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 інформаційна система для самостійного видавництва книг»
Здобувача групи ІН-91 Гончаренка Дмитра Миколайовича

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

Дмитро ГОНЧАРЕНКО
_____
(підпис)

Керівник,
старший викладач кафедри комп'ютерних наук, кандидат фізико-математичних наук Анна БАДАЛЯН
_____
(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 Гончаренка Дмитра Миколайовича

1. Тема роботи: «Веб-орієнтована інформаційна система для самостійного видавництва 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, що використовуються для проектування веб-орієнтованих інформаційних систем. 3) Програмна реалізація інформаційної системи. 4) Аналіз 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 «____» _____ 2023 р.

Завдання прийняв до виконання _____ Керівник _____
(підпис) (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, що використовуються для проектування веб-орієнтованих інформаційних систем*		
3	*Програмна реалізація інформаційної системи*		
4	*Аналіз отриманих результатів*		
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*		

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

АНОТАЦІЯ

Записка: 100 стр., 32 рис., 2 додатки, 16 використаних джерел.

Обґрунтування актуальності теми роботи – Тема кваліфікаційної роботи є актуальною, оскільки присвячена розв’язанню важливої практичної задачі в сфері видавництва книг.

Об’єкт дослідження – веб-орієнтована інформаційна система для самостійного видавництва книг.

Мета роботи – проектування та розробка веб-орієнтованої інформаційної система для самостійного видавництва книг.

Методи дослідження – системи створення та управління базами даних, технології для реалізації веб-орієнтованих інформаційних систем.

Результати – проведено дослідження інформаційних джерел, технологій, які використовуються для побудови веб-застосунків. Завдяки порівняльному аналізу обрано найкращі підходи для виконання поставлених задач. Розроблено веб-орієнтовану інформаційну систему, яка надає можливість авторам публікувати свої книги, а звичайним користувачам – читати їх.

ІНФОРМАЦІЙНА СИСТЕМА, MICROSOFT SQL SERVER, C#, ASP.NET CORE, WEB API, EF CORE, MVC, JWT, TYPESCRIPT, ANGULAR, SEMANTIC UI

ЗМІСТ

ВСТУП

Актуальність:

Традиційний процес видавництва книг вимагає значних фінансових та організаційних зусиль, що обмежує можливість багатьох авторів публікувати свої твори та залучати широку аудиторію. Звідси виникає необхідність у полегшенні процесу публікації та доступу до літератури.

Самвидав – це процес видання та поширення літературних творів без прив'язки до традиційних видавництв. Він дає можливість авторам самостійно контролювати процес створення, редагування та розповсюдження їх творів.

Самвидав книг є важливим явищем в сучасному літературному світі, оскільки він надає авторам значний ступінь незалежності та свободи у вираженні своїх творчих ідей. Завдяки Інтернету, самвидав став доступним для широкої аудиторії, але постійно зростаюча кількість самовидавців ставить перед нами нові виклики і завдання.

Веб-орієнтовані інформаційні системи займають центральне місце в цифровому світі, надаючи користувачам доступ до різноманітної інформації та сервісів у зручний спосіб.

Звідси маємо, що одним із найважливіших аспектів для самвидаву книг є необхідність наявності ефективної інформаційної системи, яка допоможе авторам у керуванні та організації процесу створення, редагування та поширення їх творів. У цьому контексті, розробка веб-орієнтованої інформаційної системи для самвидаву книг стає актуальною та значущою задачею.

Структура. Дана робота складається зі вступу, інформаційного огляду, вибору методів для розв’язання поставленої задачі, програмної реалізації, висновків, списку використаних джерел та додатків.

1. ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1. Принципи побудови інформаційних систем

Інформаційна система (ІС) – це сукупність взаємопов'язаних компонентів, що забезпечують збір, зберігання, обробку, передачу та використання інформації у певному контексті.

Інформаційна система може включати апаратне та програмне забезпечення, бази даних, мережі зв'язку, людські ресурси та процедури обробки інформації. Вона забезпечує автоматизований обмін даними між різними компонентами системи, а також надає можливість користувачам отримувати, аналізувати та використовувати інформацію для досягнення своїх цілей.

Інформаційні системи можуть бути різного типу, включаючи операційні системи, управлінські інформаційні системи, фінансові системи, системи управління виробництвом, системи управління відносинами зі споживачами тощо. Кожен тип системи має свої специфічні функції і може використовуватися для різних цілей у різних сферах діяльності.

Інформаційні системи стали невід'ємною частиною більшості організацій і галузей, оскільки вони допомагають збільшити ефективність, покращити прийняття рішень, забезпечити точність та доступність інформації, а також сприяють автоматизації багатьох процесів.

1.2. Принципи побудови веб-застосунків

Побудова будь-якого веб-застосунку базується на таких основних принципах:

- Клієнт-серверна архітектура.

Веб-застосунок базується на моделі клієнт-сервер, де клієнт (зазвичай веб-браузер) звертається до сервера для отримання даних або виконання операцій. Сервер обробляє запити клієнтів і надсилає їм результати.

- Протоколи передачі даних.

Веб-застосунки використовують різні протоколи передачі даних, такі як HTTP (Протокол передачі гіпертексту) і HTTPS (захищений протокол передачі гіпертексту), для комунікації між клієнтом і сервером.

- Фронтенд.

Фронтенд веб-застосунка відповідає за те, що користувач бачить і взаємодіє з ним. Він створюється завдяки HTML (мова розмітки гіпертексту), CSS (мова каскадних таблиць стилів) і JavaScript, і використовується для створення інтерфейсу користувача.

- Бекенд.

Бекенд веб-застосунка відповідає за обробку логіки, збереження даних і взаємодію з базами даних. Він може бути реалізований за допомогою різних технологій, таких як мови програмування (Python, Java, C#), фреймворки (Django, Ruby on Rails, ASP.NET Core) і системи керування базами даних (MySQL, PostgreSQL, MS SQL Server).

- Безпека.

Веб-застосунки повинні приділяти особливу увагу безпеці, оскільки вони часто піддаються різним видам атак. Для забезпечення безпеки використовуються різні методи шифрування, валідації даних та захисту від вразливостей.

- Масштабованість.

Веб-застосунки повинні бути здатні масштабуватися для обробки зростаючої кількості користувачів і даних.

1.3. Аналіз подібних застосунків

На жаль, в Україні практично відсутні платформи, де автори-початківці можуть розміщувати свої твори. Можна виділити лише одну, функціонал якої є задовільним – Booknet.ua. Детальна характеристика наведена в таблиці 1.1:

Таблиця 1.1 – Аналіз платформи-конкурента.

Характеристика	Опис
Зручність використання інтерфейсу	Користувачам відносно просто навігуватися по платформі, знаходити необхідні функції.
Функціональні можливості	Деякий функціонал є недосконалим – редагування тексту. А деякий навіть надлишковим.
Швидкодія	Сайт відносно швидко відгукується на дії користувачів і обробляє їх запити.
Сумісність з різними пристроями	Платформа працює на різних пристроях, таких як комп'ютери, планшети, мобільні телефони, з різними операційними системами.
Безпека даних	Платформа використовує Cookie, які інколи сповільнюють роботу системи.
Наявність підтримки	Підтримка дуже загальмовано. Відповідь від модераторів інколи можна чекати кілька годин. Слабка модерація контенту.
Масштабованість	Складно визначити масштабованість не маючи доступу до системи, але посилаючись на швидкодію можна зробити висновок, що система достатньо масштабована.

Отже, майже відсутність подібних систем та недосконаліть існуючих відкриває багато простору для створення та впровадження нових.

1.4. Постановка задачі

Веб-орієнтована інформаційна система для самвидаву книг «Library» має відкривати нові перспективи для авторів, дозволяючи їм швидко та ефективно представляти свої твори в онлайн-середовищі.

Важливими аспектами Library повинна бути зручність, доступність та ефективність. Автори мають отримати можливість просувати свої твори в онлайн-середовищі, залучати нову аудиторію. Крім того, необхідно, щоб Library сприяла зменшенню залежності авторів від традиційних видавництв, дозволяючи їм більш вільно виражати свої ідеї та контролювати процес публікації.

Також Library повинна надавати можливість зручного і швидкого доступу до літературних творів для читачів.

Однак, важливо зазначити, що Library може зіштовхнутися з певними викликами та обмеженнями. Обов’язково, потрібно забезпечити безпеку, надійність та доступність системи для користувачів.

Звідси маємо перелік основних завдань:

- Реалізувати надійну та гнучку систему зберігання даних (про книжки, авторів тощо).
- Реалізувати ядро додатку та механізм прийому, обробки та відповідей на запити користувачів.
- Реалізувати зручний Web-інтерфейс.
- Реалізувати систему безпеки для користувачів (Authentication and Authorization).

2. ВИБІР МЕТОДІВ РОЗВ'ЯЗАННЯ ПОСТАВЛЕНОЇ ЗАДАЧІ

2.1. Вибір методів для реалізації системи зберігання даних

Порівняння СУБД наведено в таблиці 2.1.

Таблиця 2.1 – порівняння СУБД

Критерій	MS SQL Server	Oracle	MongoDB	SQLite
Тип	Реляційна	Реляційна	NoSQL	Вбудована
Мова запитів	SQL	SQL	JSON-based	SQL
Схеми	Так	Так	Ні	Ні
Транзакції	Так	Так	Ні	Так
Індекси	Так	Так	Так	Так
Реплікація	Так	Так	Так	Ні
Шардування	Так	Так	Так	Ні
Підтримка JSON	Так	Так	Так	Так
Повний текстовий пошук	Так	Так	Ні	Так
Підтримка ACID	Так	Так	Ні	Так
Строковий контроль доступу	Так	Так	Ні	Ні
Використання пам'яті	Високе	Високе	Середнє	Низьке
Розширюваність	Висока	Висока	Висока	Обмежена

Маємо два оптимальних варіанти MS SQL Server та Oracle. Але у даному застосунку буде використано MS SQL Server так як він краще пристосований при використання у зв'язці з мовою C#.

Microsoft SQL Server [1;7;8] є однією з провідних реляційних баз даних, розроблених компанією Microsoft. Він надає потужні можливості для зберігання, керування та обробки даних у великомасштабних корпоративних середовищах.

Основні властивості Microsoft SQL Server:

- Реляційна модель.

SQL Server базується на реляційній моделі даних, що дозволяє зберігати дані у вигляді таблиць зі зв'язками між ними.

- Масштабованість.

SQL Server підтримує роботу з великими обсягами даних та може бути масштабований для виконання вимог великих підприємств.

- Мова запитів.

SQL Server використовує мову запитів SQL (Structured Query Language), яка дозволяє виконувати різноманітні операції з даними, включаючи створення, читання, оновлення та видалення записів.

- Транзакційна безпека.

SQL Server забезпечує механізми транзакцій для забезпечення цілісності та безпеки даних. Це дозволяє здійснювати групування дій з даними в одну атомарну операцію.

- Безпека та автентифікація.

SQL Server надає різні механізми безпеки, включаючи автентифікацію на рівні сервера та бази даних, доступ до об'єктів на основі ролей та прав доступу.

- Інтеграція з іншими продуктами Microsoft.

SQL Server має сильну інтеграцію з іншими продуктами Microsoft, такими як Visual Studio, .NET Framework та Azure, що робить його популярним в розробці додатків на платформі Microsoft.

Microsoft SQL Server є потужним та надійним рішенням для зберігання та керування даними. Він широко використовується для створення різних застосунків, включаючи веб-додатки.

Entity Framework Core [2;9] (EF Core) – це сучасний та гнучкий інструмент для роботи з базами даних у додатках .NET. Він є популярним інструментом Object-Relational Mapping (ORM), який забезпечує зв'язок між об'єктною моделлю додатка та реляційною базою даних.

EF Core надає різноманітні функції, які дозволяють розробникам працювати з даними за допомогою об'єктно-орієнтованого підходу. Він

автоматично створює таблиці бази даних на основі класів додатка та дозволяє виконувати запити та маніпулювати даними за допомогою об'єктів та LINQ (Language Integrated Query).

Основні можливості EF Core:

- Моделювання даних.

EF Core дозволяє створювати моделі даних, визначаючи класи та їх взаємозв'язки. Він підтримує різні анотації та конфігурації, які дозволяють налаштовувати модель та її властивості.

- Міграції бази даних.

EF Core забезпечує механізм міграцій, який дозволяє автоматично оновлювати структуру бази даних відповідно до змін у моделі даних. Це спрощує процес розгортання та оновлення баз даних.

- Запити та фільтрація.

EF Core надає можливість виконувати запити до бази даних за допомогою LINQ або SQL. Він також підтримує фільтрацію, сортування та групування даних.

- Відстеження змін.

EF Core відстежує зміни в об'єктах моделі та автоматично синхронізує їх з базою даних. Це спрощує збереження змін та оновлення даних.

- Підтримка різних баз даних.

EF Core підтримує кілька постачальників баз даних, включаючи Microsoft SQL Server, SQLite, MySQL, PostgreSQL та інші.

EF Core надає потужний та зручний спосіб роботи з базами даних у .NET-додатках і є одним з найпопулярніших ORM-інструментів для роботи з базами даних в екосистемі .NET.

2.2. Вибір методів для реалізації ядра додатку

ASP.NET Core [3;10;11;12] – це відкрита, платформа для розробки сучасних веб-додатків та сервісів. Вона є наступником платформи ASP.NET.

Основні функції та концепції ASP.NET Core:

- Кросплатформеність.

ASP.NET Core розроблено для роботи на різних платформах, включаючи Windows, macOS та Linux. Це дозволяє розробникам будувати додатки, які можна розгортати на різних операційних системах.

- Висока продуктивність.

ASP.NET Core оптимізовано для високопродуктивних веб-додатків. Він має легку модульну архітектуру, що дозволяє досягати високої швидкодії.

- Патерн MVC.

ASP.NET Core слідує архітектурному патерну Model-View-Controller (MVC), який розділяє додаток на три основні компоненти: модель (дані та бізнес-логіка), представлення (інтерфейс користувача) та контролер (обробка запитів).

- Впровадження залежностей.

ASP.NET Core має вбудовану підтримку для впровадження залежностей, що є шаблоном проектування програмного забезпечення. Це полегшує тестування, підтримку та розширення додатків.

- Middleware.

ASP.NET Core використовує компоненти проміжного програмного забезпечення для обробки запитів та відповідей у конвеєрі. Проміжне програмне забезпечення може виконувати різні завдання, такі як маршрутизація, аутентифікація, журналювання та обробка винятків. Розробники можуть налаштовувати конвеєр проміжного програмного забезпечення під конкретні вимоги додатку.

- Розробка веб-API.

ASP.NET Core має вбудовану підтримку для створення веб-API.

- Інтеграція з сучасними фреймворками для фронтенду.

ASP.NET Core має різні інтеграції з популярними фреймворками для фронтенду, такими як Angular, React та Vue.js.

- Безпека.

ASP.NET Core має потужні функції безпеки, такі як механізми аутентифікації та авторизації. Він підтримує різні методи аутентифікації, включаючи cookie, токени тощо.

- Готовність до хмарних рішень.

ASP.NET Core розроблено з урахуванням роботи в хмарних середовищах. Він підтримує контейнеризацію за допомогою Docker і може легко розгортатися на популярних хмарних платформах, таких як Microsoft Azure та AWS.

Це універсальна платформа, яка надає розробникам потужний набір інструментів для створення сучасних, масштабованих та кросплатформеність веб-додатків та сервісів.

2.3. Вибір методів для реалізації Web-інтерфейсу

Angular [4;13;14] є одним з найпопулярніших фреймворків для розробки веб-додатків. Він розроблений командою Google і надає широкий набір інструментів і можливостей для створення потужних односторінкових та багатосторінкових сайтів.

Основні властивості Angular:

- TypeScript.

Angular використовує TypeScript як основну мову програмування. TypeScript – це розширення JavaScript, яке додає статичну типізацію та нові можливості до JavaScript, що допомагає покращити розробку, підтримку та читабельність коду.

- Компонентна архітектура.

Angular базується на компонентній архітектурі, де додаток будується з малих, самодостатніх компонентів. Кожен компонент містить шаблон (HTML), логіку (Typescript) та стилі(SCSS), що дозволяє організувати додаток у логічні блоки.

- Двостороннє зв'язування даних.

Angular надає потужний механізм двостороннього зв'язування даних між компонентами та їх шаблонами. Це дозволяє автоматично оновлювати дані в шаблоні при зміні даних в компоненті і навпаки.

- Директиви.

Angular має вбудовану систему директив, які дозволяють розширювати HTML та надавати йому нову функціональність. Наприклад, директива *ngFor використовується для ітерації по колекції.

- Сервіси та впровадження залежностей.

Angular сприяє розділенню логіки додатку та її повторному використанню за допомогою сервісів. Сервіси використовуються для обробки бізнес-логіки, спілкування з сервером та спільного використання даних між компонентами. Angular також може впроваджувати сервіси за допомогою механізму впровадження залежностей (Dependency Injection), що спрощує управління залежностями і спрощує тестування додатку.

- Маршрутизація.

Angular надає потужний механізм маршрутизації, який дозволяє визначати шляхи (routes) для різних сторінок або компонентів в додатку. Це дозволяє створювати багатосторінкові додатки.

- Тестування.

Angular має вбудовану підтримку для тестування, що дозволяє легко писати юніт-тести та інтеграційні тести для компонентів, сервісів та інших частин додатку. Це сприяє стабільності, надійності та якості розроблюваного додатку.

- Командна розробка.

Angular пропонує набір інструментів для полегшення командної розробки. Наприклад, Angular CLI (Command Line Interface) надає команди для автоматичного створення компонентів, сервісів, модулів та інших частин додатку, а також для збирання, тестування та розгортання додатку.

- Анімації.

Angular має вбудовану підтримку для анімацій, що дозволяє створювати привабливі та інтерактивні користувацькі інтерфейси. З використанням Angular Animation API можна встановлювати анімаційні ефекти для різних подій та станів компонентів.

