КВАЛІФІКАЦІЙНА РОБОТА

Група ІПЗс-2019
Баланюк І.І.

2023

ЗВО УНІВЕРСИТЕТ КОРОЛЯ ДАНИЛА

Факультет суспільних та прикладних наук

Кафедра інформаційних технологій

на правах рукопису

Баланюк Ігор Іванович

УДК 004.624

Розробка електронної бібліотеки методичного забезпечення дисциплін університету з використанням веб-технологій

Спеціальність 121 – «Інженерія програмного забезпечення»

Кваліфікаційна робота на здобуття кваліфікації бакалавра

Нормоконтроль

_____Стисло О.В.
(підпис, дата, розшифрування підпису)

Студент

_____Баланюк І.І.
(підпис, дата, розшифрування підпису)

Допускається до захисту
Завідувач кафедри

_____к.т.н., доц. Пашкевич О.П.
(підпис, дата, розшифрування підпису)

Керівник роботи

_____ к.т.н., доц. Слабінога М.О.
(підпис, дата, розшифрування підпису)

Івано-Франківськ – 2023

ЗВО УНІВЕРСИТЕТ КОРОЛЯ ДАНИЛА

Факультет суспільних та прикладних наук

Кафедра інформаційних технологій

Освітній ступінь: «бакалавр»

Спеціальність: 121 «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____

« ___ » _____ 2023 року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Баланюк Ігор Іванович

(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи
Розробка електронної бібліотеки методичного забезпечення дисциплін університету з використанням веб-технологій
керівник роботи:
Слабінога Мар'ян Остапович, кандидат технічних наук
затверджена наказом вищого навчального закладу від «11» листопада 2022 року № 155/1НВ
2. Гермін подання студентом роботи 14.06.2023
3. Вихідні дані роботи: Мова програмування Python, Flask
4. Зміст кваліфікаційної роботи (перелік питань, які потрібно розробити)
   1. Аналіз аналогів аналогів
   2. Проектування структури сайту
   3. Розробка структури веб-сайту
   4. Проведення тестування
5. Дата видачі завдання 10.10.2022

КОНСУЛЬТАНТИ РОЗДІЛІВ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Розділ	Консультант (прізвище, ініціали та посада)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1.	Пошук інформації альтернативні програми	10.01.2023	Виконано
2.	Проектування та розробка застосунку	15.01.2023	Виконано
3.	Програмна реалізація застосунку	20.02.2023	Виконано
4.	Формування висновків Охорона праці	12.04.2023	Виконано
5.	Оформлення пояснювальної записки	15.05.2023	Виконано
6.	Оформлення графічного матеріалу та	01.06.2023	Виконано
7.	Підготовка до захисту роботи	03.06.2023	Виконано

Студент _____ Баланюк І.В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Слабінога М.О.
(підпис) (прізвище та ініціали)

Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

АНОТАЦІЯ

Дипломна робота присвячена розробці електронної бібліотеки методичного забезпечення університету за допомогою мови програмування Python та його фреймворку Flask.

Перший розділ присвячений огляду аналогів, опису методів розробки електронної бібліотеки та виборі мови програмування.

В другому розділі спроектовано базу даних, описано функціональні вимоги, та розроблено серверну частину.

Третій розділ представляє процес реалізації програми мовою Python.

У четвертому розділі виконується тестування продукту та проводиться оцінка якості розробленої системи.

КЛЮЧОВІ СЛОВА: PYTHON, БІБЛІОТЕКА, ЕЛКТРОННА БІБЛІОТЕКА, УНІВЕРСИТЕТ.

SUMMARY

The thesis is dedicated to the development of an electronic library for the methodological support of the university using the Python programming language and its Flask framework.

The first chapter focuses on the overview of analogs, the description of methods for developing an electronic library, and the choice of programming language.

In the second chapter, the database is designed, functional requirements are described, and the server-side is developed.

The third chapter presents the implementation process of the program using the Python language.

The fourth chapter includes product testing and the evaluation of the developed system's quality.

KEYWORDS: PYTHON, LIBRARY, ELECTRONIC LIBRARY, UNIVERSITY.

ЗМІСТ

ВСТУП

Актуальність теми. Розвиток технологій призвів до значних змін у способах зберігання інформації, доступу до неї та обміну нею. Однією з ключових сфер, де технології зробили значний вплив, є сфера бібліотечних послуг. В останні роки зростання кількості електронних бібліотек різко збільшилося. Це пов'язано з легкістю і зручністю доступу до цифрових ресурсів з будь-якого місця і в будь-який час. Метою даної дипломної роботи є опис розробки електронної бібліотеки за допомогою мови програмування Python.

Використання Python у розробці веб-додатків в останні роки зросло. Python - це мова програмування високого рівня, яка проста у вивченні та розумінні. Вона широко використовується для розробки веб-додатків, аналізу даних і додатків для машинного навчання. Основною причиною популярності Python є її читабельність, що дозволяє розробникам писати код, який легко зрозуміти та підтримувати.

Метою даної дипломної роботи є демонстрація розробки електронної бібліотеки з використанням фреймворку Flask. Flask- це високорівневий веб-фреймворк на мові Python, який дозволяє розробникам швидко створювати складні веб-додатки. Він надає надійний набір інструментів і бібліотек, які полегшують розробку веб-додатків, включаючи аутентифікацію, підключення до баз даних і шаблонні механізми.

Електронна бібліотека, розроблена в цій дипломній роботі, надасть користувачам можливість шукати, переглядати і завантажувати книги, статті та інші цифрові ресурси. Користувачі також зможуть створювати облікові записи, керувати своєю особистою інформацією та створювати списки літератури. Електронна бібліотека буде спроектована таким чином, щоб бути масштабованою, безпечною і простою у використанні.

У першому розділі цієї дипломної роботи буде надано огляд електронних бібліотек та їхніх переваг. У другому розділі буде розглянуто мову

програмування Python та веб-фреймворк Flask. У третьому розділі буде описано дизайн та реалізацію електронної бібліотеки. У четвертому розділі буде проведена оцінка ефективності роботи електронної бібліотеки, а в п'ятому розділі буде зроблено висновок про підсумки роботи, узагальнено основні висновки та намічено напрямки подальшої роботи.

Таким чином, у цій дипломній роботі буде представлено розробку електронної бібліотеки з використанням мови програмування Python та веб-фреймворку Flask. Електронна бібліотека надасть користувачам можливість шукати, переглядати та завантажувати цифрові ресурси, а також керувати своєю особистою інформацією та списками літератури.

Мета роботи. Метою даної роботи є розробка електронної бібліотеки методичного забезпечення дисциплін університету з використанням веб-технологій, що триматиме все методичне забезпечення в одному місці.

Об’єкт роботи. Місце зберігання методичного забезпечення, та систематизація доступу до нього.

Предмет роботи. Веб-сайт для зберігання методичного забезпечення.

Завдання роботи. Відповідно до вибраної теми в роботі покладені задачі:

- пошук та аналіз уже існуючих аналогів;
- вибір мови програмування, технологій та інших суміжних програм (при необхідності);
- проектування структури бібліотеки;
- розроблення сайту з різноманітним функціоналом;
- проведення тестування продукту.

Методи роботи. У роботі використано такі методи, як аналіз, опис та моделювання, що є дослідницькими методами.

Результати роботи. Результатом виконання кваліфікаційної роботи є електронна бібліотека, яка дозволить зберігати методичне забезпечення.

Структура роботи. Розділи – 4. Загальний обсяг основної частини – 53 сторінок. Список використаних джерел – 20.

РОЗДІЛ 1. ТЕОРЕТИЧНІ АСПЕКТИ РОЗРОБКИ ЕЛЕКТРОННОЇ БІБЛІОТЕКИ

1.1 Огляд існуючих електронних бібліотек та їхніх особливостей

Електронні бібліотеки, також відомі як цифрові бібліотеки, - це веб-платформи, які надають доступ до цифрових ресурсів, таких як електронні книги, журнали, наукові роботи та інші види публікацій. Ці бібліотеки стають дедалі популярнішими завдяки зручності, яку вони пропонують з точки зору доступу до інформації з будь-якого місця і в будь-який час. У цьому розділі ми надамо огляд деяких з існуючих електронних бібліотек та їх ключових особливостей.

Однією з найвідоміших електронних бібліотек є Цифрова публічна бібліотека Америки (Digital Public Library of America, DPLA) (рис 1.1). DPLA - це національна цифрова бібліотека, яка надає доступ до мільйонів цифрових ресурсів з бібліотек, архівів та музеїв по всій території Сполучених Штатів. Платформа пропонує зручний інтерфейс, який дозволяє користувачам здійснювати пошук і перегляд колекцій, а також створювати персоналізовані колекції та ділитися ними з іншими. DPLA також пропонує інструменти та послуги для розробників, які можуть створювати додатки, використовуючи їхні дані та API.

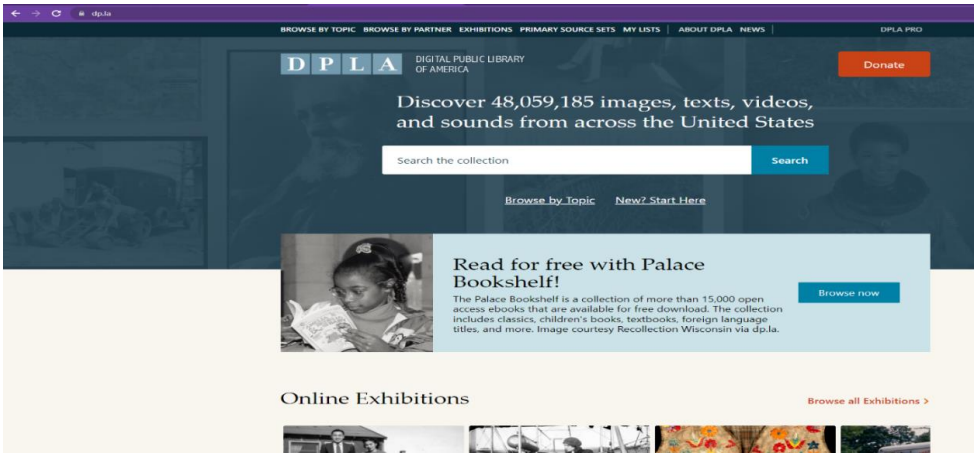

Рисунок 1.1 – Сайт DPLA

Ще однією популярною електронною бібліотекою є HathiTrust Digital Library. Ця бібліотека є партнерством між кількома академічними та дослідницькими установами і надає доступ до понад 17 мільйонів цифрових ресурсів. Цифрова бібліотека HathiTrust пропонує різноманітні інструменти пошуку та виявлення, а також можливості для користувачів створювати колекції та обмінюватися ними. На додаток до своєї величезної колекції цифрових ресурсів, Цифрова бібліотека HathiTrust (рис. 1.2) також надає послуги з інтелектуального аналізу даних та комп'ютерного аналізу свого контенту.

Рисунок 1.2 – Сайт HathiTrust

Інтернет-архів - ще одна відома електронна бібліотека, яка надає доступ до мільйонів цифрових ресурсів. Інтернет-архів пропонує різноманітні колекції, включаючи книги, музику, відео та веб-сторінки. Платформа також включає веб-архів, який дозволяє користувачам отримувати доступ до історичних версій веб-сайтів і переглядати їх. Інтернет-архів надає широкий спектр інструментів пошуку та виявлення, а також можливості для користувачів створювати колекції та обмінюватися ними.

На додаток до цих великих електронних бібліотек, існує також багато спеціалізованих електронних бібліотек, які обслуговують конкретні галузі або теми. Наприклад, електронна бібліотека arXiv.org зосереджена на наданні доступу до наукових робіт у галузі фізики, математики, інформатики та інших суміжних дисциплін. Електронна бібліотека JSTOR зосереджена на наданні доступу до академічних журналів та інших наукових публікацій.

Функції електронних бібліотек можуть відрізнятися залежно від платформи та її призначення. Однак, деякі загальні риси включають інструменти пошуку і виявлення, доступ до цифрових ресурсів, персоналізовані колекції, а також інструменти для спільного використання і співпраці. Електронні бібліотеки можуть також включати функції для інтелектуального аналізу даних, обчислювального аналізу та машинного навчання.

Таким чином, електронні бібліотеки забезпечують зручний і доступний спосіб доступу до цифрових ресурсів. Існуючі електронні бібліотеки пропонують користувачам різноманітні функції та послуги для пошуку, перегляду та відкриття цифрових ресурсів, а також для створення та обміну персоналізованими колекціями.

Однією з найвідоміших українських електронних бібліотек є Національна бібліотека України імені В. І. Вернадського (рис. 1.3). Ця бібліотека є національною бібліотекою України і надає доступ до понад 15 мільйонів ресурсів, включаючи книги, рукописи та інші архівні матеріали. Бібліотека пропонує зручний інтерфейс, який дозволяє користувачам здійснювати пошук і перегляд колекцій, а також доступ до низки цифрових ресурсів, включаючи електронні книги та онлайнові бази даних.

