КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра теорії та технології програмування

Кваліфікаційна робота

на здобуття ступеня бакалавра
за спеціальністю 122 Комп'ютерні науки
на тему:

ЕЛЕКТРОННА БІБЛІОТЕКА

Виконав студент 4-го курсу
Михайло БУБКА

(підпис)

Науковий керівник:
доцент, кандидат фіз.-мат. наук
Людмила ОМЕЛЬЧУК

(підпис)

Засвідчую, що в цій курсовій роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент

(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри теорії та технологій програмування
«05» Червня 2023 р.,
протокол № 18
Завідувач кафедри
Микола НІКІТЧЕНКО

Київ – 2023

РЕФЕРАТ

Обсяг роботи 58 сторінок, з них 52 сторінки основного тексту, 27 ілюстрацій, 11 таблиць, 23 джерела посилань, 3 додатки.

БАЗА ДАНИХ, ВЕБЗАСТОСУНОК, ЗБЕРЕЖЕННЯ ІНФОРМАЦІЇ, ІНФОРМАЦІЙНА СИСТЕМА, СИСТЕМА УПРАВЛІННЯ БАЗАМИ ДАНИХ, ТЕХНОЛОГІЇ ПРОГРАМУВАННЯ, ЕЛЕКТРОННА БІБЛІОТЕКА.

Об’єктом роботи є процес доступу та створення електронних книг. Предметом роботи є вебзастосунок для автоматизації процесу доступу та створення електронних книг за допомогою технологій ASP.NET, EF Core та React.

Метою роботи є проєктування, розробка та апробація вебсайту для збереження, створення та поширення користувацької інформації у вигляді книг. Також на меті ставиться реалізація користувацької та адміністративної підсистем з різним функціоналом.

Методи розроблення: комп’ютерне моделювання, розробка програмного продукту. Інструменти розроблення: операційна система Windows 10, безкоштовне середовище розробки Visual Studio, мови програмування C# та TypeScript, універсальна мова моделювання UML, мова тегів HTML, мова стилів CSS, мова SQL, ASP.NET Core, технологія доступу до даних EF Core, система управління базами даних Microsoft SQL Server, бібліотека для створення користувацького інтерфейсу React, бібліотеки: AutoMapper, FluentValidation, JWT token, Bulma.

Результати роботи: виконано проєктування бази даних та вебзастосунку з використанням мови UML. Розроблено серверну та клієнтську частини застосунку за допомогою ASP.NET та React відповідно. З допомогою підходу code-first створено набір міграцій та базу даних в СУБД MS SQL Server. В застосунок доданий функціонал авторизації та автентифікації за допомогою Json Web Token. Створений користувацький інтерфейс для сайту з використанням бібліотеки Bulma. Застосунок протестований з допомогою Postman.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

API – Application Programming Interface, набір правил і протоколів, які дозволяють застосункам взаємодіяти один з одним;

CRUD – Create, Read, Update, Delete, 4 основні функції управління даними «створення, читання, оновлення і видалення»;

CSS – Cascading Style Sheets, спеціальна мова стилю сторінок для опису їх зовнішнього вигляду;

DI – Dependency Injection, шаблон проєктування програмного забезпечення;

DOM – Document Object Model, програмний інтерфейс, що представляє HTML сторінку у вигляді дерева елементів, дозволяючи взаємодію і модифікацію вмісту;

DTO – Data Transfer Object, шаблон проєктування, який є структурованим контейнером для передачі даних між компонентами програми;

EF Core – Entity Framework Core, фреймворк ORM, який допомагає спростити взаємодію з базами даних за допомогою інтуїтивно зрозумілої об'єктно-орієнтованої парадигми.

HTTP – Hypertext Transfer Protocol, мережевий протокол передачі даних;

ID – Identifier, унікальний ідентифікатор об’єкта;

IDE – Integrated Design Environment, комплексний застосунок, який поєднує функції редагування коду, налагодження та управління проєктами в єдиному інтерфейсі;

JS – JavaScript, універсальна мова створення скриптів, що використовується для створення динамічного та інтерактивного вебконтенту;

JSON – JavaScript Object Notation, текстовий формат обміну даними, який використовує зрозумілий для людини синтаксис для представлення структурованих даних;

JSX – JavaScript XML, розширення синтаксису мови JavaScript, що дозволяє вбудовувати HTML-подібний код у JavaScript.

JWT – JSON Web Token, компактний засіб представлення даних для безпечної передачі через мережу;

LINQ – Language-Integrated Query, розширення мови запитів для доступу і маніпулювання даними з різних джерел, включаючи бази даних та колекції;

ORM – Object-Relational Mapping, технологія, яка дозволяє відображати та взаємодіяти з об'єктами реляційних баз даних за допомогою об'єктно-орієнтованих мов програмування;

REST – Representational State Transfer, архітектурний стиль передачі даних через мережу;

SQL – Structured Query Language, мова структурованих запитів, яка використовується для управління та маніпулювання реляційними базами даних й даними в них;

UI – User Interface, користувацький інтерфейс застосунку;

UML – Unified Modelling Language, це мова моделювання, яка використовується для проєктування, візуалізації, побудови та документування структури та поведінки застосунків;

URL – Uniform Resource Locator, це рядок символів, який містить адресу або місцезнаходження вебресурсу в інтернеті;

VS Code – Visual Studio Code, це редактор вихідного коду з широким набором розширень та функцій, що легко налаштовуються;

БД – База даних, структуроване сховище даних, що дозволяє зберігати, керувати та отримувати доступ до інформації, збереженої у вигляді таблиць;

СУБД – Система управління базами даних, програмне забезпечення, що дозволяє створювати, керувати та оптимізувати доступ до баз даних.

ВСТУП

Оцінка сучасного стану об’єкта розробки. В епоху, що характеризується стрімким технологічним прогресом, методи поширення інформації зазнали суттєвого впливу. З появою Інтернету традиційний процес читання, доступу та створення книг трансформувався. Нові цифрові майданчики докорінно змінили спосіб, у який люди взаємодіють з книгою, надавши безпрецедентні можливості для пошуку, створення та поширення своїх думок. Ці платформи переосмислили досвід читання. Відвідувачів фізичних книгарень з обмеженою кількістю полиць стає дедалі менше. На їхнє місце прийшли вебресурси з безмежними можливостями, доступними одним натисканням кнопки.

Перед людством постала потреба в переведенні значного обсягу інформації в електронний вигляд і інтегрування наявних сфер життєдіяльності, таких як освіта, наука, право тощо, з використанням новітніх інформаційних технологій. Одним з методів вирішення цього завдання є створення і підтримування актуальності електронних бібліотек.

Посилаючись на [1] можна визначити термін Електронна бібліотека (як синонім вживається “цифрова бібліотека”) – це інформаційна система призначена для накопичення, упорядкування, обліку, оброблення, зберігання, керування та використання електронних документів і для обслуговування користувачів бібліотеки через телекомунікаційні мережі.

Задачі створення електронної бібліотеки присвячено роботи українських та зарубіжних науковців: С. Назаровець [2] розглянув питання формування «Бібліотеки 4.0» з використанням передових інтернет-технологій, що демонструє зв’язок між розвитком інформаційних технологій в світі і потребою трансформації бібліотечних послуг. Н. Пасмор [3] розглянув електронні бібліотеки як елемент інформаційного суспільства.

Актуальність роботи. Створення електронної бібліотеки є актуальним, оскільки зростає популярність цифрової літератури. Технології впливають на

наше життя, тому стає важливою адаптація традиційних практик збереження інформації. Розпорядження Кабінету Міністрів України N 1579-р [4] описує актуальність проблеми з використанням, збереженням та поширенням інформації, що зберігається в бібліотеках, архівах та музеях. Це розпорядження визначає шляхи розв'язання цих проблем з допомогою переведення документів в електронний формат, в тому числі у вигляді електронних бібліотек.

Вебсайти забезпечують легкий доступ до широкого вибору літературних творів, задовольняючи попит на електронні засоби поширення інформації. Статистика свідчить, що продажі цифрових книг у світі неухильно зростають [5]. Цей тренд можна прослідкувати в Україні від початку створення легального цифрового ринку. М.Женченко [6] охарактеризувала моделі цифрової дистрибуції на книжковому ринку України. Базуючись на цій роботі можна стверджувати, що цифровий книжковий ринок України постійно зростає і є перспективним напрямком діяльності для створення відповідних інформаційних систем.

Окрім того, автори-початківці можуть використовувати платформу, щоб ділитися своїми історіями та отримувати конструктивний зворотний зв'язок, тим самим розвиваючи свої здібності. Ця платформа покликана задовольнити дедалі більший попит на цифрову літературу, а також розвивати суспільство в культурному плані.

В Україні діє законодавство про авторське право [7], яке регулює використання творів в електронному форматі. В даному контексті електронна бібліотека може стати потужним інструментом поширення інформації про законодавство України серед користувачів застосунку. Автори книг, що розміщуються на сайті, та користувачі будуть ознайомлені з авторським правом, що сприятиме усвідомленню принципів інтелектуальної власності і поширенню поваги до чужих робіт.

Мета й завдання роботи. Метою роботи є проєктування та подальша розробка вебсайту, що надає можливості користувачам для перегляду книг

онлайн, а авторам-початківцям для їх створення та поширення серед звичайних користувачів. Для досягнення цієї мети поставлено такі завдання:

- дослідити готові рішення, що наявні на ринку;
- створити функціональні вимоги до застосунку;
- спроєктувати базу даних;
- розробити серверну частину застосунку;
- створити базу даних;
- розробити клієнтську частину застосунку.

Об’єкт, методи й засоби розроблення. Об'єктом даної роботи є процес розробки інформаційного вебзастосунку, що полегшує доступ до електронних книг та їх створення, а також надає користувачам можливість для обговорення та оцінювання прочитаних робіт.

Методи розроблення включають комп'ютерне моделювання та розробку програмного продукту з використанням різних інструментів, таких як операційна система Windows 10, інтегроване середовище розробки Visual Studio, UML, C#, TypeScript, HTML, CSS, SQL, ASP.NET Core, EF Core, Microsoft SQL Server, а також бібліотеки React, AutoMapper, FluentValidation та Bulma.

Можливі сфери застосування. Вебсайт для онлайн-читання та створення книг має величезний потенціал у багатьох галузях. Він може використовуватися студентами і викладачами для доступу до підручників і дослідницьких матеріалів, що робить його цінним освітнім ресурсом. Видавництва можуть використовувати електронну бібліотеку для розповсюдження та просування цифрових публікацій. Платформа дозволяє авторам-початківцям здійснювати публікацію своїх робіт, отримувати дохід від реклами та спілкуватися з читачами.

РОЗДІЛ 1. ОГЛЯД ГОТОВИХ РІШЕНЬ

Перш ніж розпочати процес розробки електронної бібліотеки, важливо вивчити досвід наявних платформ, які є успішними на ринку. Можна отримати корисну інформацію та визначити ключові характеристики для створення успішної платформи, що відповідає поставленим задачам, оцінивши сильні та слабкі сторони таких популярних сайтів, як UkrLib [8], Wattpad [9], Goodreads [10] та Project Gutenberg [11]. Ці платформи мають широкий асортимент книг і пропонують інноваційні функції, які приваблюють читачів і заохочують потенційних авторів. Розуміючи переваги та недоліки цих платформ, можна створити вебзастосунок, який відповідатиме функціональним вимогам, задовольнятиме читачів та авторів та поширюватиме культуру читання з допомогою інтернету.

