виконати нестандартні запити. Приклад репозиторію, що містить власні запити наведено на рисунку 3.4.

```java
@Repository
public interface ProductRepository extends JpaRepository<Product, Long> {
 ✨ Codeium: Explain  Docstring
  6 usages   Andrey
  Optional<Product> findBySlug(String slug);

 ✨ Codeium: Explain  Docstring
  1 usage   Andrey
  @Query(
      value =
          """
                  SELECT p.* FROM main.product p JOIN main.product_category c ON p.category_id = c.id
                  WHERE c.path <@ public.text2ltree(:#{#category.path})
                  """,
      countQuery =
          """
                  SELECT COUNT(*) FROM main.product p JOIN main.product_category c ON p.category_id = c.id
                  WHERE c.path <@ public.text2ltree(:#{#category.path})
                  """,
      nativeQuery = true)
  Page<Product> findAllByCategory(@Param("category") Category category, Pageable pageable);
}
```

Рисунок 3.4 – Інтерфейс для роботи з даними про товари

Вся бізнес-логіка, яка необхідна для обробки даних, розміщена в класах, що називаються сервісами. Такі сервіси необхідні для перевірки наявності товару під час створення замовлення, пошуку категорій за шляхом, підготовки зображення для збереження у зовнішньому сервісі тощо. Приклад функції такого сервісу наведено на рисунку 3.5.

```java
@Override
@Transactional
public CollectionModel<ProductDto> findByCategoryPath(String path, Pageable pageable) {
  if (path == null) {
    throw new IllegalArgumentException("Path cannot be null");
  }

  String ltreePath = path.replace( target: "/",  replacement: ".").replace( target: "-",  replacement: "_");
  if (ltreePath.startsWith(".")) {
    ltreePath = ltreePath.substring( beginIndex: 1);
  }

  Category category =
      categoryRepository
          .findByPath(ltreePath)
          .orElseThrow(
              () -> new CategoryNotFoundException("Category not found with path: " + path));

  return pagedResourcesAssembler.toModel(
      productRepository.findAllByCategory(category, pageable), productModelAssembler);
}
```

Рисунок 3.5 – Функція пошуку товарів за шляхом категорії

Повний код розміщено в git-репозиторії ks-java [18].

3.3 Сховище зображень

Для забезпечення швидкого доступу та приведення завантажених до системи зображень товарів було використано CDN (Content Delivery Network). У якості провайдера таких послуг було обрано Cloudinary – комплексне рішення для управління зображеннями та відео для веб-сайтів і мобільних додатків, що охоплює все: від завантаження, зберігання, маніпуляцій, оптимізації до доставки зображень і відео.

Усі завантажені зображення приводяться до одного вигляду завдяки модифікаціям, які надає Cloudinary. Наприклад, встановлюється розмір зображення, співвідношення сторін та формат зображення. Вся обробка проходить на стороні CDN, що дозволило зменшити час розробки функцій обробки, збереження та доставки зображень. Приклад коду, який надсилає зображення до Cloudinary та вказує, як саме потрібно модифікувати зображення наведено на рисунку 3.6.

```java
public URL upload(String base64encodedImage, String filename) {
  if (!StringUtils.hasText(base64encodedImage)) {
    throw new ImageUploadException("Base64 encoded image is empty");
  }
  if (!StringUtils.hasText(filename)) {
    throw new ImageUploadException("Filename is empty");
  }

  String publicId = StringUtils.replace(filename, oldPattern: " ", newPattern: "_");
  EagerTransformation transformation =
      new EagerTransformation()
          .width(IMAGE_WIDTH)
          .height(IMAGE_HEIGHT)
          .crop(IMAGE_CROP)
          .fetchFormat(IMAGE_FORMAT);

  Map<String, Object> options = new HashMap<>();
  options.put("public_id", publicId);
  options.put("eager", List.of(transformation));

  try {
    byte[] decodedBytes = Base64.getDecoder().decode(base64encodedImage);
    Map<?, ?> res = cloudinary.uploader().upload(decodedBytes, options);

    log.info("Uploaded file: {}", res);

    if (res.get("eager") instanceof List<?> eagerMapArray) {
      for (var eagerItem : eagerMapArray) {
        if (eagerItem instanceof Map<?, ?> eagerMap
            && eagerMap.get("format") instanceof String format
            && (format.equals(IMAGE_FORMAT))) {
          return new URL(eagerMap.get("url").toString());
        }
      }
    }

    return new URL(res.get("url").toString());

  } catch (IOException exception) {
    throw new ImageUploadException(exception.getMessage());
  }
}
```

Рисунок 3.6 – Програмний код завантаження зображення до Cloudinary

3.4 Клієнтська частина

Під час розробки клієнтської частини було використано монорепозиторій – підхід, що являє собою використання одного репозиторію для декількох проєктів замість створення декількох репозиторіїв. Використання такого підходу дозволяє уникнути дублювання великої кількості коду, який використовується в багатьох проєктах, таких як опис структури об’єктів, що передаються або отримуються з API, спільних функцій тощо. Спільний код було винесено у окремий модуль під назвою «shared». Приклад спільного коду наведено на рисунку 3.7.

```
export type AuthResponse = {
  accessToken: string;
  refreshToken: string;
};

export type LoginRequest = {
  email: string;
  password: string;
};

export type RegisterRequest = {
  firstName: string;
  middleName?: string;
  lastName?: string;
  email: string;
  phoneNumber: string;
  password: string;
};

4 usages  Andrey
export interface PasswordChangeRequest {
  currentPassword: string;
  newPassword: string;
}
```

Рисунок 3.7 – Спільні типи даних, що використовуються при вході та реєстрації

Повний код інтернет-магазина та адміністративної панелі розміщено в git-репозиторії ks-react [19].

3.5 Інтернет-магазин

Фреймворк Astro, що був використаний для розробки сторінок інтернет-магазину потребує використання власної структури проєкту. Наприклад, всі сторінки маються міститися в каталозі «pages» та або його дочірніх каталогах. Іншою важливою особливістю є створення сторінок, які не мають точного шляху, наприклад, сторінки товарів – для створення таких сторінок потрібно

правильно називати файли. Для цього випадку було обрано назву файлу «[slug].astro», де «slug» – назва товару.