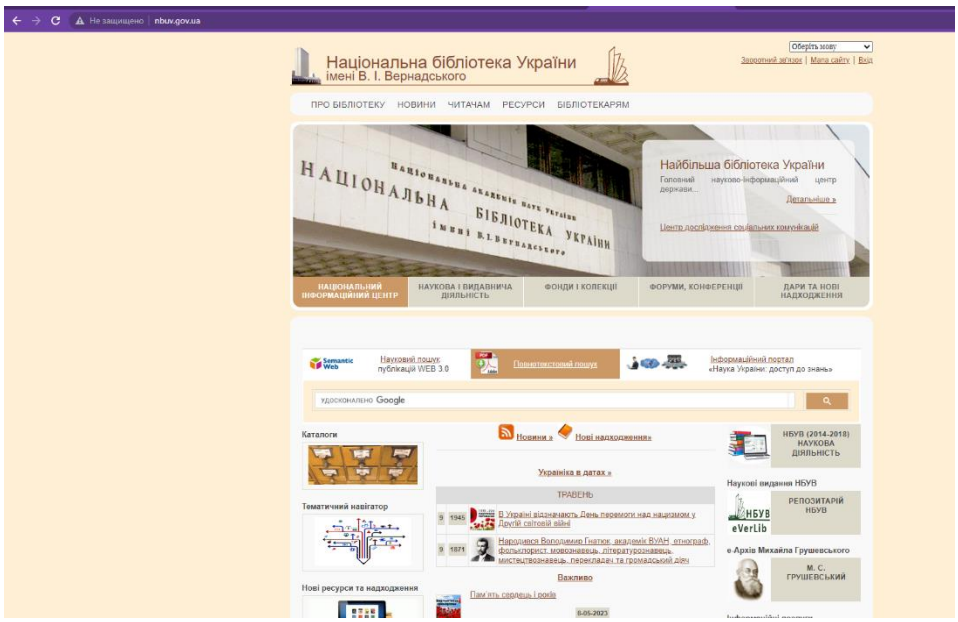

Рисунок 1.3 – Бібліотека Вернадського

Ще однією відомою українською електронною бібліотекою є Українська національна електронна бібліотека. Ця бібліотека надає доступ до низки цифрових ресурсів, включаючи книги, газети, журнали та архівні матеріали. Бібліотека пропонує різноманітні інструменти пошуку та виявлення, а також можливості для користувачів створювати та обмінюватися персоналізованими колекціями.

Бібліотека української літератури – ще одна важлива електронна бібліотека, яка зосереджена на наданні доступу до української літератури та пов'язаних з нею матеріалів. Ця бібліотека пропонує широкий спектр ресурсів, включаючи книги, журнали, аудіо- та відеозаписи, а також інструменти пошуку та виявлення, можливості для створення персоналізованих колекцій.

На додаток до цих українських електронних бібліотек, існує також багато спеціалізованих електронних бібліотек, які зосереджені на конкретних галузях або темах, пов'язаних з Україною. Наприклад, Українська наукова електронна бібліотека надає доступ до наукових публікацій з математики, фізики та інших суміжних дисциплін. Українська бібліотека фольклору пропонує широкий спектр ресурсів, пов'язаних з українським фольклором і традиційною культурою.

Функції електронних бібліотек можуть відрізнятися залежно від платформи та її призначення. Однак деякі загальні риси включають інструменти пошуку та виявлення, доступ до цифрових ресурсів, персоналізовані колекції, а також інструменти для спільного використання та співпраці. Електронні бібліотеки можуть також включати функції для інтелектуального аналізу даних, обчислювального аналізу та машинного навчання.

Таким чином, електронні бібліотеки забезпечують зручний і доступний спосіб доступу до цифрових ресурсів. Українські електронні бібліотеки, такі як Національна бібліотека України імені В. І. Вернадського, Українська національна електронна бібліотека та Бібліотека української літератури, пропонують користувачам широкий спектр цифрових ресурсів та функцій для пошуку, перегляду та відкриття цифрових ресурсів. Ці бібліотеки є важливими

інструментами для дослідників, викладачів та студентів, які цікавляться українською літературою, культурою та історією.

1.2 Основні функції електронної бібліотеки

Python та Flask є популярними інструментами для розробки веб-додатків, в тому числі електронних бібліотек. У цьому підрозділі проведено огляд методів та алгоритмів розробки електронної бібліотеки за допомогою Python та Flask.

Процес розробки електронної бібліотеки за допомогою Python та Flask можна розбити на кілька етапів. Перший етап передбачає проектування схеми бази даних та створення необхідних таблиць для зберігання метаданих та контенту бібліотеки. Другий етап - розробка користувацького інтерфейсу, що включає дизайн макету та реалізацію функцій пошуку та знаходження. Заключний етап передбачає інтеграцію інтерфейсу користувача з базою даних та реалізацію необхідного функціоналу для доступу до даних та маніпулювання ними [1-2].

Одним з найважливіших етапів розробки електронної бібліотеки є проектування схеми бази даних. Схема бази даних визначає структуру бази даних і зв'язки між різними таблицями. У типовій електронній бібліотеці схема бази даних включатиме таблиці для зберігання метаданих про книги, таких як автор, назва і дата публікації, а також таблиці для зберігання фактичного вмісту книг, таких як PDF-файли або EPUB-файли.

Після розробки схеми бази даних наступним кроком є створення необхідних таблиць у базі даних. Це можна зробити за допомогою таких інструментів, як SQLite або MySQL. Після створення таблиць наступним кроком є заповнення їх даними. Це можна зробити вручну або за допомогою автоматизованого інструменту, наприклад, веб-скрепера.

Наступним кроком у розробці електронної бібліотеки є створення користувацького інтерфейсу. Це включає в себе розробку макету та реалізацію функцій пошуку і знаходження. Одним з популярних інструментів для

проектування веб-інтерфейсів є HTML, який використовується для створення структури і змісту веб-сторінок. CSS використовується для стилізації сторінок і надання їм візуальної привабливості, а JavaScript - для додавання інтерактивності та динамічної поведінки.

Flask - популярний веб-фреймворк для Python, який надає широкий спектр можливостей для розробки веб-додатків. Flask можна використовувати для обробки запитів користувачів, реалізації автентифікації та авторизації, а також для управління зв'язком між користувацьким інтерфейсом і базою даних. Flask також надає ряд розширень і плагінів, які можна використовувати для додавання додаткової функціональності до додатку [3-4].

Для доступу до даних в базі даних з інтерфейсу користувача зазвичай використовується інтерфейс прикладного програмування (API). API надає набір функцій, які дозволяють користувацькому інтерфейсу взаємодіяти з базою даних і отримувати необхідні дані. Одним з популярних API для Python є Flask-RESTful, який надає ряд функцій для створення RESTful API.

Таким чином, розробка електронної бібліотеки за допомогою Python та Flask включає кілька етапів, серед яких проектування схеми бази даних, створення необхідних таблиць, розробка користувацького інтерфейсу та інтеграція користувацького інтерфейсу з базою даних за допомогою API. Python і Flask надають потужний набір інструментів для розробки веб-додатків, а їх популярність робить їх ідеальним вибором для розробки електронних бібліотек.

Окрім проектування схеми бази даних та розробки користувацького інтерфейсу, є ще кілька методів та алгоритмів, які можна використати для покращення функціональності та зручності електронної бібліотеки, розробленої за допомогою Python та Flask.

Одним з таких методів є повнотекстовий пошук. Повнотекстовий пошук дозволяє користувачам шукати ключові слова у змісті книг, а не просто шукати метадані, такі як назва або автор. Це можна реалізувати за допомогою пошукової системи, наприклад, Elasticsearch або Solr, які можуть індексувати зміст книг і надавати швидкі та точні результати пошуку [5-6].

Іншим важливим алгоритмом є рекомендаційні системи. Рекомендаційні системи можуть бути використані для того, щоб пропонувати користувачам книги на основі їхніх інтересів та історії читання. Це може бути реалізовано за допомогою алгоритмів машинного навчання, таких як спільна фільтрація або фільтрація на основі контенту. Колаборативна фільтрація аналізує поведінку інших користувачів і рекомендує книги на основі їхніх уподобань, тоді як фільтрація на основі контенту аналізує зміст книг і рекомендує ваідповідні книги зі схожим змістом.

Контроль доступу - ще один важливий аспект розробки електронної бібліотеки. Контроль доступу можна використовувати для обмеження доступу до певних книг або контенту на основі ролей або дозволів користувачів. Це можна реалізувати за допомогою механізмів автентифікації та авторизації, таких як OAuth або JSON Web Tokens.

Нарешті, оптимізація продуктивності також є важливим фактором при розробці електронної бібліотеки. Великі колекції книг можуть швидко стати громіздкими і повільними для пошуку, особливо якщо контент зберігається в базі даних. Одним із способів вирішення цієї проблеми є використання механізму кешування, такого як Redis, який може зберігати в пам'яті дані, до яких часто звертаються, і зменшити навантаження на базу даних.

Розробка електронної бібліотеки з використанням Python і Flask передбачає використання цілого ряду методів і алгоритмів, включаючи повнотекстовий пошук, рекомендаційні системи, контроль доступу та оптимізацію продуктивності. Використовуючи можливості цих інструментів і методів, можна створити високофункціональну і зручну в користуванні електронну бібліотеку, яка зможе задовольнити потреби користувачів.

Основні функції :

- додавання;
- видалення;
- перегляд;
- редагування.

1.3 Порівняльний аналіз технологій

Для розробки електронних бібліотек доступно багато мов програмування, кожна з яких має свої сильні та слабкі сторони. У цьому розділі ми розглянемо найпопулярніші мови програмування для розробки електронних бібліотек.

Python є однією з найпопулярніших мов програмування для розробки електронних бібліотек завдяки своїй простоті та зручності використання. Він має велику кількість бібліотек і фреймворків, які можна використовувати для швидкої та ефективної розробки електронних бібліотек. Одним із таких фреймворків є Flask, який є мікровеб-фреймворком, який ідеально підходить для розробки веб-додатків малого та середнього розміру. Flask легкий і гнучкий, тому його легко налаштовувати та розширювати [7-8].

Java — ще одна популярна мова програмування для розробки електронних бібліотек. Це мова високого рівня, яка розроблена так, щоб бути незалежною від платформи, що означає, що код, написаний на Java, може працювати на будь-якій платформі, яка підтримує віртуальну машину Java (JVM). Java широко використовується в корпоративних спеціальних програмах і має велику кількість бібліотек і фреймворків, доступних для розробки електронних бібліотек.

C++ — це мова програмування низького рівня, яка широко використовується для розробки високопродуктивних програм. Він часто використовується для розробки складних систем, таких як операційні системи та бази даних, але також може використовуватися для розробки електронних бібліотек. C++ відомий своєю швидкістю та ефективністю, що робить його ідеальним для обробки великих обсягів даних.

JavaScript — популярна мова програмування для розробки клієнтських програм, які запускаються у веб-браузері. Він широко використовується для розробки динамічних та інтерактивних веб-додатків і може бути використаний для розробки електронних бібліотек, які мають багатий інтерфейс користувача.

Фреймворки JavaScript, такі як React і Angular, можна використовувати для розробки великомасштабних програм зі складним інтерфейсом користувача.

Ruby — мова програмування високого рівня, яка часто використовується для розробки веб-додатків. Він відомий своєю простотою, швидкістю та легкістю використання, що робить його популярним вибором для розробки електронних бібліотек. Ruby on Rails — популярний фреймворк для розробки веб-додатків на Ruby.

Python:

Переваги:

- простий і легкий для вивчення синтаксис;
- велика кількість доступних бібліотек та фреймворків, включаючи Flask для веб-розробки;
- крос-платформна сумісність;
- добре підходить для швидкого створення прототипів та розробки;
- динамічна верстка забезпечує гнучкість та швидкість розробки.

Недоліки:

- нижча продуктивність порівняно з мовами нижчого рівня;
- не ідеально підходить для високопродуктивних додатків або системного програмування;
- не дуже добре підходить для великомасштабних проектів.

Java:

Переваги:

- незалежність від платформи завдяки використанню JVM;
- велика кількість доступних бібліотек та фреймворків;
- підходить для великомасштабних проектів;
- відмінна продуктивність та масштабованість;
- надійна перевірка типів забезпечує коректність коду.

Недоліки:

- більш складний синтаксис у порівнянні з іншими мовами;
- потребує значного налаштування та конфігурації;

- може займати багато пам'яті;
- крута крива навчання для початківців.

C++:

Переваги:

- відмінна продуктивність та ефективність;
- можна використовувати для системного програмування та високопродуктивних додатків;
- сильна типізація з потужним управлінням пам'яттю;
- велика кількість доступних бібліотек.

Недоліки:

- крута крива навчання через складний синтаксис;
- може бути важко налагоджувати та підтримувати;
- обмежена крос-платформенна сумісність;
- не дуже добре підходить для швидкого створення та розробки.

JavaScript:

Переваги:

- широко використовується для веб-розробки, в тому числі для написання сценаріїв на стороні клієнта;
- велика кількість доступних фреймворків та бібліотек, включаючи React та Angular;
- легкий для вивчення синтаксис;
- динамічна типізація забезпечує гнучкість та швидшу розробку.

Недоліки:

- обмежена браузерними додатками;
- можуть виникати проблеми з сумісністю з браузерами;
- не дуже добре підходить для програмування на стороні сервера;
- менш продуктивна, ніж мови нижчого рівня.

Ruby:

Переваги:

- простий і легкий для вивчення синтаксис;

- підходить для веб-розробки з фреймворком Ruby on Rails;
- сильний фокус на продуктивності розробника та простоті використання;
- велика кількість доступних бібліотек та фреймворків.

Недоліки:

- повільніша продуктивність у порівнянні з мовами нижчого рівня;
- не так широко використовується, як деякі інші мови;
- обмежена веб-розробкою та деякими скриптовими програмами.

Не дуже добре підходить для системного програмування або високопродуктивних додатків.

Підсумовуючи, існує багато мов програмування, доступних для розробки електронних бібліотек, кожна з яких має свої сильні та слабкі сторони. Python є популярним вибором через його простоту та легкість використання, тоді як Java широко використовується в корпоративних програмах. C++ відомий своєю швидкістю та ефективністю, JavaScript популярний для розробки програм на стороні клієнта, а Ruby відомий своєю простотою та легкістю використання. Вибір мови програмування буде залежати від конкретних вимог електронної бібліотеки, що розробляється.

Python - це високорівнева, інтерпретована мова програмування, яка за останні роки набула величезної популярності, і часто є мовою вибору для розробників у різних галузях. У цьому розділі ми обговоримо причини вибору Python як мови програмування для розробки електронної бібліотечної системи.

Однією з головних переваг використання Python є її простота і легкість у використанні. Python має простий і легкий для вивчення синтаксис, що робить її чудовим вибором для початківців. Його чиста і читабельна структура коду також полегшує його підтримку та оновлення, що може бути критично важливим фактором у великомасштабних проектах. Крім того, Python має велику та активну спільноту розробників, які роблять свій внесок у розвиток бібліотек та фреймворків, що можуть спростити процес розробки.

