НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри
_____комп'ютерних наук_____
(назва кафедри)

_____ /_____Голуб Б.Л._____/
(підпис) (ПІБ)

"___" _____ 2025 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА
на тему

«Програмне забезпечення системи управління інтернет магазину»

Спеціальність 121 – «Інженерія програмного забезпечення»

Гарант освітньої програми
_____к.т.н, доцент_____ _____ _____Вайганг Г.О._____
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Керівник бакалаврської кваліфікаційної роботи
___асистент _____ _____ _Баранова Т. А._
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Консультант бакалаврської кваліфікаційної роботи
___к.т.н, доцент _____ _____ _ Даков С.Ю.
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ _____Олефіренко Іван Валерійович__
(підпис) (ПІБ студента)

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри
Комп'ютерних наук
_____ Голуб Б.Л.

"___" _____ 2025 р.

ЗАВДАННЯ
на виконання бакалаврської кваліфікаційної роботи студенту
Олефіренку Івану Валерійовичу
(прізвище, ім'я, по батькові)

Спеціальність 121 – «Інженерія програмного забезпечення»

Тема бакалаврської кваліфікаційної роботи Програмне забезпечення системи управління інтернет магазину
затверджена наказом ректора НУБіП України від "16"грудня 2024 р. №2248 "с"
Термін подання завершеної роботи на кафедру _____
(рік, місяць, число)

Вихідні дані до бакалаврської кваліфікаційної роботи
_____
Перелік питань, які потрібно розробити: _____
Перелік графічних документів (за потреби)
Дата видачі завдання "16" грудня 2024 р.
Керівник бакалаврської кваліфікаційної роботи
_____ _____ Баранова Т.А.
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Консультант бакалаврської кваліфікаційної роботи

к.т.н, доцент _____ _____ Даков С.Ю.
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Завдання прийняв до виконання _____ Олефіренко Іван Валерійович
(підпис) (ПІБ студента)

Зміст

ВСТУП

У сучасному світі електронна комерція стала невід'ємною частиною економічного розвитку, а інтернет-магазини є ключовим інструментом для забезпечення доступу до товарів та послуг у цифровому просторі. Розробка програмного забезпечення системи управління інтернет-магазином передбачає створення комплексного рішення, яке охоплює функціонал для управління каталогом товарів, обробки замовлень, автентифікації користувачів та забезпечення зручного інтерфейсу. У рамках цього проєкту представлено результат реалізації проєкту – два основні компоненти: клієнтська частина (client.py), побудована на базі Streamlit, та серверна частина (server.py), реалізована з використанням Flask. Ці компоненти разом формують функціональну систему управління книжковим інтернет-магазином.

Актуальність теми зумовлена стрімким зростанням популярності онлайн-торгівлі, що вимагає створення ефективних, безпечних та масштабованих програмних рішень. Системи управління інтернет-магазинами повинні відповідати сучасним вимогам, таким як швидка обробка запитів, захист даних користувачів та інтуїтивно зрозумілий інтерфейс. Дослідження таких систем дозволяє не лише вдосконалювати технології електронної комерції, а й адаптувати їх до потреб малого та середнього бізнесу, зокрема у сфері продажу книг, де конкуренція вимагає високої якості сервісу.

Метою роботи є розробка та аналіз програмного забезпечення системи управління інтернет-магазином, яке забезпечує зручність використання для клієнтів і менеджерів, а також демонструє практичну реалізацію ключових функцій електронної комерції на прикладі книжкового магазину.

Об'єктом роботи є система управління інтернет-магазином, що включає клієнтську та серверну частини, призначені для організації торгівлі книгами в онлайн-середовищі.

Основними **завданнями** роботи є:

– Розробка клієнтської частини системи з використанням бібліотеки Streamlit для створення інтерактивного вебінтерфейсу.

– Реалізація серверної частини на базі Flask із застосуванням RESTful API для обробки запитів та взаємодії з базою даних.

– Забезпечення базового функціоналу: автентифікація користувачів, управління каталогом книг, обробка замовлень.

– Аналіз структури та можливостей створеного програмного забезпечення з точки зору його практичного застосування.

Предметом роботи є програмне забезпечення, представлене у вигляді клієнтського (client.py) та серверного (server.py) кодів, їх архітектура, функціональні можливості та взаємодія між компонентами.

У роботі використано такі **методи** дослідження:

– Аналітичний метод – для вивчення вимог до систем управління інтернет-магазинами та аналізу існуючих рішень.

– Метод програмування – для створення клієнтської та серверної частин із застосуванням Python, Streamlit, Flask та SQLite.

– Тестування – для перевірки працездатності розробленого програмного забезпечення та оцінки його відповідності поставленим завданням.

Розроблене програмне забезпечення має **практичне значення** як прототип системи управління інтернет-магазином, який може бути адаптований для реального використання у сфері онлайн-торгівлі книгами. Воно демонструє базові принципи створення таких систем і може слугувати основою для подальшого вдосконалення, наприклад, додавання модулів оплати, аналітики продажів чи інтеграції з зовнішніми сервісами. Результати роботи також можуть бути використані у навчальних цілях для вивчення технологій розробки вебдодатків.

Ключові слова: Інтернет-магазин, програмне забезпечення, система управління, Streamlit, Flask, RESTful API, автентифікація, каталог товарів, замовлення, SQLite, електронна комерція.

РОЗДІЛ 1. СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Предметна область проєкту охоплює сферу управління інтернет-магазином книжкової продукції через програмну систему, побудовану на базі сучасних технологій веб-розробки та баз даних. У сучасних умовах розвитку електронної комерції книжковий ринок потребує ефективних рішень для автоматизації продажів, забезпечення зручності користувачів та оптимізації операційних процесів [1]. Реалізовані коди програм (client.py, server.py) є результатом реалізації такої системи, яка забезпечує онлайн-торгівлю книгами з урахуванням потреб різних груп користувачів.

Основна мета предметної області полягає в організації взаємодії між клієнтами (покупцями), менеджерами та системою [2]. Клієнти можуть переглядати каталог книг, фільтрувати їх за категоріями, додавати до кошика, оформлювати замовлення та переглядати їхній статус. Менеджери відповідають за оновлення асортименту, обробку замовлень та управління складом. Система забезпечує зберігання даних, захист інформації та зручний інтерфейс для всіх учасників процесу. Таким чином, предметна область об’єднує комерційні й управлінські показники діяльності книжкового інтернет-магазину.

Ключові об’єкти предметної області представлені в таблиці 1.1.

Таблиця 1.1

Ключові об’єкти предметної області

№	Назва	Опис
1	**2**	**3**
1	Книги	Основний товар із атрибутами: назва, автор, категорія, ціна, опис, кількість на складі, зображення.

2	Користувачі	Поділяються на клієнтів і менеджерів із даними: ім’я, пароль, email, роль, адреса, телефон.
3	Замовлення	Містять дані про клієнта, перелік книг, суму, дату, статус (наприклад, «нове», «доставлене»).

Продовження таблиці 1.1

1	2	3
4	Кошик	Тимчасове сховище для обраних книг перед оформленням замовлення.
5	Категорії	Групування книг за жанрами чи тематикою для зручності пошуку.

Система функціонує завдяки інтеграції фронтенду (client.py на базі Streamlit), бекенду (server.py на Flask) та бази даних SQLite. Фронтенд забезпечує інтерактивний інтерфейс із можливістю перегляду книг, управління кошиком та профілем [3]. Бекенд обробляє запити, авторизацію, замовлення й асортимент, а база даних зберігає інформацію про книги, користувачів та замовлення [4, 5]. Основні процеси описано в таблиці 1.2.

Таблиця 1.2

Основні процеси предметної області

№	Назва	Опис
1	Реєстрація та авторизація	Створення профілю з роллю (клієнт/менеджер) і вхід у систему через токен.
2	Перегляд і вибір книг	Фільтрація за категоріями чи пошук із детальним описом кожної книги.
3	Формування замовлення	Додавання книг до кошика, вибір кількості та підтвердження замовлення.
4	Обробка замовлень	Зміна статусу менеджером, оновлення складських залишків.
5	Управління асортиментом	Додавання, редагування чи видалення книг із каталогу менеджерами.

Система враховує зовнішні фактори, такі як попит на літературу, логістичні компоненти та адаптацію до різних пристроїв. Використання Streamlit забезпечує адаптивність інтерфейсу, а дебаг-панель у client.py підвищує зручність розробки й тестування. Отже, предметна область є комплексною системою, що оптимізує торгівлю книгами, забезпечує зручність для користувачів і підвищує ефективність управління магазином. Аналіз цієї області визначає вимоги до програмного забезпечення: інтуїтивність, швидкодія, безпека даних і гнучкість функціоналу.

1.2 Аналіз вимог до програмної системи

Розробка програмного забезпечення системи управління інтернет-магазином книжкової продукції потребує детального аналізу вимог, які забезпечують її функціональність, зручність для користувачів і відповідність цілям предметної області. На основі реалізованих кодів можна визначити ключові вимоги до системи, враховуючи потреби двох основних груп користувачів — клієнтів і менеджерів, а також технічні елементи реалізації. Аналіз вимог базується на функціональних можливостях, нефункціональних характеристиках, технічних особливостях та інтерфейсі користувача.

Функціональні вимоги наведені в таблиці 1.3.

Таблиця 1.3

Функціональні вимоги

№	Вимоги	Пояснення
1	2	3
1	Реєстрація та автентифікація користувачів	Система повинна дозволяти створювати нові облікові записи та автентифікувати користувачів [6]. У client.py функції show_register_form та show_login_form реалізують введення даних (username, password, email, role) і вибір ролі (client/manager), а в server.py маршрути /api/register і /api/login обробляють ці запити з генерацією JWT-токенів. Вимога включає захист даних через унікальність username та email, що реалізовано в базі SQLite.
2	Перегляд каталогу та пошук книг	Користувачі повинні мати доступ до каталогу книг із фільтрацією за категоріями та пошуком за назвою чи автором [7]. У client.py функція show_catalog відображає книги через get_books з підтримкою параметрів category і search, а server.py обробляє запит /api/books із відповідними SQL-фільтрами. Це забезпечує зручний доступ до асортименту.

Продовження таблиці 3.1

1	**2**	3
3	Кошик і оформлення замовлень	Клієнти можуть додавати книги до кошика, змінювати кількість і оформляти замовлення [8]. У client.py функція show_cart керує кошиком (add_to_cart, update_cart_quantity), а create_order відправляє дані на сервер через /api/orders. Server.py перевіряє наявність книг на складі та оновлює stock, забезпечуючи точність транзакцій.
4	Управління профілем	Користувачі повинні переглядати та редагувати свої дані [9]. Функція show_profile у client.py відображає профіль через get_user_profile і дозволяє оновлення через update_user_profile, що інтегрується з маршрутом /api/user/profile у server.py. Це забезпечує персоналізацію та актуальність даних.
5	Адміністрування для менеджерів	Менеджери мають додавати, редагувати, видаляти книги та керувати статусами замовлень [10]. У client.py функції show_manage_books і show_manage_orders реалізують ці можливості через API-запити (/api/books, /api/orders), а server.py обмежує доступ через декоратори @manager_required. Вимога — контроль асортименту та замовлень.
6	Перегляд історії замовлень	Користувачі можуть бачити свої замовлення, а менеджери — усі [11]. У client.py show_orders і show_order_details відображають дані через get_orders і get_order, а server.py підтримує це через /api/orders із розмежуванням доступу за ролями. Це забезпечує прозорість операцій.

Нефункціональні вимоги показані в таблиці 1.4.

Таблиця 1.4

Нефункціональні вимоги

№	**Вимоги**	**Пояснення**
1	**2**	**3**
1	Продуктивність	Система має швидко обробляти запити (каталог, пошук, замовлення) [12]. У client.py використовуються асинхронні запити через requests, а в server.py — оптимізовані SQL-запити. Час відгуку не повинен перевищувати 2-3 секунди при середньому навантаженні.

Продовження таблиці 1.4

1	2	3
2	Масштабованість	Програмне забезпечення має підтримувати зростання кількості користувачів і товарів [13]. SQLite у server.py підходить для початкового етапу, але для масштабування потрібен перехід на СУБД типу MySQL або PostgreSQL.
3	Зручність використання	Інтерфейс має бути інтуїтивним [14]. У client.py Streamlit забезпечує простий дизайн із чіткими формами та кнопками (наприклад, st.button, st.selectbox), адаптований для обох ролей користувачів.
4	Надійність	Система повинна зберігати дані при збоях [15]. Server.py використовує транзакції для замовлень (conn.commit(), conn.rollback()), але потребує резервного копіювання для повної надійності.
5	Безпека	Дані користувачів мають бути захищені [15]. У server.py паролі хешуються через hash_password, а доступ контролюється JWT-токенами. Однак шифрування чутливих даних (адреса, телефон) ще не реалізовано.

Технічні вимоги наведені в таблиці 1.5.

Таблиця 1.5

Технічні вимоги

№	Вимоги	Пояснення
1	Технологічний стек	Система базується на Python: Streamlit для фронтенду [16], Flask для бекенду [17] та SQLite як СУБД [18]. Client.py використовує st.session_state для стану, а server.py — RESTful API з маршрутами типу /api/books. Це забезпечує простоту розробки та розгортання.
2	Локальне зберігання стану	Для збереження кошика та сторінок між оновленнями використовується st.session_state у client.py [19]. Це підвищує швидкість і зручність роботи без перезавантаження.
3	Інтеграція з базою даних	Server.py реалізує CRUD-операції для книг, замовлень і користувачів через SQLite [20]. Початкове створення таблиць виконується в init_db, що дозволяє швидко налаштувати систему.

Вимоги до інтерфейсу користувача показані в таблиці 1.6.

Таблиця 1.6

Вимоги до інтерфейсу користувача

№	Вимоги	Пояснення
1	Навігація	Інтерфейс включає бічну панель із меню (st.sidebar.radio) для переходу між розділами (каталог, кошик, профіль) [21]. У client.py це реалізовано через set_page, що забезпечує швидкий доступ до функцій.
2	Відображення даних	Книги показані картками (show_catalog) із назвою, автором, ціною, а деталі — розширено (show_book_details). Замовлення відображаються списком із статусами (show_orders), що полегшує контроль.

Аналіз вимог до системи управління інтернет-магазином показав, що вона поєднує широкий функціонал (каталог, кошик, адміністрування) із базовими нефункціональними характеристиками (швидкість, зручність). На основі кодів видно реалізацію більшості вимог, але безпека та масштабування потребують доопрацювання. Ці вимоги є основою для подальшого вдосконалення системи відповідно до потреб книжкової торгівлі.

1.3 Моделювання предметної області

Моделювання предметної області є важливим етапом системного аналізу, який дозволяє формалізувати структуру, взаємозв’язки та процеси в межах програмного забезпечення системи управління інтернет-магазином книжкової продукції. Цей процес спрямований на створення абстрактного представлення системи, що відображає ключові об’єкти, їхні атрибути, зв’язки та поведінку, базуючись на функціональних вимогах та потребах користувачів — клієнтів і менеджерів. У контексті кодів проєкту (client.py та server.py) моделювання предметної області ґрунтується на аналізі управління асортиментом книг, обробки замовлень, автентифікації користувачів та взаємодії з базою даних через API. Основні етапи моделювання предметної області зображені на рис. 1.1.

Рис. 1.1. Основні етапи моделювання предметної області

Ідентифікація основних сутностей: предметна область інтернет-магазину книг включає кілька ключових сутностей, які відображені у структурі бази даних SQLite [22]. Перша сутність — Книга (Books) — характеризується атрибутами: ідентифікатор (id), назва (title), автор (author), опис (description), ціна (price), кількість на складі (stock), категорія (category) та URL зображення (image_url). Друга сутність — Користувач (Users) — охоплює ідентифікатор (id), ім’я користувача (username), пароль (password), електронну пошту (email), роль (role), повне ім’я (full_name), адресу (address) та телефон (phone). Третя сутність — Замовлення (Orders) — включає ідентифікатор (id), ідентифікатор користувача (user_id), дату замовлення (order_date), загальну суму (total_price) та статус (status). Пов’язана сутність Елемент замовлення (Order_items) деталізує замовлення через атрибути: ідентифікатор (id), ідентифікатор замовлення (order_id), ідентифікатор книги (book_id), кількість (quantity) та ціну (price). Ці сутності формують основу реляційної моделі даних, реалізованої у server.py.

Визначення зв’язків між сутностями: модель предметної області передбачає чіткі реляційні зв’язки між сутностями для забезпечення цілісності даних [23]. Сутність Користувач пов’язана із Замовленням через зовнішній ключ user_id, що реалізує зв’язок "один до багатьох", дозволяючи одному користувачу мати кілька замовлень. Сутність Замовлення пов’язана із Елементом замовлення через order_id (зв’язок "один до багатьох"), оскільки одне замовлення може містити кілька позицій книг, а Елемент замовлення пов’язаний із Книгою через book_id (зв’язок "багато до одного"), ідентифікуючи конкретну книгу. Ці відношення реалізовані у server.py через SQL-запити з використанням зовнішніх ключів, що забезпечують логічну організацію даних та контроль цілісності при операціях створення чи оновлення замовлень.

Моделювання процесів: предметна область включає динамічні процеси, які відображають функціональність системи та реалізовані у client.py через

інтерфейс Streamlit та API-запити до server.py [24]. Основні процеси охоплюють:

– Реєстрація та автентифікація користувачів — функції register_user і login_user у client.py надсилають запити до ендпоінтів /api/register і /api/login, де перевіряється унікальність даних і генерується JWT-токен;

– Перегляд і пошук книг — функція get_books із параметрами category та search дозволяє фільтрувати асортимент, відображаючи результати в каталозі;

– Управління кошиком — методи add_to_cart, remove_from_cart і update_cart_quantity керують списком покупок у сесійному стані;

– Оформлення замовлення — create_order надсилає дані до /api/orders, де оновлюється склад і записуються деталі замовлення;

– Управління асортиментом і замовленнями — функції add_book, update_book, delete_book і update_order_status доступні менеджерам через спеціальні сторінки. Ці процеси моделюють поведінку системи залежно від ролі користувача та інтегруються з серверною логікою.

Діаграма прецедентів (Use Case Diagram) (рис. 1.2) відображає взаємодію акторів (Клієнт, Менеджер) із системою [25]. Клієнт виконує дії "Перегляд каталогу", "Додавання до кошика", "Оформлення замовлення" та "Перегляд профілю", тоді як Менеджер додатково має прецеденти "Додавання книги" та "Оновлення статусу замовлення". Ця діаграма допомагає визначити основні сценарії використання, реалізовані у функціях client.py.

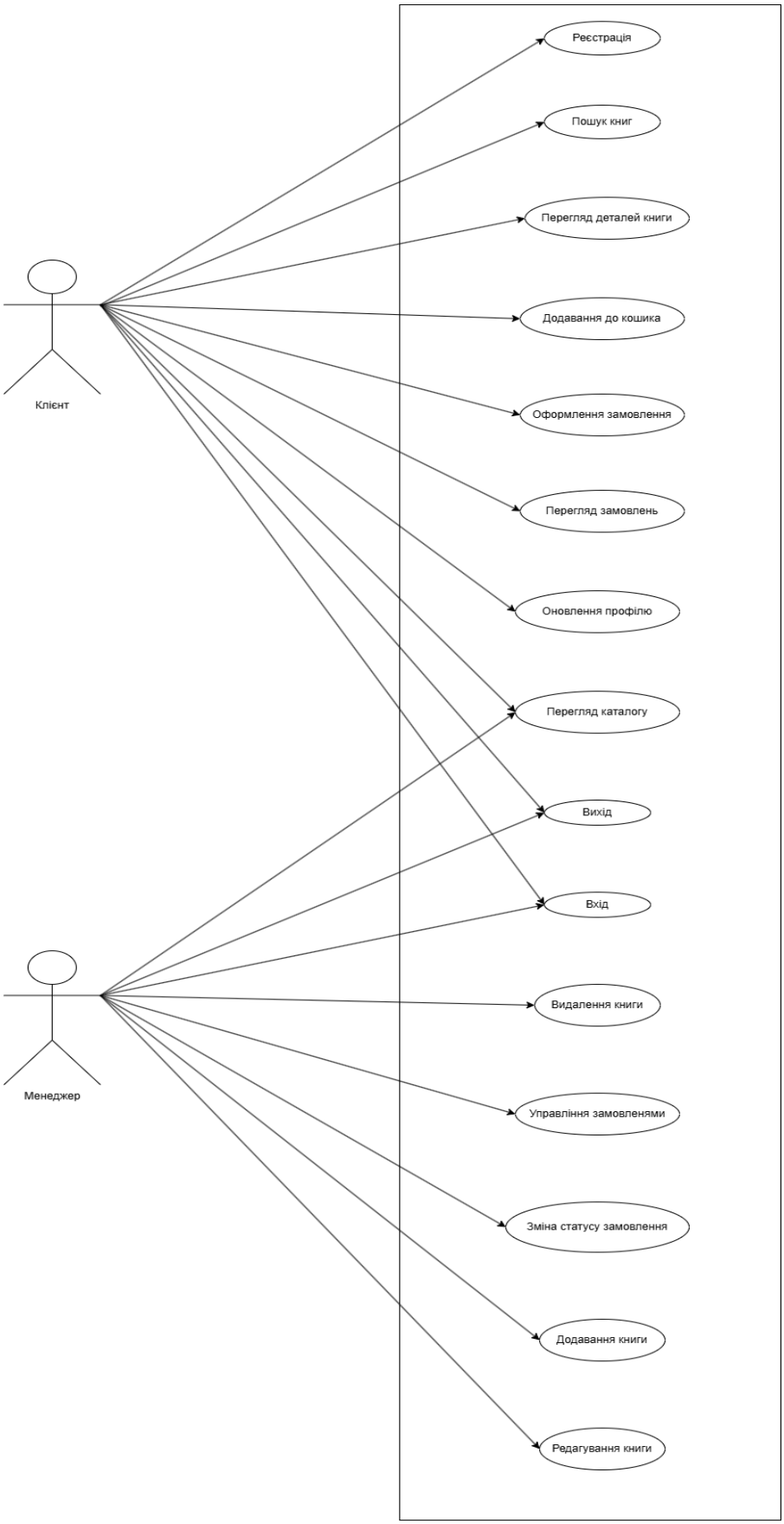

Рис. 1.2. Діаграма прецедентів

Діаграма послідовності (Sequence Diagram) (рис. 1.3) для процесу "Оформлення замовлення" показує взаємодію між Клієнтом, інтерфейсом Streamlit (client.py), API (server.py) та базою даних [26]. Клієнт викликає create_order, що ініціює POST-запит до /api/orders, сервер перевіряє наявність книг на складі й оновлює таблиці orders та order_items, повертаючи підтвердження. Це відображає синхронну обробку запитів у системі.

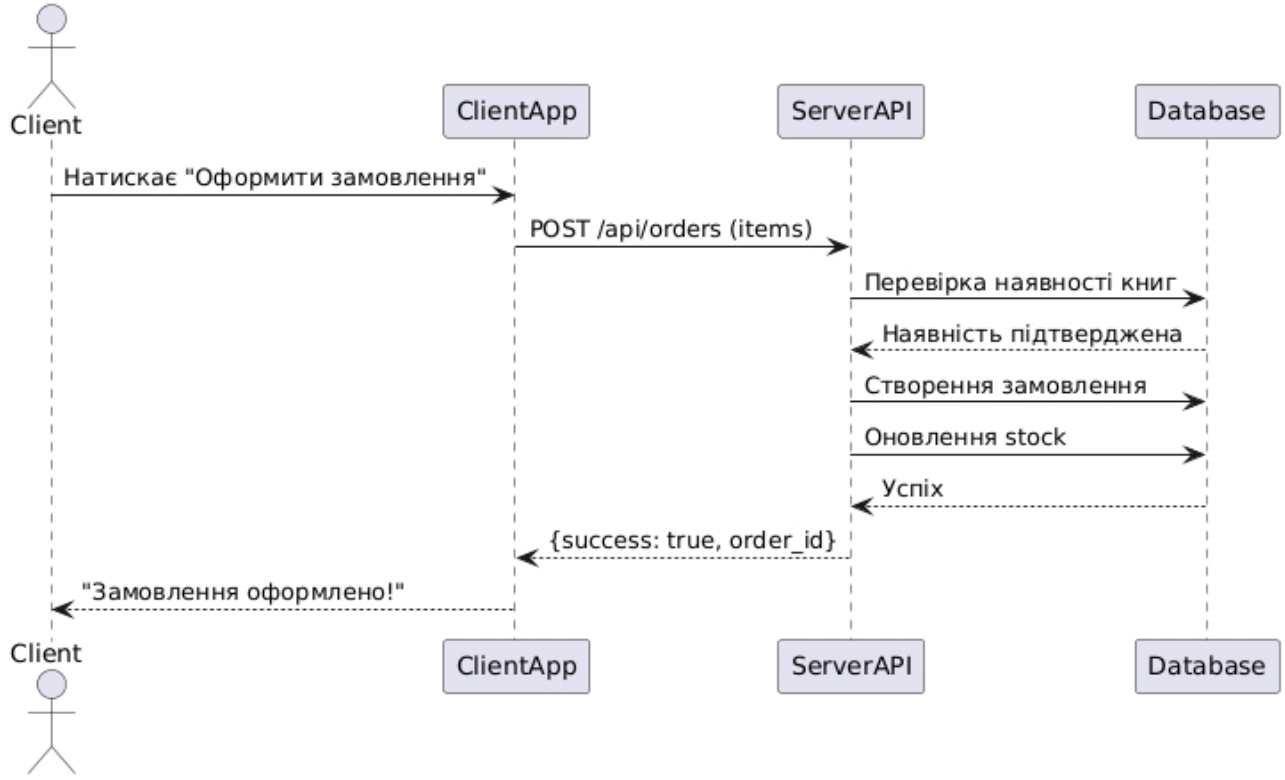

Рисунок 1.3. Діаграма послідовності

Для візуалізації предметної області використано діаграму "сутність-зв’язок" (ERD) (рис. 1.5), яка показує зв’язки між таблицями users, orders, order_items і books у базі SQLite [28]. Додатково поведінка замовлень моделюється через стани (нове → оброблене → відправлене → доставлене → скасоване), що контролюються менеджером через API-запити. Локальне

збереження кошика у st.session_state у client.py моделює тимчасові стани сесії користувача, підвищуючи зручність роботи із системою.

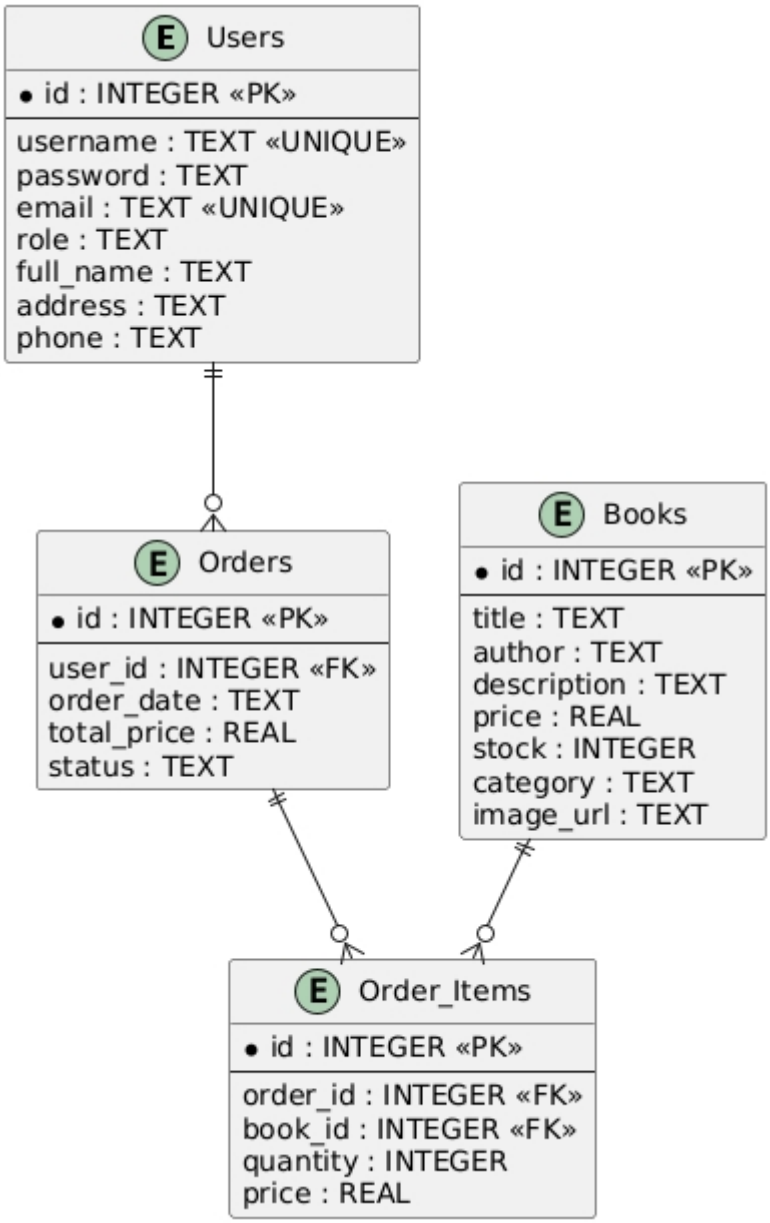

Рис. 1.5. ER-діаграма – логічна модель бази даних

Таким чином, моделювання предметної області для системи управління інтернет-магазином охоплює ідентифікацію сутностей (користувачі, книги, замовлення), визначення їхніх зв’язків і атрибутів, а також формалізацію процесів через функціональну логіку client.py і server.py. Цей підхід забезпечує

цілісність даних, підтримку ролей користувачів і адаптивність системи до потреб клієнтів і менеджерів, що є основою для її успішної реалізації.

