

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

ПОЯСНЮВАЛЬНА ЗАПИСКА

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

на тему:

«Книжний каталогізатор з системою рекомендацій на ШІ»

Виконав студент групи ПЗ-21-2
Слівко Володимир Андрійович

Керівник роботи:
Концидайло Андрій Михайлович

Рецензент:
Морозов Андрій Васильович

Житомир – 2025

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

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення
Тетяна ВАКАЛЮК
«26» лютого 2025р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: Слівко Володимир Андрійович

Керівник роботи: Концидайло Андрій Михайлович

Тема роботи: «Книжний каталогізатор з системою рекомендацій на ШІ»,
затверджена Наказом закладу вищої освіти №61/с від 14.02.2025р.

Вихідні дані для роботи: створення книжкового каталогізатора з системою
рекомендацій на основі штучного інтелекту для управління бібліотекою, читання
книг та аналізу читацьких уподобань.

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

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Концидайло А.Н.	27.02.2025	27.02.2025
2	Концидайло А.Н.	27.02.2025	27.02.2025
3	Концидайло А.Н.	27.02.2025	27.02.2025

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з вебдодатку "Книжковий каталогізатор з системою рекомендацій на основі штучного інтелекту" та пояснювальної записки. Пояснювальна записка до роботи містить 63 сторінок, 1 додаток, 27 ілюстрацій та 12 таблиць.

Метою роботи є створення зручного та багатофункціонального інструменту для каталогізації книжкових колекцій, інтеграції онлайн-читалки та забезпечення персоналізованих рекомендацій із застосуванням штучного інтелекту. Розробка спрямована на оптимізацію читацького досвіду, спрощення управління бібліотекою та стимулювання інтересу до читання завдяки інноваційним технологіям.

Архітектура додатку базується на клієнт-серверній моделі, де клієнтська частина реалізована за допомогою бібліотеки Streamlit, а серверна — на фреймворку Flask із використанням бази даних SQLite. Система рекомендацій інтегрована через Gemini API від Google, що дозволяє аналізувати читацькі вподобання та генерувати персоналізовані пропозиції. Додаток підтримує різні формати книг (EPUB, PDF, FB2, TXT), забезпечуючи витягнення метаданих та онлайн-читання.

Інтерфейс програми є інтуїтивно зрозумілим, надаючи користувачам можливість додавати книги, відстежувати прогрес читання, зберігати закладки й цитати, а також переглядати статистику читацької активності.

КЛЮЧОВІ СЛОВА: ВЕБДОДАТОК, ШТУЧНИЙ ІНТЕЛЕКТ, КАТАЛОГІЗАЦІЯ, РЕКОМЕНДАЦІЇ, КНИГИ, STREAMLIT, FLASK, SQLITE, GEMINI API, ЧИТАННЯ.

					ІПЗ.КР.Б – 121 – 25 – ПЗ			
Змн	Арк.	№ докум.	Підпис	Дата	Книжний каталогізатор з системою рекомендацій на III	Літ.	Арк.	Аркушів
Розроб.		Слівко В.А.						
Керівник		Концидайло А.Н.					3	110
Рецензент		Морозов А.В.				Житомирська політехніка, група ІПЗ-21-2		
Зав.каф.		Вакалюк Т.А.						

ABSTRACT

The graduation thesis for a bachelor's degree consists of a web application titled "Book Cataloguer with an AI-Based Recommendation System" and an explanatory note. The explanatory note contains 63 pages, 1 appendix, 27 illustrations, and 12 tables.

The purpose of the thesis is to create a convenient and multifunctional tool for cataloguing book collections, integrating an online reader, and providing personalized recommendations using artificial intelligence. The development is aimed at optimizing the reading experience, simplifying library management, and encouraging interest in reading through innovative technologies.

The application's architecture is based on a client-server model, where the client-side is implemented using the Streamlit library, and the server-side is built on the Flask framework with SQLite as the database. The recommendation system is integrated via Google's Gemini API, allowing analysis of reading preferences and generating personalized suggestions. The application supports various book formats (EPUB, PDF, FB2, TXT), enabling metadata extraction and online reading.

The program interface is intuitive, offering users the ability to add books, track reading progress, save bookmarks and quotes, and view reading activity statistics.

KEYWORDS: WEB APPLICATION, ARTIFICIAL INTELLIGENCE, CATALOGUING, RECOMMENDATIONS, BOOKS, STREAMLIT, FLASK, SQLITE, GEMINI API, READING.

Зміст

РЕФЕРАТ.....	3
ABSTRACT.....	4
ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ.....	6
ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ СТВОРЕННЯ КНИЖКОВОГО КАТАЛОГІЗАТОРА З СИСТЕМОЮ РЕКОМЕНДАЦІЙ НА ІІІ.....	9
1.1 Постановка задачі.....	9
1.2 Аналіз аналогів програмного продукту.....	12
1.3 Вибір архітектури.....	16
1.4 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення.....	21
Висновок до першого розділу.....	24
РОЗДІЛ 2. ПРОЄКТУВАННЯ КНИЖКОВОГО КАТАЛОГІЗАТОРУ З СИСТЕМОЮ РЕКОМЕНДАЦІЙ НА ІІІ.....	25
2.1. Визначення варіантів використання та об'єктно орієнтованої структури системи.....	25
2.2. Розробка структура файлів системи.....	30
2.3. Розробка бази даних системи	34
2.4. Реалізація функціоналу додатку	39
Висновок до другого розділу.....	42
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ.....	43
3.1. ПОРЯДОК ВСТАНОВЛЕННЯ ТА НАЛАШТУВАННЯ СИСТЕМИ.....	43
3.2. СТРУКТУРА ІНТЕРФЕЙСУ	48
3.3. Тестування роботи програмного продукту.....	57
Висновок до третього розділу.....	60
ВИСНОВКИ.....	61
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	62
ДОДАТКИ.....	64
ДОДАТОК А ЛІСТИНГ ПРОГРАМ.....	65

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

AI – Штучний інтелект

API – Application Programming Interface (Інтерфейс програмування застосунків)

БД – База даних

HTML – HyperText Markup Language (Мова розмітки гіпертексту)

CSS – Cascading Style Sheets (Каскадні таблиці стилів)

CRUD – Create, Read, Update, Delete (Створити, Прочитати, Оновити, Видалити)

EPUB – Electronic Publication (Електронна публікація)

FB2 – FictionBook

HTTP – Hypertext Transfer Protocol

JSON – JavaScript Object Notation (Запис об'єктів JavaScript)

REST – Representational State Transfer

SQLITE – Structured Query Language Lite

ВСТУП

Сучасний розвиток інформаційних технологій кардинально змінює підходи до організації зберігання, доступу та використання книжкової продукції. Зростання популярності цифрових бібліотек, електронних книг і персоналізованих систем рекомендацій, а також потреба у швидкому доступі до літератури роблять актуальним створення інноваційних програмних рішень для управління книжковими колекціями. Розробка "Книжкового каталогізатора з системою рекомендацій на основі штучного інтелекту" є відповіддю на ці виклики, адже вона дозволяє не лише систематизувати книги, а й надавати користувачам інтелектуальні рекомендації, адаптовані до їхніх читацьких вподобань.

Актуальність цього завдання зумовлена як технологічними тенденціями, що сприяють цифровізації бібліотечних систем, так і соціальною значимістю, адже доступність літератури та персоналізований підбір книг сприяють розвитку освіти, культури та інтелектуального потенціалу суспільства. У контексті цифрової трансформації створення подібних систем є важливим кроком для адаптації книжкової індустрії до потреб сучасного користувача.

Метою є створення зручного та багатофункціонального інструменту для каталогізації книжкової колекції, інтеграції онлайн-читалки та забезпечення інтелектуальних рекомендацій на основі штучного інтелекту.

Мета дослідження включає наступні завдання:

- надання користувачам можливості легко додавати та видаляти книги.
- зберігати закладки й цитати.
- отримувати персоналізовані рекомендації, які базуються на їхніх інтересах і читацькій історії. популяризації літератури через інтуїтивний і технологічно просунутий інтерфейс.
- редагувати інформацію про книгу.
- відстежувати прогрес читання.

- надання можливості пошуку та фільтрації книг за різними критеріями (назвою, автором, жанром).
- забезпечення функціоналу для відображення детальної інформації про книги, включаючи обкладинки, анотації та додаткові метадані.
- розробка зручного та інтуїтивно зрозумілого інтерфейсу користувача.

Предметом дослідження виступають методи та засоби розробки, що застосовуються для створення інноваційних програмних рішень для управління книжковими колекціями. Особлива увага приділяється технологіям організації зберігання та доступу до книжкової продукції, інтеграції онлайн-читалки, механізмів роботи з різними книжковими форматами (EPUB, PDF, FB2, TXT), а також методам персоналізації — зокрема, алгоритмам рекомендацій на основі штучного інтелекту (з використанням, наприклад, Gemini API) та читацької історії користувачів, включно зі збереженими закладками й цитатами.

Об'єктом дослідження є процес розробки та функціонування "Книжкового каталогізатора з системою рекомендацій на основі штучного інтелекту" як комплексного програмного додатку. Це включає як клієнтську частину (на базі Streamlit), так і серверну частину (на Flask з Python), взаємодію з базою даних SQLite, механізми обробки та вилучення даних з книжкових файлів різних форматів (EPUB, PDF, FB2, TXT), а також дослідження та реалізацію алгоритмів генерації персоналізованих рекомендацій на основі інтеграції з Gemini API.

РОЗДІЛ 1. АНАЛІЗ ЗАДАЧІ ТА ОСОБЛИВОСТЕЙ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ СТВОРЕННЯ КНИЖКОВОГО КАТАЛОГІЗАТОРА З СИСТЕМОЮ РЕКОМЕНДАЦІЙ НА ІІІ

1.1 Постановка задачі

Головною метою розробки веб-додатку «Книжковий каталогізатор з системою рекомендацій на ІІІ» є створення зручної платформи для організації, перегляду, читання та аналізу особистої бібліотеки книг із можливістю отримання персоналізованих рекомендацій на основі штучного інтелекту. Цей додаток орієнтований на широке коло користувачів, включаючи книголюбів, дослідників літератури, студентів та всіх, хто прагне систематизувати свою читацьку діяльність. Веб-додаток має забезпечити інтуїтивно зрозумілий інтерфейс, адаптивний дизайн, підтримку різноманітних форматів книг, а також інтеграцію з ІІІ для аналізу читацьких уподобань.

У рамках реалізації проєкту необхідно розробити веб-додаток, який дозволить користувачам додавати книги до каталогу, редагувати їх метадані, читати книги онлайн, відстежувати прогрес читання, зберігати цитати та закладки, а також отримувати рекомендації на основі їхньої бібліотеки та вподобань. Основними етапами реалізації цього завдання є створення серверної частини додатку.

Створення структури бази даних для зберігання інформації про книги, прогрес читання, закладки, цитати, історію читання та статистику.

Створення моделей бази даних з відповідними схемами. Зокрема, розроблено модель книг, що включає ідентифікатор, назву, автора, рік видання, жанр, опис, серію, теги, обкладинку, шлях до файлу, формат файлу, статус читання, поточну сторінку, загальну кількість сторінок та прогрес читання. Була створена модель закладок, яка включає ідентифікатор, ідентифікатор книги, позицію (номер сторінки), назву та дату створення.

Окрема модель цитат розроблена для зберігання ідентифікатора, ідентифікатора книги, тексту цитати, позиції та дати створення. Також реалізовано модель історії читання, що включає ідентифікатор, ідентифікатор книги, номер сторінки, час читання та дату. Створення API для взаємодії з базою даних. Для книг реалізовано CRUD-операції (створення, читання, оновлення, видалення), завантаження файлів книг та обкладинок, витягнення метаданих. Для прогресу читання передбачено оновлення поточної сторінки, статусу та часу читання. Для закладок і цитат розроблено CRUD-операції. Для рекомендацій забезпечено інтеграцію з Gemini API для генерації рекомендацій на основі метаданих книги, тегів або вподобань. З метою аналізу даних, створено API для статистики, що включає агрегацію даних про читання, таких як загальна кількість книг, прочитаних годин, розподіл за жанрами, авторами та роками.

Сторінка каталогу відображає книги у вигляді карток із обкладинками, статусом та кнопками дій (читати, редагувати, видалити). Сторінка додавання книги: форма для завантаження файлу, ручного введення метаданих та опціонального витягнення метаданих із файлу. Сторінка редагування книги: форма для оновлення метаданих та обкладинки. Читалка: відображення вмісту книги з можливістю вибору теми, шрифту, розміру тексту, додавання закладок та цитат.

Сторінка пошуку показує пошук за назвою, автором, жанром, роком або статусом.

Сторінка статистики відображає ключові метрик (кількість книг, прогрес) та графіків (жанри, автори, роки).

Сторінка рекомендацій демонструє вкладки для персональних рекомендацій, рекомендацій за книгою або тегами.

Бічна панель забезпечує навігацію між сторінками, кнопка додавання книги, відображення недавньої активності.

Розробка клієнтської логіки включала використання бібліотеки requests для взаємодії з серверними API. Також було реалізовано управління станом сесії (st.session_state) для збереження поточної сторінки, книги, прогресу читання та налаштувань читалки. Важливою частиною було забезпечення обробки виділення тексту в читалці для створення цитат через JavaScript. Крім того, здійснювалася генерація графіків статистики за допомогою matplotlib. Тестування клієнтської частини передбачало перевірку адаптивності, коректності відображення вмісту книг, а також належної роботи читалки та рекомендацій. Для генерації рекомендацій використовувалося Gemini API на основі даних конкретної книги (назва, автор, жанр), тегів, введених користувачем, та текстового опису вподобань. Форматування відповідей у JSON для відображення у клієнтській частині.

У результаті виконання поставленого завдання отримано веб-додаток із серверною частиною, яка забезпечує управління даними, обробку файлів книг і рекомендації через ШІ, та клієнтською частиною, що надає користувачу зручний інтерфейс для каталогу, читання та аналізу. Додаток підтримує різноманітні формати книг (EPUB, PDF, TXT, FB2), адаптований до потреб користувачів завдяки інтуїтивному дизайну та інтеграції з ШІ, а також забезпечує детальну статистику читацької активності.

1.2 Аналіз аналогів програмного продукту

Розробка книжкового каталогізатора з системою рекомендацій на основі штучного інтелекту (ШІ) є актуальним завданням у контексті сучасних інформаційних технологій. Такі системи спрямовані на автоматизацію процесів управління бібліотечними ресурсами, полегшення доступу користувачів до літератури та надання персоналізованих рекомендацій. Для оцінки розробленого програмного продукту необхідно проаналізувати існуючі аналоги, їх функціональні можливості, переваги та недоліки. Розглянуто чотири аналоги подібних систем, які функціонують на ринку програмного забезпечення, та проведено їх порівняння з розробленою системою.

Першим аналогом є Goodreads (рис. 1.1) – популярна платформа для каталогізації книг та обміну рекомендаціями [1]. Goodreads надає користувачам можливість швидко реєструватися через електронну пошту або соціальні мережі, створювати особистий кабінет, додавати книги до списку «Обране» та переглядати історію прочитаного. Система використовує алгоритми рекомендацій, базовані на оцінках користувачів та їхніх уподобаннях. Проте вона не має розширених інструментів для менеджерів бібліотек чи локалізації українською мовою, що обмежує її застосування в україномовному середовищі.

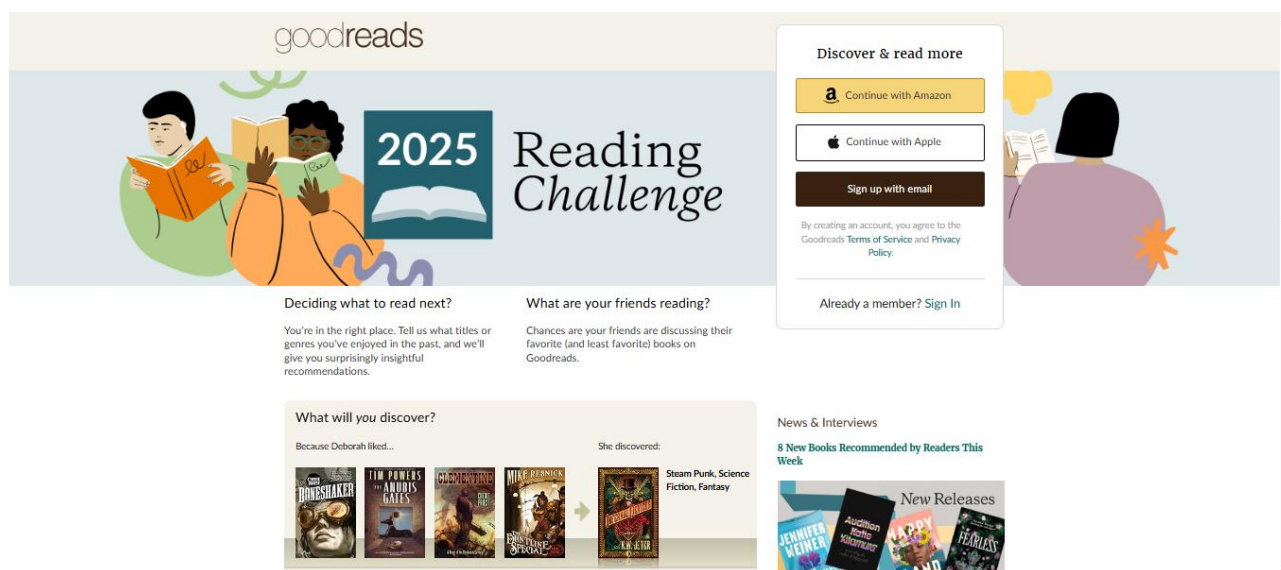


Рис. 1.1. Сторінка Goodreads [1]

Другим аналогом є LibraryThing (рис. 1.2) – онлайн-сервіс для каталогізації особистих бібліотек [2]. Цей продукт також підтримує швидку реєстрацію, особистий кабінет користувача та функцію додавання книг до обраного. LibraryThing дозволяє вести облік книг, однак не має повноцінних інструментів для менеджерів бібліотек, таких як управління замовленнями чи інвентаризація. Історія дій користувача доступна, а локалізація українською мовою підтримується частково, що робить сервіс зручним для україномовних користувачів, але недостатньо адаптованим для професійного використання.

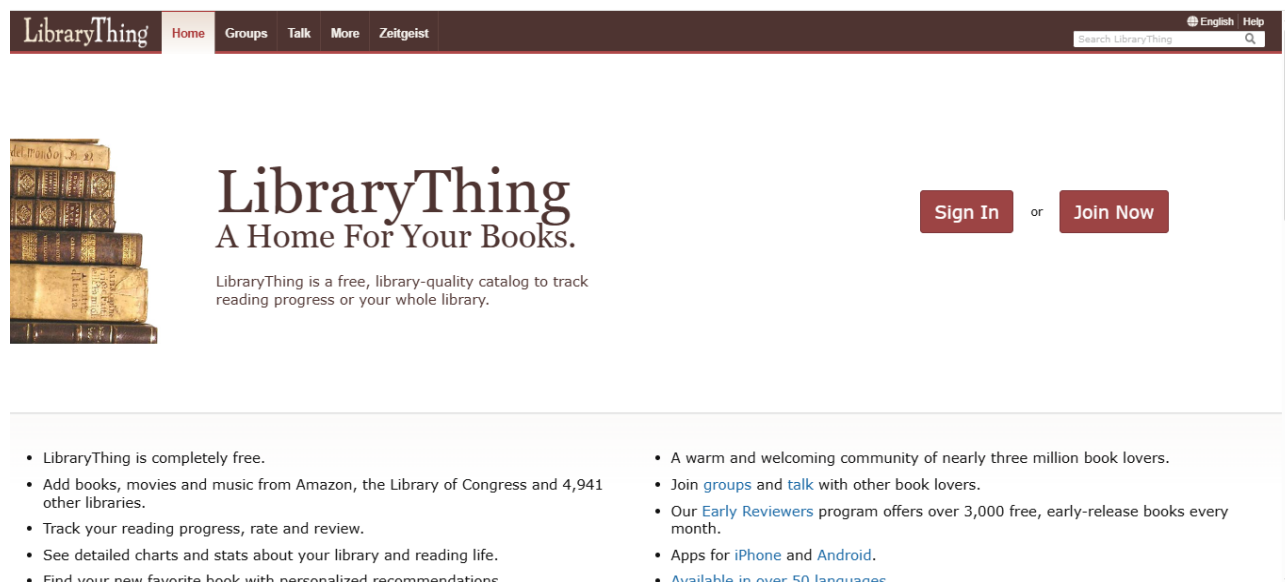


Рис. 1.2. Сторінка LibraryThing [2]

Третім аналогом виступає Calibre (рис. 1.3) – програмне забезпечення з відкритим кодом, призначене для управління електронними книгами [3]. Calibre підтримує швидку реєстрацію (хоча вона не є обов’язковою для локального використання), має особистий кабінет у вигляді бібліотеки користувача та дозволяє вести історію завантажених чи прочитаних книг. Проте функція «Обране» тут реалізована опосередковано через теги, а інструменти для менеджерів бібліотек відсутні. Локалізація українською мовою не передбачена, що ускладнює використання програми у відповідному контексті.

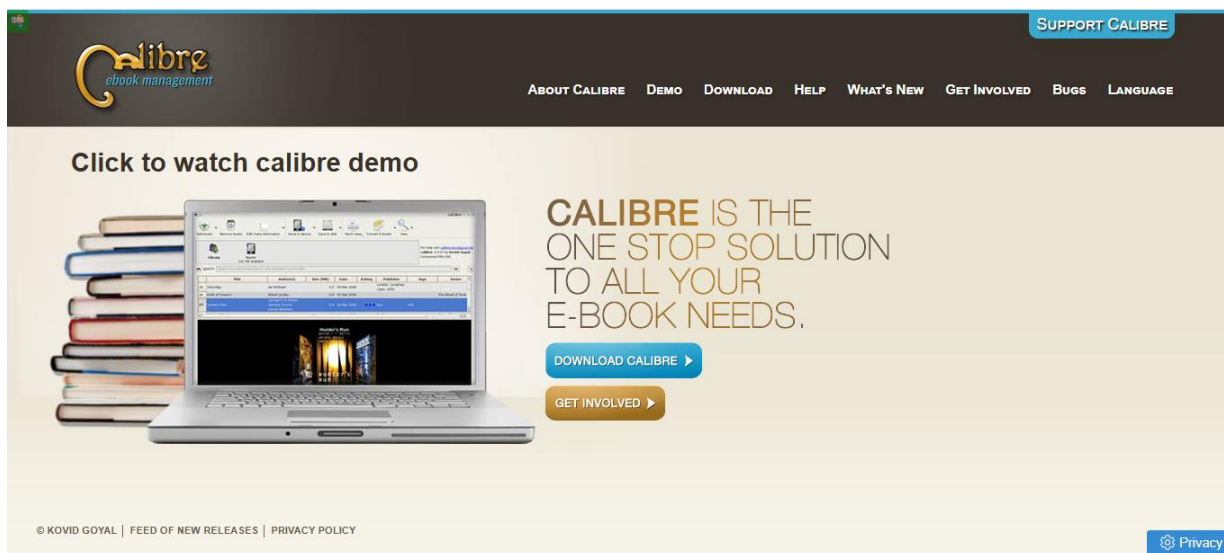


Рис. 1.3. Сторінка для завантаження Calibre [3]

Четвертим аналогом є Book Catalogue (рис. 1.4) – мобільний додаток для каталогізації книг [4]. Цей продукт дозволяє швидко реєструватися, створювати кабінет користувача, додавати книги до списку «Обране» та переглядати історію дій. Однак він не пропонує інструментів для менеджерів, таких як управління фондами чи замовленнями. Локалізація українською мовою відсутня, що робить додаток менш привабливим для україномовних користувачів у порівнянні з розробленою системою.

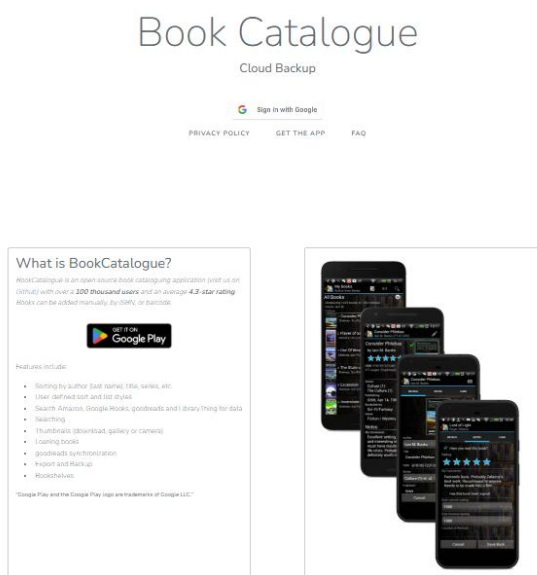


Рис. 1.4. Сторінка завантаження Book Catalogue [4]

Розроблена система, представлена у вигляді коду проєкту, поєднує ключові функції, необхідні для ефективної каталогізації та рекомендацій. Вона підтримує швидку реєстрацію, особистий кабінет користувача, список «Обране», інструменти для менеджерів (наприклад, управління замовленнями чи інвентаризацію) та історію дій. Важливою перевагою є повна локалізація українською мовою, що робить її адаптованою до потреб україномовних користувачів. Система рекомендацій на основі ШП, реалізована в проєкті, аналізує вподобання користувачів і пропонує персоналізовані пропозиції, що вигідно вирізняє її серед аналогів.

Порівняння функціональних можливостей аналогів та розробленої системи дозволяє виявити їхні сильні та слабкі сторони. Goodreads і LibraryThing орієнтовані переважно на індивідуальних користувачів і мають обмежені можливості для професійного бібліотечного менеджменту. Calibre підходить для локального використання, але не відповідає потребам онлайн-каталогізації чи україномовного середовища. Book Catalogue є зручним для мобільних пристроїв, проте поступається за функціоналом менеджменту та локалізацією. Розроблена система, натомість, поєднує широкий спектр функцій, включно з підтримкою ШП-рекомендацій, що робить її конкурентоспроможною.

Отже, порівняння за критеріями наведене в таблиці 1.1.

Таблиця 1.1

Порівняння розробленої системи з аналогами

Система	Реєстрація	Кабінет користувача	Список «Обране»	Інструменти менеджера	Історія перегляду	Локалізація (UA)
Goodreads	Швидка	+	+	-	+	-
LibraryThing	Швидка	+	+	-	+	+
Calibre	Швидка	+	-	-	+	-
Book Catalogue	Швидка	+	+	-	+	-
Розроблена система	Швидка	+	+	+	+	+

Таким чином, розроблена система вирізняється комплексним підходом до каталогізації та рекомендацій, поєднуючи зручність для користувачів і функціональність для менеджерів. Наявність локалізації українською мовою та інтеграція ШІ роблять її перспективним рішенням у порівнянні з існуючими аналогами.

1.3 Вибір архітектури

Розробка книжкового каталогізатора з системою рекомендацій на основі штучного інтелекту (ШІ) потребує ретельного вибору архітектури, яка б забезпечила ефективність, масштабованість, зручність використання та інтеграцію сучасних технологій. Важливо розглянути ключові показники вибору архітектури для реалізації проєкту, враховуючи реалізовані коди програмного забезпечення (client.py та server.py). Архітектурне рішення базується на аналізі функціональних вимог, технічних можливостей та особливостей використання інформаційних технологій для створення інтерактивного додатка.

Для реалізації книжкового каталогізатора обрано клієнт-серверну архітектуру (рис. 1.5), яка є оптимальною для розподілених систем із чітким поділом функціональних обов'язків між клієнтською та серверною частинами [5]. Цей вибір обумовлений необхідністю централізованого зберігання даних (книжкових файлів, метаданих, історії читання тощо) та забезпечення доступу до них через зручний користувацький інтерфейс. Клієнтська частина, реалізована за допомогою бібліотеки Streamlit (client.py), відповідає за візуалізацію даних, взаємодію з користувачем та відправлення запитів до сервера [6]. Серверна частина, побудована на фреймворку Flask (server.py), обробляє запити, управляє базою даних і забезпечує інтеграцію з ШІ для рекомендацій [7].

