Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка рекомендаційної системи для інтернет-магазину книг

Виконав: студент IV курсу, групи СН-41
спеціальності 122 Комп’ютерні науки
(шифр і назва спеціальності)

_____ Багрій М. Т.
(підпис) (прізвище та ініціали)

Керівник _____ Марценко С. В.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Липак Г. І.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Боднарчук І.О.
(підпис) (прізвище та ініціали)

Рецензент _____ Загородна Н. В.
(підпис) (прізвище та ініціали)

Тернопіль
2025

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
«____» червня 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)

за спеціальністю 122 Комп'ютерні науки
(шифр і назва спеціальності)

Студенту Багрій Максим Тарасович
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка рекомендаційної системи для інтернет-магазину книг

Керівник роботи Марценко Сергій Володимирович, кандидат технічних наук, доцент кафедри КН
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «*07*» *травня* 2025 року № *4/7-444*

2. Термін подання студентом завершеної роботи 23 червня 2025 р.

3. Вихідні дані до роботи Google Gemini API, бібліотека Streamlit для клієнтської частини, фреймворк Flask та база даних SQLite для серверної частини, а також дані про книги (назва, автор, категорія, ціна, опис) та профілі користувачів (логін, пароль, роль, історія замовлень).

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. 1. Аналіз предметної області та постановка завдання. 2. Проектна розробка Рекомендаційної системи. 3. Практична реалізація та аналіз результатів 4. Безпека життєдіяльності, основи охорони праці. Висновки. Перелік джерел. Додаток А. Коди програми

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
Повний перелік слайдів у презентації, включаючи перший та останній слайди (з номерами, через крапку). 1. Титулка; 2. Зміст; 3. Актуальність теми; 4. Мета і задачі; 5. Розділ 1. Аналіз предметної області та постановка завдання; 6. Розділ 2. Проектна розробка рекомендаційної системи; 7. Розділ 3. Практична реалізація та аналіз результатів; 8. Розділ 4. Безпека життєдіяльності, основи охорони праці; 9. Висновки; 10. Кінець;

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	Сенчишин В. С. кандидат технічних наук, доцент кафедри МТ	02.06.2025	23.06.2025

7. Дата видачі завдання 27 січня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	27.01.2025	*Виконано*
2.	Підбір та опрацювання літературних джерел по темі кваліфікаційної роботи	08.05.2025-15.05.2025	*Виконано*
3.	Виконання дослідження щодо типів рекомендаційних	16.05.2025-18.05.2025	*Виконано*
	систем та особливостей книжкових інтернет-магазинів		
	Розроблення структури і функціоналу	19.05.2025-22.05.2025	*Виконано*
	інтернет-магазину книг		
4.	Оформлення розділу «Аналіз предметної області та	23.05.2025-01.06.2025	*Виконано*
	постановка завдання»		
5.	Оформлення розділу «Проектна розробка	23.05.2025-01.06.2025	*Виконано*
	рекомендацій системи»		
6.	Оформлення розділу «Практична реалізація та аналіз	23.05.2025-01.06.2025	*Виконано*
	результатів»		
7.	Виконання завдання до підрозділу «Безпека життєдіяльності»	02.06.2025-05.06.2025	*Виконано*
8.	Виконання завдання до підрозділу «Основи охорони	02.06.2025-05.06.2025	*Виконано*
	праці»		
9.	Оформлення кваліфікаційної роботи	06.06.2025-10.06.2025	*Виконано*
10.	Нормоконтроль	11.06.2025-13.06.2025	*Виконано*
11.	Перевірка на плагіат	16.06.2025	*Виконано*
12.	Попередній захист кваліфікаційної роботи	18.06.2025	*Виконано*
13.	Захист кваліфікаційної роботи	24.06.2025	

Студент _____ Багрій М. Т.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Марценко С. В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Тема: Розробка рекомендаційної системи для інтернет-магазину книг // Кваліфікаційна робота освітнього рівня «Бакалавр» // Багрій Максим Тарасович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних наук, група СН-41 // Тернопіль, 2025 // С. 64, рис. – 25, табл. – 2, кресл. – 0, додат. – 1, бібліогр. – 36.

Ключові слова: рекомендаційна система, інтернет-магазин, книги, штучний інтелект, Google Gemini API, клієнт-серверна архітектура, персоналізація, електронна комерція.

Кваліфікаційна робота присвячена розробці рекомендаційної системи для інтернет-магазину книг, що забезпечує персоналізований підбір літератури на основі текстових запитів користувачів. У першому розділі проаналізовано сучасний стан рекомендаційних систем, їх типи та застосування в електронній комерції. Висвітлено особливості книжкового ринку та потреби користувачів. Розглянуто технологічні аспекти реалізації системи.

У другому розділі описано архітектуру системи на базі Streamlit, Flask та SQLite з інтеграцією Google Gemini API. Досліджено методи рекомендацій, зокрема аналіз природної мови та фільтрацію метаданих. Подано принципи проектування інтуїтивного інтерфейсу.

У третьому розділі представлено практичну реалізацію та тестування системи. Проаналізовано результати роботи, підтверджено її функціональність і зручність. Проведено оцінку ефективності рекомендацій.

Об'єкт дослідження: рекомендаційні системи в електронній комерції.

Предмет дослідження: розробка та впровадження рекомендаційної системи для книжкового інтернет-магазину.

ANNOTATION

Topic: Development of a Recommendation System for an Online Bookstore // Bachelor's Qualification Work // Bahrii Maksym Tarasovych // Ternopil Ivan Puluj National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Computer Science, Group SN-41 // Ternopil, 2025 // P. 64, fig. – 25, tabl. – 2, draw. – 0, append. – 1, bibliogr. – 36.

Keywords: recommendation system, online bookstore, books, artificial intelligence, Google Gemini API, client-server architecture, personalization, e-commerce.

The qualification work is devoted to the development of a recommendation system for an online bookstore, which provides personalized book selection based on user text queries. The first section analyzes the current state of recommendation systems, their types, and applications in e-commerce. It highlights the specifics of the book market and user needs. The technological aspects of system implementation are considered.

The second section describes the system architecture based on Streamlit, Flask, and SQLite, integrated with the Google Gemini API. It examines recommendation methods, including natural language processing and metadata filtering. The principles of designing an intuitive interface are presented.

The third section presents the practical implementation and testing of the system. The results are analyzed, confirming its functionality and usability. The effectiveness of the recommendations is evaluated.

Object of research: recommendation systems in e-commerce.

Subject of research: development and implementation of a recommendation system for an online bookstore.

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ШІ – Штучний інтелект

БД – База даних

API – Інтерфейс програмування додатків

JWT – JSON Web Token

СУБД – Система управління базами даних

UX – Користувацький досвід

CSS – Каскадні таблиці стилів

HTTP – Протокол передачі гіпертексту

REST – Представницький стан передачі

SQL – Мова структурованих запитів

GDPR – Загальний регламент захисту даних

ЗМІСТ

ВСТУП 4
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ 6
1.1 Огляд рекомендаційних систем 6
1.2 Аналіз предметної області 9
1.3 Постановка завдання 12
1.4 Висновок до першого розділу 14
РОЗДІЛ 2. ПРОЕКТНА РОЗРОБКА РЕКОМЕНДАЦІЙНОЇ СИСТЕМИ 16
2.1 Вимоги до системи 16
2.2 Архітектура системи 17
2.3 Методи рекомендацій 23
2.4 Проектування інтерфейсу 26
2.5 Висновок до другого розділу 34
РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ 35
3.1 Реалізація рекомендаційної системи 35
3.2 Тестування системи 38
3.3 Аналіз результатів 47
3.4 Висновок до третього розділу 51
РОЗДІЛ 4. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ 53
4.1 Менеджемнт безпеки ..... **Помилка! Закладку не визначено.**
4.2 Естетичне оформлення та ергономічне дослідження робочого місця оператора. ..... **Помилка! Закладку не визначено.**
4.3 Висновок до четвертого розділу 58
ВИСНОВКИ 59
ПЕРЕЛІК ДЖЕРЕЛ 61
ДОДАТОК А 66

ВСТУП

Актуальність теми. У сучасному світі, з поширенням глобалізації та швидким розвитком цифрових технологій, електронна комерція стала невід'ємною частиною повсякденного життя. Інтернет-магазини, зокрема книжкові, стикаються з викликом забезпечення персоналізованого підходу до клієнтів, що є ключовим фактором підвищення їхньої лояльності та конкурентоспроможності. Рекомендаційні системи, які використовують сучасні методи аналізу даних та штучного інтелекту, дозволяють оптимізувати взаємодію з користувачами, пропонуючи товари, що відповідають їхнім інтересам. Таким чином, розробка рекомендаційної системи для інтернет-магазину книг є актуальним напрямком сучасних досліджень у галузі інформаційних технологій та електронної комерції, оскільки сприяє підвищенню якості обслуговування та ефективності бізнес-процесів.

Мета і задачі дослідження. Метою даної кваліфікаційної роботи освітнього рівня «Бакалавр» є розробка рекомендаційної системи для інтернет-магазину книг, що підвищує якість надання персоналізованих пропозицій та сприяє зростанню продажів. Для досягнення поставленої мети необхідно виконати такі завдання:

– проаналізувати сучасний стан досліджень у галузі рекомендаційних систем та їх застосування в електронній комерції;

– вивчити особливості використання Google Gemini API для аналізу запитів користувачів та формування рекомендацій;

– розробити клієнт-серверну архітектуру інтернет-магазину книг із функцією рекомендаційної системи;

– реалізувати та протестувати рекомендаційну систему на основі спроєктованих кодів;

– оцінити ефективність розробленої системи з точки зору точності рекомендацій та зручності для користувачів.

Практичне значення одержаних результатів. Розроблена рекомендаційна система забезпечує автоматизацію процесу підбору книг, що відповідають індивідуальним уподобанням користувачів, що сприяє підвищенню їхньої задоволеності та лояльності. Впровадження системи дозволяє інтернет-магазину оптимізувати асортиментну політику, підвищити обсяги продажів та зменшити час, витрачений користувачами на пошук потрібних товарів. Крім того, отримані результати можуть бути використані для вдосконалення інших систем електронної комерції, де персоналізація відіграє ключову роль.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 Огляд рекомендаційних систем

Рекомендаційні системи є одним із ключових інструментів сучасного інформаційного суспільства, які застосовуються для персоналізації користувацького досвіду в різних сферах, зокрема в електронній комерції, медіа, освіті та соціальних мережах. Їхня основна мета полягає в наданні користувачам пропозицій щодо продуктів, послуг або контенту, які відповідають їхнім інтересам, потребам або попереднім діям. У контексті інтернет-магазину книг, як це представлено у коді проєкту, рекомендаційні системи відіграють важливу роль у підвищенні рівня залученості клієнтів, збільшенні продажів та покращенні користувацького досвіду. У цьому розділі розглядаються основні аспекти рекомендаційних систем, їхні типи, методи функціонування, а також сучасні тенденції їх розвитку.

Рекомендаційні системи виникли як відповідь на проблему інформаційного перевантаження, коли обсяг доступних даних перевищує можливості людини для їх аналізу та вибору. У книжкових інтернет-магазинах, де асортимент може включати тисячі або навіть мільйони найменувань, користувачам складно самостійно знайти книги, які відповідають їхнім уподобанням. Рекомендаційні системи вирішують цю проблему, пропонуючи релевантні пропозиції на основі аналізу даних про користувача, його поведінку, вподобання або схожість із іншими користувачами [1]. Наприклад, у коді клієнтської частини програми (client.py) реалізована функція get_gemini_recommendation, яка використовує Google Gemini API для аналізу запитів користувача та пошуку відповідних книг у каталозі, демонструючи практичне застосування таких систем.

Основними типами рекомендаційних систем є контентно-орієнтовані, колаборативні та гібридні системи. Контентно-орієнтовані системи аналізують характеристики об’єктів (у даному випадку книг, наприклад, жанр, автор, опис)

і зіставляють їх із профілем користувача, який формується на основі його попередніх виборів або явно вказаних уподобань. Такі системи ефективні, коли є детальна інформація про продукти, але вони можуть бути обмеженими у врахуванні нових інтересів користувача [2]. У коді проєкту контентно-орієнтований підхід частково реалізовано через аналіз текстового запиту користувача та зіставлення його з описами книг у каталозі.

Колаборативні системи, натомість, базуються на схожості між користувачами або між об’єктами. Вони використовують дані про поведінку користувачів, наприклад, оцінки, покупки чи перегляди, щоб знайти схожих користувачів або пропонувати продукти, які сподобалися іншим із подібними вподобаннями. Такі системи ефективні для виявлення нових інтересів, але стикаються з проблемою «холодного старту», коли недостатньо даних про нового користувача чи продукт [3]. У контексті книжкового магазину колаборативний підхід міг би аналізувати історію покупок або переглядів інших користувачів, хоча в коді проєкту цей метод не реалізований.

Гібридні системи поєднують переваги контентно-орієнтованих і колаборативних підходів, компенсуючи їхні недоліки. Вони можуть використовувати як характеристики продуктів, так і дані про поведінку користувачів, що забезпечує більш точні рекомендації. Наприклад, гібридна система може спочатку запропонувати книги на основі жанру, який цікавить користувача, а потім уточнити рекомендації, враховуючи популярність книг серед схожих користувачів [4]. У розробленій програмі гібридний підхід частково простежується у функції рекомендацій, де враховується запит користувача та каталог книг, хоча повноцінна інтеграція колаборативних методів відсутня.

Важливим аспектом функціонування рекомендаційних систем є методи збору та обробки даних. Системи можуть використовувати явні дані, такі як оцінки чи відгуки, або неявні, наприклад, історію переглядів чи покупок. У коді проєкту використовується явний підхід, коли користувач формулює текстовий запит із описом своїх побажань, що обробляється ШІ-асистентом. Такий метод

дозволяє враховувати індивідуальні потреби, але вимагає від користувача активної взаємодії [5]. Крім того, системи можуть застосовувати різні алгоритми, включаючи машинне навчання, статистичні методи або нейронні мережі, для аналізу даних і прогнозування вподобань. У коді для аналізу запитів використовується модель Gemini-1.5-flash, що є прикладом застосування сучасних технологій штучного інтелекту.

Сучасні рекомендаційні системи активно використовують технології штучного інтелекту та обробки природної мови, що дозволяє обробляти складні запити та надавати контекстно-залежні рекомендації. Наприклад, у створеній програмі ШІ-асистент аналізує текстові описи книг і запити користувача, що забезпечує гнучкість у підборі рекомендацій. Такі системи здатні не лише знаходити точні відповідності, але й пропонувати альтернативи, пояснюючи їхню релевантність [6]. Це особливо важливо для книжкових магазинів, де клієнти можуть мати специфічні запити, наприклад, книги в стилі певного автора чи на конкретну тему.

Однією з ключових проблем рекомендаційних систем є забезпечення точності та релевантності пропозицій. Неправильні рекомендації можуть знизити довіру користувача до платформи. Для підвищення точності сучасні системи використовують техніки глибокого навчання, які дозволяють моделювати складні залежності між даними користувача та продуктами. Однак такі методи потребують значних обчислювальних ресурсів і великих обсягів даних, що може бути викликом для невеликих магазинів [7]. У коді проєкту проблема частково вирішується за рахунок використання хмарного API, що зменшує потребу в локальних обчислювальних ресурсах.

Ще одним важливим аспектом є етичні питання, пов’язані з конфіденційністю даних користувачів. Рекомендаційні системи часто потребують збору особистих даних, таких як історія покупок чи пошукових запитів, що може викликати занепокоєння щодо приватності. У коді програми передбачено автентифікацію користувачів, що дозволяє обмежити доступ до

рекомендацій неавторизованим особам, однак питання захисту даних потребує додаткової уваги, особливо при використанні зовнішніх API [8].

Перспективи розвитку рекомендаційних систем пов’язані з інтеграцією нових технологій, таких як обробка природної мови, контекстно-залежні рекомендації та адаптивне навчання. У книжкових магазинах це може включати врахування настрою користувача, часу доби чи навіть його геолокації для надання більш персоналізованих пропозицій. Реалізована програма демонструє початковий етап такого розвитку, використовуючи ШІ для аналізу текстових запитів, але в майбутньому система може бути розширена за рахунок додавання контекстних даних або аналізу поведінки користувача.

Таким чином, рекомендаційні системи є потужним інструментом для персоналізації користувацького досвіду в інтернет-магазинах книг. Вони дозволяють ефективно вирішувати проблему вибору, підвищувати залученість клієнтів і сприяти зростанню продажів. У створеній програмі реалізована базова рекомендаційна система, яка використовує ШІ для аналізу запитів користувача, що є перспективним напрямком, але потребує подальшого вдосконалення для врахування ширшого спектра даних і підвищення точності рекомендацій.

1.2 Аналіз предметної області

Інтернет-магазини книг є важливою частиною сучасної електронної комерції, яка забезпечує доступ до широкого асортименту літератури через цифрові платформи. Предметна область цього дослідження охоплює розробку рекомендаційної системи для такого магазину, що передбачає аналіз особливостей книжкової індустрії, потреб користувачів, технологічних аспектів реалізації та специфіки взаємодії з клієнтами. Цей розділ присвячено детальному розгляду предметної області, включаючи ключові аспекти функціонування книжкових онлайн-платформ, характеристики цільової аудиторії, а також технологічні та інформаційні вимоги до створення ефективної рекомендаційної системи.

Сфера книжкових інтернет-магазинів характеризується високим рівнем конкуренції, оскільки споживачі мають доступ до численних платформ, таких як Amazon, Goodreads чи локальних аналогів. Основною особливістю таких магазинів є широкий асортимент товарів, що включає книги різних жанрів, авторів, форматів і цінових категорій. Книги як товар мають унікальні атрибути, такі як назва, автор, жанр, опис, ціна, наявність на складі та зображення обкладинки, що формують основу для створення рекомендацій. Крім того, книжковий ринок активно розвивається завдяки цифровізації, що дозволяє пропонувати не лише фізичні, а й електронні книги, аудіокниги та супутні продукти. Це створює потребу в персоналізованому підході до клієнтів, адже їхні вподобання можуть значно варіюватися залежно від особистих інтересів, читацьких звичок і цілей покупки [9].

Цільова аудиторія книжкових інтернет-магазинів є різноманітною і включає читачів різного віку, професій, культурного та соціального походження. Наприклад, одні користувачі можуть шукати художню літературу для дозвілля, інші – спеціалізовану літературу для навчання чи професійного розвитку. Аналіз потреб клієнтів показує, що сучасні користувачі цінують зручність інтерфейсу, швидкий доступ до інформації про книги та можливість отримати персоналізовані рекомендації, які відповідають їхнім інтересам. У цьому контексті рекомендаційна система стає ключовим інструментом для підвищення рівня задоволеності клієнтів, адже вона допомагає скоротити час на пошук потрібної книги та сприяє збільшенню продажів за рахунок пропозиції релевантного контенту [10].