Так як більшість сторінок мають дуже схожу структуру, було використано шаблони оформлення сторінки. Такі шаблони описують структуру сторінки, основні компоненти, такі як підвал сторінки та заголовок. При використанні таких шаблонів тег «slot» позначає місце, в яке буде підставлятися зміст сторінок. Приклад такого шаблону наведено на рисунку 3.8.

```
---
import Sidebar from "../components/cart/Sidebar";
import Footer from "../components/Footer.astro";
import Header from "../components/header/Header.astro";

const { title, description } = Astro.props;
---

<html lang="uk" class="min-h-full">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" type="image/jpeg" href="/favicon.ico" />
    <meta name="viewport" content="width=device-width" />
    <meta name="generator" content={Astro.generator} />
    <meta name="description" content={description} />
    <meta name="theme-color" content="#222222" />
    <link rel="apple-touch-icon" href="/logo.png" />
    <title>{title} – Кішка Стрибає</title>
  </head>
  <body class="flex flex-col min-h-screen">
    <Header />
    <slot />
    <Footer />
    <Sidebar client:idle />
  </body>
</html>
```

Рисунок 3.8 – Шаблон оформлення сторінок інтернет-магазину

Іншою важливою особливістю цього фреймворку є мінімізація використання скриптів на стороні клієнта. У випадку, якщо використання таких скриптів є необхідністю, то потрібно це зазначити в параметрі «client» компонента, який містить скрипти. Такі скрипти будуть підвантажені в залежності від значення цього параметру, наприклад: load, idle visible та ін.

Всі сторінки мають ідентичну шапку (рис. 3.9), що містить логотип, перелік батьківських категорій, кнопку для відкриття кошику та кнопку

відкриття особистого кабінету. При наведенні на будь-яку з батьківських категорій, автоматично відображаються дочірні категорії.

Рисунок 3.9 – Шапка інтернет-магазину

Також спільним для основних сторінок сайту є нижня частина сайту (рис. 3.10), що містить посилання для отримання основної інформації.

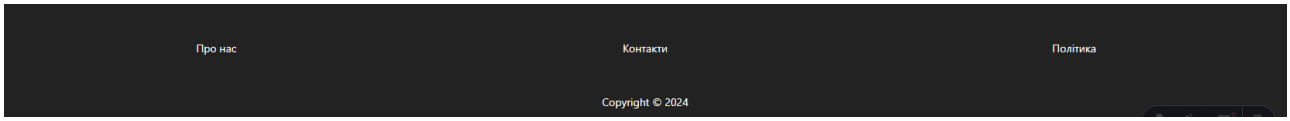

Рисунок 3.10 – Нижня частина інтернет-магазину

Основною сторінкою, з якою взаємодіятиме користувач є сторінка каталогу (рис. 3.11), на якій розміщено поле текстового пошуку та товари, які фільтруються в залежності від того, чи було виконано пошук, або обрано категорію. На сторінках каталогу початково відображається лише невеличка частина товарів. При гортанні сторінки вниз, товари автоматично завантажуються з серверу. Дану функцію було реалізовано з використанням вбудовування React-компонента (рис. 3.12) в сторінку, так як такі компоненти працюють на стороні браузера, а сторінки Astro збираються до того, як потраплять у браузер. В компоненті було створено функцію, яка посилатиме запити до сервера з вказанням сторінки, яку треба запросити, та оновлюватиме дані в залежності від відповіді. Зазначена функція викликається у разі зміни сторінки, яка відбувається в обробнику події гортання сторінки.

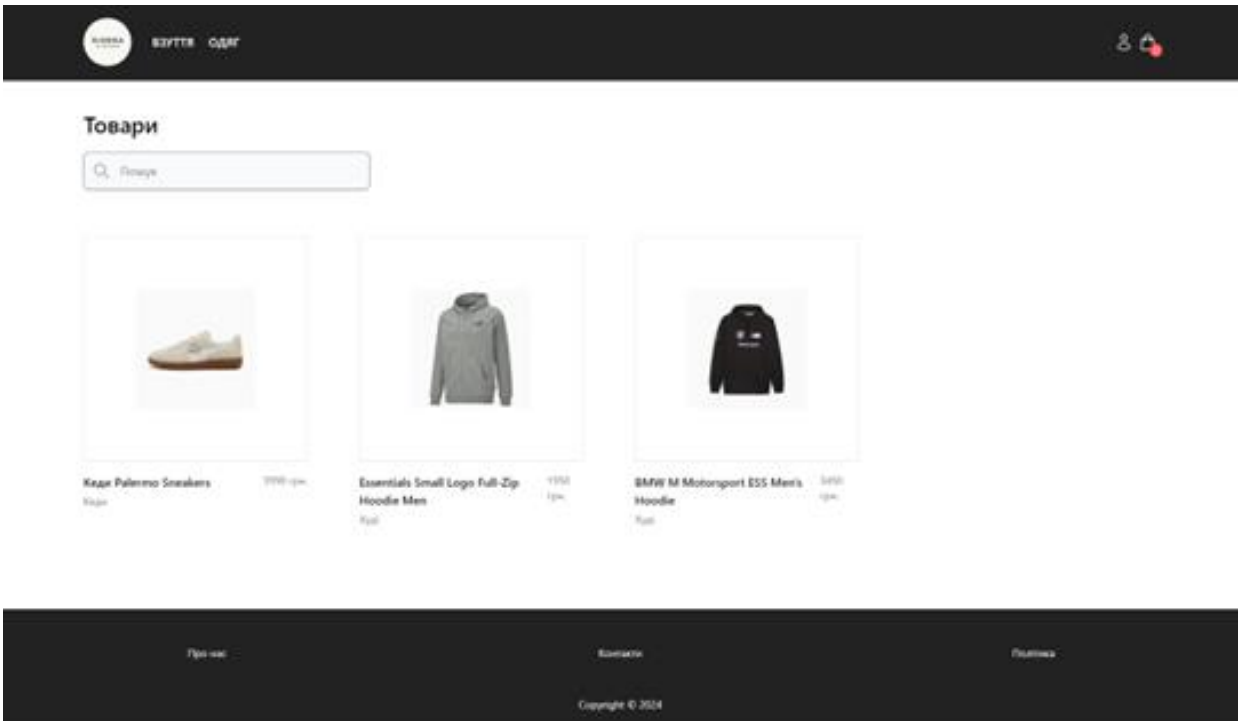

Рисунок 3.11 – Сторінка каталогу

```tsx
const Products = ({ categoryPath, query }: Props) => {
  const [products, setProducts] = useState([] as ProductType[]);
  const [isLastPage, setIsLastPage] = useState(false);

  const [isLoading, setIsLoading] = useState(true);

  const [params, setParams] = useState({
    size: "5",
    page: 0,
    q: query,
    ...(categoryPath && { categoryPath }),
  });

  useEffect(() => {
    setIsLoading(true);
    getAll(new URLSearchParams(params)).then((res) => {
      if (params.page === res.data?.page?.totalPages) {
        setIsLastPage(true);
      }
      setProducts(res.data?._embedded?.products ?? []);
      setIsLoading(false);
    });
  }, [params]);

  useEffect(() => {
    window.addEventListener("scroll", handleScroll);

    return () => {
      window.removeEventListener("scroll", handleScroll);
    };
  }, []);

  const handleScroll = () => {
    if (isLastPage) return;
    const scrollTop = document.documentElement.scrollTop;
    const offsetHeight = document.documentElement.offsetHeight;
    const clientHeight = document.documentElement.clientHeight;

    if (scrollTop + clientHeight >= offsetHeight && !isLastPage) {
      setParams({
        ...params,
        page: params.page + 1,
      });
    }
  };
```

Рисунок 3.12 – Частина компоненту, що відповідає за оновлення товарів на сторінці

При натисненні на картку товару відкривається сторінка товару (рис. 3.13), на якій є зображення, інформація про товар, елементи для вибору варіації товару та кнопка додавання обраної варіації товару до кошика.

