МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри
Ігор ШЕЛЕХОВ
(підпис)
червня 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавр

зі спеціальності 122 – Комп'ютерних наук,
освітньо-професійної програми «Інформатика»
на тему: «Веб-орієнтована інформаційно-пошукова система електронного каталогу книг»
Здобувача групи ІН-91 Безверхого Миколи Івановича

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Микола БЕЗВЕРХИЙ
(підпис)

Керівник,
асистент кафедри комп'ютерних наук,
кандидат фізико-математичних наук Олександр ВЛАСЕНКО
(підпис)

Суми – 2023

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавра

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувача групи ІН-91 Безверхого Миколи Івановича

1. Тема роботи: «Веб-орієнтована інформаційно-пошукова система електронного каталогу книг» затверджую наказом по СумДУ від _____
2. Термін здачі здобувачем кваліфікаційної роботи до 09 червня 2023 року
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Аналіз проблеми предметної області, постановка й формування завдань дослідження. 2) Огляд сучасних технологій та підходів, що використовуються для проєктування веб-орієнтованих інформаційних систем. 3) Програмна реалізація інформаційної системи каталогу книг. 4) Аналіз отриманих результатів.*
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) _____
6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «____» _____ 20 ___ р.

Завдання прийняв до виконання _____ (підпис) Керівник _____ (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз проблеми предметної області, постановка й формування завдань дослідження*		
2	*Огляд сучасних технологій та підходів, що використовуються для проєктування веб-орієнтованих інформаційних систем*		
3	*Програмна реалізація інформаційної системи каталогу книг*		
4	*Аналіз отриманих результатів*		
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*		

Здобувач вищої освіти _____ (підпис) Керівник _____ (підпис)

АНОТАЦІЯ

Записка: 79 стр., 19 рис., 1 додаток, 11 використаних джерел.

Обґрунтування актуальності теми роботи – Тема кваліфікаційної роботи є актуальною, оскільки інформаційно-пошукова система електронного каталогу книг є потрібною та важливою для читачів. Вона надає їм можливість знайти необхідну інформацію про книги швидко та просто.

Об'єкт дослідження – процес розробки веб-орієнтованої інформаційно-пошукова системи електронного каталогу книг.

Мета роботи – проектування та розробка веб-орієнтованої інформаційно-пошукової системи електронного каталогу книг за допомогою сучасних технологій.

Методи дослідження – інструменти для створення функціональної клієнтської частини, інструменти для ефективного керування та зберігання даних, інструменти, спрямовані на розробку веб-орієнтованих інформаційних систем.

Результати – розроблено та реалізовано інформаційну систему управління книгами, яка надає функціональність додавання, перегляду, пошуку, зміни та оцінювання книг. Система забезпечує ефективний доступ до інформації про книги, полегшує пошук за різними критеріями та забезпечує зручний спосіб управління літературою.

ІНФОРМАЦІЙНА СИСТЕМА, ПОШУК КНИГ, ASP.NET 6, ANGULAR, RxJS, OpenAPI, EF CORE, MICROSOFT SQL SERVER

ЗМІСТ

ВСТУП

Книги завжди відігравали важливу роль у житті людей. Вони є цінним джерелом знань та спонукають нас до вивчення нового. Вони є путівниками у світі знань і досвіду, які людина починає засвоювати ще зі шкільних років. Крім передачі інформації, книги також надають можливість насолодитись захоплюючими пригодами та цікавими історіями. Коли ми занурюємося у світ, описаний на сторінках книги, ми розширюємо свої горизонти, розуміємо себе краще та отримуємо нові знання.

З розвитком технологій, формат книг також зазнав змін. Зараз ми маємо можливість отримати доступ до книг в онлайн-форматі через Інтернет. Це надає широкі можливості для всіх охочих зробити свій внесок у літературу і поділитись своїми творами зі світом. Однак, разом зі зростанням доступності книжкового матеріалу, виникає проблема пошуку потрібної книги серед безлічі доступних варіантів. Велика кількість книжок в онлайн-середовищі може ускладнити процес знаходження потрібного твору і спричинити втрату часу та змішані почуття у читачів.

Ця проблема потребує вирішення. Метою даної роботи є використання сучасних технологій для полегшення та прискорення пошуку книг в онлайн-середовищі. Шляхом розробки ефективних та інтуїтивно зрозумілих інструментів пошуку, ми надаємо людям зручний спосіб знайти потрібну книгу серед великого обсягу доступної літератури. Сучасні технології можуть бути використані для створення алгоритмів фільтрації та сортування книг з урахуванням індивідуальних потреб і вподобань користувачів. Це допоможе забезпечити більш точні й персоналізовані результати пошуку, зменшити час на пошук та покращити загальний досвід користувачів у віртуальному світі книжок.

1. ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Загальна будова веб-застосунку

Архітектура веб-застосунку представляє собою основну структуру, яка визначає взаємодію компонентів застосунку, проміжного програмного забезпечення, інтерфейсів користувача та баз даних[1].

На найвищому рівні у застосунку можна виділити три основні компоненти, які взаємодіють між собою:

1. Клієнтська частина (фронтенд, клієнт):

Цей компонент зазвичай є веб-браузером, який відповідає за відображення інтерфейсу користувача і взаємодію з ним, тобто є компонентом з яким працює користувач безпосередньо. Веб-браузер отримує вхідні дані, відправляє запити до сервера й обробляє отримані відповіді. Також він відповідає за валідацію вхідних даних перед відправкою до сервера, але це лише для зручності користувача, оскільки клієнтом може виступати не лише браузер. Тобто за для безпеки на сервері також повинна бути присутня валідація.

2. Серверна частина (бекенд):

Серверна частина включає веб-сервер, який виконує важливі функції у веб-застосунку: обробка запитів від клієнтської частини (клієнта), взаємодію з базою даних та іншими зовнішніми сервісами, повернення результату клієнту.

Після надсилання запиту клієнтом, сервер приймає цей запит і розпочинає обробку. Веб-сервер виконує бізнес-логіку застосунку, що включає обчислення, перевірку доступу та взаємодію з іншими компонентами системи. Одним з важливих аспектів веб-сервера є маршрутизація запитів. Він визначає, який компонент або модуль системи буде обробляти конкретний запит. Це дозволяє розподілити завдання між різними частинами системи та забезпечити оптимальну обробку запитів.

Веб-сервер взаємодіє з базою даних для доступу до важливих даних для застосунку. Він виконує запити до бази даних для внесення чи оновлення інформації або отримання результатів необхідних клієнту.

Окрім цього, веб-сервер може взаємодіяти з іншими зовнішніми сервісами, такими як служби аутентифікації, платіжні системи або сервіси доставки повідомлень. Це дозволяє розширити функціональність застосунку та використовувати зовнішні ресурси для задоволення потреб користувачів.

Усі ці операції веб-сервера відбуваються відокремлено від клієнтської частини, що робить його незалежним компонентом системи. Це забезпечує гнучкість та масштабованість веб-застосунків, оскільки різні частини системи можуть працювати незалежно одна від одної й розгортатися на різних середовищах.

3. Сервер бази даних:

Сервер баз даних відіграє важливу роль у веб-застосунку, забезпечуючи збереження та доступ до даних, які використовуються в системі. Він приймає запити від веб-сервера та обробляє їх, надаючи необхідні дані. Крім того, він може виконувати бізнес-логіку, використовуючи збережені процедури, що зазвичай підвищує швидкість застосунку в цілому.

Одним із ключових аспектів роботи сервера баз даних є забезпечення цілісності даних, тобто збереження їх у правильному стані й захист від втрати або пошкодження. Він забезпечує механізми резервного копіювання та відновлення даних, а також реалізує механізми контролю доступу до даних для забезпечення конфіденційності та безпеки.

Ці компоненти співпрацюють, щоб забезпечити функціональність та ефективність веб-застосунку. Клієнтська частина взаємодіє з серверною частиною, відправляючи запити та отримуючи відповіді, а серверна частина виконує бізнес-логіку та здійснює доступ до даних, необхідних для задоволення запитів користувача.

1.2 Огляд архітектурних підходів до побудови веб-застосунку

Архітектура застосунку є важливою складовою, оскільки саме вона диктує модель поведінки для всіх компонентів застосунку. Серед архітектур веб-застосунків найпоширенішими є монолітна, мікросервісна та безсерверна[2].

1.2.1 Монолітна архітектура

Монолітна архітектура є стандартним підходом до розробки програмного забезпечення, де весь код розробляється як один великий блок. У цій моделі всі компоненти програми взаємозалежні і взаємопов'язані, і вони спільно працюють, утворюючи єдину програму.

Переваги монолітної архітектури включають простоту розробки, тестування та розгортання проекту, оскільки увесь код буде виконуватися як одна програма на одному сервері чи віртуальній машині. Коли розмір проекту невеликий, то управління та підтримка кодової бази є простою задачею.

Однак, монолітна архітектура має свої недоліки. Зі зростанням розміру проекту стає складніше керувати та розширювати систему. Навіть невеликі зміни можуть вимагати великого обсягу роботи, оскільки це може вплинути на інший функціонал. Більш того, монолітна архітектура не забезпечує гнучкість масштабування, оскільки всі компоненти пов'язані між собою, і масштабування окремих частин системи перетворюється у складне завдання. Також страждає і надійність, оскільки якщо в одному компоненті відбувається збій, то це може призвести до відмови всього застосунку. Наприклад, якщо при оновленні даних в базу занесли не валідні значення через помилку в компоненті, що відповідає за оновлення, то це може вплинути на всю систему, яка ці дані використовує.

Загалом, монолітна архітектура підходить для простих і невеликих проектів. Однак, при розробці більш складних та масштабованих систем рекомендується розглянути альтернативні архітектурні підходи.

1.2.2 Мікросервісна архітектура

Архітектура мікросервісів вирішує деякі проблеми, що виникають у монолітних додатках. У мікросервісній архітектурі код розробляється як набір слабко пов'язаних незалежних сервісів, які взаємодіють через RESTful API[3], тобто сервіси, за необхідності, спілкуються між собою за допомогою HTTP . Кожен сервіс зазвичай містить свою власну базу даних (можлива і спільна база даних для різних сервісів) і керується власною бізнес-логікою, що дає змогу легко розробляти та розгортати окремі компоненти.

Гнучкість є однією з основних переваг мікросервісної архітектури, оскільки незалежність сервісів один від одного дозволяє оновлювати певний сервіс без впливу на інші. Масштабування також стає простішим, оскільки можна горизонтально масштабувати лише окремі сервіси, які вимагають більшої потужності, а для монолітного застосунку масштабувати окремі частини неможливо. Мікросервіси також забезпечують зручну адаптацію до інноваційних технологій і швидку реакцію на зміни вимог. Це відбувається за рахунок того, що новий сервіс можна створити використовуючи найновітніші технології, бо він повністю ізольований від інших сервісів.

1.2.3 Безсерверна архітектура

Безсерверна архітектура (serverless) – це модель розробки програмних додатків, в якій управління базовою інфраструктурою здійснюється постачальником інфраструктурних послуг. Ця модель дозволяє платити лише за фактичне використання інфраструктури під час роботи додатка, тобто коли ресурс не використовується, то за нього не потрібно платити. Безсерверне обчислення дозволяє ефективно використовувати ресурси тільки під час виконання додатка, а завдання масштабування автоматично обробляються провайдером хмарних послуг. Крім того, використання безсерверної архітектури

спрощує розробку бекенду, що призводить до зниження зусиль, витрат та прискорення розробки.

Безсерверна архітектура означає, що ви розробляєте серверну частину веб-застосунку, але не маєте прямого контролю над інфраструктурою, що його підтримує. Замість того, весь необхідний код, який виконує бізнес-логіку вашого застосунку, надсилається провайдеру інфраструктурних послуг. Цей провайдер відповідає за запуск і виконання цього коду на вимогу, забезпечуючи масштабованість та високу доступність вашого застосунку. Ви зосереджуєтесь на розробці функціональності, не витрачаючи час і зусилля на управління інфраструктурою.

1.3 Огляд архітектурних підходів для побудови клієнтської частини

1.3.1 Односторінковий застосунок (Single-page application)

Односторінкові застосунки представляють собою веб-застосунки, у яких зазвичай контент і логіка завантажуються один раз при першому відкритті сторінки, а далі взаємодія з додатком відбувається без повного перезавантаження сторінок. Замість завантаження нових сторінок з сервера, SPAs завантажують лише необхідні дані і змінюють вміст на поточній сторінці, що забезпечує більш швидку та плавну взаємодію для користувача.

JavaScript використовується як основна мова програмування для розробки SPAs. Він дозволяє динамічно оновлювати контент, взаємодіяти з сервером за допомогою AJAX-запитів та реагувати на події користувача. JavaScript також має велику кількість бібліотек і фреймворків, які спрощують розробку SPAs, наприклад, React, Angular та Vue.js.

Загалом, SPAs дозволяють створювати більш динамічні та інтерактивні веб-застосунки, забезпечують зручну навігацію, швидку відповідь і покращений користувацький досвід. Цей тип архітектури клієнтів став популярним через свою

ефективність, що забезпечує більш високу продуктивність та задоволення користувачів.

1.3.2 Серверний рендеринг (Server-Side Rendering)

SSR (Server-Side Rendering) – це процес відтворення веб-сайту, побудованого з використанням клієнтського фреймворку, у HTML та CSS на сервері.. Це дозволяє швидко доставляти основні елементи сайту і поліпшує швидкість його завантаження в браузері користувача.

При використанні SSR, сервер обробляє всі дані і генерує новий HTML-документ для кожного запиту. Коли браузер отримує цей HTML-документ разом із CSS, він може відображати вміст сторінки без очікування завантаження Javascript. Це допомагає зменшити час, необхідний для відображення сторінки і поліпшує загальний досвід користувача.

Основна перевага SSR полягає в тому, що користувачі швидше отримують вмісту сайту, оскільки важливі ресурси передаються без зайвих затримок, в односторінковому застосунку доведеться чекати на завантаження всіх скриптів, які відображатимуть сторінку. Крім того, це сприяє поліпшенню індексації сайту пошуковими системами, що може позитивно позначитися на SEO.

1.4 Постановка задачі

Мета: Розробити систему управління книгами, що дозволяє користувачам додавати книги, переглядати їх інформацію, здійснювати пошук за назвою, жанром, автором чи рейтингом, змінювати книги (лише авторизовані користувачі з відповідними ролями), оцінювати книги.

Кроки для досягнення мети:

1. Розробити базу даних для зберігання інформації про книги. База даних має містити наступні поля: назва книги, жанр, опис, автор.

2. Створити функціональні можливості для авторизації користувачів. Користувачі повинні мати можливість створити обліковий запис і входити в систему з використанням свого імені користувача та пароля.

3. Реалізувати можливість додавання нових книг до системи. Користувачі повинні мати можливість ввести назву книги, жанр, опис та інформацію про автора. При додаванні книги, система повинна зберігати цю інформацію в базі даних.

4. Розробити функціонал для перегляду списку книг. Користувачі повинні мати можливість переглянути загальний список всіх доступних книг, включаючи назву, жанр, автора та рейтинг.

5. Реалізувати можливість пошуку книг за назвою, жанром або автором. Користувачі повинні мати можливість ввести критерії пошуку і отримати відповідний список книг, що задовольняють ці критерії.

6. Розробити функціонал для зміни інформації про книги. Тільки авторизовані користувачі з відповідними ролями повинні мати можливість змінювати інформацію про книгу, таку як назва, жанр, опис або автор.

7. Реалізувати можливість оцінювання книг, використовуючи числову шкалу. Оцінка повинна бути збережена в базі даних та повинна враховуватись при подальшому пошуку за рейтингом.

8. Розробити функціонал для детального перегляду книги. Користувачі повинні мати можливість переглянути всю інформацію про книгу, включаючи назву, жанр, опис, автора, рейтинг та оцінки користувачів.

9. Реалізувати можливість пошуку книг за рейтингом. Користувачі повинні мати можливість вказати мінімальний рейтинг і отримати список книг, які мають рейтинг, що перевищує вказане значення.

Загалом, реалізація цієї постановки задачі дозволить створити систему управління книгами з функціоналом додавання, перегляду, пошуку, зміни та оцінювання книг, що задовольняє потреби користувачів у зручному способі управління літературою.

2. ВИБІР МЕТОДІВ РОЗВ'ЯЗАННЯ ЗАДАЧІ

2.1 Архітектура застосунку

2.1.1 Серверна частина

Для розробки системи управління книгами буде обрана архітектура, яка має схожі риси з монолітною архітектурою, але з деякими модифікаціями. У цій архітектурі буде використана єдина база даних, один сервер, але клієнтська частина буде окремо від сервера.

Для забезпечення безпеки та автентифікації користувачів буде використано JWT (JSON Web Token)[4]. JWT токен дозволить автентифікувати користувачів та забезпечити безпеку передачі даних між клієнтом і сервером. Це забезпечить захист інформації та контроль доступу до ресурсів системи.

Використання JWT токена також дозволить зробити майбутню міграцію на мікросервісну архітектуру більш простою. За потреби можна додати окремий сервіс до поточного застосунку, і використання JWT токена дозволить забезпечити автентифікацію та безпеку між новим сервісом та існуючими компонентами системи.

Ця архітектура покращить безпеку системи та зробить її більш гнучкою для майбутнього розвитку та розширення.

JSON Web Token (JWT) – це спосіб безпечно передавати інформацію між сторонами за допомогою JSON-об'єкта. Він містить дані, цілісність яких можна перевірити. JWT може бути підписаний за допомогою секретного ключа або публічного/приватного ключа.

При автентифікації в системі, клієнту надсилається спеціальний токен (JWT), який містить інформацію про ідентичність користувача. Цей токен підписується за допомогою секретного ключа або пари публічного/приватного

ключа. Таким чином, коли клієнт надсилає виданий токен на сервер, сервер може перевірити підпис, щоб переконатися, що користувач дійсно автентифікований, має право на запитану інформацію і дані не були змінені.

За допомогою JWT можна надійно передавати інформацію між різними компонентами системи, не хвилюючись про цілісність даних. Це дозволяє забезпечити безпеку і надійність взаємодії між різними компонентами системи.

Для розробки буде обраний фреймворк ASP.NET Core [5].

ASP.NET Core – високопродуктивний та кросплатформений фреймворк, який розроблений для створення сучасних додатків, що можуть працювати в хмарному середовищі. Він є відкритим і має широкі можливості для розробки веб-додатків, мобільних бекендів та додатків для "Інтернету речей" (IoT).

ASP.NET Core надає розробникам гнучкість при виборі інструментів, що дозволяє використовувати різні технології на різних операційних системах, таких як Windows, macOS та Linux. Це робить фреймворк привабливим для широкої аудиторії. Також ASP.NET Core дозволяє розгортати додатки як в хмарному середовищі, так і на власних серверах. Це дає можливість розробникам обирати спосіб залежно від своїх потреб [6].

ASP.NET Core працює на платформі .NET Core, яка є модульною і легкою системою для розробки додатків. Вона надає високу продуктивність, масштабованість та безпеку, що робить фреймворк ідеальним вибором для розробки сучасних додатків.

Для зберігання даних буде використано SQL Server [7].

SQL Server – це система управління реляційними базами даних, яка розроблена компанією Microsoft. Вона призначена для зберігання та управління великими обсягами даних. SQL Server був створений з метою забезпечити надійну та ефективну роботу з базами даних та конкурує з іншими популярними системами, такими як MySQL та Oracle Database.

Одна з ключових особливостей SQL Server - підтримка стандарту ANSI SQL, що є уніфікованою мовою запитів до баз даних, але також в SQL Server наявна своя власна реалізація мови SQL - T-SQL (Transact-SQL), яка є розширеною версією SQL. T-SQL дозволяє використовувати змінні, обробляти винятки, створювати збережені процедури та багато іншого, що спрощує роботу з базою даних.

SQL Server Management Studio (SSMS) є основним інтерфейсом для роботи з SQL Server. Це потужний інструмент, який надає зручне середовище для розробки, адміністрування та оптимізації баз даних. SSMS підтримує як 32-бітні, так і 64-бітні операційні системи, що дає можливість використовувати його на різних платформах.

Доступ до бази даних буде відбуватися за допомогою Entity Framework Core, який використовується для забезпечення доступу до бази даних через об'єктно-реляційне відображення. Це означає, що з EF Core можна працювати з даними у базі, використовуючи об'єкти мови програмування. EF Core зменшує кількість коду, який потрібно писати для доступу до даних.

EF Core працює з моделлю даних, яка складається з класів сутностей та контексту, який представляє відображення моделі в базу даних. За допомогою контексту можна оперувати даними в базі даних. Також, завдяки цьому, EF Core може створювати автоматично міграції. Він порівнює поточну модель з моделлю, яка є в міграціях, і автоматично створює код, який необхідно виконати для того, щоб оновити базу до поточної моделі.

2.1.2 Клієнтська частина

Для розробки односторінкового (SPA) застосунку буде використаний фреймворк Angular [8].

Angular є потужним інструментом для проектування та розробки веб-додатків, який дозволяє створювати ефективні та складні односторінкові

програми. Він надає широкий набір можливостей для роботи зі структурою застосунку, управлінням станом, маршрутизацією та взаємодією з сервером. З допомогою Angular можна розробити сучасний та динамічний SPA, який забезпечить швидку та зручну взаємодію з користувачем.

Angular базується на ідеї компонентної архітектури, де додаток будується з окремих компонентів. Компоненти є самодостатніми, багатовикористовуваними блоками, що дозволяє структурувати додаток і полегшує його. Angular надає потужний шаблонний двигун, який дозволяє створювати динамічні та інтерактивні інтерфейси. За допомогою директив можна контролювати поведінку та вигляд компонентів. Цей фреймворк має вбудований механізм маршрутизації, який дозволяє переходити між різними сторінками або компонентами у додатку.

Angular підтримує реактивне програмування за допомогою бібліотеки RxJS [9]. Реактивність дозволяє зручно працювати з асинхронними операціями, подіями та потоками даних, спрощуючи керування станом застосунку.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Серверна частина

Створивши проєкт у Microsoft Visual Studio за шаблоном ASP.Net Core API, потрібно сконфігурувати сервер для використання необхідних модулів. Додамо необхідну конфігурацію в класі Startup для сервісів.

Для початку, ми створимо оголосимо контекст бази даних, який використовуватиметься для маніпуляції даними. Також ми використаємо вбудовану систему Identity в .NET Core, яка буде зберігати дані користувачів в цьому контексті EF Core.

```
services.AddDbContext<ApplicationDbContext>(options =>
    options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

services.AddIdentity<ApplicationUser, IdentityRole>()
    .AddEntityFrameworkStores<ApplicationDbContext>();
```

Рисунок 3.1 – Конфігурація бази та системи Identity

Ця вбудована система створить систему користувачів з ролями, необхідні для застосунку.

Також, після конфігурації системи автентифікації, додамо політику авторизації, яка дозволятиме доступ лише за наявності ролі адміністратора. Ця політика легко дозволить обмежувати доступ до ендпоінтів доступних лише для адміністраторів системи.

Створимо фільтр помилок, який буде в залежності від типу помилки обробляти її та повертати результат користувачу. Для цього створимо клас FilterExceptionMiddleware, який повинен містити стандартні поля для проміжного програмного забезпечення і додамо логіку для обробки помилок. Виводити помилки будемо задопомогою логера, що дозволить в майбутньому легко змінити

місце для логів, наприклад файл, консоль, сторонній сервіс для логів. Передати логер для поточного класу можна завдяки впровадженню залежностей.

```
public class FilterExceptionMiddleware
{
    private readonly RequestDelegate _next;
    private readonly ILogger<FilterExceptionMiddleware> _logger;

    public FilterExceptionMiddleware(RequestDelegate next, ILogger<FilterExceptionMiddleware> logger)
    {
        _next = next;
        _logger = logger;
    }

    public async Task InvokeAsync(HttpContext context)
    {
        try
        {
            await _next(context);
        }
        catch (Exception e)
        {
            _logger.LogError(e, "An error occurred");
```

Рисунок 3.2 – Фільтр помилок

Фільтр помилок необхідно додати до пайплайну якраз перед контролерами.

У контролерах виконується бізнес-логіка, яка може видавати помилки, тому їх ловитеме і оброблюватиме створений фільтр помилок.

Створимо моделі, якими будемо оперувати. Для цього потрібно створити сутності «Книга», «Жанр», «Оцінка», «Користувач» та «Роль» і додати взаємозв’язки між ними. Загальна схема сутностей відображена на рисунку 3.3.

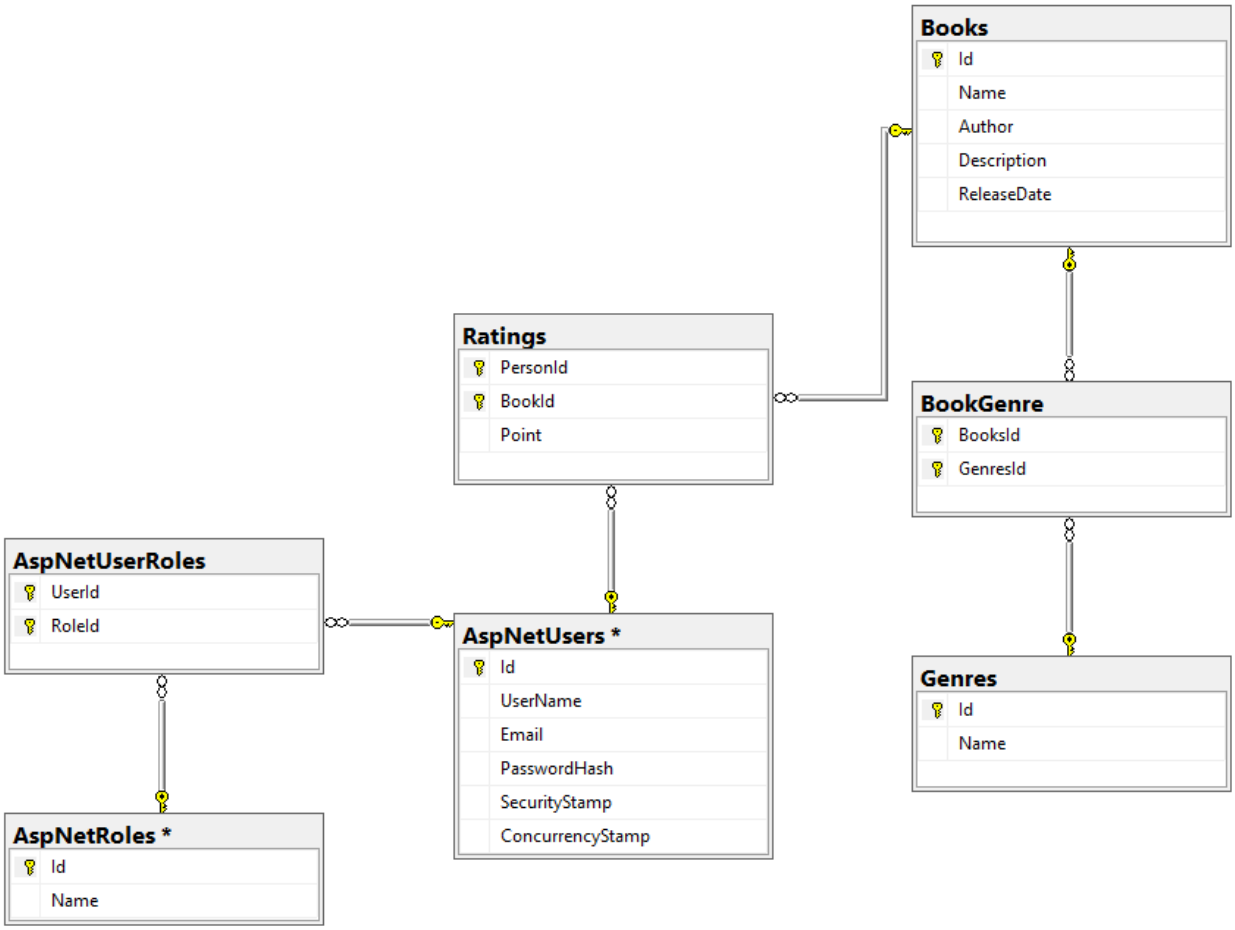

Рисунок 3.3 – Схема сутностей в базі даних

Створимо контекст бази даних ApplicationDbContext, який має розширювати IdentityDbContext – контекст, який містить оголошення користувачів з ролями. Оскільки оперуємо даними завдяки EF Core, то необхідно в контест додати множини сутностей(рис. 3.4).

```
public class ApplicationDbContext : IdentityDbContext<ApplicationUser>
{
    public DbSet<Genre> Genres { get; set; }
    public DbSet<Book> Books { get; set; }
    public DbSet<Rating> Ratings { get; set; }
```

Рисунок 3.4 – Контекст бази даних

Наступним кроком створимо репозиторії для кожної сутності. Це допоможе розділити логіку збереження даних і логіку застосунку. Реалізуємо всі методи необхідні для операцій з даними. Для отримання всіх книг посторінково необхідно повернути загальну кількість та кілька книг, які потрібні для відображення на поточній сторінці.

Для репозиторія створимо загальний інтерфейс, який диктуватиме поведінку всім репозиторіям(рис. 3.5).

```
public interface IRepository<T, K>
{
    Task<(List<T> data, int? totalCount)> List(Expression<Func<T, bool>> query = null, int skip = 0, int take = 0);
    Task<T> GetById(K id);
    void Add(T entity);
    void Update(T entity);
    Task Remove(K id);
    Task<int> SaveChangesAsync();
}
```

Рисунок 3.5 – Інтерфейс репозиторію

Інтерфейс репозиторію є шаблонним, тобто на вхід приймає тип сутності, якою оперує та тип головного ключа цієї сутності.

Для реалізації репозиторію для книг необхідно додати ще один інтерфейс репозиторію книг «IBookRepository». Це дозволить в майбутньому за потреби розширити функціонал певного репозиторію не змінюючи інтерфейс загального репозиторію(рис. 3.5).

Аналогічні репозиторії створюємо для решти сутностей.

Тепер необхідно створити сервіси для сутностей, які виконуватимуть елементарні операції зі створення, видалення, оновлення та читання даних. Додатковий шар сервісів дозволяє розмежувати логіку і дозволяє легко її перевикористати за нагоди в майбутньому.

Створимо AuthService – сервіс, яки відповідатиме за аутентифікацію користувачів.

Він має методи:

1. SignUp – реєстрація нових користувачів.
2. SignIn – аутентифікація зареєстрованих користувачів.
3. SignOut - вихід з аккаунта

Створимо BookService – сервіс, яки відповідатиме за керування книгами.

Він має методи:

1. List – повертає список всіх книг.
2. Get – повертає книгу, яку знаходить за ідентифікатором.
3. Save – створює нову книгу.
4. Update – оновлює дані про існуючу книгу.
5. Delete – видаляє книгу.
6. RateBook – дозволяє користувачу оцінити книгу.

Створимо GenreService – сервіс, яки відповідатиме за керування книгами.

Він має методи:

1. List – повертає список всіх жанрів.
2. Get – повертає жанр, який знаходить за ідентифікатором.
3. Save – створює новий жанр.
4. Update – оновлює дані про існуючий жанр.
5. Delete – видаляє жанр.

У сервісі для книг для оцінки виконується проста логіка. Маючи ідентифікатор книги та ідентифікатор користувача можна знайти чи створити сутність «Оцінка». Дана сутність має композитний ключ, тобто запис може вважатися унікальним якщо більше немає запису з такою ж парою ідентифікаторів користувача та книги. Це гарантує, що жоден користувач не зможе оцінити свою улюблену книгу двічі для підняття рейтингу. Для реалізації оцінки книги в сервісі необхідно спочатку спробувати знайти наявну сутність «оцінка» для заданого композитного ідентифікатора. Якщо він існує, то необхідно оновити його, встановивши нове значення оцінки. Якщо його не існує, то необхідно створити нову сутність з вказаним значенням.

Перейдемо на найвищий рівень до створення контролерів, які оброблятимуть вхідні дані.

При створенні контролера для книг, на весь контролер доданий атрибут авторизації. На методи якии потрібен доступ адміністратора доданий атрибут авторизації, який працює за політикою «HasAccessToConfiguration», визначену при конфігурації в класі Strartup(рис. 3.1).

Додавання атрибута авторизації на весь контролер є кращою практикою з точки зору безпеки, оскільки це гарантує, що жоден метод не буде пропущений. Для публічних методів, які не вимагають захисту, використовується атрибут «AllowAnonymous», який вимикає авторизацію лише для цих конкретних методів у контролері.

Також необхідно розробити функціонал для оцінки книг. Для цього створимо ще один метод в «BookController». Цей метод буде доступний лише для авторизованих користувачів. На вхід він буде отримувати ідентифікатор книги та оцінку. Ідентифікатор користувача необхідно взяти з інформації про авторизованого поточного користувача, який робить цей запит. Такий підхід є більш безпечним, оскільки він гарантує, що кожен користувач може оцінити книги лише від свого імені.

Реалізуєму схему методів контролера з авторизаційними атрибутами (рис.3.6).

```
[Authorize]
[Route("api/[controller]")]
[ApiController]
public class BookController : ControllerBase
{
    [AllowAnonymous]
    [HttpGet]
    public async Task<ServiceResult<List<Book>>> List([FromQuery] BookFilter filter);

    [AllowAnonymous]
    [HttpGet("{id}")]
    public async Task<ServiceResult<Book>> Get(Guid id);

    [Authorize(Policy = "HasAccessToConfiguration")]
    [HttpPost]
    public async Task<ServiceResult<Book>> Save([FromBody] Book book);

    [Authorize(Policy = "HasAccessToConfiguration")]
    [HttpPut]
    public async Task<ServiceResult<Book>> Update([FromBody] Book book);

    [Authorize(Policy = "HasAccessToConfiguration")]
    [HttpDelete("{id}")]
    public async Task Delete(Guid id);


    [HttpPost("rating")]
    public async Task<ActionResult> Rate([FromBody] BookRate rate);
}
```

Рисунок 3.6 – Схема контролера для книг

Створивши всі необхідні файли для серверної частини матимему структуру файлів та папок зображену на рис. 3.7.

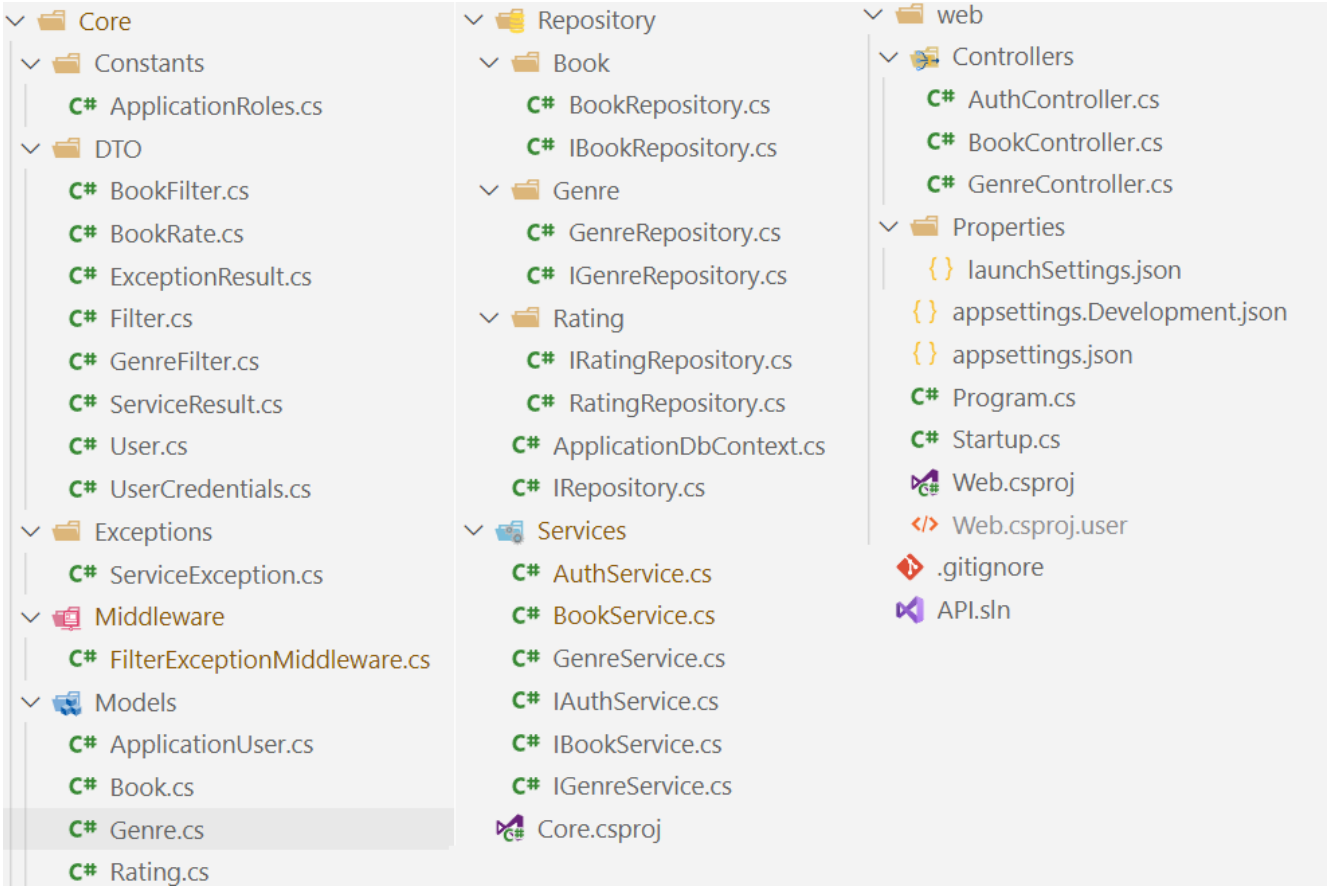

Рисунок 3.7 – Структура файлів та папок серверної частини

3.2 Клієнтська частина

Для початку необхідно встановити Angular CLI.

Angular CLI – це інструмент командного рядка (Command-Line Interface, CLI) [10], який дозволяє ініціалізувати, розробляти, створювати шаблони та підтримувати Angular-додатки безпосередньо з командного рядка.

Для встановлення необхідно виконати команду «npm install -g @angular/cli». Тепер можна створити проєкт, використовуючи щойно встановлений Angular CLI. Для цього необхідно виконати команду «ng new client» і перейти в створену директорію.

Тепер перейдемо до створення сторінки, яка відображатиме список всіх книг. У структурі Angular розміщення компонентів у різних модулях має

додаткові переваги. Поділ компонентів на окремі модулі допомагає полегшити розробку та підтримку проєкту, особливо коли масштабування стає фактором.

Одна з основних переваг використання різних модулів полягає в розділенні відповідальності та логіки. Кожен модуль може бути відповідальним за певну частину функціональності додатка або групу компонентів, що спрощує управління та розвиток проєкту. Також розділення коду на модулі полегшує співпрацю між розробниками, оскільки вони можуть працювати над окремими модулями незалежно один від одного.

Розміщення компонентів у різних модулях сприяє полегшенню маршрутизації. Кожен модуль має свої власні відносні шляхи до компонентів, що робить управління маршрутами більш організованим і зрозумілим. Це особливо важливо в проєктах з великою кількістю компонентів і сторінок.

І, не менш важливе, використання різних модулів дозволяє досягти кращої проєктної структури й зберігати код чистим і легким для розуміння. Кожен модуль може мати свою структуру, яка відповідає його функціональності, і дозволяє зосередитися на конкретній області додатка.

Створимо «configuration» модуль, який дозволить адміністратору змінювати дані в застосунку. У цьому модулі будуть розміщені компоненти, які дозволять адміністратору вносити зміни в книги та жанри. Це дає можливість зручно та централізовано керувати конфігурацією і забезпечує гнучкість в управлінні додатком. Також згенереємо компоненти, які дазволять редагувати дані.

Генерувати модулі можна завдяки команді «ng generate module <name> --routing». Аргумент «routing» вказує, що для модуля необхідно створити систему маршрутизації.

Генерувати компоненти можна завдяки команді «ng generate component <name>».

Важливо зазначити, що параметр ім’я може бути відносним. Це дозволяє генерувати компоненти чи модулі всередині інших модулів.

Створимо спільні компоненти та сервіси, які будуть використовуватися в різних частинах застосунку.

Для початку створимо сервіс сповіщень. Він буде відображати користувачу інформацію про помилки, що виникли чи звичайні повідомлення про виконану операцію. Також додамо можливість підтвердити в користувача певну дію, наприклад видалити певну книгу. Функціонал для підтвердження дій покращить користувацький досвід при оперуванні даними, оскільки він знижує практично до нуля можливість випадкового видалення даних.

NotificationService – сервіс для взаємодії з користувачем, показу сповіщень.

Він має методи:

1. showSuccess – показує текстове повідомлення про успішну операцію.
2. showError - показує текстове повідомлення про провальну операцію.
3. showConfirmation – показує користувачу повідомлення та чекає на його підтвердження або відхиленння.

Створимо індикатор завантаження, який буде оповіщувати користувача про обробку даних і буде обмежувати його дії на час час. Оскільки даний компонент буде використовуватися у різних частинах застосунку, то для нього необхідно створити «SharedModule». Даний модуль використовується як модуль для спільних компонентів. За рахунок того, що в Angular модулі є singleton об’єктами, то проблем з імпортуванням даного модуля в різних компонентах не буде, оскільки в пам’яті буде лише один екземпляр цього модуля, який буде передаватися в різні частини застосунку.

Створивши «SharedModule», необхідно створити компонент «Spinner». Даний компонент не матиме поведінки, тому необхідно додати лише його розмітку (рис. 3.8).

```
<div class="spinner-overlay">
  <div class="spinner-container">
    <mat-spinner diameter="50"></mat-spinner>
  </div>
</div>
```

Рисунок 3.8 – Розмітка індикатора завантаження

Створимо компонент для відображення списку книг. Для зберігання даних на клієнтській частині використаємо Angular DevExtreme [11].

DevExtreme використовує сутність Store для здійснення операцій з даними. Store є об'єктом, який зберігає завантажені дані, дозволяє створювати нові записи, видаляти старі та проводити фільтрацію даних. Однак, для розширених можливостей ми можемо створити власний об'єкт CustomStore, який буде керувати нашими даними. Це дозволить нам налаштовувати та розширювати стандартну поведінку Store при роботі з даними, наприклад, здійснювати асинхронне посторінкове завантаження даних.

Створення CustomStore дозволяє нам зайнятися більш гнучким та точним керуванням операціями з даними. Ми можемо виконувати власну логіку під час завантаження, фільтрації, сортування та інших операцій з даними. Це дозволяє нам адаптувати Store до наших конкретних вимог і забезпечує більш гнучкий та потужний механізм роботи з даними в рамках DevExtreme. Створимо CustomStore перевизначивши кожну операцію зміни та читання даних, щоб всі операції опрацьовувалися на сервері. Оскільки додавання нових книг та редагування існуючих буде відбуватися на окремих сторінках, то необхідно перевизначити операцію завантаження списку книг, який буде посторінково вантажити книги з серверу та операцію видалення, яка досупна на сторінці перегляду списку книг (рис 3.9). Додамо саму таблицю, використовуючи DataGrid.

Також необхідно додати сторінку редагування книг та детального перегляду, де можна в зручному форматі переглянути розширені відомості про книгу.

Для цього додамо переадресацію на різні кнопки з сторінки зі списком книг. Перейти на детальний огляд можна буде натиснувши на рядок в таблиці з книгами, а для редагування чи створення необхідно натиснути необхідні кнопки (рис. 3.9).

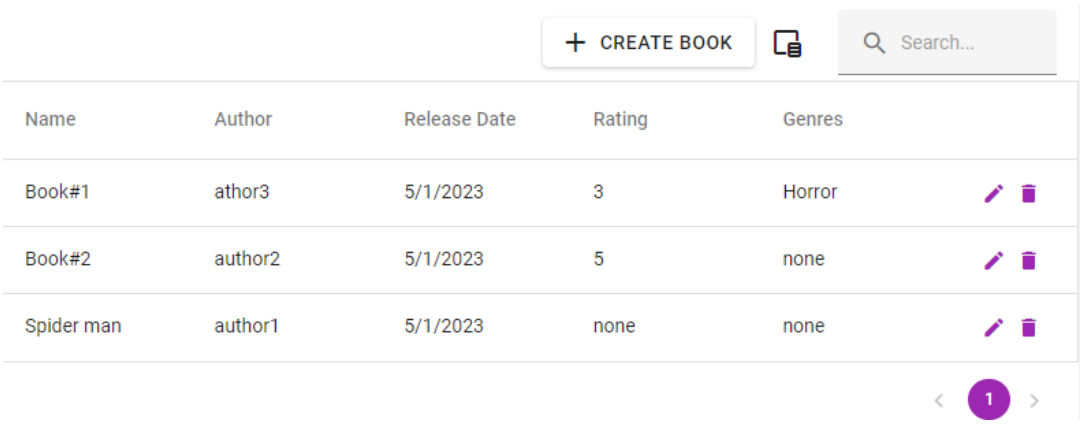

Рисунок 3.9 – Зовнішній вигляд списку книг

Кожен компонет має свій унікальний шлях.Для детального перегляду в системі маршрутизації створимо «path: 'details/:id'». Даний шлях має в собі параметр, який визначає ідентифікатор книги. Завдяки даному ідентифікатору стає можливим отримати з сервера всю інформацію про дану книгу та відобразити користувачу.

Перейдемо до системи захисту при маршрутизіції. Angular в системі маршрутизація дозволяє створити власні захисники шляхів. Створимо захисника, який дозволятиме вхід лише користувачам, які мають певні ролі. Для перевикористання в майбутньому дозволимо вказувати необхідну роль для кожного шляху окремо. Захисник спочатку перевірить чи має поточний користувач необхідну роль для доступу на цей ресурс. Якщо він має, то захисник пропустить користувача, а якщо ні, то він переадресує користувача на домашню

сторінку, якщо він авторизований. Це означає, що даний користувач не має необхідної ролі для перегляду даного ресурсу. Якщо ж користувач неавторизований, то він буде переадресований на сторінку авторизації з можливістю повернення на запитаний ресурс.

Передавати необхідну роль слід при визначенні шляхів в модулі маршрутизації.

```
{
  path: 'configuration',
  data: {
    role: 'System Admin',
  },
  canActivate: [AuthGuard],
  canLoad: [AuthGuard],
  loadChildren: () => import('./configuration/configuration.module').then(m => m.ConfigurationModule),
},
```

Рисунок 3.10 – Визначення шляху

Важливо зазначити, що при використанні параметр "loadChildren", модулі завантажуються асинхронно. Вони завантажуються лише тоді, коли користувач переходить на відповідну сторінку. Наприклад, якщо користувач не має наміру редагувати книги, його браузер не завантажуватиме код і розмітку, необхідні для цього модуля. Іншими словами, звичайним користувачам не буде доступний код для редагування, який призначений лише для адміністраторів і не буде навантажувати мережу та їхні браузери.

Створивши всі необхідні файли для клієнтської частини матимему структуру файлів та папок зображену на рис. 3.11.

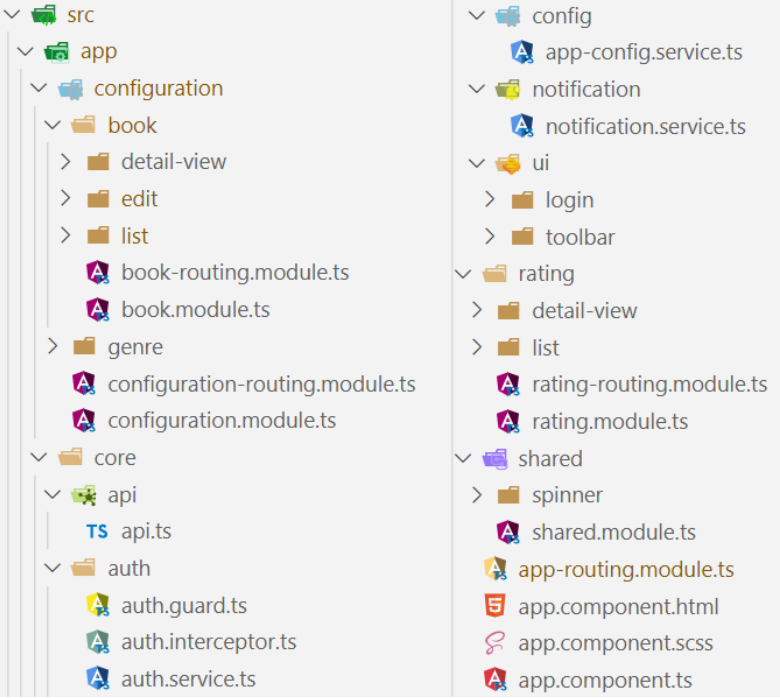

Рисунок 3.11 – Структура файлів та папок серверної частини

3.3 Приклад роботи

Перейшовши на головну сторінку застосунку користувач побачить список книг. Книги мають назву, автора, дату публікації, жанри та рейтинг, який базується на користувацьких оцінках (рис. 3.12).

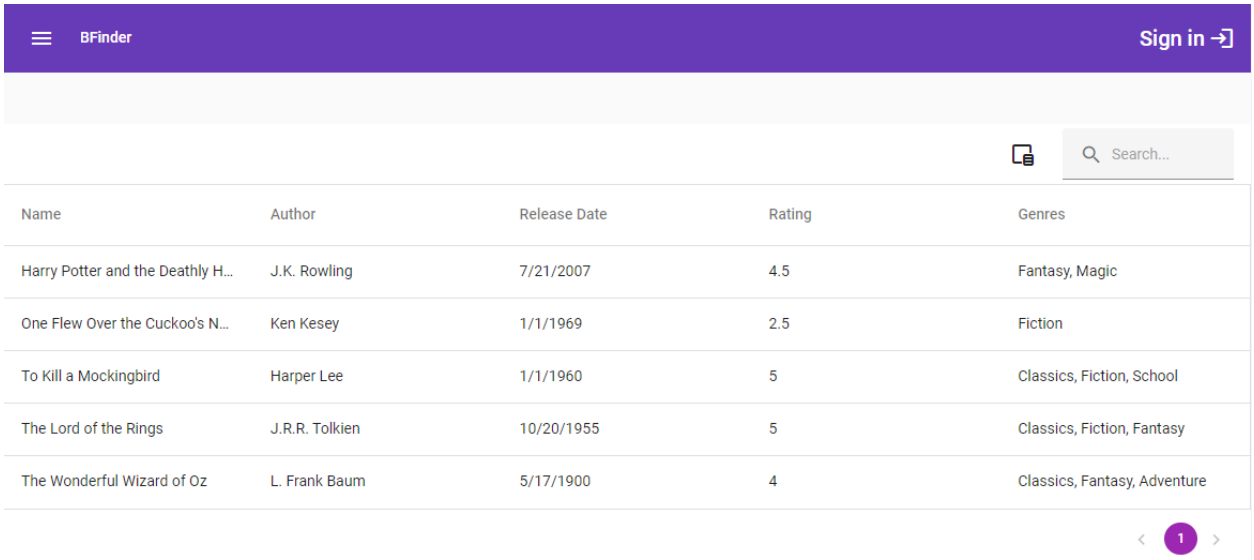

Рисунок 3.12 – Головна сторінка

При натисканні на певний рядок користувач побачить детальний огляд обраної книги (рис. 3.13).

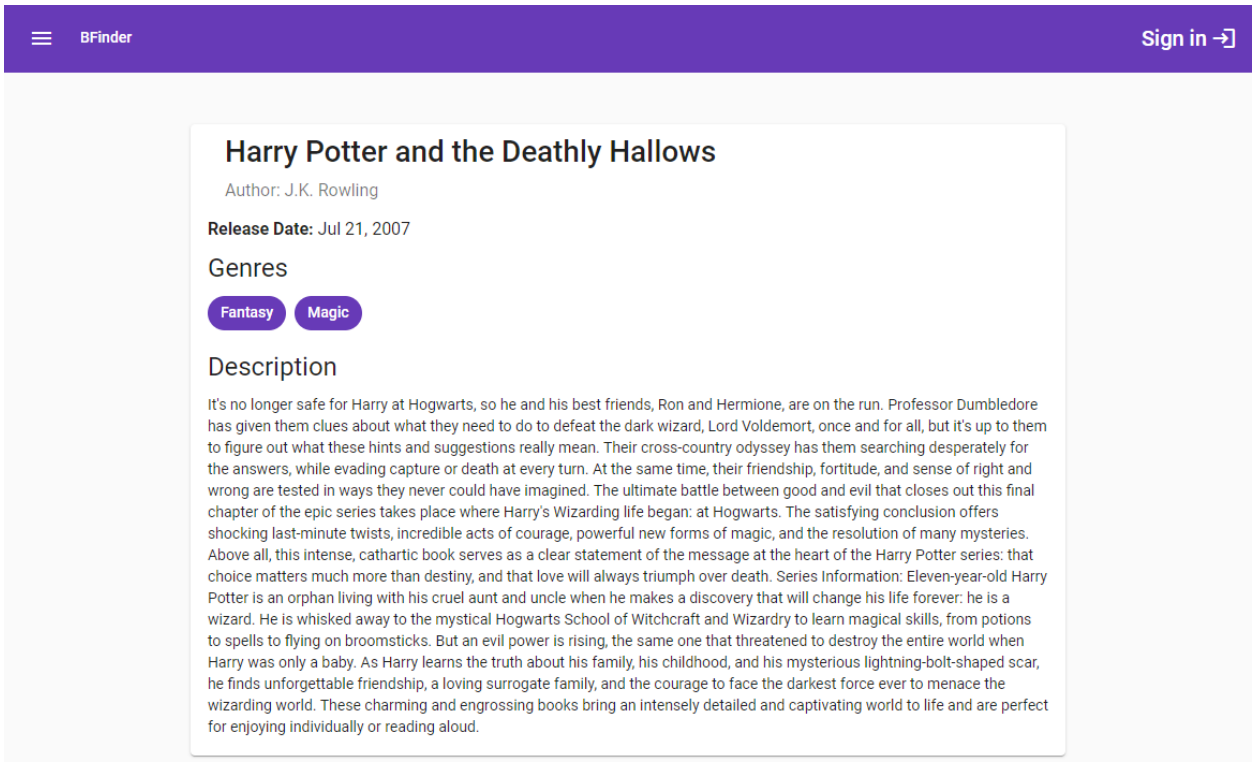

Рисунок 3.13 – Детальний огляд

Користувач може аутентифікуватися в системі. Для цього йому необхідно натиснути кнопку з написом «sign in». Після цього користувач буде переадресований на сторінку аутентифікації (рис. 3.14)

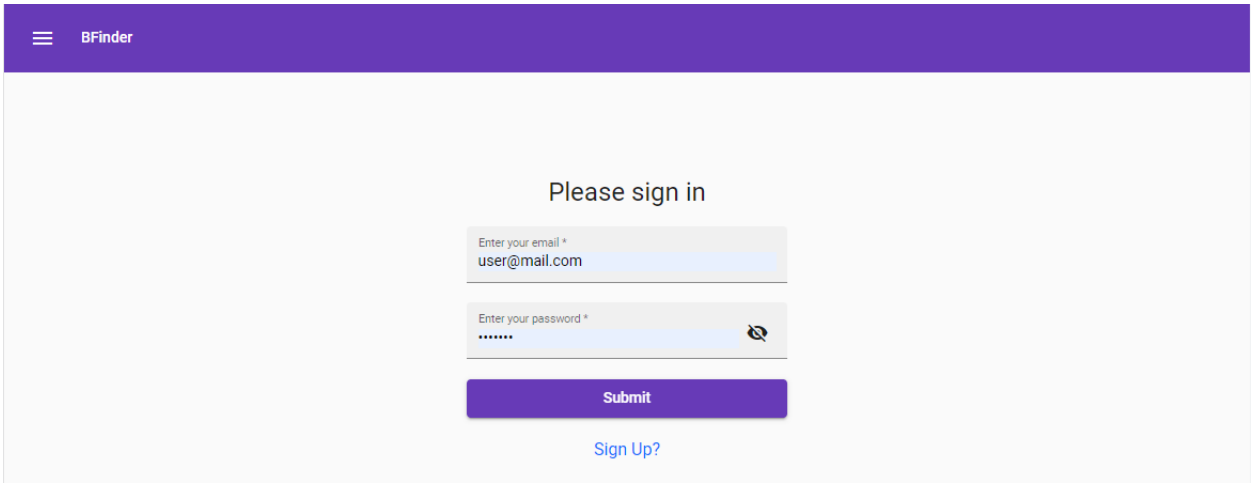

Рисунок 3.14 – Сторінка аутентифікації

Авторизованому користувачу відкривається можливість ставити оцінки книгам, на основі яких формується рейтинг. Також він може переглянути свою оцінку при детальному огляді (рис 3.15).

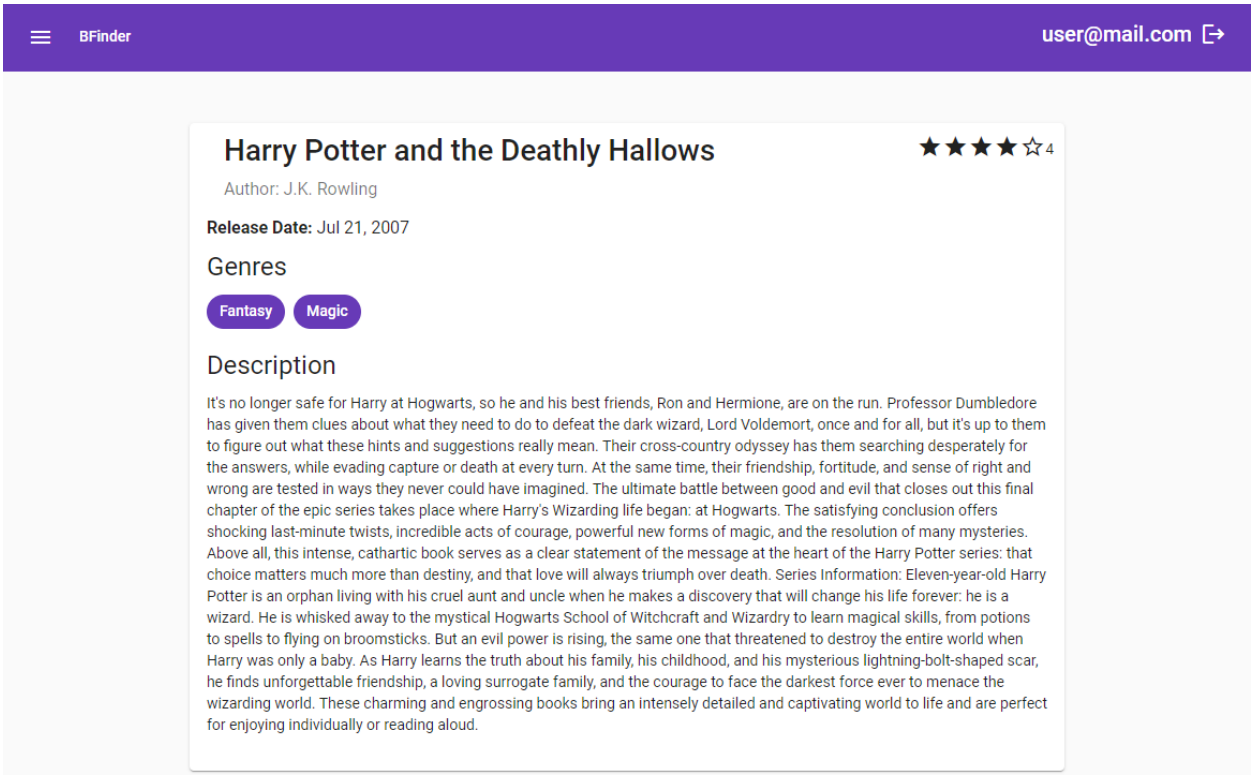

Рисунок 3.15 – Сторінка детального огляду для авторизованого користувача

Користувачі з привілегіями адміністратора мають право змінювати інформацію про книги та жанри. Для цього їм необхідно перейти на відповідну сторінку редагування (рис. 3.16), де будуть відображені відповідні кнопки для виконання необхідних дій над сутностями. Після натиснення на відповідні кнопки користувач буде переадресований на сторінку редагування (рис. 3.17) чи додавання нових сутностей (рис. 3.18).

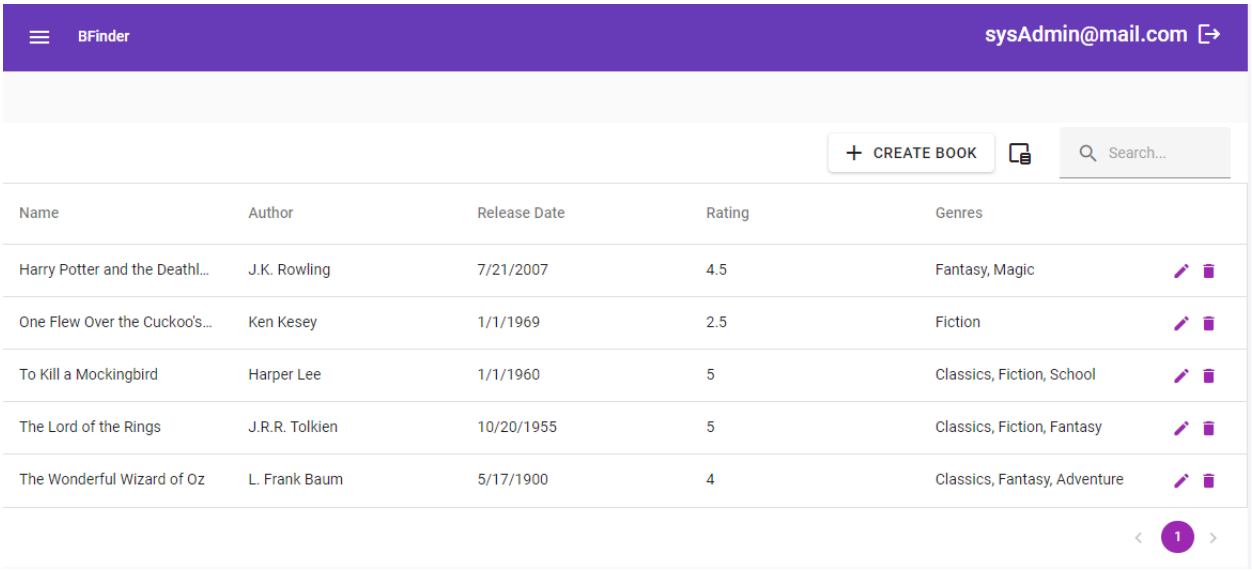

Рисунок 3.16 – Сторінка редагування книг

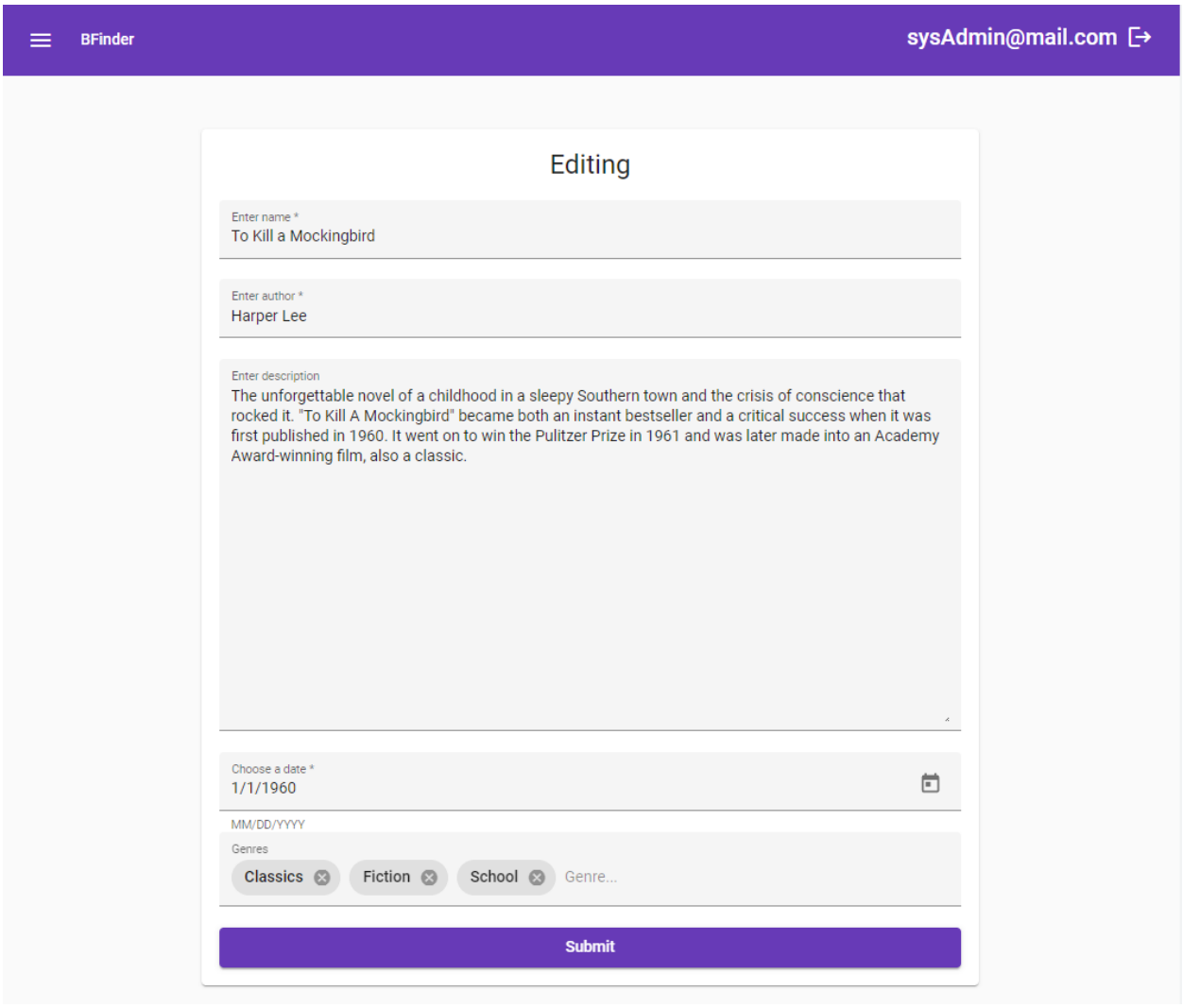

Рисунок 3.17 – Редагування книги

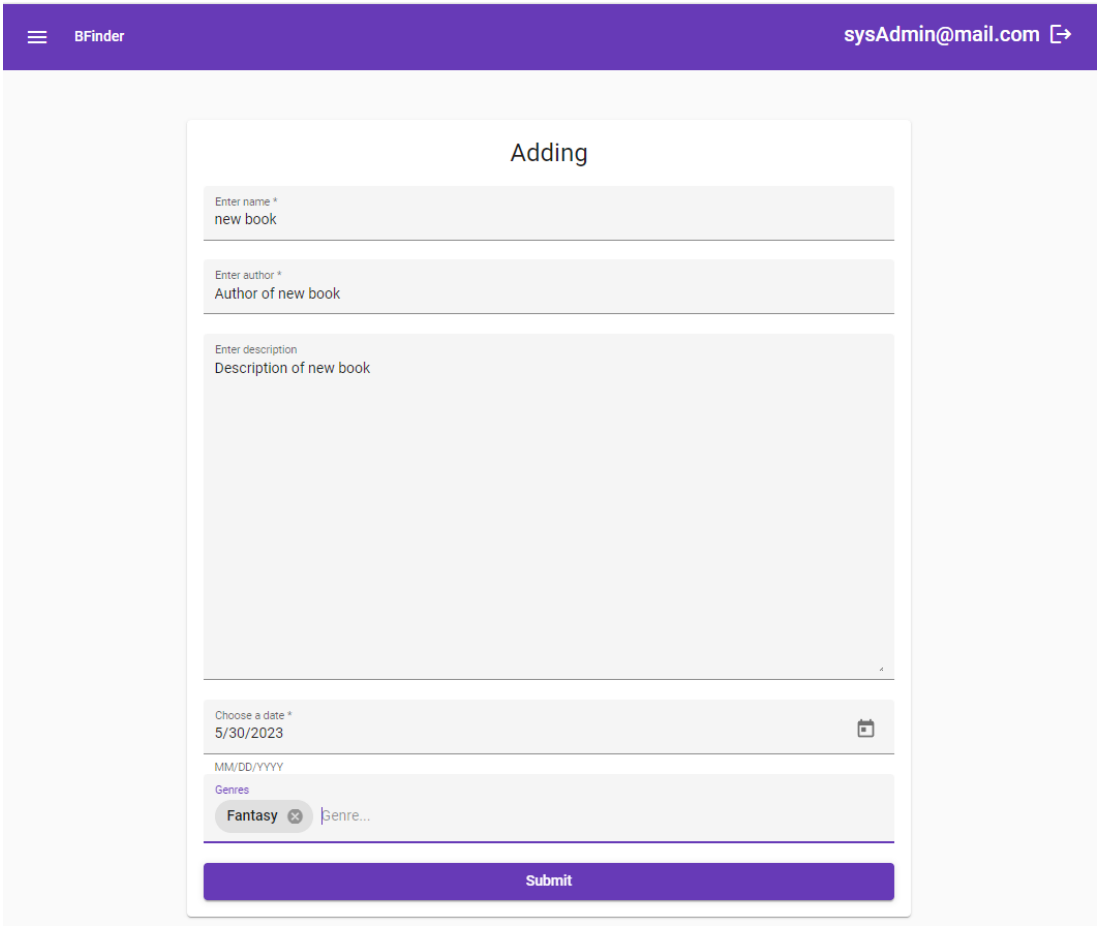

Рисунок 3.18 – Додавання нової книги

Також користувач може знайти книги за назвою, автором, рейтингом, датою публікації та жанром. Для цього йому необхідно ввести певний критерій пошуку в в поле для пошуку. Після цього йому відобразиться відфільтровані результати (рис. 3.19).

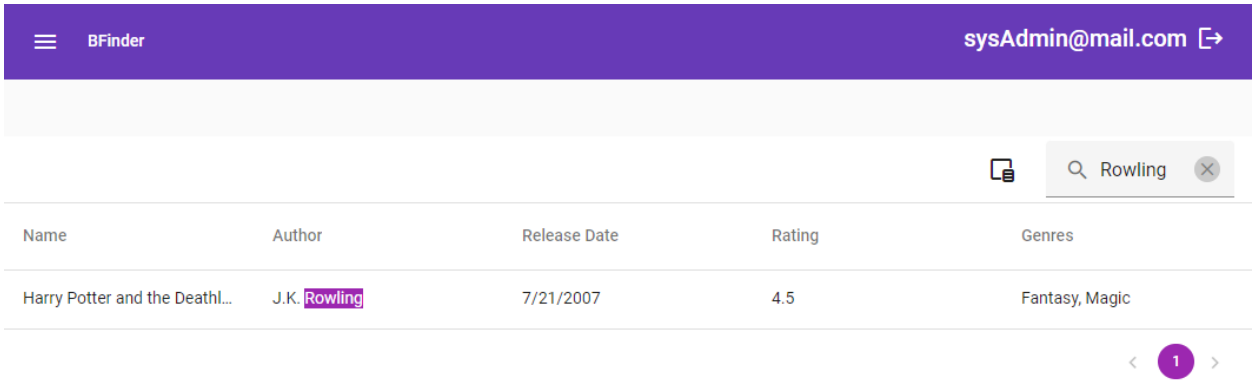

Рисунок 3.19 – Фільтрування книг

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було проведено детальний аналіз сучасних джерел, спрямований на вивчення та розуміння сучасних тенденцій у розробці веб-застосунків. Цей аналіз дав змогу визначити ключові аспекти, які необхідно врахувати при розробці системи управління книгами, а також з'ясувати перспективні підходи та методології.

На основі проведеного аналізу були сформульовані вимоги до системи управління книгами. Ці вимоги охоплюють зберігання інформації про книги та жанри, можливість фільтрування книг за різними параметрами, оцінку книг, а також функціонал реєстрації та авторизації користувачів. Реалізація цих вимог дозволить забезпечити зручну та ефективну взаємодію користувачів з системою, а також забезпечити безпеку та конфіденційність інформації.

На даному етапі була успішно реалізована система управління книгами з використанням обраних технологій. Ця реалізація включає основний функціонал, такий як додавання, видалення та редагування книг, а також функцію пошуку, фільтрування та пагінації. Система також надає можливість користувачам залишати оцінки книгам.

Для досягнення повного потенціалу системи та задоволення потреб користувачів, надалі планується розширити функціонал системи. Зокрема, розробити можливість керування користувачами, де адміністратори зможуть призначати ролі, видаляти користувачів та змінювати їм пароль. Додатково, планується реалізувати розширений фільтр та пошук, що дозволить користувачам швидко знаходити книги залежно від кількох параметрів. Також планується розширити систему рейтингів шляхом додавання можливості залишати відгуки.

Не менш важливим етапом розвитку системи є покриття усього функціоналу системи юніт-тестами та інтеграційними тестами для гарантування коректної роботи окремих компонентів та їх взаємодії.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Web Application Architecture: The Latest Guide 2022. ClickIT. URL: https://www.clickittech.com/devops/web-application-architecture/ (date of access: 10.05.2023).

2. Web Application Architecture: A Guide Through the Intricate Process of Building an App | LITSLINK Blog. Litslink. URL: https://litslink.com/blog/web-application-architecture (date of access: 10.05.2023).

3. What is RESTful API? - RESTful API Explained - AWS. Amazon Web Services, Inc. URL: https://aws.amazon.com/what-is/restful-api/?nc1=h_ls (date of access: 10.05.2023).

4. JWT.IO - JSON Web Tokens Introduction. JSON Web Tokens - jwt.io. URL: https://jwt.io/introduction (date of access: 10.05.2023).

5. .NET Core Overview. TutorialsTeacher - Learn Technologies. URL: https://www.tutorialsteacher.com/core/dotnet-core (date of access: 10.05.2023).

6. Overview of ASP.NET Core. Microsoft Learn: Build skills that open doors in your career. URL: https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-7.0 (date of access: 10.05.2023).

7. What is SQL Server? Introduction, History, Types, Versions. Guru99. URL: https://www.guru99.com/sql-server-introduction.html (date of access: 10.05.2023).

8. Angular. Angular. URL: https://angular.io/ (date of access: 10.05.2023).

9. RxJS. RxJS. URL: https://rxjs.dev/ (date of access: 10.05.2023).

10. Angular. Angular. URL: https://angular.io/cli (date of access: 10.05.2023).

11. DevExtreme - JavaScript UI Components for Angular, React, Vue and jQuery by DevExpress. DevExtreme - JavaScript UI Components for Angular, React, Vue and jQuery by DevExpress. URL: https://js.devexpress.com/ (date of access: 10.05.2023).

ДОДАТОК А

ApplicationDbContext.cs

```
public class ApplicationDbContext : IdentityDbContext<ApplicationUser>
{
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) :
base(options)
    { }

    public DbSet<Genre> Genres { get; set; }
    public DbSet<Book> Books { get; set; }
    public DbSet<Rating> Ratings { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        base.OnModelCreating(modelBuilder);

        modelBuilder.Entity<Genre>(builder => builder.Property(g =>
g.Id).HasDefaultValueSql("newsequentialid()"));
        modelBuilder.Entity<Book>(builder => builder.Property(b =>
b.Id).HasDefaultValueSql("newsequentialid()"));

        modelBuilder.Entity<Rating>(rating =>
        {
            rating.HasKey(r => new { r.PersonId, r.BookId });
            rating.HasOne(r => r.ApplicationUser)
                .WithMany(u => u.Ratings)
                .HasForeignKey(r => r.PersonId)
                .IsRequired();
            rating.HasOne(r => r.Book)
                .WithMany(f => f.Ratings)
                .HasForeignKey(r => r.BookId)
                .IsRequired();
        });

        SeedData(modelBuilder);
    }

    private void SeedData(ModelBuilder modelBuilder)
    {
        ApplicationUser[] users = {
            new ApplicationUser()
            {
                Id = "A5CD247E-2647-42E8-97B1-EAC777D33684",
                UserName = "sysAdmin@mail.com",
                NormalizedUserName = "SYSADMIN@MAIL.COM",
                Email = "sysAdmin@mail.com",
```

```
                NormalizedEmail = "SYSADMIN@MAIL.COM",
                SecurityStamp = "RCVAV4G7POXXPORBFV4MJA5CF6FK6DWC",
                PasswordHash =
"AQAAAAEAACcQAAAAEEBqf0+Sb/mN1DTq5tqzbEnTjCg26GaSOosFHIeF6R4NyymKfbcKPJx+JcBL120Vpw
==""
            }
        };

        IdentityRole[] roles = {
            new IdentityRole()
            {
                Id = "EDD8E58E-C34F-4CE8-B910-41A85389D143",
                Name = "System Admin",
                NormalizedName = "SYSTEM ADMIN"
            },
            new IdentityRole()
            {
                Id = "2867822E-73C5-4931-A18C-206122114A2A",
                Name = "User",
                NormalizedName = "USER"
            }
        };

        IdentityUserRole<string>[] userRoles = {
            new IdentityUserRole<string>()
            {
                UserId = "A5CD247E-2647-42E8-97B1-EAC777D33684",
                RoleId = "EDD8E58E-C34F-4CE8-B910-41A85389D143"
            }
        };

        modelBuilder.Entity<ApplicationUser>().HasData(users);
        modelBuilder.Entity<IdentityRole>().HasData(roles);
        modelBuilder.Entity<IdentityUserRole<string>>().HasData(userRoles);
    }
}
```

RatingRepository.cs

```
public class RatingRepository : IRatingRepository
{
    private readonly ApplicationDbContext _context;

    public RatingRepository(ApplicationDbContext context)
    {
        _context = context;
    }
```

```
    public async Task<Rating> GetById((string PersonId, Guid BookId) id)
    {
        return await _context.Ratings.FindAsync(id.PersonId, id.BookId);
    }

    public void Add(Rating rating)
    {
        _context.Ratings.Add(rating);
    }

    public void Update(Rating rating)
    {
        _context.Attach(rating);
        _context.Entry(rating).State = EntityState.Modified;
    }

    public async Task<int> SaveChangesAsync()
    {
        return await _context.SaveChangesAsync();
    }


    public Task<(List<Rating> data, int? totalCount)> List(Expression<Func<Rating,
bool>> query = null, int skip = 0, int take = 0)
    {
        throw new NotImplementedException();
    }

    public Task Remove((string PersonId, Guid BookId) id)
    {
        throw new NotImplementedException();
    }
}
```

GenreRepository.cs

```
public class GenreRepository : IGenreRepository
{
    private readonly ApplicationDbContext _context;

    public GenreRepository(ApplicationDbContext context)
    {
        _context = context;
    }
```

```
    public async Task<(List<Genre> data, int? totalCount)>
List(Expression<Func<Genre, bool>> query = null, int skip = 0, int take = 0)
    {
        return (await _context.Genres.ToListAsync(), null);
    }

    public async Task<Genre> GetById(Guid id)
    {
        return await _context.Genres.FindAsync(id);
    }

    public void Add(Genre genre)
    {
        _context.Genres.Add(genre);
    }

    public void Update(Genre genre)
    {
        _context.Attach(genre);
        _context.Entry(genre).State = EntityState.Modified;
    }

    public async Task Remove(Guid id)
    {
        Genre genre = await _context.Genres.FindAsync(id);

        if (genre != null)
        {
            _context.Genres.Remove(genre);
        }
    }

    public async Task<int> SaveChangesAsync()
    {
        return await _context.SaveChangesAsync();
    }
}
```

BookRepository.cs

```
public class BookRepository : IBookRepository
{
    private readonly ApplicationDbContext _context;

    public BookRepository(ApplicationDbContext context)
    {
        _context = context;
```

```
    }

    public async Task<(List<Book> data, int? totalCount)>
List(Expression<Func<Book, bool>> filterQuery, int skip, int take)
    {
        var query = _context.Books
            .Where(filterQuery)
            .OrderByDescending(b => b.ReleaseDate);

        int totalCount = await query.CountAsync();
        List<Book> books = await query
            .Include(b => b.Genres)
            .Include(b => b.Ratings)
            .Skip(skip)
            .Take(take)
            .ToListAsync();

        books.ForEach(CalculateRating);

        return (books, totalCount);
    }

    public async Task<Book> GetById(Guid id)
    {
        var book = await _context.Books
            .Where(book => book.Id == id)
            .Include(b => b.Genres)
            .Include(b => b.Ratings)
            .FirstOrDefaultAsync();

        CalculateRating(book);

        return book;

    }

    public void Add(Book book)
    {
        _context.Books.Add(book);

        if (book.Genres != null)
        {
            foreach (Genre genre in book.Genres)
            {
                _context.Entry(genre).State = EntityState.Unchanged;
            }
        }
```

```
    }

    public void Update(Book book)
    {
        var existingBook = _context.Books
            .Include(b => b.Genres)
            .FirstOrDefault(b => b.Id == book.Id);

        if (existingBook != null)
        {
            existingBook.Name = book.Name;
            existingBook.Author = book.Author;
            existingBook.Description = book.Description;
            existingBook.ReleaseDate = book.ReleaseDate;

            if (book.Genres != null)
            {
                // Get the existing genres from the database based on their IDs
                var existingGenres = _context.Genres.Where(g =>
book.Genres.Select(bg => bg.Id).Contains(g.Id)).ToList();

                existingBook.Genres.Clear();
                existingBook.Genres.AddRange(existingGenres);
            }
        }
    }

    public async Task Remove(Guid id)
    {
        Book book = await _context.Books.FindAsync(id);
        if (book != null)
        {
            _context.Books.Remove(book);
        }
    }

    public async Task<int> SaveChangesAsync()
    {
        return await _context.SaveChangesAsync();
    }

    private void CalculateRating(Book book)
    {
        book.Rating = book.Ratings.Count > 0 ? book.Ratings.Average(rating =>
rating.Point) : null;
    }
}
```

BookService.cs

```
public class BookService : IBookService
{
    private readonly IBookRepository _bookRepository;
    private readonly IRatingRepository _ratingRepository;

    public BookService(IBookRepository bookRepository, IRatingRepository
ratingRepository)
    {
        _bookRepository = bookRepository;
        _ratingRepository = ratingRepository;
    }

    public async Task<ServiceResult<List<Book>>> List(BookFilter filter)
    {
        var (data, totalCount) = await _bookRepository.List(buildQuery(filter),
filter.Skip, filter.Take);

        return new ServiceResult<List<Book>>()
        {
            Data = data,
            TotalCount = totalCount
        };
    }

    public async Task<ServiceResult<Book>> Get(Guid id)
    {
        return new ServiceResult<Book>()
        {
            Data = await _bookRepository.GetById(id)
        };
    }

    public async Task<ServiceResult<Book>> Save(Book book)
    {
        try
        {
            _bookRepository.Add(book);
            await _bookRepository.SaveChangesAsync();
        }
        catch (Exception)
        {
            throw new ServiceException(HttpStatusCode.InternalServerError,
"Creation error");
        }
```

```
        return new ServiceResult<Book>()
        {
            Data = book
        };
    }

    public async Task<ServiceResult<Book>> Update(Book book)
    {
        try
        {
            _bookRepository.Update(book);
            await _bookRepository.SaveChangesAsync();
        }
        catch (Exception)
        {
            throw new ServiceException(HttpStatusCode.InternalServerError,
"Updating error");
        }

        return new ServiceResult<Book>()
        {
            Data = book
        };
    }

    public async Task Delete(Guid id)
    {
        try
        {
            await _bookRepository.Remove(id);
            await _bookRepository.SaveChangesAsync();
        }
        catch (Exception)
        {
            throw new ServiceException(HttpStatusCode.InternalServerError,
"Deleting error");
        }
    }

    public async Task RateBook(BookRate rate, string userId)
    {
        var existingRating = await _ratingRepository.GetById((userId,
rate.BookId));
        if (existingRating != null)
        {
            existingRating.Point = rate.Point;
```

```
            _ratingRepository.Update(existingRating);
        }
        else
        {
            _ratingRepository.Add(new Rating()
            {
                PersonId = userId,
                BookId = rate.BookId,
                Point = rate.Point
            });
        }
        await _ratingRepository.SaveChangesAsync();
    }

    private Expression<Func<Book, bool>> buildQuery(BookFilter filter)
    {
        Expression<Func<Book, bool>> query = book => true;
        return query;
    }
}
```

GenreService.cs

```
public class GenreService: IGenreService
{
    private readonly IGenreRepository _genreRepository;

    public GenreService(IGenreRepository genreRepository)
    {
        _genreRepository = genreRepository;
    }

    public async Task<ServiceResult<List<Genre>>> List()
    {
        var (data, totalCount) = await _genreRepository.List();
        return new ServiceResult<List<Genre>>()
        {
            Data = data,
            TotalCount = totalCount
        };
    }

    public async Task<ServiceResult<Genre>> Get(Guid id)
    {
        return new ServiceResult<Genre>()
        {
            Data = await _genreRepository.GetById(id)
```

```
        };
    }

    public async Task<ServiceResult<Genre>> Save(Genre genre)
    {
        try
        {
            _genreRepository.Add(genre);
            await _genreRepository.SaveChangesAsync();
        }
        catch (Exception)
        {
            throw new ServiceException(HttpStatusCode.InternalServerError,
"Creation error");
        }

        return new ServiceResult<Genre>()
        {
            Data = genre
        };
    }

    public async Task<ServiceResult<Genre>> Update(Genre genre)
    {
        try
        {
            _genreRepository.Update(genre);
            await _genreRepository.SaveChangesAsync();
        }
        catch (Exception)
        {
            throw new ServiceException(HttpStatusCode.InternalServerError,
"Updating error");
        }

        return new ServiceResult<Genre>()
        {
            Data = genre
        };
    }

    public async Task Delete(Guid id)
    {
        try
        {
            await _genreRepository.Remove(id);
            await _genreRepository.SaveChangesAsync();
```

```
        }
        catch (Exception)
        {
            throw new ServiceException(HttpStatusCode.InternalServerError,
"Deleting error");
        }
    }
}
```

BookController.cs

```
[Authorize]
[Route("api/[controller]")]
[ApiController]
public class BookController : ControllerBase
{
    private readonly IBookService _bookService;

    public BookController(IBookService bookService)
    {
        _bookService = bookService;
    }

    [AllowAnonymous]
    [HttpGet]
    public async Task<ServiceResult<List<Book>>> List([FromQuery] BookFilter
filter)
    {
        return await _bookService.List(filter);
    }

    [AllowAnonymous]
    [HttpGet("{id}")]
    public async Task<ServiceResult<Book>> Get(Guid id)
    {
        var result = await _bookService.Get(id);
        var book = result.Data;
        string userId = User.FindFirst(ClaimTypes.NameIdentifier)?.Value;

        book.UserRating = book.Ratings.Find(rating => rating.PersonId ==
userId)?.Point ?? 0;
        return result;
    }

    [Authorize(Policy = "HasAccessToConfiguration")]
    [HttpPost]
    public async Task<ServiceResult<Book>> Save([FromBody] Book book)
```

```
    {
        return await _bookService.Save(book);
    }

    [Authorize(Policy = "HasAccessToConfiguration")]
    [HttpPut]
    public async Task<ServiceResult<Book>> Update([FromBody] Book book)
    {
        return await _bookService.Update(book);
    }

    [Authorize(Policy = "HasAccessToConfiguration")]
    [HttpDelete("{id}")]
    public async Task Delete(Guid id)
    {
        await _bookService.Delete(id);
    }

    [HttpPost("rating")]
    public async Task<ActionResult> Rate([FromBody] BookRate rate)
    {
        string userId = User.FindFirst(ClaimTypes.NameIdentifier)?.Value;

        await _bookService.RateBook(rate, userId);
        return Ok();
    }
}
```

GenreController.cs

```
[Authorize(Policy = "HasAccessToConfiguration")]
[Route("api/[controller]")]
[ApiController]
public class GenreController : ControllerBase
{
    private readonly IGenreService _genreService;

    public GenreController(IGenreService genreService)
    {
        _genreService = genreService;
    }

    [HttpGet]
    public async Task<ServiceResult<List<Genre>>> List()
    {
        return await _genreService.List();
    }
```

```
    [HttpGet("{id}")]
    public async Task<ServiceResult<Genre>> Get(Guid id)
    {
        return await _genreService.Get(id);
    }

    [HttpPost]
    public async Task<ServiceResult<Genre>> Save([FromBody] Genre genre)
    {
        return await _genreService.Save(genre);
    }

    [HttpPut]
    public async Task<ServiceResult<Genre>> Update([FromBody] Genre genre)
    {
        return await _genreService.Update(genre);
    }

    [HttpDelete("{id}")]
    public async Task Delete(Guid id)
    {
        await _genreService.Delete(id);
    }
}
```

FilterExceptionMiddleware.cs

```
public class FilterExceptionMiddleware
{
    private readonly RequestDelegate _next;
    private readonly ILogger<FilterExceptionMiddleware> _logger;

    public FilterExceptionMiddleware(RequestDelegate next,
ILogger<FilterExceptionMiddleware> logger)
    {
        _next = next;
        _logger = logger;
    }

    public async Task InvokeAsync(HttpContext context)
    {
        try
        {
            await _next(context);
        }
        catch (Exception e)
```

```
        {
            _logger.LogError(e, "An error occurred");

            ExceptionResult result = e switch
            {
                ServiceException serviceException =>
HandleServiceException(serviceException),
                Exception => HandleException(e)
            };

            context.Response.StatusCode = result.StatusCode;
            context.Response.ContentType = "application/json";

            await context.Response.WriteAsync(JsonSerializer.Serialize(new
            {
                message = result.Message
            }));
        }
    }

    private ExceptionResult HandleServiceException(ServiceException e)
    {
        return new ExceptionResult()
        {
            StatusCode = (int)e.StatusCode,
            Message = e.Message
        };
    }

    private ExceptionResult HandleException(Exception e)
    {
        return new ExceptionResult()
        {
            StatusCode = 500,
            Message = "An unexpected error occured"
        };
    }
}
```

book-detail-view.component

```
import { Component, OnDestroy, OnInit } from '@angular/core';
import { ActivatedRoute } from '@angular/router';
import { Subject, takeUntil } from 'rxjs';
import { Book, BookClient } from 'src/app/core/api/api';
```

```
@Component({
  selector: 'app-book-detail-view',
  templateUrl: './book-detail-view.component.html',
  styleUrls: ['./book-detail-view.component.scss'],
})
export class BookDetailViewComponent implements OnInit, OnDestroy {
  destroyed$: Subject<boolean>;
  loading: boolean;
  id: string;
  book: Book;

  constructor(private route: ActivatedRoute, private bookClient: BookClient) {
    this.destroyed$ = new Subject();
  }

  ngOnInit(): void {
    this.id = this.route.snapshot.paramMap.get('id');
    this.loadData(this.id);
  }

  ngOnDestroy(): void {
    this.destroyed$.next(true);
    this.destroyed$.complete();
  }

  loadData(id: string): void {
    this.loading = true;

    this.bookClient
      .get(id)
      .pipe(takeUntil(this.destroyed$))
      .subscribe(result => {
        this.book = result.data;
        this.loading = false;
      });
  }
}
```

```
<app-spinner *ngIf="loading; else detailContent"></app-spinner>

<ng-template #detailContent>
  <div class="book-container" *ngIf="!loading">
    <mat-card>
      <mat-card-header>
        <mat-card-title class="mb-4 fs-3">{{ book.name }}</mat-card-title>
```

```
        <mat-card-subtitle class="mb-3 fs-6">Author: {{ book.author }}</mat-card-
subtitle>
      </mat-card-header>

      <mat-card-content>
        <p class="mb-3 fs-6"><strong>Release Date:</strong> {{ book.releaseDate |
date }}</p>
        <p class="mb-3 fs-4">Genres</p>
        <p *ngIf="book.genres.length === 0; else genreChips">None</p>
        <ng-template #genreChips>
          <mat-chip-list class="readonly-chips">
            <mat-chip *ngFor="let genre of book.genres" [selected]="true"
[removable]="false" disableRipple>
              {{ genre.name }}
            </mat-chip>
          </mat-chip-list>
        </ng-template>

        <p class="mt-4 mb-3 fs-4">Description</p>
        <p class="text-wrap">{{ book.description }}</p>
      </mat-card-content>
    </mat-card>
  </div>
</ng-template>
```

book-edit.component

```
import { COMMA, ENTER } from '@angular/cdk/keycodes';
import { Component, ElementRef, OnDestroy, OnInit, ViewChild } from
'@angular/core';
import { FormControl, FormGroup, Validators } from '@angular/forms';
import { MatAutocompleteSelectedEvent } from '@angular/material/autocomplete';
import { MatChipInputEvent } from '@angular/material/chips';
import { ActivatedRoute, Router } from '@angular/router';
import { Observable, Subject, forkJoin, map, startWith, takeUntil } from 'rxjs';
import { Book, BookClient, Genre, GenreClient, ServiceResultOfBook,
ServiceResultOfListOfGenre } from 'src/app/core/api/api';
import { NotificationService } from
'src/app/core/notification/notification.service';

@Component({
  selector: 'app-book-edit',
  templateUrl: './book-edit.component.html',
  styleUrls: ['./book-edit.component.scss'],
})
export class BookEditComponent implements OnInit, OnDestroy {
```

```
  destroyed$: Subject<boolean>;
  loading: boolean;
  book: Book;
  form: FormGroup;

  isEditMode: boolean;

  separatorKeysCodes: number[] = [ENTER, COMMA];
  genreCtrl: FormControl;
  filteredGenres: Observable<Genre[]>;
  genres: Genre[] = [];
  allGenres: Genre[] = [];

  @ViewChild('genreInput') genreInput: ElementRef<HTMLInputElement>;

  constructor(
    private route: ActivatedRoute,
    private router: Router,
    private bookClient: BookClient,
    private genreClient: GenreClient,
    private notifyService: NotificationService,
  ) {
    this.destroyed$ = new Subject();

    this.form = new FormGroup({
      name: new FormControl('', [Validators.required]),
      author: new FormControl('', [Validators.required]),
      description: new FormControl('', []),
      releaseDate: new FormControl('', [Validators.required]),
      genres: new FormControl('', []),
    });

    this.genreCtrl = new FormControl('');

    this.filteredGenres = this.genreCtrl.valueChanges.pipe(
      startWith(null),
      map((fruit: string | null) => (fruit ? this.filterGenres(fruit) :
this.allGenres.slice()))
    );
  }

  ngOnInit(): void {
    const id = this.route.snapshot.paramMap.get('id');
    this.isEditMode = !!id;
    this.loadData(id);
  }
```

```
  ngOnDestroy(): void {
    this.destroyed$.next(true);
    this.destroyed$.complete();
  }

  loadData(id: string): void {
    this.loading = true;

    const requests: Array<Observable<ServiceResultOfListOfGenre> |
Observable<ServiceResultOfBook>> = [
      this.genreClient.list(),
    ];

    if (this.isEditMode) {
      requests.push(this.bookClient.get(id));
    }

    forkJoin(requests)
      .pipe(takeUntil(this.destroyed$))
      .subscribe({
        next: results => {
          this.allGenres = results[0].data as Genre[];
          if (this.isEditMode) {
            this.book = results[1].data as Book;
            this.form.controls.name.setValue(this.book.name);
            this.form.controls.author.setValue(this.book.author);
            this.form.controls.description.setValue(this.book.description);
            this.form.controls.releaseDate.setValue(this.book.releaseDate);
            this.genres = this.book.genres?.slice() || [];
          }
          this.loading = false;
        },
        error: (e) => {
          this.notifyService.showError(e);
          this.loading = false;
        }
      });
  }

  addGenre(event: MatChipInputEvent): void {
    const value = (event.value || '').trim();

    const genre = this.allGenres.find(genre => genre.name.toLowerCase() ===
value.toLowerCase());
    if (genre) {
      this.genres.push(genre);
    }
```

```
    // Clear the input value
    event.chipInput.clear();

    this.genreCtrl.setValue(null);
  }
  removeGenre(genre: Genre): void {
    const index = this.genres.indexOf(genre);

    if (index >= 0) {
      this.genres.splice(index, 1);
    }
  }
  selectedGenre(event: MatAutocompleteSelectedEvent): void {
    const genre = this.allGenres.find(genre => genre.name ===
event.option.viewValue);
    this.genres.push(genre);
    this.genreInput.nativeElement.value = '';
    this.genreCtrl.setValue(null);
  }

  onSubmit() {
    if (this.form.invalid) {
      return;
    }

    const book = new Book();

    book.id = this.book?.id || undefined;
    book.name = this.form.controls.name.value;
    book.author = this.form.controls.author.value;
    book.description = this.form.controls.description.value;
    book.releaseDate = this.form.controls.releaseDate.value;
    book.genres = this.genres;

    if (this.isEditMode) {
      this.bookClient.update(book).subscribe({
        next: () => {
          this.router.navigate(['../../../'], { relativeTo: this.route });
        },
        error: (e) => {
          this.notifyService.showError('An error occurred');
          console.log(e);
        }
      });
    } else {
      this.bookClient.save(book).subscribe({
```

```
        next: () => {
          this.router.navigate(['../'], { relativeTo: this.route });
        },
        error: (e) => {
          this.notifyService.showError('An error occurred');
          console.log(e);
        }
      });
    }
  }

  private filterGenres(value: string): Genre[] {
    const filterValue = value.toLowerCase();
    return this.allGenres.filter(genre =>
genre.name.toLowerCase().includes(filterValue));
  }
}
```

```
<app-spinner *ngIf="loading; else detailContent"></app-spinner>

<ng-template #detailContent>
  <div class="book-container" *ngIf="!loading">
    <mat-card class="book-card">
      <mat-card-content>
        <form [formGroup]="form" (submit)="onSubmit()">
          <div>
            <h1 class="h3 mb-3 fw-normal text-center">{{isEditMode ? "Editing" :
"Adding"}}</h1>
          </div>

          <mat-form-field appearance="fill" class="w-100">
            <mat-label>Enter name</mat-label>
            <input matInput formControlName="name" />
            <mat-error *ngIf="this.form.controls['name'].invalid">Field is
required</mat-error>
          </mat-form-field>

          <mat-form-field appearance="fill" class="w-100">
            <mat-label>Enter author</mat-label>
            <input matInput formControlName="author" />
            <mat-error *ngIf="this.form.controls['author'].invalid">Field is
required</mat-error>
          </mat-form-field>

          <mat-form-field appearance="fill" class="w-100">
```

```
            <mat-label>Enter description</mat-label>
            <textarea matInput formControlName="description" style="min-height:
300px"></textarea>
            <mat-error *ngIf="this.form.controls['description'].invalid">Field is
required</mat-error>
          </mat-form-field>

          <mat-form-field appearance="fill" class="w-100">
            <mat-label>Choose a date</mat-label>
            <input matInput formControlName="releaseDate" [matDatepicker]="picker">
            <mat-hint>MM/DD/YYYY</mat-hint>
            <mat-datepicker-toggle matSuffix [for]="picker"></mat-datepicker-
toggle>
            <mat-datepicker #picker></mat-datepicker>
            <mat-error *ngIf="this.form.controls['releaseDate'].invalid">Field is
required</mat-error>
          </mat-form-field>

          <mat-form-field class="example-chip-list" appearance="fill" class="w-
100">
            <mat-label>Genres</mat-label>
            <mat-chip-list #chipList aria-label="Genre selection">
              <mat-chip
                *ngFor="let genre of genres"
                (removed)="removeGenre(genre)">
                {{genre.name}}
                <button matChipRemove>
                  <mat-icon>cancel</mat-icon>
                </button>
              </mat-chip>
              <input
                placeholder="Genre..."
                #genreInput
                [formControl]="genreCtrl"
                [matAutocomplete]="auto"
                [matChipInputFor]="chipList"
                [matChipInputSeparatorKeyCodes]="separatorKeysCodes"
                (matChipInputTokenEnd)="addGenre($event)">
            </mat-chip-list>
            <mat-autocomplete #auto="matAutocomplete"
(optionSelected)="selectedGenre($event)">
              <mat-option *ngFor="let genre of filteredGenres | async"
[value]="genre.name">
                {{genre.name}}
              </mat-option>
            </mat-autocomplete>
          </mat-form-field>
```

```
          <button class="w-100" type="submit" mat-raised-button
color="primary">Submit</button>
        </form>
      </mat-card-content>
    </mat-card>
  </div>
</ng-template>
```

book-list.component

```
import { Component } from '@angular/core';
import { ActivatedRoute, Router } from '@angular/router';
import CustomStore, { ResolvedData } from 'devextreme/data/custom_store';
import { firstValueFrom, map } from 'rxjs';
import { Book, BookClient, ServiceResultOfListOfBook } from 'src/app/core/api/api';
import { AppConfigService } from 'src/app/core/config/app-config.service';

@Component({
  selector: 'app-book-list',
  templateUrl: './book-list.component.html',
  styleUrls: ['./book-list.component.scss'],
})
export class BookListComponent {
  dataSource: CustomStore;

  constructor(
    private bookClient: BookClient,
    public config: AppConfigService,
    private router: Router,
    private route: ActivatedRoute
  ) {
    this.dataSource = new CustomStore({
      key: 'id',
      load: loadOptions => {
        return firstValueFrom(
          this.bookClient.list(loadOptions.skip, loadOptions.take).pipe(
            map((result: ServiceResultOfListOfBook): ResolvedData<Book> => {
              return {
                data: result.data,
                totalCount: result.totalCount,
              };
            })
          )
        );
      },
```

```
      remove: (key: string) => {
        return firstValueFrom(this.bookClient.delete(key));
      },
    });

    this.onEditClick = this.onEditClick.bind(this);
  }

  onRowClick(e: { data: Book }) {
    this.router.navigate(['details', e.data.id], { relativeTo: this.route });
  }

  calculateGenres(book: Book): string {
    if (book.genres?.length > 0) {
      return book.genres.map(genre => genre.name).join(', ');
    }
    return 'none';
  }

  calculateRating(book: Book): string {
    if (book.rating === null) {
      return 'none';
    }
    return book.rating.toString();
  }

  onEditClick(e: { row: { data: Book } }): void {
    this.router.navigate(['details', e.row.data.id, 'edit'], { relativeTo:
this.route });
  }
  onCreateClick(): void {
    this.router.navigate(['create'], { relativeTo: this.route });
  }
}
```

```
<dx-data-grid id="dataGrid" [dataSource]="dataSource" [showBorders]="true"
[allowColumnReordering]="true"
  [allowColumnResizing]="true" (onRowClick)="onRowClick($event)"
  [hoverStateEnabled]="true">
  <dxo-toolbar>
    <dxi-item>
      <div *dxTemplate>
        <dx-button icon="add" text="Create book" (onClick)="onCreateClick()"></dx-
button>
      </div>
```

```
    </dxi-item>
    <dxi-item name="columnChooserButton"></dxi-item>
    <dxi-item name="searchPanel"></dxi-item>
  </dxo-toolbar>
  <dxo-column-chooser [enabled]="true"></dxo-column-chooser>
  <dxo-search-panel [visible]="true"></dxo-search-panel>

  <dxo-editing mode="popup" [allowDeleting]="true" [allowAdding]="false"
[useIcons]="true"> </dxo-editing>

  <dxi-column dataField="name"></dxi-column>

  <dxi-column dataField="author"></dxi-column>

  <dxi-column dataField="releaseDate" dataType="date"></dxi-column>

  <dxi-column dataField="rating" [calculateCellValue]="calculateRating"></dxi-
column>

  <dxi-column dataField="genres" [calculateCellValue]="calculateGenres"></dxi-
column>

  <dxi-column type="buttons">
    <dxi-button hint="Edit" icon="edit" [onClick]="onEditClick"></dxi-button>
    <dxi-button name="delete"></dxi-button>
  </dxi-column>

  <dxo-paging [pageSize]="config.pageSize"> </dxo-paging>
  <dxo-pager [visible]="true" [showNavigationButtons]="true"> </dxo-pager>

  <dxo-remote-operations [paging]="true">
  </dxo-remote-operations>
</dx-data-grid>
```

genre-list.component

```
import { Component } from '@angular/core';
import CustomStore from 'devextreme/data/custom_store';
import { firstValueFrom } from 'rxjs';
import { Genre, GenreClient } from 'src/app/core/api/api';
import { AppConfigService } from 'src/app/core/config/app-config.service';

@Component({
  selector: 'app-genre-list',
  templateUrl: './genre-list.component.html',
  styleUrls: ['./genre-list.component.scss'],
```

```
})
export class GenreListComponent {
  dataSource: CustomStore;

  constructor(private genreClient: GenreClient, public config: AppConfigService) {
    this.dataSource = new CustomStore({
      key: 'id',
      load: () => {
        return firstValueFrom(this.genreClient.list()).then(result => {
          return {
            data: result.data,
          };
        });
      },
      insert: genre => {
        return firstValueFrom(this.genreClient.save(genre));
      },
      update: (key, values) => {
        return firstValueFrom(this.genreClient.update(new Genre({ id: key, name:
values.name })));
      },
      remove: key => {
        return firstValueFrom(this.genreClient.delete(key));
      },
    });
  }
}
```

```
<dx-data-grid
  id="dataGrid"
  [dataSource]="dataSource"
  keyExpr="id"
  [showBorders]="true"
  [allowColumnReordering]="true"
  [allowColumnResizing]="true"
>
  <dxo-toolbar>
    <dxi-item name="addRowButton" showText="always"></dxi-item>
    <dxi-item name="searchPanel"></dxi-item>
  </dxo-toolbar>

  <dxo-search-panel [visible]="true"></dxo-search-panel>
  <dxo-editing mode="form" [allowUpdating]="true" [allowDeleting]="true"
[allowAdding]="true"> </dxo-editing>
```

```
  <dxi-column dataField="name">
    <dxi-validation-rule type="required" message="Name is required"></dxi-
validation-rule>
  </dxi-column>

  <dxo-paging [pageSize]="config.pageSize"> </dxo-paging>
  <dxo-pager [visible]="true" [showNavigationButtons]="true"> </dxo-pager>
</dx-data-grid>
```

configuration-routing.module

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { AuthGuard } from '../core/auth/auth.guard';

const routes: Routes = [
  {
    path: 'book',
    loadChildren: () => import('./book/book.module').then(m => m.BookModule),
    data: {
      role: 'System Admin',
    },
    canLoad: [AuthGuard],
  },
  {
    path: 'genre',
    loadChildren: () => import('./genre/genre.module').then(m => m.GenreModule),
    data: {
      role: 'System Admin',
    },
    canLoad: [AuthGuard],
  },
  {
    path: '**',
    redirectTo: 'book',
  },
];

@NgModule({
  imports: [RouterModule.forChild(routes)],
  exports: [RouterModule],
})
export class ConfigurationRoutingModule {}
```

auth.guard

```
import { Injectable } from '@angular/core';
import {
  ActivatedRouteSnapshot,
  CanActivate,
  CanLoad,
  Route,
  Router,
  RouterStateSnapshot,
  UrlTree
} from '@angular/router';
import { Observable } from 'rxjs';
import { AuthService } from './auth.service';

@Injectable({
  providedIn: 'root',
})
export class AuthGuard implements CanActivate, CanLoad {
  constructor(private authService: AuthService, private router: Router) {}

  canActivate(
    route: ActivatedRouteSnapshot,
    state: RouterStateSnapshot
  ): Observable<boolean | UrlTree> | Promise<boolean | UrlTree> | boolean | UrlTree
{
    if (this.authService.hasRole(route.data?.role)) {
      return true;
    }

    if (this.authService.isAuthenticated) {
      this.router.navigate(['/']);
    } else {
      this.authService.returnUrl = state.url;
      this.router.navigate(['/signin']);
    }

    return false;
  }

  canLoad(
    route: Route
  ): boolean | UrlTree | Observable<boolean | UrlTree> | Promise<boolean | UrlTree>
{
    return this.authService.hasRole(route.data?.role);
  }
}
```

auth.interceptor

```
import { Injectable } from '@angular/core';
import { HttpRequest, HttpHandler, HttpEvent, HttpInterceptor } from
'@angular/common/http';
import { catchError, Observable, throwError } from 'rxjs';
import { AuthService } from './auth.service';
import { Router } from '@angular/router';

@Injectable()
export class AuthInterceptor implements HttpInterceptor {
  constructor(private authService: AuthService, private router: Router) {}

  intercept(request: HttpRequest<any>, next: HttpHandler):
Observable<HttpEvent<any>> {
    request = request.clone({ withCredentials: true });

    return next.handle(request).pipe(
      catchError(errorResponse => {
        if (errorResponse.status === 401) {
          this.authService.signOut();
          this.router.navigate(['/signin']);
        }
        return throwError(() => errorResponse);
      })
    );
  }
}
```

auth.service

```
import { Injectable } from '@angular/core';
import { NavigationEnd, Router, RouterEvent } from '@angular/router';
import { BehaviorSubject, filter } from 'rxjs';
import { AuthClient, User, UserCredentials } from '../api/api';

@Injectable({
  providedIn: 'root',
})
export class AuthService {
  private _user = new BehaviorSubject<User>(null);
  user$ = this._user.asObservable();
  user: User;
  loggingin$ = new BehaviorSubject(false);
```

```
returnUrl?: string;

constructor(private router: Router, private auth: AuthClient) {
  this.router.events.pipe(filter(e => e instanceof RouterEvent)).subscribe(e => {
    if (e instanceof NavigationEnd) {
      this.loggingin$.next(e.url.includes('signin') || e.url.includes('signup'));
    }
  });

  this.user = this.getUser();
  this._user.next(this.user);
}

get isAuthenticated(): boolean {
  return !!this.user;
}

hasRole(role?: string): boolean {
  if (!role) {
    return false;
  }
  return this.user?.roles?.includes(role);
}

signUp(credentials: UserCredentials) {
  this.auth.signUp(credentials).subscribe({
    next: user => {
      this.setUser(user);
      this.loggingin$.next(false);
      this.router.navigate([this.returnUrl || '/']);
    },
    error: e => {
      console.log(e);
    },
  });
}

signIn(credentials: UserCredentials) {
  this.auth.signIn(credentials).subscribe({
    next: user => {
      this.setUser(user);
      this.loggingin$.next(false);
      this.router.navigate([this.returnUrl || '/']);
    },
    error: e => {
      console.log(e);
    },
```

```
    });
  }

  signOut() {
    this.auth.signOut().subscribe({
      next: response => {
        if (response.status === 200) {
          this.setUser(null);
          this.router.navigate(['']);
        }
      },
      error: e => {
        if (e.status === 401) {
          this.setUser(null);
        }
      },
    });
  }

  private getUser(): User | null {
    try {
      const userJson = sessionStorage.getItem('user');
      if (userJson && userJson !== '') {
        return JSON.parse(userJson);
      }
    } catch (e) {
      this.setUser(null);
    }

    return null;
  }

  private setUser(user?: User): void {
    if (user) {
      sessionStorage.setItem('user', JSON.stringify(user));
    } else {
      sessionStorage.removeItem('user');
    }

    this.user = user;
    this._user.next(this.user);
  }
}
```

notification.service

```
import { Injectable } from '@angular/core';
import { confirm } from 'devextreme/ui/dialog';
import notify from 'devextreme/ui/notify';

@Injectable({
  providedIn: 'root',
})
export class NotificationService {
  public showSuccess(message: string): void {
    notify(
      {
        message: message,
        width: 230,
        position: {
          at: 'bottom',
          my: 'bottom',
        },
      },
      'success',
      500
    );
  }

  public showError(message: string): void {
    notify(
      {
        message: message,
        width: 230,
        position: {
          at: 'bottom',
          my: 'bottom',
        },
      },
      'error',
      1500
    );
  }

  public showConfirmation(message: string, title: string): Promise<boolean> {
    return confirm(message, title);
  }
}
```

login.component

```
import { Component, OnInit } from '@angular/core';
import { FormControl, FormGroup, Validators } from '@angular/forms';
import { Router } from '@angular/router';
import { UserCredentials } from '../../api/api';
import { AuthService } from '../../auth/auth.service';

@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.scss'],
})
export class LoginComponent implements OnInit {
  form: FormGroup;
  hide = true;
  isSignUp: boolean;

  constructor(private authService: AuthService, private router: Router) {
    this.isSignUp = this.router.url === '/signup';
  }

  ngOnInit(): void {
    this.form = new FormGroup({
      email: new FormControl('', [Validators.required, Validators.email]),
      password: new FormControl('', [Validators.required]),
    });
  }

  getEmailErrorMessage() {
    if (this.form.hasError('required', 'email')) {
      return 'You must enter a value';
    }
    return this.form.hasError('email', 'email') ? 'Not a valid email' : 'Error';
  }

  getPasswordErrorMessage() {
    if (this.form.hasError('required', 'email')) {
      return 'You must enter a value';
    }
    return 'Error';
  }

  submit() {
    if (this.form.invalid) {
      return;
    }

    const credentials: UserCredentials = new UserCredentials({
```

```
      email: this.form.controls['email'].value,
      password: this.form.controls['password'].value,
    });

    if (this.isSignUp) {
      this.authService.signUp(credentials);
    } else {
      this.authService.signIn(credentials);
    }
  }
}
```

```
<div class="login-container m-auto mt-5">
  <form [formGroup]="form" (submit)="submit()">
    <div *ngIf="isSignUp; else signinText">
      <h1 class="h3 mb-3 fw-normal text-center">Sign Up</h1>
    </div>
    <ng-template #signinText>
      <h1 class="h3 mb-3 fw-normal text-center">Please sign in</h1>
    </ng-template>
    <mat-form-field appearance="fill" class="w-100">
      <mat-label>Enter your email</mat-label>
      <input matInput placeholder="example@mail.com" formControlName="email" />
      <mat-error *ngIf="this.form.controls['email'].invalid">{{
getEmailErrorMessage() }}</mat-error>
    </mat-form-field>
    <mat-form-field appearance="fill" class="w-100 mb-0">
      <mat-label>Enter your password</mat-label>
      <input matInput [type]="hide ? 'password' : 'text'"
formControlName="password" />
      <button
        mat-icon-button
        matSuffix
        (click)="hide = !hide"
        [attr.aria-label]="'Hide password'"
        [attr.aria-pressed]="hide"
      >
        <mat-icon>{{ hide ? 'visibility_off' : 'visibility' }}</mat-icon>
      </button>
      <mat-error *ngIf="this.form.controls['password'].invalid">{{
getPasswordErrorMessage() }}</mat-error>
    </mat-form-field>
    <button class="w-100" type="submit" mat-raised-button
color="primary">Submit</button>
  </form>
```

```
  <a *ngIf="!isSignUp" class="text-decoration-none link-primary" mat-list-item
routerLink="/signup">
    <h3 class="fw-normal text-center mt-3">Sign Up?</h3>
  </a>
</div>
```

toolbar.component

```
import { Component, EventEmitter, Output } from '@angular/core';
import { Router } from '@angular/router';
import { Subject, takeUntil } from 'rxjs';
import { AuthService } from '../../auth/auth.service';
import { AppConfigService } from '../../config/app-config.service';

@Component({
  selector: 'app-toolbar',
  templateUrl: './toolbar.component.html',
  styleUrls: ['./toolbar.component.scss'],
})
export class ToolbarComponent {
  @Output() toggleSidenav: EventEmitter<void> = new EventEmitter();

  isLoggingin = false;
  destroyed$ = new Subject<boolean>();

  constructor(private router: Router, public authService: AuthService, public
config: AppConfigService) {
    this.authService.loggingin$.pipe(takeUntil(this.destroyed$)).subscribe(loggingi
n => {
      this.isLoggingin = loggingin;
    });
  }

  login() {
    this.authService.returnUrl = this.router.url;
    this.router.navigate(['signin']);
  }

  logout() {
    this.authService.signOut();
  }

  goHome() {
    this.router.navigate(['/']);
  }
}
```

```
<mat-toolbar color="primary">
  <button type="button" aria-label="Toggle sidenav" mat-icon-button
(click)="toggleSidenav.emit()">
    <mat-icon aria-label="Side nav toggle icon">menu</mat-icon>
  </button>
  <div (click)="goHome()">
    <button mat-button [disableRipple]="true">
      <span>{{ config.title }}</span>
    </button>
  </div>
  <div class="w-100"></div>
  <div *ngIf="!isLoggingin">
    <div *ngIf="authService.isAuthenticated; else loginButton">
      <span>{{ authService.user?.email }}</span>
      <button mat-icon-button (click)="logout()">
        <mat-icon class="logout-icon">logout</mat-icon>
      </button>
    </div>
    <ng-template #loginButton>
      <button class="me-5" mat-icon-button [disableRipple]="true"
(click)="login()">
        <span class="fs-5 me-1">Sign in</span>
        <mat-icon>login</mat-icon>
      </button>
    </ng-template>
  </div>
</mat-toolbar>
```

rating-list.component

```
import { Component } from '@angular/core';
import { ActivatedRoute, Router } from '@angular/router';
import CustomStore, { ResolvedData } from 'devextreme/data/custom_store';
import { firstValueFrom, map } from 'rxjs';
import { Book, BookClient, ServiceResultOfListOfBook } from 'src/app/core/api/api';
import { AppConfigService } from 'src/app/core/config/app-config.service';

@Component({
  selector: 'app-rating-list',
  templateUrl: './rating-list.component.html',
  styleUrls: ['./rating-list.component.scss'],
})
export class RatingListComponent {
```

```
dataSource: CustomStore;
updatedBook: Book;

constructor(
  private bookClient: BookClient,
  public config: AppConfigService,
  private router: Router,
  private route: ActivatedRoute
) {
  this.dataSource = new CustomStore({
    key: 'id',
    load: loadOptions => {
      return firstValueFrom(
        this.bookClient.list(loadOptions.skip, loadOptions.take).pipe(
          map((result: ServiceResultOfListOfBook): ResolvedData<Book> => {
            return {
              data: result.data,
              totalCount: result.totalCount,
            };
          })
        )
      );
    }
  });

  this.onCreateClick = this.onCreateClick.bind(this);
}

onRowClick(e: { data: Book }) {
  this.router.navigate(['details', e.data.id], { relativeTo: this.route });
}

calculateGenres(book: Book): string {
  if (book.genres?.length > 0) {
    return book.genres.map(genre => genre.name).join(', ');
  }
  return 'none';
}

calculateRating(book: Book): string {
  if (book.rating === null) {
    return 'none';
  }
  return book.rating.toString();
}

onCreateClick(): void {
```

```
    this.router.navigate(['create'], { relativeTo: this.route });
  }
}
```

```
<dx-data-grid id="dataGrid" [dataSource]="dataSource" [showBorders]="true"
[allowColumnReordering]="true"
  [allowColumnResizing]="true" (onRowClick)="onRowClick($event)"
  [hoverStateEnabled]="true">
  <dxo-toolbar>
    <dxi-item name="columnChooserButton"></dxi-item>
    <dxi-item name="searchPanel"></dxi-item>
  </dxo-toolbar>
  <dxo-column-chooser [enabled]="true"></dxo-column-chooser>
  <dxo-search-panel [visible]="true"></dxo-search-panel>

  <dxi-column dataField="name"></dxi-column>
  <dxi-column dataField="author"></dxi-column>
  <dxi-column dataField="releaseDate" dataType="date"></dxi-column>
  <dxi-column dataField="rating" [calculateCellValue]="calculateRating"></dxi-
column>
  <dxi-column dataField="genres" [calculateCellValue]="calculateGenres"></dxi-
column>

  <dxo-paging [pageSize]="config.pageSize"> </dxo-paging>
  <dxo-pager [visible]="true" [showNavigationButtons]="true"> </dxo-pager>

  <dxo-remote-operations [paging]="true">
  </dxo-remote-operations>
</dx-data-grid>
```

rating-detail-view.component

```
import { Component, OnDestroy, OnInit } from '@angular/core';
import { ActivatedRoute } from '@angular/router';
import { Subject, takeUntil } from 'rxjs';
import { Book, BookClient, BookRate } from 'src/app/core/api/api';
import { AuthService } from 'src/app/core/auth/auth.service';

@Component({
  selector: 'app-rating-detail-view',
  templateUrl: './rating-detail-view.component.html',
  styleUrls: ['./rating-detail-view.component.scss']
})
export class RatingDetailViewComponent implements OnInit, OnDestroy {
```

```
  destroyed$: Subject<boolean>;
  loading: boolean;
  id: string;
  book: Book;
  stars: number[];
  rating = 0;

  constructor(private route: ActivatedRoute, private bookClient: BookClient, public
authService: AuthService) {
    this.destroyed$ = new Subject();

    this.stars = Array(5)
      .fill(0)
      .map((_, i) => i + 1);
  }

  ngOnInit(): void {
    this.id = this.route.snapshot.paramMap.get('id');
    this.loadData(this.id);
  }

  ngOnDestroy(): void {
    this.destroyed$.next(true);
    this.destroyed$.complete();
  }

  loadData(id: string): void {
    this.loading = true;

    this.bookClient
      .get(id)
      .pipe(takeUntil(this.destroyed$))
      .subscribe(result => {
        this.book = result.data;
        this.rating = this.book.userRating;
        this.loading = false;
      });
  }

  onRateSubmit(): void {
    this.loading = true;
    this.book.userRating = this.rating;

    const rate = new BookRate({
      bookId: this.id,
      point: this.rating
    });
```

```
    this.bookClient.rate(rate).subscribe({
      next: () => {
        this.loading = false;
      },
      error: () => {
        this.loading = false;
      }
    });
  }

  resetRating(): void {
    this.rating = this.book?.userRating || 0;
  }
  setRating(value: number): void {
    this.rating = value;
  }
}
```

```
<app-spinner *ngIf="loading; else detailContent"></app-spinner>

<ng-template #detailContent>
  <div class="book-container" *ngIf="!loading">
    <mat-card>
      <mat-card-header>
        <mat-card-title class="mb-4 fs-3">{{ book.name }}</mat-card-title>
        <mat-card-subtitle class="mb-3 fs-6">Author: {{ book.author }}</mat-card-
subtitle>
      </mat-card-header>

      <mat-card-content>
        <p class="mb-3 fs-6"><strong>Release Date:</strong> {{ book.releaseDate |
date }}</p>
        <p class="mb-3 fs-4">Genres</p>
        <p *ngIf="book.genres.length === 0; else genreChips">None</p>
        <ng-template #genreChips>
          <mat-chip-list class="readonly-chips">
            <mat-chip *ngFor="let genre of book.genres" [selected]="true"
[removable]="false" disableRipple>
              {{ genre.name }}
            </mat-chip>
          </mat-chip-list>
        </ng-template>

        <p class="mt-4 mb-3 fs-4">Description</p>
        <p class="text-wrap">{{ book.description }}</p>
```

```
      </mat-card-content>

      <div class="rating-container" *ngIf="authService.isAuthenticated">
        <div class="star-rating" (mouseleave)="resetRating()">
          <ng-container *ngFor="let star of stars;">
            <mat-icon
              class="star-icon"
              (mouseenter)="setRating(star)"
              (click)="onRateSubmit()"
            >
              {{ star <= rating ? 'star' : 'star_border' }}
            </mat-icon>
          </ng-container>
        </div>
        <span class="rating">{{ rating }}</span>
      </div>
    </mat-card>
  </div>
</ng-template>
```

spinner.component

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-spinner',
  templateUrl: './spinner.component.html',
  styleUrls: ['./spinner.component.scss'],
})
export class SpinnerComponent {}
```

```
<div class="spinner-overlay">
  <div class="spinner-container">
    <mat-spinner diameter="50"></mat-spinner>
  </div>
</div>
```

```
.spinner-overlay {
  position: fixed;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
```

```
  display: flex;
  align-items: center;
  justify-content: center;
  background-color: rgba(0, 0, 0, 0.5);
  z-index: 9999;
}

.spinner-container {
  display: flex;
  align-items: center;
  justify-content: center;
}
```

shared.module

```
import { CommonModule } from '@angular/common';
import { NgModule } from '@angular/core';
import { MatProgressSpinnerModule } from '@angular/material/progress-spinner';
import { SpinnerComponent } from './spinner/spinner.component';

@NgModule({
  declarations: [SpinnerComponent],
  imports: [CommonModule, MatProgressSpinnerModule],
  exports: [SpinnerComponent],
})
export class SharedModule {}
```

app.component

```
<mat-sidenav-container class="sidenav-container">
  <mat-sidenav #drawer class="sidenav" fixedInViewport [attr.role]="'dialog'"
[mode]="'over'" [opened]="false">
    <mat-tab-group>
      <mat-tab>
        <ng-template mat-tab-label>
          <mat-icon>apps</mat-icon>
        </ng-template>
        <mat-nav-list>
          <a mat-list-item routerLink="book">Books</a>
        </mat-nav-list>
      </mat-tab>

      <mat-tab *ngIf="isAdmin | async">
        <ng-template mat-tab-label>
          <mat-icon>shield</mat-icon>
```

```
        </ng-template>
        <mat-nav-list>
          <a mat-list-item routerLink="configuration/book">Books</a>
          <a mat-list-item routerLink="configuration/genre">Genres</a>
        </mat-nav-list>
      </mat-tab>
    </mat-tab-group>
  </mat-sidenav>

  <mat-sidenav-content>
    <div class="content-wrapper">
      <app-toolbar (toggleSidenav)="drawer.toggle()"></app-toolbar>
      <div class="content-body pt-5">
        <router-outlet></router-outlet>
      </div>
    </div>
  </mat-sidenav-content>
</mat-sidenav-container>

.sidenav-container {
  height: 100%;

  .content-wrapper {
    height: 100%;
    display: flex;
    flex-direction: column;

    .content-body {
      flex: 1;
    }
  }
}

.sidenav {
  .mat-toolbar {
    background: inherit;
  }
}

.mat-toolbar {
  .mat-primary {
    position: sticky;
    top: 0;
    z-index: 1;
  }
}
```

```
::ng-deep.mat-tab-label,
::ng-deep.mat-tab-label-active {
  min-width: 50px !important;
}
```