Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

Навчально-науковий інститут деревообробних та комп'ютерних технологій і дизайну
(повне найменування інституту, назва факультету (відділення))

Кафедра інформаційних технологій
(повна назва кафедри (предметної, циклової комісії))

Пояснювальна записка

до дипломної роботи

перший (бакалаврський)
(рівень вищої освіти)

на тему:**«Розробка сайту з інформаційно-пошуковою системою для бібліотеки»**

Виконав: студент 3 курсу групи КНСз-31
спеціальності
122 "Комп'ютерні науки"
(шифр і назва напряму підготовки, спеціальності)
Скоринович О.І.
(прізвище та ініціали)
Керівник асист. Івасюк Р.В.
доц. Різник О.Я.
(прізвище та ініціали)
Рецензент
(прізвище та ініціали)

Львів – 2022

Національний лісотехнічний університет України

(повне найменування вищого навчального закладу)

Інститут ННІ Деревообробних та комп'ютерних технологій і дизайну

Кафедра інформаційних технологій

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 «Комп'ютерні науки»

(шифр і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри Крошний І.М.

"__" _____2022 р.

З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Скориновичу Оресту Ігоровичу

(прізвище, ім'я, по батькові)

1. Тема роботи Розроблення програмного рішення "Сайт з Інформаційно-пошуковою системою для бібліотеки" засобами ASP.NET Core

керівник проекту ***Івасюк Руслан Васильович***

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом вищого навчального закладу від "03" 10 2022р. № С-407

2. Термін подання студентом роботи 12 грудня 2022 р.

3. Вихідні дані до роботи *Розробити сайт з інформаційно-пошуковою системою для бібліотеки за допомогою фреймворку ASP .NET Core. Розробити BackEnd та FrontEnd частини застосунку та сполучити їх. Вихідні дані та прототипи схожих систем. Огляд алгоритмів та програмних засобів для розроблення програмного застосунку.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно

1) Характеристика об'єкту проектування та постановка задачі; 2) Огляд літератури; 3) Системний аналіз; 4) Розробка та Відображення алгоритмів роботи компонентів системи; 5) Розробка програмного рішення;6) Експериментальна частина

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Підготовка презентаційного матеріалу до доповіді (слайди для доповіді загальним обсягом 10-12 слайдів)

6. Дата видачі завдання 06 жовтня 2022 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1.	*Огляд літературних даних.*	*6.10.2022р. – 20.10.2022р.*	Виконано
2.	*Розділ 1. Характеристика об'єкту проектування та постановка задачі.*	*21.10.2022р. – 28.10.2022р.*	Виконано
3.	*Розділ 2. Огляд літератури.*	*29.10.2022р. – 05.11.2022р.*	Виконано
4.	*Розділ 3. Системний аналіз.*	*05.11.2022р. – 11.11.2022р.*	Виконано
5.	*Розділ 4. Розробка та відображення алгоритмів роботи компонентів системи.*	*12.11.2022р. – 20.11.2022р.*	Виконано
6.	*Розділ 5. Розробка програмного рішення.*	*21.11.2022р. – 28.11.2022р.*	Виконано
7.	*Розділ 6. Експериментальна частина.*	*29.11.2022р. – 11.12.2022р.*	Виконано
8.	*Аналіз отриманих результатів та написання висновків. Оформлення дипломної роботи.*	*12.12.2022р. – 16.12.2022р.*	Виконано
9.	*Здача пояснювальної записки на перевірку керівнику, виправлення помилок та здача роботи рецензенту.*	*17.12.2022 р.*	Виконано

Студент _____ **Скоринович О.І.**
(підпис) (прізвище та ініціали)

Керівник роботи _____ **Івасюк Р.В.**

Керівник роботи _____ **Різник О.Я.**
(підпис) (прізвище та ініціали)

Реферат

Дана бакалаврська кваліфікаційна робота містить: сторінок 101, рисунків 48, 1 додаток та 20 джерел використаної літератури.

В дипломній роботі розроблено веб-орієнтовану інформаційно-пошукову систему для бібліотеки у вигляді сервера, та створений клієнт для роботи з нею.

Робота містить сім розділів.

У першому розділі описано характеристику досліджуваної системи та описана постановка задачі на бакалаврську роботу.

У другому розділі проводився огляд та аналіз літературних джерел.

У третьому розділі проводився системний аналіз системи, а саме визначені основні проблеми системи та основні цілі системи.

У четвертому розділі описано основні алгоритми системи, а також створені діаграми, що описують функціонування системи та її окремих блоків.

У п’ятому розділі описувалась структура системи, та проводився опис бібліотек та модулів системи.

У шостому розділі наведені інструкції користувача та створені тести для системи.

ABSTRACT

This bachelor qualification report includes: 101 pages, 48 pictures,1addition and 20 sources of used literature.

In the diploma work a web-oriented information retrieval for the library in the form of a server system was developed, and a client was created to work with it.

The work contains seven sections.

The first section describes the characteristics of the studied system and describes the problem for the bachelor's thesis.

The second section provides a review and analytical sources.

In the third section the system analysis of the system was carried out, namely the main problems of the system and the main goals of the system were identified.

The fourth section describes the basic algorithms of the system, as well as diagrams describing the operation of the system and its individual units.

The fifth section describes the structure of the system, and describes the libraries and modules of the systems.

The sixth section provides user instructions and tests for the system.

ЗМІСТ

ВСТУП

Важливою рушійною силою формування інформаційного простору є розвиток інформаційних технологій.

Сьогодні все більше і більше ресурсів переміщуються в Інтернет, та надають віддалений доступ до своїх ресурсів. Це робиться для того, щоб надавати послуги більш широкій смузі користувачів та з кращою швидкістю обробки даних.

Питання інформаційних потреб є головним питанням у бібліотеках і бібліотечній справі сьогодні, і воно було в центрі уваги експертів, але зазвичай розглядається як дослідження читацького інтересу. Його функції та діяльність змінюються з впровадженням комп'ютеризації та доступу до Інтернету. У зв'язку з цим створення та підтримка власного веб-сайту стало одним із практичних напрямків роботи сучасного книгозбирання. Веб-сайт є важливою частиною його іміджу, він може не тільки показати, що це за організація та які послуги вона надає користувачам, але й показати ступінь володіння новими інформаційними технологіями. Тим більшої актуальності набула розробка інформаційної структури веб-сайту, яка б грамотно та професійно відображала роботу та імідж бібліотеки, її електронних ресурсів і сервісів в Інтернеті.

Актуальність проекту полягає в тому, що книги читають і будуть читати завжди, проте не завжди вдається знайти потрібну книгу, чи сервіс який виконував б операції пошуку і фільтрації достатньо швидко з великим списком книг, і набагато зручніше ознайомлюватись з книгою віддалено в любій точці світу. Програмне рішення, що проектується надасть можливість здійснювати швидку фільтрацію та пошук книг, та надасть користувачам віддалений доступ до бібліотеки.

Метою роботи є розробка серверного рішення для надання фільтрації, пошуку та сортування книг, та інших функцій, а також клієнта, що буде відображати графічний інтерфейс та спілкуватись з сервером.

Об’єктом проектування є вдосконалений метод пошуку бібліотечної інформації для комфортної роботи клієнта.

Предметом розробки є правила побудови серверного рішення та процес написання серверного рішення за допомогою мови C# на базі фреймворка ASP.NET Core, використовуючи IDE Visual Studio 2019.

Відповідно до поставленої мети в роботі вирішуються наступні задачі:

1) реєстрація/авторизація користувача;

2) відображення детальної інформації про книгу з підбором схожих книжок за тематикою;

3) відображення детальної інформації про авторів з підбором найкращих книжок автора;

4) залишення відгуків для прочитаних книг;

5) адміністрування бібліотеки, добавлення, редагування, видалення книг.

Результатом виконання роботи є створений сервер, що готовий обробляти запити користувачів та клієнт, що надсилатиме запити до сервера та формуватиме користувацький інтерфейс на базі отриманих даних.

Завданням бакалаврської кваліфікаційної роботи є розробити програмне рішення, що дозволить працювати з бібліотекою віддалено та забезпечить швидку фільтрацію і пошук книг. Для досягнення поставлених цілей сформовано наступні завдання:

- вивчення та дослідження предметної області;
- огляд літературних джерел, для кращого теоретичного розуміння;
- проектування та розробка архітектурних рішень системи;
- побудова діаграм, що описують функції системи та її функціональних блоків;
- розробка програмного рішення для сервера та клієнта, відповідно до діаграм та функцій системи;
- створення тестів для робочої системи та перевірка правильності роботи;
- аналіз отриманих даних.

РОЗДІЛ 1. ХАРАКТЕРИСТИКА ОБ'ЄКТУ ПРОЕКТУВАННЯ ТА ПОСТАНОВКА ЗАДАЧІ

На даний час існує багато сайтів, веб-сервісів які призначені для перегляду книг, їх обговорення, пошуку книг, читання книг, або ознайомлення зкнигами. Таких систем дійсно багато, але зазвичай вони досить прості і дуже рідко поєднують багато функціональності зібрану в межах однієї системи з можливістю розширення системи. Також багато з вже існуючих систем не надають швидкого пошуку для системи роботи з бібліотекою.

Інформаційна система пошуку книг, буде створюватись у вигляді сервера, що прийматиме запити від користувачів, оброблятиме їх та видаватиме відповідні повідомлення у відповідь з передачею даних де потрібно. Оскільки для системи важливим є швидкість виконання операцій пошуку, сортування та фільтрації книг, то потрібно буде знайти хмарну технологію що забезпечує виконання даних функцій з великою ефективністю і великим обсягом даних, при цьому не нагружаючи розроблювальну систему.

Також для системи є важливим мобільність і здатність до розширення, тому сервер потребує грамотно розробленої архітектурної структури. Окрім цього системі потрібно буде десь зберігати дані книг, тому необхідне буде підключення до бази даних. Сервер системи повинен швидко обробляти запити клієнтів, та надавати точні відповіді у разі виникнення будь-яких помилок.Також сервер повинен зберігати усю детальну інформацію системи відкористувача.

Крім розробки сервера, потрібно буде розробити клієнт, що буденадсилати запити до сервера який їх буде обробляти, перевіряти та відповідати на них. Клієнт повинен бути розроблений з використанням сучасного UI/UX, для кращого досвіду користування користувачів. Користувач повинен одразу розуміти як взаємодіяти з системою, на що потрібно жати, і чого очікувати від того чи іншого компонента.

Як зазначалось існує багато систем, що надають схожий функціонал, тому окрім функцій пошуку, фільтрації та сортування в системі будуть присутні система також повинна виконувати:

- реєстрація/авторизація користувача у системі з використанням JWT токена;

- відображення детальної інформації про книгу з підбором схожих книжок за тематикою;

- відображення детальної інформації про авторів з підбором найкращих книжок автора;

- залишення, видалення та коригування відгуків про ту чи іншу книгу, з можливістю залишення відповідей на коментарі;

- можливість керування контентом бібліотеки (добавлення, редагування та видалення книжок);

- функція Bookshelves (книжкові полички), дозволяє користувачам зберігати книжки на віддалених книжкових полицях для подальшого їх читання або зберігання як колекції.

В системі мають бути присутні два типи користувачів а саме Читач та Адміністратор. Система повинна вміти розпізнавати з яким типом користувача вона працює, адже інтерфейс та функції у всіх типів користувачів будуть відрізнятись у деяких місцях.

РОЗДІЛ 2. ОГЛЯД ЛІТЕРАТУРИ

2.1. Організація та принципи роботи веб-сервісу

Сучасні Інтернет-додатки використовують різноманітні програмні платформи для розробки веб-додатків. Деякі програми можна розробити на Java, інші на .Net, Angular JS, Node.js. Виходячи з цього, може виникнути питання: що таке веб-сервіси і як вони використовуються? По суті, робоче середовище невидиме для звичайних користувачів.

Веб-сервіси забезпечують загальну платформу, яка дозволяє багатьом додаткам, побудованим на різних мовах програмування, спілкуватися один з одним. Щоб відповісти на запитання, що таке веб-служба, це стандартизоване середовище для поширення зв'язку між клієнтськими та серверними програмами у Всесвітній павутині. Він доступний для редагування та має пакет покращень [6] .

Крім того, веб-сервіс можна визначити як програмний модуль, призначений для виконання певного набору завдань. Структурні елементи вашої програми можна знайти в Інтернеті та назвати відповідно. Під час виклику веб-служба може надавати функціональні можливості клієнтам, які запитують цей портал. Це відбувається протягом секунд [1].

Коли мова заходить про те, що таке веб-сервіс, ми повинні розглянути умови його роботи. Це допоможе вам зрозуміти, як працює система.

Клієнт здійснює серію викликів веб-служби, надсилаючи запити до сервера, на якому розміщено фактичну веб-службу. Ці запити виконуються за допомогою так званих віддалених викликів процедур. Виклик віддаленої процедури (RPC) — це виклик методу, розміщений на відповідний запит [6].

Прикладом веб-сервісу є Amazon. Організація має мережу інтернет-магазинів і системи доставки. Він надає веб-сервіс для запиту цін на продукти, які продаються онлайн через amazon.com. Зовнішній рівень або рівень перегляду може бути .Net або Java, але кожна мова програмування має власний спосіб взаємодії з веб-службами.

Основним компонентом веб-сервісів в Інтернеті є дані, що передаються

між клієнтом і сервером, тобто XML, JSON (Extensible Markup Language). Це прості для розуміння проміжні мови, які розуміють багато мов програмування. Отже, коли програми спілкуються одна з одною, вони фактично спілкуються в XML або JSON. Він забезпечує спільну платформу для додатків, розроблених на різних мовах програмування, для спілкування один з одним.

Для повнофункціонального середовища розробки потрібні певні компоненти. Ці компоненти повинні існувати незалежно від мови розробки, яка використовується для програмування конкретного запиту. Ви можете створити свій власний веб-сервіс. Для цього вам потрібно створити портал для цього розгортання та встановити робоче середовище програмування. Існує два основних типи веб-служб: SOAP і RESTful.

2.2 Порівняння REST і SOAP технологій

У сучасному світі Інтернет — це глобальна мережа, яка об’єднує велику кількість комп’ютерів, з’єднаних для спільного використання ресурсів і інформації. Підключайте різні типи мереж, включаючи урядові мережі, академічні мережі, мережі малого бізнесу та великі корпоративні мережі.

Основою таких зв'язків є підхід «клієнт-сервер». Послуги надаються шляхом взаємодії двох процесів на комп’ютері користувача та комп’ютері-сервері [4].

Більшість інформації в Інтернеті надсилається за допомогою HTTP. Раніше цей протокол використовувався лише для надсилання невеликих обсягів інформації. За передачу великих файлів відповідав протокол FTP (FileTransferProtocol). Тепер файли будь-якого розміру передаються через HTTP, який став домінуючим протоколом.

REST — це архітектурний стиль програмного забезпечення для створення розподіленої та масштабованої мережі на основі маніпулювання ресурсами та специфікацій HTTP. REST являє собою розробку веб-архітектури, яка характеризує та регулює макровзаємодії чотирьох веб-компонентів (сервер, мережевий шлюз, проксі та клієнт), не накладаючи обмежень на окремих учасників. Отже, REST в основному визначає правильну поведінку учасників [5].

REST також має деякі архітектурні обмеження:

1) клієнт-сервер: першою архітектурою, яка успадкувала обмеження, є архітектура клієнт-сервер. Це обмеження вимагає поділу обов'язків між компонентами, які зберігають і оновлюють дані (сервер), і компонентами, які відображають дані в інтерфейсі користувача та відповідають на дії в цьому інтерфейсі (клієнт). Такий поділ дозволяє компонентам розвиватися незалежно [5].

2) без стану: взаємодія між сервером і клієнтом є без стану. Тобто кожен запит містить всю інформацію, необхідну для його обробки, і не покладається на те, що сервер знає щось із попередніх запитів. Обмеження "без стану" не означає, що ви можете створювати системи без стану лише з архітектурою REST. Без стану означає, що сервер не знає про стан клієнта і йому не потрібно запам'ятовувати порядок запитів. бо вони незалежні. Наприклад, коли клієнт запитує головну сторінку сайту, сервер відповідає на запитання і забуває про клієнта. Клієнт може тримати сторінку відкритою роками, перш ніж натиснути посилання, а потім сервер відповість іншим запитом. Тим часом сервер може відповідати на запити інших клієнтів або нічого не робити — для клієнта це не має значення [5].

3) кешування: системи, написані в цьому стилі, повинні підтримувати кешування. Це означає, що дані, надіслані сервером, повинні містити інформацію про те, чи можна їх кешувати, і якщо так, то як довго. Це може підвищити продуктивність, уникаючи надлишкових запитів, але це також знижує надійність системи, оскільки дані в кеші можуть застаріти [5].

4) уніфікований інтерфейс: усі компоненти архітектури REST

Підтримує уніфікований інтерфейс. Це зменшує зв'язки між компонентами та ви можете легко змінити компоненти за потреби. Це досягається за допомогою кількох дрібнозернистих обмежень.

4.1) ідентифікація ресурсу: URI запиту має точно визначати, який ресурс очікується та формат відповіді. Репрезентація ресурсу надається клієнту за запитом. Ресурси концептуально відокремлені від уявлень. Наприклад, сервер може надсилати дані з бази даних у форматі HTML, XML або JSON, жоден із

яких не є типом даних, які зберігаються сервером усередині.

4.2) маніпулювання ресурсами за допомогою представлення: якщо клієнт має представлення ресурсу, що містить метадані, він має достатньо інформації, щоб змінити або видалити цей ресурс.

SOAP — це простий протокол (для його використання не потрібно створювати новий протокол). Він заснований на ідеї, що обмін інформацією повинен відбуватися в певний момент розподіленої архітектури. Крім того, у системах, де процеси обробки можуть бути перевантаженими або складними, цей протокол є перевагою, оскільки вимагає мінімальних ресурсів. Усі операції можна виконувати через HTTP. Специфікація SOAP складається з трьох основних компонентів: конверта SOAP (SOAPenvelope), набору правил шифрування та засобів взаємодії між запитами та відповідями.

Давайте порівняємо підходи SOAP і REST, щоб краще зрозуміти їхні сильні та слабкі сторони.

SOAP — це ціле сімейство протоколів і стандартів, що говорить про багато і складність з точки зору машинної обробки. Тому REST працює швидше.

SOAP використовує HTTP як транспортний протокол, тоді як REST базується на ньому [4]. Це означає, що всі поточні розробки на основі протоколу HTTP, включаючи кешування на рівні сервера, масштабування тощо, продовжуватимуть працювати так само в архітектурі REST. Для SOAP вам доведеться шукати інші засоби. Натомість служби SOAP зможуть використовувати будь-який протокол транспортного рівня замість HTTP.

SOAP підходить для складних архітектур, де взаємодія об’єктів виходить за рамки теорії CRUD, але для програм, які не виходять за межі цієї теорії, REST може добре підійти через свою простоту та прозорість. Насправді, крім «створити», «читати», «змінити», «видалити», якщо об’єкти вашого сервісу не потребують нічого більш складного (в принципі, у 99% випадків цього має бути достатньо), то, можливо, Досить REST — правильний вибір. Крім того, REST більш продуктивний порівняно з SOAP, оскільки не вимагає витрат на аналіз складних XML-команд на сервері (виконуються звичайні HTTP-запити - PUT, GET, POST, DELETE). Однак SOAP більш надійний і безпечний.

Зі сказаного вище, SOAP і REST можна зробити такий висновок: навряд чи знайдеться завдання, для якого важко вирішити, чи має сенс використовувати SOAP чи REST.

2.3 Протокол HTTP

Протокол Протокол HTTP (HyperText Transfer Protocol) — це протокол прикладного рівня для обміну додатками в межах розподілених, спільних або гетерогенних інформаційних систем. Цей протокол дозволяє програмам обмінюватися даними, представленими в зручному для читання форматі [20].

Гіпертекстовий документ складається з даних, позначених тегами HTML, і є комбінацією тексту, зображень, гіперпосилань та інших засобів відображення даних. Гіперпосилання, один із найважливіших компонентів документів HTML, є інтерактивними областями, які ведуть до пошуку даних, пов’язаних із гіперпосиланням. Це дозволяє користувачам, які працюють з гіпертекстовою інформацією переміщуватись в наборі документів або навіть в Інтернеті та використовуйте контекстні гіперпосилання, щоб отримати потрібну інформацію.

Оскільки HTTP є надбудовою протоколу TCP, у передачі даних беруть участь дві сторони: клієнт і сервер. Клієнт є ініціатором підключення, запитуючи дані або послуги від сервера. Клієнт - це, в принципі, програма під назвою браузер, яка дозволяє як переглядати гіпертекстову інформацію, так і приймати файли інших форматів. Щоб отримати інформацію, клієнт надсилає запит на сервер, що містить опис запитуваної інформації.

Сервер, який передає дані за допомогою HTTP, називається веб-сервером. Ця програма обробляє запити від клієнтів і надсилає запитувані дані у вигляді відповідей. Відповідь містить отримані дані плюс метаінформацію, як описано в [20].

Отримання цікавих для користувача даних включає наступні елементи:

1) користувач в рядку браузера вводить адресу ресурсу, що цікавить.

2) браузер робить запити на основі інформації, отриманої від користувача, з урахуванням його налаштувань і конфігурації операційної системи.

3) браузер підключиться до сервера, який, ймовірно, розташований за

адресою запиту на віддаленому комп'ютері.

4) сервер аналізує запит і виконує необхідні дії. Формує відповідь, яка містить текст запитуваного документа. Для гіпертекстових документів він завантажується з файлу або генерується динамічно.

використовувати сценарій. У відповіді також міститься інформація про його дані.

5) сервер надсилає відповідь браузеру.

6) браузер аналізує відповідь і зберігає отримані дані у файл або, у випадку гіпертекстового документа, аналізує теги HTML і відображає документ на екрані.

З'єднання може бути створено за допомогою одного або кількох проміжних елементів точки підключення. Ці шляхові точки можна розділити на три групи:

1) проксі-сервер - програма-посередник, яка виконує як клієнтські, так і серверні функції для здійснення запитів від імені інших клієнтів. Запит обробляється проксі-сервером, модифікованим (у цьому випадку проксі-сервер називається непрозорим) або немодифікованим (у цьому випадку проксі-сервер називається прозорим). , пересилається на проксі-сервер. Проксі-сервер дозволяє групі комп'ютерів діяти як єдиний клієнт. Це зазвичай використовується під час підключення локальної мережі до Інтернету.

2) шлюз - як проксі-сервер, він транслює запити, але я їх не змінюю. Шлюз отримує запити від клієнтів на сервери, які містять запитуваний ресурс. Таким чином, клієнт не може визначити, чи підключається він через шлюз або безпосередньо до сервера, що містить ресурс.

3) тунель (Tunnel) - Проміжна програма, яка підтримує підключення. Тунель не вважається елементом передачі HTTP після встановлення з'єднання, але зазвичай з'єднання ініціюється самим HTTP-запитом. Тунель перестане працювати, якщо хоча б один з учасників обміну даними закриє з'єднання.

Запити та відповіді не потрібно змінювати, за винятком проксі-серверів, щоб підтримувати функціональність передачі даних під час підключення через посередників. У цьому випадку запит клієнта повинен містити додаткові поля. Однак, з точки зору сервера, дані обмінюються безпосередньо з клієнтом, тому в

запиті не відбувається жодних змін.

Запити, надіслані від клієнта до сервера, допомагають визначити точний запитуваний ресурс, а також містять інформацію, необхідну для правильної обробки запиту. За своєю структурою запит складається з трьох частин:

1) рядок запиту.

2) блок заголовка.

3) об'єкт.

Рядок запиту складається з трьох полів, розділених пробілами (ASCII-код 20h, далі SP), і двох полів: повернення каретки (ASCII-код 0Dh, далі CR) і переходу рядка (ASCII-код 0Ah, далі LF).Закінчується літерною комбінацією. . Елементи рядка запиту представлені такими полями:

1) метод - визначає метод обробки, застосований до запитуваного ресурсу. Залежно від зазначеного методу формат запиту може відрізнятися [20]. Допустимі методи:

1.1) ПАРАМЕТРИ - Отримайте параметри, які підтримуються сервером. На стороні клієнта його можна використовувати для модифікації запиту залежно від функцій, які підтримує сервер [20].

1.2) GET - Запит ресурсу. URL містить всю інформацію, необхідну серверу для пошуку та повернення ресурсу [20].

1.3) HEAD - як GET, але тіло повідомлення не надсилається. Він використовується для отримання заголовків певного ресурсу з сервера, зазвичай перевіряючи, чи ресурс змінився за міткою часу [20].

1.4) POST - створити новий ресурс. Запит POST зазвичай містить дані для створення нового ресурсу [20].

1.5) PUT - Оновлення існуючого ресурсу. Вміст може включати

Оновлені дані для ресурсів [20].

1.6) DELETE - Видалити існуючий ресурс [20].

1.7) TRACE - використовується для отримання інформації з сервера про «Хоп» (найближчий роутер, роутер за адресою на відстані одного "стрибка", пройдена запитом). Кожен проміжний проксі-сервер (програма для кешування відповідей на запити клієнтської частини програми, що надсилає в Інтернет або

WWW, працює на рівні програми. Копії отриманих веб-сторінок, файлів тощо зберігаються на сервері протягом while) , і після отримання наступних подібних запитів сервер сам надсилає доступну копію, щоб зменшити час відповіді та мережевий трафік.Ви можете фільтрувати запити, заблокувавши [20].

2) resource URI (Universal Resource Identifier) - вказує розташування запитуваного ресурсу в стандартизованому форматі. тобто адреса ресурсу. При використанні методу GET цей рядок може містити серію параметрів, які надсилаються на сервер у вигляді рядків у формі "назва_параметра = значення_параметра", розділених символами амперсанда "&". Блок параметрів знаходиться в кінці рядка URI і відокремлений від адреси знаком питання «?». [1];

3) версія протоколу HTTP - при розробці програми реалізована підтримка прийому запитів, що відповідає версіям 1.0 і 1.1. Вони відповідають рядкам «HTTP/1.0» і «HTTP/1.1» відповідно.

Блок заголовка, який слідує за рядком запиту, може складатися з одного або кількох заголовків.

1) тема запиту – діє як кваліфікатор для запиту та містить поля, включаючи

Інформація про запит і інформація про конфігурацію клієнтської машини.

2) тема об’єкта – якщо запит містить деякий об’єкт (будь-який набір даних), поля в цій таблиці описують об’єкт, надаючи його формат, кодування та інші параметри.

3) загальний заголовок - містить параметри служби, необхідні для забезпечення правильності передачі та включення додаткових послуг, таких як кешування.

Секція заголовка закінчується двома парами символів, CR і LF. Це дозволяє легко визначити, що запит отримано, оскільки сам запит не може містити таких комбінацій символів.

Відповідь, надіслана сервером клієнту, може бути отримана лише в результаті обробки запиту клієнта. Містить опис результатів запиту та, якщо запитувалися дані, запитуваний ресурс.

За своєю структурою відповідь складається з наступних частин:

1) рядок стану;

2) блок заголовка.

3) об'єкт.

Рядок стану складається з трьох полів, розділених символами SP, і закінчується серією символів CR, LF. Елементи рядка стану:

1) версія протоколу HTTP - розроблені програми завжди використовують рядок "HTTP/1.1".

2) статусний код (Status Code) - Трисимвольний цифровий код, який ідентифікує результат запиту.

Ось коди стану:

2.1) 1xx: Інформація про процес передачі. Цей клас коду HTTP/1.1 і використовується лише для попереднього зв’язку клієнт-сервер.

2.2) 2xx: Інформація про успішне отримання та обробку запитів клієнтів. Код у цьому класі сповіщає клієнта про успішну обробку запиту.

Приклад:

- 202 Прийнято: запит прийнято до обробки (не виконано), але відповідь може не містити запитуваних даних. Цей параметр корисний для асинхронної обробки на стороні сервера.

- 204 Немає вмісту: текст повідомлення не надіслано.

- 205 Скинути вміст: клієнт повинен скинути дані, введені користувачем.

- 206 Частковий вміст: вказує, що містить відповідь лише на частину ресурсу. Клієнт може надіслати додаткові заголовки, які надають інформацію про точний обсяг запитуваного ресурсу та тривалість життя вмісту.

2.3) 3xx: Передача. Цей код вказує клієнту, що потрібні подальші дії. Найпоширеніший варіант – зробити запит за іншою URL-адресою.

приклад:

- 301 Переміщено назавжди: запитуваний об’єкт назавжди переміщено за новою URL-адресою.

- 303 Дивіться інше: запитуваний об'єкт був тимчасовим

Переміщено на нову URL-адресу. У заголовку вказується тимчасова URL-адреса розташування відповіді;

- 304 Not Modified: сервер виявив, що ресурс не було змінено. Клієнти повинні використовувати кешовані копії.

2.4) 4xx: Інформація про помилки з боку клієнта. Ці коди використовуються, коли сервер вважає, що клієнт припустився помилки: неправильно сформований запит або запит на ресурс, недоступний для клієнта.