1.1 Платформа UkrLib

UkrLib [8] – це платформа, що займається популяризацією української культури. Вона поширює сучасну та класичну літератури в мережі. Платформа в основному сконцентрована на українських текстах. UkrLib пропонує величезну колекцію ресурсів, включаючи книги, статті, журнали та інші документи в різних форматах. UkrLib охоплює широкий спектр творів, що робить сайт придатним для різних користувацьких потреб.

Переваги платформи UkrLib:

- Зручний інтерфейс користувача.
- Орієнтація на українську літературу.
- Можливість завантаження книг у різному форматі.

- Регулярні оновлення вмісту.
- Збереження рідкісних книг.
- Детально пророблена головна сторінка сайту. Сайт складає хороше перше враження.
- Взаємодія з зовнішніми вебзастосунками таких як Youtube.

Недоліки платформи UkrLib:

- Відсутність профілю користувача і системи оцінювання книг.
- Відсутність зручної системи фільтрування вмісту.

1.2 Платформа Wattpad

Wattpad [9] – це платформа, якою користуються письменники та читачі по всьому світу. Wattpad приваблює авторів-початківців можливостями для розвитку і користувачів своєю великою бібліотекою користувацьких історій у різних жанрах. Сайт має понад 90 мільйонами активних користувачів станом на 2022 рік [12]. Підхід платформи, що орієнтований на спільноту, призвів до надзвичайної залученості: в середньому 23 мільярди хвилин на місяць витрачається для читання на платформі.

Переваги платформи Wattpad:

- Широкий асортимент історій, створених користувачами.
- Взаємодія в режимі реального часу та зворотний зв'язок між авторами та читачами.
- Архітектура застосунку, що стимулює активність користувачів.
- Можливість для авторів-початківців отримати увагу та конструктивну критику.

Недоліки платформи Wattpad:

- Труднощі з контролем якості через користувацький контент.
- Обмежений контроль над питаннями інтелектуальної власності та авторських прав.

1.3 Платформа Goodreads

Goodreads [10] – це вебсайт, який став притулком для любителів книг. Маючи понад 125 мільйонів користувачів [13], він пропонує безліч користувацьких рецензій, рекомендацій та обговорень книг. Goodreads полегшує персоналізований пошук книг. За часи існування накопичено колекцію з понад 3,5 мільярда книг, доданих користувачами.

Переваги платформи Goodreads:

- Широкий асортимент користувацьких рецензій та оцінок.
- Індивідуальні книжкові пропозиції на основі вподобань користувача.
- Можливість брати участь у книжкових товариствах та літературних дискусіях.
- Можливість збереження цитат з творів.

Недоліки платформи Goodreads:

- Обмежена увага до нових книг.
- Інтерфейс сайту перевантажений інформацією.
- Труднощі у просуванні маловідомих авторів та оригінальних творів.

1.4 Платформа Project Gutenberg

Project Gutenberg [11] – авторитетна платформа, присвячена збереженню та поширенню класичної літератури. Платформа надає понад 65 000 безплатних електронних книг у відкритому доступі, включаючи твори відомих авторів. Було розповсюджено понад 100 мільйонів примірників книг з допомогою "Проєкту Гутенберга". Завдяки цьому організація отримала світове визнання.

Переваги платформи Project Gutenberg:

- Широкий асортимент класичних творів в електронному форматі.
- Збереження та доступність класичних творів літератури та історичних документів.
- Можливість завантаження книг у різних форматах.
- Орієнтація на класичну літературу.

Недоліки платформи Project Gutenberg:

- Мало сучасних і нещодавно виданих творів.
- Брак інтерактивності та участі спільноти.

Проаналізувавши переваги та недоліки таких платформ, як UkrLib, Wattpad, Goodreads та Project Gutenberg, проведено аналіз вимог до застосунку на основі досвіду готових рішень. Сайт повинен мати широкий асортимент творів для зацікавлення різних читацьких груп. В короткостроковій перспективі потрібно завантажити книги, що знаходяться у вільному доступі та не обмежені авторським правом. Надалі важливим є створення активної спільноти авторів, що будуть генерувати вміст і примножувати відвідування сайту з боку їх читачів. Для цього потрібно створити вебзастосунок, що буде задовольняти наступним вимогам:

- швидкість роботи застосунку;
- функціональність при написанні та читанні тексту;

- наявність зв’язку між автором та його читачами;
- підсистема просування книг на основі статистичних даних;
- увага до робіт авторів-початківців;
- модерування вмісту книг, коментарів, відгуків;
- врахування основ UI/UX дизайну.

Попередній перелік потрібно використати як орієнтир при розробці застосунку, який сприятиме процвітанню спільноти, розширенню можливостей письменників та забезпечить користувачам якісний досвід використання застосунку.

РОЗДІЛ 2. ВИБІР ТЕХНОЛОГІЙ

2.1 Середовище програмування

В процесі розробки були використані наступні середовища програмування та операційна система:

Windows 11 від Microsoft – це зручна операційна система, яка підтримує широкий спектр програмних застосунків. Її використано для встановлення середовищ програмування та застосунків для тестування серверної частини застосунку.

Сервер застосунку був запрограмований в **Microsoft Visual Studio** – це комплексне інтегроване середовище розробки (IDE). Воно включає модифікацію коду, налагодження, тестування та розгортання, а також потужні інструменти та функції для розробки програмного забезпечення. Воно підтримує декілька мов програмування, фреймворки та безліч додаткових бібліотек.

Клієнт застосунку був створений в **Visual Studio Code** (VS Code) – це легкий редактор вихідного коду. Він легко налаштовується і підтримує широкий спектр мов програмування. VS Code пропонує такі важливі функції, як підсвічування синтаксису, заповнення та налагодження коду, а також широку екосистему розширень. Він сумісний з Windows 11, тому його використано для розробки клієнтської частини застосунку.

2.2 Серверна частина застосунку

2.2.1 Технологія ASP.NET Core

Основу серверу застосунку створено за допомогою ASP.NET Core – це надійний і гнучкий фреймворк для розробки web APIs, який включає інтеграцію з інструментами Object-Relational Mapping (ORM) та базами даних [14]. Це

поширене, багато-платформне рішення, яке дозволяє розробникам створювати масштабовані, надійні застосунки.

ASP.NET Core використано для створення REST API, що є однією з його можливостей. REST (Representational State Transfer) [15] – це архітектурний підхід, який дозволяє клієнтам взаємодіяти з сервером за допомогою стандартних методів HTTP, таких як GET, POST, PUT і DELETE. ASP.NET Core надає багатий набір функцій та інструментів, які полегшують створення REST API. Саме API відповідає за взаємодію між клієнтською частиною та сервером за допомогою запитів та відповідей на них. Схема REST API представлена на рис. 2.1

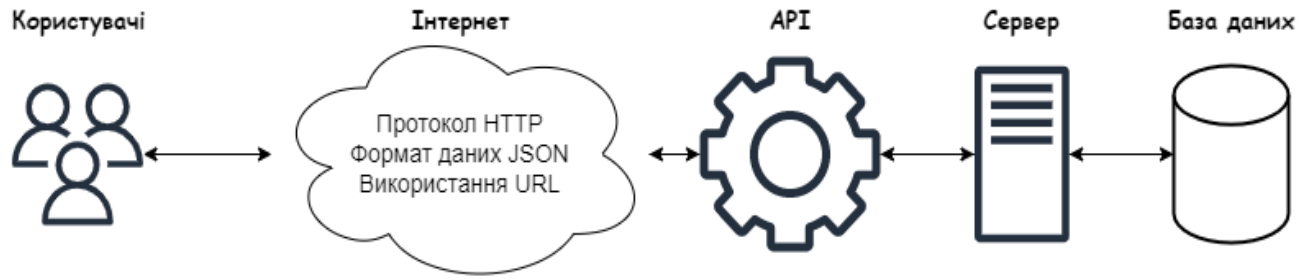


Рисунок 2.1 – Схема REST API

2.2.2 Система Microsoft SQL Server

Для збереження інформації застосунку обрано реляційну базу даних SQL Server.

Microsoft SQL Server – це СУБД, розроблена компанією Microsoft і відома своєю надійністю [16]. Основними функціями бази даних є цілісність даних, безпека, масштабованість і продуктивність. Програмне забезпечення легко інтегрується з екосистемою Microsoft, пропонуючи такі функції, як висока доступність, аварійне відновлення та можливості бізнес-аналітики.

У контексті використання SQL Server разом з Entity Framework Core (EF Core), можна скористатися функцією міграції, яку пропонує EF Core, щоб створити і контролювати схему бази даних. Процес міграції дозволяє поступово модифікувати схему бази даних, вносячи поступові зміни в модель. Entity

Framework Core автоматично створює необхідні оператори SQL для створення або модифікації схеми бази даних, залежно від визначених міграцій.

Інтеграція SQL Server і EF Core спрощує процес створення і підтримки бази даних, полегшуючи плавне включення моделі даних програми в базу даних SQL Server.

2.2.3 Фреймворк Entity Framework Core

Для доступу до бази даних SQL Server використано EF Core – багатоплатформну структуру ORM [17]. EF Core це інструмент, який спрощує відображення між об'єктами у програмному забезпеченні та таблицями реляційної бази даних. Використання строго типізованих об'єктів в EF Core полегшує доступ до бази даних, що призводить до більш спрощеного та інтуїтивно зрозумілого процесу створення застосунку.

EF Core пропонує значну перевагу в тому, що він може абстрагуватися від фундаментальних тонкощів бази даних, тим самим дозволяючи розробникам зосередитися на логіці застосунку замість того, щоб мати справу з низькорівневими операціями для доступу до бази даних. Фреймворк EF Core пропонує підтримку багатьох постачальників баз даних, зокрема SQL Server, що задовольняє вимоги проєкту.

Як було зазначено раніше, фреймворк EF Core забезпечує підтримку міграцій баз даних, дозволяючи поступово модифікувати схему бази даних за допомогою реалізації міграцій, що виконуються спочатку коду. Міграції можуть бути створені для полегшення додавання, модифікації або видалення об'єктів бази даних, при цьому EF Core автоматично створює необхідні SQL-скрипти для реалізації цих змін.

2.2.4 Бібліотека AutoMapper

AutoMapper – це бібліотека з відкритим вихідним кодом в екосистемі .NET, яка пропонує простий і практичний підхід до перетворення даних між різними типами (type casting) [18]. Процес перетворення типів вручну стає непотрібним завдяки автоматизованому налаштуванню відображень з допомогою конвенцій або явної конфігурації.

Ця бібліотека покликана розв’язати проблему повторного використання коду при перетворенні об’єктів сутностей (entity), що отримані з бази даних, до об'єктів передачі даних (DTOs), що є відповідями на HTTP-запити клієнтської частини застосунку та будуть використовуватися в користувацькому інтерфейсі.

2.2.5 Бібліотека FluentValidation

Бібліотека FluentValidation – це надійний інструмент валідації, розроблений для .NET застосунків. Мета цього інструменту – спростити реалізацію валідації даних в .NET застосунках [19]. Бібліотека забезпечує впорядковану та послідовну структуру для визначення критеріїв валідації, що дозволяє розробникам формувати складні алгоритми валідації у стислому та зрозумілому форматі використовуючи FluentValidation. Можливість адаптації дозволяє розробляти стійкі механізми валідації, які гарантують узгодженість правил і уникати повторів коду.