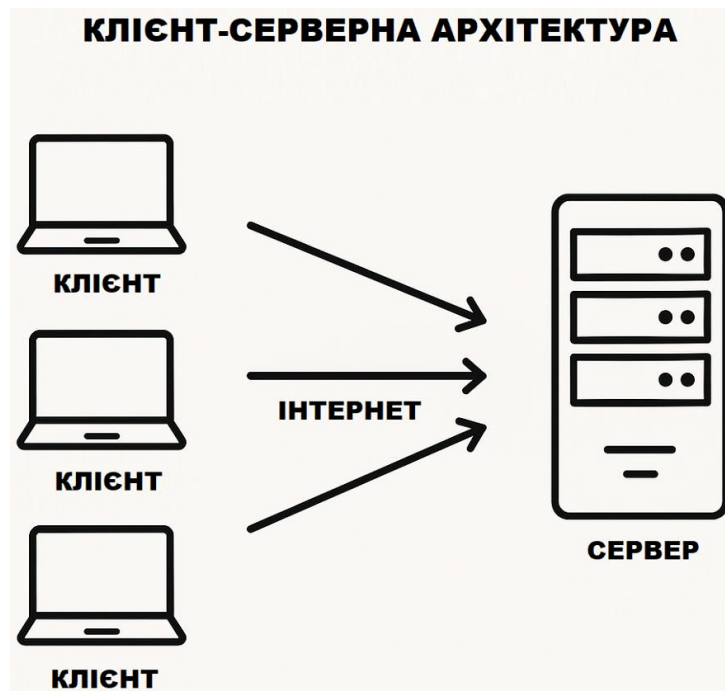


Рис. 1.5. Схема клієнт-серверної архітектури

Клієнт-серверна модель показує, що усі книги, обкладинки та метадані зберігаються на сервері, що спрощує їх обробку та оновлення. Гнучкість доступу: користувач може працювати з каталогом через веб-інтерфейс із будь-якого пристрою, підключеного до мережі.

Сервер виконує складні операції (наприклад, витягнення метаданих чи генерацію рекомендацій), тоді як клієнт зосереджений на відображенні результатів.

Такий підхід є стандартом для веб-додатків, що потребують інтерактивності та обробки великих обсягів даних, як це видно з реалізації API-ендпоінтів у `server.py` (наприклад, `/api/books`, `/api/recommendations`).

Клієнтська частина додатка побудована на основі Streamlit, що є інноваційним рішенням для створення веб-інтерфейсів із мінімальними зусиллями [9]. Вибір цього інструменту зумовлений факторами, описаними в таблиці 1.2.

Таблиця 1.2

Фактори вибору технологічного стека для клієнтської частини

№	Назва	Опис
1	Простота розробки	Streamlit дозволяє швидко створювати інтерактивні інтерфейси без необхідності глибоких знань фронтенд-технологій (HTML, CSS, JavaScript), що прискорює прототипування та розробку
2	Інтеграція з Python	оскільки серверна частина також написана на Python (Flask), використання Streamlit забезпечує єдність технологічного стека, полегшуючи інтеграцію та підтримку
3	Гнучкість кастомізації	як видно з коду client.py, додаток використовує власні CSS-стилі (local_css()) для налаштування зовнішнього вигляду, що дозволяє адаптувати інтерфейс до потреб користувачів

Функціонал клієнтської частини включає відображення каталогу книг, додавання нових книг, редагування метаданих, читання книг онлайн і перегляд рекомендацій. Наприклад, функція `display_book_card()` демонструє модульний підхід до відображення книг у вигляді карток, що сприяє зрозумілості та зручності використання.

Серверна частина реалізована на базі Flask — легковагового фреймворку, який ідеально підходить для створення RESTful API [10]. Вибір Flask обґрунтований причинами, описаними в таблиці 1.3.

Таблиця 1.3

Причини вибору Flask

№	Назва	Опис
1	Мінімалістичність	фреймворк не накладає жорстких обмежень, дозволяючи розробнику гнучко структурувати код відповідно до потреб проєкту
2	Швидкість реалізації	завдяки простоті Flask вдалося швидко створити набір ендпоінтів для управління книгами, закладками, цитатами та рекомендаціями (наприклад, /api/books/<int:book_id>/content, /api/recommendations)
3	Розширюваність	інтеграція з зовнішніми бібліотеками, такими як sqlite3 для роботи з базою даних чи google.generativeai для ШІ-рекомендацій, виконана без ускладнень

Серверна архітектура побудована за принципом REST, де кожен ресурс (книга, закладка, цитата) має власний URL і підтримує стандартні HTTP-методи (GET, POST, PUT, DELETE). Це забезпечує чітку структуру запитів і відповідей, як видно з функцій `get_books()`, `add_book()`, `update_progress()` тощо.

Для зберігання даних обрано SQLite — легковагову реляційну базу даних, яка ідеально підходить для проєктів середнього масштабу [11]. Основні аргументи на користь SQLite наведені в таблиці 1.4.

Таблиця 1.4

Причини вибору SQLite

№	Назва	Опис
1	Простота інтеграції	SQLite не потребує окремого серверного процесу, що спрощує розгортання додатка (див. <code>init_db()</code> у <code>server.py</code>)
2	Достатня продуктивність	для каталогу книг із кількома тисячами записів SQLite забезпечує швидкий доступ до даних без значних накладних витрат
3	Структурованість	база даних включає таблиці <code>books</code> , <code>bookmarks</code> , <code>quotes</code> і <code>reading_history</code> , що відображають основні сутності системи (див. функцію <code>init_db()</code>)

Реляційна модель дозволяє ефективно зв'язувати книги з їхніми закладками, цитатами та історією читання через зовнішні ключі, що підвищує цілісність даних. Наприклад, видалення книги автоматично видаляє пов'язані записи завдяки налаштуванню ON DELETE CASCADE.

Однією з ключових особливостей системи є використання ІІІ для генерації рекомендацій, що реалізовано через Gemini API від Google (google.generativeai) [12]. Вибір цього підходу обґрунтований в таблиці 1.5.

Таблиця 1.5

Причини вибору Gemini API від Google

№	Назва	Опис
1	Потужність ІІІ	Gemini API здатен аналізувати текстові запити (теги, вподобання, характеристики книг) і генерувати релевантні рекомендації у форматі JSON, як видно з функції <code>get_recommendations_from_ai()</code>
2	Гнучкість	система підтримує різні сценарії рекомендацій — на основі конкретної книги, тегів чи опису вподобань користувача (ендпоінт <code>/api/recommendations</code>)
3	Зовнішня інтеграція	використання хмарного API замість локальної моделі ІІІ зменшує обчислювальне навантаження на сервер і дозволяє легко оновлювати алгоритми рекомендацій

У коді `server.py` видно, що ІІІ генерує рекомендації як список книг із полями `title`, `author`, `year`, `description` і `genre`, які потім адаптуються для відображення на клієнтській частині.

Архітектура передбачає складну логіку обробки книжкових файлів (EPUB, PDF, TXT, FB2), що реалізовано через спеціалізовані бібліотеки (`ebooklib`, `fitz`, `BeautifulSoup`) [13, 14]. Ключові показники функції `extract_metadata_from_epub()`, `extract_metadata_from_pdf()` тощо дозволяють витягувати метадані залежно від формату файлу.

Зберігання файлів: книги та обкладинки зберігаються у файловій системі (`BOOKS_DIR`, `COVERS_DIR`), а їхні шляхи записуються в базу даних, що оптимізує доступ до великих даних.

Вміст книги конвертується в HTML для відображення в інтерфейсі (`get_book_content()`), що забезпечує кросплатформність. Такий підхід дозволяє гнучко працювати з різними форматами, зберігаючи баланс між продуктивністю та зручністю.

Вибір клієнт-серверної архітектури з використанням Streamlit і Flask, SQLite як бази даних і Gemini API для рекомендацій є збалансованим рішенням, яке враховує функціональні вимоги проєкту, простоту реалізації та потенціал для подальшого розвитку. Ця структура забезпечує зручний інтерфейс, ефективне управління даними та інноваційні можливості завдяки ШІ, що відповідає меті створення сучасного книжкового каталогізатора. Реалізовані коди (`client.py`, `server.py`) демонструють практичну успішність обраного підходу, поєднуючи технологічну гнучкість із користувацькою орієнтованістю.

1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Розробка книжкового каталогізатора з системою рекомендацій на основі штучного інтелекту (ШІ) вимагає ретельного підходу до вибору інструментальних засобів, які забезпечують ефективність, масштабованість та зручність використання системи. Необхідно обґрунтувати вибір програмних і технічних рішень, використаних у реалізації проєкту, а також визначаються мінімальні вимоги до апаратного забезпечення для забезпечення стабільної роботи системи.

Для реалізації клієнтської частини системи обрано мову програмування Python у поєднанні з бібліотекою Streamlit. Python є однією з найпопулярніших мов завдяки своїй простоті, широкій підтримці бібліотек та активній спільноті розробників [15]. Streamlit, у свою чергу, дозволяє швидко створювати інтерактивні вебінтерфейси з мінімальними затратами на написання коду для фронтенду, що є ключовою перевагою для прототипування та розробки користувацьких інтерфейсів [16]. У коді `client.py` видно, як Streamlit

використовується для побудови сторінок каталогу, читалки, пошуку та рекомендацій, що демонструє його гнучкість у реалізації складних інтерфейсів із підтримкою динамічних елементів, таких як фільтри, графіки та вкладки.

Для серверної частини (server.py) також обрано Python із фреймворком Flask. Flask є легким і гнучким вебфреймворком, який ідеально підходить для створення RESTful API, що забезпечує обмін даними між клієнтом і сервером [17]. Його мінімалістичний підхід дозволяє розробнику зосередитися на логіці додатку, уникаючи надмірної складності. Використання Flask у проєкті обґрунтоване необхідністю обробки запитів для управління книгами, закладками, цитатами та рекомендаціями, а також інтеграцією з базою даних і III-сервісами.

Для зберігання даних про книги, прогрес читання, закладки та цитати обрано SQLite – легку реляційну базу даних, яка не потребує окремого серверного процесу. SQLite інтегрована безпосередньо в додаток, що спрощує розгортання системи та зменшує залежність від зовнішнього програмного забезпечення [18]. У server.py видно створення таблиць для книг (books), закладок (bookmarks), цитат (quotes) та історії читання (reading_history), що забезпечує структуроване зберігання даних. Вибір SQLite виправданий для проєкту такого масштабу, оскільки він підтримує базові операції CRUD (Create, Read, Update, Delete) і є достатньо продуктивним для невеликих і середніх бібліотек [19].

Для роботи з різними форматами книг (EPUB, PDF, FB2, TXT) використано спеціалізовані бібліотеки: ebooklib для EPUB [20], PyMuPDF (fitz) для PDF [21], BeautifulSoup для парсингу FB2 [22]. Ці інструменти дозволяють витягувати метадані (назва, автор, рік, обкладинка) та вміст книг для відображення в онлайн-читалці. Наприклад, функція `extract_metadata_from_epub` у server.py демонструє обробку EPUB-файлів із підрахунком сторінок на основі кількості абзаців, що є практичним рішенням для адаптації вмісту до інтерфейсу читалки.

Система рекомендацій базується на API Google Gemini, що забезпечує генерацію рекомендацій на основі даних про книги, теги або користувацькі вподобання [23]. У server.py функція get_recommendations_from_ai використовує модель Gemini для створення персоналізованих пропозицій у форматі JSON. Вибір Gemini обумовлений його здатністю обробляти природну мову та генерувати структуровані відповіді, що значно спрощує інтеграцію з додатком. Альтернативи, такі як OpenAI API, могли б бути розглянуті, але Gemini обрано через доступність і сумісність із Python-екосистемою.

Для відображення статистики (графіки розподілу за жанрами, авторами, роками) використано бібліотеку matplotlib, яка інтегрована зі Streamlit для створення інтерактивних діаграм [24]. У stats_page у client.py видно використання plt.subplots для побудови стовпчикових графіків, що забезпечує наочність даних. Стилзація інтерфейсу реалізована через CSS у функції local_css, що дозволяє адаптувати зовнішній вигляд до потреб користувача (наприклад, теми читалки: світла, темна, сепія) [25].

Для стабільної роботи системи необхідне апаратне забезпечення, яке відповідає мінімальним характеристикам, описаним в таблиці 1.6.

Таблиця 1.6

Мінімальні характеристики апаратного забезпечення

№	Назва	Опис
1	Процесор	двоядерний процесор із частотою 2 ГГц або вище для обробки запитів і генерації рекомендацій
2	Оперативна пам'ять	щонайменше 4 ГБ для одночасної роботи серверної та клієнтської частин, а також обробки великих файлів книг
3	Дисковий простір	мінімум 10 ГБ для зберігання бази даних, файлів книг і обкладинок, з урахуванням можливості масштабування
4	Операційна система	Windows, Linux або macOS із підтримкою Python 3.8+ і необхідних бібліотек

Для розгортання на сервері рекомендується використовувати хмарні платформи (наприклад, AWS, Heroku), де ресурси можуть динамічно масштабуватися залежно від навантаження. Наприклад, обробка великих EPUB-

файлів або генерація рекомендацій через ШІ може потребувати додаткових обчислювальних ресурсів, таких як 8 ГБ RAM і SSD-диск для швидкого доступу до даних.

Обрані інструменти забезпечують баланс між простотою розробки, продуктивністю та зручністю для кінцевого користувача. Python із Streamlit і Flask дозволяє швидко створювати прототипи та розширювати функціонал, SQLite спрощує управління даними, а інтеграція з Gemini додає інтелектуальну складову. Водночас система залишається легкою для розгортання на персональних комп'ютерах, що робить її доступною для широкого кола користувачів – від любителів читання до бібліотекарів.

Таким чином, вибір інструментальних засобів для книжкового каталогізатора з системою рекомендацій на ШІ ґрунтується на їхній сумісності, ефективності та здатності вирішувати поставлені задачі. Вимоги до апаратного забезпечення є мінімальними для базового використання, але можуть бути адаптовані для більш масштабних сценаріїв експлуатації.

Висновок до першого розділу

Розділ присвячений аналізу задачі створення книжкового каталогізатора з системою рекомендацій на основі ШІ. У ході аналізу визначено мету проєкту – розробку зручного веб-додатку для управління бібліотекою та отримання персоналізованих рекомендацій. Проведено порівняння з аналогами (Goodreads, LibraryThing, Calibre, Book Catalogue), що виявило переваги розробленої системи, зокрема локалізацію українською мовою та інтеграцію ШІ. Обґрунтовано вибір клієнт-серверної архітектури з використанням Streamlit, Flask, SQLite та Gemini API, а також вибір інструментів, які забезпечують ефективність і масштабованість. Система відповідає потребам користувачів завдяки інтуїтивному інтерфейсу, підтримці різних форматів книг і детальній статистиці.

РОЗДІЛ 2. ПРОЄКТУВАННЯ КНИЖКОВОГО КАТАЛОГІЗАТОРУ З СИСТЕМОЮ РЕКОМЕНДАЦІЙ НА ШІ

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

Проектування додатку "Книжковий каталогізатор з системою рекомендацій на основі штучного інтелекту" передбачає чітке визначення його функціональних можливостей, архітектурних особливостей та взаємодії між компонентами. На основі реалізованих кодів (client.py та server.py) можна виділити ключові варіанти використання (use cases) та описати структуру системи, що забезпечує реалізацію поставлених завдань.

Варіанти використання додатку відображають основні сценарії взаємодії користувача з системою (рис. 2.1). Вони базуються на потребах цільової аудиторії — читачів, які прагнуть організувати свою бібліотеку, відстежувати прогрес читання та отримувати персоналізовані рекомендації.

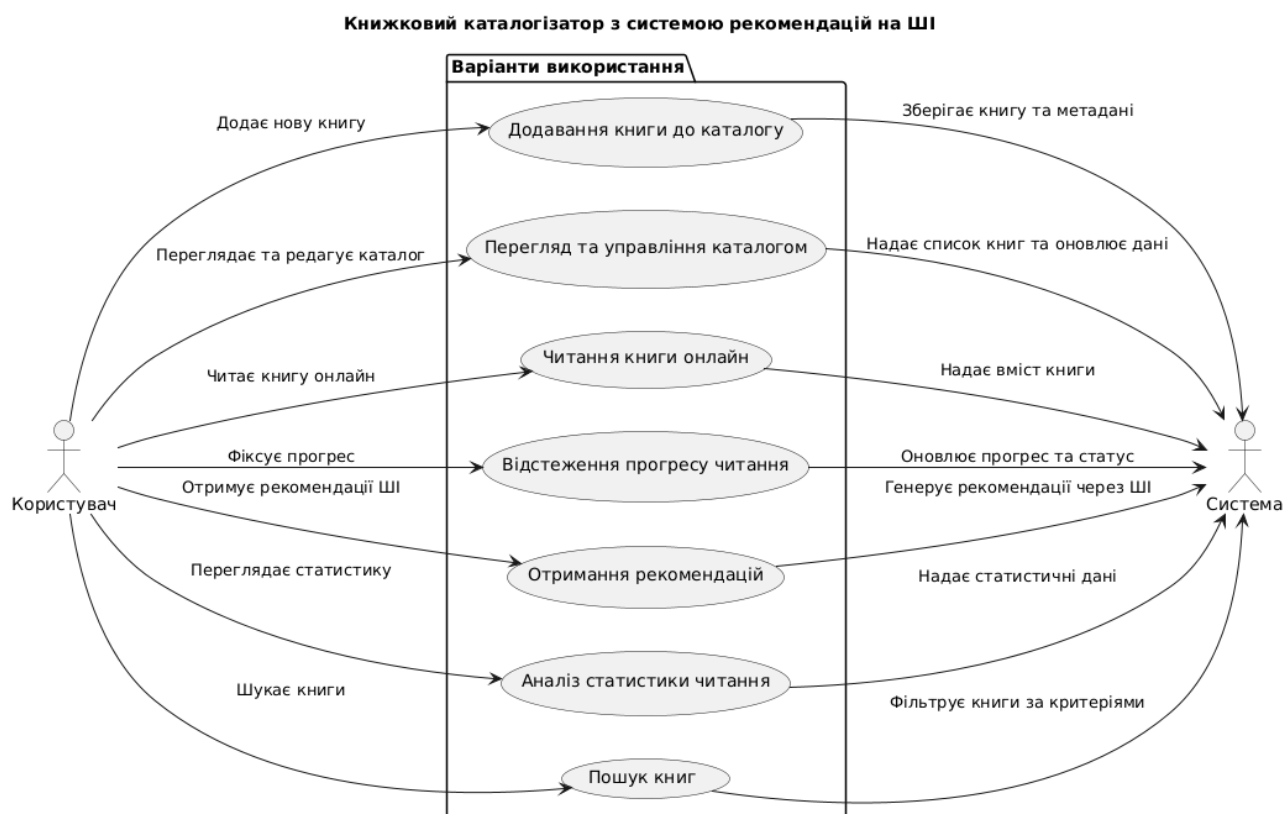


Рис. 2.1. Діаграма варіантів використання

Основні варіанти використання, ідентифіковані на основі функціоналу програми, описані в таблиці 2.1.

Таблиця 2.1

Основні варіанти використання

№	Назва	Опис
1	2	3
1	Додавання книги до каталогу	Користувач має можливість завантажити файл книги (підтримувані формати: EPUB, PDF, TXT, FB2, MOBI) та додати її до бібліотеки. Система автоматично витягує метадані (назва, автор, рік видання, жанр, обкладинка) з файлу, дозволяючи користувачу редагувати їх за потреби. Цей сценарій реалізовано через функцію <code>add_book()</code> у <code>client.py</code> та відповідний ендпоінт <code>/api/books</code> у <code>server.py</code> з методом POST
2	Перегляд та управління каталогом	Користувач може переглядати список усіх книг у вигляді карток із зображенням обкладинки, назвою, автором, роком видання та статусом читання. Функціонал включає фільтрацію за автором, жанром і статусом, що забезпечується функцією <code>search_books()</code> у <code>client.py</code> та ендпоінтом <code>/api/search</code> у <code>server.py</code> . Управління передбачає редагування (<code>update_book()</code>) та видалення книг (<code>delete_book()</code>)
3	Читання книги онлайн	Система дозволяє відкривати книгу в інтегрованій читалці з можливістю налаштування теми (світла, темна, сепія, зелена), шрифту та розміру тексту. Користувач може перегортати сторінки, додавати закладки (<code>add_bookmark()</code>) та зберігати цитати (<code>add_quote()</code>). Цей функціонал реалізовано в <code>reader_page()</code> у <code>client.py</code> та підтримується серверними ендпоінтами, такими як <code>/api/books/<book_id>/content</code>
4	Відстеження прогресу читання	Користувач може фіксувати поточну сторінку та час читання, а система автоматично розраховує відсоток прогресу та оновлює статус книги (не почато, читаю, прочитано). Реалізація здійснюється через <code>update_progress()</code> у <code>client.py</code> та ендпоінт <code>/api/books/<book_id>/progress</code> у <code>server.py</code>
5	Отримання рекомендацій на основі ШІ	Система пропонує персоналізовані рекомендації книг, використовуючи Gemini API від Google. Рекомендації можуть базуватися на конкретній книзі, тегах або текстовому описі вподобань користувача. Цей сценарій підтримується функцією <code>get_recommendations()</code> у <code>client.py</code> та серверним ендпоінтом <code>/api/recommendations</code> , який інтегрується з ШІ через <code>get_recommendations_from_ai()</code> у <code>server.py</code>

Продовження таблиці 2.1

1	2	3
6	Аналіз статистики читання	Користувач має доступ до статистичних даних: загальна кількість книг, розподіл за статусами, жанрами, авторами та роками, а також активність за останній місяць. Цей функціонал реалізовано в stats_page() у client.py та підтримується ендпоінтом /api/stats у server.py
7	Пошук книг	Користувач може шукати книги за назвою, описом, тегами, автором, жанром, роком або статусом. Реалізація забезпечується через search_page() у client.py та ендпоінт /api/search у server.py

Ці варіанти використання охоплюють основні потреби користувача, від організації бібліотеки до інтерактивного читання та аналізу читацьких звичок, що робить додаток комплексним інструментом для любителів книг.

Структура системи "Книжковий каталогізатор" побудована за клієнт-серверною архітектурою, що забезпечує розподіл функціоналу між інтерфейсом користувача (фронтенд) та серверною логікою (бекенд). Основні компоненти системи представлені на рис. 2.2 та описані в таблиці 2.2.

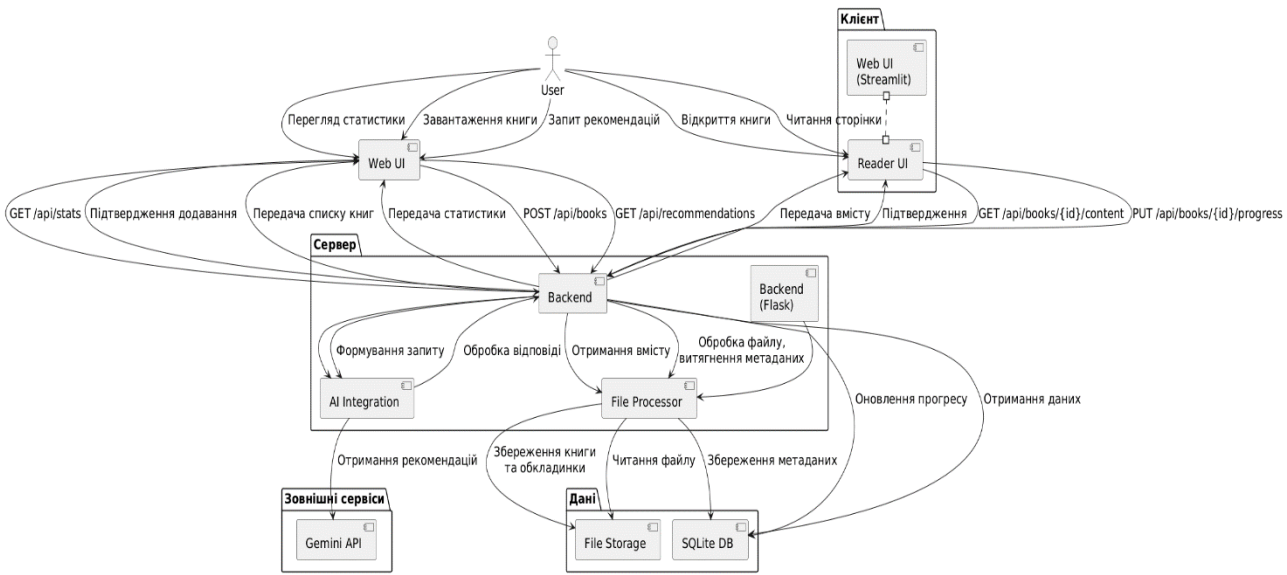


Рис. 2.2. Діаграма компонентів

Основні компоненти системи

№	Назва	Опис
1	2	3
1	Клієнтська частина (фронтенд)	
1.1	Технології	Streamlit — фреймворк для створення вебінтерфейсу на Python
1.2	Функціонал	Відображення інтерфейсу, обробка введення користувача, візуалізація даних (картки книг, графіки статистики, читалка)
1.3	Структура	: Код у client.py поділений на функції для кожної сторінки (наприклад, catalog_page(), reader_page()), що відповідають різним сценаріям використання. Використовується стан сесії (st.session_state) для збереження контексту (наприклад, поточної книги чи сторінки)
1.4	Інтерфейс	Інтуїтивний дизайн із бічною панеллю для навігації, картками книг, фільтрами та вкладками для додаткових функцій (закладки, цитати, статистика). CSS-стилі в local_css() забезпечують адаптивність і привабливий вигляд
2	Серверна частина (бекенд)	
2.1	Технології	Flask — легкий вебфреймворк для створення API, SQLite — база даних для зберігання інформації
2.2	Функціонал	Обробка запитів від клієнта, управління файлами (книги, обкладинки), інтеграція з ІШ для рекомендацій, витягнення метаданих із файлів
2.3	Структура	Код у server.py організований як набір RESTful ендпоінтів (наприклад, /api/books, /api/recommendations), кожен із яких відповідає за конкретну операцію. Логування через logging забезпечує моніторинг роботи сервера
2.4	База даних	SQLite зберігає таблиці books, bookmarks, quotes, reading_history, що пов'язані через зовнішні ключі для цілісності даних. Функція init_db() створює ці таблиці, а migrate_database() підтримує їх актуальність
3	Зовнішні сервіси	
3.1	Gemini API	Використовується для генерації рекомендацій на основі ІШ. Інтеграція реалізована через get_recommendations_from_ai() у server.py, що формує запит до моделі та обробляє відповідь у форматі JSON
3.2	Бібліотеки для обробки файлів	PyMuPDF (PDF), ebooklib (EPUB), BeautifulSoup (FB2) забезпечують витягнення метаданих і вмісту книг
4	Сховище даних	

Продовження таблиці 2.2

1	2	3
4.1	Файлова система	Книги зберігаються в директорії BOOKS_DIR, обкладинки — у COVERS_DIR. Унікальні імена файлів генеруються з використанням UUID для уникнення конфліктів
4.2	База даних	SQLite забезпечує швидкий доступ до метаданих і пов'язаних об'єктів (закладки, цитати, історія читання)

Система функціонує як єдиний механізм завдяки чіткій взаємодії між клієнтом і сервером через REST API. Наприклад, при додаванні книги користувач завантажує файл через інтерфейс Streamlit, який надсилає POST-запит до /api/books. Сервер обробляє файл, витягує метадані, зберігає його у файловій системі та записує інформацію в базу даних, повертаючи клієнту підтвердження. Аналогічно, для рекомендацій клієнт надсилає GET-запит до /api/recommendations, сервер звертається до Gemini API, обробляє результат і повертає список книг.

Така структура забезпечує модульність, масштабованість і легкість підтримки. Клієнтська частина зосереджена на зручності користувача, тоді як серверна — на обробці даних і інтеграції з ШІ, що відповідає сучасним підходам до розробки програмного забезпечення.

Визначені варіанти використання та структура системи відображають комплексний підхід до створення "Книжкового каталогізатора". Поєднання інтуїтивного інтерфейсу, потужного бекенду та інтеграції зі штучним інтелектом дозволяє реалізувати всі ключові функції — від управління бібліотекою до персоналізованих рекомендацій, що робить додаток цінним інструментом для читачів.

2.2. РОЗРОБКА СТРУКТУРИ ФАЙЛІВ СИСТЕМИ

Розробка структури файлів системи є ключовим етапом проєктування додатку «Книжковий каталогізатор з системою рекомендацій на основі штучного інтелекту». Організація файлів системи впливає на її масштабованість, зручність підтримки та ефективність взаємодії між клієнтською та серверною частинами. Важливо детально описати принципи створення структури файлів, враховуючи функціональні вимоги до додатку, а також особливості реалізації, представлені у реалізованому коді. Структура файлів розроблена з урахуванням поділу на клієнтську та серверну частини, що забезпечує чітке розмежування обов'язків і полегшує подальший розвиток системи.