Приклад:

- 400 Поганий запит: у запиті виявлено помилку.

- 401 Неавторизовано: запит вимагає авторизації, клієнт може повторити запит, додавши заголовок авторизації. Якщо клієнт уже використовував цей заголовок, це означає, що для успішної автентифікації були надані неправильні дані.

- 403 Forbidden (Заборонено): серверу заборонено доступ до клієнта вказаний ресурс.

- 405 Method Not Allowed: у рядку запиту використано недійсний метод HTTP або сервер не підтримує цей метод.

- 409 Конфлікт (конфлікт): сервер не зміг виконати запит, оскільки клієнт намагався змінити ресурс, часова позначка якого не збігалася з клієнтською. У більшості випадків конфліктні ситуації виникають, коли ресурси редагуються спільно за допомогою запитів PUT.

2.5) 5xx: Інформація про помилки з боку сервера. Цей тип коду використовується для повідомлення про помилку операції через помилку на стороні сервера. приклад:

- 501 не реалізовано: сервер наразі не підтримує функції, необхідні для обробки запиту.

- 503 Служба недоступна: сервер недоступний, обробка запитів з технічних причин або перевантаження. Зазвичай сервер навіть не відповідає, і запит перевищить ліміт часу очікування сервера (час очікування; скільки часу чекати на подію (зазвичай встановлено для операцій на периферійних пристроях), коли вона відбувається та обробляється, наприклад, у ситуації помилки). перевищувати.

3) Фраза статусу (фраза причини) - коротка фраза, що описує код статусу

виконання запиту.

Блок заголовка після рядка стану може складатися з одного або кількох заголовків.

1) тема запиту;

2) предмет об'єкта;

3) Загальна назва.

РОЗДІЛ 3. СИСТЕМНИЙ АНАЛІЗ

Описавши характеристику об’єкта та поставивши задачу розроблювальної системи, необхідно визначити методи та способи досягнення поставлених завдань. Для того, щоб обрати найкращий та найоптимальніший з можливих варіантів методів потрібно провести системний аналіз розроблювальної системи.

Системний аналіз – це набір методичних засобів, що застосовується для підвищення ступеня обґрунтованості рішень у складних питаннях наукового, соціального та економічного характеру. Системний аналіз розглядає об'єкти як системи і переважно цілеспрямований. Основні методологічні принципи системного аналізу базуються на принципах системного підходу. Необхідність вивчення методології та методики системного аналізу зумовлена необхідністю її застосування при створенні та розвитку комп’ютеризованих інформаційних систем. Між елементами існують складні й часто неструктуровані зв’язки. Неструктуровані зміни елементів і зв'язків цих систем часто призводять до неефективності їх функціонування. Отже, для системної розробки системи необхідно дослідити проблеми та цілі розроблювальної системи, у цьому нам поможуть наступні методи системного аналізу: дерево проблем та дерево цілей.

1.1. Побудова дерева проблеми та дерева цілей

Першим кроком аналізу проблеми є здійснення декомпозиції, тобто декомпозиція проблеми на її складові частини. Визначається основна проблема яка має глобальний характер, після чого здійснюється декомпозиція цієї основної проблеми. Це робиться для того, щоб отримати основну структуру проблематики системи.

Отже, побудуємо дерево проблем для даної системи:

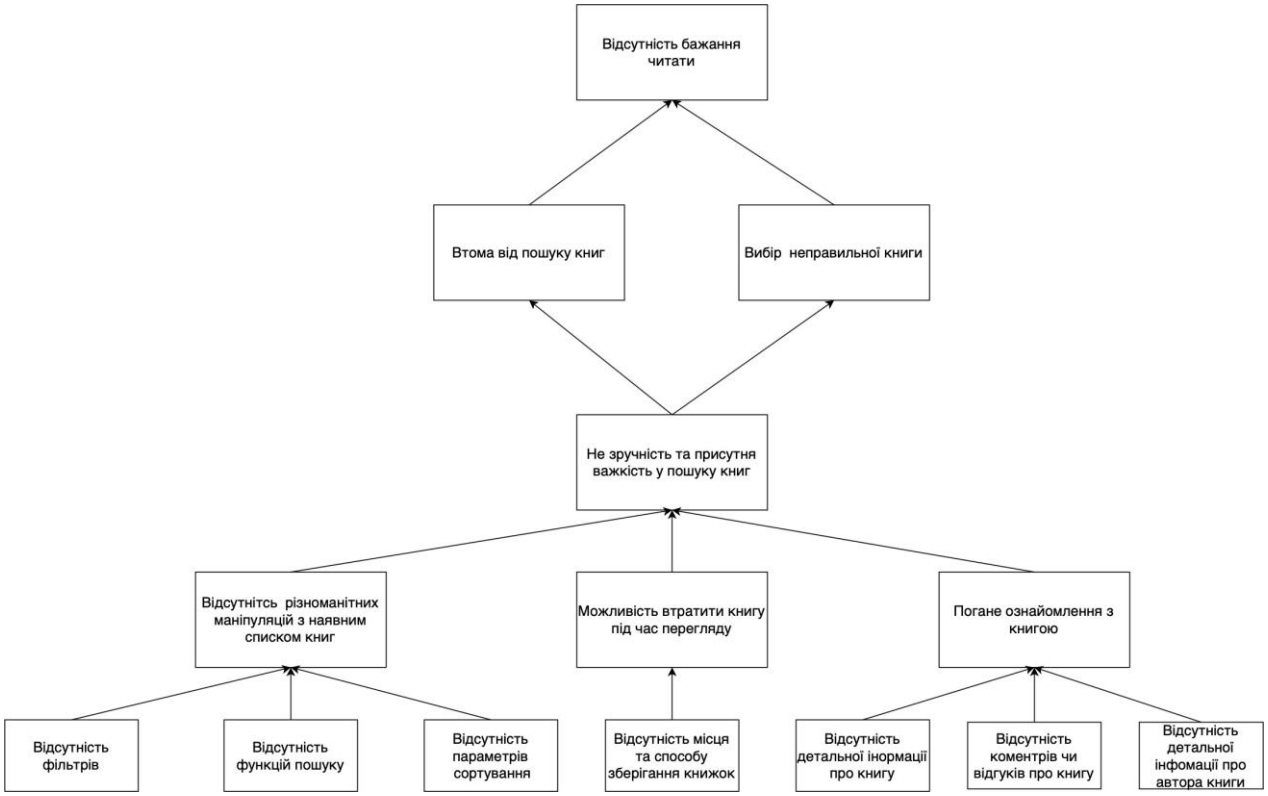


Рис. 3.1.1 - Графічне зображення дерева проблем

Як видно з Рис. 3.1.1 основною проблемою системи є незручність та присутня важкість у пошуку книг. Дана проблема спричинена під проблемами, здійснимо опис кожної з них:

1) відсутність різноманітних маніпуляцій з наявним списком книг. Це дійсно вагома проблема, адже для зручності пошуку і ознайомлення з книгами потрібно надавати функції які б дозволяли скорочувати список, точнішим вказанням параметрів пошуку. Дана проблема спричинена наступними чинниками:

1.1) відсутність фільтрів. Фільтри є дуже важливим способом, для задання критеріїв для уточнення пошуку книг. Відсутність фільтрів дуже не бажана, адже несе незручність в пошуку книгах по жанрах, або рейтингу;

1.2) відсутність пошуку. Функція пошуку є дуже важлива, адже це найшвидший спосіб знайти кнопку знаючи її назву, або деякі слова з назви;

1.3) відсутність сортування. Є важливою функцію, адже спрощує відображення списку книг і дозволяє краще розглядати його з обраним порядком;

2) можливість втратити книгу під час перегляду. Ця проблема пов’язана з тим, що коли користувач знайшов якусь книгу, що зацікавила його і переглядає її може раптово загубити її, наприклад випадково закрити браузерне вікно. Дана проблема має наступний чинник:

2.1) відсутність місця та способу зберігання книжок. Користувачам потрібно надати можливість збереження тих чи інших книг, де вони б пізніше могли їх переглянути.

3) погане ознайомлення з книгою. Дана проблема є важливою, адже для повноцінного ознайомлення з книгою потрібна детальна інформація про книгу, як наприклад короткий опис сюжету, рік видання і так далі. Ця проблема випливає з наступних:

3.1) відсутність детальної інформації про книгу. Тобто не надається короткої інформації про сюжет книги, рік видання, рейтинг, жанр, автора книги;

3.2) відсутність коментарів чи відгуків про книгу. Відгуки залишені іншими користувачами є також важливими атрибутами в процесі ознайомлення з книгами;

3.3) відсутність детальної інформації про автора книги. Окрім інформації про саму книгу, також важливо розміщувати інформацію про автора книги, адже користувача можуть зацікавити інші книги автора.

Побудувавши та проаналізувавши дерево проблем, можна переходити до побудови дерева цілей.

Дерево цілей — це впорядкована ієрархія цілей, які характеризують залежності та взаємозв’язки.

Одним з основних завдань при побудові ДЦ є встановлення повної в'язки, це означає забезпечити досягнення загальної мети побудови систем і виявлення зв'язків між цими засобами. Тому ДЦ служить єдиною, але дуже детальною ціллю для новоствореної системи. Дерева корисні як засіб представлення існуючої ієрархії в системі. Корінь дерева ідентичний всій системі, а рівні дерева

є підсистемами та елементами системи.

Так само в ДЦ корінь відповідає загальній меті, а вершини, що залишилися після розкладання, відносяться до менших цілей. Цілі стають все більш і більш фрагментованими, коли ви рухаєтеся вниз по рівню ДЦ. Розподіл загальної мети на більш дрібні цілі триває до тих пір, поки не стане можливим пов'язати цілі нижчого рівня ієрархії ДЦ із заходами, що забезпечують досягнення цих цілей. Перевага декомпозиції очевидна, оскільки глобальна родова мета в принципі не може бути пов'язана із засобами її досягнення.

Отже, спочатку потрібно виділити генеральну ціль системи, дана система створюється для забезпечення віддаленої роботи бібліотеки як веб-сервісу, тому генеральною ціллю системи є розробка системи, а саме ефективного веб-сервісу для керування роботи бібліотеки та забезпечення віддаленого доступу користувачів через зручний користувацький інтерфейс.

Дана система матиме наступні функції:

- швидка фільтрація, пошук, та сортування списку книжок;

- залишення, видалення та коригування відгуків про ту чи іншу книгу, з можливістю залишення відповідей на коментарі;

- відображення детальної інформації про книгу з підбором схожих книжок за тематикою;

- реєстрація та авторизація користувачів;

- можливість керування контентом бібліотеки (добавлення, редагування та видалення книжок);

- відображення детальної інформації про авторів з підбором найкращих книжок автора;

- функція Bookshelves (книжкові полички), дозволяє користувачам зберігати книжки на віддалених книжкових полицях для подальшого їх читання або зберігання як колекції.

Отже, побудуємо дерево цілей:

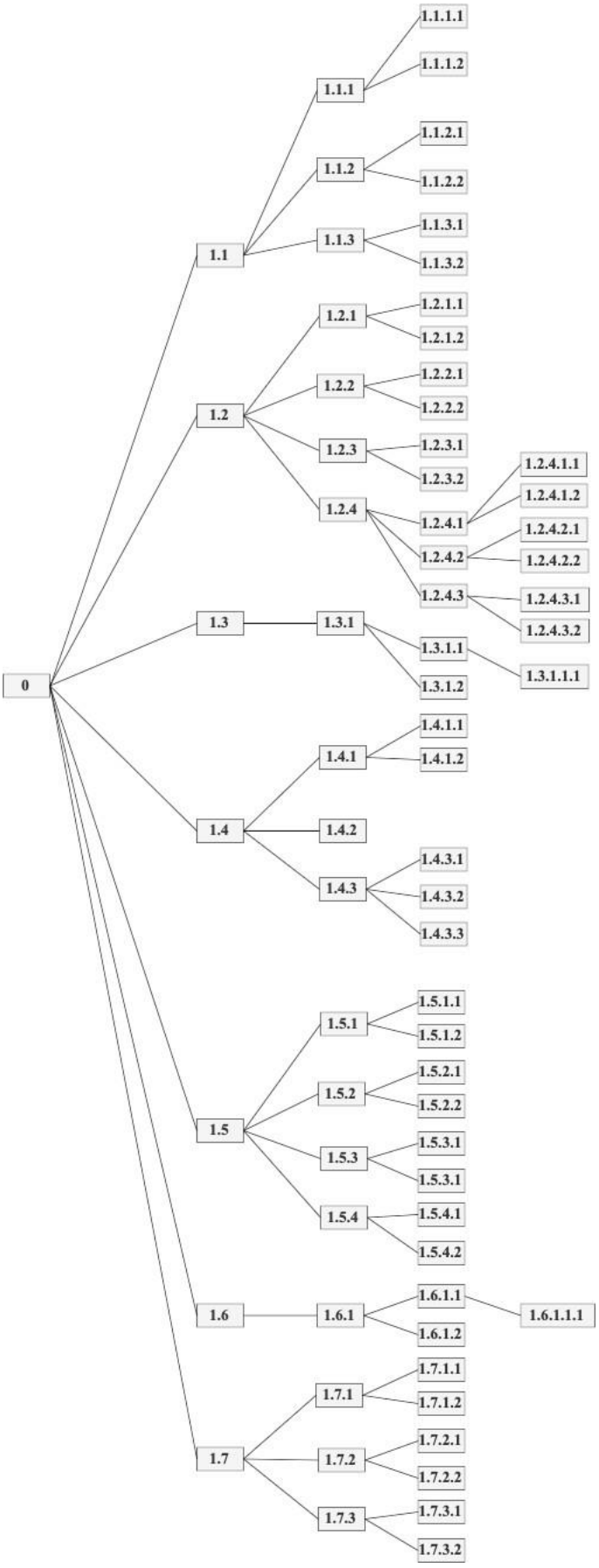


Рис. 3.1.2 - Графічне зображення дерева цілей

Наведемо опис дерева цілей зображеного на Рис. 3.1.2:

0. Генеральна ціль - розробка системи, а саме ефективного веб-сервісу для керування роботи бібліотеки та забезпечення віддаленого доступу користувачів через зручний користувацький інтерфейс.

1.1 Створення модуля, що буде забезпечувати різноманітні маніпуляції з списком книг:

1.1.1 створення підмодуля, що буде забезпечувати ефективну фільтрацію списку книг по заданим параметрам:

1.1.1.1 створення ендпоінту, що дозволить здійснювати фільтрацію по заданим параметрам;

1.1.1.2 створення компоненти користувацького рівня, що буде здійснювати запит на фільтрацію з обраними параметрами фільтрації та подальшим їх відображенням.

1.1.2 створення підмодуля, що буде забезпечувати ефективний пошук книжок, з можливістю пошуку по декільком полям:

1.1.2.1 створення ендпоінту, що буде забезпечувати ефективний пошук книжок, з можливістю пошуку по декільком полям;

1.1.2.2 створення компоненти користувацького рівня, що буде здійснювати запит на пошук книжок з подальшим відображенням.

1.1.3 створення підмодуля, що буде забезпечувати ефективне сортування книжок по заданим параметрам:

1.1.3.1 створення ендпоінту, що буде забезпечувати ефективне сортування книжок;

1.1.3.2 створення компоненти користувацького рівня, що буде здійснювати запит на сортування по заданим параметрам.

1.2 Створення модуля залишення коментарів, відгуків для конкретної книги:

1.2.1 створення підмодуля для можливості добавлення коментаря для конкретної книги:

1.2.1.1 створення ендпоінту, що буде забезпечувати добавлення коментаря для конкретної книги;

1.2.1.2 створення компоненти користувацького рівня, що буде здійснювати запит на добавлення нового коментаря.

1.2.2 створення підмодуля для можливості видалення залишеного коментаря для конкретної книги:

1.2.2.1 створення ендпоінту, що буде забезпечувати видалення коментаря для конкретної книги;

1.2.2.2 створення компоненти користувацького рівня, що буде здійснювати запит на видалення коментаря.

1.2.3 створення підмодуля для можливості коригування вже залишеного коментаря для конкретної книги:

1.2.3.1 створення ендпоінту, що буде забезпечувати коригування коментаря для конкретної книги;

1.2.3.2 створення компоненти користувацького рівня, що буде здійснювати запит на редагування коментаря.

1.2.4 створення модуля залишення відповідей для конкретного коментаря, що був залишений під конкретною книгою:

1.2.4.1 реалізація можливості добавлення відповіді для конкретного коментаря, що був залишений під конкретною книгою:

1.2.4.1.1 створення ендпоінту, що буде забезпечувати добавлення відповіді для конкретного коментаря;

1.2.4.1.2 створення компоненти користувацького рівня, що буде здійснювати запит на добавлення нового відгуку для конкретного коментаря.

1.2.4.2 реалізація можливості видалення відповіді:

1.2.4.2.1 створення ендпоінту, що буде забезпечувати видалення відповіді для конкретного коментаря;

1.2.4.2.1 створення компоненти користувацького рівня, що буде здійснювати запит на добавлення нової відповіді для конкретного коментаря.

1.2.4.3 реалізація можливості коригування вже залишеної відповіді:

1.2.4.3.1 створення ендпоінту, що буде забезпечувати коригування відповіді для конкретного коментаря;

1.2.4.3.1 створення компоненти користувацького рівня, що буде здійснювати запит на коригування відповіді для конкретного коментаря.

1.3 Створення модуля відображення детальної інформації про книгу з підбором схожих книжок за тематикою:

1.3.1 створення підмодуля для отримання та відображення детальної інформації про книгу:

1.3.1.1 створення ендпоінту, що буде забезпечувати повернення детальної інформації про книгу з бази даних:

1.3.1.1.1 реалізація сервісу, що здійснюватиме підбір книг за схожою тематикою та сортуватиме їх по рейтингу з подальшим включенням у відповідь.

1.3.1.2 створення компоненти користувацького рівня, що буде здійснювати запит на отримання детальної інформації про книгу, та буде відображати отриману інформацію.

1.4 Створення модуля реєстрації та авторизації користувачів:

1.4.1 реалізація підмодуля для забезпечення реєстрації користувачів:

1.4.1.1 створення ендпоінту, що буде забезпечувати реєстрацію нового користувача, з додаванням в базу даних;

1.4.1.2 створення компоненти користувацького рівня, що буде здійснювати запит на реєстрацію нового користувача.

1.4.2 створення та виділення окремих ролів для користувачів;

1.4.3 реалізація підмодуля для забезпечення можливості авторизації користувачів:

1.4.3.1 створення ендпоінту, що буде забезпечувати авторизацію користувача;

1.4.3.2 створення компоненти користувацького рівня, що буде здійснювати запит на авторизацію користувача;

1.4.3.3 реалізація обмежень функцій системи опираючись на поточну роль користувача.

1.5 Створення підсистеми, що буде здійснювати контроль контентом бібліотеки:

1.5.1 створення модуля, що дозволить добавляти нові книжки до системи:

1.5.1.1 створення ендпоінту, що буде забезпечувати додавання нової книги в бібліотеку;

1.5.1.2 створення компоненти на представницькому рівні, що буде здійснювати запит на додавання книги в бібліотеку.

1.5.2 створення модуля, що дозволить оновлювати інформацію про вже існуючі книги в системі:

1.5.2.1 створення ендпоінту, що буде забезпечувати оновлення книги в бібліотеці;

1.5.2.2 створення компоненти на представницькому рівні, що буде здійснювати запит на оновлення книги в бібліотеці.

1.5.3 створення модулю, що дозволить видаляти обрану книгу з бібліотеки:

1.5.3.1 створення ендпоінту, що буде забезпечувати видалення книги в бібліотеці;

1.5.3.2 створення компоненти на представницькому рівні, що буде здійснювати запит на видалення книги з бібліотеки.

1.5.4 створення модуля валідування введених даних при оновленні та добавлені книги:

1.5.4.1 визначення та створення правил валідації;

1.5.4.2 відображення відповідних повідомлень валідації при недотримані тих чи інших заданих правил валідації.

1.6 Створення модуля відображення детальної інформації про авторів з підбором найкращих книжок автора:

1.6.1 створення підмодуля для отримання та відображення детальної інформації про книгу:

1.6.1.1 створення ендпоінту, що буде повертати детальні дані з бази даних про книгу:

1.6.1.1.1 пошук книг автора з сортуванням та включенням їх у відповідь.

1.6.1.2 створення компоненти користувацького рівня, що буде надсилати запит на отримання даних та відображати їх.

1.7 Створення модуля, що дозволятиме користувачам зберігати книжки бібліотеки на віддалених книжкових полицях для подальшого їх читання або зберігання як колекції:

1.7.1 створення підмодуля для можливості додати нову книжкову поличку:

1.7.1.1 створення ендпоінту, що дозволить добавляти нову книжкову поличку;

1.7.1.2 створення компоненти користувацького рівня, що буде надсилати запит на створення нової книжкової полички.

1.7.2 створення підмодуля для можливості оновити книжкову поличку:

1.7.2.1 створення ендпоінту, що дозволить оновити книжкову поличку;

1.7.1.2 створення компоненти користувацького рівня, що буде надсилати запит на оновлення нової книжкової полички, та відображати оновлення.

1.7.3 створення підмодуля для можливості видалення книжкової полички:

1.7.3.1 створення ендпоінту, що дозволить видалити книжкову поличку;

1.7.3.2 створення компоненти користувацького рівня, що буде надсилати запит на видалення нової книжкової полички, та відображати видалення.

3.2 Аналіз і вибір методів, алгоритмів та засобів розв'язання задачі

Однією з головних функцій даної системи є зручна та ефективна фільтрація, пошук та сортування книжок. Звичайно ці функції можна реалізувати самостійно, проте реалізація цих функцій системи є досить громісткою та складною, саме тому в розроблювальній системі було використано Azure Cognitive Search.

Azure Cognitive Search — це компонент хмарної платформи Microsoft Azure, що забезпечує можливості індексування та запитів даних, завантажених на сервери Microsoft. Фреймворк призначений для надання розробникам комплексних можливостей пошуку для мобільних і веб-розробників, приховуючи вимоги складності алгоритму пошуку. Azure Cognitive Search був обраний оскільки він здатний швидко обробляти запити та повністю забезпечує функції пошуку, сортування та фільтрації, все що потрібно - це перестворювати індекси до яких потрібно добавляти документи [10].

Також варто звернути увагу на те, що система дозволяє користувачам з роллю «Admin» добавляти, видаляти, та оновлювати книги в бібліотеці таким чином при кожній такій операції потрібно буде здійснювати оновлення чи додавання документа на сервері Microsoft, тобто використання Azure Cognitive Search уповільнює функцію добавлення, оновлення та видалення проте значно пришвидшує операції отримання даних, оскільки операції маніпуляції книгами доступні лише адмінам то даний недолік не буде сильно значимим.

Для впровадження Azure Cognitive Search в бакалаврській роботі створювався окремий модуль, який здійснює запити до серверів Microsoft [8].

Не менш важливим пунктом є відображення усіх даних, клієнт розроблюваної системи створювався за принципом SPA (Single page application). SPA - це принцип, згідно з яким на сайті є тільки один файл html. Тобто лише одна сторінка, яка постійно взаємодіє з користувачем, динамічно переписує поточну сторінку та ніколи не завантажує цілу нову сторінку з сервера [7]. Під час роботи сторінка не оновлюється і не перенаправляє користувача на іншу сторінку. Взаємодія з односторінковими програмами часто передбачає

динамічний зв'язок з веб-серверами. SPA підхід було обрано по наступним причинам:

- швидкий час завантаження: завдяки підходу SPA ваша повна сторінка завантажується швидше, ніж традиційні веб-додатки, оскільки вона має завантажувати сторінку лише за першим запитом;

- розширюваність та простота створення багатофункціональних програм: додаток SPA дозволяє легко додавати розширені функції до веб-програми;

- використовує менше пропускної здатності: не дивно, що SPA споживають менше пропускної спроможності, оскільки вони завантажують веб-сторінки лише один раз. Крім того, вони також можуть добре працювати в районах з повільним підключенням до Інтернету. Отже, ним зручно користуватися всім, незалежно від швидкості Інтернету.

Оскільки більшість функцій системи передбачає тісний зв’язок з базою даних, це означає що необхідно відстежувати усі зміни та своєчасно обробляти всі запити користувачів. Коли витягуються дані в базу даних і з неї, важливо стежити за тим, що ми змінили; інакше ці дані не будуть записані назад до бази даних. Аналогічно ми повинні вставляти нові створені об’єкти та видаляти всі видалені об’єкти [17], [18].

Ми можемо вирішити цю проблему звичайно змінюючи базу даних з кожною зміною своєї об’єктної моделі, але це може призвести до великої кількості дуже малих викликів бази даних, які в кінцевому підсумку будуть дуже повільними. Крім того, це вимагає, щоб транзакція була відкрита для всієї взаємодії, що недоцільно. Щоб уникнути цих проблем був реалізований паттерн UnitOfWork. UnitOfWork відстежує все, що ми робимо, змінюєм під час бізнес-транзакції, що може вплинути на базу даних. Коли ми закінчуємо транзакцію, він визначає все, що потрібно зробити, щоб змінити базу даних в результаті нашої роботи.

3.3. Структурна схема системи з врахуванням інформаційних потоків

Проаналізувавши та вибравши методи для рішення поставлених цілей, можна зобразити структурну схему системи з врахуванням інформаційних потоків.

Структурна схема - це схема, що зображує зв’язки та відповідно взаємодію усіх програмних компонентів чи блоків системи. Дана схема також показує, чи є у системі поділ на підсистеми та відповідний зв’язок між ними. Компоненти або блоки зображені в даній схемі являють собою програми, підсистеми, програмне рішення, що виконує ту чи іншу поставлену задачу.

Інформаційні потоки в структурній схемі вказується на конкретний зв’язок, між компонентами та послідовність їхнього виконання. Іншимисловами через інформаційні потоки передаються дані між блоками, в результаті чого, блоки обробляють дану їм інформацію і передають її на виконання іншим блокам.

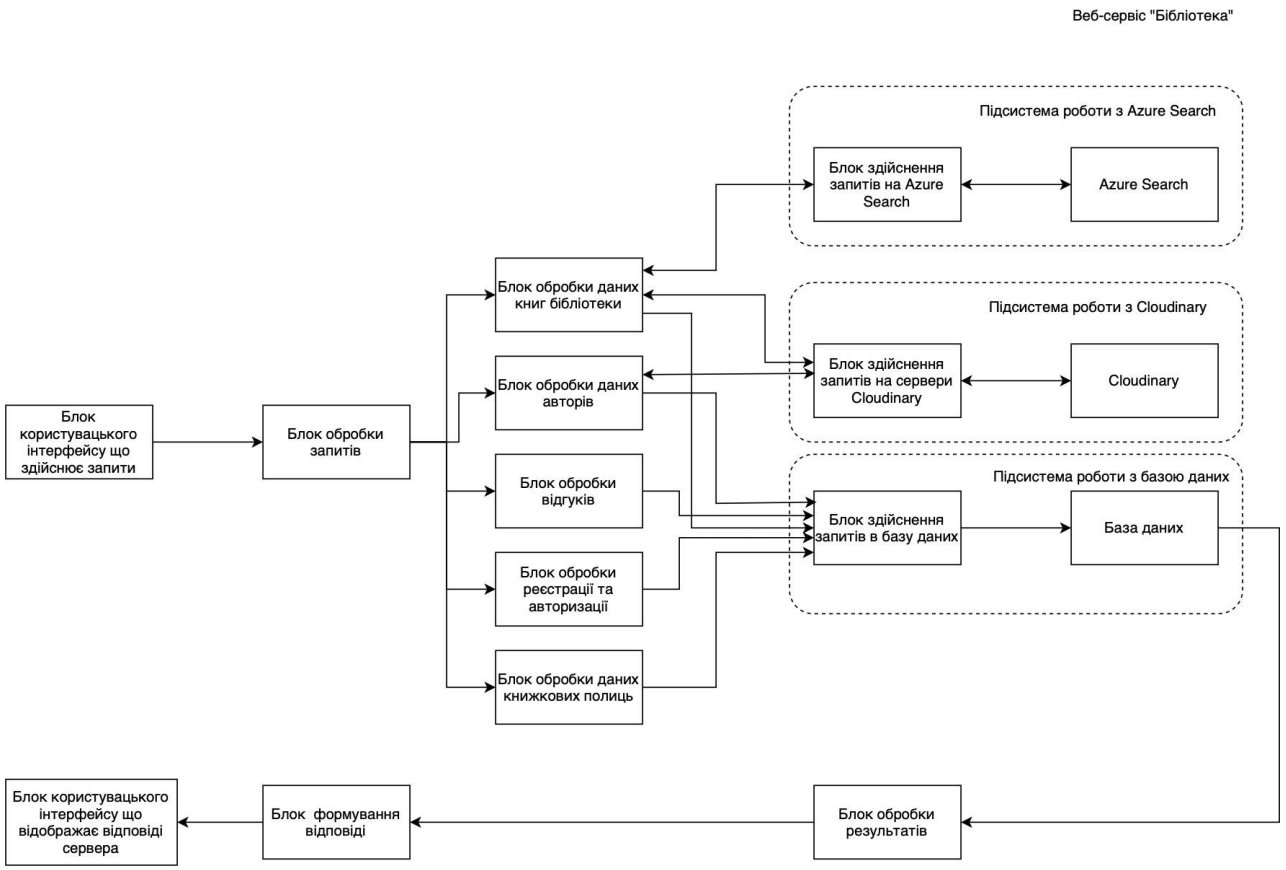


Рис. 3.2 - Структурна схема розроблювальної системи

Як видно з Рис. 3.2 структурна схема розроблювальної системи складається з 3 підсистем по 2 блоки у кожній та ще 10 блоків.

