МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ВОДНОГО ГОСПОДАРСТВА ТА ПРИРОДОКОРИСТУВАННЯ

«До захисту допущений»

Зав. Кафедри комп’ютерних наук та прикладної математики

д.т.н. професор Турбал Ю.В.

«___»_____2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Створення інтерактивної книжкової галереї за допомогою Blazor та Azure»

Виконав: Мосійчук Денис Ігорович

студент навчально-наукового інституту автоматики, кібернетики та обчислювальної техніки

група КН-41

_____

Керівник: , к.ф.-м.н, доцент Гладун Л. В.

_____

Національний університет водного господарства та природокористування

Рівне 2023

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Мосійчуку Денису Ігоровичу

1. Тема роботи Створення інтерактивної книжкової галереї за допомогою Blazor та Azure
керівник роботи к.ф.-м.н, доцент комп'ютерних наук та прикладної математики Гладун Л. В.
затверджена наказом вищого навчального закладу від «19» квітня 2023 року С №-449
2. Термін подання роботи студентом 31.05.2023
3. Вихідні дані до роботи технології, що необхідні для розробки інтерактивної книжкової галереї за допомогою Blazor та Azure.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити).

РОЗДІЛ 1. КОРОТКИЇ ОГЛЯД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.
РОЗДІЛ 2. РОЗГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ.
РОЗДІЛ 3. ХІД ВИКОНАННЯ.
РОЗДІЛ 4. РЕЗУЛЬТАТИ РОБОТИ.

5. Перелік графічного матеріалу мультимедійна презентація.
6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Розділ 1	к.ф.-м.н, доцент комп'ютерних наук та прикладної математики Гладун Л. В.	20.12.2022	20.12.2022
Розділ 2	к.ф.-м.н, доцент комп'ютерних наук та прикладної математики Гладун Л. В.	03.01.2023	03.01.2023
Розділ 3	к.ф.-м.н, доцент комп'ютерних наук та прикладної математики Гладун Л. В.	20.05.2023	20.05.2023

7. Дата видачі завдання 20.12.2022

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Вивчення літератури за обраною тематикою	01.10.2022 – 31.11.2022	виконав
2	Формулювання постановки задачі	01.12.2022 – 07.12.2022	виконав
3	Вибір стеку технологій	30.12.2022 – 08.01.2023	виконав
4	Збір та аналіз даних	09.01.2023 – 10.02.2023	виконав
5	Розробка алгоритму	11.02.2023 – 26.03.2023	виконав
6	Розробка користувацького інтерфейсу	27.03.2023 – 15.04.2023	виконав
7	Відлагодження програмного продукту	10.05.2023 – 13.05.2023	виконав
8	Загальні висновки до роботи	14.05.2023 – 23.05.2023	виконав
9	Підготовка звіту кваліфікаційної роботи	24.05.2023 – 07.06.2023	виконав
10	Підготовка мультимедійної презентації	08.06.2023 – 12.06.2023	виконав
11	Підготовка до виступу	13.06.2023 – 22.06.2023	виконав

Студент _____Мосійчук Д. І.

(підпис) (прізвище та ініціали)

Керівник роботи _____Гладун Л. В._____

(підпис) (прізвище та ініціали)

РЕФЕРАТ

Кваліфікаційна робота: 41 с., 11 малюнки, 6 джерел.

Мета роботи: Реалізувати інтерактивну книжкову галерею.

Об'єкт роботи: Програмне забезпечення, призначене для роботи з базами даних, веб-застосунками таких як Blazor та Azure та серверною частиною.

Предмет роботи: Реалізація книжкової галереї для швидкого та зручного огляду книг

Методи дослідження - Проектування архітектури та функціональності веб-додатку на основі аналізу вимог користувачів та особливостей технічних засобів.

Результатом дипломної роботи є розроблений функціональний веб-додаток інтерактивна книжкова галерея, яка дозволяє здійснювати огляд, додавання, редагування, оцінювання, здійснювати швидкий пошук необхідної літератури за тегами та обговорення книг. Використано при цьому веб-фреймворк Blazor та використано хмарний серверний сервіс Azure для серверного користування сторінкою. Також для роботи та збереження даних використано базу даних sql Server.

Ключові слова: BLAZOR, RAZOR, ENTITY FRAMFORK, MICROSOFT SQL SERVER, AZURE.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

HTML - HyperText Markup Language

БД - Бази даних

СУБД – Система управління базами даних

AZURE – це платформа хмарних обчислень, яка пропонує доступ, керування та розробку програм і послуг через глобальні центри обробки даних

BLAZOR – веб-платформа з відкритим вихідним кодом, яка дозволяє розробникам створювати веб-програми за допомогою C# і HTML

ENTITY FRAMVORK- структура об'єктно-реляційного відображення з відкритим кодом для ADO.NET.

Зміст

Анотація **6**

Вступ **7**

1. Короткиї огляд програмного забезпечення **8**

1.1 Що таке Blazor 8

1.2 Azure 11

1.3 SQL 16

1.4 Що таке Entity Framework 17

2. Розгляд предметної області **20**

2.1 Книга 20

2.2 Пошукова система 20

3. Хід виконання **21**

3.1 Класи 21

3.2 NavMenu 26

3.3 Наповнення сайту вмістом та додавання відповідних інструкцій 27

4. Результати роботи **34**

4.1 Авторизація 34

4.2 Робота пошуку 35

4.3 Добавлення коментарів та книги 37

Висновки 40

Список використаних джерел 41

Анотація

Створено книжкову галерею для легкого опису та пошуку книг. Для створення використовували VS на мові програмування C# користуючись зручним фрейворком Blazor, платформою хмарних обчислень Azure та MySQL з entity framework для реалізації бази даних та роботи з нею.

Вступ

Світовий процес переходу суспільства до інформаційно-комп'ютерних технологій має революційний вплив на всі сфери суспільного життя і вимагає суттєвих змін у багатьох галузях діяльності людини. Саме тому сьогодні актуальною стала потреба розвитку програм з використанням структурованих таблиць, бази даних. Вони суттєво підвищують продуктивність роботи користувачів.

Загалом база даних - це сукупність описів об‘єктів реального світу та зв‘язків між ними, які відносяться до конкретної предметної області (частини реального світу, до якої відносяться об‘єкти бази даних) і використовуються для вирішення задач у межах цієї області. Схема, яка відображає об‘єкти предметної області та зв'язки між ними, називається концептуальною або логічною схемою БД.

Також гостро стоїть питання літератури а саме її швидкому пошуку та можливості збереження. Оскільки світ стрімко розвивається і інформації стає дедалі більше, що сприяє збільшенню зацікавлення в сучасних книгах та книгах, що мають певне значення в минулому залежно від поставленої мети та галузі використання тих чи інших прийомів, процесів тощо. Також можливе використання літератури не лише в наукових цілях а також в розважальних, ознайомчих, соціальних та інших. Саме література розвиває свідомість людини, її почуття, волю, психіку, формує людський характер — отже, створює особистість.

Саме тому тему літератури та роботи з нею необхідно розвивати та модернізувати залежно до змін та потреб користувачів.

Розділ 1

Короткиї огляд програмного забезпечення

1.1 Що таке Blazor

Blazor — чудова ідея в технології .NET. Далі ми побачимо, що таке Blazor і як його використовувати для створення веб-додатків.

Програми Blazor базуються на компонентах. Компонент у Blazor — це елемент інтерфейсу користувача, такий як сторінка, діалогове вікно або форма введення.

Компоненти — це класи .NET C#, вбудовані в збірки .NET, наприклад:

Інтерфейс користувача визначає гнучку логіку візуалізації, підтримує керування подіями користувача, вкладення та повторне використання, його можна публікувати та розповсюджувати як бібліотеки класів Razor або пакети NuGet.

Клас компонента написаний для представлення розмітки Razor на основі файлу .razor. Компоненти Blazor офіційно називаються компонентами Razor, неофіційно — компонентами Blazor. Razor — це синтаксис, який поєднує розмітку HTML із кодом C#, призначений для підвищення продуктивності розробників. Razor підтримує розмітку HTML і C#, використовуючи той самий документ, що й довідка з програмування IntelliSense у Visual Studio. Razor Pages і MVC використовують Razor. На відміну від Razor Pages і MVC, які побудовані навколо моделі запит/відповідь, компоненти використовуються спеціально для логіки та композиції інтерфейсу користувача на стороні клієнта.

Blazor використовує природні теги HTML для створення інтерфейсу користувача. Компонент Razor (Dialog.razor) надає компонент, який відображає діалогове вікно та обробляє подію, коли користувач натискає кнопку OnYes. OnYes — це функція C#, яка замінює функцію onclick кнопки. Цей компонент призначає текст (ChildContent) і назву (Title) діалогового вікна інтерфейсу користувача, який використовує цей компонент.

Компонент Dialog містить HTML-тег іншого компонента. Передайте атрибут Title мітки в значення title властивості Title компонента Dialog. Текст компонента Dialog (ChildContent) визначається вмістом елемента <Dialog>. Коли компонент Dialog додається до компонента Index, IntelliSense прискорює розробку шляхом автоматичного завершення синтаксису та параметрів у Visual Studio.

Діалогове вікно відображається під час доступу до компонента Index у браузері. Коли користувач вибирає кнопку, консоль інструментів розробника браузера показує повідомлення, написане методом OnYes:

Компоненти представлені в пам’яті в об’єктній моделі документа (DOM) браузера, яка називається деревом візуалізації, яка використовується для гнучкого й ефективного оновлення інтерфейсу користувача.

Blazor Server дозволяє розміщувати компоненти Razor у програмі ASP.NET Core на сервері. Оновлення інтерфейсу користувача обробляються через SignalR.

SignalR — це бібліотека, яка дозволяє створювати веб-додаток, за допомогою якого клієнти можуть встановлювати постійне з’єднання та обмінюватися даними.

Середа виконання залишається на сервері та обробляє:

a) Запуск код програми C#.
b) Пересилання подій інтерфейсу користувача з браузера на сервер.
c) Застосування оновлення інтерфейсу користувача до відтвореного компонента, отриманого з сервера.
d) З’єднання Blazor Server для зв’язку з браузером також використовується для обробки викликів підключення JavaScript.

Серверні програми Blazor відтворюють вміст, який відрізняється від попередніх шаблонів для відтворення інтерфейсу користувача в програмах ASP.NET Core за допомогою Razor Views або Razor Pages. Обидві моделі використовують мову Razor для візуалізації HTML, але суттєво відрізняються способом рендерингу закладок.

Коли сторінка Razor відтворюється або відображається, кожен рядок коду Razor відображає текст HTML. Після презентації сервер видаляє екземпляр сторінки або перегляду, незалежно від згенерованого стану. Коли запитується інша сторінка, вся сторінка відображається в HTML і надсилається клієнту.

Blazor Server створює віджет для відображення в HTML або XML Document Object Model (DOM). Графік компонентів відображає стан властивостей і полів. Blazor аналізує графік компонента, щоб створити двійкове представлення, яке він надсилає клієнту для візуалізації. Після встановлення з’єднання між клієнтом і сервером попередньо відрендерені статичні елементи компонента замінюються інтерактивними елементами. Попередній перегляд вмісту на сервері робить програму більш чутливою до клієнта.