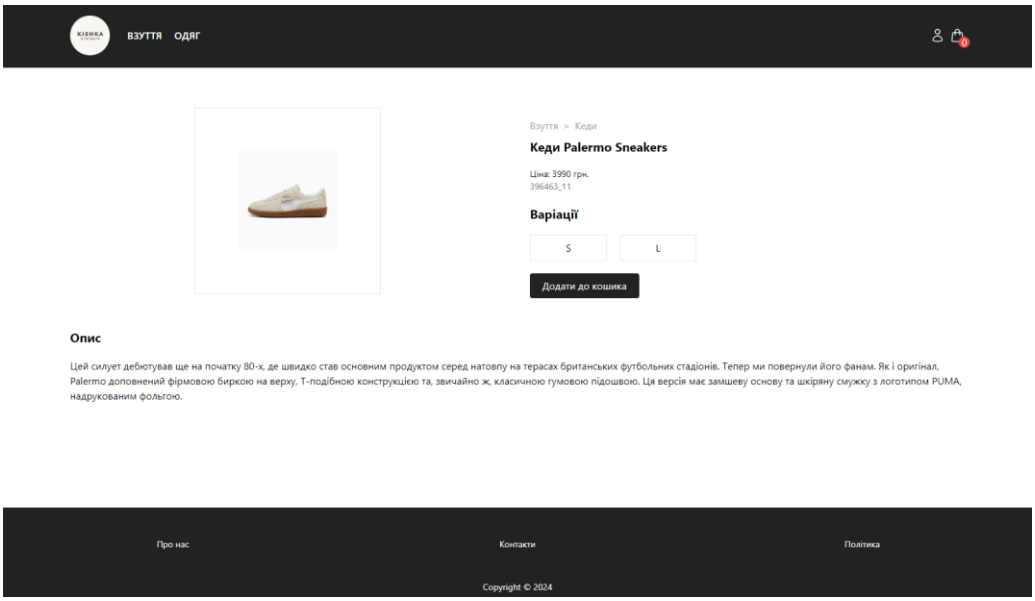

Рисунок 3.13 – Сторінка товару

Після вибору варіації товару та натискання кнопки «Додати до кошика» автоматично відкривається кошик (рис. 3.14), в якому відображається інформація про додані товари та наявні елементи для зміни кількості товарів та їх видалення. Для збереження стану кошика між сторінками було використано бібліотеку Nanostores, та створено сховище даних кошика (рис. 3.15).

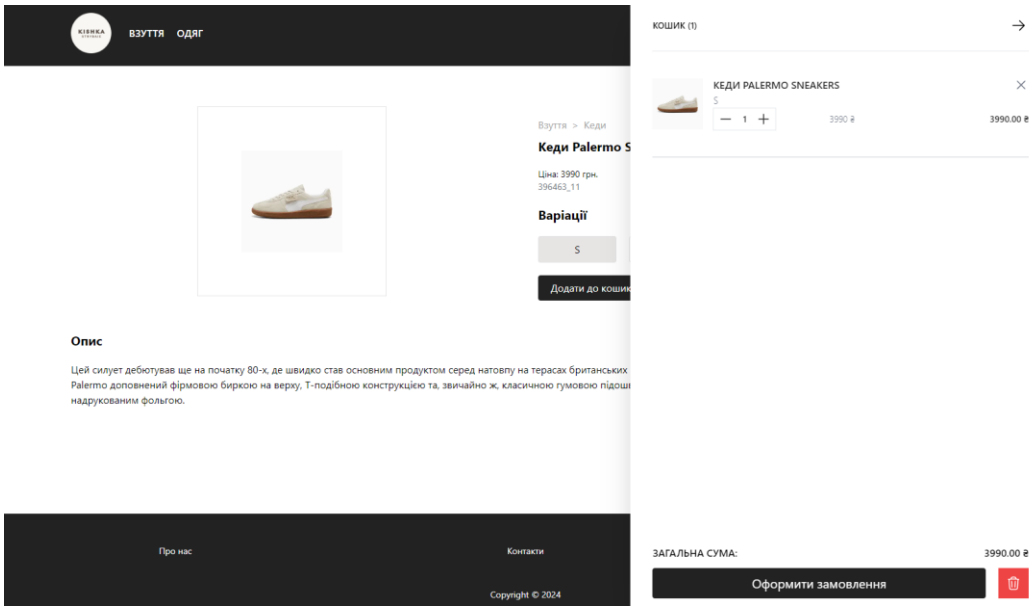

Рисунок 3.14 – Сторінка товару з відкритим кошиком

```
export const shoppingCart = persistentAtom<CartItem[]>("cart", [], {
  encode: JSON.stringify,
  decode: (value) => {
    try {
      const cart = JSON.parse(value);
      if (!Array.isArray(cart)) {
        shoppingCart.set([]);
        return [];
      }

      return cart;
    } catch (e) {
      shoppingCart.set([]);
      return [];
    }
  },
});

export const addCartItem = (product: Product, productItem: ProductItem) => {
  const existingEntry = shoppingCart
    .get()
    .find(
      (item) =>
        item.productItemId === productItem.id && item.productId === product.id,
    );

  if (existingEntry) {
    shoppingCart.set([
      ...shoppingCart.get().map((item) => {
        if (
          item.productItemId === productItem.id &&
          item.productId === product.id
        ) {
          return {
            ...item,
            quantity: item.quantity + 1,
          };
        }
        return item;
      }),
    ]);
  } else {
    shoppingCart.set([
      ...shoppingCart.get(),
      {
        ...product,
        ...productItem,
        productSlug: product.slug,
        quantity: 1,
        category: product.category.name,
        productId: product.id,
        productItemId: productItem.id,
      },
    ]);
  }
};
```

Рисунок 3.15 – Сховище стану кошика

Також було розроблено сторінку оформлення замовлення (рис. 3.16), сторінку входу до особистого кабінету та сторінки особистого кабінету (рис. 3.17-3.18).

Розроблені сторінки та компоненти забезпечують зручний доступ до основних функцій інтернет-магазину, таких як: пошук, перегляд та замовлення товарів. Варто зазначити, що всі сторінки інтернет-магазину також було адаптовано для використання з мобільних пристроїв.