Це лише деякі особливості та можливості Angular. Фреймворк дуже гнучкий та потужний, що дозволяє розробникам створювати високоякісні та сучасні веб-додатки.

Semantic UI [5;15] – це фреймворк для розробки користувацького інтерфейсу, який надає готові компоненти, стилі та шаблони для створення привабливих та сучасних веб-додатків. Він побудований на основі семантичного HTML та CSS, що дозволяє забезпечити зрозумілу та легко змінювану структуру коду.

Основні властивості Semantic UI:

- Семантичний HTML.

Semantic UI використовує семантичні класи для елементів HTML, що полегшує розуміння структури сторінки та її компонентів.

- Готові компоненти.

Semantic UI надає широкий набір готових компонентів, таких як кнопки, форми, меню, модальні вікна та багато інших. Ці компоненти можна легко використовувати та налаштовувати, що прискорює розробку інтерфейсу.

- Адаптивний дизайн.

Semantic UI має вбудовану підтримку адаптивного дизайну, що дозволяє легко створювати інтерфейси, які пристосовуються до різних розмірів екранів та пристроїв. Це забезпечує коректне відображення додатків на різних пристроях, включаючи комп'ютери, планшети та смартфони.

- Розширюваність.

Semantic UI дозволяє легко розширювати та налаштовувати компоненти та стилі. Фреймворк надає можливість створювати власні класи, додавати додаткові стилі та перевизначати існуючі правила. Це дозволяє пристосовувати вигляд інтерфейсу до власних потреб.

- Шаблони та теми.

Semantic UI містить багато шаблонів та тем, які допомагають швидко налаштовувати зовнішній вигляд додатків. Шаблони дозволяють створити базову структуру сторінки з готовими компонентами, а теми дозволяють легко змінювати кольорову схему та стиль компонентів.

- Вбудовані анімації.

Semantic UI має вбудовану підтримку анімацій, що дозволяє створювати привабливі та динамічні ефекти в інтерфейсі. Це допомагає зробити додаток більш інтерактивним для користувачів.

- Кросбраузерна підтримка.

Semantic UI забезпечує підтримку різних браузерів, включаючи останні версії Chrome, Firefox, Safari, Edge та інших популярних браузерів. Це дозволяє забезпечити сумісність та однаковий вигляд додатків на різних платформах.

- Інтеграція з іншими фреймворками.

Semantic UI може легко інтегруватись з іншими фреймворками, такими як React або Angular. Це дозволяє поєднувати можливості Semantic UI з іншими інструментами та технологіями для розробки додатків.

Semantic UI - це потужний фреймворк, який допомагає швидко створювати естетичний та функціональний користувацький інтерфейс для веб-додатків.

2.4. Вибір методів для реалізації системи безпеки

JSON Web Token (JWT) [6] – це відкритий стандарт (RFC 7519) для представлення безпечних токенів у форматі JSON. JWT використовується для аутентифікації та авторизації в розподілених системах, зокрема у веб-додатках та API.

JWT складається з трьох частин: заголовку (header), тіла (payload) та підпису (signature). Кожна частина закодована у форматі Base64 URL-safe і розділена крапкою.

Переваги JWT:

- Легкість.

JWT має простий формат у форматі JSON, що робить його легким у використанні та передачі.

- Незалежність від стану.

JWT зберігає всю необхідну інформацію у самому токені, тому не потребує зберігання стану на сервері. Це дозволяє легко масштабувати систему та розділяти її на різні сервіси.

- Перевірка цілісності.

JWT містить підпис, який перевіряє цілісність токену. Це дозволяє перевірити, чи токен не був змінений або підроблений.

- Розширюваність.

JWT може містити додаткові поля (claims), які можуть бути використані для передачі додаткової інформації про користувача або додаток.

Щоб перевірити токен, сервер розкодовує та перевіряє підпис та інші поля токену. При цьому сервер може отримати інформацію про користувача або додаток , яка міститься у полі payload токену. Таким чином, JWT дозволяє передавати інформацію про ідентифікацію та авторизацію без необхідності зберігання цієї інформації на сервері.

JWT може бути використаний у різних сценаріях, таких як:

- Аутентифікація.

Після успішної аутентифікації користувача сервер може видати JWT-токен, який містить інформацію про користувача (наприклад, його ідентифікатор або роль). Цей токен може бути переданий у заголовку або тілі запиту при кожній наступній взаємодії з сервером для підтвердження ідентичності користувача.

- Авторизація.

JWT може містити інформацію про ролі або права доступу користувача. Сервер може перевірити ці дані у токені та дозволити чи заборонити доступ до певних ресурсів або функціональності в залежності від прав користувача.

- Обмін даними між службами.

JWT може використовуватись для безпечного обміну даними між різними службами або мікросервісами в розподіленій системі. Токен може містити певну інформацію про запит та дозволити доступ до ресурсів або послуг у межах обміну.

- Захист API.

JWT може бути використаний для захисту API, де токен передається з кожним запитом і перевіряється на сервері для підтвердження дійсності та ідентичності користувача.

Використання JWT вимагає належного забезпечення конфіденційності та цілісності токенів, оскільки вони можуть містити чутливу інформацію. Також, важливо враховувати термін дії токенів і їх поновлення для забезпечення безпеки.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1. Моделювання системи

Use Case Diagram – це діаграма, що ілюструє функціональність системи та взаємодію акторів (користувачів або інші зовнішніх елементів) з цією системою. Вона використовується для моделювання вимог до системи та визначення її основних функцій.

UCD складається з таких компонентів:

- Актори.

Зовнішні сутності, які взаємодіють з системою. Наприклад, користувачі, адміністратори тощо.

- Варіанти використання.

Варіанти використання описують окремі функціональні можливості системи. Кожен варіант використання пов'язується з одним або декількома акторами та описує послідовність подій, які відбуваються при взаємодії актора з системою.

- Відношення між акторами та варіантами використання.
- Система.

У системі будуть відповідні сутності:

Актори:

- Незареєстрований користувач.
- Автор відповідної книги.
- Адміністратор.
- MS SQL Server.

Варіанти використання:

- Реєстрація.
- Авторизація.
- Перегляд списку книг.

- Перегляд книги.
- Пошук книги.
- Видалення книги.
- Створення книги.
- Редагування книги.

Результат моделювання зображено на Рисунку 3.1.

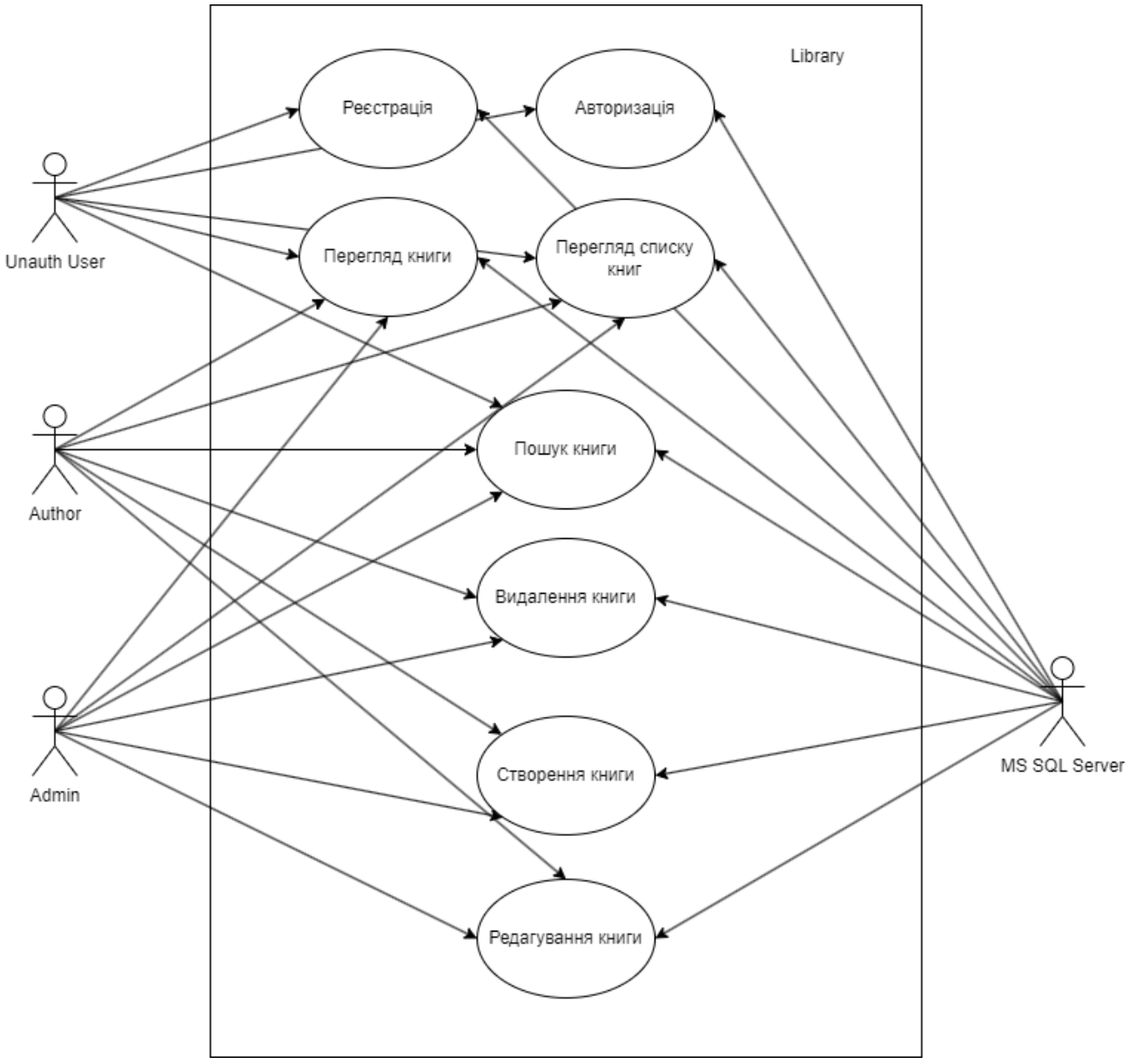

Рисунок 3.1 – UCD системи Library

3.2. Структура кодової бази

Уся кодова база знаходиться у теці «Library». Вона розбита на дві основні частини «webapi» (бекенд) та «angularapp» (фронтенд).

Бекенд (Додаток А) складається з трьох частин:

Library – головний проект рішення, точка збору та запуску програми. Структуру проекту Library зображено на Рисунку 3.2.

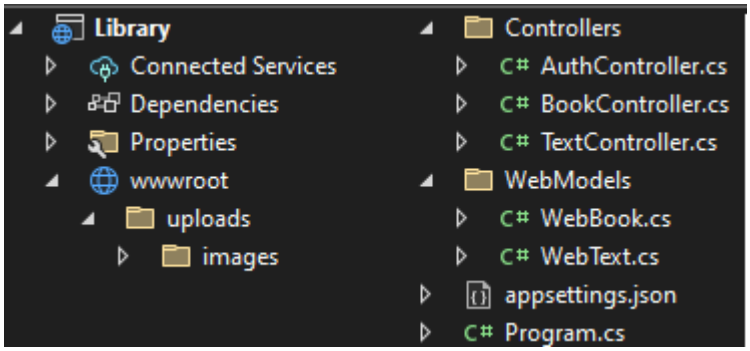

Рисунок 3.2 – Структура проекту Library

Library.DAL – бібліотека з сервісами, репозиторіями, контекстом та міграціями. Структуру проекту Library.DAL зображено на Рисунку 3.3.

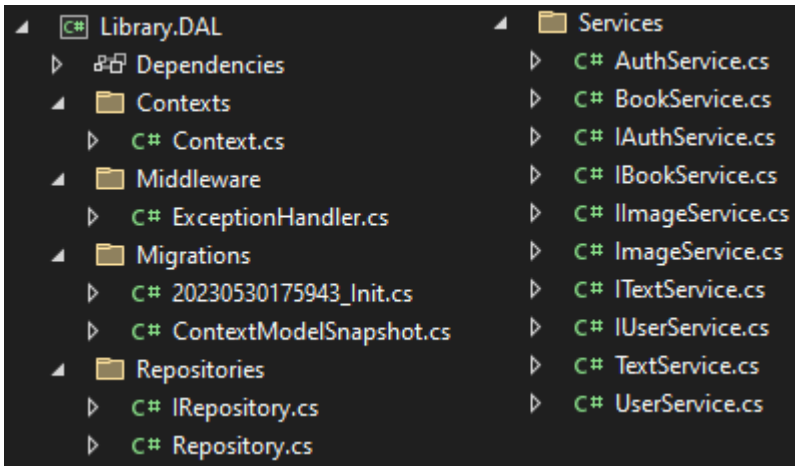

Рисунок 3.3 – Структура проекту Library.DAL

Library.Domain – бібліотека з класами моделі, константами, допоміжними класами. Структуру проекту Library.Domain зображено на Рисунку 3.4.

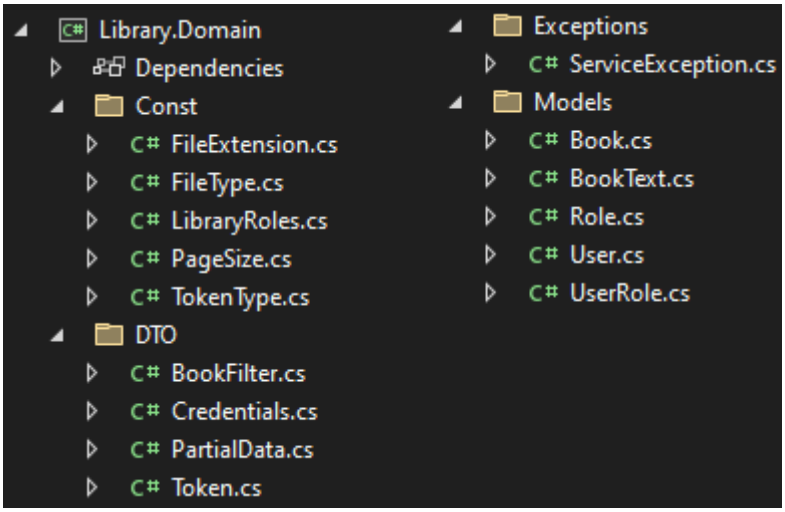

Рисунок 3.4 – Структура проекту Library.Domain

Фронтенд (Додаток Б) складається з чотирьох частин:

Components – Angular компоненти, головна частина фреймворку Angular. Структуру папки Components зображено на Рисунку 3.5.

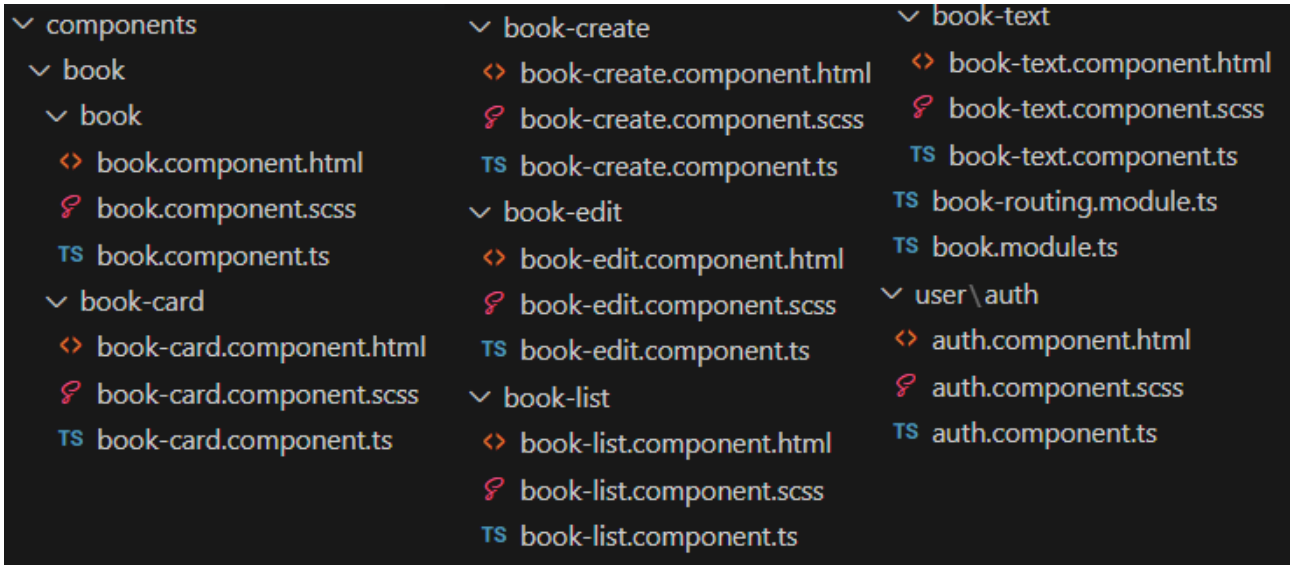

Рисунок 3.5 – Структура папки components

Interceptors – перехоплювачі запитів. Структуру папки Interceptors зображено на рисунку 3.6.

Models – моделі. Структуру папки Models зображено на рисунку 3.6.

Services – сервіси, які передаються запити на бекенд. Структуру папки Services зображено на Рисунку 3.6.

Рисунок 3.6 – Структура папок models, service, interceptors

3.3. Реалізація системи зберігання даних.

Для реалізації системи зберігання даних було використано MS SQL Server та Entity Framework Core з підходом Code First.

Code First – це підхід до розробки баз даних, що базується на Entity Framework Core. Замість того, щоб спочатку визначати схему бази даних і потім генерувати модель даних на її основі, спочатку визначається модель даних (класи або сутності) у .NET-проекті, а потім EF Core автоматично створює відповідну схему бази даних за цими моделями.

Це надає зручність і простоту в розробці, оскільки можна сфокусуватися на дизайні домену та логіці програми, а EF Core буде відповідати за генерацію схеми бази даних і взаємодію з нею.