Розглянемо схему детальніше:

1) блок користувацького інтерфейсу, що здійснює запити. Даний блок являє собою, компонент представницького рівня, що здійснює HTTP запити до розробленого серверу;

2) блок обробки запитів. Даний блок являє собою, початок серверної частини, що обробляє вхідні HTTP запити, отримує необхідні параметри та передає їх далі в відповідні блоки обробки цих параметрів;

3) блок обробки даних книг бібліотеки. Даний блок на основі отриманих даних може здійснювати отримання списку книг, видалення книги, добавлення нової книги, оновлення книги по заданій ID. Якщо виконання запиту передбачає виконання операцій фільтрації, пошуку чи сортування то даний блок починає взаємодіяти з підсистемою «Підсистема роботи з Azure Search». Якщо виконання запиту передбачає збереження нової обкладинки для книги, то даний блок починає взаємодіяти з підсистемою «Підсистема роботи з Cloudinary».Любі зміни в системі повинні фіксуватись, тому даний блок взаємодіє з підсистемою «Підсистема роботи з базою даних»;

4) блок обробки даних автора. Даний блок на основі отриманих даних може здійснювати отримання автора по ID, видалення автора, добавлення нового автора, оновлення інформації про автора. Якщо виконання запиту передбачає збереження нової аватарки для автора, то даний блок починає взаємодіяти з підсистемою «Підсистема роботи з Cloudinary». Любі зміни в системі повинні фіксуватись, тому даний блок взаємодіє з підсистемою «Підсистема роботи з базою даних»;

5) блок обробки відгуків. Даний блок на основі отриманих даних може здійснювати отримання списку відгуків для вказаної книги, видалення відгуку, добавлення нового відгуку, оновлення відгуку. Любі зміни в системі повинні фіксуватись, тому даний блок взаємодіє з підсистемою «Підсистема роботи з базою даних»;

6) блок обробки реєстрації та авторизації. Даний блок на основі отриманих даних може здійснювати реєстрацію, тобто збереження нового користувача в базу даних, та авторизацію користувача. Любі зміни в системі повинні фіксуватись, тому даний блок взаємодіє з підсистемою «Підсистема роботи з базою даних»;

7) блок обробки даних книжкових полиць. Даний блок на основі отриманих даних може створювати нові полиці, видаляти існуючі полиці, та оновлювати вже існуючі полиці. Любі зміни в системі повинні фіксуватись, тому даний блок взаємодіє з підсистемою «Підсистема роботи з базою даних»;

8) блок обробки результатів. Даний блок обробляє результати виконання транзакцій в базі даних, він перевіряє чи всі транзакції закінчились успішно та формує відповідні дані про це;

9) блок формування відповіді. Даний отримує дані про результати та на їх основі формує відповідь сервера, і відправляє ці дані користувачу у відповідь на його запит;

10) блок користувацького інтерфейсу, що відображає відповіді сервера. Отримавши відповідь від сервера даний блок здійснює, ту чи іншу зміну в інтерфейсі оновлюючи дані в залежності від запиту користувача.

Тепер опишемо кожну з підсистем:

1) підсистема роботи з Azure Search. Дана підсистема відповідає за здійснення швидкої фільтрації, сортування та пошуку книг. Підсистема складається з таких блоків:

1.1) блок здійснення запитів на Azure Search. Даний блок здійснює HTTP запити на сервери Azure, та отримує відповідні відповіді від Azure;

1.2) блок Azure Search. Це власне і є сервери Azure, які здійснюють задані операції та надсилають списки документів у відповідь.

2) підсистема роботи з Cloudinary. Дана підсистема відповідає за збереження фотографій авторів, та обкладинок книг. Підсистема складається з таких блоків:

2.1) блок здійснення запитів на сервери Cloudinary. Даний блок здійснює HTTP запити на сервери Cloudinary, та отримує відповідні відповіді від

Cloudinary;

2.2) блок Cloudinary. Це власне і є сервери Cloudinary, де зберігаються фотографії авторів та обкладинки книг [15].

3) підсистема роботи з базою даних. Дана підсистема відповідає за роботу з базою даних, здійснення всіх транзакцій на рівні бази даних. Підсистема складається з таких блоків:

3.1) блок здійснення запитів в базу даних. Даний блок формує ту чи іншу транзакцію і відправляє її до бази даних;

3.2) блок База даних. Даний блок являє собою базу даних, що здійснює виконання усіх транзакцій.

РОЗДІЛ 4. РОЗРОБКА ТА ВІДОБРАЖЕННЯ АЛГОРИТМІВ РОБОТИ КОМПОНЕНТІВ СИСТЕМИ

4.1. Блок-схеми алгоритмів

Блок-схема являє собою, графічне відображення того чи іншого алгоритму розв’язування поставленої задачі, де кожна операція чи потік даних зображується геометричними блоками.

Наведемо алгоритм роботи фільтрації, сортування, пошуку книг.

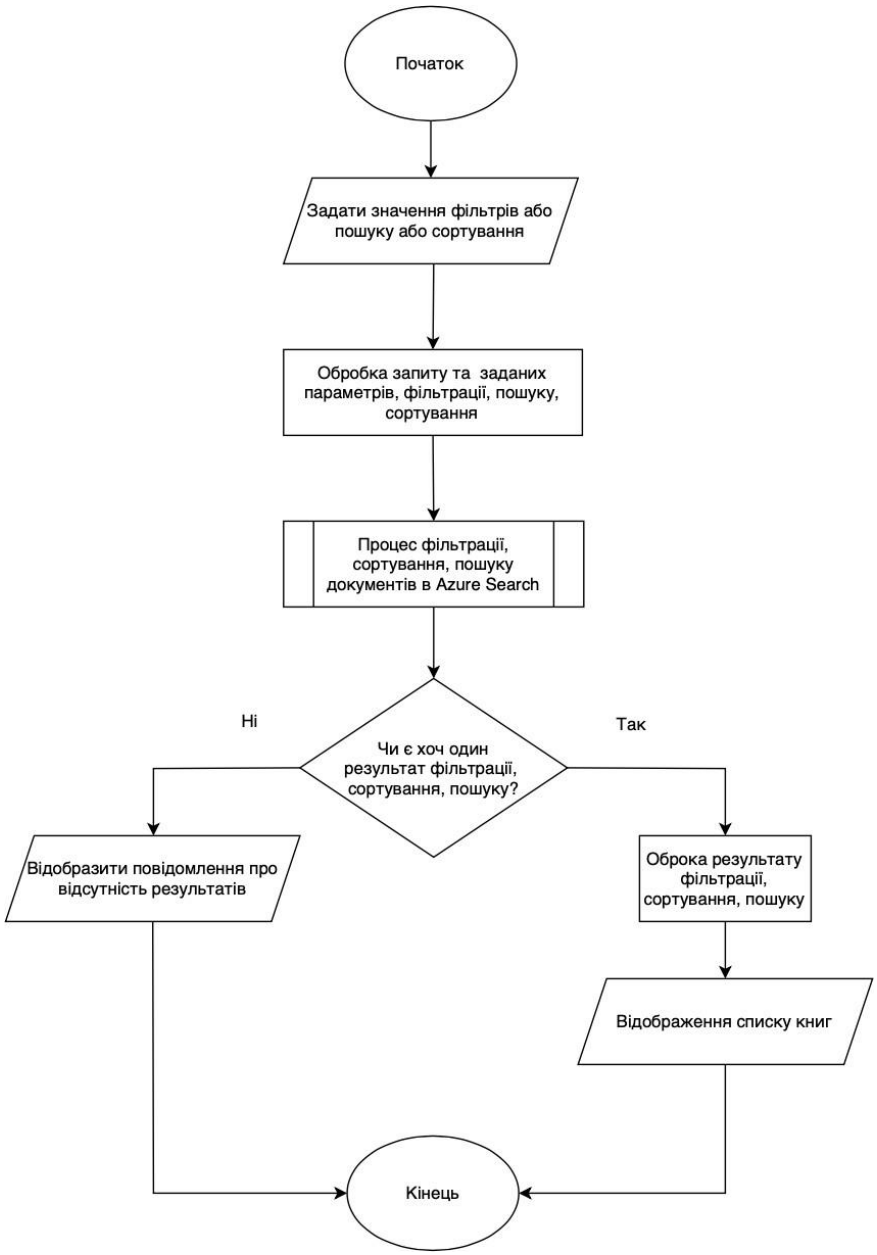


Рис. 4.1 - Блок-схема алгоритму роботи фільтрації, пошуку, сортування книг

Наведений вище алгоритм складається з наступних кроків, спочатку користувач задає відповідні параметри, після цього дані відправляються до сервера де здійснюється обробка отриманих параметрів. Після обробки дані відправляються до Azure Search, який власне буде виконувати вказані операції після чого, сервер отримує відповіді у вигляді списку документів. Якщо список документів пустий, то у такому випадку користувачу відобразиться відповідне повідомлення. Якщо список документів не пустий, то є присутній результат який і буде представлений користувачу.

Розглянемо також алгоритм авторизації та реєстрації користувачів.

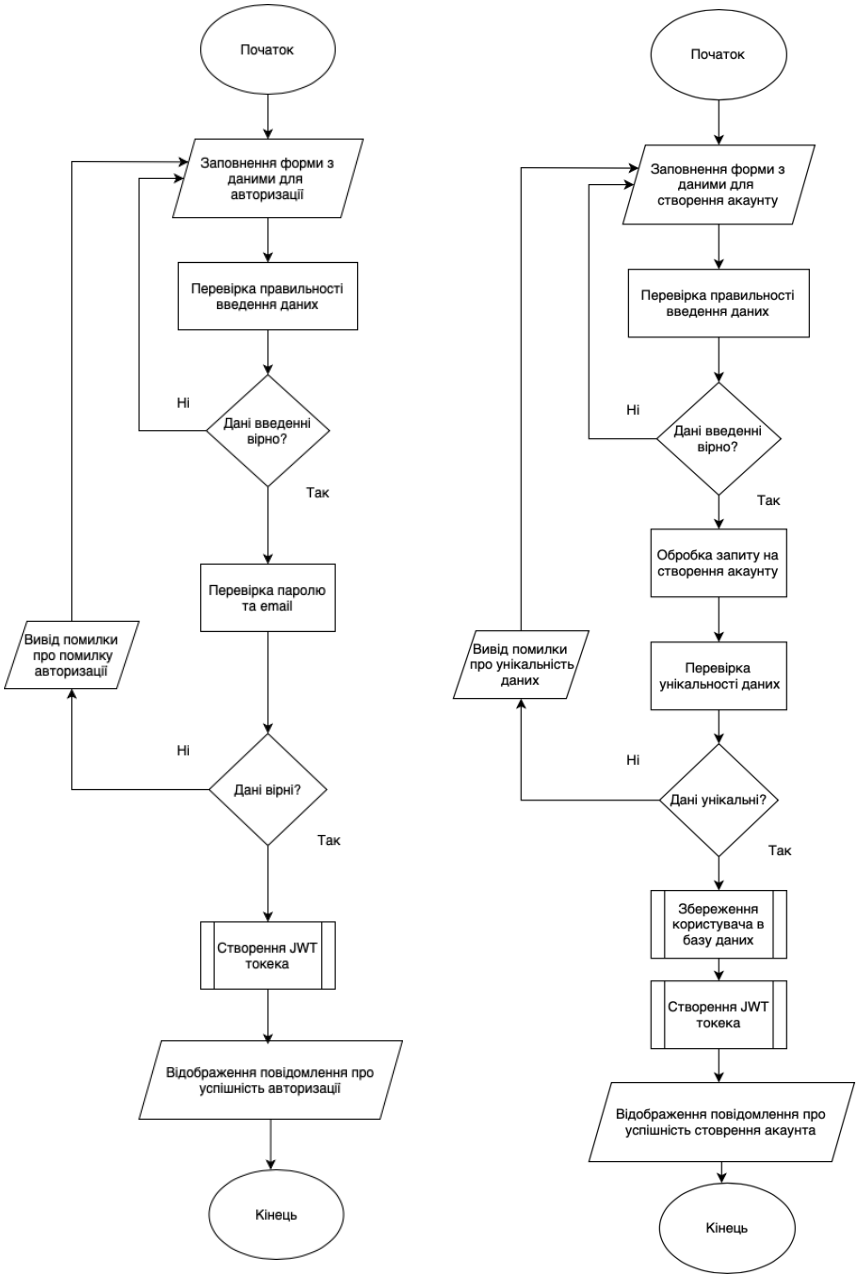


Рис. 4.2 - Блок-схеми алгоритмів реєстрації та авторизації

Як видно з Рис. 4.2 алгоритми авторизації та реєстрації досить схожі проте є і відмінності. У випадку реєстрації користувачу потрібно буде заповнювати форму для створення нового аккаунта, дана форма має валідаційні правила, що постійно перевіряються і у випадку помилок користувачу буде відображатись повідомлення про помилку. Якщо дані введені вірно, то запит відправляється на сервер, де перевіряється унікальність даних, тобто чи введений email, чи username є унікальними в системі, чи під ними вже існують акаунти. У випадку якщо дані не унікальні, то користувачу відобразиться відповідне повідомлення з проханням змінити свої дані. Якщо дані введені вірно, то відбудеться процес збереження даних в базу даних, та створення JWT токену для користувача. Після успішного виконання всіх операцій користувачу буде виводитись повідомлення, про успішність створення нового акаунта.

Що стосується авторизації то форма яку буде заповнювати користувач включатиме лише email та пароль до акаунту, після відправки форми, сервер перевірить чи існує даний користувач та чи вірно введений пароль, і якщо все введено вірно то користувачу створиться новий JWT токен, та відобразиться повідомлення про успішну авторизацію.

Побудовані блок-схеми алгоритмів будуть дуже корисні при програмній реалізації програми оскільки вони точно та послідовно вказують кожен крок алгоритму, що виконує те чи інше завдання.

4.2 Діаграми, які описують функціонування системи та її окремих блоків

Зобразивши та описавши блок-схеми основних алгоритмів, можна починати створювати діаграми, що описуються функціонування системи в цілому або її окремих підсистем чи блоків.

Першою такою діаграмою є діаграма прецедентів. Графічно прецедент намальований у вигляді еліпса. Діаграма прецедентів в UML — це діаграма, яка представляє зв’язки між акторами та прецедентами в системі. Така діаграма описує функціональність, яка надається користувачеві системи, що проектується. Він представлений за допомогою антецедентів і акторів і зв’язків

між ними. Усі попередні набори діаграм фактично визначають функціональні вимоги, на основі яких можуть бути сформульовані технічні завдання.

Контейнмент між антецедентами означає, що в одній «точці» в базовому антецеденті поведінка іншого антецеденту використовується як складова.

Відношення розширення використовуються для моделювання частин антецедентів, які користувач сприймає як необов’язкову поведінку системи.

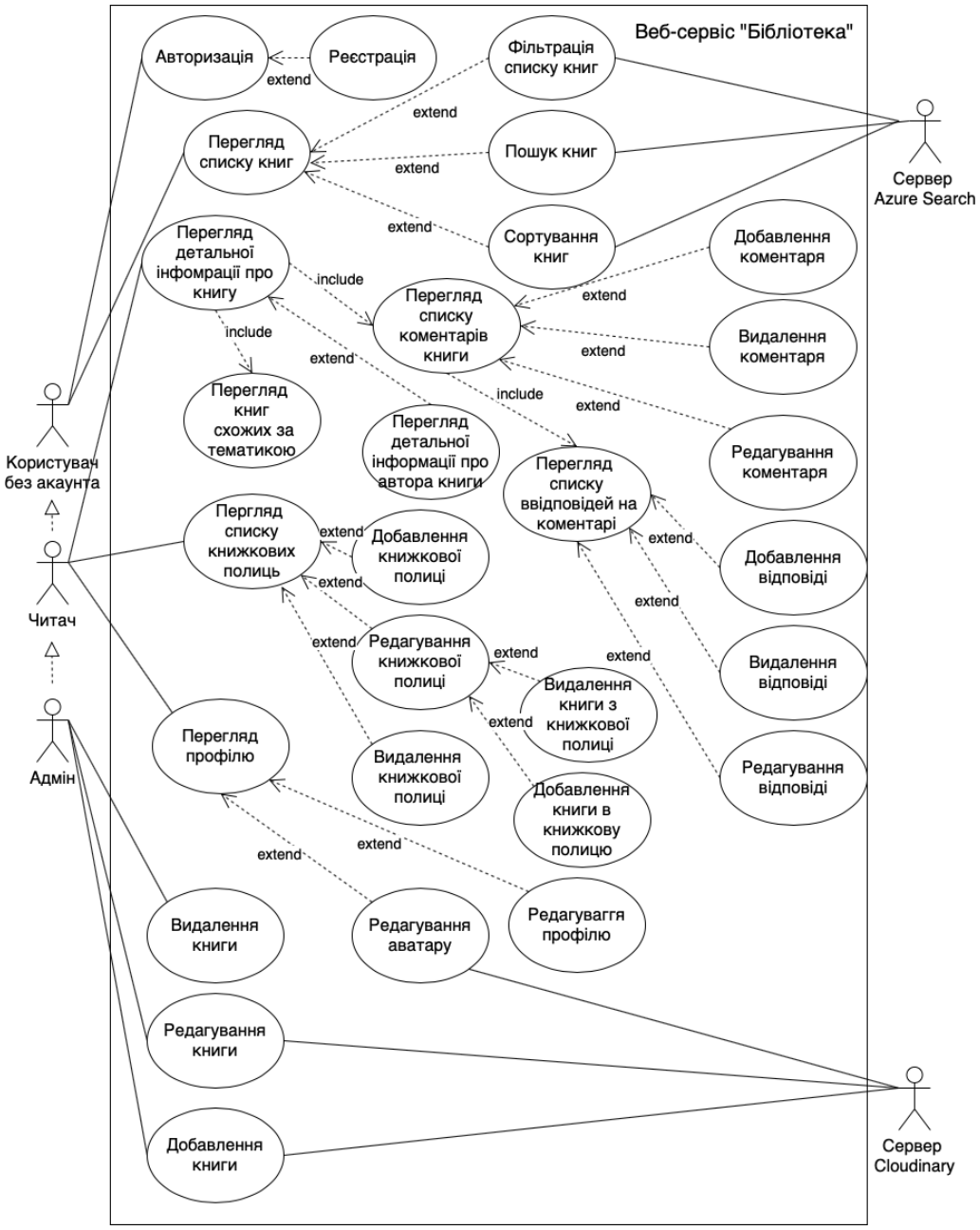


Рис. 4.2.1 - Діаграма прецидентів розроблювальної системи

Як видно з Рис. 4.2.1 на діаграмі зображено усі типи користувачів системи та доступні їм функції. Неавторизований користувач може переглядати список книг, здійснювати операції фільтрації, сортування та пошуку книг, в даних операції також буде брати участь сторонній сервіс а саме Azure Search. Також не авторизованому користувачу доступна функція авторизації та реєстрації.

Наступний тип користувача є читач, тобто авторизований користувач. Авторизований користувач може переглядати детальну інформацію вибраної книги, що завжди включатиме перегляд списку коментарів оскільки вони підтягуються автоматично. Коментарі ж користувач може видаляти, створювати та редагувати. Переглядаючи коментарі користувачу одразу будуть показані усі відповіді під тими чи іншими коментарями, відповіді також можна створювати, добавляти та видаляти. Також при відкриті детальної інформації про книгу, система автоматично пропонуватиме книги з схожою тематикою посортовані за рейтингом. А ще користувач відкривши сторінку з книгою може переглянути інформацію про автора на окремій сторінці.

Також читачу доступна функція перегляду книжкових полиць, з можливістю створювати, видаляти та редагувати книжкові полиці. Крім цього користувач може переглядати власний профіль, змінювати інформацію наприклад, username або ж редагувати свій аватар.

Наступний тип користувача є адміністратор, даний користувач може керувати контентом сайту, а саме добавляти нові книги, видаляти чи редагувати вже додані. При добавленні та редагуванні книг також буде використовуватись сторонній сервіс Cloudinary, для збереження обкладинок книг.

Тепер приступимо до побудови діаграми IDEF. В даній діаграмі система представляється як сукупність робіт або функцій, що взаємодіють разом. Функції системи аналізуються незалежно від об'єктів, якими вони оперують. Цедозволяє більш чітко змоделювати логіку й взаємодію процесів організації.

Починається все з створення контекстної діаграми, де зображується абстракція найвищого рівня системи. В ній зазначається ціль системи, її вхідні та вихідні дані, а також правила якими керується система та що використовує система. Надалі проводять декомпозиції цієї діаграми.

Отже, створимо контексну діаграму.

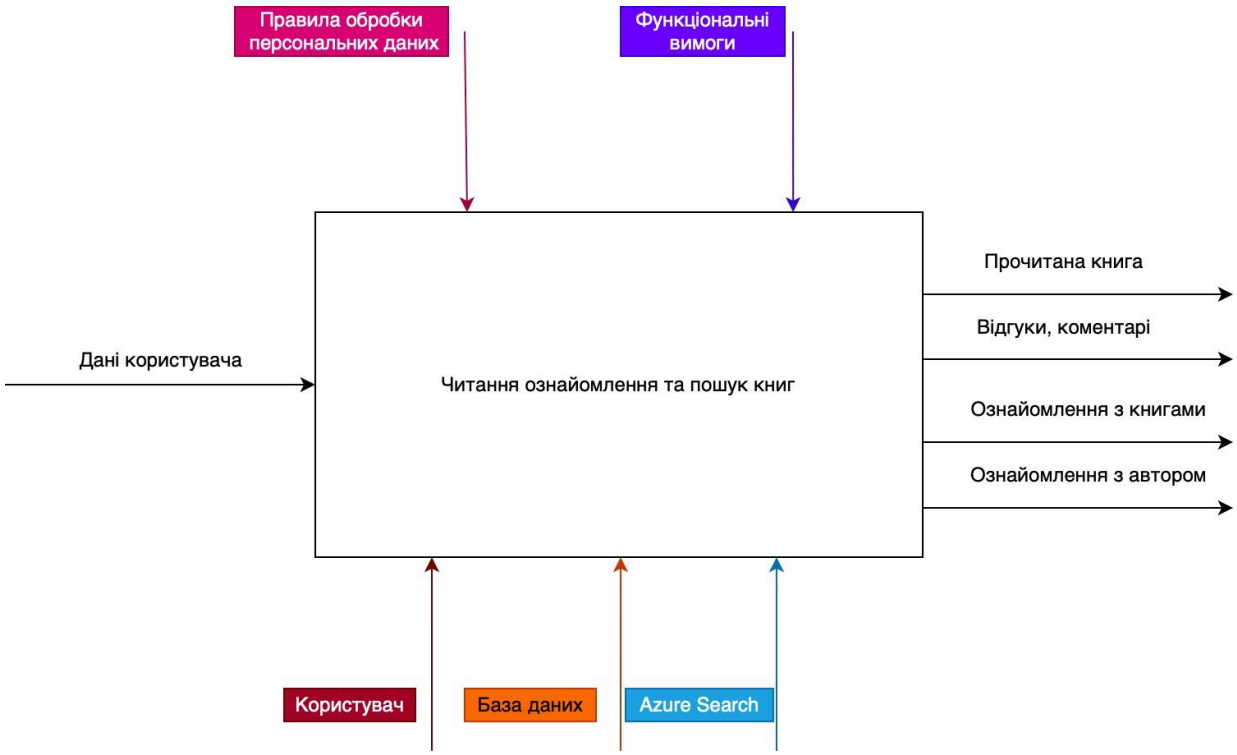


Рис. 4.2.2 - Контексна діаграма

З точки зору цієї діаграми головною ціллю є читання, ознайомлення та пошук книг. Вхідними даними системи є дані користувача, а вихідними прочитана книга, залишені відгуки або коментарі, ознайомлення з книгами та ознайомлення з автором. Щодо правил якими керується система то це є функціональні вимоги та правила обробки персональних даних. Щодо того, що використовує система то це база даних, Azure Search, та користувач.

Тепер проведемо декомпозицію першого рівня. Розбивши контекстну діаграму на під функції при цьому зберігши всі вхідні та вихідні дані та правила якими керується система. Це дозволить бачити як досягається конкретна ціль системи.

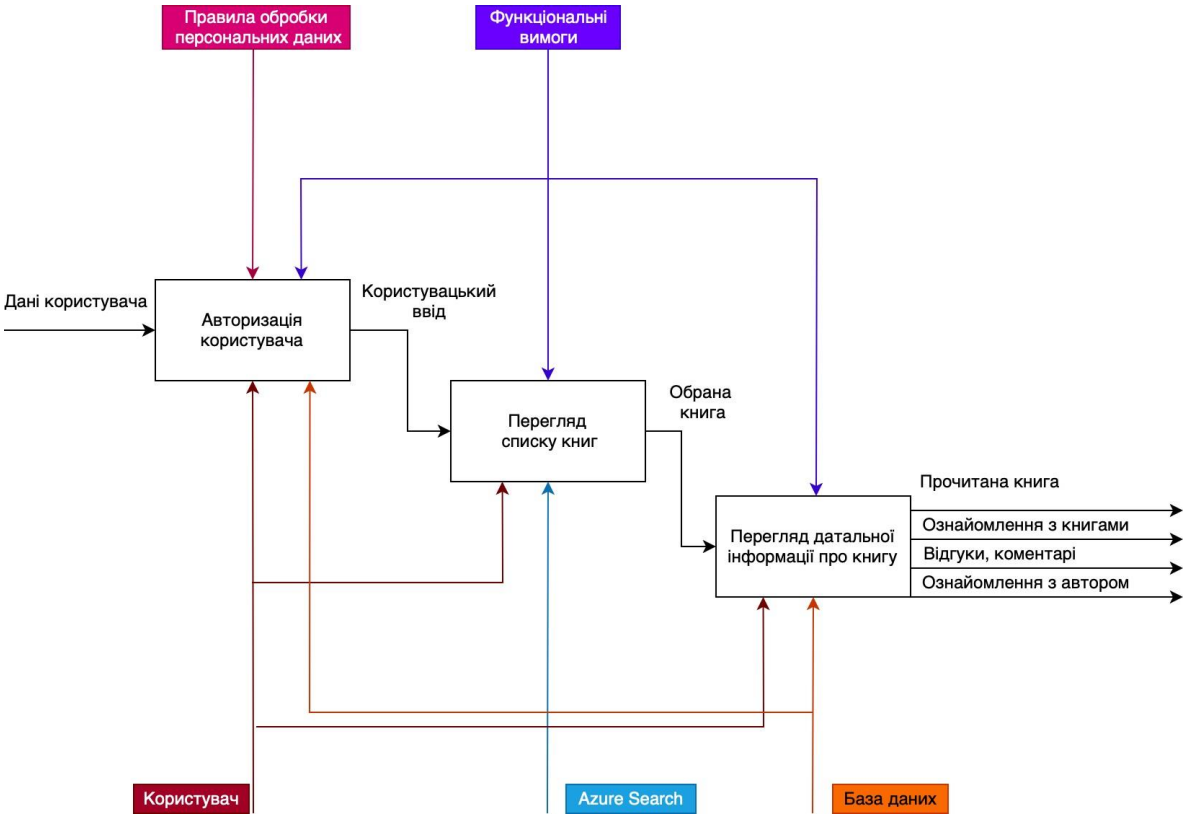


Рис. 4.2.3 - Діаграма декомпозиції 1-го рівня

Провівши декомпозицію першого рівня видно на які функції розкладається система, а саме авторизація користувача, перегляд списку книг та перегляд детальної інформації про книгу. Також кожен модуль має свій вхід та видає щось на вихід. Наприклад переглядаючи список книг користувач вибирає якусь одну тим самим передаючи її в модуль перегляду детальної інформації про цю книгу.

Декомпозиції першого рівня не завжди є достатньо оскільки дані функції можна декомпозувати далі, проведемо декомпозиції для всіх функцій зображених на Рис. 4.2.3.