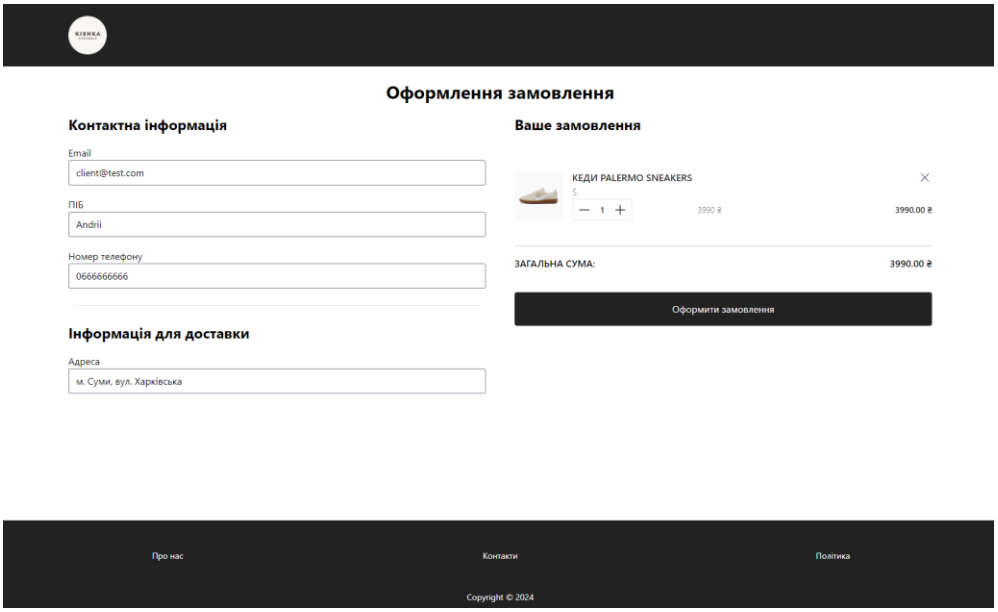

Рисунок 3.16 – Сторінка оформлення замовлення

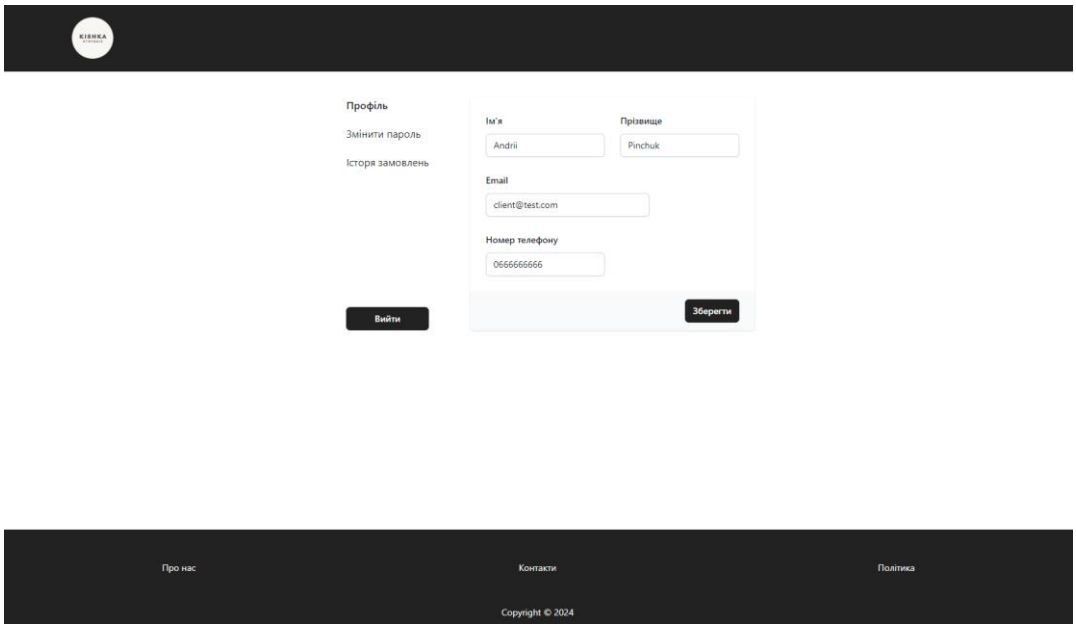

Рисунок 3.17 – Сторінка зміни особистої інформації

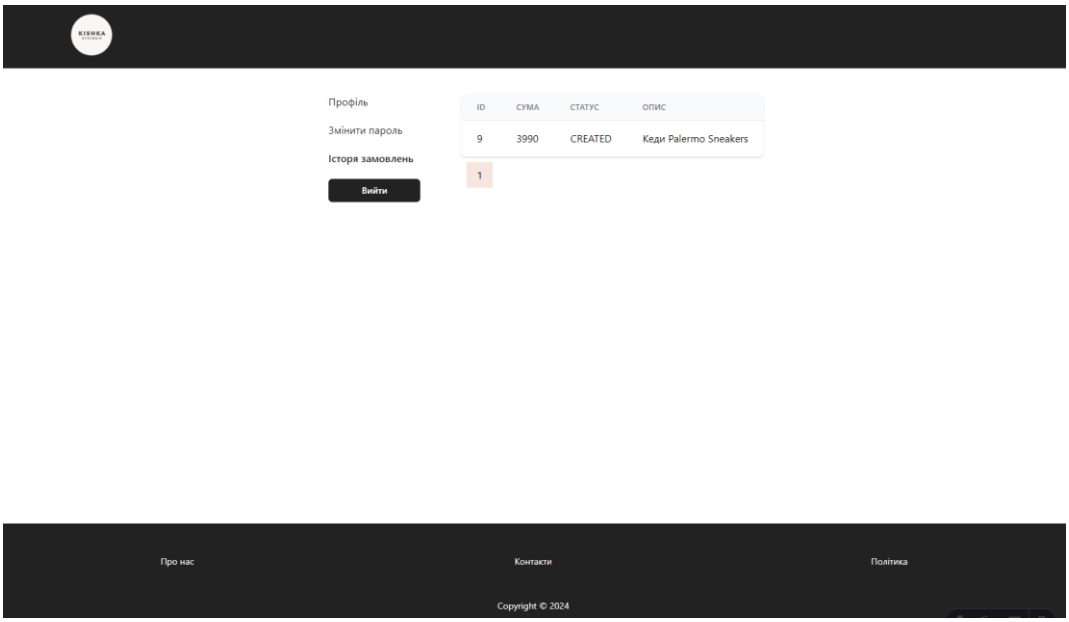

Рисунок 3.18 – Сторінка історії замовлень

3.6 Адміністративна панель

Фреймворк NextJS, що був використаний для розробки адміністративної панелі також потребує використання власної структури, яка значною мірою схожа на структуру проєкту інтернет-магазину.