Технологічна основа книжкових інтернет-магазинів включає клієнт-серверну архітектуру, яка забезпечує взаємодію між користувацьким інтерфейсом та серверною частиною, де зберігаються дані про книги, користувачів і замовлення. У коді проєкту клієнтська частина реалізована за допомогою бібліотеки Streamlit, яка забезпечує створення інтерактивного веб-інтерфейсу, а серверна – на базі фреймворку Flask із використанням бази даних SQLite. Така архітектура дозволяє ефективно обробляти запити користувачів,

зокрема для автентифікації, управління кошиком, оформлення замовлень і отримання рекомендацій. Важливим елементом є інтеграція з зовнішніми API, наприклад, Google Gemini, що використовується для аналізу текстових запитів користувачів і формування рекомендацій на основі їхніх побажань. Це відповідає сучасним тенденціям використання штучного інтелекту для обробки природної мови в рекомендаційних системах [11].

Ключовим аспектом предметної області є управління даними користувачів і товарів. У книжковому інтернет-магазині дані про користувачів включають інформацію про їхні профілі (логін, пароль, електронна пошта, роль), історію покупок і вподобання. Дані про книги охоплюють атрибути, такі як категорія, автор, ціна, опис і наявність. Ці дані є основою для роботи рекомендаційної системи, яка може використовувати як контентний підхід (аналіз характеристик книг), так і аналіз поведінки користувачів. Наприклад, система може пропонувати книги на основі попередніх покупок або аналізувати текстові запити користувача, як це реалізовано у функції get_gemini_recommendation у коді проєкту. Такий підхід дозволяє адаптувати рекомендації до індивідуальних потреб клієнта, що є важливим для забезпечення конкурентоспроможності магазину [12].

Ще однією особливістю предметної області є необхідність забезпечення безпеки та конфіденційності даних. Книжкові інтернет-магазини обробляють персональні дані користувачів, такі як адреса, номер телефону та історія замовлень, що вимагає використання захищених протоколів автентифікації, наприклад, JWT-токенів, як це реалізовано в серверній частині коду. Крім того, рекомендаційна система повинна враховувати етичні аспекти, такі як уникнення упередженості у пропозиціях книг і забезпечення прозорості рекомендацій для користувачів. Це особливо важливо в умовах зростання уваги до захисту даних і приватності [13].

Функціонування книжкового інтернет-магазину також передбачає управління замовленнями, що включає створення замовлення, оновлення його статусу та відображення деталей для користувачів і менеджерів. У коді проєкту

реалізована можливість для менеджерів змінювати статуси замовлень (наприклад, “нове”, “оброблене”, “доставлене”), що відображає реальні бізнес-процеси в електронній комерції. Рекомендаційна система в такому контексті може використовувати дані про замовлення для вдосконалення пропозицій, наприклад, пропонуючи книги, які часто купують разом із уже обраними [14].

Інтеграція рекомендаційної системи з іншими функціями магазину, такими як пошук за категоріями чи фільтрація, є ще одним важливим аспектом. У коді клієнтської частини передбачено фільтрацію книг за категоріями та пошуковими запитами, що доповнює функціонал рекомендаційної системи. Наприклад, користувач може вказати інтерес до певного жанру, а система на основі цього запиту за допомогою Google Gemini запропонує відповідні книги. Така інтеграція дозволяє створити цілісний користувацький досвід, де рекомендації гармонійно поєднуються з іншими функціями платформи [15].

Отже, предметна область розробки рекомендаційної системи для книжкового інтернет-магазину охоплює аналіз особливостей книжкової індустрії, потреб користувачів, технологічної інфраструктури та вимог до безпеки даних. Книжковий ринок вимагає персоналізованого підходу до клієнтів, що забезпечується за рахунок використання сучасних технологій штучного інтелекту та ефективного управління даними. Розуміння цих аспектів дозволяє сформулювати вимоги до рекомендаційної системи, яка має бути інтуїтивною, безпечною та адаптованою до різноманітних потреб користувачів.

1.3 Постановка завдання

Розробка рекомендаційної системи для інтернет-магазину книг є актуальним завданням, спрямованим на підвищення якості обслуговування клієнтів, зростання продажів та персоналізації взаємодії з користувачами. Метою створення такої системи є забезпечення користувачів релевантними рекомендаціями книг на основі їхніх уподобань, що сприяє підвищенню рівня задоволеності та лояльності клієнтів [16]. Основним завданням системи є аналіз

запитів користувача, зіставлення їх із наявним асортиментом книг та формування персоналізованих пропозицій, які максимально відповідають інтересам і потребам клієнтів.

Для реалізації поставленої мети необхідно розробити програмний продукт, який інтегрується з інтернет-магазином книг, використовуючи сучасні технології обробки природної мови та аналізу даних. Система має базуватися на гібридному підході, що поєднує контентно-орієнтовані методи та можливості штучного інтелекту для обробки текстових запитів користувачів [17]. Зокрема, у коді проєкту використано Google Gemini API для аналізу запитів та генерації рекомендацій, що дозволяє враховувати як точні відповідності, так і пропонувати альтернативні книги, якщо запит не має прямих збігів у каталозі.

Основні функціональні вимоги до системи включають можливість користувача вводити текстові запити щодо бажаних книг, обробку цих запитів з урахуванням наявного асортименту та формування пояснень щодо запропонованих рекомендацій. Система має забезпечувати інтуїтивно зрозумілий інтерфейс, що дозволяє клієнтам легко формулювати свої побажання та отримувати швидкі відповіді [18]. Крім того, важливим аспектом є інтеграція з базою даних магазину для забезпечення актуальності інформації про книги, їхню наявність та характеристики, що реалізовано через API-запити до серверної частини програми.

Серед нефункціональних вимог особливу увагу приділено швидкості обробки запитів, оскільки затримки можуть негативно вплинути на досвід користувача. Система має бути стійкою до помилок, таких як відсутність підключення до сервера чи некоректні запити, що забезпечується обробкою винятків у коді [19]. Безпека даних є ще одним важливим аспектом, адже система обробляє персональні дані користувачів, такі як історія замовлень та профільні дані, що вимагає використання захищених методів автентифікації, таких як JWT-токени [20].

Технічна реалізація передбачає створення клієнтської частини на основі бібліотеки Streamlit для забезпечення зручного веб-інтерфейсу та серверної

частини на базі Flask для обробки запитів і взаємодії з базою даних SQLite [21]. Рекомендаційна система має бути модульною, щоб забезпечити можливість подальшого розширення, наприклад, додавання нових джерел даних чи алгоритмів рекомендацій [22]. Важливо також забезпечити адаптивність інтерфейсу для різних пристроїв, що досягається за рахунок використання гнучких CSS-стилів у клієнтській частині.

Очікуваним результатом є підвищення ефективності взаємодії користувачів з інтернет-магазином шляхом надання персоналізованих рекомендацій, що базуються на їхніх запитах. Це сприятиме збільшенню кількості покупок, зменшенню часу на пошук потрібних книг і покращенню загального досвіду користувача [23]. Успішна реалізація системи дозволить автоматизувати процес підбору книг, зменшивши навантаження на персонал магазину, особливо в частині консультацій клієнтів. Перспективи подальшого розвитку включають інтеграцію додаткових джерел даних, таких як історія переглядів чи оцінок книг, для підвищення точності рекомендацій [24].

1.4 Висновок до першого розділу

У першому розділі кваліфікаційної роботи проведено комплексний аналіз предметної області та сформульовано завдання розробки рекомендаційної системи для інтернет-магазину книг. Встановлено, що рекомендаційні системи є важливим інструментом для персоналізації користувацького досвіду, сприяючи підвищенню залученості клієнтів і зростанню продажів. Розглянуто основні типи таких систем – контентно-орієнтовані, колаборативні та гібридні, а також їхні переваги й обмеження, що дозволяють ефективно вирішувати проблему інформаційного перевантаження. Особливу увагу приділено застосуванню сучасних технологій штучного інтелекту, зокрема обробки природної мови, для аналізу запитів користувачів, що реалізовано в розробленій програмі через Google Gemini API. Аналіз предметної області показав, що книжкові інтернет-магазини характеризуються широким асортиментом і різноманітною цільовою

аудиторією, що вимагає персоналізованого підходу до клієнтів. Визначено ключові аспекти функціонування таких платформ, включаючи управління даними, безпеку та інтеграцію з іншими функціями магазину. Постановка завдання передбачає створення модульної системи з інтуїтивним інтерфейсом, яка поєднує контентно-орієнтовані методи та можливості ШІ для забезпечення релевантних рекомендацій. Очікується, що реалізація системи сприятиме покращенню користувацького досвіду, підвищенню ефективності продажів і автоматизації консультаційних процесів, з перспективами подальшого вдосконалення шляхом інтеграції додаткових даних і алгоритмів.

РОЗДІЛ 2. ПРОЕКТНА РОЗРОБКА РЕКОМЕНДАЦІЙНОЇ СИСТЕМИ

2.1 Вимоги до системи

Розробка рекомендаційної системи для інтернет-магазину книг базується на необхідності підвищення якості користувацького досвіду, автоматизації підбору літератури та оптимізації бізнес-процесів. Аналіз кодів проєкту (Додаток А) дозволяє сформулювати ключові вимоги до системи, які охоплюють функціональні, нефункціональні аспекти, а також вимоги до безпеки та інтеграції.

Функціональні вимоги. Система повинна забезпечувати персоналізовані рекомендації книг на основі запитів користувачів, введених у текстовій формі. Для цього використовується Google Gemini API, який аналізує побажання клієнта та зіставляє їх із каталогом книг, доступних у базі даних [25]. Система підтримує автентифікацію та авторизацію користувачів через JWT-токени, дозволяючи розрізняти клієнтів і менеджерів, що забезпечує доступ до персоналізованих функцій, таких як перегляд історії замовлень чи управління каталогом [26]. Інтерфейс користувача, реалізований через Streamlit, передбачає інтуїтивно зрозумілу навігацію з підтримкою таких сторінок, як каталог, кошик, профіль, замовлення та рекомендації. Для менеджерів додатково доступні функції управління книгами (додавання, редагування, видалення) та замовленнями (зміна статусу). Система підтримує фільтрацію книг за категоріями та пошуковими запитами, а також управління кошиком із можливістю додавання, видалення та оновлення кількості товарів.

Нефункціональні вимоги. Система має бути швидкодійною, забезпечуючи миттєву обробку запитів на рекомендації та доступ до даних каталогу. Для цього використовується SQLite як легка база даних, що забезпечує достатню продуктивність для малого та середнього обсягу даних [27]. Інтерфейс повинен бути адаптивним і візуально привабливим, що реалізовано через кастомізовані CSS-стилі з чітко визначеною кольоровою гамою. Система має бути

масштабовною, дозволяючи додавання нових книг і категорій без значних змін у коді. Для забезпечення стабільності передбачено обробку помилок, таких як втрата з'єднання з сервером чи некоректні запити, із відповідними повідомленнями користувачу.

Вимоги до безпеки. Безпека даних користувачів забезпечується через шифрування паролів алгоритмом SHA-256 перед збереженням у базі даних [28]. JWT-токени використовуються для захисту API-маршрутів, із перевіркою валідності та терміну дії токена. Для менеджерських функцій застосовується додатковий рівень авторизації через декоратор @manager_required, що обмежує доступ до адміністративних операцій [29]. Система також захищає від SQL-ін'єкцій завдяки параметризованим запитам до бази даних.

Вимоги до інтеграції. Система інтегрується з Google Gemini API для обробки текстових запитів і генерації рекомендацій, що вимагає наявності валідного API-ключа та стабільного інтернет-з'єднання [30]. API сервера (Flask) забезпечує взаємодію між клієнтською частиною та базою даних через RESTful-запити, підтримуючи методи GET, POST, PUT, DELETE. Для коректної роботи передбачено використання CORS для обробки міжсайтових запитів.

Таким чином, сформульовані вимоги забезпечують створення ефективної рекомендаційної системи, яка поєднує персоналізацію, зручність використання, безпеку та інтеграцію з сучасними технологіями, що відповідає потребам користувачів інтернет-магазину книг.

2.2 Архітектура системи

Архітектура рекомендаційної системи для інтернет-магазину книг, представлена у файлах програмного проєкту client.py та server.py, побудована за клієнт-серверною моделлю з використанням сучасних технологій веб-розробки та штучного інтелекту. Система складається з двох основних компонентів: клієнтської частини, реалізованої за допомогою Streamlit, та серверної частини, побудованої на фреймворку Flask із базою даних SQLite. Клієнтська частина

забезпечує інтерактивний інтерфейс користувача для взаємодії з каталогом книг, кошиком, профілем, замовленнями та рекомендаційною системою. Серверна частина обробляє запити, керує даними та забезпечує автентифікацію користувачів. Рекомендаційна система інтегрована через Google Gemini API, що дозволяє генерувати персоналізовані рекомендації на основі текстових запитів користувачів. Архітектура системи розроблена з урахуванням модульності, безпеки та масштабованості, що забезпечує її ефективність для електронної комерції.

Клієнтська частина реалізована на Streamlit, який забезпечує швидку розробку інтерактивних веб-додатків [30]. Вона включає набір сторінок, таких як каталог книг, кошик, профіль користувача, сторінка замовлень, а також спеціалізовану сторінку рекомендацій. Користувацький інтерфейс використовує кастомізовані CSS-стилі для створення привабливого дизайну з кольоровою гамою, що відповідає бренду магазину. Для управління станом додатку застосовується st.session_state, що дозволяє зберігати дані про кошик, авторизацію та поточну сторінку. Наприклад, функція set_page забезпечує навігацію між сторінками, а debug_print фіксує логування для відстеження дій користувача.

Для опису структури клієнтської частини доцільно використовувати діаграму класів UML (рисунок 2.1), яка відображає основні компоненти клієнтського додатку (наприклад, класи для управління кошиком, профілем, каталогом) та їх взаємозв’язки. Ця діаграма ілюструє, як функції клієнтської частини взаємодіють із сесійним станом та API-сервером.

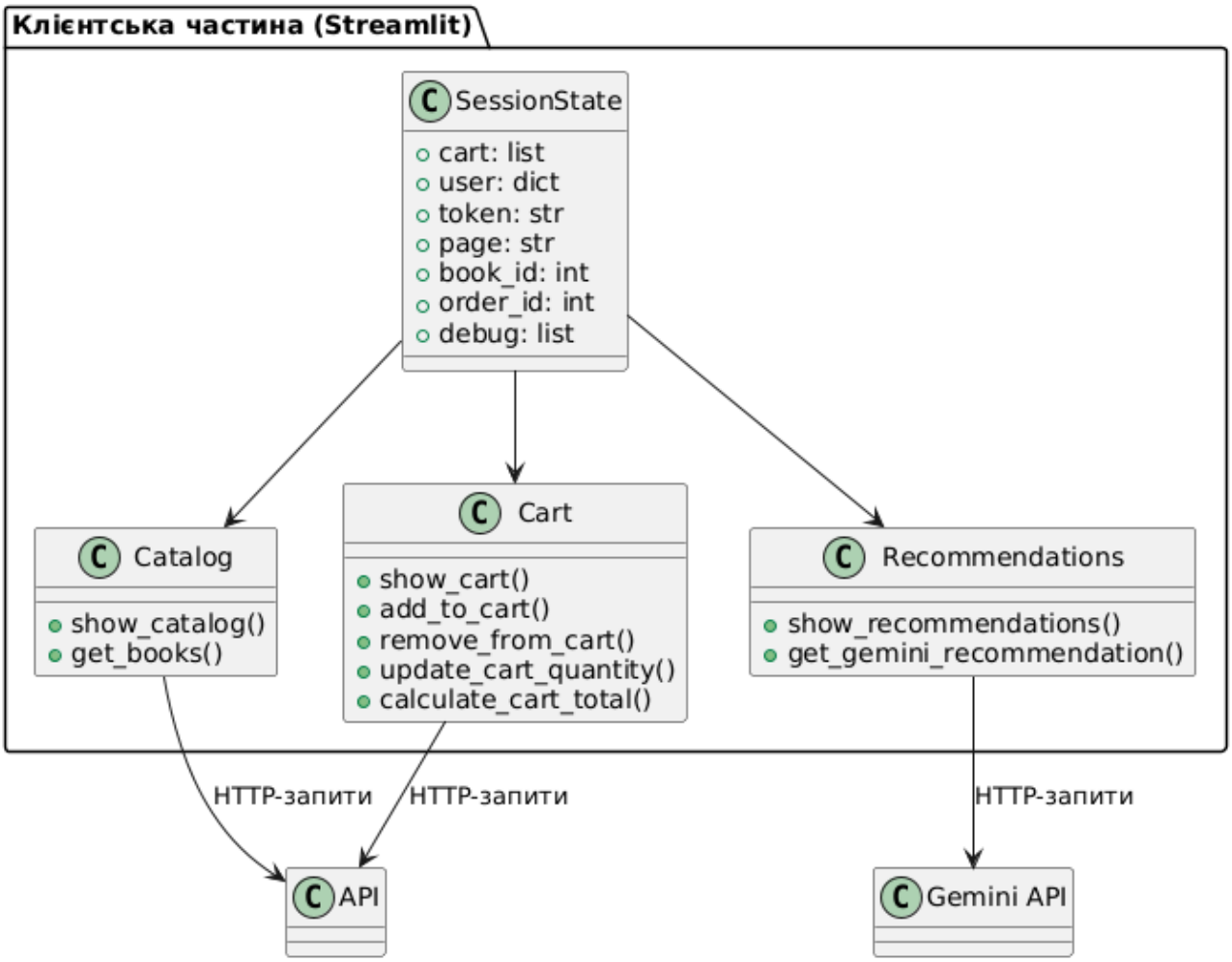

Рисунок 2.1 – Діаграма класів

Серверна частина побудована на Flask, легковаговому фреймворку для створення RESTful API [29]. Вона обробляє запити від клієнтської частини, такі як автентифікація, управління книгами, замовленнями та категоріями. База даних SQLite використовується для зберігання інформації про користувачів, книги, замовлення та їх деталі [27]. Таблиці бази даних (users, books, orders, order_items) пов’язані через зовнішні ключі, що забезпечує реляційну цілісність даних. Для захисту доступу до API застосовується JWT-автентифікація, яка перевіряє токени через декоратори token_required і manager_required [26]. Наприклад, маршрут /api/orders дозволяє менеджерам отримувати всі замовлення, а клієнтам — лише власні.

Для опису серверної логіки доречною є діаграма послідовності UML (рисунок 2.2), яка показує послідовність викликів між клієнтом, сервером і базою

даних під час виконання запиту, наприклад, створення замовлення. Ця діаграма ілюструє, як клієнтський запит обробляється сервером, включаючи перевірку автентифікації та оновлення даних.