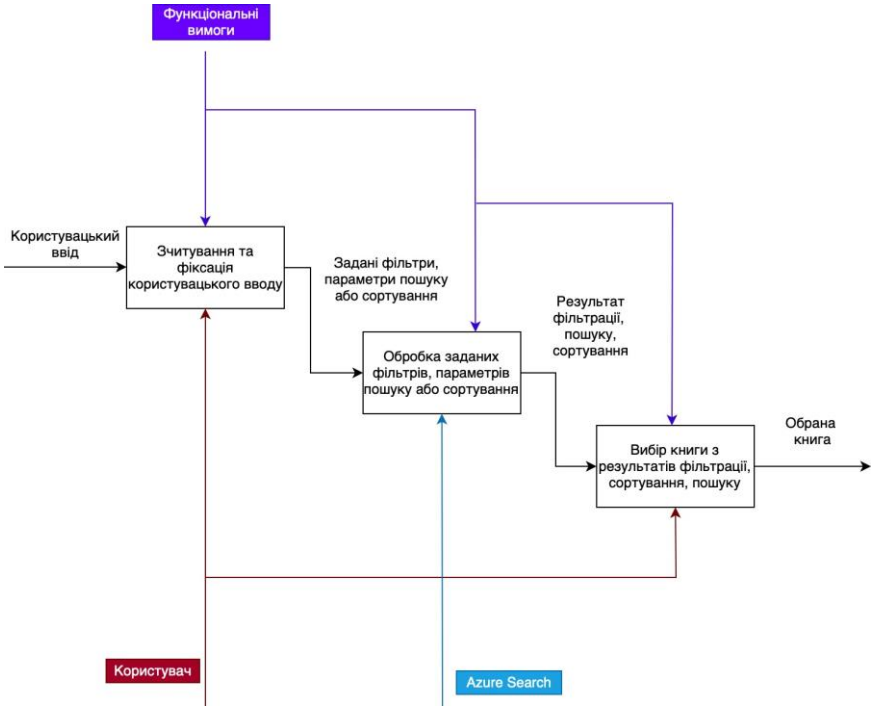


Рис. 4.2.4 - Діаграма декомпозиції 2-го рівня для функації «Перегляд списку книг»

Як видно з Рис. 4.2.4, перегляд списку книг відбувається наступним чином. Спочатку відбувається зчитування користувацького вводу, після чого задані фільтри, параметри пошуку чи сортування обробляються і з результативного списку який переглядає користувач він може вибрати одну книгу.

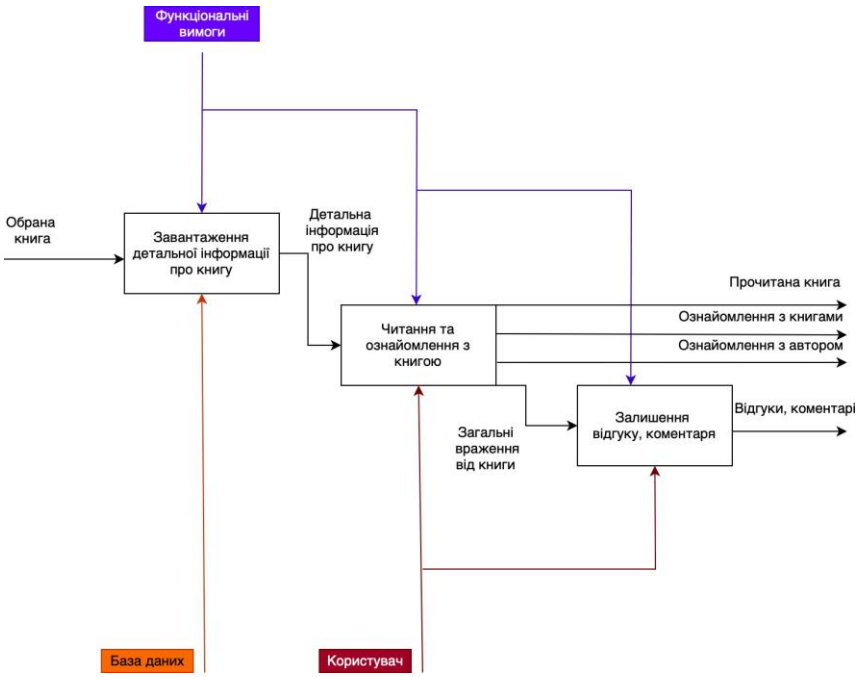


Рис. 4.2.5 - Діаграма декомпозиції 2-го рівня для функції «Перегляд детальної інформації про книгу»

Як видно з Рис. 4.2.5, перегляд детальної інформації про книгу відбувається в наступному порядку. Спочатку відбувається завантаження детальної інформації про книгу, користувач ознайомлюється з поданою інформацією та може почати перегляд книги, після ознайомлення та читання користувач може залишити відгук про книгу.

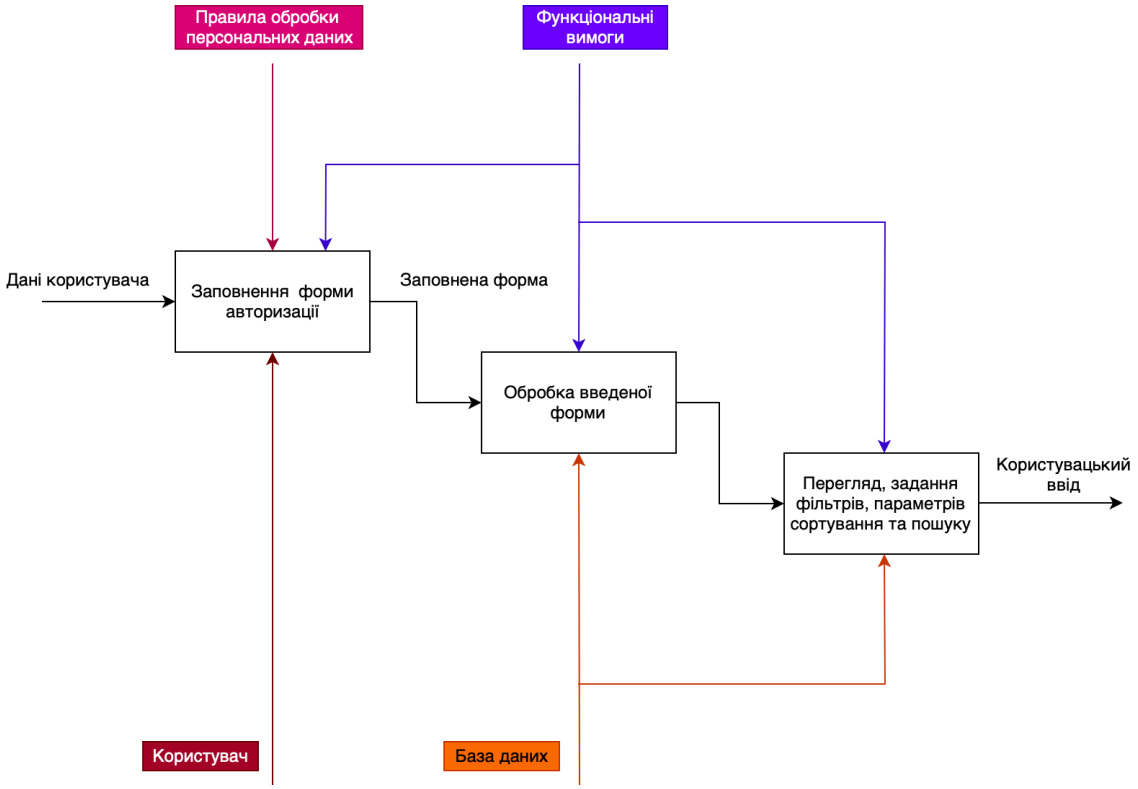


Рис. 4.2.6 - Діаграма декомпозиції 2-го рівня для функції «Авторизація користувача»

Як видно з Рис. 4.2.7 авторизація користувача складається з наступних функцій, а саме спочатку користувач заповнює форму авторизації далі відбувається обробка заповненої форми після чого користувач може задавати фільтри, параметри пошуку чи сортування та переглядати книги.

Наступною діаграмою є діаграма DFD, що розглядає систему як сукупність предметів, у цій діаграмі процеси являють собою функції системи, що перетворюють входи у виходи, самі процеси зображуються заокругленими

прямокутниками. Потоки робіт зображуються стрілками й описують рух об’єктів з однієї частини системи в іншу.

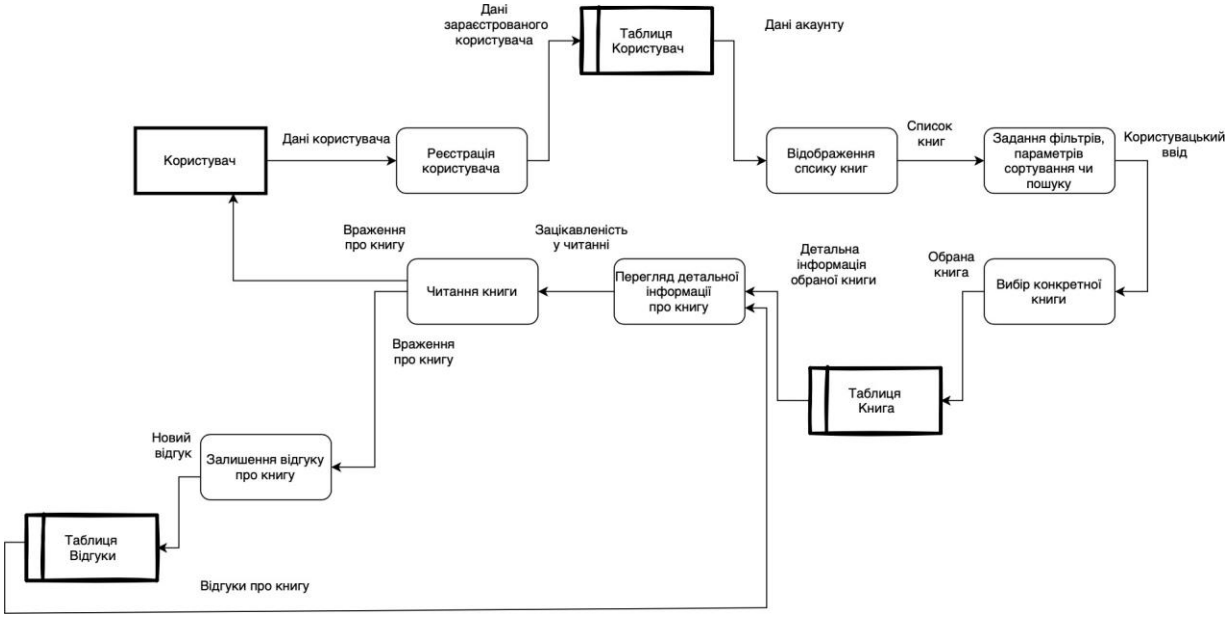


Рис. 4.2.7 - Діаграма DFD

Розглянувши Рис. 4.2.7 видно, що все починається з користувача який передає свої дані та здійснює реєстрацію, після чого його дані заносяться в таблицю Користувач та зберігаються там, зареєструвавшись, дані користувача передаються назад у систему і здійснюється відображення книг. Користувач при цьому може задати певні фільтри або параметри сортування чи пошуку. Цей користувацький ввід фіксується, і користувачу показуються результати. Тоді користувач може обрати одну з запропонованих книг, при цьому перед тим як переглянути детальну інформацію про книгу, потрібно взяти дані з таблиці Книга та з таблиці Відгуки, щоб підтягнути усі відгуки про книгу та лише потім відобразити книгу. Далі користувач ознайомлюється з книгою, та може почати читати її, тоді в нього формуються загальні враження про книгу і він може залишити відгук про неї, який зберігається в таблиці Відгуки.

4.3. Діаграма класів

Діаграми класів зображують різні класи, які разом утворюють систему, а також показує їх взаємозв'язки. Діаграми класів також показує атрибути класів та всі їхні операції або методи.

Класи позначаються прямокутники з їхніми назвами, а вже всередині будуть показані атрибути та операції.

Атрибути відображаються з їх типом, назвою самого атрибута та видимістю. Область дії атрибута: + відповідає публічним атрибутам, # відповідає захищеним атрибутам, - відповідає приватним атрибутам.

Операція — це певна дія або процес, що виконується класом. Вони також відображатимуться, принаймні за назвою, з параметрами, які вони приймають, і значеннями, які вони повертають. Область дії операції така сама, як область дії атрибута.

Розроблювальна система є досить складною тому і має дуже велике число класів, тому діаграма класів буде наведена лише для однієї з підсистем.

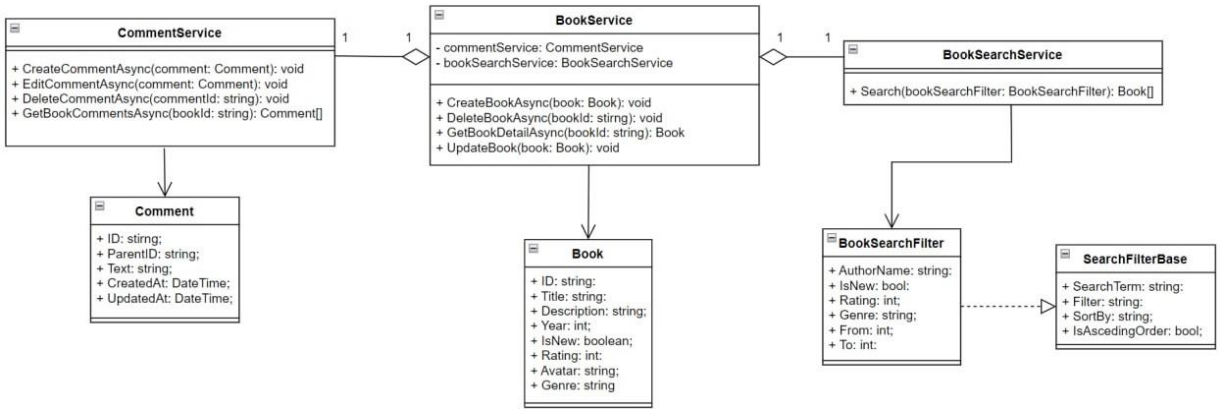


Рис. 4.3.1 - Діаграма класів

На даній діаграмі зображені класи, що виконують бізнес-логіку підсистеми роботи з книгами. BookService призначений для роботи з книгами, а саме виконання таких операцій як додавання, видалення та редагування.

Виконуючи ці операції він взаємодіє з класом Book, що виконує роль моделі, тобто об'єднує в собі всі важливі параметри книги в даній системі. BookService використовує ComentService та BookSearchService.

Перший призначений для того, щоб оперувати коментарями, при цьому він взаємодіє з класом моделлю Comment.

Другий ж призначений для здійснення пошуку книг використовуючи Azure Search, при цьому він використовує клас що зображує фільтр BookSearchFilter який в свою чергу успадковує всі атрибути класу SearchFilterBase.

4.4 Схема бази (сховища) даних

В бакалаврській роботі використовується реляційна база даних. Реляційна база даних – це база даних, в якій усі дані, доступні користувачу, організовані у вигляді таблиць, а всі операції над даними зводяться до операцій над цими таблицями. модель бази даних в даний час є найбільш широко використовуваної моделлю.

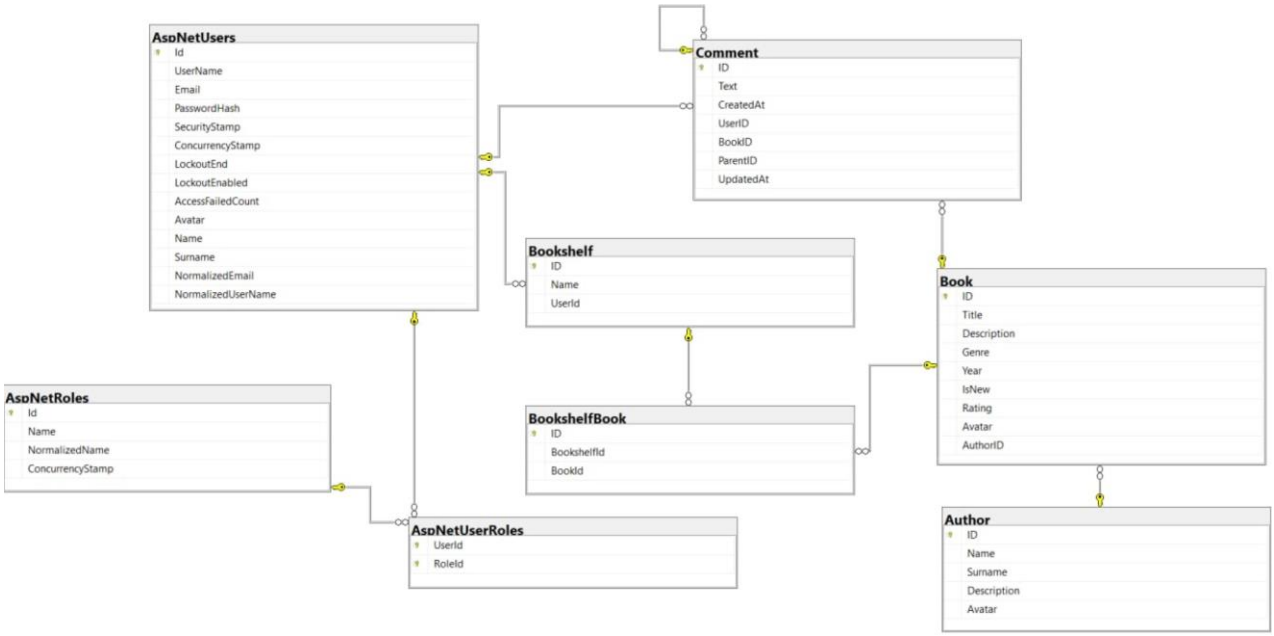


Рис. 4.3.2 - Схема бази даних

На Рис. 4.3.2 зображені всі таблиці бази даних. AspNetUsers це таблиця, що зберігає дані про користувачів та з’єднана з таблицею AspNetRoles зв’язком

багато до багатьох через табличку AspNetUserRoles. Це потрібно тому що один користувач може бути в декількох ролях, і ролі належать декільком користувачам

Також таблиця AspNetUsers має зв’язок з таблицею Comment, даний зв’язок типу один до багатьох. Оскільки всі коментарі належать одному користувачу, проте один користувач володіє списком коментарів. Крім цього AspNetUsers зв’язана з таблицею Bookshelf зв’язком один до багатьох, адже один користувач може мати багато книжкових полиць проте одна книжкова полиця завжди має одного власника.

Таблиця Book має зв'язок один до багатьох з таблицею Comment оскільки, одна книга може мати багато коментарів, проте кожен коментар належить до конкретної книги. Також таблиця Book зв’язана з таблицею Author зв’язком типу один до багатьох, адже книга може мати декілька авторів. Крім цього книга зв'язана зв’язком багато до багатьох з таблицею Bookshelf використовуючи таблицю BookshelfBook. Це потрібно тому що одні і ті самі книги можуть бути додані в різних книжкових полицях.

Також варто зазначити, що таблиця Comment має посилання на самого себе, це зроблено тому що окрім коментарів в системі можна залишати відповіді для коментарів. По своїй суті коментарі та відповідь на коментарі являють собою одну і ту ж саму сутність. Єдиною відмінністю є те, що в коментарях поле ParentId завжди рівне null, а от всі відгуки завжди зберігають Id коментаря для якого вони були залишені.

РОЗДІЛ 5. РОЗРОБКА ПРОГРАМНОГО РІШЕННЯ

5.1 Загальна структура програмного проекту

Бакалаврська робота розроблялась за допомогою мови C# та TypeScript. В роботі розроблені як клієнт який здійснює запити на сервер, так і сервер, що відповідає на запити клієнта. Отже, структура програмного проекту буде показана і для клієнта і для сервера.

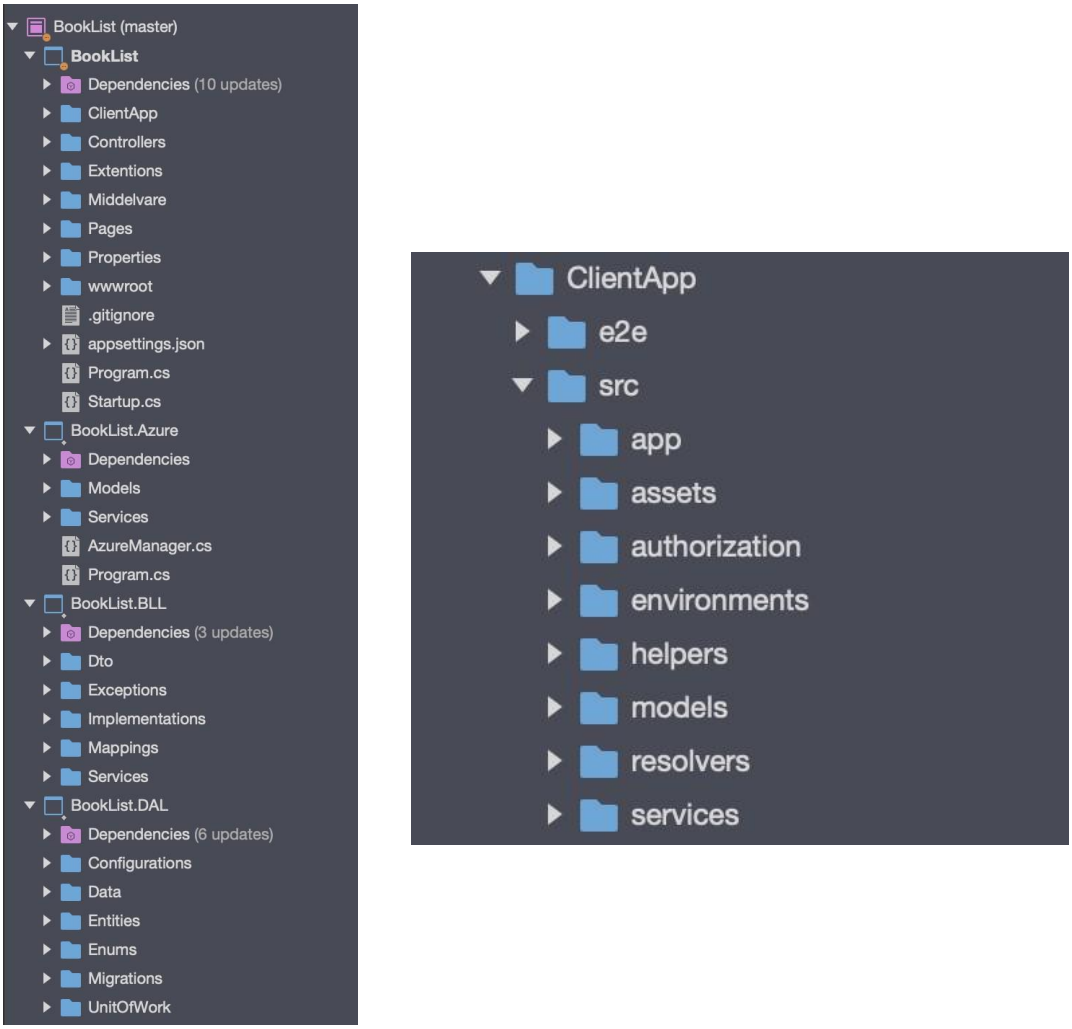


Рис. 5.1.1 - Структура сервера, та структура клієнта

Отже, розпочнемо з опису сервера. В загальному сервер складається з 4 проектів: BookList, BookList.Azure, BookList.BLL, BookList.DAL. Розпишемо кожен з них детальніше:

BookList містить все необхідне для роботи сервера, а також налаштування сервера.

BookList.Azure призначений для виконання запитів на Azure Search для фільтрації, сортування, пошуку книг.

BookList.BLL проект який виконує усю бізнес-логіку проекту.

BookList.DAL проект призначений для виконання транзакцій до бази даних.

Тепер опишемо структуру клієнта, уся структура клієнта зберігається в папці src:

1) папка app - містить всі компоненти клієнта, тобто сторінки які відображаються користувача в браузері;

2) папка assets - папка, що містить всі статичні зображення. Наприклад зображення аватарки користувача за замовчування який не встановив аватар;

3) authorization - папка, що зберігає сервіси які потрібні для перевірки того чи користувач має доступ до тої чи іншої сторінки;

4) environments - містить файл який зберігає посилання на сервер до якого потрібно звертатись;

5) helpers - містить допоміжні методи, та константні повідомлення до користувача;

6) models - містить моделі для спілкування з сервером;

7) services - містить файли сервісів, що здійснюють HTTP запити до сервера, та отримавши відповідь передають їх до компонентів.

Описавши структуру, можна почати опис сторонніх бібліотек, що використовувались у системі та модулів.

5.2 Опис використаних сторонніх бібліотек та модулів

Мовою програмування сервера була об’єктно-орієнтована мова C# написана Microsoft. Сервер створювався на фреймворку ASP.NET Core, що також розроблений командою Microsoft для створення веб-сервісів. Клієнт проектувався на мові TypeScript, використовуючи фреймворк Angular.

Фреймворк ASP.NET Core являє собою кросплатформене середовище з відкритим кодом для. Створення сучасних веб-сервісів з підключенням до Інтернету. Як було зазначено це красплатформений фреймворк, що означає що роботу на ньому можу виконувати на операційних системах Windows, MacOS, Linux.

Entity Framework (EF) Core - це проста, кросплатформенна версія популярної технології доступу до даних Entity Framework з відкритим вихідним кодом, для доступу до бази даних та виконання транзакцій в базі даних. EF Core дозволяє розробникам .NET працювати з базою даних за допомогою об'єктів .NET, усуває необхідність великого обсягу коду для доступу до даних, який зазвичай доводиться писати.

Azure Cognitive Search - це хмарна служба пошуку, яка надає розробникам інфраструктуру, інтерфейси API та засоби для створення розширених можливостей при роботі з вмістом веб-застосунків, а також мобільних та корпоративних додатків. Дана служба розроблена Microsoft і дуже ефективно здійснює пошук, фільтрацію та сортування документів.

Cloudinary API - це сервер який дозволяє зберігати фотографії та різноманітні файли. Крім цього Cloudinary дозволяє змінювати розміри фотографій, обробляти їх та модифікувати.

Фреймворк Angular це по суті платформа для розробки, написана та TypeScrript. Angular повністю заснований на компонентах фреймворк, що дозволяє створювати веб-програми легкі для масштабування, Angular містить набір вже вбудованих бібліотек для маршрутизації, керування формами, клієнт-серверну взаємодію.

Bootstrap - це бібліотека яка дозволяє використовувати деякі готові елементи з вже готовими стилями. Використання цієї бібліотеки спрощує написання розмітки та стилів до неї.

5.3. Розробка та опис програмних модулів

Як вже зазначалось система по суті поділена на 4 модулі. Дана система розроблялась в стилі Layer Architecture. Дана архітектура передбачає поділ на

такі модулі як Presentation Layer (клієнт системи, що здійснює запити), Router Layer (серверна частина що приймає запити та обробляє їх), Service Layer (уся бізнес-логіка системи), Data Access Layer (працює з базою даних).

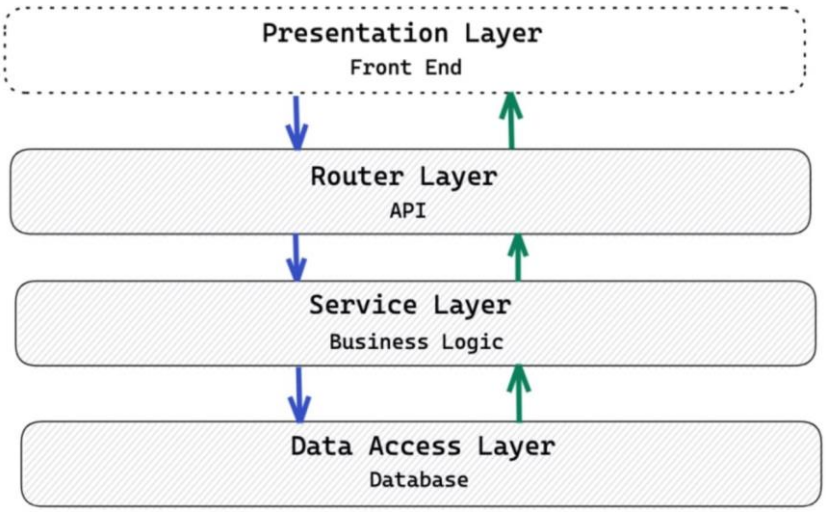


Рис. 5.3.1 - Схема Layer Architecture

Отже, дивлячись на Рис. 5.1.1, Presentation Layer - це розроблювальний клієнт, Router Layer - це BookList проект, Service Layer - це BookList.BLL, Data Access Layer - це BookList.DAL.

Тепер спираючись на Рис. 5.1.1 опишемо основні папки та файли наших модулів.

BookList містить все необхідне для роботи сервера, а також налаштування сервера. Серед важливих файлів та папок в ньому можна виділити наступні:

1) папка Controllers - в даній папці зберігаються всі контролери, що створені на базі фреймворка ASP.NET Core. Власне контролери приймають запити від клієнта та перенаправляють їхнє подальше виконання до BookList.BLL, а вже після обробки формують відповідь та відправляють клієнту;

2) файл Startup.cs - в даному файлі містяться усі конфігурації сервера. Наприклад налаштування підключення до бази даних, налаштування маршрутизації, по якій будуть здійснюватись запити від клієнта;

3) файл Program.cs - являє собою вхідну точку у програму, який запускає сервер перед тим підтягнувши файл Startup.cs.

BookList.Azure призначений для виконання запитів на Azure Search для фільтрації, сортування, пошуку книг. Серед важливих файлів та папок в ньому можна виділити наступні:

1) папка Models - містить класи моделі, для отримання даних з сервера Azure;

2) папка Services - містить сервіси для здійснення запитів та отримання відповідей від сервера Azure Search.

BookList.BLL проект який виконує усю бізнес-логіку проекту. Серед найважливіших папок варто виділити:

1) папка Implementations - папка в якій зберігаються всі сервіси, що виконують ту чи іншу логіку системи. Наприклад добавлення книг, видалення коментарів, авторизація користувача і тому подібне;

2) папка Dto - зберігає класи моделі для обміну між клієнтом та сервером.

BookList.DAL проект призначений для виконання транзакцій до бази даних. Серед папок найбільш важливими можна виділити наступні:

1) папка Entities - в даній папці зберігаються усі сутності бази даних тобто Книга, Користувач, Коментар і так далі;

2) папка Configurations - містить всі конфігурації таблиць бази даних;

3) папка Data - містить сам контекст роботи з базою даних;

4) Папка UnitOfWork - зберігає репозиторії через які власне проходять всі транзакції до бази даних.

5.4. Розробка та опис інтерфейсу користувача

З першим входженням на веб-сайт користувачу відкривається сторінка де зображуються всі книги в бібліотеці Рис. 5.4.1.

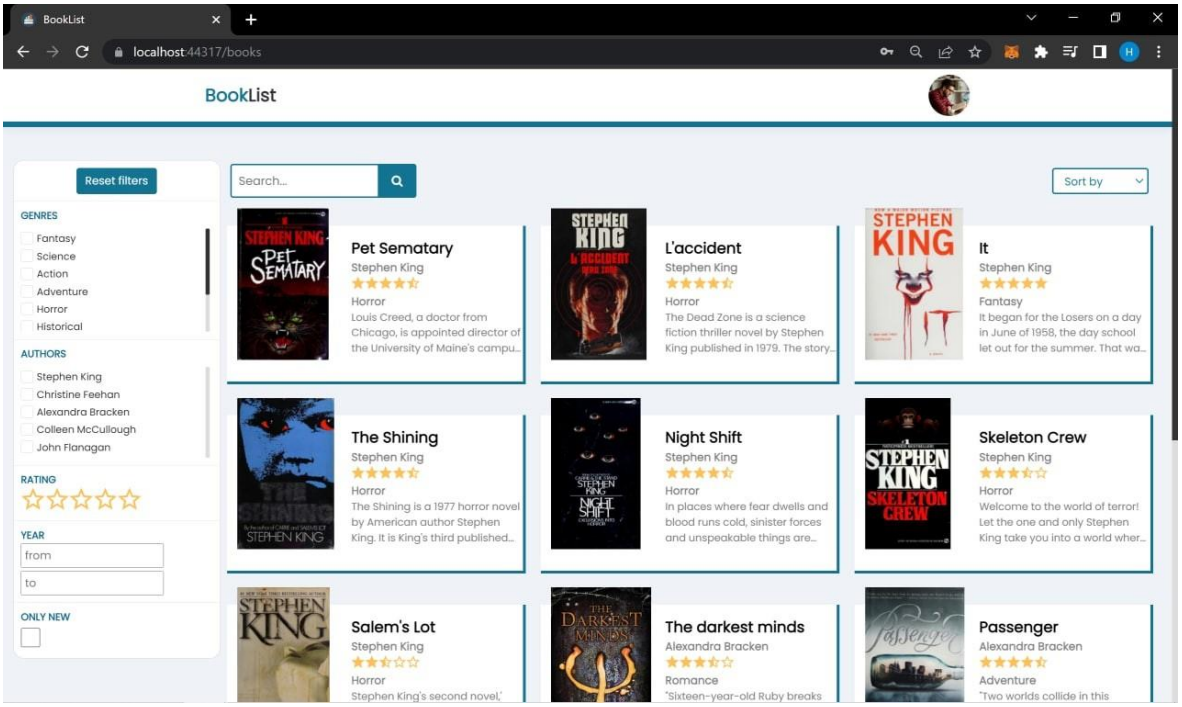


Рис. 5.4.1 - Сторінка перегляду книг

На цій сторінці користувач може здійснювати сортування, пошук, фільтрацію книг. Як тільки користувач натискає ті чи інші фільтри чи здійснює пошук, то здійснюється миттєвий запит до сервера який передає клієнту вже оновлений список книг, що буде відображатись Рис. 5.4.2.

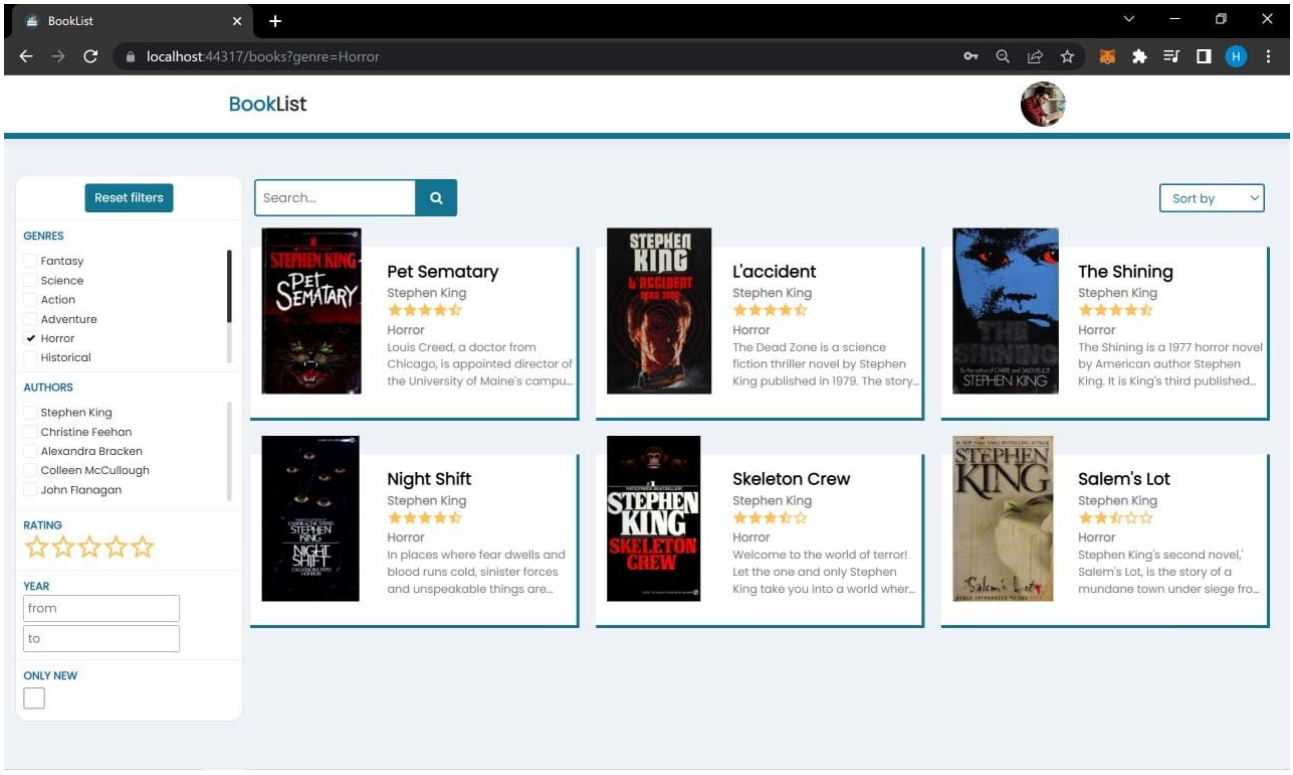


Рис. 5.4.2 - Сторінка перегляду книг з фільтром по жанру «Horror»

Також користувач може одночасно сортувати, шукати, та фільтрувати книги Рис. 5.4.3.

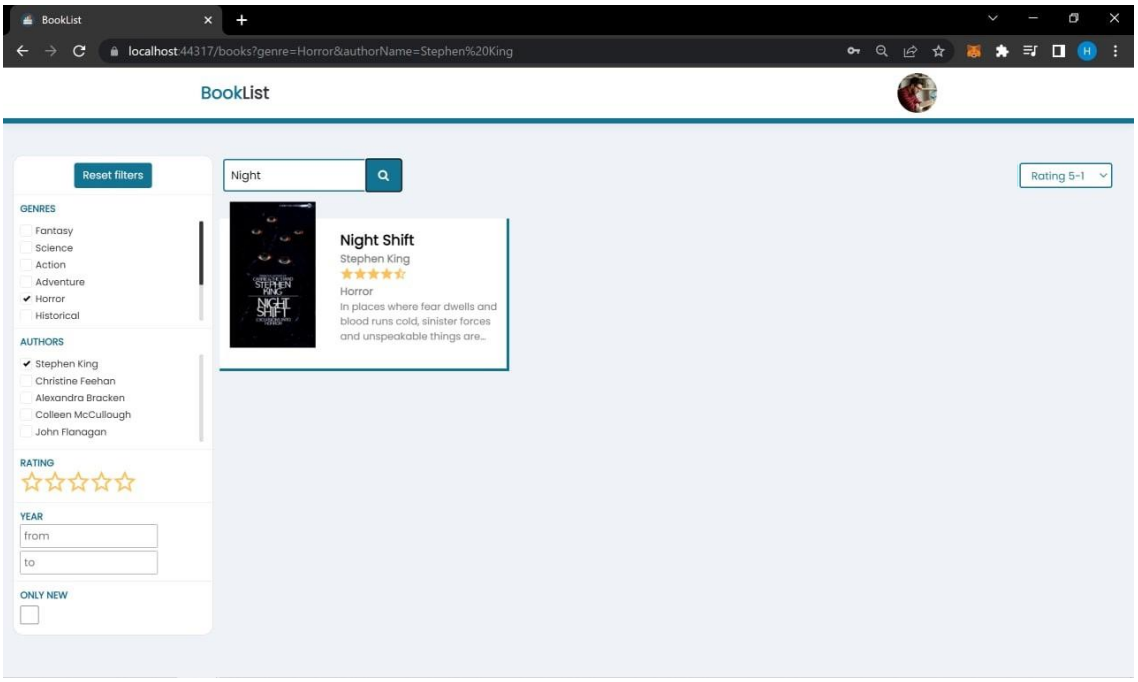


Рис. 5.4.3 - Сторінка перегляду книг з задіясними фільтрами, сортуванням та пошуком

Якщо книг за запитом не було знайдено користувачу відображається відповідне повідомлення Рис. 5.4.4.

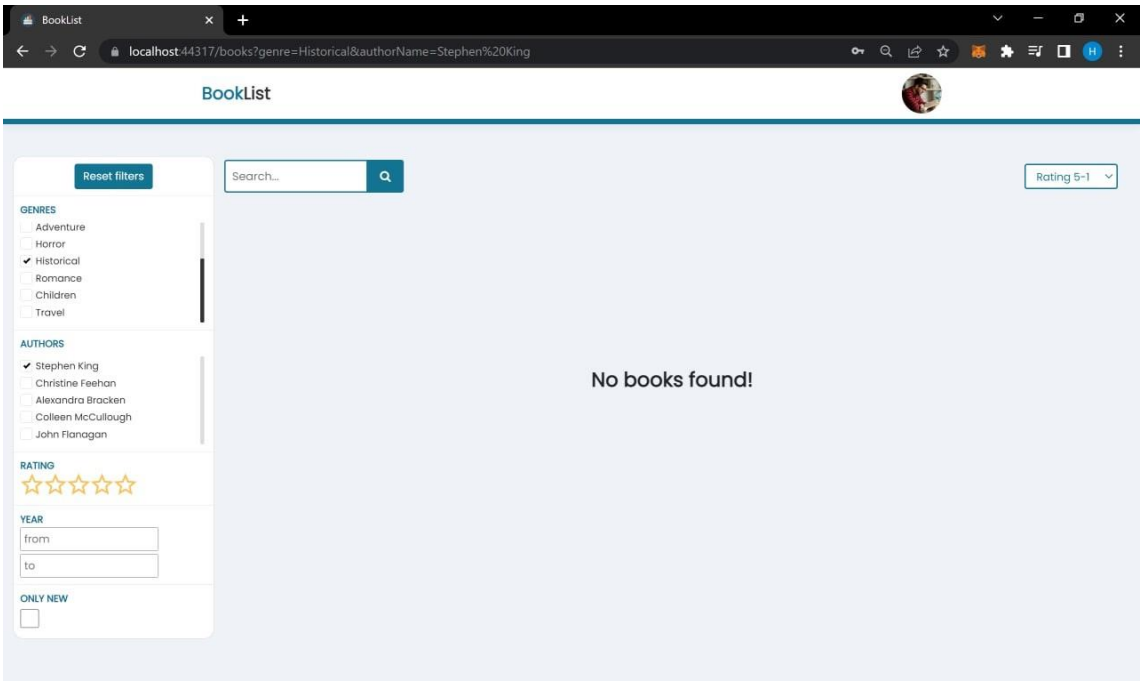


Рис. 5.4.4 - Повідомлення користувачу про те що результатів не було знайдено

Переглядаючи список книг, користувача може зацікавити якась книга, обравши та нажавши на книгу, користувачу відобразиться детальна інформація про цю книгу тільки якщо він авторизований Рис. 5.4.5.

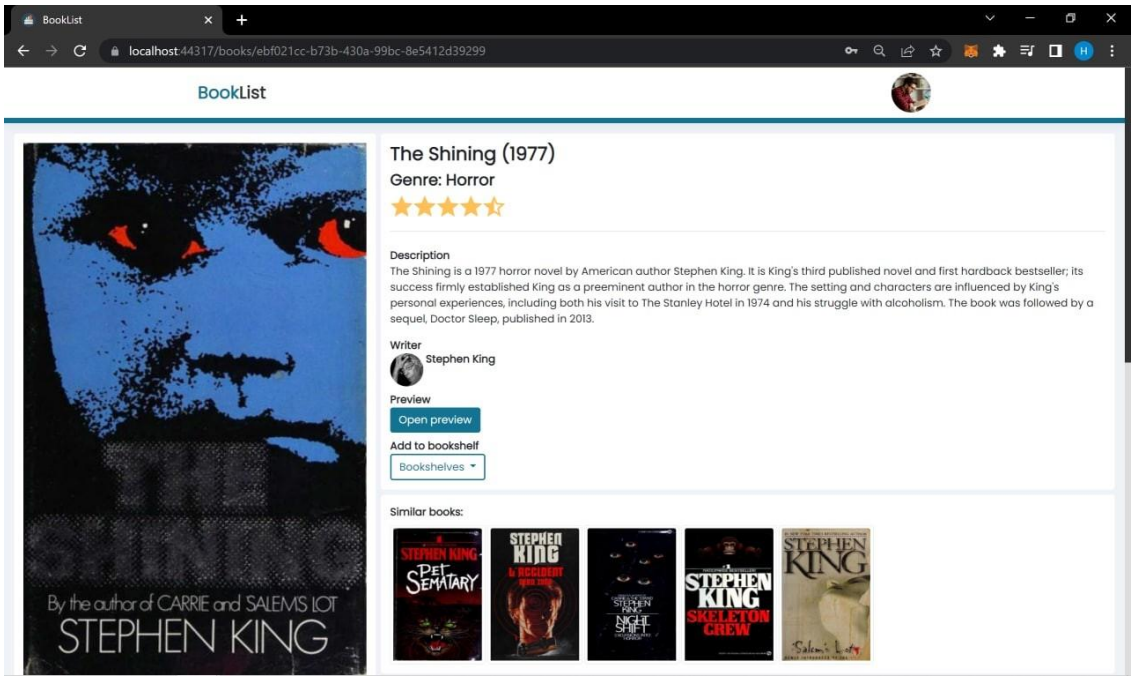


Рис. 5.4.5 - Сторінка з відображенням детальної інформації про книгу

У випадку якщо користувач не авторизований йому відобразитьсясторінка логування Рис. 5.4.6.

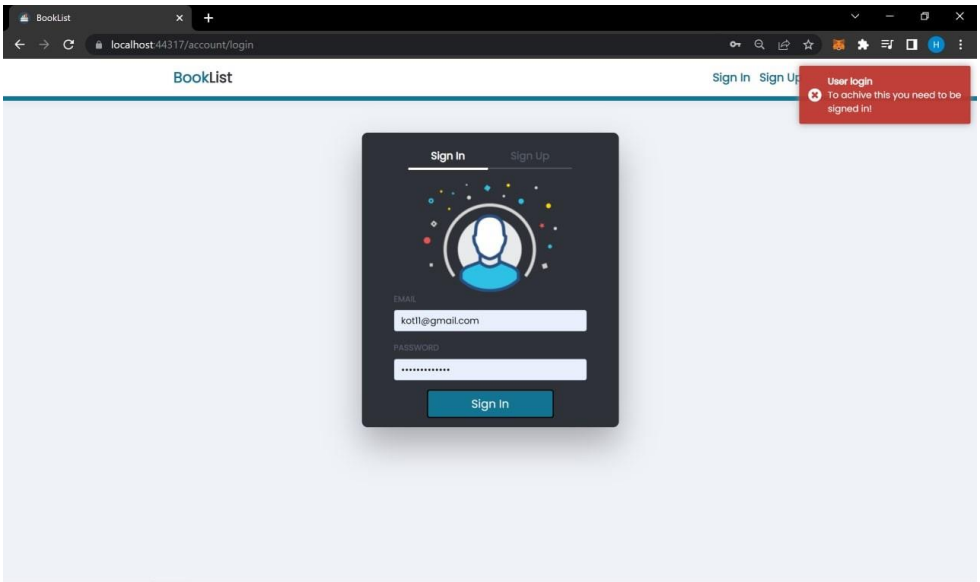


Рис. 5.4.6 - Сторінка авторизації користувача

У випадку якщо користувач ще не створив акаунт, йому потрібно нажати на Sign Up де при введені не вірних даних буде використана валідація з відображенням про помилку Рис. 5.4.7.

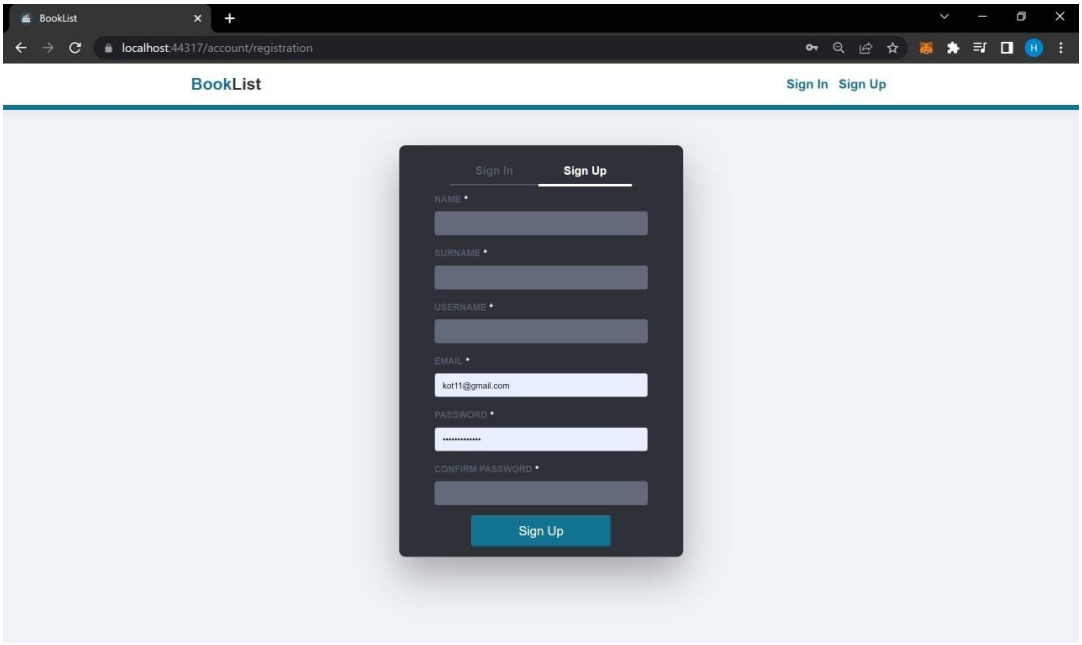


Рис. 5.4.7 - Сторінка реєстрації користувача

На сторінці з детальною інформацією також відображаються коментарі користувачів, що були залишені для цієї книги Рис. 5.4.8.

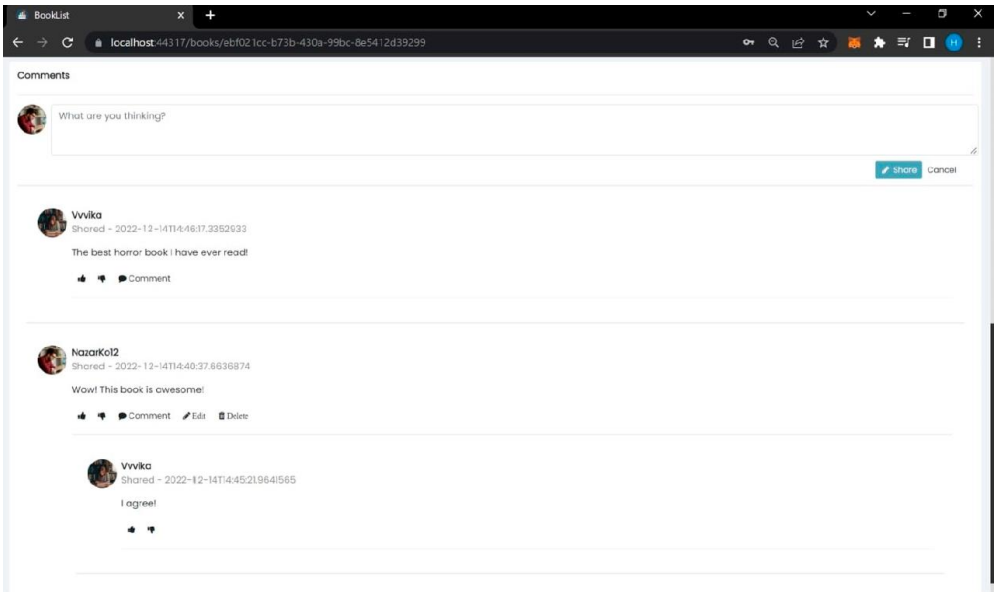


Рис. 5.4.8 - Відображення коментарів до книги

Перебуваючи на сторінці з детальною інформацією про книгу користувач може переглянути детальну інформацію про автора книги Рис. 5.4.9.

Рис. 5.4.9 - Сторінка відображення інформації про автора

Крім цього користувач може переглядати власний профіль та редагувати його Рис. 5.4.10. Також користувач може змінювати свій аватар у модальному вікні Рис. 5.4.11.

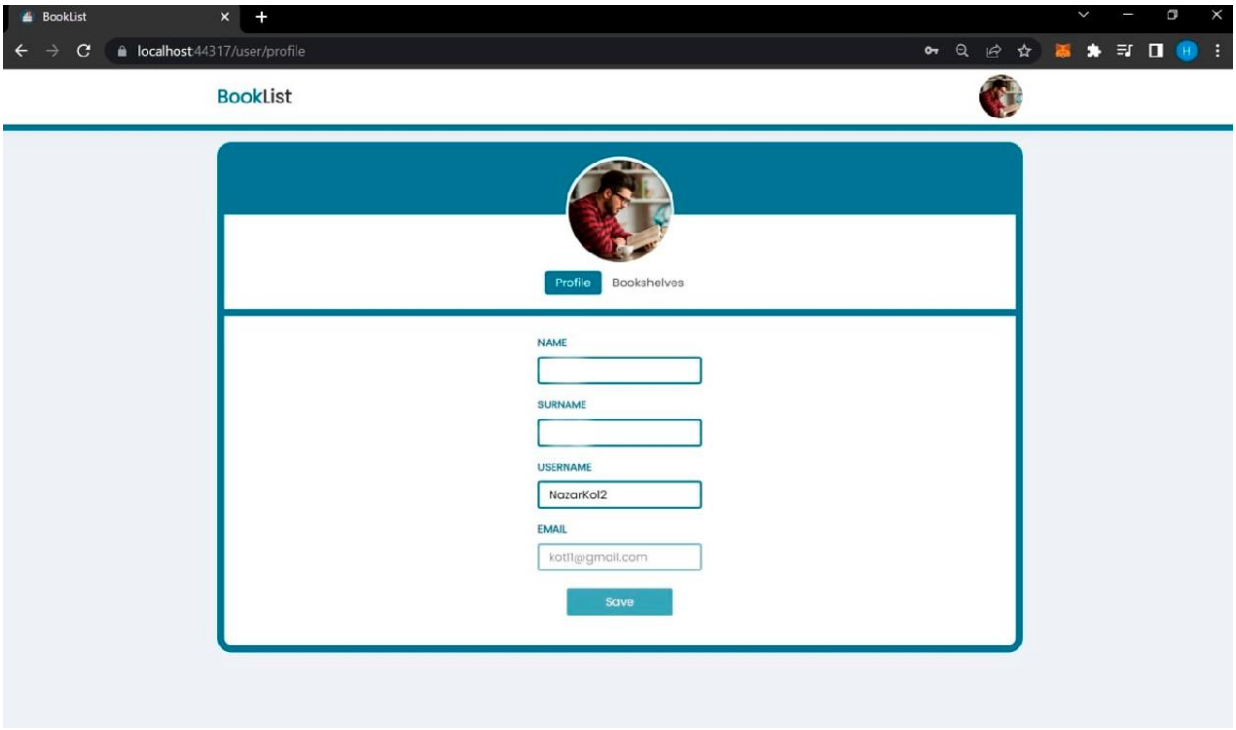


Рис. 5.4.10 - Сторінка відображення профілю користувача

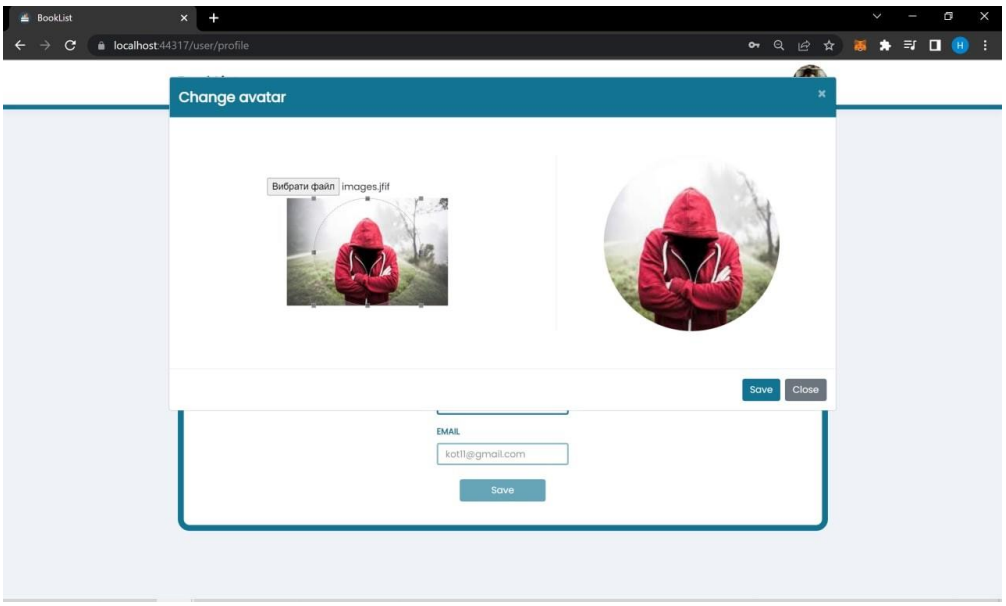


Рис. 5.4.11 - Модальне вікно для зміни аватару користувача

Також користувач може переглядати та створювати книжкові полиці. Щоб переглянути книжкові полиці користувачу на сторінці профілю потрібно нажати на кнопку Bookshelves. Тоді користувачу відобразиться список усіх його книжкових полиць у випадку якщо книжкових полиць ще не має користувачу відобразиться відповідне повідомлення Рис. 5.4.12.

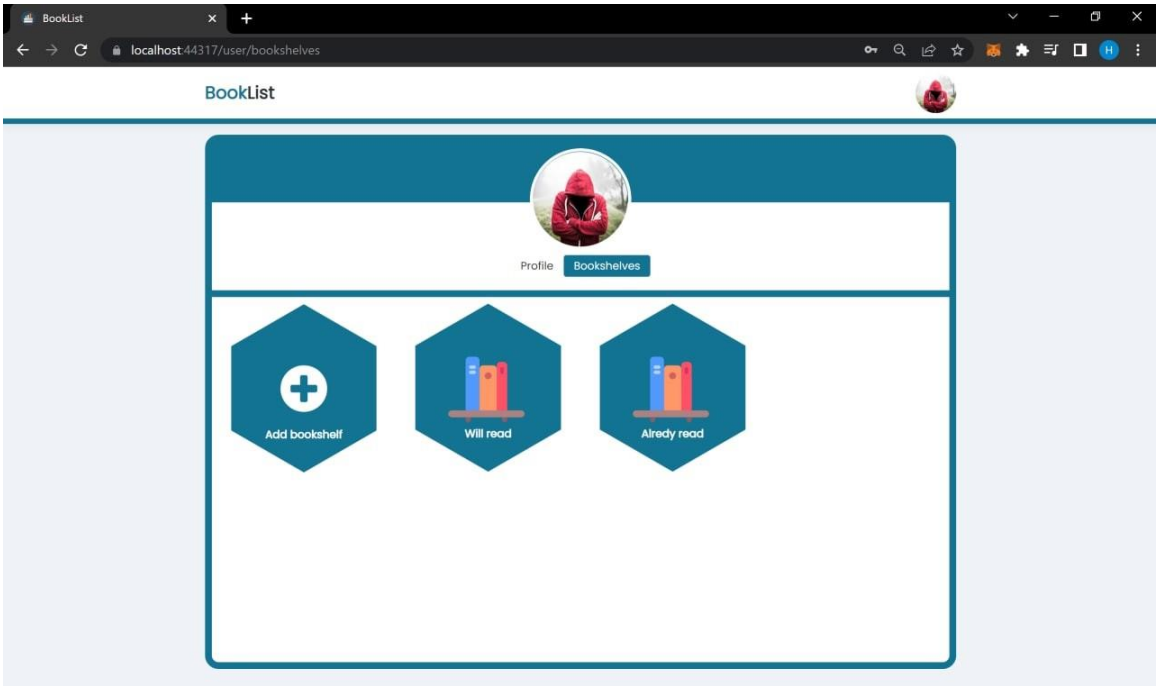


Рис. 5.4.12 - Сторінка відображення списку книжкових полиць

Переглядаючи список книжкових полиць користувач може добавити нову у модальному вікні добавлення книжкових полиць, у випадку неправильних даних присутня валідація Рис. 5.4.13.

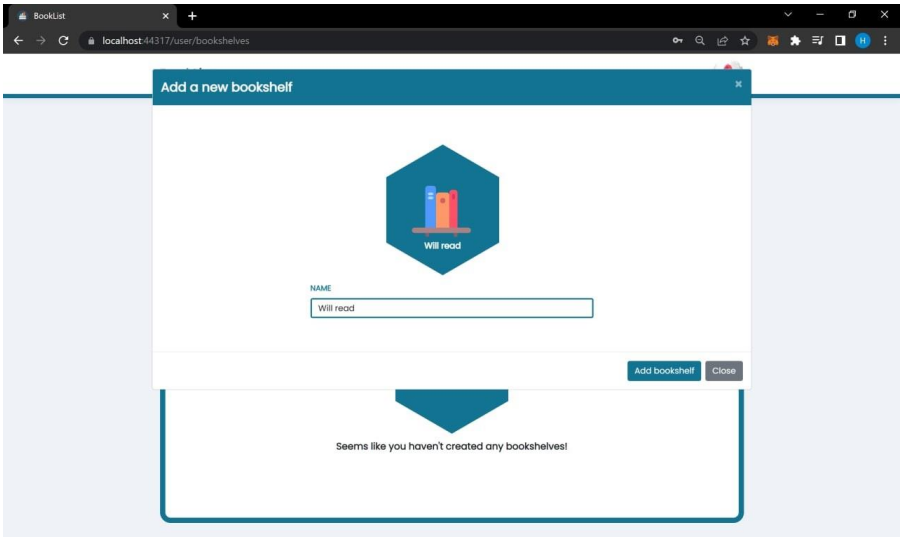


Рис. 5.4.13 - Модальне вікно добавлення книжкової таблиці

Маючи книжкові полиці користувач може нажати на будь-яку з них, тоді йому відобразяться усі книги даної книжкової полиці. Якщо книг ще немає, то користувачу відобразиться відповідне повідомлення Рис. 5.4.14.

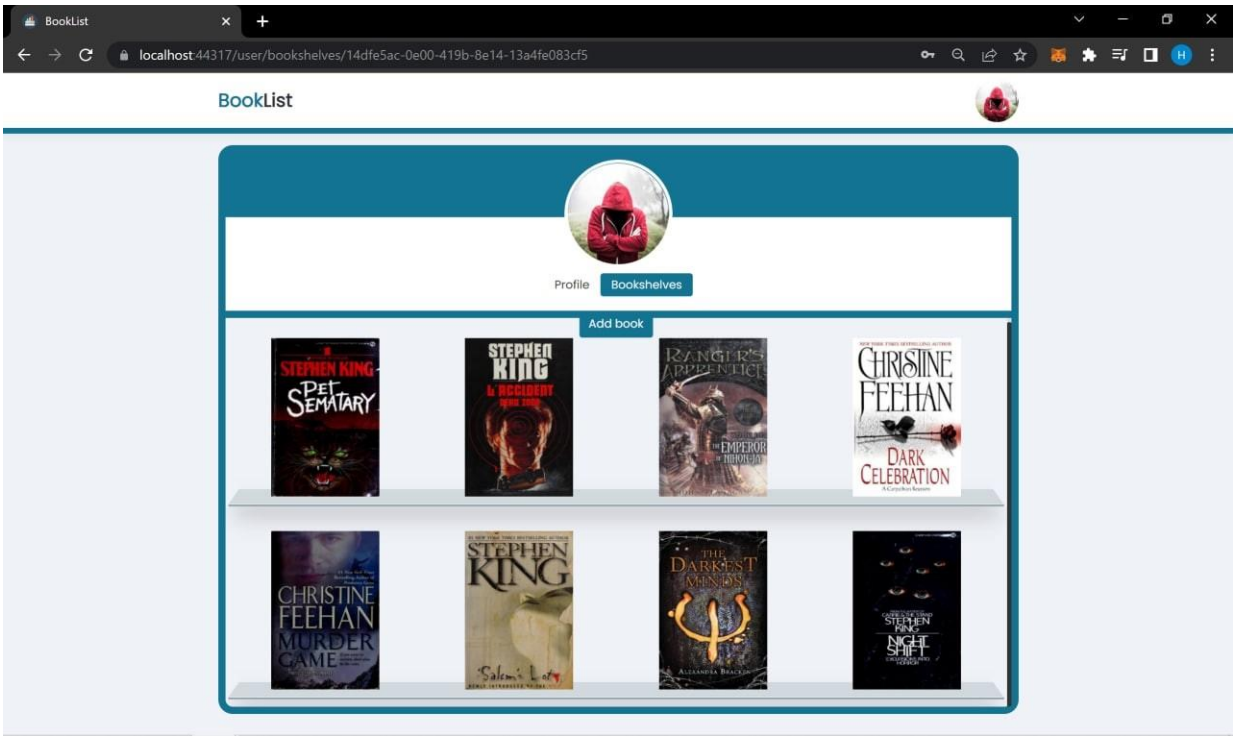


Рис. 5.4.14 - Відображення книг в книжкові полиці

Також користувач може добавляти нові книги в книжкову таблицю нажавши на кнопку Add book. І тоді у модальному вікні користувач може здійснити пошук книги та додати її до обраної полиці Рис. 5.4.15.

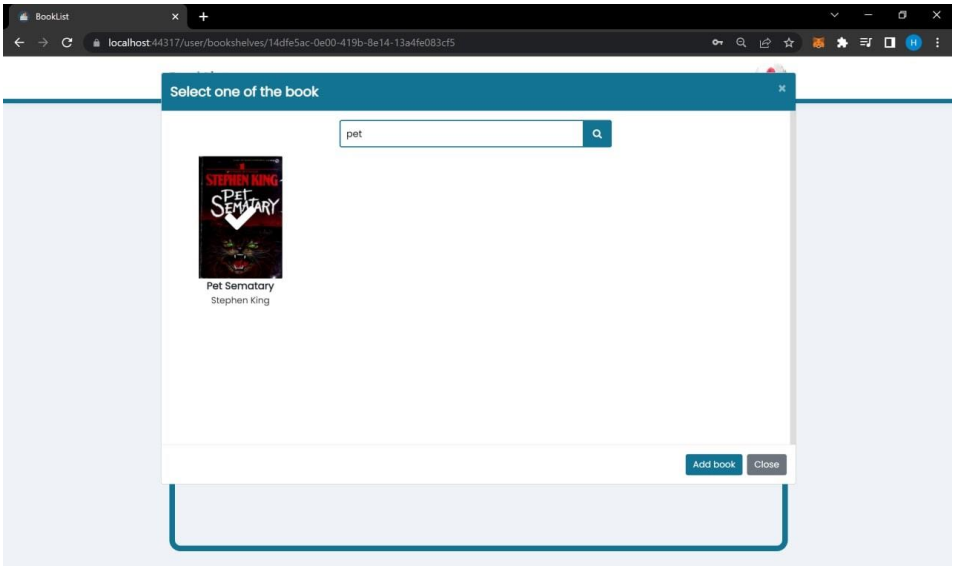


Рис. 5.4.15 - Модальне вікно добавлення книги до книжкової полиці

В системі також є адміністратори і для них інтерфейс виглядає трохи по іншому. А саме на сторінці відображення книг адміністратору доступна кнопка добавлення книги. Нажавши яку адміністратор може добавити нову книгу до бібліотеки Рис. 5.4.16.

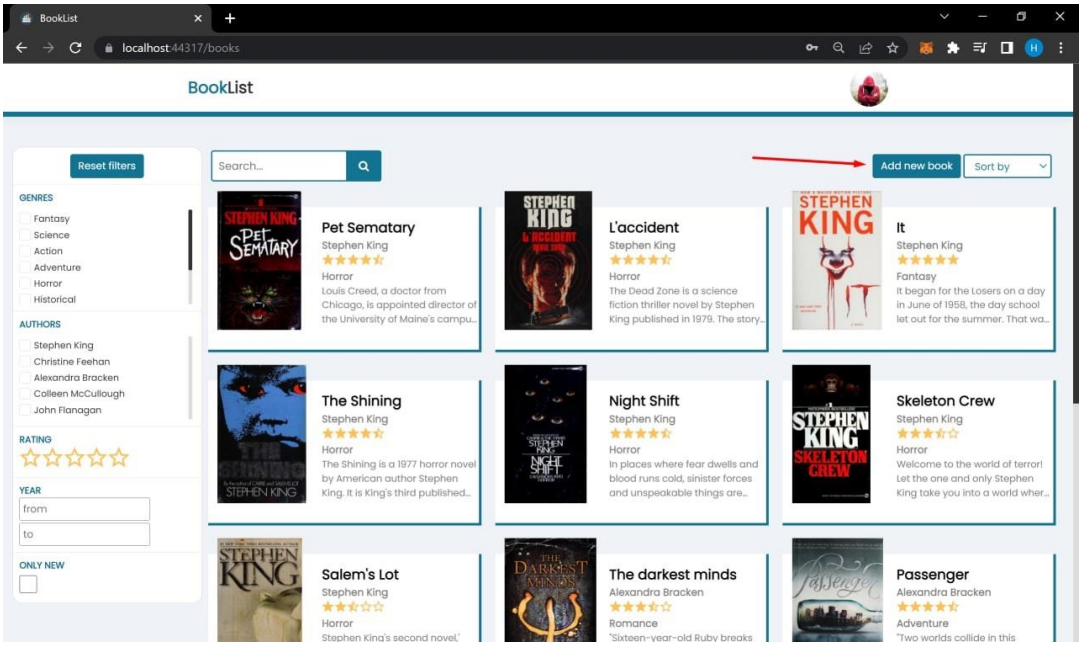


Рис. 5.4.16 - Інтерфейс адміністратора при перегляді книг

На сторінці детальної інформації про книгу адміністратори доступні кнопки редагування інформації про книгу та видалення книги Рис. 5.4.17.

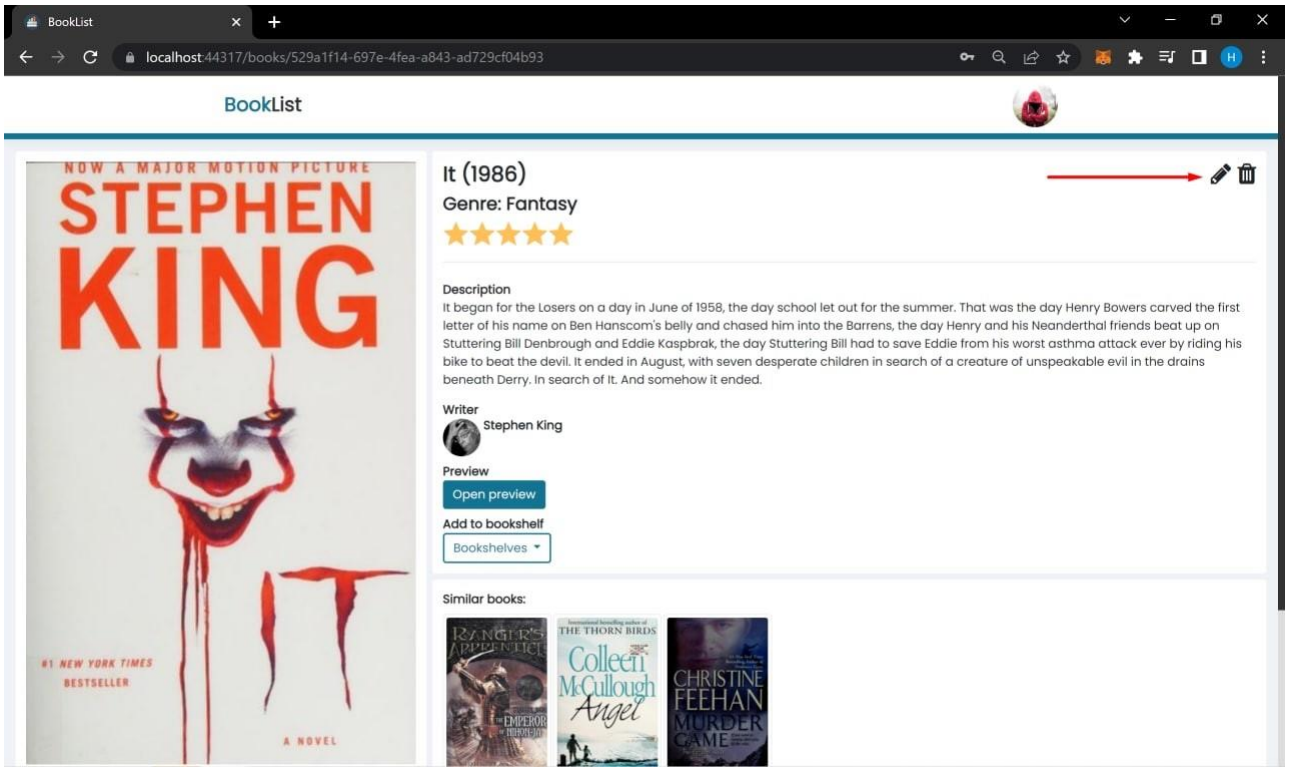


Рис. 5.4.17 - Інтерфейс адміністратора при перегляді детальної інформації про книгу

5.5. Опис альтернативних підходів, які розглядалися під час розробки

Найбільша альтернатива полягала у виборі сервісу для здійсненняоперації фільтрації, сортування та пошуку.

З одної сторони можна було реалізовувати ці функції власноруч, проте це зайняло б багато часу, на розробку алгоритмів, опрацювання запитів та конвертації даних. Тому кінцевим рішенням було пошук сервісів, що надають такі можливості. Розглядались 3 три сервіси Algolia, Azure Search та A Elasticsearch.

Algolia — це API-платформа для динамічного пошуку, фільтрації даних, яка допомагає компаніям максимізувати швидкість пошуку. Algolia забезпечує швидкий пошук, проте Algolia — одне з найдорожчих рішень, тому зазвичай не найкращий вибір для малого бізнесу та і масштабування може бути складним.

Elasticsearch — це безкоштовний сервер пошуку програмного забезпечення на основі Lucene. Elasticsearch можна використовувати для індексування та пошуку будь-яких документів. Він пропонує масштабований пошук і виконує пошук майже в реальному часі. Проте Elasticsearch через ціноутворення це не ідеальний вибір для малого бізнесу. Підтримка не завжди є ініціативною, і деякі розробники відзначають, що підтримка спільноти є ефективнішою, ніж та, яку надає компанія.

Azure Cognitive Search — це рішення від Microsoft і, за власним визначенням, «єдина служба хмарного пошуку з вбудованими можливостями AI». Мета сервісу — зробити пошук у масштабі легшим і швидшим за допомогою AI.

Він заснований на тому ж стеку природних мов, що й Bing і Office, і включає ключові функції автозаповнення, геопросторового пошуку, фільтрації та огранювання, а також OCR, виділення ключових фраз і розпізнавання іменованих об’єктів.

В результаті було обрано Azure Cognitive Search оскільки він підходить для малого бізнесу, та працює на штучному інтелекті, щоб пришвидшити пошук документів. Крім цього він розроблений компанією Microsoft тому буде краще інтегрувати з C#.

5.6 Опис проблем і нестандартних ситуацій, які виникали під час розробки та заходів для їх вирішення

Під час розробки системи поставали дві основні проблеми: де зберігати всі динамічні зображення системи (обкладинки книг, аватари користувачів), та реалізація функції вкладених коментарів.

Щодо зберігання аватарів то їх можна було б зберігати в базі даних. Дане рішення б працювало, але зображення зазвичай займають багато пам'яті тому розмір бази даних б значно виріс через зображення. Крім цього читання та запис даних відбувалось б повільніше, що теж погано. Також зображення можна було б зберігати в файловій системі, при цьому читання і запис б був швидше

але це вимагало б розробки додаткової логіки для зберігання і читання з файлової системи.

Тому було прийнято рішення зберігати фотографії віддалено на сервері. Таким сервером був обраний Cloudinary який надає 28 гігабайтів для зберігання фотографій з можливістю їх модифікації, в базі даних ж зберігаються лише посилання на фото, самі посилання займають мало місця тому дане рішення є аргументоване.

Наступною проблемою була функція вкладених коментарів, тобто можливості залишити відповідь для конкретного коментаря. Проблема полягала в тому що по своїй суті та по параметрах сутність Коментар та сутність Відповідь є однаковими. А зберігання таблиць з однаковим набором атрибутів порушило б нормалізацію бази даних. Тому було прийнято рішення про добавлення зовнішнього ключа ParendId до таблиці Коментар. В даному випадку виходить що таблиця коментар має посилання на саму себе Рис. 5.6.1.

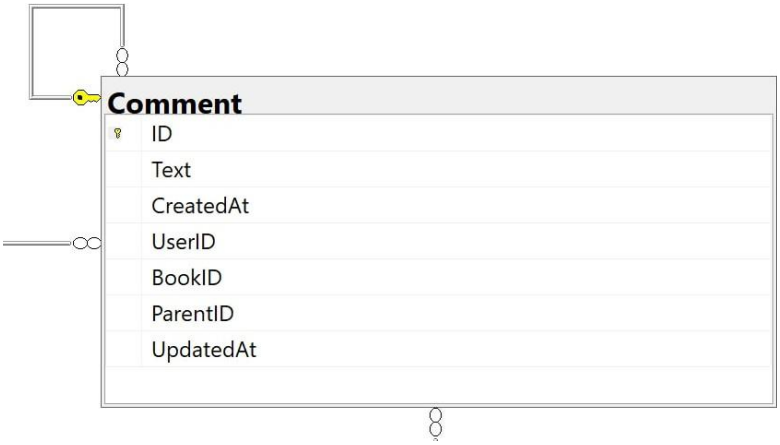


Рис. 5.6.1 - Посилання таблиці коментар на саму себе

Це потрібно для того, щоб розуміти які відповіді належать до якого коментаря. Всі коментарі мають ParentId рівний значенню null оскільки коментар не може належати якомусь коментарю, а ось для відгуків ParentId рівний Id коментаря для якого вони створювались. Саме введення додаткового атрибута дозволяє зберегти базу даних у нормальній формі та запобігти створенню схожої таблиці.

РОЗДІЛ 6. ЕКСПЕРИМЕНТАЛЬНА ЧАСТИНА

6.1 Інструкції користувачам

Якщо дану систему хоче запустити адміністратор або програміст хоче перенести її на новий комп’ютер, то їм потрібно мати усі необхідні файли системи. Система розроблялась з використанням популярного IDE Visual Studio 2019.

Отже, для коректної роботи та запуску системи потрібно зробити наступне:

- встановити Visual Studio 2019 або вищої версії, для можливості перегляду коду, його редагування та запуску системи;

- встановити SQL Server Management Studio для перегляду бази даних та роботи з нею;

- потрібно відкрити систему у Visual Studio 2019 або вищої версії;

- далі потрібно знайти файл appsettings.json (у цьому файлі зберігаються багато конфігурацій, і одна з них це стрічка підключення до бази даних). Потрібно замінити частину стрічки підключення бази даних, адже на кожному локальному комп’ютері вона своя;

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=DESKTOP-L3C6L0M\\SQLEXPRESS;Database=BookListDb;Trusted_Connection=True;MultipleActiveResultSets=true;"
  },
```

Рис. 6.1.1 - Стрічка підключення до бази даних

- стрічку підключення до своєї бази даних можна знайти скориставшись SQL Server Management Studio. Потрібно відкрити цю програму, далі у списку доступних підключень нажати ПКМ, та вибрати properties, тоді в першому полі яке називається «Name» буде видно як називається назва нашого підключення, яке потрібно буде вставити;

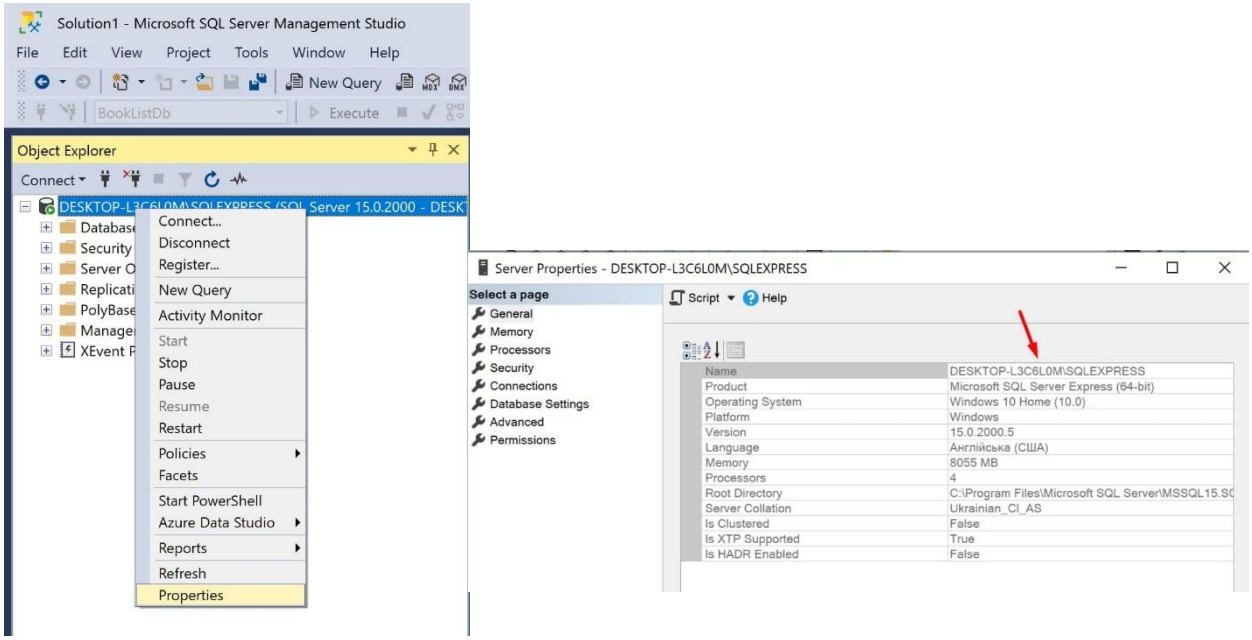


Рис. 6.1.2 - Строка підключення у SQL Server Management Studio

- після цього потрібно запустити команду у Visual Studio 2019, а саме у Package Manager Console. Команда виглядає так для систем Windows «Update-Database», і так для систем на MacOS «dotnet ef database update». Дана команда потрібна для того, щоб згенерувати базу даних з всіма конфігураційними файлами та зберегти її на SQL Server. Як тільки після виконання команди у SQL Server Management Studio з’явиться база даних BookListDB, можна запускати програму на виконання у Visual Studio 2019.

6.2 Вимоги до апаратного забезпечення

Дана система являє собою веб-сервіс з клієнтом, який запускається у браузері. Клієнт системи розроблявся в основному для браузера Google Chrome.

Вимоги до середовища:

- microsoft Windows 10, OS X El Capitan (10.11) або вище, Ubuntu 18.04 або вище;

- встановлення браузер Google Chrome.

Зовнішні вимоги:

- мова інтерфейсу – англійська, українська;

- попередньо встановлене середовище Visual Studio 2019;

- попередньо встановлене середовище SQL Server Management Studio.

Вимоги до апаратної частини:

- процесор з тактовою частотою не нижче 1.8 ГГц. Рекомендується використовувати як мінімум двоядерний процесор;

- мінімум 4 ГБ ОЗП, проте для кращої роботи системи краще 8 ГБ DDD4;

- місце на жорсткому диску мінімум 128 ГБ, при цьому краще використовувати SSD, адже він працює на порядок швидше HDD;

- відеоадаптер з мінімальним розширенням 1280 на 720 пікселів, мінімальна відеопам’ять 1 ГБ;

- монітор;

- клавіатура;

- мишка.

Вимоги до мережі:

- безперебійне з’єднання до Інтернету;

- рекомендована швидкість Інтернету 100 Мбіт/с.

6.3 Тестування

Тестування - це важливий процес в циклі розробки системи. Тестування починається після закінчення активної частини розробки системи, коли основні функції системи вже готові. Існує дуже багато видів тестування, але в основі всіх лежать два типи функціонування: функціональне тестування, та нефункціональне тестування. При функціональному тестуванні перевіряється як працює та чи інша функція системи, враховуючи її логічні та графічні чинники (інтерфейс).

Було проведено тестування основних функцій системи. Кожен тест включає в себе опис тесту, реальну поведінку функції, та очікувану поведінку функції. Якщо очікуваний результат і реальний є однаковими це значить, що

функція працює вірно. Тести були створені для того, щоб переконатись, що всі функції системи працюють правильно, та ніде немає ніяких помилок.

Опис функцій та список тестів для них:

1) авторизація та реєстрація користувача:

Тест №1. Користувач має створений акаунт та ввів вірний email та пароль на сторінці авторизації, та нажав на кнопку «Sign In».

Очікуваний результат: користувач успішно авторизується в систему, при цьому йому автоматично відкривається сторінка для перегляду списку книг.

Тест №2. Користувач ввів неправильний пароль або email, на сторінці авторизації, та нажав на кнопку «Sign In».

Очікуваний результат: користувачу відображається повідомлення про помилку, авторизація не відбувається.

2) перегляд списку книг, фільтрація, пошук, сортування:

Тест №1. Користувач може одночасно використовувати фільтрацію, пошук та сортування, що в кінцевому результаті виведе список книг.

Очікуваний результат: користувачу відобразиться список книг з всіма залученими правилами сортування, фільтрації та пошуку.

Тест №2. Користувач обрав параметри фільтрації чи пошуку і за даними параметрами нічого не знайдено.

Очікуваний результат: користувачу відобразиться повідомлення про те що результат не знайдений.

3) залишення коментарів для книги, та відгуків під коментарями:

Тест №1. Користувач хоче добавити новий коментар для прочитаної книги.

Очікуваний результат: новий коментар успішно добавляється до книги від імені користувача який його залишав, та зберігається під цією книгою.

Тест №2. Користувач хоче змінити свій залишений коментар.

Очікуваний результат: коментар змінюється, та зберігається, крім цього зверху над коментарем з’являється помітка, що коментар був змінений.

4) створення та зберігання книжкових полиць:

Тест №1. Користувач хоче добавити нову книжкову полицю з вказаним ним ім’ям, вводячи всі дані правильно.

Очікуваний результат: книжкова полиця успішно зберігається з вказаним ім’ям.

Тест №2. Користувач добавляє книжкову полицю вказуючи ім’я яке вже існує.

Очікуваний результат: якщо книжкова полиця з таким іменем вже існує, то добавлення нової книжкової полиці не відбувається, а користувачу відображається відповідне повідомлення.

5) добавлення, оновлення, та видалення книги.

Тест №1. Користувач з роллю адмін хоче добавити нову книгу вводячи всі дані правильно.

Очікуваний результат: нова книга добавляється в бібліотеку, і одразу відображається користувачам.

Тест №2. Користувач з роллю адмін хоче добавити нову книгу при цьому, забуває ввести одне з значень.

Очікуваний результат: добавлення книги не відбувається, адміністратору показуються відповідні помилки про прошуння валідації.

6.4 Опис проведених експериментів

Описавши важливі тести, можемо тепер провести відповідні експерименти, щоб переконатись, що система працює правильно та нічого не пропущено.

1) Авторизація та реєстрація користувача:

Експеримент №1. Користувач маючи вже створений акаунт, намагається здійснити авторизацію у систему та вводить свій email і пароль правильно. Як видно з Рис. 6.4.1 користувач був успішно авторизований у систему.

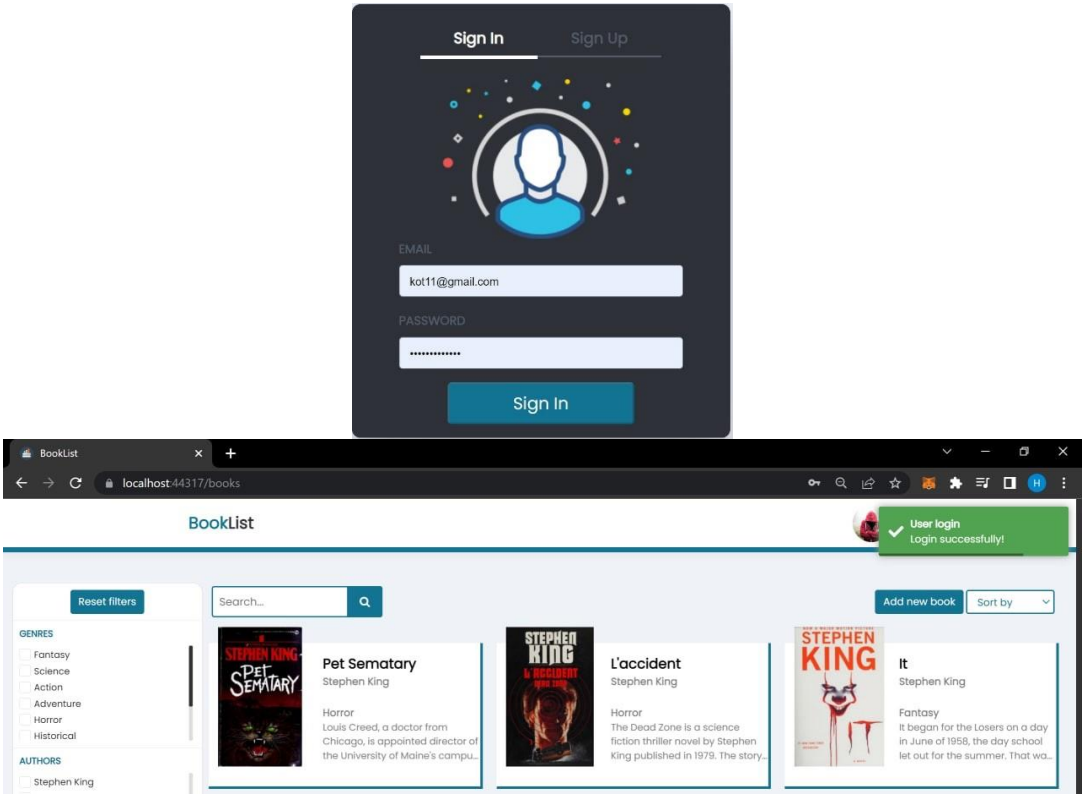


Рис. 6.4.1 - Результат Експерименту №1 для авторизації

Експеримент №2. Користувач маючи вже створений акаунт, намагається здійснити авторизацію у систему та вводить свій email або пароль не правильно. Як видно з Рис 6.4.2 користувачу відображається повідомлення про те що пароль або email був введений не вірно.

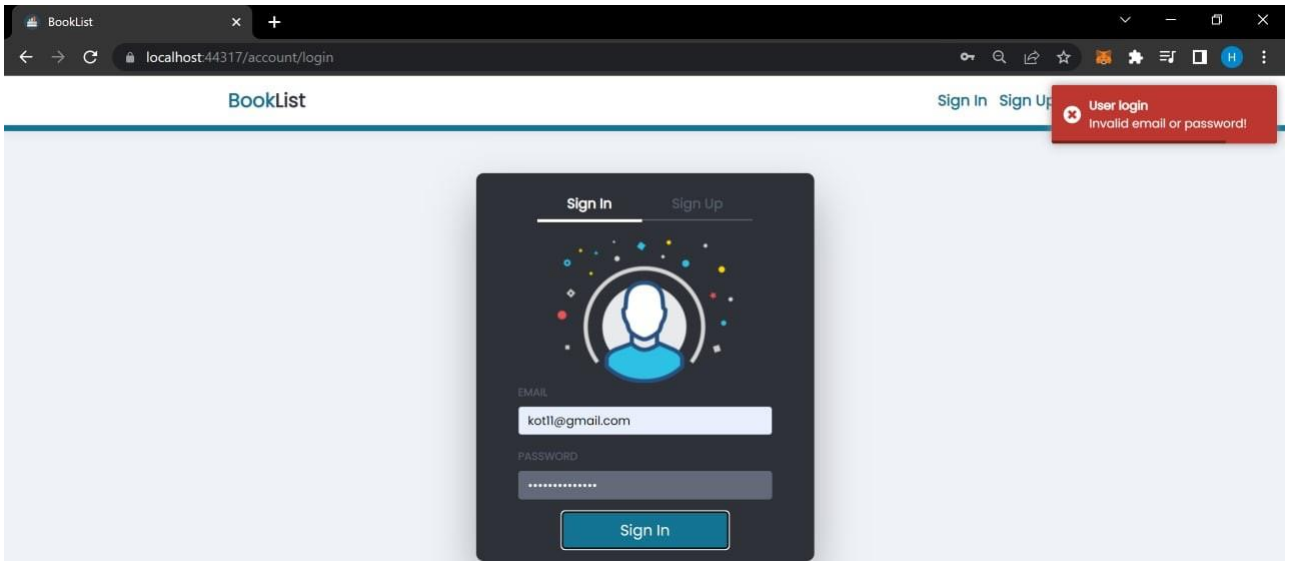


Рис. 6.4.2 - Результат Експерименту №2 для авторизації

2) Перегляд списку книг, фільтрація, пошук, сортування:

Експеримент №1. Користувач знаходячись на сторінці перегляду списку книг, задає фільтр по жанру «Adventure» для книг, шукає книги, що містять букву «a» у назві та сортує книги в алфавітному порядку по імені. Як зображенона Рис. 6.4.3 всі залучені фільтри були застосовані одночасно, і користувачу відображаються результати.

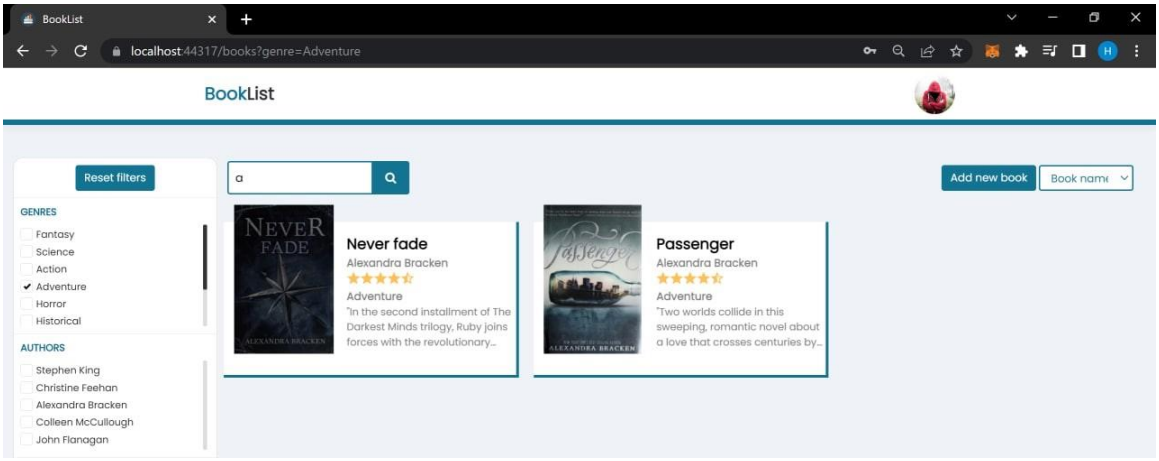


Рис. 6.4.3 - Результат Експерименту №1 для перегляду списку книг

Експеримент №2. Користувач знаходячись на сторінці перегляду списку книг, здійснює пошук книг ввівши «Dragon». Проте даних книг в бібліотеці немає. Як видно з Рис. 6.4.4 користувачу відображається повідомлення про те що таких книг в системі немає.

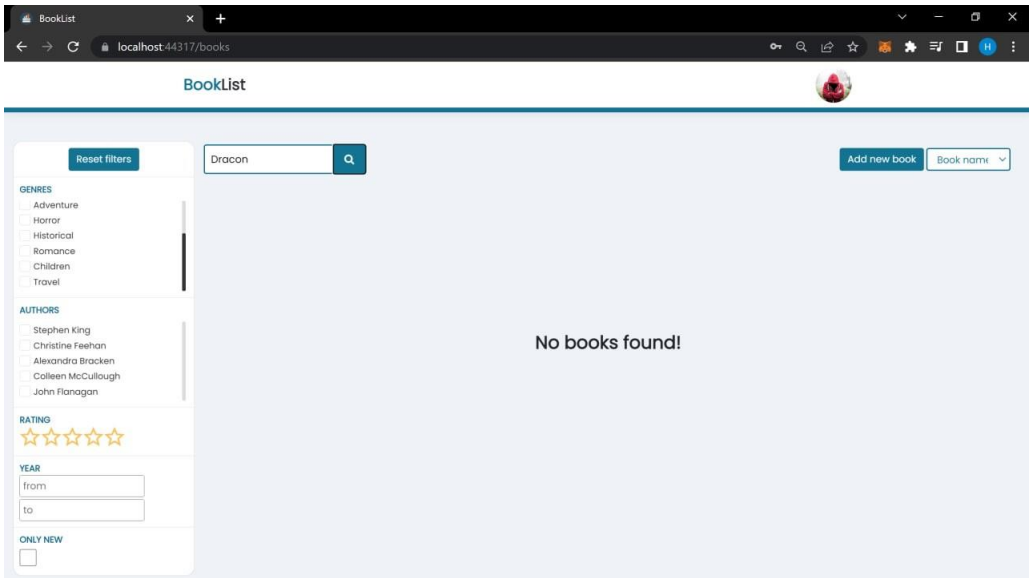


Рис. 6.4.4 - Результат Експерименту №2 для перегляду списку книг

3) Залишення коментарів для книги, та відгуків під коментарями:

Експеримент №1. Користувач будучи на сторінці де зображуєтьсядетальна інформація про книгу хоче залишити коментар для цієї книги. Як видно з Рис. 6.4.5 користувач успішно додав коментар для книги.

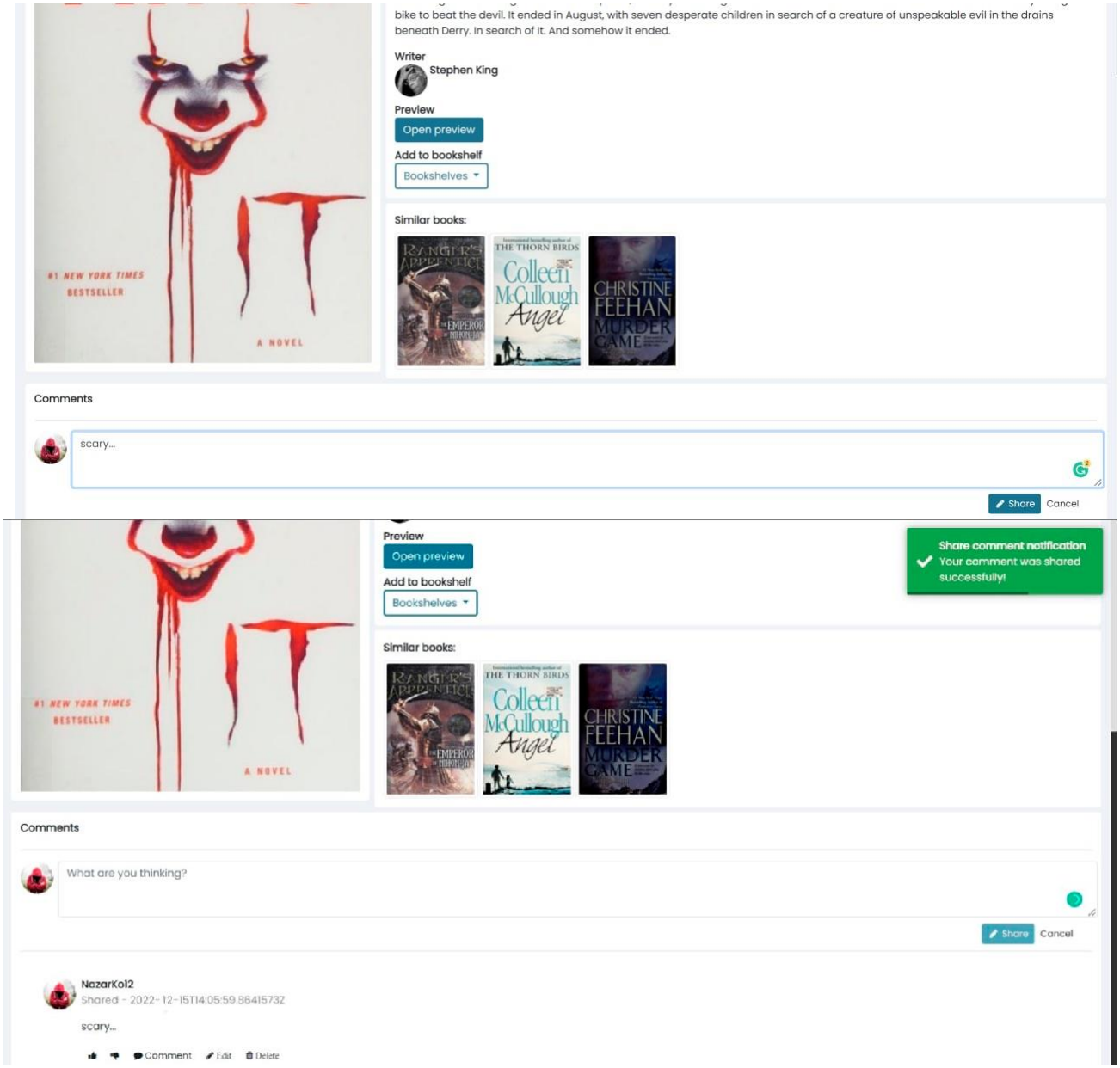


Рис. 6.4.5 - Результат Експерименту №1 для залишення коментарів

Експеримент №2. Користувач будучи на сторінці де зображуєтьсядетальна інформація про книгу хоче редагувати вже залишений коментар для

книги. Як видно з Рис. 6.4.6 користувач успішно змінив свій коментар, що вже залишав.

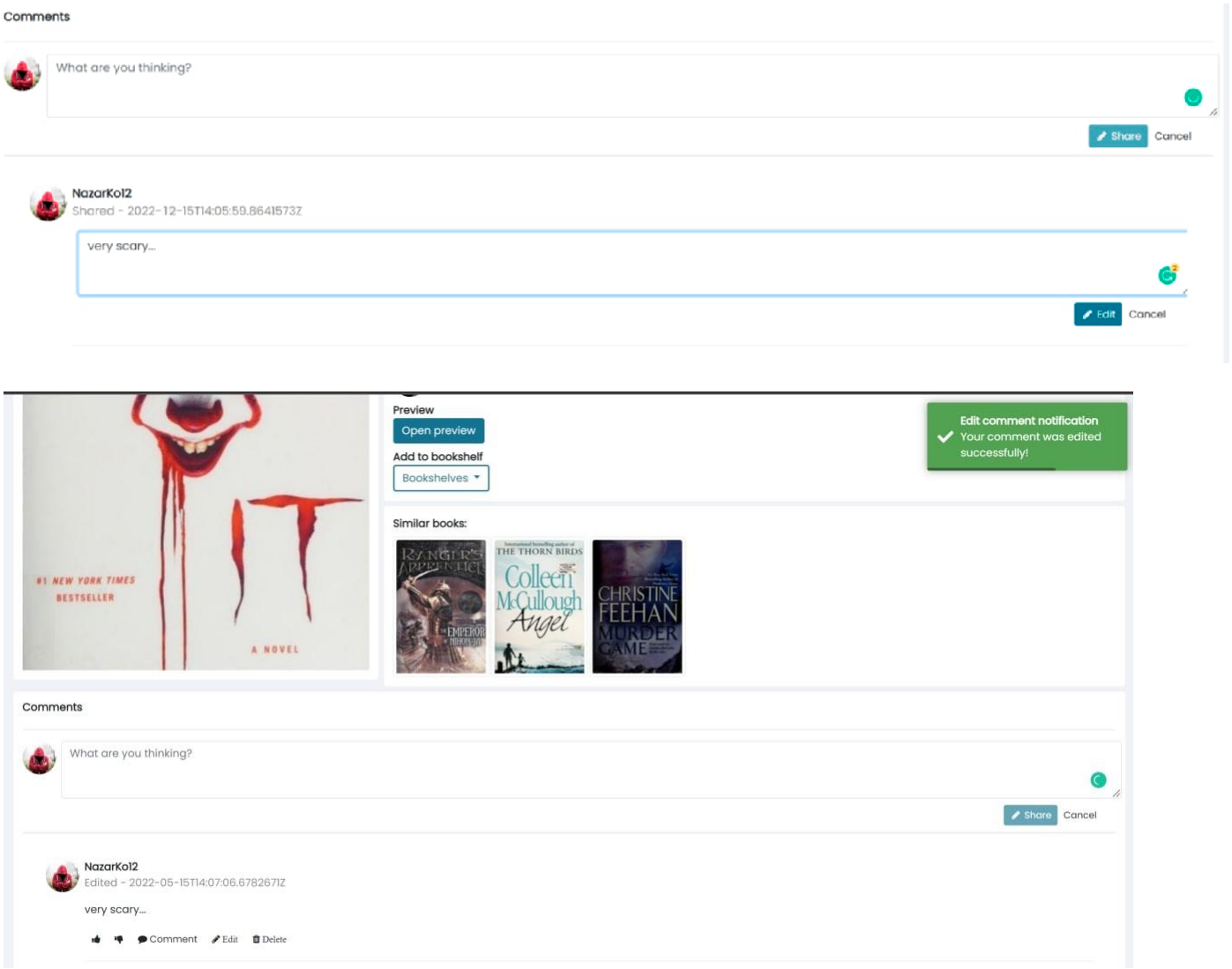


Рис. 6.4.6 - Результат Експерименту №2 для залишення коментарів

4) Створення та зберігання книжкових полиць:

Експеримент №1. Користувач будучи на сторінці перегляду книжкових полиць добавляє нову полицю вказуючи ім’я «Will read». У користувача немає книжкових полиць з таким іменем. Як видно на Рис. 6.4.7 книжкова полиця була успішно створена.

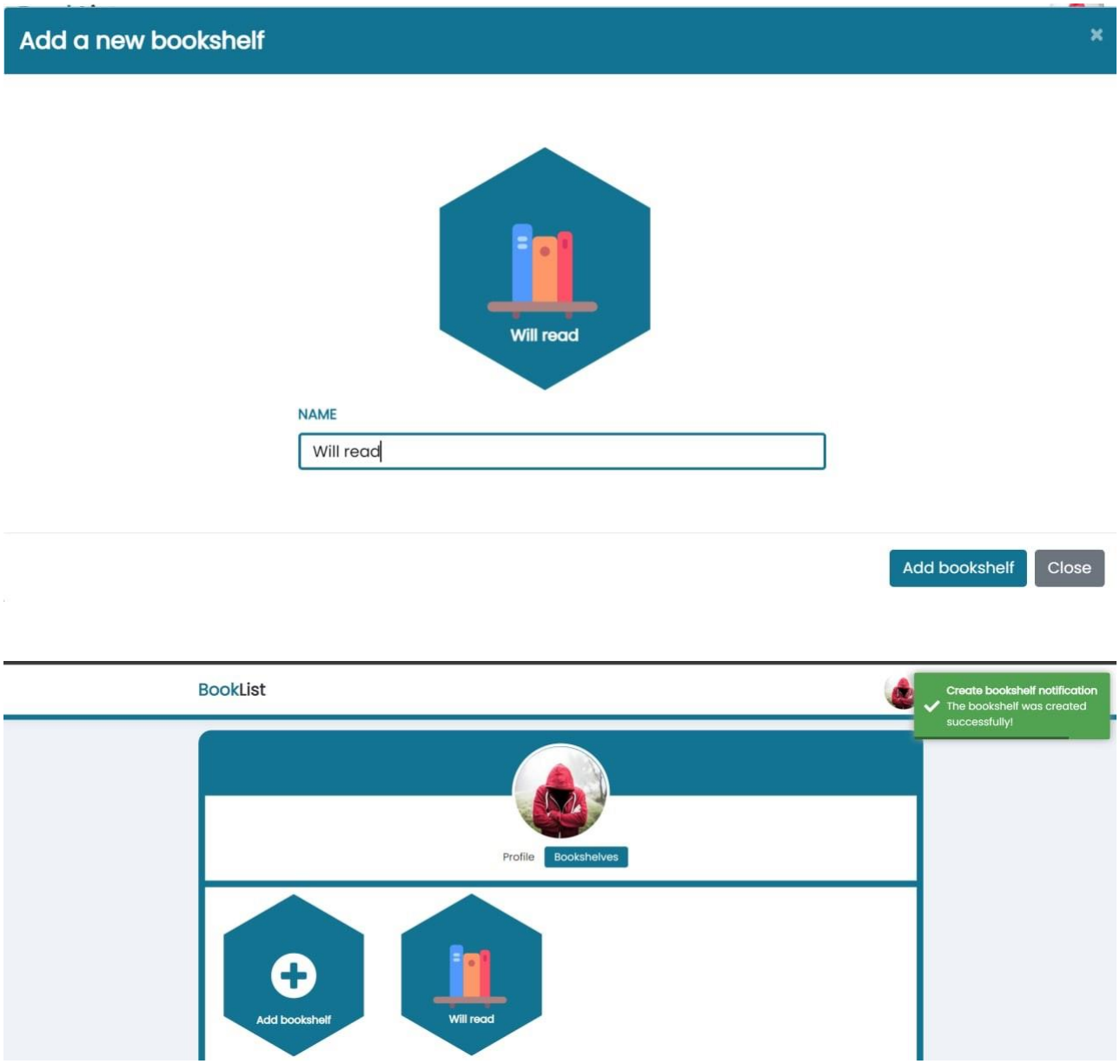


Рис. 6.4.7 - Результат Експерименту №1 для книжкових полиць

Експеримент №2. Користувач будучи на сторінці перегляду книжкових полиць добавляє нову полицю вказуючи ім’я «Will read». У користувача вже є книжкова полиця з таким іменем. Як видно на Рис. 6.4.8 книжкова полиця не була створена, і користувачу було відображено відповідне повідомлення.

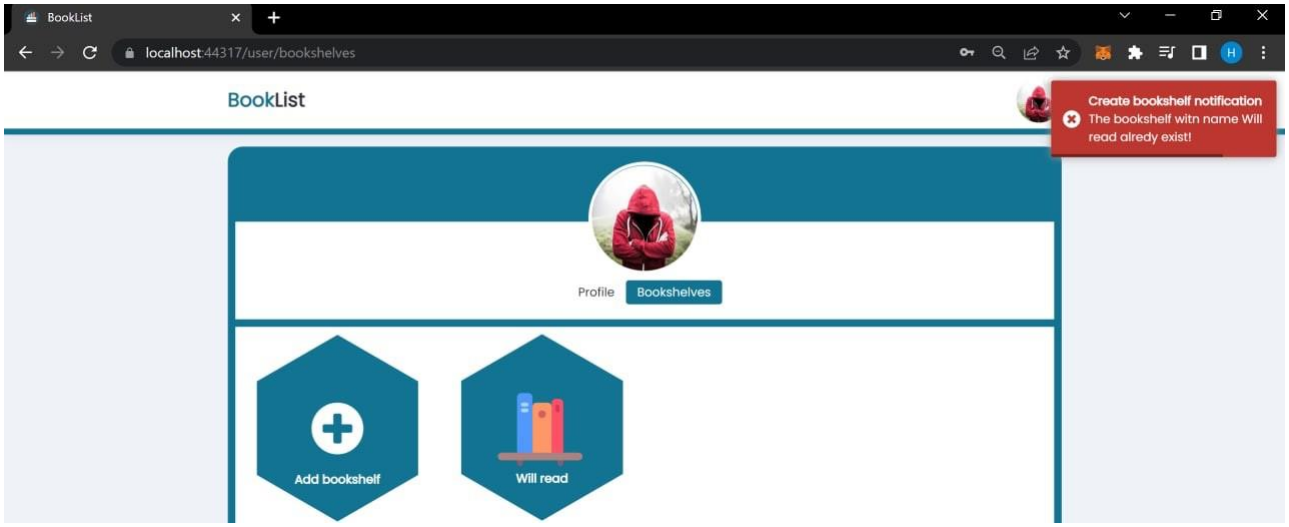
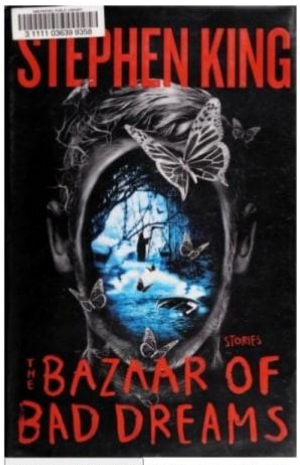


Рис. 6.4.8 - Результат Експерименту №2 для книжкових полиць

5) Добавлення, оновлення, та видалення книги:

Експеримент №1. Адміністратор бібліотеки добавляє нову книгу в бібліотеку при цьому вказуючи всі дані вірно. Як видно з Рис. 6.4.9 книга була успішно додана в систему.

Update a book



Вибрати файл Файл не вибрано

Book parameters:

TITLE: The bazaar of bad dreams

IS NEW: ☐

DESCRIPTION: A master storyteller at his best--the O. Henry Prize winner Stephen King delivers a generous collection of stories, several of them brand-new, featuring revelatory autobiographical comments on when, why, and how he came to write (or rewrite) each story. Since his first collection, Nightshift, published thirty-five years ago, Stephen King has dazzled readers with his genius as a writer of short fiction. In this new

AUTHOR: Stephen King

GENRE: Horror

YEAR: 2015

RATING: 4

Add book | Close

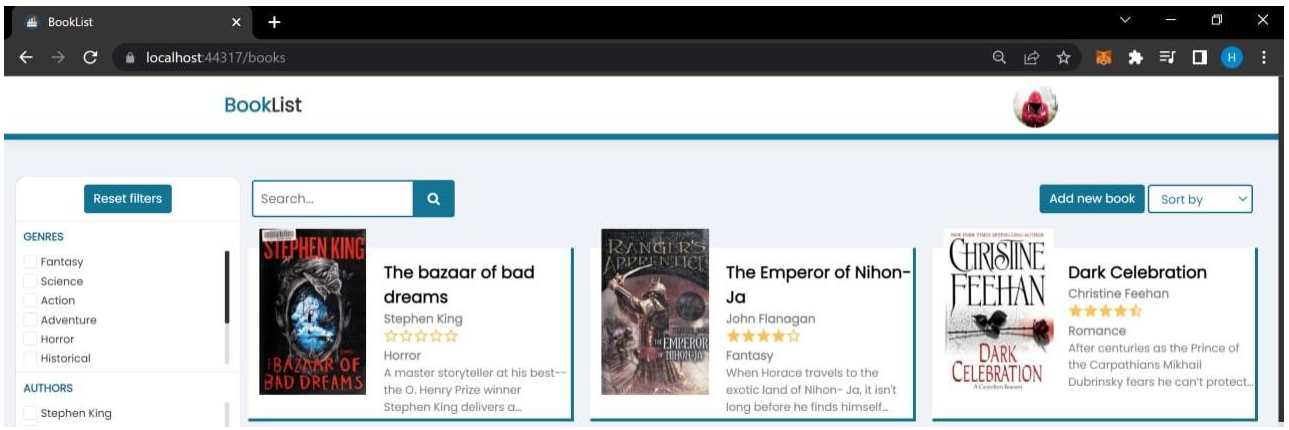


Рис. 6.4.9 - Результат Експерименту №1 для добавлення книги

Експеримент №2. Адміністратор бібліотеки добавляє нову книгу в бібліотеку при цьому забувши ввести значення року та рейтинг книги. Як видно з Рис. 6.4.10 книга не була додана в бібліотеку а адміністратору показані відповідні помилки валідації.

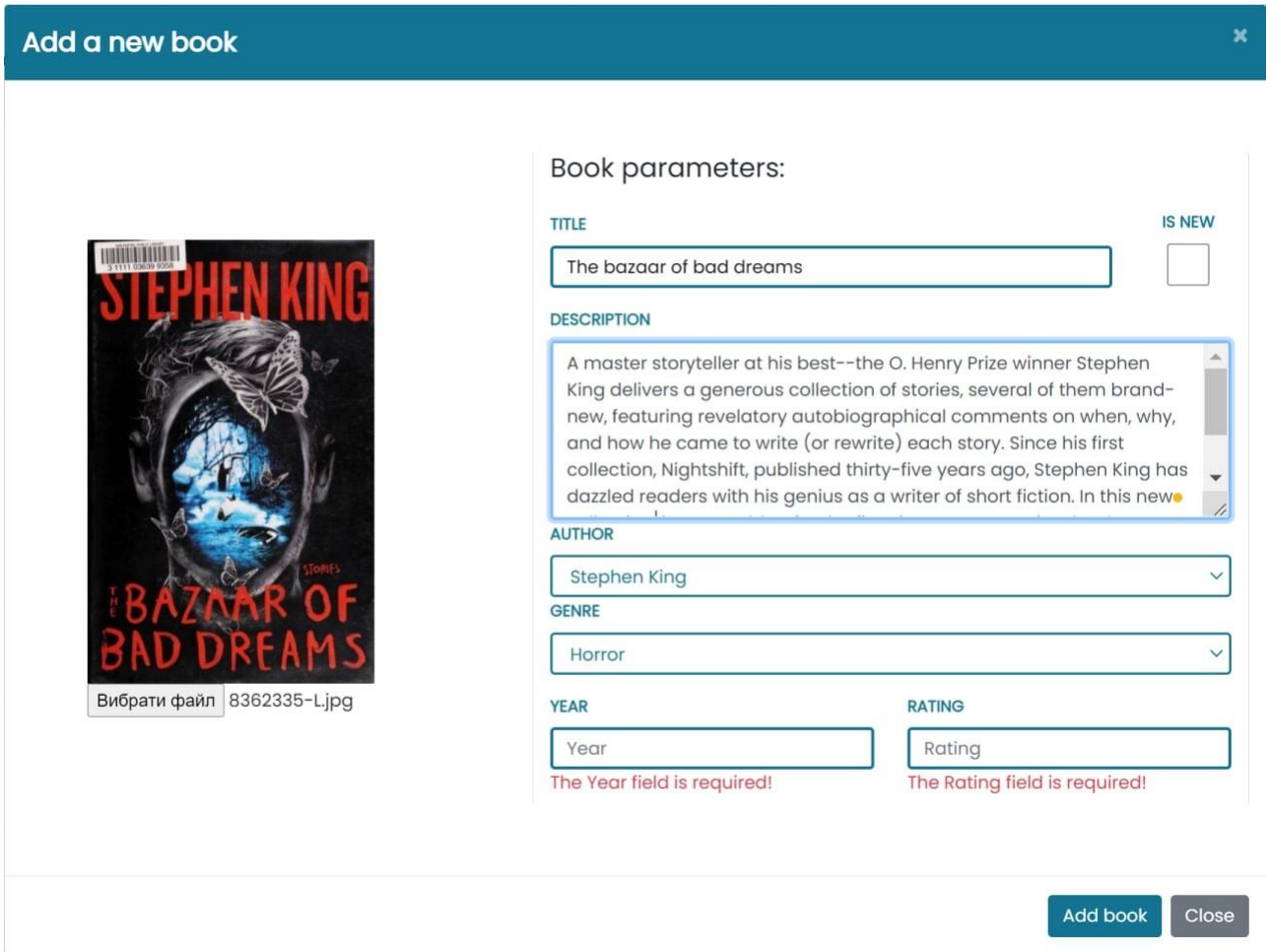


Рис. 6.4.10 - Результат Експерименту №2 для добавлення книги

6.5 Оцінювання та аналіз результатів

Провівши відповідні описані експерименти у пункті 6.4 не було виявлено ніяких помилок у роботі системи.

Розглядаючи експерименти для функцій авторизації, видно що очікуваний результат тестів та реальний результат експериментів сходяться, отже функція працює коректно.

Розглядаючи експерименти для функції перегляду списку книг, фільтрація, пошук, сортування, видно що очікуваний результат тестів та реальний результат експериментів сходяться, отже функція працює коректно.

Розглядаючи експерименти для функцій залишення коментарів для книги, та відгуків під коментарями, видно що очікуваний результат тестів та реальний результат експериментів сходяться, отже функція працює коректно.

Розглядаючи експерименти для функції створення та зберігання книжкових полиць, видно що очікуваний результат тестів та реальний результат експериментів сходяться, отже функція працює коректно.

Розглядаючи експерименти для функцій добавлення, оновлення, та видалення книги, видно що очікуваний результат тестів та реальний результат експериментів сходяться, отже функція працює коректно.

ВИСНОВКИ

В бакалаврській роботі була розглянута та спроектована веб-орієнтована інформаційно пошукова система для бібліотеки разом з клієнтом для роботи з нею, який являє собою веб-сайт.

Програмне рішення для сервера створювалось за допомогою об'єктно-орієнтованої мови C# на базі фреймоврка для побудови веб-сервісів ASP.NET Core та фреймворка роботи з базою даних Entity Framework Core. Програмне рішення клієнта було розроблено за допомогою фреймворка Angular за принципом SPA.

У першому розділі виконуваної роботи була сформована основна задача системи, та описані функції і задачі які буде вирішувати розроблена система. Основними функціями системи є: реєстрація/авторизація, швидкий пошук, фільтрація та сортування списку книг, ознайомлення з книгами та їхніми авторами, залишення відгуків про книги, можливість редагування профілю користувача, добавлення та перегляд власно сформованих книжкових полиць.

У другому розділі було оглянуто інформаційні джерела. Розглянуті інформаційні джерела були корисними для вирішення поставленої задачі, адже вони формують основу розробленої системи. Були розглянуті такі теми як: що таке веб-сервіс та принципи роботи веб-сервісу, що таке RESTful веб-сервіси, як працює та для чого потрібен протокол HTTP, а також як відбувається взаємодія між клієнтом та сервером через протокол HTTP.

У третьому розділі був наведений опис системи. Крім цього був здійснений системний аналіз системи з використанням таких методів як дерево цілей та дерево проблем. За допомогою дерево цілей були поставлені основні задачі які були реалізовані. Також були розглянуті та проаналізовані методи, алгоритми та засоби розв'язання задачі, та побудована структурна схема системи.

У четвертому розділі були зображені основні алгоритми роботи системи або її підсистем. Також були побудовані діаграми, що описують функціональність системи, а саме Use case діаграма, діаграма DFD та IDEF0,

IDEF3. Крім цього була зображена діаграма класів з відображенням класів підсистеми роботи з книгами, та зображена схема база даних.

У п’ятому розділі була наведена загальна структура програмного проекту, був наведений детальний опис усіх сторонніх бібліотек що використовувались для проектування системи. Був проведений опис основних програмних модулів, а також описаний інтерфейс користувача. Крім цього були проаналізовані альтернативні підходи, які розглядались під час розробки.

У шостому розділі були наведені інструкції адміністратору та програмісту по тому як запустити систему, або перенести її на інший комп’ютер. Були наведені бажані вимоги до апаратного забезпечення. Були побудовані та сформовані тести, та проведені на їх базі експерименти, в результаті чого був проведений аналіз та оцінювання результатів.

Отже, спроектована система являє собою готову до роботи інформаційно пошукову систему для бібліотеки. Система була спроектована таким чином, щоб додавати новий функціонал. Надалі у системі можна було б створити нотифікаційні підписки, користувач міг б підписуватись на якісь жанри книг, і коли нова книга цього жанру з'являлась в системі користувача, йому б приходило оповіщення. Також можна було б створити кімнати для обговорення книг, де користувачі могли б ділитись думками в режимі реального часу. Крім цього можна добавити функцію аналізу перегляду користувача та пропонуванняйому книг базуючись на його переглядах.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Самсонов В. В., Эрохін А. Л. Методи та засоби Інтернет-технологій; 2008, - 264с.
2. Мар'їна О. Ю. Веб-технології в бібліотеках: нові можливості розвитку комунікаційного середовища, 210с.
3. JWT токен - ознайомлення [Електронний ресурс] Режим доступу: https://jwt.io/introduction
4. Б. Кришнамурти, Дж. Рексфорд. Web-протоколи: Теорія і практика; 2002 - 592с.
5. Леонард Річардсон, RESTful Web APIs: Services for a Changing World 1st Edition; 2013 - 406с.
6. Що таке веб-сервіси: поняття, принципи роботи [Електронний ресурс] Режим доступу: https://uaeu.top/digital-online/shcho-take-veb-servisi-ponyattya-printsipi-roboti-gidnosti-ta-nedoliki.html
7. SPA принцип [Електронний ресурс] Режим доступу: https://medium.com/swlh/spa-development-principles-6f12b94908ce
8. Марк Річардс, Software Architecture Patterns; 2015 - 107с.
9. Джон Скіт, C# in Depth [3rd Edition]; 2013 - 604с.
10. What is Azure Cognitive Search? [Електронний ресурс] Режим доступу: https://docs.microsoft.com/en-us/azure/search/search-what-is-azure-search
11. Getting started with Angular [Електронний ресурс] Режим доступу: https://angular.io/start#getting-started-with-angular
12. Ерік Вогель, Beginning Entity Framework Core 5: From Novice to Professional; 2021 - 336с.
13. ASP.NET Core fundamentals [Електронний ресурс] Режим доступу: https://docs.microsoft.com/uk-ua/aspnet/core/fundamentals/?view=aspnetcore-6.0
14. TypeScript - The Basics [Електронний ресурс] Режим доступу: https://www.typescriptlang.org/docs/handbook/2/basic-types.html

15. Cloudinary - Developer get started guide [Електронний ресурс] Режим доступу: https://cloudinary.com/documentation/how_to_integrate_cloudinary
16. Introduction to Identity [Електронний ресурс] Режим доступу: https://jakeydocs.readthedocs.io/en/latest/security/authentication/identity.html
17. Георгій Гайна, Основи проектування баз даних; 2018 - 204с.
18. Repository Design Pattern [Електронний ресурс] Режим доступу: https://dotnettutorials.net/lesson/repository-design-pattern-csharp/
19. Matthias Biehl, RESTful API Design: Best Practices in API Design with REST; 2016 - 207с;
20. An Overview of HTTP [Електронний ресурс] Режим доступу: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview.

ДОДАТКИ

ДОДАТОК А

В даному додатку буде наведено програмне рішення для обробки запитів для роботи з книгами.

Клас BooksController, призначений для прийому запитів від клієнтів у вигляді HTTP запитів.

```
using BookList.Azure;
using BookList.Azure.Models.Filter;
using BookList.BLL.Dto.Book;
using BookList.BLL.Services;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using System;
using System.Threading.Tasks;

namespace BookList.Controllers
{
   [Route("api/[controller]")]
   [ApiController]
   public class BooksController : ControllerBase
   {
      private readonly IBookService _bookService;
      private readonly IBookSearchService _bookSearchService;

      public BooksController(IBookService bookService, IBookSearchService bookSearchService)
      {
         _bookService = bookService;
         _bookSearchService = bookSearchService;
      }

      [HttpGet]
      public async Task<ActionResult> Get()
      {
         var cardBooks = await _bookService.GetCardBooks();

         return Ok(cardBooks);
      }
```

```
[Authorize]
[HttpGet("{id}")]
public async Task<ActionResult> Get(string id)
{
    var bookDetail = await _bookService.GetBookDetailAsync(id);

    return Ok(bookDetail);
}

[Authorize(Roles = "Admin")]
[HttpPost]
public async Task<ActionResult> Create([FromBody] BookDetailDto bookDetailDto)
{
    await _bookService.CreateBookAsync(bookDetailDto);

    return Ok();
}

[Authorize(Roles = "Admin")]
[HttpDelete("{id}")]
public async Task<ActionResult> Delete(string id)
{
    await _bookService.DeleteBookAsync(id);

    return Ok();
}

[Authorize(Roles = "Admin")]
[HttpPut]
public async Task<ActionResult> Update(BookDetailDto bookDetailDto)
{
    await _bookService.UpdateBook(bookDetailDto);

    return Ok();
}

[HttpPost]
[Route("search")]
public async Task<ActionResult> Search(BookSearchFilter bookSearchFilter)
{
    var result = await _bookSearchService.Search(bookSearchFilter);

    return Ok(result.Entities);
}
```

```
        [Authorize(Roles = "Admin")]
        [HttpGet]
        [Route("reload")]
        public ActionResult ReloadBooksData()
        {
            try
            {
                var azureManager = new AzureManager();
                azureManager.Start();
            }
            catch (Exception ex)
            {
                throw new ApplicationException(ex.Message, ex);
            }

            return Ok();
        }
    }
}
```

Клас BookService, виконує всю бізнес логіку пов’язану з роботою книг.

```
using AutoMapper;
using  BookList.Azure.Models;
using BookList.Azure.Models.Filter;
using BookList.Azure.Services.Contracts;
using BookList.BLL.Dto.Book;
using BookList.BLL.Services;
using BookList.DAL.Entities;
using BookList.DAL.UnitOfWork;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Threading.Tasks;

namespace BookList.BLL.Implementations
{
    public class BookService : IBookService
    {
        private readonly IUnitOfWork _unitOfWork;
        private readonly IMapper _mapper;
        private readonly IImageService _imageService;
        private readonly ICommentService _commentService;
```

```
    private readonly IBookSearchService _bookSearchService;
    private readonly IAzureSearchRepository _azureSearchRepository;

    public BookService(
        IUnitOfWork unitOfWork,
        IMapper mapper,
        IImageService imageService,
        ICommentService commentService,
        IBookSearchService bookSearchService,
        IAzureSearchRepository azureSearchRepository)
    {
        _unitOfWork = unitOfWork;
        _mapper = mapper;
        _imageService = imageService;
        _commentService = commentService;
        _bookSearchService = bookSearchService;
        _azureSearchRepository = azureSearchRepository;
    }

    public async Task CreateBookAsync(BookDetailDto bookDetailDto)
    {
        var book = _mapper.Map<Book>(bookDetailDto);

        byte[] bytes = Convert.FromBase64String(bookDetailDto.Avatar);

        using Stream ms = new MemoryStream(bytes);

  var avatarLink = await _imageService.UploadImageAsync("Books", $"{book.Title}_{bookDetailDto.AuthorName}",
ms);

        book.Avatar = avatarLink;
        book.ID = Guid.NewGuid().ToString();

        await _unitOfWork.Books.CreateAsync(book);
        var success = await _unitOfWork.SaveChangesAsync();

        var document = _mapper.Map<BookAzureDto>(book);

        document.AuthorName = bookDetailDto.AuthorName;

        await _azureSearchRepository.MergeOrUploadDocumentsAsync(new List<BookAzureDto> { document });

        if (!success)
        {
            throw new Exception("The new book was not added!");
```

```
    }
}

public async Task DeleteBookAsync(string id)
{
    await _unitOfWork.Books.DeleteAsync(id);
    var success = await _unitOfWork.SaveChangesAsync();

    if (!success)
    {
        throw new Exception($"The book - {id} was not deleted!");
    }

    await _azureSearchRepository.DeleteDocumentsAsync(new List<string> { id });
}

public async Task<IEnumerable<Book>> GetAllBookAsync()
{
    var books = await _unitOfWork.Books.GetAllAsync();

    return books;
}

public async Task<BookDetailDto> GetBookDetailAsync(string id)
{
    var book = await _unitOfWork.Books.GetAsync(id, book => book.Author);

    if (book == null)
    {
        throw new Exception($"The book - {id} was not found!");
    }

    var bookDetail = _mapper.Map<BookDetailDto>(book);

    bookDetail.Comments = await _commentService.GetBookCommentsAsync(book.ID);

    var filter = new BookSearchFilter()
    {
        Genre = book.Genre.ToString(),
        PageSize = 7,
        PageNumber = 1,
        SortBy = nameof(BookAzureDto.Rating),
        IsAscedingOrder = false,
        IncludeTotalCount = true,
        Filter = $"{nameof(BookAzureDto.ID)} ne '{book.ID}'"
```

```
        };

        var searchResult = await _bookSearchService.Search(filter);

        if (searchResult.Entities != null && searchResult.Entities.Any())
        {
            bookDetail.SimilarBooks = searchResult.Entities
                .Select(card => _mapper.Map<CardBookShort>(card))
                .ToList();
        }
        else
        {
            bookDetail.SimilarBooks = new List<CardBookShort>();
        }

        return bookDetail;
    }

    public async Task UpdateBook(BookDetailDto bookDetailDto)
    {
        var book = _mapper.Map<Book>(bookDetailDto);

        if (!bookDetailDto.Avatar.Contains("cloudinary"))
        {

            byte[] bytes = Convert.FromBase64String(bookDetailDto.Avatar);

            using Stream ms = new MemoryStream(bytes);

                                    var avatarLink = await _imageService.UploadImageAsync("Books", $"{book.Title}
_{bookDetailDto.AuthorName}", ms);

            book.Avatar = avatarLink;
        }

        _unitOfWork.Books.Update(book);
        var success = await _unitOfWork.SaveChangesAsync();

        if (!success)
        {
            throw new Exception($"The book was not updated!");
        }

        var document = _mapper.Map<BookAzureDto>(book);
```

```
            document.AuthorName = bookDetailDto.AuthorName;

            await _azureSearchRepository.MergeOrUploadDocumentsAsync(new List<BookAzureDto> { document });
        }

        public async Task<IEnumerable<CardBookDto>> GetCardBooks()
        {
            var books = await _unitOfWork.Books.GetAllAsync(null, b => b.Author);

            var cardBooks = _mapper.Map<IEnumerable<CardBookDto>>(books);

            return cardBooks;
        }
    }
}
```

Клас UnitOfWork, реалізує паттерн UnitOfWork для об’єднання усіх репозиторіїв системи, що здійснюють запити до бази даних.

```
using BookList.DAL.Data;
using BookList.DAL.Entities;
using BookList.DAL.UnitOfWork.Repository;
using System.Threading.Tasks;

namespace BookList.DAL.UnitOfWork
{
    public class UnitOfWork : IUnitOfWork
    {
        private readonly BookListDbContext _bookListContext;

        private IRepository<Book> _books;

        private IRepository<Author> _authors;

        private IRepository<Comment> _comments;

        private IRepository<Bookshelf> _bookshelves;

        private IRepository<BookshelfBook> _bookshelveBooks;

        public UnitOfWork(BookListDbContext bookListContext)
        {
            _bookListContext = bookListContext;
        }
```

```
        public IRepository<Book> Books => _books ??= new Repository<Book>(_bookListContext);

        public IRepository<Author> Authors => _authors ??= new Repository<Author>(_bookListContext);

        public IRepository<Comment> Comments => _comments ?? new Repository<Comment>(_bookListContext);
       public IRepository<Bookshelf> Bookshelves => _bookshelves ?? new Repository<Bookshelf>(_bookListContext);
public IRepository<BookshelfBook> BookshelveBooks => _bookshelveBooks ?? new
Repository<BookshelfBook>(_bookListContext);

        public async Task<bool> SaveChangesAsync()
        {
            return await _bookListContext.SaveChangesAsync() > 0;
        }
    }
}
```

Клас Repository, даний клас призначений для здійснення запитів до бази даних.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Linq.Expressions;
using System.Threading.Tasks;
using BookList.DAL.Data;
using BookList.DAL.Entities;
using Microsoft.EntityFrameworkCore;

namespace BookList.DAL.UnitOfWork.Repository
{
    public struct Repository<TEntity> : IRepository<TEntity> where TEntity : class, IEntity
    {
        private readonly DbSet<TEntity> _dbSet;
        private readonly BookListDbContext _dbContext;

        public Repository(BookListDbContext context)
        {
            _dbContext = context;
            _dbSet = _dbContext.Set<TEntity>();
        }

        public async Task CreateAsync(TEntity entity)
        {
            await _dbSet.AddAsync(entity);
        }
```

```
    public async Task DeleteAsync(string entityId)
    {
        var entity = await GetAsync(entityId);

        _dbSet.Remove(entity);
    }

    public void DeleteRange(params TEntity[] entities)
    {
        _dbSet.RemoveRange(entities);
    }

    public async Task<TEntity> GetAsync(string entityId)
    {
        return await _dbSet.FirstOrDefaultAsync(entity => entity.ID == entityId);
    }

    public async Task<IEnumerable<TEntity>> GetAllAsync()
    {
        return await _dbSet.ToListAsync();
    }

    public void Update(TEntity entity)
    {
        _dbSet.Attach(entity);
        _dbContext.Entry(entity).State = EntityState.Modified;
    }

     public async Task<IEnumerable<TEntity>> GetAllAsync(Expression<Func<TEntity, bool>> condition = null,
params Expression<Func<TEntity, object>>[] includeExpressions)
    {
        IQueryable<TEntity> set = _dbSet;

        if (condition != null)
        {
            set = set.Where(condition);
        }

        foreach (var includeExpression in includeExpressions)
        {
            set = set.Include(includeExpression);
        }

        return await set.ToListAsync();
```

```
        }

                public async Task<TEntity> GetAsync(string id, params Expression<Func<TEntity, object>>[]
includeExpressions)
        {
            IQueryable<TEntity> set = _dbSet;

            set = set.Where(entity => entity.ID == id);

            foreach (var includeExpression in includeExpressions)
            {
                set = set.Include(includeExpression);
            }

            return await set.FirstOrDefaultAsync();
        }
    }
}
```

Клас ImageService, здійснює запити до Cloudinary API, для зберігання зображень.

```
using BookList.BLL.Implementations;
using CloudinaryDotNet;
using CloudinaryDotNet.Actions;
using Microsoft.Extensions.Configuration;
using System.IO;
using System.Net;
using System.Threading.Tasks;

namespace BookList.BLL.Services
{
    public class ImageService : IImageService
    {
        private readonly Cloudinary _cloudinary;
        private readonly IConfiguration _configuration;

        public ImageService(IConfiguration configuration)
        {
            _configuration = configuration;

            Account account = new Account()
            {
                Cloud = _configuration.GetValue<string>("CloudinaryApi:CloudName"),
```

```
            ApiKey = _configuration.GetValue<string>("CloudinaryApi:ApiKey"),
            ApiSecret = _configuration.GetValue<string>("CloudinaryApi:ApiSecret")
        };

        _cloudinary = new Cloudinary(account);
    }

    public async Task<string> UploadImageAsync(string folderName, string filePublicId, Stream fileStream)
    {
        if (!string.IsNullOrEmpty(folderName) && !string.IsNullOrEmpty(filePublicId))
        {
            var uploadParams = new ImageUploadParams()
            {
                Folder = folderName,
                File = new FileDescription(filePublicId, fileStream),
                PublicId = filePublicId
            };

            var uploadResult = await _cloudinary.UploadAsync(uploadParams);

            if (uploadResult.StatusCode == HttpStatusCode.OK)
            {
                return uploadResult.Url.ToString();
            }
        }

        return string.Empty;
    }
  }
}
```

Клас BookSearchService використовує клас BookAzureSearchService для здійснення запитів до Azure Search.

```
using AutoMapper;
using BookList.Azure.Models.Filter;
using BookList.Azure.Services.Contracts;
using BookList.BLL.Dto.Book;
using BookList.BLL.Dto.Search;
using BookList.BLL.Services;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace BookList.BLL.Implementations
{
```

```
    public class BookSearchService : IBookSearchService
    {
        private readonly IBookAzureSearchService _bookAzureSearchService;
        private readonly IMapper _mapper;

        public BookSearchService(IBookAzureSearchService bookAzureSearchService, IMapper mapper)
        {
            _bookAzureSearchService = bookAzureSearchService;
            _mapper = mapper;
        }

        public async Task<SearchResult<CardBookDto>> Search(BookSearchFilter bookSearchFilter)
        {
            var response = await _bookAzureSearchService.Search(bookSearchFilter);

            var result = new SearchResult<CardBookDto>()
            {
                Entities = _mapper.Map<IEnumerable<CardBookDto>>(response.Documents),
                TotalCount = response.TotalCount
            };

            return result;
        }
    }
}
```

Клас BookAzureSearchService здійснює запити до Azure Search, для фільтрації, пошуку, сортування даних.

```
using Azure;
using Azure.Search.Documents;
using  BookList.Azure.Models;
using BookList.Azure.Models.Filter;
using BookList.Azure.Models.Response;
using BookList.Azure.Services.Contracts;
using System;
using System.Globalization;
using System.Linq;
using System.Threading.Tasks;
using SearchMode = Azure.Search.Documents.Models.SearchMode;

namespace BookList.Azure.Services.Implementations
{
    public class BookAzureSearchService : IBookAzureSearchService
    {
```

```
 private string serviceName = "booklist";
private string indexName = "books";
 private readonly SearchClient _searchClient;

 public BookAzureSearchService()
{
    var serviceEndpoint = new Uri($"https://{serviceName}.search.windows.net/");
    var credential = new AzureKeyCredential(apiKey);
    _searchClient = new SearchClient(serviceEndpoint, indexName, credential);
}

public async Task<AzureSearchResponse<BookAzureDto>> Search(BookSearchFilter filter)
{
    var searchOptions = BuildSearchOptions(filter);

    var response = await _searchClient.SearchAsync<BookAzureDto>($"{filter.SearchTerm}*", searchOptions);

    var result = new AzureSearchResponse<BookAzureDto>
    {
        Documents = response?.Value?.GetResults().Select(d => d.Document),
        TotalCount = response?.Value?.TotalCount.HasValue == true ?
            Convert.ToInt32(response.Value.TotalCount) : 0
    };

    return result;
}

private SearchOptions BuildSearchOptions(BookSearchFilter filter)
{
    UpdateBookFilter(filter);

    var searchOptions = new SearchOptions()
    {
        Size = filter.PageSize,
        Skip = filter.PageNumber == default(int) ? default(int) : (filter.PageNumber - 1) * filter.PageSize,
        Filter = filter.Filter,
        IncludeTotalCount = filter.IncludeTotalCount
    };

    if (!string.IsNullOrEmpty(filter.SortBy))
    {
        string order = filter.IsAscedingOrder ? "asc" : "desc";

        searchOptions.OrderBy.Add($"{filter.SortBy} {order}");
    }
```

```
        searchOptions.SearchMode = SearchMode.Any;

        return searchOptions;
    }

    private void UpdateBookFilter(BookSearchFilter filter)
    {
        UpdateFilterString(filter, nameof(BookAzureDto.AuthorName), filter.AuthorName);
        UpdateFilterString(filter, nameof(BookAzureDto.Genre), filter.Genre);

        UpdateFilterBasedOnNew(filter, nameof(BookAzureDto.IsNew), filter.IsNew);

        UpdateFilterBasedOnRating(filter, filter.Rating);

        UpdateFilterBasedOnFromTo(filter);
    }

     private void UpdateFilterString(SearchFilterBase filterBase, string column, string columnValue, string separator =
","")
    {
        if (string.IsNullOrEmpty(column) || string.IsNullOrEmpty(columnValue)) return;

        var filterString = $"{column} eq '{columnValue}'";

        if (columnValue.Contains(separator))
        {
            filterString = $"search.in({column}, '{columnValue}', '{separator}')";
        }

                    filterBase.Filter = string.IsNullOrEmpty(filterBase.Filter) ? filterString : $"{filterBase.Filter} and
{filterString}";
    }

    private void UpdateFilterBasedOnNew(SearchFilterBase filterBase, string column, bool? columnValue)
    {
        if (columnValue == false || columnValue == null) return;

        var filterString = $"{column} eq true";

                    filterBase.Filter = string.IsNullOrEmpty(filterBase.Filter) ? filterString : $"{filterBase.Filter} and
{filterString}";
    }

    private void UpdateFilterBasedOnRating(SearchFilterBase filterBase, double rating)
```

```
    {
        var column = nameof(BookAzureDto.Rating);
        string filterString = null;

        var provider = new NumberFormatInfo();
        provider.NumberDecimalSeparator = ".";
        provider.NumberGroupSeparator = ".";

        if (rating == 0)
        {
            filterString = null;
        }
        else if (rating <= 1)
        {
            filterString = $"{column} le {Convert.ToString(rating, CultureInfo.InvariantCulture)}";
        }
        else
        {
            filterString = $"{column} ge {Convert.ToString(rating - 1, CultureInfo.InvariantCulture)} and {column} le
{Convert.ToString(rating, CultureInfo.InvariantCulture)}";
        }

        if (!string.IsNullOrEmpty(filterString))
        {
                    filterBase.Filter = string.IsNullOrEmpty(filterBase.Filter) ? filterString : $"{filterBase.Filter} and
{filterString}";
        }
    }

    private void UpdateFilterBasedOnFromTo(BookSearchFilter filter)
    {
        string filterString = null;
        var column = nameof(BookAzureDto.Year);

        if (filter.From == null) return;

        if (filter.To == null)
        {
            filterString = $"{column} ge {filter.From}";
        }
        else
        {
            filterString = $"{column} ge {filter.From} and {column} le {filter.To}";
        }
```

```
        filter.Filter = string.IsNullOrEmpty(filter.Filter) ? filterString : $"{filter.Filter} and {filterString}";
    }
}
```