Так само, як і сторінки інтернет-магазину, сторінки адміністративної панелі мають однакову структуру, для забезпечення якої було використано шаблони. Приклад шаблону наведено на рисунку 3.19.

```
return (
  <div
    className={`${darkMode ? "dark" : ""
      } overflow-hidden lg:overflow-visible`}
  >
    {isLoading ? (
      <div className="flex items-center justify-center h-screen">
        <BaseIcon path={mdiBackburger} className="animate-spin" />
      </div>
    ) : data?.role !== "ADMIN" ? (
      <div
        className={`${layoutAsidePadding} min-h-screen w-screen bg-gray-50 transition-position dark:bg-slate-800 dark:text-slate-100`}
      />
    ) : (
      <div
        className={`${layoutAsidePadding} ${isAsideMobileExpanded ? "ml-60 lg:ml-0" : ""
          } min-h-screen w-screen bg-gray-50 transition-position dark:bg-slate-800 dark:text-slate-100 lg:w-auto`}
      >
        <NavBar
          menu={menuNavBar}
          className={`${layoutAsidePadding} ${isAsideMobileExpanded ? "lg:ml-0" : ""
            }`}
        >
          <NavBarItemPlain
            display="flex lg:hidden"
            onClick={() => setIsAsideMobileExpanded(!isAsideMobileExpanded)}
          >
            <BaseIcon
              path={isAsideMobileExpanded ? mdiBackburger : mdiForwardburger}
              size="24"
            />
          </NavBarItemPlain>
          <NavBarItemPlain
            display="hidden lg:flex xl:hidden"
            onClick={() => setIsAsideLgActive(true)}
          >
            <BaseIcon path={mdiMenu} size="24" />
          </NavBarItemPlain>
        </NavBar>
        <AsideMenu
          isAsideMobileExpanded={isAsideMobileExpanded}
          isAsideLgActive={isAsideLgActive}
          menu={menuAside}
          onAsideLgClose={() => setIsAsideLgActive(false)}
        />
        {children}
        <FooterBar />
      </div>
    )}
    <ToastContainer />
  </div>
```

Рисунок 3.19 – Частина шаблону сторінок адміністративної панелі

Для забезпечення виведення повідомлень на сторінки було розроблено компонент ToastContainer (рис. 3.20), який слідкує за станом сховища повідомлень (рис. 3.21) та відображає повідомлення. Так, як адміністративна панель являє собою односторінковий додаток, це надало можливість використати фреймворк Redux, який спрощує управління глобальним станом вебсистеми. Приклад повідомлення зображено на рисунку 3.22.

```tsx
import ToastNotification from './ToastNotification';
import { useAppSelector } from '../stores/hooks';
import { Toast } from 'types/toast';

const ToastContainer = () => {
  const toasts: Toast[] = useAppSelector((state) => state.toasts.toasts);

  return (
    <div className="fixed bottom-10 z-50 right-0">
      <div className="max-w-xl mx-auto">
        {toasts &&
          toasts?.map((toast) => (
            <ToastNotification
              id={toast.id}
              key={toast.id}
              type={toast.type}
              delay={toast.delay}
              message={toast.message}
            />
          ))}
      </div>
    </div>
  );
};

export default ToastContainer;
```

Рисунок 3.20 – Програмний код компонента ToastContainer

```tsx
export const toastSlice = createSlice({
  name: "toast",
  initialState,
  reducers: {
    addToast: (state, action: PayloadAction<ToastPayloadObject>) => {
      const id = Math.random().toString(36).substring(2, 9);

      state.toasts.push({
        id: id,
        type: action.payload.toast.type,
        message: action.payload.toast.message,
        delay: action.payload.delay ?? DEFAULT_DELAY,
      });
    },

    removeToast: (state, action: PayloadAction<string>) => {
      state.toasts = state.toasts.filter(
        (toast) => toast.id !== action.payload,
      );
    },
  },
});

export const { addToast, removeToast } = toastSlice.actions;

export const toastMiddleware: Middleware = (api) => (next) => (action) => {
  if (isRejectedWithValue(action)) {
    if (action.payload.data?.message) {
      api.dispatch(
        addToast({
          toast: {
            type: ToastType.danger,
            message: action.payload.data.message,
          },
        }),
      );
    }

    for (const key in action.payload.data?.fields) {
      api.dispatch(
        addToast({
          toast: {
            type: ToastType.danger,
            message: `${key}: ${action.payload.data.fields[key]}`,
          },
        }),
      );
    }
    return next(action);
  }

  return next(action);
};
```

Рисунок 3.21 – Сховище глобального стану повідомлень

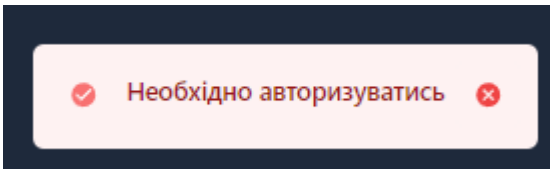

Рисунок 3.22 – Приклад повідомлення

Використання бібліотеки Redux надає можливості використання бібліотеки Redux Toolkit Query, що дозволяє спростити розробку інтерфейсів взаємодії з сервером, а також дозволяє зберігати результати запитів на стороні клієнта щоб зменшити кількість запитів до API. Приклад створення сервісу для роботи з API зображено на рисунку 3.23.

```
export const authApi = createApi({
  reducerPath: "authApi",
  baseQuery: fetchBaseQuery({
    baseUrl: baseUrl,
    prepareHeaders: (headers, { getState }) => {
      const token = (getState() as RootState).token.accessToken;
      if (token) {
        headers.set("Authorization", `Bearer ${token}`);
      }
      return headers;
    },
  }),
  endpoints: (builder) => ({
    login: builder.mutation<AuthResponse, LoginRequest>({
      query: (credentials) => ({
        url: "/login",
        method: "POST",
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify(credentials),
      }),
    }),
  }),
});
```

Рисунок 3.23 – Програмний код сервісу взаємодії з API для авторизації користувачів

Після авторизації адміністратора в систему відображається головна сторінка адміністративної панелі (рис. 3.24), на якій виведено статистику та графік замовлень за останній тиждень. У випадку, якщо користувач не авторизований, або не є адміністратором, буде виведено відповідне повідомлення.

Рисунок 3.24 – Головна сторінка адміністративної панелі

Окрім перегляду статистики, адміністратори також мають змогу переглядати наявні товари, категорії, варіації та ін. у вигляді таблиць з можливістю сортування та пагінації. Приклад перегляду замовлень наведено на рисунку 3.25.

