КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ

автоматизації і інформаційних технологій

(факультет)

інформаційних технологій

(кафедра)

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО АТЕСТАЦІЙНОЇ ВИПУСКНОЇ РОБОТИ
НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ МАГІСТРА

на тему: «Розробка інтернет-магазину електроніки»

Існюк Владислав Олегович
(прізвище, ім’я та по батькові магістра повністю)

Київ 2023 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ

автоматизації і інформаційних технологій

(факультет)

інформаційних технологій

(кафедра)

ЗАТВЕРДЖУЮ
Завідувач кафедри ІТ

Гончаренко Т.А.
„26” Червня 2023 року

ПОЯСНЮВАЛЬНА ЗАПИСКА ДО АТЕСТАЦІЙНОЇ ВИПУСКНОЇ РОБОТИ НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ МАГІСТРА

на тему: " Розробка системи автоматизованого розрахунку навантажень на нервюри крила БПЛА "

Виконав: студент 2 – го курсу, групи КН-22

Спеціальності: 122 «Комп’ютерні науки» .

(шифр і назва напряму підготовки, спеціальності)

Магістрант Існюк Владислав Олегович .
(прізвище та ініціали)

Керівник Поплавський Олександр Анатолійович.
(прізвище та ініціали)

Рецензент Горда Олена Володимирівна .
(прізвище та ініціали)

Київ, 2023 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ

Факультет: автоматизації і інформаційних технологій .
Кафедра: інформаційних технологій .
Освітній рівень: «магістр за ОПП» .
Спеціальність: 122 «Комп'ютерні науки» .

ЗАТВЕРДЖУЮ
Завідувач кафедри ІТ

Гончаренко Т.О.
„26" Червня 2023 року

З А В Д А Н Н Я
ДО ВИКОНАННЯ АТЕСТАЦІЙНОЇ ВИПУСКНОЇ РОБОТИ
НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ МАГІСТРА

---

1. Тема роботи: Розробка інтернет-магазину електроніки.
затверджена наказом ректора КНУБА № 1280/2 від «26» Червня 2023р.
2. Керівник роботи: Поплавський Олександр Анатолійович.
3. Строк подання студентом роботи до захисту: 7 Грудня 2023 р.
4. Зміст пояснювальної записки за розділами:
Р.1. Класифікація безпілотних літальних апаратів.
Р.2. Типи крил літальних апаратів.
Р.3. Методи розрахунку нервюр крила літального апарату.
Р.4. Проєктування архітектури системи.
Р.5. Програмування системи та контрольний приклад.
5. Інформаційні слайди:
С.1. Алгоритм розв'язку задачі автоматизованого розрахунку навантажень на нервюри крила БПЛА.
С.2. Конструктивно-силові схеми нервюр.

С.3. Конструктивно-силові схеми крила.
С.4. Контрольний приклад.
С.5. Структура програмного комплексу.
С.6. Схема навантаження та врівноваження нервюри.

6. Календарний план виконання атестаційної випускної роботи

Види робіт та їх зміст	Дата виконання
Р.1. Аналіз предметної області та постановка задачі.	01.10.23
Р.2. Математичне забезпечення системи.	28.10.23
Р.3. Проектування бази даних системи.	02.11.23
Р.4. Розробка програмного забезпечення.	14.11.23
Остаточне оформлення роботи	16.11.23
Направлення роботи на рецензування, перевірку на плагіат	27.11.23
Попередній захист роботи на кафедрі	18.12.23

7. Консультанти розділів атестаційної випускної роботи

Розділ	Прізвище, ініціали та посада консультанта	Перевірив	
		дата	підпис
Розділ 1.			
Розділ 2.			
Розділ 3.			
Розділ 4.			
Розділ 5.			

8. Дата видачі завдання: «26» Червня 2023р.

Керівник _____ Поплавський О. А.
(підпис) (прізвище та ініціали)

Магістрант _____ Існюк В. О.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Існюк В.О. «Розробка інтернет-магазину електроніки».

Атестаційна випускна робота магістра за спеціальністю: 122 «Комп'ютерні науки». – Київський національний університет будівництва та архітектури. – Київ, 2023 р.

Атестаційна робота магістра присвячена створенню розробці інтернет-магазину електроніки. У процесі виконання роботи вивчені сучасні тенденції у сфері електронної торгівлі, вибрані технології для розробки веб-сайту та розроблено зручний інтерфейс для користувачів.

Розглянуті аспекти безпеки та конфіденційності даних, врахування юридичних аспектів, а також розробка стратегій маркетингу для привертання цільової аудиторії. Результатом є функціональний інтернет-магазин електроніки, готовий до конкуренції на ринку електронної торгівлі.

Ключові слова: інтернет-магазин, електроніка, електронна комерція, веб-сайт, безпека даних, маркетинг.

ANNOTATION

Isniuk V.O. "Development of an online electronics store"

Attestation thesis for the master's degree in the specialty: 122 " Computer Science". – Kyiv National University of Construction and Architecture. – Kyiv, 2023.

The master's thesis is devoted to the creation of an online electronics store. In the process of performing the work, modern trends in the field of e-commerce were studied, technologies for website development were selected, and a user-friendly interface was developed.

We also considered aspects of data security and privacy, legal aspects, and developed marketing strategies to attract the target audience. The result is a functional online electronics store ready to compete in the e-commerce market.

Keywords: online store, electronics, e-commerce, website, data security, marketing.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

БД – База даних

ПЗ – Програмний застосунок

API – Application Programming Interface

CLI – Command Line Interface

CSS – Cascading Style Sheets

DOM – Document Object Model

HTML – Hyper Text Markup Language

IDE – Integrated Development Environment

JS – JavaScript

JSX – JavaScript XML

MVC – Model View

ВСТУП

У наш час люди все менше користуються реальними магазинами, віддаючи перевагу онлайн сервісам надання послуг. Тому усі компанії активно переносять свій бізнес у простори інтернету.

На сьогоднішній день яскраве зображення не значного розміру зазиває потенційного клієнта скористатись послугами компанії, а привабливий опис, зручний та привабливий дизайн, зрозумілий інтерфейс надання та отримання бажаних послуг спонукає користувача повертатися до сервісу знову. Для цього компанії витрачають значні кошти на спеціалістів цифрової галузі, таких як програмістів, Web-дизайнерів, SEO спеціалістів.

Саме це стало причиною створення веб-додатку інтернет магазину побутової техніки. З переваг якого буде унікальний, зручний інтерфейс, за допомогою якого користувачі зможуть швидко орієнтуватись в системі та швидко підбирати необхідний товар. Оформлення замовлення буде дуже простим, потрібно буде лише визначитися з товарами, додати їх до кошика, натиснути оформити замовлення та заповнити дуже просту форму, у якій потрібно вказати свої контактні данні. Через деякий час з клієнтом зв’яжеться менеджер компанії.

У веб-додатку повинна бути зручна панель адміністратора, для полегшення прийому, підтвердження замовлень та редагування товарів у базі.

Веб-додаток повинен робити вибір та замовлення відповідних товарів більш зручним ніж у компаній конкурентів.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИБІР ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ

1.1. Аналіз предметної області

У сучасному світі, де технології на високому рівні розвитку, існує розмаїття методів продажу товарів: традиційний ринок, фізичний магазин, онлайн-платформи та інтернет-магазини. З глобальної точки зору, можна виділити два основних напрямки в продажах:

- онлайн;
- офлайн.

Кожен з цих підходів має свої переваги та недоліки. Спочатку може здатися, що все просто: фізично виходите в магазин, розглядаєте товари на полицях, берете їх в руки. У випадку з інтернет-магазином, ви взаємодієте з веб-додатком та каталогами. Проте це лише одна з відмінностей між ними.

Бізнес-процеси для онлайн та офлайн магазинів в основному аналогічні. Власник розробляє бізнес-план, відкриває магазин, закуповує товари для продажу та взаємодіє з клієнтами. Крім того, він має вирішувати проблеми з організацією реклами, проведенням маркетингових компаній, орендою складських приміщень, розрахунками і сплатою податків і так далі.

Інтернет-магазин не обов'язково включає особистий контакт з клієнтами, але це не означає відсутність комунікації. Розсилки листів на електронну пошту, телефонні дзвінки, СМС та чати в месенджерах активно використовуються в інтернет-магазинах, часто автоматизовано. Використання чат-ботів, які імітують діалог із продавцем, розширює можливості впливу на покупця та спонукає його здійснити більше покупок.

Переваги інтернет-магазинів включають можливість здійснювати торгівлю в електронному форматі з мінімальними накладними витратами. У першій фазі можна обійтися і без найму працівників та складських приміщень. Популярний метод дропшопінгу дозволяє виступати посередником між покупцем і іншими магазинами, отримуючи винагороду за послуги.

Крім того, інтернет-магазини дозволяють власникам ефективно контролювати дані щодо проданих товарів, реклами тощо, надаючи цінну аналітику. Товари можуть бути доступні за більш прийнятними цінами, завдяки економії на витратах, і безкоштовна доставка стає додатковим стимулом для покупців.

У світі, де швидкість стає ключовим фактором для багатьох покупців, інтернет-магазини мають перевагу, оскільки покупки можна здійснювати за допомогою смартфона в будь-який час, коли зручно, відмінно від традиційного графіку магазинів. Географічні обмеження стають менш суттєвими, адже покупці можуть здійснювати покупки з будь-якого регіону, а кур'єрські служби доставляють товар навіть у віддалені куточки.

Анонімність покупців є важливим аспектом для багатьох, особливо тих, хто соромиться придбати певні товари в реальних магазинах. В інтернеті покупець може залишатися невпізнаним, а на посилці буде вказано лише зміст, не розкриваючи особисту інформацію.

З іншого боку, традиційні магазини також мають свої переваги. Можливість взяти товар в руки, випробувати його та оцінити якість надає унікальний досвід, який інтернет-магазини не можуть повністю забезпечити. Отримання товару одразу після покупки, без очікування доставки, може бути критичним у деяких випадках.

Повернення товару та обмін в звичайних магазинах можуть бути більш швидкими та простими порівняно з онлайн-магазинами, де цей процес може

займати більше часу через відсутність фізичної локації. Гарантії та обслуговування після покупки також можуть бути більш ефективно забезпечені у звичайних магазинах.

Універсальність вибору формату покупок залишається в сфері особистого вибору кожного покупця. Багатофакторність і відмінності між онлайн та офлайн методами продажу дають можливість кожному вибрати той, який відповідає його потребам та уподобанням.

1.2. Аналіз веб-додатків аналогів

1.2.1. Інтернет-магазин електроніки та комп'ютерної техніки «ITbox»

ITbox – Український магазин електроніки та комп'ютерної техніки техніки що має свій веб-додаток (показано на рис. 1.1).

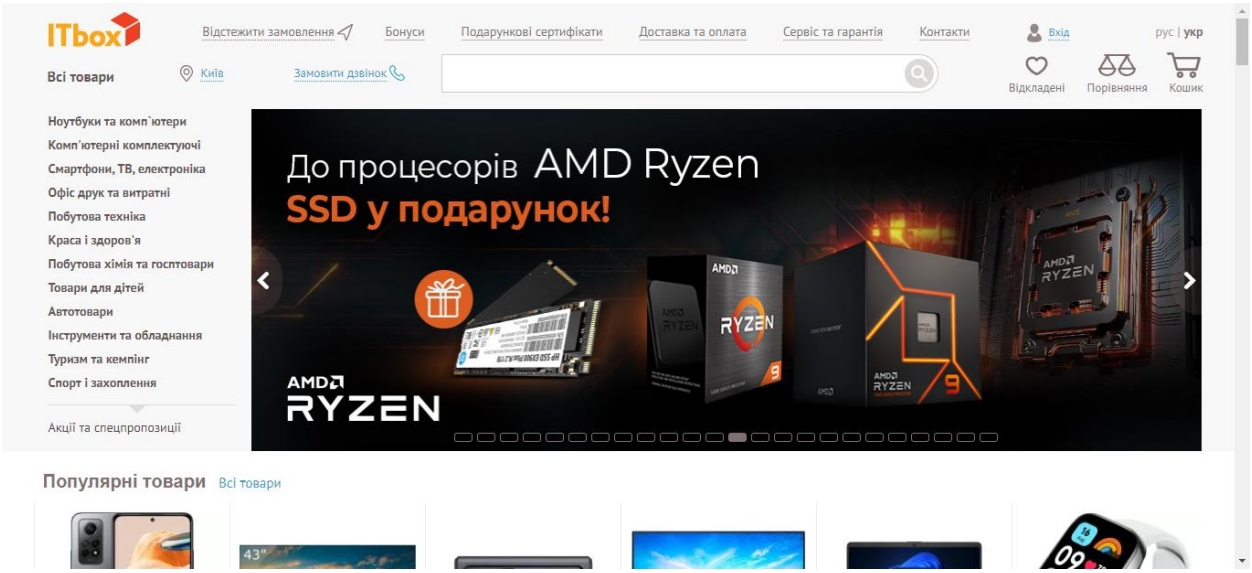

Рисунок 1.1 – Інтернет-магазин побутової техніки «ITbox»

Веб-додаток дозволяє переглядати товар магазину, відфільтрувавши його за категоріями, що дуже зручно. Також є можливість обрати ціновий

діапазон товару, бренд, та інші критерії фільтрації. Веб-додаток дозволяє обрати товар, додати його до корзини, а також оформити заказ.

Із недоліків: не дуже привабливий зовнішній вигляд сайту та незручність використання. В деяких частинах додатку дизайн інтуїтивно незрозумілий.

