НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

ФАКУЛЬТЕТІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри
комп'ютерних наук
/Голуб Б.Л., доц., к.т.н. /
підпис ПІБ, вчене звання і ступінь

«___» _____2025 р

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Програмне забезпечення інформаційної системи для продажу книгарної продукції»

Спеціальність 121 «Інженерія програмного забезпечення»

Гарант освітньої програми
к.т.н., доцент / Голуб Б.Л./
Науковий ступень та вчене звання підпис ПІБ

Керівник бакалаврської кваліфікаційної роботи : / Василюк-Зайцева С.В./
підпис ПІБ

Виконав: / Зибін Н.С./
підпис ПІБ

КИЇВ-2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

ФАКУЛЬТЕТІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ЗАТВЕРДЖУЮ

Завідувач кафедри
комп'ютерних наук
ii
_____ / Голуб Б.Л., доцент, к.т.н /
підпис
" 16 " грудня 2023 р.

ЗАВДАННЯ

на виконання бакалаврської кваліфікаційної роботи

студенту Зибіну Нікіті Сергійовичу

Спеціальність 121 «Інженерія програмного забезпечення»

1. Тема роботи: Програмне забезпечення інформаційної системи для продажу книгарної продукції Затверджена наказом ректора НУБіП України № 2248 "С" від 16.12.2024

2. Термін подання завершеної роботи на кафедру 2024 . ____. ____
рік, місяць, число

3. Вихідні дані до роботи: опис програмного забезпечення

4. Перелік питань що розглядаються:

1. Аналіз та моделювання предметної області.
2. Постановка задачі та аналіз вимог.
3. Розробка інформаційного та програмного забезпечення.
4. Впровадження, рекомендації щодо використання.

Керівник бакалаврської кваліфікаційної роботи_____ / Василюк-Зайцева С.В./
підпис ініціали та прізвище

Завдання прийняв до виконання _____ / Зибін Н.С. /
підпис ініціали та прізвище

Дата отримання завдання 2024 . 12 .
рік, місяць,

КАЛЕНДАРНИЙ ПЛАН

	Назва етапів виконання бакалаврської кваліфікаційної роботи	**Строк виконання етапів бакалаврської кваліфікаційної роботи**	**Примітка**
1.	Отримання завдання	16 лютого 2024	Виконав
2.	Аналіз предметної області	19 – 29 лютого 2024	Виконав
3.	Моделювання предметної області	1 – 24 березня 2024	Виконав
4.	Проєктування програмної системи	25 березня – 6 квітня 2024	Виконав
5.	Розробка програмної системи	7 квітня – 27 квітня 2024	Виконав
6.	Тестування програмної системи	28 квітня – 4 травня 2024	Виконав
7.	Розгортання та експлуатація програмного забезпечення	5 травня – 11 травня 2024	Виконав
8.	Оформлення записки	травень 2024	Виконав
9.	Перевірка на плагіат	26 травня – 29 травня 2024	
10.	Проходження попереднього захисту	1 червня 2024	
11.	Захист роботи	9 червня 2024	

Студент _____ / Зибін Н.С. /
(підпис) (прізвище та ініціали)

Керівник бакалаврської кваліфікаційної роботи _____ / Василюк-Зайцева С.В./
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дана робота присвячена розробці інформаційної системи для продажу книжкової продукції в онлайн-форматі. Предметна область охоплює сферу електронної комерції, зокрема книжкову торгівлю, де актуальними є автоматизація процесів продажу, управління асортиментом та взаємодія з клієнтами. Розроблена система вирішує проблеми швидкого доступу до каталогу книг, спрощення оформлення замовлень, оптимізації управління запасами та підвищення зручності для користувачів.

Для реалізації використано сучасні технології: React.js для інтерактивного інтерфейсу, Node.js з Express.js для серверної логіки та SQLite як базу даних. Система включає модулі для перегляду книг, кошика, оформлення замовлень, а також адміністрування для менеджерів. Робота спрямована на підвищення ефективності роботи книжкового магазину та популяризацію читання через зручний онлайн-доступ.

ABSTRACT

This work is dedicated to developing an information system for selling book products in an online format. The subject area encompasses the field of e-commerce, particularly the book trade, where automation of sales processes, assortment management, and customer interaction are vital. The developed system addresses issues such as quick access to the book catalog, simplification of order processing, optimization of inventory management, and enhanced user convenience.

Modern technologies were utilized for implementation: React.js for the interactive interface, Node.js with Express.js for server-side logic, and SQLite as the database. The system includes modules for book browsing, a shopping cart, order processing, as well as administration for managers. The work aims to improve the efficiency of bookstore operations and promote reading through convenient online access.

Зміст

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

Скорочення	Визначення
UML	Уніфікована мова моделювання
API	Інтерфейс прикладного програмування
CSS	Каскадні таблиці стилів
HTML	Мова розмітки гіпертексту
SQLite	Реляційна база даних
UI	User interface – інтерфейс користувача
REST	Архітектурний стиль розробки веб-сервісів
JSON	JavaScript Object Notation – текстовий формат для структурованих даних
JWT	JSON Web Token – формат токена для аутентифікації
React.js	JavaScript-бібліотека для побудови інтерфейсів
HTTP	HyperText Transfer Protocol – протокол передачі гіпертексту
Express.js	Фреймворк web-додатків для Node.js

ВСТУП

Сучасний розвиток інформаційних технологій суттєво впливає на організацію торгівлі, зокрема у сфері продажу книжкової продукції. Зростання популярності електронної комерції, зміна споживчих уподобань та необхідність швидкого доступу до широкого асортименту товарів роблять актуальним створення ефективних програмних систем для автоматизації процесів у книжкових магазинах. Розробка інформаційної системи для продажу книжкової продукції є відповіддю на ці виклики, адже вона дозволяє оптимізувати процеси управління асортиментом, замовленнями та взаємодією з клієнтами. Актуальність цього завдання зумовлена не лише економічними факторами, такими як підвищення конкурентоспроможності торговельних підприємств, а й соціальною значущістю, адже доступність книг сприяє розвитку культури, освіти та інтелектуального потенціалу суспільства. У контексті цифрової трансформації, що охоплює всі сфери життя, створення подібних систем є необхідним кроком для адаптації книжкової індустрії до сучасних умов.

Метою розробки даного програмного додатку є створення зручного та функціонального інструменту для автоматизації продажу книжкової продукції, що відповідає потребам як покупців, так і працівників книжкового магазину. З огляду на актуальність теми, мета полягає в забезпеченні швидкого доступу до каталогу книг, спрощенні процесу оформлення замовлень, управлінні запасами та наданні аналітичних даних для менеджерів. Розроблений додаток спрямований на підвищення ефективності роботи магазину, зменшення часу на обробку замовлень та покращення користувацького досвіду клієнтів. Такий підхід дозволяє не лише задовольнити потреби сучасного ринку, а й сприяти популяризації читання через зручний онлайн-доступ до літератури.

Для реалізації програмного додатку використано сучасні методи та технології розробки програмного забезпечення. Основою клієнтської частини є бібліотека React.js, яка забезпечує створення інтерактивного інтерфейсу з

високою швидкодією та модульністю. На серверній стороні застосовано технологію Node.js у поєднанні з фреймворком Express.js для обробки запитів та забезпечення взаємодії з базою даних. Як система управління базами даних обрано SQLite, що відзначається легкістю інтеграції та достатньою продуктивністю для невеликих і середніх проєктів. Для моделювання предметної області використано методи об’єктно-орієнтованого аналізу та проєктування, зокрема побудову ER-діаграм, діаграм класів і компонентів, що дозволило чітко структурувати систему. Крім того, застосовано підходи до асинхронного програмування для оптимізації взаємодії між клієнтом і сервером, а також методи тестування для забезпечення надійності системи.

Структура пояснювальної записки складається з чотирьох основних розділів, висновків, списку використаних джерел та додатків, загальним обсягом 85 сторінок, із використанням 26 джерел та 1 додаток. У першому розділі "Системний аналіз предметної області" (18 сторінок) описано предметну область книжкової торгівлі, проведено аналіз вимог, побудовано моделі предметної області, здійснено огляд існуючих рішень та сформульовано завдання розробки. Другий розділ "Проєктування інформаційного та програмного забезпечення" (16 сторінок) присвячено розробці логічної моделі даних, діаграм класів, пакетів і компонентів, що слугують основою для реалізації системи. Третій розділ "Розробка інформаційного та програмного забезпечення" (12 сторінок) деталізує створення бази даних, вибір інструментів, алгоритмізацію та програмування модулів, включаючи наведений код. Четвертий розділ "Рекомендації щодо впровадження та експлуатації системи" (27 сторінок) охоплює тестування, діаграму розгортання, вимоги до обладнання та опис роботи системи. У висновках (2 сторінки) підведено підсумки роботи, а в додатках представлено лістинг коду, діаграми та результати тестування.

Таким чином, розробка інформаційної системи для продажу книжкової продукції є важливим внеском у вдосконалення процесів електронної комерції у цій сфері.

1. СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Предметна область даного дослідження охоплює сферу продажу книжкової продукції через інформаційну систему, яка базується на використанні сучасних технологій веб-розробки та баз даних. У контексті глобалізації та стрімкого розвитку електронної комерції, книжковий ринок зазнає значних трансформацій, що зумовлює необхідність створення ефективних програмних рішень для забезпечення доступності літератури, зручності користувачів та оптимізації бізнес-процесів. Розроблена інформаційна система, представлена у вигляді кодів програми (app.js, server.js, seed.js), є прикладом реалізації такого рішення, яке спрямоване на автоматизацію торгівлі книжковою продукцією в онлайн-форматі.

Основною метою предметної області є організація взаємодії між учасниками процесу купівлі-продажу книг – покупцями, менеджерами та адміністраторами системи. Покупці отримують можливість переглядати каталог книг, обирати літературу за категоріями, додавати товари до кошика, оформлювати замовлення та здійснювати оплату. Менеджери, у свою чергу, відповідають за обробку замовлень, оновлення асортименту, контроль наявності товарів на складі та управління фінансовими потоками. Адміністративна частина системи забезпечує підтримку бази даних, захист інформації та моніторинг роботи платформи. Таким чином, предметна область охоплює як комерційні, так і операційні компоненти діяльності книжкового магазину.

Ключові об’єкти предметної області представлені в таблиці 1.1.

Таблиця 1.1

Ключові об’єкти предметної області

№	Назва	Опис
1	Книги	основний товар, який характеризується такими атрибутами, як назва, автор, категорія, ціна, опис, зображення та кількість на складі. У системі передбачено різні категорії літератури (наприклад, українська класика, дитяча література, фантастика), що відображає різноманітність читацьких інтересів
2	Користувачі	поділяються на дві основні ролі – покупці (клієнти) та менеджери. Кожен користувач має персональні дані (email, пароль, ім’я) та відповідний рівень доступу до функціоналу системи
3	Замовлення	включають інформацію про покупця, перелік обраних книг, загальну суму, спосіб доставки, статус оплати та поточний стан (наприклад, "Новий", "Оброблений", "Доставлений")
4	Кошик	тимчасове сховище обраних книг, яке дозволяє користувачу формувати замовлення перед його остаточним оформленням
5	Фінансові операції	відображають рух коштів, зокрема баланс менеджерів, який оновлюється після успішного завершення замовлень

Функціонування системи базується на взаємодії між фронтендом (інтерфейсом користувача), бекендом (серверною логікою) та базою даних [1]. Фронтенд, реалізований за допомогою React у файлі app.js, забезпечує інтерактивність та зручність навігації [2]. Користувач може переглядати головну сторінку з категоріями книг, детальну інформацію про кожну книгу, додавати товари до кошика чи обраного, а також здійснювати пошук за ключовими словами. Бекенд, описаний у server.js, обробляє запити від клієнта, взаємодіє з базою даних SQLite та забезпечує логіку авторизації, управління замовленнями та асортиментом. Файл seed.js слугує для початкового наповнення бази даних тестовими записами, що дозволяє змоделювати реальні умови роботи системи.

Основні процеси предметної області описані в таблиці 1.2.

Таблиця 1.2

Основні процеси предметної області

№	Назва	Опис
1	Реєстрація та авторизація користувачів	створення облікового запису з визначенням ролі (покупець чи менеджер) та подальший вхід у систему
2	Перегляд і вибір товарів	пошук книг за категоріями, авторами чи назвами з можливістю ознайомлення з детальною інформацією
3	Формування замовлення	додавання книг до кошика, вибір способу доставки та оплати, підтвердження покупки
4	Обробка замовлень	зміна статусу замовлення менеджером, оновлення складських запасів та фінансового балансу
5	Адміністрування	додавання, редагування чи видалення книг із каталогу, а також моніторинг стану системи

Предметна область також враховує зовнішні фактори, такі як попит на певні жанри літератури, логістичні особливості доставки та необхідність адаптації інтерфейсу до різних пристроїв (десктопів, планшетів, смартфонів). Реалізована система демонструє гнучкість завдяки адаптивному дизайну, що забезпечує комфортне використання незалежно від типу пристрою. Крім того, використання локального сховища (LocalStorage) для збереження даних кошика, обраного та переглянутих книг підкреслює орієнтацію на зручність користувача, дозволяючи зберігати вибір навіть після закриття браузера.

Отже, предметна область інформаційної системи для продажу книжкової продукції є комплексною системою, що інтегрує торговельні, управлінські та інформаційні процеси. Вона спрямована на спрощення доступу до літератури, підвищення ефективності роботи персоналу та забезпечення позитивного користувацького досвіду. Аналіз цієї області дозволяє виявити ключові вимоги до програмного забезпечення, зокрема зручність інтерфейсу, швидкість обробки запитів, надійність зберігання даних та гнучкість у налаштуванні функціоналу під потреби бізнесу.

1.2 Діаграма прецедентів

Діаграма прецедентів (рис. 1.1) розроблена для інформаційної системи продажу книжкової продукції відображає взаємодію двох основних акторів — Покупця та Менеджера — із системою. Покупець має можливість переглядати каталог книг, здійснювати пошук, додавати книги до кошика, оформлювати замовлення, переглядати обране та керувати вмістом кошика. Менеджер може додавати, редагувати й видаляти книги, керувати замовленнями та переглядати баланс. Обидва актори мають доступ до функції виходу з системи. Діаграма відображає чіткий розподіл ролей і функціональних можливостей, що узгоджується з архітектурою клієнт-серверної взаємодії та базою даних SQLite, описаною в коді.

Рис. 1.1. Діаграма прецедентів

Таким чином, вона ілюструє ключові сценарії використання системи, забезпечуючи її відповідність темі продажу книжкової продукції.

1.3 Аналіз вимог до програмної системи

Розробка програмного забезпечення інформаційної системи для продажу книжкової продукції потребує ретельного аналізу вимог, які забезпечують її функціональність, зручність використання та відповідність потребам користувачів. На основі реалізованих кодів (app.js, server.js, seed.js) можна виділити ключові фактори, які формують вимоги до системи, враховуючи особливості предметної області — книжковий онлайн-магазин. Аналіз вимог проводиться з урахуванням потреб різних груп користувачів (клієнтів та менеджерів), а також технічних і функціональних питань реалізації.

Функціональні вимоги наведені в таблиці 1.3.

Таблиця 1.3

Функціональні вимоги

№	Вимоги	Пояснення
1	2	3
1	Реєстрація та автентифікація користувачів	Система повинна забезпечувати можливість створення облікових записів для нових користувачів та автентифікацію зареєстрованих користувачів [3]. Це включає введення персональних даних (email, пароль, ім’я) та вибір ролі (клієнт або менеджер). У коді app.js функції login та register реалізують ці вимоги через API-запити до серверної частини, де в server.js обробляються відповідні маршрути /api/login та /api/register. Важливою вимогою є захист даних користувачів, що частково реалізовано через зберігання їх у базі даних SQLite із можливістю перевірки унікальності email
2	Каталог книг та пошук	Користувачі повинні мати доступ до повного каталогу книжкової продукції з можливістю фільтрації за категоріями та пошуку за ключовими словами (назва, автор) [4]. У додатку це реалізовано через компонент HomePage, який відображає книги з урахуванням категорій, та функцію searchBooks, що фільтрує книги за запитом користувача. Система має забезпечувати швидке завантаження даних про книги з бази (функція fetchBooks), а також відображення детальної інформації про книгу (компонент BookDetailsPage)

Продовження таблиці 3.1

1	2	3
3	Кошик і оформлення замовлення	Клієнти повинні мати можливість додавати книги до кошика, змінювати їх кількість та видаляти позиції [5]. Компонент CartPage у коді відповідає за ці функції, дозволяючи оновлювати кількість (updateCartItemQuantity) та видаляти товари (removeFromCart). Оформлення замовлення (компонент CheckoutPage) вимагає введення даних доставки та оплати, після чого створюється запис у базі даних через API /api/orders. Вимога до цієї підсистеми — точність підрахунку загальної вартості та збереження історії замовлень
4	Управління обраним	Користувачі можуть позначати книги як обрані для швидкого доступу в майбутньому. Функція toggleFavorite у коді дозволяє додавати або видаляти книги зі списку обраного, який зберігається в localStorage та відображається в FavoritesPage. Це підвищує зручність взаємодії з системою, відповідаючи потребам клієнтів у персоналізації
5	Адміністрування для менеджерів	Менеджери повинні мати доступ до інструментів управління книгами (додавання, редагування, видалення) та замовленнями (зміна статусів) [6]. У коді це реалізовано через компоненти ManageBooksPage та ManageOrdersPage, які взаємодіють із серверними API (/api/books, /api/orders). Вимоги включають можливість оновлення балансу менеджера (fetchManagerAccount) та відображення всіх замовлень для обробки
6	Сповіщення користувачів	Система повинна інформувати користувачів про успішність або помилки операцій (наприклад, додавання до кошика, оформлення замовлення) [7]. Функція showNotification у app.js відображає тимчасові сповіщення з різними типами (успіх, помилка, інформація), що покращує зворотний зв’язок із користувачем

Нефункціональні вимоги показані в таблиці 1.4.

Таблиця 1.4

Нефункціональні вимоги

№	Вимоги	Пояснення
1	Продуктивність	Система має швидко обробляти запити користувачів, зокрема завантаження каталогу книг, пошук і оформлення замовлень [8]. Використання асинхронних запитів (fetch) у коді та оптимізованих SQL-запитів у server.js сприяє досягненню цієї мети. Очікуваний час відгуку для основних операцій не повинен перевищувати 2-3 секунд навіть при великій кількості даних
2	Масштабованість	Програмне забезпечення має бути здатним підтримувати зростання кількості користувачів і товарів [9]. Використання бази даних SQLite (як у seed.js) є достатнім для початкового етапу, але для масштабування в майбутньому може знадобитися перехід на більш потужну СУБД, наприклад PostgreSQL
3	Зручність використання (Usability)	Інтерфейс системи має бути інтуїтивно зрозумілим як для клієнтів, так і для менеджерів [10]. У коді це враховано через модульну структуру React-компонентів (наприклад, Header, Footer, BookCard), що забезпечують чітке розмежування функціоналу та адаптивний дизайн для різних пристроїв (CSS у стилі)
4	Надійність	Система повинна гарантувати збереження даних користувачів, замовлень і каталогу навіть у разі збоїв [11]. Використання транзакцій у server.js для створення замовлень (BEGIN TRANSACTION, COMMIT, ROLLBACK) частково відповідає цій вимозі, але потрібні додаткові механізми резервного копіювання
5	Безпека	Необхідно захистити дані користувачів від несанкціонованого доступу [11]. У поточній реалізації паролі зберігаються у відкритому вигляді в базі даних, що є суттєвим недоліком. Вимогою є впровадження хешування паролів (наприклад, за допомогою bcrypt) та шифрування чутливих даних (доставка, оплата)

Технічні вимоги представлені в таблиці 1.5.

Таблиця 1.5

Технічні вимоги

№	Вимоги	Пояснення
1	Технологічний стек	Система базується на JavaScript-екосистемі: React для фронтенду [12], Node.js із Express для серверної частини [13] та SQLite як СУБД [14]. Це забезпечує кросплатформність і легкість розгортання. Код app.js демонструє використання React-хуків (useState, useEffect), a server.js — RESTful API
2	Локальне зберігання даних	Для підвищення швидкості роботи та збереження стану між сесіями система використовує localStorage для зберігання даних користувача, кошика, обраного та переглянутих книг [15]. Це видно в useEffect-хуках у app.js, які синхронізують стан із локальним сховищем
3	Інтеграція з базою даних	Серверна частина (server.js) забезпечує CRUD-операції (створення, читання, оновлення, видалення) для книг, замовлень і користувачів через SQLite [16]. Початкове заповнення бази реалізовано в seed.js, що дозволяє швидко протестувати систему з тестовими даними

Вимоги до інтерфейсу користувача показані в таблиці 1.6.

Таблиця 1.6

Вимоги до інтерфейсу користувача

№	Вимоги	Пояснення
1	Навігація	Інтерфейс має включати зручну навігацію між основними розділами (головна, кошик, обране, профіль) [17]. Компонент Header із пошуком і меню відповідає цій вимозі, дозволяючи швидко перемикатися між сторінками (setPage)
2	Відображення даних	Книги відображаються у вигляді карток (BookCard) із зображенням, назвою, автором і ціною, а деталі — у розширеному вигляді (BookDetailsPage). Замовлення та кошик представлені у вигляді таблиць із можливістю редагування

Аналіз вимог до програмної системи для продажу книжкової продукції показав, що вона має поєднувати широкий функціонал (каталог, кошик, замовлення, адміністрування) із високими нефункціональними

характеристиками (швидкість, зручність, безпека). На основі реалізованих кодів видно, що більшість функціональних вимог реалізовано, однак є простір для вдосконалення, зокрема в безпеці та масштабованості. Ці вимоги слугують основою для подальшої розробки та тестування системи, щоб вона відповідала реальним потребам користувачів у предметній області книжкової торгівлі.

1.4 Моделювання предметної області

Моделювання предметної області є ключовим етапом системного аналізу, спрямованим на формалізацію структури, процесів і взаємозв'язків у межах інформаційної системи для продажу книжкової продукції. Цей процес дозволяє створити абстрактне представлення системи, яке відображає її основні об'єкти, їхні властивості, зв'язки та поведінку. У контексті реалізованих кодів (app.js, server.js, seed.js) моделювання предметної області ґрунтується на аналізі функціональних вимог книжкового онлайн-магазину, враховуючи потреби користувачів (клієнтів і менеджерів), а також особливості управління асортиментом, замовленнями та фінансовими операціями. Основні етапи моделювання предметної області зображені на рис. 1.2.