1.4 Огляд інформаційних джерел та існуючих рішень

Розробка програмного забезпечення інформаційної системи для продажу книжкової продукції потребує глибокого розуміння предметної області, зокрема особливостей електронної комерції, потреб користувачів і сучасних технологічних підходів. Проведений аналіз інформаційних джерел та існуючих рішень дозволяє виявити актуальні тенденції, оцінити сильні та слабкі сторони аналогів, а також визначити напрями вдосконалення пропонованої системи. У цьому підрозділі розглянуто ключові літературні джерела, що стосуються розробки інформаційних систем для електронної торгівлі, а також проаналізовано приклади аналогічних рішень, які функціонують на ринку.

Laudon, K. C., & Traver, C. G. (2023). E-commerce: Business, Technology, Society є одним із фундаментальних джерел, що висвітлюють принципи побудови систем електронної комерції [29]. Автори детально аналізують архітектуру інформаційних систем, включаючи елементи управління контентом, обробки транзакцій та взаємодії з користувачами. Особливу увагу приділено інтеграції баз даних і фронтенд-компонентів, що є актуальним для реалізації книжкового онлайн-магазину, подібного до представленого у реалізованому коді (app.js, server.js). Джерело підкреслює важливість адаптивного дизайну та безпеки даних, що стало основою для розробки пропонованої системи.

Книга Іана Соммервілля Software Engineering є класичним посібником із програмної інженерії, де розглядаються методи системного аналізу та проектування інформаційних систем [30]. У контексті предметної області вона корисна для розуміння етапів створення ПЗ, таких як специфікація вимог, проектування архітектури та тестування. Ці принципи застосовані у проєкті, зокрема в організації бази даних (seed.js) та серверної логіки (server.js), що забезпечують стабільність і масштабованість системи.

Chaffey, D. Digital Business and E-commerce Management фокусується на управлінні цифровими бізнес-процесами в електронній комерції [31]. Автор аналізує сучасні платформи для онлайн-продажів, підкреслюючи важливість персоналізації, пошукових функцій і управління кошиком. У розробленій системі ці компоненти реалізовано через функції пошуку (searchBooks), кошика (addToCart) та персоналізованих рекомендацій (viewedItems), що відповідає сучасним стандартам.

Книга Мартіна Фаулера **Patterns of Enterprise Application Architecture** є важливим джерелом для розуміння архітектурних шаблонів у розробці корпоративних додатків [32]. У ній описано підходи до побудови модульних систем із чітким розподілом логіки між клієнтською та серверною частинами. Представлений код (app.js) використовує React для фронтенду та Express для бекенду, що відповідає шаблону MVC (Model-View-Controller), адаптованому до веб-додатків.

Rosenfeld, L., & Morville, P. Information Architecture for the World Wide Web. – цей ресурс присвячений архітектурі інформації у веб-системах [33]. Автори наголошують на важливості інтуїтивної навігації та структуризації контенту, що є ключовим для книжкових онлайн-магазинів. У проєкті це реалізовано через категорії книг (categories), фільтрацію та сторінку деталей книги (BookDetailsPage), що сприяє зручності користувацького досвіду.

Stair, R., & Reynolds, G. Principles of Information Systems розглядає основи інформаційних систем із акцентом на їхню роль у бізнес-процесах [34. Автори аналізують інтеграцію баз даних, обробку транзакцій і звітність, що є актуальним для менеджерських функцій у розробленій системі (manageOrders, managerAccount). Джерело підкреслює важливість автоматизації процесів, що відображено у функціях створення та оновлення замовлень (createOrder, updateOrderStatus).

Для оцінки конкурентного середовища розглянуто чотири популярні аналоги, які пропонують функціонал для продажу книжкової продукції. Аналіз

дозволяє порівняти їх із розробленою системою та визначити її унікальні особливості.

Amazon (рис. 1.2) є світовим лідером у сфері електронної комерції, зокрема у продажу книг [35]. 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.

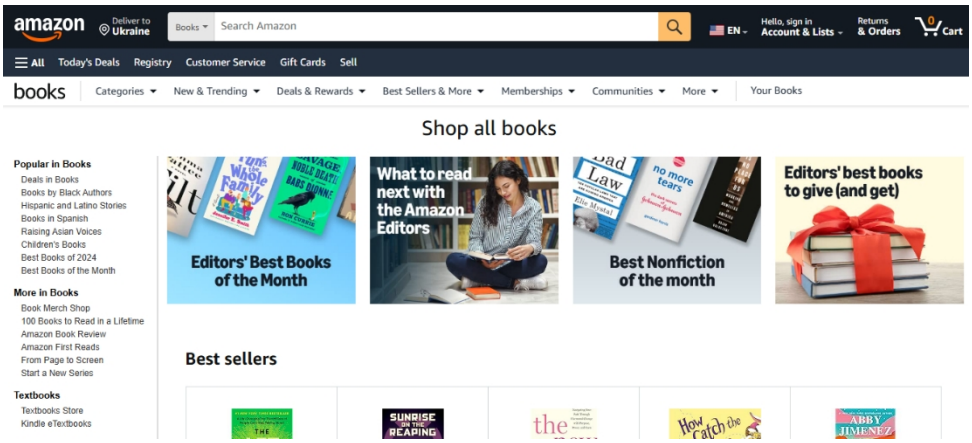

Рис. 1.2. Сторінка книжок на Amazon [35]

Yakaboo (рис. 1.3) – український онлайн-магазин книг із акцентом на локальний контент [36]. 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).

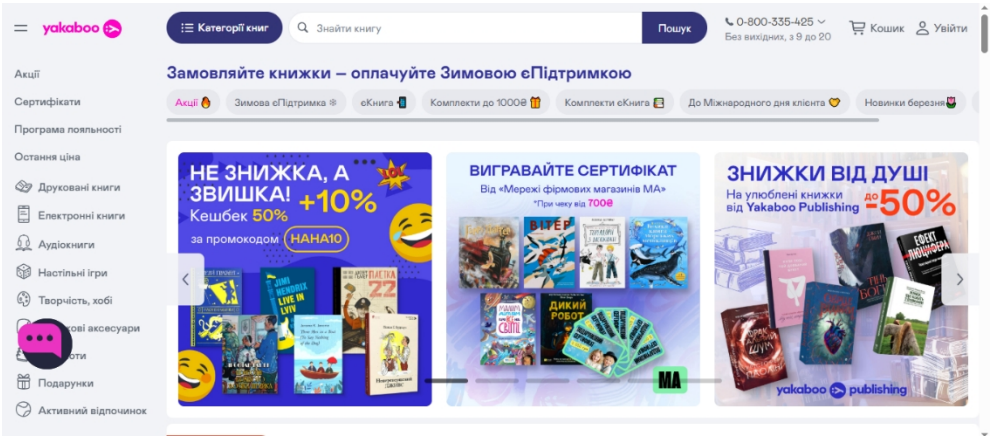

Рис. 1.3. Головна сторінка Yakaboo [36]

Сервіс **AbeBooks (рис. 1.4)** є глобальним онлайн-ринком для продажу книг, з акцентом на вживані, рідкісні та колекційні видання [37]. 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.

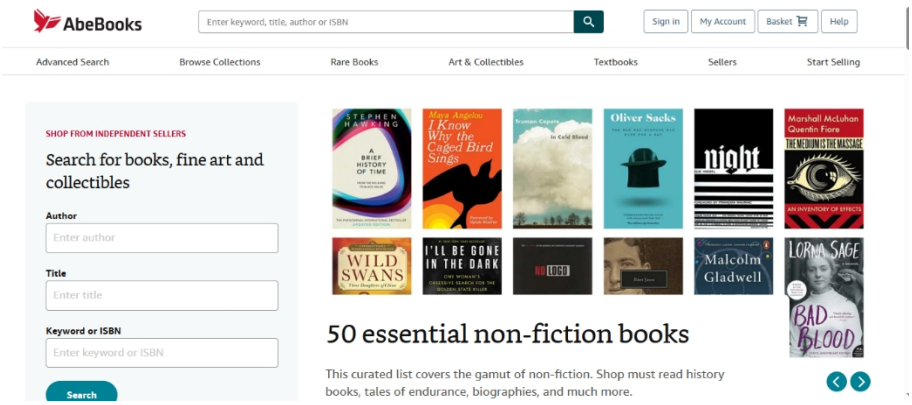

Рис. 1.4. Головна сторінка **AbeBooks [37]**

Booktopia (рис. 1.5) є провідним австралійським онлайн-магазином книг, що пропонує як нові, так і електронні книги з доставкою по всьому світу [38]. Він має зручний інтерфейс із категоріями, функцією пошуку та кошиком, а також підтримує знижки. Сильні сторони: великий вибір книг, підтримка місцевих авторів і видавців, швидка доставка в Австралії. Слабкі сторони: орієнтація переважно на австралійський ринок (інтерфейс англійською, валюта в австралійських доларах) і відсутність менеджерських інструментів для зовнішніх користувачів. Розроблена система виграє за рахунок локалізації для України (українська мова, гривні) та наявності адміністративних можливостей.

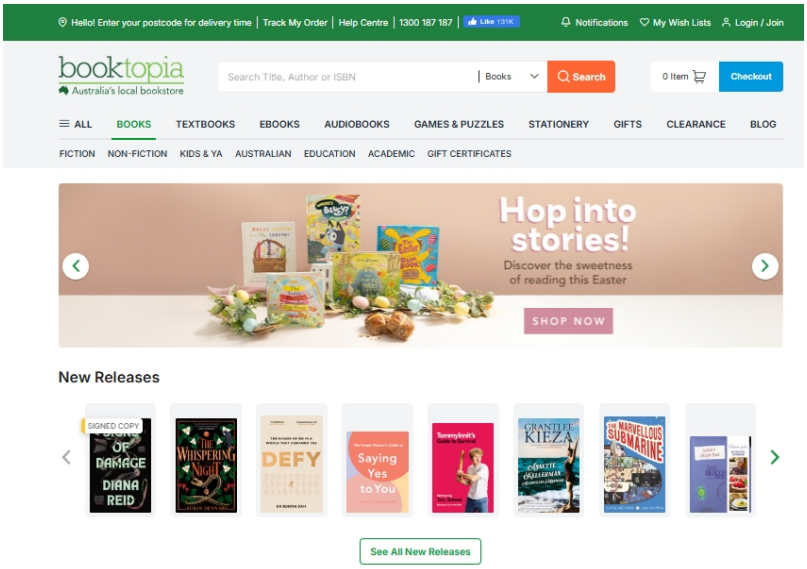

Рис. 1.5. Головна сторінка **Booktopia [38]**

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.

1.5 Постановка завдання

Сучасний розвиток інформаційних технологій створює нові перспективи для вдосконалення бізнес-процесів у сфері електронної комерції, зокрема в управлінні інтернет-магазинами книжкової продукції. Інформаційна система для онлайн-продажу книг є ключовим інструментом, який сприяє автоматизації торговельних операцій, підвищує зручність для користувачів, оптимізує управління асортиментом і замовленнями, а також зміцнює позиції магазину на ринку. У контексті предметної області — книжкової торгівлі — виникає потреба в розробці програмного забезпечення, яке враховує специфіку цього сегменту, включаючи різноманітність асортименту, особливості цільової аудиторії та вимоги до логістики. Метою даного розділу є чітка постановка завдання для створення інформаційної системи, яка задовольняє потреби покупців і менеджерів книжкового інтернет-магазину.

Предметна область охоплює функціонування онлайн-магазину книг, що передбачає продаж літератури різних категорій і форматів. Основними учасниками процесу є покупці (клієнти), менеджери (адміністратори) та система бази даних, яка забезпечує зберігання й обробку даних про книги, користувачів і замовлення. Специфіка книжкової торгівлі вимагає врахування таких показників, як: різноманітність літератури (художня, навчальна, дитяча тощо), деталізація характеристик книг (автор, назва, ціна, наявність), а також необхідність ефективного управління запасами й обробки замовлень. Важливими є також функції пошуку книг за параметрами (назва, автор, категорія) і забезпечення зручного інтерфейсу для користувачів.

Основна мета розробки інформаційної системи — створення програмного забезпечення для автоматизації продажу книг через онлайн-платформу. Система має забезпечити клієнтам зручний доступ до каталогу, можливість вибору книг і оформлення замовлень, а менеджерам — інструменти для управління асортиментом і замовленнями. Завдання полягає в розробці комплексного

рішення, яке інтегрує функції електронної комерції з внутрішніми процесами книжкового магазину.

Для реалізації мети необхідно виконати низку завдань, поділених на функціональні, технічні та організаційні категорії (таблиця 1.7).

Таблиця 1.7

Основні завдання проєкту

№	Завдання	Опис
1	Функціональні завдання	
1.1	Розробка модуля каталогу книг	Система має відображати список книг із деталями (назва, автор, ціна, категорія, наявність). Реалізовано пошук і фільтрацію за категоріями.
1.2	Реалізація кошика покупок	Користувачі можуть додавати книги до кошика, змінювати кількість або видаляти товари перед оформленням замовлення.
1.3	Оформлення замовлень	Модуль дозволяє вводити дані для доставки (ім’я, адреса, телефон) і обробляти замовлення зі зміною статусу (наприклад, «нове», «відправлене»).
1.4	Персоналізація для користувачів	Підтримка реєстрації, авторизації, збереження персональних даних і перегляду історії замовлень.
1.5	Адміністрування для менеджерів	Інтерфейс для додавання, редагування, видалення книг і управління статусами замовлень.
2	Технічні завдання	
2.1	Вибір технологічного стеку	Використано Streamlit для фронтенду, Flask для бекенду та SQLite для бази даних, що забезпечує простоту й ефективність.
2.2	Забезпечення асинхронної взаємодії	Реалізовано API-запити (GET, POST, PUT, DELETE) для обміну даними між клієнтом і сервером.
2.3	Безпека	Захист від базових загроз (хешування паролів, JWT-автентифікація) і перевірка прав доступу.
3	Організаційні завдання	
3.1	Тестування та відлагодження	Перевірка роботи кошика, замовлень і адмін-функцій із виправленням помилок.
3.2	Документація	Опис структури системи, API і базові інструкції для користувачів і менеджерів.

3.3	Розгортання	Локальний запуск із можливістю масштабування в майбутньому.

У результаті реалізації завдань створено інформаційну систему, яка включає веб-додаток із зручним інтерфейсом, серверну логіку для обробки запитів і базу даних для зберігання інформації. Система забезпечує доступ до каталогу, швидке оформлення замовлень і ефективне управління магазином, підвищуючи автоматизацію процесів і покращуючи користувацький досвід.

Обмеження проєкту: підтримка лише української мови, базова модель оплати (без реальних платіжних систем у прототипі) і локальне розгортання. Передбачається, що користувачі мають базові навички роботи з веб-додатками, а менеджери — мінімальні знання адміністрування.

Таким чином, постановка завдання визначає створення системи, яка враховує потреби всіх учасників процесу. Реалізований код є практичним втіленням цих вимог, що дозволяє перейти до подальшого аналізу й удосконалення.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних у вигляді ER-діаграми

Проєктування інформаційного та програмного забезпечення системи управління інтернет-магазином є ключовим етапом у створенні ефективного та функціонального продукту. У контексті розробки реалізованих кодів програм важливе місце посідає створення логічної моделі даних, яка забезпечує структуроване зберігання інформації та взаємодію між різними компонентами системи. Для цього використано підхід моделювання даних за допомогою ER-діаграми (Entity-Relationship Diagram) (рис. 2.1), що дозволяє візуально представити сутності, їх атрибути та зв’язки між ними [39]. Така модель є

основою для реалізації бази даних у серверній частині системи, яка в даному випадку базується на SQLite.

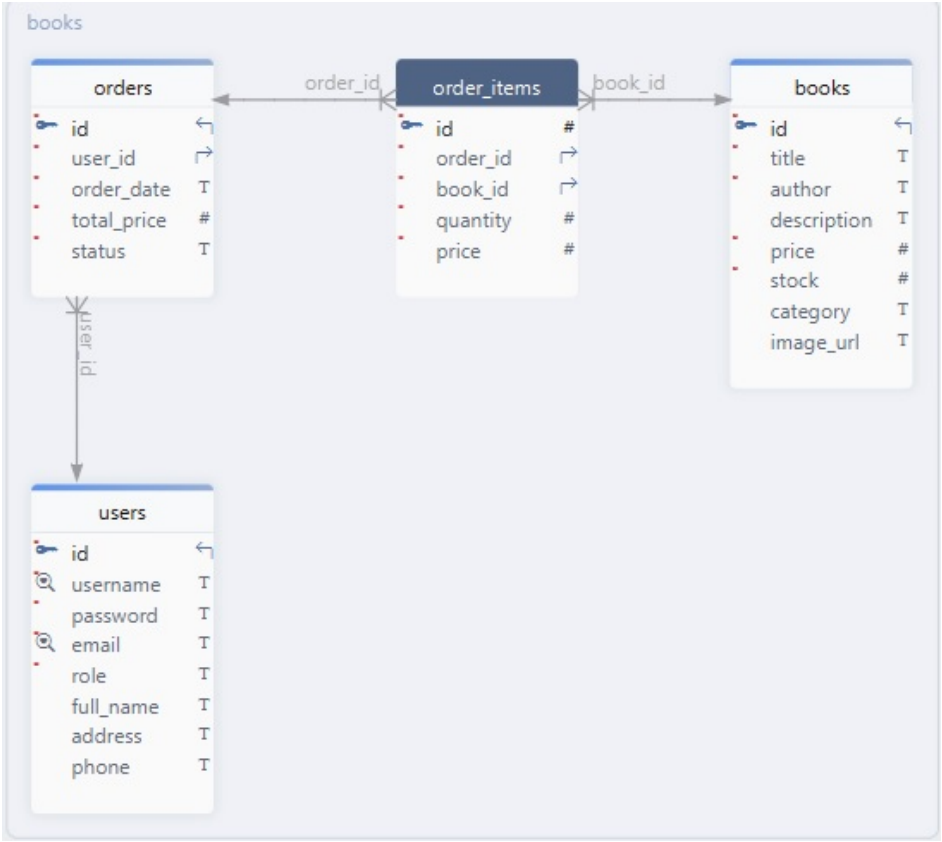

Рис. 2.1. ER-діаграма

Логічна модель даних системи управління інтернет-магазином складається з чотирьох основних сутностей: **Users** (Користувачі), **Books** (Книги), **Orders** (Замовлення) та **Order_Items** (Елементи замовлення). Кожна з цих сутностей має набір атрибутів, які відображають ключові характеристики об’єктів, а також визначені зв’язки, що забезпечують цілісність даних і логіку роботи системи.

Сутність "Users" представляє користувачів системи, які можуть мати різні ролі – клієнт ("client") або менеджер ("manager"). Основні атрибути цієї сутності включають:

– id (ціле число, первинний ключ) – унікальний ідентифікатор користувача;

– username (текст, унікальний) – ім’я користувача;

– password (текст) – хешований пароль для автентифікації;

– email (текст, унікальний) – електронна пошта;

– role (текст) – роль користувача;

– full_name (текст, необов’язковий) – повне ім’я;

– address (текст, необов’язковий) – адреса доставки;

– phone (текст, необов’язковий) – номер телефону.

Ця сутність є основою для авторизації та управління доступом у системі, а також пов’язана з сутністю "Orders" через зовнішній ключ.

Сутність "Books" відображає товари, доступні для продажу в інтернет-магазині. Її атрибути включають:

– id (ціле число, первинний ключ) – унікальний ідентифікатор книги;

– title (текст) – назва книги;

– author (текст) – автор книги;

– description (текст, необов’язковий) – опис книги;

– price (дійсне число) – ціна книги;

– stock (ціле число) – кількість одиниць на складі;

– category (текст, необов’язковий) – категорія книги;

– image_url (текст, необов’язковий) – посилання на зображення книги.

Ця сутність є центральною для каталогу товарів і пов’язана з сутністю "Order_Items" через зовнішній ключ.

Сутність "Orders" описує замовлення, зроблені користувачами. Атрибути:

– id (ціле число, первинний ключ) – унікальний ідентифікатор замовлення;

– user_id (ціле число, зовнішній ключ) – ідентифікатор користувача, який зробив замовлення;

– order_date (текст) – дата та час створення замовлення;

– total_price (дійсне число) – загальна сума замовлення;

– status (текст) – статус замовлення (наприклад, "нове", "оброблене", "доставлене").

Сутність "Orders" пов’язана з "Users" через атрибут user_id (зв’язок "один до багатьох") та з "Order_Items" через атрибут id (зв’язок "один до багатьох").

Ця сутність є допоміжною і відображає товари в межах конкретного замовлення. Атрибути:

– id (ціле число, первинний ключ) – унікальний ідентифікатор елемента;

– order_id (ціле число, зовнішній ключ) – ідентифікатор замовлення;

– book_id (ціле число, зовнішній ключ) – ідентифікатор книги;

– quantity (ціле число) – кількість одиниць книги в замовленні;

– price (дійсне число) – ціна одиниці товару на момент замовлення.

Сутність "Order_Items" забезпечує зв’язок "багато до багатьох" між "Orders" і "Books", дозволяючи зберігати деталі кожного замовлення.

Зв’язки між сутностями

– Між "Users" і "Orders" існує зв’язок "один до багатьох": один користувач може мати багато замовлень, але кожне замовлення належить лише одному користувачу.

– Між "Orders" і "Order_Items" також зв’язок "один до багатьох": одне замовлення може включати багато елементів, але кожен елемент належить лише одному замовленню.

– Між "Books" і "Order_Items" зв’язок "один до багатьох": одна книга може бути частиною багатьох елементів замовлення, але кожен елемент посилається лише на одну книгу.

Таким чином, ER-діаграма відображає логічну структуру даних, яка підтримує основні функції інтернет-магазину: управління користувачами, каталогом книг, кошиком та замовленнями. У реалізації серверної частини (server.py) ці сутності втілені у вигляді таблиць бази даних SQLite, а зв’язки забезпечуються через зовнішні ключі. Клієнтська частина (client.py), побудована на Streamlit, використовує цю модель для відображення даних і взаємодії з користувачем через API-запити. Такий підхід гарантує цілісність даних,

ефективну обробку запитів і гнучкість системи для подальшого розширення функціональності.

2.2 Діаграма класів

У процесі проєктування інформаційного та програмного забезпечення системи управління інтернет-магазином важливим етапом є створення діаграми класів предметної області та подальший аналіз кооперацій між ними. Цей підхід дозволяє чітко визначити основні сутності системи, їхні атрибути та взаємозв’язки, що є основою для реалізації функціоналу, представленого в реалізованих кодах.

Діаграма класів (рис. 2.2) предметної області відображає ключові сутності системи управління інтернет-магазином книг та асоціації між ними [40]. Основними класами є "Користувач" (User), "Книга" (Book), "Замовлення" (Order) та "Елемент замовлення" (OrderItem). Клас "Користувач" включає атрибути, такі як ідентифікатор, ім’я користувача, електронна адреса, роль, повне ім’я, адреса та телефон, що відображає профіль користувача в системі. Клас "Книга" містить ідентифікатор, назву, автора, ціну, кількість на складі та категорію, представляючи товари магазину. Клас "Замовлення" характеризується ідентифікатором, ідентифікатором користувача, датою замовлення, загальною вартістю та статусом, тоді як клас "Елемент замовлення" описує конкретні позиції в замовленні через ідентифікатор, ідентифікатор замовлення, ідентифікатор книги, кількість та ціну.

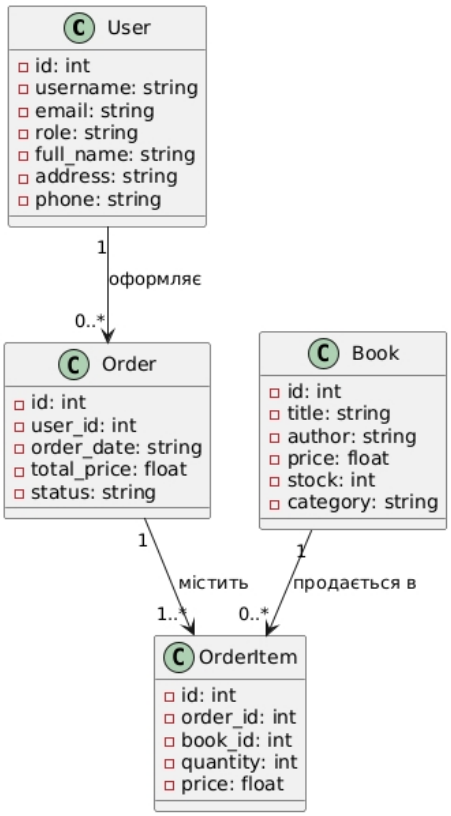

Рис. 2.2. Діаграма класів

Асоціації між класами визначають їхні взаємозв’язки: один користувач може оформити нуль або більше замовлень (1 до 0..), одне замовлення містить один або більше елементів замовлення (1 до 1..), а одна книга може бути присутньою в нульох або більше елементах замовлення (1 до 0..*). Така структура відображає логіку взаємодії між сутностями в системі та забезпечує основу для реалізації функціоналу управління користувачами, каталогом книг та обробкою замовлень.

2.3 Діаграма пакетів

Діаграма пакетів (рис. 2.6) є важливим інструментом у процесі проєктування програмного забезпечення, оскільки вона дозволяє візуально представити організаційну структуру системи, розподіл її компонентів на логічні модулі (пакети) та залежності між ними [41]. У контексті розробки програмного забезпечення системи управління інтернет-магазином, діаграма пакетів

відображає ієрархію та взаємозв’язки між основними складовими системи, забезпечуючи чітке розуміння її архітектури.

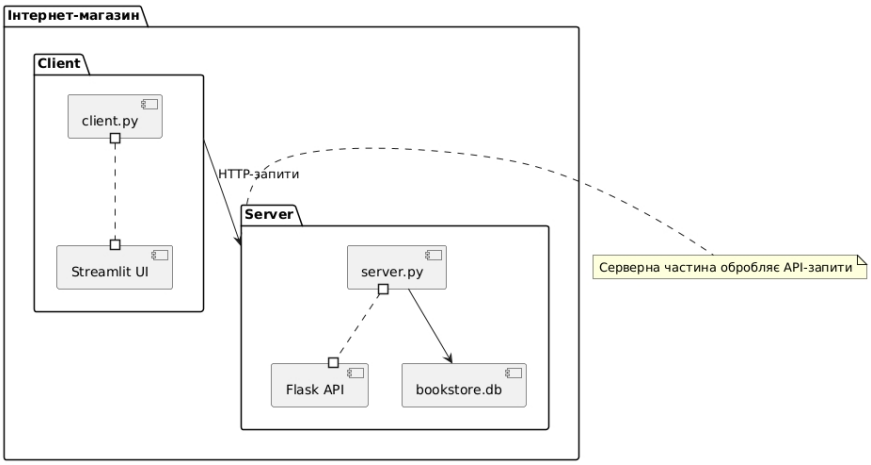

Рис. 2.6. Діаграма пакетів

Організаційна структура програмного забезпечення інтернет-магазину поділена на три ключові пакети: Client, Server та Database. Кожен із них виконує специфічні функції, а їх взаємодія забезпечує цілісність роботи системи. Пакет Client включає клієнтську частину, реалізовану за допомогою фреймворку Streamlit. Цей пакет відповідає за створення користувацького інтерфейсу (UI) та надсилання запитів до серверної частини через API. Файл client.py є основним компонентом цього пакету, забезпечуючи відображення сторінок (наприклад, каталогу, кошика, профілю) та обробку дій користувача, таких як додавання товарів до кошика чи оформлення замовлення.

Пакет Server представляє серверну частину, побудовану на базі фреймворку Flask. Цей пакет обробляє HTTP-запити від клієнта, реалізує бізнес-логіку системи та забезпечує доступ до бази даних. Основний файл server.py містить маршрути API (наприклад, /api/books, /api/orders), які дозволяють виконувати операції з книгами, замовленнями та профілями користувачів. Серверна частина відіграє роль посередника між клієнтом і базою даних, обробляючи запити та повертаючи відповідні результати.

Пакет Database представляє рівень зберігання даних, реалізований за допомогою SQLite. У файлі bookstore.db зберігаються таблиці, такі як users, books, orders та order_items, які пов’язані між собою через зовнішні ключі. Цей пакет забезпечує персистентність даних, дозволяючи системі зберігати інформацію про користувачів, товари та замовлення.

Залежності між пакетами мають чітко визначений напрямок. Пакет Client залежить від Server, оскільки надсилає HTTP-запити для отримання даних чи виконання операцій. У свою чергу, Server залежить від Database, адже звертається до нього для читання, запису чи оновлення даних. Таким чином, утворюється послідовний ланцюжок взаємодій: Client → Server → Database. Ця структура підкреслює модульність системи, де кожен пакет має власну зону відповідальності, що полегшує підтримку та масштабування програмного забезпечення.

Такий підхід до проєктування дозволяє ефективно організувати кодову базу, розділити клієнтську та серверну логіку, а також забезпечити надійне зберігання даних. Діаграма пакетів слугує основою для подальшої розробки, допомагаючи розробникам зрозуміти архітектуру системи та уникнути надмірної зв’язаності між компонентами.

2.4 Діаграма компонентів

Діаграма компонентів (рис. 2.7) є важливим інструментом у процесі моделювання структури програмного забезпечення, оскільки вона відображає організацію вихідного коду та основні етапи створення системи [42]. У контексті розробки програмного забезпечення системи управління інтернет-магазином, діаграма компонентів дозволяє чітко структурувати взаємозв’язки між ключовими модулями проєкту та визначити послідовність їхньої реалізації. У даному випадку розглядається структура, що включає три основні компоненти: client.py, server.py та bookstore.db, які разом формують функціональну основу системи.

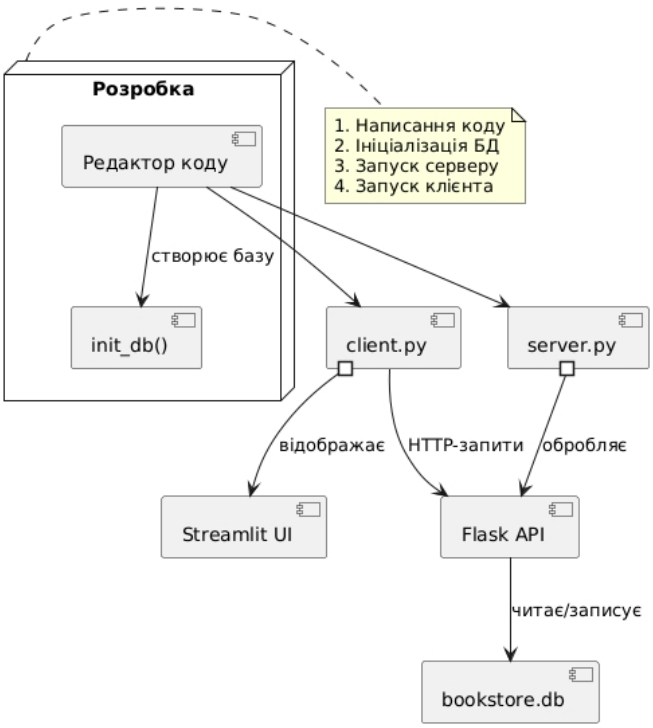

Рис. 2.7. Діаграма компонентів

Компонент client.py відповідає за реалізацію інтерфейсу користувача. Він базується на бібліотеці Streamlit, яка забезпечує створення інтерактивного вебінтерфейсу для відображення каталогу книг, управління кошиком, обробки замовлень та інших функцій, доступних користувачам. Цей модуль відіграє роль клієнтської частини системи, забезпечуючи зручну взаємодію з користувачем через графічний інтерфейс. Він надсилає HTTP-запити до серверної частини для отримання даних або виконання операцій, таких як додавання товару до кошика чи оформлення замовлення.

Компонент server.py є серверною частиною системи, побудованою на фреймворку Flask. Цей модуль реалізує API, що обробляє запити від клієнта, виконує бізнес-логіку (наприклад, автентифікацію, управління замовленнями чи оновлення даних про книги) та забезпечує взаємодію з базою даних. Він виступає посередником між інтерфейсом користувача та сховищем даних, гарантуючи коректну обробку запитів і повернення відповідей у форматі JSON.

Третій компонент, bookstore.db, представляє базу даних, реалізовану за допомогою SQLite. Ця база даних зберігає всю необхідну інформацію про

користувачів, книги, замовлення та їхні деталі. Вона підключена до серверної частини через відповідні SQL-запити, що дозволяють читати, записувати та оновлювати дані. Ініціалізація бази даних відбувається за допомогою функції init_db(), яка створює необхідні таблиці під час першого запуску системи.

Етапи створення програми, відображені в діаграмі, включають кілька ключових кроків. Спочатку здійснюється написання коду для client.py та server.py у редакторі коду, а також розробка структури бази даних через функцію ініціалізації. Наступним етапом є тестування API, що передбачає перевірку коректності обробки запитів сервером. Після цього відбувається інтеграція з базою даних, що забезпечує повноцінну взаємодію між усіма компонентами. Завершальним кроком є розгортання системи, коли сервер запускається для обробки запитів, а клієнтська частина стає доступною для користувачів.

Таким чином, діаграма компонентів чітко ілюструє структуру вихідного коду системи управління інтернет-магазином, демонструючи взаємозв’язки між клієнтською частиною, серверною логікою та базою даних, а також послідовність етапів розробки. Цей підхід сприяє зрозумілості проєкту та полегшує його подальше вдосконалення чи масштабування.

РОЗДІЛ 3. РОЗРОБКА ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Система управління інформаційною базою

У процесі розробки програмного забезпечення системи управління інтернет-магазином ключову роль відіграє система управління інформаційною базою, яка забезпечує зберігання, обробку та доступ до даних. У проєкті використано SQLite – легку реляційну базу даних, що вирізняється простотою інтеграції, мінімальними вимогами до ресурсів і відсутністю необхідності в окремому серверному процесі. SQLite є оптимальним вибором для невеликих і середніх за обсягом проєктів, таких як інтернет-магазин, завдяки своїй портативності та підтримці стандартних SQL-запитів. Ця база даних зберігає всю

інформацію в одному файлі, що спрощує розгортання системи та її резервне копіювання, забезпечуючи при цьому достатню продуктивність для обробки запитів користувачів і менеджерів.

Модель даних фізичного рівня (рис. 3.1) реалізовано через чотири основні таблиці: users, books, orders та order_items, які відображають ключові сутності системи – користувачів, товари, замовлення та їх деталі [28].

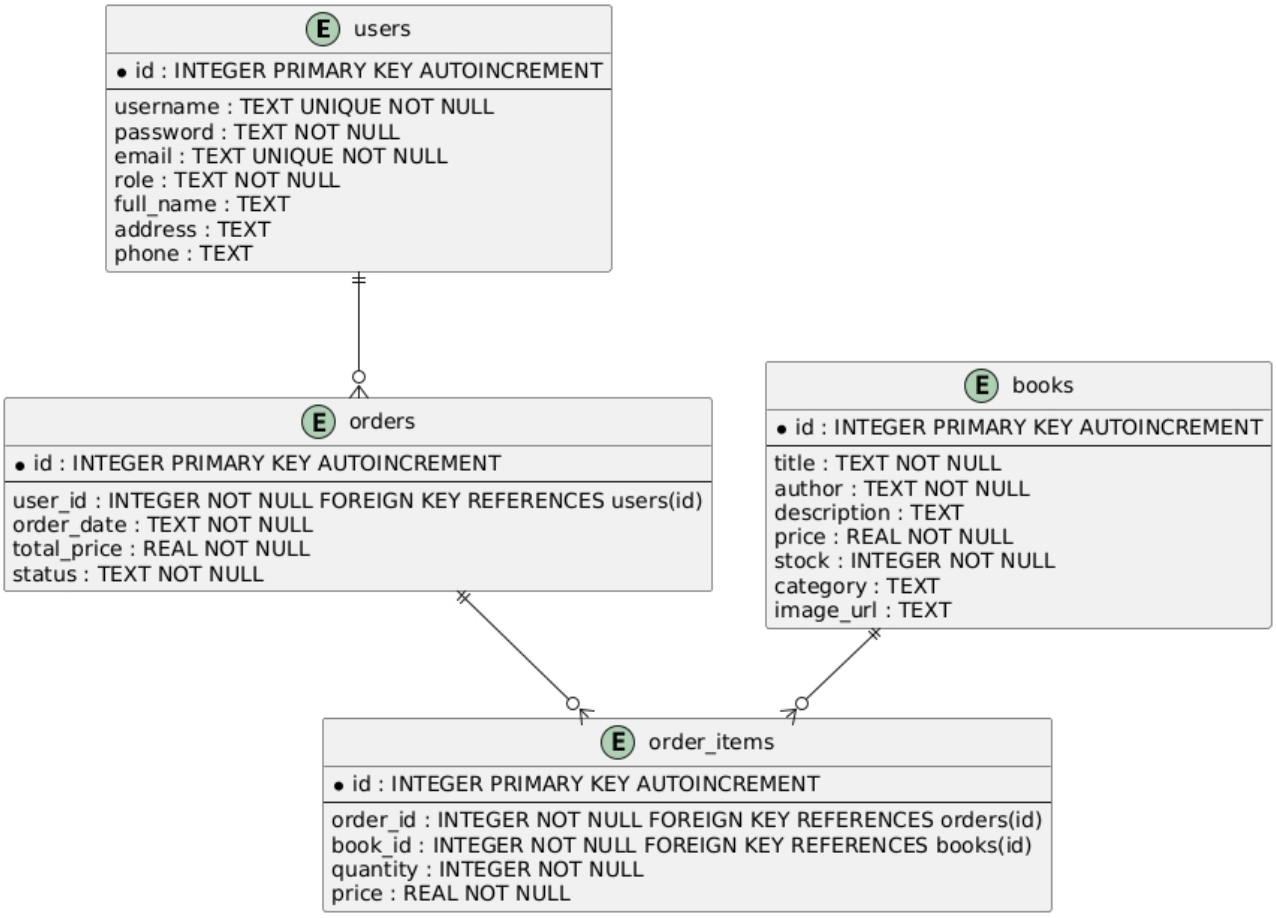

Рис. 3.1. Модель даних фізичного рівня

Детальний опис структури таблиць, їх полів, типів даних і зв’язків, наведено в таблиці 3.1.

Таблиця 3.1

Опис структури таблиць

№	Назва	Опис

1	2	3
1	Таблиця users	призначена для зберігання інформації про користувачів системи. Вона містить такі поля:
1.1	id (INTEGER, первинний ключ, автоінкремент)	унікальний ідентифікатор користувача
1.2	username (TEXT, унікальне, NOT NULL)	ім’я користувача
1.3	password (TEXT, NOT NULL)	хешований пароль
1.4	email (TEXT, унікальне, NOT NULL)	електронна пошта

Продовження таблиці 3.1

1	2	3
1.5	role (TEXT, NOT NULL)	роль користувача (client або manager)
1.6	full_name (TEXT)	повне ім’я
1.7	address (TEXT)	Адреса
1.8	phone (TEXT)	номер телефону
2	Таблиця books	зберігає дані про книги, доступні в магазині
2.1	id (INTEGER, первинний ключ, автоінкремент)	унікальний ідентифікатор книги
2.2	title (TEXT, NOT NULL)	назва книги
2.3	author (TEXT, NOT NULL)	Автор
2.4	description (TEXT)	опис книги
2.5	price (REAL, NOT NULL)	Ціна
2.6	stock (INTEGER, NOT NULL)	кількість на складі
2.7	category (TEXT)	категорія книги
2.8	image_url (TEXT)	посилання на зображення
3	Таблиця orders	фіксує інформацію про замовлення:
3.1	id (INTEGER, первинний ключ, автоінкремент)	унікальний ідентифікатор замовлення
3.2	user_id (INTEGER, NOT NULL)	ідентифікатор користувача, який зробив замовлення
3.3	order_date (TEXT, NOT NULL)	дата і час створення замовлення
3.4	total_price (REAL, NOT NULL)	загальна сума
3.5	status (TEXT, NOT NULL)	статус замовлення (наприклад, "нове", "доставлене")
4	Таблиця order_items	деталізує товари в кожному замовленні:
4.1	id (INTEGER, первинний ключ, автоінкремент)	унікальний ідентифікатор запису
4.2	order_id (INTEGER, NOT NULL)	ідентифікатор замовлення
4.3	book_id (INTEGER, NOT NULL)	ідентифікатор книги
4.4	quantity (INTEGER, NOT NULL)	кількість одиниць

4.5	price (REAL, NOT NULL)	ціна за одиницю на момент замовлення

Зв’язки між таблицями реалізовано через зовнішні ключі, що забезпечують цілісність даних. Поле user_id у таблиці orders є зовнішнім ключем, який посилається на id у таблиці users, пов’язуючи замовлення з конкретним користувачем. У таблиці order_items поле order_id посилається на id у таблиці orders, а book_id – на id у таблиці books, формуючи зв’язок між замовленнями та товарами. Така структура дозволяє ефективно управляти інформацією, підтримуючи операції додавання, оновлення та видалення даних у системі інтернет-магазину.

3.2 Розробка інформаційної бази

Розробка інформаційного та програмного забезпечення системи управління інтернет-магазином є ключовим етапом створення ефективного інструменту для автоматизації бізнес-процесів. Одним із основних компонентів цього процесу є створення інформаційної бази, яка забезпечує зберігання, обробку та доступ до даних. У рамках даного проєкту інформаційна база реалізована за допомогою реляційної бази даних SQLite, що інтегрована у серверну частину програми (файл server.py). У цьому підрозділі розглядається ініціалізація бази даних через функцію init_db та реалізація CRUD-операцій (Create, Read, Update, Delete) для основних таблиць, що забезпечують функціонування системи.

3.2.1.Ініціалізація бази даних

Функція init_db у файлі server.py відповідає за початкове налаштування структури бази даних. Вона створює чотири основні таблиці: users (користувачі), books (книги), orders (замовлення) та order_items (деталі замовлень). Кожна таблиця має чітко визначену структуру з полями, що відображають сутності

системи інтернет-магазину. Наприклад, таблиця users містить поля id, username, password, email, role, full_name, address і phone, що дозволяють зберігати інформацію про зареєстрованих користувачів, включаючи їхню роль (клієнт або менеджер). Таблиця books включає поля id, title, author, description, price, stock, category та image_url, які описують товари магазину. Таблиці orders і order_items пов’язані між собою та з іншими таблицями через зовнішні ключі (user_id, book_id), що забезпечує реляційну цілісність даних.

Процес ініціалізації відбувається шляхом виконання SQL-запитів із використанням бібліотеки sqlite3. Функція викликається при запуску серверної програми, що гарантує наявність необхідної структури бази даних перед початком роботи системи. Використання SQLite як СУБД обумовлено її легкістю, портативністю та достатньою функціональністю для невеликих і середніх проєктів, таких як даний інтернет-магазин. Завдяки цьому підходу забезпечується швидке розгортання системи без необхідності встановлення додаткових серверних компонентів.

3.2.2. Реалізація CRUD-операцій для таблиць

Для забезпечення повноцінної роботи з інформаційною базою в server.py реалізовано CRUD-операції, які дозволяють створювати, читати, оновлювати та видаляти записи в таблицях. Ці операції інтегровані у вигляді маршрутів API, що обробляються сервером Flask, і доступні клієнтській частині (client.py) через HTTP-запити (таблиця 3.2).

Таблиця 3.2

CRUD-операції

№	Назва	Опис
1	**2**	**3**
1	**Create (Створення)**	Операція створення реалізована, наприклад, у маршруті /api/register для додавання нового користувача. Функція отримує дані у форматі JSON, хешує пароль за допомогою алгоритму SHA-256 і вставляє запис у таблицю users за допомогою SQL-запиту INSERT. Аналогічно маршрут /api/books (метод POST) дозволяє менеджерам додавати нові

		книги до таблиці books, перевіряючи наявність обов’язкових полів (title, author, price, stock)

Продовження таблиці 3.2

1	2	3
2	**Read (Читання)**	Операція читання даних представлена у маршрутах, таких як /api/books (метод GET), який повертає список книг із можливістю фільтрації за категорією та пошуковим запитом. Запит до бази формується динамічно залежно від параметрів, що передаються через URL. Наприклад, для отримання деталей конкретної книги використовується маршрут /api/books/<int:book_id>, який повертає повну інформацію про книгу за її ідентифікатором
4	**Update (Оновлення)**	Оновлення даних реалізовано, зокрема, у маршруті /api/user/profile (метод PUT), де користувач може змінити свої персональні дані (email, ім’я, адресу тощо). Для таблиці books маршрут /api/books/<int:book_id> (метод PUT) дозволяє менеджерам оновлювати інформацію про книгу, наприклад, ціну чи кількість на складі. Операція супроводжується перевіркою існування запису, щоб уникнути помилок
5	**Delete (Видалення)**	Видалення даних підтримується, наприклад, у маршруті /api/books/<int:book_id> (метод DELETE), де менеджер може видалити книгу з бази. Перед виконанням операції перевіряється наявність запису, а після видалення повертається підтвердження успіху

Усі операції захищені механізмами автентифікації та авторизації. Декоратор @token_required перевіряє наявність і валідність JWT-токена, а @manager_required обмежує доступ до операцій управління (створення, оновлення, видалення) лише для користувачів із роллю менеджера. Це забезпечує безпеку даних і відповідність функціоналу ролям користувачів.

Таким чином, розроблена інформаційна база та реалізовані CRUD-операції створюють надійну основу для роботи системи управління інтернет-магазином. Поєднання простоти SQLite із гнучкістю Flask дозволяє ефективно обробляти запити клієнтів, забезпечуючи стабільність і масштабованість системи.

3.3 Вибір інструментарію для створення прикладного програмного забезпечення

Розробка прикладного програмного забезпечення для системи управління інтернет-магазином є ключовим етапом реалізації проєкту, що вимагає ретельного підходу до вибору інструментів, які забезпечують ефективність, масштабованість та зручність використання системи. У контексті створення програмного забезпечення для інтернет-магазину, представленого у реалізованому коді, вибір інструментарію базується на потребах проєкту, зокрема на необхідності створення інтерактивного клієнтського інтерфейсу та надійного серверного забезпечення для обробки запитів, управління даними та забезпечення безпеки. У цьому підрозділі обґрунтовано вибір інструментів для реалізації клієнтської та серверної частин системи, враховуючи їх функціональні можливості, доступність документації, підтримку спільноти та сумісність із сучасними технологіями.

Для розробки клієнтської частини системи обрано **Streamlit** — фреймворк на основі Python, призначений для створення інтерактивних веб-додатків із мінімальними зусиллями. Вибір Streamlit зумовлений його простотою у використанні, що дозволяє швидко створювати графічні інтерфейси без необхідності глибоких знань фронтенд-технологій, таких як HTML, CSS чи JavaScript. Цей інструмент ідеально підходить для проєктів, де акцент робиться на функціональності та швидкому прототипуванні, що є важливим для системи управління інтернет-магазином. У коді client.py видно, як Streamlit використовується для побудови сторінок каталогу, кошика, профілю користувача та інших компонентів, із застосуванням таких функцій, як st.button, st.text_input та st.markdown для створення інтерактивних елементів і стилізованого відображення даних. Додатковою перевагою Streamlit є його інтеграція з Python-екосистемою, що дозволяє легко підключати бібліотеки для роботи з даними, такі як pandas, яка використовується в проєкті для обробки табличних даних. Водночас, Streamlit забезпечує автоматичне оновлення

інтерфейсу при зміні стану програми, що спрощує розробку динамічних компонентів, таких як кошик чи список замовлень.

Для серверної частини системи обрано **Flask** — легкий і гнучкий веб-фреймворк на Python, який забезпечує створення RESTful API для обробки запитів від клієнта. Вибір Flask обґрунтований його мінімалістичним підходом, що дозволяє розробнику контролювати архітектуру додатка, а також його широкою популярністю серед розробників, що гарантує наявність великої кількості навчальних матеріалів і розширень. У коді server.py Flask використовується для реалізації маршрутів API, таких як /api/register, /api/books та /api/orders, що забезпечують реєстрацію користувачів, управління книгами та обробку замовлень. Для підвищення безпеки застосовано бібліотеку **PyJWT**, яка реалізує автентифікацію на основі JSON Web Tokens (JWT), дозволяючи захищати доступ до ресурсів через токени, як видно у декораторі @token_required. Крім того, Flask інтегрується з **Flask-CORS** для підтримки крос-доменних запитів, що є необхідним для взаємодії між клієнтом і сервером у веб-додатках.

Для управління даними обрано **SQLite** як легку вбудовану базу даних, що не потребує окремого серверного процесу. SQLite підходить для проєктів середнього масштабу, таких як даний інтернет-магазин, завдяки своїй простоті в налаштуванні та достатній продуктивності для обробки запитів від невеликої кількості користувачів. У коді server.py видно, як SQLite використовується для створення таблиць (users, books, orders, order_items) та виконання SQL-запитів для зберігання й обробки інформації про користувачів, товари та замовлення. Вибір SQLite також зумовлений її повною сумісністю з Python через стандартну бібліотеку sqlite3, що усуває потребу в додаткових залежностях.

Додаткові бібліотеки, такі як **requests** на клієнтській стороні, забезпечують зручну взаємодію з серверним API, дозволяючи надсилати HTTP-запити для отримання даних чи виконання дій, наприклад, додавання книги до кошика чи оновлення статусу замовлення. Бібліотека **hashlib** використовується на сервері для хешування паролів, що підвищує безпеку зберігання даних користувачів. Усі

ці інструменти разом створюють цілісну екосистему, яка відповідає вимогам проєкту.

Отже, вибір інструментарію для створення прикладного програмного забезпечення системи управління інтернет-магазином базується на принципах простоти, ефективності та сумісності. Streamlit забезпечує швидку розробку інтуїтивно зрозумілого інтерфейсу, Flask пропонує гнучке рішення для серверної логіки, а SQLite гарантує надійне зберігання даних із мінімальними ресурсними затратами. Цей набір інструментів дозволяє реалізувати функціональну систему, яка відповідає потребам як клієнтів, так і менеджерів магазину, забезпечуючи зручність використання, безпеку та потенціал для подальшого масштабування.

3.4 Алгоритмізація та програмування програмних модулів

Розробка програмного забезпечення системи управління інтернет-магазином передбачає створення клієнтської та серверної частин, які взаємодіють між собою для забезпечення функціональності системи. У рамках даного проєкту клієнтська частина реалізована у файлі client.py з використанням бібліотеки Streamlit, а серверна — у файлі server.py на базі фреймворку Flask. Важливо розглянути детальний опис основних функцій клієнтського модуля, API-ендпоінтів серверного модуля, а також ключових алгоритмів, що забезпечують безпеку та логіку роботи системи.

3.4.1. Опис основних функцій у client.py

Клієнтська частина (client.py) відповідає за взаємодію користувача з системою через графічний інтерфейс, побудований за допомогою Streamlit. Основні функції зосереджені на авторизації, управлінні кошиком та оформленні замовлень, що є ключовими компонентами функціональності інтернет-магазину.

Авторизація користувачів: функціонал авторизації реалізований через функції login_user(), register_user() та logout_user():

– Функція login_user(username, password) надсилає POST-запит до серверного ендпоінту /api/login із введеними користувачем даними (ім’я користувача та пароль). У разі успішної авторизації сервер повертає JWT-токен і дані користувача (ID, ім’я, роль), які зберігаються у сесійному стані Streamlit (st.session_state). Це дозволяє системі ідентифікувати користувача під час подальшої роботи.

– Функція register_user(username, password, email, role, full_name, address, phone) виконує реєстрацію нового користувача шляхом надсилання POST-запиту до /api/register. Після успішної реєстрації користувач отримує токен і автоматично перенаправляється на сторінку входу.

– Функція logout_user() очищає сесійні дані (токен, інформацію про користувача, кошик), завершуючи сеанс роботи.

Ці функції забезпечують базову автентифікацію та авторизацію, що є основою для персоналізованого доступу до системи.

Управління кошиком реалізовано через набір функцій: add_to_cart(book), remove_from_cart(index), update_cart_quantity(index, quantity) та calculate_cart_total():

– add_to_cart(book) додає книгу до кошика, який зберігається у сесійному стані як список словників. Якщо книга вже присутня, її кількість збільшується; інакше створюється новий запис із початковою кількістю 1.

– remove_from_cart(index) видаляє елемент із кошика за індексом, оновлюючи список товарів.

– update_cart_quantity(index, quantity) дозволяє змінювати кількість одиниць товару в кошику, а якщо кількість стає нульовою, викликає видалення товару.

– calculate_cart_total() обчислює загальну вартість товарів у кошику, множачи ціну кожного товару на його кількість і сумуючи результати.

Ці функції забезпечують гнучке управління кошиком, дозволяючи користувачу додавати, видаляти та редагувати товари перед оформленням замовлення.

Процес оформлення замовлень реалізований через функцію create_order(items), яка надсилає POST-запит до ендпоінту /api/orders із вмістом кошика. У разі успіху кошик очищається, а користувач перенаправляється на сторінку "Мої замовлення". Функція get_orders() отримує список замовлень користувача через GET-запит до /api/orders, а get_order(order_id) — деталі конкретного замовлення через /api/orders/<order_id>. Для менеджерів додатково доступна функція update_order_status(order_id, status), яка оновлює статус замовлення через PUT-запит до /api/orders/<order_id>/status.

Цей функціонал забезпечує повний цикл роботи із замовленнями — від створення до перегляду та управління.

3.4.2. Опис API-ендпоінтів у server.py

Серверна частина (server.py) надає RESTful API для обробки запитів від клієнта. Основні ендпоінти включають автентифікацію, управління книгами та замовленнями (таблиця 3.3).

Таблиця 3.3

Основні ендпоінти

№	Назва	Опис
1	**2**	**3**
1	/api/register **(POST)**	Реєструє нового користувача, зберігаючи його дані (ім’я, хешований пароль, email, роль тощо) у базі даних SQLite. Повертає JWT-токен і ID користувача у разі успіху (код 201) або помилку, якщо користувач із таким ім’ям чи email уже існує (код 409)
2	/api/login **(POST)**	Перевіряє введені ім’я користувача та хешований пароль у базі даних. У разі збігу генерує JWT-токен і повертає його разом із даними користувача (код 200). Інакше повертає помилку автентифікації (код 401)

Продовження таблиці 3.3

1	2	**3**
3	/api/books **(GET, POST)**	
3.1	GET	Повертає список книг із можливістю фільтрації за категорією та пошуковим запитом. Дані отримуються з таблиці books і повертаються у форматі JSON
3.2	POST	Додає нову книгу до бази даних (доступно лише менеджерам). Вимагає автентифікацію через токен і перевірку ролі
4	api/orders **(GET, POST)**	
4.1	GET	Повертає список замовлень. Для клієнтів — лише їхні замовлення, для менеджерів — усі замовлення. Включає деталі товарів із таблиці order_items
4.2	POST	Створює нове замовлення, перевіряючи наявність товарів на складі, оновлюючи запаси та записуючи дані в таблиці orders і order_items
5	/api/orders/<order_id>/status **(PUT)**	Оновлює статус замовлення (доступно лише менеджерам). Перевіряє існування замовлення та записує новий статус у базу даних

Ці ендпоінти забезпечують повноцінну взаємодію між клієнтом і сервером, підтримуючи основні операції інтернет-магазину.

3.4.3. Алгоритми

Основні алгоритми показані на рис. 3.2.

Рис. 3.2. Основні алгоритми системи

Хешування паролів (hashlib): для забезпечення безпеки паролі користувачів хешуються за допомогою бібліотеки hashlib. Функція hash_password(password) використовує алгоритм SHA-256 для перетворення пар

пароля у рядок фіксованої довжини (64 символи). Алгоритм працює наступним чином:

– Вхідні дані (пароль) кодуються у байтовий формат через .encode().

– Застосовується функція hashlib.sha256() для створення хешу.

– Результат конвертується у шістнадцятковий формат через .hexdigest().

Хеш зберігається у базі даних і використовується для перевірки під час автентифікації, що унеможливлює відновлення оригінального пароля.

Генерація та перевірка JWT-токенів (PyJWT): JWT-токени використовуються для автентифікації та авторизації.

Генерація (generate_token(user_id)**):**

– Формується словник payload із полями user_id та exp (час закінчення дії токена, який обчислюється як поточний час UTC плюс 24 години).

– Викликається jwt.encode(payload, SECRET_KEY, algorithm='HS256'), де SECRET_KEY — випадковий ключ, згенерований через os.urandom(24).

– Результат — закодований токен у форматі Base64.

Перевірка (verify_token(token)**):**

– Токен декодується через jwt.decode(token, SECRET_KEY, algorithms=['HS256']).

– У разі успіху повертається user_id. Якщо токен прострочений або недійсний, повертається None.

Цей механізм забезпечує безпечну передачу ідентифікатора користувача між клієнтом і сервером.

Оновлення статусу замовлень: алгоритм оновлення статусу реалізовано у функції update_order_status(order_id, status):

– Отримується токен із заголовка запиту через get_auth_header().

– Надсилається PUT-запит до /api/orders/<order_id>/status із новим статусом.

– Сервер перевіряє:

- ✓ Наявність замовлення в базі даних через SQL-запит SELECT id FROM orders WHERE id = ?.
- ✓ Права доступу (лише менеджери можуть змінювати статус).

– У разі успіху виконується UPDATE orders SET status = ? WHERE id = ?, і зміни фіксуються в базі даних.

– Клієнт отримує підтвердження або повідомлення про помилку, яке відображається в інтерфейсі.

Цей алгоритм забезпечує контрольоване управління життєвим циклом замовлення.

Розроблені програмні модулі client.py та server.py забезпечують повноцінну роботу системи управління інтернет-магазином. Клієнтська частина реалізує інтуїтивно зрозумілий інтерфейс для авторизації, роботи з кошиком і замовленнями, тоді як серверна частина надає API для обробки запитів і управління даними. Використання алгоритмів хешування паролів, JWT-токенів і оновлення статусів замовлень гарантує безпеку та ефективність системи, відповідаючи сучасним стандартам розробки програмного забезпечення.

РОЗДІЛ 4. РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

4.1 Тестування системи

Успішне впровадження та подальша експлуатація системи управління інтернет-магазином значною мірою залежать від ретельного тестування всіх її компонентів. На основі кодів проєкту (client.py та server.py), що є результатом реалізації проєкту, тестування системи має охоплювати три ключові напрями: тестування API, тестування інтерфейсу користувача та тестування безпеки. Кожен із цих показників відіграє важливу роль у забезпеченні стабільності, зручності використання та захисту системи від потенційних загроз.

4.1.1. Тестування API (Postman): перевірка ендпоінтів

Першим етапом тестування є перевірка коректності роботи серверної частини системи, реалізованої у файлі server.py на базі фреймворку Flask. Для цього рекомендується використовувати інструмент Postman, який дозволяє проводити автоматизоване та ручне тестування RESTful API. Необхідно перевірити усі ендпоінти, визначені у коді, зокрема /api/register, /api/login, /api/books, /api/orders тощо. Наприклад, для ендпоінта /api/register слід протестувати успішну реєстрацію нового користувача з коректними даними (очікуваний статус відповіді – 201), а також обробку помилок, таких як спроба реєстрації з уже існуючим ім’ям користувача чи email (очікуваний статус – 409). Аналогічно, ендпоінт /api/orders потребує перевірки створення замовлення з валідними даними, а також реакції системи на некоректні запити, наприклад, замовлення з недостатньою кількістю товару на складі (очікуваний статус – 400). Для автоматизації цього процесу доцільно створити набір тестових сценаріїв у Postman, які включають позитивні та негативні кейси, а також перевіряють коректність обробки JWT-токенів для захищених маршрутів. Результати тестування слід документувати, зазначаючи статус відповіді, час виконання запиту та відповідність повернених даних специфікації API.

4.1.2. Тестування інтерфейсу: коректність відображення каталогу, кошика, замовлень

Другим етапом є тестування клієнтської частини системи, реалізованої у файлі client.py з використанням Streamlit. Основна мета – переконатися, що інтерфейс користувача функціонує коректно та відповідає потребам кінцевих користувачів. Перевірка каталогу книг (функція show_catalog) має включати тестування відображення списку книг із застосуванням фільтрів за категоріями та пошуковими запитами, а також коректність переходу до деталей книги після

натискання кнопки "Деталі". Для кошика (функція show_cart) необхідно протестувати додавання товарів, зміну їх кількості, видалення позицій та розрахунок загальної суми, враховуючи, що ці дії мають оновлювати стан сесії (st.session_state.cart) без помилок. Тестування сторінки замовлень (show_orders та show_order_details) передбачає перевірку відображення списку замовлень користувача, детальної інформації про замовлення та, для менеджерів, можливості зміни статусу. Особливу увагу слід приділити адаптивності дизайну, коректності стилів CSS (визначених у main) та обробці крайніх випадків, таких як порожній кошик чи відсутність замовлень. Рекомендується провести тестування на різних пристроях і браузерах для забезпечення кросплатформної сумісності.

4.1.3. Тестування безпеки: авторизація, права доступу

Третій етап тестування зосереджений на безпеці системи, що є критично важливим для захисту даних користувачів та запобігання несанкціонованому доступу. У серверній частині реалізовано механізм автентифікації на основі JWT-токенів (функції generate_token, verify_token), який потребує перевірки. Тестування авторизації включає успішний вхід із правильними обліковими даними (ендпоінт /api/login), а також обробку невдалих спроб із некоректними даними. Окремо слід перевірити поведінку системи при використанні прострочених або підроблених токенів – у таких випадках захищені маршрути (наприклад, /api/user/profile) повинні повертати статус 401. Для тестування прав доступу необхідно переконатися, що функції, позначені декоратором @manager_required (наприклад, /api/books/<int:book_id> для редагування книг), доступні лише користувачам із роллю "manager", тоді як звичайні клієнти отримують статус 403. На клієнтському рівні рекомендується перевірити, що сторінки "Управління товарами" та "Управління замовленнями" відображаються лише для менеджерів, а спроби доступу з боку клієнтів перенаправляють їх до каталогу з відповідним попередженням. Додатково доцільно провести

тестування на вразливість до базових атак, таких як SQL-ін’єкції, враховуючи використання параметризованих запитів у server.py, що має мінімізувати подібні ризики.

Таким чином, комплексне тестування системи управління інтернет-магазином охоплює перевірку API на сервері, функціональності та зручності інтерфейсу, а також надійності механізмів безпеки. Результати тестування стануть основою для усунення виявлених недоліків перед впровадженням системи в експлуатацію, забезпечуючи її стабільну роботу та відповідність заявленим вимогам.

4.2 Діаграма розгортання

Розробка програмного забезпечення системи управління інтернет-магазином, представлена у вигляді кодів client.py та server.py, потребує чіткого розуміння архітектури її розгортання для забезпечення стабільної роботи та ефективної взаємодії між компонентами. Діаграма розгортання (Deployment Diagram) є важливим інструментом, який відображає фізичну структуру системи, розподіл її складових між апаратними вузлами та їх взаємозв’язки. У даному підрозділі розглядається діаграма розгортання системи з урахуванням специфіки клієнтської частини (на базі Streamlit), серверної частини (на базі Flask) та бази даних SQLite, а також адаптація топології до умов навчального корпусу.

Діаграма розгортання системи управління інтернет-магазином зображення на рис. 4.1 [43].

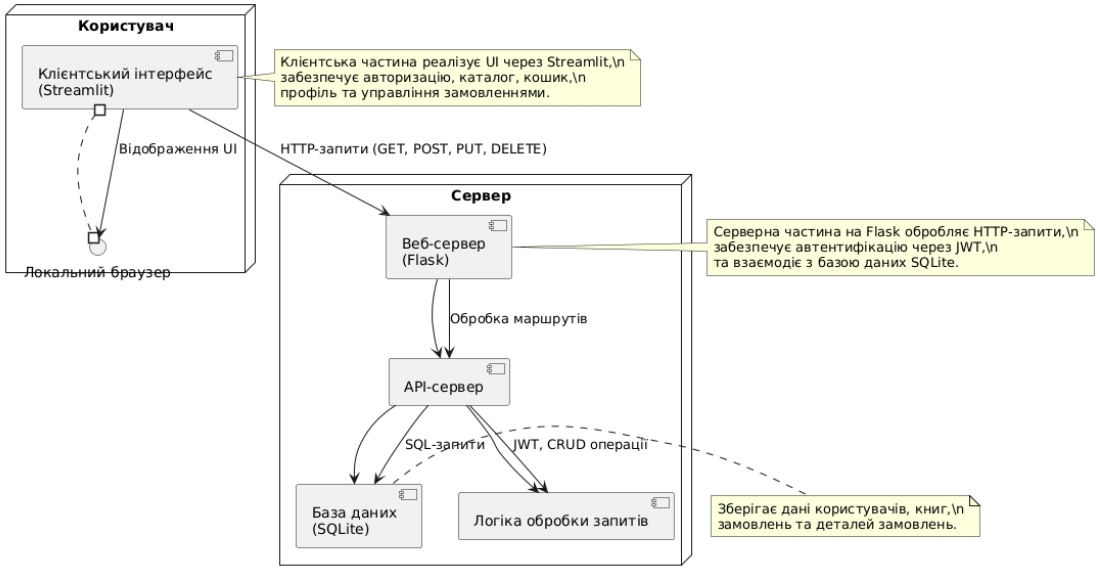

Рис. 4.1. Діаграма розгортання системи управління інтернет-магазином

Архітектура системи управління інтернет-магазином, побудована на основі створених кодів, передбачає три основні компоненти: клієнтську частину, серверну частину та базу даних. Клієнтська частина, реалізована у файлі client.py з використанням бібліотеки Streamlit, виконує роль інтерфейсу користувача. Вона розгортається на локальному комп’ютері або сервері користувача як веб-додаток, доступний через браузер. Streamlit забезпечує динамічне відображення сторінок (наприклад, каталогу книг, кошика, профілю користувача) та надсилає HTTP-запити до серверної частини через API-ендпоінти, визначені у server.py. Для роботи клієнтської частини необхідне стабільне підключення до мережі та встановлення Python із відповідними бібліотеками (Streamlit, Requests тощо).

Серверна частина, реалізована у файлі server.py на базі фреймворку Flask, функціонує як RESTful API-сервер, що обробляє запити від клієнта, виконує бізнес-логіку та взаємодіє з базою даних. Вона розгортається на локальному сервері (наприклад, за адресою http://localhost:5000), що зазначено в коді як API_URL. Flask-сервер підтримує маршрути для автентифікації користувачів, управління книгами, обробки замовлень тощо, використовуючи токени JWT для забезпечення безпеки. Для зберігання даних система використовує базу SQLite

(bookstore.db), яка розгортається на тому ж апаратному вузлі, що й серверна частина, завдяки її легкості та простоті інтеграції.

На діаграмі розгортання ці компоненти зображуються як окремі вузли:

– **Клієнтський вузол**: веб-браузер користувача, що запускає Streamlit-додаток через локальний сервер (наприклад, localhost:8501).

– **Серверний вузол**: фізичний сервер або комп’ютер, на якому розгорнуто Flask-сервер та базу даних SQLite.

– **Канали зв’язку**: HTTP-запити між клієнтом і сервером через мережу (локальну або Інтернет).

Взаємодія між вузлами відбувається через мережеві протоколи, де клієнт надсилає запити (GET, POST, PUT, DELETE) до API, а сервер повертає відповіді у форматі JSON. База даних SQLite, будучи файловою системою, не потребує окремого серверного процесу, що спрощує розгортання, але може обмежувати масштабованість при великих навантаженнях.

Діаграма розгортання, що демонструє топологію системи в навчальному корпусі показана на рис. 4.2.

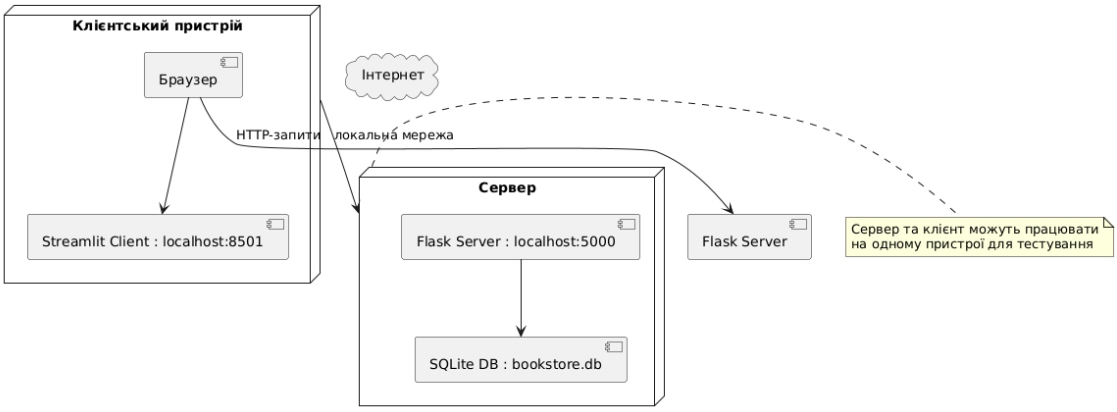

Рис. 4.2. Діаграма розгортання, що демонструє топологію системи в навчальному корпусі

У контексті навчального корпусу діаграма розгортання адаптується до локальної інфраструктури. Клієнтська частина може бути розгорнута на комп’ютерах у комп’ютерних класах або особистих пристроях студентів і викладачів, підключених до локальної мережі навчального закладу. Для запуску

Streamlit-додатку достатньо одного комп’ютера з доступом до мережі, який виконуватиме роль хоста для клієнтського інтерфейсу. Серверна частина (Flask) та база даних розгортаються на виділеному сервері в межах навчального корпусу, наприклад, у серверній кімнаті або на комп’ютері викладача, що виконує роль центрального вузла. Локальна мережа (LAN) забезпечує зв’язок між клієнтами та сервером, що дозволяє уникнути залежності від зовнішнього Інтернету та підвищує швидкість обробки запитів.

Топологія системи в навчальному корпусі передбачає:

– **Клієнтські вузли**: комп’ютери в аудиторіях (наприклад, 10–20 одиниць), що запускають Streamlit через локальну мережу.

– **Серверний вузол**: один сервер (або потужний комп’ютер) із встановленим Flask та SQLite, доступний за внутрішньою IP-адресою (наприклад, 192.168.1.100:5000).

– **Мережева інфраструктура**: комутатори та маршрутизатори локальної мережі навчального корпусу, що забезпечують стабільний зв’язок.

Для забезпечення працездатності системи рекомендується перевірити наявність необхідного програмного забезпечення (Python, Flask, Streamlit) на всіх вузлах, а також налаштувати мережеві параметри (наприклад, відкрити порти 5000 для Flask і 8501 для Streamlit). У разі розгортання в навчальному середовищі доцільно передбачити резервне копіювання бази даних та обмеження доступу до серверного вузла для підвищення безпеки.

Таким чином, діаграма розгортання відображає модульну та гнучку архітектуру системи управління інтернет-магазином, яка легко адаптується до умов навчального корпусу. Її реалізація сприяє ефективному навчанню та практичному застосуванню розробленого програмного забезпечення.

4.3 Вимоги до апаратного та програмного забезпечення

Розроблене програмне забезпечення системи управління інтернет-магазином, представлене у вигляді кодів client.py та server.py, є результатом реалізації проєкту, що забезпечує функціонування веб-додатку для продажу

книг. Для коректної роботи системи необхідно дотримуватися певних вимог до апаратного та програмного забезпечення. Ці вимоги враховують як клієнтську, так і серверну частини додатку, забезпечуючи стабільність, продуктивність і доступність для користувачів.

Апаратне забезпечення відіграє ключову роль у забезпеченні ефективної роботи системи, зокрема для обробки запитів користувачів, зберігання даних та виконання серверних операцій. На основі аналізу потреб проєкту визначено мінімальні технічні характеристики, які подано в таблиці 4.1.

Таблиця 4.1

Вимоги до апаратного забезпечення

Компонент	Мінімальні вимоги	Примітки
Процесор	2 ядра, 2 ГГц	Забезпечує обробку запитів і обчислень
Оперативна пам’ять	4 ГБ	Для одночасної роботи серверу та клієнта
Диск	10 ГБ вільного місця	Зберігання бази даних SQLite і файлів додатку

Процесор із двома ядрами та частотою 2 ГГц є достатнім для виконання базових операцій, таких як обробка HTTP-запитів та взаємодія з базою даних SQLite. Оперативна пам’ять обсягом 4 ГБ дозволяє одночасно запускати сервер Flask та клієнтський інтерфейс Streamlit без значних затримок. Вільне місце на диску в 10 ГБ необхідне для зберігання бази даних bookstore.db, яка включає інформацію про користувачів, книги та замовлення, а також для розміщення програмних файлів і тимчасових даних.

Програмне забезпечення, необхідне для функціонування системи, охоплює операційну систему та набір інструментів і бібліотек, які забезпечують реалізацію серверної та клієнтської логіки. Вимоги до програмного забезпечення подано в таблиці 4.2.

Таблиця 4.2

Вимоги до програмного забезпечення

Компонент	Вимоги	Призначення

Операційна система	Windows/Linux/macOS	Базове середовище для запуску додатку
Python	Версія 3.8 або вище	Інтерпретатор для виконання коду
Бібліотека Streamlit	Остання версія	Формування клієнтського веб-інтерфейсу
Бібліотека Flask	Остання версія	Реалізація серверної частини API
Бібліотека SQLite3	Вбудована в Python	Управління базою даних
Бібліотека Requests	Остання версія	Виконання HTTP-запитів із клієнта до сервера
Бібліотека PyJWT	Остання версія	Робота з JWT-токенами для автентифікації
Бібліотека Pandas	Остання версія	Обробка даних (опціонально)

Операційна система (Windows, Linux або macOS) забезпечує універсальність розгортання системи, що робить її доступною для широкого кола користувачів. Python версії 3.8+ є основою для виконання коду, оскільки підтримує всі необхідні бібліотеки. Streamlit використовується для створення інтерактивного інтерфейсу користувача (client.py), тоді як Flask реалізує серверну логіку та API (server.py). SQLite3, вбудована в Python, забезпечує легке управління базою даних без необхідності додаткових серверів. Бібліотека Requests дозволяє клієнту взаємодіяти з сервером через HTTP-запити, PyJWT підтримує автентифікацію за допомогою JWT-токенів, а Pandas може застосовуватися для аналізу даних, якщо це передбачено подальшим розвитком проєкту.

Таким чином, зазначені вимоги до апаратного та програмного забезпечення є оптимальними для забезпечення стабільної роботи системи управління інтернет-магазином, дозволяючи ефективно виконувати основні функції: відображення каталогу, обробку замовлень і адміністрування. Ці характеристики враховують як поточні потреби проєкту, так і можливість його масштабування в майбутньому.

4.4 Опис роботи програмного забезпечення

Розроблене програмне забезпечення системи управління інтернет-магазином, представлене у вигляді клієнтської частини (client.py) та серверної частини (server.py), забезпечує комплексну взаємодію між користувачами та базою даних для реалізації функціоналу онлайн-книгарні. Система побудована на основі клієнт-серверної архітектури, де клієнтська частина, реалізована з використанням бібліотеки Streamlit, відповідає за інтерфейс користувача, а серверна частина, створена на базі фреймворку Flask, обробляє запити та забезпечує доступ до даних через RESTful API.

Клієнтська частина (client.py) є інтерактивним веб-додатком, який дозволяє користувачам виконувати основні операції: перегляд каталогу книг, додавання товарів до кошика, оформлення замовлень, а також управління профілем і перегляд історії покупок. Інтерфейс адаптований для двох типів користувачів — клієнтів і менеджерів, що реалізовано через динамічне відображення меню залежно від ролі користувача, збереженої у стані сесії (st.session_state). Наприклад, менеджери мають доступ до додаткових функцій, таких як управління асортиментом товарів і статусами замовлень. Для забезпечення зручності роботи передбачено детальну систему відладки (debug_print), яка фіксує ключові дії користувача та системи у логах, доступних через бічну панель. Клієнт взаємодіє з API сервера через HTTP-запити, використовуючи функції, такі як get_books(), create_order(), update_order_status(), що обробляють отримані дані та відображають їх у зрозумілому вигляді з використанням HTML-розмітки та CSS-стилів.

Серверна частина (server.py) виконує роль бекенду, забезпечуючи обробку запитів, автентифікацію, авторизацію та взаємодію з базою даних SQLite (bookstore.db). База даних містить таблиці для зберігання інформації про користувачів (users), книги (books), замовлення (orders) та деталі замовлень (order_items), що дозволяє ефективно управляти асортиментом і транзакціями. Автентифікація реалізована через JWT-токени, які генеруються функцією

generate_token() і перевіряються декоратором @token_required. Доступ до певних операцій, таких як редагування книг чи зміна статусів замовлень, обмежено для менеджерів за допомогою декоратора @manager_required. API підтримує CRUD-операції (створення, читання, оновлення, видалення) для всіх основних сутностей, а також забезпечує фільтрацію книг за категоріями та пошуковими запитами.

Робота системи розпочинається з ініціалізації бази даних (init_db()) та запуску серверного додатку, після чого клієнтський додаток стає доступним через веб-інтерфейс. Користувач може зареєструватися, увійти в систему, переглянути каталог, додати книги до кошика та оформити замовлення. Менеджери, у свою чергу, отримують можливість додавати нові книги, редагувати існуючі та контролювати процес обробки замовлень. Усі дії супроводжуються відповідними повідомленнями про успіх чи помилку, що підвищує прозорість взаємодії. Таким чином, програмне забезпечення забезпечує повноцінне функціонування інтернет-магазину, поєднуючи зручність використання з надійністю обробки даних.

ВИСНОВКИ

Розробка програмного забезпечення системи управління інтернет-магазином книжкової продукції, представлена у даній роботі, є прикладом комплексного підходу до створення інформаційної системи, що інтегрує сучасні технології електронної комерції з потребами книжкової торгівлі. Основна мета проєкту полягала в автоматизації ключових процесів онлайн-продажу книг, забезпеченні зручності для користувачів і підвищенні ефективності управління асортиментом та замовленнями. Реалізована система, побудована на основі технологій React.js, Node.js з Express.js, SQLite, а також Streamlit і Flask у прототипі, демонструє практичну цінність як для клієнтів, так і для менеджерів, створюючи зручний інструмент для взаємодії в цифровому середовищі.

Процес розробки охопив системний аналіз предметної області, що дозволив чітко визначити вимоги до функціональності, структури даних і взаємодії компонентів. Логічна модель даних, представлена ER-діаграмою, забезпечила цілісність зберігання інформації про користувачів, книги та замовлення, а діаграми класів, кооперацій і компонентів підкреслили модульність і гнучкість архітектури системи. Використання RESTful API для обміну даними між клієнтською та серверною частинами сприяло швидкій обробці запитів і адаптивності інтерфейсу, що є важливим для сучасних веб-додатків. Огляд існуючих рішень, таких як Amazon чи Yakaboo, підтвердив конкурентні переваги розробленої системи, зокрема її локалізацію для українського ринку та простоту використання, хоча й виявив потенціал для вдосконалення в плані персоналізації та масштабованості.

Практична реалізація проєкту втілилася у двох основних компонентах — client.py та server.py, які разом формують функціональну платформу. Клієнтська частина забезпечує інтуїтивно зрозумілий інтерфейс для перегляду каталогу, управління кошиком і оформлення замовлень, тоді як серверна логіка обробляє автентифікацію, транзакції та адміністрування. Використання SQLite як бази даних виявилося ефективним для прототипу, однак для реального впровадження потребує переходу на більш потужні СУБД, такі як PostgreSQL, щоб підтримувати більшу кількість користувачів і даних. Безпека системи, реалізована через хешування паролів і JWT-токени, відповідає базовим стандартам, але потребує додаткового шифрування чутливих даних для повного захисту.

Результати роботи свідчать про успішне досягнення поставленої мети — створення прототипу, який демонструє ключові принципи електронної комерції та може бути адаптований для реального використання. Система не лише спрощує доступ до книжкової продукції, але й сприяє популяризації читання через зручний онлайн-формат. Водночас проєкт відкриває перспективи для подальшого розвитку, зокрема інтеграції платіжних систем, аналітики продажів і розширення функціоналу для менеджерів. Практичне значення роботи

підкріплюється її навчальним потенціалом, адже вона може слугувати основою для вивчення технологій веб-розробки. Таким чином, розроблене програмне забезпечення є збалансованим рішенням, яке враховує потреби користувачів і відповідає сучасним тенденціям у сфері онлайн-торгівлі, залишаючи простір для майбутніх удосконалень.

ІНСТРУКЦІЯ КОРИСТУВАЧА

Крок 1. Запуск

На головній сторінці в бічній панелі виберіть пункт меню "Реєстрація" (рис. Б.1) для створення нового облікового запису або "Вхід" (рис. Б.2) для авторизації. У разі реєстрації заповніть поля: ім’я користувача, електронна пошта, пароль, підтвердження пароля, роль ("client" або "manager"), а також необов’язкові дані (повне ім’я, адреса, телефон). Натисніть кнопку "Зареєструватися". Після успішної реєстрації система перенаправить вас на сторінку входу. Для авторизації введіть ім’я користувача та пароль і натисніть "Увійти". Успішний вхід відкриває доступ до каталогу книг та персоналізованих функцій.

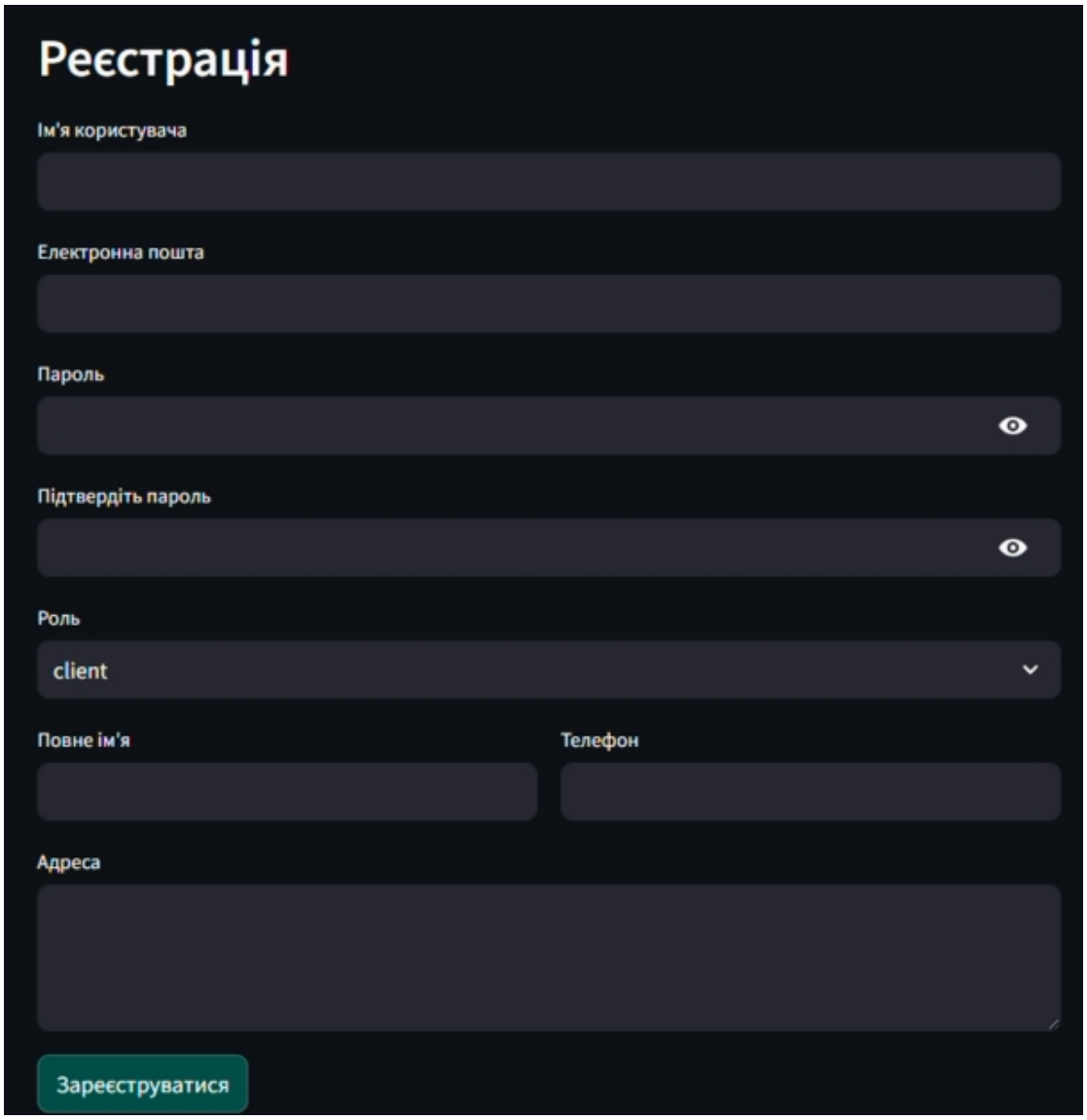

Рис. Б.1. Форма реєстрації

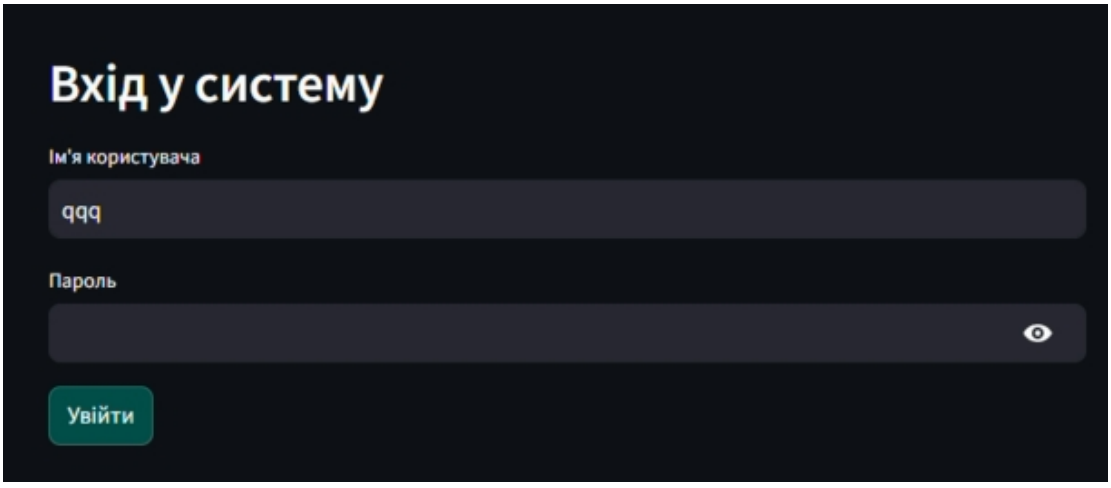

Рис. Б.2. Вхід у систему

Крок 2. Перегляд каталогу та вибір товарів

Після входу система автоматично відображає сторінку "Каталог" (рис. Б.3). Тут доступні фільтри за категоріями (випадаючий список) та пошуковий рядок (рис. Б.4) для введення назви чи автора книги. Перегляньте список книг, представлених у вигляді карток із назвою, автором, ціною та наявністю. Для детальнішого ознайомлення натисніть "Деталі" біля обраної книги (рис. Б.5), що відкриє сторінку з описом та зображенням (за наявності). Щоб додати книгу до кошика, натисніть "До кошика" на сторінці каталогу або деталей книги. Повідомлення підтвердить додавання товару.

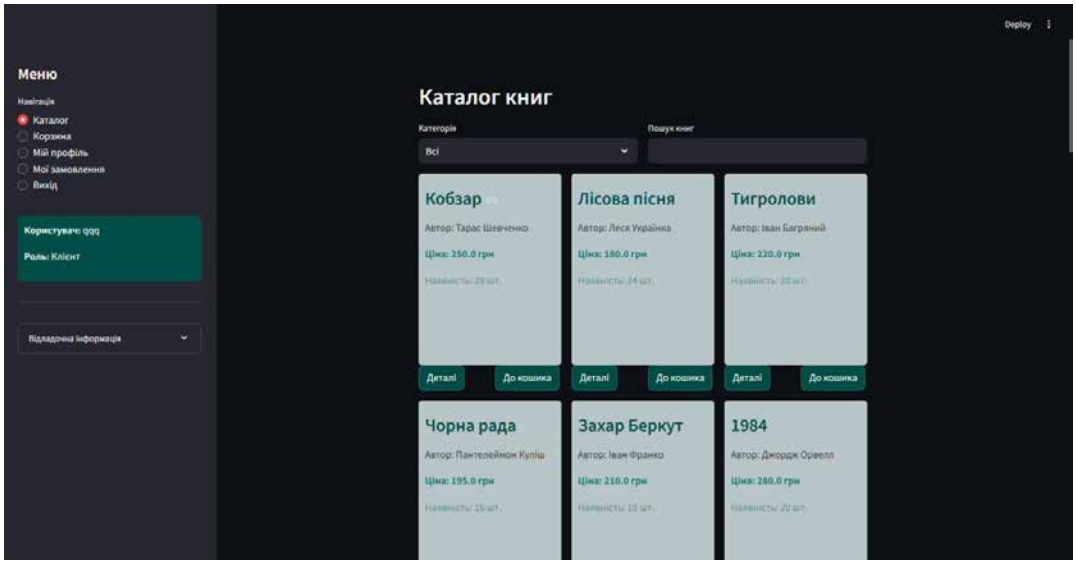

Рис. Б.3. Каталог книг

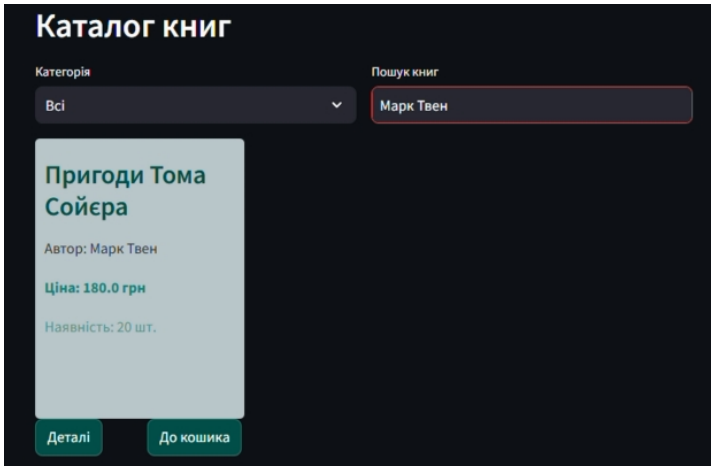

Рис. Б.4. Пошук книг

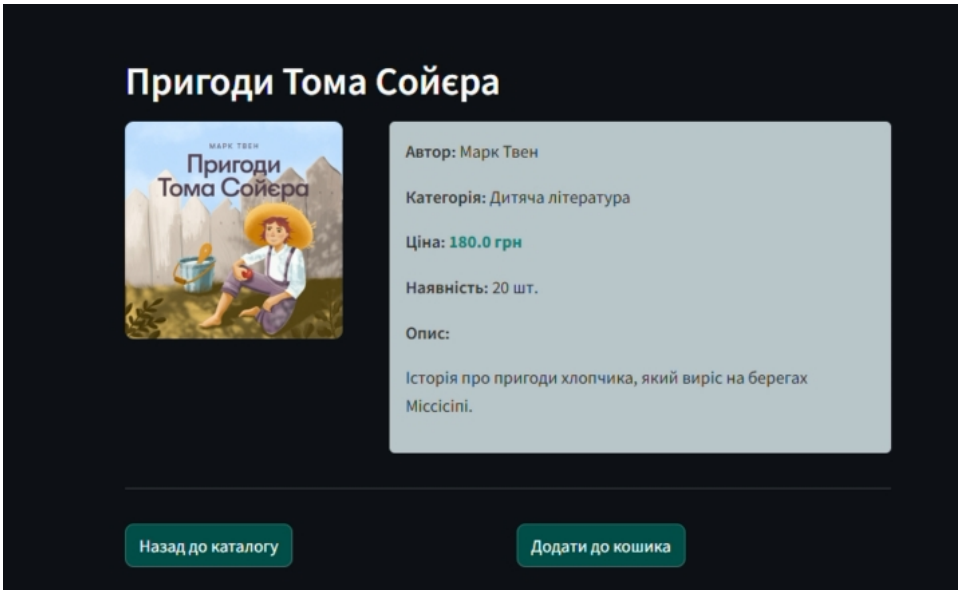

Рис. Б.5. Деталі книги

Крок 3. Робота з кошиком

Перейдіть до кошика, вибравши пункт "Корзина" в меню. У кошику відображаються додані книги із зазначенням назви, ціни, кількості та загальної суми (рис. Б.6). Змініть кількість товарів за допомогою числового поля або видаліть позицію кнопкою "Видалити". Для продовження покупок натисніть "Продовжити покупки", щоб повернутися до каталогу. Щоб оформити замовлення, натисніть "Оформити замовлення". У разі успіху кошик очиститься, а система перенаправить вас до розділу "Мої замовлення".

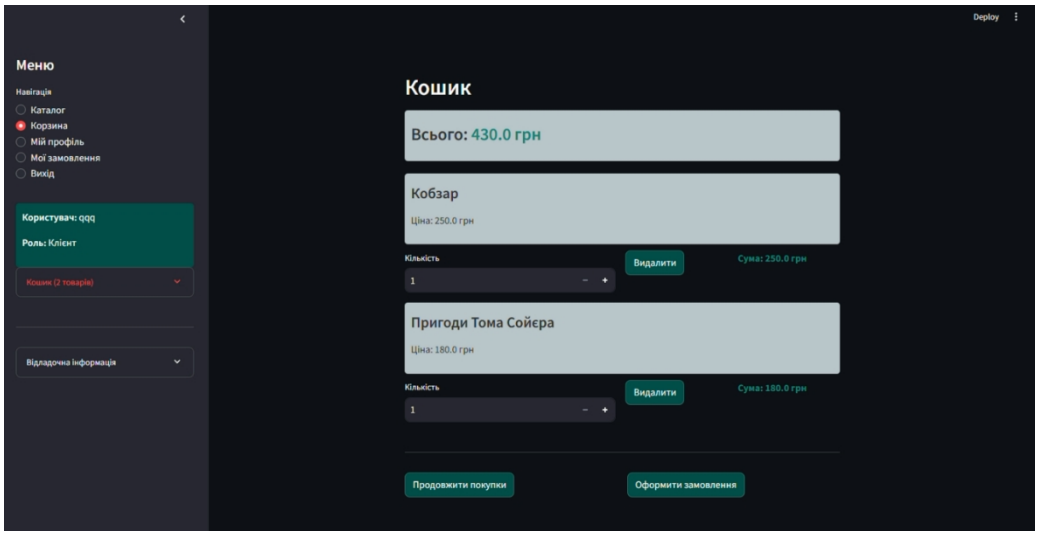

Рис. Б.6 – Перегляд кошика

Крок 4. Перегляд та управління замовленнями

У меню виберіть "Мої замовлення" (рис. Б.7) для перегляду списку замовлень (для клієнтів) або "Управління замовленнями" (для менеджерів) (рис. Б.8). Клієнти можуть натиснути «Переглянути деталі» для ознайомлення з інформацією про замовлення (дата, статус, товари, сума). Менеджери додатково мають змогу змінювати статус замовлення (наприклад, «нове», «оброблене», «відправлене») через випадаючий список і кнопку «Зберегти статус». Усі зміни зберігаються автоматично.

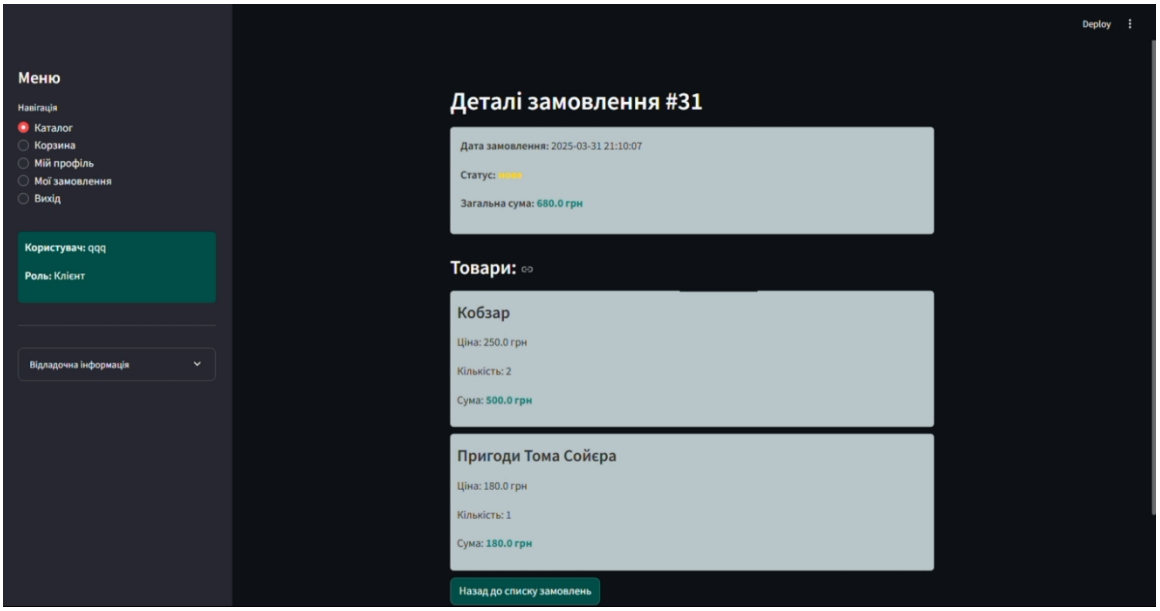

Рис. Б.7. Замовлення клієнта

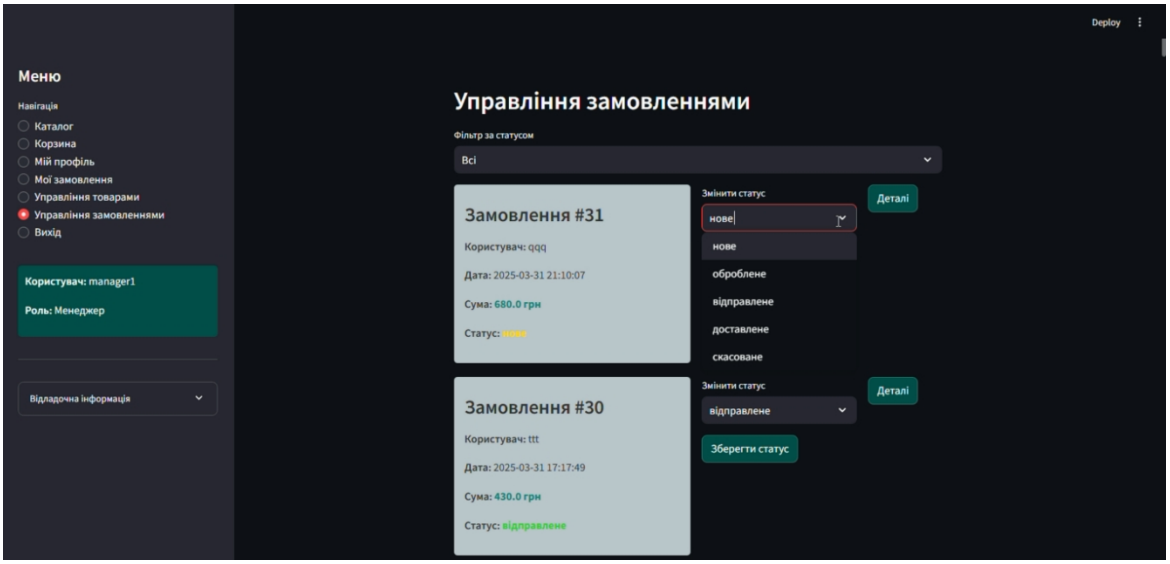

Рис. Б.8. Управління замовленнями для менеджера

Крок 5. Управління профілем

Для редагування особистих даних виберіть "Мій профіль" (рис. Б.9) у меню. Перегляньте поточну інформацію (ім’я, email, роль) та розгорніть розділ "Редагувати профіль" (рис. Б.10). Внесіть зміни до полів (email, повне ім’я, телефон, адреса, новий пароль) і натисніть "Зберегти зміни". Успішне оновлення підтверджується повідомленням. Цей функціонал дозволяє підтримувати актуальність контактних даних.

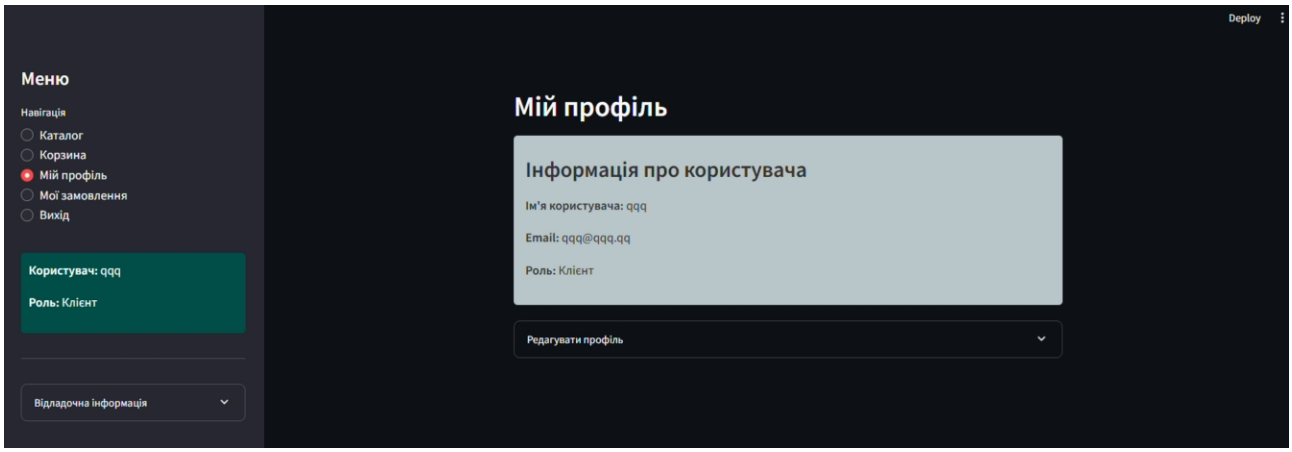

Рис. Б.9. Перегляд профілю

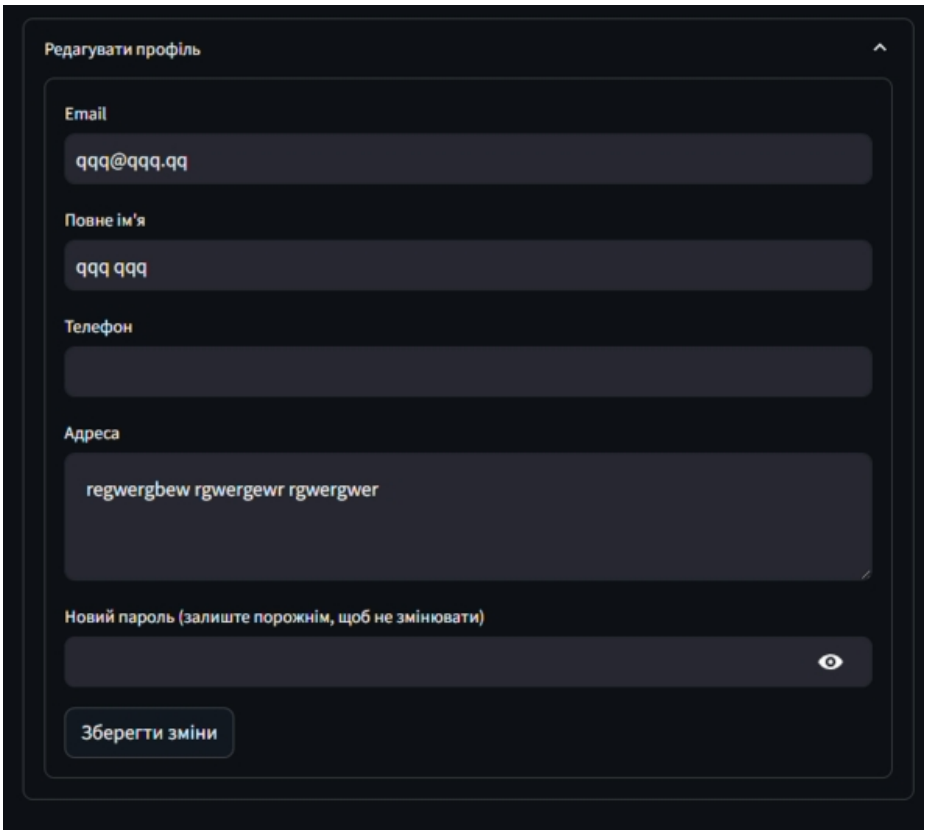

Рис. Б.10. Редагування профілю

Крок 6. Управління товарами (для менеджерів)

Менеджери можуть обрати "Управління товарами" в меню (рис. Б.11). У вкладці "Список книг" відображаються всі товари з опціями "Редагувати" (рис. Б.12) та "Видалити" (рис. Б.13). Для додавання нової книги перейдіть до вкладки "Додати нову книгу" (рис. Б.14), заповніть форму (назва, автор, ціна, кількість тощо) і натисніть "Додати книгу". У режимі редагування змініть дані книги та збережіть їх. Усі дії супроводжуються повідомленнями про результат.

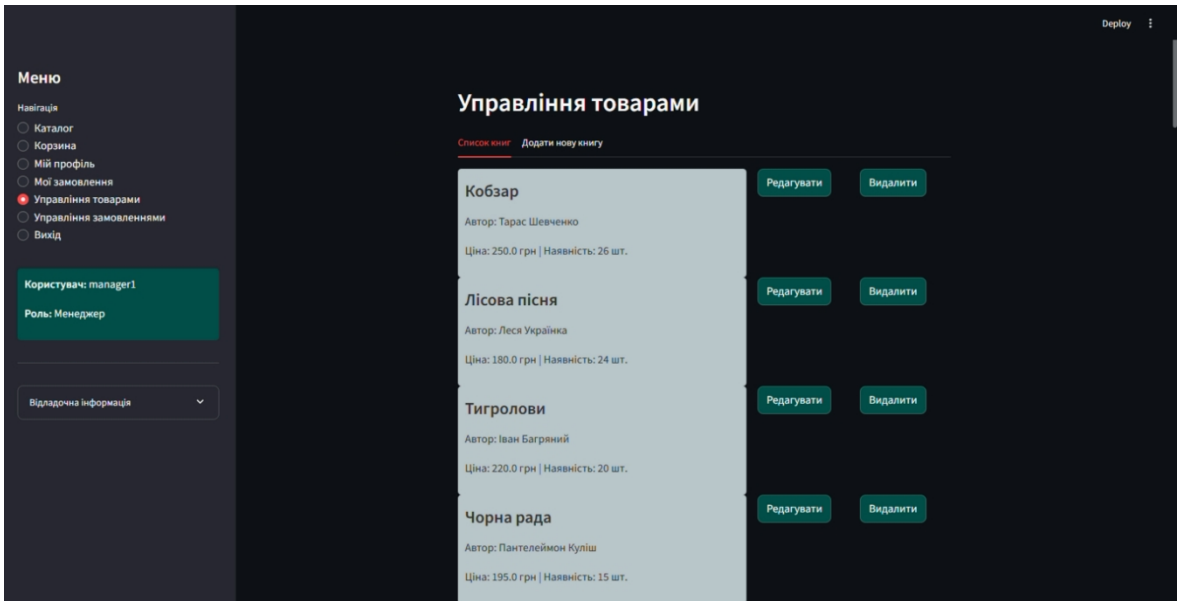

Рис. Б.11. Вкладка управління товарами

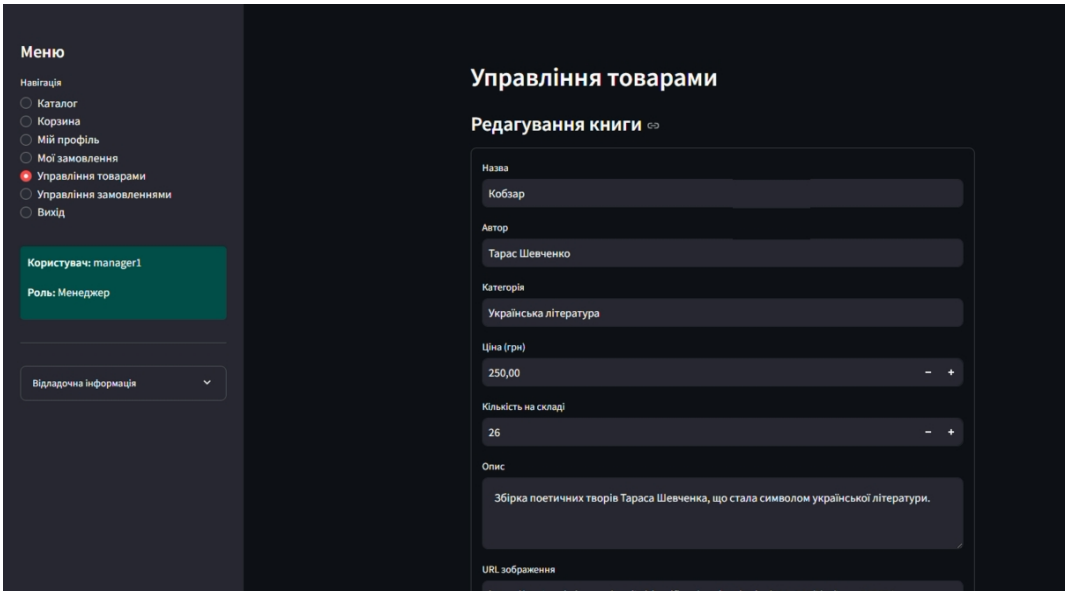

Рис. Б.12. Форма редагування товарів

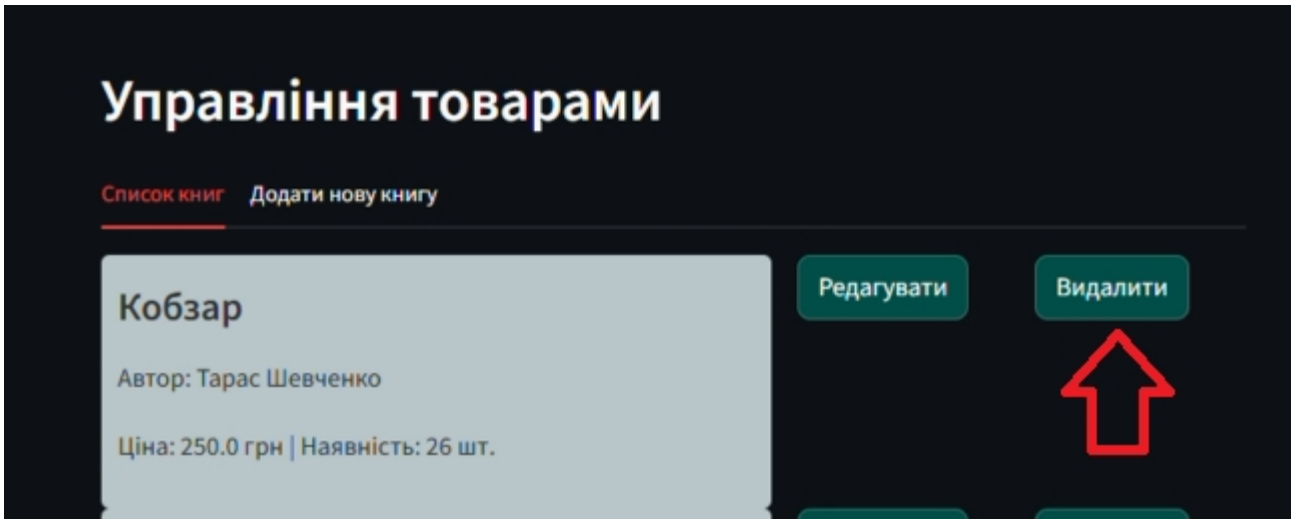

Рис. Б.13. Кнопка видалення товарів

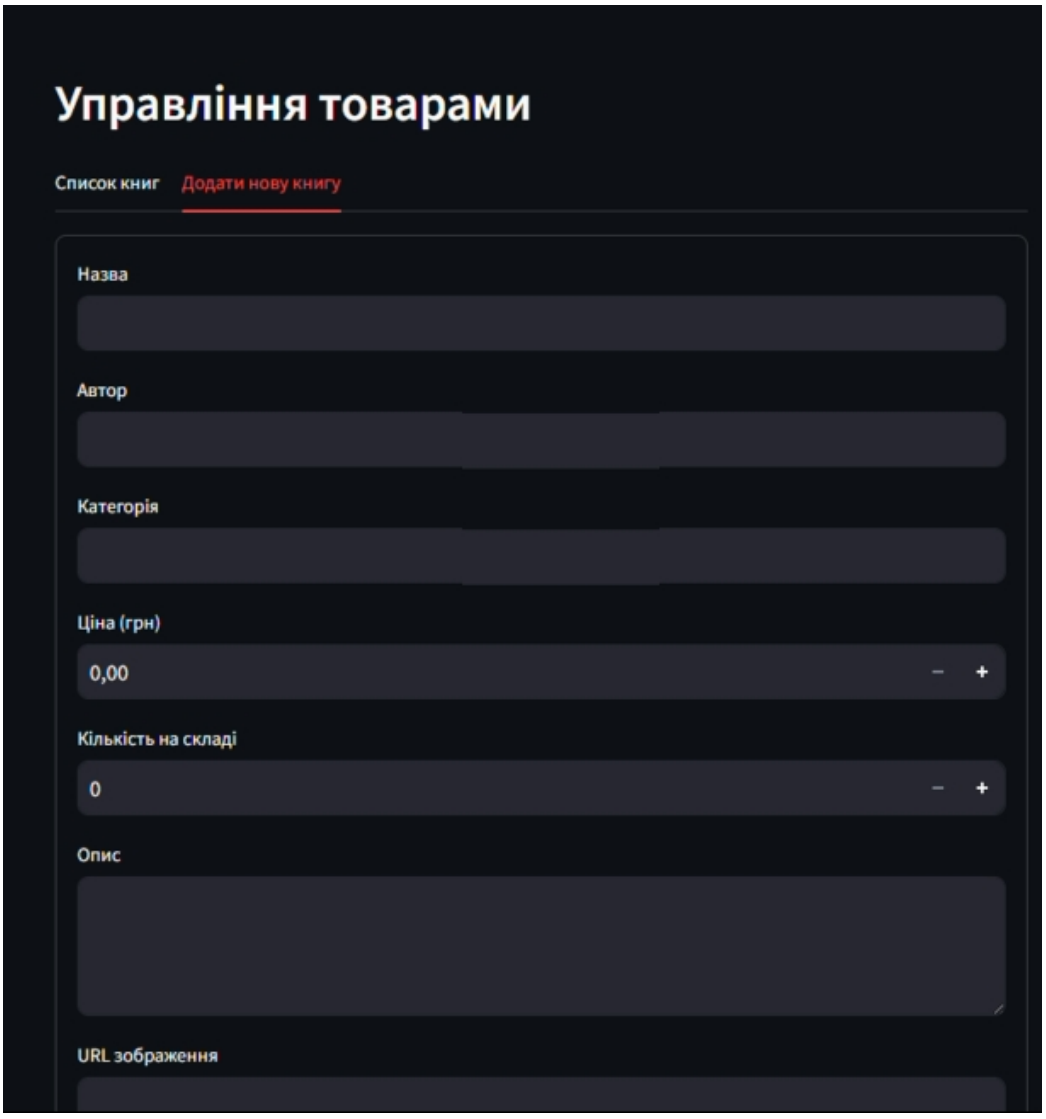

Рис. Б.14. Форма додавання нової книги

Крок 7. Завершення роботи

Для завершення сеансу виберіть "Вихід" у меню (рис. Б.1). Система очистить дані сесії (кошик, токен авторизації) і перенаправить вас на сторінку каталогу в гостьовому режимі.

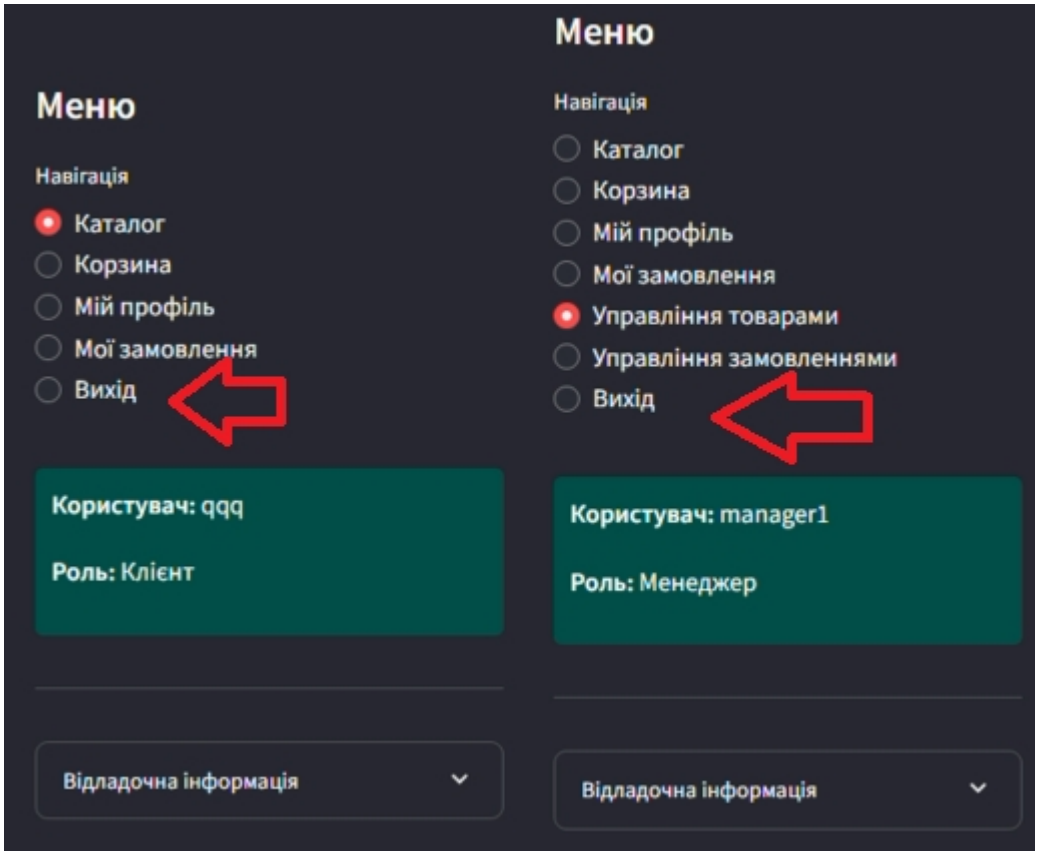

Рис. Б.15. Вихід із системи

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Крилов Д. В. Розвиток електронної комерції в Україні в сучасних умовах. Проблеми сучасних трансформацій Серія економіка та управління. 2024

2. Попова Н. В., Катаєв А. В., Базалієва Л. В., Кононов О. І., Муха Т. А. МАРКЕТИНГОВІ КОМУНІКАЦІЇ. Підручник. Харків. 2020

3. Що таке фронтенд розробка: складові, етапи та технології URL: https://wezom.com.ua/ua/blog/chto-takoe-front-end-razrabotka

4. Що таке бекенд і навіщо його розуміння бізнесу? URL: https://asabix.com.ua/the-backbone-of-business-what-is-backend/

5. Використання транзакцій в реляційних базах даних. Досвід дата-інженерки з Boosters URL: https://journal.gen.tech/post/tranzakciy_v-relyatsiynykh-bazakh-danykh

6. Аутентифікація і авторизація: що це і в чому відмінність URL: https://qagroup.com.ua/publications/autentyfikatciia-i-avtoryzatciia/

7. Як правильно організувати каталог інтернет-магазину URL: https://wezom.com.ua/ua/blog/kak-pravilno-organizovat-katalog-internet-magazina

8. Функції кошика в інтернет-магазині URL: https://simferopil.rozrobka-sajtiv.in.ua/rozrobka-sajtiv-blog/funkcziyi-koshyka-v-internet-magazyni/

9. Profile Management URL: https://docs.informatica.com/data-security-group/test-data-management/10-5/user-guide/data-discovery/profile-management.html

10. Контент-менеджмент та адміністрування сайтів URL: https://scratch-studio.com/kontent-menedzhment-ta-administruvannia-saitiv/

11. View Order History URL: https://help.salesforce.com/s/articleView?id=commerce.om_view_order_history.htm&type=5

12. Оптимізація продуктивності сайту: Поради та методи URL: https://www.ranktracker.com/uk/blog/optimizing-website-performance-tips-and-techniques/

13. Масштабованість як вимога до програмного забезпечення, значення та визначення URL: https://uk.itpedia.nl/2021/07/20/schaalbaarheid-als-software-requirement-betekenis-en-definitie/

14. Принципи юзабіліті веб-сайту URL:https://wezom.com.ua/ua/blog/12-sposobov-uluchshit-juzabiliti

15. Безпека Даних на Сайті: Кращі Практики Маркетингових Агентств для Наслідування URL: https://protocol.ua/ua/bezpeka_danih_na_sayti_krashchi_praktiki_marketingovih_agentstv_dlya_nasliduvannya/

16. A faster way to build and share data apps URL: https://streamlit.io/

17. Welcome to Flask’s documentation URL: https://flask.palletsprojects.com/en/stable/

18. Що таке SQlite? URL: https://freehost.com.ua/ukr/faq/wiki/chto-takoe-sqlite/

19. LocalStorage, sessionStorage URL: https://uk.javascript.info/localstorage

20. Що таке CRUD простими словами URL: https://highload.tech/uk/shho-take-crud-prostimi-slovami-funktsiyi-perevagi-ta-prikladi/

21. Система навігації сайту та особливості її створення URL: https://gl.ua/blog/systema-navihatsiyi-saytu-ta-osoblyvosti-yiyi-stvorennya

22. Ідентифікація сутності інформаційної системи обліку та її основні характерні риси URL: https://core.ac.uk/outputs/268453013/

23. 2.3.2. Визначення взаємозв'язків між сутностями URL: https://studfile.net/preview/7078141/page:4/

24. Предметна область, моделювання предметної області [URL: https://studfile.net/preview/5474325/page:2/

25. Що таке діаграма варіантів використання UML: символи, шаблони, інструмент і посібник URL: https://www.mindonmap.com/uk/blog/what-is-a-uml-use-case-diagram/

26. Діаграма послідовності (Sequence Diagrams) URL: https://www.maxzosim.com/sequence-diagrams/

27. Дізнайтеся все про діаграму активності UML з методами URL: https://www.mindonmap.com/uk/blog/uml-activity-diagram/

28. Моделювання даних (Data Modelling) URL: https://www.maxzosim.com/data-modelling/

29. Laudon, K. C., & Traver, C. G. E-commerce: Business, Technology, Society Published by Pearson (May 1, 2023) © 2024

30. Sommerville, I. Software Engineering. Pearson Studium. p. 896. 2018

31. Chaffey Dave. Digital Business i E-Commerce Management. PWN. p. 666. 2016

32. Fowler, M. Patterns of Enterprise Application Architecture. 2002

33. Rosenfeld, L., & Morville, P. Information Architecture for the World Wide Web. O'Reilly & Associates, Inc.103A Morris St. Sebastopol, CAUnited States. P. 350. 2002

34. Stair, R., & Reynolds, G. Principles of Information Systems. Publisher Cengage Learning. 13th Edition. P. 752. 2017

35. Amazon URL: https://www.amazon.com/gp/browse.html/ref=glow_cls?node=283155&nodl=0&ref=books_dsk_sn_books-logo-1b731

36. Yakaboo URL: https://www.yakaboo.ua/

37. AbeBooks URL: https://www.abebooks.com/

38. Booktopia URL: https://www.booktopia.com.au/

39. What is an Entity Relationship Diagram (ERD)? URL: https://www.lucidchart.com/pages/er-diagrams

40. Що таке діаграма класів UML і найкращий творець діаграм класів UML URL: https://www.mindonmap.com/uk/blog/what-is-uml-class-diagram/

41. Діаграма пакетів URL: https://uk.wikipedia.org/wiki/Діаграма_пакетів

42. Повне розуміння діаграми компонентів UML за допомогою легкого методу URL: https://www.mindonmap.com/uk/blog/uml-component-diagram/

43. Діаграма розгортання: Підручник з UML із ПРИКЛАДОМ URL: https://www.guru99.com/uk/deployment-diagram-uml-example.html

ДОДАТОК А

ЛІСТИНГ

ПРОГРАМИ

client.py

```
from time import time
import streamlit as st
import requests
import json
from datetime import datetime
import pandas as pd
# Налаштування стилів та кольорової гами
primary_color = "#004D47"  # Синій павлин
secondary_color = "#128277"  # Глибока вода
accent_color = "#52958B"  # Лишайник
light_color = "#B9C4C9"  # Туман
text_color = "#FFFFFF"  # Білий текст для темного фону
dark_text_color = "#333333"  # Темний текст для світлого фону
# Ініціалізація сесійних станів
if 'cart' not in st.session_state:
    st.session_state.cart = []
if 'user' not in st.session_state:
    st.session_state.user = None
if 'token' not in st.session_state:
    st.session_state.token = None
if 'page' not in st.session_state:
    st.session_state.page = "Каталог"
if 'book_id' not in st.session_state:
    st.session_state.book_id = None
if 'order_id' not in st.session_state:
    st.session_state.order_id = None
if 'redirect' not in st.session_state:
    st.session_state.redirect = False
if 'debug' not in st.session_state:
    st.session_state.debug = []
# URL сервера API
API_URL = "http://localhost:5000/api"
# Функція для відладочних виводів
def debug_print(message):
    """Додає повідомлення до відладочного логу в сесійному стані"""
    timestamp = datetime.now().strftime("%H:%M:%S")
    st.session_state.debug.append(f"[{timestamp}] {message}")
    # Обмежуємо кількість повідомлень у логу
    if len(st.session_state.debug) > 100:
```

```
        st.session_state.debug = st.session_state.debug[-100:]
# Ключова функція для переходів між сторінками
def set_page(page, book_id=None, order_id=None):
    # Запам'ятаємо поточну сторінку для перевірки кільцевих переходів
    current_page = st.session_state.page
    debug_print(f"Перехід зі сторінки '{current_page}' на '{page}'")
    # Встановлюємо нову сторінку
    st.session_state.page = page
    if book_id is not None:
        st.session_state.book_id = book_id
        debug_print(f"Встановлено book_id: {book_id}")
    if order_id is not None:
        st.session_state.order_id = order_id
        debug_print(f"Встановлено order_id: {order_id}")
    # Встановлюємо прапор переспрямування тільки якщо сторінка змінилася
    if current_page != page:
        st.session_state.redirect = True
        debug_print(f"Перезавантаження сторінки")
        st.rerun()
def update_order_status_direct(order_id, new_status):
    debug_print(f"Виклик update_order_status_direct: order_id={order_id},
new_status={new_status}")
    result, status_code = update_order_status(order_id, new_status)
    debug_print(f"Результат update_order_status: status_code={status_code},
result={result}")
    if status_code == 200:
        # Збережемо прапор успішного оновлення
        st.session_state.status_updated = True
        # Видаляємо можливий прапор помилки
        if 'status_update_error' in st.session_state:
            del st.session_state.status_update_error
        debug_print("Статус успішно оновлено")
        return True
    else:
        # Зберігаємо повідомлення про помилку
        st.session_state.status_update_error = result.get("message", "Помилка при оновленні
статусу")
        # Видаляємо можливий прапор успіху
        if 'status_updated' in st.session_state:
            del st.session_state.status_updated
        debug_print(f"Помилка оновлення статусу: {st.session_state.status_update_error}")
        return False

# Функції для роботи з API
def get_auth_header():
    if st.session_state.token:
        debug_print("Отримано токен авторизації")
        return {"Authorization": f"Bearer {st.session_state.token}"}
    debug_print("Токен авторизації відсутній")
    return {}
def register_user(username, password, email, role, full_name='', address='', phone=''):
    debug_print(f"Реєстрація користувача: username={username}, email={email}, role={role}")
    try:
        response = requests.post(f"{API_URL}/register", json={
            "username": username,
            "password": password,
            "email": email,
            "role": role,
            "full_name": full_name,
            "address": address,
            "phone": phone
        })
        data = response.json()
        debug_print(f"Відповідь сервера: status_code={response.status_code}, data={data}")
        if response.status_code == 201 and data.get("success"):
```

```
            # Зберігаємо дані в сесію
            st.session_state.token = data.get("token", "")
            st.session_state.user = {
                "user_id": data.get("user_id"),
                "username": username,
                "role": role
            }
            debug_print(f"Користувача успішно зареєстровано: user_id={data.get('user_id')}")
        return data, response.status_code
    except requests.exceptions.ConnectionError:
        debug_print("Помилка з'єднання з сервером")
        return {"success": False, "message": "Неможливо підключитися до сервера"}, 500
def login_user(username, password):
    debug_print(f"Спроба входу: username={username}")
    try:
        response = requests.post(f"{API_URL}/login", json={
            "username": username,
            "password": password
        })
        debug_print(f"Відповідь сервера: status_code={response.status_code}")
        if response.status_code == 200:
            data = response.json()
            # Зберігаємо токен та інформацію про користувача
            st.session_state.token = data.get("token", "")
            st.session_state.user = {
                "user_id": data.get("user_id"),
                "username": data.get("username"),
                "role": data.get("role")
            }
            debug_print(f"Успішний вхід: user_id={data.get('user_id')},
role={data.get('role')}")
            return True
        debug_print("Невдалий вхід")
        return False
    except requests.exceptions.ConnectionError:
        debug_print("Помилка з'єднання з сервером")
        st.error("Неможливо підключитися до сервера")
        return False
def logout_user():
    debug_print("Вихід користувача")
    try:
        headers = get_auth_header()
        requests.post(f"{API_URL}/logout", headers=headers)
        debug_print("Запит на вихід надіслано до сервера")
    except requests.exceptions.ConnectionError:
        debug_print("Помилка з'єднання з сервером при виході")
        pass
    finally:
        # Завжди очищаємо дані сесії
        st.session_state.user = None
        st.session_state.token = None
        st.session_state.cart = []
        debug_print("Дані сесії очищено")
def get_user_profile():
    debug_print("Запит профілю користувача")
    try:
        headers = get_auth_header()
        response = requests.get(f"{API_URL}/user/profile", headers=headers)
        debug_print(f"Відповідь сервера: status_code={response.status_code}")
        if response.status_code == 200:
            profile = response.json()
            debug_print("Профіль користувача отримано успішно")
            return profile
        elif response.status_code == 401:
            debug_print("Помилка авторизації при отриманні профілю")
```

```
            st.warning("Сесія закінчилася. Будь ласка, увійдіть знову.")
            st.session_state.user = None
            st.session_state.token = None
            set_page("Вхід")
        return None
    except requests.exceptions.ConnectionError:
        debug_print("Помилка з'єднання з сервером")
        st.error("Неможливо підключитися до сервера")
        return None
def update_user_profile(data):
    debug_print(f"Оновлення профілю користувача: {data}")
    try:
        headers = get_auth_header()
        response = requests.put(f"{API_URL}/user/profile", json=data, headers=headers)
        debug_print(f"Відповідь сервера: status_code={response.status_code}")
        if response.status_code == 401:
            debug_print("Помилка авторизації при оновленні профілю")
            st.warning("Сесія закінчилася. Будь ласка, увійдіть знову.")
            st.session_state.user = None
            st.session_state.token = None
            set_page("Вхід")
        return response.json(), response.status_code
    except requests.exceptions.ConnectionError:
        debug_print("Помилка з'єднання з сервером")
        return {"success": False, "message": "Неможливо підключитися до сервера"}, 500
def get_books(category=None, search=None):
    debug_print(f"Запит списку книг: category={category}, search={search}")
    try:
        params = {}
        if category:
            params['category'] = category
        if search:
            params['search'] = search
        response = requests.get(f"{API_URL}/books", params=params)
        debug_print(f"Відповідь сервера: status_code={response.status_code}")
        if response.status_code == 200:
            books = response.json()
            debug_print(f"Отримано {len(books)} книг")
            return books
        debug_print("Помилка при отриманні книг")
        return []
    except requests.exceptions.ConnectionError:
        debug_print("Помилка з'єднання з сервером")
        st.error("Неможливо підключитися до сервера")
        return []
def get_book(book_id):
    debug_print(f"Запит книги з ID={book_id}")
    try:
        response = requests.get(f"{API_URL}/books/{book_id}")
        debug_print(f"Відповідь сервера: status_code={response.status_code}")
        if response.status_code == 200:
            book = response.json()
            debug_print(f"Книгу отримано: {book['title']}")
            return book
        debug_print("Книгу не знайдено")
        return None
    except requests.exceptions.ConnectionError:
        debug_print("Помилка з'єднання з сервером")
        st.error("Неможливо підключитися до сервера")
        return None
def add_book(data):
    debug_print(f"Додавання нової книги: {data}")
    try:
        headers = get_auth_header()
        response = requests.post(f"{API_URL}/books", json=data, headers=headers)
```

```
            debug_print(f"Відповідь сервера: status_code={response.status_code}")
            if response.status_code == 401:
                debug_print("Помилка авторизації при додаванні книги")
                st.warning("Сесія закінчилася. Будь ласка, увійдіть знову.")
                st.session_state.user = None
                st.session_state.token = None
                set_page("Вхід")
            return response.json(), response.status_code
        except requests.exceptions.ConnectionError:
            debug_print("Помилка з'єднання з сервером")
            return {"success": False, "message": "Неможливо підключитися до сервера"}, 500
    def update_book(book_id, data):
        debug_print(f"Оновлення книги з ID={book_id}: {data}")
        try:
            headers = get_auth_header()
            response = requests.put(f"{API_URL}/books/{book_id}", json=data, headers=headers)
            debug_print(f"Відповідь сервера: status_code={response.status_code}")
            if response.status_code == 401:
                debug_print("Помилка авторизації при оновленні книги")
                st.warning("Сесія закінчилася. Будь ласка, увійдіть знову.")
                st.session_state.user = None
                st.session_state.token = None
                set_page("Вхід")
            return response.json(), response.status_code
        except requests.exceptions.ConnectionError:
            debug_print("Помилка з'єднання з сервером")
            return {"success": False, "message": "Неможливо підключитися до сервера"}, 500
    def delete_book(book_id):
        debug_print(f"Видалення книги з ID={book_id}")
        try:
            headers = get_auth_header()
            response = requests.delete(f"{API_URL}/books/{book_id}", headers=headers)
            debug_print(f"Відповідь сервера: status_code={response.status_code}")
            if response.status_code == 401:
                debug_print("Помилка авторизації при видаленні книги")
                st.warning("Сесія закінчилася. Будь ласка, увійдіть знову.")
                st.session_state.user = None
                st.session_state.token = None
                set_page("Вхід")
            return response.json(), response.status_code
        except requests.exceptions.ConnectionError:
            debug_print("Помилка з'єднання з сервером")
            return {"success": False, "message": "Неможливо підключитися до сервера"}, 500
    def create_order(items):
        debug_print(f"Створення замовлення з {len(items)} товарами")
        try:
            headers = get_auth_header()
            response = requests.post(f"{API_URL}/orders", json={"items": items},
    headers=headers)
            debug_print(f"Відповідь сервера: status_code={response.status_code}")
            if response.status_code == 401:
                debug_print("Помилка авторизації при створенні замовлення")
                st.warning("Сесія закінчилася. Будь ласка, увійдіть знову.")
                st.session_state.user = None
                st.session_state.token = None
                set_page("Вхід")
            return response.json(), response.status_code
        except requests.exceptions.ConnectionError:
            debug_print("Помилка з'єднання з сервером")
            return {"success": False, "message": "Неможливо підключитися до сервера"}, 500
    def get_orders():
        debug_print("Запит списку замовлень")
        try:
            headers = get_auth_header()
            response = requests.get(f"{API_URL}/orders", headers=headers)
```

```
        debug_print(f"Відповідь сервера: status_code={response.status_code}")
        if response.status_code == 200:
            orders = response.json()
            debug_print(f"Отримано {len(orders)} замовлень")
            return orders
        elif response.status_code == 401:
            debug_print("Помилка авторизації при отриманні замовлень")
            st.warning("Сесія закінчилася. Будь ласка, увійдіть знову.")
            st.session_state.user = None
            st.session_state.token = None
            set_page("Вхід")
        return []
    except requests.exceptions.ConnectionError:
        debug_print("Помилка з'єднання з сервером")
        st.error("Неможливо підключитися до сервера")
        return []
def get_order(order_id):
    debug_print(f"Запит замовлення з ID={order_id}")
    try:
        headers = get_auth_header()
        response = requests.get(f"{API_URL}/orders/{order_id}", headers=headers)
        debug_print(f"Відповідь сервера: status_code={response.status_code}")
        if response.status_code == 200:
            order = response.json()
            debug_print(f"Замовлення отримано: ID={order['id']}, статус={order['status']}")
            return order
        elif response.status_code == 401:
            debug_print("Помилка авторизації при отриманні замовлення")
            st.warning("Сесія закінчилася. Будь ласка, увійдіть знову.")
            st.session_state.user = None
            st.session_state.token = None
            set_page("Вхід")
        return None
    except requests.exceptions.ConnectionError:
        debug_print("Помилка з'єднання з сервером")
        st.error("Неможливо підключитися до сервера")
        return None
def update_order_status(order_id, status):
    debug_print(f"Оновлення статусу замовлення: order_id={order_id}, status={status}")
    try:
        headers = get_auth_header()
        response = requests.put(f"{API_URL}/orders/{order_id}/status", json={"status":
status}, headers=headers)
        debug_print(f"Відповідь сервера: status_code={response.status_code},
відповідь={response.text}")
        if response.status_code == 401:
            debug_print("Помилка авторизації при оновленні статусу замовлення")
            st.warning("Сесія закінчилася. Будь ласка, увійдіть знову.")
            st.session_state.user = None
            st.session_state.token = None
            set_page("Вхід")
        return response.json(), response.status_code
    except requests.exceptions.ConnectionError:
        debug_print("Помилка з'єднання з сервером")
        return {"success": False, "message": "Неможливо підключитися до сервера"}, 500
def get_categories():
    debug_print("Запит списку категорій")
    try:
        response = requests.get(f"{API_URL}/categories")
        debug_print(f"Відповідь сервера: status_code={response.status_code}")
        if response.status_code == 200:
            categories = response.json()
            debug_print(f"Отримано {len(categories)} категорій")
            return categories
        return []
```

```
    except requests.exceptions.ConnectionError:
        debug_print("Помилка з'єднання з сервером")
        st.error("Неможливо підключитися до сервера")
        return []
# Функції для роботи з кошиком
def add_to_cart(book):
    debug_print(f"Додавання до кошика: book_id={book['id']}, title={book['title']}")
    # Перевірка, чи книга вже в кошику
    for item in st.session_state.cart:
        if item['book_id'] == book['id']:
            item['quantity'] += 1
            debug_print(f"Книга вже в кошику, збільшено кількість до {item['quantity']}")
            return
    # Додавання нової книги до кошика
    st.session_state.cart.append({
        'book_id': book['id'],
        'title': book['title'],
        'price': book['price'],
        'quantity': 1
    })
    debug_print(f"Книгу додано до кошика, поточна кількість товарів:
{len(st.session_state.cart)}")
def remove_from_cart(index):
    debug_print(f"Видалення з кошика: index={index},
title={st.session_state.cart[index]['title']}")
    st.session_state.cart.pop(index)
    debug_print(f"Товар видалено, залишилося товарів: {len(st.session_state.cart)}")
def update_cart_quantity(index, quantity):
    debug_print(f"Оновлення кількості в кошику: index={index},
title={st.session_state.cart[index]['title']}, quantity={quantity}")
    if quantity > 0:
        st.session_state.cart[index]['quantity'] = quantity
        debug_print(f"Кількість оновлено до {quantity}")
    else:
        debug_print("Кількість <= 0, товар буде видалено")
        remove_from_cart(index)
def calculate_cart_total():
    total = sum(item['price'] * item['quantity'] for item in st.session_state.cart)
    debug_print(f"Розрахунок суми кошика: {total} грн")
    return total
# Сторінки додатку
def show_login_form():
    st.header("Вхід у систему")
    debug_print("Відображення форми входу")
    username = st.text_input("Ім'я користувача", autocomplete="username")
    password = st.text_input("Пароль", type="password", autocomplete="current-password")
    if st.button("Увійти"):
        debug_print(f"Натиснуто кнопку входу: username={username}")
        if username and password:
            if login_user(username, password):
                st.success("Успішний вхід!")
                debug_print("Успішний вхід, перехід до каталогу")
                set_page("Каталог")
            else:
                st.error("Невірне ім'я користувача або пароль")
                debug_print("Невдалий вхід: невірні облікові дані")
        else:
            st.error("Будь ласка, заповніть всі поля")
            debug_print("Невдалий вхід: порожні поля")
def show_register_form():
    st.header("Реєстрація")
    debug_print("Відображення форми реєстрації")
    username = st.text_input("Ім'я користувача", autocomplete="username")
    email = st.text_input("Електронна пошта", autocomplete="email")
    password = st.text_input("Пароль", type="password", autocomplete="new-password")
```

```
    confirm_password = st.text_input("Підтвердіть пароль", type="password",
autocomplete="new-password")
    role = st.selectbox("Роль", ["client", "manager"])
    col1, col2 = st.columns(2)
    with col1:
        full_name = st.text_input("Повне ім'я", autocomplete="name")
    with col2:
        phone = st.text_input("Телефон", autocomplete="tel")
    # Видалено атрибут autocomplete
    address = st.text_area("Адреса")
    if st.button("Зареєструватися"):
        debug_print(f"Натиснуто кнопку реєстрації: username={username}, email={email},
role={role}")
        if not all([username, email, password, confirm_password]):
            st.error("Будь ласка, заповніть всі обов'язкові поля")
            debug_print("Невдала реєстрація: заповнені не всі обов'язкові поля")
        elif password != confirm_password:
            st.error("Паролі не співпадають")
            debug_print("Невдала реєстрація: паролі не співпадають")
        else:
            result, status_code = register_user(username, password, email, role, full_name,
address, phone)
            if status_code == 201:
                st.success("Реєстрація успішна! Ви можете увійти в систему.")
                debug_print("Успішна реєстрація, перехід на сторінку входу")
                set_page("Вхід")
            else:
                st.error(result.get("message", "Помилка при реєстрації"))
                debug_print(f"Невдала реєстрація: {result.get('message', 'невідома
помилка')}")
def show_catalog():
    st.header("Каталог книг")
    debug_print("Відображення каталогу книг")
    # Отримання категорій
    categories = get_categories()
    # Фільтрація
    col1, col2 = st.columns(2)
    with col1:
        selected_category = st.selectbox("Категорія", ["Всі"] + categories)
        debug_print(f"Обрана категорія: {selected_category}")
    with col2:
        search_query = st.text_input("Пошук книг")
        if search_query:
            debug_print(f"Пошуковий запит: {search_query}")
    category_filter = selected_category if selected_category != "Всі" else None
    # Отримання книг з фільтрацією
    books = get_books(category=category_filter, search=search_query)
    if not books:
        st.info("Книги не знайдено")
        debug_print("Книги не знайдено за вказаними фільтрами")
        return
    debug_print(f"Відображення {len(books)} книг у каталозі")
    # Відображення книг
    for i in range(0, len(books), 3):
        cols = st.columns(3)
        for j in range(3):
            if i + j < len(books):
                book = books[i + j]
                with cols[j]:
                    st.markdown(f"""
                    <div style="background-color: {light_color}; padding: 10px; border-
radius: 5px; height: 300px; overflow: hidden;">
                        <h3 style="color: {primary_color};">{book['title']}</h3>
                        <p style="color: {dark_text_color};">Автор: {book['author']}</p>
```

```
                        <p style="color: {secondary_color}; font-weight: bold;">Ціна:
{book['price']} грн</p>
                        <p style="color: {accent_color};">Наявність: {book['stock']} шт.</p>
                    </div>
                    """, unsafe_allow_html=True)
                    col_a, col_b = st.columns(2)
                    with col_a:
                        if st.button("Деталі", key=f"details_{book['id']}"):
                            debug_print(f"Натиснуто кнопку 'Деталі' для книги
ID={book['id']}")
                            set_page("Деталі книги", book_id=book['id'])
                    with col_b:
                        if st.session_state.user and st.button("До кошика",
key=f"cart_{book['id']}"):
                            debug_print(f"Натиснуто кнопку 'До кошика' для книги
ID={book['id']}")
                            add_to_cart(book)
                            st.success(f"{book['title']} додано до кошика")
def show_book_details():
    book_id = st.session_state.book_id
    debug_print(f"Відображення деталей книги ID={book_id}")
    if not book_id:
        st.error("Книгу не знайдено")
        debug_print("Book_id не встановлено, перехід до каталогу")
        set_page("Каталог")
        return
    book = get_book(book_id)
    if not book:
        st.error("Книгу не знайдено")
        debug_print(f"Книгу з ID={book_id} не знайдено в базі даних, перехід до каталогу")
        set_page("Каталог")
        return
    st.header(book['title'])
    col1, col2 = st.columns([1, 2])
    with col1:
        if book.get('image_url'):
            st.image(book['image_url'], width=200)
            debug_print(f"Відображення зображення книги: {book['image_url']}")
        else:
            st.markdown(f"""
            <div style="background-color: {light_color}; width: 200px; height: 300px;
                display: flex; align-items: center; justify-content: center; color:
{dark_text_color};">
                Немає зображення
            </div>
            """, unsafe_allow_html=True)
            debug_print("Зображення книги відсутнє")
    with col2:
        st.markdown(f"""
        <div style="background-color: {light_color}; padding: 15px; border-radius: 5px;
color: {dark_text_color};">
            <p><strong>Автор:</strong> {book['author']}</p>
            <p><strong>Категорія:</strong> {book.get('category', 'Не вказано')}</p>
            <p><strong>Ціна:</strong> <span style="color: {secondary_color}; font-weight:
bold;">{book['price']} грн</span></p>
            <p><strong>Наявність:</strong> {book['stock']} шт.</p>
            <p><strong>Опис:</strong></p>
            <p>{book.get('description', 'Опис відсутній')}</p>
        </div>
        """, unsafe_allow_html=True)
    st.markdown("---")
    col3, col4 = st.columns(2)
    with col3:
        if st.button("Назад до каталогу"):
            debug_print("Натиснуто кнопку 'Назад до каталогу'")
```

```
            set_page("Каталог")
    with col4:
        if st.session_state.user and st.button("Додати до кошика"):
            debug_print(f"Натиснуто кнопку 'Додати до кошика' для книги ID={book['id']}")
            add_to_cart(book)
            st.success(f"{book['title']} додано до кошика")
def show_cart():
    st.header("Кошик")
    debug_print("Відображення кошика")
    if not st.session_state.user:
        st.warning("Для використання кошика необхідно увійти в систему")
        debug_print("Спроба доступу до кошика без авторизації")
        if st.button("Увійти"):
            debug_print("Натиснуто кнопку 'Увійти'")
            set_page("Вхід")
        return
    if not st.session_state.cart:
        st.info("Ваш кошик порожній")
        debug_print("Кошик порожній")
        if st.button("Перейти до каталогу"):
            debug_print("Натиснуто кнопку 'Перейти до каталогу'")
            set_page("Каталог")
        return
    total = calculate_cart_total()
    debug_print(f"Загальна сума кошика: {total} грн")
    st.markdown(f"""
    <div style="background-color: {light_color}; padding: 10px; border-radius: 5px; margin-
bottom: 20px; color: {dark_text_color};">
        <h3>Всього: <span style="color: {secondary_color};">{total} грн</span></h3>
    </div>
    """, unsafe_allow_html=True)
    for i, item in enumerate(st.session_state.cart):
        st.markdown(f"""
        <div style="background-color: {light_color}; padding: 10px; border-radius: 5px;
margin-bottom: 10px; color: {dark_text_color};">
            <h4>{item['title']}</h4>
            <p>Ціна: {item['price']} грн</p>
        </div>
        """, unsafe_allow_html=True)
        col1, col2, col3 = st.columns([2, 1, 1])
        with col1:
            new_quantity = st.number_input(f"Кількість", min_value=1,
value=item['quantity'], key=f"quantity_{i}")
            if new_quantity != item['quantity']:
                debug_print(f"Оновлення кількості товару в кошику: index={i}, з
{item['quantity']} на {new_quantity}")
                update_cart_quantity(i, new_quantity)
                st.rerun()
        with col2:
            if st.button("Видалити", key=f"remove_{i}"):
                debug_print(f"Видалення товару з кошика: index={i}, title={item['title']}")
                remove_from_cart(i)
                st.rerun()
        with col3:
            st.markdown(f"<p style='font-weight: bold; color: {secondary_color};'>Сума:
{item['price'] * item['quantity']} грн</p>", unsafe_allow_html=True)
    st.markdown("---")
    col1, col2 = st.columns(2)
    with col1:
        if st.button("Продовжити покупки"):
            debug_print("Натиснуто кнопку 'Продовжити покупки'")
            set_page("Каталог")
    with col2:
        if st.button("Оформити замовлення"):
            debug_print("Натиснуто кнопку 'Оформити замовлення'")
```

```
            result, status_code = create_order(st.session_state.cart)
            debug_print(f"Результат створення замовлення: status_code={status_code},
result={result}")
            if status_code == 201:
                st.session_state.cart = []
                st.success("Замовлення успішно оформлено!")
                debug_print("Замовлення успішно оформлено, перехід до списку замовлень")
                set_page("Мої замовлення")
            else:
                st.error(result.get("message", "Помилка при оформленні замовлення"))
                debug_print(f"Помилка при оформленні замовлення: {result.get('message',
'невідома помилка')}")
def show_profile():
    st.header("Мій профіль")
    debug_print("Відображення профілю користувача")
    if not st.session_state.user:
        st.warning("Для перегляду профілю необхідно увійти в систему")
        debug_print("Спроба доступу до профілю без авторизації")
        if st.button("Увійти"):
            debug_print("Натиснуто кнопку 'Увійти'")
            set_page("Вхід")
        return
    profile = get_user_profile()
    if not profile:
        st.error("Не вдалося завантажити дані профілю")
        debug_print("Помилка при завантаженні даних профілю")
        return
    debug_print(f"Профіль користувача завантажено: username={profile['username']},
role={profile['role']}")
    st.markdown(f"""
    <div style="background-color: {light_color}; padding: 15px; border-radius: 5px; margin-
bottom: 20px; color: {dark_text_color};">
        <h3>Інформація про користувача</h3>
        <p><strong>Ім'я користувача:</strong> {profile['username']}</p>
        <p><strong>Email:</strong> {profile['email']}</p>
        <p><strong>Роль:</strong> {"Менеджер" if profile['role'] == "manager" else
"Клієнт"}</p>
    </div>
    """, unsafe_allow_html=True)
    with st.expander("Редагувати профіль"):
        debug_print("Відкрито форму редагування профілю")
        with st.form("profile_form"):
            email = st.text_input("Email", value=profile['email'], autocomplete="email")
            full_name = st.text_input("Повне ім'я", value=profile.get('full_name', ''),
autocomplete="name")
            phone = st.text_input("Телефон", value=profile.get('phone', ''),
autocomplete="tel")
            # Видалено атрибут autocomplete
            address = st.text_area("Адреса", value=profile.get('address', ''))
            password = st.text_input("Новий пароль (залиште порожнім, щоб не змінювати)",
type="password", autocomplete="new-password")
            if st.form_submit_button("Зберегти зміни"):
                debug_print("Натиснуто кнопку 'Зберегти зміни' в формі профілю")
                data = {
                    'email': email,
                    'full_name': full_name,
                    'phone': phone,
                    'address': address
                }
                if password:
                    data['password'] = password
                    debug_print("Оновлення профілю з новим паролем")
                else:
                    debug_print("Оновлення профілю без зміни пароля")
                result, status_code = update_user_profile(data)
```

```
                debug_print(f"Результат оновлення профілю: status_code={status_code},
result={result}")
                if status_code == 200:
                    st.success("Профіль успішно оновлено")
                    debug_print("Профіль успішно оновлено")
                else:
                    st.error(result.get("message", "Помилка при оновленні профілю"))
                    debug_print(f"Помилка при оновленні профілю: {result.get('message',
'невідома помилка')}")
def show_orders():
    st.header("Мої замовлення")
    debug_print("Відображення списку замовлень")
    if not st.session_state.user:
        st.warning("Для перегляду замовлень необхідно увійти в систему")
        debug_print("Спроба доступу до замовлень без авторизації")
        if st.button("Увійти"):
            debug_print("Натиснуто кнопку 'Увійти'")
            set_page("Вхід")
        return
    orders = get_orders()
    if not orders:
        st.info("У вас ще немає замовлень")
        debug_print("Замовлення відсутні")
        if st.button("Перейти до каталогу"):
            debug_print("Натиснуто кнопку 'Перейти до каталогу'")
            set_page("Каталог")
        return
    debug_print(f"Завантажено {len(orders)} замовлень")
    for order in orders:
        status_color = {
            'нове': '#FFD700',  # Золотистий
            'оброблене': '#1E90FF',  # Синій
            'відправлене': '#32CD32',  # Зелений
            'доставлене': '#006400',  # Темно-зелений
            'скасоване': '#FF0000'  # Червоний
        }.get(order['status'].lower(), secondary_color)
        st.markdown(f"""
        <div style="background-color: {light_color}; padding: 15px; border-radius: 5px;
margin-bottom: 20px; color: {dark_text_color};">
            <h3>Замовлення #{order['id']}</h3>
            <p><strong>Дата:</strong> {order['order_date']}</p>
            <p><strong>Сума:</strong> <span style="color: {secondary_color}; font-weight:
bold;">{order['total_price']} грн</span></p>
            <p><strong>Статус:</strong> <span style="color: {status_color}; font-weight:
bold;">{order['status']}</span></p>
        </div>
        """, unsafe_allow_html=True)
        if st.button("Переглянути деталі", key=f"view_order_{order['id']}"):
            debug_print(f"Натиснуто кнопку 'Переглянути деталі' для замовлення
ID={order['id']}")
            set_page("Деталі замовлення", order_id=order['id'])
def show_order_details():
    if not st.session_state.order_id:
        st.error("Замовлення не знайдено")
        debug_print("Order_id не встановлено, перехід до списку замовлень")
        st.session_state.page = "Мої замовлення"
        st.rerun()
        return
    order_id = st.session_state.order_id
    debug_print(f"Відображення деталей замовлення ID={order_id}")
    order = get_order(order_id)
    if not order:
        st.error("Замовлення не знайдено")
        debug_print(f"Замовлення з ID={order_id} не знайдено в базі даних")
        st.session_state.page = "Мої замовлення"
```

```
            st.rerun()
            return
        st.header(f"Деталі замовлення #{order['id']}")
        status_color = {
            'нове': '#FFD700',  # Золотистий
            'оброблене': '#1E90FF',  # Синій
            'відправлене': '#32CD32',  # Зелений
            'доставлене': '#006400',  # Темно-зелений
            'скасоване': '#FF0000'  # Червоний
        }.get(order['status'].lower(), secondary_color)
        st.markdown(f"""
        <div style="background-color: {light_color}; padding: 15px; border-radius: 5px; margin-
bottom: 20px; color: {dark_text_color};">
            <p><strong>Дата замовлення:</strong> {order['order_date']}</p>
            <p><strong>Статус:</strong> <span style="color: {status_color}; font-weight:
bold;">{order['status']}</span></p>
            <p><strong>Загальна сума:</strong> <span style="color: {secondary_color}; font-
weight: bold;">{order['total_price']} грн</span></p>
        </div>
        """, unsafe_allow_html=True)
        st.subheader("Товари:")
        for item in order['items']:
            st.markdown(f"""
            <div style="background-color: {light_color}; padding: 10px; border-radius: 5px;
margin-bottom: 10px; color: {dark_text_color};">
                <h4>{item['title']}</h4>
                <p>Ціна: {item['price']} грн</p>
                <p>Кількість: {item['quantity']}</p>
                <p>Сума: <span style="color: {secondary_color}; font-weight:
bold;">{item['price'] * item['quantity']} грн</span></p>
            </div>
            """, unsafe_allow_html=True)
        # Кнопка повернення до списку замовлень розміщена на початку
        if st.button("Назад до списку замовлень", key="back_btn"):
            debug_print("Натиснуто кнопку 'Назад до списку замовлень'")
            st.session_state.page = "Мої замовлення"
            st.rerun()
        # Якщо користувач - менеджер, показуємо пряме оновлення статусу
        if st.session_state.user and st.session_state.user['role'] == 'manager':
            debug_print("Відображення форми зміни статусу замовлення (режим менеджера)")
            st.markdown("---")
            st.subheader("Зміна статусу замовлення")
            status_options = ["нове", "оброблене", "відправлене", "доставлене", "скасоване"]
            selected_status = st.selectbox(
                "Статус замовлення",
                status_options,
                index=status_options.index(order['status']) if order['status'] in status_options
else 0
            )
            debug_print(f"Обрано статус: {selected_status}, поточний статус: {order['status']}")
            # ВИПРАВЛЕНО: Викликаємо нашу функцію update_order_status_direct замість прямого
API-запиту
            if st.button("Змінити статус", key="update_status_btn"):
                debug_print(f"Натиснуто кнопку 'Змінити статус', новий статус:
{selected_status}")
                # Використовуємо нашу функцію, яка обертає API
                success = update_order_status_direct(order_id, selected_status)
                debug_print(f"Результат оновлення статусу: {success}")
                if success:
                    st.success(f"Статус змінено на '{selected_status}'")
                    debug_print("Статус успішно змінено, оновлення даних замовлення")
                    # Встановлюємо прапор для запобігання навігації під час перезавантаження
                    st.session_state.prevent_navigation = True
                    # Оновлюємо дані замовлення
                    order = get_order(order_id)
```

```
                # Немає необхідності перезавантажувати сторінку, оскільки Streamlit
автоматично перезавантажить її після натискання кнопки
            else:
                error_message = st.session_state.get('status_update_error', 'Помилка при
оновленні статусу')
                st.error(error_message)
                debug_print(f"Помилка при оновленні статусу: {error_message}")
                # Встановлюємо прапор для запобігання навігації під час перезавантаження
                st.session_state.prevent_navigation = True
def show_manage_books():
    st.header("Управління товарами")
    debug_print("Відображення сторінки управління товарами")
    if not st.session_state.user or st.session_state.user['role'] != 'manager':
        st.warning("Доступ дозволено тільки менеджерам")
        debug_print("Спроба доступу до управління товарами без прав менеджера")
        if st.button("Перейти до каталогу"):
            debug_print("Натиснуто кнопку 'Перейти до каталогу'")
            set_page("Каталог")
        return
    # Перевіряємо, чи вказано edit_book у query params
    query_params = st.query_params
    edit_book_id = query_params.get("edit_book_id", None)
    debug_print(f"Параметр edit_book_id з query params: {edit_book_id}")
    if edit_book_id:
        # Режим редагування книги
        debug_print(f"Перехід до редагування книги ID={edit_book_id}")
        show_edit_book_form(int(edit_book_id))
    else:
        # Звичайний режим управління книгами
        tab1, tab2 = st.tabs(["Список книг", "Додати нову книгу"])
        with tab1:
            debug_print("Відображення вкладки 'Список книг'")
            books = get_books()
            if not books:
                st.info("Книги не знайдено")
                debug_print("Книги не знайдено в базі даних")
            else:
                debug_print(f"Завантажено {len(books)} книг для відображення")
                for i, book in enumerate(books):
                    col1, col2, col3 = st.columns([3, 1, 1])
                    with col1:
                        st.markdown(f"""
                        <div style="background-color: {light_color}; padding: 10px; border-
radius: 5px; color: {dark_text_color};">
                            <h4>{book['title']}</h4>
                            <p>Автор: {book['author']}</p>
                            <p>Ціна: {book['price']} грн | Наявність: {book['stock']}
шт.</p>
                        </div>
                        """, unsafe_allow_html=True)
                    with col2:
                        if st.button("Редагувати", key=f"edit_book_{book['id']}_{i}"):
                            debug_print(f"Натиснуто кнопку 'Редагувати' для книги
ID={book['id']}")
                            # Замість experimental_set_query_params використовуємо оновлений
підхід
                            query_params["edit_book_id"] = book['id']
                            debug_print(f"Встановлено query параметр
edit_book_id={book['id']}")
                            st.rerun()
                    with col3:
                        if st.button("Видалити", key=f"delete_book_{book['id']}_{i}"):
                            debug_print(f"Натиснуто кнопку 'Видалити' для книги
ID={book['id']}")
                            result, status_code = delete_book(book['id'])
```

```
                            debug_print(f"Результат видалення книги:
status_code={status_code}, result={result}")
                            if status_code == 200:
                                st.success("Книгу успішно видалено")
                                debug_print("Книгу успішно видалено, перезавантаження
сторінки")
                                st.rerun()
                            else:
                                st.error(result.get("message", "Помилка при видаленні
книги"))
                                debug_print(f"Помилка при видаленні книги:
{result.get('message', 'невідома помилка')}")
        with tab2:
            debug_print("Відображення вкладки 'Додати нову книгу'")
            with st.form("add_book_form"):
                title = st.text_input("Назва")
                author = st.text_input("Автор")
                category = st.text_input("Категорія")
                price = st.number_input("Ціна (грн)", min_value=0.0, step=0.1)
                stock = st.number_input("Кількість на складі", min_value=0, step=1)
                description = st.text_area("Опис")
                image_url = st.text_input("URL зображення")
                if st.form_submit_button("Додати книгу"):
                    debug_print(f"Натиснуто кнопку 'Додати книгу': title={title},
author={author}")
                    if not title or not author:
                        st.error("Назва та автор обов'язкові")
                        debug_print("Помилка: не заповнені обов'язкові поля")
                    else:
                        result, status_code = add_book({
                            'title': title,
                            'author': author,
                            'category': category,
                            'price': price,
                            'stock': stock,
                            'description': description,
                            'image_url': image_url
                        })
                        debug_print(f"Результат додавання книги: status_code={status_code},
result={result}")
                        if status_code == 201:
                            st.success("Книгу успішно додано")
                            debug_print("Книгу успішно додано, перезавантаження сторінки")
                            st.rerun()
                        else:
                            st.error(result.get("message", "Помилка при додаванні книги"))
                            debug_print(f"Помилка при додаванні книги:
{result.get('message', 'невідома помилка')}")
def show_edit_book_form(book_id):
    st.subheader("Редагування книги")
    debug_print(f"Відображення форми редагування книги ID={book_id}")
    # Отримуємо дані книги
    book = get_book(book_id)
    if not book:
        st.error("Книгу не знайдено")
        debug_print(f"Книгу з ID={book_id} не знайдено в базі даних")
        if st.button("Повернутися"):
            debug_print("Натиснуто кнопку 'Повернутися'")
            # Очищаємо query params
            st.query_params.clear()
            debug_print("Очищено query параметри")
            st.rerun()
        return
    debug_print(f"Завантажено дані книги: title={book['title']}, author={book['author']}")
    # Показуємо форму редагування
```

```
    with st.form("edit_book_form"):
        title = st.text_input("Назва", value=book['title'])
        author = st.text_input("Автор", value=book['author'])
        category = st.text_input("Категорія", value=book.get('category', ''))
        price = st.number_input("Ціна (грн)", min_value=0.0, step=0.1,
value=float(book['price']))
        stock = st.number_input("Кількість на складі", min_value=0, step=1,
value=int(book['stock']))
        description = st.text_area("Опис", value=book.get('description', ''))
        image_url = st.text_input("URL зображення", value=book.get('image_url', ''))
        col1, col2 = st.columns(2)
        with col1:
            if st.form_submit_button("Зберегти зміни"):
                debug_print(f"Натиснуто кнопку 'Зберегти зміни' для книги ID={book_id}")
                if not title or not author:
                    st.error("Назва та автор обов'язкові")
                    debug_print("Помилка: не заповнені обов'язкові поля")
                else:
                    result, status_code = update_book(book_id, {
                        'title': title,
                        'author': author,
                        'category': category,
                        'price': price,
                        'stock': stock,
                        'description': description,
                        'image_url': image_url
                    })
                    debug_print(f"Результат оновлення книги: status_code={status_code},
result={result}")
                    if status_code == 200:
                        st.success("Книгу успішно оновлено")
                        debug_print("Книгу успішно оновлено, очищення query параметрів")
                        # Очищаємо query params і перезавантажуємо сторінку
                        st.query_params.clear()
                        st.rerun()
                    else:
                        st.error(result.get("message", "Помилка при оновленні книги"))
                        debug_print(f"Помилка при оновленні книги: {result.get('message',
'невідома помилка')}")
    # Кнопка "Назад" поза формою
    if st.button("Повернутися без збереження"):
        debug_print("Натиснуто кнопку 'Повернутися без збереження'")
        st.query_params.clear()  # Очищаємо query params
        debug_print("Очищено query параметри")
        st.rerun()
def show_edit_book():
    st.header("Редагування книги")
    debug_print("Відображення сторінки редагування книги (застаріла функція)")
    if not st.session_state.user or st.session_state.user['role'] != 'manager':
        st.warning("Доступ дозволено тільки менеджерам")
        debug_print("Спроба доступу до редагування книги без прав менеджера")
        if st.button("Перейти до каталогу"):
            debug_print("Натиснуто кнопку 'Перейти до каталогу'")
            set_page("Каталог")
        return
    # Отримуємо ID книги з сесії
    book_id = st.session_state.book_id
    debug_print(f"Редагування книги з ID={book_id}")
    if not book_id:
        st.error("Книгу не знайдено")
        debug_print("Book_id не встановлено")
        if st.button("Повернутися"):
            debug_print("Натиснуто кнопку 'Повернутися'")
            set_page("Управління товарами")
        return
```

```
    # Отримуємо дані книги
    book = get_book(book_id)
    if not book:
        st.error("Книгу не знайдено")
        debug_print(f"Книгу з ID={book_id} не знайдено в базі даних")
        if st.button("Повернутися"):
            debug_print("Натиснуто кнопку 'Повернутися'")
            set_page("Управління товарами")
        return
    debug_print(f"Завантажено дані книги: title={book['title']}, author={book['author']}")
    # Показуємо форму редагування
    with st.form("edit_book_form"):
        title = st.text_input("Назва", value=book['title'])
        author = st.text_input("Автор", value=book['author'])
        category = st.text_input("Категорія", value=book.get('category', ''))
        price = st.number_input("Ціна (грн)", min_value=0.0, step=0.1,
value=float(book['price']))
        stock = st.number_input("Кількість на складі", min_value=0, step=1,
value=int(book['stock']))
        description = st.text_area("Опис", value=book.get('description', ''))
        image_url = st.text_input("URL зображення", value=book.get('image_url', ''))
        submit_button = st.form_submit_button("Зберегти зміни")
    # Обробка відправки форми поза формою, щоб уникнути проблем з st.form
    if submit_button:
        debug_print(f"Натиснуто кнопку 'Зберегти зміни' для книги ID={book_id}")
        if not title or not author:
            st.error("Назва та автор обов'язкові")
            debug_print("Помилка: не заповнені обов'язкові поля")
        else:
            result, status_code = update_book(book_id, {
                'title': title,
                'author': author,
                'category': category,
                'price': price,
                'stock': stock,
                'description': description,
                'image_url': image_url
            })
            debug_print(f"Результат оновлення книги: status_code={status_code},
result={result}")
            if status_code == 200:
                st.success("Книгу успішно оновлено")
                debug_print("Книгу успішно оновлено, перехід до управління товарами")
                # Затримка, щоб користувач побачив повідомлення про успіх
                import time
                time.sleep(1)
                set_page("Управління товарами")
            else:
                st.error(result.get("message", "Помилка при оновленні книги"))
                debug_print(f"Помилка при оновленні книги: {result.get('message', 'невідома
помилка')}")
    # Кнопка "Назад" поза формою
    if st.button("Повернутися без збереження"):
        debug_print("Натиснуто кнопку 'Повернутися без збереження'")
        set_page("Управління товарами")
def show_manage_orders():
    st.header("Управління замовленнями")
    debug_print("Відображення сторінки управління замовленнями")
    if not st.session_state.user or st.session_state.user['role'] != 'manager':
        st.warning("Доступ дозволено тільки менеджерам")
        debug_print("Спроба доступу до управління замовленнями без прав менеджера")
        if st.button("Перейти до каталогу"):
            debug_print("Натиснуто кнопку 'Перейти до каталогу'")
            set_page("Каталог")
        return
```

```
    orders = get_orders()
    if not orders:
        st.info("Замовлення не знайдено")
        debug_print("Замовлення не знайдено в базі даних")
        return
    # Фільтр за статусом
    status_options = ["Всі", "нове", "оброблене", "відправлене", "доставлене", "скасоване"]
    status_filter = st.selectbox("Фільтр за статусом", status_options)
    debug_print(f"Вибрано фільтр за статусом: {status_filter}")
    filtered_orders = orders
    if status_filter != "Всі":
        filtered_orders = [order for order in orders if order['status'] == status_filter]
        debug_print(f"Застосовано фільтр, залишилося {len(filtered_orders)} замовлень")
    if not filtered_orders:
        st.info(f"Замовлення зі статусом '{status_filter}' не знайдено")
        debug_print(f"Не знайдено замовлень зі статусом '{status_filter}'")
        return
    # Перевіряємо, чи є повідомлення про оновлення статусу
    if 'status_updated' in st.session_state and st.session_state.status_updated:
        st.success("Статус замовлення успішно оновлено")
        debug_print("Відображено повідомлення про успішне оновлення статусу")
        # Видаляємо прапор, щоб повідомлення не відображалося знову
        del st.session_state.status_updated
    # Перевіряємо, чи є повідомлення про помилку оновлення статусу
    if 'status_update_error' in st.session_state and st.session_state.status_update_error:
        st.error(st.session_state.status_update_error)
        debug_print(f"Відображено повідомлення про помилку:
{st.session_state.status_update_error}")
        # Видаляємо прапор, щоб повідомлення не відображалося знову
        del st.session_state.status_update_error
    for order in filtered_orders:
        status_color = {
            'нове': '#FFD700',  # Золотистий
            'оброблене': '#1E90FF',  # Синій
            'відправлене': '#32CD32',  # Зелений
            'доставлене': '#006400',  # Темно-зелений
            'скасоване': '#FF0000'  # Червоний
        }.get(order['status'].lower(), secondary_color)
        # Змінюємо структуру сітки, щоб додати колонку для статусу
        col1, col2, col3 = st.columns([3, 2, 1])
        with col1:
            st.markdown(f"""
            <div style="background-color: {light_color}; padding: 15px; border-radius: 5px;
margin-bottom: 20px; color: {dark_text_color};">
                <h3>Замовлення #{order['id']}</h3>
                <p><strong>Користувач:</strong> {order['username']}</p>
                <p><strong>Дата:</strong> {order['order_date']}</p>
                <p><strong>Сума:</strong> <span style="color: {secondary_color}; font-
weight: bold;">{order['total_price']} грн</span></p>
                <p><strong>Статус:</strong> <span style="color: {status_color}; font-weight:
bold;">{order['status']}</span></p>
            </div>
            """, unsafe_allow_html=True)
        with col2:
            # Додаємо випадаючий список статусів прямо на картці замовлення
            new_status = st.selectbox(
                "Змінити статус",
                status_options[1:],  # Виключаємо "Всі" з опцій
                index=status_options[1:].index(order['status']) if order['status'] in
status_options[1:] else 0,
                key=f"status_select_{order['id']}"
            )
            # Додаємо кнопку для підтвердження зміни статусу
            if st.button("Зберегти статус", key=f"save_status_{order['id']}"):
```

```
                debug_print(f"Натиснуто кнопку 'Зберегти статус' для замовлення
ID={order['id']}, новий статус: {new_status}")
                success = update_order_status_direct(order['id'], new_status)
                if success:
                    st.session_state.status_updated = True
                    debug_print(f"Статус замовлення ID={order['id']} успішно оновлено до
'{new_status}'")
                    # Перезавантажуємо сторінку, щоб відобразити оновлення
                    st.rerun()
                else:
                    debug_print(f"Помилка при оновленні статусу замовлення
ID={order['id']}")
        with col3:
            if st.button("Деталі", key=f"view_manage_order_{order['id']}"):
                debug_print(f"Натиснуто кнопку 'Деталі' для замовлення ID={order['id']}")
                set_page("Деталі замовлення", order_id=order['id'])
# Додамо панель відладки для перегляду логів
def show_debug_panel():
    st.sidebar.markdown("---")
    with st.sidebar.expander("Відладочна інформація", expanded=False):
        st.write("Останні відладочні повідомлення:")
        for msg in st.session_state.debug[-30:]:  # Показуємо останні 30 повідомлень
            st.write(msg)
        if st.button("Очистити лог"):
            st.session_state.debug = []
            st.rerun()
# Головна функція додатку
def main():
    # Застосування CSS-стилів
    st.markdown(
        f"""
        <style>
        .main .block-container {{
            padding-top: 2rem;
        }}
        .stButton>button {{
            background-color: {primary_color};
            color: {text_color};
        }}
        .stButton>button:hover {{
            background-color: {secondary_color};
        }}
        .sidebar .sidebar-content {{
            background-color: {light_color};
        }}
        h1, h2, h3 {{
            color: {primary_color};
        }}
        .stTextInput>div>div>input {{
            border-color: {accent_color};
        }}
        .stExpander {{
            border-color: {accent_color};
        }}
        </style>
        """,
        unsafe_allow_html=True,
    )
    # Відображення бічної панелі
    st.sidebar.title("Меню")
    # Опції меню відповідно до ролі користувача
    menu_options = ["Каталог", "Корзина"]
    if st.session_state.user:
        menu_options.extend(["Мій профіль", "Мої замовлення"])
        if st.session_state.user["role"] == "manager":
```

```
            menu_options.extend(["Управління товарами", "Управління замовленнями"])
        menu_options.append("Вихід")
    else:
        menu_options.extend(["Вхід", "Реєстрація"])
    # Вибрана сторінка в меню
    selected_page = st.sidebar.radio("Навігація", menu_options,
                                     index=menu_options.index(st.session_state.page) if
st.session_state.page in menu_options else 0)
    # Обробка зміни сторінки через меню
    if selected_page != st.session_state.page and not st.session_state.redirect:
        debug_print(f"Зміна сторінки через меню: з '{st.session_state.page}' на
'{selected_page}'")
        set_page(selected_page)
        return
    # Скидання прапора перенаправлення
    st.session_state.redirect = False
    # Відображення інформації про користувача
    if st.session_state.user:
        st.sidebar.markdown(f"""
        <div style="background-color: {primary_color}; padding: 10px; border-radius: 5px;
color: {text_color}; margin-top: 20px;">
            <p><strong>Користувач:</strong> {st.session_state.user['username']}</p>
            <p><strong>Роль:</strong> {"Менеджер" if st.session_state.user['role'] ==
"manager" else "Клієнт"}</p>
        </div>
        """, unsafe_allow_html=True)
    # Відображення корзини в бічній панелі
    if st.session_state.user and st.session_state.cart:
        with st.sidebar.expander(f"Кошик ({len(st.session_state.cart)} товарів)"):
            for item in st.session_state.cart:
                st.markdown(f"""
                <div style="background-color: {light_color}; padding: 5px; border-radius:
5px; margin-bottom: 5px; color: {dark_text_color};">
                    <p><strong>{item['title']}</strong> x {item['quantity']}</p>
                    <p>{item['price'] * item['quantity']} грн</p>
                </div>
                """, unsafe_allow_html=True)
            st.markdown(f"<p><strong>Всього:</strong> {calculate_cart_total()} грн</p>",
unsafe_allow_html=True)
            if st.button("Перейти до кошика"):
                debug_print("Натиснуто кнопку 'Перейти до кошика' в бічній панелі")
                set_page("Корзина")
    # Відображення панелі відладки
    show_debug_panel()
    # Відображення відповідної сторінки на основі st.session_state.page
    debug_print(f"Відображення поточної сторінки: {st.session_state.page}")
    if st.session_state.page == "Каталог":
        show_catalog()
    elif st.session_state.page == "Деталі книги":
        show_book_details()
    elif st.session_state.page == "Корзина":
        show_cart()
    elif st.session_state.page == "Вхід":
        show_login_form()
    elif st.session_state.page == "Реєстрація":
        show_register_form()
    elif st.session_state.page == "Мій профіль":
        show_profile()
    elif st.session_state.page == "Мої замовлення":
        show_orders()
    elif  st.session_state.page == "Деталі замовлення":
            if st.session_state.order_id:
                show_order_details()
            else:
                st.error("Замовлення не знайдено")
```

```
                debug_print("Order_id не встановлено при спробі відкрити деталі замовлення")
                st.session_state.page = "Мої замовлення"
                st.rerun()
    elif st.session_state.page == "Управління товарами":
        show_manage_books()
    # Видалено: elif st.session_state.page == "Редагувати книгу":
    # Видалено:     show_edit_book()
    elif st.session_state.page == "Управління замовленнями":
        show_manage_orders()
    elif st.session_state.page == "Вихід":
        debug_print("Виконується вихід користувача")
        logout_user()
        set_page("Каталог")
if __name__ == "__main__":
    main()
```