1.2.2. Byttechnika – інтернет-магазин побутової техніки.

Веб-додаток магазину виглядає дуже привабливо, має зручний інтерфейс (рис 1.2). Функціонал дозволяє обрати певну категорію товару, відфільтрувати його. Магазин надає послуги гарантії та доставки.

Але фільтрація товару виглядає дуже «зім'ято», та не зрозуміло.

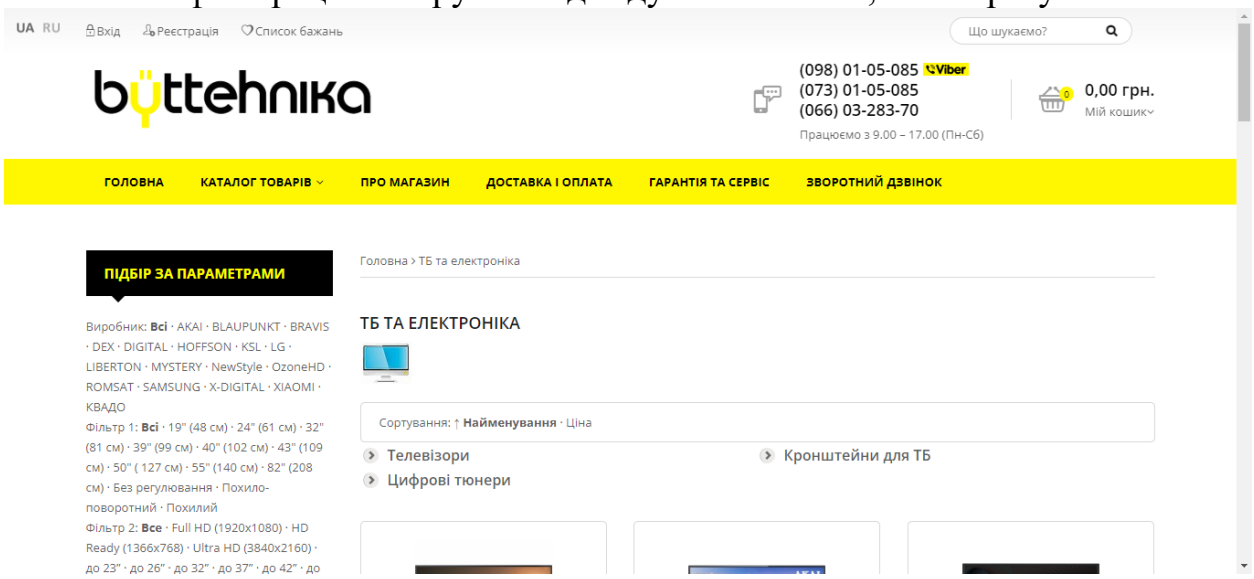

Рисунок 1.2 – Ітерфейс веб-додатку Byttechnika

1.2.3. Інтернет-магазин XCOM

XCOM - магазин не тільки побутової техніки, а також комп'ютерної, портатиної, авто-електроніки, тощо (рис. 1.3). Магазин має великий вибір товару, але дизайн веб-додатку дуже не зручний та інтуїтивно не зрозумілий. Багато товарів відображається не коректно. Зображення товару на сторінці каталогу перекриває опис товару, що також виглядає дуже не професійно.

Веб-додаток має інтегрований чат, для зв’язку з продавцем, що є великою перевагою.

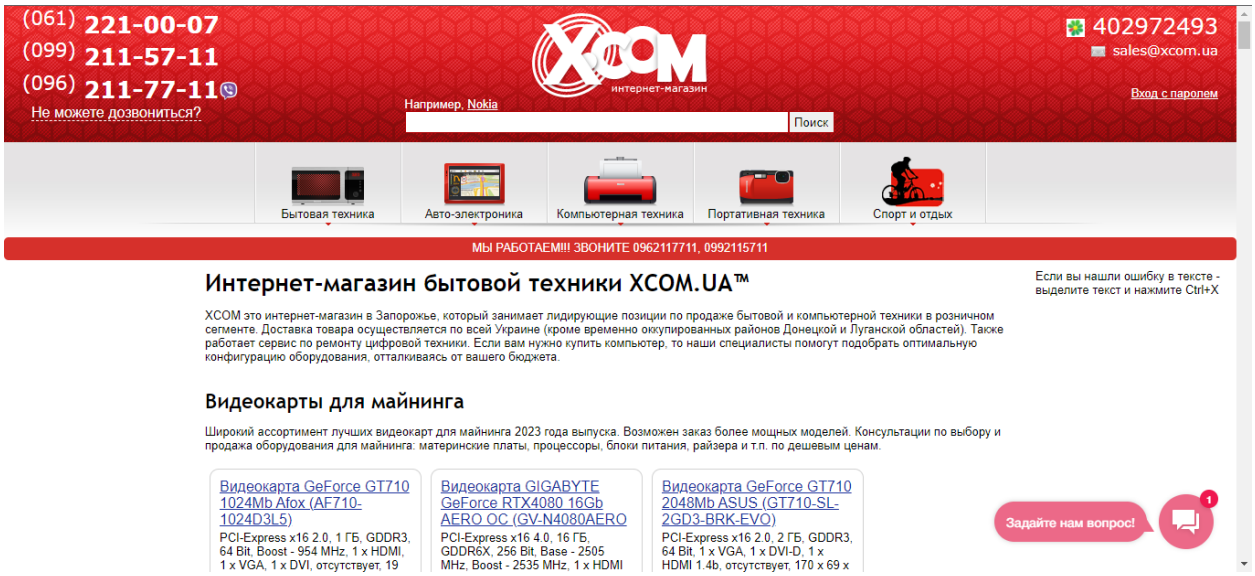

Рисунок 1.3 - Інтернет магазин XCOM

РОЗДІЛ 2. Математичне забезпечення системи

2.1. Реляційні бази SQL

Реляційна база даних, відома також як база даних SQL, розроблена в Structured Query Language (SQL) і представляє собою структурований спосіб зберігання даних, схожий на телефонну книгу. Започаткована IBM у 1970-х роках, реляційна база даних включає дві або більше таблиць із стовпцями та рядками. Кожен рядок відображає запис, а кожен стовпець містить конкретний тип інформації, такий як ім’я, адреса та номер телефону. Взаємозв'язки між таблицями та типами полів визначають схему. У реляційній базі даних схема повинна бути чітко визначена перед введенням будь-яких даних.

Для забезпечення ефективності реляційної бази даних, зберігані дані повинні мати структурований вигляд. Грамотно розроблена схема зменшує зайвість даних та запобігає розсинхронізації таблиць, що є ключовою функцією для багатьох підприємств, особливо тих, що реєструють фінансові операції. Невдало розроблена схема може спричинити помилки в організації через її негнучкість. Наприклад, для стовпця, призначеного для зберігання телефонних номерів, може бути встановлено обмеження в 10 цифр, що враховує стандарт для телефонних номерів в Україні. Це дозволяє відхиляти будь-які недійсні значення (наприклад, якщо в номері відсутній код міста). Але при необхідності змінити схему (наприклад, додати можливість введення міжнародного телефонного номера, який містить більше 10 цифр), потрібно редагувати всю базу даних.

Мова структурованих запитів (SQL) використовується для проектування реляційних баз даних у системах, таких як MySQL, Sybase, Oracle або IBM DM2. В SQL ви можете виконувати запити, отримувати дані та редагувати дані, оновлюючи, видаляючи або додаючи нові записи. SQL є декларативною мовою, яка полегшує взаємодію з реляційною базою даних,

діючи як версія сценарію бази даних на стороні сервера. Однією з ключових переваг SQL є його простий, але потужний оператор JOIN, який дозволяє розробникам отримувати відповідні дані з декількох таблиць за допомогою одного запиту.

2.2. Нереляційні бази NoSQL

Технології NoSQL з'явилися ще в 1960-х і носили різні назви, але нинішній сплеск їх популярності пов'язаний з тим, що картина даних змінюється, а організатори повинні адаптуватися до величезних обсягів і масивів даних, що надходять із хмари, мобільних пристроїв, соціальних мереж і систем обробки великих даних.

Якщо вимоги до даних не зрозумілі з самого початку, або якщо мається справа з величезними обсягами неструктурованих даних, може не бути можливості створювати реляційну базу даних з чітко визначеною схемою. Використовуються нереляційні бази даних, які пропонують набагато більшу гнучкість, ніж їхні традиційні аналоги. Нереляційні бази даних, більше схожі на папки файлів, що збирають відповідну інформацію всіх типів. Якщо в блозі WordPress використовується база даних NoSQL, кожен файл може зберігати дані для публікації в блозі: соціальні лайки, фотографії, текст, показники, посилання.

Неструктуровані дані з Інтернету можуть включати дані датчіків, соціальний доступ, особисті налаштування, фотографії, інформацію на основі місцезнаходження, діяльність в Інтернеті, показники використання тощо. Спроба зберігати, обробляти та аналізувати всі ці неструктуровані дані призвела до розробки без схем альтернатив SQL. У сукупності ці альтернативи називаються NoSQL, що означає "Не тільки SQL". Хоча термін NoSQL охоплює широкий спектр альтернатив реляційним базам даних, загальним для них є те, що вони дозволяють обробляти дані більш гнучко.

Замість таблиць бази даних NoSQL орієнтовані на документи. Таким чином, неструктуровані дані (такі як статті, фотографії, дані в соціальних мережах, відео чи вміст у дописі блогу) можуть зберігатися в одному документі, який можна легко знайти, але не обов'язково класифікувати на такі поля, як робить реляційна база даних. Для масового зберігання даних потрібні додаткові зусилля з обробки та більше місця, ніж високоорганізовані дані SQL. Ось чому Hadoop, платформа обчислень та аналізу даних з відкритим кодом, здатна обробляти величезні обсяги даних у хмарі, популярна у поєднанні зі стеками баз даних NoSQL.

Бази даних NoSQL пропонують ще одну велику перевагу, особливо для розробників додатків: простоту доступу. Реляційні бази даних пов'язані з програмами, написаними об'єктно-орієнтованими мовами програмування, такими як Java, PHP та Python. Бази даних NoSQL часто можуть уникнути цієї проблеми за допомогою API, які дозволяють розробникам виконувати запити без необхідності вивчати SQL або розуміти основну архітектуру їхньої системи баз даних.

2.3. Порівняння SQL та NoSQL

Щодо технологій баз даних, у цій області не існує універсального рішення. Тому багато підприємств використовують як реляційні, так і нереляційні бази даних для вирішення різних завдань. Незважаючи на те, що бази даних NoSQL завойовують популярність завдяки своїй швидкості та масштабованості, існують ситуації, коли високоструктурована база даних SQL може бути виправданою.

Ось декілька причин вибору бази даних SQL:

– Якщо необхідно забезпечити відповідність ACID (атомарність, узгодженість, ізольованість, довговічність). ACID забезпечує цілісність бази

даних та захищає від аномалій, що є важливим для багатьох електронних комерцій та фінансових додатків.

– Якщо дані структуровані та стали. Для бізнесу, який не масштабується і працює лише з послідовними даними, може виправдано використання системи з чіткою структурою.

– В той час як усі інші компоненти серверної програми розроблені для ефективної та безперебійної роботи, бази даних NoSQL не утруднюють обробку великих обсягів даних. Вони можуть працювати з великими об'ємами даних, що робить їх популярними для застосувань, таких як MongoDB, CouchDB, Cassandra та HBase.

Щодо вибору бази даних NoSQL:

– Для зберігання великих обсягів неструктурованих даних. Бази даних NoSQL дозволяють зберігати різні типи даних без необхідності завчасної дефініції їх структури.

– Для максимального використання хмарного зберігання та обчислень. NoSQL дозволяють ефективно масштабувати обробку даних в хмарі.

– З метою швидкого розвитку. Якщо розробка вимагає частих змін структури даних без великих простоїв, база даних NoSQL є зручним вибором.

2.4. Огляд існуючих реляційних СУБД

Oracle - це об'єктно-реляційна система управління базами даних (СУБД), розроблена на мові програмування C++. Якщо ви шукаєте рішення з повним спектром послуг та надійністю при наявності відповідного бюджету, Oracle може бути оптимальним вибором. Поточна версія Oracle

спеціально призначена для використання в хмарних середовищах і може бути розгорнута на одному або кількох серверах, що дозволяє ефективно управляти великими обсягами даних, включаючи мільярди записів. Деякі з інноваційних функцій останньої версії Oracle включають в себе використання grid framework та взаємодію як з фізичними, так і з логічними структурами.

Це означає, що управління фізичними даними не має впливу на доступ до логічних структур. При цьому забезпечено високий рівень безпеки, оскільки кожна транзакція ізольована від інших.

Sybase - це серверна реляційна система управління базами даних, яка в основному призначена для підприємств і активно використовується в операційних системах Unix. Ця СУБД була однією з перших, що підтримує Linux на рівні підприємства.

MS SQL Server - продукт корпорації Microsoft, розроблений для управління базами даних на рівні підприємства, і підтримує як SQL, так і NoSQL архітектури. Особливість роботи сервера баз даних SQL полягає в транзакційній обробці даних, де кожен запит обробляється та зберігається з мінімальним обсягом інформації. Використання SQL Server дозволяє автоматизувати рішення різних бізнес-задач, підтримувати аналітику даних в режимі онлайн та ефективно управляти транзакціями.

