Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедр Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *Уваровського Максима Дмитровича*
(ПІБ)

академічної групи *121-20-1*
(шифр)

спеціальності *121 Інженерія програмного забезпечення*
(код і назва спеціальності)

освітньої програми *Інженерія програмного забезпечення*
(назва освітньої програми)

на тему: Розробка програмного забезпечення веб застосунку інтернет магазину іграшок мовою JS

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**Інституційною**	
кваліфікаційної роботи	*проф. Лактіонов І.С.*			
розділів:				
спеціальний	*проф. Лактіонов І.С.*			
економічний	*доц. Касьяненко Л.В.*			
Рецензент				
Нормоконтролер	*доц. Мартиненко А.А.*			

Дніпро
2024

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

_____ М.О. Алексєєв
(підпис) (прізвище, ініціали)

« » _____ 2024 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *121-20-1* *Уваровського Максима Дмитровича*
(група) (прізвище та ініціали)

тема кваліфікаційної роботи Розробка програмного забезпечення веб застосунку інтернет магазину іграшок мовою JS

затверджена наказом ректора НТУ «ДП» від *23.05.2024 р.* № *469-с*

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів проєктно-технологічної та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі. Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***10.06.2024 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки***	***14.06.2024 р.***

Завдання видав _____ *проф. Лактіонов І.С.*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Уваровський М.Д.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2024 р.*

Термін подання кваліфікаційної роботи до ЕК: *17.06.2024 р.*

Реферат

Пояснювальна записка: 101 с., рис. 43, 3 дод., 21 джерел.

Об'єкт розробки: програмне забезпечення веб застосунку інтернет магазину іграшок мовою JS

Мета кваліфікаційної роботи: розробка програмного забезпечення веб застосунку інтернет магазину іграшок мовою JS, що дозволить клієнтам зручно купувати товари, а адміністраторам налаштовувати контент сайту.

У вступній частині кваліфікаційної роботи описується зв'язок проблеми, що вирішується, з об'єктом діяльності фахівця напряму; встановлюється мета роботи та галузь застосування програмного забезпечення, що розробляється; обґрунтовується актуальність теми; визначаються конкретні цілі та завдання, які потрібно виконати під час написання кваліфікаційної роботи.

У першому розділі кваліфікаційної роботи розкриваються загальні відомості з предметної області, описується призначення розробки та галузь застосування, висвітлюються підстави для розробки програмного забезпечення, зазначаються вимоги до функціональних характеристик, інформаційної безпеки інформаційної та програмної сумісності, відбувається постановка завдання.

У другому розділі кваліфікаційної роботи зазначається функціональне призначення програми; описується використана архітектура та шаблони проектування; висвітлюється перелік та опис використаних технологій та мов програмування; описується структура програми, алгоритми її функціонування, база даних; описується організація вхідних та вихідних даних програми; відбувається опис розробленого програмного продукту.

У третьому, економічному, розділі вказуються розрахунки наступних значень: трудомісткість розробленого продукту та вартість його розробки; витрати та приблизний час на створення програмного забезпечення.

Практичне значення полягає у розробці веб застосунку інтернет магазину іграшок, що дозволить клієнтам зручно та у будь-який час, у будь-якому місці, купувати потрібні товари.

Розробка інтернет-магазину іграшок є актуальною з кількох важливих причин. По-перше, електронна комерція продовжує стрімко зростати, оскільки сучасні споживачі надають перевагу онлайн-покупкам завдяки їх зручності та швидкості. Це дозволяє купувати товари будь-де та будь-коли, що є надзвичайно важливим в умовах постійної зайнятості сучасних сімей. По-друге, зміна поведінки споживачів, зокрема батьків, які все частіше користуються онлайн-ресурсами для придбання товарів для своїх дітей, зумовлена не лише зручністю, але й можливістю порівняти ціни та характеристики товарів.

Ключові слова: ЕЛЕКТРОННА КОМЕРЦІЯ, БАЗА ДАНИХ, REACT, JAVASCRIPT, NODE.JS, AXIOS API, EXPRESS.JS, MONGODB, MONGOOSE, HTML, CSS, NOSQL.

Abstract

Explanatory note: 101 p., figures 43, 3 add., 21 sources.

Object of development: web application software of an online toy store in JS language

The purpose of the qualification work: development of software for a web application of an online toy store in the JS language, which will allow customers to conveniently buy goods, and administrators to configure the content of the site.

In the introductory part of the qualification work, the relationship between the problem being solved and the object of the specialist's activity is described; the purpose of the work and the field of application of the software being developed are established; the relevance of the topic is substantiated; specific goals and tasks to be accomplished during the writing of the qualification paper are defined.

In the first section of the qualification work, general information on the subject area is revealed, the purpose of the development and the field of application are described, the grounds for software development are highlighted, the requirements for functional characteristics, information security, information and software compatibility are specified, and the task is set.

The functional purpose of the program is noted in the second section of the qualification work; the used architecture and design patterns are described; the list and description of the used technologies and programming languages is highlighted; the structure of the program, the algorithms of its operation, the database are described; the organization of input and output data of the program is described; there is a description of the developed software product.

In the third, economic, section, calculations of the following values are specified: the labor intensity of the developed product and the cost of its development; costs and estimated time to create the software.

The practical significance lies in the development of a web application of an online toy store, which will allow customers to conveniently and at any time, in any place, buy the desired goods.

The development of an online toy store is relevant for several important reasons. First, e-commerce continues to grow rapidly as today's consumers prefer online shopping due to its convenience and speed. This allows you to buy goods anywhere and anytime, which is extremely important in the conditions of constant employment of modern families. Secondly, the change in the behavior of consumers, in particular parents, who increasingly use online resources to purchase goods for their children, is due not only to convenience, but also compare prices and product characteristics.

Keywords: E-COMMERCE, DATABASE, REACT, JAVASCRIPT, NODE.JS, AXIOS API, EXPRESS.JS, MONGODB, MONGOOSE, HTML, CSS, NOSQL.

Зміст

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – Application Programming Interface

HTML - Hyper Text Markup Language

CSS - Cascading Style Sheets

NoSQL – Not only Structured Query Language

SQL –Structured Query Language

JS – JavaScript

ПК - Персональний комп’ютер

ПЗ - Програмне забезпечення

ІС - Інформаційна система

ОС - Операційна система

БД - База даних

СУБД - Система управління базами даних

ВСТУП

Проблема, яку вирішує розробка інтернет-магазину іграшок, має прямий зв’язок з об’єктом діяльності фахівця в сфері інформаційних технологій та електронної комерції. Сучасний фахівець з програмної інженерії зобов'язаний володіти навичками розробки веб-застосунків, що забезпечують зручний і безпечний користувацький досвід. Інтернет-магазин іграшок є яскравим прикладом такого застосунку, де поєднуються технології, спрямовані на оптимізацію торгівельних процесів, покращення взаємодії з клієнтами та забезпечення безперебійної роботи системи.

Метою цієї кваліфікаційної роботи є розробка програмного забезпечення веб-застосунку інтернет-магазину іграшок на основі технологій React.js та Express.js. Такий підхід дозволяє створити інтерактивний, ефективний та масштабований інтерфейс користувача, який відповідатиме сучасним стандартам веб-розробки. Галузь застосування розробленого інтернет-магазину охоплює електронну комерцію, де він зможе обслуговувати широку аудиторію, забезпечуючи зручний доступ до асортименту іграшок, подушок, світильників та інших дитячих товарів.

Об'єктом розробки є веб-додаток інтернет-магазину іграшок мовою JS

Актуальність теми зумовлена стрімким розвитком електронної комерції, яка стає невід'ємною частиною сучасного життя. Зростання популярності онлайн-покупок обумовлює необхідність створення надійних, зручних та функціональних інтернет-магазинів. Споживачі все частіше обирають онлайн-шопінг завдяки його зручності, економії часу, можливості порівнювати ціни та читати відгуки інших покупців. Розробка інтернет-магазину іграшок дозволить задовольнити потреби сучасних сімей, які шукають якісні дитячі товари, не виходячи з дому.

Конкретизація постановки завдання цієї кваліфікаційної роботи включає кілька ключових складових. По-перше, необхідно розробити зручний і інтуїтивно зрозумілий інтерфейс користувача за допомогою бібліотеки React.js,

яка забезпечує високу продуктивність та можливість повторного використання компонентів. По-друге, інтеграція серверної частини з використанням фреймворку Express.js дозволить створити надійну і масштабовану серверну інфраструктуру. По-третє, важливо створити ефективну базу даних для зберігання інформації про товари, клієнтів та замовлення, що забезпечить безперебійну роботу магазину і задоволення потреб користувачів.

Отже, розробка інтернет-магазину іграшок є актуальною та необхідною в сучасних умовах розвитку електронної комерції. Вона сприятиме поліпшенню користувацького досвіду, підвищенню конкурентоспроможності бізнесу та задоволенню потреб сучасних споживачів. Ця кваліфікаційна робота спрямована на досягнення поставлених цілей і вирішення актуальних завдань у сфері розробки веб-застосунків.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Загальні відомості з предметної галузі

Торгівля як процес обміну товарно-матеріальними цінностями відома ще з кам'яного віку. Люди обмінювалися різними предметами та ресурсами, задовольняючи свої потреби. З розвитком цивілізації та технологій, форми і методи торгівлі постійно еволюціонували. З моменту, як інтернет став загальнодоступним, він увійшов в усі галузі життя людини, у тому числі і покупка товарів. Зараз в інтернеті можна купити будь-що, від продуктів із супермаркету до автомобіля. Тому розробка інтернет магазинів буде завжди актуальною в IT-сфері.

Інтернет-магазин - це web-сайт, за допомогою якого можна торгувати будь-яким товаром. Користувач може зробити замовлення онлайн на сайті в браузері або у мобільному додатку. Інтернет-магазини набули величезної популярності через зручність і доступність, вони стали важливим елементом сучасної торгівлі.

Інтернет-магазин має ряд переваг серед звичайних магазинів:

− Можливість покупки не виходячи з дому.

− Великий асортимент.

− Легке порівняння цін в різних магазинах.

− Можливість читання відгуків інших покупців.

− Легкий доступ до інформації про товар.

− Економія часу.

− Безконтактні покупки.

− Міжнародний шопінг.

− Цілодобовий доступ до магазину, без прив'язки до графіку роботи.

− Можливість отримати персоналізовані рекомендації на основі попередніх покупок та вподобань.

Серед недоліків покупки в інтернеті можна виділити:

− Неможливість побачити товар наживо.

− Витрати коштів на доставку.

− Час очікування замовлення.

− Проблеми з поверненням товару.

− Неможливість живого особистого обслуговування.

− Невідповідність опису товару.

− Ризик потрапляння на шахрайські сайти.

− Можливість отримання пошкодженого товару під час транспортування.

− Необхідність введення особистих даних та ризик їх несанкціонованого використання.

З огляду на вищевказані переваги та недоліки, можна зробити висновок, що інтернет-торгівля є незамінним інструментом сучасної економіки. Розвиток технологій, таких як штучний інтелект, великі дані та блокчейн, сприяє подальшій еволюції інтернет-магазинів, роблячи їх ще більш зручними та безпечними для користувачів.

1.1.2. Аналіз готових існуючих рішень

Для аналізу був обраний інснуючий інтернт-магазин − umka.dp.ua − дніпровський інтернет-магазин іграшок.

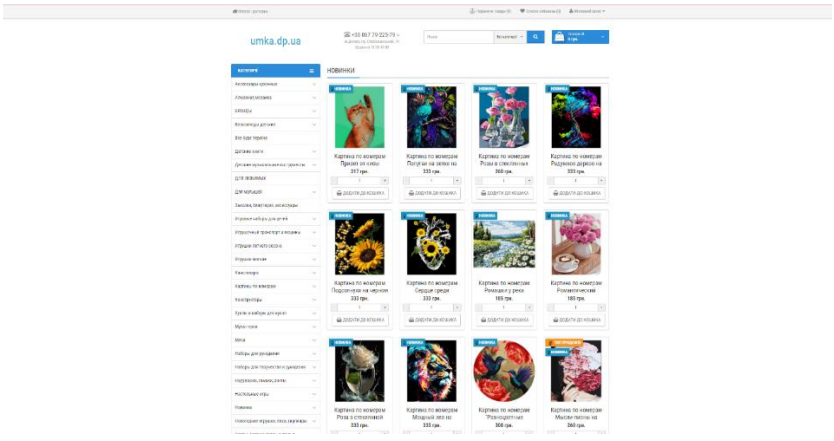

Рис. 1.1. Головна сторінка

На головній сторінці (рис. 1.1) зверху розміщено блок з елементами "Порівняти товари", "Список побажань", "Обліковий запис", нижче лого, номер, пошук, корзина; зліва розміщені категорії. В основній частині сайту розміщені товари-новинки.

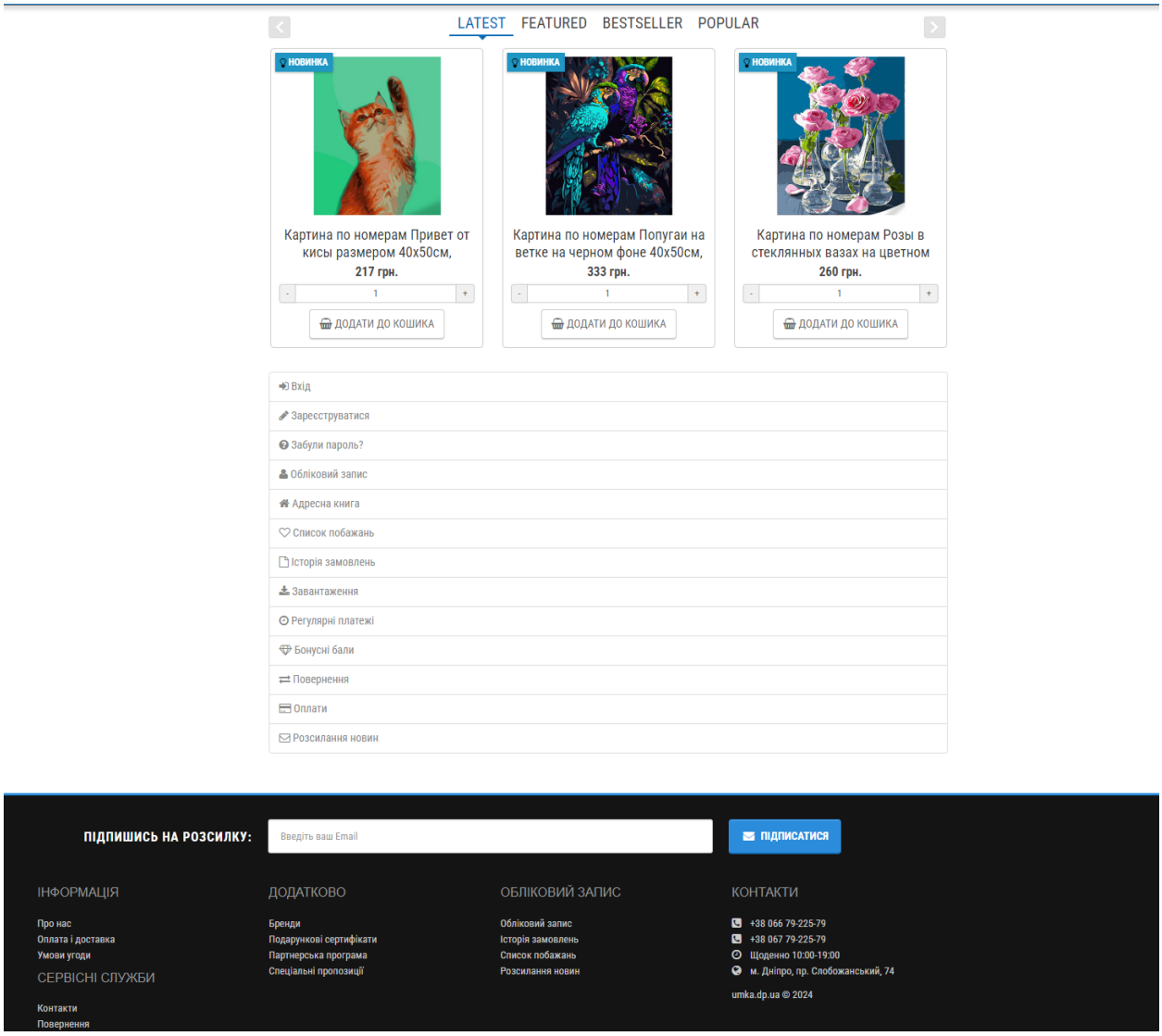

Рис. 1.2. Нижня частина та «футер» сайту

В нижній частині розміщені карусель з категоріями "LATEST", "FEATURED", "BESTSELLER", "POPULAR".

Нижче меню для користувачів з елементами "вхід", "реєстрація", "забули пароль?" та ін. Також є кнопка про підписку на розсилку та стандартний футер з додатковою інформацією та контактами.

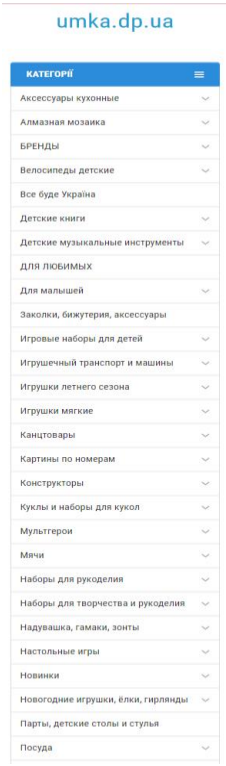

Рис 1.3. Бокове меню

Розглянемо бокове меню (рис 1.3) з категоріями товарів. Категорій багато, вони не дуже логічні, чомусь на російській мові (хоча сайт на українській). З телефонної версії (на iPhone 11 pro Max) меню погано адаптовано - підкатегорії не вміщаються на екран, потрібно скролити вправо, що видно на рис. 1.4.

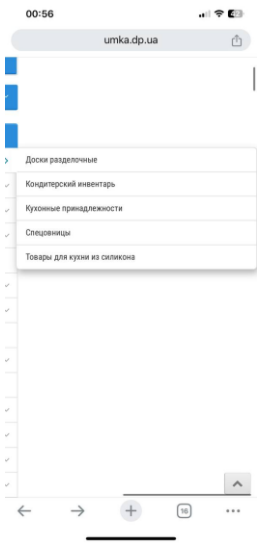

Рис. 1.4. Телефонна версія бокового меню

Тепер перейдемо до розгляду певного товару.

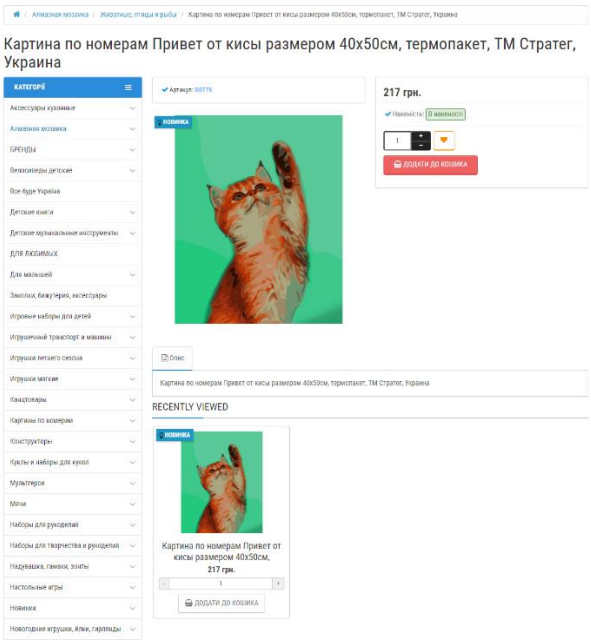

Рис. 1.5. Сторінка товару

Зверху є назва товару, потім артикул, фото, справа ціна, статус наявності, кнопка вибору кількості та кнопка "додати до кошика". Нижче розміщений опис товару (рис. 1.5).

У процесі аналізу інтернет-магазину umka.dp.ua було виявлено, що сайт є доволі стандартним, зрозумілим, зручним. Проте я виявив і недоліки - поганий адаптив, нелогічні категорії, невелика кількість інформації про товар, різні мови написання категорій, опису і фільтрів. Також з головної сторінки фото певного товару ніяк не змінюється при взаємодії з ним (наприклад, приближення фото при наведені мишкою). Всі ці недоліки будуть виправлені під час виконання даної кваліфікаційної роботи.

1.2 Призначення розробки та галузь застосування

У межах кваліфікаційної роботи буде розроблено програмне забезпечення веб застосунку інтернет магазину іграшок "mimi_shop©", який спеціалізується

на продажі м'яких іграшок, подушок, світильників та інших товарів. Сайт створюється мовою програмування JavaScrip за допомогою, бібліотек React.js та express.js.

Інтернет-магазин використовується у електронній комерційній діяльності. Причиною розробки сайту є необхідність розміщення товарів для продажу, а також створення зручної комунікації з клієнтами, підключення системи оплати та створення бази даних.

Необхідність розробки веб-застосунку інтернет-магазину "mimi_shop©" обумовлена кількома ключовими факторами:

Необхідність розміщення товарів для продажу: Створення платформи, де клієнти можуть переглядати, вибирати та купувати товари онлайн.

Зручна комунікація з клієнтами: Надання можливості взаємодії з клієнтами через онлайн-чат, електронну пошту та інші засоби комунікації.

Підключення системи оплати: Інтеграція безпечних платіжних систем для обробки транзакцій.

Створення бази даних: Забезпечення ефективного зберігання інформації про товари, клієнтів та замовлення.

Система, що розроблюється, може бути застосована в різних галузях:

Роздрібна торгівля: Онлайн-продаж товарів кінцевим споживачам.

Підприємства малого та середнього бізнесу: Створення зручної та доступної платформи для малих і середніх підприємств для виходу на ринок електронної комерції.

Іграшкова індустрія: Спеціалізація на продажу іграшок та супутніх товарів.

Дизайн інтер'єру та подарункові товари: Пропонування продуктів, які можуть використовуватися для декорування інтер'єру або як подарунки.

Багатоканальна торгівля (Omnichannel commerce): Інтеграція онлайн- та офлайн-каналів продажу для створення безшовного досвіду для клієнтів.

Розробка інтернет-магазину "mimi_shop©" сприятиме не лише зростанню бізнесу в галузі електронної комерції, але й наданню клієнтам зручного та

безпечного способу здійснення покупок. Інтеграція сучасних технологій, таких як React.js та Express.js, дозволить створити функціональний і привабливий веб-застосунок, який відповідатиме сучасним вимогам користувачів.

1.3 Підстава для розробки

Підставами для розробки та виконання кваліфікаційної роботи є:

− освітня програма 121 «Інженерія програмного забезпечення»;

− навчальний план та графік навчального процесу;

− наказ ректора Національного технічного університету «Дніпровська політехніка» № 469-с від 23.05.2024 р;

− завдання на кваліфікаційну роботу на тему «Розробка програмного забезпечення веб застосунку інтернет магазину іграшок мовою JS».

1.4 Постановка завдання

1.4.1 Постановка завдання до розробки системи

В цій кваліфікаційній роботі описується написання сайту інтернет-магазину з продажу іграшок.

Мета роботи: Розробка програмного забезпечення веб застосунку інтернет магазину іграшок мовою JS.

Об’єкт розробки: програмне забезпечення веб застосунку інтернет магазину іграшок мовою JS

Веб застосунок створюється для інтернет торгівлі товарами онлайн, залишення замовлень клієнтами для зв'язку з ними, створення бази даних.

Аудиторією є цільова аудиторія даного інтернет-магазину. Це в основному жіноча аудиторія, з України, віком 14-40 років.