Рис. 1.2. Основні етапи моделювання предметної області

Ідентифікація основних сутностей: предметна область книжкового онлайн-магазину включає кілька ключових сутностей, які відображені у структурі бази даних та логіці програми [18]. Першою сутністю є Книга (Books), яка характеризується такими атрибутами, як ідентифікатор (id), назва (title), автор (author), категорія (category), ціна (price), опис (description), зображення (image) та кількість на складі (inStock). Друга сутність — Користувач (Users) — охоплює клієнтів і менеджерів із атрибутами: ідентифікатор (id), електронна

пошта (email), пароль (password), роль (role) та ім’я (name). Третя сутність —Замовлення (Orders) — включає ідентифікатор (id), ідентифікатор користувача (userId), загальну суму (totalAmount), дані про доставку (shipping), платіжну інформацію (payment), статус (status) і дату (date). Пов’язана з нею сутність Елемент замовлення (Order_items) деталізує склад замовлення через атрибути: ідентифікатор (id), ідентифікатор замовлення (orderId), ідентифікатор книги (bookId), назва книги (title), ціна (price) та кількість (quantity). Нарешті, сутність Рахунок менеджера (Manager_account) відображає фінансовий компонент із атрибутами ідентифікатора (id) та балансу (balance). Ці сутності формують основу моделі даних, яка реалізується через SQLite у файлі server.js.

Визначення зв’язків між сутностями: модель предметної області передбачає чіткі зв’язки між сутностями, які забезпечують цілісність системи [19]. Сутність Користувач пов’язана із Замовленням через зовнішній ключ userId, що вказує на те, хто саме оформив замовлення. Зв’язок типу "один до багатьох" між Користувачем і Замовленнями відображає можливість одного користувача мати кілька замовлень. Сутність Замовлення пов’язана із Елементом замовлення через orderId, реалізуючи зв’язок "один до багатьох", оскільки одне замовлення може містити кілька позицій книг. У свою чергу, Елемент замовлення пов’язаний із Книгою через bookId, що дозволяє ідентифікувати конкретну книгу в замовленні (зв’язок "багато до одного"). Сутність Рахунок менеджера має опосередкований зв’язок із Замовленнями, оскільки баланс оновлюється на основі суми успішних транзакцій, що відображено в логіці server.js при створенні замовлення (POST /api/orders). Ці зв’язки формують реляційну структуру, яка забезпечує логічну організацію даних.

Моделювання процесів: окрім статичних сутностей, предметна область включає динамічні процеси, які моделюються через функціональність системи [20]. Основні процеси охоплюють: 1) Реєстрація та авторизація — реалізовані через API-ендпоінти /api/register і /api/login у server.js, де перевіряється унікальність email та відповідність пароля; 2) Перегляд і пошук книг — функція

searchBooks у app.js фільтрує книги за назвою чи автором, а fetchBooks завантажує асортимент із бази; 3) Формування кошика — методи addToCart, removeFromCart і updateCartItemQuantity дозволяють користувачу керувати списком покупок із збереженням у localStorage; 4) Оформлення замовлення — функція createOrder у app.js надсилає дані на сервер, де вони записуються в таблиці orders і order_items; 5) Управління асортиментом і замовленнями — методи addBook, updateBook, deleteBook і updateOrderStatus доступні менеджерам для адміністрування. Ці процеси моделюються як послідовність дій, що залежать від ролі користувача (customer або manager), і відображені у React-компонентах, таких як HomePage, CartPage чи ManageBooksPage.

Для формального представлення предметної області можна використати діаграму "сутність-зв’язок" (ERD), яка візуалізує описані сутності та їхні відношення. Наприклад, таблиця "users" пов’язана з "orders" через userId, а "orders" — із "order_items" через orderId, що утворює ієрархічну структуру. Додатково модель поведінки системи може бути представлена через діаграму станів для сутності Замовлення, де стани (Новий → Оброблений → Відправлений → Доставлений або Скасований) змінюються залежно від дій менеджера (updateOrderStatus). У коді це реалізовано через логіку оновлення статусу в API /api/orders/:id. Локальне зберігання даних (cart, favorites, viewedItems) у localStorage моделює тимчасові стани користувацької сесії, що підвищує зручність взаємодії із системою.

Таким чином, моделювання предметної області для інформаційної системи продажу книжкової продукції охоплює ідентифікацію сутностей (книги, користувачі, замовлення тощо), визначення їхніх зв’язків і атрибутів, а також формалізацію ключових процесів. На основі цього створено реляційну модель даних і функціональну логіку, які відображені у реалізованому коді. Цей підхід забезпечує цілісність, масштабованість і адаптивність системи до потреб користувачів, що є основою для її успішної реалізації.

1.5 Огляд існуючих рішень

Для оцінки конкурентного середовища розглянуто три популярні аналоги, які пропонують функціонал для продажу книжкової продукції. Аналіз дозволяє порівняти їх із розробленою системою та визначити її унікальні особливості.

Amazon (рис. 1.3) є світовим лідером у сфері електронної комерції, зокрема у продажу книг [21]. Система пропонує розширений пошук, персоналізовані рекомендації на основі історії переглядів, а також інтеграцію з логістичними сервісами. Сильні сторони: широкий асортимент, відгуки користувачів, підтримка кількох мов. Слабкі сторони: складність інтерфейсу для нових користувачів і перевантаженість функціями. У порівнянні з Amazon, розроблена система має простіший інтерфейс і локалізована для українського ринку, що є її перевагою.

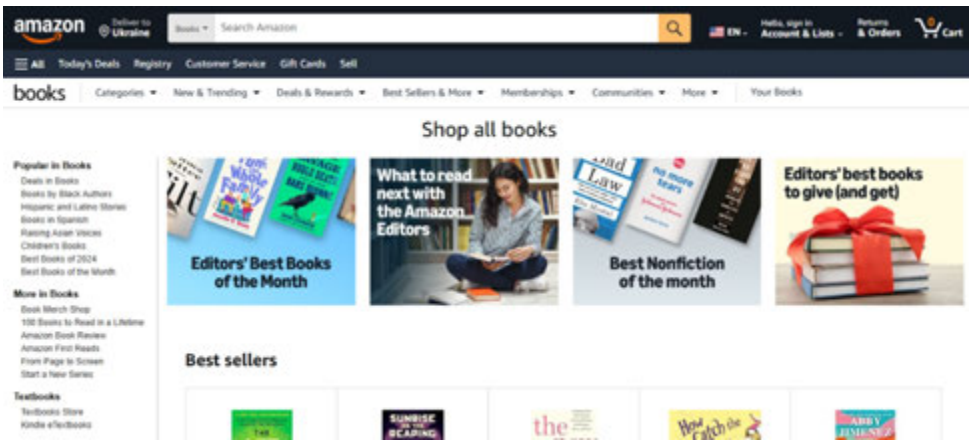

Рис. 1.3. Сторінка книжок на Amazon [21]

Yakaboo (рис. 1.4) – український онлайн-магазин книг із акцентом на локальний контент [22]. Платформа підтримує пошук за категоріями, фільтри за ціною та автором, а також функцію кошика й оформлення замовлень. Сильні сторони: зручна навігація, підтримка україномовного контенту. Слабкі сторони: обмежена інтеграція з менеджерськими функціями та відсутність персоналізації для переглянутих товарів. Розроблена система перевершує Yakaboo завдяки

функціям управління книгами та замовленнями (manageBooks, manageOrders), а також відстеженню переглянутих книг (viewedItems).

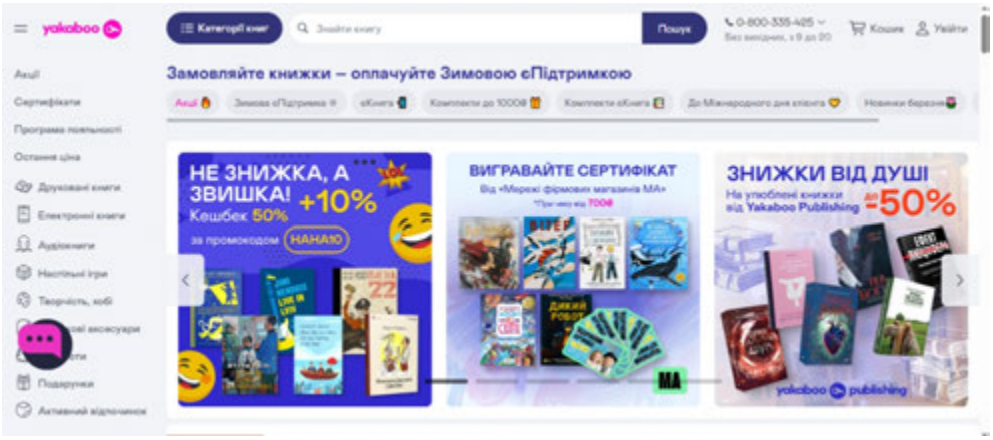

Рис. 1.4. Головна сторінка Yakaboo [22]

Сервіс AbeBooks (рис. 1.5) є глобальним онлайн-ринком для продажу книг, з акцентом на вживані, рідкісні та колекційні видання [23]. Він пропонує широкий вибір літератури від незалежних продавців із понад 50 країн, простий пошук за категоріями та базові функції кошика. Сильні сторони: величезний асортимент унікальних книг, можливість знайти видання, яких немає в інших магазинах. Слабкі сторони: відсутність локалізації для українського ринку (інтерфейс англійською, валюта в доларах) та обмежені інструменти для менеджерів. Розроблена система перевершує завдяки адаптації до українського користувача (інтерфейс українською, ціни в гривнях) і наявності функцій адміністрування.

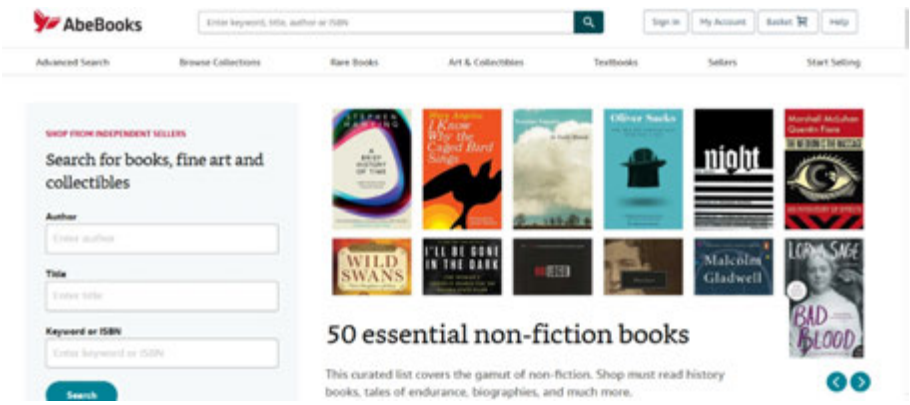

Рис. 1.5. Головна сторінка AbeBooks [23]

Аналіз інформаційних джерел показав, що сучасні інформаційні системи для продажу книг повинні поєднувати зручність інтерфейсу, ефективну обробку даних і гнучкість архітектури. Розроблена система враховує ці принципи, використовуючи React для інтерактивного фронтенду, SQLite для зберігання даних і Express для серверної логіки. Порівняння з аналогами виявило, що система має конкурентні переваги, такі як локалізація, менеджерські функції та простота використання, хоча поступається глобальним гравцям за масштабом і глибиною персоналізації. Подальший розвиток може включати інтеграцію відгуків користувачів і розширення рекомендаційних алгоритмів, що підвищить її привабливість на ринку.

1.6 Постановка завдання

Сучасний розвиток інформаційних технологій відкриває нові можливості для оптимізації бізнес-процесів у сфері торгівлі, зокрема у продажу книжкової продукції. Інформаційна система для продажу книг онлайн є важливим інструментом, що дозволяє не лише автоматизувати процеси купівлі-продажу, але й забезпечити зручність для користувачів, ефективність управління асортиментом і замовленнями, а також підвищити конкурентоспроможність підприємства на ринку. У контексті предметної області — книжкової торгівлі — виникає потреба у створенні програмного забезпечення, яке б враховувало специфіку цього сегменту, включаючи різноманітність літератури, особливості цільової аудиторії та вимоги до логістики й оплати. Метою даного розділу є детальна постановка завдання для розробки інформаційної системи, яка відповідає потребам як покупців, так і менеджерів книжкового магазину.

Предметна область охоплює діяльність онлайн-магазину книжкової продукції, що передбачає продаж книг різних жанрів, категорій і форматів. Ключовими учасниками процесу є покупці (кінцеві користувачі), менеджери (адміністратори системи) та система управління базою даних, яка забезпечує зберігання й обробку інформації про книги, користувачів, замовлення та фінансові операції. Специфіка книжкової торгівлі полягає у необхідності врахування таких факторів, як: широкий асортимент літератури (класика, сучасна проза, дитяча література, наукові видання тощо), наявність унікальних характеристик книг (автор, видавництво, рік видання, обкладинка), а також потреба у гнучкому управлінні запасами та обробці замовлень. Крім того, важливим є забезпечення зручного пошуку книг за різними критеріями (назва, автор, категорія), інтеграція з платіжними системами та надання користувачам персоналізованих функцій, таких як обране чи історія переглядів.

Основною метою створення інформаційної системи є розробка програмного забезпечення, яке забезпечить автоматизацію процесів продажу

книжкової продукції через онлайн-платформу. Система має надавати користувачам зручний інтерфейс для перегляду асортименту, вибору книг, оформлення замовлень і здійснення оплат, а менеджерам — інструменти для адміністрування каталогу, обробки замовлень і моніторингу фінансових показників. Таким чином, завдання полягає у створенні комплексного рішення, яке поєднує функціональність електронної комерції з елементами управління внутрішніми процесами книжкового магазину.

Для досягнення поставленої мети необхідно вирішити ряд конкретних завдань, які можна поділити на функціональні, технічні та організаційні напрямки (таблиця 1.7).

Таблиця 1.7

Основні завдяння проєкту

№	Завдання	Опис
1	2	3
1	Функціональні завдання	
1.1	Розробка модуля каталогу книг	Система повинна забезпечувати відображення повного переліку книг із детальною інформацією (назва, автор, категорія, ціна, опис, зображення, наявність на складі). Передбачається можливість фільтрації за категоріями та пошуку за ключовими словами
1.2	Реалізація кошика покупок	Користувачі матимуть змогу додавати книги до кошика, змінювати їх кількість або видаляти товари перед оформленням замовлення. Кошик має зберігатися локально для зручності повторного доступу
1.3	Оформлення замовлень	Необхідно створити модуль для введення даних доставки (ПІБ, адреса, телефон) та оплати (номер картки, термін дії, CVV), а також обробки замовлення з автоматичним оновленням статусу (наприклад, «Новий», «Оброблений», «Відправлений»)
1.4	Персоналізація для користувачів	Система має підтримувати функції авторизації та реєстрації, збереження обраних книг, історії переглянутих товарів і персональних даних користувача
1.5	Адміністрування для менеджерів	Передбачається створення окремого інтерфейсу для менеджерів із можливістю додавання, редагування та видалення книг, управління замовленнями (зміна статусу) та перегляду фінансового балансу магазину

Продовження таблиці 1.7

1	2	3
1.6	Система сповіщень	Користувачі та менеджери отримуватимуть повідомлення про успішні дії (додавання до кошика, оформлення замовлення) чи помилки (невдала авторизація, відсутність товару)
2	Технічні завдання	
2.1	Вибір технологічного стеку	Для реалізації системи обрано React для фронтенду (інтерфейсу користувача), Node.js із Express для бекенду (серверної логіки) та SQLite як базу даних для зберігання інформації. Ці технології забезпечують швидкість розробки, масштабованість і простоту інтеграції
2.2.	Забезпечення асинхронної взаємодії	API-запити (GET, POST, PUT, DELETE) мають бути реалізовані для обміну даними між клієнтською та серверною частинами, наприклад, для отримання списку книг чи оновлення статусу замовлення
2.3	Локальне збереження даних	Використання LocalStorage для зберігання тимчасових даних користувача (кошик, обране) з автоматичним синхронізацією після авторизації
2.4	Безпека	**.** Реалізація базового захисту даних (перевірка введених даних, уникнення SQL-ін’єкцій) та автентифікація користувачів через email і пароль
3	Організаційні завдання	
3.1	Тестування та відлагодження	Проведення тестування функціональності системи (перевірка коректності роботи кошика, оформлення замовлень, адмін-панелі) та виправлення виявлених помилок
3.2	Документація	Підготовка технічної документації, яка описує структуру системи, принципи роботи API та інструкції для користувачів і адміністраторів
3.3	Розгортання	Забезпечення запуску системи на локальному сервері з можливістю подальшого перенесення на хмарний хостинг

У результаті реалізації поставлених завдань має бути створена інформаційна система "Книжковий Онлайн", яка включає веб-додаток із сучасним інтерфейсом, серверну частину для обробки запитів і базу даних для зберігання інформації. Система забезпечить зручний доступ до асортименту книг, швидке оформлення замовлень і ефективне управління магазином.

Очікується, що програмне забезпечення підвищить рівень автоматизації бізнес-процесів, зменшить час на обробку замовлень і покращить користувацький досвід завдяки інтуїтивно зрозумілому дизайну та персоналізованим функціям.

Розробка системи передбачає певні обмеження: підтримка лише однієї мови інтерфейсу (української), базова модель оплати (без інтеграції з реальними платіжними шлюзами на етапі прототипу), а також локальне розгортання без повноцінної хмарної інфраструктури. Припускається, що користувачі мають базові навички роботи з веб-додатками, а менеджери — мінімальні знання адміністрування.

Таким чином, постановка завдання охоплює комплексний підхід до створення інформаційної системи, яка враховує потреби всіх сторін — від покупців до адміністраторів. Реалізація проекту, представлена у реалізованому коді, є практичним втіленням цих вимог, що дозволяє перейти до наступних етапів аналізу та вдосконалення системи.

2. ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних у вигляді ER-діаграми

Проєктування інформаційного забезпечення інформаційної системи для продажу книжкової продукції є ключовим етапом розробки, що визначає структуру даних та їх взаємозв'язки. Логічна модель даних, представлена у вигляді ER-діаграми (Entity-Relationship Diagram), відображає сутності, їх атрибути та зв'язки між ними, що забезпечують функціонування системи [25]. На основі реалізованих кодів (app.js, server.js, seed.js) можна детально описати логічну модель даних, яка лежить в основі цієї системи.

Сутності та їх атрибути показані в таблиці 2.1.

Таблиця 2.1

Сутності та їх атрибути

№	Сутності	Атрибути	Опис
1	2	3	4
1	Users (Користувачі) - сутність представляє користувачів системи, які можуть мати різні ролі: покупці (customer) або менеджери (manager	id (INTEGER, первинний ключ, автоінкремент) – унікальний ідентифікатор користувача	Ця сутність є основою для автентифікації та авторизації, а також пов'язує користувачів із замовленнями
		email (TEXT, унікальний) – електронна пошта користувача, що використовується для авторизації	
		password (TEXT) – пароль для входу в систему	
		role (TEXT) – роль користувача (customer або manager)	
		name (TEXT) – ім'я користувача	

Продовження таблиці 2.1

1	2	3	4
2	Books (Книги) - сутність відображає книжкову продукцію, доступну для продажу	id (INTEGER, первинний ключ, автоінкремент) – унікальний ідентифікатор книги	Сутність є центральною для каталогу товарів і використовується для відображення асортименту, пошуку та оформлення замовлень
		title (TEXT) – назва книги	
		author (TEXT) – автор книги	
		category (TEXT) – категорія книги (наприклад, "Українська класика", "Фантастика")	
		price (REAL) – ціна книги в гривнях	
		description (TEXT) – опис книги	
		image (TEXT) – URL зображення книги	
		inStock (INTEGER) – кількість книг у наявності	
3	Orders (Замовлення) – сутність представляє замовлення, створені користувачами	id (INTEGER, первинний ключ, автоінкремент) – унікальний ідентифікатор замовлення	Сутність фіксує інформацію про замовлення та пов’язує користувачів із придбаними книгами
		userId (INTEGER, зовнішній ключ до Users) – ідентифікатор користувача, який зробив замовлення	
		totalAmount (REAL) – загальна сума замовлення	
		shipping (TEXT) – дані про доставку (у форматі JSON: ім’я, адреса, місто тощо)	
		payment (TEXT) – дані про оплату (у форматі JSON: метод, останні цифри картки)	
		status (TEXT) – статус замовлення (наприклад, "Новий", "Доставлений")	
		date (TEXT) – дата створення замовлення у форматі ISO	

Завершення таблиці 2.1

1	2	3	4
4	Order_Items (Елементи замовлення) – сутність деталізує складові кожного замовлення, тобто конкретні книги та їх кількість	id (INTEGER, первинний ключ, автоінкремент) – унікальний ідентифікатор елемента	Сутність забезпечує зв’язок "багато до багатьох" між Orders і Books, дозволяючи зберігати деталі замовлення
		orderId (INTEGER, зовнішній ключ до Orders) – ідентифікатор замовлення	
		bookId (INTEGER, зовнішній ключ до Books) – ідентифікатор книги	
		title (TEXT) – назва книги (дублюється для зручності)	
		price (REAL) – ціна книги на момент замовлення	
		quantity (INTEGER) – кількість одиниць книги у замовленні	
5	Manager_Account (Рахунок менеджера) – сутність відображає фінансовий баланс менеджерів, який оновлюється при обробці замовлень	id (INTEGER, первинний ключ, автоінкремент) – унікальний ідентифікатор запису	Використовується для відстеження доходів від продажів, хоча в системі передбачено лише один запис (id=1)
		balance (REAL) – поточний баланс менеджера	

Зв’язки між сутностями описані в таблиці 2.2

Таблиця 2.2

Зв’язки між сутностями

№	Назва	Опис
1	Users – Orders (Один до багатьох)	Один користувач може мати багато замовлень, але кожне замовлення належить лише одному користувачу
		Зв’язок реалізовано через зовнішній ключ userId у таблиці Orders, що посилається на id у таблиці Users
2	Orders – Order_Items (Один до багатьох)	Одне замовлення може містити багато елементів (книг), але кожен елемент належить лише одному замовленню
		Зв’язок реалізовано через зовнішній ключ orderId у таблиці Order_Items, що посилається на id у таблиці Orders

Продовження таблиці 2.2

1	2	3
3	Books – Order_Items (Один до багатьох)	Одна книга може бути частиною багатьох елементів замовлень, але кожен елемент замовлення посилається лише на одну книгу
		Зв’язок реалізовано через зовнішній ключ bookId у таблиці Order_Items, що посилається на id у таблиці Books
4	Orders – Manager_Account (Багато до одного, опосередковано)	Кожне замовлення впливає на баланс менеджера, але сутність Manager_Account не має прямого зв’язку з Orders у вигляді зовнішнього ключа. Логіка оновлення балансу реалізована на рівні серверного коду (server.js)

Особливості моделі показані в таблиці 2.3.

Таблиця 2.3

Особливості моделі

№	Назва	Опис
1	Нормалізація	Модель відповідає принципам нормалізації (до 3NF), уникаючи надлишковості даних. Наприклад, інформація про книги дублюється в Order_Items (title, price) для збереження історичних даних на момент замовлення
2	Гнучкість	Використання JSON у полях shipping і payment дозволяє зберігати структуровані дані без створення додаткових таблиць, що спрощує модель, але може ускладнити пошук за цими полями
3	Розширюваність	Додавання нових сутностей (наприклад, відгуків чи промокодів) можливе без значних змін у структурі

Візуалізація ER-діаграми подана на рис. 2.1.

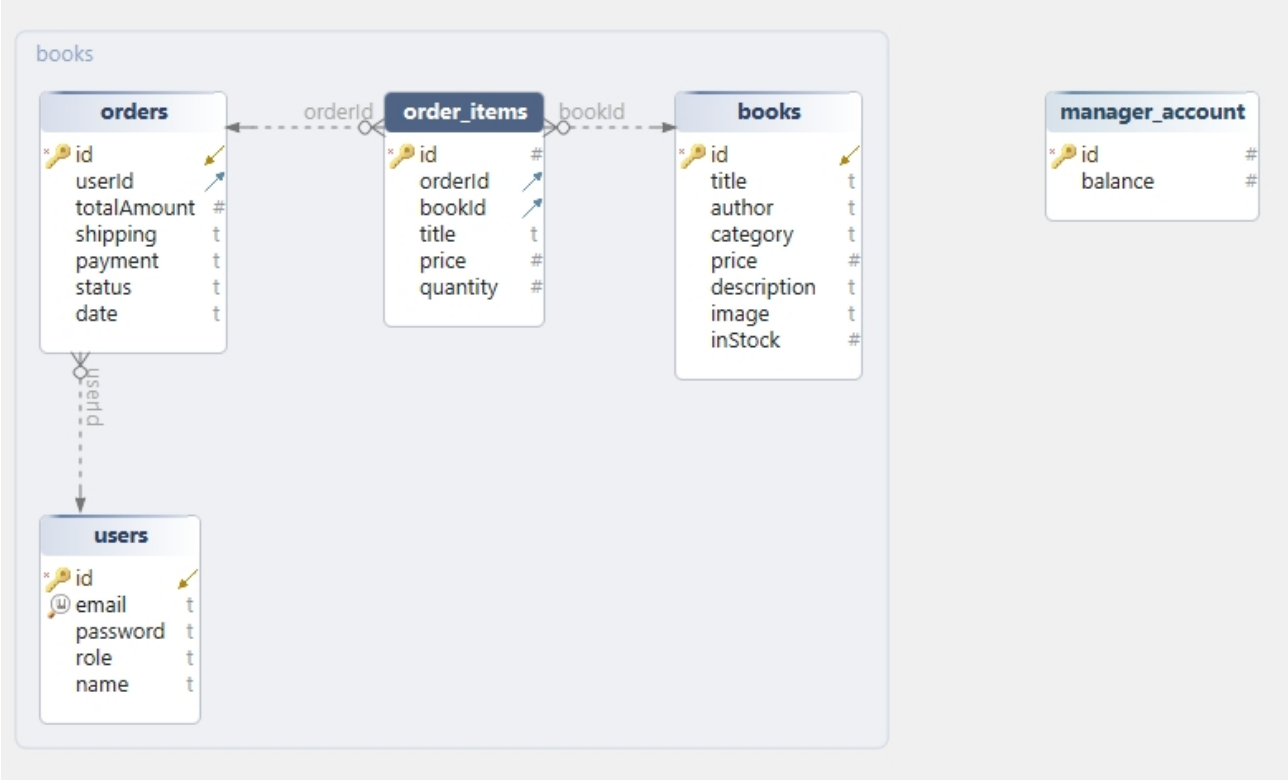

Рис. 2.1. ER-діаграма

Таким чином, логічна модель даних забезпечує ефективне зберігання та обробку інформації про користувачів, книги, замовлення та фінансовий стан, відповідаючи функціональним вимогам системи продажу книжкової продукції.

2.2 Діаграма класів та кооперацій

Проєктування інформаційного та програмного забезпечення інформаційної системи для продажу книжкової продукції є ключовим етапом розробки, що визначає структуру, взаємодію компонентів та логіку функціонування системи. У контексті реалізованих кодів (app.js, server.js, seed.js) основна увага приділяється створенню об’єктно-орієнтованої моделі, яка відображає сутності системи, їхні атрибути, методи та взаємозв’язки. Діаграма класів та кооперацій слугує інструментом для формалізації цих даних, забезпечуючи чітке розуміння архітектури системи як для розробників, так і для інших зацікавлених сторін.

Діаграма класів у даному проєкті базується на аналізі структури React-компонентів (app.js), серверної логіки (server.js) та моделі даних (seed.js) [26]. Основні класи відображають сутності системи, такі як користувачі, книги, замовлення, кошик, обране тощо, а також їхні взаємозв’язки. Ключові класи детально описані в таблиці 2.4.

Таблиця 2.4

Ключові класи

№	Класи та складники	Опис
1	2	3
1	Клас User (Користувач)	
1.1	Атрибути	id (унікальний ідентифікатор), email (електронна пошта), password (пароль), role (роль: customer або manager), name (ім’я)
1.2	Методи	відсутні прямі методи в коді, але логіка авторизації (login) та реєстрації (register) реалізована через API-запити до серверної частини
1.3	Опис	Представляє користувача системи, який може бути покупцем або менеджером. Роль визначає доступ до функціоналу (наприклад, менеджер має доступ до керування книгами та замовленнями)
2	Клас Book (Книга)	
2.1	Атрибути	id (унікальний ідентифікатор), title (назва), author (автор), category (категорія), price (ціна), description (опис), image (посилання на зображення), inStock (кількість на складі)
2.2	Методи	методи додавання (addBook), оновлення (updateBook) та видалення (deleteBook) реалізовані на сервері через API
2.3	Опис	Моделює книжкову продукцію, доступну для продажу. Клас пов’язаний із кошиком, обраним та замовленнями
3	Клас Order (Замовлення)	
3.1	Атрибути	id (унікальний ідентифікатор), userId (посилання на користувача), totalAmount (загальна сума), shipping (дані доставки), payment (дані оплати), status (статус: Новий, Оброблений тощо), date (дата створення), items (список товарів)
3.2	Методи	створення (createOrder), оновлення статусу (updateOrderStatus)

Продовження таблиці 2.4

1	2	3
3.3	Опис	Відображає замовлення, яке формується з товарів у кошику. Пов’язаний із класами User та OrderItem
4	Клас OrderItem (Елемент замовлення)	
4.1	Атрибути	id (унікальний ідентифікатор), orderId (посилання на замовлення), bookId (посилання на книгу), title (назва книги), price (ціна), quantity (кількість)
4.2	Опис	Деталізує складові замовлення, пов’язуючи його з конкретними книгами
5	Клас Cart (Кошик)	
5.1	Атрибути	масив об’єктів із полями id, title, price, quantity тощо (динамічна структура)
5.2	Методи	addToCart (додавання книги), removeFromCart (видалення), updateCartItemQuantity (оновлення кількості)
5.3	Опис	Тимчасово зберігає товари, які користувач планує придбати. Дані синхронізуються з localStorage
6	Клас Favorites (Обране)	
6.1	Атрибути	масив об’єктів типу Book
6.2	Методи	toggleFavorite (додавання/видалення з обраного)
6.3	Опис	Зберігає книги, які користувач позначив як улюблені
7	Клас ManagerAccount (Рахунок менеджера)	
7.1	Атрибути	id (унікальний ідентифікатор), balance (баланс)
7.2	Методи	оновлення балансу через транзакції в createOrder
7.3	Опис	Відстежує фінансовий стан менеджера, який залежить від виконаних замовлень

Взаємозв’язки між класами показані в таблиці 2.5.

Таблиця 2.5

Взаємозв’язки між класами

№	Назва	Опис
1	Асоціація	User має зв’язок "один до багатьох" із Order (користувач може мати кілька замовлень)
2	Композиція	Order включає OrderItem (замовлення складається з елементів)
3	Асоціація	Book пов’язаний із Cart, Favorites та OrderItem через ідентифікатор книги
4	Узагальнення	ManagerAccount асоціюється з роллю менеджера в User

Діаграма класів зображена на рис. 2.2.

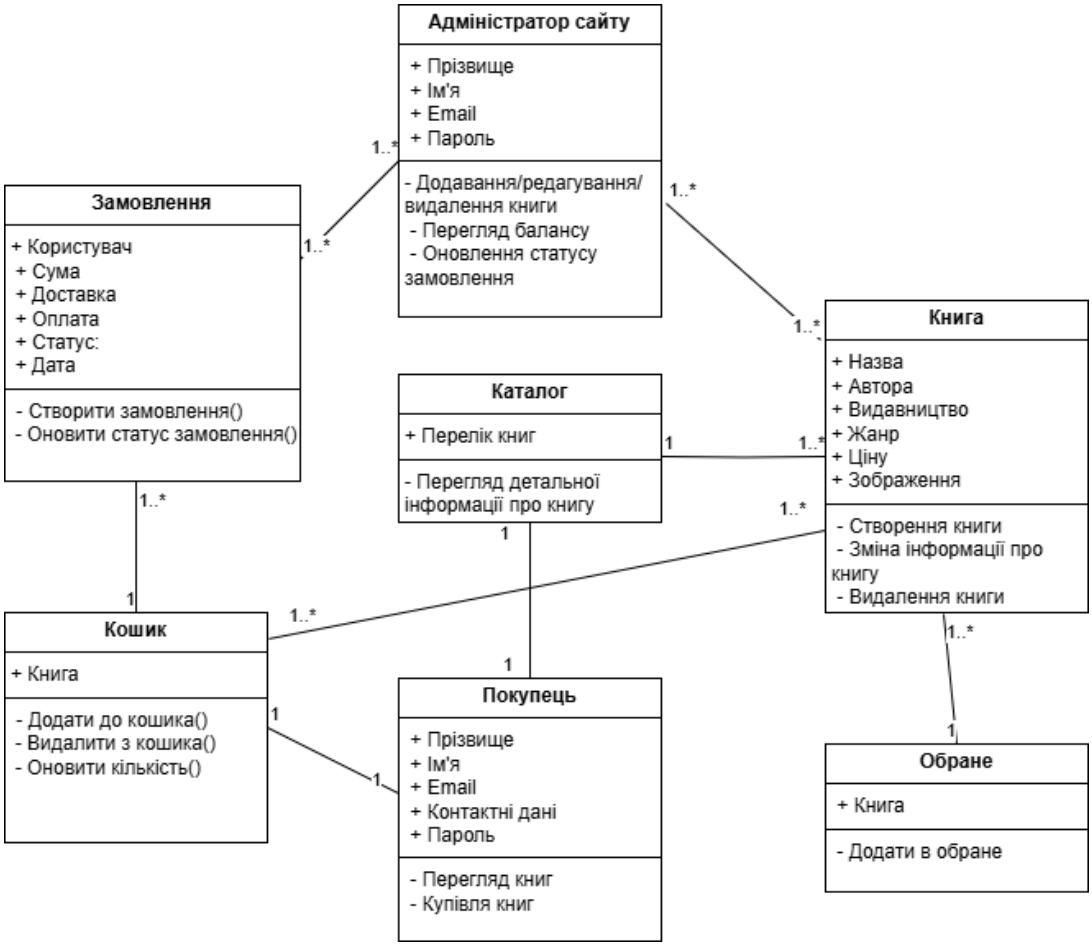

Рис. 2.2. Діаграма класів

Діаграма кооперацій описує взаємодію між об’єктами класів під час виконання ключових сценаріїв використання, таких як авторизація, додавання книги до кошика, оформлення замовлення. Наприклад:

1) Сценарій "Оформлення замовлення":
   - Користувач (User) через компонент CheckoutPage викликає метод createOrder.
   - Cart передає список товарів у Order.
   - Order взаємодіє з сервером через API-запит, створюючи запис у базі даних.

– ManagerAccount оновлює баланс після успішного створення замовлення.
– Повідомлення (showNotification) відображається користувачу.

2) Сценарій "Керування книгами":
– Менеджер (User із роллю manager) через ManageBooksPage викликає методи addBook, updateBook або deleteBook.
– Сервер оновлює таблицю books, а клієнтська частина оновлює стан books.

Таким чином, діаграма кооперацій (рис. 2.3) ілюструє динамічну взаємодію між об’єктами, підкреслюючи асинхронний характер запитів до сервера та реактивність інтерфейсу.

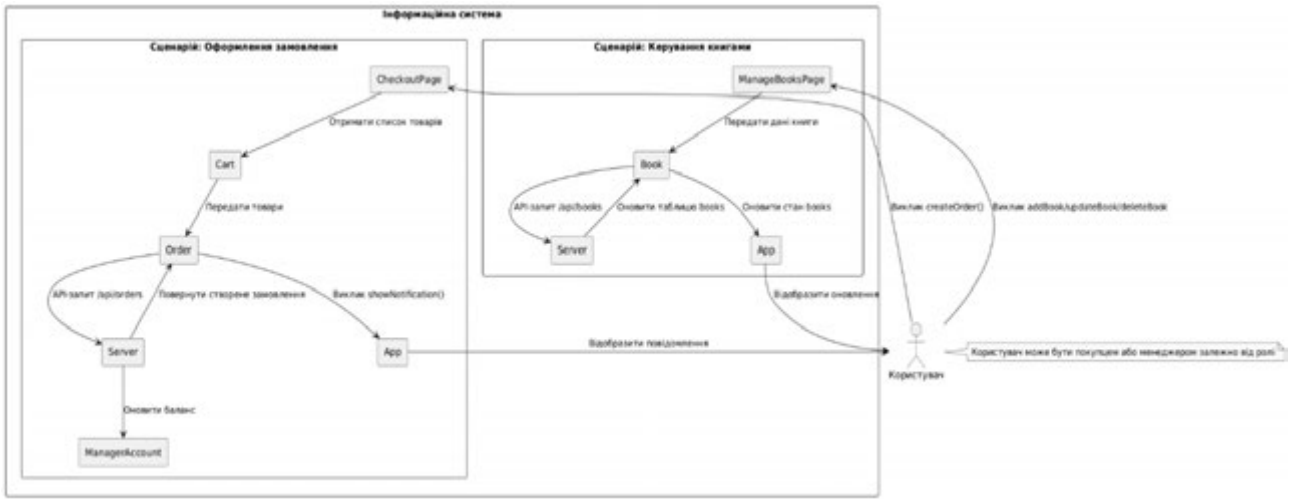

Рис. 2.3. Діаграма кооперацій

Діаграма класів та кооперацій у проєкті інформаційної системи для продажу книжкової продукції забезпечує чітку структуризацію сутностей та їхньої взаємодії. Вона відображає як статичну архітектуру (класи, атрибути, зв’язки), так і динамічну поведінку (сценарії використання). На основі реалізованих кодів видно, що система побудована з урахуванням принципів модульності, повторного використання коду та розподілу обов’язків між клієнтською та серверною частинами.

2.3 Діаграма пакетів

Проєктування інформаційного та програмного забезпечення інформаційної системи для продажу книжкової продукції є ключовим етапом розробки, що визначає архітектурну структуру системи та взаємодію її компонентів. Одним із важливих інструментів на цьому етапі є діаграма пакетів, яка відображає організацію коду у вигляді модулів (пакетів) та їхні залежності. У контексті реалізованих кодів (app.js, server.js, seed.js) діаграма пакетів дозволяє систематизувати структуру проєкту, враховуючи як клієнтську, так і серверну частини, а також допоміжні компоненти для роботи з базою даних.

На основі реалізованих кодів можна виділити три основні рівні системи: клієнтський інтерфейс (frontend), серверна логіка (backend) та рівень даних (data layer). Клієнтська частина, реалізована в app.js, базується на бібліотеці React і включає компоненти для відображення сторінок, обробки користувацьких дій та взаємодії з сервером через API. Серверна частина, описана в server.js, побудована на фреймворку Express.js і забезпечує обробку запитів, взаємодію з базою даних SQLite та надання даних клієнту. Файл seed.js виконує функцію ініціалізації бази даних, заповнюючи її тестовими даними, що є частиною підготовчого етапу розробки.

Діаграма пакетів для даної системи може бути структурована за описом в таблиці 2.6.

Таблиця 2.6

Структура діаграми пакетів

№	Назва	Опис	Підпакети	Залежності
1	Пакет "Client" (Клієнтська частина)	Містить код клієнтського інтерфейсу, реалізований у файлі app.js. Цей пакет включає компоненти React, які відповідають за відображення сторінок (наприклад, HomePage, BookDetailsPage, CartPage), а також логіку управління станом (state management) через хуки useState та useEffect	Components: Логічно об'єднує окремі компоненти інтерфейсу, такі як Header, Footer, BookCard. Ці компоненти є перевикористовуваними модулями, що формують структуру сторінок Pages: Включає компоненти, які представляють окремі сторінки системи (наприклад, LoginPage, CheckoutPage, ManageBooksPage). Кожен компонент сторінки інкапсулює власну логіку та відображення API: Містить функції для взаємодії з сервером (наприклад, fetchBooks, login, createOrder), які використовують асинхронні запити через fetch	Залежить від зовнішньої бібліотеки React та серверного пакета "Server" для отримання даних через API
2	Пакет "Server" (Серверна частина)	Опис: Реалізований у файлі server.js, цей пакет відповідає за серверну логіку, обробку HTTP-запитів та взаємодію з базою даних. Використовує Express.js як основний фреймворк	Routes: Включає маршрути API (наприклад, /api/books, /api/orders, /api/login), які обробляють запити від клієнта та повертають відповіді у форматі JSON Database: Містить логіку роботи з базою даних SQLite через бібліотеку better-sqlite3. Сюди входять запити для створення таблиць (initDB), а також операції CRUD (Create, Read, Update, Delete) для сутностей, таких як книги, замовлення та користувачі	Залежності: Залежить від пакета "Data" для доступу до бази даних та зовнішніх бібліотек (express, body-parser, better-sqlite3)

Продовження таблиці 2.6

1	2	3	4	5
3	Пакет "Data" (Рівень даних)	Представлений у файлі seed.js, цей пакет відповідає за структуру даних та їх початкове заповнення. Він включає логіку створення таблиць (users, books, orders, order_items, manager_account) та генерацію тестових даних	Schema: Визначає схему бази даних, включаючи таблиці та їх зв’язки (наприклад, зовнішні ключі між orders та users)	Залежить від бібліотеки better-sqlite3 та використовується пакетом "Server" для ініціалізації даних
			Seeder**:** Містить функції для заповнення бази даних тестовими записами (користувачі, книги, замовлення)	

Пакет "Client" залежить від "Server" через API-запити, які забезпечують обмін даними (наприклад, отримання списку книг або створення замовлення). Пакет "Server", у свою чергу, залежить від "Data" для доступу до бази даних та її ініціалізації. Зворотних залежностей між пакетами немає, що свідчить про чіткий односпрямований потік даних: від клієнта до сервера, а від сервера до бази даних.

Діаграма пакетів зображена на рис. 2.4.

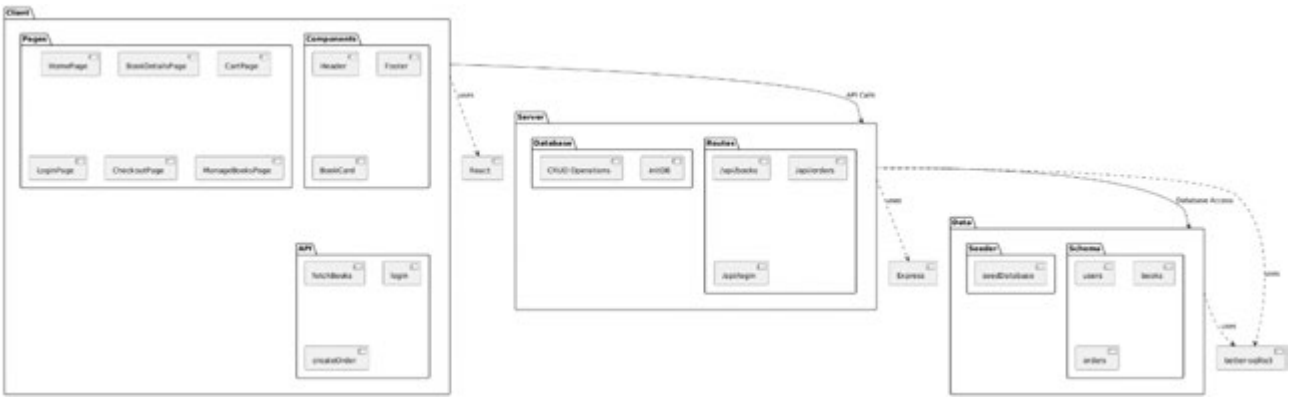

Рис. 2.4. Діаграма пакетів

Така організація пакетів забезпечує модульність системи, полегшує її масштабування та підтримку. Наприклад, додавання нових сторінок чи API-маршрутів може бути реалізовано шляхом розширення відповідних підпакетів без значних змін у решті системи. Використання React на клієнтській стороні та Express.js на серверній дозволяє ефективно розділити відповідальність між інтерфейсом і бізнес-логікою.