Ще однією причиною вибору Python є його велика екосистема бібліотек та фреймворків. Python має велику колекцію готових бібліотек, які можна використовувати для спрощення процесу розробки. Це, зокрема, бібліотеки для маніпулювання даними, веб-розробки та машинного навчання. Flask, популярний веб-фреймворк, написаний мовою Python і може використовуватися для розробки веб-додатків.

Наявність готових бібліотек та фреймворків може пришвидшити час розробки та зменшити кількість коду, який потрібно написати.

Ще однією перевагою Python є його крос-платформенна сумісність. Код на Python можна запускати на різних операційних системах без необхідності вносити значні зміни. Це робить її ідеальним вибором для розробки електронних бібліотек, які можна використовувати на платформах і пристроях.

Динамічна типізація Python також забезпечує гнучкість та пришвидшує розробку. Динамічна типізація означає, що тип даних змінної визначається під час виконання, що зменшує потребу в оголошеннях типів і робить розробку коду простішою. Це може бути особливо корисно при роботі над масштабними проектами, які вимагають значного кодування.

Нарешті, популярність Python в індустрії є ще одним фактором, який сприяє його вибору як мови програмування для розробки електронних бібліотек. Python широко використовується в різних галузях і сферах, зокрема в науці про дані, машинному навчанні та веб-розробці.

Це означає, що існує великий пул розробників і ресурсів, доступних для підтримки, що може бути корисним при розробці складних додатків.

Отже, вибір Python як мови програмування для розробки електронної бібліотечної системи виправданий його простотою, великою екосистемою бібліотек і фреймворків, крос-платформною сумісністю, динамічною типізацією та популярністю в індустрії. Ці фактори роблять Python ідеальним вибором для розробки ефективних, масштабованих і простих в обслуговуванні електронних бібліотек.

Висновки до розділу 1

В розділі проаналізовано існуючі електронні бібліотеки, розглянуто їхні особливості та недоліки. Після огляду аналогів описано методи та алгоритми розробки електронної бібліотеки. Вибрано мову та технології розробки. Також обгрунтовано вибір мов.

РОЗДІЛ 2. ОПИС ФУНКЦІОНАЛЬНИХ ВИМОГ ДО ЕЛЕКТРОННОЇ БІБЛІОТЕКИ

2.1 Опис функціональних вимог до електронної бібліотеки

Щоб показати функції приведемо діаграму Use Case (рис 2.1)

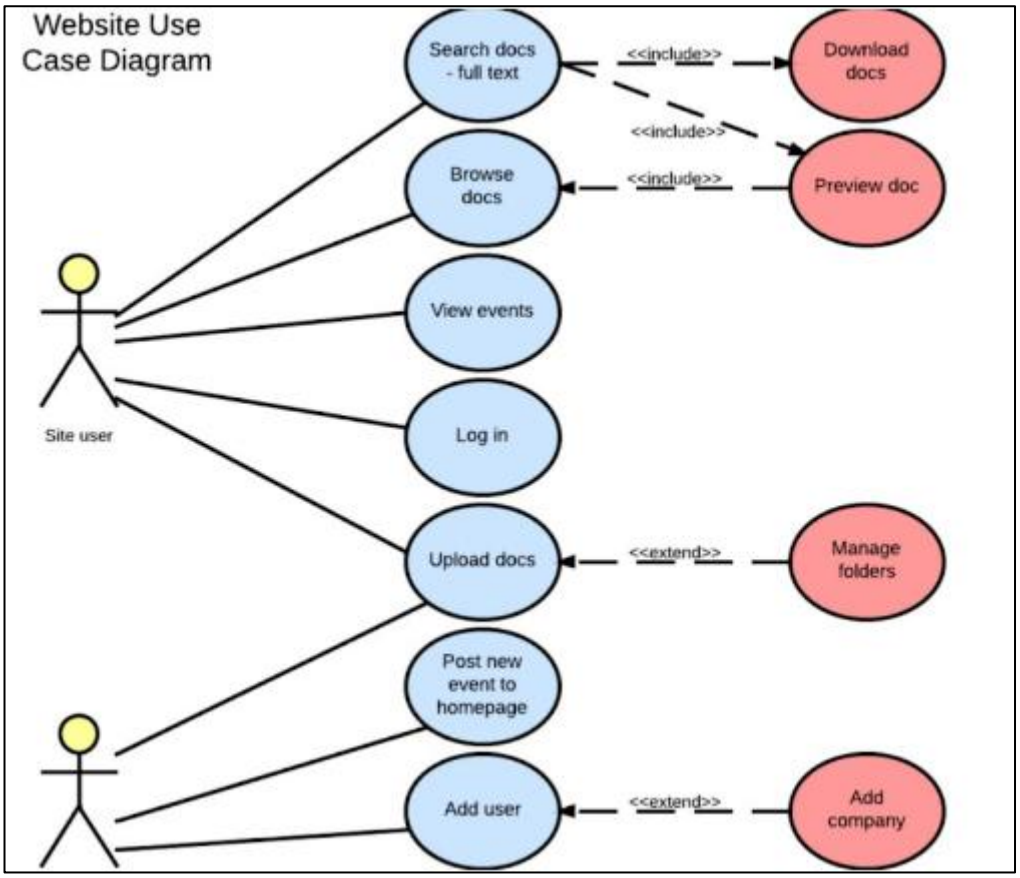

Рисунок 2.1 – Діаграма Use Case на тему “Електронна бібліотека”

Однією з основних функцій електронної бібліотечної системи є надання можливості користувачам реєструватися та входити в систему. Система повинна дозволяти користувачам створювати обліковий запис, вказуючи своє ім'я, адресу електронної пошти та пароль. Після реєстрації користувачі повинні мати можливість увійти в систему, використовуючи свою адресу електронної пошти та пароль.

Система повинна дозволяти користувачам шукати книги в бібліотеці за назвою, автором або ключовим словом. Функція пошуку повинна бути інтуїтивно зрозумілою і зручною, щоб користувачі могли легко знаходити книги, які вони шукають.

Система повинна дозволяти користувачам виписувати книги з бібліотеки. Коли користувач виписує книгу, система повинна відстежувати термін її повернення і надсилати нагадування користувачеві до того, як він її отримає. Коли користувач повертає книгу, система повинна оновити статус книги в інвентарі бібліотеки.

Користувачі повинні мати можливість резервувати книги, які в даний момент виписані іншими користувачами. Коли зарезервована книга стає доступною, система повинна повідомити про це користувача, який її зарезервував, і утримувати книгу протягом певного періоду часу.

Система повинна дозволяти користувачам керувати своїми профілями, в тому числі оновлювати особисту інформацію, переглядати історію позик та керувати бронюванням книг.

Користувачі повинні мати можливість залишати відгуки та оцінки прочитаних книг. Система повинна відображати ці відгуки та оцінки на сторінці книги, дозволяючи іншим користувачам бачити, що інші думають про книгу.

Система повинна включати адміністративну панель, яка дозволяє адміністратору керувати інвентарем бібліотеки, переглядати історію користування та керувати обліковими записами користувачів. Панель також повинна надавати аналітику про використання бібліотеки, включаючи найпопулярніші книги та кількість книг, взятих за певний період часу.

Система повинна мати можливість інтегруватися із зовнішніми системами, такими як онлайн-платіжні шлюзи та постачальники книг, щоб забезпечити безперебійну роботу користувачів та ефективно управляти бібліотечним інвентарем.

Отже, електронна бібліотечна система повинна мати зручний та інтуїтивно зрозумілий інтерфейс, а також відповідати переліченим вище функціональним

вимогам. Використання Python та Flask як інструментів розробки забезпечить швидкий та ефективний процес розробки, а також створить надійну та надійну систему [9].

2.2 Проектування архітектури бази даних

Проектування архітектури бази даних (рис 2.2) - важливий крок у створенні електронної бібліотеки. База даних - це центральний компонент, який зберігає всю інформацію про книги, авторів, користувачів та їхні взаємовідносини. База даних повинна бути спроектована таким чином, щоб бути ефективною, масштабованою і простою в обслуговуванні.

У цьому розділі ми обговоримо ключові аспекти проектування архітектури бази даних для електронної бібліотеки. Ми почнемо з визначення сутностей та їхніх зв'язків, після чого визначимо атрибути і типи даних. Потім ми обговоримо процес нормалізації та різні рівні нормалізації. Нарешті, ми торкнемося стратегій індексування та розбиття на розділи для оптимізації продуктивності бази даних.

Є чотири основні сутності: книги, автори, користувачі та транзакції. Сутність книги має зв'язок багато-до-багатьох з авторами, що означає, що книга може мати кількох авторів, а автор може написати кілька книг. Сутність транзакції має зв'язок "багато-до-одного" як з книгами, так і з користувачами, що означає, що користувач може позичити кілька книг, а книга може бути позичена кількома користувачами.

Після визначення сутностей та їх зв'язків, наступним кроком є визначення атрибутів та типів даних для кожної сутності. Атрибут - це характеристика сутності, наприклад, назва книги або ім'я автора. Тип даних визначає тип даних, які можуть зберігатися в атрибуті, наприклад, рядок, ціле число або дата.

Наприклад, сутність books може мати такі атрибути:

- ідентифікатор (ціле число);
- назва (рядок);
- опис (рядок);

- дата публікації (дата);
- ISBN (рядок);
- зображення обкладинки (бінарні дані);
- жанр (рядок).

Аналогічно, сутність author може мати такі атрибути:

- ID (ціле число);
- ім'я (рядок);
- біографія (рядок);
- зображення профілю (бінарні дані).

Типи даних для кожного атрибуту залежатимуть від вимог програми та системи управління базами даних, що використовується.

Нормалізація - це процес організації структури бази даних таким чином, щоб зменшити залежність. Це допомагає поліпшити цілісність даних, зменшити вимоги до зберігання даних і підвищити продуктивність запитів.

Існують різні рівні нормалізації, починаючи від першої нормальної форми (1НФ) до п'ятої нормальної форми (5НФ). Кожен рівень нормалізації має свій власний набір правил і рекомендацій. Загалом, чим вищий рівень нормалізації, тим складнішою стає структура бази даних.

Для електронної бібліотеки база даних повинна бути нормалізована принаймні до третьої нормальної форми (3НФ). Це означає, що кожна таблиця повинна мати первинний ключ, а кожен неключовий атрибут повинен залежати тільки від первинного ключа.

Індексування та розбиття на розділи - це дві стратегії оптимізації продуктивності бази даних. Індексування передбачає створення індексів на одному або декількох стовпцях таблиці, що дозволяє базі даних швидко шукати певні дані. Розбиття на розділи передбачає поділ великої таблиці на менші розділи на основі певного критерію, наприклад, дати або географічного розташування (рис. 2.2).

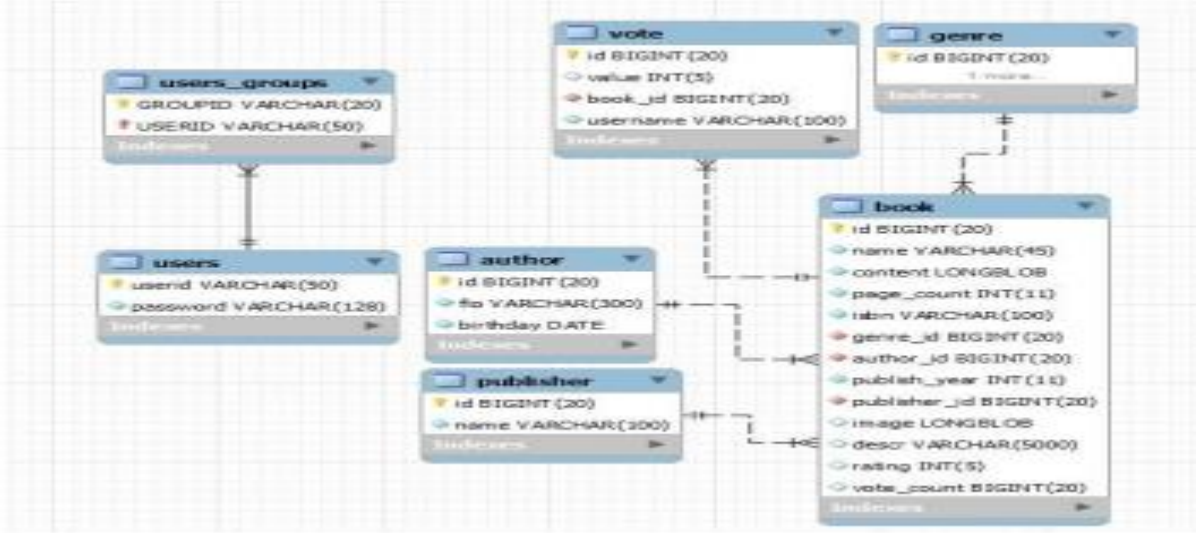

Рисунок 2.2 – Модель спроектованої бази даних

Таблиця "user_groups":

user_id: унікальний ідентифікатор користувача.

group_id: унікальний ідентифікатор групи, до якої належить користувач.

Таблиця "genre":

genre_id: унікальний ідентифікатор жанру книги.

genre_name: назва жанру, наприклад, "Action", "Adventure", "RPG" і т.д.

Таблиця "publisher":

publisher_id: унікальний ідентифікатор видавця.

publisher_name: назва видавця,

Таблиця "vote":

vote_id: унікальний ідентифікатор голосу.

user_id: унікальний ідентифікатор користувача, який проголосував.

game_id: унікальний ідентифікатор книги, за яку було віддано голос.

rating: оцінка, яку користувач присвоїв грі.

2.3 Розробка інтерфейсу користувача

Інтерфейс користувача (UI) — це частина програмної системи, з якою взаємодіють користувачі. Метою дизайну інтерфейсу користувача є створення

зручного та інтуїтивно зрозумілого інтерфейсу, який дозволяє користувачам легко та ефективно орієнтуватися в системі. У випадку електронної бібліотеки інтерфейс користувача відіграє вирішальну роль у забезпеченні користувачам легкого доступу до ресурсів і функцій бібліотеки [13-15].