PostgreSQL - це об'єктно-реляційна СУБД на рівні підприємства, яка використовує процедурні мови, такі як Perl та Python. Двигун бази даних може бути розміщений в різних середовищах, включаючи віртуальні, фізичні та хмарні. Остання версія PostgreSQL підтримує обробку великих обсягів даних та збільшення кількості одночасно працюючих користувачів. Підвищена безпека досягнута завдяки підтримці DBMS_SESSION.

SQLite - це компактна вбудована СУБД. Двигун SQLite не є окремим процесом; він представляє собою бібліотеку, яку програма використовує для компіляції і яка стає невід'ємною частиною програми. Вся база даних,

включаючи визначення, таблиці, індекси і дані, зберігається в єдиному стандартному файлі на комп'ютері, де виконується програма. Декілька процесів або потоків можуть читати дані з однієї бази даних одночасно без проблем. Запис в базу даних можливий лише в разі відсутності інших запитів, що обслуговуються; в іншому випадку спроба запису завершиться невдачею, і в програму буде повернуто код помилки. Також можна встановити таймаут операцій, щоб підключення, зіткнувшись із зайнятістю БД, чекало кілька секунд перед тим, як отримати помилку SQLITE_BUSY.

2.5. Порівняння PostgreSQL, SQLite

Бібліотека SQLite відзначається своєю компактністю, маючи розмір менше 600 КБ, і вважається однією з найменших у цьому переліку. Ця особливість робить її ідеальною для вбудованих пристроїв. Це особливо ефективно використовується в сценаріях з низьким та середнім обсягом трафіку, наприклад, на веб-сайтах із приблизно 100 тисячами запитів на день.

Однією з ключових переваг SQLite є його можливість діяти як додаткове рішення для клієнт-серверних систем управління базами даних. Наприклад, вона може кешувати дані з цих систем локально, що дозволяє зменшити час очікування запитів та забезпечує незалежність від корпоративних систем у випадку відключення.

При розгляді можливостей паралельної обробки, PostgreSQL виділяється, займаючи перше місце (порівняно з MySQL), особливо при виконанні тривалих запитів SELECT у аналітичних додатках. PostgreSQL завжди вважалася найбільш підходящою для аналітичних процесів, таких як зберігання даних. Прикладом є БД Timescale, яка може обробляти вставку 1 мільйона записів в секунду. Ця база даних також успішно використовується в OLTP-додатках та в фінансовій галузі завдяки суворому дотриманню

вимог ACID. Розширюваність PostgreSQL робить її ідеальним вибором для наукових та дослідницьких проектів.

Отже, з урахуванням всіх перелічених факторів, було прийнято рішення вибрати СУБД PostgreSQL.

2.6. Огляд мов програмування

2.6.1. Мова розмітки - HTML та CSS

HTML - це мова розмітки гіпертексту, призначена для створення і структурування розділів, параграфів, заголовків, посилань і блоків для веб-сторінок і додатків. HTML не є мовою програмування і не має можливості створювати динамічні функції; замість цього вона дозволяє організовувати і форматувати документи, аналогічно Microsoft Word.

При роботі з HTML використовуються прості структури коду, такі як теги і атрибути, для розмітки сторінки веб-сайту. HTML теги - це спеціальні команди для браузера, які вказують, як інтерпретувати вміст, наприклад, визначаючи, що текст слід розглядати як заголовок чи абзац.

Теги в HTML побудовані за принципом відкриваючий куточок, ім'я тега, а потім закриваючий куточок, наприклад, <ім'я тега>. Ім'я тега може містити англійські букви і цифри. Приклади тегів: <h1>, <p>, <b>. Теги зазвичай пишуться парами - відкриваючий тег і відповідний йому закриваючий. Різниця між ними полягає в тому, що в закриваючому тезі перед закриваючим куточком ставиться слеш, як у <ім'я тега/>.

CSS (Cascading Style Sheets, каскадні таблиці стилів) - це мова опису зовнішнього вигляду HTML-документа. Ця технологія є необхідною для створення сучасних веб-сайтів, оскільки вона взаємодіє з HTML, надаючи документу вигляд. HTML структурує документ і впорядковує інформацію, а CSS взаємодіє з браузером, щоб задати стиль та оформлення. Ці дві мови працюють в єдиній зв'язці для створення зручних і естетичних веб-сторінок.

2.6.2. JavaScript

При генерації сторінок веб-сайтів виникає дилема, пов'язана з архітектурою «клієнт-сервер». Сторінки можна генерувати як на стороні клієнта, так і на стороні сервера. У 1995 році фахівці компанії Netscape створили механізм управління сторінками на клієнтській стороні, розробивши мову програмування JS. Таким чином, JavaScript - це мова керування сценаріями перегляду гіпертекстових сторінок веб-сайтів на стороні клієнта.

Основна ідея JavaScript полягає в можливості зміни значень атрибутів HTML-контейнерів і властивостей середовища відображення в процесі перегляду HTML-сторінки користувачем. Перезавантаження сторінки не відбувається.

JavaScript - це мова програмування, яка:

має високий рівень: забезпечує абстракції, які дозволяють ігнорувати деталі машини, де вона працює. Автоматично керує пам'яттю за допомогою збирача сміття, тому можна зосередитись на коді, а не керувати пам’яттю, як це було б потрібно іншим мовам, як C, і надає безліч конструкцій, які дозволяють мати справу з надзвичайно потужними змінними та об’єктами;

динамічна: на відміну від статичних мов програмування, динамічна мова виконує під час виконання багато речей, які робить статична мова під час компіляції. У цьому є плюси і мінуси, і це дає нам потужні функції, такі як динамічне введення тексту, пізнє прив'язування, відображення, функціональне програмування, зміна часу виконання об'єктів, закриття та багато іншого;

динамічно набрана: змінна не застосовує тип. Можна перепризначити змінну будь-якого типу, наприклад, присвоївши ціле число змінній, що містить рядок;

інтерпретується: це загальновідома мова як інтерпретована, що означає, що їй не потрібен етап компіляції перед запуском програми, на

відміну від C, Java або Go, наприклад. На практиці браузери компілюють JavaScript перед його виконанням з міркувань продуктивності, але це виглядає прозоро - додаткових кроків не передбачено;

мультипарадигма: мова не застосовує жодної конкретної парадигми програмування, на відміну від Java, наприклад, яка змушує використовувати об'єктно-орієнтоване програмування, або C, що змушує імперативне програмування. Можна писати JavaScript за допомогою об’єктно-орієнтованої парадигми, використовуючи прототипи та новий (на ES6) синтаксис класів. Можна писати JavaScript у функціональному стилі програмування з його першокласними функціями або навіть у імперативному стилі (подібному до C).

2.6.3. Огляд існуючих фреймворків

Vue.js - це інноваційний фреймворк, придатний для створення інтерфейсів користувача. Назва цього фреймворка взаємодіє з поняттям "view" (вигляд), що вказує на його спрямованість на модель-вигляд-контролер (MVC).

Vue.js спеціалізується на рівні представлення і легко інтегрується з іншими бібліотеками та проектами. Це означає, що Vue.js можна поступово впроваджувати в існуючі проекти, розширюючи їхню функціональність. Ця прогресивність робить Vue.js відмінним від інших фреймворків.

Простота інтеграції є однією з ключових переваг Vue.js, особливо в поєднанні з можливістю взаємодії з бекенд-фреймворками. Ще однією сильною стороною є його легкість освоєння, чудова документація і висока продуктивність.

Vue.js ідеально підходить для створення односторінкових додатків. Основна концепція фреймворка полягає в використанні компонентів - невеликих частин інтерфейсу, які можна повторно використовувати. Таким

чином, додаток сам складається з компонентів, які можуть включати інші компоненти, утворюючи деревоподібну ієрархію.

AngularJS - це структурований фреймворк для динамічних веб-додатків. Він дозволяє використовувати HTML як мову шаблонів та розширювати його синтаксис для зручного програмування. З використанням Data-binding і Dependency injection, AngularJS допомагає зменшити кількість коду, працюючи як з браузерним, так і серверним JavaScript.

Angular визначається як те, чим би був HTML, якщо він був би створений для додатків. ReactJS, створений розробниками Facebook, постійно працює з DOM, ефективно перемальовуючи його при змінах умов. Його використання JSX, надбудови на JavaScript, дозволяє використовувати синтаксис подібний до XML в коді JavaScript, спрощуючи спостереження за змінами в одному місці під час розробки.

ReactJs - це JS фреймворк, представлений розробниками Facebook. Існує багато різних за між собою думок з приводу користі і доцільності використання даного продукту.

ReactJs постійно працює з DOM, перемальовуючи його при зміні умов (та частина DOM, яку змінює ReactJs, називається компонентом). Раніше подібна практика сильно б відбилася на продуктивності додатка, але розробники ReactJs підійшли до вирішення даного питання кардинально: вони повністю переписали DOM на Javascript.

Важливою особливістю ReactJs є використання JSX. Це надбудова на JS, що дозволяє використовувати про-XML синтаксис в Javascript коді. JSX - це поєднання javascript і html, які в зв'язці є незвичним синтаксисом для більшості розробників. Стандартом вважається поділ JS частини від розмітки, що ускладнює спостереження за змінами HTML, JS, HTML. JSX дозволяє бачити всі процеси в одному місці, не відволікаючись на складності грамотного і валідного коду. Після компіляції JSX виходить чистий JS.

2.7. Порівняння та вибір фреймворку

Переваги Angular:

– Angular-language-service - забезпечує інтелектуальні можливості і автозаповнення шаблону HTML-компонента;

– нові функції, такі як generation Angular, що використовують бібліотеки npm з CLI, generation, і розробка компонентів, що використовує Angular;

– Angular використовується разом з Typescript. Він має виняткову підтримку для цього;

– детальна документація, що дозволяє розробнику отримати всю необхідну інформацію, не вдаючись до допомоги його колег. Однак це вимагає більше часу для навчання;

– одностороння прив'язка даних, яка забезпечує виняткову поведінку додатка, що зводить до мінімуму ризик можливих помилок;

– впровадження залежностей від компонентів, пов'язаних з модулями і модульність в цілому;

– структура і архітектура, спеціально створені для великої масштабованості проекту;

– MVVM (Model-View-ViewModel), яка дозволяє розробникам працювати окремо над одним і тим же розділом програми, використовуючи один і той же набір даних.

Переваги VueJs:

– посилений HTML. Це означає, що Vue.js має багато характеристик схожих з Angular, а це, завдяки використанню різних компонентів, допомагає оптимізації HTML- блоків;

– детальна документація. Vue.js має дуже детальну документацію, яка може прискорити процес навчання для розробників і заощадити багато

часу на розробку програми, використовуючи тільки базові знання HTML і JavaScript;

– адаптивність. Може бути здійснений швидкий перехід від інших фреймворків до Vue.js через схожість з Angular і React з точки зору дизайну і архітектури;

– чудова інтеграція. Vue.js можна використовувати як для створення односторінкових додатків, так і для більш складних веб-інтерфейсів додатків. Важливо, що невеликі інтерактивні елементи можна легко інтегрувати в існуючу інфраструктуру без негативних наслідків;

– масштабування. Vue.js може допомогти в розробці досить великих шаблонів багаторазового використання, які можуть бути зроблені майже за той же час, що і більш прості;

– не значний розмір. Vue.js важить близько 20 КБ, зберігаючи при цьому свою швидкість і гнучкість, що дозволяє досягти набагато кращої продуктивності в порівнянні з іншими платформами.

Переваги ReactJS:

– легко вивчити, завдяки простому дизайну, використання JSX (HTML- подібний синтаксис) для шаблонів і дуже докладної документації. Розробники витрачають більше часу на написання сучасного JavaScript і менше турбуються про код, специфічному для фреймворка;

– дуже швидка, завдяки реалізації React Virtual DOM і різним оптимізаціям рендеринга;

– відмінна підтримка рендеринга на стороні сервера, що робить його потужною платформою для контент-орієнтованих додатків.

– першокласна підтримка Progressive Web App (PWA) завдяки генератору додатків `create-react-app`;

– прив'язка даних є односторонньою, що означає менше небажаних побічних ефектів;

– Redux, найпопулярніша платформа для управління станом додатків в React, її легко вчити і використовувати;

– React реалізує концепції функціонального програмування (FP), створюючи простий в тестуванні і багаторазово використовуваний код;

– додатки можуть бути створені за допомогою TypeScript або Facebook's Flow, що мають вбудовану підтримку JSX;

– перехід між версіями, як правило, дуже простий: Facebook надає

– «кодові модулі» для автоматизації більшої частини процесу.

– Отже, виходячи з усіх вище перерахованих факторів, було вирішено обрати фреймворк ReactJs.

2.8. Вибір засобів розробки серверної частини

Для розробки серверної частини додатку була обрана мова програмування JavaScript за допомогою середовища виконання NodeJs. В якості фреймворку NodeJs використовувався ExpressJs. Для взаємодії з базою даних на сервері використовувалась ORM-бібліотека Sequelize.

Node.js - це крос-платформне середовище виконання JavaScript з відкритим вихідним кодом, яке виконує код JavaScript поза межами браузера. Головним чином JavaScript використовується для створення клієнтських сценаріїв, які вбудовуються в HTML-код веб-сайту і виконуються в браузері.

Node.js дозволяє розробникам використовувати JavaScript для створення інструментів командного рядка. На стороні сервера він виконує необхідні сценарії для обробки динамічного вмісту веб-сторінки перед тим, як вона стане доступною у браузері користувача. Таким чином, Node.js втілює ідею "JavaScript всюди", дозволяючи розробляти веб-додатки на одній мові як для серверних, так і для клієнтських сценаріїв.

Express.js - це простий і швидкий фреймворк для Node.js, який використовується як проміжний обробник для управління серверами і маршрутами.