Структура об'єктів інформаційної системи:

− База даних товарів і клієнтів;

− Функціонал пошуку та фільтрації товарів;

− Кошик покупок та оформлення замовлення;

− Адміністративний розділ для управління товарами**;**

Призначення вихідної інформації:

− Інформація про товари, їх опис, характеристики та наявність на складі;

− Інформація про клієнтів, їх замовлення та історію покупок;

− Деталі замовлення та інформація про оплату та доставку

1.4.2 Опис інтерфейсу веб-додатку

Веб застосунок інтернет магазин іграшок "mimi_shop©" складається з двох частин: користувацької та адміністративної. Нижче описано їх інтерфейс.

Користувацька частина:

− Головна сторінка: зверху розміщено поле з інформацією "про нас", "повернення" та ін. Потім лого, поле пошуку по сайту, кнопка "Зателефонуйте нам", натиснувши на яку з мобільного пристрою відбудеться перехід в режим набору номера, корзина. Потім меню категорій, що розміщено горизонтально і займає менше місця. Далі карусель з акційними пропозиціями, поле з особливлостями та перевагами даного магазину. Нижче поля "Хіт продажів", "Новинки", що також є каруселлю. При наведені мишкою на фото якогось товару воно змінюється. Нижче футер та іконки комунікації.

− Меню є випадаючим списком, що активується під час наведені на нього мишкою.

− Сторінка конкретного товару - шапка сайту не змінюється. Зверху є шлях до цього товару, назва товару, потім артикул, фото, справа ціни, кнопка вибору кількості та кнопка "додати до кошика". Нижче розміщений опис товару та характеристики, також інформація про оплату й доставку, та відгуки про товар. Інформація змінюється при кліку мишею по заголовку. Фото товару приближається при наведені на нього мишкою.

Адміністративна частина:

− Головна сторінка: містить форми, на яких здійснюється управління над певними елементами сайту. Кожна з цих форм має поля, у які записується вся необхідна інформація про товар, категорію або фільтр, кнопка для додавання елементу та їх перегляд

Рис. 1.6. Структурна схема інтернет-магазину

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

При розробці інтернет-магазину для зручного користування повинні бути дотримані такі вимоги:

− зручний та простий інтерфейс;

− швидке завантаження сторінок;

− швидкий зручний пошук;

− сортування по категоріям;

− фільтрація товарів;

− детальна інформація про товари;

− швидке оформлення замовлення;

− декілька способів оплати;

− система повернення товарів;

− відповідність реальних характеристик з характеристиками на сайті;

− адміністративна частина для управління товарами (додавання до каталогу, видалення, редагування);

− адміністративна частина для управління категоріями товарів, фільтрами та ін.

1.5.2. Вимоги до інформаційної безпеки

1. Захист платежів та даних

− Transport Layer Security для шифрування особистих даних

− Secure Socket Layer теж саме

− використання безпечних платіжних систем, наприклад PayPal, Stripe, або Authorize.Net

− використання двохфакторної аутентифікації для більш високого рівня безпеки, який запитує підтвердження платежу через телефон, електронну адресу, мобільний додаток, тощо

2. Захист від кібератак

− використання антивірусного ПО для захисту від шпигунів

− системи для виявлення вторгнень

− Content Delivery Network та Web Application Firewall для захисту від DDoS-атак

3. Захист особистих даних

− чітке описання політики конфіденційності для інформування клієнтів щодо використання їх особистих даних

− дотримання законів про захист даних, таким як GDPR (Загальний регламент захисту даних)

− захист збережених даних: використання найбільш безпечних алгоритмів шифрування для збереження даних клієнтів.

4. Ролі доступу та права

5. Навчання персоналу кібербезпеці

− навчання персоналу кібербезпеці для запобігання кібер атак

− навчання персоналу запобігання фішингових атак

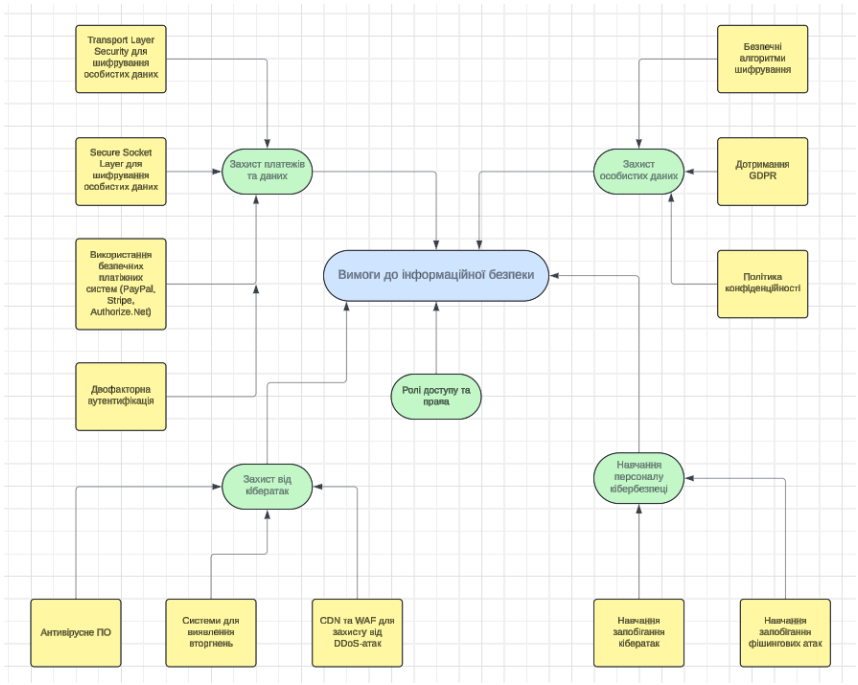

Рис. 1.7. Структурна схема вимог до інформаційної безпеки

1.5.3. Вимоги до складу та параметрів технічних засобів

Перелік складу та параметрів технічних засобів, необхідних для функціонування веб-додатку:

− пристрої вводу: клавіатура, миша;

− пристрої виведення: монітор, екран мобільного телефону та ін.

− стабільний доступ до мережі Інтернет;

− встановлена одна з операційних систем: Windows, Linux, MacOS, Android, IOS;

− будь-який веб-браузер з підтримкою JavaScript(Google Chrome, Microsoft Edge та ін.);

− оперативна пам’ять об’ємом 4ГБ та більше;

1.5.4. Вимоги до інформаційної та програмної сумісності

Для розробки додатку за темою кваліфікаційної роботи мають задовільнятися нижче вказані вимоги:

− доступ до мережі Інтернет;

− встановлена одна з операційних систем: Windows, Linux, MacOS;

− будь-який веб-браузер з підтримкою JavaScript(Google Chrome, Microsoft Edge та ін.);

− оперативна пам’ять об’ємом 4ГБ та більше;

− будь-який редактор коду(для розробки використовується Visual Studio Code);

− 15 ГБ вільного місця на жорсткому диску;

Для створення інтерфейсу інтернет-магазину була обрана мова програмування JavaScript(бібліотека React), для створення серверної частини − Node.js(express.js, mongoose), СУБД − MongoDB.

Висновки за розділом

У першому розділі розглянуто загальні відомості з предметної галузі, включаючи аналіз аналогів, ступінь розв'язання завдань, технічні протиріччя та прогалини знань. Визначено призначення розробки та галузь застосування програми або програмного виробу. Проведено аналіз підстав для розробки, вказавши документи та організації, що їх затвердили. Встановлено постановку завдання, включаючи опис характеристик завдання та умов для його вирішення. Висунуто вимоги до програми або програмного виробу, включаючи функціональні характеристики, інформаційну безпеку, склад та параметри технічних засобів, інформаційну та програмну сумісність. Цей аналіз допоможе зрозуміти контекст розробки та визначити основні вимоги до програмного забезпечення.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ТА РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1. Функціональне призначення системи

Об’єктом розробки проекту даної кваліфікаційної роботи є веб застосунок інтернет магазину, написаний за допомогою мови програмування мовою JS(бібліотеки React), що дозволить клієнтам шукати, переглядати та замовляти товари; адміністраторам – керувати продуктовою базою. Тому функціональне призначення програми має містити:

− фільтрування товарів за категоріями та підкатегоріями для пошуку конкретної групи товарів;

− відображення товарів в магазині;

− пошук товару за його назвою;

− корзину товарів, куди користувач додає бажані для покупки товари. Також має бути можливість видаляти з корзини як поодиноко товари, так і всі разом;

− автентифікація та авторизація клієнтів;

− можливість виходу з акаунту;

− особистий кабінет для авторизованих користувачів, де вони зможуть передевитись список своїх замовлень;

− зворотній зв’язок через чат не виходячи з сайту, який під’єднується за допомогою API відповідного ресурсу;

Експлуатаційне призначення програми має містити:

− зручний та інтуїтивно зрозумілий користувацький інтерфейс, завдяки якому купувати товари зможуть клієнти будь-якого віку.

− вбудована можливість керування продуктовою базою та фільтрами для адміністратора;

− скорочення витрат на аренду приміщення та зарплати шляхом автоматизації обробки замовлень;

2.2. Опис застосованих математичних методів

Під час проектування та розробки веб-додатка не застосовувалися складні математичні методи чи алгоритми. Всі розрахунки та обчислення обмежувалися простими арифметичними операціями.

2.3 Опис використаної архітектури та шаблонів проектування

Архітектура програмного забезпечення даного веб-застосунку базується на клієнт-серверній моделі. Ця модель розподіляє функціональність між сервером, який забезпечує зберігання даних та бізнес-логіку, та клієнтом, який відповідає за інтерфейс користувача та взаємодію з користувачем. Серверна частина веб-застосунку реалізована з використанням фреймворку Express.js, що дозволяє ефективно управляти запитами та відповідями, а також інтегрувати різноманітні middleware[1] для обробки даних. Клієнтська частина розроблена на основі бібліотеки React, що надає можливості створення динамічних і реактивних інтерфейсів користувача.

Вибір клієнт-серверної архітектури зумовлений кількома ключовими факторами. По-перше, така архітектура дозволяє розділити обов'язки між клієнтом і сервером, що полегшує підтримку та масштабування застосунку. По-друге, використання React для клієнтської частини забезпечує високу продуктивність та зручність у розробці складних інтерфейсів завдяки компонентному підходу. Express.js, у свою чергу, забезпечує надійність та гнучкість серверної частини, спрощуючи інтеграцію з базами даних та іншими зовнішніми сервісами.

Щодо шаблонів проектування, в даному проекті використовується патерн Model-View-Controller (MVC)[2]. Цей патерн сприяє розділенню логіки додатку на три основні компоненти: Model (модель), що відповідає за управління даними та бізнес-логіку; View (подання), що відображає дані користувачу; та Controller (контролер), що обробляє запити користувачів і взаємодіє з моделлю для

оновлення подання. Використання MVC уможливлює легше тестування та модифікацію окремих частин додатку, знижуючи ризик виникнення помилок та спрощуючи підтримку.

При виборі моделі життєвого циклу програмного забезпечення для розробки цього веб-застосунку було обрано Agile. Цей підхід дозволяє ефективно адаптуватися до змін вимог та швидко реагувати на зворотній зв'язок від користувачів. Завдяки ітеративному процесу розробки, кожен етап проекту розділяється на невеликі, легко керовані спринти, що дозволяє регулярно перевіряти і вдосконалювати програмне забезпечення. Agile підхід також сприяє тісній співпраці між розробниками та іншими зацікавленими сторонами, забезпечуючи прозорість процесу розробки та зменшуючи ризики невідповідності кінцевого продукту очікуванням замовника.

Таким чином, обґрунтування вибору архітектури та шаблонів проектування для програмного забезпечення веб-застосунку інтернет-магазину іграшок показує, що клієнт-серверна модель, підтримувана React і Express.js, у поєднанні з MVC патерном і Agile моделлю життєвого циклу, є оптимальним рішенням для досягнення високої продуктивності, гнучкості та надійності програмного забезпечення.

2.4. Опис використаних технологій та мов програмування

1) Мови програмування та розмітки

– Javascript та його бібліотеки і фреймворки

- React: «front-end» частина веб-додатку;
- Express.js: «back-end» частина веб-додатку;
- React-redux: керування станом кошику веб-додатку;
- Slick-carousel: відображення товарів у вигляді «каруселі»;
- React-slick: відображення товарів у вигляді «каруселі»;
- Styled-components: динамічне змінення стилів товару;
- Axios: запити, що посилаються з «front-end» до «back-end»;

- mongoose: налаштування структури елементів бази даних;

– CSS: стилі інтерфейсу веб-додатку;

– HTML: розмітка інтерфейсу веб-додатку;

2) Бази даних

– MongoDB: сховище даних веб-додатку;

Огляд обраних технологій

Javascript

JavaScript – це потужна і гнучка мова програмування, яка відіграє важливу роль в багатьох областях програмування. Javascript використовують для написання web-додатків, ігор, серверної частини, мобільних додатків, десктопних додатків, штучного інтелекту, інтернету речей, тощо. Високорівнева, інтерпретована, прототипна – це характеристики властиві мові JavaScript. Назву «JavaScript» вона отримала виключно із-за маркетингових міркувань. У 1995 році, коли ця мова була розроблена, була популярна інша мова програмування – Java. Хоч і назву було запозичено, ці дві мови зовсім різні. JavaScript -має багато особливостей.

Одна з основних – це динамічна типізація. Це означає, що тип змінної не є статичним типом даних. Наприклад, якщо змінна на початку оголошення – це масив, то після деяких операцій ця змінна може перетворитися на строковий тип даних. Ця властивість зроблена для того, щоб робота з даними була більш гнучка та швидка. Особливо при роботі з даними з різних джерел, наприклад, через API, можно легко маніпулювати з данними.

Інтерпретованість надає змогу коду виконуватись одразу без попередньої компіляціїю. Це робить процес розробки та налагодження більш швидким та зручнішим.

Типи даних мови JavaScript поділяються на два типи – це примітивні та об’єкти. До примітивних типів даних належать Undefined (Значення змінної, яка

була оголошена, але не ініціалізована.), Boolean (Логічні значення true або false), Number (Числа), String (рядки тексту), BigInt (числа, які більше 2^53), Symbol (унікальні та незмінні ідентифікатори). Об’єкти – це складні типи даних, які мають структуру у вигляді властивостей та методів. Це можуть бути не тільки класичні об’єкти, а й функції, масиви, дата та інше. Така особливість надає змогу полегшити організацію коду через інкапсуляцію. Об’єкти можуть бути зв’язані через ланцюжок [[Prototype]]. Це дозволяє реалізувати спадкування і створювати об’єкти з загальними методами і властивостями.

Також важлива особливість – це асинхронність. Вона є корисною при веб-розробці, де потрібно працювати с UI без блокування основного потоку виконання. Наприклад, коли завантажуються дані з бази даних, користувацький інтерфейс може працювати тому, що завантаження робиться асинхронно.
Кросплатформеність. Мова JavaScript працює на всіх основних платформах та браузерах.

JavaScript – це найпопулярніша мова програмування у світі[3]. Саме її розробники обирають найчастіше. Це можна побачити на графіку на рис. 2.1
Причини популярності цієї мови – це універсальність, кросплатформеність, популярність серед роботодавців, інтеграція з браузером, велика підтримка та багато іншого [4].

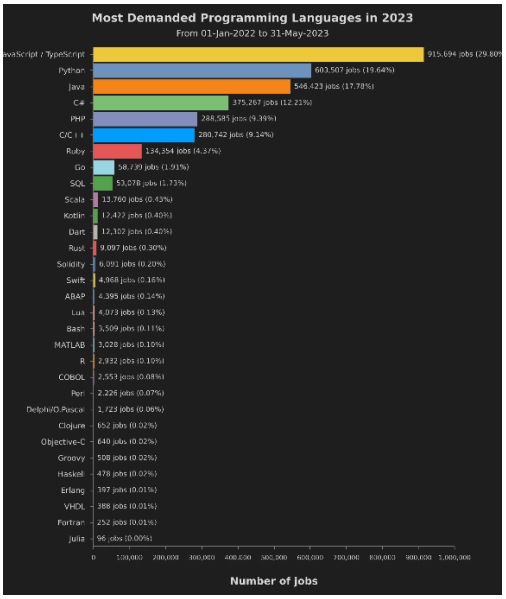

Рис. 2.1. Рейтинг популярності мов програмування

React

React – це одна з найпопулярніших бібліотек JavaScript. Її завдання створювати користувацький інтерфейс, за допомогою компонентів, які є основними одиницями в React. Компонент визначає поведінку якоїсь частини інтерфейсу. Вони бувають функціональні (створені за допомогою функції) та класові (за допомогою класу). Компонент приймає вхідні дані у вигляді пропсів, містить методи та повертає JSX (HTML структуру зі змінними та методами). Такий підхід дозволяє дуже зручно та швидко працювати с DOM деревом. React використовує віртуальний DOM для оптимізації роботи з DOM. React оновлює віртуальний DOM і потім порівнює його з реальним DOM, щоб знайти мінімальний набір змін для оновлення замість безпосередньої взаємодії в реальним DOM при кожній зміні. Це можна побачити на рис. 2.2

React має дуже різноманітну екосистему. До неї належать багато бібліотек. Одними із основних – це Redux, React Router, Webpack, Babel та багато інших. Ці інструменти спрощують розробку, підвищують продуктивність та якість продукту [5].

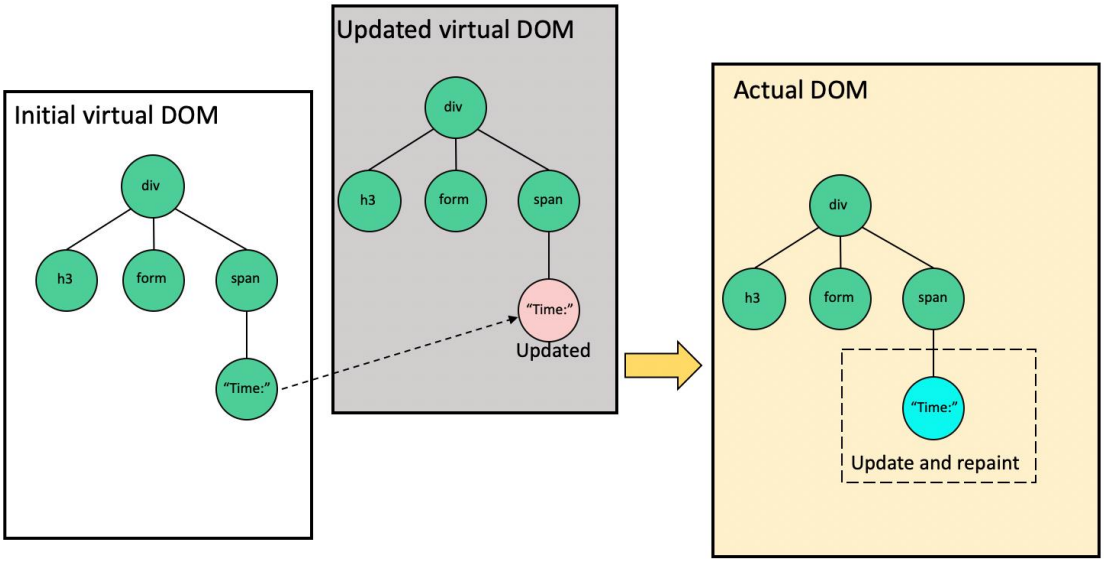

Рис. 2.2. Взаємодія з DOM

Express.js

Express.js – це фреймворк для Node.js, який потрібен для швидкого створювання веб-додатків та API. Він також є одним із найпопулярніших інструментів для побудови серверної частини веб-додатків. Express.js дуже простий та розширювальний.

Фреймворк має багато особливостей. Одна з таких – це маршрутизація. Вона надає змогу визначати URL-шляхи та оброблювати HTTP-запити.

Друга властивість – це middleware. Це функції, які мають доступ до об’єкта запиту, відповіді та наступної функції middleware.

Express.js підтримує використання шаблонізаторів для генерації HTML на сервері.

Фреймворк надає різні методи для створення HTTP-запитів (GET, POST, PUT, DELETE і тд). Це дозволяє легко обробляти данні, які приходять із клієнтів і відправляти їм відвовіді. [6]

Ract-Redux

React-Redux – це бібліотека, яка надає змогу управляти станом додатка. Вона має декілька понять: стан (state), він зберігається в об’єкті під назвою store; дії (actions), це об’єкти, які описують, як змінився додаток; редукторі (reducers), визначають зміну стану додатку відповідно до дій, які були відправлені; стор (store), об’єкт стану додатка.

Процес роботи бібліотеки полягає в тому, що коли потрібну змінити стан додатку, потрібну створити дію та відправити її до стора за допомогою функції dispatch. Потім редуктор обробляє дані та повертає новий стан. Після цього оновлюється компонент. Цей процес можна побачити на рис. 2.3 [7].

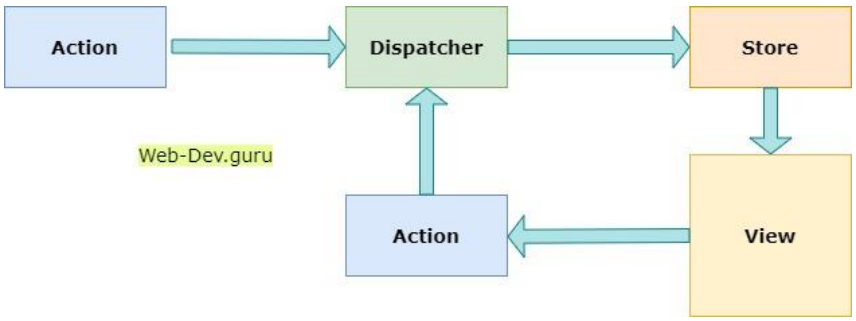

Рис. 2.3. Процес роботи Redux

Slick-carousel

Slick-carousel – це бібліотека для створення каруселей (слайдів) на веб-сторінках. Вона містить функції для створення інтерактивних і адаптивних слайдерів, що можуть містити не тільки зображення а й інший контент. Ця бібліотека є адаптивною та конфігуруємкою та підтримується на багатьох пристроях та браузерах [8].

React-slick

React-slick – це бібліотека, яка створена для інтеграції популярного слайдера Slick в проекти, що використовують бібліотеку React.

React-slick також є адаптивною бібліотекою, яка інтегрована з React. Основна мета її – це створення адаптивних і інтерактивних каруселей на веб-сторінках, використовуючи все функції бібліотеки Slick-carousel [9].

Styled-components

Styled-components – це бібліотека для стилізації компонентів React за допомогою CSS. Вона дозволяє інтегрувати CSS в компоненти React. Це спрощує роботу зі стилями і зберігає їх локально до компонентів [10].

Axios

Axios – це бібліотека яка надає простий та зручний інтерфейс для здійснення HTTP-запитів. Через свою простоту на потужність вона є одною з найпопулярніших бібліотек для мережевої взаємодії. Вона підтримує промиси

для оброблення асинхронних запитів, може перехоплювати помилки в HTTP, дозволяє відправляти файли через HTTP-запити. Одна з найважливіших переваг є те, що вона підтримується як в браузері так і Node.js [11].

Mongoose

Mongoose – це бібліотека на основі Node.js для роботи з базою даних. Вона надає змогу за допомогою схем визначати структуру даних, де є типи даних, валідація і тд. Mongoose дозволяє створювати моделі для виконання CRUD-операцій (create, read, update, delete). Також ця бібліотека може підтримувати плагіни для розширення функціональності і повторного використання логіки. Також її використовують часто у зв'язці з MongoDB (база данних) [12].

MongoDB

MongoDB – документо-орієнтована система керування базами даних (СКБД) з відкритим вихідним кодом, яка не потребує опису схеми таблиць. MongoDB займає нішу між швидкими і масштабованими системами, що оперують даними у форматі ключ/значення, і реляційними СКБД, функціональними і зручними у формуванні запитів [13].

2.5. Опис структури програми та алгоритмів її функціонування

2.5.1. Структура системи

Програма складається з двох основних частин: клієнтської та серверної частини.

Серверна частина розроблена на основі Node.js з використанням фреймворку Express.js та бази даних MongoDB. Вона включає декілька основних компонентів:

− Моделі (Models) визначають структуру даних, що зберігаються в базі даних MongoDB.

– Роутери (Routers) визначають шляхи для взаємодії клієнтської частини з сервером. Наприклад, шлях /itemProducts використовується для отримання, створення, оновлення та видалення продуктів. Шляхи /filters використовуються для роботи з фільтрами. Шляхи /register та /login використовуються для реєстрації та авторизації користувачів. Шлях /me дозволяє отримати дані поточного користувача, а шлях /orders використовується для створення замовлень.

– Мідлвари (Middlewares) включають auth, яка відповідає за аутентифікацію користувачів на основі JWT токенів.

Клієнтська частина розроблена на основі React.js і включає кілька компонентів, які забезпечують функціональність веб-застосунку. Маршрутизація, зроблена за допомогою бібліотеки «react-router-dom», налаштована наступним чином:

```
<Router>
      <Routes>
        <Route exact path="/" element={<MainPage/>}></Route>
        <Route exact path="/admin" element={<AdminPage/>}></Route>
        <Route exact path="/itemPage/:productId" element={<ItemPage/>}></Route>
                            <Route      exact      path="/itemSearchPage/:productId"
element={<SearchItemPage/>}></Route>
                              <Route      exact      path="/itemFilterPage/:filter"
element={<SubFilterPage/>}></Route>
                           <Route      exact      path="/itemCategoryPage/:category"
element={<FilterPage/>}></Route>

      </Routes>
    </Router>
```

Алгоритми функціонування програми

Алгоритм реєстрації користувача починається з того, що користувач заповнює форму реєстрації на клієнтській частині. Форма відправляє POST запит на /register на сервері. Сервер перевіряє, чи існує вже користувач з таким ім'ям, і якщо не існує, хешує пароль користувача та зберігає дані нового користувача в базі даних. Сервер відповідає клієнту повідомленням про успішну реєстрацію. Нижче наведено код цього процесу:

```
app.post('/register', async (req, res) => {
    const { username, password } = req.body;

    try {
        const existingUser = await User.findOne({ username });
        if (existingUser) {
            return res.status(400).json({ error: 'User already exists' });
        }

        const hashedPassword = await bcrypt.hash(password, 10);
        const newUser = new User({ username, password: hashedPassword });
        await newUser.save();

        res.status(201).json({ message: 'User registered successfully' });
    } catch (error) {
        console.error('Error registering user:', error);
        res.status(500).json({ error: 'Internal server error' });
    }
});
```

Алгоритм авторизації користувача починається з того, що користувач заповнює форму входу на клієнтській частині. Форма відправляє POST запит на /login на сервері. Сервер перевіряє, чи існує користувач з таким ім'ям, і якщо існує, перевіряє правильність введеного паролю. Якщо дані вірні, сервер генерує JWT токен, який повертається клієнту та зберігається в localStorage для подальшого використання. Нижче наведено код цього процесу:

```
app.post('/login', async (req, res) => {
    const { username, password } = req.body;

    try {
        const user = await User.findOne({ username });
        if (!user) {
            return res.status(400).json({ error: 'Invalid credentials' });
        }

        const isMatch = await bcrypt.compare(password, user.password);
        if (!isMatch) {
            return res.status(400).json({ error: 'Invalid credentials' });
        }

        const token = jwt.sign({ userId: user._id }, secret, { expiresIn: '1h' });

        // Устанавливаем токен в куки
```

```
        res.cookie('token', token, { httpOnly: true });

        res.json({ token });
    } catch (error) {
        console.error('Error logging in:', error);
        res.status(500).json({ error: 'Internal server error' });
    }
});
```

Алгоритм отримання даних користувача передбачає відправлення клієнтом GET запиту на /me на сервері з JWT токеном у заголовку запиту. Сервер верифікує токен та повертає дані користувача з бази даних. Нижче наведено код цього процесу:

```
app.get('/me', auth, async (req, res) => {
    try {
        const user = await User.findById(req.user.userId).populate('orders');
        res.json(user);
    } catch (error) {
        console.error('Error fetching user data:', error);
        res.status(500).json({ error: 'Internal server error' });
    }
});
```

Алгоритм оформлення замовлення починається з того, що користувач відкриває корзину та натискає кнопку "Перейти до оформлення замовлення". Відкривається модальне вікно OrderModal, де відображаються дані користувача та товари в корзині. Користувач підтверджує замовлення, і форма відправляє POST запит на /orders на сервері. Сервер зберігає замовлення в базі даних та оновлює дані користувача. Сервер відповідає клієнту повідомленням про успішне оформлення замовлення. Нижче наведено код цього процесу:

```
app.post('/orders', auth, async (req, res) => {
    const { items, totalCost } = req.body;

    try {
        const newOrder = new Order({
            userId: req.user.userId,
            items,
            totalCost
```

```
        });

        await newOrder.save();
       await User.findByIdAndUpdate(req.user.userId, { $push: { orders: newOrder._id
} });

        res.status(201).json(newOrder);
    } catch (error) {
        console.error('Error creating order:', error);
        res.status(500).json({ error: 'Internal server error' });
    }
});
```

Стан корзини реалізовано за допомогою бібліотеки Redux. Створюється ред'юсер для керування станом корзини покупок. Він використовує Redux для централізованого управління станом і включає різні дії (actions), які визначають, як стан змінюється у відповідь на різні типи дій. Нижче наведений код реалізації ред’юсера:

```
const initialState = {
  items: [],
  status: "idle",
};

const reducer = (state = initialState, action) => {
  switch (action.type) {
    case "ITEMS_FETCHING":
      return {
        ...state,
        status: "loading",
      };
    case "ITEMS_FETCHED":
      return {
        ...state,
        items: action.payload,
        status: "idle",
      };
      case 'REMOVE_ITEM':
        return {
          ...state,
          items: state.items.filter(item => item.id !== action.payload)
        };
    case "ITEMS_FETCHING_ERROR":
      return {
        ...state,
        status: "error",
```

```
      };

    default:
      return state;
  }
};

export default reducer;
```

Як можна побачити, спочатку створюється об'єкт, який визначає початковий стан корзини. Він містить два поля:

– items: масив товарів, що додаються до корзини. Початково він порожній.

– status: рядок, що вказує на поточний стан асинхронних операцій. Початково значення встановлено на "idle", що означає, що система не зайнята ніякими операціями.

Після цього створюється сам ред’юсер, у якому вказано, як програма буде реагувати на певну дію. Наприклад, "ITEMS_FETCHING" змінює стан, коли починається асинхронна операція завантаження товарів (наприклад, запит до API). "ITEMS_FETCHED" виконується, коли товари успішно завантажені.

Алгоритм динамічного створення сторінки товару працює наступним чином: кожен товар має ідентифікатор, який додається до маршруту компонента. Також, через стан браузера, передаються потрібні дані. Нижче наведено код реалізації цього алгоритму:

```
<Link to={`/itemPage/${id}`} state={{ id, img, cost, name }}>
          <img className='card-img' src={img} alt={name} />
        </Link>
```

Компонент ItemPage отримує передані дані, та формує на їх основі сторінку товару. Отримання даних реалізовано за допомогою хука useLocation:

```
 const location = useLocation();
  const id = location.state?.id;
  const img = location.state?.img;
  const cost = location.state?.cost;
  const name = location.state?.name;
```

JWT аутентифікація забезпечує безпечне зберігання та передачу даних користувача між клієнтом та сервером. Це дозволяє зберігати сесію користувача без використання серверних сесій, що знижує навантаження на сервер. Чітке розділення функцій між компонентами дозволяє забезпечити модульність і зрозумілість коду. Кожен компонент відповідає за конкретну частину функціоналу, що полегшує підтримку та розширення програми. Використання асинхронних запитів забезпечує швидкий і ефективний обмін даними між клієнтом та сервером, що підвищує продуктивність і покращує користувацький досвід.

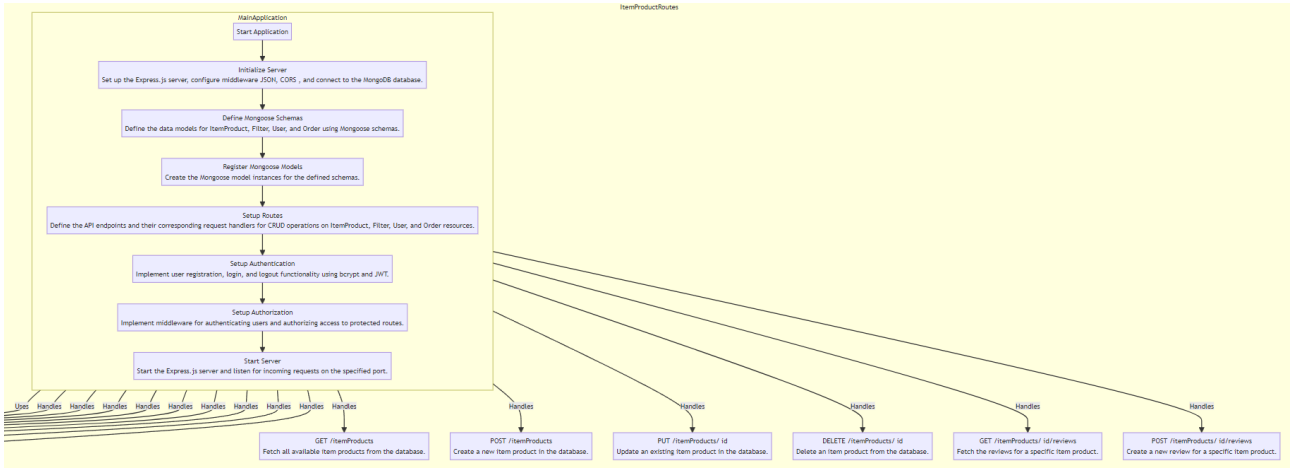

Рис. 2.4. Схема алгоритму функціонування програми

Рис. 2.5 Схема алгоритму функціонування програми (продовження)

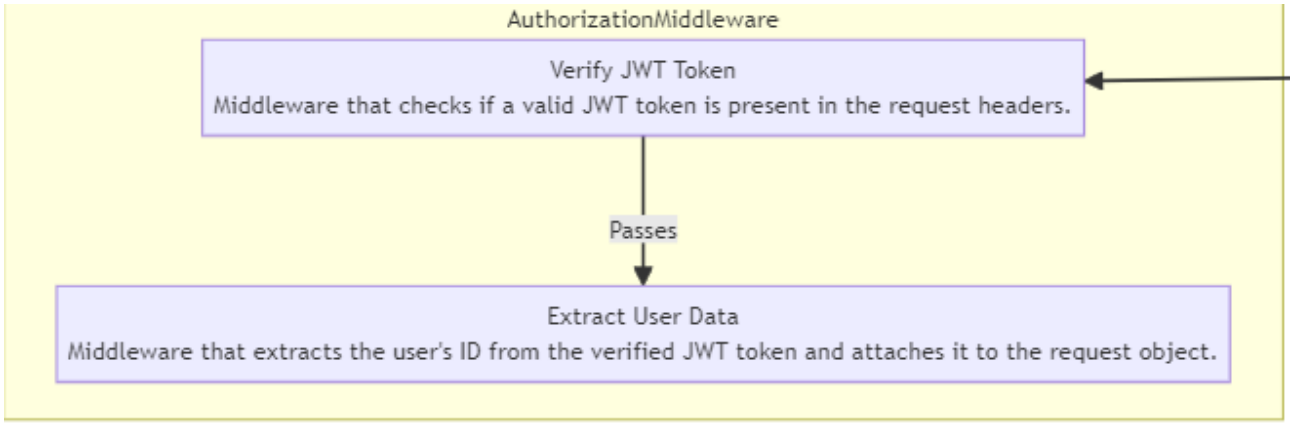

Рис. 2.6. Схема алгоритму функціонування програми (продовження (2))

2.5.2. Структура бази даних

Сховищем даних веб-додатку інтернет-магазину є NoSQL(нереляційна) база даних MongoDB. Бази даних MongoDB, замість таблиць, складаються з таких складових, як колекції, які у свій час складаються з документів, що зберігають дані у форматі BSON.

Перелік колекцій та їх документів

База даних містить такі колекції:

1) filters – зберігає дані про категорії та підкатегорії магазину. Документ цієї колекції складається з таких елементів:

– _id: унікальний ідентифікатор документа, встановлюється MongoDB;

– id: ідентифікатор, який встановлює адміністратор;

– name: назва категорії;

– category: ключове слово, за яким відбувається сортування на категорії товарів;

– subfilters: масив підфільтрів категорій, елементи якого складаються з наступних атрибутів:

– _id: унікальний ідентифікатор підфільтра, встановлюється MongoDB;

– id: ідентифікатор, який встановлює адміністратор;

– filterName: ключове слово, за яким відбувається сортування на підфільтри категорій товарів;

– name: назва підфільтру;

2) itemProducts: зберігає дані про товари магазину. Документ цієї колекції складається з таких елементів:

– _id: унікальний ідентифікатор документа, встановлюється MongoDB;

– id: ідентифікатор, який встановлює адміністратор;

– name: назва товару;

– img: зображення товару;

– cost: ціна товару:

– species: додатковий фільтр, за яким товари розбиваються на такі блоки, як «Нові товари», «Найкраще».

– category: ключове слово, за яким товар відноситься до відповідної категорії;

– subcategory: ключове слово, за яким товар відноситься до відповідного підфільтру;

– colors: масив кольорів та відповідних зображень товарів, елементи якого складаються з наступних атрибутів:

– _id: унікальний ідентифікатор документа, встановлюється MongoDB;

– color: колір товару;

– image: зображення товару відповідного кольору;

3) users – зберігає дані про зареєстрованих клієнтів магазину. Документ цієї колекції складається з таких елементів:

– _id: унікальний ідентифікатор документа, встановлюється MongoDB;

– username: ім’я клієнта;

– password: захешований пароль клієнта;

– phone: номер телефону клієнта;

– orders: массив, що зберігає ідентифікатори замовлень клієнта;

4) orders – зберігає дані про замовлення клієнтів. Документ цієї колекції складається з таких елементів:

– _id: унікальний ідентифікатор документа, встановлюється MongoDB;

– userId: ідентифікатор клієнта, який створюється при реєстрації клієнта. За ним замовлення відносяться до відповідного клієнта;

– totalCost: загальна ціна;

– status: статус замовлення;

– items: массив замовлених товарів, елементи якого складаються з наступних атрибутів:

– _id: унікальний ідентифікатор товару, встановлюється MongoDB;

– name: назва товару;

– cost: ціна товару;

– quantity: кількість товару;

2.6. Обґрунтування та організація вхідних та вихідних даних програми

Вхідними даними сайту є:

− дані, що вводяться при реєстрації та вході(паролі хешуються);

− дані, що вводяться при пошуку;

− дані, що вводяться адміністратором при створенні елементів сайту;

Вихідних даними сайту є:

− дані всіх елементів сайту(товари, фільтри та ін.);

− дані користувача в особистому кабінеті та список замовлень;

2.7. Опис розробленого програмного продукту

2.7.1. Використані технічні засоби

Проект даної кваліфікаційної роботи розроблений та протестований на ПК з наступними характеристиками:

Таблиця 2.1

Технічні характеристики ноутбука Acer Nitro

Характеристика	**Деталі**
Основні	
Модельний ряд	Acer Nitro
Тип	Ноутбук
Конструкція	Класична
Екран	
Діагональ дисплея	15.6"

Продовження табл. 2.1

Роздільна здатність дисплея	1920 x 1080
Тип матриці	IPS
Частота оновлення екрана	144 Гц
Процесор	
Серія процесора	AMD Ryzen 5
Модель процесора	5600H
Базова частота процесора	3.3 ГГц
Кількість ядер	6
Оперативна пам'ять/ОЗП	
Об'єм ОЗП	16 Гб
Тип пам'яті	DDR4
Накопичувачі даних/Жорсткий диск	
Тип накопичувача	SSD
Об'єм SSD диска	512 Гб
Відеокарта	
Тип відеокарти	Дискретна
Виробник відеокарти	NVIDIA
Модель відеокарти	GeForce RTX 3050
Об'єм відеопам'яті	4 Гб

2.7.2. Використані програмні засоби

Під час розробки веб-додатку інтернет-магазину іграшок були використані наступні програмні засоби:

– Visual Studio Code;

– MongoDB Atlas;

– MongoDB Database Tools;

– Node.js;

– Figma;

Опис використаних програмних засобів

Visual Studio Code

Visual Studio Code (VS Code) — це популярний, безкоштовний та відкритий редактор коду, розроблений компанією Microsoft. Він був представлений у 2015 році і з того часу здобув величезну популярність серед розробників завдяки своїм потужним можливостям, розширюваності та зручному інтерфейсу. Однією з основних характеристик VS Code є підтримка багатьох мов програмування. З коробки він підтримує JavaScript, Python, Java, C++, HTML, CSS та багато інших мов. Додатково, через розширення, можна додати підтримку ще більшої кількості мов. VS Code забезпечує розширену функціональність автозаповнення коду через IntelliSense, яка надає підказки та автодоповнення на основі синтаксису, визначень функцій і навіть імпортованих модулів [14].

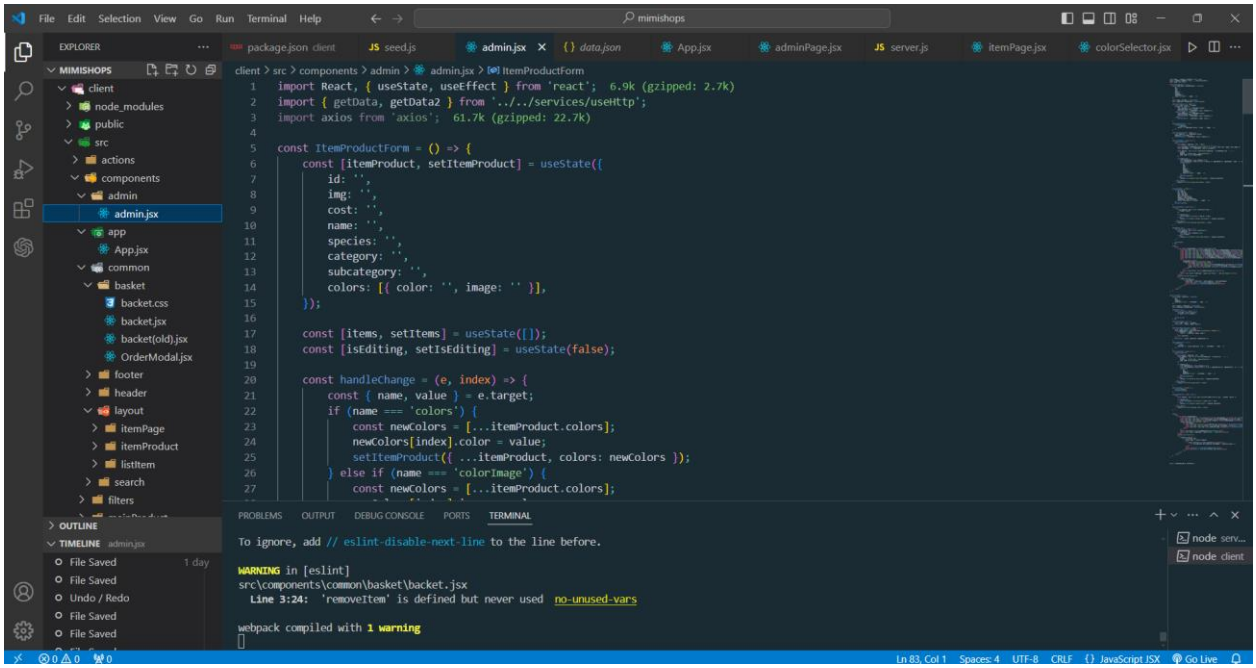

Рис. 2.7. Інтерфейс Visual Studio Code

MongoDB Atlas

MongoDB Atlas — це хмарна служба баз даних, що надається компанією MongoDB Inc. Вона пропонує повністю керовану версію бази даних MongoDB,

що дозволяє розробникам легко створювати, розгортати і управляти базами даних у хмарному середовищі. MongoDB Atlas підтримує основні хмарні платформи, такі як Amazon Web Services (AWS), Google Cloud Platform (GCP) та Microsoft Azure, забезпечуючи високий рівень доступності, безпеки та продуктивності.

Основні характеристики MongoDB Atlas включають автоматизоване масштабування бази даних, що дозволяє легко адаптуватися до змін у вимогах додатків. Користувачі можуть збільшувати або зменшувати ресурси, такі як CPU, пам'ять і зберігання даних, залежно від поточних потреб. Також MongoDB Atlas надає автоматичне резервне копіювання та відновлення, що дозволяє захистити дані від втрати та забезпечити швидке відновлення у разі непередбачених ситуацій [15].

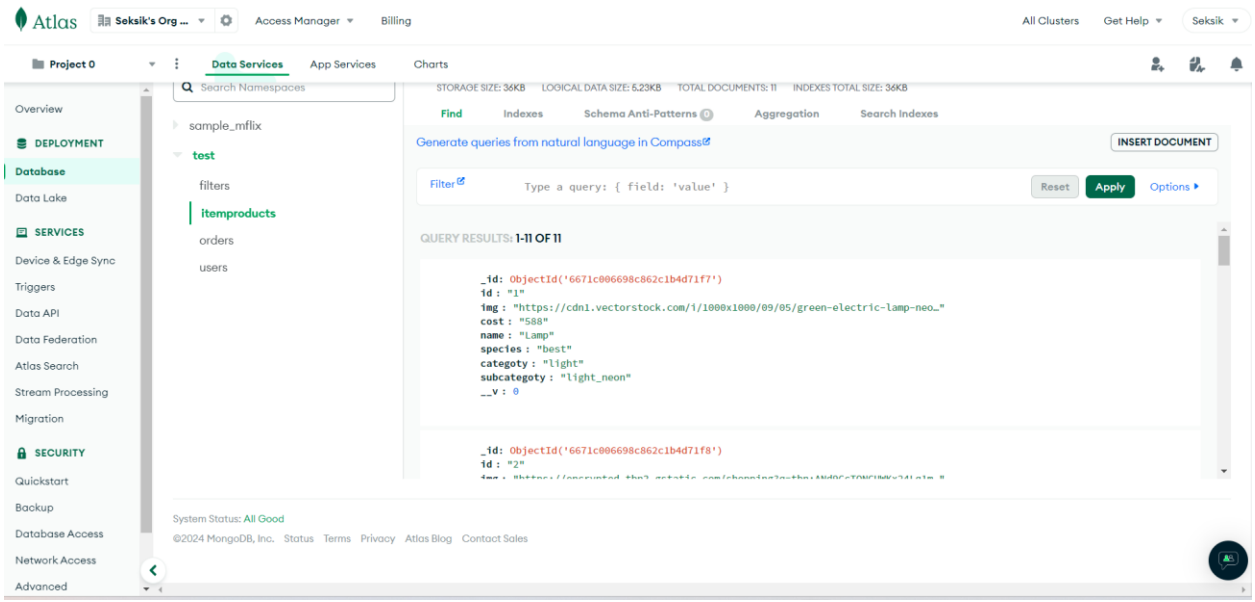

Рис. 2.8. Робоча площина MongoDB Atlas

MongoDB Database Tools

MongoDB Database Tools — це набір командних утиліт, які допомагають розробникам і адміністраторам баз даних виконувати різні завдання з управління базами даних MongoDB. Ці інструменти є важливими для роботи з даними,

забезпечення їхньої безпеки, міграції та аналізу. Основні інструменти, що входять до MongoDB Database Tools, включають:

1. mongodump і mongorestore: Ці утиліти використовуються для створення резервних копій баз даних і їх відновлення. mongodump створює дамп даних, який може бути збережений і використаний для відновлення бази даних у випадку втрати даних або для міграції на інший сервер. mongorestore відновлює дані з резервної копії, створеної за допомогою mongodump.

2. mongoimport і mongoexport: Ці інструменти дозволяють імпортувати та експортувати дані у форматі JSON, CSV або TSV. mongoimport використовується для завантаження даних з файлів у базу даних MongoDB, а mongoexport — для вивантаження даних з бази даних у файли. Це зручно для переносу даних між різними системами або для аналізу даних за допомогою зовнішніх інструментів.

3. mongostat: Ця утиліта забезпечує моніторинг стану сервера MongoDB в реальному часі. Вона відображає інформацію про операції читання та запису, використання пам'яті, з'єднання та інші важливі метрики, що дозволяє адміністраторам баз даних оцінювати продуктивність і виявляти потенційні проблеми.

4. mongotop: показує статистику використання часу на рівні колекцій у базі даних. Вона надає інформацію про те, скільки часу було витрачено на читання та запис даних у кожну колекцію, що дозволяє визначити найактивніші колекції та оптимізувати їх роботу.

5. bsondump: Ця утиліта перетворює файли BSON (бінарне представлення JSON) у формат JSON, що дозволяє розробникам і адміністраторам баз даних переглядати та аналізувати вміст дампів MongoDB у зручному для читання форматі.[16]

Node.js

Node.js — це середовище виконання JavaScript, яке дозволяє запускати JavaScript-код на сервері. Розроблене Райаном Далем у 2009 році, Node.js використовує движок V8 від Google Chrome для виконання JavaScript,

забезпечуючи високу продуктивність та ефективність. Його асинхронна, неблокуюча модель вводу/виводу робить його ідеальним для створення масштабованих мережевих додатків.

Однією з ключових особливостей Node.js є його подієво-орієнтована архітектура, яка дозволяє обробляти велику кількість одночасних запитів без блокування. Це досягається через використання подій та зворотних викликів (callbacks), що дозволяє серверам швидко реагувати на запити без необхідності чекати завершення попередніх операцій.

Node.js також має багату екосистему модулів та пакетів, що управляються через npm (Node Package Manager). Npm дозволяє розробникам легко встановлювати, оновлювати та керувати зовнішніми бібліотеками та інструментами, що значно спрощує процес розробки. Бібліотека модулів Node.js охоплює широкий спектр функціональностей, від роботи з базами даних і веб-серверами до інструментів для тестування та аналітики.[17]

Figma

Figma - це онлайн інструмент для дизайну і прототипування інтерфейсів, який дозволяє створювати макети, векторні графіки та інтерактивні прототипи. Основні переваги включають можливість колаборації в реальному часі, роботу у веб-браузері без потреби у встановленні додаткового ПЗ, векторний редактор для створення складних ілюстрацій, анімацій і систем компонентів для консистентного дизайну. Figma інтегрується з іншими інструментами розробки, полегшуючи співпрацю та обмін ресурсами між командами.

Початковий макет інтернет-магазину був створений у Figma. По ходу створення сайту вносилися певні зміни. Нижче можна побачити макет сайту з Figma [18].

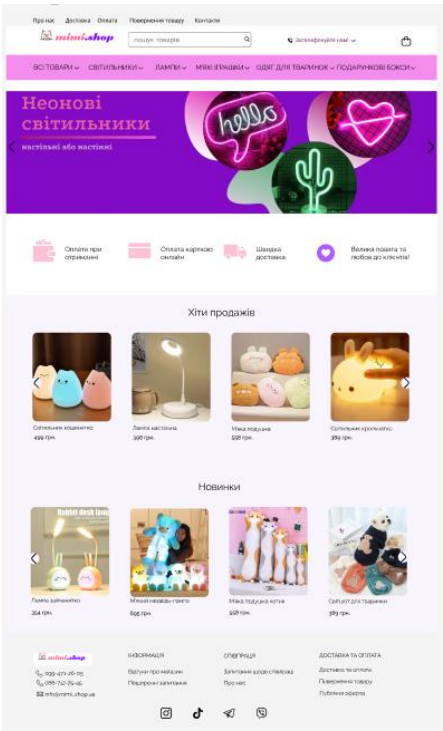

Рис. 2.9. Макет сайту у Figma

2.7.3. Виклик та завантаження програми

Виклик та завантаження програми відбувалися з локального сервера ПК (localhost:3000 – React, localhost:5000 – Express app). Для цього потрібно виконати наступні дії:

– Встановити середовище виконання JS – Node.js;

– Встановити node modules, потрібні для коректної роботи додатка;

– Налаштувати перенаправлення запитів з клієнтської частини на серверну за допомогою додавання рядка "proxy": "http://localhost:5000" у файл package.json React-додатку. Також можна це зробити за допомогою бібліотеки cors.

– Запустити React-додаток (запускається базово на порті 3000) та Express-додаток на порті, відмінному від React (для тесту було обрано порт 5000)

– Ввести у адресний рядок браузера URL: localhost:3000;

– Якщо все встановлено правильно та введена дійсна адреса, відкривається головна сторінка.

2.7.4. Опис інтерфейсу користувача.

Після завантаження сайту, відкривається головна сторінка (рис. 2.10).

Рис. 2.10. Головна сторінка

Вона складається з трьох елементів: «хедер» (рис. 2.11), тобто шапка сайту, основна частина та «футер» (рис. 2.21), тобто підвал сторінки.

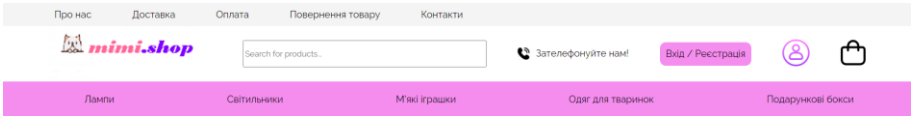

Рис. 2.11. «Хедер» сайту

У свою чергу, шапка складається з трьох частин: верхня стрічка з посиланнями на інформативні сторінки сайту; середня частина, що містить логотип, поле для пошуку, посилання на дзвінок менеджеру, кнопка для відкриття модального вікна входу / реєстрації, кнопка для переходу в особистий кабінет та кошик; нижня стрічка, яка містить категорії товарів(при натисканні відкривається сторінка з товарами відповідної категорії, а при наведенні курсора відображаються підфільтри категорій (рис. 2.12), які ведуть на сторінку з відфільтрованими товарами).

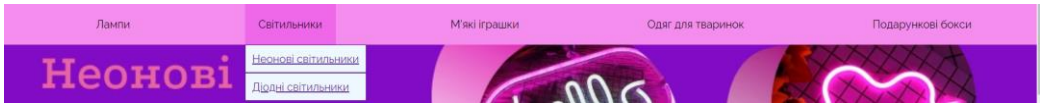

Рис. 2.12. Результат наведення курсора на кнопку категорії

Натиснувши на кнопку «Зателефонуйте нам!», відкривається запрошення на здійснення дзвінка (рис. 2.13).

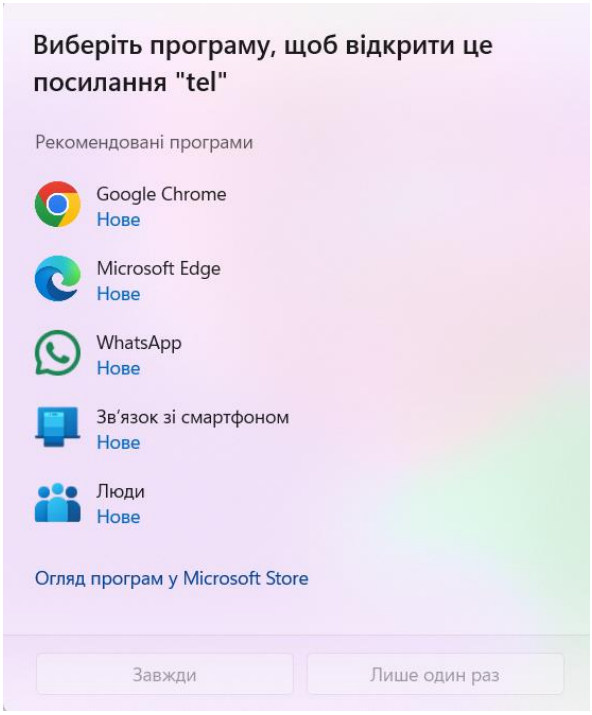

Рис. 2.13. Запрошення на здійснення дзвінка

Натиснувши на кнопку «Вхід / Реєстрація», відкривається модальне вікно входу та реєстрації (рис. 2.14)

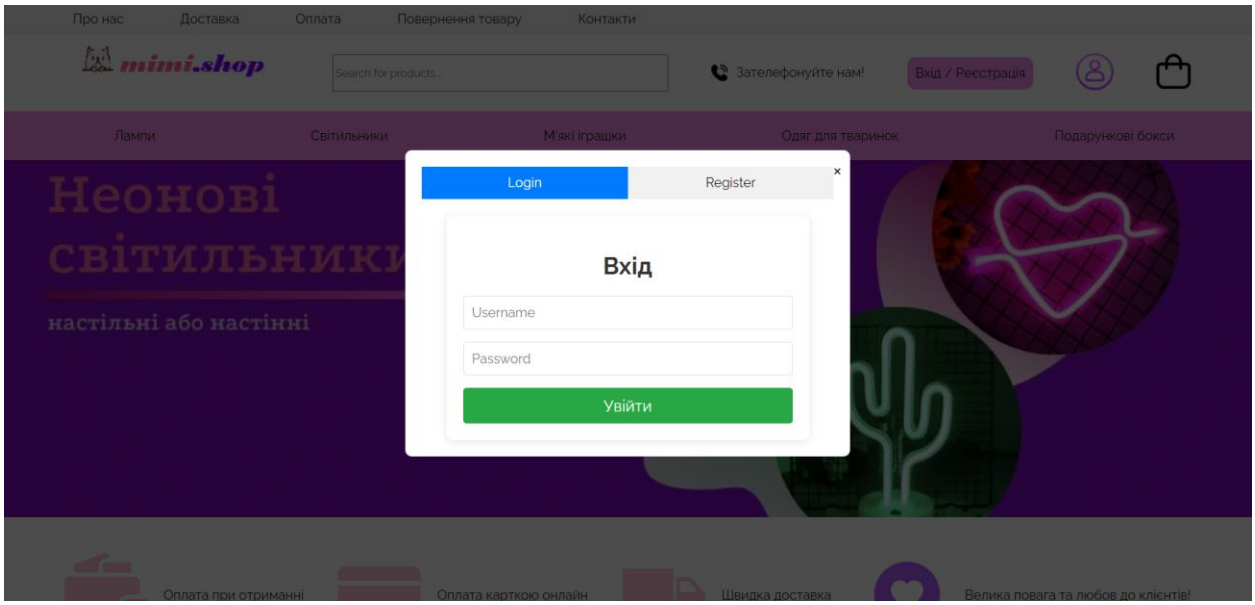

Рис. 2.14. Модальне вікно входу та реєстрації

Перед входом у свій акаунт, користувачу потрібно заповнити форму реєстрації, яка відкривається при натисканні кнопки «Register». Після успішної реєстрації користувач бачить відповідне повідомлення (рис. 2.15) та має змогу увійти до особистого кабінета

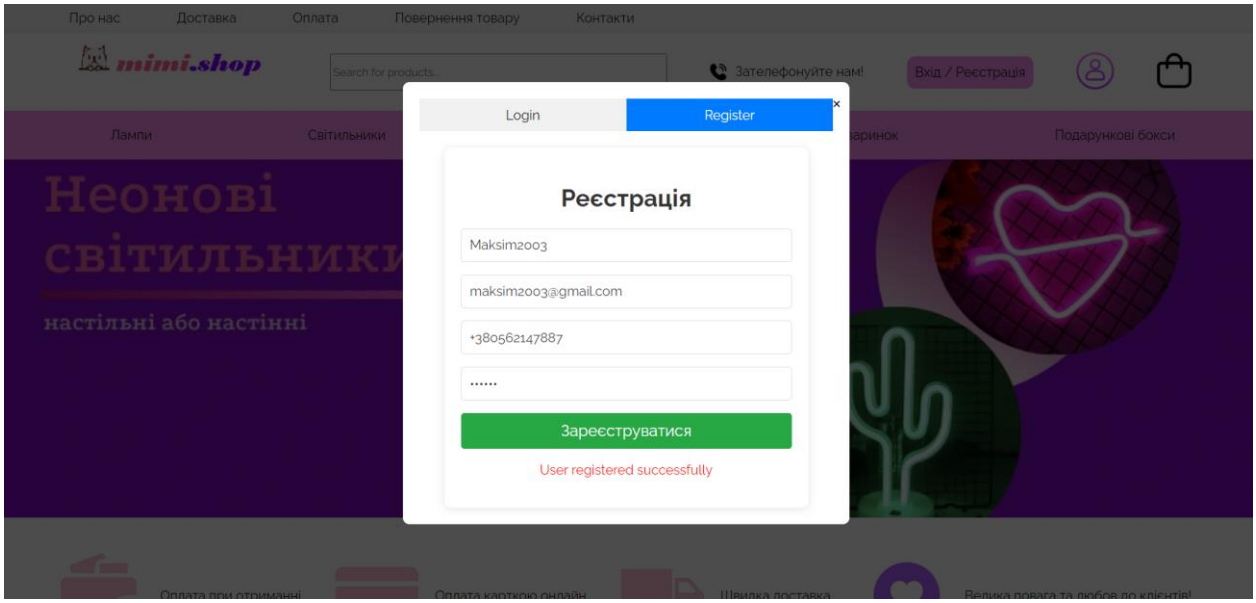

Рис. 2.15. Результат успішної реєстрації

Після реєстрації вхід здійснюється автоматично. Натиснувши кнопку справа від «Вхід / Реєстрація», користувач потрапляє до особистого кабінету (рис. 2.16)

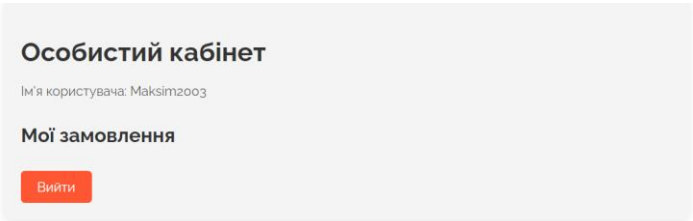

Рис. 2.16. Особистий кабінет

Навівши курсор на іконку кошика, відкривається вікно кошика (рис. 2.17). Якщо він порожній, то містить відповідне повідомлення.

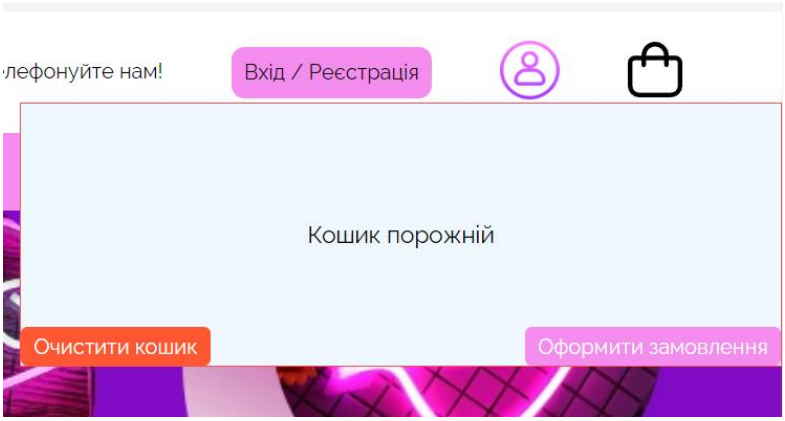

Рис. 2.17. Порожній кошик

Основна частина головної сторінки складається с таких елементів: банер (рис. 2.18), карусель з товарами категорії «Нові товари» (рис. 2.19), карусель з товарами категорії «Найкраще» (рис. 2.20).

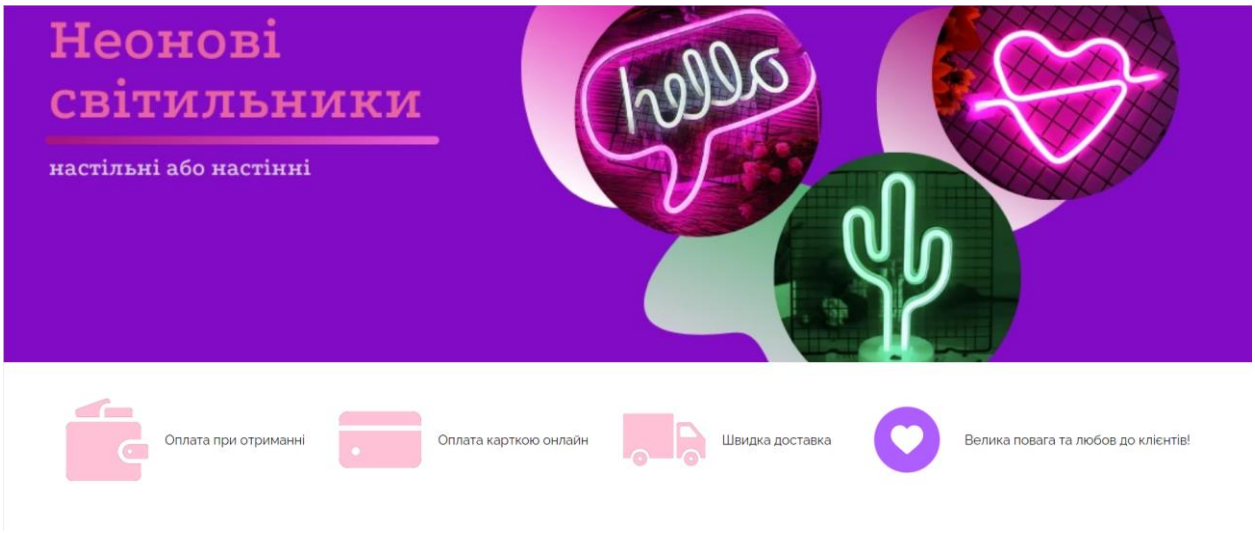

Рис. 2.18. Банер сайту

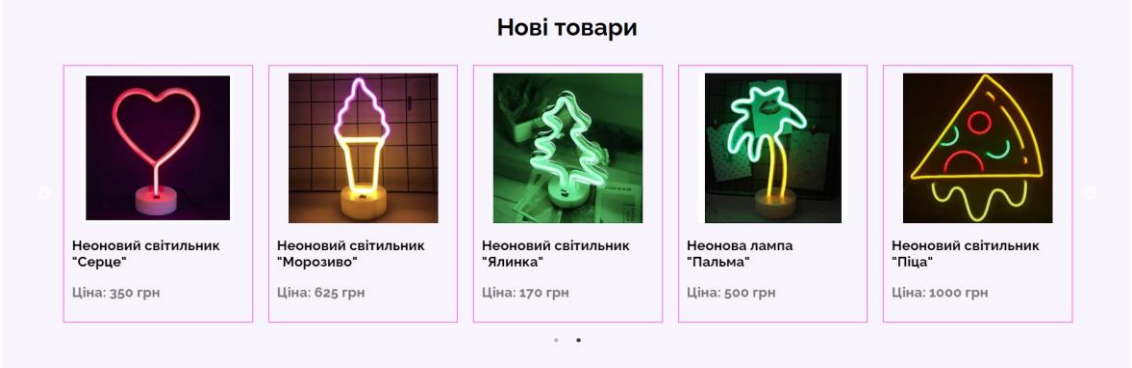

Рис. 2.19. Карусель з товарами категорії «Нові товари»

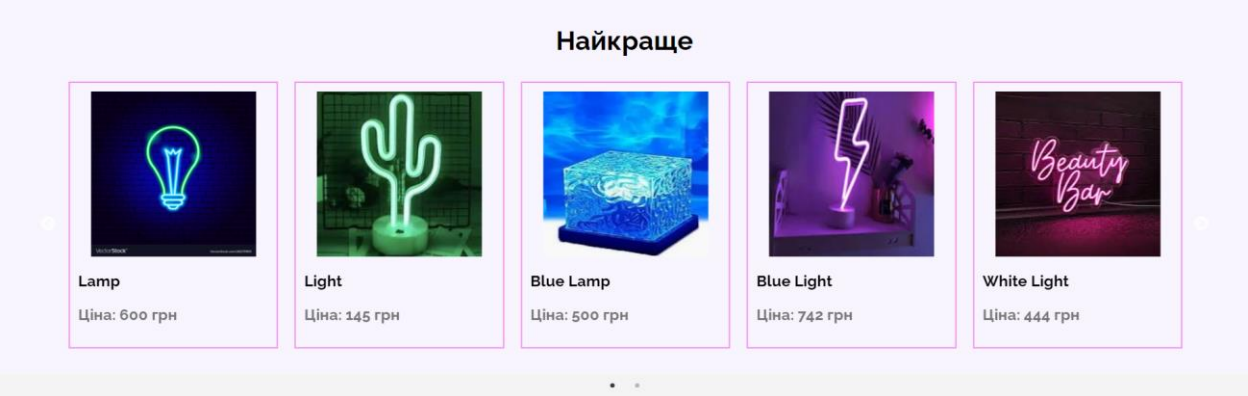

Рис. 2.20. Карусель з товарами категорії «Найкраще»

«Футер» (рис. 2.21) складається з логотипу, переліку контактних даних та посилань на інформативні сторінки сайту.

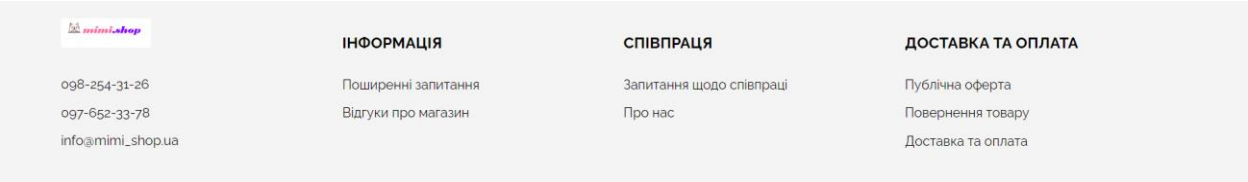

Рис. 2.21. «Футер» сайту

Натиснувши на кнопку категорії (для огляду обрана категорія «Світильники»), користувач потрапляє на сторінку каталогу. Вона також складається з «хедеру», «футеру» та основної частини. На рис. 2.22 Можна побачити основну частину каталогу.

Рис. 2.22. Основна частина каталогу

Обравши підфільтр каталогу (для огляду обраний підфільтр «Діодні світильники»), відображаються товари відповідної фільтрації (рис. 2.23)

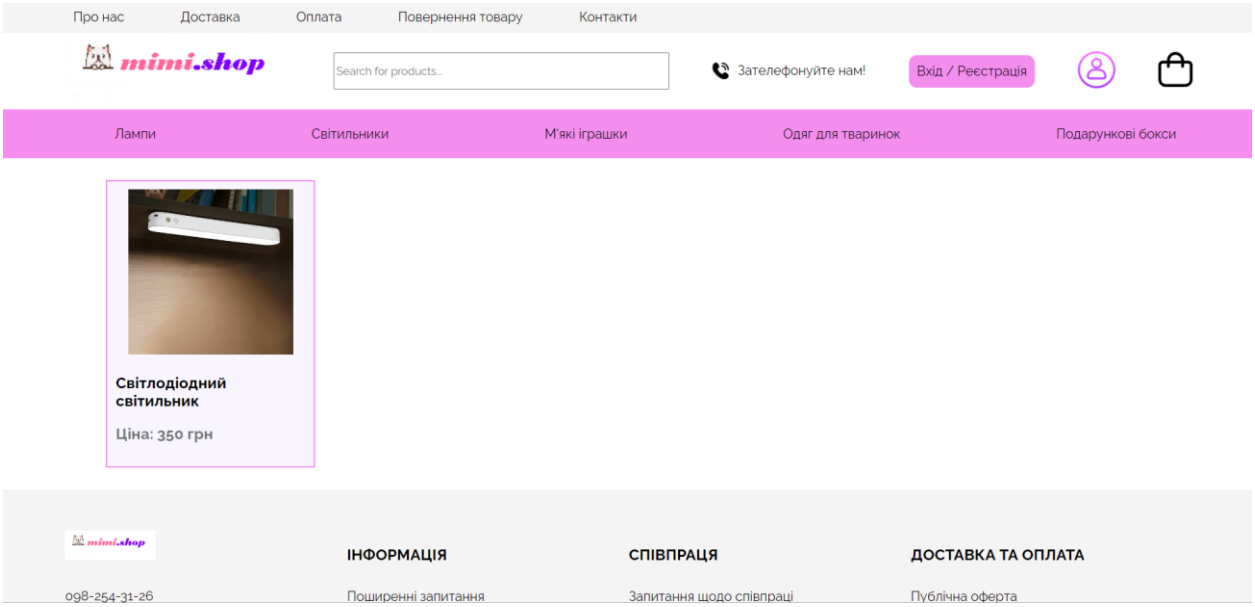

Рис. 2.23. Підфільтр «Діодні світильники»

Натиснувши на карту товару, користувач потрапляє на сторінку відповідного товару. Вона складається з тих же частин, що і попередні сторінки. Основна частина цієї сторінки складається із блоку товару (рис. 2.24), що містить зображення товару, його найменування, ціни, блоку для вибору кольору, поля для введення кількості товару, загальної вартості та кнопки додавання товару у кошик.

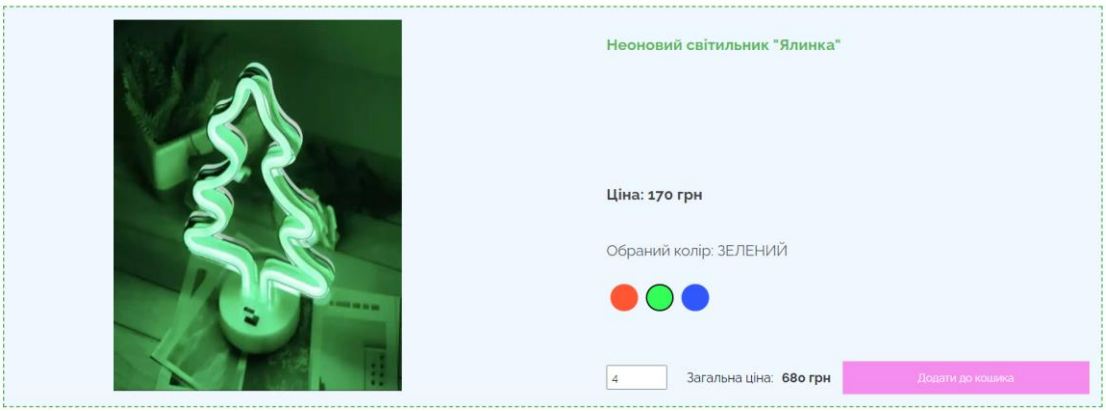

Рис. 2.24. Блоку товару

Нижче знаходиться блок з інформацією про товар (рис. 2.25), доставку та оплату (рис. 2.26), відгуки про товар (рис. 2.27)

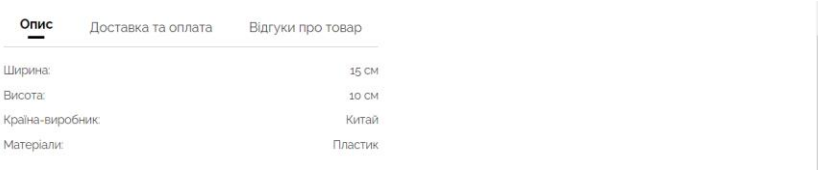

Рис. 2.25. Опис товару

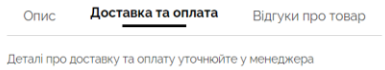

Рис. 2.26. Доставка та оплата

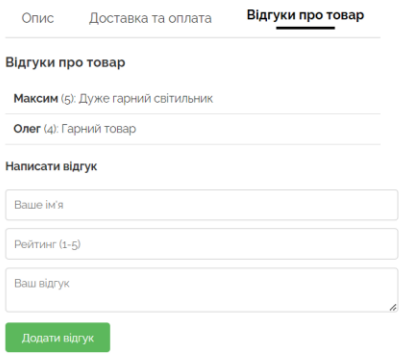

Рис. 2.27. Відгуки про товар

Після додавання товарів у кошик, вони з’являються у ньому (рис. 2.28)

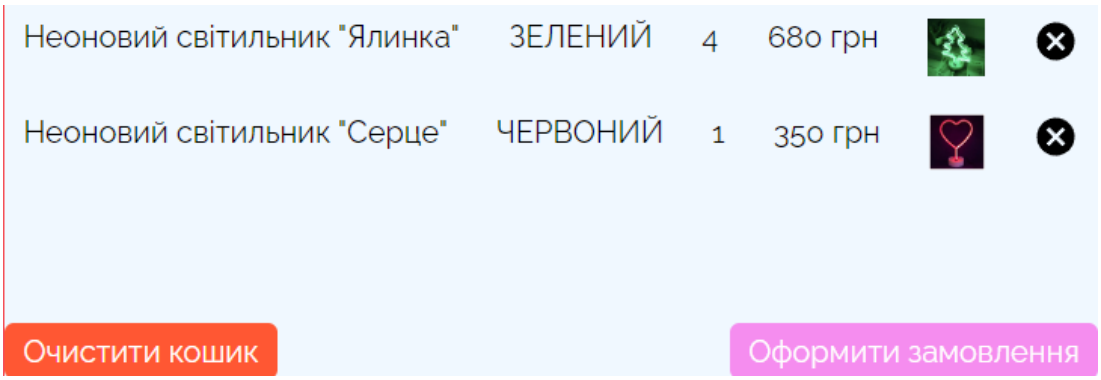

Рис. 2.28. Кошик із доданими товарами

Натиснувши кнопку «Оформити замовлення», відкривається вікно з підтвердженням замовлення (рис. 2.29).

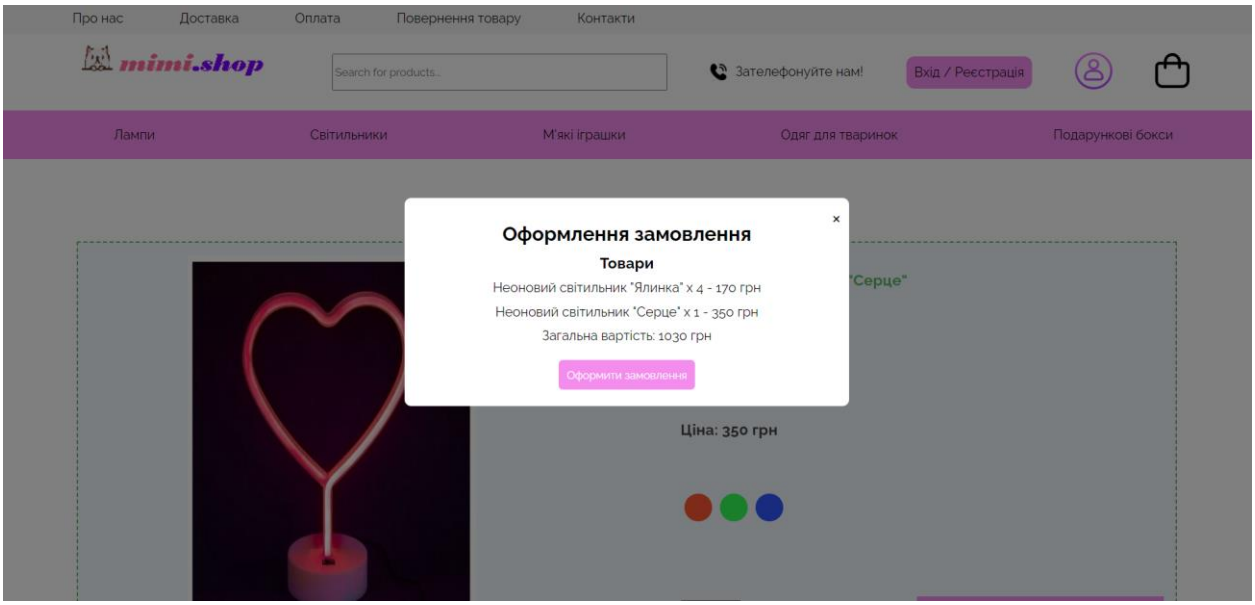

Рис. 2.29. Підтвердження замовлення

Після оформлення замовлення, воно відображається в особистому кабінеті користувача (рис. 2.30).

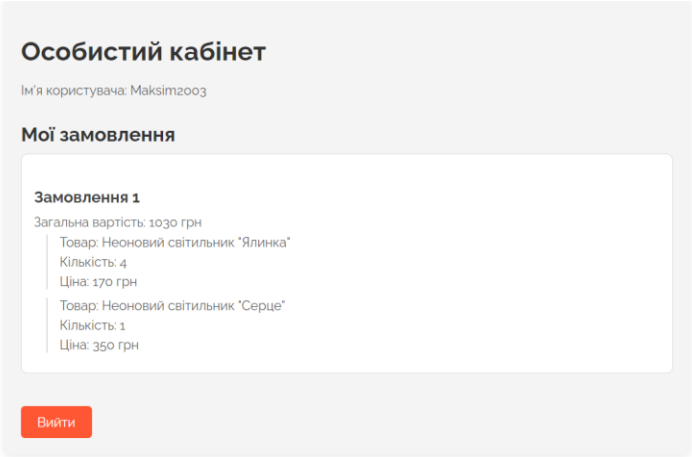

Рис. 2.30. Замовлення в особистому кабінеті

Опис адміністративної частини

Адміністративна частина складається з двох форм: для додавання товарів (рис. 2.31 – рис. 2.32) та для додавання категорій, фільтрів (рис. 2.33). Також є перелік доданих товарів (рис. 2.34) та категорій, фільтрів (рис. 2.35).

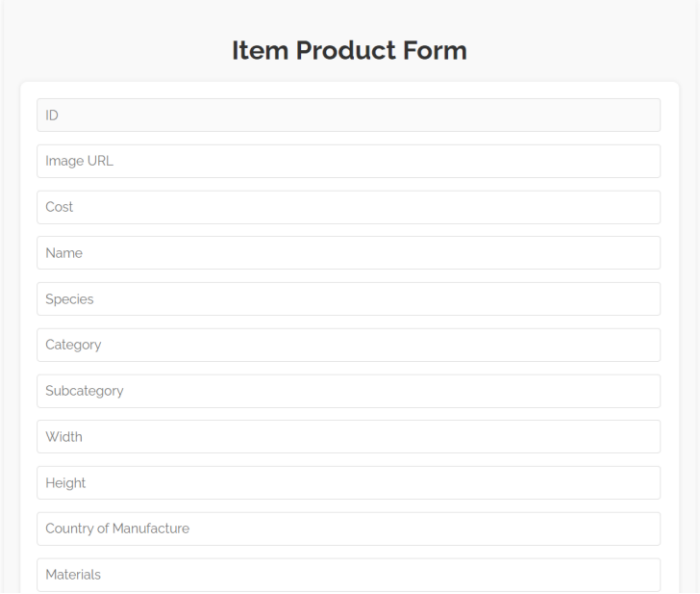

Рис. 2.31. Форма для додавання товарів

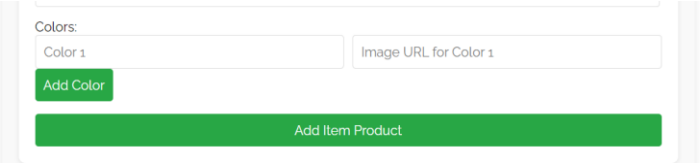

Рис. 2.32. Продовження форми для додавання товарів

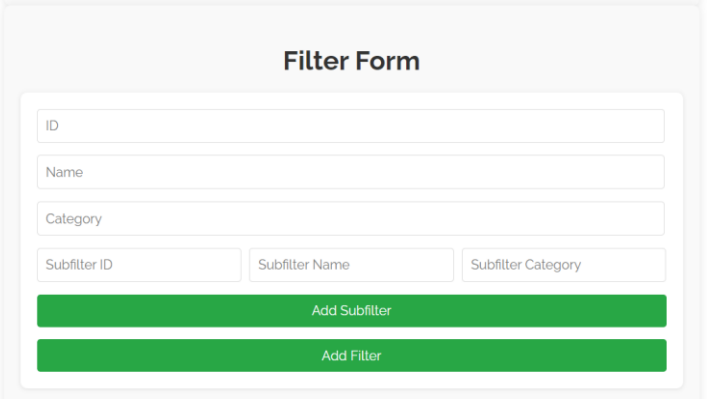

Рис. 2.33. Форма для додавання категорій, фільтрів

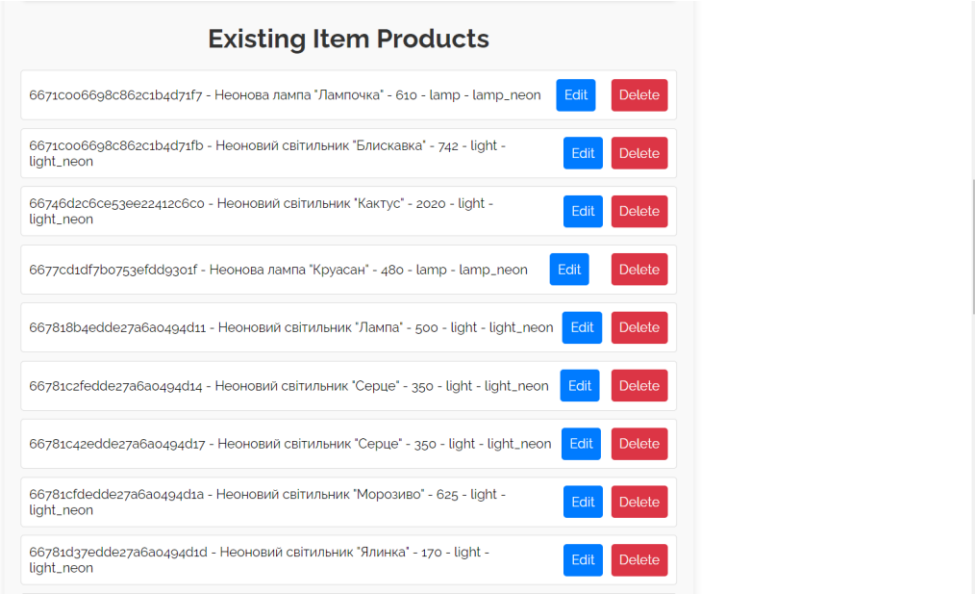

Рис. 2.34. Перелік товарів

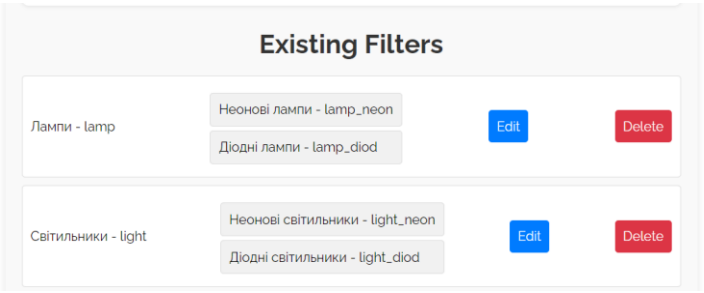

Рис. 2.35. Перелік категорій фільтрів

Натиснувши кнопку «Delete», елемент видаляється. Натиснувши кнопку, яка містить слово «Add», елемент додається. Натиснувши кнопку «Edit», активується режим редагування, поля форми заповнюються інформацією відповідного товару (рис. 2.36)

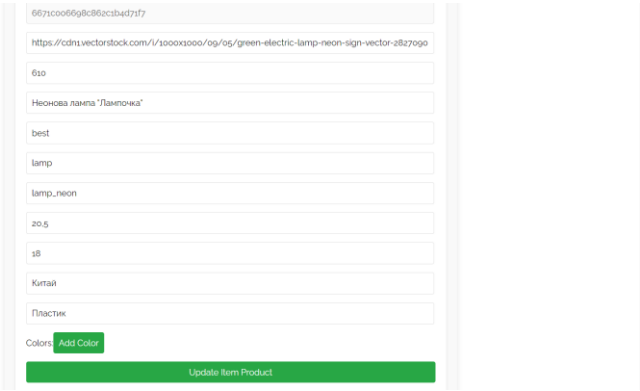

Рис. 2.36. Режим редагування

Змінивши якісь дані та натиснувши кнопку «Update Item Product», редагується обраний елемент. Таким же чином редагуються категорії, фільтри.

Висновки за розділом

У другому розділі «ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ» було розглянуто всі необхідні аспекти проектування та розробки програмного продукту. Від функціонального призначення до опису розробленого програмного продукту, кожен підрозділ зосереджувався на ключових аспектах, що визначають якість та ефективність програми. Детально описані функції програми, використані математичні методи, архітектура та шаблони проектування, технології та мови програмування, структура програми та алгоритми її функціонування, а також організація вхідних та вихідних даних. Підрозділ, присвячений опису розробленого програмного продукту, надає повну інформацію про продукт, включаючи використані технічні та програмні засоби, способи виклику та завантаження програми, а також інтерфейс користувача. Цей розділ є ключовим у процесі створення програмного продукту, оскільки він встановлює основні принципи та параметри, які будуть визначати подальшу розробку та функціонування програми.

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

3.1 Розрахунок трудомісткості та вартості розробки програмного продукту

Початкові дані:

1) Передбачуване число операторів програми: 1984;

2) Коефіцієнт складності програми: 1,4;

3) Коефіцієнт корекції програми в ході розробки: 0,2;

4) Годинна заробітна плата розробника – 241,03 ₴/год.

Згідно зі статистикою онлайн-ресурсу «DOU»[19], присвяченому IT-індустрії, середня місячна зарплата Junior Software Engineer (SE) в Україні на червень 2023 року становить 1000 доларів США. Станом на 21 червня 2024 року, офіційний обмінний курс Національного банку України[20] становить 40,51 гривень за 1 долар США. Зазвичай, програмісти працюють за графіком з п’ятиденним робочим тижнем і восьмигодинним робочим днем.

Погодинну заробітну плату програміста можна розрахувати наступним чином: 1000 доларів США ділимо на (21 робочий день * 8 годин), що дорівнює 5,95 доларів США на годину. В гривнях це становить 5,95 доларів США * 40,51 гривень за долар, що дорівнює 241,03 гривень на годину.

5) Коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі: 1,3;

6) Коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності: 1;

7) Вартість машино-години ЕОМ: 3,5 грн/год;

Інтернет-магазин, що розроблявся в межах даної кваліфікаційної роботи, було створено на ноутбуці. Середнє споживання енергії сучасного ноутбука з 15-дюймовим екраном становить близько 60 Ватт на годину, що призводить до витрат на електроенергію в розмірі: 0,06 кВт * 4,32 грн/кВт·год = 0,26 грн/год.

Станом на 15 червня 2024 року, відповідно до даних сайту “Yasno”[21], вартість електроенергії становить 4,32 гривні за кіловат-годину.

Витрати на інтернет за тарифом інтернет-провайдера «Київстар» становлять 300 гривень на місяць. Загальна вартість машино-години електронно-обчислювальної машини (ЕОМ) визначена як 3,5 грн/год.

Нормування праці в процесі створення ПЗ істотно ускладнено в силу творчого характеру праці програміста. Тому трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_o + t_u + t_a + t_n + t_{отл} + t_д \text{, людино-годин,} \qquad (3.1)$$

де t_o- витрати праці на підготовку й опис поставленої задачі (приймається 50 людино-годин);

t_u - витрати праці на дослідження алгоритму рішення задачі;

t_a- витрати праці на розробку блок-схеми алгоритму;

t_n-витрати праці на програмування по готовій блок-схемі;

$t_{отл}$-витрати праці на налагодження програми на ЕОМ;

$t_д$ - витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у програмному забезпеченні, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q \cdot C \cdot (1 + p) \qquad (3.2)$$

де q - передбачуване число операторів;

C - коефіцієнт складності програми;

p - коефіцієнт корекції програми в ході її розробки.

Звертаючись до формули (3.2), розрахуємо умовне число операторів (підпрограм):

$$Q = 1984 * 1,4 * (1 + 0,2) = 3333,12$$

Витрати праці на вивчення опису задачі t_u визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{Q \cdot B}{(75..85) \cdot k} \text{ , людино-годин} \qquad (3.3)$$

де B - коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі;

k - коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності.

Звертаючись до формули (3.3), розрахуємо значення витрат праці на вивчення опису задачі t_u:

t_u= (3333,12 * 1,2) / 83 * 1 = 48,19 людино-годин

Витрати праці на розробку алгоритму рішення задачі визначаються так:

$$t_a = \frac{Q}{(20\ldots25) \cdot k} \text{ , людино-годин} \qquad (3.4)$$

де Q – умовне число операторів програми;

k – коефіцієнт кваліфікації програміста.

Звертаючись до формули (3.4), розрахуємо значення витрат праці на розробку алгоритму рішення задачі:

t_a= 3333,12 / (20 * 1) = 166,65 людино-годин.

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20\ldots25)\cdot k} \text{ , людино-годин} \qquad (3.5)$$

Звертаючись до формули (3.5), розрахуємо значення витрат на складання програми по готовій блок-схемі:

t_n = 3333,12 / (25 * 1) = 133,32 людино-годин.

Витрати праці на налагодження програми на ЕОМ:

- за умови автономного налагодження одного завдання:

$$t_{отл} = \frac{Q}{(4..5)\cdot k} \text{ , людино-годин} \qquad (3.6)$$

Звертаючись до формули (3.6), розрахуємо значення витрат праці на налагодження програми на ЕОМ за умови автономного налагодження одного завдання

$t_{отл}$ = 3333,12 / (4*1) = 833,28 людино-годин.

- за умови комплексного налагодження завдання:

$$t_{отл}^{k} = 1{,}5 \cdot t_{отл} \text{ , людино-годин} \qquad (3.7)$$

Звертаючись до формули (3.7), розрахуємо значення витрат праці на налагодження програми на ЕОМ за умови комплексного налагодження завдання

$t^k_{отл}$= 1,5 · 833,28 = 1249,92 людино-годин.

Витрати праці на підготовку документації визначаються за формулою:

$$t_д = t_{др} + t_{до} \text{ , людино-годин} \quad (3.8)$$

де $t_{др}$-трудомісткість підготовки матеріалів і рукопису:

$$t_{др} = \frac{Q}{(15..20) \cdot k} \text{ , людино-годин} \quad (3.9)$$

$t_{до}$ - трудомісткість редагування, печатки й оформлення документації:

$$t_{до} = 0,75 \cdot t_{др} \text{ , людино-годин.} \quad (3.10)$$

Звертаючись до формул (3.8 – 3.10), розрахуємо наступні значення:

$t_{др}$ = 3333,12 / (15 * 1) = 222,21 людино-годин. (3.9)
$t_{до}$ = 0,75 · 222,21 = 166,65 людино-годин. (3.10)
$t_д$ = 222,21 + 166,65 = 388,86 людино-годин. (3.8)

Звертаючись до формули (3.7), розрахуємо значення повної оцінки трудомісткості розробки програмного забезпечення:

t = 50 + 48,19 + 166,65 + 133,32 + 833,28 + 388,86 = 1620,3 людино-години.

3.2. Розрахунок витрат на створення програмного забезпечення

Витрати на створення ПЗ $К_{ПО}$ включають витрати на заробітну плату виконавця програми $З_{ЗП}$ і витрат машинного часу, необхідного на налагодження програми на ЕОМ:

$$К_{ПО} = З_{ЗП} + З_{МВ}, \text{ грн} \quad (3.11)$$

Заробітна плата виконавців $З_{ЗП}$ визначається за формулою:

$$З_{ЗП} = t \cdot С_{ПР}, \text{ грн} \quad (3.12)$$

де: t - загальна трудомісткість, людино-годин;

$С_{ПР}$ - середня годинна заробітна плата програміста, грн/година

Звертаючись до формули (3.12), розрахуємо заробітну плату виконавців:

$З_{ЗП}$ = 1620,3 * 241,03 = 390 540 грн.

Вартість машинного часу, необхідного для налагодження програми на ЕОМ, визначається за формулою:

$$З_{мв} = t_{отл} \cdot С_{мч}, \text{ грн}, \quad (3.13)$$

де $t_{отл}$ - трудомісткість налагодження програми на ЕОМ, год;

$С_{мч}$ - вартість машино-години ЕОМ, грн/год.

Звертаючись до формули (3.13), розрахуємо значення вартості машинного часу, необхідного для налагодження програми на ЕОМ:

$З_{мв}$ = 833,28 * 3,5 = 2916,48 грн.

Звертаючись до формули (3.11), розрахуємо значення витрат на створення ПЗ:

$K_{ПО}$ = 390540 + 2916,48 = 393456,48 грн.

Звертаючись до формули (3.13), розрахуємо очікуваний період створення ПЗ:

$$T = \frac{t}{B_k \cdot F_p}, \text{ міс.} \quad (3.14)$$

де B_k - число виконавців;

F_p - місячний фонд робочого часу (176 годин).

Очікуваний період створення ПЗ:

T = 1620,3 / (1 * 176) ≈ 9,2 міс.

ВИСНОВКИ

У цій роботі була досягнута основна мета, яка полягала в розробці програмного забезпечення веб-застосунку інтернет-магазину іграшок мовою JavaScript. Розроблене програмне забезпечення дозволяє клієнтам зручно купувати товари, а адміністраторам налаштовувати контент сайту.

Результатом роботи стало створення повнофункціонального веб-додатку, що дозволяє клієнтам купувати іграшки з будь-якої точки світу. Для цього був розроблений інтерфейс клієнтської частини за допомогою бібліотеки React у поєднанні з HTML та CSS, що забезпечує зручність та швидкість використання. Серверна частина веб-додатку була розроблена на основі веб-фреймворку Express.js та Mongoose, що забезпечує надійне управління даними та взаємодію з базою даних. Місцем зберігання даних стала база даних MongoDB, яка забезпечує швидкий доступ до інформації та високу масштабованість.

Окрім цього, веб-додаток має адміністративну частину, яка дозволяє адміністраторам легко заповнювати каталог товарів, додавати нові фільтри та керувати контентом сайту. Це значно спрощує процес адміністрування та дозволяє швидко реагувати на потреби ринку.

У ході розробки веб-додатку інтернет магазину іграшок були використані такі технології: інтерфейс клієнтської частини розроблений за допомогою бібліотеки React, у зв’язці з HTML та CSS; серверна частина розроблена за допомогою веб-фреймворку Express.js у зв’язці з Mongoose; місцем зберігання даних є БД MongoDB.

У ході написання кваліфікаційної роботи було установлено, що час для створення веб-додатку інтернет магазину, при робочому графіку 5/2, становить приблизно 9,2 місяці. При цьому було встановлено, що витрати для розробки програми сягають 390 843,1 грн. при заробітній платі 241,03 ₴/год та трудомісткості розробки 1620,3 людино-години

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Middleware Express.js. URL: https://expressjs.com/uk/guide/writing-middleware.html (дата звернення: 05.06.2024).
2. What is model-view-controller (MVC). URL: https://www.techtarget.com/whatis/definition/model-view-controller-MVC (дата звернення: 05.06.2024).
3. Top 8 Most Demanded Programming Languages in 2023. URL: https://www.devjobsscanner.com/blog/top-8-most-demanded-programming-languages/ (дата звернення: 05.06.2024).
4. Kyle Simpson. You Don't Know JS: Up & Going. — O'Reilly Media, 2015. — 30 с.
5. React Docs. URL: https://legacy.reactjs.org/docs/getting-started.html (дата звернення: 06.06.2024).
6. Express.js інструкція. URL: https://expressjs.com/ru/guide/routing.html (дата звернення: 06.06.2024).
7. Redux.js tutorial. URL: https://redux.js.org/tutorials/essentials/part-1-overview-concepts (дата звернення: 06.06.2024).
8. Slick Carousel. URL: https://htmlstream.com/preview/front-v2.1.1/documentation/plugins/slick-carousel.html (дата звернення: 07.06.2024).
9. React Slick. URL: https://codesandbox.io/examples/package/react-slick (дата звернення: 07.06.2024).
10. Styled-components. URL: https://styled-components.com/docs (дата звернення: 07.06.2024).
11. Axios docs. URL: https://axios-http.com/ru/docs/intro (дата звернення: 05.06.2024).
12. Mongoose - elegant mongodb object modeling for node.js. URL: https://mongoosejs.com/ (дата звернення: 07.06.2024).

13. MongoDB documentation. URL: https://www.mongodb.com/solutions/developer-data-platform (дата звернення: 08.06.2024).
14. VS Code docs. URL: https://code.visualstudio.com/docs (дата звернення: 08.06.2024).
15. MongoDB Atlas docs. URL: https://www.mongodb.com/docs/atlas/getting-started/ (дата звернення: 08.06.2024).
16. The MongoDB Database Tools Documentation. URL: https://www.mongodb.com/docs/database-tools/ (дата звернення: 08.06.2024).
17. About Node.js. URL: https://nodejs.org/en/about (дата звернення: 09.06.2024).
18. Figma guide. URL: https://www.figma.com/ (дата звернення: 09.06.2024).
19. Dou. URL: https://dou.ua/lenta/articles/salary-report-devs-summer-2023/ (дата звернення: 09.06.2024).
20. НБУ. URL: https://bank.gov.ua/ (дата звернення: 09.06.2024).
21. Yasno. URL: https://yasno.com.ua/ (дата звернення: 09.06.2024).

ДОДАТОК А

Код програми

Server.js // Серверна частина програми

```
// Підключення бібліотек
const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');
// Створення додатку express
const app = express();
const secret = 'jwt_secret';

// Налаштування використання JSON та CORS
app.use(express.json());
app.use(cors());

// Підключення до MongoDB
mongoose.connect('mongodb+srv://seksikoleg5:se4HivNRYKdydnzc@cluster0.pdc2rrh.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0', {
    useNewUrlParser: true,
    useUnifiedTopology: true,
});

// Повідомлення про успішне підключення до MongoDB
mongoose.connection.once('open', () => {
    console.log('Connected to MongoDB');
});

// Схема для продуктів
const itemProductSchema = new mongoose.Schema({
    id: String,
    img: String,
    cost: String,
    name: String,
    species: String,
    category: String,
    subcategory: String,
    colors: [{ color: String, image: String }],
    colorImages: { type: mongoose.Schema.Types.Mixed },
    width: String,
    height: String,
    countryOfManufacture: String,
    materials: String,
    reviews: [{ username: String, rating: Number, text: String }]
});

// Схема для фільтрів
const filterSchema = new mongoose.Schema({
    id: String,
    name: String,
    category: String,
    subfilters: [{
        id: String,
        filterName: String,
        name: String,
    }],
});

// Схема для користувачів
const userSchema = new mongoose.Schema({
```

```
    username: { type: String, required: true, unique: true },
    email: { type: String, required: true, unique: true },
    phone: { type: String, unique: true },
    password: { type: String, required: true },
    orders: [{ type: mongoose.Schema.Types.ObjectId, ref: 'Order' }]
});

// Схема для замовлень
const orderSchema = new mongoose.Schema({
    userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
    items: [{ item: { type: mongoose.Schema.Types.ObjectId, ref: 'ItemProduct' }, quantity: Number, name: String,
cost: Number }],
    totalCost: Number,
    status: { type: String, default: 'Pending' }
});

const ItemProduct = mongoose.model('ItemProduct', itemProductSchema);
const Filter = mongoose.model('Filter', filterSchema);
const User = mongoose.model('User', userSchema);
const Order = mongoose.model('Order', orderSchema);

// Маршрут для отримання всіх продуктів
app.get('/itemProducts', async (req, res) => {
    try {
        const items = await ItemProduct.find();
        res.json(items);

    } catch (error) {
        console.error('Error fetching item products:', error);
        res.status(500).json({ error: 'Internal server error' });
    }
});

// Маршрут для додавання нового продукту
app.post('/itemProducts', async (req, res) => {
    try {
        const newItem = await ItemProduct.create(req.body);
        res.status(201).json(newItem);
    } catch (error) {
        console.error('Error creating item product:', error);
        res.status(500).json({ error: 'Internal server error' });
    }
});

// Маршрут для оновлення продукту за його ID
app.put('/itemProducts/:id', async (req, res) => {
    const { id } = req.params;
    console.log(id)
    const updatedItemData = req.body;
    try {
        if (!mongoose.Types.ObjectId.isValid(id)) {
            return res.status(400).json({ error: 'Invalid ObjectId' });
        }

        const updatedItem = await ItemProduct.findByIdAndUpdate(id, updatedItemData, { new: true });
        console.log(updatedItem)
        if (!updatedItem) {
            return res.status(404).json({ error: 'Item product not found' });
        }

        res.json(updatedItem);
    } catch (error) {
        console.error('Error updating item product:', error);
```

```
        res.status(500).json({ error: 'Internal server error' });
    }
});

// Маршрут для видалення продукту за його ID
app.delete('/itemProducts/:id', async (req, res) => {
    const { id } = req.params;
    try {
        const deletedItem = await ItemProduct.findByIdAndDelete(id);
        if (!deletedItem) {
            return res.status(404).json({ error: 'Item product not found' });
        }
        res.json(deletedItem);
    } catch (error) {
        console.error('Error deleting item product:', error);
        res.status(500).json({ error: 'Internal server error' });
    }
});

// Новий маршрут для отримання відгуків до продукту за його ID
app.get('/itemProducts/:id/reviews', async (req, res) => {
    const { id } = req.params;
    console.log(id)
    try {
        const item = await ItemProduct.findById(id);
        if (!item) {
            return res.status(404).json({ error: 'Item product not found' });
        }
        res.json(item.reviews);
    } catch (error) {
        console.error('Error fetching reviews:', error);
        res.status(500).json({ error: 'Internal server error' });
    }
});

// Новий маршрут для додавання відгуку до продукту за його ID
app.post('/itemProducts/:id/reviews', async (req, res) => {
    const { id } = req.params;
    const { username, rating, text } = req.body;

    // Перевірка наявності всіх необхідних полів
    if (!username || !rating || !text) {
        return res.status(400).json({ error: 'Missing required fields: username, rating, and text' });
    }

    try {
        if (!mongoose.Types.ObjectId.isValid(id)) {
            return res.status(400).json({ error: 'Invalid ObjectId' });
        }

        const item = await ItemProduct.findById(id);
        if (!item) {
            return res.status(404).json({ error: 'Item product not found' });
        }

        const newReview = { username, rating, text };
        item.reviews.push(newReview);
        await item.save();

        res.status(201).json(newReview);
    } catch (error) {
        console.error('Error creating review:', error);
        res.status(500).json({ error: 'Internal server error' });
```

```
    }
});

// Маршрут для отримання всіх фільтрів
app.get('/filters', async (req, res) => {
    try {
        const filters = await Filter.find();
        res.json(filters);
    } catch (error) {
        console.error('Error fetching filters:', error);
        res.status(500).json({ error: 'Internal server error' });
    }
});

// Маршрут для додавання нового фільтру
app.post('/filters', async (req, res) => {
    try {
        const newFilter = new Filter(req.body);
        const savedFilter = await newFilter.save();
        res.status(201).json(savedFilter);
    } catch (error) {
        console.error('Error saving filter:', error);
        res.status(500).json({ error: 'Internal server error' });
    }
});

// Маршрут для оновлення фільтру за його ID
app.put('/filters/:id', async (req, res) => {
    const { id } = req.params;
    try {
        const updatedFilter = await Filter.findByIdAndUpdate(id, req.body, { new: true });
        if (!updatedFilter) {
            return res.status(404).json({ error: 'Filter not found' });
        }
        res.json(updatedFilter);
    } catch (error) {
        console.error('Error updating filter:', error);
        res.status(500).json({ error: 'Internal server error' });
    }
});

// Маршрут для видалення фільтру за його ID
app.delete('/filters/:id', async (req, res) => {
    const { id } = req.params;
    try {
        const deletedFilter = await Filter.findByIdAndDelete(id);
        if (!deletedFilter) {
            return res.status(404).json({ error: 'Filter not found' });
        }
        res.json(deletedFilter);
    } catch (error) {
        console.error('Error deleting filter:', error);
        res.status(500).json({ error: 'Internal server error' });
    }
});

// Маршрут для виходу з системи
app.post('/logout', (req, res) => {
    res.clearCookie('token').json({ message: 'Logged out successfully' });
});

// Маршрут для реєстрації нового користувача
app.post('/register', async (req, res) => {
```

```
    const { username, email, password } = req.body;

    try {
        if (!email) {
            return res.status(400).json({ error: 'Email is required' });
        }

        const existingUser = await User.findOne({ email });
        if (existingUser) {
            return res.status(400).json({ error: 'User with this email already exists' });
        }

        const hashedPassword = await bcrypt.hash(password, 10);
        const newUser = new User({ username, email, password: hashedPassword });
        await newUser.save();

        res.status(201).json({ message: 'User registered successfully' });
    } catch (error) {
        console.error('Error registering user:', error);
        res.status(500).json({ error: 'Internal server error' });
    }
});

// Маршрут для входу користувача
app.post('/login', async (req, res) => {
    const { username, password } = req.body;

    try {
        const user = await User.findOne({ username });
        if (!user) {
            return res.status(400).json({ error: 'Invalid credentials' });
        }

        const isMatch = await bcrypt.compare(password, user.password);
        if (!isMatch) {
            return res.status(400).json({ error: 'Invalid credentials' });
        }

        const token = jwt.sign({ userId: user._id }, secret, { expiresIn: '1h' });

        res.cookie('token', token, { httpOnly: true });

        res.json({ token });
    } catch (error) {
        console.error('Error logging in:', error);
        res.status(500).json({ error: 'Internal server error' });
    }
});

// Мідлвар для авторизації
const auth = (req, res, next) => {
    const token = req.header('Authorization').replace('Bearer ', '');

    if (!token) {
        return res.status(401).json({ error: 'No token, authorization denied' });
    }

    try {
        const decoded = jwt.verify(token, secret);
        req.user = decoded;
        next();
    } catch (error) {
        res.status(401).json({ error: 'Token is not valid' });
```

```
    }
};

// Маршрут для отримання інформації про поточного користувача
app.get('/me', auth, async (req, res) => {
    try {
        const user = await User.findById(req.user.userId).populate('orders');
        res.json(user);
    } catch (error) {
        console.error('Error fetching user data:', error);
        res.status(500).json({ error: 'Internal server error' });
    }
});

// Маршрут для створення нового замовлення
app.post('/orders', auth, async (req, res) => {
    const { items, totalCost } = req.body;

    try {
        const newOrder = new Order({
            userId: req.user.userId,
            items,
            totalCost
        });

        await newOrder.save();
        await User.findByIdAndUpdate(req.user.userId, { $push: { orders: newOrder._id } });

        res.status(201).json(newOrder);
    } catch (error) {
        console.error('Error creating order:', error);
        res.status(500).json({ error: 'Internal server error' });
    }
});

// Запуск сервера
const PORT = process.env.PORT || 5000;
app.listen(PORT, () => {
    console.log(`Server is running on port ${PORT}`);
});
```

App.jsx // Головний модуль програми

```
// Імпорт бібліотек та ресурсів
import MainPage from "../pages/mainPage/mainPage";
import {ItemPage, SearchItemPage} from "../common/layout/itemPage/itemPage";
import SubFilterPage from "../pages/subfilterPage/subfilterPage";
import FilterPage from "../pages/filterPage/filterPage";
import AdminPage from "../pages/adminPage/adminPage";
import UserProfile from "../pages/regPages/UserProfile";
import { BrowserRouter as Router, Route, Routes } from 'react-router-dom';

function App() {
  return (
    // Навігація по сторінках
    <Router>
      <Routes>
        <Routes>
          // Головна сторінка
          <Route exact path="/" element={<MainPage/>}></Route>
          // Сторінка адміністратора
          <Route exact path="/admin" element={<AdminPage/>}></Route>
          // Сторінка продукту
```

```
        <Route exact path="/itemPage/:productId" element={<ItemPage/>}></Route>
        // Сторінка пошуку продукту
        <Route exact path="/itemSearchPage/:productId" element={<SearchItemPage/>}></Route>
        // Сторінка підфільтру
        <Route exact path="/itemFilterPage/:filter" element={<SubFilterPage/>}></Route>
        // Сторінка категорії
        <Route exact path="/itemCategoryPage/:category" element={<FilterPage/>}></Route>
        // Сторінка профілю користувача
        <Route exact path="/cabinet" element={<UserProfile/>}></Route>
      </Routes>    </Router>
  );
}

export default App;
```

backet.jsx // Корзина

```
import React, { useState } from 'react';
import { useSelector, useDispatch } from 'react-redux';
import { itemsFetched, removeItem } from '../../../actions';
import OrderModal from './OrderModal';
import "./backet.css";

import CartLogo from '../../../resources/img/shopping-bag.svg';

const Backet = () => {
    // Використання хука useState для управління станами ховера і відкриття модального вікна
    const [isHovered, setIsHovered] = useState(false);
    const [isModalOpen, setIsModalOpen] = useState(false);

    // Використання хука useSelector для отримання елементів з глобального стану Redux
    const { items } = useSelector((state) => state);
    const dispatch = useDispatch();

    // Функції для управління ховером на іконці кошика
    const handleMouseEnter = () => {
        setIsHovered(true);
    };

    const handleMouseLeave = () => {
        setIsHovered(false);
    };

    // Функція для очищення кошика
    const clearBasket = () => {
        dispatch(itemsFetched([]));
        localStorage.clear();
    };

    // Функція для видалення елемента з кошика
    const handleRemove = (id) => {
        const updatedCart = items.filter((item) => item.id !== id);
        localStorage.setItem('cart', JSON.stringify(updatedCart));
        dispatch(itemsFetched(updatedCart));
    };

    // Функції для відкриття і закриття модального вікна оформлення замовлення
    const openModal = () => {
        setIsModalOpen(true);
    };

    const closeModal = () => {
        setIsModalOpen(false);
    };
```

```
    // Функція для отримання назви кольору за його кодом
    const getColorName = (color) => {
        switch (color) {
            case '#FF5733':
                return 'ЧЕРВОНИЙ';
            case '#33FF57':
                return 'ЗЕЛЕНИЙ';
            case '#3357FF':
                return 'СИНІЙ';
            default:
                return 'КОЛІР НЕ ОБРАНО';
        }
    };

    return (
        <div>
            <div>
                <img src={CartLogo} alt="" onMouseEnter={handleMouseEnter} />
            </div>
            {isHovered && (
                <div className="cart_content-block" onMouseLeave={handleMouseLeave}>
                    <div className="cart-elements">
                        {/* Відображення елементів кошика */}
                        {items.map((item, index) => (
                            <div className="cart-element" key={index}>
                                <div>{item.name}</div>
                                <div>{getColorName(item.color)}</div>
                                <div>{item.quantity}</div>
                                <div>{item.totalCost} грн</div>
                                <img className="small-img" src={item.img} alt={item.name} />
                                <button onClick={() => handleRemove(item.id)}>Видалити</button>
                            </div>
                        ))}
                        {/* Повідомлення, якщо кошик порожній */}
                        {items.length === 0 ? <div className="cart-null">Кошик порожній</div> : null}
                    </div>
                    {/* Кнопки для очищення кошика і оформлення замовлення */}
                    <button className="clear-cart" onClick={clearBasket}>
                        Очистити кошик
                    </button>
                    <button className="checkout-button" onClick={openModal}>
                        Оформити замовлення
                    </button>
                </div>
            )}
            {/* Модальне вікно оформлення замовлення */}
            {isModalOpen && <OrderModal isOpen={isModalOpen} onClose={closeModal} items={items} />}
        </div>
    );
};

export default Backet;
```

OrderModal.jsx // Модальне вікно замовлення

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';
import "./backet.css";

// Компонент OrderModal для оформлення замовлення
const OrderModal = ({ isOpen, onClose, items }) => {
    const [user, setUser] = useState(null); // Стан для зберігання даних користувача
    const [totalCost, setTotalCost] = useState(0); // Стан для зберігання загальної вартості замовлення
```

```
    // Використання useEffect для завантаження даних користувача та обчислення загальної вартості
    useEffect(() => {
      const fetchUserData = async () => {
        const token = localStorage.getItem('token');
        if (token) {
          try {
            const response = await axios.get('/me', {
              headers: { 'Authorization': `Bearer ${token}` }
            });
            setUser(response.data); // Встановлення даних користувача у стан
          } catch (error) {
            console.error('Error fetching user data:', error);
          }
        }
      };

      const calculateTotalCost = () => {
        const cost = items.reduce((acc, item) => acc + item.quantity * item.cost, 0);
        setTotalCost(cost); // Обчислення загальної вартості замовлення
      };

      fetchUserData();
      calculateTotalCost();
    }, [items]);

    // Обробник для подання замовлення
    const handleSubmit = async (e) => {
      e.preventDefault();
      const token = localStorage.getItem('token');
      if (token) {
        try {
          await axios.post('/orders',
            { items, totalCost },
            { headers: { 'Authorization': `Bearer ${token}` } }
          );
          onClose(); // Закриття модального вікна після успішного подання замовлення
        } catch (error) {
          console.error('Error creating order:', error);
        }
      } else {
        alert('Пожалуйста, войдите в систему для оформления заказа'); // Повідомлення, якщо користувач не авторизований
      }
    };

    // Перевірка, чи модальне вікно відкрите
    if (!isOpen) {
      return null;
    }

    return (
      <div className="modal">
        <div className="modal-content">
          <span className="close" onClick={onClose}>&times;</span> {/* Кнопка закриття модального вікна */}
          <h2>Оформлення замовлення</h2>
          <form onSubmit={handleSubmit}>
            {user && (
              <>
                <p>Имя пользователя: {user.username}</p> {/* Відображення імені користувача */}
              </>
            )}
```

```
                        <h3>Товари</h3>
                        {items.map((item, index) => (
                            <div key={index}>
                                <p>{item.name} x {item.quantity} - {item.cost} грн</p> {/* Відображення деталей товарів у замовленні */}
                            </div>
                        ))}
                        <p>Загальна вартість: {totalCost} грн</p> {/* Відображення загальної вартості замовлення */}
                        <button className='order-btn' type="submit">Оформити замовлення</button> {/* Кнопка подання замовлення */}
                    </form>
                </div>
            </div>
        );
    };

    export default OrderModal;
```

footer.jsx // «Футер» сайту

```
    import LogoImg from "../../../resources/img/logo.png"
    import "./footer.css"

    const Footer = () => {
        return (
            <div className="footer-container">
                <div className="container">
                    <div className="footer-content">
                        <div className="footer-cols">
                            <div className="footer-cols-header">
                                <img src={LogoImg} alt="" />
                            </div>
                            <div className="footer-cols-elements">

                                <div className="footer-cols-element flex">
                                    <img src="" alt="" />
                                    <span>098-254-31-26</span>
                                </div>
                                <div className="footer-cols-element flex">
                                    <img src="" alt="" />
                                    <span>097-652-33-78</span>
                                </div>
                                <div className="footer-cols-element flex">
                                    <img src="" alt="" />
                                    <span>info@mimi_shop.ua</span>
                                </div>
                            </div>
                        </div>
                        <div className="footer-cols">
                            <div className="footer-cols-header">
                                <h3>
                                    ІНФОРМАЦІЯ
                                </h3>
                            </div>
                            <div className="footer-cols-elements">
                                <div className="footer-cols-element">
                                    Поширенні запитання
                                </div>
                                <div className="footer-cols-element">
                                    Відгуки про магазин
                                </div>
                            </div>
                        </div>
```

```
                    <div className="footer-cols">
                        <div className="footer-cols-header">
                            <h3>
                                СПІВПРАЦЯ
                            </h3>
                        </div>
                        <div className="footer-cols-elements">

                            <div className="footer-cols-element">
                                Запитання щодо співпраці
                            </div>
                            <div className="footer-cols-element">
                                Про нас
                            </div>

                        </div>
                    </div>
                    <div className="footer-cols">
                        <div className="footer-cols-header">
                            <h3>
                                ДОСТАВКА ТА ОПЛАТА
                            </h3>
                        </div>
                        <div className="footer-cols-elements">
                            <div className="footer-cols-element">
                                Публічна оферта
                            </div>
                            <div className="footer-cols-element">
                                Повернення товару
                            </div>
                            <div className="footer-cols-element">
                                Доставка та оплата
                            </div>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    )
}

export default Footer;
```

header.jsx // «Хедер» сайту

```
import SearchProducts from "../search/search";
import Backet from "../basket/backet";
import Filters from "../../filters/filters";
import LogoImg from "../../../resources/img/logo.png";
import PhoneImg from "../../../resources/img/phone-call.svg";
import { Link, useNavigate } from "react-router-dom";
import { useState } from "react";
import Modal from "../modal/modal";
import Login from "../../pages/regPages/Login";
import Registration from "../../pages/regPages/Registration";
import CabinetImg from "../../../resources/img/cabinet.png";
import './header.css';

const Header = () => {
    const [isModalOpen, setIsModalOpen] = useState(false); // Стан для відстеження відкриття модального
вікна
    const [activeTab, setActiveTab] = useState('login'); // Стан для відстеження активної вкладки модального
вікна
    const navigate = useNavigate(); // Хук для навігації
```

```
const openModal = () => {
  setIsModalOpen(true); // Відкриття модального вікна
}

const closeModal = () => {
  setIsModalOpen(false); // Закриття модального вікна
}

const handlePersonalCabinetClick = () => {
  const token = localStorage.getItem('token'); // Отримання токена з localStorage
  if (token) {
    navigate('/cabinet'); // Навігація до особистого кабінету, якщо користувач авторизований
  } else {
    openModal(); // Відкриття модального вікна для входу або реєстрації
  }
}

return (
  <div className="header-container">
    <div className="info-header">
      <div className="container">
        <ul className="pagesList">
          <li>Про нас</li>
          <li>Доставка</li>
          <li>Оплата</li>
          <li>Повернення товару</li>
          <li>Контакти</li>
        </ul>
      </div>
    </div>
    <div className="main-header">
      <div className="container">
        <div className="logo-block">
          <Link to={'/'}>
            <img className="logo-img" src={LogoImg} alt="Logo" /> {/* Логотип */}
          </Link>
        </div>
        <div className="search-block">
          <SearchProducts /> {/* Компонент пошуку */}
        </div>
        <div className="tell_me-block">
          <div className="tell_me">
            <img className="tell_me-img" src={PhoneImg} alt="Phone" /> {/* Іконка телефону */}
            <a href="tel:+380982543126" className="tell_me-txt">
              <span>Зателефонуйте нам!</span>
            </a>
          </div>
        </div>
        <div className="reg-btn-container">
          <div className="reg-btn" onClick={openModal}>Вхід / Реєстрація</div> {/* Кнопка входу
/ реєстрації */}
          <Modal isOpen={isModalOpen} onClose={closeModal}>
            <div className="tabs">
              <button
                className={activeTab === 'login' ? 'active' : ''}
                onClick={() => setActiveTab('login')}
              >
                Login
              </button>
              <button
                className={activeTab === 'register' ? 'active' : ''}
                onClick={() => setActiveTab('register')}
```

```
                        >
                            Register
                        </button>
                    </div>
                    {activeTab === 'login' ? <Login /> : <Registration />} {/* Компоненти входу / реєстрації */}
                </Modal>
            </div>
            <div style={{ display: "flex", alignItems: "center" }}>
                <div onClick={handlePersonalCabinetClick} style={{ cursor: 'pointer' }}>
                    <img className='cabinet-img' src={CabinetImg} alt="Cabinet" /> {/* Іконка особистого кабінету */}
                </div>
            </div>
            <div className="cart-block">
                <Backet /> {/* Компонент кошика */}
            </div>
        </div>
    </div>
    <div className="menu-header">
        <Filters /> {/* Компонент фільтрів */}
    </div>
</div>
);
}

export default Header;
```

itemPage.jsx // Сторінка товару

```
import ItemProduct from "../itemProduct/ItemProduct";
import { useLocation } from 'react-router-dom'; // Імпорт хука, який дозволяє отримувати доступ до поточного об'єкта локації (URL)
import Header from "../../header/header";
import Footer from "../../footer/footer";
import MainProduct from "../../../mainProduct/mainProduct";
import { useState, useEffect } from "react";
import './tabenu.css';

const ItemPage = () => { // Оголошення компоненту ItemPage
    const location = useLocation(); // Отримання поточної локації
    const id = location.state?.id; // Отримання ідентифікатора товару з локації
    const img = location.state?.img; // Отримання зображення товару з локації
    const cost = location.state?.cost; // Отримання ціни товару з локації
    const name = location.state?.name; // Отримання назви товару з локації
    const width = location.state?.width; // Отримання ширини товару з локації
    const height = location.state?.height; // Отримання висоти товару з локації
    const countryOfManufacture = location.state?.countryOfManufacture; // Отримання країни виробника з локації
    const materials = location.state?.materials; // Отримання матеріалів товару з локації

    // Стан для активної вкладки та відгуків про товар
    const [activeTab, setActiveTab] = useState('description');
    const [reviews, setReviews] = useState([]);
    const [newReview, setNewReview] = useState({ username: '', rating: '', text: '' });

    // Ефект для отримання відгуків про товар при завантаженні сторінки
    useEffect(() => {
        const fetchReviews = async () => {
            try {
                const response = await fetch(`/itemProducts/${id}/reviews`);
                if (response.ok) {
                    const data = await response.json();
                    setReviews(data);
```

```
            } else {
                console.error('Failed to fetch reviews:', response.statusText);
            }
        } catch (error) {
            console.error('Error fetching reviews:', error);
        }
    };

    fetchReviews();
}, [id]);

// Функція для зміни вмісту вкладок
const renderContent = () => {
    switch (activeTab) {
        case 'description':
            return (
                <div>
                    <p>Ширина: <span className="detail">{width} см</span></p>
                    <p>Висота: <span className="detail">{height} см</span></p>
                    <p>Країна-виробник: <span className="detail">{countryOfManufacture}</span></p>
                    <p>Матеріали: <span className="detail">{materials}</span></p>
                </div>
            );
        case 'delivery':
            return <p>Деталі про доставку та оплату уточнюйте у менеджера</p>;
        case 'reviews':
            return (
                <div className="reviews-tab">
                    <h3>Відгуки про товар</h3>
                    {reviews.length > 0 ? (
                        <ul className="reviews-list">
                            {reviews.map((review, index) => (
                                <li key={index} className="review-item">
                                    <strong>{review.username}</strong> ({review.rating}): {review.text}
                                </li>
                            ))}
                        </ul>
                    ) : (
                        <p>Немає відгуків.</p>
                    )}
                    <h4>Написати відгук</h4>
                    <div className="review-form">
                        <input
                            type="text"
                            name="username"
                            value={newReview.username}
                            onChange={handleReviewChange}
                            placeholder="Ваше ім'я"
                        />
                        <input
                            type="number"
                            name="rating"
                            value={newReview.rating}
                            onChange={handleReviewChange}
                            placeholder="Рейтинг (1-5)"
                        />
                        <textarea
                            name="text"
                            value={newReview.text}
                            onChange={handleReviewChange}
                            placeholder="Ваш відгук"
                        ></textarea>
                        <button type="button" onClick={addReview}>Додати відгук</button>
```

```
                    </div>
                  </div>
                );
              default:
                return null;
            }
          };

          // Повертаємо JSX компоненту з вмістом сторінки та стилями
          return (
            <div>
              <Header /> {/* Відображення заголовка */}
              <div className="container">
                <ItemProduct id={id} img={img} cost={cost} name={name} customStyles={customStyles} /> {/*
Відображення товару */}
              </div>
              <div>
                {/* Відображення вкладок з вмістом */}
                <div className="tabs-container">
                  <div className="tabs-header">
                    <button
                      className={activeTab === 'description' ? 'active' : ''}
                      onClick={() => setActiveTab('description')}
                    >
                      Опис
                    </button>
                    <button
                      className={activeTab === 'delivery' ? 'active' : ''}
                      onClick={() => setActiveTab('delivery')}
                    >
                      Доставка та оплата
                    </button>
                    <button
                      className={activeTab === 'reviews' ? 'active' : ''}
                      onClick={() => setActiveTab('reviews')}
                    >
                      Відгуки про товар
                    </button>
                  </div>
                  <div className="tabs-content">
                    {renderContent()} {/* Відображення вмісту активної вкладки */}
                  </div>
                </div>
              </div>
              <div className="bests">
                <h1 style={{ textAlign: "center" }}>Рекомендуємо</h1>
                <MainProduct speciec="best" /> {/* Відображення рекомендованих товарів */}
              </div>
              <Footer /> {/* Відображення підвалу */}
            </div>
          );
        };

        export default ItemPage;
```

itemProduct.jsx // Картка товару

```
import React, { useEffect, useState } from 'react';
import { useSelector, useDispatch } from 'react-redux';
import { itemsFetched } from '../../../../actions';
import { Link } from 'react-router-dom';
import "./itemProduct.css"
import styled, { css } from 'styled-components';
import ColorSelector from './colorSelector';
```

```
// Стилізований компонент ItemProduct
const StyledItemProduct = styled.div`
  border: 2px solid #F58DEF;
  background-color: #F8F5FF;
  padding: 10px;
  margin: 10px;
  transition: background-color 0.3s ease, border-color 0.3s ease;
  display: block;
  width: 230px;

  .nameProduct {
    font-weight: bold;
  }

  .costProduct {
    color: #777;
  }

  .leftSide{
  display: flex;
    justify-content: center;}

  .colorSelect {
    display: none;
  }

  .productBtns{
    display: none;
  }

  img {
    width: 200px;
    height: 200px;
    margin: 0 auto;
  }

  button {
    background-color: #5cb85c;
    color: white;
    border: none;
    padding: 10px;
    cursor: pointer;

    &:hover {
      opacity: 0.8;
    }
  }

  /* Применение дополнительных стилей через пропсы */
  ${({ customStyles }) => customStyles && css`
    /* Дополнительные стили, переданные через пропсы */
    ${customStyles}
  `}
`;

// Компонент ItemProduct
const ItemProduct = ({ id, img, cost, name, width, height, countryOfManufacture, materials, customStyles })
=> {
  // Отримання списку товарів зі стану за допомогою useSelector
  const items = useSelector(state => state.items);
  const dispatch = useDispatch(); // Отримання диспетчера Redux за допомогою хука useDispatch
  const [selectedColor, setSelectedColor] = useState(''); // Стан для вибраного кольору товару
```

```
  const [quantity, setQuantity] = useState(1); // Стан для кількості товару, що додається до кошика

  // Ефект для завантаження збереженого кошика при завантаженні компонента
  useEffect(() => {
    const savedCart = localStorage.getItem('cart');
    if (savedCart) {
      dispatch(itemsFetched(JSON.parse(savedCart)));
    }
  }, [dispatch]);

  // Функція для додавання товару до кошика
  const addToBasket = () => {
    const totalCost = cost * quantity;
    const newItem = { id, name, img, cost, color: selectedColor, quantity, totalCost };
    const itemExists = items.some(item => item.id === id && item.color === selectedColor);
    if (itemExists) {
      const updatedItems = items.map(item => {
        if (item.id === id && item.color === selectedColor) {
          return { ...item, quantity: item.quantity + quantity, totalCost: item.totalCost + totalCost };
        }
        return item;
      });
      dispatch(itemsFetched(updatedItems));
      localStorage.setItem('cart', JSON.stringify(updatedItems));

    } else {
      const updatedItems = [...items, newItem];
      dispatch(itemsFetched(updatedItems));
      localStorage.setItem('cart', JSON.stringify(updatedItems));
    }
    console.log(selectedColor, quantity, totalCost);
  };

  // Функція для зміни кількості товару
  const handleQuantityChange = (e) => {
    const value = parseInt(e.target.value, 10);
    if (value > 0) {
      setQuantity(value);
    }
  };

  // Повернення JSX компоненту зі стилізованим виглядом
 return (
    <StyledItemProduct customStyles={customStyles}>
      <div className='leftSide'>
        <Link to={`/itemPage/${id}`} state={{ id, img, cost, name, width, height, countryOfManufacture,
materials }}>
          <img className='card-img' src={img} alt={name} />

        </Link>
      </div>
      <div className='rightSide'>

        <h3 className="nameProduct">{name}</h3>
        <h3 className="costProduct">Ціна: {cost} грн</h3>

        <div className='colorSelect'>
          <ColorSelector
            colors={['#FF5733', '#33FF57', '#3357FF']}
            onSelect={setSelectedColor}
            selectedColor={selectedColor}
          />
        </div>
```

```
            <div className='productBtns'>

              <input
                type="number"
                value={quantity}
                onChange={handleQuantityChange}
                min="1"
              />
              <div className='totalCost-block'>Загальна ціна: <span className="totalCost">{cost * quantity}
грн</span></div>

              <button onClick={addToBasket}>Додати до кошика</button>
            </div>
          </div>
        </StyledItemProduct>
      );
    };

    listFilterItem.jsx // Список фідфільтрованих елементів
    import { useHttp, getData } from "../../../../services/useHttp";
    import { useEffect, useState } from "react";
    import ItemProduct from "../itemProduct/ItemProduct";
    import "./listFilterItem.css";

    // Компонент ListFilterItem
    const ListFilterItem = ({ filterName, species, categoryName }) => {
      const { request } = useHttp(); // Використання функції request з кастомного хука useHttp
      const [listItem, setListItem] = useState([]); // Стан для списку елементів

      // Ефект для завантаження та оновлення списку елементів
      useEffect(() => {
        const fetchData = async () => {
          try {
            const items = await getData(); // Виклик функції для отримання даних
            if (items) {
              setListItem(settingData(items)); // Встановлення даних в стан
              console.log(items);
            }
          } catch (error) {
            console.error("Не удалось получить данные", error); // Обробка помилок
          }
        };

        fetchData(); // Виклик функції для завантаження даних
      }, [request, filterName, species, categoryName]); // Залежності ефекту

      // Функція для обробки та відображення отриманих даних
      function settingData(arr) {
        const items = arr
          .filter((item) =>
            (filterName && item.subcategory === filterName) || // Фільтрація за підкатегорією
            (species && item.species === species) || // Фільтрація за видом
            (categoryName && item.category === categoryName) // Фільтрація за категорією
          )
          .map((item) => (
            <li key={item._id}>
              {/* Виклик компонента товару з отриманими параметрами */}
              <ItemProduct
                _id={item._id}
                id={item._id}
                name={item.name}
                img={item.img}
                cost={item.cost}
```

```
                width={item.width}
                height={item.height}
                materials={item.materials}
                countryOfManufacture={item.countryOfManufacture}
              />
            </li>
          ));
        return <ul className="catalog_list-blok">{items}</ul>; // Повернення списку з елементами
      }

      // Повернення JSX компоненту
      return (
        <div className="container">
          {listItem} {/* Відображення списку елементів */}
        </div>
      );
    };

    export default ListFilterItem;
```

modal.jsx // Каркас для модального вікна

```
    import React from 'react';
    import './modal.css';

    // Компонент Modal
    const Modal = ({ isOpen, onClose, children }) => {
      if (!isOpen) return null; // Перевірка, чи модальне вікно відкрите; якщо ні, повертаємо null

      // Повернення JSX компоненту модального вікна
      return (
        <div className="modal-overlay">
          <div className="modal-content">
            <span className="modal-close" onClick={onClose}>&times;</span> {/* Кнопка закриття модального
вікна */}
            {children} {/* Вміст модального вікна, переданий як дочірні елементи */}
          </div>
        </div>
      );
    }

    export default Modal; // Експорт компонента Modal
```

search.jsx // Реалізація пошуку товарів

```
    import React, { useEffect, useState } from "react"; // Імпорт бібліотеки React
    import { useHttp, getData } from "../../../services/useHttp"; // Імпорт функції useHttp та getData з сервісу
    import { Link } from "react-router-dom"; // Імпорт компонента Link для роутингу
    import "./search.css"; // Імпорт файлу стилів для компоненту пошуку

    // Компонент SearchProducts
    const SearchProducts = () => {
      const { request } = useHttp(); // Використання хука useHttp для виконання HTTP-запитів
      const [products, setProducts] = useState([]); // Стейт для зберігання списку продуктів
      const [query, setQuery] = useState(""); // Стейт для зберігання значення запиту користувача
      const [results, setResults] = useState([]); // Стейт для зберігання результатів пошуку

      // Ефект для завантаження списку продуктів при монтуванні компоненту
      useEffect(() => {
        // Виклик функції getData для отримання даних
        getData()
          .then((data) => {
            setProducts(data); // Встановлення отриманих даних у стейт products
          })
          .catch((error) => {
```

```
      console.error("Не вдалося отримати дані", error); // Обробка помилки при отриманні даних
    });
  }, [request]); // Залежність від змінної request для перезавантаження даних при кожному HTTP-запиті

  // Обробник зміни значення в полі введення
  function handleInputChange(event) {
    const input = event.target.value; // Отримання значення з поля введення
    setQuery(input); // Встановлення нового значення запиту у стейт query
    if (input.trim() === '') {
      setResults([]); // Якщо поле введення порожнє, очищуємо результати пошуку
    } else {
      // Фільтрація продуктів за назвою відповідно до введеного запиту
      const filteredProducts = products.filter(product =>
        product.name.toLowerCase().includes(input.toLowerCase())
      );
      setResults(filteredProducts); // Встановлення відфільтрованих продуктів у стейт results
    }
  };

  // Повернення JSX компоненту
  return (
    <div>
      {/* Поле введення для пошуку продуктів */}
      <input
        className="search-input"
        type="text"
        placeholder="Search for products..."
        value={query}
        onChange={handleInputChange}
      />
      {/* Відображення результатів пошуку, якщо вони є */}
      {
        results.length > 0 && <ul className="search_results-block">
          {/* Мапування результатів пошуку у список елементів */}
          {results.map(product => (
            <li key={product.id}>
              {/* Посилання на сторінку з детальною інформацією про продукт */}
              <Link to={`/itemSearchPage/${product.id}`} state={{ product }}>
                {/* Блок для відображення зображення та назви продукту */}
                <div className="search_results-element">
                  <img className="search_results-img" src={product.img} alt="" /> {/* Зображення продукту */}
                  <div>{product.name}</div> {/* Назва продукту */}
                </div>
              </Link>
            </li>
          ))}
        </ul>
      }
    </div>
  );
};

export default SearchProducts;
```

filters.jsx // Фільтри та категорії

```
import { useHttp } from "../../services/useHttp";
import { useEffect, useState } from "react";
import { Link } from 'react-router-dom';
import "./filters.css";

const Filters = () => {
  const { request } = useHttp(); // Використання хука для виконання HTTP-запитів
  const [filters, setFilters] = useState([]); // Стейт для зберігання списку фільтрів
```

```
  const [isHovered, setIsHovered] = useState(false); // Стейт для відстеження наведення миші на фільтр
  const [hoveredIndex, setHoveredIndex] = useState(null); // Стейт для зберігання індексу наведеного фільтру

  useEffect(() => {
   request("http://localhost:5000/filters") // Виклик HTTP-запиту для отримання списку фільтрів з сервера
    .then((data) => {
     setFilters(data); // Оновлення стейту зі списком фільтрів
    })
    .catch((error) => {
     console.error("Не вдалося отримати дані", error); // Обробка помилки при отриманні даних з сервера
    });
  }, []); // Ефект виконується лише після першого рендеру компоненту

  const handleMouseEnter = (index) => {
   setIsHovered(true); // Встановлення прапорця наведення миші на фільтр
   setHoveredIndex(index); // Встановлення індексу наведеного фільтру
  };

  const handleMouseLeave = () => {
   setIsHovered(false); // Скасування прапорця наведення миші на фільтр
   setHoveredIndex(null); // Очищення індексу наведеного фільтру
  };

  return (
   <div className="filters">
    <div className="container">
     <ul className="filters-list"> {/* Список фільтрів */}
      {filters.map((filter, index) => (
       <li
        key={index}
        onMouseEnter={() => handleMouseEnter(index)} // Обробник наведення миші на фільтр
        onMouseLeave={handleMouseLeave} // Обробник відведення миші від фільтру
       >
        <Link style={{textDecoration:"none", color:"black"}} to={`/itemCategoryPage/${filter.category}`}
state={{ filter }}>{filter.name}</Link> {/* Посилання на сторінку з продуктами відповідного фільтру */}
        {/* Блок елементів фільтру, якщо миша наведена на фільтр */}
        <div className="filters_elements-block">
         {isHovered && hoveredIndex === index
          ? filter.subfilters.map((item) => <Link to={`/itemFilterPage/${item.name}`} state={{ item }}><div
className="filters_element">{item.filterName}</div></Link>)
          : null}
        </div>
       </li>
      ))}
     </ul>
    </div>
   </div>
  );
 };

 export default Filters; // Експорт компоненту фільтрів
```

mainProduct.jsx // Відображення продуктів у вигляді каруселі

```
 import Slider from "react-slick";
 import "slick-carousel/slick/slick.css";
 import "slick-carousel/slick/slick-theme.css";
 import { getData } from "../../services/useHttp";
 import ItemProduct from "../common/layout/itemProduct/ItemProduct";
 import { useEffect, useState } from "react";

 const MainProduct = ({speciec}) => {
  const [items, setItems] = useState([]); // Стейт для зберігання списку товарів
```

```
  useEffect(() => {
    getData("/itemProduct") // Виклик функції getData для отримання списку товарів
      .then((items) => {
        if (Array.isArray(items)) {
          setItems(settingData(items)); // Оновлення стейту зі списком товарів
        } else {
          console.error("Отримані дані не є масивом:", items); // Виведення помилки, якщо отримані дані не є
масивом
        }
      })
      .catch((error) => {
        console.error("Не вдалося отримати дані", error); // Обробка помилки при отриманні даних
      });
  }, []); // Ефект виконується лише після першого рендеру компоненту

  function settingData(arr) {
    return arr.filter((item) => item.species === speciec); // Фільтрація товарів за вказаною спецією
  }

  const settings = { // Налаштування Slider
    dots: true, // Відображення крапок для навігації
    infinite: true, // Нескінченна прокрутка
    speed: 500, // Швидкість прокрутки
    slidesToShow: 5, // Кількість відображуваних слайдів одночасно
    slidesToScroll: 5, // Кількість слайдів, які прокручуються за один раз
    arrows: true, // Відображення стрілок для навігації
    responsive: [ // Налаштування для різних роздільних здатностей екрану
      {
        breakpoint: 1024, // Значення роздільної здатності, при якому застосовуються налаштування
        settings: {
          slidesToShow: 2,
          slidesToScroll: 1,
          infinite: true,
          dots: true
        }
      },
      {
        breakpoint: 600, // Значення роздільної здатності, при якому застосовуються налаштування
        settings: {
          slidesToShow: 1,
          slidesToScroll: 1
        }
      }
    ]
  };

  return (
    <div className="container"> {/* Контейнер для Slider */}
      <Slider {...settings}> {/* Використання Slider з налаштуваннями */}
        {items.map((item) => (
          <div key={item.id}> {/* Унікальний ключ для кожного товару */}
            <ItemProduct id={item._id} name={item.name} img={item.img} cost={item.cost}
width={item.width} height={item.height} materials={item.materials}
countryOfManufacture={item.countryOfManufacture}/> {/* Відображення товару */}
          </div>
        ))}
      </Slider>
    </div>
  );
};

export default MainProduct; // Експорт компоненту MainProduct
```

mainPage.jsx // Головна сторінка

```
import React, { useState } from 'react';
import MainProduct from "../../mainProduct/mainProduct";
import Footer from "../../common/footer/footer";
import Header from "../../common/header/header";
import Slide from "../../../resources/img/slide.svg";
import WalletImg from "../../../resources/img/wallet.svg";
import CardImg from "../../../resources/img/card.svg";
import CarImg from "../../../resources/img/car.svg";
import HeartImg from "../../../resources/img/heart.svg";
import './mainPage.css';

const MainPage = () => {

  return (
    <div className="App">

      <Header />
      <div className="slider">
        <img src={Slide} alt="" />
      </div>
      <div className="info_img-block">
        <div className="container">
          <div className="info_img-element">
            <img src={WalletImg} alt="" />
            <div className="info_img_element-txt">
              Оплата при отриманні
            </div>
          </div>
          <div className="info_img-element">
            <img src={CardImg} alt="" />
            <div className="info_img_element-txt">
              Оплата карткою онлайн
            </div>
          </div>
          <div className="info_img-element">
            <img src={CarImg} alt="" />
            <div className="info_img_element-txt">
              Швидка доставка
            </div>
          </div>
          <div className="info_img-element">
            <img src={HeartImg} alt="" />
            <div className="info_img_element-txt">
              Велика повага та любов до клієнтів!
            </div>
          </div>
        </div>
      </div>
      <div style={{ backgroundColor: "#F8F5FF" }}>
        <div className="news">
          <h1>Нові товари</h1>
          <MainProduct speciec="novetly" />
        </div>
        <div className="bests">
          <h1 style={{ textAlign: "center" }}>Найкраще</h1>
          <MainProduct speciec="best" />
        </div>
      </div>
      <div style={{ backgroundColor: "#F4F4F4" }}>
        <Footer />
      </div>
    </div>
```

```
  );
}

export default MainPage;
```

filterPage.jsx // Сторінка товарів, відфільтрованих за певною категорією

```
import ListFilterItem from "../../common/layout/listItem/listFilterItem"
import { useLocation } from 'react-router-dom';
import Header from "../../common/header/header";
import Footer from "../../common/footer/footer";

const FilterPage = () => {
  const location = useLocation(); // Отримання поточного маршруту з хука useLocation
  const filter = location.state?.filter.category; // Отримання категорії фільтра з даних маршруту

  return (
    <div>
      <Header />
      <ListFilterItem category="category" categoryName={filter} />{/* Відображення відфільтрованих товарів за категорією */}
      <div style={{ backgroundColor: "#F4F4F4" }}>
        <Footer />
      </div>
    </div>

  )
}

export default FilterPage;
```

subfilterPage.jsx // Сторінка товарів, відфільтрованих за підфільтром категорії

```
import ListFilterItem from "../../common/layout/listItem/listFilterItem"
import { useLocation } from 'react-router-dom';
import Header from "../../common/header/header";
import Footer from "../../common/footer/footer";

const SubFilterPage = () => {
  const location = useLocation(); // Отримання поточного маршруту з хука useLocation
  const filter = location.state?.item.name; // Отримання підфільтра категорії з даних маршруту

  return (
    <div>
      <Header />
      <ListFilterItem subcategory="subcategory" filterName={filter} />
      <div style={{ backgroundColor: "#F4F4F4" }}>
        <Footer />
      </div>
    </div>
  )
}

export default SubFilterPage;
```

Login.jsx // Вхід у систему

```
import React, { useState } from 'react';
import axios from 'axios';
import './AuthForms.css';

const Login = () => {
  const [username, setUsername] = useState(''); // Стан для збереження введеного користувачем імені
  const [password, setPassword] = useState(''); // Стан для збереження введеного користувачем пароля
  const [message, setMessage] = useState(''); // Стан для збереження повідомлення
```

```
    const handleSubmit = async (e) => { // Функція для обробки подання форми
      e.preventDefault(); // Зупинка стандартної поведінки форми
      try {
        const response = await axios.post('/login', { username, password }); // Відправлення POST-запиту на сервер для автентифікації
        localStorage.setItem('token', response.data.token); // Збереження токену в локальному сховищі
        setMessage('Ви успішно увійшли'); // Встановлення повідомлення про успішний вхід
      } catch (error) {
        setMessage(error.response.data.error); // Встановлення повідомлення про помилку, отримане від сервера
      }
    };

    return (
      <div className="auth-form-container"> {/* Контейнер для форми автентифікації */}
        <h2>Вхід</h2> {/* Заголовок форми */}
        <form onSubmit={handleSubmit}> {/* Форма для входу */}
          <input
            type="text"
            placeholder="Ім'я користувача"
            value={username}
            onChange={(e) => setUsername(e.target.value)}
          /> {/* Поле для введення імені користувача */}
          <input
            type="password"
            placeholder="Пароль"
            value={password}
            onChange={(e) => setPassword(e.target.value)}
          /> {/* Поле для введення пароля */}
          <button type="submit">Увійти</button> {/* Кнопка для відправлення форми */}
        </form>
        {message && <p>{message}</p>} {/* Відображення повідомлення, якщо воно є */}
      </div>
    );
  };

  export default Login;
```

Registration.jsx // Форма реєстрації

```
  import React, { useState } from 'react';
  import axios from 'axios';
  import './AuthForms.css';

  const Registration = () => {
    const [username, setUsername] = useState(''); // Стан для збереження введеного користувачем імені
    const [email, setEmail] = useState(''); // Стан для збереження введеного користувачем email
    const [phone, setPhone] = useState(''); // Стан для збереження введеного користувачем телефону
    const [password, setPassword] = useState(''); // Стан для збереження введеного користувачем пароля
    const [message, setMessage] = useState(''); // Стан для збереження повідомлення

    const handleSubmit = async (e) => { // Функція для обробки подання форми
      e.preventDefault(); // Зупинка стандартної поведінки форми
      try {
        const response = await axios.post('/register', { username, email, phone, password }); // Відправлення POST-запиту на сервер для реєстрації
        setMessage(response.data.message); // Встановлення повідомлення про успішну реєстрацію
      } catch (error) {
        setMessage(error.response.data.error); // Встановлення повідомлення про помилку, отримане від сервера
      }
    };

    return (
```

```
            <div className="auth-form-container">
              <h2>Реєстрація</h2> {/* Заголовок форми */}
              <form onSubmit={handleSubmit}> {/* Форма для реєстрації */}
                <input
                  type="text"
                  placeholder="Ім'я користувача"
                  value={username}
                  onChange={(e) => setUsername(e.target.value)}
                /> {/* Поле для введення імені користувача */}
                <input
                  type="email"
                  placeholder="Email"
                  value={email}
                  onChange={(e) => setEmail(e.target.value)}
                /> {/* Поле для введення email */}
                <input
                  type="phone"
                  placeholder="Телефон"
                  value={phone}
                  onChange={(e) => setPhone(e.target.value)}
                /> {/* Поле для введення телефону */}
                <input
                  type="password"
                  placeholder="Пароль"
                  value={password}
                  onChange={(e) => setPassword(e.target.value)}
                /> {/* Поле для введення пароля */}
                <button type="submit">Зареєструватися</button> {/* Кнопка для відправлення форми */}
              </form>
              {message && <p>{message}</p>} {/* Відображення повідомлення, якщо воно є */}
            </div>
          );
        };

        export default Registration;
```

UserProfile.jsx // Особистий кабінет користувача

```
        import React, { useState, useEffect } from 'react';
        import axios from 'axios';
        import { Link } from 'react-router-dom';
        import './userProfile.css';

        const UserProfile = () => {
          const [user, setUser] = useState(null); // Стан для збереження даних користувача

          useEffect(() => {
            const fetchUserData = async () => {
              const token = localStorage.getItem('token'); // Отримання токена з локального сховища
              if (token) {
                try {
                  const response = await axios.get('/me', { // Відправлення GET-запиту на сервер для отримання
даних користувача
                    headers: { 'Authorization': `Bearer ${token}` } // Додавання заголовка з токеном для
автентифікації
                  });
                  setUser(response.data); // Збереження отриманих даних користувача
                } catch (error) {
                  console.error('Error fetching user data:', error); // Обробка помилки під час отримання даних
користувача
                }
              }
            };
            fetchUserData();
```

```
    }, []);

    const handleLogout = async () => {
      try {
        const response = await axios.post('/logout'); // Відправлення POST-запиту на сервер для виходу з облікового запису
        if (response.status === 200) {
          // Clear user data or perform any additional actions upon logout
          setUser(null); // Очищення даних користувача
          localStorage.removeItem('token'); // Видалення токена з локального сховища
          // Redirect or display message as needed
          console.log('Logged out successfully'); // Виведення повідомлення про успішний вихід з облікового запису
        }
      } catch (error) {
        console.error('Error logging out:', error); // Обробка помилки під час виходу з облікового запису
      }
    };

    if (!user) {
      return <p>Загрузка...</p>; // Повідомлення про завантаження, якщо дані користувача ще не отримані
    }

    return (
      <div className="user-profile">
        <h2>Особистий кабінет</h2>
        <p>Ім'я користувача: {user.username}</p> {/* Виведення імені користувача */}
        <h3>Мої замовлення</h3>
        {user.orders.map((order, index) => ( // Виведення списку замовлень користувача
          <div key={index} className="order">
            <h4>Замовлення {index + 1}</h4>
            <p>Загальна вартість: {order.totalCost} грн</p>
            {order.items.map((item, idx) => ( // Виведення списку товарів у замовленні
              <div key={idx} className="order-item">
                <p>Товар: {item.name}</p>
                <p>Кількість: {item.quantity}</p>
                <p>Ціна: {item.cost} грн</p>
              </div>
            ))}
          </div>
        ))}
        <Link to={'/'}>
          <button className="btn-logout" onClick={handleLogout}>Вийти</button> {/* Кнопка для виходу з облікового запису */}
        </Link>
      </div>
    );
  };

  export default UserProfile;

  reducer.js // Стан корзини, створений за допомогою Redux
  const initialState = { // Початковий стан додатку
    items: [], // Масив товарів
    status: "idle", // Статус запиту
  };

  const reducer = (state = initialState, action) => { // Редуктор для управління станом
    switch (action.type) { // Обробка різних типів дій
      case "ITEMS_FETCHING": // Дія при отриманні товарів
        return {
          ...state,
```

```
            status: "loading", // Зміна статусу на "loading" (завантаження)
           };
         case "ITEMS_FETCHED": // Дія при успішному отриманні товарів
           return {
            ...state,
            items: action.payload, // Оновлення масиву товарів
            status: "idle", // Зміна статусу на "idle" (перерва)
           };
         case 'REMOVE_ITEM': // Дія при видаленні товару
           return {
            ...state,
            items: state.items.filter(item => item.id !== action.payload) // Фільтрація товарів: видалення товару з
вказаним id
           };
         case "ITEMS_FETCHING_ERROR": // Дія при помилці отримання товарів
           return {
            ...state,
            status: "error", // Зміна статусу на "error" (помилка)
           };

         default:
           return state; // Повернення поточного стану за замовчуванням
       }
      };

      export default reducer;
```

index.js // Перелік методів для ред'юсера

```
      export const itemsFetching = () => { // Дія: початок отримання товарів
        return {
          type: 'ITEMS_FETCHING' // Тип дії
        }
      }

      export const itemsFetched = (items) => { // Дія: успішне отримання товарів
        return {
          type: 'ITEMS_FETCHED', // Тип дії
          payload: items // Дані: отримані товари
        }
      }

      export const itemsFetchingError = () => { // Дія: помилка при отриманні товарів
        return {
          type: 'ITEMS_FETCHING_ERROR' // Тип дії
        }
      }

      export const removeItem = (id) => ({ // Дія: видалення товару
        type: 'REMOVE_ITEM', // Тип дії
        payload: id // Дані: id товару, який потрібно видалити
      });
```

admin.jsx // Форми адміністративної частини

```
      import React, { useState, useEffect } from 'react';
      import { getData, getData2 } from '../../services/useHttp';
      import "./AdminPanel.css"

      // Компонент форми елемента продукту
      const ItemProductForm = () => {
        // Стан для зберігання даних елемента продукту та списку елементів
        const [itemProduct, setItemProduct] = useState({
          _id: '',
          img: '',
```

```
    cost: '',
    name: '',
    species: '',
    category: '',
    subcategory: '',
    colors: [{ color: '', image: '' }],
    width: '',
    height: '',
    countryOfManufacture: '',
    materials: ''
});

const [items, setItems] = useState([]); // Стан для списку елементів
const [isEditing, setIsEditing] = useState(false); // Стан для визначення режиму редагування

// Обробник зміни в полі вводу
const handleChange = (e, index) => {
    const { name, value } = e.target;
    if (name === 'colors') {
        const newColors = [...itemProduct.colors];
        newColors[index].color = value;
        setItemProduct({ ...itemProduct, colors: newColors });
    } else if (name === 'colorImage') {
        const newColors = [...itemProduct.colors];
        newColors[index].image = value;
        setItemProduct({ ...itemProduct, colors: newColors });
    } else {
        setItemProduct({ ...itemProduct, [name]: value });
    }
};

// Додавання поля для кольору
const addColorField = () => {
    setItemProduct({
        ...itemProduct,
        colors: [...itemProduct.colors, { color: '', image: '' }],
    });
};

// Видалення поля для кольору
const removeColorField = (index) => {
    const newColors = [...itemProduct.colors];
    newColors.splice(index, 1);
    setItemProduct({ ...itemProduct, colors: newColors });
};

// Відправка форми
const handleSubmit = async (e) => {
    e.preventDefault();
    try {
        const method = isEditing ? 'PUT' : 'POST';
        const colorsData = itemProduct.colors.map(color => ({ color: color.color, image: color.image }));
        const requestBody = { ...itemProduct, colors: colorsData };
        delete requestBody._id;

        const response = await fetch(`/itemProducts${isEditing ? `/${itemProduct._id}` : ''}`, {
            method,
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify(requestBody)
        });

        if (response.ok) {
            const updatedItem = await response.json();
```

```
                setItems(isEditing ? items.map(item => item._id === updatedItem._id ? updatedItem : item) :
[...items, updatedItem]);
                setItemProduct({
                  _id: '',
                  img: '',
                  cost: '',
                  name: '',
                  species: '',
                  category: '',
                  subcategory: '',
                  colors: [{ color: '', image: '' }],
                  width: '',
                  height: '',
                  countryOfManufacture: '',
                  materials: ''
                });
                setIsEditing(false);
              } else {
                console.error('Failed to save item product:', response.statusText);
              }
            } catch (error) {
              console.error('Error saving item product:', error);
            }
          };

          // Обробник для редагування
          const handleEdit = (item) => {
            setItemProduct({
              _id: item._id,
              img: item.img,
              cost: item.cost,
              name: item.name,
              species: item.species,
              category: item.category,
              subcategory: item.subcategory,
              colors: item.colors || [{ color: '', image: '' }],
              width: item.width || '',
              height: item.height || '',
              countryOfManufacture: item.countryOfManufacture || '',
              materials: item.materials || ''
            });
            setIsEditing(true);
          };

          // Видалення елемента продукту
          const handleDelete = async (_id) => {
            try {
              const response = await fetch(`/itemProducts/${_id}`, {
                method: 'DELETE'
              });

              if (response.ok) {
                setItems(items.filter(item => item._id !== _id));
              } else {
                console.error('Failed to delete item product:', response.statusText);
              }
            } catch (error) {
              console.error('Error deleting item product:', error);
            }
          };

          // Ефект для отримання списку елементів
          useEffect(() => {
```

```
        const fetchItems = async () => {
            const response = await fetch('/itemProducts');
            if (response.ok) {
                const data = await response.json();
                setItems(data);
            } else {
                console.error('Failed to fetch item products:', response.statusText);
            }
        };

        fetchItems();
    }, []);

    // Повернення розмітки
    return (
        <div className="admin-panel">
            <h2>Item Product Form</h2>
            <form onSubmit={handleSubmit} className="item-product-form">
                {/* Рядки для введення даних */}
                {/* Кнопки для додавання та видалення полів для кольору */}
                {/* Кнопка для відправки форми */}
            </form>
            <h2>Existing Item Products</h2>
            <ul>
                {/* Відображення списку елементів з кнопками для редагування та видалення */}
            </ul>
        </div>
    );
};
import React, { useState, useEffect } from 'react';
import { getData, getData2 } from '../../services/useHttp';
import "./AdminPanel.css"

// Компонент форми фільтра
const FilterForm = () => {
    // Стан для зберігання даних фільтра та списку фільтрів
    const [filter, setFilter] = useState({
        id: '',
        name: '',
        category: '',
        subfilters: [{ id: '', filterName: '', name: '' }]
    });

    const [filters, setFilters] = useState([]); // Стан для списку фільтрів
    const [isEditing, setIsEditing] = useState(false); // Стан для визначення режиму редагування

    // Ефект для отримання списку фільтрів з сервера при завантаженні
    useEffect(() => {
        const fetchFilters = async () => {
            const data = await getData2(); // Передбачаю, що ця функція отримує дані з сервера
            if (data) setFilters(data); // Оновлення стану фільтрів з отриманими даними
        };

        fetchFilters(); // Виклик функції отримання даних фільтрів
    }, []);

    // Обробник зміни значення в полі вводу
    const handleChange = (e) => {
        const { name, value } = e.target;
        setFilter({ ...filter, [name]: value });
    };

    // Обробник зміни підфільтра
```

```
    const handleSubfilterChange = (index, e) => {
      const { name, value } = e.target;
      const newSubfilters = filter.subfilters.map((subfilter, subIndex) => {
        if (subIndex === index) {
          return { ...subfilter, [name]: value };
        }
        return subfilter;
      });
      setFilter({ ...filter, subfilters: newSubfilters });
    };

    // Додавання нового підфільтра
    const addSubfilter = () => {
      setFilter({
        ...filter,
        subfilters: [...filter.subfilters, { id: '', filterName: '', name: '' }]
      });
    };

    // Відправка форми
    const handleSubmit = async (e) => {
      e.preventDefault();
      try {
        const method = isEditing ? 'PUT' : 'POST'; // Вибір методу HTTP-запиту на основі режиму
        const response = await fetch(`/filters${isEditing ? `/${filter.id}` : ''}`, {
          method,
          headers: { 'Content-Type': 'application/json' },
          body: JSON.stringify(filter)
        });

        if (response.ok) {
          const updatedFilter = await response.json(); // Оновлення фільтра з отриманими даними з сервера
          setFilters(isEditing ? filters.map(f => f.id === updatedFilter.id ? updatedFilter : f) : [...filters,
updatedFilter]); // Оновлення списку фільтрів
          setFilter({
            id: '',
            name: '',
            category: '',
            subfilters: [{ id: '', filterName: '', name: '' }]
          }); // Очищення форми після відправки
          setIsEditing(false); // Вимкнення режиму редагування
        } else {
          console.error('Failed to save filter:', response.statusText);
        }
      } catch (error) {
        console.error('Error saving filter:', error);
      }
    };

    // Обробник для редагування фільтра
    const handleEdit = (filter) => {
      setFilter(filter); // Встановлення вибраного фільтра для редагування
      setIsEditing(true); // Увімкнення режиму редагування
    };

    // Обробник видалення фільтра
    const handleDelete = async (_id) => {
      try {
        const response = await fetch(`http://localhost:5000/filters/${_id}`, { method: 'DELETE' }); // Виклик
HTTP-запиту на видалення фільтра

        if (response.ok) {
```

```
                setFilters(filters.filter(filter => filter._id !== _id)); // Оновлення списку фільтрів, видаляючи видалений фільтр
            } else {
                console.error('Failed to delete filter:', response.statusText);
            }
        } catch (error) {
            console.error('Error deleting filter:', error);
        }
    };

    // Повернення розмітки компонента
    return (
        <div className="admin-panel">
            <h2>Filter Form</h2>
            <form onSubmit={handleSubmit}>
                {/* Поля вводу для основних даних фільтра */}
                {/* Поля вводу для підфільтрів */}
                <button type="button" onClick={addSubfilter}>Add Subfilter</button>
                <button type="submit">{isEditing ? 'Update Filter' : 'Add Filter'}</button>
            </form>
            <h2>Existing Filters</h2>
            <ul>
                {/* Відображення списку фільтрів з підфільтрами та кнопками дій */}
            </ul>
        </div>
    );
};

export default FilterForm;
```

adminPage.jsx // Сторінка адміністратора

```
import { ItemProductForm, FilterForm} from "../../admin/admin";

const AdminPage = () => {
    return(
        <div>
            <ItemProductForm/>
            <FilterForm/>
        </div>
    )
}

export default AdminPage;
```

ДОДАТОК Б

ВІДГУК
керівника економічного розділу
на кваліфікаційну роботу бакалавра
на тему:
" Розробка програмного забезпечення веб застосунку інтернет магазину іграшок мовою JS"

студента групи 121-20-1 Уваровського Максима Дмитровича

Керівник економічного розділу
доцент каф. ПЕП та ПУ, к.е.н **Л. В. Касьяненко**

ДОДАТОК В

Перелік файлів на диску

ПЕРЕЛІК ДОКУМЕНТІВ НА МАГНІТНОМУ НОСІЇ

Ім’я файлу	**Опис**
Пояснювальні документи	
Кваліфікаційна робота.docx	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Кваліфікаційна робота.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF
Програма	
програма.rar	Архів. Містить коди програми і скомпільовану програму
Презентація	
презентація.ppt	Презентація кваліфікаційної роботи