

**КИЇВСЬКИЙ СТОЛИЧНИЙ УНІВЕРСИТЕТ ІМЕНІ БОРИСА
ГРІНЧЕНКА
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА МАТЕМАТИКИ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК**

«Допущено до захисту»

Завідувач кафедри
доц., к.т.н. Машкіна І.В.

(підпис)

«___»_____2025 р.

**КВАЛІФІКАЦІЙНА РОБОТА
на здобуття освітнього ступеня «бакалавр»
Спеціальність 122 «Комп'ютерні науки»**

Тема:

Платформа для автоматизації процесів діяльності книжкового магазину

Виконала:

студентка групи
ІНБ-2-21-4.0д

Гутира Світлана Юріївна


(підпис)

Науковий керівник:

к.т.н. Іваніченко Є. В.

(підпис)

Київ – 2025

Київський столичний університет імені Бориса Грінченка
Факультет інформаційних технологій та математики
Кафедра комп'ютерних наук

«Затверджую»

Завідувач

кафедри комп'ютерних наук,
кандидат технічних наук, доцент
(науковий ступінь, вчене звання)

Ірина МАШКІНА

(підпис)

« ____ » _____ 2025 р.

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА
«Платформа для автоматизації процесів діяльності книжкового магазину»
Виконавець – студентка групи ІНб-2-21-4.0д,
спеціальності 122 Комп'ютерні науки,
освітньої програми 122.00.01 Інформатика
Гутира Світлана Юріївна

1. Вихідні дані: *Публікації з автоматизації бізнес-процесів, веб-розробки, UX-дизайну, використання технологій Node.js, React та MongoDB у розробці онлайн платформ.*

2. Основні завдання:

- Провести огляд літературних джерел та сучасних публікацій за темою автоматизації бізнес-процесів у сфері торгівлі.
- Обґрунтувати мету, об'єкт і предмет дослідження.
- Розробити завдання, структуру та зміст бакалаврської кваліфікаційної роботи.
- Визначити вимоги до функціоналу платформи для автоматизації книжкового магазину.
- Обрати та обґрунтувати стек технологій для реалізації платформи (Node.js, React, MongoDB).
- Розробити архітектуру клієнт-серверної системи.
- Спроекувати базу даних для зберігання інформації про книги, замовлення та користувачів.
- Реалізувати серверну та клієнтську частини програмного забезпечення з урахуванням авторизації, ролей користувачів (менеджер, клієнт), обробки замовлень та інтеграції з платіжною системою.

- Провести тестування програмного продукту, оцінити ефективність функціонування платформи.
- Оформити пояснювальну записку та підготувати презентаційні матеріали до захисту.
- Підготувати наукову публікацію (тези доповіді) за темою роботи.

3. Пояснювальна записка: *Обсяг – до 60 стор. формату А4 комп'ютерного набору з дотриманням вимог стандарту і методичних рекомендацій кафедри.*

4. Графічні матеріали: *діаграми структури БД, архітектура клієнт-серверної взаємодії, інтерфейси користувача, діаграма випадків використання*

5. Додатки: *не передбачено*

6. Строк подання роботи на кафедру: «02» червня 2025 р.

Науковий керівник:
кандидат технічних наук
Іваніченко Євген Вікторович

Виконавець:
Гутира Світлана Юріївна

Дата:

Дата:

Календарний план

№	Етап виконання роботи	Термін виконання	Примітка
1	Формування теми дипломної роботи бакалавра та її затвердження	07.10.24-14.10.24	виконано
2	Аналіз предметної області, огляд аналогів, формування цілей і задач	25.11.24-08.12.24	виконано
3	Вибір технологічного стеку (Node.js, React, MongoDB), обґрунтування вибору	09.12.24-15.12.24	виконано
4	Проектування структури бази даних та архітектури клієнт-серверної взаємодії	20.01.25-07.02.25	виконано
5	Реалізація серверної частини: API, автентифікація, робота з базою даних	08.02.25-22.02.25	виконано
6	Реалізація клієнтської частини: реєстрація, каталог, корзина, історія замовлень	23.02.25-20.03.25	виконано
7	Інтеграція платіжної системи, налаштування ролей (адмін/користувач)	21.03.25-28.03.25	виконано
8	Тестування функціоналу, виправлення помилок, UX-оптимізація	30.03.25-07.04.25	виконано
9	Підготовка пояснювальної записки, графічних матеріалів, реферату	09.04.25-30.04.25	виконано
10	Подання роботи на перевірку, внесення правок	01.05.25-22.05.25	виконано
11	Захист кваліфікаційної роботи	17.06.2025	виконано

Здобувач *Гутира С.Ю.*

Керівник роботи *Іваніченко Є.В.*

РЕФЕРАТ

Кваліфікаційна робота: 61 с., 24 рис., 22 посилання.

Актуальність: в умовах сучасного розвитку інформаційних технологій автоматизація бізнес-процесів стає необхідною умовою підвищення ефективності підприємств. Книжкові магазини, як і інші сфери роздрібної торгівлі, потребують зручних інструментів для управління асортиментом, замовленнями, клієнтською базою та продажами. Розробка онлайн платформи для автоматизації діяльності книжкового магазину дозволить мінімізувати витрати часу на рутинні операції, підвищити точність ведення обліку та поліпшити взаємодію з клієнтами.

Об'єкт дослідження: автоматизація діяльності книжкового магазину.

Предмет дослідження: методи, засоби та технології створення онлайн платформи для автоматизації бізнес-процесів книжкового магазину.

Мета роботи: розробка функціональної онлайн платформи для автоматизації основних бізнес-процесів книжкового магазину, яка забезпечує зручне управління каталогом товарів, замовленнями, та продажами.

Завдання:

- Аналіз бізнес-процесів книжкового магазину та визначення вимог до автоматизації.
- Розробка архітектури платформи з використанням Node.js та React.
- Реалізація функціоналу управління каталогом товарів, обліком замовлень та клієнтів.
- Створення інтерфейсу для адміністраторів та клієнтів.
- Забезпечення інтеграції з платіжними системами.
- Тестування та оптимізація роботи платформи.
- Оцінка ефективності впровадженої системи.

Основні результати: у ході роботи було розроблено онлайн платформу, яка забезпечує автоматизацію управління асортиментом книжкового магазину, включаючи додавання, редагування та видалення товарів. Система дозволяє вести облік замовлень, контролювати їх статус та переглядати історію виконання. Крім цього, платформа підтримує організацію обліку клієнтської бази та фіксацію історії покупок. Для зручності користувачів було інтегровано платіжні системи, що забезпечують комфортну оплату онлайн. Завдяки сучасному дизайну платформа є зручною та інтуїтивно зрозумілою як для адміністраторської сторони, так і для клієнтської.

Практичне значення дослідження: полягатиме у впровадженні розробленої онлайн платформи в діяльність книжкових магазинів, що дозволить підвищити продуктивність праці, зменшити помилки в обліку та поліпшити сервіс для клієнтів.

Ключові слова: автоматизація, книжковий магазин, онлайн платформа, Node.js, React, MongoDB, облік замовлень, клієнтська база, каталог товарів, інтеграція.

ЗМІСТ

ВСТУП	5
РОЗДІЛ I ПОСТАНОВКА ЗАДАЧІ	7
1.1 Мета та завдання розробки платформи	7
1.2 Опис функціональних вимог для системи	8
1.3 Визначення типів користувачів (менеджер та клієнт)	9
1.4 Актуальність і значення автоматизації процесів для книжкового магазину	11
РОЗДІЛ II АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ	14
2.1 Огляд ринку книжкових магазинів і необхідність автоматизації	14
2.2 Аналіз існуючих рішень для автоматизації процесів у магазинах	15
2.3 Вибір технологій та інструментів для розробки	18
РОЗДІЛ III МОДЕЛЮВАННЯ ТА ПРОЕКТУВАННЯ	20
3.1 Моделювання основних процесів системи (управління каталогом, покупка, кошик, історія замовлень)	20
3.2 Проектування бази даних (структура даних для книг, замовлень, користувачів)	22
3.3 Проектування архітектури системи (клієнт-серверна модель)	24
РОЗДІЛ IV ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ	27
4.1 Реалізація серверної частини (функції для роботи з даними користувачів, книг та замовлень)	27
4.2 Реалізація клієнтської частини для клієнтів (перегляд каталогу, кошик, історія покупок)	41
4.3 Реалізація клієнтської частини для менеджера (редагування каталогу).	54
ВИСНОВКИ	59
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	62

ВСТУП

У сучасному світі електронна комерція стала одним із основних каналів реалізації товарів і послуг. Зокрема, книжковий ринок, що історично характеризується значними сезонними коливаннями попиту, також активно адаптується до цифрових технологій. З розвитком інтернет-платформ, підприємства, які займаються продажем книг, мають можливість оптимізувати свої бізнес-процеси, забезпечити зручність для покупців та ефективно управляти асортиментом товарів, замовленнями, платежами та доставкою. Запровадження новітніх технологій автоматизації в книжкових магазинах сприяє суттєвому зменшенню витрат, покращенню якості обслуговування клієнтів та збільшенню обсягів продажу.

Метою цієї роботи є розробка платформи для автоматизації діяльності книжкового магазину, яка дозволить забезпечити всі основні етапи продажу книг – від перегляду товарів і оформлення замовлення до обробки платежів і доставки. Платформа буде побудована з використанням популярних веб-технологій Node.js та React, що забезпечить високу продуктивність, гнучкість і зручність як для кінцевих користувачів, так і для адміністраторів магазину. Основними функціями платформи будуть каталог книг, можливість перегляду детальної інформації про товари, кошик для зберігання вибраних товарів, оформлення та оплата замовлення, а також управління контентом для сторони адміністраторів.

У сучасних умовах ринку, де покупці дедалі більше акцентують увагу на швидкості, зручності та безпеці процесу придбання, необхідно розробити систему, яка ефективно оброблятиме великі обсяги даних, надаючи користувачам безпечний доступ до особистої інформації та зручний і водночас простий процес покупки. Впровадження онлайн платформи для книжкового магазину є необхідним кроком для досягнення конкурентоспроможності та підвищення задоволеності клієнтів.

Вибір Node.js та React як технологічних стеків для розробки обумовлений їхньою здатністю забезпечити високу продуктивність при обробці великої кількості запитів і ефективно працювати в умовах високої навантаженості. Node.js дозволяє створювати серверну частину з високою продуктивністю, а React – надає динамічний та зручний інтерфейс для користувачів.

У результаті реалізації проекту буде створена інтерактивна онлайн платформа, яка не тільки автоматизує роботу книжкового магазину, але й покращить досвід покупців, підвищуючи ефективність і зручність покупок у цифровому середовищі.

РОЗДІЛ I

ПОСТАНОВКА ЗАДАЧІ

1.1 Мета та завдання розробки платформи

Метою розробки онлайн платформи є створення сучасного програмного рішення, яке дозволить автоматизувати ключові бізнес-процеси книжкового магазину. Система повинна стати ефективним інструментом для управління асортиментом товарів, обробки замовлень, ведення обліку клієнтської бази та організації продажів. Електронна комерція стрімко розвивається, і в таких умовах платформа відіграє ключову роль: вона не просто впорядковує роботу всередині компанії, а й робить спілкування з клієнтами зручнішим і ефективнішим. У результаті це підвищує рівень сервісу та допомагає бізнесу залишатися конкурентоспроможним.

Розробка платформи має вирішити проблему ручного обліку, який часто супроводжується помилками, великими затратами часу та ресурсів. Автоматизація управління асортиментом дозволить адміністраторам магазину легко додавати, редагувати та видаляти товари, підтримуючи актуальність інформації про наявність книг, їх ціну, категорії та інші параметри, що в результаті сприятиме точному управлінню запасами та мінімізації ризиків дефіциту або надлишку товарів.

Окремим напрямом є обробка замовлень, що включає відстеження їхнього статусу, організацію доставки та формування звітності. Інтеграція цього процесу в єдину платформу допоможе значно прискорити виконання замовлень і підвищити їхню точність. У клієнтів з'явиться можливість швидко оформлювати покупки, відстежувати свої замовлення та оплачувати їх за допомогою інтегрованих платіжних систем, що зробить процес зручним і прозорим.

Ще однією важливою частиною платформи є управління замовленнями. Впровадження модуля, який дозволяє відслідковувати замовлення та їхній

статус, дасть змогу магазину забезпечити ефективний контроль над процесом обробки замовлень і забезпечити їх своєчасне виконання, що в свою чергу сприятиме покращенню організації роботи магазину та підвищенню рівня задоволеності клієнтів.

Завдяки сучасному інтерфейсу платформа стане зручною як для клієнтів, так і для адміністративного персоналу. Інтуїтивно зрозумілий дизайн допоможе швидко освоїти систему навіть користувачам без технічних навичок. Загалом, реалізація цього проекту має на меті створити ефективний інструмент для автоматизації бізнесу, який сприятиме підвищенню продуктивності магазину, зниженню операційних витрат та збільшенню доходів, що є важливим для молодих книжкових бізнесів особливо у воєнний час.

1.2 Опис функціональних вимог для системи

Онлайн платформа для книжкового магазину повинна надавати низку функціональних можливостей для користувачів та адміністраторів, аби ефективно управляти всіма аспектами продажу книг. Для користувачів, перш за все, важливою функцією є каталог книг, який має бути зручним і інтуїтивно зрозумілим з правильно вибудованою юзер логікою. Користувачі повинні мати можливість переглядати широкий асортимент книг, застосовуючи пошук по назві книги. Каталог має бути динамічним і дозволяти сортувати книги за ціною чи назвою, щоб забезпечити персоналізований підхід до покупок.

Окрім цього, система повинна дозволяти користувачам переглядати конкретну книгу з детальним описом. Також повинна бути можливість додавання конкретної книги до кошика. Інтерфейс повинен бути спрощеним, щоб користувачі могли без проблем додавати книги до кошика, редагувати кількість товарів або видаляти їх. Кошик має надавати повну інформацію про вибрані товари, включаючи кількість, ціну та загальну суму замовлення.

Оформлення та оплата замовлення – це один із найвідповідальніших етапів у всьому процесі покупки, і саме тут платформа не повинна втрачати

користувачів. Після того як клієнт додав усі потрібні товари в кошик, йому має бути просто й зрозуміло завершити замовлення. Інтерфейс має бути інтуїтивним, без зайвих кроків чи затримок. Дуже важливо також надати максимально широкий вибір способів оплати – від міжнародних банківських карток і європейських онлайн-банків до популярних електронних платіжних систем, щоб уникнути ситуацій, коли потенційний покупець залишає замовлення саме через складність або незручність цього фінального кроку.