Для забезпечення модульності та зрозумілості системи було обрано підхід, який базується на поділі проєкту на два основні компоненти: клієнтську частину (фронтенд) та серверну частину (бекенд). Клієнтська частина, реалізована за допомогою бібліотеки Streamlit, відповідає за інтерфейс користувача, відображення даних і взаємодію з сервером через API. Серверна частина, побудована на фреймворку Flask, забезпечує обробку запитів, управління базою даних, збереження файлів книг і обкладинок, а також інтеграцію з системою рекомендацій на основі штучного інтелекту (Gemini API). Такий поділ дозволяє ізолювати логіку обробки даних від їх представлення, що сприяє легшій модифікації окремих компонентів без впливу на інші частини системи.

Клієнтська частина представлена єдиним основним файлом `client.py`, який містить повну логіку інтерфейсу користувача та взаємодії з сервером. Структура цього файлу побудована за принципом функціонального поділу, що відображає основні модулі системи (таблиця 2.3).

Таблиця 2.3

Основні модулі системи

№	Назва	Опис
1	Налаштування та ініціалізація	У верхній частині файлу визначено імпорти бібліотек (Streamlit, Requests, Pandas тощо), базову URL-адресу сервера (BASE_URL), конфігурацію сторінки (за допомогою <code>st.set_page_config</code>), а також словники тем читалки (READER_THEMES) і шрифтів (FONTS). Це забезпечує централізоване управління конфігурацією інтерфейсу
2	API-функції	Набір функцій для взаємодії з сервером через HTTP-запити (наприклад, <code>get_books</code> , <code>add_book</code> , <code>update_progress</code>) ізольовано для забезпечення повторного використання коду та спрощення обробки помилок. Кожна функція відповідає окремому ендпоінту API, що сприяє чіткості структури
3	Стилі та дизайн	Функція <code>local_css</code> визначає власні CSS-стилі для елементів інтерфейсу (картки книг, контейнер читалки тощо), що дозволяє кастомізувати зовнішній вигляд без додаткових файлів
4	Логіка сторінок	Основна функція <code>main</code> координує відображення різних сторінок системи (каталог, читалка, пошук, статистика, рекомендації) залежно від стану сесії (<code>st.session_state</code>). Кожна сторінка реалізована як окрема функція (наприклад, <code>catalog_page</code> , <code>reader_page</code>), що полегшує підтримку та розширення функціоналу
5	Управління станом	Використання <code>st.session_state</code> для збереження даних між діями користувача (наприклад, поточна сторінка, ідентифікатор книги, налаштування читалки) забезпечує безперервність взаємодії без необхідності додаткових файлів конфігурації

Хоча клієнтська частина зосереджена в одному файлі, її внутрішня організація дозволяє легко виділити окремі модулі в майбутньому, якщо проєкт потребуватиме масштабування.

Серверна частина представлена файлом `server.py`, який відповідає за обробку запитів, управління файлами та базою даних, а також інтеграцію з AI для рекомендацій. Для забезпечення коректної роботи серверної частини створено додаткові директорії для зберігання файлів. Структура серверної частини описана в таблиці 2.4.

Структура серверної частини

№	Показники	Опис
1	Основний файл server.py	Цей файл містить повну логіку бекенду, включаючи налаштування Flask, ініціалізацію бази даних, обробку API-запитів і функції для роботи з файлами та метаданими. Поділ на секції (налаштування, ініціалізація БД, обробка файлів, ендпоінти API) забезпечує логічну організацію коду
2	Директорія static	Використовується для зберігання статичних файлів, таких як обкладинки книг. У середині static створено піддиректорії
2.1	covers	Зберігає файли обкладинок книг, які генеруються під час додавання книги або витягнення метаданих. Назви файлів формуються з урахуванням назви книги та унікального ідентифікатора для уникнення конфліктів
2.2	images	Містить статичні зображення, зокрема стандартну обкладинку (default_cover.png), яка використовується, якщо обкладинка книги відсутня
3	Директорія books	Призначена для зберігання файлів книг (формати EPUB, PDF, TXT, FB2 тощо). Кожен файл отримує унікальне ім'я, що включає ідентифікатор і безпечну версію назви книги, для забезпечення унікальності та безпеки
4	База даних bookcatalog.db	Файл SQLite, який зберігається в кореневій директорії проекту. База даних містить таблиці для книг (books), закладок (bookmarks), цитат (quotes) і історії читання (reading_history). Її розташування в корені спрощує доступ із серверного коду
5	Лог-файл app.log	Зберігається в кореневій директорії та використовується для запису подій сервера (інформаційних повідомлень, помилок тощо). Це полегшує діагностику та моніторинг роботи системи

При розробці структури файлів дотримано кількох ключових принципів такі як мінімалізм для спрощення розгортання та підтримки проєкту. Використано мінімальну кількість файлів і директорій. Наприклад, клієнтська частина зосереджена в одному файлі client.py, а серверна — в server.py, що зменшує складність управління проєктом на початковому етапі. Файли книг і обкладинок зберігаються в окремих директоріях (books і covers), що запобігає змішуванню різних типів даних і полегшує резервне копіювання.

Для файлів книг і обкладинок використовуються унікальні ідентифікатори (UUID) у поєднанні з безпечними назвами, що унеможлиблює конфлікти імен.

Наявність лог-файлу (app.log) забезпечує можливість відстеження подій і діагностики проблем без необхідності додаткових інструментів.

Гнучкість структури дозволяє легко додавати нові формати файлів або розширювати функціонал (наприклад, додавання нових ендпоінтів API чи сторінок у клієнтській частині) без значних змін у коді.

Структура файлів підтримує чітку взаємодію між клієнтом і сервером через REST API. Клієнт (client.py) надсилає запити до сервера (server.py) за допомогою бібліотеки Requests, використовуючи ендпоінти, визначені в server.py (наприклад, /api/books, /api/books/<id>/content). Сервер обробляє ці запити, звертаючись до бази даних (bookcatalog.db) або файлової системи (books, covers), і повертає відповідь у форматі JSON. Така організація забезпечує незалежність клієнтської частини від серверної, дозволяючи, наприклад, замінити Streamlit на інший фреймворк для інтерфейсу без змін у серверній логіці.

Поточна структура файлів є оптимальною для прототипу або невеликого додатку, але для масштабування системи можливі вдосконалення такі як поділ client.py на окремі модулі (наприклад, pages.py, api.py, styles.py) для кращої організації коду; виділення логіки обробки різних форматів книг у server.py в окремі модулі (наприклад, epub_parser.py, pdf_parser.py); використання системи управління конфігурацією (наприклад, .env-файлу) для зберігання констант, таких як BASE_URL чи API_KEY_GEMINI; Додавання директорії для тестів (tests), щоб забезпечити покриття коду тестами перед розгортанням.

Таким чином, розроблена структура файлів системи «Книжковий каталогізатор» є логічно організованою, відповідає функціональним вимогам і забезпечує зручність розробки та підтримки.

2.3. Розробка бази даних системи

Розробка бази даних є ключовим етапом у проектуванні додатку "Книжковий каталогізатор з системою рекомендацій на основі штучного інтелекту". База даних слугує основою для зберігання, управління та обробки інформації про книги, прогрес читання, закладки, цитати та історію активності користувача. Вадливим етапом проекту є детальний розгляд процесу створення структури бази даних, яка базується на аналізі функціональних вимог додатку та враховує його архітектурні особливості. Структура реалізується за допомогою реляційної моделі даних у SQLite, що забезпечує простоту інтеграції та достатню продуктивність для додатку такого типу (рис. 2.3).

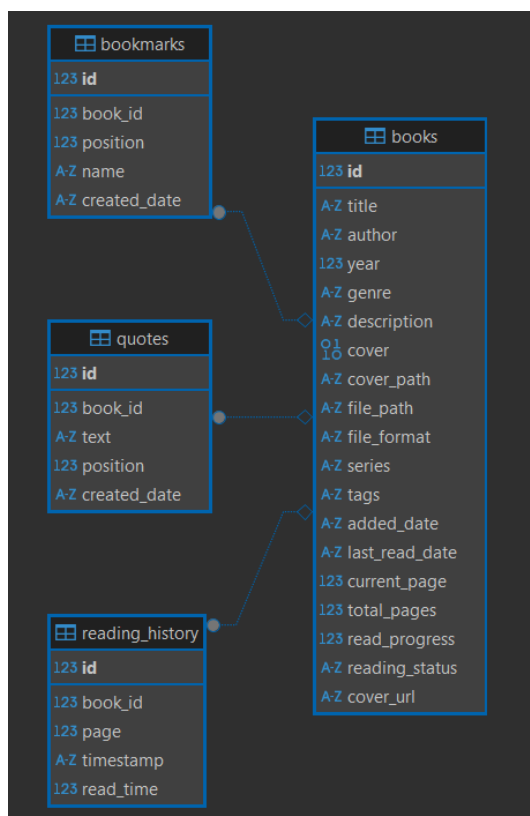


Рис. 2.3. Структура бази даних

Перед створенням структури бази даних було проведено аналіз функціональних потреб додатку. Основними вимогами стали: зберігання метаданих книг, управління файлами книг та їх обкладинками, відстеження прогресу читання з можливістю оновлення статусу та поточної сторінки, зберігання закладок і цитат, пов'язаних із конкретними книгами, ведення історії читання для аналізу активності користувача, забезпечення можливості пошуку та фільтрації книг за різними критеріями. Ці вимоги визначили набір сутностей та їх взаємозв'язки, які необхідно реалізувати у базі даних.

Для реалізації бази даних обрано SQLite – легку реляційну систему управління базами даних (СУБД), яка не потребує окремого серверного процесу та ідеально підходить для локальних додатків із середнім обсягом даних. SQLite підтримує стандарт SQL, що дозволяє створювати складні запити для пошуку, фільтрації та аналізу даних. Інтеграція з Python, який використовується у серверній частині додатку (файл `server.py`), забезпечується через бібліотеку `sqlite3`, що спрощує розробку та розгортання.

Структура бази даних складається з чотирьох основних таблиць, кожна з яких відповідає за певний показник функціональності додатку. Детальний опис кожної таблиці БД наведено в таблиці 2.5.

Таблиці БД

№	Таблиці/Атрибути	Опис
1	2	3
1	Таблиця books (Книги)	Ця таблиця є центральною у структурі бази даних і призначена для зберігання інформації про книги. Таблиця забезпечує зберігання як метаданих, так і інформації про стан читання, що дозволяє реалізувати функції каталогу та читалки
1.1	id (INTEGER, PRIMARY KEY, AUTOINCREMENT)	унікальний ідентифікатор книги
1.2	title (TEXT, NOT NULL)	назва книги, обов'язкове поле
1.3	author (TEXT)	автор книги
1.4	year (INTEGER)	рік видання
1.5	genre (TEXT)	жанр книги
1.6	description (TEXT)	опис книги
1.7	cover_path (TEXT)	шлях до файлу обкладинки на сервері
1.8	file_path (TEXT, NOT NULL)	шлях до файлу книги, обов'язкове поле
1.9	file_format (TEXT)	формат файлу (наприклад, EPUB, PDF)
1.10	series (TEXT)	серія, до якої належить книга
1.11	tags (TEXT)	теги для категоризації та пошуку
1.12	added_date (TEXT)	дата додавання книги у форматі ISO
1.13	last_read_date (TEXT)	дата останнього читання
1.14	current_page (INTEGER, DEFAULT 0)	поточна сторінка читання
1.15	total_pages (INTEGER, DEFAULT 0)	загальна кількість сторінок
1.16	read_progress (REAL, DEFAULT 0)	прогрес читання у відсотках
1.17	reading_status (TEXT, DEFAULT 'not_started')	статус читання ("not_started", "reading", "completed")

Продовження таблиці 2.5

2	Таблиця bookmarks (Закладки)	Призначена для зберігання закладок, які користувач додає під час читання. Зовнішній ключ book_id пов'язує закладки з книгами та забезпечує каскадне видалення при видаленні книги
2.1	id (INTEGER, PRIMARY KEY, AUTOINCREMENT)	унікальний ідентифікатор закладки
2.2	book_id (INTEGER, FOREIGN KEY)	посилання на книгу з таблиці books
2.3	position (INTEGER)	позиція (номер сторінки) закладки
2.4	name (TEXT)	назва закладки (наприклад, "Сторінка 15").
2.5	created_date (TEXT)	дата створення закладки у форматі ISO
3	Таблиця quotes (Цитати)	Використовується для збереження цитат, виділених користувачем. Як і у таблиці bookmarks, зовнішній ключ забезпечує зв'язок із книгою
3.1	id (INTEGER, PRIMARY KEY, AUTOINCREMENT)	унікальний ідентифікатор цитати
3.2	book_id (INTEGER, FOREIGN KEY)	посилання на книгу з таблиці books
3.3	text (TEXT)	текст цитати
3.4	position (INTEGER)	сторінка, на якій розташована цитата
3.5	created_date (TEXT)	дата додавання цитати
4	Таблиця reading_history (Історія читання)	Зберігає дані про активність читання для аналізу та статистики. Ця таблиця дозволяє відстежувати динаміку читання та будувати статистику
4.1	id (INTEGER, PRIMARY KEY, AUTOINCREMENT)	унікальний ідентифікатор запису
4.2	book_id (INTEGER, FOREIGN KEY)	посилання на книгу з таблиці books
4.3	page (INTEGER)	сторінка, на якій відбувалося читання
4.4	timestamp (TEXT)	час запису у форматі ISO
4.5	read_time (INTEGER)	тривалість читання в секундах

Усі допоміжні таблиці (bookmarks, quotes, reading_history) пов'язані з таблицею books через зовнішній ключ book_id. Тип зв'язку – "один до багатьох": одна книга може мати багато закладок, цитат і записів в історії читання. Використання зовнішніх ключів із правилом ON DELETE CASCADE гарантує,

що при видаленні книги автоматично видаляються всі пов'язані записи, забезпечуючи цілісність даних.

Для підвищення продуктивності запитів, особливо пошукових, передбачено можливість додавання індексів на часто використовувані поля, такі як title, author, genre у таблиці books. Хоча у реалізованому коді (server.py) індекси явно не створюються, їх додавання може бути реалізовано у майбутніх версіях для великих обсягів даних. SQLite автоматично створює індекси для первинних і зовнішніх ключів, що забезпечує швидкий доступ до записів за id і book_id.

Структура бази даних реалізується у функції init_db() у файлі server.py. Ця функція створює таблиці з відповідними полями та зв'язками при першому запуску додатку. Наприклад, створення таблиці books показано на рис. 2.4.

```
cursor.execute('''
CREATE TABLE IF NOT EXISTS books (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    title TEXT NOT NULL,
    author TEXT,
    year INTEGER,
    ...
)
''')
```

Рис. 2.4. Створення таблиці books

Подібним чином створюються інші таблиці з урахуванням їх специфіки. Використання IF NOT EXISTS запобігає помилкам при повторному запуску.

Для підтримки еволюції додатку передбачено механізм міграції бази даних (migrate_database() у server.py). Наприклад, додавання колонки cover_path для переходу від зберігання обкладинок у BLOB до файлової системи демонструє гнучкість структури. Це дозволяє адаптувати базу даних до нових вимог без втрати існуючих даних.

Отже, розроблена структура бази даних забезпечує надійне зберігання інформації, ефективну взаємодію між компонентами додатку та підтримку всіх

необхідних функцій – від каталогу до системи рекомендацій. Її реляційна природа та продумана організація сутностей сприяють зручності використання та подальшому масштабуванню проєкту.

2.4. Реалізація функціоналу додатку

Розробка додатку "Книжковий каталогізатор з системою рекомендацій на базі штучного інтелекту" передбачала створення комплексного рішення для управління бібліотекою електронних книг, їхнього читання та отримання персоналізованих рекомендацій. Основних функціональні компоненти додатку показані на рис. 2.5. Реалізація здійснювалася з урахуванням принципів модульності, зручності використання та інтеграції сучасних технологій ШІ.

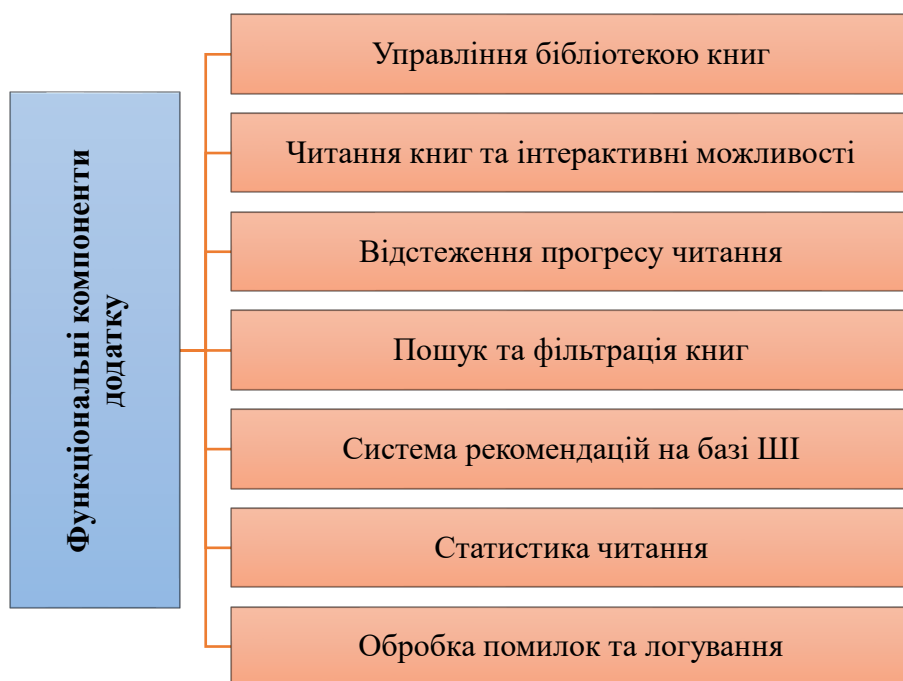


Рис. 2.5. Основні функціональні компоненти додатку

Основою функціоналу додатку є можливість додавання, редагування, видалення та перегляду книг. У файлі `client.py` реалізована клієнтська частина інтерфейсу з використанням бібліотеки Streamlit, яка забезпечує інтерактивний вебінтерфейс. Функція `add_book_page()` дозволяє користувачу завантажувати файли книг у форматах EPUB, PDF, TXT, FB2 або MOBI, витягувати метадані

(наприклад, назву, автора, рік видання) за допомогою функції `extract_metadata()` та зберігати їх на сервері через API-запит `add_book()`. На серверній стороні (`server.py`) ендпоінт `/api/books` обробляє POST-запити, зберігаючи файли у директорії `BOOKS_DIR` та записуючи метадані в базу даних SQLite (`bookcatalog.db`).

Редагування книг реалізовано через функцію `edit_book_page()` на клієнті та ендпоінт `/api/books/<book_id>` з методом PUT на сервері, що дозволяє оновлювати метадані та обкладинки. Видалення книг підтримується функцією `delete_book()` та відповідним DELETE-запитом, який також видаляє пов'язані файли з файлової системи. Перегляд бібліотеки доступний через сторінку каталогу (`catalog_page()`), де книги відображаються у вигляді карток із обкладинками, статусом читання та прогресом.

Функціонал читання реалізовано через сторінку `reader_page()` у `client.py`, яка забезпечує онлайн-перегляд вмісту книг. Користувач може налаштувати тему (світла, темна, сепія, зелена), шрифт і розмір тексту для комфортного читання. Вміст книги отримується через ендпоінт `/api/books/<book_id>/content`, який на сервері обробляє файли різних форматів (PDF, EPUB, TXT, FB2) за допомогою бібліотек `PyMuPDF`, `EbookLib` та `BeautifulSoup`. Наприклад, для EPUB-файлів текст розбивається на сторінки по 15 параграфів, що забезпечує зручну навігацію. Додаткові інтерактивні можливості включають збереження закладок (`add_bookmark()`) та цитат (`add_quote()`), які доступні через відповідні POST-запити до `/api/books/<book_id>/bookmarks` та `/api/books/<book_id>/quotes`. Ці дані зберігаються в таблицях `bookmarks` та `quotes` бази даних, дозволяючи користувачу швидко повертатися до важливих моментів книги.

Для відстеження прогресу читання реалізовано функцію `update_progress()` у `client.py`, яка викликає ендпоінт `/api/books/<book_id>/progress` на сервері. Прогрес обчислюється як відсоток прочитаних сторінок від загальної кількості (`current_page / total_pages * 100`) і оновлюється в базі даних разом із статусом читання ("не почато", "читаю", "прочитано"). Історія читання зберігається в

таблиці `reading_history`, де фіксуються сторінки, часові позначки та тривалість читання, що дозволяє аналізувати читацьку активність.

Функція пошуку реалізована через сторінку `search_page()` у клієнтській частині та ендпоінт `/api/search` на сервері. Користувач може шукати книги за назвою, автором, жанром, роком видання чи статусом читання. Запит до бази даних формується динамічно з урахуванням введених параметрів, використовуючи SQL з умовними операторами LIKE та точними збігами. Результати повертаються у вигляді списку книг із можливістю переходу до читання чи редагування.

Однією з ключових особливостей додатку є система рекомендацій, інтегрована через Google Gemini API. На клієнтській стороні сторінка `recommendations_page()` пропонує три типи рекомендацій: персональні, на основі вибраної книги та за тегами/вподобаннями. Серверна функція `get_recommendations()` (`/api/recommendations`) формує запит до ШІ, передаючи інформацію про книгу (назва, автор, жанр), теги чи текстові вподобання користувача. Gemini API повертає список рекомендованих книг у форматі JSON, який адаптується для відображення у фронтенді з додаванням тимчасових обкладинок через Placeholder API. Цей підхід забезпечує гнучкість і персоналізацію, хоча потребує стабільного інтернет-з'єднання та валідного API-ключа.

Функціонал статистики реалізований через `stats_page()` на клієнті та ендпоінт `/api/stats` на сервері. Користувач отримує інформацію про загальну кількість книг, розподіл за статусами, жанрами, авторами та роками видання. Дані агрегуються з бази даних за допомогою SQL-запитів із функціями COUNT, GROUP BY та SUM, а візуалізація здійснюється через бібліотеку Matplotlib у вигляді стовпчикових діаграм. Додатково відображається активність за останній місяць, включаючи години читання та кількість завершених книг.

Для підвищення надійності додатку на сервері впроваджено систему логування через модуль logging. Помилки, такі як відсутність файлів, невдалі запити до API чи проблеми з базою даних, записуються у файл `app.log` із

зазначенням часу та деталей. На клієнтській стороні помилки відображаються користувачу через функцію `st.error()`, забезпечуючи зворотний зв'язок без переривання роботи додатку.

Таким чином, реалізація функціоналу додатку охоплює повний цикл роботи з бібліотекою книг – від завантаження та каталогізації до читання, аналізу та отримання рекомендацій. Інтеграція ІІІ через Gemini API додає інноваційності, а використання SQLite як бази даних забезпечує легкість розгортання. Водночас модульна структура коду дозволяє легко розширювати функціонал, наприклад, додаванням нових форматів книг чи вдосконаленням алгоритмів рекомендацій.

Висновок до другого розділу

Розділ присвячений проектуванню додатку "Книжковий каталогізатор з системою рекомендацій на основі ІІІ" і охоплює визначення варіантів використання, розробку структури системи, файлів і бази даних, а також реалізацію функціоналу. Розроблена клієнт-серверна архітектура з інтеграцією ІІІ через Gemini API забезпечує зручне управління бібліотекою, читання книг і персоналізовані рекомендації. Структура бази даних SQLite і файлова організація сприяють ефективності та масштабованості, а модульний код спрощує подальший розвиток. Проєкт поєднує інтуїтивний інтерфейс із потужною серверною логікою, відповідаючи потребам сучасних читачів. Реалізований функціонал забезпечує можливість перегляду списку книг, відстеження останньої активності читання, показуючи до трьох останніх книг з їх назвами, авторами та розрахованим часом читання в годинах. Користувачі можуть додавати та керувати закладками, де час читання розраховується як різниця між поточним часом та часом початку читання сесії.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ

3.1. ПОРЯДОК ВСТАНОВЛЕННЯ ТА НАЛАШТУВАННЯ СИСТЕМИ

Розроблений проєкт "Книжковий каталогізатор з системою рекомендацій на основі штучного інтелекту" є комплексним програмним рішенням, що складається з клієнтської частини (`client.py`), побудованої на базі Streamlit, та серверної частини (`server.py`), реалізованої з використанням Flask. Для успішного запуску та функціонування системи необхідно виконати низку етапів встановлення та налаштування. Ці кроки враховують особливості обох компонентів програми, їх залежності, а також інтеграцію з зовнішніми сервісами, такими як Gemini API для рекомендацій. Процес описано з урахуванням наукового підходу, але подано в доступній формі для зручності користувачів.

Підготовка робочого середовища: перед початком встановлення необхідно переконатися, що на комп'ютері користувача встановлено базові інструменти для роботи з Python-проєктами. Основні вимоги включають: операційна система: Windows, macOS або Linux, Python: версія 3.8 або новіша. Рекомендується перевірити версію командою `python --version` у терміналі. У разі відсутності Python його можна завантажити з офіційного сайту (python.org) та встановити, додавши до системної змінної PATH. Цей етап є фундаментом для подальшої роботи, адже правильна конфігурація середовища забезпечує коректне виконання всіх наступних дій.

Для завантаження вихідного коду користувачу необхідно отримати файли `client.py` та `server.py`. Це можна зробити двома способами. Скопіювати вручну текст програм із реалізованого лістингу та зберегти їх у файли з відповідними назвами (`client.py` та `server.py`) у робочій директорії, наприклад, `book-catalog`.

Завантажити з репозиторію якщо код розміщено в Git-репозиторії, виконайте команду `git clone <URL репозиторію>` у терміналі, де `<URL репозиторію>` замінюється на адресу проєкту. Після цього код буде доступний у локальній папці.

Рекомендується створити окрему директорію для проєкту, наприклад, за допомогою команди `mkdir book-catalog` і перейти до неї командою `cd book-catalog`, щоб організувати робочий простір.

Для роботи програми необхідно встановити бібліотеки, які використовуються в коді. Залежності для клієнтської та серверної частин частково перетинаються, але їх можна встановити одним списком. У терміналі, перебуваючи в директорії проєкту, виконайте команду, представлену на рис. 3.1.

```
pip install streamlit requests pandas pillow matplotlib ebooklib beautifulsoup4
PyMuPDF flask sqlite3 google-generativeai
```

Рис. 3.1. Команда встановлення залежностей

Існують такі основні бібліотеки як, `streamlit` для створення інтерактивного вебінтерфейсу клієнта, `flask` для реалізації серверної логіки, `requests` для обробки HTTP-запитів між клієнтом і сервером, `pandas` і `matplotlib` для аналізу та візуалізації статистики читання, `ebooklib`, `beautifulsoup4`, `PyMuPDF` для обробки різних форматів книг, `pillow` для роботи із зображеннями обкладинок.

У разі виникнення помилок під час встановлення (наприклад, через несумісність версій), можна створити файл `requirements.txt` із переліком бібліотек та їх версій і встановити їх командою `pip install -r requirements.txt`.

Налаштування API ключа для рекомендацій базується на Gemini API, тому для її роботи потрібен API ключ. Користувачу необхідно зареєструватися на платформі Google Cloud (або іншій, що надає доступ до Gemini API) і згенерувати ключ. Потім відкрити файл server.py у текстовому редакторі та знайти рядок (рис. 3.2):

```
API_KEY_GEMINI = "YOUR_GEMINI_API_KEY"
```

Рис. 3.2. Рядок ключа Gemini

Замінити "YOUR_GEMINI_API_KEY" на отриманий ключ, (рис. 3.3).

```
API_KEY_GEMINI = "AIzaSyBkHBZ5ErdhYAB_YdjGFYwtw7YgMLDKnUc"
```

Рис. 3.3. Заміна ключа на отриманий

Зберегти зміни у файлі. Без цього ключа функція рекомендацій не працюватиме, хоча основний функціонал каталогу залишиться доступним.

Запуск серверної частини: серверна частина відповідає за обробку запитів, збереження даних у базі SQLite та інтеграцію з AI. Для її запуску: У терміналі, перебуваючи в директорії проєкту, виконайте команду з рис. 3.4.

```
python server.py
```

Рис. 3.4. Запуск скрипту серверу

Після запуску сервер ініціалізує базу даних (bookcatalog.db), створить директорії для книг (books) та обкладинок (static/covers), а також стандартну обкладинку, якщо її немає. У консолі з'явиться повідомлення про успішний запуск, наприклад рис. 3.5.

```
* Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
```

Рис. 3.5. Повідомлення про успішний запуск

Переконайтеся, що сервер працює на порту 5000 (за замовчуванням). Якщо порт зайнятий, його можна змінити, відредагувавши рядок `app.run(debug=True, host='0.0.0.0', port=5000)` у `server.py`.