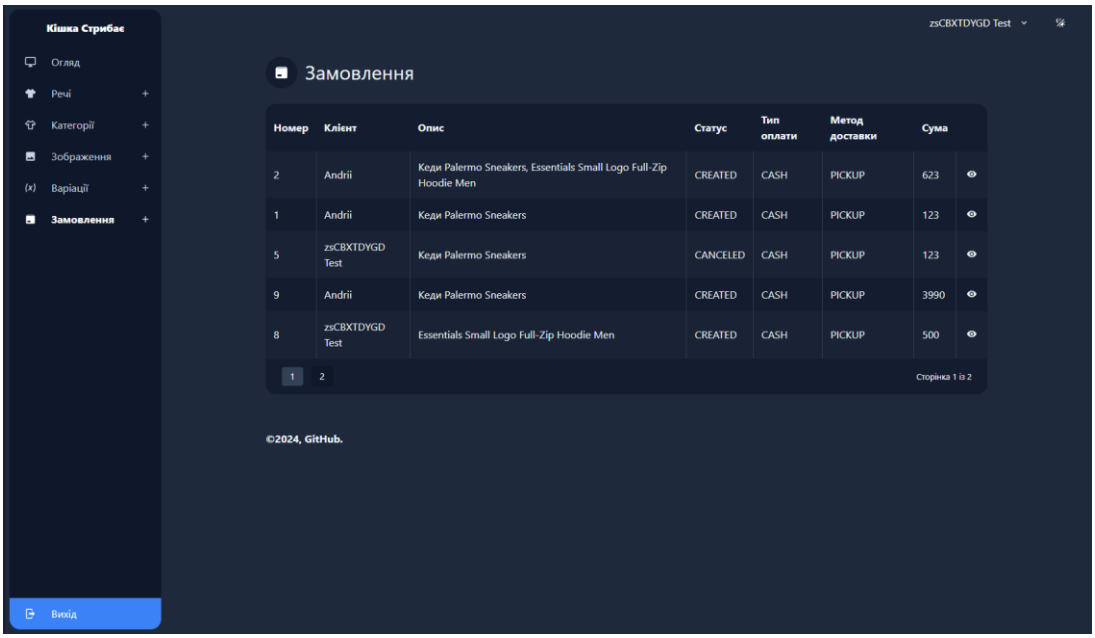

Рисунок 3.25 – Сторінка перегляду замовлень

Також було реалізовано можливість перегляду детальної інформації та редагування (рис. 3.26-3.27).

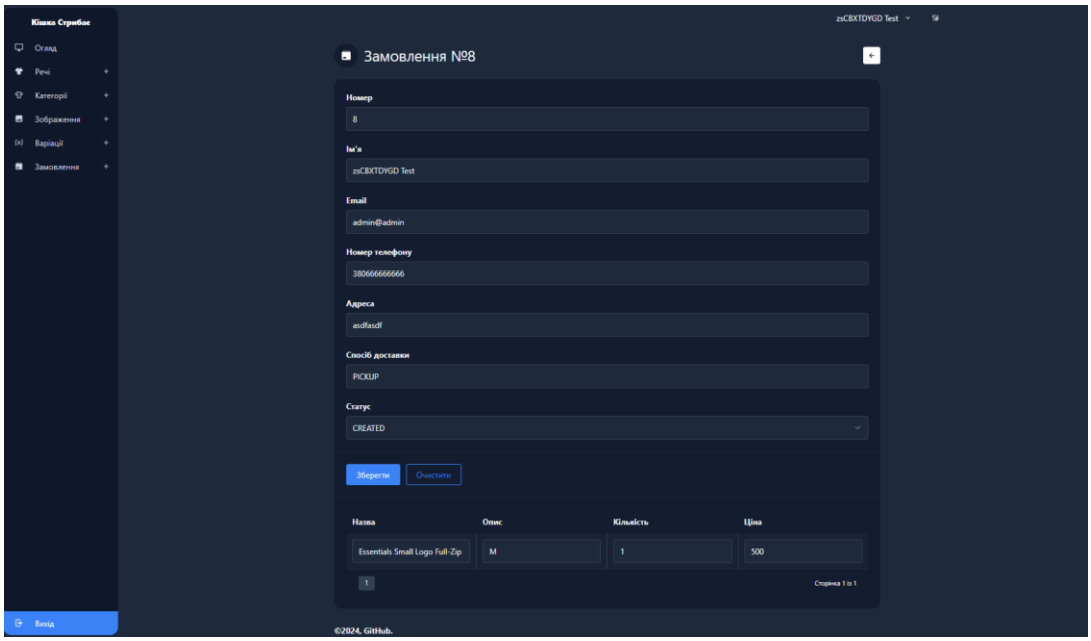

Рисунок 3.26 – Сторінка детальної інформації про замовлення

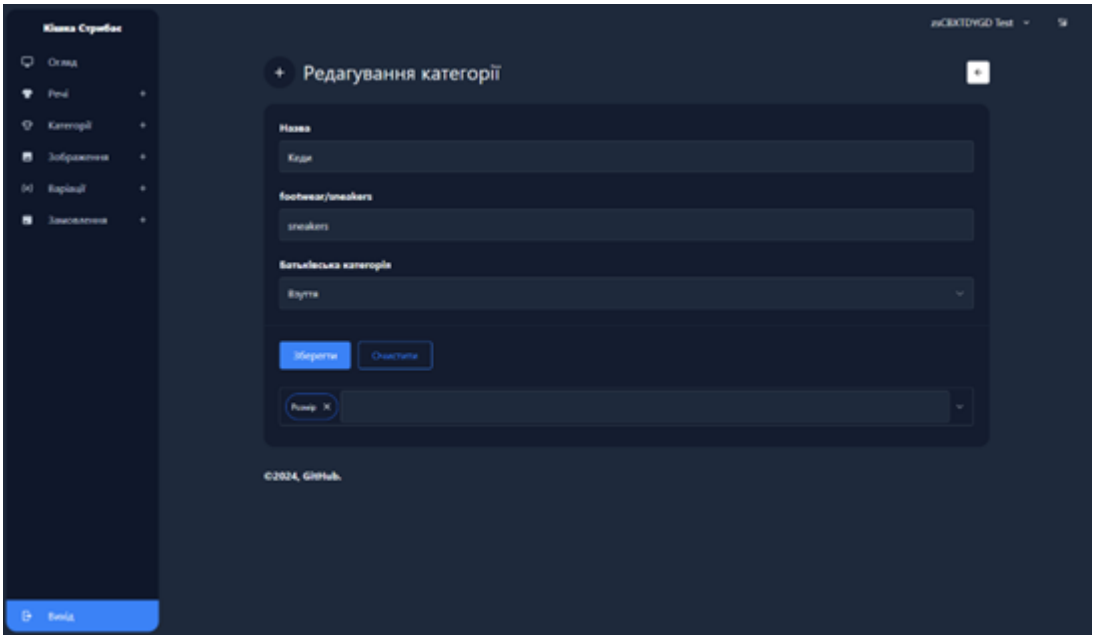

Рисунок 3.27 – Сторінка редагування інформації про категорію

Для адміністраторів також було розроблено сторінку оновлення особистої інформації та паролю (рис. 3.28).