Для адміністраторів платформи система повинна включати можливість створення та оновлення книг у каталозі. Адміністратори мають доступ до інтерфейсу, що дозволяє їм додавати нові книги, редагувати їхні дані (ціна, опис будь-якого розміру, зображення), а також видаляти застарілі або недоступні книги з каталогу, щоб у користувачів була актуальна і точна інформація. Адміністратори також повинні мати доступ до всіх замовлень, що були здійснені користувачами. Вони можуть переглядати деталі замовлення, змінювати його статус, наприклад, позначати його як виконане або скасоване або в процесі.

Ще один важливий елемент зручного сервісу – можливість швидкої реєстрації та входу користувачів. Користувачам має бути просто створити особистий акаунт, не витрачаючи зайвого часу: достатньо лише електронної пошти та пароля, щоб зберігати історію своїх покупок. Усе – без складних форм чи підтверджень на пів сторінки. Для безпеки користувачів система повинна надавати можливість скидання пароля у разі його забуття та захищати дані шифруванням.

Загалом, функціональні вимоги платформи спрямовані на забезпечення зручності як для кінцевих користувачів, так і для адміністраторів, ставлячи на меті ефективно управляти асортиментом, замовленнями, користувачами та фінансовими операціями на платформі.

1.3 Визначення типів користувачів (менеджер та клієнт)

У межах розробки онлайн платформи для книжкового магазину існують два основних типи користувачів – менеджер самого магазину (адміністратор) та клієнт. Кожен з цих користувачів має специфічні функції та обов’язки, що зумовлюють різний доступ до даних та можливостей системи. Для забезпечення зручності та ефективності роботи платформи, важливо чітко розмежувати права кожного з типів користувачів і забезпечити належну безпеку інформації, яку кожен з них обробляє та створити для них різні інтерфейси.

Клієнт – це основний тип користувача платформи, який використовує систему для перегляду асортименту книг, оформлення замовлень та здійснення покупок. Для клієнта доступна вільна навігація по сайту, перегляд каталогу книг, детальну інформацію про кожну книгу, а також можливість додавати книги в кошик та оформляти/оплачувати замовлення. Клієнт може створювати особистий обліковий запис для збереження історії покупок, зручного доступу до своїх замовлень, оновлення контактної інформації та змінення налаштувань профілю. Після завершення покупки, клієнт має можливість здійснити оплату через доступні платіжні системи, а також перевіряти статус виконання замовлення. Оскільки клієнт взаємодіє з платформою для особистих потреб, його права обмежуються лише доступом до даних, які безпосередньо стосуються його користувацького досвіду, таких як каталог товарів, інформація про замовлення і профіль.

Менеджер (адмін.) – це тип користувача, що має доступ до адміністративної частини платформи та відповідає за управління внутрішніми процесами магазину. Менеджер має розширені права, які дозволяють йому створювати та редагувати контент на платформі, управляти каталогом книг, додавати нові товари, змінювати інформацію про книги, такі як опис, ціна, наявність та зображення. Також він має доступ до замовлень клієнтів, щоб переглядати деталі кожного замовлення, змінювати його статус (наприклад, позначати як “в обробці”, “відправлено” або “доставлено”) та відповідати на

запити користувачів. Менеджер також має можливість переглядати статистику продажів, аналізувати ефективність різних категорій товарів та здійснювати інші адміністративні функції для забезпечення безперебійної роботи магазину.

Важливим аспектом є роль безпеки доступу: клієнт має доступ лише до інформації, що стосується його персонального профілю, замовлень і платіжних даних, тоді як менеджер може працювати з усіма аспектами бізнесу, включаючи управління товарами, обробку всіх замовлень і доступ до загальної статистики продажів. В додачу для забезпечення конфіденційності інформації клієнтів і захисту від несанкціонованого доступу, облікові записи клієнтів та менеджерів повинні бути захищені паролями, а доступ до персональної інформації має бути обмежений відповідними правами.

Щоб усе працювало чітко та без зайвих ризиків, система має "знати", хто саме перед нею – адміністратор чи звичайний користувач. Грамотне розмежування прав доступу дозволяє не лише спростити роботу кожному, а й захистити важливі дані від випадкового або небажаного втручання. Кожен бачить і керує лише тим, що йому потрібно – без плутанини, але з повним контролем. Це як охайно впорядкована шафа: нічого зайвого, і все на своєму місці.

1.4 Актуальність і значення автоматизації процесів для книжкового магазину

За сучасного розвитку технологій та електронної комерції автоматизація стає важливою частиною оптимізації будь-якого бізнесу, навіть книжкового магазину. Це створить багато компетентного управління та операцій у магазині; отже, краще обслуговування клієнтів у світлі постійно зростаючої конкуренції та вимог до ефективності операцій. У сфері книг це особливо важливо: покупці очікують не просто знайти потрібну назву, а отримати її без зайвого очікування. Автоматизована система дозволяє легко обробляти замовлення, підтримувати

актуальний каталог і залишатися на плаву навіть в умовах жорсткої конкуренції та високих очікувань клієнтів.

Одним з основних аспектів автоматизації є оптимізація управління сортуванням. Як для онлайн, так і для офлайн-книжкових магазинів важлива точна інформація про стан запасів; також важливо оновлювати його оперативно, коли надходять нові товари або змінюються ціни. Перегляд класифікацій вручну та внесення змін до каталогу займають багато часу та можуть призвести до помилок. В результаті автоматизації попередні етапи замінюються системою, яка автоматично оновлює інформацію про наявність книг, ціну та інші параметри продукту, що знижує пов'язані з цим ризики та робить процес більш зручним для адміністратора.

Обробка замовлень є ще одним процесом, який необхідно автоматизувати. У звичайному книжковому магазині цей процес передбачає ручне введення даних, перевірку запасів і обробку кожного замовлення. При автоматизації покупці самі розміщують замовлення через онлайн-платформу, система автоматично перевіряє наявність книжок на складі, товар додається в кошик і здійснюється оплата. Після цього адміністратор отримує повідомлення про нове замовлення, яке, безсумнівно, автоматично реєструється в системі, і це справді значно прискорює обробку замовлення та зменшує ризик помилок при введенні деталей.

Впровадження платіжного модуля в структуру книжкового інтернет-магазину є ще одним вагомим кроком щодо автоматизації, що дасть змогу значно суттєво прискорити процес покупки. В результаті забезпечення такої можливості, покупці зможуть здійснювати покупки з будь-якої країни на світі, до того ж територія збуту зросте. Платіжна система може автоматично сервісувати транзакції задля зручності покупців та адміністраторів. Дану платіжну платформу використовуючи для безготівкових розрахунків, дає можливість витрачати менше часу на обробку платіжних документів, знижує ймовірність помилок, і навіть підвищує безпеку ваших платіжних обіходів.

РОЗДІЛ II

АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

2.1 Огляд ринку книжкових магазинів і необхідність автоматизації

Сучасний книжковий ринок активно змінюється під впливом технологічних інновацій та розвитку інтернет-комерції. Онлайн-магазини, як Amazon і локальні платформи, створюють сильну конкуренцію для традиційних книжкових магазинів. Це змушує останні адаптуватися до нових вимог споживачів і технологічних тенденцій. Онлайн-платформи пропонують зручність покупки, швидке оформлення замовлень і різноманітні способи оплати, що стає важливим для збереження конкурентоспроможності.

Одне з найбільших випробувань для книжкових магазинів – це тримати асортимент під контролем і встигати швидко обробляти замовлення. Коли все ведеться вручну, легко щось загубити, переплутати або просто не встигнути – і замість задоволеного читача маємо розчарованого клієнта. Автоматизація змінює гру: дані про наявність книжок оновлюються миттєво, замовлення не зависають у повітрі, а персонал нарешті може займатися справді важливим – допомагати людям знаходити свої наступні улюблені історії.

Автоматизація також полегшує інтеграцію з платіжними системами, що забезпечує зручність для клієнтів при покупках, дозволяючи здійснювати оплату онлайн. Крім того, завдяки автоматизованим системам можна збирати та аналізувати дані про покупців, що дозволяє створювати персоналізовані рекомендації і пропонувати індивідуальні акції або знижки. Це сприяє підвищенню лояльності клієнтів і збільшенню обсягу продажів.

Швидке оформлення й надійність – ключові складові позитивного враження від покупки, і саме автоматизація дозволяє книжковим магазинам відповідати цим очікуванням. Система бере на себе рутину: обробка замовлень відбувається без затримок, всі дані в безпеці, а клієнт отримує саме те, що

замовив, і вчасно. У результаті покупці задоволені, а магазин – конкурентоспроможний.

2.2 Аналіз існуючих рішень для автоматизації процесів у магазинах

Для автоматизації процесів у книжкових магазинах існують різні онлайн платформи, які допомагають з управлінням асортиментом, продажами, обробкою замовлень і взаємодією з клієнтами. Три популярні онлайн платформи для продажу книг – Bookshop.org, Lulu і Book Bolt мають різні можливості та підходи до автоматизації. Розглянемо їхні порівняльні характеристики.

Першою розглянемо Bookshop.org (рис. 1.1).

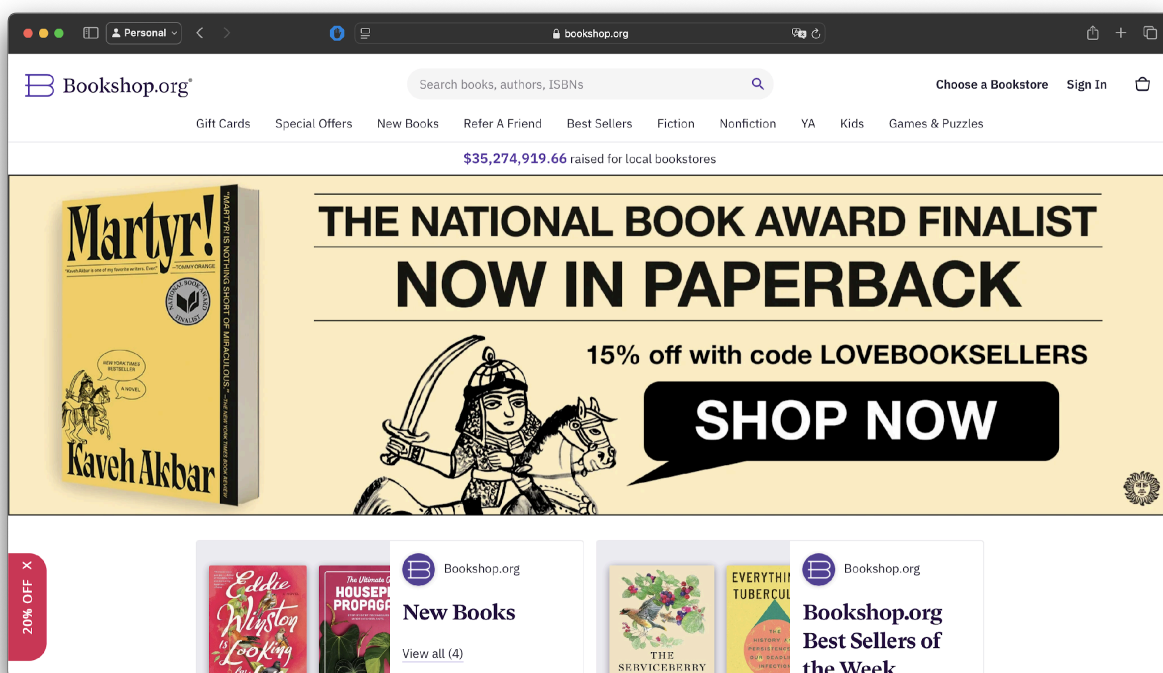


Рисунок 1.1. Головна сторінка Bookshop.org

Переваги:

- Bookshop.org дає можливість локальним книгарням мати онлайн-продажі без необхідності створювати власний інтернет-магазин. Це ідеально підходить для підтримки незалежних маленьких книжкових магазинів.
- Допомогає розвивати локальні ринки і надає можливість для авторів та видавців, дозволяючи авторам продавати свої книги, зберігаючи частину доходу.
- Інтуїтивно зрозумілий інтерфейс, який робить процес купівлі та продажу зручним для користувачів і продавців.

Недоліки:

- Платформа не дозволяє вносити багато змін до дизайну та функціональності інтернет-магазину.
- Платформа більше орієнтована на невеликі книжкові магазини або окремих авторів, і має недостатню масштабованість для великих мереж.

Далі поговоримо про Lulu.com (рис. 1.2).

Переваги:

- Lulu дає змогу авторам та видавцям публікувати книги без посередників і продавати їх через власний інтернет-магазин, а також інші канали, як Amazon, Barnes & Noble.
- Платформа підтримує різні формати книг (паперові видання, електронні книги та аудіокниги).
- Надає інструменти для управління замовленнями, обробки платежів та моніторингу продажів, що полегшує роботу авторів і видавців.

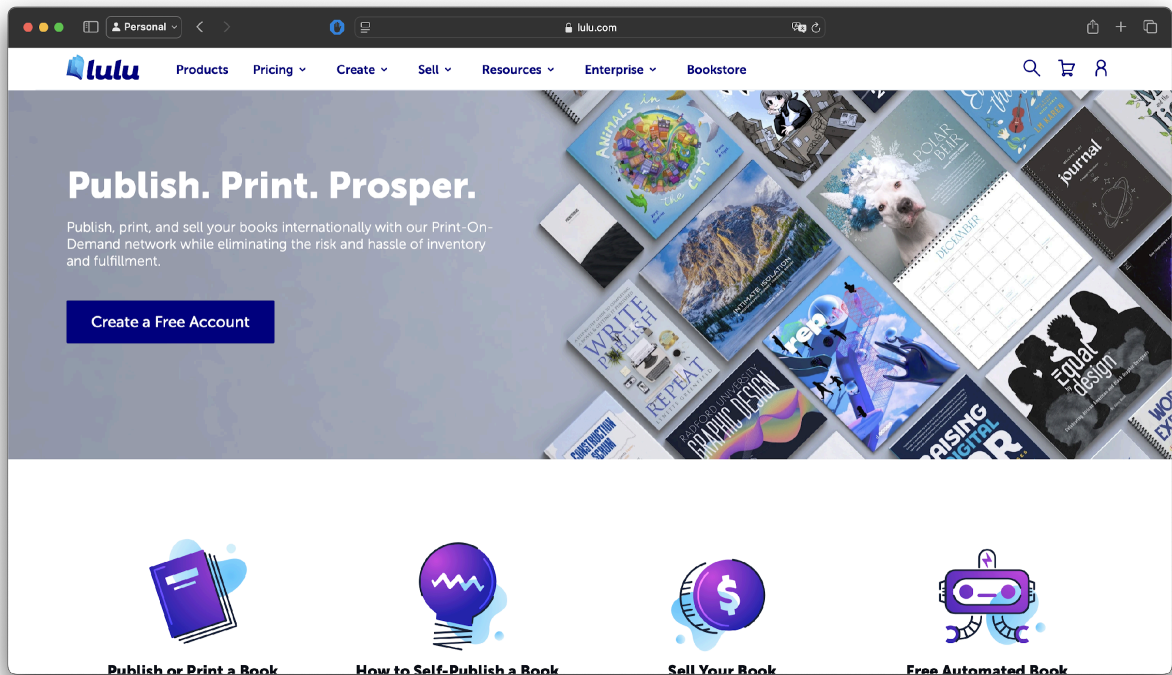


Рисунок 1.2. Lulu.com

Недоліки:

- Враховуючи популярність платформи, новим авторам може бути важко виділятися серед численних публікацій.
- Деякі послуги на платформі, як друк на вимогу або додаткові функції, можуть бути досить дорогими для авторів.