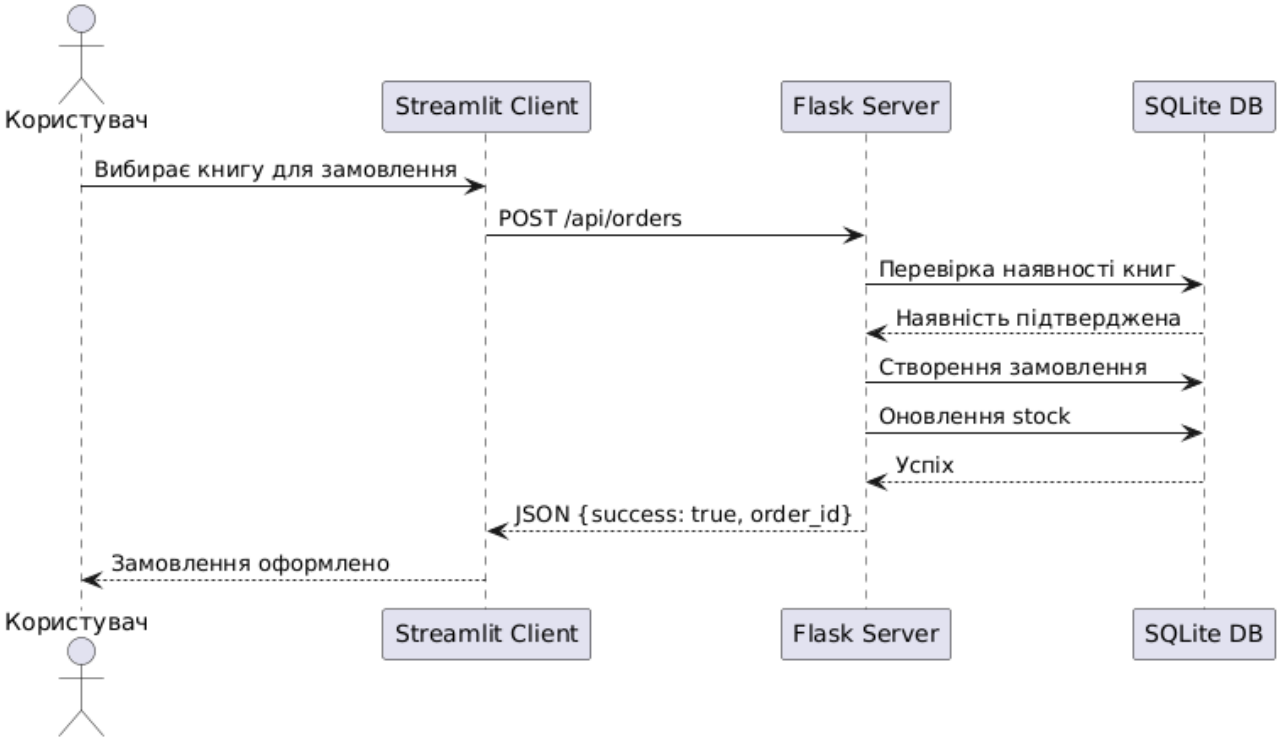

Рисунок 2.2 – Діаграма послідовності

Рекомендаційна система реалізована через інтеграцію з Google Gemini API, що дозволяє обробляти текстові запити користувачів і генерувати рекомендації на основі каталогу книг [25]. Функція get_gemini_recommendation формує текстовий контекст із інформацією про книги (назва, автор, категорія, опис, ціна) і передає його разом із запитом користувача до моделі Gemini-1.5-flash. Модель аналізує запит, шукає точні відповідності в каталозі або пропонує альтернативні книги, надаючи пояснення вибору. Наприклад, якщо користувач шукає книгу про штучний інтелект, система може порекомендувати книгу з відповідною категорією або схожим описом.

Для моделювання роботи рекомендаційної системи підходить діаграма варіантів використання UML (рисунок 2.3), яка відображає взаємодію користувача з функцією рекомендацій, включаючи введення запиту, обробку

API та відображення результатів. Ця діаграма підкреслює ролі користувача та системи в процесі генерації рекомендацій.

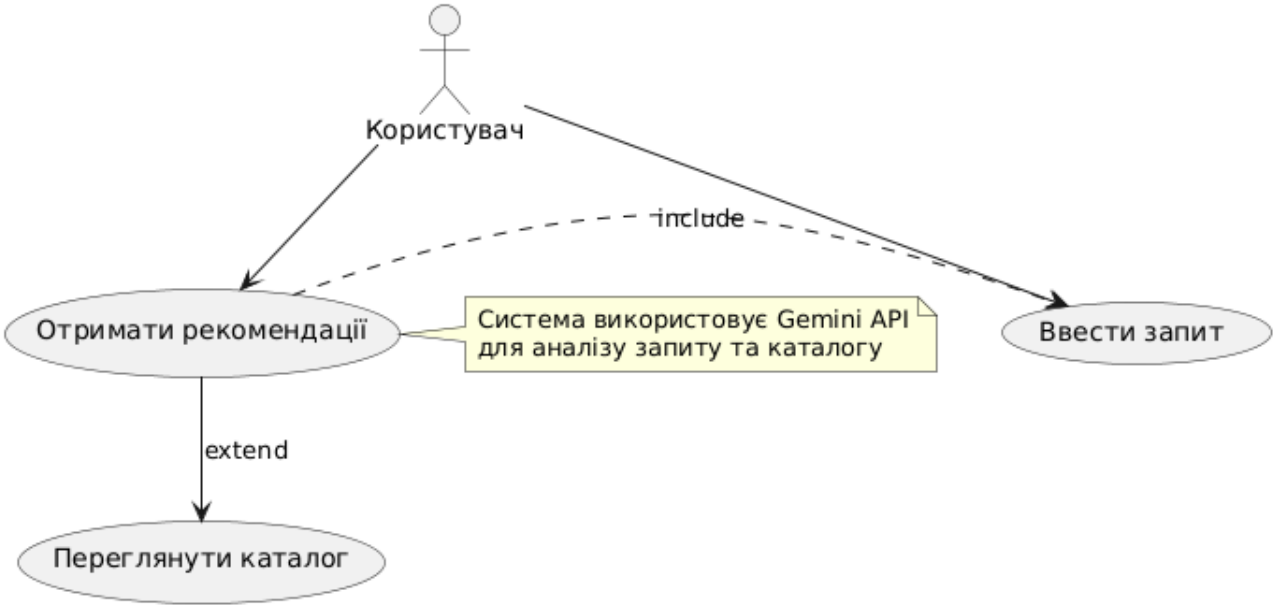

Рисунок 2.3 – Діаграма варіантів використання

Клієнтська та серверна частини взаємодіють через RESTful API, де клієнт надсилає HTTP-запити (GET, POST, PUT, DELETE) до сервера, який повертає JSON-відповіді. Наприклад, запит до /api/books з параметрами категорії чи пошуку повертає відфільтрований список книг. Безпека забезпечується через хешування паролів за допомогою SHA-256 [28] та JWT-токени, які перевіряють автентичність користувача та його роль. Рекомендаційна система працює асинхронно: клієнтський додаток надсилає запит до Gemini API, а отримані рекомендації відображаються на сторінці рекомендацій.

Архітектура системи є модульною, що дозволяє легко додавати нові функції, наприклад, нові типи рекомендацій чи додаткові ролі користувачів. Використання SQLite як легковагової бази даних підходить для прототипу, але для масштабування системи доцільно перейти на більш потужну СУБД, таку як PostgreSQL [31]. Крім того, інтеграція з Gemini API може бути розширена шляхом додавання кешування запитів для зменшення навантаження на сервер [32].

Для опису загальної структури системи доречна діаграма компонентів UML (рисунок 2.4), яка показує взаємозв’язки між клієнтською частиною, сервером, базою даних та зовнішнім Gemini API. Ця діаграма підкреслює модульність системи та розподіл обов’язків між компонентами.

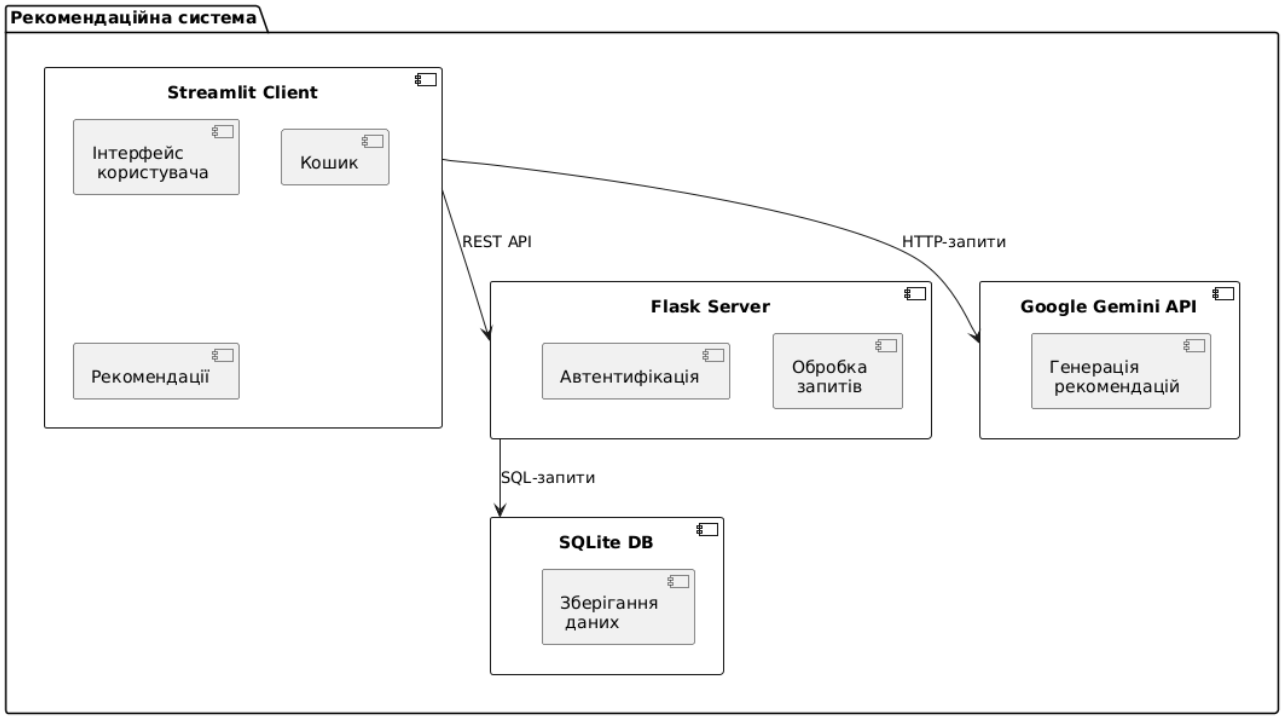

Рисунок 2.4 – Діаграма компонентів

Система використовує JWT для автентифікації, що забезпечує захист від несанкціонованого доступу [26]. Хешування паролів за допомогою SHA-256 підвищує безпеку зберігання даних [28]. Для підвищення продуктивності застосовується асинхронна обробка запитів до Gemini API, а також локальне зберігання стану кошика в st.session_state. Однак для великих обсягів даних може знадобитися оптимізація, наприклад, індексація бази даних чи кешування популярних запитів [32].

Архітектура рекомендаційної системи для інтернет-магазину книг поєднує клієнт-серверну модель із сучасними технологіями штучного інтелекту, забезпечуючи зручність, безпеку та персоналізацію. Streamlit забезпечує інтуїтивний інтерфейс, Flask і SQLite — надійну серверну логіку, а Google

Gemini API додає інтелектуальні можливості для рекомендацій. Модульна структура дозволяє легко розширювати функціонал, а використання UML-діаграм (класів, послідовності, варіантів використання, компонентів) сприяє чіткому розумінню системи та її подальшому вдосконаленню.

2.3 Методи рекомендацій

Розробка рекомендаційної системи для інтернет-магазину книг, представлена в кодових файлах (client.py та server.py), базується на використанні сучасних підходів до обробки природної мови та аналізу даних для забезпечення персоналізованих рекомендацій користувачам. Важливо розглянути методи, які застосовуються для створення рекомендацій, їх особливості, а також принципи їх інтеграції в систему (рисунок 2.5).

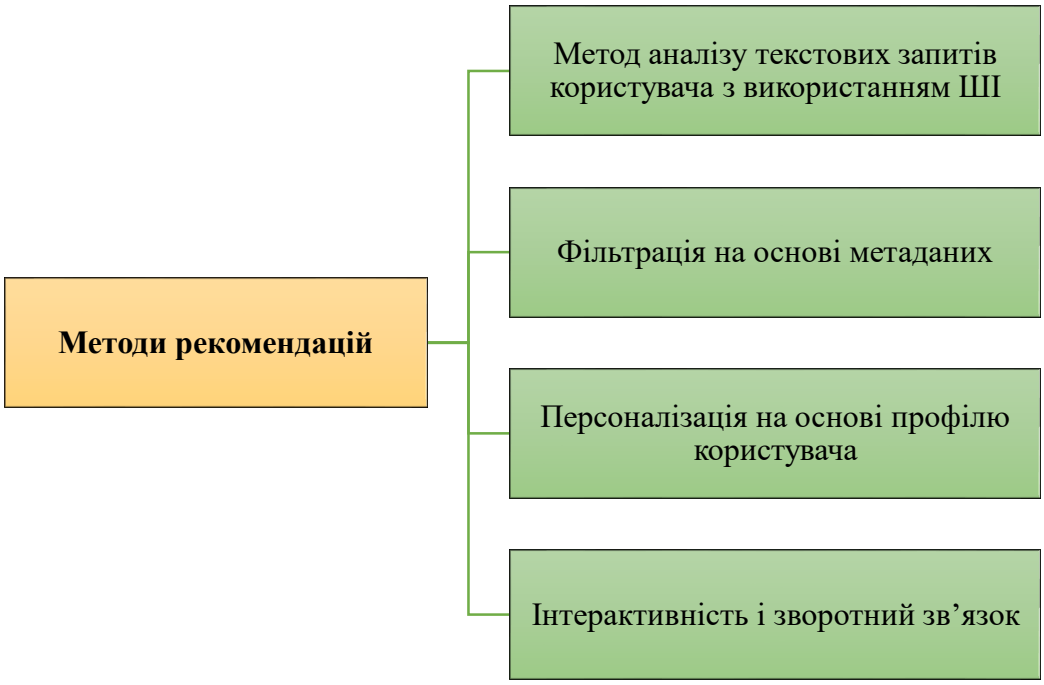

Рисунок 2.5 – Методи застосовані для створення рекомендацій

Метод аналізу текстових запитів користувача з використанням ШІ. Основним методом рекомендаційної системи є аналіз текстових запитів користувача за допомогою моделі штучного інтелекту Google Gemini API,

зокрема моделі gemini-1.5-flash. Цей підхід дозволяє обробляти природномовні запити, в яких користувач описує свої вподобання, наприклад, бажані жанри, теми чи стилі книг. У коді функція get_gemini_recommendation формує контекст, що включає повний каталог книг, та передає його разом із запитом користувача до моделі ШІ. Модель виконує два основні етапи:

Етап 1. Пошук точних відповідностей

ШІ аналізує каталог книг і перевіряє, чи є книги, які прямо відповідають запиту користувача (наприклад, за назвою, автором чи категорією).

Етап 2. Пропозиція альтернатив

Якщо точних відповідностей немає, модель пропонує книги, які максимально відповідають інтересам користувача, пояснюючи, чому вони можуть бути цікавими. Наприклад, якщо користувач шукає "фентезі з драконами", але таких книг немає, ШІ може порекомендувати фентезі з магією чи епічними пригодами.

Цей метод є контекстно-залежним і дозволяє обробляти складні та різноманітні запити, враховуючи як явні (жанр, автор), так і неявні (настрій, стиль) побажання користувача. Використання моделі ШІ забезпечує гнучкість і адаптивність рекомендацій, що є важливим для підвищення задоволення користувачів.

Фільтрація на основі метаданих. Допоміжним методом є фільтрація книг за метаданими, що реалізується через API-запити до серверної частини (server.py). Користувач може вказати категорію чи ключові слова для пошуку, які передаються як параметри до маршруту /api/books. Сервер виконує SQL-запит до бази даних, фільтруючи книги за категорією або вмістом назви чи автора. Цей метод є базовим і використовується для первинного відбору книг перед передачею їх до ШІ для глибшого аналізу. Наприклад, якщо користувач обирає категорію "Фантастика" та вводить пошуковий запит "космос", система повертає книги, що відповідають цим критеріям, які потім можуть бути уточнені ШІ.

Персоналізація на основі профілю користувача. Система враховує персоналізацію через збереження даних користувача в сесії (st.session_state.user). Хоча в поточній реалізації рекомендації не базуються безпосередньо на історії покупок чи переглядів, профіль користувача (наприклад, роль "клієнт" чи "менеджер") визначає доступ до функції рекомендацій. У майбутньому метод може бути розширений шляхом аналізу попередніх замовлень (/api/orders) для створення профілю вподобань користувача, що дозволить пропонувати книги на основі його минулих покупок або переглядів.

Інтерактивність і зворотний зв’язок. Рекомендаційна система підтримує інтерактивність через форму введення запитів (show_recommendations), де користувач може описати свої побажання. Для спрощення взаємодії надаються приклади запитів (наприклад, "Шукаю детективи в стилі Агати Крісті"), які допомагають користувачам сформулювати чіткі запити. Зворотний зв’язок реалізується через відображення результатів у зрозумілому вигляді з поясненнями від ШІ, що підвищує довіру до рекомендацій. Користувач також може перейти до каталогу для подальшого ознайомлення з книгами.

Використання Google Gemini API забезпечує високу якість рекомендацій завдяки здатності ШІ розуміти контекст і обробляти природну мову. Однак метод залежить від наявності API-ключа та стабільного з’єднання з сервером Google, що може створювати обмеження в разі технічних збоїв. Фільтрація за метаданими є швидкою та ефективною, але менш гнучкою, оскільки обмежується чіткими критеріями (категорія, ключові слова). Персоналізація на основі профілю користувача поки що обмежена, але має потенціал для розвитку шляхом інтеграції історії покупок чи оцінок.

Рекомендаційна система інтернет-магазину книг поєднує сучасні методи обробки природної мови з традиційною фільтрацією за метаданими, створюючи ефективний інструмент для надання персоналізованих рекомендацій. Використання ШІ дозволяє обробляти складні запити, тоді як фільтрація забезпечує швидкий доступ до релевантних книг. У майбутньому система може

бути вдосконалена шляхом додавання аналізу поведінки користувача та машинного навчання для підвищення точності рекомендацій.

2.4 Проектування інтерфейсу

Проектування інтерфейсу рекомендаційної системи для інтернет-магазину книг є ключовим етапом розробки, оскільки від якості інтерфейсу залежить зручність взаємодії користувачів із системою, їхнє залучення та задоволеність. Інтерфейс має бути інтуїтивно зрозумілим, естетично привабливим і функціональним, щоб забезпечити ефективну взаємодію як для клієнтів, так і для менеджерів. У даній системі, реалізованій у коді (файли client.py та server.py), використано бібліотеку Streamlit для створення веб-інтерфейсу, що забезпечує швидке прототипування та адаптивність. Дизайн інтерфейсу базується на принципах мінімалізму, чіткості та зручності навігації, що відповідає сучасним стандартам веб-додатків. Колірна палітра включає відтінки синього, зеленого та нейтральних тонів, що створюють приємну атмосферу та сприяють комфортному сприйняттю інформації.

Основні принципи проектування інтерфейсу:

– Інтуїтивність гарантує, що інтерфейс відповідає потребам різних категорій користувачів, включаючи клієнтів, які шукають книги, та менеджерів, що керують асортиментом і замовленнями. Навігація реалізована через бічну панель із чіткими кнопками, що значно скорочує час на пошук необхідних функцій.

– Адаптивність забезпечується використанням Streamlit, що дозволяє коректно відображати систему на різних пристроях. Завдяки цьому користувачі можуть комфортно взаємодіяти із застосунком незалежно від того, чи використовують настільний комп’ютер, планшет або смартфон.

– Візуальна ієрархія структурує інформацію таким чином, щоб заголовки, підзаголовки, картки та контрастні кольори сприяли легкому сприйняттю даних, роблячи інтерфейс зрозумілим та впорядкованим.

– Відгук системи реалізований через миттєві повідомлення, що інформують користувача про успішні або помилкові дії. Наприклад, система повідомляє про успішне додавання книги до кошика або виникнення помилки при оновленні статусу, що сприяє підвищенню довіри користувачів.

– Інтеграція рекомендаційної системи передбачає можливість введення текстових запитів на відповідній сторінці, а отримані відповіді від Google Gemini відображаються у структурованому вигляді, що спрощує їхню інтерпретацію та покращує користувацький досвід.

Форма входу (show_login_form) є початковою точкою для авторизації користувачів (рисунок 2.6). Вона містить два текстові поля для введення імені користувача та пароля, а також кнопку "Увійти". Інтерфейс виконаний у світлих тонах із контрастними елементами (наприклад, синьо-зелені кнопки на світлому фоні). У разі помилки введення даних (наприклад, невірний пароль) відображається повідомлення про помилку червоного кольору. Після успішного входу користувач перенаправляється на сторінку каталогу. Форма проста, мінімалістична, але функціональна, що відповідає потребам швидкого доступу до системи.

Рисунок 2.6 – Екранна форма входу

Форма реєстрації (show_register_form) призначена для створення нових облікових записів (рисунок 2.7). Вона включає поля для введення імені користувача, електронної пошти, пароля, підтвердження пароля, ролі (клієнт або

менеджер), а також додаткові поля для повного імені, адреси та телефону. Поля розташовані у двох колонках для компактності, а кнопка "Зареєструватися" має акцентний колір, що привертає увагу. У разі успішної реєстрації користувач отримує повідомлення та перенаправляється на сторінку входу. Форма підтримує валідацію, наприклад, перевірку збігу паролів, що підвищує її надійність.

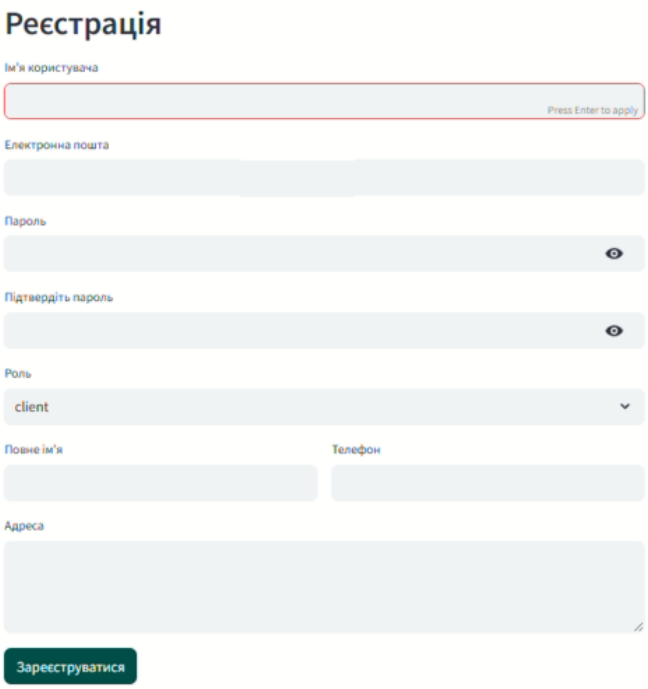

Рисунок 2.7 – Екранна форма реєстрації

Сторінка каталогу (show_catalog) є центральною для клієнтів, де відображається список доступних книг (рисунок 2.8). Інтерфейс включає фільтри за категоріями (випадаючий список) та пошуковий рядок для швидкого знаходження книг. Книги представлені у вигляді карток, розташованих у сітці по три в ряд. Кожна картка містить назву, автора, ціну, кількість на складі та кнопки "Деталі" та "До кошика". Картки оформлені у світлому кольорі з акцентами для цін і назв, що полегшує сприйняття. Фільтри дозволяють користувачам швидко звузити вибір, а кнопки забезпечують інтуїтивну взаємодію.

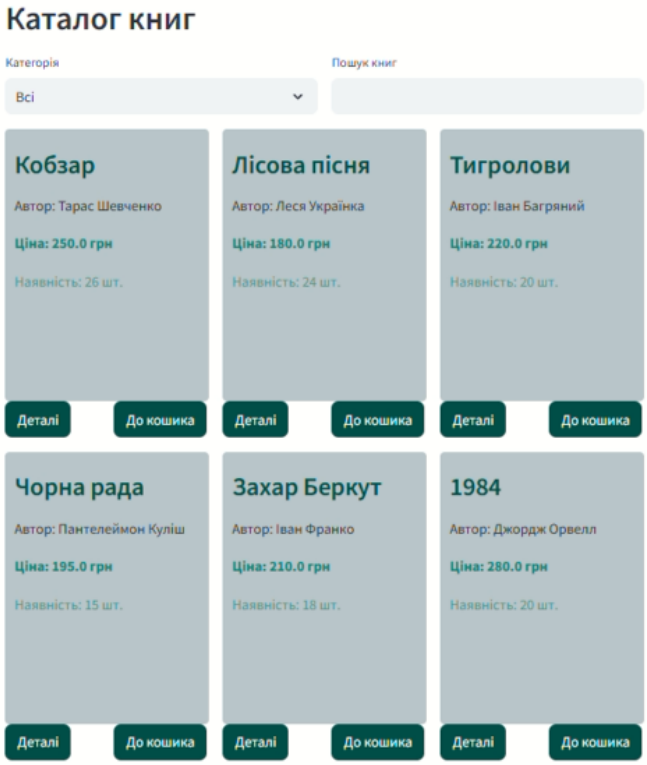

Рисунок 2.8 – Екранна форма каталогу

Сторінка деталей книги (show_book_details) відображає повну інформацію про обрану книгу, включаючи назву, автора, категорію, ціну, наявність, опис та зображення (або заповнювач, якщо зображення відсутнє) (рисунок 2.9). Інтерфейс поділений на дві колонки: зображення зліва, інформація справа. Кнопки "Назад до каталогу" та "Додати до кошика" розташовані внизу, що забезпечує зручність навігації. Сторінка використовує чітку типографіку та контрастні кольори для виділення ключової інформації, наприклад, ціни.

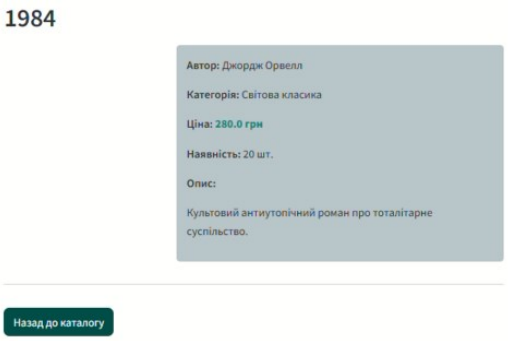

Рисунок 2.9 – Екранна форма сторінки деталей книги

Сторінка кошика (show_cart) відображає список доданих книг із можливістю зміни кількості або видалення товарів (рисунок 2.10). Кожна книга представлена карткою з назвою, ціною, кількістю та сумою. Загальна сума кошика виділена контрастним кольором у верхній частині. Кнопки "Продовжити покупки" та "Оформити замовлення" дозволяють користувачу або повернутися до каталогу, або завершити покупку. Інтерфейс забезпечує швидкий доступ до редагування кошика та чітке відображення фінансової інформації.

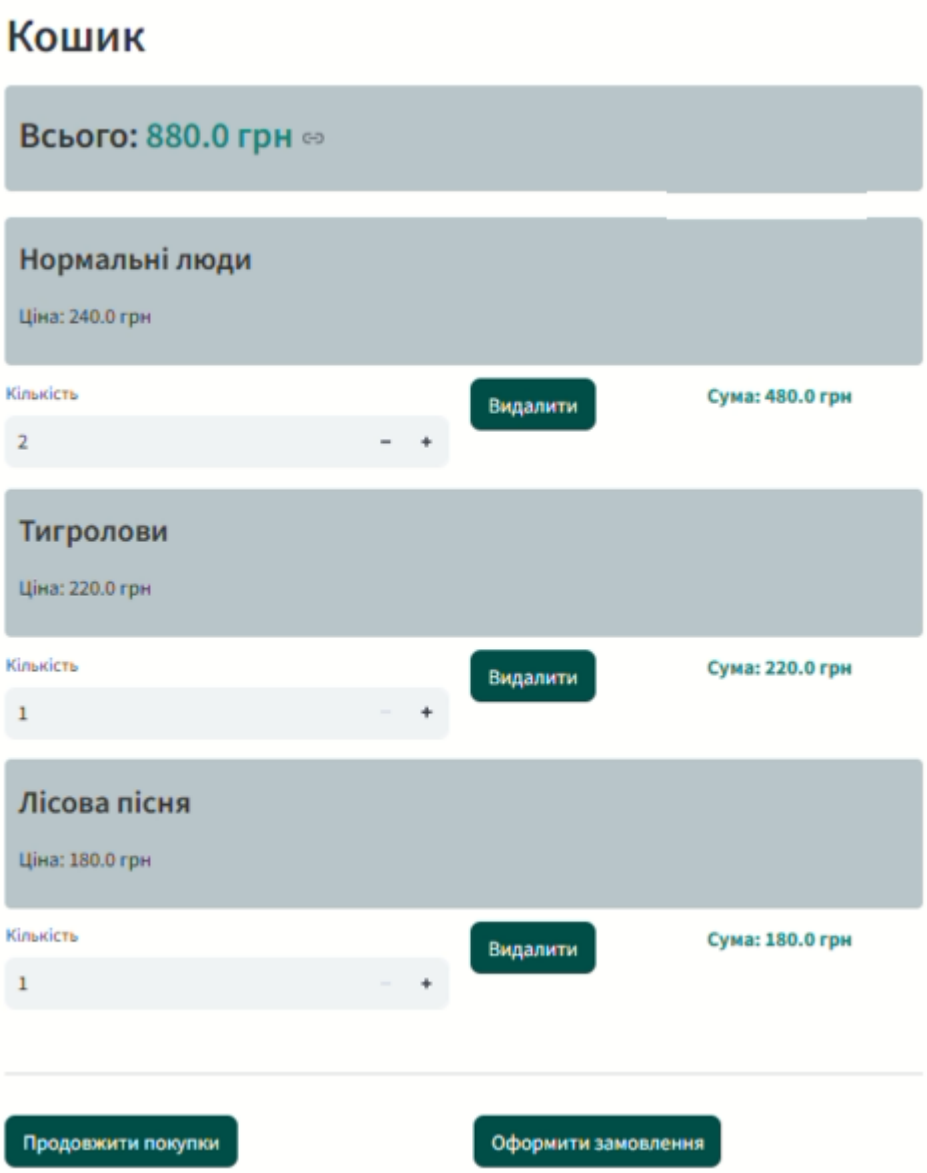

Рисунок 2.10 – Екранна форма кошика

Сторінка рекомендацій (show_recommendations) дозволяє користувачам вводити текстові запити для отримання персоналізованих пропозицій (рисунок 2.11). Форма включає текстове поле для опису побажань і кнопку "Отримати рекомендації". Відповідь від Google Gemini відображається у структурованому вигляді з поясненнями, чому запропоновані книги підходять. Додатково надаються приклади запитів, розподілені по двох колонках, що мотивують користувачів спробувати різні запити. Інтерфейс має інформативний блок із поясненням роботи системи, що підвищує довіру.

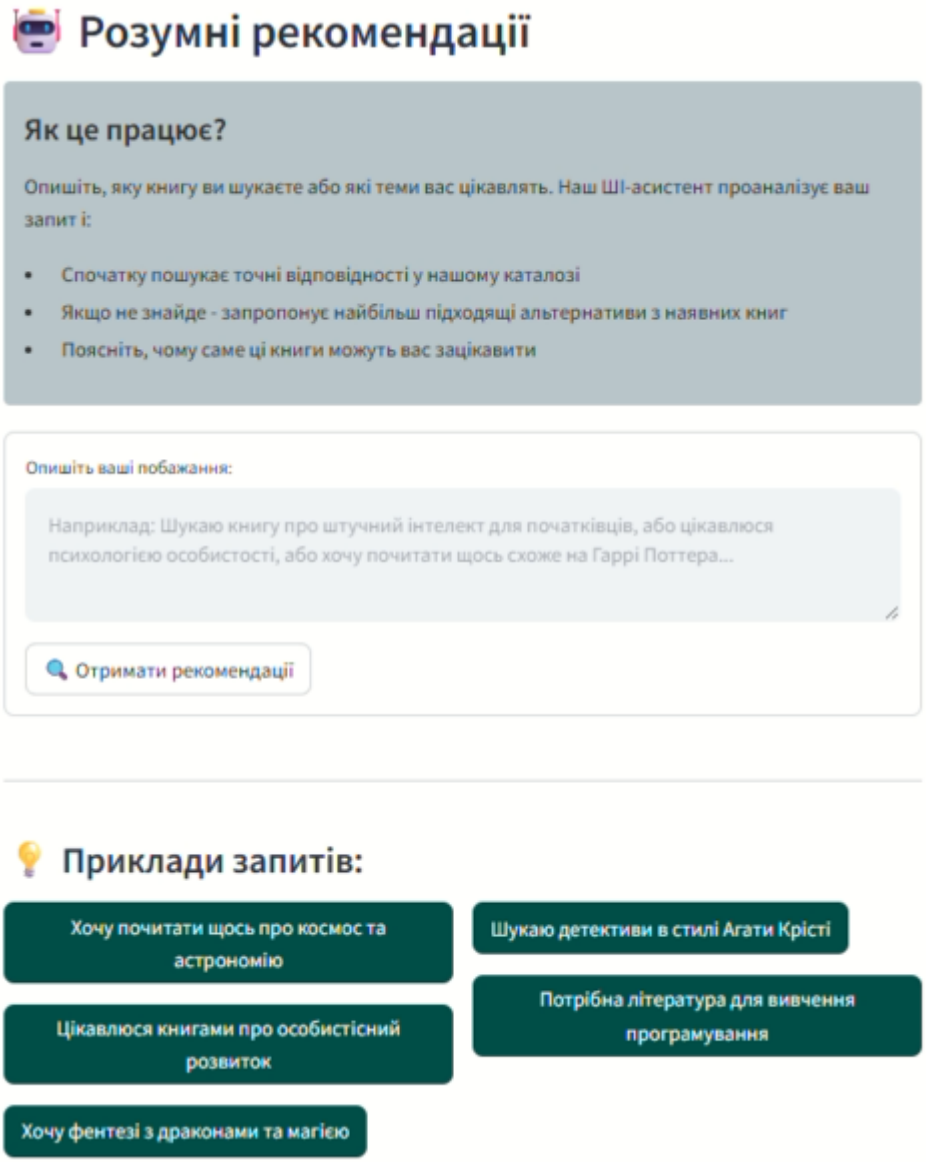

Рисунок 2.11 – Екранна форма рекомендацій

Сторінка управління товарами (show_manage_books) доступна лише менеджерам і містить вкладки для перегляду списку книг та додавання нових (рисунок 2.12). У списку книг кожна позиція відображається у картці з інформацією про назву, автора, ціну та кількість. Кнопки "Редагувати" та "Видалити" дозволяють швидко вносити зміни. Форма додавання/редагування книги включає поля для назви, автора, категорії, ціни, кількості, опису та URL зображення. Інтерфейс чітко структурований, із акцентом на функціональність для менеджерів.

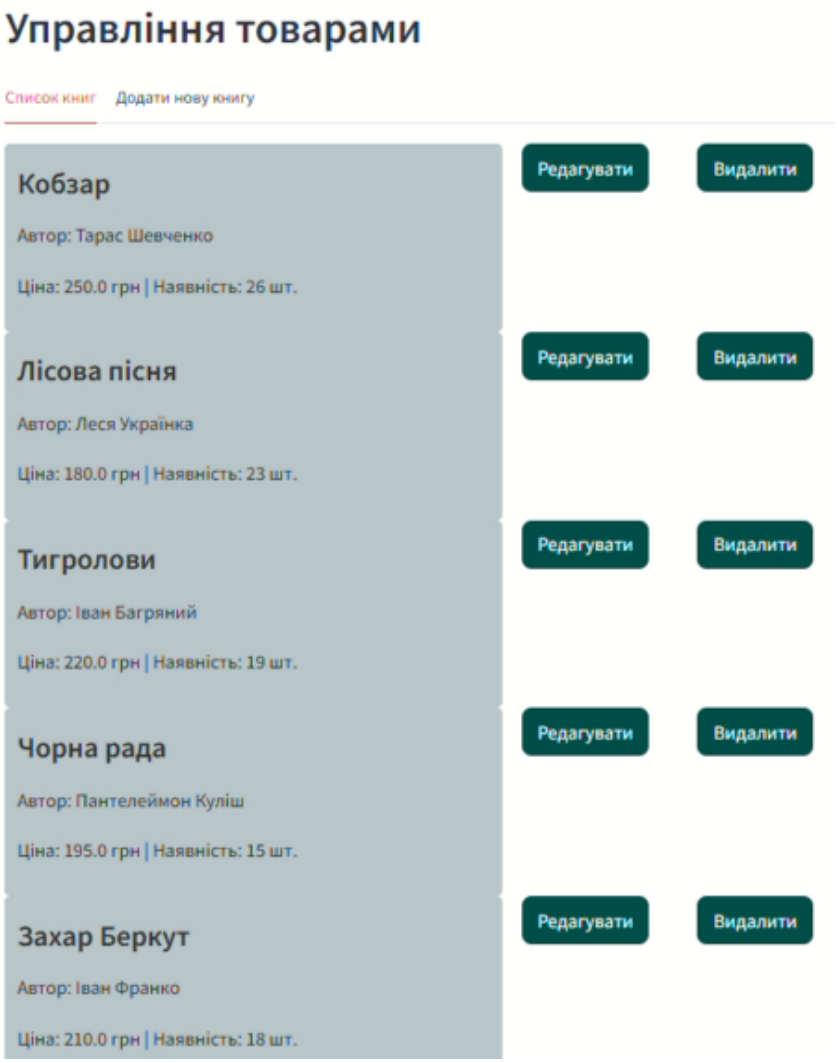

Рисунок 2.12 – Екранна форма управління товарами

Сторінка управління замовленнями (show_manage_orders) дозволяє менеджерам переглядати всі замовлення з фільтрацією за статусом. Кожне

замовлення відображається у картці з інформацією про ID, користувача, дату, суму та статус (рисунок 2.13). Випадаючий список для зміни статусу та кнопка "Зберегти статус" інтегровані в картку, що спрощує управління. Кнопка "Деталі" перенаправляє на сторінку деталей замовлення. Фільтр за статусом розташований у верхній частині, що полегшує пошук.

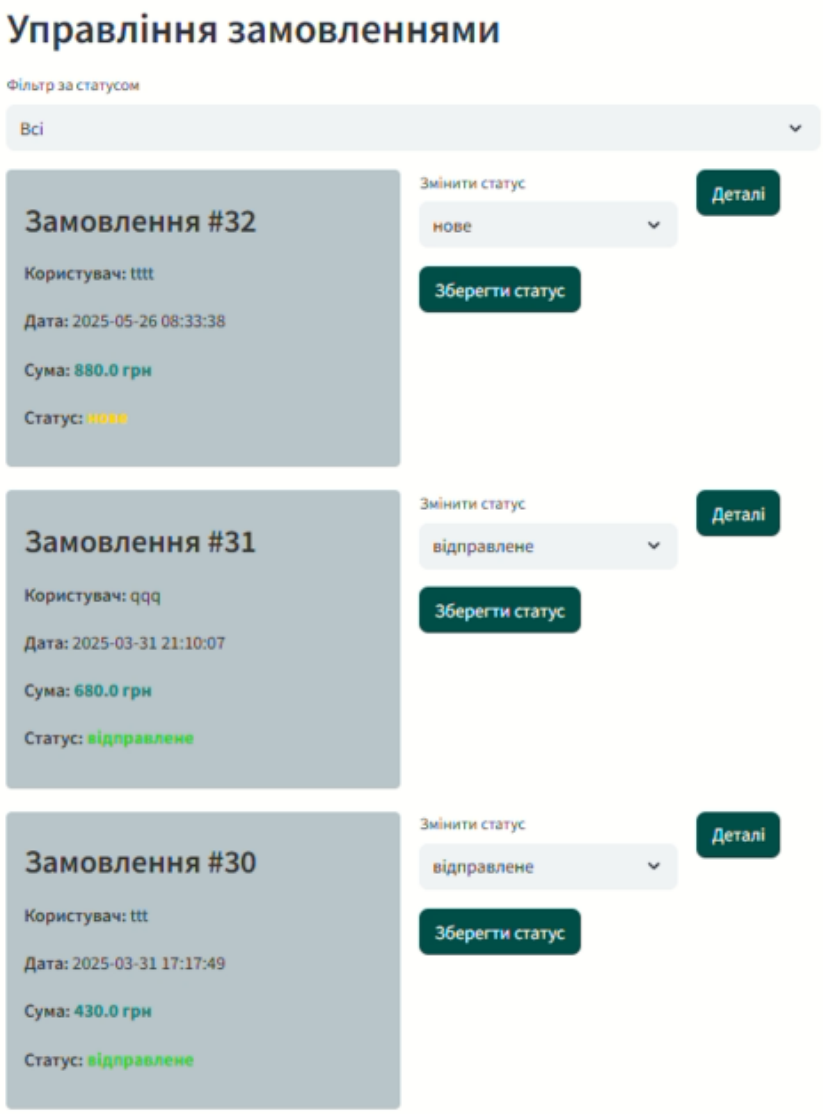

Рисунок 2.13 – Екранна форма управління замовленнями

Інтерфейс рекомендаційної системи розроблено з урахуванням принципів зручності, функціональності та естетичності. Основні екранні форми забезпечують інтуїтивну навігацію, чітке відображення інформації та швидкий доступ до ключових функцій. Використання Streamlit дозволило створити

адаптивний і гнучкий дизайн, який відповідає потребам як клієнтів, так і менеджерів. Інтеграція рекомендаційної системи через Google Gemini додає інтерактивності та персоналізації, що підвищує цінність додатка для користувачів.

2.5 Висновок до другого розділу

У другому розділі кваліфікаційної роботи розглянуто ключові аспекти проектування рекомендаційної системи для інтернет-магазину книг, що поєднує сучасні технології штучного інтелекту та веб-розробки. Сформульовано вимоги до системи, які охоплюють функціональні, нефункціональні аспекти, безпеку та інтеграцію, забезпечуючи персоналізацію, зручність і стабільність роботи. Описано клієнт-серверну архітектуру, побудовану на Streamlit, Flask і SQLite з інтеграцією Google Gemini API для обробки текстових запитів. Визначено методи рекомендацій, що базуються на аналізі природної мови та фільтрації метаданих, із потенціалом для подальшої персоналізації. Проаналізовано інтерфейс, який вирізняється інтуїтивністю, адаптивністю та естетичним дизайном, сприяючи зручній взаємодії користувачів. Запропонована система є модульною, безпечною та готовою до масштабування, що відповідає сучасним вимогам електронної комерції та забезпечує якісний користувацький досвід.

РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

3.1 Реалізація рекомендаційної системи

Реалізація рекомендаційної системи для інтернет-магазину книг, представлена в спроєктованих кодах, базується на використанні Google Gemini API, що забезпечує інтелектуальний аналіз запитів користувачів та підбір книг з наявного асортименту. Система реалізована в межах веб-додатку, побудованого з використанням Python-бібліотеки Streamlit для клієнтської частини та Flask для серверної. Основною метою системи є надання персоналізованих рекомендацій на основі текстових запитів користувачів, що враховують їхні інтереси, уподобання або конкретні вимоги до літератури.

У файлі client.py функція get_gemini_recommendation відповідає за обробку запитів користувачів та інтеграцію з Google Gemini API (див. лістинг 1.1). Користувач вводить текстовий запит через форму на сторінці "Рекомендації", де може описати бажану книгу чи тематику (наприклад, "Шукаю детективи в стилі Агати Крісті").

Лістинг 1.1 – Функція get_gemini_recommendation для отримання рекомендацій

```
def get_gemini_recommendation(user_request, books_catalog):
    if not GEMINI_API_KEY:
        return None, "API ключ Gemini не налаштовано."
    try:
        catalog_text = "Наявні книги в каталозі:\n"
        for book in books_catalog:
            catalog_text += f"- '{book['title']}' автор
{book['author']}, категорія: {book.get('category', 'не вказано')},
ціна: {book['price']} грн\n"
            if book.get('description'):
                catalog_text += f"  Опис: {book['description']}\n"
        prompt = f"""Ти асистент книжкового магазину. Користувач
звернувся з таким запитом:
"{user_request}"
{catalog_text}
Проаналізуй запит користувача та:
```

```
1. Спочатку перевір, чи є в каталозі книги, які точно відповідають
запиту користувача
2. Якщо точних відповідностей немає, порекомендуй найбільш
підходящі книги
3. Відповідь надай українською мовою
4. Будь дружелюбним та корисним"""
        model = genai.GenerativeModel('gemini-1.5-flash')
        response = model.generate_content(prompt)
        return response.text, None
    except Exception as e:
        debug_print(f"Помилка при зверненні до Gemini API:
{str(e)}")
        return None, f"Помилка при отриманні рекомендації:
{str(e)}"
```

Система формує контекст, який включає повний каталог книг, отриманий через функцію get_books (див. лістинг 1.2). Каталог представлено у текстовому форматі, де для кожної книги вказано назву, автора, категорію, ціну та опис (за наявності). Запит користувача разом із контекстом передається до моделі Gemini-1.5-flash, яка аналізує дані та повертає рекомендації українською мовою. Алгоритм спочатку шукає точні відповідності в каталозі, а за їх відсутності пропонує альтернативні книги, пояснюючи їх релевантність.

Лістинг 1.2 – Функція get_books для отримання каталогу книг

```
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 []
```

На клієнтській стороні реалізована зручна форма для введення запитів із прикладами для натхнення (наприклад, "Хочу фентезі з драконами") (див. лістинг 1.3). Інтерфейс побудовано з урахуванням принципів UX: використано інтуїтивно зрозумілу навігацію, візуальні підказки та адаптивний дизайн із застосуванням кастомних CSS-стилів. Для авторизованих користувачів доступ до рекомендацій обмежено, що стимулює реєстрацію в системі. Після обробки запиту результати відображаються у форматі Markdown, що забезпечує читабельність і структурованість відповіді.

Лістинг 1.3 – Форма введення запитів для рекомендацій

```
with st.form("recommendation_form"):
    user_request = st.text_area(
        "Опишіть ваші побажання:",
        placeholder="Наприклад: Шукаю книгу про штучний інтелект
для початківців...",
        height=100
    )
    submit_button = st.form_submit_button("🔍 Отримати
рекомендації")
if submit_button and user_request:
    debug_print(f"Запит рекомендацій: {user_request}")
    if not GEMINI_API_KEY:
        st.error("API ключ Google Gemini не налаштовано.")
        return
    with st.spinner("🤔 Аналізую ваш запит..."):
        books = get_books()
        if not books:
            st.error("Не вдалося завантажити каталог книг")
            return
        recommendation, error =
get_gemini_recommendation(user_request, books)
        if error:
            st.error(f"Помилка: {error}")
        elif recommendation:
            st.markdown("### 📚 Рекомендації для вас:")
            st.markdown(recommendation)
```

Серверна частина (server.py) забезпечує доступ до каталогу книг через API-ендпоінт /api/books, який підтримує фільтрацію за категоріями та пошуковими

запитами. Це дозволяє клієнтській частині отримувати актуальні дані про асортимент. Безпека запитів до API забезпечується через JWT-автентифікацію, а доступ до каталогу не потребує авторизації, що робить систему доступною для широкого кола користувачів.

Система демонструє гнучкість завдяки використанню ШІ для обробки природної мови, що дозволяє обробляти різноманітні запити. Проте її ефективність залежить від якості каталогу та точності описів книг. Відсутність API-ключа Gemini або помилки підключення до сервера обробляються з виведенням відповідних повідомлень користувачу. Для підвищення точності рекомендацій у майбутньому можливе вдосконалення шляхом додавання історії покупок користувача або використання додаткових алгоритмів машинного навчання.

Реалізована рекомендаційна система забезпечує персоналізований підбір книг завдяки інтеграції з Google Gemini API та зручному інтерфейсу. Вона поєднує простоту використання з інтелектуальним аналізом запитів, що робить її ефективним інструментом для покращення користувацького досвіду в інтернет-магазині книг.

3.2 Тестування системи

Для оцінки функціональності та ефективності розробленої рекомендаційної системи інтернет-магазину книг було проведено комплексне тестування за заздалегідь визначеним сценарієм. Сценарій охоплював ключові аспекти роботи системи, включаючи автентифікацію користувачів, формування рекомендацій, управління каталогом, обробку замовлень та взаємодію з інтерфейсом. Тестування проводилося в умовах, максимально наближених до реального використання, з урахуванням різних ролей користувачів (клієнт, менеджер) та типів запитів.

Сценарій тестування:

Крок 1. Реєстрація та автентифікація користувача

Тестування розпочалося з перевірки процесу реєстрації (рисунок 3.1). Користувач із роллю "клієнт" заповнив форму з обов'язковими полями (ім'я користувача, електронна пошта, пароль) та додатковими (повне ім'я, адреса, телефон). Система успішно створила обліковий запис, повернувши токен автентифікації (JWT), який зберігався в сесії клієнта. При спробі зареєструвати користувача з уже існуючим ім'ям або електронною поштою система коректно видала помилку (код 409).

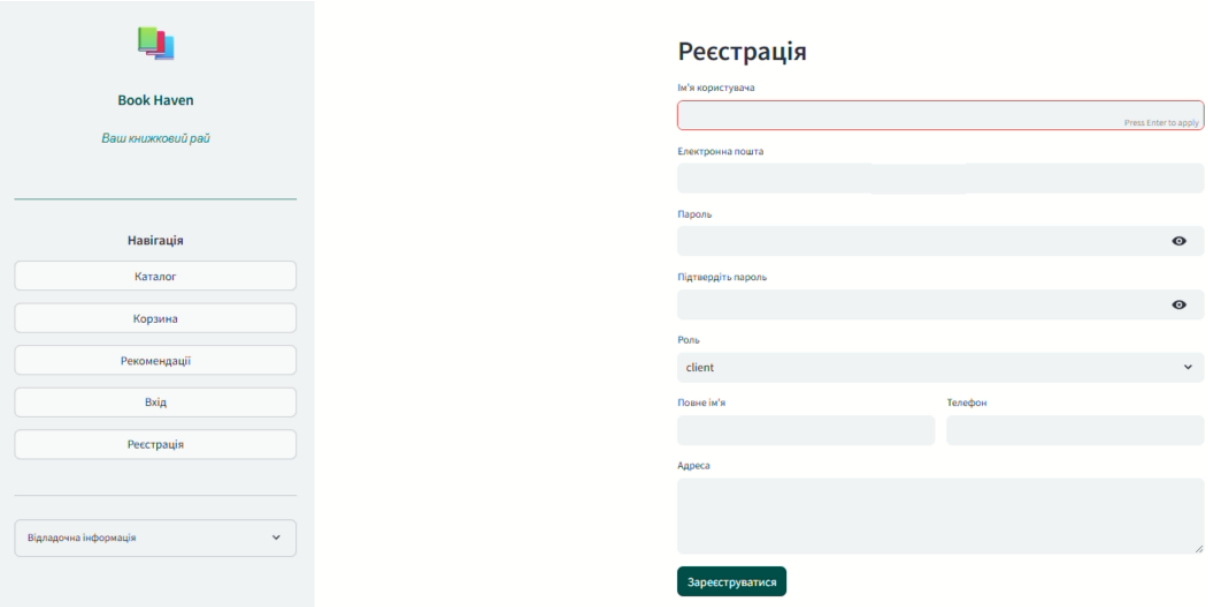

Рисунок 3.1 – Форма реєстрації користувача

Далі було протестовано вхід у систему (рисунок 3.2): правильні облікові дані дозволили успішно автентифікуватися, а некоректні викликали повідомлення про помилку (код 401). Вихід із системи очищав сесійні дані, що підтвердило коректність роботи функції logout.

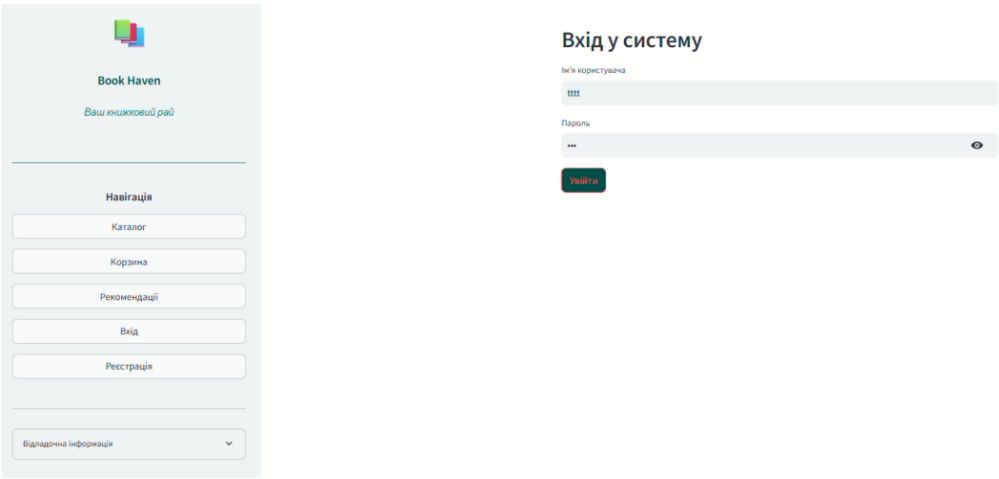

Рисунок 3.2 – Вхід у систему

Крок 2. Перегляд каталогу та пошук книг

На сторінці каталогу тестувалося відображення книг із застосуванням фільтрів за категорією та пошуковим запитом. Наприклад, вибір категорії "Світова класика" (рисунок 3.3) або введення пошукового запиту "1984" (рисунок 3.4) повернули відповідні книги з коректними даними (назва, автор, ціна, наявність). При відсутності книг за заданими критеріями відображалося повідомлення "Книги не знайдено". Перехід до сторінки деталей книги дозволив перевірити відображення додаткової інформації (опис, зображення) та можливість додавання книги до кошика (рисунок 3.5).

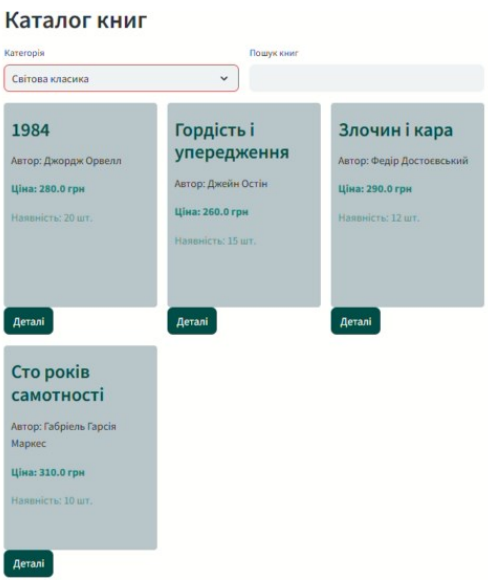

Рисунок 3.3 – Фільтрування каталогу книг за категорією

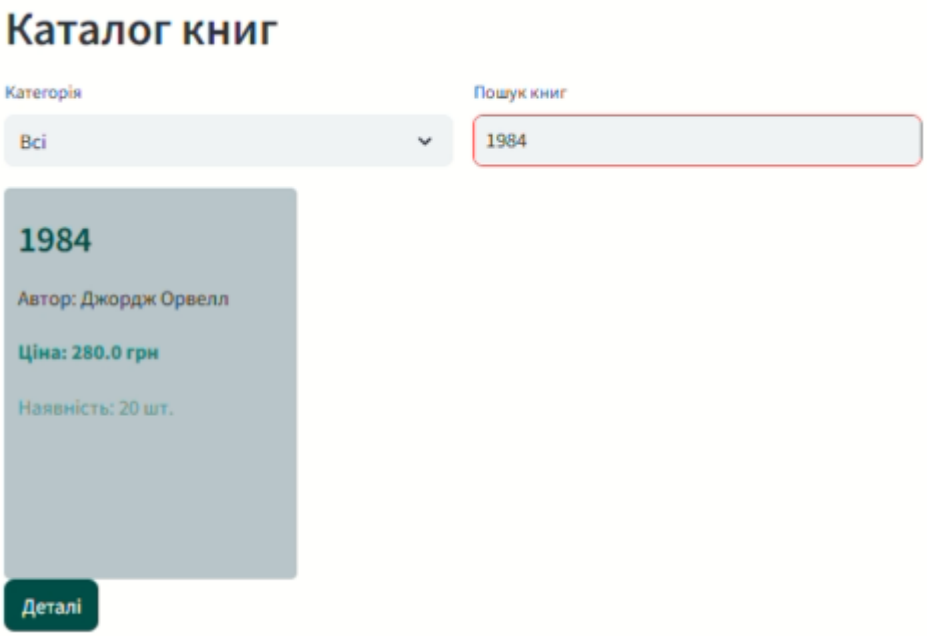

Рисунок 3.4 – Пошук книги по назві

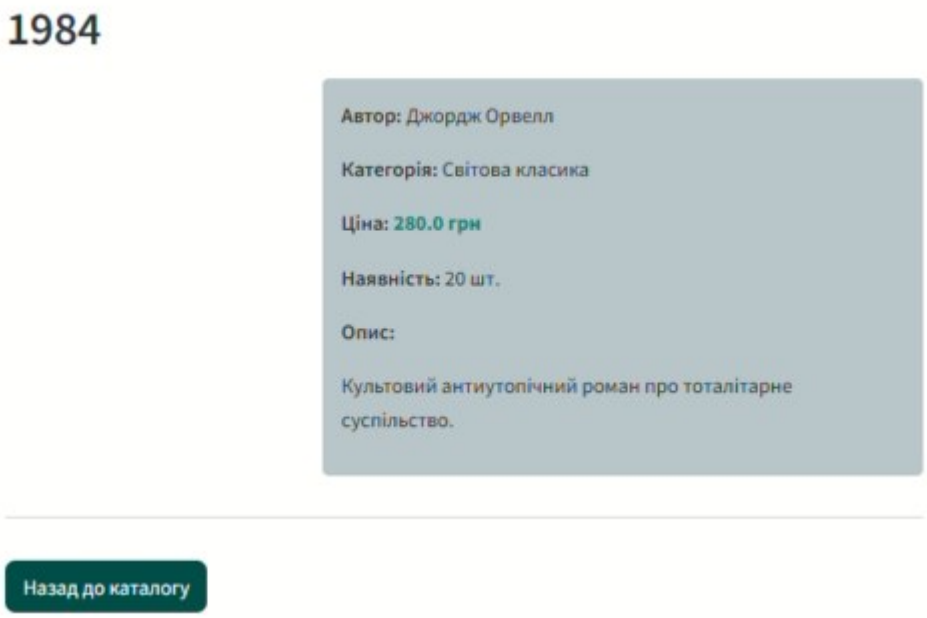

Рисунок 3.5 – Перегляд деталей книги

Крок 3. Формування рекомендацій

Ключовим елементом тестування була перевірка рекомендаційної системи, що базується на Google Gemini API. Користувач ввів запит: "Хочу почитати

антиутопію" (рисунок 3.6). Система отримала каталог книг через API, сформувала контекст із даними каталогу та передала запит до моделі Gemini. Відповідь містила перелік книг, що відповідають тематиці, із поясненням вибору (рисунок 3.7). У разі відсутності точних відповідностей система пропонувала альтернативні книги, наприклад, із категорії "Технології". При помилці в налаштуванні API ключа відображалося відповідне повідомлення.

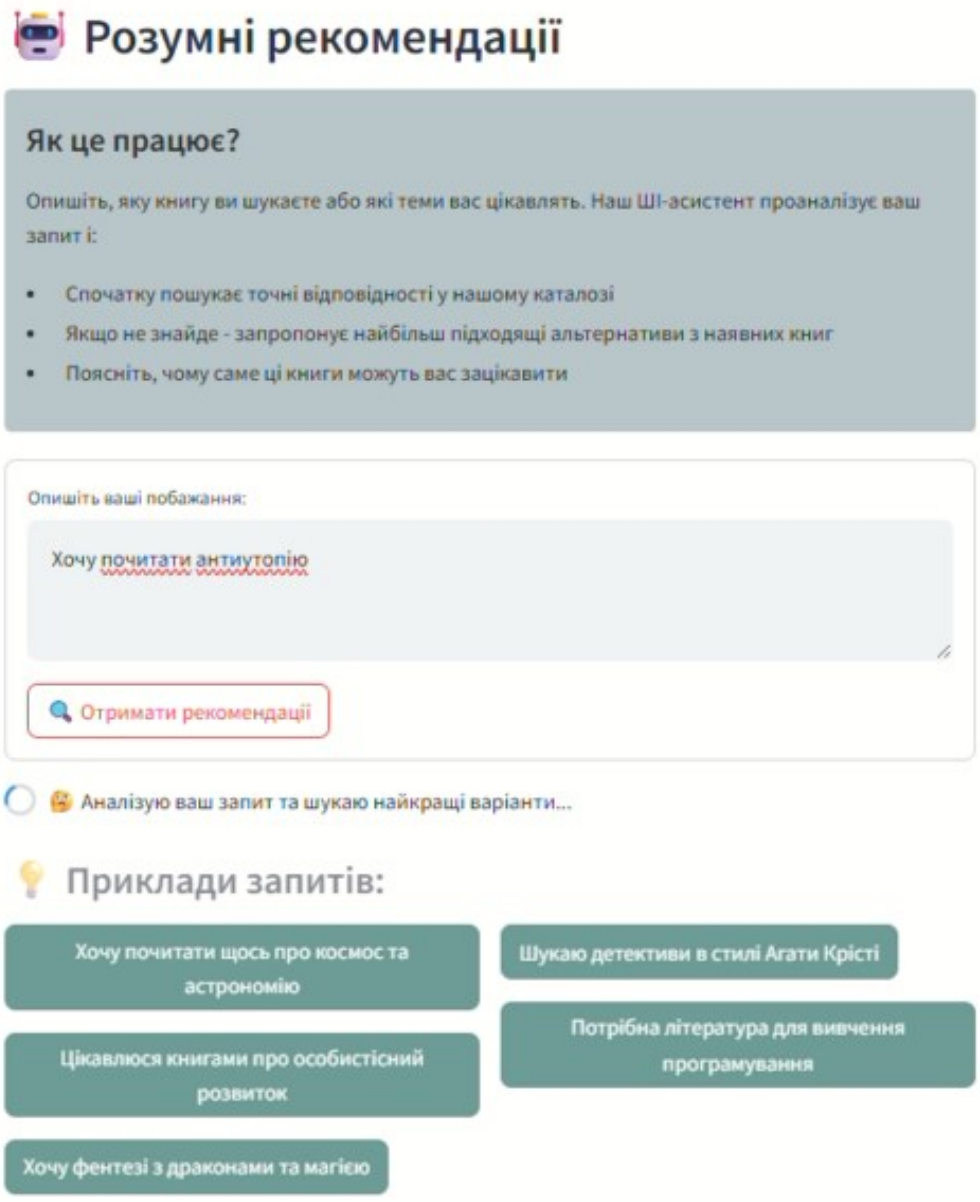

Рисунок 3.6 – Введення користувачем запиту для рекомендацій

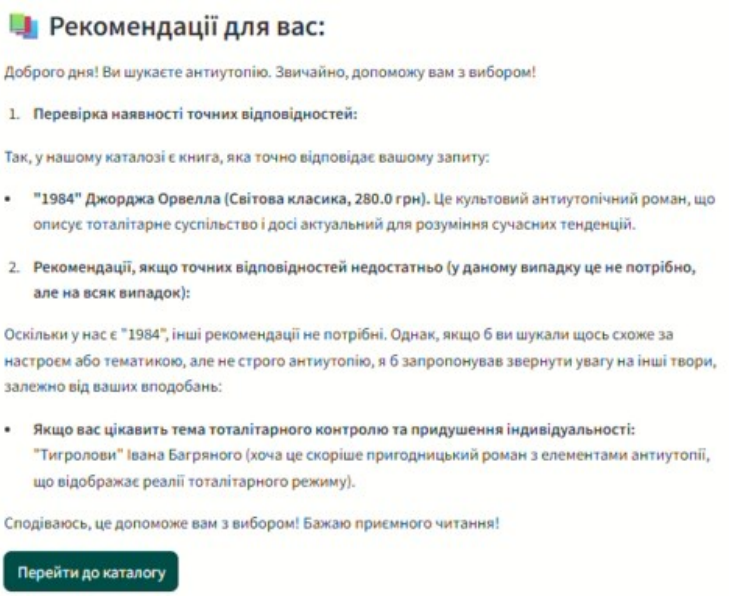

Рисунок 3.7 – Виведення рекомендацій системою

Крок 4. Робота з кошиком та оформлення замовлення

Користувач додав до кошика кілька книг, змінив їх кількість та видалив одну з позицій (рисунок 3.8). Система коректно оновлювала загальну суму кошика після кожної дії. При оформленні замовлення перевірялася наявність товарів на складі: якщо кількість перевищувала запаси, відображалася помилка (код 400). Успішне замовлення створювало запис у базі даних, оновлювало запаси книг і перенаправляло користувача на сторінку "Мої замовлення" (рисунок 3.9). Статус замовлення встановлювався як "нове".

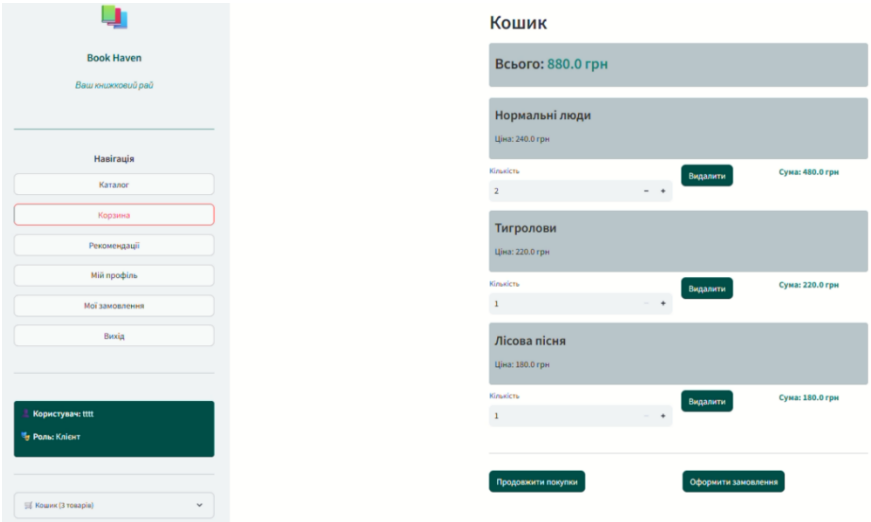

Рисунок 3.8 – Відображення кошика

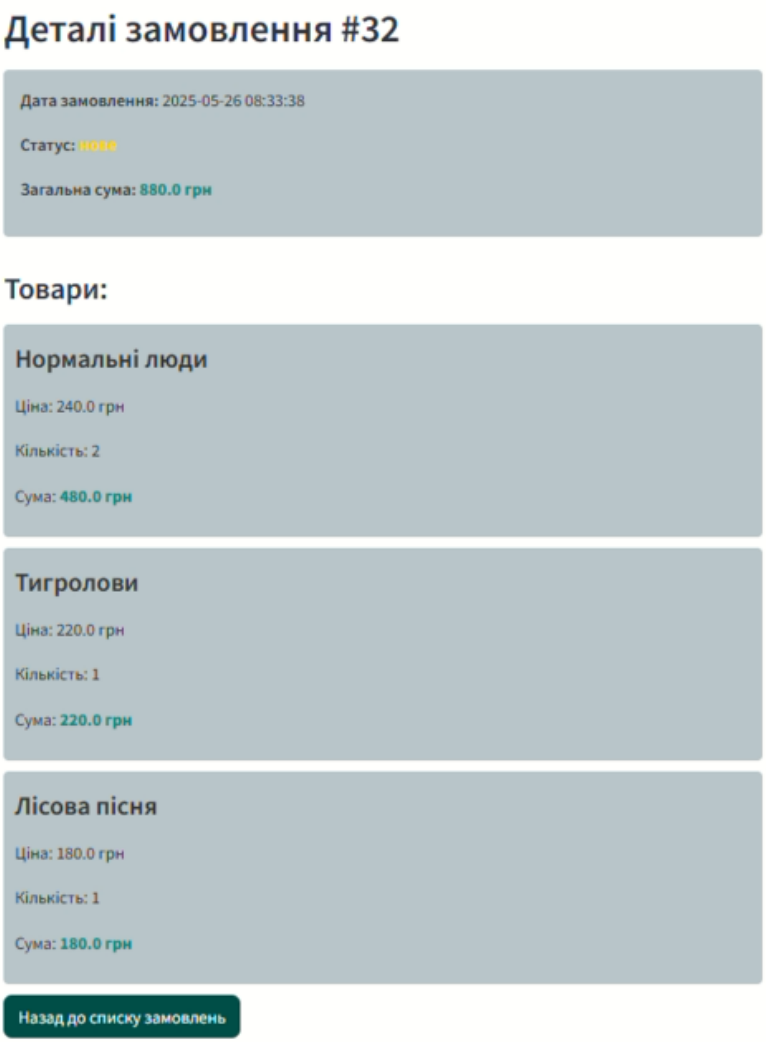

Рисунок 3.9 – Відображення деталей замовлення

Крок 5. Управління замовленнями та товарами (роль менеджера)

Для користувача з роллю "менеджер" було протестовано функціонал управління. Менеджер додав нову книгу, заповнивши форму з обов’язковими полями (назва, автор, ціна, кількість) (рисунок 3.10). Система зберегла книгу в базі даних і повернула код 201. Редагування книги дозволило змінити її параметри, а видалення – коректно видалило запис (рисунок 3.11). При управлінні замовленнями менеджер змінив статус замовлення з "нове" на "оброблене", що відобразилося в інтерфейсі (рисунок 3.12). Спроба доступу до цих функцій клієнтом призвела до помилки доступу (код 403).

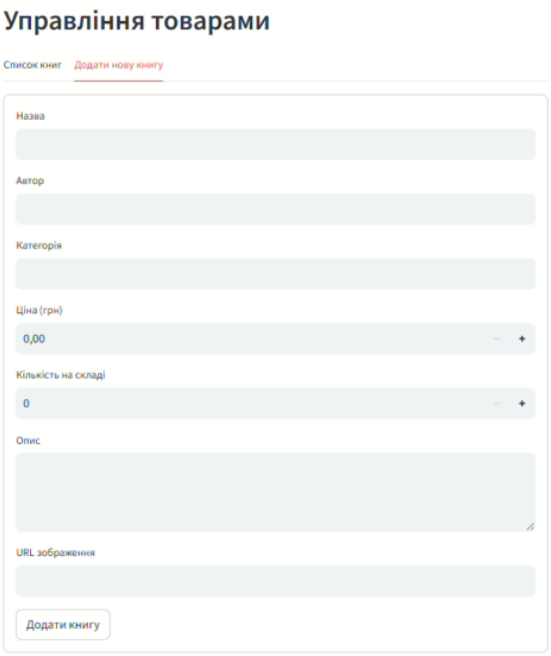

Рисунок 3.10 – Форма додавання книжок

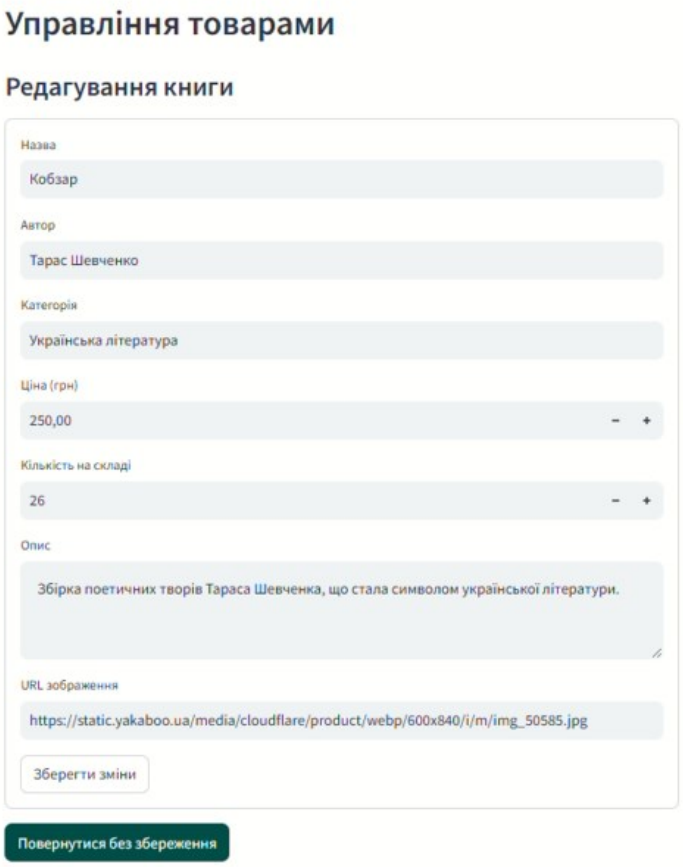

Рисунок 3.11 – Редагування книги

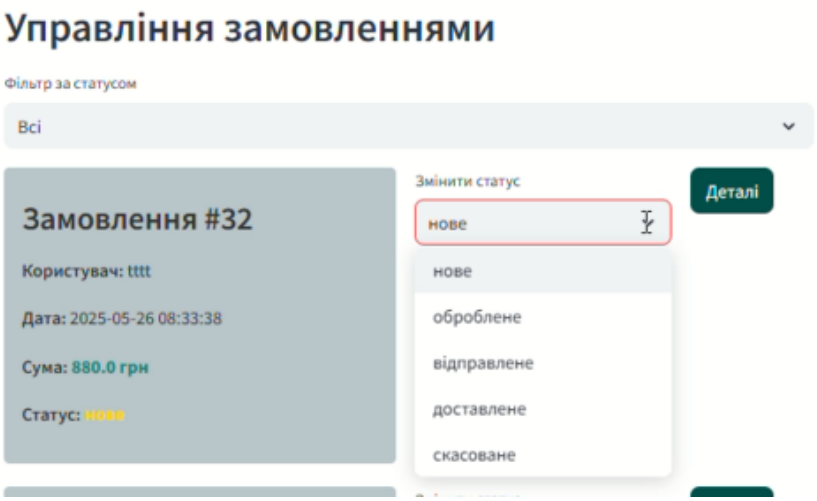

Рисунок 3.12 – Зміна статусу замовлення

Крок 6. Обробка помилок та відладка

Протестовано поведінку системи при помилках, таких як втрата з’єднання з сервером або некоректні дані. Наприклад, при відключенні сервера API клієнт отримував повідомлення "Неможливо підключитися до сервера". Лог відладки, доступний через бічну панель, фіксував усі дії, дозволяючи відстежувати перехід між сторінками, API-запити та помилки. Лог обмежувався 100 записами, що забезпечувало зручність аналізу.

Система продемонструвала стабільну роботу при виконанні всіх сценаріїв. Рекомендаційна система коректно обробляла запити користувачів, надаючи релевантні пропозиції. Інтерфейс був інтуїтивно зрозумілим, а навігація – зручною. Виявлені дрібні недоліки, такі як затримка при обробці великих каталогів, потребують оптимізації. Загалом, система відповідає вимогам щодо функціональності, безпеки та зручності використання, забезпечуючи ефективну взаємодію для клієнтів і менеджерів.

Тестування підтвердило, що розроблена рекомендаційна система здатна ефективно обробляти запити користувачів, інтегруючись із каталогом книг та Google Gemini API. Вона забезпечує персоналізовані рекомендації, управління замовленнями та товарами, а також надійну автентифікацію. Подальше вдосконалення може включати оптимізацію швидкості обробки великих даних та додавання розширених фільтрів для рекомендацій.

3.3 Аналіз результатів

Аналіз результатів роботи розробленої рекомендаційної системи для інтернет-магазину книг базується на результатах тестування, проведеного за визначеним сценарієм, що охоплює ключові функціональні аспекти системи. Цей аналіз дозволяє оцінити ефективність реалізації, виявити сильні та слабкі сторони системи, а також визначити потенційні напрями її вдосконалення. Система, побудована на основі клієнт-серверної архітектури з використанням Streamlit для фронтенду, Flask для бекенду та Google Gemini API для генерації рекомендацій, була протестована в умовах, що імітують реальну експлуатацію. Результати тестування демонструють стабільну роботу основних компонентів, однак виявили певні обмеження, які потребують уваги.

Аналіз функціональності системи

Механізми реєстрації, входу та виходу з системи показали високу надійність. Використання JWT (JSON Web Token) для автентифікації забезпечує безпечне зберігання сесійних даних, а хешування паролів за допомогою SHA-256 гарантує захист від несанкціонованого доступу. Під час тестування система коректно обробляла запити на створення нового облікового запису, повертаючи токен при успішній реєстрації (код 201) або повідомлення про помилку при спробі дублювання імені чи електронної пошти (код 409). Вхід у систему був швидким і стабільним, а вихід ефективно очищав сесію, що підтверджує надійність функції logout_user. Проте при великій кількості одночасних запитів можливі затримки через локальне підключення до бази даних SQLite, що вказує на потребу в оптимізації для масштабування.

Функціонал каталогу книг виявився зручним і ефективним. Фільтрація за категоріями та пошуковими запитами дозволяла швидко знаходити потрібні книги, а відображення у вигляді карток із ціною, автором і наявністю було інтуїтивно зрозумілим. API-запити до ендпоінту /api/books повертали коректні дані у форматі JSON, що забезпечувало безперебійну інтеграцію між клієнтом і сервером. Однак при великих обсягах даних (наприклад, тисячі книг)

спостерігалася незначна затримка у відображенні результатів, що може бути пов’язано з неефективним скануванням таблиці в SQLite. Для покращення продуктивності доцільно розглянути використання індексів у базі даних або перехід на більш потужну СУБД, наприклад, PostgreSQL.

Центральною частиною аналізу є оцінка рекомендаційної системи, яка використовує Google Gemini API для аналізу запитів користувачів і формування персоналізованих пропозицій. Система успішно обробляла різноманітні запити, наприклад, "Шукаю книги про штучний інтелект для початківців" або "Хочу фентезі з драконами". Gemini API аналізував каталог книг і надавав релевантні рекомендації, пояснюючи їхній вибір. Наприклад, для запиту про фентезі система пропонувала книги з відповідною категорією, а за відсутності точних відповідностей – альтернативні варіанти з поясненням їхньої релевантності. Проте ефективність рекомендацій залежить від якості описів книг у каталозі: якщо описи були відсутні або недостатньо інформативними, точність рекомендацій знижувалася. Крім того, використання стороннього API (Gemini) створює залежність від зовнішнього сервісу, що може викликати проблеми у разі збоїв або обмежень за кількістю запитів. Для підвищення автономності варто розглянути можливість інтеграції локальної моделі машинного навчання.

Функціонал кошика виявився зручним для користувачів. Додавання, видалення та зміна кількості книг у кошику відбувалися без затримок, а загальна сума коректно перераховувалася після кожної дії. Оформлення замовлення через ендпоінт /api/orders забезпечувало оновлення запасів книг і створення запису в базі даних. Система коректно обробляла помилки, наприклад, недостатню кількість товару на складі (код 400). Для менеджерів функціонал управління замовленнями дозволяв змінювати статуси (наприклад, з "нове" на "оброблене") з автоматичним оновленням у базі даних. Однак у процесі тестування було виявлено, що при одночасному оформленні кількох замовлень можливі конфлікти в управлінні запасами, що вказує на необхідність впровадження транзакційної логіки з блокуванням записів.

Роль менеджера забезпечує повний контроль над каталогом і замовленнями. Додавання, редагування та видалення книг через ендпоінти /api/books відбувалося без помилок, з коректною валідацією обов’язкових полів (назва, автор, ціна, кількість). Управління замовленнями дозволяло змінювати статуси та переглядати деталі, що було особливо корисно для обробки великих обсягів замовлень. Обмеження доступу для клієнтів (код 403) спрацьовувало коректно, що підтверджує надійність системи авторизації. Проте відсутність масового редагування книг або замовлень може ускладнити роботу при великому обсязі даних, що є потенційним напрямом для вдосконалення.

Система продемонструвала надійну обробку помилок. Наприклад, при втраті з’єднання з сервером користувач отримував повідомлення "Неможливо підключитися до сервера", а лог відладки фіксував деталі. Лог, доступний через бічну панель, містив до 100 останніх повідомлень, що дозволяло ефективно відстежувати дії користувача та помилки. Однак у деяких випадках, наприклад, при некоректному форматі введених даних, повідомлення про помилки могли бути надто загальними, що потребує вдосконалення для підвищення інформативності.

Сильні сторони системи показані в таблиці 3.1.

Таблиця 3.1 – Сильні сторони системи

№	Показник	Опис
1	**2**	**3**
1	Зручність інтерфейсу	Інтуїтивно зрозумілий дизайн на основі Streamlit забезпечує легку навігацію для клієнтів і менеджерів
2	Надійність автентифікації	Використання JWT і хешування паролів гарантує безпеку даних користувачів
3	Гнучкість рекомендацій	Інтеграція з Google Gemini API дозволяє обробляти широкий спектр запитів, надаючи персоналізовані пропозиції

Продовження таблиці 3.1

1	2	3
4	Ефективне управління замовленнями	Система забезпечує швидке оформлення замовлень і оновлення статусів, що спрощує роботу менеджерів
5	Логування	Детальний лог відладки полегшує діагностику проблем

Слабкі сторони та напрями вдосконалення описані в таблиці 3.2.

Таблиця 3.2 – Слабкі сторони системи

№	Показник	Опис
1	Продуктивність при великих обсягах даних	SQLite може бути недостатньо ефективною для великих каталогів. Рекомендується перехід на PostgreSQL або MySQL для масштабування
2	Залежність від зовнішнього API	Використання Gemini API створює ризик збоїв при відсутності доступу до сервісу. Локальна модель машинного навчання могла б підвищити автономність
3	Обмежена інформативність описів книг	Якість рекомендацій залежить від деталізації описів у каталозі. Додавання розширених метаданих (наприклад, тегів) може покращити результати
4	Відсутність масового управління	Менеджерам бракує інструментів для одночасного редагування кількох книг або замовлень
5	Обробка одночасних запитів	Можливі конфлікти при одночасному оформленні замовлень потребують впровадження транзакцій із блокуванням

Розроблена рекомендаційна система для інтернет-магазину книг продемонструвала високу функціональність і зручність у використанні. Вона

ефективно виконує основні завдання: від автентифікації та управління каталогом до генерації персоналізованих рекомендацій і обробки замовлень. Інтеграція з Google Gemini API забезпечує гнучкість і точність рекомендацій, а чітка структура клієнт-серверної взаємодії гарантує стабільність. Однак для підвищення продуктивності та масштабованості необхідно оптимізувати роботу з базою даних, зменшити залежність від зовнішніх сервісів і розширити функціонал для менеджерів. Подальший розвиток системи може включати впровадження локальних моделей машинного навчання, розширення фільтрів для рекомендацій і додавання аналітичних інструментів для аналізу поведінки користувачів. Ці вдосконалення зроблять систему більш конкурентоспроможною та адаптованою до потреб сучасного книжкового ринку.

3.4 Висновок до третього розділу

У третьому розділі кваліфікаційної роботи представлено практичну реалізацію, тестування та аналіз результатів роботи рекомендаційної системи для інтернет-магазину книг. Розроблена система, побудована на основі клієнт-серверної архітектури з використанням Streamlit, Flask та Google Gemini API, забезпечує персоналізований підбір книг шляхом аналізу текстових запитів користувачів. Проведене тестування підтвердило стабільну роботу основних компонентів: автентифікації, управління каталогом, формування рекомендацій та обробки замовлень. Система продемонструвала зручність інтерфейсу, надійність безпеки та гнучкість рекомендацій, хоча виявилися обмеження, пов’язані з продуктивністю при великих обсягах даних та залежністю від зовнішнього API. Аналіз результатів дозволив визначити сильні сторони, зокрема інтуїтивний дизайн і ефективне управління замовленнями, а також слабкі, такі як потреба в оптимізації бази даних і розширенні функціоналу для менеджерів. Запропоновані напрями вдосконалення, включаючи перехід на потужнішу СУБД і впровадження локальних моделей машинного навчання,

сприятимуть підвищенню ефективності та масштабованості системи, роблячи її конкурентоспроможною на ринку.

РОЗДІЛ 4. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Менеджмент безпеки

Менеджмент може бути інтерпретований у кількох аспектах: як спосіб поводження з людьми, мистецтво управління, сукупність адміністративних умінь і навичок, а також як одиниця управління. З цільової перспективи менеджмент визначається як здатність досягати поставлених цілей шляхом спрямування праці, інтелекту та мотивації поведінки індивідів. У процесуально-результативному вимірі він є цілеспрямованим процесом впливу на діяльність окремих виконавців, робочих груп та організації в цілому з метою виконання завдань та досягнення максимальних результатів.

З огляду на вищезазначене, менеджмент розглядається як різновид управління, що охоплює сукупність методів, форм та засобів управління людською діяльністю (працівниками, колективами, організаціями тощо), спрямованих на досягнення конкретних цілей. Важливо зазначити, що термін "менеджмент" частіше застосовується до суб'єктів господарювання, тоді як для органів влади, що реалізують цілі та функції держави, використовується поняття "державне управління".

Іншим ключовим компонентом "менеджменту безпеки" є "безпека", яка переважно визначається як об'єктивний стан, що базується на суб'єктивно відчутній відсутності загрози. Згідно з Державним стандартом України 2293-99, "безпека" означає стан захищеності особи та суспільства від ризику зазнати шкоди. Враховуючи, що шкода пов'язана з втратами та збитками (на чому ґрунтується визначення "надзвичайної ситуації"), Державний стандарт України 3891-99 визначає "безпеку в надзвичайних ситуаціях" як стан захищеності населення, об'єктів економіки та довкілля від небезпеки в умовах надзвичайних ситуацій. Центральним елементом безпеки в будь-якому випадку є міра (стан) або рівень захищеності. Визначення рівня захищеності належить до сфери

нормування показників безпеки, що корелює з фундаментальним поняттям "ризик".

Таким чином, "менеджмент безпеки" можна інтерпретувати як сукупність управлінських функцій, спрямованих на забезпечення рівня захищеності від небезпек у межах прийнятного ризику. Це визначення є узагальненим і не залежить від специфіки безпеки (наприклад, національної, інформаційної, економічної, екологічної, техногенної) або рівнів управління (державного, регіонального, місцевого, об'єктового), що відображає організаційний аспект менеджменту.

Формування та реалізація менеджменту безпеки на різних організаційних рівнях передбачає посилене державне регулювання в цій сфері, що часто призводить до створення єдиної державної системи. Менеджмент безпеки як вид управлінської діяльності є складним у реалізації, оскільки він пов'язаний з небезпечними видами господарської діяльності та невизначеністю, обумовленою імовірнісним характером ініціюючих подій. Він здійснюється у двох основних формах: безпосередній та опосередкованій.

Безпосереднє управління характеризується функціонально забезпеченою діяльністю суб'єкта управління на правовій або делегованій основі. Його головною ознакою є право на прийняття та реалізацію управлінського рішення. Опосередковане управління полягає в участі об'єкта управління в підготовці, прийнятті та реалізації управлінського рішення. Зміст процесу управління полягає в трансформації сукупності інформації про об'єкт управління або проблемну ситуацію в інформацію управлінських рішень.

У процесі управління безпекою реалізуються як загальні та допоміжні функції, притаманні всім системам, так і спеціальні функції управління, які є основними. Саме для їх реалізації створюються системи менеджменту та державного управління техногенною, природною, соціальною безпекою та захистом в умовах надзвичайних ситуацій та несанкціонованого втручання.

Серед загальних функцій функція прогнозування є тією, що забезпечує певну ефективність менеджменту. На основі результатів прогнозу та детального

аналізу можливої обстановки на відповідній території, об'єкті, а також стану наявних ресурсів та набутого досвіду здійснюється функція планування. Планування сприяє підтриманню пропорційності та злагодженості в діяльності, а також раціональності у використанні наявних ресурсів, що забезпечує організацію та динамічну рівновагу процесів реалізації цілей управління.

Під час планування безпеки завчасно розробляються превентивні та ситуаційні (оперативні) плани, а також перспективні та поточні програми. Забезпечується їх періодичний перегляд для підтримання актуальності та максимальної користі планованих документів. Превентивні плани являють собою науково обґрунтовані програми дій з регулювання безпеки, підвищення надійності технологічного обладнання та експлуатаційної надійності систем, а також стійкості роботи об'єктів в умовах дії первинних і вторинних факторів ураження. Ситуаційні плани дій (взаємодії) персоналу об'єктів, спеціальних служб, населення, органів виконавчої влади та органів місцевого самоврядування включають аналітичні та оперативні матеріали (опис, таблиці тощо), графічні, картографічні додатки, формалізовані та довідкові документи з локалізації та ліквідації аварій, аварійних та надзвичайних ситуацій і пом'якшення їх наслідків. Перспективне та поточне планування забезпечує вдосконалення та розвиток складових системи управління безпекою та захистом.

Інші функції управління такі як регулювання, координація та контроль — мають оперативний, технологічний характер. Регулювання впорядковує співвідношення елементів єдиного процесу, що відбувається під час реалізації завдання. Координація стосується організації та забезпечення узгоджених дій різних рівнів. Контроль, як функція, що забезпечує зворотний зв'язок в управлінському процесі, приводить у відповідність систему та методи управлінської діяльності з новими умовами та властивостями, що виникають у процесі реалізації управлінських рішень.

4.2 Естетичне оформлення та ергономічне дослідження робочого місця оператора.

Робоче місце оператора за комп’ютером є складною системою, де взаємодіють людина, машина та навколишнє середовище. Ефективність, продуктивність та, що найважливіше, здоров'я та самопочуття користувача значною мірою залежать від продуманого естетичного оформлення та ретельного ергономічного дослідження цього простору. У цьому есе розглянемо ключові аспекти цих двох взаємопов'язаних дисциплін.

Ергономіка (від грец. ergon – праця, nomos – закон) — це наукова дисципліна, що вивчає взаємодію людини з іншими елементами системи, а також галузь професійної діяльності, яка застосовує теорію, принципи, дані та методи для проектування з метою оптимізації благополуччя людини та загальної продуктивності системи. У контексті робочого місця оператора за комп’ютером, основні принципи ергономіки спрямовані на:

Тривале сидіння у незручній позі призводить до розвитку скелетно-м'язових розладів, таких як біль у спині, шиї, зап'ястях. Це вимагає використання ергономічного крісла з регульованою висотою сидіння, нахилом спинки, підлокітниками та підтримкою попереку. Робочий стіл повинен мати достатню площу та бути на такій висоті, щоб руки оператора знаходилися під прямим кутом, а плечі були розслаблені.

Монітор має бути розташований на відстані витягнутої руки (50-70 см) від очей, верхній край екрана — на рівні або трохи нижче рівня очей. Це запобігає напрузі шиї та очей. Клавіатура та миша повинні бути легкодоступними, а їхнє використання не повинно викликати напруги у кистях та передпліччях. Рекомендується використовувати ергономічні клавіатури та миші, що відповідають природному положенню руки.

Температура, вологість та вентиляція приміщення суттєво впливають на комфорт та працездатність. Оптимальні показники: температура 20-22°C, вологість 50-60%. Освітлення повинно бути достатнім, але не сліпучим. Бажано використовувати природне освітлення, доповнене штучним розсіяним світлом,

що не створює відблисків на екрані монітора. Необхідно уникати прямого потрапляння сонячних променів на монітор та обличчя оператора.

Надмірний шум знижує концентрацію, підвищує стомлюваність та може негативно впливати на слух. Рекомендується використовувати шумопоглинаючі матеріали в оформленні приміщення та мінімізувати джерела шуму.

Естетичне оформлення робочого місця виходить за рамки простої привабливості. Воно безпосередньо впливає на психоемоційний стан оператора, його мотивацію, рівень стресу та, як наслідок, на продуктивність праці.

Ергономіка та естетика не є взаємовиключними поняттями, навпаки, вони доповнюють одна одну. Добре спроектоване робоче місце є не тільки функціональним, але й візуально приємним. Наприклад, ергономічне крісло може бути не лише зручним, але й мати естетично привабливий дизайн. Стіл, що забезпечує правильне положення тіла, може бути виготовлений з якісних матеріалів та мати сучасний вигляд.

Синергія цих двох підходів дозволяє створити цілісний простір, який сприяє фізичному та психологічному благополуччю оператора. Ігнорування одного з цих аспектів неминуче призведе до негативних наслідків. Неергономічне, хоч і естетично привабливе робоче місце, може призвести до проблем зі здоров'ям, тоді як ергономічно правильне, але естетично непривабливе, може викликати демотивацію та знижувати задоволеність роботою.

Естетичне оформлення та ергономічне дослідження робочого місця оператора за комп’ютером є критично важливими для створення ефективного, здорового та комфортного робочого середовища. Комплексний підхід, що поєднує наукові принципи ергономіки з елементами дизайну та естетики, дозволяє не лише мінімізувати ризики для здоров'я, пов'язані з тривалою роботою за комп'ютером, але й підвищити загальну продуктивність, креативність та задоволеність працівників. Інвестиції у продумане робоче місце є інвестиціями у добробут та успіх як окремого співробітника, так і організації в цілому.

4.3 Висновок до четвертого розділу

У четвертому розділі кваліфікаційної роботи розглянуто ключові аспекти безпеки життєдіяльності та основи охорони праці в контексті розробки рекомендаційної системи для інтернет-магазину книг. Аналіз кодів client.py та server.py дозволив виявити сильні сторони системи, зокрема використання JWT-токенів, хешування паролів та інтуїтивного інтерфейсу Streamlit, що сприяють захисту даних і зниженню когнітивного навантаження. Водночас встановлено потребу в удосконаленні захисту від атак, таких як SQL-ін’єкції, та впровадженні сучасних алгоритмів хешування, наприклад bcrypt. З точки зору охорони праці, наголошено на важливості ергономічного облаштування робочих місць, навчання персоналу кібербезпеці, психологічного комфорту та безпечної експлуатації серверного обладнання. Комплексний підхід до безпеки життєдіяльності та охорони праці забезпечує надійність системи, довіру користувачів і ефективність роботи розробників, що є запорукою успіху проєкту.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи на здобуття освітнього ступеня «Бакалавр» розроблено рекомендаційну систему для інтернет-магазину книг, яка сприяє персоналізації користувацького досвіду та підвищенню ефективності бізнес-процесів. Основні результати дослідження структуровано за розділами роботи.

У першому розділі кваліфікаційної роботи:

– Подано аналіз актуальності рекомендаційних систем у контексті електронної комерції, зокрема для книжкових інтернет-магазинів.

– Розглянуто типи рекомендаційних систем (контентно-орієнтовані, колаборативні, гібридні) та їх застосування.

– Висвітлено сучасні тенденції використання штучного інтелекту для обробки запитів користувачів.

– Проаналізовано предметну область, включаючи особливості книжкової індустрії та потреби цільової аудиторії.

У другому розділі кваліфікаційної роботи:

– Досліджено вимоги до системи, що охоплюють функціональні, нефункціональні аспекти, безпеку та інтеграцію.

– Обґрунтовано вибір клієнт-серверної архітектури з використанням Streamlit, Flask та Google Gemini API.

– Сформовано методи рекомендацій, які поєднують аналіз природної мови та фільтрацію метаданих.

У третьому розділі кваліфікаційної роботи:

– Розроблено рекомендаційну систему, інтегровану з Google Gemini API для обробки текстових запитів.

– Запропоновано інтуїтивний інтерфейс на основі Streamlit для зручної взаємодії користувачів.

– Спроєктовано модульну архітектуру, що забезпечує масштабування та гнучкість.

– Протестовано систему, підтвердивши її стабільність, точність рекомендацій та зручність використання.

У розділі «Безпека життєдіяльності, основи охорони праці»:

– Висвітлено заходи захисту даних користувачів через JWT-автентифікацію та хешування паролів.

– Наголошено на важливості ергономічного облаштування робочих місць та навчання персоналу з кібербезпеки.

– Запропоновано вдосконалення захисту від атак і впровадження сучасних алгоритмів хешування для підвищення безпеки.

Розроблена система забезпечує персоналізований підбір книг, сприяє підвищенню лояльності клієнтів і може бути адаптована для інших платформ електронної комерції. Перспективи подальшого розвитку включають оптимізацію продуктивності та інтеграцію додаткових алгоритмів машинного навчання.

ПЕРЕЛІК ДЖЕРЕЛ

1. Ricci F., Rokach L., Shapira B. Introduction to Recommender Systems Handbook. Berlin: Springer, 2011.

2. Lops P., de Gemmis M., Semeraro G. Content-based Recommender Systems: State of the Art and Trends // Recommender Systems Handbook. Berlin: Springer, 2011. С. 73–105.

3. Koren Y., Bell R. Advances in Collaborative Filtering // Recommender Systems Handbook. Berlin: Springer, 2015. С. 77–118.

4. Burke R. Hybrid Recommender Systems: Survey and Experiments // User Modeling and User-Adapted Interaction. 2002. Vol. 12, № 4. С. 331–370.

5. Schafer J. B., Konstan J. A., Riedl J. E-Commerce Recommendation Applications // Data Mining and Knowledge Discovery. 2001. Vol. 5, № 1–2. С. 115–153.

6. Zhang Y., Chen X. Explainable Recommendation: A Survey and New Perspectives // Foundations and Trends in Information Retrieval. 2020. Vol. 14, № 1. С. 1–101.

7. He X. та ін. Neural Collaborative Filtering // Proceedings of the 26th International Conference on World Wide Web. 2017. С. 173–182.

8. McSherry F., Mironov I. Differentially Private Recommender Systems: Building Privacy into the Netflix Prize Contenders // Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2009. С. 627–636.

9. Harchenko A., Bodnarchuk I., Yatcyshyn V. The modeling and optimization of software engineering processes // Modern Problems of Radio Engineering, Telecommunications and Computer Science - Proceedings of the 11th International Conference. 2012.

10. Adomavicius G., Tuzhilin A. Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions // IEEE Transactions on Knowledge and Data Engineering. 2005. Vol. 17, № 6. С. 734–749.

11. Gomez-Uribe C. A., Hunt N. The Netflix Recommender System: Algorithms, Business Value, and Innovation // ACM Transactions on Management Information Systems. 2016. Vol. 6, № 4. Article 13. 19 p.

12. Devlin J., Chang M.-W., Lee K., Toutanova K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding // Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics. 2019. С. 4171–4186.

13. Aggarwal C. C. Recommender Systems: The Textbook. Cham: Springer, 2016.

14. Dwork C., Roth A. The Algorithmic Foundations of Differential Privacy // Foundations and Trends in Theoretical Computer Science. 2014. Vol. 9, № 3–4. С. 211–407.

15. Sarwar B., Karypis G., Konstan J., Riedl J. Item-Based Collaborative Filtering Recommendation Algorithms // Proceedings of the 10th International Conference on World Wide Web. 2001. С. 285–295.

16. Kharchenko O., Raichev I., Bodnarchuk I., Zagorodna N. Optimization of software architecture selection for the system under design and reengineering // 14th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering, 2018.

17. Jannach D., Zanker M., Felfernig A., Friedrich G. Recommender Systems: An Introduction. Cambridge: Cambridge University Press, 2010.

18. Bobadilla J., Ortega F., Hernando A., Gutiérrez A. Recommender Systems Survey // Knowledge-Based Systems. 2013. Vol. 46. С. 109–132.

19. Pazzani M. J., Billsus D. Content-Based Recommendation Systems // The Adaptive Web. 2007. С. 325–341.

20. Pu P., Chen L., Hu R. A User-Centric Evaluation Framework for Recommender Systems // Proceedings of the 5th ACM Conference on Recommender Systems. 2011. С. 157–164.

21. Herlocker J. L., Konstan J. A., Terveen L. G., Riedl J. T. Evaluating Collaborative Filtering Recommender Systems // ACM Transactions on Information Systems. 2004. Vol. 22, № 1. С. 5–53.

22. Bertocci V. Modern Authentication with Azure Active Directory for Web Applications. Redmond: Microsoft Press, 2016.

23. Amatriain X., Basilico J. Recommender Systems in Industry: A Netflix Case Study // Recommender Systems Handbook. Berlin: Springer, 2015. С. 385–419.

24. Isinkaye F. O., Folajimi Y. O., Ojokoh B. A. Recommendation Systems: Principles, Methods and Evaluation // Egyptian Informatics Journal. 2015. Vol. 16, № 3. С. 261–273.

25. Schafer J. B., Konstan J. A., Riedl J. Recommender Systems in E-Commerce // Proceedings of the 1st ACM Conference on Electronic Commerce. 1999. С. 158–166.

26. Konstan J. A., Riedl J. Recommender Systems: From Algorithms to User Experience // User Modeling and User-Adapted Interaction. 2012. Vol. 22, № 1–2. С. 101–123.

27. Bodnarchuk I., Lisovyi V., Kharchenko O., Galai I. Adaptive method for assessment and selection of software architecture in flexible techniques of design // 13th International Scientific and Technical Conference on Computer Sciences and Information Technologies, 2018.

28. Google. Google Gemini API Documentation [Електронний ресурс]. 2024. Режим доступу: https://ai.google.dev/docs/gemini_api

29. Reback G. JWT Handbook [Електронний ресурс]. Auth0, 2023. Режим доступу: https://auth0.com/resources/ebooks/jwt-handbook

30. SQLite. SQLite Documentation [Електронний ресурс]. 2024. Режим доступу: https://www.sqlite.org/docs.html

31. Schneier B. Cryptography Engineering: Design Principles and Practical Applications. Hoboken: Wiley, 2010.

32. Grinberg M. Flask Web Development: Developing Web Applications with Python. Sebastopol: O'Reilly Media, 2018.

33. Streamlit. Streamlit Documentation [Електронний ресурс]. 2024. Режим доступу: https://docs.streamlit.io/

34. PostgreSQL Global Development Group. PostgreSQL Documentation [Електронний ресурс]. 2024. Режим доступу: https://www.postgresql.org/docs/

35. Redis Labs. Redis Documentation [Електронний ресурс]. 2024. Режим доступу: https://redis.io/documentation

36. Kharchenko A., Halay I., Bodnarchuk I. Multicriteria architecture choice of software system under design and reengineering // Computer Sciences and Information Technologies - Proceedings of the 11th International Scientific and Technical Conference, Institute of Electrical and Electronics Engineers Inc. , 2016

37. Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. Boston: Addison-Wesley, 2004.

38. Booch G., Rumbaugh J., Jacobson I. The Unified Modeling Language User Guide. Boston: Addison-Wesley, 2005.

39. Sommerville I. Software Engineering. Harlow: Pearson, 2016.

40. Harchenko A., Bodnarchuk I., Halay I. Decision support system of software architect // Proceedings of the 2013 IEEE 7th International Conference on Intelligent Data Acquisition and Advanced Computing Systems, 2013.

41. Pressman R. S. Software Engineering: A Practitioner’s Approach. New York: McGraw-Hill, 2014.

ДОДАТКИ

ДОДАТОК А

Коди програми

client.py

```
from time import time
import streamlit as st
import requests
import json
from datetime import datetime
import pandas as pd
from streamlit import logger
import google.generativeai as genai
import os
# 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
# Налаштуйте свій ключ
GEMINI_API_KEY = "AIzaSyBkHBZ5ErdhYAB_YdjGFYwtw7YgMLDKnUc"
try:
    genai.configure(api_key=GEMINI_API_KEY)
except Exception as e:
    logger.warning(f"Не вдалося налаштувати Gemini API: {e}")
# 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
# Функція для отримання рекомендацій від Gemini
def get_gemini_recommendation(user_request, books_catalog):
    """
    Використовує Google Gemini API для аналізу побажань користувача
    та надання рекомендацій з наявного асортименту
    """
    if not GEMINI_API_KEY:
        return None, "API ключ Gemini не налаштовано. Встановіть змінну середовища GEMINI_API_KEY."
    try:
        # Підготовка контексту з каталогу книг
        catalog_text = "Наявні книги в каталозі:\n"
        for book in books_catalog:
            catalog_text += f"- '{book['title']}' автор {book['author']}, категорія: {book.get('category', 'не вказано')}, ціна: {book['price']} грн\n"
            if book.get('description'):
                catalog_text += f"  Опис: {book['description']}\n"
        # Формування запиту до Gemini
        prompt = f"""Ти асистент книжкового магазину. Користувач звернувся з таким запитом: "{user_request}"
{catalog_text}
Проаналізуй запит користувача та:
1. Спочатку перевір, чи є в каталозі книги, які точно відповідають запиту користувача
2. Якщо точних відповідностей немає, порекомендуй найбільш підходящі книги з наявного асортименту, поясни чому саме ці книги можуть зацікавити користувача
3. Відповідь надай українською мовою
4. Будь дружелюбним та корисним"""
        # Налаштування моделі
        model = genai.GenerativeModel('gemini-1.5-flash')
        # Генерація відповіді
        response = model.generate_content(prompt)
        return response.text, None
    except Exception as e:
        debug_print(f"Помилка при зверненні до Gemini API: {str(e)}")
        return None, f"Помилка при отриманні рекомендації: {str(e)}"
# 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_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'])
def show_recommendations():
    """Сторінка рекомендацій з використанням Google Gemini"""
    st.header("🤖 Розумні рекомендації")
    debug_print("Відображення сторінки рекомендацій")
    if not st.session_state.user:
        st.warning("Для отримання персональних рекомендацій необхідно увійти в систему")
        debug_print("Спроба доступу до рекомендацій без авторизації")
        if st.button("Увійти"):
            debug_print("Натиснуто кнопку 'Увійти'")
            set_page("Вхід")
        return
    st.markdown(f"""
    <div style="background-color: {light_color}; padding: 15px; border-radius: 5px; margin-
bottom: 20px; color: {dark_text_color};">
        <h4>Як це працює?</h4>
        <p>Опишіть, яку книгу ви шукаєте або які теми вас цікавлять. Наш ШІ-асистент проаналізує
ваш запит і:</p>
        <ul>
            <li>Спочатку пошукає точні відповідності у нашому каталозі</li>
            <li>Якщо не знайде - запропонує найбільш підходящі альтернативи з наявних книг</li>
            <li>Поясніть, чому саме ці книги можуть вас зацікавити</li>
        </ul>
    </div>
    """, unsafe_allow_html=True)
    # Форма для введення побажань
    with st.form("recommendation_form"):
        user_request = st.text_area(
            "Опишіть ваші побажання:",
            placeholder="Наприклад: Шукаю книгу про штучний інтелект для початківців, або
цікавлюся психологією особистості, або хочу почитати щось схоже на Гаррі Поттера...",
            height=100
        )
        submit_button = st.form_submit_button("🔍 Отримати рекомендації")
    if submit_button and user_request:
        debug_print(f"Запит рекомендацій: {user_request}")
        # Перевірка налаштування API ключа
        if not GEMINI_API_KEY:
            st.error("API ключ Google Gemini не налаштовано. Зверніться до адміністратора.")
```

```
            st.info("Для роботи рекомендацій необхідно встановити змінну середовища GEMINI_API_KEY")
            debug_print("Gemini API ключ не встановлено")
            return
        with st.spinner("🤔 Аналізую ваш запит та шукаю найкращі варіанти..."):
            # Отримуємо весь каталог книг
            books = get_books()
            if not books:
                st.error("Не вдалося завантажити каталог книг")
                debug_print("Помилка завантаження каталогу для рекомендацій")
                return
            # Отримуємо рекомендації від Gemini
            recommendation, error = get_gemini_recommendation(user_request, books)
            if error:
                st.error(f"Помилка при отриманні рекомендацій: {error}")
                debug_print(f"Помилка Gemini API: {error}")
            elif recommendation:
                st.markdown("### 📚 Рекомендації для вас:")
                st.markdown(recommendation)
                # Додаємо кнопку для переходу до каталогу
                if st.button("Перейти до каталогу"):
                    debug_print("Перехід до каталогу з рекомендацій")
                    set_page("Каталог")
            else:
                st.warning("Не вдалося отримати рекомендації. Спробуйте ще раз.")
                debug_print("Порожня відповідь від Gemini API")
    # Приклади запитів для натхнення
    st.markdown("---")
    st.subheader("💡 Приклади запитів:")
    example_requests = [
        "Хочу почитати щось про космос та астрономію",
        "Шукаю детективи в стилі Агати Крісті",
        "Цікавлюся книгами про особистісний розвиток",
        "Потрібна література для вивчення програмування",
        "Хочу фентезі з драконами та магією"
    ]
    cols = st.columns(2)
    for i, example in enumerate(example_requests):
        with cols[i % 2]:
            if st.button(example, key=f"example_{i}"):
                st.session_state.recommendation_example = example
                debug_print(f"Обрано приклад запиту: {example}")
                st.rerun()
    # Якщо був обраний приклад, автоматично вставляємо його
    if 'recommendation_example' in st.session_state:
        st.info(f"Використовуйте цей приклад: {st.session_state.recommendation_example}")
        del st.session_state.recommendation_example
# 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};
        }}
        /* Прибираємо стилі для радіокнопок */
        .stRadio > div {{
            display: none !important;
        }}
        </style>
        """,
        unsafe_allow_html=True,
    )
    # Відображення бічної панелі з логотипом
    st.sidebar.markdown(
        f"""
        <div style="text-align: center; padding: 20px 0;">
            <div style="font-size: 48px; color: {primary_color};">📚</div>
            <h2 style="color: {primary_color}; margin: 10px 0;">Book Haven</h2>
            <p style="color: {secondary_color}; font-style: italic;">Ваш книжковий рай</p>
        </div>
        <hr style="border-color: {accent_color};">
        """,
        unsafe_allow_html=True
    )
    # Опції меню відповідно до ролі користувача
    menu_options = ["Каталог", "Корзина", "Рекомендації"]
    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"])
    # Замість радіокнопок використовуємо кнопки для навігації
    st.sidebar.markdown("<h3 style='text-align: center;'>Навігація</h3>",
unsafe_allow_html=True)
    for option in menu_options:
        # Підсвічуємо активну сторінку
        is_active = st.session_state.page == option
        button_style = f"background-color: {secondary_color};" if is_active else ""
        if st.sidebar.button(
            option,
            key=f"nav_{option}",
            use_container_width=True,
            help=f"Перейти до {option}"
        ):
            debug_print(f"Натиснуто навігаційну кнопку: {option}")
            if option == "Вихід":
```

```
                logout_user()
                set_page("Каталог")
            else:
                set_page(option)
    # Відображення інформації про користувача
    if st.session_state.user:
        st.sidebar.markdown("---")
        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:
        st.sidebar.markdown("---")
        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", use_container_width=True):
                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()
    elif st.session_state.page == "Управління замовленнями":
        show_manage_orders()
    elif st.session_state.page == "Рекомендації":
        show_recommendations()
```

```
    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)
```