Компоненти стають інтерактивними в клієнті, а інновації інтерфейсу користувача покращуються завдяки взаємодії з користувачем і функціональності програми. Коли відбувається оновлення, віджет перемальовується, а зміна інтерфейсу користувача (відмінність) перевіряється. Це незначні зміни DOM, необхідні для оновлення інтерфейсу клієнта. Зміни надсилаються клієнту в двійковому форматі та керуються браузером.

Компонент видаляється, коли користувач відходить від нього

Blazor WebAssembly — це платформа Single Page Application (SPA) для створення інтерактивних веб-клієнтських програм за допомогою .NET.

Запуск коду .NET у веб-браузерах виконується WebAssembly (скорочено wasm). WebAssembly — це компактний формат байт-коду, який швидко завантажується та оптимізований для високої швидкості виконання. WebAssembly — це веб-програма з відкритим вихідним кодом, яка підтримується браузерами без плагінів. WebAssembly працює у всіх сучасних веб-браузерах, включаючи мобільні браузери. [1]

Код WebAssembly, який отримує доступ до всіх функцій у веб-браузері через JavaScript, називається JavaScript interop, часто скорочено до JavaScript interop або JS interop. Код .NET, який виконується за допомогою WebAssembly у браузері, працює в пісочниці JavaScript браузера, а захист від зловживань пісочницею забезпечується клієнтським пристроєм.

Коли ви створюєте та запускаєте програму Blazor WebAssembly:

a) Компілюйте файли коду C# і файли Razor у збірки .NET.
b) Завантажте збірки .NET і час виконання з браузера.

Blazor WebAssembly завантажує середовище виконання .NET і налаштовує середовище виконання для програми для завантаження збірок. Середовище виконання Blazor WebAssembly використовує взаємодії JavaScript для обробки маніпуляцій об’єктною моделлю документа (DOM) і викликів API браузера.

Розмір представленої програми, розмір корисного навантаження, відіграє ключову роль у визначенні її зручності використання. Завантаження великої веб-програми у ваш браузер займає деякий час, що впливає на взаємодію з користувачем. Blazor WebAssembly оптимізує розмір корисного навантаження, щоб зменшити час завантаження:

Невикористаний код видаляється з програми, коли видаляється тример Intermediate Language (IL). Відповіді HTTP увімкнено. Збірки середовища виконання та .NET кешуються в браузері.

Гібридні додатки використовують поєднання нативних і веб-технологій. Додаток Blazor Hybrid використовує Blazor у рідній клієнтській програмі.

Компоненти Razor працюють безпосередньо в процесі .NET, а веб-інтерфейс надає вбудований елемент керування Web View із локальним каналом взаємодії. WebAssembly не використовується в гібридному режимі. Гібридні програми включають такі технології:

a) .NET Multi-platform App UI (.NET MAUI): крос-платформна структура для розробки мобільних і настільних програм на C# та XAML.
b) Windows Presentation Foundation (WPF): фреймворк інтерфейсу користувача, що не залежить від роздільної здатності, заснований на

механізмі векторної візуалізації, призначеному для використання переваг сучасного графічного обладнання.

c) Windows Forms: структура інтерфейсу користувача, яка створює багатофункціональні клієнтські програми для робочого столу для Windows. Авторська платформа Windows Forms дозволяє створювати широкий спектр функціональних можливостей, включаючи елементи керування, діаграми, введення даних і введення користувачами.

Для програм, яким потрібен доступ до сторонніх бібліотек JavaScript і API браузера, компоненти взаємодіють із JavaScript. Компоненти можуть використовувати будь-яку бібліотеку чи API, які підтримуються JavaScript. Код C# може викликати код JavaScript, а JavaScript може викликати код C#.

Blazor сумісний із .NET Standard, що дозволяє проектам Blazor посилатися на бібліотеки, які відповідають специфікаціям .NET Standard. Стандарт .NET — це формальна специфікація API .NET, спільна для реалізацій .NET. Бібліотеки класів .NET Standard можна спільно використовувати на таких платформах .NET, як Blazor, .NET Framework, .NET Core, Xamarin, Mono та Unity.

Якщо API не реалізовано у веб-переглядачі, створюється виняткова ситуація PlatformNotSupportedException. [3, 4]

1.2 Azure

Microsoft вперше оголосила про свій намір запропонувати службу хмарних обчислень під назвою Windows Azure у 2008 році. Попередні версії служби були оголошені та запущені, щоб стати комерційно доступними на початку 2010-х років, але ранні версії хмарних служб Azure відставали від більш усталених пропозицій, таких як Портфоліо AWS продовжує розширюватися та підтримує широку базу мов програмування, фреймворків та операційних систем.

До початку 2014 року Microsoft додала й оновила кілька функцій, зокрема Azure SQL, Windows Azure CTP, Windows Azure Connect, Traffic Manager і HPC Scheduler. Корпорація Майкрософт визнає, що вплив хмарних обчислень виходить за межі Windows, і перейменувала службу в Microsoft Azure. Крім того, Azure є першою публічною пропозицією своїх послуг машинного навчання.

У найближчі роки Azure представить SONiC (кросплатформний дистрибутив Linux), Azure ARM Portal (2015), Azure Service Fabric (2016), Azure Service Fabric Mesh (2018) і Azure IoT Central (2018). Сьогодні Azure вважається сильним комерційним конкурентом іншим публічним хмарним провайдерам. Microsoft Azure, раніше відома як Windows Azure, є публічною платформою хмарних обчислень Microsoft. Він пропонує широкий спектр хмарних послуг, включаючи обчислення, аналітику, зберігання та мережу. Користувачі можуть вибирати з цих служб для створення та масштабування нових програм або керування існуючими програмами в загальнодоступній хмарі.

Платформа Azure має на меті допомогти компаніям керувати ризиками та досягати організаційних цілей. Він пропонує інструменти, які обслуговують усі галузі, включаючи електронну комерцію, фінанси та різноманітні компанії зі списку Fortune 500, і включає технології з відкритим кодом. Це надає користувачам гнучкість у використанні улюблених інструментів і технологій. Крім того, Azure пропонує чотири різні типи хмарних обчислень: інфраструктура як послуга (IaaS), платформа як послуга (PaaS), програмне забезпечення як послуга (SaaS) і безсерверне.

Корпорація Майкрософт виставляє рахунки Azure на основі оплати за використання (PAYG), що означає, що передплатникам щомісяця виставляються рахунки лише за певні ресурси та послуги, якими вони користуються.

Після реєстрації в Azure клієнти матимуть доступ до всіх послуг, доступних на порталі Azure. Абоненти можуть використовувати ці послуги для створення хмарних ресурсів, таких як віртуальні машини та бази даних. Потім продукти та служби Azure можна згрупувати в робочі області, які використовуються для розміщення робочих навантажень і зберігання даних.

Крім того, що Microsoft надає його через портал Azure, багато сторонніх постачальників також пропонують програмне забезпечення безпосередньо через Azure. Плата за сторонні програми різна, але може включати плату за підписку на програму, а також плату за інфраструктуру, яка використовується для розміщення програми.

Microsoft пропонує такі п’ять варіантів підтримки клієнтів для Azure:

a) Основний
b) Стандарт
c) Компанія.
d) Автор
e) Професійний прямий

Ці плани підтримки клієнтів відрізняються за обсягом і ціною. Базова підтримка доступна для всіх облікових записів Azure, але Microsoft стягує плату за додаткові пропозиції підтримки. Підтримка розробників коштує 29 доларів на місяць, тоді як стандартна підтримка коштує 100 доларів на місяць, а пряма професійна підтримка коштує 1000 доларів на місяць. Корпорація Майкрософт не розголошує розмір плати за підтримку підприємств.

Оскільки Microsoft Azure складається з різноманітних пропозицій ресурсів і послуг, варіанти його використання надзвичайно різноманітні. Запуск віртуальних машин або контейнерів у хмарі є одним із найпопулярніших способів використання Microsoft Azure. Ці обчислювальні ресурси можуть розміщувати компоненти інфраструктури, такі як сервери системи доменних імен (DNS); служби Windows Server, такі як інформаційні служби Інтернету (IIS); мережеві послуги, такі як брандмауери; або програми сторонніх розробників.

Microsoft також підтримує використання сторонніх операційних систем, таких як Linux.

Azure також широко використовується як платформа для розміщення баз даних у хмарі. Microsoft пропонує безсерверні реляційні бази даних, такі як Azure SQL, і нереляційні бази даних, такі як NoSQL.

Крім того, платформа часто використовується для резервного копіювання та аварійного відновлення. Багато організацій використовують Azure для зберігання архівів, щоб задовольнити вимоги довгострокового збереження даних або аварійного відновлення (DR).

Microsoft розподіляє хмарні служби Azure майже на два десятки категорій. Кожна категорія може включати численні конкретні екземпляри або типи послуг. До найпопулярніших категорій послуг відносяться наступні:

Мережа. До цієї групи входять віртуальні мережі, виділені з'єднання та шлюзи, а також служби для керування трафіком і діагностики, балансування навантаження, DNS-хостинг і захист мережі від розподілених атак типу «відмова в обслуговуванні» (DDoS).

Зберігання. Ця категорія послуг надає масштабоване хмарне сховище для структурованих і неструктурованих даних. Він також підтримує проекти великих даних, постійне та архівне зберігання.

Інтернет. Ці служби підтримують розробку та розгортання веб-додатків. Вони також пропонують функції для пошуку, доставки вмісту, керування API, сповіщення та звітування.

Мобільні. Ці продукти допомагають розробникам створювати хмарні додатки для мобільних пристроїв, надаючи служби сповіщень, підтримку внутрішніх завдань, інструменти для створення інтерфейсів прикладних програм (API) і здатність поєднувати геопросторовий контекст з даними.

Аналітика. Ці служби забезпечують розподілену аналітику та зберігання, а також функції для аналітики в реальному часі, аналітики великих даних, озер даних, машинного навчання, бізнес-аналітики, потоків даних Інтернету речей (IoT) і сховищ даних.

Обчислювальні. Ці служби дозволяють користувачеві розгортати віртуальні машини, контейнери та пакетні завдання та керувати ними, а також підтримувати віддалений доступ до програм. Для обчислювальних ресурсів, створених у хмарі Azure, можна налаштувати загальнодоступні або приватні IP-адреси залежно від того, чи має ресурс бути доступним для зовнішнього світу.

Мережа доставки медіа та контенту (CDN). Ці послуги CDN включають потокове передавання на вимогу, захист цифрових прав, кодування, а також відтворення та індексування медіа.

IoT. Ці сервіси допомагають користувачам отримувати, контролювати й аналізувати дані IoT із датчиків та інших пристроїв. Послуги включають сповіщення, аналітику, моніторинг і підтримку кодування та виконання.

Ідентичність. Ці пропозиції гарантують, що лише авторизовані користувачі можуть отримати доступ до служб Azure, і допомагають захистити ключі шифрування та іншу конфіденційну інформацію в хмарі. Послуги включають підтримку Azure Active Directory і багатофакторну автентифікацію.

Інтеграція. Це сервіси резервного копіювання серверів, відновлення сайтів і підключення приватних і публічних хмар.

DevOps. Ця група надає інструменти для проектів і співпраці, такі як Azure DevOps — раніше Visual Studio Team Services — які полегшують процеси розробки програмного забезпечення DevOps. Він також пропонує функції для діагностики додатків, інтеграції інструментів DevOps і тестових лабораторій для тестування збірки та експериментів.

Безпека. Ці продукти надають можливості для виявлення та реагування на хмарні загрози безпеки, а також керування ключами шифрування та іншими конфіденційними активами.