Запуск клієнтської частини забезпечує графічний інтерфейс користувача. Для її запуску відкрийте новий термінал (залишивши серверний активним) і перейдіть до директорії проєкту та виконайте команду рис. 3.6.

```
streamlit run client.py
```

Рис. 3.6. Запуск скрипту клієнт

Після цього автоматично відкриється веббраузер із адресою `http://localhost:8501`, де відобразиться інтерфейс програми. Якщо браузер не відкрився, введіть цю адресу вручну.

Клієнт зв'язується із сервером за адресою `http://localhost:5000/api` (змінна `BASE_URL` у `client.py`), тому сервер має бути запущений заздалегідь. Після запуску обох компонентів користувач має перейти до бічної панелі в інтерфейсі Streamlit і перевірити доступність пунктів меню ("Мій каталог", "Пошук" тощо). Додати тестову книгу через кнопку

"Додати нову книгу", завантаживши файл (наприклад, EPUB або PDF), і переконатися, що вона з'являється в каталозі.

Якщо рекомендації не працюють, перевірити правильність API ключа та лог-файл (app.log), який створюється сервером для діагностики. Для підвищення зручності чи адаптації системи можна змінити порт сервера або клієнта у відповідних файлах, якщо стандартні порти (5000 для Flask, 8501 для Streamlit) зайняті. Налаштувати автоматичний запуск через скрипт (наприклад, bash-скрипт для Linux або bat-файл для Windows). Розгорнути систему на віддаленому сервері, змінивши BASE_URL у client.py на публічну IP-адресу або домен.

Таким чином, процес встановлення та налаштування системи є логічно структурованим і доступним навіть для користувачів із базовими технічними навичками. Завдяки модульній архітектурі проєкту (клієнт-сервер) та докладним логам у app.log, будь-які проблеми можна швидко діагностувати та усунути. Успішне виконання всіх етапів гарантує готовність "Книжкового каталогізатора" до роботи, дозволяючи користувачу ефективно управляти бібліотекою та отримувати персоналізовані рекомендації.

3.2. СТРУКТУРА ІНТЕРФЕЙСУ

Розроблений додаток "Книжковий каталогізатор з системою рекомендацій на основі штучного інтелекту" має продуманий і зручний інтерфейс, який забезпечує користувачам інтуїтивно зрозумілу взаємодію з системою. Інтерфейс створено з використанням бібліотеки Streamlit для клієнтської частини (client.py) та Flask для серверної (server.py), що дозволяє поєднати гнучкість веб-додатків із простотою використання. Структура інтерфейсу базується на модульному підході, де кожна сторінка відповідає за конкретний функціонал, а бічна панель забезпечує швидкий доступ до основних розділів. Важливо детально розглянути структуру інтерфейсу та порядок роботи з системою, з акцентом на користувацький досвід і логіку взаємодії.

Інтерфейс додатку складається з двох основних компонентів: бічної панелі (sidebar) та основної робочої області. Бічна панель (рис. 3.7) слугує навігаційним меню, яке дозволяє користувачу швидко переходити між ключовими розділами: "Мій каталог", "Пошук", "Статистика", "Рекомендації" та "Додати нову книгу". Кожен пункт меню реалізовано як кнопка, при натисканні на яку змінюється стан сесії (st.session_state.current_page), що викликає відображення відповідної сторінки в основній області.

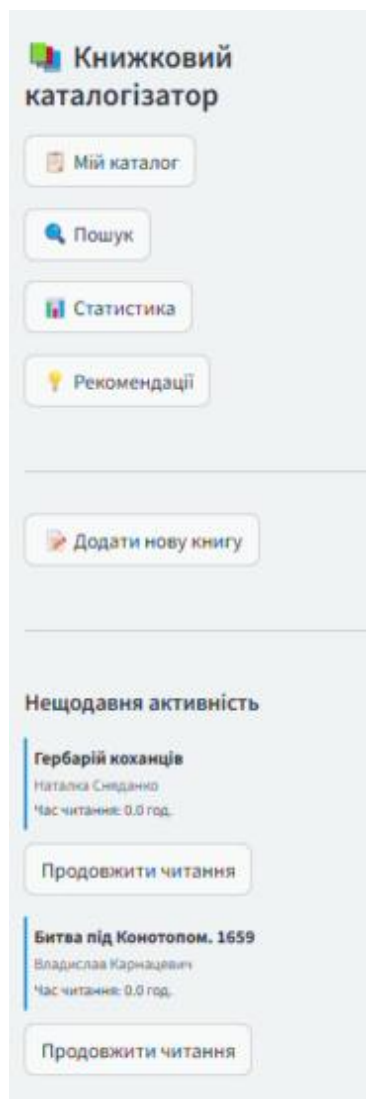


Рис. 3.7 – Бічна панель

Основна робоча область (рис. 3.8) є динамічною і відображає вміст залежно від обраного розділу. Її дизайн адаптивний, виконаний у форматі "wide" (широкий), що оптимізує використання простору екрана. Для підвищення зручності застосовуються власні стилі CSS (функція `local_css()`), які визначають зовнішній вигляд елементів, таких як картки книг, прогрес-бар, кнопки дій тощо. Наприклад, картки книг (`.book-card`) мають ефект підйому при наведенні курсору, що додає інтерактивності, а читацький контейнер (`.reader-container`) підтримує прокручування вмісту та адаптацію до обраної теми.

Мій книжковий каталог



Рис. 3.8 – Приклад основної робочої області

Сторінка "Мій каталог" (catalog_page) (рис. 3.9) – є основним розділом для перегляду бібліотеки користувача. У верхній частині розташована заголовна назва "Мій книжковий каталог", а нижче — розширюваний блок "Фільтри" з полями для введення автора, жанру та вибору статусу (наприклад, "Читаю", "Прочитано"). Книги відображаються у вигляді сітки з чотирьох колонок, де кожна картка містить обкладинку, назву, автора, рік видання, прогрес-бар і кнопки дій ("Читати", "Редагувати", "Видалити"). Такий підхід дозволяє швидко оцінити стан бібліотеки та розпочати взаємодію з книгою.

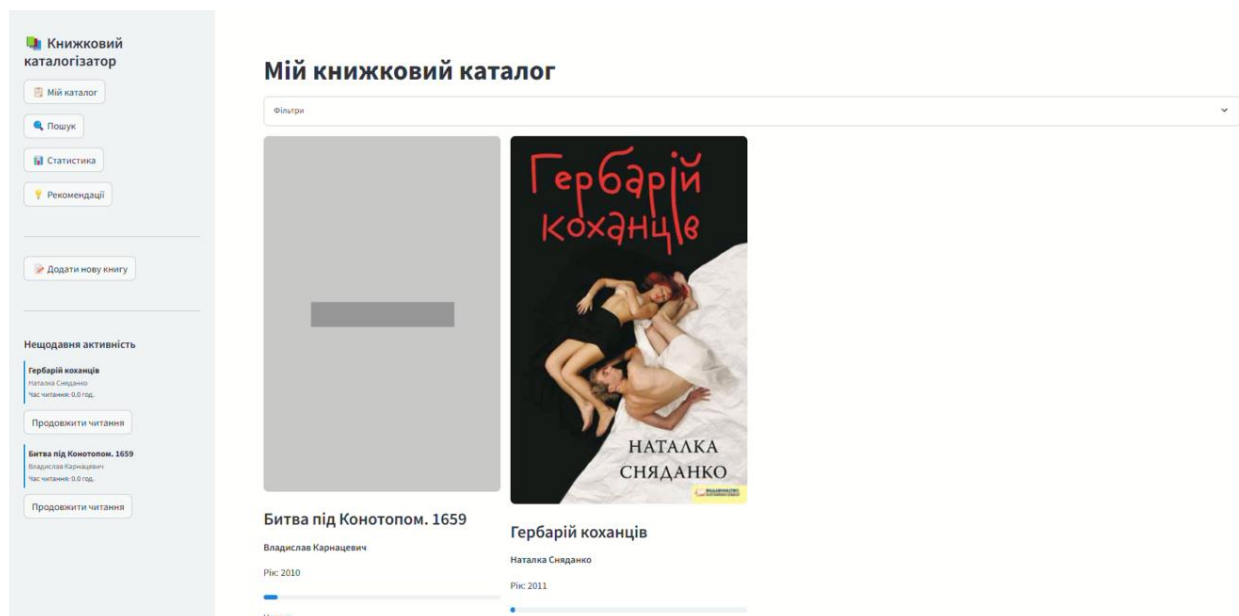


Рис. 3.9. Сторінка "Мій каталог"

Сторінка "Додати нову книгу" (add_book_page) (рис. 3.10): інтерфейс цього розділу призначений для введення нової книги в каталог. Користувач завантажує файл книги (підтримуються формати EPUB, PDF, TXT, FB2, MOBI) через елемент file_uploader. Після завантаження доступна кнопка "Витягнути дані з файлу", яка викликає аналіз метаданих (функція extract_metadata()) і заповнює поля, такі як назва, автор, рік, жанр і опис. Додатково передбачено завантаження обкладинки або введення її URL. Кнопка "Додати книгу" завершує процес, відправляючи дані на сервер через POST-запит.

Додати нову книгу

Виберіть файл книги

Drag and drop file here
Limit 200MB per file • EPUB, PDF, TXT, FB2, MOBI

Browse files

dim.epub 0.6MB

×

Витягнути дані з файлу

Назва книги*

Опис

Автор

Рік видання*

Серія

Жанр

Теги (розділені комами)

Обкладинка книги

Завантажити обкладинку

Drag and drop file here
Limit 200MB per file • JPG, JPEG, PNG

Browse files

АБО вкажіть URL обкладинки (не обов'язково)

Додати книгу

Рис. 3.10. Додавання нової книги

Сторінка "Читання книги" (reader_page) (рис. 3.11) – розділ призначений для онлайн-читання. У верхній частині відображається назва книги та кнопка повернення до каталогу. Нижче розташовані елементи налаштування: вибір теми (світла, темна, сепія, зелена), шрифту (Arial, Times New Roman тощо) та розміру шрифту (12–24 пікселі). Текст книги виводиться в контейнері з прокручуванням, стиль якого адаптується до обраної теми. Навігація між сторінками здійснюється кнопками "Попередня" та "Наступна" (рис. 3.12), а прогрес зберігається автоматично (update_progress()). Додатково доступні вкладки "Закладки", "Цитати" та "Деталі книги" для роботи з анотаціями та завантаження файлу.

Читання: Тістечка з ягодами

[← Повернутися до каталогу](#)

Тема

Світла

Шрифт

Arial

Розмір шрифту

12

16

24

Тістечка з ягодами

Ізабелла Сова

Рік: 2005

Не почато

[Читати](#)[Редагувати](#)[Видалити](#)

ПЕРЕДОСТАННІЙ ДЕНЬ ЧЕРВНЯ

ПЕРЕДОСТАННІЙ ДЕНЬ ЧЕРВНЯ

— Ми мусимо поговорити про купу. Через годину в його кабінеті.

— І що буде?

— Піду, — низила я плечима. — Все з'ясує. Якщо вдасться, то повернуся до комп'ютера й завершу звіг.

— А якщо не вдасться?

— Тоді клацну кілька знімків, кактус під пахву, іграшки до коробки і...

— Я це передчувала! — Магда нервово прикусила ніготь, оздоблений діамантиками. — Я від початку казала, аби ми не йшли до тієї клятої ворожки.

Клята ворожка

Вона все сплутала в мому впорядкованому житті одного холодного травневого вечора, зразу після «Новин». За півгодини до того ми закінчили інтеграційні заняття в одному із сонних містечок Підкарпаття. Вимучені дитсадівськими розвагами, які мусили суттєво посилити в нас відчуття групових сув'язей, ми дочовгали до найближчої кнайпи. Офіцантка, сильно похляпана лискучим лаком брюнетка, принесла нам теплувате пиво й пачку гіржавого арахісу. Ми вицмулили по одній гальбі, по другій, по третій. У повній мовчанці, бо про що тут балакати, коли на заняттях ми вибовкали одне одному ледь чи не все, окрім, звісно, розміру зарплати.

Рис. 3.11. Сторінка читання

[← Попередня сторінка](#)

Сторінка 1 з 232

[→ Наступна сторінка](#)

Назва закладки

Сторінка 1

[Додати закладку](#)

Закладки

Цитати

Деталі книги

Закладок поки немає

Рис. 3.12. Навігація між сторінками

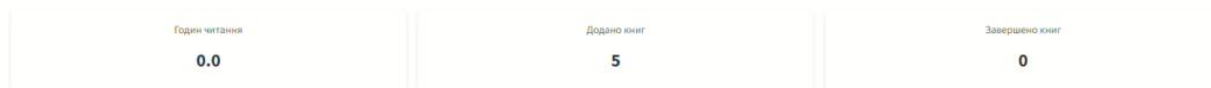
Сторінка "Пошук" (search_page) включає текстове поле для введення запиту та розширюваний блок із додатковими фільтрами (автор, жанр, рік, статус). Результати відображаються у вигляді сітки, аналогічної каталогу, що забезпечує консистентність дизайну. Кнопка "Пошук" активує запит до API (/api/search), повертаючи відфільтровані книги.

Сторінка "Статистика" (stats_page) (рис. 3.13) - розділ візуалізує статистику читання. У верхній частині — чотири картки з основними показниками (загальна кількість книг, прочитані, у процесі, не розпочаті), нижче — статистика за місяць (години читання, додані книги, завершені). Вкладки "Жанри", "Автори", "Роки" містять графіки, побудовані за допомогою Matplotlib, що додають наочності.

Статистика читання



Активність за останній місяць



Статистика за категоріями

Жанри **Автори** Роки

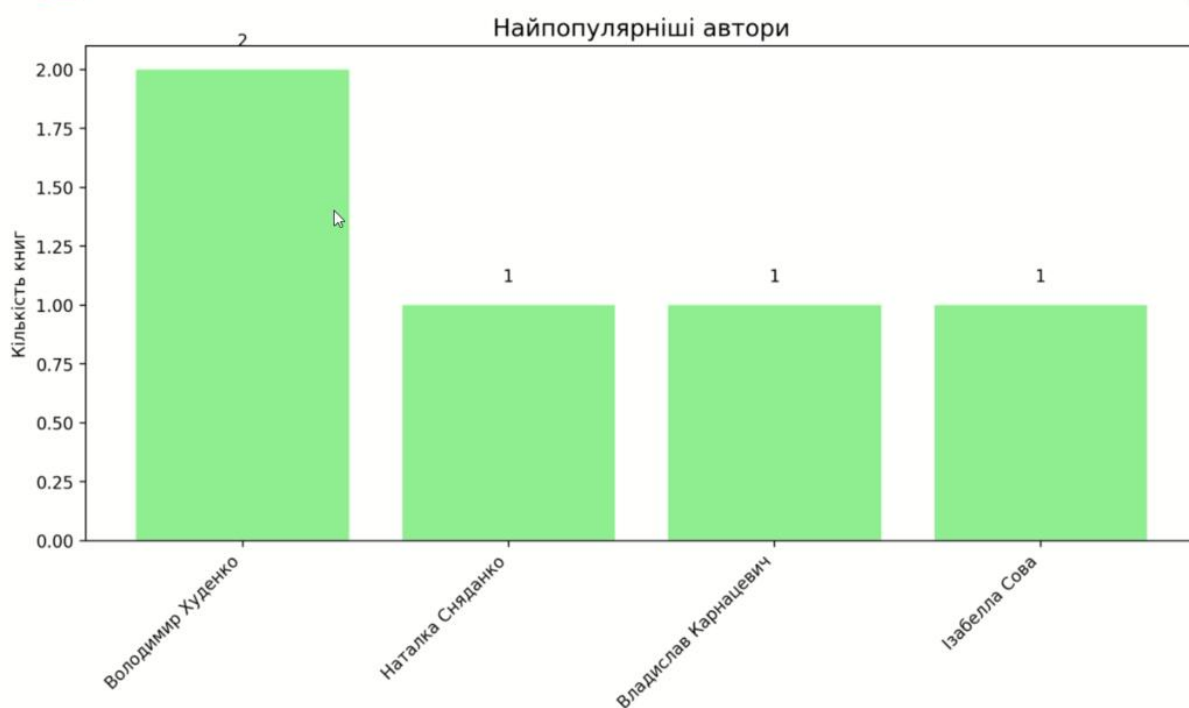


Рис. 3.13. Статистика

Сторінка "Рекомендації" (recommendations_page) розділена на три вкладки такі як "Для вас" (персональні) (рис. 3.14), "На основі книги" (за обраною книгою) (рис. 3.15) та "За тегами" (за введеними тегами чи вподобаннями) (рис. 3.16). Кожна вкладка має кнопку для генерації рекомендацій, які відображаються у вигляді карток книг із placeholder-обкладинками (рис. 3.17), отриманими через Gemini API.

Рекомендації для читання

Для вас На основі книги За тегами

Персональні рекомендації

Отримати персональні рекомендації

Аналізуємо ваші вподобання...

Рис. 3.14. Пошук персональних рекомендацій

Рекомендації для читання

Для вас На основі книги За тегами

Рекомендації на основі книги

Виберіть книгу

Дим (Володимир Худенко)

Отримати рекомендації

Шукаємо схожі книги...

Рис. 3.15. Пошук рекомендацій на основі книги

Рекомендації для читання

Для вас На основі книги За тегами

Рекомендації за тегами та вподобаннями

Теги (розділені комами)

драма

Опишіть, що вам подобається

Про собак, сумне

Знайти книги за вподобаннями

Шукаємо книги за вашими вподобаннями...

Рис. 3.16. Пошук рекомендацій за тегами та вподобаннями

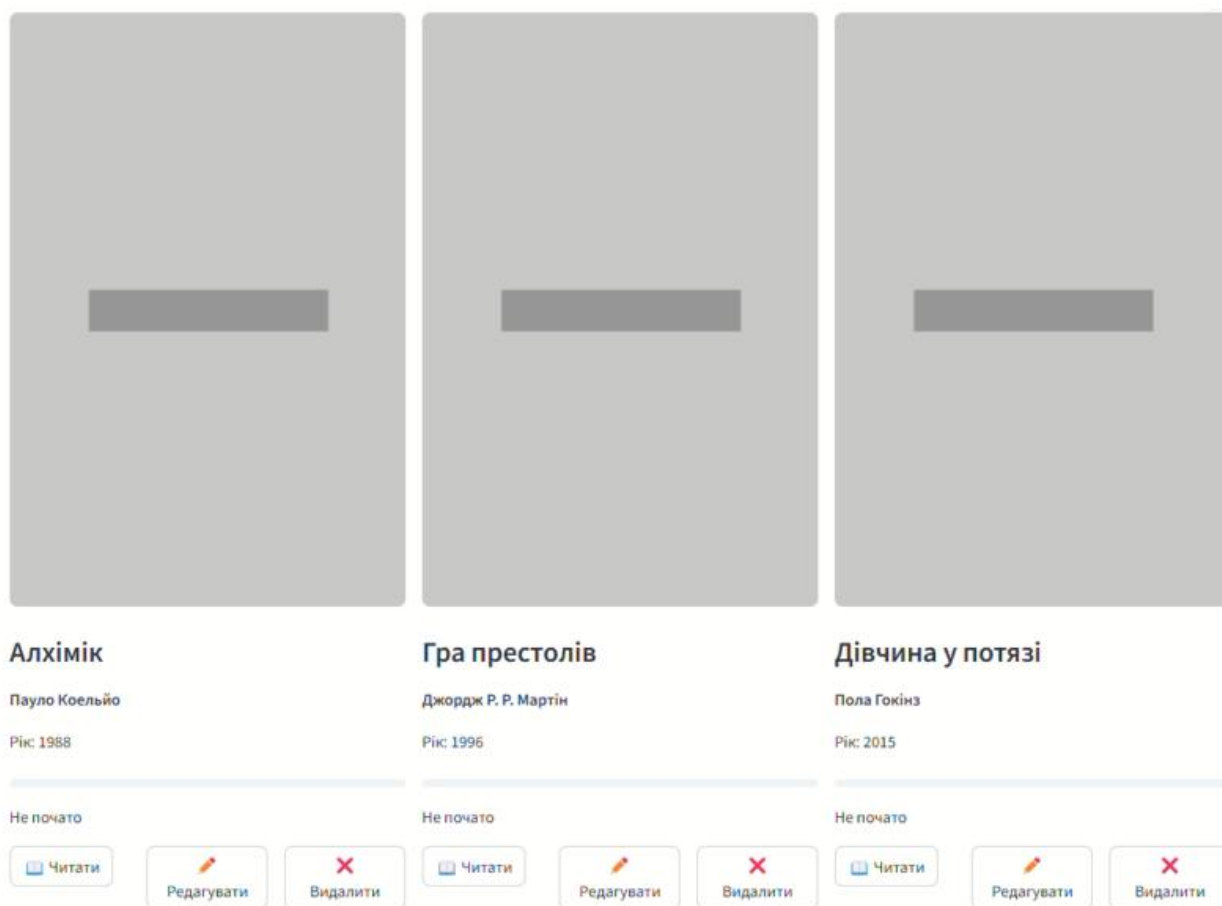


Рис. 3.17. Відображення рекомендацій

Користувач запускає client.py через Streamlit (команда streamlit run client.py) і server.py через Flask (python server.py). Сервер працює на localhost:5000, обробляючи API-запити, а клієнтська частина доступна через браузер.

На бічній панелі користувач обирає потрібний розділ. Наприклад, для додавання книги натискається "Додати нову книгу", після чого відкривається відповідна сторінка. Недавня активність (останні прочитані книги) відображається внизу бічної панелі, дозволяючи швидко повернутися до читання.

На сторінці "Додати нову книгу" користувач завантажує файл, витягує метадані (якщо потрібно) і заповнює поля. Після збереження книга з'являється в каталозі. Для редагування обирається книга в каталозі, натискається "Редагувати", і вносяться зміни.

Вибравши книгу та натиснувши "Читати", користувач переходить до reader_page, де налаштовує вигляд тексту, гортає сторінки, додає закладки чи цитати (виділивши текст і натиснувши "Зберегти як цитату"). Прогрес зберігається автоматично при зміні сторінки або поверненні до каталогу.

На сторінці пошуку вводиться запит, застосовуються фільтри, і переглядаються результати. У "Статистиці" користувач аналізує свої читацькі звички, а в "Рекомендаціях" отримує пропозиції від ШІ, вказавши теги чи вподобання.

Інтерфейс підтримує адаптивність до дій користувача: стан сесії зберігає поточну сторінку, книгу чи прогрес читання. Використання асинхронних запитів до сервера (наприклад, через requests) забезпечує швидке оновлення даних. Система також інформує про помилки (st.error) і успіх (st.success), що полегшує взаємодію.

Таким чином, структура інтерфейсу "Книжкового каталогізатора" є логічно організованою, з чітким розподілом функцій між сторінками та інтуїтивним порядком роботи, що робить його зручним інструментом для управління бібліотекою та читання.

3.3. ТЕСТУВАННЯ РОБОТИ СИСТЕМИ

Тестування роботи системи "Книжковий каталогізатор з системою рекомендацій на основі штучного інтелекту" є ключовим етапом реалізації проекту, що дозволяє оцінити її функціональність, стабільність та відповідність поставленим вимогам. У процесі тестування було виконано низку перевірок, які охоплюють основні модулі додатку: каталогізацію книг, онлайн-читалку, систему рекомендацій, а також взаємодію клієнтської та серверної частин. Детальний опис тестування наведений в таблиці 3.1.

Таблиця 3.1

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						57

Опис тестування

№	Напрямок тестування	Опис
1	2	3
1	Перевірка функціональності додавання книги	Першим етапом тестування стала оцінка процесу додавання нової книги до каталогу. Для цього було використано файли різних форматів (EPUB, PDF, TXT, FB2), які завантажувалися через інтерфейс клієнтської частини (client.py). У ході тестування перевірялася коректність витягнення метаданих (назва, автор, рік видання, жанр, опис) із файлів за допомогою функції <code>extract_metadata</code> у <code>server.py</code> . Наприклад, при завантаженні EPUB-файлу система успішно розпізнала метадані, включаючи обкладинку, що відобразилася в інтерфейсі. У випадку PDF-файлу без метаданих користувачу пропонувалося вручну заповнити поля, що свідчить про гнучкість системи. Помилки, такі як завантаження файлу без назви або некоректного формату, оброблялися належним чином із виведенням відповідних повідомлень (наприклад, “Файл книги не надіслано”), що підтверджує стабільність обробки винятків.
2	Тестування відображення каталогу книг	Наступним кроком було тестування функції відображення каталогу (<code>catalog_page</code> у <code>client.py</code>). Після додавання кількох книг (наприклад, 5 одиниць із різними статусами: “Не почато”, “Читаю”, “Прочитано”) перевірялася коректність їх представлення у вигляді карток із обкладинками, прогресом читання та кнопками дій (“Читати”, “Редагувати”, “Видалити”). Усі книги відобразилися у сітці з чотирма колонками, а при наведенні курсору на картку спрацьовував ефект підйому (<code>transform: translateY(-5px)</code> у CSS), що підвищує зручність взаємодії. Фільтрація за автором, жанром та статусом також працювала коректно: при введенні “фантастика” у поле жанру відображалися лише відповідні книги. Єдиною заувагою стало незначне сповільнення при завантаженні великої кількості книг (понад 50), що може потребувати оптимізації в майбутньому.

Продовження таблиці 3.1

1	2	3
3	Оцінка роботи онлайн-читалки	Функціональність онлайн-читалки (reader_page) тестувалася шляхом відкриття книг у різних форматах. Наприклад, при виборі EPUB-файлу вміст розбивався на сторінки (по 15 параграфів на сторінку), а для PDF використовувався текстовий вміст кожної сторінки. Перевірка показала, що перемикання сторінок за допомогою кнопок “Попередня” та “Наступна” відбувалося без затримок, а прогрес читання зберігався через виклик update_progress. Налаштування теми та шрифту застосовувалися миттєво, що забезпечило комфортне читання. Додавання закладок і цитат також протестовано: при виділенні тексту з’являлася опція збереження цитати, яка коректно записувалася до бази даних. Невелике зауваження стосувалося обробки великих TXT-файлів, де розбиття на сторінки іноді було нерівномірним через різну довжину рядків
4	Тестування системи рекомендацій на основі ШІ	Система рекомендацій, реалізована через Gemini API, перевірялася у трьох сценаріях: персональні рекомендації, рекомендації на основі книги та за тегами. У першому випадку, без достатньої кількості прочитаних книг, система повертала повідомлення “Недостатньо даних для рекомендацій”, що є логічним для початкового етапу використання. При виборі конкретної книги (наприклад, “Дюна” Френка Герберта) API повернув три схожі книги з жанру фантастики, що свідчить про коректну обробку метаданих. Введення тегів “фантастика, пригоди” дало аналогічно релевантні результати. Проте в одному з тестів із некоректним API-ключем система повернула порожній список, що підкреслює залежність від зовнішнього сервісу та потребу в резервному механізмі рекомендацій
5	Перевірка статистики читання	Модуль статистики (stats_page) тестувався шляхом аналізу відображення даних: загальної кількості книг, розподілу за статусами, жанрами, авторами та роками. Після додавання 10 книг із різними параметрами система коректно відобразила 4 завершені, 3 у процесі та 3 не розпочаті книги. Графіки, побудовані за допомогою matplotlib, чітко ілюстрували розподіл: наприклад, бар-діаграма жанрів показала перевагу “фантастики” (5 книг). Час читання за місяць (у годинах) також підраховано правильно на основі записів у таблиці reading_history. Єдиним недоліком було відображення порожніх графіків при недостатній кількості даних, що супроводжувалося інформативним повідомленням “Недостатньо даних”

Завершення таблиці 3.1

1	2	3
6	Оцінка стабільності та обробки помилок	Для перевірки стабільності системи проводилися тести з некоректними діями: видалення неіснуючої книги, завантаження пошкодженого файлу, введення невалідних даних (наприклад, рік “-2023”). У всіх випадках сервер повертав відповідні повідомлення про помилки (коди 404, 400 тощо), а клієнт відображав їх у зрозумілій формі (наприклад, “Помилка з’єднання з сервером”). Логуювання подій у app.log дозволяло відстежити деталі, такі як “Помилка при читанні EPUB: invalid file”. При відключенні сервера клієнт коректно повідомляв про відсутність зв’язку, що свідчить про надійність обробки винятків.

Тестування показало, що система “Книжковий каталогізатор” функціонує стабільно та відповідає заявленим вимогам. Основні функції — додавання книг, читання, рекомендації та статистика — працюють коректно, а інтерфейс є інтуїтивно зрозумілим. Водночас виявлені дрібні недоліки, такі як сповільнення при великій кількості книг чи залежність рекомендацій від зовнішнього API, можуть бути усунуті в подальших ітераціях розробки. Загалом, система готова до використання, але потребує додаткового навантажувального тестування для оцінки її масштабованості.