І останній на черзі Book Bolt (рис. 1.3).

Переваги:

- Пропонує все необхідне для створення low-content книг — від генератора обкладинок і шаблонів інтер'єру до інструментів дослідження ключових слів та аналізу конкурентів.
- Платформа має зручний інтерфейс
- Вбудовані функції для аналізу ключових слів і категорій допомагають оптимізувати книги для Amazon KDP для кращої видимості та продажам.

- Надає шаблони для створення різноманітних low-content книг (зошити, планери, журнали)

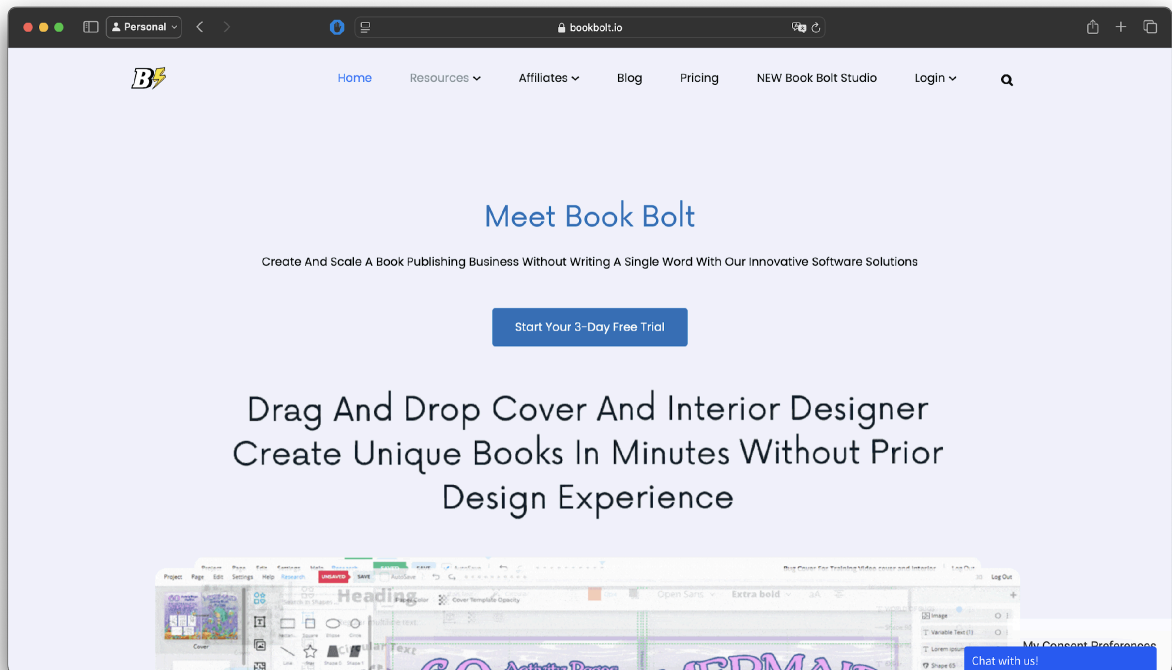


Рисунок 1.3. Інтерфейс bookbolt.io

Недоліки:

- Book Bolt більше орієнтований на створення простих low-content книг і може не відповідати потребам авторів, які працюють над складнішими проектами або потребують більш гнучких інструментів.
- Платформа працює за моделлю підписки і за відгуками користувачів – достатньо дорого коштує в порівнянні з конкурентами.

2.3 Вибір технологій та інструментів для розробки

Для створення онлайн-платформи, яка автоматизує роботу книжкового магазину, були обрані сучасні технології, здатні забезпечити стабільну роботу системи за високого навантаження, її гнучкість та подальше масштабування.

Node.js був обраний для серверної частини платформи завдяки своїй асинхронній обробці запитів, що дозволяє забезпечити швидку реакцію серверу при великому навантаженні, а також можливості роботи з JavaScript на сервері, що зручно для розробників, знайомих із цією мовою. Node.js також надає доступ до великого набору готових бібліотек і модулів, що значно спрощує реалізацію складних функцій і пришвидшує розробку загалом.

Для фронтенду використовується React, оскільки він дозволяє створювати швидкі та інтерактивні інтерфейси користувача завдяки використанню компонентної архітектури та можливості ефективно оновлювати тільки ті частини сторінки, які змінились, без перезавантаження всього додатку. React також має великий екосистему інструментів, таких як React Router для навігації і Redux для керування станом.

У моїй професійній діяльності я вже мала досвід роботи з цим технологічним стеком (Node.js+React) у реальних умовах високого навантаження. Один із проєктів, над яким ми з командою працювали, обслуговував понад 75 тисяч активних користувачів на місяць та витримував тисячі одночасних запитів без збоїв завдяки добре налаштованій архітектурі та оптимізації роботи з даними.

В якості бази даних я обрала MongoDB, оскільки вона є документоорієнтованою NoSQL базою даних і надає гнучкість у зберіганні даних, особливо коли структура даних може змінюватися або є складними. MongoDB добре підходить для роботи з великими обсягами даних і дозволяє легко масштабувати додаток.

Ці технології забезпечують надійну та швидку платформу, яка може ефективно справлятися з автоматизацією бізнес-процесів книжкового магазину.

РОЗДІЛ III

МОДЕЛЮВАННЯ ТА ПРОЕКТУВАННЯ

3.1 Моделювання основних процесів системи (управління каталогом, покупка, кошик, історія замовлень)

У рамках онлайн платформи для книжкового магазину важливими етапами є управління каталогом, покупка книг, взаємодія з кошиком, перегляд історії замовлень, а також можливість адміністрування книг.

Перший важливий процес – управління каталогом. Користувачі повинні мати змогу переглядати всі доступні книги, які представлені на платформі. Кожна книга містить основну інформацію, таку як назва, опис, ціна, зображення обкладинки та інші дані, що допомагають користувачу зробити вибір. Адміністратор має можливість додавати нові книги, редагувати їхні дані або видаляти, якщо книга більше не продається.

Другим важливим процесом є покупка та взаємодія з кошиком. Користувачі можуть додавати книги до кошика, переглядати його зміст і редагувати кількість товарів перед оформленням покупки. Кошик є тимчасовим сховищем для книг, які користувач планує придбати. Коли покупець готовий завершити покупку, він може перейти до оформлення замовлення.

Історія замовлень дозволяє користувачам переглядати всі здійснені покупки в минулому. Це зручно для повторних покупок або для перегляду інформації про раніше придбані книги. Користувач може побачити, які книги він придбав, коли це було зроблено та у якій кількості.

Адміністратор також має можливість створювати нові книги, редагувати їхні дані або видаляти вже застарілі записи. Це забезпечує актуальність та точність інформації в каталозі.

Нарешті, реєстрація та вхід користувача є важливою частиною системи. Користувач повинен мати можливість створювати обліковий запис для

збереження історії покупок, управління кошиком і замовленнями. Система також повинна підтримувати функції входу для зареєстрованих користувачів, що забезпечує доступ до персоналізованого досвіду на платформі.

Можемо представити дані процеси у вигляді діаграми випадків використання (рис. 2.1).

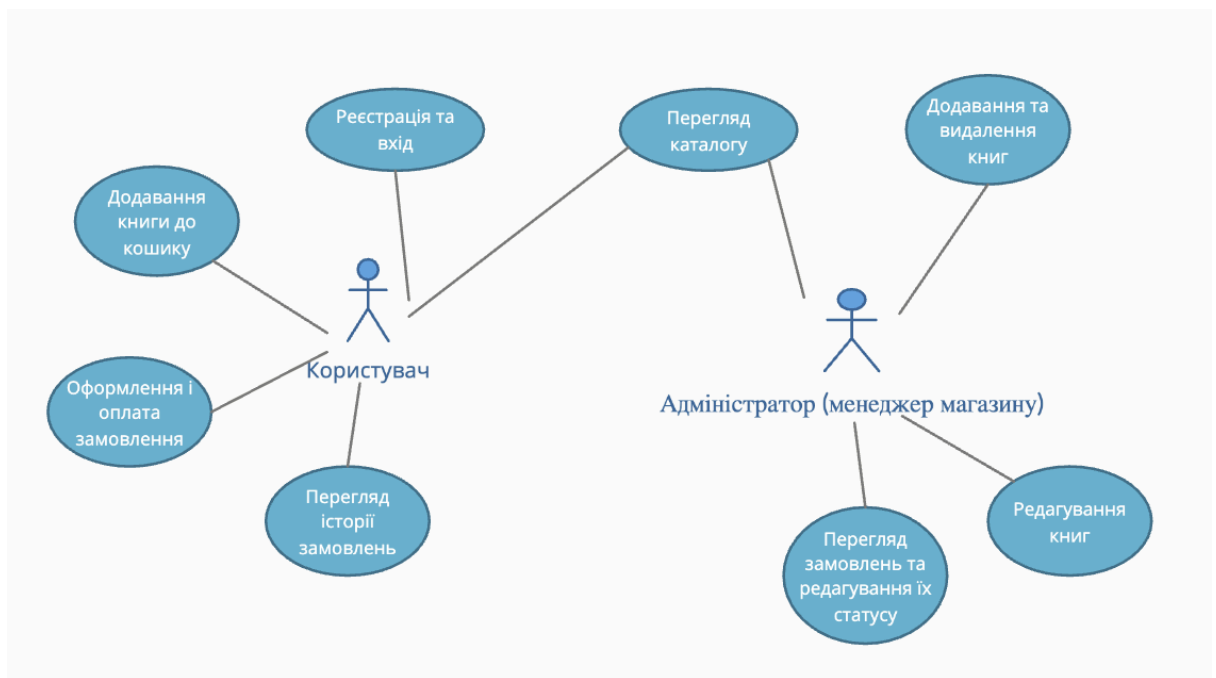


Рисунок 2.1. Діаграма випадків використання

3.2 Проектування бази даних (структура даних для книг, замовлень, користувачів)

Проектування бази даних для онлайн платформи книжкового магазину є важливим етапом у створенні системи, оскільки вона визначає структуру збереження інформації про книги, користувачів та замовлення, і якщо ця структура порушена – платформа не буде працювати коректно. Для забезпечення ефективного зберігання та обробки даних було вибрано використання MongoDB як бази даних, оскільки вона добре підходить для роботи з неструктурованими або частково структурованими даними. Структура

даних для книг, користувачів і замовлень представлена за допомогою моделей в MongoDB, описаних через Mongoose.

Модель бази даних наведена на рисунку 2.2:

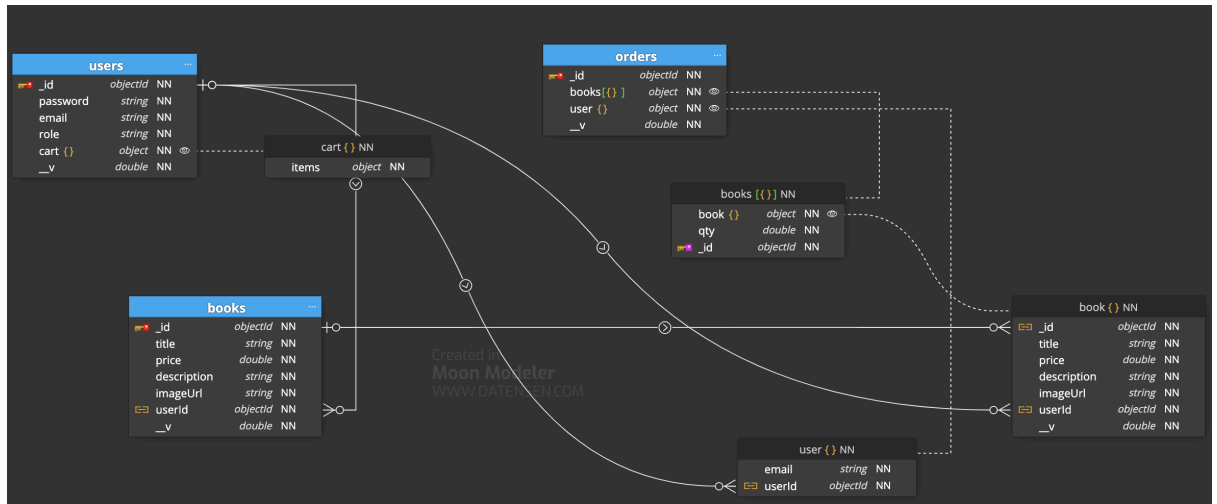


Рисунок 2.2. Модель бази даних

Структура даних для книг:

- title – назва книги (тип: String, обов'язкове поле).
- price – ціна книги (тип: Number, обов'язкове поле).
- description – опис книги (тип: String, обов'язкове поле).
- imageUrl – посилання на зображення обкладинки книги (тип: String, обов'язкове поле).
- userId – ідентифікатор користувача, який додав книгу (тип: ObjectId, посилання на модель 'User', обов'язкове поле).

Структура даних для користувачів:

- password – пароль користувача (тип: String, обов'язкове поле).
- resetToken – токен для скидання пароля (тип: String).
- resetTokenExpDate — дата закінчення терміну дії токена для скидання пароля (тип: Date).
- email – електронна пошта користувача (тип: String, обов'язкове поле).

- `role` – роль користувача (тип: `String`, може бути “адміністратор” або “клієнт”).
- `cart` – кошик користувача, який містить список товарів (книг) з кількістю. Це поле є об’єктом, що містить масив об’єктів з ідентифікатором книги та її кількістю.

Структура даних для замовлень:

- `books` – масив об’єктів, що містять книгу та кількість замовленої книги.
- `user` – інформація про користувача, який зробив замовлення, включаючи його електронну пошту та ідентифікатор в системі.

Взаємозв’язки між моделями:

- Кожен користувач може мати один кошик, який містить список книг із зазначенням кількості.
- Кожне замовлення прив’язано до конкретного користувача, і зберігає інформацію про замовлені книги, їх кількість і загальну суму.
- Книги прив’язані до конкретних користувачів, щоб дозволити відслідковувати, які книги були додані до каталогу, а також редагувати та видаляти книги адміністратором.