Розвиток. Ці служби допомагають розробникам програм обмінюватися кодом, тестувати програми та відстежувати можливі проблеми. Azure підтримує ряд мов програмування додатків, включаючи JavaScript, Python, .NET і Node.js. Інструменти в цій категорії також включають підтримку Azure DevOps, комплекти розробки програмного забезпечення (SDK) і блокчейн.

DevOps. Ця група надає інструменти для проектів і співпраці, такі як Azure DevOps — раніше Visual Studio Team Services — які полегшують процеси розробки програмного забезпечення DevOps. Він також пропонує функції для діагностики додатків, інтеграції інструментів DevOps і тестових лабораторій для тестування збірки та експериментів.

ШІ та машинне навчання. Це широкий спектр сервісів, які розробник може використовувати для впровадження можливостей ШІ, машинного навчання та когнітивних обчислень у програми та набори даних.

Міграція. Цей набір інструментів допомагає організації оцінити витрати на міграцію робочого навантаження та виконати фактичну міграцію робочих навантажень із локальних центрів обробки даних у хмару Azure.

Бази даних. Ця категорія включає пропозиції баз даних як послуг (DBaaS) для SQL і NoSQL, а також інших екземплярів баз даних, таких як Azure Cosmos DB і Azure Database для PostgreSQL. Він також включає підтримку сховища даних SQL Azure, кешування та функції інтеграції та міграції гібридної бази даних. Azure SQL є флагманською службою баз даних платформи. Це реляційна база

даних, яка забезпечує функціональність SQL без необхідності розгортання сервера SQL.

Міграція. Цей набір інструментів допомагає організації оцінити витрати на міграцію робочого навантаження та виконати фактичну міграцію робочих навантажень із локальних центрів обробки даних у хмару Azure.

Блокчейн. Azure Blockchain Service дозволяє вам приєднатися до блокчейн-консорціуму або створити власний.

Змішана реальність. Ці служби розроблені, щоб допомогти розробникам створювати вміст для середовища Windows Mixed Reality.

Управління. Ці служби надають ряд інструментів резервного копіювання, відновлення, відповідності, автоматизації, планування та моніторингу, які можуть допомогти хмарному адміністратору керувати розгортанням Azure.

Intune. Microsoft Intune можна використовувати для реєстрації пристроїв користувачів, що дає змогу надсилати політики безпеки та мобільні програми на ці пристрої. Мобільні програми можна розгортати або для груп користувачів, або для колекції пристроїв. Intune також надає інструменти для відстеження того, які програми використовуються. Функція віддаленого стирання дозволяє безпечно видаляти дані організації з пристроїв без видалення мобільних додатків користувача під час процесу.[2]

Деякі організації використовують Azure для резервного копіювання даних і аварійного відновлення. Організації також можуть використовувати Azure як альтернативу власному сховищу в центрі обробки даних. Загальнодоступні хмари виявилися придатними для великої кількості короткострокових завдань, таких як аналіз даних. Організації можуть використовувати майже необмежену ємність хмарного сховища для зберігання великих наборів даних, виконання аналітичних завдань, а потім видаляти дані, коли вони стають застарілими або непридатними для використання — і все це без покупки або розгортання апаратного забезпечення в локальному центрі обробки даних. Цей тип корисних обчислень був ключовим фактором впровадження публічної хмари з моменту її створення.

Замість того, щоб інвестувати у внутрішні сервери та сховище, все більше організацій вирішують запускати деякі або всі свої бізнес-програми на Azure. Щоб забезпечити доступність, Microsoft має центри обробки даних Azure по всьому світу. Станом на січень 2020 року служби Microsoft Azure доступні в 140 країнах у 55 регіонах. На жаль, не всі послуги доступні у всіх регіонах. Таким чином, клієнти Azure повинні переконатися, що їхні робочі навантаження та зберігання даних відповідають будь-яким чинним вимогам відповідності або іншим юридичним вимогам.

Питання безпеки даних і дотримання нормативних вимог щодо конфіденційності є основними проблемами для абонентів хмари. Щоб вирішити ці проблеми, Microsoft створила онлайн-центр довіри, який надає детальну інформацію про ініціативи компанії щодо безпеки, конфіденційності та відповідності. Відповідно до Trust Center, Microsoft використовуватиме дані клієнтів лише для надання узгоджених послуг і ніколи не розголошуватиме дані клієнтів державним установам, якщо цього не вимагає закон.

У той же час Azure надає багато послуг, таких як автентифікація та керування доступом, брандмауер та інші служби безпеки, щоб допомогти користувачам створити безпечну інфраструктуру та вчасно контролювати вторгнення. Служби безпеки мають вирішальне значення для впровадження публічної хмари, допомагаючи клієнтам підтримувати конфіденційність конфіденційних даних і критично важливих робочих навантажень.

Як і інші хмарні постачальники, Azure використовує платіжну модель ціноутворення, яка в основному стягує плату за використання. Однак, якщо програма використовує більше ніж одну службу Azure, ціна може включати кілька стягнень. Одна послуга зазвичай використовує підмножину інших послуг, що збільшує загальну вартість запланованої служби. [2]

1.3 MySQL

MySQL — це система керування базами даних.

Це може бути будь-що: від простого списку покупок до галереї зображень або великого обсягу інформації в корпоративній мережі. Щоб додавати, отримувати та маніпулювати даними, що зберігаються в комп’ютерній базі даних, вам потрібна система керування базами даних, наприклад MySQL Server.

Реляційна база даних зберігає дані в окремих таблицях замість того, щоб поміщати всі дані в одне велике сховище. Структури бази даних організовані у фізичні файли, оптимізовані для швидкості. Логічна модель із такими об’єктами, як бази даних, таблиці, подання, рядки та стовпці, це гнучке середовище програмування. Також визначаються правила та вирази між різними таблицями, які регулюють зв’язки між різними полями даних, наприклад «один-до-одного», «багато-до-багатьох», унікальні, обов’язкові чи необов’язкові. База даних забезпечує дотримання цих правил, тому з добре спроектованою базою даних ваша програма ніколи не побачить неузгоджених, дублікатів, загублених, застарілих або відсутніх даних.

Частина SQL «MySQL» означає «Мова структурованих запитів». SQL є найпоширенішою стандартизованою мовою, яка використовується для доступу до баз даних. Залежно від середовища програмування ви можете отримати прямий доступ до SQL (наприклад, для створення звітів), вбудувати оператори SQL у код, написаний іншою мовою, або використовувати API для певної мови, який приховує синтаксис SQL.

Сервер MySQL може використовуватися іншими програмами, веб-серверами тощо. Може комфортно працювати на настільному комп'ютері або ноутбуці. не звертаючи ні найменшої уваги. Є можливість налаштувати параметри для використання всієї доступної пам'яті, потужності процесора та потужності введення/виведення. MySQL також дозволяє масштабуватися до кластерів мережевих машин.

MySQL Server спочатку був розроблений для обробки великих баз даних набагато швидше, ніж існуючі рішення, і успішно використовувався у виробничих середовищах з високим попитом протягом кількох років. Незважаючи на постійний розвиток, MySQL Server сьогодні пропонує багатий і корисний набір функцій. Його підключення, швидкість і безпека роблять MySQL Server добре придатним для доступу до баз даних в Інтернеті.

Програмне забезпечення бази даних MySQL — це багатопотокова система клієнт/сервер SQL Server, яка підтримує різноманітні серверні модулі, кілька різних клієнтських програм і бібліотек, інструменти керування та різноманітні інтерфейси прикладного програмування (API).

Ми також пропонуємо MySQL Server як вбудовану багатоцільову бібліотеку, яку ви можете зв'язати зі своїм додатком для меншого, швидшого та простішого в управлінні окремого продукту.

MySQL Server має практичний набір функцій, розроблений у тісній співпраці з нашими клієнтами. Швидше за все, ваше улюблене програмне забезпечення або мова підтримує сервер бази даних MySQL. [6]

1.4 Що таке Entity Framework

До .NET 3.5 часто використовували ADO.NET або Enterprise Data Access Block для написання коду для зберігання або отримання даних програми з основної бази даних. Раніше ми відкривали підключення до бази даних, створювали DataSet для отримання або надсилання даних до бази даних і перетворювали дані з DataSet в об'єкти .NET або навпаки за допомогою бізнес-правил у базі даних. Це був трудомісткий і схильний до помилок процес. Корпорація Майкрософт представила структуру під назвою Entity Framework для автоматизації всіх дій, пов'язаних із базою даних для вашої програми.

Entity Framework — це платформа ORM з відкритим кодом для додатків .NET, яка підтримується Microsoft. Це дозволяє розробникам працювати з даними за допомогою об'єктів класу домену, не турбуючись про таблиці бази даних і стовпці, які зберігають ці дані. Завдяки Entity Framework розробники можуть працювати на вищому рівні абстракції, коли мова заходить про дані, а також створювати й підтримувати керовані даними програми з меншим кодом, ніж традиційні програми. [3]

Entity Framework знаходиться між бізнес-сутностями (класами домену) і базою даних. Він зберігає дані, що зберігаються у властивостях бізнес-сутності, отримує дані з бази даних і автоматично перетворює їх на об'єкти бізнес-сутності.

Офіційне визначення: «Entity Framework — це об'єктно-реляційне відображення (O/RM), яке дозволяє розробникам .NET спілкуватися з базами даних за допомогою об'єктів .NET. Це усуває потребу у більшості коду доступу до даних, який зазвичай доводиться писати розробникам. [5]

До основних функцій Entity framwork належать:

Моделювання: EF (Entity Framework) створює EDM (Entity Data Model) на основі сутностей POCO (Plain Old CLR Object) із властивостями get/set різних типів даних. Він використовує цю модель, коли надсилає запити або зберігає дані сутності в базовій базі даних.

Збереження: коли викликається метод EF SaveChanges(), він виконує команди INSERT, UPDATE і DELETE на основі змін ваших об'єктів у базі даних. EF також надає асинхронний метод SaveChangesAsync().

Кросплатформенність: EF Core — це кросплатформна структура, яка може працювати на Windows, Linux і Mac.

Запити: EF дозволяє використовувати запити LINQ (C#/VB.NET) для отримання даних із основної бази даних. Постачальник бази даних перекладе ці запити LINQ на мову запитів, специфічну для бази даних (наприклад, SQL для реляційної бази даних). EF також дозволяє нам виконувати необроблені запити SQL безпосередньо до бази даних.

Відстеження змін: відстежує зміни екземплярів об'єктів (значення властивостей), які потрібно надіслати до бази даних EF.

Кешування: EF включає кешування першого рівня з коробки. Отже, повторні запити повертатимуть дані з кешу, а не в базу даних.

Міграція: EF надає набір команд міграції, які можна запускати в консолі NuGet Package Manager або CLI, щоб створити базову схему бази даних або керувати нею.

Транзакції: EF виконує автоматичне керування транзакціями під час запиту або зберігання даних. Він також надає параметри конфігурації керування транзакціями.

Вбудовані угоди: EF дотримується угод про шаблон програмування конфігурації та містить набір правил за замовчуванням, які автоматично налаштовують модель EF.

Паралелізм: EF за замовчуванням використовує оптимістичний паралелізм, щоб захистити зміни після того, як інший користувач видалив їх із бази даних.

