МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: 123 «Комп'ютерна інженерія»

Освітньо-професійна програма:

«Обслуговування комп'ютерних систем і мереж»

Група: 4КС-58

Дипломний проект

здобувача освіти денної форми навчання
КС.58.07.000.ДП

ЄПУРА
ЮРІЯ ОЛЕКСАНДРОВИЧА

м. Одеса
2025 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: **123 «Комп'ютерна інженерія»**

Освітньо-професійна програма: **«Обслуговування комп'ютерних систем і мереж»**

Група: **4КС-58**

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломного проекту на тему:

Розробка інтерактивного веб-порталу віртуального клубу читачів

Проектний матеріал складається з пояснювальної записки на 85 сторінках та графічного (презентаційного) матеріалу на 14 аркушах (слайдах)

Дипломник _____ (Єпур Ю.О.)

Керівник _____ (Закроєв Ю.М.)

Консультанти:

з економічного розділу _____ (Канський М.Ю.)

з розділу охорони праці та техніки безпеки _____ (Чорновол Н.І.)

з нормоконтролю _____ (Петрашова В.І.)

старший консультант _____ (Кривченко Ю.В.)

До захисту допущений

Голова циклової комісії _____ (Кривченко Ю.В.)

Завідувач відділення _____ (Краснокутська К.Г.)

Захист «20» червня 2025 р. Протокол ЕК № 1

Оцінка ЕК 5 (відмінно) / 95 б.

Секретар ЕК _____

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Відділення *комп'ютерних систем* Комісія *КТ та ПІ*

Спеціальність *123 «Комп'ютерна інженерія»*

Освітньо-професійна програма *«Обслуговування комп'ютерних систем і мереж»*

ЗАТВЕРДЖУЮ:

Заст. дир. з НВР *Беркань І.В.*

"19" 05 2025 р.

ЗАВДАННЯ

на дипломний проект

Здобувачеві освіти *Єпуру Юрію Олександровичу*

(прізвище, ім'я, по батькові)

1. Тема проекту *Розробка інтерактивного веб-порталу віртуального клубу читачів*

затверджена наказом по коледжу від "*14*" *листопада* 2024р. № *246*

2. Термін здачі закінченого проекту *16.06.25*

3. Вихідні данні до проекту

1. Використовувати СУБД PostgreSQL;

2. Застосовувати фреймворк Django;

3. Застосовувати дружній UX веб-системи;

4. Передбачити сучасний графічний дизайн користувача (GUI) веб-системи;

5. Передбачити використання сторонніх API.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які необхідно розробити)

1. Аналіз предметної області; 2. Технології та засоби розробки (проектування);

3. Проектування веб-дизайну; 4. Проектування архітектури веб-застосунку;

5. Розробка веб-застосунку; 6. Тестування створеного веб-застосунку;

7. Економічний розрахунок; 8. Аспекти охорони праці та техніки безпеки.

5. Перелік графічного (презентаційного) матеріалу (з точним зазначенням обов'язкових креслень, кількості слайдів)

Стек технологій веб-застосунку; Етапи створення веб-застосунку;

Структура веб-застосунку; Архітектура веб-застосунку; Створення моделей;

Тестування веб-застосунку; Схема бази даних; Створення маршрутів; Файлова структура застосунку; Взаємодія з Google Book API; Огляд сторінок веб-системи.

6. Консультанти по проекту, із зазначенням розділів проекту, що їх стосується

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Основний розділ	*Закроєв Ю.М.*		
Економічний розділ	*Канський М.Ю.*		
Розділ охорони праці	*Чорновол Н.І.*		
Нормоконтроль	*Петрашова В.І.*		
Старший консультант	*Кривченко Ю.В.*		

7. Дата видачі завдання 12.05.2025

Керівник *Закроєв Ю.М.* (підпис)

Завдання прийняв до виконання *Єпур Ю.О.* (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/р	Назва етапів дипломного проекту	Термін виконання етапів дипломного проекту (роботи)	Відмітка про виконання
1	*Формування вступу*	22.05.25	виконав
2	*Огляд існуючих рішень*	23.05.25	виконав
3	*Аналіз предметної області*	26.05.25	виконав
4	*Підбір технічної літератури*	27.05.25	виконав
5	*Аналіз технології розробки (проєктування)*	29.05.25	виконав
6	*Проєктування дизайну веб-застосунку*	30.05.25	виконав
7	*Проєктування архітектури веб-застосунку*	31.05.25	виконав
8	*Реалізація веб-застосунку*	01.06.25	виконав
9	*Тестування створенного веб-застосунку*	03.06.25	виконав
10	*Оформлення пояснювальної записки*	04.06.25	виконав
11	*Оформлення графічної (презентаційної) частини*	06.06.25	виконав
12	*Економічний розрахунок*	02.06.25	виконав
13	*Опис охорони праці та техніки безпеки*	08.06.25	виконав
14	*Аналіз результатів розробки (проєктування)*	09.06.25	виконав
15	*Підготовка доповіді для захисту*	12.06.25	виконав

Дипломник (підпис)

Керівник (підпис)

Формат	Зона	Поз.	Позначення	Назва	Кіл.	Примітка
				Документація		
			КС 58. 07 000. 00 ДП	Дипломний проект		
А4			КС. 58. 07 000. 00 ДП ПЗ	Пояснювальна записка	1	

Зм.	Арк.	№ докум.	Підпис	Дата	КС 58. 07 000. 00 ДП		
Розробив		Єпур Ю.О.		16.06	Розробка інтерактивного веб-порталу віртуального клубу читачів	Літ.	Аркуш 4 / Аркушів 85
Перевірив		Закроєв Ю.М.		16.06		Н Д П	
Н. контр.		Петрашова В.І.		16.06.15		ВСП "ОТФК ОНТУ" гр.4КС-58	
Затверд.		Кривченко Ю.В.		16.06			

ЗМІСТ

ВСТУП

Технології сьогодення проникають у кожну сферу життя інформаційні технології все активніше інтегруються в повсякденне життя людей, змінюючи способи комунікації, споживання контенту та організації дозвілля. Зокрема, спостерігається зростання інтересу до онлайн-ресурсів, які дозволяють користувачам об’єднуватися за інтересами, обмінюватися думками та брати участь у спільнотах незалежно від фізичного розташування. В умовах зростаючої популярності читання як хобі та освітньої діяльності, актуальним постає створення віртуального простору для любителів книг, де кожен охочий зможе поділитися своїми враженнями, знайти нову літературу та поспілкуватися з однодумцями.

Цей проєкт виник як спроба дати відповідь на запит сучасних читачів – створити місце, де книга стає приводом для діалогу, а не просто об’єктом споживання. розробка онлайн простір читацького клубу, що забезпечить користувачам можливість реєстрації, вибору книг, участі в обговореннях, ведення особистих читацьких нотаток, а також отримання персоналізованих рекомендацій на основі інтересів. Така система покликана не лише модернізувати спосіб взаємодії між читачами, але й сприяти поширенню культури читання серед широкого кола користувачів.

У процесі розробки платформи після знайомства з аналогами стало очевидно, що більшості з них бракує живого спілкування між користувачами, що й стало поштовхом до переосмислення функціоналу в цій галузі, виявлено їхні недоліки та визначено шляхи вдосконалення функціоналу. Основними задачами стали проєктування архітектури системи, створення бази даних для зберігання інформації про книги, користувачів та нотатки, реалізація клієнтської частини з інтуїтивно зрозумілим інтерфейсом, а також впровадження REST API для обміну даними між клієнтом і сервером.

Для реалізації платформи було використано набір перевірених інструментів, який не лише відповідає технічним вимогам, а й дозволяє розробляти рішення гнучко й швидко, зокрема фреймворк Django та Django REST Framework для створення серверної логіки та API. Як система керування базами даних

використовується PostgreSQL, що дозволяє ефективно зберігати та обробляти великі обсяги інформації. Клієнтська частина розроблена з використанням HTML, CSS та JavaScript, що забезпечує зручний і адаптивний інтерфейс. Додатково було інтегровано зовнішній API для отримання даних про книги, що значно розширює функціональні можливості платформи [1][2].

Результатом розробки є повнофункціональний веб-сервіс, який дозволяє користувачам брати активну участь у діяльності читацького клубу в онлайн-форматі. Проєкт не обмежується поточними рамками – його задум спонукає до еволюції, до нових ідей і розширення функціональності відповідно до запитів спільноти й масштабування, що робить його актуальним як для приватного використання, так і для впровадження в освітніх чи культурних установах.

.

1 ОСНОВНИЙ РОЗДІЛ

1.1 Аналіз предметної області

1.1.1 Огляд існуючих аналогів

Сучасна цифрова епоха суттєво змінила способи, якими люди взаємодіють із літературою. Якщо раніше основним джерелом інформації про книги були бібліотеки або друковані каталоги, то в даний момент переважна більшість користувачів звертається до спеціалізованих онлайн-сервісів, які дозволяють не лише знаходити книги, а й обговорювати їх, вести власну віртуальну бібліотеку, отримувати персоналізовані рекомендації та ділитися враженнями з іншими читачами.

В ході пошукі виявилося, що із найпоширеніших рішень у цій сфері є платформа Goodreads. Сервіс може забезпечувати потужну екосистему для створення каталогів книжок, написання рецензій, участі в тематичних групах по інтересах, обговореннях та викликах на читання. Крім того, також існує можливість підключення авторів до платформи, що надає їм змогу спілкуватися з читачами та рекламувати власні твори. Хоча незважаючи на велику функціональність, інтерфейс ресурсу є застарілим, а відкритий API надає невеликий набір функцій. Додатковим недоліком є орієнтованість ресурсу переважно на англомовну аудиторію [3].

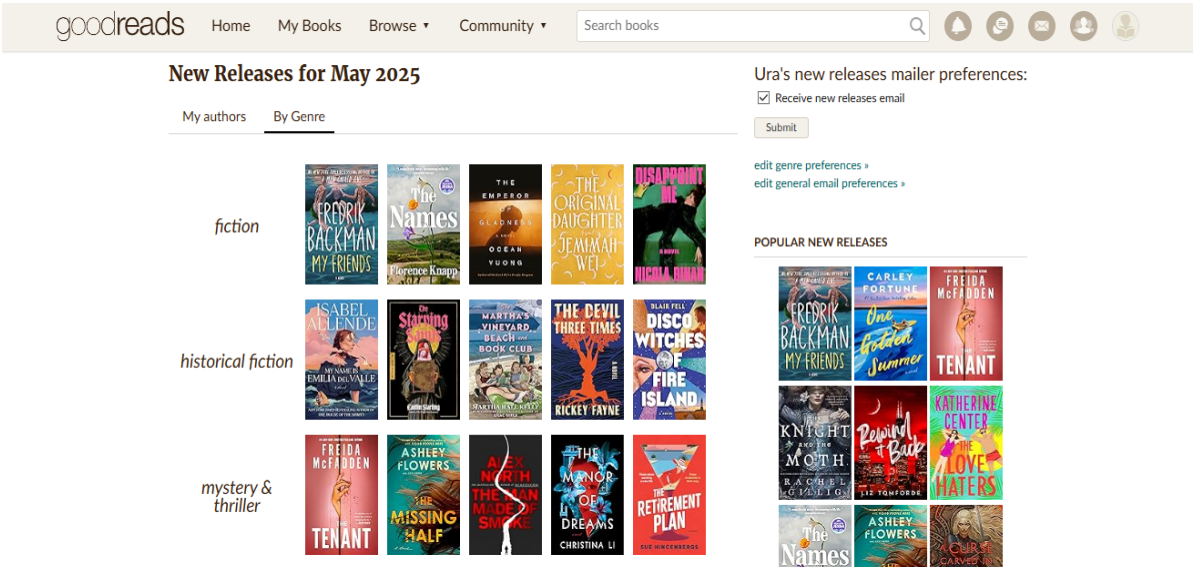


Рисунок 1.1. Інтерфейс застосунку Goodreads

Схожі сервіси, наприклад одним із таких є LibraryThing, він також надає інструменти для управління власною бібліотекою, проте він має більшу орієнтованість на наукову та навчальну літературу. Функціональність таких платформ менш зручна для звичайного користувача, що шукає рекомендації або можливості соціальної взаємодії [4].

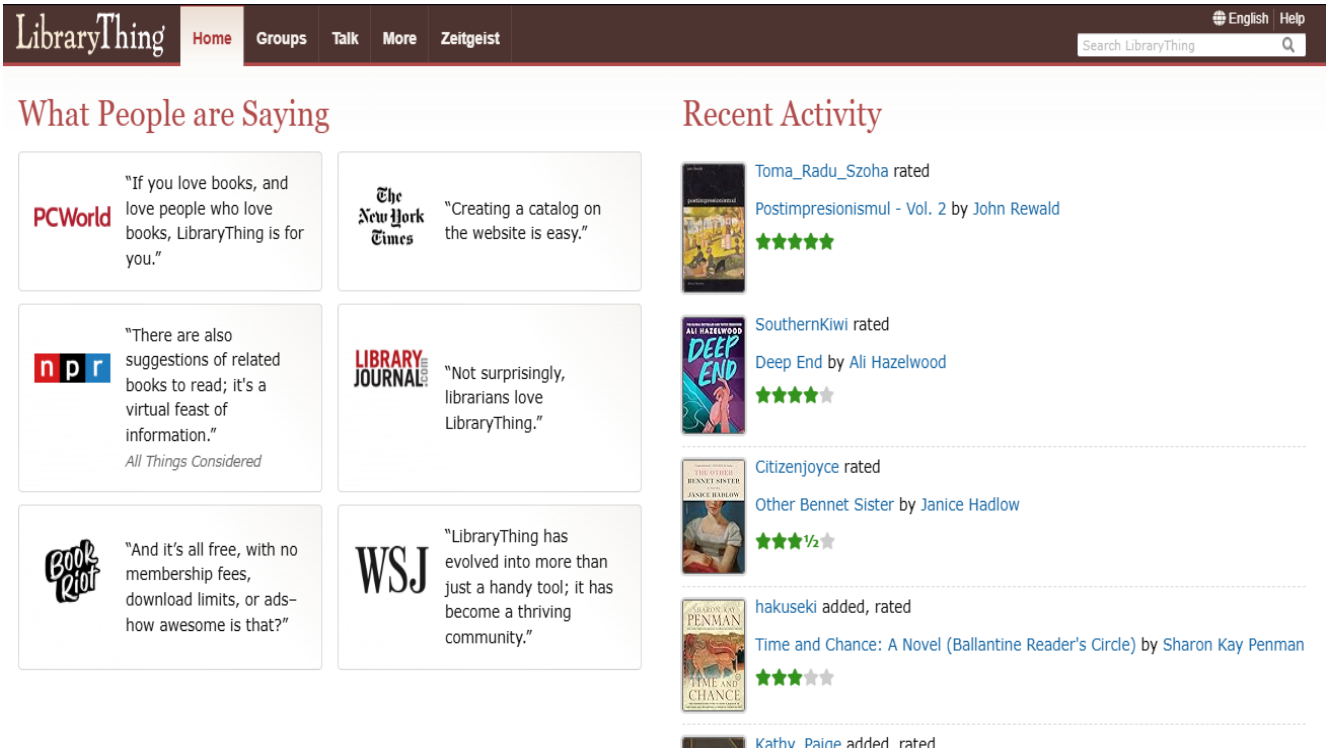


Рисунок 1.2. Інтерфейс застосунку LibraryThing

У локальному просторі поширення набули платформи на зразок “Книгоманія” та BookSter, які надають україномовним користувачам можливість вести особисту бібліотеку, оцінювати книги, залишати рецензії та формувати списки для читання. Проте ці ресурси часто мають проблеми з актуальністю баз даних, а також не надають широких можливостей щодо інтеграції з іншими системами або доступу до відкритих API [5][6].

Слід зазначити, що більшість із вище перечислених платформ мають недостатню або зовсім не мають адаптацію під мобільні пристрої, а їхній інтерфейс часто є перевантаженим або морально застарілим. Водночас, відповідно до сучасних тенденцій у веб-розробці, акцент має робитися на адаптивному дизайні, щоб було зручному для взаємодії як на великих екранах, так і на смартфонах чи планшетах.

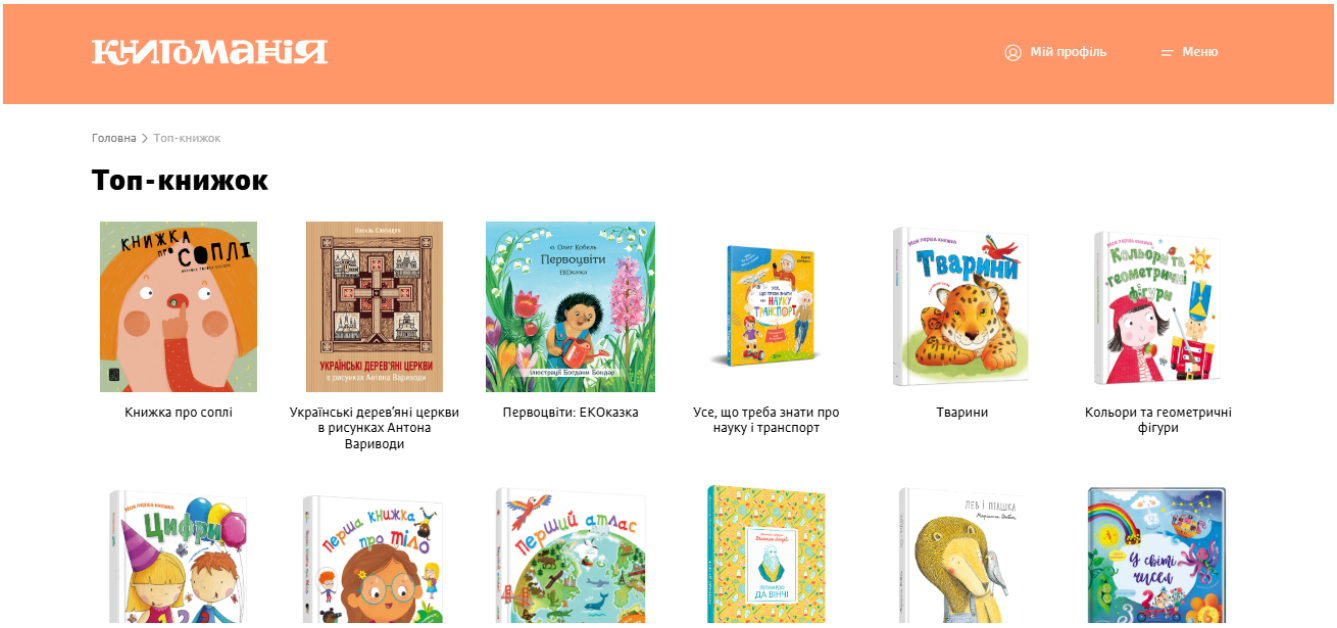


Рисунок 1.3. Інтерфейс застосунку Книгоманія

На підставі вивчених прикладів було сформовано перелік ключових функціональних та технічних вимог до нового веб-порталу:

- підтримка інтерактивного обговорення літератури;
- можливість авторизації через обліковий запис Google;
- пошук книг із використанням Google Books API;
- реалізація системи рекомендацій;
- використання сучасного frontend з підтримкою адаптивності.

Усі із цих перечисленних пунктів лягли як умови в основу розробки інтерактивного веб-порталу віртуального клубу читачів, що дозволяє поєднати переваги класичних книжкових сервісів із сучасними інженерними рішеннями у сфері веб-технологій

Таблиця 1.1 Порівняльна характеристика аналогів

Критерії	Goodreads	LibraryThing	Книгоманія	Bookster
Мова інтефейсу	Англійська	Англійська	Українська	Українська
Основна аудиторія	Міжнародна	США/Європа	Україна	Україна
Каталог книг	Величезний (Amazon)	Великий(Ручне додавання)	Середній	Серендній (акцент на укр. книги)
Соціальні функції	Рецензії, групи	Форуми, теги	Рецензії, підбірки	Чат-клуби

Продовження таблиці 1.1

Критерії	Goodreads	LibraryThing	Книгоманія	Bookster
Рекомендації	Алгоритмічні	на основі тегів	ручні підбірки	персоналізовані
Українські книги	Мало	Рідко	Основний контент	Основний контент
Мобільний додаток	Так(незручний)	Немає	Немає	Так(зручний)
Унікальна фішка	Інтеграція з Kindle	Каталогізація бібліотеки	Фокус на укр. авторів	Ігррові механіки(Читацькі челенджи)

1.1.2 Огляд існуючих технологій

Проєктування та реалізація сучасного веб-додатку потребує багато часу роботи та ретельного вибору технологій, які будуть забезпечувати стабільність, масштабованість, безпеку сервісу та зручність його розробки. З огляду на поставлену мету цього проекту – створення інтерактивного веб-порталу для обговорення літератури з підтримкою авторизації через Google та інтеграції з Google Books API – було обрано стек технологій, який охоплює одні із найсучасніших фреймворків, мови програмування, бібліотеки інтерфейсу, а також засоби контейнеризації та тестування.

Основним інструментом для розробки серверної частини було обрано мову програмування Python, яка відзначається читабельністю синтаксису, широкою екосистемою пакетів та дуже активною спільнотою.

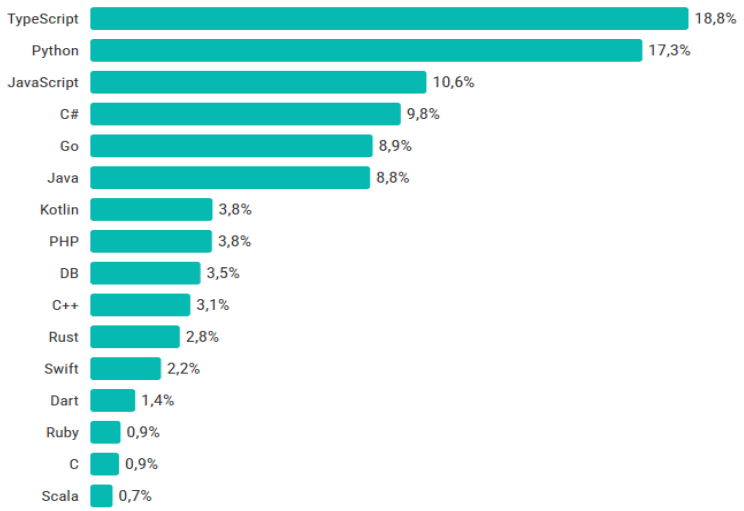


Рисунок 1.4. Рейтинг мов програмування

У динаміці за шість років помітно захопливе зростання TypeScript та Python, але це ще не максимум можливої популярності. Частка «традиційного програмування» на C# та Java зменшується. Можна шукати пояснення у відході від VM та поширенні PAAS-платформ, на кшталт Amazon Lambda або Cloudfare Workers, що орієнтовані на TypeScript/JavaScript екосистему. Або у відносній складності та багатослівності C# і Java та людській потребі шукати нових рішень [7].

Як основний backend-фреймворк використано Django, що забезпечує швидку розробку, вбудовану систему аутентифікації, ORM, підтримку шаблонів, а також інтеграцію з REST-фреймворками. Основний сервіс системи реалізоано саме на базі Django [8].

Для побудови REST-інтерфейсів було використано Django REST Framework (DRF) – це дуже потужне розширення до Django, яке дозволяє швидко формувати API, серіалізатори та обробку запитів.

Для реалізації клієнтської частини застосовано такі стандартні веб-технології як: HTML5, CSS3 та JavaScript. Використання чистого JavaScript дозволило мати повний контроль над логікою взаємодії з API, DOM-структурою та динамічним оновленням сторінок.

Додатково було застосовано адаптивну верстку, що забезпечує коректне відображення контенту як на десктопних пристроях, так і на мобільних.

У майбутньому допускається перехід до використання таких бібліотек, як React або Vue.js для створення компонентного інтерфейсу.

Щоб зберігати інформацію про користувачів, обговорення, книги (додані вручну), було обрано PostgreSQL – потужну об’єктно-реляційну систему управління базами даних яка підтримує складні запити, транзакції та розширення.

Завдяки сумісності з Django ORM, PostgreSQL дозволяє реалізувати гнучке управління даними без необхідності прямого написання SQL-запитів у більшості випадків, що дуже спрочує розробку та підтримання даного проекту.

Щоб отримати інформації про книги (обкладинки, авторів, опис тощо) використовується Google Books API. Взаємодія з цим API відбувається за

допомогою HTTP-запитів, а отримані дані серіалізуються та передаються до клієнтської частини через REST-інтерфейси, де вони збурігаються у базу даних.

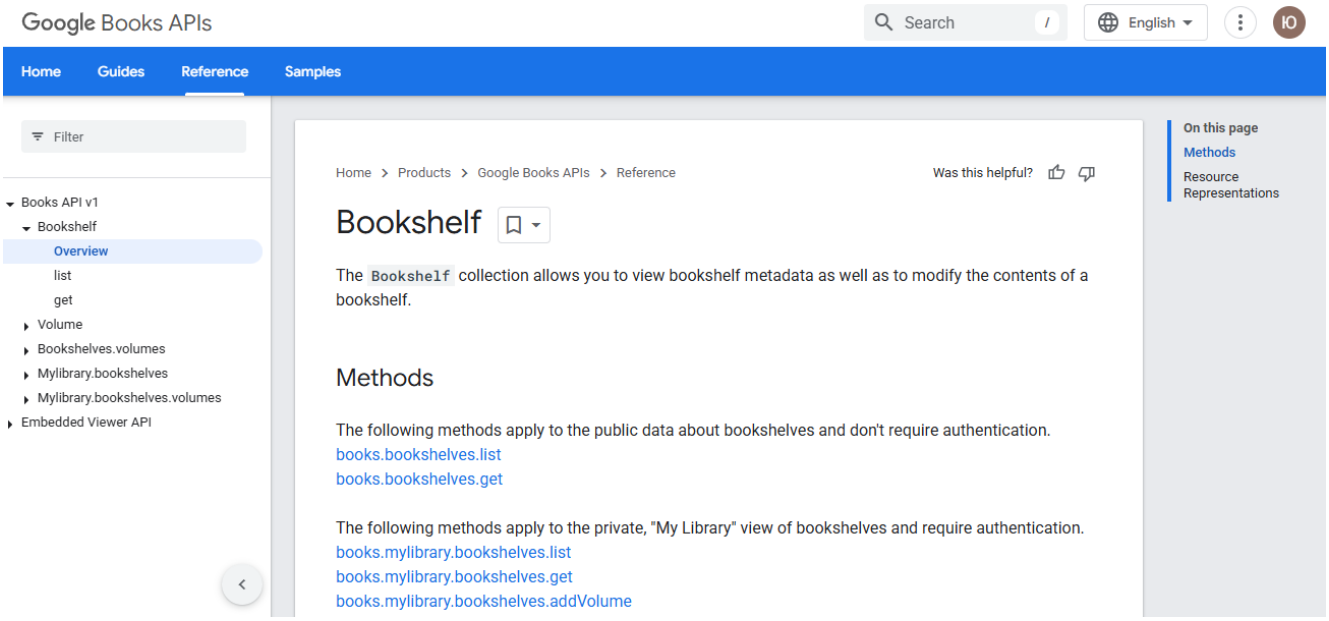


Рисунок 1.5. Інтерфейс Google Books API

І звісно було реалізоано можливість авторизації користувачів за допомогою OAuth 2.0 через облікові записи Google, що спрочує та робить безпечнішою авторизацію у багато разів.

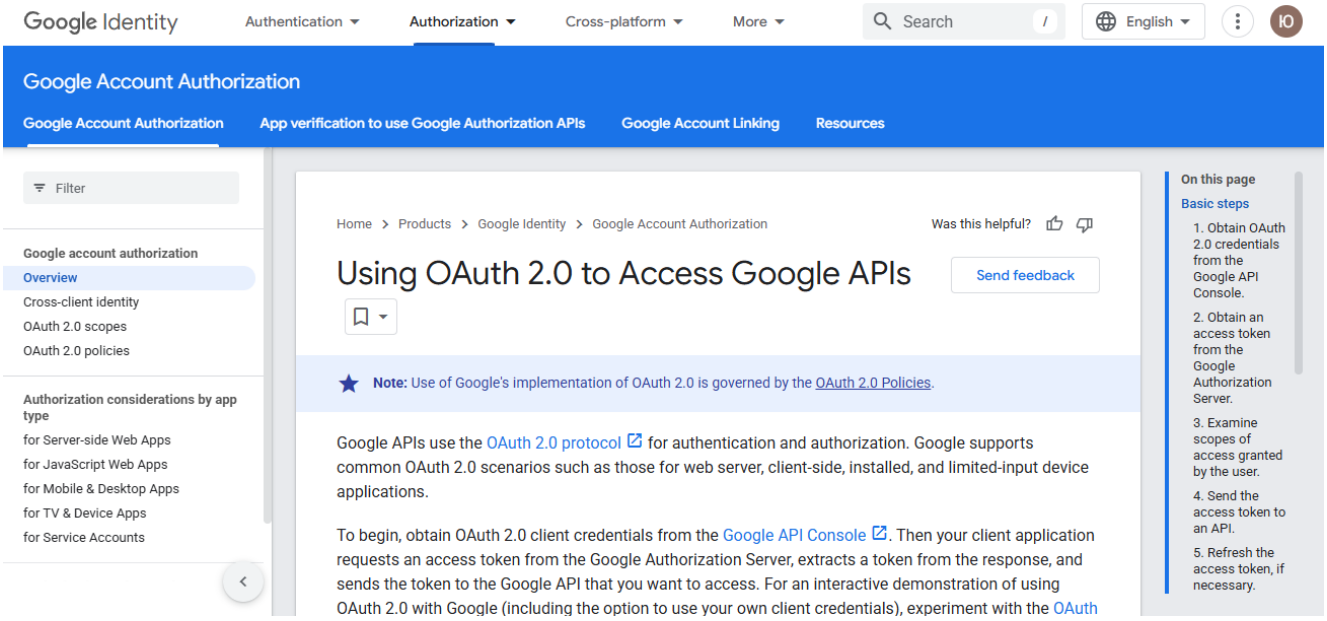


Рисунок 1.6. Інтерфейс Google OAuth 2.0

Щоб зробити тестування функціональності застосовано вбудовані можливості Django (TestCase), що є дуже комфортиним. Також використовувався

такий інструмент як Postman для ручного тестування REST API.

Весь код цього проєкту зберігається у репозиторії Git, що дозволяє ефективно відстежувати зміни, також дозволяє створювати гілки для окремих функцій, і головне дозволяє працювати в команді, що буде дуже доречно якщо проект буде розширюватися. Репозиторій розміщено на платформі GitHub, що забезпечує зручний веб-інтерфейс для перегляду змін, керування задачами та інтеграцію з CI/CD.

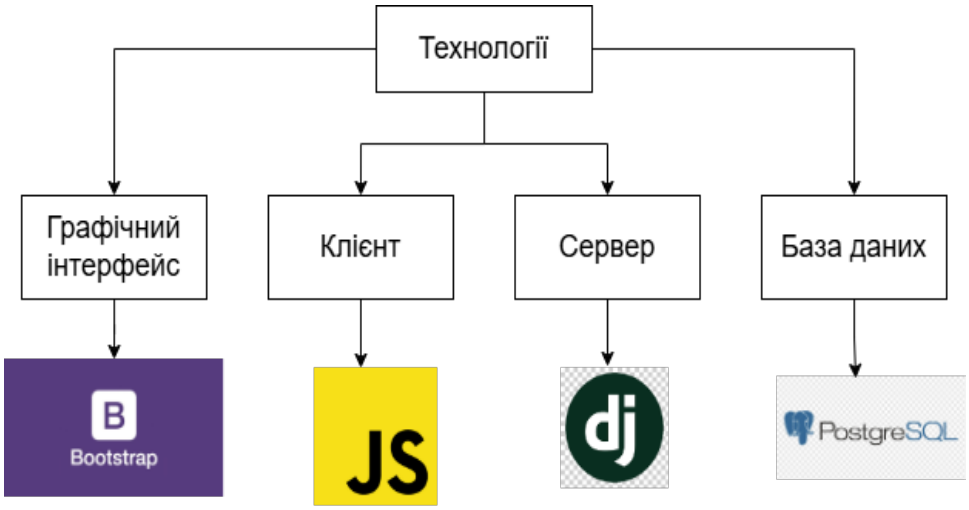


Рисунок 1.7. Схема використаних технологій

1.2 Проєктування веб-системи

1.2.1 Проєктування програмної архітектури веб-системи

У рамках цієї дипломної роботи розроблюється сучасна веб-платформа – віртуальний клуб читачів, що може об'єднувати людей, зацікавлених у книжках і всьому, що з ними пов'язано. Цей портал покликаний не просто забезпечити зручний пошук літератури, а й стати місцем для живого спілкування, обміну думками та формування активної спільноти навколо читання, що є дуже цікавим. На фоні стрімкого цифрового розвитку дозвілля і змін у культурі споживання інформації, така ініціатива виглядає не лише доречною, а й необхідною.

Уся логіка платформи зосереджена в одному цілісному застосунку, що значно полегшує розгортання й технічну підтримку, розроблений на основі фреймворку Django у зв’язці з Django Rest Framework. Такий підхід дозволяє зосередити всю бізнес-логіку в межах єдиного середовища, що повино дуже

спростити розробку проекту, налагодження та підтримку проєкту, а також знижує технічні ризики на ранніх етапах реалізації. У якості бази даних використовується PostgreSQL, яка добре сплітається з фрейморком Django та забезпечує стабільну роботу навіть при зростанні кількості користувачів.

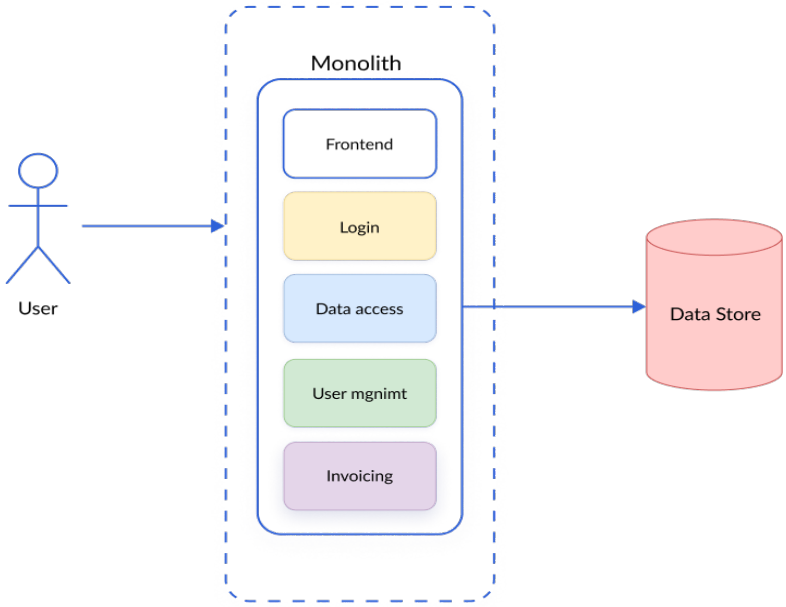


Рисунок 1.8. Схема монолітної архітектури

Платформа передбачає широкий спектр функцій, необхідних для повноцінної взаємодії з користувачем. Одним із перших кроків є реєстрація – як звичайна через електронну пошту, так і спрощена за допомогою інтеграції з Google-акаунтом. Це дозволяє швидко почати користування сервісом, не витрачаючи час на заповнення численних форм. У майбутньому можливе підключення інших варіантів входу, зокрема через популярні соціальні мережі.

Важливу роль відіграє модуль, відповідальний за роботу з книжковим контентом. Дані про книги отримуються з Google Books API, після чого обробляються на стороні сервера: зберігаються ключові атрибути, такі як назва, автор, рік видання, короткий опис і зображення обкладинки. Увесь цей матеріал подається у зручному вигляді на сторінках порталу, де кожна книга має окрему інформаційну сторінку.

Одним із центральних елементів системи є можливість залишати коментарі до книжок та брати участь у дискусіях. Це своєрідний форум, де кожен може висловити свою думку, поставити запитання чи запропонувати власну

інтерпретацію прочитаного. Повідомлення мають прив’язку до відповідних книжок, зберігаються в базі даних разом із іменем автора та часом публікації, що робить обговорення більш структурованими.

Щоб зробити платформу максимально корисною, реалізується механізм персоналізованих рекомендацій. На основі активності користувача – його пошукових запитів, переглянутих книжок, участі в обговореннях – система поступово формує індивідуальний список творів, які можуть бути цікавими саме цій людині. Це дозволяє зробити процес читання більш захопливим і наближеним до персональних інтересів користувача.

Суттєву увагу приділено інтерфейсу. Він створений із урахуванням адаптивного дизайну, тому однаково добре працює як на мобільних пристроях, так і на великих екранах. Усі тексти і навігація подані українською мовою, що сприяє збереженню мовної ідентичності та забезпечує зручність для основної аудиторії.

Розгортання проєкту виконується в середовищі Docker, що забезпечує стабільну та передбачувану роботу системи у різних умовах – від локального тестування до продуктивного сервера. Контейнери дозволяють швидко запускати нові версії застосунку, не залежачи від конфігурацій конкретної машини, що особливо важливо на етапі розробки та демонстрації.

Як бачимо, дипломний проєкт охоплює не лише реалізацію конкретних функцій, а й більш глибокі аспекти – архітектурні рішення, зручність інтерфейсу, безпеку, масштабованість та перспективу подальшого розвитку. Це не просто сайт, а повноцінна платформа для комунікації, самоосвіти та взаємодії, яка здатна стати актуальним інструментом для всіх, хто цінує читання і шукає однодумців.

1.2.2 Проєктування структури бази даних веб-системи

Організація сховища даних є одним із ключових аспектів при розробці інтерактивних веб-платформ. У контексті створення віртуального клубу читачів, де передбачено інтенсивну взаємодію між користувачами, збереження особистих даних, обговорення, а також динамічний обмін інформацією про книги, постає необхідність у надійній, масштабованій та логічно структурованій базі даних. У рамках даного проєкту базу даних реалізоано на основі системи управління

PostgreSQL, яка зарекомендувала себе як стабільне та високопродуктивне рішення з відкритим вихідним кодом, що підтримує розширений SQL-стандарт та забезпечує ACID-сумісність.

Проєктування структури бази даних розпочалося з визначення основних сутностей, які репрезентують ключові об'єкти системи. Однією з базових сутностей є користувач. Вона включає унікальний ідентифікатор, електронну пошту, хешований пароль, ім'я (при наявності), дату реєстрації, а також індикатори, пов'язані з методом авторизації – скажімо, ознаку входу через Google OAuth. Окрема увага приділяється захисту персональних даних, для чого реалізоано відповідні механізми шифрування та контроль доступу до критичних полів.

Наступною сутністю виступає книга, яка, на відміну від класичних систем, не зберігається повністю у локальній базі даних. Основні дані про книги отримуються через зовнішнє API сервісу Google Books, проте для забезпечення зручної роботи з ними створюється допоміжна таблиця-кеш. У ній зберігаються найбільш затребувані атрибути книги: заголовок, автор, рік видання, унікальний ідентифікатор Google Books, посилання на зображення обкладинки та короткий опис. Такий підхід дозволяє скоротити кількість звернень до зовнішнього API та пришвидшити рендеринг сторінок.

Важливою складовою структури бази даних є таблиці, що відповідають за обговорення. Кожен коментар зв'язується з певною книгою та конкретним користувачем, має часову мітку створення, текст повідомлення та індикатор, що може бути використаний для модерації (скажімо, для позначення неприйнятного контенту). Ієрархія коментарів організована у вигляді простого списку без вкладеності, що дозволяє зберегти читаємість та уникає надмірної складності під час реалізації інтерфейсу.

З урахуванням подальшого впровадження рекомендаційної системи структура бази даних передбачає можливість додавання нових таблиць, що фіксуватимуть історію дій користувача. Йдеться, зокрема, про запити до пошуку, переглянуті книги, участь в обговореннях, поставлені вподобання тощо. Ці дані

можуть слугувати вхідними параметрами для алгоритмів машинного навчання або евристичних моделей, які формуватимуть персоналізовані добірки.

У проєкті було зроблено ставку на Django ORM для роботи з даними, і цей вибір цілком виправдав себе. Замість традиційного написання SQL-запитів ми працюємо з даними через звичні Python-класи, що значно прискорює процес розробки. Автоматичне створення таблиць, міграції "з коробки" та вбудовані механізми валідації – усе це дозволяє зосередитись на бізнес-логіці, а не на технічних деталях роботи з базою даних [5].

Безпека даних – один з наших пріоритетів, і тут Django ORM пропонує вигідні рішення. Вбудований захист від SQL-ін'єкцій, типізовані операції та чіткий контроль доступу знімають з розробників значну частину клопотів. Ми особливо цінуємо те, що ці механізми працюють за замовчуванням, не вимагаючи додаткових зусиль з нашого боку [6].

Архітектура нашої бази даних спеціально спроектована плоскою – це свідомий вибір, обумовлений конкретними вимогами проєкту. Такий підхід мінімізує навантаження на сервер, прискорює виконання запитів і спрощує майбутнє масштабування. Крім того, він робить систему більш зрозумілою для нових членів команди, що важливо для довгострокової підтримки проєкту [7].

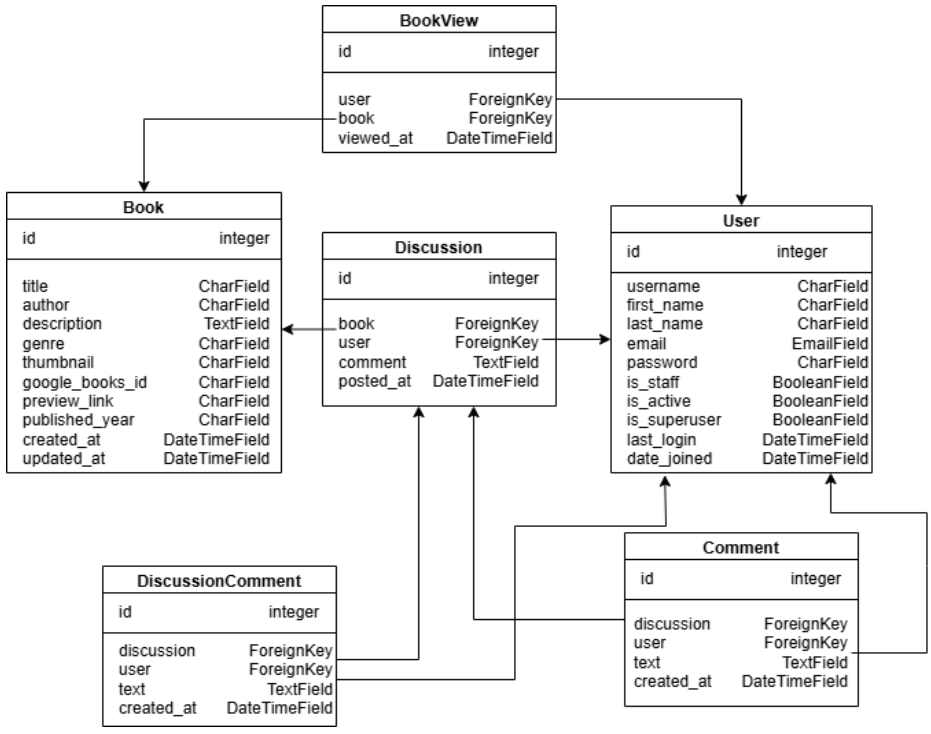


Рисунок 1.9. Схема бази даних

Сьогоднішні веб-додатки, включаючи наш, мають бути швидкими, гнучкими та зручними для розширення. Наша реалізація на Django ORM повністю відповідає цим вимогам. Ми змогли досягти оптимального балансу між продуктивністю, безпекою та зручністю розробки, що особливо важливо для монолітних систем. Досвід показує, що надмірно складна структура даних у таких випадках може серйозно ускладнити підтримку та погіршити швидкодію.

Ці технічні рішення дозволяють нам гарантувати стабільну роботу додатку навіть при зростанні навантаження та ускладненні функціоналу. У сучасних умовах, коли користувачі очікують миттєвого відгуку та безперебійної роботи, такі архітектурні рішення стають не просто бажаними, а обов'язковими для успішного проєкту.

1.2.3 Проєктування взаємодії з API Google Books

Однією з найцікавіших і водночас найпрактичніших функцій платформи для читачів є можливість миттєво отримувати доступ до великого обсягу книжкової інформації з усього світу. Це реалізоано не шляхом збереження локального каталогу або ручного наповнення бази даних, а завдяки інтеграції з зовнішнім джерелом – Google Books API. Фактично, платформа стає посередником між користувачем і глобальною цифровою бібліотекою, забезпечуючи швидкий і зручний доступ до книжкових описів, обкладинок, авторів, жанрів, років публікації та іншої супутньої інформації [10].

Рішення використовувати саме Google Books API було прийняте не лише з огляду на технічні переваги цього інструменту, а й завдяки його практичній цінності для цільової аудиторії. Уявити собі сучасного читача без цифрових інструментів пошуку інформації вже складно. Люди звикли знаходити потрібну книжку не за тиждень, як колись у бібліотеці, а за лічені секунди – зі смартфона чи ноутбука. Саме таку швидкість і масштаб ми намагалися забезпечити. API від Google має мільйони позицій у своєму каталозі, доступ до яких відкривається простим HTTP-запитом. Проте за простотою на перший погляд ховається низка викликів, з якими система повинна вміти працювати. Google Books API надає зручний інтерфейс для роботи з книжками.

У рамках розробки онлайн простір було прийнято рішення не розділяти систему на окремі сервіси, а натомість реалізувати всю логіку в межах монолітної архітектури, побудованої на Django, хоча в майбутньому для підтримання цього прокту буде краще перейти до мікросервісної архітектури, це покрачить розширюваність цього проекту. Це не лише прискорило розробку та спростило процес деплойменту, а й зробило систему більш передбачуваною для підтримки. Оскільки Google Books API – є зовнішнім джерелом, яке працює за своїми правилами, особливу увагу було приділено стабільності інтеграції. Наприклад, під час кожного звернення система не просто надсилає запит, а ретельно формує параметри відповідно до запиту користувача: це може бути ключове слово з назви книги, ім’я автора або конкретний жанр. Усі параметри проходять попередню перевірку, щоб уникнути помилок на зразок “Bad Request”, які можуть з’явитися при некоректному форматі.

Одним із викликів стало те, що структура відповідей від Google Books API не завжди є передбачуваною. Наприклад, у деяких книжок може бути вказано кількох авторів або жодного, стало тяжким виклом; опис іноді є дуже розгорнутим, чи його і може зовісм не бути; обкладинки представлені різними типами зображень або зовсім не надаються. Щоб уникнути порушення логіки відображення на frontendi, система адаптує кожну відповідь під єдиний шаблон. Якщо якесь поле відсутнє – генерується дефолтне значення або попередження, що інформація неповна. Тобто користувач завжди отримує зрозумілу й коректно відформатовану відповідь, навіть якщо початкові дані були фрагментарними, що має стати для користувача більш привабливим.

Ще одне важливе питання – це кількість запитів. Google, як і більшість зовнішніх сервісів, обмежує частоту доступу до свого API. Щоб уникнути перевищення ліміту, система зберігає найчастіші запити у внутрішній базі даних. Кешування відбувається автоматично: якщо запит на одну й ту саму книжку повторюється кілька разів, відповідь береться не з інтернету, а з локального джерела. Це дозволяє не лише уникати зайвого навантаження на Google Books API, а й значно прискорює відображення результатів пошуку для користувача, роблячи

Google Books API оптимальним рішенням для вирішення поставленої задачі.

Окрема увага приділяється зберіганню ключа доступу до API. Щоб уникнути випадкового оприлюднення, ключ не вшито у відкритий код, а зберігається у змінних середовища, які не потрапляють до репозиторію. Це стандартна, але критично важлива практика безпеки, що дозволяє уникнути блокування акаунта Google у разі несанкціонованого доступу.

На завершення хочеться відзначити, що така інтеграція відкриває нові горизонти для подальшого розвитку платформи. Дані, отримані з Google Books, можна не лише відображати, але й аналізувати. Вже зараз система готується до впровадження персоналізованих рекомендацій, формування тематичних добірок і динамічного фільтрування за жанрами. Усе це стало можливим завдяки продуманій роботі з API, що не просто додає функціонал, а створює передумови для розумнішої й більш чутливої до користувача системи.

1.2.4 Проєктування архітектури веб-порталу

Архітектура програмного забезпечення є базовим фактором, що визначає організацію коду, логіку взаємодії компонентів і можливості подальшого розвитку системи. У ході розробки веб-порталу віртуального клубу читачів було прийнято рішення застосувати монолітну архітектуру яка передбачає об'єднання всіх функціональних компонентів у межах одного додатку. Такий підхіід значно спрощує розгортання проекту, забезпечує повну цілісність системи та полегшує початкову підтримку й відлагодження. Хоча в майбутньому краще перейти на мікросервісну архітекутру, яка дозволяє будувати архітектуру куди краще якщо в меті стоїть розширюваність проекту. В даному проекті не було використано мікросервісної архітектури тому що це зайняло багато часу на навчання і розробку.

Система реалізована на основі фреймворку Django, який забезпечує зручну інтеграцію між усіма рівнями додатку: від бази даних до представлення користувачу. Усі основні функції – автентифікація та авторизація користувачів (зокрема через Google OAuth), керування обліковими записами, пошук книг, доступ до зовнішнього API Google Books, обговорення, збереження книжок, додавання коментарів – реалізоано в межах одного цілісного застосунку. Відмова від

розділення на окремі сервіси дозволила мінімізувати складність налаштування взаємодії між компонентами та уникнути проблем, пов'язаних із мережею чи синхронізацією [9][10].

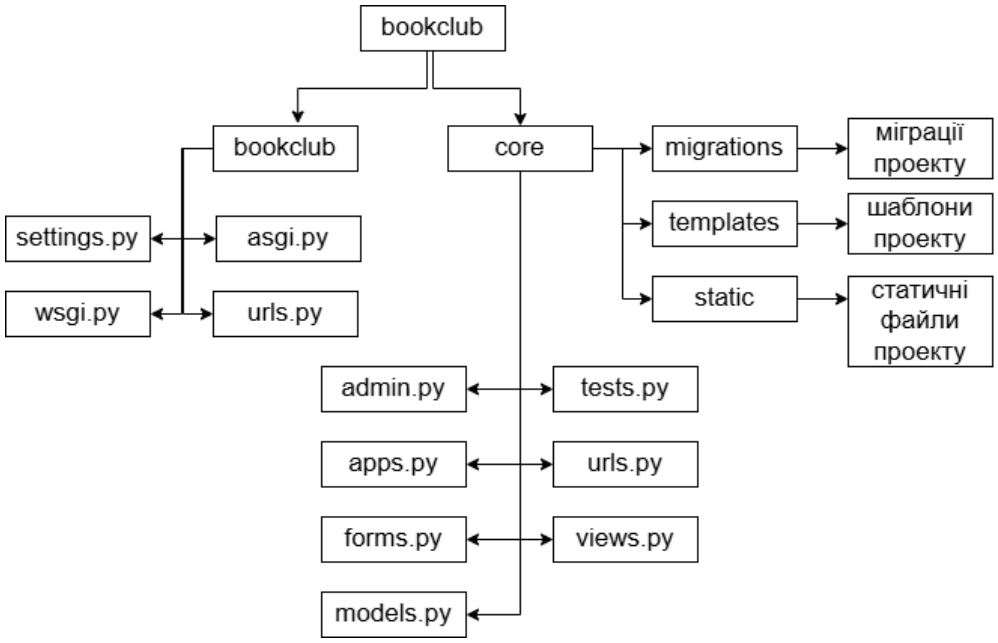


Рисунок 1.10. Схема розміщення файлів у проекті

Всі зовнішні інтеграції, зокрема взаємодія з Google Books API, реалізуються безпосередньо у вигляді окремих модулів у структурі Django-проєкту. Це дозволяє системі витягувати інформацію про книги, виконувати пошукові запити та кешувати отримані дані у локальній базі. Виходить, функціональність, яка у мікросервісному підході покладалася б на окремий сервіс, інтегрована безпосередньо в основний код додатку.

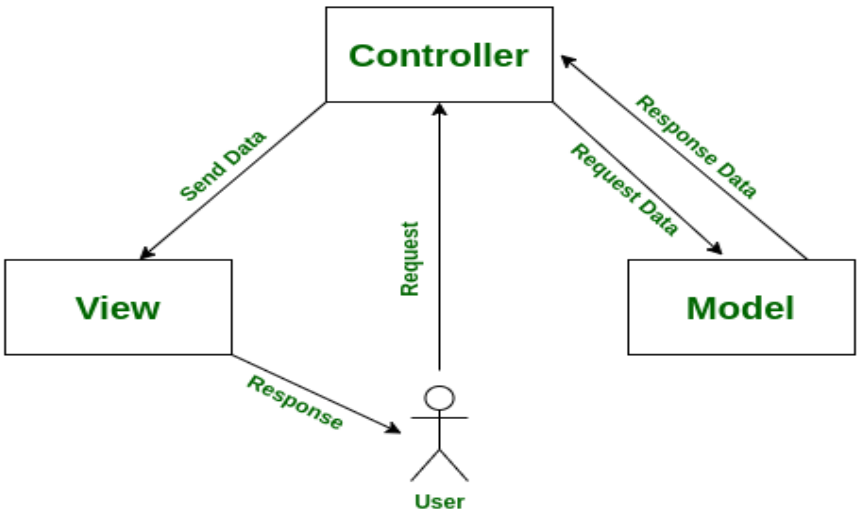


Рисунок 1.11. Архітектура MVC

Архітектура платформи слідує класичній парадигмі MVC (Model-View-Controller), яка природно реалізована у Django. Модельний рівень включає визначення всіх сутностей у вигляді Django-моделей, що зберігаються у базі даних PostgreSQL. Рівень подання реалізоано як через HTML-шаблони для багатосторінкової взаємодії, так і через REST API для обміну даними у форматі JSON. Контролери, представлені у вигляді view-функцій або класів, відповідають за обробку логіки запитів, взаємодію з моделями, авторизацію та маршрутизацію [12].

Для забезпечення стабільності розгортання використовується контейнеризація за допомогою Docker. Увесь застосунок, включно з сервером бази даних PostgreSQL, розміщений у межах окремих контейнерів, що дозволяє точно відтворити робоче середовище як у процесі розробки, так і під час розгортання в продакшн-оточенні. Незважаючи на монолітну структуру, система зберігає можливість масштабування, наприклад, шляхом запуску декількох екземплярів веб-сервера або оптимізації кешування зовнішніх запитів.

Архітектура проекту

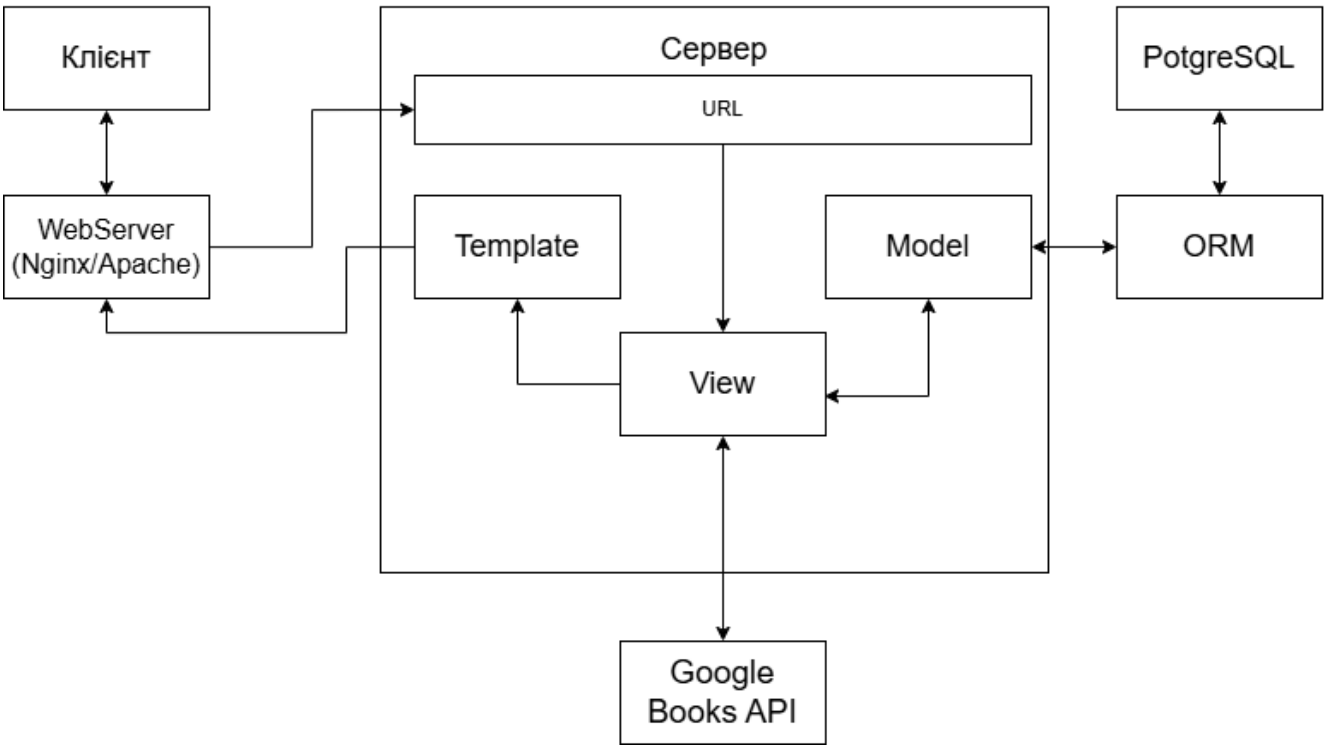


Рисунок 1.12. Архітектура проекту

Уся структура даних централізована та зберігається в єдиній реляційній базі PostgreSQL. Вона об’єднує всі об’єкти предметної області – від інформації про користувачів і обговорення до збережених книжок і кешованих відповідей з Google Books. Такий підхід спрощує управління транзакційністю та забезпечує цілісність даних у межах єдиного сховища [13].

Обрана архітектура демонструє ефективний баланс між простотою розробки та функціональною повнотою. Вона дозволяє швидко впроваджувати нові можливості, підтримувати узгоджену логіку взаємодії між компонентами й легко адаптувати систему до змін вимог. У майбутньому, за потреби розширення, існує можливість поступового переходу до мікросервісної архітектури шляхом відокремлення окремих підсистем без суттєвих змін основного ядра. Як бачимо, монолітна модель забезпечує не лише поточну ефективність, а й стратегічну гнучкість.

1.2.5 Проєктування інтерфейсу користувача

Проєктування інтерфейсу користувача є важливою складовою розробки веб-порталу, оскільки саме через візуальне представлення забезпечується безпосередній контакт користувача з функціональністю системи. Інтерфейс не лише визначає зручність навігації, але й формує загальне враження від використання платформи, що впливає на залученість відвідувачів та тривалість їх взаємодії з ресурсом.

Під час розробки інтерфейсної частини проєкту особлива увага приділялася структурній простоті, інтуїтивній зрозумілості та відповідності сучасним веб-стандартам. Була поставлена мета досягти такого результату, за якого навіть новий користувач зміг би без додаткових інструкцій легко зареєструватися, знайти цікаві книги, долучитися до обговорення або переглянути відгуки інших учасників клубу.

Загальна структура інтерфейсу орієнтована на логічну організацію інформації. Центральне місце займає головна сторінка, яка виконує роль навігаційного вузла: на ній розміщено пошук, найпопулярніші книги, новинки та короткі анонси обговорень. Для швидкого доступу передбачено верхнє меню, яке залишається доступним незалежно від сторінки, на якій перебуває користувач.

Внутрішні розділи включають окремі екрани для перегляду книги, обговорення, профілю користувача, а також сторінки авторизації та реєстрації.

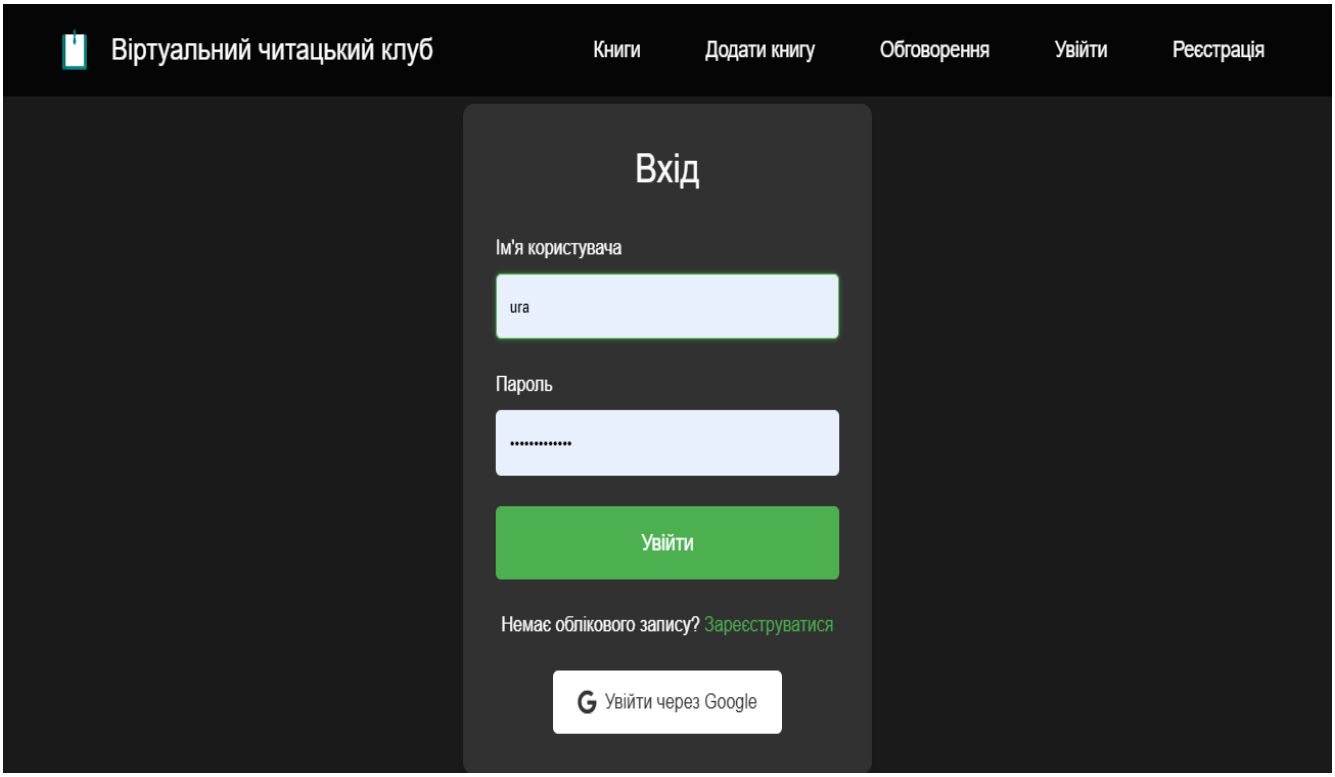


Рисунок 1.13. Сторінка логіну користувача у проекті

Сторінки, пов’язані з реєстрацією, були реалізовані з урахуванням сучасних вимог до захищеної автентифікації. Передбачено два способи входу: через електронну пошту з підтвердженням пароля та через Google-акаунт за допомогою OAuth 2.0. У обох випадках реалізоано базову валідацію введених даних, з чіткими підказками та повідомленнями про помилки, що сприяє мінімізації помилок користувача під час взаємодії з формами [9].

Обговорення є ключовим елементом інтерфейсу, адже саме вони створюють атмосферу живої читацької спільноти. Сторінка кожного обговорення містить тему, пов’язані книги, пости користувачів, дату публікації, лічильник реакцій та форму для залишення коментаря.

Для зручного ознайомлення з книгами реалізоано подання у вигляді карток. Кожна картка містить обкладинку, назву, автора, рік публікації та короткий опис. При натисканні відкривається сторінка з розширеною інформацією, де користувач може ознайомитися з жанром, рейтингом, а також перейти до пов’язаного обговорення або додати книгу до списку обраного.

Рисунок 1.14. Сторінка перегляду інформації книги у веб застосунку

Особливу увагу приділено адаптивності. Незважаючи на те, що повноцінна оптимізація під мобільні пристрої ще не завершена, структура макету вже передбачає можливість масштабування, зміну сітки елементів та використання медіазапитів у CSS. У перспективі передбачається повна підтримка смартфонів та планшетів з окремими стилями для вузьких екранів.

Важливою особливістю інтерфейсу є використання єдиного візуального стилю: обрана кольорова палітра є стриманою, з перевагою пастельних відтінків, що створює комфортну атмосферу для читання. Активні елементи, такі як кнопки, посилання або сповіщення, виділяються за допомогою контрасту, при цьому не створюючи візуального навантаження.

Для реалізації frontend у використовувалися класичні веб-технології – HTML, CSS та JavaScript. Основна частина логіки відображення покладається на серверну генерацію сторінок через шаблони Django, а динамічна поведінка реалізована за допомогою JavaScript-функцій, які взаємодіють із REST API. Це дозволяє поєднати швидкість завантаження та зручність інтерактивної взаємодії.

Підводячи висновки, інтерфейс веб-порталу розроблено з урахуванням принципів доступності, зручності, інтуїтивної зрозумілості та збереження загальної стилістичної єдності. На сьогоднішній день, питання зручності графічного інтерфейсу стає як ніколи важливим. Такий підхід дозволяє створити

комфортні умови для кожного учасника віртуального клубу, сприяючи регулярному використанню платформи.

1.3 Реалізація веб системи

1.3.1 Проєктування структури бази даних

Процес розробки веб-застосунку у сучасному інженерному середовищі передбачає чітку організацію інструментів, технологій і процедур, які забезпечують ефективність, повторюваність та стабільність результатів. Формування стабільного та керованого середовища розробки є однією з основоположних передумов для запобігання технічним помилкам, що виникають унаслідок конфігураційних відмінностей між середовищами розробників, а також для забезпечення пришвидшеного розгортання і спрощення масштабування системи.

У контексті обраної монолітної архітектури проєкту було ухвалено рішення реалізувати всі функціональні складові в межах єдиного застосунку на базі фреймворку Django. Зокрема, модулі, що відповідають за автентифікацію, реєстрацію, обробку запитів користувача, взаємодію з Google Books API, відображення інформації про книги, обговорення та пошук, були інтегровані в один логічно цілісний програмний блок. Такий підхід дозволяє спростити розгортання, зменшити накладні витрати на підтримку взаємодії між сервісами та значно підвищити узгодженість логіки обробки даних.

Враховуючи потребу у формуванні однакового середовища на всіх етапах життєвого циклу програмного забезпечення – від локальної розробки до тестування і продуктивного розгортання – ключовим рішенням стала контейнеризація проєкту з використанням інструментарію Docker. Це дало змогу створити ізольоване, повторюване середовище з попередньо визначеним набором залежностей, системних утиліт, бібліотек та конфігурацій, яке гарантує стабільність та відсутність неочікуваних збоїв, пов’язаних із відмінностями середовищ.

На етапі налаштування середовища розробки було використано ОС Windows

як основну платформу для роботи, що зумовлено її широкою розповсюдженістю, підтримкою інструментів віртуалізації, а також наявністю зручного програмного забезпечення для розробки. У ролі основного середовища розробки було обрано IDE PyCharm, що забезпечує інтеграцію з системами контролю версій (зокрема Git), підтримку роботи з Docker-контейнерами, засоби інтерактивного налагодження коду, автодоповнення та зручне керування проєктною структурою.

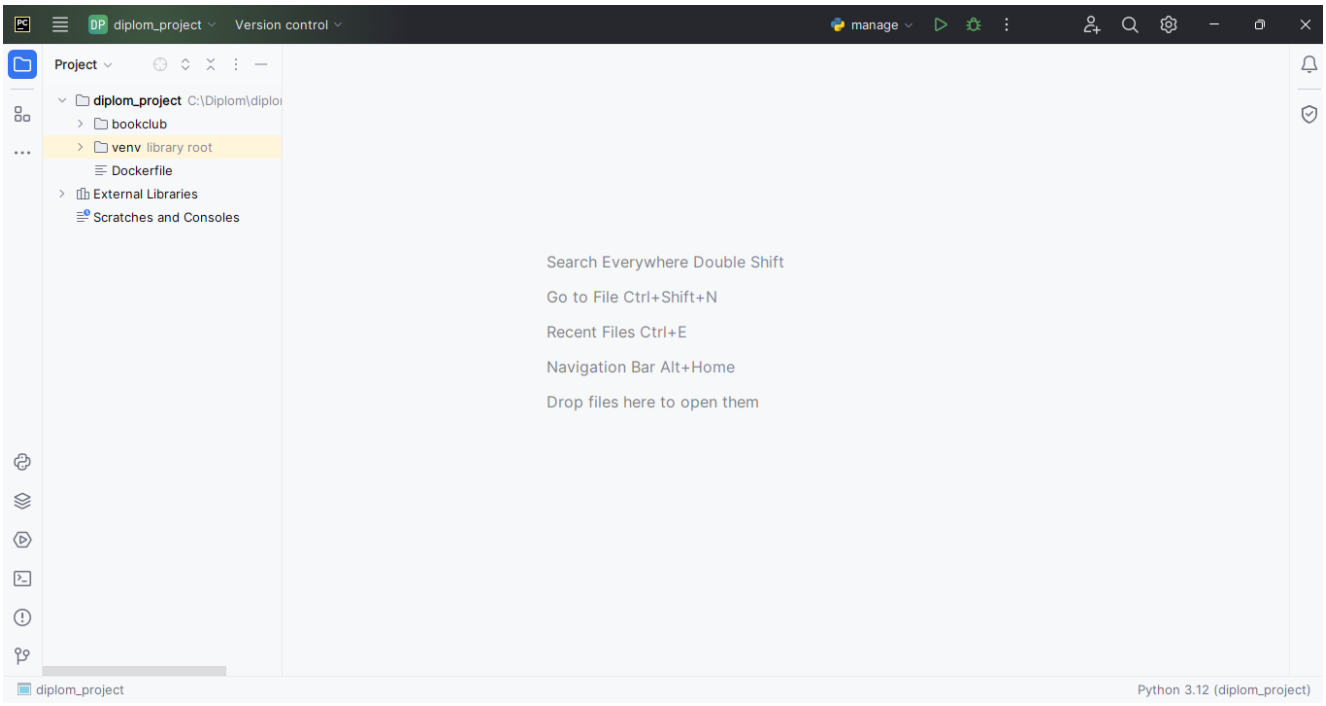


Рисунок 1.15. Інтерфейс IDE PyCharm

Організація структури проекту в монолітному форматі дозволила нам уникнути поділу на окремі каталоги для кожного сервісу , натомість реалізувавши єдину структуру Django з відповідними додатками, які реалізують окремі частини логіки. Наприклад, в рамках одного додатку Django були створені модулі для роботи з Google Books API, який раніше планувалося реалізувати як окремий сервіс на Flask або FastAPI. Як ми бачимо, функціонал для взаємодії із зовнішнім API був інтегрований безпосередньо в серверну частину Django, що спростило як маршрутизацію, так і логіку контролерів і моделей.

Контейнеризація із використанням Docker включала створення спеціалізованого Dockerfile для побудови образу застосунку, у якому було визначено базовий образ з Python, інсталяцію необхідних бібліотек (зокрема Django, requests, djangorestframework, gunicorn тощо), копіювання вихідного коду,

налаштування середовища виконання та запуску.

Важливою складовою конфігурації стало забезпечення безперервного зворотного зв’язку між змінами у кодовій базі та результатами їх виконання у контейнеризованому середовищі. Це було реалізоано за допомогою механізмів автоматичного перезапуску контейнерів при зміні вихідного коду, що дозволяє миттєво бачити ефекти внесених змін без потреби ручного втручання у процес оновлення.

Інтеграція модулів у межах одного застосунку дозволила скоротити час на координацію логіки між компонентами, знизити складність налаштування авторизації між сервісами та зробити проєкт більш керованим у рамках обмеженого обсягу дипломної роботи. З точки зору безпеки, архітектура була побудована із врахуванням обмеження доступу до API-запитів із зовнішнього середовища, обробки помилок на рівні middleware, а також логування ключових подій для можливого аудиту.

Слід відзначити, що підхід, обраний у проєкті, відповідає сучасним DevOps-практикам, де автоматизація, стандартизація середовища та контрольований процес деплою розглядаються як необхідні умови для створення якісного та довготривало підтримуваного програмного забезпечення. Завдяки застосуванню Docker, система стала незалежною від середовища операційної системи, забезпечила можливість швидкого відтворення на будь-якій машині, а також надала фундамент для потенційної CI/CD інтеграції у майбутньому.

У підсумку, реалізація середовища розробки на основі монолітної архітектури у поєднанні з контейнеризацією за допомогою Docker дозволила досягти високого рівня керованості, узгодженості, стабільності та масштабованості веб-застосунку. Це, у свою чергу, стало запорукою успішного завершення проєкту в рамках навчального процесу та забезпечило передумови для його подальшого розвитку.

1.3.2 Реалізація сервісу

Сервіс є центральним логічним вузлом архітектури веб-застосунку, що відповідає за обробку основної функціональності, пов’язаної з аутентифікацією

користувачів, організацією обговорень, системою маршрутизації запитів, а також збереженням та обробкою структурованих даних. Його побудовано з урахуванням принципів модульності, масштабованості та розширюваності, що дозволяє ефективно інтегрувати нові компоненти без порушення загальної структури системи.

Основою для реалізації цього сервісу слугує фреймворк Django – потужний інструмент для побудови веб-застосунків із чіткою архітектурною побудовою на основі патерна Model–View–Template (MVT). Однією з ключових переваг Django є його «всеосяжна» структура, що дозволяє значною мірою стандартизувати підхід до розробки, уникнути дублікації логіки та мінімізувати помилки, пов'язані з несумісністю між модулями [12].

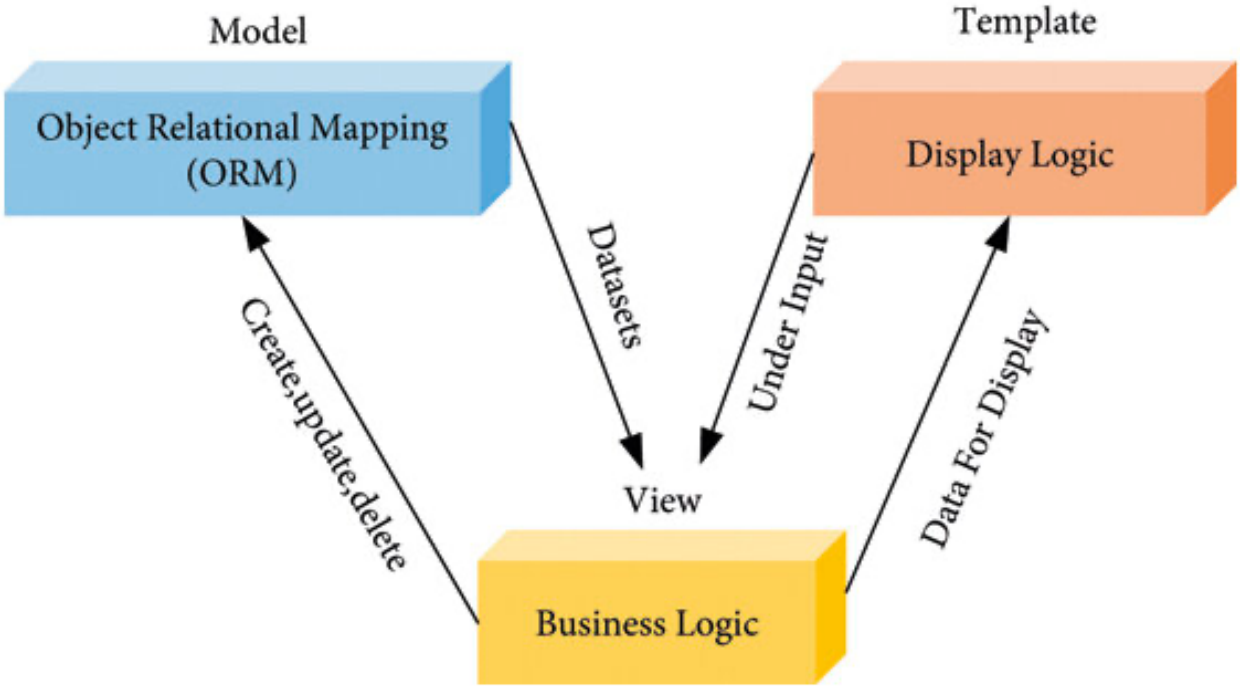


Рисунок 1.16. Архітектура патерна MVT

Першим етапом у створенні сервісу стало ініціалізування проєкту за допомогою стандартного механізму django-admin startproject, після чого було створено окремий додаток users для обробки логіки реєстрації, авторизації та управління обліковими записами. Такий підхід відповідає принципу відокремлення відповідальностей (Separation of Concerns), що дозволяє

локалізувати зміни, пов’язані з конкретною функціональністю, у межах одного модуля.

Для забезпечення зберігання даних було розроблено набір моделей, що формують логічну схему бази даних. Всі моделі створювались із урахуванням майбутньої потреби в нормалізації даних, а також із розумінням потенційного навантаження, яке може виникнути при зростанні кількості користувачів. Усі зовнішні зв’язки були реалізовані через ForeignKey, ManyToManyField та OneToOneField, залежно від контексту.

Окремо було реалізоано систему обговорень, яка дозволяє користувачам створювати теми, залишати коментарі, відповідати на чужі пости, а також здійснювати пошук серед наявних обговорень. Для цього створено моделі Discussion, Post, Comment, які формують ієрархічну структуру взаємодії між учасниками. Ця структура забезпечує логічну узгодженість даних та дає можливість реалізувати як прямі, так і вкладені відповіді. Крім того, було реалізоано механізм тегування, який дозволяє класифікувати обговорення за тематиками.

Маршрутизація в рамках Django здійснюється за допомогою системи URL-шаблонів. Кожен функціональний блок – автентифікація, перегляд профілю, створення постів, участь в обговореннях – має власний набір маршрутів, що формуються в окремих файлах urls.py відповідних застосунків. Це дозволяє не лише полегшити навігацію проєктом, але й забезпечити гнучку систему керування доступом. Авторизовані користувачі отримують розширений функціонал, тоді як анонімним відвідувачам надається обмежений набір функцій у рамках принципу least privilege.

Особлива увага приділялася шаблонізації інтерфейсу – системі, яка реалізує представлення даних для кінцевого користувача. Використання шаблонів Django із вбудованими фільтрами, тегами та контекстними змінними дозволило забезпечити як гнучкість, так і безпеку у виведенні динамічного контенту. Крім того, вся верстка інтерфейсу була адаптована під різні типи пристроїв, що сприяє доступності платформи.

Ще одним важливим аспектом було впровадження системи обробки винятків та помилок. Усі критичні маршрути сервісу були забезпечені механізмами обробки виняткових ситуацій, таких як відсутність об'єкта, спроба несанкціонованого доступу, недопустимі запити тощо. Це підвищило як стабільність застосунку, так і рівень довіри з боку користувачів.

На етапі тестування функціональності сервісу були використані вбудовані засоби Django для написання unit-тестів. Було протестовано основну бізнес-логіку: створення користувачів, авторизація, створення та редагування постів, робота з пошуком та фільтрами. Результати тестування підтвердили коректність реалізації й готовність сервісу до інтеграції з іншими мікросервісами.

Виходить, реалізація сервісу стала наріжним каменем проєкту. Вона не лише забезпечила базову функціональність, але й задала високі стандарти якості коду, структурної організації та відповідності принципам інженерного проєктування. Логічна побудова, суворе дотримання архітектурних патернів та продумана взаємодія з іншими сервісами дозволяють розглядати як гнучкий, безпечний і масштабований компонент загальної системи.

1.3.3 Реєстрація та авторизація користувачів

У контексті розробки сервісу ключову роль відіграє механізм автентифікації та авторизації користувачів, адже саме цей етап є початковою точкою взаємодії між відвідувачем платформи та її функціональністю. Ефективна реалізація входу до системи є не лише технічним викликом, а й важливим чинником загального користувацького досвіду, оскільки впливає на довіру, безпеку та зручність доступу до персоналізованого контенту.

Основу функціональності автентифікації становить класичний механізм реєстрації облікового запису за допомогою електронної пошти та пароля. Такий підхід передбачає перевірку унікальності введених даних, зокрема адреси електронної пошти, а також дотримання вимог до складності пароля з метою забезпечення базового рівня безпеки. При реалізації серверної логіки реєстрації враховано необхідність серверної валідації даних та обробки помилок, пов’язаних із дублюванням облікових записів або некоректним форматом введеної інформації.

Важливим компонентом стала реалізація перевірки автентичності через обліковий запис Google, що дозволяє користувачеві увійти на платформу без необхідності створення нового локального профілю. Для цього було інтегровано механізм OAuth 2.0, який дозволяє безпечно делегувати авторизацію сторонньому провайдеру. На відміну від традиційного входу, цей підхід не вимагає збереження паролів у базі даних системи, що позитивно впливає на рівень інформаційної безпеки. Реалізація OAuth передбачала налаштування облікового запису в Google Cloud Console, реєстрацію редірект-URI, створення облікових даних додатка та інтеграцію логіки з Django через відповідні бібліотеки авторизації [9].

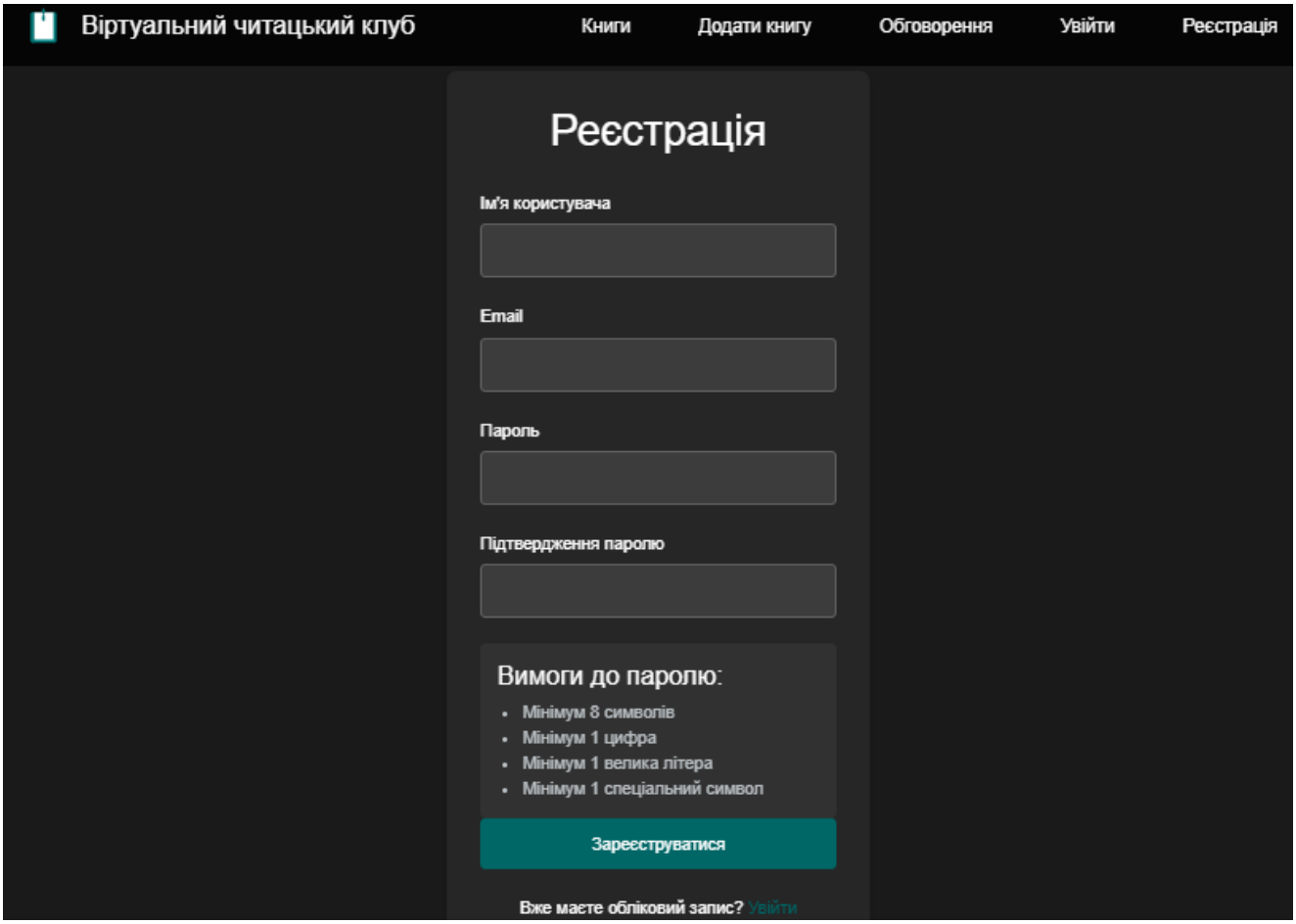


Рисунок 1.17. Сторінка реєстрації користувачів

У процесі побудови механізму входу та реєстрації особливу увагу приділено структурі маршрутів і обробці переходів між станами: від реєстрації – до підтвердження, від логіна – до авторизованого інтерфейсу. Було створено шаблони сторінок з формами введення, які інтегруються з backend-логікою через POST-запити. Передбачено обробку стандартних сценаріїв помилок – наприклад,

неправильний пароль, відсутність користувача або спроба повторної реєстрації.

Для збереження інформації про користувача використовується модель User, розширена за допомогою механізму кастомного user model у Django, що забезпечує гнучкість при розширенні атрибутів облікового запису у майбутньому. Активація користувачів відбувається автоматично після завершення Google-авторизації або вручну після підтвердження через стандартний інтерфейс. Збереження сесій реалізоано на основі cookie-механізму, що дозволяє підтримувати авторизований стан користувача впродовж заданого інтервалу.

```
def register_view(request):
    """Регістрація"""
    if request.method == 'POST':
        form = RegisterForm(request.POST)
        if form.is_valid():
            user = form.save()
           login(request, user, backend='django.contrib.auth.backends.ModelBackend')
            return redirect('index')
    else:
        form = RegisterForm()
    return render(request, 'core/register.html', {'form': form})
```

У цьому блоці коду реалізовано процес реєстрації нових користувачів. Коли відправляється форма, перевіряються правильність даних і створюється обліковий запис. Після цього користувач автоматично входить у систему, щоб одразу розпочати роботу з сайтом.

```
def login_view(request):
    """Логін"""
    if request.method == 'POST':
        form = AuthenticationForm(request, data=request.POST)
        if form.is_valid():
            user = form.get_user()
            login(request, user)
            return redirect('index')
    else:
        form = AuthenticationForm()
    return render(request, 'core/login.html', {'form': form})
```

Цей обробник відповідає за вхід користувачів у систему. Якщо користувач заповнив форму та дані вірні, система авторизує його та перекидає на головну сторінку. Це дозволяє користувачеві отримати доступ до персоналізованих розділів сайту.

```
def logout_view(request):
    """Вихід з облікового запису"""
    logout(request)
    return redirect('index')
```

Виконуючи функцію виходу, цей обробник видаляє авторизаційні дані користувача із сесії. З цього можна вивести результати що, користувач гарантовано покидає свій обліковий запис і повертається на головну сторінку, щоб зберегти безпеку облікового запису.

У цьому контексті особливу роль відіграє інтеграція двох систем – локального входу та входу через сторонній провайдер. Складність полягає у необхідності забезпечити однаковий інтерфейс користувача, незалежно від обраного способу автентифікації, а також коректну обробку сесій, щоб уникнути дублювання профілів. Було реалізоано перевірку наявності користувача з такою ж електронною адресою та автоматичне злиття облікових записів у разі співпадіння.

Крім того, передбачено перевірку CSRF-токенів у формах входу, що захищає систему від міжсайтових запитів, а також реалізоано обробку некоректних запитів із повідомленням користувачеві про помилку. Весь функціонал проходив етап тестування з акцентом на типові сценарії поведінки користувача, включаючи відновлення пароля, перевірку правильності редіректів після входу, недопущення неавторизованих доступів до захищених маршрутів та інші аспекти.

Виходить, механізм автентифікації та авторизації, реалізований у межах - сервісу , є не просто технічним модулем, а ключовим елементом взаємодії між користувачем і системою, що поєднує зручність, безпеку та гнучкість в архітектурному плані. Він формує основу довіри до платформи, створює передумови для персоналізованого користування й водночас забезпечує відповідність сучасним стандартам безпеки та зручності.

1.3.4 Створення моделей, представлень, маршрутизація

Після успішного впровадження механізмів автентифікації та авторизації користувачів одним із ключових етапів реалізації стало створення моделей, представлень і маршрутизації. Ці компоненти визначають логіку збереження, обробки та представлення даних, а також формують основу для інтерактивної взаємодії між користувачем і платформою. Саме в цій частині розробки відбувається перехід від абстрактної архітектурної моделі до конкретного програмного втілення функціональних можливостей.

Структура моделей розроблялася з урахуванням логічної цілісності та потенційного масштабування. В основі лежить концепція нормалізованої бази даних, яка дозволяє мінімізувати дублювання інформації та полегшити підтримку даних у майбутньому. Основними сутностями, представленими у вигляді Django-моделей, стали: профіль користувача, тема обговорення, коментар, закладки, а також допоміжні структури, що пов’язані із системною аналітикою або обліком активності [12].

Для моделювання обговорень реалізоано модель Discussion, яка містить зв’язок із автором (ForeignKey до User), посилання на ідентифікатор книги, текст обговорення, мітку часу створення, а також статус публікації. Це дозволяє користувачам створювати теми на обговорення окремих книг, зберігаючи при цьому гнучкий зв’язок між мікросервісами. Коментарі до обговорень реалізоано через модель Comment, яка пов’язується як із користувачем, так і з відповідною темою, що забезпечує вкладену логіку та дозволяє будувати багаторівневу систему відгуків.

```
User = get_user_model()


class Book(models.Model):
    title = models.CharField(max_length=200)
    author = models.CharField(max_length=200)
    description = models.TextField()
    genre = models.CharField(max_length=100)
    thumbnail = models.URLField()
    google_books_id  =  models.CharField(max_length=100,  unique=True,  null=True,
blank=True)
    preview_link = models.URLField(null=True, blank=True)
    published_year = models.CharField(max_length=4, null=True, blank=True)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    def __str__(self):
        return self.title

    @property
    def average_rating(self):
        return round(self.ratings.aggregate(models.Avg('rating'))['rating__avg'] or
0, 2)


class Discussion(models.Model):
    book = models.ForeignKey(Book, on_delete=models.CASCADE)
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    comment = models.TextField()
    posted_at = models.DateTimeField(auto_now_add=True)
```

```
    def __str__(self):
        return f'Обговорення для {self.book.title}'


class DiscussionComment(models.Model):
    discussion = models.ForeignKey(Discussion, on_delete=models.CASCADE,
related_name='comments')
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    text = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)

    class Meta:
        ordering = ['-created_at']

    def __str__(self):
        return f'Коментар від {self.user.username} до обговорення
{self.discussion.book.title}'


class Comment(models.Model):
    book = models.ForeignKey(Book, on_delete=models.CASCADE,
related_name='comments')
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    text = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)

    class Meta:
        ordering = ['-created_at']

    def __str__(self):
        return f'Коментар від {self.user.username} до {self.book.title}'


class BookView(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE,
related_name='book_views')
    book = models.ForeignKey(Book, on_delete=models.CASCADE, related_name='views')
    viewed_at = models.DateTimeField(default=timezone.now)

    class Meta:
        ordering = ['-viewed_at']
        # Додаємо унікальне обмеження, щоб у користувача був лише один запис для
кожної книги
        unique_together = ['user', 'book']

    def save(self, *args, **kwargs):
        # Оновлюємо час перегляду при повторному перегляді
        self.viewed_at = timezone.now()
        super().save(*args, **kwargs)


class BookRating(models.Model):
    book = models.ForeignKey(Book, on_delete=models.CASCADE, related_name='ratings')
    user = models.ForeignKey(User, on_delete=models.CASCADE,
related_name='book_ratings')
    rating = models.PositiveSmallIntegerField(choices=[(i, str(i)) for i in range(1,
6)])  # 1-5 зірок
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    class Meta:
        unique_together = ('book', 'user')
```

```
    def __str__(self):
        return f"{self.user.username} – {self.book.title}: {self.rating}★"
```

Цей код описує моделі для роботи з книгами, їхніми обговореннями, коментарями, переглядами та оцінками в Django. Модель Book містить основну інформацію про книгу, а інші моделі пов'язані з нею через зовнішні ключі, що дозволяє зберігати дискусії, коментарі користувачів, записи переглядів і рейтинги.

Для кожного користувача фіксуються перегляди книг з унікальним обмеженням, щоб уникнути дублювання, а також рейтинги від 1 до 5 зірок, які агрегуються для отримання середньої оцінки книги. Це забезпечує комплексну взаємодію з контентом і спільнотою.

З точки зору маршрутизації, в проєкті використано модуль urls.py, який визначає відповідність між URL-шляхами та відповідними представленнями. Структура маршрутів сформована ієрархічно, відповідно до REST-підходу: шляхи типу /discussions/, /comments/ дають змогу легко масштабувати інтерфейс програмування без порушення логічної структури. Для полегшення навігації між маршрутами на боці клієнта реалізоано систему іменованих маршрутів, що дозволяє викликати представлення через символічні посилання, а не через жорстко закодовані шляхи.

```
urlpatterns = [
    path('', views.index, name='index'),
    path('add-book/', views.add_book, name='add_book'),
    path('register/', views.register_view, name='register'),
    path('login/', views.login_view, name='login'),
    path('logout/', views.logout_view, name='logout'),
    path('search/', views.book_search, name='book_search'),
    re_path(r'^book/(?P<book_id>[^/]+)/$', views.book_detail, name='book_detail'),
    path('fetch-books/', views.fetch_and_save_books, name='fetch_books'),
    path('update-books/', views.update_books, name='update_books'),
                    path('recommendations/',          views.book_recommendations,
name='book_recommendations'),
    path('discussions/', views.discussions_list, name='discussions_list'),
              path('discussions/<int:discussion_id>/',      views.discussion_detail,
name='discussion_detail'),
    path('discussions/create/', views.create_discussion, name='create_discussion'),
    path('book-search/', views.book_search, name='book_search'),
    path('book-search-api/', views.book_search_api, name='book_search_api'),
]
```

Цей фрагмент коду визначає маршрути URL для вебзастосунку на Django. Кожен шлях відповідає певній функції у views, яка обробляє запити користувачів,

наприклад, показ головної сторінки, додавання книги, реєстрацію, вхід і вихід, пошук книг, перегляд деталей книги, обговорень та отримання рекомендацій.

Особливу увагу під час побудови моделей і маршрутів приділено обмеженням доступу та контролю доступу до даних. Для цього застосовано класи дозволів (permissions) і обмеження на рівні запитів. Наприклад, лише автор коментаря має право змінювати або видаляти його, а обговорення можна редагувати тільки користувачеві, що його створив. Інші користувачі мають лише доступ до читання. Це забезпечує не лише функціональну цілісність даних, але й дотримання принципів прозорості й захищеності користувацького контенту.

Усі реалізовані маршрути також були протестовані за допомогою інструментів Postman та Django-тестів, що дало змогу переконатися у коректності обробки даних, відповідності статус-кодів, правильності обробки помилок та повернення даних у потрібному форматі.

З цього виходить що, створення моделей, представлень і маршрутизації стало ядром функціонального шару сервісу , забезпечивши ефективну взаємодію між користувачем, даними і зовнішніми сервісами. У поєднанні з правильно спроєктованою структурою даних це дозволяє гнучко адаптувати веб-систему до змін функціональних вимог і забезпечити її розширення в майбутньому.

1.3.5 Реалізація пошуку книжок

Організація ефективної навігації в межах масштабного цифрового контенту потребує використання динамічного та адаптивного пошукового механізму. У контексті онлайн простір віртуального клубу читачів ця потреба набуває особливої актуальності, оскільки саме зручність і точність доступу до книжкових даних значною мірою формують користувацький досвід. У зв’язку з цим пошук реалізоано не як додаткову функцію, а як структурну складову інтерфейсу взаємодії, що істотно впливає на загальну якість системи.

З метою розширення інформаційної бази й забезпечення актуальності контенту було ухвалене рішення про інтеграцію Google Books API як зовнішнього джерела даних. Такий підхід дозволив уникнути потреби в ручному наповненні та оновленні бази книжок, водночас відкривши доступ до багатомовного й тематично

різноманітного бібліографічного масиву. Завдяки цьому система отримала можливість оперативно надавати користувачам релевантні результати пошуку без значного навантаження на локальні ресурси. Інтеграція Google Books API також сприяла підвищенню якості пошукових запитів, дозволяючи користувачам отримувати детальну інформацію про книги, включаючи анотації, відгуки та доступ. Крім того, автоматизоване оновлення даних через API забезпечило стабільну актуальність контенту навіть при швидкому розширенні бібліотеки.

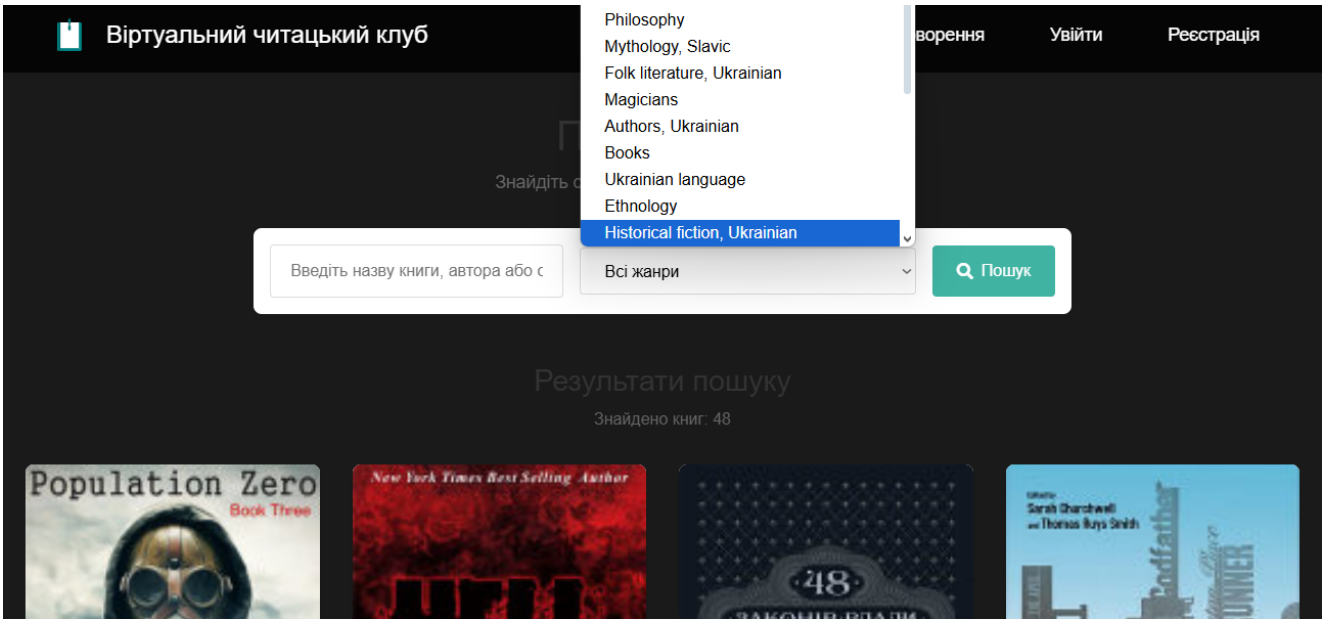


Рисунок 1.18. Сторінка пошуку у проекті

Особлива увага в межах реалізації була зосереджена на забезпеченні стійкості до варіативності користувацьких запитів. У реальних умовах введення часто супроводжується орфографічними помилками, неточними формулюваннями або неповними назвами. Для підвищення чутливості пошукової системи реалізоано механізми первинної нормалізації тексту: зокрема, видаляються надлишкові пробіли, застосовується перетворення до нижнього регістру, а також часткова лематизація. Попри обмеженість цих заходів у порівнянні з повноцінною семантичною обробкою, вони дозволяють суттєво підвищити точність видачі.

Результати пошуку формуються у вигляді структурованого JSON-об’єкта, який містить ключову бібліографічну інформацію: назву книги, авторів, короткий опис, дату публікації та URL-адресу зображення обкладинки. На рівні інтерфейсу ці дані відображаються у форматі карток, організованих у центральній частині

сторінки. При цьому візуальна частина спроєктована з урахуванням таких критеріїв, як читабельність, швидкість завантаження та адаптивність, що забезпечує комфортну взаємодію як на стаціонарних, так і на мобільних пристроях.

```
def book_search(request):
    """Пошук книг"""
    query = request.GET.get('q', '')
    genre = request.GET.get('genre', '')
    page = request.GET.get('page', 1)

    # Спочатку шукаємо книги в локальній базі
    books = Book.objects.all()

    if query:
        books = books.filter(
            Q(title__icontains=query) |
            Q(author__icontains=query) |
            Q(description__icontains=query)
        )

    if genre:
        books = books.filter(genre=genre)

    # Якщо локальний пошук не дав результатів, шукаємо через Google Books API
    if not books.exists() and query:
        try:
            # Формуємо URL для запиту до Google Books API з фільтром за українською
та англійською мовами
            api_url                                                              =
f"https://www.googleapis.com/books/v1/volumes?q={query}&maxResults=40&langRestrict=
uk,en"
            if genre:
                api_url += f"&subject:{genre}"

            response = requests.get(api_url)
            data = response.json()

            if 'items' in data:
                for item in data['items']:
                    volume_info = item['volumeInfo']

                    # Перевіряємо мову книги
                    language = volume_info.get('language', '')
                    if language not in ['uk', 'en']:
                        continue

                    # Перевіряємо наявність російських букв у назві та авторі
                    title = volume_info.get('title', '').lower()
                    author   =   ',   '.join(volume_info.get('authors',   ['Невідомий
автор'])).lower()

                    # Список російських букв для перевірки
                    russian_chars = 'ёъыэ'
                    russian_words  =  ['россия',  'российский',  'русский',  'русь',
'москва', 'петербург']

                    # Перевіряємо наявність російських букв і слів
                    if (any(char in title for char in russian_chars) or
                        any(char in author for char in russian_chars) or
                        any(word in title for word in russian_words) or
```

```
                        any(word in author for word in russian_words)):
                        continue

                    # Перевіряємо, чи вже є така книга в базі
                    if                                                    not
Book.objects.filter(google_books_id=item['id']).exists():
                        # Створюємо нову книгу
                        book = Book(
                            title=volume_info.get('title', ''),
                            author=', '.join(volume_info.get('authors', ['Невідомий
автор'])),
                            description=volume_info.get('description', ''),
                            genre=genre if genre else volume_info.get('categories',
['Невідомий жанр'])[0],
                            thumbnail=volume_info.get('imageLinks',
{}).get('thumbnail', ''),
                            google_books_id=item['id'],
                            preview_link=volume_info.get('previewLink', '')
                        )
                        book.save()

                # Після додавання книг, робимо новий запит до локальної бази
                books = Book.objects.all()
                if query:
                    books = books.filter(
                        Q(title__icontains=query) |
                        Q(author__icontains=query) |
                        Q(description__icontains=query)
                    )
                if genre:
                    books = books.filter(genre=genre)
        except Exception as e:
            print(f"Помилка при отриманні даних з Google Books API: {e}")

    # Пагінація
    paginator = Paginator(books, 12)
    try:
        books = paginator.page(page)
    except PageNotAnInteger:
        books = paginator.page(1)
    except EmptyPage:
        books = paginator.page(paginator.num_pages)

    # Отримуємо список усіх жанрів для фільтрації
    genres = Book.objects.values_list('genre', flat=True).distinct()

    context = {
        'books': books,
        'query': query,
        'genre': genre,
        'genres': genres,
    }

    return render(request, 'core/book_search.html', context)
```

Ця функція реалізує пошук книг у Django. Спочатку вона шукає книги у локальній базі даних за назвою, автором або описом, а також фільтрує за жанром. Якщо результатів немає, виконується запит до Google Books API, де відбираються книги лише українською або англійською мовами та відсіюються ті, що містять

російські літери чи слова. Знайдені книги зберігаються в базі даних. Далі результати відображаються з пагінацією (по 12 книг на сторінку) та передаються в шаблон разом зі списком жанрів.

Додаткову гнучкість забезпечує впровадження розширених фільтрів і механізмів сортування. Користувач має змогу звузити результати за такими параметрами, як жанр, рік публікації чи автор, що значно підвищує релевантність видачі. Ці функції базуються на метаданих, отриманих із API, проте їх обробка локально дозволяє створити ілюзію роботи з персоналізованою колекцією. Візуально фільтри реалізовані у вигляді інтуїтивно зрозумілих елементів інтерфейсу – випадаючих списків, кнопок категорій та індикаторів популярності, що підвищує загальну зручність і заохочує до дослідження контенту.

Одним із викликів при реалізації системи пошуку стала неповнота даних, що надходять із Google Books API. У деяких випадках записи не містять необхідних полів – таких як ім’я автора, опис або дата видання. Задля підвищення якості відображуваного контенту реалізоано механізм валідації отриманої інформації: записи з критичною відсутністю метаданих автоматично фільтруються, що гарантує користувачу доступ лише до змістовно повних результатів.

На наступному етапі розвитку функціоналу передбачається впровадження базових елементів персоналізованої рекомендаційної системи. Зокрема, планується аналіз історії пошукових запитів і формування індивідуальних добірок на основі користувацьких уподобань. Такий підхід дозволить перейти від пошуку за ключовими словами до контекстно адаптованої взаємодії, що значною мірою підвищить цінність платформи як інтелектуального середовища для читачів.

1.3.6 Робота з Google Books API

Робота з Google Books API є ключовим елементом у межах сервісу , що забезпечує зв’язок між основним ядром платформи та ресурсами Google для отримання книжкових даних. Задачею цього етапу було створення надійної, масштабованої інтеграції з Google Books API, враховуючи специфіку запитів, обмеження API та вимоги до архітектурної стабільності системи.

Для інтеграції веб-порталу з Google Books API було необхідно створити відповідний ключ доступу в хмарному сервісі Google Cloud. Цей процес передбачає реєстрацію проєкту в Google Cloud Console, активацію API Google Books та генерацію унікального API-ключа, який використовується для автентифікації запитів до стороннього сервісу. Отриманий ключ було збережено в конфігураційних налаштуваннях веб-застосунку, що забезпечує можливість надсилання HTTP-запитів до Google Books API для отримання інформації про книги, авторів, жанри, обкладинки тощо. Такий підхід гарантує безпечну взаємодію з зовнішнім API та дозволяє масштабувати функціонал системи без суттєвих змін у програмному коді [14].

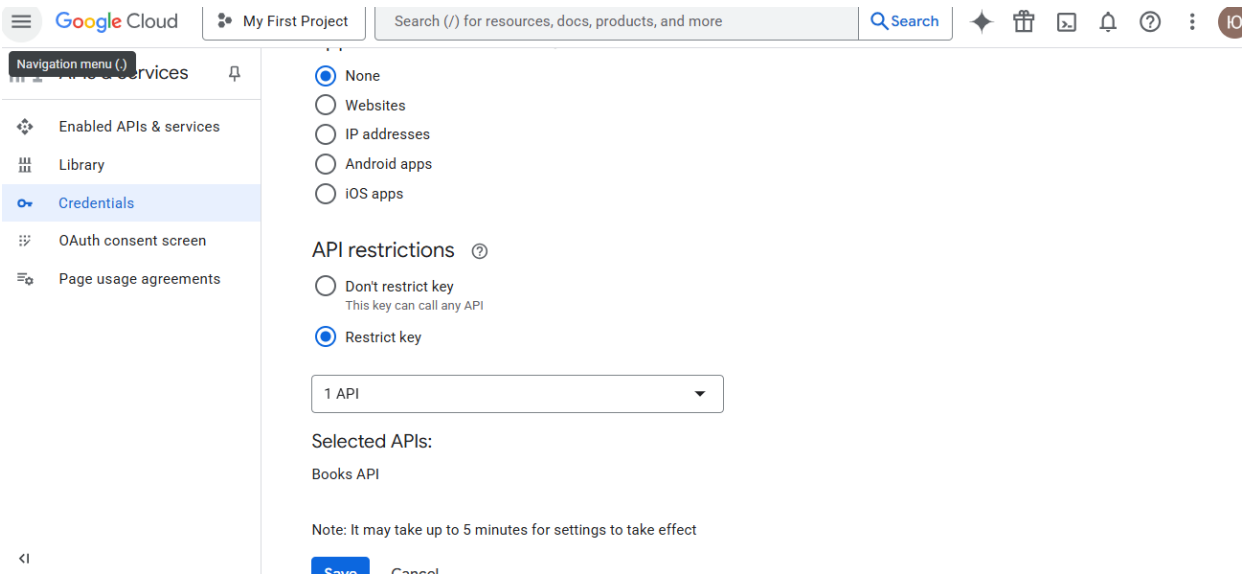


Рисунок 1.19. Створення ключів до Google Books API

Взаємодія з API здійснюється через HTTP-запити, що формуються на основі параметрів, заданих користувачем або -сервісом через REST API. Для забезпечення гнучкості пошуку запити підтримують різноманітні типи, включаючи загальний пошук за ключовими словами, ISBN, автором, назвою чи жанром. В обробці запитів у використовується серіалізація параметрів відповідно до синтаксису Google Books API (наприклад, intitle:, inauthor:, subject:), що дозволяє створювати точніші запити.

Обробка відповіді API є важливим етапом, оскільки JSON-структури мають складну ієрархічну форму, де багато полів є необов'язковими або мати різну форму залежно від джерела. Для цього виконує роль нормалізатора: перетворює дані у

внутрішній формат, перевіряючи на null-значення та замінюючи дефолтні значення, якщо дані відсутні. Структура відповіді приводиться до єдиного шаблону, що полегшує обробку даних.

Окрім цього, реалізоано фільтрацію нерелевантних результатів, таких як журнали, комікси або книги без обкладинок та важливих метаданих. Записи без назви, автора або з типом magazine виключаються з результатів. Для забезпечення точності пошукових результатів також передбачено пріоритетне сортування, яке враховує повноту метаданих, мову оригіналу, наявність фрагменту книги та популярність запису.

Щоб підвищити продуктивність, система реалізує кешування запитів. Перед відправкою запиту до Google Books API перевіряється наявність його відповіді в кеші (Redis або іншому кешуючому шарі). Якщо відповідь уже є, вона витягується з кешу, що зменшує навантаження на API та підвищує швидкість роботи системи.

Ще однією важливою складовою є обробка помилок, пов’язаних із мережевими неполадками або квотами API. У разі помилок, таких як 403 (Forbidden) або 429 (Too Many Requests), система повертає чітко структуроване повідомлення про помилку з кодом, описом та рекомендаціями для повторного запиту.

Як бачимо, реалізація взаємодії з Google Books API в дозволила створити гнучку, надійну архітектуру, що здатна адаптуватися до змін у зовнішньому API та мінімізувати вплив зовнішніх факторів на стабільність системи. Цей компонент не лише забезпечує доступ до зовнішніх книжкових даних, але й є важливим елементом захисту користувачів від нестабільності API.

1.3.7 Обробка запитів до стороннього API

Інтеграція з Google Books API є ключовою функціональною складовою розробленої онлайн простір та реалізується безпосередньо в межах монолітного Django-застосунку. Незважаючи на відмову від мікросервісної архітектури, усі раніше заплановані логічні блоки взаємодії з зовнішнім API були збережені та адаптовані до єдиної структури додатка. В результаті цього інтеграція з Google Books API була реалізована як частина модуля, відповідального за зовнішні дані,

що забезпечує централізовану обробку запитів до API, їхнє логічне групування, обробку помилок, кешування та нормалізацію отриманих результатів.

Головною метою реалізації даного функціоналу є створення надійного, гнучкого та масштабованого механізму доступу до глобального репозитарію книжкової інформації, що надається компанією Google через відкритий API. Цей механізм має відповідати вимогам до швидкодії, стабільності, захисту від перевантажень, а також до якості і точності отриманих даних. Зважаючи на це, в Django-проєкті була реалізована ціла низка внутрішніх механізмів, які дозволяють не лише ефективно надсилати запити, але й обробляти й адаптувати відповіді відповідно до потреб кінцевого користувача.

Формування HTTP-запитів до Google Books API виконується на основі параметрів, які задаються або безпосередньо через користувацький інтерфейс, або генеруються програмно – наприклад, у процесі формування рекомендацій. Система підтримує декілька типів пошукових сценаріїв: загальний пошук за ключовими словами, пошук за автором, назвою книги, ISBN, тематикою чи мовою. Для цього був створений окремий клас генерації запитів, що динамічно формує параметри з урахуванням синтаксису Google Books API (таких як intitle:, inauthor:, subject: тощо). Це дозволяє гнучко комбінувати параметри та створювати запити, що максимально відповідають пошуковим намірам користувача.

Ще одним критично важливим аспектом є обробка відповіді, отриманої від API. Враховуючи складну ієрархічну структуру JSON-документів Google Books, була реалізована окрема система нормалізації та стандартизації даних. Більшість полів у відповіді є необов’язковими, а структура вкладених об’єктів значно варіюється. Наприклад, дані про авторів, видавництво, дату публікації, обкладинку, категорії чи наявність перегляду фрагменту книги можуть бути відсутні або представлені у різних форматах. Відповідно, у Django-проєкті реалізований модуль обробки кожного окремого результату пошуку, який перевіряє наявність ключових полів, заповнює дефолтні значення у разі їх відсутності, проводить конвертацію до уніфікованого формату та зберігає результат у внутрішній структурі даних системи.

```
def index(request):
     """Головна сторінка: спочатку популярні книги з Google Books API, потім
локальні"""
    books_api = []
    try:
                                                                  url               =
'https://www.googleapis.com/books/v1/volumes?q=bestsellers&maxResults=10&orderBy=re
levance'
        response = requests.get(url)
        data = response.json()
        if 'items' in data:
            for item in data['items']:
                volume_info = item.get('volumeInfo', {})
                image_links = volume_info.get('imageLinks', {})
                books_api.append({
                    'title': volume_info.get('title', 'Невідома назва'),
                        'author': ', '.join(volume_info.get('authors', ['Невідомий
автор'])),
                   'description': volume_info.get('description', 'Опис відсутній'),
                    'published_year': volume_info.get('publishedDate', '')[:4],
                      'genre': ', '.join(volume_info.get('categories', ['Невідомий
жанр'])),
                                           'thumbnail':   image_links.get('thumbnail',
'https://via.placeholder.com/300x450?text=No+Image'),
                    'preview_link': volume_info.get('previewLink', '#'),
                    'google_books_id': item.get('id')
                })
    except Exception as e:
        books_api = []

    # Якщо не вдалося отримати з API, показуємо локальні
    if not books_api:
        popular_books = Book.objects.all().order_by('-created_at')[:10]
        books_api = list(popular_books)

    return render(request, 'core/index.html', {'popular_books': books_api})
```

Ця функція index(request) — головна сторінка веб-додатку на Django. Вона спочатку намагається отримати список популярних книг із Google Books API (топ-10 за популярністю), обробляє отримані дані й формує список словників із потрібною інформацією про книги (назва, автор, опис тощо). Якщо API недоступний або сталася помилка, тоді замість цього підвантажуються локальні книги з бази даних (Book.objects.all().order_by('-created_at')[:10]). Далі функція передає список книг у шаблон core/index.html для відображення на сторінці.

Особлива увага приділяється фільтрації нерелевантного або неякісного контенту. У процесі обробки результатів запиту виконується автоматичне виключення записів, що не відповідають мінімальним критеріям якості – наприклад, книжок без назви або автора, журналів (типу magazine), коміксів без метаданих чи записів без обкладинок. Такий підхід дозволяє значно підвищити

релевантність відображуваних результатів для кінцевого користувача та зменшити інформаційне навантаження інтерфейсу.

З метою підвищення продуктивності та зменшення навантаження на зовнішній API, у додатку впроваджено механізм кешування відповідей. Кожен сформований запит перед відправленням проходить перевірку на наявність у кеші, де ідентифікація здійснюється за хешованим ключем, сформованим із параметрів пошуку. Якщо результат уже збережено, він витягується з Redis (або іншого кешуючого шару), минаючи зовнішній API. Це дозволяє зменшити кількість запитів до Google Books API, що особливо важливо через обмеження на кількість запитів, встановлені Google. Додатково це сприяє прискоренню відповіді системи, особливо при повторному пошуку за популярними авторами чи назвами.

Ще одним важливим функціональним блоком є обробка виняткових ситуацій. При роботі з Google Books API існує висока ймовірність виникнення різних типів помилок – мережевих, помилок авторизації, перевищення ліміту запитів (наприклад, помилки типу 403 або 429). У рамках Django-застосунку реалізоано централізований механізм обробки таких помилок, який дозволяє не лише правильно ідентифікувати проблему, але й сформувати стандартизовану відповідь для відображення у frontend-частині або для подальшої обробки іншими модулями. Система повертає користувачеві повідомлення з відповідним кодом помилки, описом ситуації, а також можливими рекомендаціями щодо дій, таких як повторення запиту через певний інтервал часу.

У підсумку, реалізація повного циклу взаємодії з Google Books API у межах єдиного монолітного Django-додатка дозволила досягти високого рівня функціональності, стабільності та масштабованості без необхідності виділення окремого сервісу . Інтеграція із зовнішнім API реалізована у вигляді незалежного програмного модуля всередині проєкту, який виконує роль логічного посередника між зовнішніми джерелами інформації та внутрішніми компонентами платформи. Завдяки цьому вдалося створити архітектуру, що ефективно ізолює складність взаємодії з API від інших частин системи, підвищує надійність у випадку змін у

зовнішньому інтерфейсі та забезпечує користувача максимально точними і релевантними результатами пошуку.

1.4 Тестування веб-системи

1.4.1 Мануальне тестування авторизації та реєстрації

Процес забезпечення надійності функціональних можливостей веб-системи передбачає проведення ретельного тестування критичних модулів, зокрема механізмів авторизації та реєстрації користувачів. Враховуючи важливість цих компонентів для забезпечення безпечного доступу до системи, на першому етапі було реалізоано мануальне тестування, яке дозволило емпірично перевірити логіку роботи кожного кроку взаємодії користувача з інтерфейсом.

Тестування охоплювало декілька типових сценаріїв використання: успішна реєстрація нового користувача, спроба повторної реєстрації з уже використаною електронною поштою, введення некоректного пароля, а також ситуації з пропущеними полями. Для авторизації перевірялись випадки входу з коректними та некоректними обліковими даними, відновлення доступу, а також сценарій, за якого користувач неактивний або обліковий запис не підтверджено.

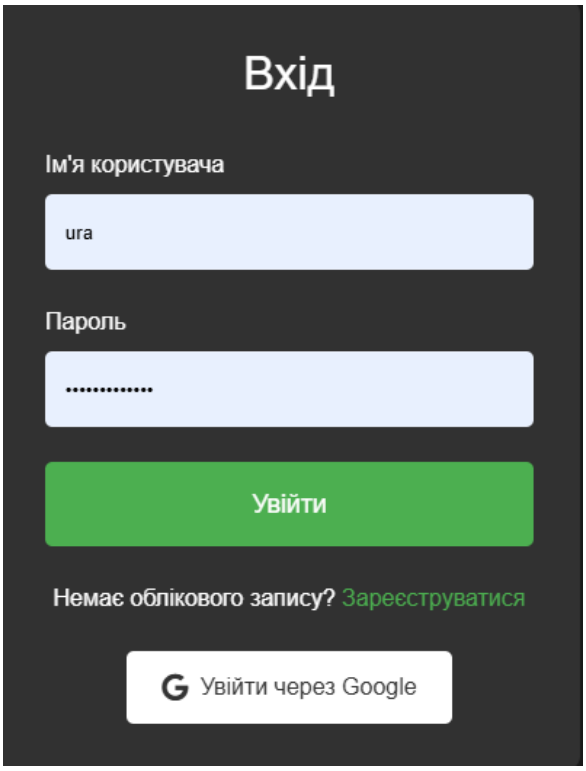


Рисунок 1.20. Тестування авторизації на сайті

Важливою частиною тестування стало оцінювання валідації форми – як на клієнтському, так і на серверному боці. Було встановлено, що валідація працює відповідно до заданих правил: форма реагує на порожні поля, повідомляє про помилки вводу й надає зрозумілий зворотний зв'язок для користувача. Повідомлення про помилки відображаються без перезавантаження сторінки, що сприяє підвищенню зручності використання інтерфейсу.

Крім того, проводилась перевірка збереження сесії користувача після авторизації: здійснювався перехід на захищені сторінки, перевірялась коректність виходу з системи, а також автоматичне перенаправлення неавторизованих користувачів при спробі доступу до ресурсів, які вимагають автентифікації.

Окрема увага приділялась стійкості до неправильних запитів – зокрема, перевірялась реакція системи на введення SQL-подібних конструкцій у поля вводу, а також відповідність повідомлень про помилки сучасним стандартам безпеки (скажімо, уникнення виводу повної інформації про типи помилок сервера).

Результати тестування показали, що реалізована система автентифікації та реєстрації відповідає очікуваним функціональним вимогам, забезпечує коректну обробку основних сценаріїв використання й демонструє стабільну роботу за умов стандартного та некоректного вводу даних.

1.4.2 Тестування запитів до зовнішнього API Google Books

Одним із ключових функціональних елементів реалізоаного веб-застосунку є механізм взаємодії з відкритим API Google Books, який забезпечує доступ до бази даних книжкових ресурсів та дозволяє реалізувати пошук літератури за різними критеріями. На відміну від варіанту реалізації з використанням окремого мікросервісу для обробки запитів. Наприклад, у розробленому проєкті вся логіка пошуку та обробки книжкової інформації інтегрована безпосередньо в основний модуль Django-застосунку. Це передбачає пряму реалізацію HTTP-запитів до стороннього API з подальшим аналізом і трансформацією отриманих даних та спрощує архітектуру системи, оскільки усуває потребу в додатковому шарі взаємодії між мікросервісами. Однак це також може призвести до зниження масштабованості [10].

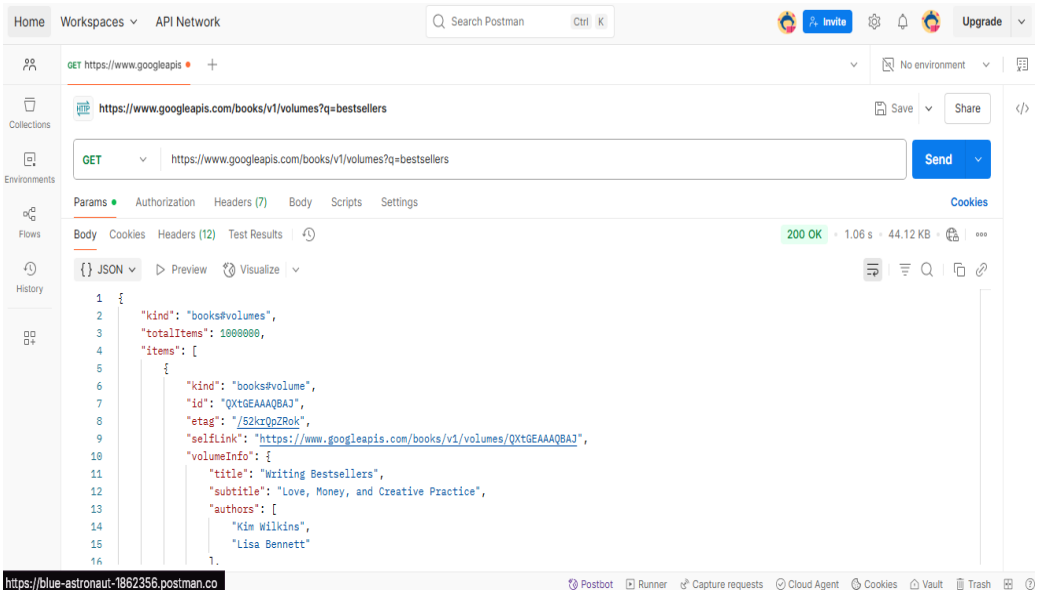


Рисунок 1.21. Тестування запитів до Google Books API

Процес тестування взаємодії із зовнішнім API мав на меті перевірити стабільність роботи функціональності пошуку, коректність формування запитів, адекватність обробки відповідей та наявність механізмів обробки помилок у нестандартних ситуаціях. Було реалізоано низку ручних та напівавтоматизованих сценаріїв, які моделювали типові та нетипові дії кінцевого користувача, включно з тестами на межові умови, помилки запиту, нестабільність мережевого підключення та валідацію структури відповіді.

Основна частина перевірок стосувалася коректного формування запитів на основі вхідних параметрів, таких як ключові слова, ім’я автора або назва книжки. При цьому система формує GET-запити до Google Books API з урахуванням необхідних параметрів (наприклад, q=search+terms, maxResults, startIndex) і очікує у відповідь стандартизований JSON-об’єкт, що містить базову інформацію про книги. Було підтверджено, що серверна частина застосунку коректно інтерпретує вхідні дані з форми пошуку, екранує спеціальні символи, виключає потенційно небезпечні введення та формує запити у відповідності до специфікацій API.

Одним із критично важливих аспектів стало тестування обробки відповідей, що надходять із боку API. Отриманий JSON-об'єкт проходить перевірку на наявність обов’язкових полів, таких як title, authors, description, publishedDate, imageLinks.thumbnail, industryIdentifiers, та ін. Система перевіряє структуру відповіді на предмет відповідності очікуваним шаблонам, після чого дані

форматуються і передаються до шаблону для рендерингу карток книг на стороні користувача. У разі виявлення відсутності критичних полів (наприклад, відсутній автор або обкладинка) реалізовані механізми graceful degradation – відповідний блок замінюється повідомленням «дані відсутні», що дозволяє зберегти цілісність відображення.

Тестування охопило також крайні випадки використання, включаючи надмірно довгі пошукові запити, використання недопустимих символів, запити порожнім рядком або з неіснуючими словами. У таких випадках система демонструє очікувану поведінку: або повідомляє користувача про відсутність результатів, або відображає повідомлення про помилкове введення. При цьому сервер не генерує винятків або збоїв, що свідчить про ефективну реалізацію механізмів обробки потенційно некоректних сценаріїв. Крім того, система успішно обробляє запити з різними наборами символів, забезпечуючи коректну роботу з багатомовними даними без втрати продуктивності.

Особливу увагу було приділено тестуванню поведінки застосунку у випадку проблем із мережею або недоступності API Google Books. Для цього моделювалися ситуації з навмисно уповільненим інтернет-з’єднанням, відключенням зовнішнього ресурсу або симуляцією тайм-ауту під час виконання запиту. Система в таких випадках не припиняла роботу або не повертала невизначену поведінку, а формувала стандартне повідомлення про недоступність стороннього сервісу з порадою повторити запит пізніше. Це стало можливим завдяки використанню конструкцій обробки винятків (try/except) під час надсилання HTTP-запитів за допомогою бібліотеки requests.

Було також перевірено, як система поводиться у разі частих послідовних запитів, що потенційно можуть перевищити ліміт API. У межах навантажувального тестування було згенеровано серію автоматизованих запитів, що імітують дії великої кількості користувачів. Система зберегла стабільність, а у разі перевищення квоти API демонструвала передбачувану реакцію – повідомлення про перевищення ліміту з відповідним інформуванням користувача.

Крім технічних перевірок, особливу увагу було приділено перевірці узгодженості між backend-частиною, яка здійснює запит до API, та frontend-шаблонами, що виводять отримані дані. Було підтверджено, що жодна зміна в структурі відповіді (наприклад, зміна імен ключів або відсутність певних елементів) не призводить до помилок відображення або краху сторінки. Це було досягнуто за рахунок вбудованих перевірок даних у шаблонах та використання конструкцій умовного рендерингу у Django-шаблонах.

Підсумовуючи, результати проведеного тестування свідчать про високу стійкість реалізоаного механізму інтеграції із зовнішнім сервісом Google Books. Веб-застосунок ефективно обробляє як стандартні, так і виняткові ситуації, забезпечуючи користувачеві надійний та інформативний інтерфейс для пошуку книжкової інформації. Незважаючи на відсутність окремого мікросервісу , централізована реалізація функціоналу в межах монолітного застосунку не знижує надійність або масштабованість системи, що підтверджується результатами тестування.

1.4.4 Тестування пошуку та обговорень

Пошук книжок і обговорення віртуальними учасниками є одними з ключових функціональних складових розробленої веб-системи. Вони створюють простір для інтелектуального взаємозв'язку між користувачами, а також забезпечують гнучкий доступ до потрібної інформації. У процесі тестування було сфокусовано увагу як на технічній надійності реалізації, так і на загальній якості користувацького досвіду.

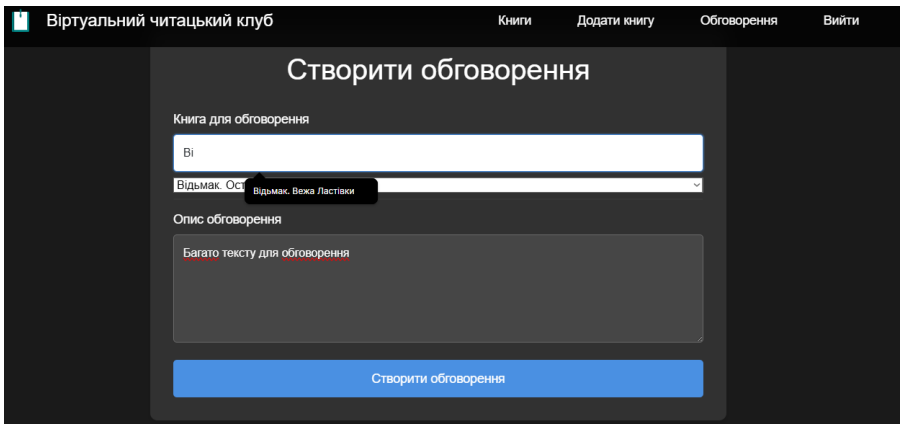


Рисунок 1.22. Тестування створення обговорень на сайті

Функція пошуку реалізована через інтеграцію між frontend-інтерфейсом, сервісом на Django та мікросервісом , який взаємодіє з Google Books API. У результаті цього ланцюга запитів формується динамічний перелік книжок, що відповідають ключовим словам користувача. Ретельне тестування охопило як типові сценарії використання, так і граничні ситуації. Зокрема, було перевірено поведінку системи при введенні неточних запитів, використанні символів різних мов і некоректних формулювань. У кожному випадку важливою метою було виявлення не лише факту отримання відповіді, але й її релевантності, швидкості обробки та зручності для кінцевого користувача.

З технічної точки зору, пошуковий інтерфейс показав стабільну роботу навіть при великій кількості повторюваних запитів у короткий проміжок часу. Це дозволяє зробити висновок, що логіка кешування, обробки помилок та обмеження запитів до зовнішнього API налаштована належним чином. Сторінка пошуку динамічно оновлюється, а її поведінка залишається передбачуваною, незалежно від обсягу вхідних даних.

Щодо блоку обговорень, то особливу увагу було приділено вивченню динаміки відображення коментарів та стабільності взаємодії з базою даних. Система дозволяє організовувати тематичні дискусії під конкретними книжками, формуючи виходить, що міні-форуми навколо кожної одиниці контенту. Було змодельовано низку сценаріїв, включно з інтенсивним потоком повідомлень, навмисним введенням некоректних даних, змінами в структурі DOM при оновленні, а також взаємодією декількох користувачів у режимі майже реального часу. Ключовим критерієм у тестуванні була не лише відповідність функціоналу очікуваному, але й емоційна реакція користувача: чи система викликає довіру, чи її логіка передбачувана, чи інтерфейс інтуїтивно зрозумілий.

Іншою важливою частиною перевірки стала оцінка стійкості системи до помилок. У межах тестування симулювались різноманітні порушення логіки з’єднання: відсутність інтернет-доступу, переривання з’єднання із сервером бази даних, затримки у відповіді зовнішнього API. Усі ці обставини дозволили виявити рівень надійності механізмів обробки виняткових ситуацій. Позитивним

результатом стало те, що навіть при неочікуваних збоїх користувач отримував зрозуміле повідомлення про проблему, а структура сторінки зберігалася без викривлення.

Отже, тестування пошуку та функціоналу обговорень підтвердило їхню повну працездатність, стійкість до непередбачених ситуацій і відповідність основним вимогам щодо інтерфейсної інтерактивності та ефективності взаємодії. Це свідчить про належний рівень реалізації системних зв'язків у межах мікросервісної архітектури й забезпечує якісну основу для масштабування платформи в майбутньому. Крім того, система забезпечує швидке відновлення роботи після тимчасових збоїв, мінімізуючи вплив на користувацький досвід. Результати тестування також підтвердили ефективність механізмів модерації контенту в дискусіях, що сприяє створенню безпечного та комфортного середовища для спілкування. Таким чином, платформа демонструє високий рівень готовності до реального використання та подальшого розвитку.

1.4.4 Тестування взаємодії внутрішніх модулів веб-системи

Після завершення етапу розробки всіх функціональних складових веб-застосунку виникає необхідність у всебічному тестуванні взаємодії між його окремими модулями. На відміну від мікросервісного підходу, де взаємодія забезпечується через окремі API-запити між незалежними сервісами, у межах монолітної архітектури всі функції інтегровані в єдину програмну структуру, яка працює в одному середовищі виконання. Це передбачає, що кожен логічний компонент застосунку повинен коректно взаємодіяти з іншими через чітко визначені внутрішні інтерфейси, виклики функцій, обробники подій та модулі обробки даних.

Веб-застосунок, створений у рамках дипломного проєкту, реалізований за допомогою фреймворку Django, що забезпечує високий рівень інтеграції між його компонентами. Основними функціональними блоками, які підлягали перевірці, були: модуль реєстрації та авторизації користувачів, у тому числі через Google OAuth; система пошуку книжок із використанням зовнішнього API Google Books; модуль обговорення літератури; механізми маршрутизації та виведення контенту

на інтерфейс; система обробки помилок та повідомлень; логіка створення та збереження користувацьких коментарів і обговорень; а також механізм кешування результатів пошуку для підвищення швидкодії.

Одним із ключових завдань було забезпечити безперебійну та логічно цілісну взаємодію між модулем пошуку книжок і модулем обговорення. Було критично важливо гарантувати, що кожен результат пошуку передається з усіма необхідними атрибутами (назва, автор, рік, обкладинка, опис), які потім можуть бути використані іншими компонентами – наприклад, для створення сторінки книги, збереження в локальну базу даних або відображення у формі для обговорення. Для цього було проведено інтеграційне тестування, під час якого імітувалися послідовні дії користувача: виконання пошукового запиту, перегляд результатів, вибір конкретної книги, перехід на її детальну сторінку та створення публікації в обговоренні. Усі ці етапи відбувалися в рамках єдиного процесу Django-застосунку, тому будь-яка помилка в логіці передачі даних між модулями могла призвести до збоїв у роботі.

Тестування показало, що дані, отримані із зовнішнього API, коректно передаються внутрішніми функціями системи, проходять обробку, серіалізацію та відображення в шаблонах без втрати інформації. Особливу увагу було приділено перевірці унікальних ідентифікаторів книг, які використовуються як ключ для зв’язування різних компонентів, зокрема під час створення обговорень. Усі перевірки підтвердили відсутність дублікатів, неправильних прив’язок або порушення цілісності зв’язків.

Крім стандартних сценаріїв взаємодії, було реалізоано низку тестів, які перевіряли поведінку системи у виняткових ситуаціях. Наприклад, було змодельовано ситуації, коли зовнішній API недоступний або повертає пошкоджені дані. У таких випадках система демонструвала належний рівень обробки помилок: виводилося повідомлення про недоступність функції, а неочікувані відповіді не призводили до помилок рендерингу чи краху програми. Це стало можливим завдяки використанню конструкцій обробки винятків (try/except), валідації

структури вхідних JSON-даних і обов'язкової перевірки ключових полів перед передачею інформації до інших компонентів.

Окремо тестувалася інтеграція модуля авторизації з іншими функціональними частинами системи. Зокрема, перевірялося, що доступ до обговорень можливий лише для автентифікованих користувачів, і що сесії правильно ідентифікуються в рамках єдиного застосунку. Було підтверджено, що після входу користувача всі необхідні дані зберігаються у відповідних сесіях і передаються до шаблонів без втрати контексту. Навіть при тривалому користуванні системою, під час переходу між сторінками або при оновленні сторінки, сесійні дані залишались актуальними.

Також було здійснено навантажувальне тестування, яке мало на меті перевірити, як система реагує на десятки одночасних запитів, зокрема до модуля пошуку, перегляду сторінок книг та створення коментарів. Для цього застосовувалися симуляції паралельної активності з боку кількох умовних користувачів. Система продемонструвала стабільність, швидке опрацювання запитів і правильну маршрутизацію відповідей. Випадки колізій або блокування потоків не спостерігалися.

Завдяки використанню єдиної ORM (Object-Relational Mapping) моделі Django, перевірка зв'язків між об'єктами (користувачами, книгами, коментарями) виявилася зручною та прозорою. Було підтверджено, що при створенні обговорення воно автоматично прив'язується до потрібної книги, а вся інформація про автора, час публікації та вміст зберігається відповідно до вимог бази даних.

```
.C:\Diplom\diplom_project\bookclub\core\views.py:207: UnorderedObjectListWarning: Pagination may yield incons
stent results with an unordered object_list: <class 'core.models.Book'> QuerySet.
  paginator = Paginator(books, 12)
.....
----------------------------------------------------------------------
Ran 7 tests in 10.576s

OK
Destroying test database for alias 'default'...
PS C:\Diplom\diplom_project>
```

Рисунок 1.23. Результати тестування взаємодій внутрішніх модулів у проекті

В результаті тестування було розроблено 7 тестів для перевірки взаємодії

внутрішніх модулів у проекті. Як це видно із рисунка 1.21 всі тести було пройдено. Кожен тест не виявив ніяких помилок у взаємодії внутрішніх модулів в проекті.

У підсумку проведене тестування довело, що монолітна архітектура веб-застосунку забезпечує високий рівень цілісності, надійності та узгодженості роботи всіх модулів. Незважаючи на те, що система не розділена на окремі сервіси, кожен логічний компонент працює автономно в межах чітко визначеної структури, а загальна інтеграція між модулями здійснюється надійно, ефективно та з дотриманням сучасних принципів побудови веб-застосунків. Отже, реалізована архітектура повністю задовольняє функціональні вимоги, забезпечує масштабованість і відповідає очікуванням кінцевих користувачів.

1.4.5 Перевірка роботи авторизації через Google

Інтеграція сторонньої авторизації, зокрема через Google OAuth 2.0, стала важливою складовою функціоналу онлайн простір. Такий підхід не лише покращує зручність доступу для користувачів, а й підвищує рівень безпеки, оскільки передбачає делегування автентифікації перевіреним сервісам. З огляду на критичну роль цього механізму, було проведено ретельне мануальне тестування всіх етапів взаємодії з Google API.

На першому етапі перевірялася коректність перенаправлення користувача до офіційного інтерфейсу Google після натискання на відповідну кнопку входу. Було підтверджено, що посилання генерується динамічно, включає всі необхідні параметри – client_id, redirect_uri, scope та response_type, – і відповідає вимогам Google OAuth 2.0.

Після успішного входу в акаунт Google тестувався механізм обробки токена авторизації. Серверна частина приймала авторизаційний код, обмінювала його на access token через захищений HTTPS-запит до сервера авторизації Google і отримувала профіль користувача (ім’я, електронну адресу, унікальний ідентифікатор). Особлива увага приділялась валідації отриманих даних та запобіганню помилкам у разі, якщо деякі поля виявлялись порожніми або неочікуваними.

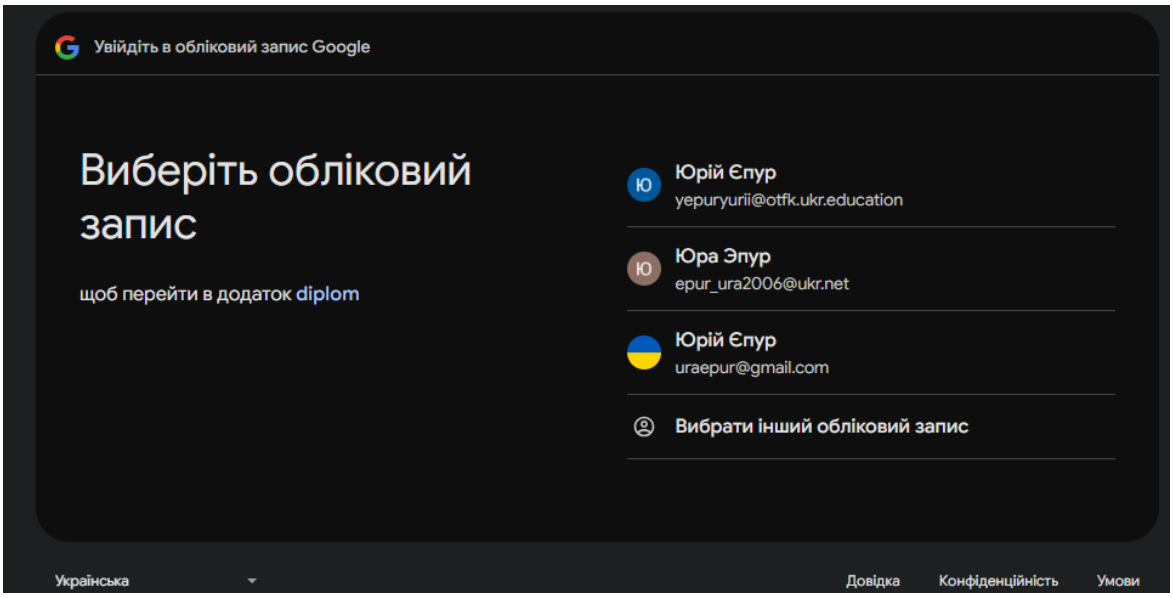


Рисунок 1.24. Тестування авторизації через Google

Було перевірено сценарії первинного входу користувача, коли акаунт ще не існує в базі даних. У таких випадках створювався новий запис з автоматично заповненими полями профілю, а користувач перенаправлявся на головну сторінку. Повторний вхід призводив до коректного розпізнавання існуючого акаунта, без дублювання записів, що підтвердило наявність логіки унікальної перевірки за email-адресою.

Тестувались також помилкові або нестандартні сценарії. Зокрема, вручну моделювалась ситуація з використанням простроченого access token, спроби зміни redirect URI та навмисне переривання процесу входу. У всіх випадках система реагувала передбачувано – або блокуючи небезпечні запити, або повертаючи користувачу зрозуміле повідомлення про помилку.

Особливий акцент був зроблений на безпечності реалізації – токени не зберігались у відкритому вигляді, а дані користувача проходили серіалізацію перед збереженням. Тестування підтвердило, що весь процес автентифікації відповідає стандартам безпеки та забезпечує захист персональних даних.

Загалом результати тестування засвідчили стабільну та передбачувану роботу авторизації через Google. Реалізований механізм надає користувачам швидкий доступ до платформи, не потребуючи створення нового пароля, що позитивно впливає на зручність використання системи та сприяє залученню нових учасників.

2 ЕКОНОМІЧНИЙ РОЗДІЛ

В дипломному проекты було розроблено веб-систему, яка надавала змогу користувачам спілкуватися між собою на тему своїх улюблених книг, обговорювати їх, та рекомундувати книги один одному, також можна знайти книгу для себе по зручному пошуку книг із улюбленими жанрами чи від улюбленого автора.

Даний проект був розроблений з урахуванням найновіших технологічних рішень і має зручний інтерфейс для користувача.

При розгляді оцінки ефективності розробленого веб-системи слід виходити з того, що залежно від характеру ефекту, що досягається, можуть бути визначені наступні види ефективності сайту: функціональна та соціальна ефективність.

У дипломному проєкті веб-система, що розроблялася, не є комерційною та не приноситиме доходу, тому оцінюються лише витрати на її розробку, а також функціональну та соціальну ефективність веб-системи.

Загальні витрати (Вз) на створення сайту складаються з декількох параметрів:

$$Вз = Вр + Вв + Ве \quad (2.1)$$

де:

- Вр – витрати на розробку сайту;
- Вв – витрати на впровадження сайту;
- Ве – витрати на експлуатацію сайту.

Витрати на розробку сайту (Вр) є одноразовими та складаються з вартості наступних видів робіт зі створення сайту:

1. Дизайн сайту: розробка макетів для головної та внутрішніх сторінок сайту; створення фірмового стилю і логотипу.
2. Розробка сторінок-шаблонів
3. Наповнення сайту інформацією: наповнення та форматування web-сторінок; обробка малюнків для публікації на web-сторінках, верстка(переклад в HTML-формат) web-сторінок

4. Програмна розробка сайту: створення програмного коду сайту, програмування динамічних елементів (анімаційних елементів, флеш-заставок)

Для визначення витрат на розробку сайту (Вр) розраховуємо оплату працівників, безпосередньо притягнених до її виконання. Над реалізацією веб-застосунку працював дипломник.

Для визначення трудомісткості розробки сайту (Вр) складено план-графік по розробці веб-системи і тривалості виконання робіт. Розподіл робіт по етапах і видах виконавців наведено в таблиці 2.1.

Таблиця 2.1 План-графік по розробці веб-системи

№	Назва етапу	Час виконання (годин)	Посада виконавця
1	Визначання вимог до функціональності, архітектури сайту	20	Дипломник
2	Розробка дизайну інтерфейсу (UI)	25	Дипломник
3	Створення структури і зв’язків бази даних	15	Дипломник
4	Написання функціональності систему для веб-застосунку	90	Дипломник
ВСЬОГО:		150 годин	

Для оцінки складності створення веб-сайту (Вр) був складений план-графік робіт із розробки веб-системи та було визначено тривалість виконання окремих операцій. Розподіл задач по етапах та виконавцям подано у таблиці 2.2.

Таблиця 2.2. Витрати на заробітну плату

№	Персонал	Етап розробки	Кількість робочих годин	Погодинна ставка грн.	Заробітна плата грн.
1	Дипломник	Визначання вимог до функціональності, архітектури сайту	20	110 грн./год	2200 грн.
2	Дипломник	Розробка дизайну інтерфейсу (UI)	25	110 грн./год	2750 грн.

Продовження таблиці 2.2

№	Персонал	Етап розробки	Кількість робочих годин	Погодинна ставка грн.	Заробітна плата грн.
3	Дипломник	Створення структури і зв’язків бази даних	15	110 грн./год	1650 грн.
4	Дипломник	Написання функціональності систему для веб-застосунку	90	110 грн./год	9900 грн.
				ВСЬОГО:	Взп= 16500 грн.

До витрат на оплату праці включаються також податки, збори та інші обов'язкові платежі, що передбачені чинною системою оподаткування. Розмір єдиного соціального внеску становить 22% від нарахованої заробітної плати та обчислюється за такою формулою:

$$Вєсв = Взп \times 0,22 = 16500 \times 0,22 = 3630 \text{ грн.} \quad (2.2)$$

Загальні витрати (Вр) на розробку веб-сайту розраховуютьсяяксумавитрат на заробітну плату праці персоналу (Взп) та єдиного соціальноговнеску(Вєсв):

$$Вр = Взп + Вєсв = 16500 + 3630 = 20130 \text{ грн} \quad (2.3)$$

Витрати на впровадження сайту (Вв) складаються з двох складових:

1. Витрати на реєстрацію доменного імені .com.ua становить 615 грн(Вв1) дані взяті з сайту imena.ua.

2. Реєстрація в пошукових системах, таких як Google через інструмент Google Search Console, є абсолютно безкоштовними.

$$Вв = Вв1 + Вв2 = 595 \text{ грн.} \quad (2.4)$$

Витрати на утримання веб-ресурсу (Ве) охоплюють кошти на підтримку сайту в активному стані та плату за поновлення доменного імені на річний термін. Я планую бюджет на хостинг на рік, який також покриває витрати на конфігурацію параметрів хостингового серверу, гарантування щомісячного захисту веб-системи та створення резервних копій. Окремо враховуються витрати на технічне обслуговування, включаючи оновлення програмного забезпечення і контроль безпеки. У таблиці 2.3 відображаються постійні витрати, що являють собою суму

витрат на створення та підтримку сайту протягом року, що надає чітке уявлення про необхідні ресурси для стабільної роботи веб-платформи.

Таблиця 2.3. Постійні витрати

№	**Стаття витрат**	**Вартість за рік, грн.**
1	Хостинг на рік	6400 грн
	ВСЬОГО:	$В_е$ = 6400

Загальні витрати (Вз) на розробку, впровадження та експлуатацію веб-сайту розраховуються за наступною формулою:

Вз = Вр+ (Вв+ Ве) = 20130+ (595 + 6400) = 27125 грн. (2.5)

Функціональна активність.

Функціональна ефективність інтерактивного веб-порталу віртуального клубу читачів полягає в забезпеченні повноти, точності та зручної доступності інформації для користувачів. Портал дозволяє зберігати, оновлювати та отримувати актуальні дані про книги, авторів, жанри, відгуки, рейтинги, а також інформацію про користувачів і їхні інтереси. Це гарантує, що кожен учасник клубу матиме швидкий доступ до необхідної інформації в будь-який час.

Система значно спрощує процеси пошуку, додавання книг до особистої бібліотеки, участі в обговореннях і формування персоналізованих рекомендацій. Користувач може легко зареєструватися, створити свій профіль і налаштувати вподобання, що автоматично впливає на пропозиції книг та контенту.

Вся інформація зберігається у централізованій базі даних, що дозволяє уникнути дублювання та помилок, пов’язаних із введенням даних. Автоматизовані функції збирання відгуків і генерації списків рекомендованих книг дають змогу значно підвищити ефективність взаємодії користувачів із порталом.

Інтерактивний веб-портал суттєво підвищує продуктивність користувачів, оскільки багато операцій відбуваються автоматично, що зменшує час на пошук інформації та організацію спільнот. Портал також дозволяє швидко відстежувати нові публікації, популярні книги та оновлення в клубі, що сприяє більш ефективній організації читального процесу.

Отже, функціональна ефективність веб-порталу полягає у забезпеченні високого рівня організації даних, їх точності, доступності, а також значномуспрощенні та прискоренні процесів взаємодії користувачів із книжковим контентом і одне з одним.

Соціальна ефективність.

Веб-портал віртуального клубу читачів сприяє формуванню позитивного іміджу як інноваційного та сучасного цифрового середовища для обміну літературними інтересами. Використання сучасних технологій і автоматизація процесів обробки даних підвищують якість обслуговування користувачів та демонструють відкритість до новітніх інформаційних рішень.

Портал покращує взаємодію між читачами, авторськими спільнотами та адміністрацією ресурсу, сприяючи більшій комунікації та активній участі в житті клубу. Це підвищує рівень обізнаності, мотивації та задоволеності учасників, що, у свою чергу, стимулює регулярне відвідування порталу та активне залучення нових користувачів.

Забезпечення конфіденційності даних і безпеки інформації користувачів створює довіру до платформи, що є важливим фактором для формування стабільної і активної спільноти читачів.

Як ми бачимо, результатом впровадження веб-порталу є покращення комунікації, підвищення рівня задоволеності користувачів та зміцнення позитивного іміджу цифрового книжкового клубу як сучасного інструменту популяризації читання.

3 РОЗДІЛ ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

3.1 Основні положення

У цьому розділі розглядаються питання охорони праці та техніки безпеки які є важливими при розробці інтерактивного веб- порталу віртуального клубу читачів. Головна мета охорони праці це створення та підтримання безпечних і здорових умов для працівників які беруть участь у розробці та функціонуванні ІТ-проєкту.

3.2 Аналіз умов праці та гарантування безпеки при виконанні головних різновидів робіт на об'єкті дипломного проектування

3.2.1 Небезпечні та шкідливі виробничі чинники

Під час роботи над розробкою веб-порталу робітники можуть стикатися з різноманітними виробничими факторами, які можуть причинити шкоду їхньому здоров'ю. Серед таких основних факторів є тривале сидіння за комп'ютером що може призводитти до перевтомлення опорно-рухового апарату, в особливості шиї, спини та кінцівок. Постійне навантаженя очей через роботу з монітором викликає зорову тяжкість, сухість та подразнення очей. Шум від комп'ютера та клацаня миші з клавіатурою та некомфортний мікроклімат можуть спричиняти стрес і зниження працездатності робітника. Електромагнітне випромінювання від комп'ютерної техніки також може негативно впливати на загальне самопочуття.

Ці всі фактори разом потребують уваги для створення безпечних і комфортних умов роботі, що зможе допомогти зберегти здоров'я працівників і покращити продуктивність їх праці.

3.2.2 Заходи безпеки на робочому місці

Для створення безпечних умов праці та зниження негативного впливу виробничих факторів слід впроваджувати комплекс профілактичних заходів. По-перше, необхідно організувати освітлення робочих місць, яке поєднує природне та штучне світло, що забезпечує оптимальний зоровий комфорт і зменшує напруження очей. Важливо уникати відблисків і тіней на екранах комп'ютерів, що

підвищує ефективність роботи та знижує ризик виникнення головного болю. По-друге, використання ергономічних меблів дозволяє працівникам підтримувати правильну поставу, запобігаючи перевантаженню м’язів і суглобів, що особливо актуально при тривалій роботі за комп’ютером.

Регулярні перерви для відпочинку, а також спеціальні вправи для очей сприяють зменшенню втоми, покращенню кровообігу і підтримці загального здоров’я. Крім того, слід проводити інструктажі з правильного використання обладнання та заходів безпеки, що допомагає уникнути травматичних випадків. Для додаткового захисту працівників рекомендується забезпечити їх засобами індивідуального захисту, зокрема спеціальними окулярами, які знижують вплив синього світла від моніторів. Такий комплексний підхід гарантує покращення умов праці, підвищення продуктивності та збереження здоров’я персоналу.

3.3 Гігієнічні вимоги до виробництва

3.3.1 Приміщення

Приміщення для роботи над веб-порталом має відповідати санітарним нормам і забезпечувати комфортні умови для працівників. Кожне робоче місце повинно мати площу не менше 6 м² та об’єм повітря не менше 20 м³, що запобігає перенаселеності і створює оптимальний простір для роботи. Важливо, щоб приміщення було обладнане системою вентиляції для постійного оновлення повітря та підтримувало комфортний температурний режим за допомогою кондиціонування. Такі умови сприяють збереженню здоров’я працівників і підвищенню їх продуктивності.

3.3.2 Освітлення

Освітлення робочих місць має відповідати санітарним нормам і забезпечувати комфортні умови для зорової роботи. Рекомендується використовувати комбіноване освітлення — природне у поєднанні зі штучним, що дозволяє підтримувати достатню яскравість упродовж усього робочого дня незалежно від зовнішніх умов. Інтенсивність освітлення не повинна бути меншою за 500 люкс, що допомагає зменшити напруження очей і запобігти головним болям.

Правильне освітлення сприяє підвищенню продуктивності та збереженню здоров’я працівників.

3.3.3 Шум

Джерела шуму можуть включати офісне обладнання, комп'ютери, системи вентиляції, зовнішній транспорт або голосні розмови співробітників. Рівень шуму на робочому місці повинен бути контрольованим і не перевищувати 50 дБ. Такий рівень забезпечує комфортну акустичну атмосферу, що сприяє зосередженню працівників та знижує ризик виникнення стресу і втоми. Надмірний шум може негативно впливати на працездатність та загальний стан здоров’я, тому важливо застосовувати шумоізоляційні матеріали і розташовувати робочі місця з урахуванням акустичних вимог. Методи захисту від шуму передбачають використання шумоізоляційних матеріалів, звукопоглинаючих панелей, акустичних перегородок, а також правильне планування простору для мінімізації шумового впливу.

3.3.4 Мікроклімат

Комфортний мікроклімат є важливою складовою здорових умов праці. Температура повітря в робочому приміщенні підтримується в межах 18-22°C, що забезпечує оптимальні умови для концентрації та зниження втоми. Відносна вологість повітря має бути у діапазоні 40-60%, що запобігає пересушуванню слизових оболонок і шкіри, а також створює сприятливі умови для дихальної системи. Забезпечення таких параметрів мікроклімату сприяє збереженню здоров’я працівників та підвищенню їх продуктивності. Це досягається за допомогою систем опалення, кондиціонування, вентиляції, а також використання зволожувачів повітря.

3.3.5 Організація робочого місця

Для забезпечення комфортної та безпечної роботи важливо правильно організувати робоче місце. Ергономічні столи та стільці дозволяють підтримувати правильну поставу, що знижує навантаження на м’язи та хребет. Розташування монітора на рівні очей допомагає уникнути напруження шийного відділу і сприяє

збереженню здоров’я працівника. Крім того, варто передбачити достатній простір для руху та розміщення необхідного обладнання, що підвищує продуктивність та комфорт під час роботи.

3.3.6 Електробезпека

Забезпечення електробезпеки є обов’язковою умовою для безпечної роботи в офісному середовищі. Важливо використовувати справне електрообладнання, яке відповідає встановленим стандартам і проходить регулярний технічний огляд. Необхідно також регулярно перевіряти стан електромережі та заземлення для запобігання коротким замиканням, перенапруженням і можливим аварійним ситуаціям. Дотримання цих заходів мінімізує ризики травматизму та пошкодження обладнання.

3.4 Пожежна безпека

Пожежна безпека є важливим аспектом організації робочого простору, особливо в офісах із використанням великої кількості електронного обладнання. Для забезпечення пожежної безпеки передбачається розміщення портативних вогнегасників, які відповідають вимогам безпеки та розраховані на гасіння різних типів пожеж. У приміщенні з ПК доцільно передбачити вогнегасник вуглекислотного типу, оскільки він ефективно гасить електропристрої без ризику пошкодження обладнання або залишків на його поверхні.

Також важливим елементом пожежної безпеки є встановлення автоматичних систем пожежного оповіщення та сигналізації. Ці системи можуть включати димові детектори, теплові датчики та інші сенсори, які оперативно виявляють загрозу та повідомляють про неї персонал і службу безпеки. Автоматичне сповіщення дозволяє швидко реагувати на пожежу, що значно знижує ймовірність масштабних пошкоджень і створює безпечніші умови для евакуації працівників.

Передбачено навчання персоналу і їх практичні тренування, що допомагає працівникам швидко та ефективно діяти у разі пожежі, зменшуючи ризики для життя і здоров’я.

3.5 Висновки до розділу охорони праці

У ході аналізу умов праці при розробці інтерактивного веб-порталу віртуального клубу читачів було виявлено низку факторів, які можуть негативно впливати на здоров’я працівників та знижувати їх продуктивність. Основними з них є недостатнє освітлення, неправильна організація робочих місць, підвищений рівень шуму та електромагнітне випромінювання від комп’ютерної техніки. Для мінімізації цих ризиків запропоновано комплекс заходів, що включають ергономічне облаштування робочих місць, оптимізацію мікроклімату, забезпечення комбінованого освітлення та організацію регулярних перерв для відпочинку.

Реалізація цих заходів сприяє створенню безпечних та комфортних умов праці, що забезпечує збереження здоров’я працівників, підвищення їх концентрації та загальної продуктивності. Забезпечення належної електробезпеки та пожежної безпеки також є важливими складовими ефективної системи охорони праці, що спрямована на запобігання нещасним випадкам та аварійним ситуаціям. Впровадження цих рекомендацій є необхідною умовою для стабільної та безпечної роботи колективу розробників.

ВИСНОВКИ

У дипломному проєкті створено повнофункціональний веб-портал віртуального клубу читачів, який дозволяє користувачам знаходити книги, ділитись враженнями, брати участь в обговореннях та отримувати рекомендації на основі взаємодії із системою.

Основну увагу було приділено реалізації Back-End частини системи за допомогою фреймворку Django. Забезпечено реєстрацію та авторизацію користувачів, у тому числі через облікові записи Google, що підвищує зручність користування сайтом. Роутинги, пов’язані з розміщенням коментарів, створенням тем обговорень та іншими функціями, проходять обробку через відповідні перевірки, що гарантує стабільну роботу та безпеку системи.

Усі функціональні компоненти веб-порталу — від отримання даних через Google Books API до створення рекомендацій — об'єднано у цілісну монолітну архітектуру. Для зберігання інформації використано PostgreSQL. Статичні сторінки реалізовані з використанням HTML, CSS та JavaScript, що забезпечує адаптивність та сучасний вигляд інтерфейсу. Розгортання та тестування здійснювалося в середовищі Windows, з використанням інструментів розробника PyCharm. Docker використовувався переважно на етапах тестування та перевірки працездатності, без обов’язкового включення його конфігурацій у фінальну документацію.

У пояснювальній записці докладно розглянуто всі питання, що відповідають технічному завданню дипломного проєкту. Проведено огляд предметної області, розглянуто існуючі аналогічні платформи, зокрема Goodreads, LibraryThing, ReadRate, Livelib, і наведено порівняльну характеристику. Пояснено обґрунтування вибору архітектури застосунку, описано реалізацію основних функціональних компонентів, таких як рекомендації книг, система обговорень, пошук та інтеграція з зовнішніми сервісами. Проведено етапи тестування функціоналу, зокрема перевірку правильності обробки запитів, стабільності роботи сторінок, зручності інтерфейсу.

ПЕРЕЛІК ВИКОРИСТАНИХ ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. Подоба В. Веб-розробка з Python та Django для початківців. – 2016.
2. Томка Ю. Я., Талах М. В., Ушенко Ю. О. Python та Django Full Stack веб-розробка. – 2022.
3. Goodreads [Електронний ресурс]. – Режим доступу: https://www.goodreads.com – Дата звернення: 15.05.2025.
4. LibraryThing [Електронний ресурс]. – Режим доступу: https://www.librarything.com – Дата звернення: 15.05.2025.
5. Bookster [Електронний ресурс]. – Режим доступу: https://bookster.com – Дата звернення: 16.05.2025.
6. Книгоманія [Електронний ресурс]. – Режим доступу: https://knygomania.com – Дата звернення: 16.05.2025.
7. Рейтинг мов програмування 2025 [Електронний ресурс]. – Режим доступу: https://dou.ua/lenta/articles/language-rating-2025/ – Дата звернення: 17.05.2025.
8. Офіційна документація Django [Електронний ресурс]. – Режим доступу: https://docs.djangoproject.com/en/5.2/ – Дата звернення: 20.05.2025.
9. Using OAuth 2.0 to Access Google APIs [Електронний ресурс]. – Режим доступу: https://developers.google.com/identity/protocols/oauth2 – Дата звернення: 22.05.2025.
10. Офіційна документація Google Books API [Електронний ресурс]. – Режим доступу: https://developers.google.com/books/docs/v1/reference/bookshelves – Дата звернення: 23.05.2025.
11. Офіційна документація Python [Електронний ресурс]. – Режим доступу: https://docs.python.org/3/ – Дата звернення: 24.05.2025.
12. MVC – Brander [Електронний ресурс]. – Режим доступу: https://brander.ua/technologies/mvc – Дата звернення: 26.05.2025.
13. Офіційна документація PostgreSQL [Електронний ресурс]. – Режим доступу: https://www.postgresql.org/docs/ – Дата звернення: 27.05.2025.
14. Google Cloud [Електронний ресурс]. – Режим доступу: https://cloud.google.com – Дата звернення: 01.06.2025.

ДОДАТОК А. Програмний код MVC-архітектури

```
// urls.py

from django.urls import path, re_path
from . import views

urlpatterns = [
    path('', views.index, name='index'),
    path('add-book/', views.add_book, name='add_book'),
    path('register/', views.register_view, name='register'),
    path('login/', views.login_view, name='login'),
    path('logout/', views.logout_view, name='logout'),
    path('search/', views.book_search, name='book_search'),
    re_path(r'^book/(?P<book_id>[^/]+)/$', views.book_detail, name='book_detail'),
    path('fetch-books/', views.fetch_and_save_books, name='fetch_books'),
    path('update-books/', views.update_books, name='update_books'),
    path('recommendations/', views.book_recommendations, name='book_recommendations'),
    path('discussions/', views.discussions_list, name='discussions_list'),
    path('discussions/<int:discussion_id>/', views.discussion_detail,
name='discussion_detail'),
    path('discussions/create/', views.create_discussion, name='create_discussion'),
    path('book-search/', views.book_search, name='book_search'),
    path('book-search-api/', views.book_search_api, name='book_search_api'),
]

//models.py

from django.db import models
from django.contrib.auth import get_user_model
from django.utils import timezone

User = get_user_model()


class Book(models.Model):
    title = models.CharField(max_length=200)
    author = models.CharField(max_length=200)
    description = models.TextField()
    genre = models.CharField(max_length=100)
    thumbnail = models.URLField()
    google_books_id = models.CharField(max_length=100, unique=True, null=True,
blank=True)
    preview_link = models.URLField(null=True, blank=True)
    published_year = models.CharField(max_length=4, null=True, blank=True)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    def __str__(self):
        return self.title

    @property
    def average_rating(self):
        return round(self.ratings.aggregate(models.Avg('rating'))['rating__avg'] or 0,
2)


class Discussion(models.Model):
    book = models.ForeignKey(Book, on_delete=models.CASCADE)
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    comment = models.TextField()
    posted_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
```

```
        return f'Обговорення для {self.book.title}'


class DiscussionComment(models.Model):
    discussion = models.ForeignKey(Discussion, on_delete=models.CASCADE,
related_name='comments')
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    text = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)

    class Meta:
        ordering = ['-created_at']

    def __str__(self):
        return f'Коментар від {self.user.username} до обговорення
{self.discussion.book.title}'


class Comment(models.Model):
    book = models.ForeignKey(Book, on_delete=models.CASCADE, related_name='comments')
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    text = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)

    class Meta:
        ordering = ['-created_at']

    def __str__(self):
        return f'Коментар від {self.user.username} до {self.book.title}'


class BookView(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE, related_name='book_views')
    book = models.ForeignKey(Book, on_delete=models.CASCADE, related_name='views')
    viewed_at = models.DateTimeField(default=timezone.now)

    class Meta:
        ordering = ['-viewed_at']
        # Добавляем уникальное ограничение, чтобы у пользователя была только одна запись
для каждой книги
        unique_together = ['user', 'book']

    def save(self, *args, **kwargs):
        # Обновляем время просмотра при повторном просмотре
        self.viewed_at = timezone.now()
        super().save(*args, **kwargs)


class BookRating(models.Model):
    book = models.ForeignKey(Book, on_delete=models.CASCADE, related_name='ratings')
    user = models.ForeignKey(User, on_delete=models.CASCADE,
related_name='book_ratings')
    rating = models.PositiveSmallIntegerField(choices=[(i, str(i)) for i in range(1,
6)])  # 1-5 звёзд
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    class Meta:
        unique_together = ('book', 'user')

    def __str__(self):
        return f"{self.user.username} — {self.book.title}: {self.rating}★"
```

```
//views.py

def index(request):
    """Головна сторінка: спочатку популярні книги з Google Books API, потім локальні"""
    books_api = []
    try:
        url =
'https://www.googleapis.com/books/v1/volumes?q=bestsellers&maxResults=10&orderBy=relevan
ce'
        response = requests.get(url)
        data = response.json()
        if 'items' in data:
            for item in data['items']:
                volume_info = item.get('volumeInfo', {})
                image_links = volume_info.get('imageLinks', {})
                books_api.append({
                    'title': volume_info.get('title', 'Невідома назва'),
                    'author': ', '.join(volume_info.get('authors', ['Невідомий
автор'])),
                    'description': volume_info.get('description', 'Опис відсутній'),
                    'published_year': volume_info.get('publishedDate', '')[:4],
                    'genre': ', '.join(volume_info.get('categories', ['Невідомий
жанр'])),
                    'thumbnail': image_links.get('thumbnail',
'https://via.placeholder.com/300x450?text=No+Image'),
                    'preview_link': volume_info.get('previewLink', '#'),
                    'google_books_id': item.get('id')
                })
    except Exception as e:
        books_api = []

    # Якщо не вдалося отримати з API, показуємо локальні
    if not books_api:
        popular_books = Book.objects.all().order_by('-created_at')[:10]
        books_api = list(popular_books)

    return render(request, 'core/index.html', {'popular_books': books_api})


def add_book(request):
    """Сторінка додавання книг"""
    if request.method == 'POST':
        form = BookForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('index')  # або інший URL
    else:
        form = BookForm()

    return render(request, 'core/add_book.html', {'form': form})


def register_view(request):
    """Реєстрація"""
    if request.method == 'POST':
        form = RegisterForm(request.POST)
        if form.is_valid():
            user = form.save()
            login(request, user, backend='django.contrib.auth.backends.ModelBackend')
            return redirect('index')
    else:
        form = RegisterForm()
    return render(request, 'core/register.html', {'form': form})
```

```
def login_view(request):
    """Вхід"""
    if request.method == 'POST':
        form = AuthenticationForm(request, data=request.POST)
        if form.is_valid():
            user = form.get_user()
            login(request, user)
            return redirect('index')
    else:
        form = AuthenticationForm()
    return render(request, 'core/login.html', {'form': form})


def logout_view(request):
    """Вихід з облікового запису"""
    logout(request)
    return redirect('index')
def book_search(request):
    """Пошук книг"""
    query = request.GET.get('q', '')
    genre = request.GET.get('genre', '')
    page = request.GET.get('page', 1)

    # Спочатку шукаємо книги в локальній базі
    books = Book.objects.all()

    if query:
        books = books.filter(
            Q(title__icontains=query) |
            Q(author__icontains=query) |
            Q(description__icontains=query)
        )

    if genre:
        books = books.filter(genre=genre)

    # Якщо локальний пошук не дав результатів, шукаємо через Google Books API
    if not books.exists() and query:
        try:
            # Формуємо URL для запиту до Google Books API з фільтром по українській та
англійській мовах
            api_url =
f"https://www.googleapis.com/books/v1/volumes?q={query}&maxResults=40&langRestrict=uk,en
"
            if genre:
                api_url += f"&subject:{genre}"

            response = requests.get(api_url)
            data = response.json()

            if 'items' in data:
                for item in data['items']:
                    volume_info = item['volumeInfo']

                    # Перевіряємо мову книги
                    language = volume_info.get('language', '')
                    if language not in ['uk', 'en']:
                        continue

                    # Перевіряємо наявність російських букв у назві та авторі
                    title = volume_info.get('title', '').lower()
                    author = ', '.join(volume_info.get('authors', ['Невідомий
автор'])).lower(
```

```
                    # Список російських букв для перевірки
                    russian_chars = 'ёъыэ'
                    russian_words = ['россия', 'российский', 'русский', 'русь',
'москва', 'петербург']
                    # Перевіряємо наявність російських букв і слів
                    if (any(char in title for char in russian_chars) or
                        any(char in author for char in russian_chars) or
                        any(word in title for word in russian_words) or
                        any(word in author for word in russian_words)):
                        continue
                    # Перевіряємо, чи є вже така книга в базі
                    if not Book.objects.filter(google_books_id=item['id']).exists():
                        # Створюємо нову книгу
                        book = Book(
                            title=volume_info.get('title', ''),
                            author=', '.join(volume_info.get('authors', ['Невідомий
автор'])),
                            description=volume_info.get('description', ''),
                            genre=genre if genre else volume_info.get('categories',
['Невідомий жанр'])[0],
                            thumbnail=volume_info.get('imageLinks', {}).get('thumbnail',
''),
                            google_books_id=item['id'],
                            preview_link=volume_info.get('previewLink', '')
                        )
                        book.save()
                # Після додавання книг, робимо новий запит до локальної бази
                books = Book.objects.all()
                if query:
                    books = books.filter(
                        Q(title__icontains=query) |
                        Q(author__icontains=query) |
                        Q(description__icontains=query))
                if genre:
                    books = books.filter(genre=genre)
        except Exception as e:
            print(f"Помилка при отриманні з Google Books API: {e}")
    # Пагінація
    paginator = Paginator(books, 12)
    try:
        books = paginator.page(page)
    except PageNotAnInteger:
        books = paginator.page(1)
    except EmptyPage:
        books = paginator.page(paginator.num_pages)
    # Отримуємо список усіх жанрів для фільтра
    genres = Book.objects.values_list('genre', flat=True).distinct()
    context = {
        'books': books,
        'query': query,
        'genre': genre,
        'genres': genres,}

    return render(request, 'core/book_search.html', context)
```

ДОДАТОК Б. Слайди мультимедійної презентації

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

ДИПЛОМНИЙ ПРОЕКТ

КС.58.07.000.00 ДП

Єпура Юрія Олександровича

На тему:
Розробка інтерактивного веб-порталу віртуального клубу читачів

Спеціальність: 121 – «Комп'ютерна інженерія»
Освітньо-професійна програма: «Обслуговування комп'ютерних систем і мереж»

Стек технологій для розробки веб-системи

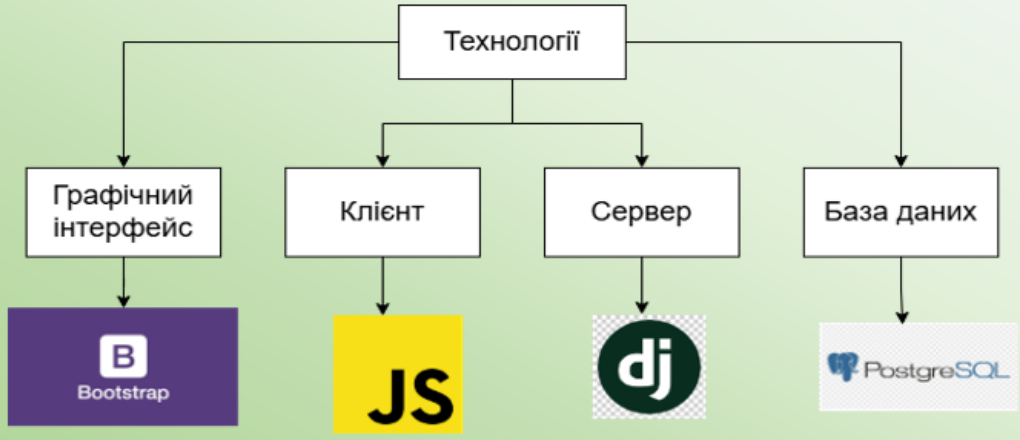
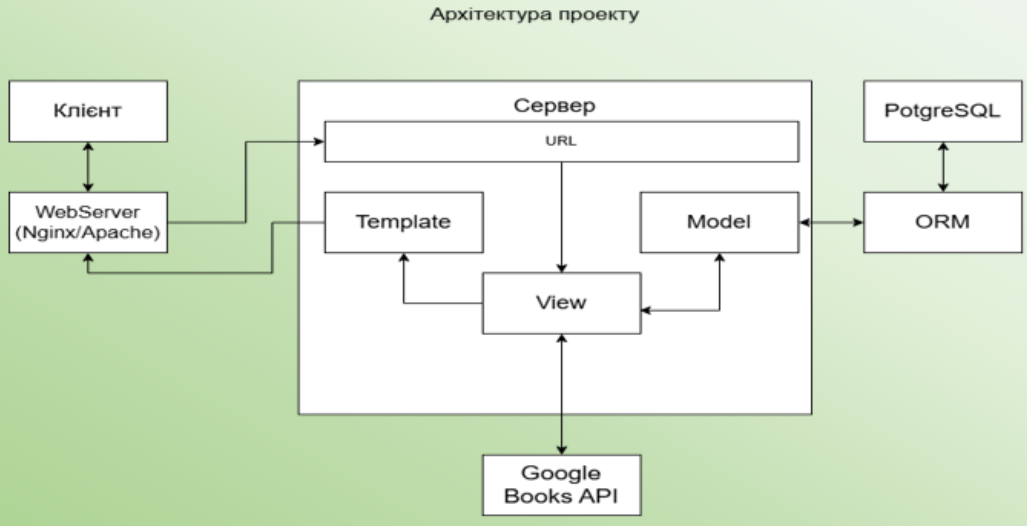


Схема архітектури проекту



Файлова схема проєкту

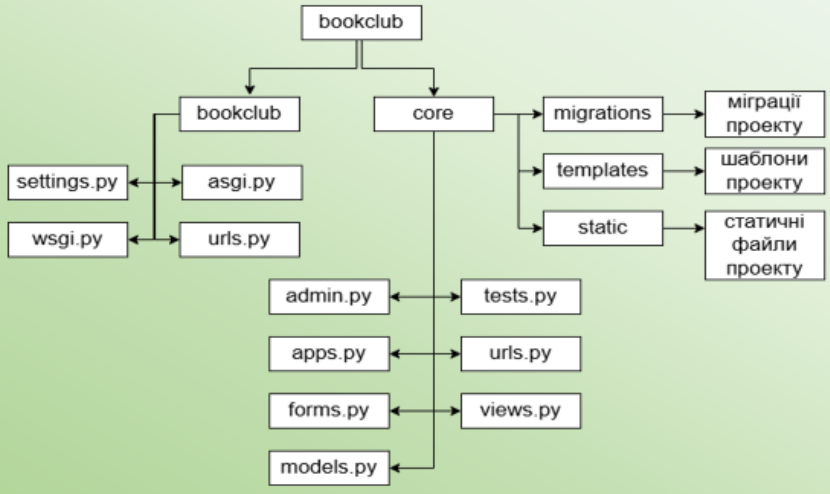
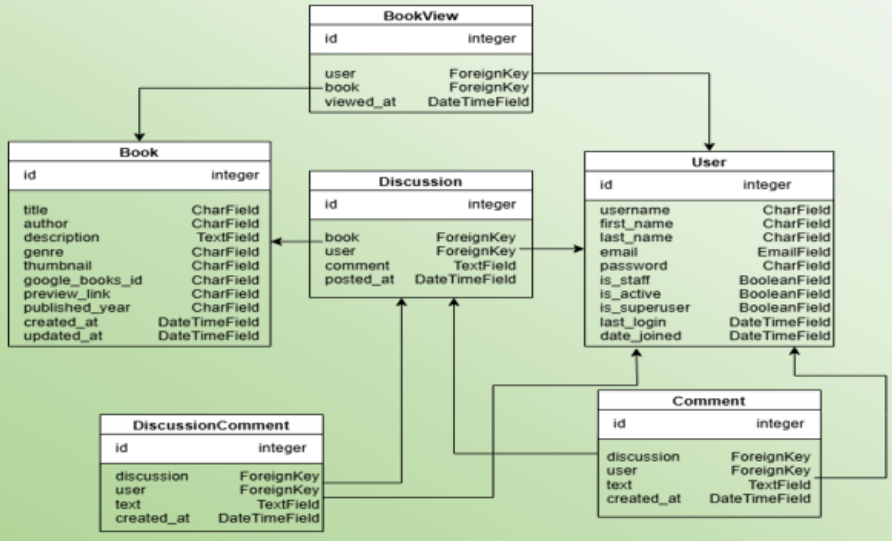
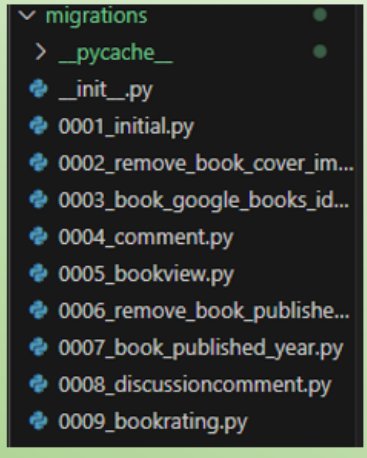