У папці [Library\webapi\Library.Domain\Models\] знаходяться файли з класами які використовуються як модель для Entity Framework Core, див. Додаток А.1. Структура папки показана на Рисунку 3.7.

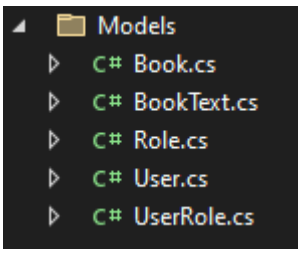

Рисунок 3.7 – Файли з класами для моделі бази даних

У файлі [Library\webapi\Library.DAL\Contexts\Context.cs] реалізовано логіку створення бази даних з відповідними таблицями, див. Додаток А.2. Сутності таблиць показано на Рисунку 3.8.

```
0 references
public DbSet<Book> Books { get; set; }
0 references
public DbSet<BookText> BookTexts { get; set; }
0 references
public DbSet<User> Users { get; set; }
0 references
public DbSet<Role> Roles { get; set; }
0 references
public DbSet<UserRole> UserRoles { get; set; }
```

Рисунок 3.8 – Сутності бази даних

Метод класу Context.OnModelCreating – метод в якому реалізована логіка зв’язків між сутностями (наприклад у одного автора може бути багато книжок).

Одна з ключових особливостей EF Core Code First - це міграції. Міграції використовуються для збереження та застосування змін в схемі бази даних. Можна впроваджувати міграції для створення початкової схеми бази даних, додавання нових таблиць, зміни або видалення полів тощо. EF Core розрізняє, які зміни вже застосовані до бази даних і автоматично генерує SQL-скрипти для нових міграцій.

У папці [Library\webapi\Library.DAL\Migrations\] знаходяться файли міграцій згенеровані автоматично Entity Framework Core. Щоб створити нову

міграцію достатньо в консолі написати команду «add-migration Migration-Name» а Entity Framework Core виконає відповідні дії генерації її коду.

У точці запуску програми (файл [Library\webapi\Library\Program.cs], див. Додаток А.3) міграції впроваджуються завдяки ділянці коду, яку показано на Рисунку 3.9.

```
using (var scope = app.Services.CreateScope())
{
    var serviceProvider = scope.ServiceProvider.GetRequiredService<Context>();
    serviceProvider.Database.Migrate();
}
```

Рисунок 3.9 – Впровадження міграцій

Результатом буде створення нової бази даних, якщо її ще не існує або впровадження відповідних змін.

Маємо структуру бази даних додатку на MS SQL Server, яку показано на Рисунку 3.10.

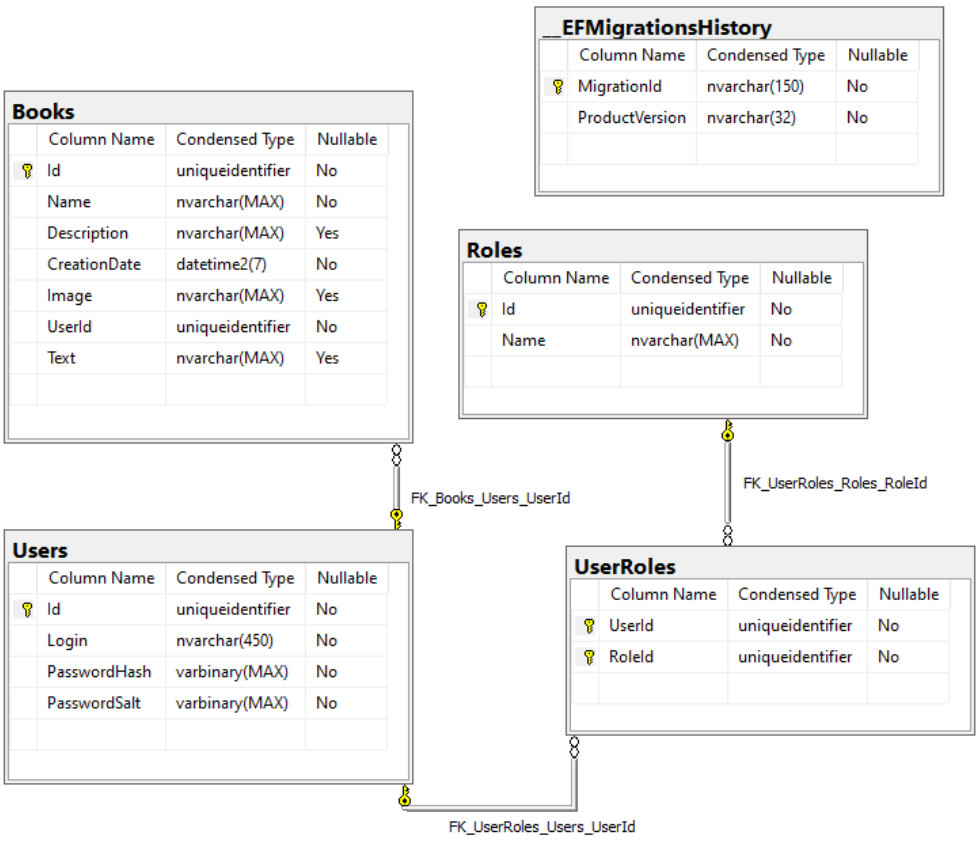

Рисунок 3.10 – Структура бази даних librarydb

Також реалізована підтримка статичних файлів які розміщуються у файловій системі сервера [Library\webapi\Library\wwwroot\uploads\]. Структура папки зі статичними файлами зображена на Рисунку 3.11.

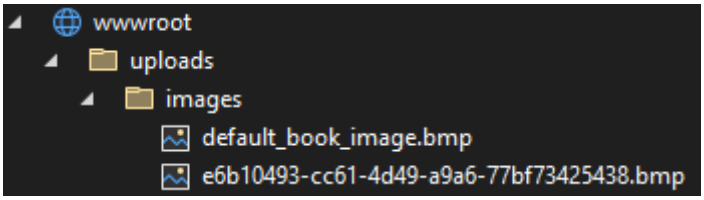

Рисунок 3.11 – Папка із статичними файлами

3.4. Реалізація ядра додатку

Застосунок «Library» має в своїй основі патерн MVC.

Model-View-Controller (Модель-Вид-Контролер) – це архітектурний шаблон програмного забезпечення широко використовується для розробки веб- та десктоп-додатків. Він розділяє логіку програми на три взаємопов'язані компоненти:

Model: Модель представляє дані та бізнес-логіку програми. Вона інкапсулює дані та проводить операції з ними, такі як збереження, оновлення, видалення та отримання.

View: Вид відповідає за відображення даних моделі користувачу. Він представляє інтерфейс користувача, з яким користувач може взаємодіяти, і відображає дані з моделі у зрозумілому для користувача вигляді.

Controller: Контролер приймає вхідні дані від користувача, взаємодіє з моделлю для оновлення даних та керує взаємодією між моделлю та видом. Він виконує логіку додатку, обробляє події та передає відповідні команди моделі та виду.

Застосування шаблону MVC дозволяє розділити відповідальності між компонентами програми, полегшуючи розробку, тестування та зміну коду. Крім того, цей шаблон сприяє покращенню повторного використання коду і сприяє структуруванню додатку.

Загальна структура Бекенду зображена на Рисунку 3.12.

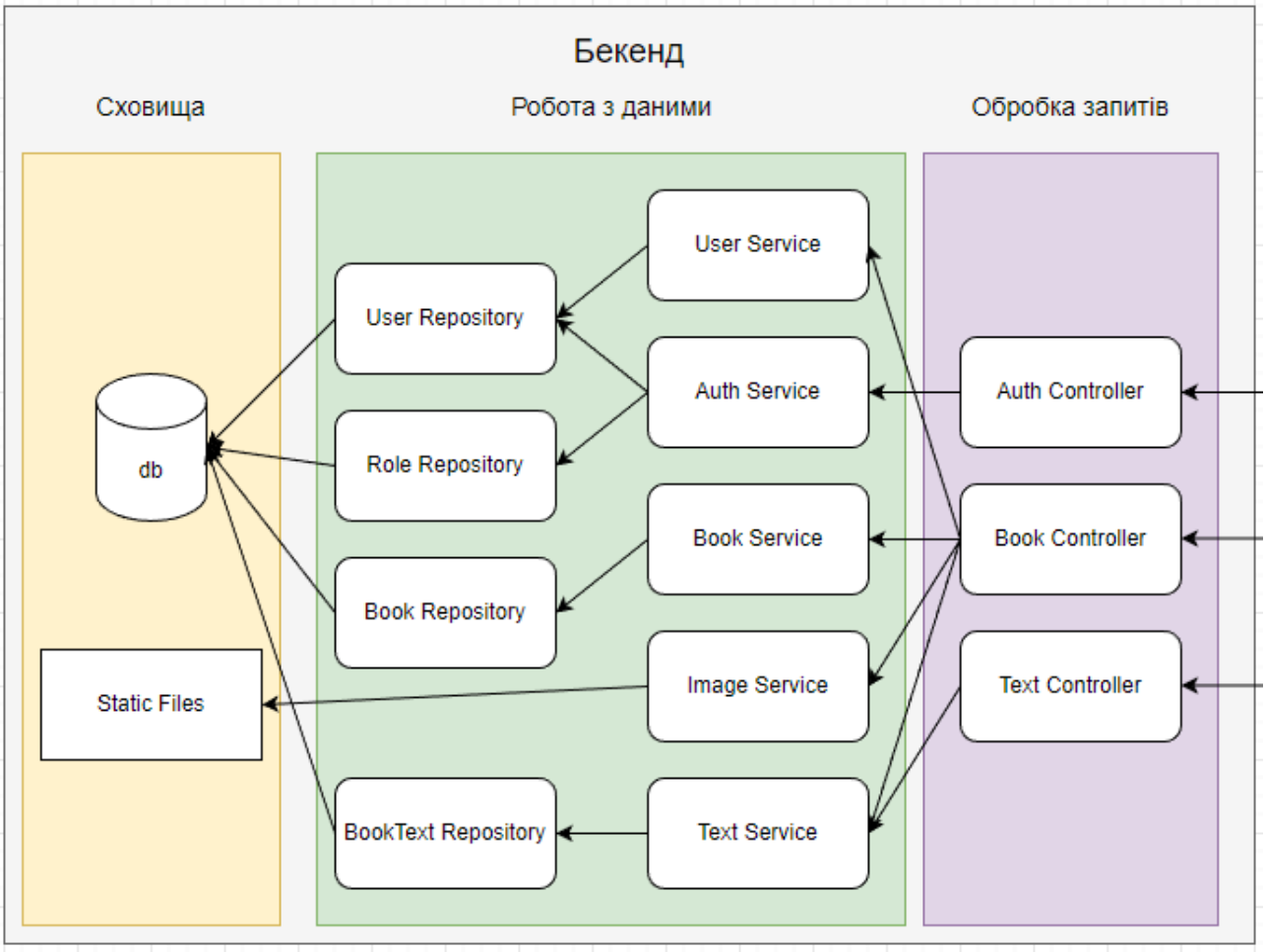

Рисунок 3.12 – Структура бекенду

Подальша взаємодія з даними з БД буде відбуватися за допомогою патерну Repository.

Repository є шаблоном проектування, який використовується для розділення логіки доступу до даних від бізнес-логіки додатку. Він надає абстракцію між доменною моделлю додатку і джерелом даних.

Основна ідея патерну Repository полягає в тому, щоб створити інтерфейс або абстрактний клас, який визначає набір методів для роботи з даними, таких як створення, зчитування, оновлення або видалення. Потім реалізувати цей інтерфейс або клас у конкретному репозиторії, який взаємодіє з фактичним джерелом даних.

Переваги використання патерну Repository:

- Розділення відповідальності.

Патерн дозволяє розділити бізнес-логіку додатку від логіки доступу до даних. Це полегшує тестування і підтримку коду, оскільки зміни в логіці доступу до даних не впливають на бізнес-логіку та навпаки.

- Покращена переносимість.

Репозиторій надає абстракцію для доступу до даних, що дозволяє легко змінювати джерело даних без необхідності внесення змін у коді, який використовує цей репозиторій. Наприклад, можна замінити репозиторій бази даних на репозиторій, що працює зі зовнішнім веб-сервісом, а бізнес-логіка додатку залишиться незмінною.

- Чистіший та краще організований код.

Використання патерну Repository допомагає уникнути розсіяння коду, оскільки всі операції з даними зосереджені в одному місці - репозиторії. Це полегшує розуміння та підтримку коду, зменшує ймовірність помилок і сприяє збереженню його чистоти.

- Можливість застосування додаткової логіки.

Репозиторій може служити місцем для виконання додаткової логіки, яка пов'язана з доступом до даних. Наприклад, можна додати кешування, логування або перехоплення помилок в репозиторії, що спрощує роботу з даними на рівні додатку.

- Спрощене тестування.

Завдяки розділенню бізнес-логіки від логіки доступу до даних, можна легко створювати тести для бізнес-логіки додатку, використовуючи мок-об'єкти або фейк-репозиторії. Це дозволяє ефективно тестувати різні сценарії без прив'язки до реальної бази даних.

Загалом, патерн Repository дозволяє розділити логіку доступу до даних від бізнес-логіки, полегшує тестування, підтримку та переносимість коду. Використання цього патерну сприяє чистоті та організації коду, а також дозволяє застосовувати додаткову логіку на рівні доступу до даних.

Для зменшення кількості коду було створено універсальний (generic) Repository<TEntity, TKey>. Де TEntity – сутність (книга, автор), а TKey – тип ідентифікатора (у додатку використано рекомендована структура Microsoft – Guid).

У папці [Library\webapi\Library.DAL\Repositories\] знаходиться інтерфейс і клас універсального репозиторію, див. Додаток А.4. Методи інтерфейсу зображено на Рисунку 3.13.

```
15 references
public interface IRepository<TEntity, TKey> : IDisposable where TEntity : class
{
    2 references
    Task<PartialData<TEntity>> GetItemListAsync(Expression<Func<TEntity, bool>>? predicate,
        int? skip, int? take, params Expression<Func<TEntity, object?>>[] includes);
    6 references
    Task<TEntity?> GetItemAsync(TKey id);
    5 references
    Task<TEntity> GetItemAsync(Expression<Func<TEntity, bool>> predicate,
        params Expression<Func<TEntity, object?>>[] includes);
    3 references
    Task CreateAsync(TEntity entity);
    2 references
    void Update(TEntity entity);
    2 references
    Task DeleteAsync(TKey id);
    5 references
    Task SaveAsync();
}
```

Рисунок 3.13 – Інтерфейс IRepository

Основна логіка програми знаходиться в сервісах у папці [Library\webapi\Library.DAL\Services\]. Структуру папки показано на Рисунку 3.14.

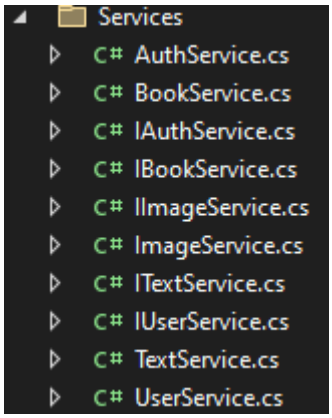

Рисунок 3.14 – Структура папки Services

Кожен сервіс відповідає за конкретні завдання (бізнес-логіку).

Auth Service (Додаток А.5):

- Реєстрація нового користувача.
- Авторизація користувача в системі.
- Валідація облікових даних.

Book Service (Додаток А.6):

- Створення книг.
- Оновлення книг.
- Видалення книг.
- Знаходження книги за певними параметрами.
- Знаходження списку книг.

Image Service (Додаток А.7):

- Валідація вхідних даних.
- Завантаження обкладинки книги.
- Заміна обкладинки книги.
- Видалення обкладинки книги.
- Рефакторинг вхідного зображення.

Text Service (Додаток А.8):

- Валідація вхідних даних.
- Зчитування з файлу (який надає користувач) тексту книги та занесення його до бази даних.
- Розбиття тексту книги на сторінки.
- Виведення розбитого на сторінки тексту.

User Service (Додаток А.9):

- Знаходження користувачів за їх клеймами.
- Валідація користувачів.

Кожен сервіс та репозиторій реєструються в додатку за допомогою функціоналу ASP.NET Core у точці запуску програми, файл

[Library\webapi\Library\Program.cs], див. Додаток А.3. Додавання сервісів продемонстровано на Рисунку 3.15.

```
builder.Services.AddScoped<IUserService, UserService>();
builder.Services.AddScoped<IAuthService, AuthService>();

builder.Services.AddScoped<IBookService, BookService>();
builder.Services.AddScoped<IImageService, ImageService>();
builder.Services.AddScoped<ITextService, TextService>();

builder.Services.AddScoped<IRepository<User, Guid>, Repository<User, Guid>>();
builder.Services.AddScoped<IRepository<Role, Guid>, Repository<Role, Guid>>();

builder.Services.AddScoped<IRepository<Book, Guid>, Repository<Book, Guid>>();
builder.Services.AddScoped<IRepository<BookText, Guid>, Repository<BookText, Guid>>();
```

Рисунок 3.15 – Реєстрація сервісів та репозиторіїв

Обробка запитів які надходять від користувачів відбувається в контролерах у папці [Library\webapi\Library\Controllers\]. Структура папки показана на Рисунку 3.16.

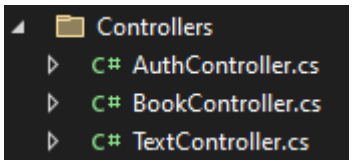

Рисунок 3.16 – Структура папки Controllers

Кожен контролер відповідає за конкретні завдання (конкретну сукупність запитів).

Auth Controller (Додаток А.10):

- Запит на реєстрацію.
- Запит на авторизацію.

Book Controller (Додаток А.11):

- Запит на виведення списку книг.
- Запит на виведення докладної інформації про книги.
- Запит на створення нової книги.
- Запит на редагування існуючої книги.

- Запит на видалення книги.