Конфігурації: EF дозволяє нам налаштовувати модель EF за допомогою атрибутів анотації даних або API Fluent для перевизначення домовленостей.

Відстеження змін: відстежує зміни екземплярів об’єктів (значення властивостей), які потрібно надіслати до бази даних EF. [1]

Microsoft представила Entity Framework у 2008 році з .NET Framework 3.5. Відтоді було випущено багато версій Entity Framework. Наразі існує дві останні версії Entity Framework: EF 6 і EF Core. У таблиці нижче наведено важливі відмінності між EF 6 і EF Core. [1]

Розділ 2

Розгляд предметної області

2.1 Книга

Книга являє собою об’єкт, що описує безпосередньо книгу. Користувач має можливість ознайомитися з тією книгою, що його цікавить. Поля, що описують даний об’єкт, що відкривається користувачеві при виборі є коротка анотація до книги, автор, що написав книгу та короткий опис для зацікавлення та ознайомлення коротко потенційного читача. Також пошук по спеціальному полі яке відповідає за певні міні розділи (теги) які дають змогу орієнтуватися, на що очікувати в книзі.

Коментар являє собою об’єкт, що дає можливість залишити оцінку та враження чи можливо свої побажання, зауваження, питання до даної книги. Тобто до кожної книги ще є свої поля для коментарів та оцінки, що оприлюднюються іншими користувачами.

Автор книги це людина яка написала дану книгу. З описом книги також буде вказуватися і ім’я автора. Це допомагає користувачу зорієнтуватися і зрозуміти, що його чекати від даної книги, якщо він знайомий з автором, мати представлення про темп оповідання. Та дає можливість здійснювати пошук за автором.

Функція, що дозволить вносити зміни до книги. Це може бути як поле, що належить автору книги так і назва самої книги, опис також можна змінити. Тобто в загальному можна повністю редагувати дану книгу, натиснувши на відповідну кнопку на сайті після внесення змін сторінка перезапише дані в базу і користувач побачить їх вже зміненими. Редагування дозволяє вносити зміни по книзі з метою покращення опису книги та можливості постійно оновлювати інформацію.

Користувачі мають змогу залишати свою думку, ставлення до книги можливі побажання у вигляді коментаря. Це поле, що буде розміщено безпосередньо на сторінці вже даної книги. Тобто до кожної книги є змога залишити свій коментар. Вони відображаються для всіх користовачів сайту. Також з коментарем можна лишити оцінку до даної книги, яка підкреслить в загальному якість книги. На сайті всі результати сумуються та показується середній результат, тим самим формується рейтинг книги.

2.2 Пошукова система

На головний сторінці також розміщено пошукову стрічку, яка дає змогу шукати користувачеві і знаходити, чи перевіряти наявність саме тієї книги на яку він очікує з великого обсягу наявних. Пошук може здійснюватися по автору, або по короткому опису книги чи можливо по назві. Також реалізовано пошук по спеціальному полі яке відповідає за певні міні розділи (теги) що дасть змогу швидко шукати літературу яка необхідна певному користувачу.

Розділ 3

Хід виконання

3.1 Класи

Організовуємо класи book, comment, imageProcesing, uploadedFile для опису книжок, коментарів, фото та статусу фото, робота з файлом відповідно:

Описуємо поля класу книги:

```
public class Book
    {
        public int Id { get; set; }
        [Required]
        [StringLength(100)]
        public string Title { get; set; }
        [Required]
        [StringLength(100)]
        public string Author { get; set; }

        public string Image { get; set; }
        public string Description { get; set; }
        public virtual List<Comment>? Comments { get; set; }

        public string Tags { get; set; }
          }Відповідно продемонстровано інші класи:
```

Коментарі

```
public class Comment
    {
        public int Id { get; set; }
        [Required]
        public string Name { get; set; }
        public string Content { get; set; }
        public int Rate { get; set; }
        public int BookId { get; set; }
        public virtual Book Book { get; set; }
       }
```

Клас для роботи з зображеннями

```
public class ImageProcesing
    {
        public string Name { get; set; }
        public string ImageState { get; set; }
       }
```

Поле статусу потрібне безпосередньо для відслідкування статусу зображення чи завантажилося воно чи виникла помилка

Та робота з файлами

```
public class UploadedFile
    {
        public string FileName { get; set; }
        public byte[] FileContent { get; set; }
       }
```

Описано відповідно до кожного класу інтерфейси для використання даних так як нам необхідно

Для книги створено відповідні методи для роботи з нею такі як: отримання всіх книжок, додавання відповідної книги, отримання певної книги по id, редагування книги, та пошук по полям книги

```
public interface IBookService
    {
        Task<List<Book>> GetAllBooksAsync();
        Task<Book> AddBookAsync(Book book);
        Task<Book> GetBookByIdAsync(int Id);
        Task<Book> UpdateBookAsync(Book book);
        Task<List<Book>> SearchByString(string data);

          }
```

Також створюємо інтерфейс для коментарів який буде містити поля методів отримання всіх відгуків по книзі, та додавання коментаря до відповідної

```
public interface ICommentService
    {
        Task<List<Comment>> GetAllCommentsByBookIdAsync(int bookId);
        Task<Comment> AddCommentAsync(Comment comment);

       }
```

Далі інтерфейс файлів

```
public interface IFileService
    {
        Task<string> UploadFile(UploadedFile uploadedFile);
        string GetBlob(string fileName);
          }
```

Та відповідно зображень

```
public interface IImageService
    {
        Task<ImageProcesing> UploadImage(IBrowserFile file);
          }
```

Далі необхідно добавити безпосередньо сервіси які будуть наслідувати наші інтерфейси відповідно та описати створені в них методи бля роботи з ними

Почнемо з книги

```
namespace BlazorBookApp.Services
{
    public class BookService: IBookService
    {
        private ApplicationDbContext _context;

        public BookService(ApplicationDbContext applicationContext)
        {
            _context = applicationContext;
        }

        public async Task<List<Book>> GetAllBooksAsync()
        {
            return await _context.Books.ToListAsync();
        }

        public async Task<Book> AddBookAsync(Book book)
        {
            await _context.Books.AddAsync(book);
            await _context.SaveChangesAsync();
            return book;
```

```
        }

        public async Task<Book> GetBookByIdAsync(int Id)
        {
            var books = await _context.Books.Where(x => x.Id ==
Id).FirstOrDefaultAsync();
            return books;
        }

        public async Task<Book> UpdateBookAsync(Book book)
        {
            _context.Books.Update(book);
            await _context.SaveChangesAsync();
            return book;
        }

        public async Task<List<Book>> SearchByString(string data)
        {
            var result = await _context.Books.Where(x =>
x.Title.Contains(data)).ToListAsync();
            result.AddRange(await _context.Books.Where(x =>
x.Author.Contains(data)).ToListAsync());
            return result.DistinctBy(x => x.Id).ToList();
        }
    }
```

Далі реалізуємо відповідні методи для коментарів

```
namespace BlazorBookApp.Services
{
    public class CommentService : ICommentService
    {
        private ApplicationDbContext _context;

        public CommentService(ApplicationDbContext applicationContext)
        {
            _context = applicationContext;
        }

        public async Task<List<Comment>> GetAllCommentsByBookIdAsync(int bookId)
        {
            return await _context.Comments.Where(x => x.BookId ==
bookId).ToListAsync();
        }

        public async Task<Comment> AddCommentAsync(Comment comment)
        {
            await _context.Comments.AddAsync(comment);
            await _context.SaveChangesAsync();

            return comment;
        }


    }
   }
```

Тепер необхідно добавити сервіси для файлів та обробку відповідних файлів використовуючи сервіси Azure та обробку виключень, щоб передбачити вийнятки та передбачити певні помилки в роботі

```
 namespace BlazorBookApp.Services
{
    public class FileService : IFileService
    {
        private readonly string azureConnectionString;
        private IHostingEnvironment Environment;
```

```
        public FileService(IConfiguration configuration, IHostingEnvironment env)
        {
            azureConnectionString =
configuration.GetConnectionString("AzureConnectionString");
            Environment = env;
        }
        public async Task<string> UploadFile(UploadedFile uploadedFile)
        {
            string path = Path.Combine(Environment.WebRootPath, "Photos");
            if (!Directory.Exists(path))
            {
                Directory.CreateDirectory(path);
            }
            using (var ms = new MemoryStream(uploadedFile.FileContent))
            {
                using (var fs = new FileStream($"{path}/{uploadedFile.FileName}",
FileMode.Create))
                {
                    ms.WriteTo(fs);
                }
            }
            return uploadedFile.FileName;
        }

    public string GetBlob(string fileName)
        {
            string path = Path.Combine(Environment.WebRootPath, "Photos");
            if (!Directory.Exists(path))
            {
                Directory.CreateDirectory(path);
            }
            var file = File.ReadAllBytes($"{path}/{fileName}");
            var format = "image/png";
            var imageDataUrl = $"data:{format};base64,{Convert.ToBase64String(file)}";
            return imageDataUrl;

        }
       }
```

Даний код використовувався на раніх етапах створення програми без врахування сервісу Azure. Тобто всі дані зберігалися локально а не на сервісах які дозволяє використовувати безпосередньо Azure.

Наступний код вже реалізовує взаємодію з сервісом Azure

```
namespace BlazorBookApp.Services
{
    public class FileService : IFileService
    {
        private readonly string azureConnectionString;
        public FileService(IConfiguration configuration)
        {
            azureConnectionString =
configuration.GetConnectionString("AzureConnectionString");
        }
        public async Task<string> UploadFile(UploadedFile uploadedFile)
        {
            try
            {
                // Azure connection string and container name passed as an argument to
get the Blob reference of the container.
                var container = new BlobContainerClient(azureConnectionString,
"some");
                var createResponse = await container.CreateIfNotExistsAsync();
```

```
                // If container successfully created, then set public access type to
Blob.
                if (createResponse != null && createResponse.GetRawResponse().Status
== 201)
                    await
container.SetAccessPolicyAsync(Azure.Storage.Blobs.Models.PublicAccessType.Blob);

                // Method to create a new Blob client.
                var blob = container.GetBlobClient(uploadedFile.FileName);

                // If a blob with the same name exists, then we delete the Blob and
its snapshots.
                await
blob.DeleteIfExistsAsync(Azure.Storage.Blobs.Models.DeleteSnapshotsOption.IncludeSnaps
hots);

                // Create a file stream and use the UploadSync method to upload the
Blob.
                Stream stream = new MemoryStream(uploadedFile.FileContent);
                await blob.UploadAsync(stream);
                return uploadedFile.FileName;

            }
            catch (Exception e)
            {
                throw new Exception("Upload failed");
            }
            return uploadedFile.FileName;
        }

    public string GetBlob(string fileName)
        {
            var container = new BlobContainerClient(azureConnectionString, "some");
            var blob = container.GetBlobClient(fileName);

            return blob.Uri.AbsoluteUri;

           }
```