Express.js підходить для створення простих додатків, які можуть обробляти декілька запитів одночасно і використовують можливості Express.
Для цього фреймворку існує багато докладних інструкцій та описів, складених розробниками, які перевірили його ефективність на практиці.
Основна особливість Express полягає в тому, що він надає мінімальний обсяг базового функціоналу. Всі інші необхідні функції можна додати за допомогою зовнішніх модулів. Фактично, Express в чистому вигляді - це сервер і він може працювати без жодного модуля.
Завдяки цьому мінімалізму розробник відразу отримує легкий і швидкий інструмент, який можна розширювати і розвивати. Тому рекомендується починати роботу з Express, якщо ви маєте намір навчитися створювати додатки на платформі Node.js.
Sequelize - це ORM-бібліотека для додатків на Node.js, яка виконує зіставлення таблиць в базі даних і відносин між ними з класами. При використанні Sequelize можна уникнути написання SQL-запитів і працювати з даними як зі звичайними об'єктами.

ORM, або Object-relational mapping (Об'єктно-реляційне відображення), є технологією програмування, яка дозволяє перетворювати несумісні типи моделей в ООП, зокрема між сховищем даних і об'єктами програмування. ORM спрощує процес збереження об'єктів в реляційну базу даних і їхнє отримання, при цьому сама піклується про перетворення даних між двома несумісними станами. Більшість ORM-інструментів значною мірою покладаються на метадані бази даних і об'єктів, так що об'єктам не потрібно знати про структуру бази даних, а базі даних - про структуру даних в додатку. ORM забезпечує повне розділення завдань в добре спроектованих додатках, де і база даних, і додаток можуть працювати з даними у своїх вихідних формах.

РОЗДІЛ 3. Проектування бази даних системи

3.1 Створення прототипу веб-застосунку

Розпочаток розробки веб-застосунку включав у себе створення прототипу (моделі) веб-застосунку, де були визначені ключові елементи, які повинні бути відображені на сторінках.

Навігаційна панель веб-застосунку містить логотип компанії, посилання на головну сторінку, сторінку товарів та спадне меню. Останнє включає посилання "Вхід у існуючий акаунт" і "Створення нового акаунту", якщо користувач не авторизований, або ім'я користувача та посилання на вихід, якщо авторизований.

Нижній колонтитул містить контактні дані компанії для зручності користувачів, які бажають зв'язатися з компанією.

Оскільки веб-застосунок призначений для інтернет-магазину і має привертати увагу клієнтів, головна сторінка містить коротку інформацію про компанію та причини обрати її сервіси та товари. Також на головній сторінці розташована кнопка-посилання для перегляду товарів компанії. Поточний прототип головної сторінки представлений на рисунку 3.1:

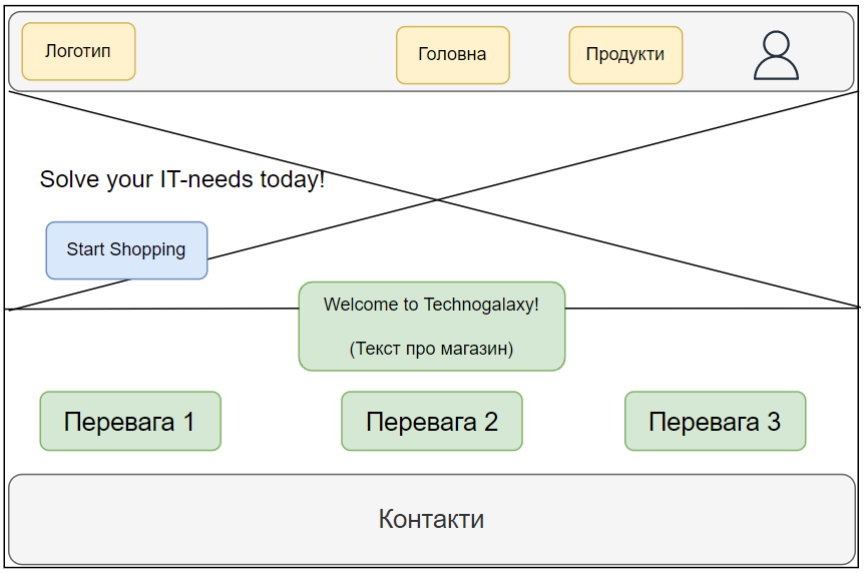

Рисунок 3.1 – Прототип головної сторінки

Сторінка товарів включає у себе інформацію про товари та посилання наних, як зображено на рисунку 3.2:

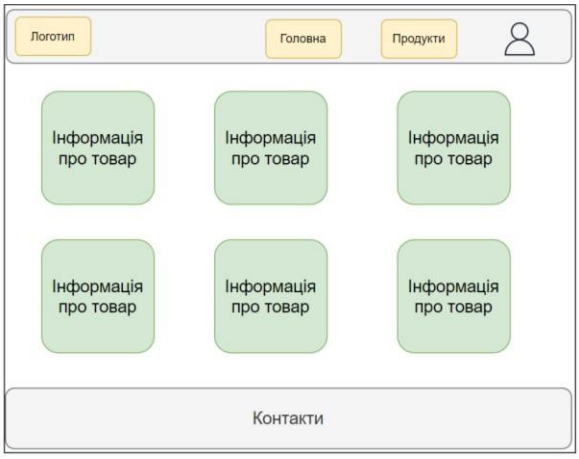

Рисунок 3.2 – Прототип сторінки товарів

Сторінка реєстрації користувача включає в себе форму для введення даних користувача, таких як ім’я користувача, електронна пошта та пароль. У складі цієї форми розміщені елементи для введення вищезазначених даних, а також кнопка для завершення реєстрації. Додатково, на сторінці присутнє посилання "У мене є акаунт", яке можна використовувати у випадку наявності облікового запису користувача. Прототип сторінки реєстрації користувача ілюструється на рисунку 3.3:

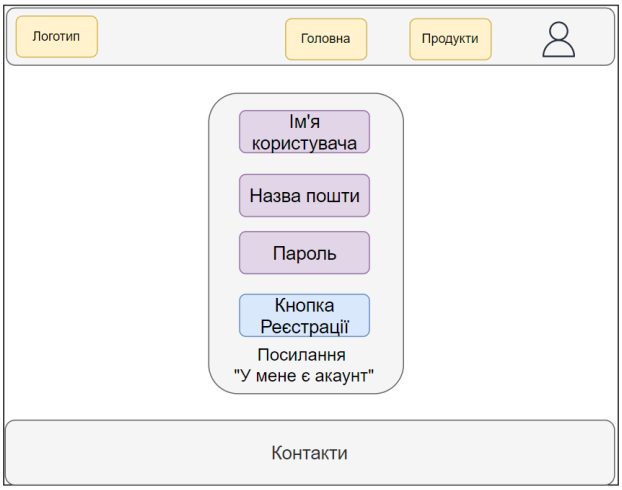

Рисунок 3.3 – Прототип сторінки реєстрації користувача

Сторінка входу користувача в існуючий акаунт містить форму для заповнення існуючих даних користувача (назва пошти, пароль), кнопку входу в акаунт та посилання на створення нового акаунту. Прототип сторінки входу користувача в існуючий акаунт зображено на рисунку 3.4:

Рисунок 3.4 – Прототип сторінки входу користувача в існуючий акаунт

Сторінка кошику містить відображення кількості різних видів товарів, які були додані до кошика. Кожен блок представляє тип товару і включає зображення, назву, кількість, ціну та кнопку для видалення товару з кошика. Крім того, сторінка кошика відображає загальну суму всіх доданих товарів та має кнопку "Оформлення замовлення". Прототип сторінки кошика наведено на рисунку 3.5.

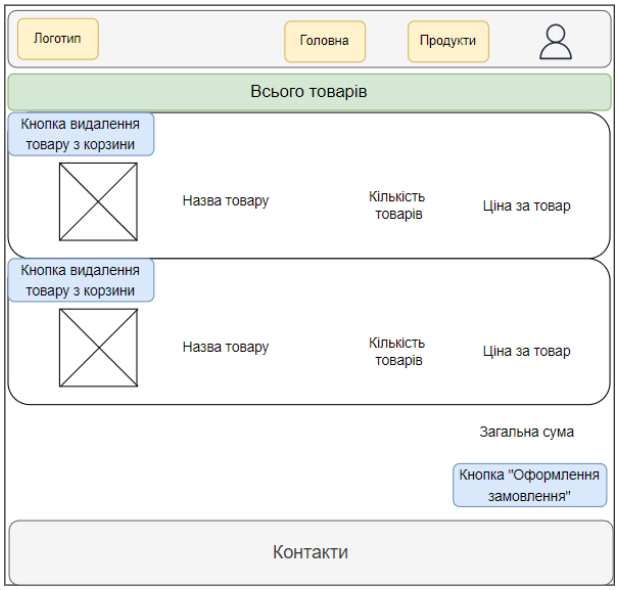

Рисунок 3.5 – Прототип сторінки кошика

Сторінка адреси доставки включає у себе форму для заповнення інформації щодо адреси користувача, як зображено на рисунку 3.6:

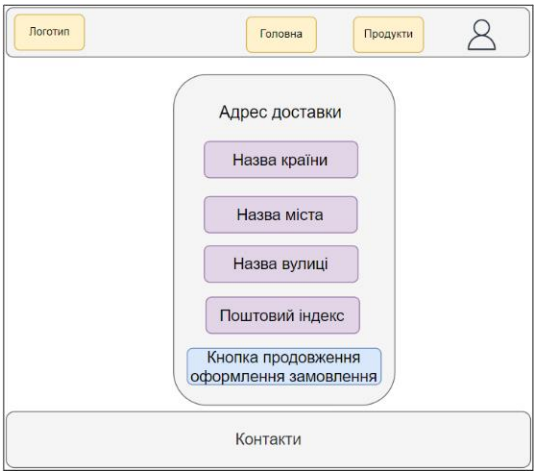

Рисунок 3.6 – Прототип сторінки інформації щодо адреси доставки

Сторінка процесу оформлення замовлення включає блок інформації про користувача (ім’я та електронна пошта користувача), дані про адресу доставки (країна, місто, вулиця), блоки інформації про додані товари у кошик, відображену суму для доставки, загальну суму замовлення та кнопку для оформлення замовлення. Прототип сторінки оформлення замовлення зображено на рисунку 3.7.

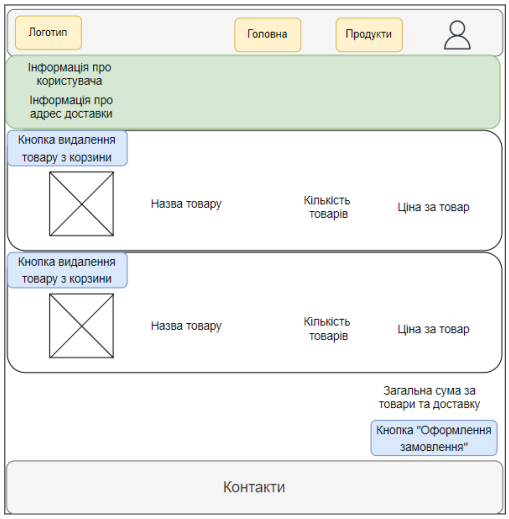

Рисунок 3.7 – Прототип сторінки оформлення замовлення

3.2 Створення прототипу веб-застосунку

Перед розпочатком роботи над версткою сторінок був використаний React-застосунок, створений за допомогою команди npx create-react-app. Під час етапу верстання розглядались основні елементи, які повинні завжди відображатися в веб-застосунку, незалежно від обраної користувачем сторінки. До цих елементів входили навігаційна панель та нижній колонтитул, оскільки користувач повинен мати постійний доступ до посилань на інші сторінки («Головна», «Сторінка товарів»), а також до спадного меню з відповідними посиланнями, в залежності від того, чи відбувся вхід користувача в особистий акаунт. Крім того, інформація про контакти завжди відображається в нижньому колонтитулі. Таким чином, ці елементи залишаються постійними в веб-застосунку, тоді як інші елементи змінюються в залежності від обраної користувачем сторінки.

Логотип компанії був створений за допомогою конструктору wix, доступного за посиланням https://wix.com/logo/maker, та зображений на рисунку 3.8:

Рисунок 3.8 – Логотип сайту

Для здійснення заміни елементів один на одного в залежності від адреси сторінки, ці елементи були розміщені в окремих компонентах. Компонента представляє собою функцію, яка повертає JSX-розмітку. Після створення компоненти можна використовувати її як тег.

Для реалізації можливості переходу за посиланнями на інші сторінки був створений тег Routes (об’єкт, який містить набір маршрутів), в якому вказані теги Route (компонента, яка відслідковує URL) разом із шляхами path (за яким шляхом має відображатися відповідна компонента) та атрибутами element (яка саме компонента має відображатися).

Також всю структуру компоненти App було вкладено у тег BrowserRouter, оскільки цей тег необхідний для правильної роботи маршрутизації.