Text Controller (Додаток А.12):

- Запит на виведення тексту книги.

На контролери також можуть надходити форми які структуровані у відповідні класи, див. Додаток А.13.

Також було реалізовано тек зване середовище проміжного програмного забезпечення (Middleware). Для ASP.NET Core Web API воно є важливою складовою, яка дозволяє обробляти HTTP-запити з великими можливостями налаштування та розширення. Проміжне програмне забезпечення розташовується між веб-сервером та потоком обробки запитів додатку, що дозволяє перехоплювати, змінювати або відповідати на запити.

Компоненти проміжного програмного забезпечення виконуються послідовно в тому порядку, в якому вони додаються до потоку обробки, кожен компонент має можливість обробляти вхідний запит та опціонально передавати його наступному компоненту або скорочувати потік обробки та генерувати відповідь безпосередньо.

Якщо під час виконання запиту користувача виникне помилка то вона буде оброблена за допомогою Middleware – реалізованого класом ExceptionHandler, див. Додаток А.14.

При створенні бекенду були застосовані певні допоміжні класи, константи, перечислення, див. Додаток А.15.

3.5. Реалізація Web-інтерфейсу

Весь фронтенд можна розділити на декілька частин, які продемонстровано на Рисунку 3.17.

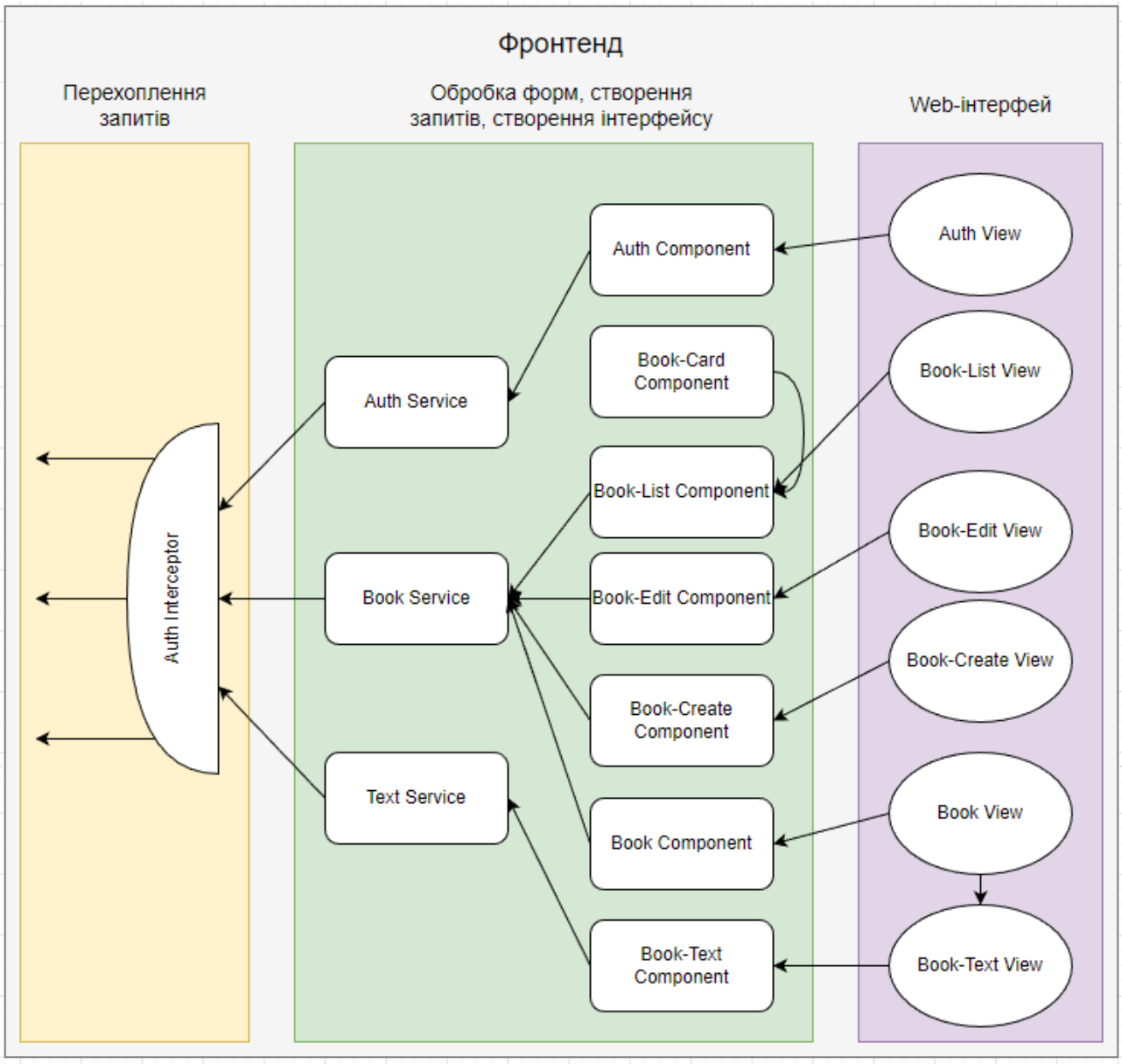

Рисунок 3.17 – Структура фронтенду

Кожен сервіс відповідає за надсилання запитів на сервер. Папка з сервісами [Library\angularapp\src\app\services\].

Auth Service (Додаток Б.1):

- Запит на реєстрацію.
- Запит на авторизацію.

Book Service (Додаток Б.2):

- Запит на виведення списку книг.
- Запит на виведення докладної інформації про книги.
- Запит на створення нової книги.

- Запит на редагування існуючої книги.
- Запит на видалення книги.

Text Service (Додаток Б.3):

- Запит на виведення тексту книги.

Перехоплення запитів (Interceptor, див. Додаток Б.4) спрямоване на те, щоб до кожного запиту додавати якусь потрібну інформацію. Наприклад токен авторизації [Library\angularapp\src\app\interceptors\auth.interceptor.ts]. Клас перехоплення зображено на Рисунку 3.18.

```
@Injectable()
export class AuthInterceptor implements HttpInterceptor {
    intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
        const token = localStorage.getItem('authToken');

        if (token) {
            req = req.clone({
                setHeaders: { Authorization: `Bearer ${token}` }
            });
        }

        return next.handle(req);
    }
}
```

Рисунок 3.18 – Додавання токену авторизації до запитів

Основною частино фронтенду є компоненти (Components), які можуть об’єднуватися в модулі (Modules). У додатку є два модулі App – головний модуль додатку, Book – модуль, який об’єднує компоненти пов’язані з сутністю книги.

Component – це будівельні блоки додатку, які складаються з HTML-шаблону, CSS-стилів і TypeScript-коду. Кожен компонент відповідає за певну функціональність та вигляд елемента інтерфейсу додатку. Наприклад, компонент може бути кнопкою, формою, таблицею, меню або будь-яким іншим елементом інтерфейсу.

Компоненти в Angular можуть мати вхідні дані (Input) та вихідні дані (Output). Вхідні дані – це дані, які передаються в компонент ззовні і

використовуються для відображення або обробки внутрішнього стану компонента. Вихідні дані – це дані, які компонент може відправляти назовні, наприклад, після певної дії користувача.

Компоненти в Angular також можуть мати свої власні стилі, що дозволяє легко налаштовувати вигляд компонента. Крім того, Angular надає можливість розширювати компоненти за допомогою директив, які впливають на поведінку та вигляд компонентів.

З використанням компонентного підходу Angular дозволяє створювати додатки, що легко масштабувати та підтримувати, оскільки кожен компонент виконує свою чітко визначену функцію.

Компоненти знаходяться у папці [Library\angularapp\src\app\components\]. App (Додаток Б.5) – головний компонент програми.

Auth (Додаток Б.6):

- Створення сторінки реєстрації/авторизації.
- Створення форми реєстрації/авторизації.
- Обробка форми реєстрації.
- Надсилання запиту реєстрації на сервер.
- Обробка форми авторизації.
- Надсилання запиту авторизації на сервер.
- Отримання токену авторизації.

Book-Card (Додаток Б.7):

- Створення карток.

Book-List (Додаток Б.8):

- Надсилання запиту на сервер для виведення списку книг.
- Створення сторінки списку карток книг.

Book (Додаток Б.9):

- Надсилання запиту на сервер для виведення докладної інформації про книгу.
- Створення сторінки книги.

- Надсилання запиту на сервер для видалення книги.

Book-Text (Додаток Б.10):

- Надсилання запиту на сервер для виведення тексту книги.
- Створення елементу тексту.

Book-Create (Додаток Б.11):

- Створення сторінки створення нової книги.
- Створення форми створення нової книги.
- Надсилання запиту створення нової книги на сервер.

Book-Edit (Додаток Б.12):

- Створення сторінки редагування книги.
- Створення форми редагування книги.
- Надсилання запиту редагування книги на сервер.

Для кожного компоненту є своє візуальне представлення (View).

Auth:

Форми реєстрації/авторизації з відповідними полями (Login, Password). Валідація лежить на серверній частині. Якщо валідацію провалено то сервер поверне помилку з відповідним повідомленням інакше буде створено нового користувача/користувач увійде в систему. Вигляд форми авторизації зображено на Рисунку 3.19, форми реєстрації – Рисунок 3.20.

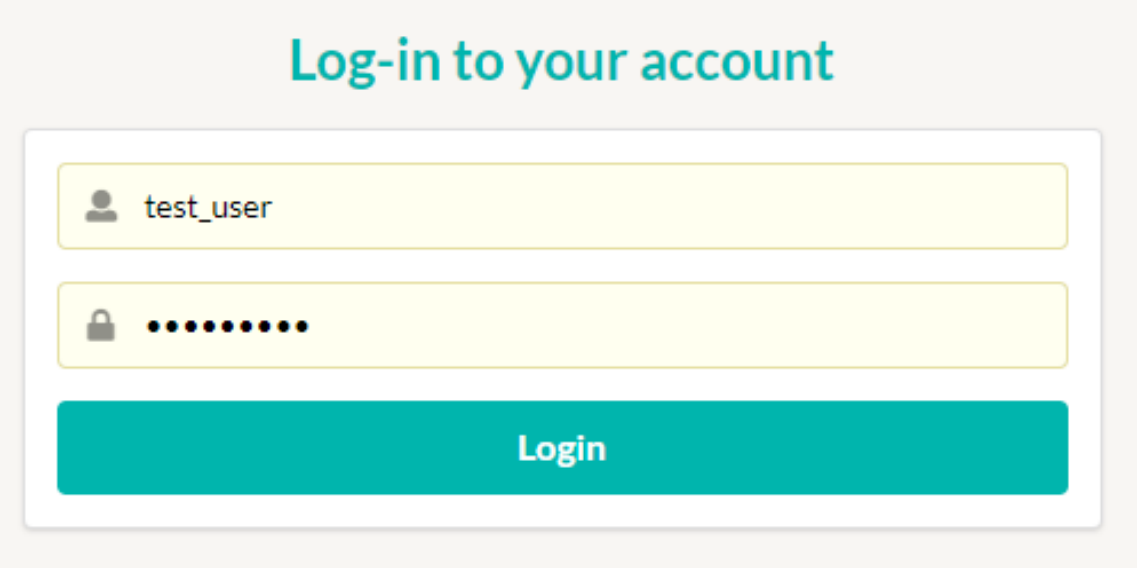

Рисунок 3.19 – Форма авторизації

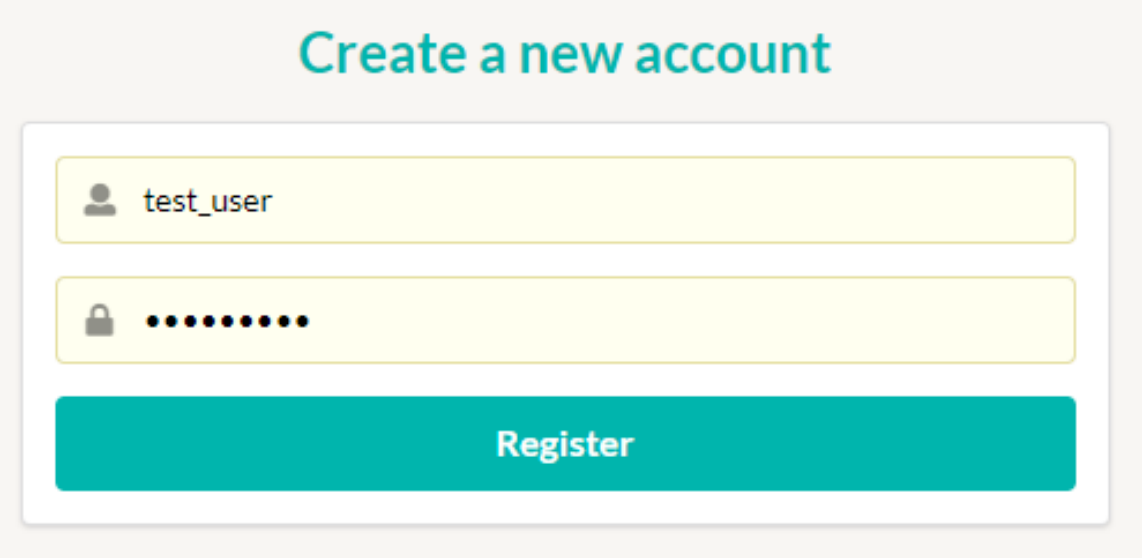

Рисунок 3.20 – Форма реєстрації

Book-List частиною якого є Book-Card:

Головною сторінкою сайту є сторінка зі списком книг. Header складається з кнопок навігації/командних кнопок:

Library – кнопка переходу на головну сторінку.

Create New Book – кнопка переходу на сторінку створення книги (якщо користувач авторизований).

Login – кнопка переходу на сторінку авторизації (якщо користувач не авторизований).

Register – кнопка переходу на сторінку реєстрації (якщо користувач не авторизований) абоз user_name – login користувача (якщо користувач авторизований).

Logout – кнопка виходу з системи (якщо користувач авторизований).

Вигляд Header для незареєстрованого користувача зображено на Рисунку 3.21.

Рисунок 3.21 – Header незареєстрованого користувача

Список книг складається з карток. Картки книг складаються з короткого опису, обкладинки, та є кнопкою навігації на сторінку книги. Якщо обкладинки до книги не додано то замість неї буде стандартна заглушка. Якщо поточний користувач є автором книги це відобразиться на її картці. Список книг зображено на Рисунку 3.22, а список книг, які влаштовують критерій пошуку – Рисунок 3.23.

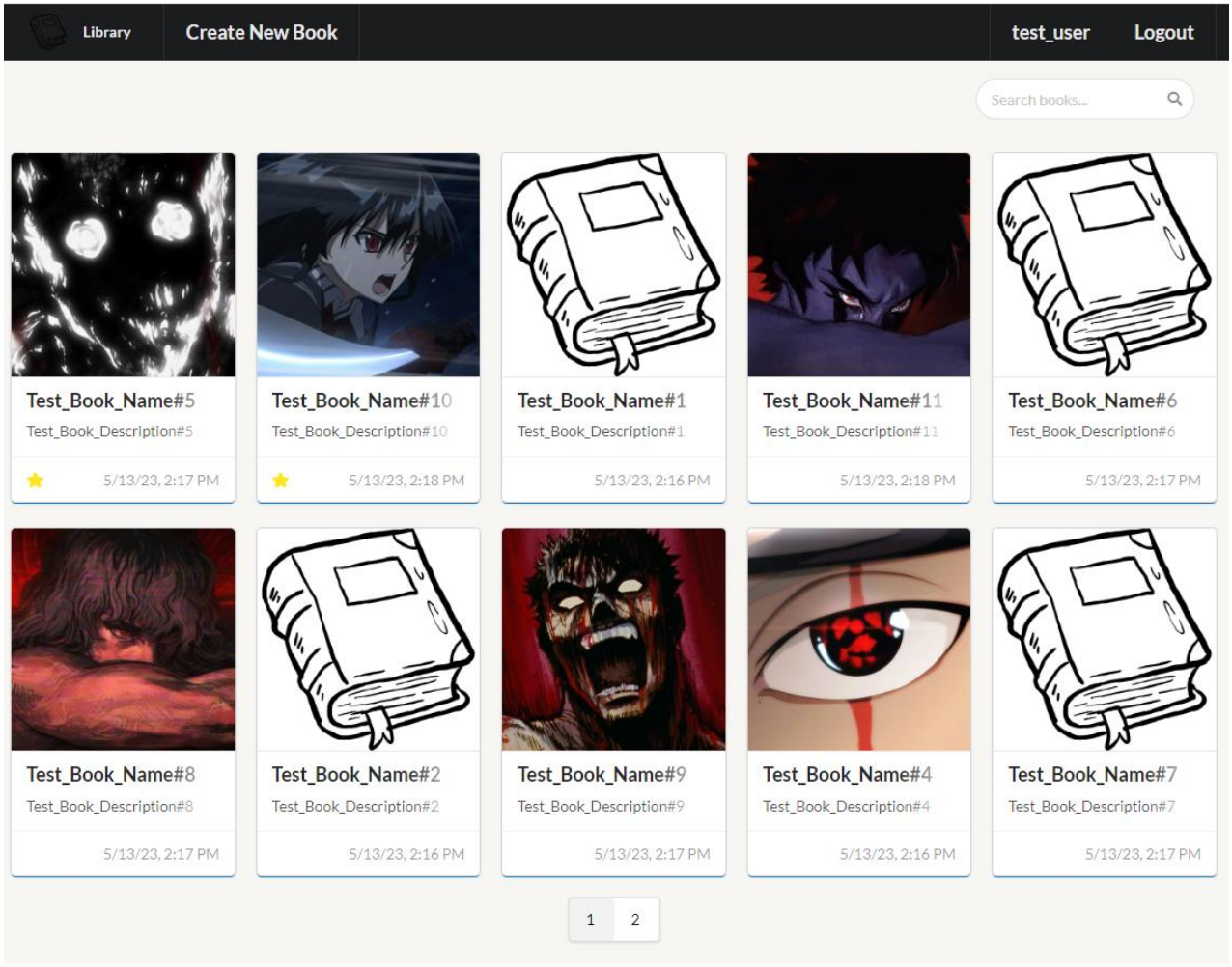

Рисунок 3.22 – Представлення списку книг

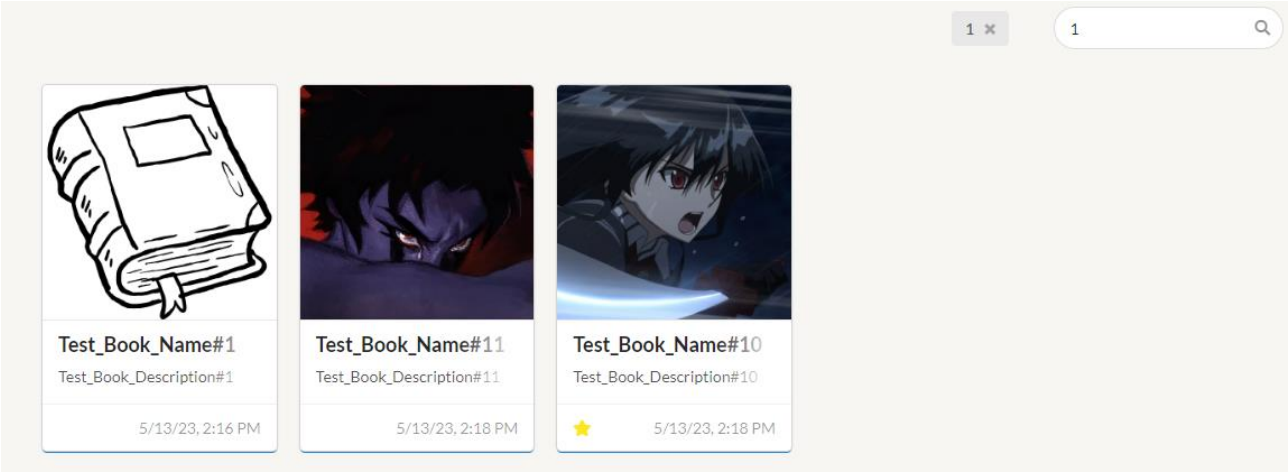

Рисунок 3.23 – Пошук книги по за її назвою

Book частиною якого є Book-Text:

Сторінка книги з детальною інформацією про неї. Також на ній є кнопка навігації на сторінку редагування книги та кнопка керування – кнопка видалення книги.