server.py

```
from flask import Flask, request, jsonify
from flask_cors import CORS
import sqlite3
import hashlib
import os
import jwt
from datetime import datetime, timedelta
from functools import wraps
app = Flask(__name__)
CORS(app, supports_credentials=True)
# Секретний ключ для JWT
SECRET_KEY = os.urandom(24)
# Термін дії токена (24 години)
TOKEN_EXPIRATION = timedelta(hours=24)
# Ініціалізація бази даних
def init_db():
    conn = sqlite3.connect('bookstore.db')
    c = conn.cursor()
   # Створення таблиці користувачів
    c.execute('''
    CREATE TABLE IF NOT EXISTS users (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        username TEXT UNIQUE NOT NULL,
        password TEXT NOT NULL,
        email TEXT UNIQUE NOT NULL,
        role TEXT NOT NULL,
        full_name TEXT,
        address TEXT,
        phone TEXT
    )
    ''')
    # Створення таблиці книг
    c.execute('''
    CREATE TABLE IF NOT EXISTS books (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        title TEXT NOT NULL,
        author TEXT NOT NULL,
        description TEXT,
        price REAL NOT NULL,
        stock INTEGER NOT NULL,
        category TEXT,
        image_url TEXT
    )
    ''')
    # Створення таблиці замовлень
```

```
    c.execute('''
    CREATE TABLE IF NOT EXISTS orders (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        user_id INTEGER NOT NULL,
        order_date TEXT NOT NULL,
        total_price REAL NOT NULL,
        status TEXT NOT NULL,
        FOREIGN KEY (user_id) REFERENCES users (id)
    )
    ''')
    # Створення таблиці деталей замовлення
    c.execute('''
    CREATE TABLE IF NOT EXISTS order_items (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        order_id INTEGER NOT NULL,
        book_id INTEGER NOT NULL,
        quantity INTEGER NOT NULL,
        price REAL NOT NULL,
        FOREIGN KEY (order_id) REFERENCES orders (id),
        FOREIGN KEY (book_id) REFERENCES books (id)
    )
    ''')
    conn.commit()
    conn.close()
# Ініціалізація бази даних при запуску
init_db()
# Допоміжні функції
def hash_password(password):
    return hashlib.sha256(password.encode()).hexdigest()
def generate_token(user_id):
    payload = {
        'user_id': user_id,
        'exp': datetime.utcnow() + TOKEN_EXPIRATION
    }
    return jwt.encode(payload, SECRET_KEY, algorithm='HS256')
def verify_token(token):
    try:
        payload = jwt.decode(token, SECRET_KEY, algorithms=['HS256'])
        return payload['user_id']
    except (jwt.ExpiredSignatureError, jwt.InvalidTokenError):
        return None
# Декоратор для перевірки автентифікації через токен
def token_required(f):
    @wraps(f)
    def decorated_function(*args, **kwargs):
        token = None
        auth_header = request.headers.get('Authorization')
        if auth_header and auth_header.startswith('Bearer '):
            token = auth_header.split(' ')[1]
        if not token:
            return jsonify({"success": False, "message": "Необхідна автентифікація"}), 401
        user_id = verify_token(token)
        if not user_id:
            return jsonify({"success": False, "message": "Недійсний або застарілий токен"}),
401
        return f(user_id, *args, **kwargs)
    return decorated_function
# Декоратор для перевірки ролі менеджера
def manager_required(f):
    @wraps(f)
    def decorated_function(user_id, *args, **kwargs):
        conn = sqlite3.connect('bookstore.db')
        c = conn.cursor()
        c.execute("SELECT role FROM users WHERE id = ?", (user_id,))
        user = c.fetchone()
```

```
        conn.close()
        if not user or user[0] != 'manager':
            return jsonify({"success": False, "message": "Необхідні права менеджера"}), 403
        return f(user_id, *args, **kwargs)
    return decorated_function
# Маршрути API
@app.route('/api/register', methods=['POST'])
def register():
    data = request.get_json()
    if not all(key in data for key in ['username', 'password', 'email', 'role']):
        return jsonify({"success": False, "message": "Відсутні обов'язкові поля"}), 400
    if data['role'] not in ['client', 'manager']:
        return jsonify({"success": False, "message": "Невірна роль користувача"}), 400
    conn = sqlite3.connect('bookstore.db')
    c = conn.cursor()
    try:
        c.execute("INSERT INTO users (username, password, email, role, full_name, address,
phone) VALUES (?, ?, ?, ?, ?, ?, ?)",
                  (data['username'], hash_password(data['password']), data['email'],
data['role'],
                   data.get('full_name', ''), data.get('address', ''), data.get('phone',
''')))
        conn.commit()
        # Отримання ID нового користувача
        c.execute("SELECT id FROM users WHERE username = ?", (data['username'],))
        user_id = c.fetchone()[0]
        # Генерація токена для нового користувача
        token = generate_token(user_id)
        return jsonify({
            "success": True,
            "message": "Користувача успішно зареєстровано",
            "user_id": user_id,
            "token": token
        }), 201
    except sqlite3.IntegrityError:
        return jsonify({"success": False, "message": "Користувач з таким ім'ям або
електронною адресою вже існує"}), 409

    finally:
        conn.close()
@app.route('/api/login', methods=['POST'])
def login():
    data = request.get_json()
    if not all(key in data for key in ['username', 'password']):
        return jsonify({"success": False, "message": "Відсутні обов'язкові поля"}), 400
    conn = sqlite3.connect('bookstore.db')
    c = conn.cursor()
    c.execute("SELECT id, username, role FROM users WHERE username = ? AND password = ?",
              (data['username'], hash_password(data['password'])))
    user = c.fetchone()
    conn.close()
    if user:
        # Генерація JWT токена
        token = generate_token(user[0])
        return jsonify({
            "success": True,
            "user_id": user[0],
            "username": user[1],
            "role": user[2],
            "token": token
        }), 200
    else:
        return jsonify({"success": False, "message": "Невірне ім'я користувача або
пароль"}), 401
@app.route('/api/logout', methods=['POST'])
```

```
def logout():
    # З JWT немає необхідності в серверній логіці для виходу
    # Клієнт просто видаляє токен у себе
    return jsonify({"success": True, "message": "Вихід виконано успішно"}), 200
@app.route('/api/user/profile', methods=['GET'])
@token_required
def get_profile(user_id):
    conn = sqlite3.connect('bookstore.db')
    c = conn.cursor()
    c.execute("SELECT id, username, email, role, full_name, address, phone FROM users WHERE
id = ?", (user_id,))
    user = c.fetchone()
    conn.close()
    if user:
        return jsonify({
            "id": user[0],
            "username": user[1],
            "email": user[2],
            "role": user[3],
            "full_name": user[4],
            "address": user[5],
            "phone": user[6]
        }), 200
    else:
        return jsonify({"success": False, "message": "Користувача не знайдено"}), 404
@app.route('/api/user/profile', methods=['PUT'])
@token_required
def update_profile(user_id):
    data = request.get_json()
    conn = sqlite3.connect('bookstore.db')
    c = conn.cursor()
    # Оновлюємо лише дозволені поля
    allowed_fields = {
        'email': data.get('email'),
        'full_name': data.get('full_name'),
        'address': data.get('address'),
        'phone': data.get('phone')
    }
    if 'password' in data and data['password']:
        allowed_fields['password'] = hash_password(data['password'])
    # Формування SQL запиту для оновлення
    fields = ", ".join([f"{field} = ?" for field in allowed_fields.keys()])
    values = list(allowed_fields.values())
    values.append(user_id)
    try:
        c.execute(f"UPDATE users SET {fields} WHERE id = ?", values)
        conn.commit()
        return jsonify({"success": True, "message": "Профіль оновлено успішно"}), 200
    except sqlite3.IntegrityError:
        return jsonify({"success": False, "message": "Така електронна адреса вже
використовується"}), 409

    finally:
        conn.close()
# Маршрути для книг
@app.route('/api/books', methods=['GET'])
def get_books():
    conn = sqlite3.connect('bookstore.db')
    c = conn.cursor()
    # Параметри фільтрації
    category = request.args.get('category')
    search = request.args.get('search')
    query = "SELECT id, title, author, description, price, stock, category, image_url FROM
books"
    params = []
```

```
    if category and search:
        query += " WHERE category = ? AND (title LIKE ? OR author LIKE ?)"
        params = [category, f"%{search}%", f"%{search}%"]
    elif category:
        query += " WHERE category = ?"
        params = [category]
    elif search:
        query += " WHERE title LIKE ? OR author LIKE ?"
        params = [f"%{search}%", f"%{search}%"]
    c.execute(query, params)
    books = c.fetchall()
    conn.close()
    result = []
    for book in books:
        result.append({
            "id": book[0],
            "title": book[1],
            "author": book[2],
            "description": book[3],
            "price": book[4],
            "stock": book[5],
            "category": book[6],
            "image_url": book[7]
        })
    return jsonify(result), 200
@app.route('/api/books/<int:book_id>', methods=['GET'])
def get_book(book_id):
    conn = sqlite3.connect('bookstore.db')
    c = conn.cursor()
    c.execute("SELECT id, title, author, description, price, stock, category, image_url FROM
books WHERE id = ?", (book_id,))
    book = c.fetchone()
    conn.close()
    if book:
        return jsonify({
            "id": book[0],
            "title": book[1],
            "author": book[2],
            "description": book[3],
            "price": book[4],
            "stock": book[5],
            "category": book[6],
            "image_url": book[7]
        }), 200
    else:
        return jsonify({"success": False, "message": "Книгу не знайдено"}), 404
@app.route('/api/books', methods=['POST'])
@token_required
@manager_required
def add_book(user_id):
    data = request.get_json()
    if not all(key in data for key in ['title', 'author', 'price', 'stock']):
        return jsonify({"success": False, "message": "Відсутні обов'язкові поля"}), 400
    conn = sqlite3.connect('bookstore.db')
    c = conn.cursor()
    c.execute("INSERT INTO books (title, author, description, price, stock, category,
image_url) VALUES (?, ?, ?, ?, ?, ?, ?)",
              (data['title'], data['author'], data.get('description', ''), data['price'],
data['stock'],
               data.get('category', ''), data.get('image_url', '')))
    book_id = c.lastrowid
    conn.commit()
    conn.close()
    return jsonify({"success": True, "message": "Книгу додано успішно", "book_id":
book_id}), 201
```

```
@app.route('/api/books/<int:book_id>', methods=['PUT'])
@token_required
@manager_required
def update_book(user_id, book_id):
    data = request.get_json()
    conn = sqlite3.connect('bookstore.db')
    c = conn.cursor()
    # Перевірка, чи існує книга
    c.execute("SELECT id FROM books WHERE id = ?", (book_id,))
    if not c.fetchone():
        conn.close()
        return jsonify({"success": False, "message": "Книгу не знайдено"}), 404
    # Оновлення книги
    c.execute("UPDATE books SET title = ?, author = ?, description = ?, price = ?, stock =
?, category = ?, image_url = ? WHERE id = ?",
              (data.get('title'), data.get('author'), data.get('description', ''),
data.get('price'),
               data.get('stock'), data.get('category', ''), data.get('image_url', ''),
book_id))
    conn.commit()
    conn.close()
    return jsonify({"success": True, "message": "Книгу оновлено успішно"}), 200
@app.route('/api/books/<int:book_id>', methods=['DELETE'])
@token_required
@manager_required
def delete_book(user_id, book_id):
    conn = sqlite3.connect('bookstore.db')
    c = conn.cursor()
    # Перевірка, чи існує книга
    c.execute("SELECT id FROM books WHERE id = ?", (book_id,))
    if not c.fetchone():
        conn.close()
        return jsonify({"success": False, "message": "Книгу не знайдено"}), 404
    # Видалення книги
    c.execute("DELETE FROM books WHERE id = ?", (book_id,))
    conn.commit()
    conn.close()
    return jsonify({"success": True, "message": "Книгу видалено успішно"}), 200
# Маршрути для корзини та замовлень
@app.route('/api/orders', methods=['POST'])
@token_required
def create_order(user_id):
    data = request.get_json()
    if 'items' not in data or not data['items']:
        return jsonify({"success": False, "message": "Корзина порожня"}), 400
    conn = sqlite3.connect('bookstore.db')
    c = conn.cursor()
    try:
        # Перевірка наявності товарів на складі
        for item in data['items']:
            c.execute("SELECT stock FROM books WHERE id = ?", (item['book_id'],))
            stock = c.fetchone()
            if not stock:
                return jsonify({"success": False, "message": f"Книга з ID {item['book_id']}
не знайдена"}), 404
            if stock[0] < item['quantity']:
                return jsonify({"success": False, "message": f"Недостатня кількість книги з
ID {item['book_id']} на складі"}), 400
        # Розрахунок суми замовлення
        total_price = 0
        for item in data['items']:
            c.execute("SELECT price FROM books WHERE id = ?", (item['book_id'],))
            price = c.fetchone()[0]
            total_price += price * item['quantity']
        # Створення замовлення
```

```
        current_date = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        c.execute("INSERT INTO orders (user_id, order_date, total_price, status) VALUES (?,
?, ?, ?)",
                  (user_id, current_date, total_price, 'нове'))
        order_id = c.lastrowid
        # Додавання товарів до замовлення
        for item in data['items']:
            c.execute("SELECT price FROM books WHERE id = ?", (item['book_id'],))
            price = c.fetchone()[0]
            c.execute("INSERT INTO order_items (order_id, book_id, quantity, price) VALUES
(?, ?, ?, ?)",
                      (order_id, item['book_id'], item['quantity'], price))
            # Оновлення кількості на складі
            c.execute("UPDATE books SET stock = stock - ? WHERE id = ?", (item['quantity'],
item['book_id']))
        conn.commit()
        return jsonify({"success": True, "message": "Замовлення створено успішно",
"order_id": order_id}), 201
    except Exception as e:
        conn.rollback()
        return jsonify({"success": False, "message": f"Помилка при створенні замовлення:
{str(e)}"}), 500
    finally:
        conn.close()
@app.route('/api/orders', methods=['GET'])
@token_required
def get_orders(user_id):
    conn = sqlite3.connect('bookstore.db')
    c = conn.cursor()
    # Перевірка ролі
    c.execute("SELECT role FROM users WHERE id = ?", (user_id,))
    role = c.fetchone()[0]
    if role == 'manager':
        # Менеджер бачить всі замовлення
        c.execute("""
            SELECT o.id, o.user_id, u.username, o.order_date, o.total_price, o.status
            FROM orders o
            JOIN users u ON o.user_id = u.id
            ORDER BY o.order_date DESC
        """)
    else:
        # Клієнт бачить лише свої замовлення
        c.execute("""
            SELECT o.id, o.user_id, u.username, o.order_date, o.total_price, o.status
            FROM orders o
            JOIN users u ON o.user_id = u.id
            WHERE o.user_id = ?
            ORDER BY o.order_date DESC
        """, (user_id,))
    orders = c.fetchall()
    result = []
    for order in orders:
        # Отримання деталей замовлення
        c.execute("""
            SELECT oi.book_id, b.title, oi.quantity, oi.price
            FROM order_items oi
            JOIN books b ON oi.book_id = b.id
            WHERE oi.order_id = ?
        """, (order[0],))
        items = c.fetchall()
        order_items = []
        for item in items:
            order_items.append({
                "book_id": item[0],
                "title": item[1],
```

```
                "quantity": item[2],
                "price": item[3]
            })
        result.append({
            "id": order[0],
            "user_id": order[1],
            "username": order[2],
            "order_date": order[3],
            "total_price": order[4],
            "status": order[5],
            "items": order_items
        })
    conn.close()
    return jsonify(result), 200
@app.route('/api/orders/<int:order_id>', methods=['GET'])
@token_required
def get_order(user_id, order_id):
    conn = sqlite3.connect('bookstore.db')
    c = conn.cursor()
    # Перевірка ролі
    c.execute("SELECT role FROM users WHERE id = ?", (user_id,))
    role = c.fetchone()[0]
    if role == 'client':
        # Перевірка, що замовлення належить користувачу
        c.execute("SELECT user_id FROM orders WHERE id = ?", (order_id,))
        order_user = c.fetchone()
        if not order_user or order_user[0] != user_id:
            conn.close()
            return jsonify({"success": False, "message": "Доступ заборонено"}), 403
    # Отримання інформації про замовлення
    c.execute("""
        SELECT o.id, o.user_id, u.username, o.order_date, o.total_price, o.status
        FROM orders o
        JOIN users u ON o.user_id = u.id
        WHERE o.id = ?
    """, (order_id,))
    order = c.fetchone()
    if not order:
        conn.close()
        return jsonify({"success": False, "message": "Замовлення не знайдено"}), 404
    # Отримання деталей замовлення
    c.execute("""
        SELECT oi.book_id, b.title, oi.quantity, oi.price
        FROM order_items oi
        JOIN books b ON oi.book_id = b.id
        WHERE oi.order_id = ?
    """, (order_id,))
    items = c.fetchall()
    order_items = []
    for item in items:
        order_items.append({
            "book_id": item[0],
            "title": item[1],
            "quantity": item[2],
            "price": item[3]
        })
    result = {
        "id": order[0],
        "user_id": order[1],
        "username": order[2],
        "order_date": order[3],
        "total_price": order[4],
        "status": order[5],
        "items": order_items
    }
```

```
    conn.close()
    return jsonify(result), 200
@app.route('/api/orders/<int:order_id>/status', methods=['PUT'])
@token_required
@manager_required
def update_order_status(user_id, order_id):
    data = request.get_json()
    if 'status' not in data:
        return jsonify({"success": False, "message": "Статус не вказано"}), 400
    conn = sqlite3.connect('bookstore.db')
    c = conn.cursor()
    # Перевірка, чи існує замовлення
    c.execute("SELECT id FROM orders WHERE id = ?", (order_id,))
    if not c.fetchone():
        conn.close()
        return jsonify({"success": False, "message": "Замовлення не знайдено"}), 404
    # Оновлення статусу
    c.execute("UPDATE orders SET status = ? WHERE id = ?", (data['status'], order_id))
    conn.commit()
    conn.close()
    return jsonify({"success": True, "message": "Статус замовлення оновлено успішно"}), 200
@app.route('/api/categories', methods=['GET'])
def get_categories():
    conn = sqlite3.connect('bookstore.db')
    c = conn.cursor()
    c.execute("SELECT DISTINCT category FROM books WHERE category IS NOT NULL AND category
!= ''")
    categories = [row[0] for row in c.fetchall()]
    conn.close()
    return jsonify(categories), 200
if __name__ == '__main__':
    app.run(debug=True, port=5000)
```