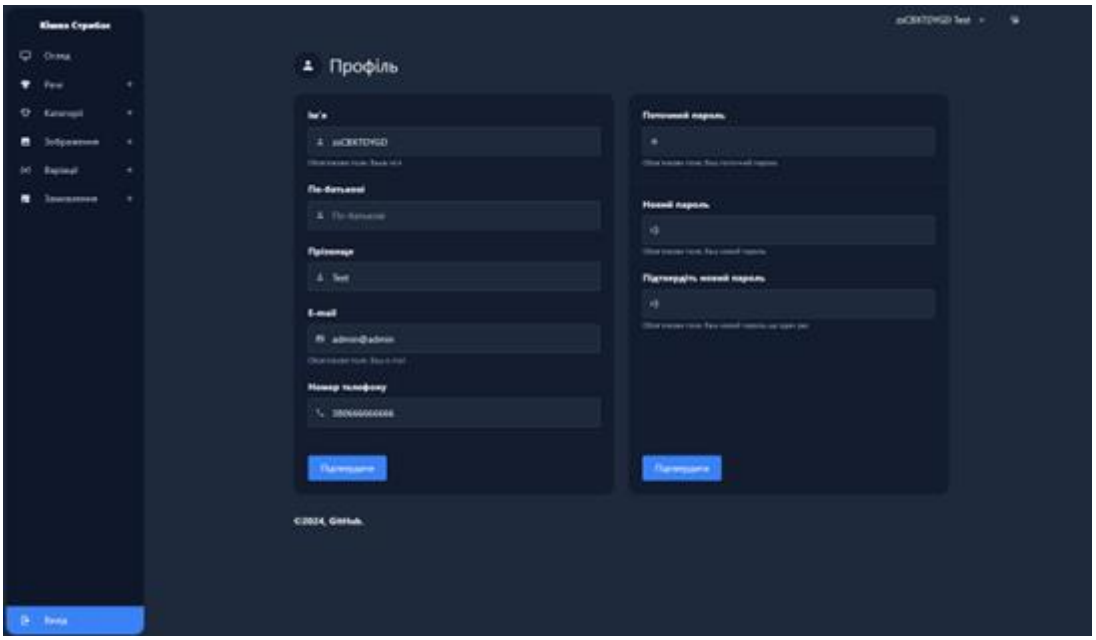

Рисунок 3.28 – Сторінка редагування особистої інформації

Розроблена адміністративна панель надає змогу оновлювати та додавати категорії, товари та варіації товарів, а також переглядати та змінювати статус замовлень, що дозволяє ефективно управляти інтернет-магазином.

3.7 Тестування та підтримка

Тестування є необхідним інструментом для підвищення ефективності, якості та стабільності програмного забезпечення, а також для зниження витрат на довгострокову підтримку. Правильно проведене тестування дозволяє знаходити помилки ще під час розробки – до того, як вони потраплять на робоче середовище. Це дозволяє виправити помилки в коді програмного забезпечення до того, як вони переростуть в проблеми, з якими стикнуться користувачі.

Серверну частину застосунку було протестовано з використанням автоматизованого тестування. Автоматизоване тестування має кілька суттєвих переваг, які роблять його необхідним інструментом для забезпечення якості програмного забезпечення. Автоматизоване тестування дозволяє перевірити певні сценарії за значно менший час, ніж при звичайному ручному тестуванні. Іншою перевагою є те, що такі тести виконуються кожного разу однаково, виключаючи людський фактор, що забезпечує стабільність їх результатів.

Існує декілька типів автоматизованого тестування, і для тестування серверної частини було використано два з них: модульне та інтеграційне тестування.

Модульне тестування – це тип програмного тестування, де перевіряються окремі компоненти програмного забезпечення. Воно виконується під час розробки додатка і полягає у тестуванні окремих компонентів, таких як функції чи процедури. Модульне тестування дозволяє переконатися, що кожен окремий модуль працює правильно. Модульне тестування допомагає перевірити бізнес-логіку та реалізацію, ізолюючи програму від зовнішніх сервісів, таких як бази даних та системи обміну повідомленнями. Однак значна частина коду програми може бути присвячена інтеграції з цими зовнішніми сервісами. Щоб переконатися в повній функціональності програми, слід писати інтеграційні тести разом з модульними.

Інтеграційне тестування – це процес перевірки взаємодії між двома або більше модулями програмного забезпечення. Основна увага приділяється дотриманню інтерфейсу між цими модулями. Мета інтеграційного тестування - виявити помилки в комунікації та взаємодії між інтегрованими модулями. Інтеграційне тестування проводиться після того, як усі модулі пройшли модульне тестування.

Для проведення модульного тестування було використано бібліотеку jUnit5 – поточне покоління фреймворку тестування JUnit, який забезпечує сучасну основу для тестування на стороні розробника для мов JVM. Таким чином було виконано тестування більшості розроблених функцій. У випадках, коли можливе модульне тестування, проте функція також передбачає використання іншого модуля програми, було використано бібліотеку Mockito – бібліотеку, яка дозволяє підміняти модулі на такі, яким можна встановити необхідну поведінку під час тестування. Приклад модульного тесту, який використовує Mockito наведено на рисунку 3.29. В цьому тесті «підставним» є відразу декілька об’єктів: репозиторії варіантів товарів та замовлень, а також

об’єкт, що використовується для «збірки» моделі замовлення, яку буде переслано у відповідь на запит. У даному випадку «підставні» об’єкти симулюють правильну роботу відповідних модулей.

```
@Test
void shouldSave() {
  // given
  ArgumentCaptor<Order> captor = ArgumentCaptor.forClass(Order.class);

  given(productItemRepository.findById(1L)).willReturn(Optional.of(productItem1));
  given(productItemRepository.findById(2L)).willReturn(Optional.of(productItem2));
  given(orderRepository.save(captor.capture())).willReturn(order);
  given(orderModelAssembler.toModel(order)).willReturn(orderDto1);

  // when
  OrderDto result = orderService.save(orderRequestDto);

  // then
  then(captor.getValue()) ObjectAssert<Order>
      .usingRecursiveComparison() RecursiveComparisonAssert<capture of ?>
      .ignoringFields( ...fieldNamesToIgnore: "id", "items") capture of ?
      .isEqualTo(order);
  then(result).usingRecursiveComparison().isEqualTo(orderDto1);
}
```

Рисунок 3.29 – Приклад модульного тесту з використанням Mockito

Для виконання інтеграційного тестування було використано Testcontainers – фреймворк з відкритим вихідним кодом, який дозволяє запускати тести з одноразовими, легкими екземплярами баз даних, чи будь-чого іншого в контейнерах Docker. Даний фреймворк вирішує проблему створення середовища для тестування, за яким необхідно слідкувати: підготовлювати тестові дані та видаляти дані по завершенню тестування. Його було використано для розгортання тестової бази даних, що значно полегшило проведення інтеграційного тестування.