3.3 Проектування архітектури системи (клієнт-серверна модель)

Архітектура системи онлайн-платформи для книжкового магазину реалізована за клієнт-серверною моделлю, що дозволяє розподілити функціональність між клієнтською та серверною частинами. Це немов сцена й закулісся в театрі. На "сцені" (клієнтській частині) користувач бачить красивий, зручний інтерфейс, обирає книжки, формує замовлення. А за лаштунками (на сервері) відбувається вся важка робота: обробка запитів, взаємодія з базою даних, перевірка оплати тощо. Цей підхід перевірений роками і дає змогу досягти масштабованості, гнучкості та легкості в обслуговуванні системи.

Основними елементами архітектури є клієнтська частина, серверна частина та база даних.

Клієнтська частина була створена з використанням бібліотеки React для інтерактивного та динамічного інтерфейсу для користувачів. Вона відповідає за відображення даних, отриманих із сервера, і передачу користувацьких запитів назад на сервер.

Основні функції клієнтської частини:

- Відображення каталогу книг, інформації про окрему книгу, кошика та історії замовлень.
- Реалізація форм для реєстрації, входу, оформлення замовлення.
- Інтерактивна взаємодія користувача із системою, включаючи додавання книг до кошика, пошук книг, тощо.
- Використання REST API для отримання та передачі даних на сервер.

Серверна частина створена на Node.js із застосуванням фреймворку Express. Вона виконує обробку запитів від клієнтської частини, керує бізнес-логікою та забезпечує взаємодію з базою даних.

Основні функції серверної частини:

- Управління каталогом книг: додавання, редагування, видалення книг (для адміністраторів).
- Обробка замовлень: збереження деталей замовлень, оновлення статусу.
- Управління користувачами: реєстрація, автентифікація, авторизація.
- Робота з базою даних MongoDB для збереження даних про книги, користувачів і замовлення.
- Захист API за допомогою JWT (JSON Web Token) для безпечного доступу до ресурсів.

База даних MongoDB використовується для зберігання структурованої інформації. Основні колекції:

- Books: зберігає інформацію про книги.
- Users: зберігає дані користувачів, включаючи ролі (клієнт чи адміністратор) і кошик.
- Orders: зберігає інформацію про замовлення, включаючи деталі книг і користувачів.

Взаємодія компонентів:

- Користувач, використовуючи клієнтський інтерфейс, надсилає запити (наприклад, для перегляду каталогу чи оформлення замовлення).
- Сервер отримує запити через REST API, обробляє їх, взаємодіє з базою даних, виконує відповідні операції (збереження замовлення, оновлення даних) і повертає відповідь клієнту.
- Клієнтська частина оновлює інтерфейс на основі отриманих даних.

Особливості архітектури:

- Модульність: кожен елемент системи (клієнт, сервер, база даних) виконує конкретно визначені завдання.
- Масштабованість: можливість збільшення продуктивності сервера або клієнта без значних змін у структурі.
- Безпека: використання авторизації за допомогою токенів для захисту даних користувачів та замовлень.
- Реактивність: клієнтська частина побудована так, щоб забезпечити швидку і зручну взаємодію для користувачів.

Схематичне зображення архітектури можна побачити на рисунку 2.3.



Рисунок 2.3. Архітектура застосунку

РОЗДІЛ IV

ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

4.1 Реалізація серверної частини (функції для роботи з даними користувачів, книг та замовлень)

Почнемо з визначення файлової структури серверної частини. Її можна побачити на рисунку 3.1.

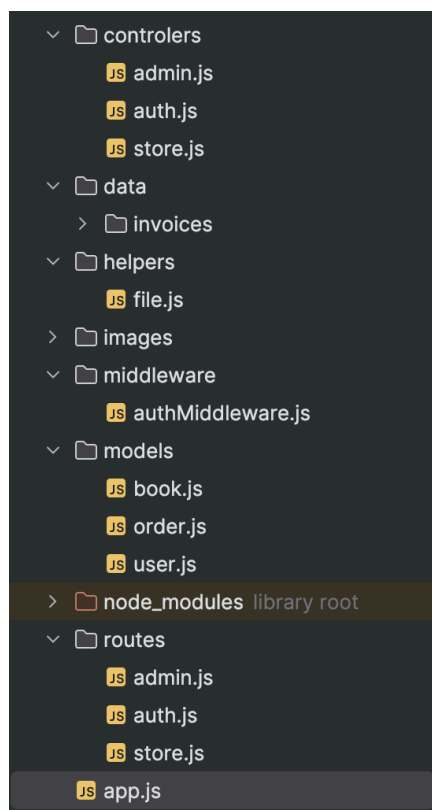


Рисунок 3.1. Файлова структура моєї платформи

Першим кроком у розробці серверної частини є створення головного файлу `app.js`, який відповідає за запуск сервера. У цьому файлі ми ініціалізуємо Express, підключаємо базу даних та налаштовуємо основні маршрути.

Для початку додаємо підключення до бази даних та ініціалізацію Express:

```
const MONGODB_URI =
'mongodb+srv://testuser873479:Daxvu3-ciskiq-buhpon@cluster0.1bkxc.mongodb.net/book
-store?retryWrites=true&w=majority&appName=Cluster0'

const app = express()
```

Додамо імпорти файлів з шляхами з папки routes і модель користувача з папки models, щоб забезпечити зв'язок між серверною логікою і клієнтськими запитами, аби обробляти запити від користувачів, управляти книгами та замовленнями, а також здійснювати авторизацію:

```
const adminData = require('./routes/admin')
const storeRoutes = require('./routes/store')
const authRoutes = require('./routes/auth')
const User = require('./models/user')
```

Перед обробкою всіх запитів потрібно перевіряти, чи передано JWT-токен у заголовок запиту. Це необхідно для автентифікації користувачів та захисту маршрутів, які вимагають авторизації. Якщо токен є, необхідно розшифрувати його та отримати відповідного користувача з бази даних, що дозволить ідентифікувати його рівень доступу та забезпечити персоналізовану взаємодію із системою.

```
app.use(async (req, res, next) => {
  const token = req.header('Authorization');
  if (token) {
    try {
      const decoded = jwt.verify(token.split(' ')[1], 'secret-key');
      req.user = await User.findById(decoded.userId)
      ...
    } catch (err) {
      // token is invalid
    }
  }
  next()
})
```

Після цього можна підключити всі маршрути до нашого застосунку:

```
app.use('/admin', adminData.routes)
app.use(storeRoutes)
app.use(authRoutes)
```

І можемо під'єднатись до бази даних і запустити сервер (таким чином, серверна частина готова до обробки запитів, а система автентифікації забезпечує захист користувацьких даних):

```
const connect = async () => {
  try {
    await mongoose.connect(MONGODB_URI)
    app.listen(3000)
    ...
  }
  connect()
}
```

Далі потрібно додати так звану *middleware* – спеціальну функцію, яка виконується між отриманням запиту і його обробкою. Вона дозволяє перевіряти, чи користувач авторизований, і якщо так, визначати його роль для захисту шляхів якщо користувач не авторизований і якщо авторизований користувач не адміністратор:

```
function verifyToken(req, res, next) {
  const token = req.header('Authorization');
  ...
  try {
    jwt.verify(token.split(' ')[1], 'secret-key');
    next();
  }
  ...
}

function verifyTokenAdmin(req, res, next) {
  const token = req.header('Authorization');
  if (!token || req.user.role === 'user') return res.status(401).json({error: 'Access denied'});
  try {
    jwt.verify(token.split(' ')[1], 'secret-key');
    next();
  }
  ...
}
```

Тепер додамо виклики методів контролерів до відповідних маршрутів у роутері. Перед їх додаванням потрібно забезпечити валідацію введених даних та захист маршрутів там, де це необхідно. Спочатку налаштуємо маршрути для входу та реєстрації користувачів:

```
router.post('/login', [...], authController.postLogin)
router.post('/signup', [...], authController.postSignup)
router.post('/reset', authController.postReset)
router.post('/new-password', [...], authController.postNewPassword)
router.get('/user', authController.getUser)
```

Далі додамо шляхи для адміністратора:

```

router.get('/books', verifyTokenAdmin, adminController.getBooks)
router.post('/books', [
  verifyTokenAdmin,
  ...
], adminController.postAddBook)
router.put('/books/:bookId', [
  verifyTokenAdmin,
  ...
], adminController.postEditBook)
router.delete('/books/:bookId', verifyTokenAdmin, adminController.deleteBook)

```

І останніми додамо шляхи для взаємодії звичайного користувача з магазином:

```

router.get('/books', storeController.getBooks)
router.get('/books/:bookId', storeController.getBook)
router.get('/cart', verifyToken, storeController.getCart)
router.post('/cart/:bookId', verifyToken, storeController.postCart)
router.delete('/cart/:bookId', verifyToken, storeController.deleteCartBook)
router.get('/checkout', verifyToken, storeController.getCheckout)
router.get('/checkout/success', storeController.getCheckoutSuccess)
router.get('/orders', verifyToken, storeController.getOrders)
router.get('/orders/:orderId', verifyToken, storeController.getInvoice)

```

Тепер можемо перейти до створення логіки контролерів для взаємодії з базою даних (для того щоб обробити запити від користувачів, виконати необхідні операції з даними та повернути результати). Першим кроком буде реалізація авторизації, яка дозволяє користувачам створювати облікові записи, входити в систему та отримувати доступ до персоналізованих функцій магазину.

Метод для реєстрації нового акаунту:

```
const postSignup = async (req, res, next) => {
  const { email, password } = req.body
  const errors = validationResult(req)
  if (!errors.isEmpty()) {
    return res.status(422).json({ error: errors.array()[0].msg });
  }
  const hashedPassword = await bcrypt.hash(password, 12)
  const user = new User({ email: email, password: hashedPassword, role: 'user',
  cart: { items: [] } })
  await user.save()
  res.status(200).json({ message: 'User created' });
}
```

Тут ми перевіряємо, чи виникли помилки під час валідації даних. Якщо валідація пройшла успішно, створюємо нового користувача та зберігаємо його пароль у зашифрованому вигляді для безпеки.

Далі реалізуємо метод для входу в акаунт:

```
const postlogin = async (req, res, next) => {
  const { email, password } = req.body
  ...
  const user = await User.findOne({ email: email })
  if (!user) {
    return res.status(401).json({ error: 'User not found' });
  }
  const doMatch = await bcrypt.compare(password, user.password)
  if (!doMatch) {
    return res.status(401).json({ error: 'Passwords do not match' });
  }
  const token = jwt.sign({ userId: user._id }, 'secret-key');
  res.status(200).json({ token });
}
```

Тут ми перевіряємо, чи існує користувач із вказаною електронною поштою. Якщо він знайдений і введений пароль збігається, авторизація проходить успішно, і ми повертаємо користувачеві JWT-токен.

Також потрібно реалізувати функціонал зміни пароля. Перший крок – надсилання електронного листа на адресу, з якою зареєстрований акаунт, та генерація токена для скидання пароля. Для відправлення листа використовуватимемо SendGrid (сервіс для надсилання електронних листів, який забезпечує надійну доставку повідомлень):

```
const postReset = async (req, res, next) => {
  const { email } = req.body
  return crypto.randomBytes(32, async (err, buf) => {
    ...
    const token = buf.toString('hex')
    const user = await User.findOne({ email })
    ...
    user.resetToken = token;
    user.resetTokenExpDate = Date.now() + 3600000
    await user.save()
    res.status(200).json({ message: 'Email was sent' });
    transporter.sendMail({
      to: email,
      from: '****',
      html: `
        <p>You requested a password reset</p>
        <p>Click this <a
href=${req.get('referer')}reset-password/${token}>link</a> to set a new
password.</p>
      `,
      subject: 'Password reset'
    })
  })
}
```


Інший метод відповідає за зміну пароля після переходу за посиланням з електронного листа. Він перевіряє токен, що був згенерований для скидання пароля, і якщо він дійсний, дозволяє користувачеві встановити новий пароль.

```
const postNewPassword = async (req, res, next) => {
  const { password, passwordToken } = req.body
  ...
  const user = await User.findOne({ resetToken: passwordToken, resetTokenExpDate:
  { $gt: Date.now() } })
  ...
  user.password = await bcrypt.hash(password, 12)
  user.resetToken = undefined
  user.resetTokenExpDate = undefined
  await user.save()
  res.status(200).json({ message: 'Password updated' });
}
```

Тут ми перевіряємо, чи існує користувач із вказаним токеном для зміни пароля. Якщо він знайдений, дозволяємо змінити пароль і видаляємо використаний токен, щоб запобігти повторному використанню.

Далі перейдемо до методів для адміністратора. Першим розглянемо метод для додавання книги:

```
const postAddBook = async (req, res, next) => {
  const { title, price, description } = req.body
  const { file } = req
  ...
  const book = new Book({ title, imageUrl: file.path, price, description, userId:
  req.user })
  try {
    await book.save()
    res.status(200).json({ message: 'Book created' });
  } catch (err) {
    res.status(500).json({ error: 'Adding book failed' });
  }
}
```

Метод для редагування книги:

```
const postEditBook = async (req, res, next) => {  
  const { title, price, description } = req.body  
  const { bookId } = req.params  
  const { file } = req  
  ...  
  const book = await Book.findById(bookId)  
  ...  
  book.title = title  
  if (file) {  
    deleteFile(book.imageUrl)  
    book.imageUrl = file.path  
  }  
  book.price = price  
  book.description = description  
  try {  
    await book.save()  
    res.status(200).json({ message: 'Book updated' });  
  } catch (err) {  
    res.status(500).json({ error: 'Updating book failed' });  
  }  
}
```

Тут ми перевіряємо, чи присутній файл в запиті. Якщо файл є, то ми видаляємо старе зображення, що було завантажено раніше, і оновлюємо всі інші поля книги (наприклад, назву, опис, автора тощо). Аби адміністратор міг змінювати інформацію про книгу без зайвих помилок чи втрат.

Метод для видалення книги:

```
const deleteBook = async (req, res, next) => {
  const { bookId } = req.params
  try {
    const book = await Book.findById(bookId)
    deleteFile(book.imageUrl)
    await Book.deleteOne({ _id: bookId, userId: req.user._id })
    res.status(200).json({ message: 'Success!' })
  }
  ...
}
```

Якщо книга з відповідним ідентифікатором знайдена, ми видаляємо зображення книги з папки і саму книгу з бази даних, аби видалення книги з платформи та очищення пов'язаних даних відбувалось коректно.