Редагувати або видаляти книгу може тільки її автор або адміністратор системи.

Також на сторінці книги є її текст для читання. Текст розбито на сторінки по яким можна переходити за допомогою панелі навігації. Вигляд сторінки книги показано на Рисунку 3.24.

Book-Create та Book-Edit:

На сторінках створення/редагування книги знаходяться відповідні форми. Валідація лежить на серверній частині. Якщо валідацію провалено то сервер поверне помилку з відповідним повідомленням інакше буде створено нову книгу/відредаговано існуючу. Форма створення нової книги показана на Рисунку 3.25.

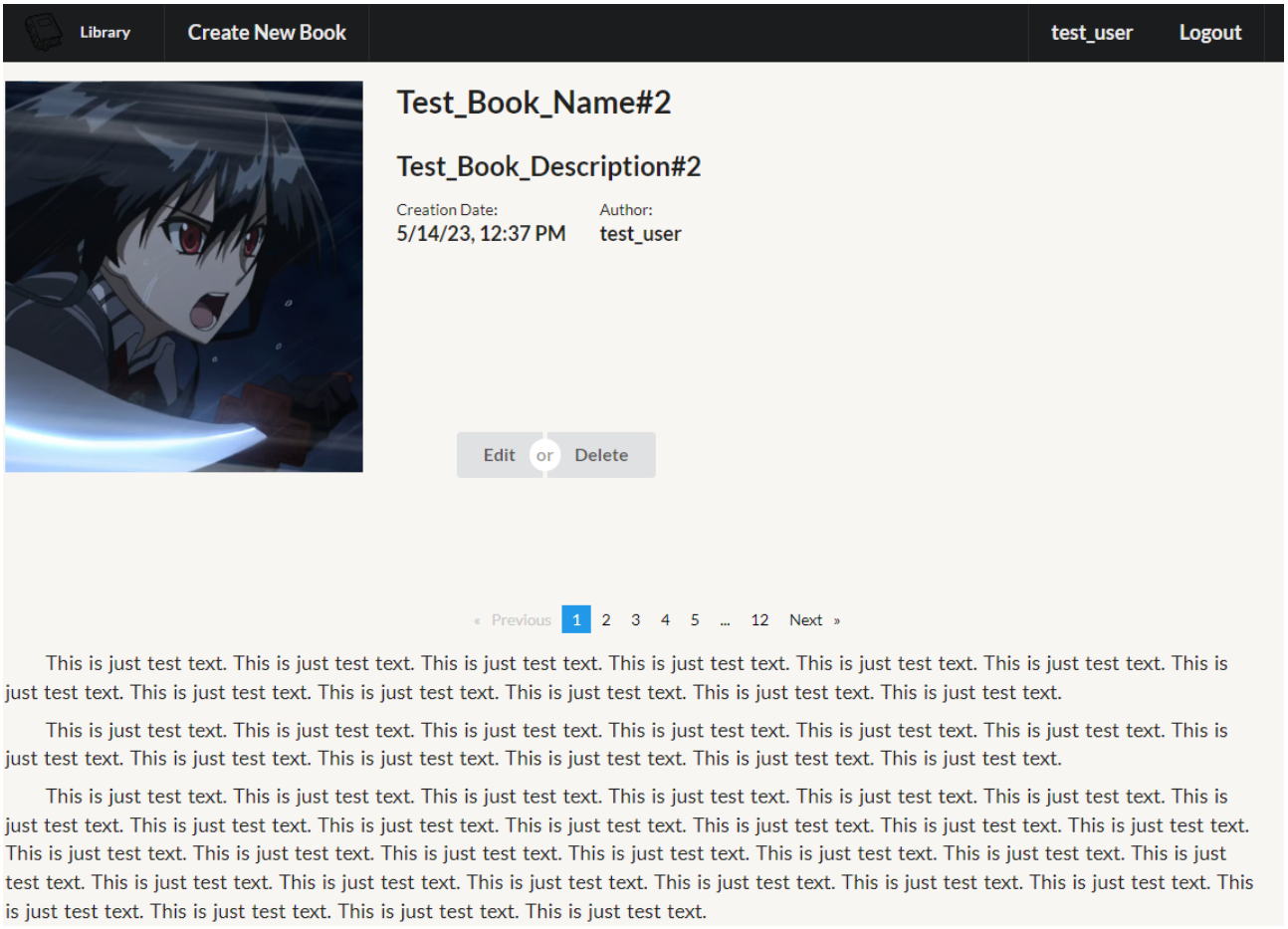

Рисунок 3.24 – Сторінка книги

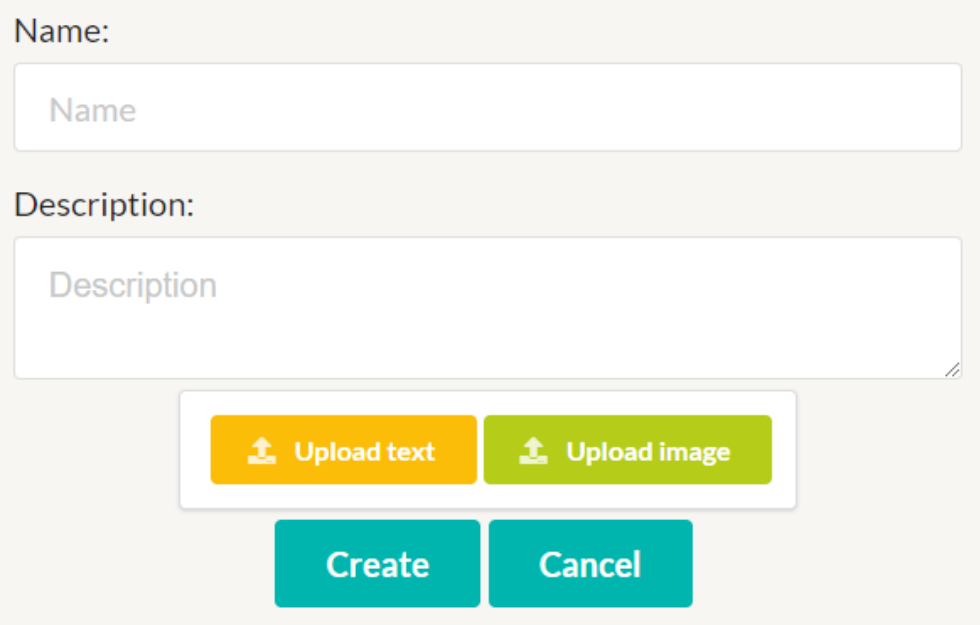

Рисунок 3.25 – Форма створення нової книги

Форма редагування книги показана на Рисунку 3.26.

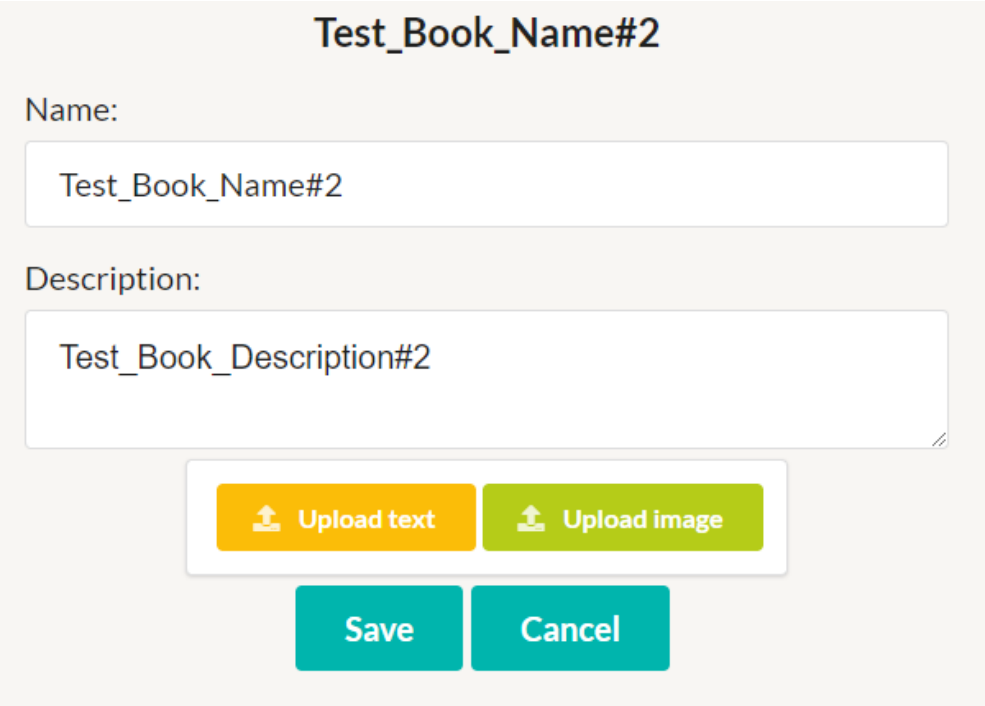

Рисунок 3.26 – Форма редагування книги

Для навігації по сторінкам використано Angular Routing.

Routing – це функціонал фреймворку Angular, який дозволяє переходити між різними сторінками або компонентами в рамках додатку. Він надає можливість визначати маршрути для різних URL-адрес та пов'язувати їх з певними компонентами. Файли з класами навігації:

[Library\angularapp\src\app\components\book\book-routing.module.ts]

[Library\angularapp\src\app\app-routing.module.ts]

Можливі переходи показано на Рисунках 3.27 – 3.28.

```
const routes: Routes = [
    {
        path: '',
        component: BookListComponent
    },
    {
        path: 'book/create',
        component: BookCreateComponent
    },
    {
        path: 'book/:id',
        component: BookComponent
    },
    {
        path: 'book/:id/edit',
        component: BookEditComponent
    }
];
```

Рисунок 3.27 – Навігація по модулю Book

```
const routes: Routes = [
    {
        path: '',
        loadChildren: () => import('./components/book/book.module').then(m => m.BookModule)
    },
    {
        path: 'auth/login',
        component: AuthComponent
    },
    {
        path: 'auth/register',
        component: AuthComponent
    },
    {
        path: '**',
        redirectTo: ''
    }
];
```

Рисунок 3.28 – Навігація по додатку

3.6. Реалізація системи безпеки.

Відповідає за систему безпеки клас AuthService, який знаходиться у файлі [Library\webapi\Library.DAL\Services\AuthService.cs], див. Додаток А.5.

Після надходження запиту на реєстрацію відбудеться валідація облікових даних користувача (довжина паролю більше 5 символів і тд.). Якщо

валідація успішна то пароль буде зашифровано (так як зберігати незашифрований пароль у базі даних небезпечно) та створено нового користувача з відповідними Login та PasswordHash.

Метод шифрування паролю зображено на Рисунку 3.29.

```
1 reference
private static void GeneratePasswordHash(string password, out byte[] passwordHash, out byte[] passwordSalt)
{
    using var hmac = new HMACSHA512();
    passwordHash = hmac.ComputeHash(Encoding.UTF8.GetBytes(password));
    passwordSalt = hmac.Key;
}
```

Рисунок 3.29 – Шифрування паролю

Для шифрування паролю використано стандарт HMACSHA512.

HMACSHA512 – це криптографічний алгоритм, який використовується для перевірки автентичності та цілісності повідомлення або даних. HMACSHA512 часто використовується для перевірки автентичності та цілісності повідомлень в різних протоколах безпеки та програмах.

HMACSHA512 поєднує функцію хешування SHA-512 з секретним ключем та повідомленням, щоб створити хеш-пов'язаний код аутентифікації повідомлення. Отриманий код можна використовувати для перевірки цілісності та автентичності повідомлення, порівнюючи його з розрахованим HMAC на боці отримувача.

Під час авторизації введені облікові дані (пароль) також буде закодовано та перевірено чи він збігається з хеш-кодом з бази даних.

Метод верифікації паролю показано на Рисунку 3.30.

```
1 reference
private static bool VerifyPasswordHash(string password, in byte[] passwordHash, in byte[] passwordSalt)
{
    using var hmac = new HMACSHA512(passwordSalt);
    byte[] computedHash = hmac.ComputeHash(Encoding.UTF8.GetBytes(password));
    return computedHash.SequenceEqual(passwordHash);
}
```

Рисунок 3.30 – Перевірка відповідності паролю

Після успішної авторизації буде згенеровано JSON Web Token та відправлено на клієнт. Сам токен складається з клеймів (Name, Identifier, Role).

Додання клеймів до токену продемонстровано на Рисунку 3.31.

```
List<Claim> claims = new()
{
    new Claim(ClaimTypes.Name, user.Login.ToString()
        ?? throw new ServiceException("Invalid user name.", HttpStatusCode.BadRequest)),
    new Claim(ClaimTypes.NameIdentifier, user.Id.ToString()
        ?? throw new ServiceException("Invalid user identifier.", HttpStatusCode.BadRequest))
};

foreach (string role in user.Roles.Select(role => role.Name))
{
    claims.Add(new Claim(ClaimTypes.Role, role));
}
```

Рисунок 3.31 – Додавання клеймів до токена

Ці клейми в подальшому будуть використовуватися на клієнті. Наприклад перевірки чи поточний користувач є автором книги для її видалення.

```
public register(user: Credentials) {
    this.authService.register(user).subscribe({
        next: (token: Token) => {
            localStorage.setItem('authToken', token.data!);
            this.return();
        },
        error: (error: any) => {
            this.toastr.error(error.error.message);
        },
        complete: () => {
            this.toastr.success("Login successful.");
        }
    });
}
```

Рисунок 3.32 – Метод реєстрації користувача

Метод реєстрації користувача, результатам якого є отримання токена авторизації з серверу показано на Рисунку 3.32.

Після того, як клієнт отримав токен з сервера його буде додано в Local Storage. [Library\angularapp\src\app\components\user\auth\auth.component.ts]

ВИСНОВКИ

У ході виконання кваліфікаційної роботи була розроблена веб-орієнтована інформаційна система "Library" для самвидаву книг. Для реалізації системи було використано сучасні технології, зокрема MS SQL Server, Entity Framework Core, ASP.NET Core, Angular, Semantic UI та JSON Web Tokens.

У результаті роботи над проектом було створено зручну та функціональну інформаційну систему, яка дозволяє користувачам самостійно видавати свої книги у веб-форматі. Система надає можливість для авторів завантажувати, редагувати та поширювати свої книги.

За допомогою MS SQL Server та Entity Framework Core була реалізована потужна база даних, яка забезпечує ефективне зберігання та обробку інформації про книги та користувачів. ASP.NET Core був використаний для створення веб-сервера, який забезпечує взаємодію з клієнтським додатком.

Фронтенд системи було розроблено з використанням Angular та Semantic UI, що забезпечує зручний та сучасний інтерфейс для користувачів. Безпека системи була покращена завдяки використанню JSON Web Tokens для автентифікації та авторизації користувачів.

Ця система має великий потенціал для подальшого розвитку. Використання сучасних технологій дозволило створити ефективну та зручну систему, яка в подальшому може бути розширена. Наприклад впровадженням:

- рейтингу книг.
- відгуків про книги.
- системи рекомендацій.
- внутрішній редактор тексту книги.
- монетизації авторів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. MS SQL Server: Microsoft. MS SQL Server Documentation. URL: https://learn.microsoft.com/sql/sql-server/ (дата звернення: 17.04.2023).

2. Entity Framework Core: Microsoft. Entity Framework Core Documentation. URL: https://docs.microsoft.com/ef/core (дата звернення: 25.04.2023).