Також, для тестування контролерів було використано бібліотеку RestAssured, яка дозволяє тестувати та перевіряти REST-сервіси, маючи при

цьому легкий для читання синтаксис. Приклад тесту, який використовує RestAssured наведено на рисунку 3.30.

```
@Test
void shouldSaveWhenAdmin() {
  OrderItemRequestDto orderItemRequestDto = new OrderItemRequestDto(orderItemId,  quantity: 1);
  OrderRequestDto orderRequestDto =
      new OrderRequestDto(
          phoneNumber: "380501234567",
          userEmail: "admin@admin",
          customerFullName: "admin",
          PaymentType.CASH,
          address: "address",
          ShippingMethod.PICKUP,
          OrderStatus.CREATED,
          Set.of(orderItemRequestDto));

  given() RequestSpecification
      .auth() AuthenticationSpecification
      .oauth2(token) RequestSpecification
      .body(orderRequestDto)
      .contentType(ContentType.JSON)
      .when()
      .post( s: "/api/orders") Response
      .then() ValidatableResponse
      .statusCode(HttpStatus.CREATED.value())
      .body( s: "totalPrice", equalTo( operand: 100.0F));
}
```

Рисунок 3.30 – Приклад інтеграційного тесту з використанням RestAssured

Для забезпечення виконання тестів перед тим, як застосувати зміни до проєкту, за допомогою сервісу GitHub, в якому зберігається код застосунку, було налаштовано правила ведення гілок, зокрема заборонено внесення змін до основної гілки напряму. Наступним кроком стало створення скриптів для GitHub Actions – інструменту для налаштування CI\CD проєктів. В скрипті було налаштовано запуск автотестів при створенні запиту на злиття гілок, що дозволило проводити автоматичне тестування при завершенні розробки нової функції або після виправлення помилок.

Важливим етапом після проведення автоматизованого тестування є перевірка за іншими показниками – перевірка кількісного показника покриття тестів у новому коді та статична перевірка коду на можливу наявність багів.

Цю проблему було вирішено за допомогою сервісу SonarCloud, до якого відправляється інформація про проведене тестування за допомогою GitHub Actions. SonarCloud аналізує надану інформацію та новий код проєкту та повідомляє про результати перевірки, що дозволяє покращити якість програмного коду ще до того, як він потрапить до виробничого середовища.

Такий підхід дозволяє забезпечити стабільність програмного застосунку та спростити його подальшу підтримку.

ВИСНОВКИ

Було проведено інформаційний огляд та досліджено важливість створення інтернет-магазину. Також було розглянуто та вибрано методи вирішення задач, а саме було обрано необхідні інструменти для кожної з частин інформаційної системи. Для цього було проаналізовано наявні на ринку технології та програмні рішення, які використовуються в електронній комерції.

Однією з ключових частин роботи було проєктування архітектури інтернет-магазину. Це включало огляд існуючих рішень, вибір необхідного підходу проєктування застосунку та створення моделі бази даних.

В ході виконання роботи було виконано такі завдання:

1) Розроблено інформаційну модель;

2) Обрано засоби для програмної реалізації;

3) Програмно реалізовано інтернет-магазин спортивного одягу;

4) Проведено тестування розробленого програмного продукту.

Розроблена система надає бізнесу можливість розширити сферу впливу, залучити нових клієнтів та підвищити ефективність продажів. Крім того, розроблена система є гнучкою та надає можливість створення та редагування сутностей системи, що значно спрощує ведення онлайн-бізнесу, а також система має можливість перегляду статистики основних показників.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Aswar, K., Ermawati. E-Commerce Adoption by Small Medium Enterprises: An Extensive Literature Review. Information Management and Business Review, 12(4(I)). C. 14-16. URL: https://doi.org/10.22610/imbr.v12i4(I).3123 (дата звернення: 15.04.2024).

2. A. Dhanalakshmi, Xu Hui, Roopini. R, R. Supriya. Technological Advancements in E-Commerce and Customer Relationship Management. International Journal of Engineering and Management Research, 10(6). C. 14–15. URL: https://doi.org/10.31033/ijemr.10.6.2 (дата звернення: 15.04.2024).

3. S, Dharmar. "A STUDY ON RECENT TRENDS IN E-COMMERCE". C. 10-12. URL: https://jconsortium.com/index.php/scholar/article/view/121 (дата звернення: 16.04.2024).

4. E-Commerce Worldwide. URL: https://www.statista.com/topics/871/online-shopping/ (дата звернення: 16.04.2024).

5. Michael Keenan. "Global Ecommerce Explained: Stats and Trends to Watch". URL: https://www.shopify.com/enterprise/global-ecommerce-statistics (дата звернення: 16.04.2024).

6. Statista. "Ecommerce - Ukraine | Statista Market Forecast.". URL: https://www.statista.com/outlook/dmo/ecommerce/ukraine (дата звернення: 17.04.2024).

7. Кириченко А. В. "РОЗВИТОК УКРАЇНСЬКОЇ ЕЛЕКТРОННОЇ КОМЕРЦІЇ В КОНТЕКСТІ РОСІЙСЬКО-УКРАЇНСЬКОЇ ВІЙНИ". С. 133-136. URL: https://doi.org/10.30525/978-9934-26-223-4-18 (дата звернення: 17.04.2024).

8. X. Liu, J. Heo, L. Sha. "Modeling 3-tiered web applications," in 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2005. C. 307–310.

9. S. Ibrahim, B. He, and H. Jin, “Towards pay-as-you-consume cloud computing,” in 2011 IEEE International Conference on Services Computing, 2011, С. 370–377.

10. E. Brynjolfsson, P. Hofmann, S. L. i. P. Alto, and J. Jordan, “Cloud computing and electricity: Beyond the utility model: Communications of the acm: Vol 53, no 5,” 2010. URL: https://dl.acm.org/doi/fullHtml/10.1145/1735223.1735234 (дата звернення: 18.04.2024).

11. Angle, I. C. (n.d.). Towards Cloud-Based cost-effective serverless information system. EWU Digital Commons. С. 12-13. URL: https://dc.ewu.edu/theses/780/ (дата звернення: 18.04.2024).

12. Інтернет-магазин Puma. URL: https://us.puma.com (дата звернення: 18.04.2024).

13. Інтернет-магазин Sportcenter. URL: https://sportcenterstore.com.ua (дата звернення: 18.04.2024).

14. Інтернет-магазин Asos. URL: https://asos.com (дата звернення: 18.04.2024).

15. Інтернет-магазин Answear. URL: https://answear.ua/ (дата звернення: 18.04.2024).

16. Інтернет-магазин Zara. URL: https://www.zara.com/ua/ (дата звернення: 18.04.2024).

17. Astro Islands. URL: https://docs.astro.build/en/concepts/islands/

18. Ks-java. URL: https://github.com/pawl1n/ks-java

19. Ks-react. URL: https://github.com/pawl1n/ks-react