Діаграма пакетів для інформаційної системи продажу книжкової продукції відображає три основні пакети: "Client", "Server" і "Data", кожен із яких має власні підпакети та чітко визначені функції. Ця структура відповідає сучасним підходам до розробки веб-додатків, забезпечуючи гнучкість, читабельність коду та легкість інтеграції нових функцій.

2.4 Діаграма компонентів

Проєктування інформаційного та програмного забезпечення інформаційної системи для продажу книжкової продукції передбачає створення структурованої архітектури, яка забезпечує ефективну взаємодію між різними компонентами системи. Діаграма компонентів є ключовим інструментом у цьому процесі, оскільки вона відображає основні модулі системи, їхні функції та зв’язки між ними. На основі реалізованих кодів (app.js, server.js, seed.js) можна детально описати компонентну структуру системи, враховуючи її клієнт-серверну архітектуру, використання бази даних та інтерактивний інтерфейс користувача.

Система складається з трьох основних рівнів: клієнтського (frontend), серверного (backend) та рівня даних (data layer). Кожен із цих рівнів включає окремі компоненти, які виконують спеціалізовані функції, такі як відображення інтерфейсу, обробка запитів, управління даними тощо. Детальний опис компонентів системи та їхньої взаємодії подано в таблиці 2.7.

Таблиця 2.7

Опис компонентів системи

№	Рівень	Компоненти та їх опис
1	2	3
1	Клієнтський рівень (Frontend) реалізовано в app.js за допомогою бібліотеки React, яка забезпечує створення динамічного вебінтерфейсу	Web UI (Вебінтерфейс користувача): Відповідає за відображення сторінок, таких як головна сторінка (HomePage), сторінка деталей книги (BookDetailsPage), кошик (CartPage), оформлення замовлення (CheckoutPage) тощо. Цей компонент обробляє введення даних користувачем (наприклад, пошук книг, додавання до кошика, введення даних для оплати) та передає запити до серверного рівня через API. Він також відображає сповіщення (notification) про успішність операцій
		State Management (Управління станом): Використовує React hooks (useState, useEffect) для локального зберігання даних, таких як список книг (books), кошик (cart), обране (favorites), переглянуті товари (viewedItems) тощо. Дані синхронізуються з localStorage для збереження стану між сеансами. Цей компонент забезпечує логіку клієнтської сторони, наприклад, фільтрацію книг за категоріями чи оновлення кількості товарів у кошику
2	Серверна частина (Backend), реалізована в server.js на базі Express.js, виконує роль посередника між клієнтом і базою даних	API Controller (Контролер API): Обробляє HTTP-запити від клієнта (GET, POST, PUT, DELETE) через ендпоінти, такі як /api/books, /api/orders, /api/login, /api/register. Цей компонент відповідає за автентифікацію користувачів, управління книгами (додавання, оновлення, видалення), створення та оновлення замовлень. Він також повертає відповіді у форматі JSON із результатами операцій
		Business Logic (Бізнес-логіка): Виконує обробку даних, наприклад, розрахунок загальної суми замовлення (totalAmount), оновлення балансу менеджера при створенні замовлення, перевірку наявності книг у базі даних перед їхньою обробкою. Цей компонент інтегрує логіку транзакцій для забезпечення цілісності даних (наприклад, BEGIN TRANSACTION і COMMIT у SQLite)

Продовження таблиці 2.7

1	2	3
3	Рівень даних (Data Layer) базується на SQLite (bookstore.db), що ініціалізується в server.js та заповнюється початковими даними через seed.js.	SQLite DB (База даних SQLite): Зберігає всі дані системи, включаючи таблиці users (користувачі), books (книги), orders (замовлення), order_items (елементи замовлення) та manager_account (баланс менеджера). База даних підтримує зв’язки між таблицями через зовнішні ключі (наприклад, userId у orders посилається на users). Компонент забезпечує CRUD-операції (створення, читання, оновлення, видалення) через SQL-запити, які виконуються серверним рівнем

Взаємодія компонентів:

– Користувач (User) взаємодіє з Web UI, наприклад, реєструється чи переглядає книги. Web UI надсилає запити до API Controller (наприклад, POST /api/register або GET /api/books).

– API Controller обробляє запит, звертаючись до Business Logic для виконання обчислень чи валідації, а потім до SQLite DB для збереження чи отримання даних.

– Після обробки API Controller повертає відповідь до Web UI, яка оновлює інтерфейс через State Management.

– Дані, такі як кошик чи обране, тимчасово зберігаються в State Management і синхронізуються з localStorage, а при оформленні замовлення передаються на сервер для збереження в SQLite DB.

Діаграма компонентів показана на рис. 2.5.

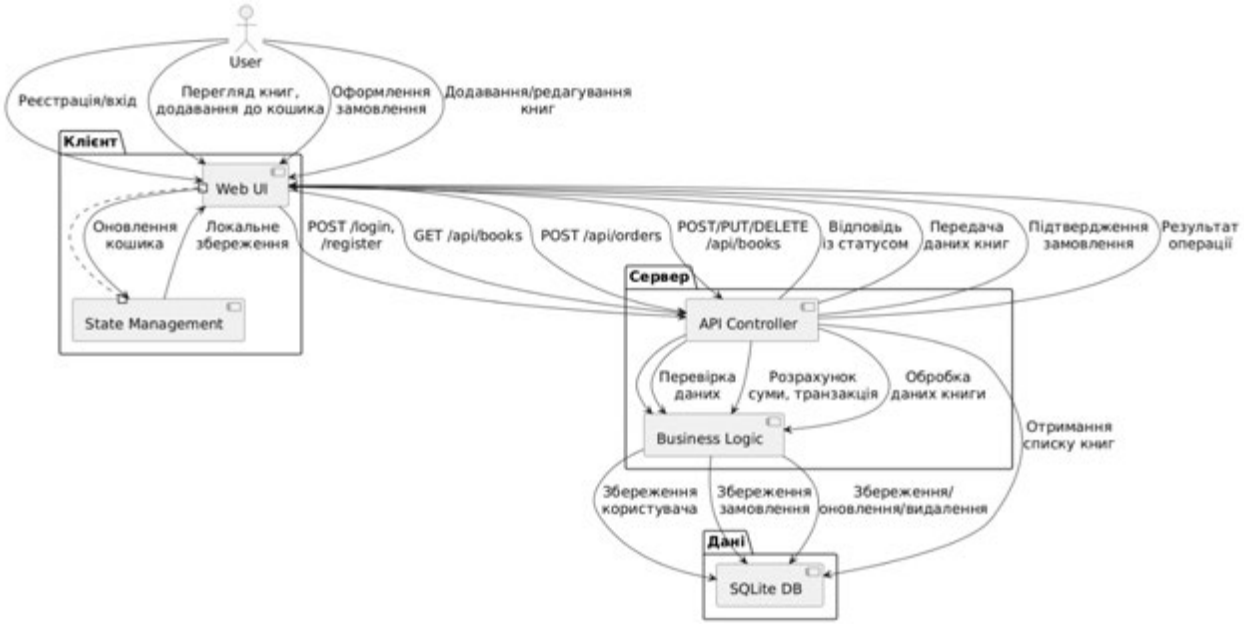

Рис. 2.5. Діаграма компонентів

Така архітектура забезпечує модульність, масштабованість і зручність підтримки системи. Клієнтський рівень ізольовано від серверного, що дозволяє легко оновлювати інтерфейс без змін у логіці сервера. Використання SQLite як легкої бази даних спрощує розгортання системи, хоча при зростанні навантаження може знадобитися перехід на більш потужну СУБД.

3. РОЗРОБКА ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Система управління інформаційною базою

Розробка інформаційного та програмного забезпечення для інформаційної системи продажу книжкової продукції передбачає створення ефективної системи управління інформаційною базою, яка є основою для забезпечення функціональності, надійності та зручності використання системи. У даному проєкті система управління інформаційною базою реалізована з використанням легкої реляційної бази даних SQLite, що інтегрована з серверною частиною на основі Node.js та фреймворку Express, а також клієнтською частиною, побудованою на React. Такий підхід дозволяє забезпечити швидкий доступ до даних, їхню структурованість та підтримку основних операцій CRUD (Create, Read, Update, Delete).

Інформаційна база складається з п’яти основних таблиць, кожна з яких відповідає за зберігання певного типу даних (таблиця 3.1).

Таблиця 3.1

Основні таблиці інформаційної бази

№	Назва	Опис
1	2	3
1	Таблиця «users»	зберігає інформацію про користувачів системи, включаючи унікальний ідентифікатор (id), електронну пошту (email), пароль (password), роль (role: «customer» або «manager») та ім’я (name). Ця таблиця є основою для автентифікації та авторизації користувачів
2	Таблиця «books»	містить дані про книжкову продукцію: ідентифікатор (id), назву (title), автора (author), категорію (category), ціну (price), опис (description), URL зображення (image) та кількість на складі (inStock). Ця таблиця є центральною для каталогу товарів

Продовження таблиці 3.1

1	2	3
3	Таблиця "orders"	призначена для зберігання інформації про замовлення: ідентифікатор (id), ідентифікатор користувача (userId), загальну суму (totalAmount), дані про доставку (shipping), платіжну інформацію (payment), статус (status) та дату створення (date). Вона пов’язана з таблицею "users" через зовнішній ключ userId
4	Таблиця "order_items"	деталізує складові кожного замовлення, включаючи ідентифікатор (id), ідентифікатор замовлення (orderId), ідентифікатор книги (bookId), назву книги (title), ціну (price) та кількість (quantity). Ця таблиця забезпечує зв’язок між "orders" та "books" через зовнішні ключі
5	Таблиця "manager_account"	використовується для зберігання балансу менеджера (balance), що оновлюється при обробці замовлень, відображаючи фінансовий стан системи

Управління інформаційною базою здійснюється через серверну частину (файл server.js), яка реалізує RESTful API для взаємодії з базою даних. Наприклад, ендпоінт /api/books дозволяє отримувати список усіх книг (GET), додавати нову книгу (POST), оновлювати існуючу (PUT) або видаляти (DELETE). Аналогічно, ендпоінт /api/orders забезпечує створення та управління замовленнями. Для підвищення надійності операцій із замовленнями використовуються транзакції (BEGIN TRANSACTION, COMMIT, ROLLBACK), що гарантують цілісність даних у разі збоїв.

На клієнтській стороні (файл app.js) застосовується технологія LocalStorage для тимчасового зберігання даних користувача, кошика, обраних книг та переглянутих товарів. Це дозволяє зберігати стан системи навіть після закриття браузера, забезпечуючи безперервність взаємодії користувача з інтерфейсом. Наприклад, функція React.useEffect синхронізує дані з LocalStorage при завантаженні сторінки та оновлює їх при змінах стану.

Система підтримує динамічне оновлення даних завдяки асинхронним запитам до API. Функції, такі як fetchBooks, fetchOrders та fetchManagerAccount, отримують актуальні дані з сервера та оновлюють стан компонентів React. Для

користувачів із роллю "manager" передбачено додаткові можливості, такі як управління каталогом книг (додавання, редагування, видалення) та зміна статусів замовлень, що реалізовано через функції addBook, updateBook, deleteBook та updateOrderStatus.

SQLite обрано через його легкість і відсутність необхідності в окремому сервері баз даних, що спрощує розгортання системи. Однак для захисту від SQL-ін’єкцій використовується параметризований підхід у запитах (наприклад, db.prepare(...).run()), що забезпечує безпечну обробку введених даних. Також система включає базову валідацію на клієнтській стороні (наприклад, перевірка формату email та пароля у LoginPage та RegisterPage), що зменшує навантаження на сервер.

Файл seed.js відповідає за початкове наповнення бази даних тестовими даними, включаючи користувачів, книги та замовлення. Це дозволяє протестувати систему перед реальним використанням. Процес включає очищення таблиць, скидання автоінкременту та створення записів із випадковими значеннями (наприклад, кількість книг на складі генерується у межах 5–55).

Таким чином, система управління інформаційною базою забезпечує структуроване зберігання даних, їх швидкий доступ і обробку, а також підтримку основних функцій книжкового онлайн-магазину. Поєднання серверної логіки, клієнтського інтерфейсу та локального кешування створює надійну основу для ефективної роботи системи, відповідаючи вимогам сучасних інформаційних технологій.

3.2 Розробка інформаційної бази

Розробка інформаційної бази для програмного забезпечення інформаційної системи продажу книжкової продукції є ключовим етапом, що забезпечує ефективне зберігання, обробку та доступ до даних. У даному проєкті

інформаційна база реалізована з використанням реляційної бази даних SQLite, яка обрана завдяки своїй легкості, портативності та достатній функціональності для потреб системи. SQLite інтегрується з серверною частиною, написаною на Node.js з використанням бібліотеки better-sqlite3, що забезпечує швидкий і безпечний доступ до даних. У цьому підрозділі детально розглядається структура інформаційної бази, її таблиці, зв’язки між ними, а також процес початкового заповнення даними (seed).

Інформаційна база складається з п’яти основних таблиць: users, books, orders, order_items та manager_account. Кожна таблиця має чітко визначену структуру, що відображає функціональні вимоги системи, а саме: управління користувачами, каталогом книг, замовленнями, деталями замовлень та фінансовим балансом менеджерів. Детальний опис кожної таблиці наведено в таблиці 3.2.

Таблиця 3.2

Детальний опис таблиць бази

№	Назва та опис	Структура таблиць
1	2	3
1	Таблиця users призначена для зберігання інформації про користувачів системи — як покупців, так і менеджерів. Ця таблиця є основою для автентифікації та авторизації, дозволяючи системі розрізняти права доступу	id (INTEGER, PRIMARY KEY, AUTOINCREMENT) — унікальний ідентифікатор користувача
		email (TEXT, UNIQUE) — електронна пошта, що слугує логіном
		password (TEXT) — пароль (зберігається у відкритому вигляді для спрощення, хоча в реальних системах рекомендується хешування)
		role (TEXT) — роль користувача (customer або manager)
		name (TEXT) — ім’я користувача
2	Таблиця books містить каталог книжкової продукції. Таблиця забезпечує зберігання повного переліку товарів, доступних для продажу, та їх	id (INTEGER, PRIMARY KEY, AUTOINCREMENT) — унікальний ідентифікатор книги
		title (TEXT) — назва книги
		author (TEXT) — автор
		category (TEXT) — категорія (наприклад, "Українська класика", "Фантастика")

	характеристик	price (REAL) — ціна в гривнях
		description (TEXT) — опис книги
		image (TEXT) — URL зображення книги
		inStock (INTEGER) — кількість одиниць на складі
3	Таблиця orders зберігає інформацію про замовлення користувачів. Ця таблиця пов’язана з users через зовнішній ключ userId, що забезпечує відстеження замовлень за користувачами	id (INTEGER, PRIMARY KEY, AUTOINCREMENT) — ідентифікатор замовлення
		userId (INTEGER, FOREIGN KEY) — посилання на користувача з таблиці users
		totalAmount (REAL) — загальна сума замовлення
		shipping (TEXT) — дані про доставку у форматі JSON (ім’я, адреса, телефон тощо)
		payment (TEXT) — дані про оплату у форматі JSON (метод, останні цифри картки)
		status (TEXT) — статус замовлення (наприклад, "Новий", "Доставлений")
		date (TEXT) — дата створення у форматі ISO
4	Таблиця order_items деталізує товари в кожному замовленні. Таблиця є проміжною між orders і books, дозволяючи зберігати кілька товарів у межах одного замовлення	id (INTEGER, PRIMARY KEY, AUTOINCREMENT) — ідентифікатор запису
		orderId (INTEGER, FOREIGN KEY) — посилання на замовлення з таблиці orders
		bookId (INTEGER, FOREIGN KEY) — посилання на книгу з таблиці books
		title (TEXT) — назва книги (дублюється для зручності)
		price (REAL) — ціна на момент замовлення
		quantity (INTEGER) — кількість одиниць
5	Таблиця manager_account використовується для відстеження фінансового балансу менеджерів. Баланс оновлюється при створенні замовлень (крім скасованих), відображаючи загальний дохід	id (INTEGER, PRIMARY KEY, AUTOINCREMENT) — ідентифікатор запису
		balance (REAL) — поточний баланс

Реляційна модель бази даних побудована на основі зовнішніх ключів:

– orders.userId пов’язує замовлення з користувачем із таблиці users;

– order_items.orderId зв’язує деталі замовлення з таблицею orders;

– order_items.bookId пов’язує товари в замовленні з таблицею books.

Така структура забезпечує цілісність даних і дозволяє ефективно виконувати запити, наприклад, для отримання всіх книг у конкретному замовленні чи історії покупок користувача.

Процес створення таблиць і початкового заповнення реалізовано у файлі server.js (функція initDB) та seed.js. У initDB створюються таблиці, якщо вони відсутні, а також додається початковий запис у manager_account з балансом 0. Файл seed.js відповідає за наповнення бази тестовими даними:

– Додаються 2 менеджери та 5 клієнтів до users;

– Вносяться 25 книг із різними категоріями до books (зображення — заглушки з via.placeholder.com);

– Генеруються 10 замовлень із випадковими товарами та статусами до orders і order_items;

– Баланс у manager_account оновлюється залежно від суми замовлень.

Цей підхід дозволяє протестувати систему перед реальним використанням.

Розроблена інформаційна база є гнучкою та масштабовною, відповідаючи потребам системи продажу книг. Використання SQLite спрощує розгортання, а чітка структура таблиць і зв’язків забезпечує швидкий доступ до даних і їхню цілісність. У майбутньому базу можна вдосконалити, додавши індекси для оптимізації запитів або перейшовши на більш потужну СУБД (наприклад, PostgreSQL) при зростанні обсягу даних.

3.3 Вибір інструментарію для створення прикладного програмного забезпечення

Розробка інформаційної системи для продажу книжкової продукції потребує ретельного підходу до вибору інструментарію, який забезпечить ефективність, масштабованість, зручність використання та відповідність сучасним стандартам веб-розробки. У даному проєкті, на основі реалізованих кодів (app.js, server.js, seed.js), вибір інструментів був здійснений з урахуванням

специфіки задачі – створення інтерактивного веб-додатку з функціями електронної комерції. Важливо детально розглянети обрані технології, їх переваги та обґрунтування доцільності використання.

Для створення клієнтської частини інформаційної системи обрано бібліотеку React.js, яка є одним із найпопулярніших інструментів для побудови динамічних інтерфейсів користувача. React дозволяє розробляти компонентно-орієнтовані додатки, що значно спрощує управління станом, повторне використання коду та підтримку проєкту. У коді app.js видно, що основний компонент App використовує хуки (useState, useEffect) для управління станом користувача, кошика, обраного тощо, а також для асинхронного завантаження даних із сервера. Переваги React включають високу продуктивність завдяки віртуальному DOM, велику екосистему бібліотек і активну спільноту розробників. Обраний інструмент ідеально підходить для реалізації таких функцій, як відображення каталогу книг, пошук, кошик і сторінки оформлення замовлення, оскільки забезпечує швидке оновлення інтерфейсу без перезавантаження сторінки.

Для серверної частини використано Node.js у поєднанні з фреймворком Express.js, що є оптимальним вибором для створення RESTful API. Node.js дозволяє використовувати JavaScript як єдину мову програмування для фронтенду і бекенду, що спрощує розробку та зменшує поріг входження для команди. Express.js, як видно з server.js, забезпечує маршрутизацію запитів (наприклад, /api/books, /api/orders), обробку HTTP-методів (GET, POST, PUT, DELETE) та інтеграцію з базою даних. Цей інструментарій вирізняється легкістю, швидкістю роботи та можливістю масштабування, що є критично важливим для системи електронної комерції з потенційно великою кількістю запитів.

Для зберігання даних обрано легку реляційну базу даних SQLite, яка інтегрована через модуль better-sqlite3. SQLite є вбудованою базою даних, що не потребує окремого серверного процесу, що робить її зручною для невеликих і

середніх проєктів, таких як цей. У seed.js видно, що база даних використовується для зберігання інформації про користувачів, книги, замовлення та баланс менеджера. Переваги SQLite включають простоту налаштування, високу швидкість роботи з невеликими обсягами даних і відсутність необхідності в складному адмініструванні. Хоча для великих систем із високим навантаженням доцільніше було б обрати PostgreSQL або MySQL, для даної задачі SQLite є достатньою завдяки своїй компактності та ефективності.

Для обробки HTTP-запитів на сервері використано body-parser, що дозволяє зручно парсити JSON-дані з клієнтських запитів. Статичні файли (HTML, CSS) обслуговуються через Express, що спрощує розгортання фронтенду. У seed.js використано модуль better-sqlite3 для підготовки початкових даних, що демонструє зручність роботи з транзакціями та масовими вставками. Для стилізації інтерфейсу, хоча CSS не надано в повному обсязі, передбачається використання модульного підходу з можливістю інтеграції CSS-фреймворків, таких як Bootstrap, для адаптивного дизайну.

Вибір React, Node.js, Express і SQLite зумовлений кількома факторами. По-перше, це забезпечення єдиної мови програмування (JavaScript), що оптимізує процес розробки та зменшує ймовірність помилок при інтеграції фронтенду і бекенду. По-друге, ці інструменти є відкритими, безкоштовними та мають широку підтримку спільноти, що полегшує пошук рішень і документації. По-третє, легкість розгортання та низькі вимоги до ресурсів (особливо завдяки SQLite) роблять систему доступною для тестування та запуску навіть на локальних машинах. Нарешті, модульність і гнучкість обраного стеку дозволяють легко розширювати функціонал, наприклад, додавати автентифікацію через JWT або інтеграцію з платіжними системами.

Обраний інструментарій – React.js для фронтенду, Node.js з Express.js для бекенду та SQLite для бази даних – є збалансованим рішенням для створення прикладного програмного забезпечення інформаційної системи продажу книг. Цей стек забезпечує швидку розробку, зручність підтримки та достатню

продуктивність для реалізації всіх необхідних функцій: від перегляду каталогу до обробки замовлень. У майбутньому, за потреби масштабування, можлива заміна SQLite на більш потужну СУБД або додавання інструментів для кешування (наприклад, Redis), але на поточному етапі обрані технології повністю відповідають поставленим вимогам.

3.4 Алгоритмізація та програмування програмних модулів

Розробка інформаційного та програмного забезпечення інформаційної системи для продажу книжкової продукції передбачає створення структурованої сукупності програмних модулів, які забезпечують функціональність системи, її взаємодію з користувачами та обробку даних. У даному проєкті, реалізованому у вигляді веб-додатку, використано сучасні технології, зокрема React для клієнтської частини, Node.js із Express для серверної частини та SQLite як базу даних. Алгоритмізація та програмування модулів базуються на чіткому розподілі функціональних завдань, що дозволяє забезпечити ефективність, масштабованість і зручність використання системи.

Алгоритмізація є ключовим етапом розробки, який визначає логіку роботи кожного модуля. У проєкті можна виділити кілька основних функціональних блоків: автентифікація користувачів, управління каталогом книг, обробка кошика та замовлень, а також адміністративні функції для менеджерів. Для кожного блоку розроблено відповідні алгоритми (таблиця 3.3).

Таблиця 3.3

Основні алгоритми системи

№	Назва	Опис
1	2	3
1	Автентифікація та реєстрація користувачів	Алгоритм автентифікації включає перевірку введених даних (email і пароль) на відповідність записам у базі даних. У разі успішної автентифікації користувачу надається доступ до персоналізованих функцій (наприклад, кошика чи кабінету). Реєстрація передбачає валідацію введених даних (перевірка унікальності email, відповідності формату пароля) та збереження нового запису в таблиці users. Логіка реалізована в серверних ендпоінтах /api/login і /api/register, де використовуються SQL-запити для взаємодії з базою даних
2	Управління каталогом книг	Алгоритм відображення каталогу передбачає завантаження списку книг із бази даних через ендпоінт /api/books, фільтрацію за категоріями та пошук за ключовими словами. Для додавання, редагування чи видалення книг (функції менеджера) розроблено алгоритми, які включають валідацію даних (наприклад, перевірка наявності обов’язкових полів) і відповідні операції CRUD (Create, Read, Update, Delete) у базі даних через ендпоінти /api/books (POST, PUT, DELETE)
3	Обробка кошика та замовлень	Алгоритм роботи з кошиком включає додавання книг, зміну кількості одиниць і видалення позицій. Дані кошика зберігаються локально у localStorage, а при оформленні замовлення передаються на сервер через ендпоінт /api/orders. Алгоритм створення замовлення передбачає транзакційну обробку: збереження даних замовлення в таблиці orders, додавання позицій у order_items та оновлення балансу менеджера в manager_account. У разі помилки транзакція відкочується, що забезпечує цілісність даних
4	Адміністративні функції	Для менеджерів реалізовано алгоритми управління замовленнями (оновлення статусу через /api/orders/:id) та перегляду фінансового балансу (/api/manager/account). Логіка включає вибірку даних із бази та їх відображення у зрозумілому форматі

Програмування модулів виконано з урахуванням модульності та повторного використання коду. Основні напрями реалізації показані в таблиці 3.4.

Таблиця 3.4

Основні напрями реалізації

№	Назва	Опис
1	Клієнтська частина (app.js)	У файлі app.js реалізовано головний компонент React-додатку App, який управляє станом системи (користувач, сторінки, книги, кошик тощо) за допомогою хуків useState і useEffect. Компонент включає функції для взаємодії з API (наприклад, fetchBooks, login, addToCart), а також рендеринг сторінок залежно від стану (renderContent). Окремі компоненти, такі як Header, Footer, HomePage, BookCard, відповідають за відображення інтерфейсу та обробку подій користувача. Наприклад, BookCard реалізує логіку додавання книги до кошика чи обраного, використовуючи пропси для передачі функцій
2	Серверна частина (server.js)	Сервер побудовано на базі Express.js і забезпечує RESTful API для взаємодії з базою даних SQLite через бібліотеку better-sqlite3. Модуль включає обробку запитів (GET, POST, PUT, DELETE) для книг, замовлень і користувачів. Наприклад, ендпоінт /api/orders реалізує транзакційну логіку створення замовлення, а /api/books/:id — оновлення чи видалення книги. Використання підготовлених SQL-запитів (db.prepare) підвищує безпеку та продуктивність
3	Ініціалізація даних (seed.js)	Модуль seed.js призначений для початкового заповнення бази даних тестовими даними. Алгоритм включає очищення таблиць, створення користувачів, книг і замовлень із випадковими параметрами (наприклад, кількість книг у наявності чи статус замовлення). Це дозволяє протестувати систему в реальних умовах

Алгоритмізація та програмування модулів інформаційної системи для продажу книжкової продукції виконані з урахуванням принципів модульності, безпеки та зручності. Використання React забезпечило динамічний інтерфейс, Express — швидку серверну обробку, а SQLite — легке управління даними. Розроблені алгоритми є логічно завершеними, а код — структурованим, що полегшує подальше масштабування та підтримку системи. Реалізація проєкту

демонструє ефективне поєднання фронтенд- і бекенд-технологій для створення повноцінного веб-додатку.

4. РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

4.1 Тестування системи

Тестування інформаційної системи для продажу книжкової продукції є ключовим етапом її впровадження, що дозволяє забезпечити стабільність, надійність та відповідність функціональних можливостей заявленим вимогам. Розроблена система, представлена у реалізованому коді (app.js, server.js, seed.js), базується на технологіях React для клієнтської частини, Node.js з Express для серверної частини та SQLite як бази даних. У зв’язку з цим тестування має охоплювати як фронтенд, так і бекенд, а також їхню взаємодію. Важливо детально розглянути опис процесу тестування, включаючи типи тестів, методики, інструменти та рекомендації щодо їх реалізації.

Модульне тестування (Unit Testing) спрямоване на перевірку окремих компонентів системи, таких як функції та методи. Для клієнтської частини (React) необхідно протестувати компоненти, наприклад, BookCard, Header, CartPage, а також хуки React, такі як useState і useEffect. На серверній частині (Node.js) слід перевірити API-ендпоінти, такі як /api/books, /api/orders, та логіку обробки запитів.

Інструменти: для React рекомендується використовувати бібліотеки Jest та React Testing Library, які дозволяють створювати тести для компонентів та їхньої поведінки. Для серверної частини підійде Jest у поєднанні з Supertest для тестування HTTP-запитів.

Приклад: перевірка функції addToCart у компоненті App на коректне додавання книги до кошика, враховуючи випадки, коли книга вже є в кошику (збільшення кількості) або додається вперше. На сервері можна протестувати ендпоінт /api/login, переконавшись, що він повертає правильний статус (200 при успіху, 401 при помилці).

Рекомендація: розробити не менше 80% покриття коду тестами, щоб гарантувати стабільність основних функцій.

Інтеграційне тестування (Integration Testing) перевіряє взаємодію між компонентами системи, зокрема між фронтендом і бекендом, а також між сервером і базою даних. Наприклад, необхідно переконатися, що запит на створення замовлення (createOrder) коректно передає дані до сервера, оновлює базу даних і повертає відповідь користувачу.

Інструменти: Cypress або Playwright для енд-ту-енд тестування взаємодії клієнтської частини з сервером. Supertest для перевірки API у зв’язці з SQLite.

Приклад: тестування сценарію оформлення замовлення: користувач додає книгу до кошика, переходить до сторінки CheckoutPage, вводить дані доставки та оплати, після чого система створює запис у таблиці orders і очищає кошик.

Рекомендація: перевірити всі ключові сценарії користувацької взаємодії (реєстрація, авторизація, пошук книг, оформлення замовлення) та обробку помилок (наприклад, відмова сервера або некоректні дані).

Функціональне тестування (Functional Testing) оцінює відповідність системи специфікаціям. Кожен модуль (авторизація, кошик, пошук, керування книгами для менеджерів) має бути протестований на коректність виконання заявлених функцій.

Інструменти: ручне тестування за допомогою підготовлених тестових сценаріїв або автоматизоване тестування через Selenium чи Cypress.

Приклад: перевірка функції пошуку (searchBooks): введення запиту "Кобзар" має повернути книгу Тараса Шевченка, а порожній запит — очистити результати. Для менеджерів тестується додавання книги через ManageBooksPage з перевіркою оновлення списку книг.

Рекомендація: скласти чек-лист функціональних вимог і протестувати кожен пункт, звертаючи увагу на поведінку системи при крайових випадках (наприклад, додавання книги з від’ємною ціною).

Тестування продуктивності (Performance Testing): оскільки система працює з базою даних SQLite і може обробляти одночасні запити від користувачів, необхідно оцінити її продуктивність під навантаженням.

Інструменти: JMeter або Artillery для симуляції одночасних запитів до API.

Приклад: Симуляція 100 одночасних запитів до /api/books для оцінки часу відповіді та стабільності сервера. Перевірка часу завантаження головної сторінки (HomePage) при великій кількості книг (наприклад, 1000 записів у базі).

Рекомендація: Забезпечити, щоб час відповіді не перевищував 2 секунд при 50 одночасних користувачах, а система витримувала до 200 запитів за хвилину без збоїв.

Тестування безпеки (Security Testing): система обробляє конфіденційні дані (email, паролі, платіжну інформацію), тому важливо перевірити її на вразливості.

Інструменти: OWASP ZAP для автоматичного сканування вразливостей, Burp Suite для ручного тестування.

Приклад: Перевірка ендпоінта /api/login на стійкість до SQL-ін’єкцій (наприклад, введення ' OR '1'='1 у поле пароля). Тестування захисту від XSS-атак шляхом введення скриптів у поля форм (наприклад, <script>alert('XSS')</script> у полі пошуку).

Рекомендація: Впровадити шифрування паролів (наприклад, за допомогою bcrypt) та валідацію всіх вхідних даних на сервері.

Тестування сумісності (Compatibility Testing): система має коректно працювати в різних браузерах і на різних пристроях, враховуючи адаптивний дизайн, описаний у CSS.

Інструменти: BrowserStack або ручне тестування на реальних пристроях.

Приклад: перевірка відображення сторінки HomePage у Chrome, Firefox, Safari та на мобільних пристроях (iOS, Android) з різними роздільними здатностями екрану (576px, 768px, 992px).

Рекомендація: забезпечити коректне відображення та функціональність на 90% цільових платформ, враховуючи медіа-запити в CSS.

Підготовка тестового середовища: створити окреме середовище з копією бази даних, заповненою тестовими даними через seed.js. Використовувати локальний сервер (порт 5000) для тестування, уникаючи впливу на продуктивну систему.

Розробка тестових сценаріїв: скласти детальні сценарії для кожного типу тестування. Наприклад:

– Сценарій 1: Користувач реєструється, входить у систему, додає книгу до кошика, оформляє замовлення.

– Сценарій 2: Менеджер додає нову книгу, змінює її статус, видаляє її.

– Сценарій 3: Перевірка поведінки при введенні некоректних даних (від’ємна кількість товару, невалідний email).

Виконання тестів: спочатку провести модульні тести для виявлення базових помилок, потім інтеграційні та функціональні тести. Тестування продуктивності та безпеки виконувати на завершальних етапах, коли основна функціональність підтверджена.

Аналіз результатів і виправлення помилок: зафіксувати всі виявлені дефекти у системі відстеження помилок (наприклад, Jira). Пріоритетність виправлення визначати за критичністю: помилки авторизації чи оплати мають вищий пріоритет, ніж косметичні дефекти інтерфейсу.

Для підвищення ефективності тестування рекомендується автоматизувати повторювані сценарії, такі як авторизація, додавання до кошика та оформлення замовлення. Автоматизовані тести можна інтегрувати в процес CI/CD (наприклад, через GitHub Actions), щоб запускати їх при кожному оновленні коду. Це дозволить швидко виявляти регресії та підтримувати стабільність системи під час її експлуатації.

Важливим етапом тестування проєкту є покрокове виконання програми.

Крок 1. Запуск

Система розпочинає роботу з запуску серверної частини через команду в терміналі, яка активує Node.js сервер на порту 5000. База даних SQLite ініціалізується, а тестові дані завантажуються автоматично для подальшого використання.

Крок 2. Ініціалізація бази даних

На цьому етапі створюються необхідні таблиці в базі даних, якщо їх ще немає, і завантажуються початкові дані, такі як користувачі, книги та замовлення, для забезпечення функціональності системи.

Крок 3. Відкриття головної сторінки

Користувач відкриває браузер і переходить на локальну адресу, де відображається головна сторінка магазину з переліком книг і категорій, доступних для перегляду.

Крок 4. Реєстрація користувача

Новий користувач натискає на кнопку реєстрації в заголовку, вводить свої дані, такі як ім’я, email і пароль, після чого система створює обліковий запис і перенаправляє на головну сторінку (рис. 4.1).

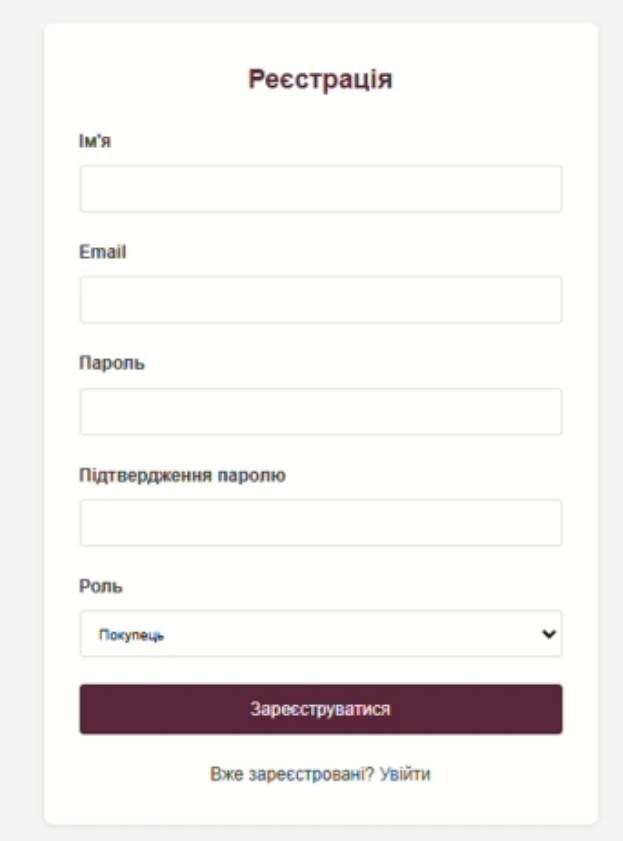

Рис. 4.1. Вікно реєстрації

Крок 5. Авторизація

Зареєстрований користувач вводить свої email і пароль на сторінці входу, система перевіряє дані, і в разі успіху користувач отримує доступ до персоналізованих функцій (рис. 4.2, рис. 4.3).

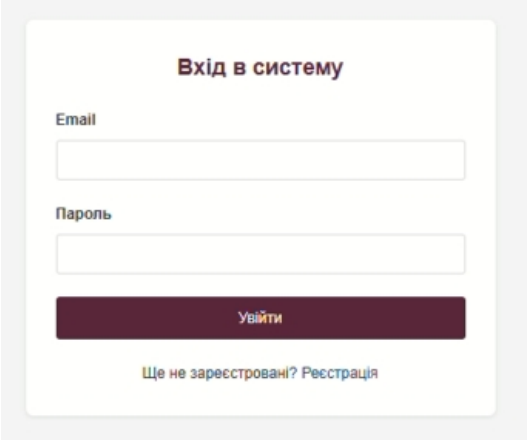

Рис. 4.2. Вхід у систему

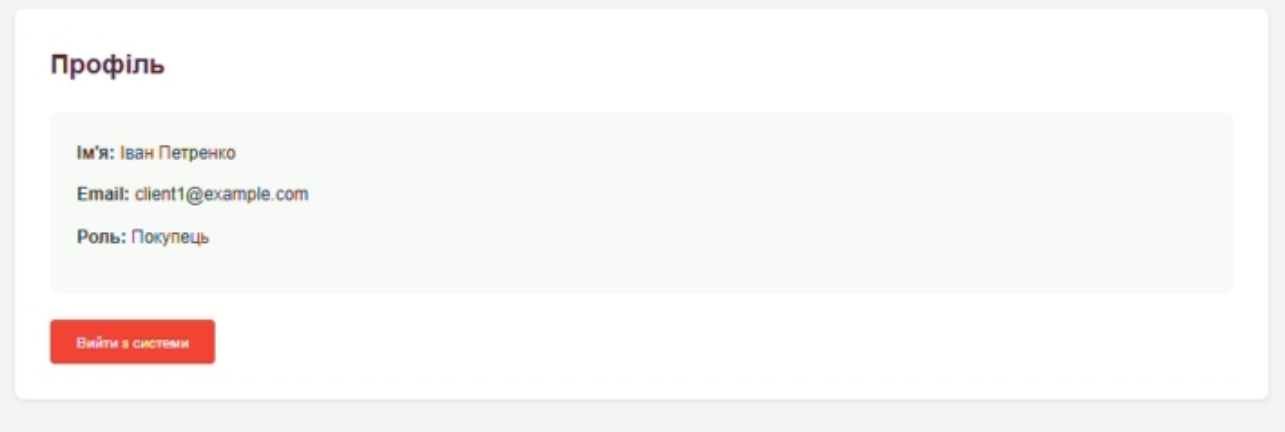

Рис. 4.3. Профіль користувача

Крок 6. Перегляд книг

Користувач переглядає список книг на головній сторінці, обирає категорію або використовує пошук, щоб знайти потрібну книгу, наприклад, за назвою чи автором (рис. 4.4).

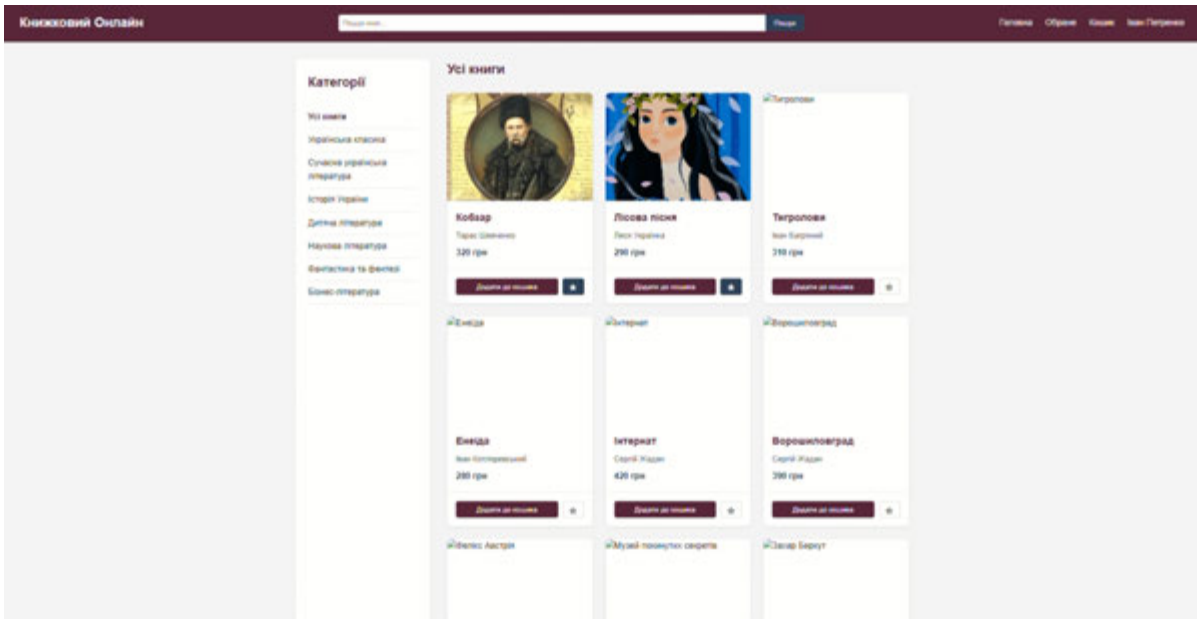

Рис. 4.4. Перегляд каталогу книг

Крок 7. Додавання книги до кошика

Обрану книгу користувач додає до кошика, натиснувши відповідну кнопку, після чого система оновлює стан кошика і відображає сповіщення про успішне додавання (рис. 4.5).

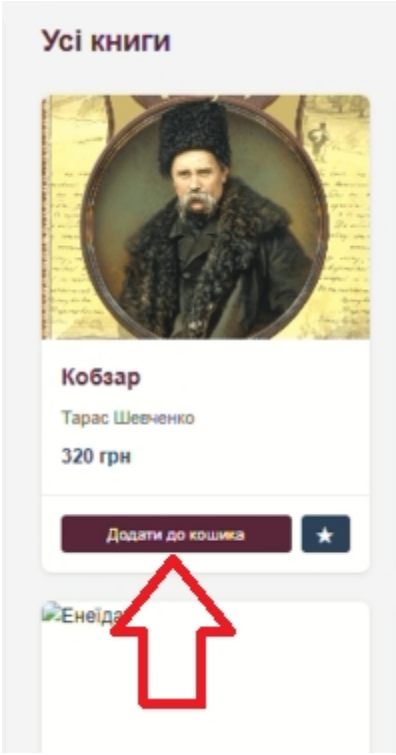

Рис. 4.5. Додавання книги до кошику

Крок 8. Перегляд кошика

Користувач переходить до сторінки кошика через навігацію, де бачить список доданих книг, їхню кількість і загальну суму, з можливістю змінити кількість або видалити позиції (рис. 4.6).

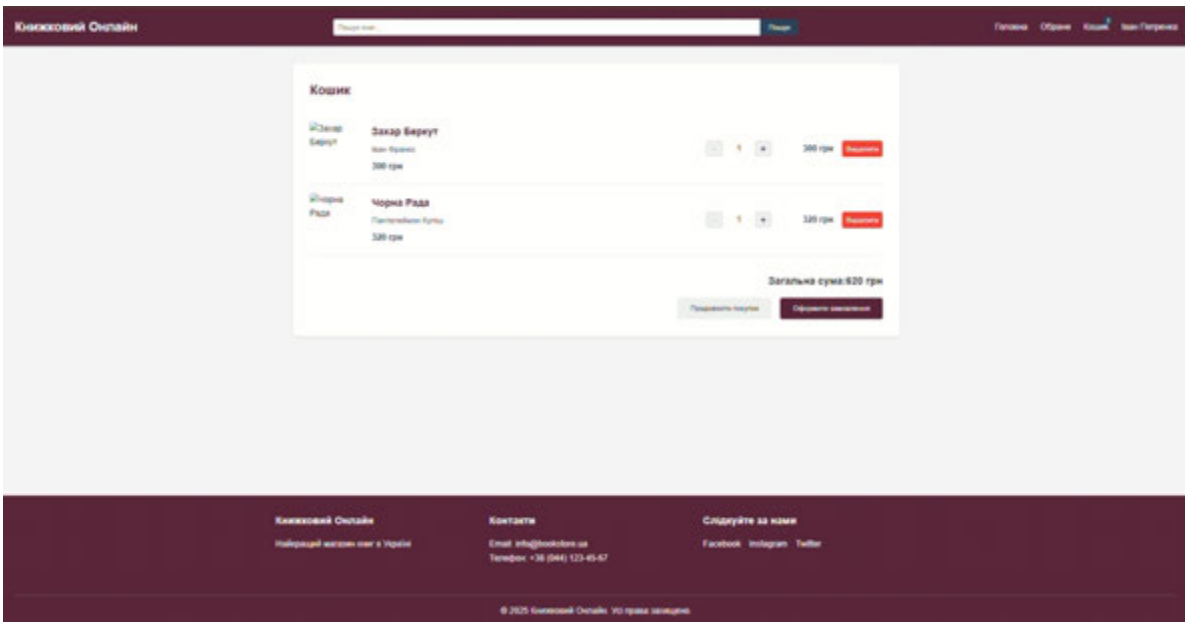

Рис. 4.6. Перегляд кошику

Крок 9. Оформлення замовлення

На сторінці кошика користувач натискає кнопку оформлення, вводить дані доставки та оплати на сторінці checkout, після чого система створює замовлення і очищає кошик (рис. 4.7).

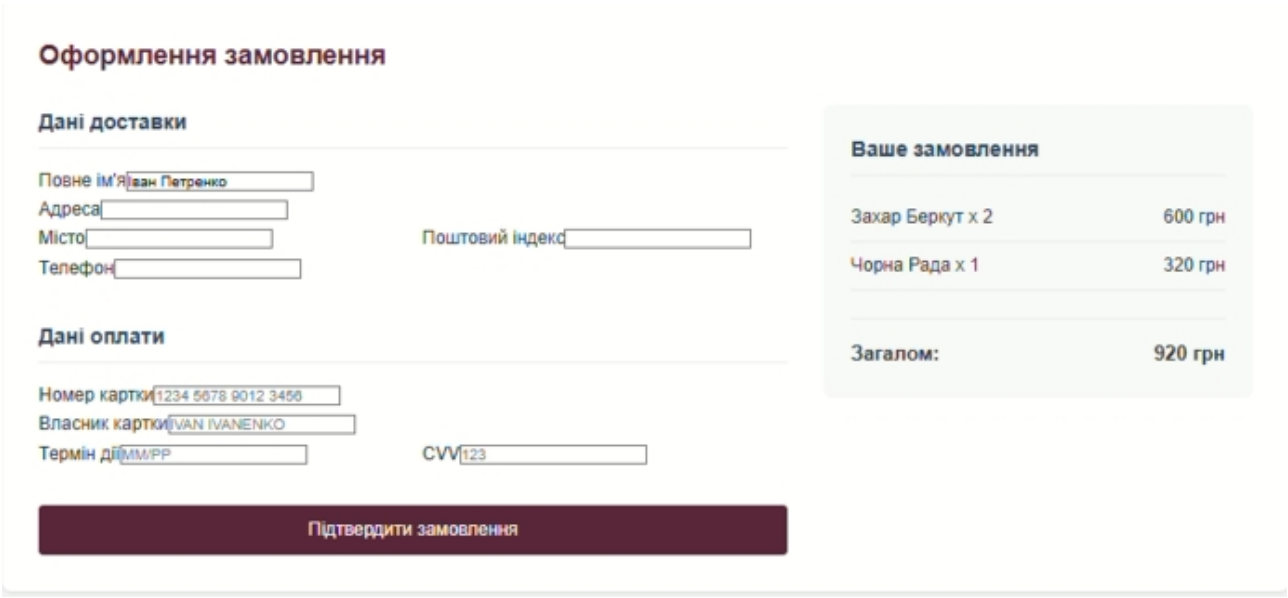

Рис. 4.7. Форма оформлення замовлення

Крок 10. Підтвердження замовлення

Після успішного оформлення користувач бачить сторінку підтвердження з подякою та інформацією про подальші дії, наприклад, очікування зв’язку з менеджером (рис. 4.8).

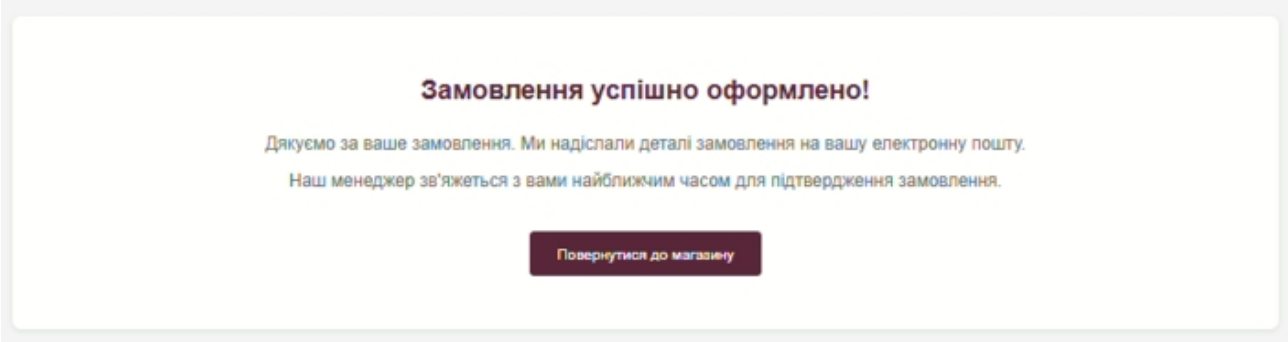

Рис. 4.8. Підтвердження замовлення

Крок 11. Перегляд обраного

Користувач може переглядати книги додані в обране (рис. 4.9).

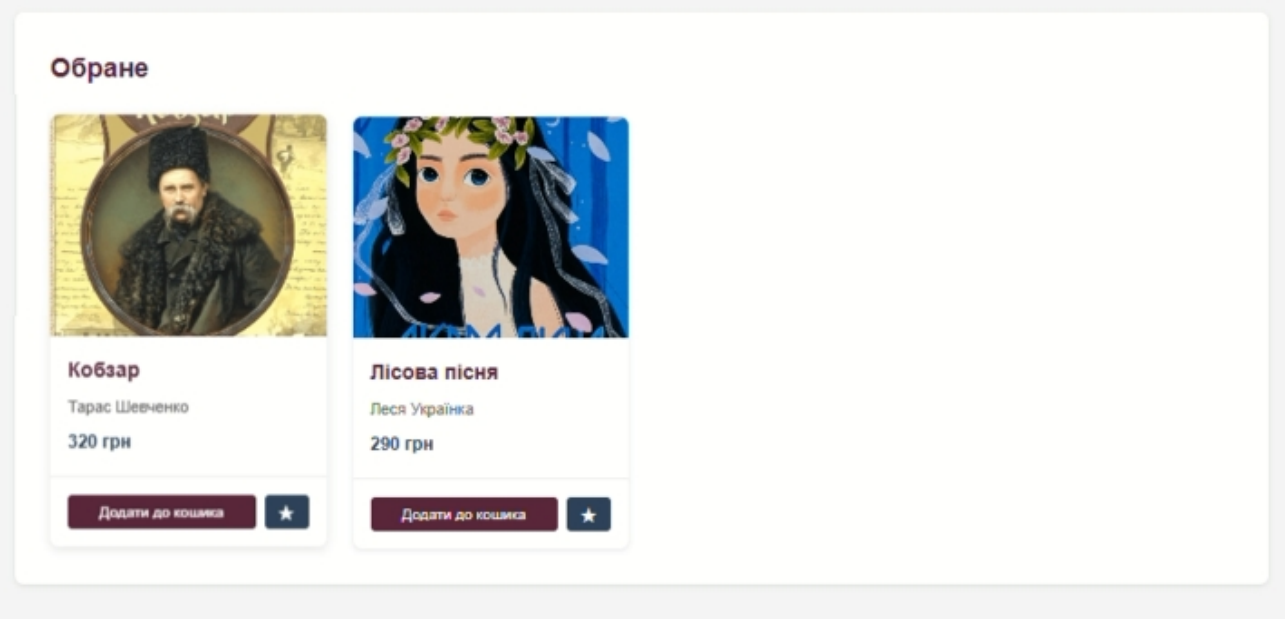

Рис. 4.9. Сторінка «Обране»

Крок 12. Робота менеджера

Менеджер входить у систему під своїм обліковим записом, переглядає список замовлень (рис. 4.10), змінює їхній статус(рис. 4.11), наприклад, на "Відправлений", додає нові книги до каталогу (рис. 4.12) та редагує наявні книги у каталозі (рис. 4.13, рис. 14).

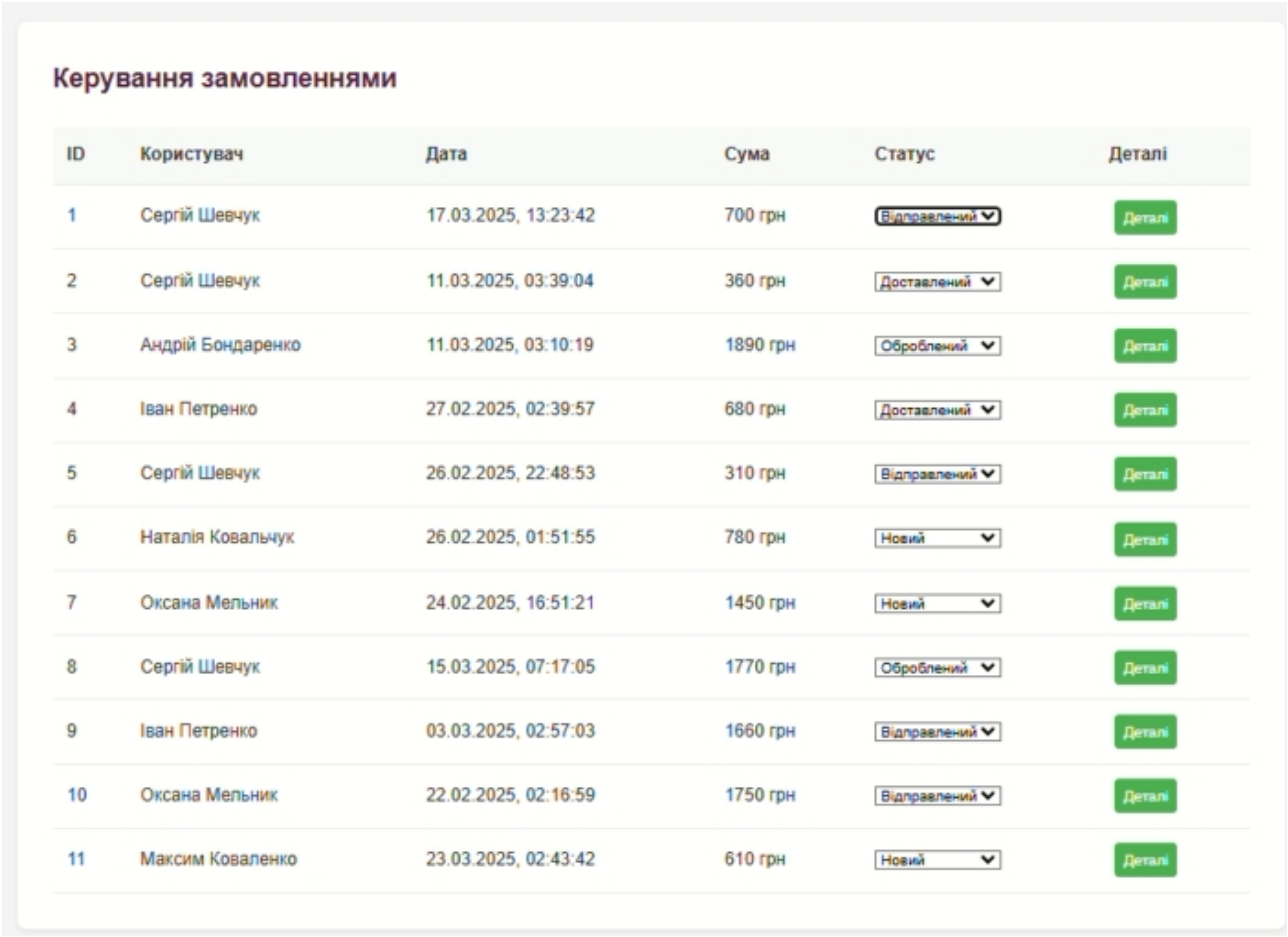

Рис. 4.10. Перегляд списку замовлень

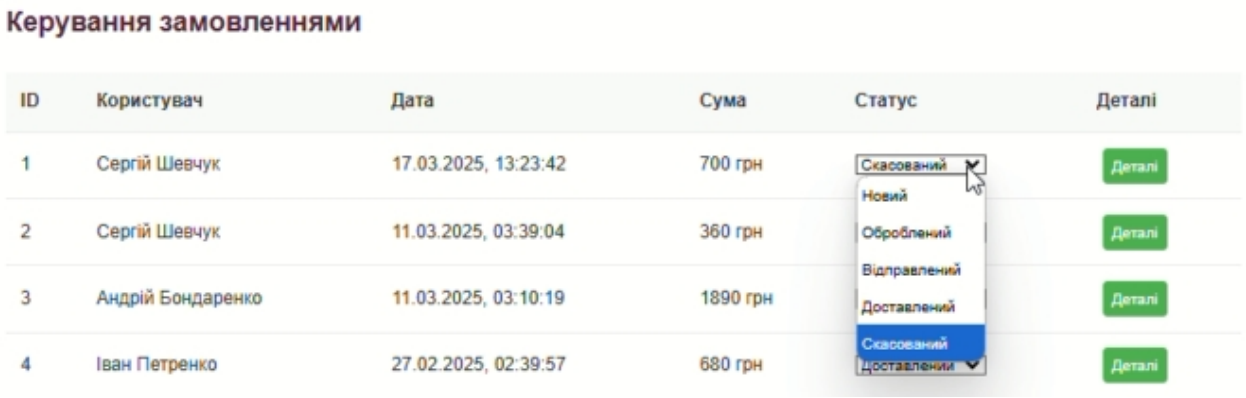

Рис. 4.11. Змінення статусу замовлення

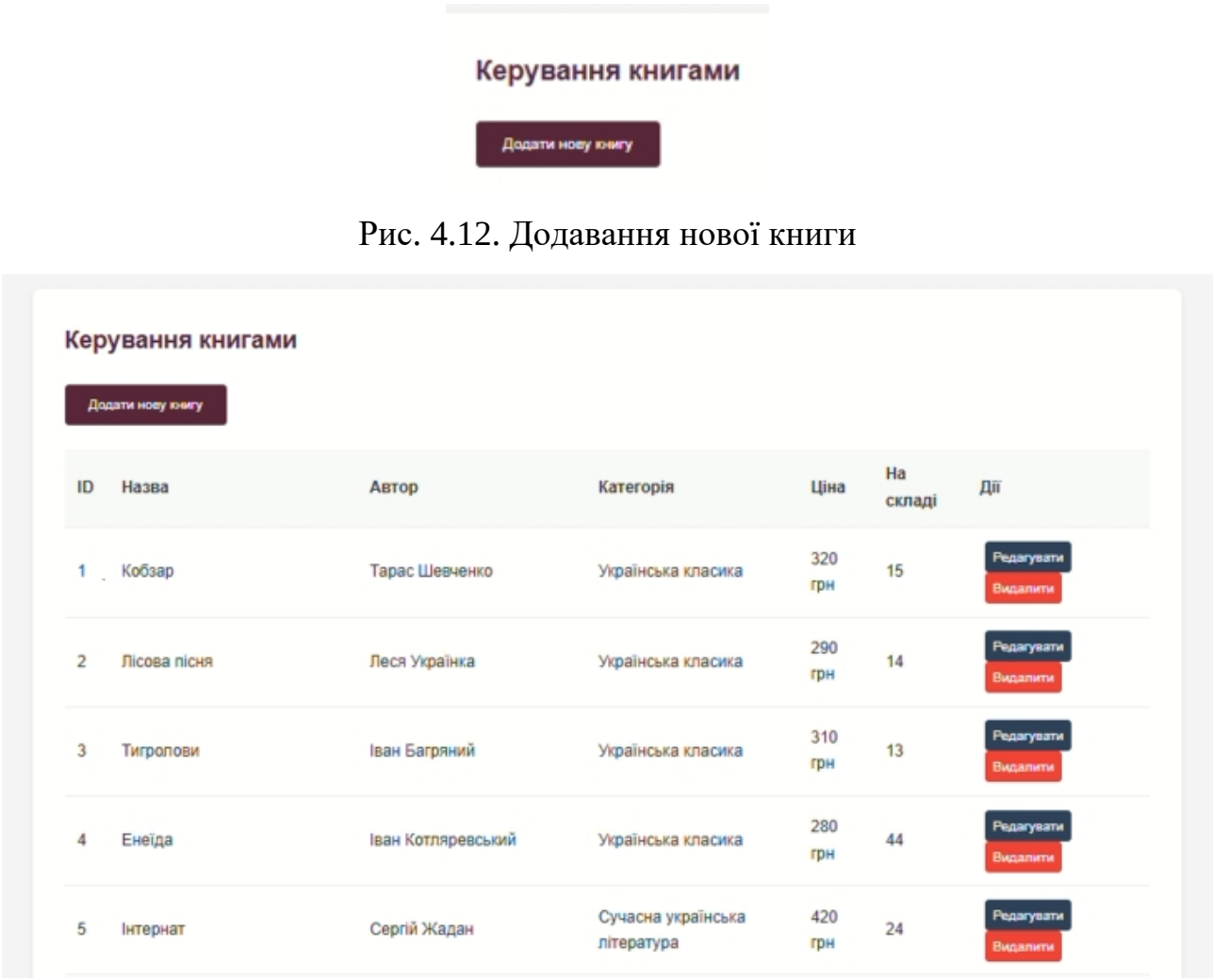

Рис. 4.12. Додавання нової книги

Рис. 4.13. Вікно керування книгами (редагування видалення, додавання)

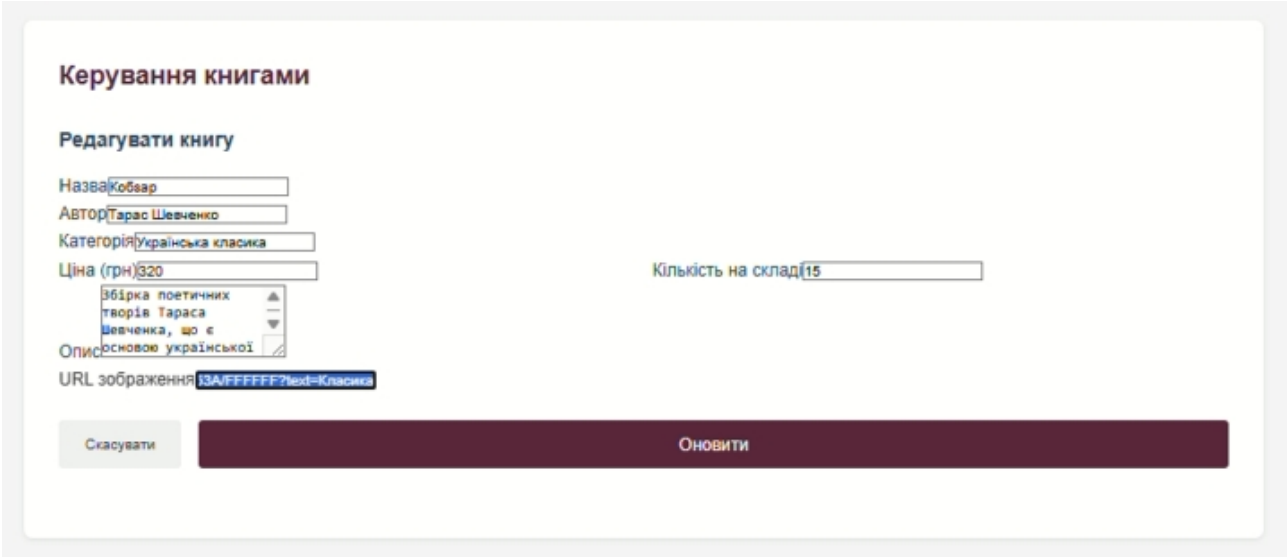

Рис. 4.14. Редагування книг у каталозі

Крок 13. Вихід із системи

Користувач або менеджер натискає кнопку виходу в профілі, система завершує сеанс, видаляє дані авторизації з локального сховища і повертає на головну сторінку (рис. 4.15).

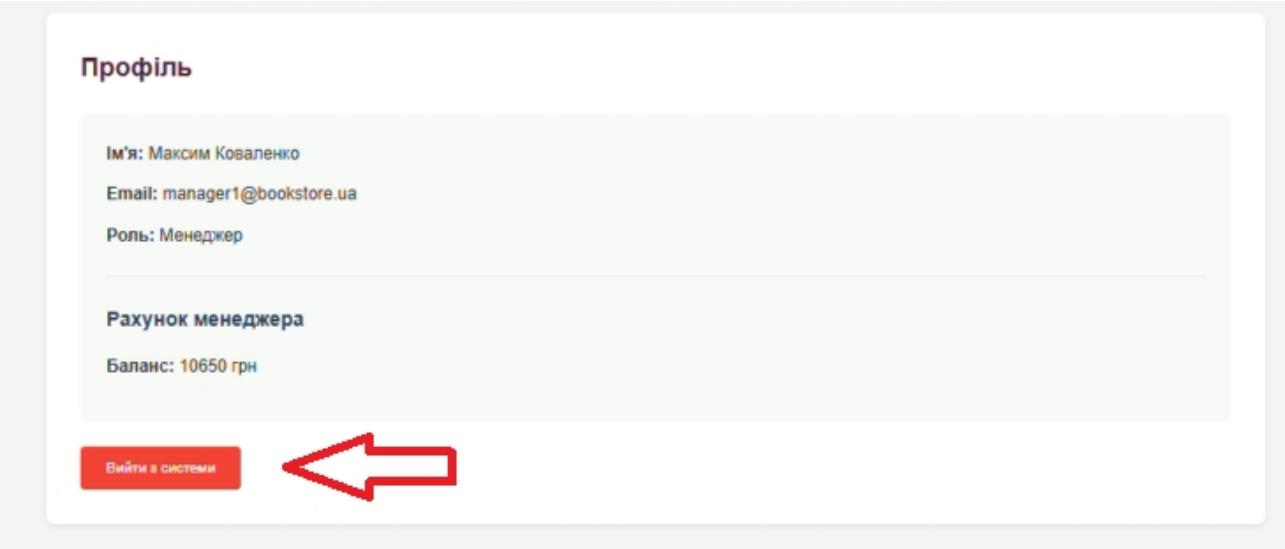

Рис. 4.15. Вихід із системи

Крок 13. Завершення

Робота з системою завершується закриттям браузера або зупинкою сервера через термінал, при цьому база даних зберігає всі зміни для наступного запуску.

Тестування системи має бути комплексним і охоплювати всі сторони її роботи — від окремих функцій до поведінки під навантаженням і захисту від атак. Ретельна підготовка тестових сценаріїв, використання сучасних інструментів і автоматизація ключових перевірок забезпечать високу якість продукту. Після завершення тестування система буде готова до впровадження, а виявлені недоліки можна усунути, підвищивши її надійність і зручність для користувачів.

4.2 Діаграма розгортання

Розроблена інформаційна система для продажу книжкової продукції, представлена у реалізованому коді, є веб-додатком, який базується на технологіях React для клієнтської частини, Node.js з Express для серверної частини та SQLite як системи управління базами даних (СУБД). Для успішного впровадження та стабільної експлуатації системи необхідно врахувати архітектурні особливості її розгортання, які відображаються у діаграмі розгортання. Цей розділ детально описує структуру розгортання системи, включаючи взаємодію між компонентами, їх розміщення на апаратних вузлах та рекомендації щодо конфігурації.

Клієнтський вузол (Користувач) системи представлений веб-інтерфейсом, реалізованим за допомогою React (файл app.js). Цей компонент розгортається у веб-браузері користувача (наприклад, Google Chrome, Mozilla Firefox тощо). Користувач взаємодіє з системою через графічний інтерфейс, який включає сторінки для перегляду книг, кошика, оформлення замовлень, авторизації та управління (для менеджерів). Веб-інтерфейс отримує статичні файли (HTML, CSS, JavaScript) від серверного вузла та здійснює динамічну взаємодію через API-запити. Для забезпечення коректної роботи рекомендується використовувати сучасні браузери з підтримкою ECMAScript 6+ та стабільне інтернет-з’єднання зі швидкістю не менше 5 Мбіт/с.

Серверний вузол (Сервер) системи розгортається на одному фізичному або віртуальному сервері, який виконує кілька ключових функцій:

– Веб-сервер (Express): цей компонент, реалізований у файлі server.js, обробляє HTTP-запити від клієнтського інтерфейсу. Він забезпечує доставку статичних файлів (наприклад, index.html) та маршрутизацію API-запитів (наприклад, /api/books, /api/orders). Express працює на базі Node.js і слухає запити на порту 5000 (або іншому, визначеному змінною середовища PORT).

Рекомендується конфігурація сервера з операційною системою Linux (наприклад, Ubuntu 20.04), що забезпечує стабільність і легкість масштабування.

– Сервер додатків: логіка обробки запитів (авторизація, управління книгами, замовленнями) також реалізована у server.js. Цей компонент взаємодіє з базою даних через модуль better-sqlite3, виконуючи операції читання, запису та оновлення даних. Для підвищення продуктивності рекомендується використовувати сервер з щонайменше 2 ГБ оперативної пам’яті та двоядерним процесором.

– База даних (SQLite): СУБД SQLite, яка зберігає дані про користувачів, книги, замовлення та баланс менеджера, розгортається на тому ж сервері у вигляді файлу bookstore.db. SQLite обрано через його легкість і відсутність необхідності в окремому сервері баз даних, що спрощує розгортання для невеликих систем. Однак для масштабування (наприклад, при зростанні кількості користувачів до 10 000+) рекомендується міграція на реляційну СУБД, таку як PostgreSQL, із розгортанням на окремому вузлі.

Взаємодія між компонентами:

– Клієнтський веб-інтерфейс надсилає запити до веб-сервера через протокол HTTPS (рекомендується налаштувати SSL-сертифікат для безпеки). Наприклад, виклик fetch('/api/books') у app.js отримує список книг.

– Веб-сервер передає запити до сервера додатків, який обробляє логіку та звертається до бази даних. Наприклад, при створенні замовлення (POST /api/orders) сервер додатків записує дані в таблиці orders і order_items.

– База даних повертає результати запитів серверу додатків, які потім форматуються у JSON і передаються назад клієнту через веб-сервер.

Рекомендації щодо конфігурації наведені в таблиці 4.1.

Таблиця 4.1

Рекомендації щодо конфігурації

№	Напрям	Рекомендації
1	Безпека	Налаштуйте HTTPS із сертифікатом від Let’s Encrypt, щоб захистити дані користувачів. У коді server.js додайте перевірку CSRF-токенів для захисту від атак типу "підробка міжсайтових запитів".
2	Масштабованість	Для високого навантаження розгорніть веб-сервер за допомогою Nginx як зворотного проксі, що розподілятиме запити між кількома екземплярами Node.js (наприклад, через PM2)
3	Резервне копіювання	Регулярно створюйте резервні копії файлу bookstore.db (наприклад, щоденно о 00:00 за допомогою cron) і зберігайте їх на віддаленому сховищі
4	Моніторинг	Використовуйте інструменти на кшталт Prometheus і Grafana для відстеження продуктивності сервера та часу відповіді API

Діаграма розгортання показана на рис. 4.16.

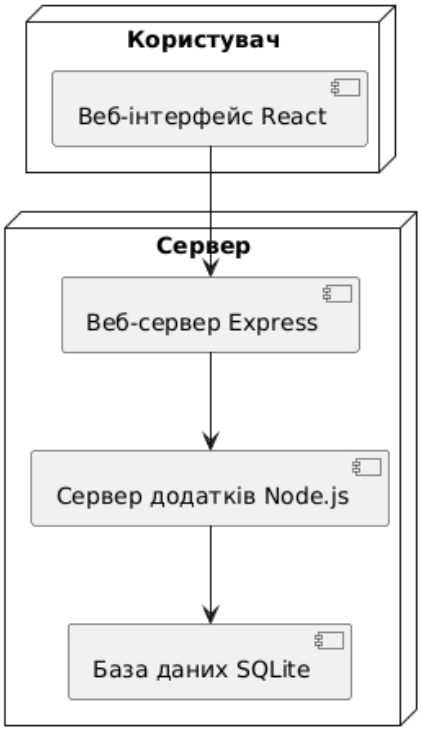

Рис. 4.16. Діаграма розгортання

Така архітектура є оптимальною для невеликих і середніх книжкових магазинів із кількістю користувачів до 1000. Вона проста у розгортанні та економічна, оскільки не потребує складних серверних конфігурацій. Проте при зростанні навантаження SQLite може стати "вузьким місцем" через обмеження на одночасні записи, що потребуватиме переходу на розподілену систему.

4.3 Вимоги до апаратного та програмного забезпечення

Для успішного впровадження та експлуатації інформаційної системи для продажу книжкової продукції, розробленої на основі лістингу програми, необхідно чітко визначити вимоги до апаратного та програмного забезпечення. Ці вимоги є ключовими для забезпечення стабільної роботи системи, її масштабованості, безпеки та зручності використання як для користувачів (клієнтів), так і для адміністраторів (менеджерів). У цьому розділі детально розглядаються мінімальні та рекомендовані характеристики апаратного забезпечення, а також необхідне програмне забезпечення для серверної та клієнтської частин системи.

4.3.1. Апаратне забезпечення

Інформаційна система базується на клієнт-серверній архітектурі, де серверна частина відповідає за обробку запитів, зберігання даних у базі даних SQLite та забезпечення API для взаємодії з клієнтською частиною. Для стабільної роботи серверної частини необхідне апаратне забезпечення, яке відповідає вимогам, описаним в таблиці 4.2.

Таблиця 4.2

Вимоги до апаратного забезпечення для серверної частини

№	Назва	Мінімальні вимоги	Рекомендовані вимоги	Обґрунтування
1	2	3	4	5
1	Процесор	Двоядерний процесор із частотою 2.0 ГГц (наприклад, Intel Core i3 або еквівалент AMD)	Чотириядерний процесор із частотою 3.0 ГГц або вище (наприклад, Intel Core i5/i7 або AMD Ryzen 5/7). Це забезпечить швидшу обробку запитів, особливо при одночасному доступі великої кількості користувачів	Система використовує Node.js (з бібліотекою Express) для обробки HTTP-запитів, що є відносно легким фреймворком, але при зростанні навантаження (наприклад, обробка замовлень або пошук книг) багатопоточність і продуктивність процесора стають важливими
2	Оперативна пам’ять (RAM)	4 ГБ	8 ГБ або більше	База даних SQLite працює в оперативній пам’яті для швидкого доступу до даних, а Node.js потребує додаткових ресурсів для обробки асинхронних операцій. При інтенсивному використанні (наприклад, одночасна робота кількох менеджерів і клієнтів) обсяг пам’яті впливає на швидкодію
3	Накопичувач	20 ГБ вільного місця на жорсткому диску (HDD)	SSD-накопичувач обсягом від 50 ГБ	SQLite зберігає всі дані в одному файлі (bookstore.db), і швидкість доступу до нього значно вища на SSD порівняно з HDD. З урахуванням потенційного зростання бази даних (книги, замовлення, користувачі) рекомендується передбачити запас місця
4	Мережеве обладнання	Підключення до мережі зі швидкістю 10 Мбіт/с	Швидкість 100 Мбіт/с або вище (Ethernet або Wi-Fi)	Система працює через API, тому стабільне та швидке з’єднання необхідне для передачі даних між

				сервером і клієнтами, особливо при завантаженні зображень книг або обробці великих замовлень
5	Резервне живлення	-	Джерело безперебійного живлення (ДБЖ) із потужністю від 500 Вт для захисту від збоїв електроживлення	У разі відключення електроенергії ДБЖ забезпечить коректне завершення роботи сервера та збереження даних у базі

Клієнтська частина системи реалізована як вебдодаток на базі React і доступна через браузер. Вимоги до апаратного забезпечення для користувачів (клієнтів і менеджерів) є менш жорсткими, але все ж потребують уваги (таблиця 4.3).

Таблиця 4.3

Вимоги до апаратного забезпечення для клієнтської частини

№	Назва	Мінімальні вимоги	Рекомендовані вимоги	Обґрунтування
1	2	3	4	5
1	Процесор	Одноядерний процесор із частотою 1.5 ГГц	Двоядерний процесор із частотою 2.0 ГГц або вище	React використовує JavaScript для динамічного оновлення інтерфейсу, що потребує певної обчислювальної потужності, особливо при відображенні великих списків книг або анімацій
2	Оперативна пам’ять (RAM)	2 ГБ	4 ГБ або більше	Сучасні браузери (наприклад, Chrome, Firefox) споживають значний обсяг пам’яті, особливо при відкритих кількох вкладках. Для комфортної роботи рекомендується більший обсяг RAM
3	Екран	Роздільна здатність 1024x768 пікселів	Роздільна здатність 1366x768 пікселів або вище	Адаптивний дизайн системи (CSS із медіа-запитами) підтримує різні розміри екранів, але для оптимального відображення всіх елементів

				інтерфейсу потрібна достатня роздільна здатність
4	Інтернет-з’єднання	Швидкість 2 Мбіт/с	Швидкість 10 Мбіт/с або вище	Завантаження сторінок, зображень книг і відправка запитів до сервера потребують стабільного з’єднання. Вища швидкість покращить користувацький досвід

4.3.2. Програмне забезпечення

Для роботи серверної частини системи необхідно встановити та налаштувати таке програмне забезпечення, описане в таблиці 4.4.

Таблиця 4.4

Вимоги до програмного забезпечення для серверної частини

№	Назва	Мінімальні вимоги	Рекомендовані вимоги	Обґрунтування
1	2	3	4	5
1	Операційна система	Windows Server 2016, Ubuntu 18.04 LTS або інша Linux-дистрибуція	Ubuntu 20.04 LTS або Windows Server 2019	Node.js і SQLite сумісні з усіма основними ОС, але Linux-дистрибутиви (наприклад, Ubuntu) є кращими для серверів через стабільність і меншу ресурсоємність
2	Середовище виконання	Node.js версії 14.x або новіше (рекомендується LTS-версія, наприклад, 16.x або 18.x)	-	Серверна логіка написана на JavaScript із використанням Express, що потребує Node.js як середовища виконання
3	База даних	SQLite 3.x (вбудована через бібліотеку better-sqlite3)	-	SQLite обрано як легку та просту у використанні базу даних, що не потребує окремого серверного процесу, але підходить лише для невеликих і середніх навантажень

4	Додаткові бібліотеки	Встановлення залежностей через npm (express, body-parser, better-sqlite3 тощо), як зазначено у файлі package.json	-	Ці бібліотеки необхідні для роботи API, обробки запитів і взаємодії з базою даних
5	Система контролю версій (опціонально)	-	Git для управління кодом і розгортання оновлень	Дозволяє відстежувати зміни в коді та швидко розгортати нові версії системи

Для доступу до системи з боку користувачів потрібне програмне забезпечення, описане в таблиці 4.5.

Таблиця 4.5

Вимоги до програмного забезпечення для клієнтської частини

№	Назва	Мінімальні вимоги	Рекомендовані вимоги	Обґрунтування
1	2	3	4	5
1	Веббраузер	Google Chrome 80+, Mozilla Firefox 75+, Microsoft Edge 80+, Safari 13+	Останні версії Chrome або Firefox	React використовує сучасні стандарти JavaScript (ES6+), які підтримуються в актуальних версіях браузерів. Старіші версії можуть некоректно відображати інтерфейс

Продовження таблиці 4.5

1	2	3	4	5
2	Операційна система	Windows 7, macOS 10.12, Ubuntu 16.04 або будь-яка сучасна ОС для мобільних пристроїв (Android 9+, iOS 13+)	Windows 10/11, macOS 11+, Ubuntu 20.04+.	Адаптивний дизайн системи дозволяє працювати на різних пристроях, але новіші ОС забезпечують кращу сумісність із браузерами
3	Антивірусне ПЗ (опціонально)	-	Встановлення антивірусного програмного забезпечення для захисту від шкідливого коду	Захищає користувачів від потенційних загроз під час роботи в Інтернеті

Вимоги до апаратного та програмного забезпечення для інформаційної системи продажу книжкової продукції враховують її архітектуру, обсяг даних і потенційне навантаження. Для серверної частини ключовими є продуктивність процесора, обсяг оперативної пам’яті та швидкість накопичувача, тоді як для клієнтської частини важливими є сучасний браузер і стабільне інтернет-з’єднання. Дотримання цих вимог забезпечить ефективну роботу системи, її доступність для користувачів і можливість масштабування в майбутньому. При впровадженні рекомендується провести тестування на відповідність зазначеним характеристикам, щоб уникнути проблем із продуктивністю чи сумісністю.

4.4 Опис роботи програмного забезпечення

Програмне забезпечення інформаційної системи для продажу книжкової продукції, представлене у коді програми, є комплексним рішенням, розробленим

для автоматизації процесів онлайн-продажу книг. Воно базується на сучасних технологіях веб-розробки, зокрема React для клієнтської частини, Node.js із фреймворком Express для серверної частини та SQLite як системи управління базами даних. Система забезпечує зручний інтерфейс для користувачів (покупців) та менеджерів, а також підтримує ключові функції електронної комерції, такі як перегляд асортименту, пошук книг, оформлення замовлень, управління кошиком та адміністрування. Необхідно детально розглянути опис роботи програмного забезпечення з урахуванням його структури та функціональних можливостей.

Програмне забезпечення складається з трьох основних компонентів:

– Клієнтська частина (app.js): Реалізована за допомогою бібліотеки React, яка забезпечує динамічний та інтерактивний інтерфейс користувача. Основний компонент App управляє станом додатку, включаючи дані про користувача, книги, кошик, обране, замовлення тощо. Використовуються хуки React (useState, useEffect) для управління станом та асинхронними операціями.

– Серверна частина (server.js): Побудована на базі Node.js із використанням Express для обробки HTTP-запитів. Сервер забезпечує API-ендпоінти для взаємодії з базою даних, автентифікації, управління книгами та замовленнями. SQLite використовується як легка та ефективна база даних.

– Скрипт ініціалізації даних (seed.js): Служить для початкового заповнення бази даних тестовими даними, включаючи користувачів, книги та замовлення, що полегшує тестування та демонстрацію системи.

Основні функції та їх реалізація описані в таблиці 4.6.

Таблиця 4.6

Основні функції та їх реалізація

№	Функції	Реалізація
1	2	3
1	Автентифікація та реєстрація користувачів	Користувачі можуть увійти в систему через сторінку LoginPage або зареєструватися через RegisterPage. Функції login та register у компоненті App відправляють POST-запити до серверних ендпоінтів /api/login та /api/register. Успішна автентифікація зберігає дані користувача в локальному сховищі (localStorage), що дозволяє зберігати сесію між перезавантаженнями сторінки
		Сервер перевіряє введені дані (email та пароль) у базі даних SQLite і повертає відповідь із інформацією про користувача або повідомлення про помилку
2	Перегляд асортименту та пошук книг	Головна сторінка (HomePage) відображає список книг, отриманих через запит до /api/books. Книги групуються за категоріями, які динамічно формуються на основі унікальних значень поля category у базі даних. Користувач може фільтрувати книги за категоріями, переглядати деталі книги (BookDetailsPage) або додавати їх до кошика чи обраного
		Функція пошуку реалізована через компонент Header, де введений запит обробляється функцією searchBooks. Вона фільтрує книги за назвою або автором і відображає результати на сторінці SearchPage
3	Управління кошиком	Кошик (CartPage) дозволяє користувачу додавати книги (addToCart), видаляти їх (removeFromCart) або змінювати кількість (updateCartItemQuantity). Дані кошика зберігаються в стані React та синхронізуються з localStorage, що забезпечує збереження вмісту кошика навіть після закриття браузера
		Загальна сума замовлення обчислюється динамічно на основі цін та кількості товарів

Продовження таблиці 4.6

1	2	3
4	Оформлення замовлення	Сторінка CheckoutPage збирає дані про доставку та оплату, які валідуються перед відправленням. Після підтвердження замовлення функція createOrder відправляє POST-запит до /api/orders, передаючи інформацію про користувача, товари, доставку та оплату. Успішне оформлення очищає кошик і перенаправляє користувача на сторінку підтвердження (OrderConfirmationPage)
		На сервері замовлення записується в таблиці orders та order_items, а баланс менеджера оновлюється в таблиці manager_account
5	Управління для менеджерів	Менеджери мають доступ до додаткових сторінок: ManageBooksPage для додавання, редагування та видалення книг, а також ManageOrdersPage для перегляду та оновлення статусів замовлень. Ці функції реалізовані через запити до ендпоінтів /api/books (POST, PUT, DELETE) та /api/orders (GET, PUT)
		Наприклад, функція addBook додає нову книгу до бази даних, а updateOrderStatus змінює статус замовлення, що відображається в реальному часі
6	Сповіщення	Система сповіщень реалізована через функцію showNotification, яка відображає повідомлення (успіх, помилка, інформація) у правому нижньому куті екрана. Сповіщення автоматично зникають через 3 секунди завдяки setTimeout

Взаємодія з базою даних описана в таблиці 4.7.

Таблиця 4.7

Взаємодія з базою даних

№	Функції	Реалізація
1	2	3
1	Завантаження даних	при вході користувача або зміні стану (наприклад, вибір ролі менеджера) викликаються функції fetchBooks, fetchOrders, fetchManagerAccount, які отримують дані з бази через API-запити

Продовження таблиці 4.7

1	2	3
2	Збереження даних	дані користувача, кошика, обраного та переглянутих книг зберігаються в localStorage для забезпечення безперервного досвіду. Замовлення та зміни в асортименті записуються безпосередньо в базу даних SQLite
3	Транзакції	сервер використовує транзакції для створення замовлень, що гарантує цілісність даних у разі помилок (наприклад, відкат при збої)

Інтерфейс побудований за принципом SPA (Single Page Application), де перемикання між сторінками (home, cart, checkout тощо) відбувається без перезавантаження завдяки функції setPage. Кожен компонент (Header, Footer, BookCard) є модульним і повторно використовуваним, що спрощує підтримку та розширення системи. Стилі CSS адаптовані для різних розмірів екранів, забезпечуючи коректне відображення на десктопах і мобільних пристроях.

Безпека та стабільність описані в таблиці 4.8.

Таблиця 4.8

Безпека та стабільність

№	Напрям	Опис
1	Обробка помилок	Усі асинхронні операції (наприклад, запити до API) включають блоки try-catch, які відображають користувачу повідомлення про помилки через систему сповіщень
2	Валідація	Форми автентифікації, реєстрації та оформлення замовлення містять перевірки введених даних (наприклад, формат email, довжина пароля, номер картки), що запобігає некоректним операціям
3	Ролі користувачів	Система розрізняє ролі (customer, manager), обмежуючи доступ до адміністративних функцій для звичайних користувачів

Програмне забезпечення є повнофункціональним рішенням для онлайн-продажу книг, яке поєднує зручність для користувачів із ефективним управлінням для менеджерів. Його модульна структура, інтеграція з базою даних

та адаптивний інтерфейс роблять систему гнучкою для подальшого розвитку, наприклад, додавання нових функцій (рекомендації, відгуки) чи інтеграції з платіжними системами. Реалізація на основі React та Node.js забезпечує високу продуктивність і масштабованість, що є важливим для комерційного використання.

ВИСНОВКИ

Розробка інформаційної системи для продажу книгарної продукції, представлена в даній роботі, є важливим кроком у напрямі автоматизації та оптимізації процесів електронної комерції у сфері книжкової торгівлі. Сучасний розвиток інформаційних технологій, зростання популярності онлайн-покупок та потреба у швидкому доступі до широкого асортименту літератури підкреслюють актуальність створення такого програмного забезпечення. Ця система спрямована на забезпечення зручності для покупців, ефективного управління асортиментом і замовленнями для менеджерів, а також підвищення конкурентоспроможності книжкового магазину в умовах цифрової трансформації.

Метою роботи було створення функціонального інструменту, який би відповідав потребам як клієнтів, так і працівників магазину, забезпечуючи швидкий доступ до каталогу книг, спрощення оформлення замовлень, управління запасами та надання аналітичних даних. У процесі реалізації використано сучасні технології, такі як React.js для створення інтерактивного інтерфейсу, Node.js з Express.js для серверної логіки та SQLite як легку й ефективну базу даних. Застосування об’єктно-орієнтованого аналізу та проєктування, зокрема побудова ER-діаграм, діаграм класів і компонентів, дозволило чітко структурувати систему та забезпечити її логічну цілісність. Асинхронне програмування та методи тестування сприяли оптимізації взаємодії між клієнтом і сервером, а також підвищенню надійності продукту.

Розроблена система охоплює ключові аспекти предметної області книжкової торгівлі, включаючи каталог книг з детальною інформацією, кошик покупок, оформлення замовлень, персоналізацію для користувачів через функції обраного та історії переглядів, а також адміністративні можливості для менеджерів. Локальне зберігання даних у LocalStorage забезпечує зручність користувацького досвіду, дозволяючи зберігати вибір навіть після закриття

браузера, тоді як адаптивний дизайн гарантує комфортне використання на різних пристроях. Проведений аналіз вимог показав, що система відповідає основним функціональним і нефункціональним характеристикам, таким як швидкість обробки запитів, зручність інтерфейсу та базова безпека, хоча в майбутньому потребує вдосконалення, зокрема в аспектах масштабованості та захисту даних через впровадження хешування паролів і шифрування.

Таким чином, створена інформаційна система є комплексним рішенням, яке успішно інтегрує торговельні, управлінські та інформаційні процеси, сприяючи спрощенню доступу до літератури та підвищенню ефективності роботи книжкового магазину. Використання сучасних методів і технологій забезпечило її відповідність актуальним потребам ринку, а гнучка архітектура відкриває можливості для подальшого вдосконалення, наприклад, через додавання відгуків користувачів, інтеграцію з платіжними шлюзами чи розширення рекомендаційних алгоритмів. Ця робота не лише демонструє практичну реалізацію теоретичних знань, а й вносить внесок у розвиток електронної комерції в Україні, сприяючи популяризації читання через зручний онлайн-доступ до книг.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Фронтенд і бекенд: дві сторони однієї медалі [Електронний ресурс]. Посилання: https://voll.com.ua/uk/blog/frontend-i-bekend-dve-storony-odnoj-medali

2. Що повинен знати FrontEnd розробник у 2024 році [Електронний ресурс]. Посилання: https://itvdn.com/ua/blog/article/frontend-developer-tech-skills

3. Аутентифікація і авторизація: що це і в чому відмінність [Електронний ресурс]. Посилання: https://qagroup.com.ua/publications/autentyfikatciia-i-avtoryzatciia/

4. Як правильно організувати каталог інтернет-магазину [Електронний ресурс]. Посилання: https://wezom.com.ua/ua/blog/kak-pravilno-organizovat-katalog-internet-magazina

5. Функції кошика в інтернет-магазині [Електронний ресурс]. Посилання: https://simferopil.rozrobka-sajtiv.in.ua/rozrobka-sajtiv-blog/funkcziyi-koshyka-v-internet-magazyni/

6. Контент-менеджмент та адміністрування сайтів [Електронний ресурс]. Посилання: https://scratch-studio.com/kontent-menedzhment-ta-administruvannia-saitiv/

7. Сповіщення користувачів [Електронний ресурс]. Посилання: https://esputnik.com/uk/blog/sho-take-web-push

8. Оптимізація продуктивності сайту: Поради та методи [Електронний ресурс]. Посилання: https://www.ranktracker.com/uk/blog/optimizing-website-performance-tips-and-techniques/

9. Масштабованість як вимога до програмного забезпечення, значення та визначення [Електронний ресурс]. Посилання: https://uk.itpedia.nl/2021/07/20/schaalbaarheid-als-software-requirement-betekenis-en-definitie/

10. Принципи юзабіліті веб-сайту [Електронний ресурс]. Посилання: https://wezom.com.ua/ua/blog/12-sposobov-uluchshit-juzabiliti

11. Безпека Даних на Сайті: Кращі Практики Маркетингових Агентств для Наслідування [Електронний ресурс]. Посилання: https://protocol.ua/ua/bezpeka_danih_na_sayti_krashchi_praktiki_marketingovih_agentstv_dlya_nasliduvannya/

12. React The library for web and native user interfaces [Електронний ресурс]. Посилання: https://react.dev/

13. Детальний огляд та розбір Node.js [Електронний ресурс]. Посилання: https://wezom.com.ua/ua/blog/vse-chto-nuzhno-znat-o-nodejs

14. Що таке SQlite? [Електронний ресурс]. Посилання: https://freehost.com.ua/ukr/faq/wiki/chto-takoe-sqlite/

15. LocalStorage, sessionStorage [Електронний ресурс]. Посилання: https://uk.javascript.info/localstorage

16. Що таке CRUD простими словами: функції, переваги та приклади [Електронний ресурс]. Посилання: https://highload.tech/uk/shho-take-crud-prostimi-slovami-funktsiyi-perevagi-ta-prikladi/

17. Система навігації сайту та особливості її створення [Електронний ресурс]. Посилання: https://gl.ua/blog/systema-navihatsiyi-saytu-ta-osoblyvosti-yiyi-stvorennya

18. ІДЕНТИФІКАЦІЯ СУТНОСТІ ІНФОРМАЦІЙНОЇ СИСТЕМИ ОБЛІКУ ТА ЇЇ ОСНОВНІ ХАРАКТЕРНІ РИСИ [Електронний ресурс]. Посилання: https://core.ac.uk/outputs/268453013/

19. 2.3.2. Визначення взаємозв'язків між сутностями [Електронний ресурс]. Посилання: https://studfile.net/preview/7078141/page:4/

20. Предметна область, моделювання предметної області [Електронний ресурс]. Посилання: https://studfile.net/preview/5474325/page:2/

21. Amazon [Електронний ресурс]. Посилання: https://www.amazon.com/gp/browse.html/ref=glow_cls?node=283155&nodl=0&ref=books_dsk_sn_books-logo-1b731

22. Yakaboo [Електронний ресурс]. Посилання: https://www.yakaboo.ua/

23. AbeBooks [Електронний ресурс]. Посилання: https://www.abebooks.com/

24. What is an Entity Relationship Diagram (ERD)? [Електронний ресурс]. Посилання: https://www.lucidchart.com/pages/er-diagrams

25. Що таке діаграма класів UML і найкращий творець діаграм класів UML [Електронний ресурс]. Посилання: https://www.mindonmap.com/uk/blog/what-is-uml-class-diagram/

26. Діаграма кооперацій [Електронний ресурс]. Посилання: https://studfile.net/preview/5010027/page:4/

ДОДАТОК А

ЛІСТИНГ ПРОГРАМИ

app.js

```
// Сторінка реєстрації
const RegisterPage = ({ register, setPage }) => {
    const [email, setEmail] = React.useState('');
    const [password, setPassword] = React.useState('');
    const [confirmPassword, setConfirmPassword] = React.useState('');
    const [name, setName] = React.useState('');
    const [role, setRole] = React.useState('customer');
    const [errors, setErrors] = React.useState({});
    const validateForm = () => {
        const newErrors = {};
        if (!name.trim()) {
            newErrors.name = 'Введіть ваше ім\'я';
        }
        if (!email.trim()) {
            newErrors.email = 'Введіть ваш email';
        } else if (!/\S+@\S+\.\S+/.test(email)) {
            newErrors.email = 'Введіть коректний email';
        }
        if (!password) {
            newErrors.password = 'Введіть пароль';
        } else if (password.length < 6) {
            newErrors.password = 'Пароль повинен містити не менше 6 символів';
        }
        if (password !== confirmPassword) {
            newErrors.confirmPassword = 'Паролі не співпадають';
        }
        setErrors(newErrors);
        return Object.keys(newErrors).length === 0;
    };
    const handleSubmit = (e) => {
        e.preventDefault();
        if (validateForm()) {
            register(email, password, name, role);
        }
    };
        return (
        <div className="auth-page">
            <h2>Реєстрація</h2>
            <form className="auth-form" onSubmit={handleSubmit}>
                <div className="form-group">
                    <label htmlFor="name">Ім'я</label>
                    <input
                        type="text"
                        id="name"
                        value={name}
                        onChange={(e) => setName(e.target.value)}
                        className={errors.name ? 'error' : ''}
                    />
                    {errors.name && <span className="error-
message">{errors.name}</span>}
                </div>
                <div className="form-group">
                    <label htmlFor="email">Email</label>
                    <input
                        type="email"
                        id="email"
                        value={email}
                        onChange={(e) => setEmail(e.target.value)}
                        className={errors.email ? 'error' : ''}
                    />
                    {errors.email && <span className="error-
message">{errors.email}</span>}
```

```
                </div>
                <div className="form-group">
                    <label htmlFor="password">Пароль</label>
                    <input
                        type="password"
                        id="password"
                        value={password}
                        onChange={(e) => setPassword(e.target.value)}
                        className={errors.password ? 'error' : ''}
                    />
                    {errors.password && <span className="error-
message">{errors.password}</span>}
                </div>
                <div className="form-group">
                    <label htmlFor="confirmPassword">Підтвердження паролю</label>
                    <input
                        type="password"
                        id="confirmPassword"
                        value={confirmPassword}
                        onChange={(e) => setConfirmPassword(e.target.value)}
                        className={errors.confirmPassword ? 'error' : ''}
                    />
                    {errors.confirmPassword && <span className="error-
message">{errors.confirmPassword}</span>}
                </div>
                <div className="form-group">
                    <label htmlFor="role">Роль</label>
                    <select
                        id="role"
                        value={role}
                        onChange={(e) => setRole(e.target.value)}
                    >
                        <option value="customer">Покупець</option>
                        <option value="manager">Менеджер</option>
                    </select>
                </div>
                <button type="submit" className="btn-submit">Зареєструватися</button>
            </form>
            <p className="auth-link">
                Вже зареєстровані? <span onClick={() =>
setPage('login')}>Увійти</span>
            </p>
        </div>
    );
};


// Сторінка оформлення замовлення
const CheckoutPage = ({ cart, createOrder, user }) => {
    const [shippingDetails, setShippingDetails] = React.useState({
        fullName: (user && user.name) || '',
        address: '',
        city: '',
        postalCode: '',
        phone: ''
    });
    const [paymentDetails, setPaymentDetails] = React.useState({
        cardNumber: '',
        cardHolder: '',
        expiryDate: '',
        cvv: ''
    });
    const [errors, setErrors] = React.useState({});
```

```
    const totalAmount = cart.reduce((total, item) => total + item.price *
item.quantity, 0);
    const validateForm = () => {
        const newErrors = {};
        // Перевірка даних доставки
        if (!shippingDetails.fullName.trim()) {
            newErrors.fullName = 'Введіть ваше ім\'я';
        }
        if (!shippingDetails.address.trim()) {
            newErrors.address = 'Введіть адресу доставки';
        }
        if (!shippingDetails.city.trim()) {
            newErrors.city = 'Введіть місто';
        }
        if (!shippingDetails.postalCode.trim()) {
            newErrors.postalCode = 'Введіть поштовий індекс';
        }
        if (!shippingDetails.phone.trim()) {
            newErrors.phone = 'Введіть номер телефону';
        } else if (!/^\+?\d{10,12}$/.test(shippingDetails.phone.replace(/\s/g, ''))) {
            newErrors.phone = 'Введіть коректний номер телефону';
        }
        // Перевірка платіжних даних
        if (!paymentDetails.cardNumber.trim()) {
            newErrors.cardNumber = 'Введіть номер картки';
        } else if (!/^\d{16}$/.test(paymentDetails.cardNumber.replace(/\s/g, ''))) {
            newErrors.cardNumber = 'Номер картки повинен містити 16 цифр';
        }
        if (!paymentDetails.cardHolder.trim()) {
            newErrors.cardHolder = 'Введіть ім\'я власника картки';
        }
        if (!paymentDetails.expiryDate.trim()) {
            newErrors.expiryDate = 'Введіть термін дії картки';
        } else if (!/^\d{2}\/\d{2}$/.test(paymentDetails.expiryDate)) {
            newErrors.expiryDate = 'Використовуйте формат MM/PP';
        }
        if (!paymentDetails.cvv.trim()) {
            newErrors.cvv = 'Введіть CVV код';
        } else if (!/^\d{3}$/.test(paymentDetails.cvv)) {
            newErrors.cvv = 'CVV код повинен містити 3 цифри';
        }
        setErrors(newErrors);
        return Object.keys(newErrors).length === 0;
    };
    const handleShippingChange = (e) => {
        const { name, value } = e.target;
        setShippingDetails({ ...shippingDetails, [name]: value });
    };
    const handlePaymentChange = (e) => {
        const { name, value } = e.target;
        setPaymentDetails({ ...paymentDetails, [name]: value });
    };
    const handleSubmit = async (e) => {
        e.preventDefault();
        if (validateForm()) {
            const orderData = {
                userId: user.id,
                items: cart.map(item => ({
                    id: item.id,
                    title: item.title,
                    price: item.price,
                    quantity: item.quantity
                })),
                                totalAmount,
```

```
                shipping: shippingDetails,
                payment: {
                    method: 'card',
                    lastFour: paymentDetails.cardNumber.slice(-4)
                },
            };
            const order = await createOrder(orderData);
        }
    };
    return (
        <div className="checkout-page">
            <h2>Оформлення замовлення</h2>
            <div className="checkout-container">
                <div className="checkout-form">
                    <form onSubmit={handleSubmit}>
                        <div className="form-section">
                            <h3>Дані доставки</h3>
                            <div className="form-group">
                                <label htmlFor="fullName">Повне ім'я</label>
                                <input
                                    type="text"
                                    id="fullName"
                                    name="fullName"
                                    value={shippingDetails.fullName}
                                    onChange={handleShippingChange}
                                    className={errors.fullName ? 'error' : ''}
                                />
                                {errors.fullName && <span className="error-
message">{errors.fullName}</span>}
                            </div>
                            <div className="form-group">
                                <label htmlFor="address">Адреса</label>
                                <input
                                    type="text"
                                    id="address"
                                    name="address"
                                    value={shippingDetails.address}
                                    onChange={handleShippingChange}
                                    className={errors.address ? 'error' : ''}
                                />
                                {errors.address && <span className="error-
message">{errors.address}</span>}
                            </div>
                            <div className="form-row">
                                <div className="form-group">
                                    <label htmlFor="city">Місто</label>
                                    <input
                                        type="text"
                                        id="city"
                                        name="city"
                                        value={shippingDetails.city}
                                        onChange={handleShippingChange}
                                        className={errors.city ? 'error' : ''}
                                    />
                                    {errors.city && <span className="error-
message">{errors.city}</span>}
                                </div>
                                <div className="form-group">
                                    <label htmlFor="postalCode">Поштовий
індекс</label>
                                    <input
                                        type="text"
                                        id="postalCode"
                                        name="postalCode"
```

```
                                        value={shippingDetails.postalCode}
                                        onChange={handleShippingChange}
                                        className={errors.postalCode ? 'error' : ''}
                                    />
                                    {errors.postalCode && <span className="error-
message">{errors.postalCode}</span>}
                                </div>
                            </div>
                            <div className="form-group">
                                <label htmlFor="phone">Телефон</label>
                                <input
                                    type="text"
                                    id="phone"
                                    name="phone"
                                    value={shippingDetails.phone}
                                    onChange={handleShippingChange}
                                    className={errors.phone ? 'error' : ''}
                                />
                                {errors.phone && <span className="error-
message">{errors.phone}</span>}
                            </div>
                        </div>
                        <div className="form-section">
                            <h3>Дані оплати</h3>
                            <div className="form-group">
                                <label htmlFor="cardNumber">Номер картки</label>
                                <input
                                    type="text"
                                    id="cardNumber"
                                    name="cardNumber"
                                    placeholder="1234 5678 9012 3456"
                                    value={paymentDetails.cardNumber}
                                    onChange={handlePaymentChange}
                                    className={errors.cardNumber ? 'error' : ''}
                                />
                                {errors.cardNumber && <span className="error-
message">{errors.cardNumber}</span>}
                            </div>
                            <div className="form-group">
                                <label htmlFor="cardHolder">Власник картки</label>
                                <input
                                    type="text"
                                    id="cardHolder"
                                    name="cardHolder"
                                    placeholder="IVAN IVANENKO"
                                    value={paymentDetails.cardHolder}
                                    onChange={handlePaymentChange}
                                    className={errors.cardHolder ? 'error' : ''}
                                />
                                {errors.cardHolder && <span className="error-
message">{errors.cardHolder}</span>}
                            </div>
                            <div className="form-row">
                                <div className="form-group">
                                    <label htmlFor="expiryDate">Термін дії</label>
                                    <input
                                        type="text"
                                        id="expiryDate"
                                        name="expiryDate"
                                        placeholder="MM/PP"
                                        value={paymentDetails.expiryDate}
                                        onChange={handlePaymentChange}
                                        className={errors.expiryDate ? 'error' : ''}
                                    />
```

```
                                    {errors.expiryDate && <span className="error-
message">{errors.expiryDate}</span>}
                                </div>
                                <div className="form-group">
                                    <label htmlFor="cvv">CVV</label>
                                    <input
                                        type="text"
                                        id="cvv"
                                        name="cvv"
                                        placeholder="123"
                                        value={paymentDetails.cvv}
                                        onChange={handlePaymentChange}
                                        className={errors.cvv ? 'error' : ''}
                                    />
                                    {errors.cvv && <span className="error-
message">{errors.cvv}</span>}
                                </div>
                            </div>
                        </div>
                        <button type="submit" className="btn-submit">
                            Підтвердити замовлення
                        </button>
                    </form>
                </div>
                <div className="order-summary">
                    <h3>Ваше замовлення</h3>
                    <div className="order-items">
                        {cart.map(item => (
                            <div className="order-item" key={item.id}>
                                <span>{item.title} x {item.quantity}</span>
                                <span>{item.price * item.quantity} грн</span>
                            </div>
                        ))}
                    </div>
                    <div className="order-total">
                        <span>Загалом:</span>
                        <span>{totalAmount} грн</span>
                    </div>
                </div>
            </div>
        </div>
    );
};
// Сторінка результатів пошуку
const SearchPage = ({ results, query, viewBook, addToCart, toggleFavorite, favorites
}) => {
    if (results.length === 0) {
        return (
            <div className="empty-search">
                <h2>Результатів не знайдено</h2>
                <p>Ми не знайшли книг за запитом "{query}"</p>
                <button className="btn-primary" onClick={() => setPage('home')}>
                    Повернутися до магазину
                </button>
            </div>
        );
    }
    return (
        <div className="search-page">
            <h2>Результати пошуку для "{query}"</h2>
            <div className="books-grid">
                {results.map(book => (
                    <BookCard
                        key={book.id}
```

```
                        book={book}
                        viewBook={viewBook}
                        addToCart={addToCart}
                        toggleFavorite={toggleFavorite}
                        isFavorite={favorites.some(item => item.id === book.id)}
                    />
                ))}
            </div>
        </div>
    );
};
```

server.js

```
// server.js
const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const Database = require('better-sqlite3');
const app = express();
const port = process.env.PORT || 5000;
const db = new Database('bookstore.db');
app.use(bodyParser.json());
app.use(express.static(path.join(__dirname, 'public')));
// Створення таблиць у БД
function initDB() {
    db.exec(`
    CREATE TABLE IF NOT EXISTS users (
      id INTEGER PRIMARY KEY AUTOINCREMENT,
      email TEXT UNIQUE,
      password TEXT,
      role TEXT,
      name TEXT
    );
    CREATE TABLE IF NOT EXISTS books (
      id INTEGER PRIMARY KEY AUTOINCREMENT,
      title TEXT,
      author TEXT,
      category TEXT,
      price REAL,
      description TEXT,
      image TEXT,
      inStock INTEGER
    );
    CREATE TABLE IF NOT EXISTS orders (
      id INTEGER PRIMARY KEY AUTOINCREMENT,
      userId INTEGER,
      totalAmount REAL,
      shipping TEXT,
      payment TEXT,
      status TEXT,
      date TEXT,
      FOREIGN KEY (userId) REFERENCES users(id)
    );
    CREATE TABLE IF NOT EXISTS order_items (
      id INTEGER PRIMARY KEY AUTOINCREMENT,
      orderId INTEGER,
      bookId INTEGER,
      title TEXT,
      price REAL,
      quantity INTEGER,
      FOREIGN KEY (orderId) REFERENCES orders(id),
      FOREIGN KEY (bookId) REFERENCES books(id)
    );
    CREATE TABLE IF NOT EXISTS manager_account (
```

```
      id INTEGER PRIMARY KEY AUTOINCREMENT,
      balance REAL
    );
  `);
    // Перевірка наявності запису балансу менеджера
    const managerAccount = db.prepare('SELECT * FROM manager_account LIMIT 1').get();
    if (!managerAccount) {
        db.prepare('INSERT INTO manager_account (balance) VALUES (0)').run();
    }
}
// Ініціалізація БД при запуску сервера
initDB();
// API для авторизації та реєстрації
app.post('/api/login', (req, res) => {
    const { email, password } = req.body;
    const user = db.prepare('SELECT * FROM users WHERE email = ? AND password =
?').get(email, password);
    if (user) {
        res.json({
            success: true,
            user: { id: user.id, email: user.email, role: user.role, name: user.name }
        });
    } else {
        res.status(401).json({ success: false, message: 'Невірний email або пароль'
});
    }
});
app.post('/api/register', (req, res) => {
    const { email, password, name, role = 'customer' } = req.body;
    try {
        const existingUser = db.prepare('SELECT * FROM users WHERE email =
?').get(email);
        if (existingUser) {
            return res.status(400).json({ success: false, message: 'Користувач з таким
email вже існує' });
        }
        const result = db.prepare('INSERT INTO users (email, password, role, name)
VALUES (?, ?, ?, ?)').run(email, password, role, name);
        const userId = result.lastInsertRowid;
        res.json({
            success: true,
            user: { id: userId, email, role, name }
        });
    } catch (error) {
        res.status(500).json({ success: false, message: 'Помилка сервера' });
    }
});
```