Отже, структура маршрутизації тегу Routes разом із компонентами Route зображена на рисунку 3.9:

```
<Routes>
  <Route path="/diplom" element={<Main />} />
  <Route exact path="products" element={<Products />} />
  <Route exact path="/products/:id" element={<Product />} />
  <Route path="/cart/:id" element={<CartPage />} />
  <Route path="/cart" element={<CartPage />} />
  <Route path="/login" element={<Login />} />
  <Route path="/register" element={<Register />} />
  <Route exact path="/login/shipping" element={<ShippingPage />} />
  <Route exact path="/placeorder" element={<PlaceorderPage />} />
</Routes>
```

Рисунок 3.9 – структура маршрутизації веб-застосунку

Під час переходу на нову сторінку відбувається відображення верхньої частини сторінки через використання компоненти ScrollToTop. Це досягається обертанням компоненти Routes у вказану компоненту, яка містить метод window.scrollTo(). Деталі структури компоненти ScrollToTop наведені на рисунку 3.10:

```
const ScrollToTop = (props) => {
  const location = useLocation();
  useEffect(() => {
    window.scrollTo(0, 0);
  }, [location]);
  return <>{props.children}</>;
};
```

Рисунок 3.10 – Структура ScrollToTop

З метою полегшення читання коду компонента головної сторінки Main було розчленовано на дві компоненти: MainBlock та MainInfo. Компонента MainBlock відображає слоган компанії та кнопку-посилання на сторінку з товаром, із зображенням. Посилання та інформацію про автора було винесено до списку використаних джерел. Деталі структури MainBlock зображено на рисунку 3.11:

Рисунок 3.11 – Компонента MainBlock

Компонент MainInfo включає блок інформації про діяльність компанії і компоненту MainBenefits, що включає в себе компоненти MainBenefit. Компонента MainBenefits розкриває ключові аспекти сервісів і послуг, спрямовані на привертання уваги користувачів. Вона складається з трьох компонент MainBenefit, які кожен викликається з атрибутами image, title та text, як це показано на рисунку 3.12:

```
<MainBenefit
  img={<FontAwesomeIcon icon={faTruckLoading} size="2x" color="darkslateblue" />}
  title="Free Shipping"
  text="Free delivery over $100"
/>
```

Рисунок 3.12 – Структура одного з тегів компоненти MainBenefit

В якості зображення використовувався тег FontAwesomeIcon з атрибутами icon, size та color. Цей тег відноситься до бібліотеки Font Awesome.

З бібліотеки були вибрані три іконки, які відповідають змісту компоненти MainBenefit. Зокрема, це іконка faTruckLoading (зображення вантажівки й посилок), faTruckFast (зображення вантажівки) та faArrowRotateLeft (зображення стрілки).

Деталі структури компонента MainInfo представлені на рисунку 3.13:

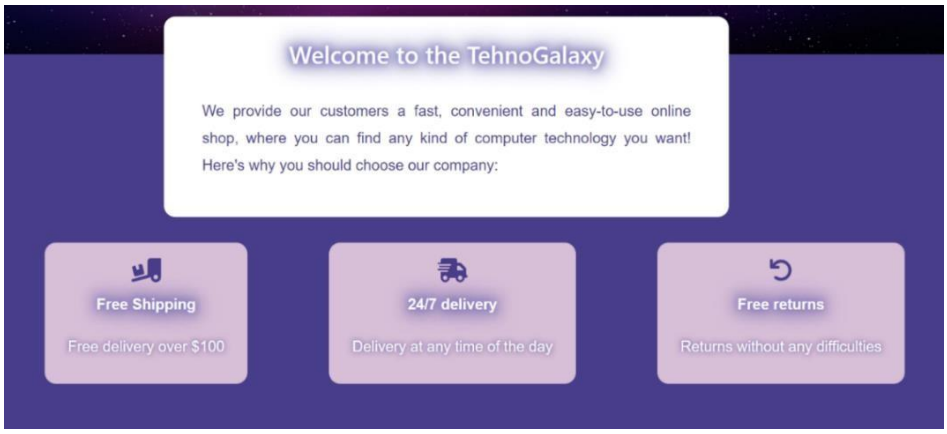

Рисунок 3.13 – Структура компонента MainInfo

Блок елементу нижнього колонтитулу був виділений в окрему компоненту під назвою Footer для полегшення роботи з кодом. У цій компоненті розміщена інформація про компанію (контактний телефон, контактну пошту), компоненти FontAwesomeIcon із визначеними атрибутами icon, size та color, а також інформація щодо умов авторських прав.

Атрибут icon компоненти FontAwesomeIcon включає іконки faPhone (зображення телефону) та faEnvelope (зображення поштової листівки), які відповідають конкретній інформації про компанію.

Деталі структури компоненти FontAwesomeIcon представлені на рисунку 3.14:

Рисунок 3.14 – Структура компонент FontAwesomeIcon

Для відображення процесу завантаження сторінки з товарами та сторінки з інформацією про конкретний товар була розроблена компонента Loading. Процес завантаження відображається за допомогою класу spinner-border з бібліотеки Bootstrap. Структура компоненти Loading представлена на рисунку 3.15:

Рисунок 3.15 – Структура компоненти Loading

Інші сторінки були ретельно розкладені на окремі компоненти. Посилання на зображення товарів та авторів цих зображень включені до переліку використаних джерел. Таким чином, після переходу на сторінку з товаром відображається компонента Products, на сторінці конкретного товару – компонента Product, на сторінці входу в акаунт – компонента Login, на сторінці реєстрації – компонента Register, на сторінці кошика – компонента CartPage, на сторінці введення адреси доставки – компонента ShippingPage, на сторінці оформлення замовлення – компонента

PlaceorderPage. Всі компоненти знаходяться у папці components. Структура цієї папки представлена на рисунку 3.16:

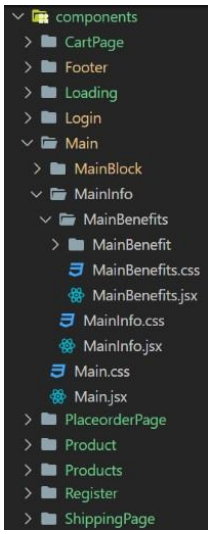

Рисунок 3.16 – Структура папки components

Таким чином, вся структура проєкту на кінець етапу верстання сторінок має вигляд, який зображено на рисунку 3.17:

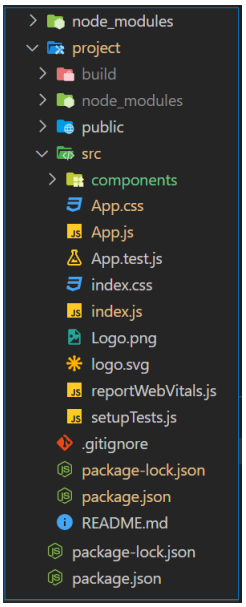

Рисунок 3.17 – структура проєкту після завершення етапу верстання сторінок

РОЗДІЛ 4. Розробка програмного забезпечення та приклад програми

4.1. Створення серверної частини та файлів із даними про товари й користувачів

У етапі розробки функціональної частини веб-застосунку спочатку була створена директорія server. Після цього в цій папці була викликана команда npm init для створення файлу package.json всередині папки server. Результат виконання цієї команди представлено на рисунку 4.1:

```
About to write to C:\Users\Xiaomi\Desktop\2_semestr\diplom\technogalaxy\server\package.json:

{
  "name": "server",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
```

Рисунок 4.1 – Результат команди npm init

Далі на черзі – встановлення пакетів-залежностей (dependencies) та пакетів-залежностей для розробки, необхідних для наступних етапів. Серед них входять bcryptjs, dotenv, express, express-async-handler, jsonwebtoken, mongoose, concurrently та nodemon.

Після цього була створена додаткова папка data всередині вже існуючої папки server, призначена для зберігання даних про товари. У цій папці був створений файл Products.js, де визначено змінну products. Ця змінна містить масив об'єктів, кожен з яких представляє інформацію про окремий товар. Ключі об'єкта товару включають name (назва товару), image (зображення товару), price (ціна товару), countInStock (наявна кількість товару) та rating (рейтинг товару).

Приклад одного з об'єктів, що містить інформацію про товар, подано на рисунку 4.2:

```
{
  name: "USB Mini connector Cable",
  image: "https://live.staticflickr.com/1345/897282770_ad55879502_b.jpg",
  description:
    "Lorem Ipsum is simply dummy text of printing and typesetting industry.",
  price: 5,
  countInStock: 15,
  rating: 4,
},
```

Рисунок 4.2 – Структура одного з об’єктів з інформацією про товар

Далі, наступним етапом, було створення файлу server.js всередині папки server. Після цього внесені зміни до файлу package.json:

– в якості точки входу в застосунок вказано server.js;

– в поле сценаріїв (scripts) додано сценарії "start" та "server", як це зображено на рисунку 4.3:

```
"scripts": {
  "start": "node ./server.js",
  "server": "nodemon ./server.js --ignore client"
},
```

Рисунок 4.3 – Поля сценаріїв (scripts)

– для того, щоб Node.js інтерпретував файли .js як ті, що використовують синтаксис модуля ES було задано “type”: “module”.

Для створення зв’язку між сервером та фронтендом у файлі було задано “proxy”: “http://localhost:5000”.

4.2. Підключення проекту до бази даних MongoDB та створення моделей Mongoose

Для підключення проєкту до бази даних MongoDB спочатку був створений новий файл .env всередині папки server, який містить змінні. Технологія dotenv була імпортована для зчитування змінних з файлу .env.

Після цього був створений новий проєкт у MongoDB, побудована нова база даних, як це показано на рисунку 4.4. Також був створений новий користувач, який отримав доступ до цієї бази даних. Для цього була визначена IP-адреса 0.0.0.0/0 для забезпечення необмеженого доступу.

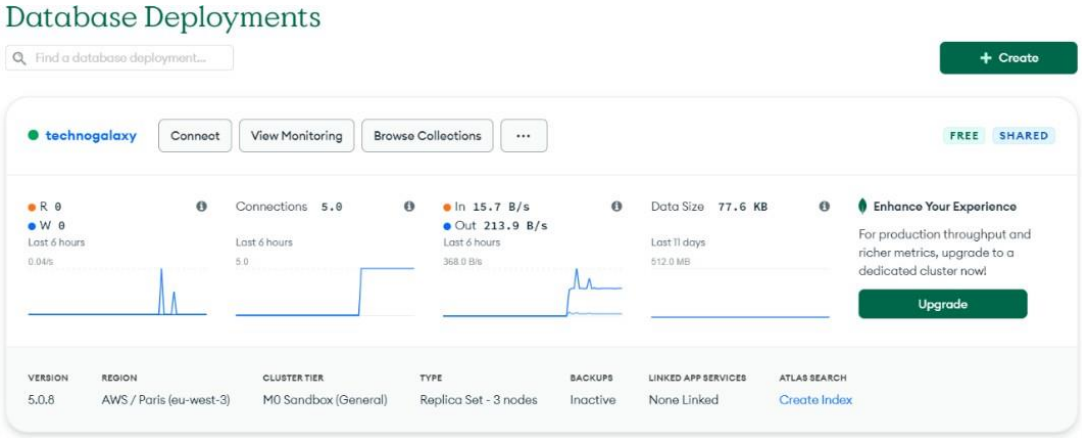

Рисунок 4.4 – база даних techno-shop

Для підключення веб-застосунку до бази даних був скопійований необхідний код для підключення з джерела MongoDb, і цей код був вставлений всередину нової змінної файлу .env під ім'ям MONGO_URL, як це зображено на рисунку 4.5:

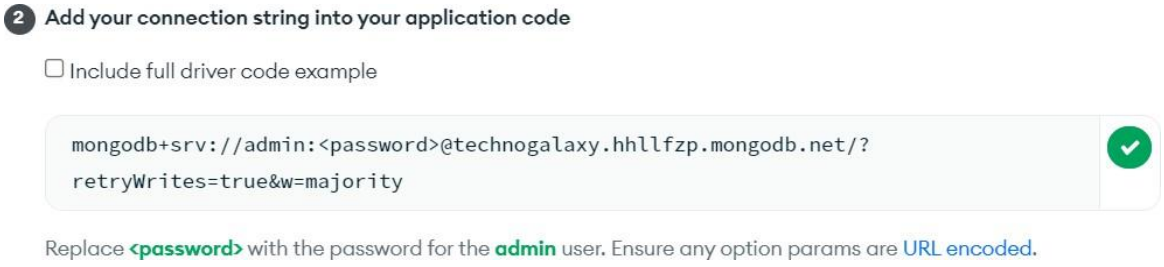

Рисунок 4.5 – код для підключення до бази даних

Поле <password> було замінено на фактичний пароль користувача бази даних, а між символами "/" та "?" була вказана назва бази даних.

Після цього була створена нова папка config всередині папки server із новим файлом MongoDb.js, який виконує функцію підключення веб-застосунку до бази даних. Структура файлу MongoDb.js представлена на рисунку 4.6:

```
import mongoose from "mongoose";
const connectDatabase = async () => {
  try {
    const connection = await mongoose.connect(process.env.MONGO_URL, {
      useUnifiedTopology: true,
      useNewUrlParser: true,
    });
    console.log("Mongo Connected");
  } catch (error) {
    console.log(`Error: ${error.message}`);
    process.exit(1);
  }
};
export default connectDatabase;
```

Рисунок 4.6 – Структура файлу MongoDb.js

Для підключення використовується метод mongoose .connect() з вказанням назви змінної для підключення та двох параметрів: useUnifiedTopology: true та useNewUrlParser: true.

Параметр useUnifiedTopology вилучає підтримку декількох параметрів підключення, які більше не є актуальними для нового механізму топології.

Параметр useNewUrlParser слугує прапорцем підключення до MongoDB. Щоб перевірити, що підключення працює, виводимо в консоль "Mongo Connected". В іншому випадку виконується блок catch, який виводить в консоль конкретну помилку.

Після цього було додано connectDatabase(); до файлу server.js.

Наступним етапом було створення моделей Schemas у mongoose. З цією метою була створена окрема папка для моделей з назвою Models всередині папки server, і туди були додані три нові файли: ProductModel.js, UserModel.js та OrderModel.js.

Схема товару (productSchema) містить властивості, такі як name, image, rating, price, countInStock. Кожна з цих властивостей складається з ключа типу даних, required (чи обов’язкове існування цієї властивості), при необхідності початкового значення (default) та опції timestamps.

Після додання опції timestamps: true, Mongoose автоматично додає дві властивості: createdAt та updatedAt. createdAt вказує на дату і час створення цього документу, а updatedAt – на дату і час останнього оновлення цього документу. Структура productSchema зображена на рисунку 4.7:

Рисунок 4.7– Структура productSchema

Схема користувача (userSchema) складається з визначених властивостей: name, email, password, isAdmin. Кожна з цих властивостей подібна до відповідної властивості схеми productSchema і містить ключі: type (тип даних), required (чи обов'язкове існування цієї властивості), а також, при необхідності, початкове значення (default). Опційно, до обох схем може бути додана опція timestamps. Структура userSchema проілюстрована на рисунку 4.8:

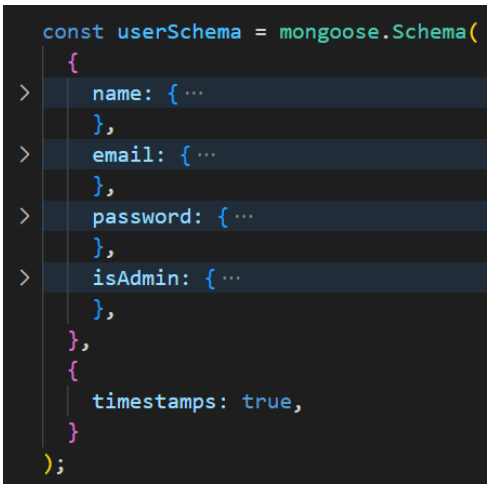

Рисунок 4.8 – Структура userSchema

Схема замовлення (orderSchema) включає в себе наступні властивості: user, shippingPrice, totalPrice, isPaid, paidAt, isDelivered, deliveredAt. Кожна з цих властивостей подібна до відповідної властивості схеми productSchema і включає ключі: type (тип даних), required (чи обов'язкове існування цієї властивості), а також, при необхідності, початкове значення (default). Додатково, до схеми можуть бути додані orderItems, shippingAddress, а також опція timestamps. Структура orderSchema зображена на рисунку 4.9:

```
const orderSchema = mongoose.Schema(
  {
    user: {···
    },
    orderItems: [···
    ],
    shippingAddress: {···
    },
    shippingPrice: {···
    },
    totalPrice: {···
    },
    isPaid: {···
    },
    paidAt: {···
    },
    isDelivered: {···
    },
    deliveredAt: {···
    },
  },
  {
    timestamps: true,
  }
);
```

Рисунок 4.9 – Структура orderSchema

Властивість orderItems буде містити інформацію про товари замовлення, тому ця властивість представлена як масив елементів. Вона включає ключі: name, qty (кількість конкретного товару у замовленні), image, price, а також властивість product. Остання включає ключ type із id конкретного товару, ключ required: true (означає, що ця інформація про товар обов'язкова) та ключ ref. Структура властивості orderItems проілюстрована на рисунку 4.10:

```
orderItems: [
  {
    name: { type: String, required: true },
    qty: { type: Number, required: true },
    image: { type: String, required: true },
    price: { type: Number, required: true },
    product: {
      type: mongoose.Schema.Types.ObjectId,
      required: true,
      ref: "Product",
    },
  },
],
```

Рисунок 4.10 – Структура властивості orderItems

Властивість shippingAddress буде містити інформацію про адресу доставки товару. Вона включає ключі: address, city, postalCode, та country, кожен з яких містить ключ type із значенням String та ключ required: true. Структура властивості shippingAddress проілюстрована на рисунку 4.11:

```
shippingAddress: {
  address: { type: String, required: true },
  city: { type: String, required: true },
  postalCode: { type: String, required: true },
  country: { type: String, required: true },
},
```

Рисунок 4.11 – Структура властивості shippingAddress

Наступним етапом було створення файлу user.js всередині папки data для даних користувачів. Цей файл експортує змінну user, яка буде складатися з масиву об’єктів. Кожен об'єкт міститиме інформацію про окремого користувача, включаючи ключі: name (ім’я користувача), email (пошта користувача), password (пароль користувача), та умову isAdmin. Пароль користувача буде зашифровано за допомогою методу hashSync бібліотеки bcrypt. Приклад одного з об’єктів із інформацією про користувача показано на рисунку 4.12:

```
{
  name: "Admin",
  email: "admin@technogalaxy.com",
  password: bcrypt.hashSync("123456", 10),
  isAdmin: true,
},
```

Рисунок 4.12 – Об’єкт з інформацією про користувача

4.3. Створення запитів та надіслання даних про товари та користувачів у базу даних MongoDb

Для вирішення даної задачі створено файл DataImport.js у папці server. У цьому файлі створено змінну ImportData, яка містить значення express.Router() (тобто через доступ до цієї змінної можна взаємодіяти з об’єктом Router() та його методами). Після виклику цієї змінної використано метод post для передачі інформації про товари та користувачів. Функція remove() видаляє дані з колекції та вставляє нові. Робота з даними користувачів зображена на рисунку 4.13:

```
const ImportData = express.Router();

ImportData.post(
  "/user",
  asyncHandler(async (req, res) => {
    await User.remove({});
    const importUser = await User.insertMany(users);
    res.send({ importUser });
  })
);
```

Рисунок 4.13 – Виконання методу post з даними користувачів

Після цього був викликаний метод use за конкретною адресою, який встановлює вказану функцію або функції проміжного програмного забезпечення за вказаним шляхом. У даному випадку функцією є змінна ImportData.

Після цього були надіслані дані у MongoDB за допомогою Postman. Для цього були обрані адреси: http://localhost:5000/api/import/products для передачі інформації про товари, http://localhost:5000/api/import/user для передачі інформації про користувачів, а тип запиту – POST, як видно на рисунку 4.14:

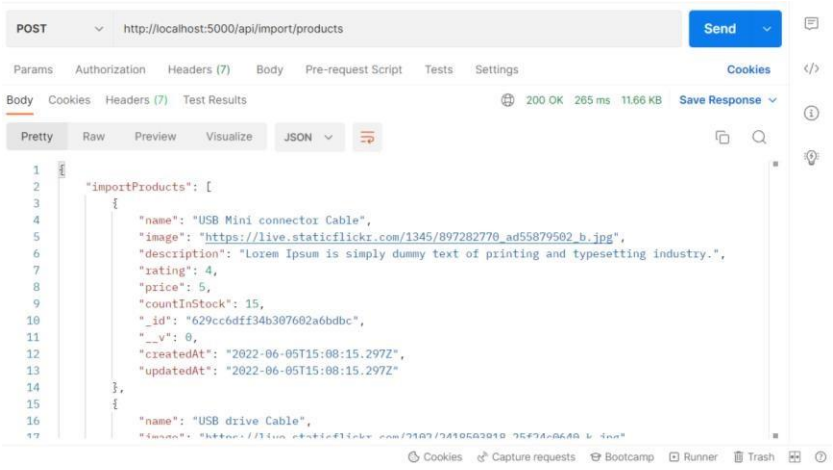

Рисунок 4.14 – Виконання запиту POST у Postman

Після натискання кнопки Send – база даних technogalaxy була оновлена, як зображено на рисунку 4.15:

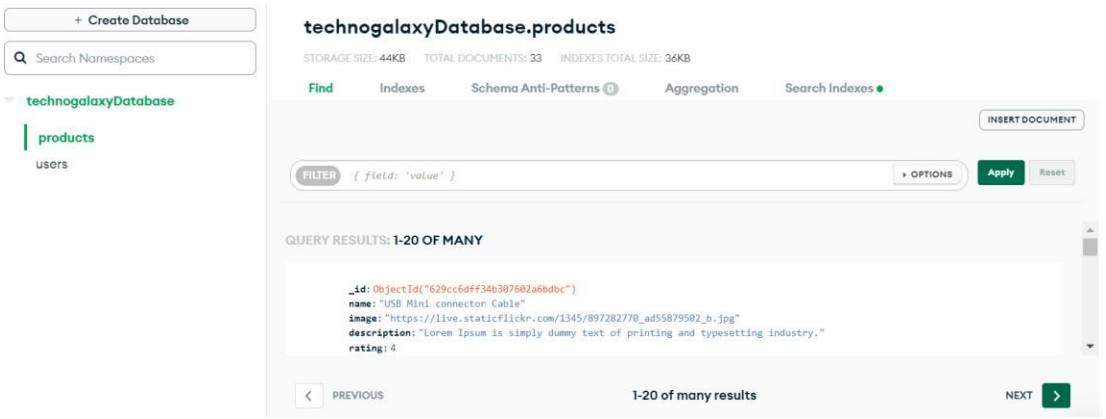

Рисунок 4.15 – Отримані дані про товари та користувачів у MongoDb

Для отримання даних про товари з MongoDB була створена папка Routes всередині папки server з файлом ProductRoute.js. У цьому файлі був використаний метод get для отримання інформації про товари та конкретний товар. Отримання інформації про всі товари зображено на рисунку 4.16:

```
const productRoute = express.Router();

//GET ALL PRODUCT
productRoute.get(
  "/",
  asyncHandler(async (req, res) => {
    const products = await Product.find({});
    res.json(products);
  })
);
```

Рисунок 4.16 – Метод get для отримання інформації про всі товари з базиданих

Після цього був викликаний метод use за конкретною адресою, де функцією є змінна productRoute. Таким чином, доступ до цих даних можна отримати за конкретним посиланням за допомогою Postman.

4.4. Створення проміжного обробнику (middleware) для відображення помилок

Для цього було створено папку Middleware всередині папки server, яка містить файл Errors.js. Структуру цього файлу зображено на рисунку 4.17:

Рисунок 4.17 – Структура файлу Errors.js

Отже, для відображення повідомлення про помилку через метод express.use() використовувалися змінні notFound та errorHandler у файлі server.js при кожному запуску веб-застосунку. Для взаємодії з товарами у веб-застосунку було завантажено розширення Redux DevTools, яке автоматично відображатиме стани товарів.

4.5. Робота з бібліотекою Redux

4.5.1 Створення папки для роботи з Redux

Наступним кроком було створено папку Redux з файлом store.js, який буде об’єднувати всі частини, необхідні для роботи над станами елементів. Початкова структура store складається з initialState (початкового стану веб-застосунку), методу combineReducers, який буде створювати окремі редуктори для стану застосунку, createStore, який буде створювати сховище

для збереження повного дерева стану веб-застосунку, функції thunk, яка виконує асинхронну роботу й всередині себе диспатчить інші звичайні дії (actions), функції applyMiddleware, яка буде передавати thunk у createStore (тобто приймає в якості аргументів middleware та застосовує їх), composeWithDevTools, який автоматично додає devtools до всього, що було передано в нього.

Для коректної роботи store компоненту App всередині файлу index.js було обернено в тег Provider (метод react-redux) з атрибутом store, який дорівнює store.

4.5.2. Створення папки для роботи з Redux

Далі було створено папку Redux з файлом store.js, який об'єднує всі необхідні частини для роботи зі станами елементів. Початкова структура store включає initialState (початковий стан веб-застосунку), метод combineReducers, який формує окремі редуктори для стану застосунку, createStore, який створює сховище для збереження повного дерева стану веб-застосунку, функцію thunk, що виконує асинхронні операції та диспетчеризує інші звичайні дії (actions), функцію applyMiddleware, яка передає thunk у createStore (тобто приймає middleware в якості аргументів та застосовує їх), composeWithDevTools, який автоматично додає devtools до всього, що було передано в нього. Для належної роботи компоненту App в файлі index.js його обернуто тегом Provider (з методу react-redux) з атрибутом store, що рівний store.

4.5.3. Створення дій (actions) та редукторів для списку товарів

Дія (Action) – це спосіб оновлення стану Redux-застосунку.
Редуктор (Reducer) обробляє ці дії стосовно стану веб-застосунку.
Для роботи з типами дій було створено папку Constants всередині папки Redux з файлом ProductConstants.js, де зберігаються змінні з типами дій.
До них відносяться PRODUCT_LIST_REQUEST (запит на список товарів), PRODUCT_LIST_SUCCESS (успішне отримання списку товарів) та PRODUCT_LIST_FAIL (неуспішне отримання списку товарів). Ці змінні представлені на рисунку 4.18:

```
export const PRODUCT_LIST_REQUEST = "PRODUCT_LIST_REQUEST";
export const PRODUCT_LIST_SUCCESS = "PRODUCT_LIST_SUCCESS";
export const PRODUCT_LIST_FAIL = "PRODUCT_LIST_FAIL";
```

Рисунок 4.18 – Змінні з типами дій (actions) над списком товарів

Для дій (actions) було створено окрему папку Actions з файлом ProductActions.js, в якій функція dispatch оголошує дії. Спочатку відбувається дія запиту на список товарів (PRODUCT_LIST_REQUEST), і якщо ця дія успішно виконана, виконується дія PRODUCT_LIST_SUCCESS, де всі дані про продукти передаються в спеціальну властивість payload. У випадку невдалої дії виконується дія PRODUCT_LIST_FAIL, яка передає помилку в payload. Структура дій з товарами зображена на рисунку 4.19:

```
export const listProduct = () => async (dispatch) => {
  try {
    dispatch({ type: PRODUCT_LIST_REQUEST });
    const { data } = await axios.get("/api/products");
    dispatch({ type: PRODUCT_LIST_SUCCESS, payload: data });
  } catch (error) {
    dispatch({
      type: PRODUCT_LIST_FAIL,
      payload:
        error.response && error.response.data.message
          ? error.response.data.message
          : error.message,
    });
  }
};
```

Рисунок 4.19 – Структура дій над товарами

З метою позначення редукторів, було вирішено розмістити їх у власній теки, і саме така тека "Reducers" була створена всередині основної теки "Redux" разом із файлом "ProductReducers.js". У цьому файлі було проведено редагування типів дій, пов'язаних із товарами, у розділі case оператора. При виклику редуктору на вході отримується стан "state", який представляє собою масив товарів "products", та стандартний параметр дії Redux, який дозволяє обробляти різноманітні дії, визначаючи їх тип у відповідному розділі case оператора.

У випадку запиту щодо товарів, умова завантаження має значення "true", а значення "products" залишається порожнім масивом. При успішному отриманні даних про товари, умова завантаження стає "false", а в "products" зберігаються дані про товари. У випадку невдачі у виконанні запиту на товари, умова завантаження залишається "false", а умова "error" отримує значення помилки. За замовчуванням повертається поточний стан (state).

Після створення редуктора для товарів під назвою "productListReducer", його було додано у файл "store.js" всередину методу "combineReducers". Структура "productListReducer" зображена на рисунку 4.20:

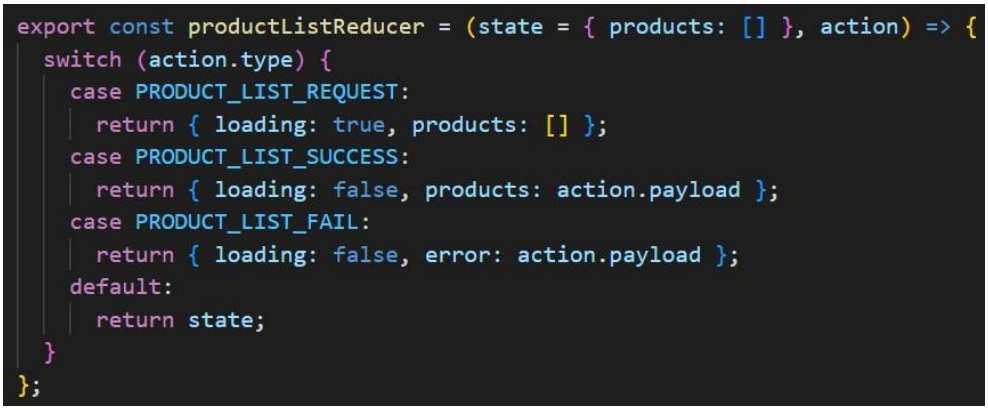

Рисунок 4.20– Структура редуктеру productListReducer

У файлі Products.jsx, що показує усі товари у веб-застосунку, було включено хук useSelector для отримання інформації про товари зі стану (state). Також додано хук useEffect, який виконує дії listProduct після завершення завантаження сторінки товарів. Змінні loading, error та products були передані у змінну productList для відображення процесу завантаження сторінки, можливих помилок та обробки списку товарів, як показано на рисунку 4.21.

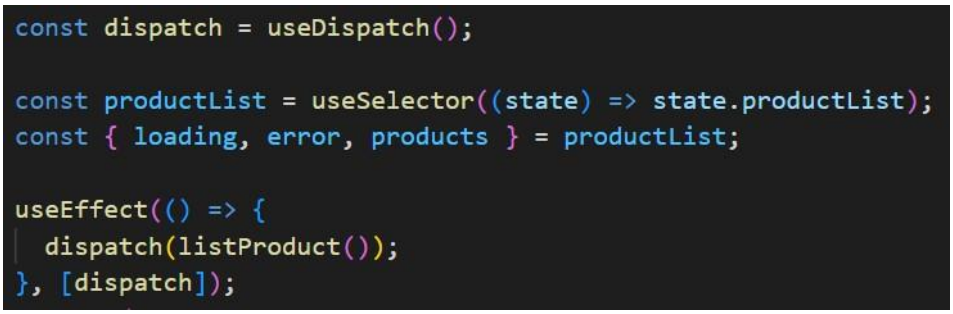

Рисунок 4.21 – Умови, необхідні для відображення списку товарів

У структуру HTML було додано умовний (тернарний) оператор, який розглядає три випадки: процес завантаження сторінки, наявність помилки та відображення списку товарів за допомогою методу map, як зображено на рисунку 4.22:

Рисунок 4.22 – Умовний (тернарний) оператор сторінки товарів

4.5.4. Створення дій (actions) та редукторів для одного товару

Спочатку були визначені відповідні змінні для відображення різних типів дій: PRODUCT_DETAILS_REQUEST вказує на запит даних про один товар, PRODUCT_DETAILS_SUCCESS – на успішне отримання даних про один товар, а PRODUCT_DETAILS_FAIL – на невдачу у відображенні даних про один товар.

Змінна listProductDetails містить набір дій, які будуть виконуватися з відношенням до конкретного товару.

При типі дії PRODUCT_DETAILS_REQUEST виконується запит get з використанням бібліотеки axios для отримання даних за конкретним ідентифікатором товару. Цей ідентифікатор автоматично генерується в базі даних MongoDB при передачі інформації про товари.

При типі дії PRODUCT_DETAILS_SUCCESS інформація про конкретний товар передається у поле payload. У випадку невдалого отримання даних (тип дії PRODUCT_DETAILS_FAIL) у полі payload передається інформація про виниклу помилку.

Структура дій щодо конкретного товару проілюстрована на рисунку 4.23:

```
export const listProductDetails = (id) => async (dispatch) => {
  try {
    dispatch({ type: PRODUCT_DETAILS_REQUEST });
    const { data } = await axios.get(`/api/products/${id}`);
    dispatch({ type: PRODUCT_DETAILS_SUCCESS, payload: data });
  } catch (error) {
    dispatch({
      type: PRODUCT_DETAILS_FAIL,
      payload:
        error.response && error.response.data.message
          ? error.response.data.message
          : error.message,
    });
  }
};
```

Рисунок 4.23 – Структура дій listProductDetails

Редуктор для конкретного товару був описаний у змінній productDetailsReducer. На вхід передається стан state, який буде дорівнювати об’єкту product та стандартний аргумент Redux action.

Структура редуктору, який буде виконувати дії над конкретним товаром має вигляд, аналогічний вигляду редуктору списку товарів, як зображено на рисунку 4.24:

```
export const productDetailsReducer = (state = { product: {} }, action) => {
  switch (action.type) {
    case PRODUCT_DETAILS_REQUEST:
      return { ...state, loading: true };
    case PRODUCT_DETAILS_SUCCESS:
      return { loading: false, product: action.payload };
    case PRODUCT_DETAILS_FAIL:
      return { loading: false, error: action.payload };
    default:
      return state;
  }
};
```

Рисунок 4.24 – Структура productDetailsReducer

Для використання цього редуктора в веб-застосунку його було додано до combineReducers у файлі store.js. Умови, необхідні для відображення конкретного товару, аналогічні умовам для списку товарів, за винятком того, що був доданий додатковий параметр id за допомогою хука useParams(). Після повного завантаження сторінки з конкретним товаром виконується хук useEffect для здійснення дій над конкретним товаром за відповідним id, як показано на рисунку 4.25:

```
const { id } = useParams();
const dispatch = useDispatch();
const productDetails = useSelector((state) => state.productDetails);
const { loading, error, product } = productDetails;

useEffect(() => {
  dispatch(listProductDetails(id));
}, [dispatch, id]);
```

Рисунок 4.25 – Умови, необхідні для відображення конкретного товару

Також у склад структури HTML було додано умовний (тернарний) оператор, який розглядає три випадки: процес завантаження сторінки, наявність помилки та відображення конкретного товару.

4.5.5. Створення функціоналу додавання товарів до кошика ті їх видалення з кошика

Кнопці “Add to cart”, яка відображена на сторінці одного товару була надана подія (event) onClick (подія, яка буде відбуватися при кліці на цю кнопку), яка виконує команди змінної AddToCartHandle. Ця змінна є функцією, яка виконує метод preventDefault() для скасування дій за замовчуванням та змінну navigate, яка містить хук useNavigate(хук для переадресації).

У адресі navigate використовуються змінні id товару та qty (кількість доданого товару у кошик). Qty – один з параметрів масиву хуку useState. Хук

useState – це функція, яка використовується для збереження стану у функціональній компоненті. Він приймає аргумент як початкове значення та повертає масив з двома елементами (перший елемент – теперішне значення стану, а друге – функція оновлення стану).

Таким чином, при кліці на кнопку “Add to cart” – буде відбуватися переадресація на сторінку кошика. Структура зміної AddToCartHandle зображена на рисунку 4.26:

```
const AddToCartHandle = (e) => {
  e.preventDefault();
  navigate(`/cart/${id}?qty=${qty}`);
};
```

Рисунок 4.26 – Структура змінної AddToCartHandle

Для відображення зміни обраної кількості товарів у тег select було передано два атрибути: value із значенням qty та onChange, яка містить функцію, що виконує функцію оновлення стану setQty значення target.value.

Назви типів дій були розміщені у файлі CartConstants.js. До цих дій відносяться CART_ADD_ITEM та CART_REMOVE_ITEM.

Щоб описати дії кошика, був створений окремий файл cartActions.js. Змінна addToCart містить дію додавання товару до кошика (CART_ADD_ITEM). На вхід addToCart подається змінна id товару та qty (кількість товару, яка буде додана до кошика). Вся інформація про конкретний товар отримується за допомогою методу axios.get за відповідним посиланням конкретного товару та додається до змінної data. Після цього виконується дія типу CART_ADD_ITEM, яка передає інформацію про товар у payload.

Після виконання дії CART_ADD_ITEM у веб-сховище localStorage зберігається значення товару за допомогою методу setItem(). Структура змінної addToCart зображена на рисунку 4.27:

Рисунок 4.27 – Дія додавання товару до кошика (CART_ADD_ITEM)

Змінна removeitem містить дію видалення товару з кошика (CART_REMOVE _ITEM). На вхід removeitem подається змінна id товару. Дія типу CART_REMOVE _ITEM передає інформацію про id товару у payload. Після цього у веб-сховищі відбувається оновлення інформації про стан товару кошика (він зникає з масиву об’єктів cartItems). Структура змінної removeitem зображена на рисунку 4.28:

Рисунок 4.28 – Дія видалення товару з кошика (CART_REMOVE_ITEM)

Для виконання дій кошика було створено редуктор cartReducer із секцією case оператора switch.

У випадку дії CART_ADD_ITEM відбувається перевірка на те, чи був вже конкретний товар доданий до кошика. У цьому випадку відбувається оновлення інформації про товару кошику, інакше – цей товар просто додається до кошика. Структура випадку (case) CART_ADD_ITEM зображена на рисунку 4.29:

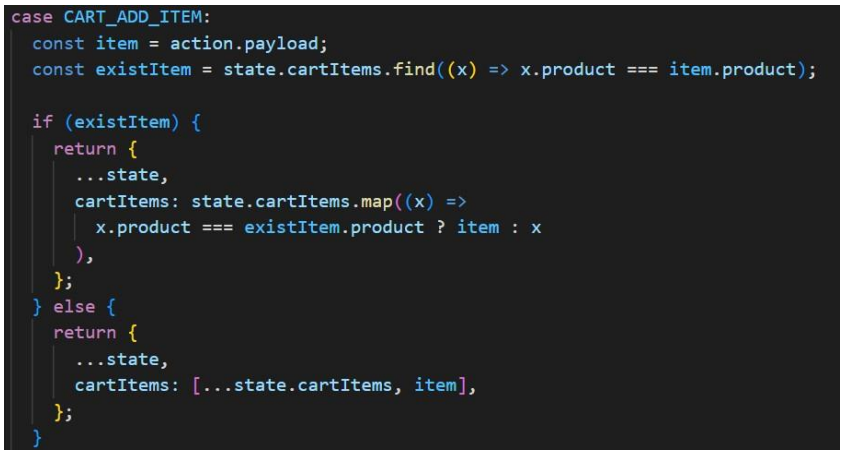

Рисунок 4.29 – Структура випадку (case) CART_ADD_ITEM

У випадку дії CART_REMOVE_ITEM відбувається видалення стану товару з кошика. Структура випадку (case) CART_REMOVE_ITEM зображена на рисунку 4.30:

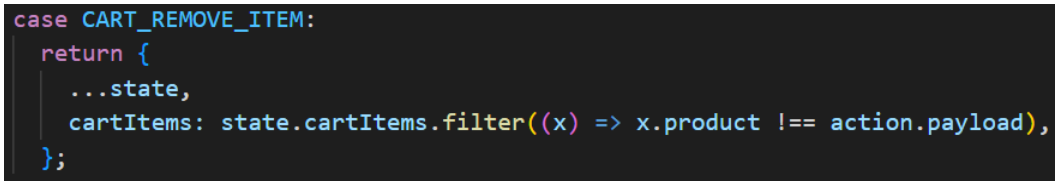

Рисунок 4.30 – Структура випадку (case) CART_REMOVE_ITEM

Для використання редуктору cartReducer у веб-застосунку його було додано до combineReducers у файлі server.js.

Також було створено змінну cartItemsFromLocalStorage для збереження даних про кошик товарів (cartItems) із веб-сховища localStorage. Якщо інформація існує – виконується команда JSON.parse, яка перетворює дані в об’єкт. Якщо інформації про товари у кошику немає – повертається порожній масив.

Структура змінної cartItemsFromLocalStorage зображена на рисунку 4.31:

```
const cartItemsFromLocalStorage = localStorage.getItem("cartItems")
  ? JSON.parse(localStorage.getItem("cartItems"))
  : [];
```

Рисунок 4.31 – Структура змінної cartItemsFromLocalStorage

Ця змінна була включена до складу початкового стану (initial state). У файлі CartPage.jsx, який відображає сторінку з корзиною товарів, було використано хук useSelector для отримання інформації про товари в корзині з поточного стану (state). Також був використаний хук useEffect, який, після завантаження сторінки з товарами, виконує дії addToCart з ідентифікатором товару та кількістю (qty) товару, якщо ідентифікатор товару присутній. Змінна qty була отримана шляхом пошуку числа в URL-адресі сторінки корзини.

Кнопка видалення товару з корзини включає обробник подій removeItemHandle, який отримує ідентифікатор товару. За цим ідентифікатором виконується дія removeitem, як показано на рисунку 4.32:

```
const removeItemHandle = (id) => {
  dispatch(removeitem(id));
};
```

Рисунок 4.32 – Структура події removeItemHandle

Для почергового відображення товарів з кошику було використано метод map. Розрахунки повної суми за всі товари у кошику були виконані у змінній totalSum з використанням методів reduce та toFixed, як зображено на рисунку 4.33:

```
const totalSum = cartItems
  .reduce((x, y) => x + y.qty * y.price, 0)
  .toFixed(2);
```

Рисунок 5.33 – Структура змінної totalSum

4.5.6. Створення функціоналу оформлення замовлень

У папці Routes був створений файл OrderRoutes.js для налаштування маршрутів. Змінній orderRouter було призначено значення express.Router(). Під час виклику post за адресою “/” відбувається перевірка доступності товарів під час складання замовлення. У випадку відсутності товарів повертається помилка, яка вказує на їх відсутність. У протилежному випадку створюється новий об'єкт замовлення та його збереження, як зазначено на рисунку 4.34:

```
} else {
  const order = new Order({
    orderItems,
    user: req.user._id,
    shippingAddress,
    itemsPrice,
    shippingPrice,
    totalPrice,
  });
  const createOrder = await order.save();
  res.status(201).json(createOrder);
}
```

Рисунок 4.34 – Створення нового об’єкту замовлення та його збереження

Для використання конкретних назв дій замовлення були створенні відповідні змінні у файлі orderConstants.js . Ці дії були описані у змінній createOrder файлу orderActions.js.

Спочатку виконується дія запиту на створення замовлення (ORDER_CREATE_REQUEST). Наступний тип дії – ORDER_CREATE_SUCCESS. Якщо цей тип дії виконується успішно – у payload передаються дані змінної data, яка виконує запит axios.post для передачі даних про замовлення.

Також виконується тип дії CART_CLEAR_ITEMS та відбувається видалення інформації про товари з кошику веб-сховища. В іншому випадку виконується тип дії ORDER_CREATE_FAIL, яка повертає повідомлення про помилку у payload.

Виконання дій створення замовлення було виконано у редукторі orderCreateReducer файлу OrderReducers.js у вигляді секції case оператора switch. Структуру редуктору для дій створення замовлення зображено на рисунку 4.35:

```
case ORDER_CREATE_REQUEST:
  return { loading: true };
case ORDER_CREATE_SUCCESS:
  return { loading: false, success: true, order: action.payload };
case ORDER_CREATE_FAIL:
  return { loading: false, error: action.payload };
case ORDER_CREATE_RESET:
  return {};
default:
  return state;
```

Рисунок 4.35 - Структура редуктору для дій створення замовлення

Назву файлу із редуктором для створення замовлення додано до об'єднання файлів у combineReducers у файлі store.js. У структурі файлу PlaceorderPage.jsx, що відображає сторінку оформлення замовлення, використано хук useSelector для отримання інформації про дані замовлення та хук useEffect. У випадку успішного оформлення замовлення виконується переадресація на сторінку товарів, а також виконується дія типу ORDER_CREATE_RESET, як показано на рисунку 4.36:

```
useEffect(() => {
  if (success) {
    navigate(`/products`);
    dispatch({ type: ORDER_CREATE_RESET });
  }
}, [navigate, dispatch, success, order]);
```

Рисунок 4.36 – Використання хуку useEffect у файлі PlaceorderPage.jsx

До складу кнопки оформлення замовлення було додано подію onClick із значенням функції PlaceOrderHandler, яка буде виконувати дію створення замовлення, як зображено на рисунку 4.37:

```
const placeOrderHandler = (e) => {
  console.log(userInfo);
  dispatch(
    createOrder({
      orderItems: cart.cartItems,
      shippingAddress: cart.shippingAddress,
      itemsPrice: cart.itemsPrice,
      shippingPrice: cart.shippingPrice,
      totalPrice: cart.totalPrice,
    })
  );
};
```

Рисунок 4.37– Структура змінної PlaceOrderHandler

Після оформлення замовлення інформація про це замовлення потрапляє до бази даних technogalaxy MongoDB.

4.5.7. Створення функціоналу закріплення товарів

Останнім етапом було створення унікальної можливості для веб-застосунку: можливість закріпити кілька товарів на екрані для зручного порівняння. Для цього було використано файл PinConstants.js для зберігання типів дій, таких як PIN_ITEM та UNPIN_ITEM. Ці типи дій було визначено у файлі PinAction.js. Дію PIN_ITEM було описано в змінній PinItem, де спочатку отримуються дані про товар за допомогою методу axios.get, вказуючи посилання на товар з відповідним ідентифікатором. При виконанні дії типу PIN_ITEM у полі payload передаються дані про товар. Значення закріпленого товару зберігається в localStorage за допомогою методу setItem(). Структура змінної pinItem показана на рисунку 4.38:

```
export const pinItem = (id) => async (dispatch, getState) => {
  const { data } = await axios.get(`/api/products/${id}`);
  dispatch({
    type: PIN_ITEM,
    payload: {···
    },
  });
  localStorage.setItem("pinnedItems", JSON.stringify(getState().pinningItem.pinnedItems));
};
```

Рисунок 4.38 – Структура змінної pinItem

Дію UNPIN_ITEM було описано у змінній removePin, при виконанні якої у payload передається значення id товару. Далі у веб-сховищі localStorage оновлюється інформація за допомогою методу setItem(). Структура змінної removePin зображена на рисунку 4.38:

```
export const removePin = (id) => (dispatch, getState) => {
  dispatch({
    type: UNPIN_ITEM,
    payload: id,
  });
  localStorage.setItem("pinnedItems",JSON.stringify(getState().pinningItem.pinnedItems));
};
```

Рисунок 4.38 – Структура змінної removePin

Редуктор, пов'язаний із діями закріплення та відкріплення товарів, був описаний у файлі PinReducers.js і має назву pinReducer. Він включає секцію case в операторі switch. У випадку дії PIN_ITEM відбувається перевірка на те, чи був вже конкретний товар закріплений. Якщо товар вже є закріпленим, повторне закріплення не відбувається; в іншому випадку товар закріплюється. Цикл перевірки на закріплення товару в сценарії (case) PIN_ITEM зображений на рисунку 4.39:

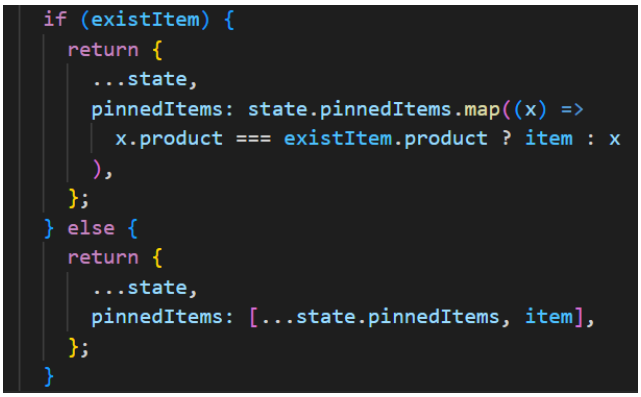

Рисунок 4.39 – Цикл випадку (case) PIN_ITEM

У випадку дії UNPIN_ITEM відбувається видалення товару із закріплених. Структура випадку (case) UNPIN_ITEM зображена на рисунку 4.40:

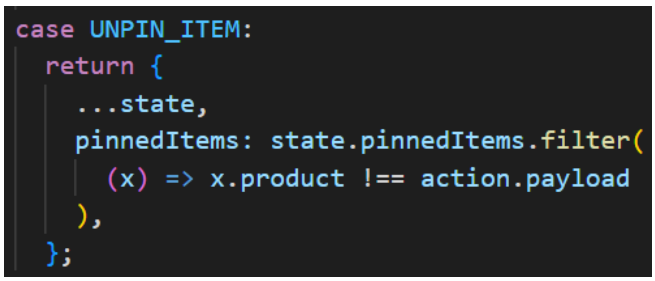

Рисунок 4.40 – Структура випадку (case) UNPIN_ITEM

Для інтеграції редуктора pinReducer у веб-застосунок його включили в файл combineReducers сервера (server.js). Також створено змінну pinnedItemsFromLocalStorage для зберігання інформації про закріплені

товари в локальному сховищі localStorage. Якщо дані існують, виконується команда JSON.parse для їх перетворення у об'єкт. У випадку відсутності інформації про закріплені товари повертається порожній масив. Структура змінної pinnedItemsFromLocalStorage представлена на рисунку 4.41:

```
const pinnedItemsFromLocalStorage = localStorage.getItem("pinnedItems")
  ? JSON.parse(localStorage.getItem("pinnedItems"))
  : [];
```

Рисунок 4.41 – Структура змінної pinnedItemsFromLocalStorage

Ця змінна була включена до складу початкового стану (initial state). У файлах Product.jsx та Products.jsx додано хук useSelector для отримання інформації про закріплені товари зі стану. Для закріплення товару створено кнопку з подією onClick, яка використовує змінну-функцію pinHandle. Ця функція виконує дію pinItem з відповідним id товару.

Для відображення закріплених товарів використано змінну pinnedItems, яка містить збережену інформацію про закріплені товари і використовує метод map. Додано кнопку для видалення товару зі списку закріплених, яка викликає функцію removePinHandle. Усередині цієї функції виконується дія removePin за відповідним id товару.

Висновок

В цьому дипломі розглянуто створення інтернет-магазину електроніки, що є важливим етапом у розвитку електронної комерції. Робота спрямована на дослідження та вирішення актуальних проблем в галузі онлайн-торгівлі електронічними товарами.

Аналізуючи динаміку сучасного ринку електроніки та враховуючи зростання популярності онлайн-покупок, було розроблено та впроваджено інтернет-магазин, спрямований на забезпечення зручності та доступності для клієнтів. Розглядаючи функціонал та інтерфейс магазину, можна визначити ефективні рішення, спрямовані на полегшення процесу вибору та покупки електронних товарів.

Особлива увага приділена аспектам безпеки онлайн-платежів та конфіденційності даних клієнтів. Застосування сучасних технологій шифрування та заходів безпеки сприяє впевненості покупців у безпеці їхніх транзакцій та особистої інформації.

Враховуючи широкий асортимент електронічних товарів у магазині, важливим елементом стало створення зручного інструменту для пошуку та фільтрації продукції. Це сприяє ефективному вибору товарів з урахуванням індивідуальних потреб кожного клієнта.

Висновки дозволяють визначити, що розроблений інтернет-магазин електроніки відповідає вимогам сучасного ринку та враховує потреби користувачів. Отримані результати можуть слугувати основою для подальших вдосконалень та розвитку електронної комерції.

Список використаних джерел

1. Інтернет-магазин ITBoxua. URL: https://www.itbox.ua/
2. Інтернет-магазин buttehnika. URL: https://byttehnika.zp.ua/
3. Інтернет-магазин XCOM URL: https://xcom.ua/
4. SQL vs. NoSQL Databases: What's the Difference? URL: https://www.upwork.com/resources/sql-vs-nosql-databases-whats-the-difference
5. SQLite. URL:https://lecturesdb.readthedocs.io/databases/sqlite.html
6. Sqlite vs. MySQL vs. PostgreSQL: A Comparison of Relational Databases. URL: https://logz.io/blog/relational-database-comparison/
7. Що таке HTML? URL:https://futurenow.com.ua/shho-take-html/
8. Основи JavaScript - Вступ в JavaScript. URL: https://www.coursehero.com/file/61100735/%D0%9E%D1%81%D0%BD%D0%BE% D0%B2%D0%B8-JavaScriptdocx/
9. Bootstrap · The most popular HTML, CSS, and JS library in the world. URL: https://getbootstrap.com/
10. The JavaScript Beginner's Handbook (2020 Edition). URL: https://www.freecodecamp.org/news/the-complete-javascript-handbook-f26b2c71719c/
11. Обзор ReactJS. URL: https://club.cnews.ru/blogs/entry/obzor_reactjs
12. React чи Angular або Vue.js. URL: https://habr.com/ru/post/476312
13. Mongoose – npm. URL: https://www.npmjs.com/package/mongoose
14. Bootstrap · The most popular HTML, CSS, and JS library in the world. URL: https://getbootstrap.com/
15. axios – npm. URL: https://www.npmjs.com/package/axios
16. Столбовский Д.Н. Основы разработки web-приложений на ASP.NET. Интернет-ун-т информ. технологий, 2009. 304 с.

17. Express – npm. URL: https://www.npmjs.com/package/express
18. Express-async-handler – npm. URL: https://www.npmjs.com/package/express-async-handler