3. ASP.NET Core: Microsoft. ASP.NET Core Documentation. URL: https://docs.microsoft.com/aspnet/core (дата звернення: 19.04.2023).

4. Angular: Angular Documentation. URL: https://angular.io/docs (дата звернення: 07.05.2023).

5. Semantic UI: Semantic UI Documentation. URL: https://semantic-ui.com (дата звернення: 12.05.2023).

6. JSON Web Tokens (JWT): JWT.IO. Introduction to JSON Web Tokens. URL: https://jwt.io/introduction (дата звернення: 02.05.2023).

7. Date C. J. SQL and Relational Theory: How to Write Accurate SQL Code. O'Reilly Media, Incorporated, 2015. 563 p.

8. Kline K., Obe R. O., Hsu L. S. SQL in a Nutshell: A Desktop Quick Reference. O'Reilly Media, Incorporated, 2022. 650 p.

9. Smith J. P. Entity Framework Core in Action, Second Edition. Manning Publications Co. LLC, 2021. 624 p.

10. Freeman A. Pro ASP.NET Core 6: Develop Cloud-Ready Web Applications Using MVC, Blazor, and Razor Pages. Apress L. P., 2022. 1086 p.

11. Lock A. ASP.NET Core in Action, Second Edition. Manning Publications Co. LLC, 2021. 832 p.

12. Marcano A. Programming ASP.NET Core 5 MVC and Web API: Examples in C#. Independently Published, 2021. 372 p.

13. Fain Y., Moiseev A. Angular Development with TypeScript. Manning Publications, 2018. 560 p.

14. Wilken J. Angular in Action. Manning Publications, 2018. 320 p.

15. Duckett J. HTML and CSS: Design and Build Websites. Wiley & Sons, Incorporated, John, 2011. 512 p.

16. Sarcar V. Design Patterns in C#: A Hands-On Guide with Real-World Examples. Apress, 2018. 488 p.

ДОДАТОК А

Бекенд

А.1 Моделі бази даних.

```
public class Book
{
        public Guid Id { get; set; }

        public string Name { get; set; }

        public string? Description { get; set; }

        public DateTime CreationDate { get; set; }

        public string? Image { get; set; }

        [JsonIgnore]
        public Guid UserId { get; set; }

        public User User { get; set; }

        [JsonIgnore]
        public BookText BookText { get; set; }
}


public class BookText
{
        public Guid Id { get; set; }

        public string? Text { get; set; }

        public Book Book { get; set; }
}


public class Role
{
        public Guid Id { get; set; }

        public string Name { get; set; }

        public List<User> Users { get; set; }
}


public class User
{
        public Guid Id { get; set; }

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

```
        [JsonIgnore]
        public byte[] PasswordHash { get; set; }

        [JsonIgnore]
        public byte[] PasswordSalt { get; set; }

        [JsonIgnore]
        public List<Role> Roles { get; set; }

        [JsonIgnore]
        public List<Book> Books { get; set; }
}


public class UserRole
{
        public Guid UserId { get; set; }

        public Guid RoleId { get; set; }
}
```

А.2 Контекст бази даних.

```
public class Context : DbContext
{
	public DbSet<Book> Books { get; set; }
	public DbSet<BookText> BookTexts { get; set; }
	public DbSet<User> Users { get; set; }
	public DbSet<Role> Roles { get; set; }
	public DbSet<UserRole> UserRoles { get; set; }

	public Context(DbContextOptions<Context> options) : base(options) { }

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

		modelBuilder.Entity<Book>()
			.HasOne(item => item.BookText)
			.WithOne(item => item.Book)
			.HasForeignKey<BookText>(item => item.Id);

		modelBuilder.Entity<Book>().ToTable("Books");
		modelBuilder.Entity<BookText>().ToTable("Books");

		modelBuilder.Entity<User>()
			.HasIndex(item => item.Login)
			.IsUnique();

		modelBuilder.Entity<User>()
			.HasMany(item => item.Roles)
			.WithMany(item => item.Users)
			.UsingEntity<UserRole>();

		modelBuilder.Entity<User>()
			.HasMany(item => item.Books)
			.WithOne(item => item.User)
			.HasForeignKey(item => item.UserId);

		SetData(modelBuilder);
	}

	private static void SetData(ModelBuilder modelBuilder)
	{
		Role[] roles = {
				new Role()
				{
					Id = Guid.Parse("e5e7f84a-685f-429b-ae38-9b44e067d76a"),
					Name = LibraryRoles.Author
				},
				new Role()
				{
					Id = Guid.Parse("2949d456-901a-4f7b-a7a3-7fd1e8c2f65d"),
					Name = LibraryRoles.Admin
				}
			};
```

```
                User admin = new()
                {
                        Id = Guid.Parse("630313d4-d313-4fa8-bb0e-bfce2dcb26b9"),
                        Login = "admin",
                        PasswordHash = Convert.FromBase64String(

        "PY5glS3zNRz1wqN8Fcw+tznBe9FEBfCVN38KxUAx2k9WDpHnxvOErceYolyjnWy5zM+659hK9wOnan5E
YIBvDA=="),
                        PasswordSalt = Convert.FromBase64String(

        "ghH/dvpMPe2CTS3d/lVfmudCAspmmEc0RJn40a6iPkez+CdYdAs3wwUVuL6l4ClUJbehuNrCDaHXhP8s
mLmpZfXGvpczTt" +

        "+j4BET+O2cWYfSquE6auNdT3chbSeJj7HAwaDftuN6HfCEcEaopEdxTu/7mWPfAfIbRE6OT09huDk=")
                };

                UserRole[] adminRole = {
                        new UserRole()
                        {
                                UserId = Guid.Parse("630313d4-d313-4fa8-bb0e-bfce2dcb26b9"),
                                RoleId = Guid.Parse("2949d456-901a-4f7b-a7a3-7fd1e8c2f65d")
                        }
                };

                modelBuilder.Entity<User>().HasData(admin);
                modelBuilder.Entity<Role>().HasData(roles);
                modelBuilder.Entity<UserRole>().HasData(adminRole);
        }
}
```

А.3 Точка збору та запуску програми.

```
public class Program
{
	public static void Main(string[] args)
	{
		var builder = WebApplication.CreateBuilder(args);

		builder.Services.AddControllers();

		builder.Services.AddEndpointsApiExplorer();
		builder.Services.AddSwaggerGen();

		builder.Services.AddCors(option => option.AddPolicy(name: "Library",
			policy =>
			{
				policy.WithOrigins("http://localhost:4200")
				.AllowAnyMethod()
				.AllowAnyHeader()
				.AllowCredentials();
			}));

		builder.Services.AddScoped<IUserService, UserService>();
		builder.Services.AddScoped<IAuthService, AuthService>();

		builder.Services.AddScoped<IBookService, BookService>();
		builder.Services.AddScoped<IImageService, ImageService>();
		builder.Services.AddScoped<ITextService, TextService>();

		builder.Services.AddScoped<IRepository<User, Guid>, Repository<User, Guid>>();
		builder.Services.AddScoped<IRepository<Role, Guid>, Repository<Role, Guid>>();

		builder.Services.AddScoped<IRepository<Book, Guid>, Repository<Book, Guid>>();
		builder.Services.AddScoped<IRepository<BookText, Guid>, Repository<BookText, Guid>>();

		var connectionString = builder.Configuration.GetConnectionString("DefaultConnection") ??
			throw new InvalidOperationException("Connection string 'DefaultConnection' not
found.");

		builder.Services.AddDbContext<Context>(options =>
			options.UseSqlServer(connectionString));
		builder.Services.AddDatabaseDeveloperPageExceptionFilter();

		builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
			.AddJwtBearer(options =>
		{
			options.TokenValidationParameters = new TokenValidationParameters()
			{
				RequireExpirationTime = false,
				ValidateIssuerSigningKey = true,
				ValidIssuer = builder.Configuration.GetSection("Jwt:Issuer").Value,
				ValidAudience = builder.Configuration.GetSection("Jwt:Audience").Value,
				IssuerSigningKey = new SymmetricSecurityKey(

	Encoding.UTF8.GetBytes(builder.Configuration.GetSection("Jwt:Key").Value!))
```

```
                    };
                });

                var app = builder.Build();

                using (var scope = app.Services.CreateScope())
                {
                    var serviceProvider = scope.ServiceProvider.GetRequiredService<Context>();
                    serviceProvider.Database.Migrate();
                }

                if (app.Environment.IsDevelopment())
                {
                    app.UseSwagger();
                    app.UseSwaggerUI();
                }

                app.UseStaticFiles();

                app.UseCors("Library");
                app.UseHttpsRedirection();

                app.UseRouting();

                app.UseMiddleware<ExceptionHandler>();

                app.UseAuthentication();
                app.UseAuthorization();

                app.MapControllers();

                app.Run();
            }
        }
```

А.4 Інтерфейс та клас репозиторію.

```
public interface IRepository<TEntity, TKey> : IDisposable where TEntity : class
{
	Task<PartialData<TEntity>> GetItemListAsync(Expression<Func<TEntity, bool>>? predicate,
		int? skip, int? take, params Expression<Func<TEntity, object?>>[] includes);
	Task<TEntity?> GetItemAsync(TKey id);
	Task<TEntity> GetItemAsync(Expression<Func<TEntity, bool>> predicate,
		params Expression<Func<TEntity, object?>>[] includes);
	Task CreateAsync(TEntity entity);
	void Update(TEntity entity);
	Task DeleteAsync(TKey id);
	Task SaveAsync();
}


public class Repository<TEntity, TKey> : IRepository<TEntity, TKey> where TEntity : class
{
	private readonly Context _context;
	private readonly DbSet<TEntity> _dbSet;

	public Repository(Context context)
	{
		_context = context;
		_dbSet = context.Set<TEntity>();
	}

	public async Task<PartialData<TEntity>> GetItemListAsync(Expression<Func<TEntity, bool>>?
predicate,
		int? skip, int? take, params Expression<Func<TEntity, object?>>[] includes)
	{
		IQueryable<TEntity> query = predicate == null ? _dbSet : _dbSet.Where(predicate);
		int totalAmount = await query.CountAsync();

		foreach (var include in includes)
		{
			query = query.Include(include);
		}

		return new PartialData<TEntity>()
		{
			Data = await query.Skip(skip ?? 0).Take(take ?? totalAmount).ToListAsync(),
			TotalAmount = totalAmount
		};
	}

	public async Task<TEntity?> GetItemAsync(TKey id)
	{
		return await _dbSet.FindAsync(id);
	}

	public async Task<TEntity> GetItemAsync(Expression<Func<TEntity, bool>> predicate, params
Expression<Func<TEntity, object?>>[] includes)
	{
		IQueryable<TEntity> query = _dbSet;
```

```
            foreach (var include in includes)
            {
                query = query.Include(include);
            }

            return await query.FirstAsync(predicate);
        }

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

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

        public async Task DeleteAsync(TKey id)
        {
            TEntity? entity = await _dbSet.FindAsync(id);

            if (entity != null)
            {
                _dbSet.Remove(entity);
            }
        }

        public async Task SaveAsync()
        {
            await _context.SaveChangesAsync();
        }

        #region dispose
        private bool disposed = false;

        public void Dispose()
        {
            Dispose(true);
            GC.SuppressFinalize(this);
        }

        private void Dispose(bool disposing)
        {
            if (!disposed)
            {
                if (disposing)
                {
                    _context.Dispose();
                }
            }

            disposed = true;
        }
        #endregion
```

```
}
```

А.5 Інтерфейс та клас сервісу авторизації

```
public interface IAuthService
{
	Task<Token> RegisterAsync(Credentials credentials);
	Task<Token> LoginAsync(Credentials credentials);
}



public class AuthService : IAuthService
{
	private readonly IConfiguration _configuration;
	private readonly IRepository<User, Guid> _userRepository;
	private readonly IRepository<Role, Guid> _roleRepository;

	public AuthService(IConfiguration configuration, IRepository<User, Guid> userRepository,
IRepository<Role, Guid> roleRepository)
	{
		_configuration = configuration;
		_userRepository = userRepository;
		_roleRepository = roleRepository;
	}

	public async Task<Token> RegisterAsync(Credentials credentials)
	{
		try
		{
			ValidateCredentials(credentials);
			GeneratePasswordHash(credentials.Password, out byte[] passwordHash, out byte[]
passwordSalt);
			Role role = await _roleRepository.GetItemAsync(role => role.Name ==
LibraryRoles.Author);

			User user = new()
			{
				Login = credentials.Login,
				PasswordHash = passwordHash,
				PasswordSalt = passwordSalt,
				Roles = new List<Role> { role }
			};

			await _userRepository.CreateAsync(user);
			await _userRepository.SaveAsync();

			return await LoginAsync(credentials);
		}
		catch (ServiceException)
		{
			throw;
		}
		catch
		{
			throw new ServiceException("Login is already in use.", HttpStatusCode.BadRequest);
		}
	}
```

```
        public async Task<Token> LoginAsync(Credentials credentials)
        {
                try
                {
                        User user = await _userRepository.GetItemAsync(user => user.Login ==
credentials.Login, user => user.Roles);

                        if (VerifyPasswordHash(credentials.Password, user.PasswordHash, user.PasswordSalt))
                        {
                                return CreateToken(user);
                        }

                        throw new Exception();
                }
                catch
                {
                        throw new ServiceException("Invalid login or password.",
HttpStatusCode.BadRequest);
                }
        }

        private static void ValidateCredentials(Credentials credentials)
        {
                if (credentials.Login.Length < 5)
                {
                        throw new ServiceException("Login must have at least 5 characters.",
HttpStatusCode.BadRequest);
                }

                if (credentials.Password.Length < 5)
                {
                        throw new ServiceException("Password must have at least 5 characters.",
HttpStatusCode.BadRequest);
                }
        }

        private static void GeneratePasswordHash(string password, out byte[] passwordHash, out byte[]
passwordSalt)
        {
                using var hmac = new HMACSHA512();
                passwordHash = hmac.ComputeHash(Encoding.UTF8.GetBytes(password));
                passwordSalt = hmac.Key;
        }

        private static bool VerifyPasswordHash(string password, in byte[] passwordHash, in byte[]
passwordSalt)
        {
                using var hmac = new HMACSHA512(passwordSalt);
                byte[] computedHash = hmac.ComputeHash(Encoding.UTF8.GetBytes(password));
                return computedHash.SequenceEqual(passwordHash);
        }

        private Token CreateToken(User user)
        {
                List<Claim> claims = new()
                        {
```

```
                                new Claim(ClaimTypes.Name, user.Login.ToString()
                                        ?? throw new ServiceException("Invalid user name.",
HttpStatusCode.BadRequest)),
                                new Claim(ClaimTypes.NameIdentifier, user.Id.ToString()
                                        ?? throw new ServiceException("Invalid user identifier.",
HttpStatusCode.BadRequest))
                                };

                        foreach (string role in user.Roles.Select(role => role.Name))
                        {
                                claims.Add(new Claim(ClaimTypes.Role, role));
                        }

                        SymmetricSecurityKey key =
new(Encoding.UTF8.GetBytes(_configuration.GetSection("Jwt:Key").Value!));

                        SigningCredentials credentials = new(key, SecurityAlgorithms.HmacSha512Signature);

                        JwtSecurityToken jwtToken = new(
                                _configuration.GetSection("Jwt:Issuer").Value,
                                _configuration.GetSection("Jwt:Audience").Value,
                                claims: claims,
                                signingCredentials: credentials);

                        return new Token()
                        {
                                Type = TokenType.Jwt,
                                Data = new JwtSecurityTokenHandler().WriteToken(jwtToken)
                        };
                }
}
```

А.6 Інтерфейс та клас сервісу обробки книг.

```
public interface IBookService
{
	Task<PartialData<Book>> GetBookListAsync(BookFilter bookFilter, params Expression<Func<Book, object?>>[] includes);
	Task<Book> GetAsync(Guid id);
	Task<Book> GetAsync(Guid id, params Expression<Func<Book, object?>>[] includes);
	Task<Book> CreateAsync(Book book);
	Task<Book> UpdateAsync(Book book);
	Task DeleteAsync(Guid id);
}



public class BookService : IBookService
{
	private readonly IRepository<Book, Guid> _bookRepository;

	public BookService(IServiceProvider services)
	{
		_bookRepository = services.GetRequiredService<IRepository<Book, Guid>>();
	}

	public async Task<PartialData<Book>> GetBookListAsync(BookFilter bookFilter, params Expression<Func<Book, object?>>[] includes)
	{
		Expression<Func<Book, bool>>? predicate = bookFilter.Name.IsNullOrEmpty() ? null :
			(book => book.Name.Contains(bookFilter.Name!));

		return await _bookRepository.GetItemListAsync(predicate,
			(bookFilter.CurrentPage - 1) * bookFilter.PageSize, bookFilter.PageSize, includes);
	}

	public async Task<Book> GetAsync(Guid id)
	{
		return await _bookRepository.GetItemAsync(id) ??
			throw new ServiceException("The appropriate book does not exist.", HttpStatusCode.BadRequest);
	}

	public async Task<Book> GetAsync(Guid id, params Expression<Func<Book, object?>>[] includes)
	{
		return await _bookRepository.GetItemAsync(item => item.Id == id, includes) ??
			throw new ServiceException("The appropriate book does not exist.", HttpStatusCode.BadRequest);
	}

	public async Task<Book> CreateAsync(Book book)
	{
		try
		{
			await _bookRepository.CreateAsync(book);
			await _bookRepository.SaveAsync();

			return book;
```

```
                }
                catch (Exception)
                {
                        throw new ServiceException("An unexpected error occurred while creating the book.",
HttpStatusCode.BadRequest);
                }
        }

        public async Task<Book> UpdateAsync(Book book)
        {
                try
                {
                        _bookRepository.Update(book);
                        await _bookRepository.SaveAsync();

                        return book;
                }
                catch (Exception)
                {
                        throw new ServiceException("An unexpected error occurred while updating the
book.", HttpStatusCode.BadRequest);
                }
        }

        public async Task DeleteAsync(Guid id)
        {
                try
                {
                        await _bookRepository.DeleteAsync(id);
                        await _bookRepository.SaveAsync();
                }
                catch (Exception)
                {
                        throw new ServiceException("An unexpected error occurred while deleting the book.",
HttpStatusCode.BadRequest);
                }
        }
}
```

А.7 Інтерфейс та клас сервісу обробки зображень.

```
public interface IImageService
{
	Task<string> ReplaceAsync(IFormFile formFile, string baseName);
	Task<string> UploadAsync(IFormFile formFile);
	Task DeleteAsync(string baseName);
}


public class ImageService : IImageService
{
	private readonly IConfiguration _configuration;

	public ImageService(IConfiguration configuration)
	{
		_configuration = configuration;
	}

	[SupportedOSPlatform("windows")]
	public async Task<string> ReplaceAsync(IFormFile formFile, string imageName)
	{
		ValidateImage(formFile.FileName);
		await DeleteAsync(imageName);
		return await SaveImageAsync(formFile);
	}

	[SupportedOSPlatform("windows")]
	public async Task<string> UploadAsync(IFormFile formFile)
	{
		ValidateImage(formFile.FileName);
		return await SaveImageAsync(formFile);
	}

	public async Task DeleteAsync(string fileName)
	{
		try
		{
			string filePath = Path.Combine(_configuration.GetSection("WebRootPath").Value!,
$"uploads\\images", fileName);

			if (File.Exists(filePath))
			{
				await Task.Run(() => File.Delete(filePath));
			}
		}
		catch (Exception)
		{
			throw new ServiceException("Deleting error: An unexpected error occurred while
deleting the image.");
		}
	}

	private static void ValidateImage(string imageName)
	{
```

```
                if (!FileExtension.ImageExtensions.Contains(Path.GetExtension(imageName)))
                {
                        throw new ServiceException(
                                $"Uploading error: Invalid file extension for image.\n"
                                + $"(Possible extensions: \"{string.Join("\" \"",
FileExtension.ImageExtensions)}\")",
                                HttpStatusCode.UnsupportedMediaType);
                }
        }

        [SupportedOSPlatform("windows")]
        private async Task<string> SaveImageAsync(IFormFile formFile)
        {
                try
                {
                        string fileName = Guid.NewGuid() + FileExtension.Default(FileType.Image);
                        string filePath = Path.Combine(_configuration.GetSection("WebRootPath").Value!,
$"uploads\\images", fileName);

                        using (Stream stream = formFile.OpenReadStream())
                        using (Image image = Image.FromStream(stream).GetThumbnailImage(500, 500, null,
IntPtr.Zero))
                        {
                                await Task.Run(() => image.Save(filePath));
                        }

                        return fileName;
                }
                catch (Exception)
                {
                        throw new ServiceException("Uploading error: An unexpected error occurred while
uploading the image.");
                }
        }
}
```

А.8 Інтерфейс та клас сервісу обробки тексту.

```
public interface ITextService
{
	Task<string> ReadAsync(IFormFile formFile);
	Task<IEnumerable<IEnumerable<string>>?> GetAsync(Guid id);
}



public class TextService : ITextService
{
	private readonly IRepository<BookText, Guid> _repository;

	public TextService(IServiceProvider services)
	{
		_repository = services.GetRequiredService<IRepository<BookText, Guid>>();
	}

	public async Task<string> ReadAsync(IFormFile formFile)
	{
		ValidateText(formFile.FileName);

		try
		{
			using StreamReader reader = new(formFile.OpenReadStream());
			return await reader.ReadToEndAsync();
		}
		catch (Exception)
		{
			throw new ServiceException("Reading error: An unexpected error occurred while
reading the file.");
		}
	}

	public async Task<IEnumerable<IEnumerable<string>>?> GetAsync(Guid id)
	{
		BookText bookText = await _repository.GetItemAsync(id) ??
			throw new ServiceException("The appropriate book does not exist.",
HttpStatusCode.BadRequest);

		return bookText.Text == null ? null : SplitText(bookText.Text);
	}

	private static void ValidateText(string textName)
	{
		if (!FileExtension.TextExtensions.Contains(Path.GetExtension(textName)))
		{
			throw new ServiceException(
				$"Uploading error: Invalid file extension for text.\n"
				+ $"(Possible extensions: \"{string.Join("\" \"",
FileExtension.TextExtensions)}\")",
				HttpStatusCode.UnsupportedMediaType);
		}
	}
```

```
        private static IEnumerable<IEnumerable<string>> SplitText(in string text)
        {
                IEnumerable<string> paragraphs = text.Split('\n')
                        .Select(s => s.Trim())
                        .Where(s => !string.IsNullOrEmpty(s));

                List<List<string>> pages = new();
                List<string> page = new();
                int pageSize = (int)PageSize.Medium;
                int currentSize = 0;

                foreach (var paragraph in paragraphs)
                {
                        if (currentSize > pageSize)
                        {
                                pages.Add(page);
                                page = new List<string>();
                                currentSize = 0;
                        }

                        page.Add(paragraph);
                        currentSize += paragraph.Length;
                }

                if (page.Any())
                {
                        pages.Add(page);
                }

                return pages;
        }
}
```

А.9 Інтерфейс та клас сервісу обробки даних про користувачів.

```
public interface IUserService
{
	Task<Guid> GetIdAsync(ClaimsPrincipal principal);

	Task<User> GetAsync(Guid id);

	Task<User> GetAsync(ClaimsPrincipal principal);

	Task<User> GetAsync(ClaimsPrincipal principal, Expression<Func<User, object?>> include);

	Task ValidateUserAsync(ClaimsPrincipal principal, Guid userId);
}


public class UserService : IUserService
{
	private readonly IRepository<User, Guid> _userRepository;

	public UserService(IServiceProvider services)
	{
		_userRepository = services.GetRequiredService<IRepository<User, Guid>>();
	}

	public async Task<Guid> GetIdAsync(ClaimsPrincipal principal)
	{
		Guid id = ParseId(principal);

		return await _userRepository.GetItemAsync(id) != null ? id
			: throw new ServiceException("The appropriate user does not exist.",
HttpStatusCode.BadRequest);
	}

	public async Task<User> GetAsync(Guid id)
	{
		return await _userRepository.GetItemAsync(id)
			?? throw new ServiceException("The appropriate user does not exist.",
HttpStatusCode.BadRequest);
	}

	public async Task<User> GetAsync(ClaimsPrincipal principal)
	{
		Guid id = ParseId(principal);

		return await _userRepository.GetItemAsync(id)
			?? throw new ServiceException("The appropriate user does not exist.",
HttpStatusCode.BadRequest);
	}

	public async Task<User> GetAsync(ClaimsPrincipal principal, Expression<Func<User, object?>>
include)
	{
		Guid id = ParseId(principal);
```

```
                return await _userRepository.GetItemAsync(user => user.Id == id, include)
                        ?? throw new ServiceException("The appropriate user does not exist.",
HttpStatusCode.BadRequest); ;
        }

        public async Task ValidateUserAsync(ClaimsPrincipal principal, Guid userId)
        {
                User user = await GetAsync(principal, user => user.Roles);

                if (user.Roles.Select(role => role.Name).Contains(LibraryRoles.Admin))
                {
                        return;
                }

                if (userId != user.Id)
                {
                        throw new ServiceException("Invalid user credentials. Only the book author can edit
it.",
                                HttpStatusCode.BadRequest);
                }
        }

        private static Guid ParseId(ClaimsPrincipal principal)
        {
                return Guid.Parse(principal.Claims.FirstOrDefault(claim => claim.Type ==
ClaimTypes.NameIdentifier)?.Value
                        ?? throw new ServiceException("Invalid user token.", HttpStatusCode.BadRequest));
        }
}
```

А.10 Клас контроллеру авторизації.

```
[Route("api/[controller]")]
[ApiController]
public class AuthController : ControllerBase
{
	private readonly IAuthService _authService;

	public AuthController(IAuthService authService)
	{
		_authService = authService;
	}

	[HttpPost("register")]
	public async Task<Token> Register([FromBody] Credentials credentials)
	{
		return await _authService.RegisterAsync(credentials);
	}

	[HttpPost("login")]
	public async Task<Token> Login([FromBody] Credentials credentials)
	{
		return await _authService.LoginAsync(credentials);
	}
}
```

А.11 Клас контроллеру для обробки запитів про книги.

```
[Route("api/[controller]")]
[ApiController]
public class BookController : ControllerBase
{
	private readonly IBookService _bookService;
	private readonly IImageService _imageService;
	private readonly ITextService _textService;
	private readonly IUserService _userService;

	public BookController(IBookService bookService, IImageService imageService,
		ITextService textService, IUserService userService)
	{
		_bookService = bookService;
		_imageService = imageService;
		_textService = textService;
		_userService = userService;
	}

	[HttpGet]
	public async Task<PartialData<Book>> List([FromQuery] BookFilter bookFilter)
	{
		return await _bookService.GetBookListAsync(bookFilter, book => book.User);
	}

	[HttpGet("{id}")]
	public async Task<Book> Get(Guid id)
	{
		return await _bookService.GetAsync(id, book => book.User);
	}

	[HttpPost, Authorize]
	public async Task<Book> Create([FromForm] WebBook webBook)
	{
		Book book = new()
		{
			Id = Guid.NewGuid(),
			Name = webBook.Name,
			Description = webBook.Description,
			CreationDate = DateTime.Now.AddDays(-22),
			UserId = await _userService.GetIdAsync(User)
		};

		if (webBook.Image != null)
		{
			book.Image = await _imageService.UploadAsync(webBook.Image);
		}

		if (webBook.BookText != null)
		{
			book.BookText = new BookText()
			{
				Text = await _textService.ReadAsync(webBook.BookText)
			};
```

```
            }

            return await _bookService.CreateAsync(book);
        }

        [HttpPut, Authorize]
        public async Task<Book> Update([FromForm] WebBook webBook)
        {
            Book book = await _bookService.GetAsync(webBook.Id);
            await _userService.ValidateUserAsync(User, book.UserId);

            book.Name = webBook.Name;
            book.Description = webBook.Description;

            if (webBook.Image != null)
            {
                book.Image = book.Image != null
                    ? await _imageService.ReplaceAsync(webBook.Image, book.Image)
                    : await _imageService.UploadAsync(webBook.Image);
            }

            if (webBook.BookText != null)
            {
                book.BookText = new BookText()
                {
                    Text = await _textService.ReadAsync(webBook.BookText)
                };
            }

            return await _bookService.UpdateAsync(book);
        }

        [HttpDelete("{id}"), Authorize]
        public async Task Delete(Guid id)
        {
            Book book = await _bookService.GetAsync(id);
            await _userService.ValidateUserAsync(User, book.UserId);

            if (book.Image != null)
            {
                await _imageService.DeleteAsync(book.Image);
            }

            await _bookService.DeleteAsync(id);
        }
    }
```

А.12 Клас контроллеру для обробки запитів про текст книги.

```
[Route("api/[controller]")]
[ApiController]
public class TextController : ControllerBase
{
	private readonly ITextService _textService;

	public TextController(ITextService textService)
	{
		_textService = textService;
	}

	[HttpGet("{id}")]
	public async Task<WebText> Get(Guid id)
	{
		return new WebText()
		{
			Id = id,
			Pages = await _textService.GetAsync(id)
		};
	}
}
```

А.13 Форми які надходять в контролери.

```
public class WebBook
{
        public Guid Id { get; set; }

        public string Name { get; set; }

        public string? Description { get; set; }

        public DateTime CreationDate { get; set; }

        public IFormFile? BookText { get; set; }

        public IFormFile? Image { get; set; }
}


public class WebText
{
        public Guid Id { get; set; }

        public IEnumerable<IEnumerable<string>>? Pages { get; set; }
}
```

А.14 Обробка виключень.

```
public class ExceptionHandler
{
	private readonly RequestDelegate _requestDelegate;

	public ExceptionHandler(RequestDelegate requestDelegate)
	{
		_requestDelegate = requestDelegate;
	}

	public async Task InvokeAsync(HttpContext httpContext)
	{
		try
		{
			await _requestDelegate(httpContext);
		}
		catch (ServiceException exception)
		{
			await HandleServiceException(httpContext, exception);
		}
		catch (Exception exception)
		{
			await HandleException(httpContext, exception);
		}
	}

	private static async Task HandleServiceException(HttpContext context, ServiceException
serviceException)
	{
		await HandleException(context, serviceException, (int)serviceException.HttpStatusCode);
	}

	private static async Task HandleException(HttpContext context, Exception exception, int
httpStatusCode = 500)
	{
		context.Response.StatusCode = httpStatusCode;
		context.Response.ContentType = "application/json";

		string json = JsonSerializer.Serialize(new
		{
			status = httpStatusCode,
			message = exception.Message
		});

		await context.Response.WriteAsync(json);
	}
}
```

А.15 Допоміжні класи, константи, перечислення.

```
public static class FileExtension
{
	#region image
	public static readonly string BMP = ".bmp";
	public static readonly string JPG = ".jpg";
	public static readonly string JPE = ".jpe";
	public static readonly string PNG = ".png";

	public static IEnumerable<string> ImageExtensions
	{
		get
		{
			yield return BMP;
			yield return JPG;
			yield return JPE;
			yield return PNG;
		}
	}
	#endregion

	#region text
	public static readonly string TXT = ".txt";

	public static IEnumerable<string> TextExtensions
	{
		get
		{
			yield return TXT;
		}
	}
	#endregion

	public static string Default(FileType fileType)
	{
		return fileType switch
		{
			FileType.Image => BMP,
			FileType.Text => TXT,
			_ => string.Empty
		};
	}
}


public enum FileType : ushort
{
	Image = 0,
	Text = 1
}


public static class LibraryRoles
```

```
{
        public static readonly string Admin = "admin";
        public static readonly string Author = "author";
}


public enum PageSize : ushort
{
        Small = 5000,
        Medium = 10000,
        Large = 15000
}


public enum TokenType : byte
{
        Jwt = 0,
}


public class BookFilter
{
        public string? Name { get; set; }

        public int? CurrentPage { get; set; }

        public int? PageSize { get; set; }
}


public class Credentials
{
        public required string Login { get; set; }
        public required string Password { get; set; }
}


public class PartialData<TEntity> where TEntity : class
{
        public IEnumerable<TEntity> Data { get; set; }
        public int TotalAmount { get; set; }
}


public class Token
{
        public TokenType Type { get; set; }
        public string Data { get; set; }
}


public class ServiceException : Exception
```

```
{
        public HttpStatusCode HttpStatusCode { get; set; }

        public ServiceException(string message, HttpStatusCode httpStatusCode =
HttpStatusCode.InternalServerError)
                : base(message)
        {
                HttpStatusCode = httpStatusCode;
        }
}
```

ДОДАТОК Б

Фронтенд

Б.1 Клас сервісу авторизації.

```
@Injectable({
  providedIn: 'root'
})
export class AuthService {
    private url = "Auth";
    constructor(private httpClient: HttpClient) { }

    public register(user: Credentials): Observable<Token> {
        return this.httpClient.post<Token>(`${environment.apiUrl}/${this.url}/register`, user);
    }

    public login(user: Credentials): Observable<Token> {
        return this.httpClient.post<Token>(`${environment.apiUrl}/${this.url}/login`, user);
    }
}
```

Б.2 Клас сервісу передачі запитів про книги.

```
@Injectable({
  providedIn: 'root'
})
export class BookService {
    private url = "Book";

    constructor(private httpClient: HttpClient) { }

    public list(params: HttpParams): Observable<PartialData<Book>> {
        return this.httpClient.get<PartialData<Book>>(`${environment.apiUrl}/${this.url}`, { params });
    }

    public get(id: string): Observable<Book> {
        return this.httpClient.get<Book>(`${environment.apiUrl}/${this.url}/${id}`);
    }

    public create(formData: FormData): Observable<Book> {
        return this.httpClient.post<Book>(`${environment.apiUrl}/${this.url}`, formData);
    }

    public update(formData: FormData): Observable<Book> {
        return this.httpClient.put<Book>(`${environment.apiUrl}/${this.url}`, formData);
    }

    public delete(id: string): Observable<Book> {
        return this.httpClient.delete<Book>(`${environment.apiUrl}/${this.url}/${id}`);
    }
}
```

Б.3 Клас сервісу передачі запитів про текст книги.

```
@Injectable({
  providedIn: 'root'
})
export class TextService {
    private url = "Text";

    constructor(private httpClient: HttpClient) { }

    public get(id: string): Observable<Text> {
        return this.httpClient.get<Text>(`${environment.apiUrl}/${this.url}/${id}`);
    }
}
```

Б.4 Перехоплювач запитів.