Однією з ключових переваг бібліотеки це можливість створювати та об'єднувати правила перевірки. Це дозволяє створювати ланцюжки процедур валідації для моделей даних. Завдяки інтеграції різних правил, можна реалізовувати складні сценарії валідації, забезпечуючи при цьому організацію та стабільність коду.

2.3 Клієнтська частина застосунку

2.3.1 Бібліотека React

React – це бібліотека JavaScript [20], для створення користувацьких інтерфейсів. Вона була розроблена співробітниками Facebook і стала досить популярною завдяки своїй простоті та ефективності. Розробники можуть використовувати React для створення застосунків, розбиваючи їх на менші, багаторазово використовувані елементи, також відомі як компоненти.

Багато відомих компаній, таких як Facebook, Instagram, Airbnb та Netflix, використовують React. Його популярність свідчить про продуктивність, масштабованість та надійність. Крім того, спільнота React є надзвичайно активною та відкритою, з великою кількістю онлайн-ресурсів, навчальних посібників та проєктів з відкритим вихідним кодом.

Одним з ключових аспектів React є його віртуальний DOM. React створює віртуальну версію вебсторінки, а не змінює її безпосередньо. Цей віртуальний DOM дозволяє React швидко визначити найменшу кількість змін, необхідних для оновлення реальної вебсторінки, коли змінюється стан компонента. React мінімізує непотрібні оновлення і в результаті забезпечує більш плавний та ефективний користувацький досвід.

React використовує декларативний підхід, що означає, що розробники можуть сконцентруватися на тому, як вони хочуть, щоб виглядав інтерфейс користувача залежно від стану застосунку.

Ще однією особливістю React є його акцент на повторному використанні компонентів. Кожен компоненти містить власну логіку та користувацький інтерфейс. Це означає, що компонент проєктується один раз і надалі повторно використовується у застосунку, заощаджуючи значну кількість часу і роботи. Крім того, це сприяє створенню чистої та впорядкованої структури коду, в якій кожен компонент має чітко визначену відповідальність, що полегшує управління та оновлення.

React має екосистему, повну цінних інструментів та бібліотек. Наприклад, React Router допомагає в навігації по застосунку, Redux керує управлінням станами, Axios оптимізує HTTP-запити, а Jest надає потужний фреймворк для тестування. Гнучкість React дозволяє використовувати його разом з іншими бібліотеками та фреймворками, забезпечуючи інтеграцію з різними технологіями та кодовими базами.

Також React можна використовувати для створення інтерфейсу мобільних застосунків. React Native – це фреймворк, побудований на основі React, який дозволяє створювати нативні мобільні застосунки для iOS та Android.

2.3.2 Фреймворк Bulma

Bulma – це CSS-фреймворк з відкритим вихідним кодом, він є відносно простим, зручним та орієнтованим на мобільні пристрої [21]. Фреймворк пропонує легкий та адаптивний набір класів CSS, які дозволяють розробникам створювати гнучкі та сучасні вебінтерфейси.

Підхід до адаптивного дизайну, який використовує Bulma, гарантує, що ваші вебзастосунки будуть бездоганно виглядати та функціонувати на різних пристроях. Система сіток на основі Flexbox дозволяє створювати макети без необхідності використання складних методів позиціювання.

Фреймворк включає численні попередньо стилізовані компоненти, зокрема навігаційні панелі, картки, заголовки, форми тощо. Ці готові до використання та легко модифіковані компоненти забезпечують міцну основу для швидкої розробки користувацьких інтерфейсів.

Документація до Bulma є вичерпною, добре організованою та зручною для користувача. Вона містить прості приклади, пояснення та інструкції з використання для кожного компонента та класу CSS, що спрощує інтеграцію Bulma у проєкти та використання її можливостей.

Сумісність Bulma з усіма сучасними браузерами є додатковою перевагою. Це гарантує, що зовнішній вигляд і функціональність вебзастосунків залишатимуться узгодженими на різних платформах і пристроях.

2.4 Технології авторизації та автентифікації

2.4.1 Стандарт JWT

JWT (JSON Web Token) – це стандарт токена доступу на основі JSON, стандартизованого в RFC 7519, який визначає компактний та безпечний спосіб передачі інформації у вигляді JSON об’єкту [22]. Він зазвичай використовується у інформаційних системах для авторизації та обміну інформацією.

Структура JWT містить три частини, що розділені крапками:

- заголовок (header);
- корисне навантаження (payload);
- підпис (signature).

Структура JWT подана на рис 2.2

Рисунок 2.2 – Структура Json Web Token

Заголовок включає специфічні для токена метадані, такі як алгоритм хешування, що використовується для токена.

Корисне навантаження складається з претензії (claims), які є твердженнями про сутність (як правило, користувача), та інших даних. Ці твердження можуть включати ім’я користувача, ролі, дозволи та інші подібні дані.

Поєднання закодованого заголовка, корисного навантаження і секретного ключа створює підпис. Він забезпечує цілісність та автентичність токена. Сервер перевіряє підпис і підтверджує токен за допомогою секретного ключа.

Сервер може автентифікувати та авторизувати користувача на основі інформації, що міститься в корисному навантаженні токена, якщо JWT включається в запити до сервера. Це забезпечує безпечний зв'язок між клієнтом і сервером.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ЗАСТОСУНКУ

3.1 Функціональні вимоги

Для проєктування функціональних вимог застосунку використано мову UML. Створено Use-case діаграму, що візуально представляє вимоги до підсистем адміністратора (див. рис. 3.1), авторизації (див. рис 3.2) та користувача (див. додаток А).

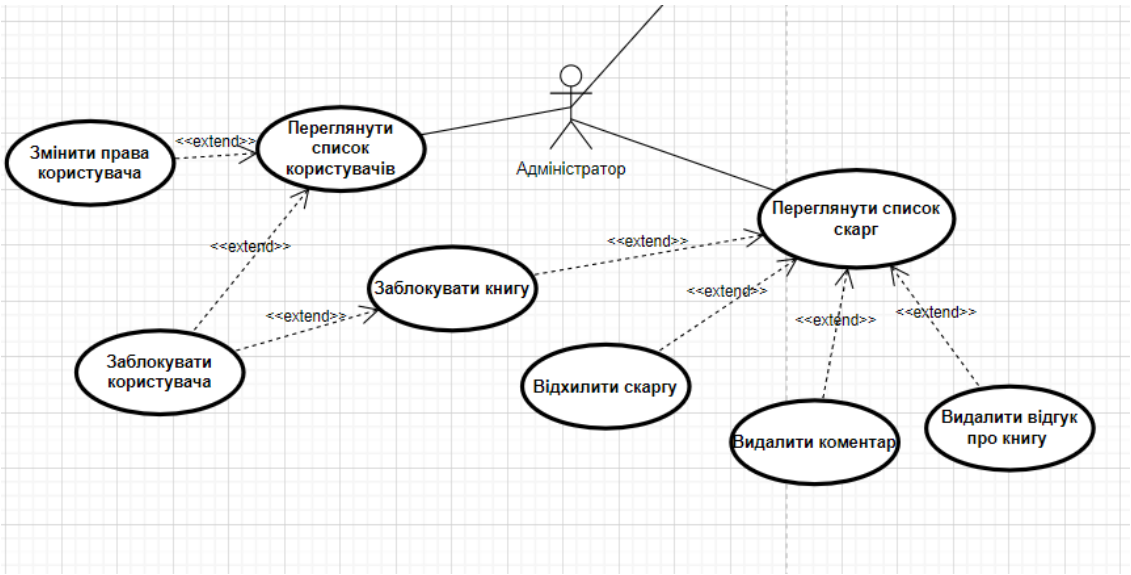


Рисунок 3.1 – Use-case діаграма підсистеми адміністратора

Рисунок 3.2 – Use-case діаграма підсистеми авторизації

В таблиці 3.1 представлені опис ключових функцій застосунку.

Таблиця 3.1 – Опис ключових вимог до застосунку

Назва	Опис
Доступ до сайту та книг	Неавторизований користувач повинен мати доступ до інформації та вмісту книг
Реєстрація та вхід	Неавторизований користувач може зареєструватися на сайті та ввійти у свій профіль
Редагування персональних даних	Авторизований користувач може редагувати дані профілю
Створення, редагування, видалення книг на сайті	Авторизований користувач повинен мати можливість створювати, редагувати та видаляти свої книги на сайті
Коментарі до розділів та відгуки до книг	Авторизований користувач може створити коментар до розділу або відгук до книги

Скарги на вміст	Користувач може поскаржитись на шкідливий вміст коментарів, відгуків чи книг
Слідкування за книгами	Авторизований користувач може почати слідкувати за книгою
Функції адміністратора	Адміністратор може видаляти профілі користувачів, книги, коментарі та відгуки

3.2 Реалізація архітектури

При створенні вебсайту, типовим рішення є впровадження клієнт-серверної архітектури. Розділивши вебзастосунок на клієнтські та серверні компоненти, можна скористатися перевагами цієї архітектури, щоб полегшити розробку.

Серверний компонент спроєктований з використанням багаторівневої архітектури. Клієнтський компонент застосунку використовує компонентну архітектуру оскільки створений з допомогою React.

Клієнт-серверна архітектура – це архітектурна парадигма, яка розділяє функціональність програми на два окремі компоненти: клієнт та сервер. Кілька клієнтів, таких як комп'ютери або мобільні пристрої, взаємодіють з централізованим сервером для доступу та використання ресурсів або функціоналу.

У цьому випадку клієнт, який також є інтерфейсом користувача, – це програмне забезпечення, яке ініціює HTTP запити до сервера з використанням web API. Він дозволяє користувачеві взаємодіяти з застосунком, роблячи запити до даних або сервісів і відображаючи результати через інтерфейс. Клієнт створений для використання в браузері на пристроях різної роздільної здатності.

Сервер, також відомий як внутрішня частина, є програмним застосунком, що відповідає за логіку отримання та обробки клієнтських запитів. Він керує та

надає доступ до спільних ресурсів, таких як бази даних, файли та бізнес-логіка сервера. Сервер відповідає на запити клієнта, виконуючи необхідні операції і повертаючи запитувані дані або помилки.

Зазвичай спілкування між клієнтом і сервером відбувається у форматі "request-response". Клієнт передає серверу запит, що містить бажану операцію та дані. Сервер отримує запит, обробляє його і формує відповідь, що містить запитувані дані або результат операції. Потім відповідь передається назад клієнту, який може відобразити результати або продовжити подальшу роботу.

В реалізованому застосунку взаємодію між компонентами клієнт-серверної архітектури можна переглянути в додатку Б.

3.3 Реалізація серверної частини застосунку

Результатом розробки серверної частини застосунку є база даних та сервер. Сервер застосунку розбитий на три рівні: Data Access Layer (DAL), Business Logic Layer (BLL), API layer.

3.3.1 База даних застосунку

У додатку В зображено діаграму бази даних застосунку. Короткий опис таблиць бази даних наведено в таблиці 3.2.

Таблиця 3.2 – Опис наявних в базі даних таблиць

Таблиця	**Опис**
Books	Таблиця для збереження інформації про книгу
Images	Таблиця, що зберігає фотографію книги, має зв’язок один до одного з таблицею Books
Reports	Таблиця, що зберігає скарги на книгу

BookGenres	Проміжна таблиця, що утворює зв’язок багато до багатьох між таблицями Books та Genres
Genres	Таблиця, що зберігає інформацію про жанри
Reviews	Таблиця, що зберігає відгуки користувачів про книгу
Users	Таблиця, що зберігає інформацію про користувачів
Comments	Таблиця, що зберігає коментарі користувачів
Chapters	Таблиця, що зберігає вміст книги у вигляді розділів

Описи атрибутів таблиць бази даних перелічені в таблицях 3.3 – 3.11.

Таблиця для збереження інформації про книги.

Таблиця 3.3 – Опис атрибутів таблиці Books

Атрибут	**Тип**	**Опис**
Id	Int	Ідентифікатор таблиці
Title	Nvarchar(100)	Назва книги
Description	Nvarchar(1000)	Опис книги
AuthorId	Int	Ідентифікатор користувача, що є автором книги
ImageId	Int	Ідентифікатор фотографії книги
Bookmarks	Int	Кількість збережень до користувацьких профілів
Views	Int	Кількість переглядів інформації про книгу

isCompleted	Bit	Логічний показник стану написання книги. (0 – книга в процесі написання, 1 – книга завершена)
AverageScore	Float	Середнє значення оцінок, що залишені у відгуках до книги

Таблиця для збереження інформації про фотографії книг.

Таблиця 3.4 – Опис атрибутів таблиці Images

Атрибут	**Тип**	**Опис**
Id	Int	Ідентифікатор фото
BookId	Int	Зовнішній ключ. Ідентифікатор книги, якій відповідає дане фото
ContentType	Nvarchar(max)	Тип фото
ImageData	Varbinary(max)	Фотографія книги у бінарному форматі

Таблиця для збереження інформації про скарги на книгу.

Таблиця 3.5 – Опис атрибутів таблиці Reports

Атрибут	**Тип**	**Опис**
Id	Int	Ідентифікатор скарги

BookId	Int	Зовнішній ключ. Ідентифікатор книги, яка має дану скаргу
Text	Nvarchar(max)	Текст скарги

Допоміжна таблиця бази даних, що утворює зв’язок багато до багатьох між таблицями жанрів та книг.

Таблиця 3.6 – Опис атрибутів таблиці BookGenres

Атрибут	**Тип**	**Опис**
Id	Int	Ідентифікатор проміжного запису між книгою та жанром
BookId	Int	Зовнішній ключ. Ідентифікатор книги
GenreId	Int	Зовнішній ключ. Ідентифікатор жанру

Таблиця для збереження інформації про жанри.

Таблиця 3.7 – Опис атрибутів таблиці Genres

Атрибут	**Тип**	**Опис**
Id	Int	Ідентифікатор жанру
Name	Nvarchar(100)	Назва жанру

Таблиця для збереження інформації про відгуки.

Таблиця 3.8 – Опис атрибутів таблиці Reviews

Атрибут	**Тип**	**Опис**
Id	Int	Ідентифікатор відгуку
UserId	Int	Зовнішній ключ. Ідентифікатор користувача, що залишив відгук
BookId	Int	Зовнішній ключ. Ідентифікатор книги, яка отримала відгук
Content	Nvarchar(max)	Текст відгуку
Score	Int	Оцінка, що залишили у відгуку (від 0 до 5)
Likes	Int	Кількість вподобань, що отримав відгук
IsReported	bit	Логічний показник, що вказує на можливий неприйнятний вміст відгуку

Таблиця для збереження інформації про користувачів.

Таблиця 3.9 – Опис атрибутів таблиці Users

Атрибут	**Тип**	**Опис**
Id	Int	Ідентифікатор користувача
Email	Nvarchar(50)	Унікальна електронна адреса користувача
Username	Nvarchar(50)	Унікальне ім’я користувача

Password	Nvarchar(100)	Пароль користувача
Role	Nvarchar(15)	Роль користувача в застосунку

Таблиця для збереження інформації про коментарі.

Таблиця 3.10 – Опис атрибутів таблиці Comments

Атрибут	**Тип**	**Опис**
Id	Int	Ідентифікатор коментаря
UserId	Int	Зовнішній ключ. Ідентифікатор користувача, що залишив коментар
ChapterId	Int	Зовнішній ключ. Ідентифікатор розділу, до якого залишили коментар
Content	Nvarchar(1000)	Вміст коментаря
Likes	Int	Кількість вподобань, що отримав коментар
IsReported	Bit	Логічний показник, що вказує на можливий неприйнятний вміст коментаря

Таблиця для збереження інформації про розділи книги.

Таблиця 3.11 – Опис атрибутів таблиці Chapters

Атрибут	**Тип**	**Опис**

Id	Int	Ідентифікатор розділу
Title	Nvarchar(100)	Назва розділу
Content	Nvarchar(max)	Вміст розділу
ChapterNumber	Int	Номер розділу
BookId	int	Зовнішній ключ. Ідентифікатор книги, що містить розділ

Розглянемо причину створення таблиці Images і її тип зв’язку (один до одного) з таблицею Books. В процесі створення застосунку стало очевидно, що фото займають набагато більше пам’яті ніж текстова інформація про книгу. Їх поєднання в одну таблицю призведе до проблем з передачею даних через мережу. Для розв’язання цієї проблеми прийнято рішення створити окрему таблицю – Images, ця таблиця зберігає фото у бінарному форматі і має відношення один до одного з таблицею книг. При завантаженні книги клієнт робитиме два HTTP запити, один для текстової інформації про книгу і другий для фотографії, в такому випадку проблеми з передачею об’ємного фото через мережу не блокуватимуть відображення текстової інформації про книг. Замість фото книги буде використано статичне фото (див. рис. 3.3), що зберігається на клієнті і сигналізує про відсутність фото. Це дозволить користувачу продовжити роботу з застосунком незалежно від наявності фото однієї чи декількох книг.

Рисунок 3.3 – Замінник фотографії

3.3.2 Рівень Data Access Layer

Основним функціоналом DAL є доступ до бази даних з використанням EF Core, підходу code-first та міграцій бази даних. Підхід code-first – це поступове написання моделі даних та її налаштування в коді сервера і з використанням міграції створення та поступове внесення змін до бази даних SQL Server. Оскільки кожна таблиця бази даних повинна мати ідентифікатор, створено клас BaseEntity, що містить ID і від якої наслідуються всі інші сутності.

Код DAL розглянуто на прикладі сутності жанрів. Клас, що представляє жанр можна побачити на рис. 3.4.

```
public class Genre : BaseEntity
{
    [Required]
    [MaxLength(100)]
    [Column(TypeName = "nvarchar(100)")]
    public string Name { get; set; }

    public virtual ICollection<BookGenre> BookGenres { get; set; }

    public Genre()
    {
        BookGenres = new HashSet<BookGenre>();
    }
}
```

Рисунок 3.4 – Клас сутність Genre

Налаштування моделі відбувається з використанням атрибутів, наприклад [Required] – позначає необхідність даних, тобто неможливо записати в базу даних жанр без імені.

З використанням атрибутів налаштовано типи колонок, імена таблиць, максимальна розмірність даних, зв’язки між таблицями тощо.

Створено контекст бази даних, що діє як міст між застосунком і базою даних. Він керує підключенням до бази даних, конфігурацією сутностей і надає методи для CRUD-операцій. Контекст відстежує зміни в сутностях, підтримує транзакції, дозволяє робити запити за допомогою LINQ і пропонує функції оптимізації продуктивності, такі як кешування і ліниве завантаження.

Код контексту бази даних можна побачити на рисунку 3.5.

```
public class LibDbContext : DbContext
{
    public virtual DbSet<Book> Books { get; set; }
    public virtual DbSet<Genre> Genres { get; set; }
    public virtual DbSet<Chapter> Chapters { get; set; }
    public virtual DbSet<User> Users { get; set; }
    public virtual DbSet<Comment> Comments { get; set; }
    public virtual DbSet<Review> Reviews { get; set; }
    public virtual DbSet<Image> Images { get; set; }
    public virtual DbSet<BookGenre> BookGenres { get; set; }
    public virtual DbSet<Report> Reports { get; set; }

    public LibDbContext(DbContextOptions<LibDbContext> options) ...

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        base.OnConfiguring(optionsBuilder);
    }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<User>(entity =>
        {
            entity.HasIndex(e => e.Email).IsUnique();
            entity.HasIndex(e => e.Username).IsUnique();
        });

        modelBuilder.Entity<Book>(entity =>
        {
            entity.HasOne(a => a.Author).WithMany(a => a.Books).HasForeignKey(s => s.AuthorId);
            entity.HasOne(i => i.Image).WithOne(b => b.Book).HasForeignKey<Image>(i => i.BookId);
            entity.HasIndex(t=>t.Title).IsUnique();
```

Рисунок 3.5 – Реалізація класу контексту з використанням EF Core

Цей контекст бази даних зареєстрований в DI в ASP.NET Core, щоб забезпечити доступ до нього, а отже й до бази даних через EF Core, з будь-якої частини сервера. Редагуючи контекст, класи сутностей та налаштування, з використанням міграцій створено базу даних, що розглянута в підрозділі 3.3.1.

3.3.3 Рівень Business Logic Layer

Business Logic Layer (BLL) – це ключовий рівень програми. У BLL створено класи сервіси (services). Кожен окремий сервіс об’єднують в собі спільний функціонал, зазвичай за певним типом даних. Наприклад, на рисунку 3.6 зображено програмний інтерфейс, який реалізує клас UserService.

```
public interface IUserService
{
    Task<OperationResult<UserModel>> GetUserAsync(string email, string password);

    Task<OperationResult<List<UserModel>>> GetAllUsersAsync();

    Task<OperationResult<UserModel>> GetUserByEmailAsync(string email);

    Task<OperationResult<UserModel>> RegisterUserAsync(RegisterModel user);

}
```

Рисунок 3.6 – Код інтерфейсу, що впроваджує клас UserService

Цей інтерфейс містить оголошення методів реєстрації, отримання та видалення користувачів. Сервіси виконують передачу даних, взаємодіють з рівнем доступу до бази даних, приводять тип класів сутностей в об'єкти передачі даних (DTO), взаємодіють з рівнем доступу до даних, валідують дані, щоб переконатися, що вони є правильними та узгодженими.

На рівні BLL створені DTO, що використовуються для полегшення передачі даних між різними рівнями сервера або компонентами застосунку. Класи DTO не містять даних з класів сутностей, які не повинен отримувати користувач, і забезпечує структурований спосіб обміну інформацією.

Наприклад, для реєстрації користувач не повинен вводити бажану роль на сайті, ця інформація встановлюється автоматично на сервері тобто є внутрішньою, відповідний DTO можна переглянути на рисунку 3.7.

```
public class RegisterModel
{
    public string Email { get; set; }

    public string Username { get; set; }

    public string Password { get; set; }

    public string ConfirmPassword { get; set; }
}
```

Рисунок 3.7 – Клас DTO, що відповідає за реєстрацію користувача

Окрім вищезгаданого, в BLL відбувається валідація даних користувача і формується відповідь на запити, надіслані з web API. За допомогою бібліотеки FluentValidation, виконується валідація DTO, для цього завдання створено набір класів валідаторів, що містять правила валідації. Наприклад, для моделі реєстрації очікується, що пароль та підтвердження паролю збігаються, клас валідатор можна побачити на рисунку 3.8.

```
public class RegisterModelValidator:AbstractValidator<RegisterModel>
{
    private readonly LibDbContext _context;

    public RegisterModelValidator(LibDbContext context)
    {
        _context = context;

        RuleFor(x=>x.Email).NotNull().EmailAddress().Custom((email, context) =>
        {
            if (_context.Users.Any(x => x.Email == email))
            {
                context.AddFailure("The email is used by another user");
            }
        });
        RuleFor(x => x.Username).NotNull().NotEmpty().Length(4, 20).Custom((username, context) =>
        {
            if (_context.Users.Any(x => x.Username == username))
            {
                context.AddFailure("The username is used by another user");
            }
        });
        RuleFor(x=>x.Password).NotNull().NotEmpty().Length(6,20);
        RuleFor(x=>x.ConfirmPassword).NotNull().NotEmpty().Equal(x=>x.Password);
    }
}
```

Рисунок 3.8 – Клас валідатор класу RegisterModel

Використання класу валідатора можна побачити на рисунку 3.9.

```
ValidationResult validationResult = _validators.registerModelValidator.Validate(registerModel);

if(!validationResult.IsValid) {
    return OperationResult<UserModel>.Failture(validationResult.Errors.Select(e=>e.ErrorMessage).ToArray());
}
```

Рисунок 3.9 – Використання класу валідатора з рис. 3.8

На рівні доступу до бази даних оголошено набір класів сутностей, що використовує EF Core в доступі до БД, але рівень бізнес-логіки повинен повертати класи DTO як результат на запити, це породжує проблему повторення великих шматків коду для приведення типів даних в кожному наявному методі. Ця

проблема вирішена з використанням бібліотеки AutoMapper. Створено клас AutoMapperProfile, що містить набір правил які перетворюють класи сутності в DTO. Правила приведення типу RegisterModel до типу User зображені на рисунку 3.10.

```
public class AutoMapperProfile : Profile
{
    public AutoMapperProfile()
    {
        CreateMap<RegisterModel, User>();
        CreateMap<UserModel, User>();
        CreateMap<User, UserModel>();
        CreateMap<RegisterModel, UserModel>();
```

Рисунок 3.10 – Правила приведення типів в AutoMapperProfile

Варто зазначити, що сервіс не потребував специфічних приведень типів, тому використовується конвенція приведення назви полів початкового класу до назви полів кінцевого класу. Поля, що не мають збігів, ігноруються при приведенні типів. Використання оголошених правил можна побачити на рисунку 3.11.

```
User user = await _context.Users.FirstAsync(x => x.Email == email);
UserModel userDTO = _mapper.Map<UserModel>(user);
return OperationResult<UserModel>.Success(userDTO);
```

Рисунок 3.11 – Використання бібліотеки AutoMapper для приведення типів

Для повернення результату з довільного сервісу створено клас шаблон OperationResult, він містить в собі поля isSuccess, Errors, Entity та статичні методи для створення екземпляра класу (конструктор класу позначений як private).

Кожен сервіс використовує try-catch блоки для обробки помилок, що виникають при валідації моделі або при роботі з базою даних через EF Core. Якщо виникає помилка, сервіс повертає клас OperationResult з негативним результатом роботи і набором помилок. Якщо операція успішна, то повертається

позитивний результат роботи і відповідна інформація. Обробку помилок можна побачити на рисунку 3.12.

```
public async Task<OperationResult<ChapterModel>> GetChapterAsync(int chapterId)
{
    if (!_context.Chapters.Any(x=>x.Id == chapterId))
        return OperationResult<ChapterModel>.Failture("The specified chapter ID was not found in the database");

    try
    {
        var chapter = await _context.Chapters.FirstAsync(x => x.Id == chapterId);
        return OperationResult<ChapterModel>.Success(_mapper.Map<ChapterModel>(chapter));
    }
    catch (Exception ex)
    {
        return OperationResult<ChapterModel>.Failture("Internal database error");
    }
}
```

Рисунок 3.12 – Обробка помилок з використанням try-catch та класу OperationResult

Окремо слід відзначити клас AuthService. Це сервіс, що відповідає за створення JWT токена для користувача. З допомогою секретного ключа, що зберігається на сервері, відбувається створення токена. Токен містить електронну пошту, ім’я та роль користувача в системі.

Відокремлюючи бізнес-логіку від інших рівнів, таких як рівень представлення та рівень доступу до даних, BLL допомагає забезпечити модульність, обслуговування та повторне використання. Це полегшує зміну та оновлення основної логіки програми, не впливаючи на інші частини.

3.3.4 Рівень API Layer

API Layer – це рівень, що представляє програмний інтерфейс для доступу до функціонала сервера через мережу. Основним його завданням є прийняття HTTP запитів, їх обробка, перенаправлення на рівень бізнес-логіки і формування HTTP відповідей з відповідними HTTP кодами та даними чи помилками.

Web API створено за допомогою ASP.NET. Реалізовано набір контролерів в яких визначено методи з відповідними адресами доступу через мережу. API Layer використовує класи сервіси, що визначені на рівні бізнес-логіки.

Наприклад код класу контролера, що визначає метод для отримання розділу певної книги за його ідентифікатором можна переглянути на рисунку 3.13.

```
[Route("api/[controller]")]
[ApiController]
public class ChapterController : ControllerBase
{
    private IChapterService _chapterService;

    public ChapterController(IChapterService chapterService)
    {
        _chapterService = chapterService;
    }

    [HttpGet("{chapterId}")]
    [Authorize]
    public async Task<IActionResult> GetChapterAsync(int chapterId)
    {
        var chapterOperation = await _chapterService.GetChapterAsync(chapterId);

        if(!chapterOperation.IsSuccess)
        {
            return BadRequest(new {errors =  chapterOperation.Errors});
        }

        return Ok(new { data = chapterOperation.Entity });
    }
```

Рисунок 3.13 – Код класу контролера для розділів

Контролери містять атрибути [Authorize], що позначають рівень доступу до методів чи контролера. Наприклад, на рисунку 3.13 даний атрибут застосовується до методу GetChapterAsync, тобто доступ до вказаного методу мають лише авторизовані користувачі. Також в атрибуті [Authorize] можна вказати набір ролей, що мають відповідний доступ.

Перед доступом до web API запити проходять через системний middleware, метод що відповідає за процес авторизації та автентифікації користувача на сервері. Запит користувача, з наявним чи відсутнім токеном надходить до даного проміжного методу, якщо токен дійсний, проміжне програмне забезпечення витягує з нього ідентифікаційні дані користувача і встановлює їх у поточному контексті виконання застосунку, роблячи доступними для сервера. Якщо токен недійсний або відсутній, проміжне програмне забезпечення налаштоване на надання анонімного доступу.

3.4 Реалізація клієнтської частини застосунку

Для доступу до програмного інтерфейсу сервера, створено окремі функції, що займаються формуванням HTTP запитів і обробкою результаті. Ці функції використовуються в компонентах для отримання даних і їх подальшого відображення для користувача.

На клієнті для уникнення передачі даних вручну через властивості компонентів на кожному рівні дерева компонентів використовується Context API. Наприклад, JWT token авторизованого користувача повинен бути доступним завжди, тому що отримання даних з сервера без токена поверне помилку.

Для навігації по клієнту використана бібліотека React Router. Компоненти з закінченням Page є такими, що представляють сторінку користувача та об’єднують інші компоненти. React Router ставить у відповідність до введеного URL в браузері певну сторінку, це можна побачити на рисунку 3.14.

```
const router = createBrowserRouter([
  {
    path: "/",
    element: <HomePage />,
    errorElement: <ErrorPage />,
  },
  {
    path: "/auth",
    element: <AuthPage />,

    children: [
      {
        path: "login",
        element: <LoginForm />,
      },
      {
        path: "register",
        element: <RegisterForm />,
      },
    ],
  },
```

Рисунок 3.14 – Використання бібліотеки React Router

Якщо URL незареєстрований в React Router то буде повернуто ErrorPage.

Клієнт створений в компонентній архітектурі, тобто створені функціональні компоненти, що повертають певні частини JSX коду які є поєднанням HTML та JS. Вони повторно використовуються в інших компонентах і таким чином утворюється дерево компонентів. Існує кореневий компонент застосунку, що містить компонент сторінки в залежності від URL. Наприклад, JSX код сторінки SearchPage, що утворена з компонентів Navbar, Footer, BookSearch та BookGrid, подано на рисунку 3.15.

```
export function SearchPage() {
  return (
    <div className="page">
      <Navbar />
      <div className=" container">
        <div className="block" style={{ width: "100%" }}>
          <BookSearch />
        </div>
        <BookGrid name="Result" />
      </div>
    </div>
  );
}
```

Рисунок 3.15 – JSX код сторінки SearchPage

Для формування інтерфейсу користувача використано готові CSS класи, що наявні в бібліотеці Bulma.

Для реалізації відображення, написання та редагування тексту на клієнті, було використано текстовий редактор Quill [23]. Він має простий і зручний користувацький інтерфейс, який дозволяє користувачам легко створювати та оновлювати текстовий контент (див. рис. 3.16). Quill надає можливість використовувати опції форматування тексту (напівжирний, курсив, підкреслення), списки, таблиці тощо.

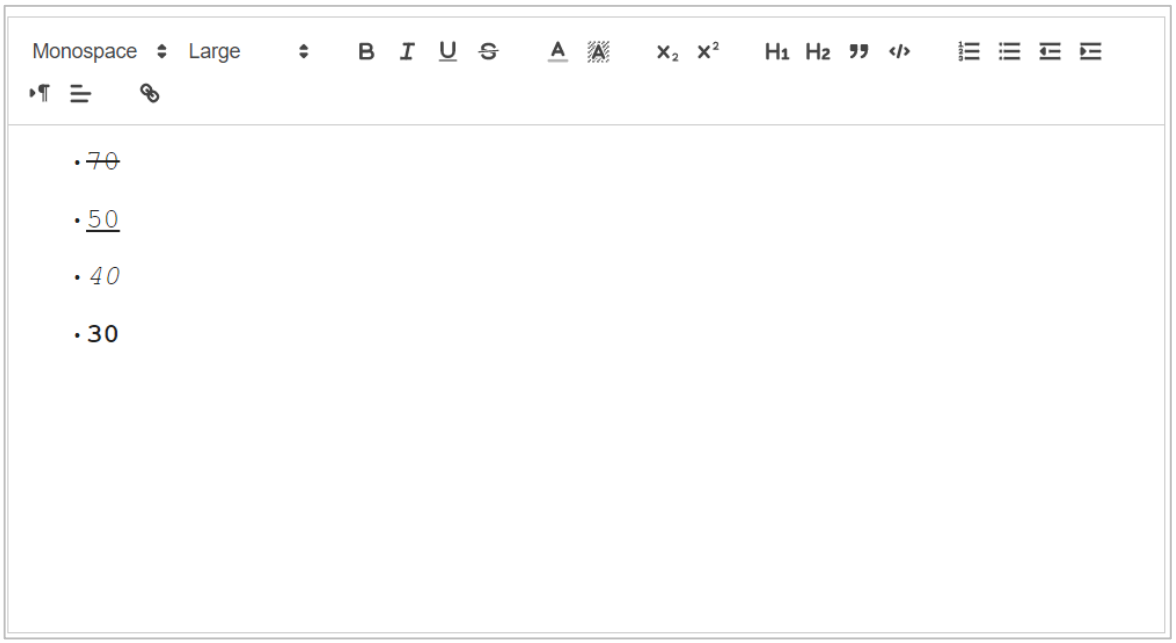


Рисунок 3.16 – текстовий редактор Quill

Результатом роботи в Quill є текст, що промаркований мовою HTML та надалі зберігається в базі даних застосунку. Оскільки застосунок орієнтований на роботу в браузері цей текст не потрібно жодним чином обробляти, лише отримати з сервера та відобразити для користувача. Даний підхід має недолік у вигляді більшої потреби у пам’яті для збереження тексту. Отже, функціональна вимога до можливостей при роботі з текстом виконана з мінімальними зусиллями.

3.5 Інструкція користувача

Неавторизований користувач має доступ до головної сторінки сайту, що містить таблиці з книгами, які підібрані за певним критерієм (див. рис. 3.17).

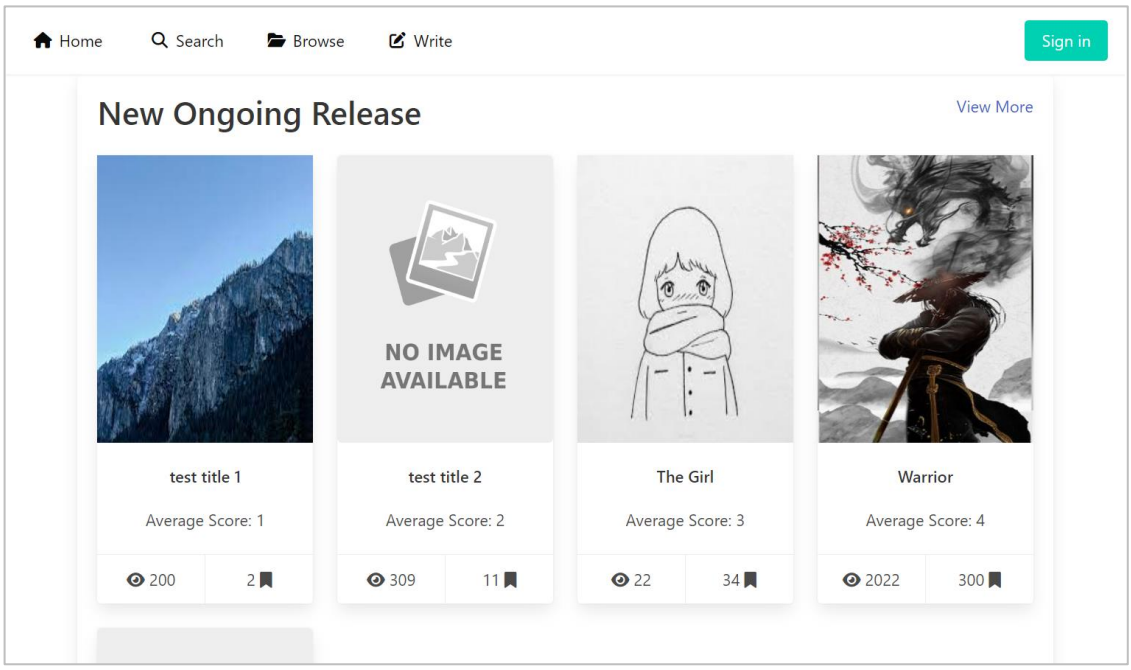


Рисунок 3.17 – Головна сторінка сайту

Зверху присутня панель навігації по вебзастосунку, де користувач може перейти на наступні сторінки: домашня сторінка, сторінка пошуку, сторінка фільтрів, сторінка авторизації, сторінка створення книг.

Сторінка авторизації містить форму для входу в профіль користувача(див. рис. 3.18) та посилання на сторінку реєстрації та відновлення пароля.

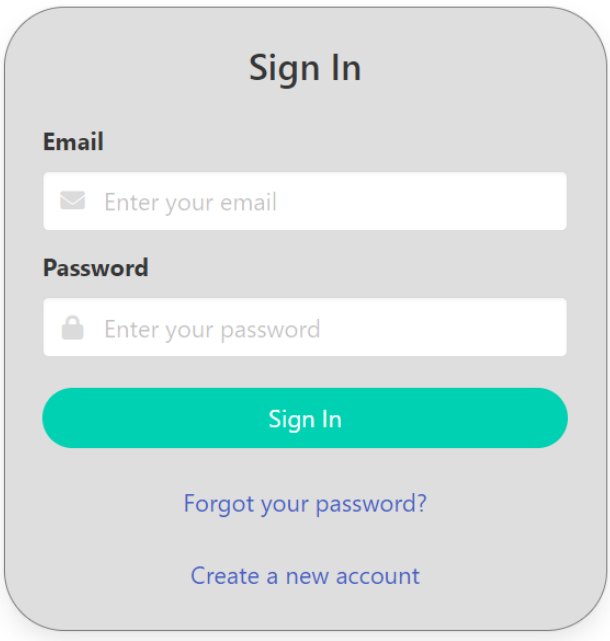


Рисунок 3.18 – Форма входу в профіль користувача

Сторінка реєстрації містить форму реєстрації користувача (див. рис. 3.19) і посилання до сторінки авторизації.

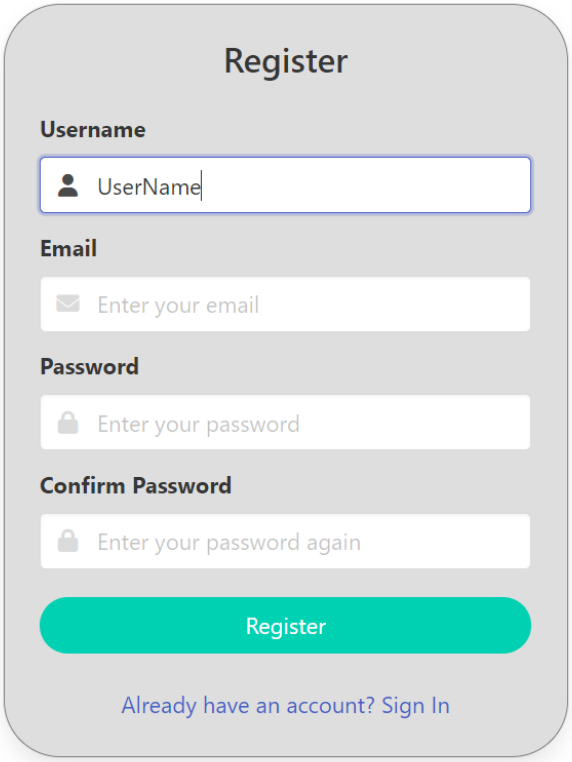


Рисунок 3.19 – Форма реєстрації користувача

На сторінці фільтрування (див. рис. 3.20) користувач може ввести назву книги, частково чи повністю, і провести пошук. Результат пошуку буде відображений у вигляді набору книг.

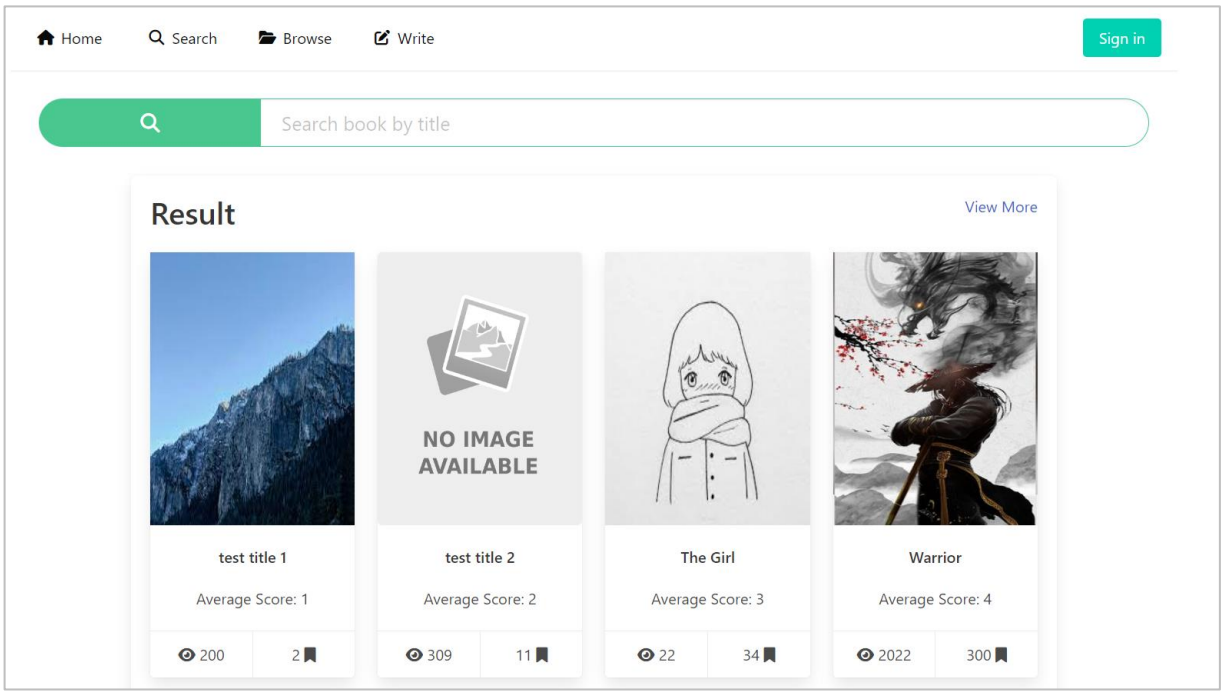


Рисунок 3.20 – Сторінка пошуку

Сторінка фільтрів (див. рис 3.21) складається з набору фільтрів й сітки результату.

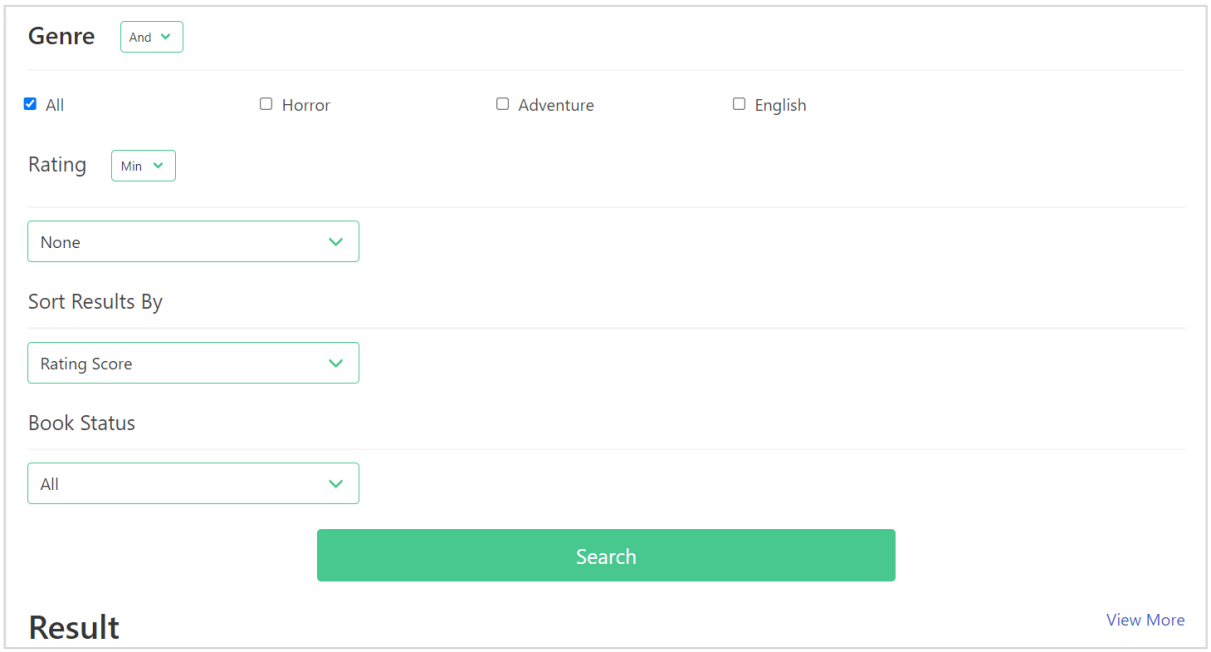


Рисунок 3.21 – Сторінка фільтрів

Користувач може фільтрувати книги за наступними параметрами:

- жанр книги. Книги фільтруються за набором жанрів, також присутній універсальний вибір всіх жанрів (‘all’). Користувач має кнопку для логічного вибору книг за жанром. ‘And’ вказує, що книга повинна мати всі обрані жанри, ‘Or’ вказує, що книга повинна мати хоча б один з обраних жанрів;
- рейтинг книги. Книги можна відбирати за рейтингом, вказавши мінімально чи максимально можливий рейтинг книги;
- статус написання книги. Користувач може обрати серед всіх книг (ігнорувати фільтр), завершених книг та книг, які в процесі написання. Цей параметр книги редагується її автором.

Для сортування результату фільтрації потрібно обрати одну з наступних характеристик книги:

- рейтинг за спаданням;
- кількість відгуків;
- кількість закладок в бібліотеку користувачів;
- кількість переглядів книги;
- кількість розділів книг;
- заголовок книги.

Елементи сітки книг є посиланнями на сторінку відповідної книги (див. рис. 3.22). Користувач може переглядати статистику, опис, жанри, список розділі (див. рис. 3.23) та відгуки до книги.

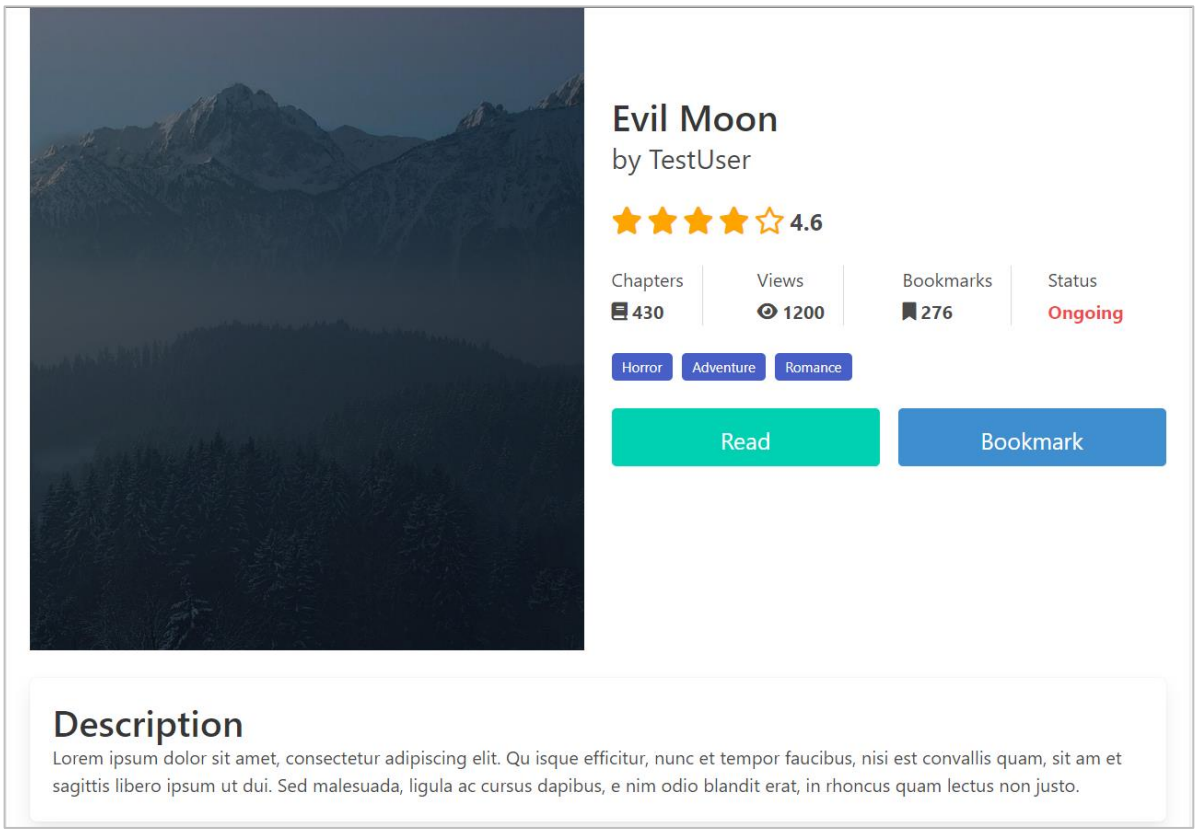


Рисунок 3.22 – Сторінка книги

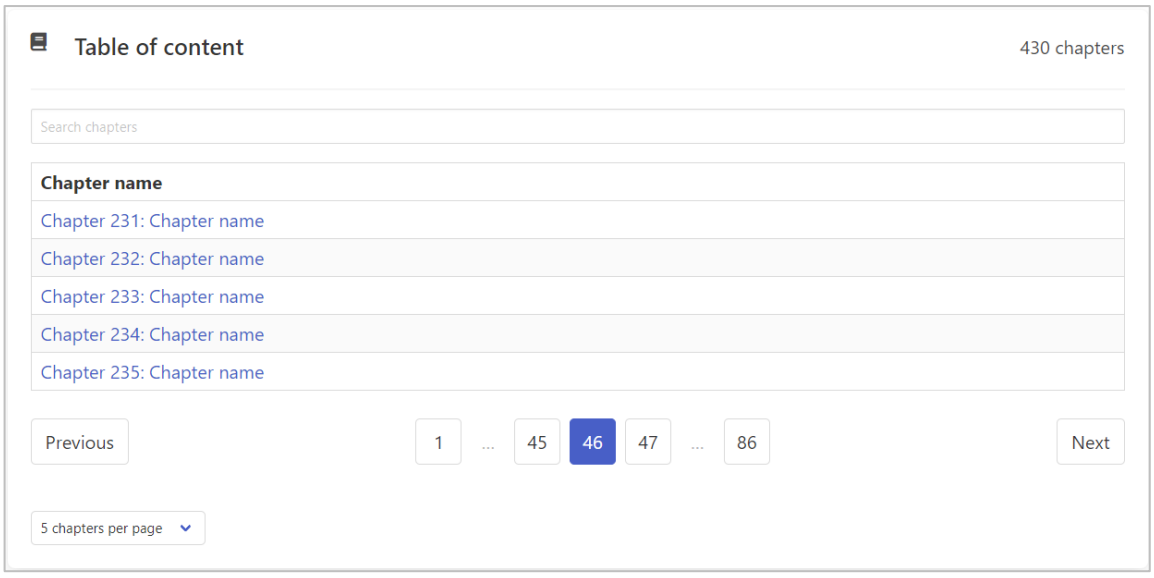


Рисунок 3.23 – Список розділів

Відкрити певний розділ можна через список розділів, що є посиланнями на відповідну сторінку. Кнопка ‘Read’ відкриває перший розділ книги.

Сторінка розділу (див. рис. 3.24) містить його вміст, а також посилання на сторінку книги, попередній та наступний розділи. Користувач може змінити розмір тексту та його стиль.

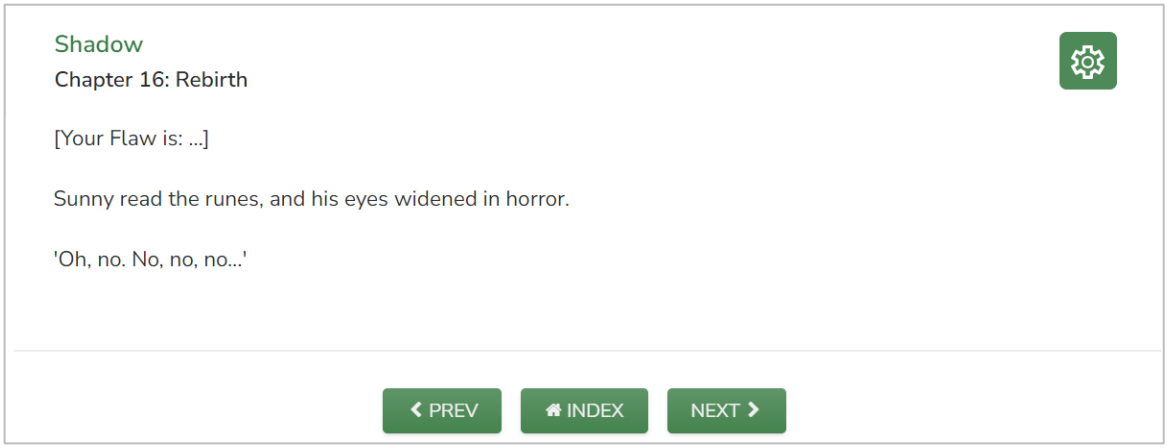


Рисунок 3.24 – Сторінка розділу

Авторизований користувач на сторінці книги, окрім згаданих речей, може додати книгу в бібліотеку та залишити відгук до книги (див. рис. 3.25).

Рисунок 3.25 – форма для відгуків до книги.

Авторизований користувач може перейти на сторінку написання книги. Для створення книги потрібно завантажити фото, вказати назву, жанри та опис. До книги можна додати розділ, для якого потрібно вказати назву та вміст. Книги та розділи можна редагувати та видаляти.

Після авторизації користувач отримує доступ до профілю. Профіль містить список книг, інформацію про користувача та усі відгуки про книги. Користувач може змінити ім’я і пароль, а також видалити свій акаунт.

3.6 Ідеї розвитку застосунку

Створений застосунок можна покращити впроваджуючи наступний функціонал:

- додати функціонал досягнень для заохочення відвідування користувачів;
- створити стрічку новин;
- розробити можливість монетизації книг, повністю або частково. Всі налаштування монетизації надати автору книги;
- інтеграція платіжних системи, наприклад GooglePay, ApplePay тощо;
- відокремити частину обов’язків адміністратора на користь модераторів, що будуть контролювати вміст книг та відгуків користувачів;
- розширення наявного функціонала.

ВИСНОВКИ

В ході виконання кваліфікаційної роботи був розроблений вебсайт, який надає можливість користувачам зберігати, створювати і ділитися інформацією у вигляді книг. Створений вебзастосунок є чудовою платформою для авторів-новачків, щоб спробувати популяризувати свою творчість. Проаналізовано найпопулярніші інформаційні застосунки, що мають схоже призначення. Їх переваги та недоліки були враховані при розробці.

В процесі виконання роботи було спроєктовано базу даних, сервер та клієнт вебзастосунку. Для побудови серверних та клієнтських компонентів було використано ASP.NET та React. За допомогою методу code-first була успішно створена база даних в MS SQL Server. Стандарт JWT був використаний для додавання методів авторизації та аутентифікації, що покращило роботу застосунку. Користувацький інтерфейс створений з допомогою бібліотеки Bulma.

Пересічний відвідувач вебзастосунку має доступ до вмісту книг. Вебзастосунок складається з підсистеми користувача, що дозволяє слідкувати за улюбленими книгами та створювати власні книги, а також підсистеми адміністратора для контролю над користувацьким вмістом і можливістю зміни внутрішньої інформації.

В процесі виконання роботи, виконавець значно покращив свої практичні навички та застосував теоретичні знання, здобуті під час навчання, на практиці. Проєкт показав, як використовувати різні методи програмування та інструменти розробки, в результаті чого був створений застосунок, який працює відповідно до функціональних вимог і є простим у використанні. Таким чином, я вважаю, що цілі цієї дипломної роботи були виконані. Також, розроблено план майбутнього розвитку застосунку, що покращить наявний функціонал та створить можливість використання вебзастосунку на ринку електронних книг.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

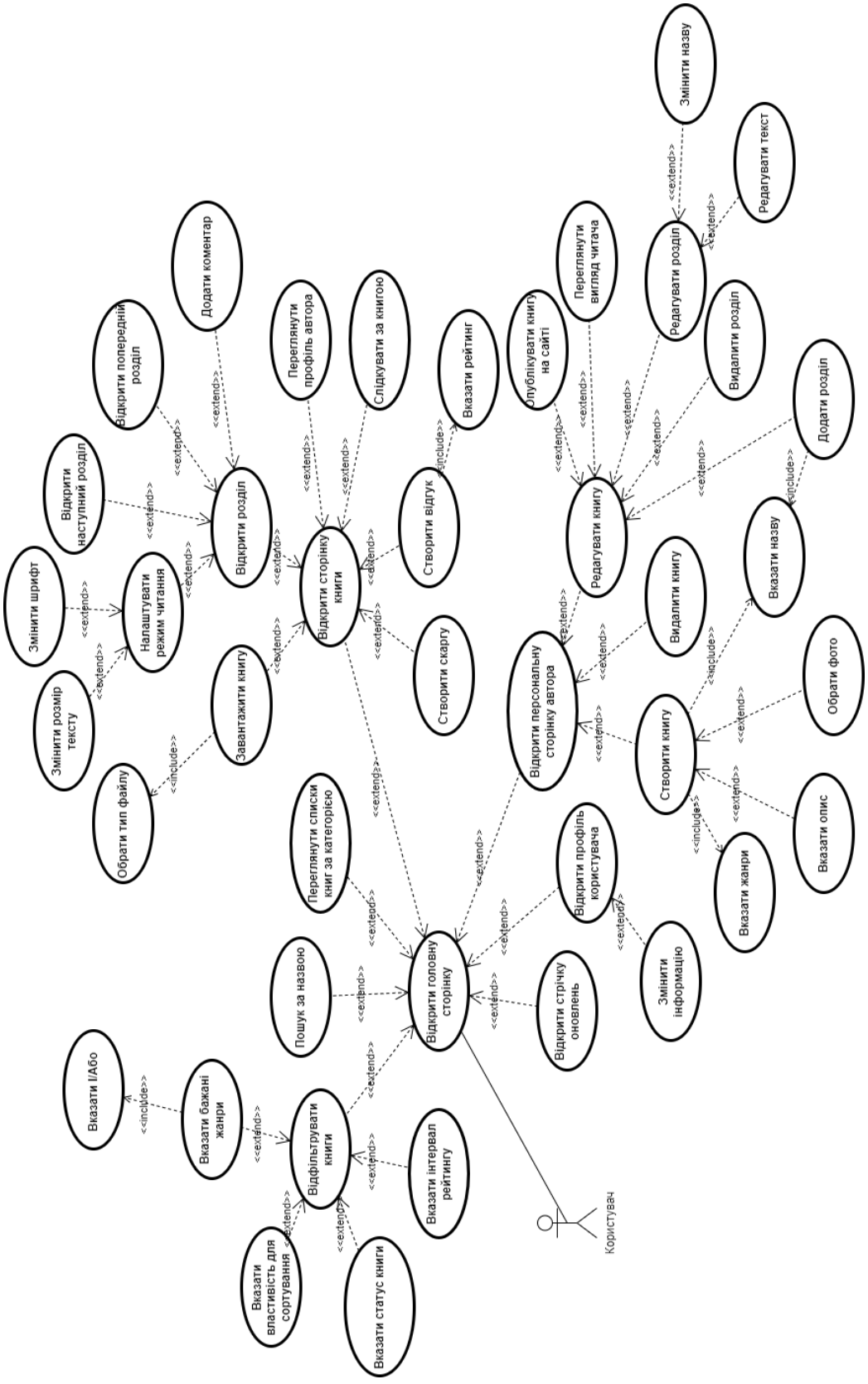
1. ДСТУ 7448:2013. Бібліотечно-інформаційна діяльність. Терміни та визначення понять. [Чинний від 2013-11-29]. Вид. офіц. Київ, 2014. – 45 с.
2. Назаровець С. Бібліотека 4.0: технології та сервіси майбутнього: [передові інтернет-технології для впровадження інноваційних бібліотечних послуг і сервісів, скерованих на задоволення потреб користувачів] / С. Назаровець, Є. Кулик // Бібліотечний вісник. – 2017. – № 5. – С. 3–14.
3. Пасмор Н. П. Електронні бібліотеки як елемент інформаційного суспільства: консенсусні рішення [Електронний ресурс] / Н. П. Пасмор // Імперативи розвитку електронних бібліотек: pro et contra = Imperatives of the Electronic Libraries Development: pro et contra : матеріали міжнар. веб-конф., Харків, 27 берез. 2014 р. – Електрон. текстові дані. – Харків, 2014. – Режим доступу до ресурсу: http://www.nbuv.gov.ua/sites/default/files/method_mg/mfiles/201410_method/imperatives.pdf
4. Про схвалення «Концепції Державної цільової національно-культурної програми створення інформаційної бібліотечної системи «Бібліотека - XXI»: Розпорядження Кабінету Міністрів України від 23.12.2009 р. № 1579-р [Текст] // Уряд. кур'єр. - 2010. – 6 січня. – С. 15-16
5. E-book Market Size & Share Analysis - Industry Research Report - Growth Trends [Електронний ресурс] – Режим доступу до ресурсу: https://www.mordorintelligence.com/industry-reports/e-book-market.
6. Женченко М. Цифрова дистрибуція на книжковому ринку України / М. Женченко // Вісник Книжкової палати. – 2013. – №10. – С. 3-5 Режим доступу до ресурсу: http://nbuv.gov.ua/UJRN/vkp_2013_10_2
7. Про авторське право і суміжні права [Електронний ресурс]: Закон України від 01.12.2022 р. № 2811-IX : станом на 15 квіт. 2023 р. – Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/2811-20#Text

8. Бібліотека української літератури УкрЛіб [Електронний ресурс] // Бібліотека української літератури УкрЛіб. – Режим доступу до ресурсу: https://www.ukrlib.com.ua/
9. Wattpad – Де живуть історії [Електронний ресурс] – Режим доступу до ресурсу: https://www.wattpad.com/?locale=uk_UA.
10. Goodreads [Електронний ресурс] – Режим доступу до ресурсу: https://www.goodreads.com/
11. Project Gutenberg [Електронний ресурс] – Режим доступу до ресурсу: https://www.gutenberg.org/.
12. 25 Interesting Wattpad Statistics and Facts [Електронний ресурс] – Режим доступу до ресурсу: https://expandedramblings.com/index.php/wattpad-statistics-facts/.
13. 10 Interesting Goodreads Facts and Statistics [Електронний ресурс]. – Режим доступу до ресурсу: https://expandedramblings.com/index.php/goodreads-facts-and-statistics/.
14. Microsoft Learn: Overview of ASP.NET Core [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-7.0.
15. What is REST - REST API Tutorial [Електронний ресурс] – Режим доступу: https://restfulapi.net/.
16. Hughes A. What is Microsoft SQL Server? A definition from WhatIs.com [Електронний ресурс] / Adam Hughes, Craig Stedman // Data Management. – Режим доступу до ресурсу: https://www.techtarget.com/searchdatamanagement/definition/SQL-Server.
17. Smith J. P. Entity Framework Core in Action, Second Edition / Jon P. Smith. – [S. l.] : Manning Publications Co. LLC, 2021. – 624 с.
18. AutoMapper [Електронний ресурс]. – Режим доступу до ресурсу: https://automapper.org/.

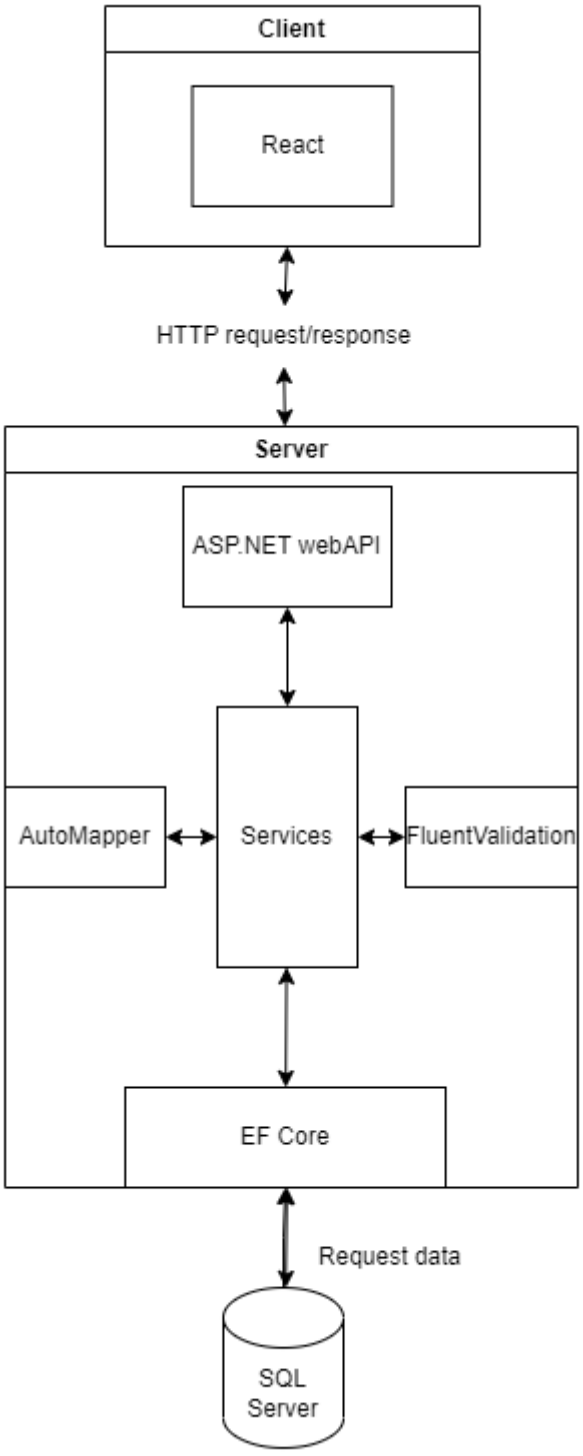
19. FluentValidation – FluentValidation documentation [Електронний ресурс]. – Режим доступу до ресурсу: https://docs.fluentvalidation.net/en/latest/.
20. Anthony A. Fullstack React: The Complete Guide to ReactJS and Friends / Accomazzo Anthony, Murray Nathaniel, Lerner Ari. – [S. l.] : Fullstack.io, 2017. – 836 с.
21. Bulma: Free, open source, and modern CSS framework based on Flexbox [Електронний ресурс]. – Режим доступу до ресурсу: https://bulma.io/.
22. JWT.IO - JSON Web Tokens Introduction [Електронний ресурс]. – Режим доступу до ресурсу: https://jwt.io/introduction.
23. Quill - Your powerful rich text editor [Електронний ресурс] // Quill. – Режим доступу до ресурсу: https://quilljs.com/.

ДОДАТКИ

Додаток А. Use-case діаграма підсистеми користувача



Додаток Б. Архітектура застосунку



Додаток В. Діаграма бази даних застосунку