Висновки до третього розділу

Розділ детально описує інтерфейс, порядок роботи та тестування "Книжкового каталогізатора з системою рекомендацій на основі ШІ". Система забезпечує зручне встановлення, інтуїтивно зрозумілий інтерфейс із модульною структурою та стабільну роботу основних функцій: каталогізації, читання, аналізу статистики та рекомендацій. Тестування підтвердило відповідність вимогам, хоча виявило незначні недоліки, такі як залежність від API та потреба в оптимізації при великих обсягах даних. Проєкт є працездатним рішенням для управління бібліотекою з потенціалом для подальшого вдосконалення.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи було спроектовано та реалізовано програмний додаток "Книжковий каталогізатор з системою рекомендацій на основі штучного інтелекту", проведено аналіз предметної області та здійснено порівняння з існуючими аналогами, такими як Goodreads, LibraryThing, Calibre та BookCatalogue, що дозволило визначити їхні переваги та недоліки. На основі цього аналізу сформульовано ключові вимоги до розробки системи, яка поєднує зручність каталогізації книг, інтеграцію онлайн-читалки та інтелектуальні рекомендації. У процесі дослідження розроблено архітектуру програмного продукту, визначено підходи до його реалізації та обрано сучасні технології, такі як Streamlit для клієнтської частини, Flask для серверної частини, SQLite для управління базою даних і Gemini API для генерації рекомендацій.

Проектування додатку включало аналіз варіантів використання, опис сценаріїв взаємодії користувача з системою та створення структури, що охоплює як клієнтську, так і серверну частини з чіткою взаємодією через REST API. Було спроектовано базу даних для зберігання метаданих книг, прогресу читання, закладок, цитат та історії активності, що забезпечує ефективне управління бібліотекою та швидкий доступ до даних. Описано алгоритми обробки книжкових файлів різних форматів, таких як EPUB, PDF, TXT і FB2, а також принципи роботи системи рекомендацій, яка аналізує вподобання користувача та пропонує персоналізовані пропозиції.

Розглянуто основний функціонал додатку, який включає додавання книг до каталогу, редагування їх метаданих, читання книг онлайн з можливістю налаштування інтерфейсу, збереження закладок і цитат, відстеження прогресу читання та отримання статистики читацької активності. Детально описано взаємодію компонентів системи, зокрема процес завантаження файлів, витягнення метаданих, збереження даних у базі та генерацію рекомендацій через інтеграцію з штучним інтелектом. Результатом кваліфікаційної роботи є готовий Книжний каталогізатор з системою рекомендацій на III.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Goodreads Book reviews, recommendations, and discussion: <https://www.goodreads.com/>
2. LibraryThing: <https://www.librarything.com/>
3. calibre - E-book management: <https://calibre-ebook.com/>
4. Book Catalogue: <https://book-catalogue.com/>
5. Розуміння Клієнт-Серверної Архітектури: <https://dou.ua/forums/topic/44636/>
6. A faster way to build and share data apps: <https://streamlit.io/>
7. Welcome to Flask's documentation: <https://flask.palletsprojects.com/en/stable/>
8. Клієнт-серверна: <https://training.qatestlab.com/blog/technical-articles/client-server-architecture/>
9. Streamlit documentation: <https://docs.streamlit.io/>
10. REST API як спосіб спілкування компонент веб-додатків: <https://foxminded.ua/shcho-take-rest-api/>
11. Що таке SQLite, для чого і як його використовувати: <https://foxminded.ua/prohramuvannia-baz-danykh-na-sqlite/>
12. Integrate Google AI models with an API: <https://ai.google.dev/>
13. Формати електронних книг і текстових: <https://xn--80aak1d.xn--j1amh/formati-elektronnih-knig-i-tekstovih-dokumentiv.html>
14. EbookLib 0.18: <https://pypi.org/project/EbookLib/>
15. Мова програмування Python та її застосування в різних галузях: <https://foxminded.ua/de-vykorystovuietsia-python/>
16. Streamlit Python: Tutorial: <https://www.datacamp.com/tutorial/streamlit>
17. Вступ до Python фреймворків: які вони бувають і для чого використовуються: <https://foxminded.ua/freimvorky-python/>
18. Що таке SQLite?: <https://freehost.com.ua/ukr/faq/wiki/chto-takoe-sqlite/>
19. Що таке CRUD простими словами: функції, переваги та приклади: <https://highload.tech/uk/shcho-take-crud-prostimi-slovami-funktsiyi-perevagi-ta-prikladi/>

20. Welcome to Ebook: <https://docs.sourcefabric.org/projects/ebooklib/en/latest/>
21. Tutorial: <https://pymupdf.readthedocs.io/en/latest/tutorial.html>
22. Beautiful Ebook Soup: <https://medium.com/@davidhenryhaase/a-beautiful-ebook-soup-fb849513cf66>
23. Gemini API and Google AI Studio now offer Grounding with Google Search: <https://developers.googleblog.com/en/gemini-api-and-ai-studio-now-offer-grounding-with-google-search/>
24. Matplotlib: Visualization with: <https://matplotlib.org/>
25. CSS для дизайнерів і не тільки: <https://dou.ua/forums/topic/38091/>
26. Requests: HTTP for: <https://docs.python-requests.org/en/latest/>
27. BeautifulEbook: <https://medium.com/@davidhenryhaase/a-beautiful-ebook-soufb849513cf66>
28. Tutorial: <https://pymupdf.readthedocs.io/en/latest/tutorial.html>
29. MySQL : <https://www.mysql.com/>
30. Python : <https://www.python.org/>
31. VS Code URL: <https://code.visualstudio.com/>

ДОДАТКИ

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						64

ЛІСТИНГ ПРОГРАМИ

client.py

```

import streamlit as st
import requests
import json
import io
import pandas as pd
import os
import base64
from PIL import Image
import matplotlib.pyplot as plt
import time
import tempfile
import ebooklib
from ebooklib import epub
from bs4 import BeautifulSoup
import fitz # PyMuPDF
# Базова URL адреса сервера
BASE_URL = "http://localhost:5000/api"
# Налаштування сторінки
st.set_page_config(
    page_title="Книжковий каталогізатор",
    page_icon="📖",
    layout="wide",
    initial_sidebar_state="expanded"
)
# Словник тем читалки
READER_THEMES = {
    "Світла": {"bg_color": "#ffffff", "text_color": "#000000"},
    "Темна": {"bg_color": "#1e1e1e", "text_color": "#ffffff"},
    "Сепія": {"bg_color": "#f8f0e3", "text_color": "#5b4636"},
    "Зелена": {"bg_color": "#e8f5e9", "text_color": "#1b5e20"}
}
# Список шрифтів
FONTS = ["Arial", "Times New Roman", "Roboto", "Open Sans", "Georgia"]
# Функції для роботи з API
def get_books():
    """Отримання списку всіх книг"""
    try:
        response = requests.get(f"{BASE_URL}/books")
        if response.status_code == 200:
            books = response.json()
            print(f"Отримано {len(books)} книг")
            return books
        else:
            st.error(f"Помилка отримання книг: {response.status_code}")
            print(f"Помилка отримання книг: {response.text}")
            return []
    except requests.RequestException as e:
        st.error(f"Помилка з'єднання з сервером: {e}")
        return []
def get_book(book_id):
    """Отримання деталей книги за ID"""
    try:
        response = requests.get(f"{BASE_URL}/books/{book_id}")
        if response.status_code == 200:
            return response.json()
        else:

```

```

        st.error(f"Помилка отримання книги: {response.status_code}")
        return None
    except requests.RequestException as e:
        st.error(f"Помилка з'єднання з сервером: {e}")
        return None
def add_book(file, title, year, author="", genre="", description="", series="", tags="",
cover=None, cover_url=None):
    """Додавання нової книги"""
    try:
        data = {
            "title": title,
            "year": str(year),
            "author": author,
            "genre": genre,
            "description": description,
            "series": series,
            "tags": tags
        }
        if cover_url:
            data["cover_url"] = cover_url

        files = {"file": file}
        if cover:
            files["cover"] = cover
        response = requests.post(f"{BASE_URL}/books", data=data, files=files)
        if response.status_code == 201:
            return response.json()
        else:
            error_msg = f"Помилка додавання книги: {response.status_code}"
            if response.text:
                error_msg += f", {response.text}"
            st.error(error_msg)
            return None
    except requests.RequestException as e:
        st.error(f"Помилка з'єднання з сервером: {e}")
        return None
def update_book(book_id, data, cover=None):
    """Оновлення інформації про книгу"""
    try:
        files = {}
        if cover:
            files["cover"] = cover
        response = requests.put(f"{BASE_URL}/books/{book_id}", data=data, files=files)
        if response.status_code == 200:
            return response.json()
        else:
            st.error(f"Помилка оновлення книги: {response.status_code}")
            return None
    except requests.RequestException as e:
        st.error(f"Помилка з'єднання з сервером: {e}")
        return None
def delete_book(book_id):
    """Видалення книги"""
    try:
        response = requests.delete(f"{BASE_URL}/books/{book_id}")
        if response.status_code == 200:
            return response.json()
        else:
            st.error(f"Помилка видалення книги: {response.status_code}")
            return None
    except requests.RequestException as e:
        st.error(f"Помилка з'єднання з сервером: {e}")
        return None
def update_progress(book_id, current_page, read_time=0):
    """Оновлення прогресу читання"""

```

```

try:
    data = {
        "current_page": current_page,
        "read_time": read_time
    }
    response = requests.put(f"{BASE_URL}/books/{book_id}/progress", json=data)
    if response.status_code == 200:
        return response.json()
    else:
        st.error(f"Помилка оновлення прогресу: {response.status_code}")
        return None
except requests.RequestException as e:
    st.error(f"Помилка з'єднання з сервером: {e}")
    return None
def get_book_content(book_id):
    """Отримання контенту книги для читання"""
    try:
        response = requests.get(f"{BASE_URL}/books/{book_id}/content")
        if response.status_code == 200:
            return response.json()
        else:
            st.error(f"Помилка отримання вмісту книги: {response.status_code}")
            return None
    except requests.RequestException as e:
        st.error(f"Помилка з'єднання з сервером: {e}")
        return None
def download_book_file(book_id):
    """Завантаження файлу книги"""
    try:
        response = requests.get(f"{BASE_URL}/books/{book_id}/download", stream=True)
        if response.status_code == 200:
            return response.content
        else:
            st.error(f"Помилка завантаження файлу книги: {response.status_code}")
            return None
    except requests.RequestException as e:
        st.error(f"Помилка з'єднання з сервером: {e}")
        return None
def get_bookmarks(book_id):
    """Отримання закладок для книги"""
    try:
        response = requests.get(f"{BASE_URL}/books/{book_id}/bookmarks")
        if response.status_code == 200:
            return response.json()
        else:
            st.error(f"Помилка отримання закладок: {response.status_code}")
            return []
    except requests.RequestException as e:
        st.error(f"Помилка з'єднання з сервером: {e}")
        return []
def add_bookmark(book_id, position, name=""):
    """Додавання закладки"""
    try:
        data = {"position": position, "name": name}
        response = requests.post(f"{BASE_URL}/books/{book_id}/bookmarks", json=data)
        if response.status_code == 201:
            return response.json()
        else:
            st.error(f"Помилка додавання закладки: {response.status_code}")
            return None
    except requests.RequestException as e:
        st.error(f"Помилка з'єднання з сервером: {e}")
        return None
def delete_bookmark(bookmark_id):
    """Видалення закладки"""

```

```

try:
    response = requests.delete(f"{BASE_URL}/bookmarks/{bookmark_id}")
    if response.status_code == 200:
        return response.json()
    else:
        st.error(f"Помилка видалення закладки: {response.status_code}")
        return None
except requests.RequestException as e:
    st.error(f"Помилка з'єднання з сервером: {e}")
    return None
def get_quotes(book_id):
    """Отримання цитат для книги"""
    try:
        response = requests.get(f"{BASE_URL}/books/{book_id}/quotes")
        if response.status_code == 200:
            return response.json()
        else:
            st.error(f"Помилка отримання цитат: {response.status_code}")
            return []
    except requests.RequestException as e:
        st.error(f"Помилка з'єднання з сервером: {e}")
        return []
def add_quote(book_id, text, position=0):
    """Додавання цитати"""
    try:
        data = {"text": text, "position": position}
        response = requests.post(f"{BASE_URL}/books/{book_id}/quotes", json=data)
        if response.status_code == 201:
            return response.json()
        else:
            st.error(f"Помилка додавання цитати: {response.status_code}")
            return None
    except requests.RequestException as e:
        st.error(f"Помилка з'єднання з сервером: {e}")
        return None
def delete_quote(quote_id):
    """Видалення цитати"""
    try:
        response = requests.delete(f"{BASE_URL}/quotes/{quote_id}")
        if response.status_code == 200:
            return response.json()
        else:
            st.error(f"Помилка видалення цитати: {response.status_code}")
            return None
    except requests.RequestException as e:
        st.error(f"Помилка з'єднання з сервером: {e}")
        return None
def get_recommendations(book_id=None, tags=None, preferences=None):
    """Отримання рекомендацій"""
    try:
        params = {}
        if book_id:
            params["book_id"] = book_id
        if tags:
            params["tags"] = tags
        if preferences:
            params["preferences"] = preferences
        response = requests.get(f"{BASE_URL}/recommendations", params=params)
        if response.status_code == 200:
            return response.json()
        else:
            st.error(f"Помилка отримання рекомендацій: {response.status_code}")
            return []
    except requests.RequestException as e:
        st.error(f"Помилка з'єднання з сервером: {e}")

```

```

        return []
def get_statistics():
    """Отримання статистики читання"""
    try:
        response = requests.get(f"{BASE_URL}/stats")
        if response.status_code == 200:
            return response.json()
        else:
            st.error(f"Помилка отримання статистики: {response.status_code}")
            return None
    except requests.RequestException as e:
        st.error(f"Помилка з'єднання з сервером: {e}")
        return None
def search_books(query="", author="", genre="", year=None, status=""):
    """Пошук книг за різними параметрами"""
    try:
        params = {}
        if query:
            params["query"] = query
        if author:
            params["author"] = author
        if genre:
            params["genre"] = genre
        if year:
            params["year"] = year
        if status:
            params["status"] = status
        response = requests.get(f"{BASE_URL}/search", params=params)
        if response.status_code == 200:
            return response.json()
        else:
            st.error(f"Помилка пошуку: {response.status_code}")
            return []
    except requests.RequestException as e:
        st.error(f"Помилка з'єднання з сервером: {e}")
        return []
def get_reading_history(book_id):
    """Отримання історії читання книги"""
    try:
        response = requests.get(f"{BASE_URL}/reading_history/{book_id}")
        if response.status_code == 200:
            return response.json()
        else:
            st.error(f"Помилка отримання історії читання: {response.status_code}")
            return []
    except requests.RequestException as e:
        st.error(f"Помилка з'єднання з сервером: {e}")
        return []
def extract_metadata(file):
    """Витягнення метаданих з файлу книги"""
    try:
        files = {"file": file}
        response = requests.post(f"{BASE_URL}/extract_metadata", files=files)
        if response.status_code == 200:
            return response.json()
        else:
            st.error(f"Помилка витягнення метаданих: {response.status_code}")
            return {}
    except requests.RequestException as e:
        st.error(f"Помилка з'єднання з сервером: {e}")
        return {}
# Налаштування дизайну
def local_css():
    """Додавання власних стилів CSS"""
    st.markdown("""

```

```

<style>
  .book-card {
    border: 1px solid #ddd;
    border-radius: 8px;
    padding: 10px;
    margin-bottom: 10px;
    transition: transform 0.2s, box-shadow 0.2s;
    background-color: white;
    height: 100%;
    display: flex;
    flex-direction: column;
  }
  .book-card:hover {
    transform: translateY(-5px);
    box-shadow: 0 5px 15px rgba(0,0,0,0.1);
  }
  .book-cover {
    width: 100%;
    height: 180px;
    object-fit: cover;
    border-radius: 5px;
    margin-bottom: 10px;
  }
  .book-title {
    font-size: 16px;
    font-weight: bold;
    margin: 5px 0;
    white-space: nowrap;
    overflow: hidden;
    text-overflow: ellipsis;
  }
  .book-author {
    font-size: 14px;
    color: #666;
    margin: 2px 0;
    white-space: nowrap;
    overflow: hidden;
    text-overflow: ellipsis;
  }
  .book-year {
    font-size: 12px;
    color: #999;
    margin: 2px 0;
  }
  .book-progress {
    height: 8px;
    background-color: #f0f0f0;
    border-radius: 4px;
    margin: 8px 0;
  }
  .book-progress-fill {
    height: 100%;
    background-color: #4CAF50;
    border-radius: 4px;
  }
  .book-status {
    font-size: 11px;
    padding: 2px 6px;
    border-radius: 10px;
    display: inline-block;
    margin-right: 5px;
  }
  .status-not-started {
    background-color: #f0f0f0;
    color: #666;

```

```

}
.status-reading {
  background-color: #2196F3;
  color: white;
}
.status-completed {
  background-color: #4CAF50;
  color: white;
}
}
.book-actions {
  display: flex;
  justify-content: space-between;
  margin-top: 8px;
  margin-top: auto;
}
}
.book-button {
  flex: 1;
  margin: 0 2px;
  font-size: 12px !important;
  padding: 2px 0 !important;
}
}
.reader-container {
  padding: 20px;
  border-radius: 5px;
  height: 70vh;
  overflow-y: auto;
  margin-bottom: 20px;
}
}
.bookmark-item {
  padding: 8px;
  margin-bottom: 5px;
  border-radius: 4px;
  background-color: #f5f5f5;
  cursor: pointer;
}
}
.quote-item {
  padding: 10px;
  margin-bottom: 10px;
  border-left: 3px solid #2196F3;
  background-color: #f9f9f9;
}
}
.sidebar-section {
  margin-bottom: 20px;
}
}
.sidebar-title {
  font-weight: bold;
  margin-bottom: 10px;
}
}
.success-message {
  background-color: #d4edda;
  color: #155724;
  padding: 10px;
  border-radius: 5px;
  margin-bottom: 15px;
}
}
.pagination {
  display: flex;
  justify-content: center;
  margin-top: 15px;
}
}
.pagination-button {
  margin: 0 5px;
}
}
.selected-text {
  background-color: #ffe33b;

```

```

    }
    .page-navigation {
        display: flex;
        justify-content: space-between;
        align-items: center;
        margin-top: 15px;
        padding: 10px;
        background-color: #f5f5f5;
        border-radius: 5px;
    }
    .page-info {
        font-size: 14px;
        font-weight: bold;
    }
    .stat-card {
        background-color: white;
        border-radius: 5px;
        padding: 15px;
        box-shadow: 0 2px 5px rgba(0,0,0,0.1);
        text-align: center;
        margin-bottom: 15px;
    }
    .stat-value {
        font-size: 24px;
        font-weight: bold;
        margin: 10px 0;
    }
    .stat-label {
        font-size: 14px;
        color: #666;
    }
    .card-content {
        flex-grow: 1;
    }
    .card-footer {
        margin-top: auto;
    }
}
</style>
"""', unsafe_allow_html=True)
# Ініціалізуємо стан сесії за потреби
if 'current_page' not in st.session_state:
    st.session_state.current_page = 'catalog'
if 'current_book_id' not in st.session_state:
    st.session_state.current_book_id = None
if 'reader_page' not in st.session_state:
    st.session_state.reader_page = 0
if 'reader_start_time' not in st.session_state:
    st.session_state.reader_start_time = None
if 'book_content' not in st.session_state:
    st.session_state.book_content = None
if 'reader_theme' not in st.session_state:
    st.session_state.reader_theme = 'Світла'
if 'font' not in st.session_state:
    st.session_state.font = 'Arial'
if 'font_size' not in st.session_state:
    st.session_state.font_size = 16
if 'search_query' not in st.session_state:
    st.session_state.search_query = ""
if 'filter_author' not in st.session_state:
    st.session_state.filter_author = ""
if 'filter_genre' not in st.session_state:
    st.session_state.filter_genre = ""
if 'filter_year' not in st.session_state:
    st.session_state.filter_year = None
if 'filter_status' not in st.session_state:

```

```

    st.session_state.filter_status = ""
if 'book_added' not in st.session_state:
    st.session_state.book_added = False
if 'book_edited' not in st.session_state:
    st.session_state.book_edited = False
if 'selected_text' not in st.session_state:
    st.session_state.selected_text = ""
# Функції навігації
def set_page(page):
    st.session_state.current_page = page
def set_current_book(book_id):
    st.session_state.current_book_id = book_id
    st.session_state.reader_page = 0
    st.session_state.book_content = None
def get_status_text(status):
    """Отримання тексту статусу читання"""
    status_map = {
        "not_started": "Не почато",
        "reading": "Читаю",
        "completed": "Прочитано"
    }
    return status_map.get(status, status)
def get_status_class(status):
    """Отримання CSS класу для статусу"""
    status_map = {
        "not_started": "status-not-started",
        "reading": "status-reading",
        "completed": "status-completed"
    }
    return status_map.get(status, "status-not-started")
# Відображення карточки книги
def display_book_card(book, col):
    with col:
        try:
            # Отримуємо основну інформацію про книгу
            title = book.get('title', 'Без назви')
            author = book.get('author', 'Невідомий автор')
            year = book.get('year', '')
            book_id = book['id']
            # URL обкладинки з мітками часу, щоб уникнути кешування
            cover_url = f"{BASE_URL}/books/{book_id}/cover?t={int(time.time())}"
            # Відображаємо обкладинку
            st.image(cover_url, use_container_width=True)
            # Інформація про книгу
            st.markdown(f"### {title}")
            st.markdown(f"***{author}**")
            st.markdown(f"Рік: {year}")
            # Прогрес читання
            progress = book.get('read_progress', 0)
            st.progress(float(progress) / 100 if progress else 0)
            # Статус читання
            status = book.get('reading_status', 'not_started')
            st.write(get_status_text(status))
            # Кнопки дій
            col1, col2, col3 = st.columns(3)
            with col1:
                if st.button("📖 Читати", key=f"read_{book_id}"):
                    set_current_book(book_id)
                    set_page('reader')
            with col2:
                if st.button("✎ Редагувати", key=f"edit_{book_id}"):
                    set_current_book(book_id)
                    set_page('edit_book')
            with col3:
                if st.button("✖ Видалити", key=f"delete_{book_id}"):

```

```

        if delete_book(book_id):
            st.success("Книгу видалено")
            if st.session_state.current_book_id == book_id:
                set_current_book(None)
            st.rerun()
    except Exception as e:
        st.error(f"Помилка відображення книги: {e}")
def load_image_with_fallback(image_url, width=None):
    """Завантажує зображення з запасним варіантом"""
    try:
        if width:
            return st.image(image_url, width=width)
        else:
            return st.image(image_url, use_container_width=True)
    except Exception as e:
        st.error(f"Не вдалося завантажити зображення: {e}")
        return st.image("https://via.placeholder.com/300x450?text=No+Cover",
            use_container_width=True)
# Бічна панель
def sidebar():
    st.sidebar.title("📖 Книжковий каталогізатор")
    menu_options = {
        "catalog": "📖 Мій каталог",
        "search": "🔍 Пошук",
        "stats": "📊 Статистика",
        "recommendations": "💡 Рекомендації"
    }
    for page_id, page_name in menu_options.items():
        if st.sidebar.button(page_name, key=f"menu_{page_id}"):
            set_page(page_id)
    st.sidebar.divider()
    # Додавання нової книги
    if st.sidebar.button("📖 Додати нову книгу", key="add_book_btn"):
        set_page('add_book')
    # Показуємо статистику останніх прочитаних книг
    st.sidebar.divider()
    st.sidebar.subheader("Нещодавня активність")
    try:
        response = requests.get(f"{BASE_URL}/reading_history/recent?days=7")
        if response.status_code == 200:
            recent_activity = response.json()
            if recent_activity:
                for activity in recent_activity[:3]: # Показуємо максимум 3 останні книги
                    book_id = activity['book_id']
                    st.sidebar.markdown(f"""
                    <div style="padding: 5px; border-left: 3px solid #2196F3; margin-bottom:
10px;">
                        <div style="font-weight: bold; font-size:
14px;">{activity['title']}</div>
                        <div style="font-size: 12px; color:
#666;">{activity['author']}</div>
                        <div style="font-size: 12px;">Час читання:
{round(activity['total_time']/3600, 1)} год.</div>
                    </div>
                    """, unsafe_allow_html=True)
                    if st.sidebar.button("Продовжити читання", key=f"continue_{book_id}"):
                        set_current_book(book_id)
                        set_page('reader')
                else:
                    st.sidebar.info("Немає нещодавньої активності")
            except:
                pass
    # Сторінка каталогу
def catalog_page():

```

```

st.title("Мій книжковий каталог")
# Показуємо повідомлення про успішне додавання книги
if st.session_state.book_added:
    st.markdown('<div class="success-message">☑ Книгу успішно додано</div>',
unsafe_allow_html=True)
    st.session_state.book_added = False
# Показуємо повідомлення про успішне редагування книги
if st.session_state.book_edited:
    st.markdown('<div class="success-message">☑ Книгу успішно оновлено</div>',
unsafe_allow_html=True)
    st.session_state.book_edited = False
# Фільтри в експандері
with st.expander("Фільтри"):
    col1, col2, col3 = st.columns(3)
    with col1:
        filter_author = st.text_input("Автор", st.session_state.filter_author)
        if filter_author != st.session_state.filter_author:
            st.session_state.filter_author = filter_author
    with col2:
        filter_genre = st.text_input("Жанр", st.session_state.filter_genre)
        if filter_genre != st.session_state.filter_genre:
            st.session_state.filter_genre = filter_genre
    with col3:
        filter_status = st.selectbox(
            "Статус",
            ["", "not_started", "reading", "completed"],
            format_func=get_status_text
        )
        if filter_status != st.session_state.filter_status:
            st.session_state.filter_status = filter_status
# Отримання книг з фільтрами
books = search_books(
    author=st.session_state.filter_author,
    genre=st.session_state.filter_genre,
    status=st.session_state.filter_status
)
if not books:
    st.info("Книги не знайдено. Додайте нову книгу через бічне меню.")
    return
# Відображення книг у сітці
col_count = 4 # Показуємо книги у 4 колонки
for i in range(0, len(books), col_count):
    cols = st.columns(col_count)
    for j in range(col_count):
        if i + j < len(books):
            display_book_card(books[i + j], cols[j])
# Сторінка додавання книги
def add_book_page():
    st.title("Додати нову книгу")
    uploaded_file = st.file_uploader("Виберіть файл книги", type=["epub", "pdf", "txt",
"fb2", "mobi"])
    if uploaded_file:
        # Додаємо кнопку для парсингу метаданих з файлу
        if st.button("Витягнути дані з файлу"):
            with st.spinner("Аналізуємо файл..."):
                metadata = extract_metadata(uploaded_file)
                if metadata:
                    st.session_state.extracted_title = metadata.get('title', '')
                    st.session_state.extracted_author = metadata.get('author', '')
                    st.session_state.extracted_year = metadata.get('year', 2023)
                    st.session_state.extracted_genre = metadata.get('genre', '')
                    st.session_state.extracted_description = metadata.get('description', '')
                    # Якщо є обкладинка в метаданих, можемо показати її
                    if 'cover_base64' in metadata:
                        st.session_state.extracted_cover = metadata['cover_base64']

```

```

        st.image(metadata['cover_base64'], caption="Витягнута обкладинка",
width=200)

        st.success("Дані успішно витягнуто з файлу!")
        st.rerun() # Оновлюємо сторінку для відображення даних
        # Отримуємо базове ім'я файлу як початкове значення для назви
        default_title = getattr(st.session_state, 'extracted_title',
os.path.splitext(uploaded_file.name)[0])
        default_author = getattr(st.session_state, 'extracted_author', '')
        default_year = getattr(st.session_state, 'extracted_year', 2023)
        default_genre = getattr(st.session_state, 'extracted_genre', '')
        default_description = getattr(st.session_state, 'extracted_description', '')
        col1, col2 = st.columns(2)
        with col1:
            title = st.text_input("Назва книги*", value=default_title)
            author = st.text_input("Автор", value=default_author)
            year = st.number_input("Рік видання*", min_value=1, max_value=2100,
value=default_year)
            genre = st.text_input("Жанр", value=default_genre)
        with col2:
            description = st.text_area("Опис", value=default_description)
            series = st.text_input("Серія")
            tags = st.text_input("Теги (розділені комами)")
        # Завантаження обкладинки
        st.subheader("Обкладинка книги")
        # Показуємо витягнуту обкладинку, якщо вона є
        extracted_cover_exists = hasattr(st.session_state, 'extracted_cover')
        if extracted_cover_exists:
            st.image(st.session_state.extracted_cover, caption="Витягнута обкладинка",
width=200)
        # Прапорець для використання витягнутої обкладинки
        use_extracted = False
        if extracted_cover_exists:
            use_extracted = st.checkbox("Використовувати витягнуту обкладинку", value=True)
        # Завантаження файлу обкладинки (альтернативний варіант)
        uploaded_cover = None
        if not use_extracted or not extracted_cover_exists:
            uploaded_cover = st.file_uploader("Завантажити обкладинку", type=["jpg", "jpeg",
"png"])
        cover_url = st.text_input("АБО вкажіть URL обкладинки (не обов'язково)")
        if st.button("Додати книгу"):
            if not title or year <= 0:
                st.error("Заповніть обов'язкові поля (назва та рік)")
                return
            with st.spinner("Додаємо книгу..."):
                # Підготовка даних
                data = {
                    "title": title,
                    "year": str(year),
                    "author": author,
                    "genre": genre,
                    "description": description,
                    "series": series,
                    "tags": tags
                }
                # Підготовка обкладинки
                cover_data = None
                cover_url_data = None
                if use_extracted and extracted_cover_exists:
                    # Отримуємо бінарні дані з base64
                    cover_data_string = st.session_state.extracted_cover.split(',')[1]
                    cover_data = base64.b64decode(cover_data_string)
                    # Створюємо тимчасовий файл для обкладинки
                    temp_cover = tempfile.NamedTemporaryFile(delete=False, suffix='.jpg')
                    temp_cover.write(cover_data)
                    temp_cover.close()