Розробка інтерфейсу користувача для електронної бібліотеки включає в себе кілька етапів, включаючи визначення вимог користувача, створення каркасів і макетів і реалізацію остаточного дизайну за допомогою HTML, CSS і JavaScript. У наступних розділах описано ці кроки більш детально.

Першим кроком у розробці інтерфейсу користувача є визначення вимог користувача. Це передбачає визначення цільової аудиторії, її потреб і переваг, а також завдань, які вони виконуватимуть у системі. Цільова аудиторія електронної бібліотеки може включати студентів, дослідників та інших осіб, яким потрібен доступ до академічних ресурсів.

Щоб визначити вимоги користувачів, дизайнери можуть проводити дослідження користувачів, такі як опитування та інтерв'ю, щоб зібрати відгуки та ідеї від потенційних користувачів. Потім цю інформацію можна використовувати для створення персонажів користувачів, які є вигаданими представленнями різних типів користувачів та їхніх характеристик.

Після визначення вимог користувача дизайнери можуть створювати каркаси та макети для візуалізації дизайну інтерфейсу користувача. Каркаси — це базові ескізи, які окреслюють макет і функціональність інтерфейсу, тоді як макети — це більш детальні представлення, які включають візуальні елементи, такі як кольори, типографіка та зображення.

Каркаси та макети дозволяють дизайнерам тестувати різні концепції дизайну та збирати відгуки від зацікавлених сторін, перш ніж вкладати час і ресурси в розробку остаточного дизайну. Вони також допомагають переконатися, що інтерфейс користувача відповідає вимогам користувача, є інтуїтивно зрозумілим і простим у використанні.

Після затвердження каркасів і макетів остаточний дизайн можна реалізувати за допомогою HTML, CSS і JavaScript. HTML використовується

для структурування вмісту інтерфейсу користувача, тоді як CSS використовується для стилізації інтерфейсу користувача та надання йому візуальної привабливості. JavaScript використовується для додавання інтерактивності та динамічних функцій інтерфейсу користувача, таких як спадні меню та поля для пошуку.

На етапі впровадження дизайнери також повинні переконатися, що інтерфейс користувача адаптований до різних розмірів екрана та пристроїв. Це особливо важливо для електронної бібліотеки, оскільки користувачі можуть отримати доступ до системи з різних пристроїв.

Після впровадження дизайну інтерфейсу користувача його необхідно перевірити, щоб переконатися, що він відповідає вимогам користувача та функціонує належним чином. Тестування може включати тестування зручності використання, коли користувачам пропонується виконувати завдання за допомогою інтерфейсу користувача та надавати відгуки про свій досвід. Цей відгук можна використати для подальшого вдосконалення дизайну інтерфейсу користувача.

Розробка інтерфейсу користувача — це повторюваний процес, і дизайнерам може знадобитися внести кілька раундів редагування на основі відгуків користувачів і результатів тестування. Важливо розставляти пріоритети потреб і вподобань користувачів протягом усього процесу проектування, щоб переконатися, що остаточний дизайн інтерфейсу користувача є інтуїтивно зрозумілим, зручним і відповідає цілям електронної бібліотеки.

2.4 Розробка серверної частини

Технологія клієнт – сервер, яка широко застосовується при роботі з базами даних в мережі, відома вже давно і найчастіше застосовувалась у великих організаціях. Сьогодні, з розвитком INTERNET, ця технологія все частіше приваблює погляди розробників програмного забезпечення, оскільки в світі

нагромаджено величезну кількість інформації по різноманітних питаннях і найчастіше ця інформація зберігається в базах даних [16-17].

Архітектура мережі визначає основні елементи мережі, характеризує її загальну логічну організацію, технічне забезпечення, програмне забезпечення, описує методи кодування. Архітектура також визначає принципи функціонування та інтерфейс користувача.

Архітектура клієнт – сервер

Архітектура клієнт – сервер (client-server architecture) – це концепція інформаційної мережі, в якій основна частина її ресурсів зосереджена в серверах, обслуговуючих своїх клієнтів (рис. 2.3). Розглянута архітектура визначає два типи компонентів: сервери і клієнти.

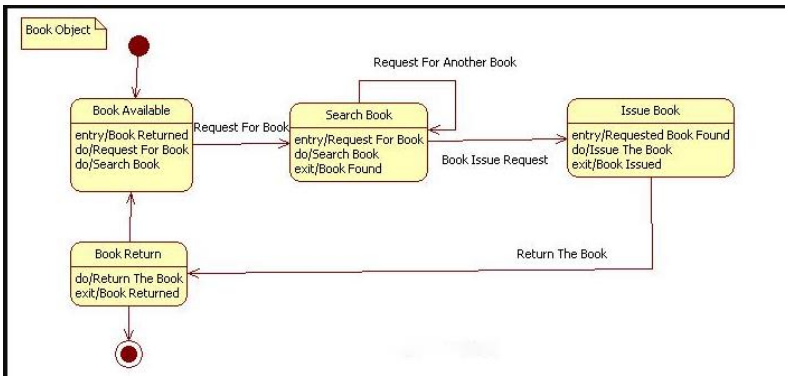

Рисунок 2.3 – Діаграма станів розробленого додатку

Сервер – це об’єкт, що дає сервіс іншим об’єктам мережі за їх запитами. Сервіс – це процес обслуговування клієнтів.

Сервер працює за завданнями клієнтів і управляє виконанням їх завдань. Після виконання кожного завдання сервер посилає отримані результати клієнту, який послав це завдання.

Сервісна функція в архітектурі клієнт – сервер описується комплексом прикладних програм, відповідно до якого виконуються різноманітні прикладні процеси.

Процес, який викликає сервісну функцію за допомогою певних операцій, називається клієнтом. Ним може бути програма або користувач. На (рис. 2.4) наведений перелік сервісів в архітектурі клієнт – сервер.

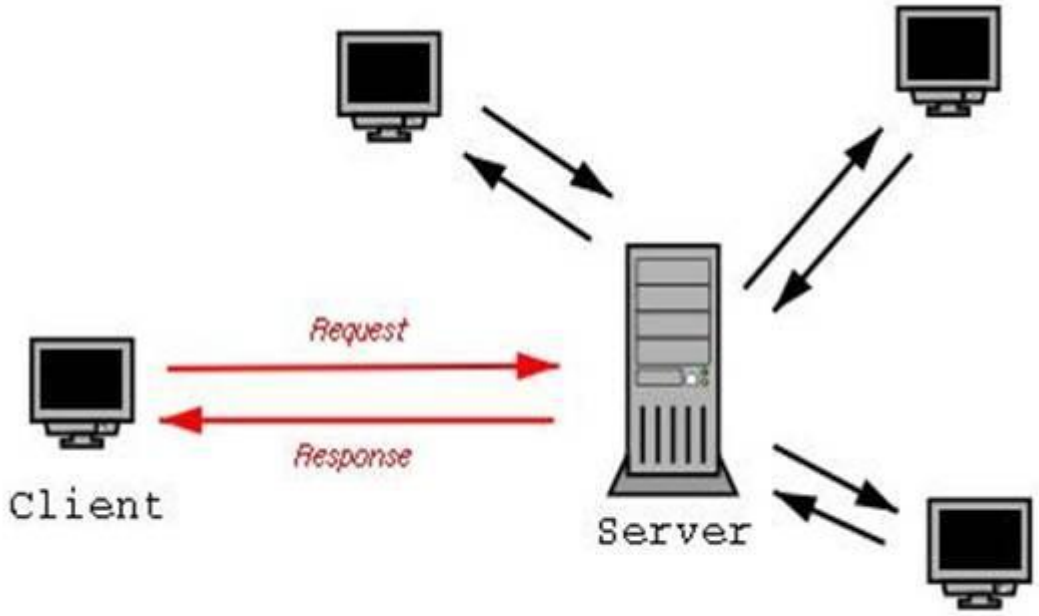

Рисунок 2.4 – Модель архітектури Clients-Server

Клієнти – це робочі станції, які використовують ресурси сервера і надають зручні інтерфейси користувача. Інтерфейси користувача це процедури взаємодії користувача з системою або мережею.

Клієнт є ініціатором і використовує електронну пошту або інші сервіси сервера. У цьому процесі клієнт запитує вид обслуговування, встановлює сеанс, отримує потрібні йому результати і повідомляє про закінчення роботи.

У мережах з виділеним файловим сервером на виділеному автономному ПК встановлюється серверна мережева операційна система. Цей ПК стає сервером. Програмне забезпечення (ПЗ), встановлене на робочій станції, дозволяє їй обмінюватися даними з сервером. Найбільш поширені мережеві операційна системи:

– NetWare фірми Novel;

– Windows NT фірми Microsoft;

– UNIX фірми AT &T;

– Linux.

Крім мережевої операційної системи необхідні мережні прикладні програми, що реалізують переваги, надані мережею.

Мережі на базі серверів мають кращі характеристики і підвищену надійність. Сервер володіє головними ресурсами мережі, до яких звертаються інші робочі станції.

У сучасній клієнт – серверній архітектурі виділяється чотири групи об’єктів: клієнти, сервери, дані і мережеві служби.Клієнти розташовуються в системах на робочих місцях користувачів. Дані в основному зберігаються в серверах. Мережеві служби є спільно використовуваними серверами і даними. Крім того служби керують процедурами обробки даних.

Мережі клієнт – серверної архітектури мають наступні переваги:

– дозволяють правильно організовувати мережі з великою кількістю робочих станцій;

– забезпечують централізоване управління обліковими записами користувачів, безпекою та доступом, що спрощує мережне адміністрування;

– ефективний доступ до мережевих ресурсів;

– користувачеві потрібен один пароль для входу в мережу і для отримання доступу до всіх ресурсів, на які поширюються права користувача.

Поряд з перевагами мережі клієнт – серверної архітектури мають і ряд недоліків:

– несправність сервера може зробити мережу непрацездатною, як мінімум втрату мережевих ресурсів;

– вимагають кваліфікованого персоналу для адміністрування;

– мають вищу вартість мереж і мережевого обладнання.

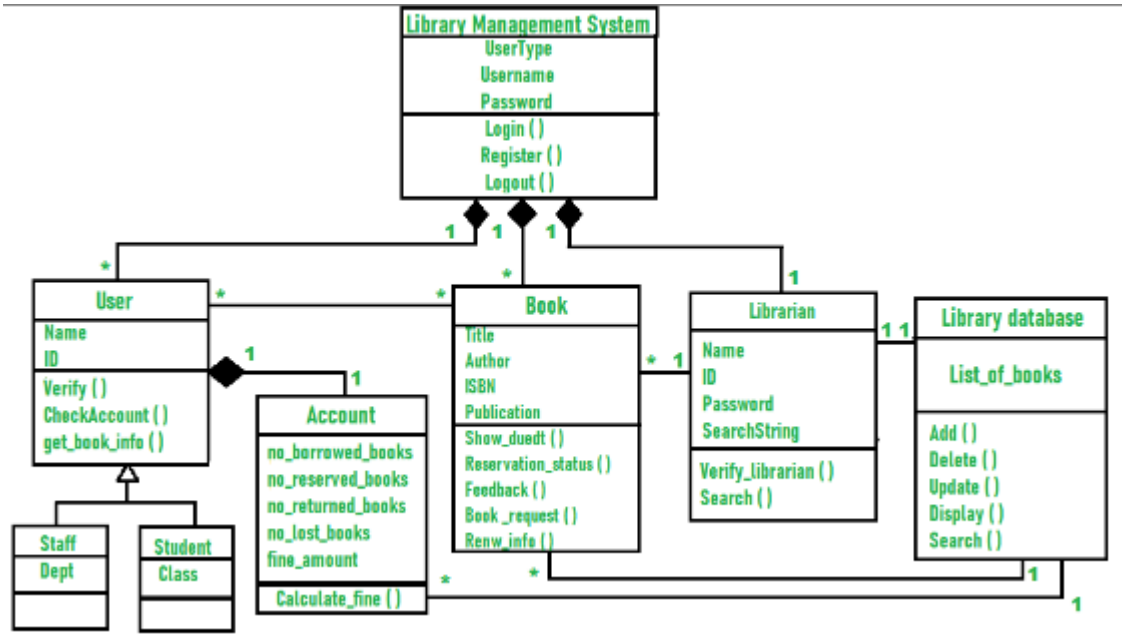

Рисунок 2.5 – Діаграма класів додатку

Висновоки до розділу 2

Розділ присвячений складанню та опису необхідних функціональних вимог до бібліотеки. Спроектовано архітектуру бази даних, таблиці, поля та зв’язки між ними.

Наступним пунктом є розробка інтерфейсу користувача в кінці якої представлена діаграма стані розробленого додатку. Після розробки інтерфейсу іде розробка серверної частини додатку описаними раніше мовами.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ЕЛЕКТРОННОЇ БІБЛІОТЕКИ НА МОВІ ПРОГРАМУВАННЯ PYTHON

3.1 Опис структури та функціональних можливостей реалізованої електронної бібліотеки

Впроваджена електронна бібліотека покликана надати користувачам комплексну та зручну платформу для доступу та управління цифровими книгами та ресурсами. Вона пропонує широкий спектр функціональних можливостей, спрямованих на покращення користувацького досвіду та полегшення ефективного пошуку, читання та управління книгами. У цьому розділі ми надамо огляд структури та ключових функціональних можливостей реалізованої електронної бібліотеки.

Реєстрація та аутентифікація користувачів: Електронна бібліотека дозволяє користувачам створювати облікові записи та аутентифікуватися для доступу до персоналізованих функцій. Користувачі можуть реєструватися, використовуючи свої адреси електронної пошти або акаунти в соціальних мережах, а їхні облікові дані надійно зберігаються і управляються.

Профілі користувачів: Кожен зареєстрований користувач має персоналізований профіль, який включає таку інформацію, як ім'я, фотографію профілю, уподобання щодо читання та історію читання. Профіль дозволяє користувачам налаштовувати свій читацький досвід і відстежувати свою взаємодію з бібліотекою.