Схема бази даних веб-застосунку

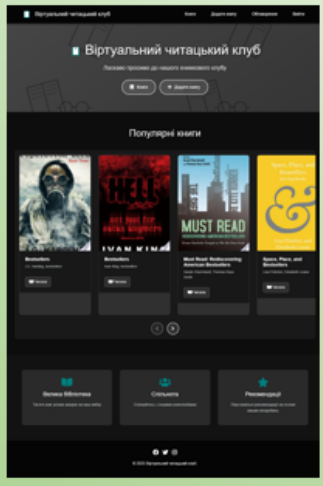


Створенні міграції



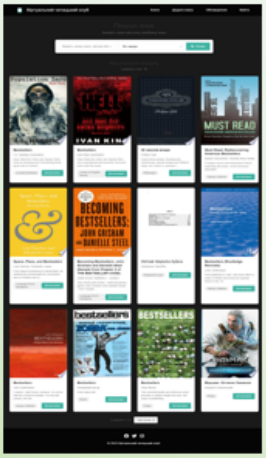

```
Migrations for 'core':
  core\migrations\0001_initial.py
    + Create model Book
    + Create model Comment
    + Create model Discussion
    + Create model DiscussionComment
    + Create model BookRating
    + Create model BookView
```

Скріншоти сторінок веб-застосунку

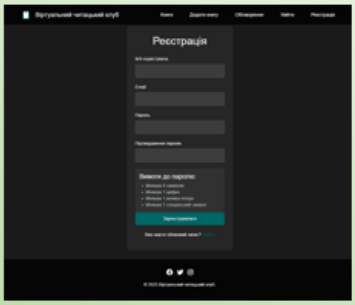


Головна сторінка

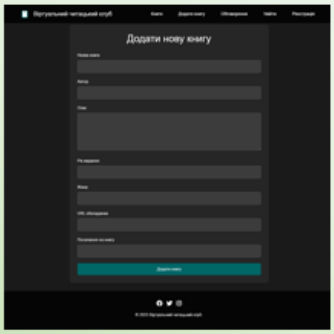
Сторінка авторизації



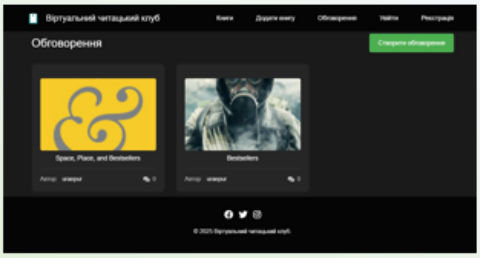
Сторінка пошуку книг



Сторінка реєстрації



Сторінка додання нових книг



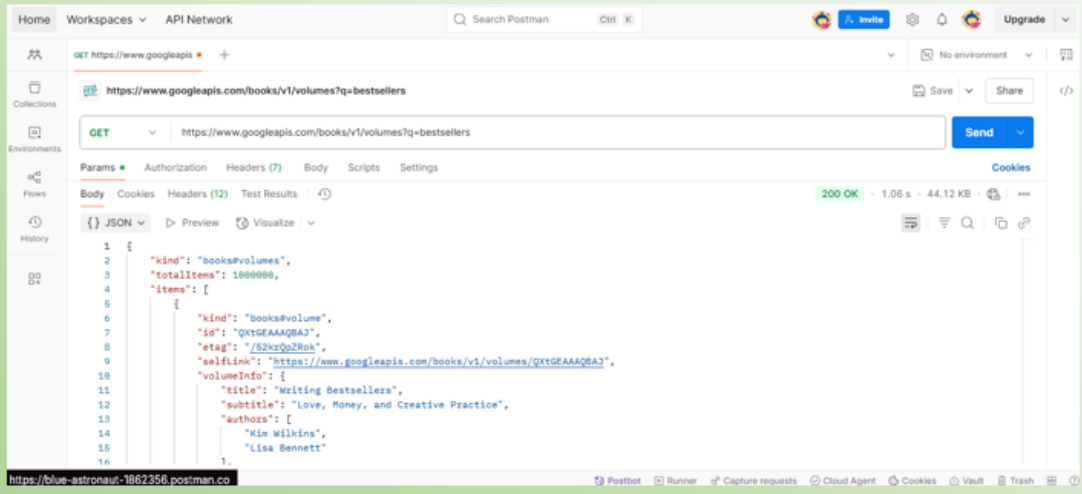
Сторінка з обговореннями книг

Створені маршрути веб-застосунку

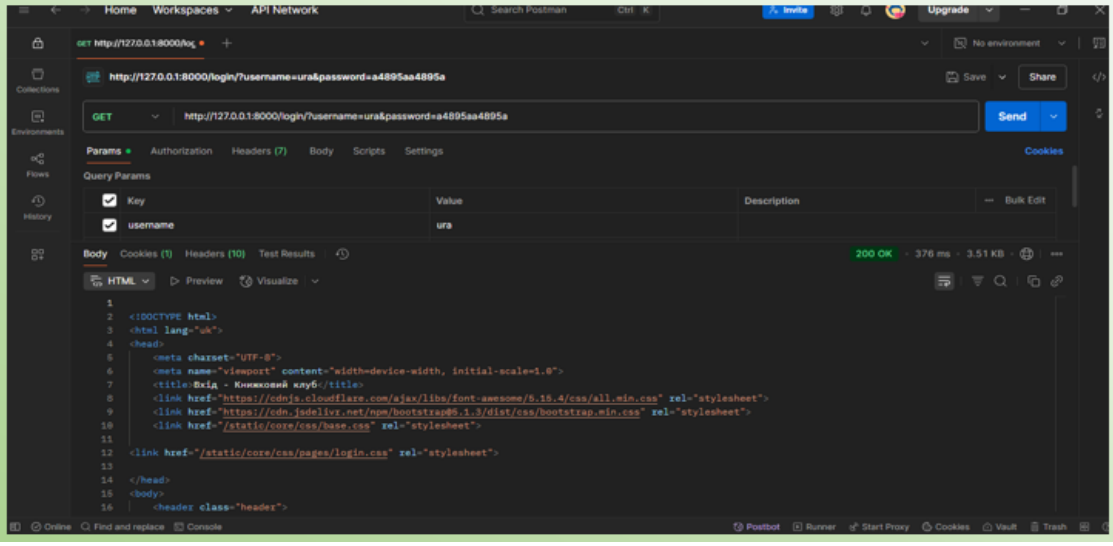
```python
from django.urls import path, re_path
from . import views

urlpatterns = [
    path('', views.index, name='index'),
    path('add-book/', views.add_book, name='add_book'),
    path('register/', views.register_view, name='register'),
    path('login/', views.login_view, name='login'),
    path('logout/', views.logout_view, name='logout'),
    path('search/', views.book_search, name='book_search'),
    re_path(r'^book/(?P<book_id>[^/]+)/$', views.book_detail, name='book_detail'),
    path('fetch-books/', views.fetch_and_save_books, name='fetch_books'),
    path('update-books/', views.update_books, name='update_books'),
    path('recommendations/', views.book_recommendations, name='book_recommendations'),
    path('discussions/', views.discussions_list, name='discussions_list'),
    path('discussions/<int:discussion_id>/', views.discussion_detail, name='discussion_detail'),
    path('discussions/create/', views.create_discussion, name='create_discussion'),
    path('book-search/', views.book_search, name='book_search'),
    path('book-search-api/', views.book_search_api, name='book_search_api'),
]
```

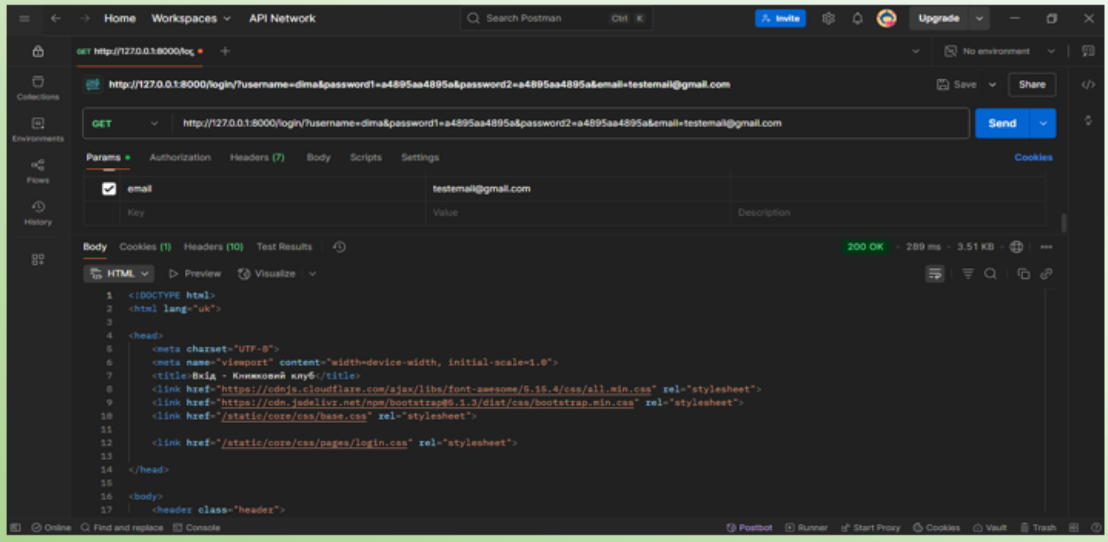
Тестування запитів до Google Books API



Тестування авторизації



Тестування реєстрації



Висновки

У дипломному проєкті створено повнофункціональний веб-портал віртуального клубу читачів на основі Django, що дозволяє користувачам шукати книги, обговорювати їх і отримувати персоналізовані рекомендації. Для реалізації було використано монолітну архітектуру з PostgreSQL та Google Books API, Front-End виконано з HTML, CSS та JavaScript. Проведено тестування функціоналу та інтерфейсу, що підтвердило стабільність і зручність системи.

Посилання на сайт

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

РЕЦЕНЗІЯ

на дипломний проект здобувача (здобувачки) освіти

відділення комп'ютерних систем

Єпура Юрія Олександровича

(прізвище, ім`я та по батькові)

Спеціальність *123 «Комп'ютерна інженерія»*

Освітня програма *«Обслуговування комп'ютерних систем і мереж»*

Керівник дипломного проекту (роботи) *Закроєв Юрій Михайлович*

(прізвище, ім`я та по батькові)

Тема дипломного проекту (роботи) *Розробка інтерактивного веб-порталу віртуального клубу читачів*

Обсяг розрахунково-пояснювальної записки *85* сторінок

Обсяг графічної (презентаційної) частини *14* аркушів (слайдів)

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЕКТУ (РОБОТИ)

а) заключення про ступінь відповідності виконаного дипломного проекту завданню

Представлений на рецензію дипломний проект відповідає затвердженій темі та виконаний відповідно технічному завданню. Дипломний проект присвячений темі веб розробки та складається з пояснювальної записки, додатку з програмним кодом та мультимедійної презентації, що містить приклади роботи програми.

б) характеристика виконання кожного розділу дипломного проекту

Пояснювальна записка складається з основного розділу (аналізу предметної області, проектування застосунку, реалізації застосунку, тестування застосунку), економічного розділу, розділу охорони праці та додатків. Перелічені розділи поетапно охоплюють розробку, виконані докладно та обґрунтовано. Розділ охорони праці містить загальну інформацію та вимоги до техніки безпеки оператора КТ. Економічний розділ проекту містить розрахунок витрат на НДР та реалізацію проекту.

в) оцінка якості виконання пояснювальної записки та графічної частини дипломного проекту

Графічна частина складається з 14 слайдів мультимедійної презентації, виконаної у програмному продукті MS PowerPoint, які містять ілюстративні схеми, скріншоти роботи програмного застосунку, передбачені технічним завданням. Пояснювальна записка виконана акуратно та у відповідності до норм. Якість виконання графічної частини проекту та пояснювальної записки добра, розробку виконано у повному обсязі.

г) перелік позитивних якостей дипломного проекту *Використання Docker забезпечує стабільне середовище розробки та розгортання. Інтеграція з Google Books API виконана із врахуванням обробки непередбачуваних відповідей і захисту API-ключів.*

Продумане проектування структури бази даних, з використанням кешування для оптимізації звернень до зовнішнього API.

д) основні недоліки дипломного проекту

Структура інтерфейсу реалізована на базових HTML/CSS/JS без сучасних frontend-фреймворків (React/Vue), що може ускладнити масштабування UI в майбутньому. Присутні деякі недоліки оформлення пояснювальної записки

Оцінка розрахункової частини *Відмінно*

Оцінка графічної частини *Відмінно*

Загальна оцінка *Відмінно*

Прізвище, ім'я, по батькові рецензента *к.т.н. Шибаєва Наталя Олегівна*

Місце роботи і посада рецензента *Національний університет «Одеська політехніка», доцент кафедри інформаційних технологій*

Підпис:

«*20*» *червня* 2025 р.

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

ВІДГУК

керівника на дипломний проект здобувача (здобувачки) освіти

відділення комп'ютерних систем

Єпура Юрія Олександровича

(прізвище, ім`я та по батькові)

Спеціальність: *123 «Комп'ютерна інженерія»*

Освітня програма: *«Обслуговування комп'ютерних систем і мереж»*

Тема дипломного проекту: *Розробка інтерактивного веб-порталу віртуального клубу читачів .*

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЕКТУ

а) обсяг і якість виконання проекту (графічного матеріалу і розрахунково-пояснювальної записки) *Дипломний проект виконано відповідно технічному завданню. Пояснювальна записка до дипломного проекту містить 85 сторінок. У пояснювальній записці описано етапи розробки інтерактивного веб-порталу віртуального клубу читачів засобами технологій Python, Django, MVC, HTT[REST API, PostgreSQL. Графічна частина складається з окремих 14 слайдів, оформлених у вигляді презентації, передбачених технічним завданням. Якість виконання пояснювальної записки та слайдів добра.*

б) самостійність роботи над проектом: *Протягом виконання дипломного проекту здобувач освіти поступово та послідовно виконував всі етапи, проявляв ініціативу в створенні загальної концепції та реалізації роботи. Всі роботи здобувач освіти виконував самостійно, з оглядом на рекомендації керівника.*

в) теоретична підготовка випускника (випускниці): *Здобувач освіти під час роботи над дипломним проектом вивчив достатньо багато літературних та інтернет-джерел за даною тематикою. Вважаю, що теоретична підготовка дипломника достатня і він готовий до захисту проекту.*

г) вміння розв'язувати виробничі та конструкторські питання *Під час виконання дипломного проекту здобувач освіти показав вміння організовано працювати над поставленим завданням, застосовувати знання у галузі пчрограмування та математики, розробляти, встановлювати та налаштовувати спеціалізоване програмне забезпечення, оформлювати слайди та складати презентації, користуючись сучасними комп'ютерними програмними засобами, такими як Python, Django, MVC, HTTP REST API, PostgreSQL.*

Оцінка розрахункової частини *Відмінно*

Оцінка графічної частини *Добре*

Загальна оцінка *Відмінно*

Прізвище, ім'я, по батькові керівника дипломного проекту *Закроєв Юрій Михайлович*

Місце роботи і посада керівника дипломного проекту *Директор ТОВ «Біг ВОШ».*

Підпис

«16» 06 2025 р.

ДОЗВІЛ
НА РОЗМІЩЕННЯ
ВИПУСКНОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
(ДИПЛОМНОГО ПРОЕКТУ)
В ЕЛЕКТРОННОМУ РЕПОЗИТАРІЇ ВСП «ОТФК ОНТУ»

Ми, що нижче підписалися,

Єпур Юрій Олександрович,
здобувач освіти гр. 4КС-58, та

Закроєв Юрій Михайлович,
керівник дипломного проекту,

не заперечуємо щодо розміщення електронного варіанту пояснювальної записки до дипломного проекту фахового молодшого бакалавра на тему:

«Розробка інтерактивного веб-порталу віртуального клубу читачів» (автор роботи – Єпур Ю.О., керівник роботи – Закроєв Ю.М.)

виконаного у ВСП «Одеський технічний фаховий коледж Одеського національного технологічного університету» в 2025 році, у повному обсязі в електронному репозитарії ВСП «ОТФК ОНТУ» для вільного доступу через мережу Інтернет.

Несемо відповідальність за ідентичність електронного та друкованого варіантів випускної кваліфікаційної роботи і даємо згоду на обробку персональних даних.

Виконавець _____ / Єпур Ю.О. /

Керівник _____ / Закроєв Ю.М. /

«16» червня 2025 р.

ДОВІДКА

циклової комісії КТ та ПІ
про допуск до захисту дипломного проєкту
здобувача (здобувачки) освіти IV курсу
відділення комп'ютерних систем групи 4КС-58

Єпура Юрія Олександровича

на тему *Розробка інтерактивного веб-порталу віртуального клубу читачів*

Висновок відповідальної особи за проведення нормоконтролю:
пояснювальна записка до дипломного проєкту виконана з деякими порушеннями ДСТУ, оформлена відповідно до вимог Положення про дипломне проєктування

(підпис) *16.06.2025* (дата) *Петрашова В.І.* (П.І.Б.)

Висновок відповідальної особи за перевірку роботи на наявність академічного плагіату *згідно звіту про перевірку від 11.06.2025 р. значення коефіцієнту подібності в роботі становить 9,00%, коефіцієнт цитування – 0,85%.*

(підпис) *16.06.2025* (дата) *Краснокутська К.Г.* (П.І.Б.)

Попередня експертиза (малий захист) дипломного проєкту

здобувача (здобувачки) освіти *Єпура Ю.В.* (П.І.Б.)

проведена « *16* » *червня* 2025 р.

Висновки *Пояснювальна записка до дипломного проєкту виконана у повному обсязі. Випускна кваліфікаційна робота (дипломний проєкт) відповідає вимогам Положення про дипломне проєктування та рекомендована до захисту.*

Голова ЦК КТ та ПІ (підпис) *Кривченко Ю.В.* (П.І.Б.)

StrikePlagiarism.com

Дата звіту 6/11/2025
Дата редагування 6/11/2025

Документ прийнятий

Звіт подібності

метадані

Назва організації
Odesa Technical Professional College of Odesa National University of Technology
Заголовок
Розробка інтерактивного веб-порталу віртуального клубу читачів

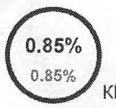
Автор Науковий керівник / Експерт
Єпур Юрій ОлександровичЗакроєв Юрій Михайлович

підрозділ
Відокремлений структурний підрозділ "Одеський технічний фаховий коледж Одеського національного технологічного університету"

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

КП 1



КЦ

25
Довжина фрази для коефіцієнта подібності 2

17262
Кількість слів

144359
Кількість символів

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв		2170
Інтервали		0
Мікропробіли		3
Білі знаки		0
Парафрази (SmartMarks)		112

Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз — Колір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	https://dou.ua/lenta/articles/language-rating-2025/	54 0.31 %
2	https://card-file.ontu.edu.ua/bitstreams/6cf43324-8f08-4031-ba42-f80b18efbbc8/download	51 0.30 %
3	https://card-file.ontu.edu.ua/bitstreams/8999d5af-6274-44f4-ae78-d23e08048d38/download	49 0.28 %
4	https://card-file.ontu.edu.ua/bitstreams/6cf43324-8f08-4031-ba42-f80b18efbbc8/download	42 0.24 %
5	https://card-file.ontu.edu.ua/bitstreams/29489599-0581-4ce6-8890-c3b13d9f2e0e/download	37 0.21 %

6	https://card-file.ontu.edu.ua/bitstreams/8999d5af-6274-44f4-ae78-d23e08048d38/download	35 0.20 %
7	https://card-file.ontu.edu.ua/bitstreams/538ada8a-2c79-4b1e-b7d2-b0c97f68bc1c/download	34 0.20 %
8	https://card-file.ontu.edu.ua/bitstreams/549ee9fe-7574-4ae5-b500-9fe2711f33e6/download	30 0.17 %
9	https://card-file.ontu.edu.ua/bitstreams/549ee9fe-7574-4ae5-b500-9fe2711f33e6/download	29 0.17 %
10	https://card-file.ontu.edu.ua/bitstreams/9908b7a9-6b3e-46f5-a46e-84d83787cfd4/download	27 0.16 %

з домашньої бази даних (0.04 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	Створення web-застосунку цифрового помічника з використанням Open AI [redacted] 5/28/2025 Odesa Technical Professional College of Odesa National University of Technology (Відокремлений структурний підрозділ "Одеський технічний фаховий коледж Одеського національного технологічного університету")	7 (1) 0.04 %

з програми обміну базами даних (0.15 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	36 Лата Богдан дип. робота [redacted] 6/17/2024 Odessa National Polytechnic University (ІЕКСУ, кафедра АЕС)	17 (2) 0.10 %
2	Звіт дипломної [redacted] 12/29/2024 Ivano-Frankivsk National Technical University of Oil and Gas (Каф. КСМ)	9 (1) 0.05 %

з Інтернету (8.81 %)

ПОРЯДКОВИЙ НОМЕР	ДЖЕРЕЛО URL	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	https://card-file.ontu.edu.ua/bitstreams/62baa43e-b968-4993-bb54-8cf8761a89b2/download	717 (64) 4.15 %
2	**https://card-file.ontu.edu.ua/bitstreams/538ada8a-2c79-4b1e-b7d2-b0c97f68bc1c/download**	**133 (8) 0.77 %**
3	**https://card-file.ontu.edu.ua/bitstreams/8999d5af-6274-44f4-ae78-d23e08048d38/download**	**100 (3) 0.58 %**
4	**https://card-file.ontu.edu.ua/bitstreams/6cf43324-8f08-4031-ba42-f80b18efbbc8/download**	**93 (2) 0.54 %**
5	https://card-file.ontu.edu.ua/bitstreams/55e2b8f2-7d3c-4235-99fc-2be51199b96d/download	86 (9) 0.50 %
6	**https://card-file.ontu.edu.ua/bitstreams/549ee9fe-7574-4ae5-b500-9fe2711f33e6/download**	**64 (3) 0.37 %**
7	**https://card-file.ontu.edu.ua/bitstreams/29489599-0581-4ce6-8890-c3b13d9f2e0e/download**	**62 (3) 0.36 %**
8	**https://card-file.ontu.edu.ua/bitstreams/9908b7a9-6b3e-46f5-a46e-84d83787cfd4/download**	**58 (5) 0.34 %**
9	**https://card-file.ontu.edu.ua/server/api/core/bitstreams/8da72e29-656f-4ee4-9b22-716dedf53ff5/content**	**54 (3) 0.31 %**
10	**https://dou.ua/lenta/articles/language-rating-2025/**	**54 (1) 0.31 %**
11	https://card-file.ontu.edu.ua/bitstreams/12d5c0ab-e979-48f2-a8ec-d5fc31f71fd5/download	31 (2) 0.18 %
12	https://card-file.ontu.edu.ua/bitstreams/0e72a3b9-bdd7-4711-a3c6-dedc1d4287cc/download	21 (3) 0.12 %
13	https://card-file.ontu.edu.ua/bitstreams/bbaf3f38-16a8-4070-bead-5562769b7c71/download	12 (1) 0.07 %

14	https://card-file.ontu.edu.ua/server/api/core/bitstreams/4bb7255e-46d4-4349-9726-9698476da02d/content	11 (1) 0.06 %
15	https://card-file.ontu.edu.ua/bitstreams/f789da43-3034-4ad8-bf34-640a47414f93/download	10 (1) 0.06 %
16	https://card-file.ontu.edu.ua/bitstreams/34a6756b-592f-4b77-a805-183aa03a6a26/download	9 (1) 0.05 %
17	https://essuir.sumdu.edu.ua/bitstream-download/123456789/92971/1/Volkov_bac_rob.pdf	5 (1) 0.03 %

Список прийнятих фрагментів (немає прийнятих фрагментів)

ПОРЯДКОВИЙ НОМЕР	ЗМІСТ	КІЛЬКІСТЬ ОДНАКОВИХ СЛІВ (ФРАГМЕНТІВ)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: 123 «Комп'ютерна інженерія»
Освітня програма: «Обслуговування комп'ютерних систем і мереж» Група: 4КС- 58

Дипломний **проект здобувача освіти денної форми навчання КС.** 58.07.000.ДП

ЄПУРА
ЮРІЯ ОЛЕКСАНДРОВИЧА

м. Одеса
2025 р. МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: 123 «Комп'ютерна інженерія»
Освітньо-професійна програма: «Обслуговування комп'ютерних систем і мереж»
Група: 4 КС-58

ПОЯСНЮВАЛЬНА ЗАПИСКА до дипломного проекту на тему:
_____
_____ Проектний матеріал складається з пояснювальної записки на _____ сторінках та графічного (презентаційного) матеріалу на _____ аркушах (слайдах) Дипломник _____ (Єпура Ю.О.)
Керівник _____ (Закроєв Ю.М.)

Консультанти:
з економічного розділу_____ (Канський М.Ю.)
з розділу охорони праці та техніки безпеки_____ (Чорновол Н.І.) з нормоконтролю _____(Петрашова В.І.) старший консультант _____(Кривченко Ю. В.) До захисту допущений Голова циклової комісії _____ (**Кривченко Ю.В.**)
Завідувач відділення _____ (Краснокутська К.Г.)

Захист «____»_____ 2025 р. Протокол ЕК No_____
Оцінка ЕК _____

Секретар ЕК _____
Розробка інтерактивного веб-порталу
віртуального клубу читачів
14
85

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»
Відділення_____**Комісія**_____
Спеціальність_____
Освітньо-професійна програма_____ **ЗАТВЕРДЖУЮ: Заст. дир. з** НВР_____
_____
"_____" _____2025 р. **З А В Д А Н Н Я на дипломний** проект
Здобувачеві освіти _____ **(прізвище, ім'я, по**