```

```

        # Відкриваємо файл для передачі у запит
        cover_data = open(temp_cover.name, 'rb')
    elif uploaded_cover:
        cover_data = uploaded_cover
    elif 'cover_url' in locals() and cover_url:
        cover_url_data = cover_url
    # Надсилання запиту на сервер
    result = None
    try:
        if cover_data:
            result = add_book(
                file=uploaded_file,
                title=title,
                year=year,
                author=author,
                genre=genre,
                description=description,
                series=series,
                tags=tags,
                cover=cover_data
            )
        else:
            result = add_book(
                file=uploaded_file,
                title=title,
                year=year,
                author=author,
                genre=genre,
                description=description,
                series=series,
                tags=tags,
                cover_url=cover_url_data
            )
        # Видаляємо тимчасовий файл обкладинки, якщо він був створений
        if use_extracted and extracted_cover_exists and 'temp_cover' in
locals():
            try:
                cover_data.close()
                os.unlink(temp_cover.name)
            except:
                pass
        if result:
            st.session_state.book_added = True
            # Очищаємо витягнуті метадані
            if hasattr(st.session_state, 'extracted_title'):
                del st.session_state.extracted_title
            if hasattr(st.session_state, 'extracted_author'):
                del st.session_state.extracted_author
            if hasattr(st.session_state, 'extracted_year'):
                del st.session_state.extracted_year
            if hasattr(st.session_state, 'extracted_genre'):
                del st.session_state.extracted_genre
            if hasattr(st.session_state, 'extracted_description'):
                del st.session_state.extracted_description
            if hasattr(st.session_state, 'extracted_cover'):
                del st.session_state.extracted_cover
            # Перехід назад до каталогу
            set_page('catalog')
        except Exception as e:
            st.error(f"Помилка при додаванні книги: {e}")
    else:
        st.info("Завантажте файл книги для початку.")
    # Сторінка редагування книги
    def edit_book_page():
        if not st.session_state.current_book_id:

```

```

        st.error("Книгу не вибрано")
        return
    book = get_book(st.session_state.current_book_id)
    if not book:
        st.error("Не вдалося отримати дані книги")
        return
    st.title(f"Редагування книги: {book['title']}")
    col1, col2 = st.columns([1, 2])
    with col1:
        # Показуємо обкладинку
        st.image(book['cover_url'], caption="Поточна обкладинка", use_column_width=True)
        uploaded_cover = st.file_uploader("Нова обкладинка (опціонально)", type=["jpg",
"jpeg", "png"])
    with col2:
        title = st.text_input("Назва книги*", value=book['title'])
        author = st.text_input("Автор", value=book.get('author', ''))
        year = st.number_input("Рік видання*", min_value=1, max_value=2100,
value=book['year'])
        genre = st.text_input("Жанр", value=book.get('genre', ''))
        description = st.text_area("Опис", value=book.get('description', ''))
        series = st.text_input("Серія", value=book.get('series', ''))
        tags = st.text_input("Теги (розділені комами)", value=book.get('tags', ''))
    # Інформація про прогрес
    st.subheader("Інформація про читання")
    st.info(f"Поточний статус: {get_status_text(book.get('reading_status',
'not_started'))}")
    st.info(f"Прогрес: Сторінка {book.get('current_page', 0)} з {book.get('total_pages', 0)}
({book.get('read_progress', 0)}%)")
    if st.button("Зберегти зміни"):
        if not title or year <= 0:
            st.error("Заповніть обов'язкові поля (назва та рік)")
            return
        with st.spinner("Оновлюємо дані книги..."):
            # Підготовка даних для оновлення
            data = {
                "title": title,
                "author": author,
                "year": year,
                "genre": genre,
                "description": description,
                "series": series,
                "tags": tags
            }
            # Оновлення книги на сервері
            result = update_book(book['id'], data, uploaded_cover)
            if result:
                st.session_state.book_edited = True
                # Перехід назад до каталогу
                set_page('catalog')
# Сторінка читання книги
def reader_page():
    if not st.session_state.current_book_id:
        st.error("Книгу не вибрано")
        return
    book = get_book(st.session_state.current_book_id)
    if not book:
        st.error("Не вдалося отримати дані книги")
        return
    # Отримуємо вміст книги, якщо ще не завантажено
    if not st.session_state.book_content:
        with st.spinner("Завантаження книги..."):
            st.session_state.book_content = get_book_content(book['id'])
            # Якщо немає вмісту, повертаємося до каталогу
            if not st.session_state.book_content:
                st.error("Не вдалося завантажити вміст книги")

```

```

        if st.button("Повернутися до каталогу"):
            set_page('catalog')
            return
        # Якщо current_page з книги > 0, встановлюємо цю сторінку
        if book.get('current_page', 0) > 0:
            st.session_state.reader_page = min(book.get('current_page', 0),

st.session_state.book_content.get('total_pages', 1) - 1)
        # Запускаємо таймер для відстеження часу читання
        if not st.session_state.reader_start_time:
            st.session_state.reader_start_time = time.time()
        # Відображення книги
        st.title(f"Читання: {book['title']}")
        # Кнопка повернення до каталогу
        if st.button("<- Повернутися до каталогу"):
            # Зберігаємо прогрес
            read_time = int(time.time() - st.session_state.reader_start_time)
            update_progress(book['id'], st.session_state.reader_page, read_time)
            st.session_state.reader_start_time = None
            set_page('catalog')
            return
        # Налаштування читання
        col1, col2, col3 = st.columns(3)
        with col1:
            theme = st.selectbox("Тема", list(READER_THEMES.keys()),
index=list(READER_THEMES.keys()).index(st.session_state.reader_theme))
            if theme != st.session_state.reader_theme:
                st.session_state.reader_theme = theme
        with col2:
            font = st.selectbox("Шрифт", FONTS, index=FONTS.index(st.session_state.font))
            if font != st.session_state.font:
                st.session_state.font = font
        with col3:
            font_size = st.slider("Розмір шрифту", 12, 24, st.session_state.font_size)
            if font_size != st.session_state.font_size:
                st.session_state.font_size = font_size
        # Отримання налаштувань теми
        theme_settings = READER_THEMES[st.session_state.reader_theme]
        # Навігація по сторінках
        total_pages = st.session_state.book_content.get('total_pages', 0)
        current_page = st.session_state.reader_page
        # Налаштування стилів для контейнера читання
        container_style = f"""
<style>
        .reader-container {{
            background-color: {theme_settings['bg_color']};
            color: {theme_settings['text_color']};
            font-family: {st.session_state.font}, sans-serif;
            font-size: {st.session_state.font_size}px;
            padding: 20px;
            border-radius: 5px;
            line-height: 1.6;
            height: 70vh;
            overflow-y: auto;
        }}
        .reader-container p {{
            margin-bottom: 1em;
        }}
        .reader-container h1, .reader-container h2, .reader-container h3 {{
            margin-top: 1.5em;
            margin-bottom: 0.5em;
        }}
        .selected-text {{
            background-color: #ffeb3b;
        }}
    """

```

```

</style>
"""
st.markdown(container_style, unsafe_allow_html=True)
# Показуємо текст поточної сторінки
if current_page < len(st.session_state.book_content.get('pages', [])):
    page_content = st.session_state.book_content['pages'][current_page]
    # Створюємо контейнер для відображення сторінки
    container = st.container()
    container.markdown(f'<div class="reader-container" id="reader-
content">{page_content}</div>', unsafe_allow_html=True)
    # JavaScript для вибору тексту
    st.markdown("""
<script>
    document.addEventListener('DOMContentLoaded', function() {
        var readerContent = document.getElementById('reader-content');
        if (readerContent) {
            readerContent.addEventListener('mouseup', function() {
                const selection = window.getSelection();
                const selectedText = selection.toString();
                if (selectedText.length > 0) {
                    window.parent.postMessage({
                        type: 'streamlit:setComponentValue',
                        value: selectedText
                    }, '*');
                }
            });
        }
    });
</script>
""", unsafe_allow_html=True)
# Навігація сторінками і кнопки дій
col1, col2, col3 = st.columns([2, 4, 2])
with col1:
    if st.button("⏪ Попередня сторінка", disabled=current_page <= 0):
        # Зберігаємо прогрес перед переходом
        if current_page > 0:
            read_time = int(time.time() - st.session_state.reader_start_time)
            update_progress(book['id'], current_page, read_time)
            st.session_state.reader_start_time = time.time()
            st.session_state.reader_page -= 1
            st.rerun()
with col2:
    st.markdown(f'<div class="page-info">Сторінка {current_page + 1} з
{total_pages}</div>', unsafe_allow_html=True)
    with col3:
        if st.button("⏩ Наступна сторінка", disabled=current_page >= total_pages - 1):
            # Зберігаємо прогрес перед переходом
            if current_page < total_pages - 1:
                read_time = int(time.time() - st.session_state.reader_start_time)
                update_progress(book['id'], current_page + 1, read_time)
                st.session_state.reader_start_time = time.time()
                st.session_state.reader_page += 1
                st.rerun()
# Додавання закладки
col1, col2 = st.columns([3, 1])
bookmark_name = col1.text_input("Назва закладки", value=f"Сторінка {current_page +
1}")
if col2.button("🔖 Додати закладку"):
    add_bookmark(book['id'], current_page, bookmark_name)
    st.success(f"Закладку додано на сторінці {current_page + 1}")
# Обробка виділеного тексту
if hasattr(st.session_state, 'selected_text') and st.session_state.selected_text:
    st.subheader("Виділений текст:")
    st.info(st.session_state.selected_text)
    if st.button("Зберегти як цитату"):

```

```

        if add_quote(book['id'], st.session_state.selected_text, current_page):
            st.success("Цитату збережено!")
            st.session_state.selected_text = ""
            st.rerun()
    else:
        st.error("Сторінку не знайдено")
# Вкладки для закладок та цитат
st.divider()
tabs = st.tabs(["Закладки", "Цитати", "Деталі книги"])
# Вкладка закладок
with tabs[0]:
    bookmarks = get_bookmarks(book['id'])
    if bookmarks:
        for bookmark in bookmarks:
            col1, col2, col3 = st.columns([5, 3, 2])
            with col1:
                st.markdown(f"*{bookmark['name']}*")
            with col2:
                if st.button("Перейти", key=f"goto_bm_{bookmark['id']}"):
                    # Зберігаємо прогрес перед переходом
                    read_time = int(time.time() - st.session_state.reader_start_time)
                    update_progress(book['id'], bookmark['position'], read_time)
                    st.session_state.reader_start_time = time.time()
                    st.session_state.reader_page = bookmark['position']
                    st.rerun()
            with col3:
                if st.button("Видалити", key=f"del_bm_{bookmark['id']}"):
                    delete_bookmark(bookmark['id'])
                    st.success("Закладку видалено")
                    st.rerun()
        else:
            st.info("Закладок поки немає")
# Вкладка цитат
with tabs[1]:
    quotes = get_quotes(book['id'])
    if quotes:
        for quote in quotes:
            col1, col2 = st.columns([5, 1])
            with col1:
                st.markdown(f"""
                <div class="quote-item">
                    "{quote['text']}"
                    <div style="font-size: 12px; text-align: right;">Сторінка
{quote['position'] + 1}</div>
                </div>
                """, unsafe_allow_html=True)
            with col2:
                if st.button("Перейти", key=f"goto_quote_{quote['id']}"):
                    # Зберігаємо прогрес перед переходом
                    read_time = int(time.time() - st.session_state.reader_start_time)
                    update_progress(book['id'], quote['position'], read_time)
                    st.session_state.reader_start_time = time.time()
                    st.session_state.reader_page = quote['position']
                    st.rerun()
                if st.button("Видалити", key=f"del_quote_{quote['id']}"):
                    delete_quote(quote['id'])
                    st.success("Цитату видалено")
                    st.rerun()
        else:
            st.info("Цитат поки немає")
# Вкладка з деталями книги
with tabs[2]:
    st.subheader("Інформація про книгу")
    st.markdown(f"*Назва:* {book['title']}")
    st.markdown(f"*Автор:* {book.get('author', 'Невідомий')}")

```

```

st.markdown(f"""Рік:** {book.get('year', '')}""")
st.markdown(f"""Жанр:** {book.get('genre', 'Не вказано')}""")
if book.get('description'):
    st.markdown(f"""Опис:**\n{book['description']}""")
# Кнопка для завантаження файлу
if st.button("⬇ Завантажити файл книги"):
    file_data = download_book_file(book['id'])
    if file_data:
        file_ext = book.get('file_format', 'epub')
        st.download_button(
            label="Підтвердити завантаження",
            data=file_data,
            file_name=f"{book['title']}.{file_ext}",
            mime=f"application/{file_ext}"
        )
# Сторінка пошуку
def search_page():
    st.title("Пошук книг")
    # Поле для пошуку
    search_query = st.text_input("Пошук за назвою, описом або тегами",
st.session_state.search_query)
    # Додаткові фільтри
    with st.expander("Додаткові фільтри"):
        col1, col2, col3 = st.columns(3)
        with col1:
            filter_author = st.text_input("Автор", st.session_state.filter_author)
        with col2:
            filter_genre = st.text_input("Жанр", st.session_state.filter_genre)
        with col3:
            filter_year = st.number_input("Рік", min_value=None, max_value=2100,
value=st.session_state.filter_year)
            filter_status = st.selectbox(
                "Статус",
                ["", "not_started", "reading", "completed"],
                format_func=get_status_text,
                index=0 if not st.session_state.filter_status else ["", "not_started",
"reading", "completed"].index(st.session_state.filter_status)
            )
    # Оновлюємо стан сесії, якщо змінилися параметри пошуку
    if search_query != st.session_state.search_query:
        st.session_state.search_query = search_query
    if filter_author != st.session_state.filter_author:
        st.session_state.filter_author = filter_author
    if filter_genre != st.session_state.filter_genre:
        st.session_state.filter_genre = filter_genre
    if filter_year != st.session_state.filter_year:
        st.session_state.filter_year = filter_year
    if filter_status != st.session_state.filter_status:
        st.session_state.filter_status = filter_status
    if st.button("Пошук"):
        # Виконуємо пошук
        books = search_books(
            query=st.session_state.search_query,
            author=st.session_state.filter_author,
            genre=st.session_state.filter_genre,
            year=st.session_state.filter_year,
            status=st.session_state.filter_status
        )
    if not books:
        st.info("Книги не знайдено. Спробуйте змінити параметри пошуку.")
        return
    # Відображення результатів пошуку
    st.subheader(f"Знайдено: {len(books)} книг")
    # Відображення книг у сітці
    col_count = 4 # Кількість колонок

```

```

        for i in range(0, len(books), col_count):
            cols = st.columns(col_count)
            for j in range(col_count):
                if i + j < len(books):
                    display_book_card(books[i + j], cols[j])
# Сторінка статистики
def stats_page():
    st.title("Статистика читання")
    # Отримуємо статистику
    stats = get_statistics()
    if not stats:
        st.error("Не вдалося отримати статистику")
        return
    # Основні метрики
    col1, col2, col3, col4 = st.columns(4)
    with col1:
        st.markdown("""
            <div class="stat-card">
                <div class="stat-label">Всього книг</div>
                <div class="stat-value">{}</div>
            </div>
            """.format(stats['total_books']), unsafe_allow_html=True)
    with col2:
        st.markdown("""
            <div class="stat-card">
                <div class="stat-label">Прочитано</div>
                <div class="stat-value">{}</div>
            </div>
            """.format(stats['completed_books']), unsafe_allow_html=True)
    with col3:
        st.markdown("""
            <div class="stat-card">
                <div class="stat-label">Читаю зараз</div>
                <div class="stat-value">{}</div>
            </div>
            """.format(stats['reading_books']), unsafe_allow_html=True)
    with col4:
        st.markdown("""
            <div class="stat-card">
                <div class="stat-label">Не розпочато</div>
                <div class="stat-value">{}</div>
            </div>
            """.format(stats['not_started_books']), unsafe_allow_html=True)
    # Додаткова статистика
    st.subheader("Активність за останній місяць")
    col1, col2, col3 = st.columns(3)
    with col1:
        st.markdown("""
            <div class="stat-card">
                <div class="stat-label">Годин читання</div>
                <div class="stat-value">{}</div>
            </div>
            """.format(stats.get('total_read_hours', 0)), unsafe_allow_html=True)
    with col2:
        st.markdown("""
            <div class="stat-card">
                <div class="stat-label">Додано книг</div>
                <div class="stat-value">{}</div>
            </div>
            """.format(stats.get('recent_books', 0)), unsafe_allow_html=True)
    with col3:
        st.markdown("""
            <div class="stat-card">
                <div class="stat-label">Завершено книг</div>
                <div class="stat-value">{}</div>
            </div>
            """.format(stats.get('recent_books', 0)), unsafe_allow_html=True)

```

```

</div>
"".format(stats.get('recently_completed', 0)), unsafe_allow_html=True)

# Графіки
st.subheader("Статистика за категоріями")
# Вкладки для різних типів графіків
tabs = st.tabs(["Жанри", "Автори", "Роки"])
# Графік жанрів
with tabs[0]:
    if stats.get('genres') and len(stats['genres']) > 0:
        genres = pd.DataFrame(stats['genres'])
        fig, ax = plt.subplots(figsize=(10, 6))
        ax.bar(genres['genre'], genres['count'], color='skyblue')
        ax.set_xticklabels(genres['genre'], rotation=45, ha='right')
        ax.set_ylabel('Кількість книг')
        ax.set_title('Розподіл книг за жанрами', fontsize=14)
        for i, v in enumerate(genres['count']):
            ax.text(i, v + 0.1, str(v), ha='center')
        plt.tight_layout()
        st.pyplot(fig)
    else:
        st.info("Недостатньо даних для відображення статистики за жанрами")
# Графік авторів
with tabs[1]:
    if stats.get('authors') and len(stats['authors']) > 0:
        authors = pd.DataFrame(stats['authors'])
        fig, ax = plt.subplots(figsize=(10, 6))
        ax.bar(authors['author'], authors['count'], color='lightgreen')
        ax.set_xticklabels(authors['author'], rotation=45, ha='right')
        ax.set_ylabel('Кількість книг')
        ax.set_title('Найпопулярніші автори', fontsize=14)
        for i, v in enumerate(authors['count']):
            ax.text(i, v + 0.1, str(v), ha='center')
        plt.tight_layout()
        st.pyplot(fig)
    else:
        st.info("Недостатньо даних для відображення статистики за авторами")
# Графік років
with tabs[2]:
    if stats.get('years') and len(stats['years']) > 0:
        years = pd.DataFrame(stats['years'])
        # Сортуємо за роками
        years = years.sort_values('year')
        fig, ax = plt.subplots(figsize=(10, 6))
        ax.bar(years['year'].astype(str), years['count'], color='salmon')
        ax.set_xticklabels(years['year'].astype(str), rotation=45, ha='right')
        ax.set_ylabel('Кількість книг')
        ax.set_title('Розподіл книг за роками видання', fontsize=14)
        for i, v in enumerate(years['count']):
            ax.text(i, v + 0.1, str(v), ha='center')
        plt.tight_layout()
        st.pyplot(fig)
    else:
        st.info("Недостатньо даних для відображення статистики за роками")
# Сторінка рекомендацій
def recommendations_page():
    st.title("Рекомендації для читання")
    # Вкладки для різних типів рекомендацій
    tabs = st.tabs(["Для вас", "На основі книги", "За тегами"])
    # Персональні рекомендації
    with tabs[0]:
        st.subheader("Персональні рекомендації")
        # Кнопка для отримання рекомендацій
        if st.button("Отримати персональні рекомендації"):
            with st.spinner("Аналізуємо ваші вподобання..."):
                personal_recs = get_recommendations()

```

```

        if not personal_recs:
            st.info("Недостатньо даних для рекомендацій. Додайте більше книг до каталогу або оцініть наявні.")
        else:
            # Відображення рекомендацій у сітці
            st.success(f"Знайдено {len(personal_recs)} рекомендацій для вас")
            col_count = 4 # Кількість колонок
            for i in range(0, len(personal_recs), col_count):
                cols = st.columns(col_count)
                for j in range(col_count):
                    if i + j < len(personal_recs):
                        display_book_card(personal_recs[i + j], cols[j])

    else:
        st.info("Натисніть кнопку вище, щоб отримати персоналізовані рекомендації на основі ваших читацьких вподобань.")
        # Рекомендації на основі книги
        with tabs[1]:
            st.subheader("Рекомендації на основі книги")
            # Вибір книги для рекомендацій
            all_books = get_books()
            if not all_books:
                st.info("Додайте книги до каталогу, щоб отримати рекомендації.")
            else:
                # Створюємо словник для селектбоксу
                book_options = {f"{book['title']} ({book.get('author', 'Невідомий автор')})":
book['id'] for book in all_books}
                selected_book_name = st.selectbox("Виберіть книгу", list(book_options.keys()))
                selected_book_id = book_options[selected_book_name]
                if st.button("Отримати рекомендації"):
                    with st.spinner("Шукаємо схожі книги..."):
                        book_recs = get_recommendations(book_id=selected_book_id)
                        if not book_recs:
                            st.info("Не знайдено схожих книг для рекомендації.")
                        else:
                            # Відображення рекомендацій у сітці
                            st.success(f"Знайдено {len(book_recs)} схожих книг")
                            col_count = 4 # Кількість колонок
                            for i in range(0, len(book_recs), col_count):
                                cols = st.columns(col_count)
                                for j in range(col_count):
                                    if i + j < len(book_recs):
                                        display_book_card(book_recs[i + j], cols[j])

        # Рекомендації за тегами
        with tabs[2]:
            st.subheader("Рекомендації за тегами та вподобаннями")
            # Введення тегів та вподобань
            tags = st.text_input("Теги (розділені комами)",
                                placeholder="наприклад: фантастика, пригоди, історія")
            preferences = st.text_area("Опишіть, що вам подобається",
                                      placeholder="наприклад: Мені подобаються книги про космічні подорожі з філософським підтекстом")
            if st.button("Знайти книги за вподобаннями"):
                if not tags and not preferences:
                    st.warning("Вкажіть теги або опишіть ваші вподобання для отримання рекомендацій.")
                else:
                    with st.spinner("Шукаємо книги за вашими вподобаннями..."):
                        tag_recs = get_recommendations(tags=tags, preferences=preferences)
                        if not tag_recs:
                            st.info("Не знайдено рекомендацій за вказаними критеріями.")
                        else:
                            # Відображення рекомендацій у сітці
                            st.success(f"Знайдено {len(tag_recs)} підходящих книг")
                            col_count = 4 # Кількість колонок
                            for i in range(0, len(tag_recs), col_count):

```

```

        cols = st.columns(col_count)
        for j in range(col_count):
            if i + j < len(tag_recs):
                display_book_card(tag_recs[i + j], cols[j])

# Головна функція
def main():
    """Головна функція додатку"""
    local_css()
    sidebar()
    # Відображення відповідної сторінки
    if st.session_state.current_page == 'catalog':
        catalog_page()
    elif st.session_state.current_page == 'add_book':
        add_book_page()
    elif st.session_state.current_page == 'edit_book':
        edit_book_page()
    elif st.session_state.current_page == 'reader':
        reader_page()
    elif st.session_state.current_page == 'search':
        search_page()
    elif st.session_state.current_page == 'stats':
        stats_page()
    elif st.session_state.current_page == 'recommendations':
        recommendations_page()
if __name__ == '__main__':
    main()

```

server.py

```

import base64
from flask import Flask, request, jsonify, send_file, abort, send_from_directory
import sqlite3
import os
import json
import datetime
import re
import requests
import tempfile
import shutil
import logging
import uuid
from io import BytesIO
from PIL import Image, ImageDraw
import fitz # PyMuPDF
import ebooklib
from ebooklib import epub
from bs4 import BeautifulSoup
import google.generativeai as genai
from google.api_core import retry
# Спроба імпортувати необхідні бібліотеки
app = Flask(__name__)
app.config['MAX_CONTENT_LENGTH'] = 100 * 1024 * 1024 # 100 МБ
# Налаштування логування
logging.basicConfig(level=logging.INFO,
                    format='%(asctime)s - %(levelname)s - %(message)s',
                    handlers=[logging.FileHandler("app.log"),
                             logging.StreamHandler()])
logger = logging.getLogger(__name__)
# Шляхи до важливих директорій
BASE_DIR = os.path.dirname(os.path.abspath(__file__))
STATIC_DIR = os.path.join(BASE_DIR, 'static')
BOOKS_DIR = os.path.join(BASE_DIR, 'books')
COVERS_DIR = os.path.join(STATIC_DIR, 'covers')
IMAGES_DIR = os.path.join(STATIC_DIR, 'images')

```

```

DATABASE = os.path.join(BASE_DIR, 'bookcatalog.db')
# Створюємо необхідні директорії
os.makedirs(BOOKS_DIR, exist_ok=True)
os.makedirs(COVERS_DIR, exist_ok=True)
os.makedirs(IMAGES_DIR, exist_ok=True)
# Шлях до стандартної обкладинки
DEFAULT_COVER_PATH = os.path.join(IMAGES_DIR, 'default_cover.png')
# API ключ для сервісу рекомендацій (Gemini)
# Налаштуйте свій ключ
API_KEY_GEMINI = "AIzaSyBkHBZ5ErdhYAB_YdjGFYwtw7YgMLDKnUc"
try:
    genai.configure(api_key=API_KEY_GEMINI)
except Exception as e:
    logger.warning(f"Не вдалося налаштувати Gemini API: {e}")
# Ініціалізація бази даних
def init_db():
    """Створює необхідні таблиці в базі даних, якщо вони не існують"""
    try:
        conn = sqlite3.connect(DATABASE)
        cursor = conn.cursor()
        # Таблиця книг
        cursor.execute('''
CREATE TABLE IF NOT EXISTS books (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    title TEXT NOT NULL,
    author TEXT,
    year INTEGER,
    genre TEXT,
    description TEXT,
    cover BLOB,
    cover_path TEXT,
    file_path TEXT NOT NULL,
    file_format TEXT,
    series TEXT,
    tags TEXT,
    added_date TEXT,
    last_read_date TEXT,
    current_page INTEGER DEFAULT 0,
    total_pages INTEGER DEFAULT 0,
    read_progress REAL DEFAULT 0,
    reading_status TEXT DEFAULT 'not_started'
)
''')
        # Таблиця закладок
        cursor.execute('''
CREATE TABLE IF NOT EXISTS bookmarks (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    book_id INTEGER,
    position INTEGER,
    name TEXT,
    created_date TEXT,
    FOREIGN KEY (book_id) REFERENCES books (id) ON DELETE CASCADE
)
''')
        # Таблиця цитат
        cursor.execute('''
CREATE TABLE IF NOT EXISTS quotes (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    book_id INTEGER,
    text TEXT,
    position INTEGER,
    created_date TEXT,
    FOREIGN KEY (book_id) REFERENCES books (id) ON DELETE CASCADE
)
''')

```

```

# Таблиця історії читання
cursor.execute('''
CREATE TABLE IF NOT EXISTS reading_history (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    book_id INTEGER,
    page INTEGER,
    timestamp TEXT,
    read_time INTEGER,
    FOREIGN KEY (book_id) REFERENCES books (id) ON DELETE CASCADE
)
''')
conn.commit()
conn.close()
logger.info("База даних успішно ініціалізована")
except Exception as e:
    logger.error(f"Помилка при ініціалізації бази даних: {e}")
# Міграція бази даних
def migrate_database():
    """Оновлює структуру бази даних та виправляє шляхи до файлів"""
    try:
        conn = sqlite3.connect(DATABASE)
        conn.row_factory = sqlite3.Row
        cursor = conn.cursor()
        # Перевіряємо наявність колонки cover_path
        columns = [column[1] for column in cursor.execute("PRAGMA
table_info(books)").fetchall()]
        # Додаємо колонку cover_path, якщо її немає
        if 'cover_path' not in columns:
            logger.info("Додаємо колонку cover_path до таблиці books")
            cursor.execute("ALTER TABLE books ADD COLUMN cover_path TEXT")
            # Перенесення зображень з BLOB в файли
            books = cursor.execute("SELECT id, title, cover FROM books WHERE cover IS NOT
NULL").fetchall()
            for book in books:
                if book['cover']:
                    try:
                        # Створення файлу обкладинки
                        safe_title = re.sub(r'^\w\-\_', '_', book['title'])
                        cover_filename = f"{safe_title}_{book['id']}.jpg"
                        cover_path = os.path.join(COVERS_DIR, cover_filename)
                        with open(cover_path, 'wb') as f:
                            f.write(book['cover'])
                        # Оновлення шляху в БД
                        cursor.execute("UPDATE books SET cover_path = ? WHERE id = ?",
                                      (cover_path, book['id']))
                        logger.info(f"Мігровано обкладинку для книги ID: {book['id']}")
                    except Exception as e:
                        logger.error(f"Помилка збереження обкладинки для книги ID
{book['id']}: {e}")
                        # Виправляємо шляхи з \ на /
                        for book in cursor.execute("SELECT id, cover_path FROM books WHERE cover_path IS NOT
NULL").fetchall():
                            if book['cover_path'] and '\\\' in book['cover_path']:
                                new_path = book['cover_path'].replace('\\', '/')
                                cursor.execute("UPDATE books SET cover_path = ? WHERE id = ?",
                                              (new_path, book['id']))
                                logger.info(f"Виправлено шлях до обкладинки для книги ID: {book['id']}")
            conn.commit()
            conn.close()
            logger.info("Міграцію бази даних завершено")
        except Exception as e:
            logger.error(f"Помилка при міграції бази даних: {e}")
# Створення стандартної обкладинки
def create_default_cover():
    """Створює стандартне зображення для обкладинки"""

```

```

if os.path.exists(DEFAULT_COVER_PATH):
    return DEFAULT_COVER_PATH
try:
    # Створення простого зображення для обкладинки
    img = Image.new('RGB', (300, 450), color=(240, 240, 240))
    draw = ImageDraw.Draw(img)
    # Намалюємо рамку
    draw.rectangle([(10, 10), (290, 440)], outline=(200, 200, 200), width=2)
    # Намалюємо символ книги
    draw.rectangle([(100, 150), (200, 300)], outline=(150, 150, 150), width=3)
    draw.line([(100, 150), (100, 300)], fill=(150, 150, 150), width=3)
    draw.line([(100, 150), (200, 150)], fill=(150, 150, 150), width=3)
    # Збережемо зображення
    img.save(DEFAULT_COVER_PATH)
    logger.info(f"Створено стандартну обкладинку: {DEFAULT_COVER_PATH}")
    return DEFAULT_COVER_PATH
except Exception as e:
    logger.error(f"Помилка створення стандартної обкладинки: {e}")
    return None

# Функції обробки файлів та метаданих
def extract_metadata_from_epub(file_path):
    """Вилучення метаданих з EPUB файлу"""
    try:
        book = epub.read_epub(file_path)
        metadata = {}
        # Назва
        if book.get_metadata('DC', 'title'):
            metadata['title'] = book.get_metadata('DC', 'title')[0][0]
        # Автор
        if book.get_metadata('DC', 'creator'):
            metadata['author'] = book.get_metadata('DC', 'creator')[0][0]
        # Рік
        if book.get_metadata('DC', 'date'):
            year_match = re.search(r'\d{4}', book.get_metadata('DC', 'date')[0][0])
            if year_match:
                metadata['year'] = int(year_match.group())
        # Жанр/категорія
        if book.get_metadata('DC', 'subject'):
            metadata['genre'] = book.get_metadata('DC', 'subject')[0][0]
        # Опис
        if book.get_metadata('DC', 'description'):
            metadata['description'] = book.get_metadata('DC', 'description')[0][0]
        # Отримання загальної кількості сторінок
        pages_count = 0
        for item in book.get_items_of_type(ebooklib.ITEM_DOCUMENT):
            content = item.get_content().decode('utf-8')
            soup = BeautifulSoup(content, 'html.parser')
            # Умовний підрахунок "сторінок" на основі кількості параграфів
            pages_count += len(soup.find_all('p')) // 15 + 1
        metadata['total_pages'] = pages_count
        # Отримання обкладинки
        for item in book.get_items_of_type(ebooklib.ITEM_IMAGE):
            if 'cover' in item.id.lower() or 'title' in item.id.lower():
                metadata['cover'] = item.content
                break
        # Якщо не знайдено обкладинку за ID, спробуємо знайти першу картинку
        if 'cover' not in metadata:
            for item in book.get_items_of_type(ebooklib.ITEM_IMAGE):
                metadata['cover'] = item.content
                break
        return metadata
    except Exception as e:
        logger.error(f"Помилка при читанні EPUB: {e}")
        return {}

def extract_metadata_from_pdf(file_path):

```

```

"""Вилучення метаданих з PDF файлу"""
try:
    doc = fitz.open(file_path)
    metadata = {}
    # Спроба отримати метадані
    pdf_metadata = doc.metadata
    if pdf_metadata.get('title'):
        metadata['title'] = pdf_metadata['title']
    if pdf_metadata.get('author'):
        metadata['author'] = pdf_metadata['author']
    # Спроба отримати рік з дати
    if pdf_metadata.get('creationDate'):
        year_match = re.search(r'D:(\d{4})', pdf_metadata['creationDate'])
        if year_match:
            metadata['year'] = int(year_match.group(1))
    # Загальна кількість сторінок
    metadata['total_pages'] = doc.page_count
    # Спроба отримати обкладинку (перша сторінка)
    if doc.page_count > 0:
        page = doc.load_page(0)
        pix = page.get_pixmap()
        metadata['cover'] = pix.tobytes()
    # Спроба витягти опис з перших сторінок
    text = ""
    for i in range(min(3, doc.page_count)):
        page = doc.load_page(i)
        text += page.get_text()
    if len(text) > 100:
        # Беремо перші 500 символів як опис
        metadata['description'] = text[:500]
    return metadata
except Exception as e:
    logger.error(f"Помилка при читанні PDF: {e}")
    return {}

def extract_book_metadata(file_path):
    """Визначає формат файлу і витягує відповідні метадані"""
    file_format = file_path.split('.')[-1].lower()
    metadata = {}
    try:
        if file_format == 'epub':
            metadata = extract_metadata_from_epub(file_path)
        elif file_format == 'pdf':
            metadata = extract_metadata_from_pdf(file_path)
        elif file_format == 'txt':
            # Для TXT файлів
            filename = os.path.basename(file_path)
            metadata = {
                'title': os.path.splitext(filename)[0],
                'year': datetime.datetime.now().year,
                'total_pages': max(1, os.path.getsize(file_path) // 3000) # Приблизно 3KB
            }
        elif file_format == 'fb2':
            # Для FB2 файлів
            try:
                with open(file_path, 'r', encoding='utf-8', errors='replace') as f:
                    content = f.read()
                    soup = BeautifulSoup(content, 'xml')
                    # Отримання назви
                    title_info = soup.find('title-info')
                    if title_info:
                        book_title = title_info.find('book-title')
                        if book_title:
                            metadata['title'] = book_title.text
                    # Отримання автора

```

```

author = title_info.find('author')
if author:
    first_name = author.find('first-name')
    last_name = author.find('last-name')
    if first_name and last_name:
        metadata['author'] = f"{first_name.text} {last_name.text}"
    elif last_name:
        metadata['author'] = last_name.text
# Отримання жанру
genre = title_info.find('genre')
if genre:
    metadata['genre'] = genre.text
# Отримання року
date = title_info.find('date')
if date:
    year_match = re.search(r'\d{4}', date.text)
    if year_match:
        metadata['year'] = int(year_match.group())
# Отримання опису
annotation = title_info.find('annotation')
if annotation:
    metadata['description'] = annotation.text.strip()
# Отримання обкладинки
cover_page = soup.find('coverpage')
if cover_page:
    image = cover_page.find('image')
    if image and image.has_attr('href'):
        href = image['href']
        href = href.lstrip('#')
        # Знаходимо відповідний бінарний вміст
        binary = soup.find('binary', id=href)
        if binary and binary.has_attr('content-type') and binary.string:
            content_type = binary['content-type']
            if 'image' in content_type:
                try:
                    image_data = base64.b64decode(binary.string)
                    metadata['cover'] = image_data
                except:
                    pass
# Підрахунок сторінок
body = soup.find('body')
if body:
    sections = body.find_all('section')
    paragraphs = sum(len(section.find_all('p')) for section in sections)
    # Приблизно 15 параграфів на сторінку
    metadata['total_pages'] = max(1, paragraphs // 15)
else:
    metadata['total_pages'] = 100 # За умовчанням
except Exception as e:
    logger.error(f"Помилка при читанні FB2: {e}")
    filename = os.path.basename(file_path)
    metadata = {
        'title': os.path.splitext(filename)[0],
        'year': datetime.datetime.now().year,
        'total_pages': 100
    }
else:
    # Для інших форматів
    filename = os.path.basename(file_path)
    metadata = {
        'title': os.path.splitext(filename)[0],
        'year': datetime.datetime.now().year,
        'total_pages': 100
    }
except Exception as e:

```

```

logger.error(f"Помилка при обробці файлу: {e}")
# Базова інформація у випадку помилки
filename = os.path.basename(file_path)
metadata = {
    'title': os.path.splitext(filename)[0],
    'year': datetime.datetime.now().year,
    'total_pages': 100
}
# Заповнюємо відсутні поля значеннями за замовчуванням
if not metadata.get('title'):
    metadata['title'] = os.path.splitext(os.path.basename(file_path))[0]
if not metadata.get('year'):
    metadata['year'] = datetime.datetime.now().year
if not metadata.get('total_pages'):
    metadata['total_pages'] = 100 # Значення за замовчуванням
return metadata, file_format
def save_cover_image(cover_data, title):
    """Зберігає обкладинку як файл і повертає шлях до неї"""
    try:
        if not cover_data:
            logger.warning("Немає даних обкладинки")
            return None
        # Створюємо унікальне ім'я для файлу обкладинки
        safe_title = re.sub(r'[^w\-\_\.]', '_', title)
        cover_filename = f"{safe_title}_{uuid.uuid4().hex[:8]}.jpg"
        cover_path = os.path.join(COVERS_DIR, cover_filename)
        # Зберігаємо обкладинку
        with open(cover_path, 'wb') as f:
            f.write(cover_data)
        logger.info(f"Обкладинку збережено: {cover_path}")
        return cover_path
    except Exception as e:
        logger.error(f"Помилка збереження обкладинки: {e}")
        return None
# Функція для отримання рекомендацій через Gemini API
def get_recommendations_from_ai(query_book=None, tags=None, preferences=None, limit=3):
    """Получает рекомендации книг через Gemini API"""
    try:
        # Проверка API ключа
        if not API_KEY_GEMINI:
            logger.warning("API ключ для Gemini не настроен")
            return []
        # Инициализация клиента
        try:
            genai.configure(api_key=API_KEY_GEMINI)
            # Получаем список доступных моделей
            available_models = [m.name for m in genai.list_models()]
            # Выбираем подходящую модель
            if "models/gemini-1.5-pro" in available_models:
                model = genai.GenerativeModel("gemini-1.5-pro")
            elif "models/gemini-pro" in available_models:
                model = genai.GenerativeModel("gemini-pro")
            else:
                logger.error("Подходящая модель Gemini не найдена")
                return []
        except Exception as config_error:
            logger.error(f"Ошибка конфигурации Gemini: {str(config_error)}")
            return []
        # Формирование промпта
        prompt_parts = [
            "Ты помощник для рекомендации книг. Предоставь ответ в строгом JSON формате:",
            "[{'title': str, 'author': str, 'year': int, 'description': str, 'genre':",
            str}]]",
            "Язык ответа: украинский. Не включай дополнительные комментарии."
        ]
    ]

```

```

if query_book:
    prompt_parts.extend([
        f"Порекомендуй {limit} книг, похожих на:",
        f"Название: {query_book.get('title', 'Неизвестно')}",
        f"Автор: {query_book.get('author', 'Неизвестен')}",
        f"Жанр: {query_book.get('genre', 'Неизвестен')}"
    ])
elif tags:
    prompt_parts.append(f"Порекомендуй {limit} книг по тегам: {tags}")
elif preferences:
    prompt_parts.append(f"Порекомендуй {limit} книг по предпочтениям:
{preferences}")
else:
    prompt_parts.append(f"Порекомендуй {limit} популярных книг разных жанров")
# Выполнение запроса
try:
    response = model.generate_content(prompt_parts)
    text_response = response.text.strip()
    # Очистка ответа от Markdown
    if text_response.startswith("```json"):
        text_response = text_response[7:-3].strip()
    elif text_response.startswith("```"):
        text_response = text_response[3:-3].strip()
    # Парсинг JSON
    books = json.loads(text_response)
    if not isinstance(books, list):
        logger.warning("Получен не список книг")
        return []
    # Добавляем необходимые поля для фронтенда
    for i, book in enumerate(books):
        book['id'] = -i-1 # Отрицательные ID для рекомендаций
        book['cover_url'] = (
            f"https://via.placeholder.com/150x200/607D8B/FFFFFF?"
            f"text={book.get('title', 'Book').replace(' ', '+')}[:10]}")
        book['reading_status'] = 'not_started'
        book['read_progress'] = 0
    return books[:limit]
except json.JSONDecodeError as e:
    logger.error(f"Ошибка парсинга JSON: {str(e)}\nОтвет: {text_response[:500]}")
    return []
except Exception as api_error:
    logger.error(f"Ошибка API: {str(api_error)}")
    return []
except Exception as e:
    logger.error(f"Критическая ошибка: {str(e)}")
    return []
# Створення статичних файлів для використання за замовчуванням
def create_static_files():
    """Створює статичні файли, необхідні для роботи (обкладинки за замовчуванням тощо)"""
    # Створення стандартної обкладинки, якщо вона не існує
    if not os.path.exists(DEFAULT_COVER_PATH):
        create_default_cover()
# Конфігурація та ініціалізація додатка
def setup_app():
    """Налаштування додатку перед запуском"""
    # Створюємо директорії, якщо їх немає
    os.makedirs(STATIC_DIR, exist_ok=True)
    os.makedirs(COVERS_DIR, exist_ok=True)
    os.makedirs(BOOKS_DIR, exist_ok=True)
    os.makedirs(IMAGES_DIR, exist_ok=True)
    # Ініціалізуємо базу даних
    init_db()
    # Міграція даних
    migrate_database()

```

```

# Створюємо статичні файли
create_static_files() # Змінено з create_static_default_files
logger.info(f"Сервер налаштовано. Шляхи до директорій:")
logger.info(f" - Статика: {STATIC_DIR}")
logger.info(f" - Обкладинки: {COVERS_DIR}")
logger.info(f" - Книги: {BOOKS_DIR}")

# Ендпоінти API
# Ендпоінт для отримання статичних файлів
@app.route('/static/<path:filename>')
def serve_static(filename):
    """Віддає статичні файли"""
    return send_from_directory('static', filename)

# Отримання списку всіх книг
@app.route('/api/books', methods=['GET'])
def get_books():
    """Повертає список всіх книг в бібліотеці"""
    try:
        conn = get_db_connection()
        conn.row_factory = dict_factory
        books = conn.execute('''
            SELECT id, title, author, year, genre, series,
                   read_progress, reading_status, current_page, total_pages, tags,
                   cover_path
            FROM books
            ORDER BY title
        ''').fetchall()
        conn.close()
        # Додаємо посилання на обкладинки та перевіряємо їх наявність
        for book in books:
            # Додаємо URL обкладинки
            book['cover_url'] = f"/api/books/{book['id']}/cover"
            # Видаляємо шлях з відповіді
            if 'cover_path' in book:
                del book['cover_path']
        return jsonify(books)
    except Exception as e:
        logger.error(f"Помилка при отриманні списку книг: {e}")
        return jsonify({"error": str(e)}), 500

# Отримання інформації про конкретну книгу
@app.route('/api/books/<int:book_id>', methods=['GET'])
def get_book(book_id):
    """Отримання детальної інформації про книгу"""
    try:
        conn = get_db_connection()
        conn.row_factory = dict_factory
        book = conn.execute('''
            SELECT id, title, author, year, genre, description, series, tags,
                   read_progress, reading_status, current_page, total_pages,
                   file_format, added_date, last_read_date
            FROM books WHERE id = ?
        ''', (book_id,)).fetchone()
        conn.close()
        if book is None:
            return jsonify({"error": "Книгу не знайдено"}), 404
        # Додаємо посилання на обкладинку і файл
        book['cover_url'] = f"/api/books/{book['id']}/cover"
        book['file_url'] = f"/api/books/{book['id']}/file"
        return jsonify(book)
    except Exception as e:
        logger.error(f"Помилка при отриманні книги: {e}")
        return jsonify({"error": str(e)}), 500

# Додавання нової книги
@app.route('/api/books', methods=['POST'])
def add_book():
    """Додає нову книгу в бібліотеку"""

```

```

try:
    # Перевіряємо наявність файлу
    if 'file' not in request.files:
        return jsonify({"error": "Файл книги не надіслано"}), 400
    file = request.files['file']
    if file.filename == '':
        return jsonify({"error": "Порожнє ім'я файлу"}), 400
    # Зберігаємо файл у тимчасову директорію
    temp_dir = tempfile.mkdtemp()
    temp_file_path = os.path.join(temp_dir, file.filename)
    file.save(temp_file_path)
    # Отримуємо метадані з файлу
    metadata, file_format = extract_book_metadata(temp_file_path)
    # Отримуємо дані з форми
    title = request.form.get('title') or metadata.get('title', '')
    author = request.form.get('author') or metadata.get('author', '')
    year_str = request.form.get('year')
    year = int(year_str) if year_str else metadata.get('year',
datetime.datetime.now().year)
    genre = request.form.get('genre') or metadata.get('genre', '')
    description = request.form.get('description') or metadata.get('description', '')
    series = request.form.get('series', '')
    tags = request.form.get('tags', '')
    total_pages = metadata.get('total_pages', 0)
    # Обробка обкладинки
    cover_data = None
    cover_path = None
    # 1. Спочатку перевіряємо файл обкладинки
    if 'cover' in request.files and request.files['cover'].filename:
        cover_file = request.files['cover']
        cover_data = cover_file.read()
        logger.info(f"Отримано обкладинку з файлу, розмір: {len(cover_data)} байт")
    # 2. Далі перевіряємо URL обкладинки
    elif 'cover_url' in request.form and request.form['cover_url']:
        url = request.form['cover_url']
        try:
            response = requests.get(url, timeout=5)
            if response.status_code == 200 and 'image' in response.headers.get('Content-
Type', ''):
                cover_data = response.content
                logger.info(f"Завантажено обкладинку з URL, розмір: {len(cover_data)}
байт")
        except Exception as e:
            logger.error(f"Помилка завантаження обкладинки з URL: {e}")
    # 3. Нарешті, використовуємо обкладинку з метаданих
    elif 'cover' in metadata:
        cover_data = metadata['cover']
        logger.info(f"Використовуємо обкладинку з метаданих, розмір: {len(cover_data)}
байт")
    # Зберігаємо обкладинку як файл, якщо вона є
    if cover_data:
        cover_path = save_cover_image(cover_data, title)
        logger.info(f"Збережено обкладинку: {cover_path}")
    # Зберігаємо файл книги у постійне місце
    unique_id = uuid.uuid4().hex[:8]
    safe_title = re.sub(r'[^\w\-\_\.]', '_', title)
    final_filename = f"{unique_id}_{safe_title}.{file_format}"
    final_path = os.path.join(BOOKS_DIR, final_filename)
    shutil.copy2(temp_file_path, final_path)
    logger.info(f"Збережено файл книги: {final_path}")
    # Видаляємо тимчасовий файл
    try:
        os.remove(temp_file_path)
        os.rmdir(temp_dir)
    except:

```

```

        pass
    # Додаємо книгу до бази даних
    conn = get_db_connection()
    now = datetime.datetime.now().isoformat()
    cursor = conn.execute('''
        INSERT INTO books (
            title, author, year, genre, description, cover_path,
            file_path, file_format, series, tags, added_date,
            reading_status, total_pages
        ) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
    ''', (
        title, author, year, genre, description, cover_path,
        final_path, file_format, series, tags, now,
        'not_started', total_pages
    ))
    book_id = cursor.lastrowid
    conn.commit()
    conn.close()
    return jsonify({
        "id": book_id,
        "message": "Книгу успішно додано"
    }), 201
except Exception as e:
    logger.error(f"Помилка при додаванні книги: {e}")
    return jsonify({"error": str(e)}), 500

# Редагування книги
@app.route('/api/books/<int:book_id>', methods=['PUT'])
def update_book(book_id):
    """Оновлює інформацію про книгу"""
    try:
        # Перевіряємо, чи існує книга
        conn = get_db_connection()
        book = conn.execute('SELECT cover_path FROM books WHERE id = ?',
            (book_id,)).fetchone()
        if not book:
            conn.close()
            return jsonify({"error": "Книгу не знайдено"}), 404
        # Отримуємо дані для оновлення
        data = request.form.to_dict()
        # Обробка обкладинки, якщо вона є
        if 'cover' in request.files and request.files['cover'].filename:
            cover_file = request.files['cover']
            cover_data = cover_file.read()
            # Видаляємо стару обкладинку, якщо вона існує
            old_cover_path = book['cover_path']
            if old_cover_path and os.path.exists(old_cover_path):
                try:
                    os.remove(old_cover_path)
                except Exception as e:
                    logger.error(f"Помилка видалення старої обкладинки: {e}")
            # Зберігаємо нову обкладинку
            title = data.get('title', '') or conn.execute('SELECT title FROM books WHERE id
                = ?', (book_id,)).fetchone()['title']
            cover_path = save_cover_image(cover_data, title)
            if cover_path:
                data['cover_path'] = cover_path
        # Виключаємо поля, які не можна змінити напряму
        excluded_fields = ['id', 'file_path', 'file_format', 'added_date']
        for field in excluded_fields:
            if field in data:
                del data[field]
        # Оновлюємо дані, якщо вони є
        if data:
            fields = ', '.join([f"{key} = ?" for key in data.keys()])
            values = list(data.values())

```

```

        values.append(book_id)
        conn.execute(f"UPDATE books SET {fields} WHERE id = ?", values)
        conn.commit()
    conn.close()
    return jsonify({"message": "Книгу успішно оновлено"}), 200
except Exception as e:
    logger.error(f"Помилка при оновленні книги: {e}")
    return jsonify({"error": str(e)}), 500
# Видалення книги
@app.route('/api/books/<int:book_id>', methods=['DELETE'])
def delete_book(book_id):
    """Видаляє книгу та пов'язані файли"""
    try:
        conn = get_db_connection()
        conn.row_factory = dict_factory
        # Отримуємо інформацію про книгу
        book = conn.execute('SELECT file_path, cover_path FROM books WHERE id = ?',
        (book_id,)).fetchone()
        if not book:
            conn.close()
            return jsonify({"error": "Книгу не знайдено"}), 404
        # Видаляємо файл книги, якщо він існує
        if book['file_path'] and os.path.exists(book['file_path']):
            try:
                os.remove(book['file_path'])
                logger.info(f"Видалено файл книги: {book['file_path']}")
            except Exception as e:
                logger.error(f"Помилка видалення файлу книги: {e}")
        # Видаляємо файл обкладинки, якщо він існує
        if book['cover_path'] and os.path.exists(book['cover_path']):
            try:
                os.remove(book['cover_path'])
                logger.info(f"Видалено файл обкладинки: {book['cover_path']}")
            except Exception as e:
                logger.error(f"Помилка видалення файлу обкладинки: {e}")
        # Видаляємо запис з бази даних та пов'язані дані
        conn.execute('DELETE FROM bookmarks WHERE book_id = ?', (book_id,))
        conn.execute('DELETE FROM quotes WHERE book_id = ?', (book_id,))
        conn.execute('DELETE FROM reading_history WHERE book_id = ?', (book_id,))
        conn.execute('DELETE FROM books WHERE id = ?', (book_id,))
        conn.commit()
        conn.close()
        return jsonify({"message": "Книгу успішно видалено"})
    except Exception as e:
        logger.error(f"Помилка при видаленні книги: {e}")
        return jsonify({"error": str(e)}), 500
# Отримання обкладинки книги
@app.route('/api/books/<int:book_id>/cover', methods=['GET'])
def get_book_cover(book_id):
    """Повертає обкладинку книги"""
    try:
        conn = sqlite3.connect(DATABASE)
        conn.row_factory = lambda cursor, row: {col[0]: row[idx] for idx, col in
        enumerate(cursor.description)}
        # Отримуємо інформацію про книгу та обкладинку
        book = conn.execute('SELECT cover_path, cover FROM books WHERE id = ?',
        (book_id,)).fetchone()
        conn.close()
        if not book:
            logger.warning(f"Книгу з ID {book_id} не знайдено")
            return send_file(DEFAULT_COVER_PATH, mimetype='image/png')
        # Перевірка шляху до обкладинки
        if book.get('cover_path') and os.path.exists(book['cover_path']):
            logger.info(f"Повертаємо обкладинку за шляхом: {book['cover_path']}")
            return send_file(book['cover_path'], mimetype='image/jpeg')

```

```

# Перевірка BLOB даних
elif book.get('cover'):
    logger.info(f"Повертаємо обкладинку з BLOB для книги {book_id}")
    # Спробуємо зберегти BLOB як файл для подальшого використання
    try:
        book_title = conn.execute('SELECT title FROM books WHERE id = ?',
                                   (book_id,)).fetchone()['title']
        cover_path = save_cover_image(book['cover'], book_title)
        if cover_path:
            # Оновлюємо шлях в базі даних
            conn = sqlite3.connect(DATABASE)
            conn.execute('UPDATE books SET cover_path = ? WHERE id = ?',
                          (cover_path, book_id))
            conn.commit()
            conn.close()
            return send_file(cover_path, mimetype='image/jpeg')
        except Exception as e:
            logger.error(f"Помилка збереження BLOB як файл: {e}")
            # Якщо не вдалося зберегти, повертаємо BLOB напряму
            return send_file(BytesIO(book['cover']), mimetype='image/jpeg')
            # Якщо немає ні шляху, ні BLOB - повертаємо стандартну обкладинку
            logger.warning(f"Обкладинку не знайдено для книги {book_id}")
            return send_file(DEFAULT_COVER_PATH, mimetype='image/png')
        except Exception as e:
            logger.error(f"Помилка при отриманні обкладинки: {e}")
            return send_file(DEFAULT_COVER_PATH, mimetype='image/png')
@app.route('/api/books/<int:book_id>/extract_cover', methods=['POST'])
def extract_book_cover(book_id):
    """Витягує обкладинку з файлу книги та зберігає її"""
    try:
        conn = sqlite3.connect(DATABASE)
        conn.row_factory = lambda cursor, row: {col[0]: row[idx] for idx, col in
        enumerate(cursor.description)}
        # Отримуємо інформацію про книгу
        book = conn.execute('SELECT id, title, file_path, file_format FROM books WHERE id =
        ?', (book_id,)).fetchone()
        if not book:
            conn.close()
            return jsonify({"error": "Книгу не знайдено"}), 404
        if not os.path.exists(book['file_path']):
            conn.close()
            return jsonify({"error": "Файл книги не знайдено"}), 404
        # Витягаємо метадані з файлу
        metadata, _ = extract_book_metadata(book['file_path'])
        # Перевіряємо наявність обкладинки у метаданих
        if 'cover' not in metadata or not metadata['cover']:
            conn.close()
            return jsonify({"error": "Обкладинку не знайдено у файлі"}), 404
        # Зберігаємо обкладинку
        cover_path = save_cover_image(metadata['cover'], book['title'])
        if not cover_path:
            conn.close()
            return jsonify({"error": "Не вдалося зберегти обкладинку"}), 500
        # Оновлюємо шлях до обкладинки в базі даних
        conn.execute('UPDATE books SET cover_path = ? WHERE id = ?', (cover_path, book_id))
        conn.commit()
        conn.close()
        return jsonify({
            "success": True,
            "message": "Обкладинку успішно витягнуто та збережено",
            "cover_path": cover_path
        })
    except Exception as e:
        logger.error(f"Помилка при витягненні обкладинки: {e}")
        return jsonify({"error": str(e)}), 500

```

```

@app.route('/api/maintenance/restore_covers', methods=['GET'])
def restore_all_covers():
    """Спроба відновити обкладинки для всіх книг"""
    try:
        conn = sqlite3.connect(DATABASE)
        conn.row_factory = lambda cursor, row: {col[0]: row[idx] for idx, col in
enumerate(cursor.description)}
        # Отримуємо всі книги без обкладинок або з відсутніми файлами
        books = conn.execute('''
            SELECT id, title, file_path
            FROM books
            WHERE cover_path IS NULL OR cover_path = "" OR cover IS NULL
        ''').fetchall()
        results = {
            "total_books": len(books),
            "success": 0,
            "failed": 0,
            "details": []
        }
        for book in books:
            try:
                if not os.path.exists(book['file_path']):
                    results["details"].append({
                        "id": book['id'],
                        "title": book['title'],
                        "status": "failed",
                        "reason": "Файл книги не знайдено"
                    })
                    results["failed"] += 1
                    continue
                # Витягаємо метадані з файлу
                metadata, _ = extract_book_metadata(book['file_path'])
                # Перевіряємо наявність обкладинки у метаданих
                if 'cover' not in metadata or not metadata['cover']:
                    results["details"].append({
                        "id": book['id'],
                        "title": book['title'],
                        "status": "failed",
                        "reason": "Обкладинку не знайдено у файлі"
                    })
                    results["failed"] += 1
                    continue
                # Зберігаємо обкладинку
                cover_path = save_cover_image(metadata['cover'], book['title'])
                if not cover_path:
                    results["details"].append({
                        "id": book['id'],
                        "title": book['title'],
                        "status": "failed",
                        "reason": "Не вдалося зберегти обкладинку"
                    })
                    results["failed"] += 1
                    continue
                # Оновлюємо шлях до обкладинки в базі даних
                conn.execute('UPDATE books SET cover_path = ? WHERE id = ?', (cover_path,
book['id']))
                results["details"].append({
                    "id": book['id'],
                    "title": book['title'],
                    "status": "success",
                    "cover_path": cover_path
                })
                results["success"] += 1
            except Exception as e:
                results["details"].append({

```

```

        "id": book['id'],
        "title": book['title'],
        "status": "failed",
        "reason": str(e)
    })
    results["failed"] += 1
    conn.commit()
    conn.close()
    return jsonify(results)
except Exception as e:
    logger.error(f"Помилка при відновленні обкладинок: {e}")
    return jsonify({"error": str(e)}), 500
# Отримання файлу книги для читання
@app.route('/api/books/<int:book_id>/file', methods=['GET'])
def get_book_file(book_id):
    """Повертає файл книги для читання"""
    try:
        conn = get_db_connection()
        conn.row_factory = dict_factory
        # Отримуємо інформацію про файл книги
        book = conn.execute('SELECT file_path, file_format, title FROM books WHERE id = ?',
(book_id,)).fetchone()
        conn.close()
        if not book:
            return jsonify({"error": "Книгу не знайдено"}), 404
        # Перевіряємо, чи існує файл
        if not os.path.exists(book['file_path']):
            return jsonify({"error": "Файл книги не знайдено"}), 404
        # Визначаємо MIME тип файлу
        mime_types = {
            'pdf': 'application/pdf',
            'epub': 'application/epub+zip',
            'txt': 'text/plain',
            'fb2': 'application/xml',
            'mobi': 'application/x-mobipocket-ebook'
        }
        mimetype = mime_types.get(book['file_format'], 'application/octet-stream')
        # Повертаємо файл
        return send_file(
            book['file_path'],
            mimetype=mimetype,
            as_attachment=False
        )
    except Exception as e:
        logger.error(f"Помилка при отриманні файлу книги: {e}")
        return jsonify({"error": str(e)}), 500
# Завантаження файлу книги (для збереження)
@app.route('/api/books/<int:book_id>/download', methods=['GET'])
def download_book_file(book_id):
    """Повертає файл книги для завантаження"""
    try:
        conn = get_db_connection()
        conn.row_factory = dict_factory
        # Отримуємо інформацію про файл книги
        book = conn.execute('SELECT file_path, file_format, title FROM books WHERE id = ?',
(book_id,)).fetchone()
        conn.close()
        if not book:
            return jsonify({"error": "Книгу не знайдено"}), 404
        # Перевіряємо, чи існує файл
        if not os.path.exists(book['file_path']):
            return jsonify({"error": "Файл книги не знайдено"}), 404
        # Визначаємо MIME тип файлу
        mime_types = {
            'pdf': 'application/pdf',

```

```

        'epub': 'application/epub+zip',
        'txt': 'text/plain',
        'fb2': 'application/xml',
        'mobi': 'application/x-mobipocket-ebook'
    }
    mimetype = mime_types.get(book['file_format'], 'application/octet-stream')
    # Формуємо безпечну назву файлу для завантаження
    safe_title = re.sub(r'^\w\_-.', '_', book['title'])
    download_name = f"{safe_title}.{book['file_format']}"
    # Повертаємо файл як вкладення
    return send_file(
        book['file_path'],
        mimetype=mimetype,
        as_attachment=True,
        download_name=download_name
    )
except Exception as e:
    logger.error(f"Помилка при завантаженні файлу книги: {e}")
    return jsonify({"error": str(e)}), 500
# Отримання вмісту книги для відображення в онлайн-читалці
@app.route('/api/books/<int:book_id>/content', methods=['GET'])
def get_book_content(book_id):
    """Повертає вміст книги для онлайн-читання"""
    try:
        conn = get_db_connection()
        conn.row_factory = dict_factory
        # Отримуємо інформацію про книгу
        book = conn.execute('SELECT file_path, file_format FROM books WHERE id = ?',
                             (book_id,)).fetchone()
        conn.close()
        if not book:
            return jsonify({"error": "Книгу не знайдено"}), 404
        # Перевіряємо, чи існує файл
        if not os.path.exists(book['file_path']):
            return jsonify({"error": "Файл книги не знайдено"}), 404
        # Обробка різних форматів книг
        file_format = book['file_format'].lower()
        content = {"pages": [], "total_pages": 0}
        if file_format == 'pdf':
            # Обробка PDF файлу
            doc = fitz.open(book['file_path'])
            # Отримуємо вміст кожної сторінки
            for page_num in range(doc.page_count):
                page = doc.load_page(page_num)
                page_text = page.get_text()
                # Створюємо HTML-представлення сторінки
                page_html = f"<div class='pdf-page'>{page_text.replace('\n', '<br>')}</div>"
                content["pages"].append(page_html)
            content["total_pages"] = doc.page_count
        elif file_format == 'epub':
            # Обробка EPUB файлу
            book_content = epub.read_epub(book['file_path'])
            # Отримуємо вміст документів
            for item in book_content.get_items_of_type(epublib.ITEM_DOCUMENT):
                html_content = item.get_content().decode('utf-8', errors='ignore')
                # Очищення HTML
                soup = BeautifulSoup(html_content, 'html.parser')
                body = soup.find('body')
                if body:
                    # Розбиваємо текст на "сторінки"
                    paragraphs = body.find_all(['p', 'div', 'h1', 'h2', 'h3', 'h4', 'h5',
                                                'h6'])
                    # Групуємо параграфи по 15 в одну "сторінку"
                    for i in range(0, len(paragraphs), 15):
                        page_content = "".join(str(p) for p in paragraphs[i:i+15])

```

```

        content["pages"].append(page_content)
    content["total_pages"] = len(content["pages"])
elif file_format == 'txt':
    # Обробка TXT файлу
    with open(book['file_path'], 'r', encoding='utf-8', errors='ignore') as f:
        text = f.read()
        # Розбиваємо на рядки
        lines = text.split('\n')
        # Групуємо рядки по 40 в одну "сторінку"
        for i in range(0, len(lines), 40):
            page_content = "<br>".join(lines[i:i+40])
            content["pages"].append(f"<div class='txt-page'>{page_content}</div>")
            content["total_pages"] = len(content["pages"])
elif file_format == 'fb2':
    # Обробка FB2 файлу
    try:
        with open(book['file_path'], 'r', encoding='utf-8', errors='ignore') as f:
            fb2_content = f.read()
            # Парсимо XML
            soup = BeautifulSoup(fb2_content, 'xml')
            # Отримуємо текст із секцій body
            paragraphs = []
            for body in soup.find_all('body'):
                if body.get('name') != 'notes': # Пропускаємо примітки
                    for section in body.find_all('section'):
                        paragraphs.extend(section.find_all('p'))
            # Групуємо параграфи по 15 в одну "сторінку"
            for i in range(0, len(paragraphs), 15):
                page_content = "".join(str(p) for p in paragraphs[i:i+15])
                content["pages"].append(page_content)
                content["total_pages"] = len(content["pages"])
    except Exception as e:
        logger.error(f"Помилка при обробці FB2: {e}")
        content["error"] = "Помилка читання FB2 файлу"
else:
    content["error"] = f"Формат {file_format} не підтримується для онлайн-читання"
# Оновлюємо загальну кількість сторінок в БД
if content["total_pages"] > 0:
    conn = get_db_connection()
    conn.execute('UPDATE books SET total_pages = ? WHERE id = ?',
                  (content["total_pages"], book_id))
    conn.commit()
    conn.close()
    return jsonify(content)
except Exception as e:
    logger.error(f"Помилка при отриманні вмісту книги: {e}")
    return jsonify({"error": str(e)}), 500
# Оновлення прогресу читання
@app.route('/api/books/<int:book_id>/progress', methods=['PUT'])
def update_progress(book_id):
    """Оновлює прогрес читання книги"""
    try:
        data = request.json
        if not data or 'current_page' not in data:
            return jsonify({"error": "Не вказано поточну сторінку"}), 400
        current_page = int(data['current_page'])
        read_time = int(data.get('read_time', 0)) # Час читання в секундах
        conn = get_db_connection()
        # Отримуємо інформацію про книгу
        book = conn.execute('SELECT total_pages FROM books WHERE id = ?',
                             (book_id,)).fetchone()
        if not book:
            conn.close()
            return jsonify({"error": "Книгу не знайдено"}), 404
        # Розрахунок прогресу у відсотках

```

```

total_pages = max(1, book['total_pages']) # Запобігає діленню на нуль
progress = min(100, (current_page / total_pages) * 100)
# Визначення статусу читання
if progress <= 0:
    reading_status = 'not_started'
elif progress < 100:
    reading_status = 'reading'
else:
    reading_status = 'completed'
# Оновлюємо інформацію в БД
conn.execute('''
    UPDATE books
    SET read_progress = ?, reading_status = ?, current_page = ?, last_read_date = ?
    WHERE id = ?
''', (progress, reading_status, current_page, datetime.datetime.now().isoformat(),
book_id))
# Додаємо запис в історію читання
if read_time > 0:
    conn.execute('''
        INSERT INTO reading_history (book_id, page, timestamp, read_time)
        VALUES (?, ?, ?, ?)
    ''', (book_id, current_page, datetime.datetime.now().isoformat(), read_time))
conn.commit()
conn.close()
return jsonify({
    "message": "Прогрес успішно оновлено",
    "progress": progress,
    "status": reading_status,
    "current_page": current_page
})
except Exception as e:
    logger.error(f"Помилка при оновленні прогресу: {e}")
    return jsonify({"error": str(e)}), 500
# Робота з закладками
@app.route('/api/books/<int:book_id>/bookmarks', methods=['GET'])
def get_bookmarks(book_id):
    """Отримує всі закладки для книги"""
    try:
        conn = get_db_connection()
        conn.row_factory = dict_factory
        bookmarks = conn.execute('''
            SELECT * FROM bookmarks
            WHERE book_id = ?
            ORDER BY position
        ''', (book_id,)).fetchall()
        conn.close()
        return jsonify(bookmarks)
    except Exception as e:
        logger.error(f"Помилка при отриманні закладок: {e}")
        return jsonify({"error": str(e)}), 500
@app.route('/api/books/<int:book_id>/bookmarks', methods=['POST'])
def add_bookmark(book_id):
    """Додає нову закладку"""
    try:
        data = request.json
        if not data or 'position' not in data:
            return jsonify({"error": "Не вказано позицію закладки"}), 400
        position = int(data['position'])
        name = data.get('name', f"Сторінка {position + 1}")
        conn = get_db_connection()
        # Перевіряємо, чи існує книга
        book = conn.execute('SELECT id FROM books WHERE id = ?', (book_id,)).fetchone()
        if not book:
            conn.close()
            return jsonify({"error": "Книгу не знайдено"}), 404

```

```

# Додаємо закладку
cursor = conn.execute('''
    INSERT INTO bookmarks (book_id, position, name, created_date)
    VALUES (?, ?, ?, ?)
''', (book_id, position, name, datetime.datetime.now().isoformat()))
bookmark_id = cursor.lastrowid
conn.commit()
conn.close()
return jsonify({
    "id": bookmark_id,
    "message": "Закладку додано"
}), 201
except Exception as e:
    logger.error(f"Помилка при додаванні закладки: {e}")
    return jsonify({"error": str(e)}), 500
@app.route('/api/bookmarks/<int:bookmark_id>', methods=['DELETE'])
def delete_bookmark(bookmark_id):
    """Видаляє закладку"""
    try:
        conn = get_db_connection()
        # Перевіряємо, чи існує закладка
        bookmark = conn.execute('SELECT id FROM bookmarks WHERE id = ?',
(bookmark_id,)).fetchone()
        if not bookmark:
            conn.close()
            return jsonify({"error": "Закладку не знайдено"}), 404
        # Видаляємо закладку
        conn.execute('DELETE FROM bookmarks WHERE id = ?', (bookmark_id,))
        conn.commit()
        conn.close()
        return jsonify({"message": "Закладку видалено"})
    except Exception as e:
        logger.error(f"Помилка при видаленні закладки: {e}")
        return jsonify({"error": str(e)}), 500
# Робота з цитатами
@app.route('/api/books/<int:book_id>/quotes', methods=['GET'])
def get_quotes(book_id):
    """Отримує всі цитати для книги"""
    try:
        conn = get_db_connection()
        conn.row_factory = dict_factory
        quotes = conn.execute('''
            SELECT * FROM quotes
            WHERE book_id = ?
            ORDER BY created_date DESC
        ''', (book_id,)).fetchall()
        conn.close()
        return jsonify(quotes)
    except Exception as e:
        logger.error(f"Помилка при отриманні цитат: {e}")
        return jsonify({"error": str(e)}), 500
@app.route('/api/books/<int:book_id>/quotes', methods=['POST'])
def add_quote(book_id):
    """Додає нову цитату"""
    try:
        data = request.json
        if not data or 'text' not in data:
            return jsonify({"error": "Не вказано текст цитати"}), 400
        text = data['text']
        position = int(data.get('position', 0))
        conn = get_db_connection()
        # Перевіряємо, чи існує книга
        book = conn.execute('SELECT id FROM books WHERE id = ?', (book_id,)).fetchone()
        if not book:
            conn.close()

```

```

        return jsonify({"error": "Книгу не знайдено"}), 404
    # Додаємо цитату
    cursor = conn.execute('''
        INSERT INTO quotes (book_id, text, position, created_date)
        VALUES (?, ?, ?, ?)
    ''', (book_id, text, position, datetime.datetime.now().isoformat()))
    quote_id = cursor.lastrowid
    conn.commit()
    conn.close()
    return jsonify({
        "id": quote_id,
        "message": "Цитату додано"
    }), 201
except Exception as e:
    logger.error(f"Помилка при додаванні цитати: {e}")
    return jsonify({"error": str(e)}), 500
@app.route('/api/quotes/<int:quote_id>', methods=['DELETE'])
def delete_quote(quote_id):
    """Видаляє цитату"""
    try:
        conn = get_db_connection()
        # Перевіряємо, чи існує цитата
        quote = conn.execute('SELECT id FROM quotes WHERE id = ?', (quote_id,)).fetchone()
        if not quote:
            conn.close()
            return jsonify({"error": "Цитату не знайдено"}), 404
        # Видаляємо цитату
        conn.execute('DELETE FROM quotes WHERE id = ?', (quote_id,))
        conn.commit()
        conn.close()
        return jsonify({"message": "Цитату видалено"})
    except Exception as e:
        logger.error(f"Помилка при видаленні цитати: {e}")
        return jsonify({"error": str(e)}), 500
# Отримання рекомендацій за допомогою Gemini API
@app.route('/api/recommendations', methods=['GET'])
def get_recommendations():
    """Повертає рекомендації на основі книги або тегів"""
    try:
        book_id = request.args.get('book_id', type=int)
        tags = request.args.get('tags')
        preferences = request.args.get('preferences')
        # Логіка рекомендацій
        if book_id:
            # Рекомендації на основі книги
            conn = get_db_connection()
            conn.row_factory = dict_factory
            # Отримуємо інформацію про вибрану книгу
            book = conn.execute('''
                SELECT title, author, genre, description, tags
                FROM books WHERE id = ?
            ''', (book_id,)).fetchone()
            conn.close()
            if not book:
                return jsonify({"error": "Книгу не знайдено"}), 404
            # Отримуємо рекомендації від API
            recommendations = get_recommendations_from_ai(book, None, None)
        elif tags:
            # Рекомендації за тегами
            recommendations = get_recommendations_from_ai(None, tags, None)
        elif preferences:
            # Рекомендації за вподобаннями
            recommendations = get_recommendations_from_ai(None, None, preferences)
        else:
            # Загальні рекомендації

```

```

        recommendations = get_recommendations_from_ai()
        # Додаємо URL для обкладинок (тестові дані)
        for book in recommendations:
            book['cover_url'] =
f"https://via.placeholder.com/150x200/607D8B/FFFFFF?text={book.get('title',
'Book').replace(' ', '+')}"
            # Додаємо ID для фронтенду (від'ємні для рекомендацій)
            book['id'] = -hash(book.get('title', '') + book.get('author', '')) % 10000
        return jsonify(recommendations)
    except Exception as e:
        logger.error(f"Помилка при отриманні рекомендацій: {e}")
        return jsonify({"error": str(e)}), 500
# Отримання статистики читання
@app.route('/api/stats', methods=['GET'])
def get_statistics():
    """Повертає статистику читання"""
    try:
        conn = get_db_connection()
        conn.row_factory = dict_factory
        # Загальна кількість книг
        total_books = conn.execute('SELECT COUNT(*) as count FROM
books').fetchone()['count']
        # Книги за статусом
        completed_books = conn.execute('SELECT COUNT(*) as count FROM books WHERE
reading_status = "completed"').fetchone()['count']
        reading_books = conn.execute('SELECT COUNT(*) as count FROM books WHERE
reading_status = "reading"').fetchone()['count']
        not_started_books = conn.execute('SELECT COUNT(*) as count FROM books WHERE
reading_status = "not_started"').fetchone()['count']
        # Середній прогрес
        avg_progress = conn.execute('SELECT AVG(read_progress) as avg FROM books WHERE
reading_status = "reading"').fetchone()['avg'] or 0
        # Розподіл за жанрами
        genres = conn.execute('''
            SELECT genre, COUNT(*) as count
            FROM books
            WHERE genre IS NOT NULL AND genre != ""
            GROUP BY genre
            ORDER BY count DESC
        ''').fetchall()
        # Розподіл за авторами
        authors = conn.execute('''
            SELECT author, COUNT(*) as count
            FROM books
            WHERE author IS NOT NULL AND author != ""
            GROUP BY author
            ORDER BY count DESC
            LIMIT 10
        ''').fetchall()
        # Розподіл за роками
        years = conn.execute('''
            SELECT year, COUNT(*) as count
            FROM books
            GROUP BY year
            ORDER BY year
        ''').fetchall()
        # Статистика за останній місяць
        month_ago = (datetime.datetime.now() - datetime.timedelta(days=30)).isoformat()
        # Книги, додані за останній місяць
        recent_books = conn.execute('''
            SELECT COUNT(*) as count
            FROM books
            WHERE added_date > ?
        ''', (month_ago,)).fetchone()['count']
        # Книги, прочитані за останній місяць

```

```

recently_completed = conn.execute('''
    SELECT COUNT(*) as count
    FROM books
    WHERE reading_status = "completed" AND last_read_date > ?
''', (month_ago,)).fetchone()['count']
# Час читання за останній місяць
total_read_time = conn.execute('''
    SELECT SUM(read_time) as total
    FROM reading_history
    WHERE timestamp > ?
''', (month_ago,)).fetchone()['total'] or 0
# Перетворюємо секунди в години
total_read_hours = total_read_time / 3600
conn.close()
# Формуємо відповідь
return jsonify({
    'total_books': total_books,
    'completed_books': completed_books,
    'reading_books': reading_books,
    'not_started_books': not_started_books,
    'avg_progress': round(avg_progress, 2),
    'genres': genres,
    'authors': authors,
    'years': years,
    'recent_books': recent_books,
    'recently_completed': recently_completed,
    'total_read_hours': round(total_read_hours, 1)
})
except Exception as e:
    logger.error(f"Помилка при отриманні статистики: {e}")
    return jsonify({"error": str(e)}), 500
# Витягнення метаданих з файлу
@app.route('/api/extract_metadata', methods=['POST'])
def extract_metadata_endpoint():
    """Витягнення метаданих з файлу книги"""
    temp_file = None
    try:
        if 'file' not in request.files:
            return jsonify({'error': 'Файл не надіслано'}), 400
        file = request.files['file']
        if file.filename == '':
            return jsonify({'error': 'Порожнє ім'я файлу'}), 400
        # Створюємо тимчасовий файл
        temp_dir = tempfile.mkdtemp()
        file_path = os.path.join(temp_dir, file.filename)
        file.save(file_path)
        # Отримуємо метадані з файлу
        try:
            metadata, file_format = extract_book_metadata(file_path)
            # Якщо є обкладинка, кодуємо її в base64
            if 'cover' in metadata and metadata['cover']:
                cover_base64 = base64.b64encode(metadata['cover']).decode('utf-8')
                metadata['cover_base64'] = f"data:image/jpeg;base64,{cover_base64}"
                del metadata['cover'] # Видаляємо бінарні дані
            # Додаємо формат файлу до відповіді
            metadata['file_format'] = file_format
        except Exception as e:
            logger.error(f"Помилка при витягненні метаданих: {str(e)}")
            return jsonify({'error': str(e)}), 500
        finally:
            # Видаляємо тимчасовий файл
            try:
                if os.path.exists(file_path):
                    os.remove(file_path)
                os.rmdir(temp_dir)
            
```

```

        except Exception as e:
            logger.error(f"Помилка видалення тимчасового файлу: {e}")
        return jsonify(metadata), 200
    except Exception as e:
        logger.error(f"Помилка при витягненні метаданих: {str(e)}")
        return jsonify({'error': str(e)}), 500

# Пошук книг
@app.route('/api/search', methods=['GET'])
def search_books():
    """Пошук книг за різними критеріями"""
    try:
        # Отримуємо параметри пошуку
        query = request.args.get('query', '')
        author = request.args.get('author', '')
        genre = request.args.get('genre', '')
        year = request.args.get('year', type=int)
        status = request.args.get('status', '')
        # Будуємо SQL запит
        sql = '''
            SELECT id, title, author, year, genre, series,
                   read_progress, reading_status, current_page, total_pages, tags
            FROM books
            WHERE 1=1
            ...
        '''
        params = []
        # Додаємо умови фільтрації
        if query:
            sql += ' AND (title LIKE ? OR description LIKE ? OR tags LIKE ?)'
            query_param = f'%{query}%'
            params.extend([query_param, query_param, query_param])
        if author:
            sql += ' AND author LIKE ?'
            params.append(f'%{author}%')
        if genre:
            sql += ' AND genre LIKE ?'
            params.append(f'%{genre}%')
        if year:
            sql += ' AND year = ?'
            params.append(year)
        if status:
            sql += ' AND reading_status = ?'
            params.append(status)
        # Виконуємо запит
        conn = get_db_connection()
        conn.row_factory = dict_factory
        books = conn.execute(sql, params).fetchall()
        conn.close()
        # Додаємо URL обкладинок
        for book in books:
            book['cover_url'] = f"/api/books/{book['id']}/cover"
        return jsonify(books)
    except Exception as e:
        logger.error(f"Помилка при пошуку книг: {e}")
        return jsonify({"error": str(e)}), 500

# Отримання історії читання
@app.route('/api/reading_history/<int:book_id>', methods=['GET'])
def get_reading_history(book_id):
    """Отримання історії читання для книги"""
    try:
        conn = get_db_connection()
        conn.row_factory = dict_factory
        history = conn.execute('''
            SELECT id, page, timestamp, read_time
            FROM reading_history
            WHERE book_id = ?
        ''')
    
```

```

        ORDER BY timestamp DESC
    ''' , (book_id,)).fetchall()
    conn.close()
    return jsonify(history)
except Exception as e:
    logger.error(f"Помилка при отриманні історії читання: {e}")
    return jsonify({"error": str(e)}), 500
# Отримання недавньої активності
@app.route('/api/reading_history/recent', methods=['GET'])
def get_recent_activity():
    """Отримання інформації про нещодавню активність"""
    try:
        days = request.args.get('days', default=7, type=int)
        limit = request.args.get('limit', default=5, type=int)
        date_limit = (datetime.datetime.now() - datetime.timedelta(days=days)).isoformat()
        conn = get_db_connection()
        conn.row_factory = dict_factory
        recent_activity = conn.execute('''
            SELECT rh.book_id, b.title, b.author,
                   SUM(rh.read_time) as total_time,
                   MAX(rh.timestamp) as last_read,
                   b.current_page, b.total_pages
            FROM reading_history rh
            JOIN books b ON rh.book_id = b.id
            WHERE rh.timestamp > ?
            GROUP BY rh.book_id
            ORDER BY last_read DESC
            LIMIT ?
        ''', (date_limit, limit)).fetchall()
        conn.close()
        # Перетворюємо час у години та додаємо URL обкладинок
        for activity in recent_activity:
            activity['total_hours'] = round(activity['total_time'] / 3600, 1)
            activity['cover_url'] = f"/api/books/{activity['book_id']}/cover"
        return jsonify(recent_activity)
    except Exception as e:
        logger.error(f"Помилка при отриманні недавньої активності: {e}")
        return jsonify({"error": str(e)}), 500
# Допоміжні функції для роботи з базою даних
def get_db_connection():
    """Створює підключення до бази даних з фабрикою рядків Row"""
    conn = sqlite3.connect(DATABASE)
    conn.row_factory = sqlite3.Row
    return conn
def dict_factory(cursor, row):
    """Конвертує результати запиту в словник"""
    d = {}
    for idx, col in enumerate(cursor.description):
        d[col[0]] = row[idx]
    return d
# Діагностика та перевірка системи
@app.route('/api/diagnostics', methods=['GET'])
def diagnostics():
    """Повертає діагностичну інформацію про систему"""
    try:
        result = {
            "server": {
                "version": "1.0",
                "directories": {
                    "base_dir": os.path.exists(BASE_DIR),
                    "static_dir": os.path.exists(STATIC_DIR),
                    "books_dir": os.path.exists(BOOKS_DIR),
                    "covers_dir": os.path.exists(COVERS_DIR)
                },
            },
            "default_cover": os.path.exists(DEFAULT_COVER_PATH)
        }
    
```

```

    },
    "database": {
        "file": os.path.exists(DATABASE)
    }
}
# Перевіряємо доступ до Gemini API
if API_KEY_GEMINI and API_KEY_GEMINI != "YOUR_GEMINI_API_KEY":
    try:
        model = genai.GenerativeModel("gemini-pro")
        response = model.generate_content("Hello, are you working?")
        result["ai"] = {
            "status": "working",
            "message": response.text[:100]
        }
    except Exception as e:
        result["ai"] = {
            "status": "error",
            "message": str(e)
        }
else:
    result["ai"] = {
        "status": "not_configured",
        "message": "API ключ не налаштований"
    }
# Отримуємо статистику бази даних
conn = get_db_connection()
# Кількість записів у таблицях
result["database"]["tables"] = {
    "books": conn.execute("SELECT COUNT(*) as count FROM books").fetchone()[0],
    "bookmarks": conn.execute("SELECT COUNT(*) as count FROM
bookmarks").fetchone()[0],
    "quotes": conn.execute("SELECT COUNT(*) as count FROM quotes").fetchone()[0],
    "reading_history": conn.execute("SELECT COUNT(*) as count FROM
reading_history").fetchone()[0]
}
conn.close()
return jsonify(result)
except Exception as e:
    logger.error(f"Помилка діагностики: {e}")
    return jsonify({"error": str(e)}), 500
# Запуск сервера
if __name__ == '__main__':
    # Налаштування перед запуском
    setup_app()
    # Запуск сервера
    app.run(debug=True, host='0.0.0.0', port=5000)

```