Та створено також сервіси для роботи з зображеннями відповідно. Де ми обробляємо фото відповідно до встановлених вимог, щоб не змінювати загального вигляду нашої галереї та сторінки в цілому

```
namespace BlazorBookApp.Services
{
    public class ImageService : IImageService
    {
        private readonly IFileService _fileService;
        public ImageService(IFileService fileService)
        {
            _fileService = fileService;
        }
        public async Task<ImageProcesing> UploadImage(IBrowserFile file)
        {
            var imageProcesing = new ImageProcesing();
            try
            {
                var fileUpload = await ResizeImage(file, 400, 600);
                imageProcesing.Name = await _fileService.UploadFile(fileUpload);
                imageProcesing.ImageState = $"{imageProcesing.Name} file(s) uploaded
on server";
                return imageProcesing;
            }
            catch (Exception ex)
            {
                imageProcesing.ImageState = ex.Message;
                return imageProcesing;
```

```
            }
        }

        private async Task<UploadedFile> ResizeImage(IBrowserFile file, int width, int
heigth)
        {
            MemoryStream stream = new MemoryStream();
            MemoryStream ms = new MemoryStream();
            var maxSize = 1024 * 1024 * 15;
            await file.OpenReadStream(maxSize).CopyToAsync(stream);
            Bitmap sourceImage = new Bitmap(stream);
            using (Bitmap objBitmap = new Bitmap(width, heigth))
            {
                objBitmap.SetResolution(sourceImage.HorizontalResolution,
sourceImage.VerticalResolution);
                using (Graphics objGraphics = Graphics.FromImage(objBitmap))
                {
                    // Set the graphic format for better result cropping
                    objGraphics.SmoothingMode =
System.Drawing.Drawing2D.SmoothingMode.AntiAlias;

                    objGraphics.InterpolationMode =
System.Drawing.Drawing2D.InterpolationMode.HighQualityBicubic;
                    objGraphics.DrawImage(sourceImage, 0, 0, width, heigth);

                    objBitmap.Save(ms, System.Drawing.Imaging.ImageFormat.Jpeg);
                }
            }
            UploadedFile uploadedFile = new UploadedFile();
            uploadedFile.FileName = file.Name.Replace(" ", "-");
            uploadedFile.FileContent = ms.ToArray();

            return uploadedFile;
        }
    }
   }
```

3.2NavMenu

Добавляємо у згенерований файл файл NavMenu.razor наші поля, що будуть відображатися та описуємо їх для роботи з ними при натисканні. Також необхідно добавити відображення цієї панельки лише коли ми авторизовані:

```
<div class="top-row ps-3 navbar navbar-dark">
    <div class="container-fluid">
        <a class="navbar-brand" href="">BlazorBookServer</a>
        <button title="Navigation menu" class="navbar-toggler"
@onclick="ToggleNavMenu">
            <span class="navbar-toggler-icon"></span>
        </button>
    </div>
</div>

<div class="@NavMenuCssClass" @onclick="ToggleNavMenu">
    <nav class="flex-column">
        <div class="nav-item px-3">
            <NavLink class="nav-link" href="" Match="NavLinkMatch.All">
                <span class="oi oi-home" aria-hidden="true"></span> Home
            </NavLink>
        </div>
        <AuthorizeView>
        <Authorize>
        <div class="nav-item px-3">
            <NavLink class="nav-link" href="addbook">
                <span class="oi oi-plus" aria-hidden="true"></span> Add Book
            </NavLink>
        </div>
```

```
        </Authorize>
        </AuthorizeView>
    </nav>
</div>


@code {
    private bool collapseNavMenu = true;

    private string? NavMenuCssClass => collapseNavMenu ? "collapse" : null;

    private void ToggleNavMenu()
    {
        collapseNavMenu = !collapseNavMenu;
    }
}
```

3.3 Наповнення сайту вмістом та додавання відповідних інструкцій для роботи

Організовано роботу по сторінці, а саме коли ми авторизовані буде відображено всі книги на головній сторінці та описано роботу пошуку книг за раніше описаним методом searchbook. Та відповідне оновлення сторінки після здійсненого пошуку. І тут відображення відповідних повідомлень якщо користувач ще не залогінився.

```
@page "/"

@using BlazorBookApp.Data;
@using BlazorBookApp.Services.Interfaces;
@inject IBookService bookService
@inject IFileService fileService;
@inject NavigationManager navigationManager

<PageTitle>Index</PageTitle>
<AuthorizeView>
    <Authorized>
            <h1>Blazing books</h1>
            <div class="searchContainer">
                <input type="search" placeholder='Search...' class='search' @bind-
value="SearchText">
                <button class="custom-btn btn-6"
@onclick="SearchBooks">Search</button>
            </div>
            <ul class="book-cards">
                @if (books != null)
                {
                    @foreach (var book in books)
                    {
                        <li @onclick='@(() =>
@navigationManager.NavigateTo($"/book/{@book.Id}"))' style="background-image:
url('@fileService.GetBlob(book.Image)')">
                            <div class="book-info">
                                <span class="title">@book.Title</span>
                                @book.Author
                            </div>
                        </li>
                    }
                }
            </ul>
    </Authorized>
    <NotAuthorized>
        <h1>Hello, world!</h1>

        Please log in to see content!
    </NotAuthorized>
```

```
</AuthorizeView>

@code {
    [CascadingParameter]
    public string SearchText { get; set; } = String.Empty;
    List<Book> books = new List<Book>();
    string imageDataUrl = String.Empty;


    protected override async Task OnInitializedAsync()
    {
        books = await bookService.GetAllBooksAsync();
    }

    private async Task SearchBooks()
    {
        books = await bookService.SearchByString(SearchText);
        StateHasChanged();
    }
}
```

Далі реалізовано роботу з книгою при натискані на неї, відображення її рейтингу відносно коментарів, та апдейт книги для оновлення даних книги та додавання самого коментаря та пошук середнього рейтингу. Html сторінка добавлена відповідно до потреб.

`@page "/book/{Id:int}"` **в даному полі реалізовано передачу книги на сторінку по id**

```
@page "/book/{Id:int}"
@using BlazorBookApp.Data;
@using BlazorBookApp.Services.Interfaces;
@using Microsoft.AspNetCore.Authorization;
@attribute [Authorize]
@inject IBookService bookService
@inject IFileService fileService;
@inject ICommentService commentService
@inject NavigationManager navigationManager

@if(showBook == null)
{
    <p>Loading....</p>
}
else
{

<div class="container">
    <div class="column">
            <img src="@fileService.GetBlob(showBook.Image)" class="show-big-image" />
    </div>
    <div class="column">
         <h1><b>Title Name:</b> </h1>
        <h1>@showBook.Title</h1>
        <h2><b>Author:</b></h2>
         <h2> @showBook.Author</h2>
        <h5><b>Description</b></h5>
        <p>@showBook.Description</p>
            <h5><b>Tags</b></h5>
            <p>@showBook.Tags</p>
        <h5><b>Rate of the book</b></h5>
        <p>@GetOveralRate()</p>
            <button class="custom-btn btn-6" type="button" @onclick='@(() =>
@navigationManager.NavigateTo($"/UpdateBook/{@showBook.Id}"))'>Edit</button>

    </div>
    </div>
```

```
    <div>
        <div>
        @foreach (var comment in comments)
        {
            <h4>
                <b>Name:</b> @comment.Name
                <b>Rate:</b> @comment.Rate
            </h4>

            <p>
                @comment.Content
            </p>
        }
    </div>
        <EditForm model="@newComment" OnValidSubmit="@AddCommentAsync">
            <DataAnnotationsValidator />
            <div class="row bg_1">
                <div class="col-2">
                    <label for="rate">Rate</label>
                    <InputNumber id="rate" class="effect-1" @bind-
Value="@newComment.Rate" />
                </div>
                <div class="col-2">
                    <label for="Name">Name</label>
                    <InputText id="Name" class="effect-1" placeholder="Name" @bind-
Value="@newComment.Name" />
                    <span class="focus-border"></span>
                </div>
                <div class="col-1">
                    <label for="content">Content</label>
                    <br>
                    <InputTextArea id="content" class="effect-1" @bind-
Value="@newComment.Content" />
                </div>
            </div>
            <ValidationSummary />
            <button class="custom-btn btn-6" type="submit">Submit</button>
        </EditForm>
        @if (showMessage)
        {
            <div>
                @commentState
            </div>
        }
        </div>

}



@code {
    private Book showBook{ get; set; }
    [Parameter]
    public int Id { get; set; } = 0;
    private Comment newComment { get; set; } = new Comment();
    private string commentState = string.Empty;
    bool showMessage = false;

    private List<Comment> comments{ get; set; }

    protected override async Task OnInitializedAsync()
    {
        showBook = await bookService.GetBookByIdAsync(Id);
        comments = await commentService.GetAllCommentsByBookIdAsync(Id);
    }

    private async Task AddCommentAsync()
```

```
    {
        newComment.BookId = Id;
        await commentService.AddCommentAsync(newComment);
        comments.Add(newComment);
        showMessage = true;

        commentState = "Comment Added";
        newComment = new Comment();
        StateHasChanged();
        await Task.Delay(5000);
        showMessage = false;
    }

    private double GetOveralRate()
    {
        return comments.Sum(x => x.Rate) / (comments.Count() != 0 ? comments.Count() :
1);
    }
}
```

Реалізовуємо відображення та оновлення книги на сайті. Та відповідні повідомлення при оновленнях на сайтах. Додано показ зміни файлу який ми будемо змінювати. Файлт приходять і ми їх міняємо

```
@page "/updateBook/{id:int}"
@using BlazorBookApp.Data;
@using BlazorBookApp.Services.Interfaces;
@using Microsoft.AspNetCore.Authorization;
@attribute [Authorize]
@inject IBookService bookService
@inject IImageService imageService
@inject IFileService fileService;

<div class="container">
    <div class="column">
        <img src="@fileService.GetBlob(newBook.Image)" class="show-big-image" />
    </div>
    <div class="column">
        <EditForm Model="@newBook" OnValidSubmit="@UpdateBookToDb">
            <DataAnnotationsValidator />
            <div class="row bg_1">
                <div class="col-2">
                    <label for="title">Title</label>
                    <br />
                    <InputText id="title" class="effect-1" placeholder="Title" @bind-
Value="newBook.Title" />
                    <span class="focus-border"></span>
                </div>
                <div class="col-2">
                    <label for="author">Author</label>
                    <br />
                    <InputText id="author" class="effect-1" placeholder="Author"
@bind-Value="newBook.Author" />
                    <span class="focus-border"></span>
                </div>
                <div class="col-1">
                    <label for="descript">Description</label>
                    <br />
                    <InputTextArea id="descript" class="textarea-style"
placeholder="Enter description" @bind-Value="newBook.Description" />
                </div>
                <div class="col-1">
                    <label for="tags">Tags</label>
                    <br />
```

```
                    <InputTextArea id="tags" class="textarea-style" placeholder="Enter
tags" @bind-Value="newBook.Tags" />
                </div>
            </div>
            <ValidationSummary />
            <button class="custom-btn btn-6" type="submit">Submit</button>

        </EditForm>
        <div>
            <span>@result</span>
        </div>
    </div>
</div>

@code {
    private Book newBook = new Book();
    private Book updateBook = new Book();
    private string result = string.Empty;
    string imageDataUrl = string.Empty;
    string fileState = String.Empty;
    IBrowserFile file;


    [Parameter]
    public int Id{ get; set; }

    protected override async Task OnInitializedAsync()
    {
        newBook = await bookService.GetBookByIdAsync(Id);
        updateBook = newBook;
    }

    private async Task UpdateBookToDb()
    {
        newBook = await bookService.UpdateBookAsync(newBook);
        result = $"Updated Title {newBook.Title}";
    }

    private async void OnInputFileChange(InputFileChangeEventArgs e)
    {
        file = e.GetMultipleFiles().First();
        var format = "image/png";
        var resize = await file.RequestImageFileAsync(format, 180, 320);
        var buffer = new byte[resize.Size];
        await resize.OpenReadStream().ReadAsync(buffer);

        imageDataUrl = $"data:{format};base64,{Convert.ToBase64String(buffer)}";
        this.StateHasChanged();
    }

}
```