Реалізовуємо метод для отримання всіх книг створених даним адміністратором для того щоб отримати список усіх книг, які він додав на платформу:

```
const getBooks = async (req, res, next) => {
  const books = await Book.find({userId: req.user._id})
  res.status(200).json({ books })
}
```

Створимо метод для отримання всіх замовлень для адміністратора, щоб мати можливість відстежувати процес продажу та виконання замовлень:

```
const getOrders = async (req, res, next) => {
  const orders = await Order.find()
  res.status(200).json({ orders })
}
```

І останнім додамо метод для оновлення статусу змовлення (наприклад, з 'In progress' на 'Completed'):

```
const postChangeOrderStatus = async (req, res, next) => {
  const { status } = req.body
  const { orderId } = req.params
  const order = await Order.findById(orderId)
  ...
  order.status = status
  try {
    await order.save()
    res.status(200).json({ message: 'Order updated' });
  } catch (err) {
    console.log(err)
    res.status(500).json({ error: 'Updating order failed' });
  }
}
```

Тепер перейдемо до методів для звичайного користувача. Першим розглянемо отримання всіх книг доступних в магазині, щоб переглядати весь асортимент та вибирати потрібні товари:

```
const getBooks = async (req, res, next) => {
  const books = await Book.find()
  res.status(200).json({ books })
}
```

Метод для отримання всієї детальної інформації про конкретну книгу за її ідентифікатором (ID), включаючи назву, автора, опис, ціну та зображення:

```
const getBook = async (req, res, next) => {
  try {
    const book = await Book.findById(req.params.bookId)
    res.status(200).json({ book })
  } catch (err) {
    res.status(401).json({ error: 'Book not found' })
  }
}
```

Метод для отримання корзини з інформацією про товар для надання можливості користувачеві переглянути вміст його корзини:

```
const getCart = async (req, res, next) => {  
  const populatedUser = await req.user.populate('cart.items.bookId')  
  res.status(200).json({ books: populatedUser.cart.items })  
}
```

Метод для додавання товару до корзини:

```
const postCart = async (req, res, next) => {  
  const { bookId } = req.params  
  try {  
    const book = await Book.findById(bookId)  
    await req.user.addToCart(book)  
    res.status(200).json({ message: 'Book added to cart' })  
  } catch (err) {  
    res.status(401).json({ error: 'Error adding book to cart' })  
  }  
}
```

Тут ми викликаємо користувацький метод, який створимо в моделі Book. Він працюватиме таким чином: якщо товар ще не було в корзині, ми додаємо його з кількістю 1, а якщо товар вже є в корзині, просто збільшуємо кількість на 1.

```

userSchema.methods.addToCart = function (book) {
  const cartBookInd = this.cart.items.findIndex(el => el.bookId.toString() ===
book._id.toString())
  let qty = 1
  if (cartBookInd !== -1) {
    this.cart.items[cartBookInd].qty += 1
  } else {
    this.cart.items.push({ bookId: book._id, qty })
  }
  return this.save()
}

```

Метод для видалення товару з корзини користувачем (наприклад якщо користувач вирішив змінити свій вибір або не бажає купувати певну книгу):

```

const deleteCartBook = async (req, res, next) => {
  const { bookId } = req.params

  try {
    await req.user.removeFromCart(bookId)
    res.status(200).json({ message: 'Book removed from cart' })
  } catch (err) {
    res.status(401).json({ error: 'Error removing book from cart' })
  }
}

```

Метод для розрахунку загальної вартості при оформленні замовлення, враховуючи кількість товарів у корзині та їхню ціну:

```

const getCheckout = async (req, res, next) => {
  const populatedUser = await req.user.populate('cart.items.bookId')
  const books = populatedUser.cart.items
  const session = await stripe.checkout.sessions.create({
    payment_method_types: ['card'],
    mode: 'payment',
    line_items: books.map(p => {
      ...
    })
  },
  success_url:
`${req.protocol}://${req.get('host')}/checkout/success?userId=${req.user._id}&referer=${req.get('referer')}`,
  cancel_url: `${req.get('referer')}/cart`,
  ))

  res.status(200).json({ id: session.id })
}

```

Тут ми використовуємо Stripe для проведення оплати в тестовому режимі. Тестовий режим Stripe дозволяє здійснювати транзакції без фактичного переведення грошей, що дає можливість перевірити процес оплати через список фейкових номерів карток на сайті перед запуском у реальному режимі. І якщо оплата пройшла успішно, то будемо перенаправляти на метод для створення замовлення, а якщо оплата не вдалась, то повертаємо на сторінку корзини користувача

Метод для створення замовлення при успішній оплаті (коли оплата пройшла успішно, цей метод створює замовлення для користувача, додаючи всі товари з його корзини до бази даних як частину нового замовлення):

```

const getCheckoutSuccess = async (req, res, next) => {
  const { userId, referer } = req.query
  const user = await User.findById(userId)
  const populatedUser = await user.populate('cart.items.bookId')
  const books = populatedUser.cart.items.map(el => {
    return {
      qty: el.qty, book: { ...el.bookId._doc }
    }
  })
  const order = new Order({
    user: {
      email: user.email,
      userId: user
    },
    books
  })
  try {
    await order.save()
    await user.clearCart()
    res.redirect(`${referer}orders`)
  } catch (err) {
    res.redirect(`${referer}cart`)
  }
}

```

Якщо замовлення було створено успішно, користувач буде перенаправлений на сторінку з його замовленнями, а корзина очищується. У разі невдачі, користувача повертаємо назад на сторінку корзини, де він може виправити помилку та спробувати знову.

Метод отримання замовлень:


```
const getOrders = async (req, res, next) => {
  const orders = await Order.find({ 'user.userId': req.user._id })
  res.status(200).json({ orders })
}
```

Далі я реалізувала метод для отримання виписки по конкретному замовленню в форматі PDF, для створення файлу я використовувала бібліотеку `pdfkit`, яка дозволяє перетворити дані замовлення у зручний для зберігання та перегляду форматі і дозволяє генерувати PDF-документи з HTML-контенту.

```
const getInvoice = async (req, res, next) => {
  const { orderId } = req.params
  const order = await Order.findById(orderId)
  ...
  const invoiceName = `invoice-${orderId}.pdf`
  const invoicePath = path.join('data', 'invoices', invoiceName)
  res.setHeader('Content-Type', 'application/pdf');
  res.setHeader('Content-Disposition', `inline; filename=${invoiceName}`);
  const pdfDoc = new PDFDoc()
  pdfDoc.pipe(fs.createWriteStream(invoicePath))
  pdfDoc.pipe(res)
  pdfDoc.fontSize(26).text('Invoice', {
    underline: true
  })
  pdfDoc.text('-----')
  let totalPrice = 0
  order.books.forEach(el => {
    totalPrice += el.qty * el.book.price
    pdfDoc.fontSize(16).text(`${el.book.title} - ${el.qty} x ${el.book.price}`)
  })
  pdfDoc.fontSize(18).text(`\nTotal price: ${totalPrice}`)
  pdfDoc.end(res)
}
```

У цьому методі ми встановлюємо правильний хедер, щоб сервер повернув не стандартний JSON, а PDF-файл. Після цього створюємо PDF-документ, зберігаємо його в відповідну папку на сервері, а після завершення процесу генерації відправляємо файл користувачеві для завантаження.

4.2 Реалізація клієнтської частини для клієнтів (перегляд каталогу, кошик, історія покупок)

Почнемо з головного файлу для запуску проекту – App.js. У цьому файлі ми використаємо react-router для навігації по сайту аби ефективно керувати маршрутизацією між різними сторінками. Ми також додамо обмеження на доступ до певних маршрутів для зареєстрованих та незареєстрованих користувачів, а також для адміністраторів. Це дозволяє забезпечити правильну рольову авторизацію, де кожен тип користувача має доступ лише до дозволених ресурсів. Додатково, ми виконуємо запит на отримання даних про користувача, якщо в LocalStorage присутній токен щоб зберігати стан автентифікації користувача між сесіями.

Також ми створюємо метод для виходу з акаунту, який видаляє токен з LocalStorage та перенаправляє користувача на головну сторінку аби коректно завершити сесію користувача. Код виглядає наступним чином:

```
function App() {
  const [user, setUser] = useState(null)
  const navigate = useNavigate();
  const fetchData = async () => {
    const user = await fetch(`${API_URL}/user`, { headers: getHeaders() })
    const data = await user.json()
    setUser(data.user);
  }
  useEffect(() => {
    if (localStorage.getItem(LOCALSTORAGE_KEY)) {
      fetchData()
    }
  }, [])
  const logout = useCallback(() => {
    localStorage.removeItem(LOCALSTORAGE_KEY)
    setUser(null)
    navigate('/')
  }, [navigate])
}
```

```
return (
  <>
    <header className="main-header">
      <NavBar user={user} logout={logout}/>
    </>
  </>
)
```

Ця компонента буде повертати header та основний контент сторінки в залежності від поточного маршруту. Вона також реалізує перехід на шлях по замовчуванню:

```

return (
  <>
    <header className="main-header">
      <NavBar user={user} logout={logout}/>
    </header>
    <Routes>
      <Route index element={<Shop isLoggedIn={!!user && user?.role !==
'admin'}/>}/>
      <Route path="book-details/:bookId" element={<BookDetails
isLoggedIn={!!user && user?.role !== 'admin'}/>}/>
      {
        !user &&
        <>
          <Route path="login" element={<Login updateUser={fetchData} />}/>
          ...
        </>
      }
      {
        user && user.role === 'user' && <>
          <Route path="cart" element={<Cart/>}/>
          ...
        </>
      }
      {
        user && user.role === 'admin' &&
        <>
          <Route path="add-book" element={<AddEditBook/>}/>
          ...
        </>
      }
    </Routes>
  </>
)

```

Тепер можемо створити сторінку магазину, яка буде доступна всім користувачам, незалежно від того, чи ввійшли вони в аккаунт. На головній сторінці буде представлений список доступних книг для перегляду.

```
const Shop = ({isLoggedIn}) => {
  const [books, setBooks] = useState([])
  const [isLoading, setIsLoading] = useState(true)
  ...
  return (
    <main>
      <div className="grid">{books.map(el => <Book book={el} key={el._id}
isLoggedIn={isLoggedIn}/>)}</div>
      {
        !isLoading && books.length === 0 && <h2 className="centered">No
books found</h2>
      }
    </main>
  )
}
```

Вигляд головної сторінки можна побачити на рисунку 3.2:

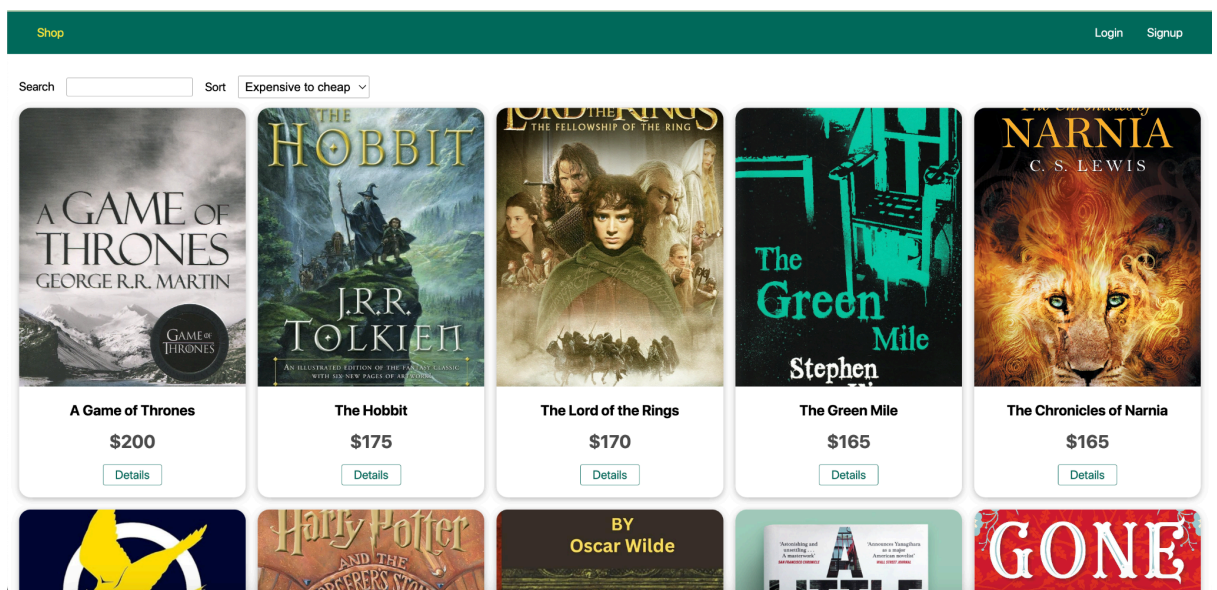


Рисунок 3.2. Головна сторінка

Однією з важливих функцій цієї сторінки є можливість пошуку та фільтрації книг аби дозволити користувачам швидко знаходити потрібні книги за різними критеріями, такими назва та ціна. Давайте реалізуємо цю логіку, щоб покращити зручність користування магазином:

```

const [search, setSearch] = useState('');
const [sort, setSort] = useState('-');
const [displayList, setDisplayList] = useState([]);
...
useEffect(() => {
    let filtered = books;
    if (search) {
        filtered = books.filter(book =>
            book.title.toLowerCase().includes(search.toLowerCase())
        );
    }
    if (sort !== '-') {
        filtered = [...filtered].sort((a, b) => {
            switch (sort) {
                ...
                default:
                    return 0;
            }
        });
    }

    setDisplayList(filtered);
}, [books, search, sort]);

```

Тут ми здійснюємо одночасно і фільтрацію і сортування щоб мати єдине джерело правди.