Книжковий каталог (рис 3.1): бібліотека має всеосяжний книжковий каталог, який охоплює величезну колекцію цифрових книг різних жанрів, тематик і мов. Каталог надає розширені можливості пошуку та фільтрації, щоб допомогти користувачам знаходити книги відповідно до їхніх інтересів, авторів, назв або ключових слів.

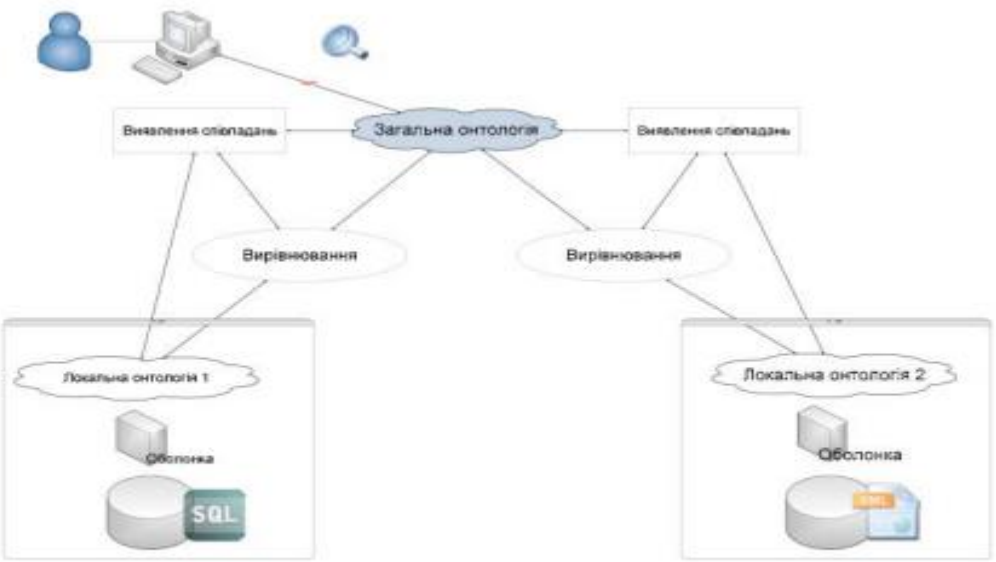

Рисунок 3.1 – Робота з каталогом додатку

Детальна інформація про книгу та попередній перегляд: Кожна книга в каталозі має окрему сторінку, на якій представлена детальна інформація, включаючи назву, автора, дату публікації, опис та зображення обкладинки. Користувачі можуть переглянути прев'ю книги, наприклад, уривки або зразки розділів, щоб оцінити його зміст, перш ніж прийняти певне рішення про читання або позику.

Читання книг: Електронна бібліотека пропонує зручний інтерфейс для читання книг, який дозволяє користувачам читати цифрові книги безпосередньо на платформі. Інтерфейс для читання надає такі можливості, як налаштування стилів шрифту, розмір шрифту, закладки сторінок, виділення та можливість робити нотатки, щоб покращити досвід читання.

Запозичення та позика: Користувачі можуть брати книги в бібліотеці на певний час, а система відстежує історію користування, щоб забезпечити добросовісне використання. Крім того, бібліотека може впровадити функцію позички, яка дозволяє користувачам ділитися власними цифровими книгами з іншими користувачами, сприяючи створенню спільної читацької спільноти.

Книжкові рекомендації: Система використовує вподобання користувачів, історію читання та метадані книг для надання персоналізованих рекомендацій. Ці рекомендації допомагають користувачам знаходити нові книги, які відповідають їхнім інтересам і читацьким звичкам, покращуючи їхній загальний досвід читання.

Соціальна взаємодія: Електронна бібліотека може включати соціальні функції, які дозволяють користувачам взаємодіяти один з одним. Користувачі можуть ділитися книжковими рекомендаціями, оцінювати і рецензувати книги, приєднуватися до читацьких груп або спільнот, а також брати участь в обговореннях, пов'язаних з книгами або конкретними темами.

Робота складається з таки модулів library/

```
├── app.py
├── models.py
├── routes.py
├── utils.py
├── templates/
│   ├── index.html
│   └── book.html
└── static/
    └── style.css
```

app.py: Головний модуль, який створює та налаштовує Flask додаток. Він містить код для запуску сервера.

models.py: Модуль, який містить класи моделей для представлення даних електронної бібліотеки. Наприклад, клас Book має властивості, такі як id, title, author, description, cover_image і т.д.

routes.py: Модуль, який містить маршрути (роути) для обробки HTTP запитів. Має маршрути для відображення списку книжок, окремої книжки, додавання нової книжки, видалення книжки тощо. Кожен маршрут пов'язаний зі своєю функцією-обробником, яка виконує потрібні дії.

utils.py: Модуль, який містить допоміжні функції, які використовуватись в інших модулях. Наприклад, ви можете мати функцію для збереження зображень обкладинок книжок або функцію для валідації даних перед збереженням у базу даних.

templates/: Директорія, яка містить HTML шаблони для відображення сторінок електронної бібліотеки. Наприклад, index.html може містити список книжок, а book.html - детальну інформацію про окрему книжку.
static/: Директорія, яка містить статичні файли, такі як CSS (style.css). Ви можете використовувати CSS для стилізації сторінок вашої електронної бібліотеки.

Основні функціональні можливості реалізованої електронної бібліотеки включають:

- відображення списку книжок і окремої книжки;
- додавання нових книжок до бази даних;
- видалення книжок з бази даних;
- завантаження обкладинок книжок і збереження їх у файловій системі або хмарному сховищі;
- пошук книжок за заголовком, автором або іншими параметрами;
- редагування і оновлення інформації про книжку;
- аутентифікація та авторизація користувачів (якщо потрібно);
- інтеграція з зовнішніми сервісами, наприклад, для отримання додаткової інформації про книжки (ISBN, рейтинги тощо).

Управління книжками (рис 3.2): Бібліотека надає користувачам інструменти для управління їхніми особистими книжковими колекціями. Користувачі можуть створювати віртуальні книжкові полиці, класифікувати книги, додавати теги та організовувати книги на основі їхнього статусу читання (прочитані, ті, що зараз читаються, ті, що треба прочитати). Ця функціональність допомагає користувачам відстежувати прогрес у читанні та підтримувати впорядковану віртуальну бібліотеку.

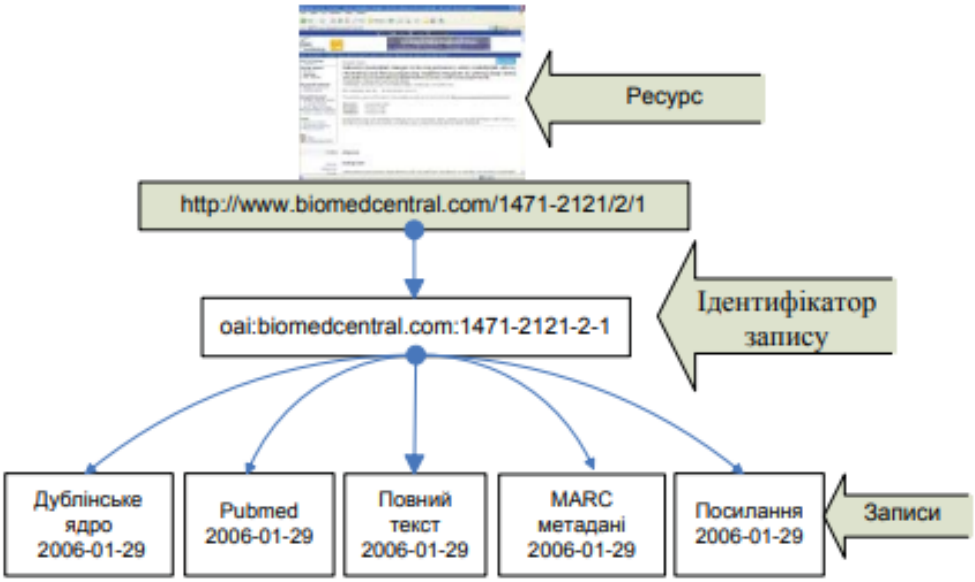

Рисунок 3.2 – Схмема отримання інформації

Доступність та сумісність з пристроями: Електронна бібліотека розроблена таким чином, щоб бути доступною на різних пристроях, включаючи стаціонарні комп'ютери, ноутбуки, планшети і смартфони. Вона забезпечує чуйний і оптимізований користувальницький інтерфейс, який адаптується до різних розмірів екрану і роздільної здатності, пропонуючи послідовний досвід роботи на різних пристроях.

Впроваджена електронна бібліотека поєднує в собі добре структуровану архітектуру, інтуїтивно зрозумілий інтерфейс та широкий спектр функціональних можливостей, щоб забезпечити захоплюючий та зручний досвід читання для користувачів. Завдяки всеосяжному каталогу книг, персоналізованим функціям та елементам соціальної взаємодії, електронна бібліотека має на меті сприяти створенню активної читацької онлайн-спільноти, задовольняючи різноманітні потреби та вподобання своїх користувачів.

3.2 Опис особливостей реалізації серверної частини

Реалізація електронної бібліотечної системи на стороні сервера охоплює кілька ключових особливостей і міркувань для забезпечення безперебійної

роботи та ефективної обробки запитів користувачів. Цей розділ містить огляд специфічних особливостей та основних моментів реалізації на стороні сервера.

Масштабована архітектура сервера: Реалізація на стороні сервера використовує масштабовану архітектуру, щоб пристосуватися до зростаючої бази користувачів і обробляти збільшений трафік. Вона використовує такі технології, як балансувальники навантаження, механізми кешування та методи горизонтального масштабування для розподілу робочого навантаження між декількома серверами, забезпечуючи оптимальну продуктивність та масштабованість.

RESTful API дизайн: Реалізація на стороні сервера відповідає архітектурному стилю RESTful, де ресурси доступні через чітко визначені кінцеві точки. Це дозволяє клієнтам взаємодіяти з сервером за допомогою стандартних методів HTTP, таких як GET, POST, PUT та DELETE. Кінцеві точки API розроблені так, щоб бути інтуїтивно зрозумілими і відповідати найкращим практикам для ефективного пошуку та маніпулювання даними.

Заходи безпеки: Реалізація на стороні сервера включає в себе надійні заходи безпеки для захисту даних користувача та запобігання несанкціонованому доступу. Використовуються протоколи захищеного зв'язку (HTTPS) для шифрування передачі даних, механізми автентифікації та авторизації користувачів, а також безпечні методи кодування для зменшення поширених вразливостей безпеки, таких як SQL-ін'єкції або міжсайтовий скриптинг (XSS).

Управління базами даних: Реалізація на стороні сервера використовує систему управління базами даних (СУБД) для ефективного зберігання та отримання даних. Вона використовує реляційні моделі баз даних для забезпечення цілісності даних і застосовує передові методи індексування та кешування для оптимізації продуктивності запитів. Реалізація дотримується принципів нормалізації баз даних для мінімізації надмірності та покращення загального управління даними.

Кешування та оптимізація продуктивності: Для підвищення продуктивності та зменшення навантаження на сервер, реалізація на стороні сервера включає механізми кешування. Вона використовує рішення для кешування в пам'яті, такі як Redis або Memcached, для кешування даних, до яких часто звертаються, зменшуючи потребу в повторюваних запитах до бази даних. Крім того, для підвищення загальної продуктивності застосовуються методи оптимізації запитів, такі як індексування та аналіз плану запитів.

Асинхронна обробка: Реалізація на стороні сервера використовує методи асинхронної обробки для ефективного виконання трудомістких завдань. Довготривалі операції, такі як створення прев'ю книг або виконання складних запитів до бази даних, вивантажуються у фонові процеси або робочі потоки, щоб уникнути блокування сервера та підтримувати швидкість реакції.

Обробка помилок та ведення журналів: Реалізація на стороні сервера включає в себе надійні механізми обробки помилок та ведення журналів. Застосовуються методи обробки винятків, що дозволяють ефективно обробляти помилки та винятки, гарантуючи стабільну та відмовостійку роботу сервера. Детальні журнали генеруються для фіксації відповідної інформації, що допомагає в усуненні несправностей та обслуговуванні системи.

Інтеграція зі сторонніми сервісами: Реалізація на стороні сервера може інтегруватися із зовнішніми сервісами для розширення функціональності. Наприклад, вона може інтегруватися з платіжними шлюзами для управління підписками або інтегруватися зі сторонніми API для книжкових рекомендацій чи пошуку метаданих. При взаємодії з цими зовнішніми сервісами застосовуються належні заходи автентифікації та безпеки.

Моніторинг та аналітика: Реалізація на стороні сервера включає в себе інструменти моніторингу та аналітики для збору інформації про продуктивність системи, шаблони використання та поведінку користувачів. Такі показники, як час відгуку, рівень помилок і використання ресурсів, збираються для виявлення вузьких місць і оптимізації продуктивності системи. Користувацька аналітика

може допомогти покращити персоналізацію та вдосконалити алгоритми рекомендацій.

Розгортання та практики DevOps: Реалізація на стороні сервера відповідає кращим практикам розгортання і використовує принципи DevOps. Для автоматизації процесів збірки, тестування та розгортання створюються конвеєри безперервної інтеграції та розгортання (CI/CD). Технології контейнеризації, такі як Docker, можуть бути використані для забезпечення узгодженості та переносимості в різних середовищах.

Серверна реалізація електронної бібліотечної системи включає в себе такі функції, як масштабована архітектура, RESTful API, заходи безпеки, ефективне управління базами даних, механізми кешування, асинхронна обробка, обробка помилок, інтеграція зі сторонніми сервісами, моніторинг та найкращі практики розгортання. Ці функції в сукупності сприяють створенню надійної та високопродуктивної інфраструктури на стороні сервера, яка підтримує безперебійну роботу та оптимальний користувацький досвід електронної бібліотечної системи.

REST (рис 3.3) – це такий принцип, що включає в себе набір архітектурних підходів для створення гнучкої мережі, важливо розуміти, що це не протокол, а саме концепція. Він займається взаємодією компонентів в мережі між собою і надає доступ до мережевих ресурсів. Також кожен REST протокол не має залежати від мережевого рівня, не має зберігати проміжної інформації про запити і відповіді.

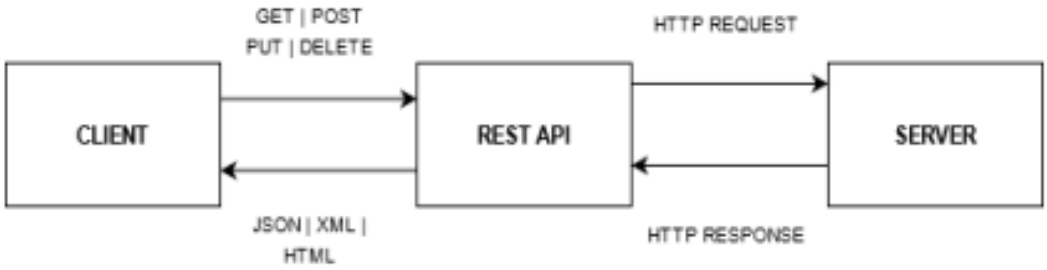

Рисунок 3.3 – Приклад REST

Завдяки цьому можна будувати продуктивні, завдяки використанню кешування, масштабовані і надійні системи, через те, що можна просто змінювати і підтримувати їх до нових технологій, та нових вимог до сервісів.

Даний принцип був створений Роєм Філіднгом, він же і займався створенням протоколу HTTP. Дані у REST мають передаватися невеликими порціями в різних форматах, таких як HTML, XML, JSON. REST протоколи мають підтримувати кешування.

Основні особливості REST архітектури:

1. Продуктивність - взаємодія компонентів системи може бути домінуючим фактором продуктивності і ефективності мережі з точки зору користувача.

2. Масштабованість, яка забезпечує велику кількость компонентів і взаємодій компонентів.

3. Простий узагальнений інтерфейс, який вказує яка має бути ідентифікація ресурсів, якого вигляду мають бути повідомлення.

4. Компоненти архітектури відкриті до змін і покращення для задоволення потреб користувачів. Можна вводити зміни навіть при працюючій на цьому принципу системі.

5. Зв’язки між компонентами зрозумілі і прості.

6. Компоненти можна просто переносити з одного місця в інше разом з програмним кодом.

7. Стійкість до відмов на рівні системи, коли з ладу виходить якась одна складова.

Вимоги, того, аби називати додаток REST-системою:

1. Архітектура клієнт-сервер.

2. Відсутність стану, тобто не потрібно зберігати стан клієнта на сервері, це називається протоколом без зберігання стану.

3. Присутність кешування.

4. Єдиний узагальнений інтерфейс, що дозволяє кожному сервісу розвиватися самостійно.

5. Система має бути поділена на шари, кожен з яких займається своєю логікою.

Окрім REST існує також SOAP . Це вже протокол обміну повідомленнями та віддаленого виклику процедур (RPC). RPC – це клас технологій, які дозволяють викликати функції з інших систем, що можуть знаходитися навіть на іншій машині. Зазвичай до складу такої технології входить мережевий протокол для взаємодії клієнта та сервера і мові серіалізації об’єктів. Як правило такі технології є реалізаціями архітектур CORBA або SOA, з протоколів використовується UDP, TCP та HTTP.

Головною відмінністю є те, що SOAP – це протокол і він оперує тільки XML, в свою чергу REST – це архітектурний стиль, який може оперувати будь-яким форматом даних. Зазвичай SOAP використовують в складних та навантажених системах, де потрібно дотримуватися строгого формату даних, де їм потрібний додатковий шар безпеки. SOAP може використовувати будь-які протоколи прикладного рівня:

HTTP, HTTPS, FTP, SMTP, проте немає узагальненої роботи з усіма, під кожен протокол потрібно вносити певні зміни. SOAP також використовує WSDL для опису веб-сервісів і доступу до них на основі XML.

Звичайне повідомлення SOAP складається з наступних елементів:

1. Envelope – головний батьківський елемент, який визначає простір імен, яке буде використувуватися в документі.
2. Header – містить атрибути повідомлення, такі як інформація про безпеку чи маршрутизацію в мережі.
3. Body – містить повідомлення, якими обмінюються складові системи, в якій використовується протокол.
4. Fault – необов’язковий елемент, який надає інформацію про помилки, які відбулися при обробці повідомлення:

```
from flask import Flask, request, jsonify
app = Flask(__name__)

# Це простий приклад бази даних, який можна використовувати для зберігання книжок
books = [
```

```
    {
        'id': 1,
        'title': 'Програмування на Python',
        'author': 'John Doe'
    },
    {
        'id': 2,
        'title': 'Алгоритми і структури даних',
        'author': 'Jane Smith'
    }
]

# Маршрут для отримання списку всіх книжок
@app.route('/books', methods=['GET'])
def get_books():
    return jsonify(books)

# Маршрут для отримання окремої книжки за ідентифікатором
@app.route('/books/<int:book_id>', methods=['GET'])
def get_book(book_id):
    book = next((book for book in books if book['id'] == book_id), None)
    if book:
        return jsonify(book)
    else:
        return jsonify({'error': 'Книжка не знайдена'}), 404

# Маршрут для додавання нової книжки
@app.route('/books', methods=['POST'])
def add_book():
    new_book = {
        'id': len(books) + 1,
        'title': request.json['title'],
        'author': request.json['author']
    }
    books.append(new_book)
    return jsonify(new_book), 201

# Маршрут для видалення книжки за ідентифікатором
@app.route('/books/<int:book_id>', methods=['DELETE'])
def delete_book(book_id):
    book = next((book for book in books if book['id'] == book_id), None)
    if book:
        books.remove(book)
        return jsonify({'message': 'Книжка успішно видалена'})
    else:
        return jsonify({'error': 'Книжка не знайдена'}), 404

if __name__ == '__main__':
    app.run()
```

Данний приклад було знайдено та доповнено відповідно до умов даного завдання.

Цей приклад використовує фреймворк Flask для створення серверної частини. Він містить чотири маршрути:

/books (GET): Повертає список всіх книжок у форматі JSON.

/books/<book_id> (GET): Повертає окрему книжку за її ідентифікатором у форматі JSON.

/books (POST): Додає нову книжку до бази даних. Дані книжки передаються у форматі JSON у тілі запиту.

/books/<book_id> (DELETE): Видаляє книжку за її ідентифікатором.

3.3 Розгляд можливостей розширення та модифікації електронної бібліотеки

Проектування та впровадження електронної бібліотечної системи було здійснено з урахуванням можливості розширення та кастомізації. Система розроблена таким чином, щоб дозволити легку інтеграцію нових функцій, модулів і вдосконалень, а також налаштування відповідно до конкретних вимог або уподобань. У цьому розділі ми розглянемо різні можливості та міркування щодо розширення та модифікації системи електронної бібліотеки.

Архітектура плагінів: Система електронної бібліотеки може бути розроблена з архітектурою плагінів, яка дозволяє розробникам створювати та інтегрувати власні плагіни. Плагіни можуть додавати нові функціональні можливості, розширювати існуючі функції або забезпечувати інтеграцію із зовнішніми системами. Визначивши чіткі інтерфейси плагінів і використовуючи ін'єкцію залежностей, розробники можуть легко створювати та інтегрувати кастомні модулі в систему.

Розширення API: Електронна бібліотечна система може надавати API, який дозволяє розробникам розширювати її функціональність за допомогою користувацьких кінцевих точок API. Ці розширення можна використовувати для інтеграції із зовнішніми сервісами, створення власних робочих процесів або надання додаткових можливостей пошуку даних чи маніпулювання ними. API може бути гнучким і модульним, що дозволяє розробникам додавати власні кінцеві точки та логіку.

Тематизація та кастомізація: Електронна бібліотечна система може підтримувати варіанти оформлення та кастомізації, щоб задовольнити різні візуальні вподобання та вимоги до брендингу. Користувачі або адміністратори можуть налаштовувати зовнішній вигляд інтерфейсу користувача, вибираючи попередньо визначені теми або створюючи власні теми. Така гнучкість дозволяє адаптувати систему до різних організацій або індивідуальних уподобань користувачів.

Модульні компоненти: Система може бути розроблена з використанням модульних компонентів, які можна легко замінити або розширити. Наприклад, функція пошуку книг може бути реалізована як окремий модуль, який за бажанням можна замінити на іншу реалізацію пошукової системи. Такий модульний підхід дозволяє легко модифікувати або замінювати окремі компоненти, не впливаючи на загальну архітектуру системи.

Подієво-орієнтована архітектура: Електронна бібліотечна система може використовувати архітектуру, керовану подіями, що дозволяє вільно підключати та легко інтегрувати нові функціональні можливості. Події можуть ініціюватися в різних точках системи, і розробники можуть підписуватися на ці події, щоб виконувати власну логіку або виконувати додаткові операції. Цей підхід, заснований на подіях, дозволяє легко інтегрувати нові функції без жорсткої прив'язки до основної системи.

Користувацькі метадані та таксономії: Система може надавати механізми для визначення полів метаданих і таксономій та управління ними. Це дозволяє створювати спеціалізовані поля метаданих для збору додаткової інформації про книгу або специфічних для користувача атрибутів. Адміністратори можуть визначати власні таксономії для категоризації книг на основі певних критеріїв, покращуючи організацію та доступність контенту.

Налаштування робочого процесу: Електронна бібліотечна система може запропонувати варіанти для налаштування робочих процесів і процесів. Адміністратори можуть визначати власні робочі процеси для придбання книг, каталогізації чи затвердження, пристосовуючи їх до конкретних потреб

організації. Така можливість кастомізації гарантує, що система може адаптуватися до різних бізнес-правил і вимог.

Інтеграція із зовнішніми системами: Система може бути розроблена для інтеграції із зовнішніми системами, такими як платіжні шлюзи, постачальники контенту або аналітичні платформи. Така інтеграція забезпечує безперешкодний обмін даними та інтероперабельність між електронною бібліотечною системою та іншими системами, розширюючи її функціональність та покращуючи користувацький досвід.

Постійне вдосконалення та зворотній зв'язок: Електронна бібліотечна система може включати механізми для збору відгуків користувачів та пропозицій щодо вдосконалення. Відгуки користувачів можна збирати за допомогою опитувань, форм зворотного зв'язку або користувацьких форумів, і ця інформація може бути використана для подальшої розробки, вдосконалення функцій або можливостей налаштування. Такий ітеративний підхід гарантує, що система розвивається і адаптується до мінливих потреб користувачів.

Документація та ресурси для розробників: Щоб полегшити розширення та модифікацію електронної бібліотечної системи, слід надати вичерпну документацію та ресурси для розробників. Сюди входить документація до API, посібники з розробки плагінів, приклади коду та найкращі практики. Пропонуючи ці ресурси, розробники можуть легко зрозуміти архітектуру системи, інтерфейси та можливості налаштування, що дозволить їм ефективно розширювати та модифікувати електронну бібліотечну систему.

Версії та сумісність: Система повинна підтримувати належні механізми керування версіями та сумісності для забезпечення плавного переходу під час оновлення або модифікації. Нові функції або вдосконалення слід впроваджувати у зворотньо-сумісний спосіб, щоб існуючі налаштування та розширення продовжували функціонувати без проблем. Необхідно надати чіткі настанови щодо версій та примітки до випусків, щоб допомогти розробникам зрозуміти вплив системних змін на їхні кастомізації.

Спільнота та співпраця: Заохочення активної спільноти навколо електронної бібліотечної системи може сприяти співпраці та подальшому розширенню її можливостей. Надання платформи для розробників для обміну своїми налаштуваннями, плагінами чи інтеграціями сприяє колективному навчанню та інноваціям. Форуми користувачів, спільноти розробників або спеціальні канали можуть слугувати центрами для дискусій, обміну знаннями та співпраці.

Тестування та забезпечення якості: Розширюючи або модифікуючи електронну бібліотечну систему, важливо дотримуватися суворих практик тестування і забезпечення якості. Ретельному тестуванню підлягають кастомізації, плагіни та інтеграції, щоб забезпечити їхню сумісність, стабільність і продуктивність. Тестові кейси повинні охоплювати різні сценарії, включно з граничними ситуаціями та обробкою помилок, щоб перевірити стійкість і надійність налаштувань.

Шляхи оновлення та стратегії міграції: З розвитком електронної бібліотечної системи важливо забезпечити чіткі шляхи оновлення та стратегії міграції для існуючих налаштувань. Необхідно надати належну документацію та інструкції, щоб допомогти розробникам перенести свої налаштування на новіші версії системи. Перевірка сумісності, автоматизовані сценарії міграції або рівні зворотної сумісності можуть спростити процес переходу та мінімізувати збої в роботі.

Оптимізація продуктивності та масштабованість: При розширенні або модифікації системи слід враховувати оптимізацію продуктивності та масштабованість. Налаштування повинні бути розроблені з урахуванням ефективності, мінімізації споживання ресурсів та оптимізації запитів до бази даних або мережі. Необхідно враховувати потенційні проблеми масштабування, гарантуючи, що налаштування можуть ефективно масштабуватися в міру зростання користувацької бази або обсягу даних.

Таким чином, електронна бібліотечна система пропонує різні шляхи для розширення та налаштування її функціональності відповідно до конкретних

вимог. Завдяки архітектурі плагінів, розширенню API, опціям тем, модульним компонентам, архітектурі, керованій подіями, а також кастомізації робочих процесів і метаданих, система забезпечує гнучкість і адаптивність. Інтеграція із зовнішніми системами, постійне вдосконалення на основі відгуків користувачів та вичерпна документація ще більше розширюють можливості розширення та кастомізації. Дотримуючись найкращих практик, підтримуючи сумісність і розвиваючи спільноту, що співпрацює, електронна бібліотечна система може продовжувати розвиватися і задовольняти різноманітні потреби своїх користувачів.

Висновки до розділу 3

Третій розді почався з опису структури та функціональних можливостей реалізованої бібліотеки. Описано процес розробки та особливості серверної частини. Перевагою бібліотеки є те, що вона залишає за собою можливість розширення та добавленні нових модифікацій.

РОЗДІЛ 4. ТЕСТУВАННЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ РОЗРОБЛЕНОЇ ЕЛЕКТРОННОЇ БІБЛІОТЕКИ

4.1 Опис методів тестування

Тестування є важливим аспектом розробки програмного забезпечення, в тому числі й електронної бібліотечної системи. Воно слугує важливим цілям і відіграє вирішальну роль у забезпеченні якості, надійності та ефективності системи. У цьому розділі ми розглянемо причини, чому тестування необхідне і чому воно має бути невід'ємною частиною процесу розробки.

Перевірка функціональності: Тестування дозволяє перевірити, чи функціонує електронна бібліотечна система так, як було задумано, і чи відповідає вона заданим вимогам. Виконуючи тестові кейси та сценарії, ми можемо переконатися, що всі функції, можливості та взаємодії в системі працюють правильно. Це допомагає виявити будь-які дефекти або помилки на ранній стадії процесу розробки, що дозволяє оперативно вирішувати проблеми та запобігати їх потраплянню до кінцевих користувачів [18-19].

Виявлення дефектів: Тестування допомагає ідентифікувати та виявити дефекти або помилки в електронній бібліотечній системі. Виконуючи різні тести, ми можемо виявити проблеми, пов'язані з функціональністю, продуктивністю, безпекою, зручністю використання або сумісністю. Чим раніше ці дефекти будуть виявлені, тим легше і дешевше їх виправити. Тестування дає можливість вирішити ці проблеми до розгортання системи, зменшуючи ймовірність незадоволення клієнтів і негативного впливу на користувацький досвід.

Підвищення якості: Тестування відіграє важливу роль у підвищенні загальної якості електронної бібліотечної системи. Ретельно оцінюючи систему за допомогою різних методів тестування, ми можемо гарантувати, що вона відповідає стандартам якості та забезпечує надійний і стабільний користувацький досвід. Тестування допомагає виявити недоліки дизайну,

помилки реалізації або прогалини у функціоналі, що дозволяє розробникам виправити їх і підвищити якість, стабільність і продуктивність системи.

Перевірка вимог: Тестування слугує засобом підтвердження того, що електронна бібліотечна система відповідає визначеним вимогам. Воно дозволяє порівняти фактичну поведінку системи з очікуваною поведінкою, визначеною в документації з вимогами. Завдяки систематичному тестуванню ми можемо переконатися, що всі функціональні та нефункціональні вимоги виконані, гарантуючи, що система відповідає своєму призначенню і задовольняє потреби користувачів.

Забезпечення задоволеності користувачів: Тестування сприяє задоволенню користувачів, гарантуючи, що електронна бібліотечна система функціонує, як очікується, добре працює і забезпечує безперебійну роботу користувачів. Виявляючи і вирішуючи проблеми за допомогою тестування, ми можемо звести до мінімуму виникнення системних збоїв, помилок або несподіваної поведінки, які можуть розчарувати або створити незручності для користувачів. Ретельне тестування допомагає зміцнити впевненість, довіру та задоволеність користувачів системою.

Зниження ризиків: Тестування допомагає зменшити ризики, пов'язані з розробкою та розгортанням електронної бібліотечної системи. Виявляючи і вирішуючи потенційні проблеми на ранній стадії життєвого циклу розробки, ми можемо зменшити ймовірність збоїв системи, втрати даних, порушень безпеки або вузьких місць у продуктивності. Тестування дозволяє оцінювати ризики, гарантуючи, що система працює надійно, безпечно та ефективно.

Відповідність стандартам і правилам: Тестування допомагає переконатися, що електронна бібліотечна система відповідає відповідним галузевим стандартам, нормам і найкращим практикам. Воно дозволяє перевірити дотримання протоколів безпеки, інструкцій з доступності, правил захисту даних або вимог до інтероперабельності. Тестування забезпечує гарантію того, що система відповідає необхідним стандартам відповідності, захищаючи конфіденційність, безпеку та юридичні зобов'язання користувачів.

Постійне вдосконалення: Тестування - це ітеративний процес, який сприяє постійному вдосконаленню системи електронної бібліотеки. Завдяки регулярним циклам тестування ми можемо збирати відгуки, визначати сфери для вдосконалення та враховувати пропозиції користувачів. Тестування підтримує безперервний цикл зворотного зв'язку, що дозволяє впроваджувати оновлення, виправляти помилки і додавати нові функції на основі реального використання та відгуків користувачів.

Тестування є невід'ємною частиною процесу розробки електронної бібліотечної системи. Воно передбачає перевірку функціональності, продуктивності та надійності системи, щоб переконатися, що вона відповідає бажаним вимогам. У цьому розділі ми обговоримо різні методи тестування, які можуть бути використані під час розробки та оцінки електронної бібліотеки.

Модульне тестування: Модульне тестування фокусується на тестуванні окремих компонентів або одиниць коду в ізоляції. У випадку електронної бібліотечної системи це може включати тестування функцій, класів або модулів, щоб переконатися, що вони видають очікуваний результат і правильно обробляють крайні випадки. Модульні тести, як правило, автоматизовані і допомагають виявити помилки або проблеми на ранній стадії процесу розробки.

Інтеграційне тестування: Інтеграційне тестування має на меті перевірити взаємодію та сумісність між різними компонентами або модулями електронної бібліотечної системи. Воно перевіряє, наскільки добре ці компоненти працюють разом і чи правильно функціонує система в цілому. Інтеграційні тести можуть включати тестування кінцевих точок API, інтеграцію з базами даних або зв'язок між різними підсистемами.

Функціональне тестування: Функціональне тестування оцінює відповідність системи функціональним вимогам, визначеним для електронної бібліотеки. Воно включає в себе тестування різних функцій і можливостей, щоб переконатися, що вони працюють так, як очікується з точки зору користувача. Функціональні тести імітують взаємодію з користувачем, наприклад, пошук

книг, додавання товарів до кошика або доступ до профілів користувачів, і перевіряють, чи правильно система поводиться у відповідь на ці дії.

Тестування продуктивності: Тестування продуктивності оцінює швидкість реагування, масштабованість і стабільність системи електронної бібліотеки за різних умов навантаження. Воно вимірює показники продуктивності системи, такі як час відгуку, використання ресурсів і пропускну здатність, щоб виявити будь-які вузькі місця або проблеми з продуктивністю. Тестування продуктивності допомагає оптимізувати роботу системи і гарантує, що вона зможе ефективно обробляти очікуваний користувацький трафік.

Тестування безпеки: Тестування безпеки фокусується на виявленні вразливостей і забезпеченні захисту електронної бібліотечної системи від потенційних загроз. Воно включає в себе тестування на наявність загальних ризиків безпеки, таких як SQL-ін'єкції, міжсайтовий скриптинг (XSS) або несанкціонований доступ. Тестування безпеки допомагає захистити дані користувачів, захистити їх від витоку та забезпечити відповідність стандартам безпеки.

Юзабіліті-тестування: Юзабіліті-тестування оцінює зручність і простоту використання електронної бібліотечної системи. Воно передбачає збір відгуків від репрезентативних користувачів, які виконують конкретні завдання, та оцінку їхнього досвіду. Юзабіліті-тестування допомагає виявити будь-які проблеми, пов'язані з зручністю використання, навігацією, або області для поліпшення користувальницького інтерфейсу і загального користувацького досвіду.

Регресійне тестування: Регресійне тестування гарантує, що зміни або доповнення до електронної бібліотечної системи не призведуть до появи нових помилок або проблем, зберігаючи при цьому існуючу функціональність. Воно передбачає повторний запуск раніше виконаних тестів для перевірки того, що основні функції системи все ще працюють належним чином після внесених змін. Регресійне тестування допомагає підтримувати стабільність і надійність системи протягом усього процесу розробки.

Тестування сумісності: Тестування сумісності оцінює сумісність електронної бібліотечної системи на різних платформах, браузерах і пристроях. Воно гарантує, що система функціонує правильно і забезпечує послідовний користувацький досвід у різних середовищах. Тестування сумісності допомагає виявити будь-які проблеми сумісності, такі як неузгодженість макетів або розбіжності у функціоналі, а також дозволяє внести необхідні корективи для підтримки різних конфігурацій користувачів.

Тестування прийнятності для користувача (UAT): Користувацьке тестування передбачає залучення кінцевих користувачів до процесу тестування для перевірки відповідності системи їхнім конкретним вимогам та очікуванням. Це дозволяє реальним користувачам взаємодіяти з системою і надавати відгуки про її зручність, функціональність і загальну придатність. UAT допомагає гарантувати, що електронна бібліотечна система відповідає потребам її цільових користувачів і відповідає їхнім очікуванням.

Автоматизоване тестування: Автоматизоване тестування передбачає використання інструментів і скриптів для автоматичного виконання тестів, що зменшує ручну працю і пришвидшує цикли тестування. Автоматизовані тести можуть бути написані для охоплення різних аспектів тестування, таких як модульне тестування, інтеграційне тестування або регресійне тестування. Вони забезпечують систематичний і повторюваний підхід до тестування і допомагають ефективно виявляти проблеми.

Навантажувальне тестування: Навантажувальне тестування оцінює продуктивність і поведінку електронної бібліотечної системи в умовах очікуваного або пікового навантаження. Воно імітує велику кількість одночасних користувачів або запитів, щоб визначити, як система справляється з навантаженням. Навантажувальне тестування допомагає виявити вузькі місця в продуктивності, обмеження масштабування або ресурси, а також дозволяє оптимізувати систему, щоб вона могла впоратися з очікуваним користувацьким трафіком.

Тестування доступності: Тестування доступності гарантує, що електронна бібліотечна система є доступною для користувачів з обмеженими можливостями та відповідає стандартам і рекомендаціям щодо доступності. Воно передбачає тестування системи за допомогою допоміжних технологій, таких як зчитування з екрану або навігація лише за допомогою клавіатури, щоб переконатися, що всі користувачі можуть отримати доступ до системи і ефективно нею користуватися. Тестування доступності сприяє інклюзивності та забезпечує рівний доступ до інформації та послуг.

Тестування локалізації: Тестування локалізації оцінює готовність системи до роботи з різними мовами, культурами та регіонами. Воно передбачає тестування електронної бібліотечної системи з різними мовними налаштуваннями, кодуваннями символів і регіональними уподобаннями, щоб забезпечити належну локалізацію контенту, форматів дати і часу, валютних символів та інших локалізованих елементів. Тестування локалізації допомагає гарантувати, що система може обслуговувати глобальну базу користувачів.

Тестування обробки помилок і відновлення: Тестування обробки помилок і відновлення фокусується на тому, як електронна бібліотечна система обробляє помилки, винятки і несподівані сценарії. Воно передбачає навмисне спричинення помилок або збоїв, щоб перевірити, чи може система впоратися з ними, відобразити інформативні повідомлення про помилки і відновитися після збоїв без втрати даних або нестабільності системи. Тестування обробки помилок і відновлення підвищує надійність і стійкість системи.

Крос-браузерне тестування: Крос-браузерне тестування оцінює сумісність електронної бібліотечної системи з різними веб-браузерами, забезпечуючи узгодженість поведінки та зовнішнього вигляду. Воно включає тестування функціональності, макета та продуктивності системи в таких популярних браузерах, як Chrome, Firefox, Safari та Internet Explorer/Edge. Кросбраузерне тестування допомагає виявити будь-які проблеми, пов'язані з конкретним браузером, і забезпечує безперебійну роботу користувача в різних браузерах.

Важливо відзначити, що тестування є ітеративним і безперервним процесом протягом усього життєвого циклу розробки. Вибір і комбінація методів тестування можуть змінюватися залежно від вимог проекту, ресурсів і термінів. Комплексна стратегія тестування повинна охоплювати поєднання цих методів, щоб забезпечити ретельне охоплення і створити якісну, надійну і зручну для користувача електронну бібліотечну систему.

У нашому випадку було написано тест кейс для перевірки функцій додатку:

```
import unittest
from app import app

class LibraryAppTestCase(unittest.TestCase):

    def setUp(self):
        app.testing = True
        self.client = app.test_client()

    def test_display_book_list(self):
        response = self.client.get('/books')
        self.assertEqual(response.status_code, 200)
        self.assertTrue('List of Books' in response.get_data(as_text=True))
        self.assertTrue('Book 1' in response.get_data(as_text=True))
        self.assertTrue('Book 2' in response.get_data(as_text=True))

    def test_display_single_book(self):
        response = self.client.get('/books/1')
        self.assertEqual(response.status_code, 200)
        self.assertTrue('Book 1 Details' in response.get_data(as_text=True))
        self.assertTrue('Title: Book 1' in response.get_data(as_text=True))
        self.assertTrue('Author: Author 1' in response.get_data(as_text=True))

    def test_add_book(self):
        data = {'title': 'New Book', 'author': 'New Author'}
        response = self.client.post('/books', json=data)
        self.assertEqual(response.status_code, 201)
        self.assertTrue('New Book' in response.get_data(as_text=True))
        self.assertTrue('New Author' in response.get_data(as_text=True))

    def test_delete_book(self):
        response = self.client.delete('/books/1')
        self.assertEqual(response.status_code, 200)
        self.assertTrue('Book 1 successfully deleted' in response.get_data(as_text=True))

if __name__ == '__main__':
    unittest.main()
```

Даний код перевіряє чи програма може додавати,відображати,переглядати та редагувати книги

4.2 тестування застосунку

Після дослідження які методи тестування бувають та розробки програмного забезпечення переходимо до перевірки всіх функцій.

Робота розпочинається з головного меню (рис 4.1) де користувач бачить додані книги та додавати або видаляти ще книги або журнали

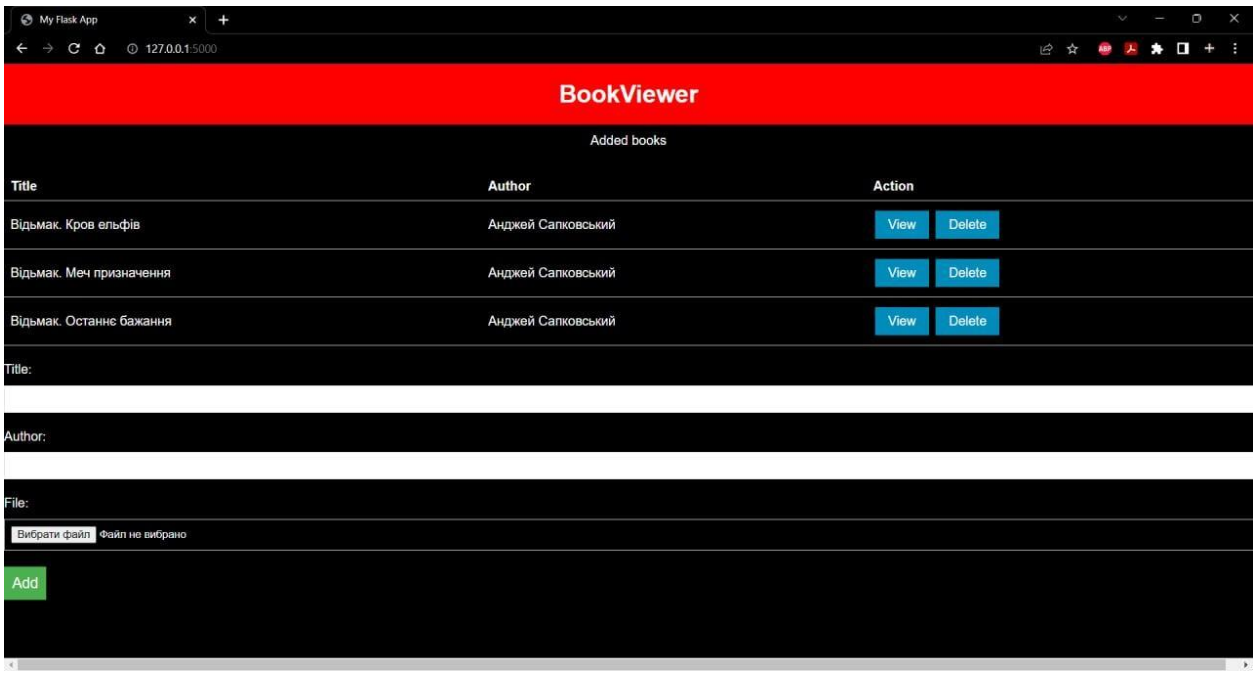

Рисунок 4.1 – Головне меню додатку

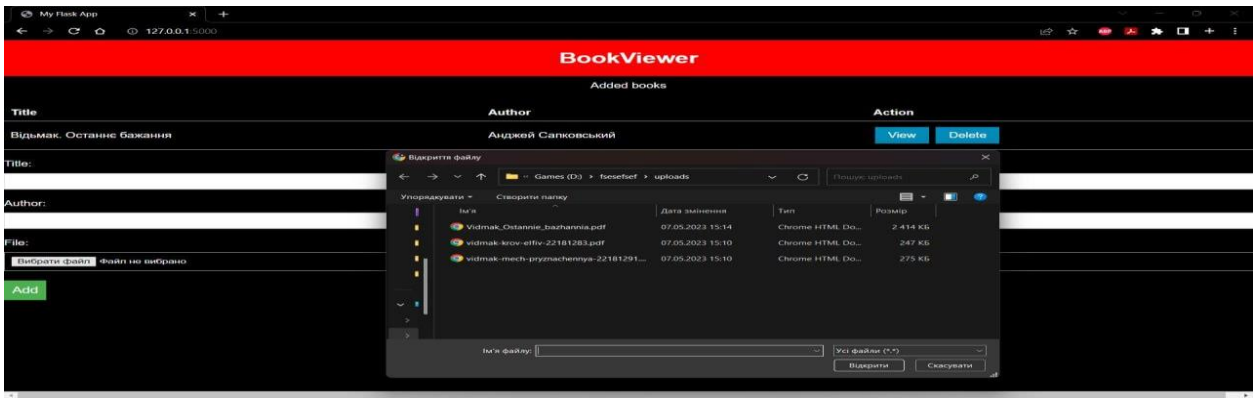

Рисунок 4.2 – Перевірка файлів на сервері

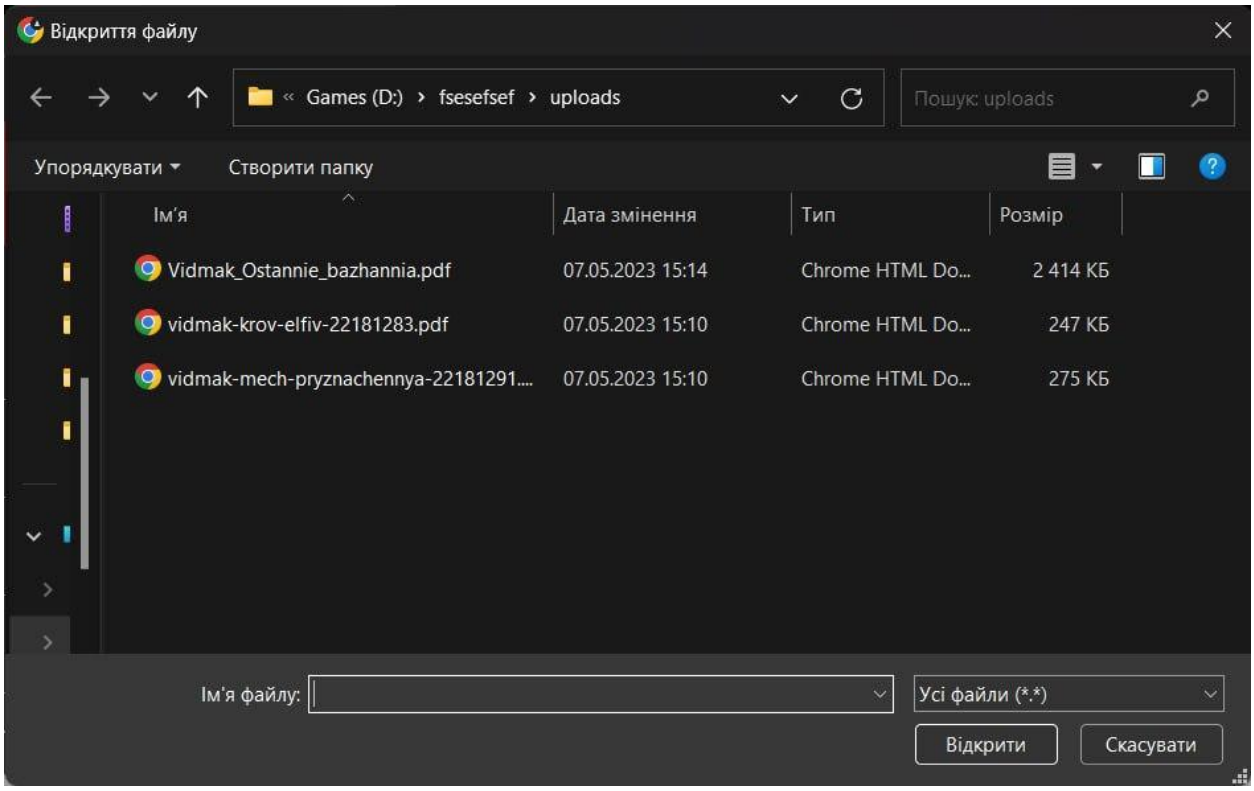

Рисунок 4.3 – Відкриття файлу

Програма пройшла тестування оскільки виконує всі функції які були необхдні тому у свою чергу програма має такі функції:

- додавання книги;
- перегляд книги;
- додавання інформації про автора;
- видалення.

Програма розроблена мовою Python з використанням фреймворку Flask.

Висновки до розділу 4

Результатом четвертого розділу є успішне тестування розробленого додатку. Почався розділ з опису методів тестування, після чого було вибрано ті, які підходять для роботи. Протестовано електронну бібліотеку, критичних багів не виявлено. Наведено зображення, які показують зовнішній вигляд розробленої системи електронної бібліотеки.

ВИСНОВКИ

Отже, розробка електронної бібліотечної системи з використанням Python та Flask як основних технологій пропонує численні переваги та можливості. Протягом усього процесу були розглянуті різні теоретичні аспекти, такі як огляд існуючих бібліотек та мов програмування, для прийняття обґрунтованих рішень. В якості мови програмування було обрано Python завдяки його простоті, універсальності та широкій бібліотечній підтримці.

Функціональні вимоги до електронної бібліотечної системи були ретельно визначені, щоб вона ефективно задовольняла потреби користувачів. Ці вимоги охоплювали такі функції, як пошук книг, автентифікація користувачів, позичання та повернення книг, управління обліковими записами користувачів та надання доступу до цифрових ресурсів. Архітектура системи була розроблена з урахуванням надійної структури бази даних з використанням SQLAlchemy, що забезпечує ефективне зберігання та пошук даних.

Інтерфейс користувача електронної бібліотеки був розроблений таким чином, щоб бути інтуїтивно зрозумілим, зручним і візуально привабливим. Завдяки використанню HTML, CSS та JavaScript було створено адаптивний та інтерактивний інтерфейс, що дозволяє користувачам легко орієнтуватися в системі та отримувати доступ до потрібної інформації.

Розробка серверної частини включала реалізацію необхідного функціоналу за допомогою Python та Flask. Це включало обробку запитів користувачів, обробку даних, управління автентифікацією та авторизацією, а також полегшення комунікації між клієнтом і базою даних. Легкість і модульність Flask у поєднанні з широкою екосистемою розширень зробили його ідеальним вибором для побудови серверних компонентів системи.

Реалізація системи електронної бібліотеки на Python продемонструвала її універсальність та ефективність у створенні надійного та багатофункціонального додатку. Використання принципів об'єктно-орієнтованого програмування в

поєднанні з читабельністю та виразністю мови Python сприяло легкості обслуговування, використання та розширюваності системи.

Тестування відіграло вирішальну роль у забезпеченні якості та надійності електронної бібліотечної системи. Різні методології тестування, такі як функціональне тестування, юзабіліті-тестування, тестування продуктивності, тестування безпеки та тестування сумісності, були використані для виявлення та усунення дефектів, покращення користувацького досвіду та перевірки системних вимог. Автоматизоване тестування сприяло підвищенню ефективності та результативності процесу тестування.

Загалом, розробка електронної бібліотечної системи продемонструвала потужність і гнучкість Python і Flask у створенні складних веб-додатків. Проект об'єднав теоретичні основи, практичні міркування та ретельне тестування, щоб забезпечити надійне, зручне та ефективне рішення для електронної бібліотеки. Успіх проекту підкреслює важливість ретельного планування, уваги до деталей та системного підходу при розробці програмних систем.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Офіційний веб-сайт мови програмування Python: веб-сайт URL: https://www.python.org/ (дата звернення: 11.01.2023)

2. Flask Documentation - Офіційна документація Flask : веб-сайт URL: https://flask.palletsprojects.com/en/2.3.x/ (дата звернення: 12.01.2023)

3. SQLAlchemy Documentation, Документація SQLAlchemy : веб-сайт URL: https://docs.sqlalchemy.org/en/20/ (дата звернення: 13.01.2023)

4. Django Documentation - Офіційна документація Django : веб-сайт URL: https://docs.djangoproject.com/en/4.2/ (дата звернення: 13.01.2023)

5. Python Package Index (PyPI) : веб-сайт URL: https://pypi.org/ (дата звернення: 16.01.2023)

6. Real Python : веб-сайт URL: https://realpython.com/ (дата звернення: 18.01.2023)

7. Full Stack Python : веб-сайт URL: https://www.fullstackpython.com/ (дата звернення: 20.01.2023)

8. The Flask Mega-Tutorial : веб-сайт URL: https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world (дата звернення: 23.01.2023)

9. Flask Web Development with Python Tutorial Series: веб-сайт URL: https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world (дата звернення: 01.02.2023)

10. The Digital Public Library of America: веб-сайт URL: https://dp.la/ (дата звернення: 10.01.2023)

11. HathiTrust Digital Library: веб-сайт URL: https://www.hathitrust.org/ (дата звернення: 10.01.2023)

12. Бібліотека Вернадського: веб-сайт URL: http://www.nbuv.gov.ua/ (дата звернення: 10.01.2023)

13. Інтерфейс користувача Wikipedia.com: вебсайт. URL: https://uk.wikipedia.org/wiki/Інтерфейс_користувача (дата звернення: 10.03.2023)

14. Дегтярьова Л. М. Навчальний посібник з дисципліни «Технології розробки програмного забезпечення» для студентів спеціальності 123 «Комп'ютерна інженерія». Полтава: ПолтНТУ, 2017. 218 с. URL: http://reposit.nupp.edu.ua/handle/PoltNTU/4461 (дата звернення: 10.03.2023)

15. Роберт М. Чиста архітектура: мистецтво розробки програмного забезпечення"/ Роберт Мартін, Фабула, 2019. 416 с

16. Мартін Р. Чистий код. Р.Мартін М.: Фабула, 2019. 416 с.

17. Grady Booch. Unified Modeling Language User Guide: 2nd Edition. Addison-Wesley Professional, 2017. 504 p.

18. Крепич С. Я., Співак І.Я. Якість програмного забезпечення та тестування: базовий курс: навч. посіб. Тернопіль: ФОП Паляниця В. А., 2020. 478 с. URL: http://dspace.wunu.edu.ua/handle/316497/39773 (дата звернення: 10.04.2023)

19. Якість програмного забезпечення та тестування: опор. конспект лекцій: уклад. О. Л. Козак Тернопіль: ТНЕУ, 2012. 72с. URL: http://dspace.tneu.edu.ua/handle/316497/7362 (дата звернення: 11.04.2023)

20. База Даних Wikipedia.com: вебсайт. URL: https://uk.wikipedia.org/wiki/База_даних (дата звернення: 02.03.2023)