Додано роботу для додавання книги, загрузка файлу

```
 @page "/addbook"
@using Azure.Storage.Blobs;
@using BlazorBookApp.Data;
@using BlazorBookApp.Services.Interfaces;
@using Microsoft.AspNetCore.Authorization;
@attribute [Authorize]
@inject IBookService bookService
@inject IImageService imageService;
@inject IFileService fileService;


<div class="container">
    <div class="column">
<form>
    @if (imageDataUrl != string.Empty)
```

```
    {
        <img src="@imageDataUrl" class="show-image"/>
    }
        <br/>
        <InputFile OnChange="OnInputFileChange" multiple />
        <br />
    <br/><br/>
            <button class="custom-btn btn-6" type="reset" @onclick="(() =>
imageDataUrl = string.Empty)">Reset</button>
    <br/>
    <label>@fileState</label>
</form>
</div>
    <div class="column">
<EditForm Model="@newBook" OnValidSubmit="@AddBookToDb">
<DataAnnotationsValidator/>
            <div class="row bg_1">
                <div class="col-2">
                    <label for="title">Title</label>
                    <br />
                    <InputText id="title" class="effect-1" placeholder="Title" @bind-
Value="newBook.Title" />
                    <span class="focus-border"></span>
                </div>
                <div class="col-2">
                    <label for="author">Author</label>
                    <br />
                    <InputText id="author" class="effect-1" placeholder="Author"
@bind-Value="newBook.Author" />
                    <span class="focus-border"></span>
                </div>
                <div class="col-1">
                    <label for="descript">Description</label>
                    <br />
                    <InputTextArea id="descript" class="textarea-style"
placeholder="Enter description" @bind-Value="newBook.Description" />
                </div>
                <div class="col-1">
                    <label for="tags">Tags</label>
                    <br />
                    <InputTextArea id="tags" class="textarea-style" placeholder="Enter
tags" @bind-Value="newBook.Tags" />
                </div>
                <ValidationSummary />
               <button class="custom-btn btn-6" type="submit">Submit</button>
            </div>
</EditForm>
<div>
    <span>@result</span>
</div>
</div>
</div>


@code {
    private Book newBook = new Book();
    IBrowserFile file;

    string result = String.Empty;
    string imageDataUrl = String.Empty;
    string fileState = String.Empty;


    private async void OnInputFileChange(InputFileChangeEventArgs e)
    {
        file = e.GetMultipleFiles().First();
        var format = "image/png";
        var buffer = new byte[file.Size];
```

```
        var maxSize = 1024 * 1024 * 15;
        var fileContent = new StreamContent(file.OpenReadStream(maxSize));
        buffer = await fileContent.ReadAsByteArrayAsync();

        imageDataUrl = $"data:{format};base64,{Convert.ToBase64String(buffer)}";
        this.StateHasChanged();
    }


    private async Task AddBookToDb()
    {
        var fileDate = await imageService.UploadImage(file);
        newBook.Image = fileDate.Name;
        fileState = fileDate.ImageState;
        await bookService.AddBookAsync(newBook);

        result = $"Added Title {newBook.Title}";
        newBook = new Book();
    }
}
```

Далі додано раніше створені сервіси безпосередньо в програму перед тим як прописали aplicationContext відповідно.

```
var connectionString = builder.Configuration.GetConnectionString("DefaultConnection");
builder.Services.AddDbContext<ApplicationDbContext>(options =>
    options.UseSqlServer(connectionString));
builder.Services.AddDatabaseDeveloperPageExceptionFilter();
builder.Services.AddDefaultIdentity<IdentityUser>(options =>
options.SignIn.RequireConfirmedAccount = true)
    .AddEntityFrameworkStores<ApplicationDbContext>();
builder.Services.AddRazorPages();
builder.Services.AddServerSideBlazor();
builder.Services.AddScoped<AuthenticationStateProvider,
RevalidatingIdentityAuthenticationStateProvider<IdentityUser>>();

builder.Services.AddTransient<IBookService, BookService>();
builder.Services.AddTransient<IImageService, ImageService>();
builder.Services.AddTransient<IFileService, FileService>();
builder.Services.AddTransient<ICommentService, CommentService>();

      var app = builder.Build();
```

Розділ 4. Результати роботи

4.1 Авторизація

Для користування сервісом необхідно авторизуватися на сайті використавши свій логін та пароль (рис 4.1)

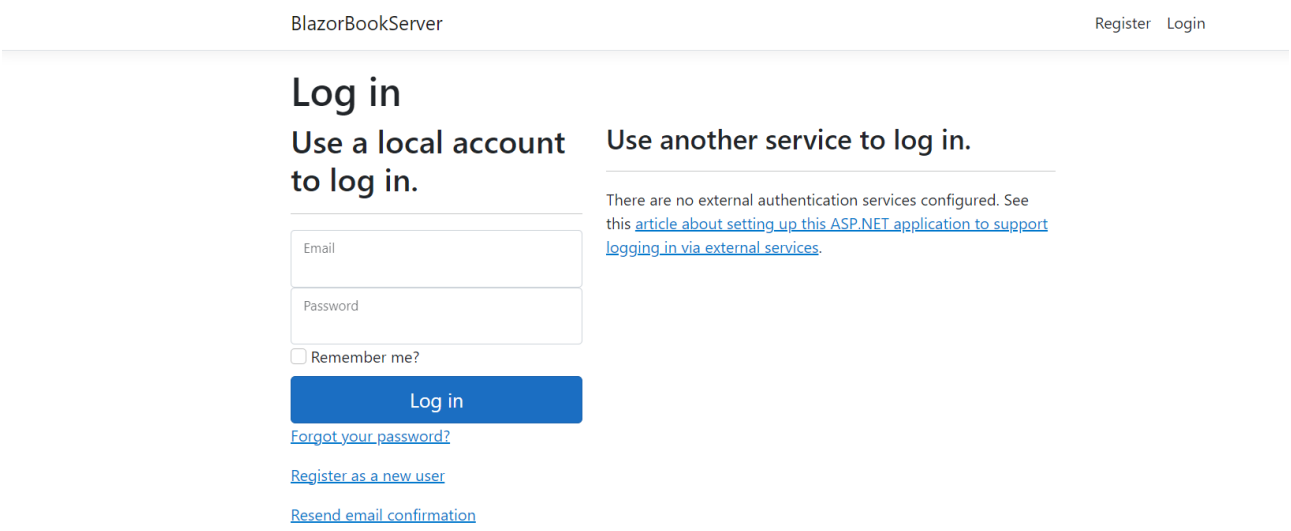

Рис. 4.1

Якщо користувач ще не зареєстрований то необхідно зареєструватися на сайті (рис. 4.2)

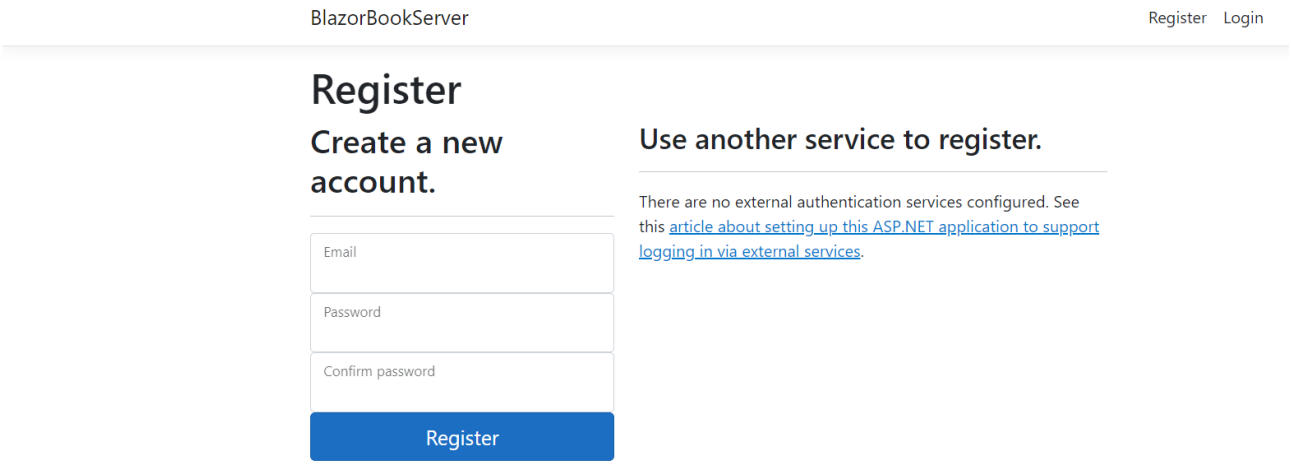

Рис. 4.2 Реєстрація на сайті

Якщо при реєстрації буде вказано невірний пароль, вискочить попередження про це. (рис. 4.3)

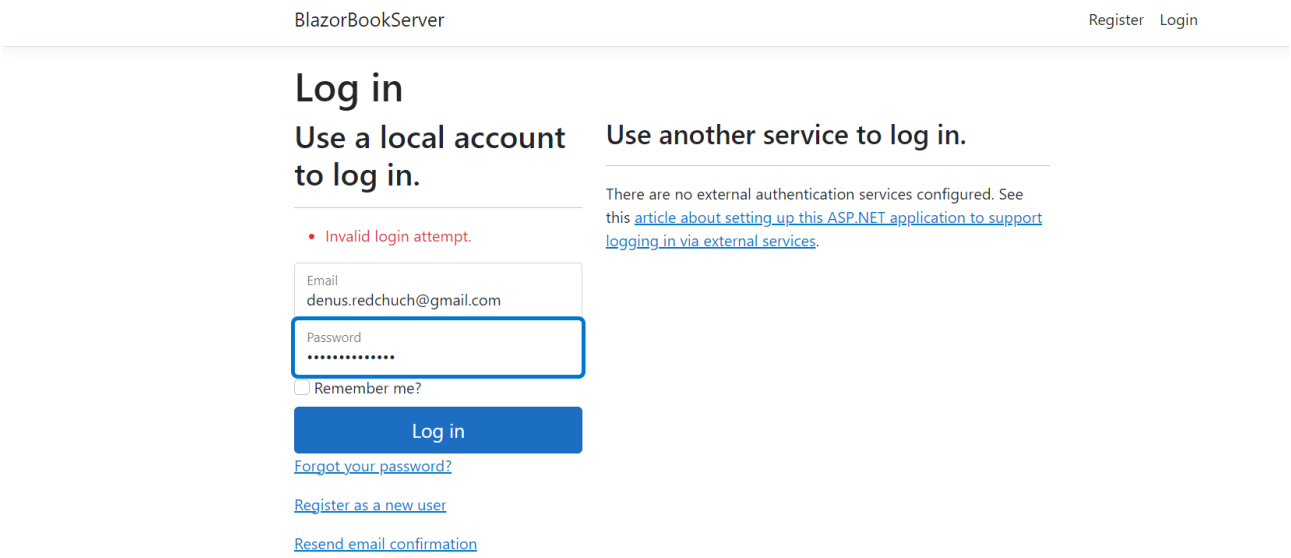

Рис. 4.3

При вірному написані паролю здійсниться вхід і відобразить початкова сторінка.