Приклад застосування сортування наведений на рисунках 3.3 та 3.4:

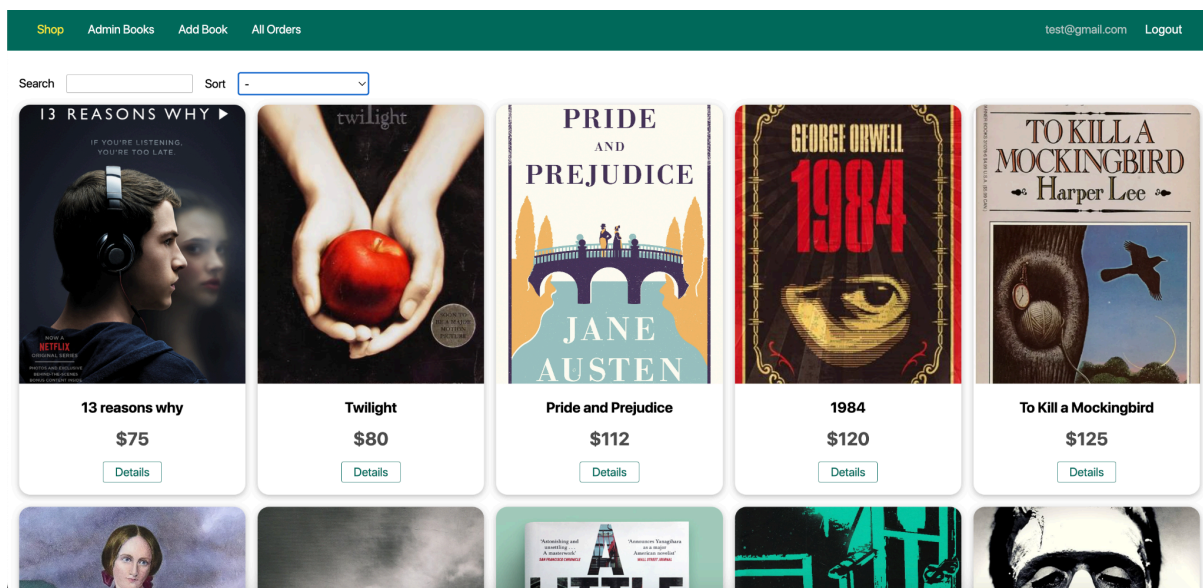


Рисунок 3.3. Книги без сортування

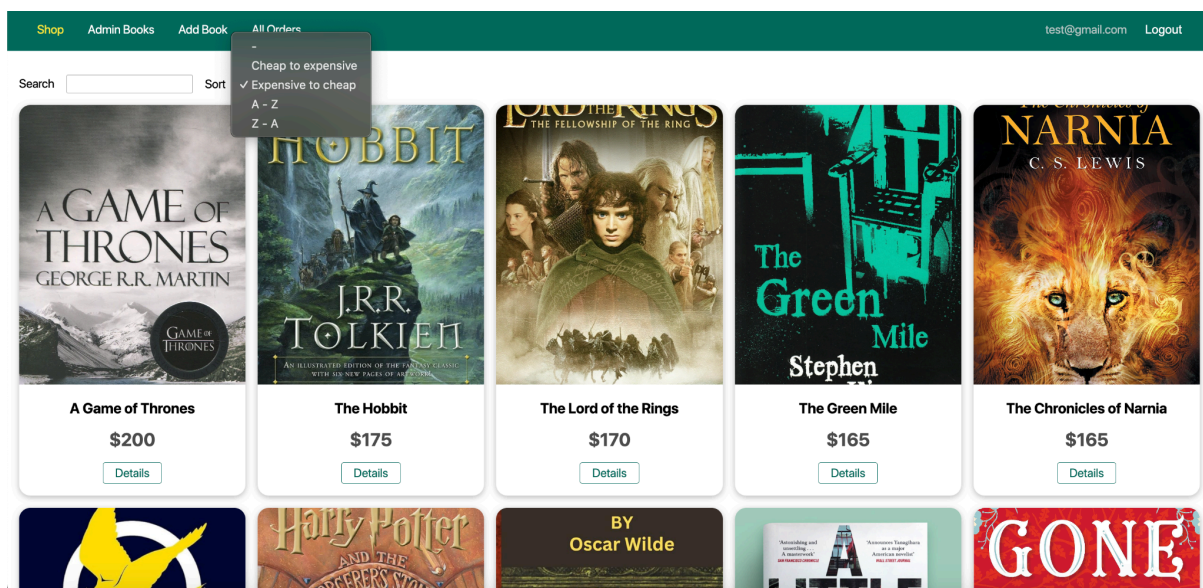


Рисунок 3.4. Відсортовані книги

Наступним кроком є створення сторінки для реєстрації користувача:

```

const Signup = () => {
  const [email, setEmail] = useState('')
  ...
  const onSignup = useCallback(async (e) => {
    ...
  }, [confirmPassword, email, navigate, password])

  return (
    <main>
      <form className="login-form" onSubmit={onSignup}>
        <div className="form-control">
          <label htmlFor="email">Email</label>
          <input
            className={validationError.toLocaleLowerCase().includes('email') ? 'invalid' : ''}
            type="text" name="email" id="email" value={email}
            onChange={onEmailChange}/>
        </div>
        ...
      </main>
    )
  }
}

```

На цій сторінці ми відправляємо запит на створення нового користувача на маршрут `/signup`. Якщо користувача успішно створено, користувач буде перенаправлений на сторінку входу. Результат цього процесу показано на рисунку 3.5:

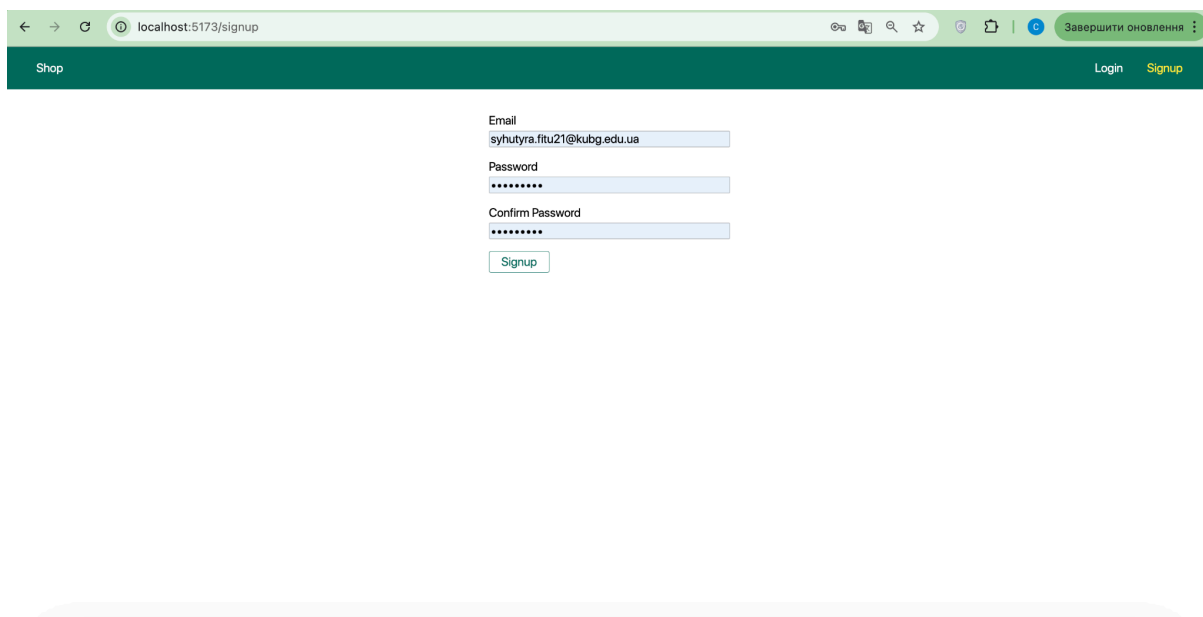


Рисунок 3.5. Сторінка для реєстрації

І також переглянемо базу даних на рисунку 3.6, щоб переконатися, що користувач був успішно створений:

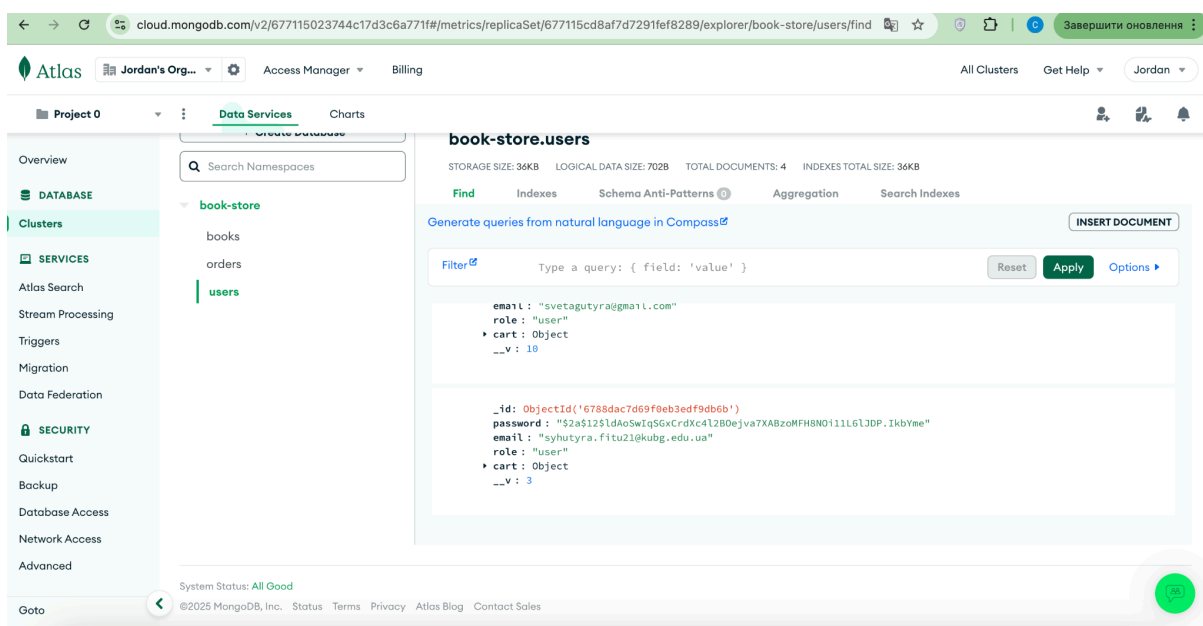
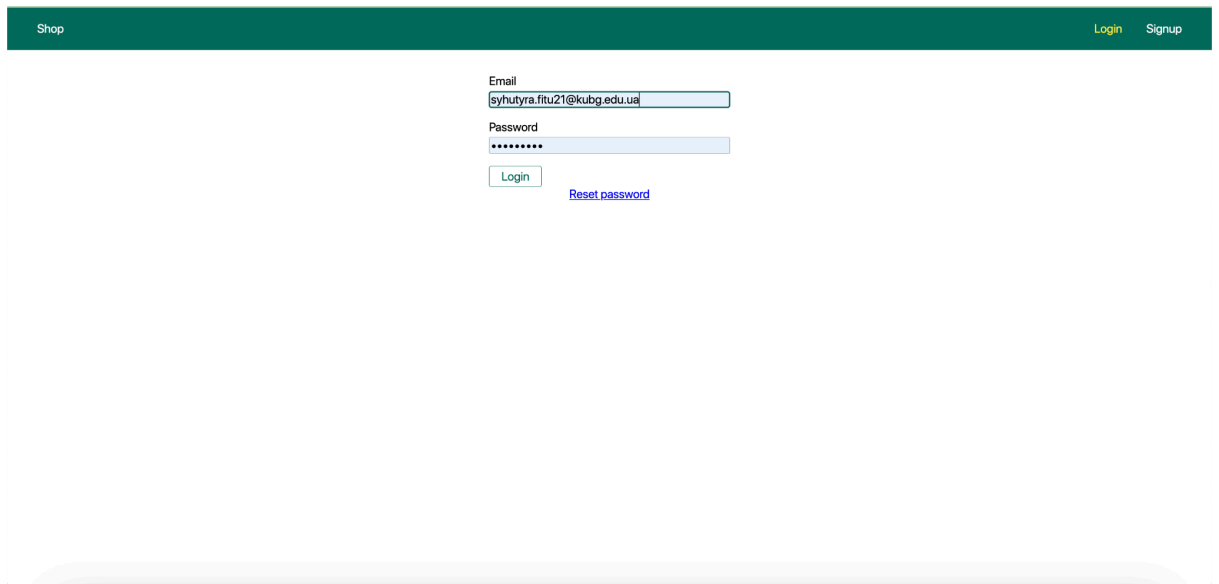


Рисунок 3.6. Створений користувач в базі даних

Далі перейдемо до сторінки з входом:

```
const Login = ({ updateUser }) => {
  const [email, setEmail] = useState('')
  ...
  return (
    <main>
      <form className="login-form" onSubmit={onSubmit}>
        <div className="form-control">
          <label htmlFor="email">Email</label>
          <input
            className={validationError.toLocaleLowerCase().includes('email') ||
            validationError.toLocaleLowerCase().includes('user') ? 'invalid' : ''}
            type="text" name="email" id="email" value={email}
            onChange={onEmailChange}/>
          ...
        </div>
      </form>
    </main>
  )
}
```

Результат видно на рисунку 3.7:



Shop Login Signup

Email
syhutyra.fitu21@kubg.edu.ua

Password

Login [Reset password](#)

Рисунок 3.7. Сторінка входу в акаунт

Після успішного входу користувача буде перенаправлено на головну сторінку, де в хедері відображатиметься емейл поточного користувача, як видно на рисунку 3.8.

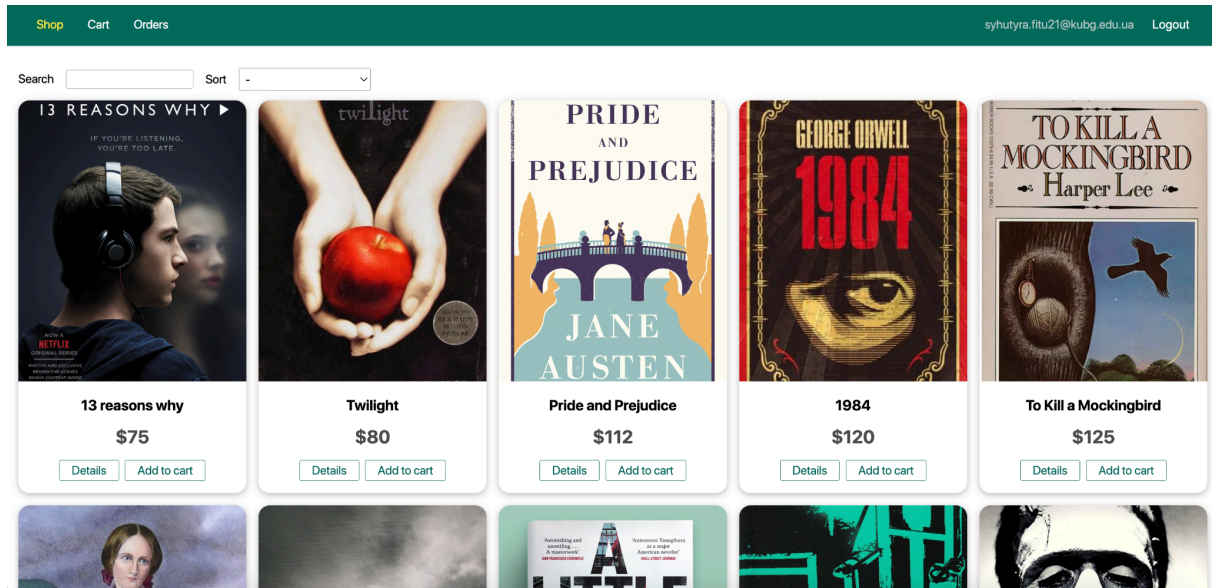


Рисунок 3.8. Головна сторінка після входу в додаток

Далі створимо сторінку корзини:

```
const Cart = () => {
  const [cartItems, setCartItems] = useState([])
  const [isLoading, setIsLoading] = useState(true)

  const fetchData = async () => {
    ...
  }

  const onDelete = useCallback(async (e, id) => {
    await fetch(`${API_URL}/cart/${id}`, {method: 'DELETE', headers:
getHeaders()})
    await fetchData()
  }, [])
}
```

```

const onOrder = useCallback(async () => {
    const sessionRes = await fetch(`${API_URL}/checkout`, {headers:
getHeaders()})

    const sessionData = await sessionRes.json()
    const stripe = await loadStripe(STRIPE_KEY)
    await stripe.redirectToCheckout({
        sessionId: sessionData.id,
    })
}, [])

return (
    <main>
        <ul className="cart__item-list">
            {cartItems.map(cartItem => (
                <li className="cart__item" key={cartItem.bookId._id}>
                    ...
                </li>
            ))}
        </ul>
        {
            cartItems.length > 0 && (
                <>
                    <hr/>
                    <div className="centered">
                        ...
                    </div>
                </>
            )
        }
    </main>
)

```

Тут ми отримуємо всі книги, які знаходяться в корзині, і створюємо дві функції: перша — для видалення товару з корзини, яка відправляє відповідний запит на сервер, і друга — для оформлення замовлення, перед яким буде здійснена оплата.

