МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА ІНТЕРНЕТ-МАГАЗИНУ З ВИКОРИСТАННЯМ HTML, CSS, JAVA SCRIPT, NODE.JS, MYSQL»**

Виконав: студент 4 курсу, групи 6.1260
спеціальності 126 Інформаційні системи та технології
(шифр і назва спеціальності)
освітньої програми Інформаційні системи та штучний інтелект
(назва освітньої програми)
І.В. Савченко
(ініціали та прізвище)
Керівник доцент кафедри програмної інженерії, PhD Чопорова О.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент доцент кафедри фундаментальної та прикладної математики, доцент, к.ф.-м.н. Панасенко Є.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 126 Інформаційні системи та технології
(шифр і назва)

Освітня програма Інформаційні системи та штучний інтелект

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

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Савченку Іллі Володимировичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка інтернет-магазину з використанням HTML, CSS, Java Script, Node.js, Mysql

керівник роботи Чопорова Оксана Володимирівна, PhD
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

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

2. Строк подання студентом роботи 03.06.2024 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.
3. Реалізація програмного коду.
4. Тестування програмної реалізації.

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

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

6. Консультанти розділів роботи

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

7. Дата видачі завдання 25.12.2023 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	09.01.2024	
2.	Збір вихідних даних.	15.01.2024	
3.	Обробка методичних та теоретичних	05.02.2024	
	джерел.		
4.	Розробка першого та другого розділу.	19.03.2024	
5.	Розробка третього розділу.	20.05.2024	
6.	Оформлення та нормоконтроль	27.05.2024	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	22.06.2024	

Студент _____ І.В. Савченко
(підпис) (ініціали та прізвище)

Керівник роботи _____ О.В. Чопорова
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

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

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка інтернет-магазину з використанням HTML, CSS, Java Script, Node.js, Mysql»: 72 с., 87 рис., 10 джерел, 1 додаток.

АДАПТИВНИЙ ДИЗАЙН, БЕКЕНД-РОЗРОБКА, ЖИВИЙ ПОШУК, КОШИК ПОКУПОК, ФІЛЬТРАЦІЯ, ФРОНТЕНД-РОЗРОБКА, API, CSS, HTML, JAVA SCRIPT, MYSQL, NODE.JS, UX/UI.

Об’єкт дослідження – процес розробки функціонального та зручного інтернет-магазину, що відповідає сучасним вимогам до електронної комерції.

Предмет дослідження – архітектура та ключові компоненти інтернет-магазину, відповідно до електронної комерції, що базується на платформах HTML, CSS, JavaScript, Node.js, MySQL для клієнтської та серверної частин.

Мета роботи: створення повноцінного інтернет-магазину з привабливим інтерфейсом та зручною навігацією, можливістю фільтрування та пошуку, додавання товару до кошику та оформлення замовлень.

Методи дослідження – аналіз сучасних вебтехнологій та принципів UX/UI дизайну, аналіз архітектур інтернет-магазину за моделями «клієнт-сервер», аналіз сучаних фронтенд-розробки та бекенд-розробки із використанням таких фреймворків: HTML, CSS, JavaScript, React.js та Node.js, Express.js. Аналіз сучасних СУБД для створення бази даних та використання MySQL в інтернет-магазинах. Аналіз видів тестування та оптимізація функціоналу інтернет-магазину.

При розробці додатку був проведений аналіз предметної області, обрано та спроєктовано архітектуру за допомогою аналізу сучасних методів створення інтернет-магазину.

В результаті роботи отримано інтернет-магазин із повним базовим функціоналом та відповідними критеріями.

SUMMARY

Bachelor’s qualifying paper “Development of an Online Store Using HTML, CSS, Java Script, Node.js, Mysql”: 72 pages, 87 figures, 10 references, 1 supplement.

ADAPTIVE DESIGN, API, BACKEND DEVELOPMENT, CSS, FILTERING, FRONTEND DEVELOPMENT, HTML, JAVASCRIPT, LIVE SEARCH, MYSQL, NODE.JS, SHOPPING CART, UX/UI.

The object of the study is the development of a functional and user-friendly online store that meets modern e-commerce requirements.

The aim of the study is creation of a fully functional online store with an attractive interface and easy navigation, allowing users to easily browse, search, filter products, and place orders.

The methods of research are analysis of modern web technologies and UX/UI design principles, analysis of online store architectures using “client-server” models, analysis of modern frontend and backend development using frameworks such as HTML, CSS, JavaScript, React.js, Node.js and Express.js. Analysis of modern DBMS for creating databases and the use of MySQL in online stores. Analysis of testing types and optimization of online store functionality.

As a result, a fully functional online store meeting the necessary criteria was achieved.

ЗМІСТ

ВСТУП

Онлайн-торгівля в наш час динамічно зростає, стаючи невід'ємною частиною сучасного життя. Розвиток інтернет-технологій та збільшення кількості користувачів Інтернету зробили покупки в онлайн-магазинах звичним явищем для багатьох.

Сучасні інтернет-магазини вирізняються значною функціональністю та зручністю використання. Вони пропонують широкий асортимент товарів, зручну систему пошуку та фільтрації, швидке оформлення замовлень та різноманітні методи оплати і доставки. З розвитком мобільних технологій з’явилися мобільні версії інтернет-магазинів, що дозволяють купувати товари з будь-якого місця і в будь-який час.

Ця дипломна робота присвячена розробці інтернет-магазину, що є актуальним і важливим завданням у світі сучасної електронної комерції. Мета роботи – створити функціональний та зручний інтернет-магазин з привабливим інтерфейсом і широкими можливостями для клієнтів, що зможе успішно конкурувати на сучасному ринку.

Інтернет-магазини стали важливою ланкою в сучасному бізнес-екосистемі, надаючи компаніям можливість досягати своїх цільових аудиторій за допомогою Інтернету. Компанії все частіше обирають електронну торгівлю для збільшення свого потенціалу з продажу товарів і послуг.

Ця дипломна робота прагне дослідити ці аспекти і надати практичні рекомендації для успішного створення та ефективного функціонування інтернет-магазину.

1 ОГЛЯД СУЧАСНОЇ ТЕХНОЛОГІЇ РОЗРОБКИ ВЕБЗАСТОСУНКУ

1.1 Фронтенд-розробка

Фронтенд-розробка є однією з ключових складових процесу створення вебзастосунків. Вона охоплює все, що користувач бачить і взаємодіє з на вебсторінці або в вебзастосунку. Основні технології, що використовуються для фронтенд-розробки, включають HTML, CSS і JavaScript. Кожна з цих технологій відіграє важливу роль у створенні сучасних вебзастосунків, забезпечуючи структуру, стиль і поведінку вебсторінок.

1.1.1 HTML

HTML (HyperText Markup Language) є основною мовою розмітки, яка використовується для створення вебсторінок та вебдодатків. Вона є фундаментальною технологією в структурі вебсторінки, що забезпечує базову розмітку та організацію контенту. HTML визначає структуру документа за допомогою різних елементів, таких як заголовки, абзаци, посилання, зображення, таблиці та форми.

HTML версії 5 (HTML5) принесла значні покращення в порівнянні з попередніми версіями. Вона ввела нові семантичні теги, які роблять структуру документа більш зрозумілою як для розробників, так і для пошукових систем [6]. Серед таких тегів:

– <header>: визначає заголовок документа або розділу [6];
– <footer>: містить кінцеву інформацію або навігаційні посилання [6];
– <article>: призначений для незалежного контенту, такого як стаття чи блог-пост [6];
– <section>: визначає розділ документа [6];

– <nav>: містить навігаційні посилання [6].

HTML5 також підтримує вбудовані мультимедійні елементи, такі як <audio> та <video>, що дозволяють додавати аудіо- та відео-контент без використання зовнішніх плагінів. Крім того, HTML5 включає нові формові елементи та атрибути, які полегшують створення інтерактивних форм та покращують користувацький досвід. Приклади таких елементів включають:

– <input> з типами email, date, url, що дозволяють легко вводити і перевіряти відповідні дані [6];
– атрибути placeholder, required, pattern, що покращують функціональність форм та валідацію даних на стороні клієнта [6].

Ці нові можливості HTML5 роблять його потужним інструментом для створення сучасних, адаптивних і доступних вебзастосунків, що відповідають вимогам сьогодення і забезпечують зручний та ефективний користувацький досвід.

1.1.2 CSS

CSS (Cascading Style Sheets) – це мова стилів, яка використовується для опису вигляду і форматування документів, написаних мовами розмітки, такими як HTML. CSS дозволяє відокремити структуру вебсторінки від її презентаційного шару, що сприяє більш зручному управлінню стилями та полегшує підтримку і оновлення вебзастосунків.

Основні можливості CSS включають:

– стилізація тексту: CSS дозволяє налаштовувати шрифти, їх розмір, колір, інтервал між символами, вирівнювання тексту та інші параметри [5];
– макетування: CSS допомагає розміщувати елементи на вебсторінці за допомогою властивостей позиціонування, таких як margin, padding, border, width, height і display [5];

– колір і фон: CSS надає можливість встановлювати кольори фону, тексту, градієнти, зображення фону та інші візуальні ефекти [5];
– адаптивний дизайн: CSS медіа-запити дозволяють створювати адаптивні вебсторінки, які підлаштовуються під різні розміри екранів та пристрої [5].

CSS є незамінним інструментом для створення сучасних, адаптивних і візуально привабливих вебзастосунків. З його допомогою можна створювати різноманітні дизайни, забезпечуючи водночас високий рівень зручності для користувачів.

1.1.3 Java Script

JavaScript є однією з найпопулярніших мов програмування, що використовується для реалізації динамічної поведінки вебсторінок. Вона додає інтерактивність до вебсайтів, роблячи їх більш живими і функціональними.

Основні можливості JavaScript включають:

– маніпуляція DOM: JavaScript може змінювати вміст сторінки в реальному часі, додаючи, видаляючи або змінюючи елементи HTML і CSS [7];
– обробка подій: такий спосіб дозволяє реагувати на дії користувача, такі як натискання кнопок, введення тексту або переміщення миші [7];
– взаємодія з сервером: JavaScript може взаємодіяти з сервером, використовуючи AJAX (асинхронні JavaScript та XML) запити, для отримання або відправлення даних без перезавантаження сторінки [7];
– фронтенд фреймворки: існують різноманітні фронтенд фреймворки, такі як React, Angular та Vue.js, які спрощують розробку вебдодатків, використовуючи JavaScript [7];

– взаємодія з API: JavaScript може взаємодіяти з різними API (Application Programming Interface), отримуючи доступ до даних і сервісів з інших додатків чи вебсайтів [7].

JavaScript є важливою складовою сучасних вебзастосунків, яка дозволяє створювати багатофункціональні та взаємодіючі додатки. Ця мова програмування поєднується з HTML та CSS для створення повноцінних вебзастосунків, які можуть відповідати потребам користувачів у реальному часі [2, 10].

1.2 Бекенд-розробка

Бекенд-розробка – це процес створення серверної частини вебзастосунків, яка відповідає за обробку запитів користувачів, взаємодію з базою даних та надання необхідної інформації фронтенду. Бекенд реалізується за допомогою різних технологій та мов програмування, таких як Node.js, Python, Ruby, PHP, Java та інші.

1.2.1 Node.js

Node.js – це середовище виконання JavaScript на сервері, яке дозволяє розробникам створювати швидкі та масштабовані серверні додатки. Однією з головних переваг Node.js є те, що цей фреймворк використовує неблокуючий, асинхронний підхід до програмування, що дозволяє обробляти багато запитів одночасно без блокування виконання програми [3].

Основні особливості Node.js включають:

– швидкодія: Node.js побудована на движок V8, який реалізується в Chrome, що забезпечує висока швидкість виконання JavaScript-

коду [9];

– асинхронність: завдяки використанню асинхронних операцій вводу-виводу (I/O), Node.js може ефективно обробляти багато запитів одночасно, що робить її ідеальним вибором для розробки високонавантажених додатків [9];
– масштабованість: Node.js підтримує горизонтальне та вертикальне масштабування, що дозволяє розробникам розширювати додатки при зростанні обсягу даних та навантаження [9].

Node.js є популярним вибором для створення бекенду вебзастосунків завдяки своїй ефективності, масштабованості та гнучкості. Це дозволяє розробникам створювати швидкі та ефективні додатки, які можуть витримати великі навантаження та забезпечувати зручний інтерфейс для користувачів.

1.2.2 MySQL

MySQL є однією з найпопулярніших відкритих реляційних баз даних, яка використовується для зберігання даних у вебдодатках. Вона надає потужні можливості для організації та маніпулювання даними, забезпечуючи швидкий доступ до інформації та надійність в роботі.

Основні характеристики MySQL включають:

– реляційна модель даних: MySQL базується на реляційній моделі даних, що дозволяє зберігати дані у вигляді таблиць зі зв’язками між ними, що спрощує організацію та обробку даних [8];
– масштабованість: MySQL підтримує горизонтальне та вертикальне масштабування, що дозволяє розширювати обсяг даних та обсяг робочих навантажень у вебдодатках [8];
– підтримка великих обсягів даних: MySQL може ефективно обробляти великі обсяги даних та великі навантаження, що робить

дану СУБД ідеальним вибором для високонавантажених вебдодатків [8].

MySQL є важливим компонентом бекенду вебзастосунків, який дозволяє зберігати, організовувати та маніпулювати даними у вебсерверних додатках. Ця СУБД є надійним та потужним інструментом для роботи з даними та забезпечує швидкий та ефективний доступ до інформації для користувачів вебзастосунків.

1.2.3 API

API (Application Programming Interface) є набором програмних інструкцій та структур даних, які використовуються для взаємодії між різними програмами. У веброзробці API використовується для спілкування між клієнтською та серверною частинами вебдодатків.

Основні характеристики API включають:

– структура даних: API визначає структуру та формат даних, що передаються між різними програмами [4];
– методи інтерфейсу: API визначає набір методів, які можуть бути викликані для взаємодії з програмою [4];
– протоколи комунікації: API вказує протоколи, які використовуються для обміну даними між програмами, такі як HTTP, WebSocket [4];
– документація: API може мати документацію, яка описує можливості, методи, параметри та приклади використання [4].

API використовується для взаємодії між різними компонентами вебдодатків, а також для спілкування між різними вебдодатками. API дозволяє розробникам створювати різноманітні програми та сервіси, які можуть взаємодіяти між собою та з іншими програмами, що дозволяє створювати складні та масштабовані вебзастосунки.

1.3 Висновки до розділу 1

Сучасна розробка вебзастосунків є складним процесом, що вимагає глибокого розуміння як фронтенд, так і бекенд технологій. Фронтенд розробка, відповідає за візуальний інтерфейс користувача, яка ґрунтується на HTML, CSS і JavaScript. Дані мови дають можливість створювати динамічні та адаптивні інтерфейси, що забезпечують комфортне користування. Бекенд розробка, яку не видно користувачеві, включає в себе фреймворки такі як Node.js, бази даних, забезпечуючи обробку даних та функціональність застосунку.

API, як інструмент взаємодії між фронтендом та бекендом, дає можливість створювати комплексні системи, що легко масштабуються та інтегруються з іншими сервісами.

Оволодіння різними технологіями фронтенду та бекенду, а також розуміння принципу роботи API, дозволяє розробнику створювати сучасні та ефективні вебзастосунки, що відповідають вимогам користувачів.

2 РОЗРОБКА ІНТЕРНЕТ-МАГАЗИНУ

2.1 Архітектура інтернет-магазину

Архітектура розробленого інтернет-магазину побудована за принципом «клієнт-сервер» з чітким розподілом відповідальності між фронтендом та бекендом. Фронтенд, реалізований з використанням HTML, CSS, JavaScript та бібліотеки React, відповідає за візуальне представлення інтернет-магазину та взаємодію з користувачем. Бекенд, розроблений на базі Node.js та фреймворку Express.js, обробляє запити від фронтенду, взаємодіє з базою даних MySQL та забезпечує бізнес-логіку магазину.

Взаємодія між фронтендом та бекендом здійснюється через API (Application Programming Interface), що дозволяє отримувати та оновлювати дані про товари, користувачів, замовлення. Така архітектура забезпечує гнучкість, масштабованість та високу продуктивність інтернет-магазину.

2.1.1 Модель «клієнт-сервер»

Розроблений інтернет-магазин базується на класичній моделі «клієнт-сервер», яка передбачає розподіл обчислювальних ресурсів та функціональності між двома основними компонентами: клієнтом (фронтендом) та сервером (бекендом).

Клієнт (фронтенд) в цій моделі представляє собою вебінтерфейс, з яким безпосередньо взаємодіє користувач. Ця частина відповідає за:

– відображення: показує користувачеві інтерфейс інтернет-магазину, включаючи каталог товарів, сторінки товарів, кошик, форми замовлення тощо;
– взаємодію: обробляє дії користувача, такі як натискання кнопок,

введення даних у форми, прокручування сторінок, вибір товарів;

– звернення до сервера: надсилає запити до сервера для отримання необхідних даних (наприклад, інформації про товари, статусу замовлення) та виконання певних дій (наприклад, додавання товару до кошика, оформлення замовлення).

Сервер (бекенд), у свою чергу, працює «за лаштунками», обробляючи запити від клієнта. Його основні функції:

– обробка запитів: приймає запити від клієнта, аналізує їх та виконує відповідні дії;
– взаємодія з базою даних: зберігає та обробляє дані інтернет-магазину, такі як інформація про товари, користувачів, замовлення;
– формування відповідей: готує та надсилає відповіді клієнту у форматі, зрозумілому для фронтенду.

Взаємодія між клієнтом та сервером відбувається через мережу, зазвичай за допомогою протоколу HTTP. Клієнт надсилає запит до сервера, а сервер повертає відповідь.

2.1.2 Розподіл відповідальності

У розробленій архітектурі інтернет-магазину відповідальність між фронтендом та бекендом розподілена з акцентом на візуальне представлення та інтерактивність на стороні клієнта та обробку даних та логіку фільтрації на стороні сервера.

Фронтенд (клієнтська частина) відповідає за:

а) візуальне представлення:

– відображення інтерфейсу інтернет-магазину, включаючи головну сторінку з слайдерами, каталог товарів, кошик;
– забезпечення зручної навігації між розділами сайту;
– стилізацію та візуальне оформлення елементів сайту;

б) взаємодію з користувачем:
- обробку подій користувача:
- натискання кнопок;
- введення даних у форму пошуку;
- прокручування слайдерів;
- вибір фільтрів за ціною та брендом;
- динамічне оновлення інтерфейсу при виборі фільтрів та введенні пошукового запиту;

в) взаємодію з бекендом:
- надсилання запитів до API бекенду для:
- отримання списку товарів;
- отримання інформації про конкретний товар;
- отримання списку доступних брендів;
- виконання пошуку за назвою товару;
- обробку відповідей API та відображення отриманих даних на
- сторінках сайту.

Бекенд (серверна частина) відповідає за:

а) обробку запитів API від фронтенду:
- прийом та аналіз запитів API;
- отримання даних з бази даних згідно з параметрами запиту (фільтрація, пошук);

б) взаємодію з базою даних MySQL:
- зберігання та оновлення даних про товари, включаючи їх назву, опис, ціну, зображення, бренд, рейтинг;

в) реалізацію бізнес-логіки:
- фільтрацію товарів за ціною та брендом;
- пошук товарів за назвою;
- формування відповідей API у форматі JSON.

2.2 Функціонал

Розроблений інтернет-магазин має ключовий набір функцій, що забезпечують зручний перегляд товарів, пошуку, фільтрацію та додавання товарів до кошику.

2.2.1 Кошик

Функціонал кошика:

- додавання товарів: користувач може додати товар до кошика на сторінці товару, натиснувши кнопку «Купити»;
- перегляд кошика: користувач може переглянути вміст кошика на окремій сторінці, де відображаються обрані товари, їх кількість та загальна сума замовлення;
- зміна кількості товарів: користувач може змінити кількість товарів в кошику за допомогою спеціальних кнопок «+» та «-»;
- оформлення замовлення: користувач може оформити замовлення, натиснувши кнопку «Оформити покупку».

2.2.2 Фільтрація за ціною та брендом

На сторінці каталогу товарів користувачам доступні фільтри за ціною та брендом, що дозволяють звузити список товарів відповідно до їх переваг.

Фільтрація за ціною: реалізовано за допомогою бібліотеки RC-Slider. Користувач може встановити мінімальну та максимальну ціну за допомогою повзунків.

Фільтрація за брендом: користувач може обрати один або декілька брендів зі списку, що відображається у вигляді прапорців.

2.2.3 Живий пошук

Функціонал живого пошуку дозволяє користувачам швидко знаходити потрібні товари за назвою. Під час введення пошукового запиту система динамічно відображає список товарів, що відповідають запиту. Це значно прискорює процес пошуку та зменшує кількість дій, які потрібно зробити користувачеві.

Живий пошук реалізовано за допомогою запитів до API бекенду. При введенні кожного символу в поле пошуку фронтенд надсилає запит до бекенду, який повертає список товарів, що містять введений текст в назві. Фронтенд динамічно оновлює список результатів пошуку, відображаючи їх користувачеві. Пошук не тільки виводить назву товарів які підходять під запит користувача, дана функція також виводить ціну та зображення товарів.

2.3 Висновки до розділу 2

Розроблений інтернет-магазин, побудований на архітектурі «клієнт-сервер», демонструє ефективне розподілення відповідальності між фронтендом та бекендом. Такий підхід забезпечує гнучкість, масштабованість та високу продуктивність системи. Фронтенл, реалізований з використанням React, забезпечує зручний та інтерактивний інтерфейс користувача. Бекенд побудований на Node.js та MySQL, обробляє дані та надає функціональність магазину.

Ключові функції магазину, такі як слайдери, кошик, фільтрація та живий пошук, спрощують користувацький досвід та підвищують ефективність роботи з сайтом. Реалізація функціоналу базується на сучасних технологіях, що забезпечує швидку та надійну роботу магазину.

Даний розділ демонструє успішне застосування технологій фронтенду та бекенду для створення повноцінного інтернет-магазину, здатного задовольнити потреби сучасного користувача.

3 РЕАЛІЗАЦІЯ ПРОЄКТУ ІНТЕРНЕТ-МАГАЗИНУ

3.1 Розробка та впровадження

В даному розділі буде розглянуто розробку проєкту, буде описано усі сторінки, приклад створення за допомогою обраних методів та розглянуто фрагменти програмного коду.

Головна сторінка є візитною карткою інтернет-магазину та виконує важливу роль у формуванні першого враження у користувача. На цій сторінці розміщено ключову інформацію про магазин, акційні пропозиції, новинки та інші важливі елементи, що заохочують користувачів до подальшого перегляду сайту та покупок.

Розглянемо реалізовані функції.

Слайдери:

– слайдер знижок: відображає товари, що пропонуються зі знижкою, привертаючи увагу користувачів до вигідних пропозицій;
– слайдер новинок: демонструє нові товари, які недавно були додані в асортимент товару.

Меню категорій:

– надає користувачам зручну навігацію по різних категоріях товарів в інтернет-магазині.

Живий пошук:

– дозволяє користувачам швидко знаходити потрібні товари за назвою, динамічно відображаючи результати пошуку під час введення запиту.

Наведемо опис реалізації сторінок з програмним кодом.

Компонент головної сторінки (Homepage.js).

Розглянемо усі реалізовані функції.

Стан та початкові значення:

– **activeSlideIndex**: використовується для зберігання індексу активного

слайду. Див. додаток А (рис. А.2);

– **searchTerm**: зберігає термін пошуку, введений користувачем. Див. додаток А (рис. А.2);
– **searchResults**: зберігає результати пошуку. Див. додаток А (рис. А.2).

Функція обробки зміни слайду (рис. 3.1).

```
const handleSlideChange = (index) => { // Функція обробки зміни слайду
  setActiveSlideIndex(index); // Встановлення активного індексу слайду
}; Див. додаток А (рис. А.2)
```

Рисунок 3.1

Дана функція приймає індекс нового слайду і оновлює стан activeSlideIndex.

Ефект для ініціалізації головного слайдера (рис. 3.2).

```
useEffect(() => { // Ефект для ініціалізації головного слайдера
  const slider = new ChiefSlider('.slider', { // Створення нового екземпляра ChiefSlider
    loop: true, // Зациклювання слайдів
    autoplay: true, // Автовідтворення слайдів
    interval: 5000, // Інтервал автовідтворення (5000 мілісекунд = 5 секунд)
  });

}, []); // Запуск ефекту лише після монтування компонента.
Див. додаток А (рис. А.2)
```

Рисунок 3.2

Описана функція useEffect виконується тільки один раз після монтування компонента. Ця функція ініціалізує слайдер за допомогою бібліотеки ChiefSlider з параметрами loop (зациклювання), autoplay (автовідтворення) і interval (інтервал 5000 мілісекунд).

Функція для отримання результатів пошуку (рис. 3.3).

```
useEffect(() => { // Ефект для отримання результатів пошуку
  const fetchResults = async () => { // Асинхронна функція для отримання результатів пошуку
    try {
      const response = await fetch(`http://localhost:3024/api_search/search?query=${encodeURIComponent(searchTerm)}`);      console.log('Response status:', response.status);
      if (!response.ok) {
        throw new Error('Network response was not ok');
      }
      const contentType = response.headers.get('content-type');
      console.log('Content-Type:', contentType);
      if (contentType && contentType.indexOf('application/json') !== -1) {
        const data = await response.json();
        console.log('Search results:', data);
        setSearchResults(data);
      } else {
        throw new Error('Response is not in JSON format');
      }
    } catch (error) {
      console.error(Помилка при отриманні результатів пошуку:', error);
    }
  };
  if (searchTerm) {
    fetchResults();
  } else {
    setSearchResults([]);
  }
}, [searchTerm]); // Запуск ефекту при зміні терміну пошуку.
```

Див. додаток А (рис. А.2)

Рисунок 3.3

Асинхронна функція fetchResults виконує HTTP-запит до сервера для отримання результатів пошуку.

Перевірка відповіді контролює статус та тип контенту. Якщо відповідь успішна і тип контенту JSON, дані змінюються на інший формат і зберігаються в стані searchResults. В разі помилки – в консоль відображається помилка типу даних.

При завантаженні useEffect(), перевіряється термін пошуку (searchTerm), якщо цей стан не порожній – викликається fetchResults, інакше – searchResults очищуються. useEffect() виконується щоразу, коли змінюється searchTerm.

Компоненти слайдерів (CustomSlider_discount.js, CustomSlider_news.js).

Представлені слайдерів реалізують логіку та візуальне представлення, використовуючи бібліотеки ChiefSlider та React-Slick.

Стан та початкові значення (рис. 3.4).

```
function CustomSlider({ className }) {
const sliderRef = useRef(null); // Створюємо посилання на компонент слайдера. Див. додаток А (рис. А.8)
```

Рисунок 3.4

SliderRef: використовується для створення посилання на DOM-елемент слайдера. Це дозволяє отримати доступ до методів слайдера та маніпулювати безпосередньо.

Налаштування слайдера (див. рис. 3.5):

- **infinite**: слайдер працює в безкінечному режимі, тобто після останнього слайду з’являється перший;
- **speed**: швидкість зміни слайдів у мілісекунд;
- **autoplay**: слайдер автоматично змінює слайди;
- **slidesToShow**: кількість слайдів, що відображаються одночасно;
- **slidesToScroll**: кількість слайдів, які перегортаються за один раз;
- **arrows**: відключення стрілок для ручного перегортування слайдів;
- **responsive**: налаштування для різних розмірів екранів.

```
const settings = {
  // Налаштування слайдера
  infinite: true, // Безкінечний режим
  speed: 5000, // Швидкість зміни слайдів (мс)
  autoplay: true, // Автовідтворення слайдів
  slidesToShow: 5, // Кількість слайдів, що відображаються одночасно
  slidesToScroll: 1, // Кількість слайдів, які перегортаються за раз
  arrows: false, // Вимкнення стрілок перегортування
  responsive: [
    // Адаптивні налаштування для різних розмірів екрану
    {
      breakpoint: 1024, // Поріг ширини екрану
      settings: {
        slidesToShow: 1, // Кількість слайдів для екранів ширше 1024px
      },
    },
    {
      breakpoint: 768, // Поріг ширини екрану
      settings: {
        slidesToShow: 2, // Кількість слайдів для екранів ширше 768px
      },
    },
    {
      breakpoint: 480, // Поріг ширини екрану
      settings: {
        slidesToShow: 3, // Кількість слайдів для екранів ширше 480px
      },
    },
  ],
}; Див. додаток А (рис. А.8)
```

Рисунок 3.5

Обробка прокрутки миші (рис. 3.6).

```
// Функція для обробки прокрутки слайдера за допомогою колеса миші
const handleSliderScroll = (e) => {
  if (sliderRef.current) { // Якщо компонент слайдера існує
    sliderRef.current.slickGoTo(sliderRef.current.innerSlider.state.currentSlide + e.deltaY / 100); // Прокрутити на відстань, залежну від переміщення колеса миші
  }
}; Див. додаток А (рис. А.8)
```

Рисунок 3.6

Представлений фрагмент коду функції обробляє прокрутку слайдера за допомогою колеса миші.

Перевірка наявності слайдера на існування (sliderRef.current).

Прокрутка слайдера, якщо слайдер існує, викликається метод slickGoTo, який змінює поточний слайд на новий. Новий слайд обчислюється на основі поточного слайду та величини прокрутки миші (e.deltaY / 100).

Компонент живого пошуку (api_search.js).

Даний компонент обробляє введення користувача в поле пошуку, надсилає запити до бекенду та динамічно відображає результати пошуку.

Маршрут обробки GET-запиту для пошуку в api_search.js (див. рис. 3.7).

Даний фрагмент коду демонструє такі функції обробки GET-запиту:

– отримання та декодування пошукового терміну: отримання пошукового терміну з параметрів запиту (req.query.query) та декодування пошукового терміну (decodeURIComponent(rawSearchTerm));
– перевірка на порожній термін: якщо пошуковий термін порожній, повертається порожній масив із статусом 200;
– запити до бази даних: створюється масив SQL-запитів для різних таблиць бази даних; для кожного запиту створюється «Promise», яке виконує SQL-запит до бази даних і повертає результати або помилку;
– виконання всіх запитів та об’єднання результатів: використовується «Promise.all» для очікування виконання всіх запитів; об’єднується результати всіх запитів у один масив; відправляються об’єднані результати у форматі JSON;
– обробка помилок: якщо виникає помилка під час виконання запиту або обробки запиту, повертається статус 500 та повідомлення про помилку.

```
router.get('/search', (req, res) => {
  const rawSearchTerm = req.query.query;
  console.log('Raw search term:', rawSearchTerm);

  try {
    const searchTerm = decodeURIComponent(rawSearchTerm);
    console.log('Decoded search term:', searchTerm);

    if (!searchTerm) {
      console.log('Empty search term, returning empty array');
      return res.status(200).json([]); // Відправляємо пустий масив з кодом 200
    }

    const queries = [
      'SELECT id, name, price, "smartphone" as type FROM smartphone WHERE name LIKE
?',
      'SELECT id, name, price, "tv" as type FROM tv WHERE name LIKE ?',
      'SELECT id, name, price, "product" as type FROM products WHERE name LIKE ?'
    ];

    const promises = queries.map(query => {
      return new Promise((resolve, reject) => {
        db.query(query, [`%${searchTerm}%`], (err, results) => {
          if (err) {
            console.error(Помилка запиту до бази даних:', err);
            return reject(err);
          }
          console.log(`Query results for query (${query}):`, results);
          resolve(results);
        });
      });
    }); Див. додаток А (рис. А.12)

    Promise.all(promises)
      .then(results => {
        const mergedResults = [].concat(...results);
        res.setHeader('Content-Type', 'application/json');
        res.json(mergedResults); // Відправляємо результати у форматі JSON
      })
      .catch(err => {
        res.status(500).json({ error: Помилка сервера' });
      });
  } catch (error) {
    console.error('Error processing request:', error);
    res.status(500).json({ error: Помилка обробки запиту });
  }
}); Див. додаток А (рис. А.13)
```

Рисунок 3.7

3.1.1 Сторінки товарів

Компонент сторінки список товарів (Notebook.js, SmartPhone.js, Television.js).

Компоненти цих сторінок відповідають за товари які можна купити, далі буде переглянуто на прикладі коду зі сторінки Notebook.js.

Notebook.js – це компонент React, який відповідає за відображення детальної інформації про обраний товар (в даному випадку, ноутбук). Даний компонент отримує ID товару з URL, використовує API для завантаження даних про цей товар та відображає всю інформацію у зручному для користувача форматі.

Функціонал:

- отримання ID товару: компонент отримує ID товару з URL за допомогою useParams;
- завантаження даних про товар: використовує useEffect для отримання даних про товар з API за його ID;
- додавання товару в кошик: кнопка «Купити» запускає функцію handleAddToCart;
- додавання товару до списку товарів в кошик (selectedItems) та оновлює localStorage з списком товарів;
- перехід до кошика: кнопка кошика в верхній панелі перенаправляє користувача на сторінку кошика за допомогою useNavigate.

Програмний код Notebook.js.

Оголошення станів:

- **const navigate = useNavigate()**: ініціалізує хук для навігації;
- **useState**: використовується для створення станів компоненту;
- **products:** зберігає список продуктів;
- **minPrice та maxPrice:** зберігають мінімальну та максимальну ціну продуктів;
- **sliderValue:** зберігає поточне значення цінового слайдера;

- **isFiltered:** визначає, чи застосовуються фільтри;
- **brands:** зберігає список доступних брендів;
- **selectedBrands:** зберігає обрані бренди для фільтрації;
- **initialProducts:** зберігає початковий список продуктів;
- **filterApplied** та **priceFilterApplied:** відстежують стан застосування фільтрів;
- **selectedItems:** зберігає список продуктів, доданих до кошика;

Функції для обробки подій миші (рис. 3.8).

```
const handleMouseEnter = (productId) => {
  const logoTovar = document.getElementById(`logo_tovar_${productId}`);
  if (logoTovar) {
    logoTovar.classList.add('hidden');
  }
};

const handleMouseLeave = (productId) => {
  const logoTovar = document.getElementById(`logo_tovar_${productId}`);
  if (logoTovar) {
    logoTovar.classList.remove('hidden');
  }
}; Див. додаток А (рис. А.19)
```

Рисунок 3.8

HandleMouseEnter та handleMouseLeave обробники подій наведення та відведення миші. Дані функції ховають або показують зображення товару при наведенні та відведенні миші відповідно.

Ефект для завантаження продуктів та брендів (рис. 3.9).

```
useEffect(() => {
  fetchProducts();
  fetchBrands();
}, []); Див. додаток А (рис. А.20)
```

Рисунок 3.9

UseEffect виконує побічні ефекти після монтування компоненту. Даний ефект викликає функції fetchProducts та fetchBrands для завантаження продуктів і брендів при першому рендерінгу компоненту.

Ефект для фільтрації продуктів (рис. 3.10).

```
useEffect(() => {
   if (priceFilterApplied || isFiltered) {
     filterProducts();
     setFilterApplied(true);
     setPriceFilterApplied(false);
   }
  }, [isFiltered, sliderValue, priceFilterApplied, selectedBrands]);
```

Див. додаток А (рис. А.20)

Рисунок 3.10

UseEffect виконує фільтрацію продуктів при зміні станів isFiltered, sliderValue, priceFilterApplied або selectedBrands.

Функції для обробки фільтрів (див. рис. 3.11):

– handleSliderChange обробляє зміну значення слайдера;
– handleInputChange обробляє зміну значення в інпутах для фільтрації за ціною;
– handleFilterButtonClick застосовує фільтрацію за ціною при натисканні кнопки фільтрації;
– handleBrandChange обробляє зміну вибраного бренду для фільтрації.

Функції для завантаження продуктів та брендів з сервера (див. рис. 3.12):

– fetchProducts завантажує продукти з сервера, обробляє їх, додає зображення до кожного продукту і встановлює мінімальні та максимальні ціни;
– fetchBrands завантажує бренди з сервера.

```
const handleSliderChange = (value) => {
    setSliderValue(value);
    if (priceFilterApplied) {
      setPriceFilterApplied(false);
    } else {
      setIsFiltered(false);
    }
  };

  const handleInputChange = (e, index) => {
    const newValue = parseInt(e.target.value, 10) || 0;
    const newSliderValue = [...sliderValue];
    newSliderValue[index] = newValue;
    setSliderValue(newSliderValue);
    setIsFiltered(true);
  };

  const handleFilterButtonClick = () => {
    setIsFiltered(true);
    setPriceFilterApplied(true);
  }; Див. додаток А (рис. А.20)

  const handleBrandChange = (brand) => {
    if (selectedBrands.includes(brand)) {
      setSelectedBrands(selectedBrands.filter(item => item !== brand));
    } else {
      setSelectedBrands([...selectedBrands, brand]);
    }
    setIsFiltered(true);
  };
Див. додаток А (рис. А.21)
```

Рисунок 3.11

```
const fetchProducts = () => {
  fetch(`http://localhost:3024/api/products`)
    .then(response => {
      if (!response.ok) {
        throw new Error('Network response was not ok');
      }
      return response.json();
    })
    .then(data => {
      console.log(data);
      const prices = data.map(product => product.price);
      const newMinPrice = Math.min(...prices);
      const newMaxPrice = Math.max(...prices);

      setMinPrice(newMinPrice);
      setMaxPrice(newMaxPrice);
      setSliderValue([newMinPrice, newMaxPrice]);

      const productsWithImages = data.map(product => ({
        ...product,
        image: `catalog/tovar/notebook/${product.id}/logo_1.png`
      }));

      setProducts(productsWithImages);
      setInitialProducts(productsWithImages);
    })
    .catch(error => console.error('Error fetching products:', error));
};

const fetchBrands = () => {
  fetch(`http://localhost:3024/api/brands`)
    .then(response => {
      if (!response.ok) {
        throw new Error('Network response was not ok');
      }
      return response.json();
    })
    .then(data => {
      setBrands(data);
    })
    .catch(error => console.error('Error fetching brands:', error));
}; Див. додаток А (рис. А.22)
```

Рисунок 3.12

Функції для фільтрації продуктів (рис. 3.13):

– filterProductsByPrice фільтрує продукти за ціною;
– filterProductsByBrand фільтрує продукти за брендом;
– filterProducts основна функція для фільтрації продуктів, яка використовує інші функції фільтрації за ціною та брендом.

```
const filterProductsByPrice = (productsToFilter, min, max) => {
  return productsToFilter.filter(product =>
    product.price >= min && product.price <= max
  );
}; Див. додаток А (рис. А.24)

const filterProductsByBrand = (productsToFilter) => {
  if (selectedBrands.length === 0) {
    return productsToFilter;
  }
  return productsToFilter.filter(product =>
    selectedBrands.includes(product.brand)
  );
}; Див. додаток А (рис. А.24)

const filterProducts = () => {
  let filteredProducts = initialProducts;

  if (isFiltered) {
    filteredProducts = filterProductsByPrice(filteredProducts, sliderValue[0], sliderValue[1]);
    filteredProducts = filterProductsByBrand(filteredProducts);
  }

  setProducts(filteredProducts);
}; Див. додаток А (рис. А.25)
```

Рисунок 3.13

Функції для роботи з кошиком (рис. 3.14):

– handleAddToCart додає продукт до кошика та зберігає оновлений список у localStorage;
– handleCartClick навігація до сторінки кошика з передачею даних про обрані товари.

```
  const handleAddToCart = (product) => {
    const updatedItems = [...selectedItems, { ...product, image:
`catalog/tovar/notebook/${product.id}/logo_1.png` }];
    setSelectedItems(updatedItems);
    localStorage.setItem('cartItems', JSON.stringify(updatedItems));
  }; Див. додаток А (рис. А.26)

  const handleCartClick = () => {
    navigate('/cart', { state: { cartItems: selectedItems } });
  }; Див. додаток А (рис. А.27)
```

Рисунок 3.14

Представлений код сторінки ноутбуків, не є автономним, для вирішення цієї проблеми, є виклики до зовнішніх файлів, а саме:

– Rating.js: сторінка яка виводить рейтинг товарів зірочками;
– Cart.js: сторінка кошика в якій зберігаються товари, які були обрані на сторінці ноутбуків;
– Api.js: код Api, який з’єднується з БД і робить запити до сайту ноутбуків.

Далі розглянемо код цих трьох сторінок.

Програмний код для сторінки Rating.js.

Компонент Rating (рис. 3.15).

```
// Компонент рейтингу, що відображається за допомогою зірок
const Rating = ({ value }) => {
  const stars = []; Див. додаток А (рис. А.33)
```

Рисунок 3.15

Компонент Rating, функціональний компонент, який приймає пропс value. Пропс value визначає кількість зірок, які будуть відображені як заповнені. Const stars = [] – створюється порожній масив stars, який буде заповнений елементами зірок.

Цикл для створення зіркового рейтингу (рис. 3.16).

```
// Цикл для створення зіркового рейтингу
for (let i = 0; i < 5; i++) {
  if (i < value) {
    stars.push(<span key={i}>&#9733;</span>); // Зірка
  } else {
    stars.push(<span key={i}>&#9734;</span>); // Пуста зірка
  }
} Див. додаток А (рис. А.33)
```

Рисунок 3.16

Цикл for – виконується п’ять разів, по одному разу для кожної потенційної зірки рейтингу.

If (i < value) – якщо індекс «i» менше значення value, додається заповнена зірка (★).

Else – якщо індекс «i» більше або дорівнює значенню value, додається пуста зірка (☆).

Програмний код для сторінки Cart.js.

Оголошення станів (рис. 3.17): const [cartItems, setCartItems] = useState(JSON.parse(localStorage.getItem('cartItems')) || []) – ініціалізується стан cartItems, який зберігає товари у кошику. Спочатку функція намагається завантажити дані з локального сховища (localStorage). Якщо даних немає, використовується порожній масив.

```
function Cart() {
  const [cartItems, setCartItems] = useState(
    JSON.parse(localStorage.getItem('cartItems')) || []
  ); Див. додаток А (рис. А.34)
```

Рисунок 3.17

Ефект для збереження елементів у локальному сховищі (рис. 3.18): useEffect(() => { ... }, [cartItems]) – цей ефект виконується кожного разу, коли змінюється стан cartItems. Даний ефект зберігає оновлені дані корзини у локальному сховищі, перетворюючи їх у формат JSON.

```
useEffect(() => {
  localStorage.setItem('cartItems', JSON.stringify(cartItems));
}, [cartItems]); Див. додаток А (рис. А.35)
```

Рисунок 3.18

Функція для зміни кількості товару в корзині (рис. 3.19): const handleQuantityChange = (index, newQuantity) => { ... } – дана функція приймає індекс товару в масиві cartItems та нову кількість newQuantity. Розглянута функція оновлює кількість товару та оновлює стан cartItems.

```
const handleQuantityChange = (index, newQuantity) => {
  const updatedCartItems = [...cartItems];
  updatedCartItems[index].quantity = newQuantity;
  setCartItems(updatedCartItems);
}; Див. додаток А (рис. А.36)
```

Рисунок 3.19

На рисунку 3.20 наведено const handleRemoveItem = (index) => { ... } – ця функція видаляє товар з корзини за індексом. Функція створює новий масив updatedCartItems, який містить всі елементи, крім того, що потрібно видалити та оновлює стан cartItems.

```
const handleRemoveItem = (index) => {
  const updatedCartItems = cartItems.filter((_, i) => i !== index);
  setCartItems(updatedCartItems);
}; Див. додаток А (рис. А.37)
```

Рисунок 3.20 – Функція для видалення товару з корзини

Розрахунок загальної ціни товарів у корзині (рис. 3.21): const totalPrice = cartItems.reduce((total, item) => total + (parseFloat(item.price) * item.quantity || 0), 0) – даний вираз обчислює загальну вартість товарів у корзині. Функція reduce проходить по кожному товару, додаючи його вартість (ціна помножена на кількість) до загальної суми total.

```
const totalPrice = cartItems.reduce(
  (total, item) => total + (parseFloat(item.price) * item.quantity || 0),
  0
); Див. додаток А (рис. А.38)
```

Рисунок 3.21

Програмний код для сторінки Api.js (див. рис. 3.22): router.get('/products', (req, res) => { ... }) – обробляє GET-запити на маршрут /products:

– виконує SQL-запит для отримання всіх продуктів з таблиці Products;
– якщо виникає помилка, дана помилка виводиться у консоль, а клієнту повертається відповідь з помилкою (500);
– у разі успіху результати запиту повертаються у вигляді JSON.

```
router.get('/products', (req, res) => {
    db.query('SELECT * FROM Products', (err, results) => {
        if (err) {
            console.error('Помилка запиту до бази даних:', err);
            res.status(500).json({ error: 'Помилка сервера' });
            return;
        }
        res.json(results);
    });
}); Див. додаток А (рис. А.43)
```

Рисунок 3.22 – Маршрут для отримання всіх товарів

Маршрут для фільтрації продуктів за ціною (рис. 3.23): router.get('/products/filter', (req, res) => { ... }) – обробляє GET-запити на

маршрут /products/filter:

– отримує мінімальну та максимальну ціну з параметрів запиту;
– виконує SQL-запит для отримання продуктів у вказаному ціновому діапазоні;
– у разі помилки повертається статус 500, у разі успіху результати запиту повертаються у форматі JSON.

```
router.get('/products/filter', (req, res) => {
   const minPrice = parseFloat(req.query.minPrice);
   const maxPrice = parseFloat(req.query.maxPrice);
   db.query('SELECT * FROM Products WHERE price >= ? AND price <= ?', [minPrice,
maxPrice], (err, results) => {
      if (err) {
         console.error('Помилка запиту до бази даних:', err);
         res.status(500).json({ error: 'Помилка сервера' });
         return;
      }
      res.json(results);
   });
}); Див. додаток А (рис. А.44)
```

Рисунок 3.23

Маршрут для отримання унікальних брендів (рис. 3.24): router.get('/brands', (req, res) => { ... }) – обробляє GET-запити на маршрут /brands:

– виконує SQL-запит для отримання унікальних значень брендів з таблиці Products;
– у разі помилки повертається статус 500, у разі успіху результати запиту (список брендів) повертаються у форматі JSON.

```
router.get('/brands', (req, res) => {
   db.query('SELECT DISTINCT brand FROM Products', (err, results) => {
      if (err) {
         console.error('Помилка запиту до бази даних:', err);
         res.status(500).json({ error: 'Помилка сервера' });
         return;
      }
      const brands = results.map(result => result.brand);
      res.json(brands);
   });
}); Див. додаток А (рис. А.45)
```

Рисунок 3.24

3.2 Тестування проєкту інтернет-магазину

У цьому розділі детально розглянуто функціональність, реалізовану в розділі 3.1. Буде проведено тестування сторінок, оцінивши їх візуальний вигляд та працездатність.

1.2.1 Тестування головної сторінки

В першу чергу буде розглянуто візитну картку сайту: «Головна сторінка» та її аспекти (рис. 3.25 – 3.26).

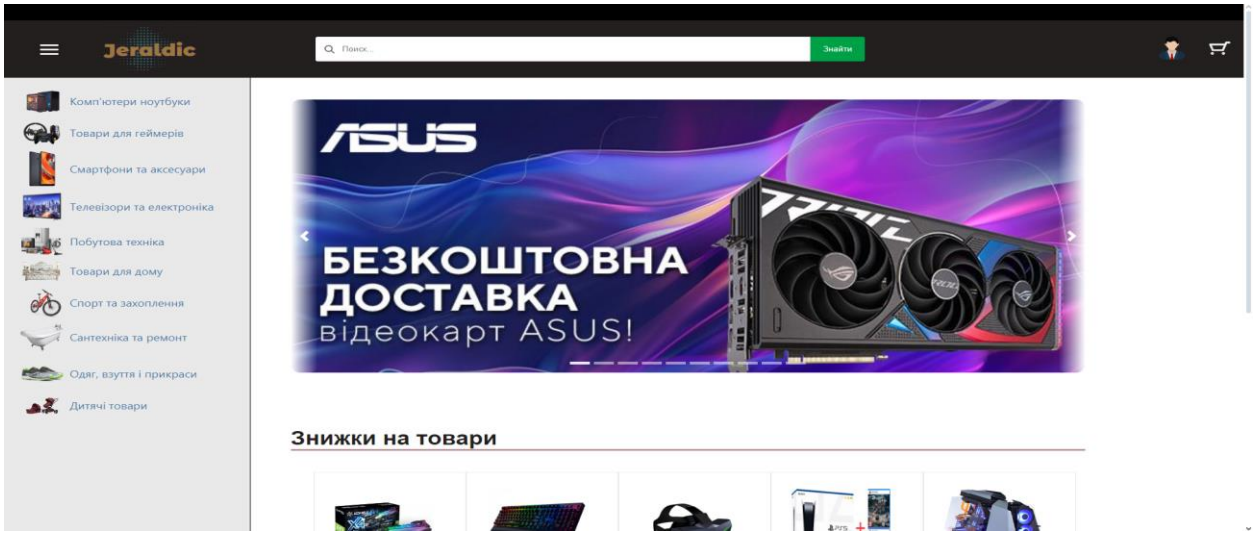

Рисунок 3.25 – Перша частина головної сторінки

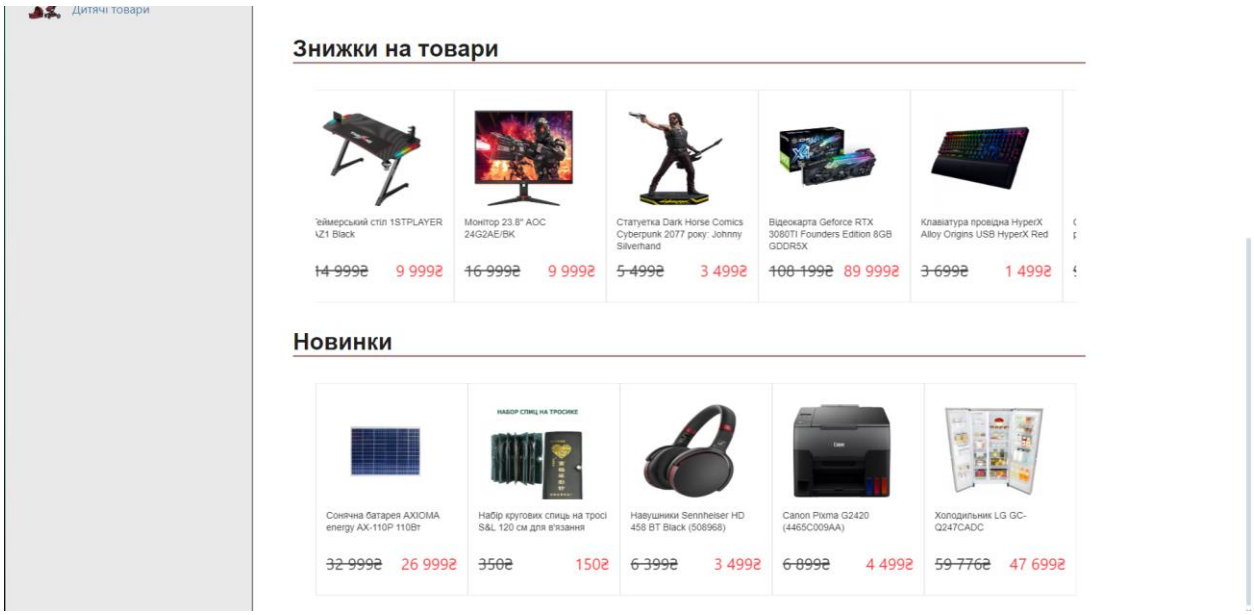

Рисунок 3.26 – Друга частина головної сторінки

Далі буде розглянуто рисунки 3.25 – 3.26 на окремі пункти.

Перша частина. Верхня частина сайту – це шапка, в якій розміщуються поле пошуку, іконка користувача, іконка кошика.

Далі було протестована поле пошуку. Введено в поле пошуку «Ноутбук».

На рисунку 3.27 можна побачити пошуковий результат, а саме: ноутбуки, їх повну назву, ціну, логотип.

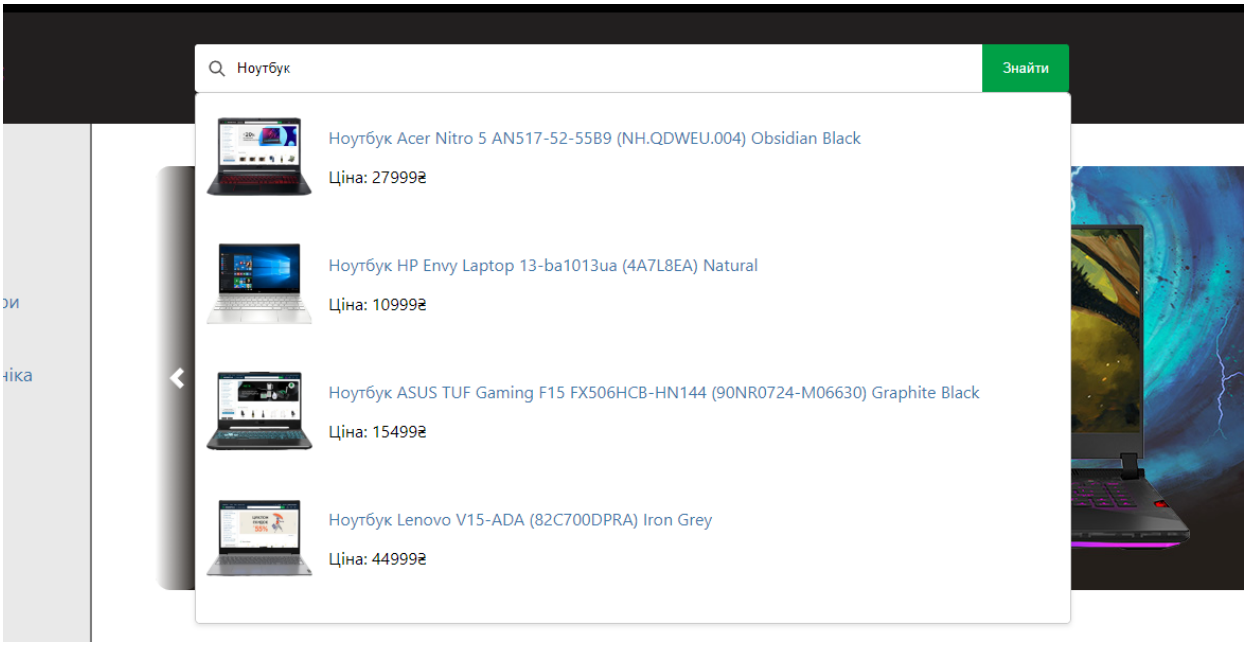

Рисунок 3.27 – Результат пошуку «Ноутбук»

Далі було проведено другий тест, введено в поле пошуку введемо «Xiaomi».

На рисунку 3.28 можна побачити, що результати за пошуком пройшли успішно. Даний тест показує різноманітність товарів на сайті.

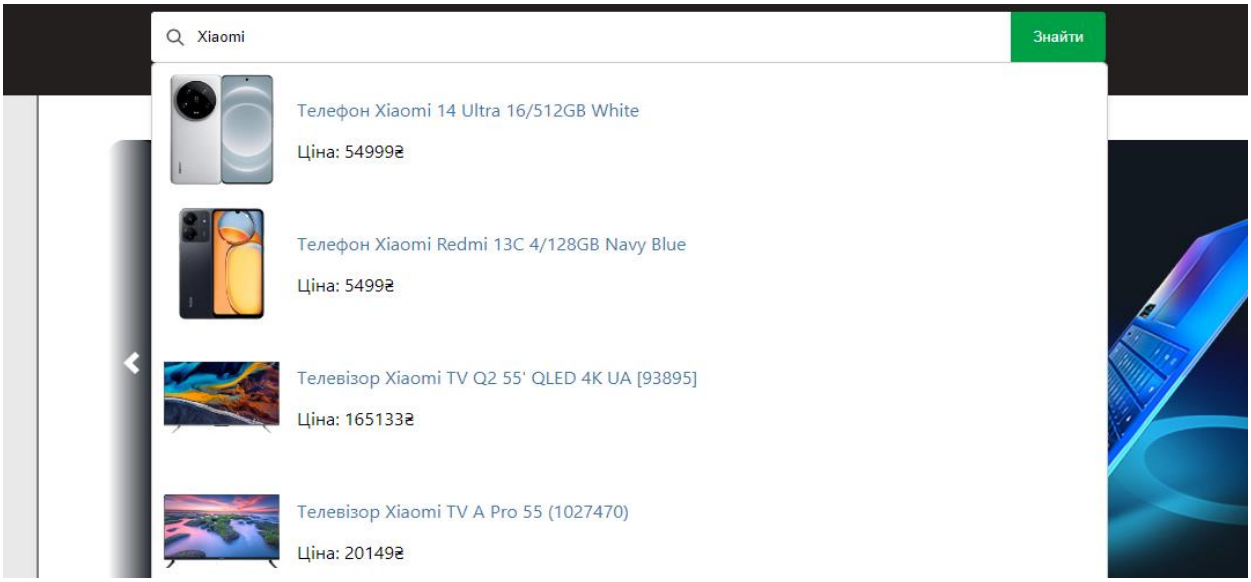

Рисунок 3.28 – Результат пошуку «Xiaomi»

Друга частина. Ліва частина сайту – це каталог, в якому розміщується логотипи та назви каталогів товару (рис. 3.29).

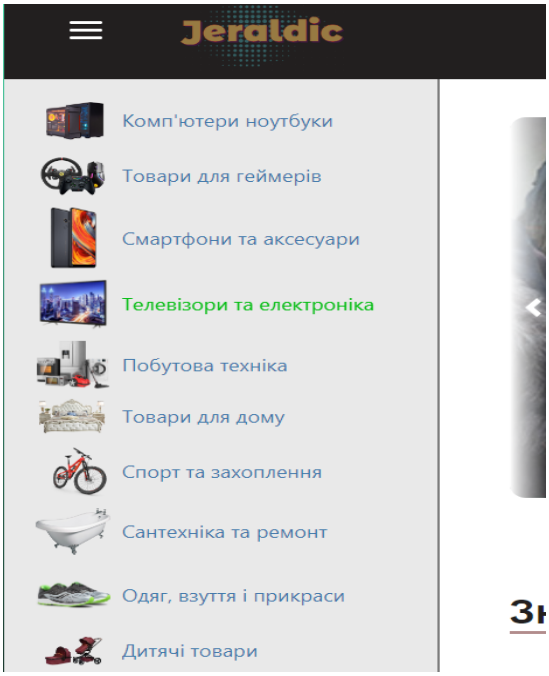

Рисунок 3.29 – Каталог товарів

Третя частина. По центру розташований головний слайдер, який автоматично змінюється на новий слайд кожні 5 секунд, це можна побачити на рисунках 3.30 – 3.31.

Рисунок 3.30 – Перший слайд який заміниться через 5 секунд

Рисунок 3.31 – Другий слайд, на який замінився 1 слайд

Четверта частина. Знизу сторінки розташовані два додаткові слайдери. Один демонструє нові товари, що додалися най сайт, а другий – товари зі знижками. Їх функціонування відрізняється від головного слайдера. Кожні 5 секунд один товар ліворуч зникає, а праворуч з'являється новий. Функціонування можна побачити на рисунках 3.32 – 3.34.

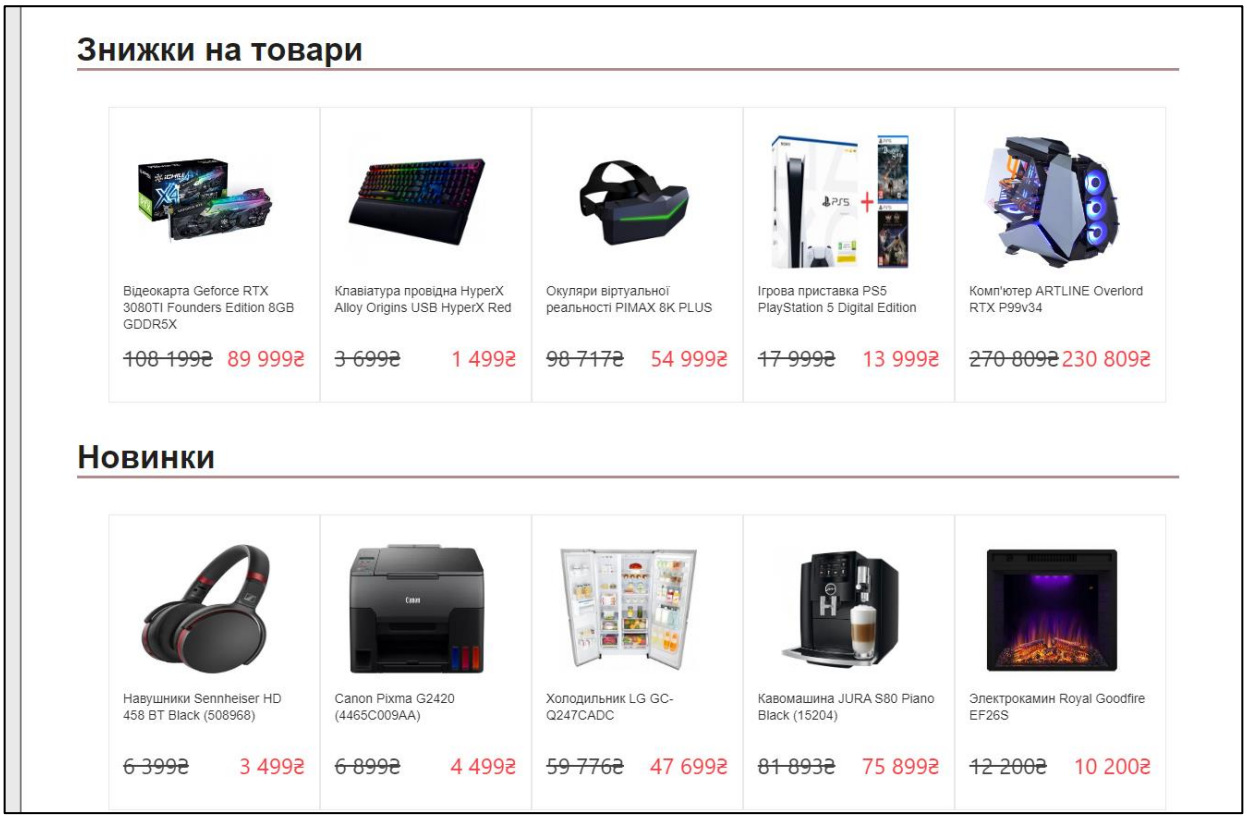

Рисунок 3.32 – Слайдери до того, як активували заміну слайду зліва

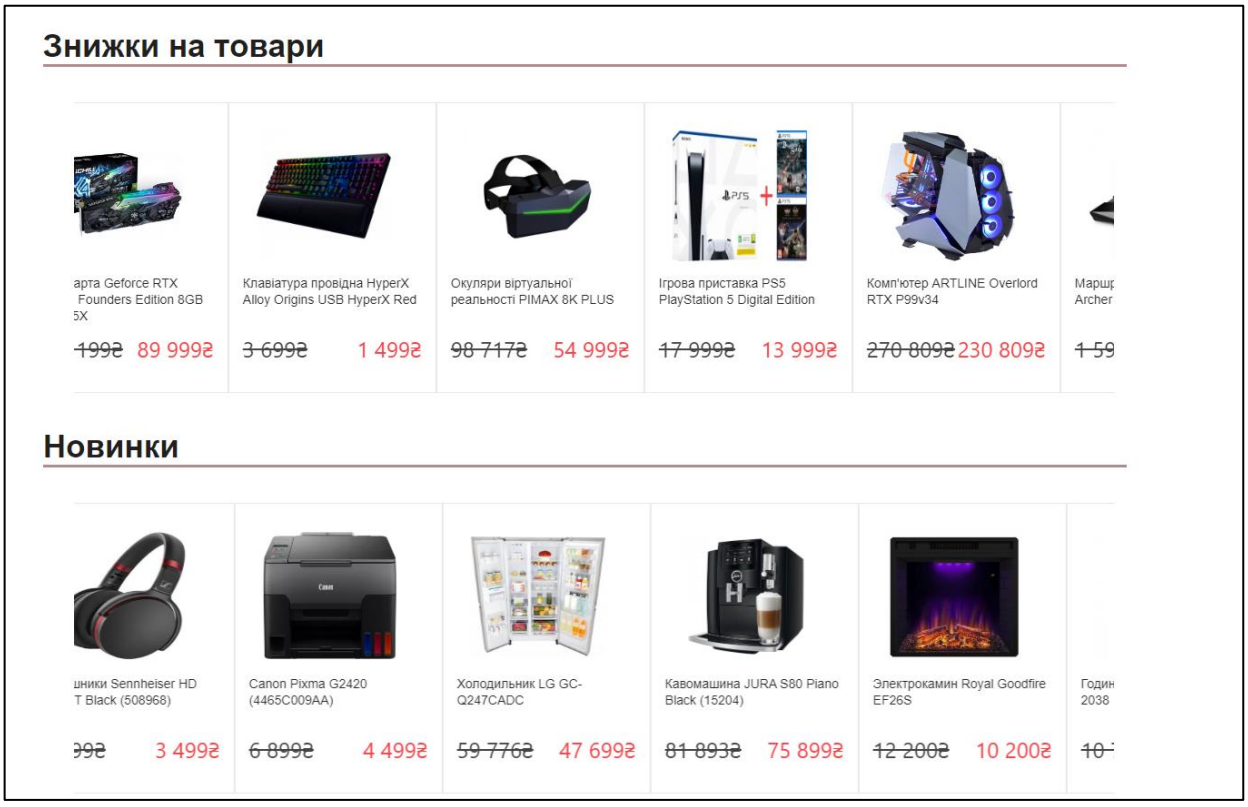

Рисунок 3.33 – Слайдери під час активації, видно що зліва зникає товар, справа з’являється

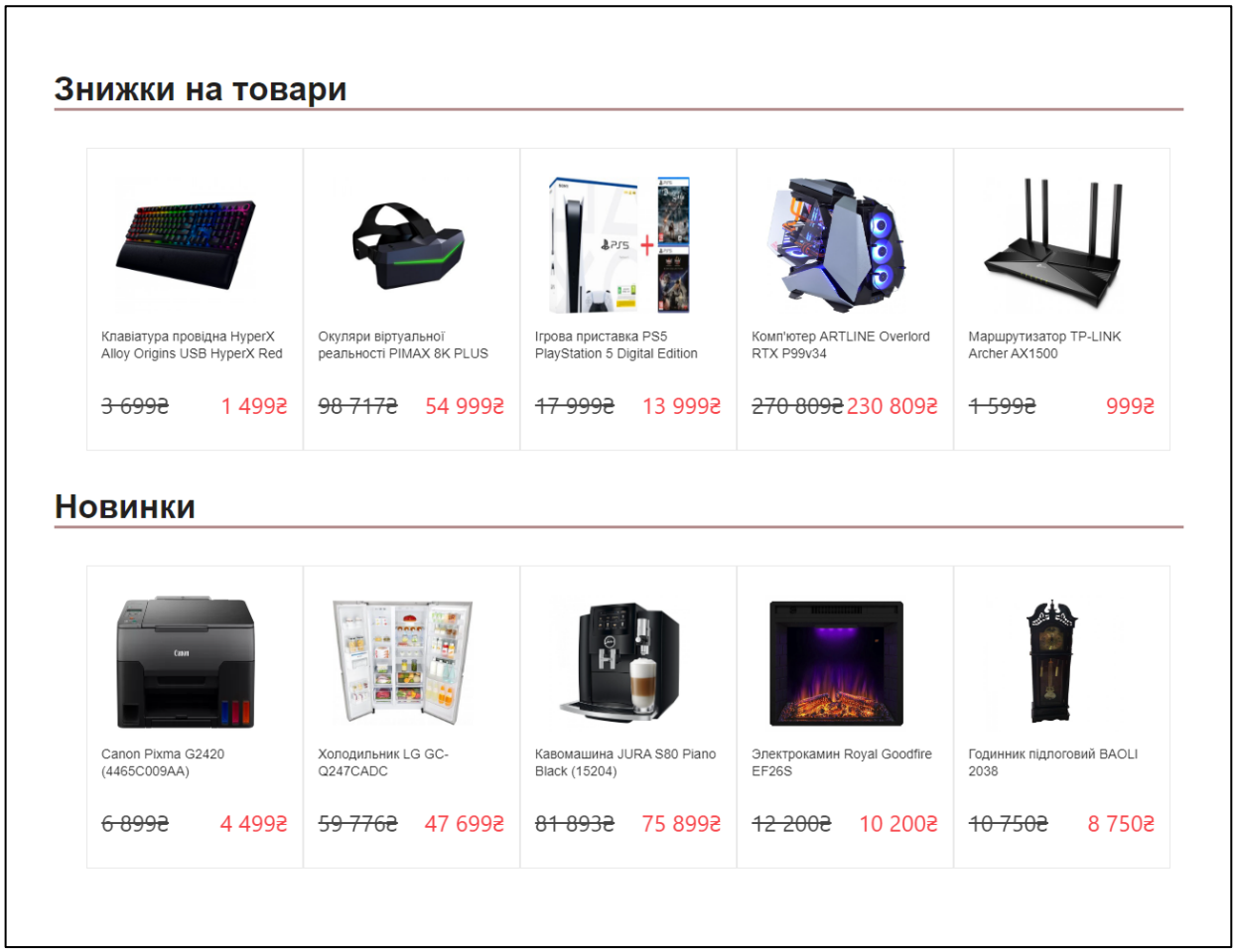

Рисунок 3.34 – Слайдер закінчив свою активацію

В цьому підрозділі було протестовано увесь функціонал та візуальний вигляд головної сторінки. Тести були проведені для слайдерів та поле пошуку.

1.2.2 Тестування сторінки каталогу товарів

В цьому розділі буде протестовано каталог товарів, який знаходиться ліворуч в головній сторінці, наприклад в «Телевізори та електроніка», і як виглядає сторінка та які категорії товарів існують.

Як можна побачити на рисунку 3.35, різноманітні категорії в головному каталозі «Телевізори та електроніка», тут можна побачити назви кожної категорії та їх логотипи для більшої ясності, які саме товари в цій категорії будуть.

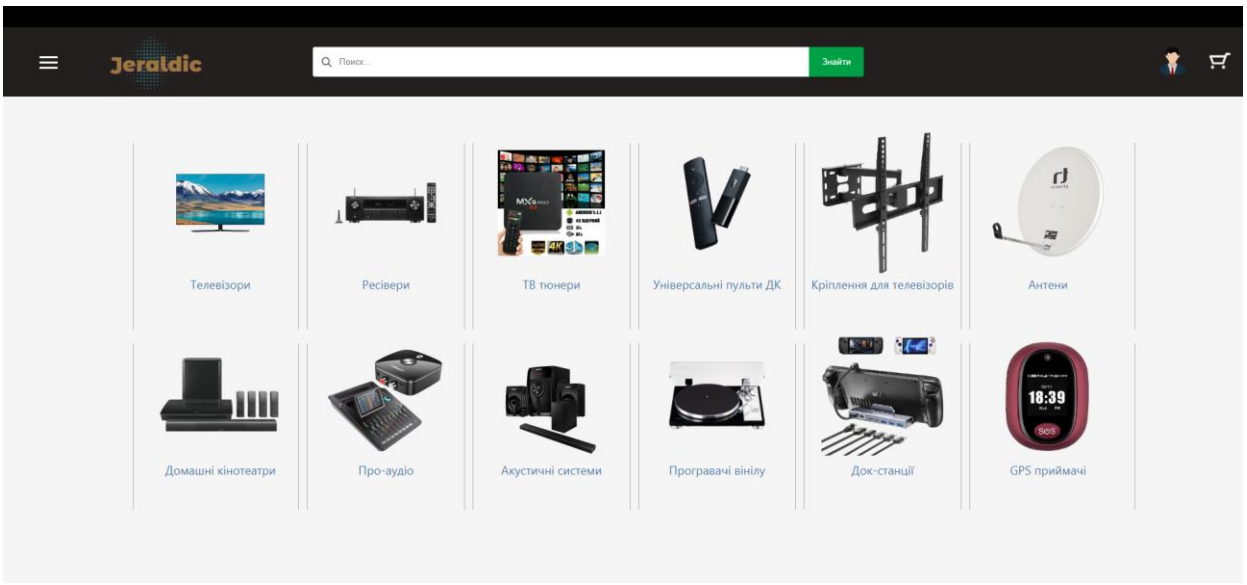

Рисунок 3.35 – Категорії товарів в каталозі «Телевізори та електроніка»

1.2.3 Тестування сторінки товарів

Далі було протестовано сторінку товарів. Є 3 сторінки товарів, а саме: сторінка ноутбуків, телевізорів, телефонів.

Розглянемо на прикладі ноутбуків, як виглядає сторінка, функціонування, тестування.

Як можна побачити на рисунку 3.36, на сайті є такі функціонали: навігаційне меню, за допомогою якого можна повернутися на попередні сторінки; меню фільтрації, яке знаходиться на лівій стороні сторінки, й самі товари, які виводяться за допомогою БД та API. Кожен товар є унікальним, в кожного товару є своя назва, ціна, рейтинг та кнопка, яка додає товар до кошику.

Протестовано використання фільтрація, яка була реалізовані.

Фільтрація за ціною: Від 7000 до 12000 тисяч.

На рисунку 3.37 можна побачити що товари ноутбуків виводяться лише ті які вказані за ціновим діапазоном.

Далі буде протестовано фільтрацію за брендом. Обрані бренди: Asus, Apple.

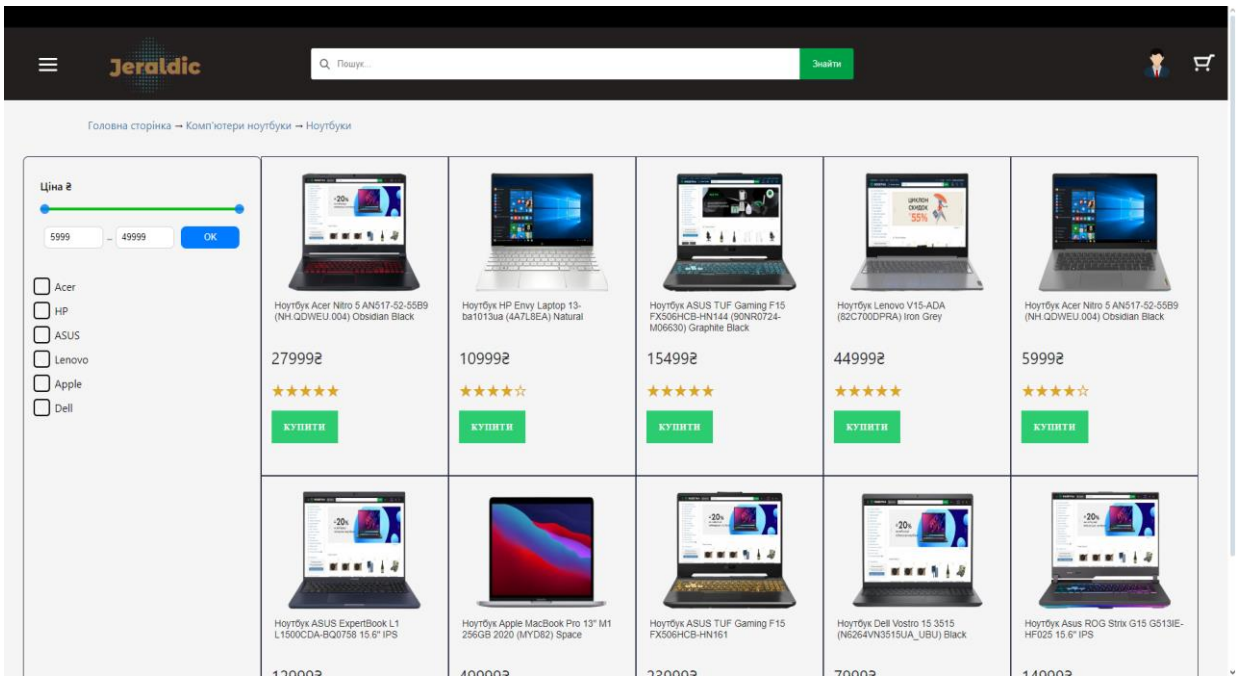

Рисунок 3.36 – Початковий вигляд сторінки ноутбуків

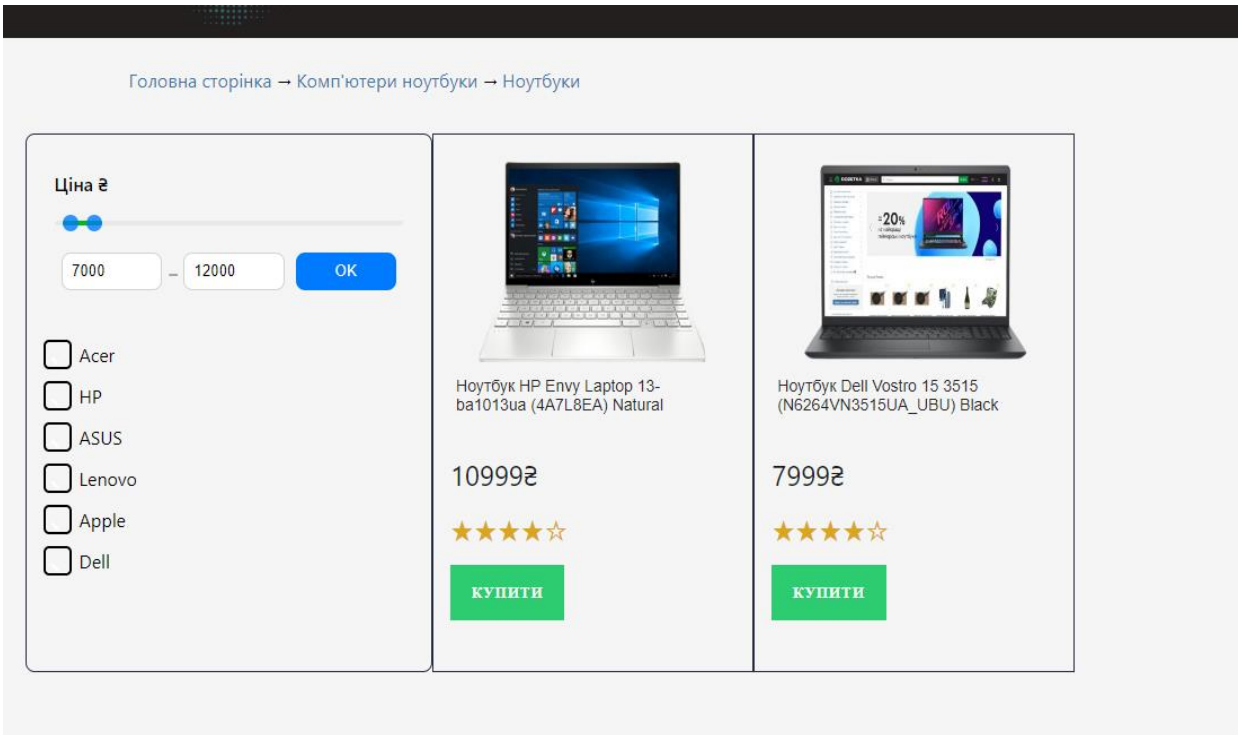

Рисунок 3.37 – Результат фільтрації за ціною

На рисунку 3.38 можна побачити, що в полі фільтрів за брендом, галочкою були обрані бренди Asus, Apple. В полі товарів можна побачити, що товари виводяться тільки ті, які були обрані у фільтрі.

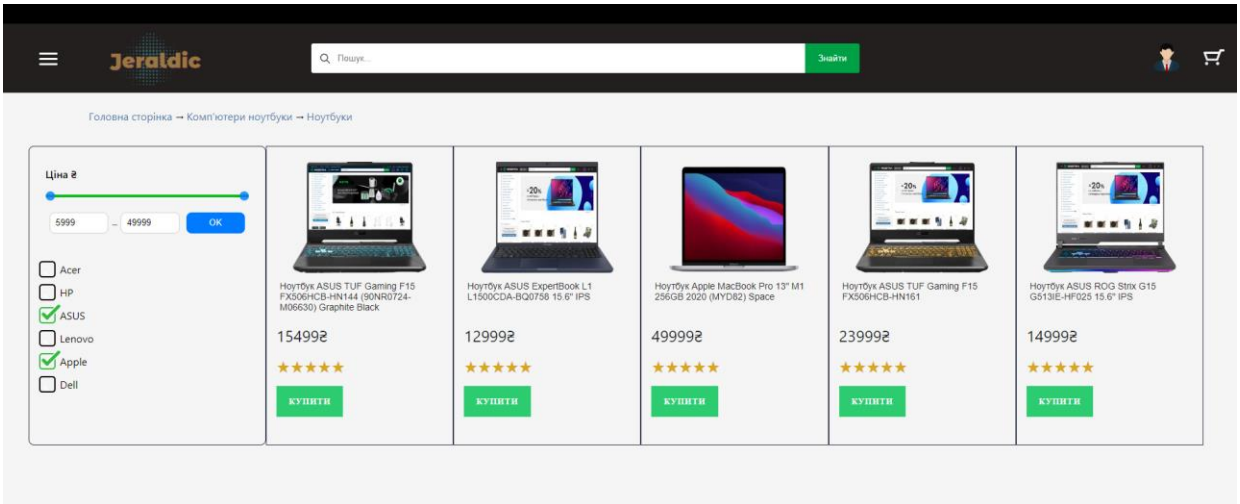

Рисунок 3.38 – Результат фільтрація за брендом

Далі протестовано одночасне використання двох фільтрів:

– фільтр за ціною: діапазон 5к-15к;

– фільтр за брендом: Acer, HP, Dell.

Фільтрація за ціною та брендом прекрасно разом працюють. За результатами наведеними на рисунку 3.39, можна побачити, що даний тест знайшов за максимальним й мінімальним діапазоном вказані бренди ноутбуків.

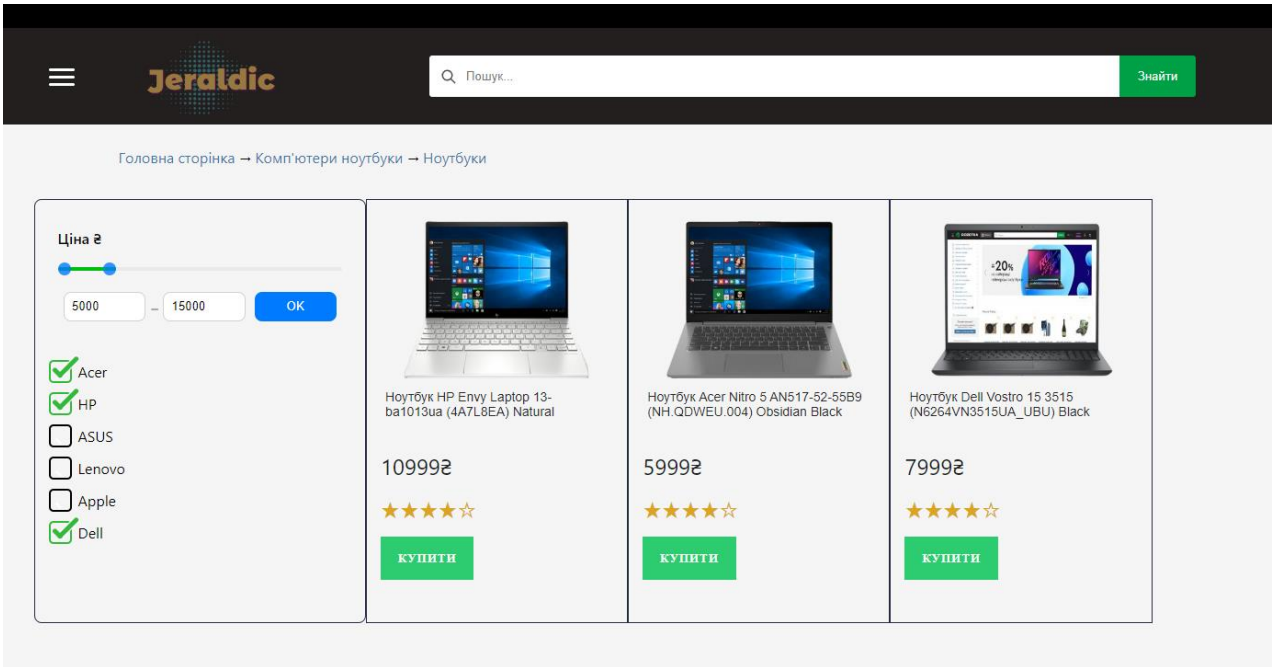

Рисунок 3.39 – Результат фільтрація за ціною та брендом

Далі тестування охоплює додавання декількох товарів у кошик. Після цього у верхньому правому кутку знаходиться іконка кошика. Щоб додати товар до кошика, потрібно натиснути кнопку «Купити», ця кнопка є над кожним товаром.

Протестуємо кошик, додавши перші два товари.

Як видно на рисунку 3.40, після натискання на кнопку «Купити» на перші два товари і переходу до сторінки кошика, можна побачити обрані товари, їх логотип, назву, кількість, ціну, загальну ціну, а також кнопки видалення товару та оформлення замовлення.

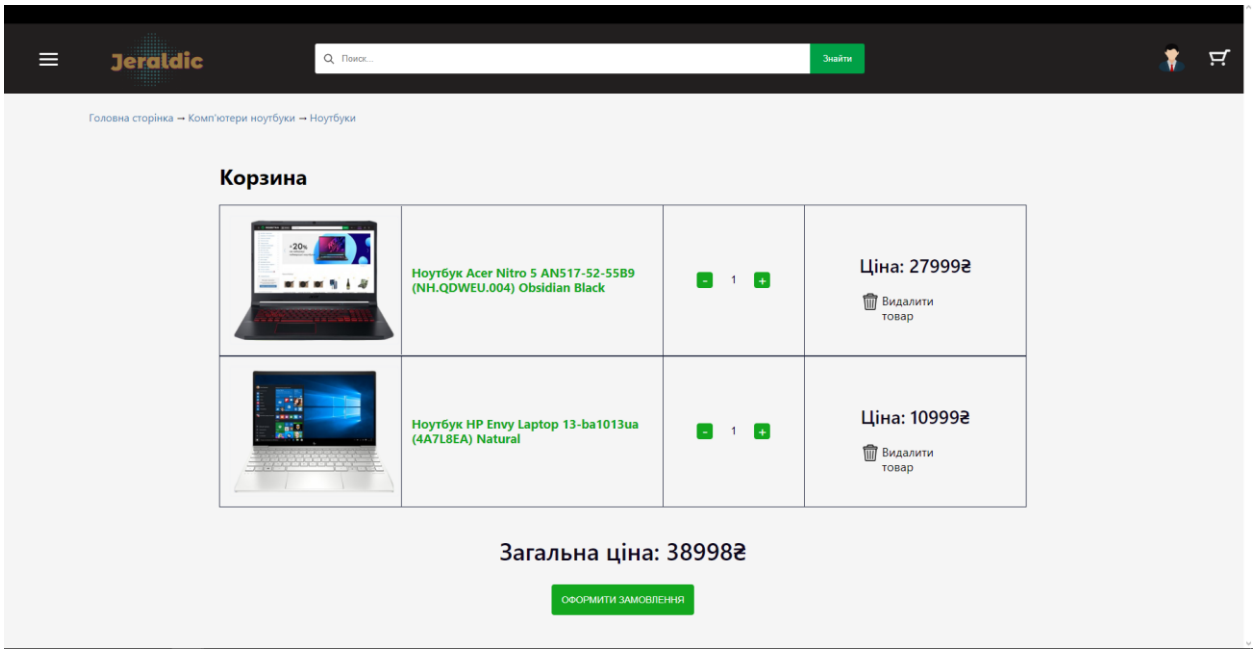

Рисунок 3.40 – Сторінка кошика після додавання товару

Наступний тест передбачає додавання кількості для першого ноутбуку до трьох і розглянемо, як змінюється загальна ціна.

Після змінення кількості першого товару, можна побачити на рисунку 3.41, що загальна ціна змінилася.

Далі протестуємо як видаляється другий товар із кошика.

На рисунку 3.42, можна побачити, що після натискання на кнопку «Видалити товар», товар зникає повністю.

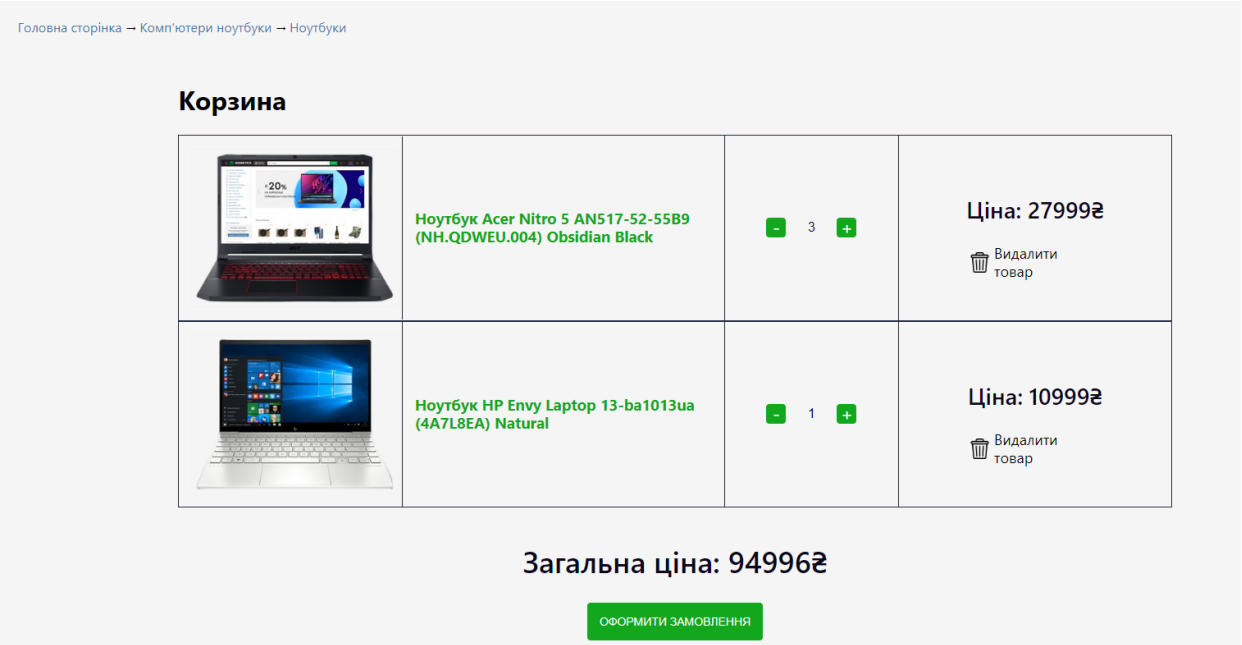

Рисунок 3.41 – Результат після зміни кількості першого товару до трьох

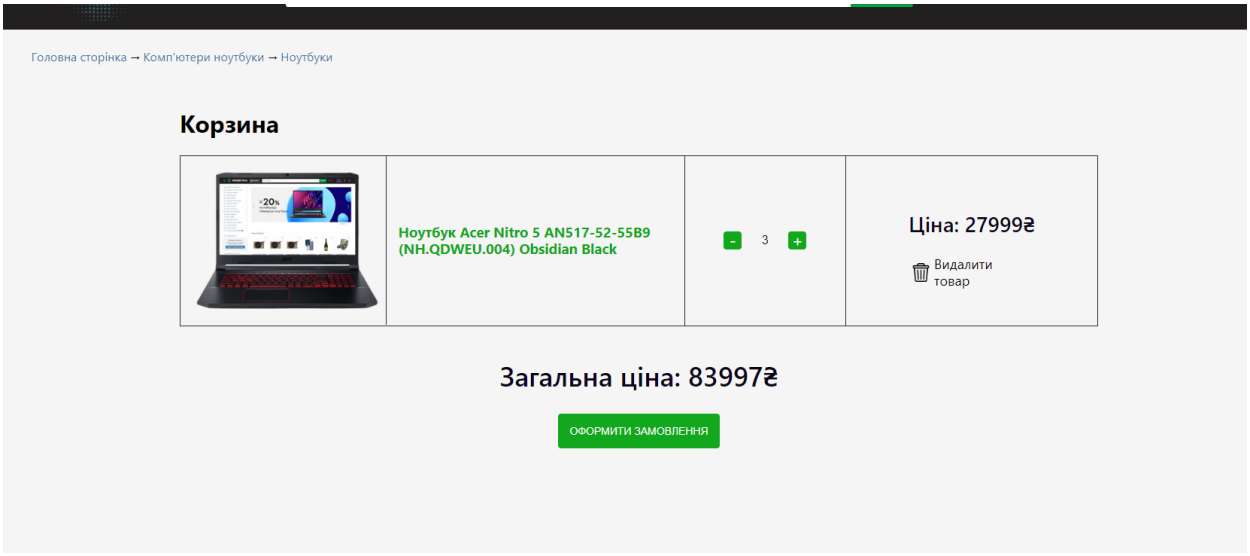

Рисунок 3.42 – Результат після видалення другого товару з кошика

1.3 Висновки до розділу 3

Розділ 3 детально розглядає реалізацію проєкту інтернет-магазину, надаючи глибоке занурення в процес розробки кожної сторінки. Докладний аналіз програмного коду, включаючи компоненти фронтенду та бекенду, дає чітке уявлення про архітектуру та функціональність системи.

На прикладі сторінки «Ноутбуки» було показано, як React-компоненти забезпечують інтерактивність та динамічність інтерфейсу, а API-зв’язок з бекендом на Node.js та Mysql дає можливість отримувати та оновлювати дані про товари.

Тестування проведене на головній сторінці, сторінці каталогу та сторінці товарів підтвердило ефективну роботу реалізованих функцій, таких як пошук, фільтрація, додавання до кошика та обробка товарів у кошику.

Розділ 3 не лише демонструє технічну реалізацію проєкту, але й підкреслює важливість чіткого розподілу відповідальності між фронтендом та бекендом для створення зручного та функціонального інтернет-магазину.

ВИСНОВКИ

Дана кваліфікаційна робота на тему «Розробка інтернет-магазину з використанням HTML, CSS, Java Script, Node.js, Mysql» дає можливість майбутнім розробникам більш детально розглянути розробку подібних проєктів.

За допомогою даного проєкту було досягнуто наступних цілей.

Реалізація інтерфейс користувача: вдалося створити простий інтерфейс з використанням HTML, CSS та Java Script. Даний інтерфейс працює коректно, але вимагає додаткової оптимізації для кращого зовнішнього вигляду.

Реалізують функціонал за критеріями: пошук товарів, перегляд товарів, додавання до кошика. Однак в майбутньому, деякі з них потребують оптимізації.

Реалізовано backend частину, а саме створене просте API на Node.js, яке взаємодіє з базою даних Mysql. Ця частина проєкту виявилася більш складною, ніж очікувалося, в подальшому потребує додаткових зусиль для поліпшення стабільності та продуктивності.

Для створення даного проєкту було набуто досвід із таких мов програмування та фреймворків:

– HTML, CSS, Java Script: покращення роботи розмітки та стилізацію сторінок, додавання функціоналу для стилізації;
– Node.js: навички поєднування бекенду та фронтенду для взаємодії;
– MySQL: розробка бази даних для подібних вебзастосунків.

В майбутньому планується продовжити дану розробку, інтернет-магазин, для подальшого додавання нових функцій та покращення зовнішнього вигляду. Планується додавання реєстрації користувачів, функції оплати та доставки, розширення асортименту, додатковий опис для кожного товару на сторінці – своя сторінка для кожного товару.

Планується оптимізація швидкості та продуктивності застосунку, для зручності користувачів.

ПЕРЕЛІК ПОСИЛАНЬ

1. React.js. URL: https://legacy.reactjs.org/docs/getting-started.html (дата звернення: 29.02.2024).
2. JavaScript. URL: https://www.w3schools.com/jsrEF/default.asp (дата звернення: 25.03.2024).
3. Node.js. URL: https://nodejs.org/en/learn/getting-started/introduction-to-nodejs (дата звернення: 04.04.2024).
4. Morales G. API with Node.js, Express and Prisma. 2023. 224 p.
5. HTML and CSS QuickStart Guide: The Simplified Beginners Guide to Developing a Strong Coding Foundation, Building Responsive Websites, and Mastering ... (Coding & Programming – QuickStart Guides). 2021. 352 p.
6. Fielding J. Beginning Responsive Web Design with HTML5 and CSS3. New York : Apress, 2014. 304 p.
7. JavaScript: The Definitive Guide: Master the World’s Most-Used Programming Language. O’Reilly Media, 2020. 704 p.
8. Mastering MySQL Administration: High Availability, Security, Performance, and Efficiency. Apress, 2024. 754 p.
9. Node.js: The Comprehensive Guide to Server-Side JavaScript Programming (Rheinwerk Computing). Rheinwerk Computing, 2022. 834 p.
10. Learning Web Design: A Beginner’s Guide to HTML, CSS, JavaScript, and Web Graphics. O’Reilly Media, 2018. 808 p.

ДОДАТОК А

Реалізація програмного коду

```
import React, { useEffect, useRef, useState } from 'react';
import './css/App.css'; // Імпорт стилів для головного компонента
import ChiefSlider from './chief-slider'; // Імпорт компонента ChiefSlider
import CustomSlider from './CustomSlider_discount'; // Імпорт компонента CustomSlider для знижок
import CustomSlider_news from './CustomSlider_news'; // Імпорт компонента CustomSlider_news для новин
import SearchResults from './search/SearchResults'; // Імпорт компонента SearchResults для відображення результатів пошуку

```

Рисунок А.1 – Імпорт документів та бібліотек на сторінці Homepage

```
function Homepage() {
  const [activeSlideIndex, setActiveSlideIndex] = useState(0); // Стан для визначення активного індексу слайду
  const [searchTerm, setSearchTerm] = useState(''); // Стан для збереження терміну пошуку
  const [searchResults, setSearchResults] = useState([]); // Стан для збереження результатів пошуку

  const handleSlideChange = (index) => { // Функція обробки зміни слайду
  setActiveSlideIndex(index); // Встановлення активного індексу слайду
  };

  const mainSliderRef = useRef(null); // Створення посилання для головного слайдера
  useEffect(() => { // Ефект для ініціалізації головного слайдера
    const slider = new ChiefSlider('.slider', { // Створення нового екземпляра ChiefSlider
      loop: true, // Зациклювання слайдів
      autoplay: true, // Автовідтворення слайдів
      interval: 5000, // Інтервал автовідтворення (5000 мілісекунд = 5 секунд)
    });
  }, []); // Запуск ефекту лише після монтування компонента

  useEffect(() => { // Ефект для отримання результатів пошуку
    const fetchResults = async () => { // Асинхронна функція для отримання результатів пошуку
      try {
        // Виконання запиту на сервер для отримання результатів пошуку
        const response = await fetch(`http://localhost:3024/api_search/search?query=${encodeURIComponent(searchTerm)}`);
        // Логування статусу відповіді
        console.log('Response status:', response.status);
        if (!response.ok) { // Перевірка на успішність відповіді
          throw new Error('Network response was not ok'); // Викидання помилки, якщо відповідь не успішна
        }
        const contentType = response.headers.get('content-type'); // Отримання типу контенту відповіді
        console.log('Content-Type:', contentType); // Логування типу контенту
        if (contentType && contentType.indexOf('application/json') !== -1) { // Перевірка, чи тип контенту - JSON
          const data = await response.json(); // Парсинг відповіді у форматі JSON
          console.log('Search results:', data); // Логування результатів пошуку
          setSearchResults(data); // Збереження результатів пошуку у стані
        } else {
          throw new Error('Response is not in JSON format'); // Викидання помилки, якщо формат відповіді не JSON
        }
      } catch (error) {
        console.error('Ошибка при получении результатов поиска:', error); // Обробка помилки
      }
    };

    if (searchTerm) { // Перевірка, чи є термін пошуку
      fetchResults(); // Виклик функції отримання результатів пошуку
    } else {
      setSearchResults([]); // Якщо термін пошуку відсутній, очищення результатів пошуку
  }
  }, [searchTerm]); // Запуск ефекту при зміні терміну пошуку

```

Рисунок А.2 – Головні функції слайдерів та живого пошуку на сторінці Homepage

```jsx
return (
    <div className="main_site">
      {/* Верхня частина сайту */}
      <div className="container_logo_top">
        {/* Логотип */}
        <img className="logo" src="" alt="" />
      </div>
      {/* Верхня панель з кнопкою меню, логотипом і полем пошуку */}
      <div className="container_top">
        {/* Кнопка "бургер" для відкриття меню */}
        <div>
          <button className="burger">
            <span></span>
            <span></span>
            <span></span>
          </button>
        </div>
        {/* Логотип сайту */}
        <div className="wrap">
          <a href="#">
            <img className="logo_sites" src="images/logo_sites.png" alt="" />
          </a>
        </div>
        {/* Поле пошуку з результатами */}
        <div className="search_input">
          <form className="form_search search_input" onSubmit={(e) => e.preventDefault()}>
            <div className="search_form_inner">
              <input
                autoComplete="off"
                className="search_pole"
                type="text"
                id="search"
                placeholder="Поиск..."
                value={searchTerm}
                onChange={(e) => setSearchTerm(e.target.value)}
              />
              {/* Відображення результатів пошуку */}
              <SearchResults results={searchResults} />
              {/* Кнопка пошуку */}
              <button id="" className="submit" type="submit" name="Найти">
                Знайти
              </button>
            </div>
          </form>
        </div>
        {/* Блок з іконками користувача, кошика */}
        <div className="block_header">
          <a className="header" href="">
            <img className="icon_avatar_cleave" src="images/avatar.png" alt="" />
          </a>
          <a className="header" href="#">
            <img className="icon_avatar_cleave" src="images/cleave.png" alt="" />
          </a>
        </div>
      </div>
```

Рисунок А.3 – Верхня частина сайту з полем пошуку, іконками користувача, кошика

```jsx
{/* Основний контент */}
<div className="container_main">
  {/* Меню категорій */}
  <div id="menu" className="catalog div-active">

    <ul className="catalog_list">
      {/* Пункти меню */}
      <li>
      <a className="menu" href="/computer-notebook">
        <img className="photo_logo" src="images/burger/computer.png" alt="Computer and notebook" />
        Комп'ютери ноутбуки
      </a>
      </li>
      <li>
        <a className="menu" href="Computer_Notebook.html">
          <img className="photo_logo" src="images/burger/gamer.png" alt="Computer and notebook" />
          Товари для геймерів
        </a>
      </li>
      <li>
        <a className="menu" href="/Smartphones_accessories">
          <img className="photo_logo" src="images/burger/phone.png" alt="Computer and notebook" />
          Смартфони та аксесуари
        </a>
      </li>
      <li>
        <a className="menu" href="/TvElectronics">
          <img className="photo_logo" src="images/burger/tv.png" alt="Computer and notebook" />
          Телевізори та електроніка
        </a>
      </li>
      <li>
        <a className="menu" href="Computer_Notebook.html">
          <img className="photo_logo" src="images/burger/icebox.png" alt="Computer and notebook" />
          Побутова техніка
        </a>
      </li>
      <li>
        <a className="menu" href="Computer_Notebook.html">
          <img className="photo_logo" src="images/burger/home.png" alt="Computer and notebook" />
          Товари для дому
        </a>
      </li>
      <li>
        <a className="menu" href="Computer_Notebook.html">
          <img className="photo_logo" src="images/burger/bike.png" alt="Computer and notebook" />
          Спорт та захоплення
        </a>
      </li>
      <li>
        <a className="menu" href="Computer_Notebook.html">
          <img className="photo_logo" src="images/burger/washing.png" alt="Computer and notebook" />
          Сантехніка та ремонт
        </a>
      </li>
      <li>
        <a className="menu" href="Computer_Notebook.html">
          <img className="photo_logo" src="images/burger/boots.png" alt="Computer and notebook" />
          Одяг, взуття і прикраси
        </a>
      </li>
      <li>
        <a className="menu" href="Computer_Notebook.html">
          <img className="photo_logo" src="images/burger/baby.png" alt="Computer and notebook" />
          Дитячі товари
        </a>
      </li>
    </ul>
  </div>
```

Рисунок А.4 – Меню категорій товарів на сторінці Homepage

```
{/* Контейнер для слайдерів */}
<div className="container">
  {/* Головний слайдер */}
<div className="slider" ref={mainSliderRef}>
    <div className="slider__wrapper">
      <div className="slider__items">
        {/* Слайди */}
        <div className="slider__item">

        </div>
        <div className="slider__item">

        </div>
        <div className="slider__item">

        </div>
        <div className="slider__item">

        </div>
        <div className="slider__item">

        </div>
        <div className="slider__item">

        </div>
        <div className="slider__item">

        </div>
        <div className="slider__item">

        </div>
        <div className="slider__item">

        </div>
        <div className="slider__item">

        </div>
        <div className="slider__item">

        </div>
      </div>
    </div>

<a href="#" className="slider__control" data-slide="prev"></a>
<a href="#" className="slider__control" data-slide="next"></a>
<ol className="slider__indicators">
  <li data-slide-to="0"></li>
  <li data-slide-to="1"></li>
  <li data-slide-to="2"></li>
  <li data-slide-to="3"></li>
  <li data-slide-to="4"></li>
  <li data-slide-to="5"></li>
  <li data-slide-to="6"></li>
  <li data-slide-to="7"></li>
  <li data-slide-to="8"></li>
  <li data-slide-to="9"></li>
</ol>
</div>
```

Рисунок А.5 – Головний слайдер на сторінці Homepage

```
245                 {/* Блок зі знижками на товари */}
246                 <h2 className='new_tovar'>Знижки на товари</h2>
247                 <div className='block_tovarov'>
248                 {/* Власний компонент слайдера для знижок */}
249                 {/* activeSlideIndex - індекс активного слайду */}
250                 {/* onSlideChange - функція для зміни активного слайду */}
251                 <CustomSlider activeSlideIndex={activeSlideIndex} onSlideChange={handleSlideChange} />
252                 </div>
253
254                 {/* Блок з новинками */}
255                 <h2 className='new_tovar'>Новинки</h2>
256                 {/* Власний компонент слайдера для новин */}
257                 {/* activeSlideIndex - індекс активного слайду */}
258                 {/* onSlideChange - функція для зміни активного слайду */}
259                 <div className='block_tovarov'>
260                 <CustomSlider_news activeSlideIndex={activeSlideIndex} onSlideChange={handleSlideChange} />
261                 </div>
262             </div>
263           </div>
264         </div>
265           );
266         }
```

Рисунок А.6 – Функції виклику слайдерів знижок, новинок на сторінці Homepage

```
1   import React, { useRef } from 'react';
2   import Slider from 'react-slick'; // Імпортуємо компонент слайдера
3   import 'slick-carousel/slick/slick.css'; // Імпортуємо стилі для слайдера
4   import 'slick-carousel/slick/slick-theme.css'; // Імпортуємо тему для слайдера
5   import './css/App.css'; // Імпортуємо власні стилі
```

Рисунок А.7 – Імпорт документів та бібліотек на сторінці CustomSlider_discount

```
7   function CustomSlider({ className }) {
8     const sliderRef = useRef(null); // Створюємо посилання на компонент слайдера
9     const settings = {
10      // Налаштування слайдера
11      infinite: true, // Безкінечний режим
12      speed: 5000, // Швидкість зміни слайдів (мс)
13      autoplay: true, // Автовідтворення слайдів
14      slidesToShow: 5, // Кількість слайдів, що відображаються одночасно
15      slidesToScroll: 1, // Кількість слайдів, які перегортаються за раз
16      arrows: false, // Вимкнення стрілок перегортування
17      responsive: [
18        // Адаптивні налаштування для різних розмірів екрану
19        {
20          breakpoint: 1024, // Поріг ширини екрану
21          settings: {
22            slidesToShow: 1, // Кількість слайдів для екранів ширше 1024px
23          },
24        },
25        {
26          breakpoint: 768, // Поріг ширини екрану
27          settings: {
28            slidesToShow: 2, // Кількість слайдів для екранів ширше 768px
29          },
30        },
31        {
32          breakpoint: 480, // Поріг ширини екрану
33          settings: {
34            slidesToShow: 3, // Кількість слайдів для екранів ширше 480px
35          },
36        },
37      ],
38    };
39
40    // Функція для обробки прокрутки слайдера за допомогою колеса миші
41    const handleSliderScroll = (e) => {
42      if (sliderRef.current) {
43        // Якщо компонент слайдера існує
44        sliderRef.current.slickGoTo(sliderRef.current.innerSlider.state.currentSlide + e.deltaY / 100); // Прокрутити на відстань, залежну від переміщення колеса миші
45      }
46    };
```

Рисунок А.8 – Функції налаштування слайдера на сторінці CustomSlider_discount

```
return (
  // Компонент слайдера з налаштуваннями та класом
  <Slider {...settings} className={`custom-slider ${className}`} ref={sliderRef} onWheel={handleSliderScroll}>
    {/* Кожен слайд товару */}
    <div className='new_tovar_block'>
      {/* Логотип товару */}
      <div className='src_logo_new_tovar'>
        <a className="" href="">
          <img className="tovar_new logo_catalog" src="images/new_tovar/logo_1.jpeg" alt="" />
        </a>
      </div>
      {/* Назва товару */}
      <a className="text_tovar text_tovar_new" href="#">Відеокарта Geforce RTX 3080TI Founders Edition 8GB GDDR5X</a>
       {/* Цінова інформація */}
      <div className='Money'>
        {/* Стара ціна */}
        <div className="text_money_old">
          <span>108 199₴</span>
        </div>
         {/* Нова ціна */}
        <div className="text_money_new">
          <span>89 999₴</span>
        </div>
      </div>
    </div>
    {/* Далій йде те саме, але інша назва ціна та логотип */}
```

Рисунок А.9 – Блоки товарів для слайдера з усіма компонентами функціонування

```
import React from 'react';

function SearchResults({ results }) {
  // Логуємо результати пошуку в консоль
  console.log('Results:', results);

  // Перевіряємо, чи є результати пошуку
  if (results.length === 0) {
    // Якщо результати відсутні, повертаємо null (нічого не відображаємо)
    return null;
  }

  // Якщо є результати, відображаємо їх
  return (
    // Блок результатів пошуку
    <div className="search-results">
      {/* Відображення кожного результату окремо */}
      {results.map((result) => (
        <div className="search-results-item" key={result.type + '-' + result.id}>
          {/* Зображення результату */}
          <img
            className="search-results-image"
            src={`catalog/tovar/${result.type}/${result.id}/logo_1.png`}
            alt={result.name}
          />
          {/* Деталі результату */}
          <div className="search-results-details">
            {/* Назва результату */}
            <p className="search-results-name"><a>{result.name}</a></p>
            {/* Ціна результату */}
            <p className="search-results-price">Ціна: {result.price}₴</p>
          </div>
        </div>
      ))}
    </div>
  );
}

export default SearchResults;
```

Рисунок А.10 – Функціональне налаштування та відображення результата за пошуком

```
// Імпортуємо необхідні модулі
const express = require('express'); // Express.js для створення серверу
const mysql = require('mysql'); // MySQL для взаємодії з базою даних

// Створюємо маршрутизатор Express
const router = express.Router();

// Підключаємося до бази даних MySQL
const db = mysql.createConnection({
  host: 'localhost', // Хост бази даних
  user: 'root', // Ім'я користувача бази даних
  password: 'password', // Пароль користувача бази даних
  database: 'shop', // Назва бази даних
  charset: 'utf8_general_ci' // Кодування бази даних
});

// Встановлюємо з'єднання з базою даних і перевіряємо стан підключення
db.connect((err) => {
  if (err) {
    console.error('Ошибка подключения к базе данных:', err); // Виводимо помилку підключення
    return;
  }
  console.log('Подключение к базе данных успешно'); // Повідомляємо про успішне підключення
});

```

Рисунок А.11 – Функція підключення до бази даних з перевіркою

```
// Маршрут обробки GET-запиту для пошуку
router.get('/search', (req, res) => {
  // Отримуємо сировий пошуковий термін з запиту
  const rawSearchTerm = req.query.query;
  console.log('Raw search term:', rawSearchTerm);

  try {
    // Декодуємо пошуковий термін
    const searchTerm = decodeURIComponent(rawSearchTerm);
    console.log('Decoded search term:', searchTerm);

    // Якщо пошуковий термін порожній, повертаємо пустий масив
    if (!searchTerm) {
      console.log('Empty search term, returning empty array');
      return res.status(200).json([]); // Відправляємо пустий масив з кодом 200
    }

    // Запити для пошуку в різних таблицях бази даних
    const queries = [
      'SELECT id, name, price, "smartphone" as type FROM smartphone WHERE name LIKE ?',
      'SELECT id, name, price, "tv" as type FROM tv WHERE name LIKE ?',
      'SELECT id, name, price, "product" as type FROM products WHERE name LIKE ?'
    ];

    // Створюємо обіцянки для кожного запиту
    const promises = queries.map(query => {
      return new Promise((resolve, reject) => {
        // Виконуємо запит до бази даних
        db.query(query, [`%${searchTerm}%`], (err, results) => {
          if (err) {
            console.error('Ошибка запроса к базе данных:', err); // Виводимо помилку запиту
            return reject(err);
          }
          console.log(`Query results for query (${query}):`, results); // Виводимо результати запиту
          resolve(results);
        });
      });
    });
```

Рисунок А.12 – Створення функції запиту API до БД пошуку з перевірками цих запитів

```
    // Очікуємо виконання всіх обіцянок
    Promise.all(promises)
      .then(results => {
        // Об'єднуємо результати запитів
        const mergedResults = [].concat(...results);
        console.log('Merged results:', mergedResults); // Виводимо об'єднані результати
        res.setHeader('Content-Type', 'application/json');
        res.json(mergedResults); // Відправляємо результати у форматі JSON
      })
      .catch(err => {
        console.error('Ошибка выполнения запросов к базе данных:', err); // Виводимо помилку виконання запитів
        res.status(500).json({ error: 'Ошибка сервера' }); // Відправляємо статус помилки 500
      });
  } catch (error) {
    console.error('Error processing request:', error); // Виводимо помилку обробки запиту
    res.status(500).json({ error: 'Ошибка обработки запроса' }); // Відправляємо статус помилки 500
  }
});

// Обробник для всіх інших запитів
router.use((req, res, next) => {
  console.log(`404 Not Found: ${req.originalUrl}`); // Виводимо повідомлення про 404 помилку
  res.status(404).send('Not Found'); // Відправляємо статус помилки 404
});

```

Рисунок А.13 – Функція очікування відповіді від БД та вивід результатів відповіді

```
// Імпортуємо необхідні модулі та компоненти
import './css/App.css'; // Імпортуємо CSS стилі для додатку
import React from 'react'; // Імпортуємо бібліотеку React
import Homepage from './Homepage.js'; // Імпортуємо компонент Homepage
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom'; // Імпортуємо компоненти маршрутизації
import Notebook from './Categories/Notebook.js'; // Імпортуємо компонент Notebook

```

Рисунок А.14 – Імпорт модулів та компонентів на сторінці ComputerNotebook

```
    {/* Головний контейнер */}
     <div className="container_main">
        {/* Контейнер каталогу */}
       <div className="container_catalog">
         {/* Перший список категорій */}
         <ul className="catalog_Gamer">
          {/* Елемент списку - категорія */}
          <li>
            {/* Контейнер з назвою та зображенням категорії */}
           <div className="title_logo">
            {/* Посилання на сторінку категорії */}
            <a className="title_picture" href="/Notebook">
              <img className="logo_catalog" src="images/categories/pc/logo_1.png" alt="" />
            </a>
            {/* Текстова назва категорії */}
            <a href="/Notebook" className="menu_Catalog">Ноутбуки</a>
           </div>
          </li>
```

Рисунок А.15 – Код каталог товарів з назвою та логотипом

```jsx
return (
    <div className="main_site">
      {/* Верхня частина сайту */}
      <div className="container_logo_top">
        {/* Логотип */}
        <img className="logo" src="" alt="" />
      </div>
      {/* Верхня панель з кнопкою меню, логотипом і полем пошуку */}
      <div className="container_top">
        {/* Кнопка "бургер" для відкриття меню */}
        <div>
          <button className="burger">
            <span></span>
            <span></span>
            <span></span>
          </button>
        </div>
        {/* Логотип сайту */}
        <div className="wrap">
          <a href="#">
            <img className="logo_sites" src="images/logo_sites.png" alt="" />
          </a>
        </div>
        {/* Поле пошуку з результатами */}
        <div className="search_input">
          <form className="form_search search_input" onSubmit={(e) => e.preventDefault()}>
            <div className="search_form_inner">
              <input
                autoComplete="off"
                className="search_pole"
                type="text"
                id="search"
                placeholder="Поиск..."
                value={searchTerm}
                onChange={(e) => setSearchTerm(e.target.value)}
              />
              {/* Відображення результатів пошуку */}
              <SearchResults results={searchResults} />
              {/* Кнопка пошуку */}
              <button id="" className="submit" type="submit" name="Найти">
                Знайти
              </button>
            </div>
          </form>
        </div>
        {/* Блок з іконками користувача, кошика */}
        <div className="block_header">
          <a className="header" href="">
            <img className="icon_avatar_cleave" src="images/avatar.png" alt="" />
          </a>
          <a className="header" href="#">
            <img className="icon_avatar_cleave" src="images/cleave.png" alt="" />
          </a>
        </div>
      </div>
```

Рисунок А.16 – Верхня частина сторінки з полем пошуку, та іконками користувача, кошика

```
import React, { useState, useEffect } from 'react'; // Імпортуємо необхідні бібліотеки та компоненти React
import '../css/App.css'; // Імпортуємо стилі для додатку
import '../css/catalog.css'; // Імпортуємо стилі для каталогу
import Homepage from '../Homepage.js'; // Імпортуємо компонент Homepage
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom'; // Імпортуємо компоненти маршрутизації
import ComputerNotebook from '../ComputerNotebook'; // Імпортуємо компонент ComputerNotebook
import Slider from 'rc-slider'; // Імпортуємо компонент Slider
import 'rc-slider/assets/index.css'; // Імпортуємо стилі для Slider
import Rating from './Rating'; // Імпортуємо компонент Rating
import Cart from './Cart'; // Імпортуємо компонент Cart
import { useNavigate } from 'react-router-dom'; // Імпортуємо хук для навігації
```

Рисунок А.17 – Імпорт модулів та компонентів на сторінці Notebook

```
// Визначаємо компонент Notebook
function Notebook() {
  // Хук для навігації
  const navigate = useNavigate();
  // Стан для збереження списку продуктів
  const [products, setProducts] = useState([]);
  // Стани для фільтрації за ціною
  const [minPrice, setMinPrice] = useState(0);
  const [maxPrice, setMaxPrice] = useState(0);
  const [sliderValue, setSliderValue] = useState([0, 0]);
  const [isFiltered, setIsFiltered] = useState(false);
  // Стани для фільтрації за брендом
  const [brands, setBrands] = useState([]);
  const [selectedBrands, setSelectedBrands] = useState([]);
  // Стан для збереження початкового списку продуктів
  const [initialProducts, setInitialProducts] = useState([]);
  // Стани для відображення застосованих фільтрів
  const [filterApplied, setFilterApplied] = useState(false);
  const [priceFilterApplied, setPriceFilterApplied] = useState(false);
  // Стан для вибраних елементів (для корзини)
  const [selectedItems, setSelectedItems] = useState([]);

```

Рисунок А.18 – Усі змінні, які знадобляться для функціоналу на сторінці Notebook

```
// Функції для обробки подій миші (виведення та приховування зображення товару)
const handleMouseEnter = (productId) => {
  // Обробник події наведення миші
  const logoTovar = document.getElementById(`logo_tovar_${productId}`);
  if (logoTovar) {
    logoTovar.classList.add('hidden');
  }
};

const handleMouseLeave = (productId) => {
  // Обробник події відведення миші
  const logoTovar = document.getElementById(`logo_tovar_${productId}`);
  if (logoTovar) {
    logoTovar.classList.remove('hidden');
  }
};
```

Рисунок А.19 – Функціонал який змінює логотип товару на інший логотип при наводці курсору на товар

```
// Ефект для завантаження списку продуктів та брендів при завантаженні компонента
useEffect(() => {
  fetchProducts();
  fetchBrands();
}, []);

// Ефект для фільтрації продуктів при зміні станів isFiltered, sliderValue, priceFilterApplied, selectedBrands
useEffect(() => {
  if (priceFilterApplied || isFiltered) {
    filterProducts();
    setFilterApplied(true);
    setPriceFilterApplied(false);
  }
}, [isFiltered, sliderValue, priceFilterApplied, selectedBrands]);

// Функція для зміни значення Slider
const handleSliderChange = (value) => {
  setSliderValue(value);
  if (priceFilterApplied) {
    setPriceFilterApplied(false);
  } else {
    setIsFiltered(false);
  }
};

// Функція для обробки зміни значень в інпутах
const handleInputChange = (e, index) => {
  const newValue = parseInt(e.target.value, 10) || 0;
  const newSliderValue = [...sliderValue];
  newSliderValue[index] = newValue;
  setSliderValue(newSliderValue);
  setIsFiltered(true);
};

// Функція для застосування фільтрації за ціною
const handleFilterButtonClick = () => {
  setIsFiltered(true);
  setPriceFilterApplied(true);
};
```

Рисунок А.20 – Функція завантаження компонентів фільтрації та обробка фільтрації за ціною

```
// Функція для обробки зміни брендів
const handleBrandChange = (brand) => {
  if (selectedBrands.includes(brand)) {
    setSelectedBrands(selectedBrands.filter(item => item !== brand));
  } else {
    setSelectedBrands([...selectedBrands, brand]);
  }
  setIsFiltered(true);
};
```

Рисунок А.21 – Функція обробки фільтрації за брендом товару

```
102  // Функція для завантаження продуктів з сервера
103  const fetchProducts = () => {
104    fetch(`http://localhost:3024/api/products`)
105      .then(response => {
106        if (!response.ok) {
107          throw new Error('Network response was not ok');
108        }
109        return response.json();
110      })
111      .then(data => {
112        // Логуємо отримані дані продуктів у консоль
113        console.log(data);
114        // Отримуємо мінімальну та максимальну ціну з отриманих даних
115        const prices = data.map(product => product.price);
116        const newMinPrice = Math.min(...prices);
117        const newMaxPrice = Math.max(...prices);
118
119        setMinPrice(newMinPrice);
120        setMaxPrice(newMaxPrice);
121        setSliderValue([newMinPrice, newMaxPrice]);
122
123        // Додаємо властивість зображення для кожного продукту
124        const productsWithImages = data.map(product => ({
125          ...product,
126          image: `catalog/tovar/notebook/${product.id}/logo_1.png`
127        }));
128
129        setProducts(productsWithImages);
130        setInitialProducts(productsWithImages);
131      })
132      .catch(error => console.error('Error fetching products:', error));
133  };
134
135
```

Рисунок А.22 – Функція завантаження з БД мінімальної та максимальної ціни в таблиці Products

```
138  // Функція для завантаження брендів з сервера
139  const fetchBrands = () => {
140    fetch(`http://localhost:3024/api/brands`)
141      .then(response => {
142        if (!response.ok) {
143          throw new Error('Network response was not ok');
144        }
145        return response.json();
146      })
147      .then(data => {
148        setBrands(data);
149      })
150      .catch(error => console.error('Error fetching brands:', error));
151  };
152
```

Рисунок А.23 – Функція завантаження з БД назви брендів в таблиці Products

```
155   // Функція для фільтрації продуктів за ціною
156   const filterProductsByPrice = (productsToFilter, min, max) => {
157     return productsToFilter.filter(product =>
158       product.price >= min && product.price <= max
159     );
160   };
161
162   // Функція для фільтрації продуктів за брендом
163   const filterProductsByBrand = (productsToFilter) => {
164     if (selectedBrands.length === 0) {
165       return productsToFilter;
166     }
167     return productsToFilter.filter(product =>
168       selectedBrands.includes(product.brand)
169     );
170   };
171
```

Рисунок А.24 – Функції фільтрації які викликаються користувачем на сайті, й фільтрують за брендом та за ціною

```
173   // Функція для фільтрації продуктів
174   const filterProducts = () => {
175     let filteredProducts = initialProducts;
176
177     if (isFiltered) {
178       filteredProducts = filterProductsByPrice(filteredProducts, sliderValue[0], sliderValue[1]);
179       filteredProducts = filterProductsByBrand(filteredProducts);
180     }
181
182     setProducts(filteredProducts);
183   };
```

Рисунок А.25 – Функція яка виводить на сайті відфільтрований товар

```
185   // Функція для додавання продукту до корзини
186   const handleAddToCart = (product) => {
187     const updatedItems = [...selectedItems, { ...product, image: `catalog/tovar/notebook/${product.id}/logo_1.png` }];
188     setSelectedItems(updatedItems);
189     localStorage.setItem('cartItems', JSON.stringify(updatedItems));
190   };
```

Рисунок А.26 – Функція яка додає обраний товар в корзину

```
192   // Функція для обробки кліку на кнопці "Корзина"
193   const handleCartClick = () => {
194     navigate('/cart', { state: { cartItems: selectedItems } });
195   };
196
```

Рисунок А.27 – Функція яка переміщує користувача на сторінку кошика

```jsx
{/* Блок з іконками користувача та корзини */}
<div className="block_header">
  <a className="header" href="">
    <img className="icon_avatar_cleave" src="images/avatar.png" alt="" />
  </a>
  <a className="header" onClick={handleCartClick}>
    <img className="icon_avatar_cleave" src="images/cleave.png" alt="Корзина" />
  </a>
</div>
</div>
```

Рисунок А.28 – Блок де можна перейти в кошик

```jsx
{/* Навігаційне меню */}
<nav aria-label="Breadcrumb" className="breadcrumb">
  <ul>
    <li><a href="/Homepage">Головна сторінка</a></li>
    <li><a href="/computer-notebook">Комп'ютери ноутбуки</a></li>
    <li><a href="/notebook">Ноутбуки</a></li>
  </ul>
</nav>
```

Рисунок А.29 – Навігаційне меню сторінки де можна повернутися до попередніх сторінок ієрархії

```jsx
{/* Основний блок з фільтрами та каталогом товарів */}
<div className='Main_Block'>
  {/* Блок з фільтрами */}
  <div className='filter'>
    {/* Фільтр за ціною */}
    <div className='filter_price'>
      <div className="slider-container">
        <p className=''>Ціна ₴</p>
        {/* Slider для вибору ціни */}
        <Slider
          className="custom-slider"
          min={minPrice}
          max={maxPrice}
          range={true}
          value={sliderValue}
          onChange={handleSliderChange}
          tipFormatter={(value) => `${value} ₴`}
        />
      </div>
      {/* Введення значень мінімальної та максимальної ціни */}
      <div className='input_filter'>
        <input
          className='Input_price'
          type="text"
          value={sliderValue[0]}
          onChange={(e) => handleInputChange(e, 0)}
        />
        <span className='input_price_divider'> - </span>
        <input
          className='Input_price'
          type="text"
          value={sliderValue[1]}
          onChange={(e) => handleInputChange(e, 1)}
        />
        {/* Кнопка для застосування фільтру за ціною */}
        <button className='button_filter_price' onClick={handleFilterButtonClick} >
          OK
        </button>
      </div>
    </div>
```

Рисунок А.30 – Блок де можна буде фільтрувати товари за ціновим діапазоном

```jsx
{/* Фільтр за брендами */}
<div className='filter_brand'>
  {/* Виводимо чекбокси для вибору брендів */}
  {brands.map(brand => (
    <div className='checkbox-wrapper-19' key={brand}>
      <input
        type="checkbox"
        id={brand}
        checked={selectedBrands.includes(brand)}
        onChange={() => handleBrandChange(brand)}
      />
      <label className='check-box' htmlFor={brand}>
        <span className='check-box-text'>{brand}</span>
      </label>
    </div>
  ))}
</div>
</div>
```

Рисунок А.31 – Блок де можна буде фільтрувати товари за обраними брендами

```jsx
{/* Блок з каталогом товарів */}
<div className="catalog_main">
  <div className="tovar">
    <ul className="check_list">
      {/* Виводимо список продуктів */}
      {products.map(product => (
        <li key={product.id}>
          <div className='block_source_tovar'>
            <a className="source_tovar" href="#">
              {/* Зображення товару */}
              <img
                className="logo_catalog logo_tovar"
                src={`catalog/tovar/notebook/${product.id}/logo_1.png`}
                alt={product.name}
                id={`logo_tovar_${product.id}`}
              />
              {/* Зображення товару при наведенні */}
              <img
                className="logo_catalog logo_tovar_hover"
                src={`catalog/tovar/notebook/${product.id}/logo_smena_1.png`}
                alt={product.name}
                onMouseEnter={() => handleMouseEnter(product.id)}
                onMouseLeave={() => handleMouseLeave(product.id)}
              />
            </a>
          </div>
          {/* Назва товару */}
          <div className='block_text_tovar'>
            <a href="#" className="text_tovar">{product.name}</a>
          </div>
          {/* Ціна товару */}
          <div className="text_money">
            <span>{product.price}₴</span>
          </div>
          {/* Рейтинг товару */}
          <p><Rating value={product.rating} /></p>
          {/* Кнопка "Купити" */}
          <button
            className="btn btn-2 btn-sep icon-cart"
            onClick={() => handleAddToCart(product)}
          >
            <span>Купити</span>
          </button>
        </li>
      ))}
    </ul>
  </div>
</div>
</div>
</div>
);
}
```

Рисунок А.32 – Блок виводу товару з урахуванням якщо використовуються фільтри

```jsx
import React from 'react';
import '../css/catalog.css';

// Компонент рейтингу, що відображається за допомогою зірок
const Rating = ({ value }) => {
  const stars = [];

  // Цикл для створення зіркового рейтингу
  for (let i = 0; i < 5; i++) {
    if (i < value) {
      stars.push(<span key={i}>&#9733;</span>); // Зірка
    } else {
      stars.push(<span key={i}>&#9734;</span>); // Пуста зірка
    }
  }

  // Повертаємо JSX зі списком зірок
  return <div className='Rating_star'>{stars}</div>;
};

export default Rating;
```

Рисунок А.33 – Код який робить та виводить рейтинг для товарів на сторінці товарів

```jsx
// Компонент корзини
function Cart() {
  // Стан для збереження елементів у корзині, отриманих з локального сховища або пустого масиву
  const [cartItems, setCartItems] = useState(
    JSON.parse(localStorage.getItem('cartItems')) || []
  );

```

Рисунок А.34 – Компоненти станів для кошика

```jsx
  // Ефект для збереження елементів корзини у локальному сховищі при зміні стану
  useEffect(() => {
    localStorage.setItem('cartItems', JSON.stringify(cartItems));
  }, [cartItems]);

```

Рисунок А.35 – Ефект для збереження товарів у локальному сховищі

```jsx
  // Функція для зміни кількості товару в корзині
  const handleQuantityChange = (index, newQuantity) => {
    const updatedCartItems = [...cartItems];
    updatedCartItems[index].quantity = newQuantity;
    setCartItems(updatedCartItems);
  };

```

Рисунок А.36 – Функція зміни кількості товарів

```
24  // Функція для видалення товару з корзини
25  const handleRemoveItem = (index) => {
26    const updatedCartItems = cartItems.filter((_, i) => i !== index);
27    setCartItems(updatedCartItems);
28  };
29
```

Рисунок А.37 – Функція видалення товарів

```
30  // Розрахунок загальної ціни всіх товарів у корзині
31  const totalPrice = cartItems.reduce(
32    (total, item) => total + (parseFloat(item.price) * item.quantity || 0),
33    0
34  );
35
```

Рисунок А38 – Функція загальної ціни

```
96        {/* Вміст корзини */}
97        <div className='container_main_cart'>
98          {/* Заголовок корзини */}
99          <h1>Корзина</h1>
100         {/* Список товарів у корзині */}
101         <ul className='Carts_Tovar'>
102           {/* Мапінг елементів корзини */}
103           {cartItems.map((item, index) => (
104             <li key={index}>
105               <div className='container_cart'>
106                 {/* Зображення товару */}
107                 <div className='block_tovar_carts_img'>
108                   <a className='Cart_Img_src'>
109                     <img className='Cart_img' src={item.image} alt={item.name} />
110                   </a>
111                 </div>
112                 {/* Назва товару */}
113                 <div className='block_tovar_carts_name'>
114                   <div className='text_tovar_block'>
115                     <h3 className='cart_name'>{item.name}</h3>
116                   </div>
117                 </div>
```

Рисунок А.39 – Перша частина кошику з позиціонуванням та логотипом

```
{/* Поле для зміни кількості товару */}
<div className='block_tovar_field'>
  <div className='text_tovar_block'>
    <div className="quantity-field">
      <span
        className="quantity-btn"
        onClick={() =>
          handleQuantityChange(
            index,
            Math.max(parseInt(item.quantity, 10) - 1 || 1, 1)
          )
        }
      >
        -
      </span>
      <input
        type="text"
        id={`quantity-${index}`}
        className="quantity-input"
        value={item.quantity || 1}
        disabled
      />
      <span
        className="quantity-btn"
        onClick={() =>
          handleQuantityChange(
            index,
            Math.max(parseInt(item.quantity, 10) + 1 || 1, 1)
          )
        }
      >
        +
      </span>
    </div>
  </div>
</div>
```

Рисунок А.40 – Друга частина кошика де можна змінити кількість товару

```
{/* Ціна товару і кнопка для видалення */}
<div className='block_tovar_carts_sallary'>
  <div className='price_tovar_block'>
    <p className='tovar_sallary'>Ціна: {item.price}₴</p>
    <div className='tovar_del'>
      <img className='trash_sallary' src='../catalog/notebook/trash.png'></img>
      <span onClick={() => handleRemoveItem(index)}>Видалити товар</span>
    </div>
  </div>
</div>
</div>
</li>
))}
</ul>
{/* Відображення загальної ціни та кнопка для оформлення покупки */}
<div className='container-price'>
  <div className="total-price">
    <p className='total_text_price'>Загальна ціна: {totalPrice}₴</p>
    <button className='buy_price' onClick={() => console.log('Оформити замовлення')}>Оформити замовлення</button>
  </div>
</div>
</div>
</div>
);
}

export default Cart;
```

Рисунок А.41 – Третя частина кошика де можна побачити ціну, загальну ціну, видалити товар, та оформити замовлення

```
const express = require('express');
const mysql = require('mysql');

// Створення об'єкта для маршрутизації
const router = express.Router();

// Підключення до бази даних MySQL
const db = mysql.createConnection({
    host: 'localhost',
    user: 'root',
    password: 'password',
    database: 'shop'
});

// Підключення до бази даних та виведення повідомлення про успішне підключення або помилку
db.connect((err) => {
    if (err) {
        console.error('Помилка підключення до бази даних:', err);
        return;
    }
    console.log('Підключення до бази даних успішне');
});

// Використання middleware для обробки запитів з JSON-даними
router.use(express.json());
```

Рисунок А.42 – Робимо підключення до БД та перевіряємо чи пройшло успішне підключення

```
// API для сторінки Notebook
// Маршрут для отримання всіх продуктів
router.get('/products', (req, res) => {
    db.query('SELECT * FROM Products', (err, results) => {
        if (err) {
            console.error('Помилка запиту до бази даних:', err);
            res.status(500).json({ error: 'Помилка сервера' });
            return;
        }
        res.json(results);
    });
});
```

Рисунок А.43 – Маршрут для отримання товарів

```
// Маршрут для фільтрації продуктів за ціною
router.get('/products/filter', (req, res) => {
    const minPrice = parseFloat(req.query.minPrice);
    const maxPrice = parseFloat(req.query.maxPrice);
    db.query('SELECT * FROM Products WHERE price >= ? AND price <= ?', [minPrice, maxPrice], (err, results) => {
        if (err) {
            console.error('Помилка запиту до бази даних:', err);
            res.status(500).json({ error: 'Помилка сервера' });
            return;
        }
        res.json(results);
    });
});
```

Рисунок А.44 – Фільтрація товарів за ціною

```
54  // Маршрут для отримання унікальних брендів
55  router.get('/brands', (req, res) => {
56      db.query('SELECT DISTINCT brand FROM Products', (err, results) => {
57          if (err) {
58              console.error('Помилка запиту до бази даних:', err);
59              res.status(500).json({ error: 'Помилка сервера' });
60              return;
61          }
62          const brands = results.map(result => result.brand);
63          res.json(brands);
64      });
65  });
66
```

Рисунок А.45 – Фільтрація товарів за брендом