4.2Робота пошуку

Перевіримо роботу пошукової стрічки при пошуку за назвою книги (рис. 4.4)

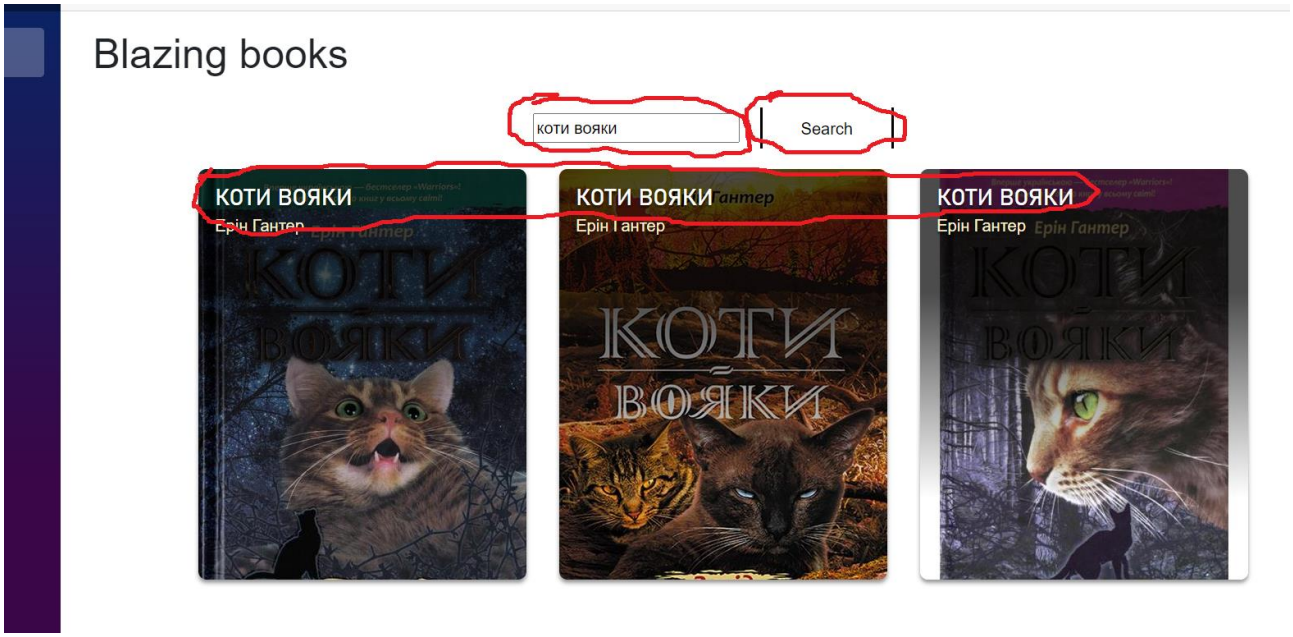

Рис. 4.4

Перевіримо роботу пошукової стрічки при пошуку за автором книги (рис 4.5)

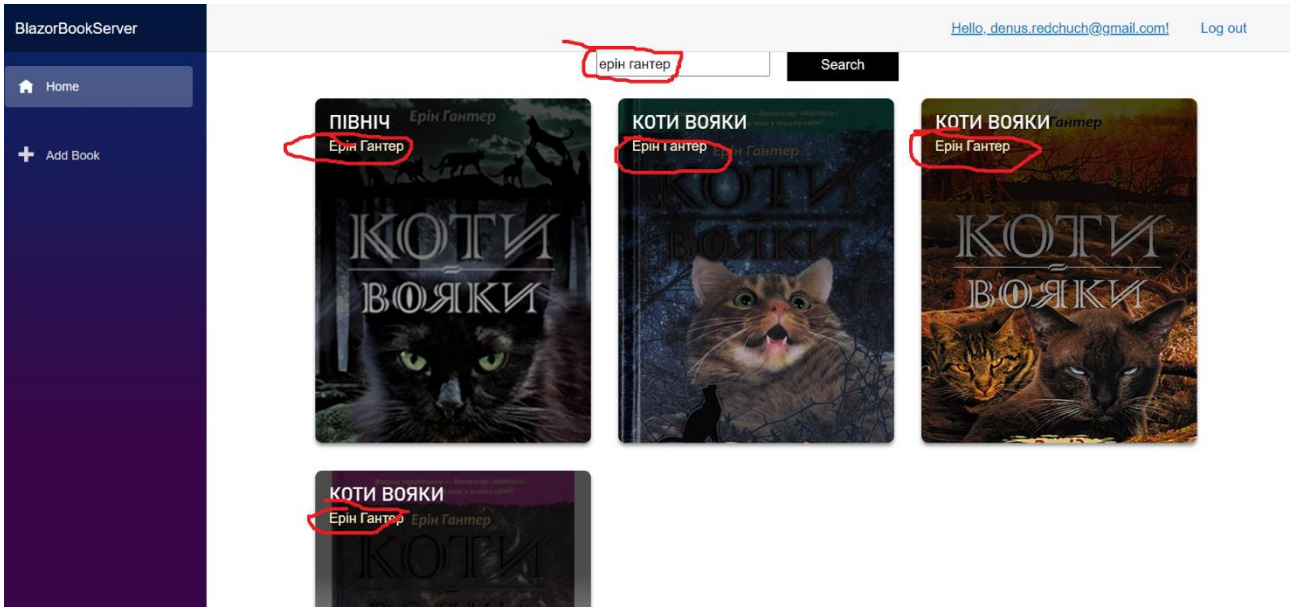

рис. 4.5

Далі перевірено роботу за тегами. Наприклад добавимо книгу жахів та добавимо їй тег жахи. (рис. 4.6) Результат пошуку:

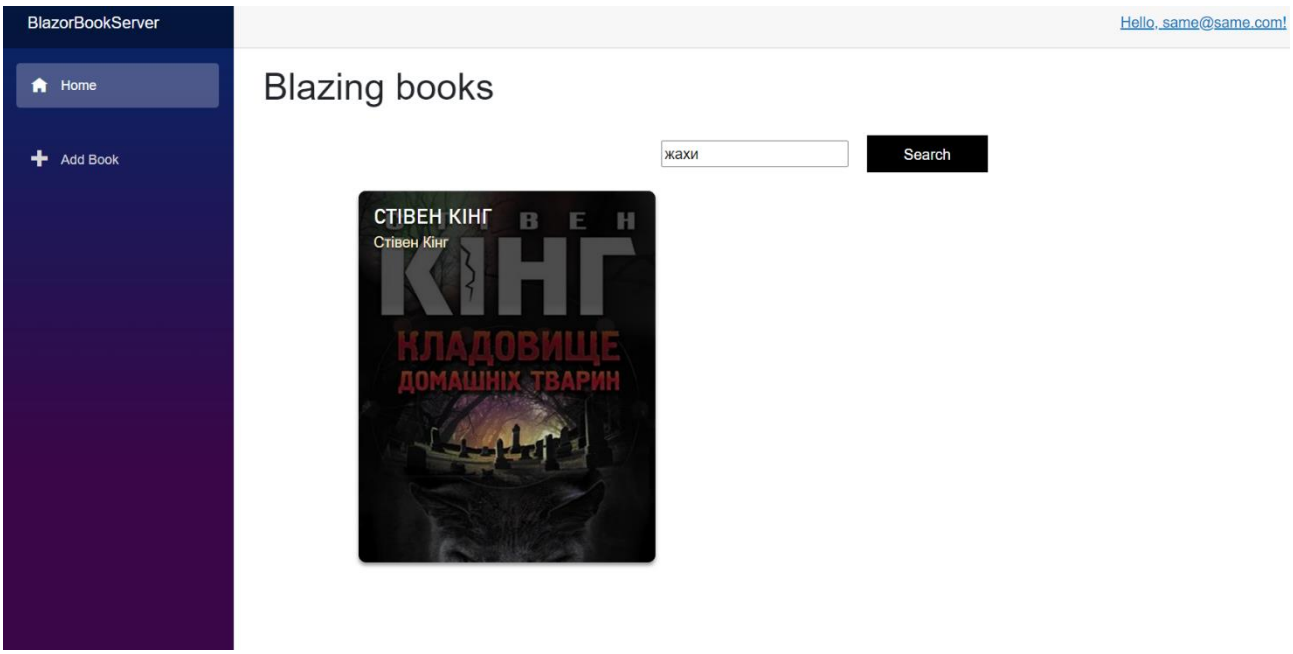

Рис. 4.6

Все працює як і має працювати.

4.3Добавлення коментарів та книги

Добавимо коментар до певної книги з нашої галереї, пишемо імя того хто залишає коментар (рис. 4.7):

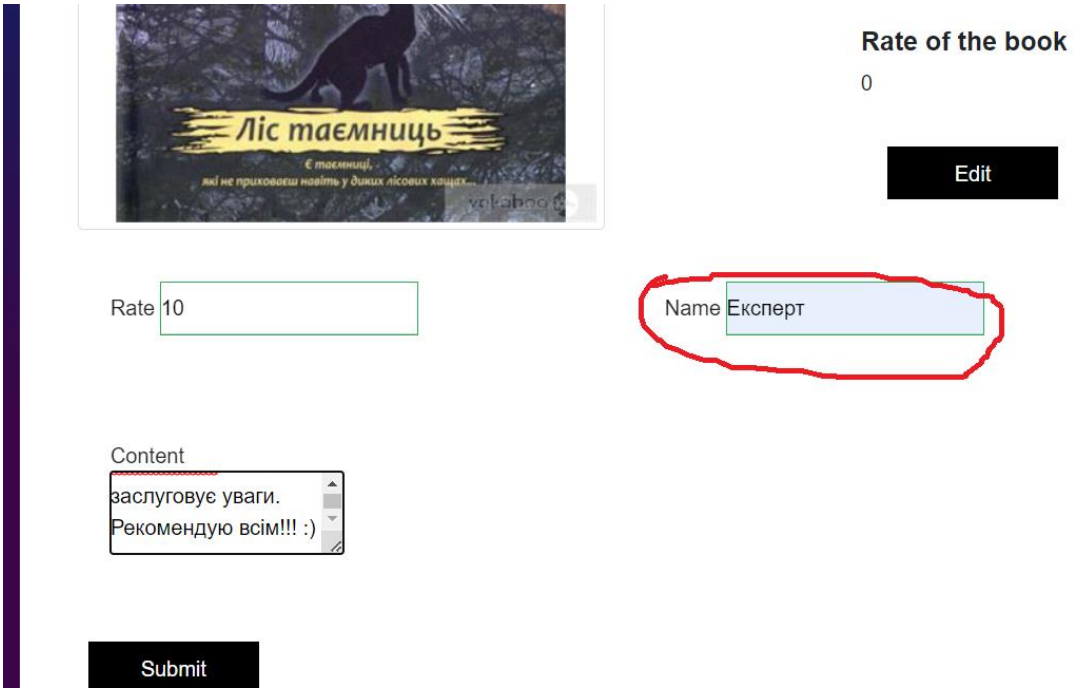

Рис. 4.7

Далі ставимо рейтинг відповідно до книги і її оцінки даним користувачем (рис. 4.8):