Протестуємо дану логіку. Додамо 1 книжку “Pride and Prejudice” 3 рази як це показано на рисунку 3.9:

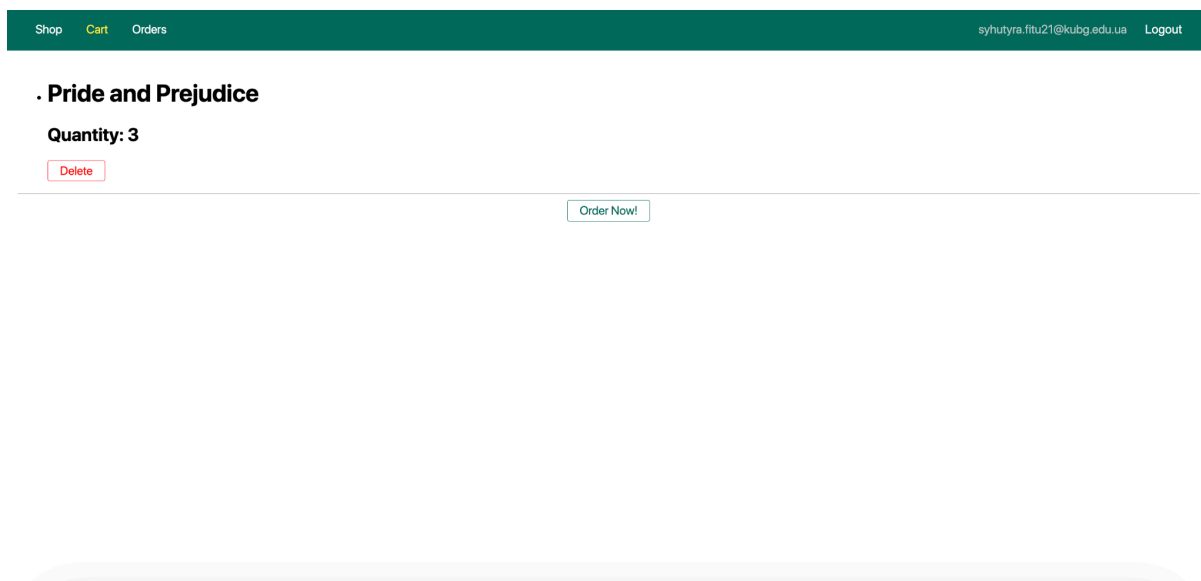


Рисунок 3.9 Товари корзини

Натиснувши на кнопку Order now, ми перейдемо на сторінку оплати, де введемо тестові дані карти, як показано на рисунку 3.10:

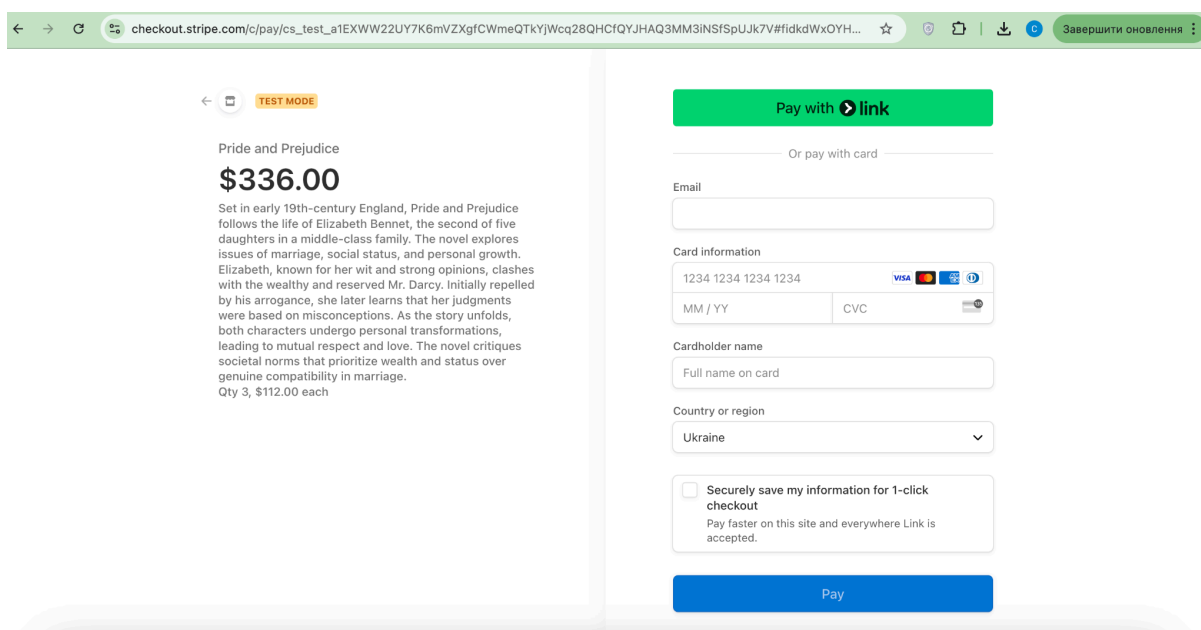


Рисунок 3.10. Сторінка з оплатою замовлення

Після цього нас перенаправить на сторінку з замовленнями, яку ми розглянемо далі.

Сторінка з замовленнями:

```
const Orders = () => {
  const [orders, setOrders] = useState([])
  const fetchData = async () => {
    ...
  }
  useEffect(() => {
    fetchData()
  }, [])
  const openInvoice = useCallback(async (id) => {...
  }, [])
  return (
    <main>
      <ul className="orders">
        {orders.map(el =>
          <li className="orders__item" key={el._id}>
            <h1 className="orders__item-header">Order - # {el._id}
              <p className="orders__item-header-invoice" onClick={()
=> openInvoice(el._id)}>Invoice</p>
            </h1>
            ...
          </li>
        )}
      </ul>
    </main>
  )
}
```

Тут ми отримуємо замовлення з бекенду та створюємо функцію для отримання виписки по замовленню. Виписка може бути використана для перегляду деталей конкретного замовлення, в нашому випадку у форматі PDF.

Сторінка з замовленнями продемонстрована на рисунку 3.11:

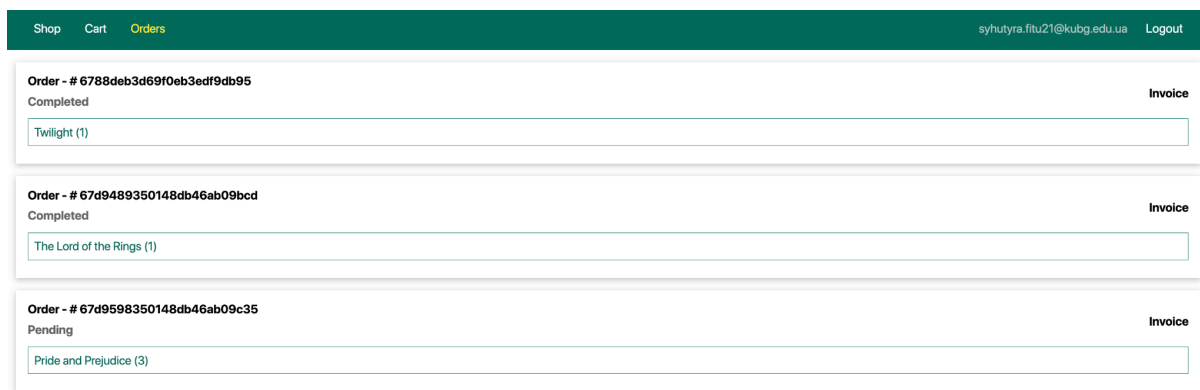


Рисунок 3.11. Сторінка з замовленнями

І також отримаємо виписку по замовленню, яке ми нещодавно зробили з трьома книгами. Результат цього можна побачити на рисунку 3.12:

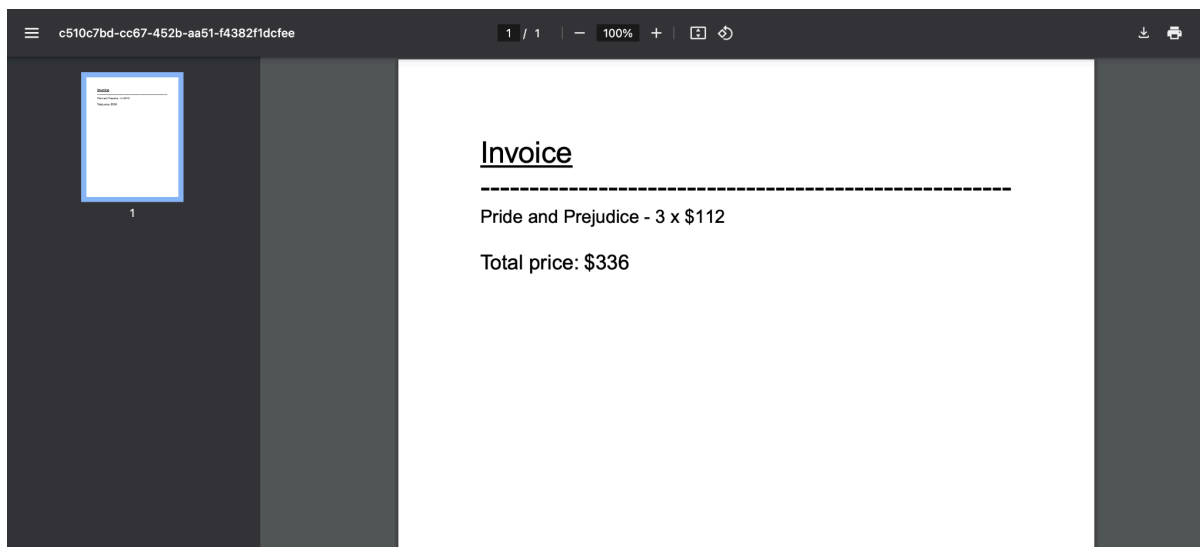


Рисунок 3.12. Виписка по замовленню наших трьох книг

4.3 Реалізація клієнтської частини для менеджера (редагування каталогу)

Далі ми переходимо в акаунт адміністратора. Після цього вигляд застосунку змінюється: зникають корзина і замовлення, натомість з'являються можливості додавання книги та каталог книг, створених цим користувачем, який доступний для редагування. Це продемонстровано на рисунку 3.13.

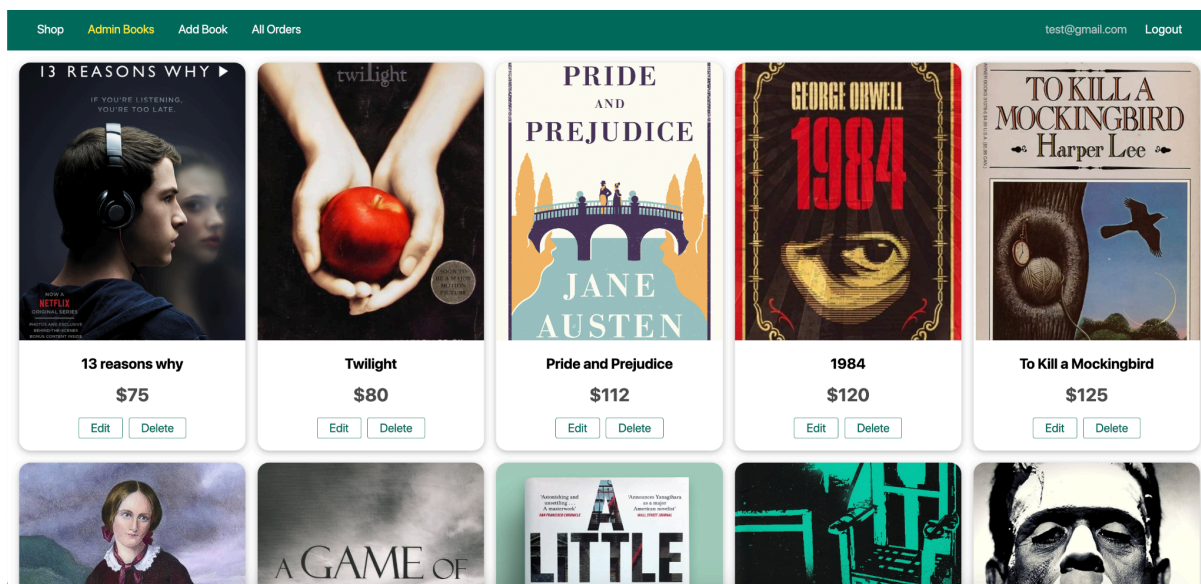


Рисунок 3.13. Головна сторінка зі сторони адміністратора

Далі розглянемо сторінку з товарами для адміністратора, яка створюється майже ідентично головній сторінці.

```
const AdminBooks = ({isLoggedIn}) => {
  const [books, setBooks] = useState([])
  const fetchData = async () => {
    ...
  }
  useEffect(() => {
    fetchData()
  }, [])
  return (
    <main>
      <div className="grid">{books.map(el => <Book book={el}
        isLoggedIn={isLoggedIn} isAdmin={true} fetchData={fetchData} key={el._id}/>)}</div>
    </main>
  )
}
```


Як результат, маємо сторінку з книгами, створеними цим користувачем.

Далі перейдемо до сторінки з додаванням та редагуванням книг:

```
const AddEditBook = () => {
  const {bookId} = useParams()
  ...
  useEffect(() => {
    ...
    const fetchData = async () => {
      const book = await fetch(`${API_URL}/books/${bookId}`)
      ...
    }
    if (bookId) {
      fetchData()
    }
  }, [bookId])
}
```

```
if (bookId) {
  fetchData()
}
}, [bookId])
const onFormSubmit = useCallback(async(e) => {
  ...
  const res = await fetch(`${API_URL}/admin/books${bookId ? `/${bookId}` : ''}`, { ...
  }, [bookId, description, imageFile, navigate, price, title])

  return (
    <form className="book-form" onSubmit={onFormSubmit}>
      <div className="form-control">
        <label htmlFor="title">Title</label>
        <input
className={validationError.toLocaleLowerCase().includes('title') ? 'invalid' : ''}
          type="text" value={title} onChange={onTitleChange}/>
        ...
        <button className="btn" type="submit">
          {bookId ? 'Update book' : 'Add book'}
        </button>
      </div>
      {
        validationError.length ?
        <div className="user-message
user-message--error">{validationError}</div> : null
      }
    </form>
  )
}
```

Якщо в шляху передано id, ми будемо редагувати існуючу книгу. Якщо ж id немає, то будемо додавати нову книгу. У разі, коли id присутнє, ми також

відправляємо запит на отримання інформації про книгу. В методі onFormSubmit ми перевіряємо, чи є id, і в залежності від цього відправляємо відповідні запити.

Ми можемо протестувати спочатку створення книги, як показано на рисунку 3.14, а потім редагування створеної книги, як видно на рисунку 3.15.

Shop Admin Books **Add Book** All Orders test@gmail.com Logout

Title
Night in Lisbon

Image URL
Вибрати файл The.Night.In...bon.cover.jpg

Price
108

Description
Night in Lisbon is a poignant novel set during World War II, revolving around the lives of two refugees, Lena and Albert, who are fleeing Nazi-occupied Europe. The story takes place on a single night in Lisbon, a city that serves as

Add book

Рисунок 3.14. Створення книги

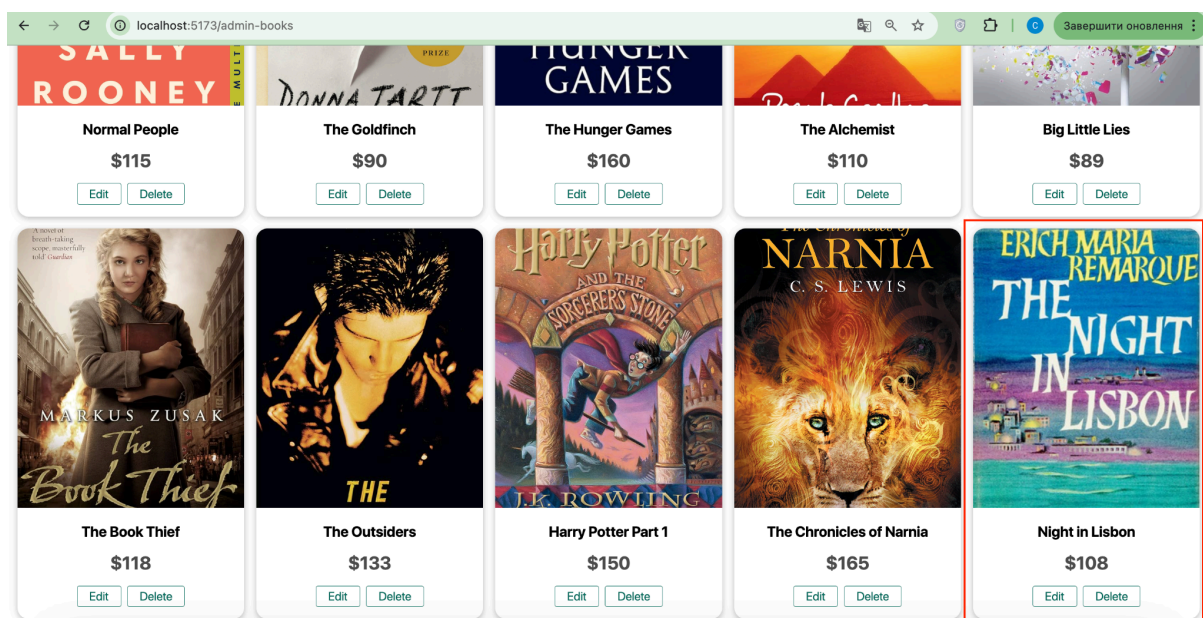


Рисунок 3.15. Створена книга в каталозі

Shop Admin Books **Add Book** All Orders
test@gmail.com Logout

Title
Night in Lisbon 2

Image URL
Вибрати файл The.Night.In...bon.cover.jpg

Price
108

Description
Night in Lisbon is a poignant novel set during World War II, revolving around the lives of two refugees, Lena and Albert, who are fleeing Nazi-occupied Europe. The story takes place on a single night in Lisbon, a city that serves as

Add book

Рисунок 3.16. Редагування даної книги (додала до назви “2”)

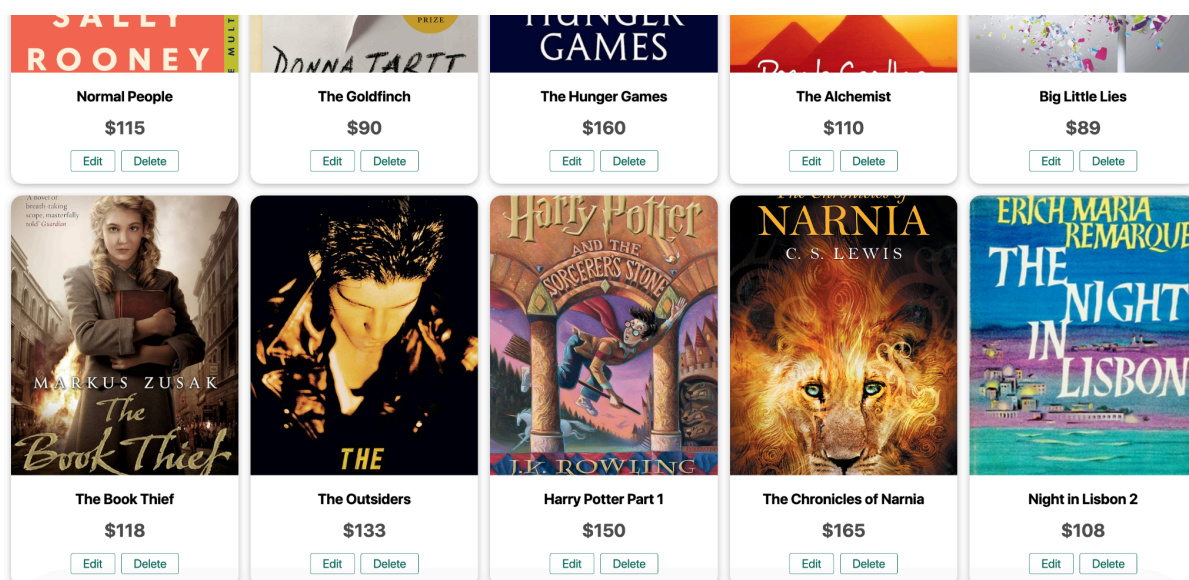


Рисунок 3.17. Успішно відредагована книга

Сторінка з усіма замовленнями та можливістю оновлення статусу продемонстрована на рисунку 3.18.

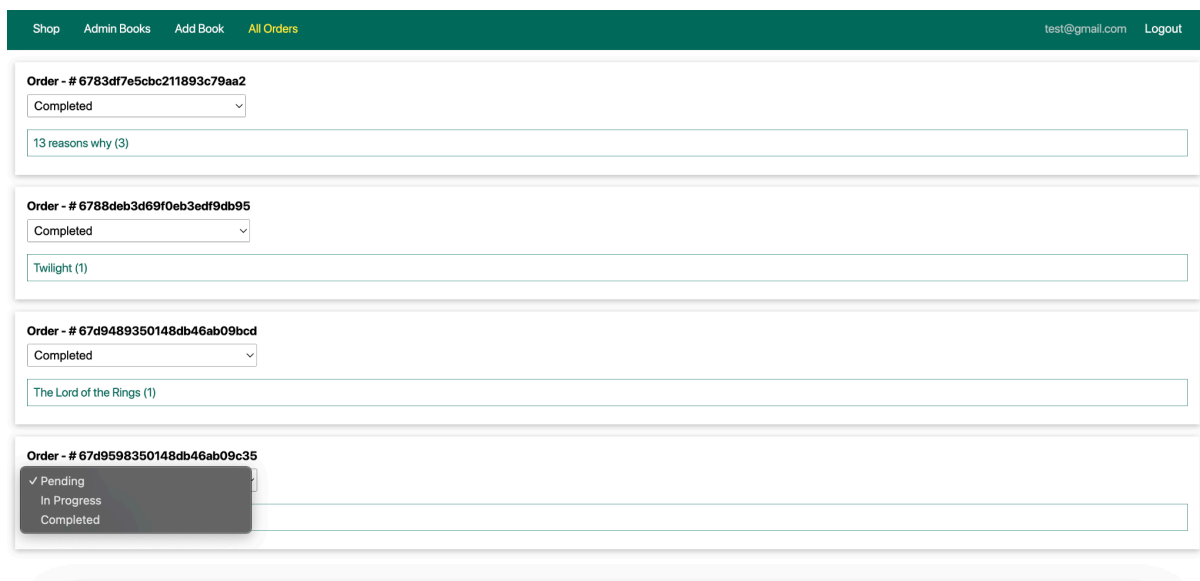


Рисунок 3.18. Оновлення статусу замовлення

ВИСНОВКИ

Результатом цієї дипломної роботи є успішна розробка сучасної веб-платформи для автоматизації діяльності книжкового магазину з використанням стеку технологій NodeJS, React і MongoDB. Розроблена платформа повністю відповідає встановленим функціональним вимогам та забезпечує ефективну автоматизацію основних бізнес-процесів книгарні. У системі реалізовано два основних типи користувачів – адміністратори та клієнти, з відповідним розподілом прав доступу та функціональних можливостей.

Аналіз предметної області та існуючих рішень визначає найкращий набір технологій розробки. Використання NodeJS забезпечує швидку та надійну роботу серверної частини, React дозволяє створювати сучасні та інтуїтивно зрозумілі інтерфейси користувача, а MongoDB забезпечує гнучкість, необхідну для обробки даних. Проектна архітектура системи базується на сучасній моделі клієнт-сервер, і це дає високу масштабованість і можливість подальшого розширення функціональності.

В рамках розробки реалізовано повний набір функціоналу для всіх користувачів системи. Клієнти отримують зручний інтерфейс для перегляду

каталогу книг, користування кошиком, розміщення замовлень та доступу до історії покупок. Системні адміністратори мають доступ до повного функціоналу для керування каталогами товарів, обробки замовлень і моніторингу активності користувачів.

Практична цінність розробленої платформи полягає в значному підвищенні продуктивності роботи персоналу магазину, покращенні якості обслуговування клієнтів та оптимізації процесів управління магазином. Загалом клієнти отримують кращий сервіс: зручніший процес пошуку книг, можливість швидко зробити замовлення та одразу отримати виписку по ньому.

Створена структура бази даних оптимально відображає зв'язки між об'єктами системи та забезпечує ефективне зберігання та обробку даних, що забезпечує міцну основу для подальшого розвитку платформи шляхом додавання нових функціональних модулів, інтеграції з іншими системами та розширення відповідно до зростаючих потреб бізнесу.

Отже, розроблена платформа є повноцінним програмним рішенням, яке відповідає вимогам сучасних веб-додатків і може бути успішно впроваджено в роботу книгарень будь-якого розміру для підвищення ефективності їх діяльності. Система має великий потенціал для подальшого розвитку та адаптації до специфічних потреб конкретних реалізацій.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. React - A JavaScript library for building user interfaces. URL:
<https://reactjs.org/> (дата звернення: 06.12.2024).
2. Node.js - JavaScript runtime built on Chrome's V8 JavaScript engine. URL:
<https://nodejs.org/> (дата звернення: 06.12.2024).
3. MongoDB – The database for modern applications. URL:
<https://www.mongodb.com/> (дата звернення: 18.12.2024).
4. Full Stack Project Tutorial – Create A Notes App Using React and Node.js.
URL:
<https://www.freecodecamp.org/news/full-stack-project-tutorial-create-a-notes-app-using-react-and-node-js/> (дата звернення: 12.01.2025).
5. Brown E. Web Development with Node and Express. O'Reilly Media, 2019. – 300 p.
6. Effective Web Design Techniques You Should Apply. URL:
<https://visualhierarchy.co/effective-web-design-techniques-that-you-should-apply> (дата звернення: 18.02.2025).
7. В Машкіна, ВО Абрамов, ТІ Носенко, ЄВ Іваніченко. Навчально-методичний посібник до виконання і захисту кваліфікаційної роботи бакалавра спеціальностей 122 Комп'ютерні науки для здобувачів вищої освіти денної форми навчання, Київський столичний університет імені Бориса Грінченка. Київ, 2024.- 43 с.
8. Теоретичні та практичні аспекти використання математичних методів та інформаційних технологій в освіті й науці: моногр. / за заг. ред. О. Литвин. — К.: Київ. ун-т ім. Б. Грінченка, 2021. — 332 с.
9. Яскевич, Владислав Олександрович (2023) *JavaScript бібліотека React Навчальний посібник для студентів спеціальності Комп'ютерні науки* Київський університет імені Бориса Грінченка, Україна, Київ.
<https://elibrary.kubg.edu.ua/id/eprint/48587/>
10. MongoDB for Beginners. URL: <https://www.mongodb.com/basics> (дата

- звернення: 17.02.2025).
11. React Patterns and Techniques. URL: <https://reactpatterns.com/> (дата звернення: 07.03.2025).
 12. Building High-Performance Full-Stack Apps with React, Node.js & MongoDB: A Journey in Scalability, Speed & Solutions. URL: <https://dev.to/mukhilpadmanabhan/building-high-performance-full-stack-apps-with-react-nodejs-mongodb-a-journey-in-scalability-speed-solutions-3fk0> (дата звернення: 20.03.2025).
 13. Alex Banks, Eve Porcello. Learning React: Functional Web Development with React and Redux. O'Reilly Media, 2017. – 350 p.
 14. Ефективна робота з базами даних MongoDB. URL: <https://www.dataversity.net/mongodb-best-practices> (дата звернення: 21.03.2025).
 15. Understanding REST APIs. URL: <https://restfulapi.net/> (дата звернення: 27.02.2025).
 16. JavaScript Best Practices for Modern Development. URL: <https://jsbestpractices.com/> (дата звернення: 28.03.2025).
 17. Mern Stack Front to Back: Full Stack React, Redux & Node.js. URL: <https://udemy.com/course/mern-stack> (дата звернення: 16.03.2025).
 18. React and Modern JavaScript Development. URL: <https://flaviocopes.com/react>
 19. Modern Web Development with Node.js and React. URL: <https://medium.com/web-development> (дата звернення: 01.04.2025).
 20. Build Robust Web Apps with MongoDB. URL: <https://university.mongodb.com/>
 21. W3Schools: MongoDB Tutorial. URL: <https://www.w3schools.com/mongodb/> (дата звернення: 25.01.2025).
 22. Express.js Guide. URL: <https://expressjs.com/en/starter/guide.html> (дата звернення: 20.01.2025).