```
@Injectable()
export class AuthInterceptor implements HttpInterceptor {
   intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
      const token = localStorage.getItem('authToken');

      if (token) {
         req = req.clone({
            setHeaders: { Authorization: `Bearer ${token}` }
         });
      }

      return next.handle(req);
   }
}
```

Б.5 Головний компонент.

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.scss']
})
export class AppComponent {
  constructor(private router: Router, private authService: AuthService) { }

  get userName(): string | undefined {
    const token = localStorage.getItem('authToken');

    if (token) {
      const decodeToken: any = jwtDecode(token);
      return decodeToken[Jwt.name];
    }

    return undefined;
  }

  get isAuthorized(): boolean {
    const token = localStorage.getItem('authToken');

    if (token) {
      return !!jwtDecode(token);
    }

    return false;
  }

  public logout() {
    localStorage.removeItem('authToken');
    this.router.navigate(['']);
  }
}
```

```
<header class="ui fixed inverted menu">
  <div class="ui container">
    <a href="" class="header item">
      <img class="logo" src="../favicon.ico">
      <div style="margin-inline: 15px;">Library</div>
    </a>
    <a class="ui header item" *ngIf="isAuthorized" [routerLink]="['book/create']">Create New Book</a>
    <a class="ui header item right" *ngIf="!isAuthorized" [routerLink]="['auth/login']">Login</a>
    <a class="ui header item" *ngIf="!isAuthorized" [routerLink]="['auth/register']">Register</a>
    <div class="ui header item right" *ngIf="isAuthorized">{{userName}}</div>
    <a class="ui header item" *ngIf="isAuthorized" (click)="logout()">Logout</a>
  </div>
</header>

<main class="ui container main">
  <router-outlet></router-outlet>
</main>
```

```
<footer class="ui inverted vertical segment">
    <div class="ui inverted container divider"></div>
</footer>


.main {
    margin-top: 70px;
    margin-bottom: 30px;
    min-height: 100%;
}
```

Б.6 Компонент авторизації.

```
@Component({
  selector: 'app-auth',
  templateUrl: './auth.component.html',
  styleUrls: ['./auth.component.scss']
})
export class AuthComponent implements OnInit {
  @Input() user = new Credentials();
  isLogin = false;

  constructor(private route: ActivatedRoute, private router: Router, private toastr: ToastrService, private
authService: AuthService) { }

  ngOnInit(): void {
    this.isLogin = this.route.snapshot.routeConfig?.path?.endsWith('login') ?? false;
  }

  public register(user: Credentials) {
    this.authService.register(user).subscribe({
      next: (token: Token) => {
        localStorage.setItem('authToken', token.data!);
        this.return();
      },
      error: (error: any) => {
        this.toastr.error(error.error.message);
      },
      complete: () => {
        this.toastr.success("Login successful.");
      }
    });
  }

  public login(user: Credentials) {
    this.authService.login(user).subscribe({
      next: (token: Token) => {
        localStorage.setItem('authToken', token.data!);
        this.return();
      },
      error: (error: any) => {
        this.toastr.error(error.error.message);
      },
      complete: () => {
        this.toastr.success("Login successful.");
      }
    });
  }

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

```
<div class="ui middle aligned center aligned grid" *ngIf="user">
```

```
    <div class="column" style="max-width: 500px; margin-top: 10%;">
      <h2 class="ui teal header">
        <div class="content" *ngIf="isLogin">
          Log-in to your account
        </div>
        <div class="content" *ngIf="!isLogin">
          Create a new account
        </div>
      </h2>
      <form class="ui large form">
        <div class="ui segment">
          <div class="field">
            <div class="ui left icon input">
              <i class="user icon"></i>
              <input [(ngModel)]="user.login" type="text" name="login" placeholder="Login">
            </div>
          </div>
          <div class="field">
            <div class="ui left icon input">
              <i class="lock icon"></i>
              <input [(ngModel)]="user.password" type="password" name="password"
placeholder="Password">
            </div>
          </div>
          <div class="ui fluid large teal submit button" (click)="login(user)" *ngIf="isLogin">Login</div>
          <div class="ui fluid large teal submit button" (click)="register(user)" *ngIf="!isLogin">Register</div>
        </div>
      </form>
    </div>
</div>
```

Б.7 Компонент картки книги.

```
@Component({
  selector: 'app-book-card',
  templateUrl: './book-card.component.html',
  styleUrls: ['./book-card.component.scss']
})
export class BookCardComponent {
  @Input() book?: Book;
  public environment = environment;
  constructor() { }

  isAuthor(authorId?: string): boolean {
    if (authorId) {
      const token = localStorage.getItem('authToken');

      if (token) {
        const decodeToken: any = jwtDecode(token);

        if (decodeToken[Jwt.nameIdentifier] === authorId) {
          return true;
        }
      }
    }

    return false;
  }
}
```

```
<div class="image">
  <img class="image" [src]="environment.imagesUrl + (book?.image || 'default_book_image.bmp')">
</div>
<div class="content">
  <div class="header hide">{{ book?.name }}</div>
  <div class="description hide">{{ book?.description }}</div>
</div>
<div class="extra content">
  <span class="floated" *ngIf="isAuthor(book?.user?.id)">
    <i class="star icon active"></i>
  </span>
  <span class="right floated">
    {{ book?.creationDate | date: 'short' }}
  </span>
</div>
```

```
.hide {
  white-space: nowrap;
  position: relative;
  overflow: hidden;
}

.hide::before {
  content: "";
```

```
    position: absolute;
    inset: 0;
    background: linear-gradient(to right, transparent 70%, white);
}
```

Б.8 Компонент списку книг.

```
@Component({
  selector: 'app-book-list',
  templateUrl: './book-list.component.html',
  styleUrls: ['./book-list.component.scss']
})
export class BookListComponent implements OnInit {
  books?: Book[];
  name?: string;
  totalAmount = 0;
  currentPage = 1;
  pageSize = 10;

  get pageCount(): number {
    return Math.ceil(this.totalAmount / this.pageSize);
  }

  constructor(private bookService: BookService) { }

  ngOnInit(): void {
    const params = new HttpParams()
      .set("name", this.name ?? "")
      .set("currentPage", this.currentPage)
      .set("pageSize", this.pageSize);

    this.bookService.list(params).subscribe((result: PartialData<Book>) => {
      this.books = result.data;
      this.totalAmount = result.totalAmount ?? 0;
    });
  }

  clear() {
    this.name = "";
    this.ngOnInit();
  }

  search() {
    this.currentPage = 1;
    this.ngOnInit();
  }

  getPage(pageNumber: number) {
    this.currentPage = pageNumber;
    this.ngOnInit();
  }

  paginate(): number[] {
    let pageCount = this.pageCount;
    let pages = new Set<number>();
    let counter = 1;

    pages.add(1);
    pages.add(this.currentPage);
    pages.add(pageCount);
```

```
        while (pages.size < 5 && pages.size < pageCount) {
            let targetPage = this.currentPage + counter;

            if (targetPage >= 1 && targetPage <= pageCount) {
                pages.add(targetPage);
            }

            targetPage = this.currentPage - counter;

            if (targetPage >= 1 && targetPage <= pageCount) {
                pages.add(targetPage);
            }

            counter++;
        }

        pages.delete(1);

        let pagesArray = Array.from(pages).sort((a, b) => a - b);
        let result = [1];

        pagesArray.forEach(element => {
            if (element - 1 > result[result.length - 1]) {
                result.push(0);
            }

            result.push(element);
        });

        return result;
    }
}
```

```
<div>
    <div class="ui search right">
        <span class="split-inline" *ngIf="this.name">
            <a class="ui large label">
                {{this.name}}
                <i class="delete icon" (click)="clear()"></i>
            </a>
        </span>

        <span class="ui icon input split-inline">
            <input [(ngModel)]="name" class="prompt" type="text" placeholder="Search books..." (blur)="search()"
                (keyup.enter)="search()">
            <i class="search icon"></i>
        </span>
    </div>

    <div class="ui five link cards split-block">
        <app-book-card class="blue card" *ngFor="let book of books" [book]="book"
            [routerLink]="['book', book?.id]"></app-book-card>
    </div>

    <div class="split-block mid" *ngIf="pageCount > 1">
```

```
      <div class="ui pagination menu">
        <div *ngFor="let pageNumber of paginate()">
          <a class="active item" *ngIf="pageNumber == this.currentPage" (click)="getPage(pageNumber)">
            {{pageNumber}}
          </a>
          <a class="disabled item" *ngIf="pageNumber === 0">
            ...
          </a>
          <a class="item" *ngIf="pageNumber != 0 && pageNumber != this.currentPage"
(click)="getPage(pageNumber)">
            {{pageNumber}}
          </a>
        </div>
      </div>
    </div>
</div>
```

```
.split-block {
    margin-top: 20px;
}

.split-inline {
    margin-inline: 20px;
}

.mid {
    display: flex;
    justify-content: center;
    align-items: center;
}

.right {
    text-align: right;
}
```

Б.9 Компонент книги.

```
@Component({
  selector: 'app-book',
  templateUrl: './book.component.html',
  styleUrls: ['./book.component.scss']
})
export class BookComponent implements OnInit {
  book?: Book;
  public environment = environment;

  constructor(private route: ActivatedRoute, private router: Router, private toastr: ToastrService, private
bookService: BookService) { }

  ngOnInit(): void {
    const id = this.route.snapshot.paramMap.get('id');
    if (id) {
      this.bookService.get(id).subscribe({
        next: (response) => {
          this.book = response;
        },
        error: (error: any) => {
          this.toastr.error(error.error.message);
        }
      });
    }

    window.scrollTo({ top: 0 });
  }

  deleteBook(id: string) {
    this.bookService.delete(id).subscribe({
      next: () => {
        this.return();
      },
      error: (error: any) => {
        this.toastr.error(error.error.message);
      },
      complete: () => {
        this.toastr.success("Deleting successful.");
      }
    });
  }

  isAuthorOrAdmin(authorId?: string): boolean {
    if (authorId) {
      const token = localStorage.getItem('authToken');

      if (token) {
        const decodeToken: any = jwtDecode(token);

        if (decodeToken[Jwt.role].includes(Roles.admin) || decodeToken[Jwt.nameIdentifier] === authorId) {
          return true;
        }
      }
```

```
      }

      return false;
    }

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

```
<div class="container" *ngIf="book">
  <div>
    <img class="image info" [src]="environment.imagesUrl + (book.image || 'default_book_image.bmp')">
  </div>
  <div class="info">
    <div class="container-top">
      <h1>{{ book.name }}</h1>
      <h2>{{ book.description }}</h2>
      <div class="container">
        <div class="info">
          Creation Date:
          <h3>{{ book.creationDate | date: 'short' }}</h3>
        </div>
        <div class="info" *ngIf="book?.user">
          Author:
          <h3>{{ book.user?.login }}</h3>
        </div>
      </div>
    </div>
    <div class="container-bot">
      <div class="ui large buttons" *ngIf="book.id && isAuthorOrAdmin(book.user?.id)">
        <button class="ui button" [routerLink]="['edit']">Edit</button>
        <div class="or"></div>
        <button class="ui button" (click)="deleteBook(book.id)">Delete</button>
      </div>
    </div>
  </div>
</div>

<app-book-text *ngIf="book && book.id" [id]="book.id"></app-book-text>
```

```
.image {
  height: 350px;
  width: 350px;
}

.container {
  display: flex;
}

.info {
  padding-right: 30px;
}
```

```
.container-top {
    height: 50%;
}

.container-bot {
    display: flex;
    flex-direction: column;
    justify-content: flex-end;
    align-items: center;
    height: 50%;
}
```

Б.10 Компонент тексту книги.

```
@Component({
  selector: 'app-book-text',
  templateUrl: './book-text.component.html',
  styleUrls: ['./book-text.component.scss']
})
export class BookTextComponent implements OnInit {
  @Input() id?: string;
  text?: Text;
  currentPage = 1;
  itemsPerPage = 1;

  constructor(private textService: TextService) { }

  ngOnInit(): void {
    if (this.id) {
      this.textService.get(this.id).subscribe((result: Text) => (this.text = result));
    }
  }

  handlePageChange(pageNumber: number) {
    this.currentPage = pageNumber;
    window.scrollTo({ top: 480, behavior: 'smooth' });
  }
}
```

```
<div *ngIf="text?.pages" style="margin-top: 100px;">
    <pagination-controls style="display: flex; justify-content: center;"
        (pageChange)="handlePageChange($event)"></pagination-controls>

    <div *ngFor="let page of text?.pages || [] | paginate: { itemsPerPage: itemsPerPage, currentPage: currentPage
}">
        <div *ngFor="let paragraph of page">
            <p class="parag">{{paragraph}}</p>
        </div>
    </div>

    <pagination-controls style="display: flex; justify-content: center;"
        (pageChange)="handlePageChange($event)"></pagination-controls>
</div>
```

```
.parag {
    font-size: 18px;
    word-spacing: 2px;
    text-indent: 2em;
    margin-bottom: 8px;
}
```

Б.11 Компонент для створення книги.

```
@Component({
  selector: 'app-book-create',
  templateUrl: './book-create.component.html',
  styleUrls: ['./book-create.component.scss']
})
export class BookCreateComponent implements OnInit {
  @Input() book?: Book;
  imageFile: any;
  textFile: any;

  constructor(private router: Router, private toastr: ToastrService, private bookService: BookService) {
  }

  ngOnInit(): void {
    this.book = new Book();
    window.scrollTo({ top: 0 });
  }

  createBook(book: Book) {
    const formData: FormData = new FormData();

    formData.append("name", book.name ?? "");
    formData.append("description", book.description ?? "");

    if (this.imageFile) {
      formData.append("image", this.imageFile);
    }

    if (this.textFile) {
      formData.append("bookText", this.textFile);
    }

    this.bookService.create(formData).subscribe({
      next: () => {
        this.return();
      },
      error: (error: any) => {
        if (error.error.errors) {
          this.toastr.error(error.error.errors.Name[0]);
        }
        else {
          this.toastr.error(error.error.message);
        }
      },
      complete: () => {
        this.toastr.success("Updating successful.");
      }
    });
  }

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

```
  selectImageFile(event: any) {
    this.imageFile = event?.target?.files[0];
  }

  selectTextFile(event: any) {
    this.textFile = event?.target?.files[0];
  }
}
```

```
<div class="mid">
  <div *ngIf="book" class="ui form huge" style="width: 50%; ">
    <h2 class="mid">{{ book.name }}</h2>
    <div class="split">
      <div class="small-split">Name:</div>
      <input [(ngModel)]="book.name" placeholder="Name" />
    </div>
    <div class="split">
      <div class="small-split">Description:</div>
      <textarea rows="2" [(ngModel)]="book.description" placeholder="Description"></textarea>
    </div>
    <div class="ui middle grid container mid">
      <div class="ui fluid segment">
        <input accept=".png, .jpg, .jpe, .bmp" type="file" (change)="selectImageFile($event)" class="inputfile"
          id="imageFile" />
        <label for="imageFile" class="ui large olive right floated button">
          <i class="ui upload icon"></i>
          Upload image
        </label>

        <input accept=".txt" type="file" (change)="selectTextFile($event)" class="inputfile" id="textFile" />
        <label for="textFile" class="ui large yellow right floated button">
          <i class="ui upload icon"></i>
          Upload text
        </label>
      </div>
    </div>
    <div class="split mid">
      <button class="huge ui teal button" (click)="createBook(book)">Create</button>
      <button class="huge ui teal button" (click)="return()">Cancel</button>
    </div>
  </div>
</div>
```

```
.split {
  margin-block: 20px;
}

.small-split {
  margin-block: 10px;
}

.text-mid {
  font-size: 100px;
```

```
}

.split-right {
    margin-right: 10px;
}

.inputfile {
    width: 0.1px;
    height: 0.1px;
    opacity: 0;
    overflow: hidden;
    position: absolute;
    z-index: -1;
}

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

Б.12 Компонент для редагування книги.

```
@Component({
  selector: 'app-book-edit',
  templateUrl: './book-edit.component.html',
  styleUrls: ['./book-edit.component.scss']
})
export class BookEditComponent implements OnInit {
  @Input() book?: Book;
  imageFile: any;
  textFile: any;

  constructor(private route: ActivatedRoute, private router: Router, private toastr: ToastrService, private
bookService: BookService) { }

  ngOnInit(): void {
    const id = this.route.snapshot.paramMap.get('id');
    if (id) {
      this.bookService.get(id).subscribe(response => (this.book = response));
    }
    window.scrollTo({ top: 0 });
  }

  updateBook(book: Book) {
    const formData: FormData = new FormData();

    formData.append("id", book.id ?? "");
    formData.append("name", book.name ?? "");
    formData.append("description", book.description ?? "");

    if (this.imageFile) {
      formData.append("image", this.imageFile);
    }

    if (this.textFile) {
      formData.append("bookText", this.textFile);
    }

    this.bookService.update(formData).subscribe({
      next: () => {
        this.return();
      },
      error: (error: any) => {
        if (error.error.errors) {
          this.toastr.error(error.error.errors.Name[0]);
        }
        else {
          this.toastr.error(error.error.message);
        }
      },
      complete: () => {
        this.toastr.success("Updating successful.");
      }
    });
  }
```

```
    return() {
      if (this.book?.id) {
        this.router.navigate(['book', this.book.id]);
      } else {
        this.router.navigate(['']);
      }
    }

    selectImageFile(event: any) {
      this.imageFile = event?.target?.files[0];
    }

    selectTextFile(event: any) {
      this.textFile = event?.target?.files[0];
    }
}
```

```
<div class="mid">
  <div *ngIf="book" class="ui form huge" style="width: 50%; ">
    <h2 class="mid">{{ book.name }}</h2>
    <div class="split">
      <div class="small-split">Name:</div>
      <input [(ngModel)]="book.name" required placeholder="Name" />
    </div>
    <div class="split">
      <div class="small-split">Description:</div>
      <textarea rows="2" [(ngModel)]="book.description" placeholder="Description"></textarea>
    </div>
    <div class="ui middle grid container mid">
      <div class="ui fluid segment">
        <input accept=".png, .jpg, .jpe, .bmp" type="file" (change)="selectImageFile($event)" class="inputfile"
          id="imageFile" />
        <label for="imageFile" class="ui large olive right floated button">
          <i class="ui upload icon"></i>
          Upload image
        </label>

        <input accept=".txt" type="file" (change)="selectTextFile($event)" class="inputfile" id="textFile" />
        <label for="textFile" class="ui large yellow right floated button">
          <i class="ui upload icon"></i>
          Upload text
        </label>
      </div>
    </div>
    <div class="split mid">
      <button class="huge ui teal button" (click)="updateBook(book)" *ngIf="book.id">Save</button>
      <button class="huge ui teal button" (click)="return()">Cancel</button>
    </div>
  </div>
</div>
```

```
.split {
  margin-block: 20px;
```

```
}

.small-split {
    margin-block: 10px;
}

.text-mid {
    font-size: 100px;
}

.split-right {
    margin-right: 10px;
}

.inputfile {
    width: 0.1px;
    height: 0.1px;
    opacity: 0;
    overflow: hidden;
    position: absolute;
    z-index: -1;
}

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