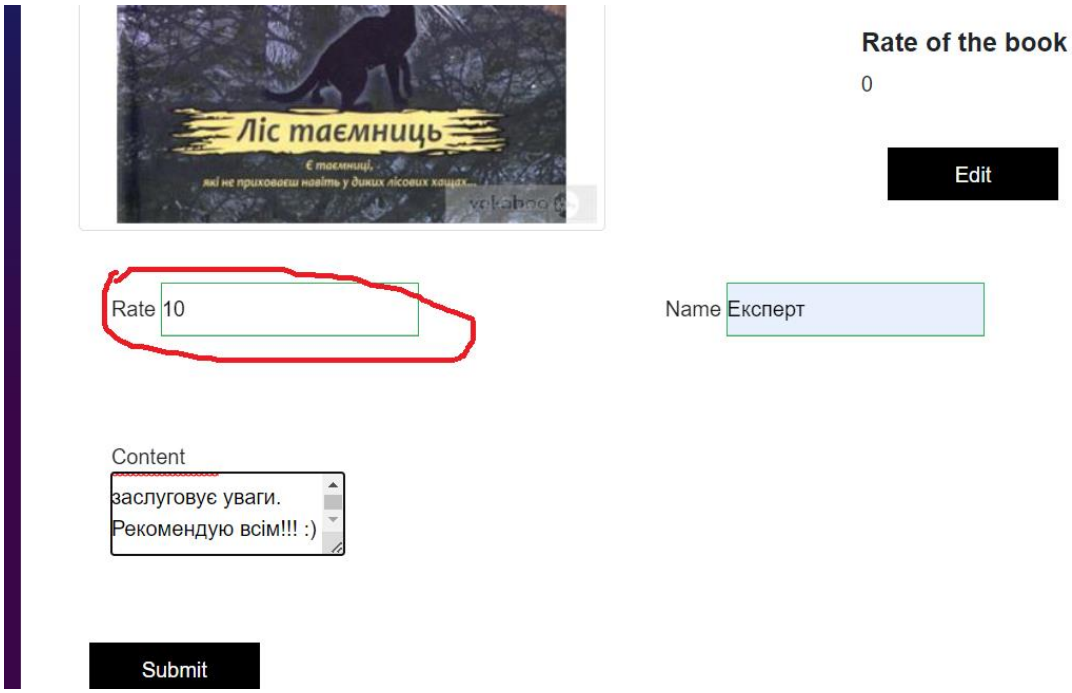

Рис. 4.8

Також сам коментар (рис. 4.9):

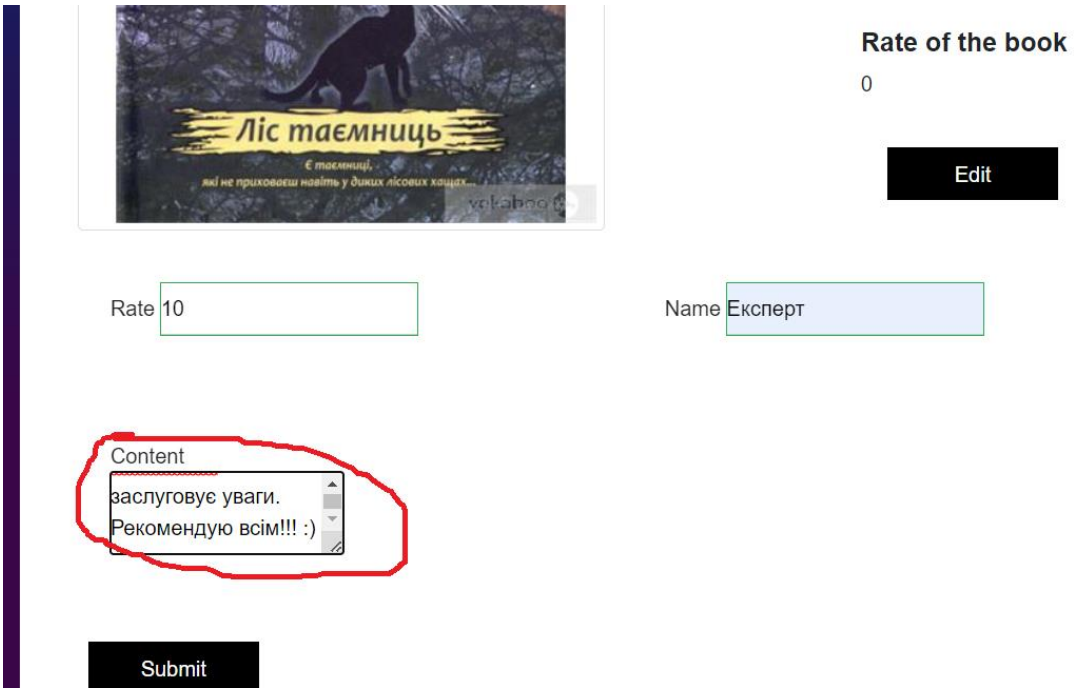

Рис. 4.9

І добавляємо це все до книги (рис. 4.10):

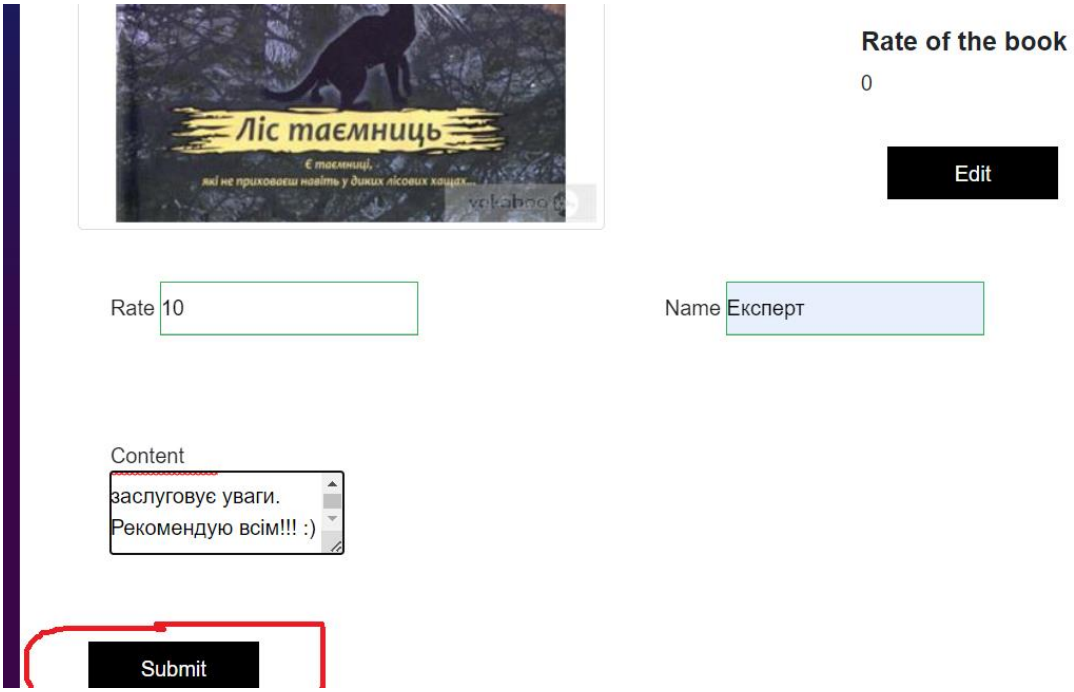

рис. 4.10

4.3 Добавлення книги

Натискаємо на раніше добавлену вкладку addbook та бачимо всі необхідні поля для заповнення. Заповнюємо їх та добавляємо книгу (рис. 4.11)

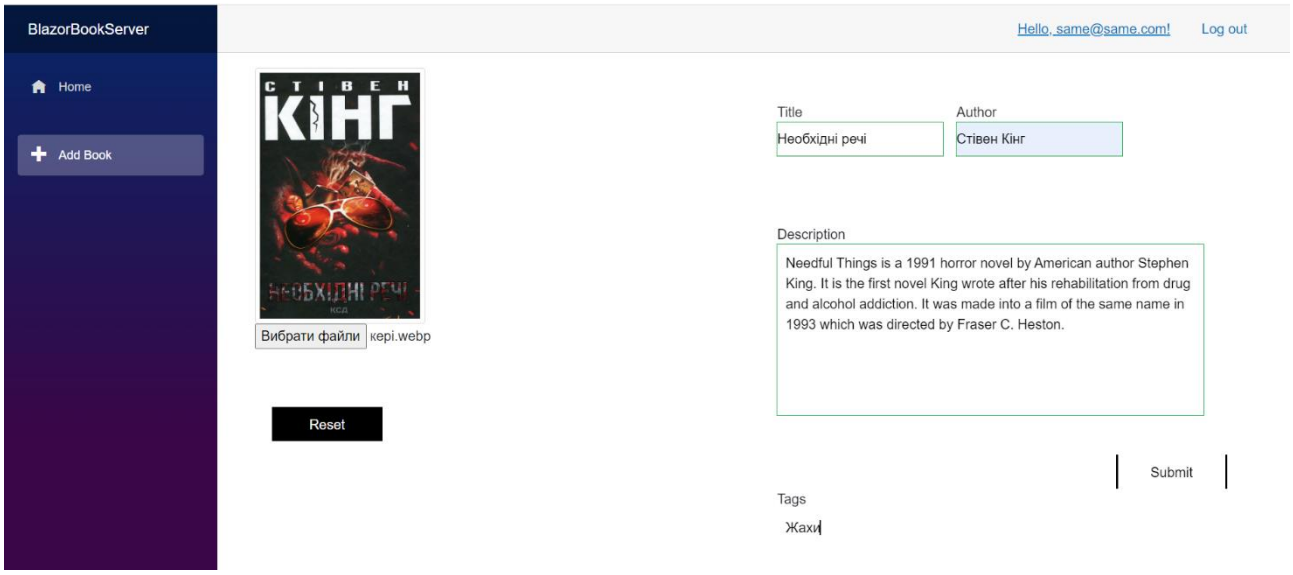

Рис. 4.11

Все працює чудово, отже реалізовано весь запланований функціонал книжкової галереї.

Висновки: Реалізовано задум бакалаврської роботи та розроблено функціональний веб-додаток інтерактивна книжкова галерея. Він дозволяє здійснювати огляд, додавання, редагування, оцінювання, здійснювати швидкий пошук необхідної літератури за тегами та обговорення книг. Використано при цьому веб-фреймворк Blazor та використано хмарний серверний сервіс Azure для серверного користування сторінкою. Також для роботи та збереження даних використано базу даних sql Server. Сайт можна використати для зберігання та швидкого пошуку літератури.

Реалізовано основний задум по збереженню різного роду літератури та запису по тегам кожної книги, що дає змогу користувачам робити записи відносно книг для подальшого користування а саме пошуку та накопичення книг.

Також задум щодо пошуку. Пошук книг відбувається по тегам, назві, автору, що допомагає значно скоротити пошуки різноманітної літератури по ключовим темам, словам, людях і тд.

Все працює належним чином відповідно до задуму.

Список використаних джерел

[1] - https://www.entityframeworktutorial.net/what-is-entityframework.aspx#:~:text=Entity%20Framework%20is%20an%20open,where%20this%20data%20is%20stored.

[2] - https://www.techtarget.com/searchcloudcomputing/definition/Windows-Azure

[3] - https://developer.mozilla.org/ru/docs/Web/Performance/Lazy_loading

[4] - https://docs.microsoft.com/en-us/dotnet/core/extensions/dependency-injection

[5] - https://docs.microsoft.com/en-us/ef/

[6] - https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver15