

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)
на тему:

«Інтернет-магазин поліграфічної продукції»

Виконав студент групи ПЗ-19-2
ЗАЙЦЕВ Нікіта Сергійович

Керівник роботи:
ВЛАСЕНКО Олег Васильович

Рецензент:
ЛОБАНЧИКОВА Надія Миколаївна

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

В.о. зав. кафедри інженерії
програмного забезпечення

Андрій МОРОЗОВ

«14» лютого 2023 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: ЗАЙЦЕВ Нікіта Сергійович

Керівник роботи: ВЛАСЕНКО Олег Васильович

Тема роботи: «Інтернет-магазин поліграфічної продукції»,
затверджена Наказом закладу вищої освіти від «14» лютого 2023 р., №70с

Вихідні дані для роботи: **проектування та розробка інтернет-магазину**

Консультанти з кваліфікаційної роботи бакалавра із зазначенням розділів,
що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Власенко О.В	16.02.2023р.	16.02.2023р.
2	Власенко О.В	16.02.2023р.	16.02.2023р.
3	Власенко О.В	16.02.2023р.	16.02.2023р.

Календарний план

№ з/п	Назва етапів випускної кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Аналізування вимог до програмного продукту	14 лютого 2023 – 25 лютого 2023	Виконано
2	Пошук, огляд та аналіз аналогічних веб-сайтів	26 лютого 2023 – 7 березня 2023	Виконано
3	Вивчення матеріалу по конкретній тематиці	8 березня 2023 – 21 березня 2023	Виконано
4	Планування структури програми	22 березня 2023 – 1 квітня 2023	Виконано
5	Проектування та реалізація архітектури веб-системи кінотеатру	2 квітня 2023 – 12 квітня 2023	Виконано
6	Написання програмного коду	13 квітня 2023 – 25 травня 2023	Виконано
7	Тестування роботи програми	25 травня 2023 – 26 травня 2023	Виконано
8	Аналіз одержаних результатів	26 травня 2023 – 28 травня 2023	Виконано
9	Оформлення пояснювальної записки	28 травня 2023 – 5 червня 2023	Виконано
10	Попередній захист	5 травня 2023	Виконано

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи складається з вступу, трьох розділів, висновків, списку використаної літератури та додатку.

Текстова частина викладена на 49 сторінках друкованого тексту. Пояснювальна записка має 11 сторінок додатків. Список використаних джерел містить 18 найменувань і займає 2 сторінки. В роботі наведено 44 рисунки. Загальний обсяг роботи – 62 сторінки.

Метою кваліфікаційної роботи є розробка та реалізація функціонального та ефективного інтернет-магазину, спеціалізованого на поліграфічних товарах.

Предметом дослідження є процес розробки та реалізації інтернет-магазину поліграфічної продукції. Дослідження охоплює аналіз вимог користувачів, проектування інтерфейсу, розробку функціональності, тестування та впровадження системи.

Об'єктом дослідження є інтернет-магазин поліграфічної продукції. Дослідження орієнтоване на вивчення та вдосконалення процесів замовлення, доставки та обробки даних, пов'язаних з продажем поліграфічної продукції через інтернет-магазин.

У першому розділі було отримано технічне завдання на розробку системи. Було виокремлено та проаналізовано вимоги до програмного продукту.

В другому розділі описано реалізацію модулю авторизації як для frontend, так і backend частин додатку.

У третьому розділі було розглянуто інтерфейс та порядок роботи інтернет-магазину поліграфічної продукції. Також було створено тест-кейси та протестовано кінцевий результат

Висновок містить в собі результати виконаної роботи.

У додатку представлений лістинг розробленого програмного продукту.

Ключові слова: NEXT JS, REACT JS, САЙТ, MONGODB, API, SSR.

					ІПЗ.КР.Б – 121 – 23 – ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата	Інтернет-магазин поліграфічної продукції	Літ.	Арк.	Аркушів
Розроб.		Н.С. Зайцев						
Керівник		О.В. Власенко					4	62
Рецензент		Н.М. Лобанчикова			Пояснювальна записка	Житомирська політехніка, група ІПЗ-19-2		
Зав. каф.		А.В. Морозов						

ABSTRACT

The explanatory note to the qualification work consists of a list of the introduction, three sections, conclusions, a list of used literature and an appendix.

The text part is laid out on 49 pages of printed text. The explanatory note has 11 pages of appendices. The list of used sources contains 18 names and occupies 2 pages. The work contains 44 drawings. The total volume of work is 62 pages.

The goal of the thesis is the development and implementation of a functional and effective online store, specialized in printed goods.

The subject of the study is the process of development and implementation of an online store of printing products. The research covers user requirements analysis, interface design, functionality development, testing and system implementation.

The object of the study is an online store of printing products. The study is focused on studying and improving the processes of ordering, delivery and data processing related to the sale of printing products through an online store.

In the first section, the technical task for the development of the system was obtained. The requirements for the software product were identified and analyzed.

The second section describes the implementation of the authorization module for both frontend and backend parts of the application.

In the third section, the interface and order of operation of the online store of printed products were considered. Test cases were also created and the final result was tested

The conclusion contains the results of the completed work.

The appendix presents a listing of the developed software product. Keywords: NEXT JS, WEB APPLICATION, SITE, MONGODB, REACT JS, API, SSR.

ЗМІСТ

РОЗДІЛ 1 АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ ІНТЕРНЕТ-МАГАЗИНУ ПОЛІГРАФІЧНОЇ ПРОДУКЦІЇ	9
1.1 Опис проекту та постановка задачі.....	9
1.2 Існуючі аналоги програмного продукту.....	10
1.3 Розробка архітектури інтернет-магазину	17
Висновки до першого розділу.....	20
РОЗДІЛ 2 ПРОЕКТУВАННЯ ІНТЕРНЕТ-МАГАЗИНУ ПОЛІГРАФІЧНОЇ ПРОДУКЦІЇ	22
2.1 Визначення варіантів використання та об'єктно-орієнтованої структури системи	22
2.2 Розробка бази даних системи.....	24
2.3 Проектування та реалізація алгоритмів роботи системи	26
2.4 Реалізація додатку інтернет-магазину поліграфічної продукції	29
Висновки до другого розділу	34
РОЗДІЛ 3 ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ ІНТЕРНЕТ-МАГАЗИНУ ПОЛІГРАФІЧНОЇ ПРОДУКЦІЇ.....	35
3.1. Порядок встановлення та налаштування параметрів системи	35
3.2 Структура інтерфейсу та порядок роботи	36
3.3 Тестування роботи програмного комплексу	44
Висновки до третього розділу.....	48
ВИСНОВКИ.....	49
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	50
ДОДАТКИ.....	52

ВСТУП

Актуальність теми. У сучасному цифровому світі, в якому люди все більше залучаються до використання інтернету для задоволення своїх потреб і бажань, поява інтернет-магазинів стала незаперечною необхідністю. Інтернет став не тільки місцем отримання інформації та спілкування, але й потужним інструментом комерції, який відкриває безмежні можливості для розвитку бізнесу.

У цьому контексті, поліграфічна галузь не залишається осторонь. З ростом конкуренції та постійними змінами в суспільстві, підприємства та організації потребують зручного та ефективного способу просування своїх товарів та послуг на ринок. Інтернет-магазин поліграфічної продукції стає відмінним рішенням, що дозволяє широкому колу клієнтів замовляти різноманітні поліграфічні вироби в будь-який час та з будь-якого місця.

Такий інтернет-магазин поліграфічної продукції має безліч переваг для покупців. Він забезпечує зручний пошук та порівняння товарів, швидке оформлення замовлення, можливість перегляду детальних описів та зображень продукції, а також інтерактивні можливості, такі як персоналізовані рекомендації та відгуки попередніх покупців. Крім того, покупці отримують доступ до широкого асортименту поліграфічних виробів від різних виробників, що дозволяє їм знайти оптимальний варіант за якістю та ціною.

З точки зору підприємств та організацій, інтернет-магазин поліграфічної продукції надає можливість розширити аудиторію та привернути нових клієнтів, підвищити рівень продажів та впливати на репутацію бренду. Крім того, використання інтернету дозволяє зменшити витрати на зберігання та демонстрацію товару, оптимізувати процеси логістики та виконання замовлень, а також забезпечити зручний механізм спілкування з клієнтами.

Отже, створення інтернет-магазину поліграфічної продукції є актуальним та перспективним напрямом розвитку поліграфічної галузі.

Ця кваліфікаційна робота присвячена дослідженню та розробці інтернет-магазину поліграфічної продукції з метою надання широкій аудиторії

можливості зручно та ефективно замовляти та придбавати різноманітні поліграфічні вироби. Основною метою цього проекту є створення онлайн-платформи, яка забезпечить замовникам доступ до широкого асортименту поліграфічних послуг та продуктів, швидкий та зручний процес оформлення замовлення.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- Аналіз ринку поліграфічних послуг та існуючих Інтернет-магазинів поліграфічної продукції для визначення переваг та недоліків існуючих рішень.
- Розробка архітектури та функціоналу інтернет-магазину поліграфічної продукції.
- Впровадження та тестування розробленого інтернет-магазину поліграфічної продукції для перевірки його ефективності, безпеки та зручності використання.
- Оцінка результатів тестування та формулювання рекомендацій для поліпшення функціоналу та ефективності інтернет-магазину поліграфічної продукції.

Ця робота дозволить вивчити особливості та виклики, пов'язані з розробкою та впровадженням інтернет-магазину поліграфічної продукції, а також допоможе покращити процес замовлення та придбання поліграфічних послуг. Результати цього дослідження можуть бути корисними для підприємств та організацій, які бажають розширити свою онлайн-присутність та поліпшити комунікацію зі своїми клієнтами за допомогою поліграфічної продукції.

РОЗДІЛ 1 АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ ІНТЕРНЕТ-МАГАЗИНУ ПОЛІГРАФІЧНОЇ ПРОДУКЦІЇ

1.1 Опис проекту та постановка задачі

Інтернет та інтернет ресурси – є невід’ємною частиною нашого життя вже багато років. Кожного року, навіть кожного дня, з’являється все більше і більше різних інтернет ресурсів та платформ, що спрощують життя, або просто допомагають. Інтернет використовують як для розваг, як інструмент роботи, так і у комерційній сфері.

Це не обійшло і сферу поліграфії. Через високу конкурентність, та бажання розвиватися, багато поліграфічних магазинів почали створювати свої інтернет-магазини для того, щоб закликати нових покупців не тільки зі свого міста, а і по всій країні, що дає велику можливість для розвитку бізнесу.

Інтернет-магазин поліграфічної продукції відкриває двері до нових ринків і аудиторій, дозволяючи поліграфічним компаніям привернути увагу клієнтів, які раніше могли бути недосяжними. Замовлення можуть бути оформлені з будь-якого місця та в будь-який час, що значно збільшує зручність для покупців.

Окрім того, інтернет-магазин поліграфічної продукції дозволяє поліграфічним компаніям ефективно керувати своїм асортиментом, вести стеження за запасами товарів та оновлювати їх у реальному часі. Це дозволяє уникнути непотрібного накопичення запасів і покращує управління логістикою.

Крім того, інтернет-магазин поліграфічної продукції може бути потужним інструментом маркетингу та просування бренду. Завдяки цифровому середовищу, компанії можуть проводити спеціальні акції, знижки та розсилати інформаційні бюлетені своїм клієнтам. Це сприяє збільшенню лояльності клієнтів та підвищенню свідомості про бренд.

У цьому контексті дослідження та аналіз напрямків використання інформаційних технологій для розробки інтернет-магазину поліграфічної

продукції є актуальним завданням, оскільки дозволяє розкрити потенціал цього інструменту для поліграфічного бізнесу та сприяє його успішному розвитку.

Постановка задачі полягає у визначенні конкретних цілей, які потрібно досягти у процесі розробки інтернет-магазину поліграфічної продукції. До основних задач проекту відносяться:

- Розробка функціоналу магазину: визначення основних можливостей, які має надавати інтернет-платформа, зокрема пошук та фільтрація товарів, процес замовлення, відстеження стану замовлення та доставка.
- Створення зручного та привабливого дизайну: розробка естетичного та інтуїтивно зрозумілого інтерфейсу, що сприятиме зручному користуванню магазином, включаючи розташування елементів, колірну гаму та типографіку.
- Розробка системи управління контентом: створення системи, яка дозволить адміністраторам магазину зручно додавати, видаляти та редагувати товари, описи, ціни та іншу інформацію.
- Процес інтеграції з іншими системами: забезпечення можливості інтеграції з іншими сервісами.

У результаті виконання поставлених задач планується створити функціональний, естетичний та безпечний інтернет-магазин поліграфічної продукції, який задовольнить потреби клієнтів та сприятиме розвитку бізнесу в галузі поліграфії.

1.2 Існуючі аналоги програмного продукту

В наші дні існує багато інтернет-магазинів поліграфічної продукції, але в Україні ця сфера не дуже розвинена, тому немає дуже відомих магазинів в цій сфері, але декілька прикладів знайшлися, такі як: Posters.com.ua, GeekPosters та Maxmall.

Всі ці сайти мають приблизно однаковий функціонал та сучасний дизайн, але через велику кількість функціоналу та нагромадження інформації втрачається зручність їх використання.

На рисунку 1.1 можна побачити каталог тематики постерів на сайті інтернет магазину Maxmall (<https://maxmall.com.ua/ua/>). На цій сторінці можна обрати тематику постерів та іншу інформацію про магазин

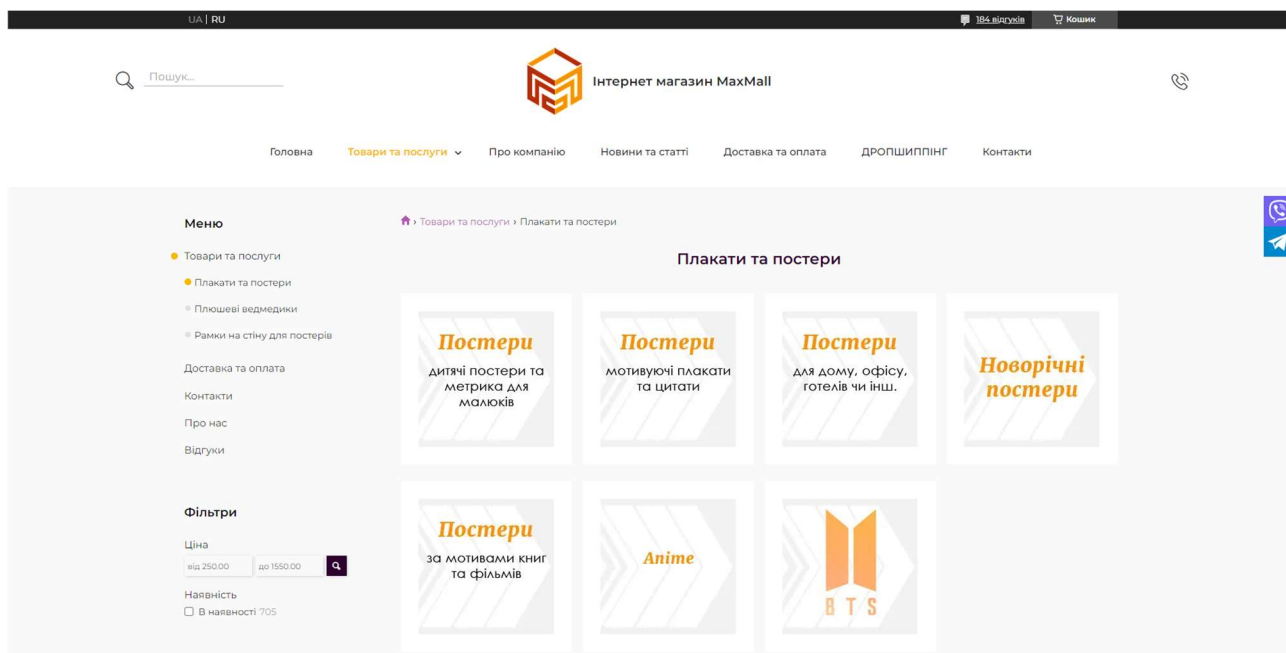


Рисунок 1.1. Каталог тематики постерів Maxmall

Навігація сайту знаходиться як з лівого боку так і у навігаційній панелі вгорі. В головній частині сайту ми можемо обрати тематику постерів готових для покупки.

На рисунки 1.2 ми можемо побачити усі доступні постери за обраною тематикою з можливістю їх придбання. У випадку зображеному на рисунку обраною тематикою є «Постери для дому, офісу, готелів чи інші.»

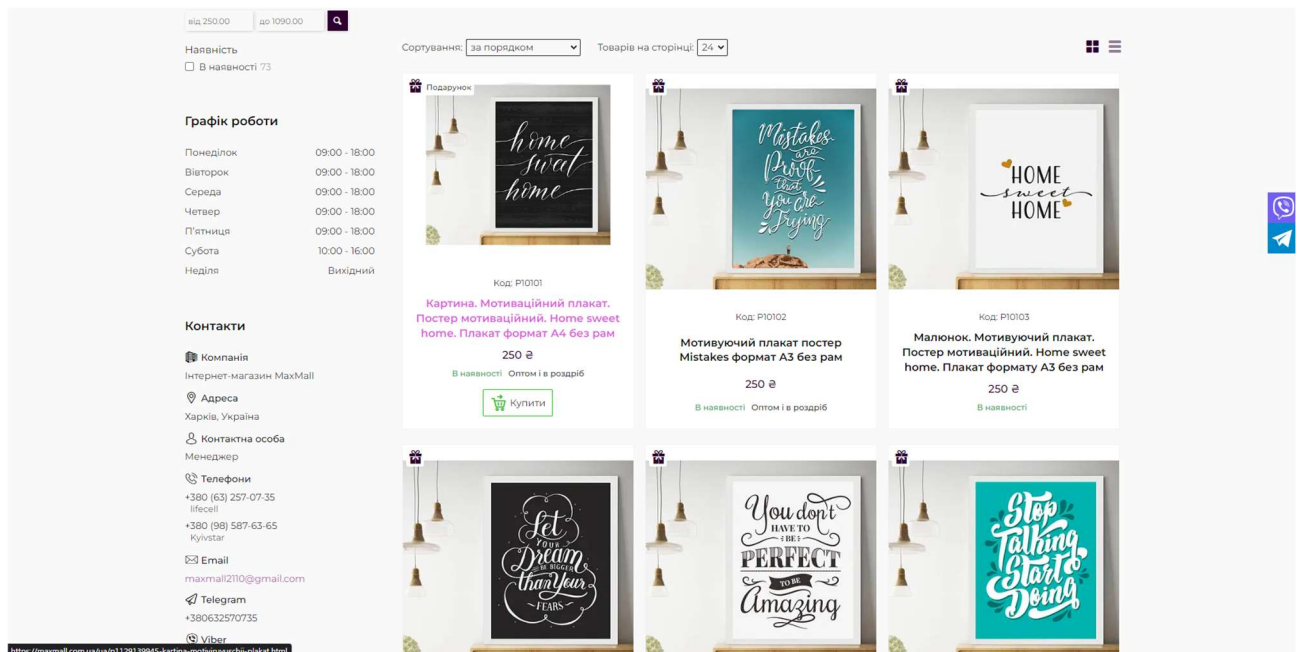


Рисунок 1.2. Каталог товарів за обраною тематикою

На цьому рисунку ми можемо побачити графік роботи, контакти, та сам товар який ми обрали. У кожного товару є назва, ціна, зображення, та можливість додати товар у кошик. Також користувач може відсортувати товари за кількома параметрами.

На рисунку 1.3 зображено сам кошик з обраним товаром. Ми можемо обрати кількість товарів які бажаємо замовити, можливість видалити товар з кошику.

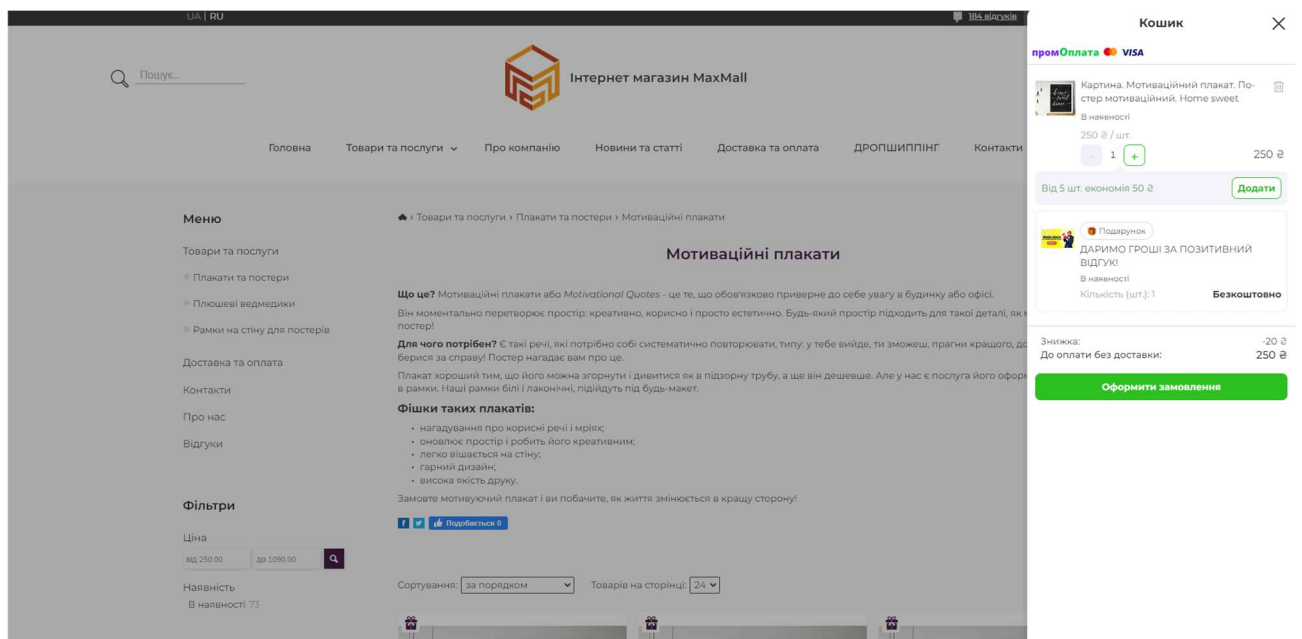


Рисунок 1.3. Кошик з замовленням

У разі якщо ми все обрали, ми можемо перейти до оформлення замовлення яке здійснюється на основі інтеграції з prom.ua. Сторінка оформлення замовлення показана на рисунку 1.4.

Рисунок 1.4. Вікно оформлення замовлення

Наступним аналогом буде «Poster.com.ua» (<https://posters.com.ua/>). У цього сайту надсучасний дизайн який привертає свою увагу, але при вході на сайт людина не одразу розуміє, що їй потрібно зробити щоб знайти потрібний товар. На рисунку 1.5 зображено головну сторінку сайту

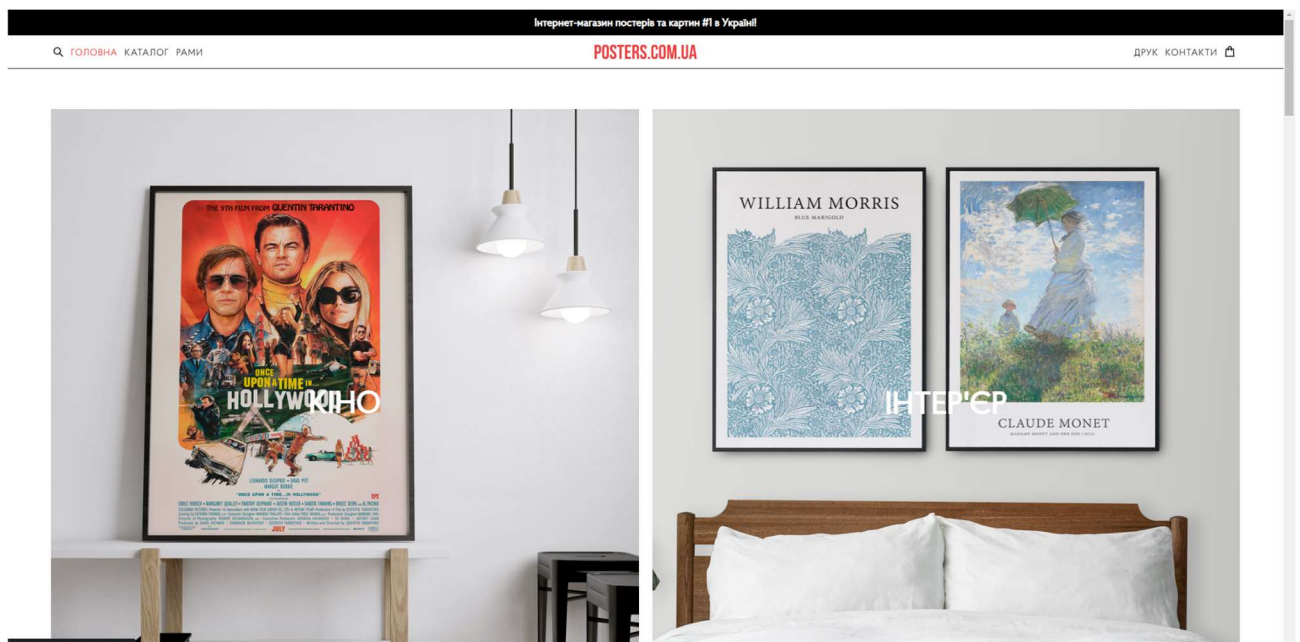


Рисунок 1.5. Головна сторінка сайту

На цій сторінці ми можемо побачити назву інтернет магазину та обрати, що саме нам потрібно. При виборі тематики, ми переходимо до каталогу товарів за обраною темою, що можемо побачити на рисунку 1.6.

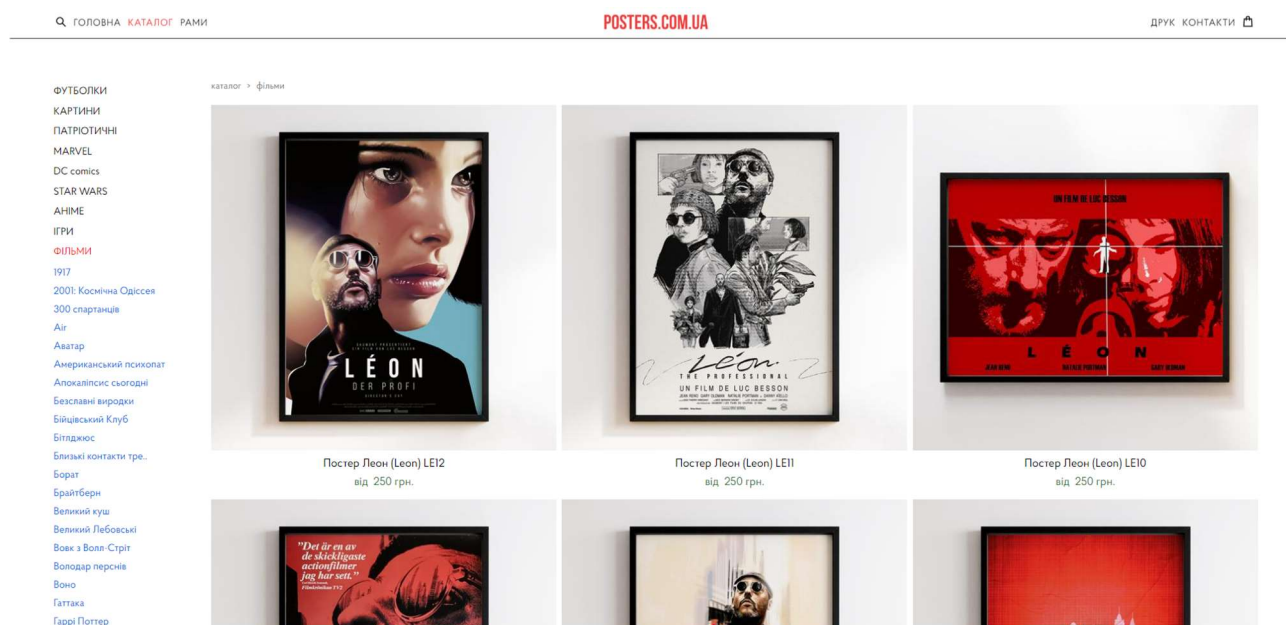


Рисунок 1.6. Каталог товарів за обраною темою

Зліва ми можемо побачити навігацію по тематикам товарів. Зверху ми можемо знайти навігацію по сайту, сторінку інформації про друк та кошик. При виборі товару ми бачимо повну інформацію про товар, можемо обрати розмір та оформити замовлення. Сторінка товару показана на рисунку 1.7.

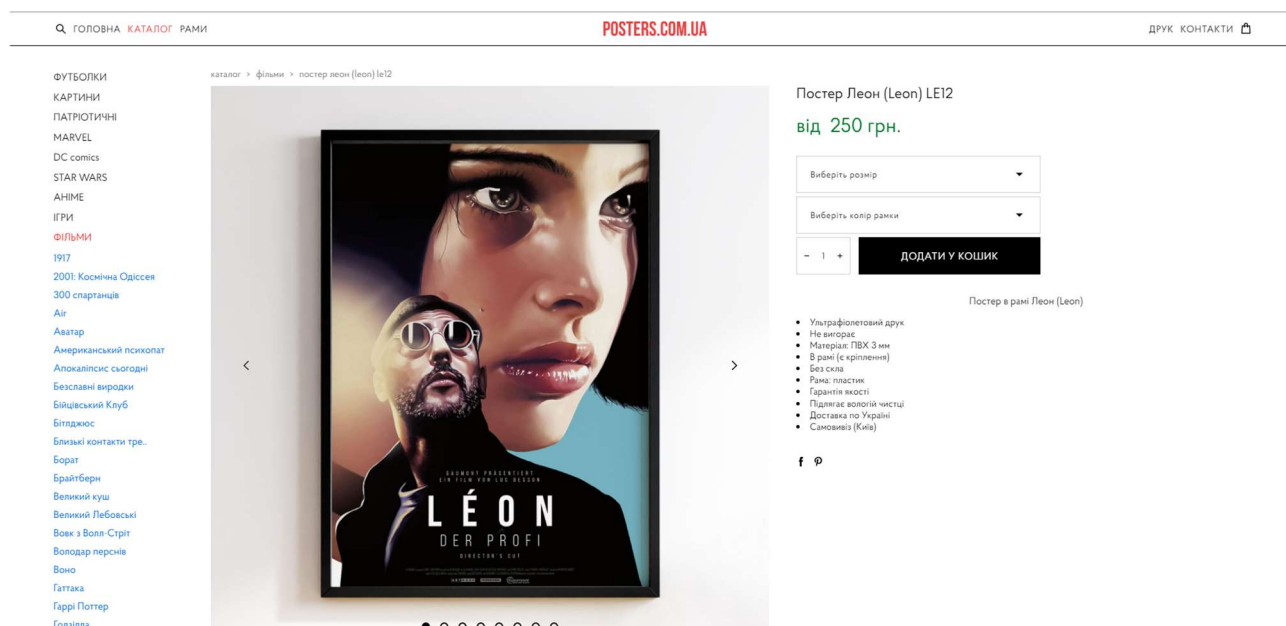


Рисунок 1.7 Сторінка обраного товару

При додаванні товару у кошик та переходу до оформлення замовлення нам потрібно заповнити форму та обрати тип оплати та доставки. На рисунку 1.8 зображено оформлення товару та доставки.

Рисунок 1.8 Оформлення замовлення

Останнім аналогом буде інтернет-магазин «Geekposters» (<https://geekposters.com.ua/ua/>). Цей інтернет магазин не сильно відрізняється функціоналом від попередніх.

На рисунку 1.9 показано головну сторінку сайту, на якій ми можемо побачити: новини сайту, меню з вибором тематики, навігаційну панель та популярні товари.

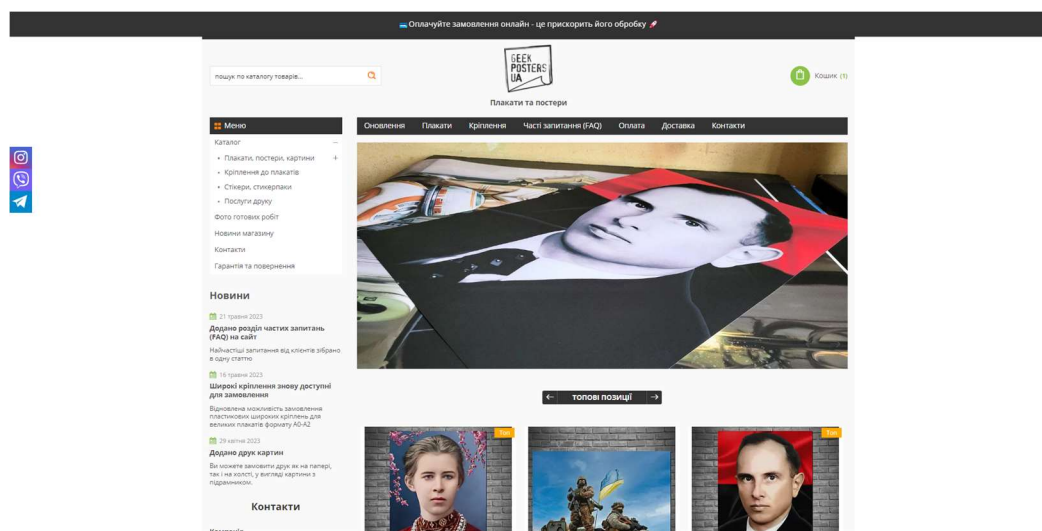


Рисунок 1.9. Головна сторінка сайту

Після пошуку необхідної тематики, ми бачимо весь каталог товарів за тематикою яку ми обрали. На рисунку 1.10 показано каталог товарів.

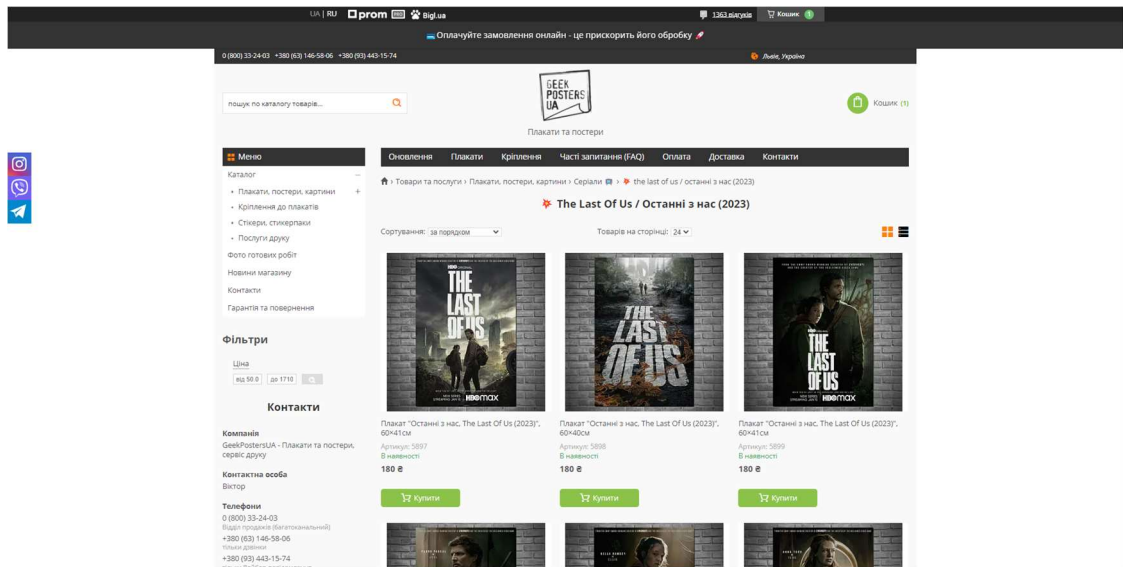


Рисунок 1.10. Каталог товарів за обраною тематикою

Далі ми можемо обрати потрібний нам постер та помістити його у кошик, після чого перейти до оформлення замовлення. Сторінка оформлення замовлення зображена на рисунку 1.11.

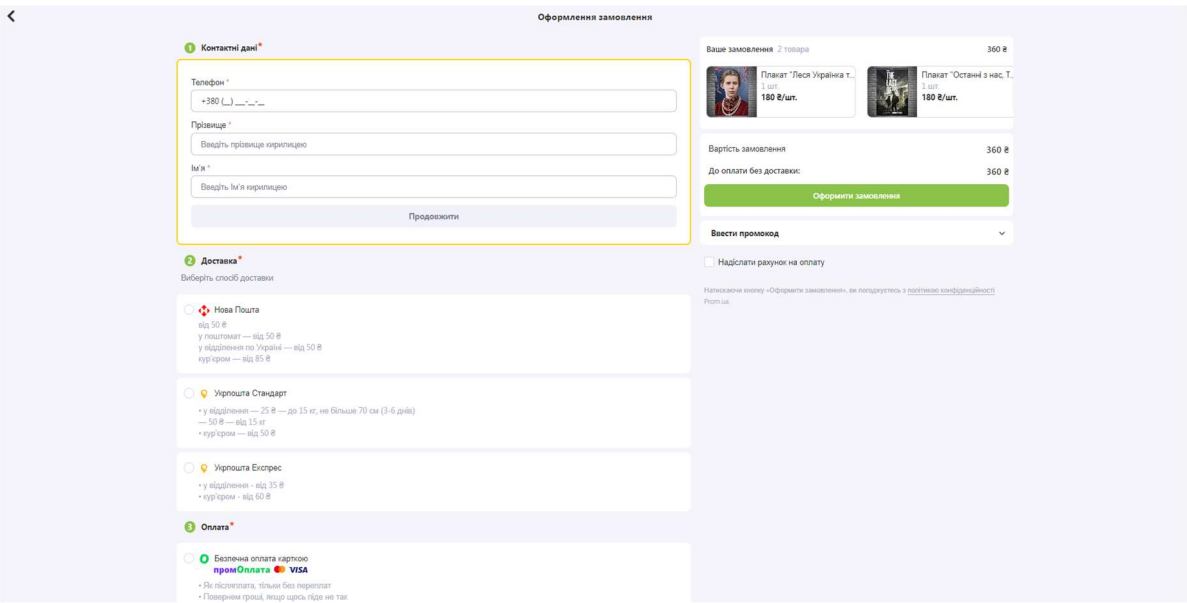


Рисунок 1.11. Оформлення замовлення

Можемо помітити, що цей магазин також використовує інтеграцію з Prom.ua для оформлення та оплати замовлення.

Отже, ознайомившись та проаналізувавши аналоги, ми можемо скласти таблицю порівняння 1.1

Порівняльна таблиця

	Maxmall	Poster.com.ua	Geekposter
Інтуїтивність інтерфейсу	+	-	+
Зручність використання	+	+	+
Кількість інформації	+	+	+

1.3 Розробка архітектури інтернет-магазину

Розробляючи додаток, важливо продумати, як він буде підтримуватися в майбутньому. Слід також розглянути можливість додавання та редагування змін. Важливо також розуміти, як зовнішні елементи інтерфейсу взаємодіють з внутрішніми процесами і як користувачі взаємодіють з додатком. У цьому може допомогти архітектура програмного забезпечення.

Архітектура програмного забезпечення – спосіб структурування програмної або обчислювальної системи, абстракція елементів системи на певній фазі її роботи. Система може складатись з кількох рівнів абстракції і мати багато фаз роботи, кожна з яких може мати окрему архітектуру [10].

Архітектура програмного проекту забезпечує створення високоякісного програмного забезпечення та подальше зниження витрат на підтримку та обслуговування.

Проектування архітектури програмного додатку є ключовим етапом в розробці будь-якого проекту. Правильно спроектована архітектура визначає стабільність, масштабованість, ефективність та зручність управління додатком.

Розробка якісної архітектури дозволяє легко та швидко додавати нові функції без великих змін в коді продукту, також має можливість витримати

велике навантаження даних, спрощує тестування програмного продукту та багато інших переваг перед розробкою без якісно розробленої архітектури.

Отже, успіх розробки якісного інтернет-магазину залежить від якісно розробленої архітектури програмного продукту.

Для створення та подальшої розробки проекту було обрано популярний фреймворк Next.js, що базується на JavaScript і використовується для побудови серверних та клієнтських частин додатків.

JavaScript (JS) — динамічна, об'єктно-орієнтована прототипна мова програмування. Реалізація стандарту ECMAScript. Найчастіше використовується для створення сценаріїв вебсторінок, що надає можливість на боці клієнта (пристрої кінцевого користувача) взаємодіяти з користувачем, керувати браузером, асинхронно обмінюватися даними з сервером, змінювати структуру та зовнішній вигляд веб-сторінки [16].

Next.js - це реактивний фреймворк для розробки веб-додатків на основі JavaScript та React. Він використовується для побудови сучасних та ефективних веб-додатків з використанням серверного рендерингу та рендерингу на стороні клієнта [5].

Next.js надає розробникам потужні можливості для створення універсальних додатків, які можуть працювати як на серверній, так і на клієнтській стороні. Він підтримує рендеринг на стороні сервера (Server-side Rendering - SSR), статичний генерацію (Static Site Generation - SSG) та рендеринг на стороні клієнта (Client-side Rendering - CSR). Це дозволяє розробникам підібрати найкращий підхід для кожної сторінки або компонента в залежності від їх потреб та вимог проекту.

Було розглянуто усі види рендерингу, такі як SSR, SSG, CSR:

SSR, або серверний рендеринг (Server-Side Rendering), є підходом до рендерингу веб-сторінок, при якому весь вміст сторінки генерується на сервері перед відправкою до клієнта. Основна перевага SSR полягає в тому, що веб-сторінка повністю готова до відображення на сервері, що дозволяє надсилати

клієнту вже готовий HTML-код. Це забезпечує кращу швидкість та оптимізацію для пошукових систем, оскільки контент відразу доступний для індексації [6].

При використанні SSR в Next.js, на сервері генерується HTML-код з даними та компонентами, і цей готовий HTML відправляється на клієнт. При переході на іншу сторінку, сервер знову генерує відповідний HTML для нової сторінки.

SSR дозволяє покращити користувацький досвід, особливо на початкових завантаженнях сторінок, забезпечуючи швидку реакцію та зручне відображення контенту для користувачів.

Крім того, SSR спрощує розробку під поглядом SEO, оскільки веб-сторінки вже мають вміст доступний для пошукових систем, що поліпшує індексацію та видимість в пошукових результатах.

Використання SSR в поєднанні з Next.js дозволяє зручно реалізовувати серверний рендеринг для ваших інтернет-магазинів поліграфічної продукції, забезпечуючи швидку та оптимізовану відповідь для користувачів.

SSG, або статичний генерація (Static Site Generation), є підходом до рендерингу веб-сторінок, при якому сторінки генеруються заздалегідь на етапі збирання проекту та зберігаються у вигляді статичних файлів. Це відрізняється від традиційного серверного рендерингу (SSR), де вміст генерується при кожному запиті [7].

CSR, або клієнтський рендеринг (Client-Side Rendering), є підходом до рендерингу веб-сторінок, при якому весь процес генерації вмісту відбувається на стороні клієнта, тобто в браузері. У випадку CSR, клієнт (браузер) отримує мінімальний HTML-файл, який містить основний JavaScript-код, необхідний для завантаження та рендерингу контенту на сторінці [8]. CSR також може мати деякі недоліки, зокрема пов'язані зі швидкістю завантаження сторінки, оскільки весь контент генерується на боці клієнта, що може зайняти багато часу при великому обсязі даних або погіршити індексацію пошуковими системами.

Проаналізувавши усі види рендерингу, було обрано використання SSR, так як він має низку переваг для розробки інтернет-магазину.

Однією з ключових особливостей Next.js є його підтримка роутингу на основі файлової системи. Це означає, що структура каталогів та файлів у проекті відображає шляхи та маршрути додатку. Це спрощує організацію та управління сторінками та компонентами, забезпечуючи логічну структуру проекту.

Крім того, Next.js має вбудовану підтримку для розширення функціональності за допомогою плагінів та middleware, що дозволяє розширювати його можливості та додавати власні функціональності зручним способом.

Next.js також пропонує велику екосистему, включаючи широкий вибір сторонніх пакетів та бібліотек, які полегшують розробку, такі як Ant Design для створення інтерфейсу [3]. Це дозволяє розробникам ефективно будувати складні та масштабовані веб-додатки з швидким відгуком та першокласними можливостями оптимізації.

Загалом, Next.js дає змогу створити потужний і ефективний інтернет-магазин із поліпшеним користувацьким інтерфейсом. Його гнучкість, простота використання і можливості SSR, CSR і SSG роблять його ідеальним вибором для такого типу проектів.

Висновки до першого розділу

У першому розділі нашої кваліфікаційної роботи ми провели детальний аналіз використання інформаційних технологій для розробки інтернет-магазину поліграфічної продукції.

Також було проведено аналіз аналогів за даною тематикою. Було визначено необхідний функціонал продукту.

Для нашого проекту було обрано фреймворк Next.js, який є широко використовуваним інструментом для розробки веб-додатків. Використання Next.js надає значні переваги, такі як рендеринг на стороні сервера та клієнта, проста маршрутизація та підтримка JavaScript як мови програмування. Для реалізації бекенду було обрано фреймворк Express, який дозволяє швидко та

ефективно створювати серверну частину додатку, а його поєднання з Next.js забезпечує нам потужний та функціональний стек технологій.

РОЗДІЛ 2 ПРОЕКТУВАННЯ ІНТЕРНЕТ-МАГАЗИНУ ПОЛІГРАФІЧНОЇ ПРОДУКЦІЇ

2.1 Визначення варіантів використання та об'єктно-орієнтованої структури системи

Після аналізу необхідних вимог ми маємо змогу побудувати діаграму варіантів використання продукту. Для створення діаграм ми використаємо додаток StarUML [1]. Діаграма зображена на рисунку 2.1

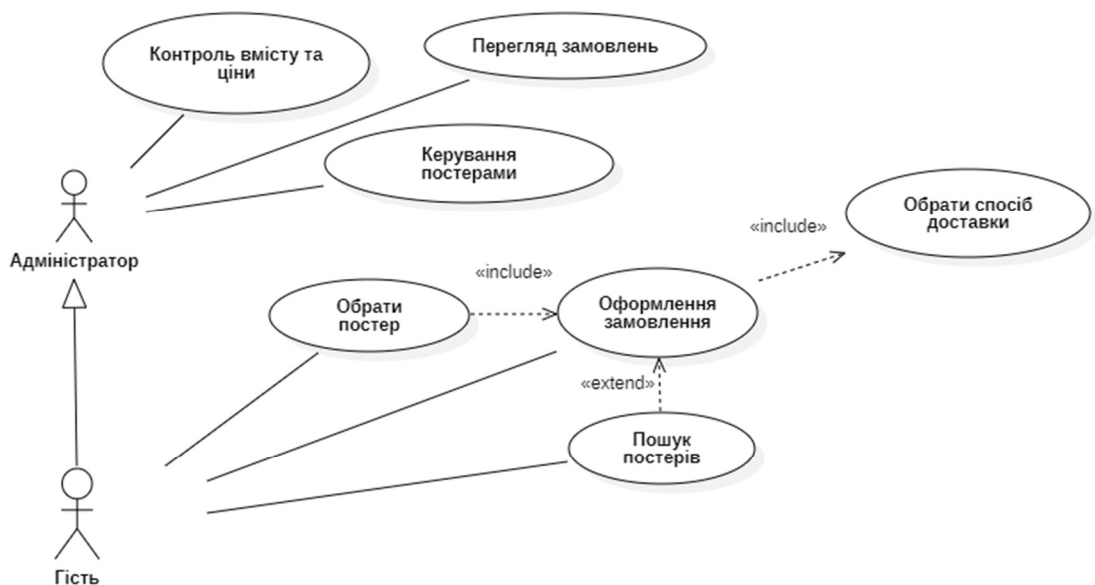


Рисунок 2.1. Діаграма варіантів використання

Ця діаграма варіантів використання продукту відображає основні функціональні можливості та ролі, які присутні в вашому інтернет-магазині поліграфічної продукції.

Гість:

- Вхід в ролі Адміністратора: Користувач може увійти в систему в якості адміністратора для отримання доступу до адміністративних функцій, таких як створення товарів, перегляд замовлень, контроль вмісту та цін, редагування та видалення товарів.

- Вибір товару: Користувач може переглядати доступний асортимент поліграфічної продукції та обирати товари, які його цікавлять. Варіант використання включає відображення списку товарів, детальний перегляд кожного товару з його характеристиками, зображеннями та ціною, а також можливість додати товар до кошика.
- Пошук товару: Користувач може здійснювати пошук товарів за різними критеріями, наприклад, назвою, категорією або ключовими словами. Варіант використання включає введення пошукового запиту користувачем та відображення результатів, що відповідають запиту.
- Оформлення замовлення: Користувач може додати обрані товари до кошика та оформити замовлення. Варіант використання включає перегляд вмісту кошика, вказання необхідних даних для доставки та оплати, підтвердження замовлення та отримання підтвердження успішного оформлення.

Адміністратор:

- Створення товарів: Адміністратор може створювати нові товари в системі. Це включає введення інформації про товар, таку як назва, опис, категорія, ціна, зображення тощо. Адміністратор також може встановлювати наявність товару на складі і надавати йому інші характеристики.
- Перегляд замовлень: Адміністратор має можливість переглядати список замовлень, що надійшли в систему. Це дозволяє адміністратору відстежувати статуси замовлень, перевіряти деталі замовлень, такі як обрані товари, адреса доставки, контактна інформація тощо.
- Контроль вмісту та цін: Адміністратор може забезпечувати контроль за вмістом сайту і цінами на товари. Це означає, що адміністратор може вносити зміни в інформацію про товари, оновлювати ціни, редагувати описи або зображення товарів для покращення інформаційного наповнення магазину.
- Редагування та видалення товарів: Адміністратор може редагувати існуючі товари, вносячи зміни в їхню інформацію або характеристики.

Також адміністратор має можливість видаляти товари з системи, якщо вони вже не продаються або більше не актуальні.

2.2 Розробка бази даних системи

Для збереження даних ми розробили базу даних в якій буде міститися вся інформація про товари, замовлення, та адміністраторів. Було прийнято рішення використовувати MongoDB. Діаграма бази даних зображена на рисунку 2.2

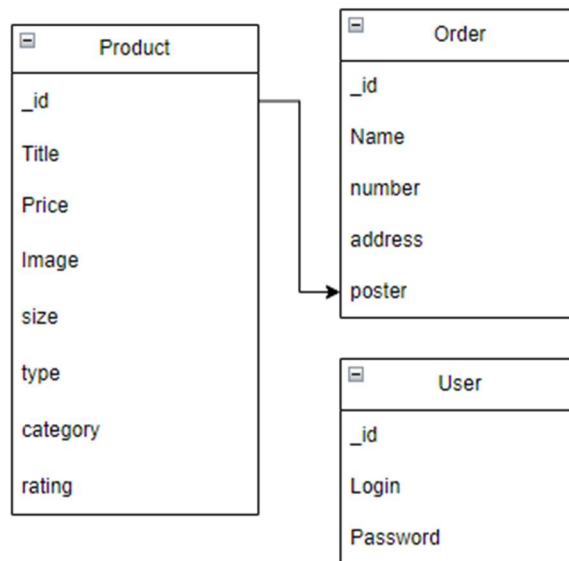


Рисунок 2.2. Діаграма бази даних

Mongodb це документ-орієнтована NoSql база даних, основна концепція якої полягає в тому, що для збереження даних відбувається у форматі JSON(JavaScript Object Notation), що робить дані легкими для обробки та їх розуміння [9][17]. Ми створили таблиці Order, User та Poster.

Таблиця User відповідає за збереження даних входу адміністратора. Її створення показано на рисунку 2.3

```
const UserSchema = new Schema( definition: {
  username: { type: String, required: true },
  password: { type: String, required: true }
})
```

Рисунок 2.3. Створення таблиці User

В таблиці присутні поля «username» та «password», що відповідають за логін та пароль відповідно. Структура таблиці вказана в таблиці 2.1

Таблиця 2.1

Структура таблиці «User»

Назва	Тип даних	Опис поля
username	String	Логін
Password	String	Пароль

В наступній створеній таблиці, а саме «Poster», зберігаються усі дані пов'язані з самим продуктом. Створення таблиці показане на Рисунку 2.4

```
const PosterSchema = new Schema( definition: {
  title: { type: String, required: true },
  price: { type: Number, required: true },
  image: { type: String, required: true },
  sizes: [{ type: Number }],
  types: [{ type: Number }],
  category: { type: Number, required: true },
  rating: { type: Number, required: true },
  ordersCount: { type: Number, required: true }
})
```

Рисунок 2.4. Створення таблиці «Poster»

В таблиці «Poster» зберігаються усі дані про товар, а саме: назва, ціна, зображення, розмір, тип, категорія, рейтинг та кількість замовлень. Все це зображено в таблиці 2.2

Таблиця 2.2

Структура таблиці «Product»

Назва	Тип даних	Опис поля
Title	String	Назва постеру
Price	Number	Ціна постеру
Image	String	Зображення
Sizes	Number	Розміри постеру

Types	Number	Тип постеру(Матовий або глянцевий)
Category	Number	Тематика постеру
Rating	Number	Рейтинг
ordersCount	Number	Кількість замовлень даного постеру

Останньою створеною таблицею є таблиця «Orders», яка зберігає дані усіх замовлень. Зображено на рисунку 2.5

```
const OrderSchema = new Schema( definition: {
  name: { type: String, required: true },
  address: { type: String, required: true },
  phoneNumber: { type: String, required: true },
  poster: { type: Array, required: true }
})
```

Рисунок 2.5 Створення таблиці «Order»

У таблиці «Order» відбувається збереження таких даних як: Ім'я замовника, адреса замовника, номер телефону та масив даних який зберігає в собі інформацію про постери, які користувач бажає замовити. У таблиці 2.3 наведено структуру таблиці «Order»

Таблиця 2.3

Структура таблиці «Order»

Назва	Тип даних	Опис поля
Name	String	Ім'я користувача
Address	String	Адреса замовлення
phoneNumber	String	Номер телефону
Poster	Array	Інформація про замовлені постери

Отже спроектована база даних має 3 таблиці

2.3 Проектування та реалізація алгоритмів роботи системи

					ІПЗ.КР.Б – 121 – 23 – ПЗ	26

Початком розробки програмного продукту є розробка діаграми активності користувачів, зображену на рисунку 2.6.

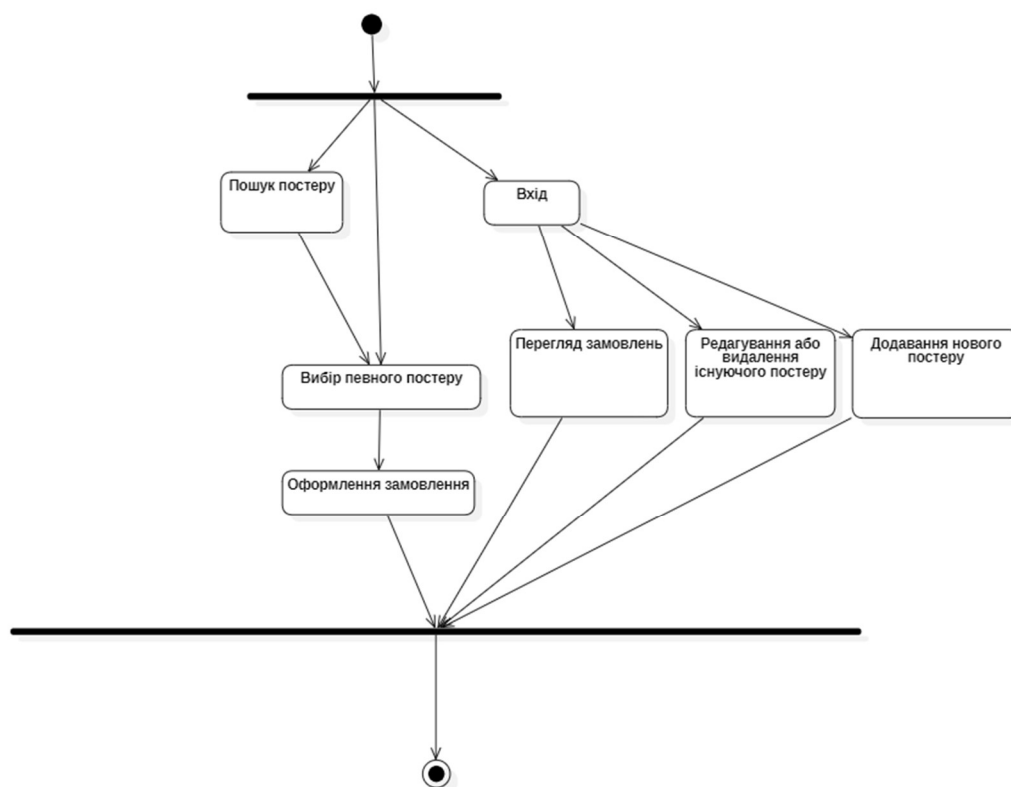


Рисунок 2.6. Діаграма активності

На даній діаграмі ми можемо побачити, що неавторизований користувач має можливість пошуку та замовлення обраного постеру, але при авторизації на сайті в ролі адміністратора, у користувача з'являється ряд нових можливостей:

- Можливість перегляду всіх замовлень, отримання інформації користувача для оформлення відправки замовлення.
- Адміністратор має можливість редагувати, або видаляти наявні постери.
- Можливість додавання нового постеру до магазину.

У нашому проєкті ми використовуємо SSR (Server-Side Rendering - рендерінг на стороні сервера) для досягнення більшої продуктивності. SSR використовується для генерації HTML-контенту на сервері, перед його надсиланням на клієнтську сторону. При кожному запиті користувача, сервер генерує відповідну сторінку з необхідними даними та шаблонами і повертає її у вигляді HTML.

Принцип роботи SSR в нашому проєкті полягає в наступному:

- Клієнтський запит надсилається на сервер.
- Сервер отримує запит та виконує необхідні обчислення та запити до бази даних.
- За допомогою шаблонування і відповідних даних, сервер генерує HTML-сторінку з динамічним контентом.
- Сформована HTML-сторінка надсилається назад на клієнтську сторону.
- Клієнт отримує HTML-сторінку та відображає її на стороні браузера.

Такий підхід дозволяє нам зменшити навантаження на клієнтську сторону, оскільки сервер виконує складні обчислення та формування контенту.

Використання SSR в нашому проєкті допомагає забезпечити ефективне та швидке відображення інформації в інтернет-магазині, забезпечуючи зручну взаємодію з додатком. Діаграма компонентів зображена на рисунку 2.7

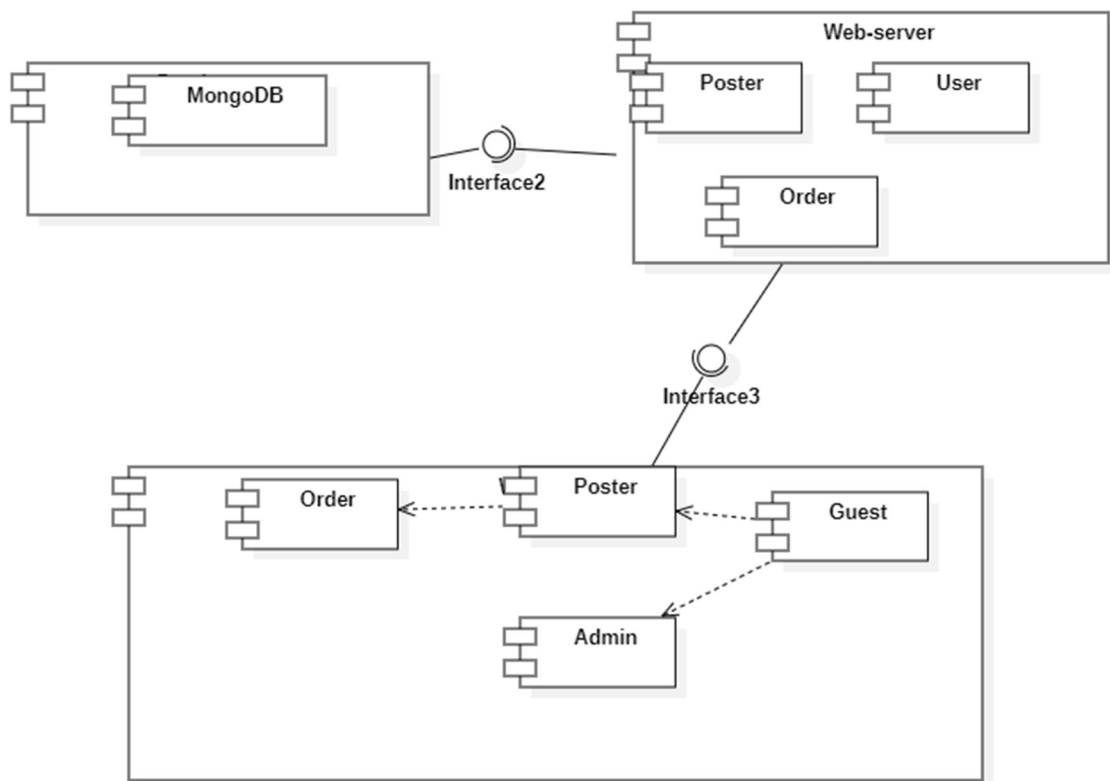


Рисунок 2.7 Діаграма компонентів

2.4 Реалізація додатку інтернет-магазину поліграфічної продукції

Для побудови запитів до приватного API було використано пакет `axios`, для налаштування роутингу – пакет `react-router-dom` і для роботи з формами – `react-hook-form`. Як UI бібліотеку було обрано `Ant Design`.

`Axios` — це HTTP-клієнт на основі обіцянок для `node.js` і браузера [12].

Для зберігання та управління даними використовується база даних `MongoDB`, підключення до якої здійснюється за допомогою бібліотеки `Mongoose`, що надає зручний інтерфейс для роботи з `MongoDB`. Завдяки цьому можна зберігати і витягувати дані про товари, замовлення і користувачів. Підключення до бази даних зображено на рисунку 2.8

```
const uri = process.env.MONGODB_URI

let client = new MongoClient(uri)
let clientPromise

if (!process.env.MONGODB_URI) throw new Error('Please add your Mongo URI to .env.local')

if (process.env.NODE_ENV === 'development') {
  if (!global._mongoClientPromise) {
    global._mongoClientPromise = client.connect()
  }
  clientPromise = global._mongoClientPromise
} else {
  clientPromise = client.connect()
}

export default clientPromise
```

Рисунок 2.8. Підключення до бази даних

Для підключення до бази даних ми спочатку дістаємо значення URI з середовища виконання (`.env`), після чого виконується перевірка цього значення. Якщо URI проходить перевірку, виконується підключення до бази даних, в іншому випадку нам повідомляє про помилку.

Налаштування та оптимізація роботи бази даних відбувається за рахунок функції `dbConnect` зображених на рисунку 2.9

```

async function dbConnect() {
  if (cached.conn) {
    return cached.conn
  }

  if (!cached.promise) {
    const opts = {}

    cached.promise = mongoose.connect(MONGODB_URI, opts).then((mongoose : Mongoose ) => {
      return mongoose
    })
  }
  cached.conn = await cached.promise
  return cached.conn
}

```

Рисунок 2.9. Оптимізація роботи бази даних

Спочатку йде перевірка підключення до бази даних, якщо підключення існує, функція повертає його без створення нового підключення. Цей код допомагає пришвидшити роботу бази даних та оптимізувати роботу з нею.

Для створення та роботи з полями створені схеми, кожна з яких відповідає своїм функціям. На рисунку 2.10 зображено схему Order

```

import mongoose, { Schema } from 'mongoose'

const OrderSchema = new Schema( definition: {
  name: { type: String, required: true },
  address: { type: String, required: true },
  phoneNumber: { type: String, required: true },
  poster: { type: Array, required: true }
})

const Order = mongoose.models.Order || mongoose.model( name: 'Order', OrderSchema)
export default Order

```

Рисунок 2.10 Створення схеми Order

Ця схема відповідає за збереження та роботу з даними замовлень і зберігає в собі ім'я замовника, адресу, номер телефону і замовлені постери.

Також були створені схеми User та Poster, що зберігали в собі дані про користувачів і про постери відповідно.

Для роботи з базою даних було створено аккаунт адміністратора. При вході ми можемо побачити таблицю, яка відображає усі пости та можливість їх редагування та видалення. Функції отримання даних та видалення посту показано на рисунку 2.11.

```
const getPosters = () => {
  axios.get( url: `http://localhost:3000/api/poster/all`).then((response : AxiosResponse<any> ) => {
    console.log(response);
    setDataSource(response.data);
  });
};

const deletePoster = (id) => {
  axios.delete( url: `http://localhost:3000/api/poster/${id}`).then((response : AxiosResponse<any> ) => {
    console.log(response);
  });
};
```

Рисунок 2.11. Функції отримання даних про постер та його видалення.

Для отримання усіх даних ми виконуємо запит до API, після чого нам повертаються усі значення, які є в базі даних. Для видалення посту, виконується запит для видалення посту по його ідентифікатору.

Для редагування посту викликається відповідна форма в якій відбувається зміна потрібних даних. Після зміни дані про постер зберігаються за допомогою функції зображеної на рисунку 2.12.

```
const onEditFinish = (e) => {
  axios
    .patch( url: `http://localhost:3000/api/poster/${editPoster._id}`, data: { ...e }) Promise<AxiosResponse<any>>
    .then((response : AxiosResponse<any> ) => console.log(response)) Promise<void>
    .catch((error) => message.error(error));

  message.success( content: 'Edited!');
  form.resetFields();
  dispatch(setPosterState(!posterState));
  dispatch(setIsOpen( state: false));
};
```

Рисунок 2.12 Функція збереження редагованих даних

Для редагування також використовується бібліотека axios, а саме відправляє запит на оновлення даних конкретного постеру, який знаходиться по його ідентифікатору.

Після розробки функцій для адміністратора, почалась розробка функцій для клієтської сторони. В першу чергу був створений блок, який відображав список усіх продуктів на головній сторінці. За нього відповідає код зображений на малюнку 2.13.

```
<div className='content__items'>
  {isLoading ? (
    [...new Array( arrayLength: 6)].map( (_, index : number ) => <Skeleton key={index} />)
  ) : items.length !== 0 ? (
    items.map((poster) => <PosterBlock {...poster} key={poster._id} />)
  ) : (
    <Result
      status='warning'
      title='По вашому запиту нічого не знайдено!'
      extra={
        <Button
          type='primary'
          key='reload'
          onClick={() => {
            router.reload();
            router.push( url: '/' );
            dispatch(setCategoryId( state: 0 ));
          }}
        >
          Перезавантажити
        </Button>
      }
    >
  )}
</div>
```

Рисунок 2.13 Генерація списку товарів

Для відображення списку товарів, ми створюємо масив з 6 елементів та повертаємо в кожне значення масиву компонент «Skeleton» який відображається поки дані не будуть завантажені. Після успішного завантаження нам повертається масив з компонентів які містять всю інформацію про товар.

Для зручності було створено пошук та сортування за деякими параметрами. На рисунку 2.14 зображено функцію пошуку.

```
function Search() {
  const { setSearchValue } = useContext(searchContext);
  const [value, setValue] = useState( initialState: '');
  const onChangeInput = (event) => {
    setValue(event.target.value);
    updateSearchValue(event.target.value);
  };

  const updateSearchValue = useCallback(
    debounce( func (value) => {
      setSearchValue(value);
    }, wait: 1000),
    deps: []
  );
}
```

Рисунок 2.14 Функція пошуку на сторінці

При введенні даних у поле пошуку викликається функція «OnChangeInput», яка змінює стан «value», після чого викликається функція яка чекає 1 секунду і якщо введені дані не змінюються передає дані до методу який виконує пошук.

Також було створено сторінку оформлення замовлення. Для полегшення оформлення замовлення було вирішено використати API Нової пошти для отримання списку міст та відділень [2]. Це полегшить клієнту етап замовлення доставки. На рисунку 2.15 зображено використання API Нової пошти.

```
useEffect( effect: () => {
  axios.post( url: 'https://api.novaposhta.ua/v2.0/json/Address/getCities', dataCities, config: {headers:{
    'Content-Type': 'application/json', 'Accept':'application/json'
  }}).then(res=>setOptions(res.data.data.map(el=>({value: el.Description}))))
}, deps: [])
const getWareHouse = (e, value)>{
  axios.post( url: 'https://api.novaposhta.ua/v2.0/json/Address/getWarehouses', data: {
    "apiKey": "e3c8dc329757354d4bfaf898a8659e40",
    "modelName": "Address",
    "calledMethod": "getWarehouses",
    "methodProperties": {
      "CityName" : value.value,
    }
  }, config: {headers:{
    'Content-Type': 'application/json', 'Accept':'application/json'
  }}).then(res=>setWarehouses(res.data.data.map(el=>({value: el.Description}))))
}
```

Рисунок 2.15 Використання API Нової пошти

Для пошуку міста та відділень виконується запит до API і отримує дані про всі міста та відділення. Після отримання усіх даних про міста, ми передаємо значення назви міста у запит з пошуку відділень, після чого усі дані передаються у форму оформлення замовлення.

Після заповнення усіх даних проходить перевірка даних, якщо все відповідає параметрам, вся інформація про замовлення передається до бази даних і відображається у таблиці замовлень.

Висновки до другого розділу

В даному розділі було проведено розгляд визначення варіантів використання та об'єктно-орієнтованої структури системи, а також розробки бази даних. Згідно аналізу вимог, була побудована діаграма варіантів використання продукту. Для зберігання даних була обрана база даних MongoDB, яка працює у форматі JSON і забезпечує легку обробку та розуміння даних. Були створені три таблиці: User для зберігання даних входу адміністратора, Poster для збереження даних про товари та Orders для збереження даних замовлень.

Початково була розроблена діаграма активності користувачів, яка дозволяє зменшити навантаження на клієнтську сторону шляхом виконання складних обчислень та формування контенту на сервері. Для реалізації додатку інтернет-магазину були використані пакети axios для запитів до приватного API, react-router-dom для налаштування роутингу та react-hook-form для роботи з формами. UI бібліотекою було обрано Ant Design. Також було забезпечено підключення до бази даних MongoDB за допомогою бібліотеки Mongoose, що спрощує роботу з даними про товари, замовлення та користувачів.

Було розглянуто створення схем для збереження та роботи з даними замовлень, користувачів та товарів. Також було наведено приклади функцій для пошуку товарів та використання API Нової пошти для отримання списку міст та відділень для оформлення доставки замовлень.

РОЗДІЛ 3 ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ ІНТЕРНЕТ-МАГАЗИНУ ПОЛІГРАФІЧНОЇ ПРОДУКЦІЇ

3.1. Порядок встановлення та налаштування параметрів системи

Наш інтернет-магазин поліграфічної продукції є веб-орієнтованою платформою, розробленою з урахуванням сумісності з різними браузерами. Цей веб-сайт дозволяє нашим клієнтам зручно користуватися ним на різних пристроях, таких як телефони, планшети і ноутбуки.

Звернемо увагу, що різні користувачі можуть мати різні пристрої з різними характеристиками, тому наш інтернет-магазин має бути доступний і функціональний для якомога більшої кількості користувачів. Для використання нашого сайту нам потрібен телефон, планшет або ноутбук, які відповідають наступним мінімальним вимогам:

- Операційна система: Android, iOS, Windows або macOS.
- Браузер: сумісність з останніми версіями популярних веб-браузерів, таких як Google Chrome, Mozilla Firefox, Safari або Microsoft Edge.
- Інтернет-з'єднання: стабільне підключення до Інтернету з мінімальною швидкістю передачі даних для безперебійної роботи сайту і зручного перегляду продукції.
- ОЗУ: 2ГБ
- Процесор: рекомендується 1,6 ГГц або швидше.

Отже, розроблений додаток не потребує специфічного ПЗ для його використання. Все що нам потрібно – це браузер і комп'ютер або мобільний пристрій.

На рисунку 3.1 зображено діаграму розгортання додатку

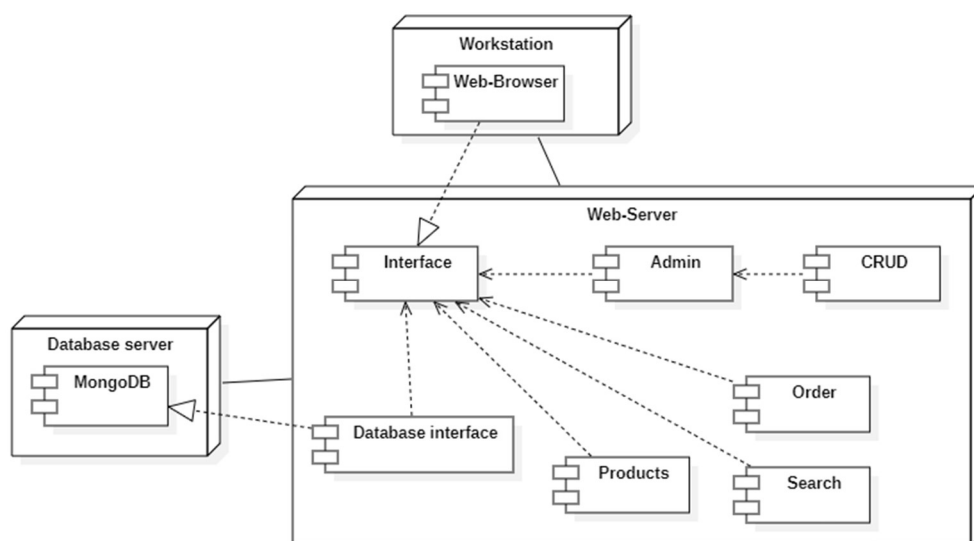


Рисунок 3.1 Діаграма розгортання додатку

Для розгортання додатку користувачу необхідний лише комп'ютер або телефон та інтернет з'єднання.

Вся обробка інформації, її зберігання і налаштування відбувається на сервері після чого користувач отримує готовий результат.

3.2 Структура інтерфейсу та порядок роботи

Для зручного користування інтерфейс має бути інтуїтивно зрозумілим і простим у використанні. Користувач має розуміти які послуги надає наш додаток і після користування сайтом має бути задоволений.

Для створення інтерфейсу ми використовували бібліотеку компонентів інтерфейсу Ant Design, так як вона пропонує широкий набір готових компонентів, які можна використовувати для створення різноманітних елементів інтерфейсу, таких як кнопки, форми, таблиці, навігаційні панелі, модальні вікна та багато інших.

Крім компонентів, Ant Design надає також набір інструментів для розробки, таких як система розмітки, іконки, кольорові схеми та інші корисні ресурси. Завдяки своїй гнучкості, простоті використання та сучасному дизайну, ця бібліотека є чудовим вибором для розробки інтерфейсу нашого додатку.

При першому візиті на сайт, можемо побачити головну сторінку на якій зображені популярні серед замовників постери та навігацію по жанрам. Також ми можемо відсортувати товари по ціні та популярності, або знайти потрібний постер у полі для пошуку. Головне меню зображене на малюнку 3.2.

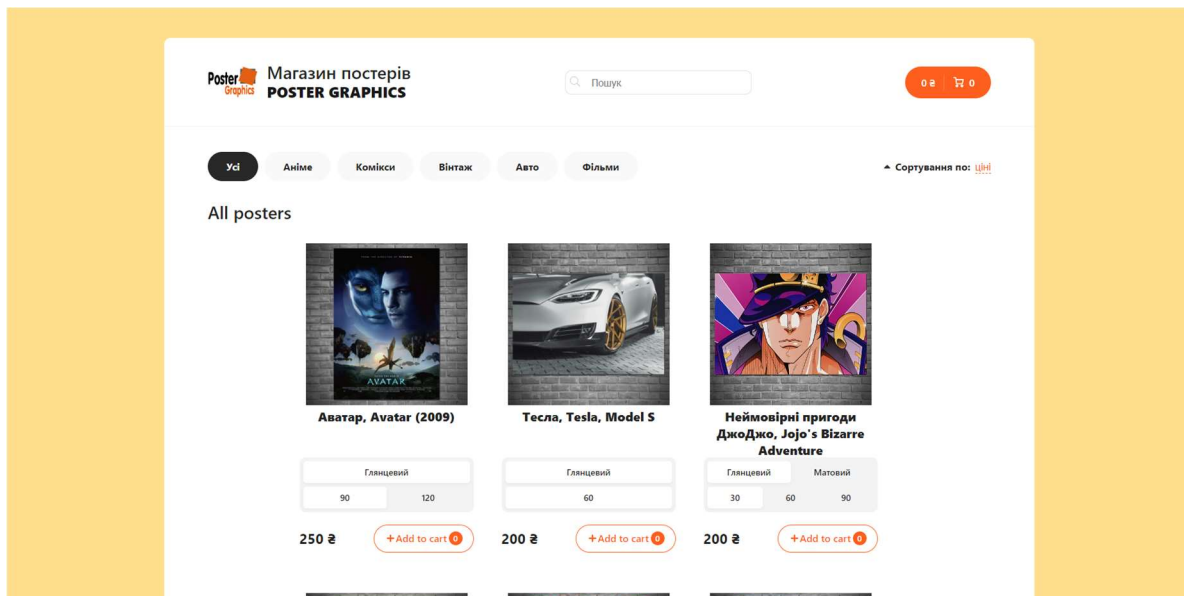


Рисунок 3.2. Головне меню сайту

Обравши потрібний постер, ми можемо натиснути кнопку «Add to cart», щоб додати товар до корзини. Також можемо обрати тип поверхні та розмір постеру.

Якщо ми не додаємо товар до корзини і перейдемо до неї то побачимо повідомлення про відсутність товару. Це зображено на рисунку 3.3.

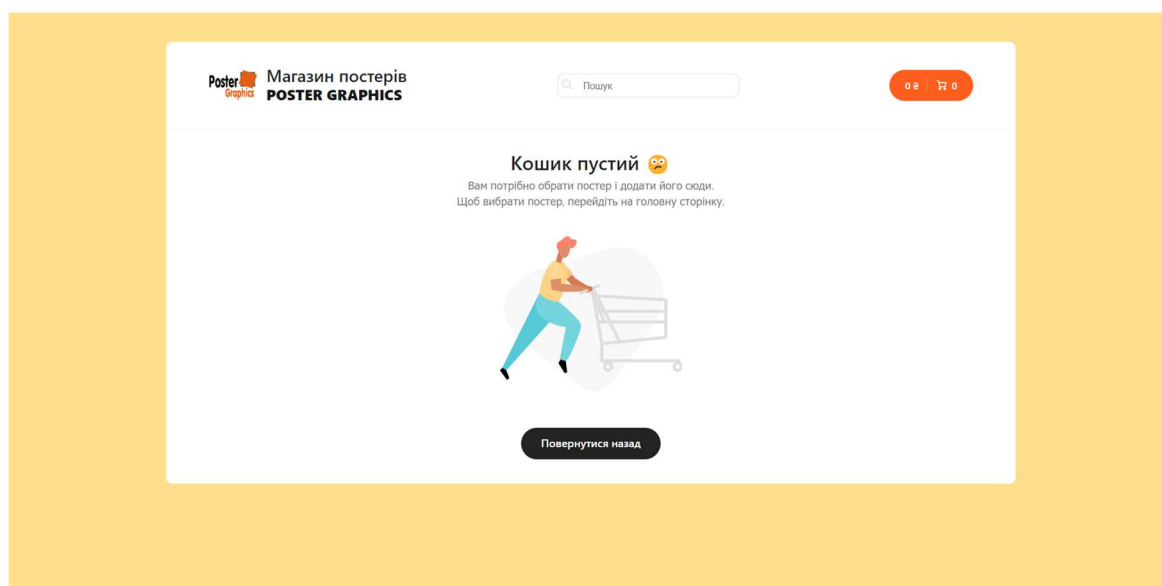


Рисунок 3.3 Порожня корзина.

У разі, якщо постер доданий до корзини, ми бачимо всю інформацію про нього, та можливість оформити замовлення. Корзина з продуктом зображена на рисунку 3.4.

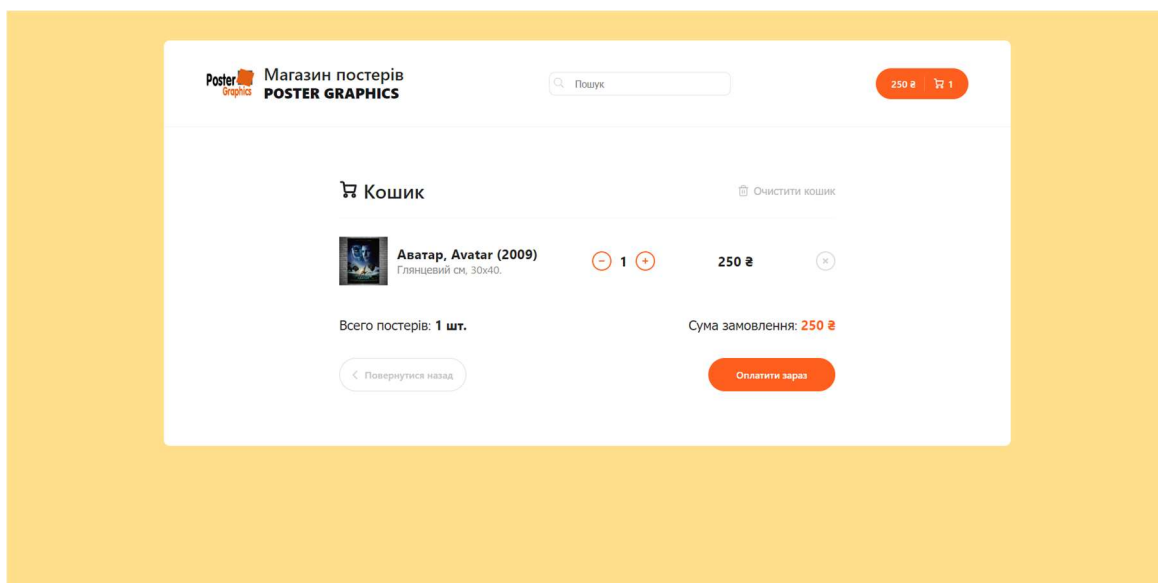


Рисунок 3.4. Кошик з обраним продуктом

У самому кошику ми можемо збільшити кількість постерів для замовлення, а також видалити його з корзини, або повернутися назад для додавання інших товарів. Після натискання на кнопку «Оформити замовлення», відкриється форма для заповнення, в якій потрібно вказати Ім'я, номер телефону, місто, номер відділення Нової пошти, електронну адресу. На рисунку 3.5 зображено форму для оформлення замовлення.

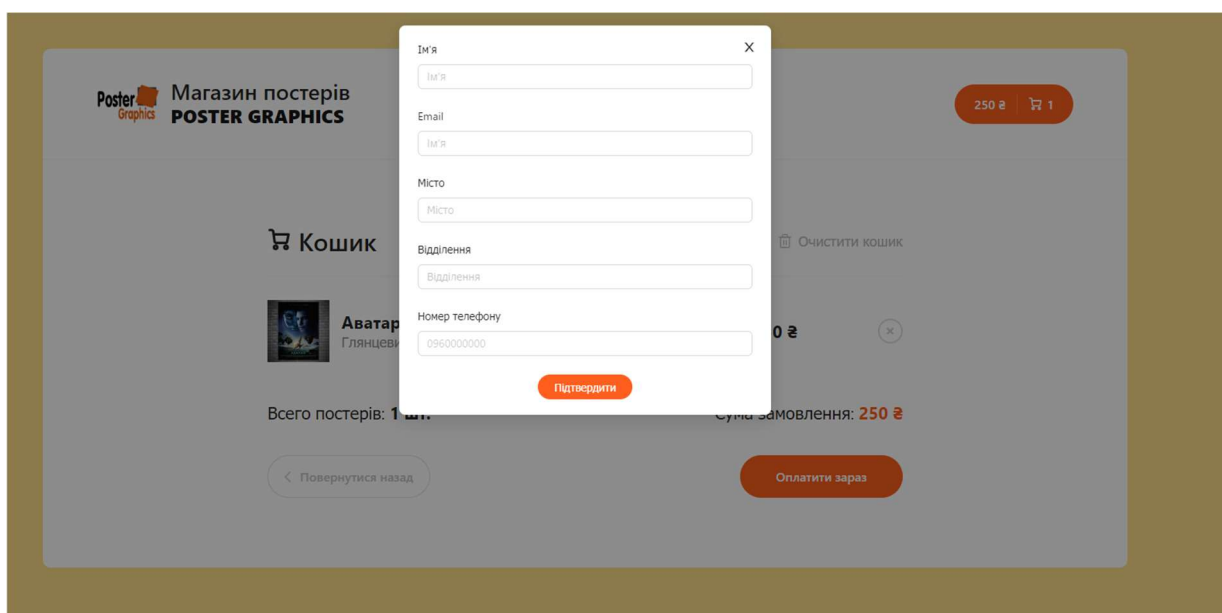


Рисунок 3.5. Форма оформлення замовлення

При вводі назви міста, з'являється випадаючий список зі всіма назвами міст які підходять до введеного тексту. Приклад зображений на рисунку 3.6.

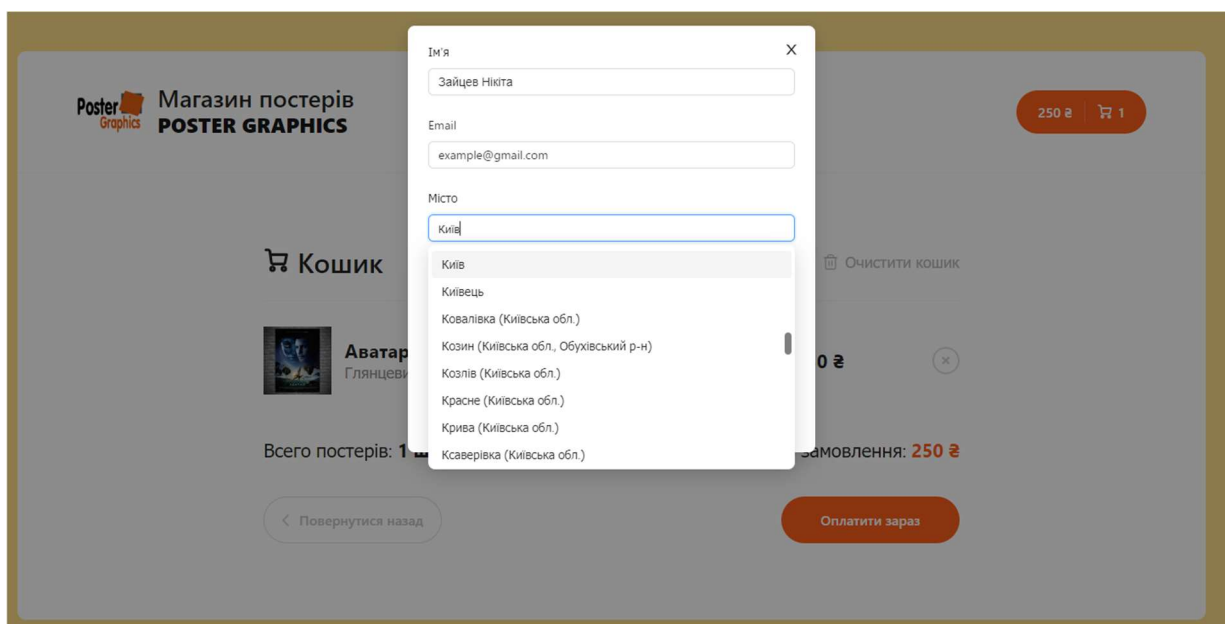


Рисунок 3.6 Вибір міста для відправки

Після вибору міста, потрібно обрати номер відділення, або поштомату, інформація про які автоматично підтягується у наступному полі для заповнення. Приклад вибору відділення зображений на рисунку 3.7.

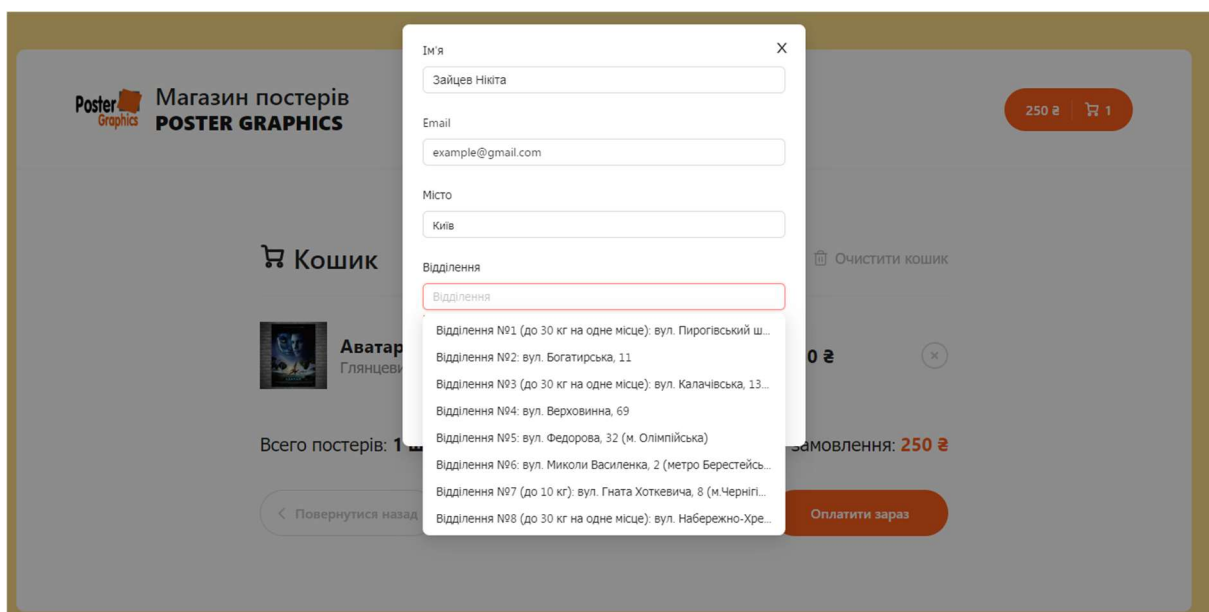


Рисунок 3.7 Вибір відділення

Після вибору відділення, потрібно вказати номер телефону. Якщо номер не вказаний, або вказаний невірно ми отримаємо повідомлення про це. Зображено на рисунку 3.8.

Ім'я
Зайцев Нікіта

Email
example@gmail.com

Місто
Київ

Відділення
Відділення №1 (до 30 кг на одне місце): вул. Пирогівський шлях, 1:

Номер телефону
0960000000
Введіть номер телефону

Підтвердити

Рисунок 3.8 Попередження про невірно вказаний номер

При вірному заповненні усіх даних ми можемо успішно оформити замовлення, після чого відбудеться перехід на головну сторінку та з'явиться повідомлення про успішне оформлення, зображене на рисунку 3.9.

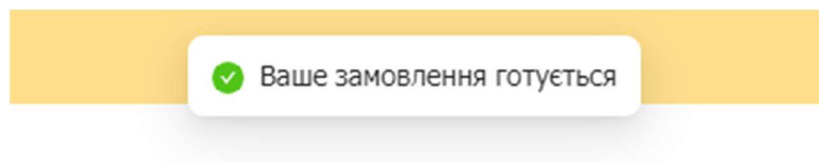


Рисунок 3.9. Повідомлення про успішне оформлення

Також є можливість авторизуватись як адміністратор, тоді нам відкриються нові функції яких немає у користувачів. Для того щоб авторизуватись, там потрібно перейти вказати логін та пароль в формі авторизації зображеній на рисунку 3.10.

Рисунок 3.10. Форма авторизації адміністратора

Після успішної авторизації, ми можемо побачити панель управління адміністратора, в якій виконуються головні функції роботи з товарами. На рисунку 3.11 зображена панель управління.

Назва	Ціна	Розміри	Категорія	Тип	Рейтинг	Кількість замовлень	Прибуток	
Аватар, Avatar (2009)	250	90, 120	Фільми/DC	Глянцевий	4.5	104	26000 ₴	
Гон, Hunter x Hunter, Gon1	180	30, 60, 90	Аніме	Матовий Глянцевий	4.5	50	9000 ₴	
Атака титанів, поява титану-копоса, Attack on Titans	180	30, 60	Аніме	Матовий	4.0	0	0 ₴	
Неймовірні пригоди ДжоДжо, JoJo's Bizarre Adventure	200	30, 60, 90	Аніме	Глянцевий Матовий	4.0	0	0 ₴	
Вбивця демонів, Kimetsu no Yaiba	180	60, 90	Аніме	Матовий	4.0	6	1080 ₴	
Людина-Павук, Spider-Man, Avengers: Infinity War (2018)	180	60, 90	Фільми/DC	Глянцевий Матовий	4.0	0	0 ₴	
Всього: 33 614 714	1 000	30	1 000	Матовий	4.5	1	10	

Рисунок 3.11. Панель управління адміністратора

Адміністратор має такі можливості як: додавання нових постерів, редагування та видалення постерів, перегляд усіх замовлень.

Для додавання нового постеру, потрібно натиснути кнопку «Add a poster», після чого з'явиться форма створення постеру (рис. 3.12).

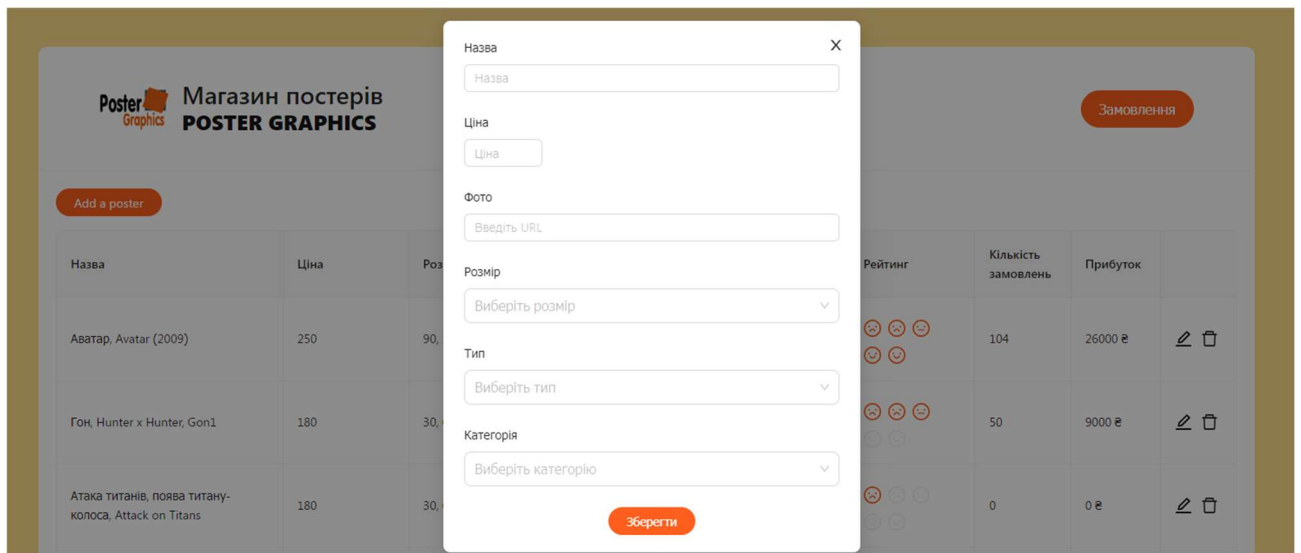


Рисунок 3.12. Створення нового постеру

Для створення нам потрібно ввести назву постеру, його ціну, вказати посилання на зображення, вибрати розмір та тип постеру і обрати його категорію (рис. 3.13).

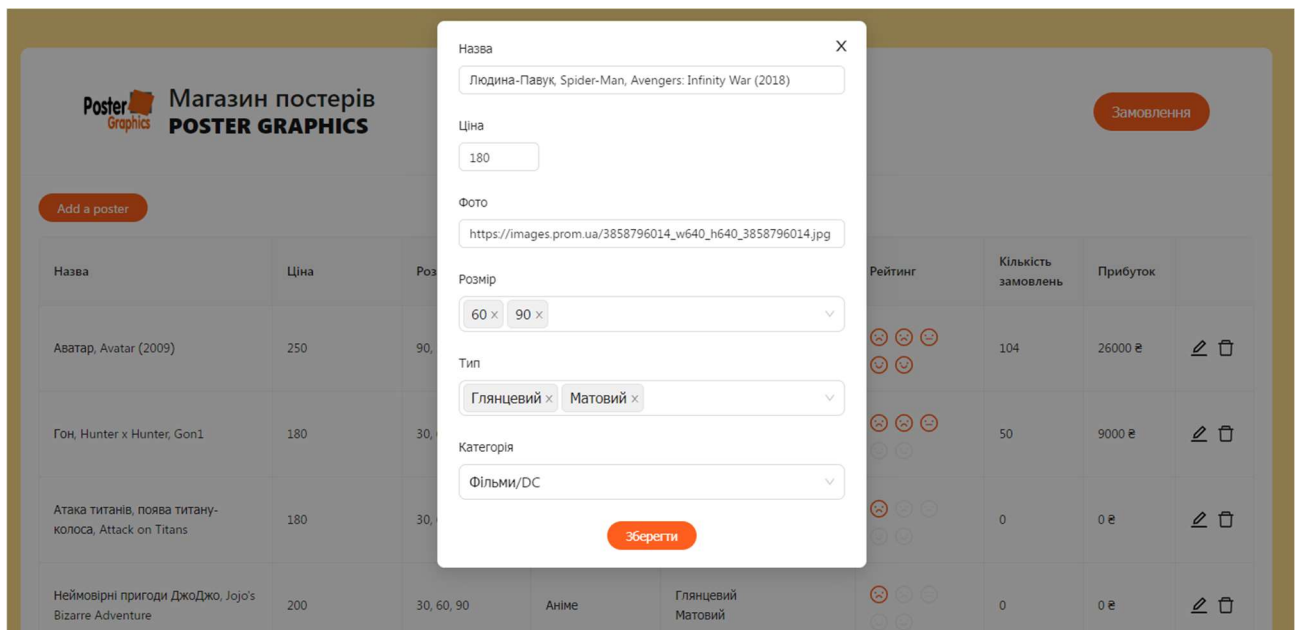


Рисунок 3.13. Постер готовий для створення

Після заповнення усіх даних, товар з'явиться у списку усіх товарів в головному меню, а також його можна буде знайти по назві, або категорії (рис. 3.14).

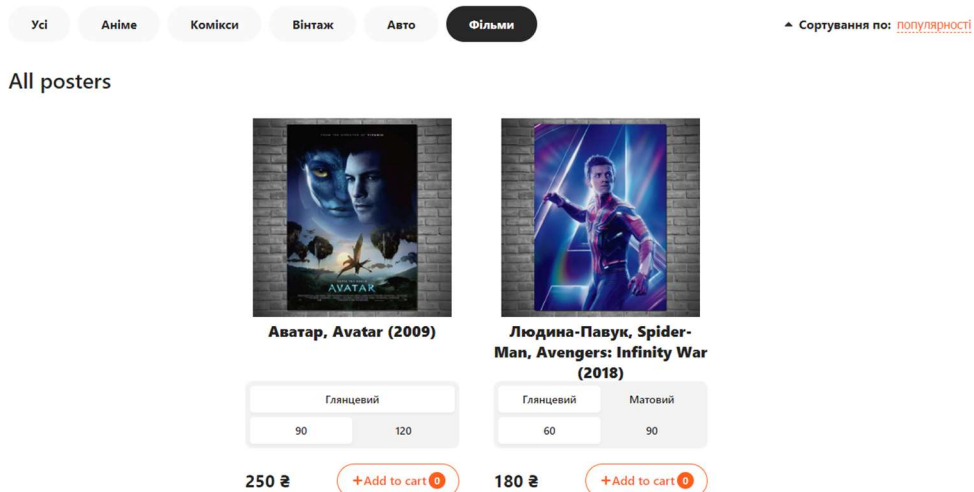


Рисунок 3.14. Створений постер у відповідній категорії

Щоб видалити постер з каталогу адміністратору потрібно натиснути відповідну кнопку та підтвердити видалення (рис. 3.15).

Родстер BMW Z4	180	90	Авто	Глянцевий		0	0 ₴	
Тесла, Tesla, Model S	200	60	Авто	Глянцевий		0	0 ₴	

Рисунок 3.15. Видалення постеру

Для перегляду усіх замовлень, адміністратор має натиснути кнопку «Замовлення» на панелі управління. Буде зображена таблиця в якій відображаються дані про всі оформленні замовлення (рис.3.16).

Магазин постерів POSTER GRAPHICS				Постери
Ім'я	Адреса	Номер телефону	Замовлення	
nikita zaytsev	Гошів (Житомирська обл.), Пункт приймання-видачі (до 30 кг.), вул. Центральна, 26	0968265868	Гон. Hunter x Hunter; Gon: 1, 20x40, Глянцевий	
Wain	Гульськ (Житомирська обл.), Пункт приймання-видачі (до 30 кг.), вул. Центральна, 26	0968265868	Гон. Hunter x Hunter; Gon: 1, 20x40, Глянцевий	
Максим	Житомир, Відділення №12: площа Скопська, 3	0968265868	Гон. Hunter x Hunter; Gon: 1, 20x40, Глянцевий	
nikita zaytsev	Гошів (Житомирська обл.), Пункт приймання-видачі (до 30 кг.), вул. Центральна, 26	0968265868	Аватар, Avatar (2009); 100, 20x40, Глянцевий	
Зайцев Нікіта	Київ, Відділення №1 (до 30 кг на одне місце) вул. Пирогівський шлях, 135	0968265868	Аватар, Avatar (2009); 1, 30x40, Глянцевий	
nikita zaytsev	Гранітне (Житомирська обл.), Пункт приймання-видачі (до 30 кг.), вул. Чехова, 7	0968265868	Гон. Hunter x Hunter; Gon: 1, 30x40, Глянцевий Аватар, Avatar (2009); 1, 30x40, Глянцевий Вбивця демонів, Kimetsu no Yaiba; 1, 30x40, Глянцевий	
nikita zaytsev	Гошів (Житомирська обл.), Пункт приймання-видачі (до 30 кг.), вул. Центральна, 26	0968265868	Аватар, Avatar (2009); 2, 30x40, Глянцевий Гон. Hunter x Hunter; Gon: 1, 30x40, Матовий	

Рисунок 3.16. Список усіх замовлень

3.3 Тестування роботи програмного комплексу

Тестування програмного забезпечення є невід'ємною частиною процесу розробки програмного забезпечення, зосереджене на виявленні помилок і виконанні низки дій для перевірки функціональності, надійності та продуктивності програмних продуктів перед їх випуском або експлуатацією.

Мета тестування – переконатися, що якість програмного продукту гарантована, надійна і відповідає вимогам користувача. Процес включає створення тестових кейсів, виконання тестів, запис результатів та аналіз даних.

Все це допоможе вам зрозуміти важливість тестування програмного забезпечення, його роль у процесі розробки та випуску, а також у забезпеченні високої якості програмного забезпечення для користувачів.

Для відображення тестування побудуємо тест-кейси.

Тест-кейс - це документований набір кроків, які необхідно виконати для перевірки певної функціональності або властивості програмного продукту. Він містить інструкції для тестувальника щодо введення початкових даних, виконання конкретних дій і перевірки очікуваних результатів.

Основна мета тест-кейсу полягає в стандартизації тестування і забезпеченні повторюваності процесу. Він дозволяє тестувальникам систематично перевіряти різні аспекти програмного продукту та виявляти помилки або неполадки.

В таблиці 3.1 наведено створені тест кейси.

Таблиця 3.1

Тестування додатку

Опис кейсу	Кроки виконання	Дані кейсу	Очікування	Результат
Пошук продукту	1.Переходимо на сторінку товарів	Поле введення – «Gon»	Виведення відповідних	Як і очікувалось

	2.Вводимо дані для пошуку		постерів згідно пошуку	
Додавання продукту в кошик	1.Обрати постер 2.Додати товар в кошик		Товар додається до кошику	Як і очікувалось
Оформлення замовлення	1.Обрати постер 2.Додати товар у кошик 3.Перейти до сторінки оформлення замовлення 4 Заповнити усі потрібні дані	Ім'я – Іван Іванов Email – example@gmail.com Місто – Житомир Відділення – 11 Номер телефону – 0960000000	Замовлення успішно оформлюється	Як і очікувалось
Авторизація адміністратора	1.Перейти на сторінку авторизації	Логін – Admin Пароль – Admin1234	Відбувається авторизація	Як і очікувалось
Створення нового товару	1.Перейти на сторінку адміністратора 2.Перейти до створення нового постеру	Назва – Аватар Ціна – 250 Посилання на зображення Розмір – 30,60,90 Тип– матовий, глянцевий Категорія – Аніме	Новий постер додається до списку усіх постерів	Як і очікувалось

	3.Заповнити усі необхідні дані			
Редагування товару	1.Перейти на сторінку адміністратора 2.Перейти до редагування постеру 3.Заповнити усі необхідні дані	Категорія – фільми	Інформація про постер оновлюється	Як і очікувалось
Перегляд замовлень	1.Перейти на сторінку адміністратора 2.Перейти до сторінка з замовленнями		Отримання інформації про всі замовлення	Як і очікувалось

Так, для запобігання помилок, коли користувач вводить неправильну або неіснуючу адресу сайту, використовується 404 сторінка. 404 сторінка (також відома як "Сторінка не знайдена" або "Error 404") використовується для повідомлення користувачеві, що запитаний ресурс не знайдено на сервері.

Ця сторінка має на меті забезпечити позитивний користувацький досвід, вказуючи на те, що сталася помилка, але також надаючи користувачеві деяку інформацію та додаткові посилання для навігації по сайту.

Для тестування додатку був використаний інструмент для тестування Postman [18]. Виконавши кілька запитів, ми перевірили правильність отримання даних з додатку і впевнилися, що вони повертаються відповідно до очікувань.

Спочатку ми протестували отримання постерів відсортованих за рейтингом (рис. 3.17).

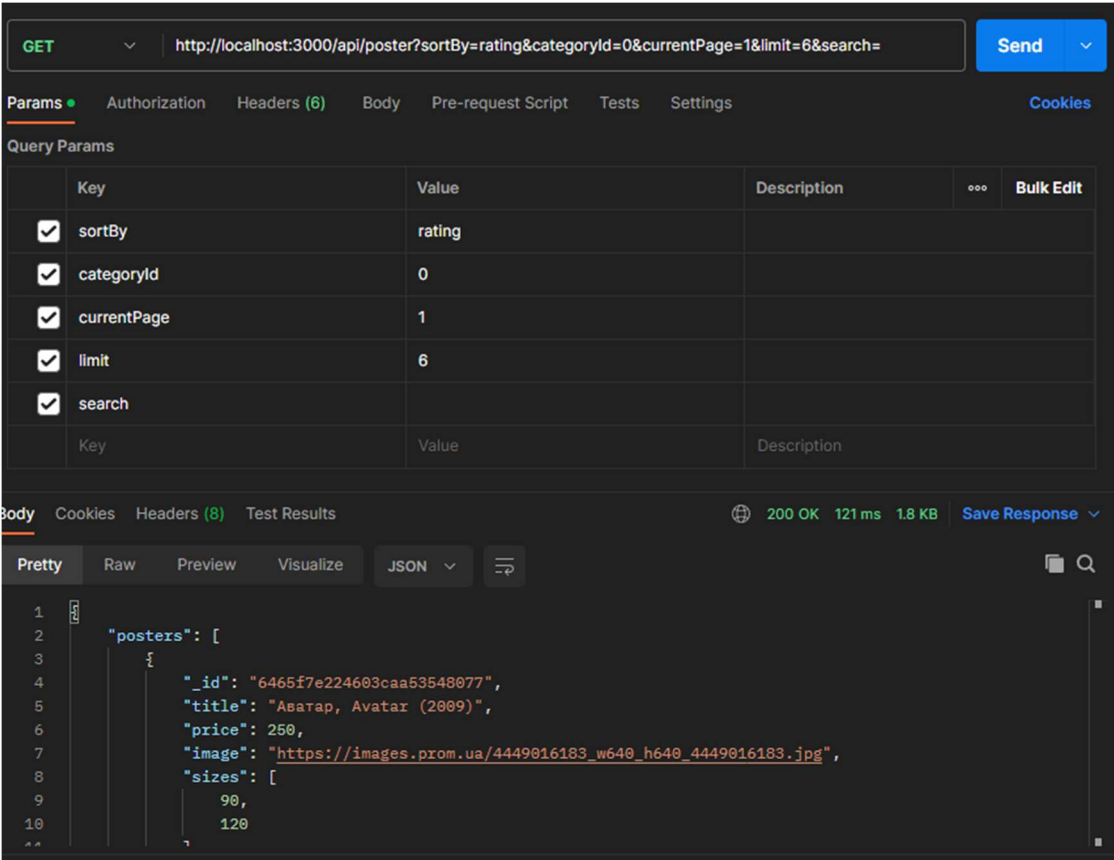


Рисунок 3.17. Запит на отримання даних відсортованих за рейтингом

Після успішного отримання даних цього запиту ми вирішили перевірити отримання даних про міста з API Нової пошти (рис. 3.18).

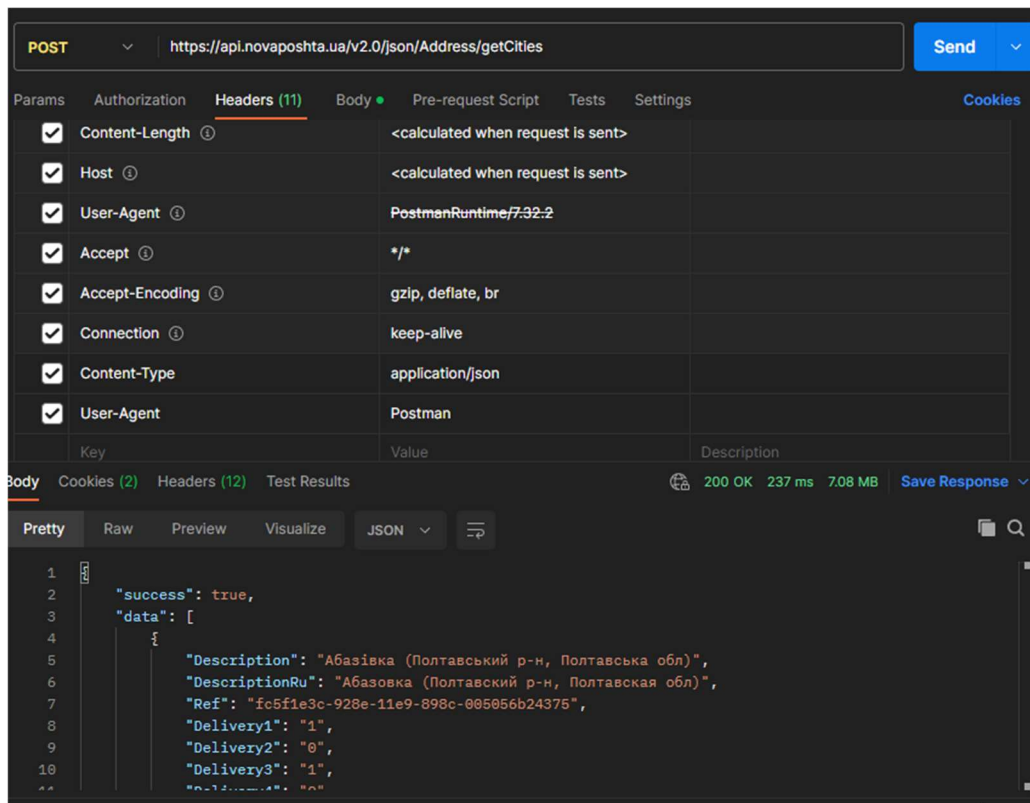


Рисунок 3.18. Запит для отримання назв міст

Висновки до третього розділу

Після аналізу інтерфейсу та порядку роботи з інтернет-магазином поліграфічної продукції можна зробити наступні висновки. Інтерфейс додатку є інтуїтивним і зрозумілим, що дозволяє користувачам легко навігувати та здійснювати покупки. Дизайн інтерфейсу є привабливим та естетичним, створюючи позитивне враження. Користувачам надається можливість швидкого пошуку та фільтрування продуктів за різними критеріями. Процес оформлення замовлення є зручним та простим, з можливістю внесення необхідних даних та вибору доставки та способу оплати. Доступність контактної інформації та підтримки користувачів сприяє забезпеченню якісного обслуговування та вирішенню потенційних питань. Загалом, інтерфейс та порядок роботи з інтернет-магазином поліграфічної продукції відповідають сучасним стандартам та вимогам, забезпечуючи задоволення користувачів та ефективність покупок.

Було проведено тестування програмного продукту та перевірка усіх функцій додатку.

ВИСНОВКИ

Під час виконання випускної кваліфікаційної роботи бакалавра було спроектовано та реалізовано інтернет-магазин поліграфічної продукції.

У першому розділі нашої кваліфікаційної роботи ми провели детальний аналіз використання інформаційних технологій для розробки інтернет-магазину поліграфічної продукції.

Також було проведено аналіз аналогів за даною тематикою. Було визначено необхідний функціонал продукту.

Було обрано фреймворк для розробки Next.js.

В другому розділі було проведено розгляд визначення варіантів використання та об'єктно-орієнтованої структури системи, а також розробки бази даних. Для зберігання даних була обрана база даних MongoDB, яка працює у форматі JSON і забезпечує легку обробку та розуміння даних.

Була розроблена діаграма активності користувачів, яка дозволяє зменшити навантаження на клієнтську сторону шляхом виконання складних обчислень та формування контенту на сервері. UI бібліотекою було обрано Ant Design. Також було забезпечено підключення до бази даних MongoDB за допомогою бібліотеки Mongoose, що спрощує роботу з даними про товари, замовлення та користувачів.

Було розглянуто створення схем для збереження та роботи з даними замовлень, користувачів та товарів. Також було наведено приклади функцій для пошуку товарів та використання API Нової пошти для отримання списку міст та відділень для оформлення доставки замовлень.

У третьому розділі було показано порядок роботи з інтернет магазином.

Загалом, інтерфейс та порядок роботи з інтернет-магазином поліграфічної продукції відповідають сучасним стандартам та вимогам, забезпечуючи задоволення користувачів та ефективність покупок.

Було проведено тестування програмного продукту та перевірка усіх функцій додатку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. StarUML documentation: Introduction. URL: <https://docs.staruml.io>.
2. Документація API Nova Poshta [Електронний ресурс] – Режим доступу до ресурсу: <https://developers.novaposhta.ua/documentation>.
3. Документація Ant Design [Електронний ресурс] – Режим доступу до ресурсу: <https://ant.design/docs/react/use-with-next>.
4. Документація React JS [Електронний ресурс] – Режим доступу до ресурсу: <https://uk.reactjs.org/>.
5. Документація Next.js [Електронний ресурс] – Режим доступу до ресурсу: <https://nextjs.org/docs>.
6. Server-Site-Rendering [Електронний ресурс] – Режим доступу до ресурсу: <https://nextjs.org/docs/pages/building-your-application/rendering/server-side-rendering>
7. Static-Site-Generation [Електронний ресурс] – Режим доступу до ресурсу: <https://nextjs.org/docs/pages/building-your-application/rendering/static-site-generation>
8. Client-Site-Rendering [Електронний ресурс] – Режим доступу до ресурсу: <https://nextjs.org/docs/pages/building-your-application/rendering/client-side-rendering>
9. Документація MongoDB [Електронний ресурс] – Режим доступу до ресурсу: <https://www.mongodb.com/docs/>
10. Архітектура програмного забезпечення [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Software_architecture
11. Документація SCSS [Електронний ресурс] – Режим доступу до ресурсу: <https://sass-lang.com/documentation/>
12. Документація Axios [Електронний ресурс] – Режим доступу до ресурсу: <https://axios-http.com/docs/intro>
13. Redux Toolkit Quick Start [Електронний ресурс] – Режим доступу до ресурсу: <https://redux.js.org/tutorials/quick-start>

14. Документація JavaScript [Електронний ресурс] – Режим доступу до ресурсу: <https://developer.mozilla.org/en-US/docs/Web/JavaScript>
15. Next.js useRoute [Електронний ресурс] – Режим доступу до ресурсу: <https://nextjs.org/docs/pages/api-reference/functions/use-router>
16. JavaScript [Електронний ресурс] – Режим доступу до ресурсу: <https://uk.wikipedia.org/wiki/JavaScript>
17. Introducing JSON [Електронний ресурс] – Режим доступу до ресурсу: <https://www.json.org/json-en.html>
18. Документація Postman [Електронний ресурс] – Режим доступу до ресурсу: <https://www.postman.com/product/tools/>

ДОДАТКИ

					ІІЗ.КР.Б – 121 – 23 – ІІЗ	
						.
						52

Лістинг програми:

Home.jsx

```

function Home() {
  const router = useRouter();
  const { push } = useRouter();
  const dispatch = useDispatch();
  const isSearch = useRef( initialValue: false);
  const isMounted = useRef( initialValue: false);
  const { categoryId, sort, currentPage } = useSelector(
    selector: (state) => state.filter
  );
  const sortType = sort.sortName;
  const { searchValue } = useContext(searchContext);
  const [items, setItems] = useState( initialState: []);
  const [total, setTotal] = useState( initialState: 0);
  const [isLoading, setIsLoading] = useState( initialState: true);

  const onClickCategory = (id) => {
    dispatch(setCategoryId(id));
    dispatch(setCurrentPage( state: 1));
  };

  const getPosters = () => {
    setIsLoading( value: true);

    axios
      .get(
        url: `http://localhost:3000/api/poster?sortBy=${sort.sortName}&categoryId=${categoryId}&currentPage=${currentPage}&limit=6&search=${searchValue}`
      )
      .then((response : AxiosResponse<any> ) => {
        setItems(response.data.posters);
        setTotal(response.data.total);
        setIsLoading( value: false);
      });
  };

  useEffect( effect: () => {
    getPosters();
    window.scrollTo( x: 0, y: 0);
  }, deps: [categoryId, sortType, searchValue, currentPage, router.pathname]);

  useEffect( effect: () => {
    if (isMounted.current) {
      const query = qs.stringify( object: {
        sortProperty: sort.sortName,
        categoryId,
        currentPage,
      });

      push( url: `?${query}`);
    }

    isMounted.current = true;
  }, deps: [categoryId, sortType, currentPage]);

  useEffect( effect: () => {
    if (window.location.search) {
      const params = qs.parse(window.location.search.substring(1));
      const sort = sortBy.find((obj : {name: string, sortName: string} ) => obj.sortName === params.sortProperty);
      dispatch(setFilters( state: { ...params, sort }));

      isSearch.current = true;
    }
  }, deps: []);
}

```

```

    {items.length !== 0 && total !== 0 && (
      <Pagination
        style={{
          display: 'flex',
          justifyContent: 'center',
        }}
        current={currentPage}
        onChange={(number :number ) => dispatch(setCurrentPage(number))}
        total={categoryId === 0 ? total : items.length}
        pageSize={6}
      />
    )}
  </div>
</div>
);
}

```

export default Home;

Order.js

```

import mongoose, { Schema } from 'mongoose'

const OrderSchema = new Schema( definition: {
  name: { type: String, required: true },
  address: { type: String, required: true },
  phoneNumber: { type: String, required: true },
  poster: { type: Array, required: true }
})

const Order = mongoose.models.Order || mongoose.model( name: 'Order', OrderSchema)
export default Order

```

User.js

```

import mongoose, { Schema } from 'mongoose'

const UserSchema = new Schema( definition: {
  username: { type: String, required: true },
  password: { type: String, required: true }
})

const User = mongoose.models.User || mongoose.model( name: 'User', UserSchema)
export default User

```

Poster.js

```
import mongoose, { Schema } from 'mongoose'

const PosterSchema = new Schema( definition: {
  title: { type: String, required: true },
  price: { type: Number, required: true },
  image: { type: String, required: true },
  sizes: [{ type: Number }],
  types: [{ type: Number }],
  category: { type: Number, required: true },
  rating: { type: Number, required: true },
  ordersCount: { type: Number, required: true }
})

const Poster = mongoose.models.Poster || mongoose.model( name: 'Poster', PosterSchema)
export default Poster
```

[id].js

```

import start from '../../common/lib/utils/start';
import Poster from '../../common/models/Poster';

start();

export default async function handler(req, res) {
  switch (req.method) {
    case 'GET':
      try {
        const poster = await Poster.findById(req.query.id);
        return res.status(201).json(poster);
      } catch (error) {
        return res.status(400).json(error);
      }
    case 'PATCH':
      try {
        const poster = await Poster.findByIdAndUpdate(req.query.id, req.body);
        return res.status(201).json(poster);
      } catch (error) {
        return res.status(400).json(error);
      }
    case 'DELETE':
      try {
        const poster = await Poster.findByIdAndDelete(req.query.id);
        return res.status(201).json(poster);
      } catch (error) {
        return res.status(400).json(error);
      }
    default:
      res.setHeader('Allow', ['GET', 'PATCH']);
      res.status(425).end(`Method ${req.method} is not allowed.`);
  }
}

```

All.js

```

import start from '../common/lib/utils/start';
import Poster from '../common/models/Poster';

start();

export default async function handler(req, res) {
  switch (req.method) {
    case 'GET':
      try {
        const poster = await Poster.find( filter: {} ).sort( arg: { rating: -1 } );
        return res.status(200).json(poster);
      } catch (error) {
        return res.status(400).json(error);
      }
    default:
      res.setHeader('Allow', ['GET']);
      res.status(425).end(`Method ${req.method} is not allowed.`);
  }
}

```

addPosterSlice.js

```

import { createSlice } from "@reduxjs/toolkit";

const initialState = {
  isOpen: false,
  posterState: false,
};

const addPosterSlice = createSlice( options: {
  name: "addPoster",
  initialState,
  reducers: {
    setIsOpen(state, action){
      state.isOpen = action.payload
    },
    setPosterState(state, action){
      state.posterState = action.payload
    },
  },
});

export const { setIsOpen, setPosterState } = addPosterSlice.actions;

export default addPosterSlice.reducer;

```

CartSlice.js

```
1  import { createSlice } from "@reduxjs/toolkit";
2
3  const initialState = {
4    totalPrice: 0,
5    totalCount: 0,
6    items: [],
7  };
8
9  const cartSlice = createSlice({
10   name: "cart",
11   initialState,
12   reducers: {
13     addItem(state, action) {
14       const findItem = state.items.find((obj) => obj.id === action.payload.id);
15
16       if (findItem) findItem.count++;
17       else {
18         state.items.push({
19           ...action.payload,
20           count: 1,
21         });
22       }
23
24       state.totalPrice = state.items.reduce(
25         (sum, obj) => obj.price * obj.count + sum,
26         0
27       );
28       state.totalCount++;
29     },
30     minusItem(state, action) {
31       const findItem = state.items.find((obj) => obj.id === action.payload.id);
32
33       if (findItem) {
34         state.totalCount--;
35         state.totalPrice -= findItem.price;
36         findItem.count--;
37       }
38     },
39     removeItem(state, action) {
40       state.items = state.items.filter((obj) => obj.id !== action.payload.id);
41
42       if (!state.items.length) {
43         state.totalPrice = 0;
44         state.totalCount = 0;
45       }
46     },
47     clearItems(state) {
48       state.items = [];
49       state.totalCount = 0;
50       state.totalPrice = 0;
51     },
52   },
53 });
54
55 export const { addItem, removeItem, clearItems, minusItem } = cartSlice.actions;
56
57 export default cartSlice.reducer;
58
```

FilterSlice.js

```
1  import { createSlice } from "@reduxjs/toolkit";
2
3  const initialState = {
4    categoryId: 0,
5    currentPage: 1,
6    sort: {
7      name: "популярності",
8      sortName: "rating",
9    },
10 };
11
12 const filterSlice = createSlice({
13   name: "filters",
14   initialState,
15   reducers: {
16     setCategoryId(state, action) {
17       state.categoryId = action.payload;
18     },
19     setSort(state, action) {
20       state.sort = action.payload;
21     },
22     setCurrentPage(state, action) {
23       state.currentPage = action.payload;
24     },
25     setFilters(state, action) {
26       state.sort = action.payload.sort;
27       state.currentPage = Number(action.payload.currentPage);
28       state.categoryId = Number(action.payload.categoryId);
29     },
30   },
31 });
32
33 export const { setCategoryId, setSort, setCurrentPage, setFilters } =
34   filterSlice.actions;
35
36 export default filterSlice.reducer;
37
```

```

1  import { useState } from 'react';
2  import { useDispatch, useSelector } from 'react-redux';
3  import { addItem } from '../../redux/slices/cartSlice';
4  import { Skeleton } from 'antd';
5
6  const typeNames = ['Глянцевый', 'Матовый'];
7  const posterSizes = ['30x40', '60x80', '90x120'];
8
9  function PosterBlock({ _id, title, price, sizes, image, types }) {
10   const dispatch = useDispatch();
11   const cartItem = useSelector((state) =>
12     state.cart.items.find((obj) => obj.id === _id)
13   );
14   const [activeCategoryIndex, setActiveCategoryIndex] = useState(0);
15   const [activeDoughTypeIndex, setActiveDoughTypeIndex] = useState(0);
16
17   const addedCount = cartItem ? cartItem.count : 0;
18
19   const onClickAdd = () => {
20     const item = {
21       id: _id,
22       title,
23       price,
24       image,
25       size: posterSizes[activeCategoryIndex],
26       type: typeNames[activeDoughTypeIndex],
27     };
28     dispatch(addItem(item));
29   };
30
31   const getClassName = (i) => {
32     if (types.length === 1) {
33       return 'active';
34     }
35
36     return activeDoughTypeIndex === i ? 'active' : '';
37   };
38

```

LoginForm.jsx

```
1  import React from 'react'
2  import { Form, Input, Button, message } from 'antd'
3  import { signIn } from 'next-auth/react';
4  import { useRouter } from 'next/router';
5
6  function LoginForm() {
7    const [form] = Form.useForm();
8    const { push } = useRouter()
9
10   const onSubmit = async (values) => {
11     const res = await signIn('credentials', {
12       username: values.login,
13       password: values.password,
14       redirect: false
15     })
16
17     if (res.ok) push('/admin')
18     message.success('Успішний вхід!')
19   }
20
21   return (
22     <Form
23       form={form}
24       layout="vertical"
25       onFinish={onSubmit}
26       rootClassName='loginForm'
27     >
28       <Form.Item
29         label="Логін"
30         name="login"
31       >
32         <Input size="large" />
33       </Form.Item>
34       <Form.Item
35         label="Пароль"
36         name="password"
37       >
38         <Input.Password size="large" />
39       </Form.Item>
40
41       <Form.Item className='loginButtonItem'>
42         <Button className='button loginButton' htmlType='submit'>
43           Login
44         </Button>
45       </Form.Item>
46     </Form>
47   )
48 }
49
50 export default LoginForm
```

Mongo index.jsx

```

1 import { MongoClient } from 'mongodb'
2
3 const uri = process.env.MONGODB_URI
4
5 let client = new MongoClient(uri)
6 let clientPromise
7
8 if (!process.env.MONGODB_URI) throw new Error('Please add your Mongo URI to .env.local')
9
10 if (process.env.NODE_ENV === 'development') {
11   if (!global._mongoClientPromise) {
12     global._mongoClientPromise = client.connect()
13   }
14   clientPromise = global._mongoClientPromise
15 } else {
16   clientPromise = client.connect()
17 }
18
19 export default clientPromise

```

DbConnect.js

```

1 import mongoose from 'mongoose'
2
3 const MONGODB_URI = process.env.MONGODB_URI
4
5 if (!MONGODB_URI) throw new Error('Please define the MONGODB_URI environment variable inside .env.local')
6
7 let cached = global.mongoose
8
9 if (!cached) {
10   cached = global.mongoose = { conn: null, promise: null }
11 }
12
13 async function dbConnect() {
14   if (cached.conn) {
15     return cached.conn
16   }
17
18   if (!cached.promise) {
19     const opts = {}
20
21     cached.promise = mongoose.connect(MONGODB_URI, opts).then((mongoose) => {
22       return mongoose
23     })
24   }
25   cached.conn = await cached.promise
26   return cached.conn
27 }
28
29 export default dbConnect

```